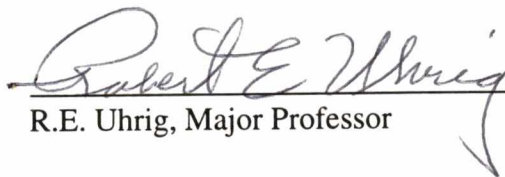
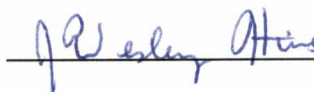


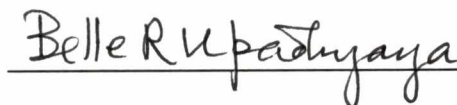
To the Graduate Council:

I am submitting herewith a thesis written by Darryl Wrest entitled. "Instrument Surveillance and Calibration Verification Through Plant Wide Monitoring Using Autoassociative Neural Networks." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Nuclear Engineering.

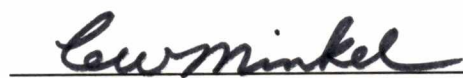

R.E. Uhrig, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


Associate Vice Chancellor and
Dean of the Graduate School

Instrument Surveillance and Calibration Verification
Through Plant Wide Monitoring
Using Autoassociative Neural Networks

A Thesis

Presented for the Master of Science Degree

The University of Tennessee, Knoxville

Darryl J. Wrest

August 1996

DEDICATED TO

My daughters,

Brittany and Sabrina,

who remind me what the good things in life are
and continue to inspire me to always do my best.

And to my wife, Michelle,

who made it possible for me to focus my efforts
on graduate school.

ACKNOWLEDGMENTS

Through my course of study at the University of Tennessee there were many people who helped make my educational experience a rewarding one. My gradatude goes out to the members of my Thesis Committee, Dr. Robert Uhrig, Dr. Wesley Hines, and Dr. Belle Upadhyaya. With their encouragement and consultation, I was able to achieve theses results.

I owe special thanks to Dr. Wesley Hines, who painstakingly took the time to guide me through this work. He has been a friend, as well as an advisor.

I would like to thank Dr. Tom Kerlin for his faith in me and support through out my course of studying by offering me a generous Fellowship.

Finally, I would like to thank the Institute of Nuclear Power Operations (INPO) for providing me with the Fellowship so that I could carry out this study.

This project was sponsored by the DOE through Sandia National Laboratories under contract document AQ-6982. I would also like to thank Florida Power Corporation for supplying the data for the project.

ABSTRACT

The approach to instrument surveillance and calibration verification (ISCV) through plant wide monitoring proposed in this paper uses an autoassociative neural network (AANN) which utilizes digitized data presently available in the Safety Parameter Display computer system from Florida Power Corporations Crystal River #3 nuclear power plant to verify correct sensor calibration during operation.

An autoassociative neural network is one in which the outputs are trained to emulate the inputs over an appropriate dynamic range. The relationships between the different variables are embedded in the weights by the training process. As a result, the output can be a correct version of an input pattern that has been distorted by noise, missing data, or faulty sensors. Plant variables that have some degree of coherence with each other constitute the inputs to the network. Once the network has been trained with normal operational data it has been shown to successfully monitor the selected plant variables to detect sensor drift or failure by simply comparing the network inputs with the outputs.

The Sequential Probability Ratio Test (SPRT) is used as a decision logic module to detect incipient changes in the sensor values using the residual signal between the network estimate and the actual sensor signal.

The entire plant wide AANN ISCV system was designed and implemented on a Personal Computer platform. The system simultaneously monitors 78 plant variables for sensor drift as well as gross sensor failures. The system proved to be very sensitive to

sensor drift with detection setpoints on the order of 0.1% to 2.5% of the full scale range of the instrument.

The AANN method of monitoring many variables not only detects sensor failures, it clearly indicates the signal channel in which the signal error has occurred.

TABLE OF CONTENTS

1.0 INTRODUCTION	1
1.1 Objectives of the Present Study	1
1.2 Related Studies	5
1.3 Background.....	6
1.4 Overview of the Methodology	7
1.5 Software and Computing Environment.....	8
1.6 Organization of the Thesis	9
2.0 DESCRIPTION OF NEURAL NETWORKS.....	10
2.1 Introduction	10
2.2 Artificial Neural Networks	11
2.2.1 Multilayer Perceptron (MLP)	13
2.3 Learning and Recall	13
2.3.1 Learning Methods	14
2.3.2 Recall.....	15
2.4 Activation Functions	15
3.0 TRAINING ALGORITHMS.....	20
3.1 Introduction	20
3.2 Optimization Techniques	23
3.2.1 Gradient Decent	24
3.2.2 Gradient Decent With Line Search	25
3.2.3 Conjugate Gradient Method	26
3.2.4 Newtons Method	28

3.2.5 Gauss-Newton Method	29
3.3 Backpropagation	30
3.3.1 Modified Backpropagation	32
3.3.1.1 Adaptive Learning Rate	33
3.3.1.2 Momentum	33
3.4 Scaled Conjugate Gradients	33
3.5 Levenberg Marquardt	35
3.6 Summary and Training Paradigm Selection	36
4.0 SEQUENTIAL PROBABILITY RATIO TEST ALGORITHM	38
4.1 Introduction	38
4.2 Decision Algorithm	39
4.3 Decision Rules	41
4.3.1 Wald's two-sided criterion	42
4.3.2 Chien's one-sided criterion	43
4.4 SIMULINK Implementation	44
5.0 ISCV SYSTEM DEVELOPMENT	49
5.1 Introduction	49
5.2 Data Selection and Processing	50
5.2.1 Data Selection	50
5.2.2 Data Processing	51
5.3 Parameter Selection	52
5.3.1 Engineering Judgment	52
5.3.2 Correlation Coefficients	53
5.3.3 Sensitivity Analysis	54
5.4 Network Architecture	58
5.4.1 Network Architecture Selection	58
5.4.1.1 Parameter Selection	60
5.4.1.2 Network Training for Architecture Selection	61
5.4.1.3 Network Testing for Architecture Selection	63
5.4.1.4 Network Testing Performance	64
5.4.1.5 Final Network Selection	72

5.4.2 Three Hidden Layer Feature Detection Network	72
5.4.2.1 Selection of Hidden Layer Nodes	75
5.4.2.2 Noise Filtering.....	78
5.5 Network Training	78
5.5.1 Training Algorithm Selection.....	79
5.5.2 Singular Value Decomposition (SVD).....	80
5.5.3 Optimal Training Error.....	82
5.5.3.1 Statistical Technique	82
5.5.3.2 Cross Validation Training.....	82
5.5.3.3 Final Network Training Errors and Comparisons	83
5.5.4 Robust Training Technique	85
5.5.5 Training and Validation.....	87
5.6 SPRT Decision Logic Module	88
5.7 System Integration.....	89
5.7.1 Individual Networks.....	90
5.7.2 Integrated System	93
5.7.3 Redundant Sensors.....	93
5.8 System Testing	95
5.8.1 Testing Data Selection	95
5.8.2 System Testing Guidelines.....	96
5.8.3 Results.....	97
5.8.3.1 Sensitivity Analysis and Robust Training	97
5.8.3.2 SIMULINK Simulation Example.....	98
5.8.3.3 Parameter Selection Refinement Techniques.....	103
5.8.3.4 Drift Detection	103
5.8.3.5 Gross Error Detection Techniques.....	105
5.8.3.6 Noise Detection Techniques	113
5.8.4 Overall System Performance	118
6.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK ..	120
6.1 Conclusions	120
6.2 Recommendations for Future Work	121
6.2.1 SPRT Optimization Algorithm	121
6.2.2 Residual Whitening	122
6.2.3 Hidden Layer Node Search	122
6.2.4 Optimal SVD Noise Elimination Parameter Value	123
6.2.5 Robust Training Samples.....	123
6.2.6 Missing Sensor Replacement	123

6.2.7 Decision Logic System.....	124
6.2.7.1 Fault Detection and Isolation.....	124
6.2.8 Model Changes and Adaptive Network Tuning	126
BIBLIOGRAPHY.....	128
APPENDICES.....	134
A Selected Tables and Figures	135
B Matlab Computer Code.....	158
VITA.....	198

List of Tables

5.1 Network Correlation Coefficient Values	54
5.2 Network AA Parameters - Secondary Loop A	55
5.3 Network AB Parameters - Secondary Loop B	56
5.4 Network B Parameters - Primary	57
5.5 Correlation Coefficients Matrix for Initial 13 Parameter Network	62
5.6 Network Training Records	63
5.7 Total Network RMS Error and Signal #5 RMS Error	64
5.8 Number of Nodes per Layer	77
5.9 Network SSE Training Goals	84
5.10 Redundant Sensor Networks	95
5.11 Drift Error Detection Results for a Drift of 0.1% per day of the Instruments maximum scale deflection value	104
5.12 Gross Error Detection Results	112
5.13 Bias and Noise Detection Logic Table	117
A5.1 Network AA Correlation Coefficient Matrix	136
A5.2 Network AB Correlation Coefficient Matrix	138
A5.3 Network B Correlation Coefficient Matrix	140

List of Figures

2.1 Schematic of a Neuron	11
2.2 Schematic Representation of a Processing Element	12
2.3 General Topology of a Multilayer Perceptron Network	14
2.4 Linear Transfer Function	16
2.5 Ramp Activation Function	17
2.6 Step Activation Function	18
2.7 Sigmoidal Activation Function	18
2.8 Hyperbolic Tangent Activation Function	19
3.1 Example of a nonlinear error surface for a single nonlinear neuron	21
3.2 Graphical Depiction of Steepest Decent with Line Search	25
3.3 Graphical Depiction of Conjugate Gradient Method	28
4.1 Simplified Block Diagram of the ISCV System	38
4.2 SPRT Functional Block Diagram	39
4.3 Graphical Display of SPRT Reasoning	42
4.4 Block of Four SPRTs	45
4.5 SPRT Block	45
4.6 Chien's One Sided Test	46
4.7 Lambda Calculation	47
4.8 New Lambda.....	47
5.1 Simplified AANN Signal Error Detection Schematic	50
5.2 Artificially created step error fault in signal A321	65
5.3 Artificially created ramp error fault in signal A321	66
5.4 Network I Step Error Estimation and Residual between the Actual Signal and the Networks Estimate	68

5.5 Network II Step Error Estimation and Residual between the Actual Signal and the Networks Estimate	69
5.6 Network III Step Error Estimation and Residual between the Actual Signal and the Networks Estimate	70
5.7 Network III Ramp Error Estimation and Residual between the Actual Signal and the Networks Estimate	71
5.8 Architecture of a Three Layer "Feature" Detection AANN	73
5.9 Nonlinear Logistic Transfer Function	75
5.10 Example of the AANNs Filtering Abilities, Tag R222-Reactor Outlet Pressure	79
5.11 Cross Validation Training Error Curves	84
5.12 Non-Robust Training Set (input = target)	86
5.13 Robust Training Set	86
5.14 Exploded View of an AANN block	90
5.15 Network AA and AB SIMULINK ISCV System Block (24 parameters modeled)	91
5.16 Network B SIMULINK ISCV System Block (22 parameters)	92
5.17 Complete Integrated AANN ISCV System.....	94
5.18 Network B Sensitivity Analysis	99
5.19 Network AA Sensitivity Analysis	100
5.20 Network AB Sensitivity Analysis	101
5.21 ISCV System Detecting a Fault Drift High in the Steam Generator Level when a Drift of 0.1% per day of the Instruments Maximum Scale Deflection is Introduced into the Sensor Signal	106
5.22 ISCV System Detecting a Fault Drift High in the Reactor Pressure when a Drift of 0.1% per day of the Instruments Maximum Scale Deflection is Introduced into the Sensor Signal	107

5.23	ISCV System Detecting a Fault Drift High in the Reactor Loop Flow when a Drift of 0.1% per day of the Instruments Maximum Scale Deflection is Introduced into the Sensor Signal	108
5.24	ISCV System Detecting a Fault Drift Low in the Main Steam Pressure when a Drift of 0.1% per day of the Instruments Maximum Scale Deflection is Introduced into the Sensor Signal	109
5.25	ISCV System Detecting a Fault Drift Low in the Reactor Outlet Temperature when a Drift of 0.1% per day of the Instruments Maximum Scale Deflection is Introduced into the Sensor Signal	110
5.26	ISCV System Detecting a "Gross" Fault Low in the Reactor Outlet Pressure when a Fault Equal to the Instruments Minimum Scale Deflection is Introduced into the Sensor Signal at Time Stamp 500	114
5.27	Response of "Gross" Fault Initiated at Time Stamp 500 for a Typical Companion Network Variable	115
5.28	Block Schematic of a Noise SPRT Detection System	117
6.1	Fault Detection and Isolation (FDI) System	125
A5.1	Network B - 5 Fault-Free Weeks of Data	142
A5.2	Network B - Non-Robust Training Set Fault Hypothesis	147
A5.3	Network B - Robust Training Set Fault Hypothesis	151
A5.4	Network Residuals for Gross Fault in Sensor 9	155

List of Abbreviations

- AANN: Autoassociative Artificial Neural Network
- ANN: Artificial Neural Network
- ANS: Artificial Neural System
- BP: Backpropagation
- B&W: Babcocks & Wilcocks
- CG: Conjugate Gradients
- FDI: Fault Detection and Isolation
- FPC: Florida Power Corporation
- FW: Feed Water
- HR: Hour
- HST: History Stack Adaptation
- ISCV: Instrument Surveillance Calibration and Verification
- LB: Pound
- LM: Levenberg Marquardt
- MLP: Multilayer Perceptron
- MSE: Mean Squared Error
- NLPCA: Non Linear Principle Component Analysis
- NN: Neural Network
- NRC: Nuclear Regulatory Agency
- NSSS: Nuclear Steam Supply System
- OTSG: Once Through Steam Generator
- PCA: Principle Component Analysis
- PE: Processing Element

- PPCS: Plant Process Computer System
- PWR: Pressurized Water Reactor
- RMS: Root Mean Squared
- SCG: Scaled Conjugate Gradients
- SG: Steam Generator
- SPRT: Sequential Probability Ratio Test
- SSE: Sum Squared Error
- SVD: Singular Value Decomposition

Chapter 1

Introduction

1.1 Objectives of the Present Study

The objective of the present study is to develop and document an Instrumentation Surveillance and Calibration Verification (ISCV) plant wide monitoring system using autoassociative neural networks for Florida Power Corporations Crystal River #3 Nuclear Power Station.

In large power generating systems and process control systems, outputs from many different channels are used in control systems, safety critical systems, and for plant state identification. It is necessary to validate these outputs to increase the reliability of operator decisions and automatic plant operations.

A common problem faced by designers of computer based systems intended to monitor the performance of a dynamic process with the objective to assure operation within specifiable constraints is that of sensor reliability. Since the output of sensors provides the only objective source of information for decision making, it is essential that the condition of these sensors be known. The dual problem of sensor validation and faulted sensor replacement must be considered as an integral part of the design of a modern monitoring, diagnostic, or control system. The technical issues that must be addressed to assure fault-tolerant sensor based monitoring systems include the following: validation (is the sensor properly functioning and/or has the component deteriorated); failure accommodations (can the system adequately function with degraded sensors); and system diagnostics (what has failed). It is important that the data generated by these sensors be processed to extract the information needed to assure that

the sensors and the systems they are monitoring remain within their intended operating specifications.

Calibration requirements for safety related and technical specification nuclear power plant instrumentation are time driven. Because of these periodic requirements, calibrations is often unnecessary and damaging to the sensors. As utilities desire to move to 24 month fuel cycles, there is a need for condition directed calibrations as an alternative to time directed instrument calibrations.

The objective of this study is to develop and document an intelligent system using artificial neural networks that continuously monitors the calibration condition of safety related and technical specification sensors. In this manner, the condition of a sensor would always be known. Thus, a sensor would be calibrated only when needed, instead of on a preset time table. The result in calibration reduction would be a great cost savings to the nuclear utility industry.

The system will be designed for a system with limited (or no) redundant sensors. This is essential for nuclear plants such as Crystal River #3 which a limited number of redundant sensors. The neural network based system will use a pattern recognition techniques in which the inputs to the network are diverse, but correlated signals.

Signal validation and process monitoring problems require the prediction of one or more (in the case of plant wide monitoring) process variables in a system. The prediction of process sensor outputs is performed traditionally using either physical or empirical models. Empirical models require knowledge of all variables having a significant effect on the signal to be modeled. The model characterizes a critical signal as a function of a set of other measurements which influence the behavior of the variable considered [Upadhyaya, August 1988], [Upadhyaya, September 1988], [Upadhyaya, Holbert and Eryurek, 1989].

Analytical redundancy based prediction assumes a fixed mathematical structure for characterizing steady state or dynamic relationships amongst process variables. The generation of an accurate model requires an effort which is proportional to the size and complexity of the system. This is a major drawback for a nuclear plant wide monitoring

system using a mathematical model; the model is possibly too complex to construct. Even if a model could be constructed, periodic changes in the model would be required to adapt to the changing plant conditions as the plant ages. This would require a tremendous effort.

A new method using neural networks (distributed parallel processing) offers several advantages in signal validation when compared to the traditional model-based techniques.

1. It is not necessary to define a functional form relating a set of process variables.
2. The functional form defined by an Artificial Neural System (ANS) is implicitly non-linear, which allows for complex non-linear mappings.
3. Neural networks do not require the detailed system specifications for the models.
4. Neural networks are by far more fault tolerant than the traditional model-based techniques [Upadhyaya, Eryurek and Mathai, 1989]. Unlike traditional model based programs, neural network models can give high performance even when there is a failure in the network structure, such as missing processing elements, or faulty weight connections [Sequin and Clay, 1990].
5. Once the neural network has been trained to represent the system model, recall is very fast. This makes neural networks excellent for real time applications.

Neural networks are intrinsically parallel and non-algorithmic methods; these features of neural networks make real time processing of data feasible. Some of the applications of neural networks in a nuclear power plant include nuclear fuel management, multi-sensor information fusion, sensor validation, control, thermodynamic modeling, vibration analysis, severe accident management, eddy current interpretation, plant-wide monitoring, transient identification, and pattern recognition for diagnostics of plant malfunctions [Uhrig, 1989, 1994].

In the present study, artificial neural networks are used in a plant wide Instrument Surveillance and Calibration System (ISCV) in which the an Autoassociative

Artificial Neural Network (AANN) is used as a tool for estimating analog plant signals. Thus, the neural network "model" is applied for interpolating process variables, and thus for estimating signals during plant operation. By comparing the neural networks estimate of the signal with the actual signal, we can determine if a sensor or channel anomaly exists.

The method chosen to perform this function is the Sequential Probability Ratio Test (SPRT). The difference between the actual and estimated value is called the residual. The residual is processed by the SPRT module to determine when the residual is statistically significant to hypothesize the sensor as faulted.

The objectives of this study are summarized as follows:

- Develop and document an Instrument Surveillance and Calibration Verification system for plant wide monitoring of safety related parameters in a Nuclear power plant using autoassociative neural networks.
- Validation of safety critical sensor outputs.
- The calibration condition of the monitored sensors would always be known.
- System monitors for sensor anomalies such as:
 - bias or slow drift
 - spurious high/low values occurring at random or in regular time intervals
 - high or low signal levels
 - increased sensor noise
 - total sensor failure
- Determination of a network selection technique for system process variables.
- Develop and implement of a multilayer robust network.
- Selection of an optimum network training algorithm.
- Development of a robust training technique.
- Selection and implementation of a fault detection algorithm.

1.2 Related Studies

The use of AANN's for plant wide monitoring was developed by the University of Tennessee and reported in NUCLEAR TECHNOLOGY [Upadhyaya and Eryurek, 1992]. This work, using data from the Experimental Breeder Reactor II, has demonstrated the practicality of this application. Related work includes monitoring of the Borssele Nuclear Power Plant using AANN techniques [Nabeshima, Susuki and Turkan, 1995]. Similar work using AANN applied to chemical process systems have also been reported [Dong and McAvoy, 1994][Kramer, 1991, 1992].

Related work in signal validation includes a parallel module based system that uses four different signal validation algorithms (generalized consistency checking, process empirical modeling, Kalman filtering, and artificial neural networks) to compensate for prediction errors during transient operating conditions [Erbay and Upadhyaya, 1995]. Each of the modules produces an estimate of a sensor output to be validated. The final sensor condition is decided by a system executive module which incorporates fuzzy logic to reach a decision on the sensors condition.

Sensor channel drift monitoring using an inferential neural network technique was studied by Black [1996]. The artificial neural network modeling technique utilizes a genetic algorithm search approach supplemented with standard linear correlations to empirically determine appropriate combinations of available input variables to perform each desired sensor modeling.

Olvera [1993] developed an ISCV system for redundant sensor monitoring in a nuclear power plant. The system uses a consistency checking algorithm and an empirical modeling scheme to provide an additional diverse point of redundancy. The sequential probability ratio test is used to track and predict calibration drift.

Wang and Holbert [1995] have developed an artificial neural network based system for signal validation and fault diagnosis. It uses a neural network realization of a linear least mean square estimator for sensor validation.

A study utilizing fuzzy logic for power plant signal validation has also been performed for systems having at least two redundant sensors [Holbert, Heger, and Ishaque, 1995].

1.3 Background

Traditional approaches to instrument calibration at nuclear power plants are expensive both in labor and money. These approaches vary from calibration by replacement, to transfer calibration using standard instruments. Technical Specifications require specific instruments be calibrated on time tables that coincide with the original fuel cycle of the plant. These calibrations require that the instrument be taken out of service and be falsely-loaded to simulate actual in-service stimuli. This can lead to damaged equipment and incorrect calibrations due to adjustments made under non-service conditions. The end result is that many instruments are unnecessarily calibrated. While proper adjustment is vital to maintaining proper plant operation, a condition based technique is desirable.

As utilities desire to move to 24 month fuel cycles, there is a need for performance based calibration requirements. When implementing performance based calibrations, the instruments are calibrated only when they are determined to be out of calibration. Monitoring instruments for calibration performance will allow nuclear utilities to reduce the efforts necessary to assure the instruments are calibrated. Benefits include an industry wide cost savings, less outage time, decreased radiation exposure to workers, a decrease in low level radioactive waste, decrease in work scheduling, and easier compliance with NRC Generic Letter 91-04 for extending calibration intervals. The EPRI/Utility On-Line Monitoring Working Group estimates an industry wide cost savings of \$40M to \$290M over the next 20 years depending on the values applied to indirect benefits.

1.4 Overview of the Methodology

This present study utilizes a MultiLayer Perceptron (MLP) feedforward Autoassociative Neural Network (AANN) for the estimation of the process variables in a nuclear power plant. The weights in the network are optimized using a modified Backpropagation (BP) algorithm to improve robustness and speed convergence. The ability of a modified backpropagation algorithm to learn arbitrary nonlinear mappings from inputs to outputs and the fault tolerance properties of a MLP network make it an ideal tool for plant wide sensor monitoring.

A neural network architecture is defined by the number of layers in the network, the number of nodes in each layer, the activation functions in each layer and the interconnections between the nodes. The standard feedforward multilayer perceptron was chosen because it is proven that it is a universal approximator [Cybenko, 1989], [Hornick, Stinchcombe, and White, 1989] and it operates globally. This means that a single nonlinear hidden layer MLP can approximate any input-output mapping of m real variables to any desired degree of accuracy and that this mapping can cover the entire input space over a compact range. However, while one hidden layer may be sufficient to provide such a mapping, there is no evidence in the literature that one hidden layer is optimal for every modeling task. In some situations, multiple hidden layers may provide a superior mapping than a single hidden layer. This is the case in this study where a three hidden layers network was found to be optimal.

In general, the multilayer perceptron is capable of performing arbitrary mappings between inputs and outputs without knowing the complete system specifications - a black box approach to modeling. Both steady state and transient behavior data can be incorporated into the network during the training phase. Neural networks allow us to easily update the model by renewing the network connection weights through re-training when new data becomes available.

An autoassociative neural network is one in which the outputs are trained to emulate the inputs over an appropriate dynamic range. Many plant variables that have some degree of correlation with each other constitute the inputs. During training to

make each output equal to the corresponding input, the interrelationship between the variables is embedded in the connection weights. In effect, each output is dependent upon all correlated inputs. As a result, any specific output shows virtually no change when the corresponding input pattern has been distorted by noise, missing data, or nonlinearities. This characteristic allows the AANN to detect drift or failure by comparing the sensor output (i.e., the network input) with the corresponding network estimate.

1.5 Software and Computing Environment

MATLAB[1993] and its toolboxes have been used to implement the ISCV system. This section will give an overview of MATLAB which is a software product that forms a technical computing environment that can run on many platforms including windows based personal computers (Windows, DOS, Linux), Macintosh, Sun, DEC, VAX and Cray. The main reason for this choice of software is that most MATLAB functions are customizable. This gives flexibility in creating specialized architectures and training paradigms. MATLAB can be used for simulation and C code can be generated to create real-time executables.

MATLAB is a language in itself and can be used at the command line or in MATLAB programs called m-files. It uses standard programming constructs such as IF, else, break, while, etc. and has file input/output capabilities such as open, read, write formatted, etc. It also has functionality for string manipulation, debugging, graphical user interface building, and has on-line help routines for all functions.

MATLAB can be linked with C and FORTRAN programs or through external interface libraries, or can be converted to C code. It can also be called from C and FORTRAN programs to perform mathematical computations. Dynamic data exchange allows MATLAB to communicate with other Windows applications and to communicate with data acquisition cards.

One of the most important and useful toolboxes is SIMULINK [1993]. SIMULINK is an environment for modeling, analyzing, and simulating linear and nonlinear dynamic systems. SIMULINK provides a graphical user interface that

supports click and dragging of blocks that can be connected to form complex systems. SIMULINK's Real Time Workshop can be used for rapid prototyping, embedded real time control, real-time simulation, and stand-alone simulation. This toolbox automatically generates stand-alone C code which leads to run-time executions that are 2 to 7 times faster than the MATLAB code implementation.

The entire ISCV system presented in this study including signal processing, network design and training, total system integration and testing has been implemented using a PC desktop computer (Pentium 90, with 24 Mbytes of RAM). This demonstrates the low cost utility of the ISCV system. All MATLAB files were used as interpreted script or .m files. For increased execution speed and memory conservation, compiled C code can be generated using the MATLAB C compiler.

1.6 Organization of the Thesis

The first chapter of this thesis presents the objectives of this study, background explaining the benefits of this study and the resulting ISCV system, an overview of the methodology, and the computing environment and software used in the study. Chapter 2 includes a description of neural networks including some of the important definitions used in Artificial Neural Systems (ANS). Chapter 3 discusses network training and presents three multilayer perceptron training algorithms; BackPropagation (BP), Scaled Conjugate Gradients (SCG), and Levenberg Marquardt (LM). The Sequential Probability Ratio Test (SPRT) detection algorithm and its implementation are discussed in Chapter 4. The main focus of the study is presented in Chapter 5. This includes data selection and processing, parameter selection techniques, network architecture selection, network training algorithm selection and training criteria, and system integration and testing. Conclusions and recommendations for future work are given in Chapter 6.

Chapter 2

Description of Neural Networks

2.1 Introduction

The human brain is probably the most complex computing device known to man. It is capable of thinking, remembering, decision making, problem solving, and pattern recognition. There have been many attempts to emulate brain functions with computer models, many of them very complex models. However, they all pale in comparison to the complex functioning of the human brain.

The fundamental cellular unit of the human brain is the *neuron*. Figure 2.1 shows a crude sketch of a neuron. It is a simple processing element that receives and combines signals from other neurons through input paths called *dendrites*. If the combined input signal is strong enough, the neuron "fires" producing an output signal along the *axon* that connects to dendrites of many other neurons. Each signal coming into a neuron along a neuron passes through a *synaptic junction*. This junction is an infinitesimal gap in the dendrite that is filled with neural transmitter fluid that either retards or accelerates the flow of electrical charges. This transfer of information across the synaptic junction is chemical in nature, but has electrical side effects.

The brain and the spinal cord consist of approximately 15 billion highly interconnected neurons [Tsoukalas and Uhrig, 1996]. The adjustment of the impedance (or conductance) of the synaptic gap is a critically important process. These are the adjustments that lead to memory and learning. As the synaptic strengths of the neurons are adjusted, the brain learns and stores information.

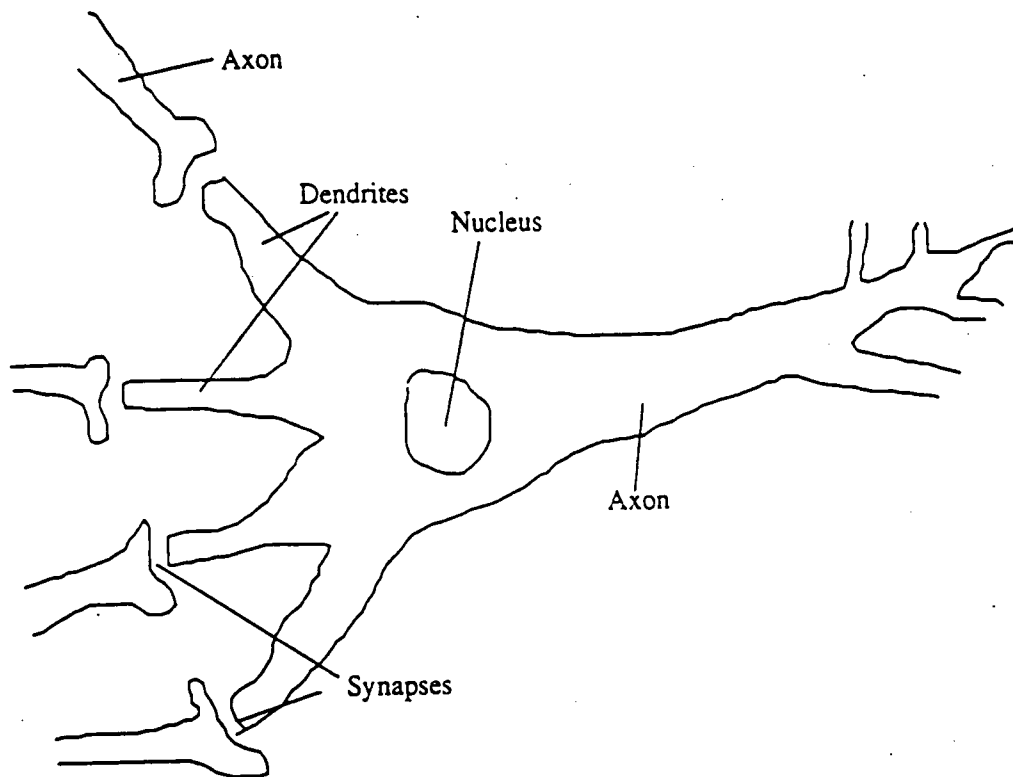


Figure 2.1 Schematic of a neuron.

2.2 Artificial Neural Networks

An artificial neural network can be defined as [Tsoukalas and Uhrig, 1996]:

" A data processing system consisting of a large number of simple, highly interconnected processing elements in an architecture inspired by the structure of the cerebral cortex of the brain."

In an artificial neural network, the unit analogous to the neuron is referred to as a "Processing element" (PE). A schematic of a processing element can be seen in Figure 2.2 on the following page. Each of the paths carrying an input to the processing element has an adjustable weight which is multiplied by the input value in the path. The combined input to the PE is then summed up where an activation function is applied to the summation value. The output from the activation function is then the single output of the PE. Activation functions are also called thresholding or "squashing" functions.

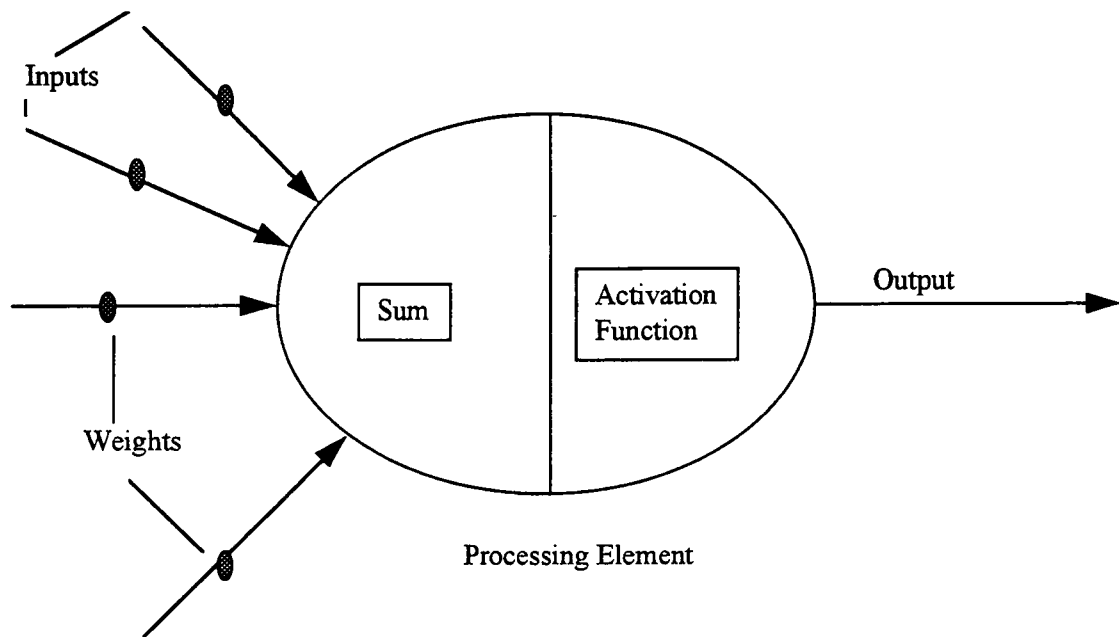


Figure 2.2 Schematic representation of a processing element.

Linear, sigmoidal, and step activation functions are discussed in section 2.5. The output of a PE is connected to input paths of other PEs through the connection weights which are similar to the synapses. The output of each PE is multiplied by the corresponding connection weights before reaching another PE. Thus, PEs combine the modified activation values from other elements.

There are two types of connections between PEs, excitatory and inhibitory. Excitatory connections increase the PEs activation level, while inhibitory connections decrease the activation level.

An artificial neural network consist of many processing elements interconnected in the above manner. In general, PEs are grouped into layers which are designed in sequence from input to output layers. The weight connections are built between the neighboring layers. The input layer, called the input buffer (because it does not have

input weights or an activation function) receives the input information and the output layer transmits the information to the outside process. All other layers, known as "hidden layers" receive and transmit information within the network. The hidden layers with non-linear activation functions are responsible for the non-linear mapping capabilities of a neural network.

A neural network is said to be autoassociative if the input and the output signals are identical. If they are not, it is called a heteroassociative network.

A typical artificial neural network is said to be "fully-connected" when there is a connection between each of the artificial neurons in any given layer with each of the artificial neurons in the next layer. When there are no lateral connections between artificial neurons in a given layer and no feedback connections to previous layers, the network is said to be a feedforward network.

2.2.1 Multilayer Perceptron (MLP)

The multilayer perceptron network [Haykin, 1994] is one of the most important historical developments in neural computing technology. It is a powerful mapping network that has been used in the majority of neural network applications [Masters, 1993]. Figure 2.3 shows the general topology of a MLP network.

The multilayer perceptron is a multilayer, feedforward artificial neural network. The nonlinear mapping capabilities of these networks come from the nonlinear activation functions used by the processing elements.

2.3 Learning and Recall

Artificial neural networks perform two main functions: learning and recall. Learning is the process of modifying the connection weights between the artificial neurons in an attempt to minimize the error between the actual and the desired output of the neural network when an input signal is presented to the input buffer of the network. Recall is the process of producing an output response in accordance with the network structure due to an input stimulus. Recall is an integral part of the learning process since a desired

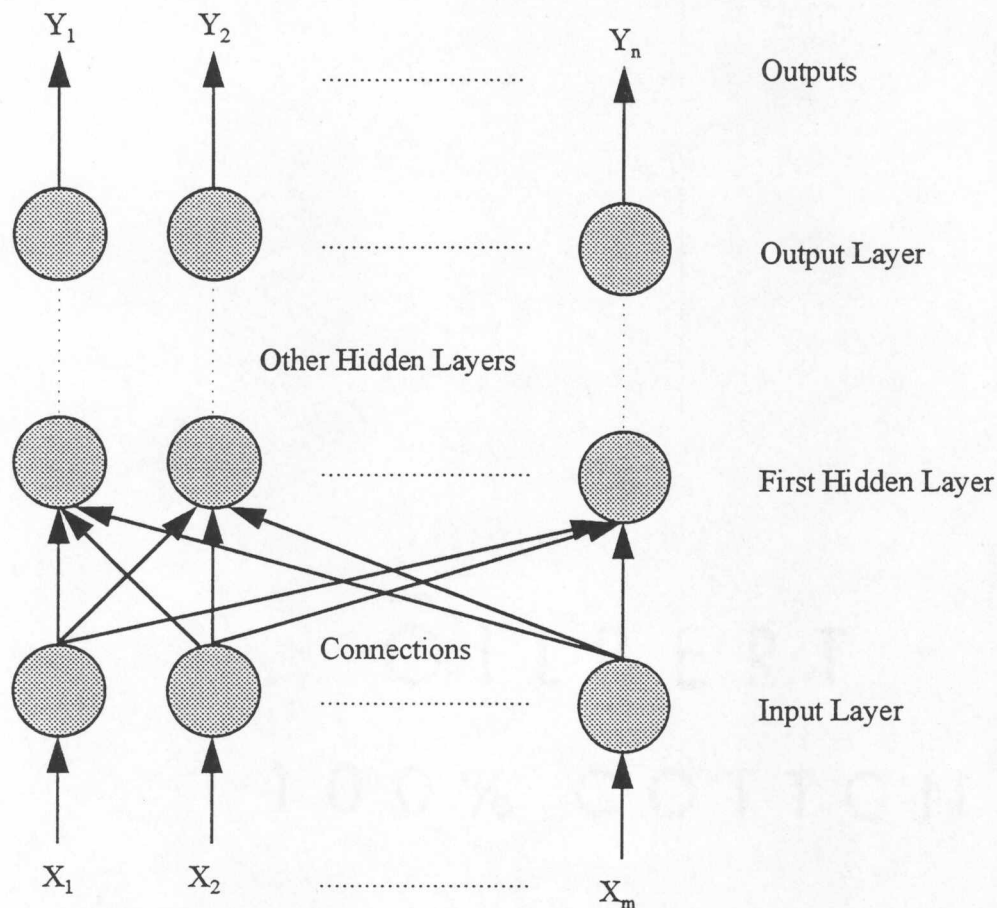


Figure 2.3 General topology of a Multilayer Perceptron Network.

response to the network must be compared to the actual output response of the network to create an error function.

2.3.1 Learning Methods

Several types of learning are commonly used. In *supervised learning*, the desired output pattern is supplied by the user. The network is trained to give the desired response to a specific input stimulus. In *unsupervised learning*, there is no desired output given, the response based on the networks ability to organize itself. Finally, in *graded learning* the output is "graded" as good or bad, and the connection weights are adjusted in occurrence with the grade.

Supervised training was used in this research. Training patterns consist of input/output relationships we would like to model with the neural network. Three supervised training algorithms will be discussed in Chapter 3.

2.3.2 Recall

Recall is the process of propagating the input pattern presented to the input buffer through the network to the output layer. The input buffer receives the input pattern and fans it out to the next layer without any processing. Each element of the first layer receives the output from all outputs of the first layer, and processes the combined inputs according to the procedure mentioned in section 2.2, and passes the output to all of the elements of the second layer. This procedure is repeated through all the layers of the network up to the last. The final output layer produces the network response to the input pattern. Overall, recall is evaluating the response of a neural network to an input pattern.

2.4 Activation Functions

Activation functions, sometimes known as transfer functions map the output of a processing element to a specific range. These functions can either be linear or nonlinear.

Any function can be used as an activation function for Backpropagation learning as long as it has the following properties: continuous, monotonically increasing, and differentiable. Some of the common activation functions are: linear, sigmoidal, and step. The step and bounded linear activation functions are most commonly used in Hebbian learning algorithms where differentiability of the activation function is not required [Haykin, 1994]. The linear transfer function, shown in Figure 2.4, has the form

$$F(x) = x \tag{2.1}$$

When the linear transfer function is bounded in the range $\{\pm \epsilon\}$, the function becomes

the nonlinear (not differentiable) ramp threshold function, shown in Figure 2.5, and represented by the equation:

$$F(x) = \begin{cases} +\varepsilon & \text{if } x \geq \varepsilon \\ x & \text{if } |x| < \varepsilon \\ -\varepsilon & \text{if } x \leq -\varepsilon \end{cases} \quad (2.2)$$

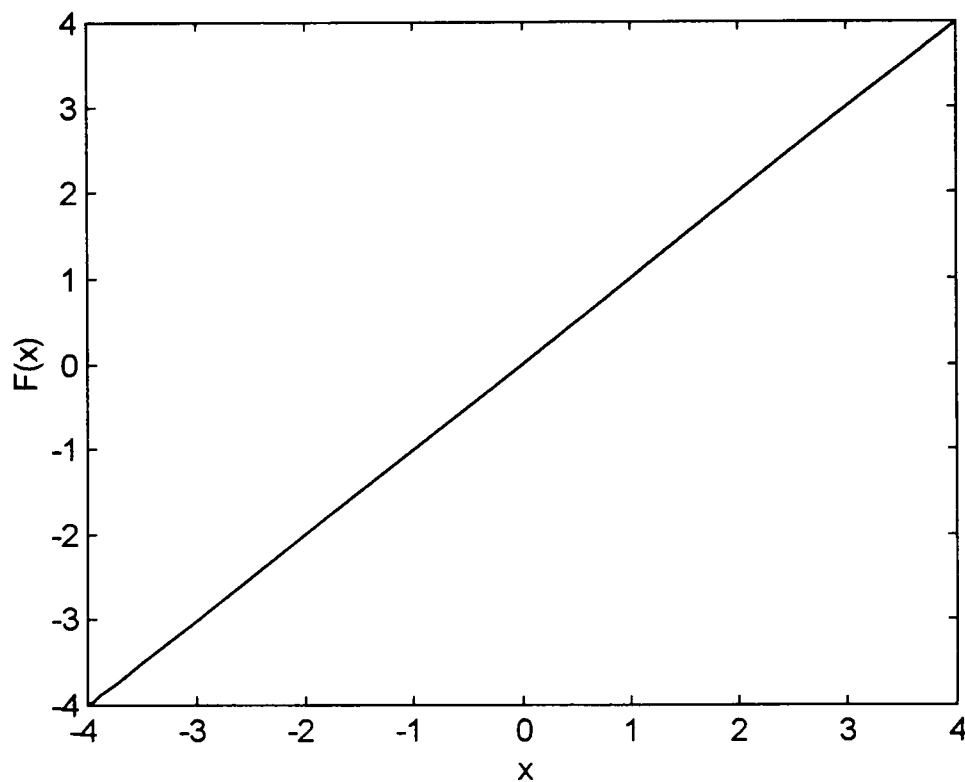


Figure 2.4 Linear transfer function.

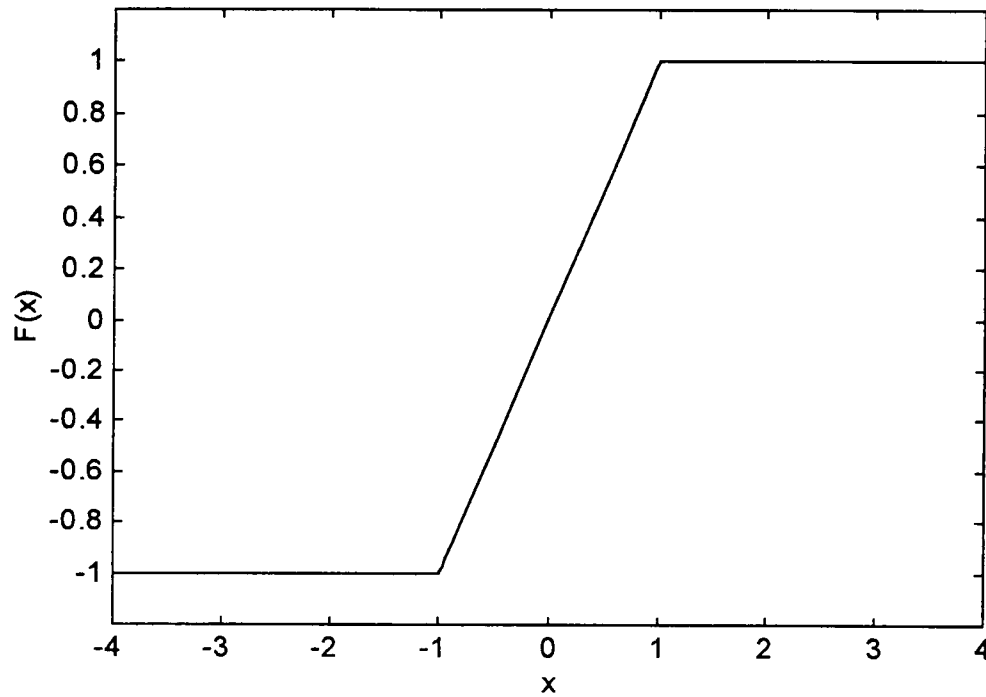


Figure 2.5 Ramp activation function.

The step activation function, shown in Figure 2.6, is represented by:

$$F(x) = \begin{cases} +\epsilon & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

The most commonly used activation function is the sigmoidal function, shown in Figure 2.7. It has the form:

$$F(x) = \frac{1}{1 + e^{-\beta x}} \quad \beta > 0 \quad (2.4)$$

where β is the shaping parameter [Uhrig, 1993].

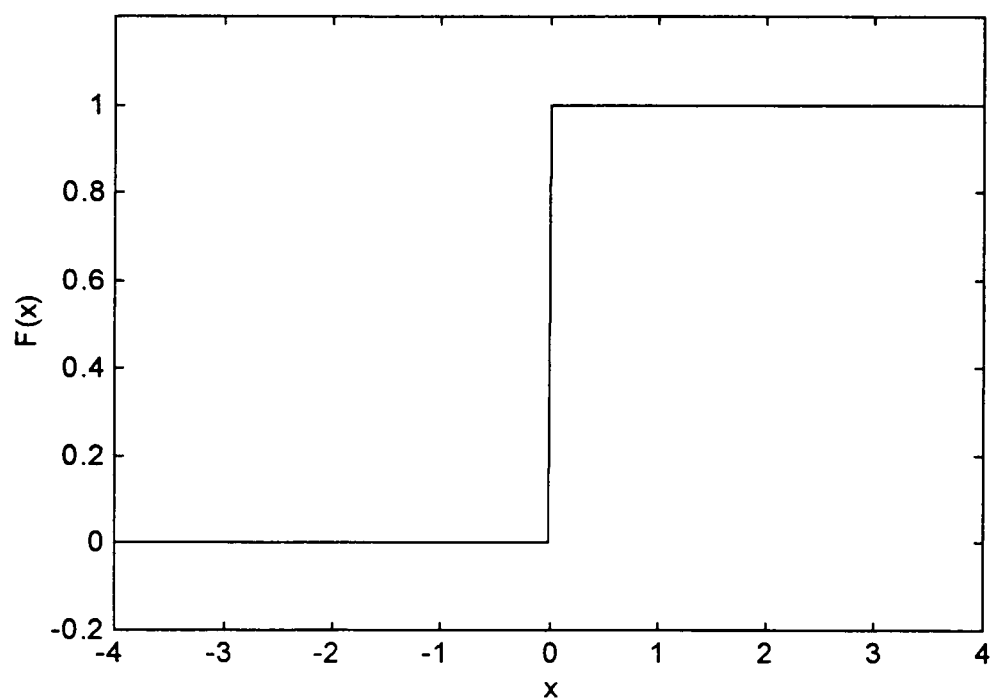


Figure 2.6 Step activation function.

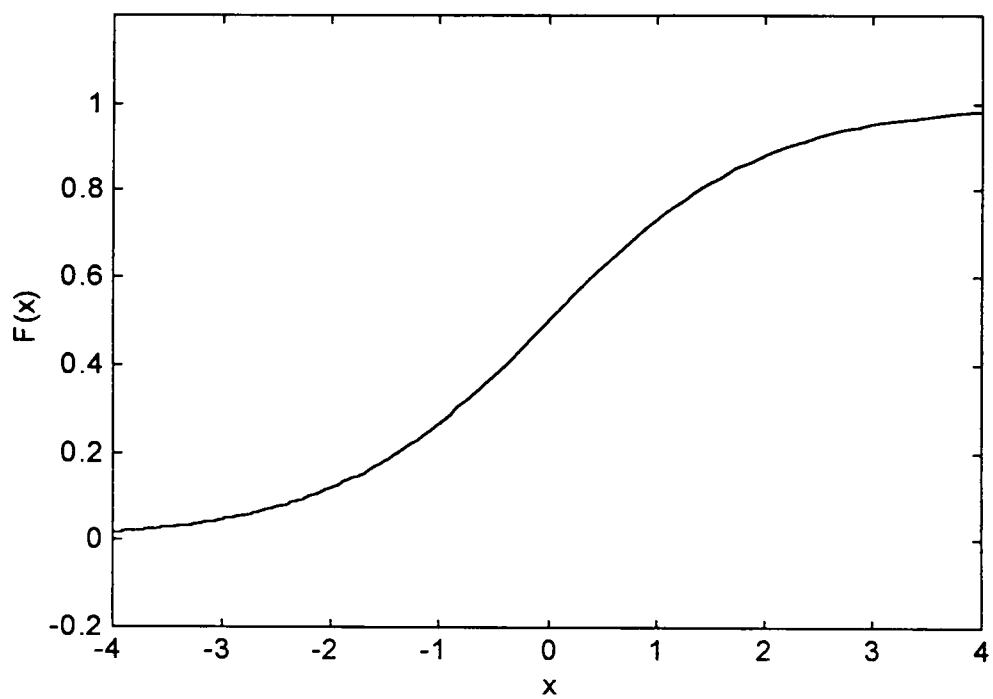


Figure 2.7 Sigmoidal activation function.

