To the Graduate Council:

I am submitting herewith a dissertation written by Andrew E. Farell entitled "System Cultivation: Process Monitoring and Control Using the Historical Database." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Chemical Engineering.

Charles F. Moore, Major Professor

We have read this dissertation
and recommend its acceptance:

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

SYSTEM CULTIVATION: PROCESS MONITORING AND CONTROL USING THE HISTORICAL DATABASE

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Andrew E. Farell

December 1990

DEDICATION

To my parents for their constant love and support, and to my wife, Teresa.

ACKNOWLEDGMENTS

The author would like to thank his major professor, Dr. C.F. Moore for his guidance in this work. The author appreciates the support of the rest of his committee: Dr. J.J. Perona, Dr. J.J. Downs, Dr. K.K. Kirby, Dr. D.D. Bruns, Dr. D. Brzakovic.

The author would also like to thank his fellow graduate students, Suzanne Roat, Dave Canter, Vera Williams, Rich Neubert, and Steve Prout, for their assistance and friendship. Special thanks are extended to Dr. James E. Hackney for sharing his expertise in many areas, the basketball court included.

Finally the author wishes to acknowledge the Measurement and Control Engineering Center and E.I. Dupont de Nemoirs and Co. for funding of this work.

ABSTRACT

Process computers are playing an increasingly important role in the chemical industry. They not only provide a system in which to implement modern control concepts but also a framework in which large amounts of process data can be collected and stored cheaply and efficiently. Today, at the fingertips of process engineers lies a vast amount of historical data, containing a wealth of information about the operation of the process. Unfortunately the engineer is often overwhelmed by the volume of data. Subtleties which could be helpful in improving the operation of the process systems can be easily lost in the vastness of numbers and records.

A recent perspective in process control, system cultivation focuses on the day-to-day analysis and evaluation of the operation of the process and its control system. As today's processes become more complex and the control systems for these processes become more sophisticated, this perspective is becoming increasingly important. The often fragile nature of these complex designs demands the ability to monitor, detect, and diagnose problems quickly. System cultivation is also concerned with studying the normal operation of the process for making continual improvements. There should always be room for improvement. Analyzing the historical data base properly will provide clues and directions which would improve even the normal conditions of operations.

This dissertation works towards three objectives. First, it refines two existing cultivation tools: hypothesis feedback modeling and time-shifted dis-

tribution analysis. Second, it introduces pattern recognition and neural networks to the cultivation framework. Finally, the primary objective develops a framework that incorporates several cultivation tools for effectively detecting and identifying subtle operational changes. Two simulated unit operations, a heat exchanger and a continuous stirred-tank reactor, aid in developing and demonstrating these cultivation concepts.

TABLE OF CONTENTS

CHAPTER		PAGE
1. Introduction		**1**
2. Statistical Process Control		**5**
3. Hypothesis Feedback Modeling		**8**
3.1	Introduction	8
3.2	Step-by-Step HF-modeling Analysis	13
3.2.1	HF-model definition	13
3.2.2	Defining resolution of each variable	15
3.2.3	Sorting into the reduced database	17
3.2.4	Multivariable statistical plots and relationship plots	17
3.2.5	Model refinement	24
3.3	Summary of HF-modeling	27
4. Time-Shifted Distribution Analysis		**29**
4.1	Description of the TSD Analysis	30
4.2	Explanation of the TSD Plot	30
4.3	TSD analysis of a PID controller	36
4.4	MIMO TSD	40
5. Pattern Recognition		**41**

5.1 Outline of Pattern Recognition 42
5.2 Classical Pattern Classification 45
5.3 Neural Networks 47
5.4 Neural Network Architecture 49
5.4.1 The learning rule 51
5.4.2 Nodal connections 54
5.4.3 Nodal transfer functions 63
5.5 Multilayer Perceptron Network with Back-propagation Learning 64
5.5.1 Decision boundary studies of a multilayer perceptron network 68
5.6 Other neural networks of interest 81

6. Enhancing System Cultivation Methods with Neural Networks 84

6.1 Neural Networks with HF-Modeling 86
6.2 Presenting Data to the Network 89
6.2.1 Selecting statistics presented to the network 90
6.2.2 Incorporating global parameters 91
6.2.3 Scaling the reduced database 92
6.2.3.1 Physical scaling 92
6.2.3.2 Normalization 94
6.2.3.3 Basecase normalized scaling 94
6.3 Problem Recognition Framework 103

7. Example Applications 106

7.1 Heat Exchanger 106

7.1.1 The system studied 107

7.1.2 HF-modeling analysis of the Heat Exchanger 117

7.1.2.1 HF-model 117

7.1.2.2 Defining resolution 117

7.1.2.3 Sorting into the reduced database 118

7.1.2.4 Refining the HF-model 119

7.1.2.5 Reporters 127

7.1.2.6 Decision tree 130

7.1.2.7 Shortcomings of decision tree analysis 132

7.1.3 Analyzing historical data directly with neural networks 133

7.1.3.1 Neural network performance 136

7.1.4 Analyzing cultivated data with neural networks 139

7.2 Reactor Example 146

7.2.1 The system studied 146

7.2.2 HF-modeling of the CSTR 155

7.2.2.1 Typical data analysis 158

7.2.3 Outline of problem recognition framework 167

7.2.4 Scaling the reduced database 171

7.2.5 Problem recognition framework applied to CSTR data . 174

7.2.5.1 The neural network 174

7.2.5.2 Training the network 175

7.2.5.3 Network performance studies 176
7.2.5.4 A comparison of scaling techniques 181
7.2.5.5 Neural network vs. one-nearest prototype classifier 183
7.2.5.6 Classifying normal operation and identifying problem magnitude 188
7.2.5.7 Indistinguishable system problems 194
7.2.6 TSD analysis of the CSTR 197
7.2.7 Summary of TSD results 198
7.2.8 Neural network interpretation of TSD plots 205
7.2.8.1 Presenting TSD information to the neural network and scaling 205
7.2.8.2 Neural network training 206
7.2.8.3 Neural network performance 207

8. Conclusions and Recommendations 211
8.1 Conclusions 211
8.1.1 HF-modeling 211
8.1.2 TSD analysis 212
8.1.3 Pattern recognition and neural networks 212
8.1.4 Problem recognition framework 213
8.2 Recommendations 214

LIST OF REFERENCES 216

VITA 223

LIST OF FIGURES

FIGURE		PAGE
2.1	Shewhart control chart	6
3.1	Cross section of viewing space	11
3.2	Dividing the viewing space into cubic regions of constant operation	16
3.3	Example reduced database for heat exchanger	18
3.4	Multivariable statistical plot	20
3.5	Multivariable statistical plot showing data compared with base-case operation	21
3.6	Relationship plot	23
3.7	Generic System	28
4.1	Overview of TSD analysis	31
4.2	Phase plot: Controller output vs error	33
4.3	Typical SISO TSD plots	35
4.4	SISO TSD plot (using ΔU)	38
4.5	SISO TSD plot (using $U^*(k)$)	39
5.1	Overview of pattern recognition analysis	43
5.2	Nonlinear element	50
5.3	Possible processing element connections	55
5.4	Single element decision boundaries	57
5.5	Decision boundary for three first-layer nodes	58

5.6 Decision boundaries for single second-layer node: piecewise linear boundary 59
5.7 Two-layer,two-input network forms two piecewise linear regions 61
5.8 Three-layer, one-output network creates any region bounded by lines A,B, and C 62
5.9 Multilayer-perceptron network 66
5.10 Decision boundary determination 70
5.11 Decision boundary for multiple prototypes 71
5.12 Decision boundary for an unequal number of prototypes 72
5.13 Nonlinear decision boundary 74
5.14 Nonlinear decision boundary- tight convergence constraints. . . 75
5.15 Decision boundaries for overlapping clusters 76
5.16 Decision boundaries for overlapping clusters- tight convergence constraints 77
5.17 Linearly separable disjoint regions 78
5.18 Decision boundaries resulting from encircled data 79
6.1 Overview of cultivation framework for diagnosing system problems 85
6.2 Neural network training 87
6.3 Cluster of basecase data 95
6.4 Shifting of cluster due to small process problem 97
6.5 Clusters that exemplify a problem with normalized scaling . . . 98
6.6 Basecase normalized scaling 100
6.7 Physically scaled data 101

6.8 Data after subtracting from basecase 102
6.9 Data after basecase normalized scaling 104
7.1 Shell and tube heat exchanger 108
7.2 Basecase heat exchanger specifications 109
7.3 Flow diagram of simulation 111
7.4 Flow diagram of feedback system 112
7.5 Operating conditions (12 hour span) 113
7.6 Operating conditions (72 hour span) 114
7.7 Manipulated variable profiles 115
7.8 Output from simulation 116
7.9 Example of reduced information database from heat exchanger 120
7.10 Sorting historical data into the reduced data base. 121
7.11 Overall statistical plot from heat exchanger data 122
7.12 Overall statisical plot from heat exchanger data 123
7.13 Example relationship plot form heat exchanger 124
7.14 Viewing space cross-section showing effect of ambient temperature on the system 126
7.15 Reporter from statistical plots for all system problems studied . 128
7.16 Reporters from relationship plots for all system problems studied 129
7.17 Decision tree for shell-and-tube heat exchanger 131
7.18 Example network classification of a single record 135
7.19 Summary of two-class example 137
7.20 Summary of three-class example 138

7.21 Summary of four-class example 140
7.22 Summary of network classification of data with same magnitude as training data 143
7.23 Classifying system problems of varying magnitudes 144
7.24 Classifying multiple problems 145
7.25 Continuous stirred-tank reactor 147
7.26 Differential equations describing reactor 149
7.27 Definition of variables 150
7.28 Summary of important reaction parameters 151
7.29 Typical disturbance profiles 152
7.30 Typical disturbance profiles 153
7.31 Typical manipulated and controlled variable responses 154
7.32 Example reduced database from CSTR 157
7.33 Multivariable statistical plot of F_{cw} for normal operation 159
7.34 Multivariable statistical plot of $F_r * T_r$ for normal operation . . 160
7.35 Relationship plot from normal operation 161
7.36 Shewhart control for data under consideration 162
7.37 Multivariable statistical plot of F_{cw} for data under analysis . . 164
7.38 Multivariable statistical plot of $F_r * T_r$ 165
7.39 Relationship plot 166
7.40 Summary of HF-modeling for all system problems 168
7.41 Problem recognition framework 169
7.42 Scaling example 173

7.43 Classification of 'new' data 177
7.44 Classification of system problems (about one-half training magnitude) 179
7.45 Classification of problems (various magnitudes) 180
7.46 Classifying multiple system problems 182
7.47 Classification with normalized scaling (same mag as training data)184
7.48 Classification with normalized scaling (one-half mag of training data) 185
7.49 1-NP classification 187
7.50 Problem magnitude vs dist from basecase for change in catalyst activity 189
7.51 Problem magnitude vs distance from basecase for $\Delta C_{p_{chill}}$ . . . 190
7.52 Problem magnitude vs distance from basecase for $\Delta C_{p_{feed}}$ . . . 191
7.53 Average distances between training data sets 195
7.54 Response of a well-tuned controller to a step in reactant flow rate along with TSD plot: K_c=12.0, τ_I=800 199
7.55 Step response along with TSD plot; $Kc = 5.0$, $\tau_I = 800$ 201
7.56 Step response along with TSD plot; $Kc = 1.0$, $\tau_I = 800$ 202
7.57 Step response along with TSD plot; $Kc = 5.0$,$\tau_I = 200$ 203
7.58 Step response along with TSD plot; $Kc = 1.0$,$\tau_I = 200$ 204
7.59 Selected examples of identifying controller performance with TSD-neural network classifier 209

7.60 Selected examples of identifying controller performance with TSD-neural network classifier 210

CHAPTER 1

Introduction

Over the past twenty years, the chemical industry has made a concerted effort to streamline operations. Before the mid-1960s the chemical industry was immune, for the most part, to overall economic swings, and the emphasis was simply to make as much product as possible. Efficiency was not a concern as long as reasonable product was produced. Today, highly competitive world markets have made efficiency and consistency essential.

Even today, for most chemical processes, correcting poor operation takes precedence over redesigning processes and control systems that already work reasonably well. Even though many chemical processes have been around for years and engineers have acquired much experience from them, many operational problems and inefficiencies go undiagnosed for prolonged periods of time. For example, a heat exchanger that becomes fouled over a period of time may go unnoticed because it has no effect on the final product, yet the incremental increase in steam needed to adjust for fouling costs a significant amount of money. System problems like this one need to be monitored, detected, and diagnosed on a daily basis.

Most system problems and inefficiencies are presently identified by methods such as examining strip chart recordings, looking at individual sensor

statistics, or simply using intuition gained through years of experience. Such techniques are inefficient for two reasons. First, they are unreliable and time consuming; the multivariable and highly interacting nature of chemical systems makes it extremely difficult for anyone to consider all of the variables and possibilities. Second, the knowledge that a highly experienced person acquires about a process is seldom passed efficiently, if at all, to his successor. System cultivation is a relatively new perspective in process control concerned with these issues.

The recent growth of system cultivation has resulted from a desire to develop techniques employing the computational power of process computers. Computers provide not only a system in which large amounts of data can be stored cheaply and efficiently but also a computational machine for systematically examining multivariable relationships in the historical database. In general terms, cultivation's concern is the day-to-day analysis and evaluation of the operation of processes and their control structures. It supplies the engineer with tools for systematically detecting and diagnosing system problems and inefficiencies and also for continually improving the normal operation of a process.

Even with the aid of high speed computers, the task of sorting through the vast amounts of process data to provide meaningful information remains a difficult one. Effective frameworks, the cultivation tools, for examining the historical data from various perspectives to enhance subtle but important process relationships must be established. These tools enable us to extract information

from the large amounts of data and to gain knowledge from that information. The work described in this study focuses on the continued development of various system cultivation tools.

Two system cultivation tools that concern this study are hypothesis feedback (HF) modeling and time-shifted distribution (TSD) analysis. HF-modeling is a framework that sorts through the mountains of information to monitor process relationships providing the engineer with a much clearer picture of the process. HF-modeling has already proven to be an extremely useful tool; however, much improvement is needed to be able to apply its concepts to any process of any complexity. TSD methods help to clarify dynamic relationships in the data; they have been used in the single-input, single-output case to define controller overcompensation and undercompensation and also to help distinguish between controller modeling error and unmeasured disturbances in the Internal Model Control structure. As with HF-modeling, however, TSD analysis can be improved, particularly in applications of PID control and multi-input, multi-output processes.

The remaining cultivation tools in this study are Artificial Intelligence tools. Artificial neural networks and some of the more basic pattern recognition tools, such as clustering analysis, are applied as cultivation tools to take the information from HF-modeling and TSD analysis and translate it into process knowledge. Both neural networks and pattern recognition have proven effective in solving practical problems. They also complement the strengths of HF-modeling and TSD analysis extremely well.

When all of the tools are brought together, the contribution is a framework that can be implemented on any real process (including multi-unit processes) with minimal engineering design effort. By examining the vast amount of plant data and learning through experience, the framework will effectively detect and diagnose system problems and inefficiencies.

This dissertation is divided into eight chapters. Chapter 2 briefly recognizes statistical process control and describes how it relates to system cultivation. The cultivation tools that concern this study, HF-modeling, TSD analysis, pattern recognition, and neural networks, are outlined in chapters three through five. Chapter 6 outlines a framework in which these tools work hand in hand to detect and identify system inefficiencies. Chapter 7 details two computer simulated examples, a shell-and-tube heat exchanger and a continuous stirred-tank reactor, that demonstrate and refine these cultivation tools and the combined framework. Finally, conclusions and recommendations are presented in Chapter 8.

CHAPTER 2

Statistical Process Control

Statistical process control (SPC) is the principal tool used in discrete manufacturing to effect continual process improvement, and it is beginning to be used extensively in the continuous-process industries as well. It focuses on making a more consistent product by defining and then eliminating sources of variation in the process. SPC works well for the discrete manufacturing industry by continually trying to reduce the tolerance in each part of the operation so the final product fits together tightly. One application is in the automotive industry. If the dimensional variation of each fender, support beam, door hinge, engine valve, and other parts of the car can be reduced, the final assembled product will make fewer rattles, run more smoothly, and last longer.

SPC is primarily a statistical focus on process measurements using various control charts. In the most common control chart, the Shewhart chart, the process measurement is plotted against time (see Figure 2.1). Upper and lower control limits are established based on the standard deviation of a historical database that represents normal operation. If points fall outside of the limits, it is understood as an event that requires attention. Such control charts can be a valuable system cultivation tool; however, the characteristics of most chemical process systems limit their effectiveness. Process dynamics, feedback

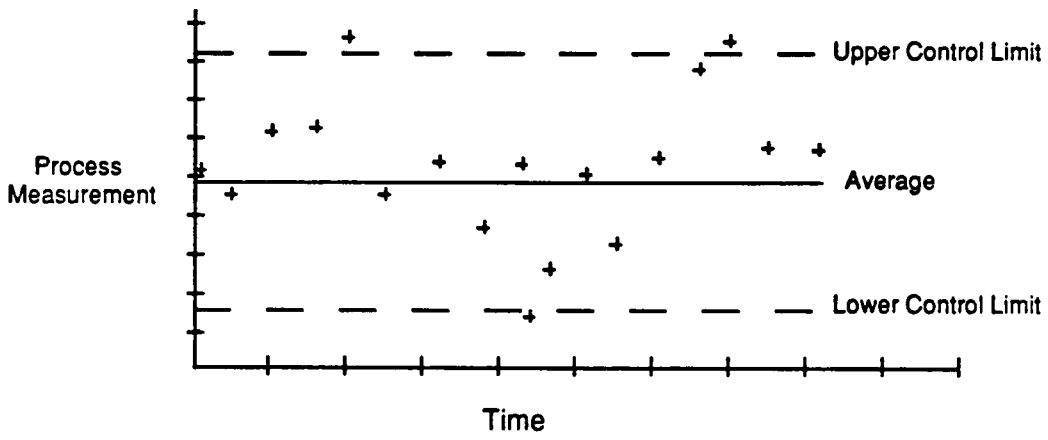


Figure 2.1: Shewhart control chart

control loops, and complex interactions between process variables all cause autocorrelation in the interpretation of the control charts. The biggest problem, however, is that chemical processes are deterministic as well as stochastic. Unavoidable disturbances in material and energy balances occur frequently and in many cases overshadow the normal statistical variation that simple control charts attempt to model.

Conventional control charts serve a role in the process industries, but it is primarily one of a very coarse filter: usually they can only detect major problems. More subtle problems operating a process and its control system are often completely missed. The following chapters present and discuss several system cultivation concepts that function at a much higher resolution than conventional control charts. These computer-based analyses approaches vary in complexity; however, they all show promise in detecting important but subtle changes in the operation of the process systems.

CHAPTER 3

Hypothesis Feedback Modeling

3.1 Introduction

In many chemical plants today, large historical databases are accessible through a mainframe computer. Currently, most of the engineering tools derived from these computers are simply glorified strip charts or single variable statistics. Although these tools allow process data to be accessible from any time period at the touch of a button, it remains difficult and time consuming to fully understand how well the process is operating. Examining time-sequenced data is difficult because of the overwhelming amount of data and the many interacting variables and relationships that must be considered. The computational power of process computers potentially offers better tools to aid in extracting multivariable relationships from large databases.

In addition, the workhorse of statistical process control, the Shewhart control chart, has limited use in a chemical production atmosphere. These control charts apply well to monitoring process variables that have random variation about a target value. In a multivariable environment, frequently non-random variations are encountered and often no target values exist. However, analogous to statistical process control, analyzing a "pattern of variation" still makes sense[49].

B.C. Moore introduced hypothesis feedback (HF) modeling with these considerations in mind[50]. Although HF-modeling and SPC focus on fundamentally different patterns of variation, the fundamental ideas of quality control still apply: to detect abnormal patterns and identify their cause, and to continually strive to improve the normal pattern of operation. The following paragraphs outline fundamental ideas underlying HF-modeling. Other articles by Moore elaborate on these basic principles [50] [47] [49] [51].

Three factors influence dependent system variables: a set of measured variables

$$(X_{m_1}, X_{m_2}, ..., X_{m_{n1}}),$$

a set of unmeasured system variables

$$(X_{u_1}, X_{u_2}, ..., X_{u_{n2}}),$$

and the state of the process (physical properties, process parameters, measuring devices, etc., that may vary over time). Both measured and unmeasured process variables have random variation associated with them. Therefore, a complete system model is given by

$$Y = f_n(X_{m_1}, ...X_{m_{n1}}, X_{u_1}, ...X_{u_{n2}}),$$

where the function f_n depends upon the state of the process. Unfortunately, X_{u_n} and physical system parameters cannot be directly measured and often cannot be fully identified anyway. HF-modeling aims to extract subtle information from patterns in the available data. Those patterns reflect system

states (inefficiencies or process problems) and important unmeasured system variables.

Consider the case in which no noise is present in any system variable. The function f_n assigns for each value of $(X_{m_1}, ...X_{m_{n1}}, X_{u_1}, ...X_{u_{n2}})$ a value for Y. Geometrically, $(Y, X_{m_1}, ...X_{m_{n1}}, X_{u_1}, ...X_{u_{n2}})$ defines a surface in the $n1+n2+1$ dimensional space. The process itself determines the shape of that surface. Each system inefficiency will probably affect this surface uniquely.

This ideal situation is clouded, however, because only the "viewing space" [49], $(Y, X_{m_1}, ...X_{m_{n1}})$, is accessible. Unmeasured inputs impose a nonrandom variation on the patterns in the viewing space. The picture is further clouded by the random effects of each variable. The resulting large uncertainty often impedes reliability of mathematical models. The nonrandom nature reduces the effectiveness of multivariate statistical analysis.

Figure 3.1 illustrates an example of a cross section of a viewing space. The plot is a cross section of (Y, X_1, X_2) taken from a simple simulation of a first order lag plus dead-time model. It shows Y vs. X_1 for constant X_2. Process inefficiencies, unmeasured disturbances, and random variation will affect the location, shape, and distribution of these cross sections.

HF-modeling seeks to characterize process inefficiencies and unmeasured disturbances by relating them to patterns in the viewing space. It seeks not only to identify subtle process problems but also inherent flaws in normal operation and unknown system variables affecting the system. The work presented here begins to bring substance to these ideas by applying them to some rela-

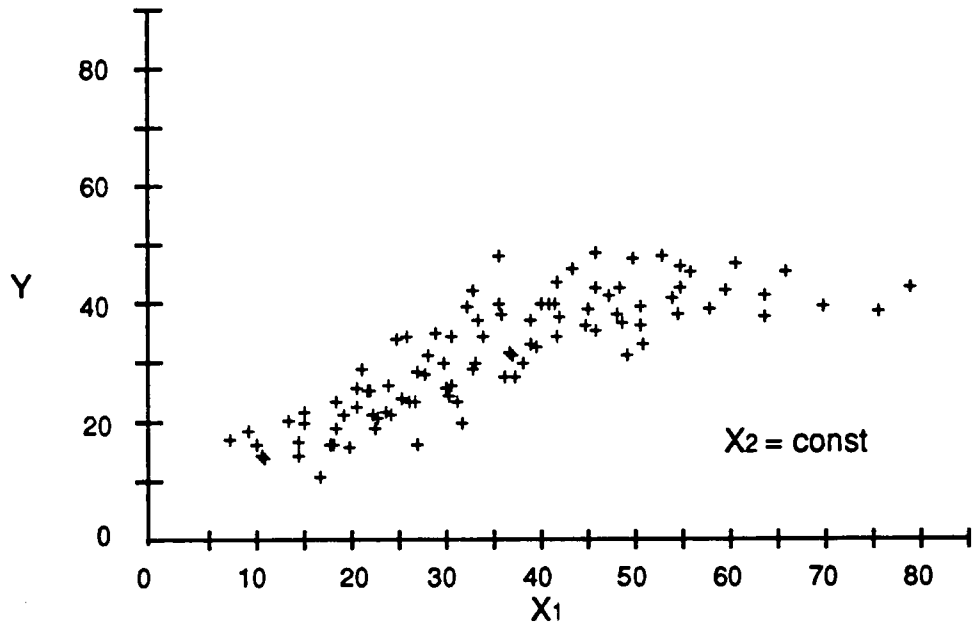


Figure 3.1: Cross section of viewing space

tively simple chemical processes.

A simplification made in this study is to limit the effects of unmeasured disturbances. Although their effects must be considered for effective real implementation, this work focuses primarily on how specific process problems affect patterns in the viewing space.

As applied in this work, HF-modeling relies heavily on two plots: multivariable statistical plots and relationship plots, as described in Section 3.2 and references[17] [16] [4] [54]. Although not as general as the plot of Figure 3.1, which plots actual plant data, multivariable statistical plots and relationship plots reflect the cross sections of the viewing space. These plots offer some advantages. First, they enhance important properties in the data. Second, they are easy to interpret and can be related to commonly used statistical charts. Third, they allow engineers to take advantage of a reduced information database in which information is more readily accessible and understandable than historical data.

Multivariable statistical plots and relationship plots do enhance certain aspects of the viewing space. However, by enhancing one aspect of the pattern, other aspects are reduced or even lost. For example, relationship plots enhance the direction of the relation, but the overall shape of the pattern is lost. In future studies, other pattern features must be enhanced and included in the reduced database. This will be important in monitoring the effect of unmeasured disturbances.

The following section details the step-by-step HF-modeling procedure,

as it is implemented in this study. Other discussions and applications can be reviewed in [17] [16] [4] [25].

3.2 Step-by-Step HF-modeling Analysis

3.2.1 HF-model definition

The first step in HF-modeling is define hypothesis model(s) for the process in the form

$$Y_{dep} = f(X_{m_1}, ...X_{m_{n1}}, X_{u_1}, ...X_{u_{n2}}).$$

In reality, only X_{m_n} are accessible; therefore, defined models will have the form

$$Y_{dep} = f(X_{m_1}, ...X_{m_{n1}}).$$

Y_{dep} are system dependent variables (outputs) and manipulated variables, X_{m_n}, are important independent system variables (inputs) and important interaction terms (e.g. $Flowrate * Temperature$). In this form, unmeasured disturbances and process changes appear unaccounted for. Remember that feedback of HF-modeling is a pattern of variation (defined by the above model) affected by process problems and unmeasured disturbances.

Hypothesis models monitor mass, energy, and momentum balances around a system. For example, the steady-state energy balance around a shell-and-tube heat exchanger is given by

$$F_{cw} = \frac{F_{hot} C_{p_{hot}} (T_{hot_{in}} - T_{hot_{out}}) + UA_{outer} (T_{cw} - T_{amb})_{lm}}{C_{p_{chill}} (T_{cw_{out}} - T_{cw_{in}})}.$$

Based on that equation, a reasonable hypothesis model might be defined as

$$F_{cw} = f(F_{hot} * T_{hot_{in}}, T_{cw}, T_{amb}).$$

System models for more complex systems will rely less on theoretical equations and more on an engineer's judgment.

The systems monitored by hypothesis models may be a single unit, like the heat exchanger, or a multi-unit plant. It is possible to envision a hierarchy of hypothesis models that monitors on a plant-wide perspective down to each individual control valve.

On processes of any scale, hypothesis models are basically the same. Effective models capture all primary important measured variables and interactions between them, while not including any variables of only minor significance. Minor variables impede the effectiveness of the analysis. Establishing an efficient model requires only basic knowledge of the process in identifying important system variables.

3.2.2 Defining resolution of each variable

The resolution of each variable comprising the hypothesis model must be defined. Defining this resolution reduces the precision of each variable, breaking down the entire span of each variable into small increments such that operation can be considered reasonably constant within each increment. Geometrically, the n-dimensional space consisting of X_{m_i} is broken down into regions of hypercubes. For example, the heat exchanger model requires the coordinates of this space to be $F_{hot} * T_{hot_{in}}$, T_{cw}, and T_{amb}. Figure 3.2 shows that defining the resolution of each variable divides the three-dimensional space into cubic regions. Inside each, constant operating conditions $(F_{hot} * T_{hot_{in}}, T_{cw}, T_{amb})$ are assumed. The regions defined by each hypercube are referred to operating conditions. Each has an address comprised of an integer value from each axis.

Defining resolution of each variable requires a trade-off. On one hand, a statistically significant amount of data must fall into each operating condition for the analysis to be meaningful. This usually requires low resolution (or low precision). On the other hand, the resolution cannot be so low that the assumption of constant operating conditions is violated. At first, such a trade-off may seem difficult to attain. It is not as difficult as it appears, but it does require a knowledge of each variable's relative impact on the process.

Reducing precision allows the viewing of "slices" of the viewing space that are of finite thickness. The resulting images will have sufficient information to yield meaningful analysis yet still be considered true cross sections of the viewing space.

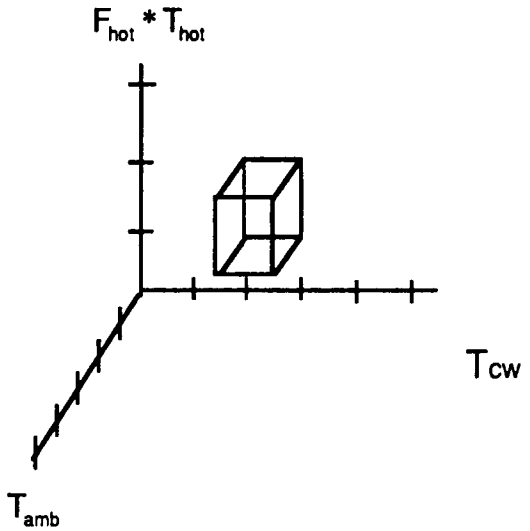


Figure 3.2: Dividing the viewing space into cubic regions of constant operation

3.2.3 Sorting into the reduced database

The reduced database is an important aspect of HF-modeling. The reduced database is up to two orders of magnitude smaller than the original historical database, yet it contains important information in an easy-to-interpret form.

The reduced database contains statistics and relationships concerning important system variables represented within each operating condition. Figure 3.3 depicts a typical reduced database for the heat exchanger model. In this database, each line of information contains important statistics and relationships representing one operating condition. The first three elements denote the operating condition address for each line. Next, a series of averages and standard deviations summarize important dependent system variables and manipulated variables. Following those, partial derivatives monitor important system relationships (these partials are average slopes rather than numerically calculated derivatives). For example, one of the relationships monitored on the heat exchanger was the partial derivative of chill water flow (the manipulated variable) with respect to ambient temperature (one independent variable) while holding all other important independent system variables constant.

3.2.4 Multivariable statistical plots and relationship plots

The reduced database described in the previous section and exemplified by Figure 3.3 comprises information contained in historical data in an enhanced and compressed form. The current objective is to present that information in

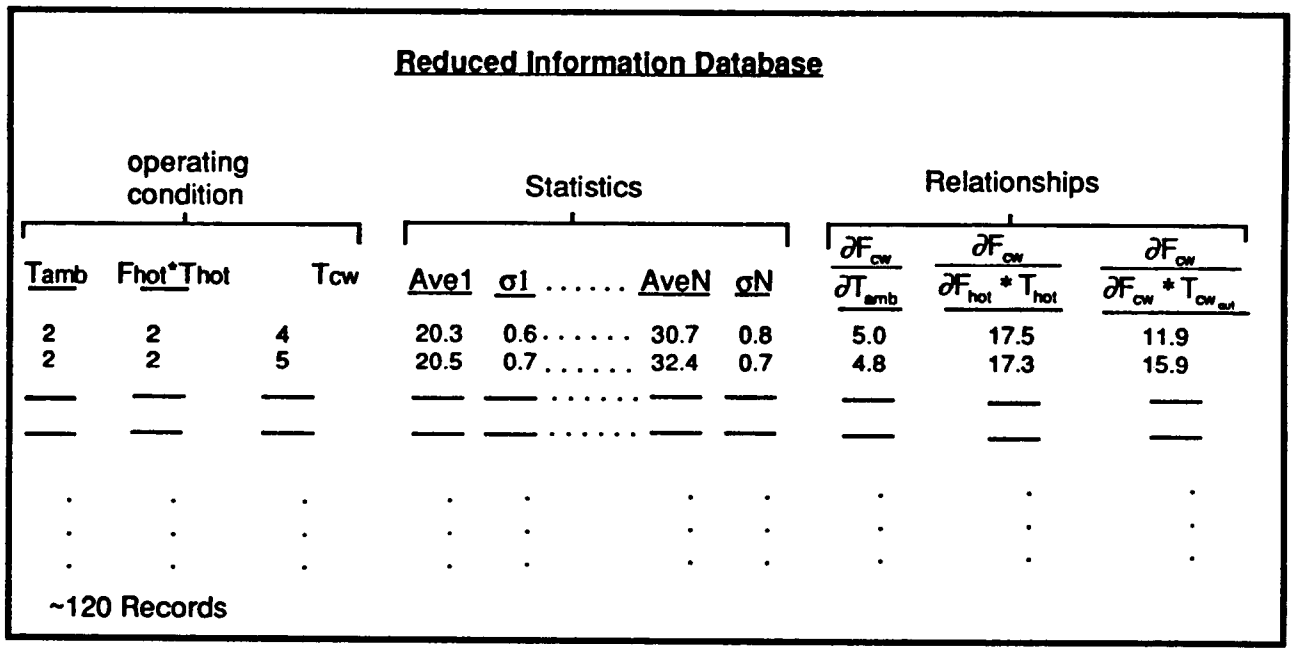


Figure 3.3: Example reduced database for heat exchanger

a simple, easy-to-interpret, graphical form. Multivariable statistical plots and relationship plots are used extensively in this study for that purpose. These graphs of cross sections of the viewing space are readily accessible from the reduced database.

Multivariable statistical plots present statistics of dependent variables (usually controlled or manipulated variables) in a multivariable framework. Figure 3.4 illustrates a typical multivariable statistical plot. These plots give average and four- or six-sigma standard deviation of a process variable against an operating condition number. Operating conditions are numbered according to increasing average process variable.

Typically, a set of data under analysis is presented in comparison to basecase operation. Figure 3.5 exemplifies such an analysis, plotting information from a simulation in which a heat transfer coefficient has been reduced by 10% with information representing basecase or normal operation. On this graph, statistics from the ΔUA data are plotted with common operating conditions of basecase. This allows comparison of the two data sets on common ground. Examples in Chapter 7 show the effectiveness of this analysis in detecting subtle process changes.

Two important "reporters" of these multivariable statistical plots are fractional overlap and separation. These reporters attempt to capture important attributes of a plot like Figure 3.5 in two simple parameters.

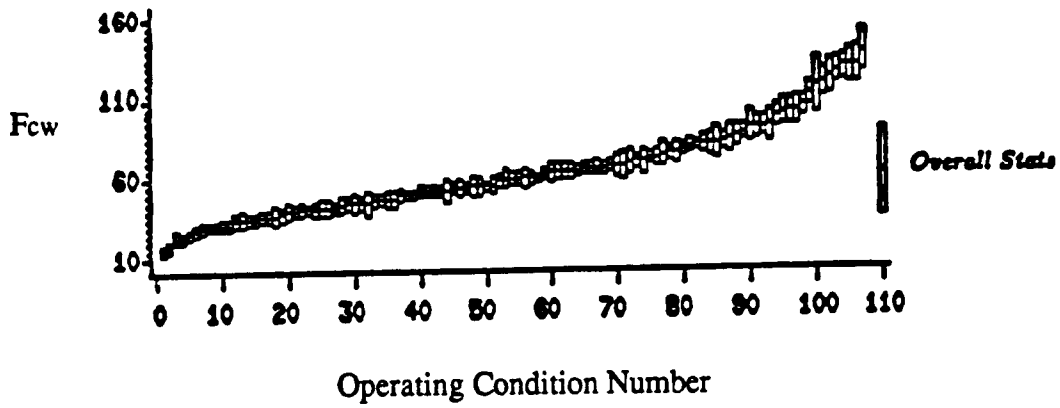
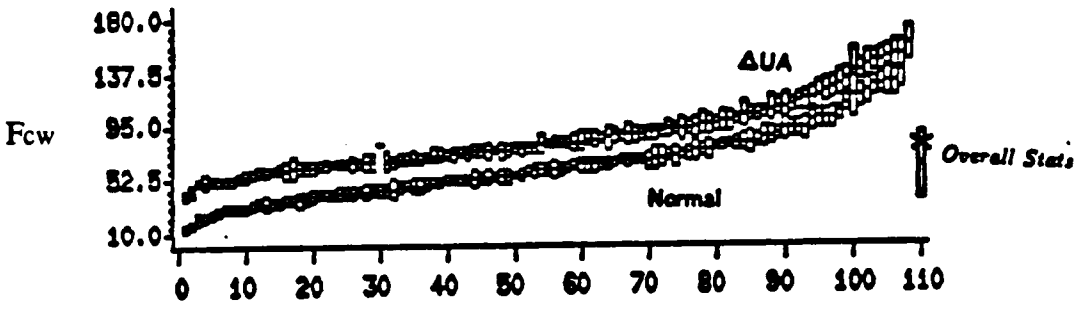


Figure 3.4: Multivariable statistical plot



X- overall ave. of ΔUA data.

Figure 3.5: Multivariable statistical plot showing data compared with basecase operation

Separation is defined as

$$Separation = \frac{1}{n}\sum_{i=1}^{n}\frac{M_{new_i} - M_{norm_i}}{\sigma_{new_i} + \sigma_{norm_i}}$$

where

M_{new_i} =mean value in operating cond. i in new data

M_{norm_i} =mean value of M_i in normal operation

n = number of operating conditions

σ = standard deviation

It is a quantitative measure of separation between the two bands. Fractional overlap is defined as the fraction of the two-sigma windows in common operating conditions that overlap. These two parameters will later help to summarize an HF-modeling study.

Figure 3.1 illustrates a viewing space cross section. It contains information concerning a relationship between a dependent system variable and a measured independent system variable while holding all other measured variables constant. It also contains information concerning unmeasured system variables and process changes embedded in the pattern of the data.

A relationship plot is a simplification of a viewing space cross section. It plots the average of a particular process variable in each operating condition against an independent system variable while all others are held constant. Figure 3.6 illustrates a typical relationship plot, showing the relationship between F_{cw} and $T_{cw_{out}}$. Each curve represents a different value of E $(=F_{hot} * T_{hot_{in}})$.

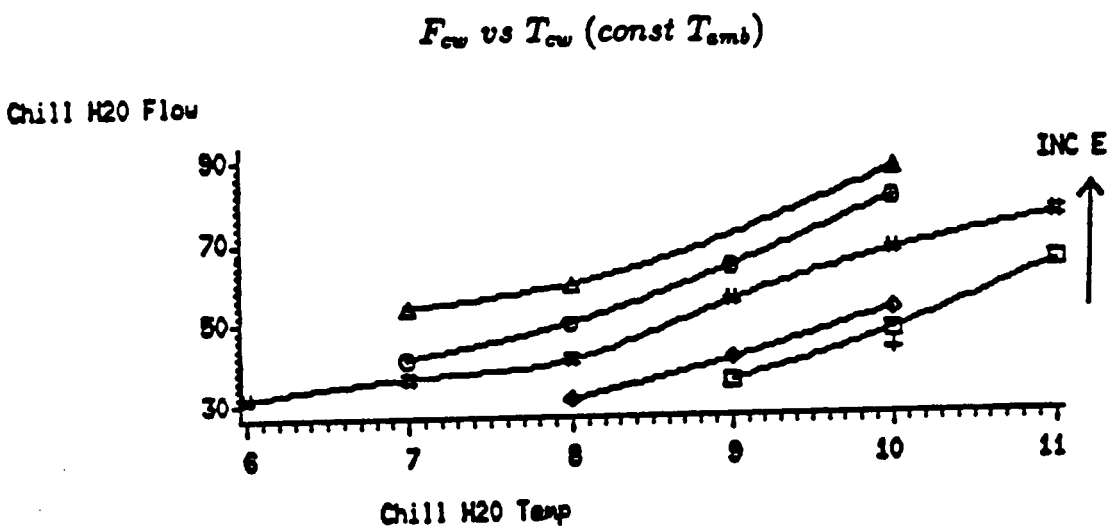


Figure 3.6: Relationship plot

On a relationship plot, the slope or partial derivative at each point or simply a best linear fit aid in detecting and identifying changes in process operation.

Note that by presenting a viewing space cross-section in the form of a relationship plot, certain features of the data are enhanced. In addition, the information is entirely contained in the reduced database, whereas the viewing space cross-section requires the entire historical database. However, in the compression of the data, some important information about the pattern of the data may be lost. In the example applications of this work, the relationship plots sufficiently describe the pattern, however, more complex situations will certainly need to incorporate other features from these cross-sectional patterns.

3.2.5 Model refinement

Although not a central focus of this work, hypothesis model refinement is an important subject. The goal is a model including a minimal number of independent variables resulting in an adequate description of the process. Process knowledge gained through years of experience and through theoretical considerations should yield most of the important system variables in the hypothesis model. Invariably, though, small model refinements will improve results. The following paragraphs describe general guidelines for these model refinements.

In evaluating a hypothesis model, we seek answers to the same questions posed in a multiple regression analysis. Those basic, frequently-asked questions are summarized by Neter[56]:

1. What is the relative importance of the effects of the different independent variables?

2. What is the magnitude of the effect of a given independent variable on the dependent variable?

3. Can any independent variable be dropped from the model because is has little or no effect on the dependent variable?

4. Should any independent variables not yet included in the model be considered for possible inclusion?

Although basic assumptions of a linear multiple regression analysis may limit the effectiveness of model evaluation, some insights gained from a statistical analysis may be helpful. Those invalid assumptions include a normally distributed variation, accessibility of important system variables, and linearity of the relationships.

Neter et al.[56] summarize two statistical methods for arriving at an optimal model. One method considers all possible models under several different statistical criteria (most are a measure of variance), then chooses the one that works best over all of the criteria. The second method is a stepwise regression procedure. A simple search technique, at each step it adds or deletes an independent variable. This method is based on the F test.

Both methods may be helpful, particularly when our process knowledge is limited or the number of process variables is enormous. Most of the

time, though, process experience or basic material and energy balances describe good hypothesis models. This holds true in the systems studied in this work. Therefore, only fine adjustments are considered. In this work, model refinement involves postulating a slight change in the model first. The resulting model is evaluated by examining variation in multivariable statistical plots and sensitivities in relationship plots.

Two insights from statistical analysis are important in this method. First, when two or more independent variables are highly correlated, one and only one of them should be included in the model. Often analysis of these variables can be deceiving. Including more than one of these intercorrelated variables yields an inefficient model, while including none of them gives an incomplete model. Second, interactions between process variables should be examined. Often, if examined independently, two variables appear to have minimal effect on the system, but interactions between the two have a major impact. For example, taken individually, flow rate and temperature of an inlet stream may appear of only minor significance, but the product of the two (an energy term) may prove to be of major importance.

Once a good model has been found, statistical plots and relationship plots need to be monitored on a daily basis. Such an analysis will better indicate process changes. Over time, process engineers and/or artificial learning algorithms can learn to better identify specific process problems and unmeasured disturbances.

3.3 Summary of HF-modeling

HF-modeling can be implemented on processes of any scale, from simple single-input, single-output process to highly interacting, multi-unit processes. Figure 3.7 depicts a generic process of any size or complexity. Implementing HF-modeling on this process requires defining important independent system variables and interactions between those variables. Based on those variables, large data sets are compressed by sorting data into physically meaningful categories. Finally, important statistics, relationships, and patterns of variation of dependent, manipulated, and controlled variables need to be monitored daily.

HF-modeling has several important characteristics. First, it incorporates process knowledge into the framework, but only in identifying important system variables and their relative impact on the process. Second, the feedback to the engineer or operator is in terms of easy-to-interpret charts and graphs. These plots reflect process operation over the time frame of the historical data. Finally, HF-modeling helps to improve understanding of the process through continual feedback on multivariable system relationships.

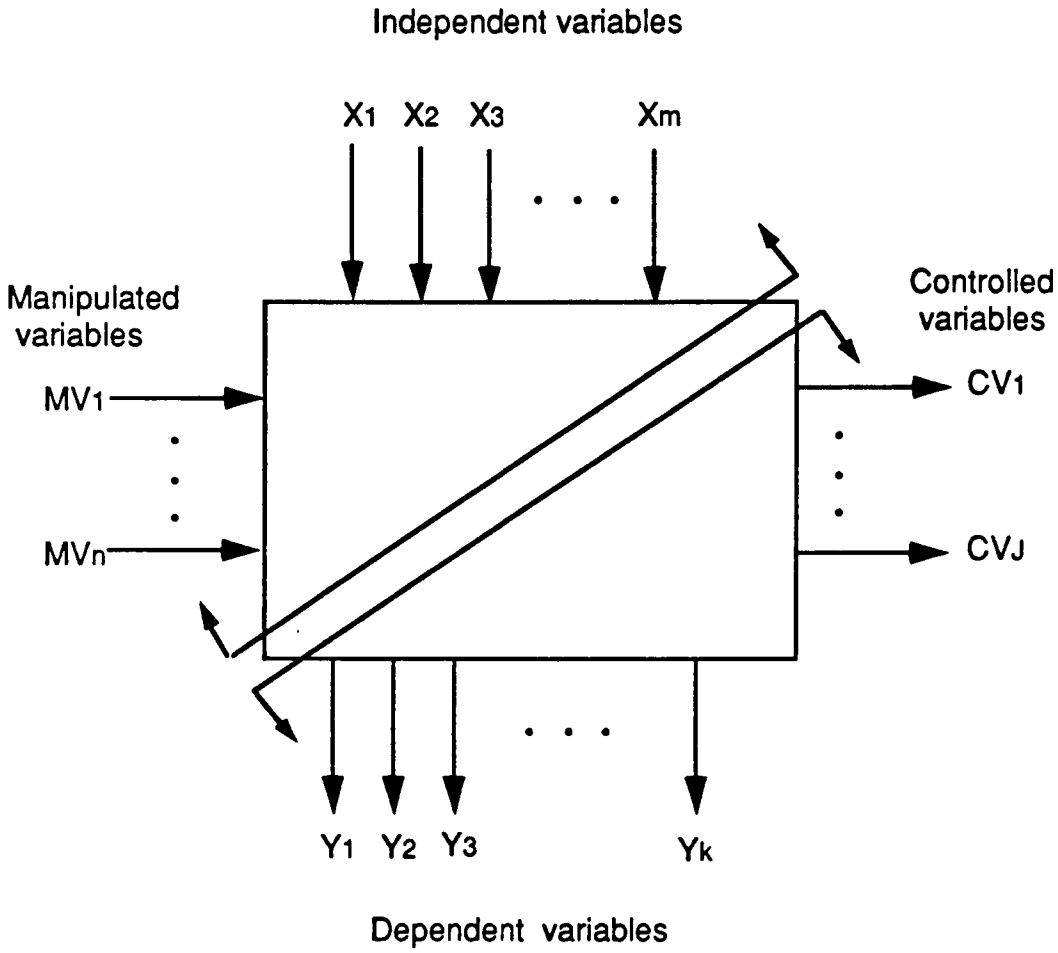


Figure 3.7: Generic System

CHAPTER 4

Time-Shifted Distribution Analysis

As a tool for distinguishing between plant-model mismatch and unmeasured system disturbances in a model predictive controller, Hackney initiated the concept of time-shifted distribution (TSD) analysis[25]. The basic concepts of TSD also proved effective in evaluating the performance of any controller, be it PID control, adaptive control, model predictive control, or simply manual control. As a cultivation tool, TSD examines historical data to identify control loops that need attention.

