1

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

1880 COUNTY ROAD 6, BARRONVILLE, N.Y. 14007 USA
TEL: 612 486 7887 FAX: 612 486 7887 TLE: 612 486 7887

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**A Neural Network Approach
To Detect Traffic Anomalies
in a Communication Network**

by

François Viens, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements of the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
August 6, 1992

© copyright
1992 François Viens

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Votre titre - Votre référence

Votre lieu - Votre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-79821-1

Canada

The undersigned recommend to the Faculty of Graduate Studies and Research

acceptance of the thesis

A NEURAL NETWORK APPROACH TO DETECT TRAFFIC ANOMALIES

IN A COMMUNICATION NETWORK

submitted by François Viens, B. Eng.

in partial fulfilment of the requirements for

the degree of Master of Engineering

Thesis Supervisor

Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University

August 6, 1992

Abstract

LEAPR (Line Element Anomaly Pattern Recognition) is a Knowledge Source (KS) used in a multiparadigm diagnostic system. This KS was trained off-line with traffic data from a simulated Wide Area Network (WAN) under normal and abnormal network operations. Abnormal network operations were characterized by different traffic increases and normal network operations were characterized by no traffic increase. After training, this KS was able to rapidly classify in real time the monitored traffic pattern as either NORMAL or ABNORMAL. After diagnosing an ABNORMAL traffic pattern, the LEAPR KS warns the other KSs used in the multiparadigm diagnostic system of an abnormal traffic increase. An abnormal traffic increase might be caused by a congestion or a hardware fault in the network that causes the traffic to be re-routed to a specific line.

This thesis presents different LEAPR KS prototypes with their respective results. The different prototypes led to the implementation of the LEAPR KS that diagnoses ABNORMAL traffic patterns of any router of the simulated WAN used in the Pegasus research project.

Acknowledgements

I wish to acknowledge the support and direction of my research supervisors, Dr. Nick Dawes and Dr. Bernard Pagurek. Without their support and comments, my work would undoubtedly not be of this quality. I thank Mr. Lyle Zary, Mr. Don Cushman and Mr. Jean-Paul DeCruynaere for their editorial assistance. Thanks also to Department of National Defence who provided me the opportunity to obtain my master degree and to Telecommunications Research Institute of Ontario (TRIO) for funding the project. Finally, I thank my wife, Carole Jodoin, for her continuous help, support and love during the course of my studies.

Contents

Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Acronyms	x
Chapter 1 : Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Overview	3
Chapter 2 : An Overview of Neural Network Technology	4
2.1 Introduction	4
2.2 Why Neural Network Now?	5
2.3 Neural Network History	5
2.4 The Elements of a Nerve Cell	6
2.5 The Elements of a Neural Network	7
2.5.1 Inputs and Outputs	8
2.5.2 Weights	9
2.5.3 Processing Element	9
2.6 Neural Network Layers	11
2.7 Neural Network Operation	12
2.8 Learning Laws	13
2.9 Neural Network Characteristics	15
2.10 Neural Network Limitations and Concerns	16
2.11 Neural Network Paradigms	17
2.12 Summary	18
Chapter 3 : Backpropagation Algorithm	20
3.1 Introduction	20
3.2 Backpropagation Algorithm Theory	21
3.3 Delta Rule	23
3.3.1 Update of the OUTPUT Layer Weights	24
3.3.2 Update of the HIDDEN Layer Weights	25

3.4	Problems Experienced with the Backpropagation Algorithm	27
3.5	Summary	28
Chapter 4 : Recent Related Neural Network Applications		29
4.1	Introduction	29
4.2	Applications in Telecommunications	30
4.3	Applications as a General Diagnostic Tool	30
4.4	Abnormality Detection in a Local Area Network	32
4.5	Neural Network Comparisons	33
Chapter 5 : Problems in Communication Networks		35
5.1	Introduction	35
5.2	The BNR Network Architecture	35
5.3	The BNR Network Overview	36
5.4	Current BNR Diagnostic System	37
5.5	BNR Diagnostic Problem Description	39
Chapter 6 : The Multiple Paradigm Diagnostic System		40
6.1	Introduction	40
6.2	The Pegasus Approach	41
6.2.1	The Blackboard	41
6.2.2	The Simulator	42
6.2.3	The Knowledge Sources	43
6.3	Thesis Justification	44
6.4	LEAPR Implementation Goal	45
Chapter 7 : Development and Results		46
7.1	Introduction	46
7.2	Experiment Setup	46
7.3	The Backpropagation Algorithm	49
7.4	LEAPR Development Sequence	51
7.5	Prototype KS #1	53
7.5.1	Congestion Problem Description	53
7.5.2	Number of Neural Network cases needed	54
7.5.3	Training and Testing for each case	56
7.5.4	Autonomous Diagnostic System.	63
7.5.4.1	Results.	64
7.6	Prototype KS #2	67
7.7	Prototype KS #3	70
7.7.1	Neural Network Learning	73
7.8	Prototype KS #4	75
7.8.1	Training Set	77
7.8.2	The Backpropagation Algorithm Architecture	78
7.8.3	Prototype KS #4 Performance	81

7.9	LEAPR	83
7.9.1	LEAPR TOP TWO and BOTTOM TWO Module	84
7.9.2	LEAPR Performance	84
7.9.3	LEAPR Versatility	85
7.9.4	The LEAPR Repetitiveness Counter (RC)	88
Chapter 8 :	Conclusions	93
8.1	Research Summary	93
8.2	Summary of Thesis Contributions	94
8.3	Future Research	94
References		96
APPENDIX 1:	Codes for the Backpropagation Algorithm	100
APPENDIX 2:	Codes for the LEAPR KS	116

List of Figures

2.1 :	Principle parts of the nervous system.	7
2.2 :	The elements of a neural network.	8
2.3 :	Transfer functions.	10
2.4 :	Principal layers of a neural network.	11
2.5 :	Curve of the gradient descent.	14
3.1 :	Topology of the backpropagation algorithm.	21
5.1 :	BNR network overview.	36
5.2 :	Router to router connections.	37
5.3 :	Overview of the current BNR diagnostic system.	38
6.1 :	Pegasus's blackboard architecture.	42
7.1 :	One day of normal simulated network operations.	48
7.2 :	bCARc2's global traffic pattern vs number of workstations.	55
7.3 :	A window of number of workstations.	56
7.4 :	Training set's histogram of case #1.	59
7.5 :	Training set's histogram of case #2.	59
7.6 :	Training set's histogram of case #3.	60
7.7 :	Training set's histogram of case #4.	60
7.8 :	Training set's histogram of case #5.	61
7.9 :	Daily traffic variation of 3 lines from bCARc2.	66
7.10:	Prototype KS #2 components.	67
7.11:	Training set's histogram of prototype KS #2.	70
7.12:	Training set's histogram of prototype KS #3.	72
7.13:	Training set for the backpropagation algorithm of prototype KS #3.	74
7.14:	The decision boundaries learned by the backpropagation algorithm.	74
7.15:	Prototype KS #4 components.	75
7.16:	Training set's histogram of the backpropagation algorithm with 9 PEs in the HIDDEN layer.	79
7.17:	Training set's histogram of the backpropagation algorithm with 12 PEs in the HIDDEN layer.	80
7.18:	Training set's histogram of the backpropagation algorithm with 16 PEs in the HIDDEN layer.	80
7.19:	LEAPR knowledge source components.	83
7.20:	Number of false alarms per day based on a given RC threshold value.	91
7.21:	Cumulative percentage of ABNORMAL traffic patterns detected given a RC threshold value.	91

List of Tables

7.1 : MOMENTUM and LCOEF parameters.	49
7.2 : Training period for each neural network case.	61
7.3 : Number of standard deviations above the traffic mean where abnormalities were diagnosed.	62
7.4 : Prototype KS #1's performance in real time (test #1).	65
7.5 : Prototype KS #1's performance fifteen minutes behind real time (test #2).	65
7.6 : Type of ABNORMAL traffic pattern cases in the training set of prototype KS #2.	68
7.7 : Number of consecutive times ABNORMAL cases were detected in different simulated networks.	72
7.8 : RC values based on the duration of the abnormality.	77
7.9 : Type of ABNORMAL traffic patterns in the training set of prototype #4.	78
7.10 : Number of ABNORMAL cases detected.	82
7.11 : LEAPR performance given the RC threshold value was set to 1.91.	86
7.12 : LEAPR performance on detecting abnormal pre-selected traffic increases.	87
7.13 : Number of detected false alarms per day based on a given RC threshold value.	89
7.14 : Number of ABNORMAL traffic patterns detected with their respective maximum RC value encountered.	90
7.15 : LEAPR KS average efficiency.	92

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
ATM	Asynchronous Transfer Mode
BANES	Basic Analysis in Nested Evidence Space
BNR	Bell Northern Research
DCD	Data Collector Device
END	Expert Network Diagnostician
INSIDE	Inertial Navigation System Interactive Diagnostic Expert
KS	Knowledge Source
LAN	Local Area Network
LCOEF	Learning Coefficient
LEAPR	Line Element Anomaly Pattern Recognition
MAN	Metropolitan Area Network
NNet	Neural Network
NOC	Network Operating Centre
PC	Personal Computer
PE	Processing Element
RC	Repetitiveness Counter
RDA	Remote Diagnostic Access
SNMP	Simple Network Management Protocol
TRIO	Telecommunications Research Institute of Ontario
WAN	Wide Area Network

Chapter 1

Introduction

1.1 Motivation

As we move into the 90s, communication network management is becoming increasingly complex due to the introduction of new technologies and new services. Some problems generated by this network complexity have been impractical to solve using conventional software techniques. However, Artificial Intelligence (AI) techniques have been proven to be able to solve some of these network management problems [CEBULKA89]. One such AI technique is an Artificial Neural Network (ANN).

Artificial neural networks have been proposed as solutions for a wide variety of tasks. Among the most promising is that of pattern classification. Fault detection and diagnosis can be directly viewed as a pattern classification activity, and thus the potential exists for fault detection via ANNs. Trained on shown examples, a neural network will be able to classify new examples. This technique has shown great promise in areas where the domain is poorly understood [CEBULKA89], such as the analysis of traffic patterns in a communication network.

1.2 Objectives

In order to diagnose communication network problems, a project entitled Pegasus was initiated to design and test a diagnostic system using many different AI techniques, including the use of neural networks.

The main purpose of this thesis is to apply neural network technology as a proactive Knowledge Source (KS) to classify network traffic patterns as either NORMAL or ABNORMAL. The objectives of this thesis are threefold:

1. to investigate the idea of using a neural network as a proactive diagnostician agent in order to detect ABNORMAL traffic patterns in a communication network;
2. to evaluate the performance of the neural network agent in the detection of traffic normalities and abnormalities; and
3. to build an autonomous neural network that will adapt to different traffic patterns and be able to diagnose any router of a Wide Area Network (WAN).

In order to pursue these objectives, a backpropagation algorithm and a knowledge source called Line Element Anomaly Pattern Recognition (LEAPR) were implemented in C on a SPARCstation 2. The backpropagation algorithm was implemented to learn to classify the traffic patterns off-line, and based on what the backpropagation algorithm learned, the LEAPR KS was implemented to classify the traffic patterns on-line.

1.3 Overview

This thesis is divided into eight chapters. Chapter 2 presents an overview of the neural network technology. Chapter 3 describes the neural network paradigm, the backpropagation algorithm, used in this thesis. Chapter 4 presents the different application areas of ANN, particularly in the telecommunication field, and as a general diagnostic tool.

Chapter 5 describes diagnostic problems in a Wide Area Network (WAN) using the Bell Northern Research (BNR) communication network as an example. To solve the problems described in Chapter 5, a Multiple Paradigm Diagnostic System for Wide Area Network was initiated under the Pegasus project. Chapter 6 describes some of the Pegasus project's co-operating knowledge sources and identifies the LEAPR KS.

Chapter 7 describes the different stages of this research in order to meet the three objectives previously described in section 1.2.

Finally, Chapter 8 provides a summary of this thesis, assesses LEAPR's contribution to the Pegasus project and suggests future research.

Chapter 2

An Overview of Neural Network Technology

2.1 Introduction

The human brain is the most complex computing device known to man. The brain's powerful thinking, remembering, and problem-solving capabilities have inspired many scientists to try to create a computer model that matches the functionality of the brain in a very highly simplified manner [KLIMASAUSKAS89]. The result is Neural Network (NNet) technology. NNet technology is not meant to duplicate the operation of the human brain, but it rather receives inspiration from known facts about how the brain works.

As outlined in [NELSON90], many terms such as parallel distributed processing models, connectivist/connectionism models, adaptive systems, self-organizing systems, neurocomputing, and neuromorphic systems, all refer to the same form of information processing as the neural network. Since the term "neural network" seems to be the one most commonly used, I will continue to use the words "neural network" to mean the

broad class of artificial neural systems.

2.2 Why Neural Network Now?

Recently, there has been a resurgence of interest in neural networks. The reasons for this resurgence are the appearance of faster digital computers on which to simulate larger networks, interest in building massively parallel computers, and the discovery of powerful network learning algorithms.

Artificial Intelligence (AI), vaunted by all sorts of high expectations, has not been able to deliver on many of its anticipated promises [NELSON90]. Logic systems have turned out to be poor at dealing with images and making analogies. Rule-based systems are limited in their ability to accommodate inaccuracies or fuzzy information.

A number of today's technological problems are in areas where neural network technology has demonstrated some potential. Some of these problems are pattern recognition, diagnostic, speech and image understanding, sensor processing, robotic control and optimization.

2.3 Neural Network History

Documented attempts to understand the human brain go back to 1890, where a text of William James contained insights into brain activity and indicated current theories [NELSON90].

In 1957, Frank Rosenblatt [KLIMASAUSKAS89] began work on a neural network model called "perceptron". Built in hardware, it is the oldest neural network and its

concepts are still used today in various forms for applications such as character recognition.

In 1959, Bernard Widrow and Marcian Hoff (Stanford) [NELSON90] developed a neural network model called ADALINE (ADaptive LINEar Elements). A multi-layer version of the ADALINE paradigm was called MADALINE (Multiple ADALINEs). One of the most famous applications of this neural network model was in the telecommunications industry. This model was used, and is still used today, to eliminate echoes on phone lines.

The introduction of the perceptron model by Rosenblatt in the late 1950s created a great deal of excitement in the research community. However in 1969, Minsky demonstrated that the perceptron could not solve non-linear separable problems. Due to that serious learning problem, actual research in perceptron learning came to a halt in the 1970s. The field saw little interest until the 1980s, when the backpropagation learning algorithm was proposed. Backpropagation was not discovered until recently, when digital computers became fast enough to support large-scale simulations of neural networks.

2.4 The Elements of a Nerve Cell

The neuron, in the human brain, is the fundamental cellular unit of the nervous system. Each neuron (Figure 2.1) is a simple microprocessing unit which receives signals coming from many other neurons through input processes called dendrites. Signals are communicated to the neuron body through synapses. If the combined signal is strong enough, the neuron activates its firing mechanism which produces an output signal

through the axon [KLIMASAUSKAS89].

Figure 2.1: Principle parts of the nervous system.

The human brain consists of tens of billions of neurons densely inter-connected. In the neuron inter-connection architecture, the axon of a neuron is split up and connected to dendrites of other neurons through a junction known as a synapse. This synaptic efficiency is what is modified when the brain learns. The synapse combined with the processing of information in the neuron probably forms the basic memory mechanism of the brain.

2.5 The Elements of a Neural Network

Based on the analogy between the brain and the neural network's technology, the two primary elements which make up an artificial neural network are the Processing Elements (PEs) and the interconnections or weights (Figure 2.2). The processing

elements, which are the ANN's equivalent of neurons, are generally simple devices that receive a number of input signals, calculate the sum of all the inputs, and finally with the transfer function, determine the output. The output signal of an individual processing element is sent to many other processing elements as input signals via the weights between processing elements.

Figure 2.2 : The elements of a neural network.

2.5.1 Inputs and Outputs

Just as there are many inputs to a neuron, there should be many input signals coming into our processing elements simultaneously. In response to the inputs, based on the transfer function, the processing element outputs a scalar value. Each processing element has many inputs but only has a single output signal.

In addition, just as real neurons are affected by things other than inputs, some networks provide a mechanism for other influences. Generally this extra input is either called the "bias term" or the "threshold value".

2.5.2 Weights

Each input is given a relative weight which affects the impact of that input to the processing element. Weights are adaptive coefficients, which throughout the training phase are modified in order to learn correctly the relations between the input signals and the output signals.

Mathematically, the inputs and the weights are vectors such as $(I_0, I_1, \dots, I_n)$ and $(W_0, W_1, \dots, W_n)$. The total input signal is the dot or inner product of the two vectors.

2.5.3 Processing Element

Two important activities take place within the design of the processing element. These activities are the summation function and the transfer function. The summation function sums the inner product of the inputs vector and the weights vector and then forwards the scalar value to the transfer function in order to determine the output value. The transfer function is generally nonlinear and is part of what determines the dynamic state of a neural network system. There are different types of transfer functions. There is the "hard limiter" or "step function" (Figure 2.3.a), which depending if the result of the summation is positive or negative, the processing element outputs 1 and -1 respectively.

Figure 2.3 : Transfer functions.

Another type of transfer function is the "ramping function" (Figure 2.3.b) which mirrors the input within a given range (from 0 to 1), and acts as a hard limiter function outside the given range. It is a linear function which has been clipped to minimum and maximum values, which then makes it nonlinear.

Two other options are the standard "sigmoid function" (Figure 2.3.c) and the "bipolar sigmoid function" (Figure 2.3.d). Both of these curves approach a minimum and maximum value at the asymptotes. Mathematically, the exciting feature of these type of curves is that both the function and its derivative are continuous, which is necessary for some neural network paradigms. These functions work fairly well and are often the transfer function of choice.

2.6 Neural Network Layers

A neural network is composed of many processing elements (PEs) which are grouped into layers. Each layer has a specific role to play in the neural network (Figure 2.4). The layer that receives the input is called the INPUT layer. This layer performs no function other than the buffering of the input signal. The neural network's outputs are generated from the OUTPUT layer. Any other layers are called HIDDEN layers because they are internal to the network and have no direct contact with the external environment.

Figure 2.4 : Principal layers of a neural network.