An alternative to the sigmoidal activation function is the hyperbolic tangent, shown in Figure 2.8. It has the form

$$F(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$

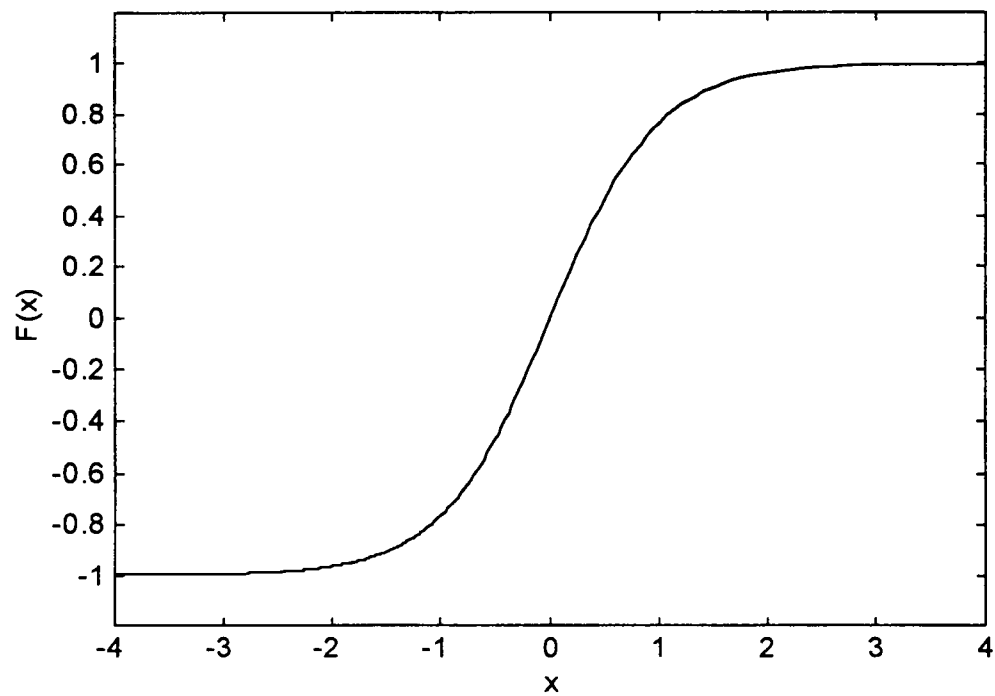


Figure 2.8 Hyperbolic tangent activation function.

Chapter 3

Training Algorithms

3.1 Introduction

Several weight optimization techniques will be discussed along with three popular training algorithms; Backpropagation (BP), Scaled Conjugate Gradients (SCG), and Levenberg Marquardt (LM).

Training is the process of changing the weights in the network to map the desired input/output relationship to a desired degree of accuracy. Many error criteria, or cost functions, between the desired network output and the actual network output are used to determine how accurate the neural network model maps the desired relationship. Several of the most popular are Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Sum Squared Error (SSE). Mean squared error is computed by averaging the squared errors of the training presentations within that iteration. Root mean square is arrived at by simply taking the square root of the mean square error. Sum squared error is found by summing the square of all the errors of the training presentations within that iteration. In particular, if we are processing p training patterns, let the desired (target) activation of output neuron j be designated as t_j , and the observed activation be o_j . If there are n output neurons, the sum squared error over all presentation is given by:

$$SSE = \sum_p \sum_j^{n-1} (t_j - o_j)^2 \quad (3.1)$$

The choice of a particular cost function depends on the particular application. Masters [1993] has a detailed discussion of cost functions and their applications.

For this study, the sum squared error was used for all network training except for cross validation (see section 5.4.3.2). For cross validation training, the mean squared error was used as an error goal criteria because in most cases the training and testing sets contain a different number of patterns.

The training process starts by initializing all weights to random, small non zero values. This creates a very nonlinear error surface that the training algorithm must negotiate to reach a global error. Figure 3.1 below shows a typical error and contour surface for a single neuron. We can see that the error surface contains a global minimum, and several local minimum, all for a single neuron. Essentially, the training algorithms function is to traverse the error surface in such a manner as to arrive at the global error minimum.

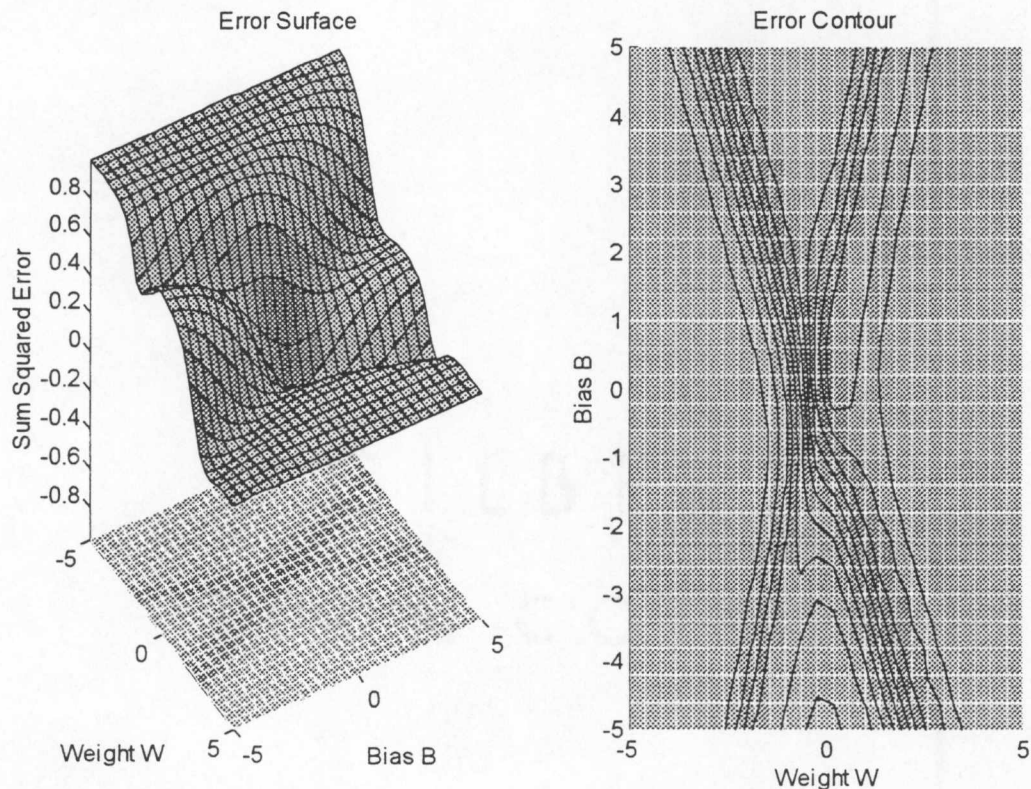


Figure 3.1 Example of a nonlinear error surface for a single nonlinear neuron.

Next, a select set of training samples (input/output exemplars) are presented to the network. A measure of the error incurred by the network is made, and the weights are updated in such a manner as to reduce the error. This process is continued until a set error goal is reached.

One training cycle (present the training set...calculate error...update weights) is called an *epoch*. The subset size of the training data (number of training exemplars used per weight update) is called the *epoch size*. Often, the entire training set is used for the epoch, this is known as a *batch training mode*. If the weights are updated after each presentation of a training pattern, this is referred to as a *pattern training mode*.

In the pattern mode of training, weight updating is performed after the presentation of each training exemplar or a subset of training exemplars. The order of presentations of training exemplars are randomized from one epoch to the next. In this way, the learning becomes a stochastic process and reduces the possibilities of getting trapped in a local minimum. In the batch mode of training, weight updating is performed after the presentation of all training exemplars that constitute the training set in an epoch.

Comparing the two modes of training, we can see that the average weight change made in the pattern mode of training is different from the corresponding weight change made in the batch mode of training. The pattern mode weight change is a local estimate of the weight change for the batch mode. Conversely, the use of batch mode of training provides a more accurate estimate of the gradient vector because it is based on the entire training set. From a memory storage point of view, the pattern mode of training is preferred because it requires less memory storage for each synaptic connection. In addition, a pattern training mode would be preferable to a batch training mode for an adaptive network that is required to continually learn new patterns. Overall, the relative effectiveness of the two training modes depends on the particular application [Haykin, 1994].

3.2 Optimization Techniques

The training process is a basic mathematical optimization problem. The problem is to minimize a scalar quantity E which is a function of several variables. In the neural network case, these variables are the network weights and biases and the quantity E is the sum of squared errors. This problem can be written as:

$$E_{\min} = f(x_1, x_2, x_3, \dots, x_n) = f(\mathbf{x}). \quad (3.2)$$

Gradient methods of solving this equation are based on the Taylor series expansion:

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{g}^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} + \dots \quad (3.3)$$

where $\mathbf{x}$ is the present vector of weights and biases and $\Delta\mathbf{x}$ is the change to the errors and biases made to reduce the error E . The following terms are now defined. The gradient vector ∇f is a row vector of first order partial derivatives of the function f .

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \dots \frac{\partial f}{\partial x_n} \right] = E'(\mathbf{x}) \quad (3.4)$$

The Jacobian gradient vector $\mathbf{g}$ is defined as the transpose of the gradient vector.

$$\mathbf{g}^T = \nabla f \quad (3.5)$$

The Hessian matrix $\mathbf{H}$ is defined as the $n \times n$ matrix of second order partial derivatives.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \dots & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} = E''(\mathbf{x}) \quad (3.16)$$

Since the neural network activation functions are continuous and differentiable, the first and second order partial derivatives exist. Therefore, at a local or global minimum, the Jacobian gradient vector is zero and the Hessian matrix is positive definite.

3.2.1 Gradient Decent

The fundamental first order method of optimization gradient decent, or also known as steepest descent. It uses the Jacobian gradient to determine the direction of movement to arrive at the minimum. The first order approximation of the change of the error (ΔE) is:

$$\Delta E = \mathbf{g}^T \Delta \mathbf{x} . \quad (3.7)$$

The gradient of a multivariate function is the direction that is most steeply "uphill". If we take a small step in the opposite direction (negative gradient), we will be going in the optimal direction to decrease the error function. Because the gradient is always perpendicular to the equal error contour lines, the path taken by the error will zigzag, bouncing from one wall to the next as it progresses down the error surface walls. This can make for a very slow learning process.

3.2.2 Gradient Decent With Line Search

This uses the same principle as gradient decent except for the use of a line search technique to determine an optimal direction. The maximum reduction of E in the direction of $-\mathbf{g}$ is found by a line search to find $\Delta\mathbf{x}=\lambda\mathbf{u}$ where $\mathbf{u}$ is a unit vector in the direction of $-\mathbf{g}$. The value of λ can be found by raising it until the error reaches a minimum in that direction. Since there will be several evaluations of E before the minimum in the direction $\mathbf{u}$ is reached, this line search can be computationally intensive. Also, this method must be repeated since the global minimum is usually not in the direction of the initial negative gradient. This method of optimization follows a zigzag path towards the minimum and learning is very slow near the minimum. Figure 3.2 is a graphical depiction of the steepest descent method with line search.

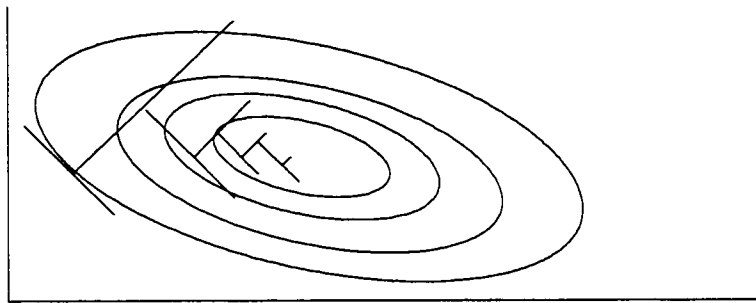


Figure 3.2 Steepest decent with line search

As shown, the subsequent line search directions are always perpendicular to each other and the length of movement gets smaller as the minimum is approached. This slow convergence near the minimum is a major drawback of the steepest decent method. One advantage of the line search over the traditional gradient descent is that the gradient only needs to be evaluated after each line search, versus each movement.

3.2.3 Conjugate Gradient Method

An improvement to the choice of search directions and step size is found using conjugate gradients. The conjugate gradient method of optimization was first developed by Fletcher and Reeves [1964]. If $\mathbf{x}$ is near the minimum, the second order Taylor approximation is:

$$f(\mathbf{x}_{\min}) \approx f(\mathbf{x}) + \mathbf{g}^T \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x} . \quad (3.8)$$

Taking the partial derivative of this quadratic representation with respect to $\Delta \mathbf{x}$, we get:

$$0 = \mathbf{g} + \mathbf{H} \Delta \mathbf{x} . \quad (3.9)$$

The critical points are the points $\mathbf{x}=[x_1 \ x_2 \ \dots \ x_n]$ which solve this linear equation. The conjugate gradient method uses conjugate directions as its search directions and the movement from the starting point to the minimum point can be expressed as:

$$\mathbf{x} - \mathbf{x}_{\min} = \sum_{i=1}^n \alpha_i \mathbf{p}_i , \quad (3.10)$$

or also written as:

$$\mathbf{x}_{\min} = \mathbf{x} + \alpha_1 \mathbf{p}_1 + \alpha_2 \mathbf{p}_2 + \dots + \alpha_n \mathbf{p}_n , \quad (3.11)$$

where $\mathbf{x}$ is the initial location on the error surface and $\mathbf{p}_i$ ($i=1,2,\dots,n$) are conjugate directions that form a basis for the n -dimensional error space.

$$\mathbf{p}_i^T \mathbf{H} \mathbf{p}_j = 0 \text{ (for } i \neq j \text{ and } i=1,2,3,\dots,n) \quad (3.12)$$

The conjugate gradient algorithm chooses the negative gradient as the first search direction p_1 and chooses the other conjugate directions as the successive search directions. The step sizes α_i are chosen to minimize the error function on the plane π_i formed by the conjugate directions $1, 2, \dots, i$. This algorithm assures a global minimum is found in a maximum of n steps for an n -dimensional quadratic function.

The successive points in the error space are defined by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_{k+1}, \quad (3.13)$$

while the new directions, $\mathbf{p}_{k+1}$, are defined by:

$$\mathbf{p}_{k+1} = \mathbf{g}_{k+1} + \beta_k \mathbf{p}_k, \quad (3.14)$$

where

$$\beta_k = \frac{(\|\mathbf{g}_{k+1}\|^2 - \mathbf{g}_{k+1}^T \mathbf{g}_k)}{\mathbf{p}_k^T \mathbf{g}_k}. \quad (3.15)$$

The step size, α_i , is defined using second order information.

$$\alpha_i = \frac{\mu_k}{\delta_k} = \frac{-\mathbf{p}_k^T \mathbf{g}_k}{\mathbf{p}_k^T \mathbf{H}_k \mathbf{p}_k} \quad (3.16)$$

The Hessian is computationally intensive to find and is sometimes estimated. Also, since the error surface is approximated by a quadratic function near the minimum, there is no guarantee that the Hessian is positive definite. Therefore, many conjugate gradient algorithms use a line search to find the optimum value of α .

Since the error surface is approximated by a quadratic function, the algorithm may not reach the desired error goal in n iterations. Therefore, if the error goal is not reached in n iterations, the process is restarted with the search direction set to the negative gradient. Figure 3.3 depicts an example using conjugate gradients to find the minimum of a quadratic error function.

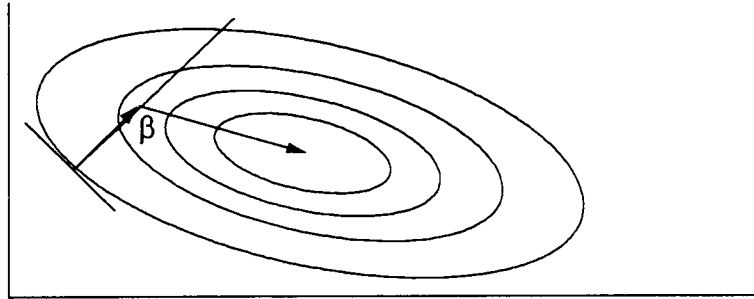


Figure 3.3 Conjugate gradient method

3.2.4 Newtons Method

To determine the movement $\Delta \mathbf{x}$, towards $\mathbf{x}_{\min}$ from a nearby point $\mathbf{x}$; we take the partial derivative with respect to $\Delta \mathbf{x}$ of the second order Taylor approximation (equation 3.8), and solve for $\Delta \mathbf{x}$. This results in

$$\Delta \mathbf{x} = -\mathbf{H}^{-1} \mathbf{g}. \quad (3.17)$$

This is called the Newton-Raphson method, or more simply, Newton 's method. Newton's method has several disadvantages: it is difficult to find the Hessian matrix, it is computationally intensive since the $n \times n$ Hessian matrix must be inverted, and it is unstable unless $\mathbf{x}$ is close to $\mathbf{x}_{\min}$.

3.2.5 Gauss-Newton Method

Several methods use an approximation of the Hessian matrix to construct second order information about the error surface. Since the error is a sum of squares of the errors of the outputs for each training vector, the total number of terms m in the error function is equal to the number of network outputs times the number of training vectors.

$$E = \sum_{k=1}^m [f_k(x)]^2 \quad (3.18)$$

and if we define

$$\mathbf{f}^T = [f_1(x) \ f_2(x) \ \dots \ f_m(x)] \quad (3.19)$$

then

$$E = \mathbf{f}^T \mathbf{f} \quad (3.20)$$

Let us now define a new matrix $\mathbf{A}$ which consists of the variation of each term m of the error function due to the variation of each parameter n (bias or weight).

$$\mathbf{A} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad (3.21)$$

By taking the partial derivatives of the error equation with respect to each parameter we get the Jacobian vector in terms of $\mathbf{A}$ and $\mathbf{f}$:

$$\mathbf{g} = 2\mathbf{A}^T \mathbf{f} \quad (3.22)$$

A second differentiation with respect to each parameter gives:

$$\mathbf{H} = 2\mathbf{A}^T \mathbf{A} + \text{second order term} \quad (3.23)$$

Neglecting this second order term results in an approximation to the Hessian matrix. The Jacobian and approximate Hessian can now be substituted into Newton's method to give us:

$$\Delta \mathbf{x} = -[\mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{f} \quad (3.23)$$

This is called the Gauss-Newton method. The Hessian approximation is positive definite near the minimum and the increment size should be limited so that the Hessian approximation, neglecting the higher order term, is valid. This method is good near the minimum but poor far from it.

3.3 Backpropagation

The basic backpropagation training algorithm is the generalized delta rule (GDR) popularized by Rumelhart and McClelland [1986] in which the sum of squares errors of outputs are minimized by changing the weights using first order gradient decent.

The training phase consist of two sweeps through the network. The first, or forward sweep starts by introducing a training pattern into the networks input layer. Using the recall procedure discussed previously, the networks response to the input pattern is evaluated. The network estimate (output) is then compared with the desired

output (target) for the given input pattern and an error is computed for each processing element in the output layer.

The second pass, or backward pass consist of passing back the computed error for use in adjusting the incoming weights to each processing element using a gradient decent technique. The errors in each output processing element are also used for assigning error to the processing elements in the layer below the output layer, and are therefore used for adjusting the weights to the layer below the output. This backpropagation of errors continues until the first layer is reached. The iterative process continues until the error between the actual and desired outputs falls below a pre-set level.

The algorithm proceeds as follows. To optimize the network weights, we compute the derivative of the error function with respect to each weight in the network and then change the weight as follows:

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}} \quad (3.24)$$

where $\Delta_p w_{ji}$ is the change of weight from the i th neuron to the j th neuron. E_p is the sum of squared errors for the training pattern p and η is the learning rate. E_p is defined as:

$$E_p = \sum_{j=0}^{n-1} (t_{pj} - o_{pj})^2 \quad (3.25)$$

where t_{pj} is the desired output at the j th neuron of the output layer for the training pattern p , and o_{pj} is the actual output produced through the network for the pattern p . o_{pj} is represented as:

$$net_{pj} = \sum_i w_{ji} o_{pi} + \theta_{pj} \quad (3.26)$$

and

$$o_{pj} = f(\text{net}_{pj}), \quad (3.27)$$

where $f(.)$ is an activation function and w_{ji} is the weight from the i th neuron to the j th neuron.

To obtain a rule for adjusting the weights, the gradient of E_p with respect to w_{ji} is replaced as follows:

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pj} \quad (3.28)$$

We need to compute δ_{pj} for each layer. It has been defined in two ways.

For a neuron in the output layer

$$\delta_{pj} = (t_{pj} - o_{pj}) f'(\text{net}_{pj}), \quad (3.29)$$

and for a neuron in an arbitrary hidden layer

$$\delta_{pj} = f'(\text{net}_{pj}) \sum_k \delta_{pk} w_{kj}. \quad (3.30)$$

3.3.1 Modified Backpropagation

Standard backpropagation suffers from slow convergence because it requires very small learning rates (η) to remain stable. If the learning rate is too small, convergence will be very slow due to the small error steps down the error surface. If the learning rate is too large, the steps down the error surface may be very large and oscillatory in nature, thus leading to instability. Another problem is that of local minimum. Since the only direction the error can travel is down the surface (negative gradient), standard backprop is prone to getting stuck in local minima. Once it walks down into a small valley (local minimum), it has no way of going up the surface to eventually fall into a lower canyon,

or global minimum. Two methods have been devised to overcome these problems, they are an adaptive learning rate, and momentum.

3.3.1.1 Adaptive Learning Rate

Since the learning rate controls the step size of the gradient decent approximation, adapting it to various error surface conditions will allow for faster convergence while eliminating learning stability problems. An adaptive learning rate keeps the learning step size as large as possible while keeping learning stable. Essentially, the learning rate is made responsive to the complexity of the local error surface. It uses a type of heuristic that says if the error is increasing, then decrease the learning rate. Or conversely, if the error is decreasing, the increase the learning rate.

3.3.1.2 Momentum

Momentum is a method used to decrease backpropagations sensitivity to small details in the error surface. Rather than allowing the search direction to thrash about as the gradient changes, a momentum is imposed on it. Each new search direction is computed as a weighted sum of the current gradient and the previous search direction. Essentially, it acts like a low pass filter by filtering out the local fluctuations in the error surface.

3.4 Scaled Conjugate Gradients

The conjugate gradient method chosen is the scaled conjugate gradient (SCG) algorithm derived by Moller [1993]. The SCG algorithm approximates the error function with a quadratic function and approximates the Hessian matrix term using first order information [Hestenes 1980].

$$s_k = \mathbf{H}_k p_k \approx \frac{\mathbf{g}_{k+1} - \mathbf{g}_k}{\sigma} \quad 0 < \sigma \ll 1 \quad (3.31)$$

This method greatly reduces the computational complexity of finding second order information. The approximation is also checked for positive definiteness. The positive definiteness of the matrix is determined by whether $\delta_k > 0$.

$$\delta_k = p_k^T H p_k = p_k^T s_k \quad (3.32)$$

If it is not positive definite, convergence is not guaranteed so the approximation is modified by adding a term times a scalar λ to it.

$$\tilde{s}_k = \frac{g_{k+1} - g_k}{\sigma_k} + \lambda_k p_k \quad (3.33)$$

When the Hessian approximation is positive definite the step size is calculated from

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{p_k^T g_k}{p_k^T s_k + \lambda_k |p_k|^2} \quad (3.34)$$

This equation shows that as λ becomes larger, the step size becomes smaller. This agrees with the theory that the approximation is only good near the operating point. If the matrix is not positive definite, then a smaller step should be taken.

The approximation that the error surface is quadratic must be validated since the Hessian approximation is modified. The following equation is a measure of how well $E_{qw}(\alpha_k p_k)$, the quadratic approximation, approximates $E(x_k + \alpha_k p_k)$, the error function.

$$\Delta_k = \frac{E(x_k) - E(x_k + \sigma_k p_k)}{E(x_k) - E_{qw}(\sigma_k p_k)} = \frac{2\delta_k [E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2} \quad (3.35)$$

A perfect approximation would have $\Delta_k = 1$. The scalar λ is adjusted to improve the quadratic approximation when Δ_k is outside of a preset range ($0.25 \geq \Delta_k \geq 0.75$).

3.5 Levenberg Marquardt

The Levenberg-Marquardt method makes use of the reliable convergence of steepest descent when far from the minimum and the rapid convergence of Gauss-Newton when close to the minimum. This is accomplished by use of a variable metric λ .

The variable metric was proposed by Goldfeld, Quandt and Trotter [1966] for use in the Newton-Raphson method,

$$\Delta \mathbf{x} = -[\lambda \mathbf{I} + \mathbf{H}]^{-1} \mathbf{g} \quad (3.36)$$

where $\mathbf{I}$ is an $n \times n$ identity matrix. The metric is varied as the minimum is approached. When far from the minimum, λ is large and $\Delta \mathbf{x}$ approaches the negative gradient. When closer to the minimum, λ is reduced and $\Delta \mathbf{x}$ approaches the Newton-Raphson increment.

Levenberg [1944] applied the same methodology to the Gauss-Newton approximation.

$$\Delta \mathbf{x} = -[\lambda \mathbf{I} + \mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{f} \quad (3.37)$$

As $\lambda \rightarrow \infty$, $\Delta \mathbf{x}$ is a step towards $-\mathbf{A}^T \mathbf{f}$ which is the negative gradient direction. And as $\lambda \rightarrow 0$, $\Delta \mathbf{x}$ tends towards the standard Gauss-Newton increment. This method makes use of the reliable convergence of the steepest descent algorithm and the rapid convergence of the Gauss-Newton methodology. It also does not require second order information.

The method for changing the metric λ was proposed by Marquardt [1963]. It involves multiplying λ by 10 until the error is reduced and incrementing $\Delta \mathbf{x}$. Then the

method divides λ by 10 and checks to see if the new Δx increment will reduce the error. If it does not reduce the error, λ is multiplied by 10 until the error reduces. This method simply reverts towards the gradient descent algorithm when the Δx does not reduce the error, and moves towards Gauss-Newton when the error is reducing. For each change in λ the gradient is not rederived, it is only necessary to rederive the gradient after the error is reduced and the weights and biases are updated with Δx .

3.6 Summary and Training Paradigm Selection

The major factors found important for the selection of the training paradigm was the size of the training set and the size of the neural network. Three methods were investigated: gradient descent based backpropagation, conjugate gradients, and Levenberg-Marquardt (LM).

Standard backpropagation is the slowest to converge but is computationally simple and reliable. It works well for large networks with large training sets. Several enhancements to the standard backpropagation algorithm were used including momentum, an adaptive learning rate, and linear layer weight estimates through regression methods.

Moller's Scaled Conjugate Gradient algorithm [1993] was also used and does have faster convergence properties than BP but is significantly slower than LM for small to medium sized problems. Its advantage over BP is that there are no user chosen parameters. Its disadvantage is that it prematurely gets stuck in local minima more often than BP.

Levenberg-Marquardt is an optimization method that trades off between the reliable convergence of the gradient descent paradigm and the quick convergence of Newton's method. Implementation of this method requires the computation of a pseudo Hessian matrix. This matrix incorporates second order information which aids convergence but is based on assumptions of being near the minimum and the error surface having a quadratic function. The mathematical manipulations of the pseudo

Hessian matrix make the LM training paradigm memory intensive and slow for large training sets and large network architectures.

Chapter 4

Sequential Probability Ratio Test Algorithm (SPRT)

4.1 Introduction

The sensor failure detection module is a component of the Instrument Surveillance and Calibration Verification (ISCV) System. Figure 4.1 shows a simplified block diagram of the ISCV system. Data from plant sensors is input to the AANN. The AANN produces an estimated sensor value which is compared to the actual sensor value. The difference between the actual and estimated value is called the residual which is used by the sensor failure detection module to determine when the AANN estimate is significantly different from the actual sensor value. The sensor failure module results are then output through an operator interface (computer screen) as the final condition of the sensor. The Sequential Probability Ratio Test (SPRT) is used to perform the sensor failure detection function.

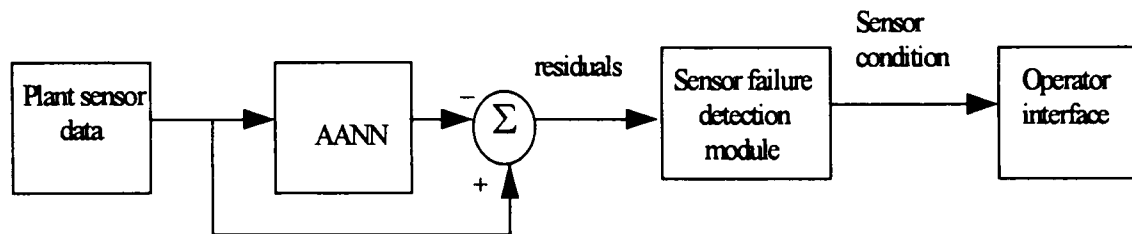


Figure 4.1 Simplified block diagram of the ISCV system.

The SPRT technique, which was originally developed by Wald [1945] and later used for sensor failure detection by Upadhyaya [1988], uses a statistical method to determine if a sensor has failed. Rather than computing a new mean and variance at every new sample, the SPRT continuously monitors the sensor's performance by processing the residuals. The residual signals, which are the differences between the sensor measurements and the estimates from the trained artificial neural networks, are used to generate a likelihood ratio. The likelihood ratio is evaluated by the decision threshold component to make a logical decision concerning the sensor status. The SPRT system is briefly illustrated in Figure 4.2.

The SPRT based method is optimal in the sense that a minimum number of samples is required to detect a fault existing in the signal. There are two types of decision rules for this test: Wald's two-sided and Chien's one-sided criteria. This technique can be used for the detection of sensor bias degradation and for the detection of sensor noise degradation with different concerns. The derivation of this algorithm will now be presented.

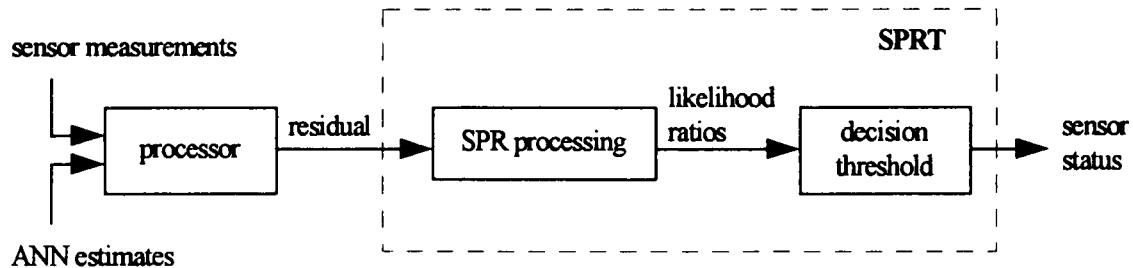


Figure 4.2 SPRT functional block diagram.

4.2 Decision Algorithm

Assume the residual signals are uncorrelated and have a Gaussian distribution, this assumption will be discussed later in this section. The Gaussian probability density

function distribution for measurement i and fault hypothesis j is given by the following equation:

$$p_i(s_i, m_i, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{(s_i - m_i)^2}{2\sigma_i^2}\right] \quad (4.1)$$

The normal hypothesis is that of zero mean (m_0) and a normal variance (σ_0^2).

$$H_0: m_0=0, \sigma_0^2 = \frac{1}{\sqrt{N-1}} \sum_{i=1}^N s_{i0}^2 \quad (4.2)$$

where s_i is the error residual signal between the measurement x_i and the estimation $\hat{x}_i$ (generated by ANN) of the sensor output, which is given as:

$$s_i = x_i - \hat{x}_i \quad (4.3)$$

For consideration of mean degradation, set $\sigma_1 = \sigma_0 = \sigma$ in the degradation hypothesis H_1 .

The likelihood ratio is a measure of the probability that a residual belongs to a faulted distribution compared to the probability that it belongs to an unfaulted distribution:

$$\lambda_m' = \frac{p_1(s, m_1, \sigma_1)}{p_0(s, m_0, \sigma_0)} = \frac{p_1(s_1|H_1)p_1(s_2|H_1)\dots p_1(s_m|H_1)}{p_0(s_1|H_0)p_0(s_2|H_0)\dots p_0(s_m|H_0)} \quad (4.4)$$

by taking the natural logarithm on both sides:

$$\lambda_m = \ln(\lambda_m') = \ln \left[\prod_{i=1}^m \frac{p_1(s_i|H_1)}{p_0(s_i|H_0)} \right] = \sum_{i=1}^m \ln \left[\frac{p_1(s_i|H_1)}{p_0(s_i|H_0)} \right] \quad (4.5)$$

so:

$$\lambda_m = \lambda_{m-1} + \ln \left[\frac{p_1(s_m|H_1)}{p_0(s_m|H_0)} \right] = \lambda_{m-1} + \frac{m_1}{\sigma^2} (s_m - \frac{m_1}{2}) \quad (4.6)$$

where: m_1 -- the degradation mean under the degradation hypothesis H_1 , which will be sensor specific and determined by the calibration specifications. In this test, the values of m_1 are ± 0.15 - 1.5% of the full scale indication of the normal signals; σ -- the normal variance under the normal hypothesis H_0 , which is obtained from the normal condition (no fault) test data.

The assumption that the signals are Gaussian has raised concern and some research has been performed to whiten the signals [Gross and Hoyer, 1995]. Even with these whitening techniques, there are spurious spikes outside the normal Gaussian distribution that cause spurious SPRT alarms. These alarms are filtered out using a logical filter (2 out of 4) after the SPRT.

Figure 4.3 presents a graphical display of the SPRT reasoning mechanism. In this example, the unfaulted probability density function of the residual is centered at a mean of 0 and has a standard deviation of 1. The faulted distribution is set to a mean of 4 with a standard deviation of 1. The likelihood ratio increases above a predefined boundary when the residuals are more likely to be from the faulted distribution than from the unfaulted distribution. This signifies that a fault is detected. The setting of these decision boundaries will now be discussed.

4.3 Decision Rules

Two decision rules are available in the literature. The two sided rule was used by Wald [1945] and the one sided rule was used by Chien [1976].

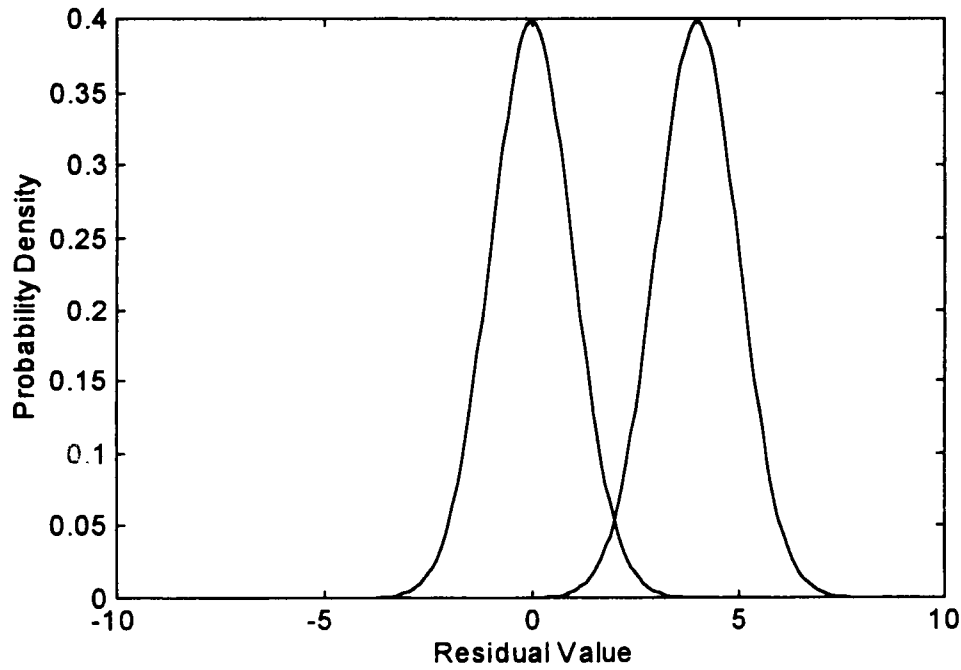


Figure 4.3 Graphical display of SPRT reasoning.

4.3.1 Wald's two-sided criterion

The decision boundary A and B are:

$$A = \ln\left(\frac{\beta}{1-\alpha}\right) \quad (4.7)$$

and

$$B = \ln\left(\frac{1-\beta}{\alpha}\right) \quad (4.8)$$

where: α = probability of false alarms; β = probability of missing alarms. Typical values are: $\alpha = 0.01\%$, $\beta = 10\%$. It is important to maintain a low false alarm probability in any failure detection systems. By setting the probabilities in this manner,

we reduce the number of false alarms so that user confidence is improved. The larger probability for missed alarms is not a problem because the monitoring is continuous and cumulative. If the alarm is missed at this sample, it will probably be detected at the next sample. Since we are concerned with slow sensor drifts (on the order of hours or days), quick detection times are not critical.

When: $\lambda_m < A$, sensor is OK;
 $\lambda_m > B$, sensor is degraded.

Where λ_m is the likelihood ratio defined in Section 4.1.

4.3.2 Chien's one-sided criterion

The decision boundary B_1 satisfies the following relation:

$$\exp(B_1) - B_1 - 1 = - \left[B + A \frac{\exp(B) - 1}{1 - \exp(A)} \right] \quad (4.9)$$

for $\exp(B_1) \gg B_1 - 1$, the above equation becomes:

$$\exp(B_1) \approx - \left[B + A \frac{\exp(B) - 1}{1 - \exp(A)} \right] \quad (4.10)$$

so B_1 can be solved as:

$$B_1 \approx \ln \left\{ - \left[B + A \frac{\exp(B) - 1}{1 - \exp(A)} \right] \right\} \quad (4.11)$$

When: $\lambda_m < 0$, sensor is OK;
 $\lambda_m > B_1$, sensor is degraded.

Where λ_m is the likelihood ratio defined in Section 4.1.

Wald's two-sided criterion can be used for either sensor bias degradation or noise degradation while Chien's one-sided criterion can only be used for sensor bias degradation, due to the lack of sufficient parameters for sensor noise degradation.

4.4 SIMULINK Implementation

The SPRT is implemented in SIMULINK to detect when the residuals are statistically significant. The SIMULINK implementation of the SPRT presented here was done by Hines [Uhrig, Hines, Black, Wrest, Xu, 1996].

There is a block of SPRTs dedicated to each AANN with each SPRT dedicated to a single sensor. The input to the SPRT is the residual. Figure 4.4 illustrates a block of four SPRTs that are dedicated to an AANN that estimates four sensor outputs. The Mux (multiplexer) block groups several scalar lines into a vector line while the Demux (demultiplexer) block simply converts a vector line into its scalar components.

Each SPRT block consists of a block that computes the likelihood ratio setpoint and a block that computes the likelihood ratio. These two numbers are compared and a fault hypothesis is made. Figure 4.5 illustrates an SPRT block that has a residual as an input and outputs a 1 for a faulty sensor and a 0 for a fault free sensor. Chien's one sided criterion is used: the summed lambda is compared to the setpoint and if lambda is greater than the setpoint, the sensor is said to be faulted and the output is a 1. When lambda is less than the setpoint, the comparison is negative and is limited to a zero.

The likelihood ratio setpoint is calculated in the Chien's block which is presented in Figure 4.6. This setpoint is calculated using the missed alarm probability (beta) and the false alarm probability (alpha).

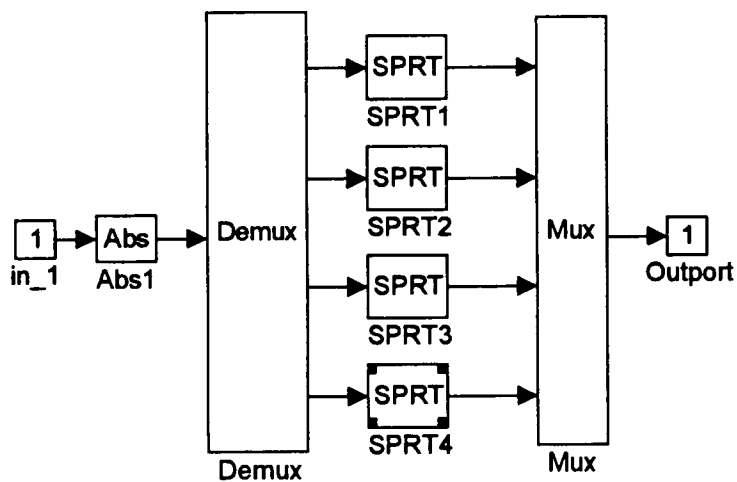


Figure 4.4 Block of four SPRTs

B1B111B1

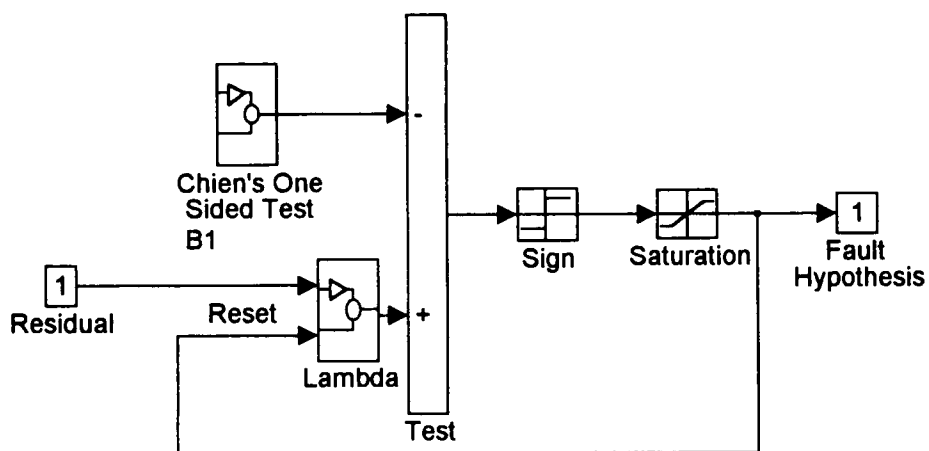


Figure 4.5 SPRT block

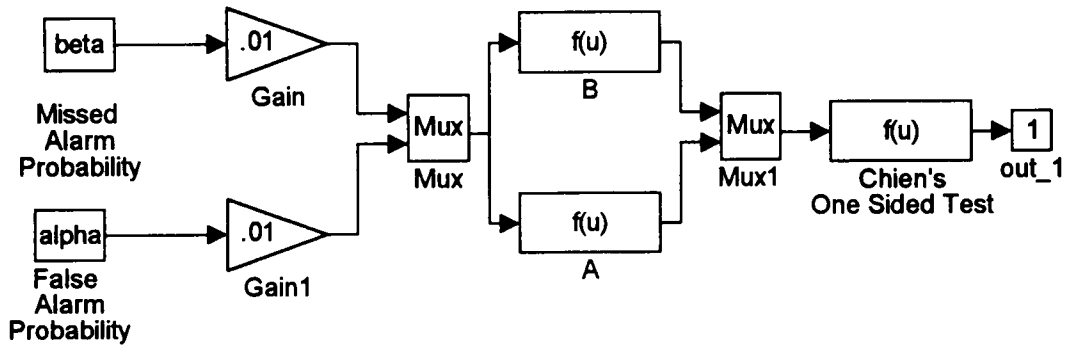


Figure 4.6 Chien's one sided test

$$B = \log((1-u(1))/u(2)) \quad (4.12)$$

$$A = \log(u(1)/(1-u(2))) \quad (4.13)$$

where: $u(1) = 0.01 \cdot \beta$ and $u(2) = 0.01 \cdot \alpha$.

$$B1 = \text{out_1} = \log\{-u(1) + u(2) \cdot (\exp(u(1)) - 1.0) / (1.0 - \exp(u(2)))\} \quad (4.14)$$

where: $u(1) = A$ and $u(2) = B$.

The block presented in Figure 4.7 implements the calculation of the likelihood ratio (λ). It adds the previous likelihood ratio to the newly calculated value. This sum is not allowed to be less than zero by the saturation block. When the sum is larger than the setpoint, a reset signal is sent and the likelihood ratio is reset to zero.

The block illustrated in Figure 4.8 calculates the new likelihood ratio (λ) to be added to the sum. It is calculated by the function:

$$\lambda = u(2) \cdot (u(1) - u(2)/2) / u(3) \quad (4.15)$$

where:

- $u(1)$ = the absolute value of the residual.
- $u(2)$ = $fmean$, the faulted mean.
- $u(3)$ = variance, the fault free residual's variance.

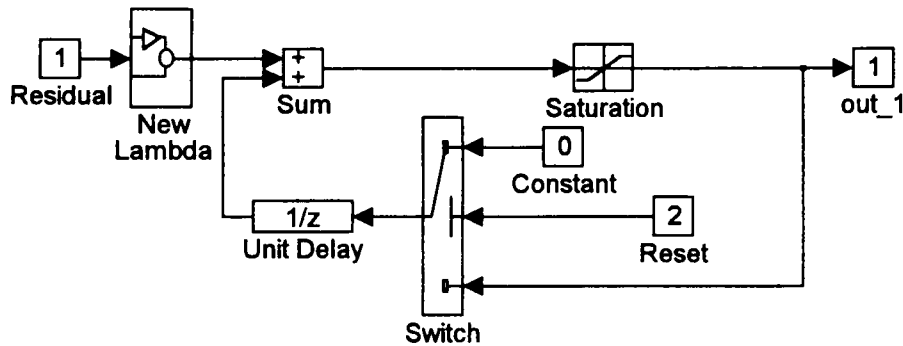


Figure 4.7 Lambda calculation

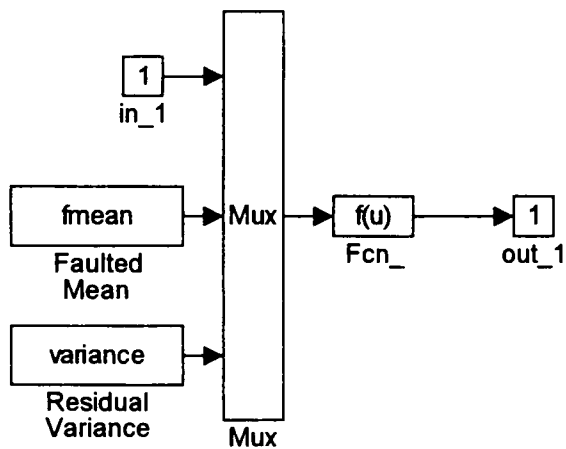


Figure 4.8 New Lambda

The faulted mean and the residual variance are determined and entered by the user. Alternatively, a MATLAB routine could calculate these parameters to optimally make the SPRT as sensitive as possible.

Chapter 5

ISCV System Development

5.1 Introduction

The approach to instrument surveillance and calibration verification (ISCV) through plant wide monitoring proposed in this section is based on the use of an autoassociative neural network (AANN).

An autoassociative neural network (sometimes called an autoassociative memory) is a neural network in which the outputs are trained to emulate the inputs over an appropriate dynamic range. The components of the desired output vector are the same as the components of the input vector. The relationships between the different variables are embedded in the weights by the training process. As a result, the output can be a correct version of an input pattern that has been distorted by noise, missing data, or nonlinearities. Plant variables that have some degree of coherence with each other constitute the inputs to the network. As with all neural networks, AANNs are inherently fault tolerant due to the large number of parallel connected paths and weights for each output. A change in one input to an AANN may not significantly change the corresponding output value unless the change has been trained into the network. Once the network has been trained with normal operational data it can be used to monitor the selected plant variables for sensor drift or failure by simply comparing the network inputs to the outputs. Due to the behavior of AANNs, monitoring the differences between the values of the individual outputs of the AANN and the corresponding measured sensor values from the nuclear plant (AANN inputs) provides a method of identifying sensor or instrumentation drift or failure. A simplified AANN signal error detection schematic is shown in Figure 5.1. The AANN method of monitoring many

variables not only indicates that there is a sensor failure, it clearly indicates the signal channel in which the signal error has occurred.

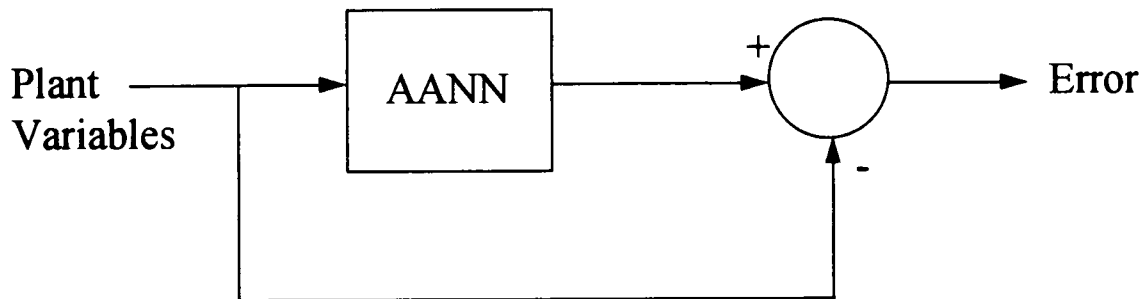


Figure 5.1 Simplified AANN signal error detection schematic.