Like HF-modeling, TSD analysis focuses on extracting relevant information from historical data. However, unlike HF-modeling, TSD is concerned with dynamic rather than steady-state relationships. In this work, TSD's main concern is examining the dynamic relationship between the manipulated variable(s) and the controlled variable(s) in a process.

TSD analysis describes a controller's performance by one of three qualities: overcompensating, undercompensating, or controlling reasonably well. The analysis simply raises a flag on a controller when it performs poorly. In a typical chemical plant where an engineer may oversee 200 or more control loops, such an analysis is important. It helps the engineer to focus effort on control loops that truly need attention.

4.1 Description of the TSD Analysis

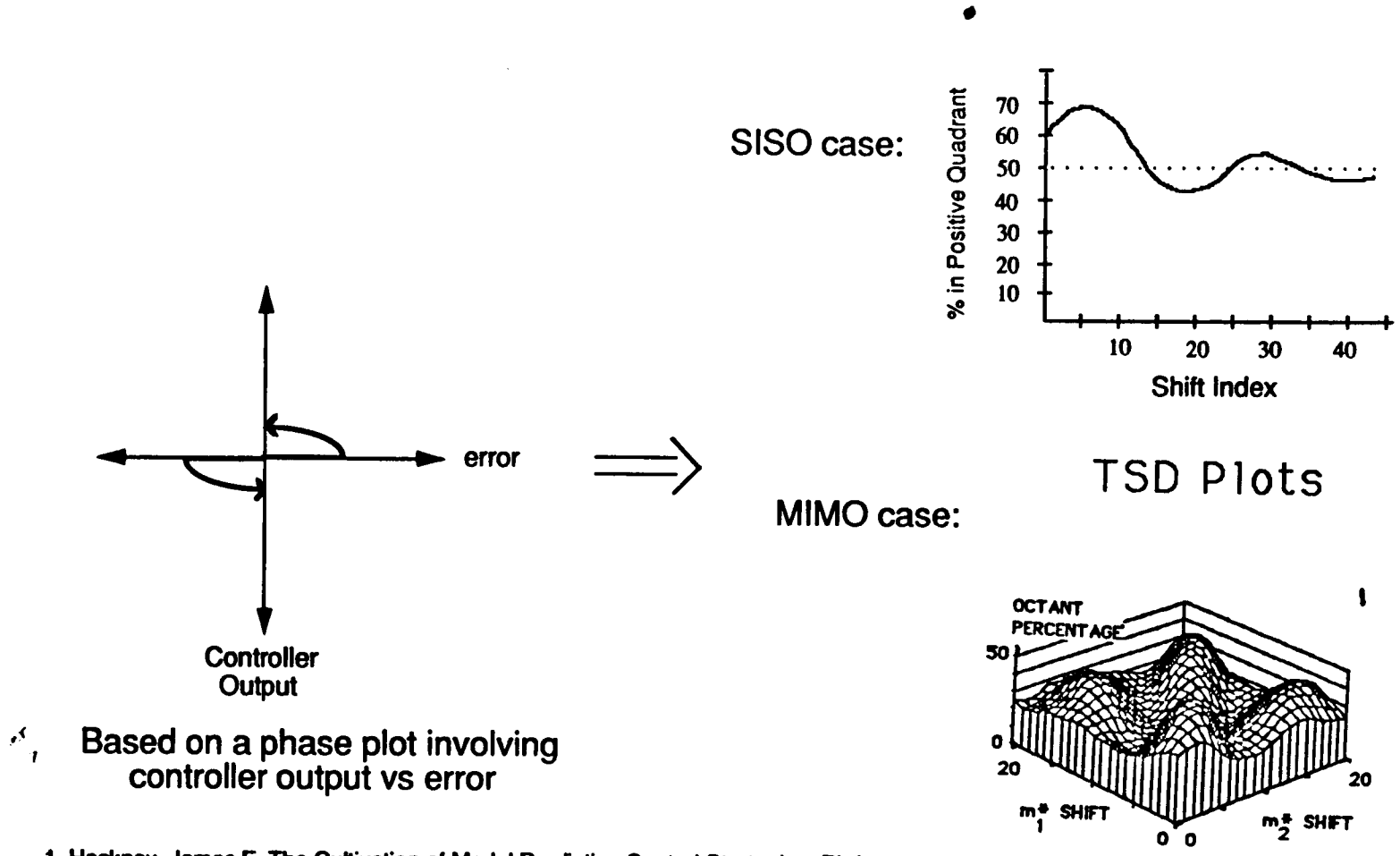
A complete discussion of TSD is detailed in a dissertation by Hackney[25] A summary of the important concepts are outlined here for completeness, along with introducing some new ideas.

Figure 4.1 depicts an overview of the TSD analysis. The basis of this analysis is a phase plot of controller output (process input) versus error (desired output - actual output). A series of shifting operations, similar to those of autocorrelation, transforms the phase plot information into the TSD plot. In the single-input, single-output (SISO) case, it is a single two-dimensional plot, like the one in the figure, for each control loop. The analysis becomes more complex in the multiple-input, multiple-output (MIMO) case, in which the result is a series of n-dimensional surfaces. These surfaces are more difficult to interpret visually.

Although it focuses on different aspects of historical data, the TSD framework is much the same as HF-modeling. Both examine specific attributes within historical data. Both result in graphical images that can be interpreted by a human or an intelligent machine.

4.2 Explanation of the TSD Plot

A well-tuned controller reacts to disturbances by moving in positive quadrants of a normalized phase plot involving input versus error. On the



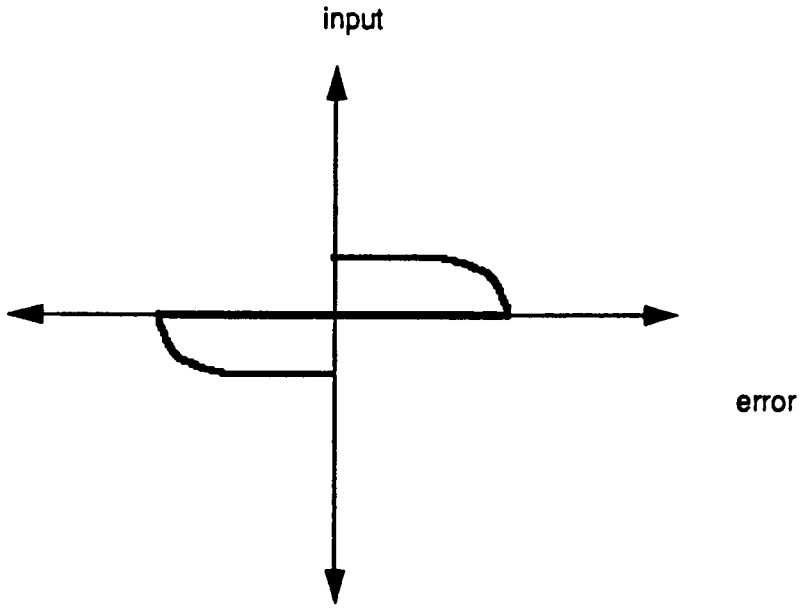
1. Hackney, James E. The Cultivation of Model Predictive Control Stratagies. Phd disertation, The University of Tennesse, Knoxville, 1989.

Figure 4.1: Overview of TSD analysis

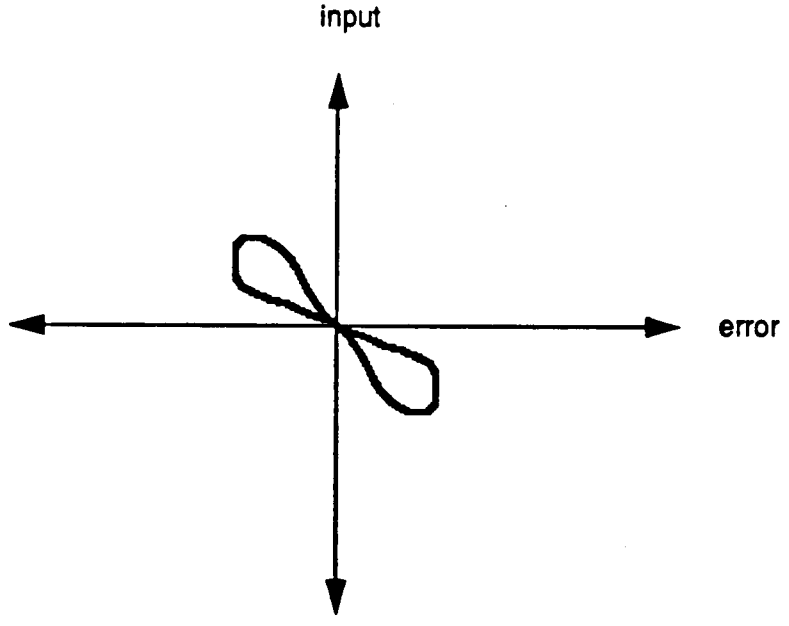
other hand, a poorly tuned controller, one which tends to instigate process errors, will move primarily in negative quadrants of the same phase plot. These concepts underlie the TSD analysis. Figure 4.2 illustrates these two cases. In Figure 4.2 A the SISO controller responds to disturbance-induced positive errors by increasing the process input, gradually reducing the error to zero. The resultant trace lies in the first quadrant of the phase plane. For negative disturbance-induced errors, the trace lies in the third quadrant. However, for controller-induced errors, unmerited inputs first increase output error before the control system reacts to reduce it. These data points lie, most frequently, in the second and fourth quadrants of the phase plane as shown in Figure 4.2 B.

An analysis employing these phase plots seems simple; calculate the percentage of the data distributed in each quadrant. However, several problems limit the effectiveness of such a simple solution. First, how should the input be normalized? Second, how can dead time be considered? Third, can the effect of different tuning parameters be better quantified in this analysis? Hackney addressed some of these issues; his solutions are summarized below.

An appropriate controller output must be defined so that the path on the phase plot returns to the origin. Two controller outputs have been investigated to this point, both are summarized below.



A) Well-tuned controller reacting to a system disturbance.



B) Controller-induced error.

Figure 4.2: Phase plot: Controller output vs error

Change in process input: $\Delta U = U(k) - U(k-1)$, and

Normalized input: $U^*(k) = U(k) - U^*_{ss}$,

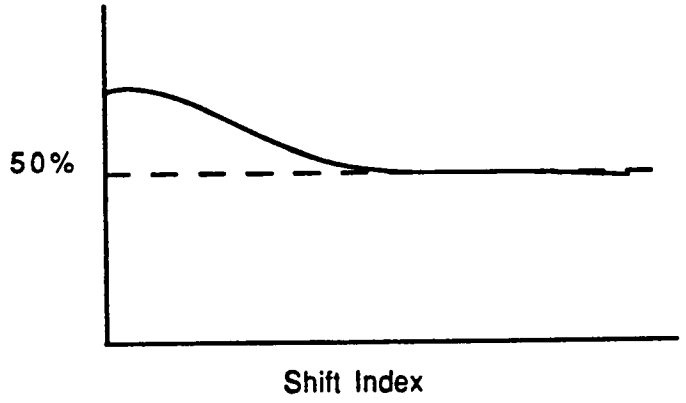
where $U(k)$ = Controller output at k^{th} sampling instant

U^*_{ss} = Predicted controller output at steady state

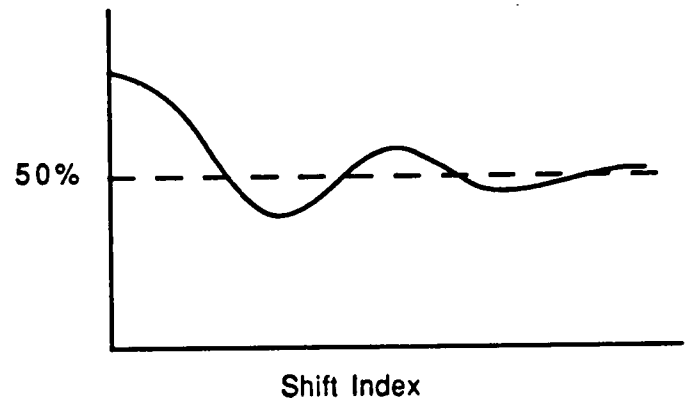
The change in input, ΔU, is the easiest input to compute and apply to any type controller. However, in many cases this input leads to data points in the "wrong" quadrants[25]. The second input subtracts the present controller output from a steady state predicted controller output. This prediction is readily available in Model Predictive Control techniques but unfortunately not in classical PID controllers. Because 95% of industrial control loops are PID, the need to develop input normalization techniques that make the TSD analysis reliable for PID control remains. Such techniques are suggested in later sections of this chapter.

In real situations, process dead time and consecutive disturbances entering a process must be taken into account. Because the time delay is not known ahead of time, shifting the data one time step at a time to some point beyond dynamic influence may be necessary. At each time shift, the quadrant distributions can be calculated. Finally, plotting the quadrant percentage versus time shift index helps evaluate the controller performance. Such a plot has been termed a TSD plot.

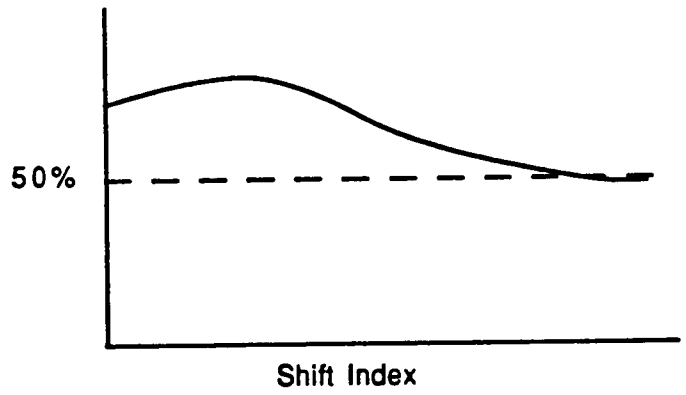
Figure 4.3 exemplifies TSD plots for a typical SISO process. The base-



A) Good Control



B) Overcompensating



C) Undercompensating

Figure 4.3: Typical SISO TSD plots

case, overcompensated case, and undercompensated case are easy to distinguish. Notice that these TSD plots look very similar to a time series system responses to a step change in a disturbance. The TSD analysis has two major advantages over step response methods. First, TSD plots reflect process operation over a much longer time frame and over a wider range of operating conditions. Therefore, the TSD plot reflects the overall performance of the controller more completely than a step response. Second, the TSD analysis employs data from the normal operation of the process, artificial perturbations on the system are not necessary as they are on step response analysis.

4.3 TSD analysis of a PID controller

Because PID controllers comprise 99% of industrial control loops, TSD analysis of PID control is addressed here. Defining a controller output (or process input) is probably the most critical component of the TSD analysis for obtaining consistently meaningful results. The last section outlined two methods investigated so far.

The first output is ΔU, the difference between two consecutive controller outputs; it applies directly to any controller and has shown to lead to reasonable results[4]. The second method, $U^*(k)$, has proven to achieve meaningful TSD plots more consistently. This section introduces an improved normalized input that applies to PID controllers, based on the second method.

Computing the $U^*(k)$ controller output requires a steady-state predicted

controller output. However, because the TSD analysis is computed off-line, evaluating $U^*(k)$ is not necessarily limited to model-based controllers. The only requirement is a model that predicts U^*_{ss} (the steady-state predicted controller output). HF-modeling provides such a model. HF-modeling is a non-parametric model formed from the data set itself; therefore it forms an accurate model based on that day's operation. Effects of plant-model mismatches in the analysis are eliminated.

The following sequence employs HF-modeling to calculate U^*_{ss} and improves the TSD analysis for PID control.

1. Perform HF-modeling analysis on the data set. The historical data must be retained along with the reduced database.

2. For each record in the historical database, find its corresponding operating condition in the reduced database. Obtain the average controller output. This is U^*_{ss}.

3. Augment the historical data with this average controller output.

4. Perform the TSD analysis using $U^*(k)$ vs. error as performed by Hackney[54].

This method has been implemented successfully. Figures 4.4 and 4.5 compare the TSD analysis of a process using ΔU versus $U^*(k)$. Those using $U^*(k)$ are nearly always clearer. The most important improvement lies in the location of the 50% line. Theoretically, at sifts beyond dynamic influence, TSD plots should settle near the 50% line with some small variations. Often

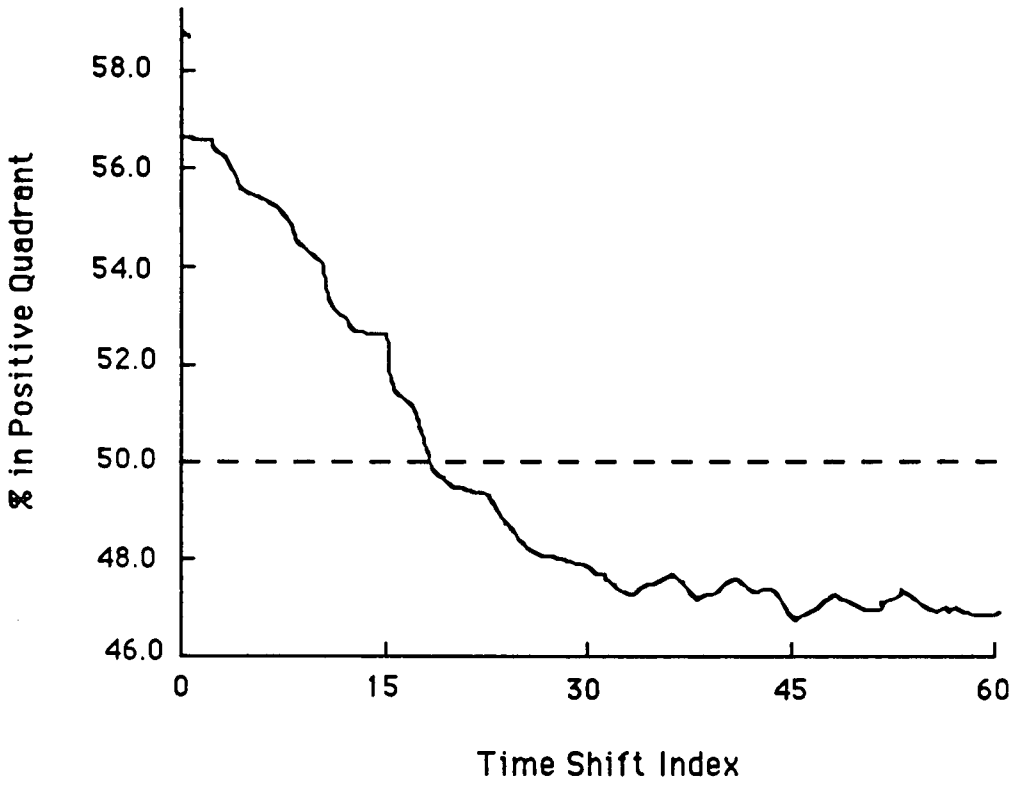


Figure 4.4: SISO TSD plot (using ΔU)

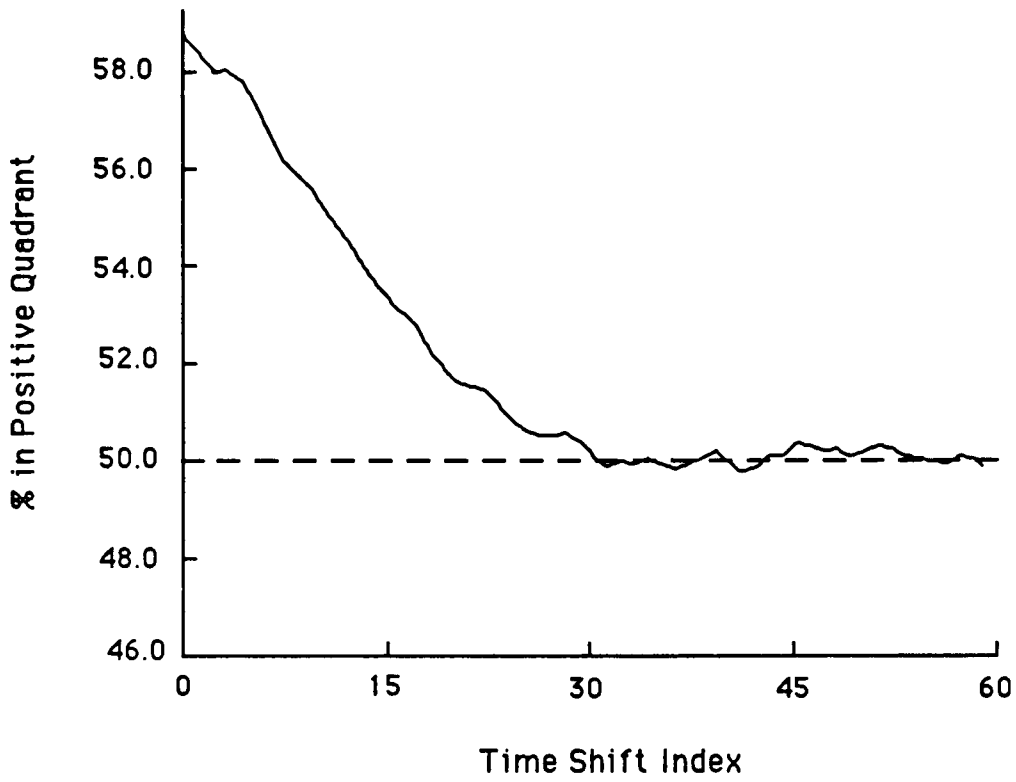


Figure 4.5: SISO TSD plot (using $U^*(k)$)

TSD plots using ΔU do not settle near this line (e.g. Figure 4.4), thus adding uncertainty to the TSD analysis. Plots employing $U^*(k)$ nearly always settle near the 50% line and are therefore much more reliable.

4.4 MIMO TSD

Extending the TSD analysis from SISO to MIMO is not straightforward. The multidimensionality of the problem hides important information. For example, on a 2-input, 2-output system, one must examine the distribution in octants of eight 3-dimensional spaces. The problem becomes intractable for a simple human analysis in higher dimensional systems. For this reason, the MIMO analysis lends itself to Automatic Pattern Recognition techniques, not only to examine regions where important information lies but to find those regions in the first place.

CHAPTER 5

Pattern Recognition

An area of study within the field of Artificial Intelligence, pattern recognition is the ability of a system (either biological or artificial) to group items of similar properties into individual groups or classes. Pattern recognition spans many scientific disciplines, searching for an answer to the common problem of classifying elements from a set of input data into identifiable classes [14]. Examples of successful pattern recognition applications are plentiful. They include character recognition, voice recognition, classification of remotely sensed data (such as aerial photography), and biomedical applications in chromosome analysis.

Pattern recognition can contribute as a cultivation tool in two ways. First, it organizes large data sets containing multiple modes of system operation into smaller subsets, each representing a single mode of operation. Second, it automatically interprets information from other cultivation tools, like HF-modeling and TSD analysis.

This chapter outlines basic pattern recognition concepts and describes how they apply as system cultivation tools. It outlines several classical pattern classifying algorithms and clustering algorithms in a general manner. This chapter focuses more specifically on neural networks and how they are useful

as pattern classifiers and cultivation tools.

5.1 Outline of Pattern Recognition

Figure 5.1 overviews a typical pattern recognition analysis. In this analysis, regions of the aerial photograph are to be identified as water, land/trees, and roads. The first step organizes the image into homogeneous regions, as shown in the figure. Next, each region is analyzed separately by extracting and enhancing important features of each region, for example, color, brightness, and the length-to-diameter ratio of the region. Each image is reduced to a pattern vector (of dimension 2 in the example) that captures the essential attributes of the image. This pattern vector is then submitted to a pattern classifier to be mapped into its corresponding class.

The above simple example describes the three general phases comprising a pattern recognition analysis: clustering or data organization, image processing, and pattern classification. The following discussion elaborates on each phase and how it relates to examining process data.

Clustering analysis organizes large data sets into smaller groups or clusters. By definition, clustering is a two-part process: cluster validity, or determining the number of clusters present; and assigning each pattern to a cluster in some optimal manner. In most applications, surprisingly, identifying the number of clusters is much more difficult than grouping the data. The information being clustered may represent an aerial photograph (as in the example),

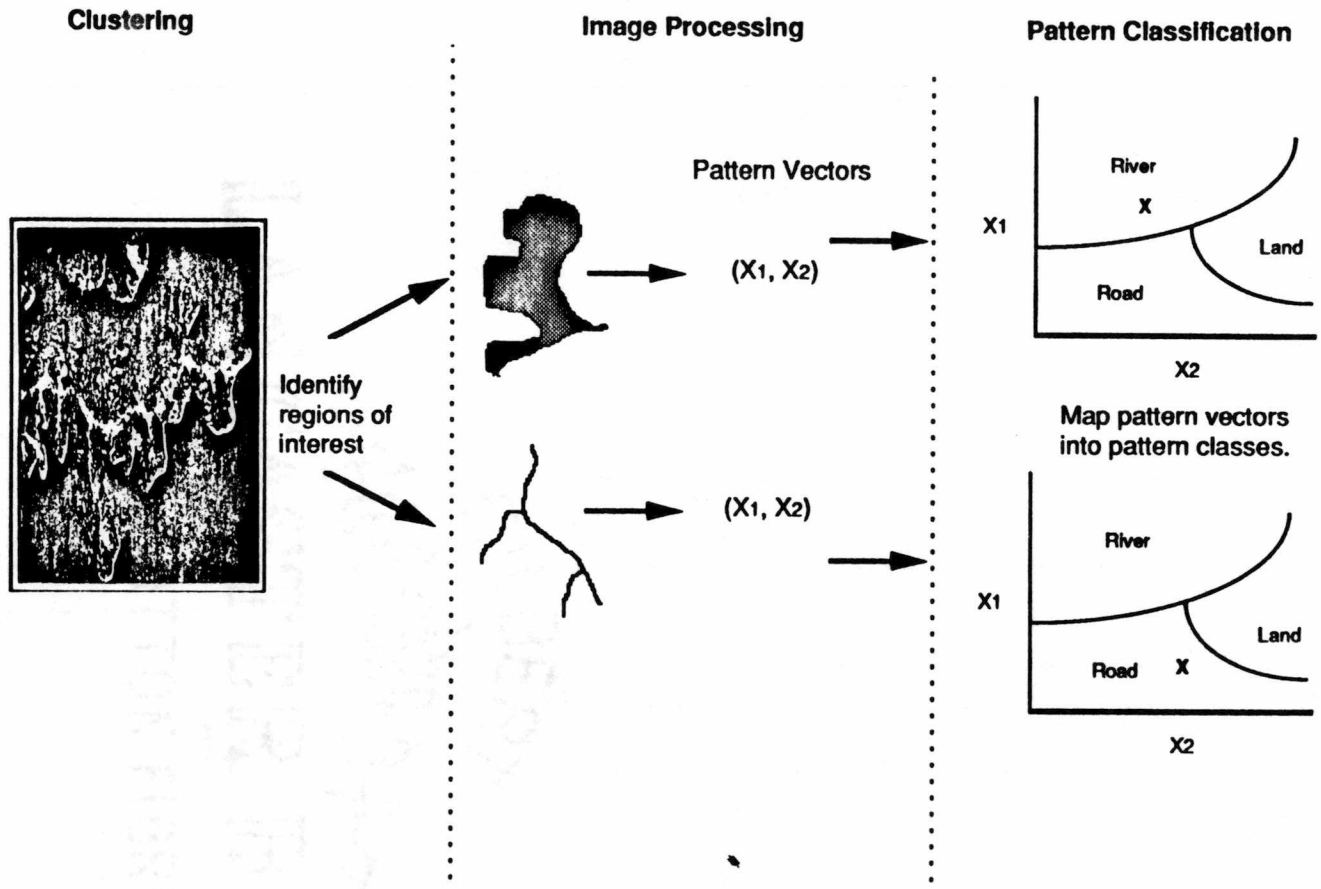


Figure 5.1: Overview of pattern recognition analysis

spectral information, historical or cultivated data from a process, or any other imaginable entity. Such an organization of the information is an obvious necessity to break a large problem down into smaller, more manageable problems. In Figure 5.1, the clustering analysis grouped the information into fairly distinct regions: water, land, and pavement. Analogously, clustering will help organize large historical data sets into regions of different process operation. Many well-known clustering algorithms exist. Maximin, K-means, Prim's, Complete Linkage, and Single Linkage are a few of the more useful ones[14] [72]. Each has unique clustering characteristics and is useful in given situations.

Image processing, the second phase of pattern recognition identifies important attributes of each pattern class and summarizes those attributes in the form of a pattern vector. Often recognized as the most important phase of pattern recognition, image processing essentially distills each cluster of information into its most important properties. Certain image processing tools are applicable to almost all situations, like scaling, but the essential ingredient to successful image processing is an understanding of the information being processed. In terms of analyzing historical data from a chemical plant, knowledge, as always, is essential.

In the final phase, a pattern classifier maps each pattern vector into its appropriate class. In Figure 5.1, the classifier maps each vector into one of three classes: water, land, or roads. A pattern classifier is basically an algorithm that draws decision boundaries in the pattern space. These algorithms range from simple distance-based pattern classifiers to Baysian or statistical

classifiers. From an engineering point of view, successful applications choose a pattern classifier whose characteristics best fit the application. Once chosen, the pattern classifier must be trained. During training, the parameters that fully specify the pattern classifier are identified, usually through successive iterations.

In this work, clustering plays a small role. Because it is critical to extracting information from large databases, clustering will most certainly be the focus of later studies. However, in this work each data set is assumed to be homogeneous: if a system problem is present, it is present throughout the data set. Image processing plays an important role in the work presented here. The primary ingredient is an understanding of the information presented by such cultivation tools as HF-modeling or TSD analysis. Of the pattern recognition analysis components, pattern classification is the primary focus in this study. A multilayer perceptron neural network will be the primary pattern classifier applied for this purpose. The next section outlines classical pattern classifiers; Following that, an extensive section on neural networks is presented.

5.2 Classical Pattern Classification

Three general approaches to pattern classification are heuristic, syntactical, and mathematical. The heuristic approach consists mainly of adhoc methods and is very problem dependent. Although this solution is important for implementation in a real situation, it does little to attack the general

problem. Syntactical pattern recognition characterizes patterns in terms of grammars and allows the application of formal language theory to the problem. This approach is more relevant to language-related problems; it has been attempted in more general problems, but with little success. The mathematical approach derives classification rules in a rigorous mathematical framework. This approach is the most relevant for engineering applications.

The mathematical approach has been addressed from both deterministic and statistical perspectives. From the deterministic perspective, decision boundaries are shaped through iteratively classifying training data. No account is made for statistical properties in the data. Deterministic algorithms include distance-based classifiers, the perceptron approach, and most clustering techniques[72].

Classifying patterns based on the minimum distance to a neighbor or to a prototype was one of the first pattern recognition concepts. This simple method is most useful when classes are reasonably separated in the pattern space. Because they form piecewise linear boundaries in the multiprototype case, these classifiers are termed "linear machines."

Another deterministic pattern recognition method derives decision boundaries based on potential functions. An energy function is created to be minimal when all patterns are classified correctly. The function is optimized through gradient descent or other optimization techniques. The perceptron algorithm and the least mean square error algorithm are two well-known examples.

Statistical methods assume some randomness and variability in the data.

The best known are based on Baysian and Neyman-Pearson strategies[19] [73] [14]. These methods are optimal in the sense that on average, they minimize the probability of misclassification. Mean and covariance matrices comprise parameters that fully identify these models.

Baysian theory is probably the most commonly applied statistical pattern recognition technique. In statistical pattern recognition, accounting for statistical properties of the pattern classes minimizes on average the probability of making a classifying error. The method is effective when the data forms normal, statistical distributions. A covariance matrix (matrix of standard deviations) and a mean matrix determine the decision function parameters. Both are calculated from teacher sets of data.

Pattern recognition is oriented towards solving practical problems and currently has many real applications. The pattern recognition methods mentioned here are most reliable for data with distinct classes that form well-separated clusters of data. Theoretically, that is what results from HF-modeling. Therefore, at present, it seems that these basic pattern recognition tools are well suited for this application.

5.3 Neural Networks

With the lofty goal of achieving human-like performance, scientists have studied artificial neural models since the mid-1950s. These neural networks are massively parallel computational models designed to emulate what is known

about biological neural systems. Recently they have received much attention because very encouraging results have been achieved from fairly simplistic models. Applications in text-to-speech conversion, handwriting recognition, and other advanced pattern recognition problems have been successful.

Although artificial neural networks are today an entire area of study within Artificial Intelligence (AI), neural nets originally grew from pattern recognition. In 1957, Frank Rosenblatt introduced the perceptron algorithm, a very simple model based on the reward-and-punishment concept of learning[64]. During the late 50s and early 60s, Rosenblatt and many others worked diligently with neural networks. However, in 1969, Minsky and Papert published a proof showing that the potential application of these networks was limited to linearly separable problems[45]. As a result, almost all of the neural network research was put on hold. Then in 1979, Rumelhart, McClelland, et al. introduced a solution to the so-called credit assignment problem, the major restricting factor in Minsky's proof. This solution fostered renewed attention in neural computing. Neural nets have since grown from a small subdivision of AI in the early 80s to an entire field of study; entire conferences and journals are currently dedicated to neural networks.

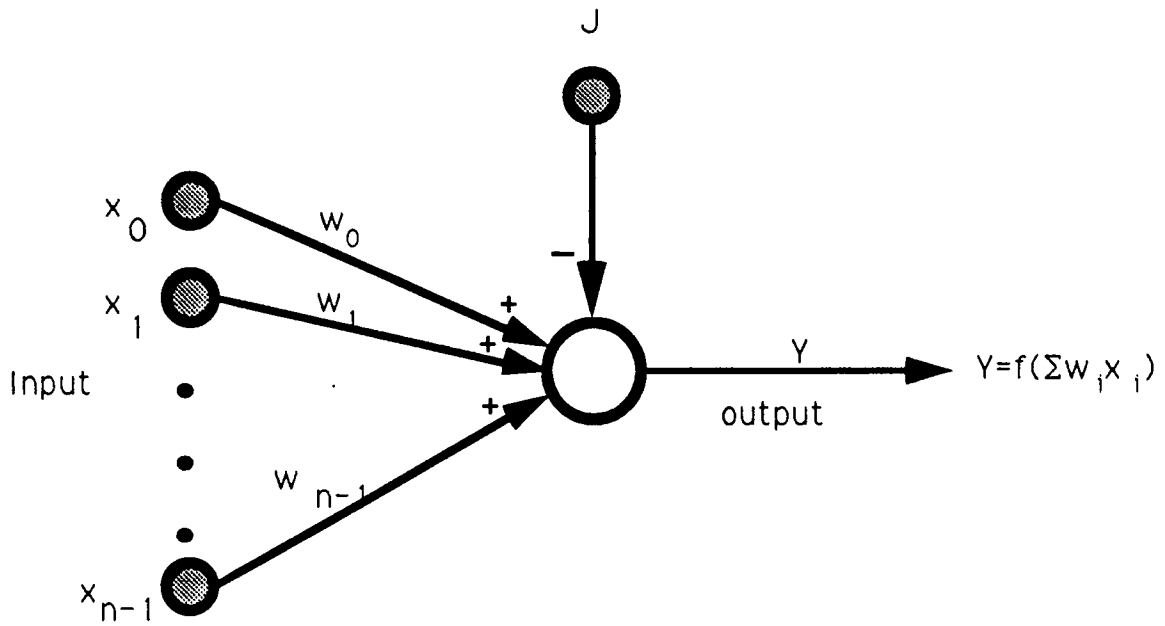
Because of their massively parallel structure, neural networks store information in a distributed-associative memory. A distributed memory stores each piece of information partially across all memory units. An associative memory's output corresponds to the nearest match stored in memory, even when presented with a partial or incomplete input. These distributed asso-

ciative memories can generalize, giving a reasonable response when presented with noisy or incomplete data, and are fault tolerant to memory damage. Unlike conventional AI methods, neural networks are not programmed but learn from example and continually adapt to new situations.

Because of these special properties, neural networks have a wide range of useful applications. Neural networks have proven adept at many advanced pattern recognition tasks, including text-to-speech conversion, natural language processing, handwriting recognition, and image compression. Potential chemical engineering applications include signal processing, adaptive filtering, fault detection and diagnosis, interpretation of spectral data, and nonlinear steady-state and dynamic modeling.

5.4 Neural Network Architecture

The building blocks of neural networks are nonlinear processing elements. Figure 5.2 illustrates a single nonlinear element. The node's output, Y, is a nonlinear function of the weighted sum of all inputs. This nonlinearity can be any function ranging from a hard-limiter (step function) to a sigmoid function. A neural network comprises anywhere from 20 to 1000 (sometimes even more) of these elements or nodes, interconnected by the weighting factors in massively parallel fashion analogous to the synapses of the brain. It has been claimed that the network's power as a whole is greater than the sum of its parts, a property of the nonlinearities and the massive parallelism of the



X = input vector
W = adjustable weighting vector
Y = node output
J = threshold
f() = nonlinear function (eg. step, sigmoid)

Figure 5.2: Nonlinear element

networks.

Many neural network architectures have been designed since the early 1950s. MADALINE, multilayer perceptron, counter-propagation, and binary associative memory are just a few examples. Although their names have little similarity, their basic elements are the same: nonlinear processing elements. Basically, neural models vary in three ways: 1) the learning rule that adjusts weighting factors, 2) the interconnections between processing elements, and 3) the nonlinearities in the transfer functions of the processing elements. Each of these plays a part in determining the overall characteristics of the neural model. The following sections review each of these aspects.

5.4.1 The learning rule

The single most important factor in determining the network characteristics is the learning rule. These algorithms adjust weighting factors to yield the desired network response. Learning algorithms fall into two categories: supervised and unsupervised. Supervised learning adjusts weighting factors based on the difference between actual network output and desired network output (or teacher data). Examples of supervised learning include delta rule, Widrow-Hoff, and stochastic learning. In unsupervised learning, only input and local information are used to adjust weighting factors; there is no comparison to teacher output. Unsupervised learning is used primarily for self-organization of data. The two most common types of learning are Hebbian learning and competitive learning.

The most common supervised learning method is delta rule learning. This learning algorithm adjusts weighting factors in proportion to the processing element output error. To help smooth out changes and reduce training time, other terms are often added, such as momentum. The most common form of delta rule learning is

$$\Delta W_{ij_{new}} = \alpha * error * X_{ij} + \beta * \Delta W_{ij_{old}},$$

where ΔW_{ij} = change in weight ij,

X_{ij} = input ij,

and α & β = learning parameters.

The procedure for implementing this rule is clear for the top layer but not for the middle and bottom layers. This is the credit assignment problem that for many years limited the usefulness of multilayered networks. The solution to this problem, introduced by Rumelhart[65], was relatively simple: the error of each node in the hidden layers is a weighted sum of the errors in the successive layer. In the same manner as an input propagates from the input to the output layer, an error "backpropagates" from the output to the input layer. This backpropagation method is quite effective; its only drawback is the extremely long learning time.

The two most common forms of unsupervised learning are Hebbian learning and competitive learning. These methods self-organize a data set; in other words, elements having similar properties from a data set are mapped

into similar outputs. The resulting algorithms are analogous to clustering techniques of classical pattern recognition. Hebbian learning was named after Donald Hebb, who described a method by which neurons might learn: when synaptic input and neural output are both active, the strength of that connection is enhanced. In other words, neural pathways are reinforced each time they are used, similar to processes that occur in biological neural systems. The simplest mathematical formulation of Hebbian learning is

$$\frac{\partial W_{ij}}{\partial t} = -W_{ij} + A_i * B_j,$$

where W_{ij} is the strength of the connection between nodes A_i and B_j. Extensions of this rule include signal Hebbian learning and differential Hebbian learning.

Competitive learning was initiated by Teuvo Kohonen in the mid-1970s. Kohonen's work was motivated by studies that examine the mapping of information received at the sensory organs into areas of the brain. In this network, each node sends a positive (excitory) signal to itself and a negative (inhibitory) signal to all other nodes in same layer. Eventually only one node with the greatest activity will be active for a given input. Inputs with alike characteristics will activate the same output node.

5.4.2 Nodal connections

The primary important factors in forming network characteristics are interconnections between nodes (processing elements) and the number of nodal layers. Interconnections determine such characteristics as whether the network learns through example or self-organizes data and whether information flow is unidirectional or bidirectional. The number of nodal layers and the number of nodes in each layer bound the network's ability to solve complex problems.

In connecting the nodes of a network, each processing element can be connected to any other processing element, including itself (see Figure 5.3). Connections that propagate signals towards the output layer are called feedforward connections; those that propagate signals towards the input layer are called feedback connections. Most of the time, feedforward and feedback signals connect adjacent nodal layers; however, there are exceptions. The final class of connections are lateral connections, which join nodes in the same layer. Any of these connections, feedforward, feedback, or lateral, can be either excitory or inhibitory. Excitory signals add to a node's activity, while inhibitory signals reduce a node's activity.

Each neural model is a unique combination of nodal connections. All neural models have feedforward connections, an obvious necessity if information is to propagate to the output. Some, like the multilayer perceptron, are fully-connected, feedforward-only networks. Others, such as the bidirectional associative memory (BAM), employ feedforward with feedback. Information in these feedback networks will reverberate across layers until some convergence

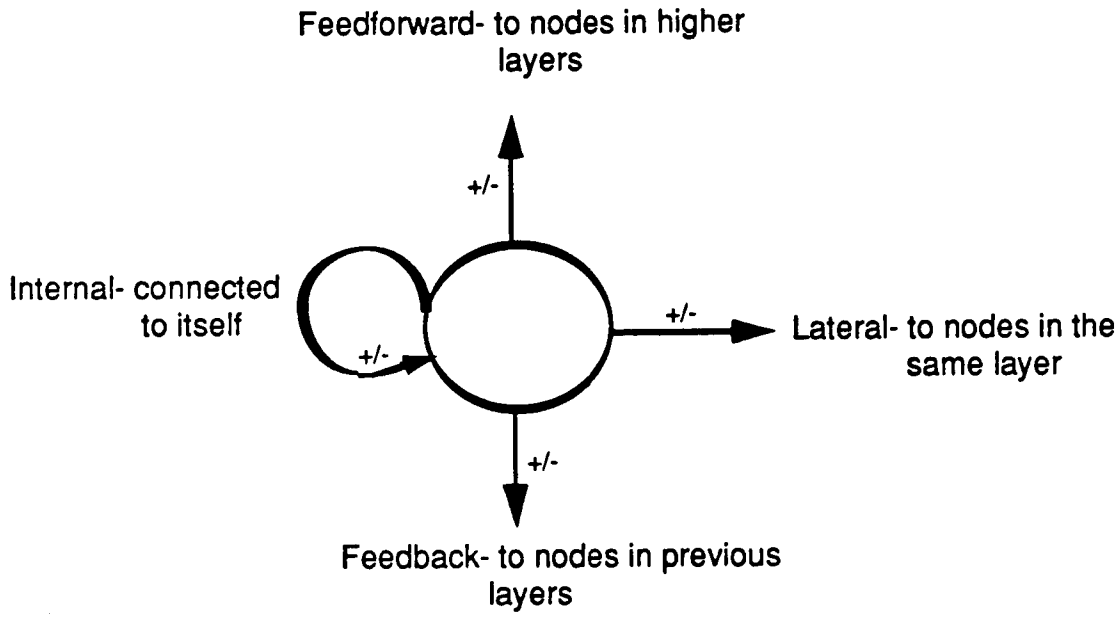


Figure 5.3: Possible processing element connections

criterion is met. Lateral connections are characteristic of the competitive networks, for example Kohonen's self-organizing network. Because of the training difficulty, no network employs fully connected feedforward, feedback, and lateral connections.

Including multiple layers in a network is key to many important properties, as demonstrated geometrically in the following two-dimensional example. By building a small three-layer network one layer at a time, this example examines the ability of a feedforward network with a hard-limiter (step function) transfer function to classify two-dimensional patterns. At each step the potential decision boundaries are discussed. The reasons for multiple layers will become clear.

A single processing element with a hard-limiter (output 1 or 0) linearly separates a two-dimensional plane into two regions (see Figure 5.4). A high output results when the input is taken from region A, a low output when taken from region B. The slope of the boundary separating these two regions is determined by the weighting factors from each connection. Such a boundary distinguishes between two classes of inputs and is called a decision boundary. More linear decision boundaries can be added to the two-dimensional plane by adding more first-layer nodes to the network. Figure 5.5 illustrates the result when more nodes are added to the first layer of the network.

