A CONTEXT BASED DATA SANITY CHECKING ALGORITHM AND ITS IMPLEMENTATION

by

Saher Lahouar

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

APPROVED:

Saifur Rahman, Chairman

Robert P. Broadwater

Jaime De La Ree

William G. Sullivan

Hugh F. VanLandingham

December 1991

Blacksburg, Virginia

A CONTEXT BASED DATA SANITY CHECKING ALGORITHM

AND ITS IMPLEMENTATION

by

Saher Lahouar

Saifur Rahman, Chairman

ELECTRICAL ENGINEERING

(ABSTRACT)

In this dissertation, we present a cost-effective, neural network-based technique for data sanity checking and small system parameter monitoring which utilizes the contextual information in which data is collected to avoid the need for multiple metering. Multiple metering is not always a feasible nor an optimal solution to the problem. In an environment where it is necessary to monitor a large number of different physical variables, the mere installation and maintenance of multiple metering equipment can prove to be very costly. Moreover, multiple measurements of the same quantity result in a phenomenon known as *data explosion.* Context-based sensor validation is achieved through cross sensor redundancy, which is not to be confused with metering redundancy. Neural networks are used to model the relationships among the various parameters and to provide context-based estimates which help in identifying sensor (versus system) malfunction. Slow tracking of the relationships among the parameters as they change over time is made possible through on-line training of the neural networks on the most recent data. This helps to account for the dependency of the relationships among system parameters on the range of external variables such as ambient temperature.

A prototypical system titled DASANEX is implemented to illustrate the validity of the technique. The system is used to monitor and filter real-time transformer and ambient temperature data. A proof-of-concept is established using field data from the city of Martinsville Electric Department. Results prove the superior ability of the technique to identify sensor malfunction and to provide real-time adequate replacement values during short downtimes of the sensors even when some sensor data are missing or contaminated.

Acknowledgments

I would like to express my deepest gratitude to my advisor, mentor, and personal friend, Dr. Saifur Rahman, for standing by my side during the ups and downs of this journey. His constant guidance and advice have been primordial elements in the success of this endeavor. Working under his guidance has been both exciting and challenging. He has the rare ability of maintaining perfect balance while dealing with people both on a professional and a social level.

I would also like to thank the members of my committee, Drs. Robert Broadwater, Jaime De La Ree, Hugh VanLandingham and William Sullivan for their time and effort.

The sponsorship of the Center for Innovative Technology and the American Public Power Association for parts of this research is gratefully acknowledged. Also, the timely support and data provided by Mr. Corekin have been invaluable for the completion of this work.

Special thanks go to all my friends in the Energy Systems Laboratory and outside, who have provided me with much help and support. I am particularly thankful to my good friends Dr. Govinda Shrestha and family, who have been my home away from home.

Last, but not least, I would like to extend my deepest thanks and appreciation to my family for their unconditional love and support, especially to my brother Souhail and his family. I cannot begin to express my gratitude to my mother and father whose only recourse during the past few long years has been patience.

Table of Contents

INTRODUCTION 1

LITERATURE REVIEW 6

2.1 Introduction 6

2.2 Data Sanity Checking in the Literature 6

2.2.1 Importance of numerical data 7

2.2.2 Outliers in statistical data 9

2.2.3 Error detection and correction 10

2.2.4 Other data checking research 10

2.3 Recent Expert Systems In Power Engineering 12

2.4 Neural Networks In Power Engineering 17

2.4.1 Alarm Processing 18

2.4.2 Analysis and Speculation 18

2.4.3 Contingency Analysis 20

2.4.4 Economic Dispatch 21

2.4.5 Fault Diagnosis 21

2.4.6 Load Forecasting 22
2.4.7 Machines 24
2.4.8 Modeling 25
2.4.9 Reactive Power and Voltage Control 26
2.4.10 Security Assessment 27
2.4.11 Switching Operation 28
2.4.12 Topological Observability 28
2.4.13 Transient Stability Problems 30
2.5 Summary 30

PROBLEM DEFINITION 33
3.1 Introduction 33
3.2 Problem Formulation 36
3.3 Data Abnormalities 37
3.3.1 Physical system setup 37
3.3.2 Description of the data collected 37
3.4 Recapitulation 44

THEORETICAL FOUNDATION 46
4.1 Introduction 46
4.2 Expert Systems 46
4.2.1 Definition 47
4.2.2 Components of an Expert System 47
4.2.3 Types of Expert Systems 49
4.2.4 Advantages & Shortcomings 51
4.3 Neural Networks 53
4.3.1 Historical Overview 53
4.3.2 Definition and Description 54
4.3.3 Advantages 59
4.3.4 Shortcomings 62

4.4 Hybrid Systems....62
4.5 The 'Context' Concept....65
4.5.1 Context-based data sanity checking....65
4.5.2 Context representation....67
4.5.3 Context tracking....68
4.6 The Algorithm....69
4.7 Summary....71

SYSTEM DESCRIPTION....72
5.1 Introduction....72
5.2 System Block Diagram....73
5.3 Data Managers....75
5.4 Multitasking Aspects....76
5.4.1 Threads....76
5.4.2 Message Passing Mechanism....80
5.4.3 System Blackboard....81
5.5 Portability Considerations....84
5.6 Module Description....85
5.6.1 The Database....85
- The DBS memory format....85
- The DBS file format....88
- Operations specific to the DBS manager....88
5.6.2 The Knowledge Base Manager....90
- The KBS memory format....90
- The KBS file format....91
- Operations specific to the KBS manager....91
5.6.3 The Training Database Manager....94
- The TDBS memory format....94
- The TDBS file format....94
- Operations specific to the TDBS manager....95

5.6.4 The Knowledge Based Expert System....95
- Functions of the KBES....95
- The rule list....95
- The parser....97
- Built-in procedures....97
5.6.5 The Central Neural Network Processor....99
- Incorporating other neural network paradigms....100
5.6.6 The User Interface....100
5.6.7 The Input Module....101
5.6.8 The Output Module....102
5.7 NeXT Implementation....102
5.7.1 NeXTStep Interface....102
5.7.2 DASANEX Implementation....103
5.8 System Operation....104
5.9 Summary....109

VALIDATION AND VERIFICATION....110
6.1 Introduction....110
6.2 Validation Procedure....110
6.3 Methodology Verification....111
6.4 Application to Transformer Data....112
6.4.1 Transformer Data Revisited....112
6.4.2 DASANEX Setup....112
• Database variables....113
• Knowledge base rules....115
• Artificial Neural Networks....115
6.4.3 System Operation....118
6.4.4 Pretraining....121
6.4.5 Simulation Results....121
6.5 Conclusion....122

CONCLUSIONS AND RECOMMENDATIONS **126**

7.1 Concluding Remarks 126

7.2 Potential Application Examples 127

7.2.1 Monitoring of Transformer Data 127

7.2.2 Validation of Meteorological Data 128

7.2.3 DASANEX as a Front-End Processor 129

7.3 Recommendations and Further Work 129

REFERENCES **131**

Vita **144**

List Of Illustrations

Figure 2.1. Typical man-machine interface....8

Figure 2.2. Historical evolution of engineering problem approaches....32

Figure 3.1. Physical setup for data collection site....38

Figure 3.2. Sample data collected (May 16th-May 31st, 1991)....41

Figure 3.3. Behavior under high load (June 8th-June 15th, 1991)....42

Figure 3.4. More data anomalies (October 19th-October 26th, 1991)....43

Figure 4.1. Expert system components....50

Figure 4.2-a. Neural network node....57

Figure 4.2-b. Typical neural network....57

Figure 4.3. Selection tree for classifier paradigms....61

Figure 4.4. Hybrid system arrangements....64

Figure 4.5. The concept of 'context'....66

Figure 4.6. The algorithm....70

Figure 5.1. DASANEX block diagram....74

Figure 5.2. The data manager concept....77

Figure 5.3. Multitasking and the thread concept....79

Figure 5.4-a. MSG_LIST components 82

Figure 5.4-b. Sample message list 82

Figure 5.5. Memory format of database variables 87

Figure 5.6. Database disk format 89

Figure 5.7. Knowledge base memory format 92

Figure 5.8. Knowledge base disk format 93

Figure 5.9. BNF representation of the rule language 98

Figure 5.10. Rule Editor interface 105

Figure 5.11. Variable Editor interface 106

Figure 5.12. Network Editor interface 107

Figure 6.1. System Operation Flow Chart 116

Figure 6.2. Pre-training Data (May 8th-May 15th, 1991) 123

Figure 6.3. Sample Simulation Results - WDT25 124

Figure 6.4. Sample Simulation Results - TOTKW 125

List of Tables

Table 2.1. Potential Power Engineering areas for the application of Expert Systems 14

Table 2.2. Power Engineering areas for the application of Neural Networks 19

Table 3.1. Data collected from Martinsville 40

Table 4.1. Differences between conventional programs and expert systems 48

Table 4.2-a. Advantages of artificial over human expertise 52

Table 4.2-b. Advantages of human over artificial expertise 52

Table 4.3. Neural network paradigms 55

Table 6.1. Functional Link Network Inputs 119

Chapter 1

INTRODUCTION

Numerical data plays a highly significant role in various scientific and engineering disciplines. In research domains, it often serves on both ends of the scientific research process. Numerical data is routinely studied and scrutinized by scientists in an attempt to formulate new rules and laws governing the physical phenomenon that the data represents, or at the limit, establish valid empirical formulae. Additionally, measured data is frequently used as a validation tool to corroborate and confirm a newly proposed theory.

The importance of data integrity is more critical in practical applications. Industry relies on data measurements in order to function smoothly and properly. A power system operator needs to continuously know the various voltages and power flows at different vital nodes in the system in order to dispatch the power economically and control its generation for the purpose of ensuring system stability and reliability. Similarly, the operator at a manufacturing plant requires knowledge of the various states of the plant in terms of input and output materials as well as designated important checkpoints. In most of these applications, appropriate probes and sensors are used to convert the physical entity of interest into a number which can then be transmitted, processed and displayed for the operator, and possibly stored. This scheme is extremely useful since it enables centralized remote monitoring of an entire plant or a distributed network system.

The information collected would not be useful to the operator, however, unless it reflects the true state of the variable of interest. In a typical arrangement, a sensor converts a physical variable into a voltage which is sampled through an analog-to-digital (A/D) converter, and the resulting digital data is transmitted to the central location. During this multi-stage trip, any of the data acquisition components along the path can temporarily fail and result in corrupted and useless information. Sensors, for example, can display aberrant behavior depending on the operating conditions and usually go out of calibration with time. Furthermore, signals transmitted over long lengths of exposed cable, such as telephone lines, can be subject to intermittent adverse weather conditions. The consequence of such an arrangement is temporary ill-behavior of the data on the receiving end. Other more permanent conditions, such as the presence of a strong electric field in the vicinity of the cable, might cause a constant scaling or offset problem in the data received.

Another common practice is to store the data collected and to process it off-line at a later time. When this data is subsequently checked for correctness, and bad entries are detected, the customary procedure is to simply ignore and discard the offending data points since no information on the context in which the data was collected is usually available.

In an automatic control setting, whereby a machine makes decisions based upon the real-time on-line data it receives, a key attribute of this data is correctness, save for which the control system's decisions and actions would be totally unpredictable and certainly unreliable. The level of accuracy and the gravity of bad readings in the data depends, evidently, on the control application at hand.

A practical setting is exemplified by a modern utility control system. The fundamental building blocks of such a system are the acquisition of data, the processing of those data for use by the operator, and operator control of remote devices. One of the schemes widely used to accomplish these functions is known as the Supervisory Control And Data Acquisition (SCADA) system. Some of the duties of a SCADA system, enumerated by Gaushell and Darlington, include [60]:

- Data acquisition

- Information display
- Supervisory control
- Alarm processing
- Information storage and reports
- Sequence of events acquisition
- Data calculations
- Special Remote Terminal Unit (RTU) processing/control

It is important to note however that, although there may be provisions built into a SCADA system to detect transmission errors between the various components, the issue of data sanity checking has not been dealt with.

For all the above scenarios, what is needed is an on-line data sanity checking system which would ensure data integrity and correctness within the context of the environment in which it was collected. Such a system would serve two major purposes. The first task of the system would be to deal with erroneous data resulting from the failure of any of the components of the data acquisition system, such as sensors and probes, signal and data transmission cables and analog to digital converters. The invalid data readings would be discarded and an estimate of the expected reading would optionally be provided by the data sanity checking system to ensure the completeness and integrity of the output data. The second task performed by such a system would be the on-line monitoring of the valid input data in order to detect the failure or malfunction of hardware components of the system being monitored, or preferably the early signs and warnings at the onset of such a failure. As the early signs are detected, the system can alert the operator who can take appropriate action to prevent further problems.

The objective of this research is the design, and implementation of a data sanity checking system which fulfills the criteria aforementioned.

In Summary, it is desirable for the above system to possess the following characteristics:

- **Versatility**

  The system should display versatility in three areas:

  - *Input*: Input data will be gathered from several sources, possibly with different hardware configurations and various software protocols. All of these have to be supported.
  - *Processing*: The types of tests that can be run on the data, and the available operations that result in an output have to be very flexible.
  - *Output*: Several output actions might be required from the system in addition to detecting, discarding and/or replacing bad data. Examples could be signaling an alarm by dialing a phone number, or initiating a process to transfer data to a mass storage device.

- **Speed**

  Data arrival rates depend on the nature of the data being collected as well as the hardware being used to collect it. The system should be fast in order to accommodate all appropriate speeds.

- **Reliability**

  Since the output data could possibly be used for automatic control, the system should be capable of detecting obvious malfunctions quickly, and taking appropriate corrective measures and decisions without constant operator supervision. Therefore, the system has to contain a certain amount of expertise about the data being collected.

- **Adaptiveness**

Due to the strong dependence of certain variables on time dependent parameters such as seasonal weather patterns, the system should be able to detect and follow gradual shifts or drifts in these dependent variables.

- **Self-Learning**

Given that the system will, in some cases, be used to collect data previously not experimented with, wherein variable inter-relationships might not be known yet and cannot be explicitly provided, the system should observe the available data and acquire its own knowledge.

- **User-Friendliness**

Given the nature of the expected users of the system, mainly plant operators and supervisors, it is imperative that the system be easy to use and have a straight forward user-friendly interface.

Chapter II

LITERATURE REVIEW

2.1 Introduction

An extensive literature search has been conducted in three areas of data sanity checking, expert systems in the domain of power engineering, and neural networks in the same domain. The need for the latter topics stems from the nature of the solution proposed, as explained in chapter V.

2.2 Data Sanity Checking in the Literature

The subject of data sanity checking, as we have defined it, has not been given enough attention by the engineering and scientific communities. The closest topics to data checking that have been tackled are statistical outliers in mathematics and gross error detection in chemical plants. In the following subsections, we will first emphasize the importance of numerical data and its correctness, then we will describe the work in the areas above as well as other work in areas not as well defined.

2.2.1 Importance of numerical data

Numerical data plays a considerably important role in our everyday life. Numbers are practically used in every aspect of our daily activities. They serve as a mean of communication to convey information about countable and/or measurable quantities. Numbers also help us to perceive, and share perceptions about our environment, as illustrated by how one would mentally react to the statement : "It is 95° outside".

On an entirely different platform, the ongoing relentless and ever increasing trend of making digital computers a part of all of our daily tasks accentuates the need for a number representation for the physical and conceptual entities that form our surroundings. The reason evidently being that, at its lowest level, the language of computers is composed of numbers. Consequently, recent decades have witnessed the birth of an entire array of sensors, transducers and devices aimed at providing the computer with sensory abilities including amid others vision and hearing.

Among the various fields that exhibit a crucial dependency on numerical data, two areas are especially worthy of mention, the domains of scientific research and industrial applications. Scientists typically rely on measurements of physical phenomena to develop and verify laws and theories, or, as the case often is, perform both. In an analogous fashion, industrial processes, such as power plants, or chemical plants, heavily depend on remotely collected system status data to monitor and control the internal states as well as the output of their system.

Proceeding with the previous scenario of an industrial process, let us focus for a moment on the system depicted in figure 2.1, described by Park [132]. The illustration represents the traditional man-machine system. The machine subsystem is equipped with controls as well as displays which may be visual, auditory, or tactual. Upon reading the displays, the human decides what adjustments to make to the controls, serving in effect as the main information processing unit. As a result of the trend towards automation, man's role has gradually evolved into that of monitoring the state and supervising the actions of a fully automatic system, intervening only in unusual situations which render the system unstable or

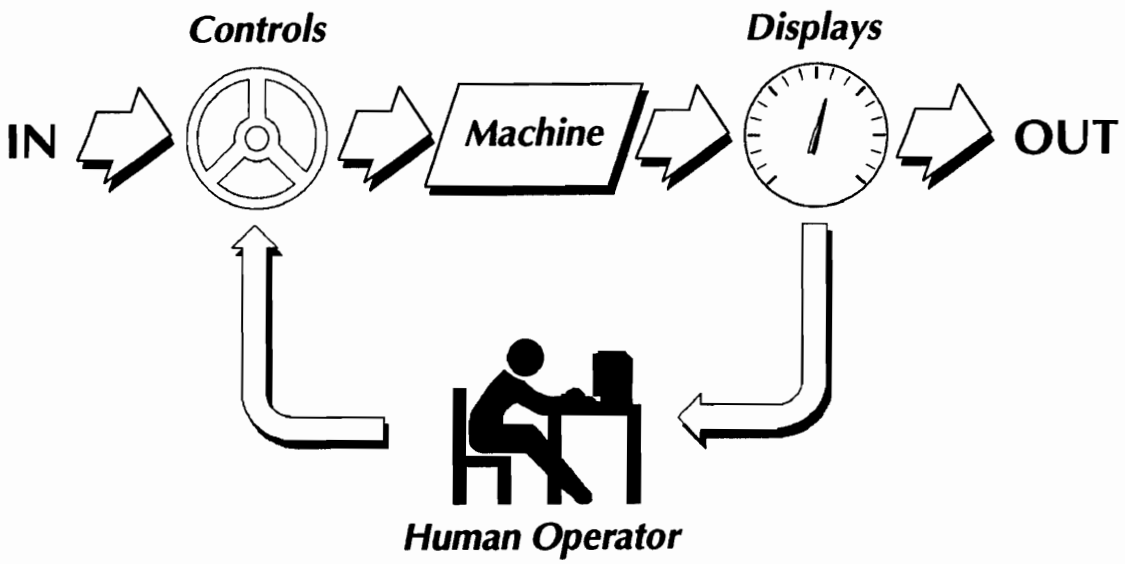


Figure 2.1. Typical man-machine interface

hazardous. This change was in a big part due to the severe consequences of a human error, which is very easy to occur as illustrated by some of the examples given by Park [132]:

- an incorrect reading of a display.
- a minor lapse of memory.
- a failure to notice a warning signal.
- a skipping of a small procedural detail.
- a misunderstanding of instructions.

Furthermore, as systems have grown increasingly complex, more information on the state and time-history of the process and system components is centralized and presented to the human operator via various displays and alarms. This informational overload degrades human cognitive performance and harbors the chance for human error.

Clearly then, there exists a strong necessity for a reliable tool to assist, if not replace the human operator in typical monitoring tasks.

2.2.2 Outliers in statistical data

Measurement errors from metering equipment typically fall into two broad categories :

- Random errors, independently and normally distributed with zero mean and known variances.
- Gross errors, caused by non-random events such as measurement biases and instrument malfunctions.

Scientific theories and hypotheses usually account for the former type of errors and have provision to explain them. The latter kind however are typically troublesome and require exceptional handling if they are to be treated as valid data points. These readings are what is referred to as outliers. Several books and dozens of articles have been written that deal with outliers and the statistical methods to handle them. Barnett and Lewis [13] offer an excellent

treatment of the subject of outliers and its various facets, and provide an extensive list of references dealing with the topic.

2.2.3 Error detection and correction

The schematic diagram of the man-machine interface shown in figure 2.1 is descriptive of numerous systems in the industry. Some examples are chemical plants, power plants and process industries. Recognizing the need for data correctness, researchers in each of these areas have sought solutions to the problem. Generally, the solution is of mathematical nature and uses some outlier detection algorithm.

In the discipline of chemical engineering, the issue of recognizing and adjusting erroneous measurement data has been tackled since the early 60's by Ripps [143]. Ripps handles single occurrences of a bad measurement by discarding each measurement in turn and computing the minimal least square relative error. Tamhane and Mah [168] formally define the process of adjusting measured variables and estimating the unmeasured variables to satisfy mass and energy balance constraints as the data reconciliation problem. Additionally, they define the task of detecting the presence of gross errors as the gross error detection problem. They propose a statistical test for gross error detection based on residuals obtained in the reconciliation step. Iordache et al [75] expand one the work by Tamhane and Mah to develop a simulation procedure that can be used to design a gross error detection scheme for any specific application. Romagnoli and Stephanopoulos [145] propose an algorithm, also based on statistical methods, to locate and rectify gross biased measurement errors. The algorithm is computationally simple and significantly reduces the data reconciliation problem.

2.2.4 Other data checking research

Several researchers have approached the issue of data sanity checking. However, all these attempts suffer from lack of generality and are usually application specific. Hence, these approaches cannot be classified under a unique research category. For instance, Adibi et al [2]

propose a methodology for remote calibration of measurements to be used for state estimation of power systems. Through monitoring of telemetered values of voltages and real and reactive power flows over several days, the technique determines correction factors to be used with the measured values. The effect is that installation errors are eliminated, systematic errors are minimized, and random errors are evaluated, providing the state estimator with zero-mean measurements, and the actual manufacturer specified standard deviations. In an EPRI technical brief [46], the authors describe an automated remote monitoring system (ARMS) which diagnoses developing faults and ensures machine protection by monitoring synchronous and asynchronous vibration data. This system is limiting since it is applicable only to vibration data. Similarly, Filbert [49] achieves fault diagnosis by continuous time parameter estimation.

The research on data sanity checking closest to our definition has been performed by Fox et al [53]. They implement a expert system called PDS in which they address the issues of spurious readings and sensor degradation related to the analysis of sensor based data. Unfortunately, they do not pursue the issue any further and shift in their research to other topics. Gaushell and Darlington [60] provide an overview of the functions of a SCADA system and the problems associated with it. They offer hardware solutions such as distributed processing to reduce data collection errors. Holzworth [71] reports on a system for on-line thermal performance monitoring. The system also provides its various users, the operator, the plant performance engineer, and management, with different types of technical and economic data tailored to their needs. Once again, this system is too application specific and cannot be easily modified to monitor an environment other than the one it was designed for.

Kim and Agogino [86] develop a knowledge-based system for performing sensor validation towards monitoring and diagnosing heat rate degradation in fossil power plants. Rowan [148] examines in a general sense the application of sensor-based, on-line, experts systems in process-manufacturing environments. The article investigates the benefits of such applications and the necessary tools required for their actual implementation. Finally, Sahba [150] presents a method for plausibility check and bad data detection and replacement in the context of power systems. The method relies on a network search technique and linear programming as well as static state estimation. Obviously, the method cannot be general and is restricted to power system networks and the like.

2.3 Recent Expert Systems In Power Engineering

Recent years have witnessed a major rush towards building expert systems in most engineering disciplines. This wave is mostly attributed to the numerous advantages and potential benefits that characterize expert systems. These characteristics are summarized by Fenley [48] and are listed below:

- Expert systems make scarce expertise more widely available within the organization, thereby helping non-experts achieve expert-like results;
- They free human experts for other activities besides repeatedly solving the problems which an expert system could address;
- They promote a standardized, consistent approach to solving relatively unstructured tasks;
- They enhance organizational effectiveness and efficiency by making readily available solutions to difficult problems which might otherwise require time-consuming research or consultation with experts to solve;
- They provide a means for capturing and storing valuable knowledge that might be lost if an expert left the organization;
- They provide a means for permanent retention of highly complex knowledge, since machine knowledge does not deteriorate with time or disuse as human knowledge tends to;
- They perform at a consistently high level tasks which humans might perform inconsistently due to fatigue or loss of concentration.

The field of Power Engineering with its ample domains provides numerous possibilities for applying expert system techniques. Some of the potential applications and areas where artificial intelligence techniques are expected to yield some improvements are investigated by

Wollenberg and Sakaguchi [187] and extensively enumerated by Schulte et al [157] and are listed in table 2.1.

Several expert systems in these areas have been implemented in the past decade. Sakaguchi et al [154] offer a comprehensive analysis of artificial intelligence techniques with an emphasis of how they can be applied to the areas above, illustrating each section by a system either under research and development or already in practical use at a commercial level.

We will conclude this section by a brief description of some of the systems classified in the categories above that have been recently reported in the literature.

Akimoto et al [6] report on a prototypical expert system to assist in system stability analysis, time simulation, problem determination and remedial measure selection. Baggini et al [11] develop a knowledge base assistant for load flow calculations which performs analysis of networks of significant size (400-500 nodes and 600-800 links) in a few tens of minutes. An added advantage of the system is the automatic load flow result analysis which provides tremendous time savings for the planner. Bhatnagar and Rahman [15] compare the results of using a signal processing technique and a proposed expert system for short term load forecasting and conclude that the expert system based method yields better results for one to six hour forecast.

Biswas et al [16] discuss the use of knowledge-based methods to risk assessment of technological processes. Chan [26] describes a real-time expert system integrated in a large SCADA system which provides remote monitoring and control facilities to an entire 22kV network. The system proves to be a useful operator support tool, especially in emergency situations. Criado et al [34] implement an expert system that performs contingency analysis and recommends control actions to take the power system to a safer state. Suggested preventive or corrective measures include topological reconfiguration, generation reallocation, tap changes, capacitor and reactor switching, and load shedding.

Table 2.1. Potential Power Engineering areas for the application of Expert Systems

- ❑ Contingency Analysis
- ❑ Alarm Processing
- ❑ State Estimation
- ❑ External Model Estimation
- ❑ Remedial Action
- ❑ Automatic Generation Control
- ❑ Economic Dispatch
- ❑ Unit Commitment
- ❑ Short Term Load Forecasting
- ❑ Interchange Evaluation
- ❑ System Restoration
- ❑ Contingency Relay Arming
- ❑ Energy Cost Reconstruction
- ❑ Load Shedding
- ❑ Load Management
- ❑ Trouble Call Analysis

Don Russel and Watson [40] investigate the possibility of replacing the passive and responsive conventional automation and control devices in a substation by an adaptive knowledge based system. Fandino et al [47] make use of an expert system to develop an optimal restoration strategy of a large UHV system including numerous nuclear power plants following a blackout. Fustar and Jelavic [56] introduce a knowledge-based expert system for weekly power system operation scheduling to help overcome the problem of improperly structured constraints which make algorithmic approaches difficult and inadequate. The proposed weekly scheduler involves load prediction, inflows prediction, run-of-river hydro production, storage hydro production, unit commitment, generator maintenance and interconnected systems power interchange.

Galiana et al [58] describe the design and implementation issues of an expert system for insulation coordination with a discussion of justification, knowledge acquisition, domain knowledge summary and knowledge representation. Giuffre and Johnson [61] propose a demand side management assistant which will provide real-time support for decision makers who manage dispatchable loads. The benefits of the proposed system include avoided cost, additional sales, and increased productivity.

From the preliminary results of attempting genetic algorithms to voltage optimization, Haida and Akimoto [64] conclude that, though not as accurate as more conventional approaches such as the simplex method, the genetic algorithm approach has the ability of improving its performance and adapting to system changes. The authors reflect on the applicability of the technique to other optimization problems in the power field. Inoue et al [74] report on the development of an easy to use expert system for annual power plant maintenance scheduling. The emphasis is placed on ease of modification of the system description, and on the user friendly graphical user interface.

Kakimoto et al [81] develop an expert system which restores a power system from a complete outage, including knowledge and control of the individual stations. The system prescribes the operations to be carried out at every station, specifying the order in which to restore the stations. The computational time required for generating all restorative operations amounted to 0.5 seconds, and the system was able to handle cases where equipment damage

was present. Keronen [83] describes an efficient, user-friendly system for real time operation planning and event analysis based on object-oriented representation and a graphic model editor, while Lee et al [97] describes enhancements made to PROSET, an expert system for setting and coordination of protective relays used in the ultra high voltage transmission systems. Similarly, Liu et al [101] report a knowledge-based system for remedial control of distribution feeders. The system provides corrective decisions in less than 30 seconds, which constitutes an acceptable processing speed.

The domain of automated load forecasting is tackled by Matsukawa [106], Rahman [136] and Rahman and Baba [137,138] who offer knowledge-based approaches which require a relatively short historical database. Additionally, Rahman and Bhatnagar [139] relate the successful implementation of a knowledge based expert system with inventive and versatile heuristics to forecast short term load. Saitoh et al [151] concentrate on the load-temperature relation and its seasonal variation to develop a system with automatic model selection.

Mokhtari et al [112] discuss and describe the implementation of an expert system to assist power system operators in scheduling the operation of generating units. Nilsson [125] examines the various phases that a knowledge engineer must undergo in order to efficiently implement an expert system. The steps are reinforced by examples from the author's own experience in implementing a rule based system to assist the operators in a typical electrical power plant. Ranade et al [141] implement a rule-based system for developing strategic guidelines and providing security assessment with respect to line overloading. The rule base developed captures the heuristic approach used in practical power system operation.

Singh and Walther [160] combine the mathematical soundness of the weighted least square state estimation algorithm with the heuristic knowledge used by experienced dispatchers to obtain a more robust, intelligent state estimator. Tomsovic and Ling [172] propose a fuzzy mathematics structure to serve as the translator between the numerical values of traditional methods of power system security analysis and heuristic approaches. The structure will allow proper integration of these methods to achieve maximum performance in information transfer. Wagenbauer and Brugger [179] describe a system which combines model-based and rule-based

approaches for intelligent alarm processing. The completed system has been performing in a satisfactory manner for over a year.

In the domain of restoration of distribution systems, Sakaguchi and Matsumoto [153] present a heuristic based expert system for automatic control of a power system in the restorative state, with special emphasis on methods to couple the highly numerical task of overload check with the purely symbolic process of reasoning with the knowledge base. Liu et al [102] approach the same problem by proposing an automatic control expert system to restore the maximal number of zones by following the steps of group restoration, zone restoration and load transfer, while Sahba [150] offers a method to perform plausibility checks and data validation in power systems.

Finally, a general discussion of the application of expert systems to power engineering problems if offered by Sakaguchi et al [154], Schulte et al [157], Wang [180], and Wollenberg and Sakaguchi [187]. Zhang et al [190] provide an extensive bibliographical survey of expert systems in electric power systems.

2.4 Neural Networks In Power Engineering

Power engineering is a very broad domain featuring a wide spectrum of problems. Most of these problems share many common characteristics: uncertain, incomplete or unknown environmental conditions, criticality of response or solution time, overwhelming amounts of complex data (information overload), etc. Since this "fuzzy" environment is where neural networks excel, it is natural that the attention of the power engineering community has been focusing more and more on the practical applications of this technology to solve power system related problems.

An extensive bibliographical survey of recent publications accounting for the application of neural networks to various areas of power industry has been conducted. The findings reveal that interest in the subject has only surfaced in the last couple of years and that it is growing rapidly. A review of some of these articles is presented here, classified by area of application. For each article, a brief statement of the problem is presented, followed by the

particular paradigm selected by the authors to address the problem, their reasoning as to why the selected paradigm is appropriate, and some of the results and conclusions they reached. The field or area breakdown is collected from Schulte et al [157] and Zhang et al [190] and is shown in table 2.2.

2.4.1 Alarm Processing

The problem of interpreting a large number of alarms in a control center under stress conditions is well known and well appreciated in the electric utility industry. The system operator is usually flooded with alarm data instead of information. Chan [27] identifies problems with conventional Boolean algorithmic and rule-based expert system techniques. These are mainly the inability of dealing with the combinatorial explosion of multiple alarms, and the difficulty of organizing and maintaining knowledge bases due to the frequently changing nature of power systems. The associative memory and on-line learning capability of neural networks solve both of these problems. The author uses a back-propagation-trained feed-forward network as a pattern classifier to identify the fault(s) corresponding to a given pattern of alarms. The resulting network achieved 100% correct interpretation when presented with complete alarm data. When incomplete data was presented, the network guessed the right fault whenever the pattern provided was not ambiguous. This behavior is similar to that of a human operator.