5.2 Data Selection and Processing

Florida Power Corporation supplied data from the entire 1992-93 fuel cycle (fuel cycle 9) which consisted of over 2000 instrument channels acquired by the MODCOMP Plant Process Computer System (PPCS) and was stored every minute by the Replacement Emergency Dose Assessment System (REDAS) at Crystal River Unit 3, a two loop OTSG B&W Pressurized Water Reactor (PWR) system. The data was stored on an optical disk allowing easy access to the data parameters using FPCs Workstation Interface Software (WIS) data retrieval software.

5.2.1 Data Selection

For an AANN monitoring system to function correctly, it must be trained on fault free operational data. Only data from certain periods of the fuel cycle were used for this study. This includes data from reactor startup, and a five week period during full power operation. Since the NRC requires all instruments included in Technical Specifications to be calibrated during re-fueling, the assumption was made that the plant instruments and sensors should be operating within specifications during reactor startup. The five

week period of data designated as "error free" was specified as such by Rollin VanAlstine, Senior I&C engineer at Florida Power Corporation. During these five weeks, the plant operation was believed to be "fault free", in that no adjustments were made to sensors or instrument channels. The five non-consecutive weeks are listed below. The two periods during the fuel cycle (startup and the five weeks) were used for training, while testing was performed with data through out the fuel cycle.

Reactor startup:

7/17/92 - 8/7/92

5 weeks:

6/28/93, 7/12/93, 7/26/93, 8/09/93, 8/23/93

5.2.2 Data Processing

Virtually all networks train more efficiently if their inputs and outputs are restricted to a "reasonable" range [Bishop, 1995]. This entails pre-processing the input data before it is presented to the network, and post-processing the outputs of the network to give the required output values. There are two reasons why pre-processing the data is advantageous (or in some cases required).

1. Networks that use nonlinear activation functions (such as the sigmoidal "logistic" function used in this study) require summed input values limited to approximately ± 3 to prevent saturation of the activation function which greatly slows (or prohibits) training. Inputs beyond these limits are in a region of the nonlinear activation function where the slope is small (see figures 2.7 and 2.8). Due to the small gradient, training speed is greatly reduced. In addition, the sigmoidal "squashing" function tends to emphasize the importance of intermediate output values (values in the range of greatest slope), while masking fine differences when the outputs are near there extreme high and low values [Masters, 1993].

2. A typical plant system has parameters that differ in magnitude and/or deviation greatly (such as temperature contrasted with pressure). Depending on the units in which the parameters are expressed, they have values which may differ by several orders of magnitude. The network would need to find a solution for the weights in which some weight values had markedly different values from others. This would greatly effect the network's generalization ability.

From the above discussion, it is apparent that the input parameters should be scaled to a narrow range near the center of the network's limits. This is the approach used in this system. The input parameters to the network are linearly prescaled between the range of 0.1 and 0.9. The slope and intercept parameters are saved and used for post-scaling the parameters to the required units at the output of the AANN.

5.3 Parameter Selection

Initially 112 parameters were selected from more than 2000 plant parameters stored on the optical disk. These parameters were selected based upon engineering judgment for plant wide NSSS and BOP monitoring systems and used as a starting point for network selection. The 35 "Technical Specification" related sensors listed by Rollin VanAlstine that must be monitored closely were included. Below are the three methods used to group the parameters into optimal AANN's.

5.3.1 Engineering Judgment

Using engineering judgment may seem rather arbitrary but it is a very important consideration when grouping parameters. This may result from one's ability to recognize nonlinear relationships that mathematical techniques are unable to recognize. Using this technique, parameters were grouped according to their respective operating systems. For this two loop plant, three network groupings were created consisting of:

1. Primary (reactor) temperature, levels, pressures, flows (labeled Network B).
2. Secondary loop A temperatures, levels, pressures, flows (labeled Network AA).

3. Secondary loop B temperatures, levels, pressures, flows (labeled Network AB).

Primary variables are grouped into a single network B. Secondary variables are grouped into two networks, AA and AB; the first character (A) designates a secondary network, and the second character is an A for loop A variables, and a B for loop B variables.

The 112 parameters were each grouped into one of the three groups above. The following techniques were used to refine and optimize the three groupings listed above. After optimization, a total of 78 parameters remained as the parameters modeled by the AANN ISCV system.

5.3.2 Correlation Coefficients

Previous studies indicate that the degree of correlation between input variables is relatively significant for an AANN to function as a complex system monitoring device [Upadhyaya 1992, Kramer 1991]. Under these circumstances, changes in one input variable due to drift, channel deterioration, or failure will not significantly effect the corresponding value of the output because the output is related to all the other correlated input variables through a large number of paths and weights.

The parameters were selected based on plant safety points that require close monitoring (labeled Technical Specification), and the degree of coherence between the parameters. This was statistically determined using linear correlation coefficients given by:

$$S(i, j) = \frac{C(i, j)}{\sqrt{C(i, i)C(j, j)}} \quad (5.1)$$

where C is the covariance matrix.

The coefficients range from ± 1 with the highly correlated parameters at the boundaries of the range. A value near ± 1 means there is a direct relationship between the change of a parameter with respect to another. If two signals showed no change

with time (flat), their correlation would be near zero. It should be noted that this is a measure of linear correlation. Parameters were retained based on their requirement to be monitored (Technical Specification point) and its overall importance to the functionality of the system (a relatively high correlation coefficient value of approximately 0.5 or greater was selected to be reasonable for this study). Table 5.1 lists the average, lowest, and highest correlation value (absolute value) for each network. The correlation matrices for each or networks AA, AB, and B with the average coefficient value at the top are shown in appendix A in Tables A5.1, A5.2, and A5.3 respectively.

Table 5.1 Network correlation coefficient values.

<i>Network</i>	<i>Average</i>	<i>Lowest</i>	<i>Highest</i>
AA	0.65	0.00	0.99
AB	0.60	0.01	0.99
B	0.28	0.00	0.99

Tables 5.2, 5.3, and 5.4 list the final three network groupings for networks AA, AB, and B respectively. Listed is the network sensor number, the computer system tag ID, the signal description, whether the signal is a Technical Specification point or has a redundant sensor, the statistical variance and median value of the parameter during full power operation, and the instrument's upper and lower range values.

5.3.3 Sensitivity Analysis

Sensitivity is defined as the ratio of the change of the output to the corresponding change in the input. The sensitivity analysis of the network can be used as a validation technique to see if the proper selections were made using correlation coefficients. A particular network parameter may be overly sensitive to small input perturbations. This

sensitivity may be an indication that the particular parameter is not sufficiently redundant with other parameters included in the network. A sensitivity analysis of each network was performed (see section 5.7.3.3) to demonstrate the utility of this linear technique for network parameter selection.

5.4 Network Architecture

The basic feature of an AANN is that its output parameters are trained to equal its input parameters, thus the number of input nodes equals the number of output nodes. The number of hidden layers (1, 2, 3 ...etc.) and the number of nodes in each layer is left up to the researcher to determine to obtain the optimal performance for the particular application.

The general approach taken in this study is to find a network architecture that has the best generalization capabilities for a plant wide monitoring system. More precisely, generalization is the ability of the network to estimate the correct sensor value even when the corresponding input signal contains faults. Once the best network architecture has been determined, training techniques are incorporated to optimize the performance of the network.

5.4.1 Network Architecture Selection

Finding an optimal network involved examining three different architectures (1, 2, and 3 hidden layers), and nonlinear and linear output layers. The three networks are trained with MATLAB's [1995] standard backpropagation with an adaptive learning rate and momentum. Network performance was evaluated with a selected testing set using normal operational data, and a testing set with an artificial error placed in one of the parameters. The RMS error between the actual parameter value and the network's estimated parameter value is used for performance comparisons.

During this study, it was found that in order for the networks to function as a robust plant monitoring system, a minimum of two nonlinear layers is required. While a standard feedforward multilayer perceptron with one hidden layer has been shown to be

a universal approximator [Cybenko, 1989], it does not say how many nodes in the single hidden layer may be required. One hidden layer may be sufficient to provide such a mapping, but there is no evidence in the literature that one hidden layer is optimal for every modeling task. In some situations, multiple hidden layers may provide a superior mapping than a single hidden layer. In addition, the task here is to not only provide a mapping of the plant system, but it is essential that the mapping be robust, with excellent generalization abilities. This may be achieved with a network that contains multiple nonlinear hidden layers.

Listed below are three network architectures examined for this study. Logistic activation functions were used in each of the nonlinear layers. The input data as well as the output data was linearly scaled to range between 0.1 and 0.9. This prevents premature saturation of the nonlinear neurons and accelerates convergence.

1). Single hidden layer network I

- One logistic hidden layer, logistic output layer.
- 13 inputs - 20 hidden neurons - 13 outputs.

2). Two hidden layers network II.

- Two logistic hidden layers, linear output layer.
- Accelerated training due to regression technique to find linear output weights.
- (13-20-13-13) number of nodes in each layer.

3). Three hidden layer network III.

- Three logistic hidden layers, linear output layer.
- Same training advantage as network II.
- Added advantage of “bottleneck” feature detector in middle hidden layer.
- (13-17-11-17-13) number of nodes in each layer.

Network I is the simplest of the three. It consist of a single hidden layer network with nonlinear hidden and output layers.

Network II is identical to network I except for an additional nonlinear hidden layer and a linear output layer. This created a network with two nonlinear hidden layers and a linear output layer. Preliminary training sessions with network A were very long (13.5 hours). In order to reduce training time a linear regression technique to solve for the exact output layer weight values was proposed. This accelerated training because these near exact weight values would then be backpropagated back to the hidden layers thus accelerating convergence. A Singular Value Decomposition (SVD) technique (section 5.5.2) was used to solve for the linear output weight values [Masters, 1993].

Network III (see section 5.4.2 for a full discussion) uses 3 nonlinear hidden layers with a linear output layer. The three hidden layers are arranged in a feature detection architecture proposed by Kramer [1991, 1992] in which the first and third hidden layers function as mapping and de-mapping layers which contain more nodes than the input and output layers. The second hidden layer, known as the "bottleneck" feature detector layer, contains fewer nodes than the input and output layers. This layer prevents the network from learning a one to one mapping of inputs to outputs. The combination forces the network to develop a compact representation of the input data that better models the underlying system parameters. In addition Network III takes advantage of the linear SVD regression technique used to set the weights in the linear output layer.

5.4.1.1 Parameter Selection

As discussed earlier, previous studies indicate that the degree of correlation between input variables is relatively significant for an AANN to function as a complex system monitoring device. Under these circumstances, changes in one input variable due to drift, channel deterioration or failure will not significantly change the corresponding value of the output, because the output is related to all the other correlated input variables through a large number of paths and weights.

Parameters were selected based on plant safety parameters that require close monitoring (designated by the NRC in this FPC's Technical Specifications), and their

correlations using statistical correlation coefficients. Below is a list of 13 plant signals selected for this study. The calculated correlation coefficient matrix is shown in Table 5.5.

Selected Computer Point Signals:

1. S290 STEAM GEN A INLET FEEDWATER TEMP (F)
2. S301 STEAM GEN A INLET FEEDWATER FLOW (KLB/HR)
3. S255 FWP A DISCHARGE PRESSURE (PSIG)
4. R200 PRESSURIZER LEVEL L1 UNCOMP (INCH) TECH SPEC
5. A321 A OTSG EFIC LOW LEVEL (INCH) TECH SPEC
6. A847 STEAM GEN A OUTLET FLOW (KLB/HR)
7. A323 A MAIN STEAM PRESSURE (PSIG) TECH SPEC
8. R212 RC OUTLET A TEMP (T-HOT) NARROW (F) TECH SPEC
9. R234 RC LOOP A FLOW-CHANNEL A (MLB/HR) TECH SPEC
10. R317 PUMP INLET TEMP 3A1/RCP-1A T-COLD (F)
11. S287 STEAM GEN A LEVEL -STARTUP (INCH)
12. R327 REACTOR HEAD TEMP LOOP 164A (F) TECH SPEC
13. P208 LINEAR POWER CHANNEL NI-5 (PERCENT)

As the Table 5.5 shows, the parameters are closely correlated to each other with the lowest value of 0.08 between the pressurizer level and the feed water pump discharge pressure. The average correlation coefficient value is 0.79.

5.4.1.2 Network Training for Architecture Selection

Data from the 13 parameters listed above was retrieved at 15 minute intervals for approximately 20 days of plant operation (July 17, 1992 to August 7, 1992). The plant data included reactor startup to about eleven days of steady state plant operation. This data set included several transients to test the networks' abilities to follow rapid changes in plant conditions.

Table 5.5 Correlation coefficients matrix for initial 13 parameter network.

Correlation Coefficients			Average = 0.79											
	S290	S301	S255	R200	A321	A847	A323	R212	R234	R317	S287	R327	P208	
S290	1.00	0.91	0.43	0.69	0.94	0.91	0.75	0.95	0.81	-0.51	0.90	0.94	0.93	
S301	0.91	1.00	0.58	0.55	0.98	1.00	0.89	0.92	0.95	-0.74	1.00	0.92	1.00	
S255	0.43	0.58	1.00	0.08	0.57	0.59	0.54	0.44	0.61	-0.62	0.60	0.44	0.57	
R200	0.69	0.55	0.08	1.00	0.57	0.55	0.43	0.80	0.34	0.10	0.52	0.80	0.58	
A321	0.94	0.98	0.57	0.57	1.00	0.98	0.84	0.92	0.93	-0.71	0.98	0.92	0.98	
A847	0.91	1.00	0.59	0.55	0.98	1.00	0.89	0.92	0.95	-0.74	1.00	0.92	1.00	
A323	0.75	0.89	0.54	0.43	0.84	0.89	1.00	0.79	0.86	-0.71	0.89	0.79	0.88	
R212	0.95	0.92	0.44	0.80	0.92	0.92	0.79	1.00	0.79	-0.43	0.90	1.00	0.94	
R234	0.81	0.95	0.61	0.34	0.93	0.95	0.86	0.79	1.00	-0.86	0.96	0.79	0.94	
R317	-0.51	-0.74	-0.62	0.10	-0.71	-0.74	-0.71	-0.43	-0.86	1.00	-0.77	-0.43	-0.72	
S287	0.90	1.00	0.60	0.52	0.98	1.00	0.89	0.90	0.96	-0.77	1.00	0.90	0.99	
R327	0.94	0.92	0.44	0.80	0.92	0.92	0.79	1.00	0.79	-0.43	0.90	1.00	0.93	
P208	0.93	1.00	0.57	0.58	0.98	1.00	0.88	0.94	0.94	-0.72	0.99	0.93	1.00	

The training data set consisted of 435 out of 1,920 possible patterns selected from the twenty day operational period listed above. These patterns were carefully selected to cover the entire operational input space of the plant, including reactor startup and several transients.

5.4.1.3 Network Testing for Architecture Selection

The three networks were compared based on training time and general network robustness. Reduced training time is self explanatory; a network that trains faster is obviously an advantage. Robustness to input faults is required for the network to function as a complex monitoring device. When a faulted input is shown to the network, a robust network will estimate the unfaulted output that is desired. Table 5.6. below shows the training time results for the three networks.

Table 5.6 Network training records.

	<i>Flops</i>	<i>Epochs</i>	<i>Time^a</i>
Network I	4.6827e+9	3613	13.5 hrs.
Network II	7.8342e+8	201	20 min
Network III	8.8359e+8	150	18 min

^aPentium 90

We clearly see the great training advantage in networks II and III with the linear output layers. Even though network I is smaller, there is a training time advantage of approximately 40 to 1 using the SVD regression technique in the linear output neurons. As stated earlier, this is due to the optimal mathematical regression solution performed by the SVD method. These optimal weight error values are backpropagated through the network, accelerating convergence. Using these optimal errors to train the network greatly speeds up the training. This could prove very beneficial for an on-line real time situation in which a network may have to be trained quickly.

5.4.1.4 Network Testing Performance

Testing data consisted of the entire twenty day operational period. From the original data, two testing data sets were created with an artificial fault in the steam generator level parameter (computer point ID 321). This parameter was chosen because of the difficulty in determining the parameter from other plant parameters, thus it would be a good candidate to test the robustness of the network. The first faulted test set has a step error decrease of 5% of the steady state value at interval 1100 which then decreases further to 10% of the steady state value at interval 1500. Figure 5.2 shows a plot of the signal with the step error introduced. The second faulted test set has a ramp error that decreases the steam generator level at the rate of 3% per day. This signal is shown in Figure 5.3.

Normal plant operational data, and the two error data sets (step and ramp) are used to test the robustness of the networks. The root mean square (RMS) error between the networks estimate of the faulted signal, and the normal unfaulted signal was calculated for the total network error, and the error for an artificially created fault in signal #5 (steam generator low level) alone. Thus it would be expected that the best performing network should have error values closer to zero because the ideal network estimate of the faulted input signal should be identical to the unfaulted signal. Table 5.7 shows the results of these calculations.

Table 5.7 Total network RMS error and signal #5 RMS error.

	<i>Network I</i>	<i>Network II</i>	<i>Network III</i>
Total Net Error	0.0641	0.0532	0.0634
with fault in signal #5	0.0674	0.0574	0.069
Percent Difference	5.15	7.89	8.83
Signal #5 Error	0.0235	0.021	0.0287
with fault in signal #5	0.0246	0.0214	0.0291
Percent Difference	4.68	1.9	1.39

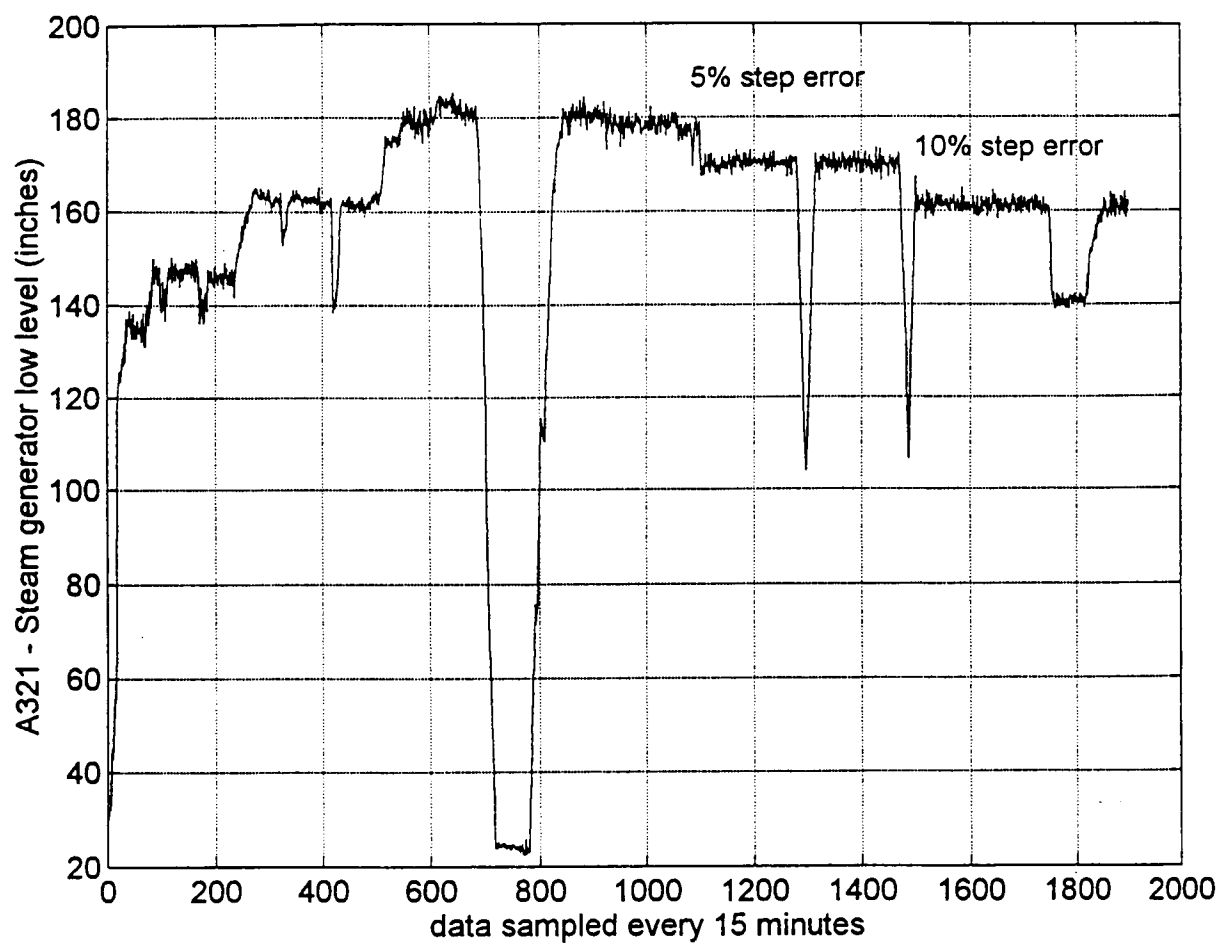


Figure 5.3 Artificially created step error fault in signal A321.

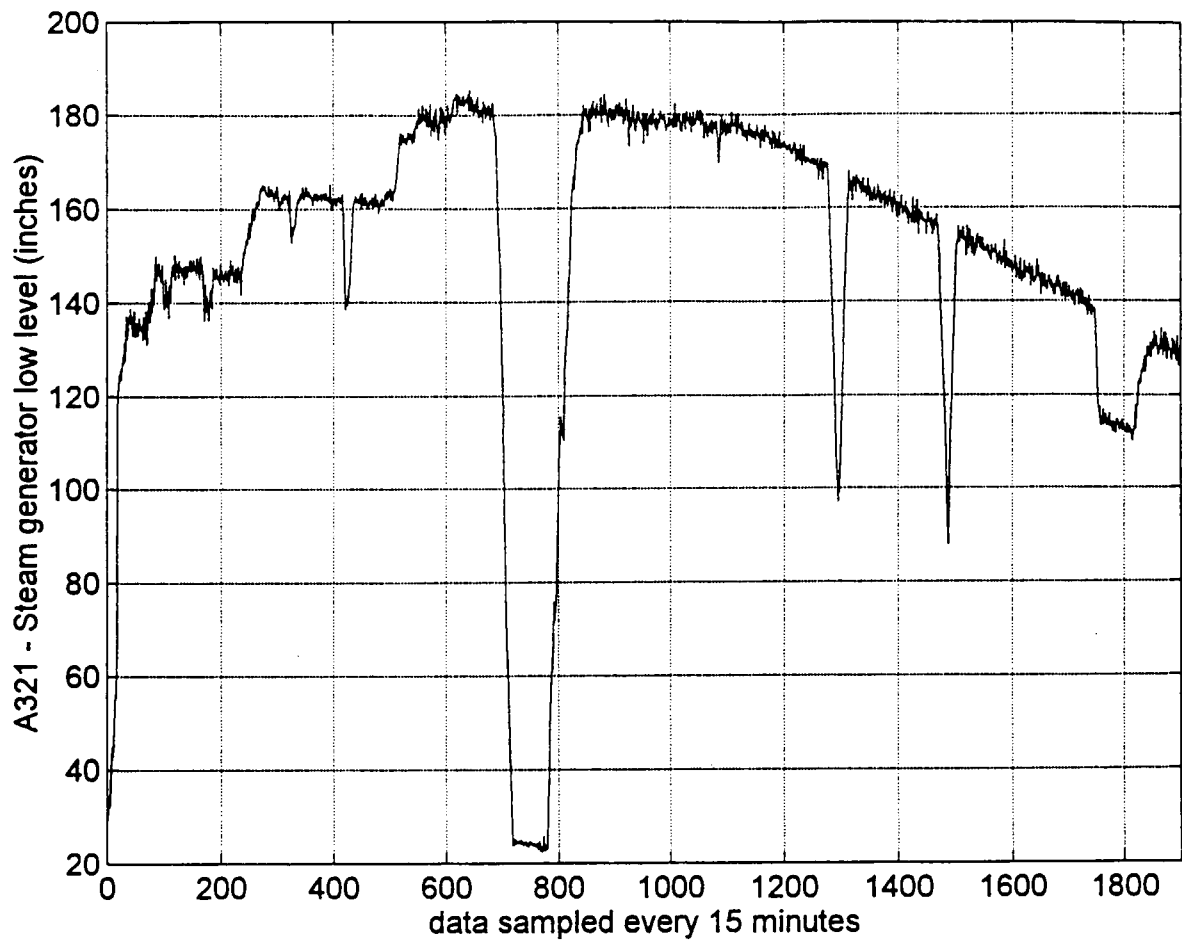


Figure 5.3 Artificially created ramp error fault in signal A321.

Note that the percent difference in total network error with a fault in signal #5 increases slightly from network I to network III (5.15 to 8.83 percent). But of more importance is the channel in which we are monitoring the fault (channel #5). The network estimate for network III is the lowest of the three networks (1.39 percent). Overall, network III was more robust to the fault than the other two network architectures. Figures 5.4, 5.5, and 5.6, show plots of the network performance with the faulted step error in signal A321 for networks I, II, and III respectively. The top plot shows the faulted signal and the network's estimate, while the bottom plot shows the residual between the two. Figure 5.7 shows the results for a ramp fault input to network III.

Each of the networks did a very good job of estimating the plant variable. To the naked eye, all three networks function approximately equally. They handle startup to steady state and all the transients including the large plant transient around interval 700. We also see that the faults at time 1100 and 1500 are clearly visible. A detectable residual level is present. This detectable residual could warn a plant operator that a failure has occurred in the sensor or channel.

Figure 5.7 shows a plot with a ramp error signal (starting at time 1100) in the same channel using network III. As described above, the error artificially decreases the level in the steam generator at a rate of 3% per day. Again, the results indicate a detectable level visible after a short time (after time 1100). This is a case where the AANN plant monitoring system could alert a plant operator to a drifting sensor or to the on-set of a sensor failure.

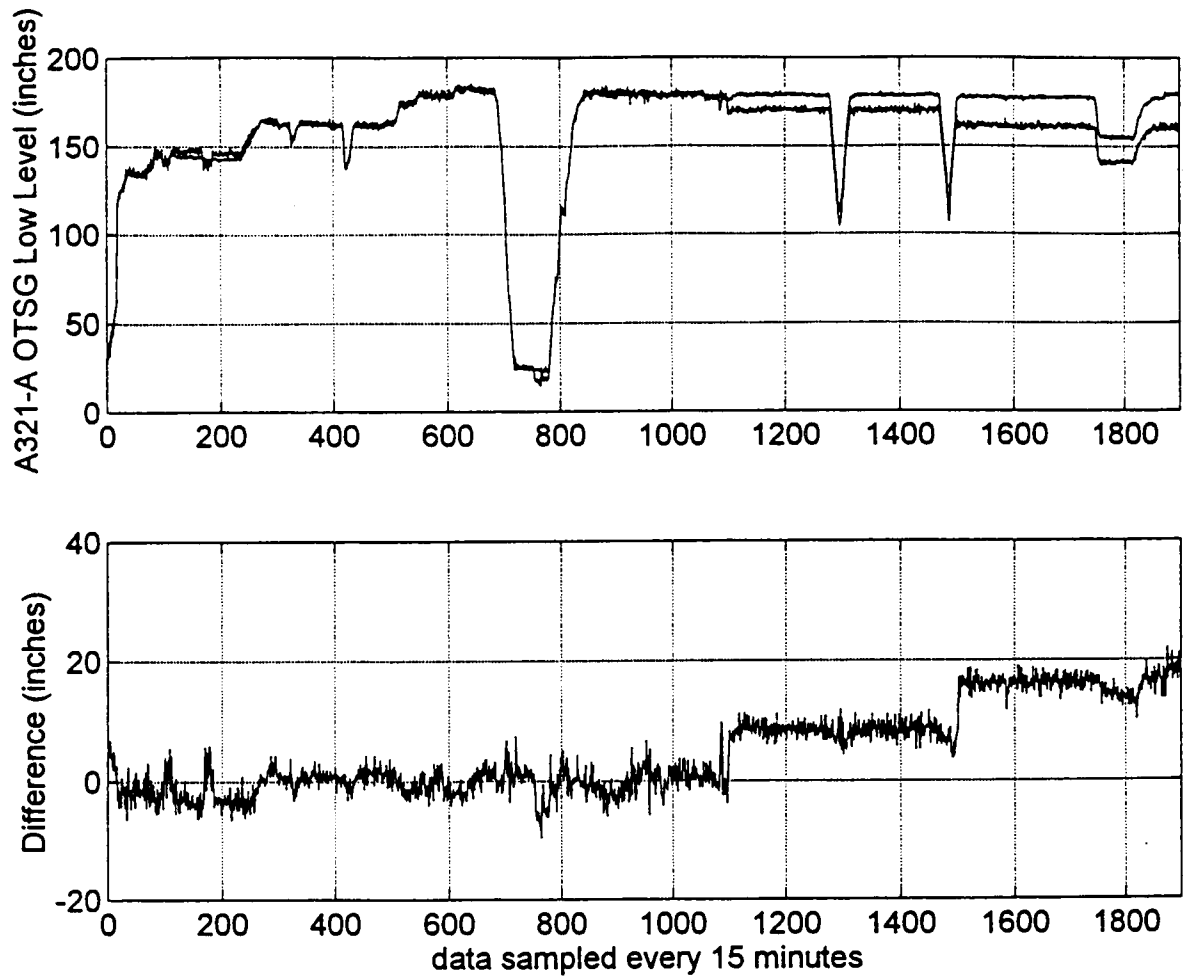


Figure 5.4 Network I step error estimation (top plot) and residual between the actual faulted signal and the networks estimate (bottom plot).

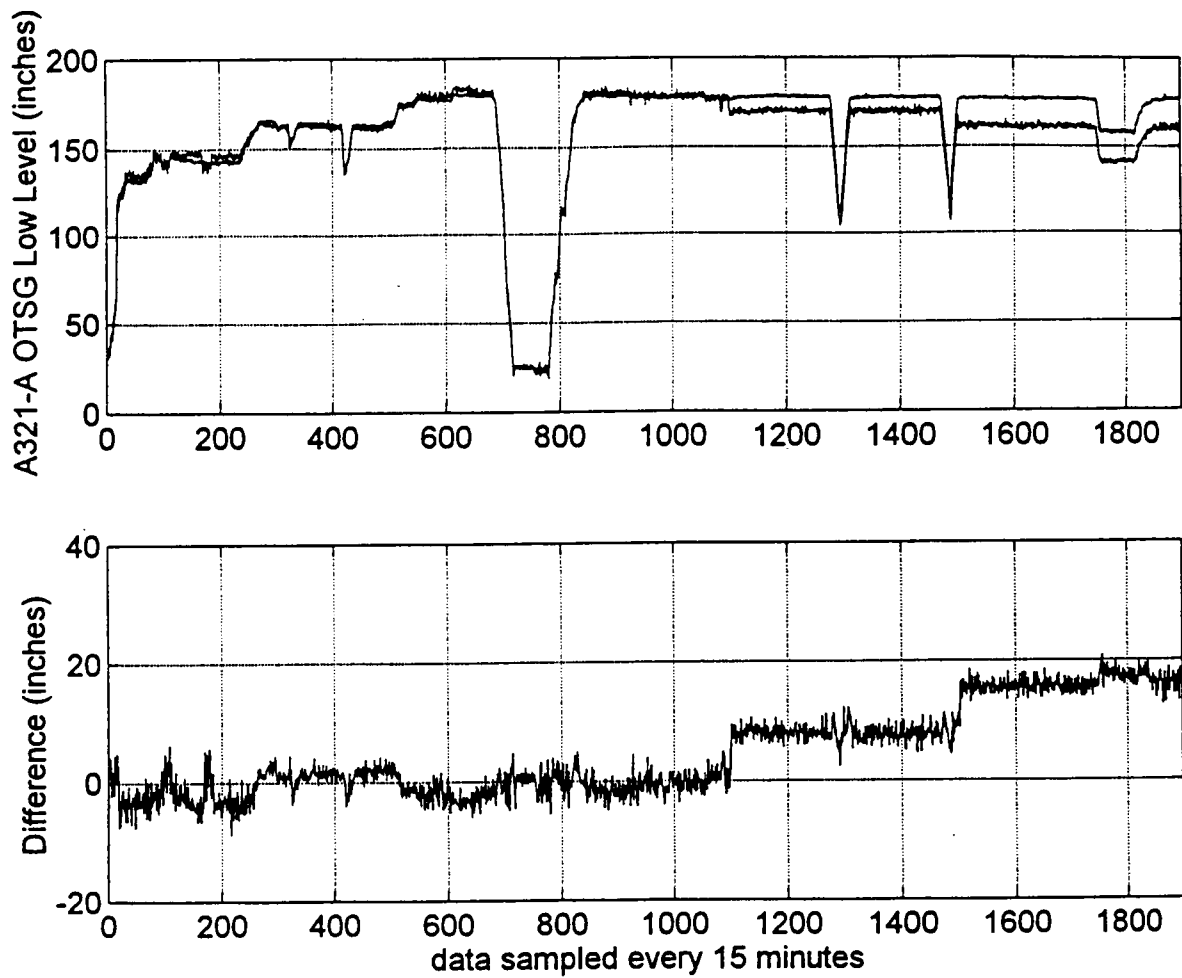


Figure 5.5 Network II step error estimation (top plot) and residual between the actual faulted signal and the networks estimate (bottom plot).

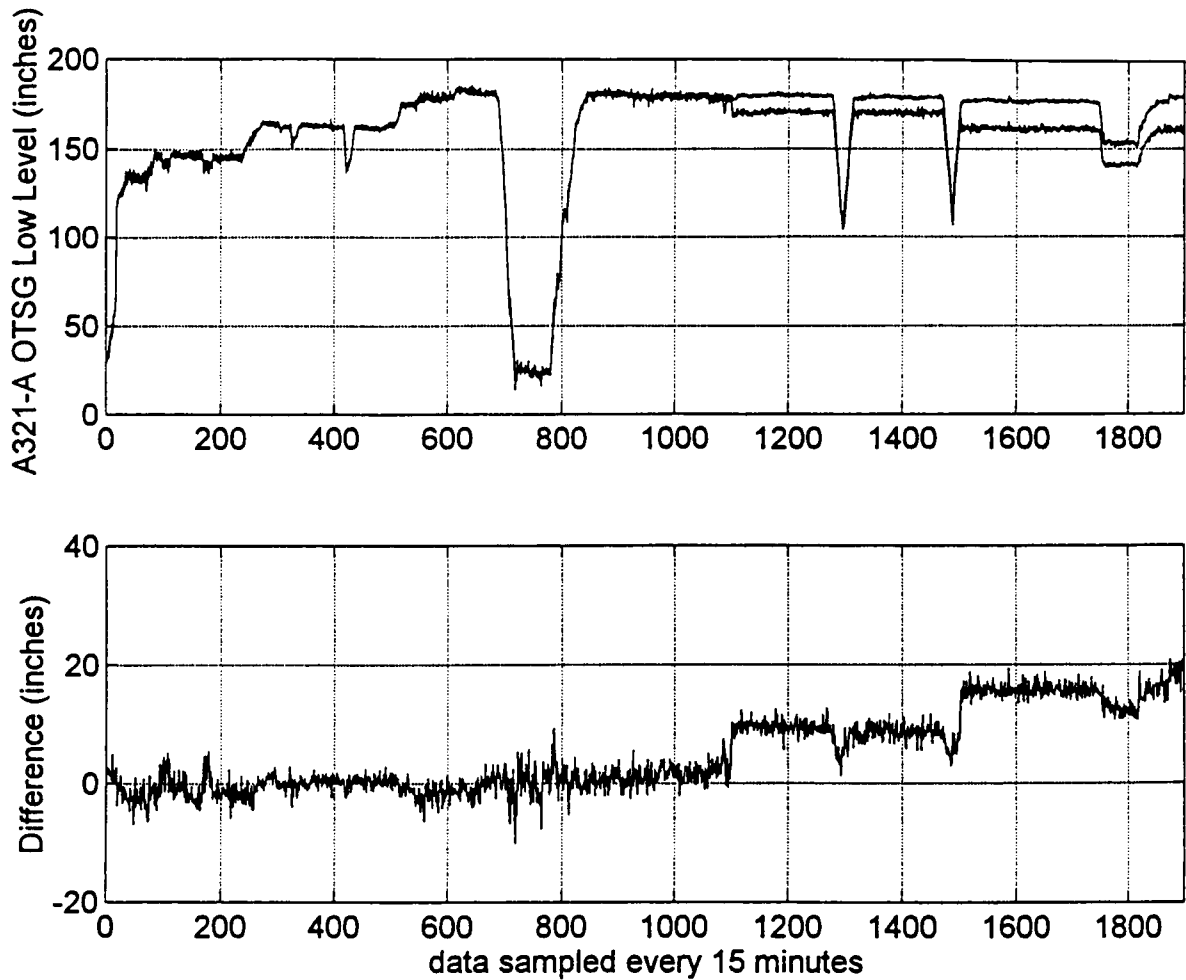


Figure 5.6 Network III step error estimation (top plot) and residual between the actual faulted signal and the networks estimate (bottom plot).

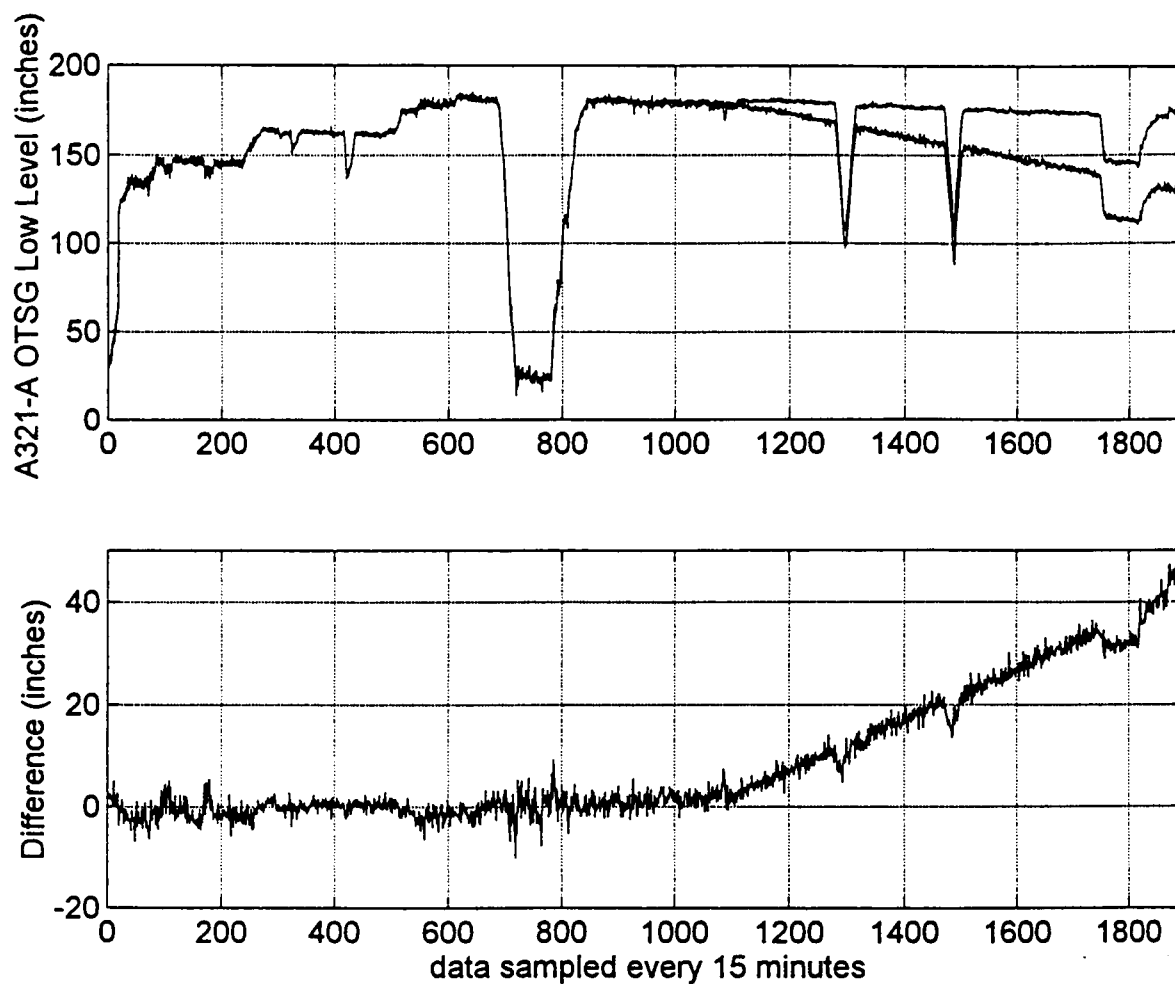


Figure 5.7 Network III ramp error estimation (top plot) and residual between the actual faulted signal and the networks estimate (bottom plot).

5.4.1.5 Final Network Selection

Based on the above results, the three hidden layer feature detection network (network III) was selected as the best autoassociative network architecture for the plant-wide monitoring system. Primary reasons include accelerated training due to linear output neurons using a linear regression Singular Value Decomposition training technique, and the network's robust characteristics. As stated earlier, robustness to faults is the main concern. Recall that the objective is to create a robust network that will estimate the correct sensor value when its input is faulted. When a fault of up to 10% was introduced into a network input, the resulting error between the networks estimate of the faulted signal and the unfaulted signal for network III changed the least at 1.39% over 800 time steps. This small change was compensated for in other network parameters as shown by the total network change of 8.83%. Network III gave the best estimation of the faulted parameter. The total network change does not present a problem due to the fact that it can be minimized by a robust training technique which will be discussed later. Overall, the three hidden layer feature detection network provides the desired robust performance needed for a complex monitoring system. The network will be described in greater detail in the next section.

5.4.2 Three Hidden Layer Feature Detection Network

The network architecture selected for the plant-wide monitoring system is a three hidden layer feedforward AANN proposed by Kramer [1991]. It consists of an input layer, 3 non-linear hidden layers, and a linear output layer. The architecture is shown in Figure 5.8. The first of the hidden layers is the *mapping layer* with dimension greater than the number of input/outputs. The second hidden later is called the *bottleneck layer*. The dimension of this layer is required to be the smallest in the network. The third hidden layer is called the *demapping* layer and has the same dimension as the mapping layer. Kramer points out that five layers are necessary for such nets in order to model nonlinear processes [Kramer, 1991].

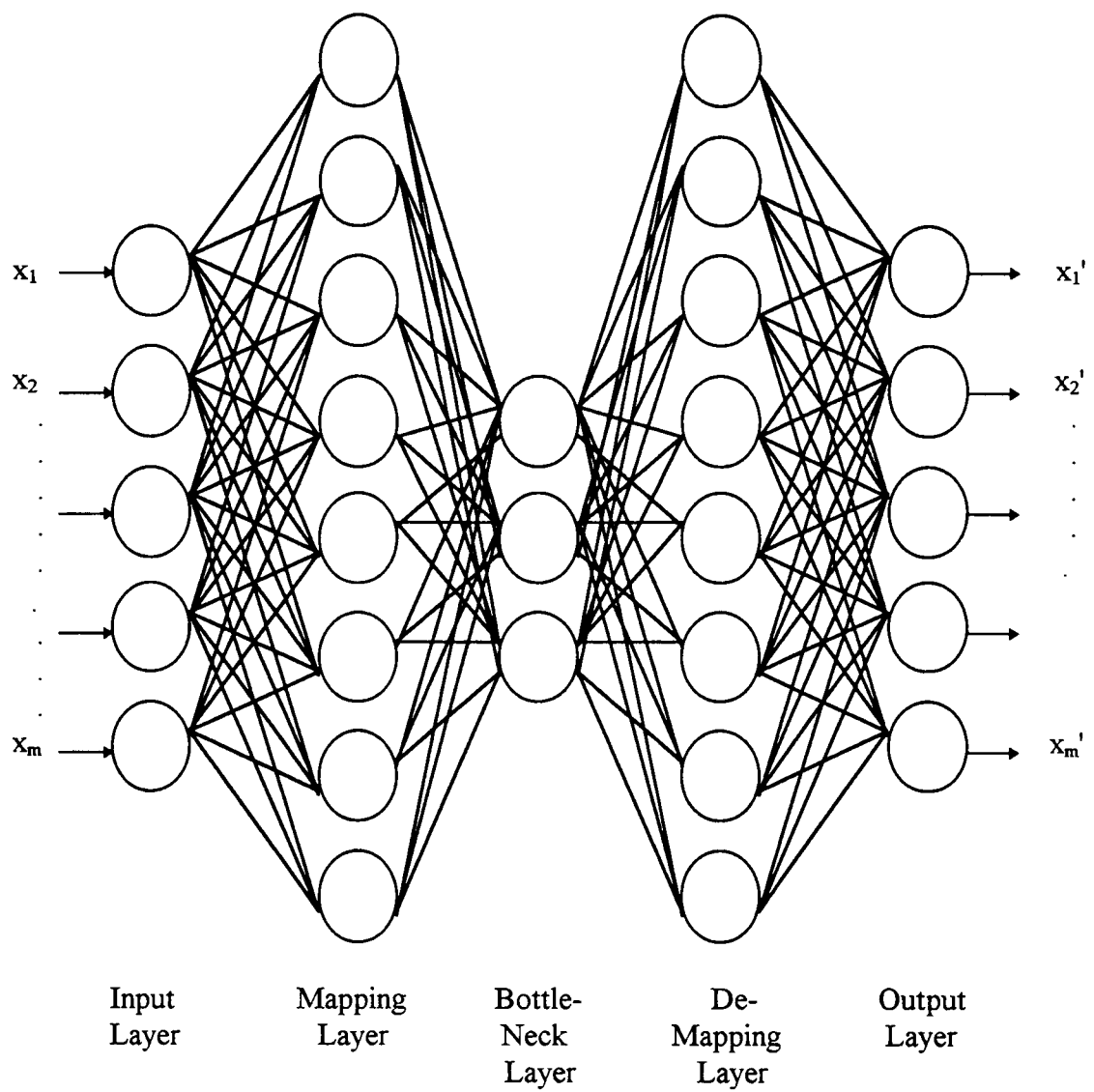


Figure 5.8 Architecture of a three layer "feature detection" AANN.

The three hidden layers form a "feature detection" architecture in which the bottleneck layer plays the key role in the identity mapping. The bottleneck layer prevents a simple one-to-one mapping from developing during training of the network. Essentially the bottleneck layer functions like a nonlinear principle component analysis (NLPCA) filter that gives a compact representation of the data: it uses a lower dimension to explain maximum information. The outputs of the bottleneck layer are nonlinear principal components, which have a clear interpretation in theory. If the network learned the identity function exactly, it would be of little value. The bottleneck layer provides an internal constraint that prevents the net from strictly learning the identity mapping.

The first two non-linear layers in the network map the input data space to the nonlinear principle component space, and the last two layers map the nonlinear principle component space to the data space (network output). The mapping-bottleneck-demapping combination forces the network to develop a compact representation of the training data that better models the underlying system parameters.

The nonlinear transfer function of the three hidden layers are sigmoidal logistic functions given by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (5.2)$$

Figure 5.9 shows a plot of the logistic transfer function. The network uses a linear output layer which allows for a mathematical regression technique (SVD) to rapidly solve for the output weights.

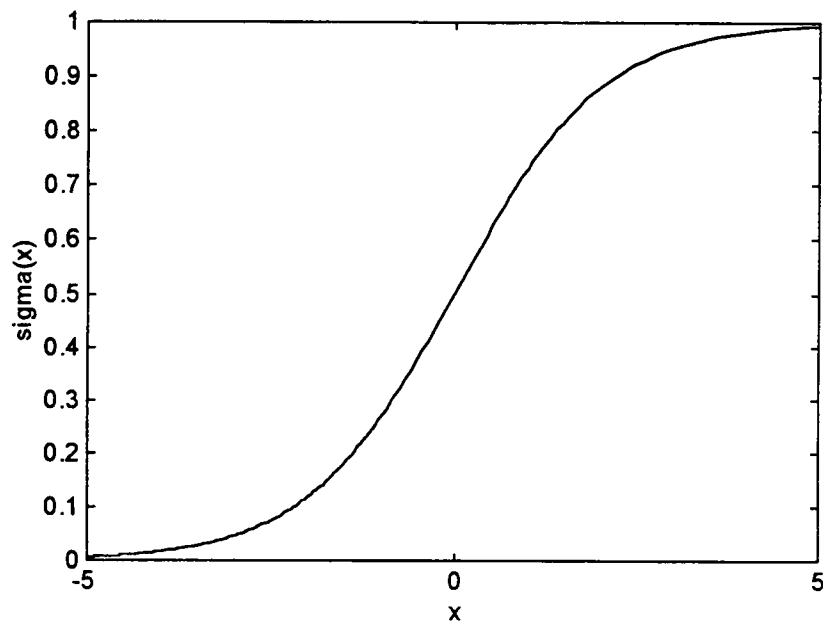


Figure 5.9 Nonlinear logistic transfer function.