Adding a single node in the second layer allows the network to form intersections and/or unions of the regions (half-planes) created by the first layer. Figure 5.6 shows one possible defined region. Additional second-layer

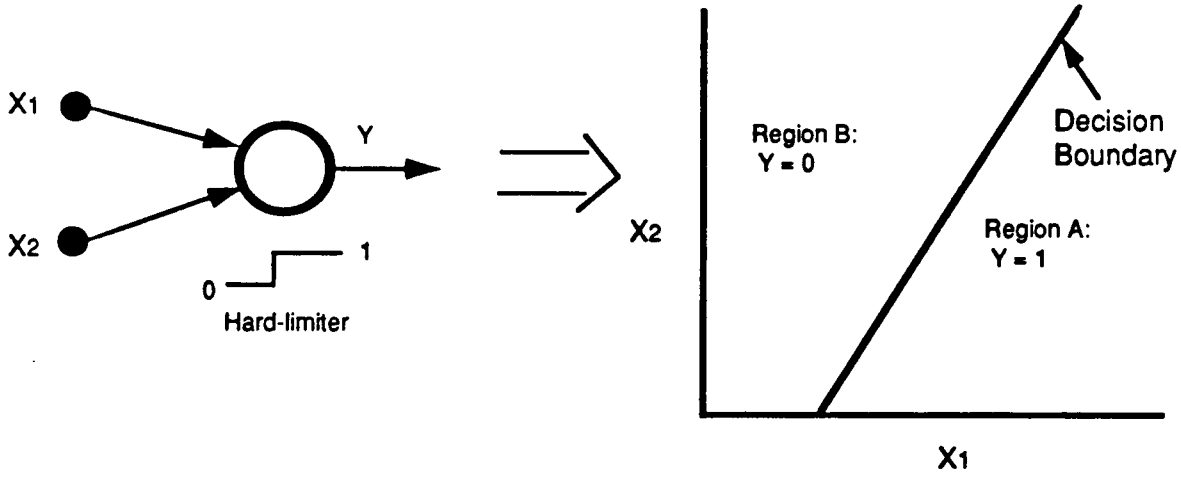


Figure 5.4: Single element decision boundaries

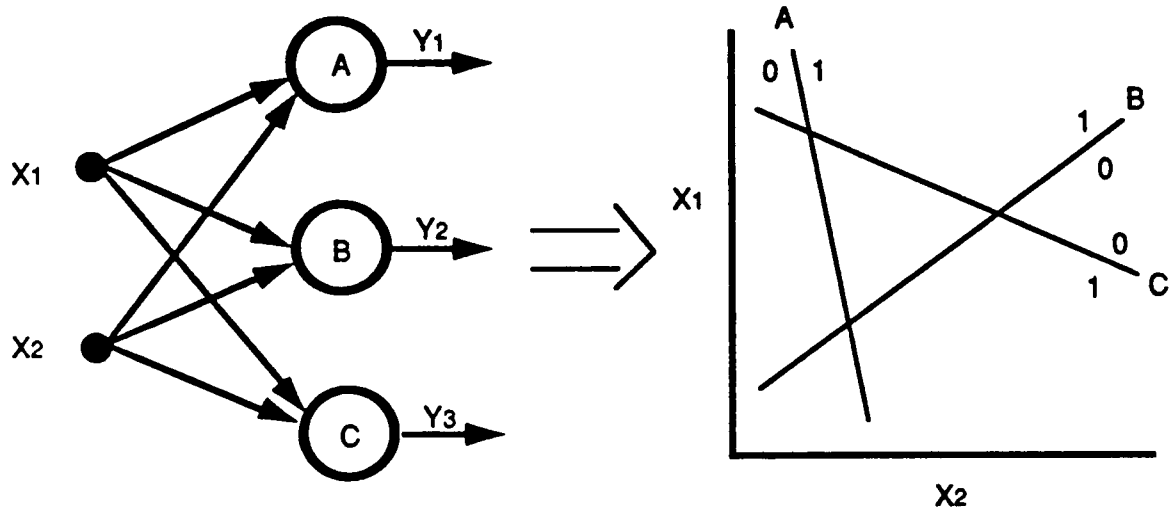


Figure 5.5: Decision boundary for three first-layer nodes

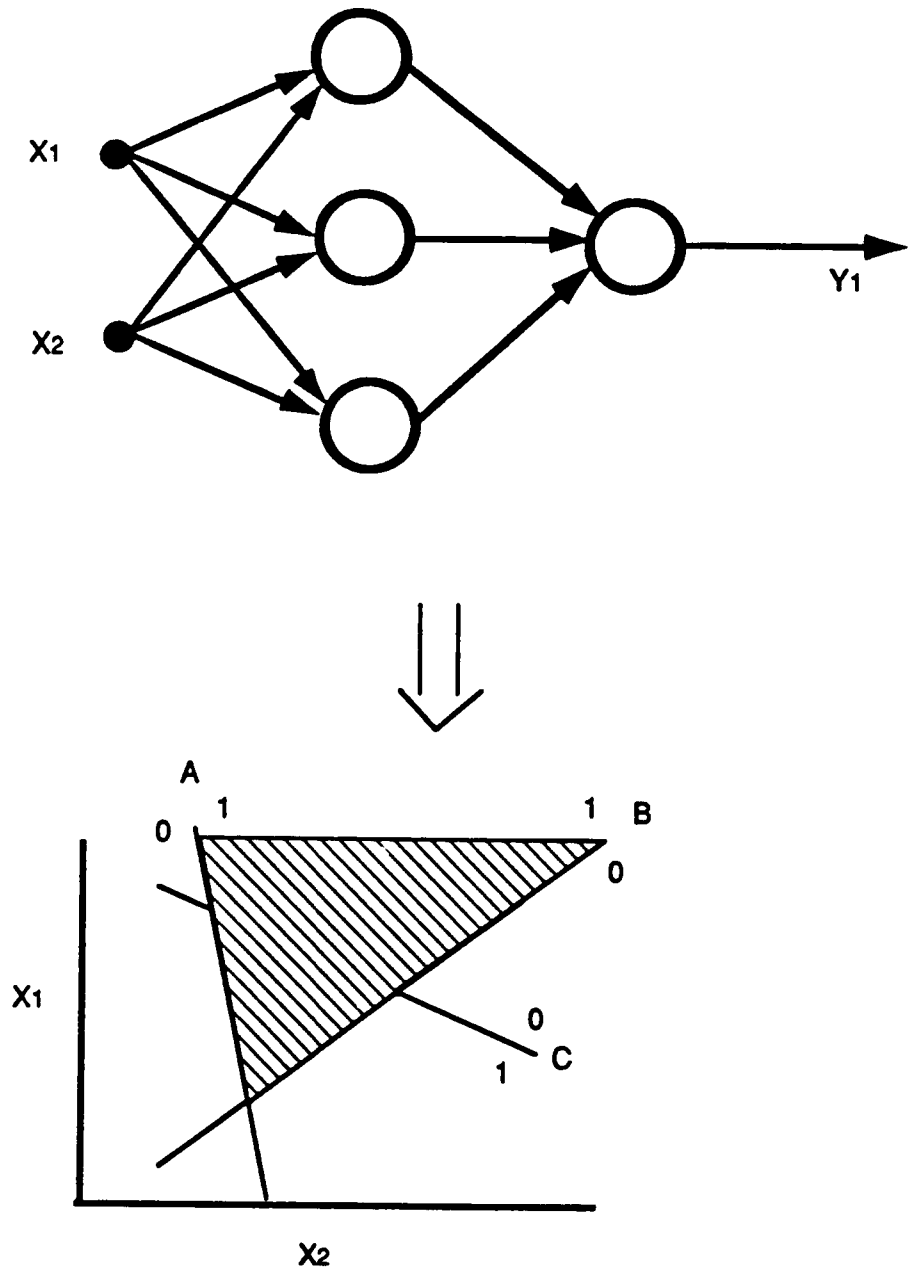


Figure 5.6: Decision boundaries for single second-layer node: piecewise linear boundary

nodes define more of these regions (Figure 5.7). The decision boundaries resulting from a two-layer network are piecewise linear and can define enclosed or disjoint regions.

Adding third-layer elements creates even more complex boundaries forming the union and/or intersection of regions formed by the second layer (Figure 5.8). Such three-layer networks can draw arbitrarily complex, piecewise-linear decision boundaries. If the network consists of enough nodes, theoretically any decision can be drawn, no matter how complex. Unfortunately, learning algorithms limit the potential of the networks.

Several important points are evident from this simple analysis.

- The number of nodes in the initial fan-out layer determines the potential complexity or number of pieces in the final decision boundary.
- Each layer forms regions that are intersections or unions of the regions formed in the preceding layer.
- The second layer creates arbitrarily complex, piecewise-linear boundaries, including disjoint and enclosed regions.
- A third layer improves the efficiency of the network in forming the desired boundaries; fewer overall nodes are required than with two-layer networks.
- A different nodal-transfer function such as a sigmoid function will create slightly different, perhaps nonlinear, boundaries.

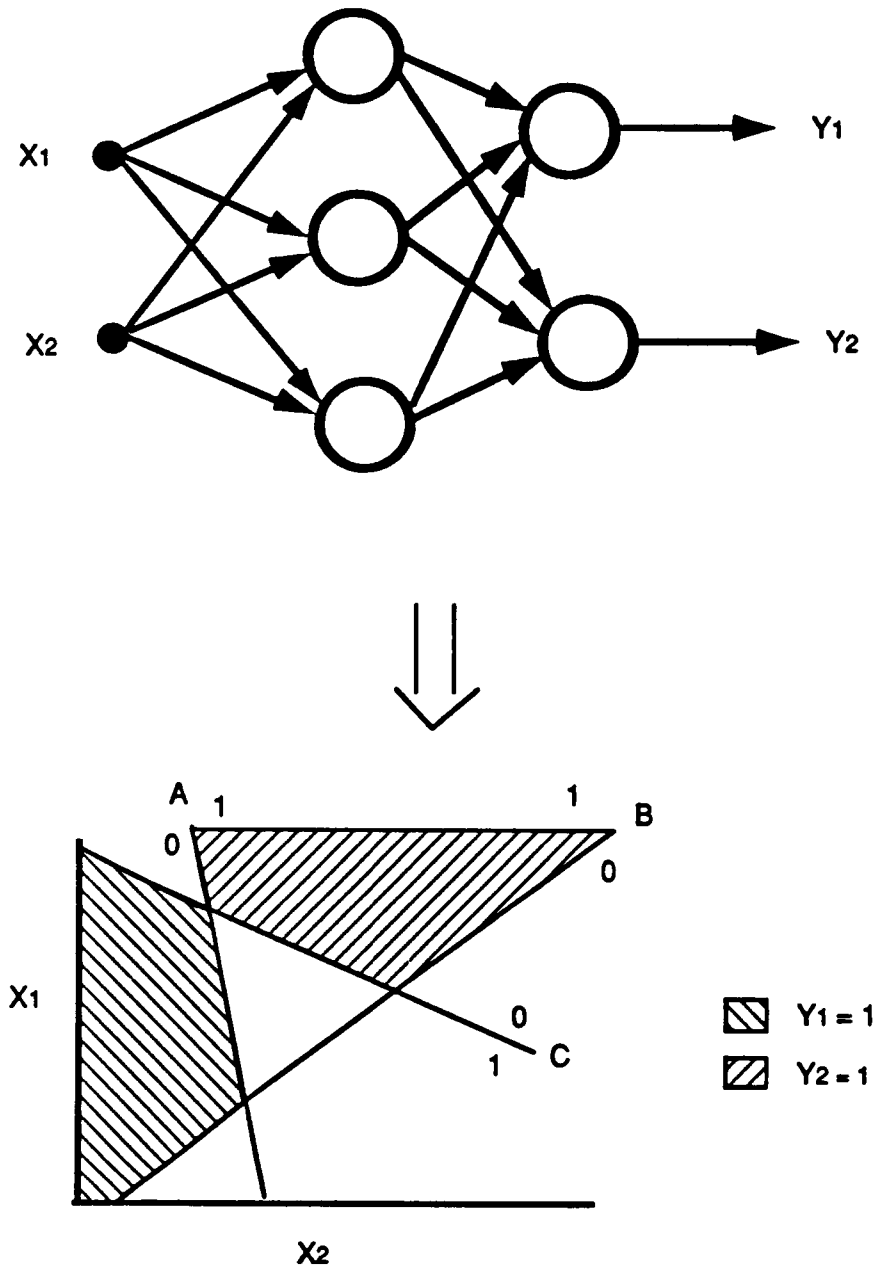


Figure 5.7: Two-layer,two-input network forms two piecewise linear regions

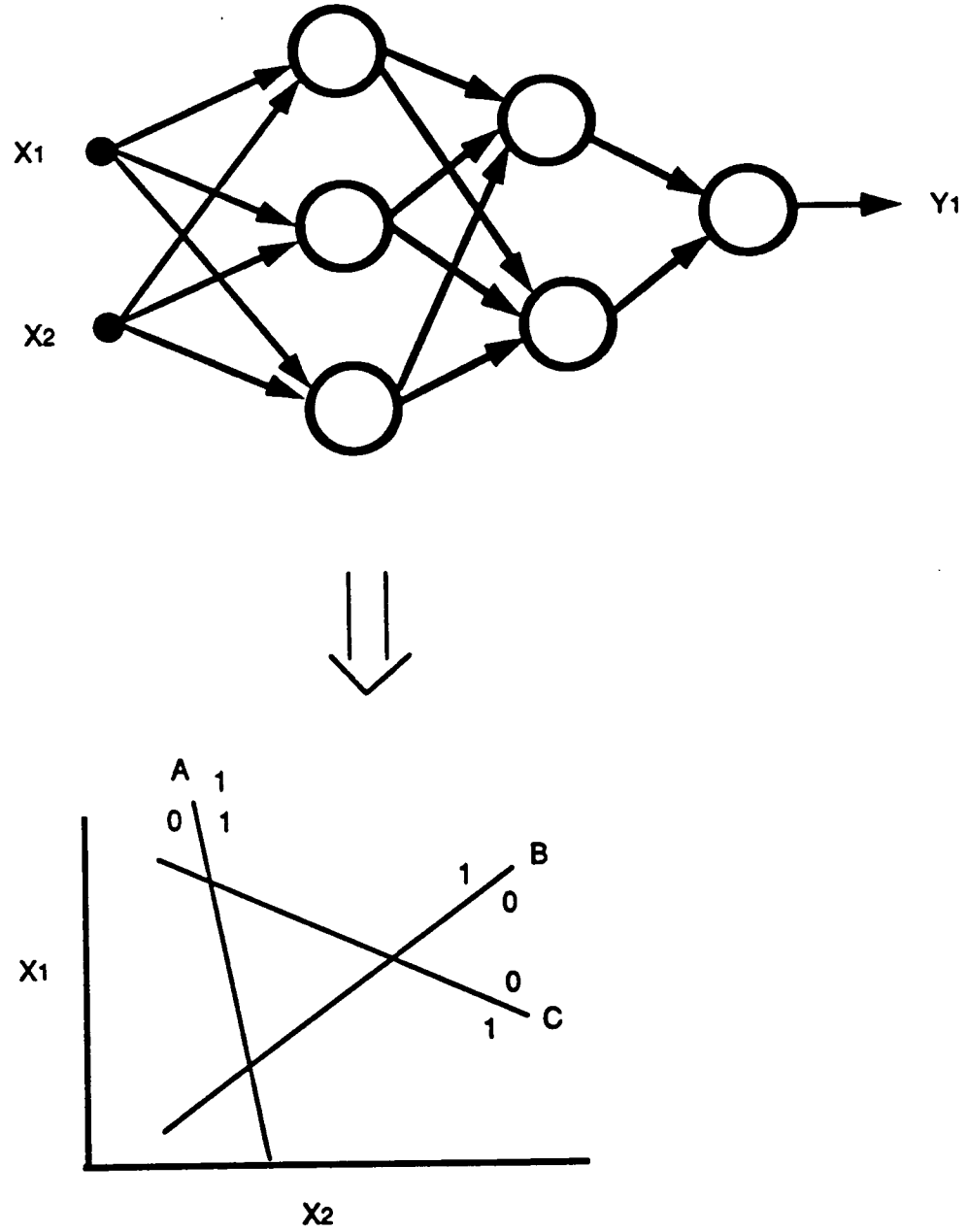


Figure 5.8: Three-layer, one-output network creates any region bounded by lines A,B, and C

5.4.3 Nodal transfer functions

The transfer function or activation function computes the output of each node from the summation of nodal inputs. These functions are the primary source of nonlinearities in neural networks. In the previous example, the activation function was a simple two-valued step that led to a partition of the space into distinct regions, equivalent to AND/OR gating in digital logic. A relatively simple extension of this concept allows the outputs to range continuously over some interval. Such an approach might be viewed as a "degree of confidence"[1] and similar to a degree of membership or conditional probability used in fuzzy logic[82]. The discussion that follows describes one commonly applied class of such activation functions.

The step function or threshold function produces a two-valued output (zero or one) based on the magnitude of activation (the input). This function is a natural for classifying patterns as belonging or not belonging to a class. A second activation function, a subthreshold summation is observed in physical and biological systems. Only when the input exceeds a threshold, T, does output become nonzero. For inputs above T, the output is a linear function of the input. Because all physical systems have a limited dynamic range, the output is constrained at a maximum value.

A commonly applied sigmoid function is a continuous function that combines the threshold properties along with an approximately linear region.

The sigmoid function is described mathematically by

$$f(X) = \frac{1}{1+e^{-(\alpha X - T)}},$$

where X = sum of node inputs

T = threshold, and

α = gain.

The gain term, α, is a "slope" of the sigmoid curve and is adjusted to improve the learning rate. The biggest advantage to the sigmoid function is that it is a continuous function, a necessity for many learning algorithms.

This section described essential pieces that comprise neural networks: learning rules, nodal connections, and nonlinear activation functions. The following section details one particular set of these elements that forms a commonly applied network architecture: the multilayer perceptron with back-propagation learning.

5.5 Multilayer Perceptron Network with Back-propagation Learning

This network is, by far, the most widely used network in practical applications. It has been applied successfully in many areas, including signal processing, signal coding, nonlinear regression analysis, nonlinear modeling, and pattern recognition. Its strengths as a pattern classifier are applied in this work.

Figure 5.9 depicts the multilayer perceptron network. A basic feedforward-only structure, this network contains at least one "hidden" layer between input and output layers. Notice that the connections from each layer are fed directly to the next layer. Training is accomplished in a supervised fashion. Effective training of hidden-layer weights through back-propagation allows the network to overcome many of the limitations of single-layer networks.

When presented to the input buffer, an input propagates upward through the network to the output layer. Each output may be continuous valued (between 0 and 1) or discrete valued (0 or 1), depending on the application. Typically in process-modeling applications, each output node is continuous and corresponds to an output variable. As a pattern classifier, each output node represents a particular pattern class and can only have a value of one or zero.

The number of nodes in the input and output layers needed for a given application is usually constrained by the application. For example, in pattern classification, the input vector's dimension equals the size of the input layer, and the number of pattern classes equals the number of output nodes. However, determining the optimal number of hidden-layer nodes is not so straightforward. Too few hidden-layer elements prevents the network from ever correctly mapping inputs to outputs, while too many elements impedes the ability of the network to generalize[1]. Too many nodes results in a lookup table effect rather than focusing on important features in the data. A theorem by Kolmogorov[42] helps some in this regard. His theorem effectively states that a three-layer perceptron network with $N(2N+1)$ nodes can describe any func-

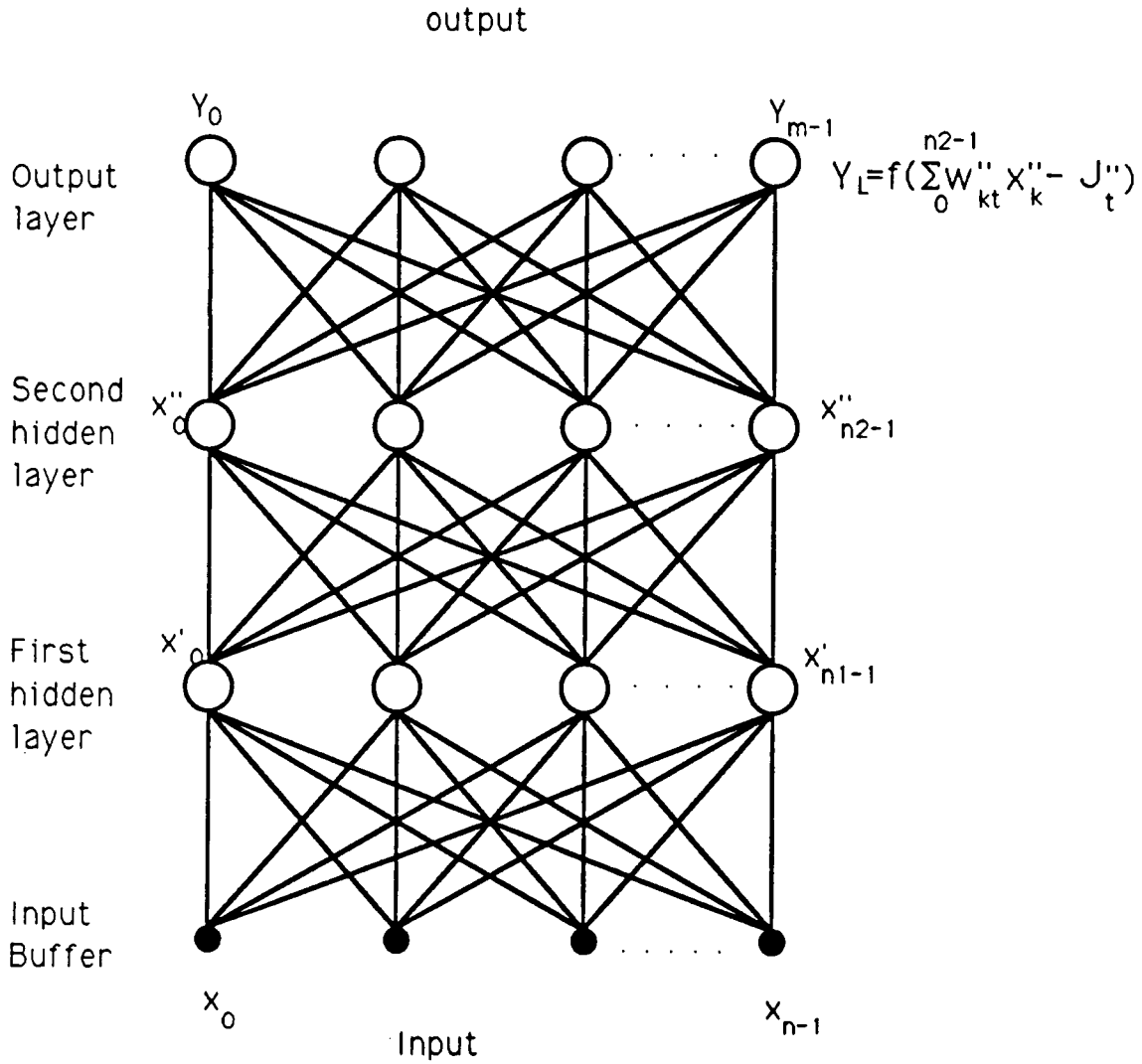


Figure 5.9: Multilayer-perceptron network

tion of N variables. Such a large number of hidden-layer nodes is usually much larger than is practical for most applications. A good rule of thumb states that $2I+1$ hidden-layer elements (where I = number of input nodes) is a reasonable number for most problems.

The back-propagation learning algorithm requires that the nodal activation functions be continuous. Almost without exception, the sigmoid function is used. It should be noted that the nonlinear characteristics and many of the important properties of multilayer perceptron networks stem from these nodal nonlinearities. A single-layer network could potentially duplicate the performance of a multiple-layer network when linear functions are used[42].

Back-propagation is a form of delta-rule learning (see Section 5.4). This algorithm is an iterative gradient descent algorithm that attempts to minimize the mean square error between the actual output and desired output. The following summarizes the steps in back-propagation.

1. Randomly initialize all weighting factors.

2. Present a single input vector, $(X_1, ...X_N)$, from a set of training data to the network and compute the network's output.

3. Compare the desired output, d_i, to the actual output, y_i, and calculate the error, e_i, by the following:

$$e_i = y_i(1 - y_i)(d_i - y_i) \text{ for output nodes;}$$

$$e_i = x_i(1 - x_i)_k e_k w_{ik} \text{ for hidden layers;}$$

where k is over all nodes in the layer above layer i.

4. Adjust weights by $w_{ij}(t+1) = w_{ij}(t) + ne_j x_i$,

n=gain term

w_{ij}=weighting factor from node i (lower layer) to node j (upper layer),

x_i=input from node i (lower layer).

To attain acceptable performance, the back-propagation algorithm must iterate hundreds, even thousands, of times through the training data.

The following example demonstrates the multilayer perceptron network with back-propagation learning as a pattern classifier. It sheds light on the network's ability to draw complex decisions and its characteristics as a pattern classifier.

5.5.1 Decision boundary studies of a multilayer perceptron network

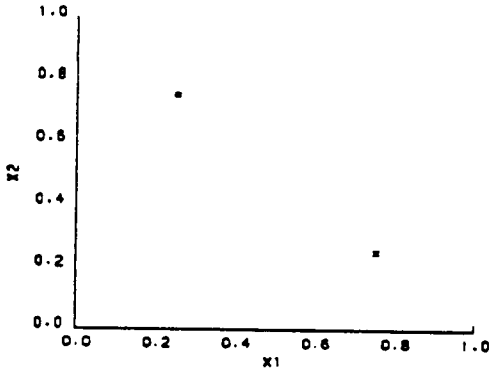
The following example gives insight into the multilayer perceptron network's general characteristics, including the form of decision boundaries it can

draw and the effects of each learning parameter. In this example, the multilayer perceptron network consists of a two-element input buffer, two hidden layers of eight elements each, and a two-element output layer. The size of the input buffer corresponds to the number of values per pattern; in this case, the dimension is two. The number of hidden-layer elements is a compromise between drawing complex decision boundaries and minimizing learning time. The output layer corresponds to the number of pattern classes.

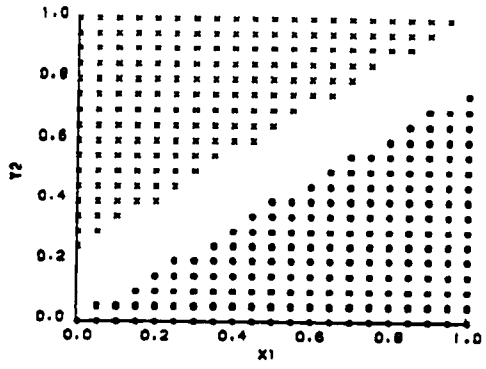
Figure 5.10 demonstrates how decision boundaries are formed. The network is first trained with teacher data sets, each consisting of one or more prototypes representing each class. In Figure 5.10 A, only one teacher data point for each class trains the network. Then, after training, the network classifies a matrix of points distributed evenly through the pattern space. Figure 5.10 B shows how the network classifies the entire two-dimensional plane. The resulting decision boundaries shown in Figure 5.10 C become apparent.

In Figure 5.10 the network draws linear decision boundaries. Any pattern lying in the lower right section of the pattern space belongs to class X; any pattern in the upper left belongs to class O. Notice the diagonal region between the two classes. This "null" class, neither class X nor class O, acts as a buffer between classes. Its size is a function of convergence parameters.

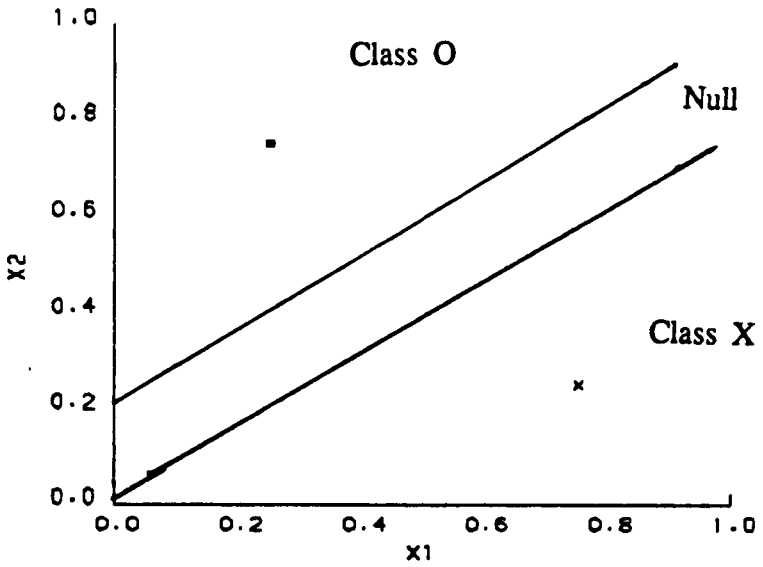
Figure 5.11 depicts decision boundaries that result in a two-class, multiprototype case. Here identical numbers of prototypes represent each class. Figure 5.12 demonstrates that resulting decision boundary remains unchanged when an unequal number of prototypes represents the classes.



A) Training data

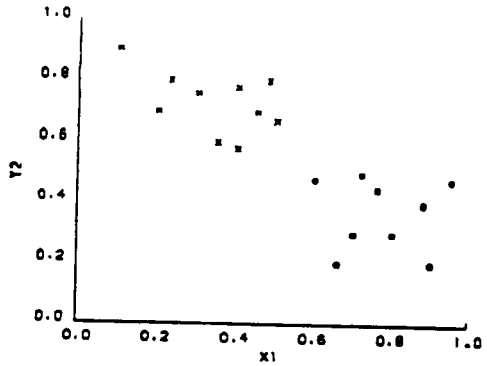


B) Classification of the 2-D space

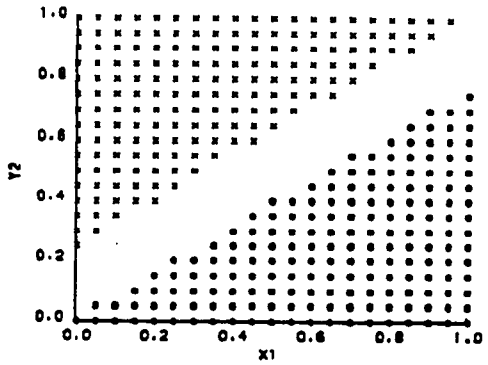


C) Resulting decision boundaries

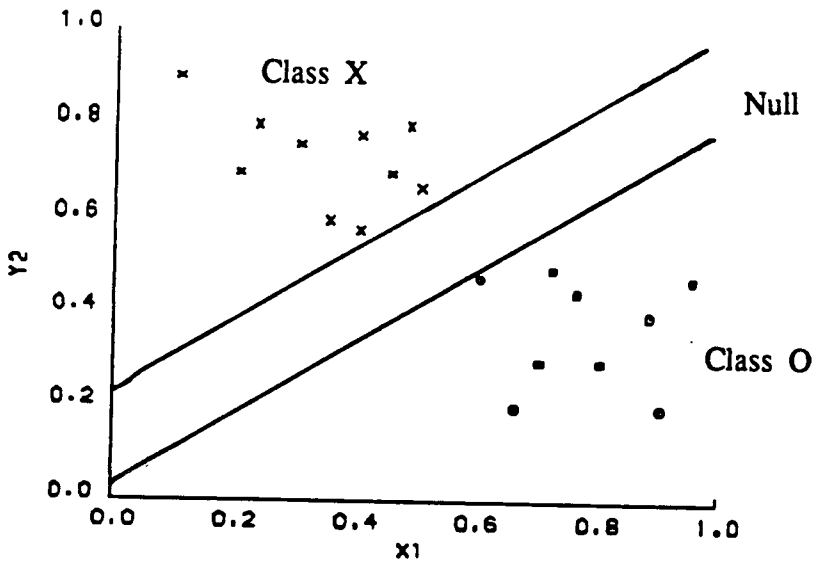
Figure 5.10: Decision boundary determination



A) Training data

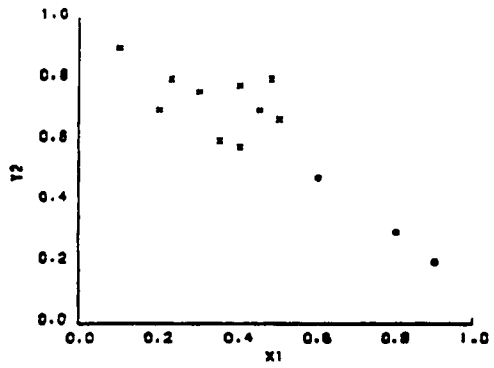


B) Classification of the 2-D space

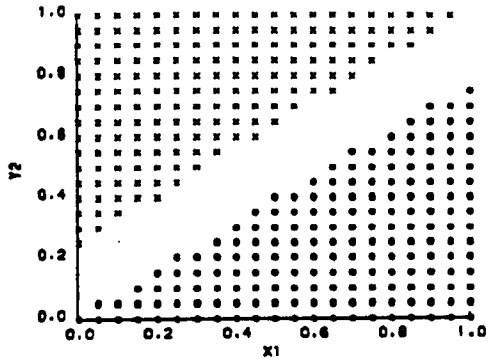


C) Resulting decision boundaries

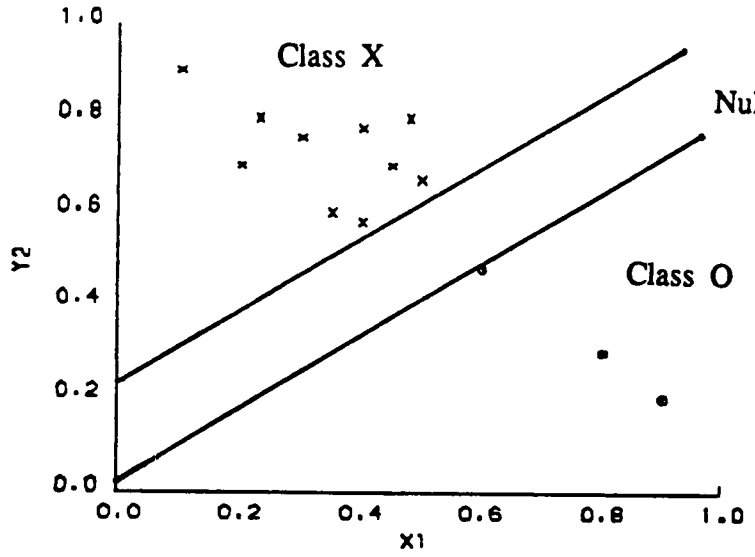
Figure 5.11: Decision boundary for multiple prototypes



A) Training data



B) Classification of the 2-D space



C) Resulting decision boundaries

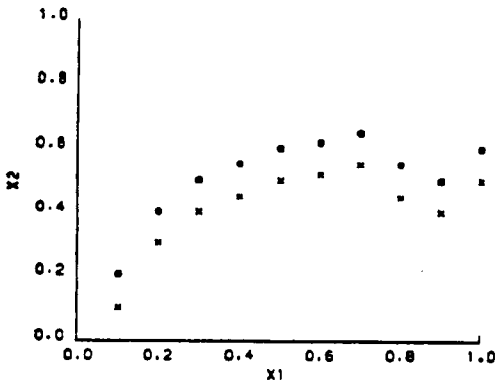
Figure 5.12: Decision boundary for an unequal number of prototypes

The network can also draw nonlinear decision boundaries, as Figure 5.13 demonstrates. An important point concerning tightness of convergence is shown here. Convergence constraints referred to here include the overall percentage of teacher data required to be correctly classified before training can end and the threshold value above which an output, Y, is considered high. In Figure 5.13, relaxed convergence constraints result in a boundary that does not completely separate training data from the two classes. Convergence constraints are tightened in Figure 5.14; the resulting decision boundary has sharper curves that differentiate completely between the classes. Thus, tighter convergence is desired in this case.

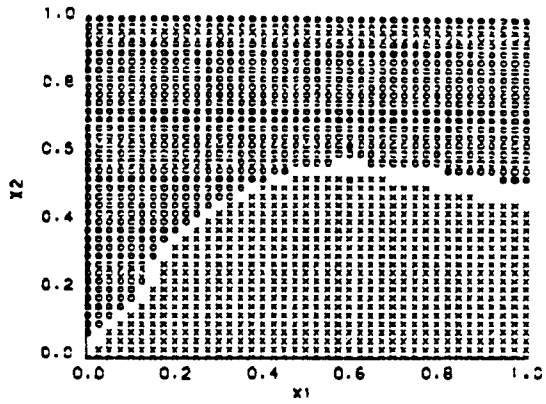
Tight convergence is not always desired, however. A more realistic two-class example is shown in Figure 5.15, which depicts two overlapping circular clusters of prototypes: one for relaxed constraints (Figure 5.15), and one for tight constraints (Figure 5.16). The resulting boundaries in Figure 5.16 (tight constraints) are not very useful. In most cases in which problem classes share a portion of the pattern space, tight convergence results in highly complex decision boundaries that may have little meaning. Notice also that in this case, the null region between the two classes is desirable, giving an indication of a region in which classification is uncertain.

Figures 5.17 and 5.18 demonstrate the network's ability to draw disjoint regions. In Figure 5.17, the regions are linearly separable. Its ability to handle a class encircled by another is shown in Figure 5.18.

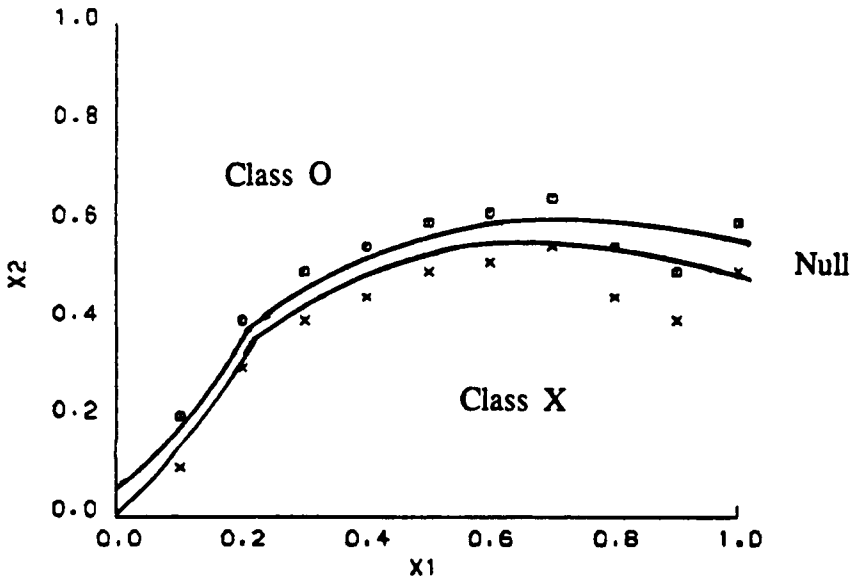
These classification studies demonstrate several important characteris-



A) Training data

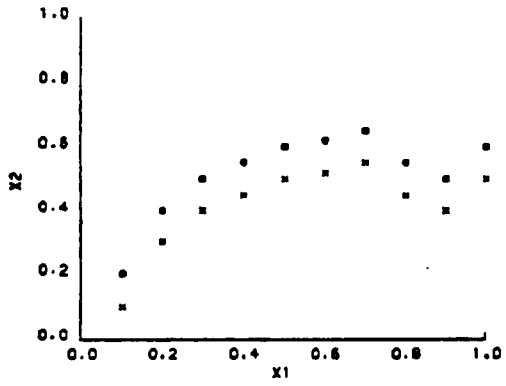


B) Classification of the 2-D space

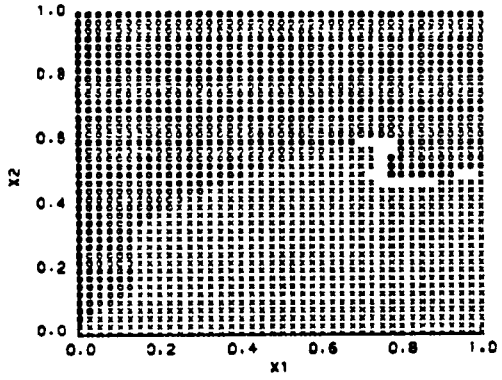


C) Resulting decision boundaries

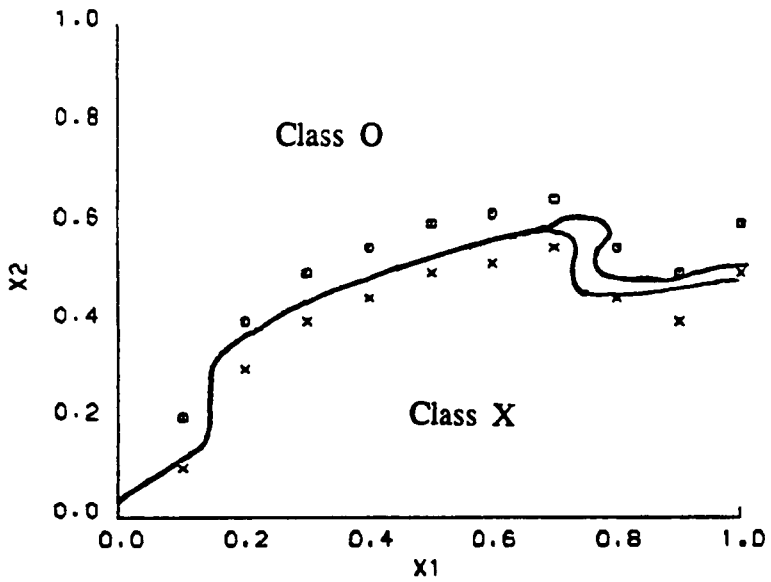
Figure 5.13: Nonlinear decision boundary



A) Training data

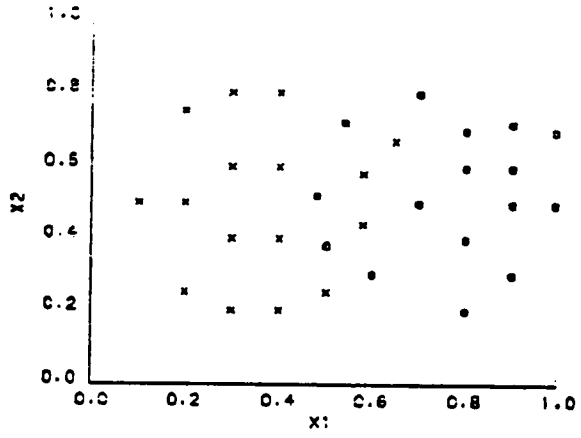


B) Classification of the 2-D space

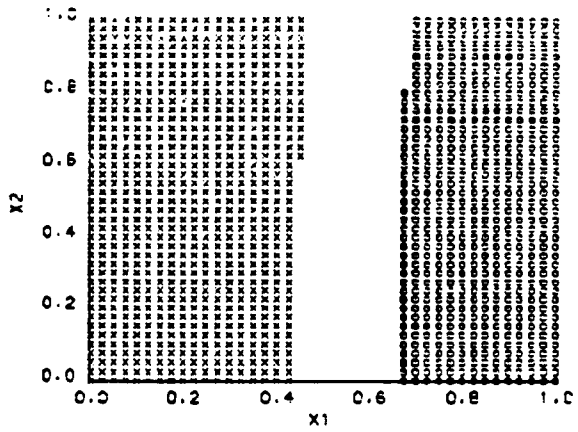


C) Resulting decision boundaries

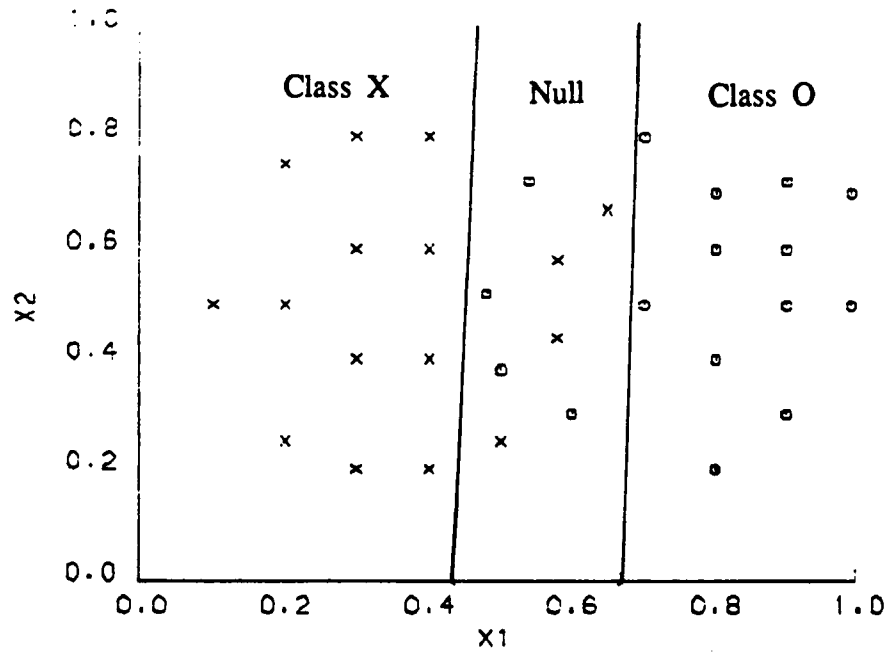
Figure 5.14: Nonlinear decision boundary- tight convergence constraints.