2.4.2 Analysis and Speculation

In this category, authors discuss the favorable properties of neural networks and the possibility of their application in solving various power system problems. Damborg et al [36] consider applying neural networks to dynamic security assessment, static security assessment, and load forecasting. They propose a possible solution to the large scale problem of power systems by the parallel application of several networks, each focused on one very narrow, local subsystem. Vadari and Venkata [178] present multiple ways in which expert systems and neural networks can be combined in order to take full advantage of both. The basic approach they suggest is one where the expert system and the artificial intelligence techniques are used off-line to train the neural network. On-line operation is then performed solely with the neural

Table 2.2. Power Engineering areas for the application of Neural Networks

- Alarm Processing
- Analysis and Speculation
- Contingency Analysis
- Economic Dispatch
- Fault Diagnosis
- Load Forecasting
- Machines
- Modeling
- Reactive Power and Voltage Control
- Security Assessment
- Switching Operation
- Topological Observability
- Transient Stability Problems

networks. They exemplify their proposed method with a system for saving the power system from voltage collapse. In a similar article, Wang [180] argues for the application of fast, parallel networks to on-line contingency, switching, and restoration problems. Yiftah [189] speculates on the possible uses of neural networks in the nuclear industry.

2.4.3 Contingency Analysis

Contingency analyses are periodically performed during normal system operation to determine whether the system can withstand a single or possibly double fault. The main constraints on the post-fault operation are the generator capacity limits and line flow limits. Due to the large number of single and combined faults possible, this analysis is usually performed off-line, following any change in system configuration. Chow et al [29] use an improved version of the Hopfield network to classify contingencies in terms of the number and type of limit violations. By formulating the contingency classification problem as a pattern recognition problem, contingency analysis can be performed continuously on-line. Fischl et al [50] use a back-propagation network to screen contingencies and to identify those which may put the power system in an insecure state. This approach effectively cuts down on the number of faults to investigate and allows on-line contingency analysis. In a related work, Fischl et al [51] use a Hopfield network to detect the limiting contingencies in power system operation. The need for detecting limiting contingencies arises when a system must restrict real power transfers because the system voltage is sensitive to the transfer levels. Traditionally, when a system has reactive power problems, it has to rely on off-line study results in real-time operations to limit system transfer. Again, by formulating the contingency ranking problem as a pattern recognition problem, the authors were able to achieve on-line detection of limiting contingencies. Yan et al [188] present a hybrid expert system/neural network solution to the problem of classifying contingencies. The expert system performs "coarse" screening to determine the type of security limit violations. It then directs the potentially harmful contingencies to the appropriate neural network to perform further analysis and achieve a "finer" screening. Finally, Zwingelstein et al [191] apply a backpropagation network for predictive maintenance activity for nuclear power plants. The results show the efficiency of the approach with noisy, incomplete

patterns. Given the large number of inputs and the lack of preselection, they report a very slow network training phase, which they plan to remedy.

2.4.4 Economic Dispatch

Economic power dispatching involves minimizing a quadratic cost function subject to system constraints. Incidentally, when a neural network is converging, it is minimizing a similar quadratic energy function. By formulating the economic dispatch problem so that the cost function coincides with the network error, Maa et al [103] are able to exploit the fast and parallel nature of neural networks. In a similar study, Matsuda and Akimoto [105] develop a method for representing large numbers in a Hopfield network and apply the resulting network to the problem of economical load dispatching. They find their approach to be superior in the ease of formulation of the problem and in its memory and computation time efficiency, both for stationary and time-varying loads. Fukuyama and Ueki [55] treat this combinatorial problem by using a Gaussian machine, which offers some improvements over the Hopfield network. The results achieved are suboptimal, yet feasible. Sendaula et al [158] present a method to solve simultaneously the unit commitment and the associated economic unit dispatch problems. This is achieved by imbedding the various constraints in a generalized energy function, then using a Hopfield-Tank type combined with a Chua-Lin type network such that the derived energy function is a Lyapunov function of the neural network. The noted advantages of this technique over conventional ones are the avoidance of slack variables when expressing inequality constraints, and the independence of the convergence on the number of units dispatched.

2.4.5 Fault Diagnosis

In normal power system operation, data reflecting system status is continuously collected. Of special interest is the data obtained immediately before and immediately after a system fault. Statistical analysis of this data has not been very successful in predicting and preventing subsequent faults. Expert system approaches were less successful given their need for explicitly stated rules, which could not be obtained from mere observation of the data.

Baumann et al [14] make use of a Kohonen self-organizing network to perform an impulse test fault diagnosis on power transformers. The technique relies on a model of the

transformer and can be used during acceptance tests. Using 28 different impulse test signals, the system was able to generalize and recognize unknown patterns with a classification error of less than 5%. Taking advantage of neural networks' ability to learn complex relationships from examples, Ebron et al [41] utilize a feed-forward back-propagation network to detect incipient faults on power distribution feeders. After being trained using simulation data, the network is tested using data from a fault-library. The network displayed the ability to detect high impedance faults, which cannot be reliably detected and cleared by conventional protective devices.

Hui and Short [73] apply the neural network approach to determine the existence of feasible load flow solutions. Voltage collapse evaluation methods suffer from the time consuming process of solving the stiff non-linear system equations, process which makes them inefficient for on-line monitoring of voltage collapse. A backpropagation network trained on 90 pattern pairs attained 1×10^{-3} accuracy and was able to evaluate system stability in a few seconds with results very close to the minimum singular value method which took over three minutes to calculate. This high performance suggests that the neural network can be used successfully on-line. By applying backpropagation to fault diagnosis, and a Boltzmann network to localization of the faulty component, Jongepier et al [79] are able to achieve 80% recognition, with corrupted patterns, for diagnosis, and 95.1% recognition for localization.

Swarup and Chandrasekharaiah [167] explore the suitability of the classification capabilities of neural networks for fault detection and diagnosis. Tested topologies using single and multiple hidden layers are found to perform similarly, albeit requiring varying training time. The resulting systems diagnosed single faults correctly and displayed graceful degradation in the presence of simultaneous multiple faults. Tanaka et al [169] test a similar arrangement on a small power system with encouraging results. They also offer suggestions on how neural networks can be applied to large power systems.

2.4.6 Load Forecasting

Load forecasting has long been an interesting and exciting problem for power system researchers. An especially important task, which is of special interest for both power

producing utilities and bulk power consumers such as factories, is predicting correctly the time and magnitude of the peak load. Several statistical approaches and knowledge-based approaches for handling this problem have been proposed over the years. The basic difficulties for all these approaches are the correct selection of pertinent variables, and the size of analysis or look-up database. Given the inherent properties of neural networks to learn from examples and to synthesize intermediate data from samples, it would be expected that neural networks should perform acceptably well in the task of load forecasting, provided the important and critical variables are identified and used as inputs to the network. Researchers who have attempted this technique arrived to the same conclusion.

Brace et al [17] run a competition comparing two neural network approaches to six other established forecasting methodologies. The networks did not rank well and were outperformed by all but one of the other techniques. The authors argue that part of the reason is the difference in forecasters' expertise. When the neural networks were trained with the same inputs that were supplied to the best technique, their performance improved dramatically. In a similar work, El-Sharkawi et al [44] test several neural network structures, addressing the issue of feature extraction, and compare the results obtained with several commercial forecasting systems. It is found that feature extraction, together with adaptive training, substantially improves the performance of the network, yielding forecasts of the same accuracy as those produced by the best and most sophisticated commercial systems.

Elliott [45] uses a back-propagation network having one hidden layer with ten nodes to predict load one hour ahead. For inputs, he uses time, temperature, current load, and load fifteen minutes earlier. Even though not extensively tested, his prototype demonstrated that its results were comparable to those of the current best alternate method. In addition, it provided the advantage of ease of implementation and speed of prediction. In a similar effort, Park [131] and Peng et al [133] demonstrate the use of a continuous learning back-propagation network which has been shown to follow the change in load pattern. They test the network on data consisting of load, maximum and minimum dry-bulb temperature for all the Tuesdays, Wednesdays, and Thursdays over one year period. The network is repeatedly trained using data for two consecutive weeks, and then used to predict the load for the following week. The results were acceptable except for instances when the given day happens to be a holiday. The

authors offer suggestions on how to modify the network to accommodate this type of information.

Srinivasan et al [166] use a traditional backpropagation network with variable learning rate and a linear output transfer function. By carefully selecting their input variables, they achieve better forecasting results than the conventional and stochastic methods. Using a similar setup, Lee et al [96] achieve forecasting errors of less than 2% for the percent relative error.

2.4.7 Machines

Unlike large rotating machines which have routine check-ups to detect conditions that could lead to faults caused by deterioration due to age, and special devices designed to protect them from disturbances and mis-operations, medium size and small machines have fewer check-ups and fewer protection devices for economic reasons. But they frequently develop common problems like conductive contamination and worn bearings. By learning from data collected during the normal operation of these machines and during the first stages of degradation, a neural network can be used to detect incipient faults in these machines in an inexpensive and efficient way. Realizing this ability, Chow et al [33] implement a layered feed-forward neural net that uses the armature current and rotor speed of a DC motor as inputs, and outputs a signal that flags the early stages of fault occurrence. The network is trained with data obtained from a simulation of the motor in various operating stages. Test results prove to be acceptable, though better results are expected from a network trained with real data or with a more complete model. In a related study, Chow and Yee [32] extend the application to induction machines. They discover that a simple feed-forward network is not sufficient to carry out the task, so they augment it to a functional-link network by adding input nodes for the products of the input variables. They conclude that the resulting neural network which performs the detection based on direct measurement from the motor, avoiding the complication of the non-linear equations and modeling errors, yields satisfactory results even in the presence of noise. In later work, Chow et al [31] use learning theory, which is derived from maximum entropy, to determine the proper number of training examples needed to reach a specific accuracy level before actually training the network. They use the results successfully in training a neural

network for incipient fault detection in induction motors without the extensive, slow and time consuming trial and error simulations.

Saitoh and Iwamato [152] combine neural networks with fuzzy control theory as a substitute for optimal control of synchronous machines. In order to solve the backpropagation convergence problem resulting from such a modification, they eliminate training data which has large errors. Although not as good as using optimal control at the linearized point, this method allows for control of the power system at a much wider range than the optimal control method.

2.4.8 Modeling

Neural networks are very adept at reproducing the input/output relationship of a system given just sample input/output pairs. Furthermore, their interpolative nature allows them to produce acceptable outputs for unknown inputs lying in the space between the sample inputs. Several researchers have taken advantage of this ability and of the speed of recall of these networks to model systems that are inherently non-linear, complex, and computationally involved. Chow and Thomas [30] use a multi-layered feed-forward network to capture the behavior of a synchronous machine. Results show that the neural network models the machine very closely, and that accuracy is increased as the number of hidden nodes is increased. Harashima et al [65] use a neural network to act as a controller for a pulse-width modulation inverter and generate the required on-off switching pattern.

Kraft et al [90] implement a rule-based expert system with an embedded neural network to model the boiler in a fossil fuel power plant. A recurrent neural network is used to model the behavior of the boiler under normal conditions. The errors between the output of the model and actual measurements trigger rules in the rule-based system which identify abnormal conditions. Pao and Sobajic [128] demonstrate the feasibility of combined artificial intelligence / neural network methodology for carrying out dynamic power system analysis in real-time. They couple an analytical model with a functional link network to produce a CNA, or combined-neural-net-analytic multi-machine power system model. The CNA features a coarseness scale which can be adjusted to tune the speed of the model response time for real-time performance.

Thomas and Ku [170] propose the use of neural networks to model the load for power system analysis. Their results verify that artificial neural networks can actually emulate the response of certain known load models, with the benefits of a much faster response time. They propose further working on simulating dynamic loads in the same manner. Werbos [183,184] applies a back-propagation network to a recurrent gas market model. He derives a generalization of the technique that utilizes its own outputs as inputs and therefore avoids storage of intermediate results.

2.4.9 Reactive Power and Voltage Control

Most reactive power compensation methods for a distribution system suffer from the long computation time required for the optimization, and the required knowledge of all loads for optimal control. Since such monitoring is impractical, these methods have traditionally been based on load forecasts, and the capacitor control has been determined off-line to be used later. A neural network approach offers fast results without the need for all the previously required measurements. Ajjarapu and Albanna [5] apply the genetic algorithm optimization method to reactive power dispatch. This approach is particularly promising since, at every iteration in the search process, searching is performed from a population point of view, not from a single point, which could locate false peaks. Additionally, genetic algorithms only need to know objective functions and avoid the usage of derivatives. The subject, particularly the representation of multi-parameter codes, are still under research. Dash et al [37] compare the performance of backpropagation and a combined backpropagation-Cauchy's learning algorithm at controlling multi-tap capacitors on the 30 bus IEEE test system. The modified net yields faster convergence time although both nets perform equally well once fully trained. Neily et al [120] attempt the application of neural networks to Joint VAR Control (JVC), which is required in power stations with more than one large synchronous machine. This being their first attempt, the results are promising, though a better choice of input variables, and possibly feature extractions are necessary. Santoso et al [155] use a three-layer feed-forward network having for inputs the active and reactive line power flows, voltage magnitudes, and the current capacitor settings at certain buses. The network outputs the required capacitor settings which would yield minimal system losses. Tests on a 30 bus distribution system give satisfactory

results in terms of speed and correctness. The parallel nature of the network is such that it is a very viable candidate for on-line implementation.

2.4.10 Security Assessment

Security assessment has three main concerns: dynamic security assessment, static security assessment, and critical clearing time estimation. Dynamic security assessment consists of determining whether or not the system will reach a steady operating state after a disturbance. Static security assessment addresses whether the steady state operating condition of the system violates given operating constraints. These constraints ensure that the power in the network is properly balanced, the magnitudes of all bus voltages are within acceptable limits, and the thermal limit of each transmission line is not exceeded. Critical clearing time is the maximum time after which the system will be unstable if the fault does not clear. Critical clearing time is a complex function of pre-fault system conditions (operating point, topology, system parameters), fault structure (type and location) and post-fault conditions that are in part dependent on the protective relaying policy. To define analytically such a relationship would be highly desirable, but the diversity of variables involved makes this task extremely complicated.

Both static and dynamic security assessment are basically pattern classification problems. If the system state variables fall within a certain boundary, the system is secure. Otherwise, it is insecure or unstable. This boundary is usually complex, but a neural network is capable of synthesizing it given enough training samples. Several researchers use a feed-forward back-propagation network to determine this boundary and classify the system status according to it. Once trained, the network can be used as an on-line aid for operators to help them determine the security of their system quickly and efficiently. Work on static security assessment was carried out by Aggoune et al [4], El-Sharkawi et al [43], and Nishimura et al [126]. Dynamic security assessment was tackled by Aggoune et al [3], El-Sharkawi et al [42], Sobajic and Pao [164], Sobajic et al [163,165], and Thomas et al [171]. In all these applications, the inputs consist of the real, reactive, and apparent power of the system at given buses. The output is binary valued indicating secure/insecure operation.

Given the dimensionality of the dynamic problem, it is infeasible in present power system control centers, using conventional methods. Kumar et al [91] investigate the formulation of DSA requirements in a manner tractable by neural networks. They conclude that the problem need to be decomposed into manageable subproblems to which an expert system, or a neural network approach would be applied as appropriate. The sub-modules would then have to be integrated into one functional unit. Mori [113] uses a multi-layer feed forward perceptron to estimate the dynamic stability index of a power system with a resulting average error of about 6%.

Critical clearing time determination was attempted by Pao and Sobajic [129]. They use a feed-forward network with several inputs and one output. The inputs consist of the rotor angles of each generator, the acceleration parameters of each generator, and the individual accelerating energy of the system generators. The output is the synthesized critical clearing time. They report successful and encouraging results.

2.4.11 Switching Operation

Due to the constantly changing power system conditions, power has to be routed through different paths to achieve smooth, optimal power flow. This is achieved through switching operations in response to variations in system loading and generation. Fukui and Kawakami [54] implement a Hopfield-based neural network to handle this problem. Using the energy function of the network to represent the non-equality constraints, they achieve promising results. Hayashi et al [66] apply the Hopfield network to the Newton-Raphson load flow calculation of polar coordinates. The network provides the output of the Jacobian matrix at each iteration. The prototype has been tested on a 6 bus system with promising results. Khaparde et al [85] use an adaline model in the protection of transmission lines. Training the relay as a pattern classifying device, they quantize the input variables and perform training off-line. The converged weight matrix is then used on-line with encouraging results.

2.4.12 Topological Observability

Topological observability analysis ensures one-to-one correspondence between measurements and nodes. It is required every time system configurations are changed before

state estimation runs can be carried out. State estimation is used to reduce uncertainties such as meter and communication errors. By nature, topological observability is a combinatorial optimization problem, matching the characteristics of Hopfield networks. Mori and Tsuzuki [115,116] and Mori [114] transform the topological observability problem into one of integer programming and apply a Hopfield network approach to solve it. They achieves more reliable results by using a stochastic Boltzmann machine which consistently converged to a global minimum of the energy function, in contrast to the Hopfield network which easily got stuck at local minima. Alves da Silva et al [8] propose the use of a feed forward neural network as a complementary tool for topology determination. The classifier is called upon to perform very fast checks (in the order of 0.05 seconds) when a certain topology is doubtful. In a similar study, Alves da Silva et al [7] compare the performance of a method they present, namely the Event-Covering method (EC), and the Optimal Non-linear Associative Memory (ONAM) at network configuration, observability analysis, and bad data processing. They conclude that EC provides better filtering capabilities than ONAM, especially in the presence of noise. However, an analog hardware implementation of the trained ONAM would be faster than a digital implementation of the EC.

Jilai and Zhuo [77] utilize a multi layer perceptron network to evaluate the external steady state equivalents of a power system from the internal and boundary system states. Trained off-line, the network performed on-line evaluation at less than 4 / 1000's of a second. Recognition errors were less than 5% for 99% of the test cases, and less than 10% for all test cases. The results show that the method is suitable for on-line operation. Nakagawa et al [119] perform a feasibility study of using a Hopfield network to perform state estimation computation in a fast time given the inherently parallel architecture of the proposed network. The need for this study arises from the fact that the state-of-the-art fast decoupled method currently used, which requires a Von-Neumann type computer, is reaching a speed limit as far as the solution techniques are concerned. The method results in estimation values very close to those of the fast decoupled method. Even though simulation results on a conventional computer took longer than the normal method, the authors estimate that, with the proper network hardware, time reduction ratios of 10 or more could be achieved.

2.4.13 Transient Stability Problems

Transient stability problems manifest themselves during system transition between two steady states. Methods to assess system stability during this period usually consist of pattern recognition in the frequency domain. Mori et al [117] use a three-layered back-propagation network to successfully identify harmonic loads. Ostojic and Heydt [127] use the superior classification capabilities of neural networks to implement a system which utilizes spectral monitoring of electromechanical oscillations to assess the transient stability of interconnected power systems. The result is an accurate and reliable real-time decision-making system which virtually eliminates both false dismissals and false alarms. Uhrig and Guo [176] demonstrate the feasibility of a back-propagation network to recognize transients of a U-tube system generator. They represent the state of the plant by a unique pattern of instrument readings, which they feed as input to the network. The network distinguished six different kinds of perturbations and gave the correct output even when the input data had very large levels of noise.

2.5 Summary

The above review reveals a clear scientific and technological trend that has been taking place in many an engineering discipline, power engineering being but one of them. Till the late seventies, most problems encountered in science and industry were tackled using some of the well established traditional techniques, usually based on statistics and other sound mathematical tools. These approaches frequently proved inadequate, however, due to the difficulty in forcing all physical phenomena to conform to a limiting, oft incomplete, mathematical model. Moreover, the only solutions possible using traditional techniques were algorithmic in nature, which is but one class of possible solution approaches.

The advent of expert systems in the late seventies provided alternate solution methods. Solutions based on heuristics could now be taken advantage of. Furthermore, a whole class of problems, symbolic in nature, has a relatively feasible solution. For several years, the scientific community undertook the task of applying this new technique to the problems to which a

mathematical solution had previously applied, and to an array of new problems that were now possible to solve.

The emergence of neural networks in the late eighties was quite welcome since expert systems had their share of shortcomings and did not always provide a practical or optimal solution. For a couple of years, attention was then focused on attempting to solve engineering problems using this new technology. Even though acceptance was slow at first, the number and frequency of publications show that the technique has finally established itself as a viable alternative to other solution techniques.

Lately, it seems that researchers have started to think of neural networks as a tool that can be used as part of a solution approach, rather than being the solution itself. Recent publications report a growing number of what is referred to as Hybrid Systems, an expert system and a neural network combined in one of several possible configurations such that maximum advantage is taken of both. Judging from previous trends, this approach, currently state of the art, will experience more popularity among researchers in the coming years, until some novel technique is brought forth. This historical evolution is summarized in figure 2.2.

Another conclusion drawn from the publications reviewed is that the subject of data sanity checking has not been accorded enough attention in the literature. The usual attempts at solving this problem are specific in nature and are restricted to the problem at hand, thereby limiting the applicability and value of the approach to other problems and disciplines.

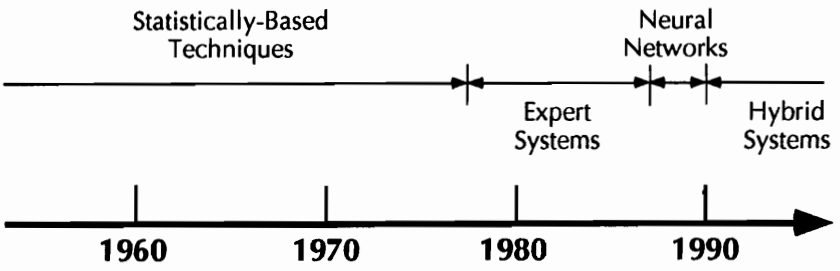


Figure 2.2. Historical evolution of engineering problem approaches

Chapter III

PROBLEM DEFINITION

3.1 Introduction

Several activities carried out in industrial and other experimental environments require automated data acquisition as part of their procedural tasks. Examples are a multi-day stress test experiment on a steel beam in a civil engineering laboratory, system bus voltage and reactive power measurements on an electrical power plant operator's console, or raw and processed material quantities at different checkpoints in a production plant. In the electric utility industry, the data collected by the supervisory control and data acquisition system (SCADA) would typically be supervised on a continuous basis, and a human operator would have to ensure normal system operation and intervene in the event of incidents or abnormal system behavior.

Generally, the data collected consists of a set of inter-related variables which, when put together, form a certain context. The elements in this set of variables fall in two classes. For the first kind, mathematical relationships linking them to others are known (ex: power flow through a line when both voltage and current are also measured). The second class are those for which relationships are not established, whether exactly or empirically, though they are known to exist (ex: the relationship linking ambient temperature to cloud cover and global solar

radiation). Taken separately, these variables will have values which may fall anywhere within a specified range. When considered as a set, however, the equations binding these variables together reduce the range of possible values to a much smaller subset.

As outlined in the previous chapter, measured data often experiences several problems which temporarily destroy the harmony of all the variables as a set. These problems result in incorrect measurements which, unless detected and corrected, cannot be used for their predefined purposes. As an example, an energy meter used for billing, which reports erroneous readings, could result in loss of revenues. In essence, the problem is two-fold: determining the occurrence of bad data; and dealing with those bad measurements.

To address the first problem, statistical outlier detection algorithms can be used. The most widely used technique, however, is data range checking. Oftentimes, simple range checks on the data are not enough to guarantee its integrity. A quantity measurement could be within the limits of its acceptable range, yet its value would be meaningless when considered in the context.

The general practice in dealing with the second point of concern (handling bad measurements) has been to work around the problem and provide temporary fixes to arrive at a result. Depending on the setting and the relative importance of the data being collected, the solutions usually considered range from simply disregarding bad data to installing multiple and redundant metering equipment. Other case specific solutions have also been offered in the literature. They usually suffer from lack of generality and do not offer generic rules or guidelines for performing data sanity checking.

Multiple metering is not always a feasible nor an optimal solution to the problem. In an environment where it is necessary to monitor a large number of different physical variables, the mere installation and maintenance of multiple metering equipment can prove to be very costly. Moreover, multiple measurements of the same quantity result in a phenomenon known as *data explosion.* Consider the case where two meters of similar reliability indices are being used and they report two different readings with an obviously large discrepancy. Which meter should be taken for its face value? Or should both measurements be averaged? Thus, instead of providing a solution, more detection and correction problems are created.

The need is therefore felt for a mechanism which can be incorporated within existing systems without the need for unnecessary (though perhaps useful) multiple metering. This mechanism should perform data sanity checking in real-time, where one cannot afford to lose incorrect measurements, and where only a limited history is available for possible data comparison and sanitization.

To further illustrate this phenomenon, we will refer to actual data, measured in a real life setting. It is important to remember, however, that this is a case-in-point, typical of many other settings, and that the conclusions it provides are extendible and applicable to the general case. In fact, the solution proposed shall be proven independent of the specifics of the example. In order to present a sample of the general class of problems to be solved, an example of monitoring transformer winding temperature is presented in the following.

Typical of many other utilities in Virginia, the Electric Department of the City of Martinsville relies on remotely collected data for equipment monitoring, control, and maintenance. Among various other pieces of equipment, substation transformers are subject to constant monitoring given their susceptibility to aging, overloading practices, and natural phenomena such as lightning strikes. All of these factors may cause the oil inside of these transformers, used for cooling and insulation, to decompose and break down, resulting in equipment failure and possible explosion. The prime indicator of the onset of such an event is the temperature at the transformer windings. In a report by Aubin [10], it has been generally convened in the industry that hot-spot temperatures at the winding in excess of 130°C (168°F) produce gas bubbles in the insulating material (usually paper) and result in turn-to-turn faults and overall malfunction. On hot and sunny days, the personnel of the Martinsville Electric Department have resorted to fan and water cooling to alleviate the risk of accidents, and have thus far managed to maintain incident free operation. However, as equipment tolerance limits decrease (because equipment is sized closer to the load), and the manpower availability goes down, it becomes increasingly important to rely on remotely sensed data to determine the transformer temperature and status.

Originally, an attempt to predict the winding temperature from other variables in the context resulted in failure. Some of these other variables were very frequently bad and

displayed values that did not fit at all within the context, thus, basing a prediction on these erroneous inputs did only result in incorrect values for the prediction. The focus was then shifted from ensuring that one particular variable made sense to the more tedious task of maintaining consistency among all variables in the set, and ensuring that these variables always changed "in concert". Interestingly, this situation (having to change the research focus because of possible bad data measurements) occurs very frequently in academic and industrial settings.

3.2 Problem Formulation

Formally then, the objective is to develop a method which would monitor sets of related physical measurements, verifying the consistency of each set a whole. If a particular set fails the consistency check, the method would identify the offending variables that are out of context, and provide more sensible replacement values for each. Moreover, the notion of *consistency* or *context* would continuously change in order to track the variable relationships which may slowly change over time. The solution methodology should be able to work with existing system setups without requiring additional hardware installation such as duplicate or redundant metering equipment. By using the proposed technique alongside current practices and performing a subjective comparison of the outcomes during an initial testing period, the validity and reliability of the method can be readily established.

Conceptually, the solution approach relies on a basic idea. Let us picture a system which, originally given sets of data values, would memorize the *relationships* among these values with a certain degree of accuracy. Subsequently, this system would be presented with a new set of values, and it would determine whether the pattern or arrangement of values is a familiar one by comparing the resulting relationships with the ones memorized. Totally unfamiliar patterns are rejected, based on the assumption that data relationships only change slowly in time. Data sets that appear somewhat familiar are memorized, taking the place of the oldest relationship pattern encountered. In effect, this constitutes a sliding window over the whole collection of acceptable data relationships.

In the remainder of this chapter, we will provide real-life illustrative examples of remote measurement aberrations. The examples will help to guide in the design phase of the solution and in justifying related design decisions.

3.3 Data Abnormalities

3.3.1 Physical system setup

The data obtained for this experiment was collected by way of a SCADA system from the substation on *Aaron Street* in Martinsville, Virginia. The substation equipment consists of transformers, circuit breakers, and voltage regulators. Of interest to this experiment are the three transformers which are configured according to figure 3.1. As shown, the transformers form a Y connection, providing three phases for a 13.2 kV distribution line stepped down from a 34.4 kV transmission line. Each transformer has a capacity of 6667 kVA, amounting to a total 20,000 kVA output capacity. The outputs of the transformers are fed into three circuit breakers which provide electricity to different sections of town.

3.3.2 Description of the data collected

Appropriate sensors have been installed inside the transformers in order to monitor pertinent data. These sensors are polled by an on-line SCADA system, and hourly average readings are obtained and stored. The SCADA system is subsequently accessed through a phone connection, and the data transferred to the computer for processing and storage. Three quantities are measured for each transformer: the winding temperature at the core, the temperature of the oil used to cool the transformer, and the pressure of this cooling liquid. In addition, some common quantities are measured. These are the total power delivered to the three phases (measured before the circuit breakers), the power delivered through each breaker, and the ambient temperature, measured at the nearby *Watt Street* substation. Finally, each input record also contains the date and time at which the data was collected. These variables, along with their units and assigned names are summarized in table 3.1.

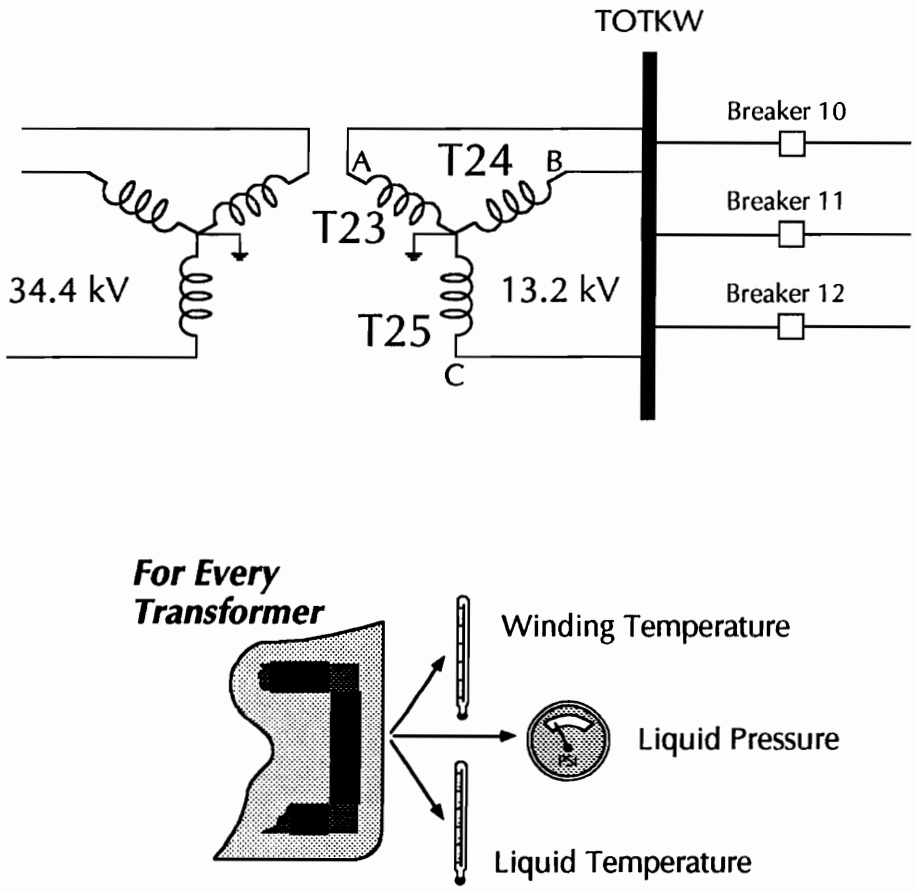


Figure 3.1. Physical setup for data collection site

A sample of the data collected is shown in figure 3.2. The three lines represent readings from three transformers. Observations of this and other samples reveal that the three transformer variables are very strongly correlated and exhibit similar qualitative behavior. Moreover, it appears that, in summer time, these variables are more sensitive to changes in the ambient temperature than to variations in load. This is also partly due to the fact that, during this period, the transformers are operated closer to their physical limitations. When the load was purposefully raised to a higher level (13MW versus the usual 8MW), winding temperature responded directly to sudden changes in the load, whereas changes in the liquid temperature and pressure were more gradual. This behavior is illustrated in figure 3.3.

Another anomaly that is worth investigating is the mismatch between the liquid pressure readings. Knowing that the transformers have the same specifications, including manufacturer and date of manufacturing, one might be tempted to attribute the mismatch to an imbalance in phases. If this were the case, however, the winding and oil temperatures would have displayed similar behavior for the particular transformer. A subsequent data set, shown in figure 3.4, displayed negative values for oil pressure, which clearly indicated a defective or miscalibrated sensor. The problem has since then been corrected as can be seen in the same figure. This case is practically almost identical to what happens when multiple meters are used to measure a quantity and their readings do not match with one another. Another example of aberrant data behavior, also apparent in figure 3.4, is the temporary malfunction of one or more sensors for a finite period of time. In this particular case, ambient temperature readings, which are taken at a nearby site, were not available for an entire day. In other instances, temperature is the only variable correctly measured, while in others, all data readings would be missing.

