Neural Networks

in

Bioprocessing and Chemical Engineering

by

D. R. Baughman

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

Doctor of Philosophy
in
Chemical Engineering

Approved:

Dr. Y. A. Liu, Chairman

Dr. William L. Conger

Dr. Peter R. Rony

Dr. Richey Davis

Dr. Henry A. McGee, Jr.

Key words: neural network, artificial intelligence, bioprocessing, chemical engineering

NEURAL NETWORKS IN BIOPROCESSING AND CHEMICAL ENGINEERING

by

D. R. Baughman

Dr. Y. A. Liu, Chairman

Chemical Engineering

(ABSTRACT)

This dissertation introduces the fundamental principles and practical aspects of neural networks, focusing on their applications in bioprocessing and chemical engineering. This study introduces neural networks and provides an overview of their structures, strengths, and limitations, together with a survey of their potential and commercial applications (Chapter 1). In addition to covering both the fundamental and practical aspects of neural computing (Chapter 2), this dissertation demonstrates, by numerous illustrative examples, practice problems, and detailed case studies, how to develop, train and apply neural networks in bioprocessing and chemical engineering. This study includes the neural network applications of interest to the biotechnologists and chemical engineers in four main groups: (1) fault classification and feature categorization (Chapter 3); (2) prediction and optimization (Chapter 4); (3) process forecasting, modeling, and control of time-dependent systems (Chapter 5); and (4) preliminary design of complex processes using a hybrid combination of expert systems and neural networks (Chapter 6).

This dissertation is also unique in that it includes the following ten detailed case studies of neural network applications in bioprocessing and chemical engineering:

- Process fault-diagnosis of a chemical reactor.
- Leonard-Kramer fault-classification problem.
- Process fault-diagnosis for an unsteady-state continuous stirred-tank reactor system.
- Classification of protein secondary-structure categories.
- Quantitative prediction and regression analysis of complex chemical kinetics.
- Software-based sensors for quantitative predictions of product compositions from fluorescent spectra in bioprocessing.
- Quality control and optimization of an autoclave curing process for manufacturing composite materials.
- Predictive modeling of an experimental batch fermentation process.
- Supervisory control of the Tennessee Eastman plantwide control problem
- Predictive modeling and optimal design of extractive bioseparation in aqueous two-phase systems

This dissertation also includes a glossary, which explains the terminology used in neural network applications in science and engineering.

Acknowledgments

It is a pleasure to thank a number of very special persons and organizations who contributed to the preparation of this dissertation.

I would like to thank the members of my advisory committee, in particular: Professor Y. A. Liu, who developed the original idea and the details of the dissertation and was the major advisor; Professor Peter Rony, whose thorough review and comments proved invaluable and who assisted me with the Tennessee Eastman plantwide control problem; Professor Richey Davis for his support and encouragement; Professor William Conger, the Department Head who provided both technical and logistic support; and finally, Professor Henry McGee, who served on the committee even while on leave at the National Science Foundation as Director of the Division of Chemical and Thermal Systems, and after his move to the Virginia Commonwealth University as Associate Provost for Engineering.

I wish to thank NeuralWare, Inc., Pittsburgh, PA for providing us with a free copy of their PC-based software, Professional II/PLUS, for use in our case studies.

In addition, I wish to express my sincere gratitude to Professor Michael Mavrovouniotis of Northwestern University, Professor Thomas McAvoy of the University of Maryland, Professor George Stephanopoulos of the Massachusetts Institute of Technology, Professor Lyle Ungar of the University of Pennsylvania, Professor Venkat

Venkatasubramanian of Purdue University, and Professor Nam-Sun Wang of the University of Maryland, who have kindly provided me with course notes, student dissertations, data files, preprints, and reprints on neural networks.

I should like to thank Dr. Gulam (Sam) Samdani, Associate Editor, Chemical Engineering Magazine, for his continued support and encouragement, particularly in providing me with examples of commercial and emerging applications of neural networks.

To Ms. Marie Paretti, Department of English, University of Wisconsin, I owe a deep gratitude for her tireless and skillful copy-editing of the entire manuscript for this dissertation.

I would like to express our appreciation to the Eastman Chemical Company, for providing a fellowship award. I wish also to thank Messrs. Robert Inge, Craig Lane, Richard Rodgers, and Ms. Ashley Wright, chemical engineering seniors at Virginia Tech (1993-94), for their undergraduate research projects that contributed to the experimental fermentation modeling and the Tennessee Eastman plantwide control problem. In addition, I are grateful to a grant from the National Science Foundation for establishing a SUCEED Center of Excellence in Undergraduate Engineering Design at Virginia Tech, which provides financial support to these undergraduate research projects.

Lastly, I wish to thank Mrs. Diane Cannaday, Department of Chemical Engineering, Virginia Tech, who patiently and graciously helped in the preparation of this writing project.

Table of Contents

List of Figures

Figure 5.45. A neutralization operation in a continuous stirred-tank reactor (CSTR) (Bhat and McAvoy, 1990).

Figure 5.46. An illustration of the known network inputs (solid line) and unknown network output (dashed line) in the development of (a) the forward model, $G_{p,nn}$; and (2) the inverse model, $G^{-1}_{p,nn}$, of a neutralization operation.

Figure 5.47. A neural network model,$G^{-1}_{p,nn}$, being used as a feedforward controller based on the setpoint in parallel with a feedback controller, $G_{c,fb}$, for the neutralization operation.

Figure 5.48. A simplified internal model control (IMC) scheme for the neutralization operation, employing both neural forward and inverse models (Bhat and McAvoy, 1990).

Figure 5.49. Block diagram of (a) the classical feedback control, and (b) the internal model control. Process model, G_p, incorporates the effects of the final control element and measurement dynamics, according to equation 5.15.

Figure 5.50. An illustration of a simple signal filter called a low-pass filter (Ogunnaike and Ray, 1994).

Figure 5.51. An illustration of the four basic elements of a model-predictive control scheme: (1) reference-trajectory specification - setpoint $y_d(t)$; (2) process-output prediction - predicted output $y_p(t)$; (3) process-output prediction - predicted output $y_p(t)$; (4) error-prediction update - output error $\varepsilon(t) = | y_p(t) - y_m(t) |$ (Ogunnaike and Ray, 1994).

Figure 5.52. A model-predictive control scheme based on a neural forward process model and a process optimizer for the neutralization operation (Donat et al., 1991).

Figure 5.53a-b. An illustration of the components of NeuCOP, the Neural Control and Optimization Package: (a) NeuCOP Modeler and Optimizer; and (b) NeuCOP Controller (NeuralWare, 1994).

Figure 5.54. The desired cell-growth profile of the fermentation, along with the actual cell concentrations c(t) at time 0,.., t, and the predicted cell concentrations $c_{nn}(t)$ at time t+1,.., t+4.

Figure 5.55. The standard form of the membership function used for the three variables ($\Delta c(t)$, $\Delta c_{nn}(t+4)$, and ΔT) which are divided into five membership groups: NL - negative large, NM - negative middle, ZE - zero, PM - positive middle, and PL - positive large.

Figure 6.33a-d. The response-surface plots of selectivity values for partitioning of protein mixtures in ATPS of PEG and Dextran of various molecular weights at increasing pH values. Note the region of optimal operating conditions with highest selectivity values in (d).

List of Tables

Table 6.9. Kidera's (a) nine characteristic properties and (b) ten characteristic factors for the twenty naturally occurring amino acids (Kidera et al., 1985).

Table 6.10a-e. Proteins and their secondary structure used in the training of the protein-property network, file *ahlxtrn.nna* and *bshttrn.nna*.

Table 6.11. The format of the file *ahlxtrn.nna* and *bshttrn.nna* used for training the protein-property network.

Table 6.12. The specifications for the protein-property network..

Table 6.13. Proteins and their secondary structure used in the testing of the protein-property network, file *ahlxtst.nna* and *bshttst.nna*.

Table 6.14. The correlation coefficients for secondary-structure prediction.

Table 6.15. The specifications of property values for five proteins used in the illustrative predictions by the protein-partitioning network.

Table 6.16. The anion and cation hard-sphere diameters in dilute aqueous solutions at 25°C (Haynes et al., 1993).

Table 6.17. The hydration numbers (h's) for anions and cations in water at 25 °C (Haynes, 1992).

Table 6.18. The ion-ion specific interaction coefficients, β_{ij}, in aqueous solutions at 25 °C (Haynes et al., 1993).

Table 6.19. The format of the files *part.nna* and *phpart.nna* used for training the protein-partitioning network.

Table 6.20. The specifications for the protein-partitioning network.

Table 6.21. The correlation coefficients for illustrative predictions of protein-partitioning coefficients for the polymer solutions at Standard Conditions, $\ln K_s$, and for variations from the isoelectric point, $\ln K_{pH}$ by the protein-partitioning neural network of Figure 6.14.

Table 6.22. The separation-efficiency classifier from selectivity and phase classes.

Table 6.23. Protein-partitioning coefficient matrix.

Table 6.P1. The protein chain length and amino-acid compositions of four proteins to be separated.

Chapter 1

Introduction to Neural Networks

This text introduces the fundamental concepts and practice of neural networks, emphasizing their applications in bioprocessing and chemical engineering. In this chapter, we introduce the basic concepts, beginning with simple definitions, and then present a brief overview of the structure of neural networks and describe their

development. Next, we explain their properties, compare them to empirical modeling techniques, and list many of their strengths and limitations. Finally, we describe the potential applications of neural networks (both in general and as applied to bioprocessing and chemical engineering), and cite some recent reports of commercial and emerging uses. We note that parts of Sections 1.1 and 1.2 have been adopted and updated from Quantrille and Liu (1991; pp. 440-445; 481-483).

1.1 Introduction

A. Artificial Intelligence

The term "neural network" resulted from artificial intelligence (AI) research, which attempts to understand and model brain behavior. According to Barr and Feigenbaum (1981):

> *Artificial intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behavior.*

This definition simply states that the goal of AI is to make computers "think," to make

them solve problems requiring human intelligence.

Focusing on the means of achieving this goal, Buchanan and Shortliffe (1983) offer another definition of AI:

> *Artificial intelligence is the branch of computer science dealing with symbolic, non-algorithmic methods of problem solving.*

This second definition of AI emphasizes two aspects of AI-based methods for problem-solving. First, AI does not use an algorithm, that is, a formal procedure specifying a step-by-step execution path that guarantees a correct or optimal solution at some point. Second, AI involves *symbolic processing*, a branch of computer science that deals with non-numerical symbols and names. In contrast, the more classical *numerical processing* deals with numerical calculations and processes.

Three major AI-based technologies with growing technical and commercial significance are expert systems, neural networks, and fuzzy-logic systems (VerDuin, 1995; Crowe and Vassiliadis, 1995). Quantrille and Liu's previous text, *Artificial Intelligence in Chemical Engineering* (Quanttrille and Liu, 1991), focuses on expert systems, while this book concentrates on neural network applications. We also briefly discuss fuzzy-logic systems in Section 2.6.D.

B. Expert Systems, Neural Networks, and Subsymbolic Processing

An expert system (see also Section 6.1.B), also known as a knowledge-based system, is a computer program that uses high-quality, in-depth, knowledge to solve complex and advanced problems typically requiring human experts. Expert systems operate *symbolically*, on a *macroscopic* scale, processing non-numerical symbols and names. They require knowledge of relationships and do not care how these relationships develop.

Neural networks, on the other hand, use *subsymbolic processing*, characterized by microscopic interactions that eventually manifest themselves as macroscopic, symbolic, intelligent behavior. Thus, Robert Hecht-Nielson (1990) defines neural networks as follows:

> *A neural network is a computing system made up of a number of simple, highly interconnected nodes or processing elements, which process information by its dynamic state response to external inputs.*

The goal of a neural network is to map a set of input patterns onto a corresponding set of output patterns. The network accomplishes this mapping by first learning from a series of past examples defining sets of input and output correspondences for the given system. The network then applies what it has learned to a new input pattern to predict the appropriate output.

To understand neural networks more clearly, let us consider how the human brain functions. Most neurologists believe that true intelligence goes beyond the kind of symbolic processing used in expert systems. In the human brain, neurons within the nervous system interact in a complex fashion. The human senses detect stimuli, and send this "input" information (via neurons) to the brain. Within the brain, other neurons are excited, and they interact with each other. Based on the input, the brain reaches a "conclusion", and sends an "output" in the form of an answer or a response. The microscopic interactions between neurons is invisible, but they manifest themselves as identifiable behavior. For example, assume that you touch the side of a pot of boiling water. Your sense of touch sends its input to your brain, which causes neurons there to interact - in ways invisible to you. As a result of these interactions, though, your brain sends an output causing you to withdraw your hand. The input - touching the hot surface - and the output - withdrawing your hand - are clear, but the processing in your brain that led from one to the other remains hidden.

Neural networks use the same type of structure for computer modeling of intelligent behavior. Neurologists and artificial intelligence researchers have proposed a highly interconnected network of "neurons," or nodes, for this purpose. Using a computer, information is input into a network of artificial nodes. These nodes mathematically interact with each other in ways unknown by the user. Eventually, based on the input, the network produces an output that maps the expected, macroscopic input-output pattern. The *microscopic, subsymbolic processing* that occurs in neural

networks manifests itself as *macroscopic, symbolic, intelligent behavior.*

C. Neural Network Architecture

Figure 1.1 shows a typical neural network formed by an interconnection of nodes. This neural networks has an *input layer*, at least one (normally 1 to 3) *hidden layer(s)*, and an *output layer*.

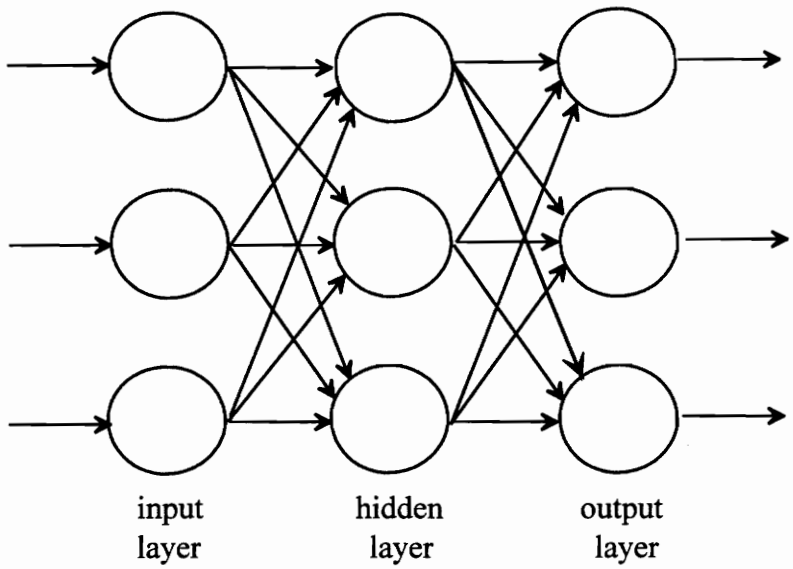


Figure 1.1. A typical multilayer neural network with one hidden layer.

Each layer is essential to the success of the network. We can view a neural network as a "black box" into which we send a specific input to each node in the input layer. The network processes this information through the interconnections between nodes, but the

entire processing step is hidden from us. Finally, the network gives us an output from the nodes on the output layer. We can summarize the purpose of each layer as follows:

- *Input Layer* - receives information from an external source, and passes this information to the network for processing.
- *Hidden Layer* - receives information from the input layer, and "quietly" does all of the information processing. The entire processing step is hidden from view.
- *Output Layer* - receives processed information from the network, and sends the results out to an external receptor.

When the input layer receives information from an external source, it becomes "activated" and emits signals to its neighbors. The neighbors receive excitation from the input layer, and in turn emit an output to their neighbors. Depending on the strength of the interconnections (i.e., the magnitude of the so-called "weight factor" that adjusts the strength of the input signal), these signals can excite *or* inhibit the nodes. What results is a pattern of activation that eventually manifests itself in the output layer. Depending on the strength of the interconnections, signals can excite *or* inhibit the nodes.

One important characteristic of neural networks is that within the network, the processing is *numerical, not symbolic* (although the results *can* manifest themselves symbolically - hence the name *subsymbolic*). The network retains information through:

(1) the magnitudes of the signals passing through the network, and

(2) the connections between nodes and their neighbors.

Because the network retains *numerical* information, it can function as a multivariable computing tool.

D. Developing Neural Networks

Developing a neural network requires three phases:

- *the training or learning phase*,
- *the recall phase*, and
- *the generalization phase*.

In the *training* or *learning* phase, we repeatedly present a set of known input-output patterns to the network in order to teach it. We adjust the weight factors between nodes until the specified input yields the desired output. Through these adjustments, the neural network "learns" the correct input-output response behavior. In neural network development, this phase is typically the longest and most time-consuming, and it is critical to the success of the network. We may use some error-correction scheme in this phase to minimize the output errors. Error-correction is numerically intensive, and there are many different ways to perform error-correction to teach the network how to respond,

as described in Chapter 2.

After the training phase, we move to the *recall* and *generalization* phases. In the recall phase, we subject the network to a wide array of input patterns seen in training, and introduce adjustments to make the system more reliable and robust. During the generalization phase, we subject the network to input patterns it has not seen before, but whose outputs are known to us, and monitor the system's performance.

1.2 Properties of Neural Networks

A. Strengths of Neural Networks

Neural networks have a number of properties that give them advantages over other computational techniques, as described below.

(1) *Information is distributed over a field of nodes.* This distribution provides greater flexibility than one finds in symbolic processing, where information is held in one fixed location.

(2) *Neural networks have the ability to learn.* If an error or a novel situation occurs that produces inaccurate system results, we can use an error-correction training technique to correct it by adjusting the strengths of the signals emitted from the nodes until the output error

disappears. At that point, the system has effectively "learned" how to handle the new input. When the system encounters that situation in the future, the network will model it properly.

In symbolic processing, such as a fault-diagnosis expert system, learning has proven a very difficult roadblock. Typically, the system is unreliable when it faces a novel situation. The programmer must develop a new set of rules (rather than simply making adjustments) to correct the situation. Consequently, a system that has the ability to learn significantly improves accuracy and efficiency.

(3) *Neural networks allow extensive knowledge indexing.* Knowledge indexing is the ability to store a large amount of information and access it easily. In a symbolic program using rule-based knowledge, knowledge indexing can be awkward. The information may not be easily accessible, and retrieving it may waste a sizable amount of time. For this reason, in symbolic computing, we often utilize some specialized data structures that organize the properties of a given object, situation, or process into a systematic hierarchy. That hierarchy facilitates knowledge indexing.

In contrast, a neural network provides inherent knowledge indexing. It can easily recall, for example, diverse amounts of information associated with a chemical name, a process, or a set of

process conditions. The network stores/retains knowledge in two forms: (a) the connections between nodes, and (b) the weight factors of these connections. Because it has so many interconnections, the network can index and house large amounts of information corresponding to the interrelations between variables.

(4) *Neural networks are better suited for processing noisy, incomplete, or inconsistent data.* No single node within a neural network is directly responsible for associating a certain input with a certain output. Instead, each node encodes a *microfeature* of the input-output pattern. The concept of microfeature implies that each node affects the input-output pattern only *slightly*, thus minimizing the effects of noisy or incomplete data in any given node. Only when we assemble *all* the nodes together into a single coordinated network, do these microfeatures map the macroscopic input-output pattern.

Other computational techniques do not include this microfeature concept. In empirical modeling, for instance, each variable used has a significant impact in most models. Consequently, if the value of one variable is off, the model will most likely yield inaccurate results. In neural networks, however, if the value of one variable is off, the model *will not* be affected substantially.

In addition to microfeatures, neural networks have another

feature that minimize noise: the signals sent to and from nodes are continuous functions. Consequently, the network can deduce proper conclusions, even from noisy, incomplete, or inconsistent input signals.

(5) *Neural networks mimic human learning processes.* Most human learning and problem-solving occurs by trial and error. For example, if a piece of equipment is not operating correctly, we observe its symptoms and recommend corrective actions. Based on the results of those actions, we recommend additional corrections. This process continues until we properly correlate symptoms with corrective actions, and the machine operates correctly.

Neural networks function in the same fashion. We can *train* them by iteratively adjusting the strength of the connections between the nodes. After numerous iterative adjustments, the network can properly predict cause-and-effect relationships.

(6) *Automated abstraction* - neural networks can determine the essentials of input-output relationships automatically. We do not need a *domain expert*, that is, an expert in a particular problem-solving domain (e.g., a separation specialist) to develop the "knowledge base" that expert systems require. A knowledge base is simply a collection of facts, rules, and heuristics in a specific domain of application. Through

training with direct (and sometimes imprecise) numerical data, the network can automatically determine cause-and-effect relations and develop its own knowledge base.

(7) *Potential for online use* - neural networks may take a very long time to train, but once trained, they can calculate results from a given input very quickly. Since a trained network may take less than a second to calculate results, it has the potential to be used online in a control system.

B. Comparison of Neural Networks to Empirical Modeling

Consider the multilayer neural network shown in Figure 1.1, where each layer (input, hidden, and output) has three nodes. This network has a total of eighteen connections, and eighteen weight factors to adjust to train the network. An engineer may say, "Hold on here! If you give me eighteen variables, I can curve-fit almost anything. This neural network is nothing but empirical modeling, which has been around for more than fifty years. You are just doing some fancy curve-fitting."

There is truth in that claim (Mah, 1991). A neural network *is* an empirical modeling tool, and it does operate by "curve-fitting." However, some notable differences exist between neural networks and typical empirical models. As a result, neural networks offer distinct advantages in some areas, as explained below, but have limitations in other areas (see Section 1.2.B).

First, neural networks have a better *filtering capacity* than empirical models because of the *microfeature* concept, as discussed early in this section. Because each node encodes only a microfeature of the overall input-output pattern, it affects the input-output pattern only slightly. Moreover, neural networks are also *massively parallel*, so that each node operates independently of the others. We can view each node as a processor in its own right, and these processors all operate in parallel. As a result, the network does not depend on a single node as heavily as, for instance, an empirical model depends on an independent variable. Because of this parallelism, neural networks have a better filtering capacity and generally perform better than empirical models with *noisy* or *incomplete* data.

Second, neural network are more adaptive than empirical models. Neural networks have specified *training algorithms*, where we adjust weight factors between nodes until we achieve the desired input-output pattern. If conditions change such that the network performance is inadequate, we can train the neural network further under these new conditions to correct its performance. In addition, we can design the network to periodically update its input-output performance, resulting in a continuous, online, self-correcting model. Typical empirical models do not have this ability.

Third, neural networks are truly multiple-input and multiple-output (MIMO) systems. Most empirical modeling tools map one, or at most two or three dependent variables. Neural networks can map many independent variables with many dependent variables as needed. Consequently, neural networks perform better at *pattern recognition*

than traditional empirical modeling systems.

C. Limitations of Neural Networks

While neural networks have many advantages, they are not a cure-all. Before you embark on a neural network project, you should understand both the strengths *and* the limitations involved. Unfortunately, in published reports on the uses of neural networks, authors tend to emphasize the *advantages*, while deemphasizing some of the *limitations* and associated *problems*. For this reason, we recommend getting a good understanding of neural networks through *hands-on* experience with minimal initial monetary investment. Through this hands-on experience, you will be better able to determine whether or not a neural network is appropriate for a given application.

Neural networks are simply another computer-modeling tool that offers some distinct strengths over some more well-known, traditional computer-modeling techniques. Some of the limitations are:

(1) *Long training times* - Training can take long enough to make the neural network impractical. Most simple problems require at least 1000 time steps to train the network, and complex problems can require up to 75,000. As personal computers become more powerful, however, time requirements are less of a problem (e.g., training time is less than 1 hour for the most complex

network in this text using IBM 486-66MHz).

(2) *Large amount of training data* - If little input-output data exist on a problem or process, we may reconsider the use of neural networks, since they rely heavily on such data. Consequently, neural networks are best suited for problems with a large amount of historical data, or those that allow us to train the neural network with a separate simulator.

(3) *No guarantee of optimal results* - Most training techniques are capable of "tuning" the network, but they do not guarantee that the network will operate properly. The training may "bias" the network, making it accurate in some operating regions, but inaccurate in others. In addition, we may inadvertently get trapped in "local minima" during training. Section 2.2.D.1 discusses this problem in more detail.

(4) *No guarantee of 100% reliability* - While this applies to all computational applications, this point is particularly true for neural networks with limited training data.

1.3 Potential Applications of Neural Networks

Applications of neural networks to bioprocessing and chemical engineering have increased significantly since 1988. One of the first application papers was by Hoskins and Himmelblau (1988), who applied a neural network to the fault diagnosis of a

chemical reactor system. Since then, the number of research publications on neural network applications in bioprocessing and chemical engineering has risen significantly.

Neural Computing (NeuralWare, 1991) provides a good overview of potential applications of neural networks, as listed below:

- *Classification* - Use input values to predict a categorical output; e.g., given symptoms and laboratory results, determine most likely disease.
- *Prediction* - Use input variables to predict an output value; e.g., given temperature, humidity and wind velocity, predict evaporation rate.
- *Data Association* - Learn associations of error-free or ideal data, then classify or associate data that contains error; e.g., learn five ideal patterns and then recognize noisy input patterns as one of five patterns.
- *Data Conceptualization* - Analyze data and determine conceptual relationships; e.g., cluster data with many attributes so that grouping relationships can be inferred.
- *Data Filtering* - Smooth an input signal; e.g., smooth a noisy electrocardiogram signal.
- *Optimization* - Determine optimal value; e.g., determine minimum-length trip for a traveling salesperson.

Neural networks hold much promise for significant applications to bioprocessing and chemical engineering problems that are *complex, nonlinear, and uncertain.* A

well-structured and perhaps a large neural network can successfully tackle a complex modeling problem. Moreover, because neural networks are nonlinear computing tools, they can easily apply to nonlinear processing problems. Lastly, when compared to empirical, curve-fitted models, neural networks are relatively less sensitive to noise and incomplete information, and can thus deal with problems of uncertainty.

This text focuses on the bioprocessing and chemical engineering applications that primarily use classification, prediction and data-filtering networks. We concentrate on the following applications:

- *Classification Networks* (Chapter 3) use networks to identify operational faults (i.e., fault diagnosis) and to characterize distinct features (i.e., feature categorization) of both time-independent (steady-state) and time-dependent (unsteady-state) processes.
- *Prediction and Optimization Networks* (Chapter 4) predict the values of process performance variables from independent operating variables based on laboratory or plant data. For quality control applications, we combine neural networks with statistical methods to identify the major independent operating variables and their respective ranges to optimize the process performance. Note that this is not the same as using an optimization network, since the optimization is performed using statistical methods and numerical techniques rather than by the network itself. In addition, prediction networks can be used

as "software-based sensors" ("soft sensors") for quantitative predictions of variables that are not easily measurable.

- *Process-Forecasting, Modeling, and Control Networks* (Chapter 5) are prediction networks similar to those previously mentioned, but modified (using feedback or recurrent loops) to handle time-dependent data. These networks are primarily used for process optimization and adaptive process control.
- *Data-Filtering Networks* (Chapter 5) are used in conjunction with many process-forecasting networks to reduce noise in time-dependent signals. Using a smoothed input signal entering a network significantly improves the prediction capability of the network.
- *Expert Networks* (Chapter 6) generally link classification or prediction networks with expert systems to identify optimal process/product designs.

1.4 Reported Commercial and Emerging Applications

This section surveys some recent commercial and emerging applications of neural networks. We begin with some general examples, then describe specific applications in: (1) product design, formulation and manufacturing; (2) process monitoring and diagnosis; (3) process modeling; (4) process control; and (5) process optimization. Neural network applications have added a new dimension in each of these areas. Although some of the work is still in the developmental stages, neural networks are sure to continue affecting

each of these areas in the future. Some areas could be revolutionized; VerDuin (1995) in his new book, *Better Products Faster*, convincingly suggests that AI-based technologies, including expert systems, neural networks, and fuzzy-logic systems, are of strategic and technical significance in today's competitive markets. The wise engineer will know both the strengths and limitations of AI techniques, including neural networks (Mathis, 1986).

A. General Applications

- The U.S. Treasury Department's Internal Revenue Service (IRS) has chosen a neural-network-based software package, NestorReader, developed by *Nestor Inc., Providence, Rhode Island,* to process tax returns. The NestorReader system will be used to read and convert hand-printed and machine-printed information on tax forms into electronic data for further processing.

  NestorReader is already being used by the Dutch government and the State of New York to process tax forms. By eliminating manual data-entry operations, the IRS expects to reduce tax processing time and improve accuracy in data entry. The IRS projects that by 2001, an estimated 312-million tax returns will be processed through the new intelligent data-processing system. Nestor estimates that its system can save 60% to 90% of the cost of manual data entry (Anonymous, 1994a; Blanchard, 1994b).

- A Singapore firm, *NIBS Pte Ltd.,* has developed a financial neural network program, NeuroForecaster, that provides strategic investment analysis for stocks, options,

fixed-income securities, foreign exchanges, interest rates, etc. The program uses only historical data for training, and is a user-friendly tool for Windows and Macintosh for time-series forecasting, classification and indicator analysis (Anonymous, 1993; Eagan, 1993).

- *Nestor Inc., Providence, Rhode Island*, has developed a Fraud Detection System that detects fraud in the credit card industry. This neural network system learns a card-holder's pattern of credit card use, then recognizes and flags behavior outside that pattern, thus protecting individual cardholders and banks against losses due to fraud. The system is being used by the Bank of America, Canadian Imperial Bank of Commerce and Europay International (Blanchard, 1993b).
- *Pfizer Inc., Groton, Connecticut*, has developed a neural network to manage its new drug approval process (called Clinical Research Forms). This system integrates scanning, optical character recognition, and accurate retrieval of multiple data types stored in a user-defined database, and it serves as an industry model for computer-assisted new drug applications (Blanchard, 1993a).
- *Microsoft, Redmond, Washington*, has developed a neural network that helps determine the people on the mailing list that should receive a second mailing (another form of pattern recognition). This process has improved their efficiency of direct mail marketing by 35% (Blanchard, 1994a).
- *Georgia Tech, Atlanta, Georgia*, is developing a neural network to predict aircraft fires and other catastrophes by analyzing risk factors and interpreting previously

unrecognized patterns from the records of the National Transportation Safety Board (Blanchard, 1994a).

- *Anderson Memorial Hospital, Anderson, South Carolina*, has implemented a neural-network-based hospital-information and patient-prediction system, which has improved the quality of care, reduced the death rate and saved the facility millions of dollars in resources. The system provides educational system and feedback to physicians and others to improve resource efficiency and patient care delivery (Blanchard, 1994b).
- *Lockheed Research Laboratories, Palo Alto, California*, is developing a pattern-recognition neural network to aid in the detection of breast cancer at its early stages. The network is being programmed with hundreds of training examples of normal and abnormal mammograms. Once the network has learned the patterns that require further examination, it is expected to be able to detect abnormalities with characteristics similar to those in its example cases. The goal is to reduce errors in reading standard mammograms and thus achieve a higher rate of detection in the early stages of the breast cancer. The development is scheduled to be completed in mid-1995 (Schreiner, 1994).

B. Product Design, Formulation and Manufacturing

- A key challenge in making many consumer and industrial products is *product*

formulation, i.e., identifying the formula that enables the product to best meet potentially conflicting functional, processing and cost requirements. Product formulators such as Tyson Gill and Joel Shutt have reported applying both statistics and neural networks in developing product formulation models for coatings and adhesives for *the Glidden Company* and *Lord Corporation* (Gill and Shutt, 1992). They find that neural networks offer several key benefits: (1) Neural networks are inherently nonlinear and of unlimited order, so that the formulation model is not artificially constrained to be linear; (2) Neural networks are more tolerant of noisy data, that is, data with gaps and errors; (3) Neural networks are better able to extract information from data that do not meet formal design-of-experiment criteria, a common situation resulting from limitations in time and available facilities to run experiments; (4) Neural networks can handle both discrete and continuous variables simultaneously, unlike statistical methods, which are designed for one or the other; and (5) Neural network models can incorporate both ingredients and process parameters, a challenging task with statistical methods. Other researchers and industrial practitioners have reported similar results in the formulation of rubber, plastics, specialty polymer products, friction materials, metal alloys, pharmaceutical biotechnology products (VerDuin, 1995).

- *AIWARE Incorporated, Cleveland, Ohio,* has developed an integrated product formulation and optimization system for chemists, called CAD/Chem (AIWARE, 1992). This integrated system consists of: (1) a neural network to estimate the

properties of a given product formulation; (2) an expert system to run the formulation model repeatedly, in essence asking many what-if questions relating to product formulation; (3) a set of user-defined design goals that apply a fuzzy-logic-inspired method to specify ranking and preferences of product properties and constraints in ingredients, processing and costs; and (4) a product optimizer to drive the repeated what-if trials in the direction required to meet these goals. Commercial applications of the CAD/Chem system have included the optimal design of formulated products such as coatings, plastics, rubber, specialty polymers, building materials (VerDuin, 1995).

- *Sandia National Laboratories, Albuquerque, New Mexico*, is developing an automated system combining near-infrared light with neural networks to sort plastics before recycling. The system recognizes the different spectral patterns produced when the light is reflected off the various types of plastics, and the neural network then analyzes those patterns. The technique has been successful in the laboratory for classifying six types of recyclable plastics used in containers and packaging. Several neural networks have achieved 100% accuracy for most of the types of plastics tested. Currently, a recycling equipment manufacturer is testing a prototype capable of scanning whole plastic items in a tenth of a second (Chementator, 1993a).
- *AIWARE Incorporated* and *Case-Western Reserve University, Cleveland, Ohio,* have developed a Rapid Foundry Tooling System (RFTS) that incorporates neural networks and advanced computer memory techniques for computer-aided design and fabrication of aluminum sandcastings at the Air Logistics Center of Kelly Air Force Base, San

Antonio, Texas (VerDuin and Pao, 1993). Aluminum sandcastings are used as replacement parts for a variety of equipment. The RFTP is used to *reverse-engineer*, that is, to recreate the design of the castings, particularly for some specialized equipment that may have been built many years ago and for which spare parts and past drawings are no longer available. Specifically, the RFTP successfully: (1) reverse-engineers a casting, producing a computer-aided design (CAD) description; (2) validates the design with rapid prototyping; (3) ensures that both part and process designs reflect good practice to minimize likelihood of defects and to support rapid delivery of parts; and (4) captures the expertise used to accomplish these tasks, including the lessons learned, so that future design and fabrication tasks can draw upon prior experience (VerDuin, 1995).

C. Process Monitoring and Diagnosis

- *Eastman Chemical Company, Kingsport, Tennessee*, has developed a neural network system to help reduce the costs of emission monitoring, which has received attention from the Environmental Protection Agency for designation in a reference document (Blanchard, 1994c).
- *AlphaMOS, Toulouse, France* and *Neural Computer Sciences, Southampton, United Kingdom,* have teamed up to develop intelligent odor-sensing systems using neural networks, which makes them suitable for online process monitoring, such as

continuously monitoring perfumes in soaps and cosmetic bases and aromas in the food industry.

The electronic nose, called Fox 2000, developed by AlphaMOS is being used for monitoring odor from sulfur compounds and natural gas (which is odorless to the human nose). The next generation of electronic nose will be equipped with hybrid arrays of different types of sensors. For example, sensors incorporating conducting polymers, which are more discriminating than metal oxide silicon, will be commercially available in mid-1995. Using sensors for surface acoustic waves allows monitoring of small molecules of toxic gases in the range of 1-10 parts per billion. The devices can typically acquire, analyze and recognize a sample within seconds, compared to about 30-60 minutes required for conventional chromatography tests. (Chementator, 1994; Blanchard, 1994b).

- *Fujitsu and Nippon Steel Corporation, Japan*, has implemented a neural network system for monitoring and detecting process faults in the continuous casting of steel. In such an operation, the molten steel enters a water-cooled mold. The outside surface of the continuous slab of steel gradually solidifies, but must be kept continuously moving and contained within the walls of the continuous caster. On occasion, a processing defect called breakout occurs. A breakout is a tear in the solidified skin of the steel as it moves through the mold. If this tear propagates too far across the solidified surface, molten steel will leak out into the continuous casting equipment, causing an expensive production line shutdown. The neural network system has been

able to detect all cases of breakout early enough, to provide fewer false alarms and to reduce costs by millions of dollars per year (VerDuin, 1995).

D. Process Modeling

- Many companies that manufacture integrated circuits are using neural networks in the semiconductor process for modeling, monitoring and control, and process diagnosis. Several universities are also currently studying their use for process optimization. Table 1.1 list some of these companies and universities and their applications (May, 1994).

Table 1.1. Neural network applications in semiconductor manufacturing (May, 1994).

Company/ Universities	Process modeling	Process optimization	Monitoring and control	Process diagnosis
AT&T	•		•	
DuPont	•			
Kopin Corporation			•	
Intel Corporation				•
National Semiconductor	•			
Pavilion Technologies	•		•	
TI/ND/Lam			•	•
Georgia Tech	•	•	•	•
MIT		•	•	•
Univ. of Cal. at Berkeley	•	•		

- *Gensym Corporation, Cambridge, Massachusetts,* has developed a visual software tool for building neural network models, called NeurOn-Line. This system utilizes Gensym's visual programming tools to provide simple graphical blocks for data handling. It automatically builds network models from examples of actual process behavior. During the training phase, the system gathers online data into a training set. Over time, the training set accumulates experience about the behavior of a process, during normal and abnormal operations (Chementator, 1993c).
- By bolstering a neural network model of its 650,000 tons/yr ethylene cracker with realtime data of naphtha-feed grades from a near-infrared spectrometer, *BP Chemicals SA, Lavera, France,* is producing ethylene of more uniform quality, and thus saving more than $1.2 million/yr. The system has been running since October 1991.

  The spectrometer analyzes the densities and the percentages of linear and branched paraffins, aromatics, and cyclic hydrocarbons in the feed. This feed-composition data are fed to a control model employing neural networks. The model then calculates the optimal temperature, naphtha and steam flowrates to match the unit's production targets.

  Compared with traditional regression analysis, neural networks can more accurately predict the best operating conditions for feedstocks for which there are no historical data. Encouraged by the results of the cracker model, BP plans to extend the spectrometer-based neural network modeling strategy to pyrolysis gasoline to control

the benzene-toluene-xylene content and the octane numbers of hydrogenated cuts (Chementator, 1992b).

E. Process Control

- When some important process variables are hard to measure directly, control of a given process can be difficult, if not impossible. A neural network system, called Intelligent Sensor, brought to market by *Fisher-Rosemount, Austin, Texas*, does away with this problem. The system can capture the interrelationships among variables based on the historical data of a customer's plant. Using those relationships, the network can then predict unmeasurable variables from measured ones. The results of those predictions are then embedded into a Fisher-Rosemount Provox or Rosemount 3 controller. Thus, plant operators can "read" the values of the difficult-to-measure variables online based on real-time data from related process variables that are more easily measured. The system can also make online prediction of future events, such as the completion time for a batch reaction. The Intelligent Sensor is said to be virtually immune to noise and nonlinearity. Suitable for both batch and continuous processes, the sensor has proved itself in pharmaceutical and pulp-and-paper pilot applications (Chementator, 1993b).
- *NeuralWare, Inc., Pittsburgh, Pennsylvania* and *Texaco, Inc., Houston, Texas*, have developed NeuCOP, a neural control and optimization package, and have successfully applied it to distillation control problems in Texaco's Puget Sound Plant. One

application involves using a neural network model together with optimization techniques to maximize the operating profits from a distillation column as a function of feed flowrate, distillate and bottoms flowrates, steam rate, and reflux ratio. Potential applications of NeuCOP include distillation towers, boilers, blending, furnace control, heat exchangers and reactors. (Anonymous, 1993a, c; Graettinger, et al., 1994a,b)

- *A multinational chemical company (Company A)* applies neural networks to the feedforward control of a coker, and *another multinational chemical company (Company B)* uses neural networks as software-based sensors (i.e., soft sensors) in the predictive control of distillation columns in an acid plant.
- *Gekkeikan Sake Company, Ltd., Kyoto, Japan*, has developed and implemented a neural network to analyze the state characteristics of sake brewing. A fuzzy-logic control system uses these brewing characteristics to target a reference pattern in the selecting of the best neural-fuzzy control scheme (Oishi, et al., 1992).
- *Eastman Kodak Company, Rochester, New York*, and an industry consortium, *Microelectronics and Computer Technology Corporation, Austin, Texas*, have jointly developed a new class of adaptive process-control software, called Process Insights. This software uses neural networks, fuzzy logic, and chaotic system technologies to tune controller setpoints using *only* historical plant-performance data. A neural network maps the process input-output pattern behavior, and an optimization routine determines the best controller setpoints (Chementator, 1992a).

- *Radian Corporation* and *Pavilion Technologies, Inc.,* both *in Austin, Texas* have formed an alliance to bring advanced neural network systems to market. Among the key technologies is Pavilion's CEM software, used to model and estimate air emissions from stationary sources (Anonymous, 1994). Pavilion Technologies, a spin-off company from the Microelectronics and Computer Technology Corporation, Austin, Texas, is actively pursuing neural network applications in process control.
- The *Defense Advanced Research Projects Agency, Washington, D.C.*, is funding *Science Applications International Corporation (SAIC), San Diego, California*, to develop a neural-network-based integrated circuit. SAIC's generic neural network chips will be interconnected to form systems for adaptive process control and pattern recognition, and to provide building blocks for distributed neurocomputers (Chementator, 1990).

F. Process Optimization

- Neural networks are cutting the use of a chemical additive by one-third and leading to higher-quality chemical intermediates, such as aldehydes and olefins, at *Eastman Chemical Company, Texas Eastman Division, Longview, Texas.*

The plant uses an expensive chemical additive to remove byproduct impurities, but fluctuations in the byproduct levels make such treatment difficult and have led to excess additive use. To solve this, researchers from the Artificial Intelligence Group

at Texas Eastman Division and from Pavilion Technologies, Austin, Texas, trained a neural network using historical data from operational records to develop a model of the plant. By learning the patterns of the process, the network has been able to predict the impurity levels that occur during production. Researchers then apply Pavilion's integrated neural network and optimization system, Process Insights, to optimize the settings of process controllers and thus feed only the correct amount of the additive. The network's success is encouraging Texas Eastman to apply neural network technology to other production lines to realize additional cost reductions (Chementator, 1990). Detailed reports of the Texas Eastman experience in applying neural network technology to model, optimize and control chemical production operations appear in Denmark et al. (1993), and in VerDuin (1995).

- *AIWARE, Incorporated, Cleveland, Ohio*, has developed a neural network system, NeuSIGHT, for optimizing combustion operations in coal-fired power plants. The system provides plant-floor staff with recommendations for changes in control settings to improve combustion efficiency and thus reduce fuel costs as well as sulfur dioxide (SO_2) production, a contributor to acid rain. Another objective is to minimize the production of nitrogen oxides (NO_x) by-products which cause smog. In one application, the system optimizes six operating variables, including main steam temperature, main steam pressure, excess oxygen, reheat temperature, feedwater temperature and economizer outlet temperature. These variables are controlled with the single objective of lowering the heat rate.

Each installation is custom-configured to incorporate available sensor data and make best use of available hardware. System installation and training costs vary but are generally in the range of $120,000 to $180,000 for a complete system. The heat rate improvement and the resulting fuel savings alone provide estimated paybacks of 6 to 12 months for typical medium-size (400 megawatt) plants. When factoring in direct reduction in SO_2 emissions, avoided capital costs for meeting NO_x requirements, and the possibility of selling SO_2 and NO_x emission credits, the expense is recaptured more rapidly (VerDuin, 1992, 1995).

- An integrated system combining an expert system and a neural network (i.e., an expert network) called the Intelligent Arc Furnace (IAF) Controller optimizes electrode positions for more accurate heat distribution in scrap metal furnaces (Hall, 1992). The IAF Controller is sold by *Neural Applications Corporation, Devenport, Iowa,* and is now in use in a number of steel plants. The controller has saved about 2 million dollars a year for each furnace it has optimized, and the National Society of Professional Engineers named it *one of the top six U.S. engineering achievements of 1991.* The details of this significant neural network application that has revolutionized steelmaking appear in Staib (1993), and Staib and Staib (1992); we shall discuss the IAF controller in Section 5.6.B.2.

1.5 Chapter Summary

- Artificial intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behavior (Barr and Feigenbaum, 1981).
- A neural network is a computing system made up of a number of simple, highly interconnected nodes, which processes information by its dynamic state response to external inputs (Hecht-Nielson, 1990).
- A neural network can be viewed as a "black box" that processes input information, through interconnections between nodes in order to map input/output patterns.
- A typical multilayer neural network has an *input layer*, at least one (normally 1 to 3) *hidden layer(s)*, and an *output layer*. The input layer receives information from an external source and passes this information into the network for processing. The hidden layer receives information from the input layer and "quietly" does all of the information processing. The entire processing step is hidden from view. The output layer receives the processed information from the hidden layer(s), and sends the results out to an external receptor.
- Developing a neural network consists of three phases: training, recall and generalization. The training phase repeatedly presents a set of known input-output patterns to the network, adjusting the weights of the interconnections between nodes until the specified input yields the desired output. The recall phase subjects the

network to a wide array of input patterns seen in training to test its memory. The generalization phase subjects the network to new input patterns, where the system hopefully performs properly.

- Some important neural networks properties include: (1) they distribute information over a field of nodes; (2) they have the ability to learn; (3) they can store a large amount of information and asses it easily (called "knowledge indexing"); (4) they are well suited for processing noisy, incomplete, or inconsistent data; and (5) they mimic human learning processes.
- The advantages of neural networks over other computer-modeling tools include: (1) adaptive behavior; (2) multivariable pattern recognition abilities; (3) good filtering capability (low sensitivity to noise and incomplete information); (4) the ability to automatically determine the essentials of relationships; and (5) the potential for online use.
- The limitations of neural networks include: (1) long training times; (2) require a large amount of training data; (3) no guarantee of optimal results; and (4) no guarantee of 100% reliability (extrapolation problems may occur when training data is limited).
- Neural networks have found commercial applications in a variety of areas in bioprocessing and chemical engineering. Some reported examples include product design; formulation and manufacturing; process monitoring and diagnosis; process modeling; process control; and process optimization.

References and Further Reading

For an excellent introduction to AI-based technologies that are revolutionizing the design and manufacturing practices in chemical process industries (CPI), we highly recommend the new book, *Better Products Faster: A Practical Guide to Knowledge-Based Systems for Manufacturers* (VerDuin, 1995). This book does an excellent job of explaining AI-based technologies, including expert systems, neural networks, and fuzzy-logic systems, from an engineering and problem-solving perspective rather than a computer-science and research viewpoint. While pointing out the strengths and limitations of each, VerDuin makes a convincing case that these technologies are of strategic and technical significance in today's competitive marketplace. This book includes detailed examples of applications of AI-based technologies to design and manufacturing in a variety of industries. For a published review of this text, see Samdani (1995).

At the time of this writing, Quantrille and Liu's book, *Artificial Intelligence in Chemical Engineering* (Quantrille and Liu, 1991), is the only available AI text in chemical engineering. There are a number of tutorial articles on AI (Crowe and Vassiliadis, 1995), expert systems (Samdani, 1992), and neural networks (Bhagat, 1990; Chitra, 1993; Samdani, 1992), and fuzzy-logic systems (Samdani, 1993). *Computers and Chemical Engineering* published a special issue on neural network applications in chemical engineering in April, 1992 (Venkatasubramanian and McAvoy, 1992).

Stephanopoulos and Han (1994) give a detailed review of the recent developments of intelligent systems in process engineering (ISPE). The Computing and Systems Technology Division of the American Institute of Chemical Engineers (AIChE) and the American Association of Artificial Intelligence (AAAI) are sponsoring an international conference on intelligent systems in process engineering in Snowmass, CO, July 9-14, 1995. The proceedings of this conference will be published by the CACHE Corporation, Austin, TX.

For reports on emerging commercial and research applications of AI-based technologies, the reader may refer to the "Applied AI News" section of the quarterly *AI Magazine*, written by David Blanchard, and to the "Chementator" section of the monthly *Chemical Engineering Magazine*, edited by Gulum (Sam) Samdani.

AIWARE, Inc., "CAD/Chem Custom Formulation System Technical Overview," Cleveland, OH (1992).

Anonymous, "NeuroForecaster for Windows and Macintosh," *Expert Systems*, **10**, 205 (1993a).

Anonymous, "NeuCOP: A Neural Network-Based Control and Optimization Package," *Expert Systems*, **10**, 267 (1993b).

Anonymous, "IRS Chooses Neural Networks to Process Tax Return," *Expert Systems*, **11**, 265 (1994a).

Anonymous, "Neural Network Controller," *Chem. Eng.*, **101**, No.8, 48, August (1994b).

Barr, A. and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, Volume I, Addison-Wesley, Reading, MA (1981).

Bhagat, P., "An Introduction to Neural Nets," *Chem. Eng. Prog.*, **86**, No. 8, 55, August (1990).

Blaesi, J. and B. Jensen, "Can Neural Networks Compete with Process Calculations," *Intech*, p.34, December (1992).

Blanchard, D., "Applied AI News," *AI Magazine*, **14**, No.3, 107, Fall (1993a).

Blanchard, D., "Applied AI News," *AI Magazine*, **14**, No.4, 85, Winter (1993b).

Blanchard, D., "Applied AI News," *AI Magazine*, **15**, No.2, 92, Summer (1994a).

Blanchard, D., "Applied AI News," *AI Magazine*, **15**, No.4, 79, Winter (1994b).

Borman, S., "Neural Network Applications in Chemistry Begin to Appear," *Chem. and Eng. News*, **67**, 24, April 25 (1989).

Buchanan, B. G. and E. H. Shortliffe, *Rule-Based Expert Systems*, Addison-Wesley, Reading, MA (1983).

Chementator, "Neural Networks Optimize Chemical Production," *Chem. Eng.*, **97**, No.8, 19, August (1990).

Chementator, "Control Package Boasts Chaos and Fuzzy Logic," *Chem. Eng.*, **99**, No.4, 18, April (1992a).

Chementator, "IR Spectroscopy Improves Steam-Cracking Control," *Chem. Eng.*, **99**, No.11, 17, November (1992b).

Chementator, "A Smart Plastics-Sorting System Boasts High Speed and Accuracy," *Chem. Eng.*, **100**, No.6, 23, June (1993a).

Chementator, "Neural Network Lends Intelligence to Online Process Control," *Chem. Eng.*, **100**, No.9, 23, September (1993b).

Chementator, "Visual Programming Makes Neural Networking a Breeze," *Chem. Eng.*, **100**, No.11, 23, November (1993c).

Chementator, "Electric Noise Smells Its Way to Online Process Control," *Chem. Eng.*, **101**, No.11, 25, November (1994).

Chitra, S. P., "Using Neural Network for Problem Solving," *Chem. Eng. Prog.*, **89**, No. 4, 44, April (1993).

Crowe, E. R. and C.A. Vassiliadis, "Artificial Intelligence: Starting to Realize Its Practical Promise," *Chem. Eng. Prog.*, **91**, No.1, 22, January (1995).

Denmark, K., M. Farren, and B. Hammack, "Turning Production Data into SPC Gold," *Intech*, p.18, December (1993).

Egan, J., "Artificially Intelligent Investing," *U.S. News and World Report*, **73**, March 15 (1993).

Gill, T. and T. Schutt, "Optimizing Product Formulations Using Neural Networks," *Scientific Computing and Automation*, p.19, September (1992).

Graettinger, T. J., N. V. Bhat, and J. S. Buck, "Adaptive Control with NeuCOP, the Neural Control and Optimization Package," *IEEE International Conference on Neural Networks*, November (1994a).

Graettinger, T. J., N. V. Bhat, K. Heckendom, and J. S. Buck, "Model Predictive Control Using Neural Networks," *AIChE National Meeting*, Atlanta, GA, March/April (1994b).

Hall, C., "Neural Net Applications: Ready for Prime Time?", *IEEE Expert*, **7**, No. 6, 2 (1992).

Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, Reading, MA (1990).

Hoskins, J. C. and D. M. Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering," *Comput. Chem. Eng.*, **12**, 881 (1988).

Lippmann, R. P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, **4**, 4 (1989).

Mah, R. S. H., "Neural Nets," Letter to the Editor, *Chem. Eng. Prog.*, **87**, No.1, 6, January (1991).

Mathis, J. F., "Chemical Engineers Meet a Changing World," *Chem. Eng. Prog.*, **83**, No.9, 21, September (1986).

NeuralWare, Inc., *Neural Computing: NeuaralWorks Professional II/PLUS and NeuralWorks Explorer*, NeuralWare, Inc., Technical Publications Group, Pittsburgh, PA, 35 (1991).

NeuralWare, Inc., *Neural Computing: A Technology Handbook for Professional II/PLUS and NeuralWorks Explorer*, 329 pages, Building 4, Suite 227, Penn Center West, Pittsburgh, PA 15276, phone (412) 787-8222 and FAX (412) 787-8220 (1993).

Oishi, K., M. Tominaga, A. Kawato and S.Imayasu, "Analysis of the State Characteristics of Sake Brewing with a Neural Network," *J. Ferm. Bioeng.*, **73**, 153 (1992).

Quantrille, T. E. and Y. A. Liu, *Artificial Intelligence in Chemical Engineering*, Academic Press, San Diego, CA (1991).

Samdani, G., "Neural Nets," *Chem. Eng.*, **97**, No. 8, 37, August (1990).

Samdani, G., "Smart Software," *Chem. Eng.*, **99**, No.3, 30, March (1992).

Samdani, G., "Fuzzy Logic: More Than a Play on Words," *Chem. Eng.*, **100**, No.2, 30, February (1993).

Samdani, G., "Book Review: Better Products Faster - A Practical Guide to Knowledge-Based Systems for Manufacturers," *Chem. Eng.*, **102**, No.1, 10, January (1995).

Schreiner, K., "Neural Networks Join Fight against Breast Cancer," *IEEE Expert*, **9**, No.4, 75 (1994).

Stephanopoulos, G. and C. Han, "Intelligent Systems in Process Engineering: A Review," *4th Intern. Symp. on Process Systems Engineering*, Kyongju, Korea, May/June (1994).

Venkatasubramamian, V. and T. J. McAvoy, Guest Editors, *Neural Network Applications in Chemical Engineering*, Special Issue, *Comput. Chem. Eng.*, **16**, No. 4, 227-423 (1992).

VerDuin, W. H., "Optimizing Combustion with Integrated Neural Networks and AI Technologies," *Control Eng.*, p.38, July (1992).

VerDuin, W. H., and Y. H. Pao, "The Rapid Foundry Tooling System (RFTS): A Cutting Edge Computer-Aided Design System," *Proceedings of the IEEE National Aerospace and Electronics Conference*, Dayton, OH, Vol. 2, p.926 (1993).

VerDuin, W. H., *Better Products Faster: A Practical Guide to Knowledge-Based Systems for Manufacturers*, Irwin Professional Publishing, Burr Ridge, IL and New York, NY, Toll-free phone number 1-800-451-7556 (1995).

Chapter 2

Fundamental and Practical Aspects of Neural Computing

This chapter introduces neural networks. We first discuss what makes up a neural network, move to the fundamental and practical aspects of neural computing, and then discuss aspects of network training (learning). We also illustrate how to develop a neural network using a commercial software package on a personal computer. Finally, we introduce a number of special neural networks that find significant applications in bioprocessing and chemical engineering.

2.1 Introduction to Neural Computing

This section provides an introduction to neural computing. Part of our discussion has been adopted and updated from Quantrille and Liu (1991; pp. 446-466). We begin with the basic network component, the *node or processing element*. After describing the structure of a node, we move on first to the *topology* of neural networks, i.e., how nodes are interconnected, then to training the networks and the different ways neural networks can "learn."

A. Components of a Node

The foundation of a neural network is the *neuron, or node* (sometimes called *neurode*), shown in Figure 2.1. In many scientific and engineering applications, this node is

frequently called a *processing element*, although we use "node" throughout this text. The nodes perform most of the calculations in the neural network.

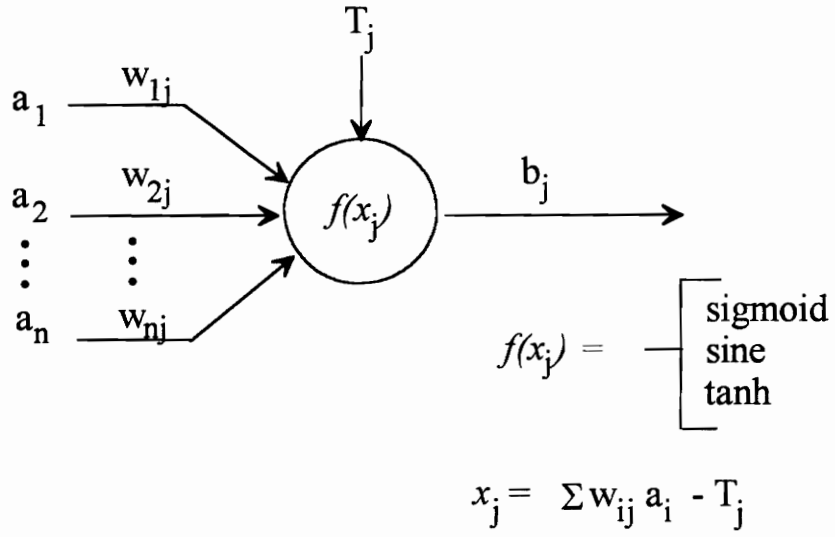


Figure 2.1. The anatomy of the j th node that transfers the input a_i to the jth output b_j through a weight factor w_{ij} and a transfer function $f(x_j)$. T_j is the internal threshold for node j.

1. Inputs and Outputs

The inputs to the jth node are represented as *an input vector*, **a**, with components a_i (i = 1 to n). The node manipulates these inputs, or activities, to give the output, b_j, which can then form the part of the input to other nodes.

2. Weight Factors

What determines the output from a node? Certainly, the values of input vector **a** have an effect. However, some additional factors also affect the output b_j. One is *the weight factor*, w_{ij}, for the i^{th} input, a_i, corresponding to the j^{th} node. Every input is multiplied by its corresponding weight factor, and the node uses this weighted input to perform further calculations. For example, let us consider the node six. The first input into the node is a_1. Multiplying this input by the corresponding weight factor gives $w_{16}a_1$.

Weight factors can have either an inhibitory or an excitatory effect. If we adjust w_{ij} such that $w_{ij}a_i$ is positive (and preferably large), we tend to excite the node. If $w_{ij}a_i$ is negative, it inhibits the node. Finally, if $w_{ij}a_i$ is very small in magnitude relative to other signals, the input signal a_i will have little or no effect.

3. Internal Thresholds

The next important factor governing the output from a node is the internal threshold. The internal threshold for the j^{th} node, denoted T_j, controls activation of that node. The node calculates all its $w_{ij}a_i$'s, sums the terms together, and then calculates the total activation x_j by subtracting the internal threshold value:

$$\text{Total Activation} = x_j = \sum_{i=1} (w_{ij}\, a_i) - T_j \qquad (2.1)$$

If T_j is large and positive, the node has a high internal threshold, which *inhibits* node-firing. Conversely, if T_j is zero (or negative, in some cases), the node has a low internal threshold, which *excites* node-firing.

Some, but not necessarily all, nodes have an internal threshold. If no internal threshold is specified, we assume T_j to be zero.

4. Transfer Functions

The final factor governing a node's output is the transfer function. Once the node calculates the dot product of vector **a** with vector $\mathbf{w_j} = [w_{1j}, w_{2j}, ..., w_{nj}]^T$, and subtracts the threshold T_j (as described above), it passes this result to a transfer function, $f(\ )$. Thus, the complete node calculation is:

$$f(\mathbf{w_j} \cdot \mathbf{a} - T_j) = f\left(\sum_{i=1}^{n} (w_{ij}\, a_i) - T_j\right) \qquad (2.2)$$

This calculation, then, is a function of the *difference* between the input function and the internal threshold.

What functional form do we choose for $f(\ )$? We could choose whatever we want -- square root, log, e^x and so on. Mathematicians and computer scientists, however, have found that the *sigmoid* (S-shaped) function is particularly useful. A typical sigmoid function, shown in Figure 2.2, is:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad (2.3)$$

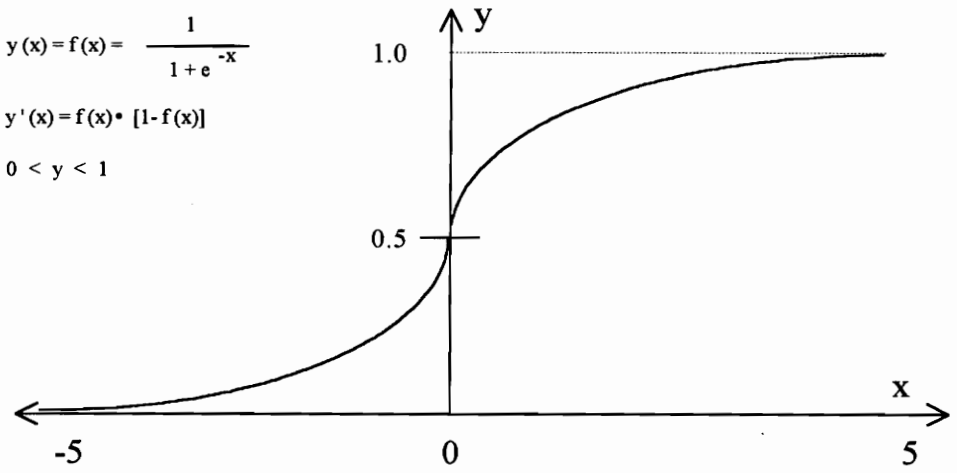


Figure 2.2. A sigmoid (S-shaped) function.

This function is monotonically increasing, with limiting values of 0 (at $x_j = -\infty$) and 1 (at $x_j = +\infty$). All sigmoid functions have upper and lower limiting values. Because of these limiting values, sigmoid functions are called *threshold functions*. At very low input values, the threshold-function output is zero. At very high input values, the output value is one.

Another useful transfer function is the hyperbolic tangent with limiting values of -1 and +1 (Figure 2.3):

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (2.4)$$

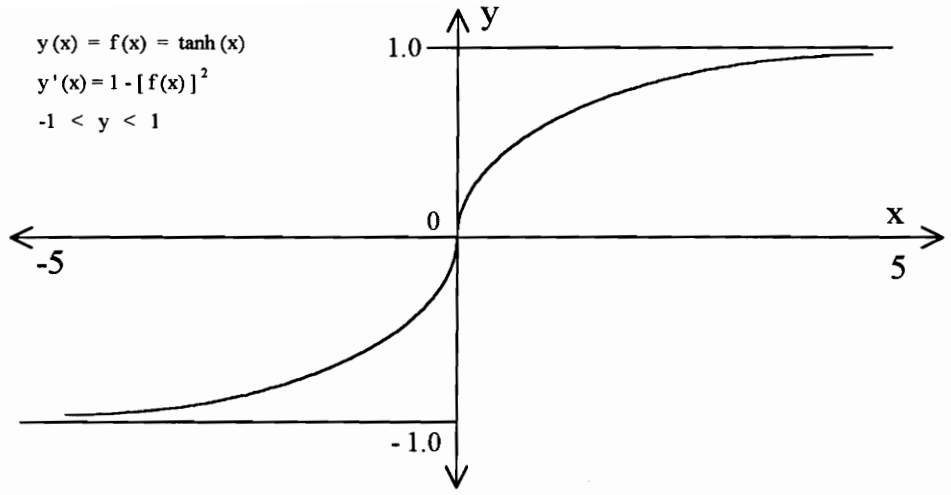


Figure 2.3. A hyperbolic tangent transfer function.

The sigmoid and hyperbolic tangent functions (S-shaped), in general, give fairly well-behaved neural networks. The inhibitory and excitatory effects of the weight factors are straightforward when we use sigmoid functions (i.e., $w_{ij} < 0$ is inhibitory, and $w_{ij} > 0$ is excitatory). Sigmoid functions are continuous and monatonic, and remain finite even as x approaches $\pm\infty$. Because they are monotonic, they also provide for more efficient *training*. We frequently move down the slope of the curve in training, and the sigmoid functions have slopes that are well-behaved as a function of x (this topic will be discussed in gradient-descent learning, section 2.2D).

Another useful transfer function is the radial basis function. The Gaussian transfer function is the most commonly used "radially symmetric" function (see Section 3.2 for

further discussion), and its equation is:

$$f(x) = \exp\left[\frac{-x^2}{2}\right] \tag{2.5}$$

Figure 2. 4 shows a graphical representation of a Gaussian transfer function. The function has maximum response, $f(x) = 1$, when the input is $x = 0$, and the response decreases to $f(x) = 0$ as the input approaches $x = \pm\infty$. This type of response pattern makes the radial basis function very advantageous for certain types of networks, such as those used for classification described in Chapter 3.

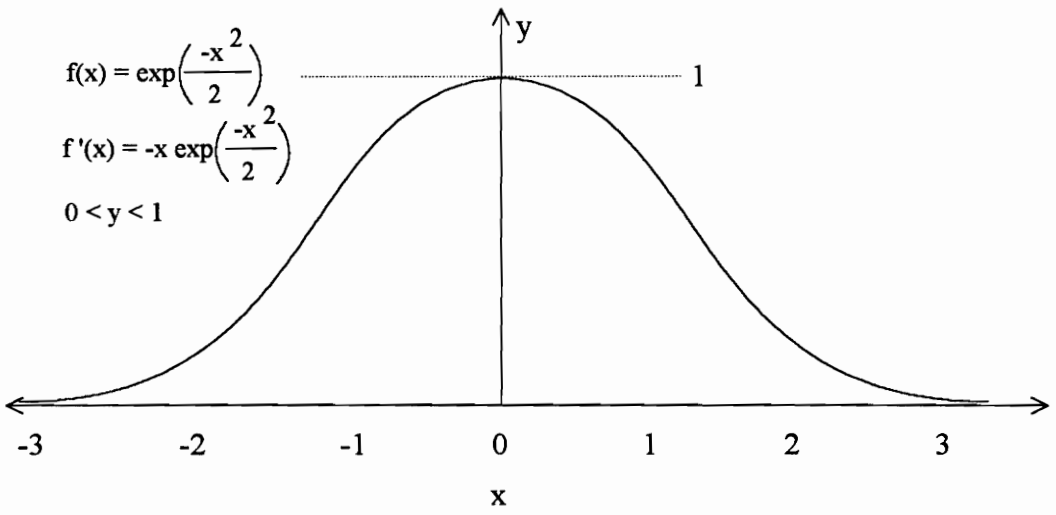


Figure 2.4. A Gaussian transfer function.

5. Summary of Node Anatomy

Figure 2.5 summarizes the basic features of a node. As seen, a node has an n-dimensional input vector, **a**, an internal threshold value, T_j, and n weight factors (w_{1j}, ..., w_{ij}, ..., w_{nj}) which multiply all inputs. If the weighted input is large enough, the node becomes active and performs a calculation based on the difference between the weighted input value and the internal threshold value. Typically, a sigmoid function is used for $f(x)$, since it is a threshold function, is well-behaved, and provides for more rapid training.

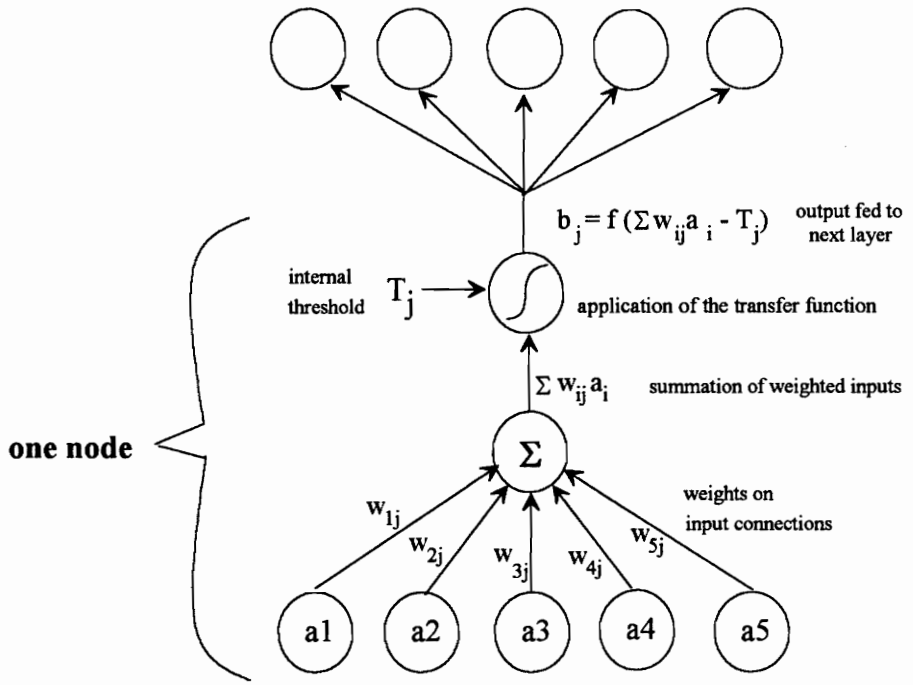


Figure 2.5. Summary of a node anatomy.

B. Topology of a Neural Network

The *topology* of a neural network refers to how its nodes are interconnected. Figure 2.6 shows a very common topology. The network has three layers, one hidden, and each node's output feeds into all nodes in the subsequent layer. We form these topologies, or architectures, by organizing the nodes into layers, connecting them, and weighting the interconnections.

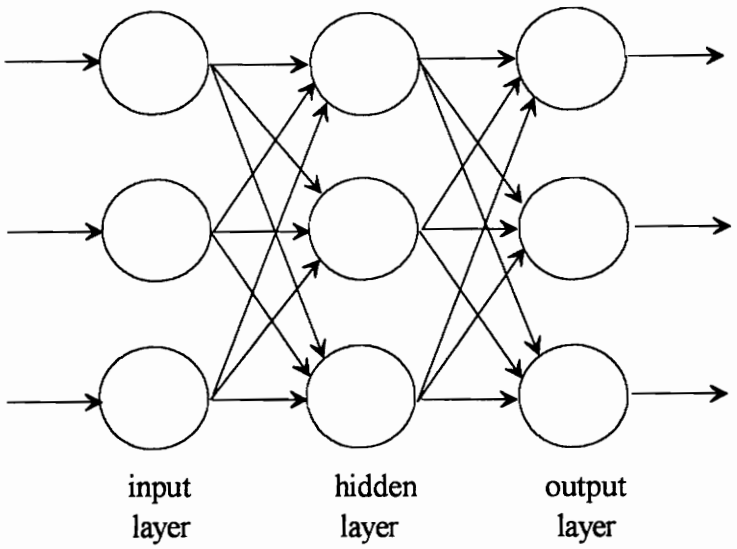


Figure 2.6. A neural network with one hidden layer.

1. Inhibitory or Excitatory Connections

As mentioned, weight factors can either inhibit or excite the node. If the weight is positive, it will excite the node, increasing the activation of the node. If the weight is negative, it will inhibit the node, decreasing the activation. If the weighted signal is highly inhibitory, it may lower the input below the threshold level and shut the node down.

2. Connection Options

We have three options for connecting nodes to one another, as shown in Figure 2.7. In *intralayer* connections, the outputs from a node feed into other nodes in the same layer. In *interlayer* connections, the outputs from a node in one layer feed into nodes in another layer. Finally, in *recurrent* connections, the output from a node feeds into itself.

Generally, when we first build a neural network, *we pre-specify the topology*. That is, we specify the interconnections, but leave the numerical values of the weights up to the training phase. In engineering applications, the most important, and hence the most frequently specified, topology is the interlayer connection.

Within the interlayer topology, we have two options: 1) *feedback connections*, and 2) *feedforward connections*, shown in Figure 2.8.

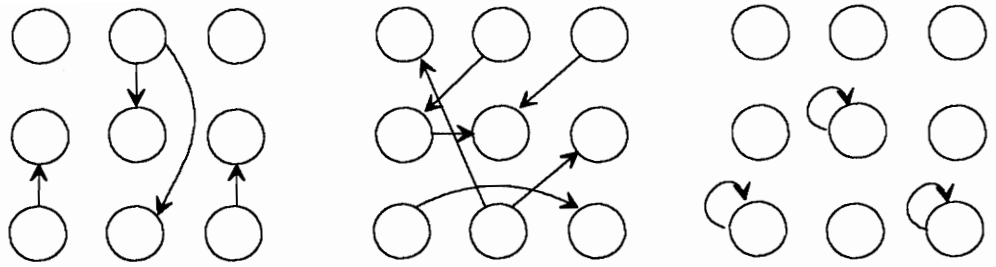


Figure 2.7. The connection options in a neural network.

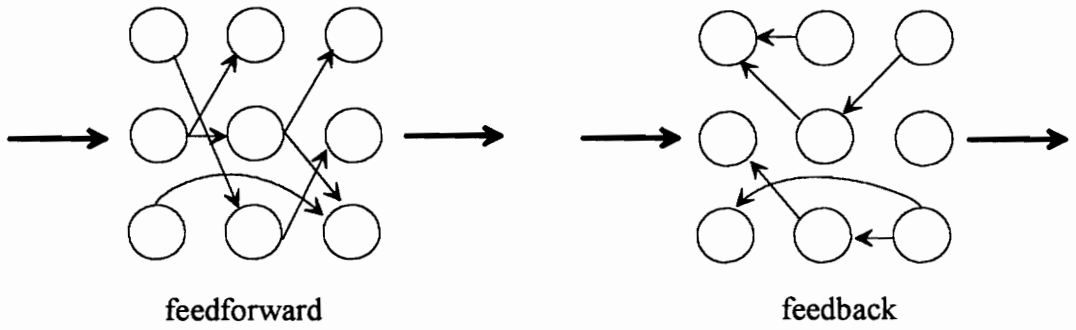


Figure 2.8. The feedback and feedforward connections.

The type of problem we are trying to solve determines which topology we favor. For example, if we wish to develop a neural network that trains itself, we would probably use feedback connections. In contrast, in dynamic modeling of a chemical reactor, where we are trying to map an output response based on an input signal, we would favor the feedforward connection.

c. Multiple Hidden Layers

Most neural networks contain one to three hidden layers. The connections are similar to those described in the previous sections (using only one hidden layer), except that we insert one or two hidden layer(s) between hidden layer 1 and the output layer. Figure 2.9 shows a typical three-hidden-layer feedforward network with connections between only adjacent layers. Most networks used in bioprocessing and chemical engineering follow this basic architecture.

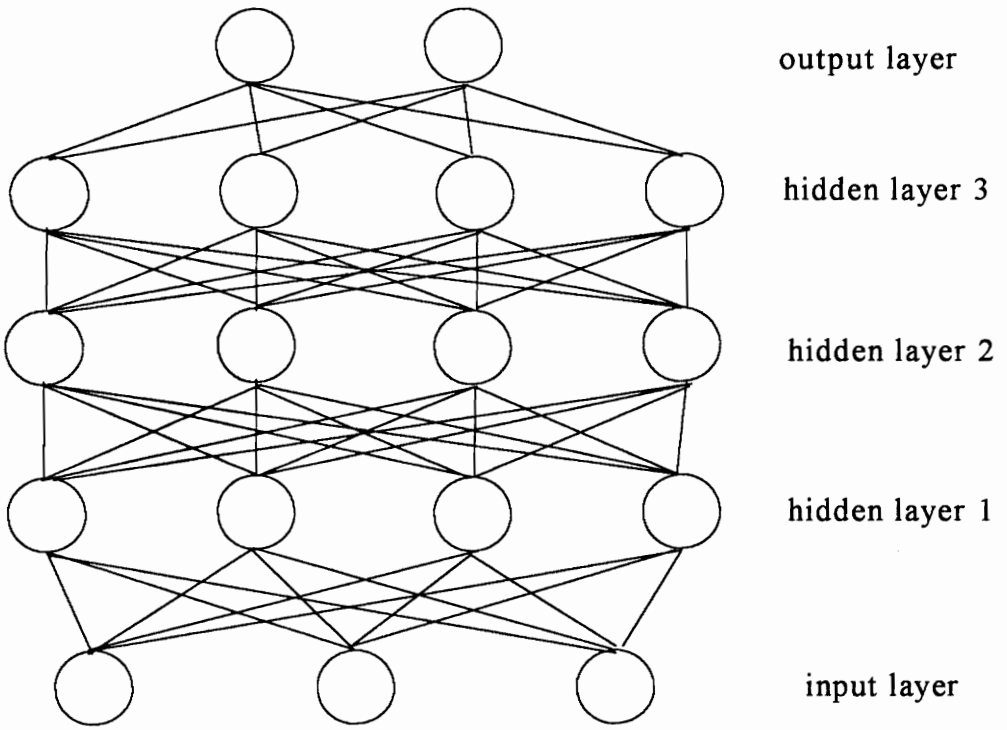


Figure 2.9. A typical three-hidden-layer feedforward network.

Throughout this text, *the hidden-layer configuration* refers to the number of nodes in each hidden layer. For example, a 30:20:10 hidden-layer configuration designates that there are 30 nodes in hidden layer 1, 20 nodes in hidden layer 2, and 10 nodes in hidden layer 3. If there are only two hidden layers, the third term is eliminated (e.g., 30:20).

C. Introduction to Learning and Training with Neural Networks

There are few rigorous mathematical theories available for training neural networks. Most approaches are based on feedforward networks, since they are the most applicable to science and engineering, the least complicated, and the most straightforward to implement.

In general, to *train* neural networks, we adjust the weight factors until the output pattern (response) calculated from the given input reflects the desired cause-and-effect relationship. *Learning* is the actual process of adjusting weight factors based on trial-and-error. To facilitate learning, neural networks have many "buttons and knobs" (i.e., network training parameters that control the adjustment of weight factors) to turn. There are many weight factors to adjust, and the saying goes, "given enough parameters, you can fit an elephant's back" - that is, in this case, with enough weight factors, sooner or later we can produce the input/output relationship we want. The challenge, however, is to systematically adjust the weight factors to give *accurate* results in an *efficient* manner.

1. Stability and Convergence

The training phase needs to produce a neural network that is both *stable* and *convergent*. A globally *stable* neural network maps any set of inputs to a fixed output. Stability guarantees a result, but it does not necessarily guarantee an accurate result.

A *convergent* neural network on the other hand, produces *accurate* input-output relations. Convergence, then, is related to network accuracy. The magnitude of the error between real-world results and network predictions is a direct measurement of the neural network's convergence.