5.4.2.1 Selection of Hidden Layer Nodes

In the combined network, let m be the number of nodes in the input and output layers, f the number of nodes in the bottleneck layer, and M the number of nodes in the mapping and de-mapping layers. The number of input and output nodes (m) depends on the number of input/output parameters in the network. However, there is no definitive method of deciding *a priori* the number of nodes in the mapping and de-mapping layers (since the number of mapping nodes is equal to the number of de-mapping nodes, from this point on they will be referred to as mapping nodes). Generally, the number of mapping layer nodes is related to the complexity of the nonlinear parameters that can be modeled by the network. Too few nodes may result in reduced accuracy due to the limited representational capacity (too few degrees of freedom) of the network. If there are too many nodes, the network is prone to overfitting, or learning the stochastic variations in the training data rather than the underlying functions. A simple approach for determining the number of mapping layer nodes is restrict the number of weights in

the network to a fraction of the number of constraints imposed by the network. For the combined architecture, assuming all nodes have biases, the number of adjustable parameters in the network is given by the inequality [Kramer, 1991]:

$$2M \ll m(n - f)/(m + f + 1) \quad (5.3)$$

where n is the number of training patterns. Kramer gives two other methods that may be used to determine the optimum number of mapping layer nodes; the Final Prediction Error (FPE), and Akaike's Information theoretic Criterion (AIC). Research on an automated method for selecting the optimal number of nodes in each layer using principle component analysis is currently underway at The University of Tennessee.

Recall that the bottleneck layer of the network functions as a Nonlinear Principle Component Analysis filter (NLPCA). The bottleneck layer requires as many nodes as there are nonlinear factors in the parameters that are modeled. Therefore, the number of bottleneck nodes can be determined using an algorithm that statistically determines the principle components in a data set.

The nonlinear approach is the same as the linear approach except that the data is summarized with a smooth curve (instead of a straight line) which is determined by nonlinear relationships among all the variables. Given a data set X which contains n samples of m parameters, the data can be expressed in terms of l nonlinear principal components (number of f nonlinear bottleneck nodes) as:

$$X = F(T) + E \quad (5.4)$$

where T is defined as the nonlinear principal component scores, and F is defined as the nonlinear principle component loading functions [Dong and McAvoy, 1996]. An example of such an algorithm to determine the nonlinear principle components can be found in Hastie and Stuetzle [1989].

Due to repeated training and testing trials, empirical formulas have been derived to determine the optimal number of hidden nodes in each of the hidden layers. They are given as ratios of one layer to another (for example input to bottleneck) and are listed below. These are only approximate ratios based on the parameter groupings used in this project.

$$M/m \cong 1.25 \quad M/f \cong 1.6 \quad m/f \cong 1.3 \quad (5.5)$$

where m is the number of nodes in the input and output layers, f is the number of nodes in the bottleneck layer, and M is the number of nodes in the mapping and de-mapping layers.

These ratios were used in conjunction with Kramer's inequality equation given above to determine the number of hidden layer nodes for each of the networks. Table 5.8 lists the number of input/output parameters, the number of mapping and de-mapping nodes, and the number of bottleneck nodes for each of the three networks.

From the table we can see that the number of bottleneck nodes is the same for the two networks with 24 input/output parameters (networks AA and AB) as it is for the network with 22 inputs/outputs (network B). This can be explained by the lower correlation in the smaller network (net B). Because the parameters are not as strongly correlated, more principle components (bottleneck nodes) are needed to represent the more complex mapping present in this group of process parameters.

Table 5.8 Number of nodes per layer.

<i>Network</i>	<i>Inputs</i>	<i>Mapping</i>	<i>Bottleneck</i>	<i>De-mapping</i>	<i>Output</i>
AA	24	31	18	31	24
AB	24	31	18	31	24
B	22	27	18	27	22

5.4.2.2 Noise Filtering

The noise filtering properties of an autoassociative network depend on the ability of the network to produce a model of the parameters that fits the systematic correlations in the data, yet excludes random variations in the signal due to sensor or channel noise.

As discussed previously, the feature detection layer of a three hidden layer AANN functions like a filter that effectively filters out the measurement noise in the signal. During the training process, the network inputs are nonlinearly compressed into f independent variables (bottleneck). If the network is properly trained (in a least squares sense fit of the training data) with adequate data, the network output will be formed from the bottleneck nodes as the closest consistent noise-free state for the given input. The underlying correlations in the parameters will be captured, while noise and non-correlated effects will be excluded. In addition, the SVD technique used to solve for the linear output weights contributes to the networks filtering properties because information believed to be noise is discarded in the weight matrix solution process.

Because of the filtering effects of the network, the residuals formed by comparing the signal to the network estimate of the signal contain random noise with nearly the same variations as the original signal. This eliminates problems due to variations in the network estimate and the signal which may cancel out or have additive effects in the residual variations. Figure 5.10 shows an example of the networks filtering ability. The top plot shows the original input signal to the network (R222 - reactor outlet pressure), while the bottom plot shows the networks filtered estimate. The network has greatly reduced the noise in the process variable. The input signal variance is 3.86 while the variance of the network estimate was 0.002.

5.5 Network Training

Network training methods and techniques will be discussed in this section. Among the topics covered will be training algorithm selection, accelerated training methods, training error goal determination, a "robust" training technique that greatly improves network generalization abilities, and an iterative validation method.

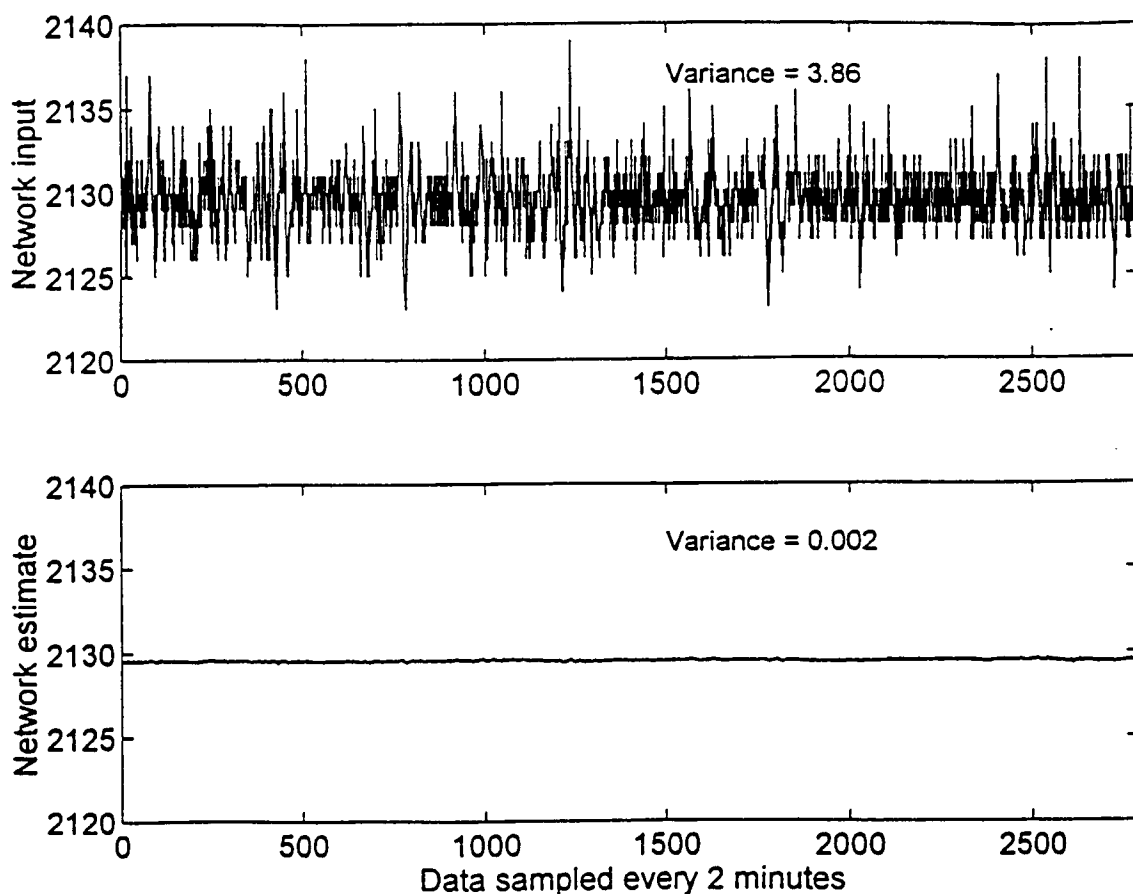


Figure 5.10 Example of the AANN filtering abilities, tag R222-Reactor outlet pressure.

5.5.1 Training Algorithm Selection

As stated in Section 3.7, the major criteria for selecting the training paradigm is the size of the training set and the size of the network. Three training methods described in Chapter 3 were investigated: standard gradient descent based backpropagation, conjugate gradients, and Levenberg-Marquardt (LM).

Based on the discussion of the three training algorithms, the enhanced (or accelerated) backpropagation algorithm was selected for the AANN training method due to its minimum memory requirements and its reliability. Due to the large size of the

autoassociative networks needed for a plant wide monitoring system, the Levenberg-Marquardt training algorithm is impractical due to its memory intensiveness. Since the system will eventually be installed in an operating plant, network training reliability is essential. Moller's Scaled Conjugate Gradients falls short on reliability due to tendencies to get trapped in local error minimums.

MATLAB's [1995] fast backpropagation training algorithm which incorporates an adaptive learning rate and momentum was selected as the training algorithm. The code was modified to allow for the singular value decomposition method to calculate the output layer weight values.

5.5.2 Singular Value Decomposition (SVD)

Since a linear output layer was used, training was greatly accelerated by solving for the output layer weights using a singular valued decomposition mathematical technique (SVD) instead of iterative training. By using the SVD method to approximate the optimal weights, not only do we reduce training time, but the weights are vastly superior to what would be attained by random iterative methods. They provide an excellent starting point for iterative methods applied to all weights in the backpropagation algorithm. A robust training technique will also be discussed which vastly improves the fault detection capabilities of the network.

Given a weight matrix W , target vectors Y , and input vectors X , a standard regression equation can be written:

$$XW=Y \quad (5.6)$$

We can solve for the weights using the general least squares solution:

$$W=(X'X)^{-1}X'Y \quad (5.7)$$

In this solution the mean squared error is minimized with extremely high precision. The weight values produced by this algorithm contain very large values (5 or 6 orders of magnitude) which results in very poor network generalization. Instead of learning the general trend of the data, the network has also learned the noise. The fit to the training sample data will be excellent, but when these weights are called upon to generalize, the result will be very poor.

The solution to this problem is to solve for the linear output weights using the SVD method [Masters, 1993]. The input matrix X is broken down into it's singular value composition given by:

$$X = USV^T \quad (5.8)$$

U = $n \times m$ orthonormal matrix of principle components (columns of U)

S = $m \times m$ diagonal matrix of singular values

V = $m \times m$ orthonormal matrix of right singular values

The final weight values are found by reconstructing the above matrices:

$$W = VS^{-1}U^TY \quad (5.9)$$

In this method only the most relevant information is retained to compute the weights. This is done by discarding the unimportant columns of U . The least important information is discarded because it is most likely resulting from noise. The amount of noise that is removed from the solution to the system is determined relative to the largest weight expected in the network. This is achieved by altering the diagonal matrix of singular value (S). Any values in the matrix which are less than a cutoff value (ϵ) multiplied by the largest weight value in S are changed to zero in the inverse matrix. Experimental results show that keeping the weights within ± 10 provides excellent

network generalization with no loss of important information. The cutoff value ϵ , used to achieve weights of this size was 3.75×10^{-3} .

5.5.3 Optimal Training Error

It was determined early on in this study that the training error goal of the network is a very important consideration for a robust network that can generalize well. Real plant operational data contains a small amount of noise (typically 2 to 3 percent) from the process, sensors, and other electronic equipment. If a network is trained to a low error value, it will tend to model the noise in the data and not the overall functions inherent in the data. This phenomenon is known as overfitting the training set and results in a network that has very poor generalization ability. Two methods are used to prevent overtraining the network. The first method is to statistically determine the amount of noise in the signals and calculate an overall training error goal (training stopping criteria) from the noise estimates. The second technique used is cross validation training to verify the results of the first method.

5.5.3.1 Statistical Technique

The statistical method of error goal determination consist of filtering out the noise in each of the parameters and then finding the standard deviation of the difference between the filtered signal and the original signal. The filter used to filter the input parameters is a zero phase shift median filter which gives the median value for a window size of 6 data points (3 forward and 3 delayed). The sum squared error (SSE) goal used to train the network is then found by multiplying the variance by the number of input parameters. This technique was found to be a good error goal estimator.

5.5.3.2 Cross Validation Training

Cross validation training is used to verify the above results. Cross validation training is a method to determine the training iteration when overfitting of the data begins to occur, so that training can be stopped [Masters, 1993]. Since the goal is to train a network so

that it will have the best performance on new data, an approach is to evaluate the error function using data which is independent of that used for training.

To do this, two data sets are used during the training of the network; the original training set, and a set drawn from the same population as the training set (but different) known as the *validation* set. The error for the training set monotonically decreases approaching an asymptote. The validation set's error decreases at the beginning of training and continues to decrease until a certain iteration when it then begins to increase. At this point the network is learning to model the random noise in the training set, and since the validation set's noise is different, its error rises. It is at this point, when the validation set error begins to increase, that training should stop. Continuing network training after this point will degrade the generalization ability of the network because it begins to learn the noise in the training data and not the underlying functional relationship [Bishop, 1995].

Figure 5.11 illustrates an example plot of the training record during a typical cross validation training session. Mean squared error is used during this type of training (instead of sum squared error) because in most cases the testing set does not contain the same number of patterns as the training set. The training error decreases rapidly at first and then approaches a minimum asymptote. The testing error initially decreases rapidly, reaches a minimum around epoch 180, and then the error begins to increase. It is at this point where the testing error reaches its minimum that training should cease. If training continues, the network is only learning the noise in the training data set and not the overall functional relationship behind the process.

5.5.3.3 Final Network Training Errors and Comparisons

Table 5.9 list the optimum sum squared error (SSE) goal determined by each of the two methods discussed above (statistical and cross validation). The SSE determined by each of the methods is in excellent agreement with each other (i.e. by a factor of approximately 2) considering the SSE goal is over all the patterns and all the parameters

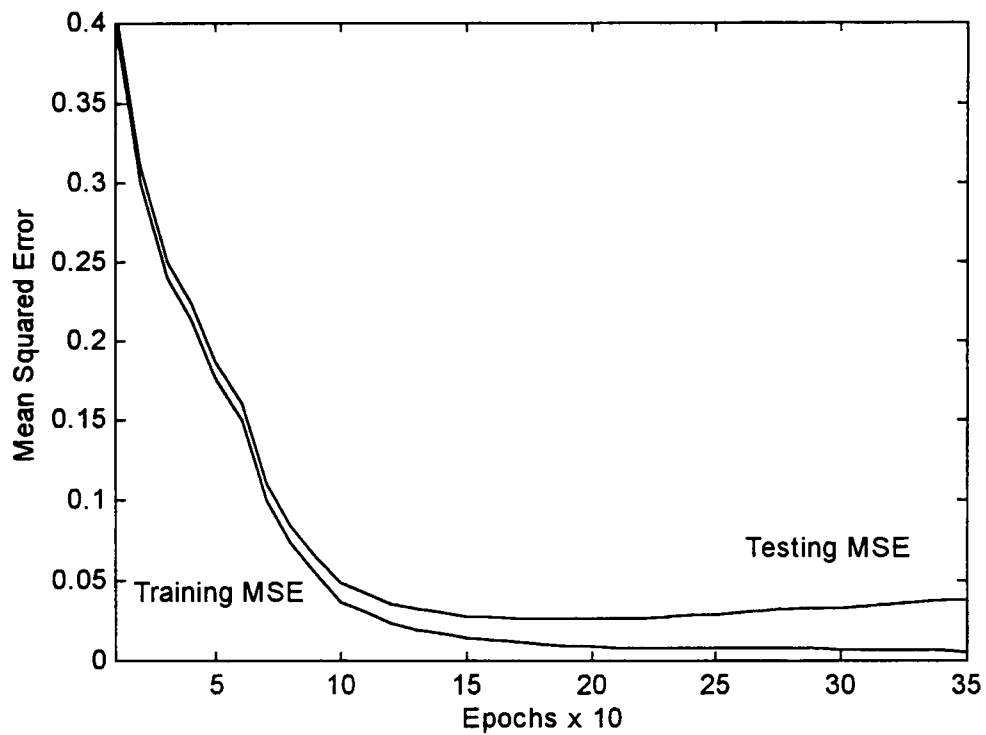


Figure 5.11 Cross validation training error curves.

Table 5.9 Network SSE training goals.

<i>Network</i>	<i>Statistical Method SSE</i>	<i>Cross Validation SSE</i>
AA	3.2	5.7
AB	2.9	5.1
B	4.1	6.3

in the training set. The SSE training goal determined by the noise levels present in the data was used to train the networks in each case.

5.5.4 Robust Training Technique

A robust network is one that estimates a correct output for a respective input that contains an error or missing data without unduly disturbing the output estimates of the other parameters in the network. Robustness to errors is not inherently a property of AANNs. Feed forward networks generally have poor extrapolation properties. Sensor failures have no precedent in the training set, therefore the network's estimate to an erroneous input is unpredictable. An error introduced into a single sensor may be detected, but at the risk of compromising the remaining signals in the network. It would be known that an error has occurred in the system, but it may not be possible to isolate which sensor is at fault.

To achieve true robustness, specifically to produce a non-corrupted output value for inputs containing errors, the network must be trained on exemplars that represent the desired input/output behavior. To achieve this behavior, the original training set is augmented by new patterns with training inputs $X' = X + \delta I_j$ $j=1, \dots, m$ (m is number of input signals), where I_j is the j th column of the identity matrix, and the target outputs are the original unfaulted target outputs Y . For each unfaulted training example, each sensor is corrupted several times using different random values of δ ranging between ± 10 percent of the original signal value. If there are n original training samples, there will be $2n$ samples in the robust training set. Using the corrupted input set X' , a robust network will be trained to produce the non-corrupted output Y . Figure 5.12 shows a plot of a standard training set where the input vectors are the same as the output vectors. Figure 5.13 shows a robust training set in which half of the input patterns are faulted as discussed above and their outputs are the original unfaulted target patterns.

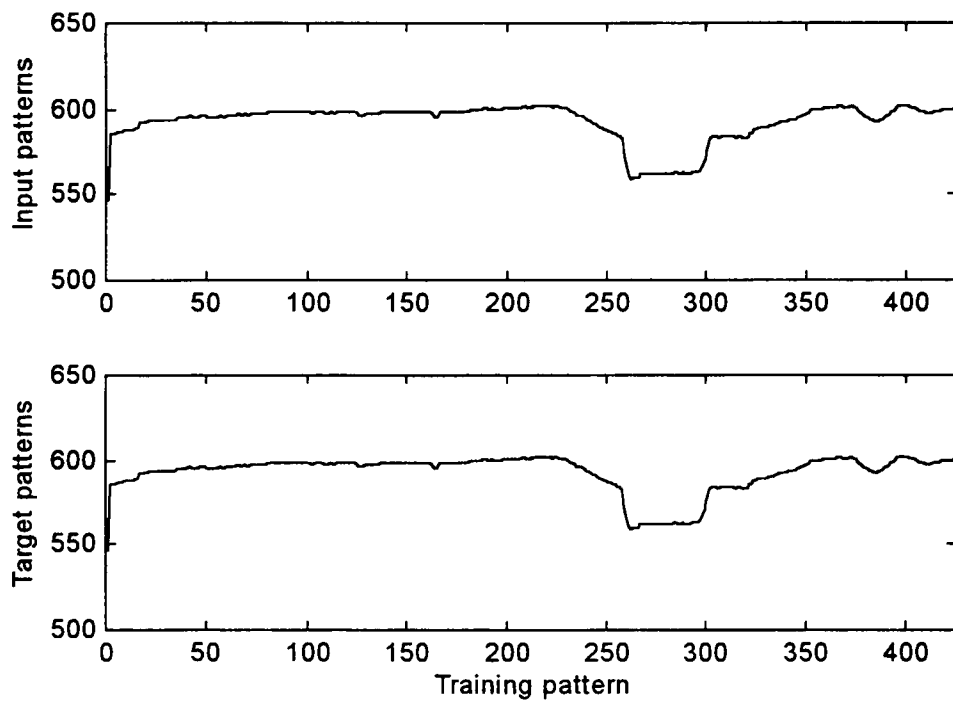


Figure 5.12 Non-Robust training set (input = target).

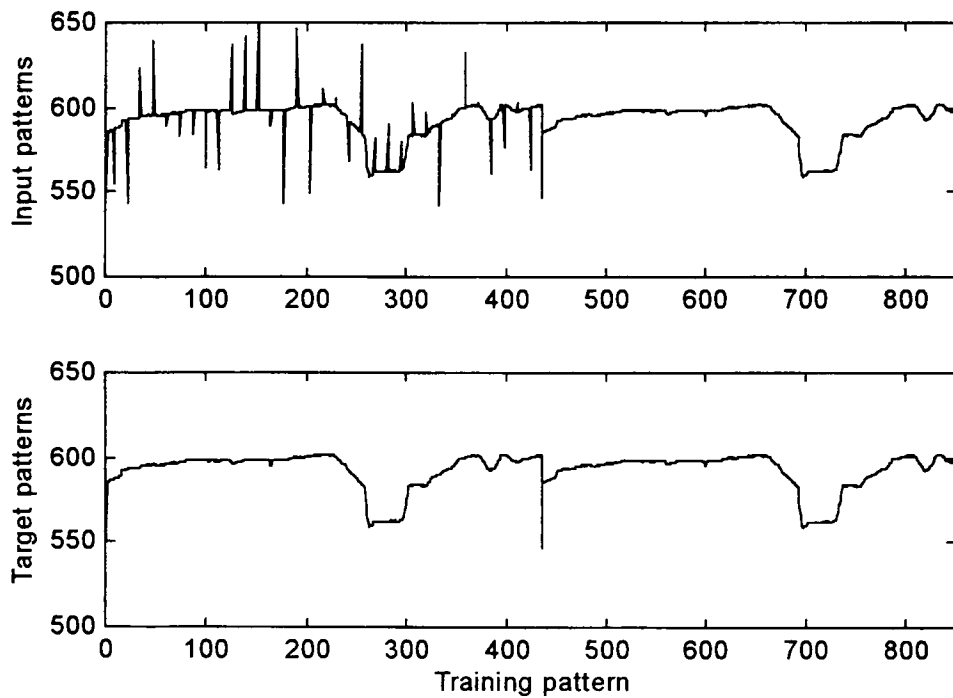


Figure 5.13 Robust training set.

5.5.5 Training and Validation

Once the inputs and outputs are selected and preprocessed, the desired architecture is chosen, and the training paradigm and error goals are selected, the networks is trained and validated. Validation involves selecting and constructing several sets of data to exercise the input space and evaluate the network performance. This is an iterative process since poor performance on the validation set may require merging it with the training set, retraining the network, and selecting a new training set. Thus an iterative train and test procedure was used to achieve optimal performance with the three networks discussed earlier.

After training is completed, data that was not used for training is used to test the network. The network outputs, which are the estimated sensor values, are compared to the measured sensor values and the residuals are calculated. The residuals are fed into the SPRT algorithm to determine if a sensor is faulty. The selection of the SPRT parameters will be discussed in the next session. The results of this test are examined and the training set is changed accordingly. False alarms during testing require data from that operating condition to be added to the training set. When the networks are properly trained not to give false alarms (the network has been trained on all reasonable input regions), data with simulated faults is used for input to the network. The outputs are examined to verify that the estimated sensor values equal the actual unfaulted sensor values within certain error tolerances. These tolerances are determined by the amount of noise in the data. The sensitivity of the network is determined by noting when the SPRT module detects the drift. If the network does not perform well, the training set must be augmented and the network retrained.

Sometimes drift in one sensor may produce alarms in other sensor channels that are not faulty. In this case, the training data is augmented with data that has simulated drifts; and the network is trained to produce the unfaulted sensor values. This training makes the networks more robust and has been cited in several references [Kramer 1992, Demuth and Beale 1994].

The iterative network training and testing task is very time consuming. The first few networks are used as examples to set up automated routines for training and testing the later versions of the networks.

5.6 SPRT Decision Logic Module

The outputs of the neural networks are estimates of the monitored parameters. These estimates are then compared to the measured parameters and the residuals are calculated. These residuals are used as inputs to decision modules which determine whether the residuals are statistically significant to warrant labeling them as in error. The sequential probability ratio test (SPRT), which was originally developed by Wald [1945], can detect a failure with the minimum number of samples based on specified missed and false alarm probabilities [Upadhyaya, 1987]. The SPRT module was derived and explained in Chapter 4.0.

The variables necessary to implement the SPRT include:

1. Missed alarm probability
2. False alarm probability
3. Measure of noise in the signal (variance)
4. A faulted mean.

After experimentation with the SPRT module, we have decided to use a 0.01% false alarm probability and a 10% missed alarm probability. As discussed in Chapter 4, it is important to maintain a low false alarm probability in any failure detection systems. By setting the probabilities in this manner, we reduce the number of false alarms so that user confidence is improved. The larger probability for missed alarms is not a problem because the monitoring is continuous and cumulative. If the alarm is missed at this sample, it will probably be detected at the next sample. Since we are concerned with slow sensor drifts (on the order of hours or days), quick detection times are not critical.

To determine a value to use for the faulted mean we consulted with Rollin VanAlstine and agreed to use maximum faulted mean of 2.5% of the value of the sensor's full range reading. The majority of the sensors were set to lower tolerances. It is also noted that many of the sensors have noise levels above this setpoint. The final faulted mean and variance values for each of the Technical Specification signals are listed in table 5.10 in Section 5.8.3.1.

5.7 System Integration

System integration includes connecting the modules of the system and testing their performance. First, each network system (Net AA, AB, B) will be explained and then a complete autoassociative block system will be explained.

To monitor the performance of the networks, the sequential probability ratio test (SPRT) was implemented using SIMULINK. As discussed earlier, SIMULINK is a program for simulating dynamic systems. Each sub-system (AANN, SPRT, filter, etc.) is represented in block form, with the blocks connected together to form a functional dynamic system. Figure 5.14 shows an exploded view of the AANN block. The parameters come in through a port and are pre-processed (scaled from 0.1 to 0.9), they then pass through the 4-layer AANN (5 layers when counting the input layer) where the network output estimates are the post-scaled to their operational range, they are then output through a port to the next block. Multiplexer and demultiplexer blocks combine or separate signals to allow the lines to carry multiple signals (vectors).

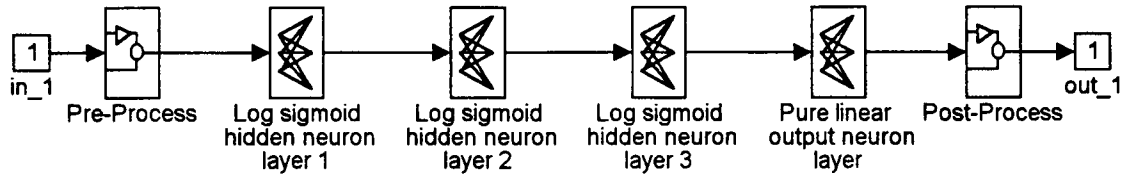


Figure 5.14 Exploded view of an AANN block.

A typical system block diagram is shown in Figure 5.15. The AANN estimate is compared to the measured signals and residuals are formed. The residuals are then sent to two separate SPRT blocks (one for drift detection and one for gross channel failure detection) which output the status of each sensor (0 = good, 1 = bad) based on the variance of the residuals, a given faulted mean, a preset false alarm probability of 0.01% and a missed alarm probability of 10%. The output of the SPRTs are then processed by a logic filter with a 2 out of 4 delayed status voting scheme. The purpose of the filter is to prevent false alarms caused by spurious spikes in the plant computer system. The graphical oscilloscopes display information to the user. In addition, information is stored in work space for further examination.

5.7.1 Individual Networks

As discussed earlier, there are three separate networks monitoring the plant parameters in this system. Networks AA and AB, each model 24 parameters, and network B which models 22 parameters. These two separate SIMULINK systems are shown in Figures 5.15 and 5.16.

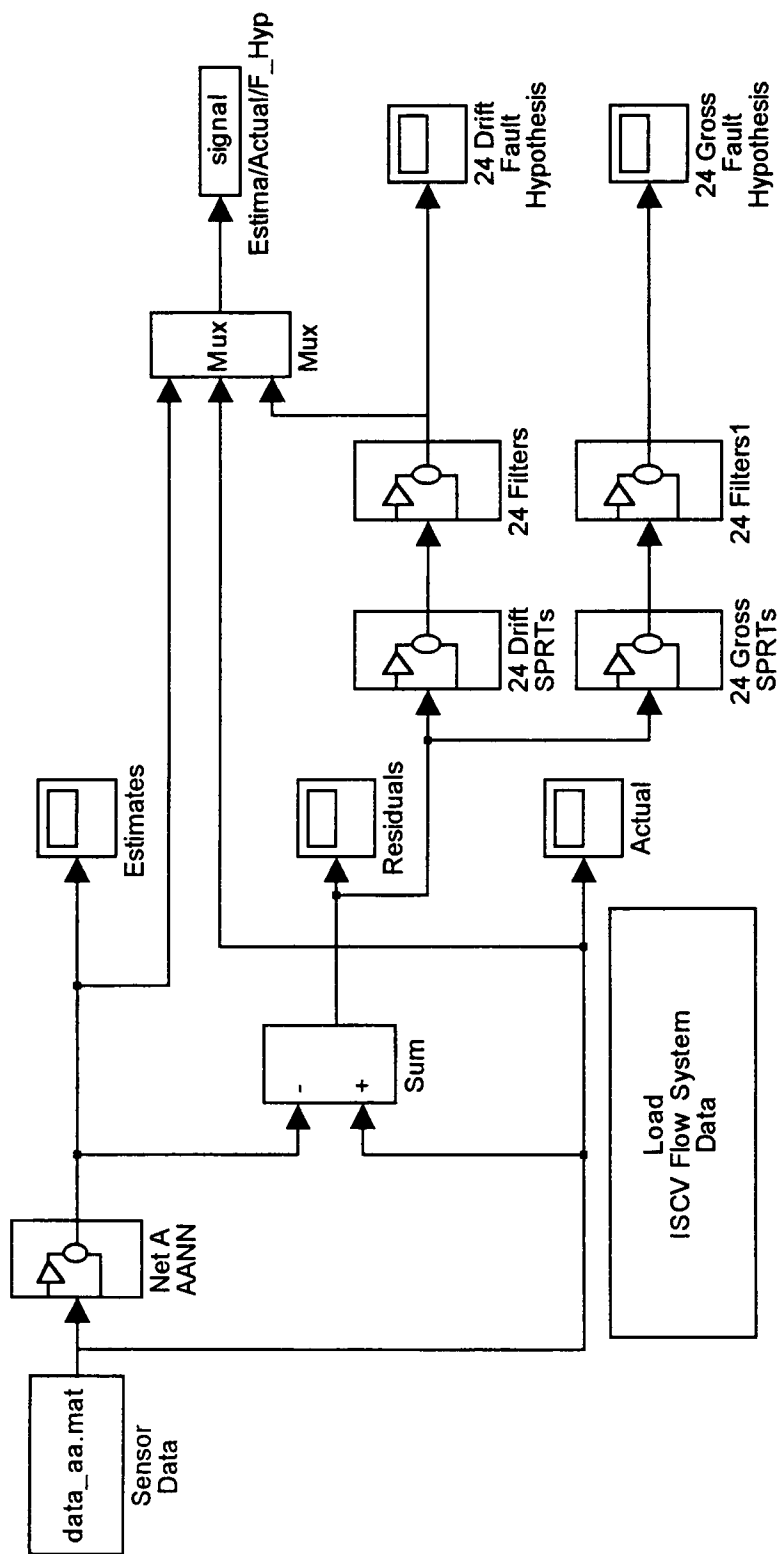


Figure 5.15 Network AA and AB SIMULINK ISCV system block (24 parameters modeled).

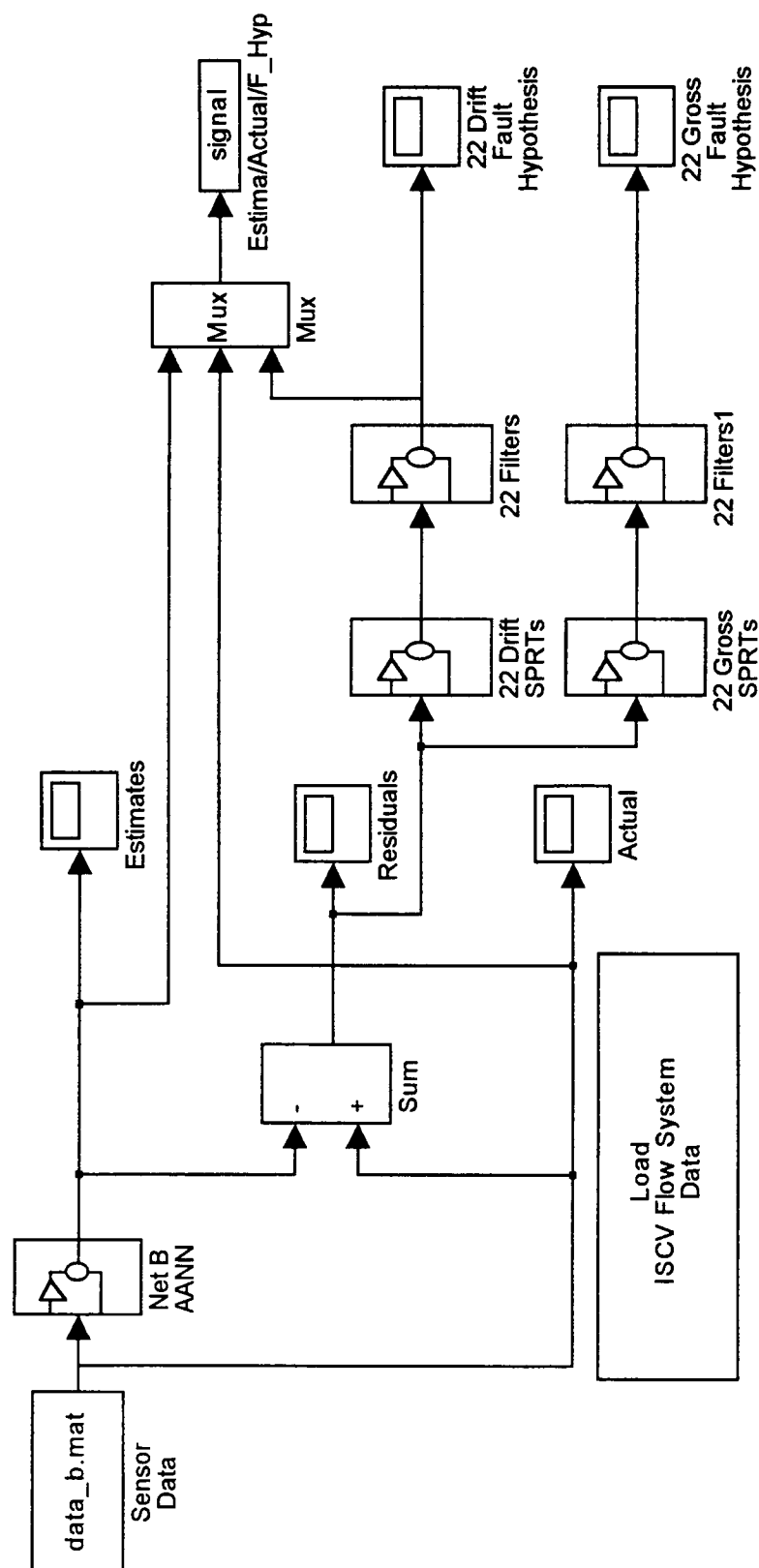


Figure 5.16 Network B SIMULINK ISCV system block (22 parameters).

5.7.2 Integrated System

The total integrated system is then formed by connecting all subsystems and its overall performance is tested. The ISCV system block diagram is included as Figure 5.17. In this figure, plant sensor data enters the system where it is separated by the demultiplexer into its respective variables for networks AA, AB, and B. The data is then pre-processed, simulated by the network, and post-processed to create the networks' sensor estimates. The estimates are then compared to the original signals (by the sum blocks) and residuals are formed. The residuals for each network then branch off in a parallel fashion; one branch to a drift detection (high sensitivity) SPRT block, the other to a gross fault detection (low sensitivity) SPRT block. As before, the SPRT outputs are filtered to remove spurious spikes in the data. The signals from each of the 6 filter blocks are then multiplexed into a decision logic module. Currently, the decision logic module is simply a group of OR gates used to group the parameters used in more than one network (for instance, tag R327 is used in network AA as well as network B). The outputs are then demultiplexed to two separate graphical oscilloscope displays for gross and drift fault detection where all parameters can be monitored simultaneously.

5.7.3 Redundant Sensors

There are several groups of redundantly measured Technical Specification variables in Florida Power Corporation's Crystal River nuclear power plant. To detect sensor failures for these groups of sensors, a neural network was designed for each group of redundant sensors. It was decided that the only inputs necessary to estimate the correct sensor outputs are the redundant signals. Including other signals in the AANN is not necessary due to the almost perfect correlation between the redundant signals. In addition, the perfect correlation of the redundant sensors in a larger multivariable network could cause the network to rely heavily upon the redundant sensors.

Small-sized, three-layer (input layer, one hidden layer and output layer) autoassociative neural networks were designed to model the redundant sensors. The Levenberg-Marquardt training algorithm, which has the fastest convergence and is

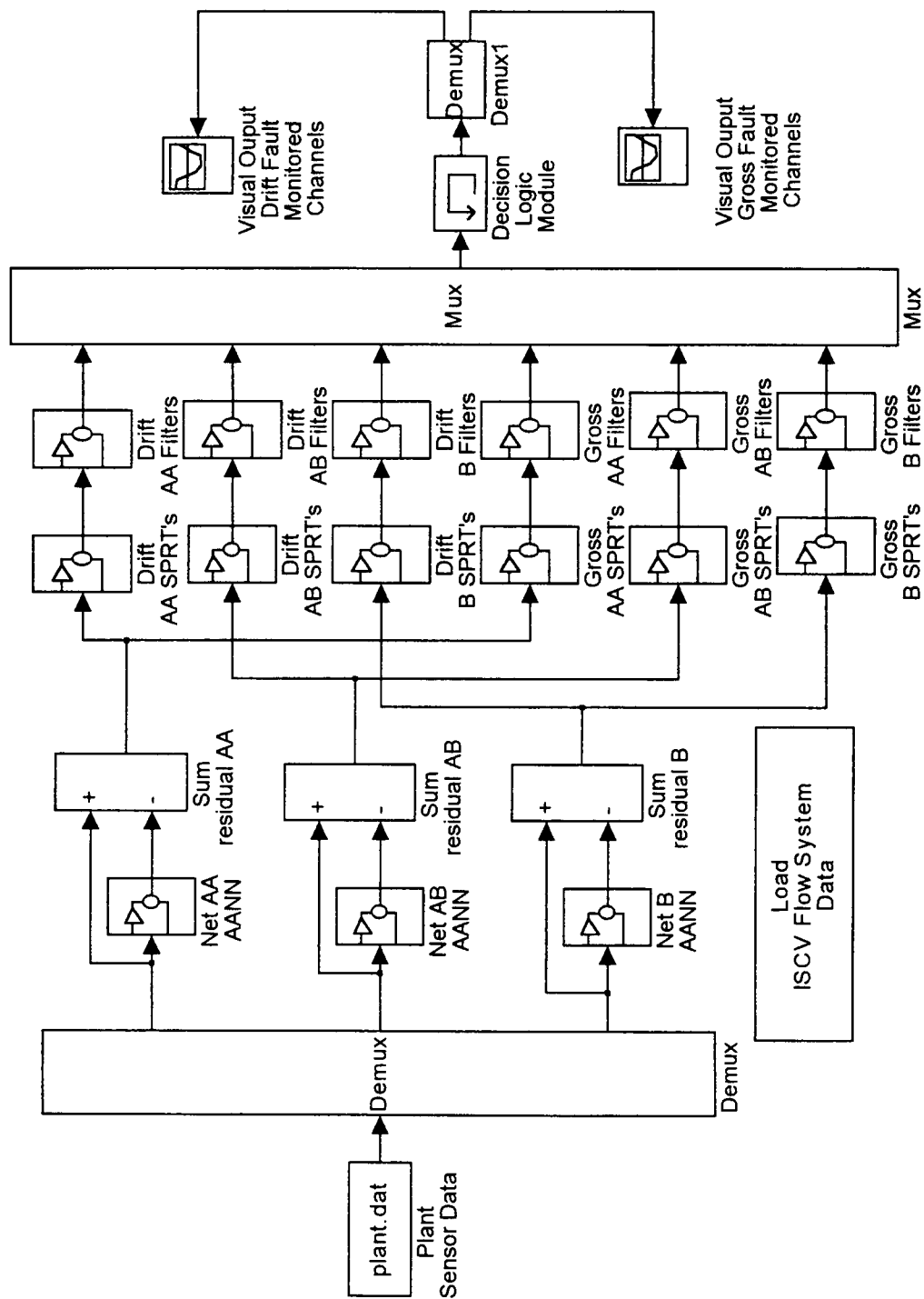


Figure 5.17 Complete integrated AANN ISCV system.

suitable for small networks and small training set sizes, is used for training the autoassociative networks. The robust training technique was used so that the network relies evenly on all of the redundant inputs.

Table 5.10 list the redundant sensors networks. The work performed on the redundant sensors was done by Xu and is included as part of a larger report prepared for Florida Power Corporation [Uhrig, Hines, Black, Wrest, and Xu, 1996]. Drift error detection results are similar to the results found in this study.

5.8 System Testing

5.8.1 Testing Data Selection

Network testing was performed using data from the "fault free" five week period during full power operation. The non-consecutive five week period was given to us by Rollin VanAlstine, Senior I&C engineer at Florida Power Corp. During these five weeks the

Table 5.10 Redundant sensor networks.

Signals	Tag ID	Redundant Network
A-Loop Outlet	R212	yes
Temperature (deg F)	R226	
	R227	
B-Loop Outlet	R213	yes
Temperature (deg F)	R228	
	R229	
Pressurizer Level (inch)	R200	yes
	R201	
	R202	
A-Loop Flow (Mlb/hr)	R234	yes
	R236	
	R238	
	R240	
B-Loop Flow (Mlb/hr)	R235	yes
	R237	
	R239	
	R241	

plant operation was believed to be "fault free", in that no adjustments were made to sensors or instrument channels. The five non-consecutive weeks are listed below.

5 weeks:

6/28/93, 7/12/93, 7/26/93, 8/09/93, 8/23/93

The five week period was primarily steady state plant operation with one transient. Since the primary goal of phase 1 of the ISCV was to investigate and document an intelligent plant wide sensor monitoring system that functioned under steady state conditions [Rutherford, 1996], a 4 day period (July 16, 1993 to July 20, 1993) of steady state operation was used for system testing. The test data was sampled every 2 minutes giving a total of 2880 test points. In addition, several tests were run using the entire 5 week period, which included the transient, with data sampled every 15 minutes.

5.8.2 System Testing Guidelines

The following list describes the tests that were performed on each of the networks. Each sensor input of each network was tested for performance with the following five test cases. The results for each of the faulted test cases is summarized in Tables 5.11 and 5.12 and include the Technical Specification signal tag number, the pre-set faulted mean, the pre-set residual variance, the time error detected, the percent drift of the signal, and the number of false alarms recorded in all the other channels in the network. Drift fault detection and gross fault detection are discussed separately. The five test cases are:

- 1.) No faults
- 2.) Drift fault "high" in each sensor equal to a ramp of 0.10 % per day of the instruments maximum scale deflection.
- 3.) Drift fault "low" in each sensor equal to a ramp of 0.10% per day of the instruments maximum scale deflection.

4.) Gross fault "high" in each sensor equal to the instruments maximum scale deflection. The will be initiated approximately 1/4 of the way into the simulation.

5.) Gross fault "low" in each sensor equal to the instrument minimum scale deflection. The fault will be initiated approximately 1/4 of the way into the simulation.

SPRT detection:

All faulted means will be set to a maximum of 2.5% of the instruments maximum scale deflection (but most much lower).

False alarm probability of 0.01% (alpha).

Missed alarm probability of 10% (beta).

5.8.3 Results

The following sections present the results of the above testing procedure. Section 5.8.3.1 shows the advantages of using a robust training method by use of sensitivity analysis on the network. It also gives a brief description of how sensitivity analysis can be used as a tool to refine the parameter groupings. Section 5.8.3.2 shows the results of drift detection testing, and Section 5.8.3.3 explains the methods used and results for gross error detection.

5.8.3.1 Sensitivity Analysis and Robust Training

As discussed earlier, sensitivity is defined as the change of the output over the corresponding change in the input. Given the input vector x and output vector y :

$$S_x^y = \frac{\partial y}{\partial x} \quad (5.10)$$

The sensitivity analysis of the network is shown by a cumulative bar plot which shows the change in each of the networks outputs due to a 5% perturbation in each of its inputs. One plot shows the sum of the output change for each signal, and the other shows a normalized output change which indicates the contribution of each of the perturbed inputs in the network to the corresponding output change in each signal. This is shown by horizontal bars in each of the outputs.

Figure 5.18a (top plot) shows the results of the sensitivity analysis for the non-robustly trained network, while Figure 5.18b (bottom plot) shows the sensitivity analysis for a robustly trained network for network B. For the non-robust case, a perturbation of 5% in each input of the network results in a change in the output of between approximately 10% up to 900% with the contribution of other signals varying greatly to the overall change in each output (as noted by the 22 horizontal bars in each of the 22 signals). Due to the large changes in the network output due to small input perturbations, the network will not perform well under faulty input conditions.

Now examine the sensitivity results for a network trained with a robust training set. A substantial improvement can be seen over the non-robust network. Each of the 22 outputs in the network respond equally to a 5% perturbation with the output sum change varying from 2% to 5%. We also note that the cumulation of sensitivities of each of the channels is approximately the same. Under faulty conditions, we would expect this "robust" network to perform well. Sensitivity plots for networks AA and AB show similar results in Figures 5.19 and 5.20 respectively. This shows that the robust training made the network outputs depend on each of the inputs equally.