Table 3.1. Data collected from Martinsville

Variable Name	Description	Units	Range
AARO_TOTKW	Total 3-phase real power	KW	0-15,000
AARO_KW10	Power Delivered - Breaker 10	KW	0-5,000
AARO_KW11	Power Delivered - Breaker 11	KW	0-5,000
AARO_KW12	Power Delivered - Breaker 12	KW	0-5,000
AARO_WDT23	Winding Temperature - Transformer 23	°C	0-100
AARO_WDT24	Winding Temperature - Transformer 24	°C	0-100
AARO_WDT25	Winding Temperature - Transformer 25	°C	0-100
AARO_LQT23	Liquid Temperature - Transformer 23	°C	0-100
AARO_LQT24	Liquid Temperature - Transformer 24	°C	0-100
AARO_LQT25	Liquid Temperature - Transformer 25	°C	0-100
AARO_PSI23	Liquid Pressure - Transformer 23	PSI	0-5
AARO_PSI24	Liquid Pressure - Transformer 24	PSI	0-5
AARO_PSI25	Liquid Pressure - Transformer 25	PSI	0-5
WATT_TEMP	Ambient Temperature	°F	(-20)-120

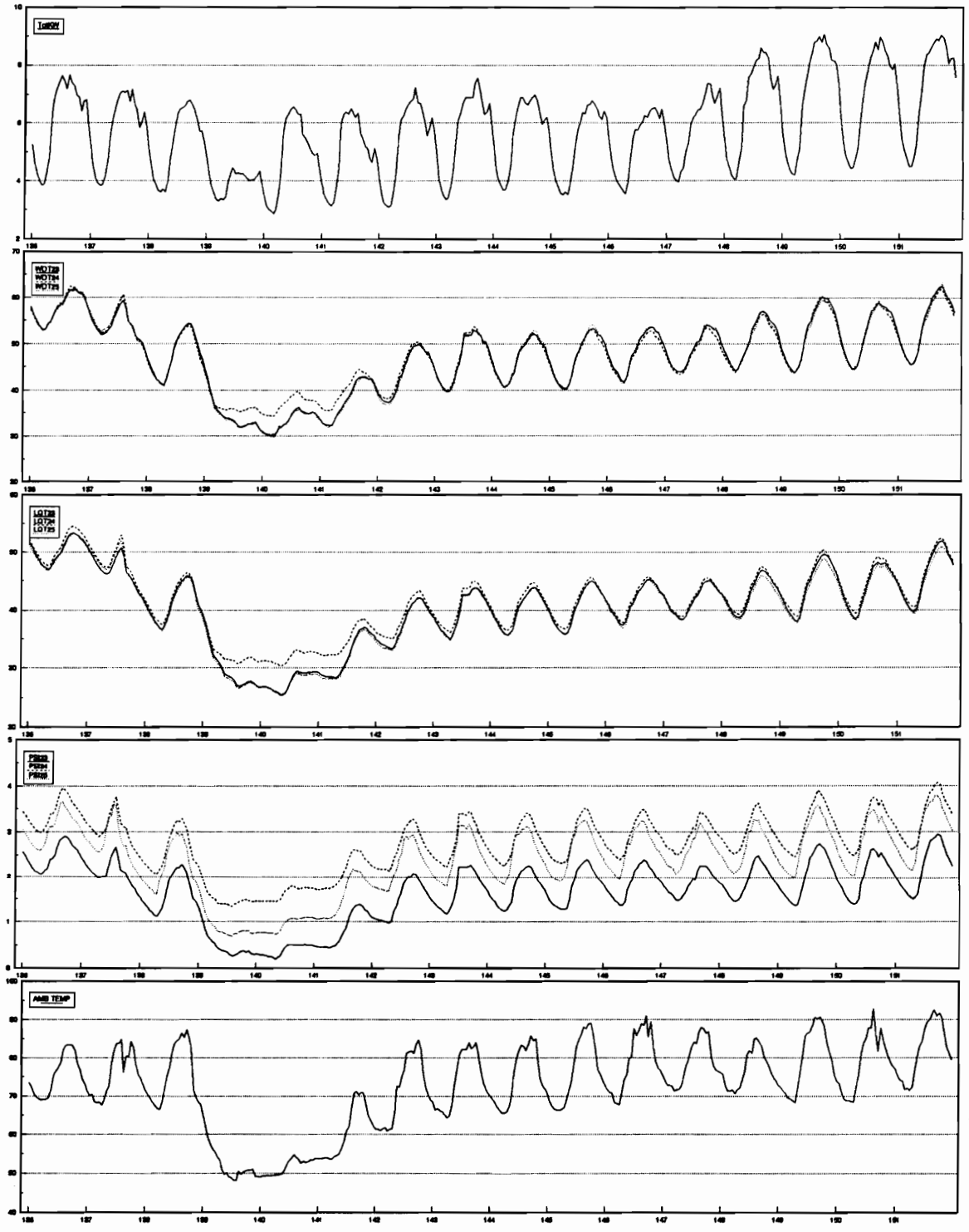


Figure 3.2. Sample data collected (May 16th-May 31st, 1991)

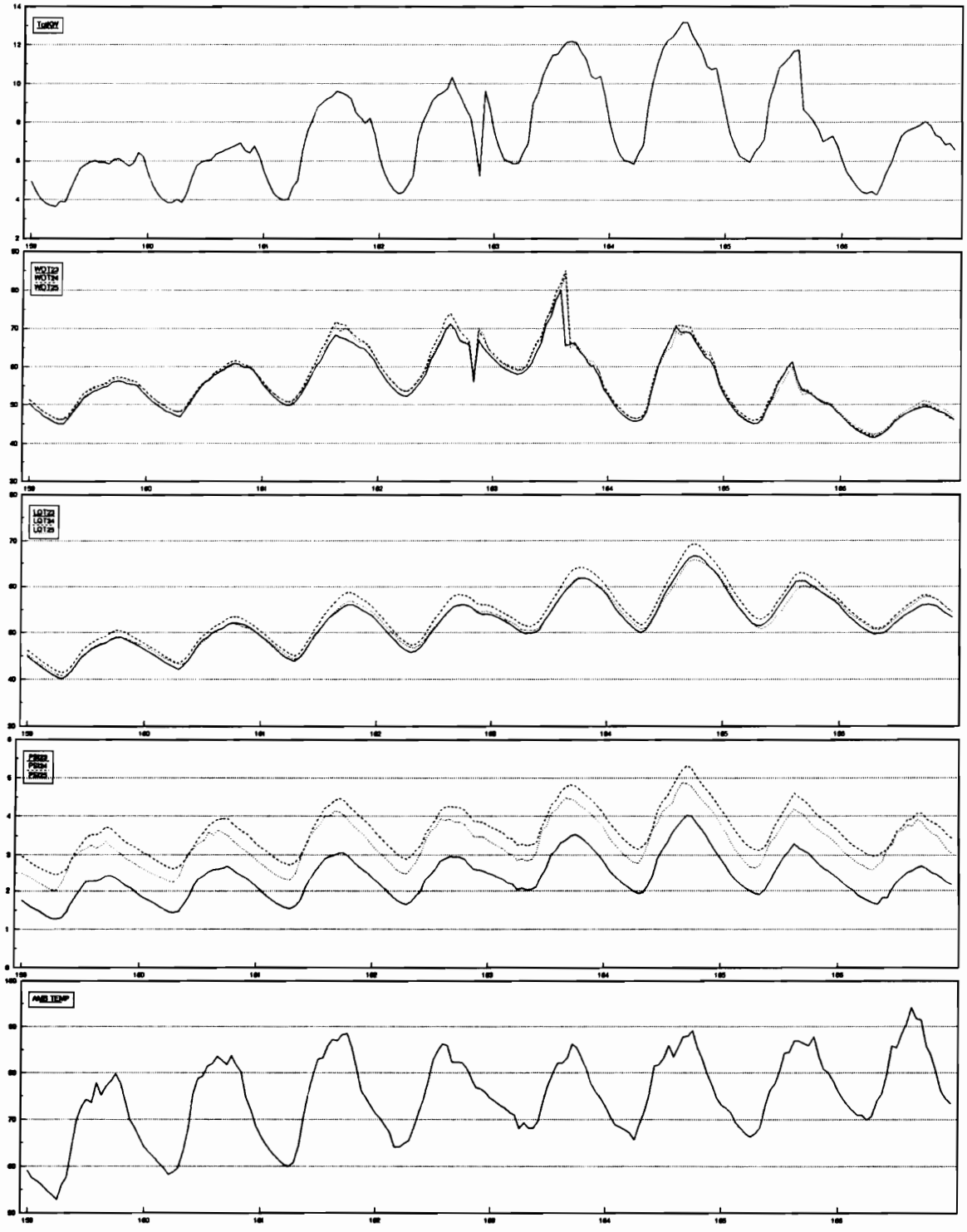


Figure 3.3. Behavior under high load (June 8th-June 15th, 1991)

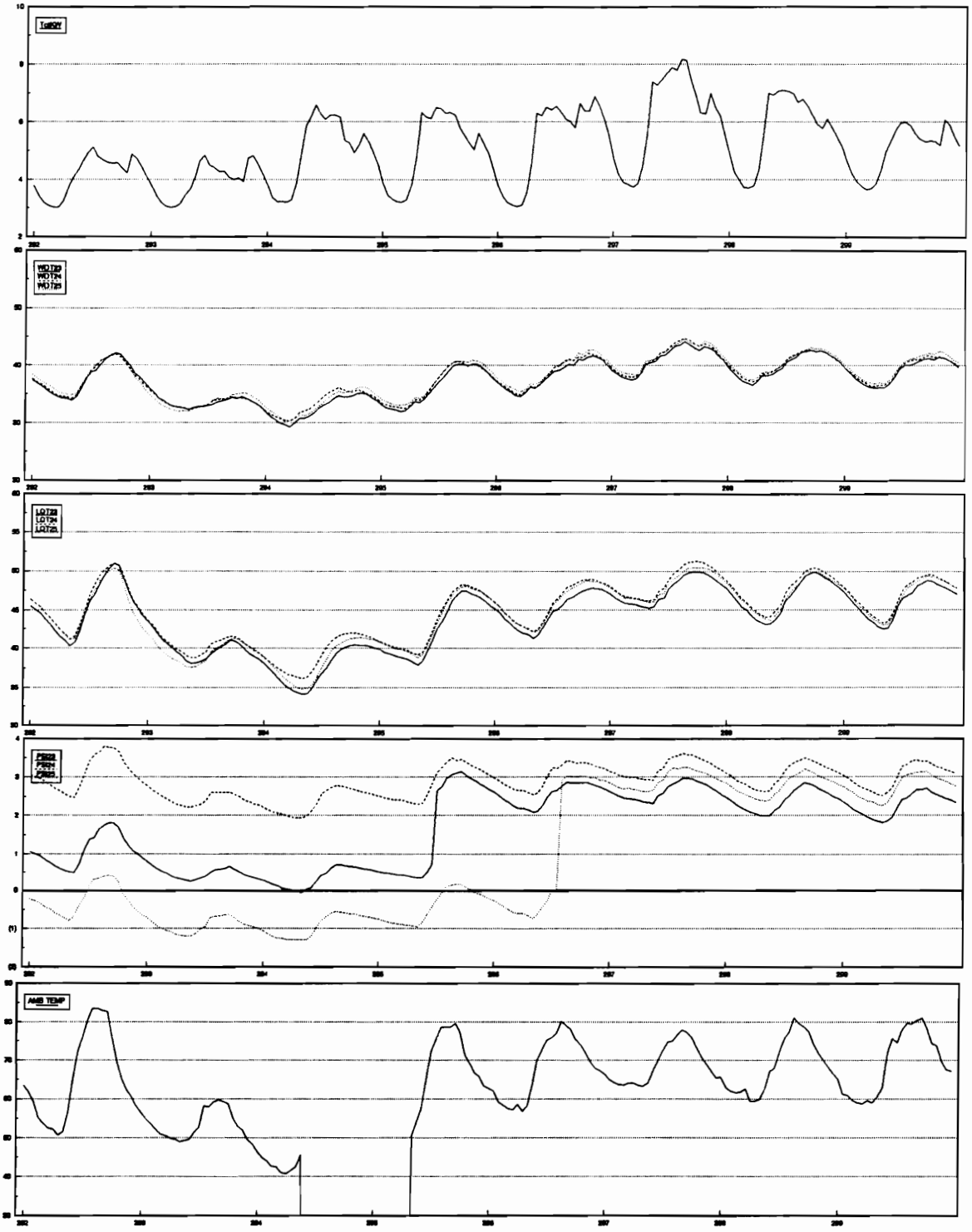


Figure 3.4. More data anomalies (October 19th-October 26th, 1991)

To summarize, the collected data exhibits the following properties:

1- The variables being measured are clearly related. We have, however, no explicit relationship, or mathematical equation, representing these quantities in terms of one another.

2- Moreover, the relationships linking these terms are apt to change in time, due to the changes in the ranges of certain variables (such as the seasonal range patterns displayed by ambient temperature).

3- Some large changes in the data are due to known physical system alterations (such as recalibration of the liquid pressure sensors for transformers 23 and 25). The date and time of these alterations are known to the operator.

4- Availability of all the variables is not always guaranteed. For short periods of time, some or all of the variables may be missing.

5- Visual inspection of data plots by an operator can indicate obvious malfunctions, although a continuous visual inspection is neither desired nor required.

3.4 Recapitulation

Various activities in the academic and industrial environments rely on remotely measured data for analysis and control. A large subclass of these measurements features, among others, the following properties:

- The elements of data collected form sets of related measurements.
- The relationships linking these elements are finite in number and are only slowly varying in time.
- Inspection of a sample of this data by an expert in order to determine the goodness of the sample is possible.

These characteristics of the data render it amenable to sanity checking in the context within which it is collected provided the correct approach is taken. This is the fundamental thesis of this research.

In what follows, we will lay a theoretical foundation over which we will base the proposed solution. We will show how the conceptual approach is formulated into a feasible solution which, in turn, is implemented. The example shown here will serve as a proof-of-concept, and the generality of the approach will be established.

Chapter IV

THEORETICAL FOUNDATION

4.1 Introduction

As will be seen in chapter V, the solution proposed for the data sanity problem will involve both an expert system and a neural network. In this chapter, we provide a brief introduction to expert systems and neural networks, concentrating on their respective advantages and shortcomings. This will be followed by an analysis of various 'hybrid' configurations which feature both expert systems and neural networks. The intent is to lay a foundation on which to base the various key decisions to be taken later in the design and implementation phases of this research.

4.2 Expert Systems

Expert systems have been in existence for the past two decades. Today, they may be artificial intelligence's most commercially viable branch. Even though the field is still relatively young, advances have resulted in a clear definition of the concepts involved. Moreover, design, building techniques and methods of expert systems are well established and well documented in the literature. Some notable sources that offer an in-depth treatment of the

subject of expert systems are the books by Addis [1], Forsyth and Rada [52], Jackson [76], Waterman [181], Weiss and Kulikowski [182], and the volume by Hayes-Roth et al [67] which organizes and relates the findings of more than 40 AI researchers in the field.

4.2.1 Definition

During the decade of the 80's, a multitude of books, introducing and describing expert systems, came to existence. The definition of expert systems offered in these books is usually a slight variation of the following. To quote Waterman [181], "an expert system is a computer program that uses expert knowledge to attain high levels of performance in a narrow problem area. These programs typically represent knowledge symbolically, examine and explain their reasoning processes, and address problem areas that require years of special training and education for humans to master."

Yet another way to characterize expert systems is to emphasize their differences from conventional programs, emanating from the fact that conventional programs manipulate data, while expert systems manipulate knowledge. This distinction has many implications, as described by Maher and Allen [104]. Their findings are summarized in table 4.1.

4.2.2 Components of an Expert System

An expert system is basically composed of two main modules, and several other support modules. The main modules are the inference engine and the knowledge base. Support modules include, among others, the user interface, and some tool to view and modify the system knowledge. A commercial scale expert system usually takes several man-years to build and is the result of cooperation of several agents. These are the domain expert, the knowledge engineer, and the system developer (which, in many cases, is the same as the knowledge engineer). The knowledge engineer interacts with the domain expert in order to collect the necessary knowledge and problem solving strategies. He decides on the format to be used in order to encode the acquired expertise. The system developer performs the actual coding and implementation of the expert system including the necessary support tools. During

Table 4.1. Differences between conventional programs and expert systems

Conventional Programs	Expert Systems
Representation and use of data	Representation and use of knowledge
Knowledge and control integrated	Knowledge and control separated
Algorithmic	Heurisitc
Repetitive process	Inferential process
Effective manipulation of large data bases	Effective manipulation of large knowledge bases
Programmer must ensure uniqueness and completeness	Knowledge engineer inevitably relaxes uniqueness and completeness restraint
Midrun explanation impossible	Midrun explanation desirable and achievable
Oriented toward numerical processing	Oriented toward symbolic processing

(after Maher and Allen)

the conception and implementation stages of the expert system, these individuals conjoin in order to refine and test the system. The final product is eventually put to use by an end-user. These various interactions are displayed in figure 4.1.

4.2.3 Types of Expert Systems

Expert systems are classified under several broad categories. The first classification results from the knowledge representation method used which will usually fit under one of the three following groups:

- Rule-based representation: The knowledge consists of production rules of the form: **if** *condition/premise* **then** *action/conclusion.*
- Frame-based representation: In this representation, features are associated with nodes representing concepts or objects. The features are described in terms of attribute/value pairs.
- Semantic net-based representation: The knowledge in this representation consists of a collection of nodes representing concepts or objects. These nodes are joined by arcs describing the relations that exist between the nodes.

The second classification steps from the type of inference used and the direction of the inference chain. There are two main groups under this classification:

- Backward chaining systems: These systems start off with a goal to prove, and decompose it into its constituent subgoals. They continue this process until they reach facts which either confirm or invalidate the results of the subgoals. The results are propagated back to assign a value to the original goal.
- Forward chaining systems: The inference process in these systems works by performing applicable rules on the collection of facts currently known to the system. The action of these rules is to add, delete, and modify facts from this 'world'. As a result, different rules become applicable and the process is repeated. Depending on the strategy being used,

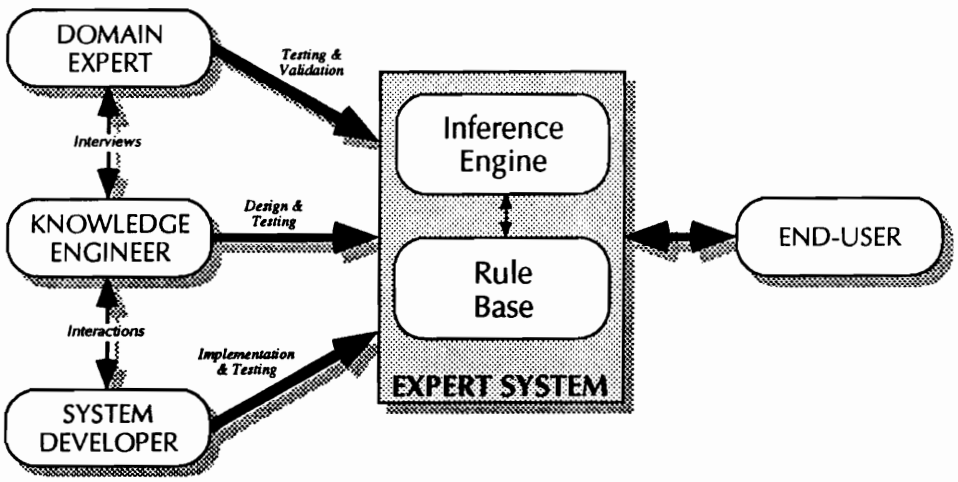


Figure 4.1. Expert system components

the 'world' may finally stabilize, yielding the necessary results, when no more rules are applicable. Alternately, the functions required of the system may be carried out implicitly in response to specific changes in the facts.

4.2.4 Advantages & Shortcomings

Being able to transform human knowledge into artificial expertise, expert systems offer several benefits which justify their importance and the need for their use. Several of these stem from the differences between human and machine expertise, which are enumerated by Kaplan [82] and are shown in table 4.2-a. Kaplan also shows the other face of the coin by listing the problems with machine expertise, which almost seem to be an extension of the benefits. These are shown in table 4.2-b.

Expert systems also feature several other drawbacks resulting from the symbolic nature of the knowledge representation. The first problem occurs during the process of soliciting the knowledge from the domain expert. To begin with, real experts in the domain are usually rare, and when they are available, they usually cannot devote the time required by a reasonable size expert system. Relying on several experts to avoid this dependence may result in problems, notably when the opinions given by two experts on the same matter are contradictory. Moreover, it is sometimes difficult for the domain expert to articulate his reasons for taking a given decision, or to formulate his thought process in arriving at a conclusion.

There are other problems originating mainly from knowledge representation. The knowledge engineer may discover, after months of system development and testing, that the representation originally chosen is not adequate to represent a new fact provided by the expert. The revision cycle will undoubtedly take considerable time, especially if the system is of a large size, wherein modification of some rules may cause some parts of the system to cease to be functional. The problems are then summarized in the difficulty of acquiring the knowledge, the difficulty in maintaining and updating it in order to ensure consistency, and finally the considerable development time and efforts required.

Table 4.2-a. Advantages of artificial over human expertise

Human Expertise	Artificial Expertise
Perishable	Permanent
Difficult to transfer	Easy to transfer
Difficult to document	Easy to document
Unpredictable	Consistent
Expensive	Affordable

(After Kaplan)

Table 4.2-b. Advantages of human over artificial expertise

Human Expertise	Artificial Expertise
Creative	Uninspired
Adaptive	Needs to be told
Sensory Experience	Symbolic input
Broad Focus	Narrow focus
Commonsense knowledge	Technical knowledge

(After Kaplan)

4.3 Neural Networks

4.3.1 Historical Overview

The human brain has long been a mystery and a challenge to scientific researchers. For nearly a century, neurobiologists have been trying to understand both the structure and function of the brain. Interest in cognitive processes and natural, as well as artificial, intelligence is nearly universal. Neural network evolution, as described by Levine [99], has experienced three waves of interest. The original inspiration that eventually led to neural network technology is due to Warren McCulloh and Walter Pitts who, in 1943, wrote a paper titled "A Logical Calculus of Ideas Imminent in Nervous Activity" [108]. Three fields emerged as a result of this paper: Digital computers, conceived by John Von Neumann; Artificial Intelligence and expert systems, attributed mainly to Marvin Minsky; and the perceptron, the grand father of today's neural networks, invented by Frank Rosenblatt.

Rosenblatt's perceptron [146] was an optical pattern recognition system designed after the retina. It was capable of some learning and identified several geometric patterns. In 1959, Bernard Widrow developed and adaptive linear element that he called *adaline* (**Ad**aptive **Li**near **Ne**uron) [186] which later evolved into the *madaline* or multiple adaline. Several applications were achieved using this element including speech recognition and weather prediction. The adaline algorithm was the basis of adaptive filters, which found several real-world uses.

Then in 1969, Marvin Minsky and Seymour Papert published a book called **Perceptrons** [110]. This book was an in-depth critique of Rosenblatt's perceptron with the main result that perceptrons could only solve problems that were linearly separable, and therefore could not model as simple a function as the XOR (Exclusive OR) function. The conclusion implied by this critique was that perceptrons were not interesting to study, or to sponsor, which signaled the end of the first wave of neural network progress.

Despite Minsky and Papert's claims, several researchers continued investigating the field of neural computing, notably James Anderson who built on the Hebbian principle [68] to develop a linear memory model called the *linear associator* [9]. The linear associator was later extended to model known as the *BSB* (brain-state-in-a-box) model. Around the same time

frame, Teuvo Kohonen was doing work on adaptive learning and associative memories. He later devised the principle of competitive learning which is the basis of unsupervised learning networks [87]. Another neural computing pioneer who persisted in the field in spite of the book **Perceptrons** was Stephen Grossberg. Grossberg was responsible for a major addition to neural knowledge, the "Adaptive Reasoning Theory" or *ART* [62]. ART networks have several desired properties such as short-term learning, immunity of the knowledge stored in the network from noisy data, and unsupervised learning.

The second wave of interest in neural computing resulted from a paper presented by John Hopfield to the National Academy of Sciences in 1982 [72]. This paper had an enormous impact on the community and restored belief in the domain. Funding, which had dried up for nearly two decades following Minsky and Papert's book, was once again flowing.

The third wave is fairly recent. Neural network conferences in the past two or three years have attracted crowds from several disciplines interested in the field: computer scientists, mathematicians, engineers, physicists, biologists, psychologists, cognitive scientists, social scientists and philosophers, among others. To date, several neural network types or paradigms exist. The most notable ones, collected from the work by Bailey and Thompson [12], Roth [147], and described in the manual by NeuralWare [121] are listed in table 4.3.

Since this field is relatively new to the academic environment, and given the recent wave of interest in applying neural networks to various engineering problems, the past years have witnessed the introduction of neural networks as a stand-alone course in several departments. Several books have also been written on the subject, notably the one by Khanna [84] and Kosko [88].

4.3.2 Definition and Description

As new neural network terms are introduced in this section, they will be displayed in *italics*. Several of the concepts in this section are adapted from a series of articles by Caudill in *AI Expert* [19-23].

Table 4.3. Neural network paradigms

1. Perceptron networks
2. Adaline and madaline networks
3. Brain-state-in-a-box (BSB) networks
4. Kohonen networks
5. Adaptive resonance theory (ART) networks
6. Hopfield networks
7. Back-propagation networks
8. Counter propagation networks
9. Outstar networks
10. Spatio-temporal pattern recognition networks
11. Bidirectional associative memory (BAM) networks
12. Recirculation networks
13. Hamming networks
14. Probabilistic neural networks
15. Boltzmann machines
16. Functional-link networks

Neural networks were inspired from the nervous system which is mainly composed of a multitude of intricately connected neurons. Each neuron has several input dendrites connected to other neurons through synaptic junctions. The neuron reacts to signals observed by the dendrites. If the combined signal exceeds a certain threshold value, the neuron fires through its axon by producing a chemical output signal that is transmitted to other neurons.

A neural network is composed of analogous computing elements known as *neurodes*. Neurodes are not meant to duplicate the physical characteristics of the biological neuron, but are rather designed to model its functionality. A typical neurode is shown in figure 4.2-a. It is equipped with several weighted input lines, a summing device to compute the weighted sum of the inputs, a *transfer function* that determines the firing criteria of the neurode, and an output path to propagate the output to other computing elements.

Neurodes are connected in *layers*. As shown in figure 4.2-b., a neural network typically consists of an input layer, a variable number of *hidden layers*, and an output layer. The size of the input and output layers are dictated by the number of inputs available and the number of outputs desired, respectively. Several parameters determine the complexity, behavior, and range of capabilities of the system:

- The number and size of hidden layers. Typically, one or two layers are used. The neurode count in these layers is subject to optimization. If the number is too large, the network has the tendency of memorizing input patterns rather than extracting and generalizing features, and thus causing the performance to degrade when unfamiliar patterns are encountered. Too small a number, on the other hand, reduces the capacity and the accuracy of recall of the network.
- The transfer function, or firing criterion for the individual processing elements. Variations on this function include (but are not limited to) a simple threshold value, a piecewise linear mapping, a sigmoid function, a hyperbolic tangent function, and a sine function. Some learning algorithms require more complex firing criteria such as the spatial Mexican sombrero function in Kohonen competitive networks .

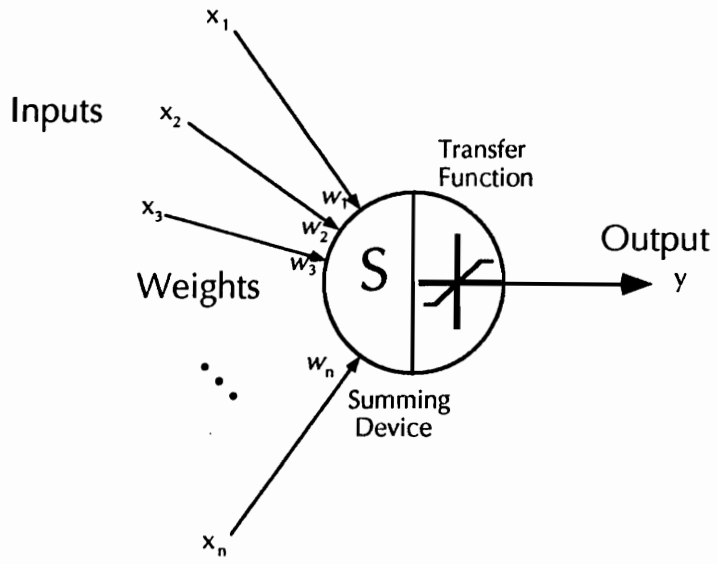


Figure 4.2-a. Neural network node

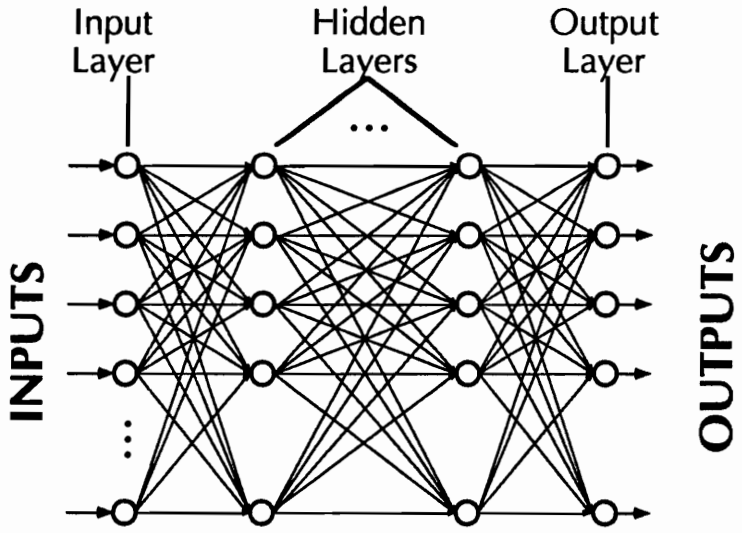


Figure 4.2-b. Typical neural network

- The *interconnections* between processing elements. Different categories of neural networks stem mostly from this architectural decision. Each neurode, or processing element in every layer of a neural network produces a single output. This output, in turn, is connected to one or more neurode inputs, or, if it belongs to the output layer, is fed out to the external world. Three independent-decisions have to be made at this point: whether the outputs of a given layer are randomly connected to the inputs of the following layer or whether they follow a specified pattern; whether the output of a neurode is fed as an input to neurodes in the same layer, as is the case in *competitive learning*; and whether a neurode's output is connected back as one of its inputs, thus forming a direct feedback loop.

- Last but not least is the *learning rule*. This is a formula or an algorithm which specifies how different *internal weights* of the network should be modified every time it is presented with an input. Subtle changes in this rule can significantly improve the speed, capacity, and performance of the network. Learning rules are what gives a neural network the capabilities of autonomous learning, self-improvement and automatic adaptability to changing environments. It is therefore natural that learning rule development and improvement has recently been an extremely active research domain.

The transfer function of a network, the learning rule, and the physical interconnection of different neurodes form what is referred to as a *paradigm*.

Neural networks generally operate in two modes, *learning* and *recall*. These modes are typically distinct, though they overlap in some networks. During learning, a network is trained. Three types of learning are possible: First, *supervised learning*, where the user has to provide the correct corresponding output for every input shown to the network. The network adjusts its internal weights based on the error computed from the generated and desired outputs. Second, *unsupervised learning*, where only input patterns are presented to the network. Third, *reinforcement learning*, where the user training the network indicates whether a given output is correct or incorrect. During the second phase, recall, the network simply computes an output based on the current input and performs no internal modifications. If the output pattern is designed to match the input pattern, then the network is said to be *auto-associative*. If, on the

other hand, the input and output are totally distinct, then the network is known as *hetero-associative.*

In a typical setting, a neural network would be started in the learning mode with all the internal weights set to random values. It is then presented with a number of input examples that may even range in the thousands. As more examples are made available, reliability and accuracy increase . Once sufficiently trained, the network is switched to recall mode where it usually performs classification, generalization or deduction in constant time.