2. Types of Learning

There are many different approaches to training neural networks. Simpson (1990), in his book *Artificial Neural Systems*, classifies a number of methods. Most fall into one of two groups:

- *Supervised learning* - An external teacher controls the learning and incorporates global information.
- *Unsupervised learning* - No external teacher is used and instead the neural network relies upon both internal control and local information. Frequently, the network develops its own models without additional input information.

The primary training method, and the one we use throughout this text is *Error-Correction Learning.* It is a form of supervised learning, where we adjust weights in proportion to the output error vector, ε. This output error vector has n components, where n is the number of nodes on the output layer. We denote the k^{th} component of the vector as ε_k, and thus obtain an error component for each output from the neural network.

We begin error-correction learning by defining the output error from a single node on the output layer as:

$$\varepsilon_k = d_k - c_k \tag{2.6}$$

where ε_k is the output error, d_k is the desired output, and c_k is the calculated output, for the k^{th} node on the *output layer only*. We then calculate the total squared error on the output layer, *E*, as:

$$E = \sum_k \varepsilon_k^2 = \sum_k (d_k - c_k)^2 \tag{2.7}$$

Knowing *E*, we can calculate the change in the weight factor for the i^{th} connection to the j^{th} node, w_{ij}:

$$\Delta w_{ij} = \eta_j a_i E \tag{2.8}$$

η_j is a linear proportionality constant for node j, called *the learning rate* (typically, $0 < \eta_j << 1$), and a_i is the i^{th} input to node j.

There are, however, a number of other approaches, as described below:

(1) *Reinforcement Learning*- a type of supervised learning that is closely related to error-correction learning. In error-correction learning, we calculate an *output error vector*, $\boldsymbol{\varepsilon}$, with components according to equation (2.6). In contrast, reinforcement learning has only one *scaler* output error value to represent to total performance of the neural network. Since we have only one error value to reckon with, reinforcement learning is generally simpler and easier than classical error-correction learning. Reinforcement learning is "selectively supervised," and requires less information, possibly at infrequent intervals.

(2) *Stochastic Learning*- utilizes statistics, probability, and/or random processes to adjust connection weights. We accept a random weight change if it reduces the output error vector, $\boldsymbol{\varepsilon}$. If the change increases $\boldsymbol{\varepsilon}$, we generally reject the change. However, we may accept this change if, according to a specifically encoded probability analysis, it has a better-than-average probability of moving us to the global minimum in error. Accepting seemingly poorer weights allows stochastic processes to escape local minima and move to the global minimum.

(3) *Hardwired Neural Networks*- have all connections and weights predetermined (hardwired). They have a speed advantage, and are used with additional *a priori* information in speech recognition, language processing, vision, and robotics.

(4) *Hebbian Learning* (named after Donald Hebb, 1949)- adjusts weights based on a correlation between the two nodes that the weight factor is associated with. The simplest form of Hebbian learning uses direct proportionality. With node a_i in one layer connected to node b_j in another layer, we adjust the weight factor w_{ij} according to the equation:

$$w_{ij,new} = w_{ij} + \eta_j a_i b_j \qquad (2.9)$$

where η_j is a learning rate for node j, and $0 < \eta_j < 1$.

3. Checking the Performance of the Neural Network

One of the most important aspects in developing neural networks is determining how well the network performs at the completion of the training. Checking the performance of a trained network involves two main steps: (1) how well does the neural network recall the predicted responses (output vector) from data sets used to train the network (called *the recall step*); and (2) how well does the network predict responses from data sets that were not used in training (called *the generalization step*).

In the recall step, we evaluate the network's performance in *recalling (retrieving)* some specific, initial input pattern used in training. Thus, we enter a previously used input pattern into the trained network, and assess the output error from the desired input-output

response. A well-trained network should be able to produce an output with very little error from the desired output.

The next step is the *generalization* phase where we feed new input patterns to the trained network. We say that the network generalizes well when it sensibly interpolates input patterns that are new to the network. As an example, consider a network that has been trained using input patterns I_1 to I_5, as shown in Figure 2.10. This figure gives good and bad generalization examples at new data points between the training data points. A well-trained network should provides input-output mapping with good generalization capability.

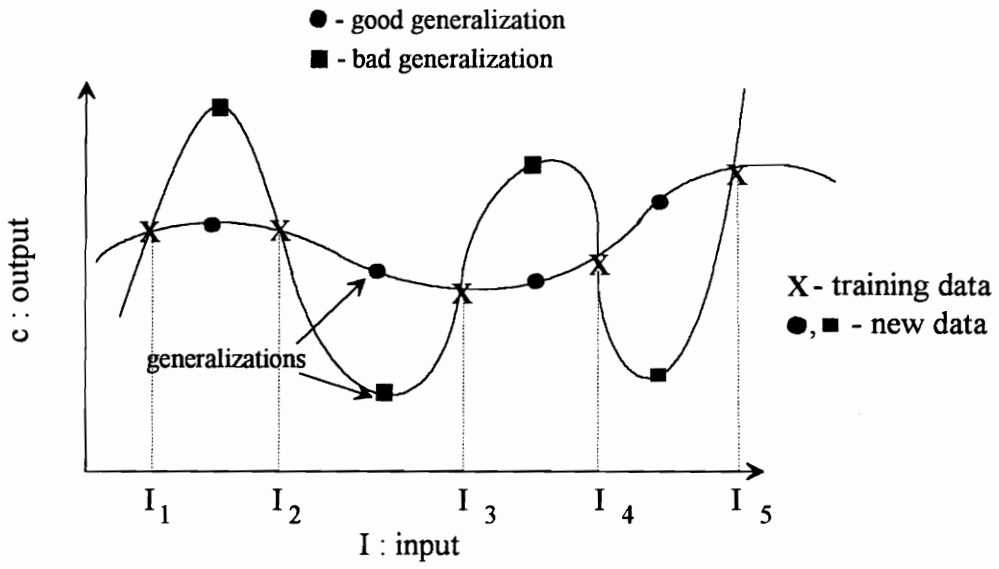


Figure 2.10. An illustration of good and bad generalizations by a trained neural network.

A good method for visualizing how well a network performs recall and generalization is to generate a *learning curve*. Figure 2.11 illustrates a typical learning curve for a well-trained network. In this figure, we plot the average error for both recall of training data sets and the generalization of testing data sets as a function of the number of examples in the training data set.

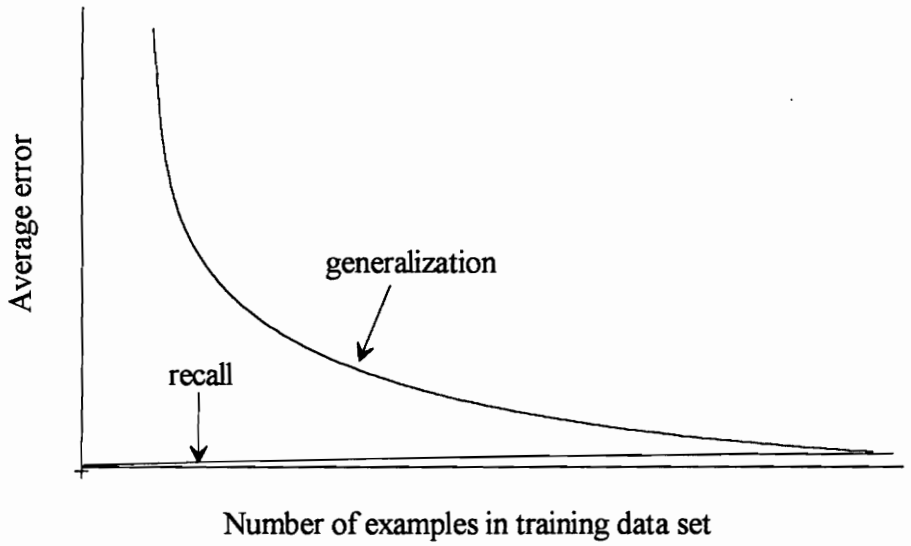


Figure 2.11. A typical learning curve for a well-trained neural network.

As seen in this figure, the network typically shows low average errors for recall with small training data sets, and the recall error increases slightly as more examples are included. In comparison, the generalization is very poor (high average error) for small

training data sets and the error decreases as more examples are included. In other words, as the network receives more information (through training examples), it has a more difficult time retaining all the information, but does a better job predicting future occurrences. For a well-trained network, as it receives more information, the recall and generalization curves approach each other. We will describe how to generate a learning curve in Section 2.3.F.

2.2 Fundamentals of Backpropagation Learning

As mentioned, the most common form of learning utilized in neural networks today is error-correction learning. Previously, mathematicians and computer scientists looked down upon error-correction learning, primarily because the technique did not work on neural networks with hidden layers. It applied only to two-layer neural networks because we could not define how much error came from the hidden nodes, and hence could not determine the weight factors for those nodes. Through a technique known as *backpropagation*, however, we can now apply error-correction learning to neural networks with hidden layers.

Another challenge in error-correction learning is determining the value of the learning rate, η_j. Historically, values for η_j have been restricted such that $0 < \eta_j << 1$, and some researchers have identified values for η_j that yield stable solutions. We will

explain learning rates more fully in Section 2.3.D, where we discuss the practical aspects of neural computing (selection of learning rate, transfer function, weight factors, number of hidden layers, etc.). First, however, let us explain backpropagation and discuss how to determine values for the weight factors, w_{ij}.

A. Requirements for Backpropagation Learning

Backpropagation requires a *perceptron* neural network, defined as a network with only *feedforward interlayer connections*, and no intralayer or recurrent connections. Each layer must feed sequentially into the next layer, with no feedback connections. Theories do exist for backpropagation on neural networks with multiple hidden layers, connections that skip over layers, recurrent connections, and even feedback connections (Simpson, 1990). However, for simplicity, we shall investigate only the three-layer, sequential perceptron shown in Figure 2.12. This network has three layers, A, B, and C. Feeding into layer A is the input vector **I**. Thus layer A has *L* nodes, a_i (i = 1 to L), one node for each input parameter. Layer B, the hidden layer, has m nodes, b_j (j = 1 to m). Note that in the drawing, *L* = m = 3; in practice, though, L and m need not be equal. Each layer may have a different number of nodes. Layer C, the output layer, has n nodes, c_k (k = 1 to n), again with one node for each output parameter. The interconnecting weight between the i^{th} node of layer A and the j^{th} node of layer B is denoted as v_{ij}, and that between the j^{th} node of layer B and the k^{th} node of layer C is w_{jk}. Finally, each node has an internal threshold

value. For layer A, the threshold is T_{Ai}, for layer B, T_{Bj}, and for layer C, T_{Ck}.

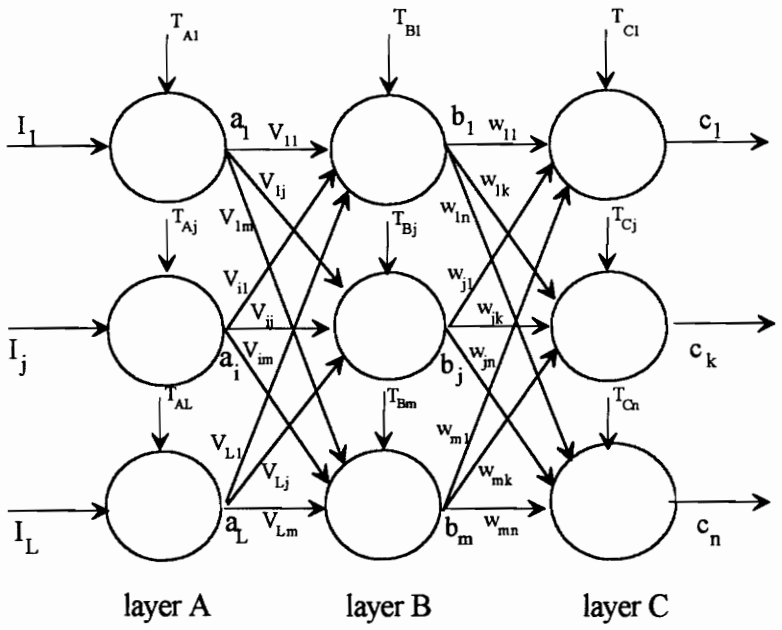


Figure 2.12. A three-layer perceptron neural network.

B. An Illustrative Example : Fault Diagnosis

To illustrate the basic concepts of backpropagation learning, let us use the three-layer, feedforward perceptron neural network shown in Figure 2.12 for fault diagnosis of process data from a chemical reactor. Table 2.1 lists the input and output vectors for the network.

Table 2.1. Input and output for fault-diagnosis network.

Input vector	Output vector
I_1 : reactor inlet temperature, °F	c_1 : low conversion
I_2 : reactor inlet pressure, psia	c_2 : low catalyst selectivity
I_3 : feed flow rate, lb/min	c_3 : catalyst sintering

The *desired* output, d_k, from the neural network is Boolean: 0 indicates that no operational fault exists, and 1 indicates that a fault does exist. The *actual* output from the neural network is a numeric value between 0 and 1, and can be viewed almost as the "probability" that a given input will produce the operational fault (0 = the fault definitely will not occur; 1 = the fault definitely will occur).

Given the above inputs and outputs, we will now illustrate how to use backpropagation to train the network to recognize the following specific conditions:

Input:	$I_1 = 300/1000$ °F $= 0.3$	*Desired Output:*	$d_1 = 1$ (low conversion)
	$I_2 = 100/1000$ psia $= 0.1$		$d_2 = 0$ (no problem)
	$I_3 = 200/1000$ lb/min $= 0.2$		$d_3 = 0$ (no problem)

Note that we have divided all the input values by 1000. In neural network training, we recommend *normalizing* the input and output values to a finite range, such as [0,1] or [-1,1]. In our problem, we divide the input values by their corresponding maximum values

to accomplish this. The need to normalize the data becomes clear in step 2 below.

C. Vanilla Backpropagation Algorithm and Its Application to Fault Diagnosis

Backpropagation, as noted, is a form of error-correction learning, and hence attempts to properly map given inputs with desired outputs by minimizing an error function. Typically, we use the sum-of-squares error. The component of the output error vector from the k^{th} node on the output layer, ε_k, is defined as: $\varepsilon_k = d_k - c_k$, where d_k is the desired output value and c_k is the calculated value. The total mean-square error function, *E,* is:

$$E = \sum_k \varepsilon_k^2 = \sum_k (d_k - c_k)^2 \qquad (2.10)$$

We adjust both interconnecting weights, v_{ij}'s and w_{jk}'s, shown in Figure 2.10 to minimize *E*. Below is the step-by-step adjustment procedure known as the *vanilla backpropagation algorithm* (Simpson, 1990). To use this algorithm, we must have sets of data that map the input vector **I** with the output c_k.

1. Vanilla Backpropagation Algorithm and Illustration

Step 1: Randomly specify numerical values for all weight factors (v_{ij}'s and w_{jk}'s) within the interval [-1, +1]. Likewise, assign internal threshold values (T_{Ai}, T_{Bj}, T_{Ck}) for

every node, also between -1 and +1. Note that at this point, we are only considering how the algorithm operates, hence we will simply assign initial values to the weight factors and internal thresholds. Section 2.3.C will explain how to select these values in more detail. Also, i = 1, 2, ..., L, where L is the number of nodes in layer A; j = 1, 2, ..., m, where m is the number of nodes in layer B; and k = 1, 2, ..., n, where n is the number of nodes in layer C.

Example: To use a computer, let layer A = layer 1, layer B = layer 2, and layer C = layer 3. We randomly assign values for v_{ij} and w_{jk}, within the interval [-1, +1]. These values are:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \qquad [w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

For the internal threshold values (T_{1i}, T_{2j}, T_{3k}), we set $T_{1i} = 0$, and randomly assign values between -1 and 1 to T_{2j} and T_{3k}. All internal thresholds are represented by the matrix:

$$[T_{ij}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & -0.5 \\ 0.0 & 0.5 & -0.5 \end{bmatrix}$$

Step 2: Introduce the input I_i into the neural network of Figure 2.10. Calculate all outputs from the first layer, using the standard sigmoid function introduced previously:

$$x_i = I_i - T_{Ai} \tag{2.11}$$

$$f(x_i) = \frac{1}{1 + e^{-x_i}} = a_i \tag{2.12}$$

Here, I_i is the *normalized* input into the i^{th} node on the input layer, T_{Ai} is internal threshold for the node, x_i is the total activation of the i^{th} node, and a_i is the output from the node.

Example : We introduce the input vector into the neural network and calculate outputs from layer 1:

$$x_1 = I_1 - T_{11} = 0.3 - 0 = 0.3$$

$$x_2 = I_2 - T_{12} = 0.1 - 0 = 0.1$$

$$x_3 = I_3 - T_{13} = 0.2 - 0 = 0.2$$

Substituting these values into the sigmoid function, we get:

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 0.57444$$

$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 0.52498$$

$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 0.54983$$

Note that had we *not* normalized our input values to the range [0,1], we would have the following results:

$$x_1 = I_1 - T_{11} = 300 - 0 = 300$$

$$x_2 = I_2 - T_{12} = 100 - 0 = 100$$

$$x_3 = I_3 - T_{13} = 200 - 0 = 200$$

and

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 1.0$$

$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 1.0$$

$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 1.0$$

Without normalizing the input values, I_1 to I_3, we would have a potential problem. If $I_1 = 300$ °F, then $a_1 = 1$. If $I_1 = 150$ °F, then $a_1 = 1$ again. That is,

the sigmoid transfer function is *insensitive* to changes in the input when $I_1 > 2$. This example clearly illustrates why we recommend normalizing the input values to a finite range, [0,1] or [-1,1]; within these ranges, the selected transfer function varies appreciably with changes in input and provides a much more *sensitive* response.

Step 3: Given the output from layer A, calculate the output from layer B, using the equation:

$$b_j = f\left(\sum_{i=1}^{L} (v_{ij}\, a_i) - T_{Bj} \right) \qquad (2.13)$$

where $f(\,)$ is the same sigmoid function.

Example: We calculate the output from each node in layer 2:

$$\begin{aligned} b_1 &= f(v_{11}a_1 + v_{21}a_2 + v_{31}a_3 - T_{21}) \\ &= f[(-1.0)(0.57444) + (1.0)(0.52498) + (0.5)(0.54983) - 0.5] \\ &= f(-0.27455) \\ &= 0.43179 \end{aligned}$$

$$b_2 = f(v_{12}a_1 + v_{22}a_2 + v_{32}a_3 - T_{22}) = f(-0.56214) = 0.36305$$

$$b_3 = f(v_{13}a_1 + v_{23}a_2 + v_{33}a_3 - T_{23}) = f(0.79965) = 0.68990$$

Step 4: Given the output from layer B, calculate the output from layer C, using the equation:

$$c_k = f\left(\sum_{j=1}^{m} (w_{jk}\, b_j) - T_{ck} \right) \qquad (2.14)$$

where $f( )$ is the same sigmoid function.

Example: We calculate the output from each node in layer 3:

$$c_1 = f(w_{11}b_1 + w_{21}b_2 + w_{31}b_3 - T_{31}) = f(0.27621) = 0.56862$$

$$c_2 = f(w_{12}b_1 + w_{22}b_2 + w_{32}b_3 - T_{32}) = f(-1.06085) = 0.25715$$

$$c_3 = f(w_{13}b_1 + w_{23}b_2 + w_{33}b_3 - T_{33}) = f(1.24237) = 0.77598$$

Steps 1 to 4 represent the *forward activation flow*; that is, the given input values I_1 to I_3 move forward in the network, activate the nodes and produce the actual output values c_1 to c_3, based on the initially assumed values of interconnecting weights, v_{ij} and w_{ij}, and internal thresholds T_{ij}. Obviously, the actual output values ($c_1 = 0.56862$, $c_2 = 0.25715$, and $c_3 = 0.77598$) deviate from the desired output values ($d_1 = 1$, $d_2 = 0$, and $d_3 = 0$).

The next few steps of the backpropagation algorithm represent *the backward error flow* in which we *propagate* the errors between the desired output d_k ($k = 1$ to 3) and the

actual output c_k *backward* through the network, and try to find the best set of network parameters (v_{ij}, w_{ij} and T_{ij}). Traditional training methods attempt to find these network parameters by minimizing the sum of squared errors:

$$E = \sum_k \varepsilon_k^2 = \sum_k (d_k - c_k)^2 \tag{2.15}$$

However, efficient training methods, such as the vanilla backpropagation algorithm, slightly modify the output-error equation, $\varepsilon_k = d_k - c_k$, by introducing the least-square (LMS) output-error equation as described in step 5 below.

Step 5: Now *backpropagate* the error through the network, starting at the output and moving backward toward the input. Calculate the k^{th} component of the output error, ε_k, for each node in layer C, according to the equation:

$$\varepsilon_k = c_k (1 - c_k) (d_k - c_k) \tag{2.16}$$

where d_k is the desired output and c_k is the actual output. We call the equation for ε_k the *least-mean-square (LMS)* output-error equation. Essentially, the multiplier $c_k (1 - c_k)$ is a characteristic term of the so-called gradient-descent learning technique involving the sigmoid transfer function. We use Newton's method (moving down a gradient on a surface) to minimize the LMS error.

The advantage of this method is that weight changes are estimated systematically rather than arbitrarily. The sigmoid function of the output is:

$$f(x_k) = \frac{1}{1+e^{-x_k}} = c_k \tag{2.17}$$

Therefore, the partial derivative of the sigmoid function is:

$$\frac{\partial f}{\partial x_k} = \frac{e^{-x_k}}{(1+e^{-x_k})^2} \tag{2.18}$$

We can rewrite this equation as:

$$\frac{\partial f}{\partial x_k} = \frac{1}{(1+e^{-x_k})}\left(1 - \frac{1}{1+e^{-x_k}}\right) \tag{2.19}$$

and therefore,

$$\frac{\partial f}{\partial x_k} = c_k\,(1 - c_k) \tag{2.20}$$

Thus, we see that the term $c_k(1-c_k)$ in Step 5 of the algorithm is actually the gradient for Newton's method (i.e., the partial derivative with respect to x_k).

Example: We backpropagate, first calculating the error for each node in layer 3:

$$\varepsilon_1 = c_1 (1 - c_1) (d_1 - c_1) = 0.10581$$

$$\varepsilon_2 = c_2 (1 - c_2) (d_2 - c_2) = -0.04912$$

$$\varepsilon_3 = c_3 (1 - c_3) (d_3 - c_3) = -0.13489$$

Step 6: Continue backpropagation, moving to layer B. Calculate the j^{th} component of the error vector, e_j, of layer B relative to each ε_k, using the equation:

$$e_j = b_j (1 - b_j) \left(\sum_{k=1}^{n} (w_{jk} \varepsilon_k) \right) \tag{2.21}$$

Example: We calculate the errors associated with layer 2, the hidden layer:

$$e_1 = b_1 (1 - b_1) (w_{11} \varepsilon_1 + w_{12} \varepsilon_2 + w_{13} \varepsilon_3) = -0.03648$$

$$e_2 = b_2 (1 - b_2) (w_{21} \varepsilon_1 + w_{22} \varepsilon_2 + w_{23} \varepsilon_3) = 0.008872$$

$$e_3 = b_3 (1 - b_3) (w_{31} \varepsilon_1 + w_{32} \varepsilon_2 + w_{33} \varepsilon_3) = 0.002144$$

Step 7: Adjust weight factors, calculating the new w_{jk}, i.e., $w_{jk,new}$, as:

$$w_{jk,new} = w_{jk} + \eta_c b_j \varepsilon_k \tag{2.22}$$

or

$$w_{jk,new} = w_{jk} + \eta_c b_j c_k (1 - c_k)(d_k - c_k) \tag{2.23}$$

for j = 1 to m and k = 1 to n. The term η_C is a positive constant controlling the learning rate in layer C. Thus, the new weight factors are calculated from the old weight factors from the previous training iteration by the following general expression:

$$\begin{bmatrix}\text{new weight}\\ \text{factor}\end{bmatrix} = \begin{bmatrix}\text{old weight}\\ \text{factor}\end{bmatrix} + \begin{bmatrix}\text{learning}\\ \text{rate}\end{bmatrix} \times \begin{bmatrix}\text{input}\\ \text{term}\end{bmatrix} \times \begin{bmatrix}\text{gradient-descent}\\ \text{correction term}\end{bmatrix} \qquad (2.24)$$

We call this weight-correction scheme the *LMS (least-mean-square) rule.*

Example: We adjust weight factors for interconnections between layers 2 and 3. For simplicity, we assume that $\eta = 0.7$ for *all* values of η. Thus, the learning rate, η_j, equals 0.7. Similarly the rates to calculate internal threshold values, i.e., η_2 and η_3, also equal 0.7. We adjust the weight factors w_{1k} (k = 1 to 3) as follows:

$$w_{11,\,new} = w_{11} + \eta\, b_1\, \varepsilon_1 = -0.9680$$

$$w_{12,\,new} = w_{12} + \eta\, b_1\, \varepsilon_2 = -0.5149$$

$$w_{13,\,new} = w_{13} + \eta\, b_1\, \varepsilon_3 = 0.4593$$

We continue these adjustments for the rest of the w_{jk}'s. A comparison of the old and new weight factors shows:

$$[w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \qquad [w_{jk,new}] = \begin{bmatrix} -0.9680 & -0.5149 & 0.4593 \\ 1.0269 & -0.0125 & 0.4657 \\ 0.5511 & -0.5237 & 0.4349 \end{bmatrix}$$

Step 8: Adjust the thresholds T_{ck} (k = 1 to n) in layer C, according to the equation:

$$T_{ck,new} = T_{ck} + \eta_c \varepsilon_k \tag{2.25}$$

Example: We adjust the internal thresholds for layer three:

$$T_{31,new} = T_{31} + \eta\varepsilon_1 = 0.0741$$

$$T_{32,new} = T_{32} + \eta\varepsilon_2 = 0.4656$$

$$T_{33,new} = T_{33} + \eta\varepsilon_3 = -0.5944$$

Thus, new and old threshold values are:

Old : $T_{31} = 0.0$ $T_{32} = 0.5$ $T_{33} = -0.5$

New : $T_{31} = 0.0741$ $T_{32} = 0.4656$ $T_{33} = -0.5944$

Step 9: Adjust weight factors v_{ij}, according to the equation:

$$v_{ij,new} = v_{ij} + \eta_B a_i e_j \tag{2.26}$$

for i = 1 to L and j = 1 to m. The term η_B is a positive constant controlling the learning rate in layer B. Note that this equation follows the same form as equation (2.22) used to calculate the weight factors for layer C.

Example: We adjust the weight factors, v_{ij}, for interconnections between layers 1 and 2. Again, $\eta = 0.7$.

$$v_{11,new} = v_{11} + \eta a_1 e_1 = -1.0147$$

$$v_{12,new} = v_{12} + \eta a_1 e_2 = -0.4964$$

$$v_{13,new} = v_{13} + \eta a_1 e_3 = 0.5009$$

We continue these adjustments for the rest of the v_{ij}'s, and the old and new weight factors are:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [v_{ij,new}] = \begin{bmatrix} -1.0147 & -0.4964 & 0.5009 \\ 0.9866 & 0.0033 & -0.4992 \\ 0.4860 & -0.4966 & 0.5008 \end{bmatrix}$$

Step 10: Adjust the thresholds T_{Bj} (j = 1 to m) in layer B, according to the equation:

$$T_{Bj,new} = T_{Bj} + \eta_B e_j \qquad (2.27)$$

Example: We adjust the internal thresholds for layer three:

$$T_{21,new} = T_{21} + \eta e_1 = 0.4745$$

$$T_{22,new} = T_{22} + \eta e_2 = 0.0062$$

$$T_{23,new} = T_{23} + \eta e_3 = -0.4985$$

Thus, new and old internal threshold values are:

Old	: $T_{21} = 0.5$	$T_{22} = 0.0$	$T_{23} = -0.5$
New	: $T_{21} = 0.4745$	$T_{22} = 0.0062$	$T_{23} = -0.4985$

Step 11: Repeat steps 2-10 until the squared error, E, or the output-error vector, ε, is zero or sufficiently small.

Example: Now, we go back to step 2 and repeat the procedure until we converge on the correct values. Programming this backpropagation algorithm is relatively straight forward, and Appendix 2A lists a simple 112-line Basic to perform the

calculations. This problem requires 3860 time steps to get results less than 2% error on variable d_k.

Number of time steps: 3860

Desired values:	$d_1 = 1$	$d_2 = 0$	$d_3 = 0$
Actual values:	$c_1 = 0.9900$	$c_2 = 0.0156$	$c_3 = 0.0098$
Percent error:	$e_1 = 1.00\%$	$e_2 = 1.56\%$	$e_3 = 0.98\%$

2. Limitations of the Vanilla Backpropagation Algorithm

The vanilla backpropagation algorithm requires a few comments. First, we do *not* adjust the internal threshold values for layer A, T_{Ai}'s. Therefore, depending on the problem being solved, we may wish to set all T_{Ai}'s equal to zero. Second, using the sigmoid function restricts the output to values between zero and one, which can cause some problems when using the neural network for empirical modeling. If we desire values outside the [0,1] interval requires some type of "normalized" output value.

A third potential problem in using the neural network architecture shown in Figure 2.12, is the presence of the internal threshold values, T_{Ai}, T_{Bj}, T_{Ck}. As these threshold values are adjusted during backpropagation, node-activation levels (as determined by weight factors from the previous training iteration) can suddenly be excessively low or

high, depending on the new threshold value. These sudden changes in node activation induce some undesirable consequences. Specifically, these changes introduce discontinuities in the network, which may, in turn, lead to poor network stability. During training, the network may be prone to more oscillations.

Because of these difficulties, many neural networks do not use internal threshold values (i.e., the T_{Ai}'s, T_{Bj}'s, and T_{Ck}'s are all set to zero). Instead, the sigmoid function itself acts as a "gradual threshold," with an S-shaped curve that eliminates the discontinuities caused by normal internal thresholds. The sigmoid function has a limiting value of zero at low activation and one at high activation, allowing it to activate and deactivate the node. When sigmoid functions are used in this way, they are called *sigmoid threshold functions*.

Despite these modifications, though, one of the biggest challenges for neural networks is the training: it is often long and tedious (>50,000 iterations). Section 2.2.D introduces a much more efficient training technique called the *generalized delta-rule algorithm* (or simply *delta-rule algorithm*), and we present this algorithm for network training with *multiple* data points (input patterns). Section 2.3 focuses more closely on several practical aspects of neural computing to facilitate network training, emphasizing how to select network parameters such as weight factors, transfer function, learning rate, etc.

D. Generalized Delta-Rule (Delta-Rule) Algorithm and Its Application to Fault Diagnosis

Many of the growing number of more sophisticated training algorithms use both the internal threshold and sigmoid threshold functions. One particularly useful example is the delta-rule algorithm, which is a gradient-descent learning technique as illustrated previously in equations (2.17) and (2.20). This algorithm is more efficient than the vanilla backpropagation algorithm, and it incorporates both types of threshold values.

1. Overview of the Delta-Rule Algorithm

As noted, one difficulty with backpropagation algorithms is the extensive time required to train the network. Depending on the size of the neural network, training can take hours or even days depending on speed and capacity of your computer. Researchers have investigated many different training procedures in an attempt to speed up the learning process. One such procedure that has been frequently applied, as we investigated with the vanilla backpropagation algorithm, is gradient-descent learning.

Another form of gradient-descent learning is the *generalized delta rule* (delta rule), an iterative method that minimizes the mean-squares error. It is related to the vanilla backpropagation algorithm, but has several key differences. First, the delta rule uses a technique known as *momentum* to speed up the training. Momentum is an extra

weight added onto the weight factors when they are adjusted. By accelerating the change in the weight factors, we improve the training rate.

To understand the physical meaning of the term "momentum," let us follow the illustration given by Caudill (1990). Suppose that we have been sledding down a hill, as shown in Figure 2.13a.

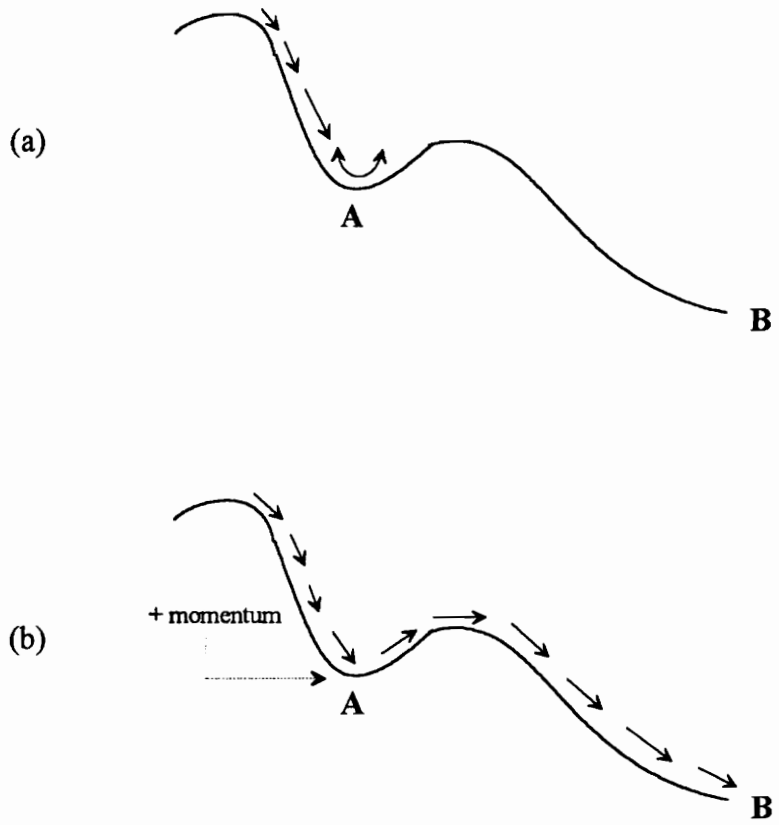


Figure 2.13. An illustration of the gradient-descent (hill-descent) method of reaching a minimum: (a) a pure gradient-descent scheme can only reach the rise (local minimum point A); (b) adding "momentum" to go over the rise and continue moving downward to reach the hill bottom (global minimum point B).

To reach the hill bottom, we always take the steepest descending route downward from the current position. Though we occasionally encounter a small rise in our downward path, we are frequently able to go over that rise and continue downhill. Because we have been sledding downhill long enough and far enough to have built up some momentum, so we tend to keep going downhill unless a really large barrier stops us. Suppose that such a large barrier does exist along the downward path. One way to overcome it is to introduce external momentum to help push us over the rise and eventually reach the hill bottom, as illustrated in Figure 2.13b.

This example is identical to the gradient-descent method for finding the neural network parameters, such as weight factors and internal thresholds, based on minimizing the total mean-squares output error. In certain situations, the *LMS (least-mean-squares) rule* for weight-factor correction, equation (2.24), leads to only a local minimum (similar to the small rise in Figure 2.13a) of the total mean-squares error.

$$\begin{bmatrix}\text{new weight}\\ \text{factor}\end{bmatrix}=\begin{bmatrix}\text{old weight}\\ \text{factor}\end{bmatrix}+\begin{bmatrix}\text{learning}\\ \text{rate}\end{bmatrix}\times\begin{bmatrix}\text{input}\\ \text{term}\end{bmatrix}\times\begin{bmatrix}\text{gradient-descent}\\ \text{correction term}\end{bmatrix} \tag{2.24}$$

To reach the global minimum (or hill bottom) of the total mean-squares error, we need to modify the LMS rule by adding a momentum term to speed up the network training. This process is the so-called *generalized delta rule (delta rule)* for weight-factor correction:

$$\begin{bmatrix}\text{new weight}\\ \text{factor}\end{bmatrix}=\begin{bmatrix}\text{old weight}\\ \text{factor}\end{bmatrix}+\begin{bmatrix}\text{learning}\\ \text{rate}\end{bmatrix}\times\begin{bmatrix}\text{input}\\ \text{term}\end{bmatrix}\times\begin{bmatrix}\text{gradient-descent}\\ \text{correction term}\end{bmatrix}+\begin{bmatrix}\text{momentum}\\ \text{coefficient}\end{bmatrix}\times\begin{bmatrix}\text{previous}\\ \text{weight change}\end{bmatrix} \tag{2.28}$$

Here, the momentum coefficient, α, is restricted such that $0 < \alpha < 1$. The momentum term thus represents a fractional value of the weight change from the previous training iteration.

Another difference between the delta rule and the vanilla backpropagation algorithm is the presence of a *bias function* instead of internal threshold values. The internal thresholds (T_{1i}, T_{2j}, and T_{3k}) become a bias function when we *add* (rather than *subtract* as in the vanilla backpropagation) a fixed number to the nodal summation. In addition, when serving as a bias function, the values of T_{1i}, T_{2j}, and T_{3k} remain *constant* throughout the training process. In the delta rule, we set $T_{1i} = 0$, and $T_{2j} = T_{3k} = 1$ for the entire training of the neural network.

By using momentum coupled with a bias function, the delta-rule algorithm is more efficient than the vanilla backpropagation algorithm. To see the comparison, we will apply the delta rule for the three-layer feedforward perceptron shown in Figure 2.12, to the fault-diagnosis problem of Section 2.2.B.

2. Generalized Delta-Rule (Delta-Rule) Algorithm and Illustration

Step 1: Randomly assign values between 0 and 1 to weights v_{ij} and w_{jk}. For the delta rule, the internal threshold values *must* be assigned as follows: all input-layer thresholds must equal zero, i.e. $T_{1i} = 0$; all hidden- and output-layer thresholds must equal one, i.e., $T_{2j} = T_{3k} = 1$.

Example: We assign the same values for v_{ij} and w_{jk} as we did in the vanilla backpropagation algorithm. For the internal threshold values (T_{1i}, T_{2j}, T_{3k}), we set the threshold values of layer 1 equal to zero, and set those of layers 2 and 3 equal to one:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

$$[T_{ij}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

This T-matrix serves as a bias function.

Step 2: Introduce the input I_i into the neural network, and calculate the output from the first layer according to the equations:

$$x_i = I_i + T_{1i} = I_i + 0 = I_i \tag{2.29}$$

$$a_i = f(x_i) = \frac{1}{1 + e^{-x_i}} \tag{2.30}$$

Note that $f(\ )$ here is again the sigmoid function.

Example: We introduce the input vector into the neural network and calculate the outputs from layer 1:

$$x_1 = I_1 + T_{11} = 0.3 + 0 = 0.3$$

$$x_2 = I_2 + T_{12} = 0.1 + 0 = 0.1$$

$$x_3 = I_3 + T_{13} = 0.2 + 0 = 0.2$$

Substituting these values into the sigmoid function, we get:

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 0.57444$$

$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 0.52498$$

$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 0.54983$$

Step 3: Knowing the output from the first layer, calculate outputs from the second layer, using the equation:

$$b_j = f\left(\sum_{i=1}^{L} (v_{ij}\, a_i) + T_{2j} \right) \qquad (2.31)$$

where $f(\,)$ is the sigmoid function. Note that $T_{2j} = 1$ in this algorithm, and we are *adding* it to the weighted inputs (rather than *subtracting* it as in the vanilla backpropagation algorithm). When used in this mode, T_{2j} acts as a bias

function.

Example: We calculate the output from each node in layer two:

$$b_1 = f(v_{11}a_1 + v_{21}a_2 + v_{31}a_3 + T_{21}) = f(1.22546) = 0.77302$$

$$b_2 = f(v_{12}a_1 + v_{22}a_2 + v_{32}a_3 + T_{22}) = f(0.43786) = 0.60775$$

$$b_3 = f(v_{13}a_1 + v_{23}a_2 + v_{33}a_3 + T_{23}) = f(1.29965) = 0.78578$$

Step 4: Knowing the output from the second layer, calculate the result from the output layer, according to the equation:

$$c_k = f\left(\sum_{j=1}^{m} (w_{jk}\, b_j) + T_{3k} \right) \tag{2.32}$$

where $f(\,)$ is the sigmoid function. Note again, $T_{3k} = 1.0$. T_{3k} acts as a bias function, and we are *adding* it to the sum.

Example: We calculate the output from each node in layer three:

$$c_1 = f(w_{11}b_1 + w_{21}b_2 + w_{31}b_3 + T_{31}) = f(1.22762) = 0.77340$$

$$c_2 = f(w_{12}b_1 + w_{22}b_2 + w_{32}b_3 + T_{32}) = f(0.22060) = 0.55493$$

$$c_3 = f(w_{13}b_1 + w_{23}b_2 + w_{33}b_3 + T_{33}) = f(2.08327) = 0.88927$$

Step 5: Continue steps 1-4 for P number of training patterns presented to the input layer. Calculate the mean-squared error, *E*, according to the following equation:

$$E = \sum_{p=1}^{P} \sum_{k=1}^{n} (d_k^p - c_k^p)^2 \qquad (2.33)$$

where P is the number of training patterns presented to the input layer, n is the number of nodes on the output layer, d_k^p is the desired output value from the k^{th} node in the p^{th} training pattern, and c_k^p is the actual output value from the k^{th} node in the p^{th} training pattern.

Example: We are training the network with just one input pattern (P = 1). With desired output values $d_1 = 1$, $d_2 = 0$, and $d_3 = 0$, our total mean-squared error is:

$$E = \sum_{k=1}^{3} (d_k - c_k)^2 = (d_1 - c_1)^2 + (d_2 - c_2)^2 + (d_3 - c_3)^2 = 1.1501$$

Step 6: Knowing the p^{th} pattern, calculate δ_{3k}^p, the gradient-descent term for the k^{th} node in the *output layer* (layer 3) for training pattern p. Use the following equation:

$$\delta_{3k}^p = (d_k^p - c_k^p) \frac{\partial f}{\partial x_k} \qquad (2.34)$$

where $f(\ )$ is the sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.35}$$

The partial derivative of the sigmoid function is:

$$\frac{\partial f}{\partial x_k} = \frac{e^{-x_k}}{(1+e^{-x_k})^2} \tag{2.36}$$

Note that x_k is the sum of the weighted inputs to the k^{th} node on the output layer plus the bias function (i.e., for the p^{th} training session):

$$x_k^p = \sum_j w_{jk}^p \, b_j^p + T_{3k}^p \tag{2.37}$$

where for training pattern p, b_j^p is the output value of the j^{th} node in the hidden layer and T_{3k}^p is the threshold value on the output layer.

Example: We calculate δ_{3k}, the gradient-descent term for layer 3. To obtain δ_{3k}, we must first calculate the gradients by setting $x_1 = c_1$, $x_2 = c_2$, and $x_3 = c_3$:

$$\frac{\partial f}{\partial x_1} = \frac{e^{-x_1}}{(1+e^{-x_1})^2} = \frac{e^{-0.77340}}{(1+e^{-0.77340})^2} = 0.27193$$

$$\frac{\partial f}{\partial x_2} = \frac{e^{-x_2}}{(1+e^{-x_2})^2} = \frac{e^{-0.55493}}{(1+e^{-0.55493})^2} = 0.23170$$

$$\frac{df}{dx_3} = \frac{e^{-x_3}}{(1+e^{-x_3})^2} = \frac{e^{-0.88927}}{(1+e^{-0.88927})^2} = 0.20643$$

Knowing the gradients, we calculate the δ_{3k}'s:

$$\delta_{31} = (d_1 - c_1) \frac{\partial f}{\partial x_1} = 0.06162$$

$$\delta_{32} = (d_2 - c_2) \frac{\partial f}{\partial x_2} = -0.12858$$

$$\delta_{33} = (d_3 - c_3) \frac{\partial f}{\partial x_3} = -0.18357$$

Step 7: Again knowing the p^{th} pattern, calculate $\delta_{2j}{}^{p}$, the gradient-descent term for the j^{th} node on the *hidden layer* (layer 2). Use the equation:

$$\delta_{2j}^{p} = \left(\sum_{k} \delta_{3k}^{p} w_{jk}^{p} \right) \frac{\partial f}{\partial x_j} \tag{2.38}$$

where the subscript k denotes a node in the output layer. Recall that x_j is defined by:

$$x_j^{p} = \sum_{i} v_{ij}^{p} a_i^{p} + T_{2j}^{p} \tag{2.39}$$

and the partial derivative of the sigmoid function, again, is:

$$\frac{\partial f}{\partial x_j} = \frac{e^{-x_j}}{(1+e^{-x_j})^2} \tag{2.40}$$

Example: We calculate δ_{2j}, the gradient-descent term for layer 2. Again, we must first calculate the gradients by setting $x_1 = b_1$, $x_2 = b_2$, and $x_3 = b_3$:

$$\frac{\partial f}{\partial x_1} = \frac{e^{-x_1}}{(1+e^{-x_1})^2} = \frac{e^{-0.77302}}{(1+e^{-0.77302})^2} = 0.21608$$

$$\frac{\partial f}{\partial x_2} = \frac{e^{-x_2}}{(1+e^{-x_2})^2} = \frac{e^{-0.60775}}{(1+e^{-0.60775})^2} = 0.23512$$

$$\frac{\partial f}{\partial x_3} = \frac{e^{-x_3}}{(1+e^{-x_3})^2} = \frac{e^{-0.78578}}{(1+e^{-0.78578})^2} = 0.21506$$

Then, the δ_{2j}'s are:

$$\delta_{21} = (\delta_{31} w_{11} + \delta_{32} w_{12} + \delta_{33} w_{13}) \frac{\partial f}{\partial x_1} = -0.019257$$

$$\delta_{22} = (\delta_{31} w_{21} + \delta_{32} w_{22} + \delta_{33} w_{23}) \frac{\partial f}{\partial x_2} = -0.070936$$

$$\delta_{23} = (\delta_{31} w_{31} + \delta_{32} w_{32} + \delta_{33} w_{33}) \frac{\partial f}{\partial x_3} = -0.026941$$

Step 8: Knowing $\delta_{2j}^{\,p}$ for the hidden layer and $\delta_{3k}^{\,p}$ for the output layer, calculate the weight changes using the equations:

$$\Delta v_{ij,new}^{p} = \eta \delta_{2j}^{p} a_i^{p} + \alpha \Delta v_{ij}^{p-1} \tag{2.41}$$

$$\Delta w_{jk,new}^{p} = \eta \delta_{3k}^{p} b_j^{p} + \alpha \Delta w_{jk}^{p-1} \tag{2.42}$$

where η is the *learning rate*, and α is the *momentum coefficient*. As mentioned, momentum is simply an added weight used to speed up the training rate. The momentum coefficient, α, is usually restricted such that $0 < \alpha < 1$. Thus, the momentum terms, $\alpha \Delta w_{jk}^{p-1}$ and $\alpha \Delta v_{ij}^{p-1}$, are fractional values of the weight changes from the previous training iteration.

Example: We calculate the changes in weights, $\Delta v_{ij,new}$, and $\Delta w_{jk,new}$. We arbitrarily set the learning rate, η, to 0.9, and the momentum coefficient, α, to 0.7. Thus, for $v_{ij,new}$:

$$\Delta v_{11,new} = \eta \delta_{21} a_1 + \alpha \Delta v_{11} = (0.9)(-0.019257)(0.57444) + (0.7)(0) = -0.009956$$

$$\Delta v_{12,new} = \eta \delta_{22} a_1 + \alpha \Delta v_{12} = (0.9)(-0.070936)(0.57444) + (0.7)(0) = -0.036674$$

$$\Delta v_{13,new} = \eta \delta_{23} a_1 + \alpha \Delta v_{13} = (0.9)(-0.026941)(0.57444) + (0.7)(0) = -0.013928$$

On this first time step, Δv_{ij} (from the "previous step") is zero, since no previous step exists. We continue this procedure for all $\Delta v_{ij,new}$'s, and end up with:

$$[\Delta v_{ij,new}] = \begin{bmatrix} -9.96*10^{-3} & -3.67*10^{-2} & -1.39*10^{-2} \\ -9.10*10^{-3} & -3.35*10^{-2} & -1.73*10^{-2} \\ -9.53*10^{-4} & -3.51*10^{-2} & -1.33*10^{-2} \end{bmatrix}$$

We also calculate the weight change $\Delta w_{jk,new}$ using the same learning rate (η = 0.9) and momentum coefficient ($\alpha = 0.7$):

$$\Delta w_{11,new} = \eta \delta_{31} b_1 + \alpha \Delta w_{11} = (0.9)(0.06162)(0.77302) + (0.7)(0) = 0.041856$$

$$\Delta w_{12,new} = \eta \delta_{32} b_1 + \alpha \Delta w_{12} = (0.9)(-0.12858)(0.77302) + (0.7)(0) = -0.089455$$

$$\Delta w_{13,new} = \eta \delta_{33} b_1 + \alpha \Delta w_{13} = (0.9)(-0.18357)(0.77302) + (0.7)(0) = -0.12771$$

Again, on this first time step, $\Delta w_{ij} = 0$, since no previous time step exists. We continue this procedure for all $w_{ij,new}$'s to yield:

$$[\Delta w_{jk,new}] = \begin{bmatrix} 4.19*10^{-2} & -8.95*10^{-2} & -1.28*10^{-1} \\ 3.31*10^{-2} & -7.03*10^{-2} & -1.00*10^{-2} \\ 3.92*10^{-2} & -9.09*10^{-2} & -1.30*10^{-1} \end{bmatrix}$$

Step 9: Knowing the weight changes, update the weights according to the equations:

$$w^{p}_{jk,new} = w^{p-1}_{jk} + \Delta w^{p}_{jk,new} \qquad (2.43)$$

$$v^{p}_{ij,new} = v^{p-1}_{ij} + \Delta v^{p}_{ij,new} \qquad (2.44)$$

where v_{ij}^{p} is the connection weight between the i^{th} element in the input layer and j^{th} element in the hidden layer, w_{jk}^{p} is the connection weight between the j^{th} element in the hidden layer and k^{th} element in the output layer, both for the p^{th} training pattern. Repeat steps 2-9 for all training patterns until the squared error is zero or sufficiently low.

Example: Knowing the weight changes, we update the weights. The old and new weights are:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [v_{ij,new}] = \begin{bmatrix} -1.0009 & -0.5367 & 0.4867 \\ 0.9909 & -0.0335 & -0.5173 \\ 0.4990 & -0.5351 & 0.4867 \end{bmatrix}$$

$$[w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [w_{jk,new}] = \begin{bmatrix} -0.9581 & -0.5895 & 0.4872 \\ 1.0331 & -0.0703 & 0.4900 \\ 0.5392 & -0.4091 & 0.4870 \end{bmatrix}$$

We have now completed one iteration. We repeat steps 2-9 until the mean-squared error calculated from equation (2.33) is below an acceptable limit. Appendix 2B gives a Basic program for network training using the delta-rule algorithm. The delta-rule algorithm requires only 784 time steps to achieve less than a 1% error on variable d_1 (i.e. $e_1 < 0.01$). The results are:

Number of time steps: 784

Desired values:	$d_1 = 1$	$d_2 = 0$	$d_3 = 0$
Actual values:	$c_1 = 0.9900$	$c_2 = 0.0099$	$c_3 = 0.0099$
Percent error:	$e_1 = 1.00\%$	$e_2 = 0.99\%$	$e_3 = 0.99\%$

The superiority of the delta-rule algorithm over the vanilla backpropagation algorithm is evident: the vanilla backpropagation algorithm required *3860 time steps* to achieve a 1% error on d_1, while the delta-rule algorithm only required 784. That is almost an 80% decrease in time steps. In addition, the delta-rule algorithm yielded lower errors for both d_2 and d_3.

Note that above, we used only one data point (input pattern). However, the delta-rule algorithm can accommodate multiple input patterns, and the procedure outlined here may be applied to a neural network with p number of input patterns. Section 2.3 describes some practical aspects of neural computing, including backpropagation learning. First, however, we wish to consider an alternative formulation of backpropagation learning.

E. Alternative Formulation of Backpropagation Learning

This section presents an alternative formulation of the 3-layer feedforward perceptron network and the delta-rule algorithm to make them easier to implement on currently

available, commercial neural network software. Figure 2.12 shows a simplified neural network with internal thresholds T_{ij} (i = 1 to 3, j = 1 to 3) and sigmoid threshold functions used in our discussion of the delta-rule algorithm.

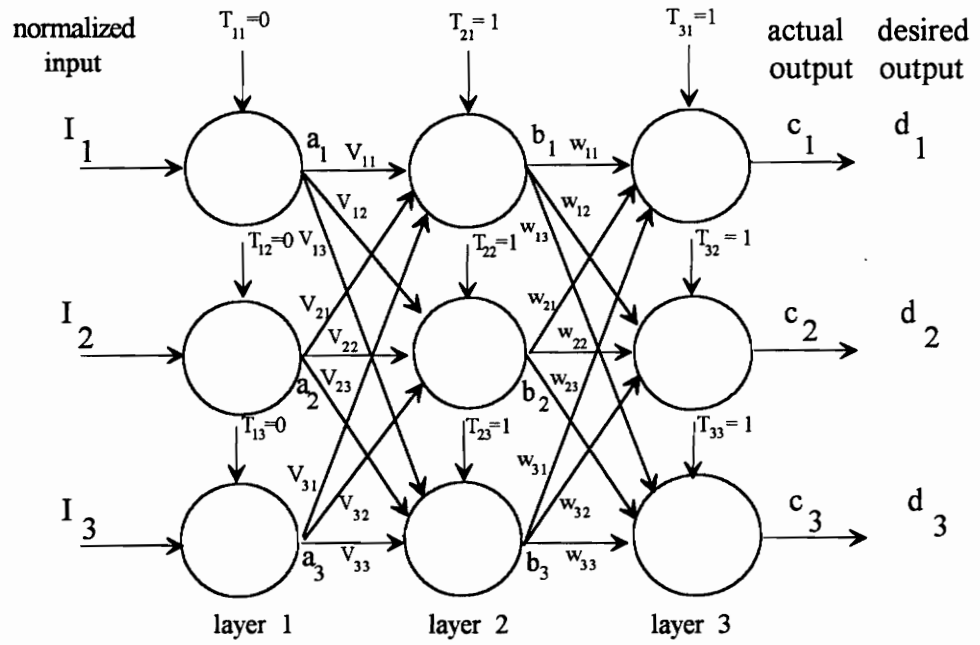


Figure 2.14. A simplified three-layer, feedforward perceptron network for the fault-diagnosis problem with internal thresholds T_{ij} (i = 1 to 3, j = 1 to 3) and sigmoid threshold functions.

We learn from steps 3 and 4 of the delta-rule algorithm, section 2.2.D.2 that:

- output from the hidden layer : $$b_j = f\left(\sum_{i=1}^{L} (v_{ij}\, a_i) + T_{2j} \right) \qquad (2.45)$$

- output from the output layer : $$c_k = f\left(\sum_{j=1}^{m} (w_{jk}\, b_j) + T_{3k} \right) \qquad (2.46)$$

For Figure 2.14, we have i = 1 to 3 (=L), j = 1 to 3 (=m), and k = 1 to 3 (=n).

In commercial neural network software, there is a popular way of incorporating the internal thresholds T_{ij} (i = 1 to 3, j = 1 to 3) and adjusting their values during the network training, as Figure 2.15 shows. Specifically, we introduce a so-called *bias* node with a constant input a_0 of 1.0. This bias input a_0 is fed to all nodes in the hidden and output layers with weight factors v_{0j} and w_{ok} (j = 1 to 3; k = 1 to 3), and the resulting weighted biases replace the internal thresholds T_{2j} and T_{3k} (j = 1 to 3; k = 1 to 3).

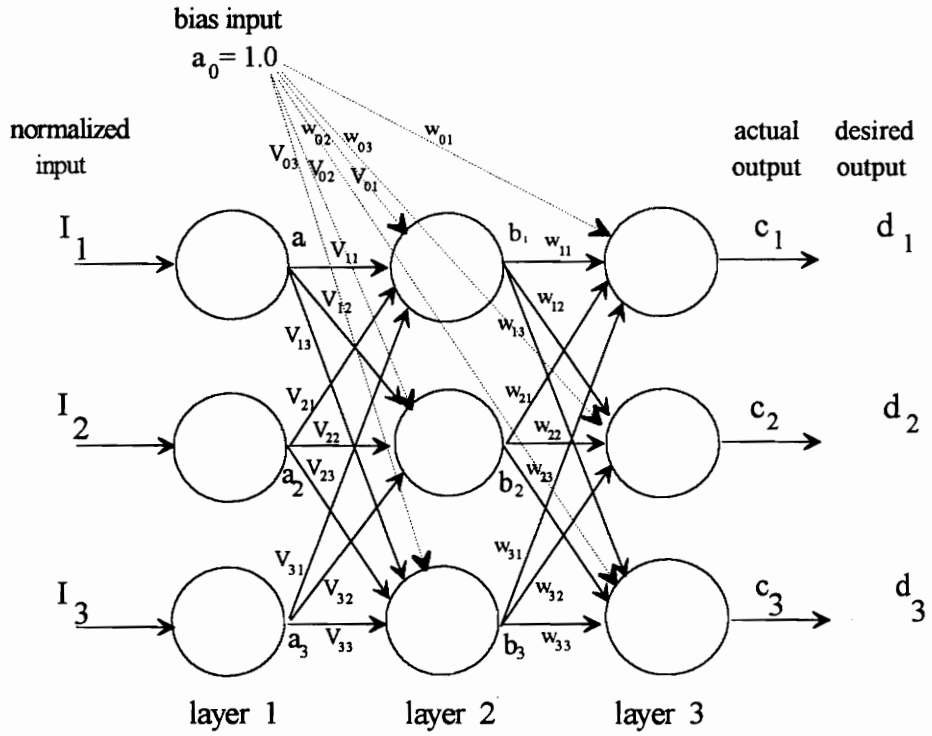


Figure 2.15. An alternative architecture of the three-layer feedforward perceptron network for the fault-diagnosis problem with a basis input and weight factors v_{oj} and w_{0k} (j = 1 to 3; k = 1 to 3) replacing internal thresholds T_{ij} (i = 1 to 3, j = 1 to 3).

In other words, we may write:

- output from the hidden layer :

$$b_j = f\left(\sum_{i=1}^{L} (v_{ij}\, a_i) + T_{2j} \right) \tag{2.47}$$

$$b_j = f\left(\sum_{i=1}^{L} (v_{ij}\, a_i) + v_{0j} \cdot a_0 \right) \tag{2.48}$$

$$b_j = f\left(\sum_{i=0}^{L} (v_{ij}\, a_i) \right) \quad (a_0 = 1.0) \tag{2.49}$$

- output from the output layer :

$$c_k = f\left(\sum_{j=1}^{m} (w_{jk}\, b_j) + T_{3k} \right) \tag{2.50}$$

$$c_k = f\left(\sum_{j=1}^{m} (w_{jk}\, b_j) + w_{0k} \cdot b_0 \right) \tag{2.51}$$

$$c_k = f\left(\sum_{j=0}^{m} (w_{jk}\, b_j) \right) \quad (a_0 = b_0 = 1.0) \tag{2.52}$$

Since we set the bias input $a_0 = 1.0$, and also require that $a_0 = b_0$, we find:

$$T_{2j} = v_{0j} \quad (j = 1 \text{ to } 3) \tag{2.53}$$

$$T_{3k} = w_{0k} \quad (k = 1 \text{ to } 3) \tag{2.54}$$

These results indicate that internal thresholds T_{2j} and T_{3k} (j = 1 to 3, k = 1 to 3) in the neural network of Figure 2.14 become the weight factors between the bias node and the

nodes in the hidden and output layers. With this formulation, we can use the delta-rule algorithm to iteratively modify the values of internal thresholds T_{2j} and T_{3k} (j = 1 to 3, k = 1 to 3) during the weight-factor correction process.

The alternative formulation incorporating a bias node and bias weight factors, shown in Figure 2.15, is basic to implementing the delta-rule algorithm with currently available commercial software. Figure 2.16 gives an example of such a network architecture as used by NeuralWorks Explorer and Professional II/PLUS, NeuralWare, Inc., Pittsburgh, PA. This formulation is also popular in neural network publications in chemical engineering. See, for example, Figure 2.17 (Venkatasubramanian et al., 1990).

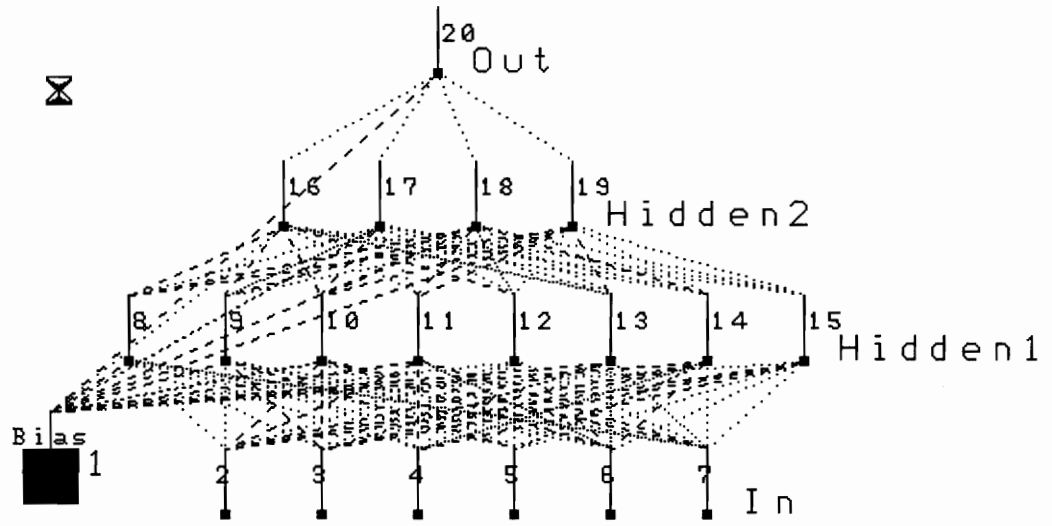


Figure 2.16. An example of a network architecture incorporating a bias node and bias weight factors being used by commercial neural network software (Neuralware, Inc., 1991).

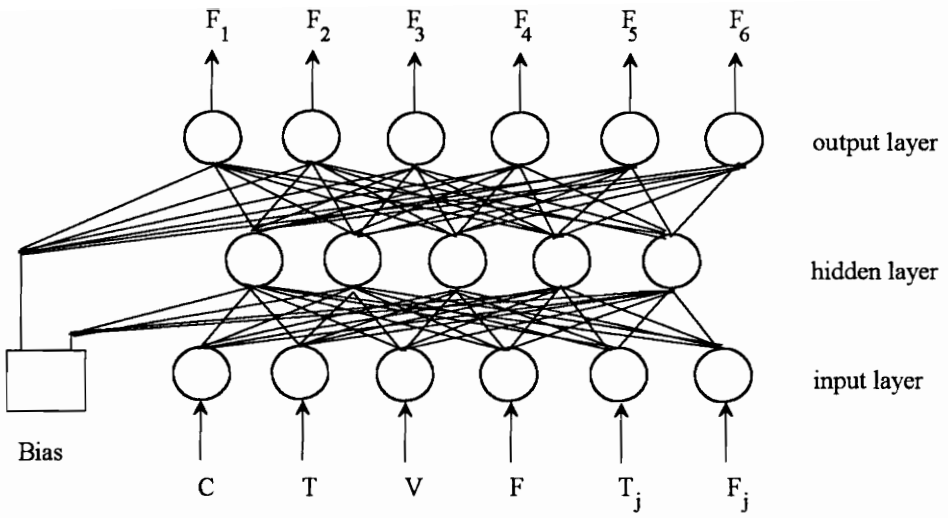


Figure 2.17. An example of a network architecture incorporating a bias node and bias weight factors being used in chemical engineering publications of neural networks (Venkatsubramanian et al., 1990)

2.3 Practical Aspects of Neural Computing

There are many neural network features which control the network's performance and prediction capability. To develop an effective neural network model, we must select these key features with care. In the following sections, we describe these practical aspects: (1) selecting the number of hidden layers; (2) normalizing input and output data sets; (3) initializing the weight-factor distribution; (4) setting the learning rate and momentum coefficient; (5) selecting the proper transfer function; and (6) generating a network learning curve.

Throughout this section, we sometimes refer to a number of illustrative examples (presented in detail in later chapters) and the practical observations from those examples that are important to network training. Our goal of citing the observations is to enable the reader to better understand the quantitative effects of various network parameters. There is no need for our reader to dwell on the details of those examples until later chapters.

A. Selecting the Number of Hidden Layers

Although the fault-diagnosis problem outlined in this chapter (Section 2.2.B) uses a network with only one hidden layer, neural networks are certainly not restricted to that structure. They may, in fact, have multiple hidden layers, and determining the number of hidden layers is a critical part of designing a network. To determine the optimal number of hidden layers, and the number of nodes in each layer, we train the network using various configurations, then select the configuration with the fewest number of layers and nodes that still yields the minimum root-mean-square (RMS) error quickly and efficiently.

As will be described in Chapter 3, we recommend the use of radial-basis-function networks for classification problems. For these problems, a single hidden layer is sufficient for effectively identifying process faults or feature categories.

In the literature, most prediction networks, such as those in Chapter 4 and most process-forecasting networks such as those in Chapter 5, have been limited to single-layer networks. However, we consistently find that adding a second hidden layer significantly

improves the network's prediction capability, *without any detrimental effects on the generalization of the testing data set*. To illustrate this, we list below the optimal hidden-layer configurations for the case studies of Chapters 4 and 5.

- The *isomerization network* (Figure 4.2) for the prediction of reaction-rate data for the catalytic isomerization of n-pentane to isopentane (Section 4.2) has an optimal network structure with a 30:15 hidden-layer configuration.
- The *tyrosine-composition prediction network* (Figure 4.2) for the prediction of biomolecule compositions from fluorescent spectra (Section 4.3) has an optimal network structure with a 30:15 hidden-layer configuration.
- The *autoclave-processing network* (Figure 4.16) for the prediction of composite thickness and void size in the autoclave curing process (Section 4.4) has an optimal network structure with a 20:10 (composite thickness) and 20:12:7 (void size) hidden-layer configuration .
- The *fermentation-processing network* (Figure 5.22) for the prediction of cell concentration in a batch cell-growth process (Section 5.4) has an optimal network structure with a 30:15 hidden-layer configuration.

These case studies consistently indicate that adding a second hidden layer effectively improves the prediction capability of neural networks. Adding a third hidden layer, however, yields prediction capabilities similar to those of the 2-hidden-layer networks, but requires longer training times due to the more complex structures. As a

result, *we recommend using a 30:15 hidden-layer configuration as the initial architecture for most networks.* The 30:15 hidden-layer configuration performs effectively for all prediction networks in this text, although it is not always the optimal network configuration.

B. Normalizing Input and Output Data Sets

Many neural network software packages, such as NeuralWare's Professional II/PLUS (1993), normalize the input and output data sets for the user. This normalization is very critical, if the input and output variables are not of the same order of magnitude, some variables may be given more significance than they otherwise would have. The training algorithm is forced to compensate for order-of-magnitude differences by adjusting the network weights, which is not very effective for many of the training algorithms (i.e., backpropagation algorithm). For example, if input variable 1 has a value of 10,000 and input variable 2 has a value of 10, the assigned weight for the second variable going into a node of hidden layer 1 must be much greater than that of the first variable to have any significance. In addition, the typical transfer function, such as a sigmoid function (2.3) or a hyperbolic tangent function (2.4), cannot distinguish between two different values of x_i when the latter are large because they yield identical threshold output values of 1.0. For example, using the sigmoid function (2.3), when $x_i = 5$, we find $f(x_i) = 0.993$; $x_i = 50, f(x_i) = 1.00$; $x_i = 500, f(x_i) = 1.00$.

This section introduces three main types of normalization procedures. The first normalizes each variable, x_i, in the data set to between 0 and 1 by dividing its value by the upper limit of that variable, $x_{i,max}$, to give a normalized variable, $x_{i,norm}$:

$$x_{i,norm} = \frac{x_i}{x_{i,max}} \tag{2.55}$$

To illustrate, let us give a variable, x_i, a normal distribution between 500 and 900 and assign a normalization factor of $x_{i,max} = 1000$. Therefore, we transform the variable, x_i, to a normal distribution between 0.5 and 0.9. One limitation of this method is that it does not utilize the entire range of the transfer function. For this example, Figure 2.18a illustrates that only a small portion of the transfer function corresponds to x_i values of 0.5 to 0.9 and -0.5 to -0.9. (Note that we include the negative range also, even though it is not represented in Figure 2.18a, because the weight factors also have negative values). The weight factors can broaden and shift this range to include a larger region of the transfer function. However, as the number of variables and weight factors increase, these adjustments become more difficult for training algorithms. As a result, this normalization method is adequate for many simple networks, but problems can arise as the network architectures become more complex. We use this normalization technique for most of the illustrative case studies in this text, since it enables the reader to easily understand data sets and interpret results without more complex data transformations.

The second method expands the normalization range so that the minimum value of

the normalized variable, $x_{i,norm}$, is set at 0 and the maximum value, $x_{i,norm}$, is set at one. We define the normalized variable $x_{i,norm}$ by using the minimum and maximum abd maximum values of the original variable, $x_{i,min}$ and $x_{i,max}$, respectively.

$$x_{i,norm} = \frac{x_i - x_{i,min}}{x_{i,max} - x_{i,min}} \tag{2.56}$$

This method significantly improves on the first method by using the entire range of the transfer function, as Figure 2.18b illustrates. (Again, note that we include the negative region of the transfer function, because the weight factors have both positive and negative values). Another benefit from this method is that every input variable in the data set has a similar distribution range, which improves training efficiency. The illustrative case study for fault diagnosis of an unsteady-state CSTR (continuous stirred-tank reactor) system (Section 3.4.D.2) uses this normalization method.

The third technique normalizes the data set between limits of -1 and +1, having the average value set at 0. We call this technique *the zero-mean normalization method* and represent the normalization variable, $x_{i,norm}$ by:

$$x_{i,norm} = \frac{x_i - x_{i,avg}}{R_{i,max}} \tag{2.57}$$

and

$$R_{i,max} = \text{maximum } [x_{i,max} - x_{i,avg},\ x_{i,avg} - x_{i,min}] \tag{2.58}$$

where x_i is an input or output variable, $x_{i,avg}$ is the average value of the variable, $x_{i,min}$ is the minimum value of the variable, $x_{i,max}$ is the maximum value of the variable, and $R_{i,max}$ is the maximum range between the average value and either the minimum or the maximum value.

As in the second method, the zero-mean method utilizes the entire range of the transfer function, and every input variable in the data set has a similar distribution range (Figure 2.18c). Moreover, this method gives some meaning to the values of the normalized variable: 0 represents the normal state (average) of the variable, -1 represents a very low level of the variable, and +1 represents a very high level of the variable. In addition, by setting all of the normal states of the variables to zero, the network will always have a standard structure that makes training more consistent from one problem to the next and also more efficient. Specifically, all networks should normally predict output responses of approximately 0 (nominal value) for a set of input variables at their nominal values of 0. Therefore, the network is essentially only training deviations in the output variable to various deviations in the input variables. We use this zero-mean normalization method in the illustrative case study for process quality control and optimization of the autoclave curing process for manufacturing composite materials in Section 4.4.

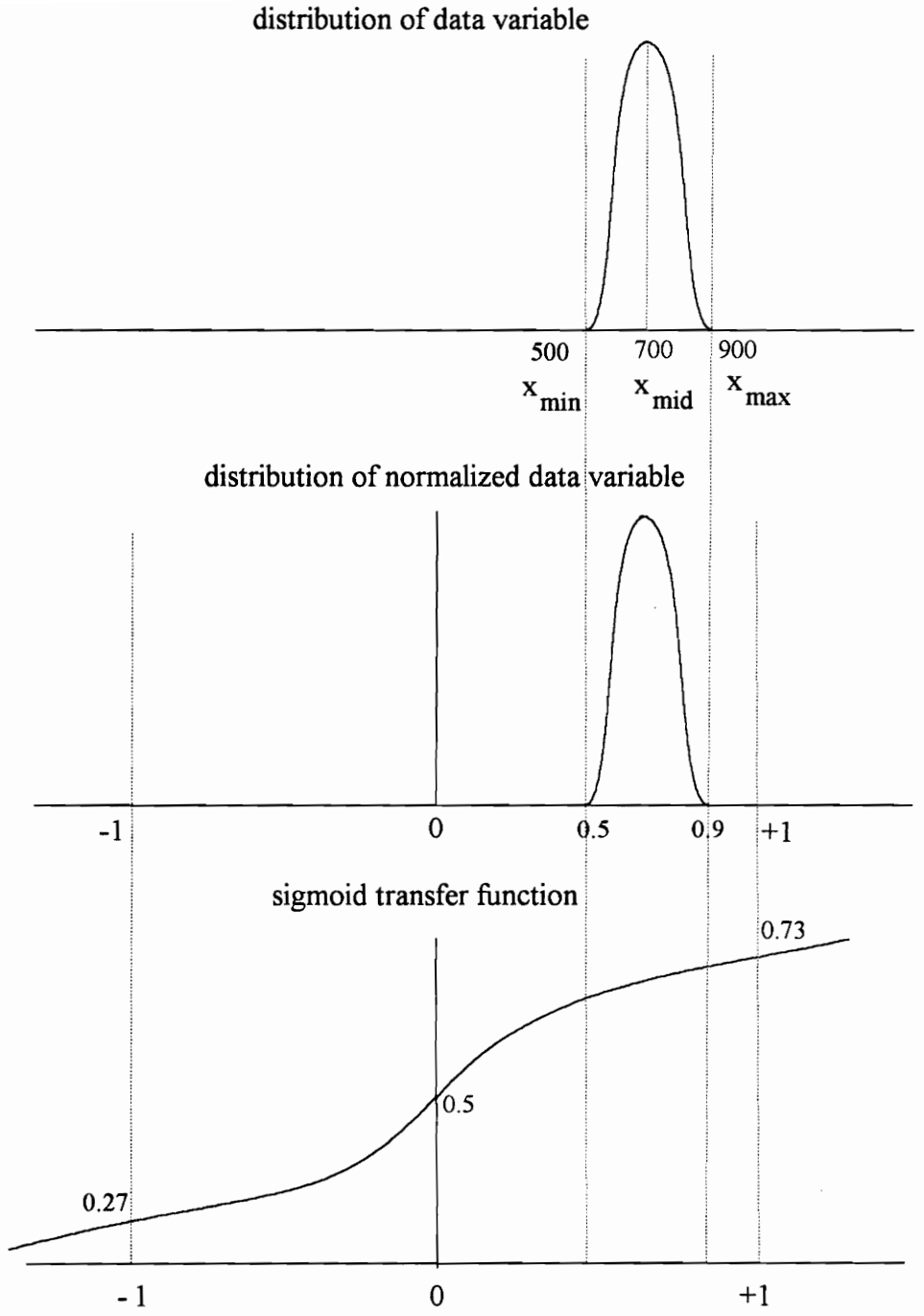


Figure 2.18a. An illustration of the normalization of a data set that ranges from 500 to 900 and its mapping onto a sigmoid transfer function using three normalization methods: method 1

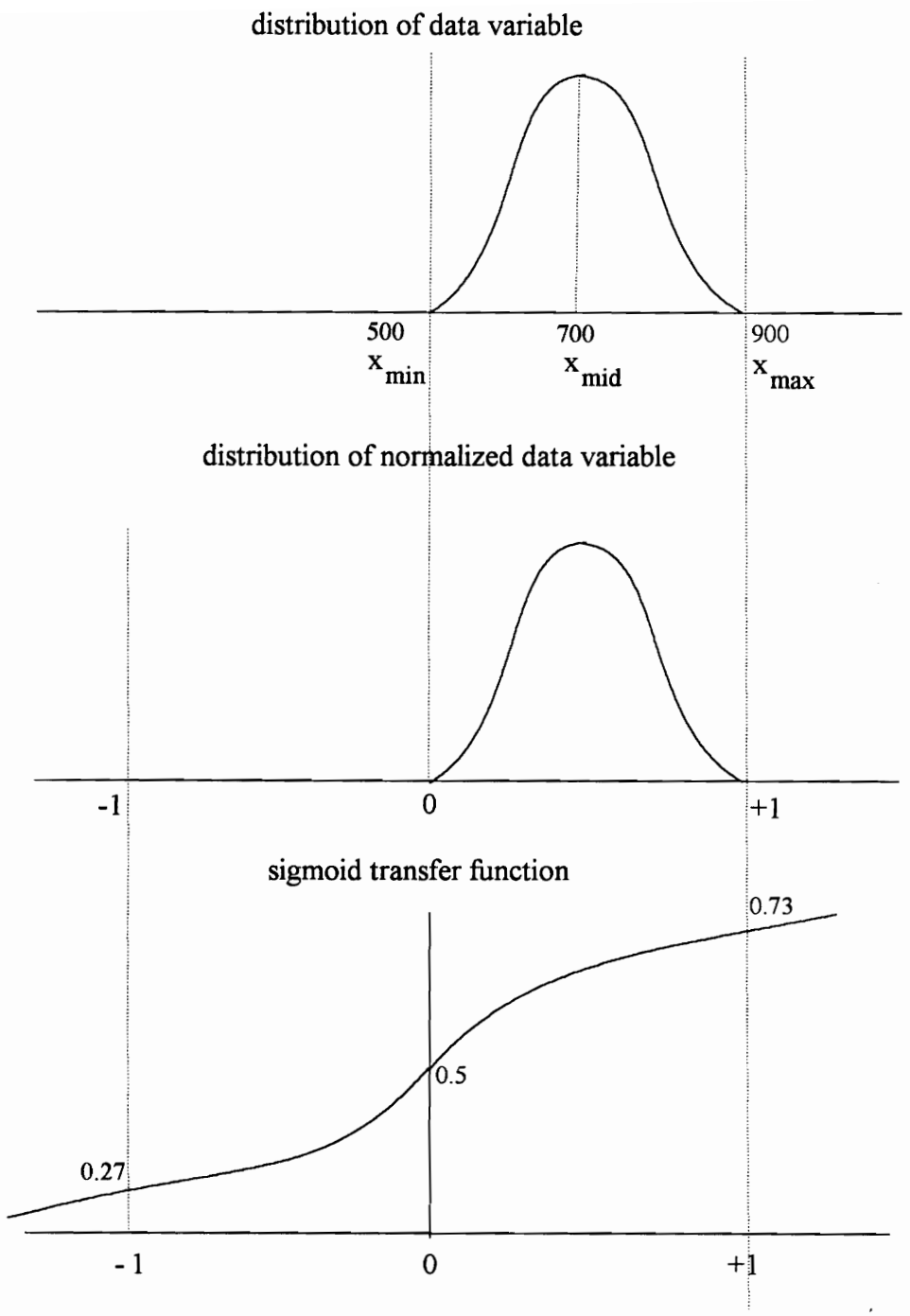


Figure 2.18b. An illustration of the normalization of a data set that ranges from 500 to 900 and its mapping onto a sigmoid transfer function using three normalization methods: method 2.