A) Training data

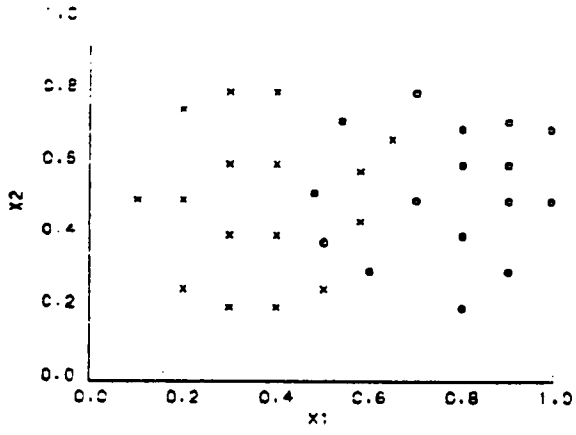


B) Classification of the 2-D space

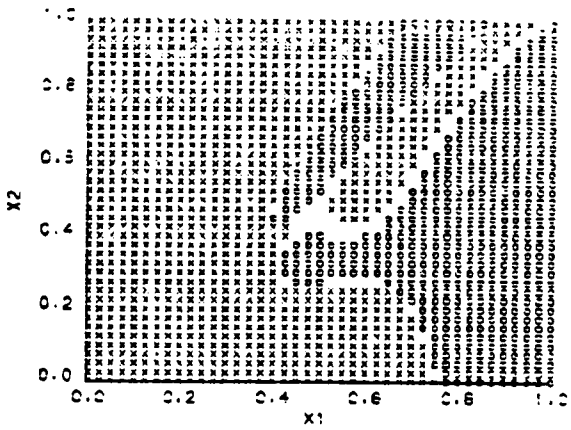


C) Resulting decision boundaries

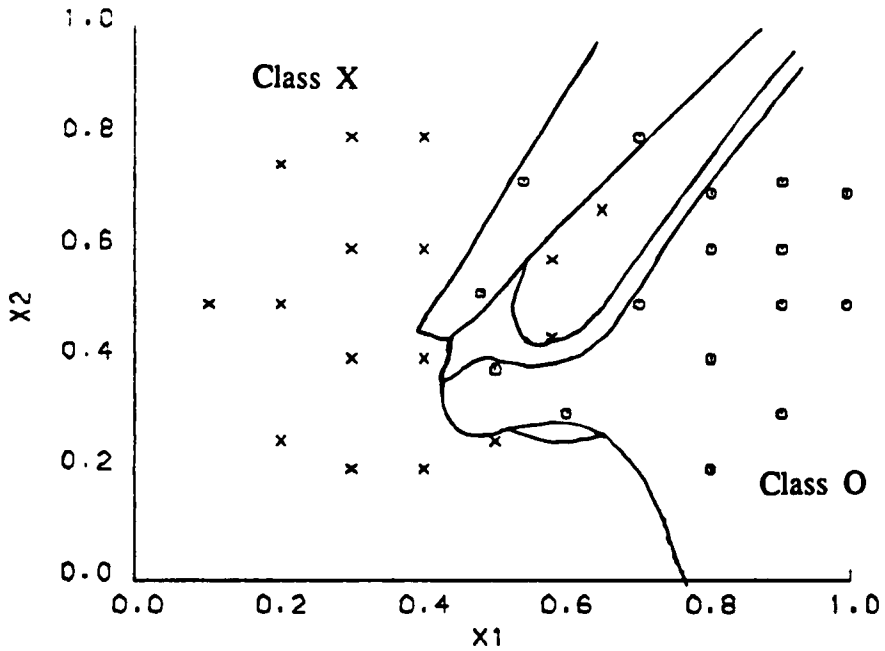
Figure 5.15: Decision boundaries for overlapping clusters



A) Training data

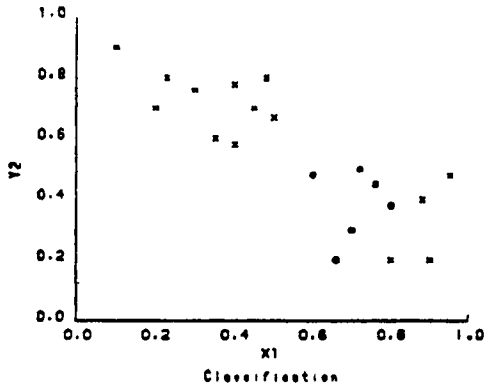


B) Classification of the 2-D space

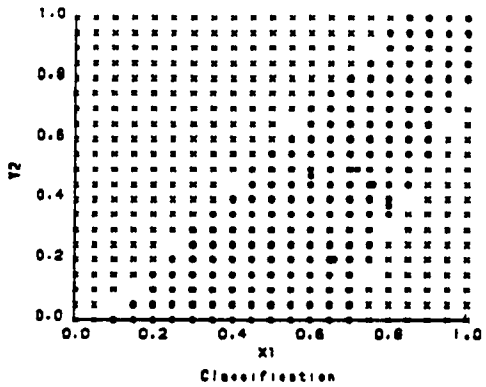


C) Resulting decision boundaries

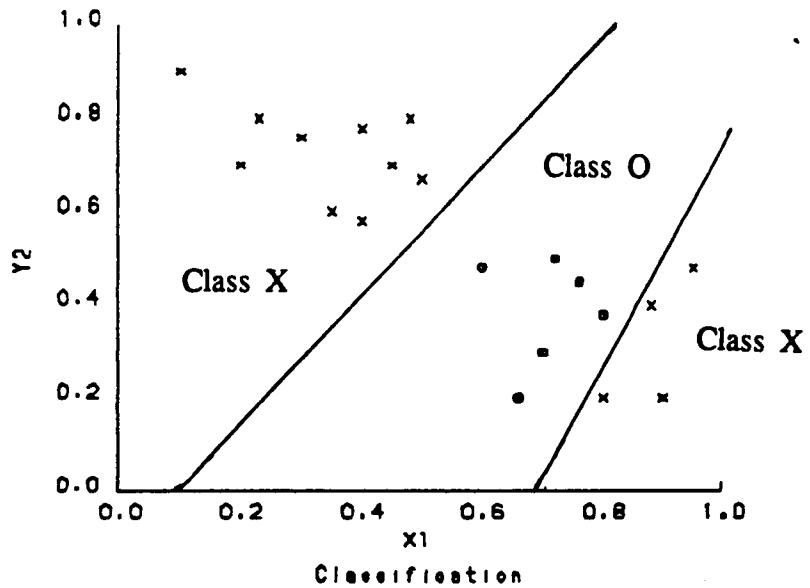
Figure 5.16: Decision boundaries for overlapping clusters- tight convergence constraints



A) Training data

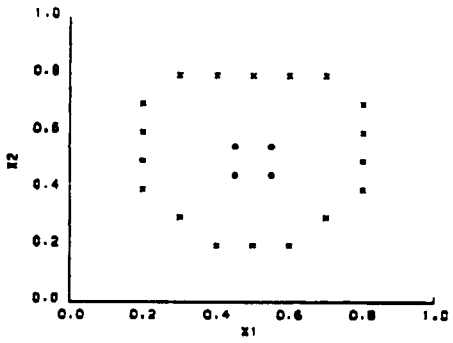


B) Classification of the 2-D space

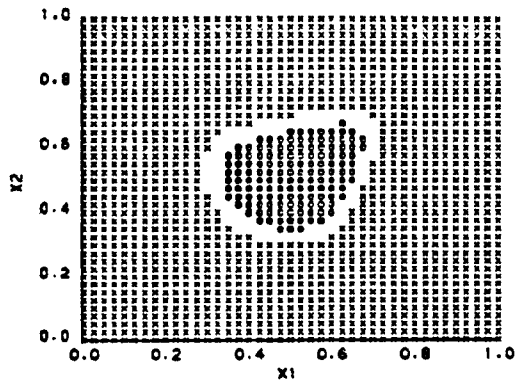


C) Resulting decision boundaries

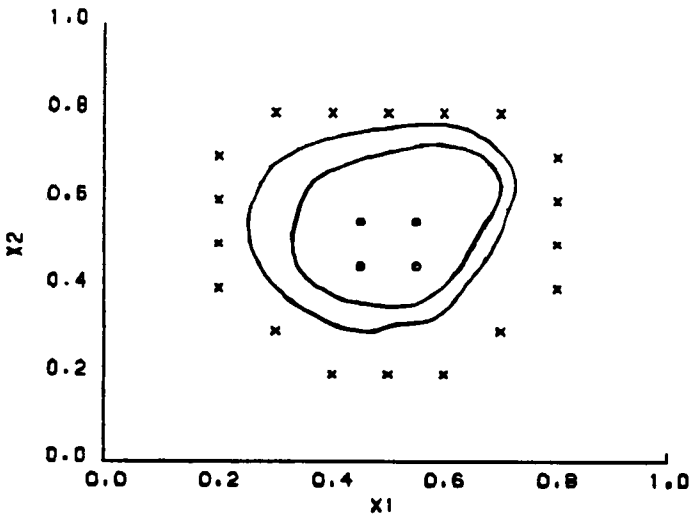
Figure 5.17: Linearly separable disjoint regions



A) Training data



B) Classification of the 2-D space



C) Resulting decision boundaries

Figure 5.18: Decision boundaries resulting from encircled data

tics of the multilayer perceptron network with back-propagation learning.

- The network first separates classes with a linear decision boundary, then draw curves as needed.
- A null region always separates the pattern classes. Its size is a function of the distance between class prototypes and the tightness of convergence constraints.
- Differences in the number of prototypes that represent each class will not affect the location of decision boundaries between them.
- The number of nodes in each layer of the network limits the complexity and sharpness of curves in the decision boundaries.
- The network can form enclosed regions or more than one distinct region for a particular class.
- When clusters of class prototypes overlap, it is best to accept that not all prototypes will be classified correctly by relaxing convergence constraints.
- If the network attempts to draw very tight curves in the decision boundaries, the learning algorithm becomes unstable.
- The network appears to draw pleasing decision boundaries between classes of teacher data. It is not yet known whether or not these boundaries are optimal in any sense.

5.6 Other neural networks of interest

Among the multitude of other neural network architectures that have been extensively tested, the Kohonen self-organization network and the counter-propagation network show the most promise in contributing significantly more than the multilayer perceptron network with back-propagation learning. Those two networks are briefly outlined in this section.

Introduced by Teuvo Kohonen in 1982[34] , the Kohonen self-organization map is a single-layer, two-dimensional network. The network is designed to map similar input patterns to the same region of the output plane, which results in grouping or clustering of like patterns in a data set. Because it employs unsupervised learning and, therefore input data alone is sufficient to train the network, this network is potentially valuable.

The learning algorithm applied in Kohonen networks is known as competitive learning. In this algorithm, processing elements compete with each other for outputs, the neuron and its neighbors that give the strongest response to a given input are strengthened. In this manner, similar inputs will result in similar two-dimensional patterns. As a result, the Kohonen network forms compressed representations, or topological maps, of complex data. In large, complex data sets which little is known of the information present, these topological maps group data and identify important features inherent in the data. For this reason, the concept of self-organization is very appealing.

A second potentially important network is the counter-propagation net-

work developed by Robert Hecht-Nielson[27]. This network is appealing because it combines the self-organizational capabilities of a Kohonen network and mapping capabilities of a Grossberg outstar (nearly identical to a perceptron layer). In addition, information flow is bidirectional; an input applied to a trained network yields an approximation of the output, and an output presented to the top layer yields an approximation of the input. Such a network has applications ranging from pattern recognition to statistical analysis to process control.

The counter-propagation network comprises a Kohonen layer feeding into a Grossberg outstar layer. Because of this, training is in two phases. First, competitive learning trains the Kohonen layer. In this network, only one neuron in the Kohonen layer is activated for a given input. Second, the Grossberg layer is trained by presenting simultaneously input to the bottom layer and output to the top layer. Because only one Kohonen layer node may be high, weighting-factor adjustment in the outstar layer is fast and does not require an iterative solution. See[27] for a complete description.

The counter-propagation network has two important features. First, it combines the positive attributes of two different networks. As neural networks evolve, they will likely become more and more specialized, each having strengths toward specific applications. Combining each specialized network in a larger structure will lead to more powerful networks. Second, the counter-propagation network is bidirectional. Inverting a nonlinear process (find a correct input given the desired output) is the focus of much attention within

process control. Networks like counter-propagation have the potential to accomplish such an inversion in a very short time.

CHAPTER 6

Enhancing System Cultivation Methods with Neural Networks

This chapter details how system cultivation tools work hand in hand. By learning through experience, the resulting framework examines data daily and gives the engineer feedback on system operation. Chapter 7 demonstrates the use of these tools with two simulated examples.

Figure 6.1 provides a general overview of the methodology. First, HF-modeling and TSD analysis process historical data. HF-modeling leads to a series of reduced information databases, while TSD analysis yields a series of n-dimensional surfaces indicative of dynamic characteristics. Then, to identify the present state of the process, neural networks and pattern recognition techniques focus on important features of each reduced database and surface. In this work, process states are restricted to a list of process problems and level of controller effectiveness.

This methodology has two essential requirements. First, for each identifiable process problem, a set of historical data must exist. Because the network learns through example, to identify a problem the network must have seen it before. However, all possible process problems need not be represented from the outset. The system easily incorporates new problems as they arise. Second, each of these databases must be fully representative of a particular problem.

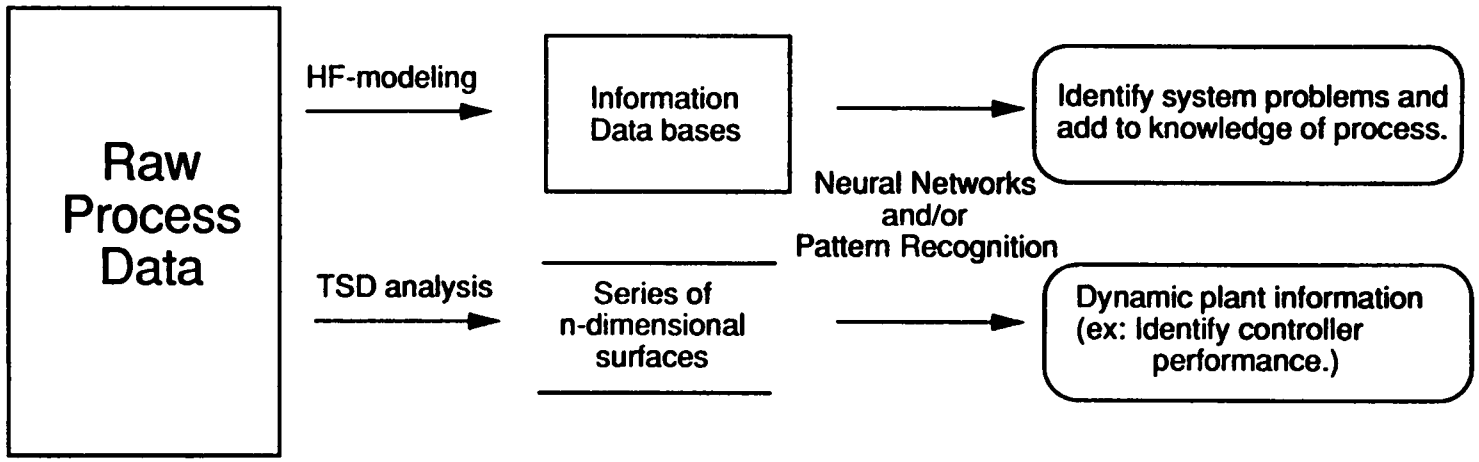


Figure 6.1: Overview of cultivation framework for diagnosing system problems

This means that each database must have sufficient data, covering a wide range of conditions, to convey all characteristics of that problem to the network.

Overall, this method embodies several important characteristics. It identifies all magnitudes of system problems very well, even if the teacher data only represents one particular magnitude; it gives an indication of the magnitude of the problem; it handles multiple problems occurring simultaneously reasonably well; and it defines when a "new" system problem has occurred.

In designing a framework that accomplishes the above characteristics, many technical questions must be addressed. What are the best neural networks or pattern recognition techniques for each task? How should data be presented to these AI tools? How should data be scaled? How much historical data is necessary to give meaningful results? The following sections address these questions and other issues.

6.1 Neural Networks with HF-Modeling

Neural networks interpret reduced databases that result from HF-modeling. Before they can identify system problems successfully, neural networks must be trained by example. Figure 6.2 shows a flow diagram representing the training phase. The process begins with several historical databases, each representing a particular process problem or normal operation. HF-modeling processes each historical database individually into reduced information databases. The neural network samples several records of information from each reduced database

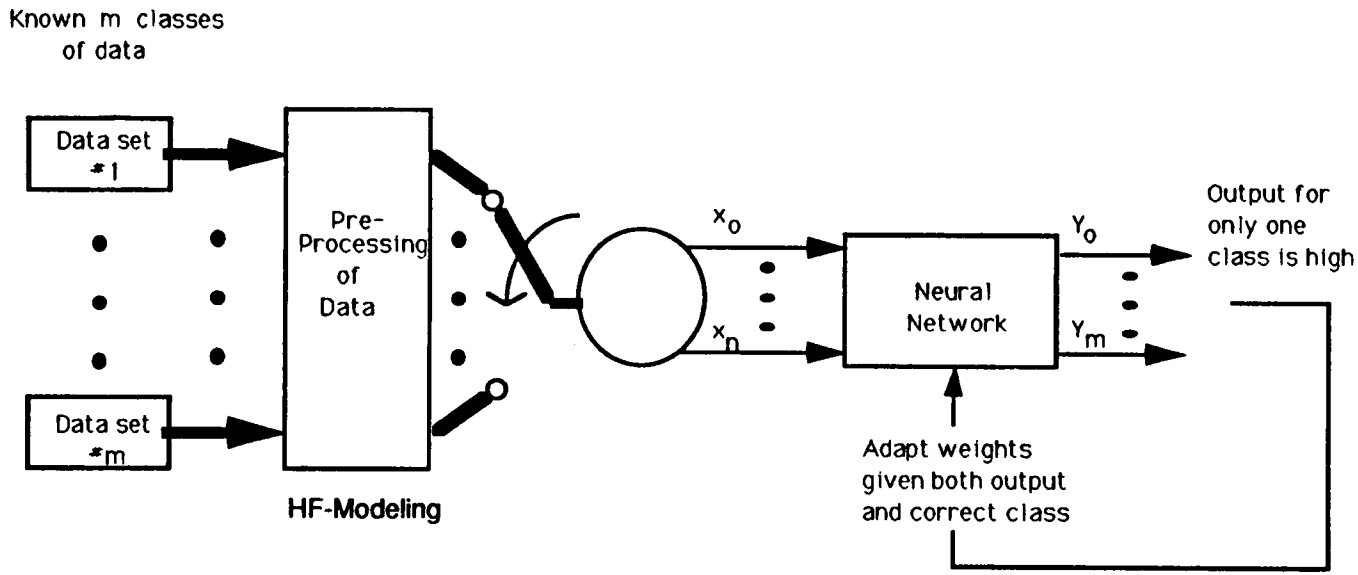


Figure 6.2: Neural network training

as it learns, as illustrated by the switching mechanism. In other words, rather than processing the databases sequentially (one by one), the network alternates among several small segments of each database until it has processed all information. There are two reasons for this technique. First, the network "forgets" important information from the first few databases as it works through the remaining ones. It tends to remember best what it did last. Second, initially giving the network a general idea of each system mode (alternating through them) expedites learning. Because the network has an idea of where each class lies in the pattern space, initial formulation of decision boundaries between classes is improved.

The network examines the data one pattern vector at a time. It classifies that vector and then adjusts its weighting factors depending upon whether or not it has classified correctly. The procedure continues until a specified percentage of data from each information database is classified correctly.

Output from the network simply indicates in which class the input belongs. With the multilayer perceptron, a one-to-one correspondence exists between the known system problems (classes) and the top-layer nodes. These nodes can have a value of 1 or 0, and only one may be high at a given time. The node that is high indicates in which class the input belongs. Note that the network does not indicate problem magnitudes or new system problems.

Once trained, the network classifies an unknown set of data in the same manner as in the training procedure; however, there is no adjustment of weighting factors. As in the training phase, HF-modeling first processes historical

data from this new data set to arrive at a reduced information database. Each record in this reduced database represents statistics and relationships within one particular operating condition. The trained network classifies this data one record at a time into a class of system problem or normal operation. The most meaningful result is the overall percentage of records from the reduced database that the network classifies into each class of system problems.

6.2 Presenting Data to the Network

Any pattern recognition technique, neural networks included, performs best when the classes form distinct, well-separated clusters in the pattern space. Neural networks cannot perform miracles; preprocessing must be done effectively. Therefore, the format in which data is presented to the network is critical to good performance, in both the training and classifying phases. First, the network itself puts constraints on these inputs; each nodal input must be between zero and one (or -1 and 1 for some networks). Beyond that, several preprocessing steps enhance the separation of problem classes in the pattern space, facilitating the job of the network. The following sections discuss several important issues concerning the presentation of cultivated data to the neural network.

6.2.1 Selecting statistics presented to the network

HF-modeling techniques make statistics on process variables and multi-variable relationships more easily accessible. Figure 3.3 exemplifies a database reduced by HF-modeling. Columns 1 to 3 define specific operating conditions; each represents values of important independent variables that are reasonably constant. The remaining columns represent statistics on important variables and relationships. For example, column 4 is the average value of the control variable. Column 5 represents the standard deviation of the control variable within an operating condition. Columns 6,7 and 8,9 consist of pairs of averages and standard deviations of other important dependent variables. The mean value of several relationships or partial derivatives (see Section 3.1) make up columns 13 through 16. Any HF-modeling property calculated within each operating condition can be incorporated into the analysis by simply adding it to this reduced database.

Choosing important statistics to make up the reduced database is of primary importance. Finding the minimum set of statistics that clearly differentiates between the classes of process problems defined would be optimal; however, this can be very difficult. Some important process variables are not measured, making them unavailable for use in the reduced database. Of the variables that are measured, important ones must be selected and unimportant ones eliminated. Because the network does a reasonably good job of filtering out unimportant information, one should be conservative in eliminating statistics. However, if too many meaningless variables are incorporated into the

reduced database, training the network becomes very time consuming.

Several techniques are available for defining statistics important enough to include in the reduced database. The best method is to simply use fractional overlap, separation, and average partial derivatives for identifying critical statistics and relationships.

6.2.2 Incorporating global parameters

The reduced database described so far contains vital information calculated within each operating condition. Many times, however, one needs to incorporate global parameters: statistics representative of the entire database. Because the neural network (or pattern recognition technique) examines each individual operating condition, development of reliable techniques incorporating global statistics is needed.

The method outlined here is a Monte Carlo type method. If the global parameter can be characterized by an average and standard deviation, this method is effective. One such global attribute stems from a relationship plot, depicted in Figure 3.6. This plot demonstrates the relationship between one dependent variable and one independent variable, while holding all other variables constant. The average slope and the variation in the slope characterize this plot. The Monte Carlo method simply assigns values to one column in the reduced database, in a pseudo-random fashion, such that the average and standard deviation of that column match the average and variance in the slopes on the relationship plot. In this manner the network understands both the

average and a reasonable variation. Both are important.

6.2.3 Scaling the reduced database

Scaling the reduced database prevents processing unit saturation and improves the ability to distinguish between classes. For best results, almost all neural networks require the range of inputs at any input node to be from -1 to 1; some require from 0 to 1. In addition to fulfilling that requirement, several scaling methods improve the separation of the pattern classes, thus improving the performance of the chosen neural network.

This section summarizes three scaling methods, discussing the advantages and disadvantages of each. The first two, normalization and physical scaling, are common techniques, but each has its imperfections in this application. The third scaling method, basecase normalized scaling, combines the positive properties of physical and directional scaling with other important advantages for defining process problems.

6.2.3.1 Physical scaling

Physical scaling simply scales each variable according to the zero and full-scale sensor values for each variable. For example, if a temperature sensor ranges from 50 to 150 degrees and inputs to the network must lie between 0 and 1, then a 50-degree temperature scales to zero and a 150-degree temperature scales to 1. Temperatures between these extremes are linearly interpolated.

Because physical scaling puts the relative importance of changes in each

variable on similar magnitudes, this method makes intuitive sense to an engineer. In other words, an increase of 0.1 in each scaled variable is of relatively equal significance, regardless of the process variable being measured. Unlike directional scaling, physical scaling retains information about the problem magnitude.

Unfortunately, because of the restricting amount of data representing each class of system problem, physical scaling alone is not always best. For example, suppose training data for a certain class of system problem represent only a limited range of problem magnitudes (which is very often the case). The neural network would not reliably classify smaller magnitudes of that particular problem; therefore, classification of a problem must be magnitude independent. While specifying the severity of the problem is also important, it should be a secondary analysis.

In addition, in employing physical scaling two problems are often discernible during the training phase, but the network performs poorly in classifying them. The primary reason is that the training data represents only a fraction of the spectrum of possible problem magnitudes. Even though the training sets that represent each problem are separable, the two problems are indistinguishable for other degrees of these problems. The network should be able to identify such a situation during the training phase. This difficulty is addressed later in this section.

6.2.3.2 Normalization

Normalization transforms each vector to a unit vector so that each pattern presented to the network represents a direction relative to the origin. This method eliminates the ambiguities in classifying different magnitudes of problems; therefore, it is magnitude independent. Pattern recognition often employs normalization.

Normalizing data is a step in the right direction for optimizing neural network performance. It allows a network to learn all magnitudes of a system problem from a single magnitude, an appealing concept. However, normalization, as presented, has a few problems. Most of its problems stem from calculating a direction relative to the origin rather than from basecase operation.

6.2.3.3 Basecase normalized scaling

Basecase normalized scaling combines the positive attributes of physical scaling and normalization in a manner well suited for neural network classification. Because it scales data so that identical changes in different scaled variables have the same magnitude of change on the process, physical scaling is important. Normalizing the information allows learning of all magnitudes of each problem from a small range of problem magnitudes.

Before discussing the basecase normalized scaling procedure, it is important to understand the ideas behind it. A two-dimensional example illustrates many of these ideas. In Figure 6.3, data from normal or basecase operation lies

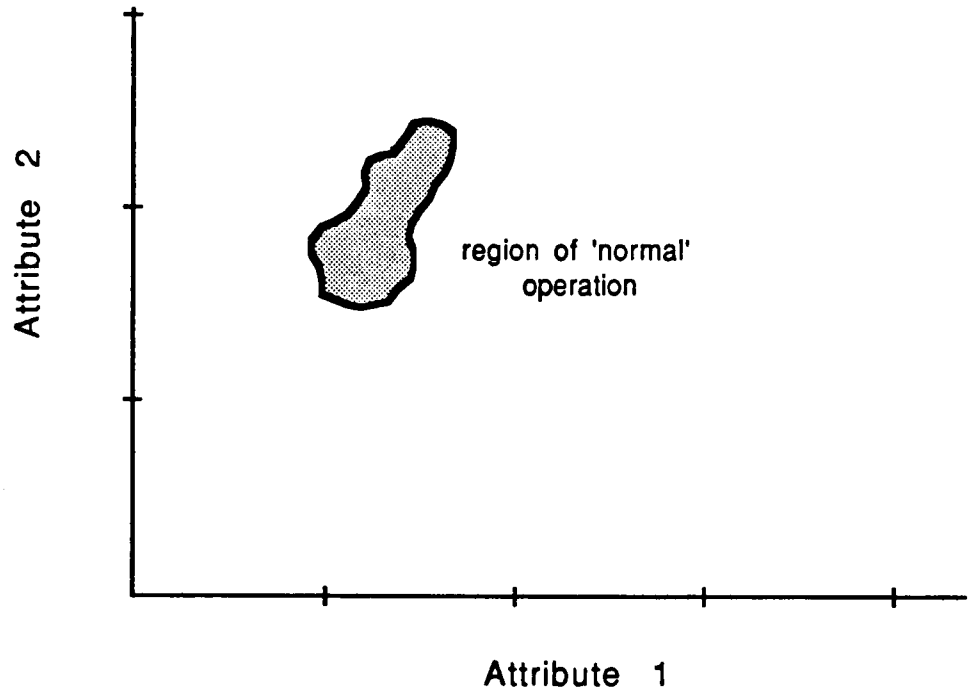


Figure 6.3: Cluster of basecase data

in the enclosed region. On this plot, elements from a two-dimensional pattern vector define the X and Y axis. Assume now that a particular process problem of very small magnitude occurs; for example, a heat transfer coefficient begins to degrade. If meaningful variables make up the two-dimensional plane, the "cluster" shifts slightly away from basecase, as shown in Figure 6.4. As the heat transfer coefficient continues to degrade over time, this cluster moves further from basecase, tracing some characteristic path in the two-dimensional plane. If the process is again operating normally and a different problem occurs, the cluster will move over a different path away from basecase.

A potential problem with normalization is exemplified in Figure 6.5. This figure shows two-dimensional pattern vectors that form three clusters. One cluster contains vectors from normal operation, another contains vectors from process problem one, a third cluster from process problem two. Obviously these problems are differentiable in original form, but when presented in normalized form, they are not, because they lie in the same direction relative to the origin.

These examples demonstrates that the primary concern in scaling should be the path that the data takes in moving away from basecase operation. Both the shape and the direction of this path are important. Ideally, each system problem moves in a unique path and the paths do not cross.

Basecase normalized scaling scales data to be well suited for examining the path each process problem takes with increasing magnitude. But, most of the time, training data represent only a small range in the magnitude of each

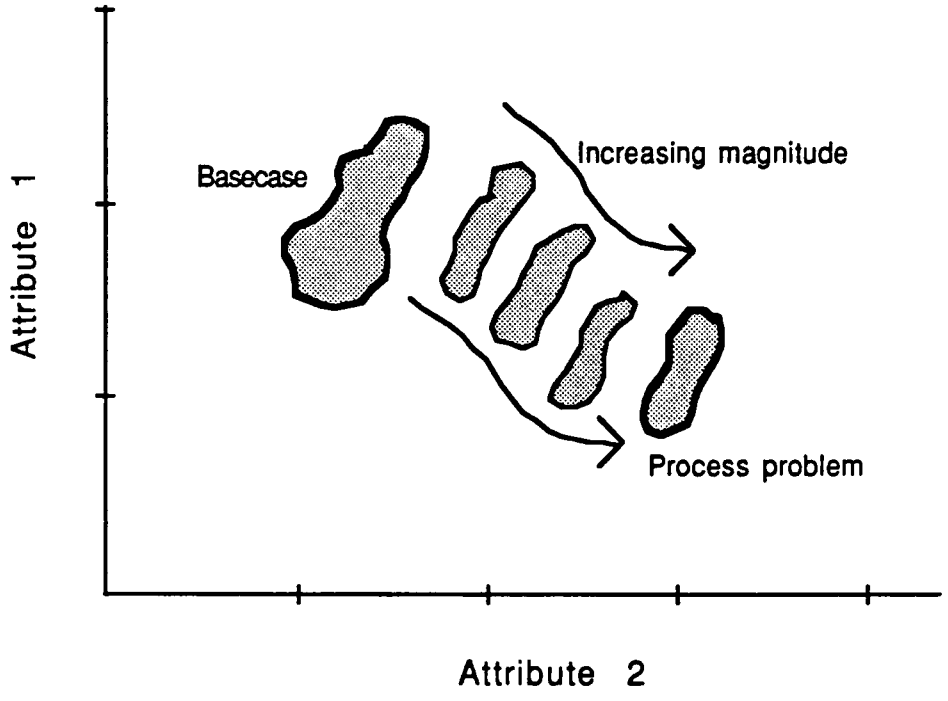


Figure 6.4: Shifting of cluster due to small process problem

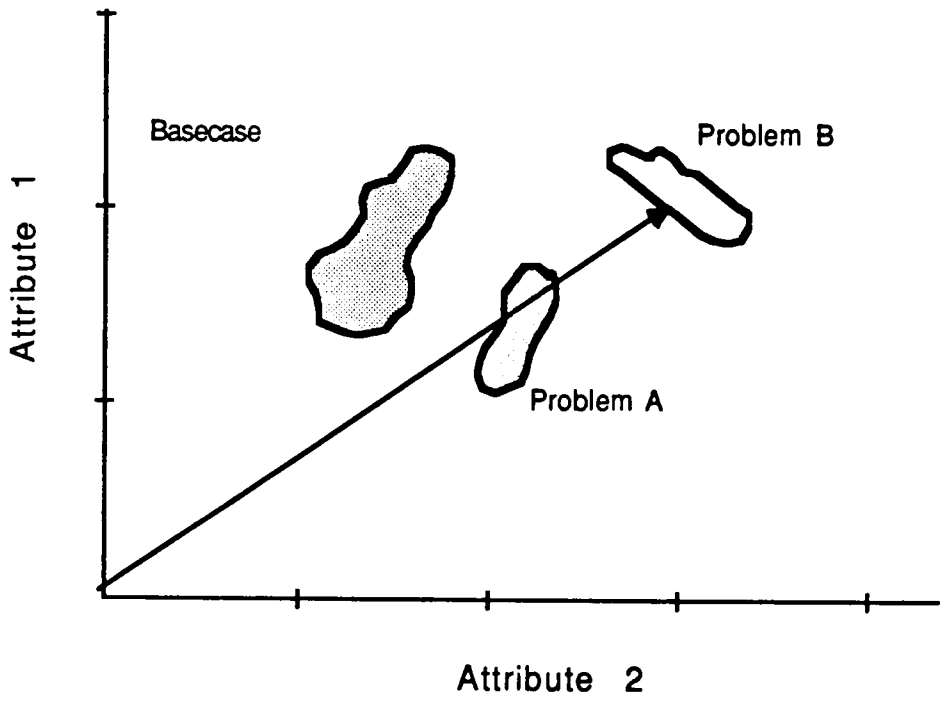


Figure 6.5: Clusters that exemplify a problem with normalized scaling

process problem. Therefore, it is not practical to examine the shape of each path. Assuming that these paths are reasonably linear, only the direction of each path relative to the basecase cluster is important. The scaling procedure subtracts each vector from a representative basecase pattern vector. This representative could be the centroid of the basecase cluster, for example. The vector is then normalized to represent the direction of the original unscaled vector relative to basecase operation. Of course, some system problems are of only a single magnitude, for example, a stuck steam trap. This type of problem is a simplified case of the type that has been addressed. Any technique that handles the more general case will also handle this simplified case.

Figure 6.6 outlines the entire procedure. Step one physically scales each vector. Step two subtracts each vector from a representative basecase vector (Section 7.2 discusses a representative vector for cultivated data). From that point, each vector is normalized, yielding a set of basecase normal scaled vectors and a set of vector norms. The scaled vectors help identify the system problem; the vector norms indicate the problem magnitude.

The following two-dimensional example, taken from a simulated continuous stirred-tank reactor (CSTR), clarifies the outlined scaling process. (see Section 7.2). Figure 6.7 plots three sets of physically scaled data, for basecase operation, from a small change in heat transfer coefficient (5%), and from a large change in heat transfer coefficient (10%). Figure 6.8 shows the result after each point in the second two data sets is subtracted from its corresponding "normal" (basecase) operating point (see section for full a description). Here,

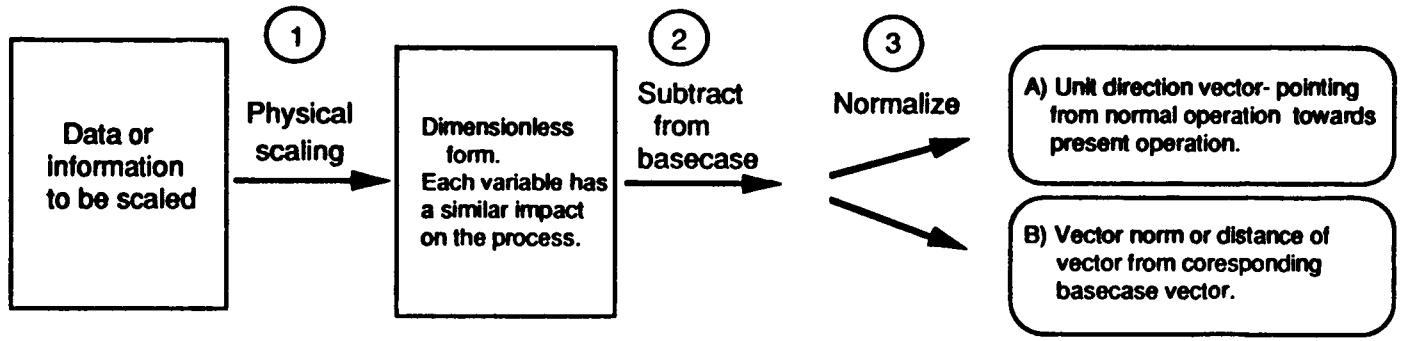


Figure 6.6: Basecase normalized scaling

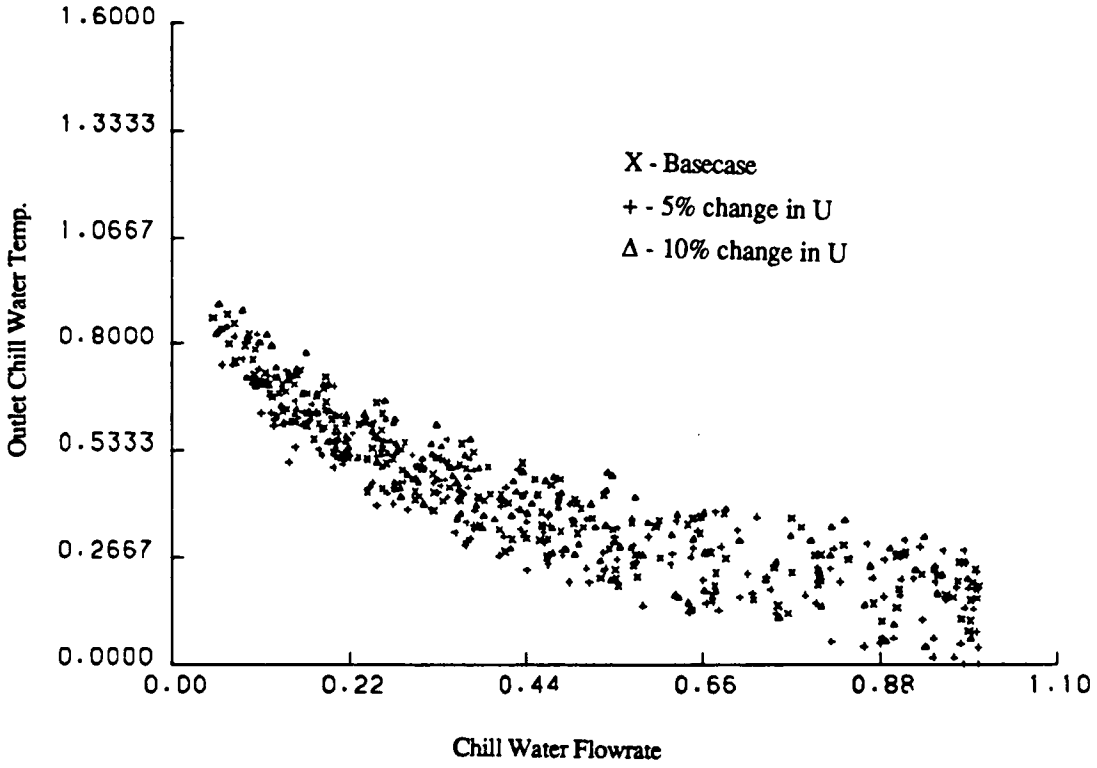


Figure 6.7: Physically scaled data

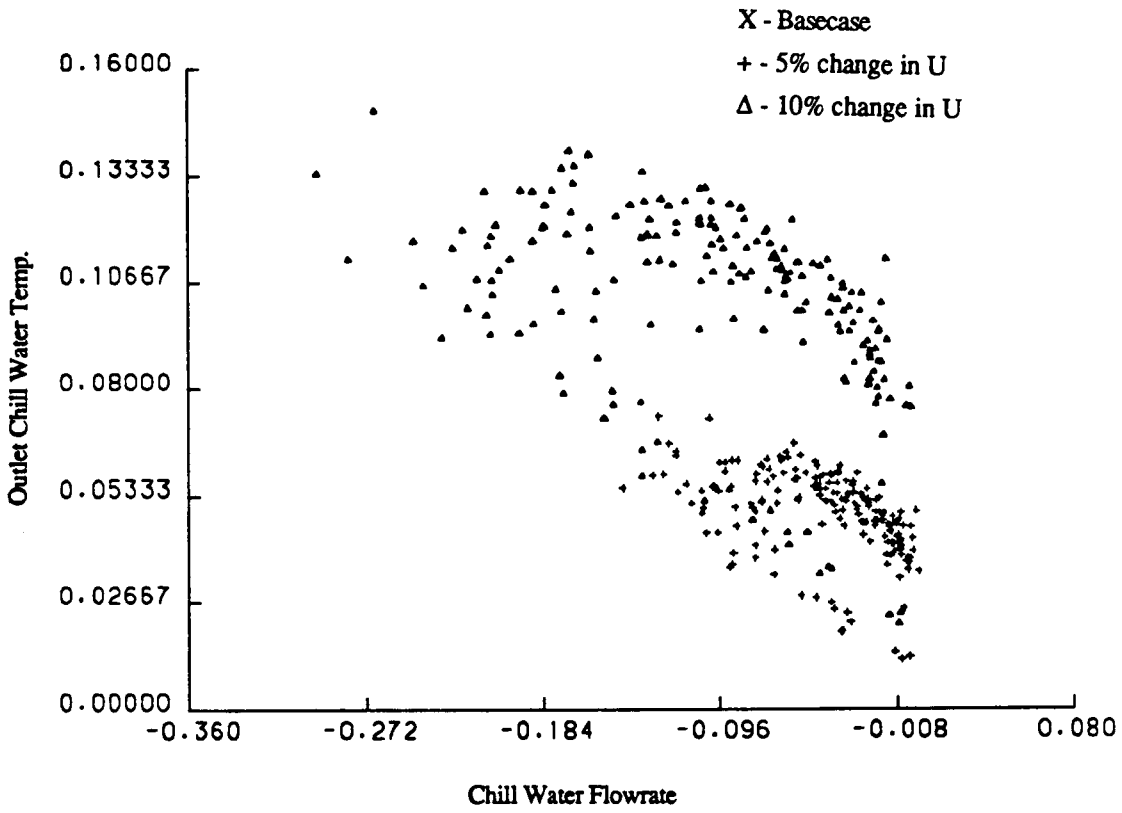


Figure 6.8: Data after subtracting from basecase

the basecase cluster has been compressed effectively to the origin, and almost all the data fit into a pie-shaped envelope that can be described by a range in direction (or angle). Figure 6.9 shows the result after each vector is normalized. Each point lies on the unit circle. A particular process problem can be clearly defined by its direction away from the origin, regardless of problem magnitude. Problem magnitude can be approximated by the norm of each vector. The scaling process for higher dimensional problems is analogous to the two-dimensional case.

The scaling method presented here incorporates several important characteristics. First, because data is physically scaled, different measurements (for example, temperature and concentration) are compared on a similar basis. Second, the resulting vector directions are independent of problem magnitude. Third, a vector norm indicates the magnitude of the problem. Finally, indistinguishable problems (given the information available) can be identified with a good degree of certainty.

6.3 Problem Recognition Framework

A refined framework that reliably identifies system problems has been developed. This problem recognition framework comprises HF-modeling and neural networks as the principle tool, along with scaling techniques and and distance-based classifiers. The importance of these minor parts of the framework to achieving good results should not be underestimated.

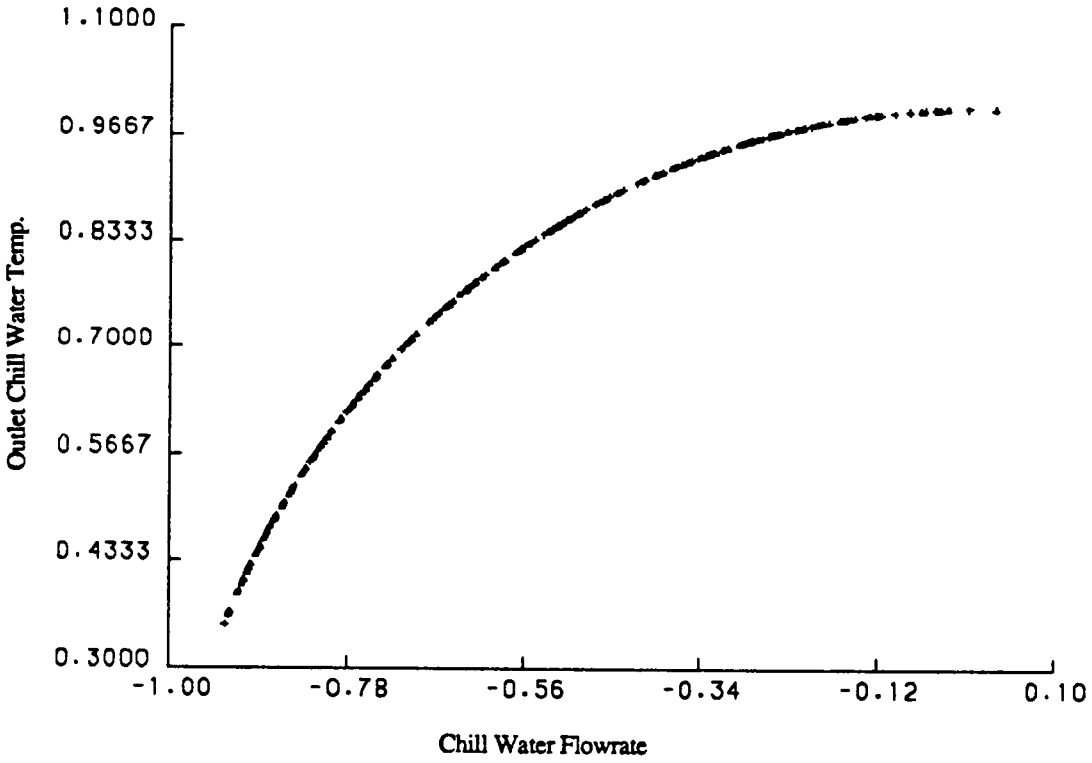


Figure 6.9: Data after basecase normalized scaling

This recognition framework classifies system problems accurately and defines problem magnitude reliably while accounting for new system problems, multiple system problems, and indistinguishable system problems.

A detailed explanation of this framework is presented in the CSTR example in Chapter 7 where an explanation will be better understood. The progression of examples in the next chapter will hopefully serve to better understand the attributes of each part of the final framework.

CHAPTER 7

Example Applications

The examples in this chapter demonstrate cultivation tools applied towards detecting and diagnosing system problems. Two simulated systems exemplify these methods, a simple shell-and-tube heat exchanger and a continuous-stirred-tank- reactor. Some of the work presented in the heat exchanger example is already detailed in[17] but is repeated here for completeness. The heat exchanger example demonstrates both HF-modeling and neural networks applied individually, and the framework combining HF-modeling and neural networks. The CSTR example applies the combined HF-modeling-neural network package to a somewhat more complex system and demonstrates the TSD analysis combined with neural networks.

7.1 Heat Exchanger

A simple shell-and-tube heat exchanger demonstrates some of the cultivation tools described in the previous chapters. This particular system was the focus of a preliminary cultivation study[17]; most of the results from that study are repeated here along with new results. This example uses cultivation tools for detecting abnormal system operation and diagnosing system problems. It consists of three parts. First, HF-modeling detects and diagnoses specific sys-

tem problems. Then neural networks are applied for the same purpose. The effectiveness of the two methods are evaluated and compared. Finally the framework coupling HF-modeling with neural networks is demonstrated.

7.1.1 The system studied

Figure 7.1 shows the liquid-liquid, shell-and-tube heat exchanger examined in this study. In this system, a hot process stream is cooled to a specified temperature by manipulating flow of a chill water stream. Heat exchanger specifications and operational parameters are summarized in Figure 7.2.

This study aims to detect and diagnose several system problems summarized below.

- Reduced inner heat transfer coefficient (fouling)
- Increased outer heat transfer coefficient (insulation problem)
- Change in heat capacity of the hot process stream
- Change in heat capacity of the chill water
- Stuck sensors
- Stuck valves
- Poorly tuned controller

Simulation of this system consists of a steady-state energy balance combined with a first order lag plus dead-time element. Figure 7.3 illustrates the

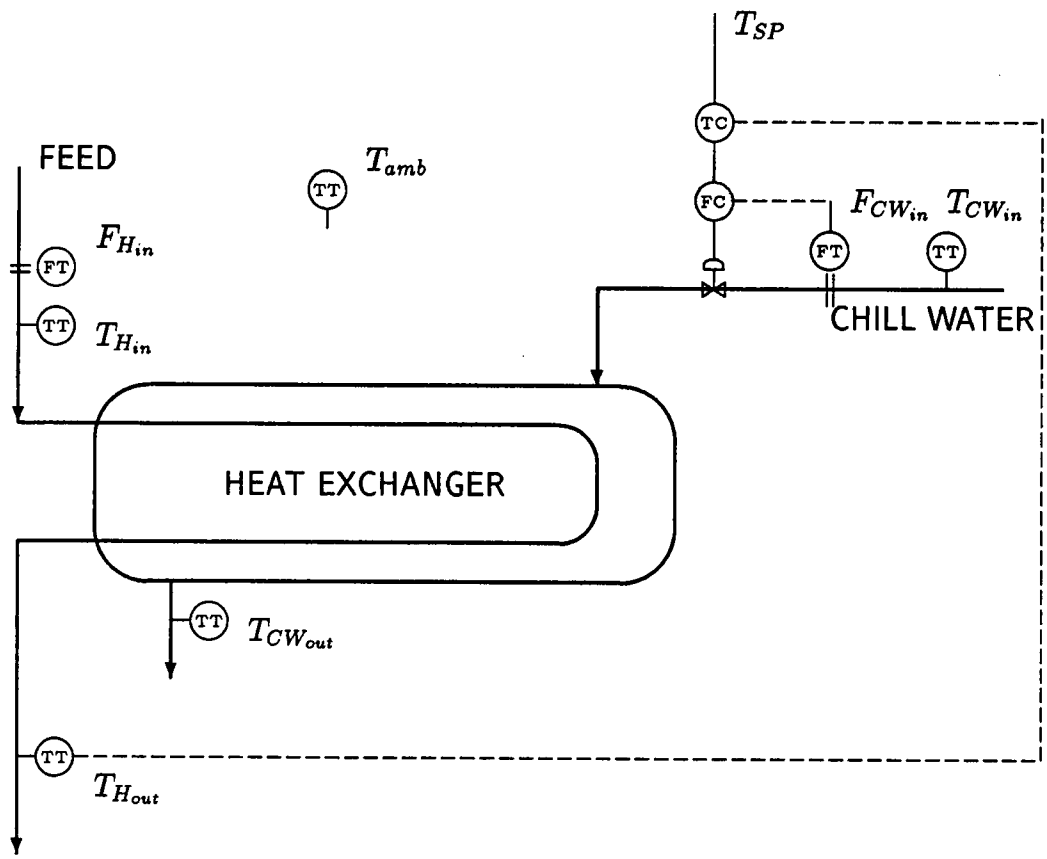


Figure 7.1: Shell and tube heat exchanger

Heat Transfer Specs

Inner area (A_{inner}): $200 ft^2$

Outer area (A_{outer}): $45 ft^2$

Inner heat transfer coef (U_{inner}):$120 \frac{BTU}{hr ft^2 \, ^\circ F}$

Outer heat transfer coef (U_{outer}): $35 \frac{BTU}{hr ft^2 \, ^\circ F}$

Tube Side: Hot Oil

heat capacity ($C_{p_{hot}}$): $0.5 \frac{BTU}{lb_m \, ^\circ F}$

ave inlet temp: $250 \, ^\circ F$

Shell Side: Chill Water

heat capacity ($C_{p_{cw}}$): $1.0 \frac{BTU}{lb_m \, ^\circ F}$

ave inlet temp: $94 \, ^\circ F$

PI Controller Settings

$K_c = 400$

$\tau_I = .01$

$Setpoint = 104.0$

Figure 7.2: Basecase heat exchanger specifications

system simulation diagram, and Figure 7.4 presents the entire feedback loop.