4.3.3 Advantages

By design, neural networks are very different from conventional artificial intelligence tools such as expert systems. The differences are such that neither neural networks nor expert systems can replace one another. Instead, they tend to complement each other in solving various artificial intelligence problems. Neural networks are not adequate in such problems as precise numerical computations. However, they perform relatively well at such tasks as recognizing a person's face from a visual image even if facial expressions change. Incidentally, these characteristics match the way humans perform given similar tasks. Several key features that give neural networks an edge over conventional systems follow :

- ***Learning by example in real-time:*** In traditional expert systems, knowledge has to be explicitly provided to the system in the form of rules. In neural networks, however, the system acquires its knowledge by observation of examples. Learning is achieved through a learning rule which adjusts internal weights in the networks in response to the examples being shown. The obvious advantages of such an approach are: (i) a short development time, since network training can be carried out gradually in real time while the system is operational, (ii) and the ability to represent knowledge even when it cannot be easily or adequately expressed as a set of rules.
- ***Distributed memory and associative recall:*** Information in a neural network is stored as weights between computing elements inside the network. Every time new information is learned, the network adjusts one or several of the internal weights according to the

learning rule used. This mechanism allows the network to select the best matching output when presented with a partial input, a property which is extended to what is referred to as *generalization*, or the ability of the network to respond reasonably well when presented with incomplete, noisy, or previously unseen input.

- ***Fault tolerance and graceful degradation:*** A benefit of distributed memory is the fact that even if some individual computing elements within the network fail to operate, the behavior of the network as a whole is only slightly degraded. This is to be contrasted with traditional computing systems which would come to a screeching halt as a result of slight damage to memory. This feature makes neural computing systems extremely well-suited for applications such as nuclear power plant operation wherein total failure of control equipment can be disastrous.
- ***Real-time pattern recognition:*** Neural networks possess several inherent capabilities that make them more suitable for pattern recognition tasks than conventional statistically based and expert system based approaches. Neural networks are parallel in nature, they are adept at classification problems even when the input data is noisy, incomplete, or distorted. Furthermore, they display an ability to select and deduce combinations of features in the input data that are pertinent to the classification problem at hand. These traits match the requirements of pattern recognition tasks.
- ***Intelligent association and synthesis:*** Another aspect of neural networks that expands their domains of applicability is their ability to remember related items even when the relationships are not clear or obvious. Analogously, certain neural networks exhibit the ability to synthesize complex continuous functions and provide continuous intermediate values from sample outputs.

When neural networks are used as classifiers, different paradigms are best applicable based upon the type of input data and the training required. Lippman [100] offers guidelines for choosing the proper paradigm for the type of classification problem at hand. The resulting paradigm selection tree is reproduced in figure 4.3.

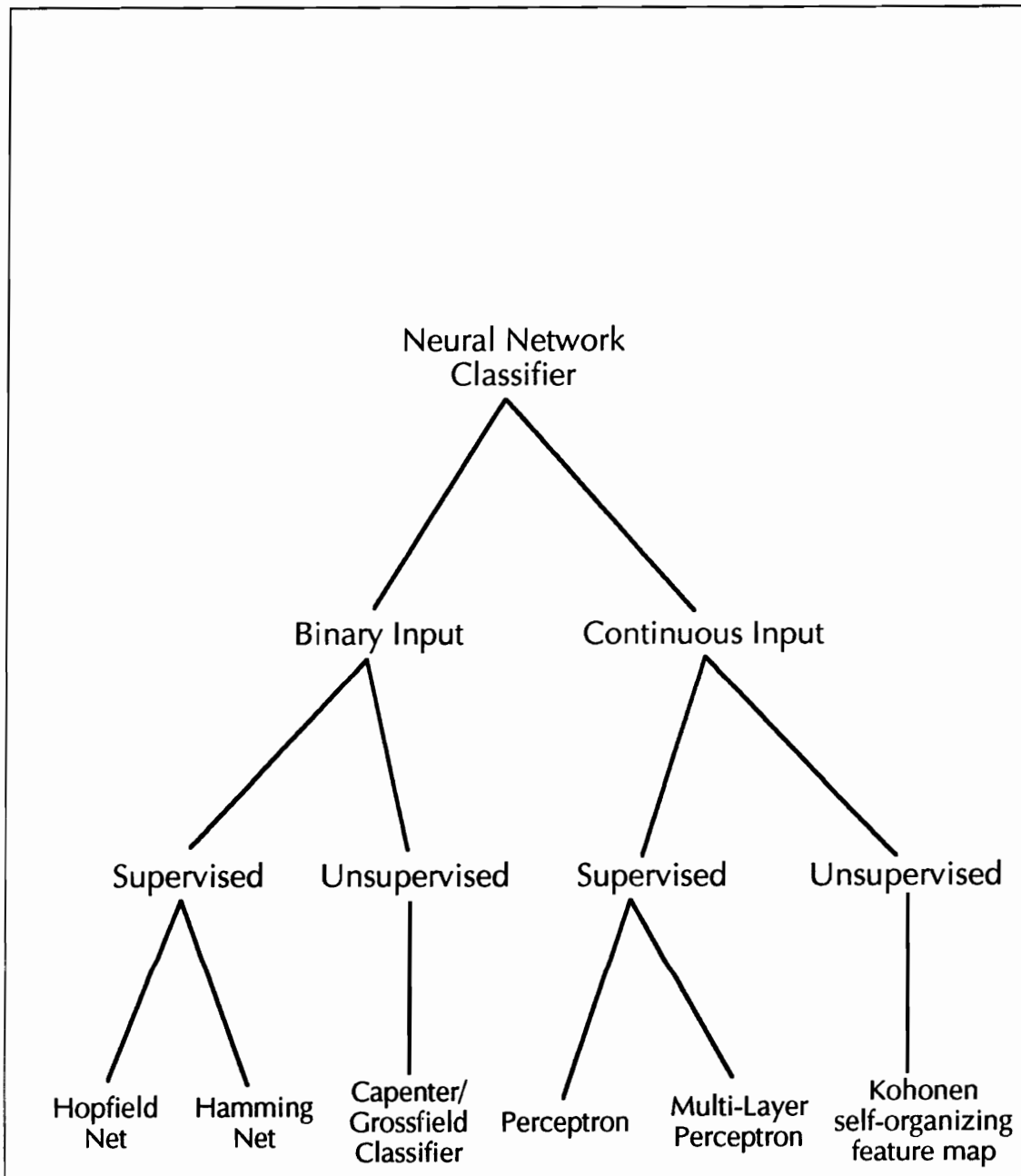


Figure 4.3. Selection tree for classifier paradigms

4.3.4 Shortcomings

Despite its numerous advantages, distributed memory causes a major flaw in neural networks. As we will see in a subsequent section, knowledge in a neural network is stored as a pattern of weights and connections. Unlike an expert system case, where the rules in the knowledge base can be easily displayed in an English-like fashion, or in the case of a conventional program, where the current values of the variables and the current statement being executed are usually sufficient to determine the state of knowledge of the program, knowledge in neural networks is unintelligible and cannot be easily viewed or modified. Weight changes are usually the compounded effect of several input patterns and cannot be easily traced back. Moreover, features extracted by a neural network do not usually, or necessarily, match the features identified by humans in the same input data. Since neural networks select their own classification features, merely displaying these features to the user would not provide an intuitive insight on how the network is operating or how it is reaching its conclusions. Consequently, explanation facilities are not simple or easy to implement in neurocomputing systems.

Another characteristic of neural networks that may present problems at times is the need for an ample supply of data for training. Inadequate amounts of data for training and testing can lead to an under-trained network that has simply memorized its limited examples and generalizes incorrectly when new data is encountered. This may be cause for concern since data collection can be costly and burdensome. Additionally, data is needed in order to build any knowledge in the system. While the task of building a simple if-then rule is relatively trivial when using a rule-based system, cloning this operation in a neural network environment can prove to be a difficult feat. Knowledge transfer to a neural network is consequently hard.

4.4 Hybrid Systems

It is interesting to notice that the 'good' and the 'bad' points of expert systems and neural networks are somewhat exclusive. These two technologies seem like good complements

to one another since the weaknesses of one seem to coincide with the strong points of the other. Thereupon, AI researchers have sought ways of combining these two methodologies to take advantage of the benefits of both and neutralize their weaknesses.

Several arrangements for combining expert systems and neural networks have been described in the literature, each specifically designed for a particular purpose. The samples presented here, meant to exemplify the possibilities rather than enumerate them, have been suggested by Caudill [24], Hillman [70] and Rich [118]. In each case, only one of the two methods acts as the main processor with the other method playing an auxiliary supporting role.

Figure 4.4-a displays the most obvious arrangement, sometimes referred to as **formal separation**, wherein the problem is broken down to several pieces, and each piece is solved by the more appropriate of the two methods. Variations on this approach are shown in figures 4.4-b and 4.4-c, respectively, and consist of adding a preprocessor which would examine each subproblem and decide whether is should be solved using the rule-based or the neural network-based approach. As shown, the processor may be an expert system, or a neural network. For this kind of arrangement, the main processor depends on the type of problem being solved.

In the system of figure 4.4-d, a neural network is embedded inside the expert system, and is used to perform fast rule selection for the expert system to test and execute.

The last class of combinations relies mainly on the neural network for processing. In figure 4.4-e, the neural network is sandwiched between two expert systems. The expert system providing input data to the neural network performs tasks such as data conversion and conditioning, user and database queries, and inference from previous data. The neural network then produces solutions and results. The output expert system is used for output decoding and interpretation, and result analysis. It also provides a feedback loop to the input expert system which may use the results of previous cases to improve its data collection capabilities. Finally, the arrangement shown in figure 4.4-f is used to overcome neural networks inability to explain its reasoning. A neural network is used to generate answers quickly and efficiently. If an explanation of the results is needed, the expert system will attempt to

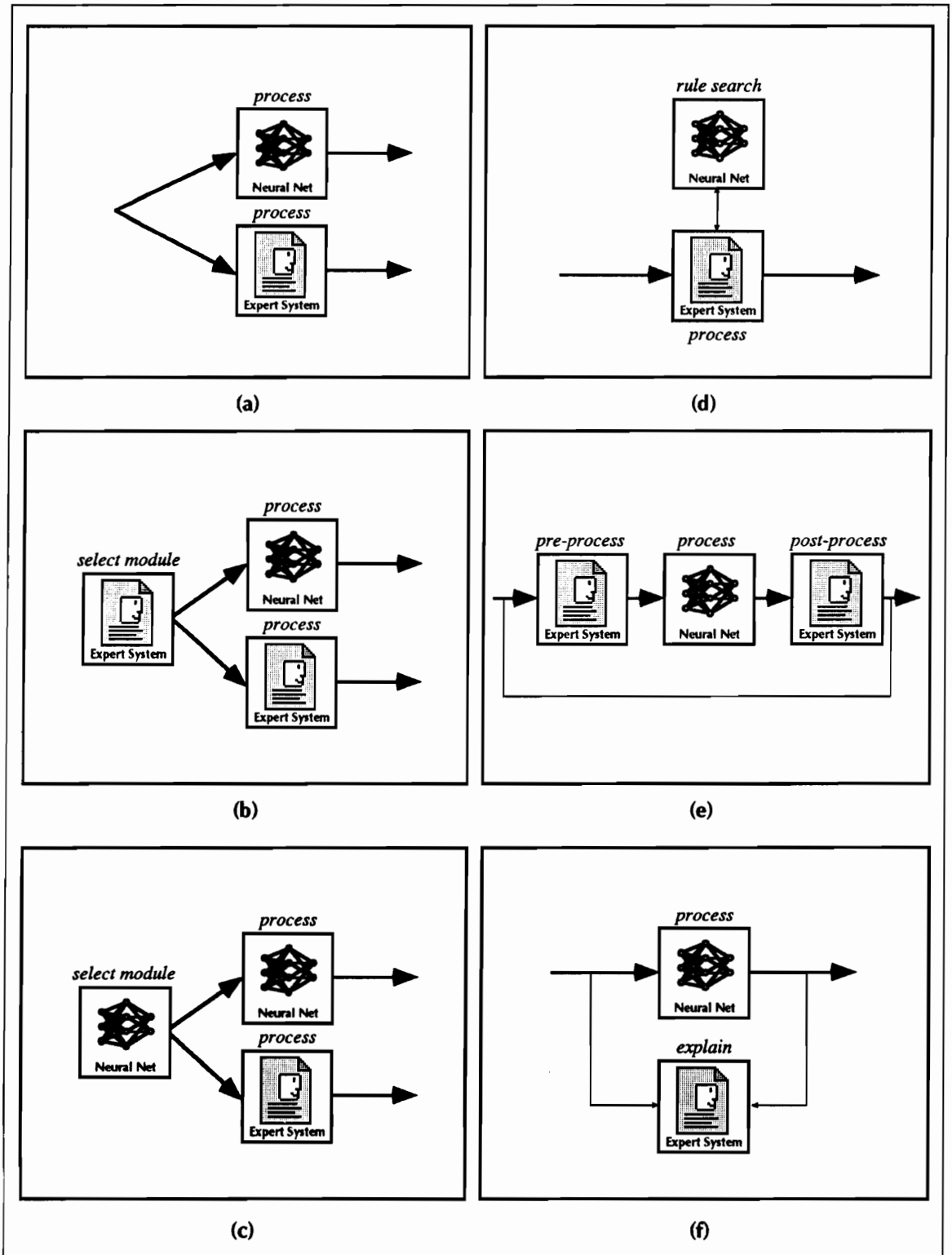


Figure 4.4. Hybrid system arrangements

backward chain from the answer obtained to the input pattern and formulate a plausible chain of reasoning in support of the network's results.

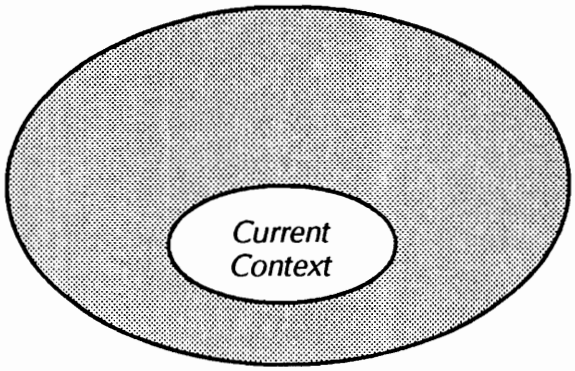
4.5 The 'Context' Concept

The approach presented in this dissertation is centered around a very important idea: the concept of ***context.*** A set of variables is measured *in context* if the data collection is performed in the same time frame and physical environment. Another requirement is that all the variables measured be directly or indirectly related. Figure 4.5-a shows an abstract representation of the space of values that a set of variables can take if unconstrained. Context limitations, however, restrict the span of values to a small subset identified in the figure as the 'current context'. As data relationships and ranges of values change in time (due, for instance, to seasonal variations), the current context slowly drifts and changes a given path, as shown in figure 4.5-b.

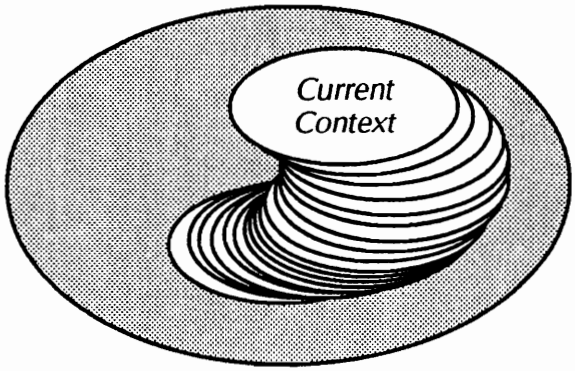
The above definition of context raises three issues: why and how is this idea useful in solving the data sanity checking problem; how to represent and 'capture' the current context; and finally, how to track this context as it changes slowly in time.

4.5.1 Context-based data sanity checking

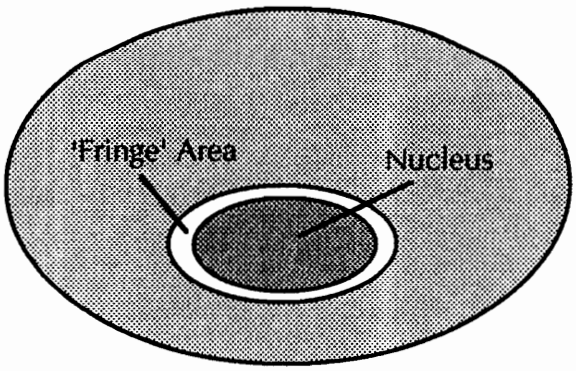
To address the first question, let us recall that the two main concerns in data sanity checking are the identification of bad readings, and, whenever possible, the provision of predictions, or replacement values, for individual variables. Let us suppose that the representation used to describe the context allows for a measure of 'error' or deviation from the center of the current context. This distance from the center can be used as an indicator of whether the new data measured is in concordance with previous, but recent, data. In fact, once the current context is defined, we can identify three different regions, as shown if figure 4.5-c. The center region, or nucleus, is the set of readings which are in harmony with the current context. The second oval permits tracking of the context. Readings falling in this range deviate only slightly from the current context and may be an indicator of a gradual change in



(a)



(b)



(c)

Figure 4.5. The concept of *'context'*

relationships. Readings which fall outside of this middle region are automatically classified as bad. These readings are out of context due to gross errors in one or more of the variables measured. The size of the two regions shown in figure 4.5-c is arbitrary and has to be determined (most likely in an experimental manner) for each type of data being checked. The size of the inner region depends on the random errors experienced with measurements, usually related to the quality of the meters used. The size of the middle region is governed by the rate of change of data relationships. If the context drifts very slowly, then very few readings will be out of context. If the context changes rather fast, then several readings will be attempting to push the edge of the inner region and to move it in a particular direction.

4.5.2 Context representation

An efficient context representation equipped with a distance measure from the center of the context will therefore allow us to easily identify bad readings simply by assessing their deviation from the current context. The representation we propose is based on neural network memorization and generalization capabilities. Each variable, or group of closely related variables will be the output of a neural network. The inputs to the network will be all the other variables in the context aside from those provided as outputs. In this manner, each variable will be predicted based on the other variables in the context, avoiding the bias introduced by using its current value as input.

Once all the required networks are fully trained, the current context will be contained in the values of the weights of the neural networks. The networks not only impose range restrictions given their memorization capabilities, they also indicate when variables violate their usual relationships, even though they may be in range. Neural network training is equivalent to reducing the error between the actual and predicted outputs. This error, automatically computed during training and recall, can be an excellent measure of deviation from the center of the currently established context. Furthermore, since a separate error is computed for each individual output variable, the distance, or deviation from the center, can be obtained for all elements of a reading, and the offending variables singled out and possibly replaced. Next, we will address the issue of tracking the context which is achieved using an updatable on-line training database.

4.5.3 Context tracking

The current context is stored in the neural network weights. The neural networks need to be trained with valid data, a process through which the internal weights settle to given values, and the prediction error is reduced below a preset level. To allow context tracking, the original data, which is used to set the initial context, is a stored in a training database. As new data is collected, readings which fall within the two inner regions of figure 4.5-c are added to the training database. The neural networks automatically train in order to maintain the error over the whole training database lower than some preset value. Since the new readings, or patterns, are only slightly different from the ones already encountered, incremental training will require a relatively low number of iterations, and the current context will be maintained with modest processing requirements.

The training database has a fixed size and is designed such that, once full, new patterns replace the oldest patterns, which should conceptually be moving out of context. Several important parameters which are crucial to the successful operation of the system have to be determined experimentally. First is the size, or depth, of the training database. This parameter directly controls the size of the nucleus in figure 5.4-c. Notice that, at any point in time, the current context is stored in the weights of the networks, while a sample of the target context is stored in the training database. The networks continually train until these contexts are the same. Consequently, if the training database is too small, the nucleus would be narrow, resulting in valid readings to be classified as bad. If, on the other hand, the training database is too large, the nucleus would be very broad, and certain bad readings may not be caught. In chapter VI, we will outline procedures that will help in selecting an appropriate size for the training database.

Directly related to the size of the training database and the size of the nucleus is the capacity of the neural network. The number of hidden layers and the size of each layer in a neural network determine its memorization as well as generalization capabilities. A balance must be achieved since a large network tends to simply memorize the patterns encountered, while too small a network may not even converge if its capacity is inadequate. Another aspect of the neural network that is of concern is the training time required to adjust the weights of the

network to a new context. Obviously, this time will increase as the network size is increased. A procedure that will help in selecting the appropriate network size will again be described in chapter VI.

4.6 The Algorithm

Now that we have defined the 'context' and selected a method to represent it and maintain it in an efficient manner, we will describe the procedure required to carry out context-based data sanity checking. The elements required at this level consist of a training database to guide the path of the context, appropriate neural networks which provide predictions for all the variables of interest, and some mechanism to control the flow of operations. As shown in the next chapter, the tool selected to carry out this latter task is a rule-based system.

The algorithm, which is summarized in the flow charts of figure 4.6, proceeds as follows:

- In a pre-training stage, form initial context in the networks as well as in the training database by performing incremental training. Incremental training requires that patterns be added to the database one at a time. All networks have to fully converge before a new pattern is added. Provided the data is time dependent and presented in chronological order, this procedure provides a faster overall training time than what would be required if all initial patterns are presented to the networks at once.
- The context is now established: error detection and correction can be initiated. For each new reading, compute error, or deviation from current context.
- If the reading falls in the nucleus, then the new data is totally in context. This new reading is considered correct and is added to the database to enforce the current context.

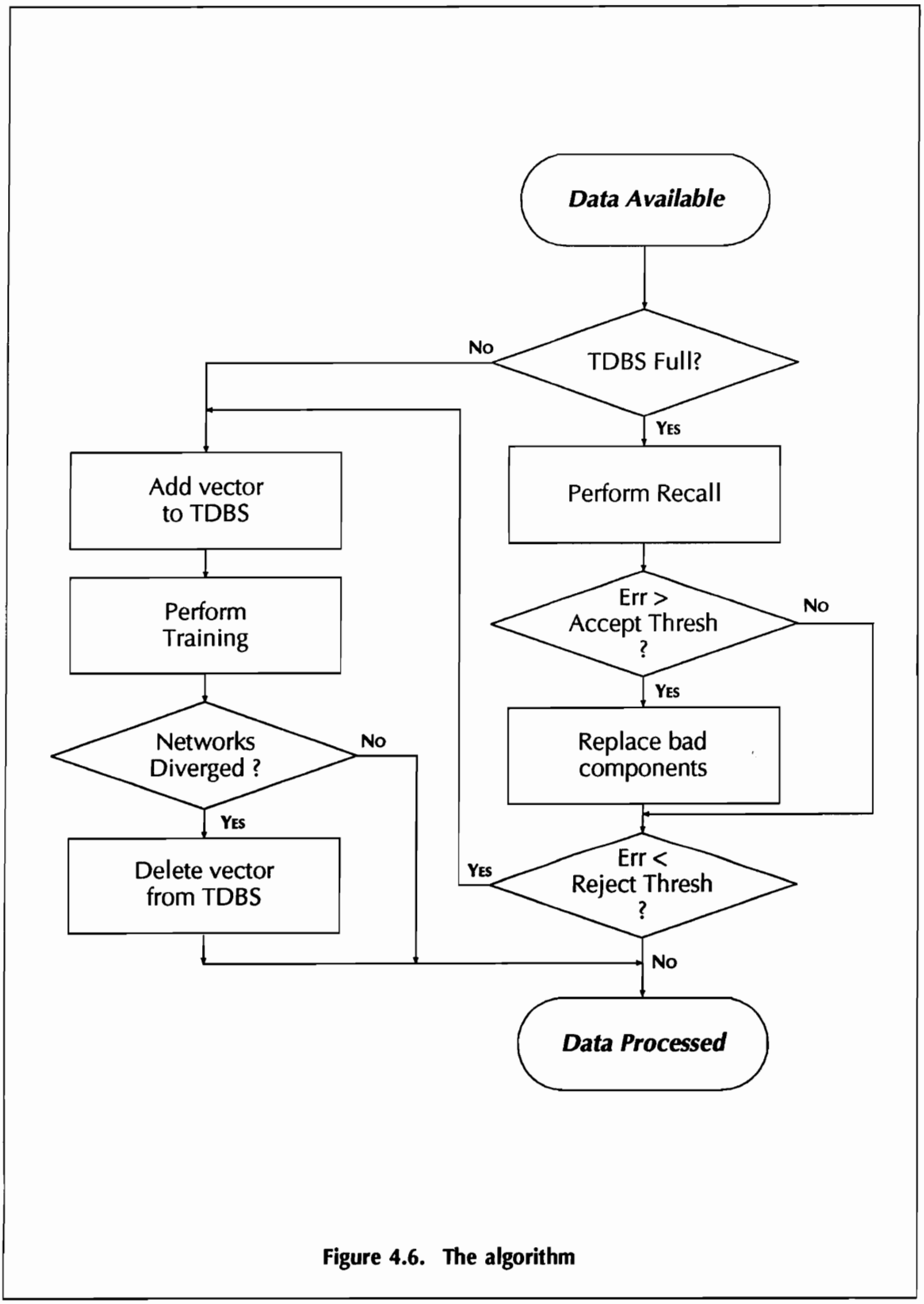


Figure 4.6. The algorithm

- If the reading falls in the 'fringe' area (or second oval), it may be a bad reading, or it may be an indication that the context is shifting. To account for both possibilities, two measures are taken. First, the particular variables in this reading which fall outside of the nucleus are replaced by predictions. This action is not drastic even if the reading was actually correct, given the relatively small size of the inner regions. The error introduced would be very small. The second action is to add the new reading to the training database. If the context is actually shifting in the direction of this new reading, similar neighboring readings will be collected, which will force the nucleus to move in the particular direction. On the other hand, if the reading was stray, its effect on the current context will be negligible.
- Finally, if the reading falls outside these two inner areas, it is considered as bad. Variables from the reading which display high errors are replaced by predictions.

4.7 Summary

This chapter offered a bird's eye view of expert systems, neural networks, and the different ways to combine them. The gist of these descriptions is the discussion of benefits versus shortcomings offered by each technology. These arguments will be used in chapter V, system design description, to justify design decisions. The concept of 'context' and its usefulness to data sanity checking was also presented. A method of representation which facilitates the task of context tracking was described, along with an algorithm to perform the task of context-based data sanity checking.

Chapter V

SYSTEM DESCRIPTION

5.1 Introduction

Based upon the problem objectives, it is necessary that the conceptual solution have to following properties:

- Memorization abilities
- Autonomous learning by example
- Handling of special cases
- Fast, limited history processing abilities
- Friendly user interaction

These capabilities, as seen in the previous chapter, are offered by hybrid systems provided the proper topology is used.

In this chapter, we will present the solution proposed for the data sanity checking problem. The solution is composed of several modules which will be treated in turn. A full description of every module, as it was designed and implemented, is given. The resulting system, entitled **DASANEX** (**DA**ta **SAN**ity **EX**pert system), offers a proof-of-concept of the

algorithm, and is usable on measurements that share the characteristics of the data in the Martinsville example. Though the system is not complete enough to handle all data sanity checking problems, careful consideration has been accorded to modularity and open architecture throughout the design phase. Consequently, independent modules can easily be added in order to augment the functionality of the system and its ability to process data with different characteristics. The proper procedure to add modules will be outlined in a later section.

In the sections that follow, the expressions 'data sanity checking' and 'filtering' will be used interchangeably.

5.2 System Block Diagram

The solution we propose is based on a configuration suggested by Hillman [70]. The architecture, depicted in figure 5.1, features both a neural network and an expert system. Such a design offers numerous benefits as outlined in chapter IV. The thick arrows in the figure outline the path taken by the data during processing by the system. This is not the physical path that the data takes, but rather the sequence of handlers of the data as it passes through. The data is initiated at the input module which, in real time, monitors a physical input port and receives the data whenever it is ready. In simulation mode, this module periodically obtains data from a designated input file and makes it available to the system. This data is then passed to the expert system which performs special case handling as well as input data conversion for the neural network by scaling and delaying. The bulk of the data processing is carried out by the neutral network which features attributes such as quick response time, short training period, and generality of application regardless of the type of data being processed. Once neural network estimates for the variables of interest are available, the expert system resumes control as a post-processor, scaling the outputs to their respective ranges, taking appropriate actions according to data significance, controlling the neural network training process, and passing the results obtained to the output module for archival. This completes the data

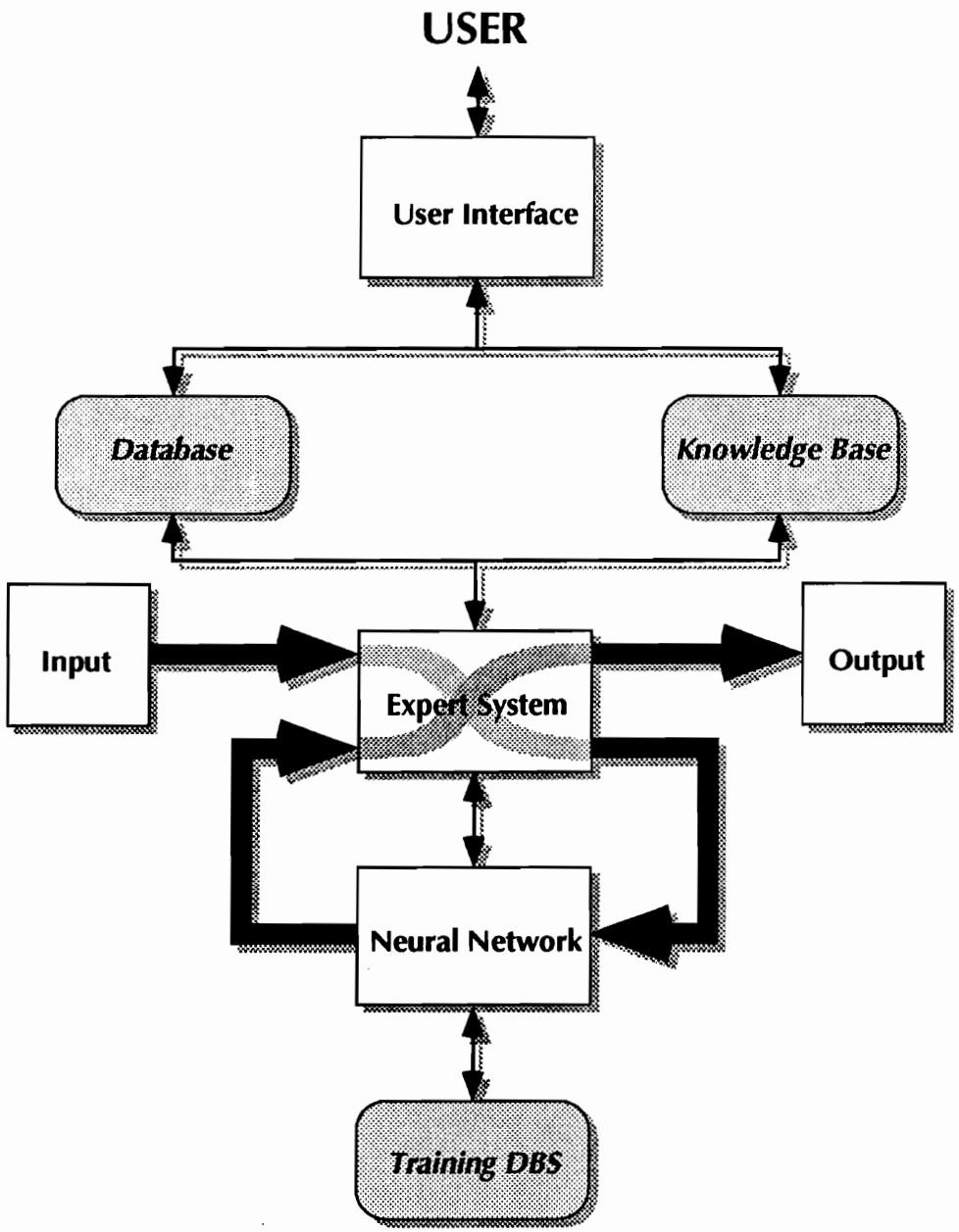


Figure 5.1. DASANEX block diagram

processing path which will be used for timing purposes. Some of the input vectors which qualify as candidates for neural network training are also routed to the training database module where they are stored until the neural network accesses them for retraining.

5.3 Data Managers

Several of the modules in the Data Sanity Checking system do not contain executable code, but rather serve as data storage and retrieval units. The knowledge base is an example of such a module. To ensure consistent and organized accessing and updating of the information contained within these modules, each data file is linked with a manager which handles data lookup and modification requests. The modularity of the manager concept offers several advantages such as consistency and generalization. Restricted access to the data through a single set of entry points guarantees that this data will always be up to date and consistent across all the modules that require it. Generalization stems from the fact that a given manager can be replaced by a full-fledged external package if the need arises. The database module would, for example, be replaced by a commercially available relational database.