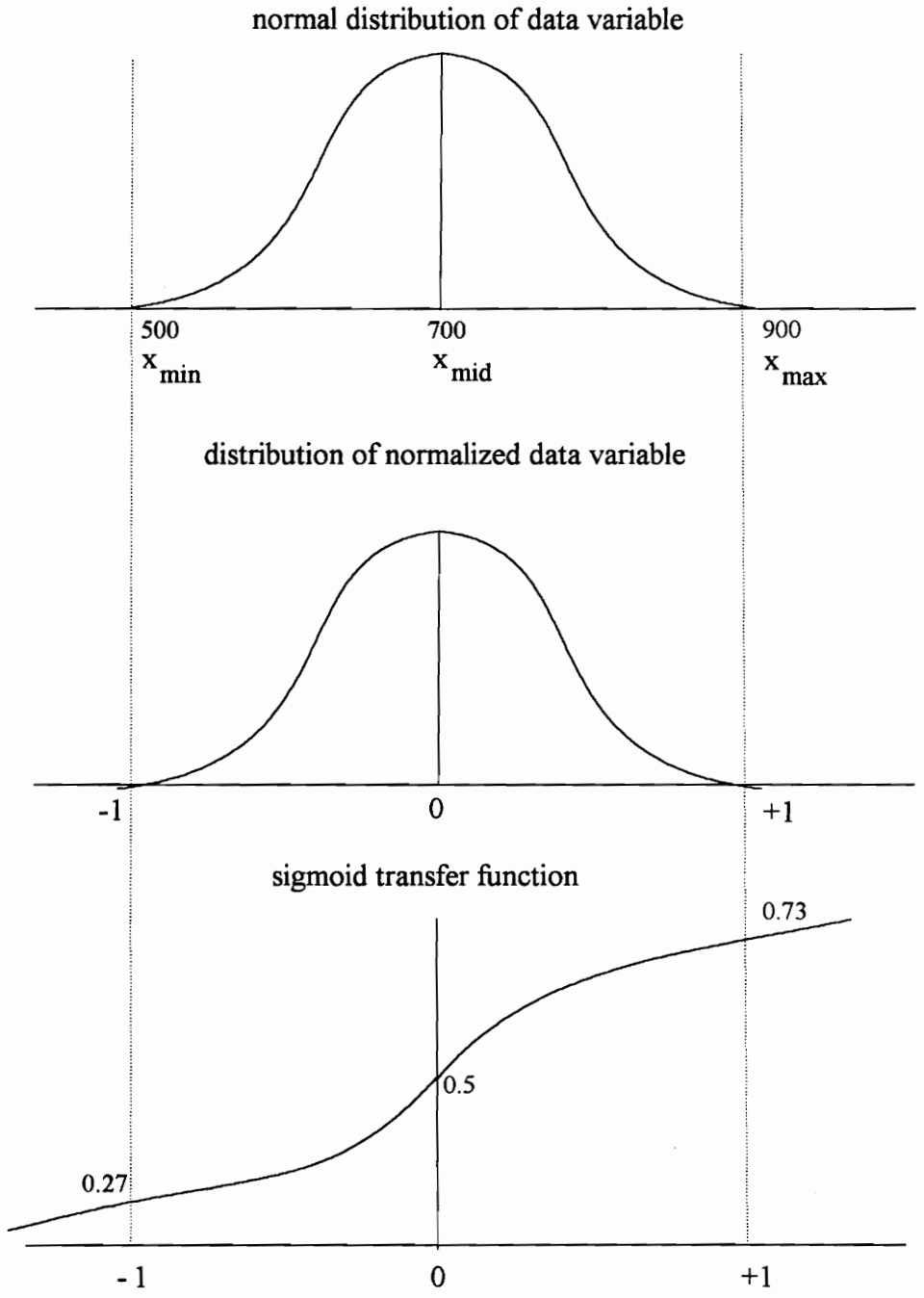


Figure 2.18c. An illustration of the normalization of a data set that ranges from 500 to 900 and its mapping onto a sigmoid transfer function using three normalization methods: zero-mean normalization.

C. Initializing the Weight-Factor Distribution

Prior to training a neural network, we must first initialize the weight factors, w_{ij} (Section 2.1.A.2), between the nodes of the hidden layers. The weight factors are set randomly with either a uniform or Gaussian distribution. We have found the Gaussian distribution effective for the cases studies in this text.

For neural networks that are relatively simple, the initial distribution of the weight factors is not particularly critical. The initial distribution set by the software package NeuralWare's Professional II/PLUS (1993), for instance, almost always performs adequately in network training. The initial weight-factor distribution is normally set to a fairly narrow range, and allowed to broaden using high learning rates and high momentum coefficients in the early stages of the training process, as Section 2.3.D will explain. Problems can occur, however, when using very large data sets and/or complex network architectures (e.g., the protein-partitioning network of Section 6.2.C). For complex networks, the weight-factor distribution does not broaden much during network training, and therefore we mus set the initial values to coincide with our normalized input and output variables.

To understand how the weight-factor distribution affects a neural network's prediction capability, consider the protein-partitioning network of Section 6.2.C. That network predicts a protein's partition coefficient contribution due to pH variations from the isoelectric point, $\ln K_{pH}$, using the data file *phpart.nna* (see Table 6.20). To show the

effect of the weight-factor distribution, we train the protein-partitioning network with multiple Gaussian distributions ranging from ±0.5 to ±2.5. Table 2.2 show the average errors for the prediction of $\ln K_{pH}$ with different weight-factor distributions. The optimal distribution, as Table 2.2 shows, is Gaussian with a range of -1.25 to 1.25. Note that the average error increases significantly with distributions having ranges less than ± 0.75 or greater than ± 2.5.

Table 2.2. Average errors for the prediction of the protein's partition coefficient contribution due to pH variations from the isoelectric point, $\ln K_{pH}$ and K_{pH}, with different weight-factor distributions.

Weight-factor distribution	Average error $\ln K_{pH}$	Average error K_{pH}
Gaussian : [-0.50 to 0.50]	0.136	0.333
Gaussian : [-0.75 to 0.75]	0.130	0.294
Gaussian : [-1.00 to 1.00]	0.104	0.253
Gaussian : [-1.25 to 1.25]	0.098	0.214
Gaussian : [-1.75 to 1.75]	0.108	0.276
Gaussian : [-2.50 to 2.50]	0.136	0.333

The zero-mean normalization can significantly improve the prediction capability of complex networks by setting all input and output variables to similar distributions. This allows the weight factors to follow a more standard distribution, without requiring them to

shift and broaden the input vectors to match their respective output vectors. Therefore, the weight factors only have to map the deviations of the output vector from their nominal value (average) in response to deviations in the input vector from their nominal value. Consequently, we are able to generate a standard set of weight-factor distributions which identify the best initial distributions for neural networks with different number of layers.

Table 2.3 lists our recommendations for good initial distributions of weight factors used in training backpropagation networks with an increasing number of input variables and a varying number of nodes in the hidden layer. The recommended initial weight-factor distributions vary only slightly among networks with 1 to 3 hidden layers, provided the total number of nodes in the hidden layer(s) is the same. Therefore, Table 2.3 lists distributions based only on the total number of nodes in the hidden layers. The recommended distributions are slightly narrowed because the distribution can always broaden when the learning rate and momentum coefficients are properly set, but the training algorithm will not tighten the distribution when required. These distributions are fairly consistent with those set by NeuralWare's Professional II/PLUS (1993) in their network initialization procedure.

Table 2.3. A standard set of interconnecting-weight distributions which identify the good initial Gaussian distributions for different total number of nodes in the hidden layer(s) for neural networks with 1 to 3 hidden layers.

Number of input variables (nodes)	Total number of nodes in the hidden layer(s)		
	30	45	100
5	-0.70 to 0.70	-0.50 to 0.50	-0.35 to 0.35
10	-0.50 to 0.50	-0.40 to 0.40	-0.30 to 0.30
20	-0.45 to 0.45	-0.35 to 0.35	-0.30 to 0.30
30	-0.40 to 0.40	-0.30 to 0.30	-0.25 to 0.25
50	-0.30 to 0.30	-0.20 to 0.20	-0.20 to 0.20
100	-0.25 to 0.25	-0.20 to 0.20	-0.20 to 0.20

D. Setting the Learning Rate and Momentum Coefficient

The learning rate and the momentum coefficient are two very important parameters that control how effectively the backpropagation algorithm trains the neural network. The learning rate is a positive constant that controls the rate at which the new weight factors are adjusted based on the calculated gradient-descent correction term. The momentum coefficient is an extra weight added onto the weight factors that accelerates the rate at which the weight factors are adjusted. The momentum coefficient helps move the

minimization routine out of local minima, as Figure 2.13 illustrates. With these two parameters, the equation for adjusting the weight factors is:

$$\begin{bmatrix}\text{new weight}\\ \text{factor}\end{bmatrix}=\begin{bmatrix}\text{old weight}\\ \text{factor}\end{bmatrix}+\begin{bmatrix}\text{learning}\\ \text{rate}\end{bmatrix}\times\begin{bmatrix}\text{input}\\ \text{term}\end{bmatrix}\times\begin{bmatrix}\text{gradient-descent}\\ \text{correction term}\end{bmatrix}+\begin{bmatrix}\text{momentum}\\ \text{coefficient}\end{bmatrix}\times\begin{bmatrix}\text{previous}\\ \text{weight change}\end{bmatrix} \tag{2.28}$$

To understand the effects of learning rate on network training, consider the autoclave-processing network (Section 4.4.B) for predicting composite thickness (file : *thcktrn.nna*). That system uses a backpropagation network with 30 nodes in hidden layer 1 and 15 nodes in layer 2, the delta learning rule, the hyperbolic tangent transfer function, and a momentum coefficient to 0. Figure 2.19 compares the root-mean-squares (RMS) error using a common learning rate (0.3), a low learning rate (0.01), and a high learning rate (5.0) over 2,000 iterations. As Figure 2.19 shows, with a learning rate of 0.3, the network reaches an RMS error of 0.1 in approximately 150 iterations. When the learning rate is very low (e.g., 0.01), the training requires more than 500 iterations to meet the same error level. This difference in the number of training iterations can be very significant in more complex networks. In addition, with a low learning rate, training may be trapped in a local minimum, as illustrated in Figure 2.20a. At the other extreme, when the learning rate is to high (e.g., 5.0), training can become unstable (Figure 2.19). Figure 2.20b shows oscillation across opposite sides of the overall minimum.

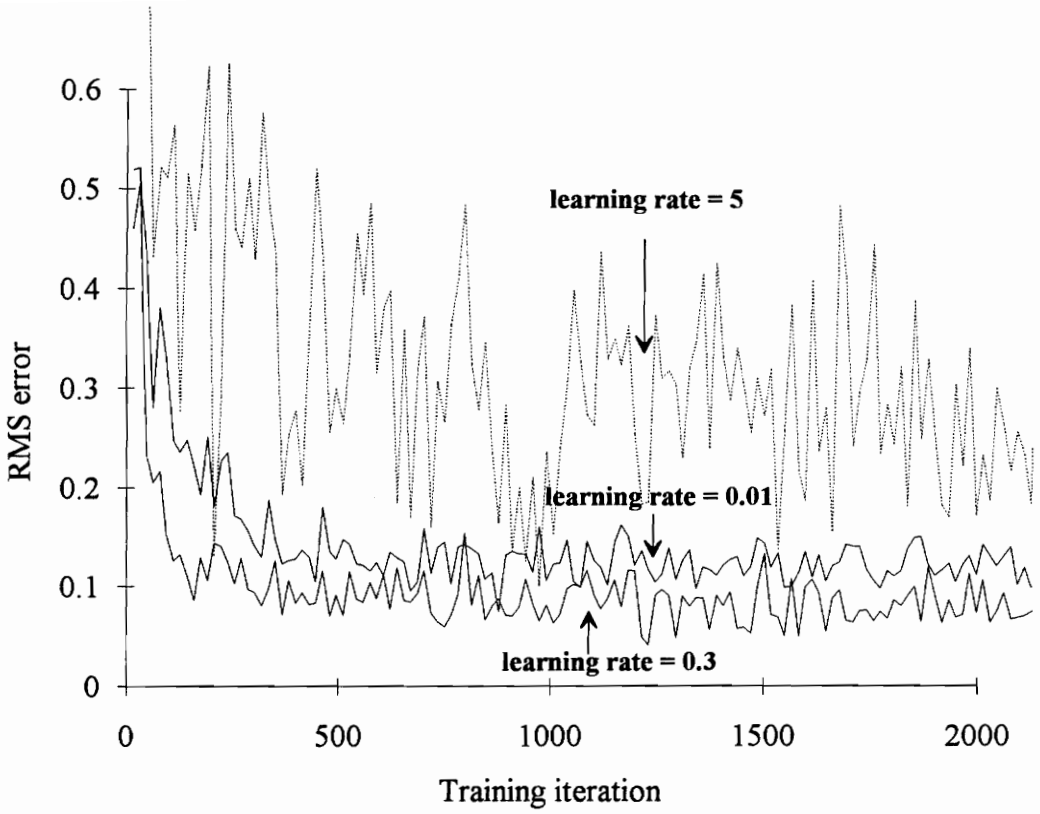


Figure 2.19. A comparison of the RMS error in training the autoclave-processing network for composite thickness with a frequently used learning rate (0.3), a low learning rate (0.01), and a high learning rate (5.0).

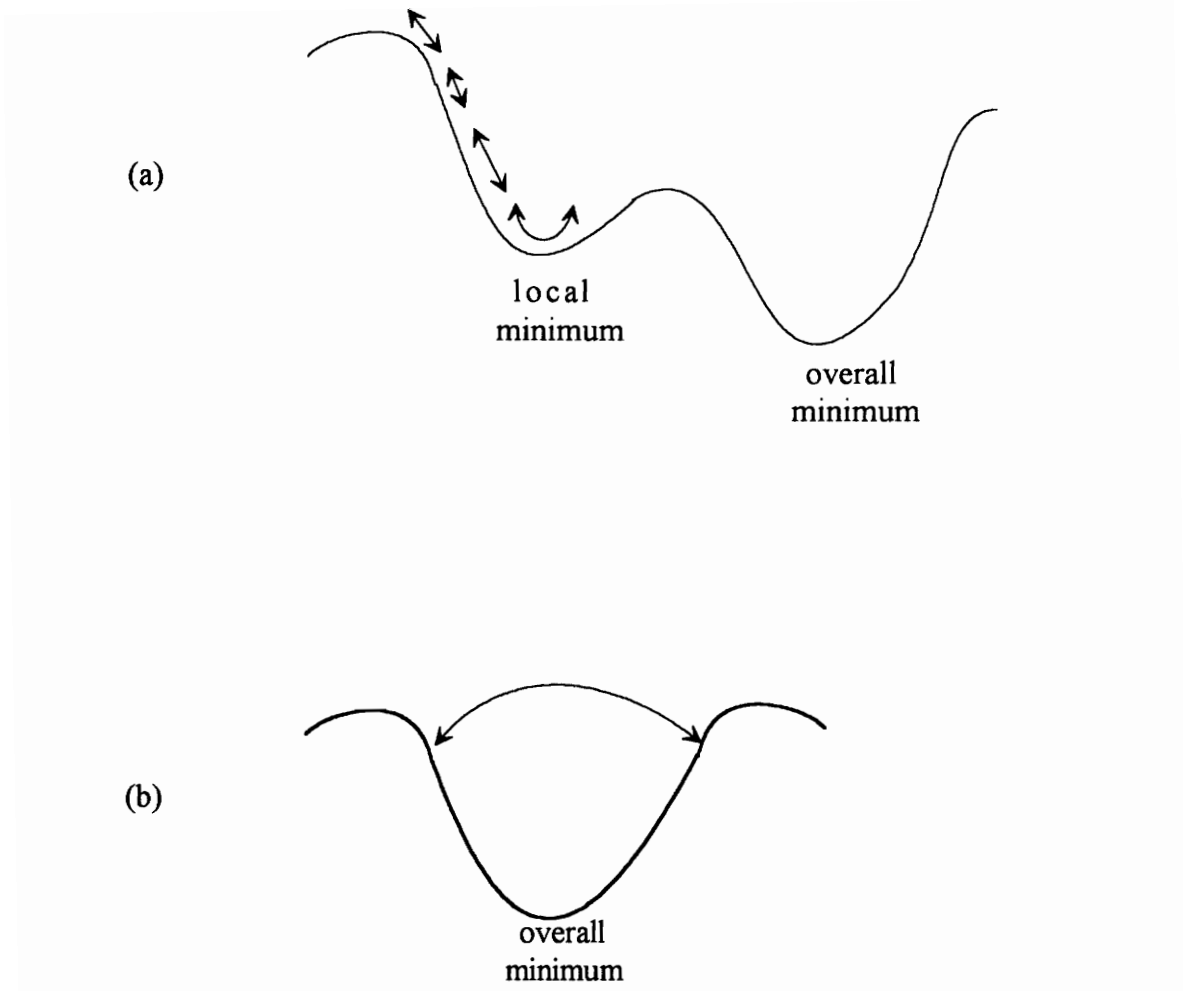


Figure 2.20a-b. Problems associated with network training using : (a) a very low learning coefficient, reaching a local minimum; and (b) a very high learning rate, causing oscillation across the overall minimum.

To illustrate the effects of both learning rate and momentum coefficient on training the autoclave-processing network, let us se the same network specifications, this time varying the learning rate from 0.01 to 4 and the momentum coefficient from 0.0 to 0.8. Table 2.4 lists the RMS errors for training the autoclave-processing network after 5000 iterations. Increasing the momentum coefficient decreases the training time, as indicated by the reduced RMS error at 5000 iterations. Eventually, the momentum coefficient reaches a threshold point at which the training algorithm begins to oscillate. Table 2.4 suggests that effective initial values for learning rate and momentum coefficient are 0.3 to 0.7, and 0.4, respectively.

Table 2.4. The RMS errors for training the autoclave-processing network for composite thickness, 30:15 hidden-layer configuration, over different learning rates and momentum coefficients.

Learning rate	Momentum coefficient				
	0.00	0.20	0.40	0.60	0.80
0.01	0.137	0.139	0.139	0.138	0.130
0.10	0.100	0.095	0.082	0.063	0.093
0.30	0.075	0.062	0.054	0.038	> 0.3
0.50	0.057	0.045	0.037	0.034	> 0.3
0.70	0.043	0.035	0.035	> 0.3	> 0.3
1.00	0.040	0.026	0.034	> 0.3	> 0.3
2.00	0.030	0.140	> 0.3	> 0.3	> 0.3
3.00	0.191	> 0.3	> 0.3	> 0.3	> 0.3
4.00	> 0.3	> 0.3	> 0.3	> 0.3	> 0.3

Since training neural networks varies from one problem to another, a useful technique is to set the learning rate and momentum coefficient initially at values that are safe but effective. For example, the default values in NeuralWare's Professional II/PLUS (1993) for the backpropagation algorithm are 0.3 for the learning rate and 0.4 for the momentum coefficient. If we set the initial learning rate and the initial momentum coefficient too high, the training algorithm may begin to oscillate irreversibly, and training must begin all over again. This oscillation occurs because the weight-factor distribution (discussed in Section 2.3.D) rapidly expands beyond a Gaussian distribution and cannot return to an effective distribution.

Importantly, though, the initial values set for the learning rate and the momentum coefficient are not optimal for all stages of the learning process. Therefore, we must develop *a learning schedule* that will train the network at the initial values for a predetermined number of iterations, then gradually reduce those values toward zero at specified iteration numbers. Table 2.5 illustrates the typical learning schedule for a 3-hidden-layer backpropagation network used throughout this text. Each layer has four sets of learning rates and momentum coefficients. For example, for hidden layer 1, the learning rate is 0.3 for the first 10,000 iterations; it then reduces to 0.15, and subsequently drops to 0.00375 at 30,000 iterations and to 0.0234 at 70,000 iterations. The learning rate also decreases as we progress from the first hidden layer to the third. The learning rate is 0.3 for the first 10,000 iterations in the first hidden layer, but only 0.25 and 0.2 in

hidden layers 2 and 3, respectively. In contrast, the momentum coefficient reduces from 0.4 to 0.00312 over 150,000 iterations, but remains constant between hidden layers 1, 2, and 3 at any given iteration.

Table 2.5. A typical learning schedule for a 3-hidden-layer backpropagation network.

Hidden layer 1				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.3	0.15	0.0375	0.0234
Momentum coefficient	0.4	0.2	0.05	0.00312

Hidden layer 2				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.25	0.125	0.03125	0.0195
Momentum coefficient	0.4	0.2	0.05	0.00312

Hidden layer 3				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.2	0.1	0.025	0.00156
Momentum coefficient	0.4	0.2	0.05	0.00312

E. Selecting the Proper Transfer Function

The three main transfer functions used in this text are the sigmoid, the hyperbolic tangent, and the radial basis functions (see Section 2.1.A.4). The sigmoid and hyperbolic tangent transfer functions perform well for the prediction networks in Chapter 4 and for the process-forecasting networks that model time-dependent systems in Chapter 5. However, they do not perform as well for classification networks in Chapter 3. Instead, the radial basis function proves more effective for those networks, and we highly recommended that function for any problems involving fault diagnosis and feature categorization.

For all case studies in this text, *the hyperbolic tangent transfer function outperforms the sigmoid transfer function*. As the biological and chemical processing systems become more complex and nonlinear, the advantages of the hyperbolic tangent transfer function become more apparent. Let us superimpose the hyperbolic tangent function over the sigmoid function. As Figure 2.21 shows, two features distinguish the hyperbolic tangent function:

(a) the slope of the hyperbolic tangent function is much greater than the slope of the sigmoid function;

(b) the hyperbolic tangent function has a negative response for a negative input value and a positive response for a positive input value, while the sigmoid function always has a positive response.

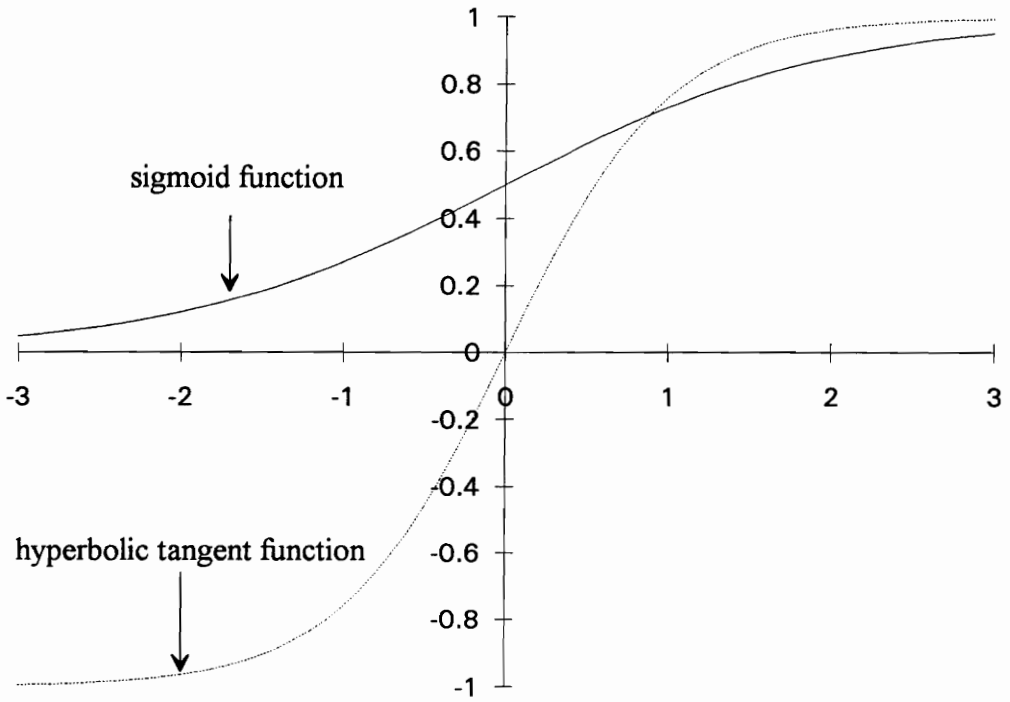


Figure 2.21. The hyperbolic tangent function superimposed over the sigmoid function.

The fact that the hyperbolic tangent function has the greater slope means that it shows a greater response to a small deviation in the input variable. Therefore, it can better distinguish between small deviations in the input variable and can generate a much more nonlinear response. For example, consider the responses of the sigmoid and

hyperbolic tangent transfer functions for normalized inputs of 0.1 and 0.2. The sigmoid function has responses of 0.525 and 0.550 (a difference of only 0.025), while the hyperbolic tangent function has responses of 0.100 and 0.197 (a difference of 0.097). For this region of the transfer function, which represents approximately normal values, the response of the hyperbolic tangent function is approximately 4 times greater that of the sigmoid function (changes of 0.097 versus 0.025, respectively).

The second main feature of the hyperbolic tangent transfer (an output response has the same sign as the input value) is critical for network nodes. Like the zero-mean normalization discussed in 2.3.B, this feature of the hyperbolic tangent transfer function gives some meaning to a node's output value: 0 represents the normal state (average) of a node, -1 represents a very low response level, and +1 represents a very high response level. As mentioned in Section 2.3.B, networks that use both the zero-mean normalization method and the hyperbolic tangent transfer function should normally predict output responses of approximately 0 (nominal value) for a set of input variables at their nominal values of 0. Within this structure, when the input variables are nominally 0, the inputs to the nodes of the first hidden layer are also 0. Consequently, the outputs of those nodes are 0 when using a hyperbolic tangent transfer function. Similarly, the inputs and outputs of the remaining hidden layers and the output layer are also 0. In short, before any training takes place, the network already correctly predicts the nominal case, and essentially only has to be trained for deviations from that case. In comparison, a 0 input to a sigmoid transfer function produces an output response of 0.5, which means that the

network must also adjust the initial weights to train the nominal case.

To further demonstrate the advantages of the hyperbolic tangent transfer function over the sigmoid transfer function for prediction networks consider the quantitative predictions of product compositions from fluorescent spectra of biomolecules (Section 4.3). Table 2.6 compares the effectiveness of both the hyperbolic tangent and sigmoid transfer functions for several networks with 1 to 3 hidden layers. The hyperbolic tangent significantly improves the prediction capability of this network for all configurations, as the comparison of average errors in Table 2.6 indicates.

Table 2.6. A comparison of the effectiveness of the hyperbolic tangent and sigmoid transfer functions in training the tyrosine-composition prediction network with several 1 to 3 hidden layer networks.

Number of nodes			Average error : tyrosine mole fraction	
hidden layer 1	hidden layer 2	hidden layer 3	hyperbolic tangent	sigmoid
5	0	0	0.071	0.126
10	0	0	0.064	0.132
20	0	0	0.058	0.126
30	0	0	0.063	0.150
40	0	0	0.061	0.154
6	3	0	0.079	0.094
10	5	0	0.035	0.097
20	10	0	0.035	0.103
30	15	0	0.019	0.103
9	6	3	0.060	0.097
15	10	5	0.049	0.096
20	12	7	0.035	0.096

When we turn to the radial basis function, we find that it consistently performs better than the sigmoid and hyperbolic tangent transfer functions for classification problems, but works less effectively for prediction and process-forecasting problems. Section 3.2.C describes the practical aspects of the radial basis function, and demonstrates why it is more effective for classification problems.

F. Generating a Network Learning Curve

As discussed in Section 2.1.C.3, one of the most important aspects in developing neural networks is determining how well the network performs at the completion of training. The performance of the network is easily tested through the generation of a learning curve. Figure 2.22 shows a typical learning curve divided in multiple training segments (t1, t2..t7). As previously defined, the learning curve is a plot of average error for both recall of training data sets and generalization of remaining data sets not used in network training as a function of the number of examples in the training data set.

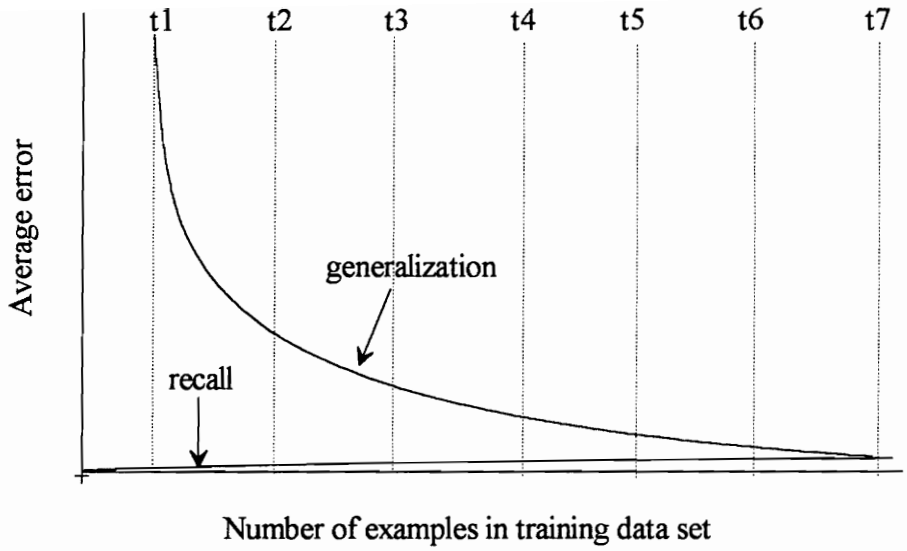


Figure 2.22. A typical learning curve for a well-trained neural network over multiple training segments t1 to t7.

We give a general procedure for generating a learning curve below:

(1) Divide all available training examples into selected groups (t1, t2, ..t7). See Sections 4.2.B (isomerization network), 4.3.C.1 (tyrosine-composition prediction network), and 5.4.D.4b (fermentation-processing network) for examples of how to divide training data sets into selected groups.

(2) Use one group to train the network.

(3) Test the network for both recall of the training data set and generalization of the remaining data groups not used in network training.

(4) Add one randomly selected generalization data group to the training data set.

(5) Retrain the network using the new training data set, reducing the learning rates and momentum coefficients. For example, we reduce our initial learning rate from 0.3 for the initial training group to 0.15 for the remaining training groups and reduce the momentum coefficient from 0.4 to 0.2 (see Table 2.5). This is done to maintain some of the integrity of the previously trained network.

(6) Repeat steps 2-5 until there are no more groups remaining in the generalization data set.

For more complex networks, the initial training data set should include examples that span both a broad set of operating conditions and the entire distribution of output responses. Such a range eliminates many of the problems associated with extrapolated predictions and new training examples outside previously modeled operating limits. Sometimes, the network must be entirely retrained for new data sets that do not follow past pattern behavior. Retraining is necessary because drastic modification of the internal model representation requires large learning rates and momentum coefficients, which in turn destroy all past network learning patterns.

G. Summary of Practical Aspects of Neural Computing

This section summarizes some of the recommendations for developing neural networks presented in Section 2.3:

- Use the zero-mean normalization method, equations (2.56) and (2.57), where the input and output vectors are normalized to between -1 and +1 with an average value of 0.
- Use a Gaussian distribution for setting initial values of weight factors.
- For classification networks such as those in Chapter 3, use the radial basis function as the transfer function.
- For prediction problems such as those in Chapter 4 and process-forecasting problems such as those in Chapter 5, use the hyperbolic tangent transfer function.
- Use 30 nodes in hidden layer 1 and 15 in hidden layer 2 as the initial architecture for most prediction and process-forecasting networks.
- Generate a learning curve to test the performance of neural networks.

2.4 Standard Format for Presenting Training Data Files and Neural Network Specifications

In this section, we present the standard format used throughout this text for describing the data file and neural network specifications. These standard formats provide all the specifications required for developing a functioning neural network and reproducing all the results reported for the illustrative case studies in this text.

The disk provided with this text includes all data files required to solve the

illustrative case studies and practice problems. All the variables in each data file have been normalized into a form directly usable by neural network software that does have normalization capability (such as NeuralWares Professional II/PLUS and NeuralWorks Explorer (1993). Table 2.7 shows the standard format for the presentation of file specifications. Every data file is composed of m input variables and n output variables, with each variable occupying the column of the data file denoted in the table. The Variable Type identifies whether the respective column of the data file is an input or output variable. A brief description of the variable appears under Normalized Variable. The File Name column is only present when a network has multiple files, which occurs when one set of input variables is used to predict multiple output variables and we choose to train a network for every variable. In this case, the output Column Number will have a, b, ... attached to the respective column to indicate the different training files. The Normalization Factor shows how the original data set is normalized (e.g., by zero-mean normalization) and what normalization factors are used, so that the reader can easily convert the data back to its original form.

Table 2.7. The standard format of file specifications.

Column number	Variable type	Normalized variable	File name	Normalization factor
1	input	$x_{in,1}$		$N(x_{in,1})$
2	input	$x_{in,2}$		$N(x_{in,2})$
•	input	•		•
•	input	•		•
m	input	$x_{in,m}$		$N(x_{in,m})$
(m+1)a	output	$x_{out,1}$	out1.nna	$N(x_{out,1})$
(m+1)b	output	$x_{out,2}$	out2.nna	$N(x_{out,2})$
•	output	•	•	•
•	output	•	•	•
(m+1)e	output	$x_{out,n}$	outn.nna	$N(x_{out,m})$

or

m+1	output	$x_{out,1}$		$N(x_{out,1})$
m+2	output	$x_{out,2}$		$N(x_{out,2})$
•	output	•		•
•	output	•		•
m+n	output	$x_{out,n}$		$N(x_{out,m})$

Table 2.8 shows the standard format we use to present the network specifications. These specifications include two sections: (1) the global network structure, which describes the type of the network, the transfer function, the learning rule, etc.; and (2) the learning schedule, which specifies the values of learning rate, momentum coefficient, etc., for each layer. We briefly describe all the components of the network specifications below.

- *Network type*: The kind of network or the algorithm used for training (e.g., backpropagation network, radial-basis-function network, etc.).
- *Training file name*: The name of the data file(s) used for training.
- *Transfer function*: The function each node uses to transform the weighted sum of the inputs into an output response (e.g., sigmoid, hyperbolic tangent, etc.). Some network types (e.g., radial-basis-function networks) will have multiple rows for every layer because they use different transfer functions. See Section 2.1.A.4 for background on transfer functions and Section 2.3.E for practical information on selecting an appropriate transfer function.
- *Learning rule*: The mathematical algorithm used for training the network (e.g., delta rule, K-means clustering, etc.). See Section 2.2.C and 2.2.D for background on the delta rule used in backpropagation networks and Section 3.2 for the K-means clustering and delta learning rules used in radial-basis-function networks.

- *Summation*: The type of weighted summation used on the inputs entering a node (e.g., sum, product, Euclidean, etc.). The standard summation is shown in Figure 2.1 of Section 2.1.A. The Euclidean summation used for the radial-basis-function network is described in Section 3.2.
- *Error*: The term used for adjusting weight factors in the training algorithms (e.g., standard, quadratic, cubic, etc.). See Sections 2.2.C and 2.2.D for a description of the error term in the backpropagation learning algorithms.
- *Weight-factor distribution*: The initial distribution of the weight factors that connect the nodes of the network (e.g., Gaussian, uniform, etc.). See Section 2.1.A.2 for background on weight factors and Section 2.3.C for practical information on selecting an initial weight-factor distribution..
- *Training iteration*: The training iteration at which the learning schedule adjusts the network training parameters (e.g., learning rate, momentum coefficient, noise, etc.). The training iteration number is high-inclusive (e.g., for hidden layer 1, a learning rate of 0.3 is used for iterations 1 through 10,000; a rate of 0.15 is used for iteration 10,001 to 30,000; and so on). Section 2.3.D. describes the learning schedule in greater detail.
- *Noise*: An extraneous signal added to input variables to avoid local minimums. The noise parameter is not required for the problems in this text, but may be required for some complex problems.

- *Learning rate*: A parameter (positive constant) that controls the rate at which the weight factors are adjusted. See Section 2.2.D for background on the learning rate and Section 2.3.D for practical information on selecting the proper learning rates.
- *Momentum coefficient*: A parameter between 0 and 1 used to increase the rate at which the weight factors are adjusted. This term is used in conjunction with the learning rate to reduce training time and help avoid local minimums. See Section 2.2.D for background on the momentum coefficient and Section 2.3.D for practical information on selecting effective momentum coefficients.
- *Error tolerance*: An error value low enough for the network to consider it zero.

Table 2.8. The standard format for presenting network specifications.

Network type			
Training file name			
Transfer function (input layer)			
Transfer function (hidden layers)			
Transfer function (output layer)			
Learning rule			
Summation			
Error			
Network weight distribution			
	Input Layer		
Training iteration	5,000		
Noise	0		
Learning rate	0.9		
Momentum coefficient	0.6		
Error tolerance	0		
	Hidden layer 1		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.3	0.15	0.04
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	Hidden layer 2		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.25	0.13	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	Hidden layer 3		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.2	0.1	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	Output layer		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.15	0.08	0.02
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

2.5 An Illustration of Developing a Neural Network Model Using a Commercial Software Package on Personal Computers

This section demonstrates how to develop and implement neural networks on commercial software packages using NeuralWare's Professional II/PLUS (1993). We show how easy it is to modify many of the key features of neural networks, such as weight-factor distribution, learning rate, momentum coefficient, transfer function, etc. (Section 2.3).

NeuralWare's Professional II/PLUS (1993) is a user-friendly software package that has the capability to develop most types of networks presently being used (e.g., backpropagation, radial-basis-function, etc.). This software provides a clear graphical depiction of the network structure, numerous decision boxes, and pull-down menus. NeuralWare also provides a very inexpensive software package, NeuralWorks Explorer, for learning and applying basic concepts of neural networks.

In the following discussion, we develop a simple backpropagation network with 4 input variables and 3 output variables on this software.

A. Constructing the Network

Constructing a neural network begins with selecting the network type through an InstaNet menu (Figure 2.23). For this example, we select the backpropagation network. This tool automatically builds a network which uses the backpropagation algorithm for training.

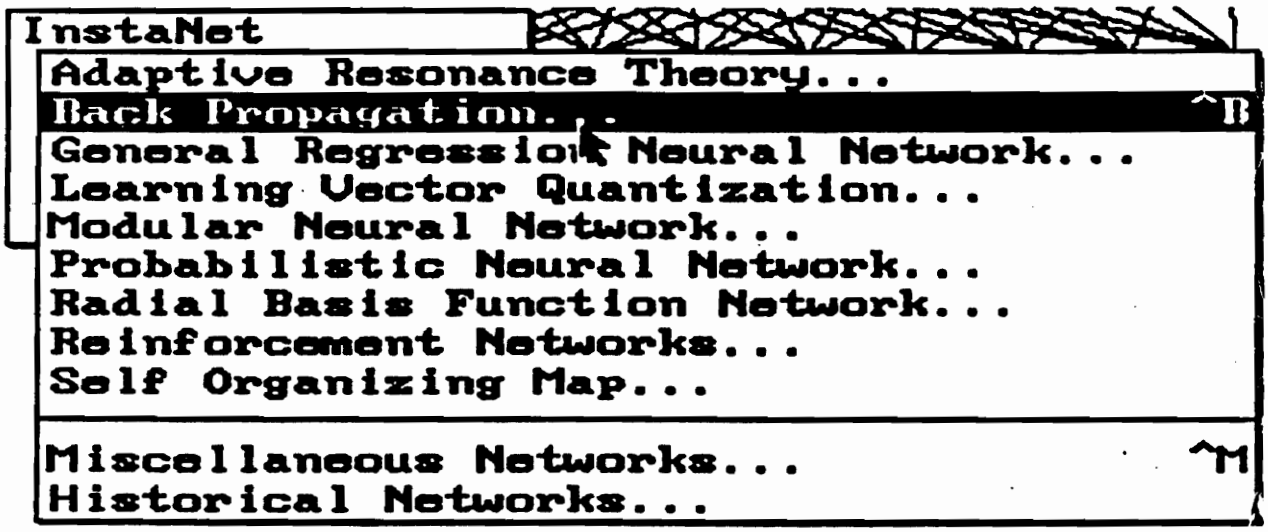


Figure 2.23. The InstaNet menu of NeuralWare's Professional II/PLUS.

Figure 2.24 shows the Backpropagation dialogue box brought up based on that selection. This dialogue box contains all the key parameters needed to develop a functioning backpropagation network with up to 3 hidden layers. We begin by entering the number of nodes or processing elements (PEs) in each layer. For this network, there are 4 PEs in the input layer, 4 PEs in hidden layer 1, 0 PE in hidden layer 2, 0 PE in hidden layer 2, and 3 PEs in the output layer.

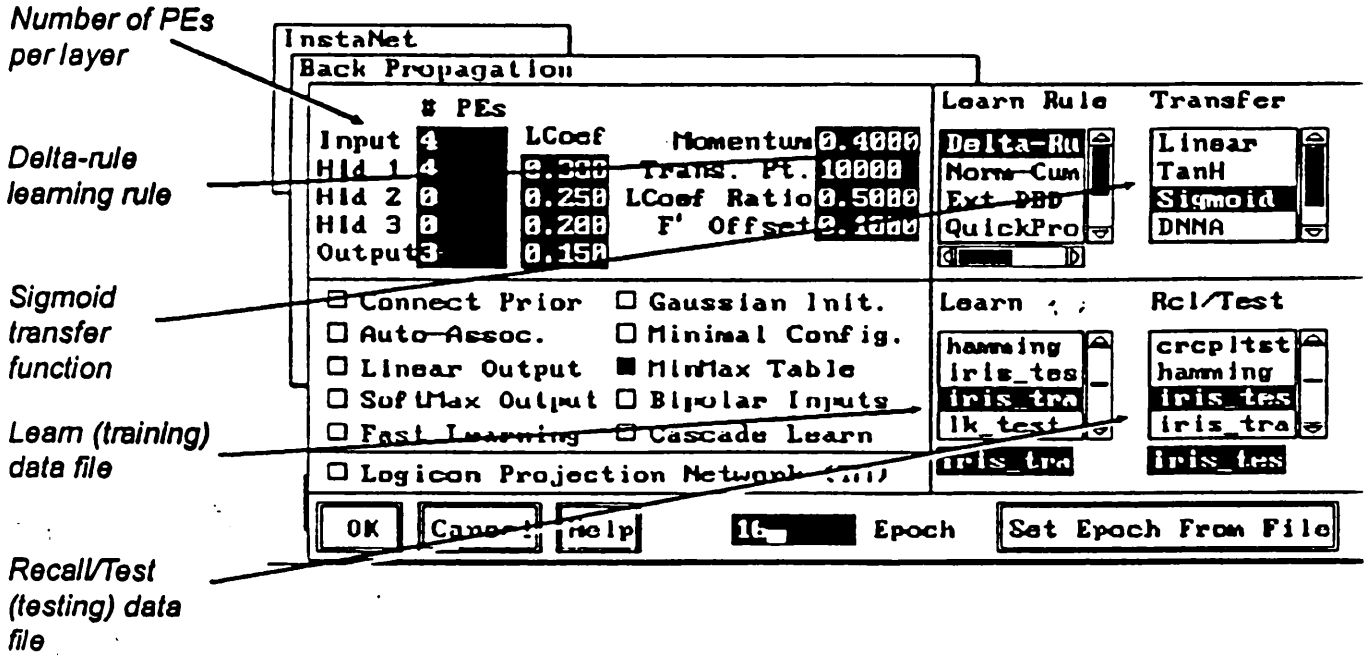


Figure 2.24. The backpropagation dialogue box of the InstaNet backpropagation builder.

We then set the initial learning rates or coefficients (LCoef) for the three hidden layers and the output layer (see Section 2.2.D and 2.3.D for further details on learning rate). Normally, we decrease the initial learning rates as we move through the network (e.g., 0.30 in hidden layer 1, 0.25 in hidden layer 2, 0.20 in hidden layer 3, and 0.15 in the output layer).

The next section of the dialog box includes momentum coefficients, transition point, learning-coefficient ratio, and F' offset. As with the learning rates, we use these

terms to generating a learning schedule. As noted previously in Table 2.4 of Section 2.3.D, default values of 0.3 and 0.4 for learning rate and momentum coefficient, respectively, are effective for training backpropagation networks. We use the remaining variables in this group to set the rest of the learning schedule. The transition point, i.e., iteration number 10000, identifies the first transition point when the learning rate and momentum coefficient are decreased (see Table 2.5 of Section 2.3.D for a detailed learning schedule). The additional transition points can be set in the learning schedule (described later in Section 2.5.B.4). The learning-coefficient ratio (LCoef Ratio) denotes the ratio by which the learning rate decreases at the transition points of the learning schedule. The F' offset is a constant applied to the derivative of the sigmoid and hyperbolic tangent transfer functions, so that saturated nodes continue to learn.

Next, we select the learning rule and the transfer function used to train the network. For this example, we choose the delta-rule algorithm (described in detail in Section 2.2.D) and select the sigmoid transfer function (see Section 2.A.4 and 2.3.E for more information on transfer function).

The training (Learn) and recall/testing (Rcl/Test) data files are identified in the bottom-right corner of the dialogue box. In this example, we use data files *iris_tra.nna* and *iris_tst.nna* to train and test the network, respectively. The most important of the remaining specifications is the MinMax Table (see Section 2.4.B.3), which normalizes the input and output variables for the user. The remaining specifications are less important to this example and we shall not discuss them.

Once the dialogue box is complete, we select "OK" and the backpropagation builder will prompt the user to select the desired instruments (graphs and tables) for analyzing the progress of the network training. Figure 2.25 shows the instrument menu which includes an RMS error, network weights, classification rates, and confusion matrices. "RMS error" plots the RMS error of the output layer throughout the training process. "Network weights" show the distribution of the weight factors (see Sections 2.1.2 and Section 2.3.C). "Classification rate" is a table showing the percentage of correctly matched classifications. "Confusion matrix" gives a scatter plot of the predicted versus the desired output responses. In this example, we only use the first three instruments.

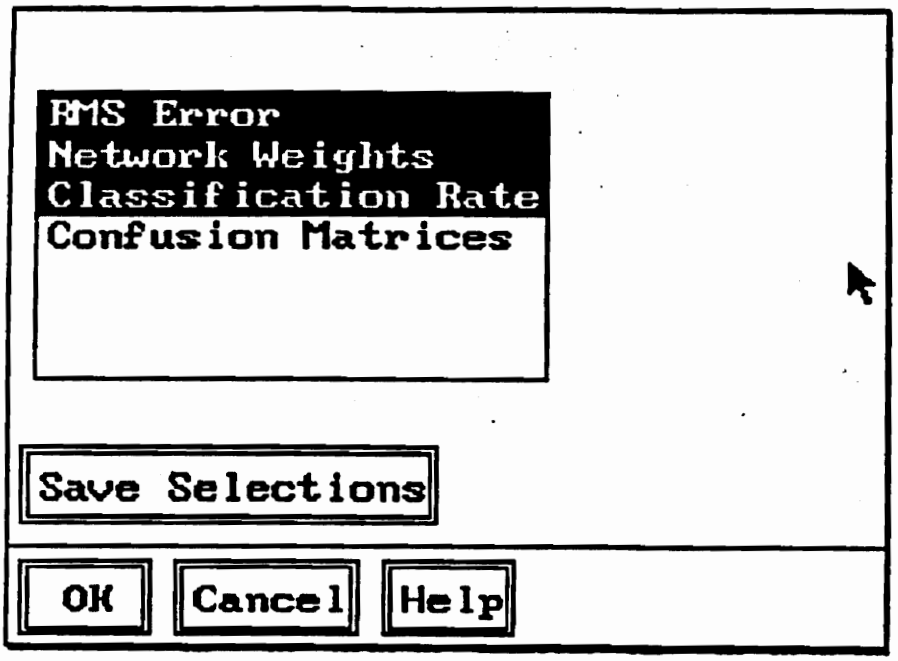


Figure 2.25. The instrument menu of the InstaNet backpropagation builder.

With the instrument menu completed, the backpropagation builder constructs the network and the instruments for graphically representing its training. Figure 2.26 shows the visual representation of the backpropagation network and the three accompanying graphs. At this point, after we complete 3 user-friendly menus, the network has all required specifications and is ready for training.

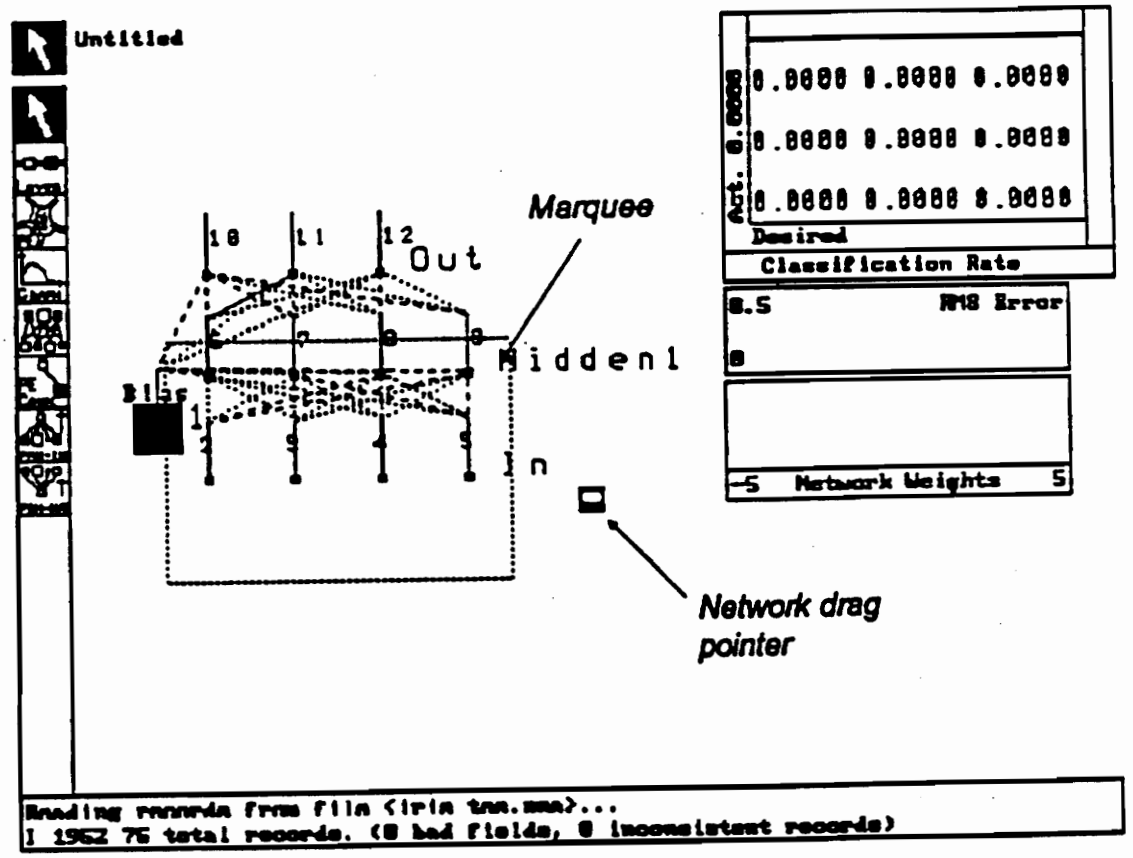


Figure 2.26. A visual representation of the backpropagation network and three accompanying graphs.

B. Important Features in NeuralWare's Professional II/PLUS

This section details some of the additional features of NeuralWare's Professional II/Plus that are important in designing more complex networks. In general, the previously outlined procedure is adequate for generating a functioning backpropagation network, but the additional features introduced here show how to alter many of the practical elements of neural networks for optimal performance. Hence, the following subsections describe many of the important dialogue boxes used for modifying those elements.

1. Processing Element (PE) /Edit Dialogue Box

Figure 2.27 shows the processing element (PE)/Edit dialogue box. Within this menu, we can examine all the main elements of a node. Each node has a name and number corresponding to its location in memory. Along the left side of the menu, we see the value of the summation (Sum), output value of the transfer function (Tran), the output value of the node (Output), the desired output value of the node (Desired), the backpropagated error (Error), the raw error field (Cur Err), and the scaling factor for the error (Err Fac). From this menu, we can use Disable PE and/or Disable Learn to freeze the output of the node and prevent the altering of the weight factors entering that node.

The weight factors between the specified node and the source nodes appear in the window on the right side of the menu. For each weight factor, the dialogue box displays

the source node (PE), the current input (Input), the weight (Weight), the weight type (Type), and the delta weight (Delta Wt). The weight type may be: (1) V - variable weight, which is active or is modified during training (normal setting); (2) F - fixed weight, which does not change during training (used for some hybrid and hierarchical networks (Section 2.6.B) which are trained in segments); (3) M - Mod weight, which is fixed during training, but is modified by a Mod factor; (4) S - set weight, which is similar to Mod weight except that it is multiplied by an input clamp; and (5) D - disabled, which disables the weight factor.

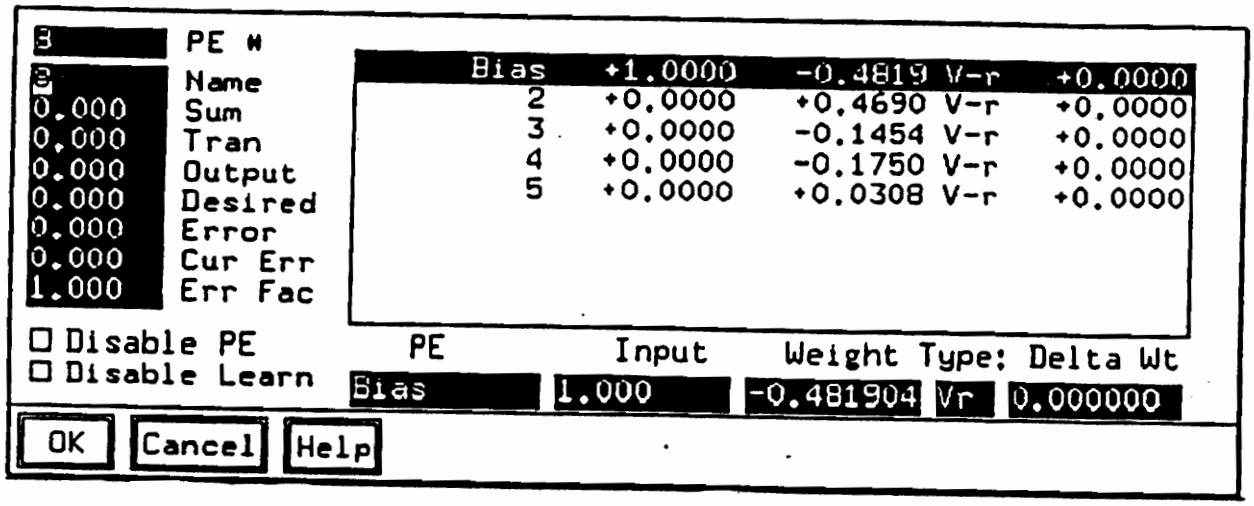


Figure 2.27. The processing element (PE)/Edit dialogue box of NeuralWare's Professional II/PLUS.

2. Layer/Edit Dialogue Box

Figure 2.28 shows the layer/edit dialogue box that contains the mathematical functions and learning rules for each layer. In the upper left corner, we see layer name (Name), number of nodes or processing elements in the layer (# PEs), number of rows used to represent the hidden layer on the screen (# rows), number of weight fields (# wgt fields), and spacing between the nodes on the screen. The number of weight fields is used to assign memory to the interconnections (and is automatically set based on learning rule).

The lower left section of this dialogue box includes: (1) shape of the nodes; (2) scale and offset used to transform data between different ranges; (3) low and high limits that set minimum and maximum values for nodes of the layer; (4) F' offset, is an offset applied to the derivative of the sigmoid and hyperbolic tangent transfer functions to allow saturated processing elements to continue to learn; and (5) initial low and high limits which set for the weight-factor distribution during initialization (Section 2.3.C explains the importance in setting this distribution).

The seven scroll windows on the right side of the dialogue box are used to select a variety of mathematical functions and learning rules that control the training process: (1) type of weighted summation; (2) transfer function; (3) output function used to determine who the winner is in competitive learning (for backpropagation, set to direct or no competition); (4) method of error transformation (e.g., standard, quadratic, cubic, etc.);

(5) learning rule used to adjust weight factors; (6) learning/recall schedule file name (see Section 2.3.D for the importance of the learning schedule); and (7) type of noise function added to the summation value before entering the transfer function.

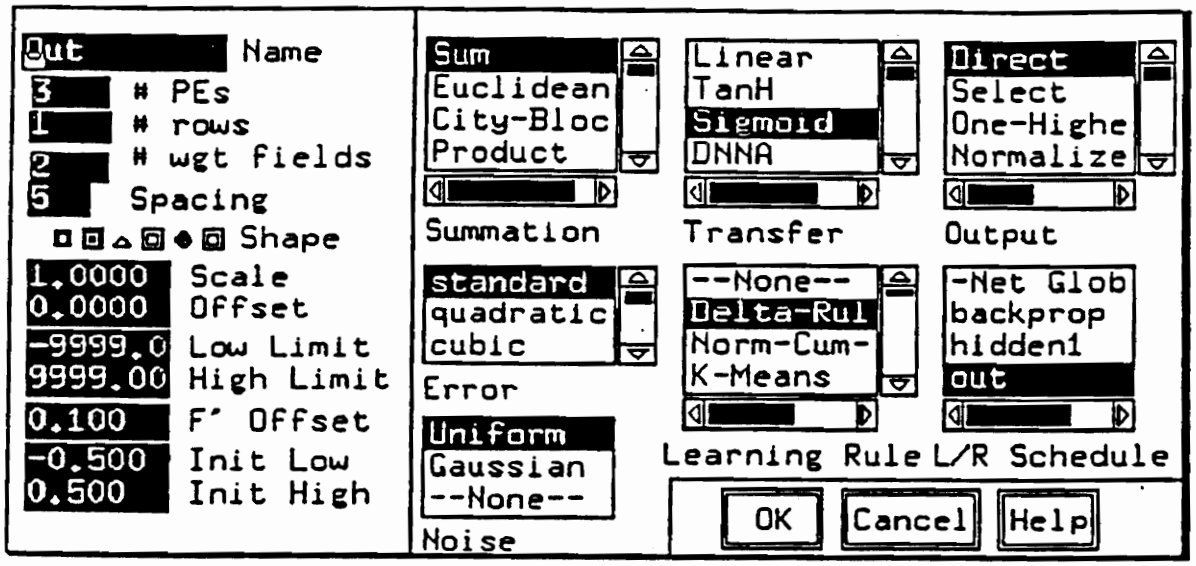


Figure 2.28. The layer/edit dialogue box of NeuralWare's Professional II/PLUS.

3. Input/Output Parameter Dialogue Box

Figure 2.29 is the input/output parameter dialogue box. In the upper left-hand corner, the learn source (Lrn) and the recall/testing (Rcl/test) source identify the process for reading the training and recall/testing data files into the network (e.g., data sets read into the network randomly or sequentially). The data files are selected in the upper right-hand

scroll windows with either **.nna* (ASCII text file) or **.nnb* (binary file) extensions. This dialogue box also allows for the inclusion of user-defined input/output programs (UIO). The user IO facility within NeuralWare's Professional II/PLUS is very powerful way to link programs written in "C" to the network-training programs, but is beyond the scope of this text.

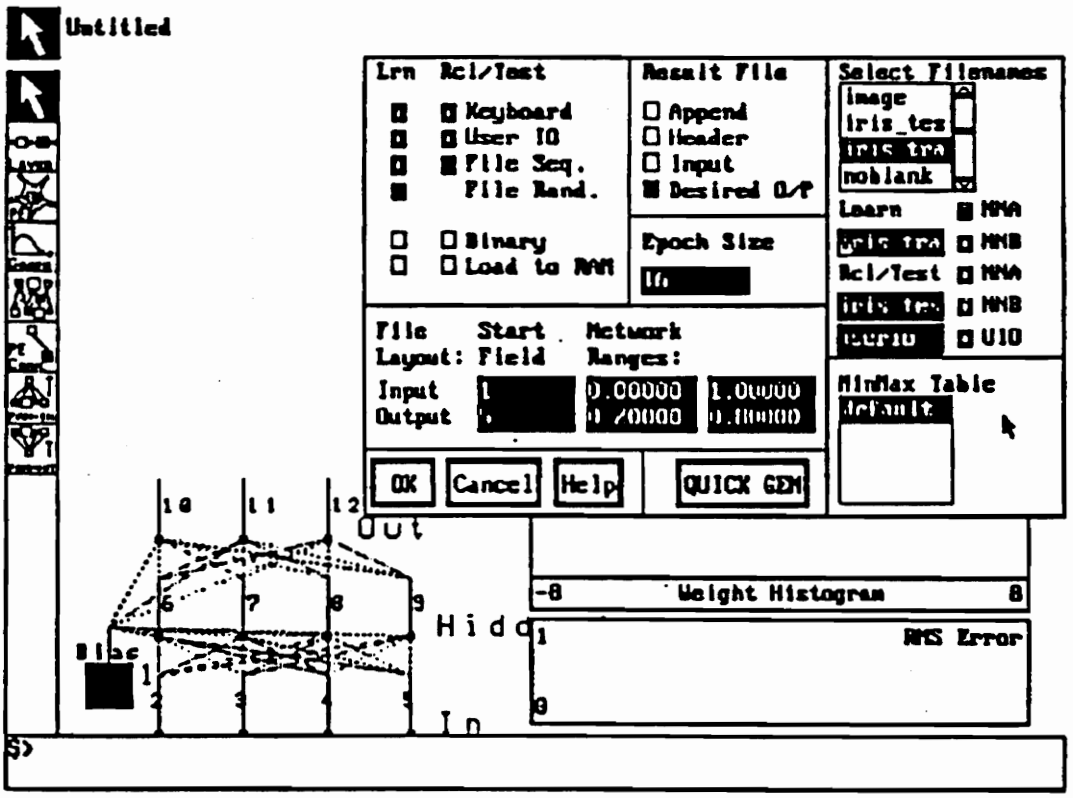


Figure 2.29. The input/output parameter dialogue box.

The most important section of the input/output dialogue box is the MinMax table, located in the lower-left section. The MinMax table is used to normalize the input and output data for the user. In this example, the input variables are normalized from 0 to 1 and the output variables are normalized from 0.2 to 0.8 (for additional information on normalization methods, see Section 2.3.B). Note that if we used the hyperbolic tangent transfer function, we would switch the normalization limits to -1 and +1 for the input and -0.8 and 0.8 for the output variables (these two sets of normalization ranges are set as default values for the sigmoid and hyperbolic tangent transfer functions within this software package).

4. Learning Schedule

We can modify the learning schedule (Section 2.3.D) for adjusting learning rates and momentum coefficients in the learning/recall dialog box (Figure 2.30). The major section of the learning schedule is the table consisting of five iteration check points (Learn Count) and four rows labeled temperature (noise function), learning rate, momentum, and error tolerance. For this learning schedule, the learning rate decreases from 0.2, to 0.1 at iteration 1000, then to 0.01875 at iteration 7000, and finally to 0.00117 at iteration 150000. The other parameters that affect the learning algorithm are reduced in a similar fashion.

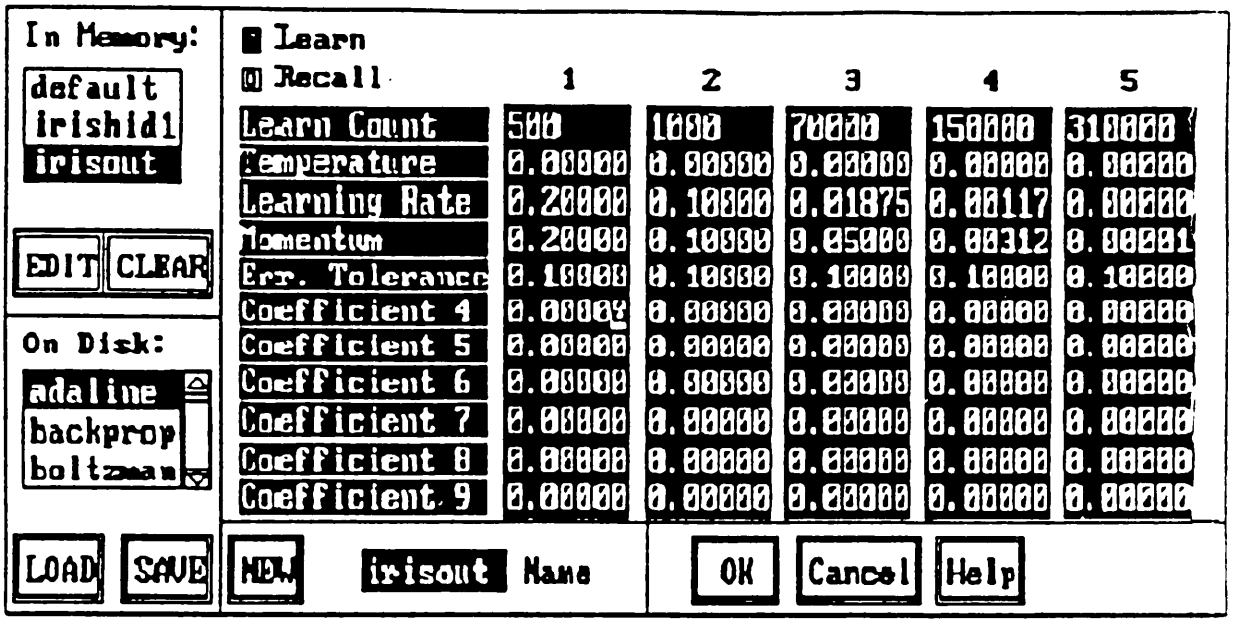


Figure 2.30. The learning/recall schedule dialogue box.

C. Training and Testing Data File Format

There are normally two data files required for developing a neural network: (1) a file for training the network (*file : iris_trn.nna*); and (2) a file for testing the generalization of the network (*file: iris_tst.nna*). As discussed in Section 2.5.B.3, we specify the data files used for training and testing in the input/output parameter dialogue box. The data files are normally presented to the network in ASCII file format (**.nna*) and then converted to

binary format (*.*nnb*) for the actual training. To illustrate the format of a data file, we list the first 10 data sets of the file *iris_tra.nna* below:

0.224	0.624	0.067	0.043	1.0	0.0	0.0
0.749	0.502	0.627	0.541	0.0	1.0	0.0
0.557	0.541	0.847	1.000	0.0	0.0	1.0
0.110	0.502	0.051	0.043	1.0	0.0	0.0
0.722	0.459	0.663	0.584	0.0	1.0	0.0
0.776	0.416	0.831	0.831	0.0	0.0	1.0
0.196	0.667	0.067	0.043	1.0	0.0	0.0
0.612	0.333	0.612	0.584	0.0	1.0	0.0
0.612	0.416	0.812	0.875	0.0	0.0	1.0
0.055	0.584	0.067	0.082	1.0	0.0	0.0

This example has seven columns - four input columns and three output columns, each separated by at least one space. Each of the actual data files for the training and testing data files contain 75 training examples.

D. Training and Testing of a Backpropagation Network

We will now demonstrate how to train and test the network. Figure 2.31 shows the run/learn dialog box used to initiate the network training. To begin the process, we enter the number of desired training iterations into the input field and select "OK".

As training proceeds, the software allows us to monitor RMS error of the output layer to see how well training is progressing (Figure 2.26). Figure 2.32 shows the RMS error plot for 20000 training iterations.

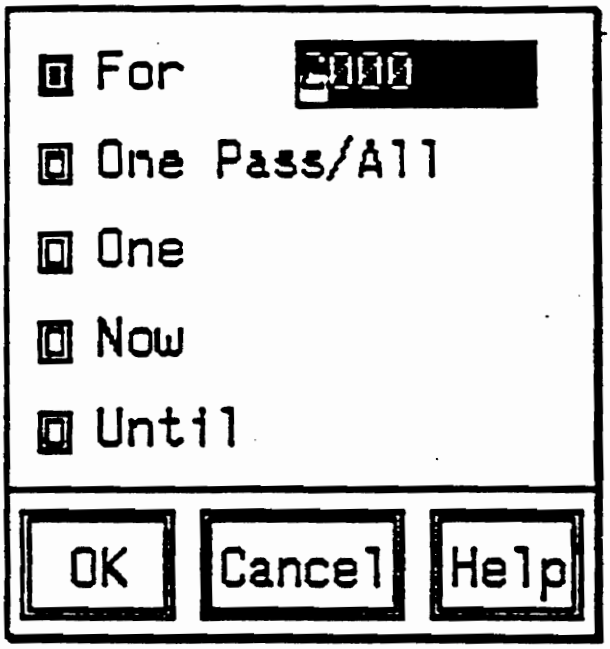


Figure 2.31. The run/learn dialogue box.

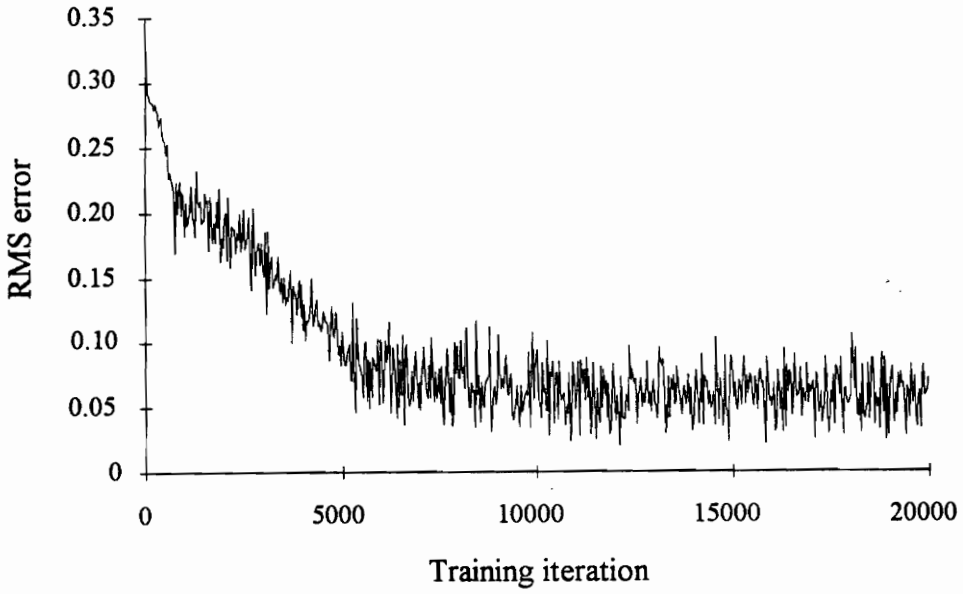


Figure 2.32. The RMS error plot for 2000 iterations.

When training is complete, we use the run/test dialogue box to test the recall and generalization of the network (Figure 2.33). We select one pass/all so that the network will test every example in the data set.

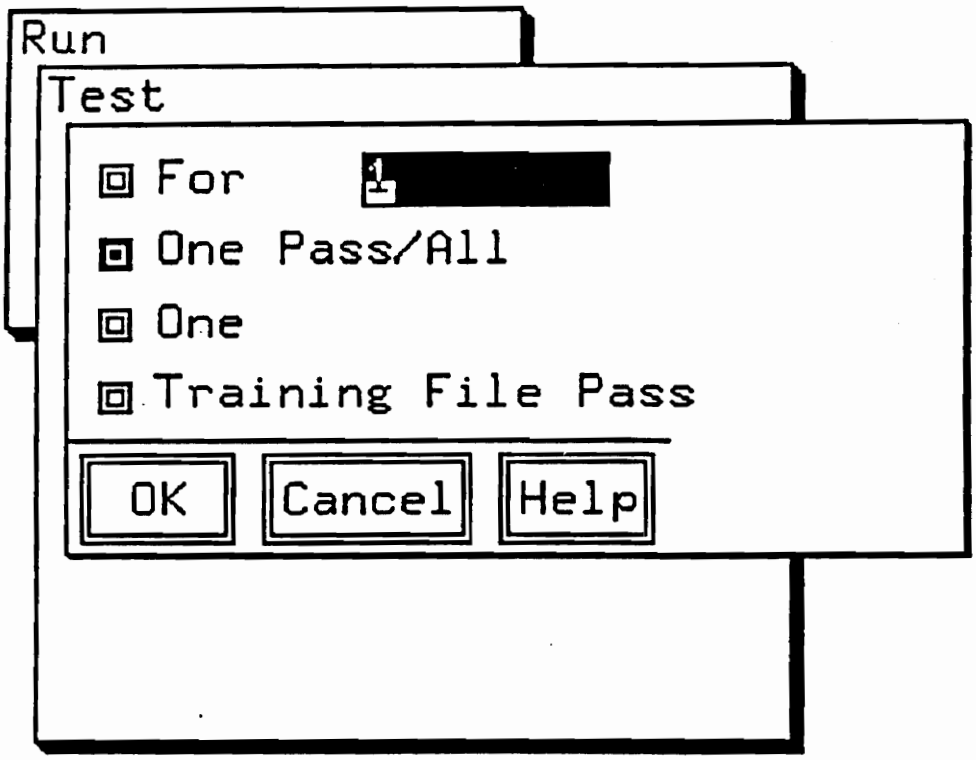


Figure 2.33. The run/test dialogue box.

During the testing of a given data set, the software generates a result data file (**.nnr*). For this example, we list the first 10 lines of the result file (*iris_tst.nnr*) using the generalization data set (file : *iris_tst.nna*):

1.000000	0.000000	0.000000	0.956902	0.050952	-0.015563
0.000000	1.000000	0.000000	-0.001416	0.974402	0.029922
0.000000	0.000000	1.000000	-0.000849	0.044788	0.952744
1.000000	0.000000	0.000000	0.984779	0.011740	-0.003656
0.000000	1.000000	0.000000	-0.005641	0.906410	0.094142
0.000000	0.000000	1.000000	-0.002823	0.116047	0.887197
1.000000	0.000000	0.000000	0.993681	0.005974	0.000502
0.000000	1.000000	0.000000	0.004225	0.928773	0.065452
0.000000	0.000000	1.000000	-0.008236	-0.031227	1.035764
1.000000	0.000000	0.000000	0.993129	0.004033	-0.002540

The first three columns of the result file are the actual output responses for the generalization data file *iris_tst.nna*, and the following three columns are predicted output responses by the network (e.g., column 1 corresponds to column 4).

2.6 Introduction to Special Neural Network Architectures

This section provides an overview of a number of special neural network architectures that are important to applications in bioprocessing and chemical engineering. The details of these networks, together with application examples, are described in more detail in subsequent chapters.

A. Autoassociative Networks

An autoassociative network correlates an input pattern to itself. Autoassociative networks have two main applications in bioprocessing and chemical engineering: (1) data compression and filtering; and (2) dimensionality reduction of an input vector. We can use data-compression and filtering networks to extract "noise-free" patterns in an input signal. For example, in process forecasting, we frequently apply data-compression networks to reduce the noise in time-dependent process variables (see Section 5.2). Dimensionality reduction of an input vector reduces the number of nodes (input variables) that enters the first layer of an autoassociative network, improving both training efficiency and prediction capability. Section 2.6.B.2 illustrates a common application of an input-compression network.

Figure 2.34 shows the standard architecture for a three-layer autoassociative network for data compression or dimensionality reduction, where the vectors **In(m)**, **B(n)**, and **Out(m)** represent the input layer, hidden layer, and output layer of the network, respectively. This network is identical to the backpropagation network except that it correlates the input pattern with itself (**In(m) = Out(m)**), rather than with an output-response pattern . Therefore, we provide only the input vector and the network automatically defines the output vector for the user. Both the input and output patterns for a signal are represented as vectors of m incoming data points.

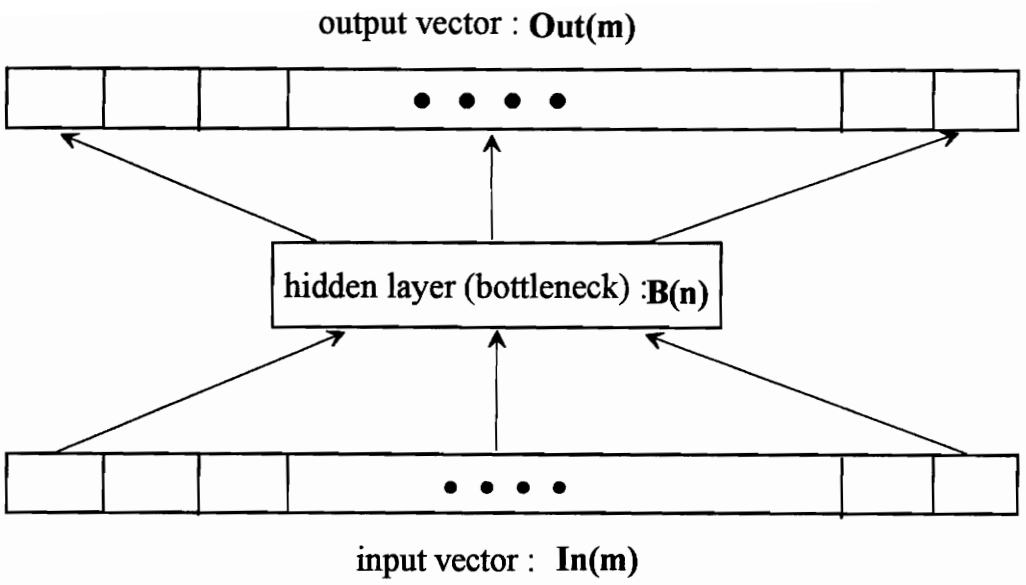


Figure 2.34. A three-layer autoassociative network.

The objective of compressing a data set is to map only the important information of the input vector onto the output vector, while removing all noise and errors. This is accomplished by first mapping the input vector onto a smaller intermediate vector **B(n)**, known as *a bottleneck*, having n nodes (where $m > n$). The bottleneck vector is a compact representation of the input pattern, and it is then mapped onto the output vector to achieve a more compact representation of the input vector. We can quantify the compression using a data-compression ratio, defined as the ratio of elements in the input vector (m) to elements in the bottleneck vector (n).

$$\text{compression ratio} = \frac{m}{n} \tag{2.61}$$

A higher compression ratio leads to increased noise and error reduction. If the compression ratio is too high, the bottleneck vector cannot sufficiently represent the input pattern and large errors in mapping the signal occur. However, with a low compression ratio (e.g., $m/n = 1$) the network maps the entire input pattern onto the output pattern without any eliminating any measurement noise or error. As a result, typical ratios are normally set between 2 and 8, depending on the signal type and noise in the system.