A simple PI (proportional-integral) control algorithm manipulates the chill water flow. Its digital algorithm is given by

$$V_n = V_{n-1} + K_c[(e_n - e_{n-1} + \frac{T}{\tau_I} e_n]$$

where

V_n =Controller output at n^{th} sampling instant

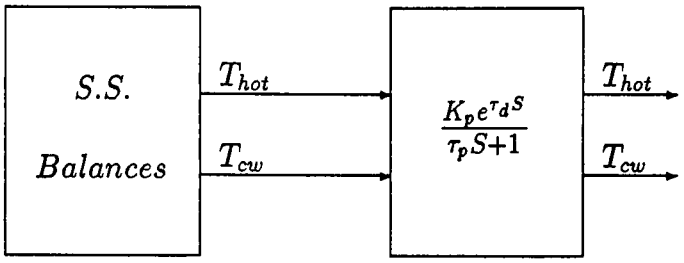
e_n =error at n^{th} sampling instant

T =Sampling period

τ_I =Integral time

The heat exchanger system is operated under conditions typical in a real environment. It experiences disturbance in several system inputs: T_{amb}, $T_{cw_{in}}$, $T_{hot_{in}}$, and F_{hot}. Typical disturbance profiles over both 12- and 72-hour time frames are shown in Figures 7.5 and 7.6. These disturbances consist of white noise superimposed on integrated, autoregressive, moving average functions. Figure 7.7 exemplifies typical system responses in terms of the manipulated variable (chill water flow) and the controlled variable (outlet temperature).

Collected simulation data comprise temperature and flow rate of the inlet hot process stream, temperature and flow rate of the inlet chill water stream, outlet temperatures of both streams, and ambient temperature. Figure 7.8 exemplifies a typical historical database.



S.S. Energy Balance

$$Q_{in} = UA_{inner}\Delta T_{lm_{inner}}$$

$$Q_{in} = M_{cw}C_{p_{cw}}(T_{cw_{out}} - T_{cw_{in}}) + Q_{outer}$$

$$Q_{in} = M_{hot}C_{p_{hot}}(T_{hot_{in}} - T_{hot_{out}})$$

$$Q_{outer} = UA_{outer}\Delta T_{lm_{outer}}$$

Figure 7.3: Flow diagram of simulation

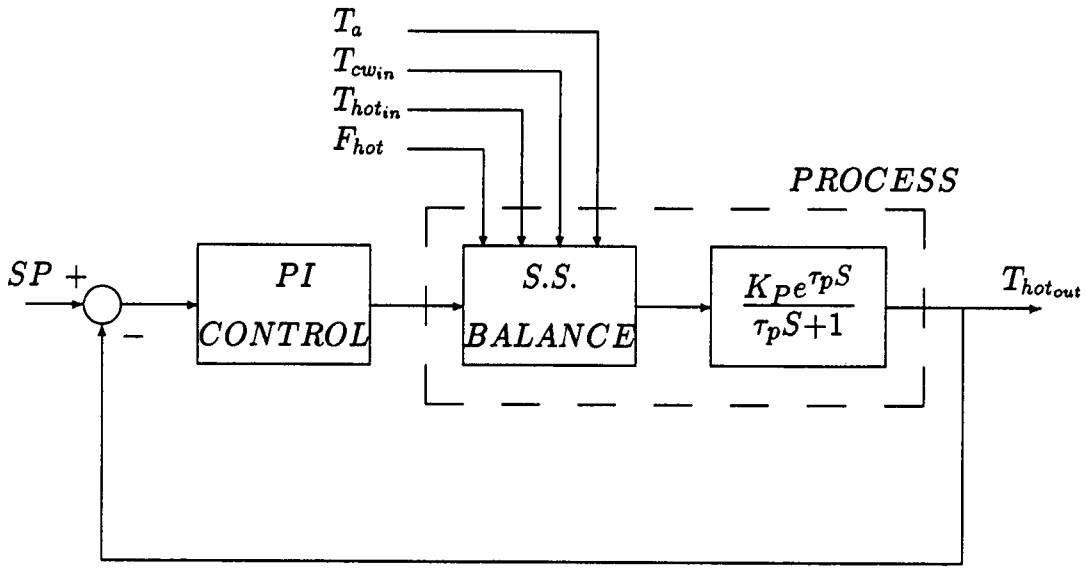


Figure 7.4: Flow diagram of feedback system

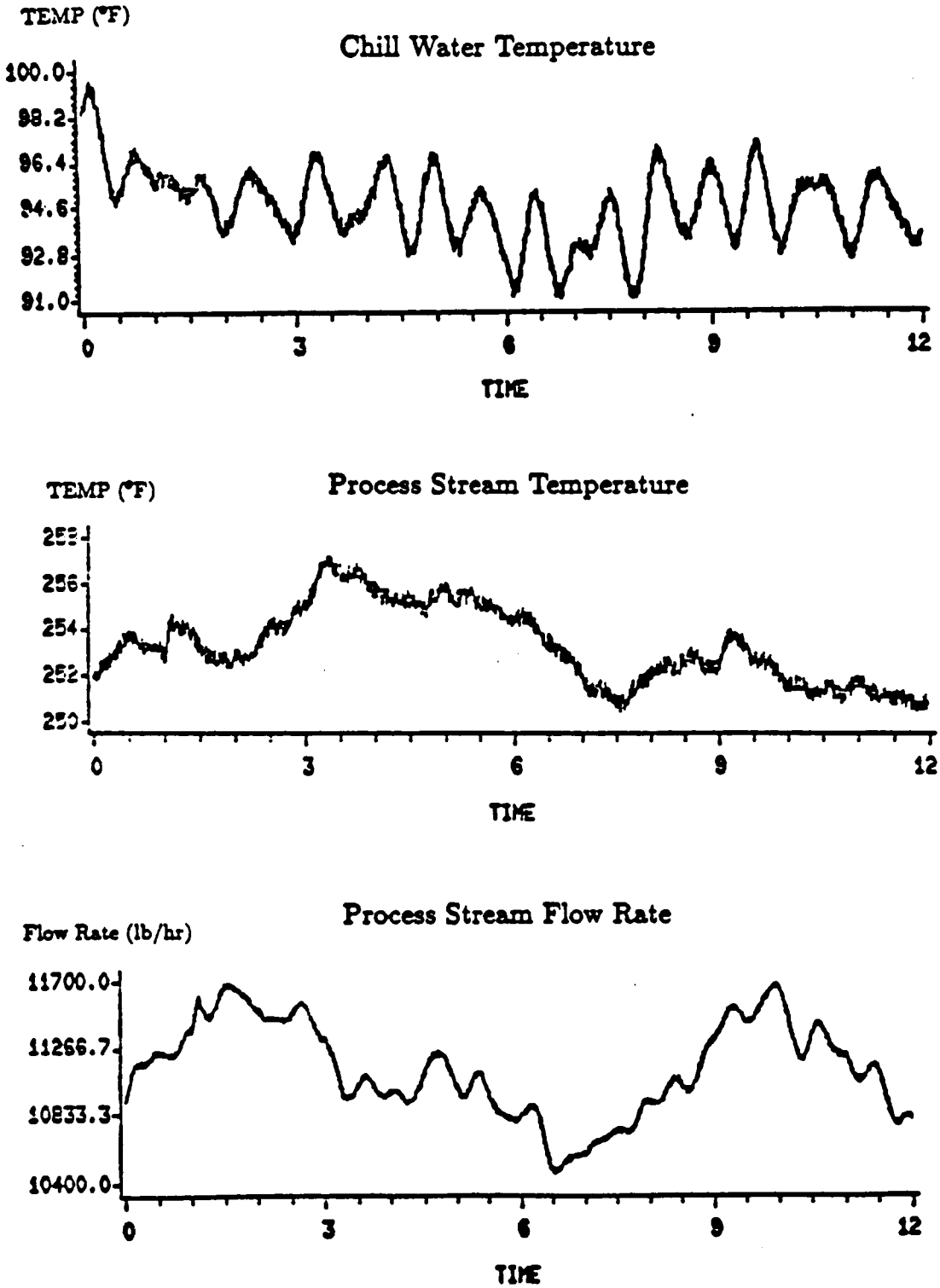


Figure 7.5: Operating conditions (12 hour span)

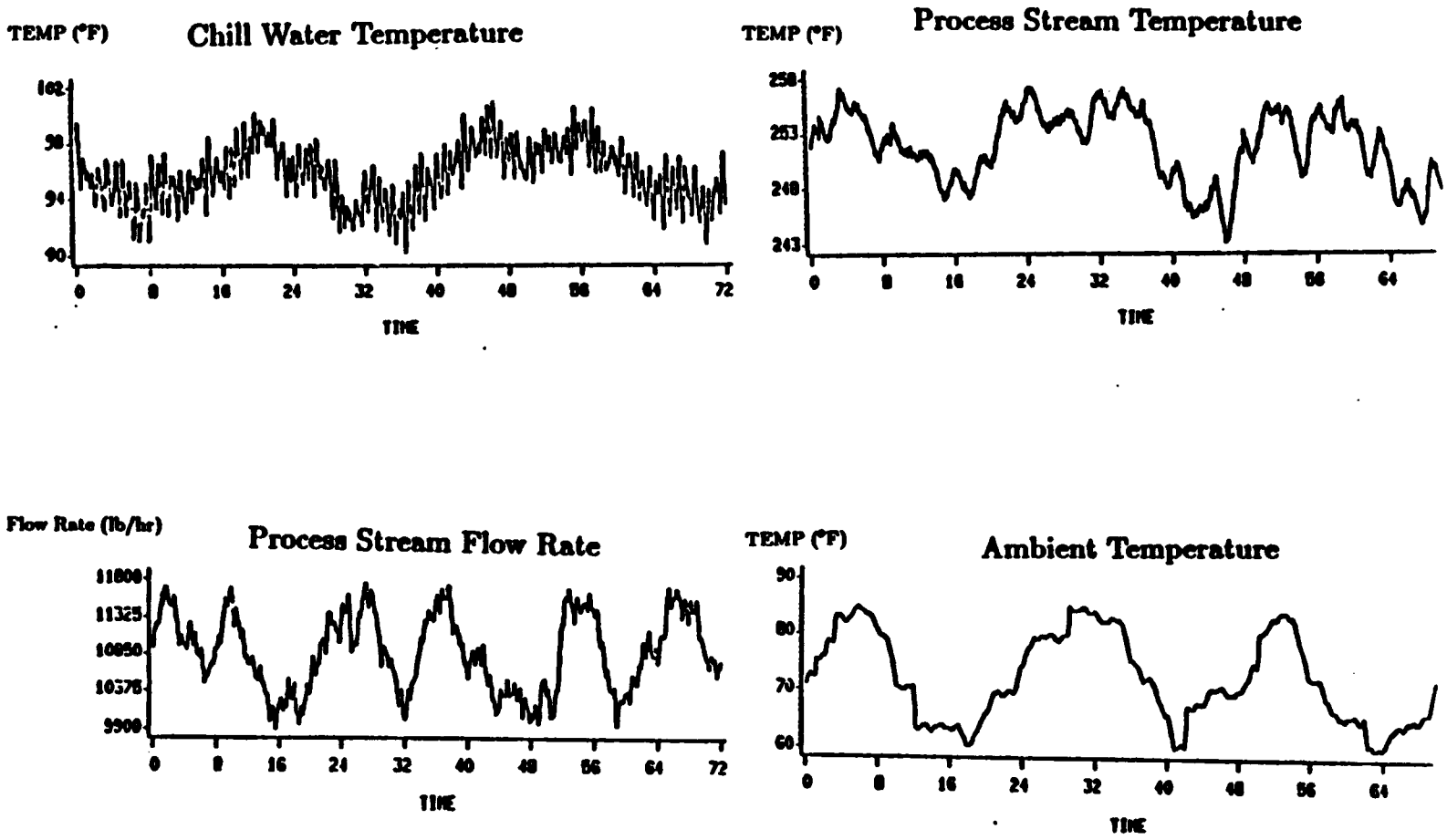
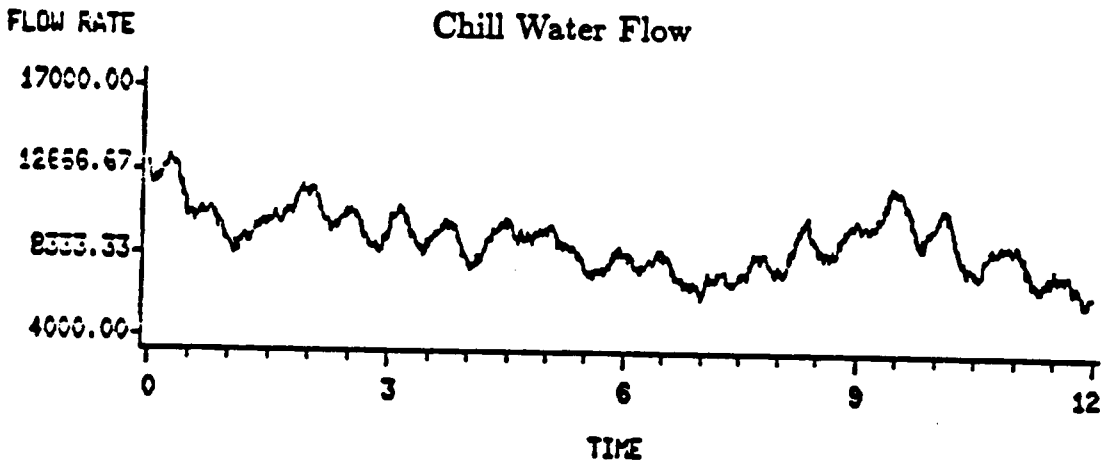


Figure 7.6: Operating conditions (72 hour span)



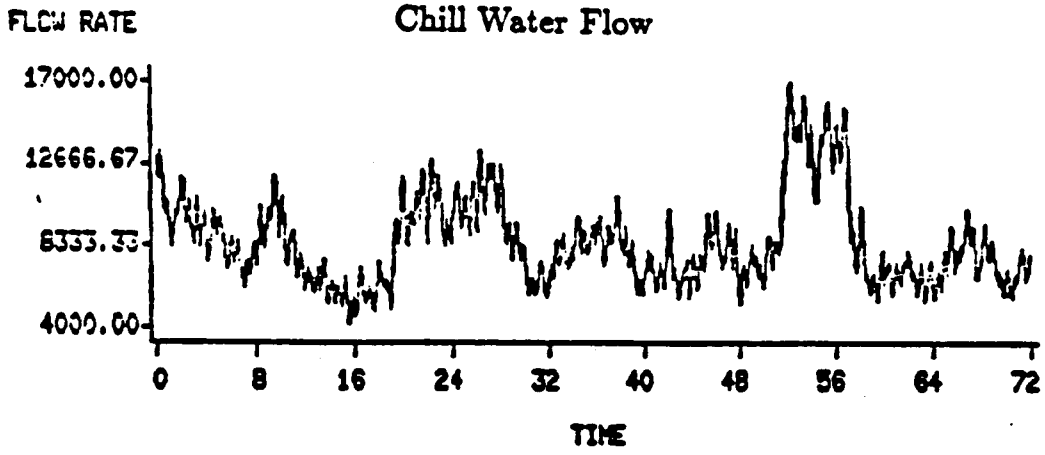


Figure 7.7: Manipulated variable profiles

DATA BASE: SIMULATION OUTPUT

TIME (HR)	INLET TEMP (DEG F)	OUT TEMP (DEG F)	FEED FR (LB/HR)	CW TEMP (DEG F)	CW FLOW RATE (LB/HR)	AMB TEMP (DEG F)
0.000	252.39	102.61	10954.11	96.55	13018.59	72.72
0.020	252.10	101.54	10958.12	96.27	11871.03	72.64
0.040	252.42	101.65	11024.05	96.57	11259.79	72.76
0.060	252.31	103.54	11031.05	96.41	11401.83	72.75
0.070	253.03	103.63	11107.69	97.07	11410.92	73.08
0.090	252.42	102.66	11059.51	96.40	10994.22	72.83
0.110	253.03	103.74	11144.08	96.93	11062.28	73.06
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
90.000	270.85	104.04	12360.09	92.95	1380.50	80.90

*NUMBER OF DATA POINTS: 10,800

Figure 7.8: Output from simulation

7.1.2 HF-modeling analysis of the Heat Exchanger

7.1.2.1 HF-model

Defining a good hypothesis model is key to obtaining meaningful results from HF-modeling. This definition requires only a basic understanding of the process to identify the important system variables. For this example, the final hypothesis model is given by

$$Y_{dep} = f(F_{hot} * T_{hot_{in}}, T_{cw}, T_{amb})$$

This model expresses dependent system variables to be a function of three important independent system variables. Those independent variables are (1) the product of flow rate and inlet hot process stream temperature (an energy variable), (2) inlet chill water temperature, and (3) ambient temperature. This particular model resulted from refinement of one initially proposed, as described later in this section.

7.1.2.2 Defining resolution

As Section 3.2 explains, a resolution must be specified for each system variable of the hypothesis model. Basically, resolution for each variable should be as large as possible so that a statistically meaningful amount of data falls into each operating condition but not so large that each variable is not reasonably constant within each increment. Such a balance requires a basic

knowledge of each variable's effect on the system. For example, a five degree change in ambient temperature may have little effect on the process, while an equal change in chill water temperature may be quite significant. The resolution on ambient temperature might be nearly 20 degrees, while that on chill water temperature may be 1 degree.

For the three variable comprising the hypothesis model for the heat exchanger, the resolution used in these studies is given below:

Process Variable	Zero	Span	Divisions
$F_{hot} * T_{hot_{in}}$	$2.4x10^6$	$1.52x10^6$	25
T_{cw}	85	20	20
T_{amb}	30	70	12

7.1.2.3 Sorting into the reduced database

Section 3.2 outlines the procedure for sorting historical data into the reduced information database. In the present example, historical data is sorted into categories defined by three variables that define regions of constant operation: $F_{hot} * T_{hot_{in}}$, T_{amb}, and T_{cw}.

All records in the historical database are mapped into their corresponding operating condition. "Mapped" data is next sorted according to operating condition; then important statistics and relationships within each condition are calculated. The reduced information database results (Figure 7.9). Figure 7.10

illustrates this sorting.

Each row of the reduced database in this example comprises the three variables that define the hypothesis model ($F_{hot} * T_{hot_{in}}$, T_{amb}, T_{cw}), average and standard deviation of chill water flow (F_{cw}), average inlet and outletprocess stream temperature ($T_{hot_{in}}$, $T_{hot_{out}}$), average outlet chill water temperature ($T_{cw_{out}}$), average product of chill water flow and outlet chill water temperature ($F_{cw} * T_{cw_{out}}$), and three global partial derivatives ($\frac{\partial F_{cw}}{\partial F_{hot} * T_{hot_{in}}}$, $\frac{\partial F_{cw}}{\partial T_{amb}}$, $\frac{\partial F_{cw}}{\partial T_{cw}}$). Section 6.2 outlines the method in which global parameters are incorporated into the reduced database.

From the reduced information database, overall statistical plots and relationship plots follow directly. Figures 7.11 and 7.12 exemplify typical overall statistical plots resulting from the heat exchanger. Figure 7.11 plots information from a simulation in which the inner heat transfer coefficient was reduced by 10% along with basecase operation. The resulting fractional overlap and separation are given. Figure 7.12 depicts a similar plot for a change in chill water heat capacity. A typical relationship plot is illustrated in Figure 7.13. Reference [17] completely documents overall statistical plots and relationship plots for each system problem studied in the heat exchanger.

7.1.2.4 Refining the HF-model

Of primary importance in HF-modeling is the refinement of hypothesis models. Hypothesis models initially proposed monitor mass, energy, or momentum system balances. While these models capture most of the impor-

Operating condition address											
Tamb	Tcw-inlet	Thot-inlet*Fhot	Freq	Fcw (lb/hr)	Thot-outlet (deg F)	Thot-inlet (deg F)	Tcw-outlet (deg F)	Fcw*Tcw-out (lb deg F/hr)	$\frac{\partial F_{cw}}{\partial E}$	$\frac{\partial F_{cw}}{\partial T_{cw_{in}}}$	$\frac{\partial F_{cw}}{\partial T_{amb}}$
3	7	14	35	45.5 ±3.1	101.5 ±1.3	252.8 ±5.7	95.5 ±2.5	4345.3 ±20.5	6.1	11.3	12.8
3	7	15	41	50.6 ±4.0	102.0 ±1.0	254.5 ±8.9	96.2 ±3.0	4867.7 ±26.9	6.5	12.2	13.3
3	7	16	31	56.2 ±3.5	101.9 ±1.1	253.2 ±7.7	97.9 ±3.1	5501.8 ±27.3	5.9	12.1	11.5
3	7	17	39	65.7 ±5.8	101.8 ±1.3	256.3 ±6.7	99.5 ±4.9	6537.2 ±40.2	6.3	11.4	12.9
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
9	15	18	25	165.1 ±4.9	102.5 ±2.0	255.8 ±6.2	94.4 ±5.2	15601.0 ±38.2	5.5	11.0	12.7

107 records

Figure 7.9: Example of reduced information database from heat exchanger

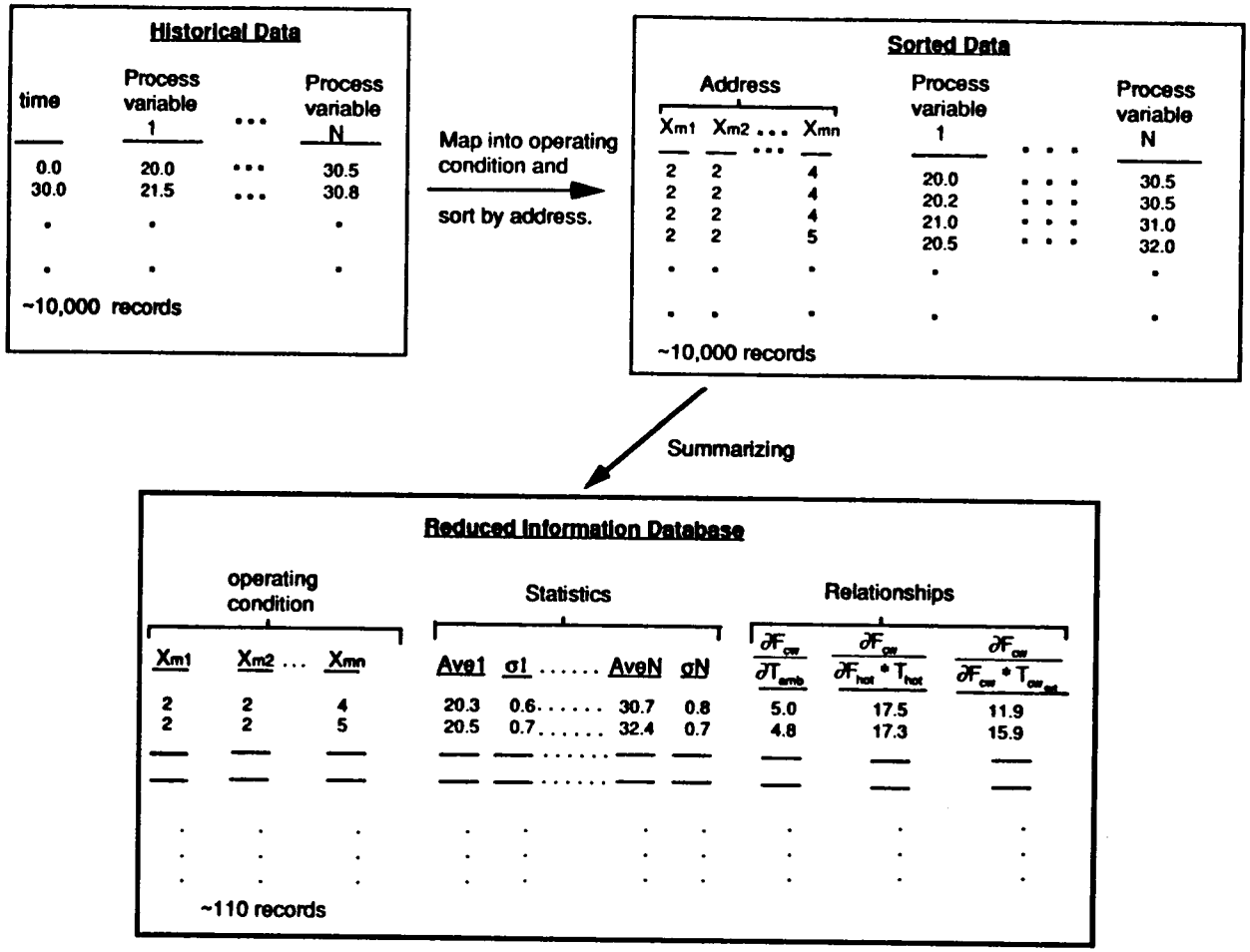


Figure 7.10: Sorting historical data into the reduced data base.

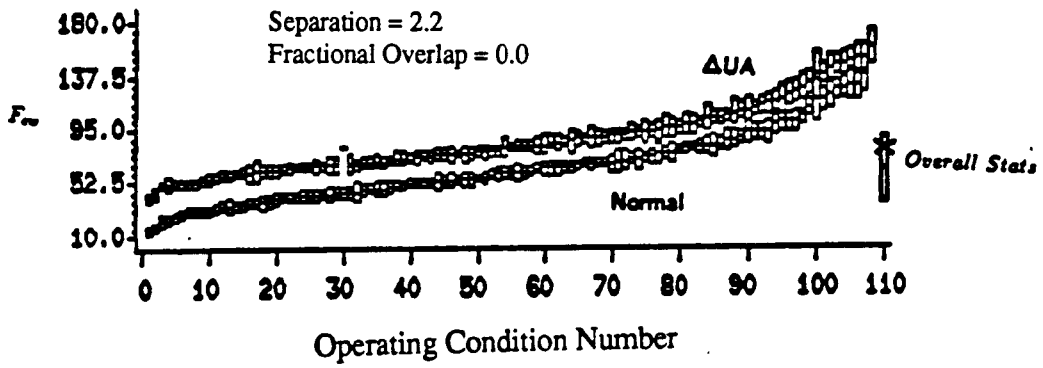


Figure 7.11: Overall statistical plot from heat exchanger data

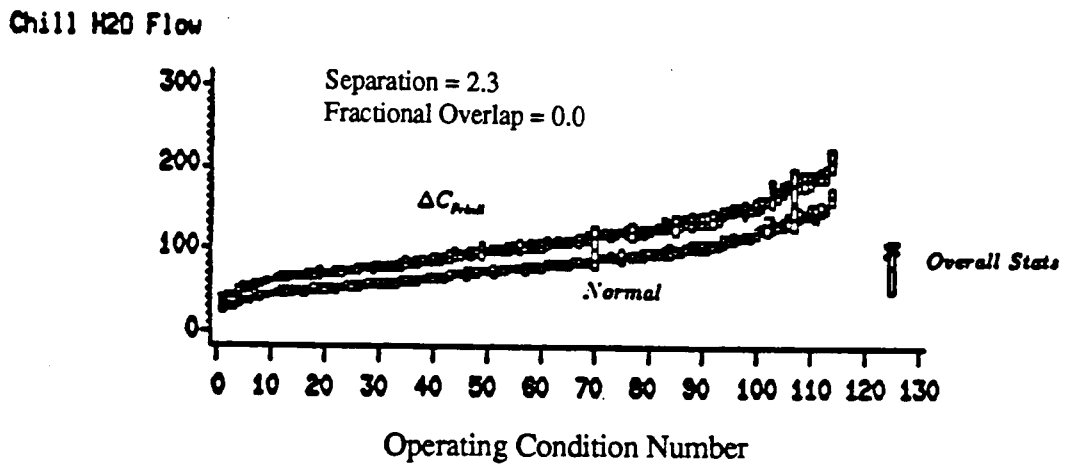


Figure 7.12: Overall statisical plot from heat exchanger data

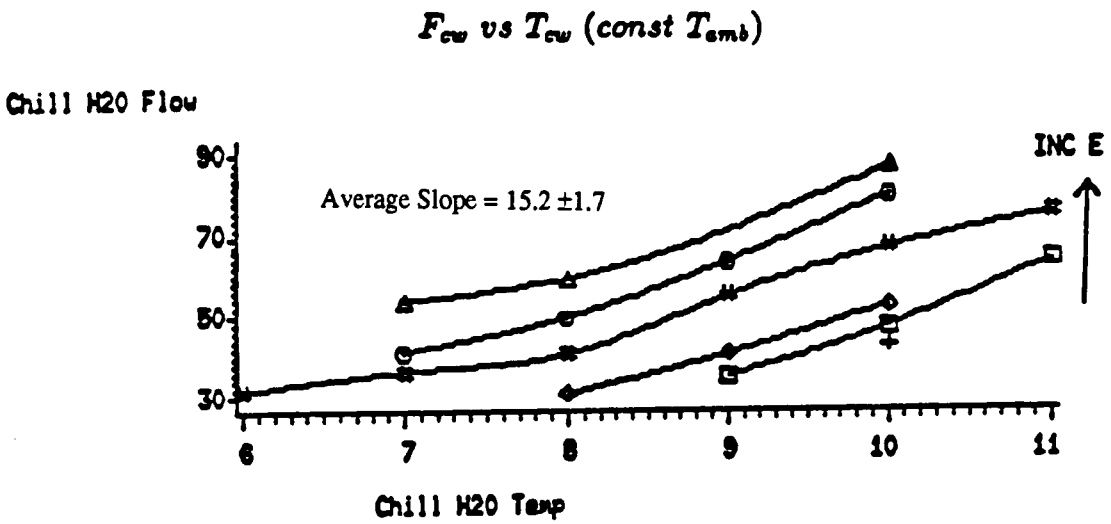


Figure 7.13: Example relationship plot form heat exchanger

tant system variables, fine-tuning models may be necessary. In this example, these feedback techniques are based on the resulting overall statistical plots and relationship plots.

One aspect of model refinement explores the relative importance of each model variable. Some variables in the model influence the process much more than others. Variables of minor significance are excluded from the hypothesis model. One excellent tool for understanding each variable's relative influence on the system is a viewing space cross-section. Like a relationship plot, these cross-sections reveal one independent variable's influence one dependent variable, while all other independent variables are held constant. Unlike the relationship plot that shows the best linear fit of the data, a viewing space cross-section simply graphs all data points. From these cross-sections, one can determine an independent variable's relative influence on the system.

Such an analysis was integral to the heat exchanger example. For example, it was unclear whether ambient temperature was a major influence on the system, but it was included as part of the initial model. Figure 7.14 exemplifies a typical cross-section that shows the effect of ambient temperature on chill water flow. This analysis demonstrated that ambient temperature has a significant influence on the system relative to other system variables and therefore a necessary component of the hypothesis model.

Another important component of model refinement is the uncovering of all important system variables not included in the model. The overall statistical plot plays a key role in this analysis. It can not specifically define important

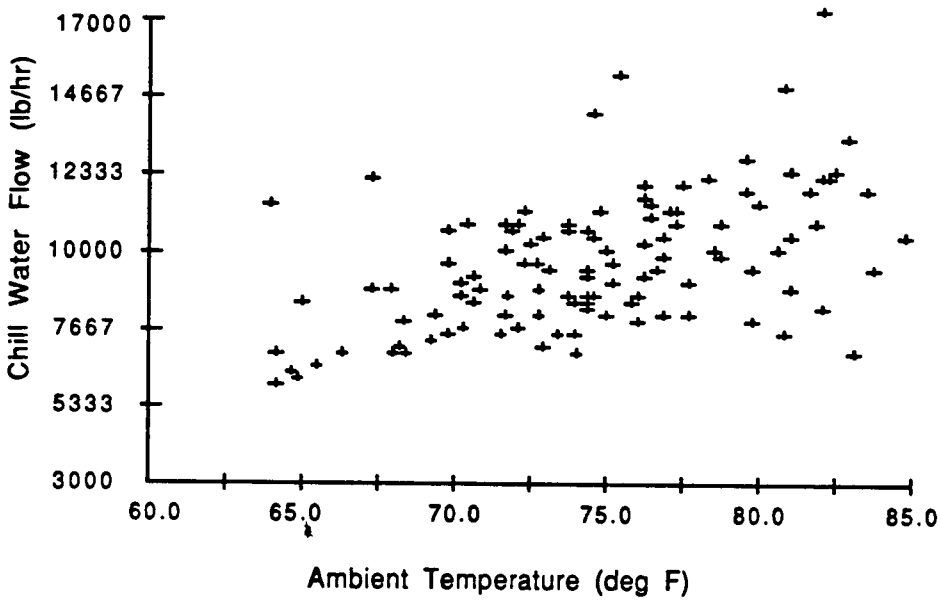


Figure 7.14: Viewing space cross-section showing effect of ambient temperature on the system

system variables, but it does indicate when the model is not fully developed. The primary focus of this analysis is to examine the relative uncertainty within each operating condition. Any uncertainty larger than sensor noise or normal system noise probably indicates the presence of an important variable not defined in the model. Variation due to unmeasured system variables must also be explored.

The heat exchanger example demonstrates this analysis. The initial hypothesis model was defined as $F_{cw} = f(T_{hot_{in}}, T_{amb}, T_{cw_{out}})$. For this model, each operating condition of the overall statistical plots clearly contained greater variation than expected from system or measurement noise. Through trial and error, uncovering the necessary component, F_{hot}, from available process measurements was not difficult. Because of the large number of system variables and the possibility of unmeasured system variables, this method may be difficult in more complex system.

7.1.2.5 Reporters

Reporters are parameters designed to capture the primary important attributes from statistical plots and relationship plots. From each plot, one or two reporter values are calculated. In this example, fractional overlap and separation (see Section 3.2) are calculated from overall statistical plots, and the average partial derivative is calculated from relationship plots.

For each system problem studied on the heat exchanger, Figures 7.15 and 7.16 summarize the reporter values of all statistical and relationship plots.

	F_{cw}		$T_{cw_{out}}$		$F_{cw} * T_{cw_{out}}$	
	Frac Overlap	*Separation*	*Frac Overlap*	*Separation*	*Frac Overlap*	*Separation*
$\Delta C_{p_{chill}}$	0.0	2.3	1.0	0.01	0.0	2.7
$\Delta C_{p_{hot}}$	0.0	3.8	0.0	−1.6	0.43	1.8
ΔUA_{inner}	0.0	2.2	0.0	−1.3	0.97	0.1
ΔUA_{outer}	0.0	2.8	1.0	0.0	0.0	3.3
Controller	1.0	0.02	1.0	0.03	1.0	0.03

Figure 7.15: Reporter from statistical plots for all system problems studied

	$\frac{\partial F_{cw}}{\partial E}$	$\frac{\partial F_{cw}}{\partial T_{chill_{in}}}$	$\frac{\partial F_{cw}}{\partial T_{amb}}$
$Normal$	5.9(.7)	10.6(1.8)	12.5(1.3)
$\Delta C_{p_{chill}}$	7.4(.8)	12.7(1.4)	15.3(1.7)
$\Delta C_{p_{hot}}$	9.1(.9)	17.1(1.5)	13.7(1.9)
ΔUA_{inner}	8.0(.5)	15.2(1.5)	14.3(1.5)
ΔUA_{outer}	6.4(.9)	12.6(1.5)	8.0(1.6)
$Controller$	5.9(.9)	10.6(1.9)	12.5(1.3)

Figure 7.16: Reporters from relationship plots for all system problems studied

The statistical plots monitored include chill water flow (F_{cw}), outlet chill water temperature ($T_{cw_{out}}$), and exit energy associated with the chill water ($F_{cw} * T_{cw_{out}}$). Relationship plots monitored include F_{cw} vs. E, F_{cw} vs. $T_{cw_{in}}$, and F_{cw} vs. T_{amb}.

7.1.2.6 Decision tree

Based on elementary sensor statistics and reporter values tabulated in the previous section, a binary decision tree that systematically defines system problems can be designed. This tree incorporates engineering judgment and knowledge of the process with numerical results from HF-modeling and basic sensor statistics routines. Figure 7.17 presents such a decision tree, and the discussion that follows details its design.

As a coarse filter, overall sensor statistics are checked to flag stuck sensors. If the data is invalid, HF-modeling gives meaningless results. Once the data is validated, the HF-modeling reporters become meaningful. Separation on an overall statistical plot of F_{cw} distinguishes normal operation from system problems. If the separation is less than 1.0 (i.e., the plots overlap), the size of the two sigma window in each box is checked. Overly large variation indicates a poorly tuned controller, but one capable of running the process. Normal two-sigma values indicate normal process operation.

If separation on an overall statistical plot of F_{cw} is greater than 1.0, something has changed in the process. The objective now is to differentiate between a change in UA_{inner}, UA_{outer}, $C_{p_{chill}}$, and $C_{p_{hot}}$. The partial derivative

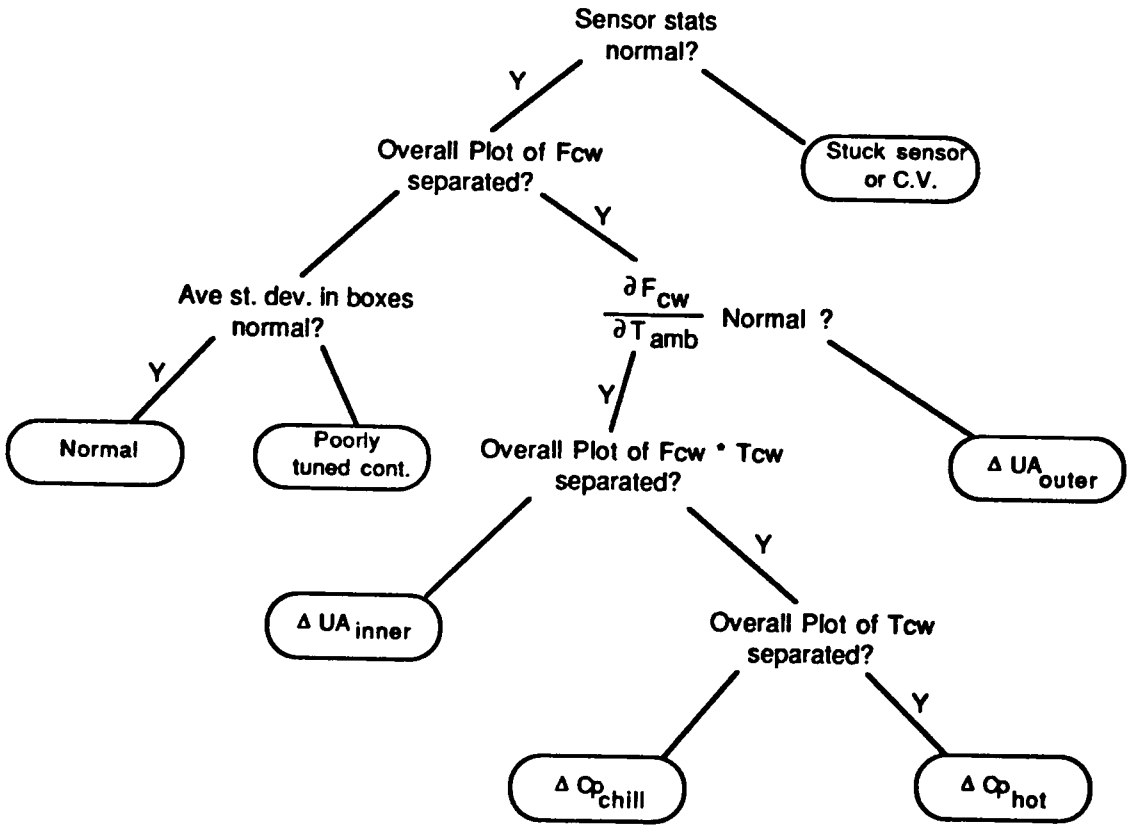


Figure 7.17: Decision tree for shell-and-tube heat exchanger

of chill water flow with respect to ambient temperature ($\frac{\partial F_{cw}}{\partial T_{amb}}$) is a sufficient indicator of a change in UA_{outer}. This is the only problem that significantly affects this partial derivative.

The two reporter values separation on an overall statistical plot of $T_{cw_{out}} * F_{cw}$ and separation on an overall plot of $T_{cw_{out}}$, can distinguish the remaining problems. Separation less than 1.0 on the first and greater than 1.0 on the second indicates a change in UA_{inner}. The converse confirms a change in $C_{p_{chill}}$. A separation greater than 1.0 on both indicates a change in $C_{p_{hot}}$. For redundancy, the value of $\frac{\partial F_{cw}}{\partial E}$ should be checked to confirm $\Delta C_{p_{hot}}$.

The decision tree could be made more complex by adding more redundancy checks and making more elaborate statistical tests. However, within the scope of the problems analyzed, this simplistic diagram is adequate 95% of the time. The major problems occur in cases of small problem magnitudes. This analysis assumes that only one system problem occurs at a given time or that one problem is significantly larger than all others. Such a restriction is rather severe. A decision tree for a maximum of two problems occurring at one time is presented in [17].

7.1.2.7 Shortcomings of decision tree analysis

The decision tree analysis requires a reasonable amount of design time and results in adequate performance. The system studied, however, is one of the simplest encountered by chemical engineers. For more complex chemical processes, this analysis leads to some shortcomings.

As system complexity increases, binary decision trees grow exponentially. In addition, the more complicated the process, the more likely two or more problems will occur simultaneously. Both of these problems entangle decision tree designs for complex systems. An engineer could conceivably develop an effective design, but the time required might be too costly.

Other shortcomings of the decision trees analysis exist. One is a lack of indicating when new system problems (ones not previously experienced) occur; such an indication is vital. Another problem lies in incorporating new system problems into the decision tree. These shortcomings are the driving force behind incorporating pattern recognition techniques and neural networks into the analysis.

7.1.3 Analyzing historical data directly with neural networks

This section describes a study in which the multilayer perceptron network (see Section 5.5) classifies raw historical data into distinct classes of system problems. The heat exchanger is simulated under normal operation and under each of the system problems (one at a time). The network is then trained with each of these "teacher" data sets. The following paragraphs outline specifics of the historical data and how they are presented to the network.

All sensor information available from the system at each time sample is presented to the network, with the exception of time. Because inputs presented to each node of the neural network's input buffer must lie between zero and one, historical data must be scaled before presented to the network. In this

example, historical data are physically scaled (see Section 6.2) such that the minimum sensor reading is scaled to zero and the maximum sensor value is scaled to one. This scaled result is the vector presented to the network for training and classification.

The network must be trained with enough data to fully represent each system problem (or normal operation) over the range of conditions encountered by the process. The amount of training data must be weighed against training time. In the heat exchanger example, a reasonable trade-off was found to be 1000 records from each teacher data set (data is sampled approximately every 30 seconds during simulation). Section 5.4 outlines neural network training. 85 to 90 percent of the data in each teacher set must be correctly classified to complete training.

After training, the network is ready to classify a set of data from some unknown condition. For example, Figure 7.18 illustrates the classification of one record from a typical data set. In this example, the network has been trained for only three problem classes: normal operation, change in heat capacity of the chill water, and change in inner heat transfer coefficient. Presenting the record to the input buffer of the network results in the outputs shown. Because the output of the node corresponding to ΔUA_{inner} is above the threshold (0.80 in this study), the record is classified as ΔUA_{inner}.

Of course, many such scaled records comprise a data set. The network classifies the database one record at a time. Overall classification of the data set is in terms of the percentage of data classified into each system problem.

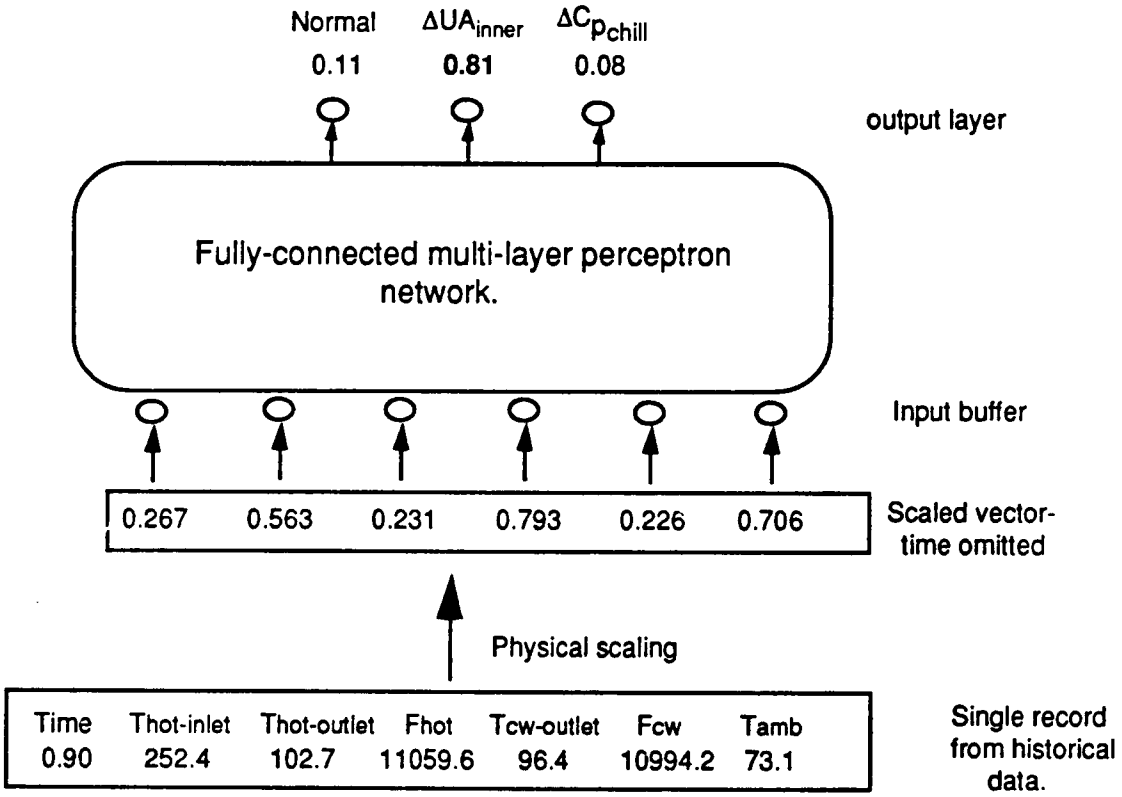


Figure 7.18: Example network classification of a single record

7.1.3.1 Neural network performance

Because of the lengthy learning time and the difficulty of converging on all training sets, a simple two-class example was studied first. Other classes of problems were added one at a time. The following is a summary of those studies.