Each manager is designed to respond to a standard set of operations as well as operations that are data or manager dependent. Common operations include:

- **READ**: Loads the appropriate information from a disk based file to memory. Necessary data structures are built or updated in response to this operation to allow fast and efficient access of data in memory.
- **WRITE**: Writes the information in memory back to file. The file format and the level of detail included are such that the information can be read back and the internal data structures rebuilt in an efficient manner.
- **ADD**: Adds a record of information to the memory copy of the given file. The new record is not actually written to the file until a **WRITE** request is issued.
- **DELETE**: Removes a record of information from the memory copy of the file.
- **LOOKUP**: Returns a copy of a record of information, usually identified by its ID number.

- **MODIFY**: Updates the memory copy of a given record of information.
- **TERMINATE**: Instructs the file manager to perform any cleanup action necessary before program termination. The manager maintains a set of status flags for various conditions, such as whether the data in memory matches the information on disk or not. These flags are properly updated upon completion of each of the operations requested of the manager. Part of the cleanup procedure for each manager, as an example, is to check the **DIRTY** status flag (marking whether memory matches disk) and to rewrite the information to disk in case of mismatch.

  This concept is depicted in figure 5.2.

5.4 Multitasking Aspects

5.4.1 Threads

In a multitasking environment, the operating system runs multiple processes in a concurrent fashion, either by making use of multiple processors, if available in the hardware, or by allotting time slices to all active processes. There exist many classifications for processes, such as foreground versus background. Of interest to us is the "task" versus "thread" classification. The main difference between a task and a thread is that each task is assigned its own virtual memory space, whereas threads are spawned by a main task with which they share memory. Both arrangements have advantages and shortcomings. Tasks need some form of inter-process communication mechanism such as pipes to share any amount of information, a procedure which can hinder the speed and independence advantages gained by multitasking. Threads have direct access to all the memory they share, but problems may arise if more than one thread attempts to modify the same variable or memory location at the same time, which is quite possible since the threads run concurrently. The common fix to this problem, described here as implemented in the operating system design for the NeXT computer [122], consists of assigning locks to the variables that are susceptible to this sort of anomaly.

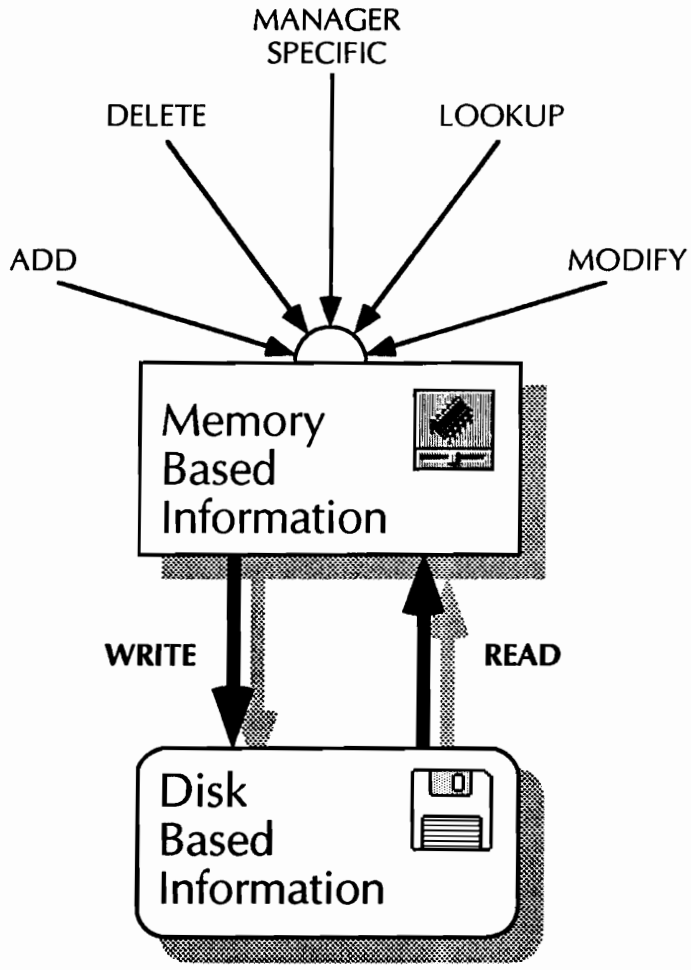


Figure 5.2. The data manager concept

Prior to modifying any shared variable, a thread would ***lock*** the variable's assigned mutual exclusion lock (also known as a ***mutex***) thereby ensuring exclusive access, and would ***unlock*** it after usage. A thread trying to lock an already locked variable would ***block*** (the process is put to sleep and the thread is put on a waiting queue) until the lock is released. Locks can also be assigned to groups of variables, resulting in a variable level of granularity. Great care should be taken in designing and assigning mutual exclusiveness locks to avoid the pitfalls of deadlock, which would occur if a program section locks a variable or resource then invokes some other program section that attempts to lock the same variable or resource. The above concepts are illustrated in figure 5.3.

Concurrent processes are desirable and justified when one or more of the following conditions prevails:

- The process or module generates or responds to asynchronous events. The concurrent approach to this type of handlers is more general than interrupts since these are usually restricted to hardware generated events. It is also more responsive than the common practice of periodic checks in a main event loop, which is very characteristic of conventional single task software.
- The process or module can continue execution after it requests a service from some other module without waiting for a reply. The module whose services are requested notifies the calling process when the action is completed.

In the case of the Data Sanity Checking system, speed is of a premium, whereas memory requirements are modest. Threads are therefore more appropriate than independent tasks. Several of the modules in the system qualify to be separate threads by virtue of their function. These are the input manager, the expert system, the neural network, the output manager, and the user interface. The justification for the implementation of a module as a thread will be provided in appropriate sections.

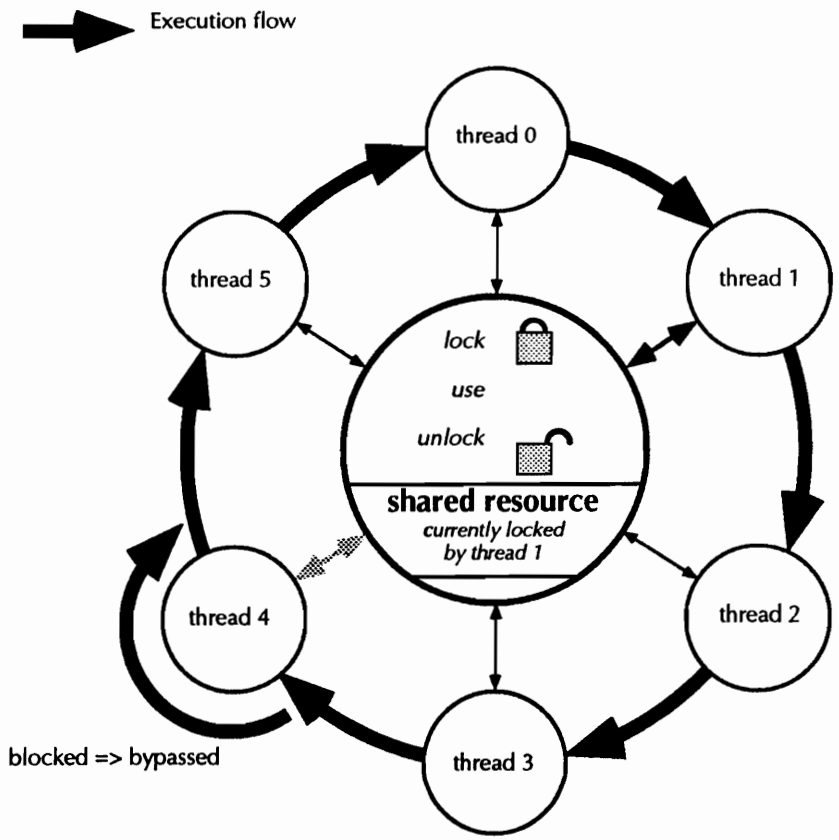


Figure 5.3. Multitasking and the thread concept

5.4.2 Message Passing Mechanism

From the outset, the Data Sanity Checking system will consist of several threads running concurrently, executing their preassigned tasks. Shared resource and memory access will be controlled by a set of locks at the appropriate levels. The threads will occasionally need to request services from one another, causing the normal execution flow of the threads involved to be temporarily modified. For example, the Neural Network could be in training mode, attempting to adapt to the data in the training database, when the Expert System receives a new set of inputs and prepares them for the recall or prediction operation. The Expert System would notify the Neural Network of the operation requested, and the latter would temporarily interrupt training to honor the recall request, then resume training once recall results are obtained. As another example, when the user chooses to quit the program, the User Interface module needs to alert all running threads of the decision so that they get a chance to perform cleanup routines such as closing any open files, saving any modifications, and terminating normally.

Hence, there is a need for a mechanism to pass requests between threads and modules in an asynchronous, orderly, and efficient manner. The message handling module was designed for these purposes. It consists of a message queue to hold the different requests, and a set of routines to add, retrieve, and delete messages from this queue. Since these routines are shared resources, they are all controlled by one lock to prevent unwanted situations, such as a thread deleting a message while another thread is attempting to use the information contained within, from arising.

A module requesting some service from another module posts a message. Messages are appended to the end of the queue and contain the following information:

- The ID of the module where the message originated (*source*). For this purpose, each module in the system is assigned a unique identification number. ID numbers are bit distinct and can be grouped together to form a mask that would still identify all the members in the group.

- The ID of the intended recipient(s) of the message (*destination*). General messages, such as the **TERMINATE** message can have multiple recipients at once.
- The nature of the service requested. The Knowledge Base manager can, for example, request the User Interface to put up an error panel stating that a given rule does not compile correctly.
- Message type information, used by the modules to respond correctly to the messages. For instance, a message can be marked as *personal*, in which case the recipient module is responsible for its deletion from the queue. A message can also require an acknowledge, either positive or negative. Similarly, a message can require the receiving module to signal receipt of the message by removing its ID from the recipient field.
- Additional information, which is message dependent, and which usually indicates the location where data is to be stored or retrieved.

Figure 5.4-a summarizes the information contained in a single element of the message queue while figure 5.4-b provides a sample instance of the queue during program execution.

All threads have a main loop in which the message queue is queried for messages for the particular module. If a message is received, its request is honored, then the appropriate notification is performed, if applicable. Otherwise the thread resumes its normal operation according to its function.

5.4.3 System Blackboard

This unit designates the section of shared memory that is constantly accessed by the various threads. Each thread requiring to look up or modify this information needs to lock the appropriate lock assigned to the variable prior to its usage, then subsequently unlock it when it is through utilizing it. The various resources and memory elements which feature lock mechanisms are described below:

- **MSG_LIST**: This data structure represents the list of messages being currently passed between modules. Part of the main loop of each thread involves querying this structure

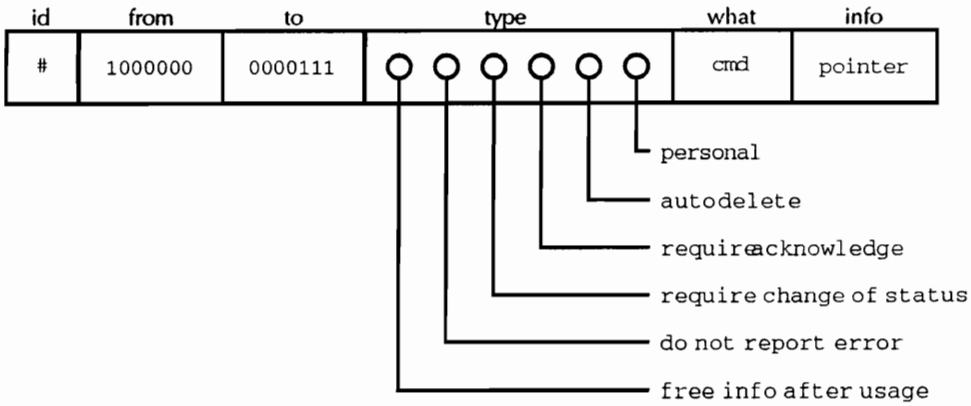


Figure 5.4-a. MSG_LIST components

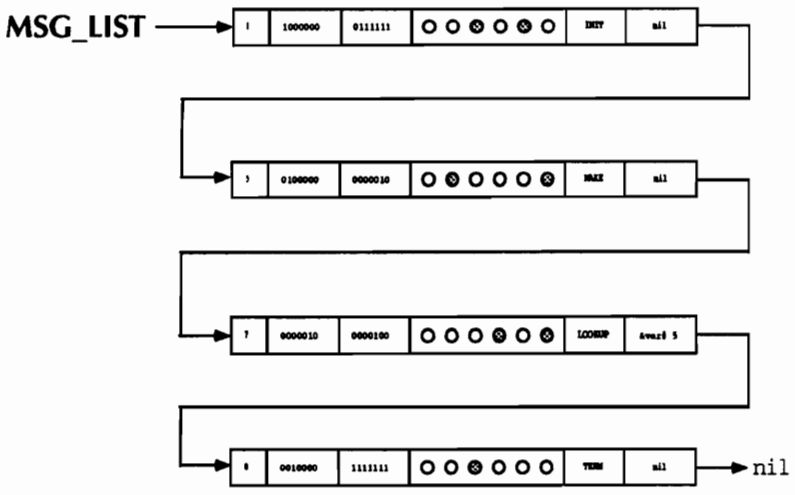


Figure 5.4-b. Sample message list

for messages intended for the particular module or thread. Access to this data structure is controlled through the mutual exclusion lock **msg_lock**.

- **GSTAT**: The status of the system at all times is reflected in a global status variable, **GSTAT**. This is a bit mapped structure consisting of several fields. Various system operations are enabled or disabled according to the values in this structure. The fields are enumerated below:
  - **dbs_in_memory**: This field reflects whether the database has been loaded from the disk-based file to memory. For instance, the user cannot check the rules in the knowledge base for correctness unless the database variables, which are sure to appear in the text of the rules, have been loaded in memory.
  - **kbs_in_memory**: Same information as above, but for the knowledge base.
  - **tdbs_in_memory**: Same information as above, but for the training database.
  - **dbs_dirty**: This flag is used to indicate that the database information contained in memory is different from that stored on disk. The user is subsequently prompted to save the changes to the database before he terminates the program.
  - **kbs_dirty**: Same information as above, but for the knowledge base.
  - **tdbs_dirty**: Same information as above, but for the training database.
  - **training**: This flag is set and cleared by the system, once the user has enabled training, to indicate whether some or none of the neural networks are currently undergoing training.
  - **running**: This flag reflects whether the system is currently in setup or run mode. In setup mode, the user is allowed to modify the database variables as well as the rules in the knowledge base. During run mode however, all editing capabilities are disabled, though the user is still capable of viewing the information.
  - **paused**: During a run cycle, especially in simulation, it is sometimes desirable to stop the execution of the program temporarily. In this paused mode, the user can examine the

state of the various internal components of the system such as the message list, the rule evaluation list, or the weights of a given neural network.

- **training_enabled**: This flag is set and cleared in response to user manually modifying the training menu. No network training is performed unless this flag is set.

- **RUN_MASK**: When a thread is initiated, it announces its presence to the rest of the modules by setting a designated identification bit in this variable. Analogously, the thread clears the bit prior to exiting (in response to a **TERMINATE** message, for example). Access to this variable is controlled by **run_lock**.

- **Error Handler**: This set of routines provides the service of error reporting to the user. A window announcing the particular error to the user is displayed on the screen. Access to this service can only be granted to one thread at a time. Therefore, mutual exclusion is guaranteed by the lock **err_lock**.

- **Data Managers**: Access to the various data managers is implemented as library calls. Since this practice does not preclude the possibility of several threads accessing the same piece of information simultaneously, each manager has an assigned lock that will prevent this event from occurring. By appropriately locking and unlocking its associated mutex, each manager will ensure that information is accessed only by one thread at a time. The mutexes assigned to the managers are **dbs_man_lock**, **kbs_man_lock**, and **tdbs_man_lock**.

- **Standard Output Stream**: Given that DASANEX is still at the development stage, some modules may require to output debugging information to the terminal screen. Access synchronization for this resource is controlled by **print_lock**.

This completes the list of resources and variables shared in the system blackboard.

5.5 Portability Considerations

A piece of code is considered *portable* if it displays the same behavior and produces the same results when compiled on two different platforms with no modifications to the code (barring minor changes such as compilation flags). ANSI implementations of the C language

promote portability by providing conditional compilation and machine independent standard libraries. In general, the user interface of any program imposes a compromise between user friendliness and portability. Program aesthetics and efficient user interaction benefit from the hardware capabilities of the target machine, such as a high resolution graphics screen, or a pointing device. This machine dependence, however, prevents portability for obvious reasons.

To work around this problem, we rely on the modularity of the design of DASANEX. Wherever possible, modules are coded with portability in mind, avoiding the use of any special features, tricks, or shortcuts provided by the development environment. Machine dependence is limited to a single module, namely the user interface. This module takes full advantage of all the features of the target machine with no restrictions, knowing that a different version of the module will be completely recoded for every target machine. It was important to take this decision early in the design process in order to maintain the resulting conventions.

A second capability that DASANEX currently requires of its target machines is multitasking (or multi-threading). This is needed in order to achieve module independence and high throughput. In a single task environment, the software would have to be slightly modified to simulate time-sharing and concurrent module execution.

5.6 Module Description

In the following sections, we will offer a description of the function of each module and provide in-depth details about its implementation.

5.6.1 The Database

The DBS manager handles data access requests to the database. The database holds all problem specific variables that the different modules require in order to complete the data transfer path.

- *The DBS memory format*

Once the database is loaded in memory, the following amount of information is retained about each variable, as illustrated by figure 5.5:

- An identification number (mainly for internal purposes).
- A symbolic name to be used in rules and status reports.
- A one line description of the variable.
- The type of the variable. Currently supported types are integers, double precision floating point numbers, characters, strings, one dimensional arrays of numbers, and finally, neural networks.
- The value of the variable, which obviously is type dependent. For integers, characters, and strings, this field holds the actual value of the variable. For double precision numbers, arrays, and neural networks, this field points to an area of memory where information about the variable is stored. These are as follows:
  - type **DOUBLE**: three quantities are retained. The current value of the variable, the minimum possible value, and the maximum possible. The last two are needed in order to scale the variable for processing by the neural network.
  - type **VECTOR**: the area pointed to contains the number of elements in the vector, and the actual elements of the vector.
  - type **NETWORK**: this information is paradigm dependent and consists of the information required for training and recall to be carried out successfully with the given network variable.
- A general purpose bit-mapped status word.
- A list of rules to consider for execution whenever the variable is modified. This list is created after the system successfully compiles the knowledge base, and is therefore not stored in the file copy of the database.

A global array, **VAR_LIST**, is used to hold all the database information.

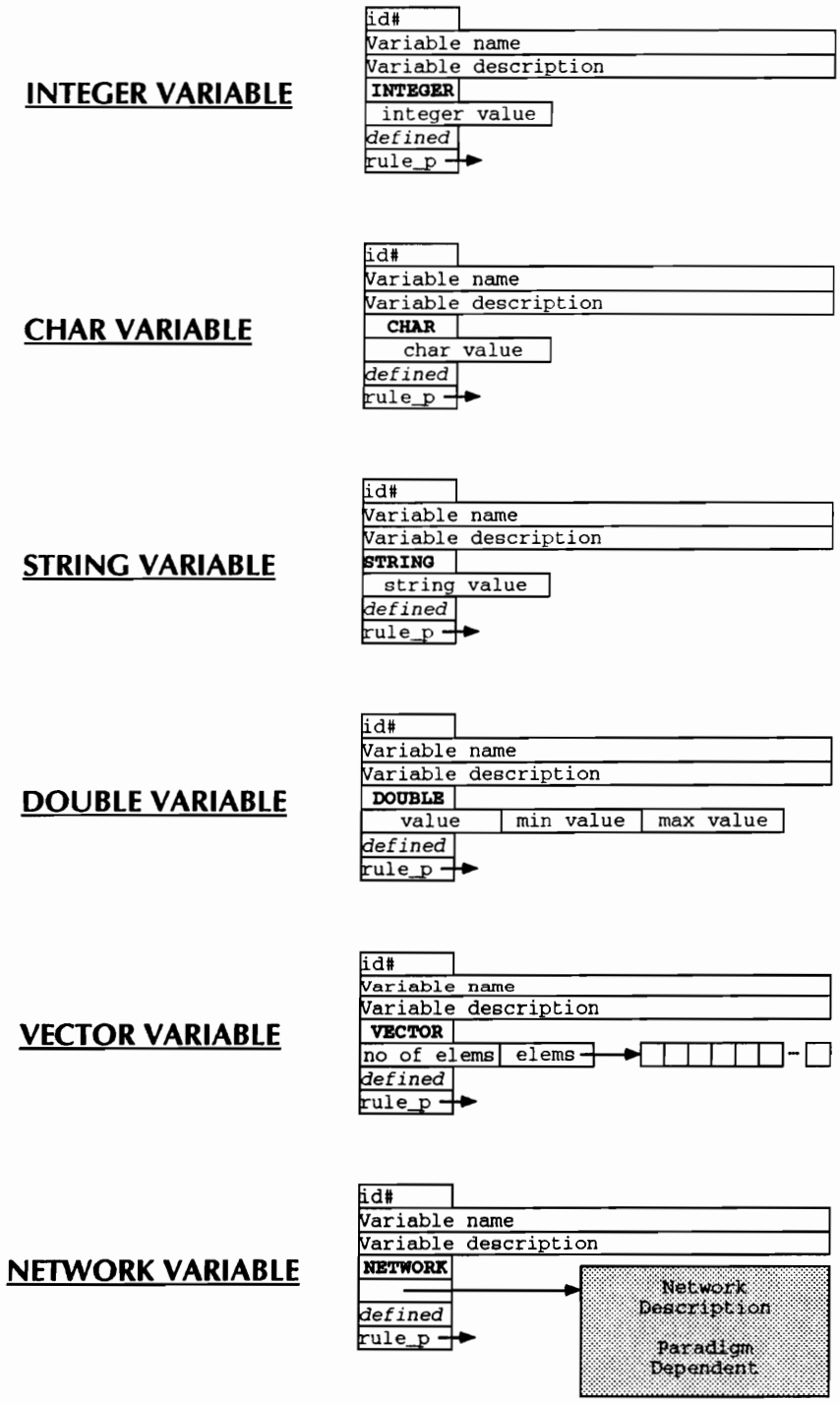


Figure 5.5. Memory format of database variables

- *The DBS file format*

The format used to store database variables on disk is very similar to the memory format. The prime concern behind this decision is the ease and speed of reading and writing to file. The database file is a simple ASCII file containing one piece of information per line. Figure 5.6, extracted from the header of every database file produced by the system, describes this format.

- *Operations specific to the DBS manager*

The DBS manager responds to several non-standard operations, all of which are variations on the lookup and modify operations. The rationale behind these services is that the DBS manager has access to all variables at once and can perform global operations. A module attempting to perform these semi-complex operations using the basic lookup and modify primitives would waste too much time on the primitives themselves. Though non-standard, the added operations are easily performed using the query features of commercial databases:

- **BYNAME:** Performs a lookup for a given variable using the symbolic name of the variable (as opposed to its ID number) to identify it. This is mainly used during compilation of rules to bind variable names encountered in the rule text to their corresponding identification numbers.
- **ZAP_RULE_LIST:** Prior to compiling the rules in the knowledge base, the KBES will request from the database manager, through this operation, to clear the rule lists in all the variable structures. As every rule is compiled, its ID will then be inserted in the list of every variable it references. The KBES cannot perform this task while compiling since variables will be encountered in a random fashion.
- **NET_GET:** This and the following operations rely on the ability of the database to access its elements globally. This particular operation returns a list of all the neural network variables that are judged to be *trainable* (Please refer to Neural Network section).

```
; Database for DASANEX
; Format for this file
; o  All lines starting with ; are ignored
; o  other lines describe variables.  Each variable requires 5 or more lines
; -  line 1 is the variable name   (up to 25 chars - no spaces)
; -  line 2 is a one-line description of the variable
; -  line 3 is the variable status (General Purpose HEX WORD)
; -  line 4 is the variable type
;             ([C]har [I]nt [D]ouble [S]tring [V]ector [N]etwork)
; -  lines 5 and beyond hold the variable value  (Type dependent)
;    == type 'DOUBLE' line 5 also contains VMIN & VMAX
;    == type 'VECTOR' No of Elems, Elem Values
;    == type 'NETWORK'
;      No of Layers
;      No of Nodes per Layer
;      variable name for input/output vector
;      index list of INPUT  nodes
;      index list of OUTPUT nodes
;      list of names for the output variables, one per line
;      Iteration_Count Max_error Average_error Std_Deviation
;      Sum of errors for last period     Sum of errors squared
;      (conditional) Max_Iter Max_Err Max_Ave_Err Max_Std_Dev
;      network status
;      if status says NETWORK_ALLOCATED, then weights of all nodes as follows
;            from_layer from_node to_node weight
;            -1 0 0 0 on the last line
;
;-----------------------------------------------
```

Examples

```
;---------------------------------------------------------------------------
pretrain_flag
Are we running or simply creating a Pretraining Database File?
8000
integer
1
;---------------------------------------------------------------------
pred_vec
Returned by Neural Network
8000
vector
10 0.3781 0.4150 0.4368 0.4254 0.5069 0.5213 0.5112 0.4746 0.6259 0.5775
```

Figure 5.6. Database disk format

- **NET_PREPARE:** This operation prepares all neural networks for training and recall by ensuring that they have been properly initialized. This operation is needed following user editing actions (ex: modifying the size of the network).
- **NET_RANDOMIZE:** This operation randomizes the internal weights of all the networks in the database. Alternately, the user can edit and randomize the desired networks individually.
- **NET_ENABLE:** This is another *batch* operation to enable the training flag of all neural networks in the database.
- **NET_DISABLE:** Similarly, this batch operation disables the training flag.

In addition, after updating the value of a variable in response to a **MODIFY** request, the database manager inserts the rules affected by the variable in the rule execution list by order of priority. These rules will later be tested and executed by the KBES.

5.6.2 The Knowledge Base Manager

The KBS manager is the main handler for the knowledge base, which contains all the rules pertinent to the data being processed.

- *The KBS memory format*

Rules stored in memory contain the following information

- The rule identification number (for internal purposes).
- The rule priority, an integer ranging between -10 and 10.
- A descriptive paragraph of the rule for the system administrator's convenience.
- The actual text of the rule, in a C-like language. This text, consisting of an antecedent and a consequent part, gets compiled to an internal representation before the Expert System can start evaluating and executing rules.
- Internal code, resulting from compiling the antecedent of the rule.

- Code resulting from compiling the consequent of the rule.
- A general status word.

The internal code, representing the system's conversion of the textual rule format, is created after successful compilation of the rules and is used by the KBES to execute the applicable rules. The exact conversion format will be shown in the KBES section. The memory copy of the rules is stored in the global array **RULE_LIST**. Figure 5.7 portrays an example of a rule in memory.

- *The KBS file format*

The file format for rules is very similar to their equivalent memory format. All fields are copied to the ASCII KBS file except for the code interpretation of the antecedent and the consequent parts of the rule. The header of every KBS file, shown in figure 5.8, reminds of the format of that file.

- *Operations specific to the KBS manager*

The KBS manager responds to several non-standard operations due to the nature of the information it manages:

- **MAKE**: Requests that all rules, or only the ones that have been modified, be compiled into the internal knowledge representation. This request is issued by the Expert System before a run session is initiated, or manually by the user, to check the correctness of his rules.

The following set of operations is dictated by the User Interface module. Because of the comment and text fields in a rule which could be of arbitrary length depending on the rule's function and complexity, the user interface presents rules to the user on a one-at-a-time basis, hence these operations:

- **FIRST_DEFINED**: Returns the first rule defined. This operation and the following enable the user to review the knowledge contained in the KBS.

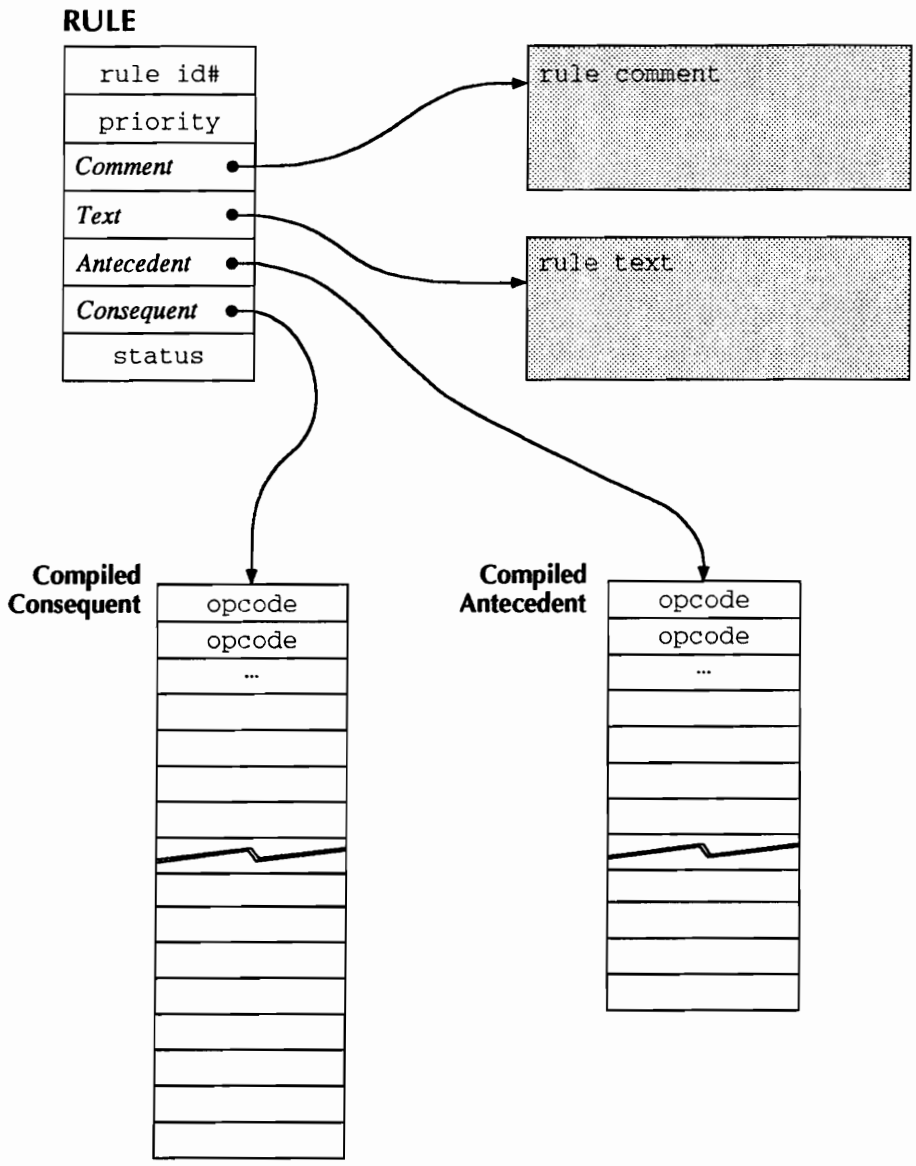


Figure 5.7. Knowledge base memory format

```
; Knowledge Base file for DASANEX
; Format for this file
; o  All initial lines starting with ; are ignored
; o  other lines describe rules.
; -  rule description starts with #I <rule-id>
; -  then a variable number of comment lines
; -  rule priority line #P <rule priority>  (default 0)
; -  rule text start with a line containing #T
;     followed by the C like representation of the rule on a variable number
;     of lines (free format)
; -  rule description is terminated with a line containing #E
;
```

Example

```
#I 50
Progress Rule
#P 5
#T
if((read_count % 50) == 0)
{
  print_value(read_count);
}
#E
```

Figure 5.8. Knowledge base disk format

- **NEXT_DEFINED:** Returns the next rule defined relative to the current one, thus allowing the user to step through all the rules sequentially without concern for their ID numbers (which may not necessarily be consecutive).
- **FIRST_EMPTY:** Returns the lowest ID number that isn't in use by a rule. A new rule can be added in this position.
- **NEXT_EMPTY:** Returns the first empty slot after the current one.

5.6.3 The Training Database Manager

The TDBS manager is invoked by the Expert System to add and delete training vectors, by the Neural Network to access these vectors for training, and by the User Interface to obtain statistics.