To easily determine the optimal compression ratio, we can generate a graph of RMS error versus compression ratio by varying the number of nodes in the hidden layer for a given training data set. In general, training data-compression networks is fairly fast and the graph can be generated in a reasonable time period. Figure 2.35 shows the standard type of curve when plotting the RMS error versus compression ratio. The optimal compression ratio is the point a which further increasing the ratio starts to produce significant increases in the RMS error.

We shall illustrate the autoassociative network applied to noise reduction of a cell-concentration signal in the time-dependent modeling of a batch fermentation process in Section 5.4.D.2.

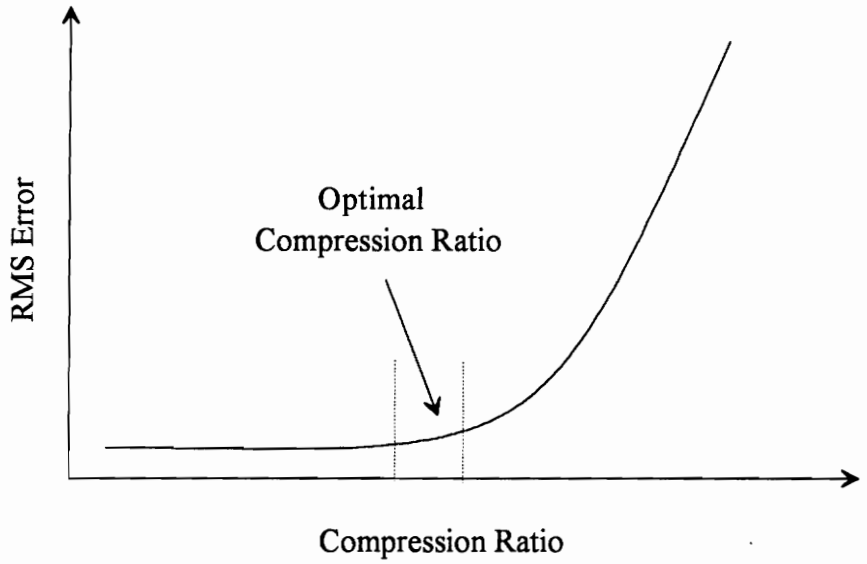


Figure 2.35. A plot of RMS error versus compression ratio used for the determination of the optimal compression ratio.

B. Hierarchical Neural Networks

In bioprocessing and chemical engineering, a very useful type of backpropagation network is the hierarchical neural network (Hecht-Nielsen, 1990; Mavrovouniotis and Chang, 1992), as shown in Figure 2.36. This architecture has several layers segmented into subnetworks, where the input vector is divided into groups that have similar effects on the output responses. The motivation behind dividing the input vector into groups is to capture the internal structure and special features of the input pattern. This structure allows the network to process the data in stages and compartmentalize independent effects

into their own sections. As the network progresses into the upper hidden layers, it takes into account higher-order interactions between the variables of different subnetworks. Each subnetwork is expected to summarize in its outputs the important features of the selected subset of the inputs. Moreover, key operating variables can appear in multiple subnetworks when required.

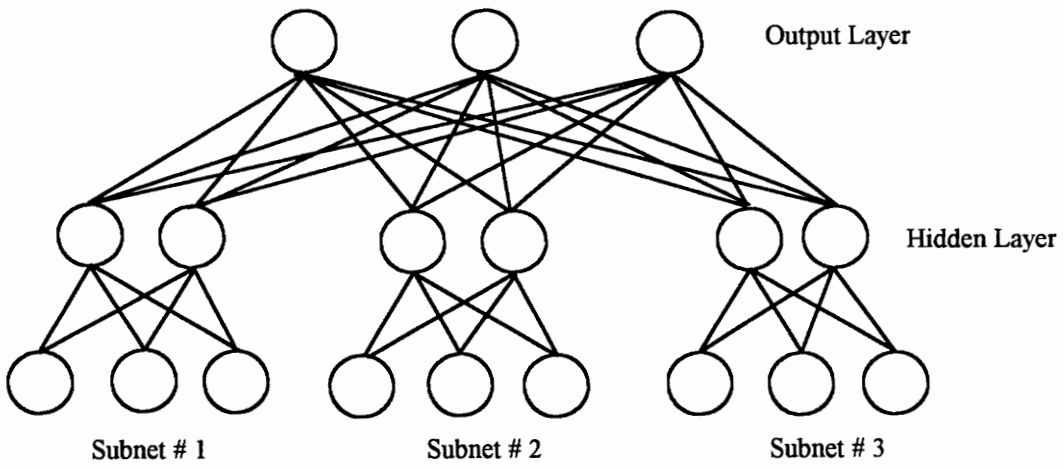


Figure 2.36. A hierarchical neural network architecture.

We turn to hierarchical networks because at times standard backpropagation network present certain difficulties when modeling complex systems in bioprocessing and chemical engineering:

(1) the representation does not predispose the network to any meaningful kinds of patterns.

(2) all patterns and relationships appear equivalent to the network.

(3) the structure of the network is too general.

The hierarchical network architecture overcomes these limitations, and it has several advantages over the standard backpropagation networks:

- Hierarchical networks use more nodes with fewer weight factors, improving efficiency. Therefore, network training requires fewer examples and less time. For example, a system with 200 input variables can have millions of weight factors, w_{ij}, making the network hard to train and very time-consuming. Dividing the input vector into subnetworks, compress it to 20 to 50 input variables in the first hidden layer (see Section 2.6.A.2), thereby reducing the number of weight factors significantly and allowing the network to be trained more efficiently.
- For complex systems, well-defined subnetworks of related variables provide hints that help the network learn in the right direction. The more structured layout gives the user a more understanding of what is occurring in the system.
- We can analyze the individual subnetworks to decipher what the different segments of the network have learned.

- Hierarchical networks are isomorphic to expert systems or model-based algorithms for the same task. We can map each useful subnetwork to a rule (or small set of rules) or a model-based local analysis.

The following sections describe two significant types of hierarchical networks used in bioprocessing and chemical engineering: (1) moving-window networks for time-dependent processes; and (2) input-compression networks for working with large input variable sets.

1. Moving-Window Networks

The moving-window network is a special hierarchical network used to model dynamic systems and unsteady-state processes. A moving window is a way to isolate subsets of a long string of time-dependent measurements, simply taking the last *n* time segments and using each segment as an input to a network. For training, a moving window provides a means for creating multiple training examples (patterns) from continuous raw data. For example, Figure 2.37 shows a process-trend scanning window for a system that has n input variables VARi(t) (i = 1 to n; t = 1 to T), where i represents the i^{th} variable, and t is the current time and T is the overall time period of the process. In addition to the current time t of the process, the window includes three past values at t-Δt, t-2Δt, and t-3Δt (VARi(t-3), VARi(t-2), VARi(t-1), and VARi(t)).

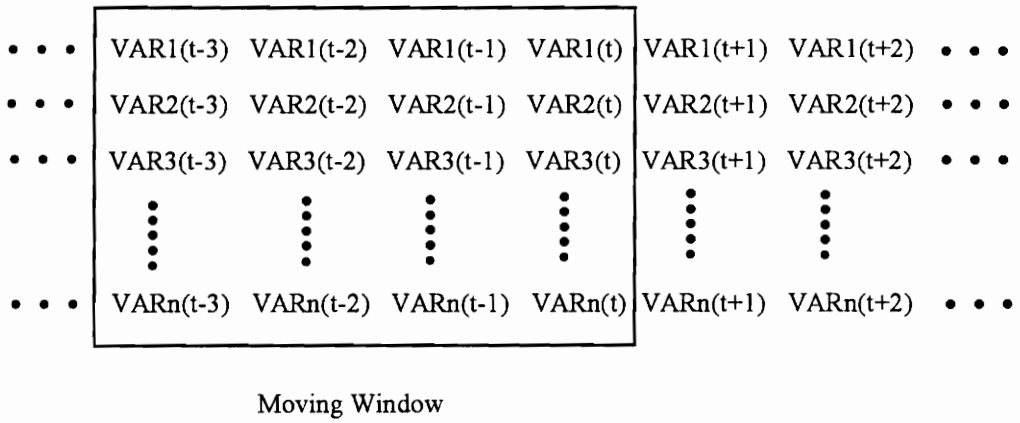


Figure 2.37. A process-trend scanning window

Figure 2.38 shows the architecture of a moving-window network. The input layer is expanded so that each input variable includes a range of values over a given time period, as demonstrated in the process-trend scanning window of Figure 2.37. Note that the number of time increments within a window can vary from one value to another based on the time constants of the variables. We use 4 time increments for each variable in Figures 2.37 and 2.38 for illustrative purposes, but in practice, VAR1 could have 3 increments, VAR2 5 increments, and so on. What remains constant from variable to variable is the total time span of the moving window, rather than the number of time increments for each variable. The three hidden layers and the output layer have the same formats as they would in the standard network type (e.g., the backpropagation network described in

Section 2.2 or the radial-basis-function network in Section 3.2). Hence, the moving-window structure effectively alters only the input layer.

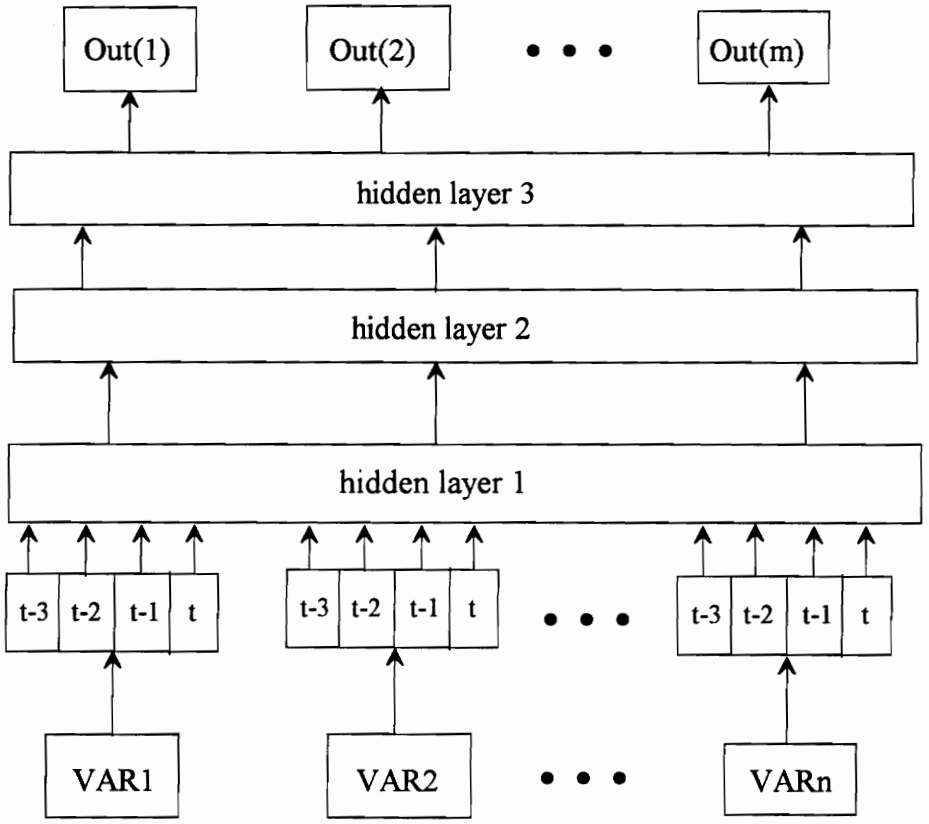


Figure 2.38. A moving-window network with a process-trend scanning window including 4 time increments.

The moving-window network is used for two types of applications in this book: (1) process fault diagnosis with unsteady-state systems (Section 3.4.D) ; and (2) process

forecasting, modeling, and control of time-dependent systems (Chapter 5). For additional information on time-dependent fault diagnosis, see Section 3.4.D.2 for an example using an unsteady-state continuous stirred-tank reactor (CSTR) system. In the predictive networks for time-dependent systems (Chapter 5), the moving window allows predictions from early process stages to be recycled and used for subsequent predictions as the process proceeds. This approach is known as a recurrent network for process forecasting, and is covered in Sections 2.6.C and 5.3 to 5.5.

2. Input-Compression Networks

The input-compression network is another important type of hierarchical network. Neural networks can become very large and difficult to train if they have a large number of input variables (e.g., >50 input variables) that influence the output responses. Therefore, decreasing the number of input variables significantly reduces the size of the network (i.e., number of weight factors), and improves the network's training efficiency and prediction capability.

Figure 2.39 shows the architecture of a typical input-compression network. The input layer consists of n property groups, where each group includes input variables with similar effects on the output responses. Each property group is compressed into a corresponding subnetwork to create a more compact representation of the input variables without losing any essential information. Therefore, the subnetworks essentially become

the input layer for training purposes, and 100 input variables can be reduced to 25 in the subnetwork layer. The remaining layers are set in the same configuration as in the type of network used, such as backpropagation networks (Section 2.2) or radial-basis-function networks (Section 3.2).

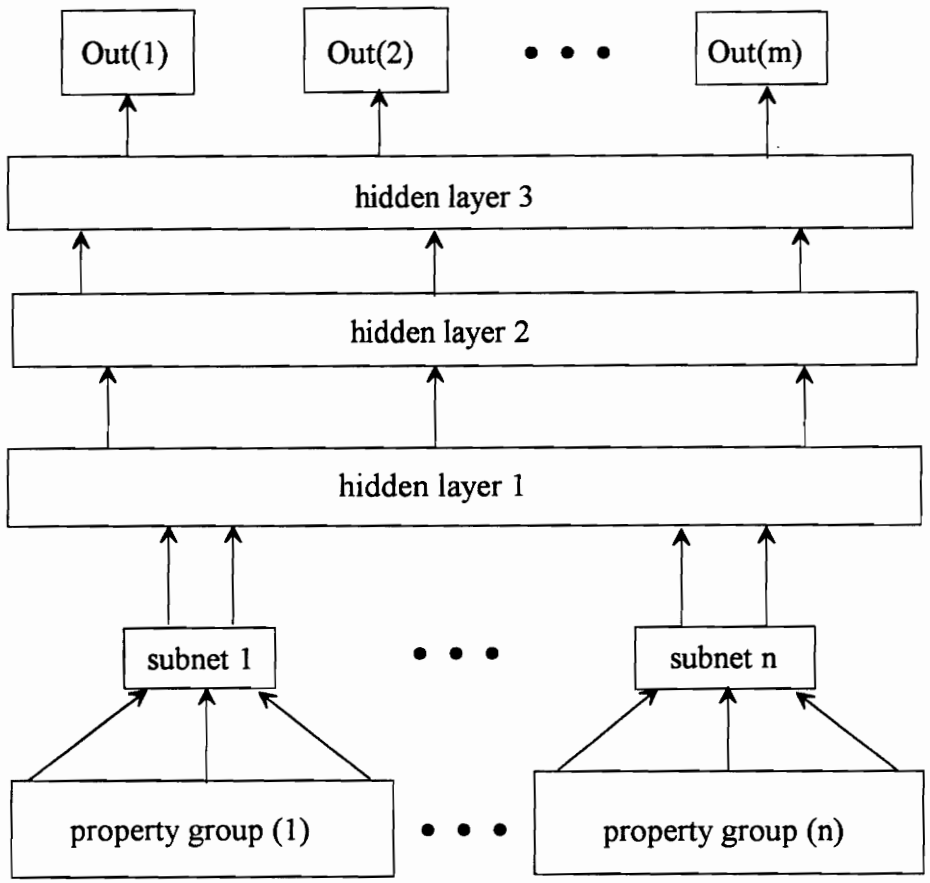


Figure 2.39. An input-compression network.

Developing an input-compression network involves two steps. First, we use an autoassociative network to reduce the dimensionality of the input vector. We then attach the input and hidden layers of the dimensionality-reduction network to a standard backpropagation network. The following two sections describe this process.

a. Dimensionality Reduction of Input Vector

The dimensionality-reduction network is an autoassociative network (Section 2.6.A) that maps a set of input variables onto itself through a hidden (bottleneck) layer that is significantly smaller than the input layer. The hidden layer is then a compact representation of the input layer. As described in Section 2.6.A, our main goal is to obtain the largest compression ratio (the ratio of the number of input-layer nodes to hidden-layer nodes), while preserving the relevant information in the input vector. To provide the input-compression network with more structural information, we can first divide the input variables into similar property groups based on their effects on the output variables. For example, we can create one property group for each raw material and for each processing stage. Note that the network would then require a separate dimensionality-reduction network for each property group.

Figure 2.40 shows the architecture of a dimensionality-reduction network for n input-variable subgroups. Every subgroup is trained independently, as described in Section 2.6.A, and we determine the optimal compression ratio by generating a graph of

RMS error versus compression ratio (Figure 2.35). These autoassociative networks are used in the following section to develop the input-compression network.

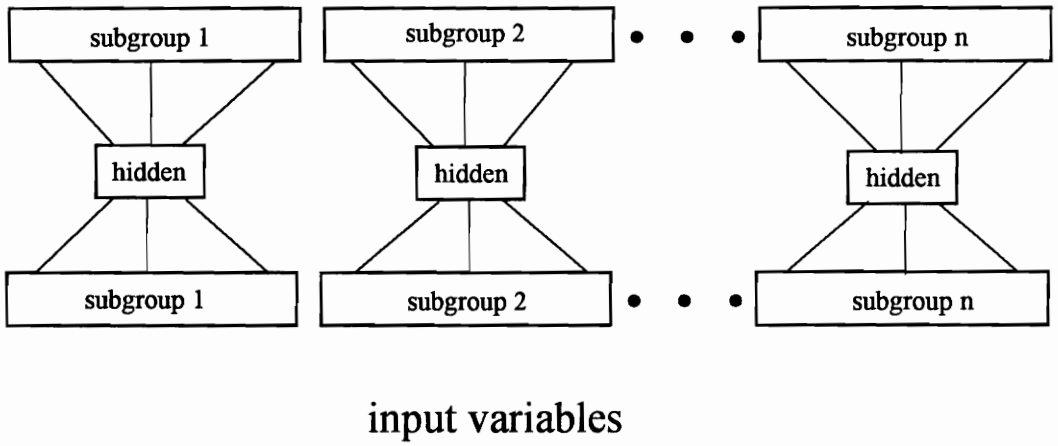


Figure 2.40. The architecture of a dimensionality-reduction network for n subgroups of input variables used in developing an input-compression network.

b. Attaching a Dimensionality-Reduction Network to a Backpropagation Network

Now that we have generated a dimensionality-reduction network for n input-variable subgroups, we will demonstrate how to attach these networks to a standard backpropagation network. Once again, note that the hidden layer contains a compact representation of the input layer and is used as the input to the input-compression network.

Figure 2.41 shows the detachment of the input and hidden layers from the previously trained dimensionality-reduction network. The weight factors between the input and hidden layers are fixed at values corresponding to the optimal compression ratios for the n input-variable subgroups. The dimensionality reduction provides a group of subnetworks considerably smaller than the original input-variable set.

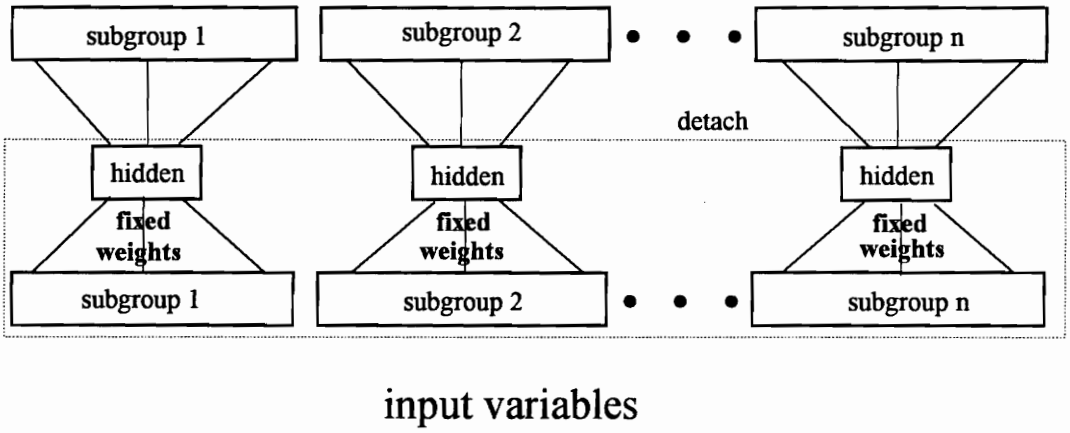


Figure 2.41. The detachment of the input and hidden layers from a dimensionality-reduction network and fixing of the weight factors for use in an input-compression network.

The output from the dimensionality-reduction network then becomes the input to a standard backpropagation network, creating a complete input-compression network. Figure 2.42 shows this network divided into a lower section containing a dimensionality-reduction network and an upper section containing a standard backpropagation network. As discussed, linking a dimensionality-reduction network (an

autoassociative network) and a backpropagation network allows us to more efficiently handle models with very large input-variable sets. Therefore, we significantly improve the prediction capability of the network, while reducing the size and training time of a standard backpropagation network.

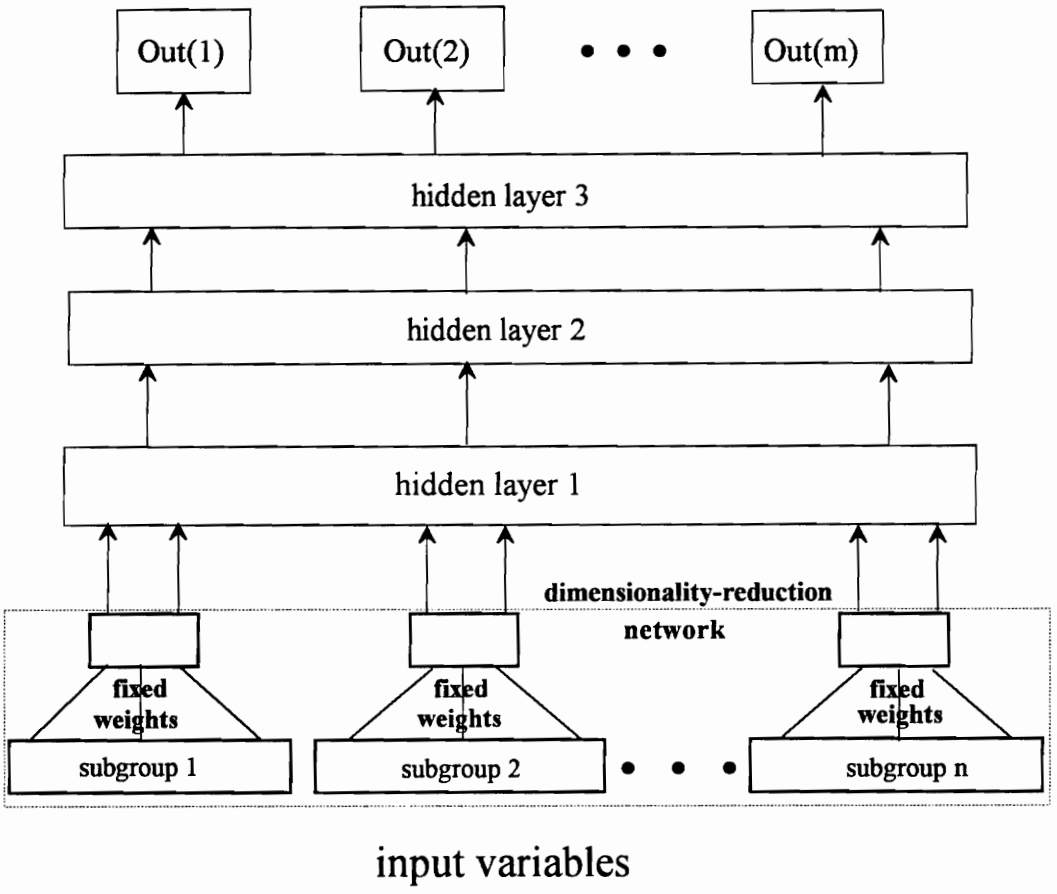


Figure 2.42. An input-compression network divided into a lower section containing a dimensionality-reduction network and an upper section containing a standard backpropagation network.

C. Recurrent Networks

A recurrent network combines the feedback and feedforward connections of neural networks (see Figure 2.8). In other words, it is simply a neural network with loops connecting the output responses to the input layer. Thus the output responses of the network function as additional input variables. This structure is critical for handling the time-dependent systems of Chapter 5.

Figure 2.43 shows a typical structure for recurrent networks. Within this network, we have a *single time-lag step* where the output responses, $y_j(t+1)$ (j = 1 to m), feed back through recurrent loops to the input layer, $y_j(t)$ (j = 1 to m), at the same time period as the input variables, $x_i(t)$ (i = 1 to n). Werbos (1988) has named this type of recurrent network *a time-lag recurrent network* or *an externally recurrent network*. For the initial prediction at time t_0, we must assign an initial conditions for the output response, $y(t_0)$, to predict $y(t_1)$. All future predictions, $y_j(t+1)$, are obtained from the input $x_i(t)$ and output $y_j(t)$ of the previous time period. Therefore, the network predicts the future responses of $y_j(t)$ from the initial condition of $y_j(t_0)$ and $x_i(t)$ at all time intervals throughout the desired time frame, t_0 to t_f (final time).

We provide a detailed description of a specific recurrent network for process forecasting in Sections 5.3 to 5.5. This type of network provides a modeling technique that can be used for process optimization and adaptive process control of time-dependent processes.

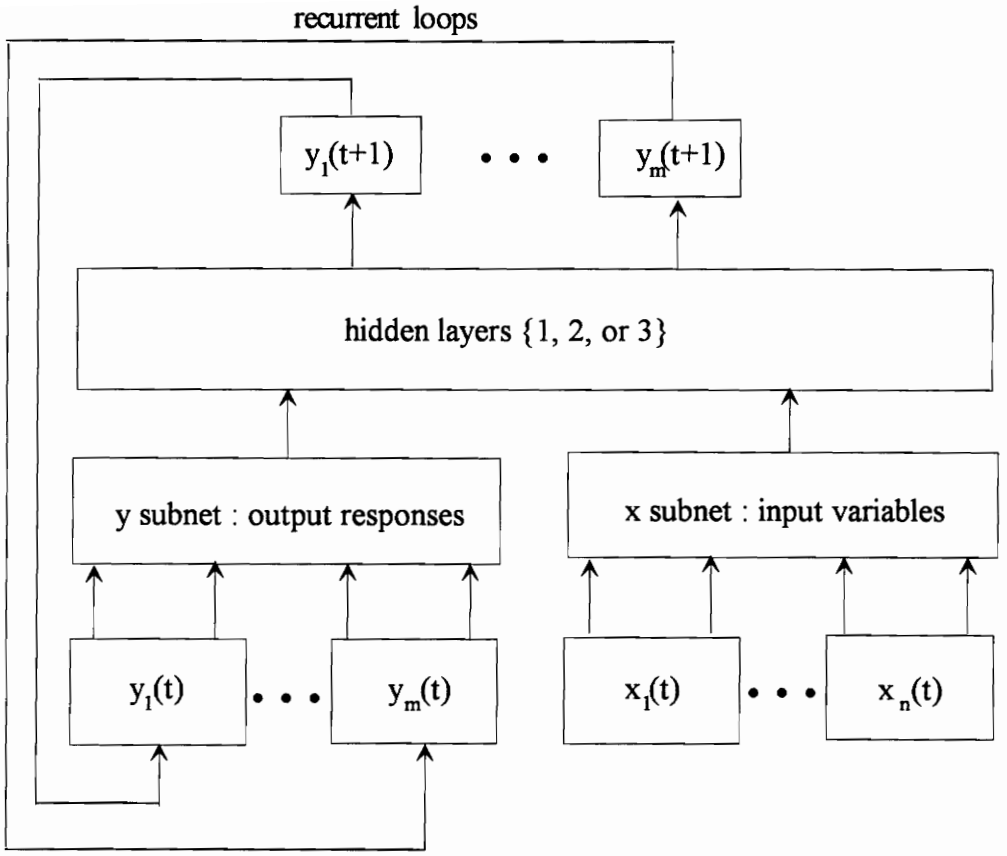


Figure 2.43. A typical recurrent network.

D. Neural-Fuzzy Networks

Fuzzy logic is becoming one of the more popular techniques for developing advanced control systems (Samdani, 1993). This technology provides a method to develop a simple and inexpensive controller for complex systems using time-dependent neural networks,

such as those in Chapter 5. These fuzzy systems can also be used to attach neural networks to expert systems, as shown in Chapter 6.

Neural-fuzzy networks implements fuzzy-logic inferencing through neural networks. In the following sections, we discuss fuzzy-logic systems and show how to attach a neural network to a fuzzy-logic system for both process control and expert systems.

1. Fuzzy-Logic Systems

Fuzzy logic grew out of a desire to quantify rule-based systems. Rule-based reasoning is grounded in qualitative knowledge representation, and fuzzy logic allows us to mesh a quantitative approach with the qualitative representation. It provides a way to quantify certain qualifiers such as *approximately*, *often*, *rarely*, *several*, *few*, and *very*. Figure 2.44 shows the relationship of fuzzy-logic systems to the two main areas of artificial intelligence (expert systems and neural networks) based on knowledge type and information framework. The knowledge type is divided into structured (based on rules) and unstructured, and the information framework is divided into symbolic and numerical, as described in Section 1.1A.

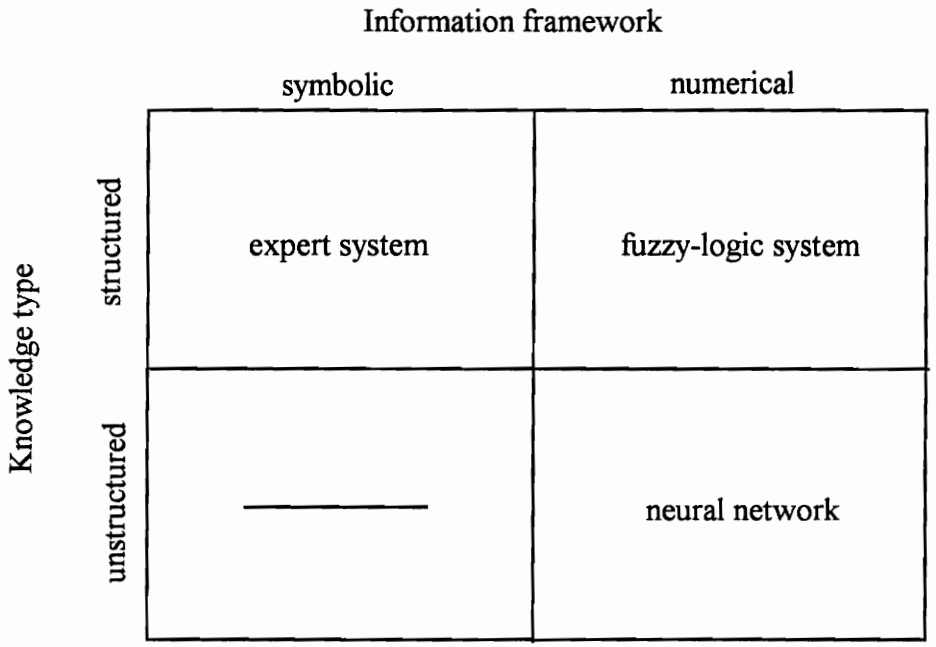


Figure 2.44. Relationship of fuzzy-logic systems to expert systems and neural networks (Kosko, 1992).

a. Representation of Fuzzy-Logic Variables

In this section, we adopt and update part of the discussion on fuzzy-logic systems in Quantrille and Liu (1991, pp. 208-210). First, we note that fuzzy logic is not a substitute for statistics. Instead, we use fuzzy logic only when statistical reasoning is inappropriate. Statistics expresses the extent of knowledge (or the lack thereof) about a value, and it relies on tools such as variance, standard deviation, and confidence intervals. Fuzzy logic, on the other hand, expresses the absence of a *sharp boundary* between sets of

information. For example, using fuzz logic we may write:

- Crude oil fractionation is clearly an energy-intensive unit operation, 1.0.
- Thermal cracking is a very energy-intensive unit operation, 0.9
- Catalytic reforming is a somewhat energy-intensive unit operation, 0.6.
- Catalytic cracking is an energy-intensive unit operation, 0.3.
- Open-air evaporation of brine to produce salt is not an energy-intensive unit operation, 0.0.

Here, the fuzzy logic delineates the lack of a sharp boundary between *clearly* energy-intensive (1.0) and *not at all* energy-intensive (0.0). Crude fractionation is very energy-intensive, while open-air evaporation of brine is not at all energy-intensive. Thermal cracking, catalytic reforming, and catalytic cracking cannot be considered either very energy-intensive or non-energy-intensive. Thus, fuzzy logic does not quantify the lack of knowledge in a statistical sense. Instead, it quantifies the *degree or extent* of certain words and boundaries between sets of information.

To use fuzzy logic, we first need a *fuzzy set*. In a fuzzy set, the transition from membership to non-membership is not well-defined. We quantify the *degree of membership* with values between 0 (not a member) and 1 (definitely a member). Figure 2.45 shows a representation of energy requirement in fuzzy terms (very low, low, moderate, high, very high). The transition from one discrete segment to another (e.g., low

to moderate) is not defined exactly. These regions overlap based on what one expert says is energy-intensive compared to what another says.

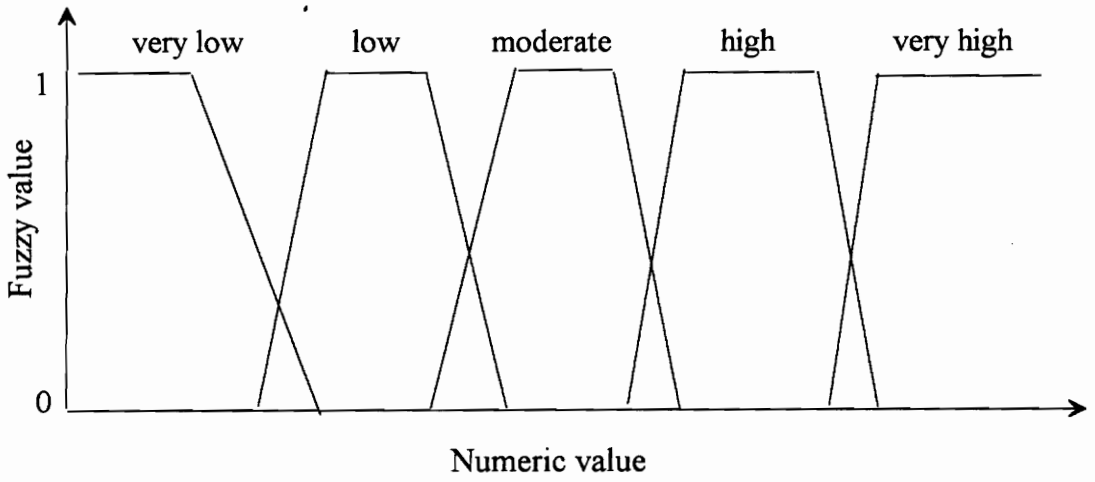


Figure 2.45. Representation of energy requirement in fuzzy terms.

b. Conversion Between Numeric and Fuzzy-Logic Variables

This section describes how to convert a numeric variable to a fuzzy-logic variable through *fuzzifier*, and to convert the fuzzy-logic variable back to a numeric variable through a *defuzzifier*.

We will use the low and moderate regions of the energy-requirement example (Figure 2.45) to demonstrate these two transformations. Figure 2.46 shows these regions

of the energy requirement with numerical values given for the transition regions. Although we recommend the use of symmetric transition regions where the sum of the member contributions equals 1 (e.g., low = 0.5 and moderate = 0.5) as in Figure 2.46, the transition regions can be staggered so that the fuzzy members do not total 1 (e.g., low = 0.5 and moderate = 0.3).

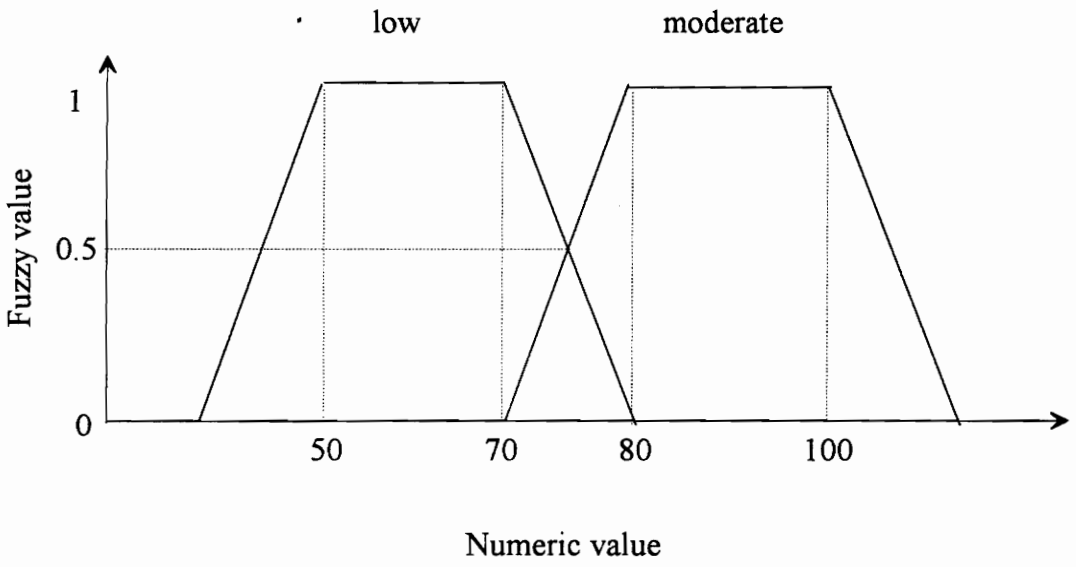


Figure 2.46. The low and moderate regions of the energy requirement represented in both numeric and fuzzy-logic variables.

In the following conversion examples, the numeric variable are denoted *numeric(x)* and the fuzzy-logic variables are denoted *fuzzy(very low, low, moderate, high, very high)*.

For numeric values of energy requirement that are definitely within a group (e.g., 50 to 70 for low, and 80 to 100 for moderate), we simply assign a value of 1 to the respective member of the group and 0 to the other members.

$$\text{numeric } (60) = \text{fuzzy } (0,1,0,0,0) \tag{2.60}$$

$$\text{numeric } (90) = \text{fuzzy } (0,0,1,0,0) \tag{2.61}$$

For numeric values in the transition regions, we use a linear interpolation between the beginning and ending values of the region.

$$\text{numeric } (74) = \text{fuzzy } \left(0,\frac{70-74}{70-80},\frac{80-74}{80-70},0,0\right) = \text{fuzzy}(0,0.4,0.6,0,0) \tag{2.62}$$

The fuzzy-logic values in equation 2.62 then represent the probability that the energy requirement is low (0.4) or moderate (0.6).

Similarly, we can convert the fuzzy-logic variable back to a numeric variable using the exact opposite process. As an example, we use a fuzzy-logic value of 0.7 for low and 0.3 for moderate.

$$\text{fuzzy}(0,0.7,0.3,0,0) = \quad 0.7 = \frac{70-x}{70-80} \quad \text{or} \quad \frac{80-x}{80-70} \qquad (x = 77) \tag{2.63}$$

c. Union and Intersection of Fuzzy Sets

Fuzzy logic plays a critical role in developing expert networks (see Chapter 6) because of its ability to use *fuzzy reasoning*. To understand fuzzy reasoning, we need two important concepts of classical set theory frequently used in expert systems: *union* and *intersection*. These concepts allow us to combine related fuzzy sets of information.

With our energy-intensive unit-operation example, the fuzzy set is:

{crude oil fractionation (1.0), thermal cracking (0.9), catalytic reforming (0.6), catalytic cracking (0.3)}

The open-air evaporation of brine to produce salt has a degree of membership of 0.0, and therefore, is not a member of the set.

We apply the union and intersection operations to fuzzy sets too. Let us define two fuzzy sets:

$$I = \{x_1/i_1 , x_2/i_2 , \ldots , x_n/i_n \}$$

$$J = \{x_1/j_1 , x_2/j_2 , \ldots , x_p/j_p \}$$

where x_1, x_2 ... are members of the set with nonzero degrees of membership i_1, i_2, ... (for set I) and j_1, j_2, ... (for set J). Note that the sets *do not* need to have the same number of

members; set I has n members, and set J has p members.

The union of two fuzzy sets is the fuzzy set containing the members of each set with the maximum degree of membership of that element in either set:

$$I \cup J = \{x_1/(\max(i_1, j_1)), x_2/(\max(i_2, j_2)), \ldots\}$$

The intersection of two fuzzy sets is the fuzzy set containing the members of each set with the minimum degree of membership of that element in both sets:

$$I \cap J = \{x_1/(\min(i_1, j_1)), x_2/(\min(i_2, j_2)), \ldots\}$$

For example, we consider the two sets:

I = {crude oil fractionation/1.0, thermal cracking/0.9, catalytic reforming/0.6, catalytic cracking/0.3}

J = {crude oil fractionation/0.8, thermal cracking/0.75, catalytic reforming/0.7, catalytic cracking/0.2, polymerization/0.1}

We perform both union and intersection:

$I \cup J$ = {crude oil fractionation/1.0, thermal cracking/0.9, catalytic reforming/0.7, catalytic cracking/0.3, polymerization/0.1}

$I \cap J$ = {crude oil fractionation/0.8, thermal cracking/0.75, catalytic reforming/0.6, catalytic cracking/0.2}

Davis and Gandikota (1990) discuss fuzzy sets, and we use their simple example here to demonstrate reasoning with fuzzy sets. If we have qualitative values for flow rate (F) and pressure (P) of a chemical process, we may write a rule which say that the system is abnormal:

The system is abnormal if:

(1) Both F and P are high, OR

(2) F is low, OR P is low.

Let us make F a fuzzy set of flow rates, and P a fuzzy set of pressures, with the following degrees of membership:

F = { low_F/0.5, high_F/0.3, normal_F/0.2}

P = { low_P/0.8, high_P/0.15, normal_P/0.05}

Now let us determine the following:

(1) *Certainty of high_F and high_P*: determined by the intersection of fuzzy sets F and P. Thus, certainty is the minimum degree of membership of high_F and high_P: certainty = min(0.3,0.15) = 0.15.

(2) *Certainty of low_F or low_P*: determined by the union of fuzzy sets F and P. Thus, certainty is the maximum degree of membership of low_F and low_P: certainty = max(0.5,0.8) = 0.8.

(3) *Overall uncertainty*: determined by taking the maximum certainties of both results, i.e., certainty = max(0.15,0.8) = 0.8.

Note again that the certainty in these rules is *not* to be interpreted as some type of "confidence limit" in the conclusion drawn. Instead, the certainty represents confidence in the qualitative values of the flow rate and pressure in the fuzzy sets.

2. Attachment of Neural Networks to Fuzzy-Logic Systems

In the following, we discuss two major methods frequently used in bioprocessing and chemical engineering for expert systems (Chapter 6) and process control of time-dependent systems (Chapter 5) for attaching neural networks to fuzzy-logic systems.

a. Neural-Fuzzy Networks for Expert Systems

Expert systems require input variables in the form of categorical or fuzzy-logic data. Therefore, the numeric output responses from a neural network must be converted into fuzzy-logic data for the expert system. Figure 2.47 shows a typical structure of a neural-fuzzy network used in conjunction with an expert system. The numeric data from a neural network is converted to fuzzy values through a *fuzzifier* for use as input into an expert system. The expert system produces output in the form of fuzzy values that is reconverted to numeric values through a *defuzzifier*. Note that the output from the neural network may already be in the form of fuzzy-logic or categorical variables based on the type of network used (e.g., the classification network of Chapter 3) and a fuzzifier may not be required. The defuzzifier may also be unnecessary depending on what form of output from the expert system we desire.

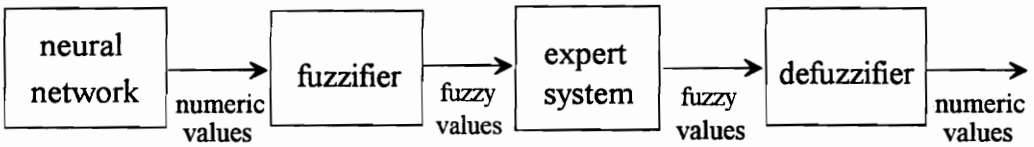


Figure 2.47. The structure of a neural-fuzzy networks used in conjunction with an expert system.

Chapter 6 describes how a neural network and an expert system are used together as *an expert network*. That chapter also presents an illustrative case study in which we develop an expert network for the extractive bioseparation of proteins in aqueous two-phase systems (Section 6.2).

b. Neural-Fuzzy Networks for Process Control

There are three main types of neural-fuzzy networks used for process control. The first uses neural networks to predict future responses of a control variable (e.g., cell concentration at the end of a fermentation process), then uses the difference from the targeted future response to determine the optimal controller settings. The second uses the pattern of a given signal (e.g., increasing, decreasing, oscillating, etc.) to determine the optimal controller settings. The final has uses a special neural network structure which functions as a fuzzy-inference system. Section 5.6.E describe neural-fuzzy controllers in greater detail.

Figure 2.48 shows a simple architecture for neural-fuzzy controllers. The neural network predicts either the future response of a control variable or its pattern type, which then becomes the input into the fuzzy-logic controller. A fuzzifier converts the neural network output and other control variables that do not require a predictor. The fuzzy controller uses a set of fuzzy rules, similar to an expert system, to obtain the optimal predetermined controller settings.

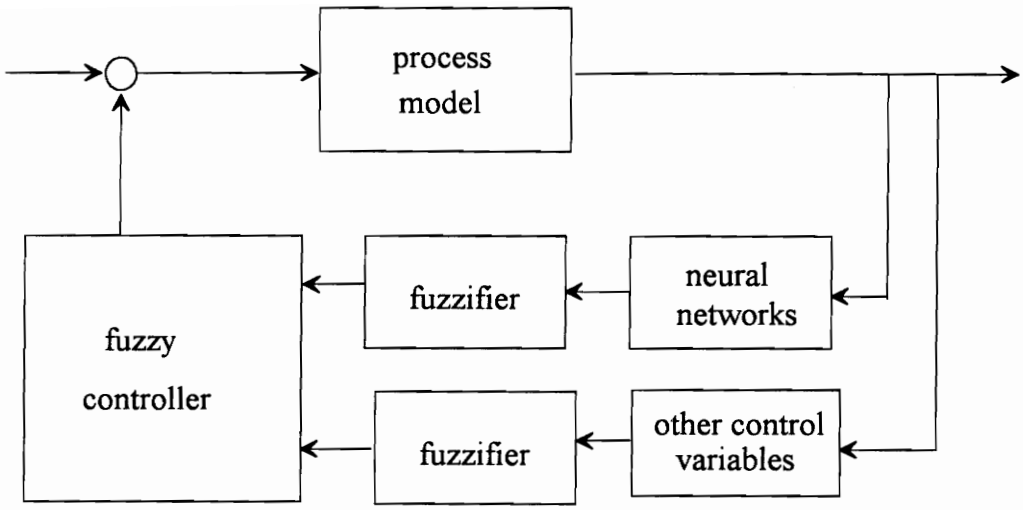


Figure 2.48. A simple architecture for neural-fuzzy controllers.

E. Other Networks

This section describes several other neural networks applicable to bioprocessing and chemical engineering not covered in detail in this introductory text. Following are several of these network types and associated references, together with some reported applications in bioprocessing and chemical engineering

- *Adaptive Resonance Theory (ART)* is primarily used for feature detection and pattern classification, and is based on the theoretical foundation of Grossberg (1976a,b). There are several implementations of the ART, resulting in neural networks called ART1,

are several implementations of the ART, resulting in neural networks called ART1, ART2, ART2-A, and ART3, etc. For general references on ART2 and ART2-A networks, we suggest Carpenter and Grossberg (1987a) and Carpenter et al. (1991). In chemical engineering, Whitley and Davis (1992, 1993, 1994) describe how ART2 networks are used to generate qualitative interpretations of 2-dimensional sensor patterns that can be used in control, monitoring, and diagnosis of chemical processes. In addition, Meagan and Cooper (1994) apply ART2-A networks to pattern-recognition-based adaptive control.

- *Learning-Vector-Quantization (LVQ) Networks* are autoassociative classification networks that learn through supervised training. This type of network is advantageous when classifying systematic patterns such as the two spiral problem presented in the dissertation by Suewatanakul (1993). In comparison, it does not perform as well as other classification networks when the data are scattered. The theoretical development of the LVQ networks is presented by Kohonen (1982, 1988). Two examples of the LVQ networks applied to chemical engineering, specifically in pattern-based adaptive process control, are Cooper et al. (1992) and Hinde and Cooper (1994).
- *Wavelet Networks* resemble the radial-basis-function networks discussed in Chapter 3, except that they use a more elaborate transfer function. The wavelet network is discussed by Bakashi (1992) and Bakashi and Stephanopoulos (1993).
- *Functional-Link Networks* simply add a functional-link layer to a backpropagation network. This additional layer improves the network's capability to represent complex

nonlinear relations. Chapter 8 of Pao (1989) gives a detailed description of the mathematical development and learning algorithms for the functional-link network. Fan et al. (1994) apply a functional-link network for the fault diagnosis of a heptane-to-toluene process.

- *Grossberg/Hopfield Neural Networks* are used in modeling time-dependent systems. The structure of this recurrent network is similar to that discussed in Section 5.3, except that the input-output dynamics of each node are governed by a simple first-order ordinary differential equation. See Pearlmutter (1989) and NeuralWare (1993) for some background discussion. Nikolaou and Hanagandi (1992, 1993) and You and Nikolaou (1993) apply this network to the control of nonlinear dynamic systems. In addition, Karim and Rivera (1992a,b) use this network for process forecasting of the batch cell-growth fermentation process. Werbos (1988) and Su et al. (1992) give a good comparative discussion of both the time-lag recurrent network (Section 2.6.C) and the Grossberg/Hopfield recurrent network.

2.7 Chapter Summary

- The foundation of a neural network is the *node* or *processing element* where most of the calculations in the neural network are performed.
- Every input entering a node is multiplied by its corresponding *weight factor*, w_{ij}. These weight factors which are adjusted in the network training algorithm, store the pattern

behavior of the system,

- The input of a node goes through a *transfer function*, the most common being the sigmoid, hyperbolic tangent, and radial basis (Gaussian) functions.
- There are two main categories of learning: (1) *supervised learning* has an external teacher controlling the learning and incorporating global information; and (2) *unsupervised learning* has no external teacher and the neural network relies upon both internal control and local information. In unsupervised learning, the neural network frequently develops its own models without additional input information.
- The *generalized delta rule* (delta rule), the most common method for training backpropagation networks, is an iterative gradient-descent method that minimizes the least-squares (LMS) output error. This technique uses a *momentum* term to accelerate the training rate.
- We recommend the following network:
    - Zero-mean normalization method
    - Gaussian weight-factor distribution
    - Radial basis function for classification networks (Chapter 3)
    - Hyperbolic tangent transfer function for prediction problems (Chapter 4) and process-forecasting problems (Chapter 5)
    - A network with 30 nodes in hidden layer 1 and 15 in hidden layer 2 as a good initial architecture for most prediction and process-forecasting networks.
- A *learning curve* provides a good method to visualize a network performance for recall

and generalization. The learning curve (Figure 2.11) plots the average error for both recall of training data sets and generalization of testing data sets as a function of the number of examples in the training data set.

- An autoassociative network correlates an input pattern to itself, and is used for data compression and filtering, and for dimensionality reduction of an input vector.
- A hierarchical neural network has several hidden layers segmented into subnetworks, where the input vectors are divided into groups based on their effects on the output responses. Two significant types of hierarchical networks are moving-window networks for time-dependent processes and input-compression networks for working with large input-variable sets.
- Recurrent networks for time-dependent systems combine the feedback and feedforward connections of neural networks, providing a means to use the output responses of the network as additional input variables through recurrent loops.
- Neural-fuzzy networks, frequently used for both process control and expert systems, implement fuzzy-logic inferencing through neural networks.

Nomenclature

a	:	input vector entering a node.
a_i	:	the i^{th} component of an input vector entering a node.
B(n)	:	bottleneck vector (hidden layer) of a autoassociative network having *n* nodes.
b	:	output vector.
b_j	:	calculated output for the j^{th} node.
b_k	:	calculated output for the k^{th} node.
d_k	:	desired output for the k^{th} node.
e_j	:	output error of j^{th} node of layer B relative to each ε_k in the three-layer perceptron network.
E	:	total squared error on the output layer.
$f(\ )$	:	transfer function.
I	:	input vector entering a neural network.
I_i	:	the i^{th} component of the input vector entering a neural network.
In(m)	:	input vector (layer 1) of an autoassociative network having *m* nodes.
L	:	number of nodes in layer A of the three-layer perceptron network.
m	:	number of nodes in layer B of the three-layer perceptron network.
m	:	number of incoming data points (size of input vector) entering a autoassociative network.
m/*n*	:	compression ratio of an autoassociative network having *m* nodes in the

input and output layers and *n* nodes in the bottleneck layer.

$N(x)$: normalization function of an input or output variable.

n : number of nodes in layer C of the three-layer perceptron network.

n : number of nodes in the bottleneck (hidden) layer of a autoassociative network.

Out(m): output vector (layer) of an autoassociative network having *m* nodes.

$R_{i,max}$: maximum range between the average value $x_{i,avg}$ and either the minimum value $x_{i,min}$ or maximum value $x_{i,max}$.

T_j : internal threshold for j^{th} node.

T_{Ai} : internal threshold for i^{th} node (layer A) of the three-layer perceptron network.

T_{Bj} : internal threshold for j^{th} node (layer B) of the three-layer perceptron network.

T_{Ck} : internal threshold for k^{th} node (layer C) of the three-layer perceptron network.

t : time.

$\mathbf{w}_j$: weight-factor vector for the j^{th} nodes.

w_{ij} : weight factor between i^{th} and j^{th} nodes.

$w_{ij,new}$: newly adjusted weight factor between the i^{th} and j^{th} nodes during network training.

w_{jk} : weight factor between j^{th} (layer B) and k^{th} (layer C) nodes in the three-layer

perceptron network.

$w_{jk,new}$: newly adjusted weight factor between j^{th} (layer B) and k^{th} (layer C) nodes in the three-layer perceptron network.

VARi(t): the i^{th} input variable at time t in a process-trend scanning window.

v_{ij} : weight factor between i^{th} (layer A) and j^{th} (layer B) nodes in the three-layer perceptron network.

$v_{ij,new}$: newly adjusted weight factor between i^{th} (layer A) and j^{th} (layer B) nodes in the three-layer perceptron network.

x_i : an input or output variable in a training or testing database.

$x_{i,avg}$: average value of x_i .

$x_{i,min}$: minimum value of x_i.

$x_{i,max}$: maximum value of x_i.

$x_{i,norm}$: normalized value of x_i.

$x_i(t)$: the j^{th} input variable at time t of a recurrent network.

x_i : total activation of a node.

$y_j(t)$: the j^{th} output response at time t of a recurrent network.

α : momentum coefficient.

δ_{ij} : gradient-descent term.

Δw_{ij} : change in weight factor for the i^{th} and j^{th} node connection.

ε_k : output error of the k^{th} node.

η_j : learning rate ($0 < \eta_j < 1$).

Practice Problems

(2.1) Follow the procedures for vanilla backpropagation (Section 2.2.C) and generalized delta-rule (delta-rule) algorithms (Section 2.2.D) for the fault-diagnosis example (Section 2.2.B), generating values for all intermediate variables. Use the following input/output pattern corresponding to the catalyst-sintering fault:

input:	$I_1 = 0.55$	*output:* $c_1 = 0$ (no problem)
	$I_2 = 1.00$	$c_2 = 0$ (no problem)
	$I_3 = 0.20$	$c_2 = 1$ (catalyst sintering)

Set the initial weight-factor (v_{ij} and w_{jk}) matrices and internal threshold (T_{ij}) matrix to:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

$$[T_{ij}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & -0.5 \\ 0.0 & 0.5 & -0.5 \end{bmatrix}$$

Compare the magnitude of the changes in weight factors, v_{ij} and w_{jk}, between these two training algorithms.

(2.2) Using the input/output pattern of problem 2.1 for the fault-diagnosis example (Section 2.2.B)

(a) Train a neural network using the vanilla backpropagation algorithm (Section 2.2.C) to less than 1% error on d_1 (Use the program in Appendix 2.A).

(b) Train a neural network using the delta-rule algorithm (Section 2.2.D) to less than 1% error on d_1 (Use the program in Appendix 2.B).

(c) Compare the number of iterations required by both algorithms to obtain 1% error in d_1.

(2.3) Using the delta-rule algorithm of Section 2.2.D (program in Appendix 2.B), train a neural network with multiple input vectors. Use the three sets of input-output data, listed below:

Input:	$I_1 = 0.30$	*Output:*	$c_1 = 1$ (low conversion)
	$I_2 = 1.00$		$c_2 = 0$ (no problem)
	$I_3 = 0.20$		$c_3 = 0$ (no problem)
Input:	$I_1 = 0.325$	*Output:*	$c_1 = 1$ (low conversion)
	$I_2 = 0.90$		$c_2 = 0$ (no problem)
	$I_3 = 0.20$		$c_3 = 0$ (no problem)

Input:	$I_1 = 0.35$	*Output:*	$c_1 = 1$ (low conversion)
	$I_2 = 0.80$		$c_2 = 0$ (no problem)
	$I_3 = 0.20$		$c_3 = 0$ (no problem)

(2.4) Write a program that will perform *recall* for the neural network trained in problem 2.2. The recall program must:

(a) Prompt the user for the input vector.

(b) Use the weight factors derived from training to calculate outputs from the input, hidden, and output layers.

(c) Report the result from the output layer to the user.

(2.5) Create a fully operational neural network, performing qualitative interpretation of process data from a chemical reactor using the same three-layer, three-input, three-output network discussed in the chapter. Note that this classification problem is developed in detail in Section 3.4.B.2. Write a program, in the computer language of your choice, that achieves the objectives described below.

(a) *Develop the training section* using the delta-rule algorithm (Section 2.2.D), capable of training with multiple input vectors (problem 2.3 above).

(b) *Develop the recall section*, achieving the three objectives of problem 2.4 above, i.e., prompt the user for the input; calculate outputs from the input, hidden, and output layers; and report the result to the user.

(c) *Train the network* using the following data (Table 2.P1) to train the network, data are given in file: *fault.nna*.

Table 2.P1. Input and output data for chemical reactor fault-diagnosis network.

x_1	x_2	x_3	x_1	x_2	x_3	y_1	y_2	y_3
400 ° F	100 psia	200 lb/min	0.40	1.00	0.20	0	0	0
420 ° F	100 psia	200 lb/min	0.42	1.00	0.20	0	0	0
380 ° F	100 psia	200 lb/min	0.38	1.00	0.20	0	0	0
400 ° F	100 psia	200 lb/min	0.40	1.00	0.20	0	0	0
400 ° F	90 psia	200 lb/min	0.40	0.90	0.20	0	0	0
400 ° F	100 psia	220 lb/min	0.40	1.00	0.22	0	0	0
400 ° F	100 psia	180 lb/min	0.40	1.00	0.18	0	0	0
300 ° F	100 psia	200 lb/min	0.30	1.00	0.20	1	0	0
325 ° F	90 psia	200 lb/min	0.325	0.90	0.20	1	0	0
350 ° F	80 psia	200 lb/min	0.35	0.80	0.20	1	0	0
370 ° F	100 psia	200 lb/min	0.37	1.00	0.20	1	0	0
380 ° F	100 psia	180 lb/min	0.38	1.00	0.18	1	0	0
400 ° F	100 psia	250 lb/min	0.40	1.00	0.25	0	1	0
400 ° F	100 psia	230 lb/min	0.40	1.00	0.23	0	1	0
550 ° F	100 psia	200 lb/min	0.55	1.00	0.20	0	0	1
525 ° F	100 psia	180 lb/min	0.525	1.00	0.18	0	0	1

(d) *Perform recall* based on the input data below, determining the network output c_k. What potential problems could arise from these operating conditions?

Input: $I_1 = 418$ °F/1000 = 0.418 *Output:* $c_1 = ?$

$I_2 = 102$ psia/100 = 1.02 $c_2 = ?$

$I_3 = 190$ lb/min/1000 = 0.19 $c_3 = ?$

Input: $I_1 = 381$ °F/1000 = 0.381 *Output:* $c_1 = ?$

$I_2 = 92$ psia/100 = 0.92 $c_2 = ?$

$I_3 = 185$ lb/min/1000 = 0.185 $c_3 = ?$

(e) Vary the learning rate from 0.01 to 5 (0.01, 0.1, 0.5, 1.0, and 5.0) and the momentum coefficient from 0 to 0.8 (0, 0.2, 0.4, 0.6, and 0.8), comparing the training time required to achieve less than 1% error on d_1.

Appendix 2.A

Code for the backpropagation algorithm.

Written in BASIC.

```
1 REM *** VANILLA BACKPROPAGATION ALGORITHM ***
2 REM ***     WRITTEN IN BASIC          ***
3 REM ****************************************
4 REM  * VARIABLES AND THEIR MEANINGS       *
5 REM  * A(I) = OUTPUT FROM NODES IN LAYER 1  *
6 REM  * B(I) = OUTPUT FROM NODES IN LAYER 2  *
7 REM  * BETA = LEARNING RATE                *
8 REM  * C(I) = OUTPUT FROM NODES IN LAYER 3  *
9 REM  * D(I) = DESIRED OUTPUT FROM LAYER 3   *
10 REM * E(I) = ERROR CALCULATED FROM LAYER 3 *
11 REM * E(I) = ERROR CALCULATED FROM LAYER 3 *
12 REM * EB(I) = ERROR CALCULATED FROM LAYER 3*
13 REM * PNPUT(I) = INPUT VECTOR INTO LAYER 1 *
14 REM * T(I,J) = INTERNAL THRESHOLD VALUES,  *
15 REM *    I = LAYER NUMBER                  *
16 REM *    J = NODE NUMBER WITHIN LAYER      *
17 REM * V(I,J) = LAYER 1-2 CONNECTION WEIGHTS*
18 REM * W(J,K) = LAYER 2-3 CONNECTION WEIGHTS*
19 REM * X(I) = POST-THRESHOLD INPUT, LAYER 1 *
20 REM ****************************************
21 DIM A(3),B(3),C(3),D(3),E(3),EB(3),PNPUT(3)
22 DIM T(3,3),V(3,3),W(3,3),X(3)
23 REM *** STEP 1: INITIALIZE THE VARIABLES ***
24 V(1,1) = -1 : V(1,2) = -.5 : V(1,3) = .5
25 V(2,1) =  1 : V(2,2) = 0.0 : V(2,3) = -.5
26 V(3,1) = .5 : V(3,2) = -.5 : V(3,3) = .5
27 W(1,1) = -1 : W(1,2) = -.5 : W(1,3) = .5
28 W(2,1) =  1 : W(2,2) =  0  : W(2,3) = .5
29 W(3,1) = .5 : W(3,2) = -.5 : W(3,3) = .5
30 T(1,1) =  0 : T(1,2) =  0  : T(1,3) =  0
31 T(2,1) = .5 : T(2,2) =  0  : T(2,3) = -.5
32 T(3,1) =  0 : T(3,2) = .5  : T(3,3) = -.5
33 PNPUT(1) = 300
34 PNPUT(2) = 100
35 PNPUT(3) = 200
36 D(1) = 1 : D(2) = 0 : D(3) = 0
37 BETA = .7
38 REM *** STEP 2: INTRODUCE INPUT VECTOR ***
39 REM *** ALSO CALCULATE LAYER ONE OUTPUTS***
40          FOR I = 1 TO 3
41          X(I) = PNPUT(I) - T(1,I)
42          A(I) = 1/(1+EXP(-X(I)))
43          NEXT I
44 REM *** STEP 3: FIND LAYER TWO OUTPUTS ***
45          FOR J = 1 TO 3
46          SUM = 0
47                    FOR I = 1 TO 3
48                    SUM = SUM + V(I,J)*A(I)
49                    NEXT I
50          B(J) = 1/(1+EXP(-(SUM-T(2,J))))
51          NEXT J
52 REM *** STEP 4: FIND LAYER THREE OUTPUTS ***
53          FOR K = 1 TO 3
54          SUM = 0
55                    FOR J = 1 TO 3
56                    SUM = SUM + W(J,K)*B(J)
57                    NEXT J
58          C(K) = 1/(1+EXP(-(SUM-T(3,K))))
59          NEXT K
60 REM *** STEP 5: FIND OUTPUT DIFFERENCE ***
61 DIF1 = D(1)-C(1)
62 DIF2 = D(2)-C(2)
63 DIF3 = D(3)-C(3)
64 PRINT "DESIRED VALUES: D1  D2  D3"
65 PRINT D(1);D(2); D(3)
66 PRINT "ACTUAL VALUES: C1  C2  C3"
67 REM *** STOP EXECUTION IF ERROR IS LOW ***
68 IF ABS(DIF3) .0001 THEN END
69 REM *** FIND OUTPUT ERROR ***
70          FOR K = 1 TO 3
71          E(K) = C(K)*(1-C(K))*(D(K)-C(K))
72          NEXT K
73 PRINT "BEGINNING BACKPROPAGATION ..."
74 REM *** FIND LAYER TWO ERROR ***
75          FOR J = 1 TO 3
76          SUM = 0
77                    FOR K = 1 TO 3
78                    SUM = SUM + W(J,K)*E(K)
79                    NEXT K
80          EB(J) = B(J)*(1-B(J))*SUM
81          NEXT J
82 REM *** STEP 7: ADJUST W(J,K) WEIGHTS ***
83 REM *** STEP 8: ADJUST T(3,K) THRESHOLDS ***
84          FOR K = 1 TO 3
85                    FOR J = 1 TO 3
86                    W(J,K) = W(J,K) + BETA*B(J)*E(K)
87                    NEXT J
88          T(3,K) = T(3,K) + BETA*E(K)
89          NEXT K
90 REM *** STEP 9: ADJUST V(I,J) WEIGHTS ***
91 REM *** STEP 10: ADJUST T(2,J) THRESHOLDS ***
92          FOR J = 1 TO 3
93                    FOR I = 1 TO 3
94                    V(I,J) = V(I,J) + BETA*A(I)*EB(J)
95                    NEXT I
96                    T(2,J) = T(2,J) + BETA*EB(J)
97                    NEXT J
98 PRINT "IF YOU WISH TO EXECUTE ANOTHER
99 PRINT "TIME-STEP, ENTER: CONT"
100 STOP
101 GOTO 40
102 REM ****************************************
103 REM *                                      *
104 REM * THE USER IS ENCOURAGED TO PLACE BOTH *
105 REM * PRINT AND STOP STATEMENTS ANYWHERE   *
106 REM * IN THE PROGRAM TO VIEW THE PROGRESS  *
107 REM * OF THE PROGRAM. WHEN THE PROGRAM     *
108 REM * CEASES EXECUTION AT A STOP STATEMENT,*
109 REM * SIMPLY ENTER CONT (CONTINUE) AT THE  *
110 REM * INTERPRETER TO RESUME EXECUTION      *
111 REM *                                      *
112 REM ****************************************
```

Appendix 2.B

Code for the Generalized Delta-Rule (Delta-Rule) Algorithm. Written in BASIC.

```
1 REM *** GENERALIZED DELTA RULE ALGORITHM ***
2 REM ***     WRITTEN IN BASIC          ***
3 REM ****************************************
4 REM  * VARIABLES AND THEIR MEANINGS        *
5 REM  * ALPHA = COEFFICIENT OF MOMENTUM     *
6 REM  * A(I) = OUTPUT FROM NODES IN LAYER 1  *
7 REM  * B(I) = OUTPUT FROM NODES IN LAYER 2  *
8 REM  * BETA = LEARNING RATE                *
9 REM  * C(I) = OUTPUT FROM NODES IN LAYER 3  *
10 REM  * D(I) = DESIRED OUTPUT FROM LAYER 3  *
11 REM * DELTA(I,J) = GRADIENT DESCENT TERM   *
12 REM *     FOR THE I-TH NODE IN J-TH LAYER  *
13 REM * DELTA(I,J) = GRADIENT DESCENT TERM   *
14 REM *     FOR THE I-TH NODE IN J-TH LAYER  *
15 REM * DELTAV(I,J) = CHANGE IN WEIGHT V(I,J) *
16 REM * DELTAW(J,K) = CHANGE IN WEIGHT W(J,K) *
17 REM * DERV(I,J) = DERIVATIVE OF SIGMOID    *
18 REM *     FUNCTION FOR THE TOTAL INPUT TO  *
19 REM *     THE I-TH NODE IN J-TH LAYER      *
20 REM * PNPUT(I) = INPUT VECTOR INTO LAYER 1  *
21 REM * SUMB(J) = POST-BIAS INPUT FOR NODE J  *
22 REM *         IN THE HIDDEN LAYER          *
23 REM * SUMC(K) = POST-BIAS INPUT FOR NODE K  *
24 REM *         IN THE OUTPUT LAYER          *
25 REM * SUMDELTA(K) = WEIGHTED SUM OF GRADIENT*
26 REM *       DESCENT TERM FROM OUTPUT LAYER *
27 REM * T(I,J) = BIAS FUNCTION FOR J-TH NODE  *
28 REM *         IN THE I-TH LAYER            *
29 REM * V(I,J) = LAYER 1-2 CONNECTION WEIGHTS *
30 REM * W(J,K) = LAYER 2-3 CONNECTION WEIGHTS *
31 REM * X(I) = POST-BIAS INPUT, I-TH NODE     *
32 REM ****************************************
33 DIM A(3),B(3),C(3),D(3),DELTA(3,3)
34 DIM DERV(3,3),PNPUT(3),SUMB(3)
35 DIM SUMC(3),T(3,3),V(3,3)
36 DIM W(3,3),DELTAV(3,3),DELTAW(3,3)
37 DIM SUMDELTA3(3)
38 REM *** STEP 1: ASSIGN WEIGHTS          ***
39 V(1,1) = -1 : V(1,2) = -.5 : V(1,3) = .5
40 V(2,1) = 1  : V(2,2) = 0   : V(2,3) = -.5
41 V(3,1) = .5 : V(3,2) = -.5 : V(3,3) = .5
42 W(1,1) = -1 : W(1,2) = -.5 : W(1,3) = .5
43 W(2,1) = 1  : W(2,2) = 0   : W(2,3) = .5
44 W(3,1) = .5 : W(3,2) = -.5 : W(3,3) = .5
45 T(1,1) = 0  : T(1,2) = 0   : T(1,3) = 0
46 T(2,1) = 1  : T(2,2) = 1   : T(2,3) = 1
47 T(3,1) = 1  : T(3,2) = 1   : T(3,3) = 1
48 PNPUT(1) = 300 : PNPUT(2)=100: PNPUT(3)=200
49 D(1) = 1 : D(2) = 0 : D(3) = 0
50 ETA = .9 : ALPHA = .6
51 REM ***  STEP 2: INTRODUCE INPUT VECTOR.  ***
52 REM *** CALCULATE OUTPUT FROM FIRST LAYER ***
53  FOR I = 1 TO 3
54  X(I) = PNPUT(I) - T(1,I)
55  A(I) = 1/(1+EXP(-X(I)))
56  NEXT I
57 REM *** STEP 3: FIND HIDDEN LAYER OUTPUTS  ***
58  FOR J = 1 TO 3
59  SUMB(J) = 0
60    FOR I = 1 TO 3
61    SUMB(J)=SUMB(J)+V(I,J)*A(I)
62    NEXT I
63  SUMB(J) = SUMB(J)+T(2,J)
64  B(J) = 1/(1+EXP(-(SUMB(J))))
65  NEXT J
66 REM *** STEP 4: FIND LAYER THREE OUTPUTS  ***
66  FOR K = 1 TO 3
67  SUMC(K) = 0
68    FOR J = 1 TO 3
70    SUMC(K)=SUMC(K)+W(J,K)*B(J)
71    NEXT J
72  SUMC(K) = SUMC(K)+T(3,K)
73  C(K) = 1/(1+EXP(-(SUMC(K))))
74  NEXT K
75 DIF1=D(1)-C(1): DIF2=D(2)-C(2): DIF3=D(3)-C(3)
76 IF ABS(DIF1) .01 THEN END
77 REM ***   STEP 5: FIND SQUARED ERROR    ***
78 SQERROR = 0
79  FOR K = 1 TO 3
80  SQERROR= SQERROR+((D(K)-C(K))^2
81  NEXT K
82 REM *** STEP 6: FIND THE GRADIENT DESCENT ***
83 REM *** TERM FOR OUTPUT LAYER NODES       ***
84  FOR K = 1 TO 3
85  DERV(3,K) = EXP(-SUMC(K))/((1 +EXP(-SUMC(K)))^2
86  DELTA(3,K) = (D(K)-C(K))*DERV(3,K)
87  NEXT K
88 REM *** STEP 7: FIND THE GRADIENT DESCENT  ***
89 REM *** TERM FOR HIDDEN LAYER NODES       ***
90 FOR J = 1 TO 3
91 DERV(2,J) = EXP(-SUMB(J))/((1 + EXP(-SUMB(J)))*(1 + EXP(-SUMB(J))))
92 SUMDELTA3(J) = 0
93  FOR K = 1 TO 3
94  SUMDELTA3(J)=SUMDELTA3(J)+DELTA(3,K)*W(J,K)
95  NEXT K
96 DELTA(2,J) = DERV(2,J)*SUMDELTA3(J)
97 NEXT J
98 REM *** STEP 8: FIND DELTA V AND DELTA W   ***
99  FOR I = 1 TO 3
100   FOR J = 1 TO 3
101   DELTAV(I,J)=ETA*DELTA(2,J)*A(I)+ALPHA* DELTAV(I,J)
102   DELTAW(I,J)=ETA*DELTA(3,J)*B(I)+ALPHA* DELTAW(I,J)
103 REM *** STEP 9: UPDATE WEIGHTS          ***
104   V(I,J) = V(I,J) + DELTAV(I,J)
105   W(I,J) = W(I,J) + DELTAW(I,J)
106   NEXT J
107 NEXT I
108 GOTO 53
109 REM *** THE USER IS ENCOURAGED TO PUT PRINT
110 REM *** AND STOP STATEMENTS IN PROGRAM   ***
```

References and Further Reading

The number of publications on neural networks is growing rapidly, and it is getting difficult to keep up. First, some books that are helpful are: (1) *Neural Computing: Theory and Practice* (Wasserman, 1989) - a good introductory and intermediate-level book on neural networks; (2) *Artificial Neural Systems* (Simpson, 1990) - a good reference book, mathematically oriented, with numerous technical references; and (3) *Introduction to Artificial Neural Systems* (Zurada, 1992) - a good recent textbook. We also highly recommend the excellent text *Neurocomputing* by Hecht-Nielsen (1990) and the useful handbook *Neural Computing* (NeuralWare, 1993) to our readers. Hecht-Nielsen gives a good discussion of the historical perspective and recent developments of neural computing, including aspects of network architecture and theory, implementation and applications. We have found the NeuralWare handbook most useful in learning about neural networks and during our preparation of this text.

At the time of this writing, most artificial intelligence textbooks unfortunately do not discuss the growing importance of neural networks. An exception is Rich and Knight (1991). Chapter 18 of this book briefly discusses the concepts of neural networks in the context of "connectionist models." Chapter 17 of our text, *Artificial Intelligence in Chemical Engineering* (Quantrille and Liu, 1991) introduces neural networks. We have adopted and updated some of the discussion in that chapter in Sections 2.1 and 2.2.

There are many companies advertising PC-based neural network software. We

have used both *NeuralWorks Explorer* and *NeuralWorks Professional II/PLUS*, available from NeuralWare, Inc., Penn Center West, Building 4, Suite 227, Pittsburgh, PA 15276, phone (412) 787-8222 and fax (412) 787-8220.

There are several specialized journals on neural networks, including: (1) *IEEE Transactions on Neural Networks*; (2) *Neural Networks*, published by Pergamon Press, Inc., Fairview Park, Elmsford, NY 10523; and (3) *Neural Computation*, published by the MIT Press, 55 Hayward Street, Cambridge, MA 02142. *The IEEE Transactions on Neural Networks* tends to be more engineering- and application-oriented, and *Neural Networks* more biologically and mathematically inclined.

Articles describing applications of neural networks to bioprocessing and chemical engineering appear frequently in such journals as *Computers and Chemical Engineering*, *AIChE Journal*, *Industrial and Engineering Chemistry Research*, and *Bioengineering and Biotechnology*.

In addition, a number of magazines and journals have had special issues dedicated to neural networks. Some of these are: (1) *IEEE Control Systems Magazine*, April 1988, 1989, and 1990, (2) *PC-AI* May/June 1990, (3) *IEEE Communications Magazine*, November 1989. Finally, *Computers and Chemical Engineering* published a special issue on neural network applications in chemical engineering in April, 1992.

Anderson, J. A. and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA (1988).

Bakashi, B. R., "Multi-resolution Methods for Modeling, Analysis and Control of Chemical Process Operations," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA (1992).

Bakashi, B. R. and G. G. Stephanopoulos, "Wave-Net: A Multiresolution, Hierarchical Neural Networks with Localized Learning," *AIChE J.*, **39**, 57 (1993).

Baum, E. B. and D. Hauser, "What Size Net Gives Valid Generalization," *Neural Computation*, **1**, 151 (1989).

Bulsari, A. B., "Training Neural Networks for Fuzzy Logic," *Complex Systems*, **6**, 443 (1992).

Bulsari, A. B. and a. Krastawski, "Fuzzy Simulation by an Artificial Neural Network," *Fuzzy Systems and A.I.: Reports and Letters*, **1**, No. 3, 43 (1992).

Candill, M., *Neural Network Primer*, 64 pages, reprints from *AI Expert*, Freeman Publications, San Francisco, CA (1990).

Carpenter, G. A., "Neural Network Models for Pattern Recognition and Associative Memory," *Neural Networks*, **2**, 243 (1989).

Carpenter, G. A. and S. Grossberg, "ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Appl. Opt.*, **26**, 4919 (1987).

Carpenter, G. A., S. Grossberg and D. B. Rosen, "ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition," *Neural Networks*, **4**, 493 (1991).