5.8.3.2 SIMULINK Simulation Example

A network simulation was performed using both the non-robust network and the robust network to verify the results of the sensitivity analysis. One network was trained using the standard non-robust training set and the other was trained using an augmented robust training set. Other than different training sets, all network parameters were the same. The two network's simulations were compared using the SIMULINK ISCV

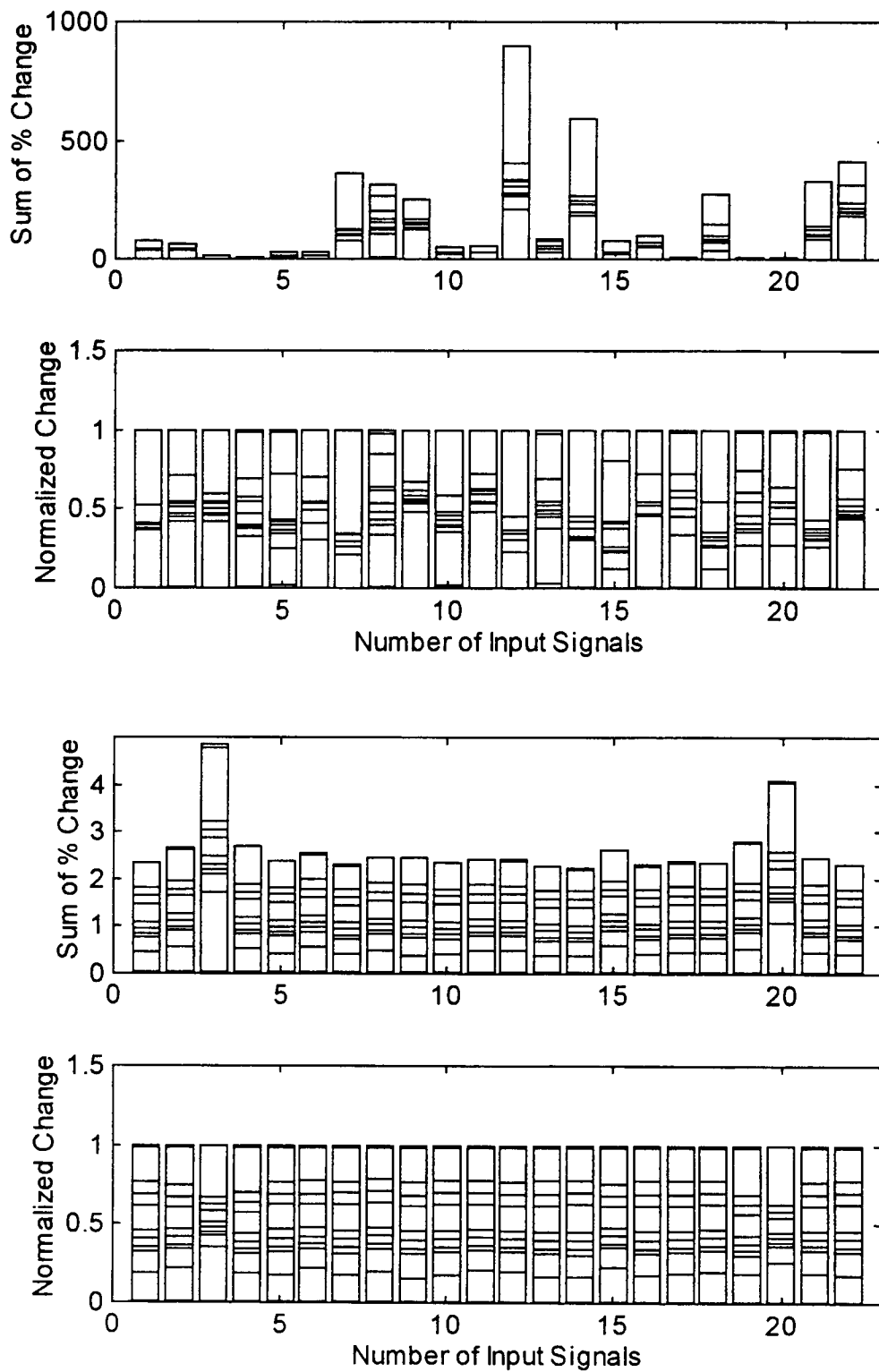


Figure 5.18 Network B sensitivity analysis a.) non-robust training b.) robust training

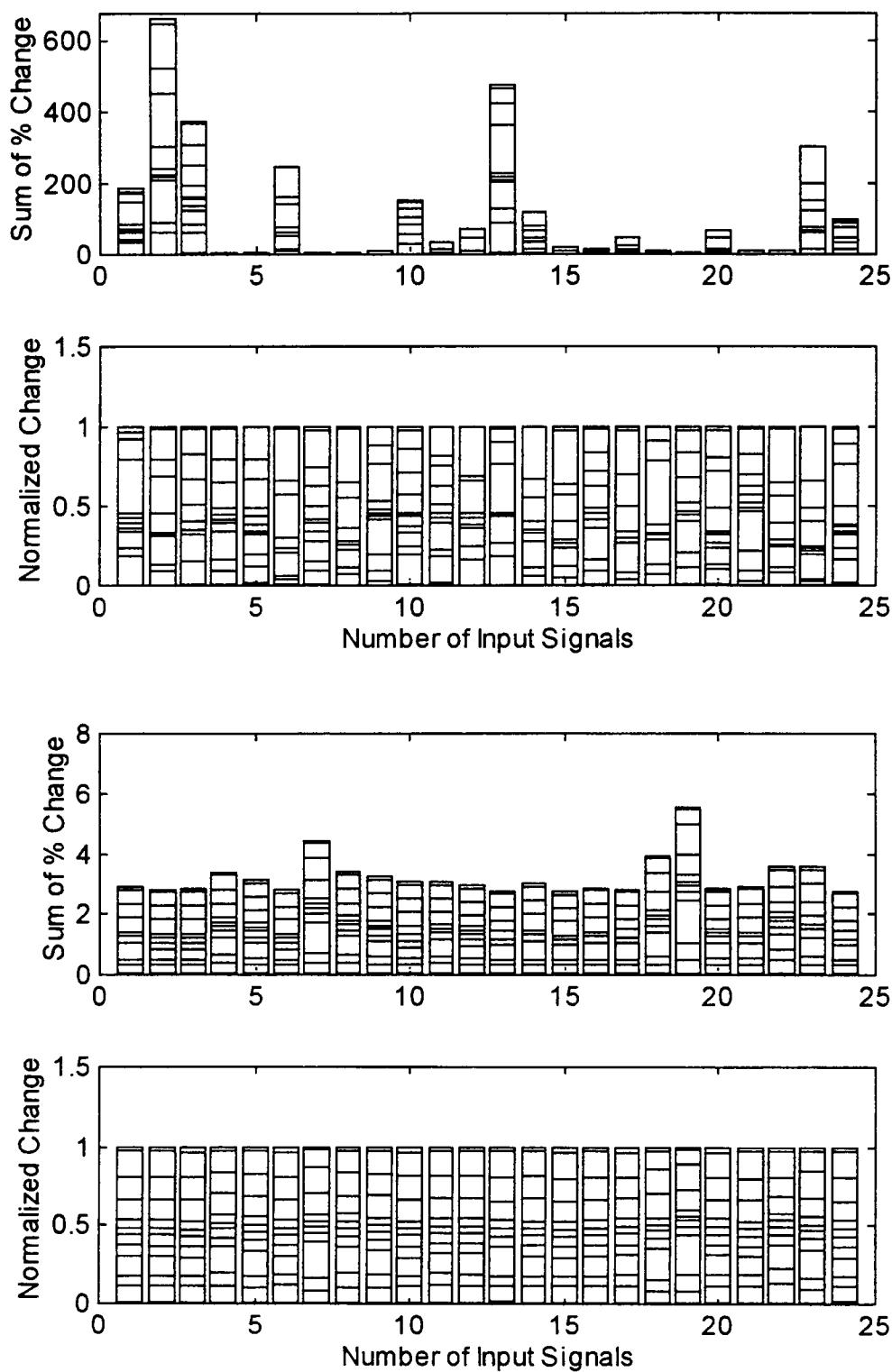


Figure 5.19 Network AA sensitivity analysis a.) non-robust training b.) robust training

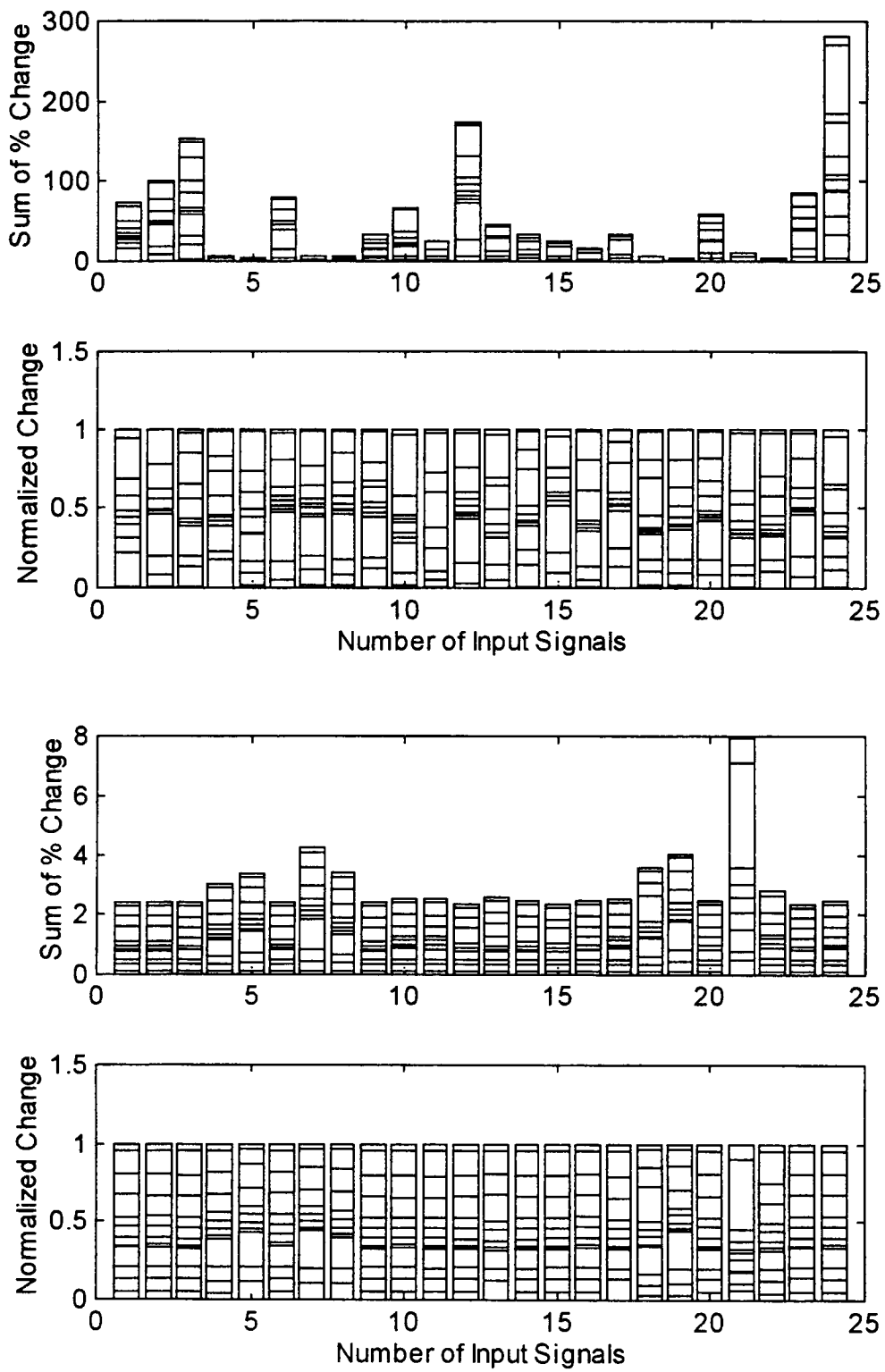


Figure 5.20 Network AB sensitivity analysis a.) non-robust training b.) robust training

system monitoring program so that the response of all 22 signals could be monitored simultaneously. For the system simulation tests, a ramp error of +0.1% per day of the instruments maximum scale deflection was artificially created in sensor 12, reactor T-hot for loop A. The test data (5 fault-free weeks) used is shown in the appendix under Figure A5.1.

Initially, a test was performed with no artificially created faults in the input data. The system correctly identified the anomalies in the actual data in sensors 6, 11, 13 and 15 and performed very well. No false alarms were recorded during the simulation. Next, a simulation was performed using the non-robust network with an artificially induced drift of 0.1% of the instruments full scale deflection (920 F) introduced into sensor 12 (reactor T-hot loop A). This resulted in a drift of approximately 1% per day (96 fifteen minute intervals per day). A residual variance of 0.016 (determined with MATLAB) with a faulted mean of 1.5 was used. The network did identify the drift in sensor 12 at about time 120 (1.25%) but alarms in other channels at slightly greater times were also noticed (channels 8, 12, 15, 17, and 19). This is in direct agreement with the sensitivity analysis results: a drift (perturbation) in the network input caused the system to have excessive output perturbations. Although a drift was detected, the channel in which the drift occurred was difficult to identify. Plots of the fault hypothesis output for all 22 signals is shown in the appendix as Figure A5.2.

Figure A5.3 in the appendix shows the SPRT fault hypothesis output for the same simulation conditions as listed above except using a network trained with a robust training set. We can see that the results are vastly improved. The network correctly identified the drifting sensor at approximately time interval 70 and continued to detect the drift throughout the test. The network correctly identified a drift at an error of 0.67% while the remaining channels in the system performed robustly. Sensor 1 recorded no false alarms during the simulation. This sensor is representative of the remainder of the sensors not shown in the figure. In addition, while the network correctly identified the drift in channel 12 throughout the simulation, it was still able to identify the multiple spikes in sensors 6, 11, 13 and 15 contained in the original data.

These results are in direct agreement with the sensitivity analysis which showed a stable network to input perturbations. In addition to testing for faults in the above sensor (12) multiple tests were performed on other sensors with similar results.

5.8.3.3 Parameter Selection Refinement Techniques

Refer to Figure 5.18 (network B sensitivity analysis). Note that inputs 2 and 20 vary the most at approximately 5%. These signals (hotwell temps A and B) are the least linearly correlated signals in the network to other signals, but are highly correlated to each other. Based on the engineers judgment, these parameters could be removed from the network, but since they only vary by a small percentage, the decision was made to leave them in. When they were removed, no improvement in network performance was noted. The sensitivity analysis for networks AA, and AB show similar results. These are shown in Figures 5.19 and 5.20 respectively.

5.8.3.4 Drift Detection

In this study, a drift error is defined as a slow rate of change in a signals expected nominal value. To test the performance of the networks, both high and low drift faults of 0.1% per day of the instruments maximum scale value was artificially created in each of the network channels. Simulations were performed to see how soon the AANN based ISCV system could detect the fault with a minimum number of false alarms. For each simulation, the time until the fault is first detected, the percent error of the drift (with respect to the full scale deflection of the signal) at the time of detection, and the number of false alarms in all the channels was recorded for both high and low drift fault scenarios. In each test case, faults were initiated at time zero. Table 5.11 summarizes the results for each of the Technical Specification safety variables specified by Florida Power Corporation. Since the focus of the study is primarily the Technical Specification parameters, for brevity only these are summarized.

Note that redundant parameters are not covered in the this section, but were covered in Section 5.7.3 of this study. Also, the four "level loop" parameters are not

Table 5.11 Drift error detection results for a drift of 0.1% per day of the instruments maximum scale deflection value.

Computer Pt. (Tag ID)	Set SPRT Faulted mean	Set SPRT Variance	% Error when detected		Detection time (days)		Number of false alarms	
			High	Low	High	Low	High	Low
A318	0.25	0.50	0.42	0.45	2.1	2.3	0	0
A319	0.20	0.06	0.21	0.24	1.8	2.5	0	0
A320	2.5	1.10	2.1	2.1	1.8	1.8	0	0
A321	0.25	1.50	0.39	0.58	2.0	3.3	0	0
A322	0.20	0.06	0.23	0.17	2.3	1.7	0	0
A323	2.5	1.10	2.83	2.33	2.4	1.9	1	0
R200	0.4	0.10	0.98	0.54	3.1	1.7	1	1
R208	7.25	4.00	2.85	2.91	1.1	1.2	0	0
R209	7.25	4.20	2.43	2.46	1.0	1.0	0	0
R210	7.25	4.20	2.57	2.60	1.0	1.0	0	0
R212	0.2	0.006	0.15	0.12	0.2	0.2	0	0
R213	0.2	0.015	0.41	0.11	0.7	0.2	0	0
R222	7.0	4.00	2.32	2.39	0.9	1.0	0	0
R223	7.0	4.00	2.43	2.43	1.0	1.0	0	0
R224	7.0	4.00	2.46	2.50	1.0	1.0	0	0
R225	7.0	4.00	2.43	2.39	1.0	1.0	0	0
R226	0.2	0.006	0.21	0.09	0.3	0.1	0	0
R228	0.2	0.006	0.17	0.13	0.3	0.2	0	0
R234	0.35	0.01	0.14	0.19	0.9	1.2	0	0
R235	0.35	0.01	0.19	0.17	1.2	1.1	1	0
R238	0.25	0.01	0.18	0.18	1.1	1.1	0	0
R239	0.25	0.01	0.19	0.19	1.2	1.2	0	0
R327	0.25	0.06	0.51	0.52	0.3	0.3	0	0
R328	0.25	0.06	0.40	0.39	0.2	0.2	0	0

included in this study. Due to their binary fashion behavior, this type of system would not be suitable for their monitoring. A much simpler technique (such as thresholding) can be used to monitor these parameters.

As the table shows, the system performed very well. The detection times generally depended on the noise level of the signal, the more noise, the longer the detection times. The average error and detection time for a low noise sensor (temperature for example) was approximately 0.25% in less than a day. For a high noise sensor, (pressure) it increases to an average of approximately 2.4% in about one day. A minimum of false alarms were recorded. Throughout the entire testing scenario listed above, only three false alarms were recorded, two of them in the same parameter (R200 -pressurizer level). This was primarily due to a sustained increase in noise in the signal for a 20 minute period in the test data.

On average, all parameters performed equally well in spite of the fact that some were highly correlated within a network (reactor temperature) and some had practically no correlation at all (pressurizer level). This can be understood by the robust training method used. Robust training (faulted input/normal output) forces each parameter in the network to rely on all the other parameters equally, not just a select few (or possibly one as shown in the sensitivity plots).

Figures 5.21 through 5.25 are plots of typical drift test cases. The top plot shows the actual drifting signal and the neural network estimate (filtered), the middle plot shows the residual between the two, and the bottom plot shows the SPRT fault hypothesis index. The plots clearly demonstrate the systems ability to detect slow drifting signals.

5.8.3.5 Gross Error Detection Techniques

Gross faults are defined in this study as drastic changes in the parameters value. A plant scenario would be a circuit that opens or shorts and a total loss of information is encountered. Gross faults are simulated by failing the sensor to its maximum or minimum full scale deflection, representing gross fault "high", or gross fault "low"

A319: Steam Generator High Level - B

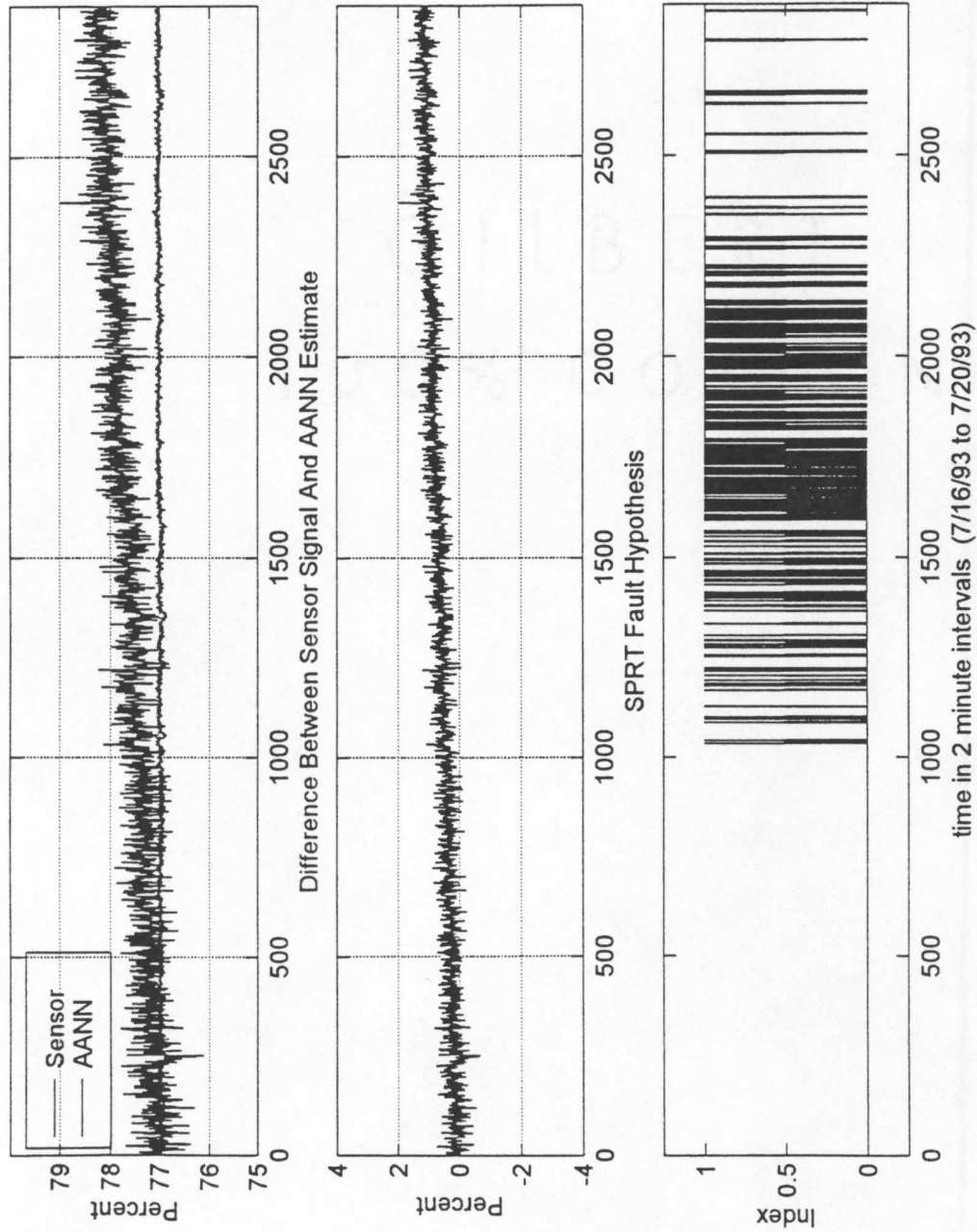


Figure 5.21 ISCV system detecting a fault drift high in the steam generator level when a drift of 0.1% per day of the instruments maximum scale deflection is introduced into the sensor signal.

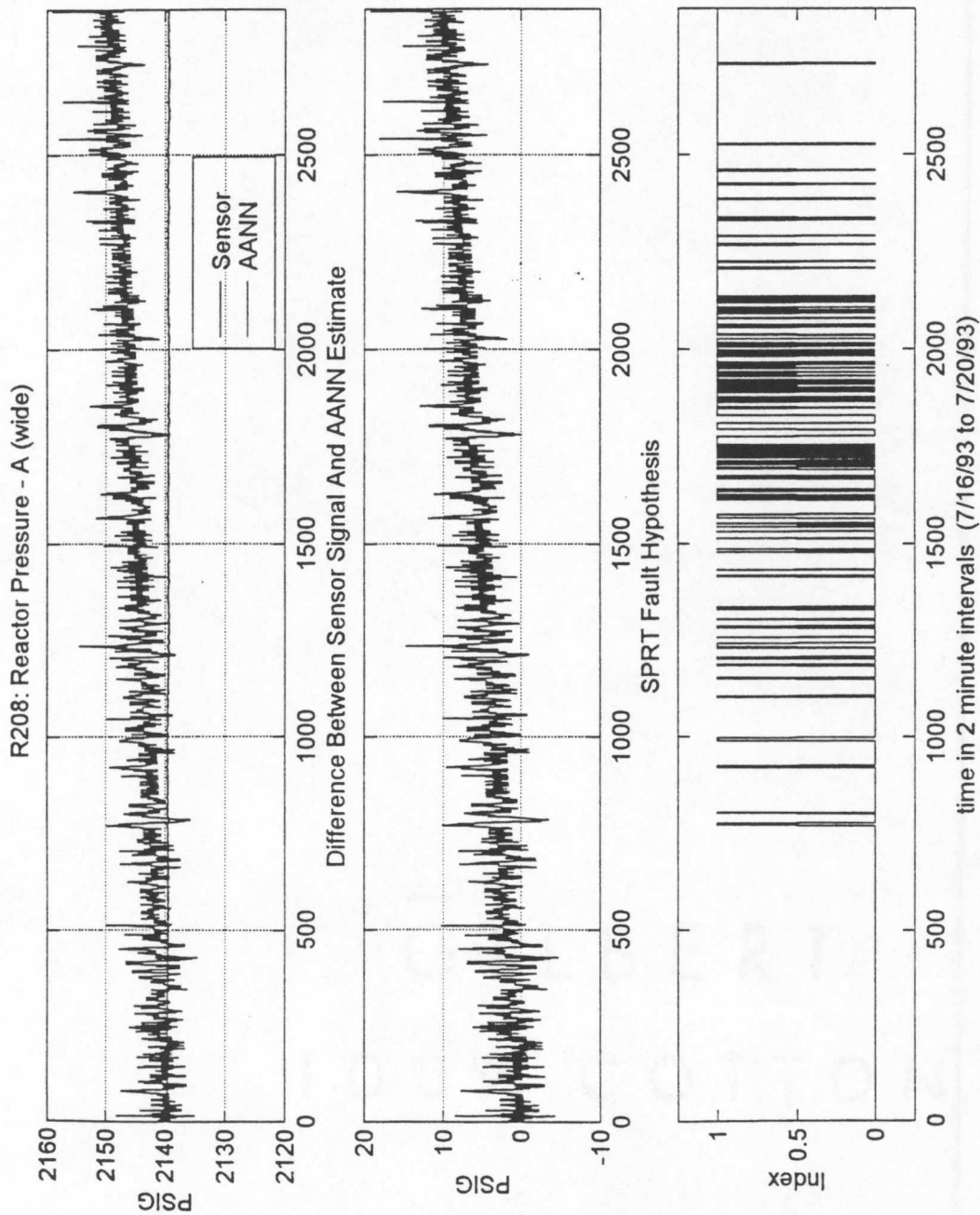


Figure 5.22 ISCV system detecting a fault drift high in the reactor pressure when a drift of 0.1% per day of the instruments maximum scale deflection is introduced into the sensor signal.

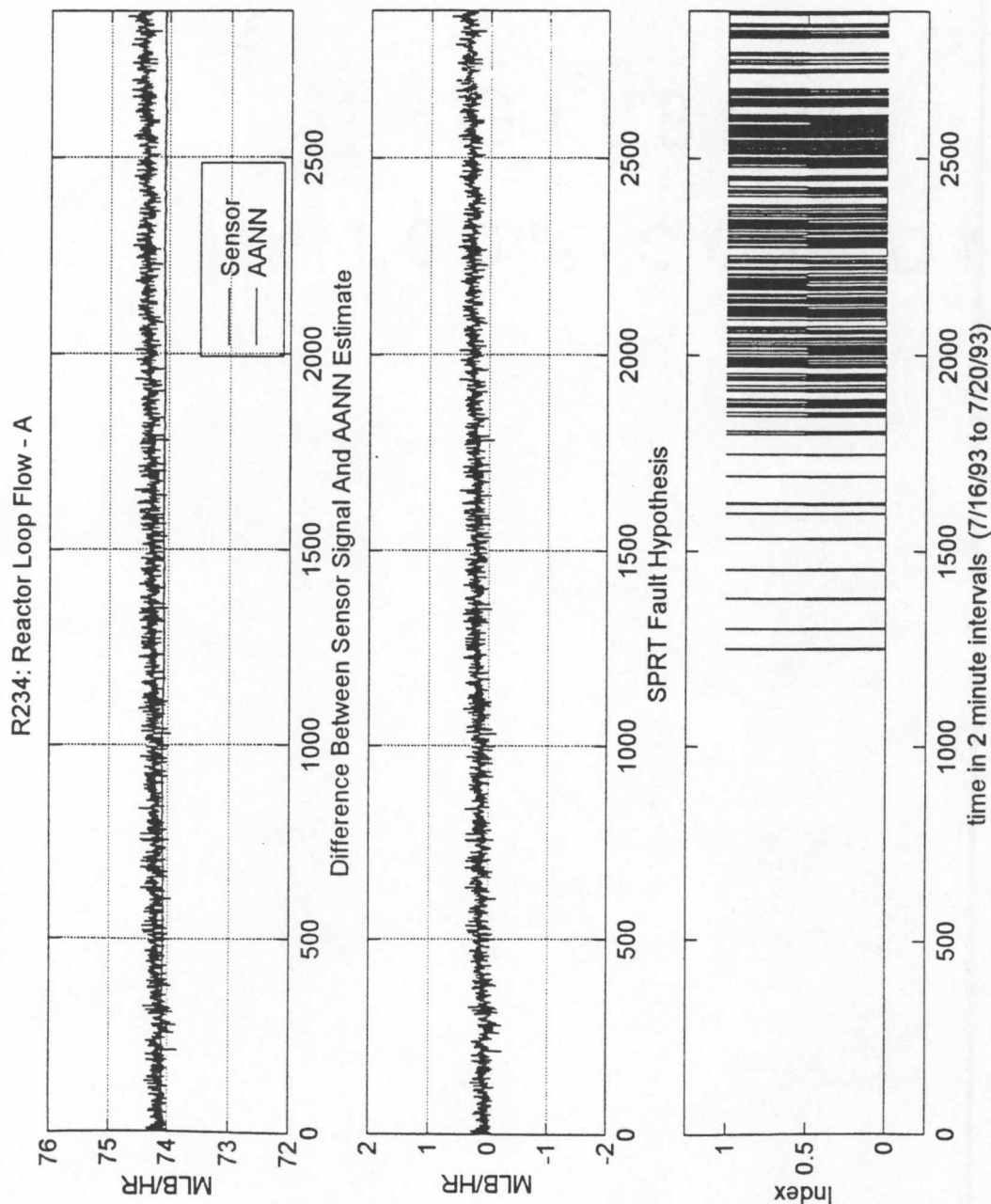


Figure 5.23 ISCV system detecting a fault drift high in the reactor loop flow when a drift of 0.1% per day of the instruments maximum scale deflection is introduced into the sensor signal.

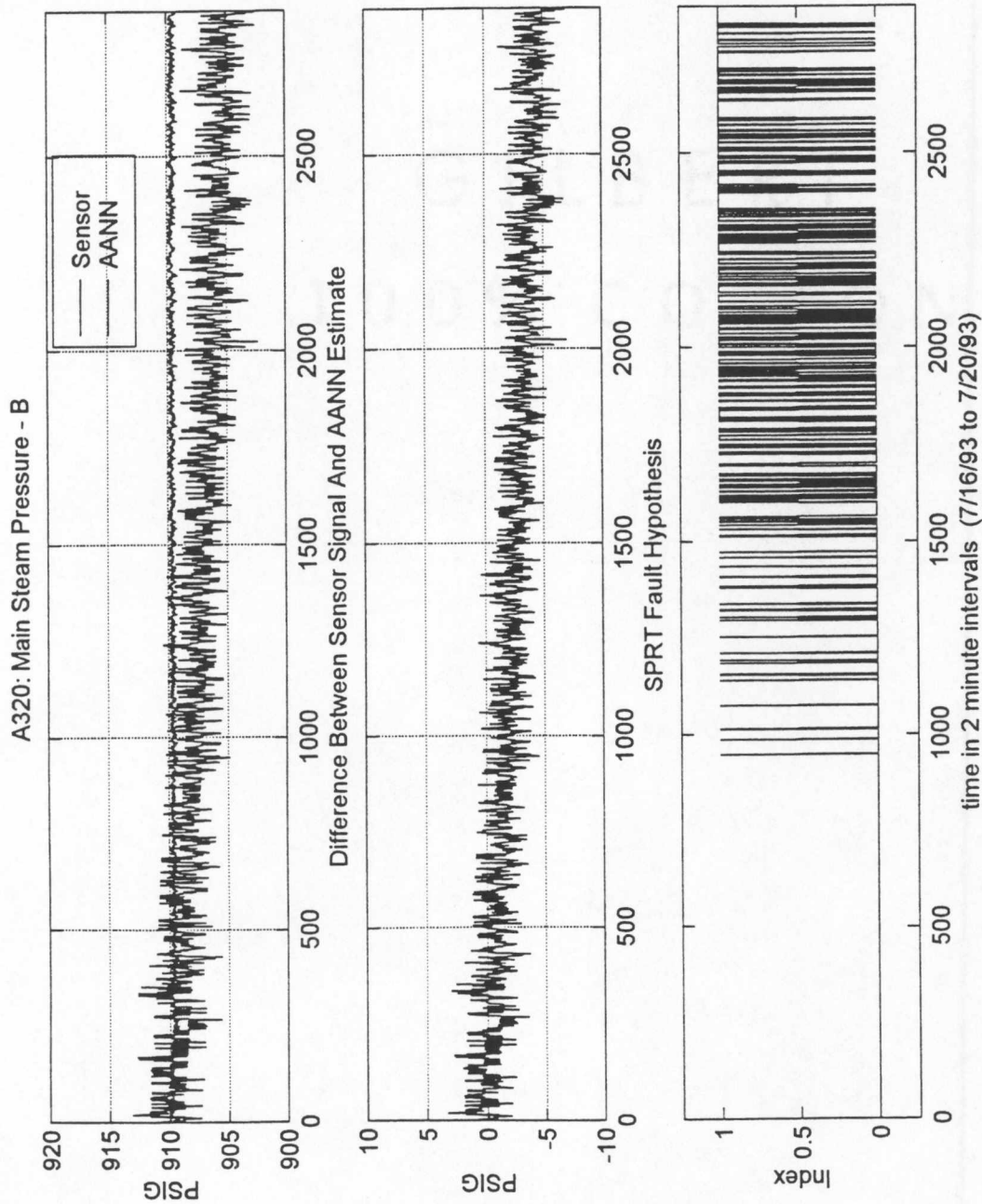


Figure 5.24 ISCV system detecting a fault drift low in the main steam pressure when a drift of 0.1% per day of the instruments maximum scale deflection is introduced into the sensor signal.

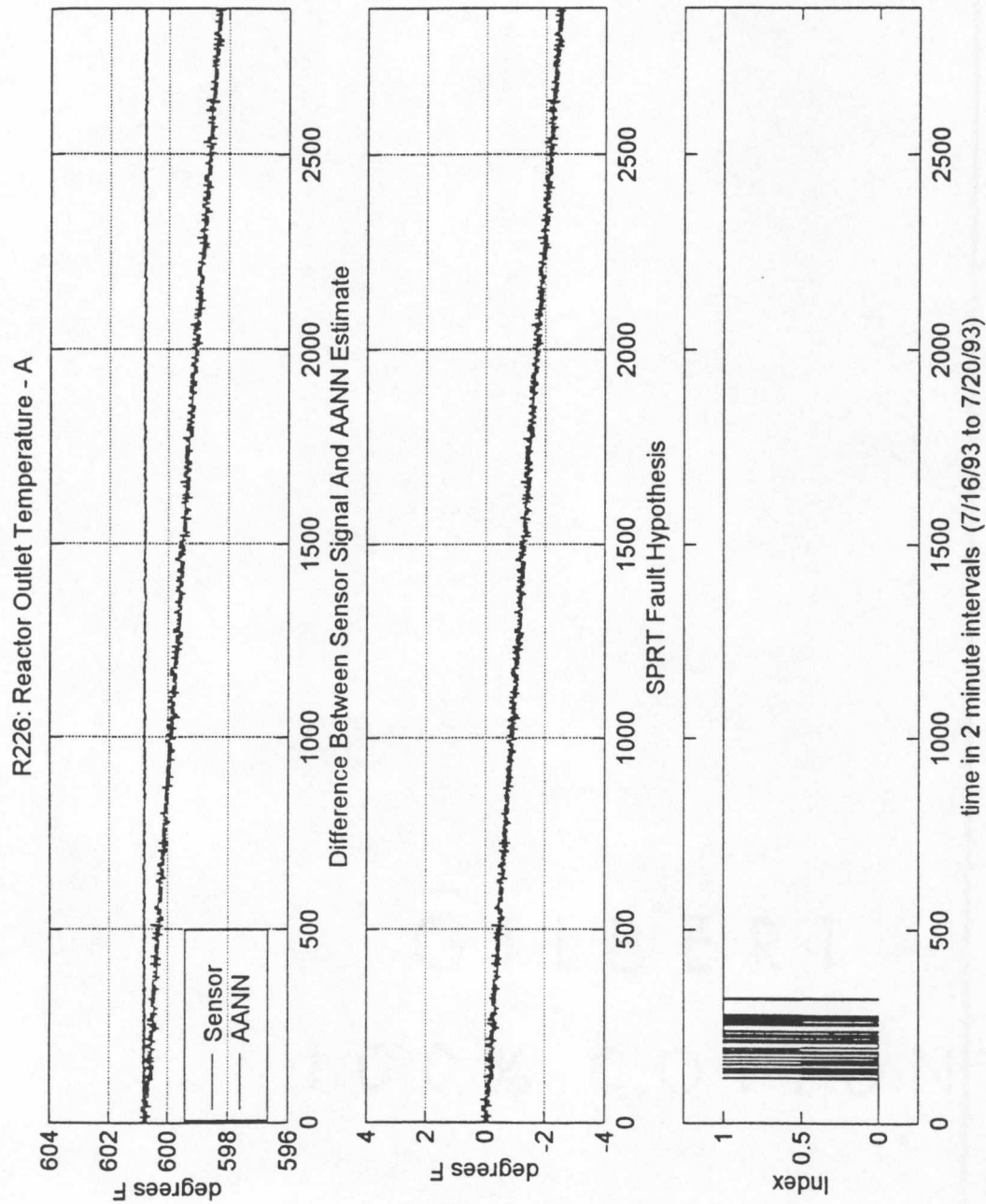


Figure 5.25 ISCV system detecting a fault drift low in the reactor outlet temperature when a drift of 0.1% per day of the instruments maximum scale deflection is introduced into the sensor signal.

respectively. Maximum and minimum scale deflection is listed for each parameter in Tables 5.2, 5.3, and 5.4 shown in section 5.2.2.

Depending on how "gross" the signal fails, the network may or may not perform as intended. A large drop (or rise) in a signal input parameter may cause several of the outputs in the network vary due to such a large loss (or gain) of information. These changes may cause several residuals to grow in magnitude. Experimental results have shown that a robust network can effectively compensate for a loss of approximately 25% of any one particular signal value. A larger fault can create false alarms in other channels. These false alarms are caused by residuals that are usually only a fraction of the amount of the residual for the signal that contains the gross fault. For example, if we simulate sensor 4 as the grossly failed sensor, then the residual for sensor 4 may be around 100, while the other variable's residuals are only around 10.

Since the drift detection SPRTs are optimized to detect very small incipient changes in the sensor, the obvious solution to reduce false alarms in other non-faulted channels is to construct a less sensitive SPRT. Figures 5.15 and 5.16 in section 5.6.1 show a parallel "gross" detection SPRT used for detection of gross sensor faults. They are identical to drift detection SPRTs except that they have faulted mean values that are considerably less constrained. They use the same variance values as the drift detection modules.

The faulted mean values for the gross SPRTs were typically set to 15 times the drift fault means. Depending on the magnitude of the gross fault, some faulted means were set at higher value for optimal performance. The greater the magnitude of the fault, the more likely the network will experience false alarms in unfaulted channels.

Like the drift test scenario, all variables were tested with the Technical Specification variables listed in this report. Each variable was grossly failed high and low (max and min scale deflection values for each variable) with the same information recorded as in the drift test. In each case, the fault was initiated at time stamp 500 (1000 minutes into the simulation). Table 5.12 below shows the test results and gross error faulted mean values for the SPRT of each Technical Specification variable.

Table 5.12 Gross error detection results.

Computer Pt. (Tag ID)	Set SPRT Faulted mean	Set SPRT Variance	Detection time (minutes)		Number of false alarms	
			High	Low	High	Low
A318	20	0.50	1000	1000	0	0
A319	15	0.06	1000	1000	0	0
A320	90	1.10	1000	1000	0	0
A321	20	1.50	1000	1000	0	0
A322	15	0.06	1000	1000	0	0
A323	90	1.10	1000	1000	0	0
R200	25	0.10	1000	1000	0	0
R208	200	4.00	1000	1000	0	1
R209	200	4.20	1000	1000	0	0
R210	200	4.20	1000	1000	0	1
R212	10	0.006	1000	1000	0	0
R213	10	0.015	1000	1000	0	0
R222	80	4.00	1000	1000	0	0
R223	80	4.00	1000	1000	0	0
R224	80	4.00	1000	1000	0	0
R225	80	4.00	1000	1000	0	0
R226	10	0.006	1000	1000	0	0
R228	10	0.006	1000	1000	0	0
R234	15	0.01	1000	1000	0	0
R235	15	0.01	1000	1000	0	0
R238	15	0.01	1000	1000	0	0
R239	15	0.01	1000	1000	0	0
R327	15	0.06	1000	1000	0	0
R328	15	0.06	1000	1000	0	0

Figure 5.26 shows an example of sensor 9 (R222 Reactor outlet pressure A) artificially failed low. At time sample 500, the sensor drops from its median value of 2130 PSIG to its minimum scale deflection of 1700 PSIG (drop of approximately 400 PSIG), with an associated change in residual mean of approximately 400 PSIG. This greatly exceeds the faulted mean value of the SPRT, thus an alarm is initiated at time sample 500 (immediately). Figure 5.27 shows the change of signal 16 (R224 Reactor outlet pressure B) due to the gross fault in sensor 9. A slight change in residual value can be seen when the fault was initiated (time sample 500), but the change was not large enough to set off the SPRT due its less stringent faulted mean setting. Figure A5.4 in the appendix shows the residuals for all 22 parameters during the gross fault condition. All residuals fluctuate around ± 10 with the exception of sensor 9, clearly indicating a large detectable level.

5.8.3.6 Noise Detection Techniques

The SPRT technique, which is described in Chapter 4, can also be used to detect sensor failures that are characterized by increases in the noise level. Wald's two-sided criterion for SPRT decisions can be applied directly. The work performed on noise detection techniques was done by Xu and is included as part of a larger report prepared for Florida Power Corporation [Uhrig, Hines, Black, Wrest, and Xu, 1996]. The following is a brief description of the methodology used to detect changes in sensor noise levels.

Unlike the SPRT module for sensor mean degradation detection, to implement a SPRT module for sensor noise degradation detection, the mean of the signal is set to be zero so that only the noise is input to the SPRT. Therefore, no neural networks are required to estimate the correct sensor output and no residual is formed. The same SPRT equations used in Chapter 4 apply here. For detection of noise degradation, the faulted and normal means are set to zero ($m_1=m_2=0$) in the degradation hypothesis H_1 .

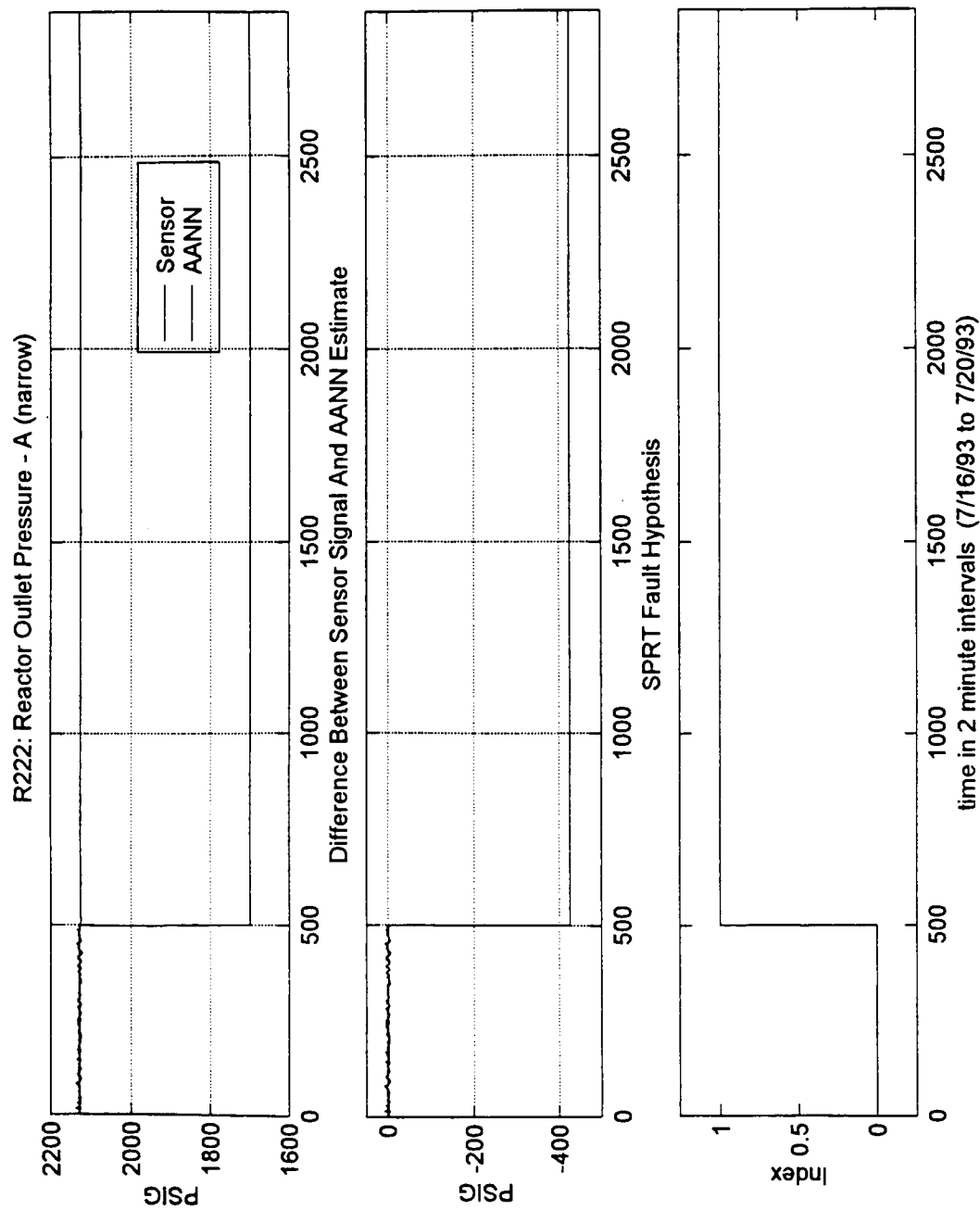


Figure 5.26 ISCV system detecting a "gross" fault low in the reactor outlet pressure when a fault equal to the instruments minimum scale deflection is introduced into the sensor signal at time stamp 500.

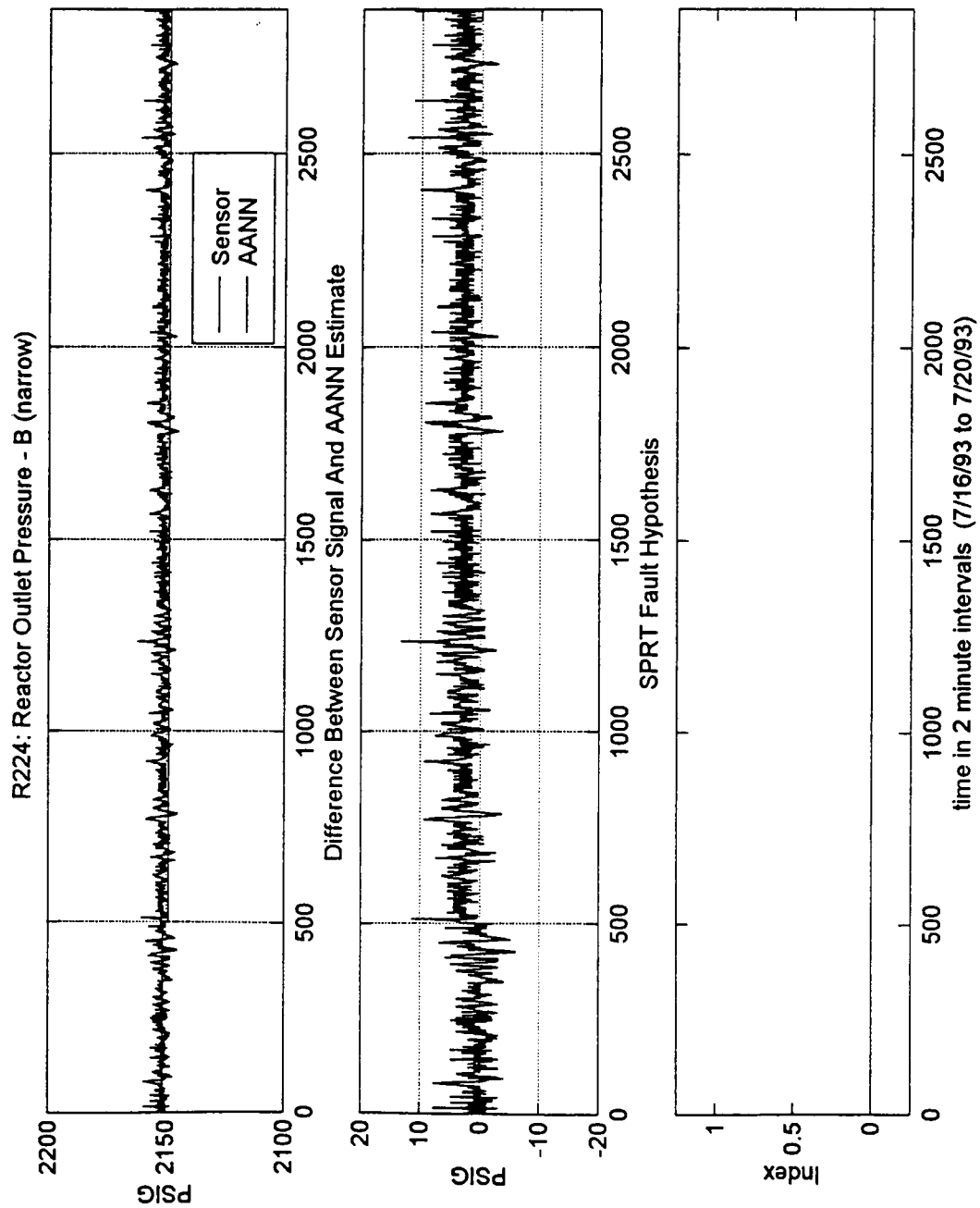


Figure 5.27 Response of "gross" fault initiated at time stamp 500 for a typical companion network variable.

The expression for the logarithm of the likelihood ratio for noise degradation is:

$$\lambda_m = \lambda_{m-1} + \ln \left[\frac{p_1(s_m|H_1)}{p_0(s_m|H_0)} \right] = \lambda_{m-1} + \frac{1}{2} \left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2} \right) x_m^2 + \ln \left(\frac{\sigma_0}{\sigma_1} \right). \quad (5.11)$$

Where: σ_0 -- the normal variance under the normal hypothesis H_0 , which is obtained from the nuclear normal condition (no fault) test data;

σ_1 -- the degradation variance under the degradation hypothesis H_1 , which will be sensor specify and determined by the calibration specifications.

x_m -- the sensor measurement at time $t = m$.