- *The TDBS memory format*

As will be described in the neural network section, the networks in this implementation obtain their input and output vectors from the elements of a single I/O vector. Before performing training or recall on a given vector, information included in the network description is used to extract separate input / output vectors from a single designated vector. The TDBS simply stores this vector in memory, along with its number of elements, and a incremental count of the instances where this particular vector resulted in a maximum training error for any of the networks in the database. This count is used to indicate possible outliers in the training database that should be avoided during training.

- *The TDBS file format*

The TDBS file format is a mirror image of the information contained in memory. Vectors are stored in binary form for quick and efficient storage and retrieval. This format differs from that used in the other managers due to the fact that inspection of the training database by the user is meaningless. Instead, the system provides adequate statistics about the contents of this training database.

- *Operations specific to the TDBS manager*

The TDBS manager responds to two non-standard operations. These are used by the neural network during training and during collection of statistics.

- **GET_RAND**: Returns a random vector from the training database.
- **GET_SEQ**: Returns training database vectors sequentially. Subsequent calls update the current position of the vector to be returned which wraps around the end of the training database, therefore implementing circular memory.

5.6.4 The Knowledge Based Expert System

- *Functions of the KBES*

The Knowledge Based Expert System fulfills several functions. These include pre-transformation of the data for input to the Neural Network, triggering of the recall and training operations, and post-transformation and analysis of the resulting outputs. The preprocessing stage is immediately followed by a Neural Network based stage. Neural networks, by nature, accept only numerical data. Additionally, this data has to fall within a certain fixed range (usually 0 to 1 or -1 to 1). Input data that does not fall in this range has to be transformed by means of translations and scaling before being processed, and the reverse operations have to be applied to the resulting output data. Non-numerical data has to be assigned suitable numbers which would provide unique representations for the possible input patterns. After processing, the output results have to be interpreted and classified back in the original input categories. Using this technique, a degree of fuzzy knowledge representation can be achieved (a text variable may for example have the values: COLD, COOL, MILD, WARM and HOT).

- *The rule list*

The versatility of the KBES is possible as a result of the forward chaining nature of this system. Changes in data cause rules to be fired and executed, resulting in more data changes, and triggering more rules. The inference engine of this system continuously evaluates and

executes rules until the system settles and no more data changes occur. The rules handled by the inference engine consist of two parts, an antecedent and a consequent. The antecedent is an expression that evaluates to **TRUE** or **FALSE**. The consequent is a sequence of actions of three types, variable modification, internal procedure invocation, and external program invocation. Internal procedures are built into the system and include actions such as causing the Neural Network to perform training. External procedures are separate programs written by the user, or available in the host operating software. Among the various pieces of information maintained about each variable in the system is a list of rules that the particular variable affects by appearing in their antecedent part. Each time a variable is modified, all the rules that it affects are added to the Expert System's rule queue. The Expert System continuously evaluates the antecedent of each rule in the queue in turn. Rules whose antecedent evaluates to **FALSE** are ignored. A positive result causes the consequent actions of the respective rule to be performed. As an example, when new data is available, the variable **data_available** is set to **TRUE**, triggering a rule which performs scaling then delaying of the input variables, and resulting in an input vector for the Neural Network. This data driven implementation improves on the traditional inference engine which iteratively evaluates the antecedent of *all* the rules in the knowledge base, and executes the consequent when a match is achieved. The system therefore performs only the necessary computations and saves on the unfruitful antecedent evaluations.

The rule queue is implemented as a stack. Rules are inserted in this stack in the proper position according to priority Duplicate rule instances are discarded. The KBES evaluates the antecedent of the rule at the top of the stack and executes the consequent part if the evaluation was successful. The top rule is then popped from the stack and the following one is considered. For debugging purposes, the user can observe the contents of the rule queue as the system is running.

- *The parser*

DASANEX features a built-in parser/interpreter which converts rules specified by the user in a C-like syntax to a more primitive internal representation. The interpreter performs the actions specified by the consequent part of a rule. The lexical parser is implemented using state-of-the-art software engineering techniques. The target language is in Reverse Polish Notation (RPN) for ease of execution. A Backus-Naur Form (BNF) representation of the language is shown in figure 5.9.

The consequent part of a rule consists of a sequence of actions. These actions are of three types: variable assignment, internal procedure calls, and external program invocations. Internal procedure calls feature parameter type checking to minimize rule specification errors. Comments in the source code indicate the conventions used for parameter specification.

- *Built-in procedures*

DASANEX features a collection of built-in procedures for which need was felt while operating the system. These procedures are designed to provide DASANEX with input/output capabilities, and to supply a substitute for repetitive statements that would have otherwise required a loop construct. These procedures are listed below:

- **NETWORK_RECALL**(*network_variable, output_storage, error_storage*): performs a recall operation with the given network. The input variable is implicit in the declaration of the network. The output result and error are stored in the designated locations.
- **PUT_IN_TDBS**(*vector*): archive the specified vector in the training database.
- **OUTPUT**(*variable,...*): concatenates the list of arguments into a string and sends the result to the output manager.
- **SCALE**(*source_variable ,destination_location*): computes the shifted and scaled value of *source_variable* which should be of type **DOUBLE** (with specification of min and max values) and stores the result in *destination_location*.

```
<rule>              ::= <antecedent> <consequent>

<antecedent>        ::= IF ( <expression> )

<expression>        ::= <simple_expression> [ <operator> <expression> ]

<simple_expression> ::= ( <expression> )
                      | ! <simple_expression>
                      | - <simple_expression>
                      | constant
                      | <variable>
                      | <function> ( <arguments> )

<variable>          ::= variable_name
                      | vector_name[constant]

<arguments>         ::= <expression> [ , <arguments> ]

<function>          ::= sin | cos | tan | exp | log | abs | max | pow

<operator>          ::= <arithmetic_oper>
                      | <logical_oper>

<arithmetic_oper>   ::= + | - | * | / | % | & | ^

<logical_oper>      ::= < | > | == | <= | >= | <>

<consquent>         ::= { <statements> }

<statements>        ::= <statement> ; [ <statements> ]

<statement>         ::= <variable> = <expression>
                      | <procedure> ( <arguments> )

<procedure>         ::= network_recall
                      | network_train
                      | compute_rel_err
                      | put_in_tdbs
                      | output
                      | scale
                      | unscale
                      | copy_vec_elems
                      | print_value
                      | call
```

Figure 5.9. BNF representation of the rule language

- **UNSCALE**(*soure_location, original_variable, destination_location*): unscales the value in *source_location* according to the min and max stored in *original_variable* and stores the result in *destination_location.*

- **COPY_VEC**(*from_vec, from_pos, to_vec, to_pos, count*): copies count elements from *from_vec,* location *from_pos,* to *to_vec,* location *to_pos.*

- **PRINT_VALUE**(*variable*): prints the value of any variable passed to it to the output screen, mainly for debugging purposes.

- **FUN_CALL**(*ext_name, arg1, arg2,...*): invokes the external program *ext_name* passing to it the arguments *arg1, arg2, ...*

- **DISPLAY**(*variable, string,...*): concatenates the list of arguments into a string and puts up a dialog box displaying the resulting string to the user.

Additional procedures will be added to DASANEX as the need arises.

5.6.5 The Central Neural Network Processor

The heart of the data sanity checking capabilities of DASANEX resides in this module. The module, as the name might imply, does not contain a neural network. Rather, it contains methods about training and recalling of neural networks. The networks themselves are stored in the database. Prior to training, this module issues a request to the database which returns a list of all the networks that are subject to training. This module then applies the methods it knows about the particular network type to perform training cycles.

When the KBES requires a recall operation, it posts a message to the neural network processor requesting the recall. The body of the message contains a reference to the network to be used during this recall cycle. This module currently supports backpropagation and functional link networks. However, it was designed in such a way that support for other paradigms can easily be incorporated into the system. The operational steps to add a neural network paradigm to the system are outlined in the following.

- *Incorporating other neural network paradigms*

Two files must be created for each network implementation, one containing the operational code which performs training and recall, and a second one which takes care of user interface modifications.

The first file should contain the following:

- A definition of the structure of the network as it should be represented in memory.
- A routine to read the network from the database file and store it in memory.
- Another to write the network to the database using the same format.
- A routine which, given an input vector, performs a training cycle.
- Another routine which performs a recall cycle and computes the resulting error.

The second file contains code which allows the user to modify the characteristics of the given network from within the program. It involves using the NeXT Interface Builder to create the necessary windows and dialog boxes. The code then handles user interaction with the objects in these windows to modify the memory copy of the network. Please refer to the next section for a clearer presentation of this topic.

5.6.6 The User Interface

The User Interface module plays a very important rule in any system that is to be used by humans. The interface is usually evaluated according to two of its facets: the capabilities it provides, and its user friendliness. User friendliness depends as much on standard human interface guidelines as it does on the hardware capabilities of the host machine it runs on, such as graphical output capabilities, window support, and pointing device support (i.e. a mouse). The capabilities offered by the interface are defined and limited by its function. In the case of DASANEX, the User Interface serves four functions: control, modification, observation, and debugging.

The user is granted full control over program flow. He can load and save all the files pertinent to the program, start, pause and stop the operation of the filter (switch between setup and run modes), and naturally, quit the Data Sanity Checking environment.

The User Interface also allows the user to edit the database and the knowledge base directly. While in setup mode, full access is provided to all variables in the database with such operations as addition, deletion, and modification. Similarly, the user can perform all the above operations on the rules in the knowledge base as well as compile any rule at will. Moreover, neural network operating parameters and training thresholds can be specified through this module.

In run mode, the user can observe data at any stage in the filtering process, as well as review rule definitions and operating parameters. Editing and changes are not allowed in this mode, however, since the system cannot operate until it establishes the validity and consistency of all user supplied information and ensures that all rules compile successfully. The mere action of deleting a variable, for instance, can be detrimental to the operation of the system since various rules may include references to that variable in their antecedent and/or consequent parts.

The debugging function of the User Interface is very useful during the development stages of the system. It provides the ability of observing and monitoring various internal variables and data structures which, although of no interest to the user of the filtering capabilities of the system, may convey a great deal of information about the proper functioning of the system to the developer. Examples of such structures are the message passing queue, the KBES rule queue, and the various system status flags.

5.6.7 The Input Module

The input module is the site where all data of the Data Sanity Checking system originates. This module operates in one of two modes, on-line and simulation. In on-line mode, the module monitors a specified input port and awaits data availability. Subsequently, it parses

the input stream according to a specified format, stores the data collected in the appropriate variables, and modifies the value of the variable **data_available** to **TRUE**. This action sets the Data Sanity Checking system in motion and triggers a chain of events that causes data to be filtered.

In simulation mode, the input is taken from a designated file at user specified intervals instead of a port. The variable **reading_interval** specifies this interval. The remainder of the procedure is identical to input from a port. Run mode is switched off when end of file is detected.

Since this module responds to and generates asynchronous events, it clearly qualifies to be a separate thread.

5.6.8 The Output Module

This module reacts to new system output by appending the new information to an output data file, and by displaying the information in a window. The user can scroll through this archival window to observe and review system outputs.

5.7 NeXT Implementation

5.7.1 NeXTStep Interface

Being the only module that performs direct machine interaction, the User Interface makes use of various machine dependent features without much concern for portability. A different User Interface module is required for each platform that the DASANEX system is to be run on. Although the functionality of the User Interface will be preserved across these platforms, the "look and feel", being very dependent on the capabilities of the machine, will vary slightly.

On the NeXT machine, the User Interface is implemented using ***Objective C*** and the ***NeXT Application Toolkit*** available from NeXT Computer Inc. [123]. The module attempts to take full advantage of the ***NeXTstep*** object oriented Graphical User Interface (GUI) [124]. The module consists of a collection of objects, some of which are graphical and can be displayed on the screen, and some of which are abstract, which operate by sending messages to one another. Graphical objects include menus, dialog boxes, panels, buttons, and windows. When the user operates one of these objects (such as clicking the mouse on a button), the object sends a specific message to another designated object.

Abstract objects are designed to accomplish certain functions according to object oriented programming (OOP) design rules. Each object responds to certain messages, and operates by sending messages to other objects, and executing straight C language code such as function calls.

5.7.2 DASANEX Implementation

The following abstract objects are used in the NeXT implementation of the User Interface for the Data Sanity Checking system:

- ***Dispatcher***: This object performs the function of linking the various menu and dialog box items that do not require involved processing to the appropriate modules in the system. For instance, when the user selects the *'Load Database'* menu command, the menu item sends a **'load_dbs'** message to ***Dispatcher*** which in turn issues a **LOAD** command to the database manager.
- ***WHandler***: This object takes care of all window related operations. The user selects menu commands to observe different variables and to monitor internal data structures. ***WHandler*** places the appropriate windows on the screen in response to such requests, then updates these windows when the information they reflect changes, and finally removes them from the screen when the user closes them. Most windows, including the message queue window, the system status window, the operating parameters window, and the observer window, are under this object's control.

- ***Rule Editor***: This object intervenes when the **'Edit_Rule_Window'** (invoked by the *'Edit Knowledge Base'* menu command) is active. It responds to the messages generated by the various items on this window when the user operates them. It allows the user to edit the priority, comment and text of any rule, as well as compile it. In case of compilation errors, the cursor is placed at the offending location in the text, and the appropriate error message is displayed. This same object is used in run mode (with editing disabled) to allow the user to review rules. The interface used by this editor is shown in figure 5.10.

- ***Variable Editor***: The **'Variable_List_Window'** is activated from the menu by selecting the *'Edit Database'* command. When this window is visible, the ***Variable Editor*** object handles all events related to it such as keyboard and mouse clicks. This object allows the user to edit variable names, descriptions, types, and values. This same object responds to similar events when the window is activated in view mode, but editing is disabled. When the window is closed, this object automatically stops receiving messages. This interface is shown in figure 5.11.

- ***Net Editor***: When the user double clicks on a variable of type NETWORK, the network editor is called upon to display specifications of the net and allow the user to modify them. The interface shown in figure 5.12 was designed specifically for backpropagation and functional link networks. For a different type of network, a Hopfield-net for example, some of the network specifications may not apply, and a new interface has to be introduced.

5.8 System Operation

This section relates the chronological events that take place in a sample session, from the moment the user invokes the Data Sanity Checking system till erroneous data gets captured and filtered out.

- The user starts up the system. The threads that form certain modules are forked. The system is in setup mode; it is idle.

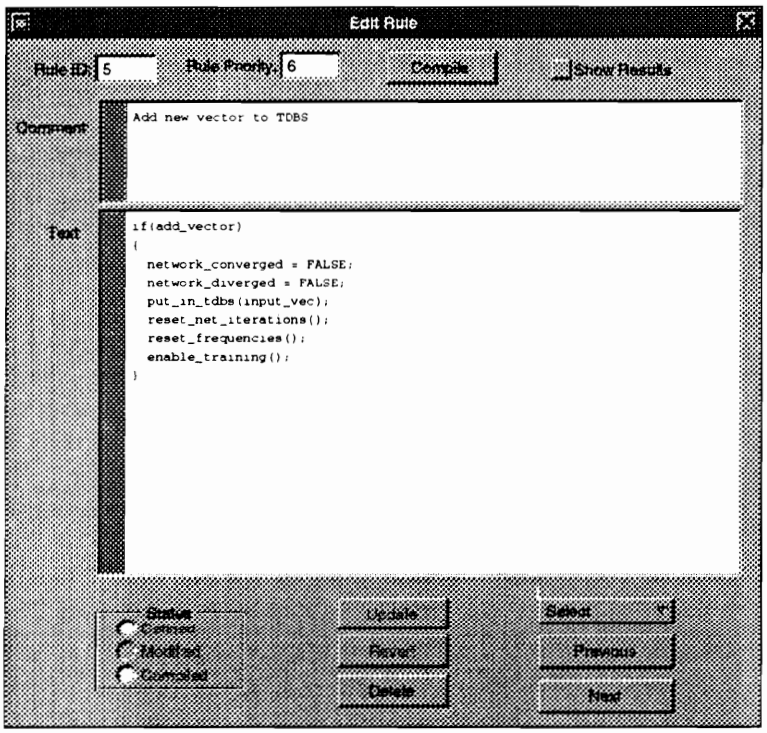


Figure 5.10. Rule Editor interface

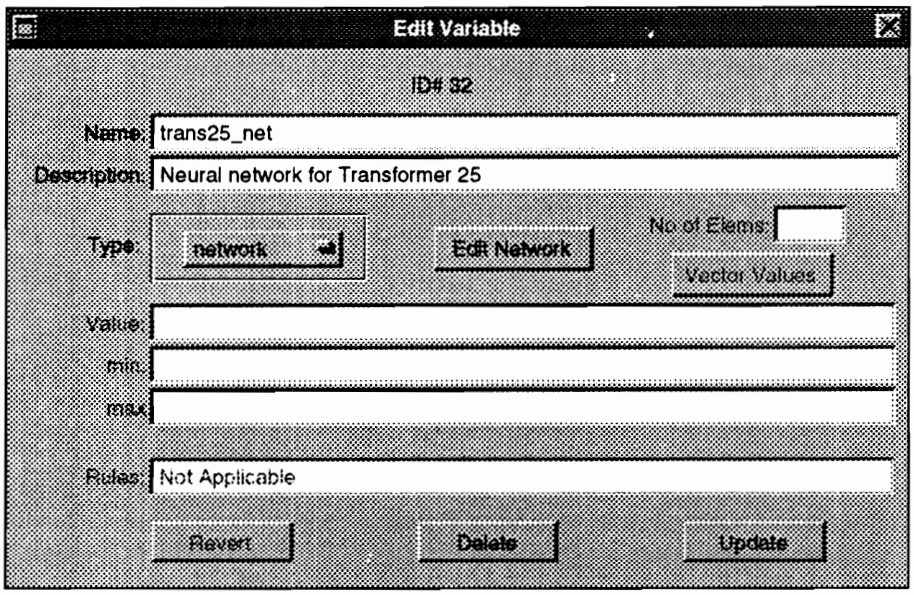


Figure 5.11. Variable Editor interface

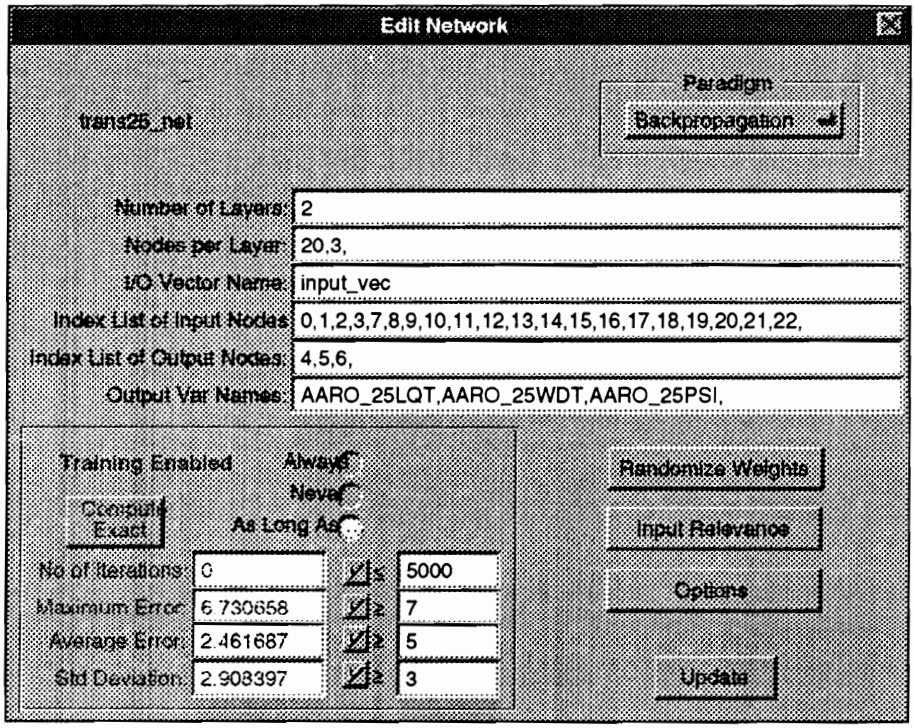


Figure 5.12. Network Editor interface

- The user switches to run mode through the *'Mode:Run'* menu command. The system, detecting that the database has not been loaded, asks for confirmation to load it. Similarly, it will confirm the loading of the knowledge base and the training database. A 'cancel' reply to any of these queries aborts the mode switching operation and restores the system to its idle state.

- Once the database and the knowledge base have been loaded, the system sends a **MAKE** message to the KBS manager requesting that the knowledge base be compiled. Rules are compiled sequentially. In the event of compilation errors, a message is sent to the User Interface which displays the offending rule in the Rule Editor and puts up a message indicating the location and the exact nature of the error. Mode switching, in this case also, is aborted.

- Upon successful compilation, and assuming a simulation (versus on-line) session, the user if prompted for the data file from which input will be obtained. Once a valid file is specified, the system starts the input and output threads and posts an **INIT** message to them.

- The input manager takes care of opening the input file and installing a timed routine which will be called periodically to read a line of input and make it available to the system. Upon success, it also asserts the value of **init_flag** which will cause the chain of rule executions to be triggered. The output manager creates a new output file in which to store the results of the simulation.

- Initialization rules, triggered by **init_flag**, are executed. They set initial values for the different variables needed in run mode.

- After the specified timer period has gone by, the timed routine obtains a new set of readings from the input file and changes the values of the corresponding variables. It then asserts the value of the variable **data_available**. This assertion causes all rules that reference **data_available** in their antecedent part to be added to the Expert System's rule queue.

- The first rule that responds to a change in data_available performs scaling. All input variables are scaled and shifted to lie in the [0-1] range.

- Now that an input vector is available, the Expert System posts a **RECALL** message to the Neural Network Processor for each **NETWORK**-type variable.
- Using the vector supplied, the Neural Network performs a recall phase and obtains history based predictions for the designated outputs. It then notifies the Expert System of the completion of the operation.
- The Expert System analyzes the errors returned by the Neural Network. Output variables for which the relative error is less than a specified level (currently 5%) are kept. Those for which the error is above 5% are replaced by the estimate.
- The Expert System notifies the output module that new output is available. In response, the output module archives the filtered data.
- If any of the resulting errors was above 5%, but less than another specified level (currently 10%), the error may be the result of slow changes in data (due to seasonal dependence, for instance). The TDBS module is instructed to archive the input vector in the training database. This will allow the system to track this type of time dependent change.
- If training is enabled, the Neural Network Processor performs background training cycles with all networks for which recall errors fall in the 5% to 10% range on random training patterns obtained from the training database.

This completes one data pass through the Data Sanity Checking system. The procedure is repeated when the timer period has elapsed and new data is obtained from the file (or in on-line mode, when new data is detected at the input port).

5.9 Summary

In this chapter, we have described the implementation of DASANEX, based on a design that we feel adequately addresses the difficulties and problems encountered while performing data sanity checking. The next chapter will focus on validation and verification of the technique in order to assess its effectiveness and generality, and delineate its domains of applicability.

Chapter VI

VALIDATION AND VERIFICATION

6.1 Introduction

In this chapter, we describe the steps taken to ensure the functionality of DASANEX and its applicability to various data sanity applications. We present a detailed example illustrating the steps required to set up the system for operation. Finally, we offer observations, results and conclusions drawn from the example given.

6.2 Validation Procedure

In the case of data sanity checking, system requirements are well defined. Therefore, an exact procedure can be outlined in order to assert the validity and reliability of the algorithm. The requirements set forth in chapter 3, imposed on the system once it is trained and operational, are as follows:

- the system should detect, correct and possibly report abnormal operation.
- the system should not report good data as incorrect.

- the system should track data changes over time and perform equally well during the period for which it was trained as well as for other time periods.

The validation procedure consists of several steps. The first step is to set up the system properly for the type of data being processed. The proper database elements and knowledge base rules have to be defined. Neural network paradigms appropriate for the particular task have to be selected. The next step is to adequately train the resulting system with good training data. Following that, testing is performed with data starting at the same time period as the training set, and progressing in the future. Wherever appropriate, visual inspection can be performed for critical sections of data during which it displays bad behavior.

6.3 Methodology Verification

The data sanity checking system is based on several different techniques which have been proven to work in other environments. For instance, given a network with adequate capacity and ample training time and data, any continuous mapping can be implemented using a neural network. Additionally, recurrent networks have been found to handle time dependent data efficiently.

Since DASANEX provides all the building blocks for performing data sanity checking by allowing the creation of any set of production rules and the usage of any type of neural network (by following the procedure to add new networks described in section 5.6.5), the task is reduced to selecting the proper network topology and the proper operating parameters (network size, training database depth, etc.). Once these parameters are properly determined, the algorithm is guaranteed to work.

In the following section, we will describe the procedure involved in selecting these parameters for the sample transformer application, and show how it can be extended for other applications.

6.4 Application to Transformer Data

6.4.1 Transformer Data Revisited

In the following, we will demonstrate the application of data sanity checking to the Martinsville transformer example introduced in section 3.3. First, we discuss how the DASANEX system is setup to handle the transformer data, following which we offer training and operation results

6.4.2 DASANEX Setup

Given the description of the data and its inherent properties, we propose to setup and configure DASANEX in order to filter the transformer data, detecting and reporting obvious malfunctions when they occur, and filling in as much data as possible when such data is missing or incorrect.

The steps involved in order to achieve this task are as follows:

- create a DASANEX database of variables which describe the entities being measured and their ranges of values.
- identify a section of historical data from the near past which does not contain obvious aberrations, to be used for preliminary off-line training.
- configure an artificial neural network for every output variable desired by determining the input and output variables for the network as well as the number of hidden layers and nodes.
- modify the existing knowledge base to account for the output variables at hand.
- use the off-line training input file to generate a pre-training binary database.
- pre-train the system under desired levels of accuracy are achieved.
- run the system on actual data. Add rules which notify the system of physical alterations whenever appropriate.

In the following sections, we will elaborate on each of the above steps, emphasizing pertinent system features.

- *Database variables*

The variables necessary for the correct operation of DASANEX are of four kinds. First are general flags that the system requires. These are:

- **INIT_FLAG**: This flag is internally set whenever a **run** command is issued. If used as a test in the antecedent part of a rule, that rule will automatically be executed at startup and can thus be used as an initialization rule.
- **DATA_AVAILABLE**: This flag is also internally set by the system whenever a new reading has been taken and the resulting data has been stored in appropriate variables.
- **DATA_PROCESSED**: This flag prevents the system from collecting a new reading if it is set to **FALSE**. Rules in the knowledge base should set this flag to **TRUE** when they are done with the current data reading and are ready for a new one.

The second class of variables are declared by the knowledge base programmer in order to facilitate his task. In this particular setup, these are:

- **READING_COUNT**: This integer variables keeps track of the number of readings that the system has processed since the last **run** command.
- **PRED_AVAILABLE**: This flag is set after the neural network module has obtained predictions for the all the output variables. It causes appropriate range checking rules to fire.
- **RES_AVAILABLE**: This flag is set to announce that a result to be printed, or archived, is available.
- **INPUT_VEC**: This vector holds the scaled (and possibly delayed) inputs that will be presented to the neural networks. This vector, when deemed appropriate, will also be archived in the on-line training database.
- **PRED_VEC**: This vector holds scaled output predictions from the neural networks.

- **ERR_VEC**: This vector holds percent prediction errors between the actual data measured and that generated by the network.

- **OUTPUT_VEC**: This vector contains output predictions scaled back to their proper ranges. These values will be used to replaced actual readings that deviate too much from the predictions.

- **PRINT_VEC**: This vector holds the actual values that form the output of the system. This vector is subsequently archived and/or presented to the operator.

- **ACCEPT_THRESH**: This variable represents the maximum error level (in percent) that will be tolerated for a variable to be judged correct. For the Martinsville data, this is set to 5% which is widely accepted as an industry standard.

- **REJECT_THRESH**: This lower limit (also in units of percent) is the cut-off point for the error on a prediction over which a variable reading will not be included in the training vector, but will instead be replaced by the predicted value. This is currently set to 10%.

- **DELAY_UNITS**: This variable sets the delay period (in hours) for the functional link networks.

- **REPLACE_COUNT**: Keeps track of how many output variables have been discarded and replaced by predictions.

- **REPLACE_THRESH**: If more than this number of variables have been replaced by predictions, do not include the current vector in the training database.

- **PRETRAIN_FLAG**: This flag indicates whether the current run is to generate a pre-training database or if it is an actual filtering run.

The third kind of variables required by DASANEX are the actual quantities being measured. The name given to each of these variables is very important since it will be used by the input manager module upon receipt of a new data set in order to assign the correct values to the proper variables. In the case of the Martinsville data, each data reading consists of 16 variables, three of which do not have corresponding names in the database and are automatically ignored by the input manager. The variables ignored are the three power measurements after the breakers, since they contain no separate phase information and are

qualitatively redundant with the total power measurement. The remaining variables are listed in table 3.1.

The last class of variables stored in the Martinsville database are the artificial neural networks used for predicting the various output variables. These will be described shortly in the neural network section.

- *Knowledge base rules*

For each input data description, the basic knowledge base has to be altered slightly to reflect the input data and neural network changes. As an option, the system can generate the required changes automatically.

A set of 30 rules has been formulated to allow operation in two modes, as controlled by the variable **PRETRAIN_FLAG**. In the first mode, the expert system performs scaling and delay in order to generate a pre-training database from the specified input file. In the second mode, the system receives data readings and generates output predictions, modifying the training database whenever appropriate.

The operation of the system is best represented by a flow chart, as is provided in figure 6.1. Control flow between the various execution blocks of the chart occurs automatically when key variable values fall within prescribed ranges, or when certain flags are explicitly set or cleared.

- *Artificial Neural Networks*

From observation of the sample plots, the Martinsville data being monitored possesses several key features that warrant the use of a particular neural network. The first observation is that, under normal operating conditions, all measured variables are continuous functions of time. These functions do not change rapidly and abruptly, and generally display no noticeable noise, possibly as a result of the averaging operation during collection.

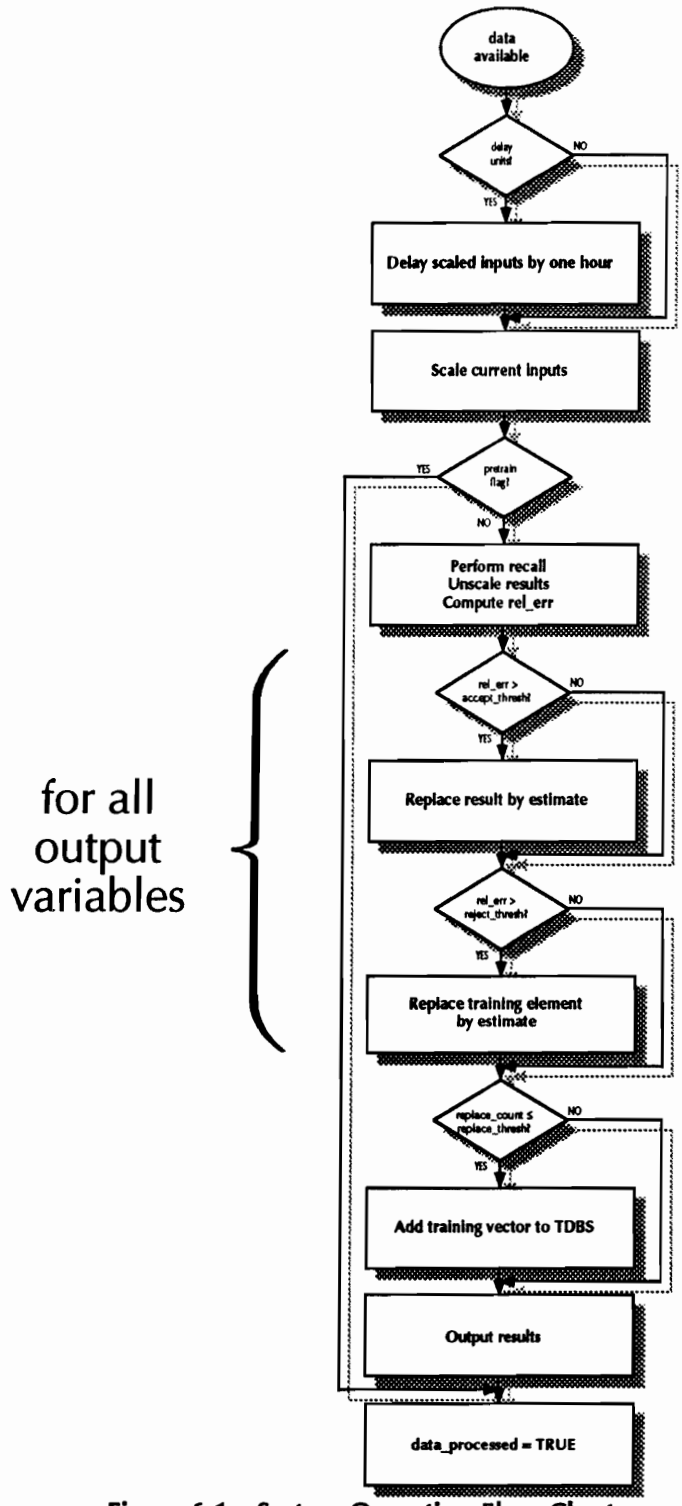


Figure 6.1. System Operation Flow Chart

In order to take advantage of the context concept, and be able to generate a variable from the context, even though its value may be corrupted or missing, every variable will be 'expressed' in terms of all other variables that may affect it excluding itself. For each variable, we will need a neural network that performs an interpolating non-linear mapping between the other variables in the context and the particular output variable. As stated by Caudill [18], the neural networks best suited for this kind of application are the feed-forward backpropagation network, and, by extension, the functional link network.