Chitra, S. P., "Use of Neural Networks for Problem Solving," *Chem. Eng. Prog.*, **89**, No.4, 44, April (1993).

Cooper, D. J., L. Meagan and R. F. Hinde, Jr., "Comparing Two Neural Networks for Pattern-Based Adaptive Process Control," *AIChE J.*, **38**, 41 (1992).

Davis, J. F. and M. S. Gandikota, "Rule-Based Systems in Chemical Engineering," in *Artificial Intelligence in Process Systems Engineering, Volume II*, G. Stephanopoulos and J. F. Davis, editors, CACHE Monograph Series, CACHE Corporation, Austin, TX (1990).

Eberhart, R. C. and R. W. Dobbins, editors, *Neural Network PC Tools: A Practical Guide*, Academic Press, San Diego, CA (1990).

Fan, J. Y., M. Nikolaou and R. E. White, "An Approach to Fault Diagnosis of Chemical Processes Via Neural Networks," *AIChE J.*, **39**, 82 (1993).

Grossberg, S., "Adaptive Pattern Classification and Universal Recording: I. Parallel Development and Coding of Neural Feature Detectors," *Biol. Cybern.*, **23**, 121 (1976a).

Grossberg, S., "Adaptive Pattern Classification and Universal Recording: II. Feedback, Expectation, Olfaction, and Illusions," *Biol. Cybern.*, **23**, 187 (1976b).

Grossberg, S., "Nonlinear Neural Networks: Principles, Mechanisms and Architecture," *Neural Networks*, **1**, 17 (1988).

Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, Reading, MA (1990).

Hinde, R.F., Jr. and D. J. Cooper, "A Pattern-Based Approach to Excitation Diagnosis for Adaptive Process Control," *Chem. Eng. Sci.*, **49**, 1403 (1994).

Hobb, D., *Organization of Behavior*, Wiley, New York, NY (1949).

Hsiung, J. T., W. Suewatanakul and D. M. Himmelblau, "Should Backpropagation Be Replaced by More Effective Optimization Algorithms," *Intern. Joint Conf. on Neural Networks*, Seattle, WA (1991).

Kohonen, T., "An Introduction to Neural Computing," *Neural Networks*, **1**, 3 (1988).

Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, NY (1989).

Kohonen, T., "An Introduction to Neural Computing," *Neural Networks*, **1**, 3 (1988).

Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, N.J. (1991).

Leonard, J. A. and M. A. Kramer, "Limitations of the Backpropagation Approach to Fault Diagnosis and Improvement with the Radial Bias Function," *AIChE Annual Meeting*, Chicago, IL, November (1990).

Leonard, J. A. and M. A. Kramer, "Improvement of the Backpropagation Algorithm for Training Neural Networks," *Comput. Chem. Eng.*, **14**, 337 (1990).

Karim, M. N. and S. L. Rivera, "Comparison of Feedforward and Recurrent Neural Networks for Bioprocess State Estimation," *Comput. Chem. Eng.*, **16**, S369 (1992a).

Karim, M. N. and S. L. Rivera, "Use of Recurrent Neural Networks for Bioprocess Identification in On-Line Optimization by Micro-genetic Algorithm," *Proceedings Amer. Control Conf.*, p. 1931, Chicago, IL, (1992b).

Mah, R. S. H., "Neural Nets," Letter to the Editor, *Chem. Eng. Prog.*, **87**, 6, January (1991).

Mavrovouniotis, M. L. and S. Chang, "Hierarchical Neural Networks," *Comput. Chem. Eng.*, **16**, 347 (1992).

McClelland, J. L. and D. E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs and Exercises*, MIT Press, Cambridge, MA (1988).

McCulloch, W. S. and W. Pitts, "A Logical Calculus of the Ideas Immanent in Neurons Activity," *Bull. Math. Biophys.*, **5**, 115 (1943).

Meagan, L. and D.J. Cooper, "Pattern-Recognition-Based Adaptive Control of Two-Input/Two-Output Systems Using ART2-A Neural Networks," *Ind. Eng. Chem. Res.*, **33**, 1510 (1994).

Minsky, M. L. and S. A. Papert, *Perceptions*, expanded edition, MIT Press, Cambridge, MA (1988).

Nelson, M. C. and W. T. Illingworth, *A Practical Guide to Neural Nets*, Addison-Wesley, Reading, MA (1991).

NeuralWare, Inc., *Neural Computing: A Technical Handbook for Professional II/PLUS and NeuralWorks Explorer*, Pittsburgh, PA (1993).

Nikolaou, M., "Neural Network Modeling of Nonlinear Dynamic Systems," *Proceedings Amer. Control Conf.*, p. 1460, San Francisco, CA, June (1993).

Nikolaou, M. and V. Hanagandi, "Recurrent Neural Networks in Decoupling Control of Multivariable Nonlinear Systems," *AIChE Annual Meeting*, Miami, FL, Nov. (1992).

Nikolaou, M. and V. Hanagandi, "Control of Nonlinear Dynamic Systems Modeled by Recurrent Neural Networks," *AIChE J.*, **39**, 1890 (1993).

Pao, Y. H., *Adaptive Pattern Recognition and Neural Networks*, pp. 197-222, "The Functional-Link Net: Basis for an Integrated Neural-Net Computing Environment," Addison-Wesley, Reading, MA (1989).

Pearlmutter, B. A., "Learning State Space Trajectories in Recurrent Neural Networks," *Neural Computation*, **1**, 263 (1989).

Quantrille, T. E. and Y. A. Liu, *Artificial Intelligence in Chemical Engineering*, Academic Press, San Diego, CA (1991).

Rich, E. and K. Knight, *Artificial Intelligence*, Chapter 18, pp. 487-525, McGraw-Hill, New York, NY (1991).

Rieger, A. K., J. M. Irvine and T. P. Voge, "Rescaling of Variables in Back Propagation Learning," *Neural Networks*, **4**, 225 (1991).

Rumelhart, D., G. Hinton and R. Williams, "Learning Representations by Backpropagating Errors," *Nature*, **323**, 533 (1986).

Rumelhart, D. E., J. L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Volume I - Foundations*, MIT Press, Cambridge, MA (1986).

Samdani, G., "Fuzzy Logic: More Than a Play on Words," *Chem. Eng.*, **100**, No. 2, 30, February (1993).

Sanchez-Sinencio, E. and C. Lan, editors, *Artificial Neural Networks: Paradigms, Applications and Implementations*, IEEE Press, New York, NY (1992).

Schillen, T. B., "Designing a Neural Network Simulator: The MENS Modeling Environment for Network Systems I," *CAB10S Review*, 7, 417 (1991).

Simpson, P. K., *Artificial Neural Systems*, Pergamon Press, New York, NY, (1990).

Suenatanakull, W., "A Comparison of Fault Detection and Classification Using Artificial Neural Networks," Ph.D. dissertation, University of Texas, Austin, TX (1993).

Venkatsubramanian, V. and T. J. McAvoy, guest editors, "Special Issue on Neural Network Applications in Chemical Engineering," *Comput. Chem. Eng.*, **16**, No.4, pp.227-423, April (1992).

Watanabe, Kajiro, S. Hirota, L. Hou and D. M. Himmelblau, "Diagnosis of Multiple Simultaneous Faults via Hierarchical Artificial Neural Networks," *AIChE J.*, **40**, 839 (1994).

Werbos, P. J., "Generalization of Backpropagation with Applications to a Recurrent Gas Market Model," *Neural Networks*, **1**, 339 (1988).

Whiteley, J. R., "Knowledge-Based Interpretation of Process Sensor Patterns," Ph.D. dissertation, Ohio State University, Columbus, OH (1991).

Whiteley, J. R. and J. F. Davis, "Knowledge-Based Interpretation of Sensor Patterns," *Comput. Chem. Eng.*, **16**, 329 (1992).

Whiteley, J. R. and J. F. Davis, "Qualitative Interpretation of Sensor Patterns," *IEEE Expert*, **8**, No.2, 54 (1993).

Whiteley, J. R. and J. F. Davis, "A Similarity-Based Approach to Interpretation of Sensor Data Using Adaptive Resonance Theory," *Comput. Chem. Eng.*, **7**, 637 (1994).

Wray, J. and G. G. R. Green, "How Neural Networks Work: The Mathematics of Networks Used to Solve Standard Engineering Problems," *Proceedings of Amer. Control. Conf.*, p. 2311, Boston, MA, June (1991).

You, Y. and M. Nikolalou, "Dynamic Process Modeling with Recurrent Neural Networks," *AIChE J.*, **39**, 1654 (1993).

Zurada, J. M., *Introduction to Artificial Neural Systems*, West Publishing Company, St. Paul, MN (1992).

Chapter 3

Classification : Fault Diagnosis and Feature Categorization

To the best of our knowledge, the first reported application of neural networks in chemical engineering is the paper by Hoskins and Himmelblau (1988), who applied a neural network to the fault diagnosis of a chemical reactor operation. Since then, the number of publications on neural networks in both bioprocessing and chemical engineering has been growing rapidly. This chapter describes how to apply neural networks specifically to classification problems in those fields. We focus on using neural networks to identify operational faults (i.e., fault diagnosis) and to characterize distinct features (i.e., feature categorization) of both time-independent (steady-state) and time-dependent (unsteady-state) processes.

3.1 Overview of Classification Neural Networks

Applications of neural networks to classification problems in bioprocessing and chemical engineering fall into two major areas: (1) identification of process faults based on the operating conditions of a given process, and (2) prediction of the most likely categorical group for a given input pattern, for example, identification of the cell-growth-phase categories (induction phase, growth phase, stationary phase, or death phase) for fermentation processes (Section 5.4).

Pattern-recognition methods such as neural networks are important for classification problems because they do not require accurate process models, which are often difficult to obtain for many biological and chemical processes. Statistical methods provide another alternative to this problem. However, according to Suewatanakul (1993), neural computing outperforms the conventional statistical approach in many chemical engineering applications. Consequently, we do not consider statistical classifiers in this book (except in the internal architecture of the radial-basis-function network, Section 3.2A).

A. Types of Pattern Classifiers

Pattern classifiers map distinct input patterns onto their respective output classes. There are two basic types of pattern classifiers: parametric and nonparametric. Leonard (1991)

introduces two additional types, discriminant and similarity classifiers. We define these four classifiers as follows:

(1) *parametric classifier* - Has a probability distribution associated with the output responses being classified, and is trained by predicting the best parameters for the statistical distribution.

(2) *nonparametric classifier* - Does not have a probability distribution for the output responses, but instead predicts the output class by comparing the output class to its nearest training points.

(3) *similarity classifier* - Uses the training examples to develop a model for each class, and compares future test cases to this model in order to assign a score to define similarity. Decisions are then made based on those scores. This technique is also called a minimum-distance classifier.

(4) *discriminant classifier* - Identifies decision regions and surfaces from training examples without modeling each class independently as similarity classifiers do.

Figure 3.1 illustrates a standard neural network architecture used in pattern classification. The neural network maps a set of input patterns (e.g., process operating conditions) to respective output classes (e.g., categorical groups). We use an input vector, **x**, and an output vector, **y**, as defined in Section 2.2.A, to represent the input pattern and output class, respectively. The output vector, **y**, from the neural network is

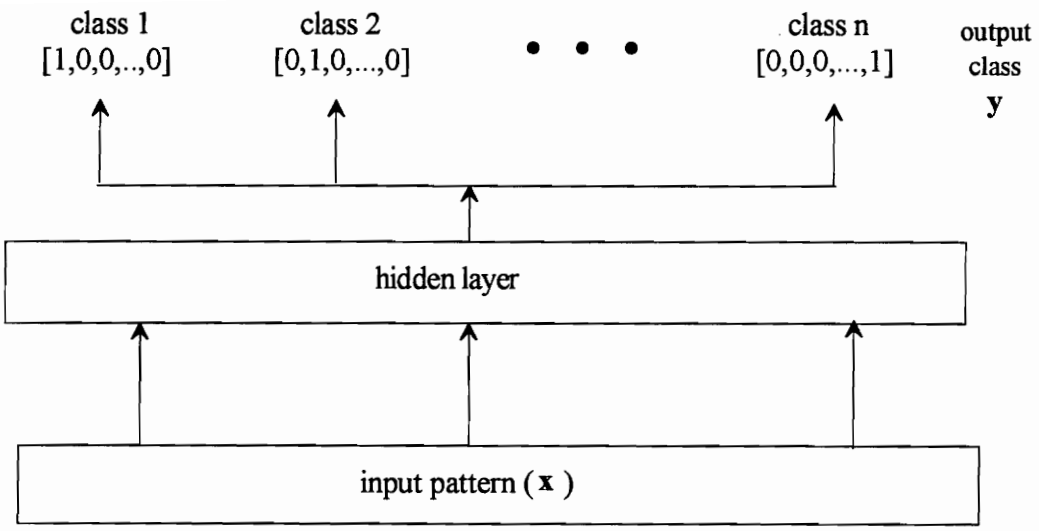


Figure 3.1. A standard neural network architecture for pattern-classification networks.

Boolean, with zero indicating that the input pattern is not within the specific class, and one indicating that it is within a specific class (e.g., "0" = not in class 2; "1" = in class 2). The *actual* output from the neural network is a numerical value between 0 and 1, and can be viewed as the "probability" that the input pattern corresponds to a specific class.

B. Two-Dimensional Classification Problems

Figure 3.2 graphically defines the input space, decision regions, decision boundaries, and transition regions for a two-dimensional classification problem. To define the input space, we use a simple two-component input pattern, input vector $\mathbf{x} = \{x_1 \text{ and } x_2\}$. The output

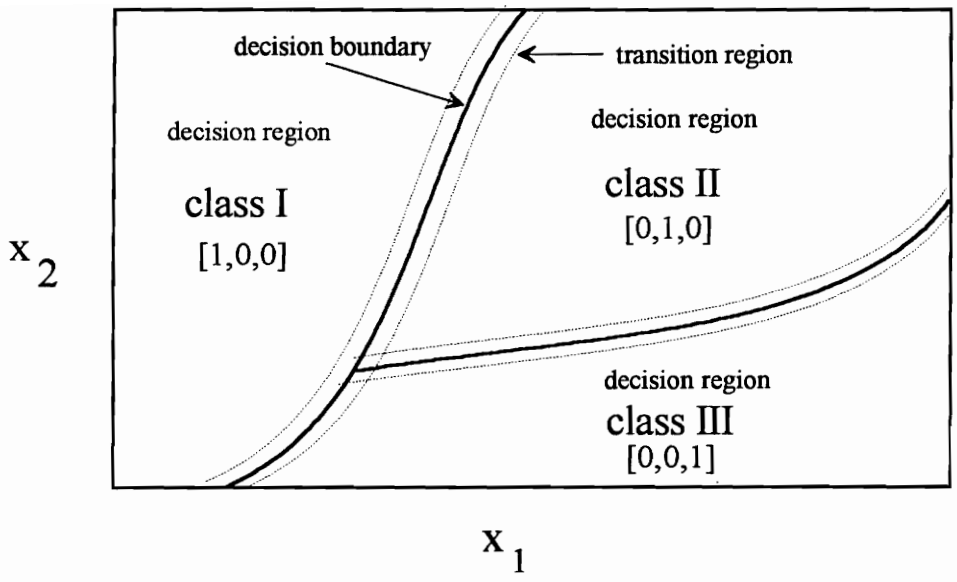


Figure 3.2. An illustration of a two-dimensional input space, decision regions, decision boundaries, and transition regions for a classification problem.

vector **y** contains three possible classes, i.e., **y** = {class I, class II, class III}. Note that for every point within the input space, there must be one and only one class specified. This example has only three possible output vectors for training the network, **y** = {[1,0,0], [0,1,0], or [0,0,1]}. The two-dimensional input space, in Figure 3.2, is constrained by feasible operating limits of the input variables, x_i (i = 1 to n), that is, (1) $x_{1,min} < x_1 < x_{1,max}$; and (2) $x_{2,min} < x_2 < x_{2,max}$. The possible output classes are mapped within this two-dimensional space.

What are the major regions in a classification problem? We define the three major regions as follows:

(1) *decision region* : a specific region within the input space which corresponds to a unique output class. All points within this region contain one and only one output class. Note that the input space can have multiple decision regions corresponding to multiple output classes. Figure 3.2 has three decision regions, one for each output class.

(2) *decision boundary* : the boundary is the intersection of two different decision regions. In Figure 3.2, the decision boundary between classes I and II would have an output vector of $\mathbf{y} = [0.5,0.5,0]$. This example has three decision boundaries: (i) between class I and class II, (ii) between class I and class III, and (iii) between class II and class III.

(3) *transition region* : this area is the buffer between two different decision regions. Here, we can make only fuzzy inferences about the classification because the predicted output vector is not $\mathbf{y} = \{[1,0,0], [0,1,0], \text{or } [0,0,1]\}$. For example, the transition region between class I and class II begins as the class I output response, then starts to decrease from 1, and ends when it reaches 0. Similarly, in the transition region, the class II output response increases from 0 to 1.

Figure 3.3 shows another effective way to visualize the transition region and decision boundaries for classification problems. This figure illustrates how the network's

predictions will move from the class I decision region (**y** = [1,0,0]) to the class II decision region (**y** = [0,1,0]).

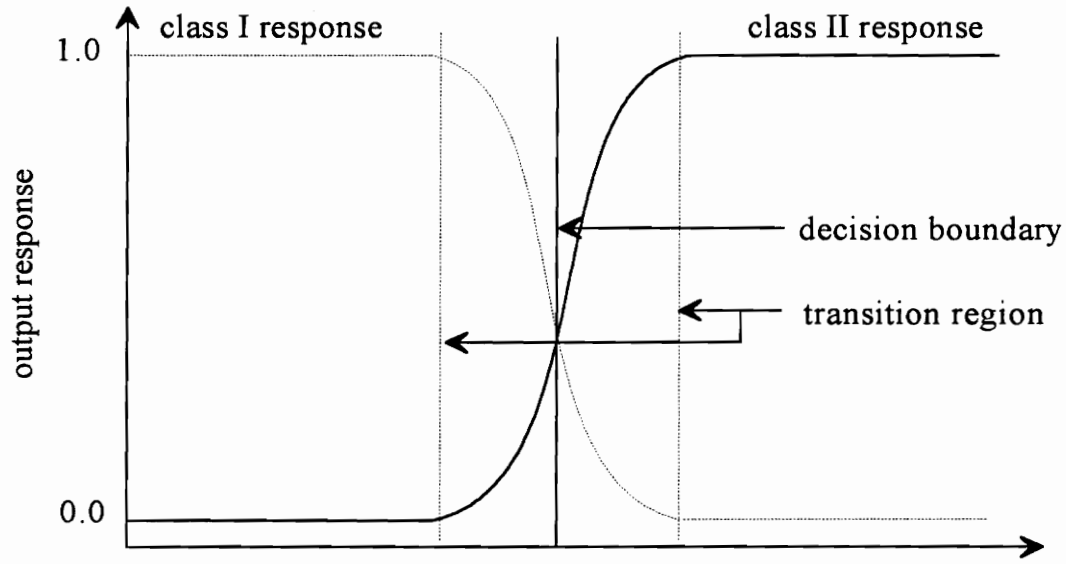


Figure 3.3. A classification network's predictions moving from the class I decision region (**y** = [1,0,0]) to the class II decision region (**y** = [0,1,0]).

C. Network Evaluation

The primary method for measuring the effectiveness of a neural network is misclassification rate, that is, the percentage of training (recall) and testing (generalization) examples misclassified from a given data set. In our design of a neural network model, we

wish to approach the *Bayes error rate*, defined as the minimal number of misclassifications based on process noise, sensor biases, etc.

Leonard (1991) identifies robustness to perturbations in the process as another critical property of classification networks. Specifically, we will define a system as either robust or brittle:

(1) *robust system* : the misclassification rate has minimal deviations with small variations in the input vector.

(2) *brittle system* : the misclassification rate has significant deviations with small variations in the input vector.

The misclassification rate and the robustness of the system are the two major model characteristics that we use to select optimal network type and configuration.

3.2 Radial-Basis-Function Networks

A. Network Architecture

The radial-basis-function network used in our software package, NeuralWare's Professional II/PLUS (1993), is based on the algorithm proposed by Moody and Darken (1989). Figure 3.4 shows a frequently used architecture for a radial-basis-function

network, which consists of three layers, including an input layer (N nodes), a hidden layer (L nodes), and an output layer (M nodes).

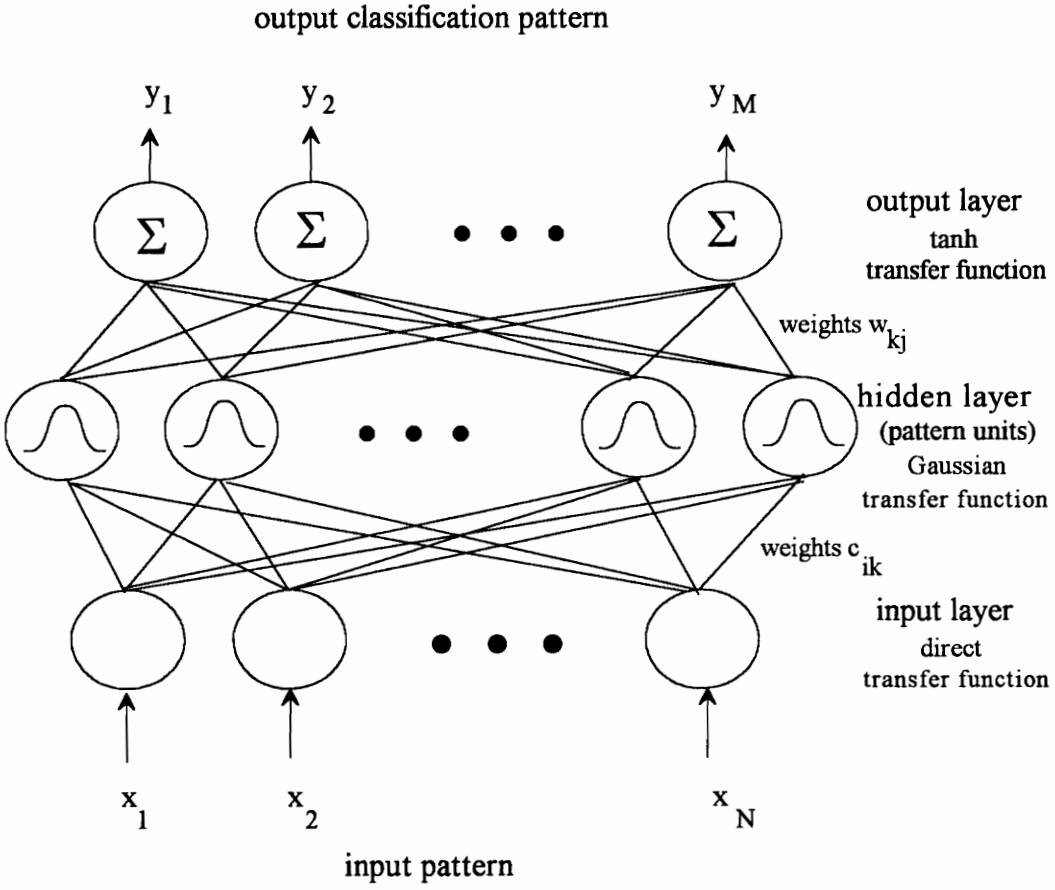


Figure 3.4. The architecture of a radial-basis-function network: input layer - N nodes (i = 1 to N); hidden layer - L nodes (k = 1 to L); and output layer - M nodes (j = 1 to M).

1. Input Layer

An input pattern enters the input layer and is subjected to a direct transfer function, i.e., the output of the node equals the input. Thus, for an input vector $\mathbf{x}$ with elements x_i (i = 1 to N), the output from the input layer is also vector $\mathbf{x}$.

2. Hidden Layer

The hidden layer is the most important processing step in a radial-basis-function network. Its nodes (also called pattern units) satisfy the unique property of being *radially symmetric*. Specifically, for a node to be radially symmetric, it must have the following:

(a) *A center vector* $\mathbf{c}_k$ in the input space, made up of cluster centers, with elements c_{ik} (i = 1 to N). The vector is typically stored as the weight factors from the input layer to the hidden layer (see Figure 3.4) ;

(b) *A distance measure* to determine how far an input vector $\mathbf{x}$, with elements x_i (i = 1 to N), is from the center vector $\mathbf{c}_k$. Typically, we use the standard Euclidean distance measure between $\mathbf{x}$ and $\mathbf{c}_k$ to define a Euclidean summation, I_k (k = 1 to L), where L is the number of nodes in the hidden layer:

$$I_k = \|\mathbf{x} - \mathbf{c}_k\| = \sqrt{\sum_{i=1}^{N} (x_i - c_{ik})^2} \qquad (3.1)$$

(c) *A transfer function* which transforms the Euclidean summation I_k (k = 1 to L) to give an output for each node. A common transfer function is a Gaussian function with a *mean* R being equal to the Euclidean norm $\|\mathbf{x}\|$ of the input vector **x** with elements x_i (i = 1 to N):

$$R = \|\mathbf{x}\| = \sqrt{\sum_{i=1}^{N} x_i^2} \tag{3.2}$$

This Gaussian function also has a width of σ_k (k = 1 to L). The resulting output from the k^{th} node, v_k (k = 1 to L), is:

$$v_k = \exp\left(\frac{I_k - R^2}{\sigma_k^2}\right) \tag{3.3}$$

To summarize, the hidden layer processes the output from the input layer in two steps, namely, a distance calculation (3.1) and a transfer function (3.3).

3. Output Layer

As Figure 3.4 shows , there are weight factors w_{kj} (k = 1 to L; j = 1 to M) between the k^{th} node in the hidden layer and the j^{th} node in the output layer. We find the output y_j (j = 1 to M) from the output layer through a standard procedure as in the backpropagation network (Section 2.2), that is, through the weighted sum of the output v_k (k = 1 to L) from the

hidden layer followed by a transfer function such as the sigmoid or hyperbolic tangent function.

B. Network Development

1. Cluster Centers, $\mathbf{c}_k$

To develop a radial-basis-function network, we must first determine the values of the stored center vector $\mathbf{c}_k$ (k = 1 to L) with elements c_{ik} (i = 1 to N; k = 1 to L) and the width of the Gaussian function σ_k (k = 1 to L).

A popular scheme to find $\mathbf{c}_k$ is an adaptive K-means clustering algorithm (Moody and Darken, 1989; NeuralWare, 1993). We start by assuming an initial set of L cluster centers, i.e., center vector $\mathbf{c}_k$, which corresponds to the L nodes in the hidden layer. The elements, c_{ik} (i = 1 to N; k = 1 to L) of $\mathbf{c}_k$ are stored as the weight factors between the input and hidden layers. We also assume that there are T training examples available to the input layer with N nodes, and represent them by T training vectors $\mathbf{x}^{(t)}$ with element $x_i^{(t)}$ (i = 1 to N; t = 1 to T). The adaptive K-means clustering algorithm then iteratively finds a desirable set of L center vectors $\mathbf{c}_k$ that will minimize the sum of the squares of the distance between T training vectors $\mathbf{x}^{(t)}$ and their nearest L centers $\mathbf{c}_k$ (k = 1 to L).

Figure 3.5 illustrates a simple example with three cluster centers $\mathbf{c}_k$ (k = 1 to 3) in a two-dimensional input space for a network having three nodes in the hidden layer (L=3).

As seen, the activation of the node or its output response, v_k, reaches its maximum at the center point and decreases as the distance between the input value, $x_i^{(t)}$, and the cluster center, c_{ik}, increases. For example, $x_1^{(t)}$ has a moderate output response, v_3, for node 3, but does not activate nodes 1 and 2 (v_1 and v_2, respectively). Note that the output response, v_k, is zero outside the cluster centers (e.g., $x_2^{(t)}$ is not within any of the three cluster groups and has no output response associated with it).

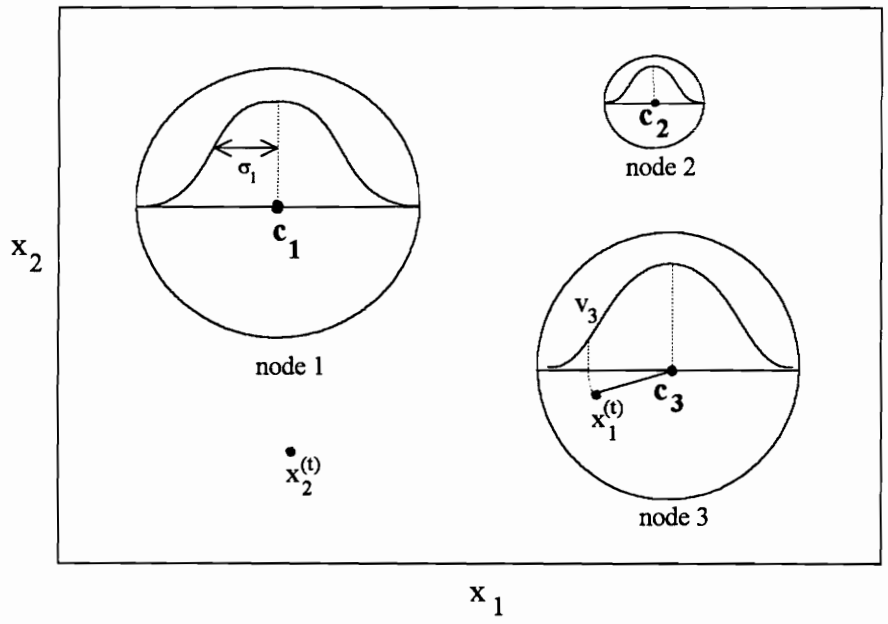


Figure 3.5. Three cluster centers $\mathbf{c}_k$ (k = 1 to 3) in a two-dimensional input space for a radial-basis-function network having three nodes in the hidden layer (L=3).

The K-means clustering algorithm used by NeuralWare's Professional II/PLUS (1993) involves the following steps.

Initialization Step:

1. Assume a set of cluster centers, $\mathbf{c}_k$, for the L nodes in the hidden layer, with the elements c_{ik} (i = 1 to N; k = 1 to L) stored as the weight factors between the input and hidden layers.

Iterative Steps:

1. Read the next training vector $\mathbf{x}^{(t)}$ with elements $x_i^{(t)}$ (i = 1 to N) into the input layer as t increases from 1 to T.
2. Modify only the cluster center $\mathbf{c}_k$ (k = 1 to L) closest to the training vector $\mathbf{x}^{(t)}$ in the Euclidean distance:

$$\mathbf{c}_k^{(new)} = \mathbf{c}_k^{(old)} + \alpha \left(\mathbf{x}^{(t)} - \mathbf{c}_k^{(old)} \right) \qquad (3.4)$$

   where α is a learning rate which decreases as t increases from 1 to T.
3. Continue this process for a fixed number of iterations.

2. Gaussian Function, σ_k

An effective scheme to find σ_k (k = 1 to L) is the P-nearest neighbor heuristic (Moody and Darken, 1989; NeuralWare, 1993). Consider a given center vector $\mathbf{c}_k$ (k = 1, 2, ..., or L) and assume that c_{k1}, c_{k2}, ..., c_{kp} ($1 \leq k1, k2, ..., kp \leq L$) are the P nearest neighboring centers. Then, the width of the Gaussian function, σ_k, is the root-mean-square (RMS) distance of the given cluster center $\mathbf{c}_k$ to the P nearest neighbor cluster center:

$$\sigma_k = \sqrt{\frac{1}{P} \sum_{p=1}^{P} \|\mathbf{c}_k - c_{kp}\|^2} \tag{3.5}$$

3. Weight Factors, w_{kj}

The weight factors w_{kj} (k = 1 to L; j = 1 to M) between the output of the hidden layer, v_k, and the output layer, y_j, are trained after the completion of the K-means clustering procedure using the conventional backpropagation training algorithms described in Chapter 2.

$$y_j = f(\Sigma w_{kj} v_k - T_j) \tag{3.6}$$

where $f(\,)$ represents the transfer function of the output nodes and T_j is the internal threshold.

4. Network Training

Training of a radial-basis-function network using the K-means clustering algorithm and the P-nearest neighbor heuristic has two important features. First, we do not use any desired output to train the input connections to the hidden layer. Second, for any given training example expressed by the t^{th} training vector $\mathbf{x}^{(t)}$ with elements $x_i^{(t)}$ (i = 1 to N), we iteratively modify only one cluster center $\mathbf{c}_k$ (k = 1, 2, ..., or L) closest to the t^{th} training vector $\mathbf{x}^{(t)}$ in the Euclidean distance. In other words, we only change those stored weight factors c_{ik} (i = 1 to N; k = 1, 2, ..., or L) between the input and hidden layers that are elements of *one selected center vector* $\mathbf{c}_k$ (k = 1, 2, ..., or L). Thus, during the network training, only the small fraction of input nodes with centers very close to the training input needs to be evaluated and trained. This localized tuning speeds up network training considerably, making the title of the well-known paper on radial-basis-function networks by Moody and Darken (1989), "Fast Learning in Networks of Locally-Tuned Processing Units," quite fitting.

Table 3.1 shows a typical training schedule that we use throughout this text for developing radial-basis-function networks. As shown, the weight factors between the input and hidden layer, c_{ik} (i = 1 to N; k = 1 to L), are determined using the K-means clustering algorithm during the first 2000 iterations. Over this same time period, the learning rate and the momentum coefficient are set to 0 and essentially *no* training of the weight factors, w_{kj} (k = 1 to L; j = 1 to M), occurs. We call this initial period the

data-clustering phase (see Figure 3.10). After the 2,000th iteration, the weight factors, c_{ik} (i = 1 to N; k = 1 to L), are fixed for the remainder of the training procedure. The weight factors w_{kj} (k = 1 to L; j = 1 to M) are then trained using a conventional backpropagation algorithm.

Table 3.1. A typical training schedule used for the radial-basis-function network.

Hidden layer				
Training iteration	0-500	501- 1,000	1001- 1,500	1501- 2,000
Learning rate	0.3	0.15	0.075	0.0375

Output layer				
Training iteration	0-2,000	2001-12,000	12,001-32,000	32,001-72,000
Learning rate	0.0	0.15	0.08	0.02
Momentum coefficient	0.0	0.4	0.2	0.05

C. Practical Aspects

A simple example will demonstrate why the radial basis function performs so well for classification, but not for prediction. This example also allows us to compare the radial basis function to the sigmoid and hyperbolic tangent functions.

Figure 3.6a shows a 2-dimensional input space, x_1 and x_2.

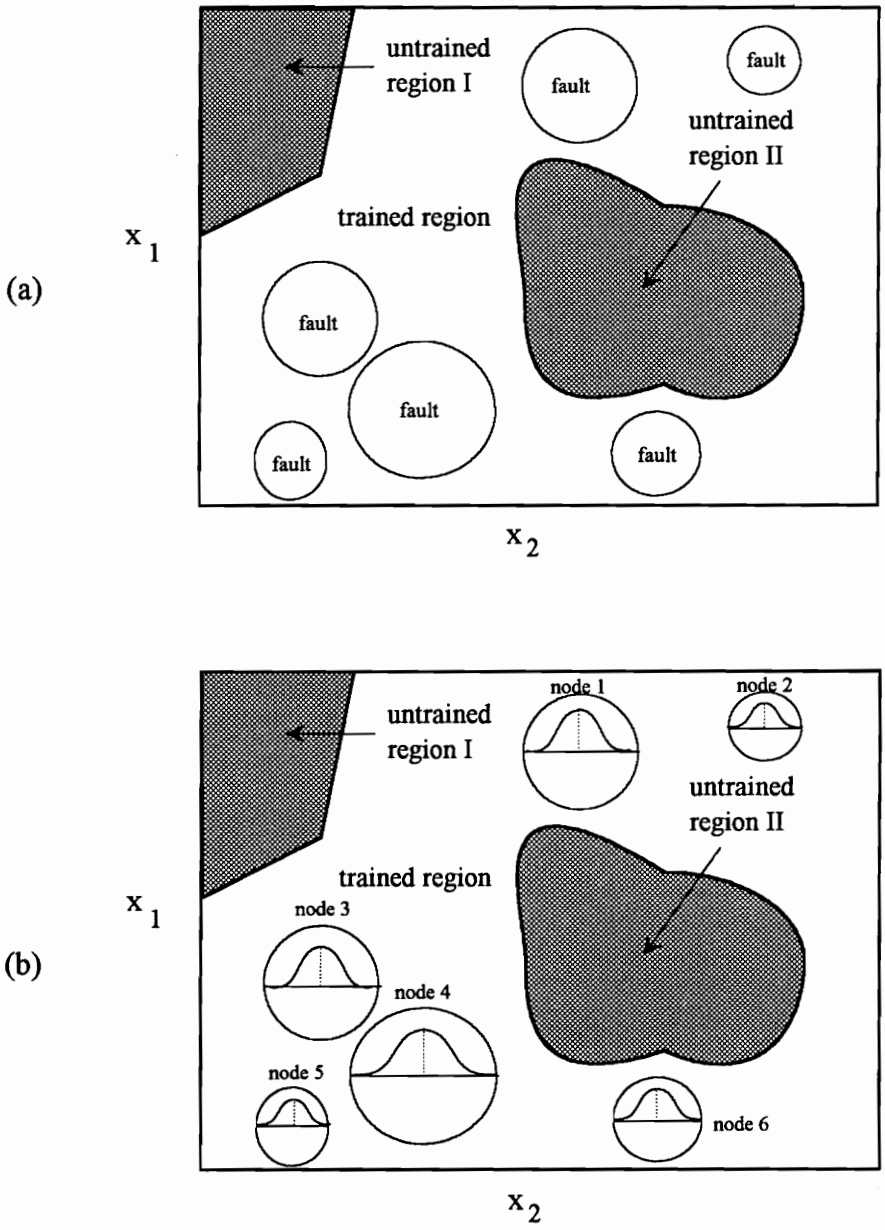


Figure 3.6. A 2-dimensional input space including a trained and an untrained region consisting of: (1) untrained region I representing an extrapolation area; and (2) untrained region II corresponding to an interpolation area. Six fault areas are identified in (a) and their respective nodes of the hidden layer in a radial basis function are shown in (b).

We divide this input space into two regions: (1) the trained region, where we have data sets to train the network; and (2) the untrained region, where we do not have data sets to train the network. We can then divide the latter region into subregions: (1) untrained region I, representing a set of input vectors where the network must *extrapolate* the output response; and (2) untrained region II, representing a set of input vectors where the network must *interpolate* the output response. Figure 3.6a also identifies six areas in the trained region where a fault exists. We assume the remaining area operates normally.

Figure 3.6b illustrates how the nodes of the hidden layer in a radial basis function correspond to the respective fault regions. The activation of the node is at a maximum in the center of the fault region, $f(x) = 1$, and decreases to $f(x) = 0$ as we move away from the fault regions. Therefore, untrained regions I and II will both have output responses of $f(x) = 0$ until the network receives training data sets in those regions. In comparison, the sigmoid and hyperbolic tangent transfer functions either increase or decrease as we move farther from $x = 0$ (e.g., for the hyperbolic tangent function, $f(x)$ approaches +1 as x increases to $+\infty$, and approaches -1 as x decreases to $-\infty$). These two transfer functions will generate a more uniform 3-dimensional surface, instead of one with 6 distinct peaks as produced by the radial basis function. Therefore, the hyperbolic tangent or the sigmoid functions may predict a fault or possible fault in untrained region II, because of the many surrounding fault regions (nodes 1, 2, 4, and 6). Predicting faults in untrained region I, may be even more difficult because that region is constrained by training data sets only on

one half of its outer boundary, possibly leading to more extreme extrapolations depending on the surrounding fault conditions. In conclusion, the sigmoid and hyperbolic tangent transfer functions can easily predict false faults and can have many fuzzy regions where fault prediction is unclear. On the other hand, the radial basis function may miss faults because the network has not seen given operating conditions during training, but it has well-defined fault boundaries and it rarely predicts false faults. Therefore, the radial-basis-function network consistently outperforms the backpropagation network for classification when there is a representative training data set.

For the same reasons the radial basis function performs so well for classification problems, it fails for prediction problems. A good predictive model, must be able to extrapolate into the untrained region I or interpolate within the untrained region II. We cannot have the network predict output responses of $f(x) = 0$ just because it has no training data in that region. The 3-dimensional surfaces of the sigmoid and hyperbolic functions, obviously, provide a much better basis for such interpolations and extrapolations. As a result, we do *not* recommend using the radial basis function for prediction problems, even with a database sufficient to eliminate untrained regions.

3.3 Comparison of Classification Neural Networks

The two most commonly used network architectures for classification problems are the backpropagation network and the radial-basis-function network. Leonard (1991) has

shown that backpropagation networks are poorly suited to fault diagnosis/classification problems, and that radial-basis-function networks are much more effective. Below are both the advantages and disadvantages of using radial-basis-function networks rather than traditional backpropagation networks (Leonard, 1991; NeuralWare 1993):

Advantages :

(1) Backpropagation networks are discriminant classifiers where the decision surfaces tend to be piecewise linear, resulting in non-robust transition regions between classification groups. In comparison, radial-basis-function networks use an explicit similarity-metric classifier to make decisions, leading to a more robust decision boundary.

(2) Training is typically faster for a radial-basis-function network.

(3) The internal representation within the hidden layers of a radial-basis-function network has a more natural interpretation. The pattern units within the hidden layer represent a density function for the input space and can be used to derive a measure of the probability that a new input vector is part of the same distribution as the training vector (Leonard, 1991).

Disadvantages :

(1) The initial learning phase of a radial-basis-function network is an

unsupervised data-clustering phase. Important discriminatory information could be lost in this phase.

(2) The radial-basis-function network may not perform well for prediction problems, which may require unbounded transfer functions (e.g., sigmoid or hyperbolic tangent functions) for effective modeling.

(3) Backpropagation networks can give a more compact distributed representation. In contrast, the radial-basis-function network requires a larger number of nodes in the hidden layer to perform adequately. In general, the backpropagation network will perform better for smaller network sizes (less than 10 to 15 nodes), but the radial-basis-function network will improve rapidly and eventually far-surpass the prediction capability of the backpropagation network as the number of nodes increases.

In addition to backpropagation and radial-basis-function networks, there are several other neural network types for classification networks, such as probabilistic network, learning-vector-quantization (LVQ) network, fuzzy artmap, and so on, as described in NeuralWare (1993). Suewatanakul (1993) compares these networks for a number of fault detection and classification problems. He finds that the best type of neural network depends on the nature of the problem. For problems with non-uniform decision regions (the Leonard and Kramer problem in Section 3.4.B.3), where the data points of each class are scattered, radial-basis-function and backpropagation networks perform

better. For problems with systematic patterns, distance-based classifiers like the learning-vector-quantization network work better. Since biological and chemical processes generally contain nonuniform decision regions, the radial-basis-function networks should be adequate for most classification problems.

3.4 Classification Neural Networks for Fault Diagnosis

A. Introduction

Neural networks are very effective in fault diagnosis for three reasons. First, through training the neural network can store knowledge about the process and learn directly from quantitative, historical fault information. We can train the network based on historically "normal" operation, and then compare that information to current operational data to determine faults. Second, neural networks can filter noise and draw conclusions in the presence of noise. We can train the networks to recognize important process information and disregard noise, enabling them to function effectively in a noisy environment. This filtering capacity makes neural networks well-suited for on-line fault detection and diagnosis. Finally, neural networks can identify causes and classify faults. Fault diagnosis requires pattern recognition, and we can train a neural network to identify both "normal" patterns of operation and patterns where faults exist. Moreover, if the network detects a

fault pattern, it can classify the fault and identify possible causes.

However practical problems can arise when using neural networks (Kramer and Leonard, 1990), particularly in fault diagnosis and classification. These are "operational" problems associated with actually implementing the neural network, and can crop up to such an extent that they significantly affect network performance. They include:

(1) *Use of undersized training sets*. For practical reasons, the amount of data available for training the network may be limited. For example, the training set may lack examples with a relatively low probability of occurrence toward the outer boundaries of the class. Consequently, new cases may fall into regions outside the training range. Networks with undersized training sets can also fail to model relative probability densities in those regions well-populated with training points, because of over-fitting.

(2) *Shifts in parent distributions of classes subsequent to training*. A process system is never static, and changes affecting the parent distributions of the fault classes may occur after the training set has been assembled and the network trained. These changes may include alterations in the process due to equipment degradation or maintenance actions, shifts in production levels or quality standards, and changes in exogenous conditions, such as the use of different raw material grades, or daily and seasonal changes. Such process changes can cause new cases to fall outside of the range of the original training data.

(3) *Corruption of incoming data by miscalibrated or malfunctioning sensors entering the network*. When sensors are out of service, substitute values must be input. Using assumed or inaccurate values may place the input outside the range of the original training data, forcing the network to extrapolate.

(4) *Appearance of a novel fault*. Because we cannot predict all possible faults, the training set may exclude certain failure modes. If a novel fault appears, it may fall into a new region of the input space. No classifier can diagnose novel fault types, but the classifier should indicate that the fault of an unknown type. Backpropagation network classifiers fail to provide such an indication, and instead classify the novel fault as one of the known types, or worse yet, as normal. Consequently, the backpropagation network could miss the movement of the process into a hazardous region. This behavior is unsuitable when the cost of the misdiagnosis is high, such as in hazardous processes or economically sensitive applications.

(5) *Use of synthetic training data*. The training examples may be from a numerical simulation of the process, rather than the process itself. In this case, the network may need to extrapolate because of process-model mismatch. A similar situation arises when training data come from a similar, but not identical process.

B. Boolean Fault Diagnosis

1. Overview of Boolean Fault Diagnosis

Venkatasubramanian and Chan (1989) introduce a method for using neural networks in fault diagnosis, and compare the results to a knowledge-based system. They diagnose faults in a fluidized catalytic cracking (FCC) unit, and identify eighteen symptoms (input nodes) and thirteen fault classifications (output nodes). The hidden layer ranges from five to twenty-seven nodes. Importantly, both the input and output in this system are *Boolean*, i.e., either 0 (no fault) or 1 (fault). Thus, we have an eighteen-dimensional Boolean input vector, such as:

(1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0)

and a thirteen-dimensional Boolean output vector, such as:

(0 0 0 0 0 0 1 0 0 0 0 0 0).

The goal is to have the network identify probable faults based on input symptoms.

Venkatasubramanian and Chan use backpropagation training, and their system performs fairly well. It can identify the correct source of faults 94% to 98% of the time.

However, because the system uses Boolean inputs rather than real-time data, the "fault" and "no fault" classifications are already in place *before* the neural network takes over. The network simply maps a Boolean input to a Boolean output. Mapping continuous input variables, on the other hand, is *far more* challenging. These limitations aside, Venkatasubramanian and Chan were among the first to successfully demonstrate the applicability of neural networks to pattern matching and fault diagnosis.

The neural network approach has two definite advantages over traditional expert-system techniques in principle:

- Neural networks can be implemented in real-time, which has proved a challenging problem in expert systems.
- Neural networks can directly utilize time-series data (or moving averages), while expert systems need some translation from numerical data into symbolic information.

However, while these statements are true *in principle*, implementing neural networks for real-time control is still a very challenging task.

2. Illustrative Example of Fault Diagnosis of a Chemical Reactor

a. Overview of the Chemical Reactor Fault-Diagnosis Network

We present a simple example of fault diagnosis using the chemical reactor described in Section 2.2.B. We use this example to compare the backpropagation network with the radial-basis-function network, and to compare different transfer functions (e.g. sigmoid and hyperbolic tangent functions) and network training rules (e.g.. delta rule and normalized cumulative delta rule). Temperature, pressure, and flow rate are the major input control variables (x_i) affecting the fault characteristics (y_j) of low conversion, low catalytic selectivity, and catalyst sintering (Table 3.2). We use this example to compare the backpropagation network with the radial-basis-function network. In addition, we compare different transfer functions (e.g. sigmoid and hyperbolic tangent) and network training rules (e.g., delta rule and normalized cumulative delta rule).

Table 3.2. Input and output for chemical reactor fault-diagnosis network.

Input vector	Output vector
x_1 : reactor inlet temperature, °F	y_1 : low conversion
x_2 : reactor inlet pressure, psia	y_2 : low catalyst selectivity
x_3 : feed flow rate, lb/min	y_3 : catalyst sintering

Figure 3.7 shows the architecture of the chemical reactor fault-diagnosis network. As in Section 2.2.B, the desired output from the neural network is Boolean: 0 indicates no problem, and 1 indicates that a problem does exist. The actual output from the neural network is a numerical value between 0 and 1, that we can almost view as the "probability" that the problem will result from this type of input (where 0 = 0% probability and 1 = 100% probability).

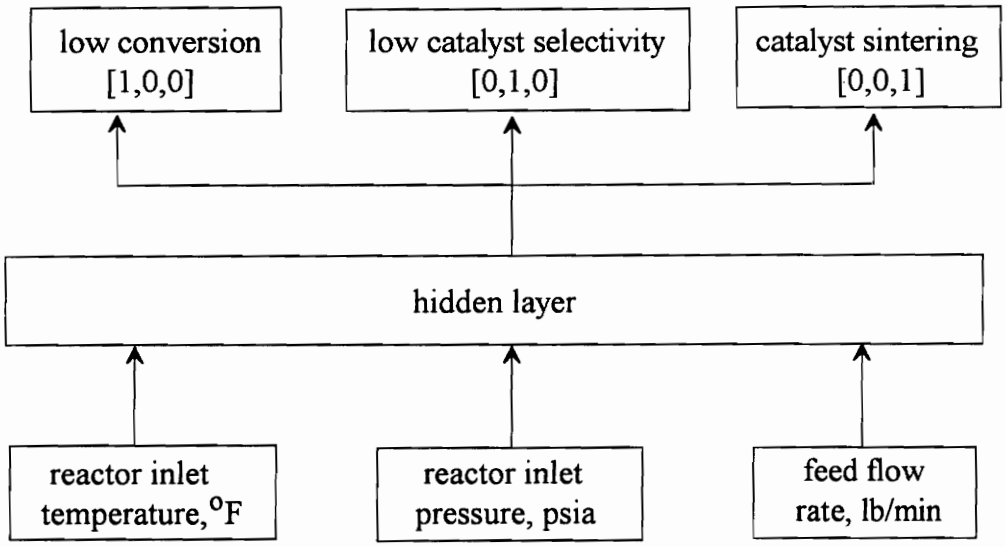


Figure 3.7. The architecture of the chemical reactor fault-diagnosis network.

Given the above inputs and outputs, we wish to create a fully operational neural network to perform fault diagnosis on process data using first a backpropagation and then a radial-basis-function network, both readily implemented on NeuralWare's Professional

II/PLUS (1993).

Table 3.3 lists the available input and output patterns for training the network. Note that the input patterns, x_1 to x_3, are normalized to between 0 and 1 prior to training (X_1 to X_3 in Table 3.3). The processing data used to train the neural networks include seven data sets with no faults, five with low conversion (y_1), two with low selectivity (y_2), and two with catalyst sintering (y_3).

Table 3.3. Input and output data for chemical reactor fault-diagnosis network.

x_1	x_2	x_3	X_1	X_2	X_3	y_1	y_2	y_3
400 ° F	100 psia	200 lb/min	0.40	1.00	0.20	0	0	0
420 ° F	100 psia	200 lb/min	0.42	1.00	0.20	0	0	0
380 ° F	100 psia	200 lb/min	0.38	1.00	0.20	0	0	0
400 ° F	100 psia	200 lb/min	0.40	1.00	0.20	0	0	0
400 ° F	90 psia	200 lb/min	0.40	0.90	0.20	0	0	0
400 ° F	100 psia	220 lb/min	0.40	1.00	0.22	0	0	0
400 ° F	100 psia	180 lb/min	0.40	1.00	0.18	0	0	0
300 ° F	100 psia	200 lb/min	0.30	1.00	0.20	1	0	0
325 ° F	90 psia	200 lb/min	0.325	0.90	0.20	1	0	0
350 ° F	80 psia	200 lb/min	0.35	0.80	0.20	1	0	0
370 ° F	100 psia	200 lb/min	0.37	1.00	0.20	1	0	0
380 ° F	100 psia	180 lb/min	0.38	1.00	0.18	1	0	0
400 ° F	100 psia	250 lb/min	0.40	1.00	0.25	0	1	0
400 ° F	100 psia	230 lb/min	0.40	1.00	0.23	0	1	0
550 ° F	100 psia	200 lb/min	0.55	1.00	0.20	0	0	1
525 ° F	100 psia	180 lb/min	0.525	1.00	0.18	0	0	1

Table 3.4 shows the format of the data file *fault.nna* used for training the chemical reactor fault-diagnosis network.

Table 3.4. The format of the file *fault.nna* used for training the chemical reactor fault-diagnosis network.

Column number	Variable type	Normalized variable	Normalization factor
1	input	ractor inlet temperature	1000 °F
2	input	reactor inlet pressure	100 psia
3	input	feed flow rate	1000 lb/min
4	output	low conversion	class [1,0,0]
5	output	low catalyst selectivity	class [0,1,0]
6	output	catalyst sintering	class [0,0,1]

b. Training with the Backpropagation Network

Table 3.5 summarizes the primary specifications used in training the chemical reactor fault-diagnosis network using backpropagation. We apply both hyperbolic tangent (tanh) and sigmoid transfer functions to compare their effectiveness for classification networks using the backpropagation algorithm. Similarly, we use both the delta and the normalized-cumulative-delta (nc-delta) learning rules for comparison.

Table 3.5. The specifications of the chemical reactor fault-diagnosis network.

Network type	backpropagation
Training file name	*fault.nna*
Transfer function (input layer)	linear
Transfer function (hidden layers)	sigmoid and tanh
Transfer function (output layer)	sigmoid and tanh
Learning rule	delta rule and nc-delta rule
Summation	sum
Error	standard
Network weight distribution	normal distribution: 3σ limits of [-1, 1]

Input Layer

Training iteration	5,000
Noise	0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0

Hidden layer

Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.3	0.15	0.04
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

Output layer

Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.15	0.08	0.02
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

We determine the optimal configuration for the hidden layers by varying the number of nodes in the hidden layer from 3 to 40. We test all four configurations of sigmoid versus hyperbolic tangent transfer functions, and delta versus nc-delta learning rules. Table 3.6 shows how the number of nodes in the hidden layer affects the network performance for each configuration. We train each network for 50,000 iterations.

Table 3.6. A comparison of the RMS error for training the chemical reactor fault-diagnosis network using the backpropagation algorithm.

# of nodes in the hidden layer	sigmoid delta-rule	sigmoid nc-delta-rule	tanh delta-rule	tanh nc-delta-rule
3	0.088	0.113	0.059	0.104
5	0.093	0.109	0.037	0.102
10	0.102	0.113	0.061	0.116
20	0.106	0.113	0.064	0.121
30	0.105	0.113	0.049	0.121
40	0.106	0.113	0.058	0.139

Figure 3.8 illustrates how the number of nodes in the hidden layer affects the RMS error for training data recall. The RMS error shows only an insignificant increase as the number of nodes in the hidden layer increases. This observation demonstrates that the backpropagation network has a compact model representation and does not require a

large network architecture to perform effectively. As Figure 3.8 shows, the minimal RMS error occurs with the hyperbolic tangent transfer function and the delta learning rule, and 5 nodes in the hidden layer.

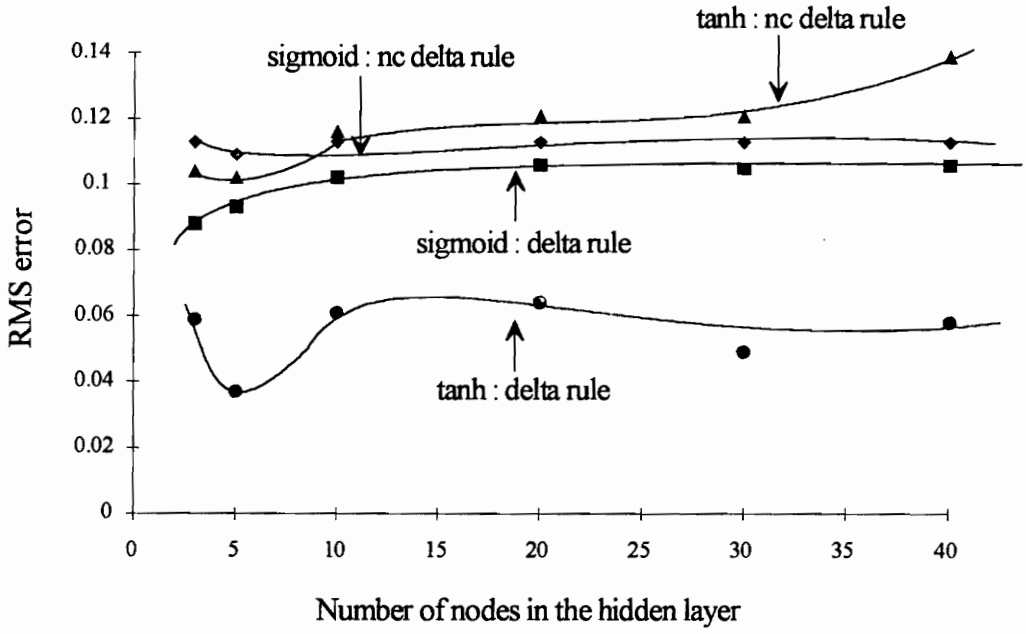


Figure 3.8. The RMS error for the recall of training data from the chemical reactor fault-diagnosis network using the backpropagation network as a function of the total number of nodes in the hidden layer.

Figure 3.9 shows the RMS error over the course of training the network under these conditions. The network is effectively trained in 20,000 to 30,000 iterations.

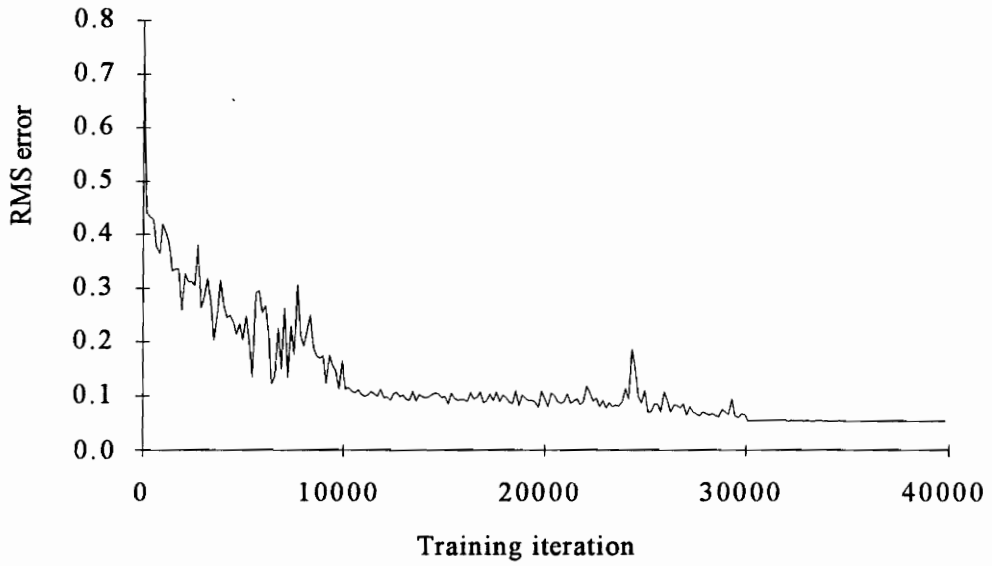


Figure 3.9. The training of the chemical reactor fault-diagnosis network using the backpropagation network with delta-learning rule and the hyperbolic tangent transfer function and 5 nodes in the hidden layer.

c. Training with the Radial-Basis-Function Network

Table 3.7 summarizes the specifications of the chemical reactor fault-diagnosis network using the radial-basis-function network. We use two different sets of learning specifications in this type of network, one for the hidden layer and one for the output layer (see Section 3.2). The output layer uses a standard backpropagation type of learning and the hidden layer uses a radial-basis-transfer function, a K-means learning rule, an Euclidean summation, and a two-nearest-neighbor (2-NN) error function.

Table 3.7. The specifications for the chemical reactor fault-diagnosis network using the radial-basis-function network.

Network type	radial-basis-function network
Training file name	*fault.nna*
Transfer function (input layer)	linear
Transfer function (hidden layer)	radial basis
Transfer function (output layer)	tanh
Learning rule (hidden layer)	K-means
Learning rule (output layer)	delta rule and nc-delta rule
Summation (hidden layer)	Euclidean
Summation (output layer)	sum
Error (hidden layer)	2-NN
Error (output layer)	standard
Network weight distribution	distribution range: [-1.0, 1.0]

Input layer

Training iteration	5,000
Noise	0.0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0.0

Hidden layer

Training iteration	500	1,000	1,500	2,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.3	0.15	0.075	0.0375
Cluster threshold	0.1	0.05	0.025	0.0

Output layer

Training iteration	2,000	12,000	32,000	72,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.0	0.15	0.075	0.01875
Momentum coefficient	0.0	0.4	0.20	0.05
Error tolerance	0.0	0.1	0.1	0.1

As in the backpropagation network training, we vary the number of nodes in the hidden layer from 3 to 40, and use the two different transfer functions and the two different learning rules to find the optimal architecture, as shown in Table 3.8 and Figure 3.10. Each configuration is trained with 50,000 iterations. The best network configuration uses the delta-learning rule and the hyperbolic tangent transfer function, with 30 nodes in the hidden layer.

Table 3.8. A comparison of the RMS error for training the chemical reactor fault-diagnosis network using the radial-basis-function network.

# of nodes in the hidden layer	simoid delta-rule	simoid nc-delta-rule	tanh delta-rule	tanh nc-delta-rule
3	0.153	0.167	0.354	0.367
5	0.110	0.126	0.187	0.222
10	0.088	0.100	0.113	0.136
20	0.039	0.058	0.029	0.054
30	0.002	0.021	0.001	0.001
40	0.001	0.005	0.001	0.001

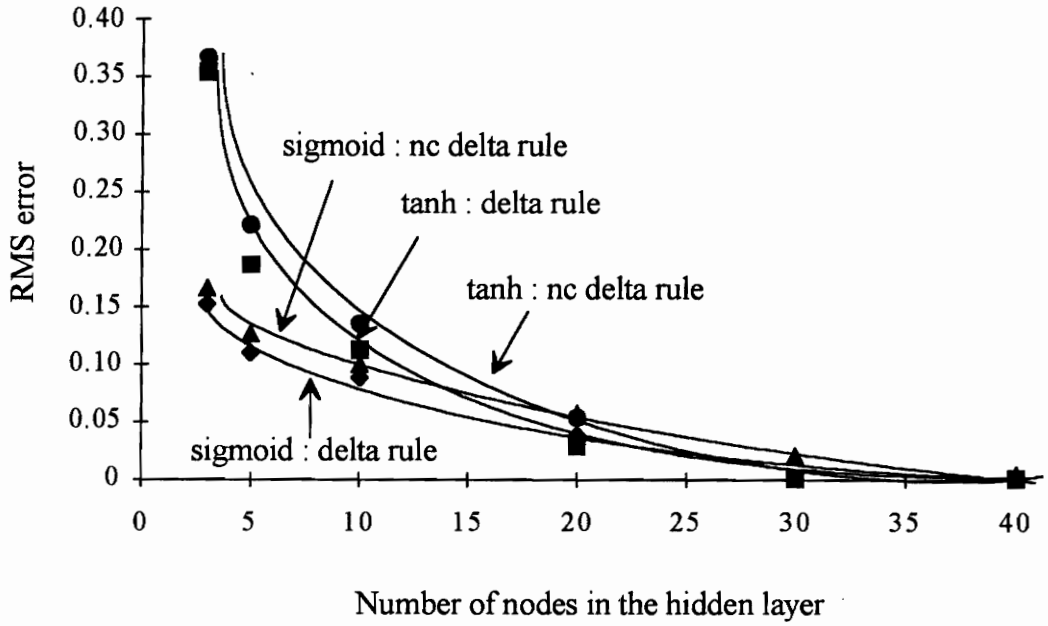


Figure 3.10. The RMS error for training the chemical reactor fault-diagnosis network using the radial-basis-function network.

Figure 3.11 illustrates the network training in terms of the RMS error versus the iteration number under optimal conditions. This figure shows the two distinct learning procedures, K-means clustering and backpropagation algorithms, as well. As discussed in Section 3.2, the weight factors between the input and hidden layer are determined using the K-means clustering algorithm during the first 2000 iterations. After the 2,000th iteration, we switch to the backpropagation algorithm.

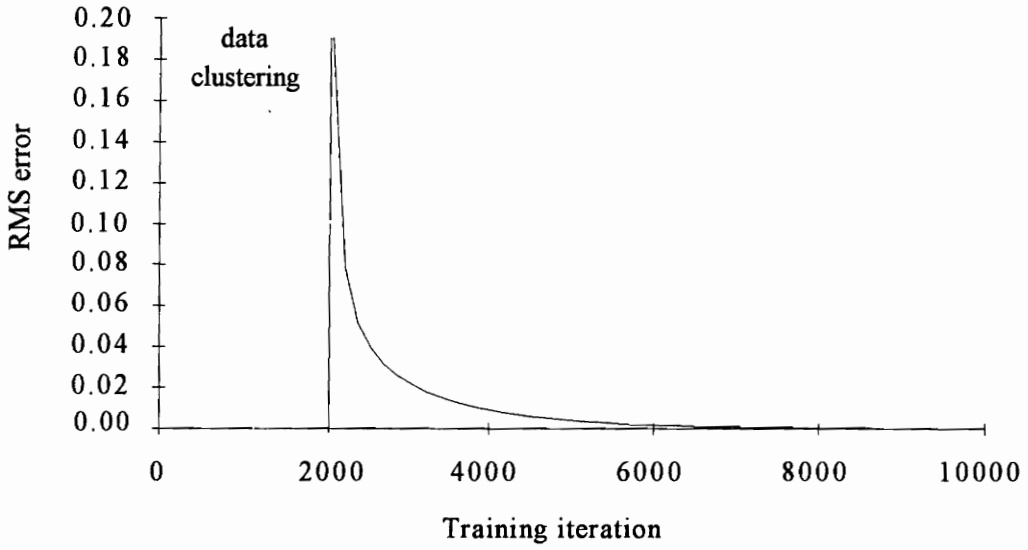


Figure 3.11. The training of the chemical reactor fault-diagnosis network using the radial-basis-function network with the delta-learning rule, the hyperbolic tangent transfer function, and 30 nodes in the hidden layer.

d. Comparison of the Backpropagation Network and Radial-Basis-Function Network

Our research confirms the findings of Leonard and Kramer (1991), showing that radial-basis-function networks perform better than backpropagation networks for classification problems. The radial-basis-function network's RMS error approaches 0 as the number of nodes increases, while the backpropagation network has a much larger RMS error of 0.037 (see Tables 3.6 and 3.8). We notice that the backpropagation network has a much more compact system representation than the radial-basis-function network, and has significantly lower RMS errors for smaller networks (e.g., less than 5

nodes). However, the prediction capability of the backpropagation network does not improve with additional nodes in the hidden layer. The radial-basis-function network also trains faster requiring only 7,000 iterations, versus 30,000 for the backpropagation network (Figures 3.8 and 3.10).

3. Illustrative Example of the Leonard-Kramer Problem

The Leonard and Kramer problem is widely used in studying classification problems and new network architectures. See, for example, Kramer and Leonard (1990), Holcomb and Morari (1991), Leonard and Kramer (1991), Bakshi and Stephanopoulos (1991), Hsiung (1992), and Suewatanakul (1993). This problem is two-dimensional, making it very easy to graphically demonstrate how the classification network works.

The Leonard and Kramer problem defines two measurement variables, x_1 and x_2. These variables consist of two process parameters, p_1 and p_2, and two noise variables, v_1 and v_2. The model for this system is simply:

$$x_1 = p_1 + p_2 + v_1 \tag{3.7}$$

$$x_2 = p_1 - p_2 + v_2 \tag{3.8}$$

The noise associated with each measurement variable, v_1 and v_2, has a Gaussian distribution with a mean of 0 and a standard deviation of 0.015. Values of 0 for the two

process parameters, p_1 and p_2, represent normal operating conditions, and deviations from 0 indicate a movement away from normal conditions. These two parameters have a Gaussian distribution with a mean of 0 and a standard deviation of 0.05. Based on this distribution, we set the process-fault classifications as:

$$\text{normal}: \quad |p_1| < 0.05 \quad |p_2| < 0.05$$

$$\text{fault 1}: \quad |p_1| > 0.05 \quad |p_2| < 0.05$$

$$\text{fault 2}: \quad |p_1| < 0.05 \quad |p_2| > 0.05$$

Figure 3.12 illustrates the operating regions of the measurement variables, x_1 and x_2, for the given distributions of the process parameters and noise variables.

Figure 3.13 shows the architecture of the Leonard-Kramer network. There are two input variables (x_1 and x_2) and three output classes (normal operating conditions, fault 1 occurring, and fault 2 occurring). For each output class, 1 means the process is operating under those conditions and 0 means it is not.

Figure 3.14 shows our data used to train and test the Leonard and Kramer network. The network uses 150 examples for training (file : *lktrn.nna)* and another 150 examples to testing (file : *lktst.nna*). Table 3.9 shows the format of these data files.

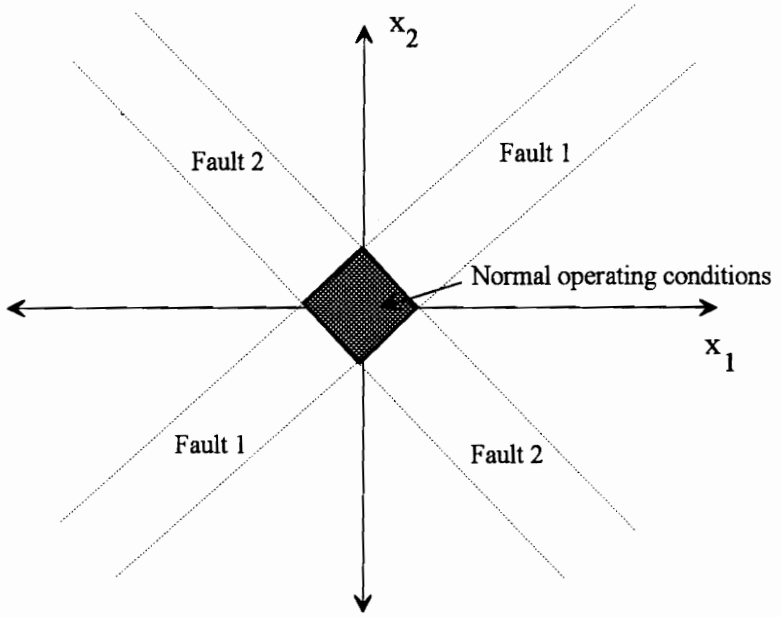


Figure 3.12. The operating regions of measurement variables, x_1 and x_2.

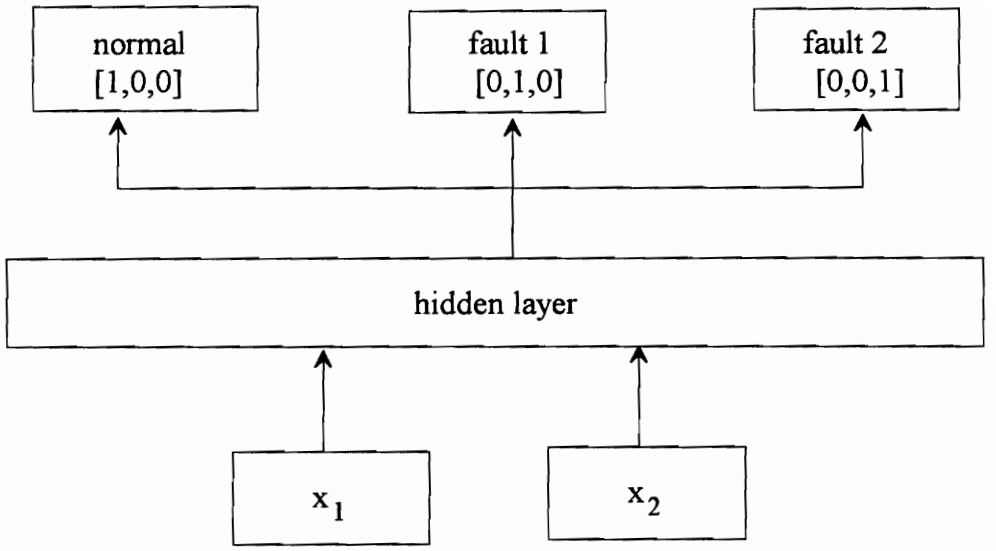


Figure 3.13. Architecture of the Leonard-Kramer network.

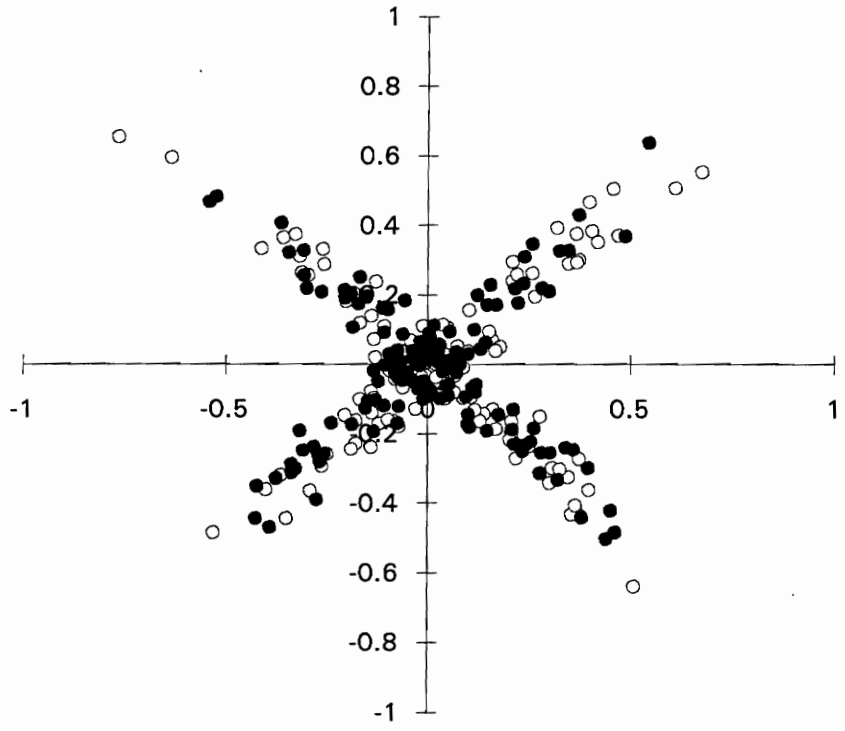


Figure 3.14. The data used to train and test the Leonard and Kramer network, (O) training data set (●) testing data set.

Table 3.9. The format of the file *lktrn.nna* and *lktst.nna* used for training and testing the Leonard-Kramer network.

Column number	Variable type	Normalized variable	Normalization factor
1	input	x_1	1
2	input	x_2	1
3	output	normal	class [1,0,0]
4	output	fault 1	class [0,1,0]
5	output	fault 2	class [0,0,1]

Here, we demonstrate the training and testing using the radial-basis-function network only. Table 3.10 summarizes the specifications of the radial-basis-function network using the delta-learning rule and the hyperbolic tangent transfer function. To compare this network with other classifying methods, we will use the results from other reported studies.

Table 3.11 and Figure 3.15 show the average error for recall of the training data set for the Leonard-Kramer network as the number of nodes in the hidden layer increases from 3 to 30. These average errors combine all three process classifications. We train each network using 50,000 iterations, and find that 15 nodes in the hidden layer are sufficient, since additional nodes produce no significant error reduction. This observation parallels the results presented by Leonard and Kramer (1991). We also find that the Leonard-Kramer network with 15 nodes in the hidden layer has a misclassification rate of 3.33 % for the training data set and 4.67 % for the testing data set.

Table 3.12 compares misclassification rates for the optimal backpropagation (6 nodes in the hidden layer) and the radial-basis-function networks (15 nodes in the hidden layer) as determined by Leonard and Kramer (1991). They use 10,000 points from the original distribution, compared to the 150 examples in our testing data set. The radial-basis-function network outperforms the backpropagation network, the networks having misclassification rates of 4.5 % and 7.7 %, respectively.

Table 3.10. The specifications for the Leonard-Kramer network using the radial-basis-function network.

Network type	radial-basis-function network
Training flie name	*lktrn.nna* and *lktst.nna*
Transfer function (input layer)	linear
Transfer function (hidden layer)	radial basis
Transfer function (output layer)	tanh
Learning rule (hidden layer)	K-means
Learning rule (output layer)	delta rule
Summation (hidden layer)	Euclidean
Summation (output layer)	sum
Error (hidden layer)	2-NN
Error (output layer)	standard
Network weight distribution	distribution range: [-1.0, 1.0]

Input layer

Training iteration	5,000
Noise	0.0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0.0

Hidden layer

Training iteration	500	1,000	1,500	2,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.3	0.15	0.075	0.0375
Cluster threshold	0.1	0.05	0.025	0.0

Output layer

Training iteration	2,000	12,000	32,000	72,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.0	0.15	0.075	0.01875
Momentum coefficient	0.0	0.4	0.20	0.05
Error tolerance	0.0	0.1	0.1	0.1

Table 3.11. A comparison of the average error for training the Leonard-Kramer network.

Number of nodes in the hidden layer	Average error
3	0.317
5	0.160
10	0.107
15	0.106
20	0.098
30	0.075

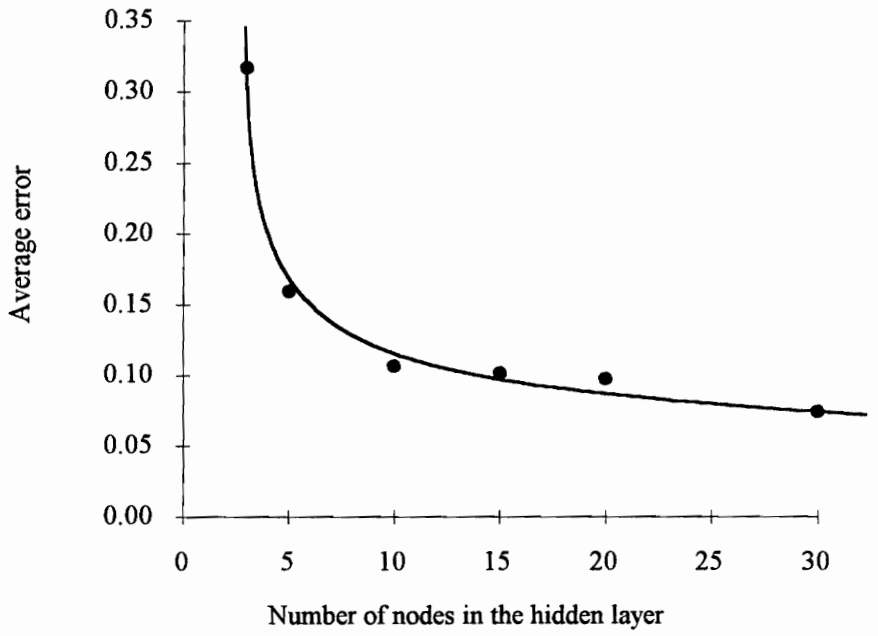


Figure 3.15. The average errors for the recall of the training data set for the Leonard-Kramer network as we increase the number of nodes in the hidden layer.