The number of processing elements in the INPUT and OUTPUT layer is determined by the number of inputs and outputs required for the system. The number of

HIDDEN layers and the number of processing elements to use in these layers is not obvious and is currently an area of experimentation [NELSON90.a]. In a number of cases [NELSON90.a], a four to one ratio of HIDDEN layer's PEs to INPUT layer's PE worked well.

2.7 Neural Network Operation

There are two main phases in the operation of a neural network: the learning phase (or training phase) and the recall phase. In most networks these are distinct.

In the learning phase, the interconnections (weights) between processing elements are modified in response to an input vector and optionally, a desired output vector. If there is a desired output vector, it can be provided by a knowledgeable person since the neural network will learn the relations between the input and desired output vectors. In this case, the learning is referred to as "supervised learning".

If the desired output vector is different from the input vector, the trained network is referred to as "hetero-associative" network. If, for all training examples, the desired output vector is identical to the input vector, the network is referred to as "auto-associative". If there is no specified desired output, the learning is called "unsupervised learning".

Whatever kind of learning is used, the main characteristic of any neural network implementation is its learning rule which determines how the weights will be modified in order to learn the training examples.

In the recall phase, the input vector is presented to the input layer of the neural

network, which upon reception, processes it and outputs a response to its output layer.

2.8 Learning Laws

Many learning laws are used by the neural network community. However, most of the commonly used ones are variations of the best known learning law called "Hebb's Rule". Researchers are conducting different experiments in order to model how the human brain handles learning. Unfortunately, since the brain is very complex, there are a lot of things we do not know yet about how the brain functions in learning. However, generally the training laws are some variation of the following three types: the Hebbian learning, the Delta rule learning and the competitive learning.

1. The Hebbian learning rule was introduced by Donald Hebb. Hebb's basic rule states that the connection weight between two processing elements is incremented according to the correlation of the values of the two processing elements it connects. This rule is used in NNet paradigms that use supervised learning.

2. The Delta rule learning is a mathematical approach to minimize the error between the desired and the actual NNet output value. Since this rule needs a desired output value, this learning rule can only be used in NNet paradigms that use supervised learning. For this rule, the weights are modified by an amount proportional to the first derivative of the error with respect to the weight. The delta rule belongs to a class of gradient-descent heuristics [JONES87].

As a gradient-descent search, the delta rule can be illustrated with the aid of a basin (Figure 2.5). From the top of the basin, the method is descending along a curve to

reach the bottom of the basin, where the error is at its minimum [Knight90].

Figure 2.5: Curve of the gradient descent.

3. The competitive learning rule can only be used in a NNet paradigm that uses unsupervised learning. In this procedure, the processing elements compete among themselves in order to learn. The processing element with the strongest output is declared the winner and has the capability of inhibiting its competitors and exciting its neighbours. Only the winner is permitted to output, and only the winner and its neighbours are permitted to adjust the weights. This mutual inhibition can be referred to as the WINNER-TAKE-ALL behaviour.

2.9 Neural Network Characteristics

Neural networks have the following characteristics:

- **Learning by examples.** Unlike traditional expert systems where knowledge is made explicit in the form of rules, neural networks generate their own rules by learning from examples. Learning is achieved through a learning rule which adapts or modifies the connection weights of the network in response to the example inputs and the desired outputs of those inputs.
- **Distributed Associative Memory.** One the most important characteristics of neural networks is the way they store information. Neural computing memory is distributed. The weights between the processing elements are the memory units of a neural network. The values of the weights represent the current state of knowledge of the network.
- **Fault Tolerance.** Whereas traditional computing systems are rendered useless by even a small amount of damage to memory, neural networks are fault tolerant. In most neural networks, if some processing elements are destroyed, disabled, or their connections slightly modified, then the behaviour of the network as a whole is only slightly degraded. If more processing elements are destroyed, the behaviour of the network is degraded slightly further and the performance will suffer, but the system does not come to an abrupt halt. Neural computing systems are fault tolerant because information is not contained in one location, but it is distributed throughout the system. This characteristic of fault tolerance makes neural computing systems extremely well-suited for applications where failure of the system means disaster.

- **Pattern Recognition.** Pattern recognition tasks require the ability to match large amounts of input information simultaneously and then generate categorical or generalized output. They also provide a reasonable response to noisy or incomplete inputs. Neural computing systems possess the ability to learn and build unique structures specific to a particular problem; this ability makes neural networks very useful in pattern recognition.
- **Automatic learning.** In expert systems, knowledge acquisition is a difficult problem and a time consuming task for the knowledge engineer. One attractive feature of the neural network technology is that it lends itself very well to automatic learning. The knowledge is automatically encoded in the weights on the connections between each processing elements.
- **Generalization.** In an expert system, rules and algorithms are written for a particular domain, without reference to any particular examples; it is anticipated that the rules are general enough to handle whatever input cases the expert system receives. In the case of a neural network, using the knowledge obtained during its training phase, it is hoped that the neural network will generalize in order to handle new input cases.

2.10 Neural Network Limitations and Concerns

The NNet characteristics make neural network technology very promising; however, there are some limitations and concerns applied to its use.

One of the most important limitations is the learning. Learning, which is based on what the network is taught through its training data, can be tedious. Unfortunately, there are no standards for either training algorithms or training data sets. The training

data set determines the neural network's success and is the most critical part of the network.

A problem that might occur with the neural network's technology is the "Memorization problem". The memorization problem occurs when the network is overtrained; rather than adapting its weights in order to generalize for different input vectors, the network memorizes all input vectors in its training data set.

Another issue that expert systems can deal with is the justification facility. The neural network cannot justify its answer as in the expert system; the user has to trust the neural network.

Other limitation issues include that neural networks are inadequate in providing precise answers and neural network design is more of an art than a science [NELSON90.a and MOONEY89]. It is more of an art because there are no rules or guidelines to design and build a neural network; people have to try different transfer functions and different network architectures in order to come up with the best solution to their problem.

2.11 Neural Network Paradigms

In the field of Neural Networks, there are several network paradigms which represent different perspectives. Some of the most popular paradigms are the Hopfield network, the Perceptron and the backpropagation algorithm.

■ In a Hopfield network [KLIMASAUŠKAS89, NELSON90 and KNIGHT90], units (processing elements) are always in one of the two states: active or inactive. Units are connected to each other by weighted symmetric connections. A positive connection

indicates that the two units tend to activate each other. A negative connection allows an active unit to deactivate a neighbouring unit.

■ **The Perceptron** [KLIMASAUSKAS89,NELSON90 and KNIGHT90] was one of the earliest neural network models. A Perceptron models a neuron by taking a weighted sum of its inputs and sending an output 1 if the sum is greater than some adjustable threshold value. Otherwise, it sends a 0.

The perceptron consists of weights, a summation processor, and an adjustable threshold processor. Learning is a process of modifying the values of the weights and the threshold.

■ **A backpropagation algorithm** [JONES87,KLIMASAUSKAS89 and NELSON90] is a multi-layer network. This network has an INPUT layer, an OUTPUT layer, and at least one HIDDEN layer. During learning, the inputs are propagated in the forward direction to the OUTPUT layer. At the OUTPUT layer, the error, which may be basically described by the difference between the desired and the actual output, is propagated back to the network and used to update the weights between processing elements. The NNet is trained until the error is either sufficiently low or zero.

2.12 Summary

The neural network is composed of a processing element and weights. Depending on the learning rule and the chosen network paradigm, the weights are modified accordingly.

Many advantages, such as learning by example, distributed associative memory,

fault tolerance, automatic learning and generalization, make neural network technology very attractive and promising. To take advantage of these facets, many NNet applications have been developed in order to resolve problems which current AI technologies could not solve.

Chapter 3

Backpropagation Algorithm

3.1 Introduction

The backpropagation algorithm was developed in the 1980s to solve the Perceptron's problem found by Minski and Papert [KNIGHT90]. This NNet paradigm has been found to be useful in addressing problems requiring recognition of complex patterns. It is designed for a multi-layered network operating in a feedforward manner and using the supervised learning mode.

Since the backpropagation algorithm is one of the most commonly used NNet techniques, and because its learning algorithm could be applied to a communication network traffic monitoring problem, the backpropagation algorithm was chosen as the neural network paradigm of choice for this thesis. This chapter describes the fundamental theory of the backpropagation algorithm and presents some of its problems.

3.2 Backpropagation Algorithm Theory

The Backpropagation algorithm learns a predefined set of input-output example pairs (training set) by using a forward pass followed by a backward pass [SIMPSON89].

In the forward pass, the input pattern is presented to the INPUT layer and is propagated to each following layer until it reaches the OUTPUT layer (Figure 3.1).

Figure 3.1 : Topology of the backpropagation algorithm.

In reference to Figure 3.1, the input vector is applied to the INPUT layer. The INPUT layer Processing Elements (PEs) distribute their values to the HIDDEN layer PEs. The summation of all the inputs to the i^{th} HIDDEN layer PE is

$$b_i = \sum_{h=1}^n (A_h \times v_{hi}) + O_i \quad (3.1)$$

where A is the input vector, v_{hi} is the weight on the connection from the h^{th} INPUT layer PE, and O_i is the i^{th} HIDDEN layer PE threshold value discussed at Section 2.5.1. With the transfer function " $f(\)$ " after the summation, the output of the i^{th} HIDDEN layer PE is

$$B_i = f(b_i) \quad (3.2)$$

Once at the HIDDEN layer, the HIDDEN layer PEs distribute their values B_i to the OUTPUT layer PEs. The summation of all the inputs to the j^{th} OUTPUT layer PE is

$$c_j = \sum_{i=1}^p (B_i \times w_{ij}) + T_j \quad (3.3)$$

where w_{ij} is the weight on the connection from the i^{th} HIDDEN layer PE, and T_j is the j^{th} OUTPUT layer PE threshold value. With the transfer function " $f(\)$ " after the summation, the output of the j^{th} OUTPUT layer PE is

$$C_j = f(c_j) \quad (3.4)$$

The transfer function used in this backpropagation algorithm is the standard sigmoid function mentioned in Section 2.5.3. The equation of the standard sigmoid function "f(x) is

$$f(x) = (1 + e^{-x})^{-1} \quad (3.5)$$

Once the OUTPUT layer is reached, the actual NNet output¹ is compared to the desired output, and an error signal is computed for each OUTPUT layer PE. Once the error signal is computed, the backward pass starts. The error signal is back-propagated through the network from the OUTPUT layer to the INPUT layer. Each weight between the OUTPUT and HIDDEN layer and between the HIDDEN and INPUT layer receives a portion of the total error that represents its contribution of the actual output. Based on the error signal received, the weights are modified through the delta rule until the network converges to a state where the difference between the desired output and the actual NNet output is either sufficiently low or zero.

3.3 Delta Rule

The learning law used in the backpropagation algorithm is the delta rule. The error minimized by this rule is the sum of the squares of the errors for all output nodes, and is represented in equation 3.6. OUT_j represents the desired output and C_j represents the actual NNet output.

1: The actual NNet output is the output processed with the current weights based on the input vector.

$$E_p = 0.5 \sum_{j=1}^q (\text{OUT}_j - C_j)^2 \quad (3.6)$$

** The factor of 0.5 in equation 3.6 is there for convenience in calculating the derivatives later.

The next two sections deal with how the weights between the OUTPUT and HIDDEN layers and the weights between the HIDDEN and INPUT layer are modified and updated in order to minimize the error between the desired output (OUT_j) and the actual NNet output (C_j) [FREEMAN91].

3.3.1 Update of the OUTPUT Layer Weights

To determine the direction in which to change the weights between the OUTPUT and HIDDEN layer, the negative of the gradient of E_p with respect to the weight w_{ij} , Δw_{ij} , is calculated as follows:

$$-\frac{\partial E_p}{\partial w_{ij}} = (\text{OUT}_j - C_j) \frac{\partial C_j}{\partial w_{ij}} \quad (3.7)$$

In reference to equation 3.4, by the chain rule, equation 3.7 can be transformed to

$$-\frac{\partial E_p}{\partial w_{ij}} = (\text{OUT}_j - C_j) \frac{\partial f(c_j)}{\partial c_j} \frac{\partial c_j}{\partial w_{ij}} \quad (3.8)$$

The derivative of c_j with respect to the weights w_{ij} is

$$\frac{\partial c_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{i=1}^P (B_i \times w_{ij}) + T_j = B_i \quad (3.9)$$

With the sigmoid function (equation 3.5) as the transfer function, the differentiation of the transfer function with respect to c_j can be expressed by the transfer function itself as follows:

$$\frac{\partial f(c_j)}{\partial c_j} = f(c_j) (1 - f(c_j)) = C_j (1 - C_j) \quad (3.10)$$

By substituting equation 3.9 and 3.10 into 3.8, equation 3.8 can be expressed as:

$$\Delta w_{ij} = \text{LCOEF} (OUT_j - C_j) C_j (1 - C_j) B_i \quad (3.11)$$

where LCOEF is the learning coefficient parameter added to the formal equation 3.7 in order to control the learning rate. This parameter is set by the NNet user.

3.3.2 Update of the HIDDEN Layer Weights

As done previously for the OUTPUT layer, in order to determine the direction in which to change the weights between processing elements in the HIDDEN layer and the INPUT layer, the negative of the gradient of E_p with respect to the weight v_{hi} , Δv_{hi} , is calculated as follows:

$$- \frac{\partial E_p}{\partial v_{hi}} = \sum_{j=1}^q (OUT_j - C_j) \frac{\partial C_j}{\partial v_{hi}} \quad (3.12)$$

Since C_j is not a function of v_{hi} , intermediate derivations have to take place as follows:

$$- \frac{\partial E_p}{\partial v_{hi}} = \sum_{j=1}^q (OUT_j - C_j) \frac{\partial f(c_j)}{\partial c_j} \frac{\partial c_j}{\partial B_i} \frac{\partial B_i}{\partial b_i} \frac{\partial b_i}{\partial v_{hi}} \quad (3.13)$$

In reference to the equations 3.1 to 3.3, the results of these derivations are as follows:

$$\frac{\partial c_j}{\partial B_i} = w_{ij} \quad (3.14)$$

$$\frac{\partial B_i}{\partial b_i} = f(b_i) (1 - f(b_i)) = B_i (1 - B_i) \quad (3.15)$$

$$\frac{\partial b_i}{\partial v_{hi}} = A_h \quad (3.16)$$

With the results of these derivations, equation 3.13 can be expressed as follows:

$$- \frac{\partial E_p}{\partial v_{hi}} = B_i (1 - B_i) A_h \sum_{j=1}^q (OUT_j - C_j) C_j (1 - C_j) w_{ij} \quad (3.17)$$

So the result of equation 3.12 is

$$\Delta v_{hi}(t) = \text{LCOEF} \cdot B_i (1 - B_i) A_h \sum_{j=1}^q (\text{OUT}_j - C_j) C_j (1 - C_j) w_{hj} \quad (3.18)$$

As mentioned for equation 3.11, LCOEF is the Learning Coefficient parameter which is set by the NNet user.

3.4 Problems Experienced with the Backpropagation Algorithm

The two most serious problems with the backpropagation algorithm are the speed of learning and the possibility of converging to a local minima.

The training cycle of the backpropagation network is generally very slow. Depending on the training data set and on the network architecture, the learning phase can take many iterations. In order to overcome the speed of the learning problem, many variations of the backpropagation algorithm are presented in [KLIMASAUSKAS89]. One of the most widely used variations of the backpropagation algorithm is the one incorporating the MOMENTUM parameter. In this algorithm, the weights are modified by two weight adjustments (Equation 3.19): the first adjustment is the current weight update and the second adjustment is a portion, determined by the MOMENTUM coefficient, of the previous weight update. This algorithm was the algorithm of choice for this thesis.

$$\Delta \text{weight}(t) = \Delta \text{weight}(t) + \text{MOMENTUM} \times \Delta \text{weight}(t - 1) \quad (3.19)$$

Another issue related to the speed of learning is that the backpropagation algorithm has to be trained off-line. The network has to be trained before it can be used in real-time.

The second most important problem of the backpropagation algorithm is the possibility of converging to a local minimum in the weight space. If a local minimum is reached, the error at the outputs may still be unacceptably high, which means that the network may have not learned all the training sets correctly. Fortunately, in practice, this problem does not seem to cause much difficulty [FREEMAN91]. However, if a network stops learning before reaching an acceptable solution, the problem can be fixed by either modifying the network's architecture or modifying the initial weights.

3.5 Summary

In the neural network community, the backpropagation algorithm is one of the most commonly used. Based on the error which is the difference between the actual NNet output and its desired output, the backpropagation algorithm learns to classify sets of input data. This supervised learning NNet paradigm was considered for use in this thesis.

The two issues related to the learning problem were overcome in this thesis by respectively training the NNet off-line and by using the backpropagation algorithm with the MOMENTUM coefficient. Concerning the possibility of converging to a local minima, since the NNet was trained off-line, if the NNet's solution was not acceptable the NNet was retrained with either a different NNet's architecture or different initial weights.

Chapter 4

Recent Related Neural Network Applications

4.1 Introduction

Neural network technology has been applied to many areas [SIMPSON89] such as system diagnostics, handwriting recognition, financial and economic modelling, industrial inspection, and target classification.

The aim of this chapter is to present different applications where the neural network technology has been used. This chapter presents NNet applications in the telecommunications field and NNet applications as general diagnostic tools. This chapter also presents a particular paper [MAXION90] describing a system that detects abnormalities in a Local Area Network (LAN). Finally, this chapter compares the neural network approach with an expert system and some statistical methods.

4.2 Applications in Telecommunications

Neural network applications in the telecommunications field have been mostly applied as a control or a management tool [KHASNABISH91, CHEN91 and ANSARI91]. A satellite communication network was monitored by a neural network in order to have high network efficiency [ANSARI91]. An adaptive control scheme based on a neural network (backpropagation algorithm) was used to solve a general quality-of-service control problem in broadband Asynchronous Transfer Mode (ATM) networks [CHEN91]. A neural network has been proposed for fast and intelligent processing of transit packets and assigning network access to the local packets in large supra-high-speed packet switched telecommunication network [KHASNABISH91].

Another interesting neural network application was the use of a neural network as a basic ATM communication network admission controller [HIRAMATSU90]. Based on the traffic and on the service qualities, the neural network decides either to accept or to reject call set-up requests. The interesting feature of this application was that at every control cycle time, based on the history of observed cases, the neural network retrained itself in order to adapt to the new traffic characteristics.