The next design decisions for every output variable are what to use for inputs, and the number of hidden layers and nodes to have. Once again, to obtain predictions that are context dependent, we will provide as inputs to the network designated for a particular variable all other inputs but that variable. The second issue, selecting the number of hidden layers and nodes, is slightly more delicate, as there are no known or applicable theorems that will impose given values. For particular disciplines such as signal processing, however, there exist rules of thumb that help in selecting suitable values.

In the absence of specific rules, we will resort to an algorithmic procedure that can be followed with acceptable results. To begin with, as proven by Minsky and Papert [110], a minimum of one hidden layer is required in order to achieve any sort of non-linear mapping. The properties of the network that we are interested in, namely capacity, speed of convergence, and generalization capabilities, are more a function of the number of weights in the network than the number of layers, the reason being that all the information contained in any given network is stored in the weights. A network with few weights may not be able to extract and store enough features from the patterns in the training database and may therefore not converge. On the other hand, a network with too many weights will converge, albeit slowly, and will tend to memorize the input patterns rather than extracting the features required for generalization. During pre-training, DASANEX will reserve about 10% randomly selected training database patterns for testing after the networks have settled and converged to within a given tolerance. The procedure to determine the proper size to use for each neural network is as follows (the term *reasonable values* used in this discussion is very data dependent. If one is not sure of the particular values to use, it suffices to perform a pre-training session with the given data to develop a feel for the appropriate range of these values):

- start with one hidden layer and a reasonable number of hidden nodes. For the Martinsville data, a starting value of 5 hidden nodes was found to be acceptable.

- perform a pre-training run. If the network fails to converge after a given number of iterations (100,000 in our case), then the number of hidden nodes is probably too small. Increase this number and repeat this step. An acceptable range of iterations required for convergence is 10,000 to 50,000. What is more important, however, is that the networks for all the predicted variables successfully train in comparatively the same number of iterations. Otherwise the predictions generated by the system will not have the same relevance, or measure of confidence.

- if the network converges in a reasonable number of iterations, perform a testing operation (using the data set aside). This will provide an indication of how well the system responds to new data. If the recall errors on the testing portion of the data are too high, then the network is too large, and has memorized the input patterns rather than extracting their features. Reduce the number of hidden nodes and repeat the previous step.

- once the network converges, observe the input relevance measures reported by the system. These values indicate the relative impact that the input variables have in deciding the value of the output. If any input variable is clearly negligible compared to the others in terms of relevance, then reassess its validity (will its relevance possible change over time?) and consider removing it from the list of input variables for that particular network. Unnecessary 'tag-along' variables hinder the training process and slow down the convergence of the particular network.

6.4.3 System Operation

The operation of the system consists of pre-training the neural networks to an acceptable error level, then performing sanity checking on the data available. For this purpose, data was collected over the period of six months, starting may 1991. The neural networks are configured as per the information in table 6.1. The selection of variables for the various networks is based on several experiments which resulted in the following rules:

Table 6.1. Functional Link Network Inputs

Id	Variable	TOTKW	LQT23	LQT24	LQT25	WDT23	WDT24	WDT25	PSI23	PSI24	PSI25	ATEMP
0	Julian_Day	X	X	X	X	X	X	X	X	X	X	X
1	sin(Time)	X	X	X	X	X	X	X	X	X	X	X
2	cos(Time)	X	X	X	X	X	X	X	X	X	X	X
3	TOTKW		X	X	X	X	X	X	X	X	X	X
4	LQT23	X		X	X	X	X	X	X	X	X	X
5	LQT24	X	X		X	X	X	X	X	X	X	X
6	LQT25	X	X	X		X	X	X	X	X	X	X
7	WDT23	X	X	X	X		X	X	X	X	X	X
8	WDT24	X	X	X	X	X		X	X	X	X	X
9	WDT25	X	X	X	X	X	X		X	X	X	X
10	PSI23	X	X	X	X	X	X	X		X	X	X
11	PSI24	X	X	X	X	X	X	X	X		X	X
12	PSI25	X	X	X	X	X	X	X	X	X		X
13	ATEMP	X	X	X	X	X	X	X	X	X	X	
14	TOTKW-Δ	X	X	X	X	X	X	X	X	X	X	X
15	LQT23-Δ	X	X			X			X			X
16	LQT24-Δ	X		X			X			X		X
17	LQT25-Δ	X			X			X			X	X
18	WDT23-Δ	X	X			X			X			X
19	WDT24-Δ	X		X			X			X		X
20	WDT25-Δ	X			X			X			X	X
21	PSI23-Δ	X	X			X			X			X
22	PSI24-Δ	X		X			X			X		X
23	PSI25-Δ	X			X			X			X	X
24	ATEMP-Δ	X	X	X	X	X	X	X	X	X	X	X
25	TOTKW-2Δ	X	X	X	X	X	X	X	X	X	X	X
26	LQT23-2Δ	X	X			X			X			X
27	LQT24-2Δ	X		X			X			X		X
28	LQT25-2Δ	X			X			X			X	X
29	WDT23-2Δ	X	X			X			X			X
30	WDT24-2Δ	X		X			X			X		X
31	WDT25-2Δ	X			X			X			X	X
32	PSI23-2Δ	X	X			X			X			X
33	PSI24-2Δ	X		X			X			X		X
34	PSI25-2Δ	X			X			X			X	X
35	ATEMP-2Δ	X	X	X	X	X	X	X	X	X	X	X
36	TOTKW-3Δ	X	X	X	X	X	X	X	X	X	X	X
37	LQT23-3Δ	X	X			X			X			X
38	LQT24-3Δ	X		X			X			X		X
39	LQT25-3Δ	X			X			X			X	X
40	WDT23-3Δ	X	X			X			X			X

Table 6.1. Functional Link Network Inputs (Continued)

Id	Variable	TOTKW	LQT23	LQT24	LQT25	WDT23	WDT24	WDT25	PSI23	PSI24	PSI25	ATEMP
41	WDT24-3Δ	X		X			X			X		X
42	WDT25-3Δ	X			X			X			X	X
43	PSI23-3Δ	X	X			X			X			X
44	PSI24-3Δ	X		X			X			X		X
45	PSI25-3Δ	X			X			X			X	X
46	ATEMP-3Δ	X	X	X	X	X	X	X	X	X	X	X
47	TOTKW-4Δ	X	X	X	X	X	X	X	X	X	X	X
48	LQT23-4Δ	X	X			X			X			X
49	LQT24-4Δ	X		X			X			X		X
50	LQT25-4Δ	X			X			X			X	X
51	WDT23-4Δ	X	X			X			X			X
52	WDT24-4Δ	X		X			X			X		X
53	WDT25-4Δ	X			X			X			X	X
54	PSI23-4Δ	X	X			X			X			X
55	PSI24-4Δ	X		X			X			X		X
56	PSI25-4Δ	X			X			X			X	X
57	ATEMP-4Δ	X	X	X	X	X	X	X	X	X	X	X
58	TOTKW-5Δ	X	X	X	X	X	X	X	X	X	X	X
59	LQT23-5Δ	X	X			X			X			X
60	LQT24-5Δ	X		X			X			X		X
61	LQT25-5Δ	X			X			X			X	X
62	WDT23-5Δ	X	X			X			X			X
63	WDT24-5Δ	X		X			X			X		X
64	WDT25-5Δ	X			X			X			X	X
65	PSI23-5Δ	X	X			X			X			X
66	PSI24-5Δ	X		X			X			X		X
67	PSI25-5Δ	X			X			X			X	X
68	ATEMP-5Δ	X	X	X	X	X	X	X	X	X	X	X
69	TOTKW-6Δ	X	X	X	X	X	X	X	X	X	X	X
70	LQT23-6Δ	X	X			X			X			X
71	LQT24-6Δ	X		X			X			X		X
72	LQT25-6Δ	X			X			X			X	X
73	WDT23-6Δ	X	X			X			X			X
74	WDT24-6Δ	X		X			X			X		X
75	WDT25-6Δ	X			X			X			X	X
76	PSI23-6Δ	X	X			X			X			X
77	PSI24-6Δ	X		X			X			X		X
78	PSI25-6Δ	X			X			X			X	X
79	ATEMP-6Δ	X	X	X	X	X	X	X	X	X	X	X

- If the output of the network is a quantity that is NOT dependent on a single transformer (i.e. TOTKW and ATEMP), the inputs provided to the network consist of current values of all the other variables in the context except for variable being predicted, as well as delayed values for all variables, including the predicted output variable (since actual measurements are available for previous time periods).
- If, on the other hand, the output variables are specific to a particular transformer, then the inputs used consist of all variables currently measured excluding those being predicted, plus delayed values of only the transformer specific variables and the common ambient temperature and total load variables.

The training database, originally containing the pre-training patterns, progressively grows as new acceptable patterns are encountered until it reaches a given limit. At that point, the oldest encountered patterns start getting replaced by the new ones (circular memory). The current training database is 6 weeks in depth. This resulted from discussions with experts in the field.

6.4.4 Pretraining

Shown in figure 6.2, the pre-training data were selected to be the second week in May. This data represents the first time period when nearly all the variables were available and displayed valid values. The exception being **AARO_PSI23** which was then miscalibrated. A week of data was considered because the load displays a weekly pattern which is clearly different on weekdays and on weekends.

6.4.5 Simulation Results

Once the system trained, a simulation run was performed on sets of measured data. For demonstration purposes, the system was used to predict the various variables for the whole month of May. The test data includes a full week before the training period which also features missing variables, the week on which the system was pre-trained, and two weeks following the training period. During simulation, the system is performing on-line training according to the algorithm in figure 4.6, which is imbedded in the flow of operations shown in figure 6.1. Sample results of this simulation operation are displayed in figures 6.3 and 6.4.

Figure 6.3 compares the measured and predicted values for the winding temperature of transformer 25 (**WDT25**). As apparent in the figure, the estimates are fairly close to the actual values. The winding temperature values measured for this period did not display any irregularities, and the system was able to track the various changes including the weekday versus weekend variations. A more interesting case is described below.

Shown in figure 6.4, we have singled out the variable **AARO_TOTKW** to illustrate some points. In the current context, **AARO_TOTKW** (total load on all the transformers) is an independent variable which causes changes in all other variables except for ambient temperature. For the first week of May, **AARO_TOTKW** readings were not available due to sensor malfunction. As figure 6.4 shows, however, the system has been able to successfully synthesize appropriate values for all the missing readings of this variable from the remaining parameters measured. When the sensor problem is corrected, we can see that the actual data nicely joins the predictions. Furthermore, the shape and values of the predicted signal for the first week conform to the general load shape and displays the same features as the actual load measured in the remaining weeks.

The ability that the system displayed in predicting missing values could not be achieved if values for all readings were missing at once for a short period of time. The fix to this inadequacy is to add time lag variables to the network input vector. In case of simultaneously missing readings, the system will then rely on the past values of the parameters to predict the current ones.

6.5 Conclusion

The theoretical concept presented in chapter 3 is tested using real world data, and its validity is demonstrated. It is shown that missing readings of various physical parameters can be reproduced by examining others which jointly reflect the working of a complex system.

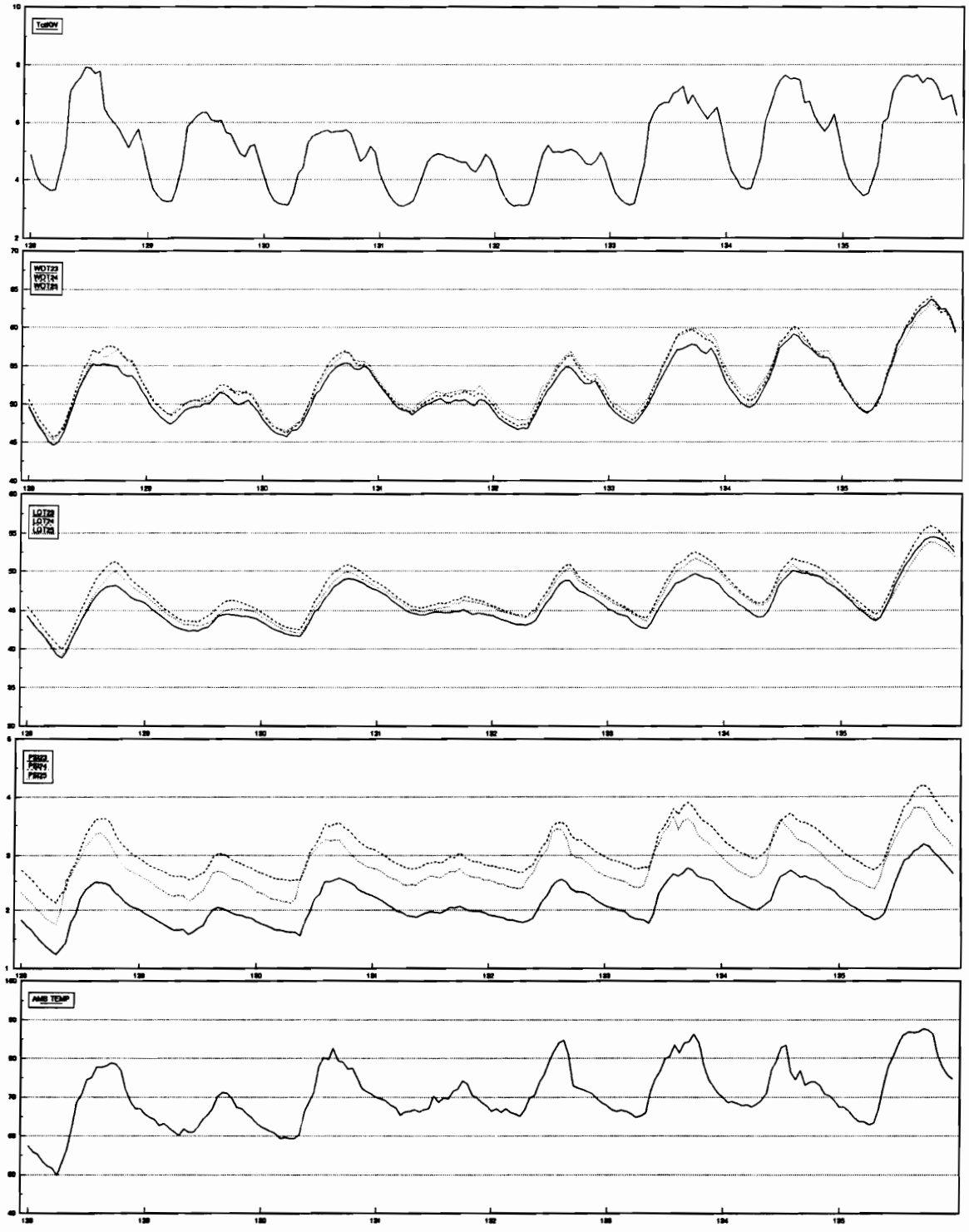


Figure 6.2. Pre-training Data (May 8th-May 15th, 1991)

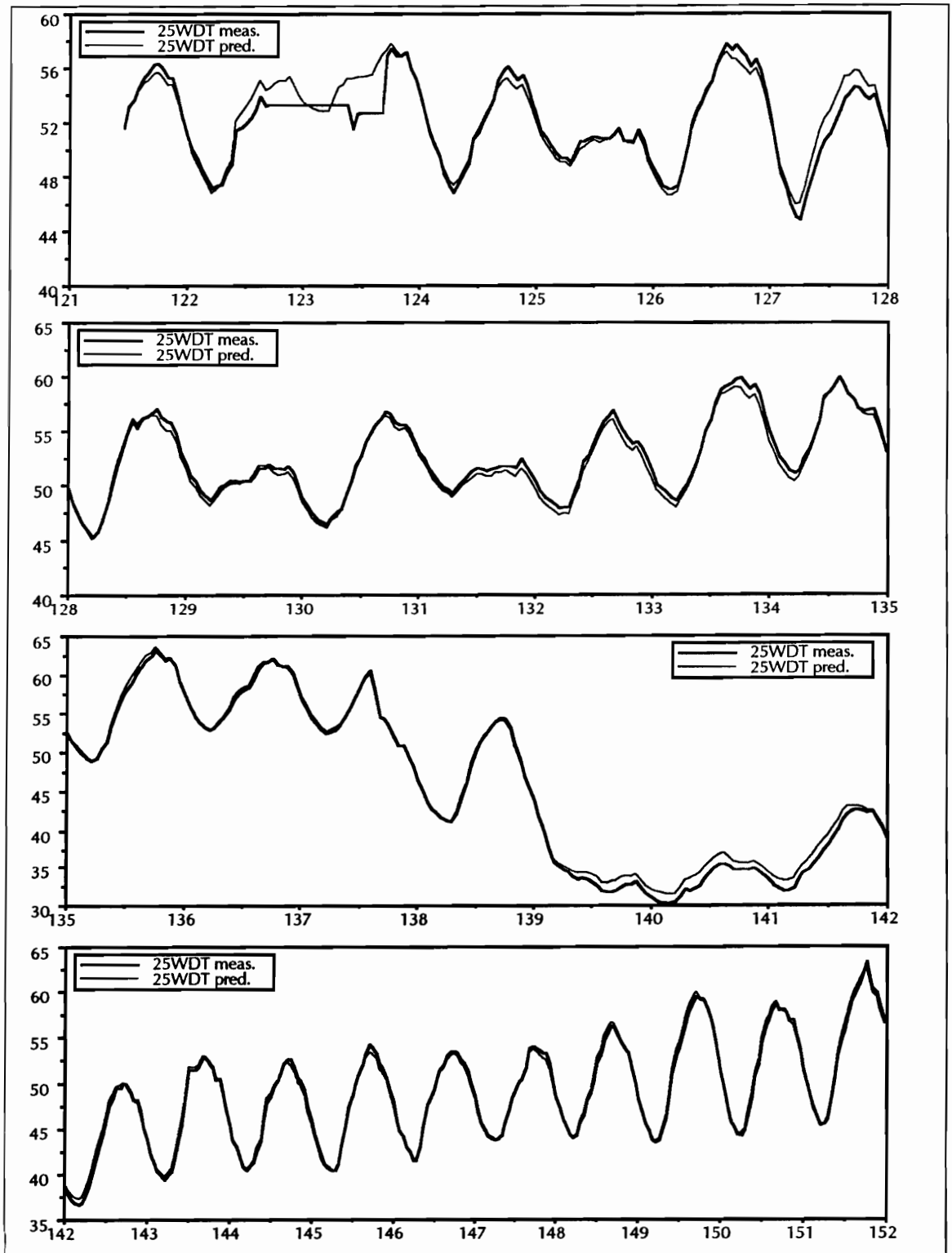


Figure 6.3. Sample Simulation Results - WDT25

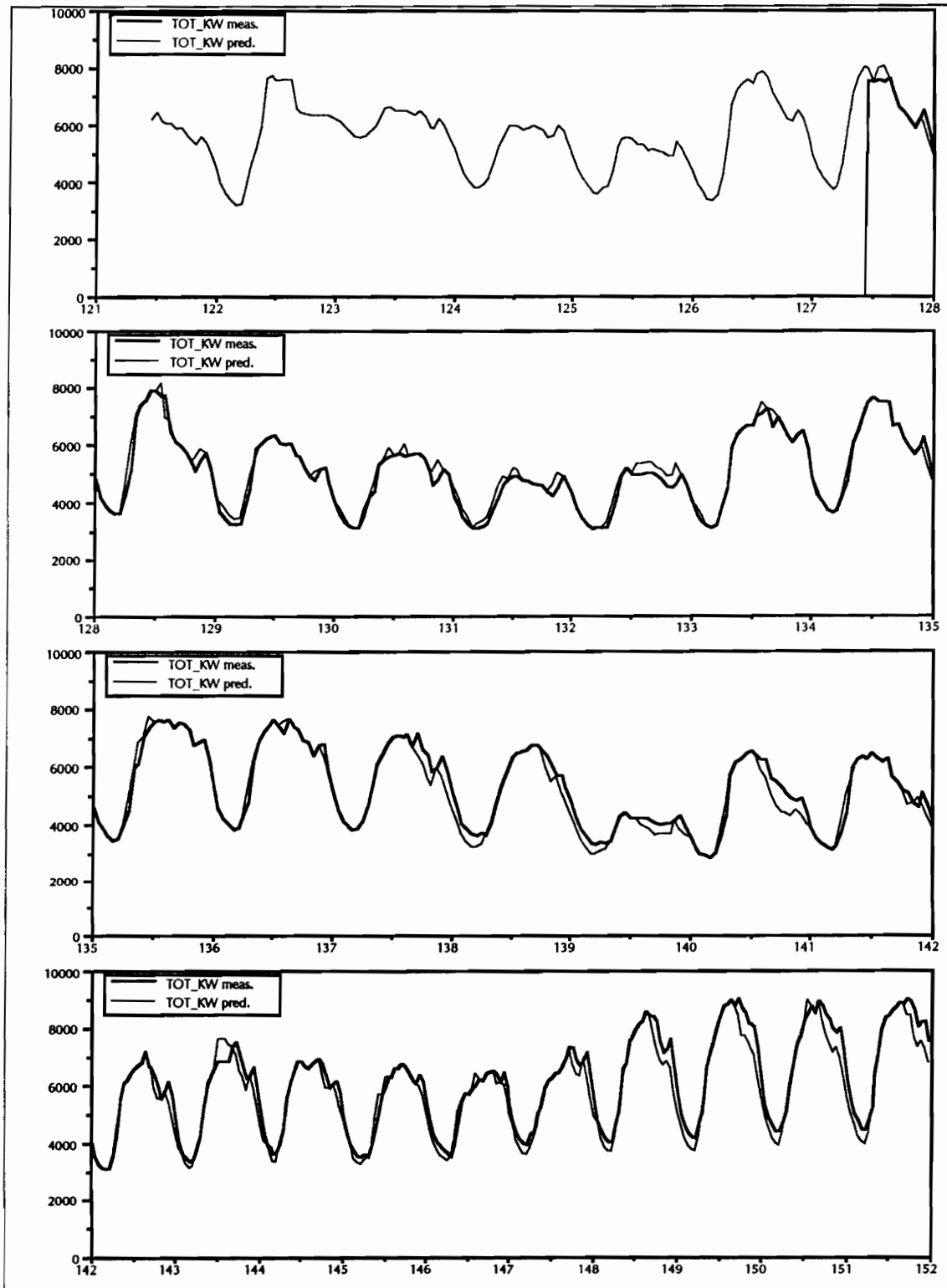


Figure 6.4. Sample Simulation Results - TOTKW

Chapter VII

CONCLUSIONS AND RECOMMENDATIONS

7.1 Concluding Remarks

During this research, we have developed a methodology for data sanity checking to be used during on-line operation as well as for off-line conditioning of data for further use. Through the combination of a rule-based system for data preparation and special case handling, and a neural network for the actual analysis of the data, the system can adapt to different types of data and handle various operating conditions automatically.

A proof-of-concept has been established using field data from the city of Martinsville Electric Department. The use of off-line data has shown that neural network based monitoring systems are able to predict the transformer winding temperature even when some sensor data are missing or contaminated.

We have implemented DASANEX, a prototype of this data sanity checking system, on the NeXT computer, and used it to apply the algorithm to the filtering and cleaning of transformer data.

We conclude that neural networks can be used to model small systems which are remotely monitored using the cross-sensor redundancy concept. The neural networks can be

used to represent parametric relationships between the different system quantities being monitored. Continuous updating of the relationships via on-line training allows the networks to adapt to the current context within which the data is being measured. Inconsistent large differences between the values estimated by the neural networks and the actual measurements can be used as an indication of data acquisition failure, rather than serious system problems. Observation of the internal weights of the networks once they converge can provide a means for measuring deviation of the internal parameters of the system monitored (e.g. transformer) from some base values determined after a system inspection, or at manufacturing time.

7.2 Potential Application Examples

In this section, we present some examples of potential practical applications of the data sanity checking algorithm proposed. The examples, described by Rahman in [135], stem from recent discussions and interactions with electrical utility personnel, and thus tend to express needs currently felt by the power system community.

7.2.1 Monitoring of Transformer Data

The Electric Department of the City of Martinsville has identified some unique applications for an automated data sanity checking algorithm. They believe that this algorithm can help them to build an early warning system for their major equipment like transformers and breakers. This would not only protect expensive equipment from failure, but would also prevent service interruption and loss of energy sales. Typical of many other utilities in Virginia, the City of Martinsville Electric Department depends on remotely collected data for equipment monitoring, control and maintenance. One example is that transformer overload is sensed by monitoring the winding temperature. On hot and sunny days, it becomes very crucial that these temperatures be monitored very accurately for taking remedial actions against overheating which can cause the transformer to blow up. The risk of such an event is not only the loss of equipment costing several hundred thousands of dollars, but also loss of electric service to many customers for several days or weeks. The temperature sensors in these transformers

lose their accuracy over time because of dirt accumulation and decomposition of oil. Through periodic maintenance and manual checking of the transformer temperature on critical days, the technicians at the City of Martinsville Electrical Department have been able to keep all their transformers in proper working condition over the years. However, as equipment tolerance limits decrease (because equipment is sized closer to the load), and the manpower availability goes down, it becomes increasingly important to rely on remotely sensed data to determine the transformer overload status.

The context based data sanity algorithm presented here can monitor the temperature readings and compare these against some other variables to determine whether the winding temperature is consistent with the physical environment. Thus it will be possible to determine whether the remotely sensed information about the transformer core is meaningful in the context within which it is operating. This would give the system operator more confidence on the transformer loading conditions, and the need for expensive manual inspections will be reduced. Once this concept is made to work, it can also be applied to monitor circuit breaker conditions and operate the system more efficiently. The context based automated data sanity checking algorithm can be applied to many other situations where crucial decisions are taken based on remotely sensed data.

7.2.2 Validation of Meteorological Data

In a different context, Virginia Power is collecting meteorological, cell temperature and solar electric data from the VISTA facility in Louisa County. The station is unmanned and the data is collected remotely from Richmond. Occasionally, the pyrheliometer (used for measuring direct solar radiation) setting goes off calibration and the direct normal radiation is affected. An experienced operator goes there once a week and adjusts the pyrheliometer based on his experience. Sometimes, this leads to over correction or under correction, and during the intervening period, incorrect data continue to be recorded. Presently, there is no on-line mechanism for sanity checks on the data being collected from the VISTA system. Some filtering is performed at the time of processing the archived data. The pre-set rules of this filter tend to be overly restrictive because the conditions existing at the time of data collection are not known, and as a result, some good data are discarded. A context based data sanity checking

algorithm would examine the data on-line, and discard the bad data or interpolate between missing entries. But most importantly, such an algorithm can be used to indicate any malfunction of the instrument, or to determine when the instrument goes off calibration. The end result would be lesser manpower requirement and more reliable data.

7.2.3 DASANEX as a Front-End Processor

The third and last example results from an ongoing project at the Energy Systems Research Laboratory at Virginia Tech. In the process of collecting data for the design of an Integrated Load Management Simulator, it has become apparent that a context based data sanity checking algorithm needs to be developed. The integrated load management simulator uses meteorological and building load data to generate control actions for load management. The error checking that is built into this simulator is based on prior knowledge of the variables of concern. This works fine for a static system where the rules and relationships do not change over time. However, in the case of building load control where fuzzy variables like "comfort" and "adequate lighting" are of concern, preset ranges of these variables can be too restrictive and may discard some good data. Ranges of the variables, such as temperature, humidity, air pressure and lighting level, are dynamic, and they change with the context and time. The currently available technology can check for transmission errors in the data, but does not check for data sanity in real time. This sometimes results in inappropriate control action causing discomfort in the conditioned space. The net result is that controls are manually over-ridden, and the purpose of optimum energy usage is defeated. An automated data sanity checking algorithm would determine the validity of the data at the source, and assist in maintaining the appropriate control action.

7.3 Recommendations and Further Work

The software implemented in this research is prototypical. Although the prime purpose of the system is to prove the feasibility of the concept, great care has been taken, throughout the design and implementation phases, to ensure portability and expandability. Implementation of more neural network modules will greatly increase the capabilities of the existing system.

The following steps should be considered in planning further research:

- Equip the system with all relevant neural network paradigms.
- Compile a library matching neural network types to data characteristics. Provide the user with a list of recommended network types given a description of data characteristics.
- Develop a method for automatic matching of neural networks to data. The system can analyze a sample of the data and recommend the appropriate network to be used.
- Provide the system with graphical output capability (graph and plot generation) to allow visual inspection of relevant sections of data.
- Provide means for batch processing in order to allow group filtering and cleaning of previously collected data.

REFERENCES

1. Addis, T.R., **Designing Knowledge-Based Systems**, Prentice Hall, 1986.

2. Adibi, M.M., Clements, K.A., Kafka, R.J. and Stovalt, J.P. "Remote Measurement Calibration," *IEEE Computer Applications in Power*, October 1990, 37-42.

3. Aggoune, M., El-Sharkawi, M.A., Park, D.C., Damborg, M.J. and Marks, R.J. "Preliminary Results on Using Artificial Neural Networks for Security Assessment," *Sixteenth Power Industry Computer Application (PICA) Conference*, Seattle, Washington, May 1-5, 1989, 252-258.

4. Aggoune, M.E., Atlas, L.E., Cohn, D.A., Damborg, M.J., El-Sharkawi, M.A. and Marks, R.J. "Artificial Neural Networks for Power System Static Security Assessment," *IEEE International Symposium on Circuits and Systems*, Portland, OR, May 8-11, 1989, Vol. 1, 490-494.

5. Ajjarapu, V. and Albanna, Z. "Application of Genetic Based Algorithms to Optimal Capacitor Placement," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 251-255.

6. Akimoto, Y., Tanaka, H., Yoshizawa, J., Klapper, D.B., Price, W.W. and Wirgau, K.A. "Transient Stability Expert System," *IEEE/PES 1988 Summer Meeting*, Portland, Oregon, July 24-29, 1988.

7. Alves da Silva, A.P., Quintana, V.H. and Pang, G.K.H. "Associative Memory Models for Bad Data Processing," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 209-216.

8. Alves da Silva, A.P., Quintana, V.H. and Pang, G.K.H. "Neural Networks for Topology Determination of Power Systems," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 297-301.

9. Anderson, J.A. "A Simple Neural Network Generating an Interactive Memory," *Biomedical Sciences*, Vol. 14, 1972, 197-220.

10. Aubin, J. "Thermal Aspect of Transformers," *CIGRÉ SC 12 Colloquium*, Rio de Janeiro, Brazil, October 1989.

11. Baggini, L., Cicoria, R., Invernizzi, A., Gallanti, M. and Pessi, E. "Expert System for Power System Planning. SELF: An Assistant to Load-Flow," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 55-62.

12. Bailey, D. and Thompson, D. "How to Develop Neural-Network Applications," *AI Expert*, Vol. 5, No. 6, June 1990, 38-47.

13. Barnett, V. and Lewis, T., **Outliers in Statistical Data**, John Wiley and Sons, New York, 1984.

14. Baumann, T., Germond, A. and Tschudi, D. "Impulse Test Fault Diagnosis on Power Transformers Using Kohonen's Self-Organizing Neural Network," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 642-647.

15. Bhatnagar, R. and Rahman, S. "Application of Knowledge Based Algorithms in Electric Utility Load Forecasting," *Proceedings of the 1986 Southeastcon*, Richmond, Virginia, March 23-25, 1986, 60-64.

16. Biswas, G., Debelak, K.A. and Kawamura, K. "Applications of Qualitative Modeling to Knowledge-Based Risk Assessment Studies," *Proceedings of the Second International Conference on IEA/AIE*, Tullahoma, Tennessee, June 6-9, 1989, Vol. 1, 92-101.