Table 3.12. Comparison of baseline misclassification rates (Leonard and Kramer, 1991).

Classifier	Classification error rate (%)			
	normal	fault 1	fault 2	mean
Backpropagation network	8.5	10.4	4.2	7.7
Radial-basis-function network	7.4	3.6	2.6	4.5

Leonard and Kramer (1991) define two types of errors that produce misclassification:

- *Detection error* occurs at the border between classes. It results from stochastic overlap of the classes and cannot be wholly avoided.
- *Extrapolation error* occurs when a new case falls beyond the range of the original training data and is misclassified.

They also find that all the error associated with the radial-basis-function network results from detection error; there is no extrapolation error, a major problem with backpropagation networks.

Figures 3.16 and 3.17 show the decision boundaries of the backpropagation and radial-basis-function networks, respectively (Leonard and Kramer, 1991). The

backpropagation network has large extrapolation errors, indicated by the additional regions where the network predicts either normal operating conditions or ties between the two fault regions. In contrast, the radial-basis-function network has more clearly defined decision boundaries with no extrapolation error present.

To better characterize the performance of the two networks, Leonard and Kramer (1991) perform a series of robustness tests. They use the original set of 10,000 testing data points and added small and large perturbations for sensor bias , sensor noise, changes in the characteristic direction of the deviation of the fault, and shifts in the steady state of the process. Table 3.13 shows the results of their robustness tests with the following perturbations:

sensor bias : $\pm$ 0.025 and $\pm$ 0.05
sensor noise : increase sensor noise by a factor of 2 or 3
direction : rotate $\pm$ 7.5° and $\pm$ 15°
process steady-state (x_o) shift : $\pm$ 0.025 and $\pm$ 0.05

The radial-basis-function network has lower misclassification rates than the backpropagation network for all cases, and has minimal increases in classification error rates for many of the process disturbances.

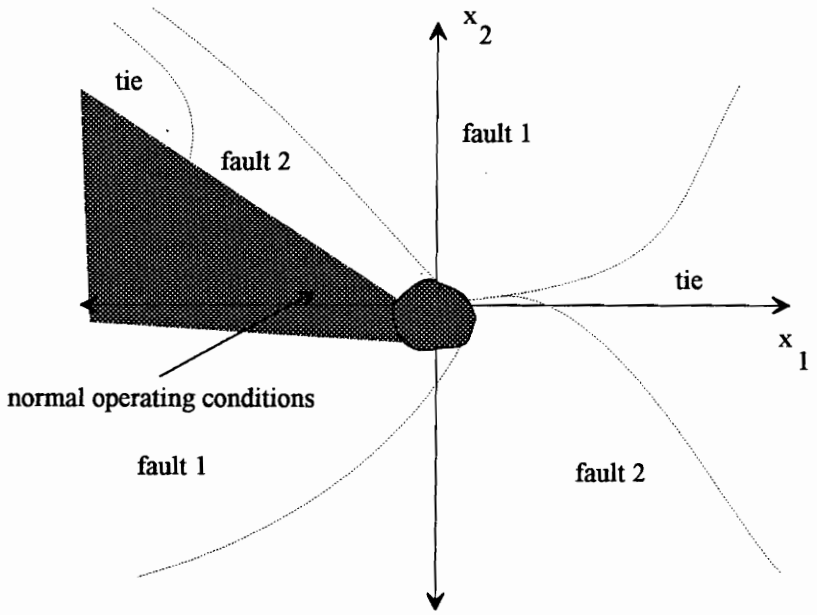


Figure 3.16. Predicted decision boundaries for the Leonard and Kramer problem using the backpropagation network with 6 nodes in the hidden layer (adapted from Leonard, 1991).

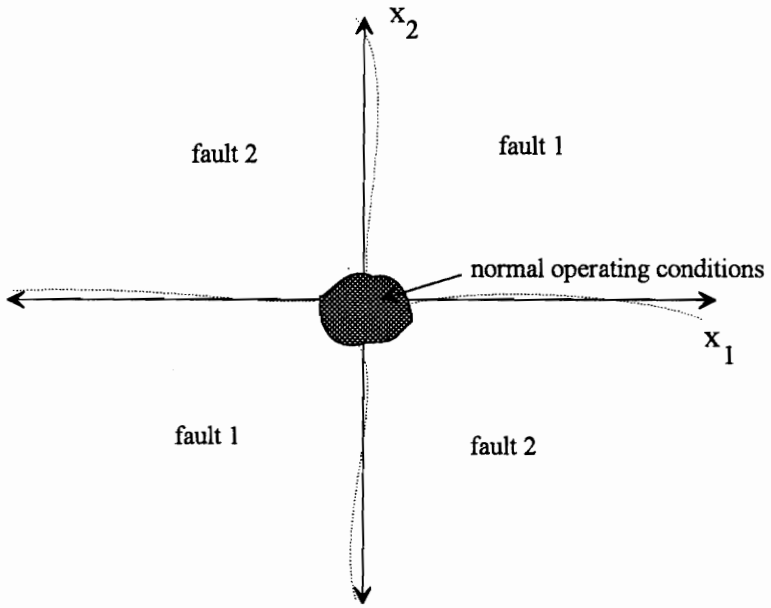


Figure 3.17. Predicted decision boundaries for the Leonard and Kramer problem using the radial-basis-function network with 15 nodes in the hidden layer (adapted from Leonard, 1991).

Table 3.13. Comparison of misclassification rates on robust diagnosis tests (Leonard and Kramer, 1991).

Classifier	Sensor bias		Sensor noise		Direction		Shift in $\mathbf{x}_0$	
	small	large	small	large	small	large	small	large
Backpropagation network	10.6	17.6	14.0	21.1	12.4	24.1	14.0	30.4
Radial-basis-function network	6.3	11.7	8.3	14.0	4.5	4.5	7.7	19.9

This case study shows that the radial-basis-function network outperforms the backpropagation network. The radial-basis-function network has a significantly lower misclassification rate (4.5 % to 7.7 %). Moreover, it is free of extrapolation errors, which are the major problem in the backpropagation network as the large normal operating region in Figure 3.15 indicates. Finally, the results from the robust diagnosis test by Leonard and Kramer (1991) further demonstrates the advantages of radial-basis-function networks for classification problems.

C. Fault Diagnosis with Continuous Variables

The Leonard and Kramer problem examined above focuses solely on a system with Boolean inputs. However, since many chemical processes operate with continuous inputs,

we now turn to several other examples. Hoskins and Himmelblau (1990b) use neural networks for fault diagnosis with *continuous* variables as inputs, considering two problems: 1) three isothermal continuous stirred-tank reactors (CSTRs) operating in series, and 2) a chemical reactor catalytically converting heptane to toluene. Note, though, that their work applies to steady-state systems only.

For the three CSTRs in series, Hoskins and Himmelblau use the neural network to identify six potential faults, listed in Table 3.14. They assume that all sensors operate properly, and use the neural network to classify input patterns based on a scalar decision function. They train the network using backpropagation, and discuss the training efficiency based on the number of hidden nodes. With only six input patterns in the training set, the neural network misdiagnoses only 20% of the time. With twelve input patterns, the system diagnoses properly 100% of the time.

Table 3.14. Faults monitored in a CSTR problem (Hoskins and Himmelblau, 1990b).

Fault	Description	Type
y_1	inlet flow rate	high
y_2	inlet flow rate	low
y_3	inlet concentration, component A	high
y_4	inlet concentration, component A	low
y_5	inlet temperature	high
y_6	inlet temperature	low

Hoskins and Himmelblau also use neural networks on a more sophisticated problem, fault diagnosis of a chemical reactor catalytically converting heptane to toluene. Here, they monitor five faults, listed in Table 3.15.

Table 3.15. Faults monitored in a heptane-conversion problem (Hoskins and Himmelblau, 1990b).

FAULT	DESCRIPTION
y_1	Physical and/or chemical deterioration of catalyst, resulting in lower frequency factor.
y_2	Fouling of reactor heat-exchange surface, reducing h_1, the reactor heat-transfer coefficient.
y_3	Fouling of pre-heater heat-exchange surface, reducing h_2, the pre-heater heat-transfer coefficient.
y_4	Plugging of rector recycle line, leading to decreased volumetric flow to the reactor.
y_5	Plugging of rector recycle line, leading to decreased volumetric flow in the recycle stream.

Hoskins et. al. (1990c) extend this work to a much more complex process, a continuous stirred-tank reactor, a plug-flow reactor, and the associated heat-exchange and separation equipment. Hoskins et. al. identify 19 process faults, average for a small-to-medium sized unit in the chemical process industry. Typically, their system identifies the fault with an 85% accuracy rate.

Another example of fault diagnosis with continuous variables is Venkatasubramanian et. al. (1990), who also perform fault diagnosis on a CSTR. They

identify six faults - interestingly, nearly the exact same faults described in Hoskins and Himmelblau (1990b) (Table 3.14). Not surprisingly, the two studies yield similar results. The neural network of Venkatasubramanian et. al. performs well, with only a 3-7% error in fault identification. Training the network requires 417 time steps.

The results of these studies point to two significant conclusions:

- One strength of neural networks is the ability to filter noise and still draw conclusions, which makes them effective for continuous steady-state systems.
- That strength may become a *liability* when, as in this example, the network incorrectly views a faulty input pattern as noise.

D. Fault Diagnosis with Unsteady-State Systems

1. Introduction

We now turn our attention from steady-state to unsteady-state systems. To our knowledge, the first reported application of neural networks to fault diagnosis with unsteady-state or time-dependent chemical processes is the paper by Vaidynathan and Venkatasubramanian (1990). Their work involves an unsteady-state continuous stirred-tank reactor (CSTR) system with the same six fault classifications listed in Table 3.14, and uses the backpropagation network. As we have already shown (Section 3.4.B),

the radial-basis-function network is better for classification problems. The following case study describes how to apply the radial-basis-function network to the fault diagnosis of the same unsteady-state CSTR system used by Vaidynathan and Venkatasubramanian (1990), and evaluates the relative merits of backpropagation and radial-basis-function networks for this problem.

2. Illustrative Case Study : An Unsteady-State CSTR System

a. Process Model

Vaidynathan and Venkatasubramanian (1990) model the CSTR as an irreversible, first-order reaction.

$$A \rightarrow B \tag{3.9}$$

They assume heat losses are negligible and the density is constant. The model consists of the following differential equations: total material balance (3.10), balance for component A, (3.11), and energy balances, (3.12) and (3.13).

$$\frac{dV}{dt} = F_0 - F \tag{3.10}$$

$$\frac{d(VC_A)}{dt} = F_0C_{A0} - FC_A - VkC_A \tag{3.11}$$

$$\frac{d(VT)}{dt} = F_0T_0 - FT - \frac{\lambda VkC_A}{\rho C_p} - \frac{UA_H}{\rho C_p}(T - T_j) \quad (3.12)$$

$$\frac{dT_j}{dt} = \frac{F_j(T_{j0} - T_j)}{V_j} + \frac{UA_H}{\rho_j V_j C_j}(T - T_j) \quad (3.13)$$

where

F_0	:	inlet flow rate
F	:	outlet flow rate
V	:	reactor volume (holdup)
C_{A0}	:	inlet concentration of component A
C_A	:	outlet concentration of component A
k	:	first-order reaction-rate constant
ρ	:	density of the reaction mixture
ρ_j	:	density of the cooling water
C_p	:	specific heat of the reaction mixture
C_j	:	specific heat of the cooling water
λ	:	heat of reaction
F_j	:	inlet flow rate of cooling water
T_{j0}	:	inlet temperature of cooling water
U	:	overall heat-transfer coefficient
T	:	outlet temperature of the reaction mixture
T_j	:	outlet temperature of cooling water

A_H : heat-transfer area between T and T_j

V_j : holdup of the cooling water within the jacket.

The Arhenius equation for the reaction-rate constant is:

$$k = \alpha_1 \exp\left(\frac{-E_a}{R_g T}\right) \tag{3.14}$$

where α_1 is the pre-exponential factor, E_a is the activation energy, and R_g is the ideal gas constant.

The equations representing the control of temperature (3.15), holdup (3.16), and concentration (3.17) are:

$$F_j = F_{j0} - K_c(T_{set} - T) \tag{3.15}$$

$$F = F_0 - K_1 (V_{set} - V) \tag{3.16}$$

$$F_0 = F_{00} + K_{c1} (C_{A0,set} - C_{A0}) \tag{3.17}$$

where K_c, K_1, and K_{c1} are controller gains.

Table 3.16 shows the process settings used by Vaidynathan and Venkatasubramanian (1990) for simulating the unsteady-state CSTR problem. The simulation has six state variables to predict the specified malfunctions:

C : outlet concentration of reaction mixture

T : outlet temperature of reaction mixture

V : reactor holdup

F : outlet flow rate of reaction mixture

T_j : outlet temperature of cooling water

F_j : cooling water flow rate

Table 3.16. CSTR settings used in the generation of the database for the training and testing of the unsteady-state CSTR fault-diagnosis network.

Normal steady-state values			
F, F_0, and F_{00}	40 ft^3/hr	V	48 ft^3
C_{A0}	0.5 $mole/ft^3$	C_A	0.245 $mole/ft^3$
T	600 °R	T_j	594.6 °R
F_{j0}	49.9 ft^3/hr	T_0	530 °R

Other parameters			
V_j	3.85 ft^3	α_1	$7.08*10^{10}$ hr^{-1}
E	30,000 BTU/mole	R_g	1.99 BTU/mole°R
U	150 $BTU/hrft^2°R$	A_H	250 ft^2
T_{j0}	530 °R	λ	-30,000 BTU/mole
C_p	0.75 $Btu/lb_m°R$	C_j	1.0 $BTU/lb_m°R$
ρ	50 lb_m/ft^3	ρ_j	62.3 lb_m/ft^3
K_c	4 $(ft^3/hr)/°R$	T_{set}	600 °R
K_1	10 $(ft^3/hr)/ft^3$	V_{set}	48 ft^3
K_{c1}	50 $(ft^3/hr)/(mole/ft^3)$	$C_{A0,set}$	0.5 $mole/ft^3$

2. Training and Testing the Network

The original data set for network training includes perturbations of 5 and 15% in the malfunctions of inlet flow rate, inlet concentration, and inlet temperature, for a total of 12 process runs. Normal operating conditions are those with deviations within 5% of the nominal steady-state values.

Vaidynathan and Venkatasubramanian (1990) use both raw time-series plant information and *average moving-window values* as input to the network. An average moving-window value is the average of the raw data from the last *n* time series. Figure 3.18 illustrates the moving window used as the input to the CSTR fault-diagnosis network. A key challenge in applying neural networks to unsteady-state or time-dependent processes is choosing both the "time-window width" (i.e., how much previous history is retained in the current calculation) and the time between data sampling points. Vaidynathan and Venkatasubramanian determine these parameters empirically based on process time-constants. They choose a window width of four sampling times at 15-minute time intervals. The equation for the average moving-window value is then:

$$\bar{x}_j(t) = \frac{\sum_{n=0}^{3} x_j(t-n\Delta t)}{4} \qquad (3.18)$$

where $x_j(t)$ is the j^{th} process variable at time t, and Δt is time increment between samples.

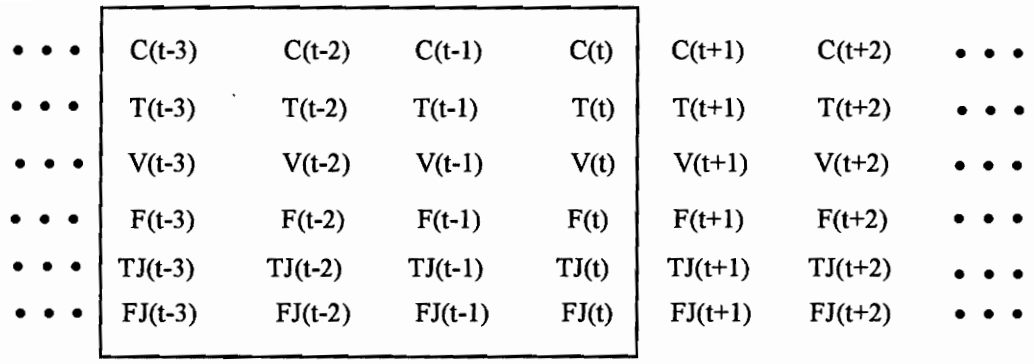


Figure 3.18. The moving window used for the input of the unsteady-state CSTR fault-diagnosis network at time t.

Figure 3.19 shows the architecture of the unsteady-state CSTR fault-diagnosis network, which includes six input variables and six fault classes. This example uses the radial-basis-function network, which, as Section 3.4.B explains, is better suited for classification problems. Vaidynathan and Venkatasubramanian (1990), in contrast, use the backpropagation network.

Table 3.17 gives the format of the data file *cstr.nna* and Table 3.18 summarizes the specifications used in training the network. The normalization ranges are set so that 0 and 1 represent the lower and upper limits, respectively.

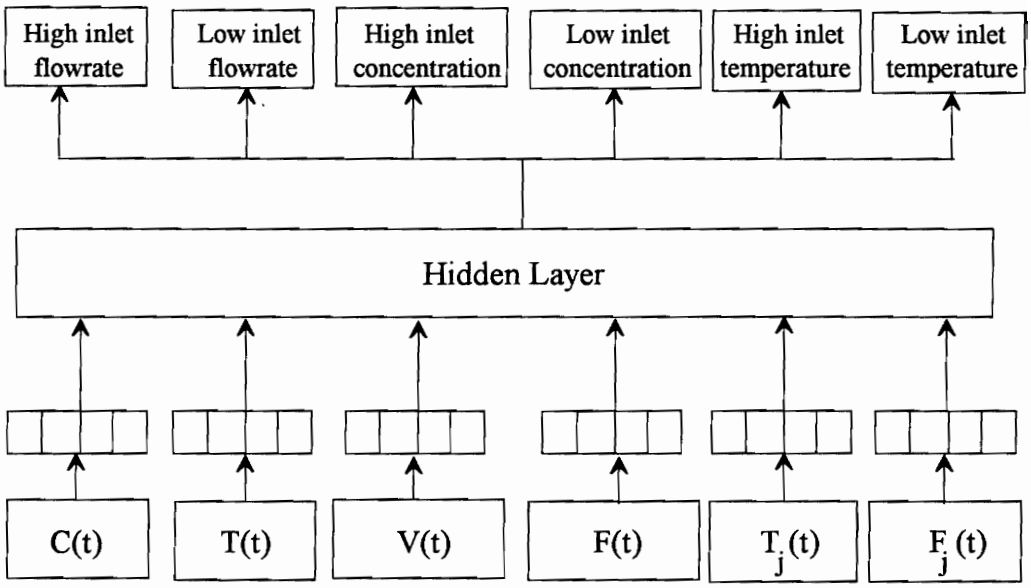


Figure 3.19. Architecture of the unsteady-state CSTR fault-diagnosis network.

Table 3.17. The format of the file *cstrtrn.nna* used for training the unsteady-state CSTR fault-diagnosis network.

Column number	Variable type	Variable	Normalization factor
1	input	C : outlet concentration of component A	0.2 to 0.3 moles/ft^3
2	input	T : outlet temperature	575 to 615 °R
3	input	V : reactor holdup	46 to 50 ft^3
4	input	F : outlet flowrate	28 to 52 ft^3/hr
5	input	T_j : outlet temperature of coolant	580 to 605 °R
6	input	F_j : coolant flowrate	0 to 110 ft^3/hr
7	output	high inlet flow rate	class [1,0,0,0,0,0]
8	output	low inlet flow rate	class [0,1,0,0,0,0]
9	output	high inlet concentration, component A	class [0,0,1,0,0,0]
10	output	low outlet concentration, component A	class [0,0,0,1,0,0]
11	output	high inlet temperature	class [0,0,0,0,1,0]
12	output	low inlet temperature	class [0,0,0,0,0,1]

Table 3.18. The specifications for the unsteady-state CSTR fault-diagnosis network using the radial-basis-function network.

Network type	radial-basis-function network
Training flie name	*cstrtrn.nna*
Transfer function (input layer)	linear
Transfer function (hidden layer)	radial basis
Transfer function (output layer)	tanh
Learning rule (hidden layer)	K-means
Learning rule (output layer)	delta rule
Summation (hidden layer)	Euclidean
Summation (output layer)	sum
Error (hidden layer)	2-NN
Error (output layer)	standard
Network weight distribution	distribution range: [-1.0, 1.0]

Input layer

Training iteration	5,000
Noise	0.0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0.0

Hidden layer

Training iteration	500	1,000	1,500	2,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.3	0.15	0.075	0.0375
Cluster Threshold	0.1	0.05	0.025	0.0

Output layer

Training iteration	2,000	12,000	32,000	72,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.0	0.15	0.075	0.01875
Momentum coefficient	0.0	0.4	0.20	0.05
Error tolerance	0.0	0.1	0.1	0.1

Table 3.19 and Figure 3.20 show the radial-basis-function network's average error for recall of the training data set as the number of nodes in the hidden layer increases from 3 to 30. The average error is a combination of all six process classifications for the network trained with 50,000 iterations. As the data reveal, 15 nodes in the hidden layer is an effective configuration, since additional nodes produce no significant error reduction.

The network also properly classifies malfunctions for each of the 12 process runs used to train the network. As an example, Figure 3.21 shows the typical fault class responses for the recall of the first process run using fault 1 with a 15 % perturbation. As shown, the network predicts the recall of fault 1 accurately, and only one of the other five faults deviates from zero. Note that this minor deviation will *not* result in misclassification of process malfunctions.

Table 3.19. A comparison of the average error for training the unsteady-state CSTR fault-diagnosis network.

Number of nodes in the hidden layer	Average error
3	0.094
5	0.083
10	0.019
15	0.009
20	0.008
30	0.006

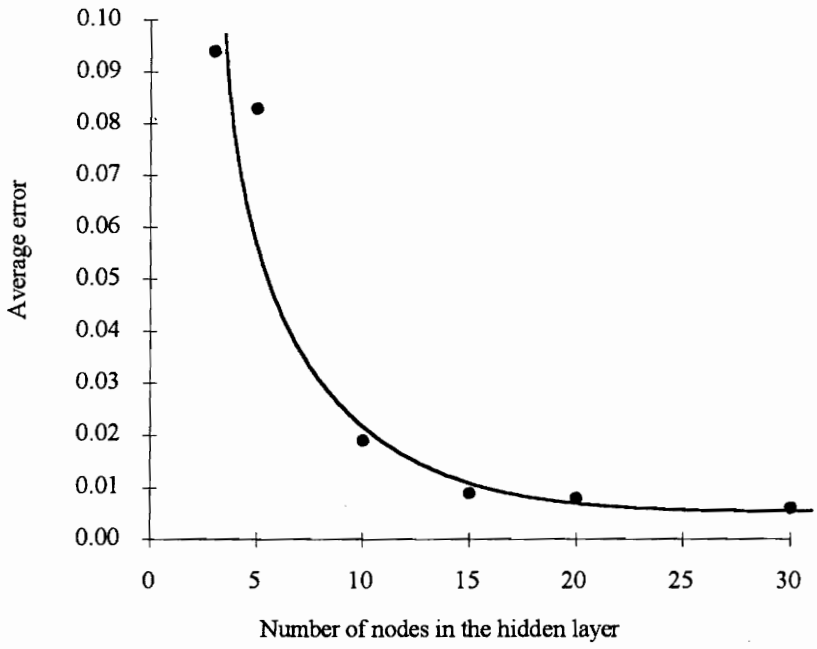


Figure 3.20. The average errors for recall of the training data set for the unsteady-state CSTR fault-diagnosis network with increasing number of nodes in the hidden layer.

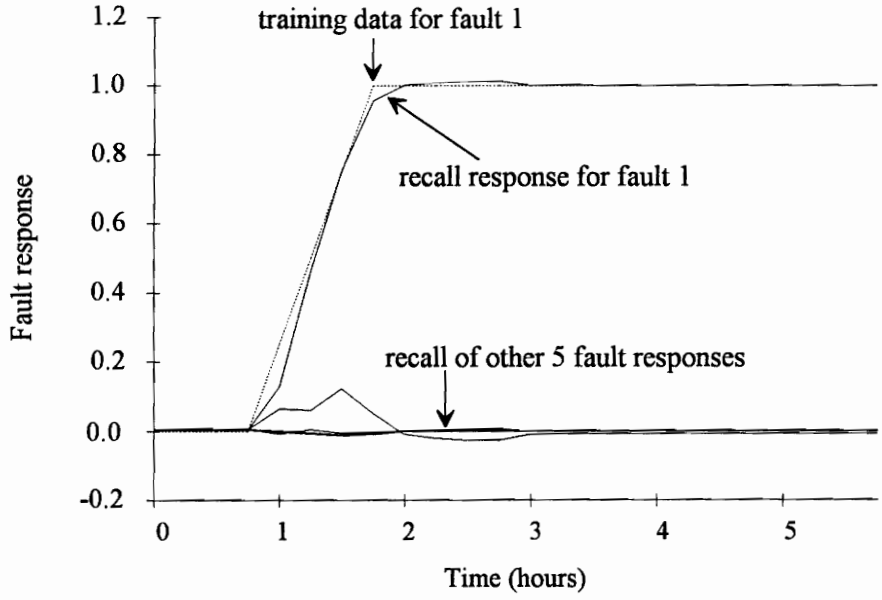


Figure 3.21. A typical fault-class response for the recall of the first process run using fault 1 with a 15 % perturbation as an example.

Vaidynathan and Venkatasubramanian's first set of simulated process runs used to test the network performance for generalization involves perturbation of 10% in inlet flow rate, concentration, and temperature at times of 0.25, 0.5, 1.5, and 5 hours, and perturbations of 25% in inlet flow rate and concentration at those same times. This data set is given in file *cstrtst1.nna*. Table 3.20 shows our network's predictions for this testing. The values for fault responses should approach 1 when their respective faults are induced with either a 10 or 25% perturbation, and should be 0 for all other cases. The network accurately predicts the respective faults for all cases, with only a few external faults also projected (e.g., fault 2 is partially induced, response of 0.33, when a 25% perturbation is added to fault 3).

Vaidynathan and Venkatasubramanian (1990) also perform *concurrent* fault analysis, i.e., identifying two faults occurring simultaneously. This task is much more challenging, and even goes beyond the ability of most human experts. Most fault-diagnosis strategies use a "fault tree" that does not take into account concurrent faults. Vaidynathan and Venkatasubramanian (1990) use the unsteady-state CSTR model to simulate dual faults, adding 10 % perturbation to 6 dual fault cases (y_3y_5, y_4y_5, y_1y_5, y_2y_5, y_1y_6, and y_2y_6). The testing data set is given in file *cstrtst2.nnr*.

For this example, we consider a fault activated if its level is greater than 0.15. The radial-basis-function network performs correctly in four cases, with errors in the other two. The results appear in Table 3.21. For comparison, Table 3.22 shows the results obtained by Vaidynathan and Venkatasubramanian (1990) for the backpropagation

network with 15 nodes in the hidden layer for a time of 1.5 hours. That network classifies only one case properly, compared to four classified by the radial-basis-function network.

This example shows some of the limitations of both networks. First, Table 3.21 demonstrates neural networks tend to *under-report* faults; i.e., they may view actual faults as noise and filter them, thereby failing to correctly report the faults. Second, neither network extrapolates well. For the concurrent fault y_4y_5, the radial-basis-function network produces low activations of both fault 1 and fault 4, in addition to properly classifying fault 5. Fault 4 has a response lower than expected because the interaction of low inlet concentration and high inlet temperature causes the outlet concentration to be much lower than the values used in network training (outlet concentrations of approximately 0.32 during training and 0.08 during testing). Therefore, the testing example lies outside the radial center of the nodes in the hidden layer and will have a lower fault-response value. The radial-basis-function network requires a training set larger than 12 process runs to more accurately represent the possible operating conditions. As shown in the Leonard and Kramer example, Section 3.4.B.3 , the backpropagation network has similar problems with extrapolation and will activate many more faults than the radial-basis-function network, as a comparison of Tables 3.21 and 3.22 shows.

Table 3.20. The fault responses for testing the generalization data set, *cstrtst1.nna*, using the unsteady-state CSTR fault-diagnosis network.

	Time (hrs)	Fault responses					
		fault 1	fault 2	fault 3	fault 4	fault 5	fault 6
fault 1 10%	0.25	0.07	-0.01	0.00	0.05	0.00	0.00
	0.50	0.25	0.00	0.01	0.05	0.00	0.00
	1.50	0.79	0.00	-0.01	0.03	0.00	-0.01
	5.00	0.79	0.00	-0.01	0.02	0.00	-0.01
fault 2 10%	0.25	-0.01	0.08	0.05	0.00	0.00	0.00
	0.50	0.00	0.26	0.05	0.01	0.00	0.00
	1.50	0.00	0.76	0.05	-0.02	-0.01	0.00
	5.00	0.00	0.77	0.04	-0.02	-0.01	0.00
fault 3 10%	0.25	0.00	0.02	0.07	-0.01	0.00	0.00
	0.50	0.00	0.04	0.24	-0.01	0.00	0.00
	1.50	0.00	0.00	0.80	0.01	0.00	0.00
	5.00	0.00	0.01	0.78	0.01	0.00	0.00
fault 4 10%	0.25	0.02	0.00	-0.02	0.07	0.00	0.00
	0.50	0.03	0.00	-0.01	0.25	0.00	0.00
	1.50	0.01	0.00	0.01	0.80	0.00	0.00
	5.00	0.01	0.00	0.01	0.80	0.00	0.00
fault 5 10%	0.25	0.03	-0.01	0.05	-0.03	0.08	-0.01
	0.50	0.04	-0.01	0.04	0.00	0.26	0.01
	1.50	-0.01	-0.03	0.00	0.00	0.83	-0.01
	5.00	-0.02	-0.03	-0.01	0.01	0.77	-0.01
fault 6 10%	0.25	-0.02	0.03	-0.03	0.06	-0.01	0.07
	0.50	-0.03	0.02	-0.02	0.06	0.00	0.28
	1.50	-0.01	-0.01	-0.01	-0.01	-0.01	0.89
	5.00	0.00	-0.01	0.00	-0.02	0.00	0.84
fault 1 25%	0.25	0.31	0.00	0.00	0.08	-0.01	-0.01
	0.50	0.77	-0.01	-0.01	0.18	-0.02	-0.01
	1.50	0.94	0.13	0.04	-0.06	0.13	0.12
	5.00	0.95	0.12	0.04	-0.06	0.13	0.12
fault 2 25%	0.25	0.00	0.31	0.08	0.00	-0.01	0.00
	0.50	-0.01	0.73	0.25	-0.01	-0.02	-0.01
	1.50	0.12	0.95	-0.08	0.07	0.18	0.10
	5.00	0.13	0.95	-0.08	0.07	0.18	0.10
fault 3 25%	0.25	0.00	0.07	0.23	0.00	-0.01	0.00
	0.50	0.00	0.09	0.70	0.00	-0.02	0.00
	1.50	0.05	0.31	0.65	0.04	0.08	0.04
	5.00	0.04	0.33	0.65	0.04	0.07	0.04
fault 4 25%	0.25	0.05	0.00	0.00	0.28	0.00	-0.01
	0.50	0.07	0.00	0.00	0.76	0.00	-0.02
	1.50	0.16	0.09	0.03	0.76	0.09	0.07
	5.00	0.16	0.09	0.03	0.75	0.10	0.08

Table 3.21. The fault responses for the testing of the two-fault generalization data set, *cstrtst2.nna*, using the CSTR fault-diagnosis network.

	Time (hrs)	Fault responses						Faults identified
		fault 1	fault 2	fault 3	fault 4	fault 5	fault 6	
fault 3 : 10% fault 5 : 10%	0.50	-0.01	-0.06	0.39	0.00	0.23	0.00	y3, y5
	1.50	-0.01	0.02	0.19	0.01	0.76	0.00	
	2.50	-0.02	0.01	0.24	0.01	0.69	0.00	
	5.00	-0.02	0.01	0.24	0.01	0.69	0.00	
fault 4 : 10% fault 5 : 10%	0.50	0.12	-0.01	-0.04	-0.02	0.30	0.00	y5
	1.50	0.14	0.01	-0.02	0.07	0.68	0.00	
	2.50	0.14	0.01	-0.02	0.10	0.63	0.00	
	5.00	0.14	0.01	-0.02	0.10	0.63	0.00	
fault 1 : 10% fault 5 : 10%	0.50	0.31	-0.02	-0.03	-0.04	0.29	0.00	y1, y5
	1.50	0.40	0.03	0.01	-0.02	0.67	0.03	
	2.50	0.43	0.02	0.00	-0.02	0.62	0.03	
	5.00	0.43	0.03	0.00	-0.02	0.62	0.03	
fault 2 : 10% fault 5 : 10%	0.50	-0.03	-0.01	0.44	0.01	0.17	-0.01	y2, y3, y5
	1.50	0.00	0.31	0.16	0.02	0.46	0.00	
	2.50	-0.02	0.19	0.48	0.01	0.25	-0.01	
	5.00	0.00	0.36	0.16	0.01	0.40	0.00	
fault 1 : 10% fault 6 : 10%	0.50	-0.01	-0.03	0.01	0.08	-0.02	0.25	y1, y6
	1.50	0.18	0.03	0.02	0.06	0.04	0.69	
	2.50	0.26	0.03	0.02	0.04	0.04	0.62	
	5.00	0.26	0.03	0.02	0.04	0.04	0.62	
fault 2 : 10% fault 6 : 10%	0.50	-0.01	0.39	-0.07	-0.03	0.00	0.20	y2, y6
	1.50	0.03	0.49	-0.04	0.00	0.05	0.51	
	2.50	0.03	0.49	-0.04	0.00	0.04	0.48	
	5.00	0.03	0.50	-0.04	0.00	0.04	0.48	

Table 3.22. The fault responses for the testing of the two-fault generalization data set, reported by Vaidynathan (1991).

	Fault responses						Faults identified
	fault 1	fault 2	fault 3	fault 4	fault 5	fault 6	
fault 3 : 10% fault 5 : 10%	0.0001	0.2005	0.2313	0.0004	0.7684	0.0001	y2, y3, y5
fault 4 : 10% fault 5 : 10%	0.1623	0.0001	0.0000	0.7190	0.6767	0.0002	y1, y4, y5
fault 1 : 10% fault 5 : 10%	0.1529	0.0002	0.0000	0.3942	0.8181	0.0001	y1, y4, y5
fault 2 : 10% fault 5 : 10%	0.0000	0.4468	0.0405	0.0003	0.7862	0.0001	y2, y5
fault 1 : 10% fault 6 : 10%	0.0006	0.0007	0.0022	0.0004	0.0000	0.8157	y6
fault 2 : 10% fault 6 : 10%	0.0000	0.0186	0.0296	0.0000	0.0000	0.8863	y6

3.5 Classification Networks for Feature Categorization

A. Introduction

Neural networks are also effective in categorizing data into identifiable groups or features. Classification neural networks used for feature categorization are very similar to fault-diagnosis networks, except that they only allow one output response for any input

pattern, instead of allowing multiple faults to occur for a given set of operating conditions. The classification network selects the category based on which output response has the highest output value.

Classification neural networks become very powerful when used in a hybrid system with the many types of predictive neural networks. In systems that have several different nonlinear operating regions, the classification neural network can first identify the process operating region, then proceed to a corresponding predictive neural network. This accomplishes two goals: (1) to lower the prediction error by training and testing similar patterned behavior; and (2) to reduce the number of training examples within a training set, making network training more efficient. For example, identifying the cell-growth-phase category for the fermentation process of Section 5.4 allows us to train the fermentation-processing network solely with data from the exponential growth phase. Therefore, we can use a smaller, more efficient network with improved prediction capability, since we do not have to model the induction, stationary, and death phases which do not follow the same growth patterns.

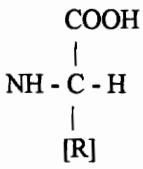
B. Illustrative Case Study : Prediction of Protein Secondary-Structure Categories

This case study illustrates a network to predict a protein's secondary-structure category based on amino-acid compositions and protein-chain length. We then show how to use

this network in conjunction with prediction networks (e.g., secondary-structure prediction or protein partitioning to be discussed in Chapter 6) to improve upon the results of a single prediction network.

1. Protein Structure

Let us start by reviewing some background information on protein structures. Proteins are complex polymers composed of a series of amino acids attached by peptide bonds. There are 20 different naturally occurring amino acids present in proteins, each with a different residual group, [R]:



The properties of the residual groups, in conjunction with their structural positions, define the solution properties of the protein. Amino acids fall into five categories: aliphatic, nonpolar, aromatic, polar and charged. Table 3.23 lists the twenty amino acids in order of ascending hydrophobicity, measured on the basis of solubility in various solvents (Branden and Tooze, 1991).

Table 3.23. The amino-acid characteristics (Brandon and Tooze, 1991).

Amino acid		Free energy (kcal/mole)	Residue category
Phe	Phenylalanine	3.7	aromatic
Met	Methionine	3.4	nonpolar
Ile	Isoleucine	3.1	aliphatic
Leu	Leucine	2.8	aliphatic
Val	Valine	2.6	aliphatic
Cys	Cysteine	2.0	nonpolar
Trp	Tyrptophan	1.9	aromatic
Ala	Alanine	1.6	aliphatic
Thr	Threonine	1.2	polar
Gly	Glycine	1.0	nonpolar
Ser	Serine	0.6	polar
Pro	Proline	- 0.2	nonpolar
Tyr	Tyrosine	- 0.7	aromatic
His	Histidine	- 3.0	aromatic
Gln	Glutamine	- 4.1	polar
Asn	Asparagine	- 4.8	polar
Glu	Glutam	- 8.2	charged (-)
Lys	Lysine	- 8.8	charged (+)
Asp	Aspartic acid	- 9.2	charged (-)
Arg	Arginine	-12.3	charged (+)

Proteins have four structural categories, shown in Figure 3.22. The primary structure is the amino-acid sequence of the polypeptide chain. The secondary structure is the conformation of the backbone (α-helix or β-sheet). The tertiary structure is the three-dimensional conformation, representing how the secondary structure folds to obtain the most favorable thermodynamic state, with hydrophobic residues on the interior and hydrophilic residues on the exterior. The quaternary structure is the arrangement of the aggregation of several polypeptide chains.

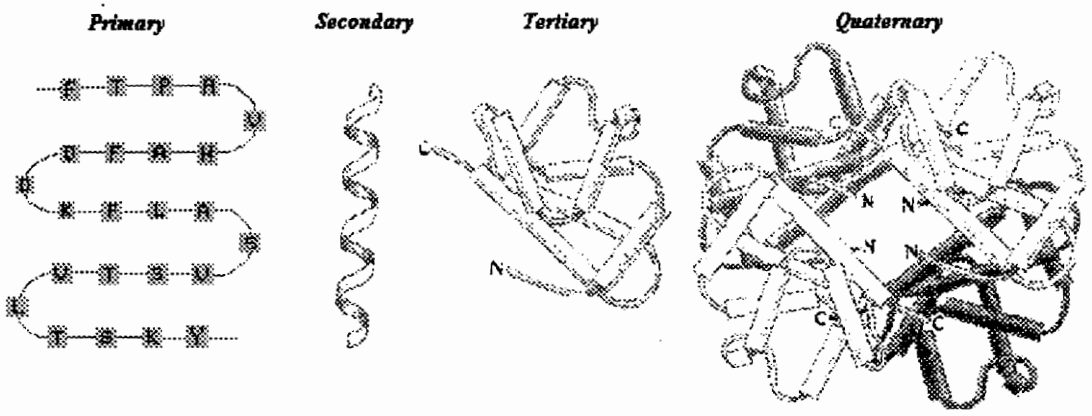


Figure 3.22. The four structural groups of proteins. Reprinted with permission from Branden, C. and Tooze, T., *Introduction to Protein Structure*, copyright 1991, Garland Publishing, Inc., New York.

There are two thermodynamically favorable conformations for polypeptide chains: the α-helix and β-sheet structures (Figures 3.23 and 3.24). The α-helix is a tight coil

with 3.6 amino acids per turn, stabilized by hydrogen bonds between the NH and CO groups. All hydrogen bonds in the α-helix point the same direction, with the NH groups towards the N-terminal side and the CO groups towards C-terminal side. Polarity differences between the CO and NH groups cause an overall dipole moment along the helical axis. This helix structure occurs most often at the protein's surface, where one side of the α-helix is hydrophobic and the other hydrophilic (Figure 3.23).

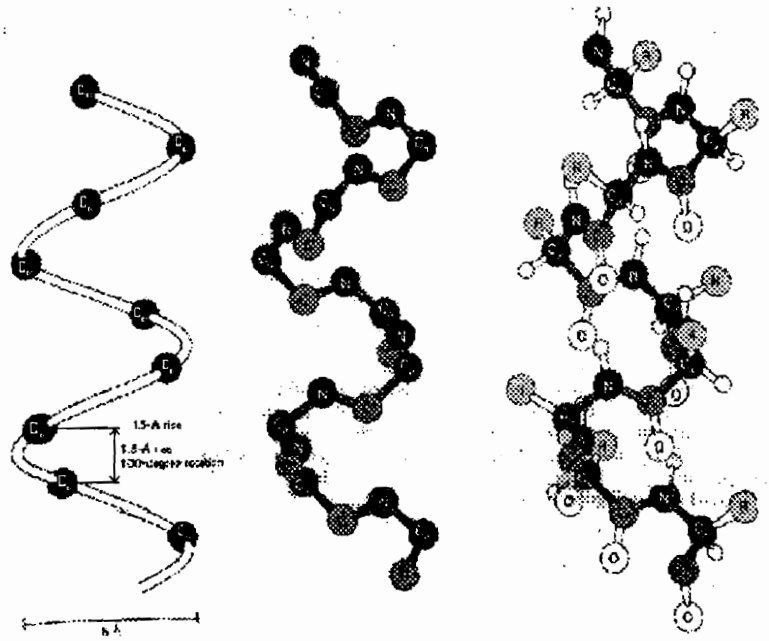


Figure 3.23. The α-helix secondary structures of proteins. Reprinted with permission from Stryer, L., *Biochemistry*, 3rd edition, copyright 1988, W. H. Freeman and Company, New York. Models of a right-handed α-helix: [A] only the α-carbon atoms are shown on a helical thread; [B] only the backbone nitrogen (N), α-carbon (C_α), and carbonyl carbon (C) atoms are shown; [C] entire helix. Hydrogen bonds between NH and CO groups stabilize the helix.

The β-sheet conformation contains amino acid strands, approximately 5 to 10 units in length, aligned side by side (Figure 3.24). The structure exists in either parallel or antiparallel form. In parallel form, all strands run in the same direction, while in the antiparallel form the strands alternate. Unlike the α-helix, the hydrogen bonds between the NH and CO groups alternate directions in both β-sheet forms. Therefore, no hydrophobic or hydrophilic side exists at the protein surface; instead hydrophobic and hydrophilic side chains are staggered.

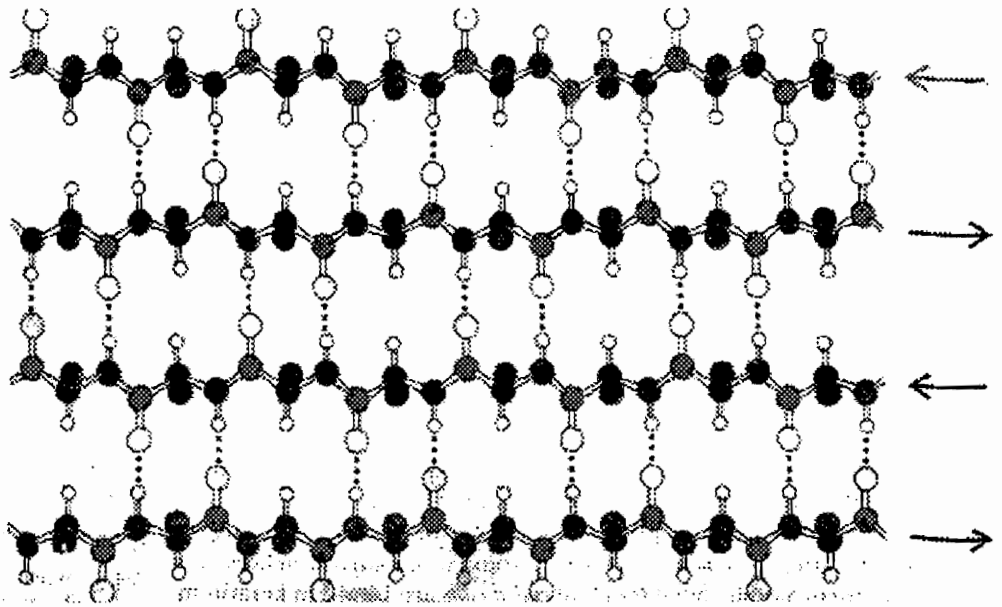


Figure 3.24. The β-sheet secondary structures of proteins. Reprinted with permission from Stryer, L., *Biochemistry*, 3rd edition, copyright 1988, W. H. Freeman and Company, New York. Antiparallel β pleated sheet. Adjacent strands run in opposite directions. Hydrogen bonds between NH and CO groups of adjacent strands stabilize the structure. The side chains are above and below the above the sheet.

2. Network Architecture

In developing the classification network, we use the four structural classes of proteins, as defined by Chou and Fasman (1989), as output categories.

- α proteins : predominantly α-helix regions with little or no β-sheets.
- β proteins : predominantly β-sheets with minimal with minimal or no α-helix regions.
- $\alpha+\beta$ proteins : α-helices and β–sheets clustered in separate domains.
- α/β proteins : alternating α-helices and β–strands.

We also use the database from Chou and Fasman (1989), listed in Table 3.24, which includes 19 α proteins, 15 β proteins, 14 $\alpha+\beta$ proteins, and 16 α/β proteins.

Table 3.24. The proteins used in the database for the training of the protein secondary-structure categorization network, obtained from Chou and Fasman (1989).

α proteins	% α-helix	% β-sheet	β proteins	% α-helix	% β-sheet
Ca binding parvalbumin	56.5	4.6	a - Chymotrypsin	8.3	40.2
Cytochrome b-562	78.6	0.0	Concanavalin	2.1	57.8
Cytochrome c	42.7	5.8	Elastase	7.5	49.2
Cytochrome c-2	43.8	3.6	Erabutoxin B	0.0	50.0
Cytochrome c-550	39.6	4.5	Immunoglobulin Fab (V_H and C_H) (human)	2.3	59.1
Cytochrome c-555	36.1	0.0	Immunoglobulin Fab (V_H and C_L) (human)	2.4	58.7
Hemerythrin (Met-)	64.6	0.0	Immunoglobulin MCG (human)	10.7	65.7
Hemerythrin (Myo-)	68.6	0.0	Immunoglobulin REI (human)	4.6	56.5
Hemerythrin (G. gouldi)	71.7	0.0	Penicillopepsin	8.7	41.8
Hemoglobin, alpha (human)	77.3	0.0	Prealbumin	6.3	45.7
Hemoglobin, beta (human)	76.7	0.0	Protease A	8.3	51.9
Hemoglobin, alpha (horse)	77.3	0.0	Protease B	4.9	56.2
Hemoglobin, beta (horse)	78.8	0.0	Rubredoxide Dismutase	0.0	25.9
Hemoglobin (glycera)	76.2	0.0	Superoxide Dismutase	4.6	50.3
Hemoglobin (lamprey)	79.1	0.0	Trypsin	0.0	41.3
Hemoglobin (midge larva)	83.1	0.0			
Hemoglobin, gamma (human)	77.4	0.0			
Myglobin (seal)	79.1	0.0			
Myglobin (sperm whale)	79.1	0.0			

α+β proteins	% α-helix	% β-sheet	α/β proteins	% α-helix	% β-sheet
Actinidin	28.0	14.2	Adenylate kinase	54.1	12.4
Cytochrome b-5	46.2	28.0	Alcohol dehydrogenase	28.3	30.8
Ferredoxin	24.1	31.5	Carbonic anhydrase B	18.9	27.7
High-protein iron protein	11.8	15.3	Carbonic anhydrase C	20.5	26.6
Insulin	49.0	23.5	Carboxypeptidase A	35.2	14.7
Lysozyme (bacteriopage T4)	65.2	12.2	Carboxypeptidase B	31.4	14.7
Lysozyme (chicken)	41.9	17.1	Dihydrofolate reductase	17.6	30.8
Papain	26.4	14.2	Flavodoxin	36.2	26.8
Phospholipase A-2	49.6	9.8	Glyceraldehyde 3-phospate dehydrogenase (lobster)	32.7	34.5
Ribonuclease S	25.0	44.4	Glyceraldehyde 3-phospate dehydrogenase (B. stearotherm)	31.1	26.7
Staphylococcal nuclease	25.5	28.9	Lactate dehydrogenase	40.4	24.0
Subtilisin inhibitor	17.7	33.6	Phosphoglycerate kinase	40.9	23.8
Thermolysin	35.4	20.6	Rhodanese	41.0	14.3
Trypsin inhibitor	19.0	27.6	Subtilisin BPN	29.1	20.0
			Thiordoxin	48.2	27.8
			Thiose phosphate isomerase	54.0	20.2

Figure 3.25 shows the architecture of the protein secondary-structure categorization network. The network uses 21 inputs to predict the 4 secondary-structure categories: 20 amino-acid compositions plus the number of amino-acid residues in the protein. Tables 3.25 and 3.26 show the format of the file *protcls.nna* and the specifications used for training the network, respectively.

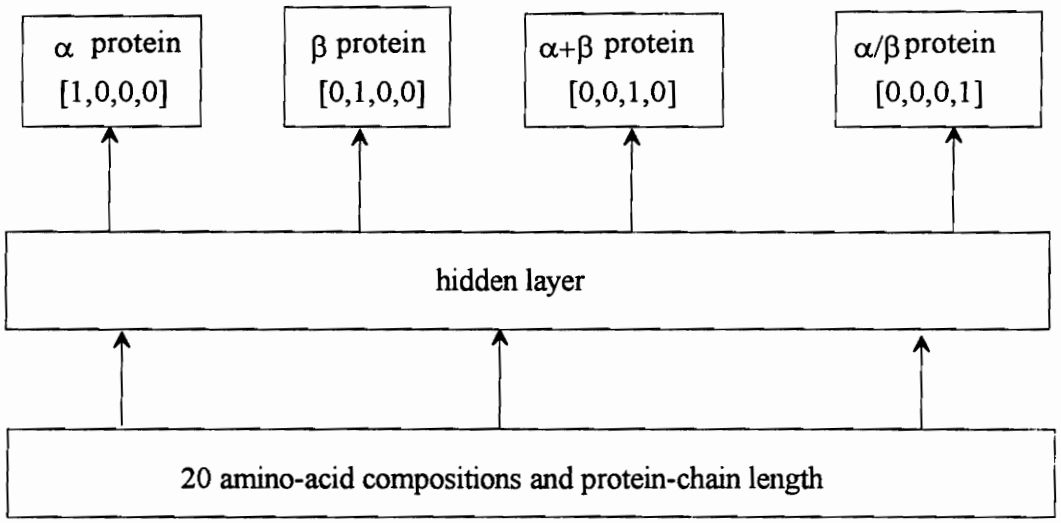


Figure 3.25. Architecture of the protein secondary-structure categorization network.

Table 3.25. The format of the file *protcls.nna* used for training the protein secondary-structure categorization network.

Column number	Variable type	Normalized variable	Normalization factor
1	input	# of amino acids in the protein	1000
2	input	% alanine	100 %
3	input	% arginine	100 %
4	input	% asparagine	100 %
5	input	% aspartic acid	100 %
6	input	% cysteine	100 %
7	input	% glutamine	100 %
8	input	% glutamic acid	100 %
9	input	% glycine	100 %
10	input	% histidine	100 %
11	input	% isoleucine	100 %
12	input	% leucine	100 %
13	input	% lysine	100 %
14	input	% methionine	100 %
15	input	% phenylalanine	100 %
16	input	% proline	100 %
17	input	% serine	100 %
18	input	% threonine	100 %
19	input	% tyrptophan	100 %
20	input	% tyrosine	100 %
21	input	% valine	100 %
22	output	α protein	class [1,0,0,0]
23	output	β protein	class [0,1,0,0]
24	output	$\alpha+\beta$ protein	class [0,0,1,0]
25	output	α/β protein	class [0,0,0,1]

Table 3.26. The specifications for the protein secondary-structure categorization network using the radial-basis-function network.

Network type	radial-basis-function network
Training flie name	*protcls.nna*
Transfer function (input layer)	linear
Transfer function (hidden layer)	radial basis
Transfer function (output layer)	tanh
Learning rule (hidden layer)	K-means
Learning rule (output layer)	delta rule
Summation (hidden layer)	Euclidean
Summation (output layer)	sum
Error (hidden layer)	2-NN
Error (output layer)	standard
Network weight distribution	distribution range: [-1.0, 1.0]

Input layer

Training iteration	5,000
Noise	0.0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0.0

Hidden layer

Training iteration	500	1,000	1,500	2,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.3	0.15	0.075	0.0375
Cluster Threshold	0.1	0.05	0.025	0.0

Output layer

Training iteration	2,000	12,000	32,000	72,000
Noise	0.0	0.0	0.0	0.0
Learning rate	0.0	0.15	0.075	0.01875
Momentum coefficient	0.0	0.4	0.20	0.05
Error tolerance	0.0	0.1	0.1	0.1

Table 3.27 and Figure 3.26 show the network's average error and misclassification rate versus number of nodes in the hidden layer for recall of the training data set. The average error is a combination of all four secondary-structure classifications for the network trained with 50,000 iterations. These results indicate that 30 nodes in the hidden layer are sufficient, since further increases fail to improve network accuracy. This example does not include a generalization data set, because the generation of a learning curve would be misleading since many of the proteins are very similar to one another (e.g., different variations of hemoglobin).

Table 3.27. A comparison of the average error and misclasiification rate for training the protein secondary-structure categorization network.

Number of nodes in the hidden layer	Average error	% Misclassified
5	0.244	23.4%
10	0.179	17.2%
20	0.124	4.7%
30	0.104	1.6%
40	0.093	0.0%

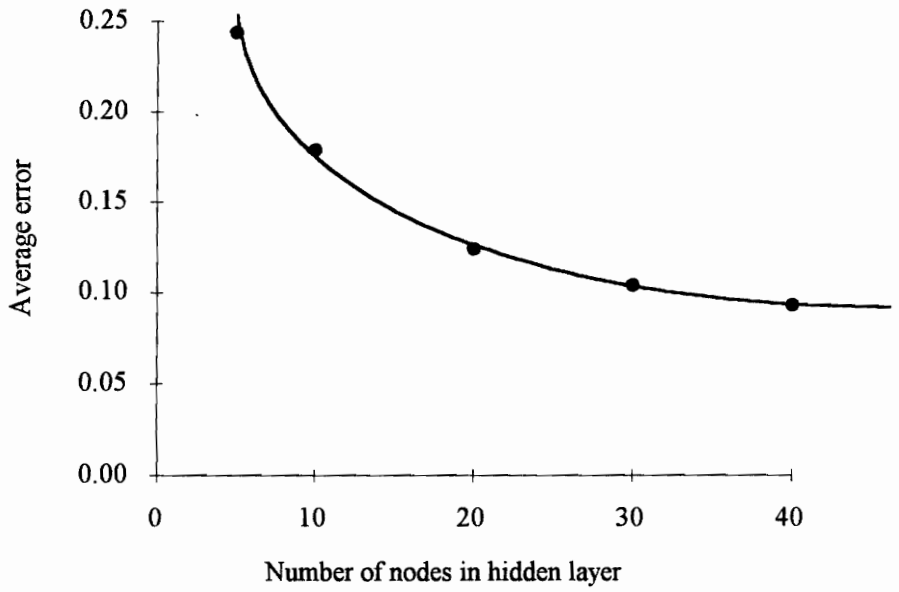


Figure 3.26. The average error for the recall of the training data set using the protein secondary-structure categorization network.

3. Hybrid Networks : Secondary-Structure Categorization and Protein Partitioning

In order to illustrate how to attach a feature categorization network to multiple prediction networks, generating a more powerful hybrid network structure, we link the protein secondary-structure categorization network to a prediction network discussed in Section 6.2.C (protein-partitioning network). Figure 3.27 shows the architecture of a hybrid system made up of the categorization network plus four prediction networks.

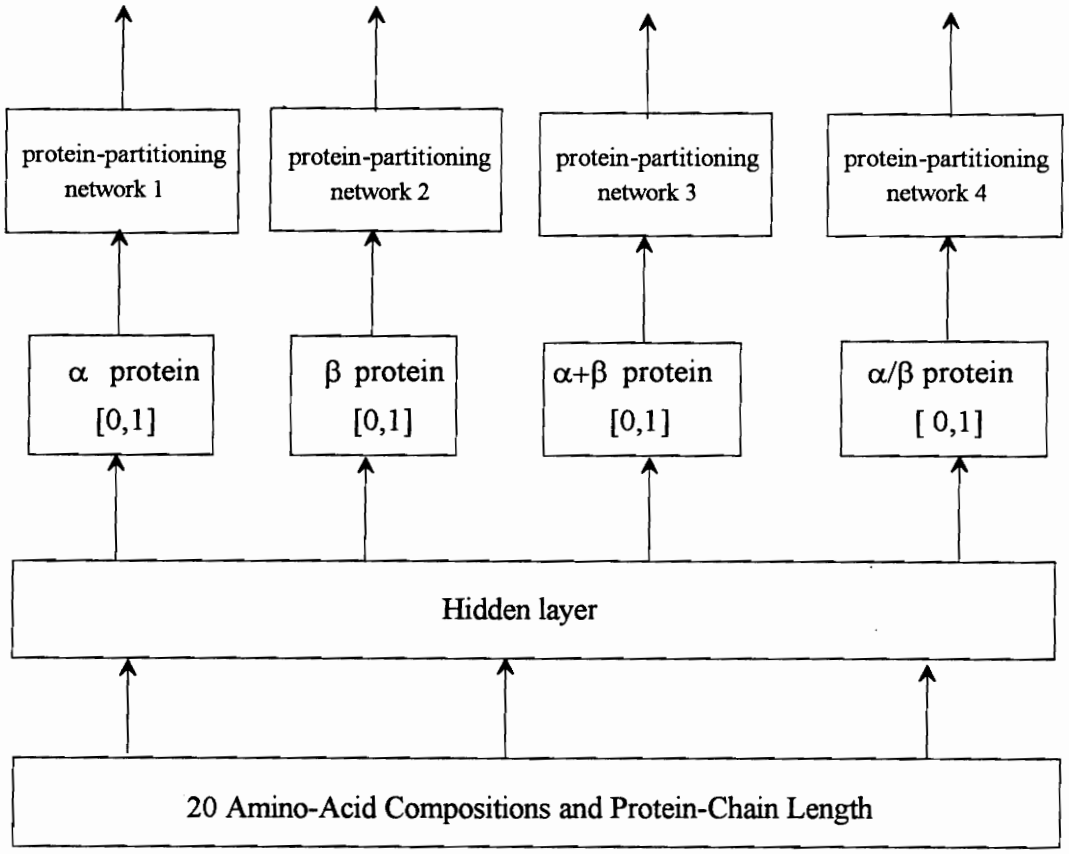


Figure 3.27. A hybrid network which links the protein secondary-structure categorization network to four protein-partitioning prediction networks.

The hybrid system activates only one protein-partitioning network at a time based on the protein secondary-structure class selected by the categorization network. For example, if the categorization network identifies the β-protein category, then the hybrid network uses the protein-partitioning network 2 for the desired prediction. This system enables us to train the four smaller, more efficient protein-partitioning networks

separately, instead of training one very large network, as in Section 6.2.C. This individual training can significantly improve prediction capability, especially when training sets are very large (e.g., thousands of proteins, plus many other independent variables) and when different structural categories produce different response patterns in the prediction network (e.g., α and β proteins have different protein-partitioning behaviors and the hidden layers of the prediction network must therefore store multiple types of input-output response patterns).

3.6 Chapter Summary

- Neural networks have two main applications in classification problems: (1) identification of process faults based on the operating conditions of a given process; and (2) identification of the most likely categorical group for a given input pattern.
- Classification networks produce Boolean output responses, i.e., zero indicates that the input pattern is not within the specific class, and one indicates that it is. The *actual* output from the neural network is a numerical value between 0 and 1, and can represent the "probability" that the input pattern corresponds to a specific class.
- Classification networks used for feature categorization activate only one output response for any input pattern, and select that category based on which output response has the highest value. In comparison, fault-diagnosis networks allow multiple

faults to occur for a given set of operating conditions, and can therefore activate multiple output responses for a given input pattern.

- A classification problem has three major regions: (1) a *decision region,* which corresponds to a unique output class within the input space; (2) a *decision boundary,* which is the intersection of two different decision regions; and (3) a *transition region,* which is the buffer between two different decision regions where we can make only fuzzy inferences about the classification.
- The radial-basis-function network is the most frequently-used network architecture for classification problems. This network consists of three layers: (1) an input layer with a direct transfer function; (2) a hidden layer with a Gaussian transfer function; and (3) an output layer with a sigmoid or hyperbolic tangent transfer function. The weight factors between the input and hidden layers, c_{ik}, are trained using a K-means clustering algorithm; the weight factors between the hidden and output layers, w_{ik}, are trained using the backpropagation algorithm.
- Radial-basis-function networks outperform backpropagation networks for all case studies in this chapter. Comparing the two networks show: (1) backpropagation networks are discriminant classifiers where the decision boundaries tend to be piecewise linear, resulting in non-robust transition regions between classification groups; in comparison, radial-basis-function networks use an explicit similarity-metric classifier to make decisions, leading to a more robust decision boundary; (2) training a radial-basis-function network is typically faster; (3) the internal representation within

the hidden layers of a radial-basis-function network has a more natural interpretation; (4) because the initial learning phase of a radial-basis-function network is unsupervised data-clustering, important discriminatory information could be lost in this phase; and (5) backpropagation networks can give a more compact representation of the problem.

- Neural networks are very effective in fault diagnosis for three reasons: (1) they can store knowledge about the process and learn directly from quantitative, historical fault information; (2) they can filter noise and draw conclusions in the presence of noise; and (3) they can identify causes and classify faults.
- The first illustrative case study, fault diagnosis in a chemical reactor, compares the effect of using different transfer functions (sigmoid and hyperbolic tangent functions) and different learning rules (delta rule and normalized cumulative delta rule) for both the backpropagation and radial-basis-function networks. The best network architecture for this problem uses a radial-basis-function network with a Gaussian transfer function in the hidden layer, a hyperbolic tangent transfer function in the output layer, and the delta learning rule.
- The second illustrative case study, the Leonard and Kramer problem, further demonstrates how the radial-basis-function network outperforms the backpropagation network for classification problems. The radial-basis-function network: (1) has a significantly lower misclassification rate; (2) does not have the extrapolation problems associated with the backpropagation network; and (3) performs much better according to the robust diagnosis tests designed by Leonard and Kramer (1991).

- The third illustrative case study, an unsteady-state CSTR, shows how to develop a moving-window network for modeling time-dependent or unsteady-state fault diagnosis problems.
- The final illustrative case study, identification of protein secondary-structure category, uses a neural network to predict whether the protein is α, β, $\alpha+\beta$, or α/β, based on amino-acid composition and protein-chain length. We show how to develop a hybrid system composed of a classification network and four prediction networks to improve the prediction capability and effectiveness of a neural network.

Nomenclature

A_H	:	heat-transfer area between T and T_j
C	:	outlet concentration of reaction mixture
C_{A0}	:	inlet concentration (component A) of the CSTR
C_A	:	outlet concentration of (component A) of the CSTR
C_p	:	specific heat of the reaction mixture
C_j	:	specific heat of the cooling water
$\mathbf{c}_k$	:	center vector or cluster center for the k^{th} node of the hidden layer
c_{ik}	:	weight factor between input and hidden layer
E_a	:	activation energy

F	:	outlet flow rate of CSTR
F_0	:	inlet flow rate of CSTR
F_j	:	inlet flow rate of cooling water for the CSTR
h_i	:	reactor heat-transfer coefficient
I_k	:	Euclidean summation that is a distance measure in the radial basis function
K	:	controller gain
k	:	first-order reaction-rate constant
L	:	number of nodes in the hidden layer of a radial-basis-function network
M	:	number of nodes in the output layer of a radial-basis-function network
N	:	number of nodes in the input layer of a radial-basis-function network
P	:	number of nearest neighbor centers
p_1	:	1st process parameter in the Leonard and Kramer problem
p_2	:	2st process parameter in the Leonard and Kramer problem
R	:	Euclidean norm $\|\mathbf{x}\|$
R_g	:	ideal gas constant
T	:	outlet temperature of reaction mixture
T	:	outlet temperature of the reaction mixture for the CSTR
T_j	:	internal threshold
T_j	:	outlet temperature of cooling water for the CSTR
T_{j0}	:	inlet temperature of cooling water for the CSTR
t	:	training vector number

U	:	overall heat-transfer coefficient
V	:	reactor volume (holdup) of the CSTR
V_j	:	holdup of the cooling water within the jacket of CSTR
v_k	:	output of the hidden layer
v_1	:	1st noise variable in the Leonard and Kramer problem
v_2	:	2st noise variable in the Leonard and Kramer problem
w_{kj}	:	weight facor between hidden and output layer
$\mathbf{x}$	:	input vector entering a network
x_i	:	input variable entering a network
x_1	:	1st measurement variable in the Leonard and Kramer problem
x_2	:	2st measurement variable in the Leonard and Kramer problem
$x_{i,min}$	:	minimum feasible operating limit of a input variable entering a network
$x_{i,max}$	:	maximum feasible operating limit of a input variable entering a network
$\mathbf{y}$	:	output vector leaving a network
y_j	:	output variable leaving a network
α	:	learning rate
λ	:	heat of reaction
ρ	:	density of the reaction mixture
ρ_j	:	density of the cooling water
σ_k	:	width of gaussian function

Practice Problems

(3.1) Verify the results of the Leonard and Kramer (1991) problem (Section 3.4.B.3) presented in Figures 3.15 and 3.16 for the backpropagation and radial-basis-function networks, respectively. Use the optimal network sizes where the backpropagation network has 6 nodes in the hidden layer and the radial-basis-function network has 15 nodes in the hidden layer.

(a) Train both network using the data file *lktrn.nna* with network specifications listed in Table 3.5 for the backpropagation network and Table 3.7 for the radial-basis-function network.

(b) Create a testing data set which spans the entire operating ranges of x_1 and x_2, -1 to +1 for both. Present the testing data to the network to generate a result file.

(c) Make a graph of the results and compare them to Figures 3.15 and 3.16.

(3.2) Develop both a backpropagation and a radial-basis-function network for fault diagnosis of a fluidized catalytic cracking (FCC) unit (Venkatasubramanian and Chan, 1989). An overview of this problem is discussed in Section 3.4.B.1 on Boolean fault diagnosis. In this example, both the input and output are Boolean.

You will diagnose faults with eighteen symptoms (18 input nodes with values of 0 or 1) and thirteen basic faults (13 output nodes with desired values of 0 or 1). Table 3.P1 lists the 18 process symptoms and 13 faults that correspond to the input and output nodes.

Table 3.P1. Classification of input and output nodes.

Input *(process symptoms)*		Output *(elemental faults)*
Description	Node	Description
Fines content decreased slightly	1	Hole in reactor plenum
Rate of loss increasing	2	Dipleg damage
Fines content decreasing with time	3	Cyclone damage
BS&W analysis shows traces of refractory	4	Damage to regenerator grid
Regenerator grid P has dropped	5	Plugged dipleg, or jammed trickle valve
Sudden high catalyst loss	6	Partial bed defluidization
Regenerator grid P has increased	7	High regenerator velocity
Regenerator has abnormal temperature profiles	8	Catalyst attrition
Losses are high and steady	9	Vanadium poisoning of catalyst
Recent loss of regenerator air	10	Sodium poisoning of catalyst
Fines content increasing with time	11	Nickel poisoning of catalyst
High Vanadium on catalyst	12	Hydrothermal deactivation
High Sodium on catalyst	13	Thermal deactivation
Coke make has increased	14	------
Ratio of H_2 to CH_4 is high	15	------
Catalyst pore size has increased	16	------
Catalyst surface area has increased	17	------
Catalyst pore size remained constant	18	------

Given the input and output nodes, we must relate symptoms with elemental faults. For example, if the H_2-to-CH_4 ratio is high (input node 15 = 1), and the amount of coke has increased (input node 14 = 1), then the likely fault is nickel poisoning of the catalyst (output node 11 = 1). Thus, for the neural network to successfully identify this fault, when input nodes 14 and 15 are one, and all other input nodes are zero, then output node 11 should be one, with all other output nodes equal to zero. The input-output pattern, then, is:

INPUT: (0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0)

OUTPUT: (0 0 0 0 0 0 0 0 0 0 1 0 0)

(a) Train the network with the sets of input-output information shown in Table 3.P2 (file: *fcc.nna*) and with specifications listed in Table 3.5 for the backpropagation and Table 3.7 for the radial-basis-function network. Vary the number of nodes in the hidden layer from 5 to 50 for both training algorithms. Compare the results obtained from the two algorithms.

(c) For each network trained, see how well the network predicts *simultaneous* faults. Input the following vector to the neural network:

INPUT: (1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

Table 3.P2. Boolean training data for FCC fault-diagnosis problem (file: *fcc.nna*).

	Node number																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I:	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O:	1	0	0	0	0	0	0	0	0	0	0	0	0					
I:	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O:	0	1	0	0	0	0	0	0	0	0	0	0	0					
I:	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O:	0	0	1	0	0	0	0	0	0	0	0	0	0					
I:	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
O:	0	1	0	1	0	0	0	0	0	0	0	0	0					
I:	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
O:	0	0	0	0	1	0	0	0	0	0	0	0	0					
I:	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
O:	0	0	0	0	0	1	0	0	0	0	0	0	0					
I:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
O:	0	0	1	0	0	0	1	0	0	0	0	0	0					
I:	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
O:	0	0	0	0	0	0	0	1	0	0	0	0	0					
I:	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
O:	0	0	0	0	0	0	0	0	1	0	0	0	0					
I:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
O:	0	0	0	0	0	0	0	0	0	1	0	0	0					
I:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
O:	0	0	0	0	0	0	0	0	0	0	1	0	0					
I:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
O:	0	0	0	0	0	0	0	0	0	0	0	1	0					
I:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
O:	0	0	0	0	0	0	0	0	0	0	0	0	1					

(3.3) Develop an effective radial-basis-function network (network specifications given in Table 3.7) for fault diagnosis for continuous stirred-tank reactors (CSTRs) in series (Hoskins and Himmelblau, 1990b). An overview of this problem appears in Section 3.4.C on fault diagnosis with continuous variables. Figure 3.P1 shows the three CSTRs in series.

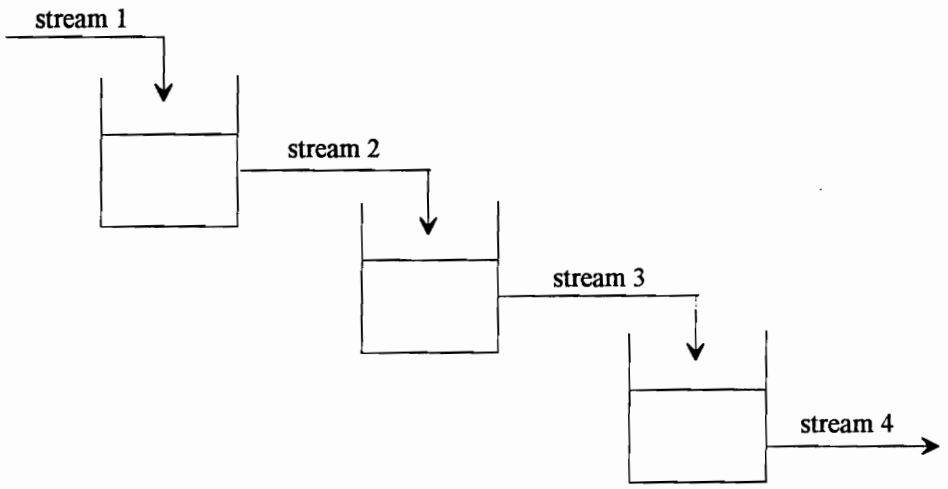


Figure 3.P1. Three continuous stirred-tank reactors (CSTRs) in series.

There are six inputs entering the network to predict six output fault conditions. The input into the network includes flowrate (FR), temperature (T), concentration of A in stream 1 (C_A^1), concentration of B in stream 1 (C_B^1), concentration of A in stream 4 (C_A^4), concentration of B in stream 4 (C_B^4). Table 3.14 lists the six faults, y_1 to y_6. There are eleven examples used for network

training given in file *cstrb.nna*. Table 3.P3 lists both the actual and normalized input variables, along with the associated faults.

Table 3.14. Faults monitored in a CSTR problem (Hoskins and Himmelblau, 1990b).

Fault	Description	Type
y_1	inlet flow rate	high
y_2	inlet flow rate	low
y_3	inlet concentration, component A	high
y_4	inlet concentration, component A	low
y_5	inlet temperature	high
y_6	inlet temperature	low

Table 3.P3. The actual and normalized input variables, along with their fault (file: *cstrb.nna*).

Measurements												Fault
FR ft³/min	T °F	C_A^1 lb-mol/ft³	C_B^1 lb-mol/ft³	C_A^4 lb-mol/ft³	C_B^4 lb-mol/ft³	**FR**	**T**	C_A^1	C_B^1	C_A^4	C_B^4	
18	190	0.3	3.18	0.2275	3.252	0.18	0.19	0.15	0.318	0.2275	0.3252	y_4
18	190	0.6	2.88	0.3755	3.104	0.18	0.19	0.30	0.288	0.3755	0.3104	y_4
18	190	1.3	2.18	0.5990	2.881	0.18	0.19	0.65	0.218	0.5990	0.2881	y_3
18	190	1.6	1.88	0.6681	2.812	0.18	0.19	0.80	0.188	0.6681	0.2812	y_3
13	190	1.0	2.48	0.4475	3.033	0.13	0.19	0.50	0.248	0.4475	0.3033	y_2
15	190	1.0	2.48	0.4777	3.002	0.15	0.19	0.50	0.248	0.4777	0.3002	y_2
22	190	1.0	2.48	0.5600	2.920	0.22	0.19	0.50	0.248	0.5600	0.2920	y_1
26	190	1.0	2.48	0.5958	2.884	0.26	0.19	0.50	0.248	0.5958	0.2884	y_1
18	150	1.0	2.48	0.8960	2.584	0.18	0.15	0.50	0.248	0.8960	0.2584	y_6
18	210	1.0	2.48	0.3102	3.170	0.18	0.21	0.50	0.248	0.3102	0.3170	y_5
18	230	1.0	2.48	0.1703	3.310	0.18	0.23	0.50	0.248	0.1703	0.3310	y_5

(3.4) Generate a learning curve (see Sections 2.1.C.3 and 2.3.F) for the unsteady-state CSTR problem of Section 3.4.D.2, using the radial-basis-function network (network specifications given in Table 3.18) with 15 nodes in the hidden layer.

Randomly select 10 training examples from the data file *cstrtrn.nna* to initially train the unsteady-state CSTR fault-diagnosis network (Figure 3.19). The training examples should come from different areas of the database. Test the network for generalization of the remaining 149 training examples. Continue adding 10 randomly selected, new training examples and testing for generalization until the recall and generalization errors closely approach each other.

(a) Generate a learning curve by plotting the prediction error for both the recall and generalization data groups versus the number of examples in the training data set.

(b) How many training examples are required to effectively train this network for single-fault cases?

(c) As shown in Tables 3.21 and 3.22, the network is not completely trained to predict multiple-fault conditions due to input-variable interactions. How would you design future tests to improve the prediction capability for multiple faults?

(3.5) Compare the effectiveness of the backpropagation network to the radial-basis-function network for the prediction of protein secondary structure (Section 3.5.B).

(a) Train and test the protein secondary-structure categorization network (Figure 3.25) using the backpropagation algorithm having 5, 10, 20, 30, and 40 nodes in the hidden layer. The training data are in file *protcls.nna*.

(b) Compare the results obtained from the backpropagation to those for the radial-basis-function network listed in Table 3.27.

References and Further Reading

Himmelblau (1983) is a good book on the general topic of fault detection and diagnosis in chemical process industries, and Hoskins and Himmelblau (1988, 1990a) report what are apparently the earliest applications of neural networks to fault-diagnosis problems in chemical engineering. A good review of competing technologies for the fault diagnosis in chemical plants is Becraft et al. (1991), and a detailed survey of the latest developments in process monitoring, diagnosis, and control is by Venkatsubramanian and Stanley (1994). For further reading on neural network applications to classification problems in bioprocessing and biotechnology, we recommend the papers by Bulsari et al. (1991),

Gonzalez and Arnaldo (1993), Huang et al. (1993), Iordahe et al. (1993), Long et al. (1991), Moallemi (1991), Monk et al. (1991), Palcic et al. (1992), and Tsai et al. (1993, 1994). For applications to classification problems in nuclear industries, see Kim et al. (1993), and Ohga and Seki (1993). We also suggest the following application papers in chemical engineering: Gupta and Narasimhan (1993), Farrell and Roat (1994), Foss and Johansen (1992), Hsu and Yu (1992), Jokinen (1991), Koshijima and Naidu, Lee and Park (1992), Luo et al. (1992), Naidu et al. (1989), Nie et al. (1994), Soroa and Koivo (1993, 1994), Unger et al. (1990), Vaidynathan and Venkatsubramanian (1992a), Venkatsubramanian et al. (1990), and Watanabe et al. (1989, 1994). Finally, we highly recommend the papers by Moody and Darken (1989) and by Leonard and Kramer (1990, 1991), and the Ph.D. dissertations by Leonard (1991), and by Swenatanakul (1993), which give excellent background on the radial-basis-function networks.