4.3 Applications as a General Diagnostic Tool

Neural networks can be used as diagnostic experts in a variety of applications. Maureen Caudill [CAUDILL90] presented different neural network applications as general diagnosticians. One of these applications was the use of a neural network in medical diagnostics. A system, designed by researchers in Milan, Italy, was developed to aid

physicians in diagnosing epilepsy. Using a backpropagation network, the researchers began with a very large binary input vector consisting of 814 yes-no elements taken from the answers to a categorization questionnaire. After training their backpropagation algorithm, they removed all PEs with weights less than 0.5. Removing these PEs reduced the size of the input vector from 814 to 724 PEs; the 90 questions corresponding to the removed PEs provided no significant information about the classification of the cases. They then retrained the pruned network and discovered they had a better classification record than the initial one on the test-data set. The significance of these results was that the neural network diagnostic system was not only a very good classifier of epileptic illnesses, but also identified portions of the questionnaire that were probably unimportant for classifying the illnesses. Another application presented by Maureen Caudill was an on-line diagnostic network. The system called DASA/LARS was built by Fred Casselman of GTE in association with Major Jody D. Jonghe Acres of the Defense Communication Agency. The problem DASA/LARS addressed was the fault diagnosis of satellite-communication broadcasts. Their system was composed of nine backpropagation networks that took their inputs from two key sources: a spectrum analyzer and a large database. The most important lesson learned from this system was the notion of modularizing difficult tasks. Instead of trying to make one neural network to learn the entire problem, they broke the complex problem into smaller pieces so that each network had a reasonable task to perform.

Another neural network application in the medical field was reported in [MCALLEMI91] diagnosing bladder cancer. Different images of cells from a urine

sample were fed to a multi-layered backpropagation algorithm which classified them into either Well or Not-Well categories. The network had an input layer with five PEs, two hidden layers with ten PEs, and an output layer with one PE; a value at the output PE greater than 0.5 signified a Well cell, while a value less than 0.5 signified a Not-Well cell. The overall accuracy rate of this system was 96.5%, with an error rate of 3.5%.

An Inertial Navigation System Interactive Diagnostic Expert (INSIDE) was designed based on a neural network model to troubleshoot an avionic line replaceable unit [TAN90]. In this application, the knowledge was acquired by learning from past cases recorded in the workshop log book. The interesting feature of this system was that the connectionist system was complemented by a flowchart module that captured the knowledge of troubleshooting flowcharts. During operation, if the connectionist module fails to derive the solution, the user will be directed to the flowchart module for guidance. After the case is solved, it can be captured as a new example to be acquired by the connectionist module. The learning process of this system highly resembles the way an expert acquires knowledge through experience.

4.4 Abnormality Detection in a Local Area Network

The paper written by R.A. Maxion [MAXION90] describes an Ethernet anomaly detector based on Local Area Network (LAN) traffic data as the only input. Maxion developed a system that acquires its own internal model of the normal conditions in its environment, and as a consequence, it is able to detect unusual events against a background of normalcy.

The data used in this study was the LAN traffic data. Based on the traffic data, a tolerance envelope was established to model normal behaviour. The tolerance envelope was formed using traffic statistics during a period of two weeks. Different models were set for week days and weekend days. Once the tolerance envelope was formed, the actual traffic was mapped to the model, and if the traffic data was outside the tolerance envelope, the traffic data was diagnosed as abnormal.

Maxion's diagnostic system operated autonomously in real time, 24 hours a day. It detected 100% of all human-detected anomalies, based on a study of trained technical personnel attempting to detect network abnormalities. The system experienced an average of 12.7 anomalies per day during October 1989, 4.7 of which were alarmed. Most alarms detected were very large file transfers or backup operations. Information from these alarms was being used to help schedule backup operations in order to utilize the network most efficiently.

4.5 Neural Network Comparisons

Neural networks have been compared to other AI techniques and to statistical methods. In [COLEMAN91], a neural network have been compared to an expert system and to a popular statistical technique, called two-pass logistic regression. Coleman stated that on a specific non-linear problem domain, the neural network has provided a much better solution than either the statistical technique or the expert system. The overall improvement of the neural network over the statistical technique and the expert system for non-linear problems has been typically between 8 to 15% [COLEMAN91].

In [FISHER89 and MOONEY89], the backpropagation algorithm was compared to the ID3 symbolic learning algorithm. In general, the two comparisons stated that ID3 ran significantly faster than the backpropagation algorithm during learning and during classification of novel examples. However, the probability of correctly classifying new examples was about the same for the two algorithms. The two comparisons also stated that the backpropagation algorithm handles both noise and missing features more reliably than ID3.

Chapter 5

Problems in Communication Networks

5.1 Introduction

The aim of this chapter is to present communication network problems that are very difficult to diagnose from the point of view of a network operator. In order to present these problems, the Bell Northern Research (BNR) network is used as a test bed for this thesis.

5.2 The BNR Network Architecture

The BNR network is composed of Local Area Networks (LANs) interconnected into a Metropolitan Area Network (MAN), and further interconnected into a Wide Area Network (WAN) to other BNR locations around the world [BNR90]. Actually, the BNR network spans four continents: North America, Europe, Australasia and Asia.

5.3 The BNR Network Overview

Each LAN is composed of workstations, primary concentrators, secondary concentrators, bridges, routers, and IBM mainframes (Figure 5.1).

Figure 5.1: BNR network overview.

Figure 5.2 illustrates a portion of the BNR network where only routers are shown. Routers connect different ethernet networks together through other routers. Each router has multiple potential paths to send packets from one location to another. Each router is typically connected to two ethernet networks and three serial lines to other routers. The serial lines range from 56 kbits/sec to 1.544 Mbits/sec [DAWES91].

Figure 5.2 : Router to router connections.

5.4 Current BNR Diagnostic System

BNR has a central monitoring location called the Network Operating Centre (NOC) that monitors and manages the entire network. The NOC has a support system application called XNETMON which has a topological representation of the network components, and which receives information from remote Data Collector Devices (DCDs) (Figure 5.3). The XNETMON application displays coloured icons. The icon's shape represents the network component's type, and its colour defines its status.

Through distributed polling and using the Simple Network Management Protocol (SNMP), the DCDs monitor bridges, routers, fastpaths and servers. Every router, bridge

and server is respectively polled every minute, three minutes and five minutes. The DCDs, which are UNIX workstations, are typically connected to about five Ethernets. All the data information detected by the DCDs are reported back to the XNETMON system for monitoring purposes.

In addition to the XNETMON display, BNR has a Remote Diagnostic Access (RDA) tool which is used as a LAN traffic analyzer. This tool, which is connected to every secondary concentrator for control purposes, consists of a matrix switch and a PC (Figure 5.3). From the PC located at the NOC, the network operator can diagnose problems all over the network through modems.

Figure 5.3: Overview of the current BNR diagnostic system.

5.5 BNR Diagnostic Problem Description

With over 3000 workstations, 300 bridges and 120 routers, the BNR diagnostic problem [PAGUREK91] can be divided into the three following major problems :

1. Because of the redundancy and multitude of interconnections, there are many possible paths between any two nodes. This causes the network diagnostic operator to generate many diagnostic procedures;
2. With the dynamic routing mechanism, the network can dynamically route packets around apparent faults or congestion. This dynamic rerouting can mask the network problem, and it can also cause overloading elsewhere in the network, leading the network diagnostic operator to an incorrect diagnosis; and,
3. With a high volume of information coming from the DCD, it is difficult to get a concise picture of the current situation. Also there can be so much "noise" that the network operators have difficulty determining when the network is starting to behave improperly. In both circumstances, it can be difficult for operators to recognize anomalous patterns and to act before the problem becomes very obvious and costly.

This thesis will tackle the second and third BNR diagnostic problems reported above.

Chapter 6

The Multiple Paradigm Diagnostic System

6.1 Introduction

In order to detect faults outlined in Section 5.5, a multiple paradigm diagnostic system seemed appropriate to be investigated because it reflected the way operators diagnose communication networks using different techniques. For this investigation, a project titled Pegasus, sponsored by the Telecommunications Research Institute of Ontario (TRIO), was initiated. The Pegasus project aim is to design and develop a multiparadigm diagnostic system that will perform automatic network fault diagnosis [PAGUREK91]. For this multiparadigm diagnostic system, different knowledge sources (KSs) were used and investigated. One of these investigations is the subject of this thesis.

This chapter briefly describes the PEGASUS project with its different knowledge sources, and presents justifications and reasons for doing this thesis.

6.2 The Pegasus Approach

The Pegasus approach is a multiparadigm approach which is composed of many KS integrated through a blackboard system (Figure 6.1). The main components of the Pegasus project are: the blackboard, the Dragon simulator, and three knowledge sources named: BANES, END and LEAPR.

6.2.1 The Blackboard

A blackboard system was used in the Pegasus project in order to integrate all the knowledge sources. From the point of view of the blackboard, each KS is seen as a specialist at solving certain aspects of the overall problem [CORKILL91]. One of its main characteristics is that it provides a suitable architecture to develop and test KSs separately. A KS can be added or removed from the blackboard at any time without affecting the other KSs.

The blackboard, which can be visualized as a global database, is subdivided into seven areas, each corresponding to a particular kind of information [IQNEIBI92]. The seven areas are : request, reply, status information, problems, hypothesis, traffic information² and finally Knowledge Source Activation Record (KSAR)³.

2: This area of the blackboard is not currently used and is not illustrated in figure 6.1. In order to get the traffic, the KS has to request the traffic via the request array.

3: This area of the blackboard is not currently used and is not illustrated in figure 6.1.

Figure 6.1: Pegasus's blackboard architecture.

The blackboard is a strictly passive component and represents a shared memory database. Each KS can independently enquire at one of the blackboard seven regions to obtain information.

6.2.2 The Simulator

In order to investigate this multiparadigm approach, a simulator that could simulate the BNR network was required. A simulator, called DRAGON, was built by one of the originators of the Pegasus project, Dr. N. Dawes. The DRAGON simulator behaves very much like the real BNR network would under normal conditions by including packet loss, error rates, node faults, data collector reports, and status messages; it even simulates

phone calls from users to the network operator in order to report problems [DAWES92]. In the Pegasus project, the DRAGON simulator posts problems and status messages to the blackboard, reads and processes requests coming from any KS via the blackboard, and then posts the replies back to the KS via the blackboard (Figure 6.1). In order to simulate a long time period, the simulator can run 50 times faster than realtime on medium scale WAN simulations.

6.2.3 The Knowledge Sources

The Pegasus project currently has three knowledge sources to diagnose particular network problems: a probabilistic paradigm approach called BANES, an expert system called END, and a neural network approach called LEAPR.

■ The BANES (Basic Analysis in Nested Evidence Space) knowledge source takes any set of evidence about some fault and reasons about the evidence using a model of the system. Its reasoning collects beliefs using a new kind of structure called Nested Evidence Space.

This knowledge source monitors the status messages provided by the simulator, and based on its reasoning system, it formulates a hypothesis to the blackboard of any fault it detects (Figure 6.1). Over extensive trials, this knowledge source detected 98 % of the faults where there was status information correlated to the fault [DAWES91a].

■ The END (Expert Network Diagnostician) knowledge source is an expert system built to diagnose workstations, routers and routing faults in a communication network [DUSSAULT92]. This knowledge source is written using the COSMIC CLIPS shell,

which is a C based expert system tool. END was designed by interviewing the BNR communication network experts to discover their methods of fault diagnosis. Their methods were then translated into rules in COSMIC CLIPS.

The END KS is divided into two expert systems (Figure 6.1): END #1 and END #2. The END #1 KS is always waiting at the blackboard hypothesis area. When there is a new hypothesis, through its rules, the END #1 KS analyses the problem and diagnoses whether it was a bad or good hypothesis. The END #2 KS always waits at the blackboard problems area. When a problem is reported, END #2 diagnoses the problem and outputs its diagnosis to the hypothesis blackboard area.

■ The LEAPR (Line Element Anomaly Pattern Recognition) knowledge source uses the neural network approach to diagnose anomalies in the communication network's traffic patterns. This knowledge source is the subject of this thesis.

Based on the traffic pattern data on selected lines of a router, the neural network learns what are the normal and abnormal traffic patterns. When these two classifications are learned by the neural network, this KS classifies the router as either NORMAL or ABNORMAL. When there is an ABNORMAL classification, the LEAPR KS outputs its diagnosis of a router problem to the blackboard. This router problem can then be further investigated by either the BANES or the END knowledge sources (Figure 6.1).

6.3 Thesis Justification

As outlined in Chapter 4, NNet techniques have been used for diagnostic purposes in medical fields, satellite communication and avionic maintenance system. In the

communication network field, NNet technologies have been mostly applied as a control mechanism [KHASNABISH91, CHEN91 and ANSARI91] and have not been fully investigated as a diagnostic tool.

Using a statistical method, an anomaly detector on a LAN with only one traffic input was investigated in [MAXION90]. This anomaly detector uses only one input; and traffic statistics for the different times of the day were used to characterize normal activity. The justification of this thesis is to generalize Maxion's idea and to use serial traffic data gathered at routers as the inputs to the neural network [PAGUREK91]. The NNet main advantage over Maxion's work is that a NNet will be able to recognise more complex patterns than Maxion's anomaly detector.

The comparisons presented in Section 4.5 reinforced the idea of using the NNet technology as a diagnostic tool in the Pegasus project.

6.4 LEAPR Implementation Goal

Due to its pattern recognition capabilities, the NNet is a very appropriate technique to provide early diagnosis of complex NORMAL / ABNORMAL traffic patterns. Early diagnosis of ABNORMAL traffic patterns will allow a network operator to correct situations immediately and to provide very efficient network operations. The LEAPR KS will also act as a proactive agent to signal abnormal traffic patterns to other KSs in the multiparadigm diagnostic system. The other KSs may use this information in their own diagnostic system, and so LEAPR enhances the overall diagnostic system.

Chapter 7

Development and Results

7.1 Introduction

The goal of this thesis is to determine if a general proactive KS could diagnose problems in any router in the simulated communication network. To achieve that goal, four prototypes were investigated before implementing the final LEAPR KS. This chapter presents the experiment setup conducted for each KS, the details of the backpropagation algorithm used in each KS, and also the different KSs investigated throughout this thesis.

7.2 Experiment Setup

The four prototypes (KSs), mentioned in Section 7.1, were developed using the following experiment setup:

- Using the DRAGON simulator, the router bCARc2 was chosen for diagnostic purposes (bCARc2 was selected randomly among all possible routers). From this router, only lines with a traffic mean above 17 Bytes/second (1,000 Bytes/minute) were

considered for diagnosis purposes. Lines traffic mean less than 17 Bytes/second were not considered because their traffic rate was often null, and when their traffic rate was not null, it was varying too much in short periods of time.

■ Once the lines were selected for diagnostic purposes, the backpropagation algorithm was trained with examples of NORMAL and ABNORMAL traffic pattern cases. These two traffic pattern cases were generated through the DRAGON simulator. Characteristics of these two cases will be explained later.

■ After having trained the NNet using the backpropagation algorithm, the classification knowledge acquired through the trained NNet weights was passed to the KS. At that stage, there was no more learning going on and traffic patterns were ready to be diagnosed by the KS.

After completion of the above three steps, each KS was separately tested with ABNORMAL and NORMAL traffic patterns. The ABNORMAL traffic patterns were simulated by introducing specific traffic increases on selected lines. The NORMAL traffic patterns were simulated during days of simulated communication network operations where the traffic varies during the day and where no errors were introduced. Figure 7.1 illustrates the average traffic variation of three lines during one day of simulated network operations: during non-working hours (from 18:00 to 08:00 hrs) the traffic is low, except between 03:00 and 04:00 where there are some management network activities occurring, and during working hours the traffic is high with a decrease in traffic during lunch time. This simulated traffic pattern variation was assumed to be realistic to any typical real network operations.

Figure 7.1 : One day of normal simulated network operations.

The reasons to conduct tests with normal simulated network operations were to get the percentage of NORMAL traffic pattern cases well classified and the false alarm⁴ ratio for each KS. The ultimate goal was to have the lowest false alarm ratio.

4: A false alarm is a diagnostic of an ABNORMAL traffic pattern case when the traffic pattern was NORMAL. I will continue to use the word false alarm to mean the above description.

7.3 The Backpropagation Algorithm

In order to capture the knowledge in traffic patterns, a three-layer⁵ backpropagation algorithm, as detailed in Chapter 3, was implemented in C on a SPARCstation 2. The backpropagation algorithm was verified with independent results from the commercially available NeuralWorks Professional II software package.

The implemented backpropagation algorithm has only one OUTPUT layer PE. Since its output varies between 0.0 and 1.0, for all prototypes in this thesis, NORMAL traffic pattern cases were trained with a desired output of 0.1 and ABNORMAL traffic pattern cases were trained with a desired output of 0.9. After the training phase, a value at the output node greater than 0.5 signifies an ABNORMAL traffic pattern, while a value less than or equal to 0.5 signifies a NORMAL traffic pattern.

Depending on the number of iterations the backpropagation algorithm went through during its training phase, the MOMENTUM and LCOEF parameters, described in Chapter 3, were modified dynamically according to Table 7.1.

	Number of iterations in the interval of				
	0 to 29,999	30,000 to 54,999	55,000 to 74,999	75,000 to 94,999	95,000 to infinity
LCOEF	0.5	0.45	0.4	0.3	0.1
MOMENTUM	0.9	0.9	0.9	0.7	0.3

Table 7.1: MOMENTUM and LCOEF parameters.

5: The term "three-layer" used in this thesis refers to an INPUT, a HIDDEN and an OUTPUT layer.

Table 7.1 represents an adaptive learning rate algorithm intended to adjust the LCOEF and the MOMENTUM parameters automatically as the learning process proceeds [KLIMASAUSKAS89]. This table starts with a basic LCOEF and MOMENTUM parameter, and as the training proceeds and reaches a pre-selected number of iterations threshold, the LCOEF and MOMENTUM parameters are changed as indicated by Table 7.1.