Wald's two-sided criterion, which was described in Chapter 4, is used with no modification to determine the SPRT threshold and is dependent only on the false alarm and missed alarm probabilities.

It was stated that since the mean value of the signal is known to be zero, no estimate of the signal is required. This implies that redundancy is not necessary for the detection of excessive noise levels in the signals. Redundancy is necessary to detect whether the increase in the noise is due to the sensor or the process. Increase in noise levels in a particular sensor and not in others can be detected and indicates that the excessive noise is generated in the sensor instead of the process. If excessive noise levels exist in all sensors, the noise is likely to be generated in the process.

The noise level detection system is used to detect changes in the noise levels of the sensor measurements. The sensor measurements are signals which include a mean value component and a fluctuating component (noise). These two components need to be separated to allow the detection of noise changes using the SPRT. To separate the noise from the mean value of the signal, two conventional 3rd order Chebychev high pass filters cascaded in series were used. The resulting signal has a mean of zero and a noise level equal to the original signal and can be applied to the SPRT for sensor noise

degradation detection. A block schematic of the noise SPRT detection system is illustrated in Figure 5.28.

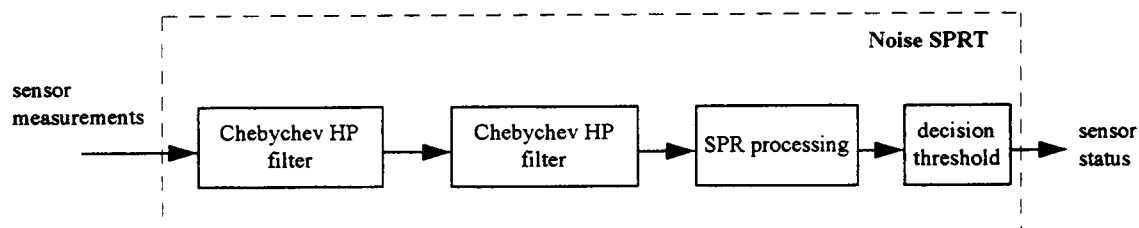


Figure 5.28 Block schematic of a noise SPRT detection system.

The results show that the system can:

- detect an increases in noise level
- large spikes existing in the signals corresponding to abrupt changes can be picked up immediately by the system

Noise and bias detection SPRT systems are implemented in parallel mode. Results given from both systems are combined in order to make a logic decision as to whether the sensor is failed in bias mode, in noise mode, or in both modes. The possible sensor failure instances and their logical decisions are presented in Table 5.13.

Table 5.13 Bias and noise detection logic table.

Sensor Instance	Bias SPRT		Noise SPRT		Logical Decision	Sensor Status
	good	drift	good	drift		
#1	x		x		good	good
#2	x			x	noise drift	bad
#3		x	x		bias drift	bad
#4		x		x	noise or/and bias drift(s)	bad

5.8.4 Overall System Performance

System performance was very good as shown above. Previously shown work used approximate faulted mean values that were not optimal for early fault detection. For a typical temperature sensor a drift of about 0.6% of the maximum full scale deflection could be detected. The faulted mean values used in this report were set by a combination of examining the residuals of the signals under non-fault conditions and by repeated testing using data that included several transients. In essence, repeated test and re-test trials. While this procedure yielded good results, they might not be optimal due to the exhaustive iterative procedure. Increasing the sensitivity of the SPRTs requires setting the faulted mean so that the faults could be detected as soon as possible without any false alarms. Recall that the input to the SPRT is a residual calculated as the difference between the actual signal value and the neural networks estimate, e.g. a signal with a mean of zero and a fixed variance. When a drift error in a sensor has occurred, the mean of the residual deviates from zero at a proportional drift rate. The distance the mean is allowed to drift away from zero until the SPRT alarms is set primarily by the faulted mean. An SPRT alarm means that there is a higher probability that the residual mean is equal to the pre-specified faulted mean value rather than the nominal mean of zero. To further optimize detection times of the system, the SPRT faulted mean value for each variable being monitored could be determined by an optimization algorithm which determines the faulted means based on a set of desired performance criteria. An on-line optimization algorithm could be easily created which could deliver more optimal system performance more efficiently.

It should be pointed out that these results were obtained with a network trained over a very broad range that includes several transients. How well the network (or any system estimation model) learns these transients is the limiting factor on how tight the sensitivity of the SPRTs can be set (through the faulted means). If the network had been trained on purely steady state data, the sensitivity of the SPRT could be set much tighter which would result in even earlier detection times. However, to be useful, the

model has to be trained to cover the entire mode of plant operation, not just steady state conditions.

As discussed previously, network B parameters are much less correlated than network AA or AB (average of 0.28 to 0.65), however no significant differences in performance are observed. This would suggest that a high correlation between the input parameters is not a strict requirement for good network performance. This behavior is due to the robust training method used in this study. Robust training means that the network is trained on patterns that represent the input/output relationship during a fault condition. Because of this, a robustly trained network has a pseudo *a priori* fault knowledge base to draw from, thus the network's equal reliance on several parameters to estimate the correct output state is greatly increased. The sensitivity analysis results support this conclusion. A network trained on a non-robust training set was shown to be sensitive to small input perturbations. The network is relying on a select few parameters (perhaps highly correlated) to help cover the loss of the faulted input signal. However, a network trained with a robust training set is much less sensitive to input perturbations due to its *a priori* input/output training domain forcing the network to focus on all parameters in the network equally, instead of a select few. Because of the robust training technique, high linear correlation between the parameters is not a strict requirement for optimal network performance.

Chapter 6

Conclusions and Recommendations for Future Work

6.1 Conclusions

A system to continuously monitor sensor calibrations during plant operation was developed using autoassociative neural networks. The system proved to be quite sensitive to sensor drift with detection setpoints on the order of 0.1% to 1.5% of the full scale range of the instrument.

The results of this study have shown that a plant wide sensor calibration monitoring system using autoassociative neural networks is not only feasible but very practical. A methodology has been described to implement such a system. This includes parameter selection utilizing linear correlation coefficients and sensitivity analysis, a feature detection network architecture, and a fast reliable training method to create a robust network. To detect the faults, a drift and gross fault detection system has been implemented using the sequential probability ratio test (SPRT). The complete ISCV system has been integrated using the MathWork's SIMULINK software.

Engineering judgment and linear correlation coefficients were used to initially select the network parameter groupings. Sensitivity analysis results were used as a verification tool to determine if the initial network parameter groups were satisfactory. Refinements were made to the initial selections if needed. Overall, due to the robust training method, high linear correlation between network parameters was found not to be a strict requirement for optimal system performance.

Two methods for determining network training error were formulated and found to be in reasonable agreement. They were a statistical filtering technique, and cross validation training. The statistical technique was a much quicker and automated way of

determining the network training error. Cross validation training could be automated as well, with minor changes to the code, but it would slow down the training process.

The three hidden layer "feature detection" autoassociative neural network trained on a robust training set was shown to have excellent generalization abilities making it ideal for a plant wide sensor monitoring system such as the one implemented in this study.

The SPRT detection module proved to be an excellent tool for detecting incipient drift faults as well as gross faults. Results show that the ISCV system using autoassociative neural networks could clearly detect a fault or drift in a single channel without affecting the other channels being monitored. Thus the network not only detects the fault, but isolates the channel in which the fault has occurred.

6.2 Recommendations for Future Work

The optimization of the ISCV system described in this study is not complete. There are several enhancements to the system that may prove extremely beneficial. These enhancements center around the notions of robustness and extended fault detection and isolation capabilities. Add-on systems that not only detect sensor faults but also detect and isolate plant process faults would also be valuable. The following sections briefly present these enhancements as opportunities for future research.

6.2.1 SPRT Optimization Algorithm

Increasing the sensitivity of the SPRTs entailed setting the faulted mean so that the faults could be detected as soon as possible without any false alarms. In this study, the optimal faulted mean values were set by a combination of examining the residuals of the signals under non-fault conditions and by repeated testing using data that included several transients. An easier and more optimal approach would be to develop an on-line optimization algorithm that could quickly and efficiently find the optimum SPRT fault values for each parameter. The result would be a system that could detect incipient faults sooner with a minimum of false alarms.

6.2.2 Residual Whitening

The SPRT has proven to be an excellent tool for signal validation and equipment operability surveillance applications. However, its domain of applications is typically limited to applications in which the input signal is Gaussian and contains uncorrelated white noise. Nongaussian variables typically produce higher than desired rates of false alarms and missed alarms in the SPRT based system. Typically, the residuals formed from the AANN estimate process are contaminated by serial correlation's (correlated residual values separated by a time lag, a non zero autocorrelation over the range of time lags of interest) and are not as "white" as desirable. To rectify this problem, either the process residuals could be "whitened" through a spectrum transform or filtering algorithm [Gross and Hoyer, 1995] so that they are normally distributed about the mean, or the SPRT module could be modified to be optimal under other conditions. The result would allow tighter tolerances on the SPRTs with improved detection abilities and fewer false alarms.

6.2.3 Hidden Layer Node Search

As discussed in Section 5.3.2.1, a guideline was given to determine the number of mapping and demapping nodes in the hidden layers. More study should be performed to determine the optimal number of mapping nodes needed for a particular process. Determining the number of bottleneck nodes (remember they represent the number of nonlinear principle components needed to represent the process) is much simpler. A reference was given [Hastie and Stuetzle, 1989] to implement an algorithm which determines the number of nodes based on NLPCA curves. The entire network architecture could be completely determined based on a sample data set of the process to be modeled.

The mapping and demapping layer sizes could be optimized by using a SVD estimation of the noise in the signal and limiting the capacity of the nodes in these layers to model the functional relationship and not the noise. Preliminary research is currently underway.

6.2.4 Optimal SVD Noise Elimination Parameter Value

Recall that the linear output weight values were determined using a singular valued decomposition (SVD) regression technique. In this method only the most relevant information is retained to compute the weights. The least important information is discarded because it is most likely arising from noise. In this study, the amount of noise that was removed from the solution to the system is determined relative to the largest output weight value desired in the network (experimental results show that keeping the weights within ± 10 provides excellent network generalization with no loss of important information). However, a more optimal technique of determining the cutoff value should be found. This could be done by studying the loss of signal information (or signal power) relative to the cutoff parameter value. This technique would parallel the work discussed for finding the number of hidden nodes in the network.

6.2.5 Robust Training Samples

The robust training technique has shown to be a very powerful tool for creating a robust network that is insensitive to input perturbations. Several methods exist to implement this technique. The question arises as to how many corrupted training samples are required for an optimum robust training set. One method is to make the number of robust training patterns equal to the number of original training patterns. Thus if there were n original training samples, there will be $2n$ robust training samples. Further study should be done to determine the optimum number of robust training patterns for each sensor. Of course, the trade off for an increased number of samples is longer training times. Both of these factors should be taken into consideration.

6.2.6 Missing Sensor Replacement

In an on-line situation, a fault may occur in a sensor. The AANN ISCV system will notify the operator that the fault has occurred. But due to operational procedures, the fault may not be corrected immediately. It would be desirable for the system to output the correct value of the failed sensor so that the plant could continue operating without

interruption. This is especially desirable when the faulty signal is a controlled variable in a feedback control system. In addition, this would minimize degradation of the ISCV system and may preclude the need for parallel gross sensor failure networks.

A method to achieve this is through a missing sensor replacement algorithm. For a missing sensor in the operational data space, the most likely value to output for this sensor is the value that minimizes the squared difference between the input data space and the corrected data space [Dong and McAvoy, 1994]. Thus, using the other parameters in the system, the algorithm could calculate the corrected output for the sensor which would allow for continued plant operation. A simpler method would be to use the AANN estimate of the process variable once a fault has been isolated in that sensor.

6.2.7 Decision Logic System

A plant wide monitoring system typically monitors a great number of plant variables. A single signal may fail, or an entire group of signals may fail due to a common mode type of failure (drifting bias on multiplexer). Because of this, the plant operator may have trouble interpreting the root cause of a multiple signal failure. An intelligent system implementation (Expert system or Fuzzy Logic system) could be implemented to notify the operator of the exact cause of a multi channel failure.

6.2.7.1 Fault Detection and Isolation

An additional module could be added to the ISCV system to detect and isolate non-sensor faults. A sensor fault is characterized by having only one large residual and by using a missing sensor replacement algorithm, described in section 6.2.6, to artificially reduce that residual to near zero. If this cannot be accomplished, the fault is not due to the sensor, and the plant has changed from its modeled condition.

The information contained in the residual signatures can be used to isolate the cause of the fault. Methods of residual structuring for fault isolation have been investigated for several years [Gertler, 1988], [Frank, 1990, 1993]. Usually, these

methods use linear models and involve complex mathematical techniques that are usually limited to linear systems theory. The use of neural networks, which are non-linear mapping devices, for the plant model allows a more robust model of the system. The use of neural networks for fault isolation allows for the detection and isolation of nonlinearities in the residual signatures. Figure 6.1 illustrates a block diagram of the fault detection and isolation (FDI) system where s_i are the sensor signals, s_i' are the sensor signal estimates, r_i are the residuals, and the output is a fault hypothesis.

Each system in the process will have its own dedicated FDI system. This allows fault decoupling between interacting systems allowing proper fault isolation. The conventional techniques of using artificial neural networks for FDI [Hines, 1994], [Sorsa, 1993], [Bartlett and Uhrig, 1992] use a single neural network architecture structure that does not allow for decoupling faults in interacting systems or missing sensor replacement. They also have not been verified by real time implementation on a process system. The proposed architecture incorporates the additional functionality, can be added to the ISCV system, and could be implemented on-line.

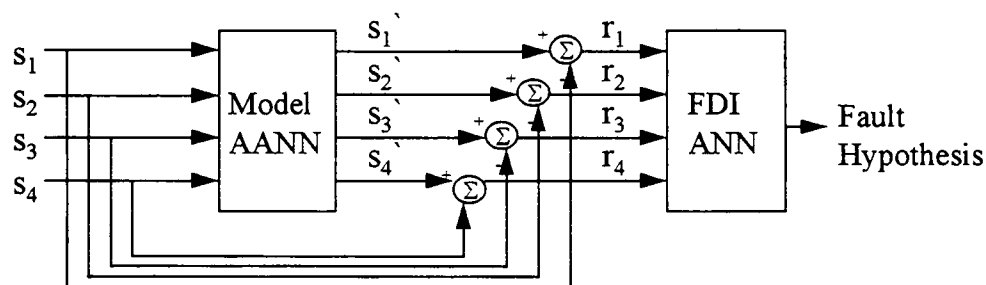


Figure 6.1 FDI system

6.2.8 Model Changes and Adaptive Network Tuning

Neural networks are not guaranteed to function as expected when operating outside of the training space. Although they have good generalization abilities inside the training space, they must be retrained when expected to operate in new regions. There are several circumstances when the operating region of a nuclear power plant may change:

1. Component replacement may change the plant operating characteristics (model).
2. The plant may be operating in an untrained-upon operating or unexpected configuration. This could be purposeful or due to operator error.
3. The plant model may change due to component degradation.
4. Cyclic or seasonal changes may effect the state of the plant.
5. Plant component failures.

All of these conditions may affect the ISCV system. Since the ANN was trained to learn a specific set of sensor correlations, if plant conditions change slightly, the system must be able to adapt to the new state. This adaptation is accomplished by retraining or tuning the neural network.

The model changes discussed above cause the actual model to be either slightly or significantly different than that embodied in the neural network. This causes several residuals to be significantly non-zero. When only one residual is non-zero, a hypothesis is formed that a sensor has failed. When several residuals are non-zero, either a non-sensor fault has occurred, or the plant model has changed from that modeled. In either case, the operator will be made aware that one of these two conditions has occurred and he must use his engineering judgment to either detect and isolate the fault or to decide to retune the model to operate in the new conditions. Automated methods of fault detection and isolation (FDI) could also be employed as discussed in section 6.2.7.1.

When the plant model changes significantly enough to effect the neural network residuals, the neural network must be retrained to operate in the new plant condition.

When this retraining or tuning is performed on-line it is called adaptation. Real time adaptive algorithms must make a trade-off between learning the new conditions and retaining the information already learned. They must also be computationally efficient in order to learn the new plant characteristics in as few time steps as possible. Since the process models may change over time during or between fuel cycles, an adaptive scheme to train the networks on-line may be necessary. Future research should include data sets over several fuel cycles so that a tuning algorithm such as History Stack Adaptation (HSA) can be developed to continuously update the neural network model to changing plant conditions [Mills, Zomaya, and Tade, 1996].

One very simple and quick method of retraining the network would be to only train the linear output neurons of the network, not the entire network [Lo, 1996]. The functional relationship between the variables is already stored in the weights of the feature detection network layers. Thus, for minor plant changes, a simple calculation of the new linear output weight values using the Singular Valued Decomposition Technique (SVD) would be all that is required to model adaptive plant stages.

Bibliography

Bibliography

Bartlett, E.B. and Uhrig, R.E., (1992), "Nuclear Plant Diagnostics Using an Artificial Neural Network", *Nuclear Technology*, Vol. 97, 272-281.

Bishop, C.M., (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, New York.

Black, C.L., "Inferential Neural Networks for Nuclear Power Plant Sensor Channel Drift Monitoring," *ANS Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies*, Pennsylvania State University, May 6-9, 1996

Chien, T.T., and Adams, M.B., (1976), "A Sequential Failure Detection Technique and Its Applications," *IEEE Transactions on Automatic Control*, Vol. AC-21, No. 5, pp. 750-757

Cybenko, G., (1989), "Approximation by Superposition of a Sigmoidal Function," *Math. Control Signals and Systems*, Vol. 2, pp. 303-314.

Dong, D., McAvoy, T., (1994), "Sensor Data Analysis Using Autoassociative Neural Networks", *Proceedings of the World Congress on Neural Networks*, San Diego, Ca, Vol. I, 161-166.

Dong, D., and McAvoy, T., (1996), "Nonlinear Principle Component Analysis - Based on Principle Curves and Neural Networks," *Computers in Chemical Engineering*, Vol. 20, No.1, pp. 65-78.

Erbay, A.S., Upadhyaya, B.R., (1995), "An On-Line Signal Validation System For Nuclear Power Plants", *9th Power Plant Dynamics, Control & Testing Symposium Proceedings*, Knoxville, Tennessee, Vol. 1, pp. 16.01-16.16.

Eryurek, E., Upadhyaya, B.R., Uhrig, R.E. (1991), "Development and Application of Multi-Layer Neural Networks for Estimation of Power Plant Variables", DOE Report DOE/NE/88ER12824-01.

Fan, J.Y., Nikolaou, White, R.E., (1993) "An Approach to Fault Diagnosis of Chemical Processes via Neural Networks," *AIChE Journal*, Vol. 39, No. 1.

- Fletcher, R., and Reeves, C.M., (1964), "Function Minimization by Conjugate Gradients", *Computer Journal*, 7, 144-154.
- Frank, P.M., (1990), "Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-based Redundancy-A Survey and Some New Results", *Automatica*, Vol. 26, No. 3, 459-474.
- Frank, P.M., (1993), "Advances in Observer-Based Fault Diagnosis", *Proc. International Conference on Fault Diagnosis*, France, 817-836, 1993.
- Funahashi, K., (1989), "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, 2, pp. 183-192.
- Gertler, J., (1988), "Survey of Model Based Failure Detection and Isolation in Complex Plants," *IEEE Control Systems Magazine*, 3-11.
- Goldfeld, S.M., Quandt, R.E., and Trotter, H.F., (1966), "Maximization by Quadratic Hill Climbing", *Econometrica*, 34, pp 541-551
- Gross, K., and Hoyer, K., (1995), "Spectrum-Transformed Sequential Testing Method for Signal Validation Applications," *9th Power Plant Dynamics, Control & Testing Symposium Proceedings*, Knoxville, Tennessee, Vol. 1, pp. 36.01-36.15.
- Hastie, T., and Stuetzle, W., (1989), "Principle Curves," *Journal of the American Statistical Association*, 84:(406), pp. 50 - 516.
- Haykin, S., (1994), *Neural Networks A Comprehensive Foundation*, Macmillan Publishing Company, Englewood Cliffs, NJ.
- Hines, J.W., (1994), "A Hybrid Approach to Fault Detection and Isolation Merging Analytical Redundancy and Neural Network Techniques", *Ph.D. Dissertation*, Ohio State University, Department of Nuclear Engineering.
- Holbert, K., Heger, A., and Ishaque, A., (1995), "Fuzzy Logic For Power Plant Signal Validation," *9th Power Plant Dynamics, Control & Testing Symposium Proceedings*, Knoxville, Tennessee, Vol. 1, pp. 20.01-20.15.
- Hornick, K., Stinchcombe, M., and White, H., (1989), "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, 2, pp. 359-366.
- Howard Demuth and Mark Beale, (1994), *Neural Networks Toolbox*, The MathWorks Inc., Natick, Mass.

Kramer, M. A., (1992), "Autoassociative Neural Networks", *Computers in Chemical Engineering*, 16:(4), pp. 313-328.

Kramer, M. A., (1991), "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks," *AIChE Journal*, Vol. 37, No. 2, pp. 233-243.

Levenberg, K., (1944), "A Method for the Solution of Certain Non-linear Problems in Least Squares", *Quarterly Journal of Applied Mathematics*, 2, 164-168.

Lippman, R.P., (April, 1987), " An Introduction to Computing With Neural Nets," *IEEE ASSP Magazine*, 4(2), 4.

Lo, J., (1996), "Adaptive System Identification by Nonadaptively Trained Neural Networks," *International Conference on Neural Networks*, Vol. 4, pp. 2066-2071, Washington, D.C.,

Marquardt, D.W., (1963), "An Algorithm for Least Squares Estimation of Non-linear Parameters", *J. SIAM*, 11, pp. 431-441.

Masters, T., (1993), *Practical Neural Network Recipes in C++*, Academic Press, San Diego, California.

MATLAB, *High Performance Numeric Computation and Visualization Software*, (1993) The Math Works, Natick Massachusetts.

Mills, P., Zomaya, A., and Tade, M., (1996), *Neuro-Adaptive Process Control*, John Wiley & Sons, West Sussex, England.

Moller, M.F., (1993), "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks*, 5, pp. 525-533.

Nabeshima, K., Susuki, K., and Turkan, T., (1995), "Real-Time Nuclear Power Plant Monitoring With Hybrid Artificial Intelligence Systems, "9th POWER PLANT DYNAMICS, CONTROL & TESTING SYMPOSIUM, Vol. 2, pp. 51.01, Univ. of Tennessee-Knoxville, May 24-26.

Olvera, J.R. (1993), "Instrument Calibration Verification," *Masters Thesis*, The University of Tennessee, Department of Nuclear Engineering.

Ragheb, M.T., and Campos, T., (1991), "Nuclear Power Plant Transients Pattern Recognition Using Neural Network Methodology", *Transactions of the American Nuclear Society*, Vol. 62, 110-111.

Rumelhart, D.E., and McClelland, J.L., (1986), *Parallel Distributed Processing*, MIT Press, Cambridge, Mass.

Rutherford, B., (1996), Statistical Methodology for the ISCV Project, *Presentation to ISCV Workshop*, Crystal River Florida.

Sequin C. H., and Clay R.D., (June, 1992), "Fault Tolerance In Artificial Neural Networks," *Proc. Int. Joint Conf. on Neural Networks*, pp. I-703-708, San Diego, CA

SIMULINK Dynamic System Simulation Software, (1993), The Math Works, Natick Massachusetts.

Soras, T., and Koivo, H., (1993), "Application of Artificial Neural Networks in Process Fault Diagnosis", *Automatica*, Vol. 29, No. 4, 843-849.

Tsoukalas, L.H., Uhrig, R.E., (1996), *Fuzzy and Neural Approaches in Engineering*, Copyright by L.H. Tsoukalas and R.E. Uhrig.

Turkan, E., Ciftcioglu, O., Upadhyaya, B. R., (1992), "On-Line Plant-Wide Monitoring Using Neural Networks," *8th POWER PLANT DYNAMICS, CONTROL & TESTING SYMPOSIUM*, Vol. 2, pp. 58.01, University of Tennessee-Knoxville, May 27-29.

Uhrig, R.E., (1989), "Enhancing Nuclear Power Plant Performance Through The Use of Artificial Intelligence," *Report Prepared by the University of Tennessee for The U.S. Department of Energy, Contract No. DE-FG-07-88ER12824*.

Uhrig, R.E., (July, 1994), "Artificial Neural Networks in Nuclear Power Plants," *Nuclear News*, Vol. 37, No. 9.

Uhrig, R.E., Hines, J.W., Black, C., Wrest, D.J., Xu, X., (1996), "Instrument Surveillance and Calibration Verification System," *Report Prepared by the University of Tennessee for Sandia National Laboratories, Contract No. AQ-6982*.

Upadhyaya, B.R., (September, 1988), "Development and Testing of an Integrated Signal Validation System for Nuclear Power Plants," *Report prepared by the University of Tennessee for the U.S. Department of Energy*, vol. DOE/NE/37959-24.

Upadhyaya, B.R., (August, 1988), "An Integrated Architecture for Signal Validation In Power Plants," *IEEE International Symposium on Intelligent Control*.

Upadhyaya, B.R., Holbert, K. E., Eryurek, E., (1989), "Automated Generation of Non-Linear System Characteristics for Sensor Failure Detection," *35th International Instrumentation Symposium*.

Upadhyaya, B.R., Eryurek, E., Mathai, G., (1989), "Application of Neural Computing Paradigms for Signal Validation," *Seventh Power Plant Dynamics, Control and Testing Symposium*.

Upadhyaya, B. R., Wolvaardt, F. P., Glockler, O., (1987), "An Integrated Approach for Sensor Failure Detection in Dynamic Systems", *Research Report prepared for the Measurement & Control Engineering Center*, Report No. NE-MCEC-BRU-87-01.

Upadhyaya, B. R., and Eryurek E., (1992), "Application of Neural Networks for Sensor Validation and Plant Monitoring," *NUCLEAR TECHNOLOGY*, Vol. 97, pp. 170-176.

Wald, A., (1945), "Sequential Tests of Statistical Hypothesis", *Ann. Math. Statist.*, Vol. 16, 117-186.

Wang, X., and Holbert, K., (1995), "A Neural Network Realization of Linear Least-Square Estimate For Sensor Validation," *9th Power Plant Dynamics, Control & Testing Symposium Proceedings*, Knoxville, Tennessee, Vol. 1, pp. 15.01-15.15.

Appendices

Appendix A
Selected Tables and Figures

Table A-1 Network AA correlation coefficient matrix (continued).

Network AA Comp Pt.	R238	T229	S226	S305	S241	A321	A322	A323	A210	E202	S225	R215
R214	-0.52	0.91	-0.96	-0.93	-0.96	-0.91	-0.96	-0.18	-0.96	-0.96	0.51	0.97
R327	0.43	-0.71	0.89	0.84	0.89	0.87	0.89	-0.06	0.84	0.82	-0.49	-0.83
S224	-0.16	0.21	-0.33	-0.28	-0.34	-0.37	-0.35	0.29	-0.27	-0.32	0.93	0.35
P208	0.50	-0.87	0.95	0.91	0.95	0.90	0.94	0.19	0.95	0.94	-0.44	-0.92
S301	0.51	-0.89	0.99	0.95	0.99	0.95	0.99	0.14	0.98	0.97	-0.52	-0.96
S288	0.31	-0.67	0.44	0.45	0.43	0.34	0.41	0.87	0.60	0.51	0.01	-0.43
S285	0.50	-0.86	0.99	0.95	0.99	0.96	0.99	0.06	0.96	0.96	-0.53	-0.96
S284	0.51	-0.87	0.99	0.95	0.99	0.96	0.99	0.09	0.97	0.97	-0.52	-0.96
S290	0.52	-0.89	0.99	0.95	0.99	0.95	0.99	0.14	0.98	0.98	-0.51	-0.96
T230	0.04	-0.22	-0.09	-0.05	-0.10	-0.17	-0.12	0.90	0.10	0.00	0.29	0.08
W352	0.22	-0.52	0.26	0.28	0.25	0.16	0.23	0.91	0.43	0.34	0.14	-0.26
R226	0.48	-0.79	0.97	0.92	0.97	0.94	0.97	0.01	0.93	0.93	-0.52	-0.90
R238	1.00	-0.50	0.51	0.48	0.51	0.48	0.50	0.14	0.52	0.52	-0.27	-0.50
T229	-0.50	1.00	-0.88	-0.85	-0.88	-0.81	-0.87	-0.40	-0.93	-0.89	0.43	0.90
S226	0.51	-0.88	1.00	0.95	0.99	0.95	0.99	0.12	0.97	0.97	-0.50	-0.96
S305	0.48	-0.85	0.95	1.00	0.95	0.91	0.95	0.15	0.94	0.93	-0.45	-0.92
S241	0.51	-0.88	0.99	0.95	1.00	0.95	0.99	0.12	0.97	0.97	-0.51	-0.96
A321	0.48	-0.81	0.95	0.91	0.95	1.00	0.95	0.03	0.91	0.92	-0.52	-0.91
A322	0.50	-0.87	0.99	0.95	0.99	0.95	1.00	0.09	0.96	0.96	-0.52	-0.96
A323	0.14	-0.40	0.12	0.15	0.12	0.03	0.09	1.00	0.29	0.23	0.14	-0.13
A210	0.52	-0.93	0.97	0.94	0.97	0.91	0.96	0.29	1.00	0.97	-0.48	-0.95
E202	0.52	-0.89	0.97	0.93	0.97	0.92	0.96	0.23	0.97	1.00	-0.50	-0.95
S225	-0.27	0.43	-0.50	-0.45	-0.51	-0.52	-0.52	0.14	-0.48	-0.50	1.00	0.52
R215	-0.50	0.90	-0.96	-0.92	-0.96	-0.91	-0.96	-0.13	-0.95	-0.95	0.52	1.00

Table A-2 Network AB correlation coefficient matrix.

Network AB Comp Pt.	R216	R328	Average S332	0.6 P208	S302	S289	S286	S336	S291	T233	W353	R228
R216	1.00	-0.09	0.87	-0.93	-0.97	-0.56	-0.97	-0.97	-0.97	0.86	-0.16	-0.92
R328	-0.09	1.00	-0.10	0.08	0.09	0.13	0.08	0.08	0.09	-0.17	0.09	0.08
S332	0.87	-0.10	1.00	-0.82	-0.84	-0.68	-0.82	-0.83	-0.84	0.84	-0.35	-0.75
P208	-0.93	0.08	-0.82	1.00	0.95	0.57	0.95	0.95	0.96	-0.79	0.18	0.92
S302	-0.97	0.09	-0.84	0.95	1.00	0.54	0.99	1.00	0.99	-0.82	0.12	0.97
S289	-0.56	0.13	-0.68	0.57	0.54	1.00	0.48	0.50	0.55	-0.62	0.82	0.43
S286	-0.97	0.08	-0.82	0.95	0.99	0.48	1.00	1.00	0.99	-0.80	0.05	0.98
S336	-0.97	0.08	-0.83	0.95	1.00	0.50	1.00	1.00	1.00	-0.80	0.07	0.98
S291	-0.97	0.09	-0.84	0.96	0.99	0.55	0.99	1.00	1.00	-0.82	0.13	0.97
T233	0.86	-0.17	0.84	-0.79	-0.82	-0.62	-0.80	-0.80	-0.82	1.00	-0.26	-0.72
W353	-0.16	0.09	-0.35	0.18	0.12	0.82	0.05	0.07	0.13	-0.26	1.00	0.01
R228	-0.92	0.08	-0.75	0.92	0.97	0.43	0.98	0.98	0.97	-0.72	0.01	1.00
R239	-0.62	0.06	-0.53	0.61	0.63	0.33	0.64	0.64	0.64	-0.53	0.03	0.60
T234	-0.08	0.09	-0.31	0.12	0.06	0.83	-0.01	0.01	0.06	-0.21	0.93	-0.05
S227	-0.97	0.10	-0.83	0.95	0.99	0.53	0.99	0.99	0.99	-0.81	0.12	0.97
S306	-0.93	0.07	-0.81	0.91	0.95	0.53	0.95	0.95	0.95	-0.77	0.14	0.93
S232	-0.97	0.09	-0.83	0.95	0.99	0.51	0.99	0.99	0.99	-0.80	0.10	0.98
A318	-0.92	0.08	-0.77	0.90	0.95	0.42	0.96	0.96	0.95	-0.75	0.01	0.94
A319	-0.97	0.08	-0.83	0.95	0.99	0.52	0.99	0.99	0.99	-0.80	0.09	0.97
A320	-0.29	0.09	-0.47	0.31	0.28	0.89	0.21	0.24	0.28	-0.38	0.87	0.17
A211	-0.42	0.06	-0.53	0.44	0.38	0.84	0.33	0.35	0.39	-0.44	0.85	0.27
E202	-0.96	0.09	-0.85	0.94	0.97	0.58	0.96	0.97	0.98	-0.81	0.21	0.95
S320	0.17	-0.11	0.09	-0.11	-0.19	0.24	-0.21	-0.19	-0.17	0.20	0.46	-0.23
R217	0.98	-0.09	0.89	-0.93	-0.97	-0.58	-0.97	-0.97	-0.97	0.87	-0.18	-0.91

Table A-2 Network AB correlation coefficient matrix (continued).

Network AB Comp Pt.	R239	T234	S227	S306	S232	A318	A319	A320	A211	E202	S320	R217
R216	-0.62	-0.08	-0.97	-0.93	-0.97	-0.92	-0.97	-0.29	-0.42	-0.96	0.17	0.98
R328	0.06	0.09	0.10	0.07	0.09	0.08	0.08	0.09	0.06	0.09	-0.11	-0.09
S332	-0.53	-0.31	-0.83	-0.81	-0.83	-0.77	-0.83	-0.47	-0.53	-0.85	0.09	0.89
P208	0.61	0.12	0.95	0.91	0.95	0.90	0.95	0.31	0.44	0.94	-0.11	-0.93
S302	0.63	0.06	0.99	0.95	0.99	0.95	0.99	0.28	0.38	0.97	-0.19	-0.97
S289	0.33	0.83	0.53	0.53	0.51	0.42	0.52	0.89	0.84	0.58	0.24	-0.58
S286	0.64	-0.01	0.99	0.95	0.99	0.96	0.99	0.21	0.33	0.96	-0.21	-0.97
S336	0.64	0.01	0.99	0.95	0.99	0.96	0.99	0.24	0.35	0.97	-0.19	-0.97
S291	0.64	0.06	0.99	0.95	0.99	0.95	0.99	0.28	0.39	0.98	-0.17	-0.97
T233	-0.53	-0.21	-0.81	-0.77	-0.80	-0.75	-0.80	-0.38	-0.44	-0.81	0.20	0.87
W353	0.03	0.93	0.12	0.14	0.10	0.01	0.09	0.87	0.85	0.21	0.46	-0.18
R228	0.60	-0.05	0.97	0.93	0.98	0.94	0.97	0.17	0.27	0.95	-0.23	-0.91
R239	1.00	0.01	0.64	0.60	0.64	0.61	0.64	0.16	0.20	0.60	-0.13	-0.61
T234	0.01	1.00	0.05	0.08	0.03	-0.06	0.04	0.91	0.78	0.12	0.43	-0.11
S227	0.64	0.05	1.00	0.95	0.99	0.95	0.99	0.26	0.39	0.97	-0.17	-0.96
S306	0.60	0.08	0.95	1.00	0.95	0.90	0.95	0.28	0.41	0.93	-0.13	-0.93
S232	0.64	0.03	0.99	0.95	1.00	0.95	0.99	0.25	0.36	0.97	-0.19	-0.96
A318	0.61	-0.06	0.95	0.90	0.95	1.00	0.95	0.16	0.28	0.92	-0.22	-0.92
A319	0.64	0.04	0.99	0.95	0.99	0.95	1.00	0.25	0.37	0.97	-0.18	-0.97
A320	0.16	0.91	0.26	0.28	0.25	0.16	0.25	1.00	0.78	0.33	0.32	-0.31
A211	0.20	0.78	0.39	0.41	0.36	0.28	0.37	0.78	1.00	0.43	0.58	-0.44
E202	0.60	0.12	0.97	0.93	0.97	0.92	0.97	0.33	0.43	1.00	-0.18	-0.96
S320	-0.13	0.43	-0.17	-0.13	-0.19	-0.22	-0.18	0.32	0.58	-0.18	1.00	0.16
R217	-0.61	-0.11	-0.96	-0.93	-0.96	-0.92	-0.97	-0.31	-0.44	-0.96	0.16	1.00

Table A-3 Network B correlation coefficient matrix.

Network B Comp Pt.	R214	R215	Average= 0.28		R200	R208	R209	R212	R222	R223	R234
R214	1.00	0.97	A216	P208	0.06	0.11	0.13	-0.92	0.11	0.15	-0.43
R215	0.97	1.00	-0.22	-0.92	0.05	0.11	0.13	-0.92	0.10	0.14	-0.40
A216	-0.22	-0.22	1.00	0.25	0.35	-0.17	-0.23	0.23	-0.11	-0.17	-0.07
P208	-0.92	-0.92	0.25	1.00	-0.01	-0.09	-0.12	0.94	-0.08	-0.11	0.38
R200	0.06	0.05	0.35	-0.01	1.00	0.07	0.05	0.00	0.12	0.11	-0.07
R208	0.11	0.11	-0.17	-0.09	0.07	1.00	0.76	-0.04	0.79	0.77	-0.05
R209	0.13	0.13	-0.23	-0.12	0.05	0.76	1.00	-0.06	0.91	0.89	-0.04
R212	-0.92	-0.92	0.23	0.94	0.00	-0.04	-0.06	1.00	-0.03	-0.05	0.40
R222	0.11	0.10	-0.11	-0.08	0.12	0.79	0.91	-0.03	1.00	0.93	-0.02
R223	0.15	0.14	-0.17	-0.11	0.11	0.77	0.89	-0.05	0.93	1.00	-0.04
R234	-0.43	-0.40	-0.07	0.38	-0.07	-0.05	-0.04	0.40	-0.02	-0.04	1.00
R327	-0.82	-0.83	0.40	0.84	0.06	-0.07	-0.08	0.87	-0.05	-0.07	0.27
R328	-0.04	-0.04	0.01	0.03	-0.04	0.00	-0.01	0.03	0.00	0.00	0.03
R210	0.01	0.01	0.01	-0.01	0.04	0.18	0.21	0.00	0.22	0.21	0.01
R213	-0.09	-0.09	0.04	0.08	-0.15	-0.01	-0.03	0.09	-0.02	-0.02	0.07
R224	0.03	0.03	0.04	-0.02	0.08	0.23	0.25	-0.01	0.29	0.28	-0.01
R225	-0.09	-0.09	-0.01	0.09	0.02	0.21	0.25	0.11	0.27	0.25	0.02
R235	-0.59	-0.58	0.13	0.60	0.00	-0.08	-0.11	0.58	-0.09	-0.11	0.27
E202	-0.96	-0.95	0.07	0.94	-0.10	-0.06	-0.06	0.97	-0.05	-0.07	0.45
A217	0.29	0.28	0.81	-0.27	0.32	-0.09	-0.12	-0.30	-0.05	-0.09	-0.33
R216	0.99	0.97	-0.23	-0.93	0.05	0.11	0.13	-0.93	0.11	0.15	-0.42
R217	0.98	0.97	-0.21	-0.93	0.06	0.11	0.13	-0.93	0.11	0.15	-0.43

Table A-3 Network B correlation coefficient matrix (continued).

Network B Comp Pt.	R327	R328	R210	R213	R224	R225	R235	E202	A217	R216	R217
R214	-0.82	-0.04	0.01	-0.09	0.03	-0.09	-0.59	-0.96	0.29	0.99	0.98
R215	-0.83	-0.04	0.01	-0.09	0.03	-0.09	-0.58	-0.95	0.28	0.97	0.97
A216	0.40	0.01	0.01	0.04	0.04	-0.01	0.13	0.07	0.81	-0.23	-0.21
P208	0.84	0.03	-0.01	0.08	-0.02	0.09	0.60	0.94	-0.27	-0.93	-0.93
R200	0.06	-0.04	0.04	-0.15	0.08	0.02	0.00	-0.10	0.32	0.05	0.06
R208	-0.07	0.00	0.18	-0.01	0.23	0.21	-0.08	-0.06	-0.09	0.11	0.11
R209	-0.08	-0.01	0.21	-0.03	0.25	0.25	-0.11	-0.06	-0.12	0.13	0.13
R212	0.87	0.03	0.00	0.09	-0.01	0.11	0.58	0.97	-0.30	-0.93	-0.93
R222	-0.05	0.00	0.22	-0.02	0.29	0.27	-0.09	-0.05	-0.05	0.11	0.11
R223	-0.07	0.00	0.21	-0.02	0.28	0.25	-0.11	-0.07	-0.09	0.15	0.15
R234	0.27	0.03	0.01	0.07	-0.01	0.02	0.27	0.45	-0.33	-0.42	-0.43
R327	1.00	0.02	0.04	0.07	-0.01	0.10	0.51	0.82	-0.04	-0.83	-0.82
R328	0.02	1.00	-0.01	0.67	0.00	0.00	0.03	0.04	-0.02	-0.04	-0.04
R210	0.04	-0.01	1.00	-0.01	0.07	0.06	0.00	-0.01	0.03	0.02	0.02
R213	0.07	0.67	-0.01	1.00	0.00	0.00	0.07	0.09	-0.02	-0.09	-0.09
R224	-0.01	0.00	0.07	0.00	1.00	0.08	0.00	-0.02	0.05	0.03	0.03
R225	0.10	0.00	0.06	0.00	0.08	1.00	0.04	0.10	-0.05	-0.09	-0.09
R235	0.51	0.03	0.00	0.07	0.00	0.04	1.00	0.59	-0.19	-0.60	-0.59
E202	0.82	0.04	-0.01	0.09	-0.02	0.10	0.59	1.00	-0.45	-0.96	-0.96
A217	-0.04	-0.02	0.03	-0.02	0.05	-0.05	-0.19	-0.45	1.00	0.29	0.30
R216	-0.83	-0.04	0.02	-0.09	0.03	-0.09	-0.60	-0.96	0.29	1.00	0.98
R217	-0.82	-0.04	0.02	-0.09	0.03	-0.09	-0.59	-0.96	0.30	0.98	1.00

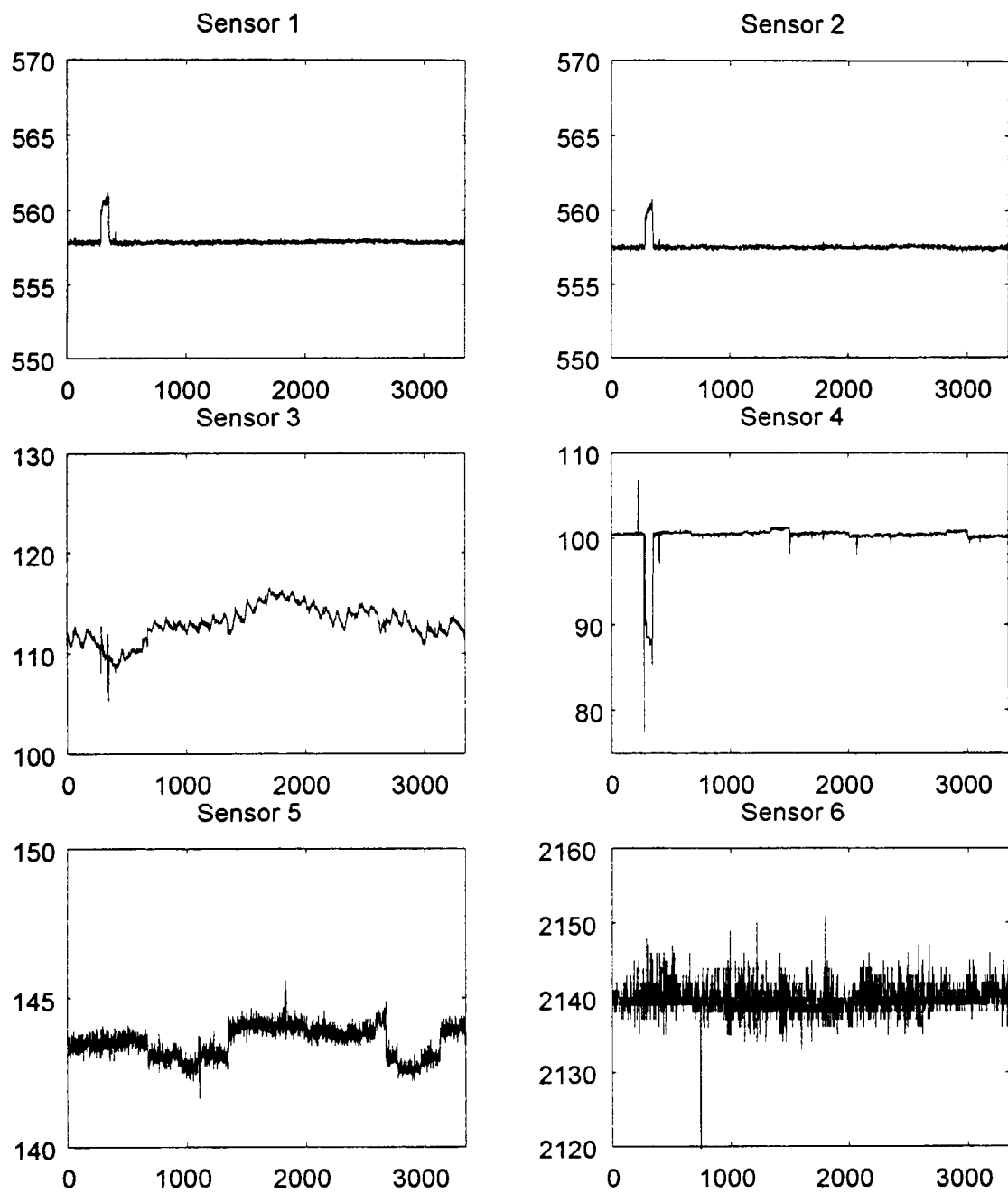


Figure A5.1 Network B - 5 fault-free weeks of data.

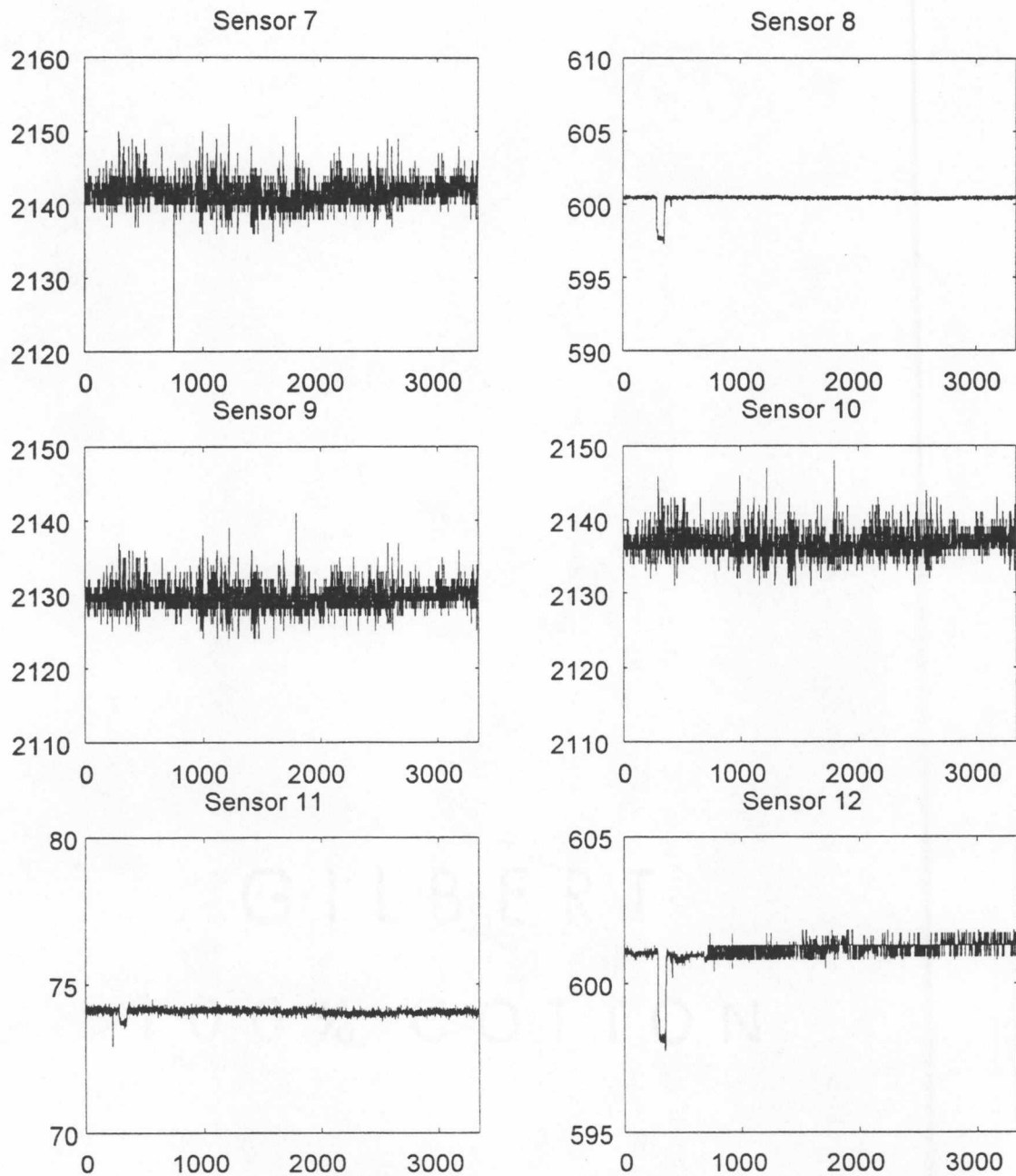


Figure A5.1 Network B - 5 fault-free weeks of data (continued).