Aldrich, C. and J. J. J. Van Deventer, "Use of Connectionist Systems to Reconcile Inconsistent Process Data," *Chem. Eng. J.*, **54**, 125 (1994).

Bakshi, B. R. and G. Stephanopoulos, "Wave-Let : A Multiresolution, Hierarchical Neural Network with Localized Learning," *AIChE J.*, **39**, 57 (1993).

Bakshi, B. R. and G. Stephanopoulos, "Representation of Process Trends - III. Multiscale Extraction of Trends from Process Data," *Comput. Chem. Eng.*, **18**, 267 (1994a).

Bakshi, B. R. and G. Stephanopoulos, "Representation of Process Trends - IV. Induction of Real-Time Patterns from Operating Data for Diagnosis and Supervisory Control," *Comput. Chem. Eng.*, **18**, 303 (1994b).

Bakshi, B. R. and G. Stephanopoulos, "Reasoning in Time: Modeling, Analysis and Pattern Recognition of Temporal Process Trends," *Advances in Chem. Eng.*, in press (1994c).

Baughman, D. R. and Y. A. Liu, "An Expert Network for Predictive Modeling and Optimal Design of Extractive Bioseparations in Aqueous Two-Phase Systems," *Ind. Eng. Chem. Res.*, **33**, in press (1994).

Branden, C. and T. Tooze, *Introduction to Protein Structure*, Garland Publishing, Inc., New York (1991)

Broomhead, D. S. and D. Lowe, "Radial Basis Functions, Multivariable Function Interpolation and Adaptive Networks," Royal Signals and Radar Establishment Memorandum, No. 4148, London, United Kingdom (1988).

Bulsari, A. B., A. Medeveder and H. Saxen, "Sensor Fault Detection Using State Vector Estimator and Neural Networks Applied to a Biochemical Process," *Acta Polytechnic Scandinavica, Chemical Technology and Metallurgy Series*, No. 199, pp. 1-20 (1991).

Bungay, H. R. and J. A. Clark, "Neural Network for Classifying Flow in a Tank," *Chem. Eng. Comm.*, **125**, 105 (1993).

Chen, S., S. A. Billings, C. F. N. Cowan and P. M. Grant, "Practical Identification of NARMAX Models Using Radial-Basis-Function Networks," *Int. J. Control*, **52**, 1327 (1990).

Chen, S., S. A. Billings, C. F. N. Cowan and P. M. Grant, "Nonlinear Systems Identification Using Radial Basis Function," *Int. J. Systems Sci.*, **21**, 2513 (1991a).

Chen, S., S. A. Billings, C. F. N. Cowan and P. M. Grant, "Orthogonal Least-Squares Learning Algorithm for Radial-Basis-Function Networks," *IEEE Trans. Neural Networks*, **2**, 302 (1991b).

Cherkassky, V. and H. Lari-Najafi, "Data Representation for Diagnostic Neural Networks," *IEEE Expert*, **7**, No. 4, 43 (1992).

Chou, P. Y. and G. D. Fasman, "Prediction of Protein Structural Classes from Amino Acid Compositions," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. D. Fasman, editor, pp. 549-586, Plenium Press, New York (1989)

Cover, T. M. and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Inform. Theory*, **IT13**, 21 (1967).

Fan, J. Y., M. Nikolaou and R. E. White, "An Approach to Fault Diagnosis of Chemical Processes via Neural Networks," *AIChE J.*, **39**, 82 (1993).

Farrell, A. E. and S. D. Roat, "Framework for Enhancing Fault Diagnosis Capabilities of Artificial Neural Networks," *Comput. Chem. Eng.*, **18**, 613 (1994).

Gonzalez, L. P. and C. M. Arnaldo, "Classification of Drug-Induced Behaviors Using a Multilayer Feedforward Neural Network," *Computer Methods and Programs in Biomedicine,* **40**, 167 (1993).

Gupta, G. and S. Narasimhan, "Application of Neural Networks for Gross Error Detection," *Ind. Eng. Chem. Res.*, **32**, 1651 (1993).

Himmbelblau, D. M., *Fault Detection and Diagnosis in Chemical and Petrochemical Industries*, Elsevier Publications, New York (1983).

Himmbelblau, D. M., "Use of Artificial Neural Networks to Monitor Faults and for Troubleshooting in the Process Industries," *IFAC Conference on Safe Processes*, Badena, Germany (1991).

Holcomb, T. and M. Morari, "Local Training for Radial-Basis-Function Networks : Toward Solving the Hidden-Unit Problem," *Proceedings of Amer. Control Conf.*, p. 23331, Chicago, IL (1992).

Hoskins, J. C. and D. M. Himmelblau, "Artificial Neural Networks for Knowledge Representation in Chemical Engineering," *Comp. Chem. Eng.*, **12**, 881 (1988).

Hoskins, J. C. and D. M. Himmelblau, "Fault Detection and Diagnosis Using Artificial Neural Networks," pp. 123-160 in *Artificial Intelligence in Process Engineering*, M. L. Mavrovouniotis, editor, Academic Press, San Diego, CA, (1990a).

Hoskins, J. C. and K. M. Kaliyur, and D. M. Himmelblau, "Fault Diagnosis in Complex Chemical Plants Using Artificial Neural Networks," *AIChE J.*, **37**, 137 (1990b).

Hoskins, J. C. and K. M. Kaliyur, and D. M. Himmelblau, "Fault Diagnosis in Complex Chemical Plants Using Artificial Neural Networks," *AIChE Annual Meeting*, Chicago, IL, November (1990c).

Hsiung, J. T., "Fault Detection via Acoustic Noise," Ph.D. dissertation, chemical engineering, University of Texas, Austin, TX (1992).

Hsu, Y. Y. and C. C. Yu, "A Self-Learning Fault Diagnosis System Based on Reinforcement Learning," *Ind. Eng. Chem. Res.*, **31**, 1937 (1992).

Huang, Y. W., R. Mithani, K. Takahashi, L. T. Fan and P. A. Seib, "Modular Neural Networks for Identification of Starches in Manufacturing Food Products," *Biotechnol. Prog.*, **9**, 401 (1993).

Iordache, O., J. P. Corriou and D. Tondeur, "Neural Network System Classification and Process Fault Detection," *Hungarian J. Ind. Chem.*, **199**, 265 (1991).

Iordache, O., J. P. Corriou, L. Garrido-Sanchez, C. Fonteix and D. Tondeur, "Neural Network Frames: Application to Biochemical Kinetic Diagnosis," *Comput. Chem. Eng.*, **17**, 1101 (1993).

Janusz, M. E. and V. Venkatsubramanian, "Automatic Generation of Qualitative Descriptions of Process Trends for Fault Detection and Diagnosis," *Eng. Applic. Artif. Intell.*, **4**, 329 (1991).

Johnston, L. P. M. and M. A. Kramer, "Probability Density Estimation Using Elliptical Basis Functions," *AIChE J.*, **40**, 1639 (1994).

Kavuri, S. N., "Robust Fault Diagnosis of Process Systems Using Neural Networks with Ellipsoidal Units," Ph.D. dissertation, chemical engineering, Purdue University, West Lafayette, IN (1993).

Kavuri, S. N. and V. Venkatasubramanian, "Improving Fault Classification by Neural Networks Using Ellipsoidal Activation Functions," *AIChE Annual Meeting*, Los Angeles, CA, Nov. (1991).

Kavuri, S. N. and V. Venkatsubramanian, "Solving the Hidden Node Problem in Networks with Ellipsoidal Units and Related Issues," Intern. Joint Conf. on Neural Networks, Baltimore, MD, June (1992a).

Kavuri, S. N. and V. Venkatsubramanian, "Combining Pattern Classification and Assumption-Based Techniques for Process Fault Diagnosis," *Comput. Chem. Eng.*, **16**, 299 (1992b).

Kavuri, S. N. and V. Venkatasubramanian, "Representing Bounded Fault Classes Using Neural Networks with Elliptical Activation Functions," *Comput. Chem. Eng.*, **17**, 139 (1993a).

Kavuri, S. N. and V. Venkatasubramanian, "Using Fuzzy Clustering with Ellipsoidal Units in Neural Networks for Fault Classification," *Comput. Chem. Eng.*, **17**, 763 (1993b).

Kavuri, S. N. and V. Venkatasubramanian, "Neural Network Decomposition Strategies for Large-Scale Fault Diagnosis," *Int. J. Control*, **59**, 767 (1994).

Kim, H. K., S. H. Lee and S. H. Chang, "Neural Network Model for Estimating Departure from Nucleate Boiling Performance of a Pressurized Water-Reactor Core," *Nuclear Technology*, **101**, 111 (1993).

Koulouris, A., B. R. Bakshi and G. Stephanopoulos, "Empirical Learning through Neural Networks: The Wave-Net Solution," *Advances in Chem. Eng.*, in press (1994).

Kramer, M. A. and Leonard, J. A., "Diagnosis Using Backpropagation Neural Networks - Analysis and Criticism," *Comput. Chem. Eng.*, **14**, 1323 (1990).

Leonard, J. A., "A Reliable Neural Network Architecture for Fault Diagnosis and Modeling of Chemical Processes," Ph.D. dissertation, chemical engineering, Massachussetts Institute of Technology, Cambridge, MA (1991).

Leonard, J. A. and M. A. Kramer, "Classifying Process Behavior with Neural Networks: Strategies for Improved Training and Generalization," *Proceedings Amer. Control Conf.*, p. 2478, San Diego, CA, May (1990).

Leonard, J. A. and M. A. Kramer, "Radial Basis Function Networks for Classifying Process Faults," *IEEE Control Systems Magazine*, **11**, 31, April (1991).

Leonard, J. A., M. A. Kramer and L. H. Ungar, "Using Radial Basis Functions to Approximate a Function and Its Error Bounds," *IEEE Trans. on Neural Networks*, **3**, 624 (1992).

Leonard, J. A. and M. A. Kramer, "Diagnosing Dynamic Faults Using Modular Neural Nets," *IEEE Expert*, **8**, No. 2, 44, April (1993).

Long, J. R., H. T. Mayfield, M. V. Henley and P. K. Kromann, "Pattern Recognition of Jet Fuel Chromatographic Data by Artificial Neural Networks with Backpropagation of Error," *Anal. Chem.*, **63**, 1256 (1991).

Luyben, W. L., *Process Modeling Simulation and Control for Chemical Engineers*, 2nd edition, McGraw Hill, New York (1990).

Mah, R. S. H. and V. Chakravarthy, "Pattern Recognition Using Artificial Neural Networks," *Comput. Chem. Eng.*, **16**, 371 (1992).

McAvoy, T. J. and D. Pan, "Application of Olfactory Neural Networks to Chemical Processes," *5th Intern. Symp. Process Systems Eng.*, Kyongju, Korea, May/June (1994).

McDuff, R. J. and P. K. Simpson, "An Adaptive Resonance Diagnostic System," *J. Neural Network Computing*, **2**, 19, Summer (1990).

Moallemi, C., "Classifying Cells for Cancer Diagnosis Using Neural Networks," *IEEE Expert*, **6**, No. 6, 8, December (1991).

Moody, J. and C. J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, **1**, 28 (1989).

Munk, M. E., M. S. Madison and E. W. Robb, "Neural Network Models for Infrared Spectrum Interpretation," *Mikrochim. Acta.*, **II**, 505 (1991).

Naidu, S., E. Zafiriou and T. J. McAvoy, "Application of Neural Networks on the Detection of Sensor Failure During Operation of a Control System," *Proceedings of Amer. Control Conf.*, p. 1336, Pittsburgh, PA, June (1989).

NeuralWare, Inc., *Neural Computations : A Technical Handbook for Professional II/PLUS and NeuralWorks Explorer*, Pittsburg, PA (1993).

Nie, X. R., B. Z. Chen and Y. R. Li, "Study on the Multiple Fault Diagnosis of Complicated Heat Exchanger Networks Using Artificial Neural Networks," *5th Intern. Symp. on Process Systems Eng.*, Kyongju, Korea, May/June (1994).

Ohga, H. and H. Seki, "Abnormal Event Identification in Nuclear Power Plants Using a Neural Network and Knowledge Processing," *Nuclear Technology*, **101**, 159 (1993).

Palcic, B., C. MacAulay, S. Shlien, W. Treurniet, H. Tezcan and G. Anderson, "Comparison of Three Different Methods for Automated Classification of Cervical Cells," *Analy. Cellular Pathology*, **4**, 429 (1992).

Pottmann, M. and D. E. Seborg, "A Nonlinear Predictive Control Strategy Based on Radial-Basis-Function Networks," *Proceedings of IFAC DYCORD+ Conference*, p. 309, College Park, MD (1992).

Powell, M. J. D., "Radial Basis Functions for Multivariable Interpolation: A Review," Technical report, Univ. of Cambridge, London, United Kingdom (1985); *Proceedings of IMA Conference on Algorithms for the Approximation of Functions and Data* (1985).

Preseig, H. A. and D. W. T. Rippin, "Theory and Application of the Modulating Function Method. I. Review and Theory of the Method and Theory of the Spline-Type Modulating Functions; II. Algebraic Representation of Maletinsky's Spline-Type Modulating Functions; III. Application to Industrial Process, A Well-Stirred Tank Reactor," *Comput. Chem. Eng.*, **17**, 1, 17 and 29 (1993a,b)

Raghaven, V. K., J. R. Whitley, J. C. DeVeanx, B. W. Colgate, J. T. Long, and B. C. Mackay, "Clustering Characteristics of Multi-Sensor Trend Patterns," *AIChE National Meeting*, Atlanta, GA, April (1994).

Raju, G. K. and C. L. Cooney, "Using Neural Networks for the Interpretation of Bioprocess Data," pp. 425-428, in *Modeling and Control of Biotechnical Processes*, M. N. Karim and G. Stephanopoulos, Editors, Pergamon Press, Oxford, United Kingdom (1992).

Ramesh, T. S., J. F. Davis and G. M. Schwenzer, "Knowledge-Based Diagnostic Systems for Continuous Process Operations Based Upon the Task Framework," *Comput. Chem. Eng.*, **16**, 109 (1992).

Reilly, D. L., L. N. Cooper and C. Elbaum, "A Neural Model for Category Learning," *Biol. Cybernetics*, **45**, 35 (1982).

Roth, M. W., "Survey of Neural Networks Technology for Automatic Target Recognition," *IEEE Trans. on Neural Networks*, **1**, 28, March (1990).

Sanner, R. M. and J. J. E. Stotine, "Gaussian Networks for Direct Adaptive Control," *IEEE Trans. Neural Networks*, **3**, 837 (1992).

Saraiva, P. M. and G. Stephanopoulous, "Continuous Process Improvement through Inductive and Analogical Learning," *AIChE J.*, **38**, 361 (1992).

Savkoric-Stevanovic, J., "A Neural Network Model for Analysis and Optimization of Processes," *Comput. Chem. Eng.*, **16**, S411 (1992).

Shao, X., "Synthesis of Waste Treatment Systems", PhD dissertation, chemical engineering, University of Cincinnati, OH (1993).

Soroa, T. and H. N. Koirvo, "Neural Networks in Process Fault Diagnosis," *IEEE Trans. on System, Man and Cybernetics*, **SMC-21**, 815 (1991).

Soroa, T. and H. N. Koirvo, "Application of Artificial Neural Networks in Process Fault Diagnosis," *Automatica*, **29**, 843 (1993).

Stephanopoulos, G., "Multi-Scale Description of Real-Time Trends and Their Impact in Pattern Recognition," *Proceedings of Amer. Control Conf.*, p. 2320, Boston, MA, June (1991).

Stork, D. G., "Self-Organization, Pattern Recognition and Adaptive Resonance Networks," *J. Neural Network Computing*, pp. 26-42, Summer (1989).

Stryer, L., *Biochemistry*, 3rd edition, W. H. Freeman and Company, New York (1989).

Suenatanakul, W., "A comparison of Fault Detection and Classification Using Artificial Neural Networks with Traditional Methods," Ph.D. dissertation, chemical engineering, University of Texas, Austin, TX (1993).

Terry, P. A. and D. M. Himmelblau, "Data Rectification and Gross Error Detection in a Steady-State Process via Artificial Neural Networks," *Ind. Eng. Chem. Res.*, **32**, 3020 (1993).

Tsai, D. Y., H. Fujita, K. Horita, T. Endo, C. Kido and S. Sakuma, "Breast Tumor Classification by Neural Networks Fed with Sequential-Dependence Factors to the Input Layer," *IEICE Trans. Inf. & Syst.*, **E-76-D**, 956 (1993).

Tsai, D. Y., H. Fujita, K. Horita, T. Endo, C. Kiod and T. Ishigaki, "Discrimination of Breast Tumors in Mammograms by Means of Artificial Intelligence: A Revised Approach," *Medical Imaging and Information Sciences*, **11**, 57 (1994).

Unger, L. H., B. A. Powell and S. N. Kamens, "Adaptive Neural Nets for Fault Diagnosis and Process Control," *Comput. Chem. Eng.*, **14**, 561 (1990).

Ungar, L. H., T. Johnson and R. D. DeVeaux, "Radial Basis Functions for Process Control," *Proceedings of Computer-Integrated Manufacturing in Process Industries*, New Brunswick, NJ (1994).

Vaidynathan, R., "Process Fault Detection and Diagnosis Using Neural Networks," Ph.D. dissertation, chemical engineering, Purdue University, West Lafayette, IN (1991).

Vaidynathan, R. and V. Venkatasubramanian, "Process Fault Detection and Diagnosis Using Neural Networks - II. Dynamic Processes," *AIChE Annual Meeting*, Chicago, IL, November (1990).

Vaidyanathan, R. and V. Venkatasubramanian, "On the Nature of Fault Space Classification Structure Developed by Neural Networks," *AIChE Annual Meeting*, Los Angeles, CA, November (1991).

Vaidyanathan, R. and V. Venkatasubramanian, "Representing and Diagnosing Dynamic Process Data Using Neural Networks," *Eng. Appl. Artif. Intell.*, **5**, 11 (1992a).

Vaidyanathan, R. and V. Venkatasubramanian, "On the Nature of Fault Space Classification Structure Developed by Neural Networks," *Eng. Appl. Artif. Intell.*, **5**, 289 (1992b).

Venkatasubramanian, V., "CATDEX: An Expert System for Diagnosing a Fluidized Catalytic Cracking Unit," *CACHE Case Studies Series: Knowledge-Based Systems in Process Engineering*, G. Stephanopoulos, editor, **1**, 41, CACHE Corporation, Austin, TX (1988).

Venkatasubramanian, V., "Recall and Generalization Performances of Neural Networks for Process Fault Diagnosis," in *Chemical Process Control IV*, Y. Arkun and W. H. Ray, editors, p. 647, CACHE Corporation, Austin, TX (1991).

Venkatasubramanian, V. and K. Chan, "A Neural Network Methodology for Process Fault Diagnosis," *AIChE J.*, **35**, 1993 (1989).

Venkatasubramanian, V. and S. N. Kavari, "Integrating Unsupervised and Supervised Learning in Neural Networks for Fault Diagnosis," *NATO Advanced Study Institute on Batch Processing Systems Engineering*, Anatalya, Turkey, May-June (1992).

Venkatasubramanian, V. and G. M. Stanley, " Integration of Process Monitoring, Diagnosis and Control : Issues and Emerging Trends," in *Foundations of Computer-Aided Process Operations*, D. W. T. Rippen, J. C. Hale, and J. F. Davis, editors, pp. 179-206, CACHE Corporation, Houston, TX (1994).

Venkatasubramanian, V., R. Vaidynathan, and Y. Yamamoto, "Process Fault Detection and Diagnosis Using Neural Networks - I. Steady-State Processes," *Comput. Chem. Eng.*, **14**, 699 (1990).

Watanabe, K. and D. M. Himmelblau, "Incipient Fault Diagnosis of Nonlinear Processes with Multiple Causes of Faults," *Chem. Eng. Sci.*, **39**, 491 (1983a).

Watanabe, K. and D. M. Himmelblau, "Fault Diagnosis in Nonlinear Chemical Processes: II. Application to a Chemical Reactor," *AIChE J.*, **29**, 250 (1983b).

Watanabe, K., I. Matsuura, M. Abe, M. Kubota, and D. M. Himmelblau, "Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks," *AIChE J.*, **35**, 1803 (1989).

Watanabe, K., S. Hirota, L. Hou and D. M. Himmelblau, "Diagnosis of Multiple Simultaneous Faults via Hierarchical Artificial Neural Networks," *AIChE J.*, **40**, 839 (1994).

Whitely, J. R. and J. F. Davis, "Back-Propagation Neural Networks for Qualitative Interpretation of Process Sensor Data," *AIChE Annual Meeting*, Chicago, IL, November (1990).

Whitely, J. R. and J. F. Davis, "Knowledge-Based Interpretations of Sensor Patterns," *Comput. Chem. Eng.*, **16**, 329 (1992a).

Whitely, J. R. and J. F. Davis, "A Similarity-Based Approach to Interpretation of Sensor Data Using Adaptive Resonance Theory," *Computers and Chem. Eng.*, in press (1992b).

Whiteley, J. R., J. F. Davis, A. Mehrotra and S. C. Ahalt, "Application of Adaptive Resonance Theory for Knowledge-Based Interpretation of Sensor Data," *AIChE Annual Meeting*, Miami, FL, Nov. (1992).

Whiteley, J. R. and J. F. Davis, "Qualitative Interpretation of Sensor Data," *IEEE Expert*, **8**, No. 4, 54 (1993).

Yamamoto, Y. and V. Venkatasubramanian, "Integrated Approach Using Neural Networks for Fault Detection and Diagnosis," *Proceedings of International Joint Conference on Neural Networks*, Vol. I, p. 317 (1990).

Yao, S. C. and E. Zafirious, "Control System Sensor Failure Detection via Networks of Localized Receptive Fields," *Proceedings of Amer. Control Conf.*, p. 2472, San Diego, CA, May (1990).

Ye, N. and N. K. Loh, "Dynamic System Identification Using Recurrent Radial-Basis-Function Network," *Proceedings of Amer. Control Conf.*, p. 2912, San Francisco, CA, June (1993).

Zhang, J. and A. J. Morris, "On-Line Process Fault Diagnosis Using Fuzzy Neural Networks," *Intelligent Systems Engineering*, **3**, 37 (1994).

Chapter 4

Prediction and Optimization

This chapter explains how to apply neural networks to predict the values of process performance variables from independent operating variables based on laboratory or plant data in bioprocessing and chemical engineering. We describe how to combine neural networks with statistical methods to identify the major independent operating variables and their respective ranges to optimize the process performance. In particular, this chapter presents three case studies that illustrate various practical aspects of applying neural networks to prediction and optimization problems in bioprocessing and chemical engineering.

4.1 Introduction

Prediction has been by far the second most popular neural network applications in bioprocessing and chemical engineering (the first being process control). A partial list of various reported applications includes:

- prediction of remote temperature measurements in aluminum manufacturing (Wizzard and Fehrman, 1991)
- estimation of mass-transfer coefficients in electrochemical refining of metals (Reiner et al., 1993)

- estimation of acid strength of mixed oxides in designing multicomponent catalysts (Kito et al., 1992)
- prediction of tray temperatures in distillation columns (Schnelle et al., 1990; Chenng et al., 1992; Ponton and Klemes, 1993)
- prediction of the silicon contents in the pig iron from blast furnace data (Bulsari and Saxen, 1991)
- estimation of isothermal transformation curves in steel processing (Depeyre et al., 1992)
- prediction of performance data in electronic materials processing, including integrated circuits (May, 1994), metal-organic chemical vapor deposition (MOCVD) (Cherian et al., 1991), and plasma-etching process (Himmel and May, 1993; Huang et al., 1994)
- prediction of secondary structures of globular proteins (Qian and Seijnowski, 1988)
- estimation of fermentation performance variables (Oishi et al., 1992; Hofland et al., 1992)
- prediction of complex kinetics in metallurgical and mineral processing (Reuter et al., 1993)
- prediction of performance of a drying system (Huang and Mujumdar, 1993)
- estimation of the performance of the carbon-in-leach gold-recovery process (Van der Walt et al., 1993
- prediction of performance data from analytical chemistry instrumentation such as infrared spectrum interpretation (Robb and Munk, 1990), spectroscopic calibration and quantification (Long et al., 1990; Wythoff et al., 1990; McAvoy et al., 1991),

chromatographic data characterization (Long et al., 1991; Peterson, 1992), identification of nuclear magnetic resonance (NMR) chemical shifts (Meyers et al., 1991; Anker and Jurs, 1992), and enzymatic glucose determination by flow-injection analysis (Campmajó et al., 1992)

- prediction of quantitative structure-activity relationships (OSAR) of pharmaceuticals (Liu et al., 1992a,b)
- prediction of product formulations in material processing (Gill and Schutt, 1992);
- prediction of autoclave performance data in composite manufacturing (Wu, 1990; Wu and Joseph, 1990; Joseph et al., 1992; Shiek, 1992; Joseph and Wang, 1993)

Neural networks have been very effective in predicting and optimizing performance data from processes and analytical instrumentation that are complex and ill-defined by first principles. We expect to see more applications of neural networks to prediction and optimization problems in analytical chemistry, metallurgical and mineral processing, and composite and electronic materials processing, in addition to those already in place in bioprocessing and chemical engineering.

The following sections present three case studies to illustrate a number of practical aspects of neural network applications to prediction and optimization problems.

4.2 Case Study 1: Neural Networks and Nonlinear Regression Analysis

In this case study, we demonstrate how to apply neural networks to prediction problems that have previously been solved using nonlinear regression analysis. Nonlinear regression analysis is a well-established method for systematically fitting experimental or plant data into empirical mathematical models, typically used for problems that are not well-defined by first principles. It is beyond the scope of this text to give a detailed overview of this topic, and we refer the reader to Bates and Watts (1988) for further information.

We use a problem presented by Bates and Watts (1988), in their text on nonlinear regression analysis, to fit the reaction-rate data for the catalytic isomerization of n-pentane to isopentane reported by Carr (1960). Since we are focusing on neural network applications, we only present the nonlinear regression model and results obtained by Bates and Watts (1988) for comparative purposes.

Table 4.1 shows the reaction-rate data for the isomerization of n-pentane (Bates and Watts, 1988) used to fit the nonlinear regression model and to train the neural network. This database contains 24 reaction rates measured at various partial pressures of hydrogen, n-pentane, and isopentane. We divide the data into 6 groups numbered 1 to 6, to be used in developing a neural network model.

Table 4.1. Reaction rate for isomerization of n-pentane to isopentane.

	Partial pressure (psia)			Reaction rate (hr^{-1})
Group	Hydrogen	n-Pentane	Isopentane	
1	205.8	90.9	37.1	3.541
	404.8	92.9	36.3	2.397
	209.7	174.9	49.4	6.694
	401.6	187.2	44.9	4.722
2	224.9	92.7	116.3	0.593
	402.6	102.2	128.9	0.268
	212.7	186.9	134.4	2.797
	406.2	192.6	134.9	2.451
3	133.3	140.8	87.6	3.196
	470.9	144.2	86.9	2.021
	300.0	68.3	81.7	0.896
	301.6	214.6	101.7	5.084
4	297.3	142.2	10.5	5.686
	314.0	146.7	157.1	1.193
	305.7	142.0	86.0	2.648
	300.1	143.7	90.2	3.303
5	305.4	141.1	87.4	3.054
	305.2	141.5	87.0	3.302
	300.1	83.0	66.4	1.271
	106.6	209.6	33.0	11.648
6	417.2	83.9	32.9	2.002
	251.0	294.4	41.5	9.604
	250.3	148.0	14.7	7.754
	145.1	291.0	50.2	11.590

A. A Nonlinear Regression Model for Predicting Reaction Rate Data for an Isomerization Reaction

Bates and Watts (1988) use the following nonlinear regression model to fit the reaction-rate data for the catalytic isomerization of n-pentane to isopentane.

$$f(x,\theta) = \frac{\theta_1\theta_2(x_2 - x_3/1.63)}{1 + \theta_2 x_1 + \theta_3 x_2 + \theta_4 x_3} \qquad (4.1)$$

where

x_1 : partial pressure of hydrogen, psia

x_2 : partial pressure of n-pentane, psia

x_3 : partial pressure of isopentane, psia

θ_i : four empirical parameters to be fitted (j = 1 to 4)

$f(x,\theta)$: reaction rate, hr^{-1}

Table 4.2 gives the parameter estimates and summary statistics for θ_1 to θ_4 based on nonlinear regression analysis (Bates and Watts, 1988). In Table 4.2, the standard error for the parameter estimates, $\bar{\theta}_j$ (j = 1 to 4), reflects the magnitude of the sum of the squared errors between the experimental reaction rates and the predicted reaction rates based on parameter estimates. The correlation-coefficient matrix shows the correlations between parameter estimates. A correlation coefficient of +1 indicates a perfect positive

correlation between two parameters, while a correlation coefficient of -1 reflects a perfect negative correlation between two parameters.

Table 4.2. Parameter estimates and summary statistics for the nonlinear regression model (4.1) (Bates and Watts, 1988).

Parameter θ_j	Estimate $\bar{\theta}_j$	Standard error	Approximate correlation-coefficient matrix			
			θ_1	θ_2	θ_3	θ_4
θ_1	35.92	8.21	1.000	-------	-------	-------
θ_2	0.0708	0.1783	-0.805	1.000	-------	-------
θ_3	0.0377	0.0998	-0.840	0.998	1.000	-------
θ_4	0.0167	0.4150	-0.790	0.998	0.995	1.000

These summary statistics suggest some potential difficulties in applying the regression model, (4.1), since most of the correlation coefficients approach +1 or -1, and some of the standard errors are relatively high (e.g., 8.21 for the parameter estimate $\bar{\theta}_1$). Bates and Watts (1988) illustrate these difficulties using pairwise plots of the 95% inference (confidence) region for parameter estimates, as shown in Figure 4.1. Both the joint 95% inference regions (solid curves) and the individual 95% inference intervals (dashed lines) clearly extend into negative parameter regions, which is questionable because parameters must be positive to be physically meaningful.

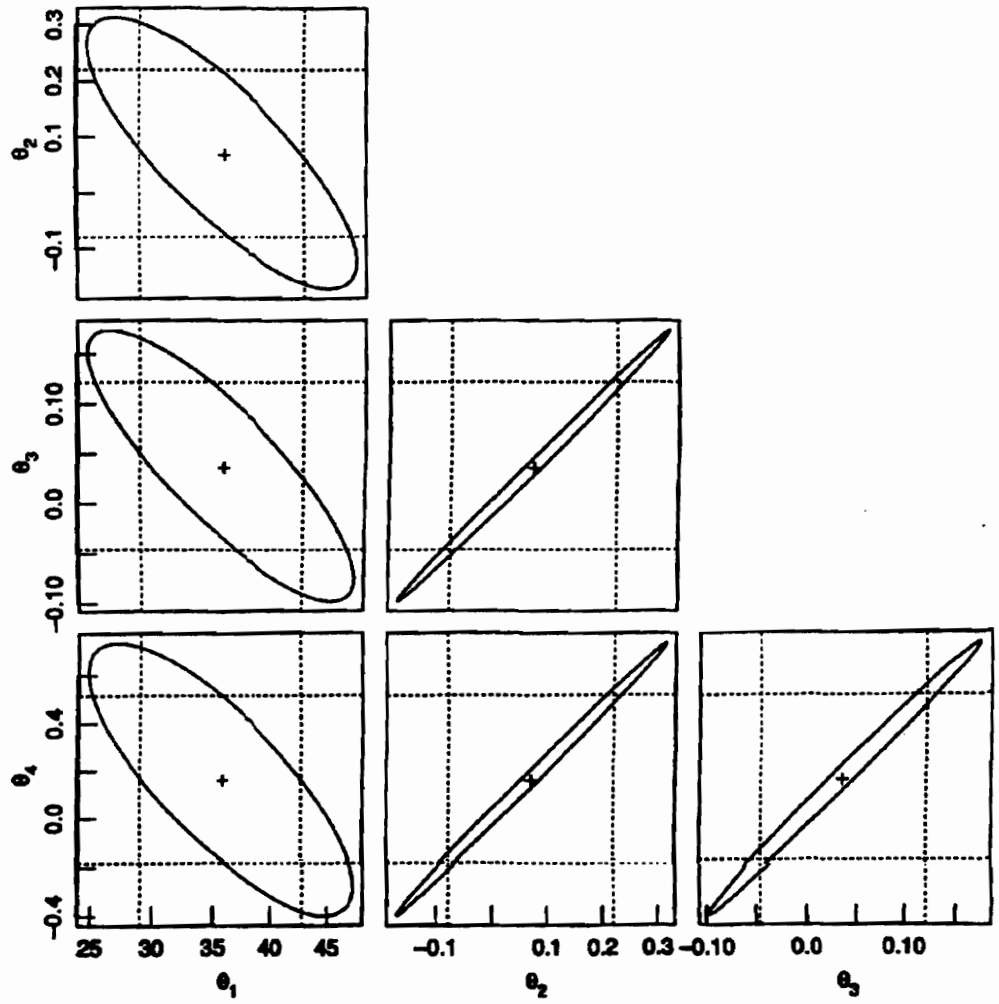


Figure 4.1. Pairwise plots of the 95% inference (confidence) region for parameter estimates for each pair of parameters. The plots show the least squares estimates (+), the joint 95% inference region (solid curve), and the individual 95% inference intervals (dotted lines).

B. A Neural Network Model for Predicting Reaction Rate Data for an Isomerization Reaction

We wish to demonstrate how to use a neural network as an effective alternative to nonlinear regression for predicting the isomerization reaction-rate data (Table 4.1). Our first practical suggestion *in applying neural networks to prediction problems with data sets of modest sizes, such as Table 4.1, is to use a backpropagation network with two hidden layers.* As the three examples in this chapter show, multiple-hidden-layer networks normally outperform the single-hidden-layer networks typically used in past work.

Consider the prediction network for the isomerization reaction shown in Figure 4.2. This network consists of 3 input variables, (the partial pressures of hydrogen, n-pentane, and isopentane), and 1 output variable - reaction rate.

Table 4.3 shows the format of the data file *isom.nna*, corresponding to the experimental data of Table 4.1, used for training the isomerization network. We follow a standard normalization procedure, dividing partial pressures and reaction rate by the normalization factors listed.

Table 4.4 summarizes the network specifications, following the standard format described in Section 2.3.

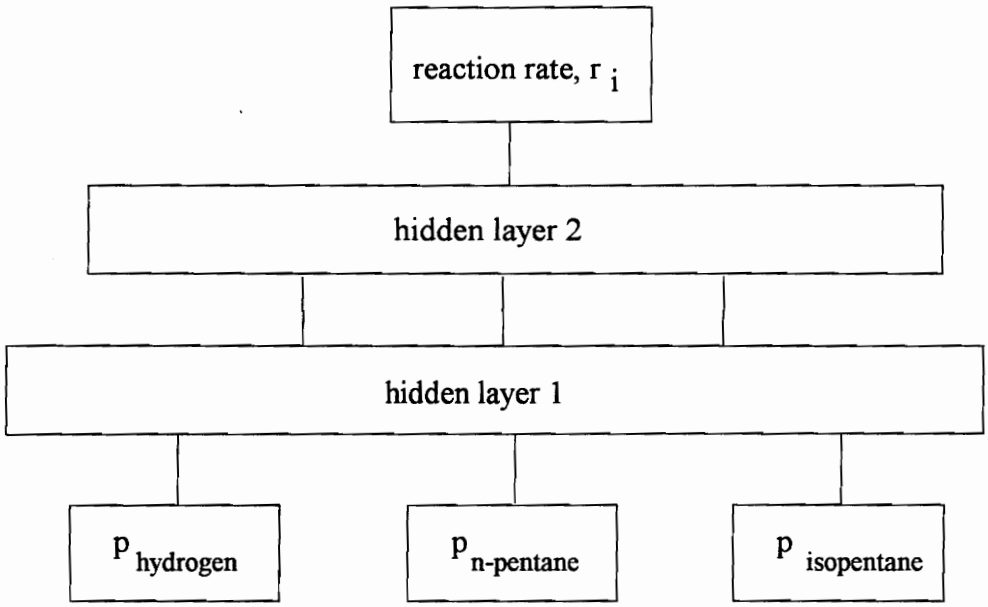


Figure 4.2. Architecture for the prediction network for the isomerization reaction.

Table 4.3. The format of the files for the isomerization network: *isom.nna*.

Column number	Variable type	Normalized variable	Normalization factor
1	input	partial pressure : hydrogen (psia)	500
2	input	partial pressure : n-pentane (psia)	300
3	input	partial pressure : isopentane (psia)	200
4	output	reaction rate (hr^{-1})	15

Table 4.4. The specifications of the prediction network for the isomerization reaction.

Network type	backpropagation
Training file name	*isom.nna*
Transfer function (input layer)	linear
Transfer function (hidden layers)	tanh
Transfer function (output layer)	tanh
Learning rule	delta rule
Summation	sum
Error	standard
Network weight distribution	normal distribution : 3 σ limits of [-1, 1]

input Layer	
Training iteration	5,000
Noise	0
Learning rate	0.9
Momentum coefficient	0.6
Error tolerance	0

hidden layer 1			
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.3	0.15	0.04
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

hidden layer 2			
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.25	0.13	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

output layer			
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.15	0.08	0.02
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

To develop our prediction network, we first determine the optimal network configuration for recalling training examples by training several 1- and 2-hidden-layer networks with different numbers of nodes in the hidden layer (s). We then select three possible configurations to generate learning curves for comparison.

Table 4.5 and Figure 4.3 show the average errors for recall of reaction rates predicted by the isomerization network. We use all 24 data sets to train these configurations with 100,000 iterations. The results show that the 2-hidden-layer networks perform significantly better than the 1-hidden-layer networks. Moreover, in single-hidden-layer networks, 10 nodes are optimal, while in 2-hidden-layer networks, the optimal configuration is 30 nodes in the first layer and 15 in the second.

Table 4.5. The average errors for reaction rates predicted by the isomerization network.

Number of nodes in specified layer		Average error reaction rate (hr^{-1})
hidden layer 1	hidden layer 2	
3	0	0.205
5	0	0.165
10	0	0.128
20	0	0.134
30	0	0.157
6	3	0.126
10	5	0.109
20	10	0.089
30	15	0.074

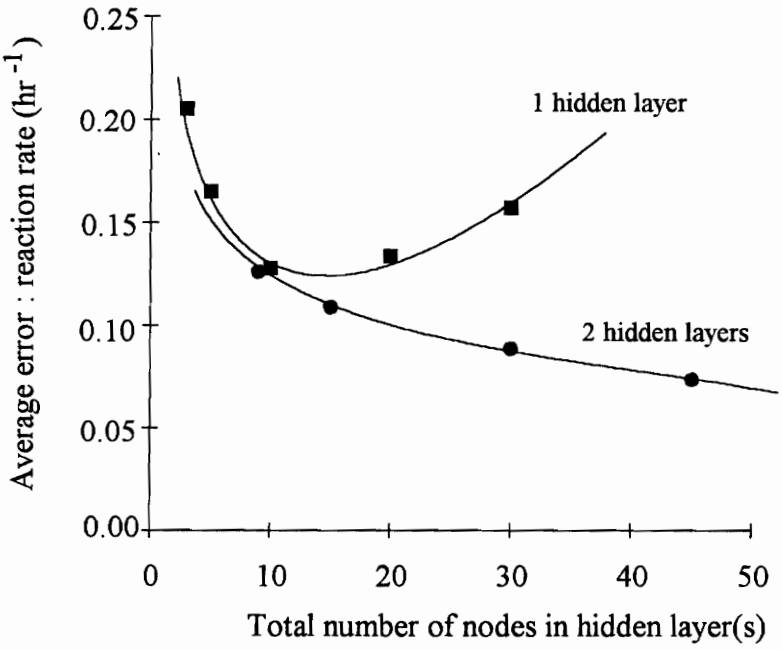


Figure 4.3. A comparison of average errors for the recall of the isomerization networks trained for various 1- and 2- hidden-layer configurations.

To identify the better of those two optimal configurations, we then compare their generalization performances. We also include a smaller 2-hidden-layer configuration of 10:5 nodes in the comparison. Table 4.6 and Figure 4.4 show the average errors for recall and generalization in the learning curves for the three network configurations. Note that "group numbers in the training set" in Table 4.6 refer to the data grouping in Table 4.1. The learning curves for the three networks are very similar, suggesting that network size has a minimal effect on generalization performance for this study. The average error of predicted reaction rates for the generalization data has a minimum of 0.5 to 0.6 hr^{-1}, which is still very large compared to the average error for recall sets, 0.07 to 0.12 hr^{-1}. This result demonstrates a limitation of neural networks: they require sufficiently large data sets

to adequately predict cases outside the original training examples. This example would probably require twice as many training examples within the given input ranges to obtain a desirable prediction accuracy.

Table 4.6. Average errors for the recall and generalization of the isomerization network used in the generation of the learning curve for three network configurations, 0:10, 10:5, and 30:15.

Hidden-layer configuration - 30:15

Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate (hr^{-1})	Avg error reaction rate (hr^{-1})
1	4	20	< 0.001	2.002
1, 6	8	16	0.022	0.887
1, 3, 6	12	12	0.015	0.763
1, 3, 5, 6	16	8	0.059	0.498
1, 2, 3, 5, 6	20	4	0.073	0.555

Hidden-layer configuration - 10:5

Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate (hr^{-1})	Avg error reaction rate (hr^{-1})
1	4	20	< 0.001	2.100
1, 6	8	16	0.027	0.720
1, 3, 6	12	12	0.029	0.699
1, 3, 5, 6	16	8	0.069	0.564
1, 2, 3, 5, 6	20	4	0.111	0.509

Hidden-layer configuration - 0:10

Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate (hr^{-1})	Avg error reaction rate (hr^{-1})
1	4	20	< 0.001	2.613
1, 6	8	16	0.013	0.912
1, 3, 6	12	12	0.050	0.644
1, 3, 5, 6	16	8	0.102	0.552
1, 2, 3, 5, 6	20	4	0.109	0.562

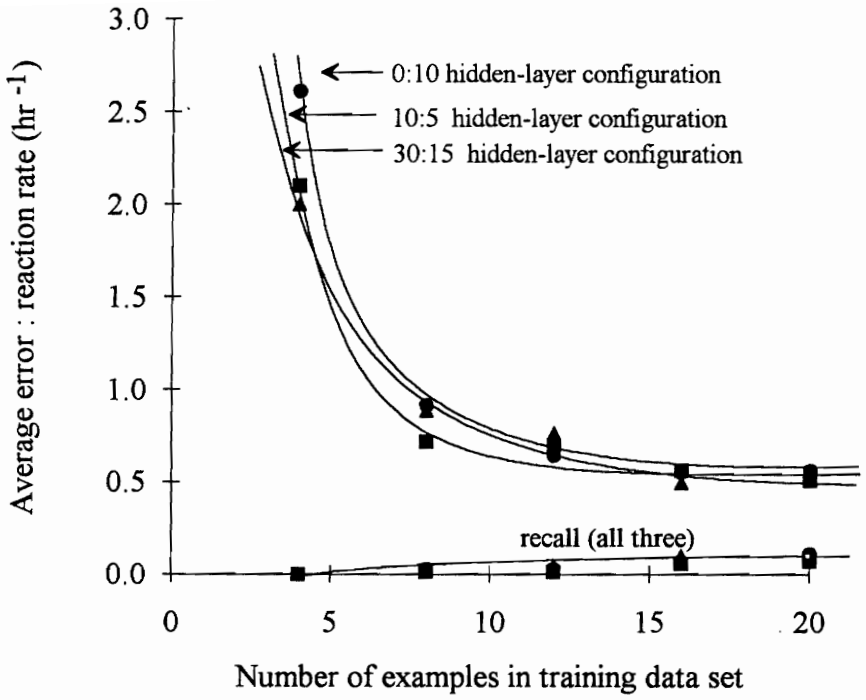
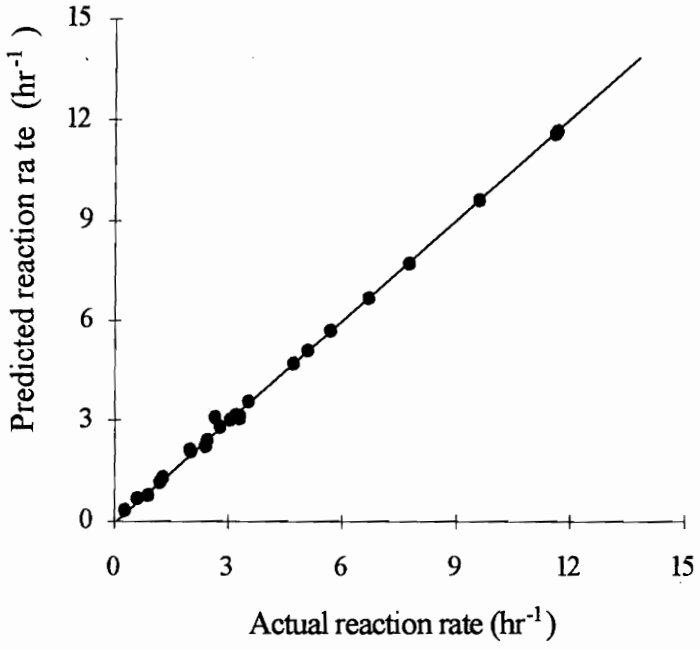


Figure 4.4. Learning curves for the isomerization network structured in 0:10, 10:5, and 30:15 hidden-layer configurations.

Despite this limitation, though, our neural network model still outperforms the nonlinear regression model, (4.1), as the comparison of predicted and experimental reaction rates in Figures 4.5a-b illustrates. In addition, the neural network model has a much lower average error of 0.073 hr^{-1}, compared to 0.298 hr^{-1} for the nonlinear regression model. We can thus conclude that neural networks provide effective alternatives to nonlinear regression in predicting the kinetics of complex chemical and biochemical reactions.

(a)



(b)

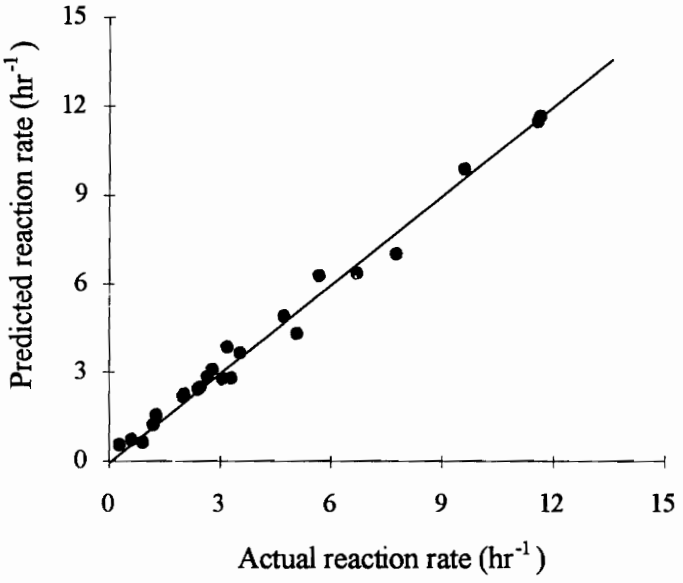


Figure 4.5. The comparison of scatter plots for (a) 30:15 neural network and (b) nonlinear regression model.

4.3 Case Study 2: Neural Networks as Soft Sensors for Bioprocessing

A. Neural Networks and Sensor Technology

Increasing productivity and improving product quality in many chemical engineering and biochemical industries presents a growing need for developing intelligent process sensor technology. In 1988, sales for all sensor industries worldwide totaled $13 billion, and they have increased 25% to 30% annually since then (Ulbrecht, 1989). In particular, there has been a high demand for integrating process sensors with intelligent software and microprocessors in order to monitor a wide range of physical, chemical, and biochemical process variables. Typical examples include pH sensors, humidity sensors, biomass sensors, carbon dioxide sensors, and fiber-optic temperature sensors. Neural networks can play a significant role in developing these new "intelligent" process sensors, serving as "software sensors" ("soft sensors") to facilitate reliable and accurate interpretations of sensor signals (McAvoy et al., 1992).

In what follows, we provide some background on spectroscopic sensors for bioprocessing, then illustrate how to use neural networks to interpret signals from a promising spectroscopic technique, namely, light-induced fluorescence, to estimate the compositions of biochemical species.

B. Spectroscopic Sensors for Bioprocessing

Phillips (1989) offers an excellent survey of the use of spectroscopic sensors in bioprocessing, and this discussion follows her introduction to the subject.

Spectrochemical analysis is based on the interactions of chemical species, including individual bonding structures, with electromagnetic radiation. The radiation on the sample of interest causes a transition between quantum energy states. The magnitude of this transition depends on frequency (or energy content) of the radiation. It is unique for each chemical species as a result of differing energy states.

Spectrochemical analysis provides two types of information:

(1) The *shape* of the spectrum qualitatively identifies the chemical species and can be used to deduce intra- and inter-molecular interactions and configurations;
(2) The *intensity* of the spectrum at one or more wavelengths quantitatively determines the concentration of the chemical species.

Table 4.7 lists the wavelength, frequency, and energy level for a wide range of electromagnetic radiation that form the basis of various spectroscopic and analytical instrumentation. We present this table to suggest that neural networks can play a significant role in facilitating both the qualitative identification and quantitative characterization of a variety of analytical and spectroscopic signals in process

measurements. See, for example, the reports by Anker et al. (1992), Long et al. (1990), McAvoy et al. (1992), Meyers et al. (1991), Robb and Munk (1990), and Wythoff et al. (1990).

Table 4.7. Frequency and energy characteristics of the electromagnetic spectrum.

Radiation	λ Wavelength (nm)	ν Frequency (Hz)	Energy (kJ/mole)
Cosmic rays	$<10^{3}$ $>3*10^{20}$	$>1.2*10^{8}$	$1.2*10^{6}$ to $1.2*10^{8}$
Gamma rays	10^{-1} to 10^{-3}	$3*10^{18}$ to $3*10^{20}$	$1.2*10^{4}$ to $1.2*10^{6}$
X-rays	10 to 10^{-1}	$3*10^{16}$ to $3*10^{18}$	$6*10^{2}$ to $1.2*10^{4}$
Far ultraviolet	200 to 10	$1.5*10^{15}$ to $3*10^{16}$	$3.2*10^{2}$ to $6*10^{2}$
Ultraviolet	380 to 200	$8*10^{14}$ to $1.5*10^{15}$	$1.6*10^{2}$ to $3.2*10^{2}$
Visible	780 to 380	$4*10^{14}$ to $8*10^{14}$	4 to $1.6*10^{2}$
Infrared	$3*10^{4}$ to 780	10^{13} to $4*10^{14}$	0.4 to 4
Far infrared	$3*10^{5}$ to $3*10^{4}$	10^{12} to 10^{13}	$4*10^{-3}$ to 0.4
Microwaves	$3*10^{7}$ to $3*10^{5}$	10^{10} to 10^{12}	$4*10^{-3}$ to 0.4
Radio frequency	10^{11} to $3*10^{7}$	10^{6} to 10^{10}	$4*10^{-7}$ to $4*10^{-3}$

In bioprocessing, an important spectroscopic technique is the use of fluorescence, which occurs when a molecule, excited into an unstable singlet electronic state by incident radiation, "relaxes" to its stable electronic state, emitting photons. Many aromatic compounds of biological significance, such as vitamins, amino acids, proteins, etc., tend to

fluoresce when excited with ultraviolet (UV) radiation or visible light of the proper wavelength.

C. Neural Networks as Soft Sensors for Quantitative Predictions of Product Compositions from Fluorescent Spectra of Biomolecules

In the following case study, we wish to demonstrate how to use neural networks to quantitatively predict product compositions from fluorescent spectra of biomolecules. To do so, we will consider the fluorescent spectra for a mixture of two aromatic amino acids, tyrosine and tryptophan. Our experimental data come from the study done by McAvoy et al. (1992), and are kindly provided by Professor Nam-Sun Wang of the University of Maryland.

Table 4.8 summarizes the total solution concentration (1.0E-03 to 1.0E-06 M), tyrosine mole fraction (0.0 to 1.0), tyrosine concentration (0.00E-6 to 1.00E-03 M), and tryptophan concentration (0.00E-6 to 1.00E-3 M) for the forty-six experimental fluorescent spectra. Figures 4.6a-h show the corresponding fluorescent spectra for the experiments of Table 4.8.

Table 4.8. The experimental runs for the laser fluoerescent spectra of the tyrosine/tryptophan binary mixture.

Experiment number	Total concentration	Tyrosine mole fraction	Tyrosine concentration	Tryptophan concentration
1	1.0E-03	0.00	0.00E-03	1.00E-03
2	1.0E-03	0.10	0.10E-03	0.90E-03
3	1.0E-03	0.20	0.20E-03	0.80E-03
4	1.0E-03	0.25	0.25E-03	0.75E-03
5	1.0E-03	0.30	0.30E-03	0.70E-03
6	1.0E-03	0.40	0.40E-03	0.60E-03
7	1.0E-03	0.50	0.50E-03	0.50E-03
8	1.0E-03	0.60	0.60E-03	0.40E-03
9	1.0E-03	0.70	0.70E-03	0.30E-03
10	1.0E-03	0.75	0.75E-03	0.25E-03
11	1.0E-03	0.80	0.80E-03	0.20E-03
12	1.0E-03	0.90	0.90E-03	0.10E-03
13	1.0E-03	1.00	1.00E-03	0.00E-03
14	1.0E-04	0.00	0.00E-04	1.00E-04
15	1.0E-04	0.10	0.10E-04	0.90E-04
16	1.0E-04	0.25	0.25E-04	0.75E-04
17	1.0E-04	0.50	0.50E-04	0.50E-04
18	1.0E-04	0.90	0.90E-04	0.10E-04
19	1.0E-04	1.00	1.00E-04	0.00E-04
20	1.0E-05	0.00	0.00E-05	1.00E-05
21	1.0E-05	0.10	0.10E-05	0.90E-05
22	1.0E-05	0.25	0.25E-05	0.75E-05
23	1.0E-05	0.50	0.50E-05	0.50E-05
24	1.0E-05	0.75	0.75E-05	0.25E-05
25	1.0E-05	0.90	0.90E-05	0.10E-05
26	1.0E-05	1.00	1.00E-05	0.00E-05
27	3.0E-05	0.00	0.000E-05	3.000E-05
28	3.0E-05	0.01	0.030E-05	2.970E-05
29	3.0E-05	0.025	0.075E-05	2.925E-05
30	3.0E-05	0.05	0.150E-05	2.850E-05
31	3.0E-05	0.10	0.300E-05	2.700E-05
32	3.0E-05	0.25	0.750E-05	2.250E-05
33	3.0E-05	0.50	1.500E-05	1.500E-05
34	3.0E-05	0.75	2.250E-05	0.750E-05
35	3.0E-05	0.90	2.700E-05	0.300E-05
36	3.0E-05	0.95	2.850E-05	0.150E-05
37	3.0E-05	0.975	2.925E-05	0.075E-05
38	3.0E-05	0.99	2.970E-05	0.030E-05
39	3.0E-05	1.00	3.000E-05	0.000E-05
40	1.0E-06	0.00	0.00E-06	1.00E-06
41	1.0E-06	0.10	0.10E-06	0.90E-06
42	1.0E-06	0.25	0.25E-06	0.75E-06
43	1.0E-06	0.50	0.50E-06	0.50E-06
44	1.0E-06	0.75	0.75E-06	0.25E-06
45	1.0E-06	0.90	0.90E-06	0.10E-06
46	1.0E-06	1.00	0.00E-06	0.00E-06

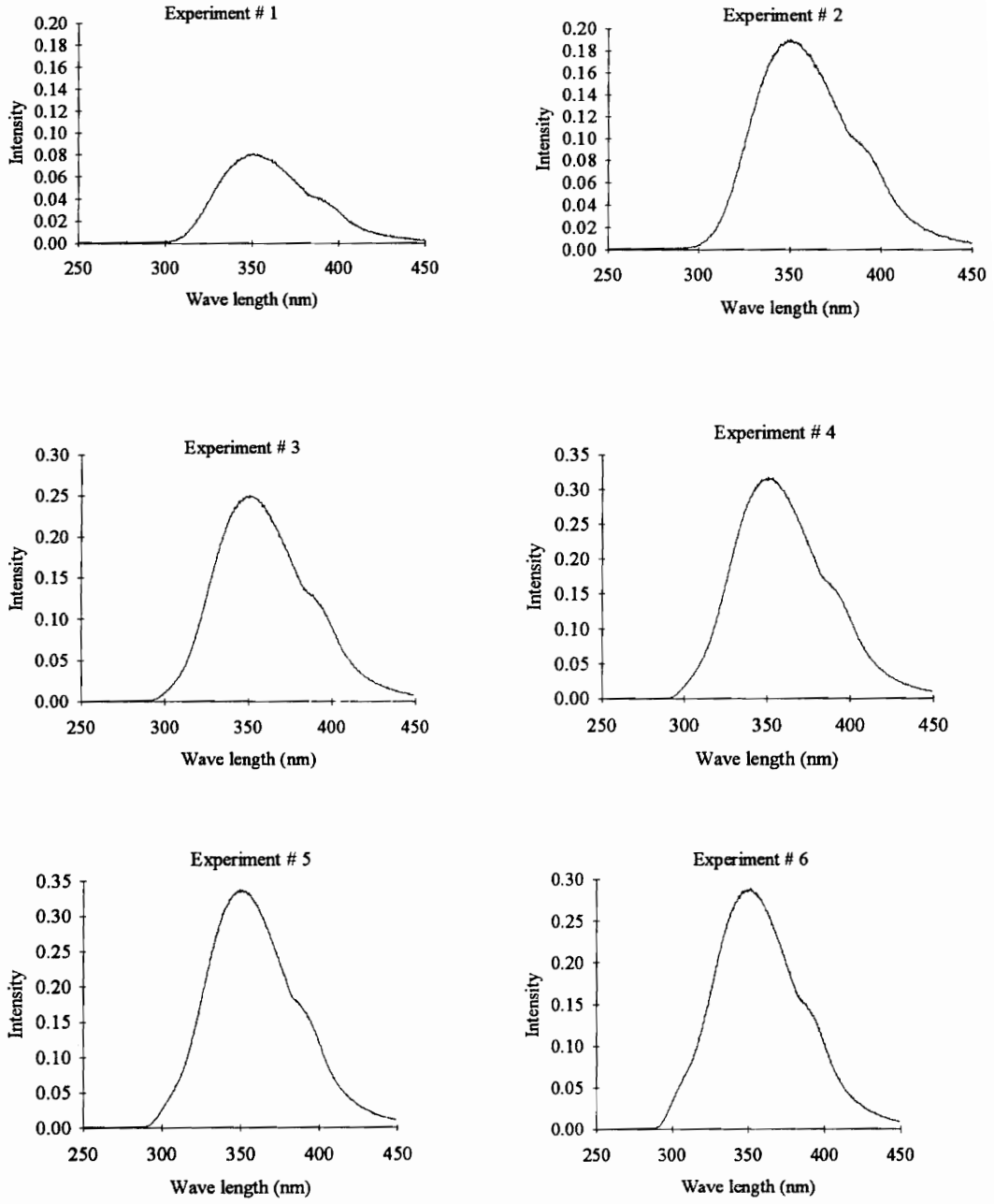


Figure 4.6a. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

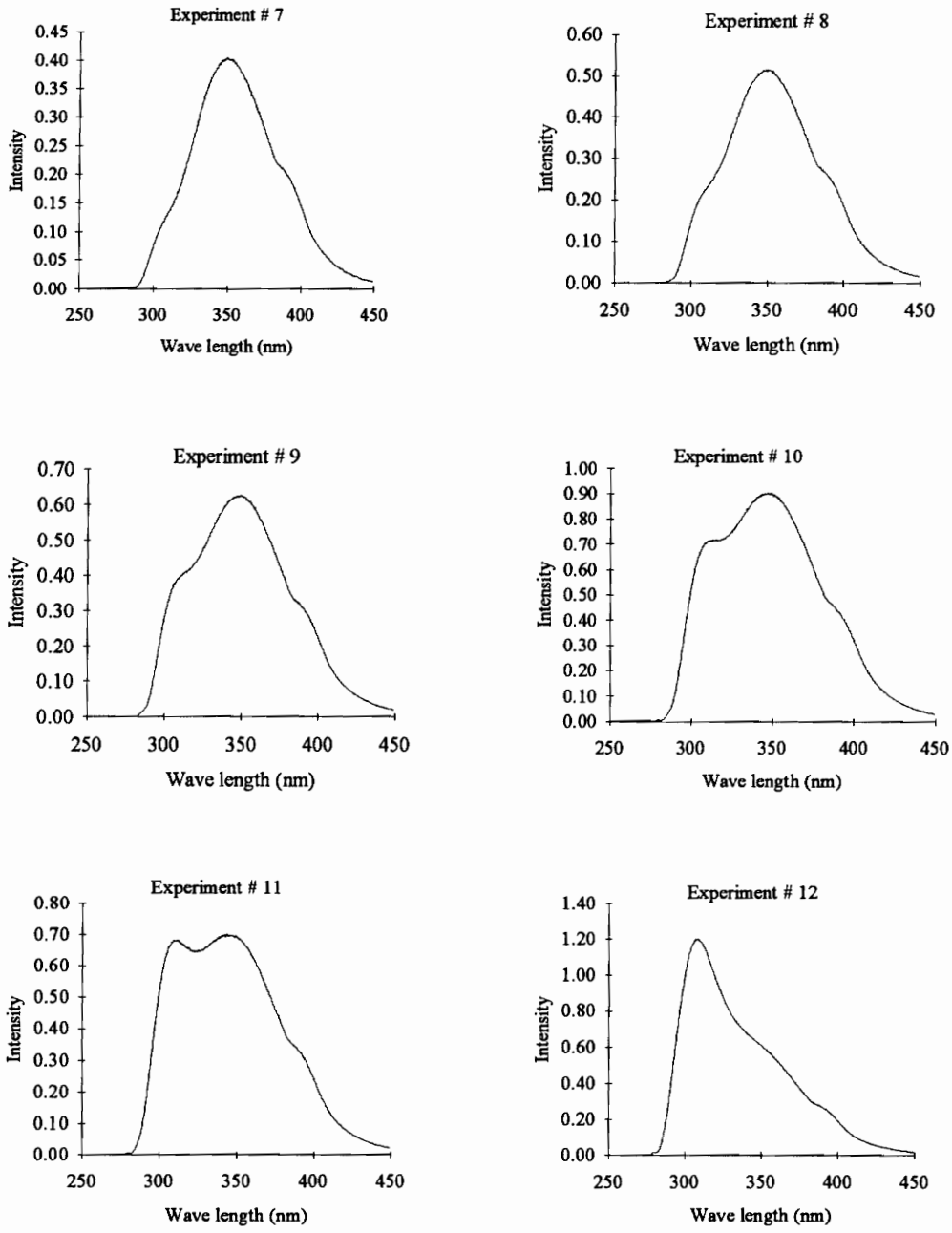


Figure 4.6b. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

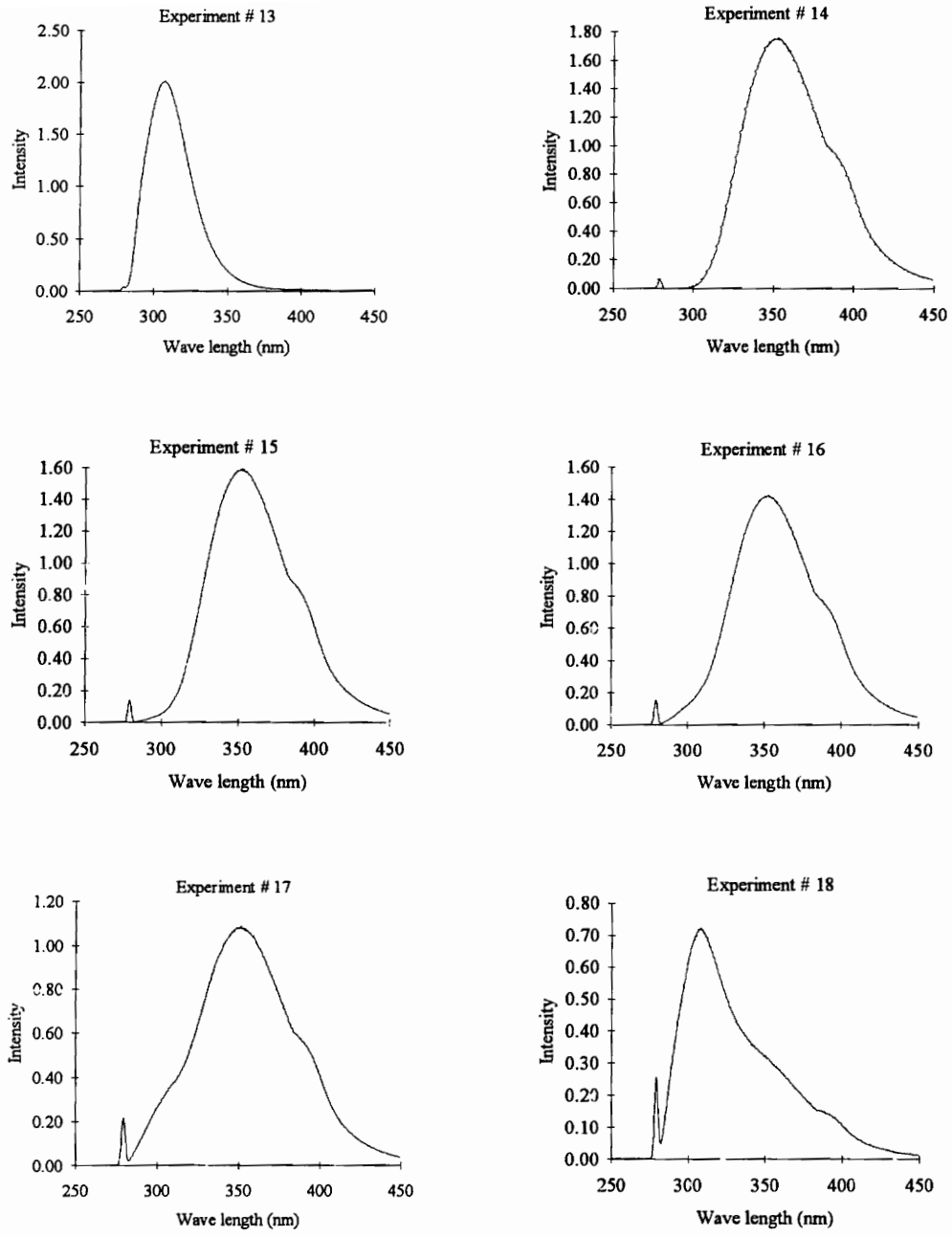


Figure 4.6c. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

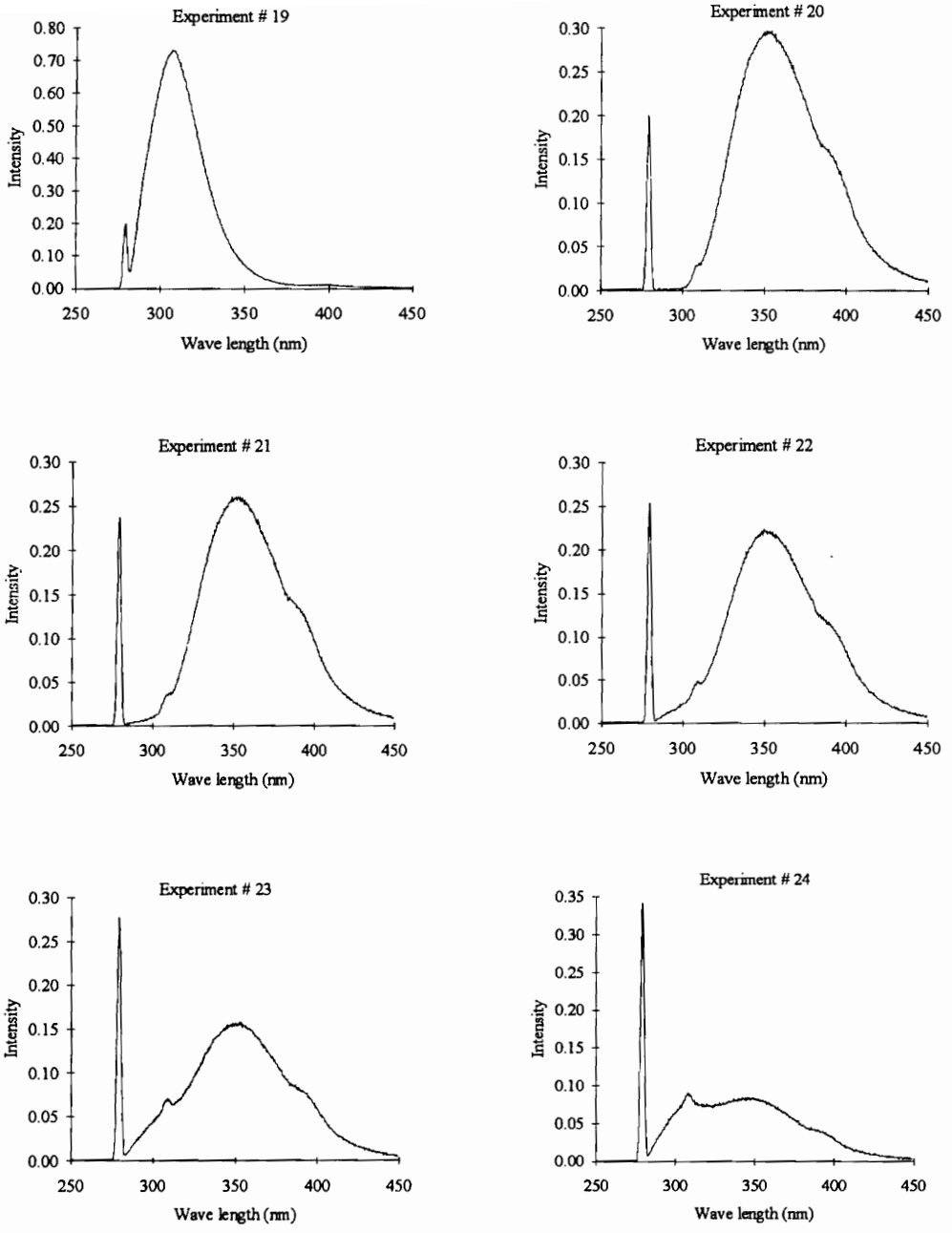


Figure 4.6d. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

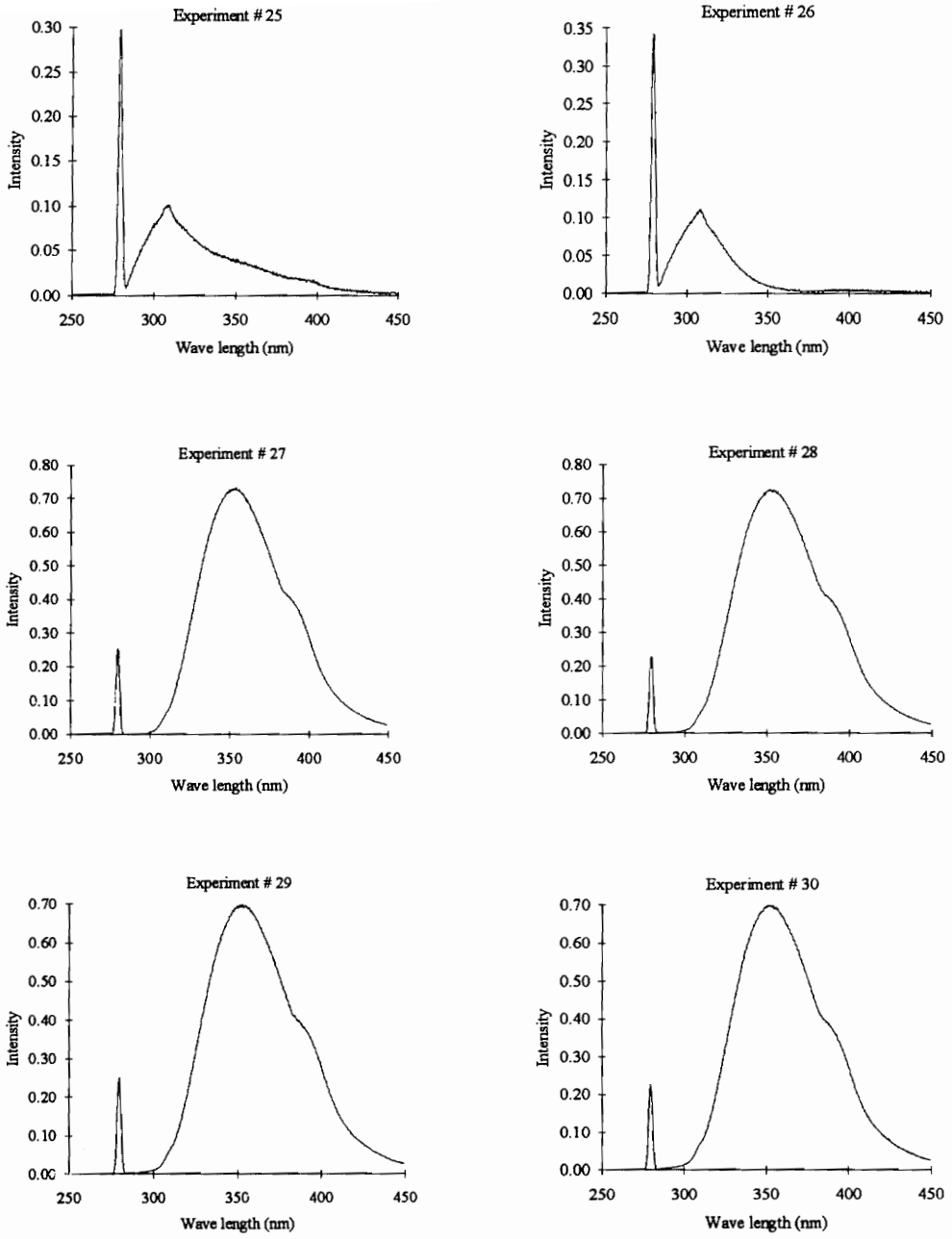


Figure 4.6e. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

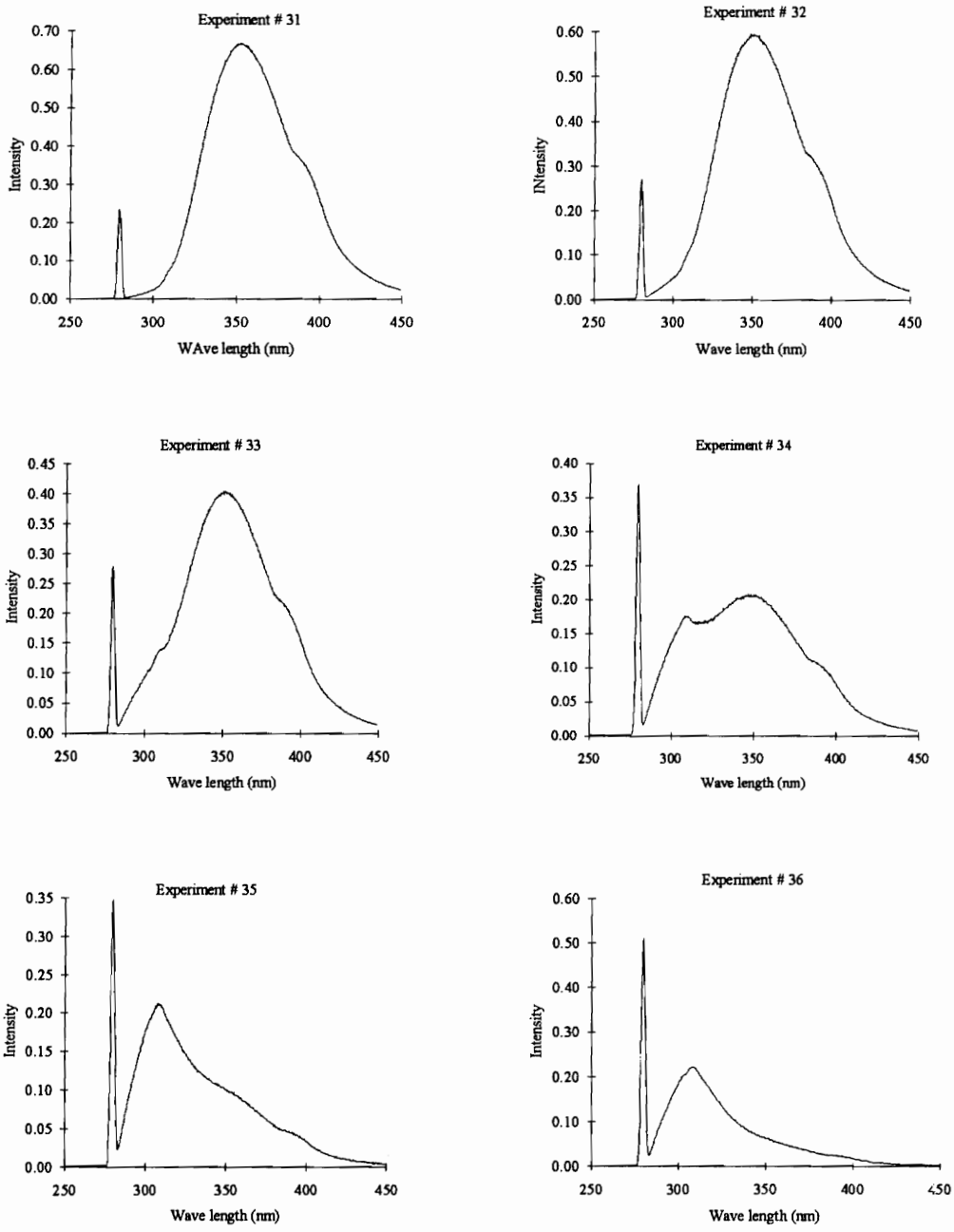


Figure 4.6f. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

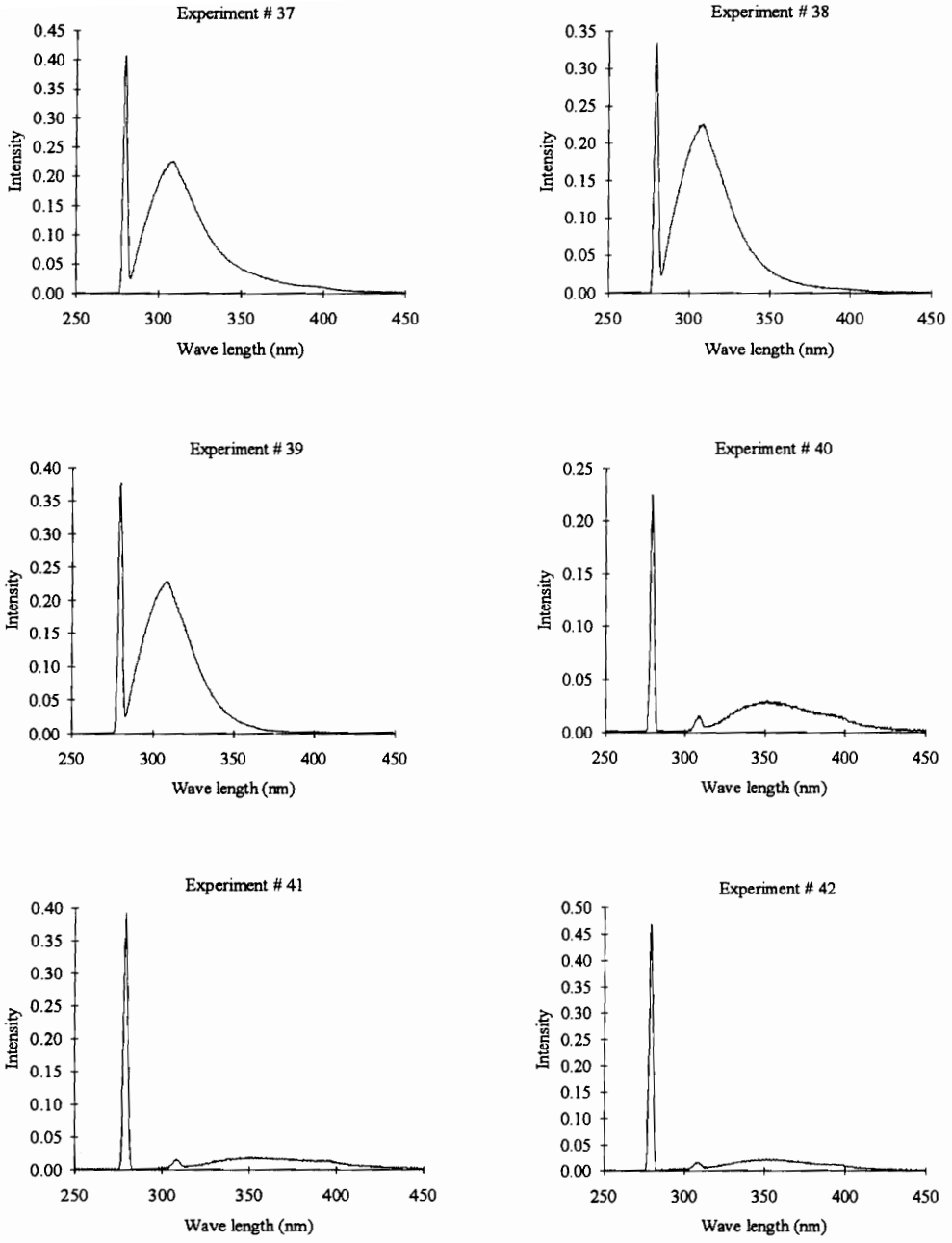


Figure 4.6g. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

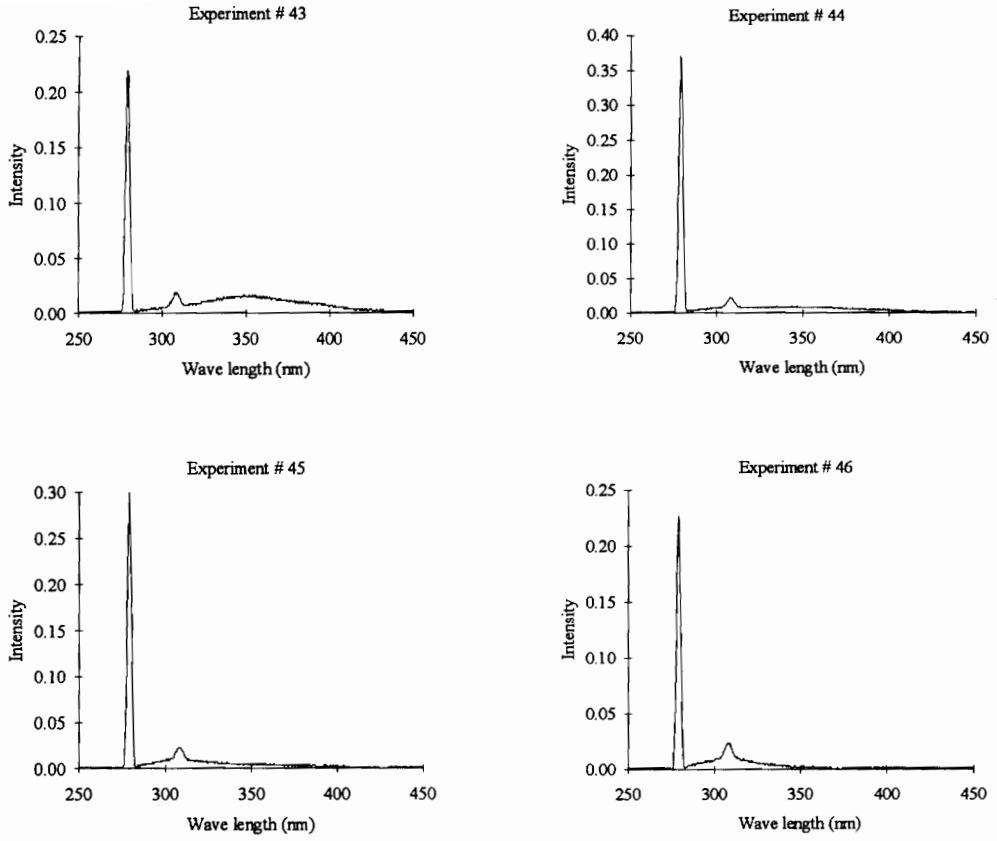


Figure 4.6h. Experimental fluorescent spectra for a mixture of tyrosine and tryptophan (McAvoy et al., 1992).