In each prototype backpropagation algorithm, the training phase is the most critical part because the backpropagation algorithm learns what is presented to it. If the training set is composed of erroneous data, the backpropagation algorithm will learn these erroneous data. In each prototype, the training set of every backpropagation algorithm had approximately the same number of NORMAL and ABNORMAL traffic patterns. Via trial and error experiments, it was noticed that with a balanced training set (approximately the same number of examples for each category), the backpropagation algorithm results were better.

Also, for each prototype, every backpropagation algorithm was trained with a convergence threshold set at 0.00001. A convergence threshold is a threshold which ends the training phase of a backpropagation algorithm based on the difference between an actual NNet output and a desired output. When the difference (convergence threshold) was less than or equal to 0.00001, the training phase of the backpropagation algorithm was terminated.

7.4 LEAPR Development Sequence

The development of the LEAPR KS was done in an incremental way. A prototype KS was implemented and evaluated, and based on its evaluation, another prototype KS was implemented. Four prototypes were implemented and tested before the implementation of the general proactive LEAPR KS. A brief description of every prototype KS and the order in which they were implemented is presented below in order to give the reader an overview of the LEAPR development conducted during this thesis. The details of each prototype will be discussed later.

For the first prototype KS, the complete traffic pattern was divided into five different cases where each traffic pattern case was individually learned by a NNet. For this prototype, an ABNORMAL traffic pattern was characterized by one line traffic increase of more than 4.7 standard deviations above its traffic mean (see Section 7.5.1). This prototype's goal was to divide the big traffic pattern problem into smaller traffic pattern problems; this idea came from the lesson learned in the DASA/LARS diagnostic system [CAUDILL90] presented in Section 4.2. During this first prototype, it was observed that diagnosing traffic patterns in real time creates many false alarms. Based on that observation, diagnosing traffic patterns fifteen minutes behind real time, as it will be described later in Section 7.5.4.1, was investigated and it reduced the number of false alarms considerably. Even if the results of the first prototype were satisfactory and as it will be shown later, this prototype was not adequate.

Based on the experience gained during the first prototype, a second prototype was built using only one NNet. The inputs fed to the NNet were ratios between the line's

traffic fifteen minutes ago and the line traffic mean during the last thirty minutes. Using this approach, the ratios of NORMAL cases and ABNORMAL cases, characterized by 4.7 standard deviations above the mean, were too close. With this similarity between the two categories, the backpropagation algorithm was not able to classify efficiently the traffic pattern. So, starting at this prototype KS, ABNORMAL cases were characterized by a line traffic increase greater than 266 Bytes/second. This second prototype KS demonstrated that a general NNet diagnostician could work efficiently. However, the number of ABNORMAL traffic increases detected by prototype #2 was very low. So, using the same approach as in the second prototype, the third prototype KS was designed to diagnose the traffic pattern in real time. The third prototype KS was much better than the second prototype because it could detect more consecutive ABNORMAL traffic patterns.

During all previous KS prototypes, only three lines were monitored for diagnosis purposes. For the fourth KS prototype, four lines were monitored. The reason for increasing the number of monitored lines from three to four was to feed the NNet with the two largest and two smallest traffic ratios. The results of the fourth KS prototype were very encouraging. Finally, based on the fourth prototype, the LEAPR KS was implemented. The LEAPR KS monitored ten different routers in the DRAGON simulator and its percentage of NORMAL cases correctly classified was greater than 99.8%. The percentage of ABNORMAL cases correctly classified was between 34% and 100% depending on the monitored router and on the RC threshold value. This percentage of ABNORMAL cases will be explained in Section 7.9.4.

7.5 Prototype KS #1

The approach used during the development of this prototype was to divide the complete traffic pattern into cases of particular traffic patterns; instead of having one large NNet to learn all possible traffic patterns, individual NNets were used to learn particular traffic patterns.

For this prototype, the DRAGON simulator was used and three lines of router bCARc2 were monitored for diagnostic purposes. The lines monitored were the line between router bCARc2 and bQNEc1, the line between bCARc2 and bFTZc1, and finally, the line between bCARc2 and bPINc1 (Figure 5.2). From these selected lines, line traffic rates (i.e. number of bytes/minute) were used for monitoring purposes.

There were four major steps to be conducted before evaluating this prototype. The first step was to define a congestion problem. The second step was to determine how many NNet cases would be needed to monitor the three bCARc2's lines. The third step was to train and test each NNet with appropriate data, and the fourth step was to integrate all these separated trained NNets into a KS and to test it.

7.5.1 Congestion Problem Description

In general, one router congestion problem is observed by the network operator per week. Because the routers are polled for monitoring purposes every minute, the shortest possible period of time a congestion problem could be observed is one minute. So the chance of having a congestion problem is greater than or equal to 1 per 10080 minutes (7 days/week $\times$ 1440 minutes/day). Since there are approximately 100 routers in the

entire BNR network, the probability of having a congestion problem on a particular router during any minute is greater than or equal to 10^{-6} . Assuming the line traffic is normally distributed and assuming the congestion problem lasts for only one minute, the probability that traffic does increase from the mean by more than 4.7 standard deviations is approximately 10^{-6} .

Based on the number of router congestion problems and on the number of routers in the BNR network, the 4.7 standard deviation mark is the boundary that has been set for this prototype KS and it has only been used by this prototype.

Using these assumptions, if one line's traffic is above the 4.7 standard deviation mark, the traffic pattern (the 3 lines monitored) is labelled ABNORMAL and if all the three lines traffic rates are below the 4.7 standard deviation mark, the traffic pattern is labelled NORMAL.

7.5.2 Number of Neural Network cases needed

Depending on the number of workstations logged in to the communication network, the normal traffic level changes. Workstations were added to the DRAGON simulator, and based on the number of workstations added, the average traffic standard deviation and traffic mean of the three bCARc2's monitored lines were computed (Figure 7.2). The approach used to determine the number of NNet cases needed to diagnose bCARc2 was to choose a window⁶ where each extremity of the window is $\pm$ one standard

6: A window is a range of extra workstations added to the basic DRAGON simulator (ex: 0 - 2,100 workstations represents the window for case #1).

deviation of the window middle point's⁷ traffic pattern (Figure 7.3). The $\pm$ one standard deviation was estimated to be reasonable since 4.7 standard deviations above the mean was used as the classification boundary between NORMAL and ABNORMAL traffic patterns. As indicated in Figure 7.2, five NNets were needed; one for each case.

Figure 7.2: bCARc2's global traffic pattern vs number of workstations.

7: The traffic pattern at the window middle point is 4.7 standard deviations above the traffic mean.

Figure 7.3: A window of number of workstations.

7.5.3 Training and Testing for each case

Once the number of NNet cases was determined, each NNet had to be trained. Each NNet case was trained using the three-layer backpropagation algorithm. For every case, the NNet was composed of 3 Processing Elements (PEs) in the INPUT layer, 12 PEs in the HIDDEN layer and 1 PE in the OUTPUT layer. Concerning the number of PEs in the HIDDEN layer, trial and error experiments revealed that with a ratio of 4 PEs in the HIDDEN layer to 1 PE in the INPUT layer, the backpropagation algorithm works better.

As stated in Section 7.3, it should be remembered that a NNet output value greater than 0.5 was used to classify an ABNORMAL case and a NNet output value less than or

equal to 0.5 was used to classify a NORMAL case. The training details of each NNet case were as follows:

■ For case #1, 1,000 workstations were randomly added to the DRAGON simulator and the training set was composed of 28 NORMAL and 24 ABNORMAL traffic pattern examples. For this case, when input to the trained NNet, all training set NORMAL traffic patterns were classified with a NNet output less than 0.2 and all training set ABNORMAL traffic patterns were classified with a NNet output above 0.8 (Figure 7.4). Therefore, there were no errors.

■ For case #2, 3,300 workstations were randomly added to the DRAGON simulator and the training set was composed of 32 NORMAL and 30 ABNORMAL traffic pattern examples. For this case, 31 out of 32 training set NORMAL traffic patterns were classified with a NNet output less than 0.5, and 28 out of 30 training set ABNORMAL traffic patterns were classified with a NNet output above 0.5 (Figure 7.5). For this case, based on the training set results, there was one false alarm and two missed ABNORMAL cases⁸.

■ For case #3, 6,000 workstations were randomly added to the DRAGON simulator and the training set was composed of 29 NORMAL and 29 ABNORMAL traffic pattern examples. For this case, all training set NORMAL traffic patterns were classified with a NNet output less than 0.2 and all training set ABNORMAL traffic patterns were classified with a NNet output above 0.7 (Figure 7.6). Based on case #3 training set results, there were no errors.

8: A missed ABNORMAL case is a diagnostic of a NORMAL case when it was an ABNORMAL case.

■ For case #4, 8,800 workstations were randomly added to the DRAGON simulator and the training set was composed of 30 NORMAL and 30 ABNORMAL traffic pattern examples. For this case, all training set NORMAL traffic patterns were classified with a NNet output less than 0.2, hence no false alarms, and 21 out of 30 training set ABNORMAL traffic patterns were classified with a NNet output above 0.5 (Figure 7.7). Based on case #4 training set results, there were nine misses of ABNORMAL traffic pattern cases; case #4 then will not be able to diagnose 100 % of all the ABNORMAL cases.

■ For case #5, 12,250 workstations were randomly added to the DRAGON simulator. The training set was composed of 30 NORMAL and 30 ABNORMAL traffic pattern examples. For this case, all training set ABNORMAL traffic patterns were classified with a NNet output greater than 0.7, and 27 out of 30 training set NORMAL traffic patterns were classified with a NNet output less than 0.5 (Figure 7.8). For this last case, based on the result of the training set, there were three false alarms; case #5 then will provide some inaccurate readings by classifying NORMAL patterns as ABNORMAL.

Figure 7.4: Training set's histogram of case #1.

Figure 7.5: Training set's histogram of case #2.

Figure 7.6: Training set's histogram of case #3.

Figure 7.7: Training set's histogram of case #4.

Figure 7.8: Training set's histogram of case #5.

For each of the previous five NNet cases, the training was completed after a different number of epochs⁹ (Table 7.2). As shown below, in Table 7.2, the training period on a SPARCstation 2 varied with the number of epochs.

Case #	Number of epochs	CPU time
1	502.8	109.3 seconds
2	304.7	85.9 seconds
3	470.2	114.0 seconds
4	137.6	33.8 seconds
5	193.2	23.3 seconds

Table 7.2: Training period for each neural network case.

After the training phase was completed, tests were conducted to see what each NNet case would diagnose as ABNORMAL traffic patterns using different data. In order

9: An epoch is the number of times the backpropagation algorithm went through the entire training set.

to do that, the number of standard deviations above the traffic mean was increased on an individual line with the two other lines constantly at their traffic mean. Table 7.3 indicates the number of standard deviations above the line traffic mean where abnormalities were diagnosed. For NNet case #1, depending on the line monitored and using the 0.5 output threshold, abnormalities were indicated when traffic was between 3.3 and 4.3 standard deviations above the respective line's traffic mean. Since abnormalities are defined as a traffic pattern at 4.7 standard deviations above the mean, when NNet case #1 is used, false alarms should be expected. As in case #1, for case #5, there should also be some false alarms because abnormalities were diagnosed between 4.1 and 4.5 standard deviations above the respective line's traffic mean. For NNet case #2, there should also be some false alarms on bCARc2's line to bPINc1 because abnormalities were diagnosed above 3.9 standard deviations above the line's traffic mean. For NNet case #3, since traffic abnormalities detected were approximately 4.7 standard deviations above the traffic mean, no false alarms or missed cases should occur. For case #4, since abnormalities were detected between 5.2 and 5.8 standard deviations above the traffic mean, some ABNORMAL traffic pattern cases should be missed.

Congestion Problem on bCARc2	Neural Network Case Number				
	1	2	3	4	5
line to bQNEc1	4.3	4.8	4.9	5.2	4.5
line to bFTZc1	4.0	5.2	4.7	5.5	4.1
line to bPINc1	3.3	3.9	4.9	5.8	4.5

Table 7.3: Number of standard deviations above the traffic mean where abnormalities were diagnosed.

7.5.4 Autonomous Diagnostic System.

In order to follow the network traffic variation, the prototype KS #1 stores the last thirty minutes of the monitored line's traffic, and based on the traffic mean for the three monitored lines during the last thirty minutes, the prototype KS #1 decides which NNet case to use for diagnostic purposes. This case mechanism can be briefly explained using the following pseudo-code:

```
INFINITE LOOP
{
  KEEP THE LAST 30 MINUTES OF TRAFFIC
  FOR EACH MONITORED LINES;

  CALCULATE THE TRAFFIC MEAN OF THE MONITORED LINES
  DURING THE LAST 30 MINUTES;

  IF (MEAN < 4,300) THEN
    USE NNet CASE #1;
  ELSE
    IF (MEAN < 5,700) THEN
      USE NNet CASE #2;
    ELSE
      IF (MEAN < 7,400) THEN
        USE NNet CASE #3;
      ELSE
        IF (MEAN < 9,400) THEN
          USE NNet CASE #4;
        ELSE
          USE NNet CASE #5;
}
```

7.5.4.1 Results.

The prototype KS #1 was tested during one day of normal simulated network operations. The purpose of that experiment was to check the efficiency of the KS to change from one NNet case to another and to check its false alarm ratio.

During this experiment, two tests were conducted. Both tests used the last 30 minutes of traffic to determine the mean that decides which NNet case to use to diagnose the network. The first test (test #1) was diagnosing the simulated network in real time (i.e. as soon as the mean is computed - within one or two seconds). For this test, the percentage of false alarms was less than 3.62 % (Table 7.4). In analyzing the first test's data carefully, it was observed that the false alarms were coming from a normal traffic variation from a lower to a higher traffic rate (Figure 7.9). This observation can be explained by the fact that the prototype KS did not change the NNet case (i.e. go from case #1 to case #2) to account for the increase in the traffic pattern. The KS did not change from one NNet case to another because the traffic pattern increase did not modify the traffic mean enough (i.e. it takes time before the traffic pattern increase modifies the traffic mean in order to change from one NNet case to another). In order to solve this problem, the idea of diagnosing the traffic pattern fifteen minutes behind real time (test #2) was introduced. For test #2, the KS kept the traffic of the last fifteen minutes for every monitored line and it classified the traffic pattern fifteen minutes behind real time based on the traffic mean for the last 30 minutes.

Number of workstations	NORMAL Traffic Pattern well classified	False Alarms
1,250	99.14 %	0.86 %
3,300	96.38 %	3.62 %

Table 7.4: Prototype KS #1's performance in real time (test #1).

In comparison with test #1, the percentage of false alarms in the second test (test #2) was much less (Table 7.5). However, when test #2 was conducted with 6,000 workstations added to the DRAGON simulator, the number of false alarms increased significantly (Table 7.5). This increase was due to the fact that the KS case-mechanism used NNet case #5, which is poor at diagnosing NORMAL cases. NNet case #5 was not reached previously because there were not enough workstations added to increase the traffic pattern above 9,400 Bytes/minute.

Number of workstations	NORMAL Traffic Pattern well classified	False Alarms
1,250	99.74 %	0.26 %
3,300	98.42 %	1.58 %
6,000	86.15 %	13.85 %

Table 7.5: Prototype KS #1's performance fifteen minutes behind real time (test #2).

Figure 7.9: Daily traffic variation of 3 lines from bCARc2.

Based on the previous results, prototype KS #1 had better results when it was diagnosing the traffic patterns fifteen minutes behind real time. However, when this KS picked NNet case #5, the number of false alarms increased. This problem can be corrected by retraining NNet case #5 in particular in order to have an overall performance with less false alarms.

The retraining phase could be done, but overall this KS was considered not flexible enough to diagnose different kind of routers. This KS would only be able to diagnose the kind of router for which it had been trained on. If the prototype KS #1 was to be used to diagnose any other routers, it would probably have to be retrained, and the number of NNet cases would probably have to be re-examined also.

7.6 Prototype KS #2

In order to generalize the KS to be able to diagnose any kind of router, the prototype KS #2 was investigated. Since the accuracy of prototype KS #1 was better diagnosing the traffic pattern fifteen minutes behind real time, prototype KS #2 also monitored the traffic pattern fifteen minutes behind real time. This prototype monitors the same three lines as prototype #1 and is composed of a pre-processing module and a neural network (Figure 7.10).

Figure 7.10: Prototype KS #2 components.

The pre-processing module calculates the line traffic mean for the last thirty minutes and keeps the traffic of the last fifteen minutes for every monitored line. With these two values, the pre-processing module calculates the ratio between the line's traffic fifteen minutes ago and the line traffic mean for the last thirty minutes. Once the line's

ratio was calculated, the pre-processing module forwarded the lines' ratio to the neural network in order to classify the traffic pattern as NORMAL or ABNORMAL.

For this prototype, 1,250 workstations were randomly added to the DRAGON simulator. The training set was composed of 64 NORMAL and 48 ABNORMAL traffic pattern cases. Rather than using the +4.7 standard deviation as the boundary between NORMAL and ABNORMAL cases (as in prototype KS #1), ABNORMAL cases were characterized by a line traffic deviation of more than 266 Bytes/second (16 KBytes/minute) above its mean. This ABNORMAL case characteristic was assumed to be realistic. The ABNORMAL cases were taken from six different combinations of line traffic increases (Table 7.6). During training, ABNORMAL traffic pattern cases were generated through the simulator and were characterized by a traffic deviation approximately between 266 Bytes/sec and 350 Bytes/sec above its mean for at least six minutes. This ABNORMAL characteristic will also apply to prototypes #3 and #4, which will be described later.

Type of ABNORMAL Traffic	lines			Type of ABNORMAL Traffic	lines		
	1	2	3		1	2	3
err #1	0	0	1	err #4	1	0	0
err #2	0	1	0	err #5	1	1	0
err #3	0	1	1	err #6	1	0	1
Legend : 1 represents a traffic increase 0 represents Normal Operations							

Table 7.6 : Type of ABNORMAL traffic pattern cases in the training set of prototype KS #2.

With the training set described above, the backpropagation algorithm training phase was completed after 795.6 epochs (89 106 iterations) and took 457.56 seconds of CPU time on a SPARCstation 2.

For this prototype, the training results were satisfactory (Figure 7.11). All training set ABNORMAL traffic pattern cases were classified with a NNet output above 0.5, and 58 out of 64 training set NORMAL traffic pattern cases were classified with a NNet output less than 0.5 (Figure 7.11). According to the training set, six NORMAL traffic patterns were classified as ABNORMAL; thus, there were six false alarms in the training set.