Two-class study

Figure 7.19 summarizes the training conditions and results of this two-class example. The network was trained with data representing $\Delta C_{p_{hot}}$ and data representing ΔUA_{inner}. The trained network is tested by classifying new data (data other than that used in training). For this two-class case, the results are very promising.

Three-class study

Figure 7.20 summarizes results from a three-class study. The normal or basecase operation class has been added to the previous classes. Once again, the network performs reliably in classifying data representing a problem magnitude equal to or greater than that of the teacher data. However, the performance degrades in classifying problems of a lower magnitude than that of the teacher data.

I. Network

Input layer: 6
1st hidden layer: 12
2nd hidden layer: 12
Output layer: 2

II. Training

Class 1: ΔCp_{hot} (+10%)
Class 2: ΔUA_{inner} (-10%)
Class ∅: Unknown (null)

Number of records in each training set: 1000 (sampled every 30 seconds)
Training Parameters:
Gain = 0.32
Iterations = 70
Minimum output for classification = 0.80
Threshold = 0.0
% of vectors correctly classified to end training = 90%
Training time (CPU): 1hr 45min

III. Classifying 'new' data [1000 records]

	% Classified		
Problem	ΔCp_{hot}	ΔUA_{inner}	Null
ΔCp_{hot} (+10%)	**100**	0	0
ΔUA_{inner} (-8%)	5	**93**	2
ΔCp_{hot} (+10%) & ΔUA_{inner} (-8%)	**79**	21	0
ΔCp_{hot} (+10%) & ΔUA_{inner} (-8%)	29	**69**	2

Figure 7.19: Summary of two-class example

I. Network

Input layer: 6
1st hidden layer: 12
2nd hidden layer: 12
Output layer: 3

II. Training

Class 1: Normal Operation
Class 2: ΔUA_{inner} (-8%)
Class 3: ΔCp_{hot} (+10%)
Class Ø: Null

Number of records in each training set: 1000
Training Parameters:
Gain = 0.32
Iterations = 70
Minimum output for classification = 0.80
Threshold = 0.0
% of vectors correctly classified to end training = 90%
Training time (CPU): 2hrs

III. Classifying 'new' data [1000 records]

Problem	Normal	ΔUA_{inner}	ΔCp_{hot}	Null
Normal Operation	**75**	20	0	5
ΔUA_{inner} (-8%)	10	**85**	0	5
ΔCp_{hot} (+10%)	0	0	**100**	0
ΔCp_{hot} (+10%) & ΔUA_{inner} (-8%)	10	35	**40**	45
ΔCp_{hot} (+13%)	0	0	**100**	0
ΔCp_{hot} (+7%)	0	5	**45**	50
ΔUA_{inner} (-10%)	5	**90**	0	5
ΔUA_{inner} (-6%)	20	**39**	0	41

Figure 7.20: Summary of three-class example

Four-class study

Figure 7.21 summarizes a four-class study that includes normal operation, ΔUA_{inner}, $\Delta C_{p_{hot}}$, and $\Delta C_{p_{chill}}$. The network was satisfactorily trained after a lengthy training period. Performance studies on the trained network demonstrated that the network can not reliably identify process problems over the range of problem magnitude desired. In classifying data with the same magnitude of problem as the teacher data, its performance was marginal, but performanc was unacceptable in identifying smaller problem magnitudes.

Although the neural network shows potential in detecting and diagnosing system problems by examining raw process data, its performance is unacceptable when four or more problems are involved on the heat exchanger system. Many adjustments to the network were made to attempt to improve network performance, including fine-tuning many learning parameters and adjusting the number of hidden-layer nodes. These efforts accomplished only minor improvements. Probably the most effective method in improving network performance lies not in the network itself, but in preprocessing information fed to the network.

7.1.4 Analyzing cultivated data with neural networks

This section presents results of the multilayer perceptron classifying reduced information databases from HF-modeling. The ability of HF-modeling to reduce the size of the database should significantly reduce training time.

I. Network

Input layer: 6
1st hidden layer: 12
2nd hidden layer: 12
Output layer: 4

II. Training

Class 1: Normal Operation
Class 2: ΔUA_{inner} (-8%)
Class 3: ΔCp_{hot} (+10%)
Class 4: ΔCp_{chill} (+15%)
Class ∅: Null

Number of records in each training set: 1000
Training Parameters:
Gain = 0.32
Iterations = 70
Minimum output for classification = 0.80
Threshold = 0.0
% of vectors correctly classified to end training = 90%
Training time (CPU): 6hrs

III. Classifying 'new' data [1000 records]

Problem	Normal	ΔUA_{inner}	ΔCp_{hot}	ΔCp_{chill}	Null
Normal Operation	**75**	6	0	14	5
ΔUA_{inner} (-8%)	10	**70**	4	16	0
ΔCp_{hot} (+10%)	11	7	**65**	10	7
ΔCp_{chill} (+15%)	6	31	12	**50**	1
ΔUA_{inner} (-6%)	40	**25**	0	0	35
ΔCp_{hot} (+7%)	22	5	**27**	6	40
$\Delta Cpchill$(+12%)	12	64	3	**20**	1

Figure 7.21: Summary of four-class example

Its ability to enhance multivariable statistics and extract subtle relationships should provide the network with a much clearer picture of system operation.

Figure 7.9 depicts the reduced database that results when HF-modeling is applied to the heat exchanger. Each line (record) summarizes important statistics and relationships from one operating condition. The first three elements of each record are integer values of $F_{hot} * T_{hot_{in}}$, T_{cw}, and T_{amb}. Those values comprise the address of the operating condition summarized by each record. The next six elements are a series of average-standard deviation pairs for important system variables (F_{cw}, $T_{cw_{out}}$, $F_{cw} * T_{c_{out}}$). The final three elements are global partial derivatives. Section 6.2 describes incorporating these global partial derivatives into the reduced database. It should be noted that each record of this reduced information database is physically scaled.

The network is initially trained with six data sets, one each for basecase, ΔUA_{inner}, ΔUA_{outer}, ΔCp_{chill}, ΔCp_{hot}, and the poorly tuned controller. The training data for each problem were represented by only a single problem magnitude. The network learns each problem and distinguishes reliably between the training data with nearly 100% accuracy, an accomplishment not attained by training the same network with raw historical data.

The trained network's performance is now evaluated by classifying data sets that represent three cases of each process problem. In the first, new data has the same magnitude of problem as the training set. In the second, data represents different and varying magnitudes of system problems. In the third, data sets reflect multiple (two or three) problems occurring simultaneously. As

before, overall classification of an entire data set is calculated in terms of the percentage of records classified into each problem class.

Figure 7.22 presents results of the first case of classifying data with the same magnitude of system problem as the teacher data. The network clearly performs well in all cases. These results are much improved over training with raw historical data. Figure 7.23 summarizes results of the second case, classifying varying magnitudes of each problem. The network performs well for magnitudes larger than that of the training data; however, for problems of a somewhat lesser magnitude, the classification shifts towards the basecase and null classes. The shift towards basecase is desirable, but the shift to the null class is not. These results can be understood through the decision boundary studies of Section 5.5.

Figure 7.24 summarizes the results of case three: multiple problem studies. In general, classification favors the problem with the more significant impact on the system. If problems are of relatively equal magnitude, the network tends to classify part in each problem class and part in the null class. Overall, this method performs reasonably well when multiple problems occur simultaneously.

This example demonstrates a vast improvement in training the network with cultivated data over raw historical data. This supports the assertion that the manner information is presented to the network is at least as important as the network itself. In this case, HF-modeling extracts subtle features, providing a much clearer distinction between problem classes.

I. Network

Input layer: 9
1st hidden layer: 20
2nd hidden layer: 20
Output layer: 6

II. Training

Class 1: Basecase
Class 2: ΔUA_{inner}
Class 3: ΔUA_{outer}
Class 4: ΔC_{phot}
Class 5: ΔC_{pchill}
Class 6: Poorly tuned controller
Class Ø: Unknown (null)

Number of records in training sets: ~110

III. Classification

Class of Problem	Magnitude	% of Data Classified 1	2	3	4	5	6	Ø
....1	--	**95**	4	0	1	0	0	0
2	8%	0	**91**	0	0	7	1	1
3	25%	0	0	**80**	19	1	0	0
4	10%	0	1	1	**92**	2	1	3
5	15%	1	7	0	1	**86**	1	4
6	--	0	0	0	0	0	**99**	1

Figure 7.22: Summary of network classification of data with same magnitude as training data

I. Network

Input layer: 9
1st hidden layer: 20
2nd hidden layer: 20
Output layer: 6

II. Training

Class 1: Basecase
Class 2: ΔUA_{inner}
Class 3: ΔUA_{outer}
Class 4: ΔC_{phot}
Class 5: ΔC_{pchill}
Class 6: Poorly tuned controller
Class Ø: Unknown (null)

Number of records in training sets: ~110

III. Classification

Class of Problem	Magnitude	% of Data Classified 1	2	3	4	5	6	Ø
2	10%	3	**79**	0	0	9	0	9
2	6%	25	**43**	0	1	10	0	21
3	30%	0	0	**85**	11	0	0	4
3	20%	0	0	**84**	10	3	0	3
4	13%	0	0	5	**95**	0	0	0
4	7%	0	1	1	**82**	6	0	10
5	18%	5	2	1	1	**82**	0	9
5	12%	36	3	0	0	**32**	2	27

Figure 7.23: Classifying system problems of varying magnitudes

I. Network

Input layer: 9
1st hidden layer: 20
2nd hidden layer: 20
Output layer: 6

II. Training

Class 1: Basecase
Class 2: ΔUA_{inner}
Class 3: ΔUA_{outer}
Class 4: ΔC_{phot}
Class 5: ΔC_{pchill}
Class 6: Poorly tuned controller
Class Ø: Unknown (null)

Number of records in training sets: ~110

III. Classification

Class of Problem	Magnitude	% of Data Classified 1	2	3	4	5	6	Ø
2&3	--	0	**0**	**92**	6	0	0	2
2&4	--	0	**0**	1	**99**	0	0	0
2&5	--	1	**5**	0	6	**57**	0	30
2&6	--	0	**0**	0	0	0	**94**	6
3&4	--	0	0	**1**	**99**	0	0	0
3&5	--	0	0	**99**	1	**0**	0	0
3&6	--	0	0	**88**	4	0	**3**	5
4&5	--	0	0	10	**90**	**0**	0	0
2&5&6	--	1	**1**	1	3	**6**	**43**	45
2&3&5	--	0	**0**	**94**	5	**0**	0	1
2&4&5	--	0	**0**	5	**95**	**0**	0	0

Figure 7.24: Classifying multiple problems

This framework still has some significant problems. First, it should classify all magnitudes of each system problem equally well. Second, it should clearly define problem magnitude. Third, the occurrence of new system problems should be identified. Finally, the framework should indicate when two or more problems are indistinguishable. Interestingly enough, all these problems can be addressed by proper scaling. The next example, a CSTR, demonstrates the entire analysis in a more refined state and implements the basecase normalized scaling policy discussed in Section 6.2 to resolve these problems.

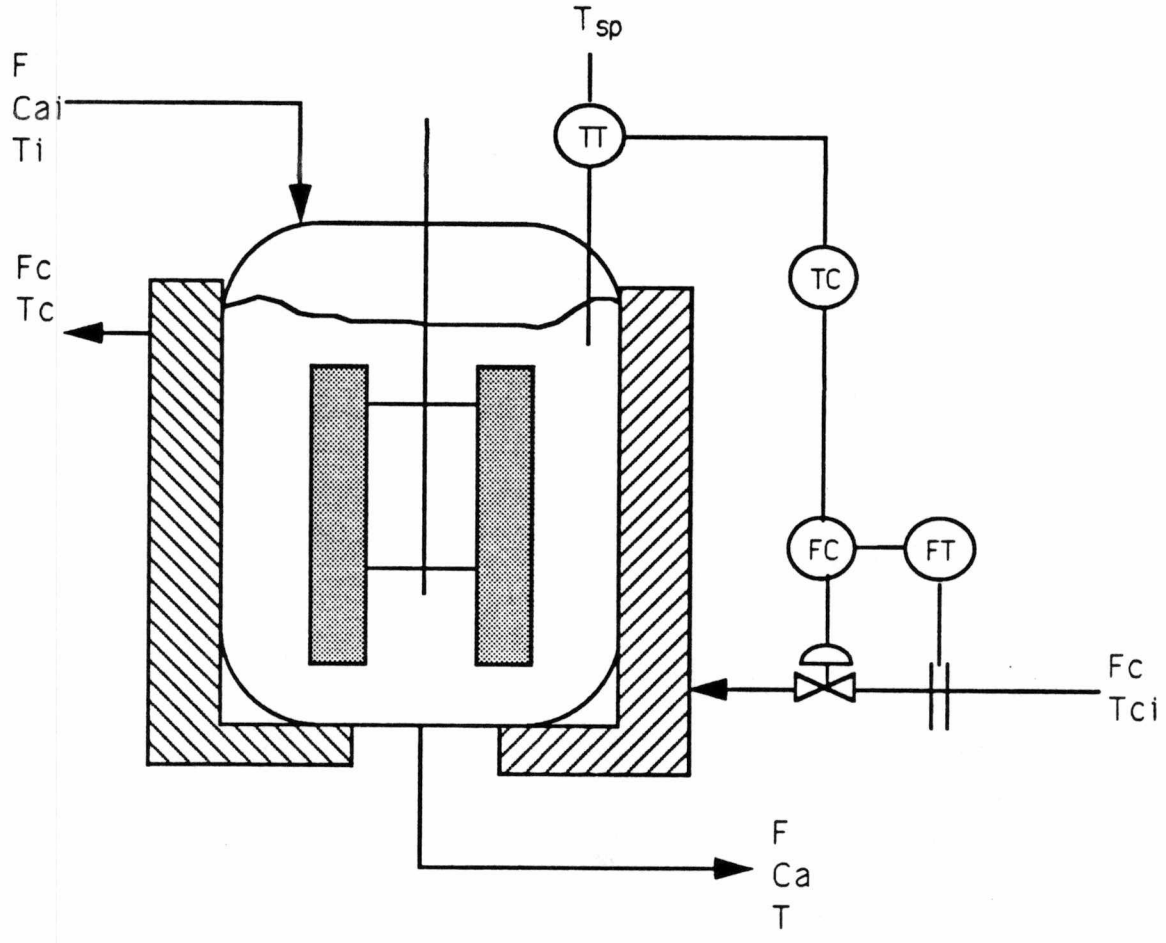
7.2 Reactor Example

This example demonstrates the refined cultivation framework for diagnosing system problems. It will exemplify HF-modeling, TSD analysis, scaling methods, and the problem recognition framework applied to data from a simulated continuous stirred-tank reactor.

7.2.1 The system studied

Figure 7.25 shows the nonlinear CSTR. This simulated reactor is an adapted version from example 9.5 of Smith and Corripio[70]. The desired reaction is a second order, catalyzed reaction in which component A reacts to form product. In addition, an undesired side reaction occurs in which component B forms a contaminant C.

System variables sampled on-line are temperature, flow rate, and com-



Reactions:

2[A] ==> [Prod] (Catalyzed)

[B] ==> [C]

Figure 7.25: Continuous stirred-tank reactor

position of component A in the inlet reactant stream; temperature and composition of component A in the reactor; flow rate and temperature of the inlet chill water stream; and outlet chill water temperature.

The desired product composition is controlled via the reaction temperature. A standard velocity algorithm PID controller manipulates the chill water flow through the reactor jacket. Figure 7.25 depicts the control loop.

The reactor simulation is accomplished through a Runge-Kutta numerical integration of a series of nonlinear differential equations describing this system. Figures 7.26 and 7.27 outline those equations. Figure 7.28 summarizes the CSTR parameters like reaction rate constants and heat transfer coefficients. Basecase flow rates, compositions, and temperatures are also given. Underlying assumptions in this simulation include constant tank level ($F_{r_{in}} = F_{r_{out}}$), equal inlet and outlet chill water flow, uniform chill water temperature in the jacket, and identical conditions (temperatures and concentrations) inside the reactor and in the product stream.

Like the heat exchanger, this system was operated under conditions typical in a real environment. The system experiences disturbances in several inputs: $F_{r_{in}}$, $C_{a_{in}}$, $T_{r_{in}}$, and $T_{cw_{in}}$. Figures 7.29 and 7.30 exemplify typical profiles of these variables over a three hour time frame. These disturbances consist of white noise superimposed on integrated, autoregressive, moving average functions. Figure 7.31 depicts typical three-hour manipulated variable (F_{cw}) and controlled variable (T_r) profiles.

An important process characteristic is an inverse response of reactor

Mass Balance on Reactant A:

$$\frac{\partial C_a}{\partial t} = \frac{F}{V}(C_{ai} - C_a) - k\alpha C_a^2$$

Energy Balance on Reactor:

$$\frac{\partial T_c}{\partial t} = \frac{F}{V}(T_i - T) - \frac{\Delta H_r \alpha}{\rho C_p} k C_a^2 - \frac{UA}{V\rho C_p}(T - T_c) - \frac{\Delta H_{r2}}{\rho C_p} k_2 C_b$$

Jacket Energy Balance:

$$\frac{\partial T_c}{\partial T} = \frac{UA}{V_c \rho_c C_{pc}}(T - T_c) - \frac{F_c}{V_c}(T_c - T_{ci})$$

Reaction Rate Coefficient:

$$k = k_0 e^{-\frac{E}{R(T+273.16)}} \qquad k_2 = k_{02} e^{-\frac{E_2}{R(T+273.16)}}$$

Temperature Sensor Lag:

$$\frac{\partial b}{\partial t} = \frac{1}{\tau_T}\left(\frac{T - T_M}{\Delta T_T} - b\right)$$

PI Controller:

$$\frac{\partial y}{\partial t} = \frac{1}{\tau_I}(m - y) \qquad 0 \le m \le 1$$

$$m = y + K_c\left(\frac{T^{set} - T_M}{\Delta T_T} - b\right)$$

Equal % Control Valve:

$$F_c = F_{c_{max}} \gamma^{-m}$$

Figure 7.26: Differential equations describing reactor

C_a = concentration of the reactant in reactor, kgmole / m^3.

C_{ai} = concentration of reactant in feed, kgmole / m^3.

C_{bi} = concentration of contaminent in feed, kgmole / m^3.

T = reactor temperature, C.

T_c = jacket temperature, C.

T_i = feed temperature, C.

T_{ci} = inlet coolant temperature, C.

b = transmitter signal (0 to 1).

F = feed flow rate, m^3 / s.

V = reactor volume, m^3.

k = primary reaction rate coefficient, m^3 / kgmole s.

k_2 = side reaction rate coefficient, m^3 / kgmole s.

ΔH_r = heat of primary reaction, J / kgmole.

ΔH_{r2} = heat of side reaction, J / kgmole.

ρ = density of reactor contents, kgmole / m^3.

C_p = reactant heat capacity, J / kgmole C.

U = overall heat transfer coefficient, j / s m^2 C.

A = heat transfer area, m^2.

V_c = jacket volume, m^3.

ρ_c = density of coolant, kg / m^3.

C_{pc} = specific heat of coolant, j / kg C.

ΔT_T = transmitter calibrated range, C.

F_c = coolant flow rate, m^3 / s.

T_M = lower limit of temp transmitter, C.

τ_T = temperature sensor time constant, s.

τ_I = controller integral time, s.

m = controller output signal (0 to 1).

K_c = controller gain.

$F_{c_{max}}$ = maximum flow through control valve, m^3 / s.

γ = valve rangeability parameter.

α = catalyst activity (0 to 1).

k_o = Arrhenius frequency parameter for primary reaction, m^3 / s kgmole

E = activation energy of primary reaction, J / kgmole.

E_2 = activation energy of secondary reaction, J / kgmole.

R = ideal gas law constant, 8314.39 j / kgmole K.

Figure 7.27: Definition of variables

$V = 7.08\ m^3$	$\Delta H_r = -9.86 \times 10^7\ J/kgmole$
$\rho = 19.2\ kgmole/m^3$	$\Delta H_{r2} = -5.0 \times 10^7\ J/kgmole$
$C_p = 1.815 \times 10^5\ j/kgmole\ C$	$U = 3550\ J/s\ m^2\ C$
$A = 5.40\ m^2$	$V = 1.82\ m^3$
$\rho_c = 1000\ kg/m^3$	$C_{pc} = 4184\ J/kg\ C$
$k_0 = 0.0744\ m^3/s\ kgmole$	$E = 1.182 \times 10^7\ J/kgmole$
$k_{02} = 0.050\ m^3/s\ kgmole$	$E_2 = 1.01 \times 10^7\ J/kgmole$
$\tau_T = 20\ s$	$F_{c_{max}} = 0.020\ m^3/s$
$\alpha = 1.0$	$T_M = 80\ C$
$\gamma = 50$	$\Delta T_T = 20\ C$

Figure 7.28: Summary of important reaction parameters

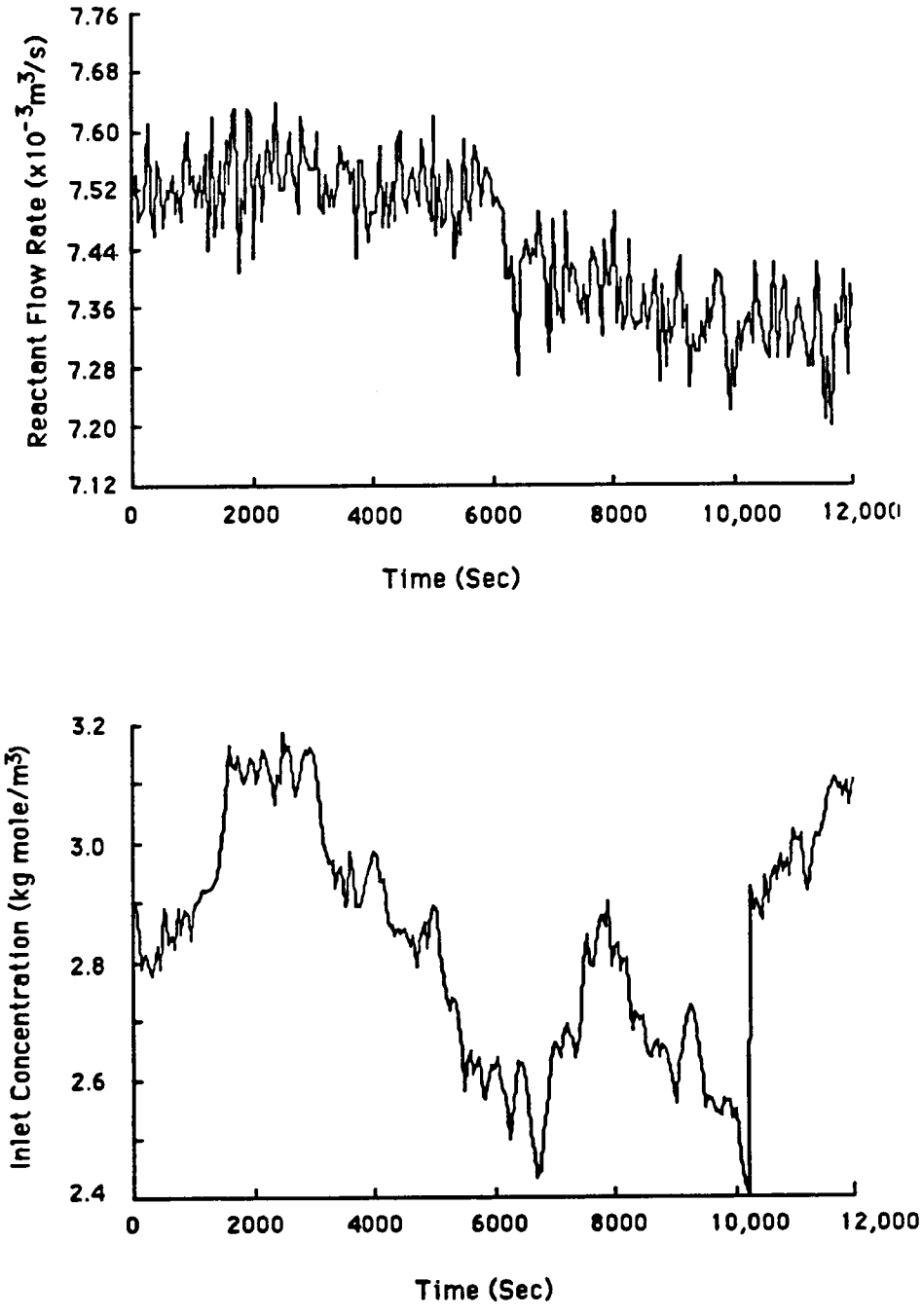


Figure 7.29: Typical disturbance profiles

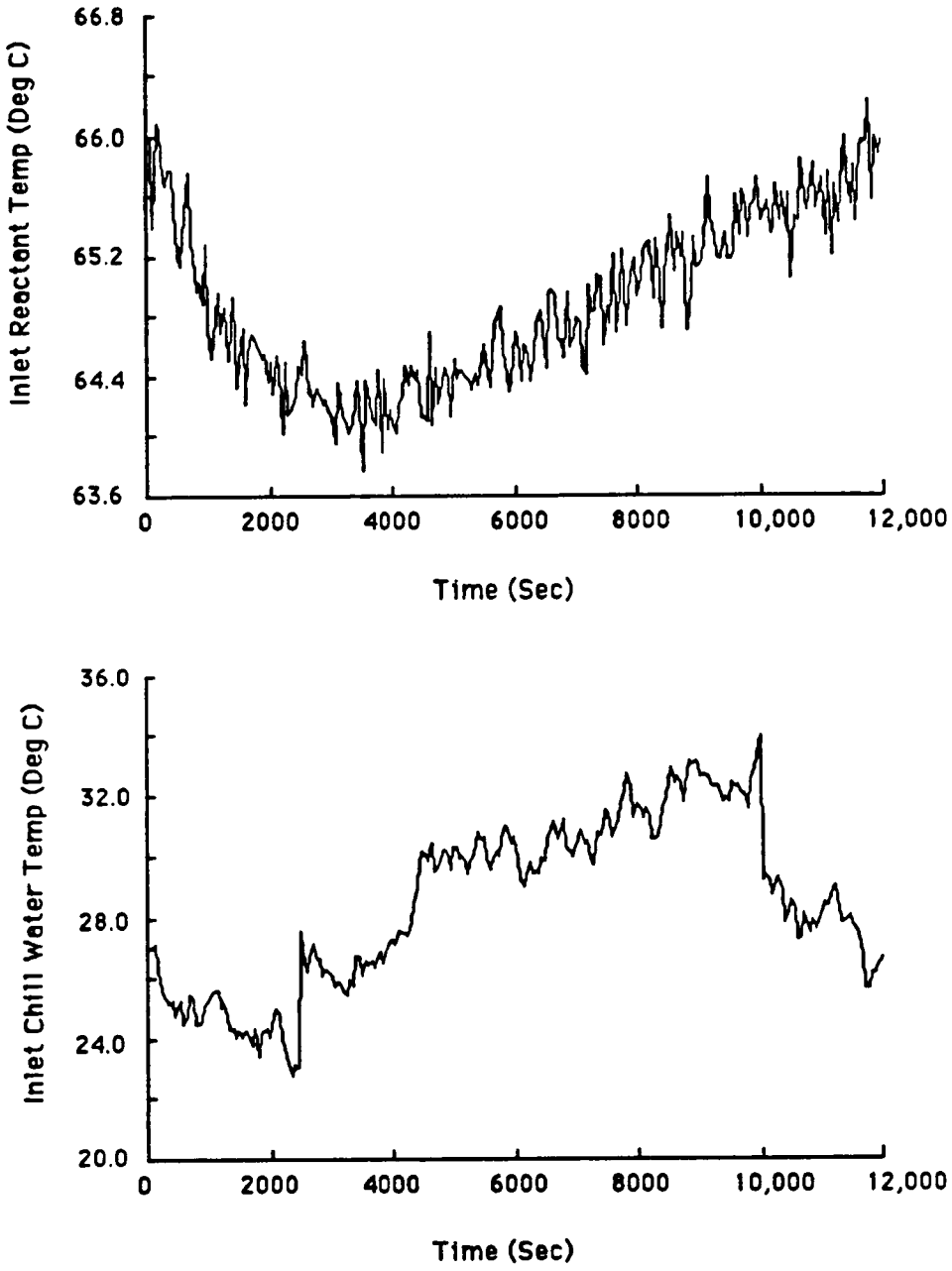


Figure 7.30: Typical disturbance profiles

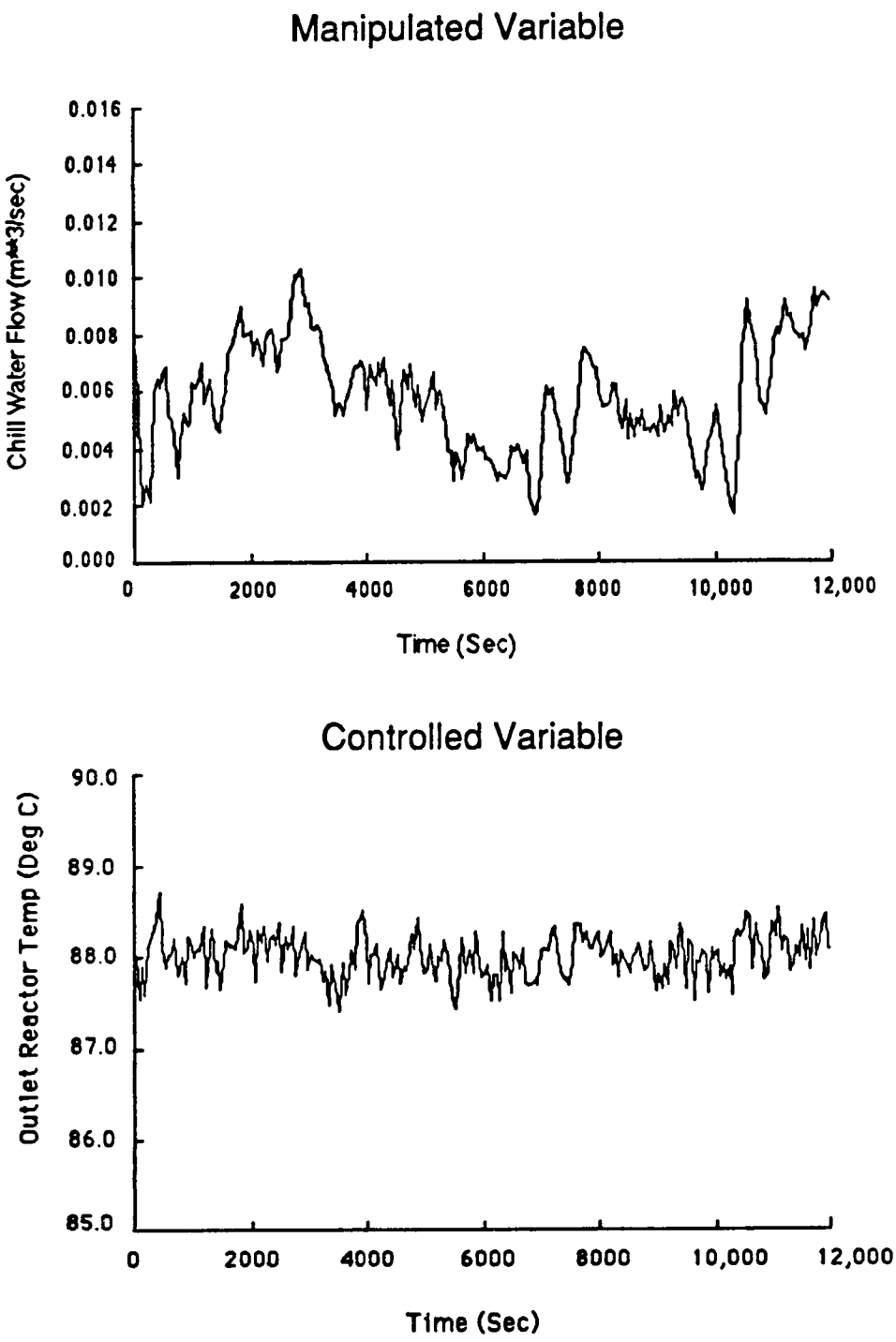


Figure 7.31: Typical manipulated and controlled variable responses

temperature to chill water flow. Such a response makes controller tuning significantly more difficult.

The system problems studied in this example follow.

- reduced heat transfer coefficient
- reduced catalyst activity
- significant side reaction
- change in heat capacity of chill water
- change in heat capacity of process stream
- poorly tuned controller

The following analysis details HF-modeling, TSD analysis, scaling, and the problem recognition framework applied to the CSTR. Through this example, important subtleties of each tool will be brought forward.

7.2.2 HF-modeling of the CSTR

This section describes the hypothesis modeling of the CSTR, closely following the step-by-step procedure outlined in Section 3.2. The hypothesis model formulated for the reactor system is

$$Y_{dep} = f(C_{a_{in}}, F_{r_{in}} * T_{r_{in}}, F_{cw} * T_{cw_{in}}).$$

This model states that the dependent variables are primarily a function of inlet concentration of component A in the feed, energy of the inlet reactant stream, and energy of the inlet chill water stream.

As section 3.2 explains, the next step in HF-modeling defines resolution of each variable comprising the model. Defining resolution requires an understanding of each component's relative impact on the system. The resolution of each variable comprising the CSTR model is summarized below:

Process Variable	Zero	Span	Divisions
$C_{a_{in}}$	1.5	2.5	20
$F_{r_{in}} * T_{r_{in}}$	0.20	0.40	25
$F_{cw} * T_{cw}$	$5x10^{-4}$	0.995	40

Next, historical data is sorted into the reduced information database. Important statistics and relationships of measured dependent and manipulated variables comprise the reduced database. Often, interactions between these variables must be considered. Examples include products of flow rate and temperature (e.g. $F_r * T_r$) or products of flow rate and composition (e.g. $F_r * C_a$). Figure 7.32 illustrates a typical reduced database employed in this example. The first three elements, integer values of $C_{a_{in}}$, $F_{r_{in}} * T_{r_{in}}$, and $F_{cw} * T_{cw_{in}}$, address the operating condition represented in each line of the database. The average F_{cw} in each condition and its standard deviation are the next two elements. Averages of T_r, $T_{cw_{out}}$, $F_r * T_r$, $F_{cw} * (T_{cw_{out}} - T_{cw})$, and $C_{a_{out}}$ comprise the next

Ca	Er	Ecw	Fcw	σ-Fcw	Tr	Tcw-out	Er-out	ΔEcw	Ca-out	$\frac{\partial F_{cw}}{\partial(F_{f_{in}} * T_{f_{in}})}$
7	8	8	2.51E-03	6.544E-04	87.96	63.48	0.649	0.102	1.01	9.23
7	8	10	2.43E-03	9.409E-04	87.92	64.87	0.658	9.42E-02	1.10	8.87
7	8	11	2.79E-03	8.378E-04	87.90	65.23	0.655	0.103	1.12	9.27
7	9	11	3.45E-03	8.476E-04	87.89	61.32	0.661	0.115	1.33	9.01
7	9	13	3.96E-03	7.337E-04	87.93	61.89	0.656	0.119	1.38	8.77
7	10	8	3.42E-03	8.596E-04	87.93	59.02	0.665	0.126	1.55	9.55
7	10	11	4.64E-03	7.669E-04	87.92	57.61	0.649	0.137	1.68	9.29
7	10	12	4.96E-03	7.370E-04	87.87	57.67	0.649	0.137	1.79	9.01
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
14	11	9	1.93E-02	4.723E-04	88.48	36.14	0.668	0.238	1.98	10.30

Figure 7.32: Example reduced database from CSTR

five columns. In the final column lies the calculated value for $\frac{\partial F_{cw}}{\partial F_{r_{in}} * T_{r_{in}}}$ within each operating condition.

Figure 7.33 illustrates a typical multivariable statistical plot of F_{cw}, representing normal or baseline operation. Figure 7.34 shows baseline operation for $F_r * T_r$. A relationship plot (F_{cw} vs $F_{r_{in}} * T_{r_{in}}$) from basecase conditions is illustrated in Figure 7.35. In the HF analysis of a new data set, these baseline graphs and multivariable plots of the remaining dependent variables comprising the reduced database are compared against those of new data. The following paragraphs outline a typical analysis.

7.2.2.1 Typical data analysis

When analyzing data from a 24-hour CSTR simulation, HF-modeling seeks to answer two basic questions: Are there deviations from normal operation? If so, what system problem is evidenced? For this example, associated with this particular set of data is a -7% change in heat transfer coefficient. The analysis of this data is detailed below, along with a comparison to single-variable statistical analysis.

This example demonstrates the advantage of monitoring statistics from a multivariable framework. For the set of data under consideration, a Shewhart control chart (single-variable) is compared with a multivariable statistical plot of F_{cw}. Figure 7.36 depicts the Shewhart control chart for the data set. The chart plots hourly averages of chill water flow over the time represented by the data. This analysis may lead one to believe operation is normal. However,

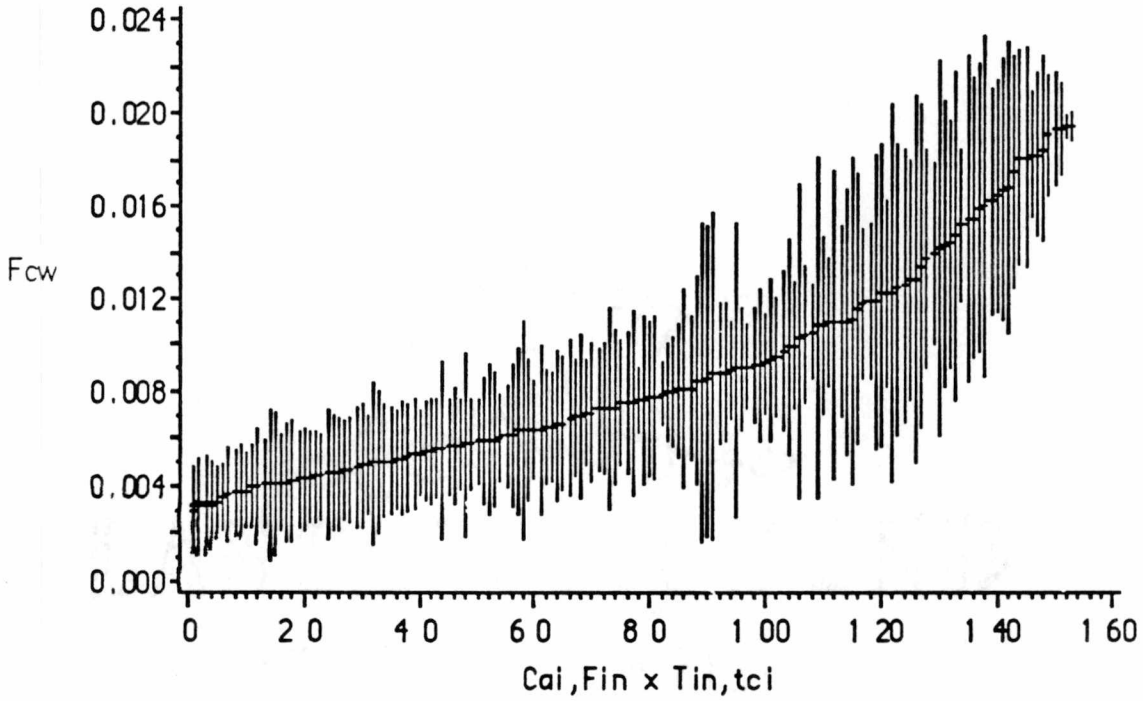


Figure 7.33: Multivariable statistical plot of F_{cw} for normal operation

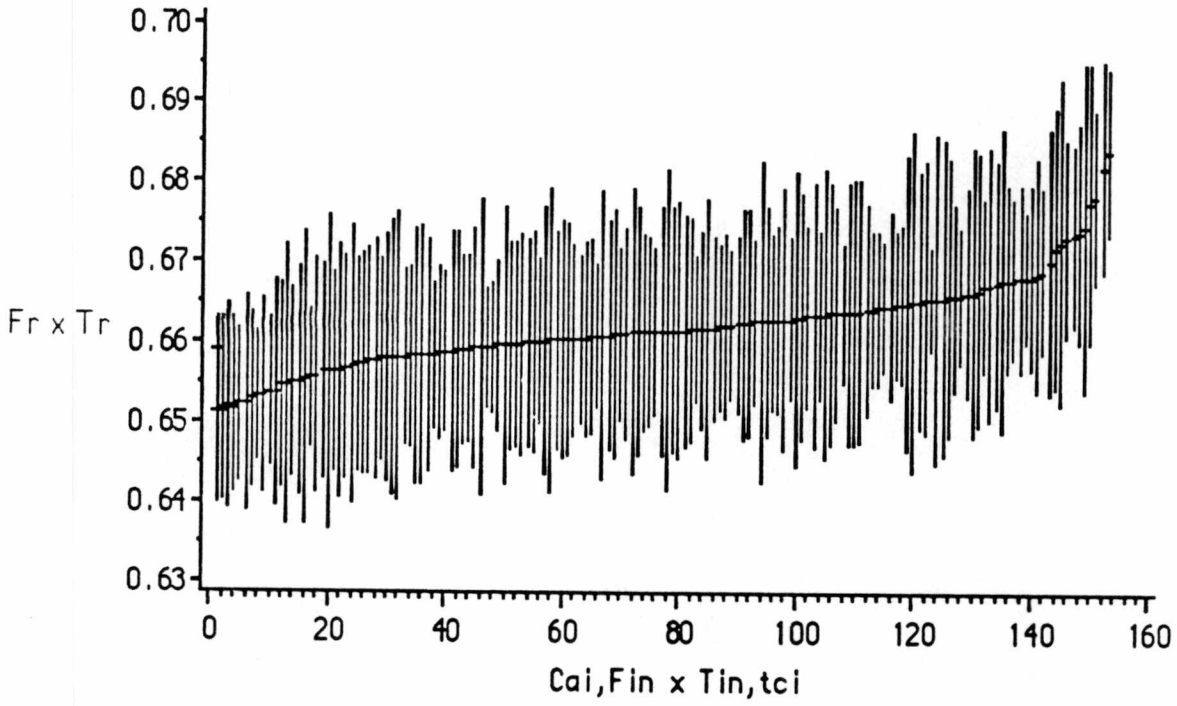


Figure 7.34: Multivariable statistical plot of $F_r * T_r$ for normal operation

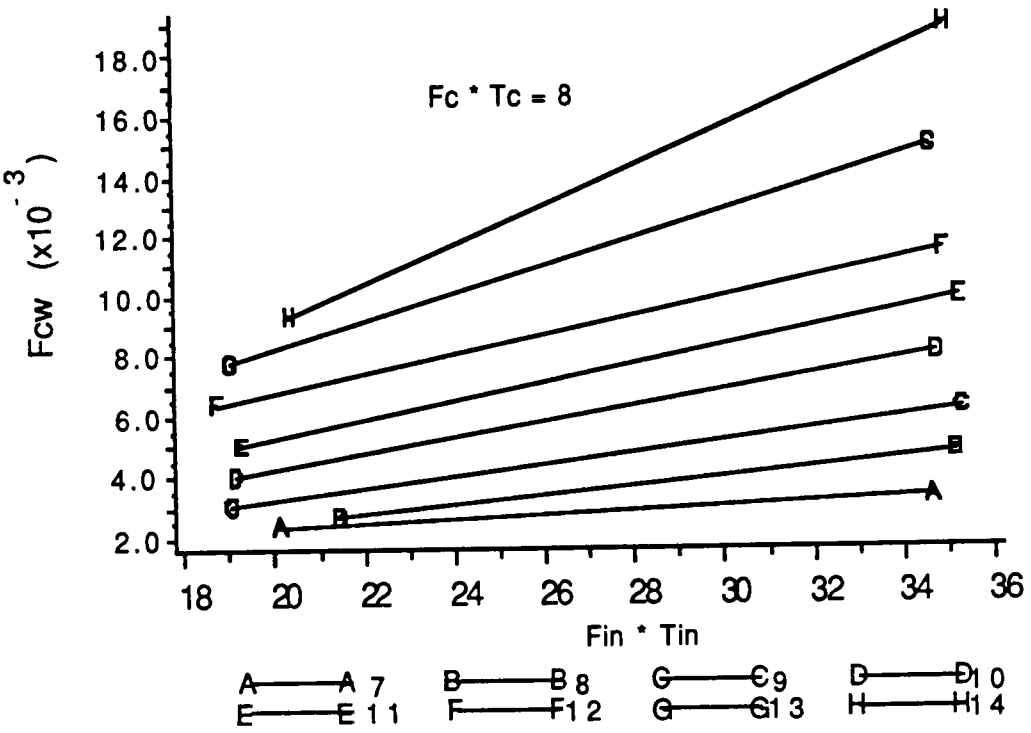


Figure 7.35: Relationship plot from normal operation

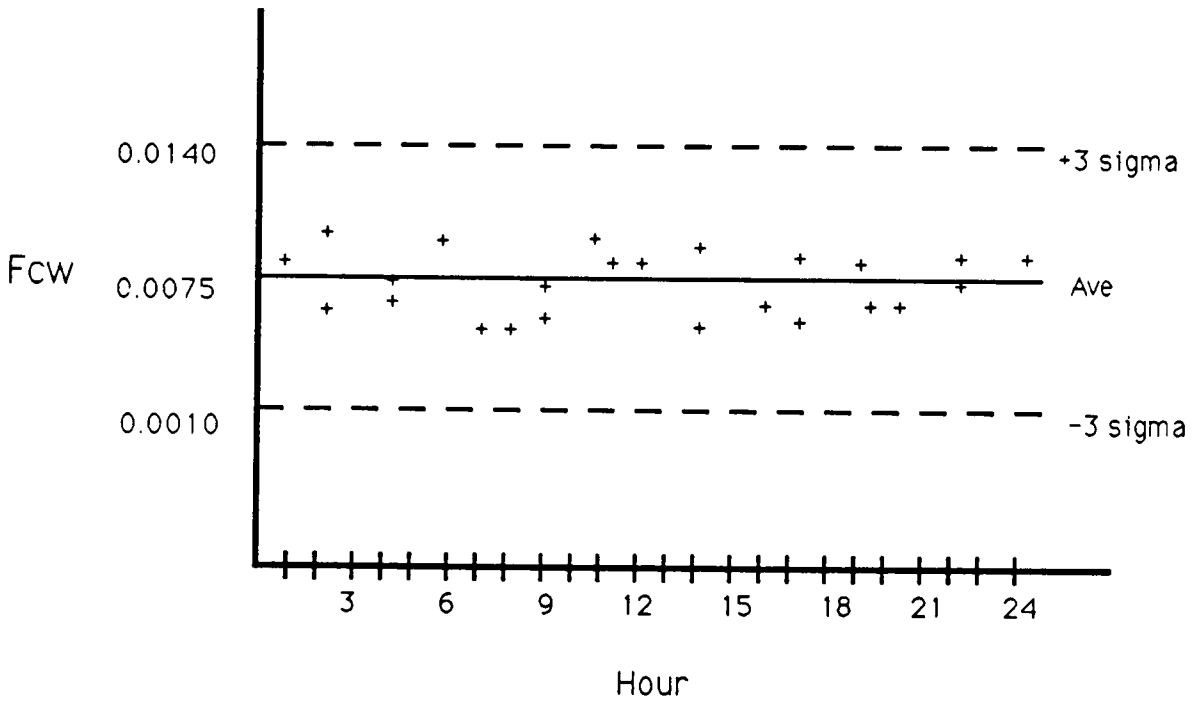


Figure 7.36: Shewhart control for data under consideration

a multivariable statistical plot of F_{cw} from the same data (see Figure 7.37) shows a subtle change. Although all averages lie within the six-sigma tolerance of normal operation, the mean has clearly shifted, denoting a change in the process.