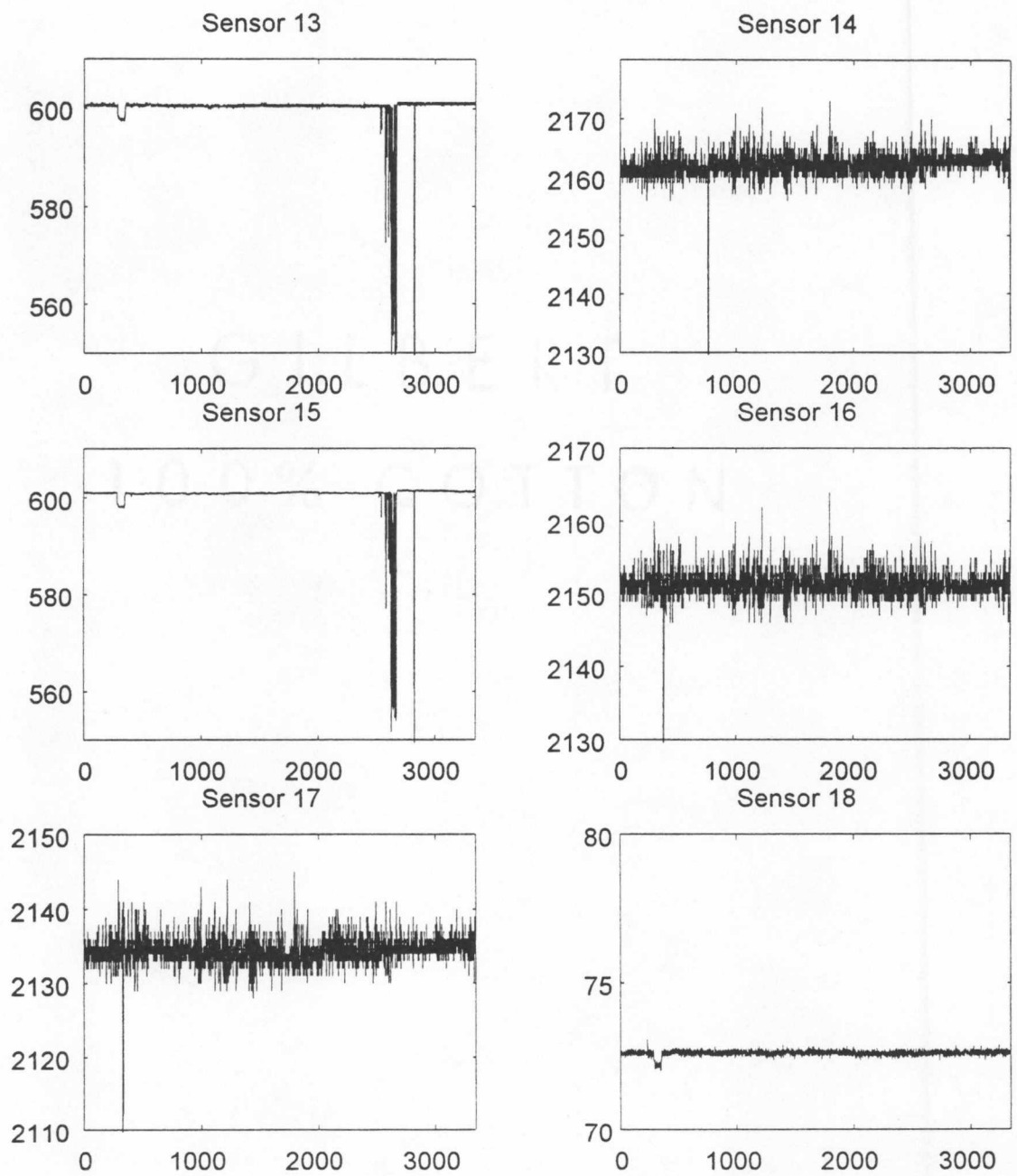


Figure A5.1 Network B - 5 fault-free weeks of data (continued).

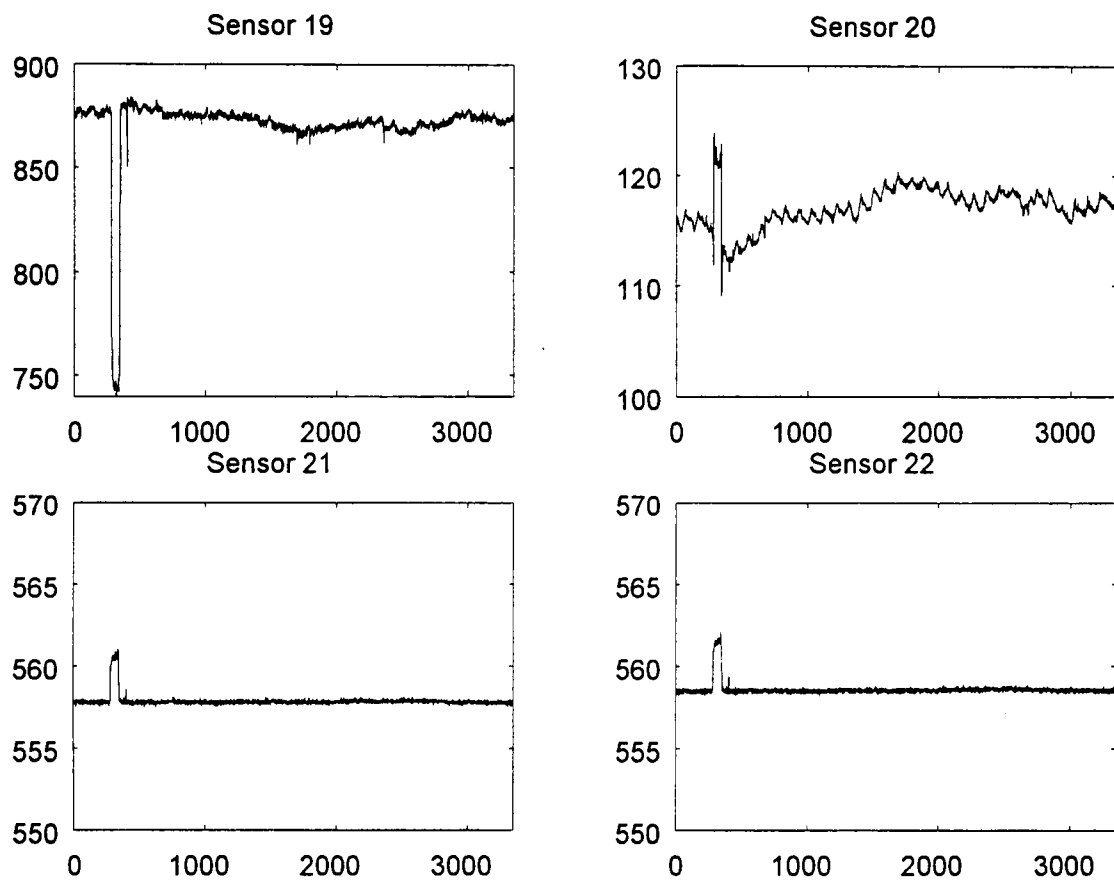


Figure A5.1 Network B - 5 fault-free weeks of data (continued).

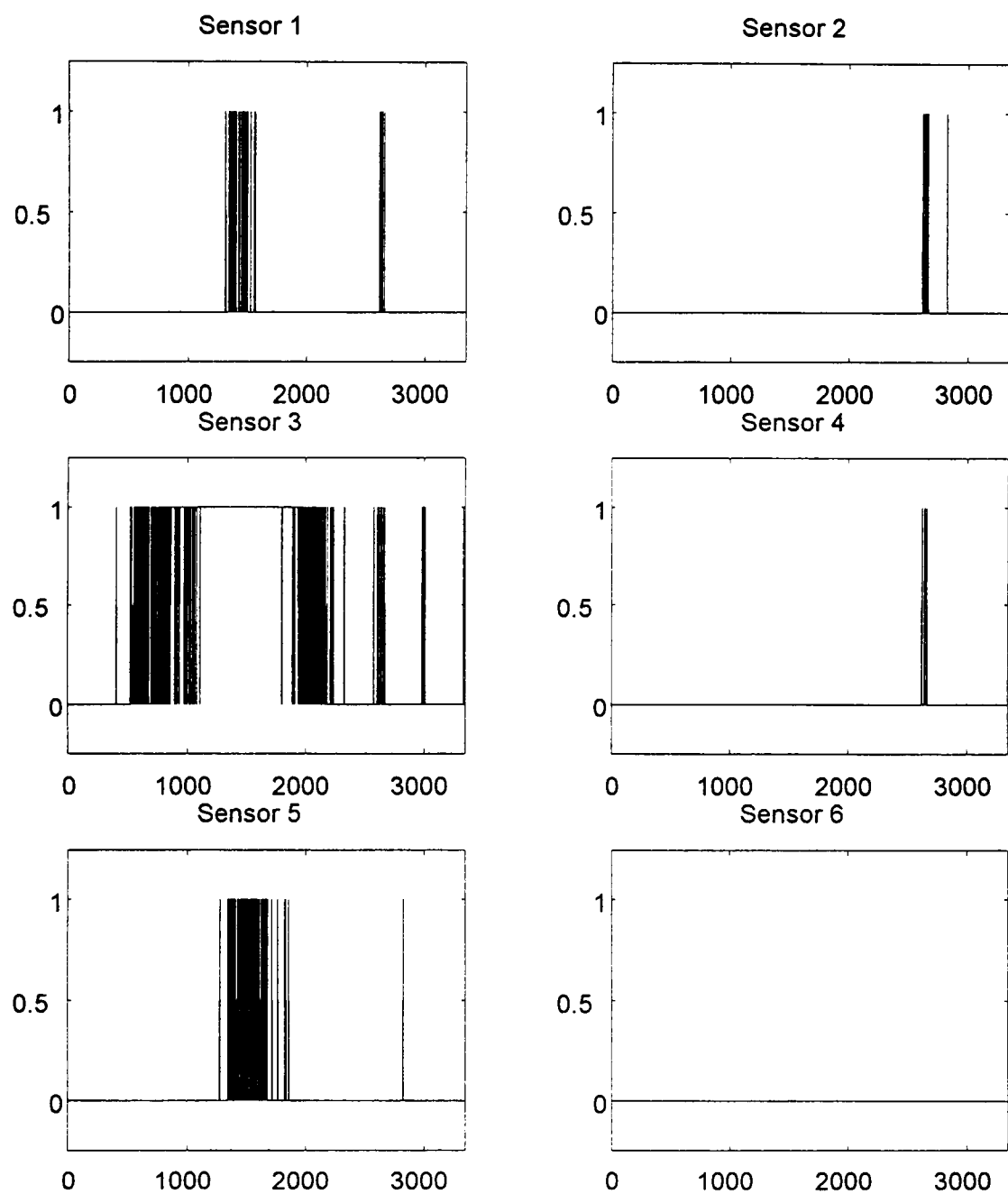


Figure A5.2 Network B - Non-robust training set fault hypothesis.

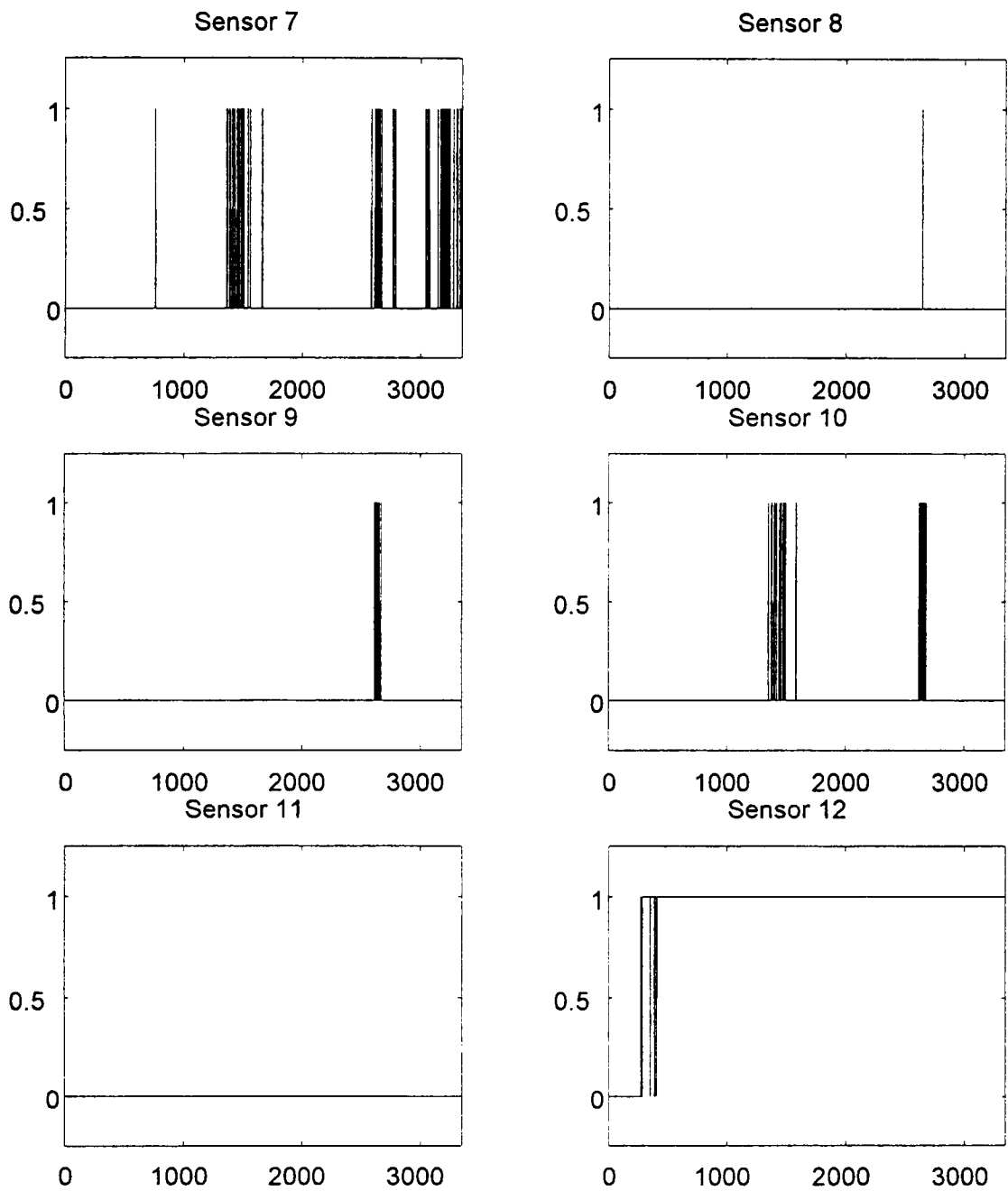


Figure A5.2 Network B - Non-robust training set fault hypothesis (continued).

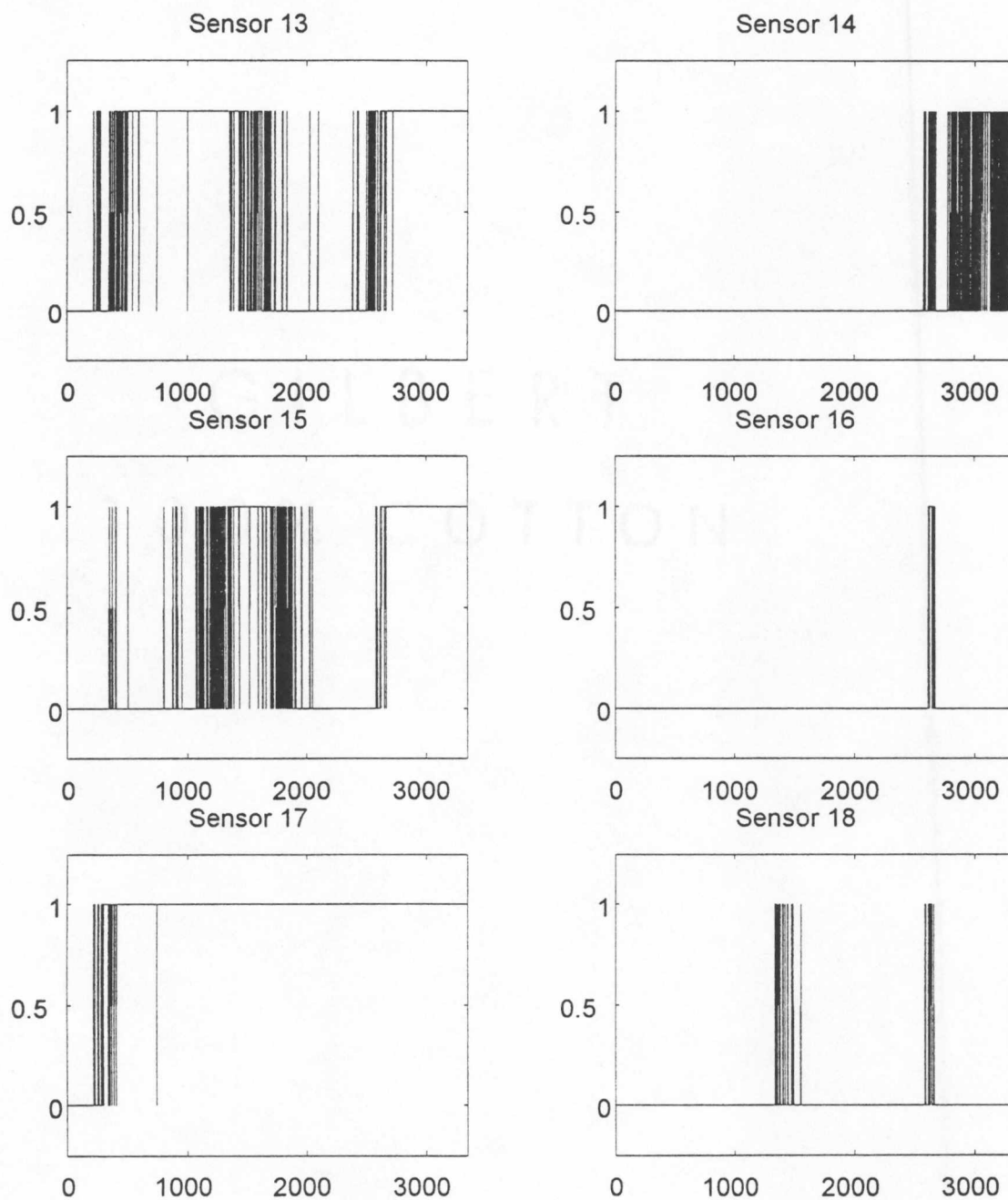


Figure A5.2 Network B - Non-robust training set fault hypothesis (continued).

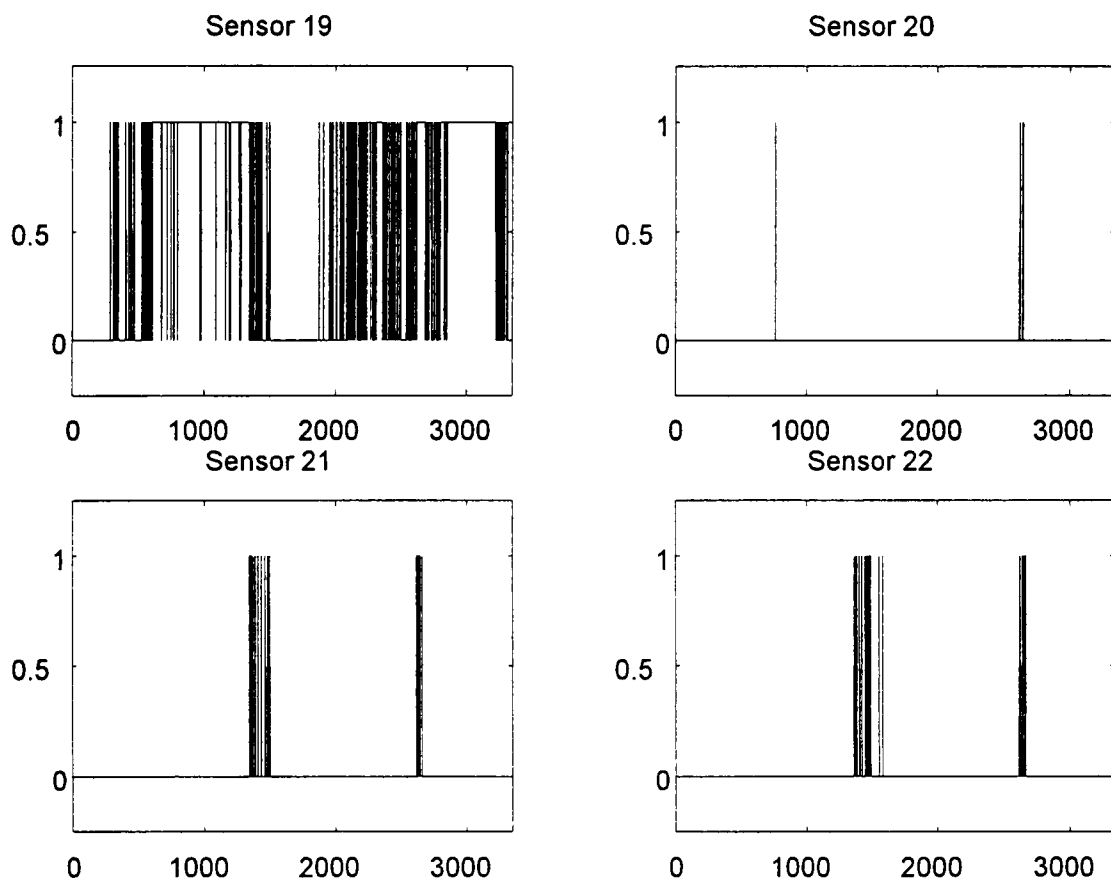


Figure A5.2 Network B - Non-robust training set fault hypothesis (continued).

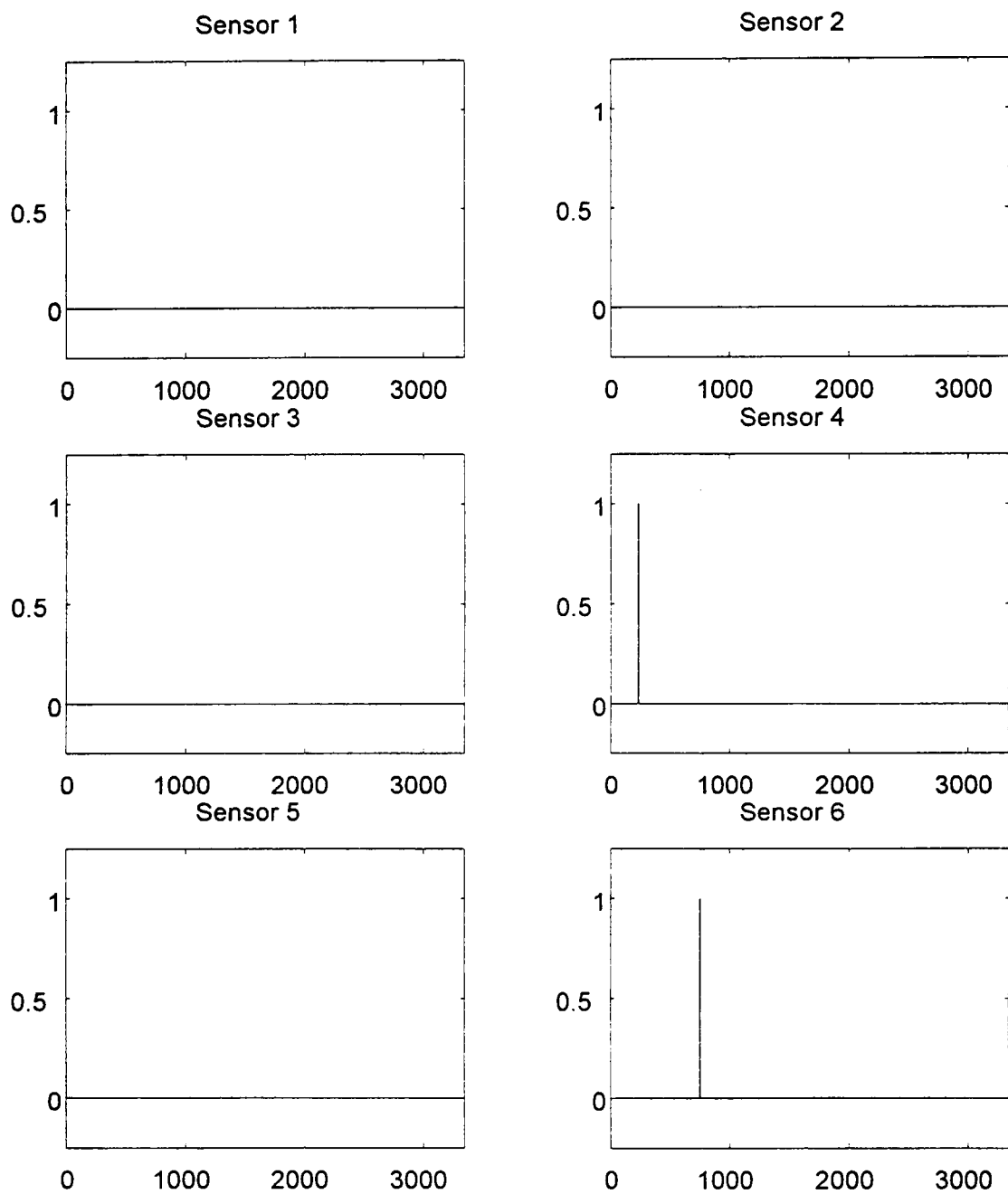


Figure A5.3 Network B - Robust training set fault hypothesis.

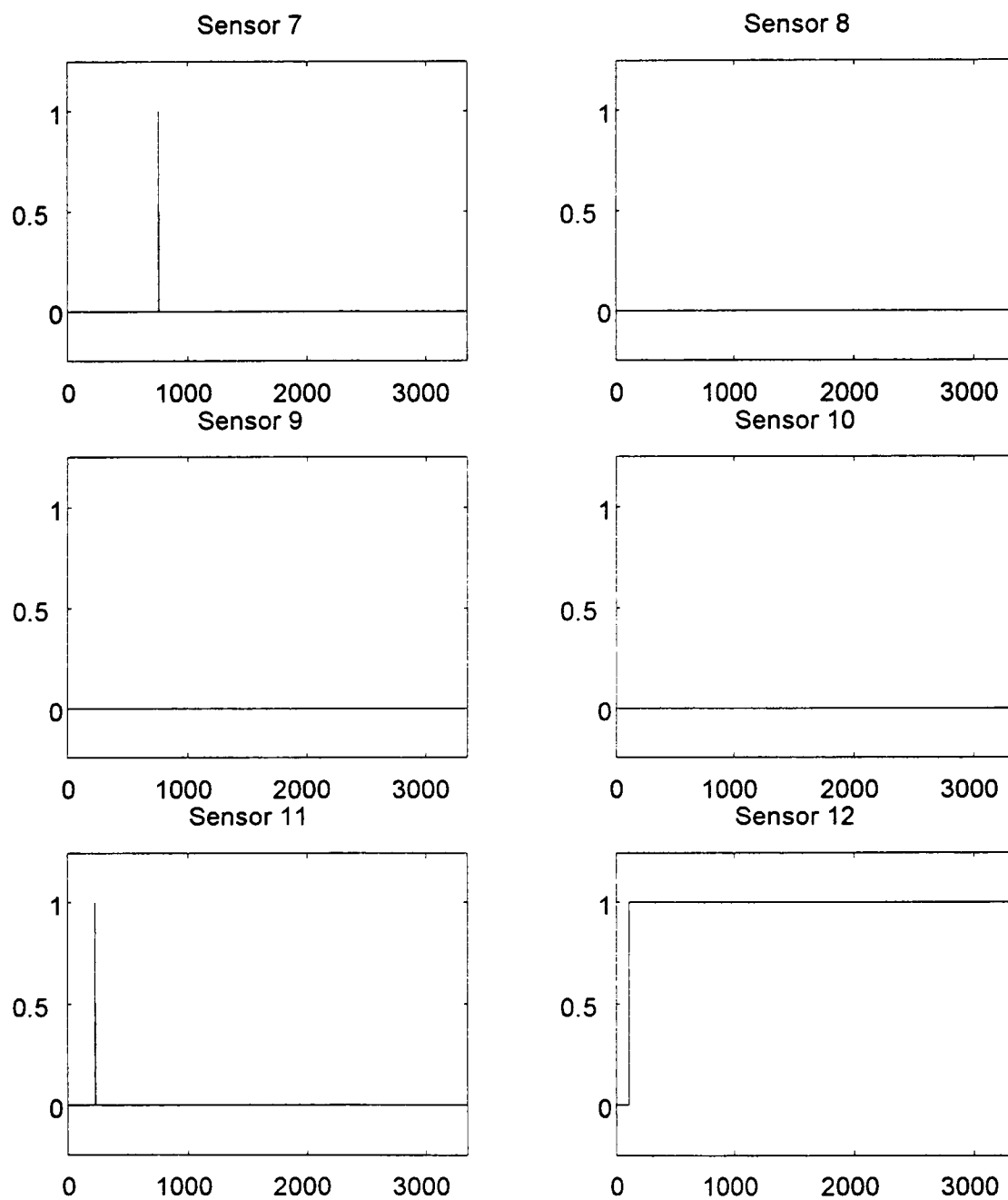


Figure A5.3 Network B - Robust training set fault hypothesis (continued).

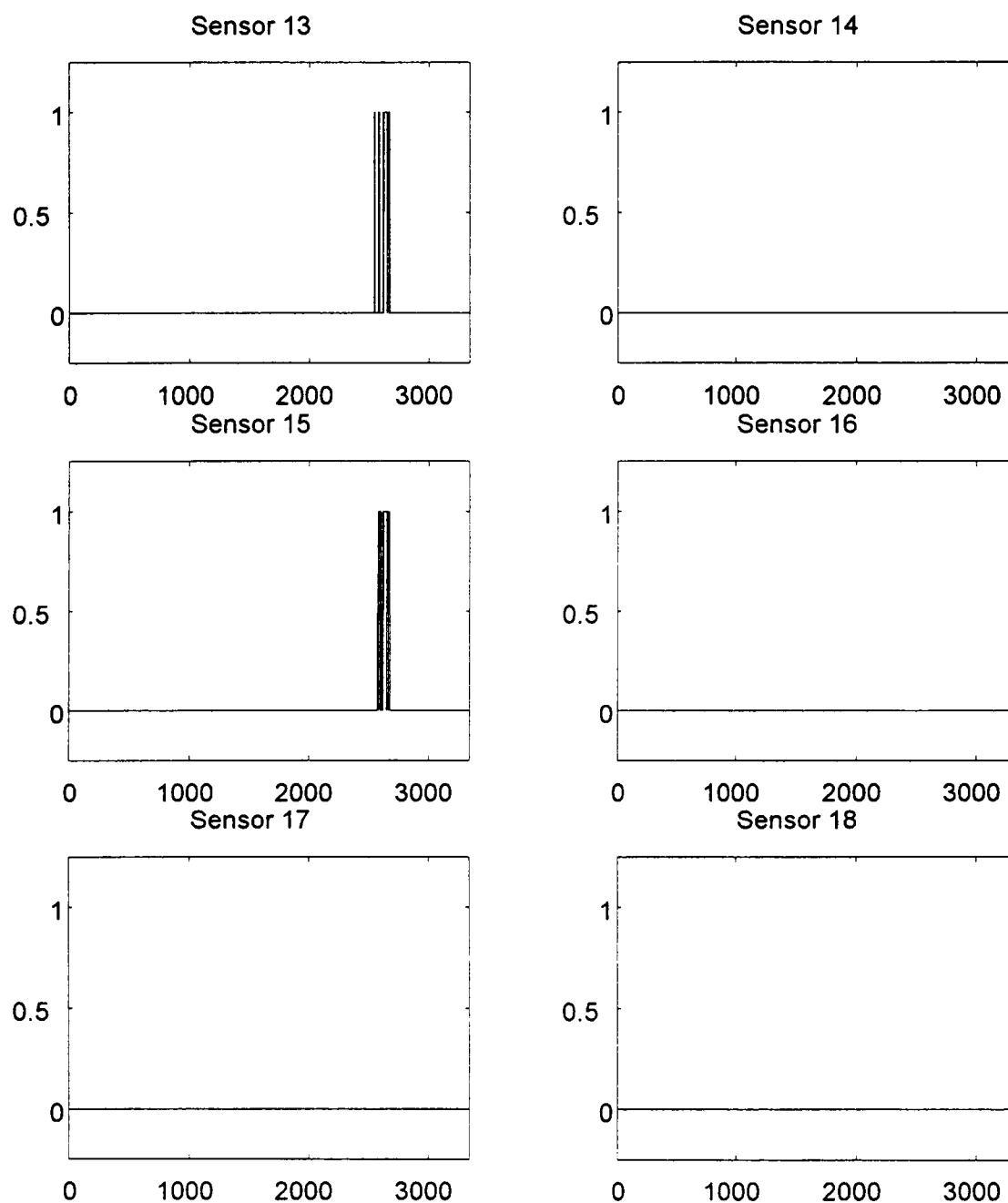


Figure A5.3 Network B - Robust training set fault hypothesis (continued).

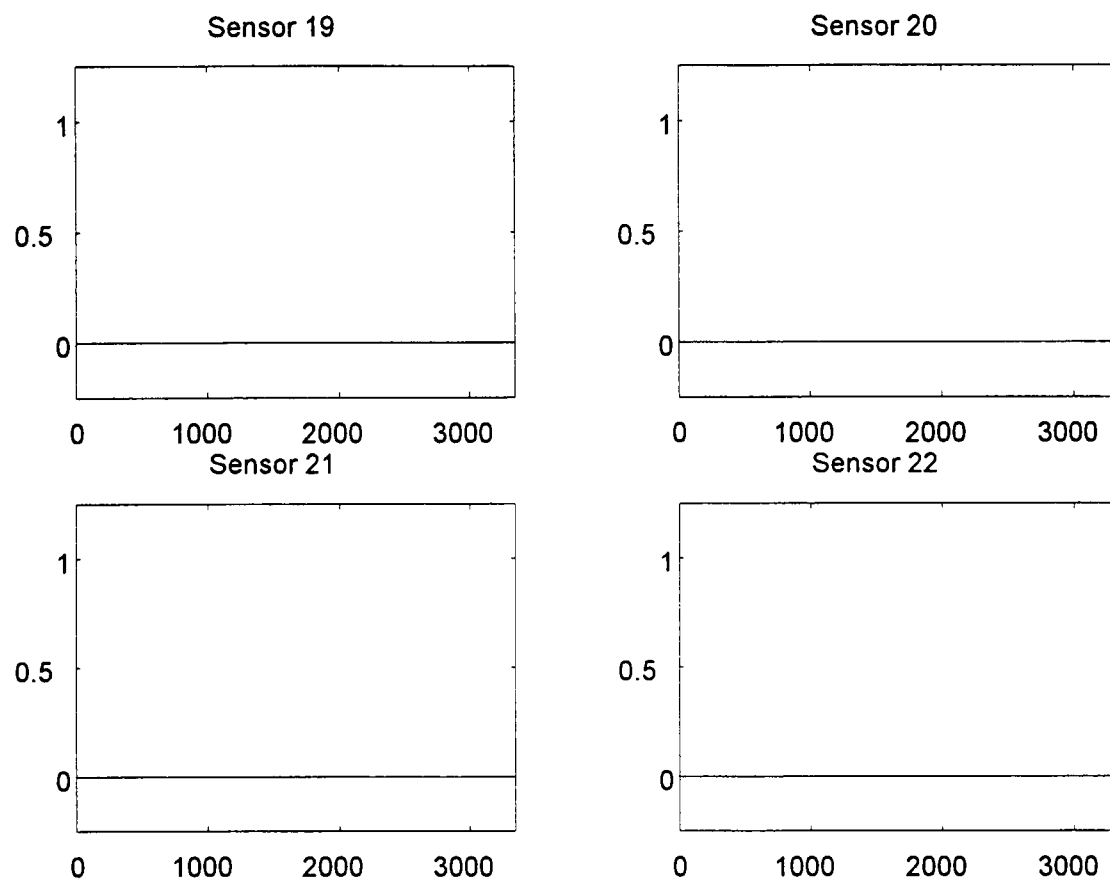


Figure A5.3 Network B - Robust training set fault hypothesis (continued).

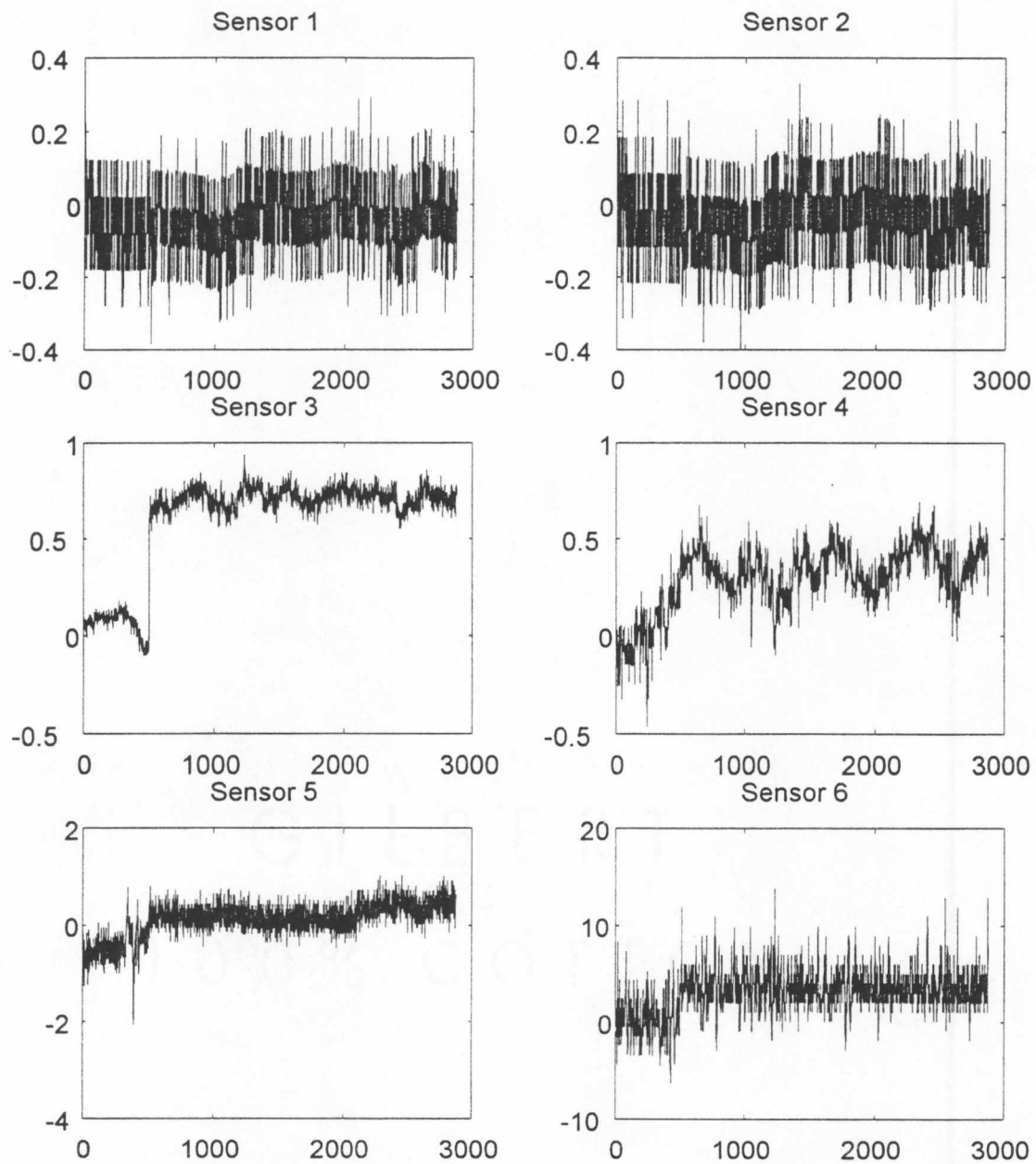


Figure A5.4 Network residuals for gross fault in sensor 9.

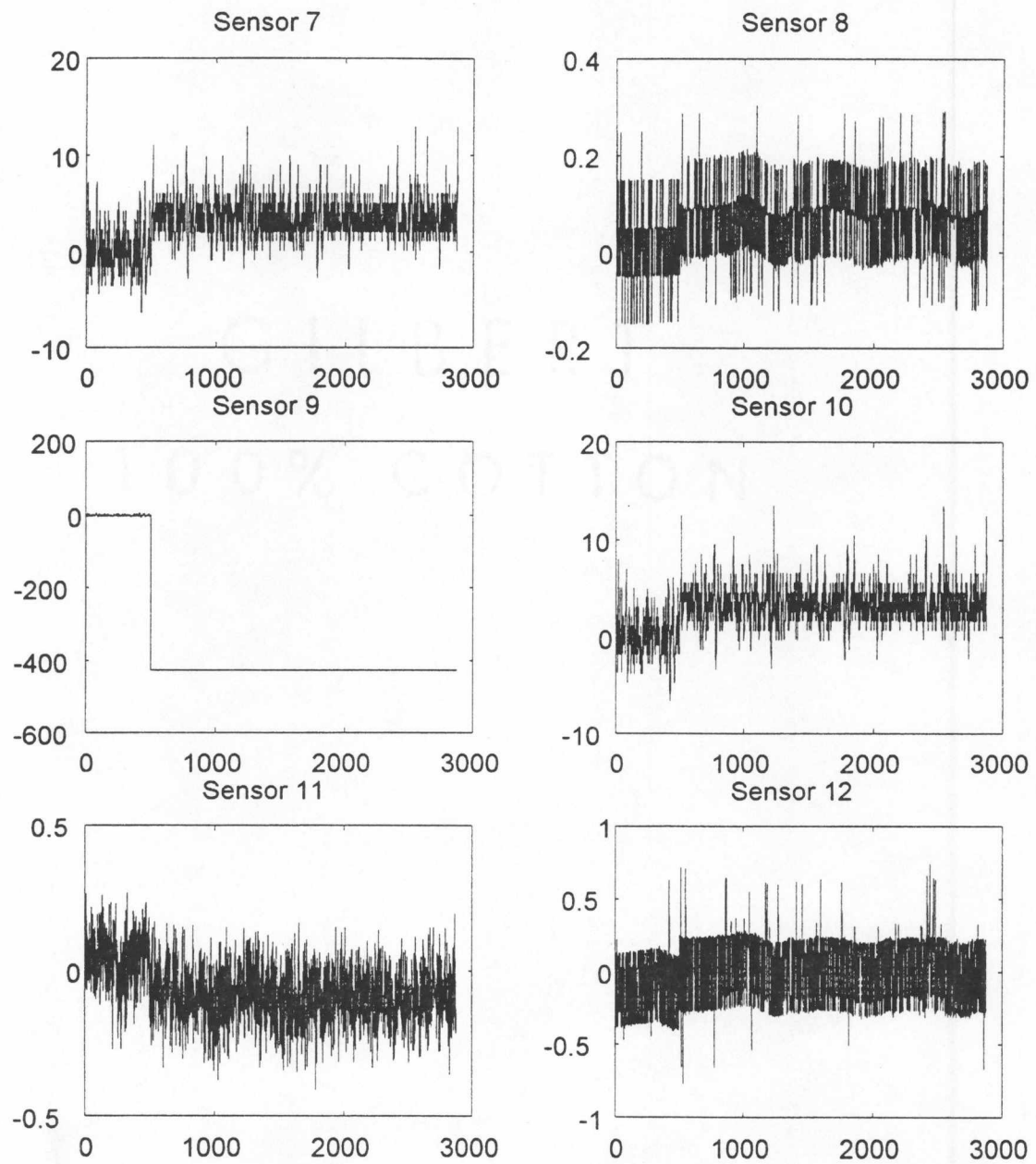


Figure A5.4 Network residuals for gross fault in sensor 9 (continued).

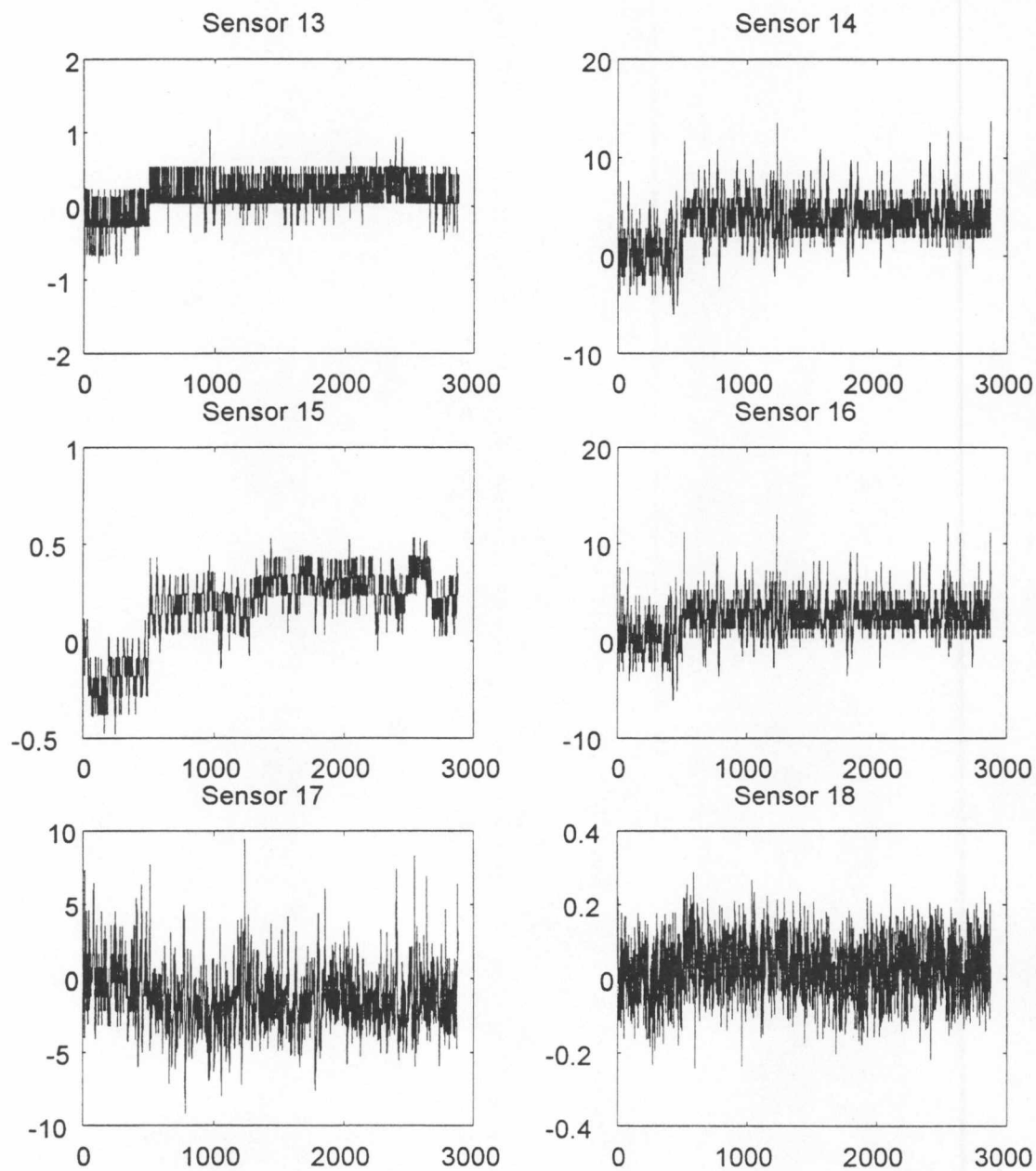


Figure A5.4 Network residuals for gross fault in sensor 9 (continued).