In order to evaluate this prototype, fifteen days of normal simulated communication network operations were monitored by this prototype. With 1250 workstations, the false alarm rate was only 0.36%. With the same amount of workstations, another test was conducted in order to determine if the KS could detect the different types of ABNORMAL traffic patterns (Table 7.6) that the backpropagation algorithm was trained on. The results indicated that the KS could diagnose all ABNORMAL traffic pattern types at least one time during a twenty minutes period of abnormalities.

Figure 7.11: Training set's histogram of prototype KS #2.

Based on these above results, this experiment has demonstrated that a general NNet diagnostician could work efficiently.

7.7 Prototype KS #3

Prototype KS #3 is exactly the same as prototype #2 except that it diagnosed the traffic pattern in real time instead of diagnosing the traffic pattern fifteen minutes behind real time. By diagnosing the traffic patterns in real time, the ratio between the actual line traffic rate and its traffic mean for the last thirty minutes varied slightly during normal simulated network operations and varied significantly when ABNORMAL traffic increases occur. This caused a bigger gap between the ABNORMAL and NORMAL cases.

For this prototype, 1,250 workstations were randomly added to the DRAGON simulator. The training set was composed of 46 NORMAL and 36 ABNORMAL traffic

pattern cases. The backpropagation algorithm training phase was completed after 276.08 epochs (22,639 iterations) and took 167.73 seconds of CPU time on a SPARCstation 2. The reason why this backpropagation algorithm was trained quicker than prototype #2 was that, with the pre-processing module of prototype #3, the difference between NORMAL and ABNORMAL cases was more apparent.

For this backpropagation algorithm, all training set NORMAL cases were classified with a NNet output less than 0.2 and all training set ABNORMAL cases were classified with a NNet output above 0.7 (Figure 7.12).

In order to evaluate this KS, sixty days of normal simulated network operations were monitored by this prototype. With 1,250 workstations¹⁰ randomly added to the simulated network, no false alarms were detected. Also for each ABNORMAL traffic pattern type, there were at least six consecutive ABNORMAL cases detected (Table 7.7).

Another test was conducted with 6,000 workstations, representing an average traffic mean of 6,700 Bytes/minute. This test was conducted for a period of two days of simulated network operations and, no false alarms were detected. The goal of this test was to see if this prototype could still diagnose a 16 Kbytes/minute line traffic increase buried in a higher normal traffic rate. As illustrated in Table 7.7, this prototype was also able to detect the different types of ABNORMAL traffic patterns detected with 1,250 workstations, but the number of ABNORMAL diagnostics was less than expected. This means that, with this approach, ABNORMAL line traffic increases are relative to the line's traffic rate; as a line's traffic rate increases, the number of 16 Kbytes/minute traffic

10: 1,2500 workstations randomly added to the simulated network (DRAGON) represent an average traffic mean of 3,700 Bytes/minute.

increases detected as ABNORMAL decreases.

By comparing prototype #3 with prototype #2, it can be said that prototype #3 is much better because the diagnostics were done in real time and because a larger number of consecutive ABNORMAL traffic patterns were detected.

Figure 7.12: Training set's histogram of prototype KS #3.

Types of ABNORMAL Pattern	A Simulated Network with	
	1,250 workstations	6,000 workstations
error 1	6	4
error 2	8	4
error 3	9	7
error 4	1	5
error 5	8	4
error 6	10	5

Table 7.7: Number of consecutive times ABNORMAL cases were detected in different simulated networks.

7.7.1 Neural Network Learning

Before progressing any further, the backpropagation algorithm learning process, through the examples given in the training set, will now be illustrated. The next two figures (Figure 7.13 and 7.14) are illustrating a 3-dimensional image with a front, a top and a side view. The t_1 , t_2 and t_3 parameters represent respectively the ratio between the actual traffic and the traffic mean of the last thirty minutes for monitored lines 1, 2 and 3. The backpropagation algorithm of prototype KS #3 was trained with traffic patterns illustrated in Figure 7.13, and based on this, it learned to classify NORMAL cases from ABNORMAL cases and vice-versa. The learning result (Figure 7.14) illustrates the non-linearity of the boundary between NORMAL and ABNORMAL traffic pattern cases. The boundary was found by flooding the backpropagation algorithm with different ratios and by picking the ratios corresponding to a NNet output close to 0.5.

Figure 7.13: Training set for the backpropagation algorithm of prototype KS #3.

Figure 7.14: The decision boundaries learned by the backpropagation algorithm.

7.8 Prototype KS #4

Based on prototype KS #3 and as an incremental step toward a general LEAPR KS implementation, prototype KS #4 was introduced to monitor four lines. The four lines monitored are the line between bCARc2 and bQNEc1, the line between bCARc2 and bFTZc1, the line between bCARc2 and bPINc1, and the line between bCARc2 and bKANc2 (Figure 5.2).

By comparison to prototype #3, this prototype KS's architecture was slightly different. Prototype #4 was composed of a "TOP TWO and BOTTOM TWO" module, a neural network and a Repetitiveness Counter (RC) module (Figure 7.15). The reason to increase the number of monitored lines from three to four was to accommodate the "TOP TWO and BOTTOM TWO" module which requires four inputs.

Figure 7.15: Prototype KS #4 components.

Before feeding the traffic pattern to the NNet for diagnosis, the traffic pattern went through the "TOP TWO and BOTTOM TWO" module. This module calculated the line's traffic mean for the last thirty minutes, and calculated the ratio between the line's actual traffic and the line traffic mean for the last thirty minutes. Once each line's ratio was calculated, the traffic ratio was sorted in order to have the biggest traffic ratio as input #1 into the NNet, the second biggest traffic ratio as input #2, the second lowest traffic ratio as input #3, and the lowest traffic ratio as input #4. This method, which considers the two largest and two lowest value, did prove to be useful in recognition of rare abnormal events from a mass of normal cases [DAWES92a].

After the "TOP TWO and BOTTOM TWO" module, there was the NNet which classified the traffic pattern as either an ABNORMAL or a NORMAL case. In both cases, the NNet output was forwarded to the RC module in order to determine the repetitiveness of a categorized ABNORMAL traffic pattern. The way the repetitiveness counter works was that when there was an ABNORMAL case, the repetitiveness counter (RC) was multiplied by a factor of 0.9 and increased by 1.0 ($RC = (RC \times 0.9) + 1.0$). When there was a NORMAL case, the repetitiveness counter was decreased by a factor of 0.1 ($RC = RC \times 0.9$). Depending on the time period over which the traffic increase is diagnosed as ABNORMAL, the RC will be increased as indicated by Table 7.8 (assuming the RC is equal to zero before the abnormality starts). For visual clarity, Table 7.8 presents the RC formula outputs when there are ABNORMAL cases.

The main reason to implement the RC module was to reduce the number of false alarms. By monitoring the repetitiveness of ABNORMAL diagnostics from the NNet, the

KS will be able, based on its RC threshold value, to either accept the NNet's ABNORMAL diagnostic or to reject it. Even if the RC module was used in this prototype, the RC value was only considered while this KS was tested to see if it could detect the different ABNORMAL traffic patterns that the backpropagation algorithm was trained on. The RC module was investigated on this prototype in order for its value to be analyzed during abnormalities detection and for it to be used in the final KS. The RC module will be explained in more details when it will be used in the LEAPR KS (Sections 7.9.3 and 7.9.4).

Time Period (minutes)	RC value	Time Period (minutes)	RC value
0	0.0	8	5.69
1	1.0	9	6.12
2	1.9	10	6.51
3	2.71	11	6.86
4	3.44	12	7.17
5	4.09	13	7.46
6	4.68	14	7.71
7	5.21	15	7.91

Table 7.8: RC values based on the duration of the abnormality.

7.8.1 Training Set

This prototype was trained with fifteen different types of ABNORMAL traffic patterns referred to as errors (Table 7.9). These fifteen types of errors represent all the possible combinations of errors for the four lines. Approximately seven cases of each type were inserted as ABNORMAL cases in the training set. The training set for this prototype was composed of 136 NORMAL and 106 ABNORMAL traffic pattern cases.

Type of ABNORMAL Patterns	lines				Type of ABNORMAL Patterns	lines			
	1	2	3	4		1	2	3	4
err #1	0	0	0	1	err #9	1	0	0	1
err #2	0	0	1	0	err #10	1	0	1	0
err #3	0	0	1	1	err #11	1	0	1	1
err #4	0	1	0	0	err #12	1	1	0	0
err #5	0	1	0	1	err #13	1	1	0	1
err #6	0	1	1	0	err #14	1	1	1	0
err #7	0	1	1	1	err #15	1	1	1	1
err #8	1	0	0	0					
Legend : 1 represents traffic increase 0 represents Normal Operations									

Table 7.9 : Type of ABNORMAL traffic patterns in the training set of prototype #4.

7.8.2 The Backpropagation Algorithm Architecture

Since the number of inputs has increased from three inputs (prototype #3) to four inputs (prototype #4), different backpropagation algorithm architectures were investigated in order to come with the best algorithm. Three different architectures were tested. All the three architectures had 4 PEs in the INPUT layer and 1 PE in the OUTPUT layer. The only difference in the three architectures was the number of PEs in the HIDDEN layer: the first architecture had 9 PEs (Figure 7.16), the second had 12 PEs (Figure 7.17) and the third had 16 PEs (Figure 7.18).

Even though the performance of all the different backpropagation algorithm architectures were similar, it was decided to choose the architecture with 12 PEs in the HIDDEN layer. The main reason for this choice was that 135 out of 136 training set NORMAL cases were classified with a NNet output less than 0.5 and that all training set

ABNORMAL cases were classified with a NNet output above 0.5. The two other architectures were detecting 134 out of 136 training set NORMAL cases.

Figure 7.16 : Training set's histogram of the backpropagation algorithm with 9 PEs in the HIDDEN layer.

Figure 7.17: Training set's histogram of the backpropagation algorithm with 12 PE in the HIDDEN layer.

Figure 7.18: Training set's histogram of the backpropagation algorithm with 16 PE in the HIDDEN layer.

7.8.3 Prototype KS #4 Performance

Once the backpropagation algorithm architecture was chosen and trained, different experiments were conducted in order to evaluate its performance. The first experiment conducted was with 1,250 workstations randomly added to the DRAGON simulator. This experiment was conducted on a period of sixty days of normal simulated network operations, and 99.966% of all the NORMAL cases were detected.

A second experiment was conducted with 6,000 workstations randomly added to the DRAGON simulator. This experiment was conducted during seven days of normal simulated network operations, and 99.99% of all the NORMAL cases were detected.

The third experiment conducted was to check if the KS could detect different ABNORMAL traffic patterns of the types presented in Table 7.9. The result of this experiment was that all fifteen different ABNORMAL traffic pattern types, which were generated for approximately twenty minutes, were detected at least seven times with a maximum RC value of at least 4.79 (Table 7.10). Table 7.10 presents the number of intervals in which ABNORMAL cases were correctly diagnosed followed by the maximum RC value encountered by this prototype KS.

Type of ABNORMAL Patterns	Number of intervals in which ABNORMAL cases were diagnosed correctly	Maximum value of the Repetitiveness Counter
error 1	9	6.12
error 2	7	4.79
error 3	11	6.51
error 4	9	5.78
error 5	11	6.86
error 6	12	6.44
error 7	14	7.12
error 8	11	6.12
error 9	12	6.86
error 10	11	6.86
error 11	12	6.72
error 12	13	7.17
error 13	13	7.17
error 14	14	7.71
error 15	15	7.71

Table 7.10: Number of ABNORMAL cases detected.

From these previous results, it can be said that prototype KS #4 was performing very well. Its false alarm ratio was very low, and it detected abnormalities during at least seven intervals for each trained ABNORMAL type with a maximum encountered RC value above 4.79. This prototype demonstrated that the RC value can be used with the NNet in order to classify the traffic patterns as NORMAL or ABNORMAL.

This prototype was only able to monitor four traffic lines. Since the thesis' ultimate goal was to have a KS that could diagnose any type of routers of the network with any number of lines, another KS had to be investigated.

7.9 LEAPR

The final prototype, the LEAPR KS, uses the same NNet and is composed of the same modules as in prototype KS #4, except that this final prototype monitors more than four lines and considers the RC value during its diagnosis (Figure 7.19). Out of all the lines connected to the "TOP TWO and BOTTOM TWO" module, only four ratios are forwarded to the NNet for diagnosis purposes.

Figure 7.19: LEAPR knowledge source components.

7.9.1 LEAPR TOP TWO and BOTTOM TWO Module

The main purpose of the "TOP TWO and BOTTOM TWO" module is to monitor any router which has more than four lines. With this module, the LEAPR KS monitors more than four lines by keeping only the two biggest and two lowest values of all the values sorted. In doing so, the values in between the two extremities are discarded for diagnosis purposes.

In a case where there are six traffic lines monitored by the LEAPR KS, a typical example of sorted traffic ratios processed by the "TOP TWO and BOTTOM TWO" module would be 2.9, 1.7, 1.1, 1.02, 0.98 and 0.97. This method, which takes only the two biggest and two lowest ratios, proved to be useful in recognition of rare abnormal events from a mass of normal cases [DAWES92a].

Using this method, the absolute sum value of the NNet's weights from the INPUT layer PE 1, 2, 3 and 4 to all the HIDDEN layer PEs are respectively 69.74, 24.46, 13.12 and 20.46. This set of weights shows that the most important traffic ratio to classify the traffic pattern is the biggest ratio.

7.9.2 LEAPR Performance¹¹

The performance tests to be conducted on this KS were the same three tests conducted on prototype KS #4, except that, instead of monitoring only four lines, the

11: For this particular performance test, as in LEAPR prototype #4, the RC value was only considered while the LEAPR KS was tested with the different ABNORMAL traffic patterns that the backpropagation algorithm was trained on.

LEAPR KS was monitoring twelve lines. The reason to conduct the same tests was to make sure that by monitoring more lines, the KS would have approximately the same results.

The results of LEAPR's three tests were very similar to the results of prototype KS #4. Since the results were very similar, it was confirmed that the LEAPR KS could diagnose more than four monitored lines accurately.

7.9.3 LEAPR Versatility

The last performance results demonstrated that the LEAPR KS could diagnose more than four lines. However, tests to see if the KS could diagnose traffic patterns from unseen routers of the simulated network had to be conducted. In order to do so, ten randomly picked routers were monitored during fourteen days of normal simulated network operations, and they were also monitored for a fixed period of time to see if they could diagnose ABNORMAL traffic patterns. For these two tests, the RC threshold was set to 1.91. It was set to 1.91 in order not to detect any abnormalities which would occur twice consecutively (assuming the repetitiveness counter is equal to zero before abnormalities start). For the ten randomly picked routers, the percentage of NORMAL traffic pattern cases correctly classified was greater than 99.802% and averaged out to 99.95% (Table 7.11). Concerning the percentage of ABNORMAL traffic pattern cases well classified, it will be presented in the next section.

21 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO 1.2 EQUIVALENT

Router monitored	Number of lines monitored	Percentage of NORMAL cases well classified	Number of False Alarms per day
nANNc1	5	99.985 %	0.21
nBDCc1	4	99.965 %	0.35
bCARc4	11	99.96 %	0.57
nCLDc1	4	99.976 %	0.5
nLAKc2	4	100 %	0.0
nLAUc1	5	99.965 %	0.5
bMERc2	12	100 %	0.0
nMTHc1	7	99.92 %	1.14
bRCHc1	5	99.955 %	0.64
nTOKc1	4	99.802 %	2.86

Table 7.11: LEAPR performance given the RC threshold value was set to 1.91.

Another test to verify what traffic increase the LEAPR KS could diagnose correctly also was conducted. In order to do so, using the same ten routers, the traffic was increased individually on each routers' line for approximately twenty minutes. The LEAPR KS was then tested to detect any ABNORMAL traffic pattern, Table 7.12 shows the results of this test. Depending on the line traffic increase, sometimes the KS could not diagnose any pre-selected traffic increases, or it could diagnose all pre-selected traffic increases. For these two possible scenarios, a column titled "minimum" was inserted in Table 7.12 to indicate the minimum limit of what the KS could diagnose as ABNORMAL traffic pattern increase.

Traffic on the line monitored (Bytes/second)	Pre-selected Traffic Increase in Bytes/sec						
	133	260	530	800	1060	1330	minimum
23	✓	✓	✓	✓	✓	✓	43
30	✓	✓	✓	✓	✓	✓	53
65	✓	✓	✓	✓	✓	✓	133
83	✓	✓	✓	✓	✓	✓	133
125		✓	✓	✓	✓	✓	
333			✓	✓	✓	✓	
450				✓	✓	✓	
633					✓	✓	
750						✓	
916							1 600

Table 7.12: LEAPR performance on detecting abnormal pre-selected traffic increases.

In Table 7.12, a ✓ indicates that the single pre-selected traffic increase is diagnosed as an ABNORMAL traffic pattern. When there is no ✓, it means the single traffic increase could not be diagnosed as an ABNORMAL traffic pattern given the RC threshold value is set to 1.91, because the increase was not big enough. This does not mean that it is a missed ABNORMAL case because, as expressed in Section 7.7 and since the NNet is using the ratio between the actual line's traffic and its mean for the last thirty minutes, ABNORMAL line traffic increases are relative to the line traffic rate.

Table 7.12 presents what the LEAPR KS will diagnose as an ABNORMAL traffic pattern if there is a single traffic increase. In the case of two simultaneous traffic increases, the LEAPR KS will diagnose much smaller traffic increases. The three following cases were generated in order to illustrate the previous statement:

- For a line where the traffic is approximately 23 Bytes/second, an increase of 43 Bytes/second will be diagnosed as ABNORMAL (Table 7.12). If there is a traffic increase of 26 Bytes/second on two lines simultaneously, where the traffic

on each line is approximately 23 Bytes/second, the traffic pattern will also be diagnosed as ABNORMAL.

■ For a line where the traffic is approximately 750 Bytes/second, an increase of 1330 Bytes/second will be diagnosed as ABNORMAL (Table 7.12). If there is a traffic increase of 800 Bytes/second on two lines simultaneously, where the traffic on each line is approximately 750 Bytes/second, the traffic pattern will also be diagnosed as ABNORMAL.