In this case, the multivariable analysis detected a subtle process change, while the more commonly used single-variable analysis did not. Such a situation is not uncommon. Single-variable analysis effectively detects only pronounced process changes. Also a single-variable analysis will frequently indicate a process change when none exists. The significant reduction of these false alarms is another argument for multivariable statistical plots.

Multivariable statistical plots of the other dependent and manipulated variables (T_r, $T_{cw_{out}}$, $F_r * T_r$, $F_{cw}(T_{cw_{out}} - T_{cw})$, and $C_{a_{out}}$), in addition to F_{cw}, must be analyzed to get a complete picture of process operation. Like the heat exchanger application, fractional overlap and separation are calculated for each. Figure 7.38 shows a multivariable statistical plot of $F_r * T_r$ for the data under analysis. Notice that the data is not consistantly above or below average baseline operation for this variable, hence $F_r * T_r$ does not significantly change under this particular problem.

Figure 7.39 illustrates a typical relationship plot for the same data set. This plot depicts the best least-squares linear fit for the relationship, F_{cw} vs $F_r * T_r$. The overall partial derivative, $\frac{\partial F_{cw}}{\partial F_r * T_r}$, can be compared against that for basecase operation.

The above example summarizes the relatively few multivariable statis-

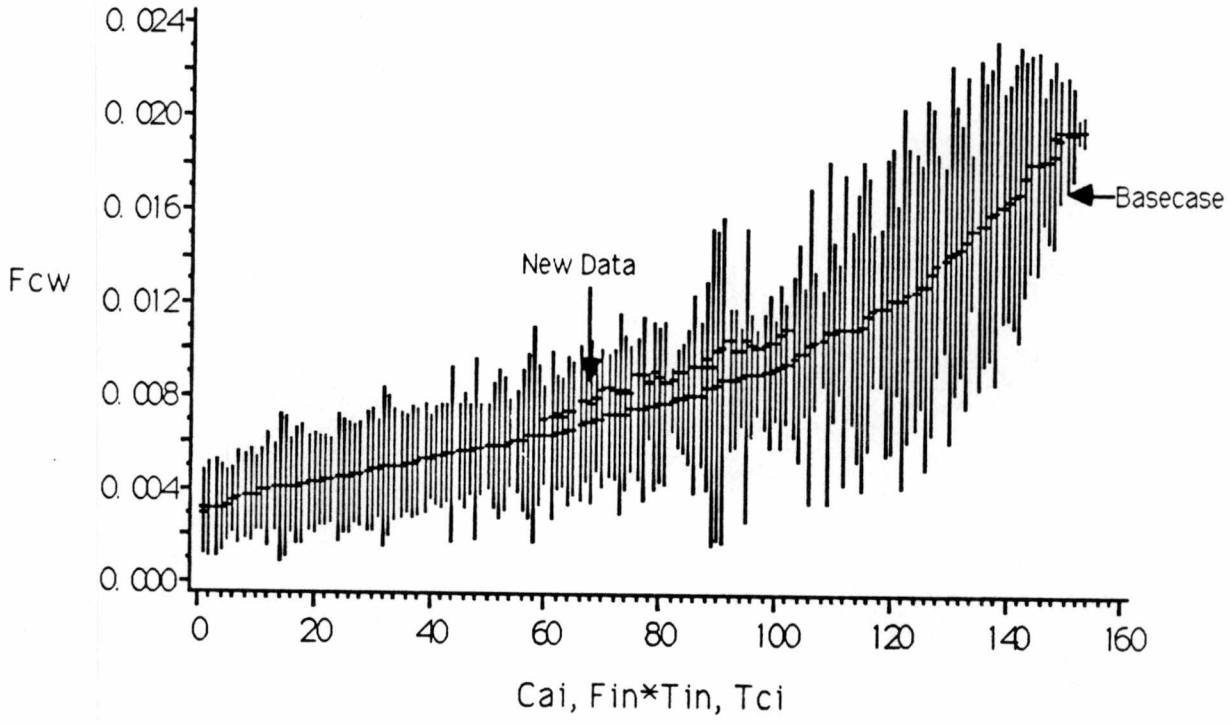


Figure 7.37: Multivariable statistical plot of F_{cw} for data under analysis

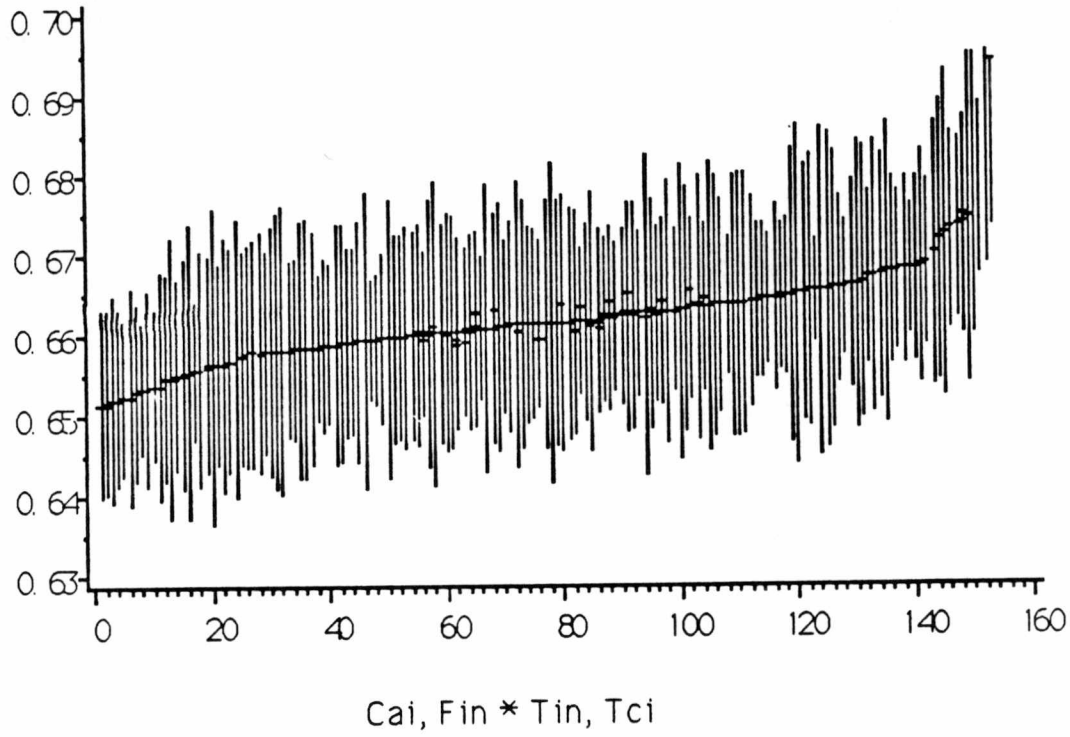


Figure 7.38: Multivariable statistical plot of $F_r * T_r$

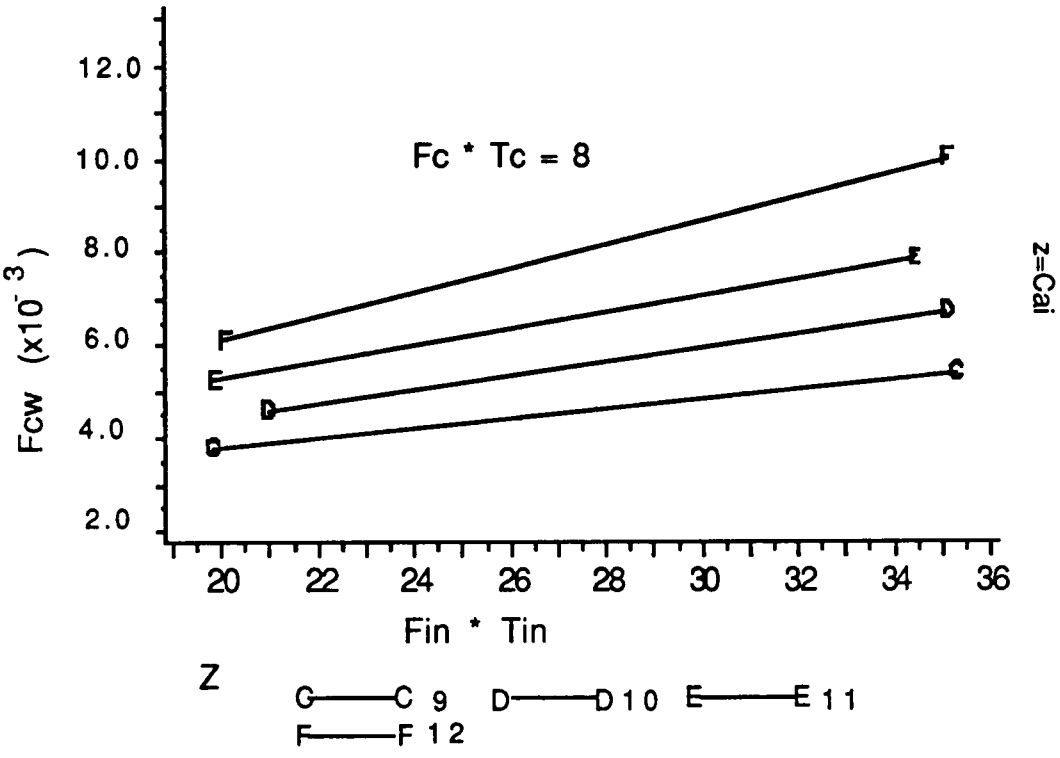


Figure 7.39: Relationship plot

tical plots and relationship plots monitored on this CSTR. The data analyzed resulted from a simulation in which the heat transfer coefficient was reduced by 7%. Similar simulations and analyses were performed for the remaining system problems (reduced catalyst activity, significant side reaction, change in C_p of the reactant, change in C_p of the chill water, poorly tuned controller). Figure 7.40 summarizes the HF analysis on those data sets in terms of reporters: separation, fractional overlap, and overall average slope. The figure also identifies the magnitude of the problem in each case.

7.2.3 Outline of problem recognition framework

This section describes the refined framework established for automatically interpreting reduced databases that result from HF-modeling; later sections detail each piece of this framework applied to the CSTR. A second framework for interpreting TSD analysis is established in Section 7.2.

This recognition framework classifies system problems accurately and defines problem magnitude reliably while accounting for new system problems, multiple system problems, and inseparable system problems. Figure 7.41 depicts the problem recognition flow diagram. The analysis comprises scaling, a distance-based threshold, a neural network, and distance-based verification. The following paragraphs outline a typical analysis of a large data set (24 hours or more). This explanation assumes the neural network is already trained.

The initial step scales the reduced database that results from HF-modeling via basecase-directional scaling. Section 6.2 outlines this scaling method. For

	Separation						Average Slope
	Fcw	Tr	Tcout	Fr x Tr	Fcw x ΔTcw	Caout	$\frac{\partial Fcw}{\partial Fr*Tr}$
ΔC_{pfeed} (+10%)	-0.247	-0.026	0.280	-0.014	-0.150	-0.0043	9.0e-2
ΔC_{pcw} (-17%)	0.261	0.040	0.025	0.025	0.536	-0.0083	9.2e-2
ΔUA_{inner} (-10%)	0.225	0.082	-0.354	0.055	-0.042	-0.0195	8.0e-2
Cat. Act (-10%)	-0.322	-0.031	0.359	-0.016	-0.184	1.480	8.6e-2
Contaminent (0.12)	0.321	0.078	-0.336	0.052	0.187	-0.018	8.8e-2
Controller	-0.128	-0.020	0.151	-0.011	-0.075	0.711	9.3e-2

Figure 7.40: Summary of HF-modeling for all system problems

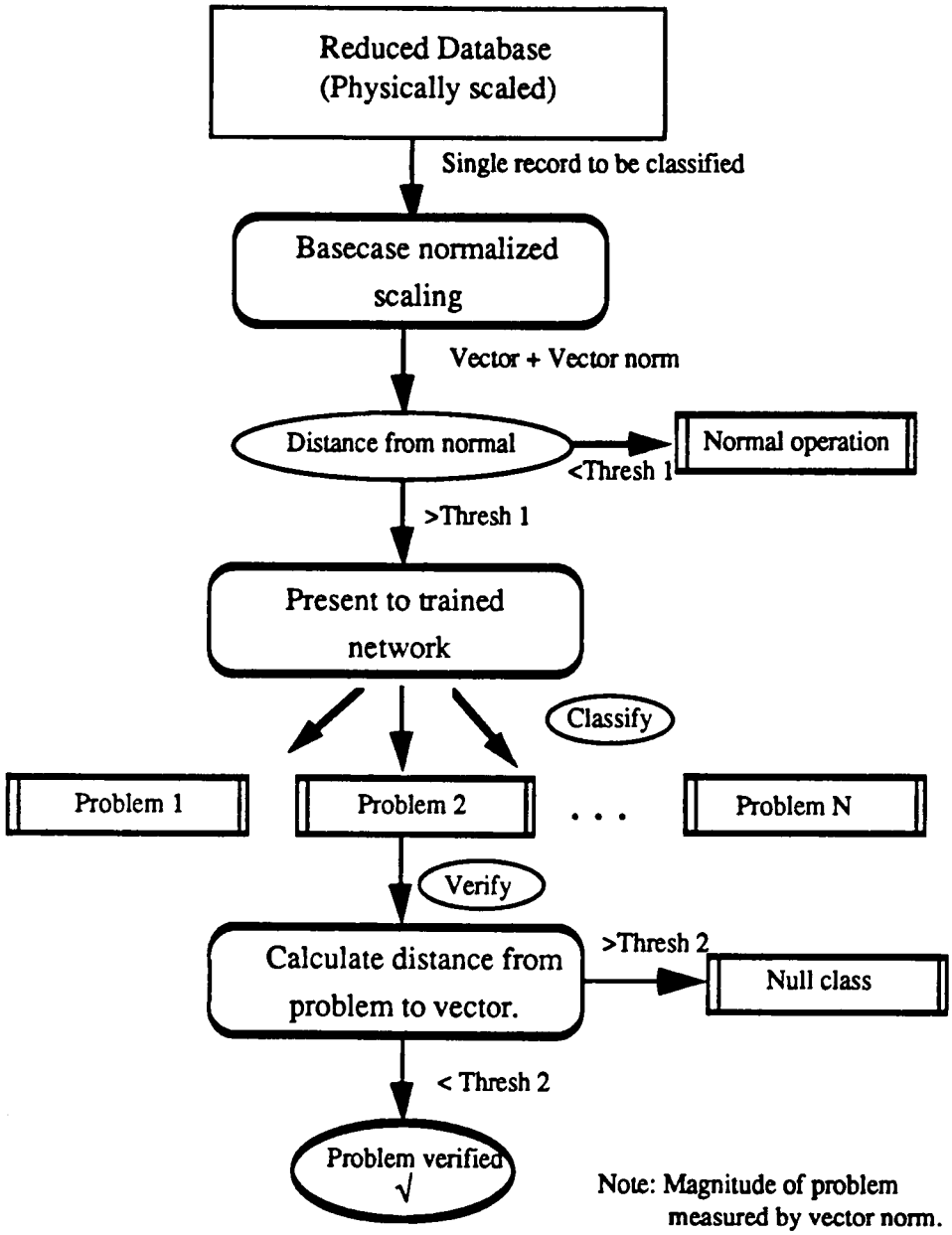


Figure 7.41: Problem recognition framework

each record in the reduced database, this scaling technique calculates a unit direction vector and distance (vector norm) relative to basecase operation. The remaining classification analysis focuses on these two pieces of information.

A simple distance-based classifier determines whether a vector norm is normal (represents basecase operation) or abnormal (represents a process change) by comparing the vector norm to a predetermined threshold value. For the CSTR, this threshold was established at 0.11. Below this value, a process problem is indistinguishable from basecase operation. This particular threshold is by no means a universal value but is process dependent. Establishing a threshold is nontrivial for processes in which a good simulation is not available. It is analogous to the threshold problem encountered in many classical clustering techniques. Techniques for establishing this threshold are discussed later in this section.

If a vector norm is greater than the threshold, its corresponding directional vector is submitted to the trained neural network. The network classifies this vector into one or more sets of predetermined problem classes. Notice that normal operation is not included in the set of identifiable classes but is signified by a small vector norm. A back-propagation network is employed in this CSTR example.

Finally, the network classification must be verified. This distance-based verification searches for vectors unlike all of the previously-experienced system problems. It is important to understand the distance calculated in this verification step. This distance measures the difference between the unit direc-

tional vector for the data being analyzed and the directional vector representing the problem class identified by the neural network. These directional vectors, rather than vector norms, are compared to eliminate problem magnitude as a factor when establishing the problem identity. The distance employed here is a Euclidean distance but can also be an angular measure. If the distance is greater than a specified threshold (0.05 for the CSTR), the vector is classified into the null class. If below the threshold, the neural network's classification is verified. Like the previous threshold, the precise value is nontrivial to establish.

Once a system problem is identified, the vector norm is a reasonable measure of the problem magnitude. The example presented later in this section elaborates on problem magnitude identification.

The preceding discussion describes the analysis of a single record from a reduced database. This record summarizes statistics and relationships of data in a single operating condition. All other records from the database must be processed in the same manner. It should be noted that if a vector from the same operating condition representing basecase operation does not exist, that vector is omitted from the analysis. Classification of the database as a whole must be made in terms of the percentage of records classified into each possible class.

7.2.4 Scaling the reduced database

Scaling the reduced database is necessary for two reasons. First, the neural network requires inputs between 0 and 1. Second, proper scaling im-

proves the performance of the network, helping to capture essential characteristics of system problems from very limited examples. Section 6.2 outlines the scaling procedure applied in this example. The following paragraphs detail the scaling procedure through an example from the CSTR.

Figure 7.42 depicts a scaling example showing two reduced information databases from HF-modeling. The top one resulted from basecase operation, while the bottom one resulted from an unknown set of conditions. Scaling of the unknown data set consists of three steps. The first step searches for operating conditions common to both reduced data sets. Only ones common to both sets of data are further processed. Figure 7.42 shows a particular condition common to both reduced data sets. The second step subtracts each record from its basecase counterpart; the resulting record is shown in the figure. Notice that the address is unchanged. The third step normalizes this difference vector (not including the address label), resulting in a unit directional vector and a vector norm. The unit direction vector points towards the new set of data with basecase operation as a reference point. The vector norm is key to evaluating problem magnitude.

All operating conditions common to basecase and new data analysis are processed in this fashion. The resulting vectors represent directions relative to basecase operation based on a common operating condition. These directional vectors are presented individually to the neural network for classification.

It should be noted that physical scaling is separate from the above discussion. Raw historical data is physically scaled before HF- modeling reduces

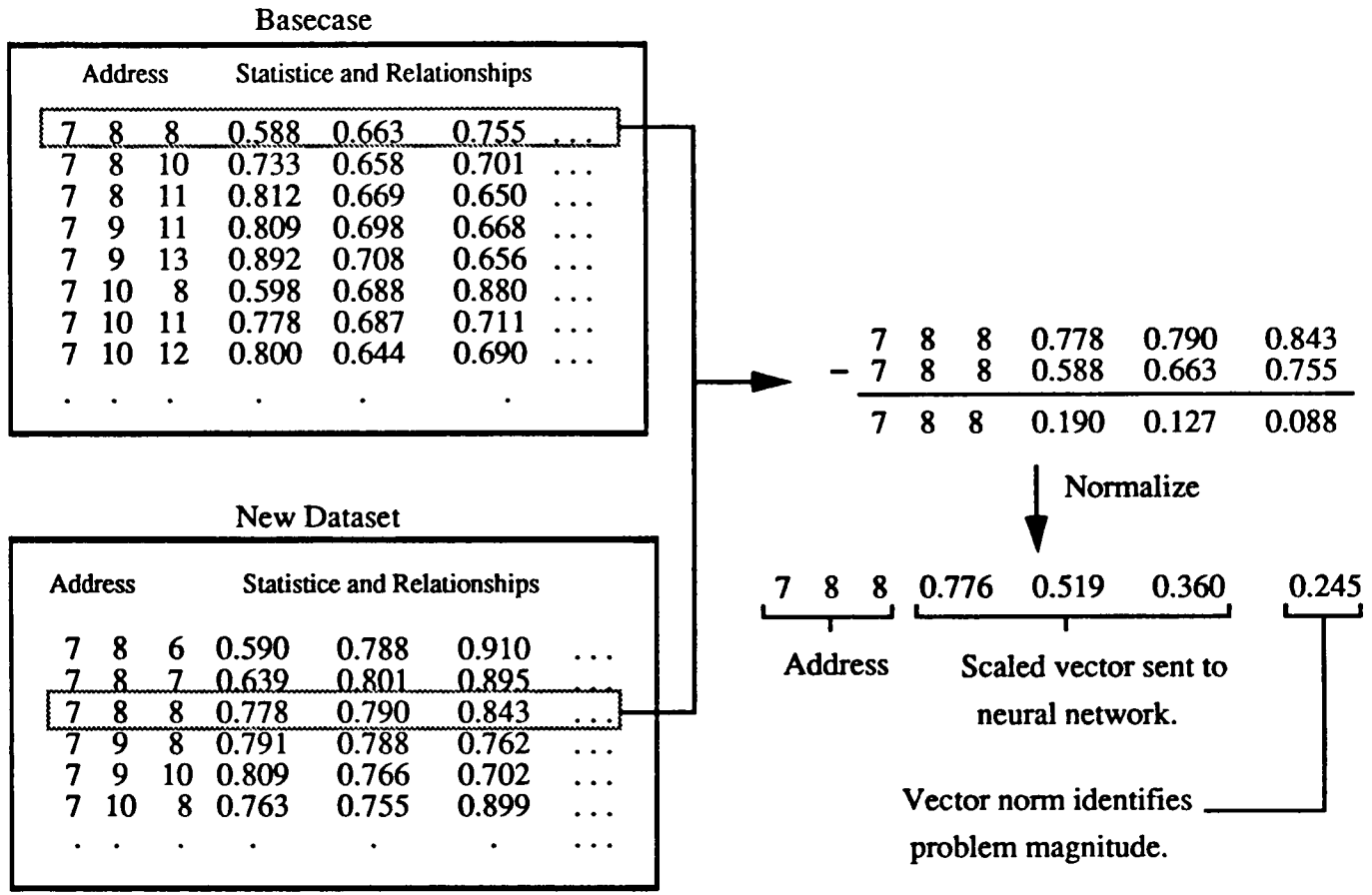


Figure 7.42: Scaling example

the database. It is an essential ingredient to the analysis but not represented in Figure 7.42.

7.2.5 Problem recognition framework applied to CSTR data

This section details each part of the problem recognition framework applied on CSTR data and studies associated with each. At the heart of this framework lies the neural network. Most of the work presented involves the network, evaluating how well the network classifies each potential system problem (after training), how well it classifies varying magnitudes of system problems, and how well it handles multiple problems. In addition, the neural network's performance is compared against a one-nearest prototype pattern classifier: a basic pattern classifier. The network performance as a function of input scaling is also investigated.

This section also elaborates on other pieces of the problem recognition framework: defining a threshold for normal operation and identifying problem magnitude, finding a threshold for detecting new system problems, and identifying indistinguishable system problems during network training.

7.2.5.1 The neural network

The back-propagation network applied in this investigation is outlined in Chapter 5. The network comprises 10 input buffers, 35 first-layer nodes, 35 second-layer nodes, and 6 output nodes. An input vector is a ten-element record from a scaled, reduced (through HF-modeling) database.

Output nodes have a one-to-one correspondence with the list of system problems. When submitted to the input buffer, an input vector's signal propagates through the network towards the output layer. For each input vector, one output node will have a high output (above 0.80). The node(s) whose output is high corresponds to the classification of the input vector. Although generally not the case, an input may be identified with more than one class.

The following sections discuss network training, classification, and other neural network studies completed on the CSTR.

7.2.5.2 Training the network

Reduced data sets representing 30 simulation hours from each process problem trained the multilayer perceptron network. Section 3.2 describes these reduced data sets. Figure 7.40 identifies the problem magnitude represented by these training sets.

Neural network training parameters are summarized below.

Gain = 0.45

Number of classes = 5

Threshold = 0.0

Number of times through = 20

Number of segments = 8

Vectors per segment = 23

Variation for correct classification = 0.20

Percent correct classification for convergence = 96%

The network had little difficulty converging on the training data and was able to classify at least 96% of the data in each training set correctly (minimum requirement for convergence). The training time was reduced almost one order of magnitude from those experienced on the heat exchanger: primarily because of proper scaling (basecase normalized scaling).

7.2.5.3 Network performance studies

This section describes several studies that evaluate the neural network's performance in classifying simulated data from the CSTR.

Classifying new data with the same magnitude of system problem

For each process problem, the trained network classified 24 hours of simulated data. Problem magnitudes represented in these data sets were identical to the those represented in training data. However, none of the data was contained in the training data (which explains the term "new" data).

Figure 7.43 summarizes the results of classifying the new data. The figure gives the percentage of records from the reduced data sets classified into each system problem. The average distance from basecase (the vector norm) is also given. Its significance is discussed in the next section.

For each of the system problems, the network classifies reliably. Note that classification of basecase data is distributed among each of the system

Class of Problem	Magnitude	% Classified 1	2	3	4	5	null	Distance from Basecase
1	+10%	**96**	0	0	2	2	0	0.240
2	+17%	0	**99**	0	0	1	0	0.250
3	-10%	2	0	**84**	0	8	6	0.231
4	-10%	0	0	0	**99**	1	0	0.405
5	0.12	0	0	1	0	**99**	0	0.250
6	--	37	13	8	6	26	10	0.108

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Network

Input layer: 10

1st hidden layer: 35

2nd hidden layer: 35

Output layer: 6

Figure 7.43: Classification of 'new' data

problems. Remember that the network is not trained explicitly with basecase data, because of scaling. Identification of basecase is described in a later section.

Classifying varying magnitudes of system problems

In a real environment, system problems can vary in magnitude. This section summarizes the neural network's performance in classifying varying degrees of system problems. Keep in mind that the network is trained with only a limited range in magnitude of each system problem.

The network classified reduced data sets whose inherent problem magnitude was less than that of the training sets. In each case the problem magnitude was about half that represented in the training sets. Figure 7.44 summarizes those results. In all cases but one (a significant side reaction), the network performed well, classifying a significant majority of the reduced data into the correct class. In all cases, the correct class had the highest percentage of records.

The network's ability to classify problem magnitudes greater than those represented in the training sets was also studied. Those studies revealed that the network performs as well, if not better than when the problem magnitudes are equivalent to those in the training data.

Figure 7.45 summarizes classification results of selected process problems at several problem magnitudes. Even though scaling theoretically negates the effects of problem magnitude in network performance, the results show that

Class of Problem	Magnitude	% Classified 1	2	3	4	5	null	Distance from Basecase
1	+5%	**75**	2	2	6	8	7	0.127
2	+9%	0	**83**	0	0	11	6	0.190
3	-5%	5	0	**57**	1	27	10	0.162
4	-5%	5	8	0	**85**	1	1	0.156
5	0.06	26	6	13	5	**35**	15	0.134

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.44: Classification of system problems (about one-half training magnitude)

Class of Problem	Magnitude	% Classified 1	2	3	4	5	null	Distance from Basecase
1	+14%	**98**	0	0	2	0	0	0.277
1	+4%	**55**	5	5	5	16	14	0.129
1	+3%	**42**	5	5	7	28	13	0.116
2	+7%	2	**67**	0	4	18	9	0.150
2	+4.5%	5	**36**	1	0	45	13	0.122
3	-5%	5	0	**57**	1	27	10	0.162
3	-3%	19	2	**36**	0	33	10	0.132
4	-6%	10	1	0	**78**	9	2	0.172
4	-5%	5	0	0	**93**	1	1	0.156
4	-3%	22	4	0	**41**	14	19	0.138
5	0.06	26	6	13	5	**35**	15	0.134
5	0.03	33	6	7	5	**37**	12	0.122

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.45: Classification of problems (various magnitudes)

in reality large problems are more reliably classified. However, the network performance is still very good, even in detecting small process changes.

Classifying multiple system problems

In real-life system operation, more than one system problem may occur at a time. In this study, two system problems occurring simultaneously were considered. Each of these problems had the same magnitude as in the teacher data.

Figure 7.46 summarizes the classifying of multiple system problems. In nearly all cases, the network classified the highest percentage of data into the system problem with the most impact on the process. In some cases, like $\Delta C_{p_{feed}}$ and ΔU, classification was split evenly between the problems. In all tests summarized by the figure, one of the system problems present was represented by the majority of classified data. However, this does not always hold true; infrequently the network classifies the highest percentage of the data into a system problem not present. In the CSTR example, such a situation occurred only once in the eight situations studied.

7.2.5.4 A comparison of scaling techniques

To demonstrate the sensitivity of network performance to scaling, this section compares the neural network's performance in classifying basecase normalized data vs. normalized data. Each of these scaling methods is outlined in Section 6.2.

Class of Problem	Magnitude	% Classified 1	2	3	4	5	null	Distance from Basecase
1&2	Same	**2**	**88**	0	4	0	6	0.190
1&3	as	**54**	0	**39**	0	1	6	0.171
1&4	training	**1**	0	0	**99**	0	0	0.488
1&5	data.	**95**	0	0	0	**2**	3	0.198
2&3		0	**24**	0	0	**67**	9	0.370
2&4		0	**4**	0	**54**	0	42	0.309
3&5		0	0	**25**	0	**68**	7	0.480
4&5		0	0	0	**94**	**2**	4	0.289

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.46: Classifying multiple system problems

The same multilayer perceptron network applied in the previous studies was trained with HF-modeling information from each class. The procedure is exactly the same as in the previous section with one exception: the data is scaled by simple normalization. Figure 7.47 summarizes the network's ability to classify new data. The problem magnitude in the data classified is equal to that in the training data. The results compare favorably with those attained by basecase normalized scaling.

Figure 7.48 presents the results of classifying data whose problem magnitude is about half that of the training set. Classification in three of the five cases is unacceptable, and therefore these results are not nearly as satisfying with basecase normalized scaling.

This result brings out an inherent problem with neural networks. They perform well only in specific instances for which they are trained. They "generalize", but not always to the extent desired. Therefore, intelligent scaling is important. A system problem must always appear the same to the network, regardless of problem magnitude.

7.2.5.5 Neural network vs. one-nearest prototype classifier

This section tests the performance of the neural network against a one-nearest prototype (1-NP) pattern classifier. The 1-NP algorithm is a basic distance-based pattern classifier. In this algorithm, a single prototype represents each pattern class. Euclidean distance of a pattern vector to each of these prototypes is the basis of classification.

Class of Problem	Magnitude	% Classified 1	2	3	4	5	6	null
1	+10%	**95**	0	0	0	0	5	0
2	+17%	0	**99**	0	0	1	0	0
3	-10%	0	0	**82**	0	14	0	4
4	-10%	0	0	0	**100**	0	0	0
5	0.12	0	0	1	0	**99**	0	0
6	--	0	0	2	0	0	**91**	7

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.47: Classification with normalized scaling (same mag as training data)

Class of Problem	Magnitude	% Classified 1	2	3	4	5	6	null
1	+5%	**52**	0	0	0	0	41	7
2	+9%	0	**12**	0	0	27	15	46
3	-5%	1	0	**37**	0	2	39	21
4	-5%	60	0	0	**40**	0	0	0
5	0.06	0	0	16	0	**30**	28	26

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.48: Classification with normalized scaling (one-half mag of training data)

The 1-NP classifier can now be applied to HF-modeling information. Because only common addresses (operating conditions) can be compared meaningfully, the prototype from a problem will be a function of the condition under consideration. For example, consider the classification of a reduced database with the 1-NP algorithm. The first three elements of this record specify the prototypes for each problem class. A new address will result in new prototypes for each problem class. The result is that only common operating conditions are compared.

This investigation compared the ability of neural networks and the 1-NP algorithm to classify data that reflected the same magnitude of system problem as the training data and also data reflecting system problems of half that magnitude.

Figure 7.49 tabulates results of the 1-NP classifier for such data from the CSTR. For each system problem of the same magnitude as the training data, the 1-NP classified the data set correctly. A second study compared the 1-NP algorithm with the neural network to classify data sets whose inherent problem magnitude was about half that in the teacher sets. Figure 7.49 summarizes the 1-NP classification of this data. In two of five cases, the 1-NP algorithm classified the data set incorrectly. In comparison, the neural network classified correctly in all cases.

In summary, when problem magnitude is large, the 1-NP algorithm identifies system problems accurately; however, the accuracy decreases on smaller problem magnitudes. Because they assign a weighting to each feature in the

Class of Problem	Magnitude	% Classified 1	2	3	4	5	6
1	+10%	**89**	0	0	0	0	11
1	+5%	**63**	0	0	0	0	37
2	+17%	0	**85**	0	0	3	12
2	+9%	5	**23**	2	0	14	56
3	-10%	3	0	**74**	0	13	10
3	-5%	10	0	**31**	0	7	52
4	-10%	0	0	0	**100**	0	0
4	-5%	42	0	0	**26**	2	30
5	0.12	7	4	4	19	**66**	0
5	0.06	0	4	22	0	**41**	33
6	--	13	0	17	0	0	**70**

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.49: 1-NP classification

pattern space, neural networks outperform the 1-NP algorithm.

7.2.5.6 Classifying normal operation and identifying problem magnitude

This section describes important pieces of the problem recognition framework that solve problems the neural network cannot. The techniques for classifying normal operation, for identifying problem magnitudes, and for classifying the null class are demonstrated through the CSTR example.

Identifying problem magnitude

The scaled vector norm contains information helpful for identifying problem magnitude. In general, the larger the norm, the more significant the problem. The exact relationship between them is a function of the process and the process problem. Here, three case studies are examined. The relationship between problem magnitude and an average vector norm are studied on three system problems: $\Delta C_{p_{feed}}$, $\Delta C_{p_{chill}}$, and reduced catalyst activity.

Figure 7.45 summarizes the classification of varying magnitudes of the three system problems along with an overall average vector norm. Because the problem magnitude is known during simulation of the CSTR, relationships between average vector norm and problem magnitude follow directly. Figures 7.50, 7.51, and 7.52 plot those relationships.

Figures 7.50 and 7.51 depict a linear relationship between % change in catalyst activity and the average distance from basecase (average vector

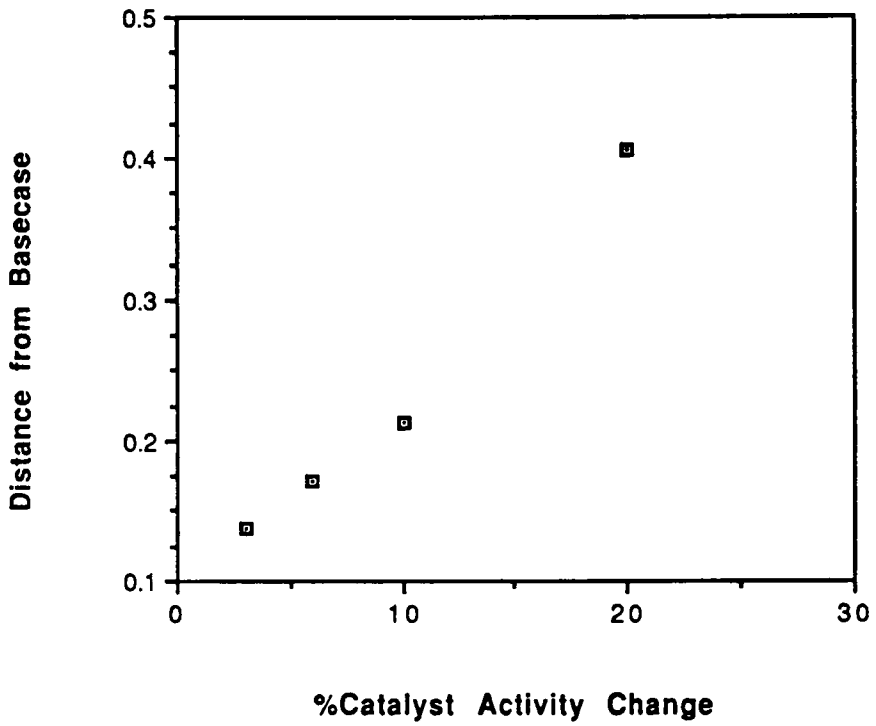


Figure 7.50: Problem magnitude vs dist from basecase for change in catalyst activity

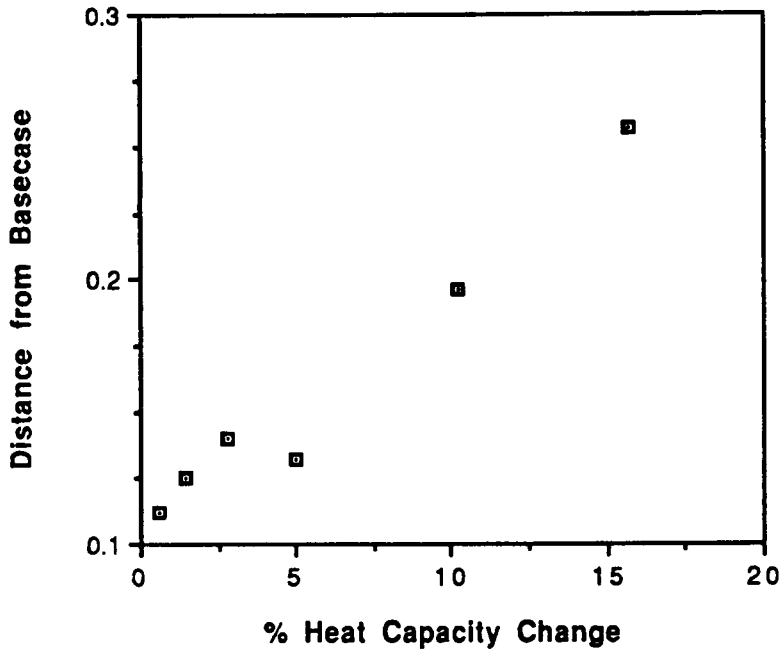


Figure 7.51: Problem magnitude vs distance from basecase for $\Delta C_{p_{chill}}$

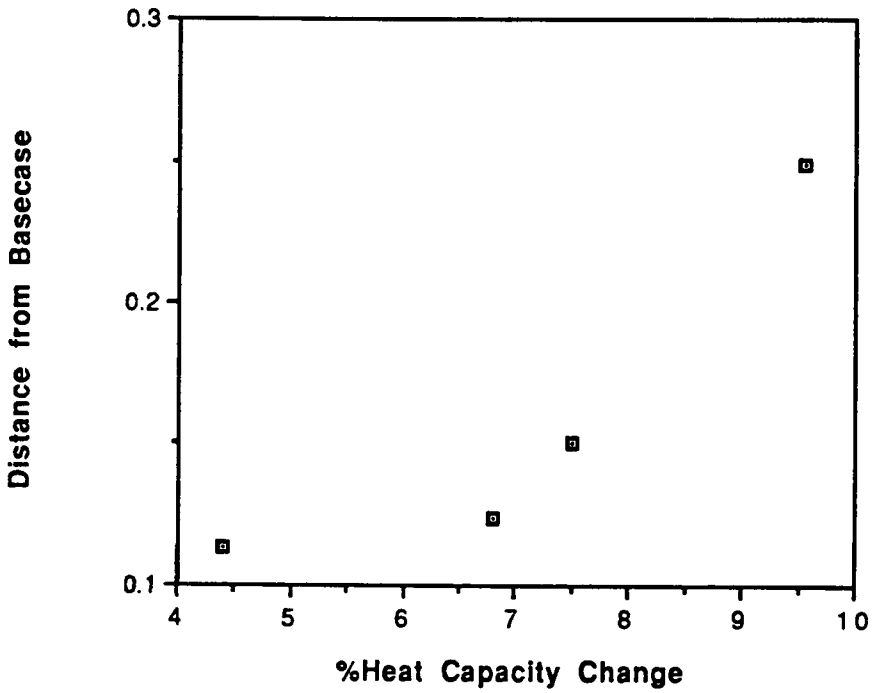


Figure 7.52: Problem magnitude vs distance from basecase for $\Delta C_{p_{feed}}$

norm); and % change in chill water heat capacity and distance from basecase. Of course, such relationships are not always linear as demonstrated by the relationship between % change in feed stream heat capacity and distance from basecase. It should be noted that these figures include no error analysis; they are simply points plotted from available data.

Basecase classification

Basecase normalized scaling compares data relative to basecase operation. Therefore, training the neural network with basecase data scaled in this manner does not make sense. Hence, basecase operation is not identified by the neural network. Normal operation must be identified by low problem magnitude. In practice, the network will classify all data as a process problem. However, the associated problem magnitude may be insignificantly small or below some threshold of reliability. This section seeks to identify that threshold.

One method seeks the smallest problem magnitude classified reliably. The average vector of such a problem magnitude is one appropriate threshold. For example, a change of 4 percent was found to be the smallest detectable change in chill water heat capacity. The resulting average vector norm for data representing this change was 0.113. A threshold in the neighborhood of 0.120 would be appropriate. This particular method is simulation based, a major disadvantage in many cases.

The average vector norm between two sets of basecase data is another criterion for defining the threshold. Normal process variation accounts for a non-zero vector norm. Process problems cannot be discerned below this threshold. For example, on the CSTR, one such average vector norm between two sets of basecase data was 0.11. This value can be used to gauge a good threshold. Generally, the threshold should be 10 to 15 percent larger than this norm.

Both methods outlined for arriving at a good threshold have their merits. The first method applies when a good simulation is available, the second incorporates process data. When possible, both should be tried. The larger of the two thresholds should be implemented in the framework. On the CSTR, a threshold of 0.12 was implemented.

Identifying new system problems

Like classifying normal operation, identifying new system problems involves a threshold determination. Unlike classifying normal operation, the threshold is not related to the average vector norm but to an average difference between unit direction vectors. Imagine unit direction vectors (a product of basecase directional scaling) representing points on a hypersphere. A cluster of points on that hypersphere represents each problem class. It is likely that the neural network will classify a new distinct cluster of datapoints (representing a new system problem) as a "nearby" system problem. Overriding this classification requires finding a distance threshold between data clusters. If the

distance between two clusters is above this threshold, a new system problem is indicated (classified as null). In this example, the distance measure chosen is the average of the distances used in the 1-NP classifier, that is, a Euclidean distance between points on the hypersphere.

As in the last section, identifying this threshold is not a trivial matter. For the CSTR, distances between all training sets were calculated to attain a good guess of a logical threshold. Figure 7.53 summarizes those distances. The closest two problem clusters are a significant side reaction and a reduced heat transfer coefficient (distance = 0.46). The threshold must be less than this distance, but not significantly less.

A better threshold value can be found if a second set of data representing one of the known process problems is available (regardless of problem magnitude). For example, distances between three different data sets representing a significant side reaction were 0.212, 0.394, and 0.339. The threshold value must be slightly greater than the largest of these distances (0.394).

The above methods give

$$0.39 < \text{Threshold} < 0.46.$$

The threshold implemented was 0.40.

7.2.5.7 Indistinguishable system problems

Indistinguishable system problems are two or more problems that affect the system in exactly the same way. No technique can reliably differentiate

Problem Class A	Problem Class B	Distance between Class A & Class B
1	2	1.80
1	3	1.86
1	4	0.97
1	5	1.90
2	3	1.14
2	4	1.62
2	5	0.78
3	4	1.65
3	5	0.46
4	5	1.68

Class 1: ΔC_{pfeed}

Class 2: ΔC_{pcw}

Class 3: ΔUA_{inner}

Class 4: Catalyst Activity

Class 5: Undesired Side Reaction (contaminent)

Class 6: Basecase

Figure 7.53: Average distances between training data sets

between such problems. It is extremely important for the framework to recognize that all system problem can not be differentiated perfectly from all others. Some data sets may converge when training, but poor performance is achieved when applied to other data. Failure to take this into consideration will result in poor performance and a loss of confidence by the unknowing user.

The inability to differentiate between system problems is of course related to the sensor data available from a process. The framework presented attempts to do the best it can given the sensor data available. In the event that it can not differentiate between two or more system problems, it will tell us during training. Those problems will then be lumped as one class.

Indistinguishable problems can be identified in two ways. One is simply to calculate all distances between all system problems (as described in the previous section). If two system problems are indistinguishable, they will be separated by small distances; precisely how small is often difficult to determine.

A second, more reliable, way of identifying these inseparable system problems is related to the training of the neural network. For such problems, if properly scaled, the network will simply not converge. In fact, it was found that the network will attain a certain point on the learning curve and get no higher. For example, only 50% of the data in a training set may be classified rather than the desired 90-100%. The network can try forever but can learn no more. The training algorithm should be smart enough to know when improvement in learning has peaked. At that point, learning should stop. It was found that upon classification of the training data at this point, it was obvious which

problems were separable.

It is important to note that a network with an enormous number of hidden-layer nodes may, in fact, allow the network to converge satisfactorily on data that is really indistinguishable. The large number of hidden-layer nodes allows the network to draw decision boundaries with many very sharp corners. Intuitively, such boundaries make no sense; They attempt to differentiate between small, insignificant differences. Enough nodes and connections must be present to store all relevant information, but it is undesirable as a pattern classifier for a network to be massively parallel (on the order of thousands of hidden-layer elements).

7.2.6 TSD analysis of the CSTR

This section details the TSD analysis of the CSTR. In this reactor system, a single PI controller manipulates chill water flow to keep the reaction temperature at setpoint. The TSD analysis of this controller identifies over-compensation, under-compensation, or good performance. It simply raises a flag when a controller needs attention.

The closed-loop response to a step in the disturbance is compared with the TSD plot for several sets of tuning parameters. This comparison demonstrates TSD plot characteristics of a well-tuned controller, an underdamped controller, and an overdamped controller on the CSTR.

7.2.7 Summary of TSD results

This study examines TSD plots of the PI temperature control loop on the CSTR. These plots reflect several sets of tuning parameters, some resulting in good performance, some resulting in overcompensation, and some resulting in undercompensation. The closed-loop response to a step change in the reactant flow is compared with the TSD analysis for each set of tuning parameters.