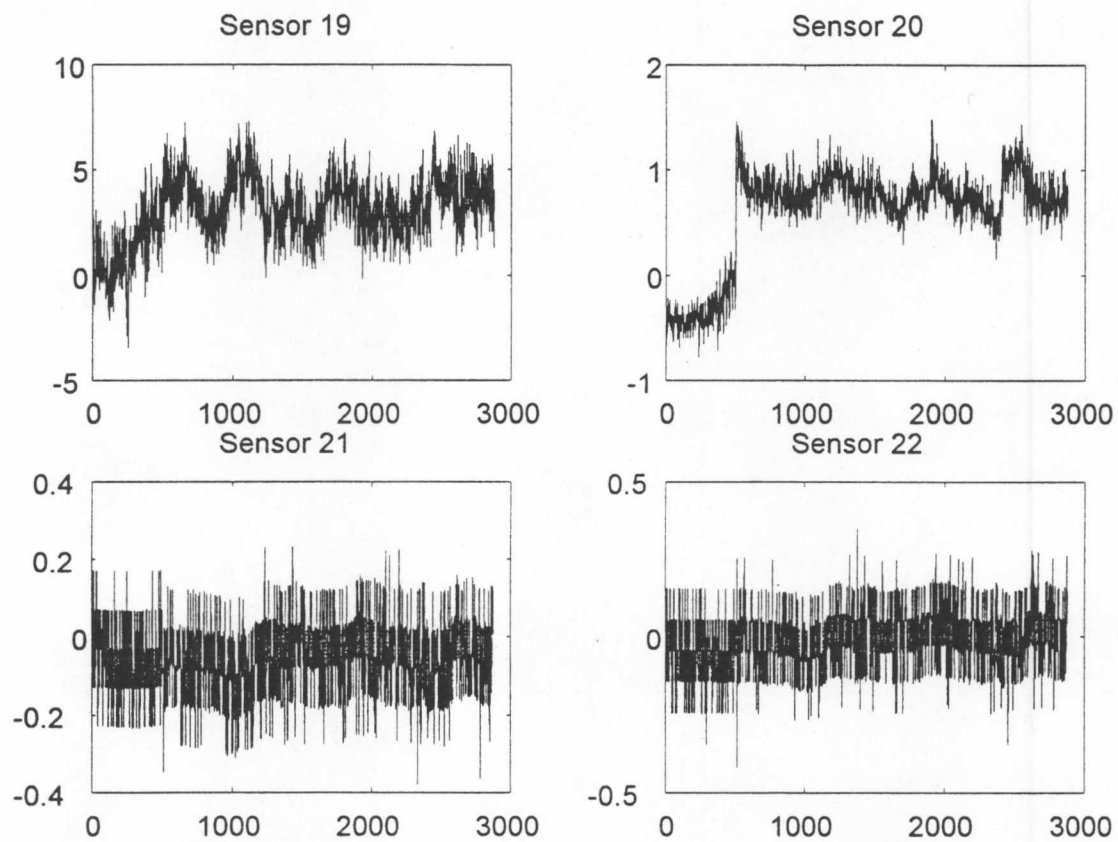


Figure A5.4 Network residuals for gross fault in sensor 9 (continued).

Appendix B

Matlab Computer Code

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% bplse4:
%
% Master file used to train a 3 hidden layer feed forward
% autoassociative neural network with accelerated
% backpropagation (adaptive learning rate and momentum)
% with a linear output layer. The linear weights are found
% using a singular value decomposition method.
% Uses modified code segments from Matlabs Neural
% Networks Toolbox.
%
% Architecture example.
%
% input - h1 - h2 - h3 - output
%
% where number of hidden nodes in h1 = h3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flops(0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters and load training data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

vecfile = input('Enter filename of input/output data vectors: ','s');
numhid1 = input('How many neurons in the first and third hidden layers? ');
numhid2 = input('How many neurons in the second hidden layer? ');
seltrn = input('Would you like to use LSE in the output layer [y n]: ','s');
robtrn = input('Would you like to use robust training? [y n]: ','s');
numhid3 = numhid1;
eval(['load ',vecfile])
[e_goal]=goal(x); % Determine sum squared error goal

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up default parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

fprintf('\n\nThe default variables are:\n\n');
fprintf('Learning rate = .03\n'); lr = .03;
fprintf('Momentum = .91\n'); mc = .91;
fprintf('Error Ratio = 1.04\n'); er = 1.04;
fprintf('\nRecomended SSE training error goal of: %4.2f\n',e_goal);
fprintf('Errors plotted in progress every 50 epochs.\n'); df= 50; plotflag = 1;
fprintf('Standard backprop training epochs = 350.\n'); cy = 350;
fprintf('Maximum number of training epochs = 500.\n'); me=500;

```

```

fprintf('Learning rate increase = 1.05.\n');
fprintf('Learning rate decrease = 0.70.\n');
fprintf('Random initialation of weights.\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Allow user to change default parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choice = input('Are you satisfied with these selections? [y,n]:','s');
if choice == 'n'
lr = input('Enter learning rate: ');
cy = input('Enter standard backprop training epochs: ');
mc = input('Enter momentum: ');
eg = input('Output error tolerance (rms): ');
df = input('How often to display errors (# of epochs): ');
me = input('Maximum number of epochs to train: ');
er = input('Error Ratio: ');
im = input('Enter learning rate increase: ');
dm = input('Enter learning rate decrease: ');
end
tp = [df me eg lr im dm mc er cy];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Printout network arameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if robtrn == 'y'
[x,y]=aarobust(x,y);
end
[nov1, numin] = size(x); % x are input vectors
[nov2,numout] = size(y); % y are target vectors
if nov1 ~= nov2
error('The number of input vectors and target vectors has to be the same.')
end
fprintf('\nThis network has:\n\n');
fprintf(' %0.f input neurons\n',numin);
fprintf(' %0.f neurons in the 1st hidden layer\n',numhid1);
fprintf(' %0.f neurons in the 2nd hidden layer\n',numhid2);
fprintf(' %0.f neurons in the 3rd hidden layer\n',numhid1);
fprintf(' %0.f output neurons\n\n',numout);
fprintf('There are %0.f input/output pairs in this training set.\n\n',nov1);
b=cputime;

```

```

%%%%%%%%%%%%%%
%      Linear Scale input and Output.
%%%%%%%%%%%%%%

```

```

[p,xm,xs] = scale(x);
[t] = scale(y,xm,xs);
p=p';
t=t';

```

```

f1='logsig';           % Set type of activation functions in each layer
f2='logsig';
f3='logsig';
f4='purelin';

```

```

%%%%%%%%%%%%%%
%      Initialize weight vectors to small values using initff.
%%%%%%%%%%%%%%

```

```

[W1,B1,W2,B2,W3,B3,W4,B4]=initff(p,numhid1,f1,numhid2,f2,numhid3,f3,t,f4);

```

```

%%%%%%%%%%%%%%
%      First Standard Backprop Training To Reduce the Error
%%%%%%%%%%%%%%

```

```

[W1,B1,W2,B2,W3,B3,W4,B4,i,tr] = ...
lsetbpx4(W1,B1,f1,W2,B2,f2,W3,B3,f3,W4,B4,f4,p,t,tp);
if seltrn == 'y'

```

```

%%%%%%%%%%%%%%
%      Train the BP/SVD neural network.
%%%%%%%%%%%%%%

```

```

df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
df3 = feval(f3,'delta');
df4 = feval(f4,'delta');
dw1 = W1*0;
db1 = B1*0;
dw2 = W2*0;
db2 = B2*0;
dw3 = W3*0;
db3 = B3*0;
dw4 = W4*0;
db4 = B4*0;

```

```

MC = 0;

% PRESENTATION PHASE
a1 = feval(f1,W1*p,B1);
a2 = feval(f2,W2*a1,B2);
a3 = feval(f3,W3*a2,B3);
a4 = feval(f4,W4*a3,B4);
e = t-a4;
SSE = sumsqr(e);

% TRAINING RECORD
tr = zeros(2,me+1);
tr(1:2,1) = [SSE; lr];

% PLOTTING FLAG
[r,q] = size(p);
[s,q] = size(t);
plottype = (max(r,s) == 1);

% PLOTTING
clg
message = sprintf('TRAINBPXLS: %%g/%%g epochs, lr = %%g, SSE = %%g.\n',me);
fprintf(message,0,lr,SSE)
if plottype
    h = plotfa(p,t,p,a4);
else
    h = plottr(tr(1:2,1),eg);
end
metot = i;
for i=1:me

%% Train the hidden layer of the network.

% CHECK PHASE
if SSE < eg, i=i-1; break, end

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);
d2 = feval(df2,a2,d3,W3);
d1 = feval(df1,a1,d2,W2);

```

```

% LEARNING PHASE
[dw1,db1] = learnbpm(p,d1,lr,MC,dw1,db1);
[dw2,db2] = learnbpm(a1,d2,lr,MC,dw2,db2);
[dw3,db3] = learnbpm(a2,d3,lr,MC,dw3,db3);
MC = mc;
new_w1 = max(-10,min(+10,W1 + dw1));
new_b1 = max(-10,min(+10,B1 + db1));
new_w2 = max(-10,min(+10,W2 + dw2));
new_b2 = max(-10,min(+10,B2 + db2));
new_w3 = max(-10,min(+10,W3 + dw3));
new_b3 = max(-10,min(+10,B3 + db3));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Train the output layer of the network using the SVD technique.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

new_a1 = feval(f1,new_w1*p,new_b1);
new_a2 = feval(f2,new_w2*new_a1,new_b2);
new_a3 = feval(f3,new_w3*new_a2,new_b3);
[new_w4,new_b4]=trlms(new_a3,t);          % call SVD algorithm
new_a4 = feval(f4,new_w4*new_a3,new_b4);
new_e = t-new_a4;
new_SSE = sumsqr(new_e);

% MOMENTUM & ADAPTIVE LEARNING RATE PHASE
if new_SSE > SSE*er
    lr = lr * dm;
    MC = 0;
else
    if new_SSE < SSE
        lr = lr * im;
    end
    W1 = new_w1; B1 = new_b1; a1 = new_a1;
    W2 = new_w2; B2 = new_b2; a2 = new_a2;
    W3 = new_w3; B3 = new_b3; a3 = new_a3;
    W4 = new_w4; B4 = new_b4; a4 = new_a4;
    e = new_e; SSE = new_SSE;

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);
d2 = feval(df2,a2,d3,W3);
d1 = feval(df1,a1,d2,W2);
end

```

```
% TRAINING RECORD
```

```
tr(:,i+1) = [SSE; lr];
```

```
% PLOTTING
```

```
if rem(i,df) == 0
```

```
fprintf(message,i,lr,SSE)
```

```
if plottype
```

```
    delete(h);
```

```
    h = plot(p,a4);
```

```
        drawnow
```

```
else
```

```
    h = plottr(tr(1:2,1:(i+1)),eg,h);
```

```
end
```

```
end
```

```
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%      Finish Plotting.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% TRAINING RECORD
```

```
tr = tr(1:2,1:(i+1));
```

```
% PLOTTING
```

```
if rem(i,df) ~= 0
```

```
    fprintf(message,i,lr,SSE)
```

```
    if plottype
```

```
        delete(h);
```

```
        plot(p,a4);
```

```
            drawnow
```

```
    else
```

```
        plottr(tr,eg,h);
```

```
    end
```

```
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%      Tolerance criteria not met by BP.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if i == me
```

```
    disp(' ')
```

```
    disp('TRAINBPX: Network error did not reach the error goal.')
```

```
    disp(' Further training may be necessary, or try different')
```

```
    disp(' initial weights and biases and/or more hidden neurons.')
```

```

disp(' ')
fprintf('\n*** SSE = %f \n\n',min(tr(1,:)))
else

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Tolerance criteria met by BP.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        fprintf('\n\n*** Training complete after %0.f epochs! ***\n',metot+i)
        fprintf('\n*** SSE = %f \n\n',min(tr(1,:)))
end
save weights4.mat W1 B1 W2 B2 W3 B3 W4 B4 xm xs
else
save weights4.mat W1 B1 W2 B2 W3 B3 W4 B4 xm xs
end

a=cputime;
time=a-b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function
[w1,b1,w2,b2,w3,b3,w4,b4,i,tr]=lsetbpx4(w1,b1,f1,w2,b2,f2,w3,b3,f3,w4,b4,f4,p,t,tp)

%LSETBPX4 Train 4-layer (3 hidden) feed-forward network w/fast backpropagation.
%
%   [W1,B1,W2,B2,W3,B3,W4,B4,TE,TR] =
%   LSETBPX4(W1,B2,F1,W1,B1,F2,W3,B3,F3,W4,B4,F4,P,T,TP)
%   Wi - Weight matrix for the ith layer.
%   Bi - Bias vector for the ith layer.
%   Fi - Transfer function (string) for the ith layer.
%   P - RxQ matrix of input vectors.
%   T - SxQ matrix of target vectors.
%   TP - Training parameters (optional).
% Returns:
%   Wi - new weights.
%   Bi - new biases.
%   TE - the actual number of epochs trained.
%   TR - training record: [row of errors]
%
% Training parameters are:
%   TP(1) - Epochs between updating display, default = 25.
%   TP(2) - Maximum number of epochs to train, default = 1000.
%   TP(3) - Sum-squared error goal, default = 0.02.
%   TP(4) - Learning rate, 0.01.
%   TP(5) - Learning rate increase, default = 1.05.
%   TP(6) - Learning rate decrease, default = 0.7.
%   TP(7) - Momentum constant, default = 0.9.
%   TP(8) - Maximum error ratio, default = 1.04.
%   TP(9) - Number of initial epochs for standard training = 300.
%   Missing parameters and NaN's are replaced with defaults.
%
% Matlab Neural Networks toolbox code modified for a three hidden layer network.
% Also modified for limited weight change per iteration.

if nargin < 13,error('Not enough arguments.');
```

```

% TRAINING PARAMETERS
if nargin == 13, tp = []; end
tp = nndef(tp,[25 1000 0.02 0.01 1.05 0.7 0.9 1.04 300]);
df = tp(1);
me = tp(2);
eg = tp(3);
lr = tp(4);
im = tp(5);

```

```

dm = tp(6);
mc = tp(7);
er = tp(8);
cy = tp(9);
me = cy;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
df3 = feval(f3,'delta');
df4 = feval(f4,'delta');
dw1 = w1*0;
db1 = b1*0;
dw2 = w2*0;
db2 = b2*0;
dw3 = w3*0;
db3 = b3*0;
dw4 = w4*0;
db4 = b4*0;
MC = 0;

```

```

% PRESENTATION PHASE

```

```

a1 = feval(f1,w1*p,b1);
a2 = feval(f2,w2*a1,b2);
a3 = feval(f3,w3*a2,b3);
a4 = feval(f4,w4*a3,b4);
e = t-a4;
SSE = sumsqr(e);

```

```

% TRAINING RECORD

```

```

tr = zeros(2,me+1);
tr(1:2,1) = [SSE; lr];

```

```

% PLOTTING FLAG

```

```

[r,q] = size(p);
[s,q] = size(t);
plottype = (max(r,s) == 1) & 0;

```

```

% PLOTTING

```

```

clg
message = sprintf('TRAINBPX: %%g/%%g epochs, lr = %%g, SSE = %%g.\n',me);
fprintf(message,0,lr,SSE)
if plottype

```

```

    h = plotfa(p,t,p,a4);
else
    h = plottr(tr(1:2,1),eg);
end

```

% BACKPROPAGATION PHASE

```

d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,w4);
d2 = feval(df2,a2,d3,w3);
d1 = feval(df1,a1,d2,w2);

```

```

for i=1:me

```

% CHECK PHASE

```

if SSE < eg, i=i-1; break, end

```

% LEARNING PHASE

```

[dw1,db1] = learnbpm(p,d1,lr,MC,dw1,db1);
[dw2,db2] = learnbpm(a1,d2,lr,MC,dw2,db2);
[dw3,db3] = learnbpm(a2,d3,lr,MC,dw3,db3);
[dw4,db4] = learnbpm(a3,d4,lr,MC,dw4,db4);
MC = mc;

```

```

new_w1 = max(-10,min(+10,w1 + dw1));
new_b1 = max(-10,min(+10,b1 + db1));
new_w2 = max(-10,min(+10,w2 + dw2));
new_b2 = max(-10,min(+10,b2 + db2));
new_w3 = max(-10,min(+10,w3 + dw3));
new_b3 = max(-10,min(+10,b3 + db3));
new_w4 = max(-10,min(+10,w4 + dw4));
new_b4 = max(-10,min(+10,b4 + db4));

```

% limited weight change

% PRESENTATION PHASE

```

new_a1 = feval(f1,new_w1*p,new_b1);
new_a2 = feval(f2,new_w2*new_a1,new_b2);
new_a3 = feval(f3,new_w3*new_a2,new_b3);
new_a4 = feval(f4,new_w4*new_a3,new_b4);
new_e = t-new_a4;
new_SSE = sumsqr(new_e);

```

% MOMENTUM & ADAPTIVE LEARNING RATE PHASE

```

if new_SSE > SSE*er
    lr = lr * dm;
    MC = 0;
else

```

```

if new_SSE < SSE
    lr = lr * im;
    end
w1 = new_w1; b1 = new_b1; a1 = new_a1;
w2 = new_w2; b2 = new_b2; a2 = new_a2;
w3 = new_w3; b3 = new_b3; a3 = new_a3;
w4 = new_w4; b4 = new_b4; a4 = new_a4;
e = new_e; SSE = new_SSE;

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,w4);
d2 = feval(df2,a2,d3,w3);
d1 = feval(df1,a1,d2,w2);
end

% TRAINING RECORD
tr(1:2,i+1) = [SSE; lr];

% PLOTTING
if rem(i,df) == 0
    fprintf(message,i,lr,SSE)
    if plotype
        delete(h);
        h = plot(p,a4);
    else
        h = plottr(tr(1:2,1:(i+1)),eg,h);
    end
end
end

% TRAINING RECORD
tr = tr(1:2,1:(i+1));

% PLOTTING
if rem(i,df) ~= 0
    fprintf(message,i,lr,SSE)
    if plotype
        delete(h);
        plot(p,a4);
    else
        plottr(tr,eg,h);
    end
end
end

```

```

% WARNINGS
if SSE > eg
    disp('')
    disp('TRAINBPX: Network error did not reach the error goal.')
    disp(' Training will resume with Least Squares Error (SVD) in the')
    disp(' linear output layer and standard backprop in the three')
    disp(' hidden layers. ')
    disp('')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ccbplse4: Cross Validation Training
% Master file used to train a 3 hidden layer feed forward
% autoassociative neural network with accelerated
% backpropagation (adaptive learning rate and momentum)
% with a linear output layer. The linear weights are found
% using a singular value decomposition method.
% Uses modified code segments from Matlabs Neural
% Networks Toolbox.
%
% x = input training matrix y = target training matrix
% xt = input testing matrix yt = target testing matrix
% Architecture example.
%
% input - h1 - h2 - h3 - output
%
% where number of hidden nodes in h1 = h3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
flops(0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters and load training data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vecfile = input('Enter filename of input/output data vectors: ','s');
numhid1 = input('How many neurons in the first and third hidden layers? ');
numhid2 = input('How many neurons in the second hidden layer? ');
seltrn = input('Would you like to use LSE in the output layer [y n]: ','s');
robtrn = input('Would you like to use robust training? [y n]: ','s');
numhid3 = numhid1;
eval(['load ',vecfile])
[e_goal]=goal(x); % Determine sum squared error goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up default parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('\n\nThe default variables are:\n\n');
fprintf('Learning rate = .03\n'); lr = .03;
fprintf('Momentum = .91\n'); mc = .91;
fprintf('Error Ratio = 1.04\n'); er = 1.04;
fprintf('\nRecomended SSE training error goal of: %4.2f\n\n',e_goal);
fprintf('Errors plotted in progress every 50 epochs.\n'); df= 50; plotflag = 1;

```

```

fprintf('Standard backprop training epochs = 350.\n');
fprintf('Maximum number of training epochs = 500.\n');
fprintf('Learning rate increase = 1.05.\n');
fprintf('Learning rate decrease = 0.70.\n');
fprintf('Random initialation of weights.\n');

cy = 350;
me=500;
im=1.05;
dm=0.7;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Allow user to change default parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choice = input('Are you satisfied with these selections? [y,n]:','s');
if choice == 'n'
    lr = input('Enter learning rate: ');
    cy = input('Enter standard backprop training epochs: ');
    mc = input('Enter momentum: ');
    eg = input('Output error tolerance (rms): ');
    df = input('How often to display errors (# of epochs): ');
    me = input('Maximum number of epochs to train: ');
    er = input('Error Ratio: ');
    im = input('Enter learning rate increase: ');
    dm = input('Enter learning rate decrease: ');
end
tp = [df me eg lr im dm mc er cy];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Printout network parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if robtrn == 'y'                % create robust training set if desired
[x,y]=aarobust(x,y);
end

[nov1, numin] = size(x);      % get size of input training matrix
[nov2,numout] = size(y);      % get size of output training matrix

if nov1 ~= nov2
error('The number of input vectors and target vectors has to be the same.')
end

[nov1t,numint] = size(xt);    % get size of input testing matrix
[nov2t,numoutt] = size(yt);   % get size of output testing matrix

if nov1t ~= nov2t

```

```

        error('The number of input vectors (xt) and target vectors (yt) has to be the
same.')
end

```

```

fprintf('\nThis network has:\n\n');
fprintf(' %0.f input neurons\n',numin);
fprintf(' %0.f neurons in the 1st hidden layer\n',numhid1);
fprintf(' %0.f neurons in the 2nd hidden layer\n',numhid2);
fprintf(' %0.f neurons in the 3rd hidden layer\n',numhid1);
fprintf(' %0.f output neurons\n\n',numout);
fprintf('There are %0.f input/output pairs in this training set.\n\n',nov1);
fprintf('There are %0.f input/output pairs in this testing set.\n\n',nov1t);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Linear Scale input and Output.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

[xntrain,xm,xs] = scale(x);           % linear scale training data
[yntrain] = scale(y,xm,xs);
p = xntrain';
t = yntrain';
[xntest,xmt,xst] = scale(xt);         % linear scale testing data
[yntest] = scale(yt,xmt,xst);
pt = xntest';
tt = yntest';

```

```

f1='logsig';           % specify activation function type for each layer
f2='logsig';
f3='logsig';
f4='purelin';

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Initialize weight vectors to small values using function initff.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

[W1,B1,W2,B2,W3,B3,W4,B4]=initff(p,numhid1,f1,numhid2,f2,numhid3,f3,t,f4);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   First Standard Backprop Training To Reduce the Error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

[W1,B1,W2,B2,W3,B3,W4,B4,epochs,tr,sse_train,sse_test] = ...
cclstbp4(W1,B1,f1,W2,B2,f2,W3,B3,f3,W4,B4,f4,p,t,pt,tt,tp);

```

```

if seltrn == 'y'                                % if selected, solve for linear output weights

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Train the BP/SVD neural network.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
df3 = feval(f3,'delta');
df4 = feval(f4,'delta');

dw1 = W1*0;
db1 = B1*0;
dw2 = W2*0;
db2 = B2*0;
dw3 = W3*0;
db3 = B3*0;
dw4 = W4*0;
db4 = B4*0;
MC = 0;

% PRESENTATION PHASE (Training)
a1 = feval(f1,W1*p,B1);
a2 = feval(f2,W2*a1,B2);
a3 = feval(f3,W3*a2,B3);
a4 = feval(f4,W4*a3,B4);
e = t-a4;
SSE = sumsqr(e);

% PRESENTATION PHASE (Testing)
a1tst = feval(f1,W1*pt,B1);
a2tst = feval(f2,W2*a1tst,B2);
a3tst = feval(f3,W3*a2tst,B3);
a4tst = feval(f4,W4*a3tst,B4);
etst = tt-a4tst;
SSET = sumsqr(etst);

% TRAINING RECORD
tr = zeros(2,me+1);
sse_train = zeros(1,me+1);
sse_test = zeros(1,me+1);

tr(1:2,1) = [SSE; lr];
sse_train(1) = [SSE];

```

```

sse_test(1) = [SSET];

% PLOTTING FLAG
[r,q] = size(p);
[s,q] = size(t);
plottype = (max(r,s) == 1);

% PLOTTING
clg
message = sprintf('TRAINBPXLS: %%g%%g epochs, lr = %%g, SSE = %%g.\n',me);
fprintf(message,0,lr,SSE)
if plottype
    h = plotfa(p,t,p,a4);
else
    h = plottr(tr(1:2,1),eg);
end

metot = i;
for i=1:me

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Train the hidden layer of the network.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CHECK PHASE
if SSE < eg, i=i-1; break, end

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);
d2 = feval(df2,a2,d3,W3);
d1 = feval(df1,a1,d2,W2);

% LEARNING PHASE
[dw1,db1] = learnbpm(p,d1,lr,MC,dw1,db1);
[dw2,db2] = learnbpm(a1,d2,lr,MC,dw2,db2);
[dw3,db3] = learnbpm(a2,d3,lr,MC,dw3,db3);
MC = mc;
new_w1 = max(-10,min(+10,W1 + dw1));           % limit weight changes
new_b1 = max(-10,min(+10,B1 + db1));
new_w2 = max(-10,min(+10,W2 + dw2));
new_b2 = max(-10,min(+10,B2 + db2));
new_w3 = max(-10,min(+10,W3 + dw3));
new_b3 = max(-5,min(+5,B3 + db3));

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Train the output layer of the network.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% PRESENTATION PHASE (Training)
new_a1 = feval(f1,new_w1*p,new_b1);
new_a2 = feval(f2,new_w2*new_a1,new_b2);
new_a3 = feval(f3,new_w3*new_a2,new_b3);
[new_w4,new_b4]=trnlms(new_a3,t);           % call SVD algorithm
new_a4 = feval(f4,new_w4*new_a3,new_b4);
new_e = t-new_a4;
new_SSE = sumsqr(new_e);

```

```

% PRESENTATION PHASE (testing)
n_a1tst = feval(f1,new_w1*pt,new_b1);
n_a2tst = feval(f2,new_w2*n_a1tst,new_b2);
n_a3tst = feval(f3,new_w3*n_a2tst,new_b3);
n_a4tst = feval(f4,new_w4*n_a3tst,new_b4);
n_etst = tt-n_a4tst;
SSET = sumsqr(n_etst);

```

```

% MOMENTUM & ADAPTIVE LEARNING RATE PHASE

```

```

if new_SSE > SSE*er
    lr = lr * dm;
    MC = 0;
else
    if new_SSE < SSE
        lr = lr * im;
    end
    W1 = new_w1; B1 = new_b1; a1 = new_a1;
    W2 = new_w2; B2 = new_b2; a2 = new_a2;
    W3 = new_w3; B3 = new_b3; a3 = new_a3;
    W4 = new_w4; B4 = new_b4; a4 = new_a4;
    e = new_e; SSE = new_SSE;

```

```

% BACKPROPAGATION PHASE

```

```

d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);
d2 = feval(df2,a2,d3,W3);
d1 = feval(df1,a1,d2,W2);
end

```

```

% TRAINING RECORD

```

```

tr(:,i+1) = [SSE; lr];

```

```

sse_train(i+1) = [SSE];
sse_test(i+1) = [SSET];

% PLOTTING
if rem(i,df) == 0
    fprintf(message,i,lr,SSE)
    if plotype
        delete(h);
        h = plot(p,a4);
        drawnow
    else
        h = plottr(tr(1:2,1:(i+1)),eg,h);
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Finish Plotting.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% TRAINING RECORD
tr = tr(1:2,1:(i+1));

% PLOTTING
if rem(i,df) ~= 0
    fprintf(message,i,lr,SSE)
    if plotype
        delete(h);
        plot(p,a4);
        drawnow
    else
        plottr(tr,eg,h);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plor Cross Validation Results (MSE vs. Epochs)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MSE_train=(nonzeros(sse_train)./nov1);           % get mean squared error
MSE_test=(nonzeros(sse_test)./nov1t);

eg2 = eg/nov1;
subplot(1,2,1)

```

```

ploterr(MSE_train, eg2)
axis([0 me 10e-5 .1]);
title('Training Mean Squared Error')
ylabel('Mean Squared Error')
subplot(1,2,2)
ploterr(MSE_test, eg2)
axis([0 me 10e-5 .1]);
title('Testing Mean Squared Error')
ylabel('Mean Squared Error')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          BPX Tolerance criteria not met.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if i == me
    disp(' ')
    disp('TRAINBPX: Network error did not reach the error goal.')
    disp(' Further training may be necessary, or try different')
    disp(' initial weights and biases and/or more hidden neurons.')
    disp(' ')
    fprintf('\n*** SSE = %f \n\n', min(tr(1,:)))
else

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Tolerance criteria met by BP.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf('\n\n*** Training complete after %0.f epochs! ***\n', metot+i)
    fprintf('\n*** SSE = %f \n\n', min(tr(1,:)))
end
save weights4.mat W1 B1 W2 B2 W3 B3 W4 B4 xm xs
else
save weights4.mat W1 B1 W2 B2 W3 B3 W4 B4 xm xs
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function [w1,b1,w2,b2,w3,b3,w4,b4,i,tr,sse_train,sse_test]=...
cclstbp4(w1,b1,f1,w2,b2,f2,w3,b3,f3,w4,b4,f4,p,t,pt,tt,tp)

%CCLSTBP4 Train 4-layer (3 hidden) feed-forward network w/fast backpropagation.
%      using Cross Validation Training
%      [W1,B1,W2,B2,W3,B3,W4,B4,TE,TR] =
%      CCLSTBP4(W1,B2,F1,W1,B1,F2,W3,B3,F3,W4,B4,F4,P,T,TP)
%      Wi - Weight matrix for the ith layer.
%      Bi - Bias vector for the ith layer.
%      Fi - Transfer function (string) for the ith layer.
%      P - RxQ matrix of input training vectors.
%      T - SxQ matrix of target training vectors.
%      PT - RxQ matrix of input testing vectors.
%      TT - SxQ matrix of target testing vectors.
%      TP - Training parameters (optional).
% Returns:
%      Wi - new weights.
%      Bi - new biases.
%      TE - the actual number of epochs trained.
%      TR - training record: [row of errors]
%
% Training parameters are:
%      TP(1) - Epochs between updating display, default = 25.
%      TP(2) - Maximum number of epochs to train, default = 1000.
%      TP(3) - Sum-squared error goal, default = 0.02.
%      TP(4) - Learning rate, 0.01.
%      TP(5) - Learning rate increase, default = 1.05.
%      TP(6) - Learning rate decrease, default = 0.7.
%      TP(7) - Momentum constant, default = 0.9.
%      TP(8) - Maximum error ratio, default = 1.04.
%      TP(9) - Number of initial epochs for standard training = 300.
%      Missing parameters and NaN's are replaced with defaults.
%
% Matlab Neural Networks toolbox code modified for a three hidden layer network
% and Cross Validation Training.
% Also modified for limited weight change per iteration.

if nargin < 15,error('Not enough arguments.');
```

```

end

% TRAINING PARAMETERS
if nargin == 13, tp = []; end
tp = nndef(tp,[25 1000 0.02 0.01 1.05 0.7 0.9 1.04 300]);
df = tp(1);
me = tp(2);

```

```

eg = tp(3);
lr = tp(4);
im = tp(5);
dm = tp(6);
mc = tp(7);
er = tp(8);
cy = tp(9);
me = cy;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
df3 = feval(f3,'delta');
df4 = feval(f4,'delta');
dw1 = w1*0;
db1 = b1*0;
dw2 = w2*0;
db2 = b2*0;
dw3 = w3*0;
db3 = b3*0;
dw4 = w4*0;
db4 = b4*0;
MC = 0;

```

```

% PRESENTATION PHASE (Training)

```

```

a1 = feval(f1,W1*p,B1);
a2 = feval(f2,W2*a1,B2);
a3 = feval(f3,W3*a2,B3);
a4 = feval(f4,W4*a3,B4);
e = t-a4;
SSE = sumsqr(e);

```

```

% PRESENTATION PHASE (Testing)

```

```

a1tst = feval(f1,W1*pt,B1);
a2tst = feval(f2,W2*a1tst,B2);
a3tst = feval(f3,W3*a2tst,B3);
a4tst = feval(f4,W4*a3tst,B4);
etst = tt-a4tst;
SSET = sumsqr(etst);

```

```

% TRAINING RECORD

```

```

tr = zeros(2,me+1);
sse_train = zeros(1,me+1);

```

```

sse_test = zeros(1,me+1);

tr(1:2,1) = [SSE; lr];
sse_train(1) = [SSE];
sse_test(1) = [SSET];

% PLOTTING FLAG
[r,q] = size(p);
[s,q] = size(t);
plottype = (max(r,s) == 1);

% PLOTTING
clg
message = sprintf('TRAINBPXLS: %%g/%%g epochs, lr = %%g, SSE = %%g.\n',me);
fprintf(message,0,lr,SSE)
if plottype
    h = plotfa(p,t,p,a4);
else
    h = plottr(tr(1:2,1),eg);
end

for i=1:me

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Train the hidden layer of the network.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CHECK PHASE
if SSE < eg, i=i-1; break, end

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);
d2 = feval(df2,a2,d3,W3);
d1 = feval(df1,a1,d2,W2);

% LEARNING PHASE
[dw1,db1] = learnbpm(p,d1,lr,MC,dw1,db1);
[dw2,db2] = learnbpm(a1,d2,lr,MC,dw2,db2);
[dw3,db3] = learnbpm(a2,d3,lr,MC,dw3,db3);
[dw4,db4] = learnbpm(a3,d4,lr,MC,dw4,db4)
MC = mc;
new_w1 = max(-10,min(+10,W1 + dw1));
new_b1 = max(-10,min(+10,B1 + db1));

```

% limit weight changes

```

new_w2 = max(-10,min(+10,W2 + dw2));
new_b2 = max(-10,min(+10,B2 + db2));
new_w3 = max(-10,min(+10,W3 + dw3));
new_b3 = max(-5,min(+5,B3 + db3));
new_w4 = max(-10,min(+10,W4 + dw4));
new_b4 = max(-5,min(+5,B4 + db4));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Train the output layer of the network.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% PRESENTATION PHASE (Training)
new_a1 = feval(f1,new_w1*p,new_b1);
new_a2 = feval(f2,new_w2*new_a1,new_b2);
new_a3 = feval(f3,new_w3*new_a2,new_b3);
new_a4 = feval(f4,new_w4*new_a3,new_b4);
new_e = t-new_a4;
new_SSE = sumsqr(new_e);

% PRESENTATION PHASE (testing)
n_a1tst = feval(f1,new_w1*pt,new_b1);
n_a2tst = feval(f2,new_w2*n_a1tst,new_b2);
n_a3tst = feval(f3,new_w3*n_a2tst,new_b3);
n_a4tst = feval(f4,new_w4*n_a3tst,new_b4);
n_etst = tt-n_a4tst;
SSET = sumsqr(n_etst);

% MOMENTUM & ADAPTIVE LEARNING RATE PHASE
if new_SSE > SSE*er
    lr = lr * dm;
    MC = 0;
else
    if new_SSE < SSE
        lr = lr * im;
    end
    W1 = new_w1; B1 = new_b1; a1 = new_a1;
    W2 = new_w2; B2 = new_b2; a2 = new_a2;
    W3 = new_w3; B3 = new_b3; a3 = new_a3;
    W4 = new_w4; B4 = new_b4; a4 = new_a4;
    e = new_e; SSE = new_SSE;

% BACKPROPAGATION PHASE
d4 = feval(df4,a4,e);
d3 = feval(df3,a3,d4,W4);

```

```

    d2 = feval(df2,a2,d3,W3);
    d1 = feval(df1,a1,d2,W2);
end

% TRAINING RECORD
tr(:,i+1) = [SSE; lr];
sse_train(i+1) = [SSE];
sse_test(i+1) = [SSET];

% PLOTTING
if rem(i,df) == 0
    fprintf(message,i,lr,SSE)
    if plottype
        delete(h);
        h = plot(p,a4);
        drawnow
    else
        h = plottr(tr(1:2,1:(i+1)),eg,h);
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Finish Plotting.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% TRAINING RECORD
tr = tr(1:2,1:(i+1));

% PLOTTING
if rem(i,df) ~= 0
    fprintf(message,i,lr,SSE)
    if plottype
        delete(h);
        plot(p,a4);
        drawnow
    else
        plottr(tr,eg,h);
    end
end

% WARNINGS
if SSE > eg
    disp(' ')

```

```

disp('TRAINBPX: Network error did not reach the error goal.')
disp(' Training will resume with Least Squares Error (SVD) in the')
disp(' linear output layer and standard backprop in the three')
disp(' hidden layers. ')
disp(' ')
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function [w1,b1,w2,b2,w3,b3,w4,b4] = initff(p,s1,f1,s2,f2,s3,f3,s4,f4)

%INITFF Initialize feed-forward network up to 4 layers.
%
%      [W1,B1,...] = INITFF(P,S1,'F1',...,Sn,'Fn')
%      P - Rx2 matrix of input vectors.
%      Si - Size of ith layer.
%      Fi - Transfer function of the ith layer (string).
%      Returns:
%      Wi - Weight matrix of the ith layer.
%      Bi - Bias (column) vector of the ith layer.
%
%      [W1,B1,...] = INITFF(P,S1,'F1',...,T,'Fn')
%      T - SnxQ matrix of target vectors.
%      Returns weights and biases.
%
%      IMPORTANT: Each ith row of P must contain expected
%                  min and max values for the ith input.
%
%      EXAMPLE: [W1,b1,W2,b2] = initff([0 10; -5 5],5,'tansig',3,'purelin');
%              p = [4; -1];
%              a = simuff(p,W1,b1,'tansig',W2,b2,'purelin')
%
%      See also NNINIT, BACKPROP, SIMUFF, TRAINBPX, TRAINLM
%
%      MATLAB code modified for a 4 layer network.

if all([3 5 7 9] ~= nargin),error('Wrong number of input arguments'),end

[R,Q] = size(p);
if (Q < 2), error('P must contain at least two elements in each row.'),end

if nargin == 3
    [S,Q] = size(s1);
    if max(S,Q) > 1, s1 = S; end
    [w1,b1] = feval(feval(f1,'init'),s1,p);

elseif nargin == 5
    [w1,b1] = feval(feval(f1,'init'),s1,p);
    x = ones(s1,1) * feval(f1,'output');
    [S,Q] = size(s2);
    if max(S,Q) > 1, s2 = S; end
    [w2,b2] = feval(feval(f2,'init'),s2,x);

```

```

elseif nargin == 7
    [w1,b1] = feval(feval(f1,'init'),s1,p);
    x = ones(s1,1) * feval(f1,'output');
    [w2,b2] = feval(feval(f2,'init'),s2,x);
    x = ones(s2,1) * feval(f2,'output');
    [S,Q] = size(s3);
    if max(S,Q) > 1, s3 = S; end
    [w3,b3] = feval(feval(f3,'init'),s3,x);

elseif nargin == 9
    [w1,b1] = feval(feval(f1,'init'),s1,p);
    x = ones(s1,1) * feval(f1,'output');
    [w2,b2] = feval(feval(f2,'init'),s2,x);
    x = ones(s2,1) * feval(f2,'output');
    [w3,b3] = feval(feval(f3,'init'),s3,x);
    x = ones(s3,1) * feval(f3,'output');
    [S,Q] = size(s4);
    if max(S,Q) > 1, s4 = S; end
    [w4,b4] = feval(feval(f4,'init'),s4,x);
end

```

```

function [y,slope,int]=scale(x,m,b)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCTION:[x,slope,int]=scale(x,m,b)
%
% Linearly scale the data between 0.1 and 0.9
%
%      y = m*x + b
%
%      x = data
%      m = slope
%      b = y intercept
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 1,error('Not enough input arguments.');
```

end

```

[nrows,ncols]=size(x);      % determine size of input matrix x.

if nargin == 1              % if data is the only argument.
    delta = max(x)-min(x);   % calculate the range of input data.
    slope = .8./delta;       % calculate the slope.
    int = .1 - slope.*min(x); % calculate the intercept.
else
    slope = m;               % use previously calculated m & b
    int = b;                 % sent to function.
end

y = (ones(nrows,1)*slope).*x + ones(nrows,1)*int; %scale data.

```

```

function [g]=goal(x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FUNCTION: [g]=goal(x)
%
%   Determines the sum squared error goal through filtering/statistical
%   means. Input is training set matrix, output is recommended SSE goal.
%   Uses a median filtering technique (zero phase shift) with a window size of 6.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nr,nc]=size(x);
[x,xm,xb]=scale(x);
xf=medfilt1(x,6);
err=(std(x(10:nr-10,:)-xf(10:nr-10,:))).^2;
g=sum(err*nr);

```

```
function [w,b]=trnlms(x,y)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FUNCTION: [w,b]=trnlms(x,y)
%
%   Performs a least means square estimate of the weight matrix using a singular
%   valued decomposition of the input matrix x (from the last hidden layer), and
%   the matrix of targets y.
%   Returns the output layer weight matrix (w) and a bias vector (b).
%   The input matrix is padded with a bias matrix.
%   Noise is eliminated in the input set by the parameter e.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if nargin < 2,error('Not enough arguments. '),end
```

```
if nargout <= 1
```

```
    w= y/x;
```

```
else
```

```
    x=x';
```

```
    y=y';
```

```
    [r,c]=size(x);
```

```
    x=[x ones(r,1)];
```

```
% create input bias vector of ones
```

```
    e=3.75e-3;
```

```
% error multiplier
```

```
    [u,s,v]=svd(x);
```

```
% singular value decomposition of input vectors
```

```
    s(find(s<e))=0*s(find(s<e));
```

```
% find singular values less than e and set to zero
```

```
    s(find(s>e))=1./s(find(s>e));
```

```
% take inverse of s matrix of singular values
```

```
    s=s';
```

```
    m=v*s*u'*y;
```

```
% reconstruct input matrix
```

```
    w=m(1:c,:);
```

```
% extract weight matrix
```

```
    b=m(c+1,:);
```

```
% extract bias vector
```

```
    w=w';
```

```
    b=b';
```

```
end
```

```

function [xout]=reduce(xin,f)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCTION reduce : [xout]=reduce(xin,f)
% Used to reduce a training set by a factor f .
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nr,nc]=size(xin);
j=1;
for i=1:nr
    if rem(i,f) == 0
        xout(j,:) = xin(i,:);
        j=j+1;
    end
end
end

```

```

function [outx,outy]=aarobust(x,y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FUNCTION: [outx,outy]=aarobust(x,y)
% AAROBUST
% Trains an autoassociative network using faults in the input set
% and error free data in the output set. Random fault values
% are placed on the diagonals of the input matrix of up to 10% of the current value.
% The number of training patterns are doubled (original set plus fault set of same size).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2,error('Not enough input arguments.');
```

<pre> end [nr,nc]=size(x); j=0; for i=1:nr j=j+1; rand('seed',sum(100*clock)); x(i,j)=x(i,j)+x(i,j).*(rand(1)-0.5)/5; if j==nc j=0; end end end outx=[x;xorig]; outy=[y;xorig]; end</pre>	<pre> % Determine size of input data set. % For row i, column j, add random fault. % Set new random seed % Random fault of +/-10%. % If at end of column reset to first column. % Append fault input data to original data set. % Append output data set to itself.</pre>
---	---

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      SENSITI4                                                    %
%                                                                    %
%      m-file that does a sensitivity analysis on a four layer      %
%      autoassociative neural network.  Scaling data should also be %
%      stored in the weight file.                                    %
%                                                                    %
%      Perturbs each input vector by 5%. Plots change in output 2 ways: %
%      1.) Total change due to all inputs                            %
%      2.) Normalized change showing contribution of each input to change. %
%                                                                    %
%      Weights: W1, W2, W3, W4                                       %
%      Biases: B1, B2, B3, B4                                         %
%      Scaling: xm, xs                                                %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

vecfile = input('Enter filename of input/output data vectors: ','s');
wgtfile = input('Enter filename of weight and bias matrices: ','s');
sel1 = input('Sigmoid, Tansigmoid, or Linear 1st hidden layer? [s,t,l]: ','s');
sel2 = input('Sigmoid, Tansigmoid, or Linear 2nd hidden layer? [s,t,l]: ','s');
sel3 = input('Sigmoid, Tansigmoid, or Linear 3rd hidden layer? [s,t,l]: ','s');
sel4 = input('Sigmoid, Tansigmoid, or Linear output layer? [s,t,l]: ','s');
sc = input('Input and Output Scaling Type MCUV=m, linear=l, none=n ?[m,l,n]:','s');

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Load training data                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

eval(['load ',vecfile])
eval(['load ',wgtfile])
[ nov1, numin] = size(x);
[ nov2,numout] = size(y);

```

```

if nov1 ~= nov2
    error('The number of input vectors and target vectors has to be the same.')
end

```

```

[numhid,numin1]=size(W1);
[nr,nc]=size(W3);
[numout1,numhid1]=size(W4);

```

```

if numin1 ~= numin | numout1 ~= numout | numhid1 ~= numhid
    error('The weight matrix dimensions do not match the training data.')

```

```

end

if nov1 ~= nov2
    error('The number of input vectors and target vectors has to be the same.')
end

fprintf('\nThis network has:\n\n');
fprintf(' %0.f input neurons\n',numin);
fprintf(' %0.f neurons in the 1st and 3rd hidden layers\n',numhid);
fprintf(' %0.f neurons in the 2nd "bottleneck" hidden layer\n',nc);
fprintf(' %0.f output neurons\n\n',numout);
fprintf('There are %0.f input/output pairs in this training set.\n\n',nov1);

%%%%%%%%%%%%%%
%      Set up Simulation Parameters
%%%%%%%%%%%%%%

if sel1 == 't'                % First layer selection
    F1 = 'tansig';
elseif sel1 == 'l'
    F1 = 'purelin';
else
    F1 = 'logsig';
end

if sel2 == 't'                % Second layer selection
    F2 = 'tansig';
elseif sel2 == 'l'
    F2 = 'purelin';
else
    F2 = 'logsig';
end

if sel3 == 't'                % Third layer selection
    F3 = 'tansig';
elseif sel3 == 'l'
    F3 = 'purelin';
else
    F3 = 'logsig';
end

if sel4 == 't'                % Fourth (Output) layer selection
    F4 = 'tansig';
elseif sel4 == 'l'

```

```

        F4 = 'purelin';
    else
        F4 = 'logsig';
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Perform sensitivity analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m=zeros(numout,numin);

for i=1:numin
    X=x;
    y=x;
    X(:,i)=X(:,i).*0.95;          % Comment these for normal error.
    if sc == 'm'                  % Perturb input by 5%
        [X] = mcuv(X,xm,xs);
    elseif sc == 'l'
        [X] = scale(X,xm,xs);
    else
        X=X;
    end
    output=simuff(X,W1,B1,F1,W2,B2,F2,W3,B3,F3,W4,B4,F4);
    if sc == 'm'
        Y = (ones(nov1,1)*xs).*output + ones(nov1,1)*xm;
    elseif sc == 'l'
        Y = (output-(ones(nov1,1)*xs))./(ones(nov1,1)*xm);
    else
        Y=output;
    end
    m(:,i)=(sum(abs((y-Y)./y))./length(y))*100; % Error between input and output
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Normalize Matrix for Plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

s=sum(m);
sm=ones(numin,1)*s;
nm=m./sm;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Plot snsitivities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

subplot(2,1,1)
bar(cumsum(m))
title('Sensitivity Analysis of Neural Network')
ylabel('Sum of % Change')
axis([0 numin+1 0 inf]);

```

```

subplot(2,1,2)
bar(cumsum(nm))
xlabel('Number of Input Signals')
ylabel('Normalized Change')
axis([0 numin+1 0 1.5]);
end

```

```

function [a1,a2,a3,a4] = simuff(p,w1,b1,f1,w2,b2,f2,w3,b3,f3,w4,b4,f4)

%SIMUFF Simulate a 4 layer feed-forward network.
%
%    SIMUFF will simulate networks with up to 4 layers.
%
%    SIMUFF(P,W1,B1,'F1',...,Wn,Bn,'Fn')
%    P - Matrix of input (column) vectors.
%    Wi - Weight matrix of the ith layer.
%    Bi - Bias (column) vector of the ith layer.
%    Fi - Transfer function of the ith layer (string).
%    Returns output of nth layer.
%
%    [A1,A2,...] = SIMUFF(P,W1,B1,'F1',...,Wn,Bn,'Fn')
%    Returns:
%    Ai - Output of the ith layer.
%
%    EXAMPLE: [w1,b1,w2,b2] = initff([0 10; -5 5],3,'tansig',2,'purelin');
%            p = [2; -3];
%            a = simuff(p,w1,b1,'tansig',w2,b2,'purelin')
%
%    See also NNSIM, BACKPROP, INITFF, TRAINBPX, TRAINLM.
%
%    MATLAB code modified for up to a 4 layer network

if all([4 7 10 13] ~= nargin),error('Wrong number of input arguments'),end

if nargin == 4
    a1 = feval(f1,w1*p,b1);

elseif nargin == 7
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);
    if nargout <= 1, a1 = a2; end

elseif nargin == 10
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);
    a3 = feval(f3,w3*a2,b3);
    if nargout <= 1, a1 = a3; end

elseif nargin == 13
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);

```

```
a3 = feval(f3,w3*a2,b3);  
a4 = feval(f4,w4*a3,b4);  
if nargout <= 1, a1 = a4; end  
end
```

Vita

Darryl James Wrest graduated from the University of Missouri at Columbia in 1991 with a Bachelor of Science in Electrical Engineering. From 1991 through 1993, he was employed by an electrical contracting firm as an electrical cost estimator and engineer. Mr. Wrest then returned to college in 1993 where he earned a Bachelor of Science degree in Nuclear Engineering with honors from the University of Missouri at Rolla in 1994. He worked the summer of 1994 at Oak Ridge National Laboratories as a research intern in the chemical technology division. He returned to the University of Missouri at Rolla in the fall of 1994 to begin work on a Masters degree in Nuclear Engineering. In 1995, he transferred to the University of Tennessee's Nuclear Engineering department where he was offered an Institute of Nuclear Power Operation (INPO) Fellowship. Mr. Wrest was awarded a Master of Science degree in Nuclear Engineering in 1996.