■ For a line where the traffic is approximately 916 Bytes/second, an increase of 1600 Bytes/second will be diagnosed as ABNORMAL (Table 7.12). If there is a traffic increase of 1060 Bytes/second on two lines simultaneously, where the traffic on each line is approximately 916 Bytes/second, the traffic pattern will also be diagnosed as ABNORMAL.

The above results illustrated the complexity of what the backpropagation algorithm learned based on the training set.

7.9.4 The LEAPR Repetitiveness Counter (RC)

The repetitiveness counter (RC) on the LEAPR KS was implemented in order to reduce the number of false alarms. By increasing the RC threshold value, the number of false alarms is reduced considerably (Table 7.13 and Figure 7.20). The results of Table 7.13 are illustrated in Figure 7.20. Beside reducing the number of false alarms, increasing the RC will cause the LEAPR KS to miss some traffic pattern abnormalities; these misses will be due to the abnormality either not lasting long enough or not being large enough

to be diagnosed as ABNORMAL. Table 7.14 presents the cumulative number of ABNORMAL traffic pattern cases that were diagnosed several times by the LEAPR KS and their respective maximum RC value encountered. As an example, if the RC threshold value was set to 2.72, the LEAPR KS would detect 84% of all ABNORMAL traffic patterns. The results of Table 7.14 are illustrated in Figure 7.21.

For Table 7.14 and Figure 7.21, only RC values above 1.91 were considered because for maximal RC values less than 1.91, the traffic increases were neither long enough nor large enough. The traffic increases presented in Figure 7.21 are mostly in the area of a RC value of 1.92 and 2.71, mainly because the goal was to identify the minimum detectable traffic increases.

Router monitored	Number of False Alarms/day with different RC					
	0	1.01	1.91	2.72	3.44	4.10
nANNc1	2.64	1.07	0.21	0	0	0
nBDCc1	2.14	0.79	0.36	0.14	0	0
bCARc4	3.29	1.50	0.57	0.07	0	0
nCLDc1	4.07	1.57	0.50	0	0	0
nLAKc2	0.43	0.14	0	0	0	0
nLAUc1	3.71	1.43	0.50	0	0	0
bMERC2	0.86	0.21	0	0	0	0
nMTHc1	5.79	3.07	1.14	0.36	0	0
bRCHc1	4.29	1.86	0.64	0	0	0
nTOKc1	8.14	5.07	2.86	1.07	0.21	0
Average	3.54	1.67	0.68	0.16	0.02	0

Table 7.13: Number of detected false alarms per day based on a given RC threshold value.

Maximum RC Values encountered in the interval of	Routers' names										Overall Percentage (%)
	n	n	b	n	n	n	b	n	b	n	
	A	B	C	C	L	L	M	M	R	T	
	N	D	A	L	A	A	E	T	C	O	
	N	C	R	D	K	U	R	H	H	K	
	c	c	c	c	c	c	c	c	c	c	
	1	1	4	1	2	1	2	1	1	1	
1.91 to 2.71	1	0	1	1	2	1	0	1	0	1	16%
2.72 to 3.43	1	0	2	0	0	0	2	0	1	0	12%
3.44 to 4.09	1	2	0	1	0	1	0	1	0	0	12%
4.10 to 4.68	3	0	1	0	0	0	0	2	1	2	18%
4.69 to 5.21	1	1	0	0	1	0	1	0	0	0	8%
5.22 to 5.69	0	0	0	1	0	1	1	0	1	0	8%
5.70 to 6.12	1	0	2	0	1	0	0	0	0	0	8%
6.13 to 6.51	0	0	0	0	0	0	1	1	0	0	4%
6.52 to 6.86	0	0	0	0	0	0	0	1	0	0	2%
6.87 to 7.17	0	0	1	0	0	0	0	0	0	0	2%
7.18 to 7.46	0	0	0	0	0	0	0	4	0	0	8%
7.47 to 7.71	0	0	0	0	0	0	0	0	0	0	0%
7.72 to 7.91	0	0	0	0	0	0	0	1	0	0	2%

Table 7.14: Number of ABNORMAL traffic patterns detected with their respective maximum RC value encountered.

Figure 7.20: Number of false alarms per day based on a given RC threshold value.

Figure 7.21: Cumulative percentage of ABNORMAL traffic patterns detected given a RC threshold value.

Based on Figure 7.20 and 7.21, depending on the router to be monitored, the RC value may be modified in accordance with Figure 7.20 and 7.21, in order to have less false alarms or more abnormalities detected.

In order to summarize the average efficiency of the LEAPR KS, based on Table 7.13 and 7.14 and on Figure 7.20 and 7.21, the average number of false alarms per day and average accuracy¹² of the LEAPR KS are presented in Table 7.15. Table 7.15 represents the average efficiency based on the results from ten different routers that were monitored by the LEAPR KS. Since the LEAPR is a general proactive diagnostician, for some routers the results were better, and for some others the results were worse (see Table 7.13 and 7.24).

RC threshold value set at	Average number of false alarms/day for 10 different routers during NORMAL operations	Average accuracy during ABNORMAL operations
0.00	3.54	100 %
1.00	1.67	100 %
1.91	0.68	100 %
2.72	0.16	84 %
3.44	0.02	72 %
4.10	0.0	60 %

Table 7.15: LEAPR KS average efficiency.

12: The term accuracy refers to the percentage of ABNORMAL traffic pattern cases well classified.

Chapter 8

Conclusions

8.1 Research Summary

This thesis met its three objectives stated in Section 1.2. Using the backpropagation algorithm as the neural network paradigm, several LEAPR KS prototypes were tested and evaluated in detecting traffic pattern normalities and abnormalities. Based on these KS prototypes, the LEAPR KS was implemented as an autonomous KS that could diagnose any router of the simulated communication network (DRAGON).

With the "TOP TWO and BOTTOM TWO" module, the LEAPR KS could accurately diagnose any router of the DRAGON simulator that has at least four lines. This module makes the LEAPR KS a very general traffic pattern diagnostician. As a general diagnostician, for some routers in the DRAGON simulator, the LEAPR KS detects more false alarms. In order to reduce the number of false alarms, a Repetitiveness Counter (RC) module was implemented, and its threshold value can be adjusted by an operator in order to have fewer false alarms. With the "TOP TWO and BOTTOM TWO"

and the RC modules, the LEAPR KS is a very reliable general traffic pattern diagnostician.

8.2 Summary of Thesis Contributions

There are three major contributions in this thesis:

1. we demonstrated that the neural network technology is able to extract the knowledge in traffic patterns in order to classify them in two categories: NORMAL and ABNORMAL;
2. we demonstrated that the LEAPR KS could diagnose small and complex traffic increases in the simulated communication network. Detecting these traffic increases suggests the LEAPR KS to be a proactive KS in the PEGASUS project; and
3. we demonstrated that the NNet technology can be an efficient tool in diagnosing traffic patterns in a communication network.

8.3 Future Research

The work on the LEAPR KS is far from complete. In addition to the need for further testing on different routers of the simulated network and testing on a real communication network, the following future research work should be considered:

1. The use of parameters other than the traffic pattern should be investigated as input to a neural network in order to classify the traffic pattern as either NORMAL or ABNORMAL. Such parameters could include the status message or the error

rates.

2. The use of other neural network paradigms as a proactive KS in a multiparadigm diagnostic system should also be investigated. A different neural network paradigm to be investigated could be the Counterpropagation algorithm which uses unsupervised learning.
3. The LEAPR classification performance should be compared to other neural network paradigms and to some traditional classification techniques. For example, a comparison between the LEAPR KS performance and the MAXION statistic approach would be very interesting and useful.
4. With its pattern matching characteristic, the neural network approach should be investigated in other areas of the Pegassus project.
5. A final recommendation is the implementation of an adaptive learning system which would be composed of a neural network and an historic table of past cases. As new cases are diagnosed, the historic table of past cases is updated and after a selected period of time the neural network gets automatically retrained in order to adapt itself to the new cases.

References

- [ANSARI91] *"The Performance Evaluation of a New Neural Network Based Traffic Management Scheme for a Satellite Communications Network"*, Nirwan Ansari and Dequan Liu, IEEE Global Telecommunications Conference, GLOBECOM'91, Phoenix, Arizona, December 2-5, 1991, pp 0110 - 0114.
- [BNR90] *"BNR Local Area Network Environment - An Introduction for the Network User"*, BNR Information Technology, July 1990.
- [CAUDILL90] *"Using Neural Nets: Diagnostic Expert Nets Part 5"*, Maureen Caudill, AI Expert, September 1990, pp 43 - 47.
- [CEBULKA89] *"Applications of Artificial Intelligence for Meeting Network Management Challenges in the 1990's"*, Kathleen D. Cebulka, Michael J. Muller, and Christine A. Riley, IEEE Global Telecommunications Conference & Exhibition, 1989, pp 501 - 506.
- [CHEN91] *"A Neural Network Approach Towards Adaptive Congestion Control in Broadband ATM Networks"*, Xiaoqiang Chen and Ian M. Leslie, IEEE Global Telecommunications Conference, GLOBECOM'91, Phoenix, Arizona, December 2-5, 1991, pp 0115 - 0119.
- [COLEMAN91] *"Neural Networks: A High Impact Technology with Applications in Telecommunications"*, Kevin G. Coleman, IEEE Global Telecommunications Conference, GLOBECOM'91, Phoenix, Arizona, December 2-5, 1991, pp 0138 - 0142.
- [CORKILL91] *"Blackboard Systems"*, Daniel Corkill, AI Expert, September 1991, pp 41 - 47.

- [DAWES91] *"Diagnostics in the BNR data communications network"*, N.W. Dawes, System and Computer Engineering, Carleton University, 21 January 1991.
- [DAWES91a] *"PEGASUS Demonstration"*, N.W. Dawes, R. Kaye, B. Pagurek, System and Computer Engineering, Carleton University, October 1991.
- [DAWES92] *"DRAGON : Draft Release 3 SCE Report"*, N.W. Dawes, R. Kaye, B. Pagurek, System and Computer Engineering, Carleton University, 16 January 1992.
- [DAWES92a] *"A New Statistical Measure of Abnormality and the detection of network faults"*, N.W. Dawes, System and Computer Engineering, Carleton University Technical Report SCE-92-13, 11 June 1992.
- [DUSSAULT92] *"A Diagnostic Expert System for Wide Area Network"*, Robert Dussault, Master of Engineering Thesis, System and Computer Engineering, Carleton University, March 1992.
- [FISHER89] *"An Empirical Comparison of ID3 and Back-propagation"*, Douglas H. Fisher and Kathleen B. McKusick, IJCAI-89, pp 788-793.
- [FREEMAN91] *"Neural Networks, Algorithms, Applications and Programming Techniques"*, James A. Freeman and David M. Skapura, Addison-Wesley Publishing Company, 1991.
- [HIRAMATSU90] *"ATM Communications Network Control by Neural Networks"*, Atsushi Hiramatsu, IEEE Transaction on Neural Networks, Vol. 1 No. 1., March 1990.
- [IQNEIBI92] *"A Blackboard Architecture to support Networks Fault Diagnosis"*, Sami M. Iqneibi , Master of Engineering Thesis, System and Computer Engineering, Carleton University, May 1992.
- [JONES87] *"Back-Propagation, A Generalized Delta Learning Rule"*, William P. Jones and Josiah Hoskins, BYTE Magazine, October 1987, pp 155 - 162.

- [KHASNABISH91] *"Congestion Avoidance in Large Supra-High-Speed Packet Networks Using Neural Arbiters"*, Bhumip Khasnabish, Majiid Ahmadi, and Malayappan Shridhar, IEEE Global Telecommunications Conference, GLOBECOM'91, Phoenix, Arizona, December 2-5, 1991, pp 0140 - 0144.
- [KLIMASAUSKAS89] *"Volume 1, Neural Computing"*, C. Klimasauska, J. Guiver and G. Pelton, NeuralWare Inc., 1989.
- [KNIGHT90] *"Connectionist Ideas and Algorithms"*, Kevin Knight, Communication of the ACM, November 1990, Vol 33, No 11.
- [MAXION90] *"Anomaly Detection for Diagnosis"*, Roy A. Maxion, 20th IEEE International Conference on Fault Tolerant Computing, 1990, pp 20-27.
- [MOALLEMI91] *"Classifying Cells for Cancer Diagnosis Using Neural Networks"*, Ciamac Moallemi, IEEE Expert Intelligent Systems and their applications, December 1991, pp 8-12.
- [MOONEY89] *"An Experimental Comparison of Symbolic and Connectionist Learning Algorithms"*, Raymond Mooney, Jude Shavlik, Geoffrey Towell and Alan Gove, IJCAI-89, pp 775-780.
- [NELSON90] *"A Practical Guide to Neural Nets"*, Marilyn McCord Nelson and W. T. Illingworth, Addison-Westley Publishing Company Inc, Massachusetts, 1990.
- [NELSON90.a] *"Neural Network: What they are ?"*, Marilyn Nelson and Bill Illingworth, Adaptive Intelligence Systems, 1990.
- [PAGUREK91] *"A Multiple Paradigm Diagnostic System for Wide Area Communication Networks"*, B. Pagurek, N.W. Dawes, A.R. Kaye, System and Computer Engineering, Carleton University Technical Report SCE-91-37, 1991.
- [SIMPSON89] *"Artificial Neural System, Foundations, Paradigms, Applications, and Implementations"*, Patrick K. Simpson, General Dynamics Electronics Division, San Diego, Pergaman Press, 1990.

[TAN90]

"Connectionist Expert System in Fault Diagnosis of Avionics Line Replaceable Units" , A.H. Tan, Q. Pan, H.C. Lui, H.H. Teh, New Challenges in Aircraft Maint Conf, Singapore, Feb 1990.

APPENDIX 1: Codes for the Backpropagation Algorithm

```

/* ++++++ BKNNETS.C */
/* BACKPROPAGATION NEURAL NETWORKS */
/*
/*
/* This program illustrates the actual mechanics of computing a
/* Backpropagation Neural Network.
/*
/* Designed by : Francois Viens
/* Student # : 187882
/* Initial programming date: 30 August 1991
/* Last modification date: 15 October 1991
/*
/* References:
/*
/* 1. "Artificial Neural Systems", Patrick K. Simpson, Pergamon Press,
/* 1989, page 115.
/*
/* 2. "Connectionist Ideas and Algorithms", Kevin Knight, Communication
/* of the ACM, November 1990, Vol 33, No 11, pp 59-74.
/* ++++++ */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>

#define MAX_INPUTS 12 /* programmer decision */
#define MAX_HIDDEN 60 /* programmer decision */
#define MAX_OUTPUT 5 /* programmer decision */

FILE *stream, *data, *f_weights, *f_nnets, *f_iterations, *f_minmax;

char NNets_name[15], /* defined by user */
      trngfile[15], /* training file */
      testfile[15]; /* testing file */

int INPUTS, /* to be modified by user */
     HIDDEN, /* to be modified by user */
     OUTPUT; /* to be modified by user */

double IN[MAX_INPUTS], /* Inputs */
        OUT[MAX_OUTPUT], /* Desired Output */
        A[MAX_INPUTS], /* number of PEs of the INPUT layer */
        B[MAX_HIDDEN], /* number of PEs in the HIDDEN layer */
        C[MAX_OUTPUT], /* number of PEs in the OUTPUT layer */
        D[MAX_OUTPUT], /* the calculated error from the output PEs */
        E[MAX_HIDDEN]; /* the calculated error from the hidden PEs */

float v[MAX_INPUTS][MAX_HIDDEN], /* weights between the INPUT */
      w[MAX_HIDDEN][MAX_OUTPUT], /* and HIDDEN layer */
      /* weights between the HIDDEN */

```

```

                                /* and OUTPUT layer */
T[MAX_OUTPUT],                /* threshold weight at each OUTPUT PEs */
O[MAX_HIDDEN],                /* threshold weight at each HIDDEN PEs */

/* Difference (delta) between two consecutive weights updated */
float delta_w[MAX_HIDDEN][MAX_OUTPUT],
delta_v[MAX_INPUTS][MAX_HIDDEN],
delta_O[MAX_HIDDEN],
delta_T[MAX_OUTPUT];

float LCOEF,                   /* Learning COEFFicient parameter */
MOMENTUM;                     /* Momentum term (increase learning) */

double sum1, sum2, sum3;      /* the summation */
int i, wait;
unsigned long iterations;

/*****
/* NNets_name.nnd = NNets' architecture file */
/* NNets_name.nnw = NNets' weights file */
/* NNets_name.ite = NNets' iterations file */
/* urgfile.nna = training data file */
/* testfile.nna = testing data file */
/* testfile.nnr = testing result file */
/* NNets_name.nnm = Maximum & Minimum file */
*****/

/*****/

double sigmoid (PE)
float PE;
{
return( 1 / (1 + exp(- PE)));
}

/*****/

double abso (number)
double number;
{
if (number < 0)
number = 0 - number;

return(number);
}

/*****/

void initialize_NNets()
{
/* Initialize the weights and the threshold between -1.0 and 1.0 */

int i, j;

```

```

sum=0;
sum+=rand(3);

for(i = 1; i <= INPUTS; i++)
{
    for(j = 1; j <= HIDDEN; j++)
    {
        v[i][j] = ( (rand() % 500) / 250.0 ) - 1.0;
        delta_v[i][j] = 0.0;
    }
}

for(i = 1; i <= HIDDEN; i++)
{
    for(j = 1; j <= OUTPUT; j++)
    {
        w[i][j] = ( (rand() % 500) / 250.0 ) - 1.0;
        delta_w[i][j] = 0.0;
    }
}

for(i = 1; i <= HIDDEN; i++)
{
    O[i] = ( (rand() % 500) / 250.0 ) - 1.0;
    delta_O[i] = 0.0;
}

for(i = 1; i <= OUTPUT; i++)
{
    T[i] = ( (rand() % 500) / 250.0 ) - 1.0;
    delta_T[i] = 0.0;
}

iterations = 0; /* initialize the iterations counter to zero */
}

/*****
/*          FORWARD PASS          */
/*          =====          */
*****/

void forward_pass()
{
    int i, j, h;
    /* STEP A : Activation between the INPUT and HIDDEN layer. */
    /* ===== Get a new value for the Processing Elements (PEs) */
    /*          in the HIDDEN layer. */

    for(i = 1; i <= HIDDEN; i++)
    {
        sum1 = 0.0;
        for(h = 1; h <= INPUTS; h++)
        {
            sum1 += (A[h] * v[h][i]);