17. Brace, M.C., Schmidt, J. and Hadlin, M. "Comparison of the Forcasting Accuracy of Neural Networks with Other Established Techniques," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 31-35.

18. Caudill, M. "Avoiding the great back-propagation trap," *AI Expert*, July 1991, 29-35

19. Caudill, M. "Neural Networks Primer, Part I," *AI Expert*, December 1987, 46-61.

20. Caudill, M. "Neural Networks Primer, Part II," *AI Expert*, February 1988, 55-61.

21. Caudill, M. "Neural Networks Primer, Part III," *AI Expert*, June 1988, 53-59.

22. Caudill, M. "Neural Networks Primer, Part IV," *AI Expert*, August 1988, 61-67.

23. Caudill, M. "Neural Networks Primer, Part V," *AI Expert*, November 1988, 57-65.

24. Caudill, M. "Using Neural Nets: Hybrid Expert Networks, Part 6," *AI Expert*, Vol. 5, No. 11, November 1990, 49-54.

25. Caudill, M. and Butler, C. **Naturally Intelligent Systems**, MIT Press, Cambridge, Massachusetts, 1990.

26. Chan, E-K. "A Real Time Expert System Integrated in a Large SCADA System," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 495-498.

27. Chan, E.H.P. "Application of Neural-Network Computing in Intelligent Alarm Processing," *Sixteenth Power Industry Computer Application (PICA) Conference*, Seattle, Washington, May 1-5, 1989, 246-251.

28. Chandrasekaran, B., Tanner, M.C. and Josephson, J.R. "Explaining Control Strategies in Problem Solving," *IEEE Expert*, Spring 1989, 9-24.

29. Chow, J-C., Fischl, R., Kam, M., Yan, H.H. and Ricciardi, S. "An Improved Hopfield Model for Power System Contingency Classification," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2925-2928.

30. Chow, M-Y. and Thomas, R.J. "Neural Network Synchronous Machine Modeling," *IEEE International Symposium on Circuits and Systems*, Portland, Oregon, May 8-11, 1989, Vol. 1, 495-498.

31. Chow, M., Bilbro, G. and Yee, S. "Application of Learning Theory to a Single Phase Induction Motor Incipient Fault Detector Artificial Neural Network," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 97-101.

32. Chow, M.-Y. and Yee, S.O. "Real Time Application of Artificial Neural Networks for Incipient Fault Detection in Induction Machines" *Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & expert Systems*, Charleston, South Carolina, July 15-18, Vol. 2, 1030-1036.

33. Chow, M.-Y., Mangum, P. and Thomas, R.J. "Incipient Fault Detection in DC Machines Using a Neural Network," *22nd Asilomar Conference on Signals, Systems and Computers*, 31 October-2 November, 1988, Vol. 2, 706-709.

34. Criado, R., Matauco, D., Lasheras, F., Fernández, J.L., Basagoiti, P. and Serna, J. "An On-Line Expert System For Contingency Analysis and Corrective Solutions in Transmission," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 68-73.

35. Crowley, T.H., Hagman, W.H., Tabors, R.D. and Cooke, C.M. "Expert Systems for On-Line Monitoring of Large Power Transformers," **Expert System Applications for the Electric Power Industry**, J.A. Naser, Ed., Hemisphere Publishing Corporation, NY, 1991, Vol.1, 639-660.

36. Damborg, M.J., El-Sharkawi, M.A., Aggoune, M.E. and Marks II, R.J. "Potential of Artificial Neural Networks in Power System Operation," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2933-2937.

37. Dash, P.K., Saha, S. and Nanda, P.K. "Artificial Neural Net Approach for Capacitor Placement in Power System," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 247-250.

38. Davis, R., Shrobe, H., Hamscher, W., Wieckert, K., Shirley, M. and Polit, S. "Diagnosis Based on Description of Structure and Function," *Proceedings of the National Conference on AI*, 1982, 137-142.

39. Dillon, T.S. and Laughton, M.A. editors, **Expert System Applications in Power Systems**, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1990.

40. Don Russel, B. and Watson, K. "Power Substation Automation Using A Knowledge Based System- Justification and Preliminary Field Experiments," *IEEE Transactions on Power Delivery*, Vol. PWRD-2, no. 4, October 1987, 1090-1097.

41. Ebron, S., Lubkeman, D.L. and White, M. "A Neural Network Approach to the Detection of Incipient Faults on Power Distribution Feeders," *IEEE Transactions on Power Delivery*, Vol. 5, No. 2, April 1990, 905-914.

42. El-Sharkawi, M.A., Marks, R.J., Aggoune, M.E., Park, D.C., Damborg, M.J and Atlas, L.E. "Dynamic Security Assessment of Power Systems Using Back Error Propagation Artificial Neural Networks," *Second Symposium on Expert System Applications to Power Systems*, Seattle, Washington, July 17-20, 1989, 366-370.

43. El-Sharkawi, M.A., Marks, R.J., Damborg, M.J., Atlas, L.E., Cohn, D.A. and Aggoune, M. "Artificial Neural Networks as Operator Aid for On-Line Static Security Assessment of

Power Systems," *Proceedings of the Tenth Power Systems Computation Conference*, Graz, Austria, August 19-24, 1990, 895-901.

44. El-Sharkawi, M.A., Oh, S., Marks II, R.J., Damborg, M.J. and Brace, C.M. "Short Term Electric Load Forecasting Using an Adaptively Trained Layered Perceptron," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 3-6.

45. Elliott, T. "Neural Networks- Next Step in Applying Artificial Intelligence," *Power*, Vol. 134, No. 3, March 1990, 45-46.

46. EPRI Technical Brief, "Automated Remote Monitoring System," *Technical Brief RP1864-1,2338-1*, 1988.

47. Fandino, J., Darnault, P.H., Bigeon, J., Sabonnadiere, J.C., Harmand, Y., Heilbronn, B. and Mondon, E. "An Expert System as a Help for Power System Restoration After a Blackout," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 487-494.

48. Fenley, C., **Expert Systems, Concepts and Applications**, Library of Congress Cataloging-in-Publication Data, 1988.

49. Filbert, D. "Fault Diagnosis in Nonlinear Electromechanical Systems by Continuous Time Parameter Estimation," *ISA Transactions*, Vol. 24, no. 3, 1985, 23-27.

50. Fischl, R., Kam, M., Chow, J.C. and Ricciardi, S. "Screening Power System Contingencies Using a Back Propagation Trained Multiperceptron," *IEEE International Symposium on Circuits and Systems*, Portland, OR, May 8-11, 1989, Vol. 1, 486-489.

51. Fischl, R., Kam, M., Chow, J.C. and Yan, H.H. "On the Design of Neural Networks for Detecting the Limiting Contingencies in Power System Operation," *Proceedings of the Tenth Power Systems Computation Conference*, Graz, Austria, August 19-24, 1990, 887-894.

52. Forsyth, R. and Rada, R., **Machine Learning - Applications in Expert Systems and Information Retrieval**, Ellis Horwood Limited, 1986.

53. Fox, M.S., Lowenfeld, S. and Kleinosky, P. "Techniques for Sensor-Based Diagnosis," *Proceedings of the eighth IJCAI*, 1983, 158-163.

54. Fukui, C. and Kawakami, J. "Switch Pattern Planning in Electric Power Distribution Systems by Hopfield-type Neural Network," *Proceedings of the 1989 International Joint Conference on Neural Networks (IJCNN)*, Washington, D.C., June 18-22, 1989, Vol. 2, 591-594.

55. Fukuyama, Y. and Ueki, Y. "An Application of Artificial Neural Network to Dynamic Economic Load Dispatching," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 261-265.

56. Fustar, S. and Jelavic, B. "A Knowledge-Based System for Power System Weekly Scheduling," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 116-122.

57. Gale, W.A, **Artificial Intelligence and Statistics**, Addison-Wesley, Reading, Massachusetts 1986.

58. Galiana, F.D., McGillis, D.T., Ahmad, B. and Bernard, J.-P. "An Expert System For Insulation Coordination", *Proceedings of Expert Systems Applications in Canadian Electric Utilities*, Regina, Saskatchewan, September 9, 1989, 1088-1095.

59. Garson, G. David "Interpreting neural-network connection weights," *AI Expert*, April 1991, 47-51.

60. Gaushell, D.J. and Darlington, H.T. "Supervisory Control and Data Acquisition," *Proceedings of the IEEE*, Vol. 75, no. 12, Dec. 1987, 1645-1658.

61. Giuffre, M. and Johnson D. "Demand Side Management Assistant (DSMA): An Expert System for Electric Utilities," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 113-115.

62. Grossberg, S. "Contour Enhancement, Short-term Memory, and Constancies in Reverberating Neural Networks," *Studies in Applied Mathematics*, Vol. 52, 1973, 213-257.

63. Guida, G. and Tasso, C. "The Issue of Knowledge Acquisition in the Design of Rule-Based Expert Systems," *Proceedings of IFAC Artificial Intelligence*, Leningrad, USSR, 1983, 31-36.

64. Haida, T. and Akimoto, Y. "Voltage Optimization by Using Genetic Algorithms," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 375-380.

65. Harashima, F., Demizu, Y., Kondo, S. and Hashimoto, H. "Application of Neural Networks to Power Converter Control," *IEEE Industry Applications Society Annual Meeting*, San Diego, CA, October 1-5, 1989, 1086-1091.

66. Hayashi, Y., Iwamoto, S., Matsuda, S. and Akimoto, Y. "Introduction of Neual Network Theory to Newton-Raphson Load Flow," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 343-350.

67. Hayes-Roth, F., Waterman, D.A. and Lenat, D.B., **Building Expert Systems**, Addison-Wesley, Reading, Massachusetts, 1983.

68. Hebb, D.O. **The Organization of Behavior**, John Wiley and Sons, New York, 1949.

69. Hillman, D. "Bridging Acquisition and Representation," AI Expert, Nov. 1987, 38-47.

70. Hillman, D.V. "Integrating Neural Nets and Expert Systems," AI Expert, June 1990, 54-59.

71. Holzworth, R.E. "On-Line Plant Performance Monitoring," *ISA Transactions*, Vol. 26, no. 1, 1987, 41-47.

72. Hopfield, J.J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, Vol. 79, 1982, 2554-2558.

73. Hui, K.D. and Short, M.J. "A Neural Networks Approach to Voltage security Monitoring and Control," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 89-96.

74. Inoue, T., Tanaka, Y., Tsukiyama, H., Suzuki, T., Wada, N. and Hiramatsu, H. "Power Plants Maintenance Scheduling Expert System," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 574-578.

75. Iordache, C., Mah, R.S.H. and Tamhane A.C. "Performance Studies of the Measurement Test for Detection of Gross Errors in Process Data," *AIChE Journal*, Vol. 31, no. 7, July 1985, 1187-1195.

76. Jackson, P., **Introduction to Expert Systems**, Addison-Wesley, Wokingham, England, 1986.

77. Jilai, Y. and Zhuo, L. "Artificial Neural Networks Based Steady State Equivalents of Power Systems," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 174-177.

78. Johnson, W.P., Woodhead R. and O'Brien J. "The FORD Motor Company DLMS," *AI Expert*, August 1989, 42-52.

79. Jongepier, A.G., Disk, H.E. and van der Sluis, L. "Neural Networks Applied to Alarm Processing," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 615-621.

80. Kahn, G., Nowlan, S. and McDermott, J. "MORE: An Intelligent Knowledge Acquisition Tool," *Proceedings of the ninth IJCAI*, August 1985, Vol. 1, 581-584.

81. Kakimoto, N., Lin., B. and Hayashi, M. "Restoration of Power System from Complete Outage by Expert System," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 262-269.

82. Kaplan, S.J. "The industrialization of artificial intelligence: from by-line to bottom line," *AI Magazine*, vol. 5, No. 2, 1984.

83. Keronen, J.J. "An Expert System Prototype for Event Diagnosis and Real-Time Operation Planning in Power System Control," *IEEE/PES Summer Meeting*, Portland, Oregon, July 24-29, 1988.

84. Khanna, T. **Foundations of Neural Networks**, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

85. Khaparde, S.A., Kale, P.B. and Agarwal, S.H. "Application of Artificial Neural Networks in Protective Relaying of Transmission Lines," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 122-126.

86. Kim, Y-J. and Agogino, A.M. "Signal Validation for Expert System Development," *proceedings of the EPRI Conference on Expert System Applications for the Electric Power Industry*, Boston, Massachusetts, September 9-11, 1991.

87. Kohonen, T. **Self Organization and Associative Memory**, Springer-Verlag, 1984.

88. Kosko, B., **Neural Networks and Fuzzy Systems**, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

89. Kowalik, J.S., editor, **Coupling Symbolic and Numerical Computing in Expert Systems**, Elsevier Science Publishers, 1985.

90. Kraft, T., Okagaki, K., Ishii, R., Surko, P., Brandon, A., DeWeese, A., Peterson, S. and Bjordal, R. "A Hybrid Neural Network and Export System for Monitoring Fossil Fuel Power plants," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 215-218.

91. Kumar, A.B.R., Ipakchi, A., Brandwajn, V., El-Sharkawi, M.A. and Cauley, G. "Neural Networks for Dynamic Security Assessment of Large-Scale Power Systems: Requirements

Overview," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 65-71.

92. Lahouar, S. and Rahman, S. "Design and Implementation of an Expert System for Data Sanity Checking," *Proceedings of the 1989 Southeastcon*, Columbia, South Carolina, April 9-12, 1989, Vol. 1, 146-151.

93. Lahouar, S. and Rahman, S. "Neural Networks and Their Applications in Electric Power Systems," *proceedings of the EPRI Conference on Expert System Applications for the Electric Power Industry*, Boston, Massachusetts, September 9-11, 1991.

94. Lawrence, J. "Data Preparation for a Neural Network," *AI Expert*, November 1991, 34-41.

95. LeClair, S.R. "Interactive Learning : A Multiexpert Paradigm for Acquiring New Knowledge," *SIGART Newsletter*, April 1989, no. 108, Knowledge Acquisition Special Issue, 34-44.

96. Lee, K.Y., Cha, Y.T. and Park, J.H., "Short-Term Load Forecasting Using an Artifical Neural Network," *IEEE/PES*, 1991 Winter Meeting, New York, New York, February 3-7, 1991.

97. Lee, S.J., Park, Y.M, Shim, J.U., Yoon, S.H., Yoon, M.C. and Lee, S.O. "Enhanced Expert System for Setting and Coordination of Protective Relays," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 290-294.

98. Levine, D.S. "Neural Population Modeling and Psychology: A Review," *Mathematical Biosciences*, Vol. 66, 1983.

99. Levine, D.S. "The Third Wave in Neural Networks," *AI Expert*, Vol. 4, No. 12, December 1989, 26-33.

100. Lippman, R.P. "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987.

101. Liu, C-C., Tsai, M-S., Mesa, V.N. and Hartwell, R. "Alleviation of Abnormal Feeder Operating Conditions with a Real-Time Expert System Environment," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 308-314.

102. Liu, C., Lee, S.J. and Venkata, S.S. "An Expert System Operational Aid for Restoration and Loss Reduction of Distribution Systems," *Proceedings of the Power Industry Computer Application Conference*, May 1987, 79-85.

103. Maa, C-Y., Chiu, C. and Shanblatt, M.A. "A Constrained Optimization Neural Network Technique for Economic Power Dispatch," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2946-2950.

104. Maher, M.L. and Allen, R. "Expert Systems Components," *Expert Systems for Civil Engineering: Technology and Application*, published by the American Society of Civil Engineers, New York, 1987, 3-14.

105. Matsuda, S. and Akimoto, Y. "Representation of Large Numbers in Neural Networks and Its Application to Economical Load Dispatching of Electric Power," *Proceedings of the 1989 International Joint Conference on Neural Networks (IJCNN)*, Washington, D.C., June 18-22, 1989, Vol. 1, 587-592.

106. Matsukawa, F. "A study on the load forecast methods," *Rec. Electr. Commun. Eng. Conversazione TokohuUniv. (Japan)*, Vol. 57, No. 4, June 1989, 225-226.

107. Matsuura, T. and Kawachi, F. "Expert System for Dissolved Gas Analysis for Power Apparatus Insulating Oil," *Kansai Electric Power Company, working paper,* 1990.

108. McCulloh, W.S. and Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* Vol. 5, 1943, 115-133.

109. McDermott, J. "Extracting Knowledge from Expert Systems," *Proceedings of the eighth IJCAI,* August 1983, 100-107.

110. Minsky, M. and Papert, S. **Perceptrons: An Introduction to Computational Geometry,** MIT Press, Cambridge, Massachusetts, 1969.

111. Mitra, G., editor, **Computer Assisted Decision Making,** Elsevier Science Publishers, 1986.

112. Mokhtari, S., Singh, J. and Wollenberg, B. "A Unit Commitment Expert System," *Proceedings of the Power Industry Computer Application Conference,* May 1987, 400-405.

113. Mori, H. "An Artificial Neural-Net Based Method for Estimating Power System Dynamic Stability Index," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems,* Seattle, Washington, July 23-26, 1991, 127-133.

114. Mori, H. "Application of a Revised Boltzmann Machine to Topological Observability Analysis," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems,* Seattle, Washington, July 23-26, 1991, 283-287.

115. Mori, H. and Tsuzuki, S. "Determination of Power System Topological Observability Using the Boltzmann Machine," *IEEE International Symposium on Circuits and Systems,* New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2938-2941.

116. Mori, H. and Tsuzuki, S. "Power System Topological Observability Analysis Using a Neural Network Model," *Second Symposium on Expert System Applications to Power Systems,* Seattle, Washington, July 17-20, 1989, 385-391.

117. Mori, H., Uematsu, H., Tsuzuki, S., Sakurai, T., Kojima, Y. and Suzuki K. "Identification of Harmonic Loads in Power Systems Using an Artificial Neural Network," *Second Symposium on Expert System Applications to Power Systems,* Seattle, Washington, July 17-20, 1989, 371-377.

118. Myers, W. "Expert Systems and Neural Networks Can Work Together: an interview with Elaine Rich," *IEEE Expert,* October 1990, 5-7.

119. Nakagawa, T., Hayashi, Y. and Iwamoto, S. "Neural Network Application of State Estimation Computation," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems,* Seattle, Washington, July 23-26, 1991, 188-194.

120. Neily, G., Barone, R., Josin, G. and Charney, D. "Joint VAR Controller Implemented in an Artificial Neural Network Environment," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems,* Seattle, Washington, July 23-26, 1991, 271-278.

121. NeuralWare Inc., Network Explorer User's Guide, Pittsburgh, PA, 1989.

122. NeXT Computer Inc., NeXT Operating System Software, Redwood City, California, 1990, Section 4-2.

123. NeXT Computer Inc., NeXTstep Concepts, Redwood City, California, 1990, Chapters 2, 3, 5, 6, 9.

124. NeXT Computer Inc., NeXTstep Reference, Redwood City, California, 1990, Volumes 1 & 2.

125. Nilsson, N.E. "Application of Computer Artificial Intelligence Techniques to Analyzing the Status of a Typical Utility Electrical Power Plant Systems," *IEEE/PES 1988 Summer Meeting*, Portland, Oregon, July 24-29, 1988.

126. Nishimura, K., Kawasaki, M. and Shimada, T. "Application of neural networks to power system state evaluation," *Proceedings of the 9th IFAC Workshop on Distributed Control Systems*, Tokyo, Japan, September 26-28, 1989.

127. Ostojic, D.R. and Heydt, G.T. "Transient Stability Assessment by Pattern Recognition in the Frequency Domain," *Proceedings of the IEEE Power Engineering Society 1990 Winter Meeting*, Atlanta, Georgia, February 4-8, 1990.

128. Pao, Y-H. and Sobajic, D.J. "A Perspective on Use of Neural-Net Computing in Training Simulator Design," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 210-214.

129. Pao, Y. and Sobajic, D.J. "Autonomous Feature Discovery for Critical Clearing Time Assessment," *Symposium on Expert Systems Application to Power Systems*, Stockholm-Helsinki, August 22-26, 1988, 5.22-5.27

130. Papadakis, M.E., Hatzjargyriou, N.D. and Gazidellis, D.K. "Interactive Data Management System for Power System Planning Studies," *IEEE Transactions on Power Systems*, Vol. 4, no. 1, Feb. 1989, 329-335.

131. Park, D.C., El-Sharkawi, M.A., Marks II, R.J., Atlas, L.E. and Damborg, M.J. "Electric Load Forecasting Using an Artificial Neural Network," *IEEE/PES 1990 Summer Meeting*, Minneapolis, Minnesota, July 15-19, 1990.

132. Park, K.S., **Human Reliability**, Elsevier, Amsterdam, 1987.

133. Peng, T.M, Hubele, N.F. and Karady G.G. "Conceptual Aproach to the Application of Neural Network for Short-Term Load Forecasting," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2942-2945.

134. Quinqueton, J. and Sallantin, J. "Algorithms for Learning Logical Formulas," *Proceedings of the eighth IJCAI*, August 1983, 476-478.

135. Rahman, S. "A Context Based Data Sanity Checking Algorithm and its Implementation," *Project proposal submitted to IIT/CIT*, June 1990.

136. Rahman, S. "Design Considerations for an Automated Utility Load Management System," Paper presented at *the International Symposium on Electronic Devices and Systems*, Kharagpur, India, 1987.

137. Rahman, S. and Baba, M. "An Integrated Load Forecasting-Load Management Simulator: Its Design and Performance," *IEEE Transactions on Power Systems*, Vol. 4, no. 1, Feb. 1989, 184-189.

138. Rahman, S. and Baba, M. "Software Design and Evaluation of a Microcomputer-Based Automated Load Forecasting System," *IEEE Transactions on Power Systems*, Vol. 4, no. 2, May 1989, 782-788.

139. Rahman, S. and Bhatnagar, R. "An Expert System Based Algorithm for Short Term Load Forecast," *IEEE Transactions on Power Systems*, Vol. 3, no. 2, May 1988, 392-399.

140. Rahman, S. and Lahouar, S. "Automated Data Sanity Checking Using Neural Networks," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 634-641.

141. Ranade, S.J., Uraguchi, M. and Tan, P. "Line Overload Expert System," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 188-194.

142. Reboh, R. "Extracting Useful Advice from Conflicting Expertise," *Proceedings of the eighth IJCAI*, August 1983, 145-150.

143. Ripps, D.L. "Adjustment of Experimental Data," *Chemical Engineering Progress Symposium Series*, Vol. 61, no. 55, 8-13.

144. Roberts, M. "Twelve Neural Network Cliches,", *AI Expert*, Aug. 1988, 40-46.

145. Romagnoli, J.A. and Stephanopoulos G. "Rectification of Process Measurement Data in the Presence of Gross Errors," *Chemical Engineering Science*, Vol. 36, no.11, 1981, 1849-1863.

146. Rosenblatt, F. **Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms**, Spartan, Washington, D.C., 1961.

147. Roth, M.W. "Neural Networks Technology and its Applications," *Johns Hopkins APL Technical Digest*, Vol. 9, No. 3, 1988, 242-253.

148. Rowan, D. "On-Line Expert Systems in Process Industries," *AI Expert*, August 1989, 30-38.

149. Rumelhart, D.E. **Parallel Distributed Processing**, MIT Press, Cambridge, Massachusetts, 1986.

150. Sahba, M. "A Method for Plausibility Checks and Data Validation in Power Systems," *IEEE Transactions on Power Systems*, Vol. 3, no. 1, Feb. 1988, 267-271.

151. Saitoh, H., Shoji, T., Yamamoto, H. and Toyoda, J. "Rule-Based Short Term Load Forecast Emphasizing Temperature Effects," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 44-50.

152. Saitoh, K. and Iwamoto, S. "Application of Neural Network Based Fuzzy Control to Power System Generator," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 144-148.

153. Sakaguchi, T. and Matsumoto, K. "Development of a Knowledge Based System for Power Restoration," *IEEE Transactions on Power Apparatus and Systems*, Vol. PAS-102, no. 2, Feb. 1983, 320-329.

154. Sakaguchi, T., Tanaka, H., Uenishi, K., Gotoh, T. and Sekine, Y. "Prospects of expert systems in power system operation," *Proceedings of the 9th Power Systems Computation Conference*, Cascais, Portugal, Aug 31-Sep 4 1987, 71-81.

155. Santoso, N.I. and Tan, O.T. "Neural-Net Based Real-Time Control of Capacitors Installed on Distribution Systems," *IEEE Transactions on Power Delivery*, Vol. 5, No. 1, January 1990, 266-272.

156. Sasaki, H., Kubokawa, J., Watanabe, M., Yokoyama, R. and Tanabe, R. "A Solution of Generation Expansion Problem By Means of Neural Network," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 219-226.

157. Schulte, R.P., Sheble, G.B., Larsen, S.L., Wrubel, J.N. and Wollenberg, B.F. "Artificial Intelligence Solutions to Power Systems Operating Problems," *IEEE Transactions on Power Systems*, Vol. PWRS-2, no. 4, Nov. 1987, 920-926.

158. Sendaula, M.H., Biswas, S.K., Eltom, A. and Parten, C. "Application of Artificial Neural Networks to Unit Commitment," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 256-260.

159. Silverman, E.B. and Klopp, G.T. "Neural Network-Based Expert System for Severe Accident Management," *proceedings of the EPRI Conference on Expert System Applications for the Electric Power Industry*, Boston, Massachusetts, September 9-11, 1991.

160. Singh, N.K. and Walther, E. "Knowledge Assisted State Estimation," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 173-179.

161. Smith, S. "Using an Expert System to Review Fluid Systems for Age-Related Degradation," *proceedings of the EPRI Conference on Expert System Applications for the Electric Power Industry*, Boston, Massachusetts, September 9-11, 1991.

162. Smith, W.M. and Braithwait, S.D. "DSM Program Monitoring," *EPRI Journal*, January/February 1989, 46-49.

163. Sobajic, D.J, Pao, Y-H., Njo, W. and Dolce, J.L. "Real-Time Security Monitoring of Electric Power Systems Using Parallel Associative Memories," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2928-2932.

164. Sobajic, D.J. and Pao, Y. "Artificial Neural-Net Based Dynamic Security Assessment for Electric Power Systems," *IEEE Transactions on Power Systems*, Vol. 4, No. 1, February 1989, 220-228.

165. Sobajic, D.J., Pao, Y. and Dolce, J. "On-line Monitoring and diagnosis of power system operating conditions using artificial neural networks," *IEEE International Symposium on Circuits and Systems*, Portland, OR, May 8-11, 1989, Vol. 3, 2243-2246.

166. Srinivasan, D., Liew, A.C. and Chen, J.S.P. "Short Term Load Forecasting Using Neural Network Approach," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 12-16.

167. Swarup, K.S. and Chandrasekharaiah, H.S "Fault Detection and diagnosis of Power Systems Using Artificial Neural Networks," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 102-106.

168. Tamhane, A.C. and Mah, R.S.H. "Data Reconciliation and Gross Error Detection in Chemical Process Networks," *Technometrics*, Vol. 27, no. 4, Nov. 1985, 409-422.

169. Tanaka, H., Matsuda, S. and Ogi, H. "Design and Evaluation of Neural Network for Fault Diagnosis," *Second Symposium on Expert System Applications to Power Systems*, Seattle, Washington, July 17-20, 1989, 378-384.

170. Thomas, R.J. and Ku, B-Y. "Approximation of Power System Dynamic Load Characteristics By Artificial Neural Networks," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 178-182.

171. Thomas, R.J., Sakk, E., Hashemi, K., Ku, B.Y. and Chiang, H-D. "On-Line Security Screening Using an Artificial Neural Network," *IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, May 1-3, 1990, Vol. 4, 2921-2924.

172. Tomsovic, K. and Ling, J.M. "Proposed Fuzzy Information Approach to Power System Security," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 427-432.

173. Tonn, B.E. "ARK, An Automated Knowledge Acquisition Tool," *presented at the EPRI Conference on Decision Support Methods for the Electric Power Industry*, Cambridge, MA, June 1990.

174. Tzeng, C.-C., Kuo, Y.-H., Kung, L.-Y. and Chen, T. "Maintaining Consistency in a Rule-Based Expert System," *Proceedings of the IEEE TENCON*, Bombay, India, November 1989, 63-66.

175. Uhrig, R.E. "Neural Networks and Their Potential Applications in Nuclear Power Plants," **Expert System Applications for the Electric Power Industry**, J.A. Naser, Ed., Hemisphere Publishing Corporation, NY, 1991, Vol.2 ,1435-1445.

176. Uhrig, R.E. and Guo, Z. "Use of Neural Networks to Identify Transient Operating Conditions in Nuclear Power Plants," *Proceedings of SPIE - International Society of Optical Engineers*, Orlando, Florida, 28-30 March 1989, Vol. 1095, Pt. 2, 851-856.

177. Utgoff, P.E. "Acquisition of Appropriate Bias for Inductive Concept Learning," *Proceedings of the National Conference on AI*, 1982, 414-417.

178. Vadari, S.V. and Venkata, S.S. "A Hybrid Artificial Neural Network/Artificial Intelligence Approach for Voltage Stability Enhancement," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 154-160.

179. Wagenbauer, M.P. and Brugger, H. "Model and Rule-Based Intelligent Alarm Processing," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 27-32.

180. Wang, P. "The Application of Intelligent Computers in Electric Power Industry," *Automated Electric Power Systems (China)*, Vol. 13, No. 2, March 1989, 3-9.

181. Waterman, D.A., **A Guide to Expert Systems**, Addison-Wesley, Reading, Massachusetts, 1986.

182. Weiss, S.M. and Kulikowski, C.A., **A Practical Guide to Designing Expert Systems**, Rowman & Allanheld, New Jersey, 1984.

183. Werbos, P.J. "Generalization of Backpropagation with Application to a Recurrent Gas Market Model," *Neural Networks*, Vol. 1, 1988, 339-356.

184. Werbos, P.J. "Maximizing Long-Term Gas Industry Profits in Two Minutes in Lotus Using Neural Network Methods," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 19, No. 2, March/April 1989, 315-333.

185. Wick, M.R. and Slagle, J.R. "An Explanation Facility for Today's Expert Systems," *IEEE Expert*, Spring 1989, 26-36.

186. Widrow, B. "An adaptive 'adaline' neuron using chemical 'memistors'," *Technical Report# 1553-2*, Stanford Electronic Laboratories, October 17, 1960.

187. Wollenberg, B.F. and Sakaguchi, T. "Artificial Intelligence in Power Systems Operations," *Proceedings of IEEE*, Vol. 75, no. 12, Dec. 1987, 1678-1685.

188. Yan, H.H., Chow, J-C., Kam, M., Fischl, R. and Sepisch, C.R. "Hybrid Expert System/Neural network Hierarchical Architecture for Classifying Power System Contingencies," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, Washington, July 23-26, 1991, 76-82.

189. Yiftah, S. "Neural Networks - Potential Application in the Nuclear Industry: Preliminary Comments and Reflections," *The Nuclear Societies of Israel Annual Meeting 1989*, Beer Sheva, Israel, 13 March 1989, 59-62A.

190. Zhang, Z.Z., Hope, G.S. and Malik, O.P. "Expert Systems in Electric Power Systems, A Bibliographical Survey," *IEEE Transactions on Power Systems*, Vol. 4, No. 4, November 1989, 1355-1362.

191. Zwingelstein, G., Masson, M.H., Dubuisson B., Pavard, M. and Mazalerat, J.M. "The Application of Neural Networks for the Predictive Maintenance of Nuclear Power Plants," *Third Symposium on Expert System Application to Power Systems*, Tokyo, Japan, April 1-5, 1991, 567-573.

Vita

Saher Lahouar was born in Hammam-Sousse, Tunisia, on October 19th, 1964. In 1981, he passed the national Baccalaureate exam with distinction and was selected as a member of an elite group to participate in a program of technology transfer from the U.S. to Tunisia.

Mr. Lahouar attended the University of Rochester in Rochester, New York, where he obtained his Bachelor of Science degree in Electrical Engineering in 1985. He received his Master of Science degree in Computer Engineering from Rensselaer Polytechnic Institute, Troy, New York, in 1987. Subsequently, he joined the Virginia Polytechnic Institute and State University in Blacksburg, Virginia an embarked on a Ph.D. program in April 1988.

Mr. Lahouar is interested in a wide variety of topics centered around machine intelligence and human/computer interfacing. Throughout his academic and research career, he has acquired a solid experience with numerous computer platforms, operating systems and programming languages. Also, he is a member of the Institute of Electrical and Electronic Engineers (IEEE), Systems Man and Cybernetics society (SMC), and the Tunisian Scientific Society (TSS).