Figure 7.54 depicts a step response and its TSD counterpart for a well-tuned controller (Kc=12.0, τ_I=800). Note that the step response covers nearly five hours of simulation time. During this period, no other disturbances enter the system. Also, noise has been removed from the output for ease in evaluating controller performance. The TSD plot, however, represents 24 hours of system operation. During that time any disturbance or set point change may enter the system. Noise is also included in the data analyzed.

The TSD plot for this well-tuned PI controller is similar to one for good dynamic matrix control studied by Hackney[25]. The plot returns to the 50% line in relatively few time shifts with a few oscillations. On this particular TSD plot, there is no significance to any information beyond a time shift of about 40 because this is beyond dynamic influence.

The following studies examine TSD plots for various controller settings. Controller performance is based on the step response of the system. The results given here are not exhaustive but exemplify typical TSD plots. Presented with each TSD plot are the corresponding disturbance response (without noise)

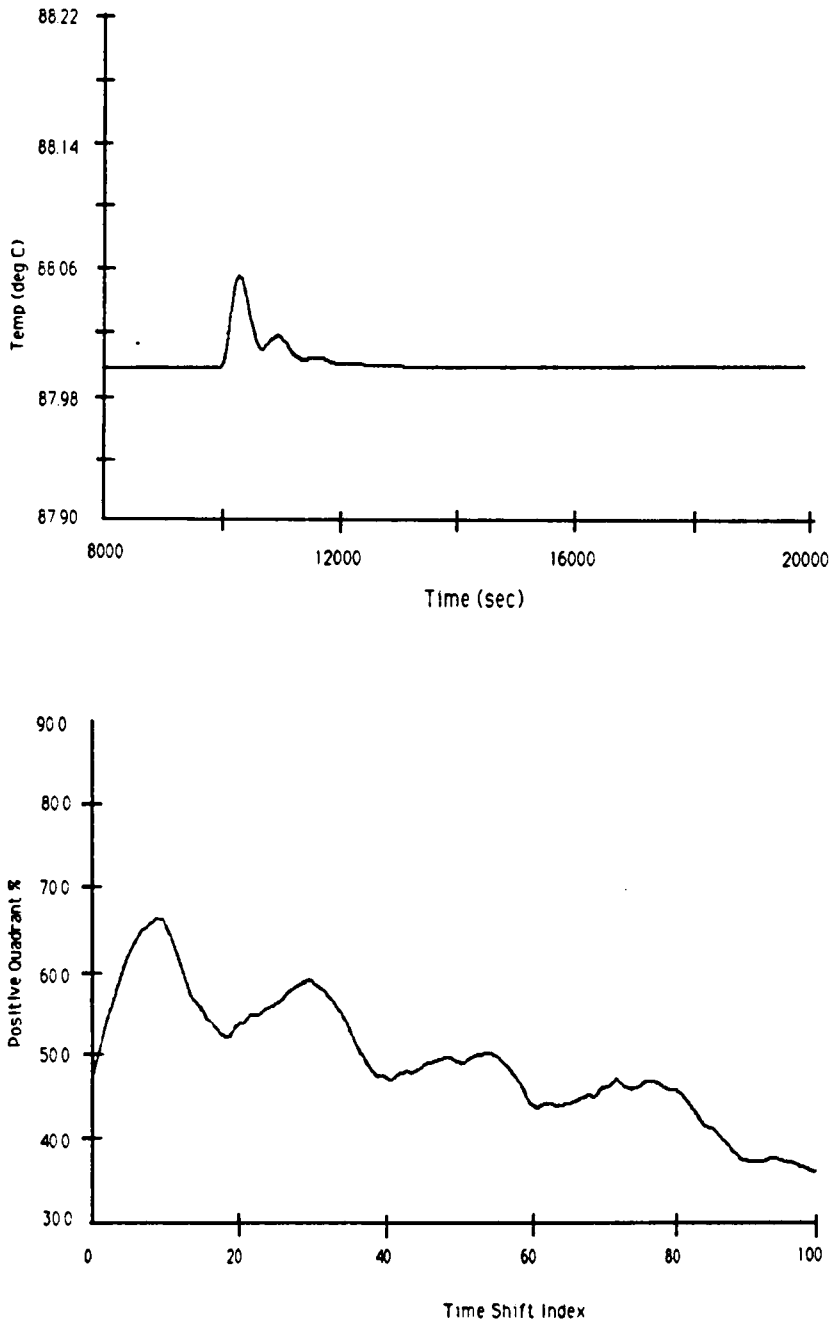


Figure 7.54: Response of a well-tuned controller to a step in reactant flow rate along with TSD plot: K_c=12.0, τ_I=800

profiles.

Figure 7.55 shows the TSD plot of a controller where Kc has been reduced (compared to Figure 7.54) from 12.0 to 5.0 with τ_I remaining the same (800 seconds). The time series plot demonstrates that although the response is more sluggish, it is probably still acceptable. This TSD plot is similar to Figure 7.54, with two notable exceptions: the 50% crossover point has stretched to a shift index of 42, and the plot has no obvious oscillations.

In Figure 7.56, the controller gain was reduced even further to 1.0. The controller performance is now even more sluggish and probably unacceptable. Again, the main differences between this TSD plot and one for good control lie in the 50% crossover point (70 vs 37) and the maximum value of positive quadrant percentage (85 vs 67).

Figure 7.57 depicts a typical TSD plot for an overcompensating controller. The tuning parameters were K_c=5.0 and τ_I=200. Oscillations about the 50% line reflect overcompensation. Such a conclusion was also reached by Hackney[25] in the analysis of a DMC control algorithm.

Figure 7.58 shows a TSD plot for a controller that displays approximately quarter-decay response. Although this response may not be as desirable as Figure 7.54, it is acceptable for many systems. This TSD plot illustrates three characteristics that summarize the findings of this TSD study. First, the maximum positive quadrant percentage is a function of the maximum initial error between setpoint and temperature. In the quarter decay system, the maximum percentage was over 80%, indicating a relatively large initial

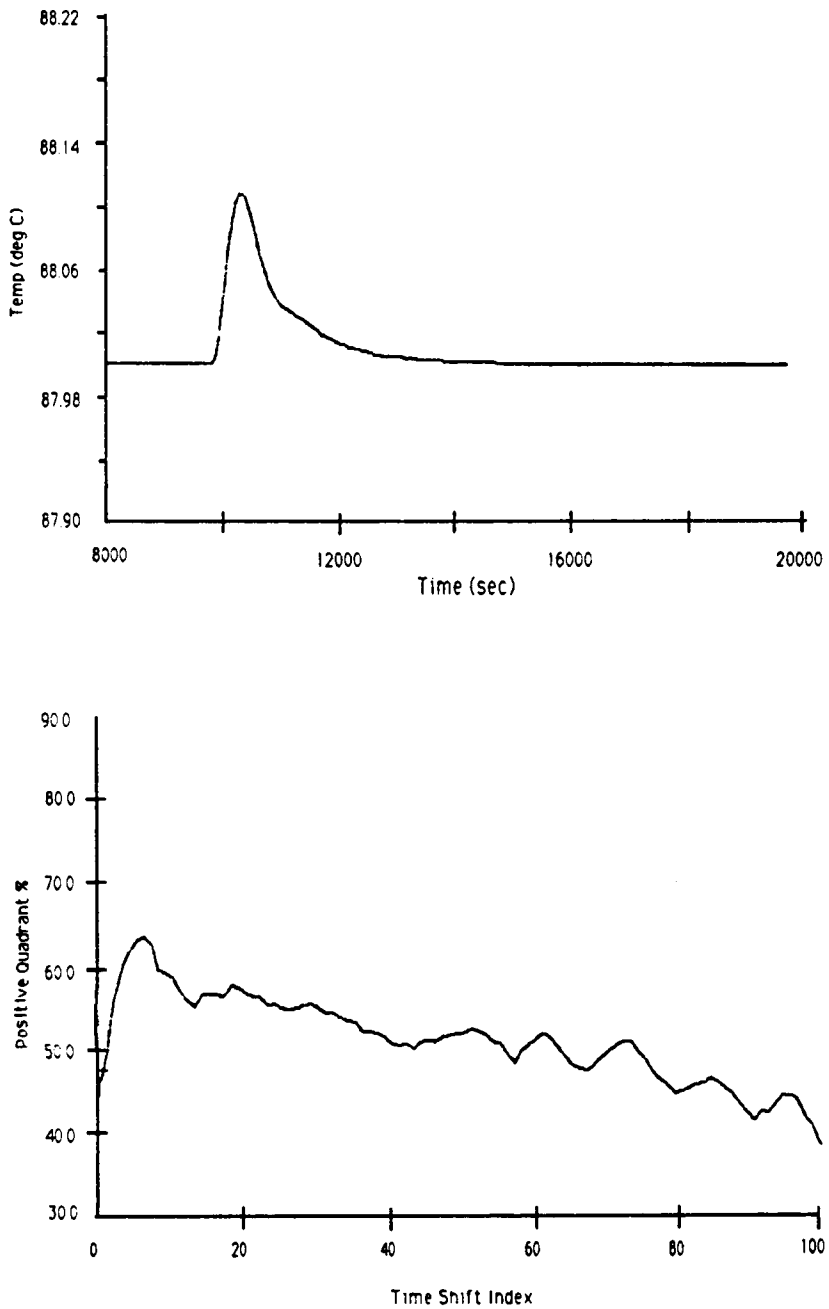


Figure 7.55: Step response along with TSD plot; $Kc = 5.0$, $\tau_I = 800$

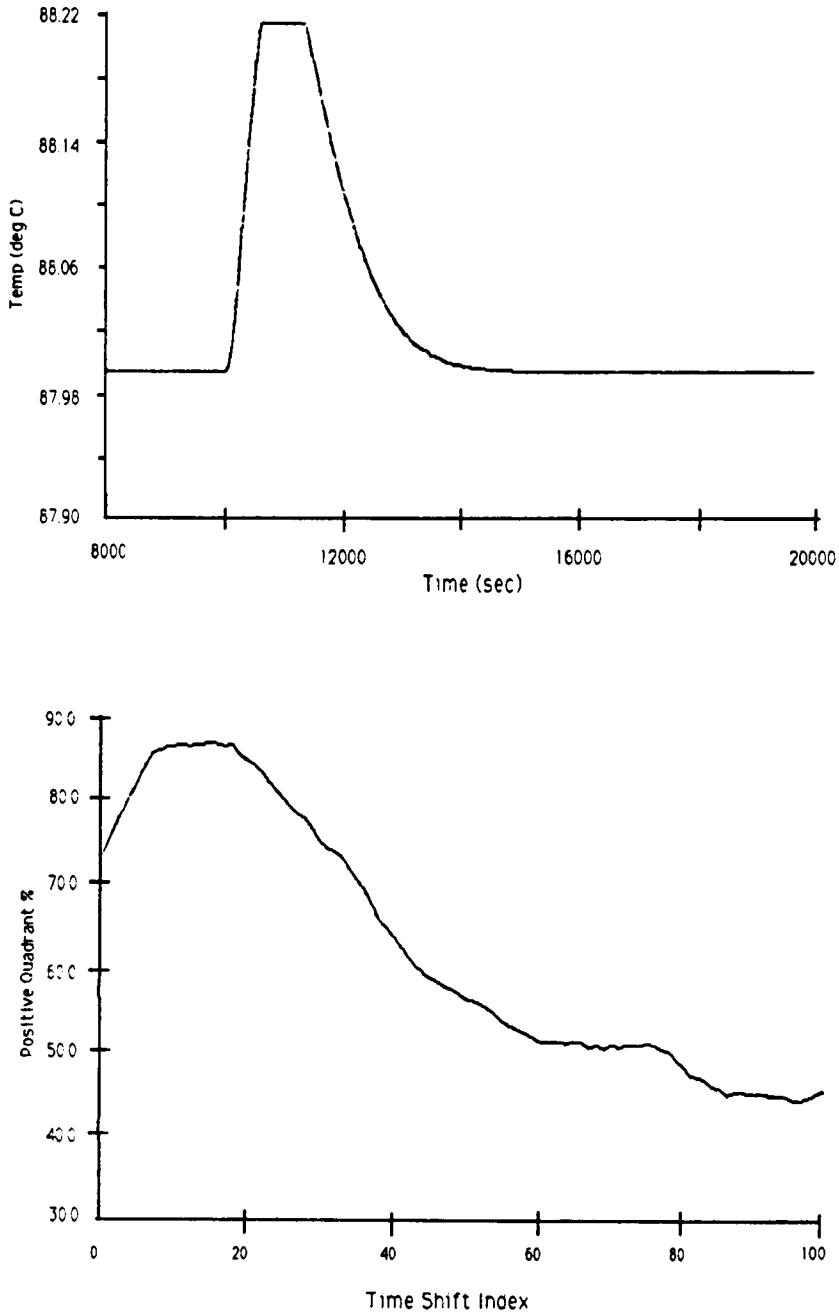


Figure 7.56: Step response along with TSD plot; $Kc = 1.0$, $\tau_I = 800$

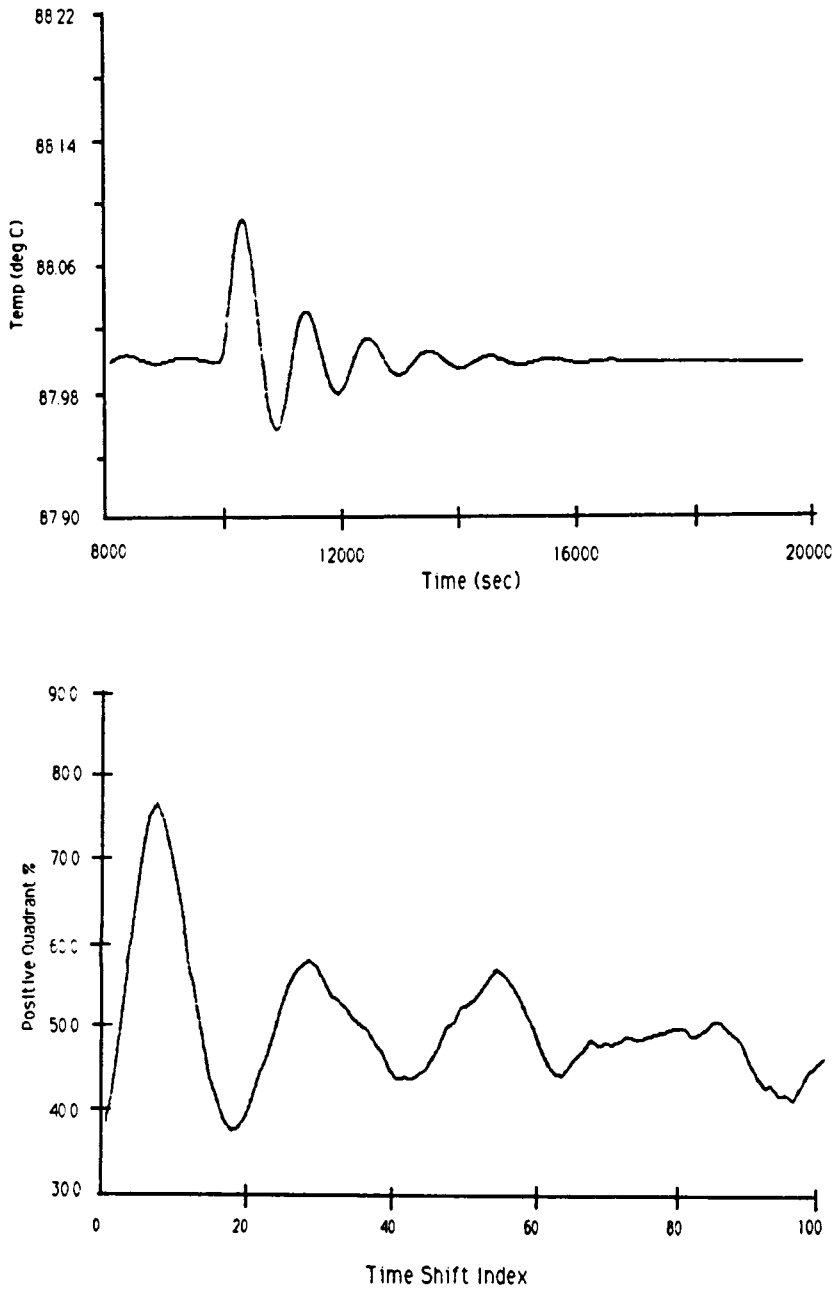


Figure 7.57: Step response along with TSD plot; $Kc = 5.0, \tau_I = 200$

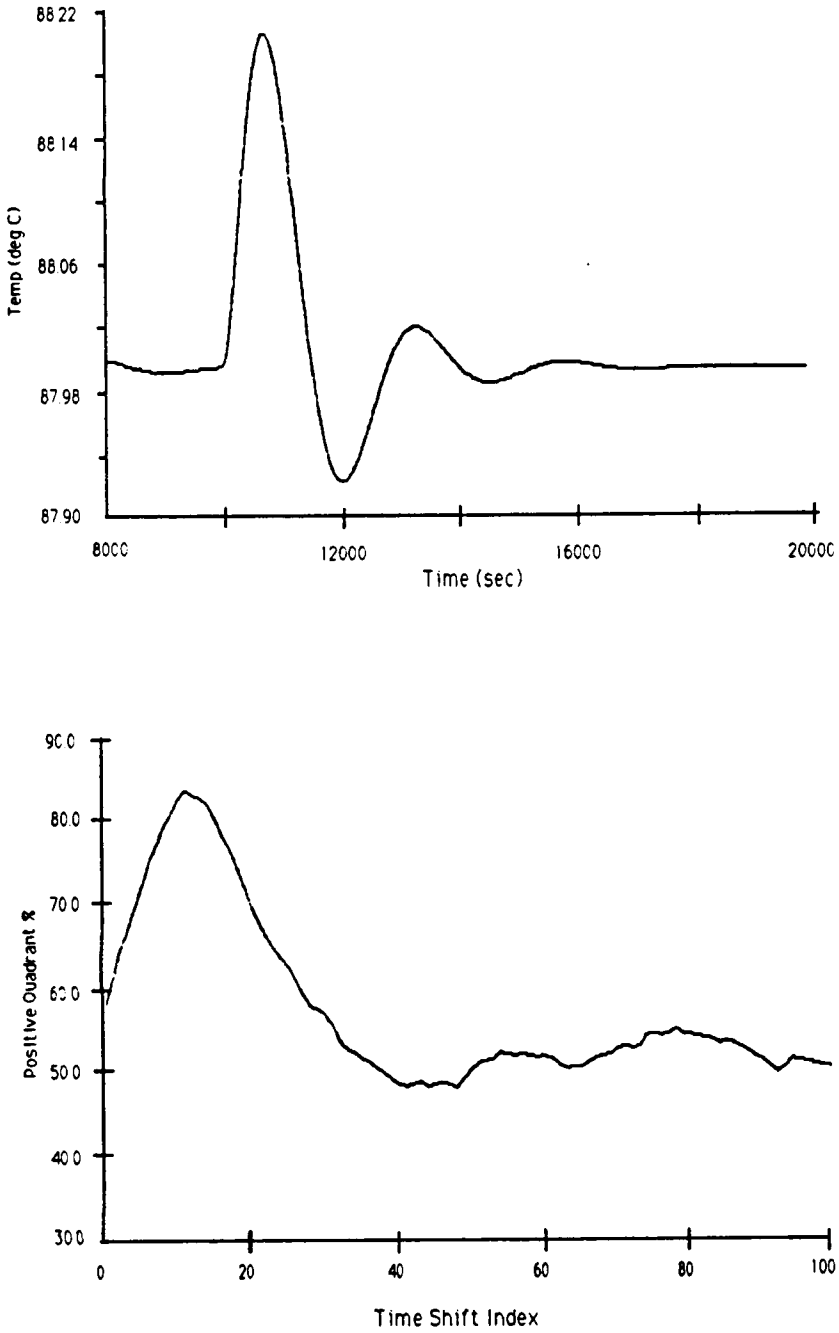


Figure 7.58: Step response along with TSD plot; $Kc = 1.0, \tau_I = 200$

error compared to that of Figure 7.54. Second, the first 50% crossover indicates the relative speed of response. In Figure 7.54, the first crossover was at near a shift index of 37, indicating a relatively quick response compared to Figure 7.56. Third, oscillations around the 50% line signify an underdamped system response (overcompensation).

7.2.8 Neural network interpretation of TSD plots

This section discusses the interpretation of TSD plots from the reactor system. Like the problem recognition framework detailed in Section 7.2.5, the neural network is the workhorse for identifying controller performance. The following sections describe how the TSD information is presented to the network, how the network is trained, and how well the trained network performs.

7.2.8.1 Presenting TSD information to the neural network and scaling

Presenting the TSD plot effectively to the network is key to good performance. In this example, a simple method effective in modeling processes and interpreting spectral data is applied. This technique presents TSD data at regular intervals along the shift index axis to the network. In this example, every third time shift out to 48 time shifts comprises a pattern vector presented to the network.

So that each element lies between zero and one, the pattern vector must be scaled before presented to the neural network. As past sections have em-

phasized, the scaling employed is key to effective network performance. Scaling via basecase normalized scaling makes sense; however, it is not possible for two reasons. First, in many instances, subtraction from basecase patterns leads to negative elements. Second, because a variety of different profiles represent acceptable controller performance, no one standard basecase pattern exists.

The scaling employed in this example was to simply present the fraction in the positive quadrant directly, as in the TSD plot (e.g., 80% = 0.80). Such a scaling technique leads to acceptable performance.

7.2.8.2 Neural network training

A multilayer perceptron network was trained to interpret the TSD plots. This network comprised 17 input buffers, two 18-element hidden-layer nodes, and three output-layer nodes. The large input buffer accommodated data at every third time shift out to 48 time shifts. Each of the three output nodes corresponds to one of three possible classes- well-tuned, overcompensating and undercompensating.

Relative to HF-modeling training sets, the size of TSD training is small. At most, three pattern vectors represented each class (overcompensating, undercompensating, and well-tuned). Accordingly, network training converged in a short time. However, small training sets places a premium on ensuring that the few vectors in each training set fully represent its class. For example, a well-tuned controller can be slightly underdamped or slightly overdamped. These two different responses must be represented during training for effective

performance. Figures 7.54, 7.55, and 7.58, represent the well-tuned class in this example.

The precise boundary between acceptable and unacceptable controller performance is fuzzy and varies from system to system. The user must decide what constitutes good controller tuning. For this reason, vectors representing the minimum unacceptable undercompensating and minimum unacceptable overcompensating responses must be included in the training sets. If not included, the neural network will draw the boundary wherever it pleases and with no process insight. Figure 7.57 depicts the minimum unacceptable overcompensating control, and Figure 7.56 shows the minimum undercompensating control employed in this example.

Note that the response of poorly tuned controllers to a disturbance looks the same for almost all systems. In addition, a comparison of the work here with that of Hackney[25] indicates that this may also be true of TSD analysis. However, more studies are needed to confirm this statement. Therefore, it may not be necessary to train the TSD plots directly from the specific controller. It may be possible to train a network via simulated data and apply it directly on a controller.

7.2.8.3 Neural network performance

The multilayer perceptron network trained with the information described in the previous section, was asked to classify performance from many controller settings on the CSTR. In general, the TSD-neural network com-

bination's performance was excellent throughout extensive tests on a range of tuning parameters. A few examples of those tests are presented by Figures 7.59 and 7.60. The figure identifies the tuning parameters, K_c and τ_I, presents a step response (with noise and other disturbances suppressed) and corresponding TSD plot (that includes noise and other disturbances), and identifies the controller's performance.

The results described above were not obtained without identifying a few essential factors. First, the network should comprise a relatively small number of hidden-layer nodes, otherwise it will function as a lookup table rather than keying on relevent attributes. Second, the training examples depicting the three classes, overcompensation, undercompensation, and basecase, should represent borderline cases. This gives the network less freedom in where decision boundaries should be drawn. Finally, one must define acceptable control for the process being monitored. What may be considered poor control on one process might be quite acceptable on another.

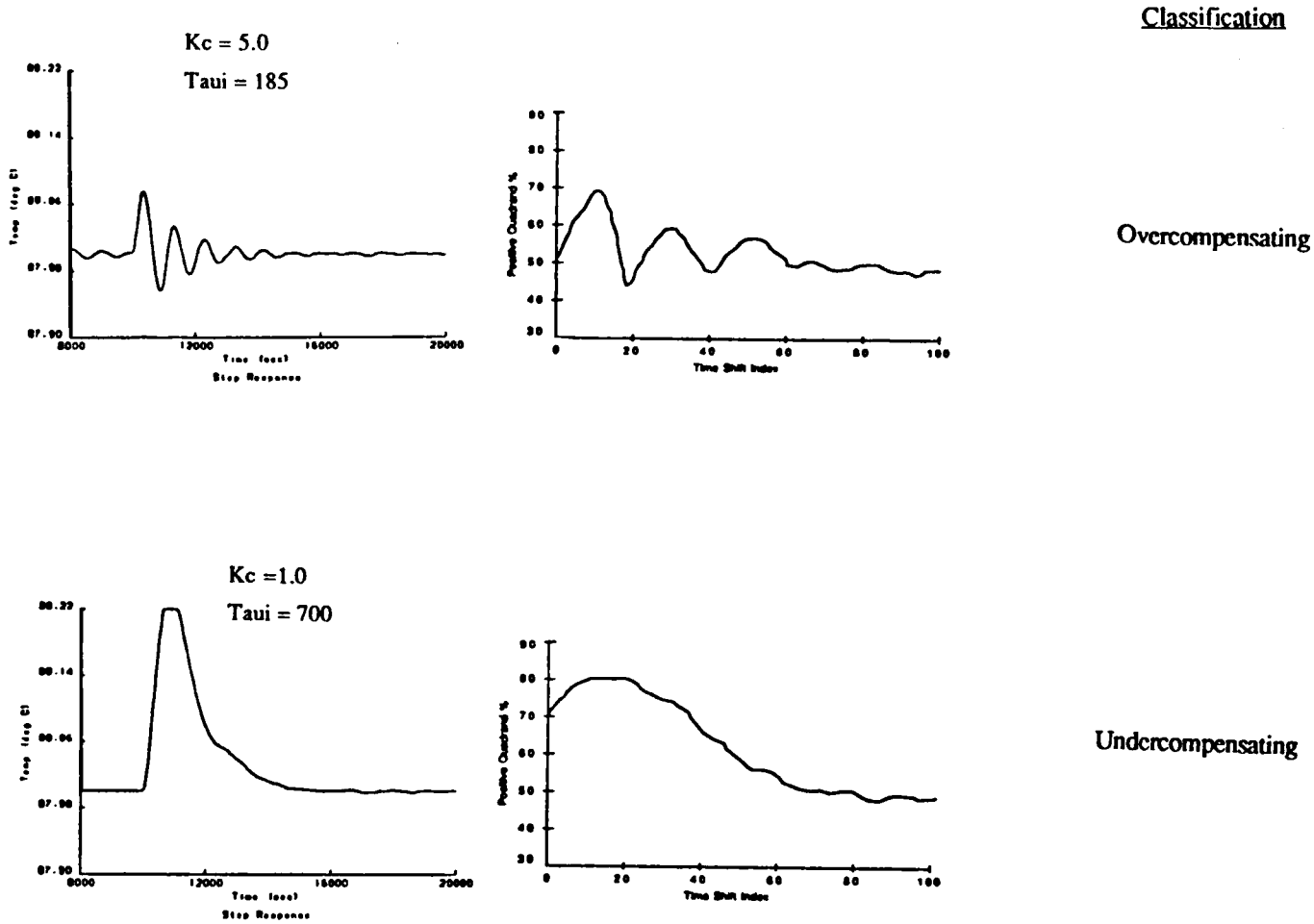


Figure 7.59: Selected examples of identifying controller performance with TSD-neural network classifier

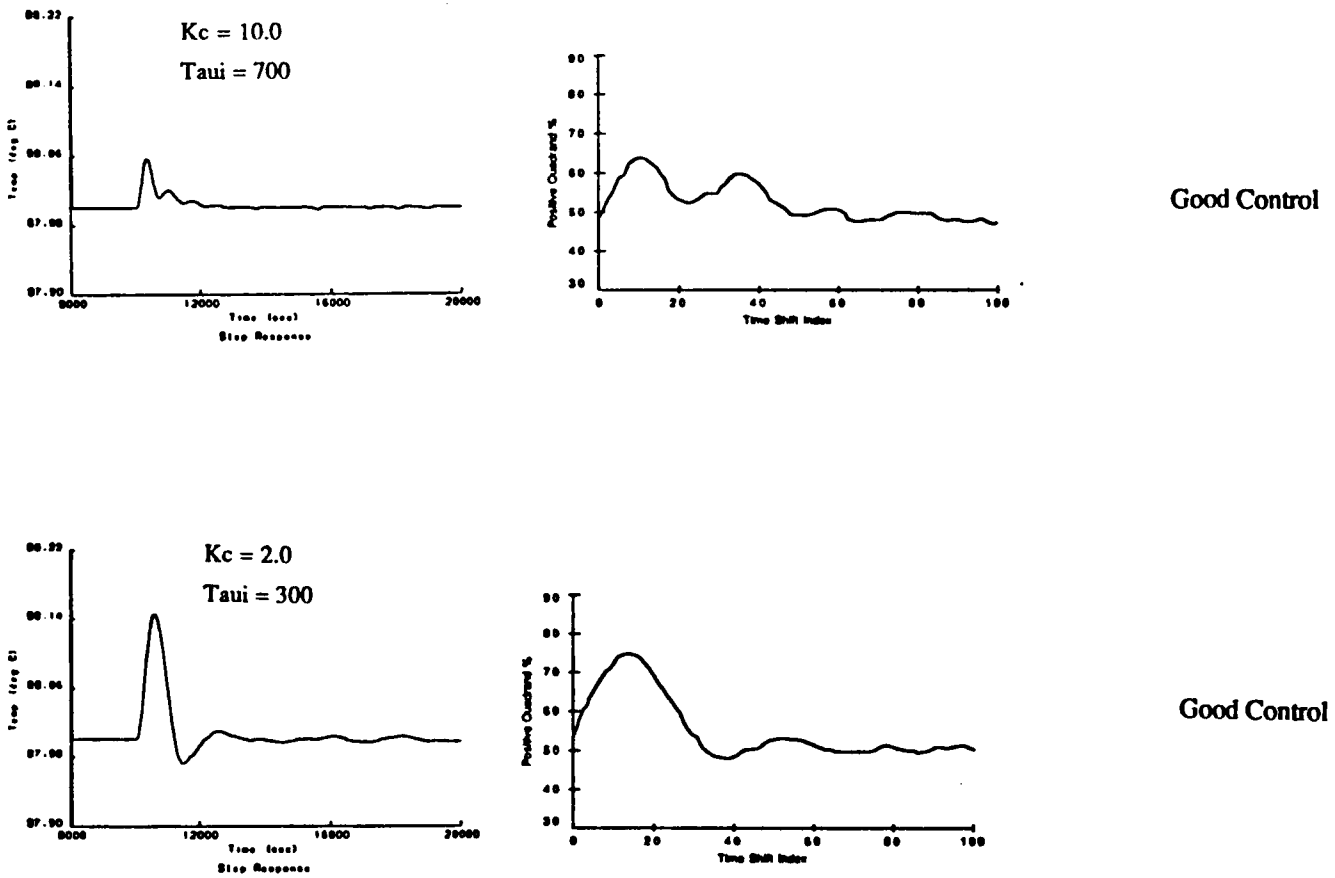


Figure 7.60: Selected examples of identifying controller performance with TSD-neural network classifier

CHAPTER 8

Conclusions and Recommendations

8.1 Conclusions

As today's processes become more complex and their control systems become more sophisticated, effective process monitoring is becoming increasingly more important. System cultivation focuses on monitoring tools that employ the computational power and storage capability of today's process computers. This work makes three contributions to system cultivation. First, it improves two existing cultivation tools: HF-modeling and TSD analysis. Second, it introduces pattern recognition techniques and neural networks to the cultivation framework. Finally, it incorporates all of these tools into a framework that effectively detects and diagnoses even subtle system changes. The following can be concluded about this work and its significance.

8.1.1 HF-modeling

1. HF-modeling takes a geometric approach to process monitoring. By incorporating only a limited knowledge of the process, it distills large amounts of data into meaningful information. It results in a series of easy-to-interpret graphs and charts in which subtle relationships and meaningful statistics are more easily monitored.

2. HF-modeling results in a steady-state nonparametric model, based entirely on historical data.

3. Monitoring graphical images resulting from HF-modeling allows subtle process problems and inefficiencies to be detected and identified.

4. Examining statistics from a multivariable framework such as HF-modeling, can detect much more subtle process changes than commonly-used, single-variable analysis.

8.1.2 TSD analysis

1. The TSD analysis examines dynamic relationships between process input and output.

2. TSD analysis reliably identifies overcompensation and undercompensation for any type of controller.

3. Incorporating a steady-state predicted controller output (via HF-modeling), U_{ss}^{*}, significantly improves the TSD analysis of a PID controller.

4. Applying TSD to MIMO processes, even for a $2x2$ case, is significantly more complex than in SISO processes.

8.1.3 Pattern recognition and neural networks

1. Neural networks are massively parallel, highly nonlinear models.

2. As a pattern classifier, the multilayer perceptron network with back-propagation learning can form nonlinear decision boundaries and form enclosed or disjoint regions. A null region between classes will always exist. The decision boundary shapes and null space size are dependent on many factors: primary network parameters such as the gain term and threshold, training convergence constraints, and the number of nodes in the hidden layers of the network.

3. The network classifies physically scaled raw data surprisingly well. However, long learning time and an inability to distinguish between all system problems limit the effectiveness of employing neural networks without intelligent data preprocessing.

8.1.4 Problem recognition framework

1. Coupling HF-modeling with neural network leads to a framework that takes full advantage of the strengths of each method. HF-modeling enhances subtle system relationships while reducing the size of training sets. Neural networks automatically interpret information by learning through experience.

2. The framework was found to be much more effective than employing neural networks or HF-modeling independently. Neural network training time was reduced by a factor of 10. The framework distinguishes between more system problems than the network alone and identifies small system

problems more reliably.

3. The framework reliably identifies problem magnitude, indicates new system problems, and labels two or more problems that are indistinguishable.

4. Scaling information presented to the neural network is a critical factor in performance. Basecase normalized scaling was found to be effective in achieving reliable classification of even very small magnitudes of process problems and helps evaluate problem magnitude.

5. Examining TSD plots with the multilayer perceptron network yielded a automatic, reliable method of identifying poor controller performance.

8.2 Recommendations

1. Test the problem recognition framework on historical data from an industrial scale operation.

2. Apply clustering analysis or self-organization maps to the analysis of real data. These techniques will help to sort large data sets into smaller data sets representing constant process operation. Such an analysis is essential prior to implementing the problem recognition framework.

3. Undertake further studies on MIMO TSD analysis. The TSD surfaces can either be simplified or systematically examined with image processing and pattern recognition.

4. Investigate the effects of unmeasured disturbances on the cultivation analysis. The changes in the present analysis necessary for this investigation will probably examine patterns in viewing space cross-sections rather than examining multivariable statistical plots and relationship plots.

5. The contributions of this work have centered on detecting and identifying process changes. Further cultivation studies should also concern the much more difficult challenge of improving upon normal process operation.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Bailey, D. and Thompson, D., "How to Develop Neural-Network", *AI Expert*, **June**:37–47, 1990.

[2] Balchen, J. and Mumme, K., *Process Control Structures and Applications*, Van Nostrand Reinhold Co. Inc., New York, 1988.

[3] Bavarian, B., "Introduction to Neural Networks for Intelligent Control", *IEEE Control Systems Magazine*, **8**(2):3–7, 1988.

[4] Bendele, M.J., *Application of System Cultivation and the Philosophy of Statistical Process Control to a PI Flow Control Loop*, Master's thesis, The University of Tennessee, Knoxville, 1989.

[5] Birdwell, J. and Cockett, J., "Inductive Inference of Model Structure Using Hypothesis Feedback", 1988, Dept. of Elect. and Computer Eng., Univ. of Tennessee.

[6] Brzakovic, D., "Pattern Recognition 4820, class notes", 1986, UTK.

[7] Carpenter, G. and Grossberg, S., "The ART of Adaptive Pattern Recognition by a Self-organizing Neural Network", *IEEE Computer*, **March**():77–88, 1988.

[8] Carpenter, G. and Grossberg, S., "A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine", *Computer Vision, Graphics, and Image Processing*, **37**():54–115, 1987.

[9] Caudill, M., "Neural Networks Primer", *AI Expert*, **November**():57–64, 1988.

[10] Deming, W. E., "Some Principles of the Shewhart Methods of Quality Control", *Mechanical Engineering*, ():, 1944.

[11] Deshpande, P. B. and Ash, R. H., *Computer Process Control*, Instrument Society of America, 1981.

[12] DeSieno, D., "Adding a Conscience to Competitive Learning", In *ICNN*, San Diego, CA, 1988.

[13] Dietz, W., Kiech, E., and Ali, M., "Pattern-Based Fault Diagnosis using Neural Networks", 1986, UT Space Institute.

[14] Duda, R. and Hart, P., *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.

[15] Elsley, R., "A Learning Architecture for Control Based on Back-propagation Neural Networks", In *ICNN*, San Diego, CA, 1988.

[16] Farell, A. E., "An Application of Neural Networks to System Cultivation Methods", 1988, A report submitted to the Measurement and Control Engineering Center at the University of Tennessee, Knoxville.

[17] Farell, A. E., *Application of Process Control Cultivation to a Liquid-Liquid Heat Exchanger*, Master's thesis, The University of Tennessee, Knoxville, 1987.

[18] Feldman, J.A. and Ballard, D.H., "Connectionist Models and Their Properties", *Cognitive Science*, **6**():205–254, 1982.

[19] Fukunaga, K., *Handbook of Pattern Recognition and Image Processing, (Chapter 1)*, Academic Press, Inc, 1986.

[20] Fukushima, K., "Cognitron: A Self-organizing Multilayered Neural Network", *Biological Cybernetics*, **20**():121–136, 1975.

[21] Gallinari, P., Thiria, S., and Soulie, F. F., "Multilayer Perceptrons and Data Analysis", In *IEEE 2nd Int. Conf. on Neural Networks. Proceedings*, pp. 391–399, San Diego, 1988.

[22] Giles, C. and Maxwell, T., "Learning, Invarience, and Generalization in High-order Neural Networks", *Applied Optics*, **26**(23):4972–4978, 1987.

[23] Grossberg, S., "Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors", *Biological Cybernetics*, **23**():125–134, 1976.

[24] Guez, A. and Eilbert, J., "Neural Network Architectures for Control", *IEEE Control Systems Magazine*, **April**():22–30, 1988.

[25] Hackney, J.E., *Cultivation of Model Predictive Control Systems*, PhD thesis, The University of Tennessee, Knoxville, 1989.

[26] Haken, H., *Synergetics*, Springer-Verlag, Berlin, 1984.

[27] Hecht-Nielson, R., "Applications of Counterpropagation Networks", *Neural Networks*, **1**():131–139, 1988.

[28] Hecht-Nielson, R., "Kolmorgorov's Mapping Neural Network Existence Theorem", In *Proceedings of IEEE Conference on Neural Networks*, San Diego, CA, 1987.

[29] Hect-Nielsen, R., "Applications of Counterpropagation Networks", *Neural Networks*, **1**:131–139, 1988.

[30] Hect-Nielson, R., "Counterpropagation Networks", *Applied Optics*, **26**(23):4979–4984, 1987.

[31] Himmelblau, D. M., *Fault Detection and Diagnosis in Chemical and Petrochemical Processes*, Elsevier Scientific Publishing Co., New York, 1978.

[32] Irie, B. and Miyake, S., "Capabilities of Three-layered Perceptrons", In *ICNN*, San Diego, CA, 1988.

[33] Kohonen, T., "Adaptive, Associative, and Self-organizing Functions in Neural Computing", *Applied Optics*, **26**(23):4910–4918, 1987.

[34] Kohonen, T., "Analysis of a Simple Self-Organizing Process", *Biological Cybernetics*, **44**():135–140, 1982.

[35] Kohonen, T., Gyorgy, B., and Chrisley, R., "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies", In *ICNN*, San Diego, CA, 1988.

[36] Kosko, B., "Adaptive Bidirectional Associative Memories", *Applied Optics*, **26**(23):4947–4960, 1987.

[37] Kosko, B., "Adaptive Inference in Fuzzy Knowledge Networks", In *ICNN*, San Diego, CA, 1988.

[38] Kosko, B., "Bidirectional Associative Memories", *IEEE Transactions on Systems, Man, and Cybernetics*, **18**(1):49–60, 1988.

[39] Kuczewski, R. M., Myers, M. H., and Crawford, W. J., "Exploration of Backward Error Propagation as a Self-Organizational Structure", In *Proceedings of IEEE Conference on Neural Networks*, San Diego, CA, 1987.

[40] Kung, S. and Hwang, J., "An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back-Propagation Learning", In *ICNN*, San Diego, CA, 1988.

[41] Kurogi, S., "A Model of Neural Network for Spatiotemporal Pattern Recognition", *Biological Cybernetics*, **57**():103–114, 1987.

[42] Lippmann, R.P., "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, **April**():4–22, 1987.

[43] Luttrell, S.P., "Self-Organising Multilayer Topographic Mappings", In *ICNN*, San Diego, CA, 1988.

[44] MacGregor, J.F., "On-Line Statistical Process Control", *Chemical Engineering Progress*, **October**():21–30, 1988.

[45] Minsky, M. and Papert, S., *Perceptrons*, MIT Press, Cambridge, 1969.

[46] Minsky, M. L., *Perceptrons*, The MIT Press, 1972.

[47] Moore, B. C., "An Academic Spy in the Real World of Industrial Computer Control: A Candid Report", 1986, Technical Report Number 8602.

[48] Moore, B. C., "EE1633F Class Notes", 1978, The University of Toronto.

[49] Moore, B. C., "Hypothesis Feedback Models for Multivariable Systems", 1986, Submitted to *IEEE Trans. on Auto. Control.*

[50] Moore, B. C., "Multivariable Challenges in the Petrochemical Industry", 1986, Technical Report Number 8601.

[51] Moore, B.C., "Systematic Attention to Detail by Production Organizations for Sustained Improvement", In *Second Nat. Sym. on Statistical Process Control*, Tempe, Az, 1987.

[52] Moore, B.C., "What and Who is in Control?", In *The Shell Process Control Workshop*, 1988.

[53] Moore, C. F., "ChE 6900 Class Notes", 1985, The University of Tennessee, Knoxville.

[54] Moore, C. F., "Multivariable Framework for Statistical Quality Control", 1986, UTK.

[55] Naidu, S.R. Zafiriou, E. and McAvoy, T., "Use of Neural Networks for Sensor Failure Detection in a Control System", *IEEE Control Systems Magazine*, 34–40, 1990.

[56] Neter, J. Wasserman, W. and M., Kutner, *Applied Linear Statistical Models*, Irwin, Homewood, Illinois, 1985.

[57] Oakland, J. S., *Statistical Process Control*, John Wiley and Sons, Inc., New York, 1986.

[58] Pao, Y.H. and Sobajic, D.J., "Metric Synthesis and Concept Discovery with Connectionist Networks", In *IEEE International Conf. on Systems, Man, and Cybernetics*, Washington, D.C., 1987.

[59] Press, W. H., et al., *Numerical Recipes*, Cambridge University Press, New York, 1986.

[60] Psaltis, D., Athanasios, S., and Yamamura, A., "A Multilayer Neural Network Controller", *IEEE Control Systems Magazine*, **April**():17–25, 1988.

[61] Ramanujam, J. and Sadayappan, P., "Optimization by Neural Networks", In *ICNN*, San Diego, CA, 1988.

[62] Ritter and Schulten, "On the Stationary State of Kohonen's Self-organizing Sensory Mapping", *Biological Cybernetics*, **54**():99–106, 1986.

[63] Ritter, H. and Schulten, K., "Kohonen's Self-Organizing Maps: Exploring their Computational Capabilities", In *ICNN*, San Diego, CA, 1988.

[64] Rosenblatt, F., "The Perceptron: A Probilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, **65**:386, 1958.

[65] Rumelhart, D. and McClelland, J., *Parallel Distributed Processing, Vol. I and II*, MIT Press, Cambridge Mass., 1987.

[66] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation", 318–362, 1987.

[67] Shewhart, W. A., *Economic Control of Quality of Monufactured Product*, Van Hontrand Co., Inc., 1931.

[68] Shipanski, J., "Fast Learning in Artificial Neural Systems: Multilayer Perceptron Training Using Optimal Estimation", In *ICNN*, San Diego, CA, 1988.

[69] Shrier, S., Barron, R. L., and Gilstrap, L. O., "Polynomial and Neural Networks: Analogies and Engineering Applications", II431–II439, 1987.

[70] Smith, C. A. and Corripio, A. B., *Principles and Practice of Automatic Process Control*, John Wiley & Sons, New York, 1985.

[71] Soulie, F., Gallinari, P., Cun, Y. L., and Thira, S., "Evaluation of Network Architectures on Test Learning Tasks", In *Proceedings of IEEE Conference on Neural Networks*, San Diego, CA, 1987.

[72] Tou, J.T. and Gonzalez, R.C., *Pattern Recognition Principles*, Addison-Wesley Publishing Company, Reading, MA, 1974.

[73] Trivedi, H., "Pattern Recognition 571, class notes", 1987, UTK.

[74] Vold, T. G., "Bayesian Image Processing, Gibbs Distributions, and Neural Networks", In *Proceedings of IEEE Conference on Neural Networks*, San Diego, CA, 1987.

[75] Von Lehmen, A. and Paek, E., "Factors Influencing Learning by Back-propagation", In *ICNN*, San Diego, CA, 1988.

[76] Widrow, B., "Adaptive Filters", 563–587, 1982.

[77] Widrow, B., "Adaptive Neural Networks. A Tutorial", 1988, Lecture Notes from NE 597 by Dr. Uhrig.

[78] Widrow, B. and Smith, F. W., "Pattern-Recognizing Control Systems", 288–317, 1964.

[79] Widrow, B. and Stearns, S., *Adaptive Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, 1985.

[80] Widrow, B. and Winter, R., "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition", *IEEE Computer*, **18**(3):25–39, 1988.

[81] Widrow, B., Winter, R. G., and Baxter, R. A., "Learning Phenomena in Layered Neural Networks", In *IEEE 1st Int. Conf. on Neural Networks. Proceedings*, pp. 411–429, San Diego, 1987.

[82] Zadeh, "Fuzzy sets", *Inform. Contr.*, **8**:338–353, 1965.

VITA

Andrew Earl Farell was born in Wilmington, Delaware on October 28, 1961. He attended public school in Parkersburg, West Virginia and graduated from Parkersburg High School in June 1980. The following September he entered The Ohio State University and in June 1985 received a Bachelor of Science degree in Chemical Engineering. He is a member of Tau Beta Pi, Phi Kappa Phi, and the American Institute of Chemical Engineers.