```

```

    }
    B[i] = sigmoid(sum1 + O[i]);
}

/* STEP B : Activation between the HIDDEN and OUTPUT layer. */
/* ===== Get a new value for the Processing Elements (PEs) */
/* in the OUTPUT layer. */

for(j = 1; j <= OUTPUT; j++)
{
    sum2 = 0.0;
    for(i = 1; i <= HIDDEN; i++)
    {
        sum2 += (B[i] * w[i][j]);
    }

    C[j] = sigmoid(sum2 + T[j]);
}

} /* end of the forward_pass() function */

/*===== BACKWARD PASS =====*/
/*===== BACKWARD PASS =====*/
void backward_pass()
{
    int i, j, h;
    float delta1, delta2, delta3, delta4;

/* The LEARNING SCHEDULE: depending on the iteration number, the */
/* MOMENTUM and LCOEF terms change. As the iteration number increases, */
/* the MOMENTUM and LCOEF terms decreases. */

    if ( iterations < 30000 )
    {
        LCOEF = 0.5;
        MOMENTUM = 0.9;
    }
    else if ( iterations < 55000 )
    {
        LCOEF = 0.45;
        MOMENTUM = 0.9;
    }
    else if ( iterations < 75000 )
    {
        LCOEF = 0.40;
        MOMENTUM = 0.9;
    }
    else if ( iterations < 95000 )
    {
        LCOEF = 0.3;
        MOMENTUM = 0.7;
    }
}

```

```

        }
        else
        {
            LCOEF = 0.1;
            MOMENTUM = 0.3;
        }
    }

/* STEP C : Compute the discrepancy (error) between the desired */
/* ===== and the computed OUTPUT value. */

    for(j = 1; j <= OUTPUT; j++)
    {
        D[j] = C[j] * ( 1.0 - C[j] ) * ( OUT[j] - C[j] );
    }

/* STEP D : Calculate the error of each HIDDEN PEs relative to */
/* ===== each discrepancy D[j]. */

    for(i = 1; i <= HIDDEN; i++)
    {
        sum3 = 0.0;
        for(j = 1; j <= OUTPUT; j++)
        {
            sum3 += (w[i][j] * D[j]);
        }
        E[i] = B[i] * ( 1.0 - B[i] ) * sum3;
    }

/* STEP E : Adjust the weights between the HIDDEN and the OUTPUT */
/* ===== layer. */

    for(i = 1; i <= HIDDEN; i++)
    {
        for(j = 1; j <= OUTPUT; j++)
        {
            delta2 = LCOEF * B[i] * D[j] + ( MOMENTUM * delta_w[i][j] );
            w[i][j] += delta2 ;
            delta_w[i][j] = delta2;
        }
    }

/* STEP F : Adjust the OUTPUT layer PEs' threshold. */
/* ===== */

    for(j = 1; j <= OUTPUT; j++)
    {
        delta3 = LCOEF * D[j] + ( MOMENTUM * delta_T[j] );
        T[j] += delta3;
        delta_T[j] = delta3;
    }

/* STEP G : Adjust the weights between the INPUT and the HIDDEN */
/* ===== layer. */

```

```

for(h = 1; h <= INPUTS; h++)
{
    for(i = 1; i <= HIDDEN; i++)
    {
        delta1 = LCOEF * A[h] * E[i] + ( MOMENTUM * delta_v[h][i] );
        v[h][i] += delta1;
        delta_v[h][i] = delta1;
    }
}

/* STEP H : Adjust the HIDDEN layer PES' threshold. */
/* ===== */

for(i = 1; i <= HIDDEN; i++)
{
    delta4 = LCOEF * E[i] + ( MOMENTUM * delta_O[i] );
    O[i] += delta4;
    delta_O[i] = delta4;
}

} /* end of the backward_pass() function */

/*****/

void save_weights()
{ /* write all the weights and thresholds to a file (NNets_name.nnw) */

    char filename[15];
    int i, j, h;

    strcpy(filename, NNets_name);
    strcat(filename, ".nnw");
    f_weights = fopen(filename, "w+");

    for (h = 1; h <= INPUTS; h++)
        for (i = 1; i <= HIDDEN; i++)
            fprintf(f_weights, "%f\n", v[h][i]);

    for (i = 1; i <= HIDDEN; i++)
        for (j = 1; j <= OUTPUT; j++)
            fprintf(f_weights, "%f\n", w[i][j]);

    for (i = 1; i <= HIDDEN; i++)
        fprintf(f_weights, "%f\n", O[i]);

    for (j = 1; j <= OUTPUT; j++)
        fprintf(f_weights, "%f\n", T[j]);

    fclose(f_weights);
}

/*****/

```



```

void get_weights()
{ /* read all the weights and thresholds from a file (NNets_name.nnw). */

char filename[15];
int i, j, h;

strcpy(filename, NNets_name);
strcat(filename, ".nnw");
if ( ( f_weights = fopen(filename, "r+") ) == NULL )
    printf("Cannot read the WEIGHT file. \n");

for (h = 1; h <= INPUTS; h++)
    for (i = 1; i <= HIDDEN; i++)
        fscanf(f_weights, "%f", &v[h][i]);

for (i = 1; i <= HIDDEN; i++)
    for (j = 1; j <= OUTPUT; j++)
        fscanf(f_weights, "%f", &w[i][j]);

for (i = 1; i <= HIDDEN; i++)
    fscanf(f_weights, "%f", &O[i]);

for (j = 1; j <= OUTPUT; j++)
    fscanf(f_weights, "%f", &T[j]);

fclose(f_weights);
}

/*****/

void input_scale()
{ /* bring all INPUT values between 0 and 1. */
int i, j;
double MAX[MAX_INPUTS], MIN[MAX_INPUTS];
char fname[15];

strcpy(fname, NNets_name);
strcat(fname, ".nnm");
if ( ( f_minmax = fopen(fname, "r+") ) == NULL )
    printf("Cannot read the MINIMUM & MAXIMUM file. \n");

for (j = 1; j <= INPUTS; j++)
{
    fscanf(f_minmax, "%lf", &MAX[j]);
    fscanf(f_minmax, "%lf", &MIN[j]);
}

fclose(f_minmax);

for(i = 1; i <= INPUTS; i++)
    A[i] = ( IN[i] - MIN[i] ) / ( MAX[i] - MIN[i] );
}

```

```

                                /*****/

void menu()
{
    printf("\n\n");
    printf("      BACKPROPAGATION NEURAL NETWORKS\n\n");
    printf("\t [ 1 ]..... CREATE  \n\n");
    printf("\t [ 2 ]..... TRAIN  with convergence threshold \n\n");
    printf("\t [ 3 ]..... TRAIN  with a training cycle limit \n\n");
    printf("\t [ 4 ]..... TEST   \n\n");
    printf("\t [ 5 ]..... QUIT   \n\n");
}

                                /*****/
                                /*      MAIN PROGRAM      */
                                /*      =====      */
                                /*****/

void main()
{
    char    filename[15], filename1[15], filename2[15];
    int     answer, j;
    float   threshold;
    unsigned long  cycles, trg_time;
    long int  resultp, clock_time, clock_time1, clock_time2;

    menu();
    printf("What do you want to do ?  1 or 2 or 3 or 4 or 5  ");
    scanf("%d", &answer);
    printf("\n\n");

    while (answer != 5)
    {
        if (answer == 1)
        {
            /* With this answer, the user selects a name for the NNet and selects the          */
            /* number of processing elements in the INPUT, HIDDEN and OUTPUT layers.          */
            /* It also randomly initialize the weights between layers.                      */

            printf("Name of the Neural networks is : ");
            scanf("%s", NNets_name);
            printf("\n\n");

            strcpy(filename, NNets_name);
            strcat(filename, ".nnd");
            f_nnets = fopen(filename, "w+");

            printf("\n  # of INPUT PEs - maximum is 12 - : ");
            scanf("%d", &INPUTS);
            printf("\n  # of HIDDEN PEs - maximum is 60 - : ");
            scanf("%d", &HIDDEN);
            printf("\n  # of OUTPUT PEs - maximum is 5 - : ");
            scanf("%d", &OUTPUT);

            fprintf(f_nnets, "%d \n", INPUTS);

```

```

fprintf(f_nnets, "%d \n", HIDDEN);
fprintf(f_nnets, "%d \n", OUTPUT);

initialize_NNets();
save_weights();
fclose(f_nnets);
}
else
if (answer == 2)
{
/* With this answer, the user enters the name of his NNet, the name of      */
/* his training file and his CONVERGENCE threshold which decides when      */
/* to stop the training. By using this sub-menu, it is assumed that        */
/* the NNet was created previously. It is also assumed that the user      */
/* has a ".nrm" file before using this sub-menu.                          */
/*                                                                           */

printf("\n\n\n");
printf("\t BACKPROPAGATION NEURAL NETWORKS -- TRAINING\n");
printf("\t with CONVERGENCE threshold\n");
printf("\t ----- \n\n\n");
printf("Name of the Neural networks is : ");
scanf("%s", NNets_name);
printf("\n\n");

strcpy(filename, NNets_name);
strcat(filename, ".nnd");
if ( ( f_nnets = fopen(filename, "r+") ) == NULL)
printf("Cannot read the NNets' architecture file \n");
else
{
fscanf(f_nnets, "%d", &INPUTS);
fscanf(f_nnets, "%d", &HIDDEN);
fscanf(f_nnets, "%d", &OUTPUT);
fclose(f_nnets);
}
printf("Name of the training file : ");
scanf("%s", trngfile);
printf("\n\n");
printf("What is the CONVERGENCE threshold you want to use ? ");
scanf("%f", &threshold);
printf("\n\n");
strcpy(filename1, trngfile);
strcat(filename1, ".nna");
if ( ( stream = fopen(filename1, "r+") ) == NULL)
printf("Cannot read the TRAINING file \n");
printf("\n\n\n");

trg_time = 0;
get_weight();

strcpy(filename, NNets_name);
strcat(filename, ".ite");
f_ite = fopen(filename, "a+");

```

```

time(&resultp);          /* Time setup */
fprintf(f_iterations, "%s\n", ctime(&resultp));
clock_time1 = clock();
fprintf(f_iterations, "Starting CPU time : %d in micro sec\n", clock_time1);

do
{
    iterations += 1;
    trg_time += 1;

    if( feof(stream) )
    {
        fclose(stream);
        stream = fopen(filename1, "r");
    }
    for (j = 1; j <= INPUTS; j++)
        fscanf(stream, "%lf ", &IN[j]);

    for (i = 1; i <= OUTPUT; i++)
        fscanf(stream, "%lf ", &OUT[i]);

    input_scale();
    forward_pass();
    backward_pass();

    if( (trg_time % 100) == 0)
        fprintf(stderr, "---> Number of cycles done so far : %ld \r", trg_time);
} while (abso (C[1] - OUT[1]) >= threshold);

save_weights();
fclose(stream);

time(&resultp);          /* Time Calculations */
fprintf(f_iterations, "%s\n", ctime(&resultp));
clock_time2 = clock();
fprintf(f_iterations, "Ending CPU time : %d in micro sec\n", clock_time2);

clock_time = clock_time2 - clock_time1;
fprintf(f_iterations, "CPU time in seconds is: %lf\n", clock_time/1000000.0);
fprintf(f_iterations, "\n\n ++++++\n\n");
fprintf(f_iterations, "Number of cycles executed was :\n");
fprintf(f_iterations, "-----> %ld \n\n", iterations);
fclose(f_iterations);

)
else
if (answer == 3)
{
    /* With this answer, the user enters the name of his NNet, the name of
    /* his training file and the number of iterations he wants the
    /* backpropagation to go through. By using this sub-menu, it is
    /* assumed that the NNet was created previously. It is also assumed
    /* that the user has a ".nnm" file before using this sub-menu.
    */
}

```

```

printf("\n\n\n");
printf("\t BACKPROPAGATION NEURAL NETWORKS -- TRAINING\n");
printf("\t with training cycles limit\n");
printf("\t ----- \n\n\n");
printf("Name of the Neural networks is : ");
scanf("%s", NNets_name);
printf("\n\n");

strcpy(filename, NNets_name);
strcat(filename, ".nnd");
if ( ( f_nnets = fopen(filename, "r+") ) == NULL)
    printf("Cannot read the NNets' architecture file\n");
else
{
    fscanf(f_nnets, "%d", &INPUTS);
    fscanf(f_nnets, "%d", &HIDDEN);
    fscanf(f_nnets, "%d", &OUTPUT);
    fclose(f_nnets);
}
printf("Name of the training file : ");
scanf("%s", trngfile);
printf("\n\n");
strcpy(filename1, trngfile);
strcat(filename1, ".nna");
if ( ( stream = fopen(filename1, "r+") ) == NULL)
    printf("Cannot read the TRAINING file\n");
printf("How many cycles ? ");
scanf("%d", &cycles);
printf("\n\n\n");

trg_time = 0;
get_weights();

strcpy(filename, NNets_name);
strcat(filename, ".ite");
f_iterations = fopen(filename, "a+");

time(&resultp); /* Time setup */
fprintf(f_iterations, "%s\n", ctime(&resultp));
clock_time1 = clock();
fprintf(f_iterations, "Starting CPU time : %d in micro sec\n", clock_time1);

do
{
    iterations += 1;
    trg_time += 1;

    if( feof(stream) )
    {
        fclose(stream);
        stream = fopen(filename1, "r");
    }
}

```

```

for (j = 1; j <= INPUTS; j++)
    fscanf(stream, "%lf ", &IN[j]);

for (i = 1; i <= OUTPUT; i++)
    fscanf(stream, "%lf ", &OUT[i]);

input_scale();
forward_pass();
backward_pass();

if( (trg_time % 100) == 0)
    fprintf(stderr, "--> Number of cycles done so far : %ld \r", trg_time);
} while ( trg_time < cycles);

save_weights();
fclose(stream);

time(&resultp);          /* Time calculations */
fprintf(f_iterations, "%s\n", ctime(&resultp));
clock_time2 = clock();
fprintf(f_iterations, "Ending CPU time : %d in micro sec\n", clock_time2);

clock_time = clock_time2 - clock_time1;
fprintf(f_iterations, "CPU time in seconds is:%lf\n", clock_time/1000000.0);
fprintf(f_iterations, "\n\n ++++++\n\n");

fprintf(f_iterations, "Number of cycles executed was :\n");
fprintf(f_iterations, "-----> %ld \n\n", iterations);
fclose(f_iterations);

}
else
    if (answer == 4)
    {
        /* With this answer, the user enters the name of his NNet and the          */
        /* name of his testing file. This sub-menu goes through the                */
        /* testing file and outputs the NNet's input, the trained desired          */
        /* output and the actual NNet's output.                                    */

        printf("\n\n\n");
        printf("\n BACKPROPAGATION NEURAL NETWORKS -- TESTING\n");
        printf("\n ----- \n\n\n");
        printf("Name of the Neural networks is : ");
        scanf("%s", NNets_name);
        printf("\n\n");

        strcpy(filename, NNets_name);
        strcat(filename, ".nnd");
        if ( ( f_nnets = fopen(filename, "r") ) == NULL)
            printf("Cannot read the NNets' architecture file \n");
        else
        {
            fscanf(f_nnets, "%d", &INPUTS);

```

```

    fscanf(f_nnets, "%d", &HIDDEN);
    fscanf(f_nnets, "%d", &OUTPUT);
    fclose(f_nnets);
}

printf("Name of the testing file : ");
scanf("%s", testfile);
printf("\n\n");
strcpy(filename1, testfile);
strcat(filename1, ".nna");
if ( ( stream = fopen(filename1, "r+") ) == NULL)
    printf("Cannot read the TESTING file ");

strcpy(filename2, testfile);
strcat(filename2, ".nnr");
data = fopen(filename2, "a+");

get_weights();

while ( !feof(stream) )
{
    for (j = 1; j <= INPUTS; j++)
    {
        fscanf(stream, "%lf ", &IN[j]);
        fprintf(data, "%lf ", IN[j]);
    }

    for (i = 1; i <= OUTPUT; i++)
    {
        fscanf(stream, "%lf ", &OUT[i]);
        fprintf(data, "%lf ", OUT[i]);
    }

    input_scale();
    forward_pass();

    for(j = 1; j <= OUTPUT; j++)
        fprintf(data, " %lf", C[j] );

    fprintf(data, "\n");
}
fprintf(data, "\n\n");
fclose(stream);
fclose(data);
}
else
{
    printf(" INVALID choice \n\n");
    sleep(2);
}

menu();

```

```
printf("What do you want to do? 1 or 2 or 3 or 4 or 5 ");
scanf("%d", &answer);
printf("\n\n");
}
    printf("BYE.... the BACKPROPAGATION NNets is terminated\n");
    sleep(2);
}
```

APPENDIX 2: Codes for the LEAPR KS

```

/*****
        filename: LEAPR.c
        version : FINAL

Programmers' Name .      .... Francois Viens
                        ..... Sami Iqneibi
Date initial            ..... 19 August 1991  (NNets.c)
Date modified.          ..... 31 May 1992

Program Description :

    This is a pro-active Knowledge Source (KS) in the PEGASUS project.

    This program posts a request to the BlackBoard and reads the reply from
    the BlackBoard. Through these requests/replies, this KS is getting
    the traffic pattern (bytes/seconds) .

    This KS is taking the mean of the last 30 minutes and sends the traffic
    ratio (traffic divided by the mean) to the TOP TWO BOTTOM TWO algorithm.

    The "TOP TWO & BOTTOM TWO" algorithm, in front the Neural Network
    applications, takes the two biggest and two lowest traffic pattern ratio
    and forward them to the Neural Network for diagnosis. If there is less
    than four lines, the algorithm will inserted default values (0.0) in
    order to complete the 4 input vector.

    Based on the traffic pattern ratio, the Neural Network diagnoses the traffic
    pattern with an output between 0.0 and 1.0. Depending on the REPETITIVENESS
    COUNTER, the traffic pattern is classified as either NORMAL or ABNORMAL.
    When there is an ABNORMAL traffic pattern, an hypothesis is forwarded to the
    blackboard.

    ****
    ** ATTENTION: The user has to modified the program where there is      **
    **          USER FLAG, in order to diagnose the proper router and      **
    **          the proper lines.                                           **
    ****

*****/

/* To compile, enter:
    cc -c LEAPR.c
    cc LEAPR.o myfunctions.o -o LEAPR -lm
*/

#include <sys/types.h>
#include <stdio.h>
#include <math.h>
#include "mytypes.h"
#include "myfunctions.h"

#define KS_NAME          "LEAPR"
#define NNets_name       "case42"
#define INPUTS           4      /* to be modified depending of the NNets */

```

```

#define HIDDEN          12  /* to be modified depending of the NNets */
#define OUTPUT          1  /* to be modified depending of the NNets */
#define MAX_INPUTS      25  /* programmer decision */
#define MAX_HIDDEN      40  /* programmer decision */
#define MAX_OUTPUT      5   /* programmer decision */

```

```
/* ***** GLOBAL VARIABLES ***** */
```

```

long   last_reply, hypo_id;
struct process_record process_env ;
struct request_record *request_array;
struct reply_record   *reply_array;
struct hypothesis_record *hypothesis_array;

```

```
FILE *f_weights, *f_minmax;
```

```
float IN[MAX_INPUTS]; /* Vector of the Inputs + desired output */
```

```

float A[MAX_INPUTS]; /* number of PEs of the INPUT layer */
/* and A[INPUTS + 1] is the desired output */

```

```

double B[MAX_HIDDEN], /* number of PEs in the HIDDEN layer */
       C[MAX_OUTPUT]; /* number of PEs in the OUTPUT layer */

```

```

float v[MAX_INPUTS][MAX_HIDDEN], /* weights between the INPUT */
/* and HIDDEN layer */
     w[MAX_HIDDEN][MAX_OUTPUT], /* weights between the HIDDEN */
/* and OUTPUT layer */
     T[MAX_OUTPUT], /* threshold weight at each OUTPUT PEs */
     O[MAX_HIDDEN]; /* threshold weight at each HIDDEN PEs */

```

```
double sum1, sum2; /* the summation */
```

```
/******
```

```

/******
/*      BLACKBOARD FUNCTIONS      */
/*      =====                  */
/******

```

```
long post_request_time() /* Ask for the TIME */
```

```

{
    long request_id;

    request_id = get_empty_request(&process_env);
    strcpy(request_array[request_id].request_str0, "TIME");
    request_array[request_id].request_para1 = 0;
    request_array[request_id].function_no = 3;
    request_array[request_id].request_para2 = 24;
    post_request(&process_env, request_id);
    return(request_id);
}

```

```

/*****/

unsigned long read_reply_time(request_id)    /* Get the TIME */
long request_id;
{
    long index;

    for (;;) {
/* get index to new reply */

        index = get_next_reply(&process_env,last_reply);

        switch (index) {
            case 0:
                wait_change(&process_env); /* wait for a change */
                break;
            default:
                last_reply = index;
                if (reply_array[last_reply].reply_no == request_id)
                    return(reply_array[last_reply].reply_para0);
        }
    }
}

/*****/

/* By specifying the NODE and the PORT, ask the traffic */
long post_request_traffic(NODE,port1)

char NODE[9]; /* Maximum 9 cahracters */
short port1;
{
    long request_id;

    request_id = get_empty_request(&process_env);
    strcpy(request_array[request_id].request_str0,NODE);
    request_array[request_id].request_para0 = 0;
    request_array[request_id].request_para1 = port1;
    request_array[request_id].function_no = 3;
    request_array[request_id].request_para2 = 4;
    post_request(&process_env,request_id);
    return(request_id);
}

/*****/

/* Get the traffic for that specific NODE and PORT */
unsigned long read_reply_traffic(request_id)
long request_id;
{
    long index;

```

```

for (;;) {
/* get index to new event */

index = get_next_reply(&process_env,last_reply);

switch (index) {
case 0:
wait_change(&process_env); /* wait for a change */
break;
default:
last_reply = index;
if (reply_array[last_reply].reply_no == request_id)
return(reply_array[last_reply].reply_para0);
}
}
}

/******
/*          NEURAL NETWORKS FUNCTIONS          */
/*          =====                          */
/******

void get_weights()
{
/* Read all the weight and threshold values from a file (case42.nnw). */

char filename[15];
int i, j, h;

strcpy(filename, NNets_name);
strcat(filename, ".nnw");
f_weights = fopen(filename, "r");

/* The weights between the INPUT and HIDDEN layers */
for (h = 1; h <= INPUTS; h++)
for (i = 1; i <= HIDDEN; i++)
fscanf(f_weights, "%f", &v[h][i]);

/* The weights between the HIDDEN and OUTPUT layers */
for (i = 1; i <= HIDDEN; i++)
for (j = 1; j <= OUTPUT; j++)
fscanf(f_weights, "%f", &w[i][j]);

/* The Threshold weights at each HIDDEN PEs */
for (i = 1; i <= HIDDEN; i++)
fscanf(f_weights, "%f", &O[i]);

/* The Threshold weights at each OUTPUT PEs */
for (j = 1; j <= OUTPUT; j++)
fscanf(f_weights, "%f", &T[j]);

fclose(f_weights);

```

```

}

/*****/

void input_scale()
{
/* Pre-processing of the INPUTS in order to bring them between the */
/* value of 0 and 1. This function needs the minimum & maximum */
/* values of every input. These values are in the file called case42.nnw. */

int i, j;
double MAX[MAX_INPUTS], MIN[MAX_INPUTS];
char fname[15];

strcpy(fname, NNets_name);
strcat(fname, ".nnm");
if ( ( f_minmax = fopen(fname, "r+") ) == NULL )
    printf("Cannot read the MINIMUM & MAXIMUM file. \n");

for (j = 1; j <= INPUTS; j++)
{
    fscanf(f_minmax, "%lf", &MAX[j]);
    fscanf(f_minmax, "%lf", &MIN[j]);
}

fclose(f_minmax);

/* Linearly bring the values between 0.0 and 1.0 */
for (i = 1; i <= INPUTS; i++)
    A[i] = ( IN[i] - MIN[i] ) / ( MAX[i] - MIN[i] );

}

/*****/

double sigmoid (PE)
float PE;
{
/* This is the standard SIGMOID function. */
return( 1 / (1 + exp(- PE)));
}

/*****/

void forward_pass()
{
int i, j, h;
/* STEP A : Activation between the INPUT and HIDDEN layer. */
/* ===== Get a new value for the Processing Elements (PEs) */
/* in the HIDDEN layer. */

for(i = 1; i <= HIDDEN; i++)
{
    sum1 = 0.0;

```

```

        for(h = 1; h <= INPUTS; h++)
        {
            sum1 += (A[h] * v[h][i]);
        }
        B[i] = sigmoid(sum1 + O[i]);
    }

/* STEP B      : Activation between the HIDDEN and OUTPUT layer.          */
/* =====    Get a new value for the Processing Elements (PEs)          */
/*              in the OUTPUT layer.                                       */

    for(j = 1; j <= OUTPUT; j++)
    {
        sum2 = 0.0;
        for(i = 1; i <= HIDDEN; i++)
        {
            sum2 += (B[i] * w[i][j]);
        }

        C[j] = sigmoid(sum2 + T[j]);
    }

} /* end of function forward_pass() */

```

```

/*****
/*              MAIN PROGRAM              */
/*              =====                  */
*****/

```

```

main()
{
    short  s, ss, n, t, j, i, k, LINES;
    float  s05, mean[25][31], RC;
    long   request_id[25], request_id_time, ii, simul_time;
    float  value[25], old[25], case_mean[25];
    long   time1, time2, starting_time;
    float  line[25], link[25];
    float  big1, big2, small1, small2;

/* Get access to the blackboard */

    if (get_access_to_bb(&process_env) == -1) {
        fprintf(stderr, "error detected while getting access to bb\n");
        exit(1);
    }

/* register */

    strcpy(process_env.KS_name, KS_NAME);

```

```

process_env.KS_type = 0;
process_env.KS_subtype = 0;

if (register_KS(&process_env) == -1)
    exit(1);

request_array = (struct request_record *)
                process_env.area[REQUEST_ARRAY].array_address;

reply_array = (struct reply_record *)
               process_env.area[REPLY_ARRAY].array_address;

hypothesis_array = (struct hypothesis_record *)
                    process_env.area[HYPOTHESIS_ARRAY].array_address;

/* Here, we express our interest in the replys_array. In effect, our KSid
   will be added to the list of KSs that should be notified when a new reply
   record is posted to the BB. if the call succeeds, an index where the new
   changes will be placed is returned.
*/

last_reply = interested(&process_env, REPLY_ARRAY);

/***** USER INTERFACE *****/

/* Duration of this program (in minutes) for monitoring purposes */

fprintf(stderr, "What is the period of TIME (number of minutes), \n");
fprintf(stderr, "you want to run this Knowledge Source for ? ");
scanf("%ld", &simul_time);
fprintf(stderr, "\n ***** The period of time is %ld minutes\n", simul_time);

/* Repetiveness Counter Threshold value */

fprintf(stderr, "What is the RC threshold value ? \n");
scanf("%f", &RC);
fprintf(stderr, "\n ***** The RC threshold value is set to %f\n", RC);

/***** End of the USER INTERFACE *****/

for (i = 0; i <= 24; i++) /* initialization purpose only */
{
    for (s = 0; s <= 30; s++)
        mean[i][s] = 0;
}

/* ***** FIRST THIRTY MINUTES OF SIMULATION ***** */

/* SYNCHRONIZATION WITH THE SIMULATOR USING POLLING */

```



```

/* ==> Reference Point for TIME <===== */

do {
    request_id_time = post_request_time();
    time1 = read_reply_time(request_id_time);
    } while ( (time1 % 60) > 2 );

/* Ask the traffic to the blackboard */
request_id[1] = post_request_traffic("nlauc1",1);
request_id[2] = post_request_traffic("nlauc1",4);
request_id[3] = post_request_traffic("nlauc1",5);
request_id[4] = post_request_traffic("nlauc1",7);
request_id[5] = post_request_traffic("nlauc1",8);

/* USER */

LINES = 5; /* Number of monitored lines */

/* USER */

/* Get the traffic from the Blackboard */
for (t = 1; t <= LINES; t++)
    old[t] = read_reply_traffic(request_id[t]);

starting_time = time1;
s = 1; /* initialize the "s" counter for the MEAN variable */

/* +++++> Based on the reference TIME point and TRAFFIC, */
/* get the first 30 minutes of traffic in order to establish the mean. */

while( (time1 - starting_time) < 1800) /* 30 minutes */
{

/* SYNCHRONIZATION WITH THE SIMULATOR USING POLLING */
/* The traffic is taken for diagnose every minute */

do {
    request_id_time = post_request_time();
    time2 = read_reply_time(request_id_time);
    usleep(50000); /* sleep for 0.05 seconds ( 50 000 micro seconds ) */
    } while ( (time2 - time1) < 60 );

/* Ask for the TRAFFIC */
request_id[1] = post_request_traffic("nlauc1",1);
request_id[2] = post_request_traffic("nlauc1",4);
request_id[3] = post_request_traffic("nlauc1",5);
request_id[4] = post_request_traffic("nlauc1",7);
request_id[5] = post_request_traffic("nlauc1",8);

/* USER */

```

```

for (t = 1; t <= LINES; t++)          /* Get the TRAFFIC */
    value[t] = read_reply_traffic(request_id[t]);

/* Calculation of the line's traffic rate */
for (j = 1; j <= LINES; j++)
{
    mean[j][s] = value[j] - old[j];
    fprintf(stderr, "%6.2f ", mean[j][s]);
}
fprintf(stderr, "\n");

for (ss = 1; ss <= 30; ss++)          /* Printing purpose only */
{
    for (j = 1; j <= LINES; j++)
        fprintf(stderr, "%6.2f ", mean[j][ss]);
}
fprintf(stderr, "\n");

/* Values transfered for next loop pass */
for (t = 1; t <= LINES; t++)
    old[t] = value[t];

s += 1;                               /* increase the "s" counter for the MEAN variable */
time1 = time2;                         /* Transfer of the time reference point */
fprintf(stderr, "\n\n s = %d\n", s);

} /* End of the WHILE LOOP */

/* Get the Neural Network memory ( i.e. the Neural Network weights ) */

get_weights();

/* ===== LOOP FOR DAYS OF OPERATIONS ===== */

s05 = 0;                               /* Initialization */

for(ii = 1; ii <= simul_time; ii++)
{
    time1 = time2;                     /* Reference point for the time */

    /* Poll until the minute interval is completed */
    do {
        request_id_time = post_request_time();

```

```

    time2 = read_reply_time(request_id_time);
} while ( (time2 - time1) < 60);

/* After one minute interval, ask the TRAFFIC */

request_id[1] = post_request_traffic("nlauc1",1);
request_id[2] = post_request_traffic("nlauc1",4);
request_id[3] = post_request_traffic("nlauc1",5);
request_id[4] = post_request_traffic("nlauc1",7);
request_id[5] = post_request_traffic("nlauc1",8);

/* Get the actual line's traffic rate */
for (t = 1; t <= LINES; t++)
    value[t] = read_reply_traffic(request_id[t]);

if ( ( (time2 - time1) < 59) || ( (time2 - time1) > 62) ) /* just in case */
{
    /* If it is not enough close to one minute interval, average the TRAFFIC */
    /* for a minute interval. */

    for (i = 1; i <= LINES; i++)
        line[i] = (value[i] - old[i]) * (60.0 / (time2 - time1));
}
else
{
    for (i = 1; i <= LINES; i++)
        line[i] = value[i] - old[i];
}

fprintf(stderr, "TIME is %d ... \n", time2 );

for (i = 1; i <= LINES; i++)
{
    case_mean[i] = 0; /* initialization to zero */

    /* Total of the line's traffic rate for the last 30 minutes */
    for (ss = 1; ss <= 30; ss++)
        case_mean[i] += mean[i][ss];

    case_mean[i] = case_mean[i] / 30.0; /* Calculation of the mean */

    /* Calculation of the ratio between the actual line's traffic rate */
    /* and the line's traffic mean for the last 30 minutes. This ratio */
    /* is in the variable link[i] . */

    link[i] = line[i] / case_mean[i];
}

/*****/

```

```

/***** TOP TWO & BOTTOM TWO ALGORITHM *****/
/*****
/* Out of all the lines monitored, this algorithm will put the biggest traffic
/* ratio value in "big1", the second biggest traffic ratio value in "big2",
/* the smallest traffic ratio value in "small1" and the second smallest
/* traffic ratio value in "small2 "
*****/

big1 = 0;    big2 = 0;          /* Initialization */
small1 = 20;  small2 = 20;      /* Initialization */

for (i = 1; i <= LINES; i++)    /* LINES is the number of lines */
{
    if (link[i] > big1)
    {
        big2 = big1;
        big1 = link[i];
    }
    else
    {
        if (link[i] > big2)
            big2 = link[i];
    }

    if (link[i] < small1)
    {
        small2 = small1;
        small1 = link[i];
    }
    else
    {
        if (link[i] < small2)
            small2 = link[i];
    }
}

} /* end FOR loop */

/*****
/* JUST IN CASE OF this Knowledge Source is diagnosing less than four
/* lines, this program will insert DEFAULT values (0.0) in the
/* TOP TWO & BOTTOM TWO algorithm.
*****/

if (LINES < 4)
{
    if (LINES == 3)
    {
        small2 = small1;
        small1 = 0;
    }

    if (LINES == 2)

```

```

    {
        small2 = 0;
        small1 = 0;
    }

    if (LINES == 1)
    {
        big2 = 0;
        small2 = 0;
        small1 = 0;
    }

    if (LINES == 0)
    {
        big1 = 0;
        big2 = 0;
        small2 = 0;
        small1 = 0;
    }
}

/***** End of the "JUST IN CASE" pass *****/

/*****/
/*      Forward the result of the TOP TWO & BOTTOM TWO algorithm to the      */
/*      Neural network applications.                                          */
/*****/

IN[1] = big1; IN[2] = big2;
IN[3] = small2; IN[4] = small1;

/***** End of the TOP TWO & BOTTOM TWO ALGORITHM *****/

/* ++++++ Neural Network Diagnostic Application ++++++ */
/* ++++++ Based on the normalized Neural Network, this Knowledge Source (KS) */
/* diagnose the traffic pattern.                                              */

input_scale();          /* get its MIN & MAX values */
forward_pass();         /* forward the data through the NNets */

/* Based on the REPETITIVENESS Counter, this Knowledge Source (KS) determines */
/* if the traffic pattern is either NORMAL or ABNORMAL. If the traffic pattern */
/* is ABNORMAL, the LEAPR KS sends an hypothesis to the blackboard.          */

if (C[1] >= 0.5 )
{

```

```

s05 = (s05 * 0.9) + 1.0;
    if (s05 > RC)
    {
        fprintf(stderr, "====> ABNORMAL %f\n", s05);

        hypo_id = get_empty_hypothesis(&process_env);
        strcpy(hypothesis_array[hypo_id].device_name, "nlauc1");
        strcpy(hypothesis_array[hypo_id].KS_name, KS_NAME);
        hypothesis_array[hypo_id].hypothesized_state = 1;
        hypothesis_array[hypo_id].confidence = 99; /* 99 % */
        post_hypothesis(&process_env, hypo_id);
    }

}
else
    s05 = s05 * 0.9;

/* ++++++ */
/* ++++++ END of the Neural Network Diagnostic Application ++++++ */
/* ++++++ */

for (t = 1; t <= LINES; t++)
{
    /* Delete the oldest line's traffic value and move the 29 other */
    /* line's traffic value of one in the mean array . */
    for (n = 2; n <= 30; n++)
        mean[t][n - 1] = mean[t][n];

    /* Get the traffic value of each lines for the */
    /* last traffic pattern received . */
    mean[t][30] = line[t];
}

/* Values transfered for next loop pass */
for (t = 1; t <= LINES; t++)
    old[t] = value[t];

} /* End of the DAYS OF OPERATIONS' loop */

/* ***** */
/* CLOSING ***** */
/* ***** */

uninterested(&process_env, REPLY_ARRAY);
check_out(&process_env);
}

```

END

24 | 05 | 93

FIN