1. Development of a Neural Network Model for Predicting Biomolecule Compositions from Fluorescent Spectra

Our second practical suggestion in applying neural networks to prediction problems is that *nonlinear data should be subdivided into linear segments*. In this case, given spectroscopic signals of widely varying spectrum intensities at different wavelengths such as those in Figures 4.6a-h, we first decompose the signals into approximately linear segments. For example in Figure 4.7, we represent the spectrum intensities at 30 incremental wavelengths as I_1, I_2, ..., I_{30}, starting at a wavelength of 250 nm and ending at a wavelength of 429.1 nm, with 4.8 nm increments. This decomposition is an essential first step in quantitatively identifying the input variables to a spectrum-based neural network for predicting other relevant variables such as biomolecule compositions.

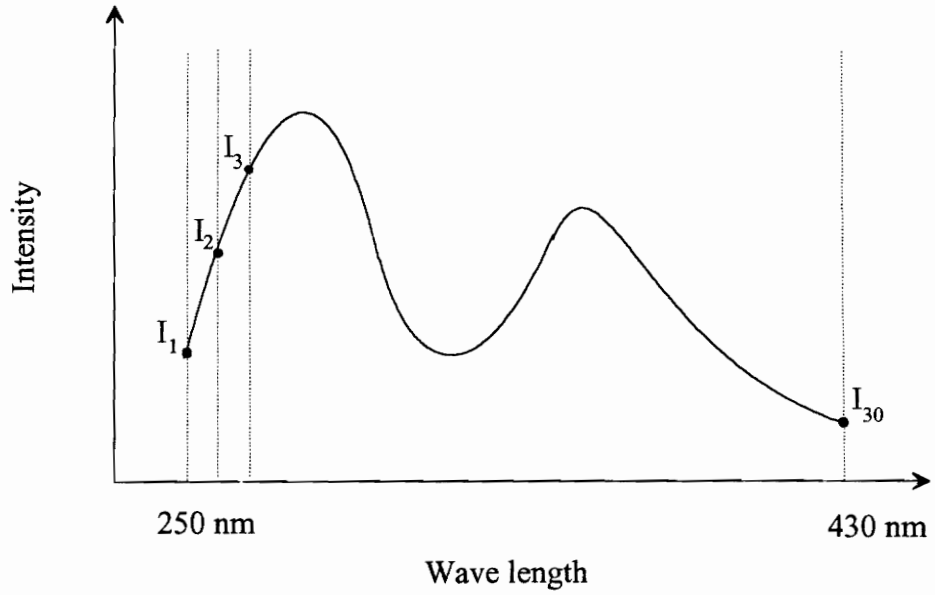


Figure 4.7. Decomposition of typical fluorescent spectrum into approximately linear segments at thirty incremental wavelengths.

As we emphasized in Section 4.2B, a backpropagation network with two or three hidden layers is an excellent architecture for prediction problems in bioprocessing and chemical engineering. Figure 4.8 shows the architecture of the tyrosine-composition prediction network. There are 30 input variables, $I_1 \ldots I_{30}$, representing spectrum intensities and one output variable, $x_{tyrosine}$ (tyrosine mole fraction). The network does not include the tryptophan composition since it depends on the tyrosine composition.

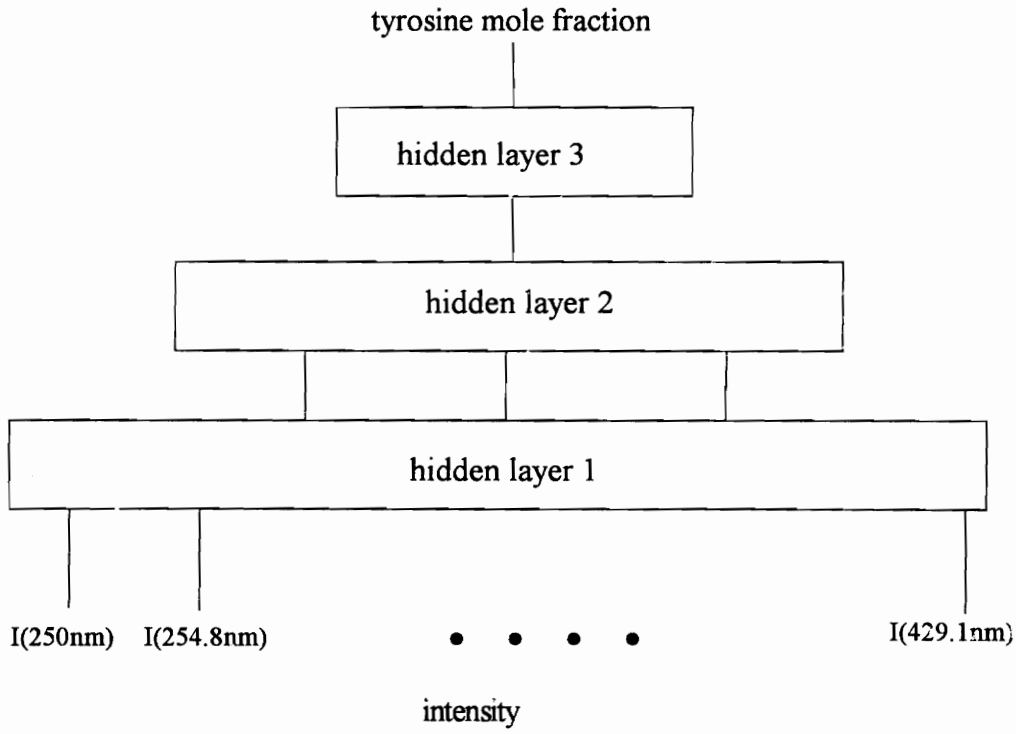


Figure 4.8. The architecture of the tyrosine-composition prediction network.

Table 4.9 shows the format of the data file *fluor.nna* used to train tyrosine-composition predict network, and Table 4.10 lists the training specifications.

Table 4.9. The format of the file *fluor.nna* used for training the tyrosine-composition prediction network.

Column number	Variable type	Normalized variable	Normalization factor
1	input	intensity (250.0 nm)	
2	input	intensity (254.8 nm)	
•	•	•	
•	•	•	* 1
•	•	•	
29	input	intensity (424.3 nm)	
30	input	intensity (429.1 nm)	
31	output	tyrosine mole fraction	* 1

Table 4.10. The specifications of the tyrosine-composition prediction network.

Network type	backpropagation		
Training file name	*fluor.nna*		
Transfer function (input layer)	linear		
Transfer function (hidden layers)	tanh		
Transfer function (output layer)	tanh		
Learning rule	delta rule		
Summation	sum		
Error	standard		
Network weight distribution	normal distribution : 3 σ limits of [-1, 1]		
	input Layer		
Training iteration	5,000		
Noise	0		
Learning rate	0.9		
Momentum coefficient	0.6		
Error tolerance	0		
	hidden layer 1		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.3	0.15	0.04
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	hidden layer 2		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.25	0.13	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	hidden layer 3		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.2	0.1	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	output layer		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.15	0.08	0.02
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

We determine the optimal configuration for the hidden layers by varying the number of layers from 1 to 3 and the total number of nodes within the layers from 5 to 45. The two-layer networks are maintained at a 2:1 ratio of nodes in the first layer to nodes in the second layer for each network tested. The three-layer networks have ratios close to 3:2:1, for layers 1, 2, and 3, respectively. Table 4.11 shows how number of layers and layer size affect network performance. Note that each network is trained with 100,000 iterations.

Table 4.11. A comparison of the effectiveness of different hidden-layer configurations for training the tyrosine-composition prediction network.

Number of nodes			Total number of nodes	Average error tyrosine mole fraction
hidden layer 1	hidden layer 2	hidden layer 3		
5	0	0	5	0.071
10	0	0	10	0.064
20	0	0	20	0.058
30	0	0	30	0.063
40	0	0	40	0.061
6	3	0	9	0.079
10	5	0	15	0.035
20	10	0	30	0.035
30	15	0	45	0.019
9	6	3	18	0.060
15	10	5	30	0.049
20	12	7	39	0.035

Figure 4.9 illustrates how the total number of nodes in the hidden layers affects the RMS error for the recall of training data. The prediction capability of the network increases significantly going from 1 to 2 hidden layers, but varies little with the addition of a third layer.

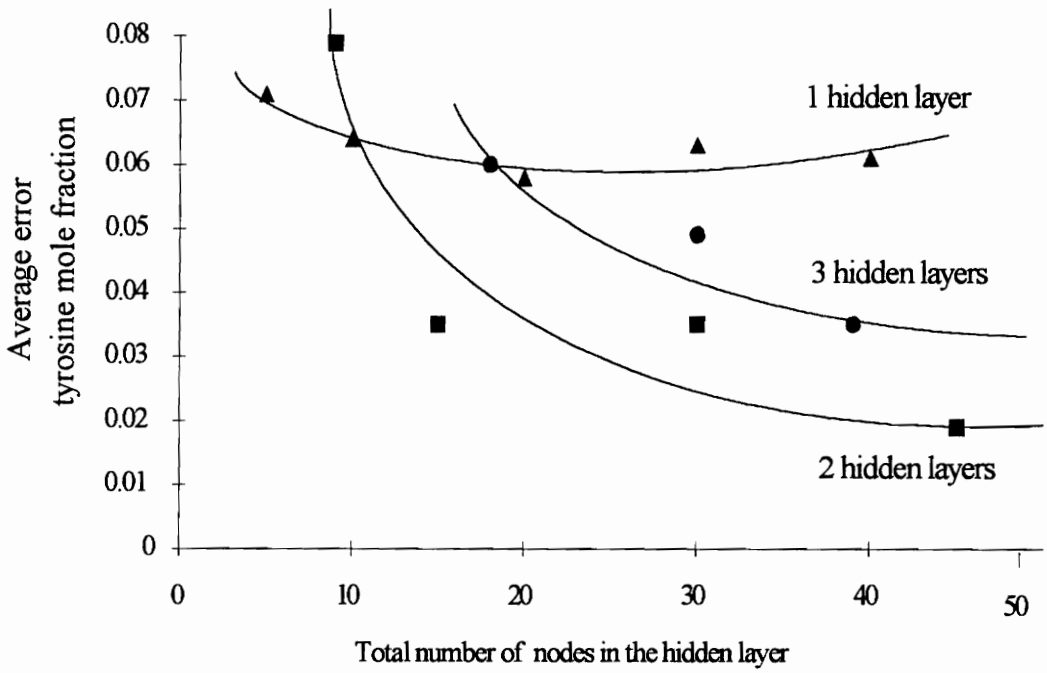


Figure 4.9. A comparison of the RMS error for the recall of training data for a varying number of total nodes in 1, 2 and 3 hidden-layer tyrosine-composition prediction network.

In general, always select the network that has the lowest number of layers that effectively models the system to avoid generalization problems later. In this case, based only on training-data recall, we select 30 nodes in the first hidden layer and 15 in the

second as the most effective configuration. We will later demonstrate how different network structures affect generalization of new training examples.

Figure 4.10 shows the RMS errors for 100,000 iterations in training the tyrosine-composition network.

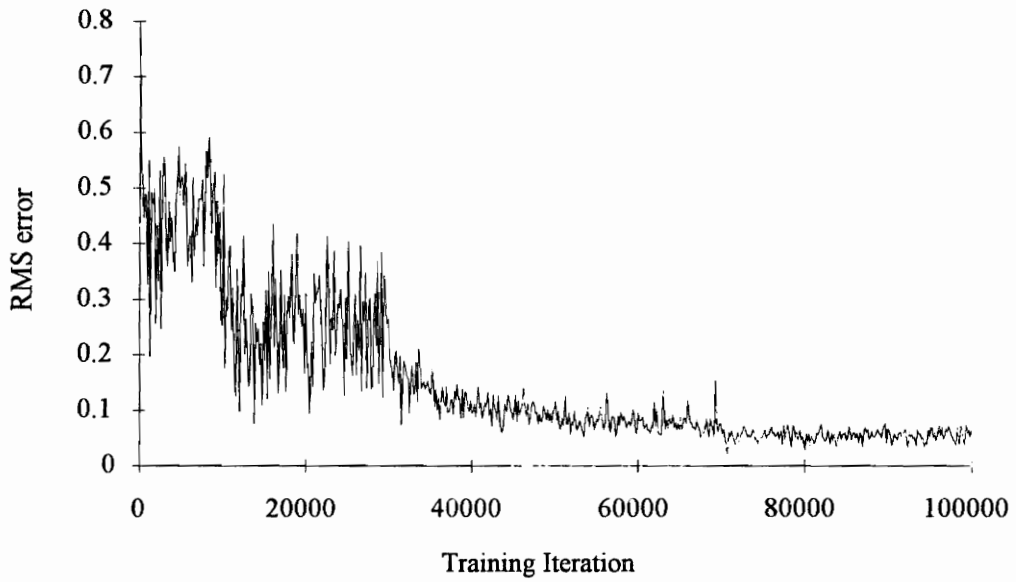


Figure 4.10. The training of the tyrosine-composition prediction network for predicting tyrosine mole fraction using a network configured with 30 and 15 nodes for hidden layers 1 and 2, respectively.

Figure 4.11 presents a scatter plot network predictions of tyrosine mole fractions versus actual values used in training.

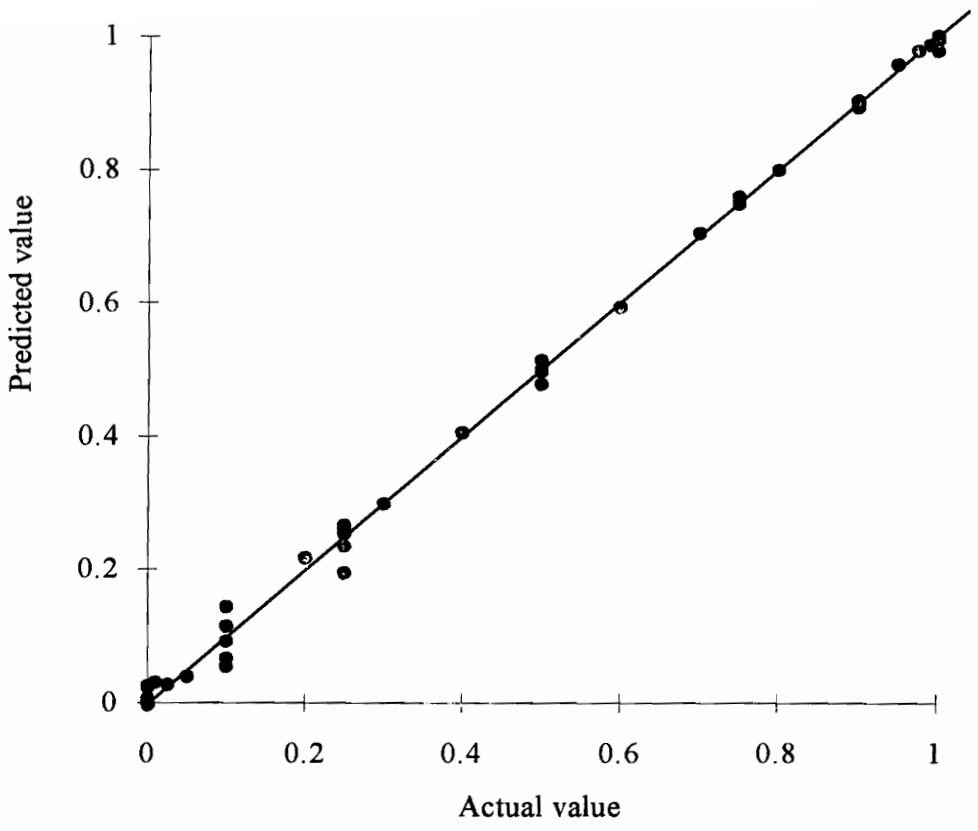


Figure 4.11. A scatter plot of predicted tyrosine mole fraction versus actual values from the training examples used in the tyrosine-composition prediction network.

We can then test the prediction network for its generalization of new training examples by generating a learning curve as described in Section 2.2.F.6. As seen in Table 4.12, we divide the forty-six experimental runs into eight groups of five experiments each and one group of six experiments. We train the network initially with one group, and test

it for the generalization of the other eight groups. We then proceed to add one group from that generalization set to the training examples, continuing the process until all the groups have been used. We then determine the overall effectiveness of the network in predicting tyrosine mole fractions under operating conditions outside of training examples based on how closely the average error in the generalization matches the average error for the training data set. We reset the iteration counter to 30,000 at the end of each training run, so that the next training run has the desired set of the network specifications (i.e., learning rate of hidden layer 1 = 0.3; etc. from Table 4.10).

The initial training set should include examples that span both a broad set of operating conditions and the entire distribution of output responses, as discussed in Section 2.2.F.6. In this study, the first data group used for the initial training includes the tyrosine mole fractions of 0.0, 0.5, and 1.0 for a total concentration of 1.0E-03 M (experimental runs 1, 7, and 13 in Table 4.8). We also include the tyrosine mole fractions of 0.5 for total concentrations 1.0E-5 and 1.0E-6 M (experimental runs 23 and 43). We put the remaining data sets randomly into the eight subsequent groups shown in Table 4.12. Table 4.12 also lists the average error obtained for recall and generalization in our incremental network training.

Table 4.12. Average errors for the recall and generalization of the tyrosine-composition prediction network (30:15 hidden-layer configuration) used in the generation of the learning curve.

Training			Recall	Generalization
Run numbers added to network	Experiments in training set	Experiments in testing set	Avg error tyrosine mole fraction	Avg error tyrosine mole fraction
1, 7, 13, 23, 43	5	41	0.001	0.316
10, 16, 26, 31, 40	10	36	0.009	0.141
3, 18, 20, 36, 46	15	31	0.014	0.097
2, 14, 24, 28, 33	20	26	0.010	0.078
4, 12, 17, 34, 41	25	21	0.011	0.053
6, 25, 39, 42, 44	30	16	0.010	0.044
11, 15, 22, 32, 37	35	11	0.013	0.025
5, 9, 19, 27, 35	40	6	0.011	0.016
8, 21, 29, 30, 38, 45	46	0	0.011	----------

Figure 4.12 illustrates the learning curves generated from the average errors presented in Table 4.12. The errors associated with recall are at reasonable levels throughout all stages of the learning process, increasing slightly but insignificantly as we introduce additional training examples. In comparison, the generalization curve has large prediction errors when only five fluorescent spectra are used for training. The average error decreases rapidly as we include each additional group of experimental runs. In the later stages of the learning curve, the average errors for generalization approaches that for recall, indicating that training is near completion and may require only a few new experimental runs.

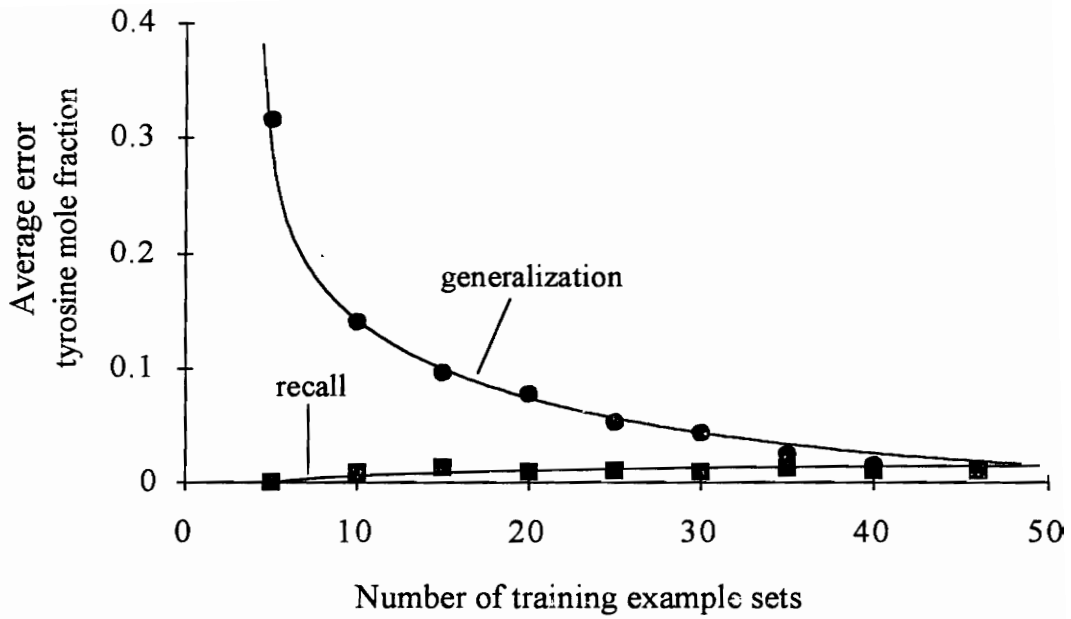


Figure 4.12. The learning curve for training of the tyrosine-composition prediction network (30:15 hidden-layer configuration).

2. Related Studies

A number of investigations have applied other sensor-data analysis techniques, together with single-hidden-layer backpropagation networks, to the preceding case study. These include partial least squares (PLS) (McAvoy et al., 1992; Holcomb and Morari, 1992), and principal component analysis (PCA) (Duong and McAvoy, 1994). An excellent reference on PLS is the tutorial article by Geladi and Kowalski (1986a). Haaland and

Thomas (1988) report on the application of PLS to spectral analysis, and MacGregor et al. (1994) describe the use of PLS in process monitoring and diagnosis. For PCA, an excellent reference is the text by Jolliffe (1986), and two useful articles are Jackson and Mudholkar (1979) and Dong and McAvoy (1994).

McAvoy et al. (1992) combine the PLS method with a backpropagation algorithm to train a network with a single hidden layer, and call their resulting training method the NNPLS algorithm. Their optimal network for the tyrosine-composition prediction problem has 5 nodes in the hidden layer. The RMS error for NNPLS is 0.154, compared to 0.263 for backpropagation only and to 0.229 for PLS only (33 data sets used in their study in comparison to our 44 data sets). Their results for backpropagation with only 5 hidden nodes are similar to ours, but we find that the RMS error decreases further in a two-hidden-layer network. As the previous section shows, a network having 30 nodes in the first hidden layer and 15 in the second *does not have memorizing problems*. The learning curve (Figure 4.12) shows that the average errors for the training and testing data sets approach each other at values of 0.011 and 0.016, respectively, confirming our confidence in the improved results from a larger two-hidden-layer network. We conclude that although the more complex training algorithms, such as NNPLS, are more accurate in training networks with a *single* hidden layer, their prediction accuracy may not be better than that of a backpropagation network with *two* hidden layers. For this reason, we do not discuss either PLS or PCA here, and instead refer readers to the references cited above for additional information. We recommend again a simple backpropagation with

two or three hidden layers for prediction problems in bioprocessing and chemical engineering.

4.4 Illustrative Case Study : Neural Networks for Process Quality Control and Optimization

A. Quality Control and Optimization in the Autoclave Curing Process for Manufacturing Composite Materials

The final case study in this chapter illustrates how to combine neural networks with statistical methods to predict and optimize performance variables in the manufacture of composite materials by autoclave curing. This section introduces some background on the composite manufacturing and the autoclave curing process (Wu, 1990; Shieh, 1992; Joseph and Wang, 1993; Pillai et al., 1994). In Sections 4.4B and C, we describe the development of the prediction network and optimization strategy for the autoclave curing process.

In this study, the term "composite material" refers to the complex material consisting of high-strength fibers embedded in a polymeric resin matrix. Composite materials have two significant advantages: (1) they are lighter and stronger than traditional structural materials, and (2) they can easily be formed into complex-shaped parts, thereby reducing the need for assembly. Because of these advantages, composite materials have seen tremendous growth over the past decade and have replaced hundreds of products

that previously involved metals. Among many areas of applications are automobiles, aircrafts, and construction.

Manufacturers use different combinations and arrangements of fibers, fillers, and resins to tailor the structures of composite materials to meet requirements in different environments. In general, manufacturing polymeric composite materials involves three steps:

- Determine the relative quantities of fiber, filler, and resin according to the final product qualities desired.
- Orient the fibers in a matrix to meet the product requirements.
- Apply temperature and pressure to treat and consolidate the resin.

Since most composite manufacturing methods rely heavily on highly skilled operators, manufacturing costs are high and quality control is difficult. As a result, fabrication is considered the bottleneck in the process.

Automating the manufacturing process provides the process stabilization and built-in quality control necessary to produce composite materials for high-volume applications in a cost-effective manner. Automation provides the ability to adjust processing conditions using a control computer in response to changing raw materials or product requirements, and the ability to handle unexpected situations satisfactorily. Moreover, a number of researchers have applied artificial intelligence techniques, including

expert systems and neural networks, to process automation in composite manufacturing (Servais et al., 1986; Joseph et al., 1992; Joseph and Wang, 1993; Pillai et al., 1994).

The autoclave curing process is used to manufacture composites with a thermosetting polymeric resin matrix. Initially, the polymer is a viscous fluid. When heat is applied, the resin solidifies through an irreversible chemical reaction, or "cure." During the cure, cross-linking occurs inside the matrix, increasing both the polymer's molecular weight and its viscosity. Eventually, a loose three-dimensional structure pervades the system, and the polymer behaves like a gel. Flow ceases, but reaction continues to finally yield a glassy solid.

Figure 4.13 shows a simplified schematic diagram for this process. The composite material, a fiber mat impregnated with unreacted thermosetting resins (called prepreg), is laid on a metal tool plate. Teflon parting sheets between the plate and the resin enable easy separation of the part from the tool. In addition, a bleeder cloth of the same shape as the desired part absorbs excess resin which may flow out, and an air breather removes the trapped air and volatized water. The whole assembly is sealed in a vacuum bag and placed in an autoclave. As the temperature and pressure in the autoclave are adjusted according to a prescribed cycle, known as the cure cycle, the prepreg is heated, melted, and allowed to flow into the desired shape and thickness. The cycle begins by curing the prepreg matrix at elevated temperatures to initiate and maintain the chemical reaction and to melt the stacked prepreg matrix in order to press it into the desired shape and thickness. The

autoclave is then pressurized with inert gases to squeeze the excess resin out of the matrix, to compress volatized bubbles, and to strengthen the consolidation of the matrix.

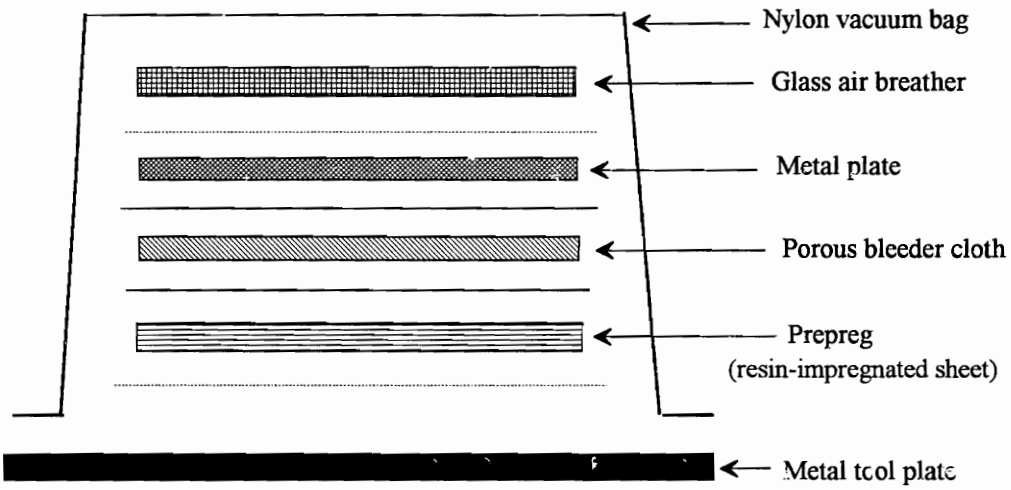


Figure 4.13. A simplified schematic diagram of the composite laminate setup in the autoclave curing process.

Figure 4.14 illustrates the temperature and pressure profiles of a typical cure cycle. The cycle consists of four temperature phases: (1) the first heating phase from temperature T_0 at time t_0 to temperature T_1 at time t_1; (2) the first holding phase at temperature T_1 from t_1 to t_2; (3) the second heating phase from temperature T_1 at time t_2 to temperature T_2 at time t_3; and (4) the second holding phase at temperature T_2 from t_3 to t_4. T_1 and T_2 are the set points of the PID controller for the temperature measured at the bottom of the

composite laminate. In conjunction with the temperature increases, the pressure increases stepwise from P_0 to P_1 at time t_{p1}. Applying pressure consolidates the prepreg and squeezes out excess resin. The stepwise increase should occur when the resin viscosity is low, that is, after the resin softens and melts and before it starts to react and cure. Unfortunately, measuring resin viscosity on-line is difficult and most of the time, we must rely on temperature measurements.

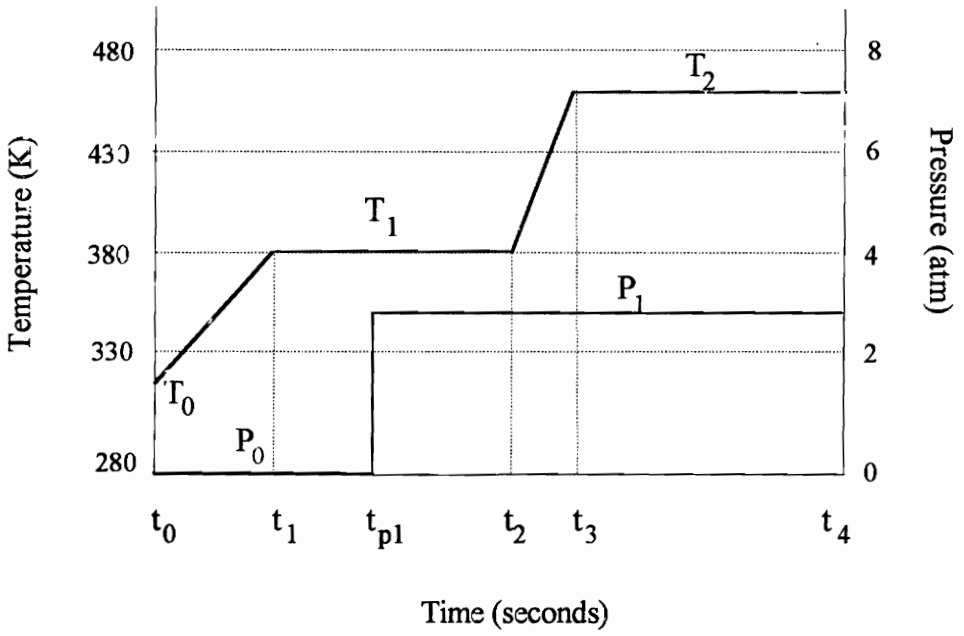


Figure 4.14. A schematic representation of temperature and pressure profiles in the autoclave in a cure cycle.

Wu (1990) and Shieh (1992) have identified the following important independent variables, operating parameters, and product properties for the autoclave curing process:

- Independent variables

  (1) initial resin weight fraction in the laminate, x_{resin}

  (2) number of plies

  (3) relative humidity

- Operating parameters

  (1) t_0, t_1, t_2, t_3, t_4: *predetermined* times for temperature setpoint changes (Figure 4.14).

  (2) T_0 : initial autoclave temperature

  (3) T_1 : first elevated temperature

  (4) T_2 : second elevated temperature

  (5) P_0 : vacuum pressure

  (6) P_1 : applied pressure

  (7) t_{p1} : time at which pressure is applied

- Product properties:

  (1) D_b : the diameter of gas bubble, which is assumed spherical (void size)

  (2) W_m : final resin weight fraction

  (3) L : final composite thickness

  (4) α : the conversion rate of reaction (extent of curing)

High product quality is defined by the absence of bubbles (i.e., small void size), a composite thickness within predetermined tolerances, and a very high extent of curing. As an example, we will set the following measures as the criteria for a good product : $D_b <$ 0.06 cm, 1.675 cm $< L <$ 1.75 cm, and $\alpha > 0.999$. Any product that falls outside these criteria is considered of poor quality.

Shieh (1992) has identified two major processing problems:

(1) Strong interactions among processing parameters:

- process temperature must be high enough to reduce the flow resistance of the matrix, but below the temperature threshold for uncontrolled curing.
- applied pressure must be high enough to suppress the growth of bubbles, but below the pressure threshold at which too much resin is squeezed out.
- if curing is completed prematurely or late, composite thickness cannot be adequately controlled; similarly, the time-applied pressure profile also affects the thickness.
- temperature and pressure gradients affect the final product performance.

(2) The delay time between the refrigeration of the prepreg matrix and the autoclave process can vary by up to two days, causing significant variability in product quality. The initial refrigeration is needed for storing the prepreg

materials to prevent them from starting to cure uncontrollably at room temperature.

B. Neural Networks in the Autoclave Curing Process

1. Background

A number of recent chemical engineering articles have described the predictive control of product quality in an autoclave curing process using artificial intelligence techniques, and specifically neural networks (Joseph et al., 1992; Joseph and Wang, 1993) and expert systems (Pillai et al., 1994). In particular, Joseph and Wang (1993) have developed a neural-network-based controller ("neurocontroller") to facilitate adjustments of operating parameters in order to achieve the specified product quality at the end of the cure cycle. The goal is to nullify the effects of feed variations, unmeasured disturbances, and changes in process equipment characteristics. However, since the article by Joseph and Wang focuses primarily on control aspects, these authors have not described their neural network architecture, nor the details of their network performance in predicting the process data from autoclave curing.

The next section demonstrates that the simple neural network architecture we recommend for prediction problems, i.e., a backpropagation network with one to two hidden layers, can accurately predict product qualities from operating parameters for an

autoclave curing process. We use the autoclave processing data from Shieh (1992) generated by a partial differential equation model with time and one spatial variable developed by Wu (1990). We use Shieh's (1992) second set of process simulator data for the autoclave, called autoclave process data II, developed to test highly nonlinear skewed data. The set of control variables used in their simulation are :

(1) initial weight fraction of the resin, x_{resin}

(2) initial autoclave temperature (K), T_0

(3) initial void size (cm), $D_{b,initial}$

(4) autoclave pressure (atm), P

(5) first elevated temperature (K), T_1

(6) second elevated temperature (K), T_2

(7) impurity in the feed, $x_{impurity}$

The major response variables that define product quality for the autoclave process are:

(1) final composite thickness (cm), L

(2) void size (cm), D_b

The simulation model generated two sets of data, including 280 examples to train the network and 98 examples to test the network. Figure 4.15a-b illustrate the distributions of the 7 process control variables and 2 response variables.

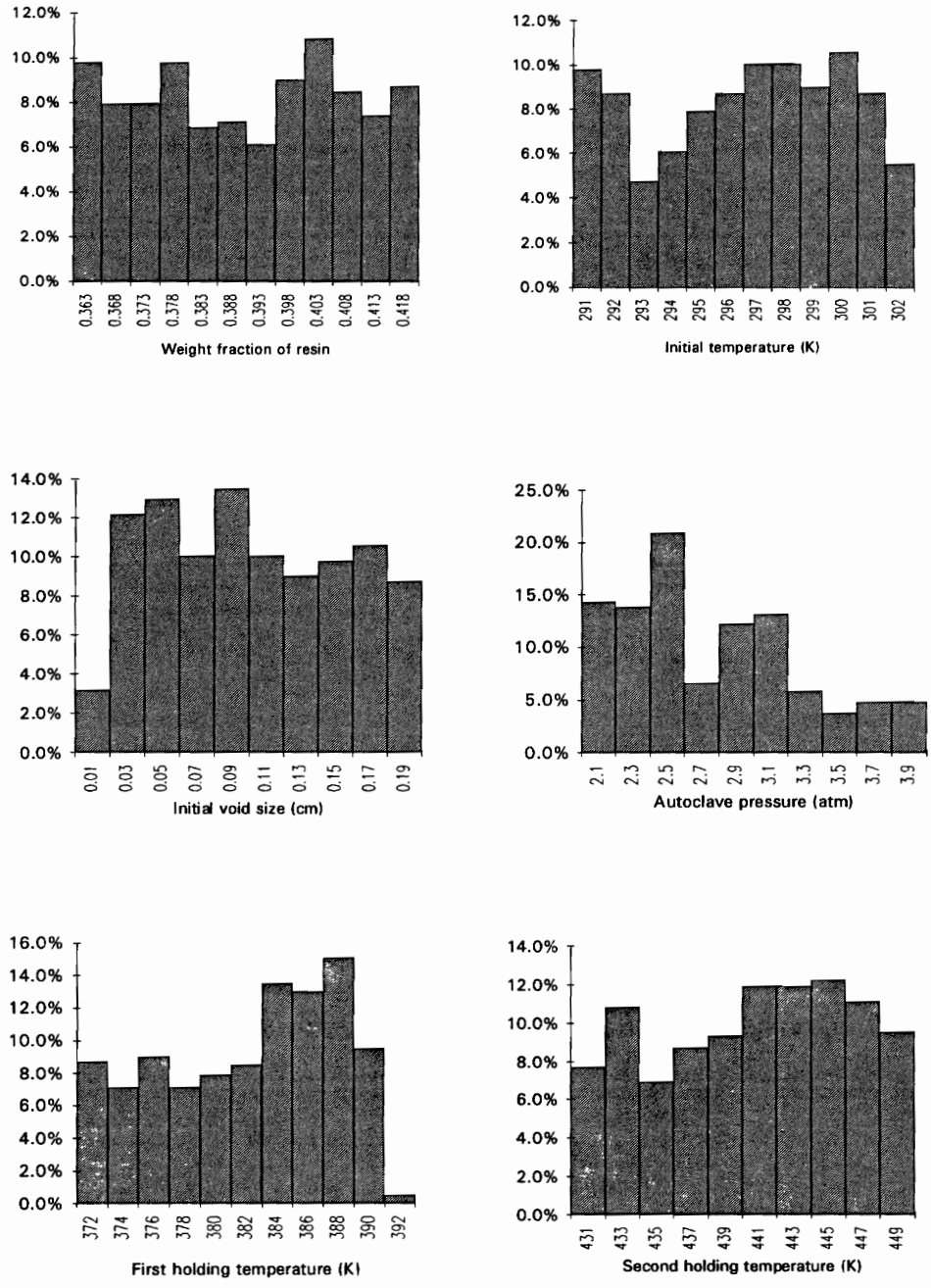


Figure 4.15a. The distributions of the 7 control variables and 2 response variables for the 280 training examples and 98 testing examples .

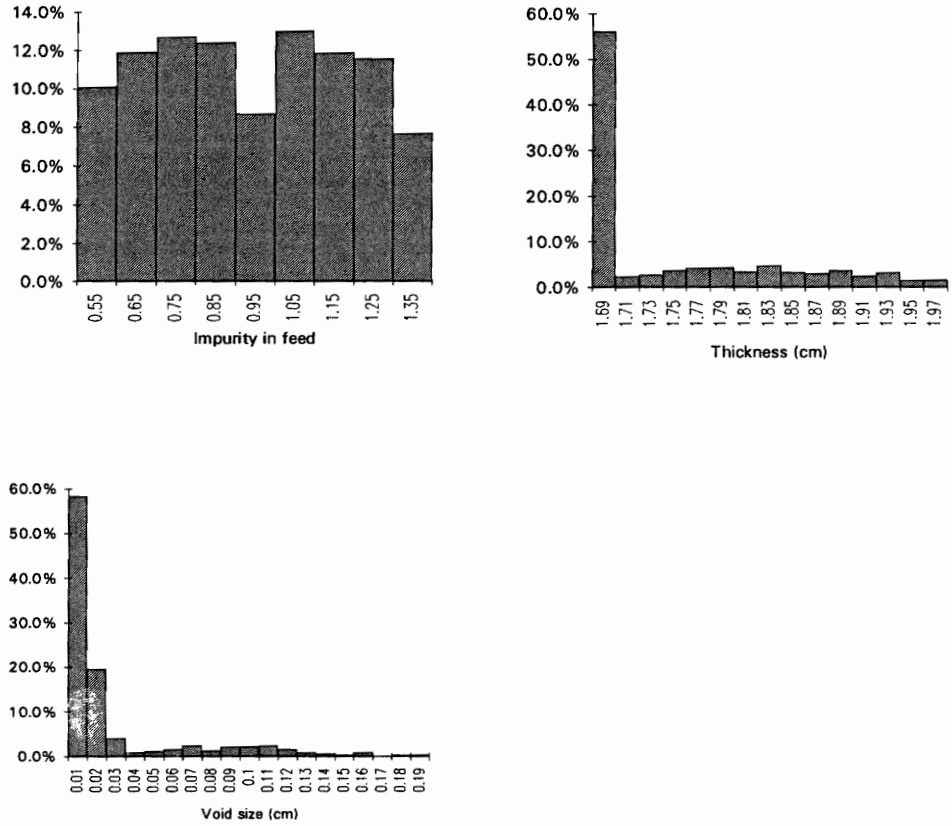


Figure 4.15b. The distributions of the 7 control variables and 2 response variables for the 280 training examples and 98 testing examples .

2. Network Architecture and Training

Figure 4.16 shows the overall architecture of the autoclave-processing network, which consists of 7 input variables and 2 output variables. The network predicts composite thickness and void size based on 7 primary control variables: initial weight fraction of the resin, x_{resin}; initial autoclave temperature, T_0; initial void size, $D_{b,initial}$; autoclave pressure, P;

first elevated temperature, T_1; second elevated temperature, T_2; and impurity in the feed, $x_{impurity}$.

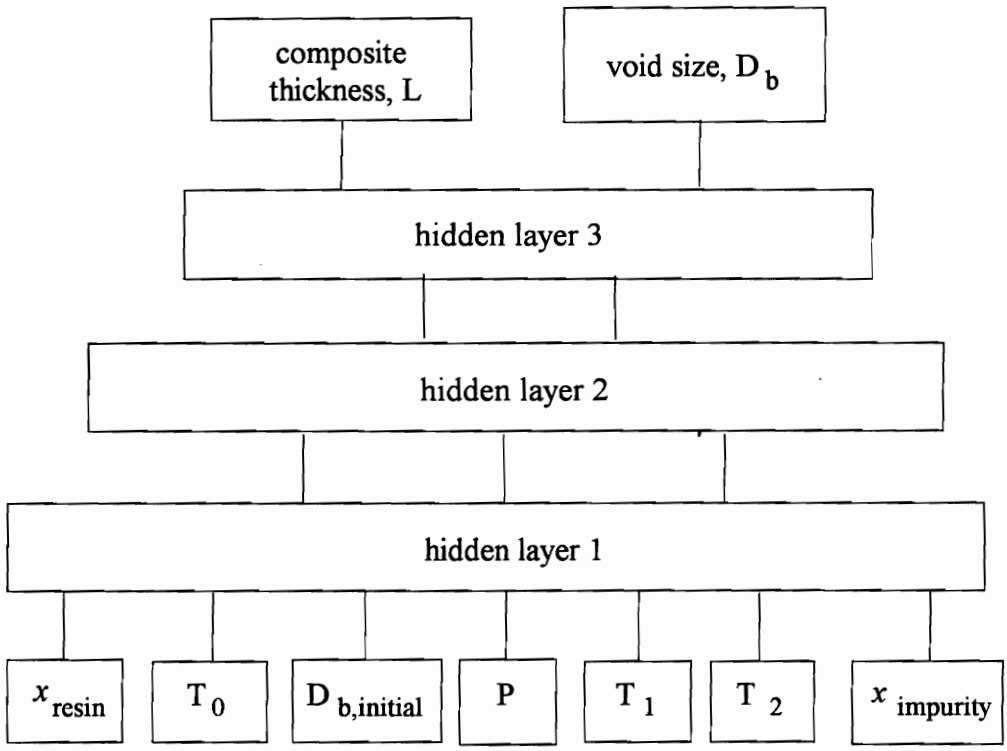


Figure 4.16. Architecture of the autoclave-processing network.

Note that the architecture of Figure 4.16 has 3 hidden layers. This results from our third practical suggestion in developing a neural network model for prediction problems: *for a complex problem, such as the autoclave curing process, begin with three hidden layers, and find out if the third hidden layer is necessary through network training and testing.*

For this network, we normalize the input and output vectors using the zero-mean normalization method, setting the mean value of each variable to zero and having the entire variable range span from -1 to +1 as discussed in Section 2.3.B. This step is critical in many nonlinear prediction models where the distribution of our input and output variables must coincide with the transfer function. The zero-mean normalization also puts the data into a format that is very compatible with response-surface optimization routines. We use the following equations to convert the raw data in the file *autoclv.dat* to the values used in our network training data sets, contained in files *thcktrn.nna, thcktst.nna, voidtrn.nna, and voidtst.nna*.

$$x_{resin} = \frac{(x_{resin} - 0.39)}{0.03} \tag{4.2}$$

$$T_0 = \frac{(T_0 - 296.1)}{6} \tag{4.3}$$

$$D_{b,initial} = \frac{(D_{b,initial} - 0.10)}{0.10} \tag{4.4}$$

$$P = \frac{(P - 2.68)}{1.4} \tag{4.5}$$

$$T_1 = \frac{(T_1 - 382.1)}{12} \tag{4.6}$$

$$T_2 = \frac{(T_2 - 440.9)}{10} \tag{4.7}$$

$$x_{impurity} = \frac{(x_{impurity} - 0.95)}{0.5} \tag{4.8}$$

$$L = \frac{(L - 1.7656)}{0.2} \tag{4.9}$$

$$D_b = \frac{(D_b - 0.0265)}{0.16} \tag{4.10}$$

Table 4.13 shows the format of the 4 files used for training and testing the autoclave-processing network. The two data files used for training the network, *thcktrn.nna* and *voidtrn.nna*, have 280 training examples each. The two files used for testing, *thcktst.nna* and *voidtst.nna*, contain 98 examples each.

Table 4.13. The format of the files for the autoclave-processing network: *thcktrn.nna*, *thcktst.nna*, *voidtrn.nna*, and *voidtst.nna*.

Column number	Variable type	Normalized variable	File name	Normalization factor
1	input	initial resin weight fraction, x_{resin}		
2	input	initial autoclave temperature, T_0 (K)		
3	input	initial void size, $D_{b,initial}$ (cm)		zero-mean
4	input	autoclave pressure, P (atm)		normalization
5	input	first elevated temperature, T_1 (K)		
6	input	second elevated temperature, T_2 (K)		
7	input	impurity in feed, $x_{impurity}$		
8a	output	final composite thickness, L (cm)	thcktrn.nna	
8b	output	final composite thickness, L (cm)	thcktst.nna	zero-mean
8c	output	void size, D_b (cm)	voidtrn.nna	normalization
8d	output	void size, D_b (cm)	voidtst.nna	

Table 4.14 lists the network specifications. We use a standard backpropagation network together with the delta rule and the hyperbolic tangent transfer function.

Table 4.14. The specifications of the autoclave-processing network.

Network type	backpropagation		
Training file name	*thcktrn.nna*, *thcktst.nna*, *voidtrn.nna*, and *voidtst.nna*.		
Transfer function (input layer)	linear		
Transfer function (hidden layers)	tanh		
Transfer function (output layer)	tanh		
Learning rule	delta rule		
Summation	sum		
Error	standard		
Network weight distribution	normal distribution : 3 σ limits of [-1, 1]		
	input Layer		
Training iteration	5,000		
Noise	0		
Learning rate	0.9		
Momentum coefficient	0.6		
Error tolerance	0		
	hidden layer 1		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.3	0.15	0.04
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	hidden layer 2		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.25	0.13	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	hidden layer 3		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.2	0.1	0.03
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1
	output layer		
Training iteration	10,000	30,000	70,000
Noise	0	0	0
Learning rate	0.15	0.08	0.02
Momentum coefficient	0.4	0.2	0.05
Error tolerance	0.1	0.1	0.1

To determine the optimal network configuration, we train and test the autoclave-processing network with a number of hidden-layer configurations. We use 1-, 2-, and 3-hidden-layer neural networks with the total number of nodes ranging from 5 to 45. Table 4.15 lists the average error for predicting the two product-quality variables, composite thickness and void size, for both the training and testing data sets for a variety of configurations. Average errors for the testing data set (not previously been seen by network) are slightly higher than those for the training data set by approximately 0.001 cm. This difference in error is reasonable, and we consider the network successfully trained.

As Table 4.15 shows, the prediction capability of the network increases significantly as we increase from one to two hidden layers, but adding a third hidden layer produces minimal improvement. This observation matches our findings in the two preceding case studies again pointing to a simple backpropagation network with two hidden layers as an excellent architecture for prediction problems. The training and testing errors vary slightly as the number of nodes within a hidden layer increases, but follow no specific trends. Therefore, we select the two-hidden-layer configuration which has the minimum errors: 20:10 for composite thickness and 10:5 for the void fraction.

Figure 4.17 shows the scatter plots for the training and testing of the autoclave-processing network. As seen, the network yields very good results for both the prediction of composite thickness and void size.

Table 4.15. The average errors for composite thickness, L, and void size, D_b, predicted by the autoclave-processing network for the training and testing data sets.

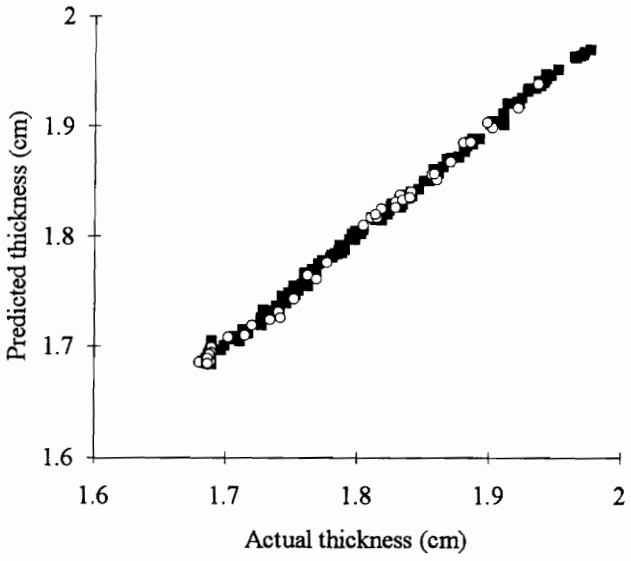
Composite thickness, L (cm)

Number of nodes in specified layer			Average error	
Hidden layer 1	Hidden layer 2	Hidden layer 3	Training	Testing
5	0	0	0.0035	0.0048
10	0	0	0.0035	0.0048
20	0	0	0.0035	0.0049
30	0	0	0.0037	0.0050
40	0	0	0.0039	0.0051
6	3	0	0.0024	0.0032
10	5	0	0.0025	0.0037
20	10	0	0.0024	0.0028
30	15	0	0.0026	0.0035
15	10	5	0.0021	0.0031
20	12	7	0.0023	0.0031

Void size, D_b (cm)

Number of nodes in specified layer			Average error	
Hidden layer 1	Hidden layer 2	Hidden layer 3	Training	Testing
5	0	0	0.0032	0.0024
10	0	0	0.0032	0.0023
20	0	0	0.0031	0.0024
30	0	0	0.0030	0.0024
40	0	0	0.0032	0.0024
6	3	0	0.0021	0.0020
10	5	0	0.0017	0.0015
20	10	0	0.0018	0.0016
30	15	0	0.0016	0.0015
15	10	5	0.0016	0.0016
20	12	7	0.0014	0.0014

(a)



(b)

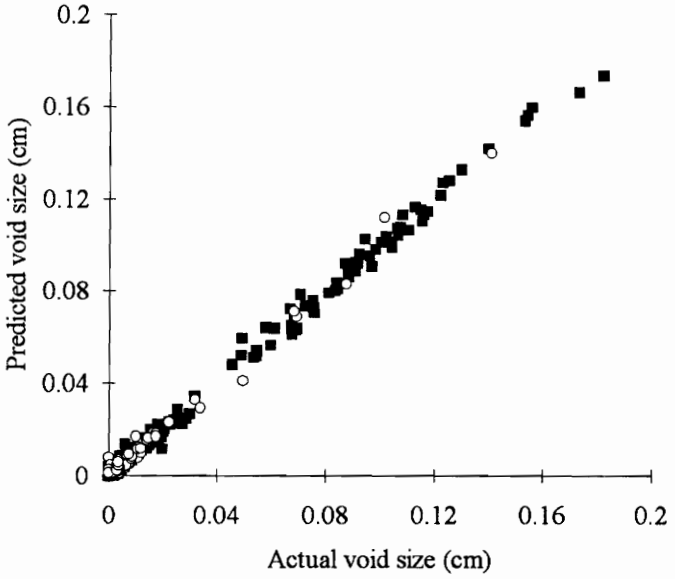


Figure 4.17. Scatter plots of composite thickness (a) and void size (b) predicted by the autoclave-processing network. (■) training data set (o) testing data set.

C. Optimization of the Autoclave Process Using Response-Surface Modeling and the Autoclave-Processing Network

In this section, we demonstrate how to use neural network predictors in conjunction with optimization techniques, such as response-surface modeling, to determine the optimal operating conditions of a process. We again focus on the autoclave process, and our objective is to minimize void size, $D_b < 0.06$ cm, while maintaining composite thickness within the specified tolerances of $1.675 \text{ cm} < L < 1.75$ cm. Starting with the first seven independent variables from the previous section, we use standard statistical methods to identify those which most significantly affect product quality. We then create a complete response-surface profile for the response variables using a neural network predictor to generate a large array of composite thicknesses and void sizes over the entire range of process operating conditions and raw material properties. Finally, we use the response-surface profiles to identify the feasible and optimal operating conditions.

1. Identification of Primary Process Factors

Because the statistical software available for our personal computer allows for only 6 factors affecting the response variable, we must eliminate one of the seven. The F-ratio test in statistics indicates that the initial void size is the least significant; hence, that is the one we eliminate from our response-surface analysis. Tables 4.16a-b show the results of the response-surface analysis and analysis-of-variance (ANOVA) report for predicting

composite thickness from the 6 control variables. For readers who are not familiar with statistic tests and analysis of variances, we recommend the excellent text by Box et al. (1978); a good reference on the response-surface analysis is the text by Khari and Cornell (1987).

Table 4.16a-b. Response-surface analysis and ANOVA report of composite thickness with respect to the 6 most significant operating parameters (factors).

(a) Response-surface analysis for composite thickness

	Sequential sum of squares	Mean square	F-ratio	Incremental R^2
Regression	63.93	2.37	241.13	0.9489
Linear	49.08	8.18	883.06	0.7285
Quadratic	3.43	0.57	58.23	0.0509
Cross-product	11.42	0.76	77.51	0.1695
Total error	3.43	0.01	1,320.67	0.0510

(b) ANOVA report for composite thickness

Factor	Last sum of squares	Mean square	F-ratio	Term R^2
x_{resin}	0.16	0.02	2.2	0.0026
T_0	0.10	0.01	1.5	0.0015
P	7.73	1.10	112.5	0.1147
T_1	19.65	2.81	285.9	0.2917
T_2	0.19	0.03	2.7	0.0028
$x_{impurity}$	41.99	6.00	610.9	0.6233

In Table 4.16a, the terms from the response-surface analysis refer to the following overall regression model for composite thickness, D_L:

$$D_L = (\text{linear terms}) + (\text{cross-product terms}) + (\text{quadratic terms}) \qquad (4.11)$$

$$= \sum_{i=1}^{6} a_i x_i + \sum_{\substack{i,j=1 \\ i \neq j}}^{6} b_{ij} x_i x_j + \sum_{i=1}^{6} c_i x_i^2$$

The incremental correlation coefficient, R^2, in Table 4.16a reflects the fit between the autoclave process data and the overall regression model, and the contributions of different terms (i.e., linear, cross-product, and quadratic terms). A perfect fit has an incremental R^2 of 1.0, and no correlation results in an incremental R^2 close to 0.0. In these terms, Table 4.16a indicates that the overall regression model shows a good fit ($R^2 = 0.9489$), and the majority of the correlation is linear (incremental $R^2 = 0.7285$), with some cross-product interaction (incremental $R^2 = 0.1695$), and relatively no quadratic interaction (incremental $R^2 = 0.0509$). In addition, the higher the F-ratio, the more important the specific term is in the overall regression model. Table 4.16a gives F-ratios of 883.06, 77.51, and 58.23 for the linear, cross-product, and quadratic terms, respectively.

Turning now to the ANOVA report, Table 4.16b, we see that the impurity concentration in the feed, $x_{impurity}$, the first elevated temperature, T_1, and the autoclave pressure, P, have the three largest F-ratios of 610.9, 285.9, and 112.5, respectively. These are considerably higher than the F-ratios of 2.7, 2.2, and 1.5 for the remaining three factors. Hence, we will use $x_{impurity}$, T_1, and P as the primary factors (independent

operating parameters) affecting the composite thickness. As in Table 4.16a, the correlation coefficients, R^2, follow the same trend as the F-ratios, with higher R^2 terms (i.e., approaching 1.0) indicating greater influence.

Tables 4.17a-b show results similar to those in Table 4.16a-b, but for void size. These results reveal that T_1 and P are the two most significant factors in determining void size.

Table 4.17a-b. Response-surface analysis and ANOVA report of void size with respect to the 6 most significant operating parameters (factors).

(a) Response-surface analysis for void size

	Sequential sum of squares	Mean square	F-ratio	Incremental R^2
Regression	14.18	0.54	40.27	0.7565
Linear	7.56	1.26	95.61	0.4033
Quadratic	4.03	0.67	51.51	0.2150
Cross-product	2.59	0.17	13.24	0.1382
Total error	4.56	0.01	1,486.19	0.2435

ANOVA report for void size

Factor	Last sum of squares	Mean square	F-ratio	Term R^2
x_{resin}	0.13	0.19	1.50	0.0071
T_0	0.09	0.01	0.96	0.0047
P	7.45	1.06	81.80	0.3972
T_1	7.74	1.10	84.80	0.4129
T_2	0.09	0.01	1.04	0.0051
$x_{impurity}$	0.25	0.04	2.71	0.0132

2. Optimization of Primary Factors

We have now identified in the prediction section that the primary factors affecting the composite thickness are T_1, P, and $x_{impurity}$, and those affecting the void size are T_1, P. In addition, we have found that the secondary factors influencing the composite thickness and void size include x_{resin}, T_0, and T_2. We now demonstrate the role of neural networks in process optimization by generating a response-surface map of only the primary factors, while setting the secondary factors to their mean value (i.e., a normalized value of 0). We then determine the optimal operating conditions for the secondary factors with the primary variables set at their optimal values. Much more sophisticated optimization routines exist, which can also incorporate a neural network predictor, in order to optimize the system over the entire set of control variables at one time. Our approach, however, allows us to show the power of neural network predictors in a more understandable format.

For composite thickness, we can use the neural network to generate a three-dimensional matrix by predicting the composite thickness over the entire operating range of the primary factors, $x_{impurity}$, T_1, and P (see Table 4.18a-b). This matrix is constrained by the normalized control variables between -1 and 1, or by their standard operating conditions if the distribution is skewed (e.g., the distribution of the autoclave pressure, Figure 4.15a, has a lower operating limit of 2.0, which corresponds to -0.48). This matrix uses increments of 0.2 (e.g., -1.0, -0.8, ... , 0.8, 1.0) for pressure and first elevated temperature, and increments of 0.5 (e.g., -1.0, -0.5, ... , 1.0) for feed impurity.

Table 4.18a. The composite-thickness matrix predicted by the autoclave-processing network over the complete operating ranges of autoclave pressure, first elevated temperature, and impurity in the feed.

Impurity in feed : 0.45

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
	-0.6	1.84	1.6895	1.6893	1.6890	1.6886	1.6880	1.6873	1.6864	1.6856	1.6848	1.6848
	-0.4	2.12	1.6895	1.6894	1.6892	1.6889	1.6885	1.6879	1.6872	1.6864	1.6855	1.6850
	-0.2	2.40	1.6895	1.6895	1.6894	1.6892	1.6888	1.6884	1.6878	1.6872	1.6865	1.6859
	0.0	2.68	1.6895	1.6895	1.6895	1.6893	1.6891	1.6888	1.6884	1.6879	1.6874	1.6869
P (atm)	0.2	2.96	1.6895	1.6895	1.6895	1.6894	1.6893	1.6891	1.6888	1.6885	1.6882	1.6879
	0.4	3.24	1.6894	1.6895	1.6895	1.6895	1.6894	1.6893	1.6892	1.6890	1.6888	1.6888
	0.6	3.52	1.6894	1.6894	1.6895	1.6895	1.6895	1.6895	1.6894	1.6893	1.6894	1.6896
	0.8	3.80	1.6894	1.6894	1.6894	1.6895	1.6895	1.6895	1.6895	1.6896	1.6898	1.6903
	1.0	4.08	1.6894	1.6894	1.6894	1.6894	1.6895	1.6895	1.6896	1.6898	1.6902	1.6908

Impurity in feed : 0.70

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
	-0.6	1.84	1.6884	1.6877	1.6870	1.6862	1.6857	1.6861	1.6894	1.7013	1.7341	1.7894
	-0.4	2.12	1.6887	1.6882	1.6875	1.6868	1.6859	1.6852	1.6853	1.6882	1.7004	1.7406
	-0.2	2.40	1.6889	1.6886	1.6880	1.6874	1.6866	1.6857	1.6850	1.6852	1.6888	1.7067
	0.0	2.68	1.6891	1.6888	1.6885	1.6880	1.6873	1.6866	1.6858	1.6853	1.6862	1.6931
	0.2	2.96	1.6892	1.6890	1.6888	1.6884	1.6879	1.6873	1.6867	1.6862	1.6864	1.6894
P (atm)	0.4	3.24	1.6893	1.6892	1.6890	1.6888	1.6884	1.6880	1.6876	1.6872	1.6874	1.6891
	0.6	3.52	1.6894	1.6893	1.6892	1.6890	1.6888	1.6885	1.6883	1.6882	1.6885	1.6898
	0.8	3.80	1.6895	1.6894	1.6893	1.6892	1.6890	1.6889	1.6888	1.6890	1.6895	1.6908
	1.0	4.08	1.6896	1.6895	1.6894	1.6893	1.6892	1.6892	1.6893	1.6896	1.6904	1.6917

Impurity in feed : 0.95

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
	-0.6	1.84	1.6876	1.6904	1.6997	1.7230	1.7611	1.7988	1.8307	1.8620	1.8947	1.9258
	-0.4	2.12	1.6864	1.6864	1.6881	1.6947	1.7138	1.7509	1.7924	1.8289	1.8660	1.9041
	-0.2	2.40	1.6866	1.6859	1.6855	1.6865	1.6914	1.7081	1.7468	1.7953	1.8383	1.8806
	0.0	2.68	1.6871	1.6863	1.6856	1.6851	1.6856	1.6900	1.7076	1.7551	1.8117	1.8578
P (atm)	0.2	2.96	1.6876	1.6870	1.6862	1.6854	1.6849	1.6855	1.6910	1.7166	1.7806	1.8361
	0.4	3.24	1.6881	1.6876	1.6869	1.6862	1.6855	1.6852	1.6867	1.6969	1.7431	1.8126
	0.6	3.52	1.6885	1.6881	1.6876	1.6870	1.6864	1.6860	1.6864	1.6906	1.7138	1.7804
	0.8	3.80	1.6889	1.6885	1.6881	1.6876	1.6872	1.6869	1.6872	1.6894	1.7001	1.7423
	1.0	4.08	1.6892	1.6889	1.6886	1.6882	1.6879	1.6879	1.6882	1.6897	1.6951	1.7153

Table 4.18b. The composite-thickness matrix predicted by the autoclave-processing network over the complete operating ranges of autoclave pressure, first elevated temperature, and impurity in the feed.

Impurity in feed : 1.20

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
P (atm)	-0.6	1.84	1.7758	1.8042	1.8292	1.8550	1.8812	1.9059	1.9282	1.9479	1.9639	1.9751
	-0.4	2.12	1.7293	1.7644	1.7946	1.8200	1.8462	1.8749	1.9043	1.9318	1.9540	1.9687
	-0.2	2.40	1.6974	1.7188	1.7536	1.7866	1.8139	1.8428	1.8765	1.9114	1.9402	1.9592
	0.0	2.68	1.6869	1.6930	1.7106	1.7452	1.7826	1.8142	1.8489	1.8883	1.9229	1.9460
	0.2	2.96	1.6848	1.6856	1.6901	1.7059	1.7428	1.7869	1.8246	1.8651	1.9030	1.9296
	0.4	3.24	1.6850	1.6844	1.6849	1.6890	1.7062	1.7523	1.8029	1.8439	1.8823	1.9109
	0.6	3.52	1.6858	1.6849	1.6843	1.6849	1.6904	1.7164	1.7771	1.8251	1.8618	1.8912
	0.8	3.80	1.6866	1.6858	1.6850	1.6847	1.6864	1.6971	1.7435	1.8055	1.8426	1.8710
	1.0	4.08	1.6874	1.6867	1.6860	1.6856	1.6861	1.6907	1.7154	1.7778	1.8235	1.8512

Impurity in feed : 1.45

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
P (atm)	-0.6	1.84	1.8687	1.8933	1.9144	1.9316	1.9460	1.9586	1.9697	1.9783	1.9840	1.9873
	-0.4	2.12	1.8333	1.8585	1.8839	1.9075	1.9286	1.9473	1.9630	1.9744	1.9814	1.9852
	-0.2	2.40	1.8027	1.8246	1.8498	1.8774	1.9053	1.9315	1.9532	1.9682	1.9770	1.9817
	0.0	2.68	1.7735	1.7960	1.8184	1.8456	1.8778	1.9113	1.9396	1.9590	1.9702	1.9763
	0.2	2.96	1.7359	1.7670	1.7923	1.8176	1.8500	1.8881	1.9223	1.9460	1.9601	1.9681
	0.4	3.24	1.7024	1.7295	1.7646	1.7946	1.8257	1.8645	1.9024	1.9296	1.9464	1.9566
	0.6	3.52	1.6881	1.6994	1.7291	1.7711	1.8064	1.8433	1.8816	1.9109	1.9299	1.9421
	0.8	3.80	1.6844	1.6876	1.7011	1.7404	1.7890	1.8257	1.8614	1.8912	1.9118	1.9258
	1.0	4.08	1.6842	1.6847	1.6895	1.7119	1.7669	1.8110	1.8432	1.8714	1.8931	1.9087

The impurity in the feed is not generally a controllable parameter because we do not always have the luxury of reducing raw material variability or preselecting batches with impurity levels in our desired region. Therefore, we must optimize autoclave pressure and first elevated temperature over the entire distribution of impurity levels: $x_{impurity}$ = 0.45 to 1.45. To do this, we add a new factor to our study that measures the magnitude of the composite-thickness variability over the entire impurity range, R_L. Table 4.19 shows the composite-thicknesses variability matrix which is predicted by the autoclave-processing network.

Table 4.19. The composite-thickness variablity matrix predicted by the autoclave-processing network over the complete operating ranges of autoclave pressure and first elevated temperature.

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
P (atm)	-0.6	1.84	0.1811	0.2056	0.2274	0.2454	0.2603	0.2725	0.2832	0.2928	0.2992	0.3024
	-0.4	2.12	0.1469	0.1721	0.1964	0.2208	0.2427	0.2621	0.2777	0.2880	0.2958	0.3002
	-0.2	2.40	0.1161	0.1387	0.1643	0.1909	0.2187	0.2458	0.2682	0.2830	0.2906	0.2958
	0.0	2.68	0.0865	0.1097	0.1329	0.1606	0.1922	0.2247	0.2539	0.2737	0.2840	0.2894
	0.2	2.96	0.0511	0.0814	0.1060	0.1322	0.1651	0.2025	0.2356	0.2598	0.2737	0.2802
	0.4	3.24	0.0174	0.0451	0.0798	0.1056	0.1402	0.1793	0.2157	0.2424	0.2590	0.2678
	0.6	3.52	0.0036	0.0145	0.0448	0.0862	0.1201	0.1573	0.1952	0.2227	0.2414	0.2525
	0.8	3.80	0.0051	0.0036	0.0160	0.0556	0.1026	0.1387	0.1742	0.2022	0.2223	0.2355
	1.0	4.08	0.0054	0.0048	0.0035	0.0263	0.0807	0.1232	0.1549	0.1818	0.2030	0.2179

Another feasible method for approaching this problem is to have a separate set of process operating conditions for each set of incoming raw material properties. However, for most processes, major deviations in the operating conditions can be a problem. As a result, it is generally desirable to select operating conditions that have minimal variability with respect to raw material properties.

The void-size matrix shown in Table 4.20 is two-dimensional, with the two primary factors being pressure and first elevated temperature.

Table 4.20. The void-size matrix predicted by the autoclave-processing network over the complete operating ranges of autoclave pressure and first elevated temperature.

			T_1 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			370.1	372.5	374.9	377.3	379.7	382.1	384.5	386.9	389.3	391.7
P (atm)	-0.6	1.84	0.0309	0.0301	0.0288	0.0259	0.0360	0.1219	0.1508	0.1713	0.1811	0.1858
	-0.4	2.12	0.0288	0.0281	0.0270	0.0251	0.0215	0.0730	0.1285	0.1575	0.1735	0.1810
	-0.2	2.40	0.0252	0.0244	0.0234	0.0219	0.0193	0.0171	0.0983	0.1347	0.1609	0.1733
	0.0	2.68	0.0200	0.0192	0.0182	0.0170	0.0152	0.0122	0.0215	0.1038	0.1409	0.1620
	0.2	2.96	0.0141	0.0133	0.0125	0.0116	0.0105	0.0088	0.0060	0.0508	0.1119	0.1468
	0.4	3.24	0.0089	0.0083	0.0078	0.0072	0.0066	0.0057	0.0043	0.0035	0.0771	0.1253
	0.6	3.52	0.0052	0.0048	0.0045	0.0042	0.0039	0.0035	0.0030	0.0018	0.0110	0.0949
	0.8	3.80	0.0028	0.0026	0.0025	0.0023	0.0022	0.0021	0.0020	0.0016	0.0006	0.0447
	1.0	4.08	0.0014	0.0013	0.0013	0.0012	0.0012	0.0012	0.0013	0.0013	0.0009	0.0018

These matrices reveal that the minimal void size, D_b, and the minimal composite-thicknesses variability, R_L, within the given specifications occur at high

autoclave pressure, P, and low first elevated temperature, T_1. Therefore, we target the initial process design in this region. However, to prevent the process from drifting into the so-called outlying areas where we do not have it well-characterized or modeled, we decrease the autoclave pressure and increase the first elevated temperature slightly from these optimal values. We set autoclave pressure and first elevated temperature to 3.8 atm and 372.5 K, respectively. At these operating conditions, the composite thickness remains within product specifications (1.675 cm < L < 1.75 cm) at all feed impurity levels. These process settings also provide a buffer zone between the product falling outside of specifications and the process operating in regions that have not been modeled by the network.

3. Optimization of Secondary Factors

The other two main processing variables that we can control are the initial autoclave temperature, T_0, and the second elevated temperature, T_2 (Note that we will ignore the two additional raw material properties, initial void size and weight fraction of resin, in this study due to their minimal influence on the response variables as shown in the response-surface analysis and ANOVA reports of Tables 4.16a-b and 4.17a-b). We can generate a response-surface profile for the initial temperature and second elevated temperature in the same way as we did for the autoclave pressure and first elevated temperature. In generating this profile, we set the autoclave pressure and the first elevated

temperature at their previously determined optimal values, and set the impurity level at its mean for the secondary optimization. Table 4.21 shows the composite-thickness matrix and the void-size matrix for the autoclave-processing network.

Table 4.21. The composite-thickness matrix predicted by the autoclave-processing network over the complete operating ranges of autoclave pressure, first elevated temperature, and impurity in the feed.

Composite Thickness (cm)

			T_2 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			430.9	432.9	434.9	436.9	438.9	440.9	442.9	444.9	446.9	448.9
T_0 (K)	-0.6	292.5	1.6883	1.6883	1.6883	1.6883	1.6883	1.6883	1.6882	1.6882	1.6882	1.6881
	-0.4	293.7	1.6885	1.6885	1.6884	1.6884	1.6884	1.6884	1.6883	1.6883	1.6882	1.6881
	-0.2	294.9	1.6886	1.6886	1.6886	1.6885	1.6885	1.6885	1.6884	1.6883	1.6883	1.6882
	0.0	296.1	1.6887	1.6887	1.6887	1.6886	1.6886	1.6885	1.6885	1.6884	1.6883	1.6882
	0.2	297.3	1.6889	1.6888	1.6888	1.6887	1.6887	1.6886	1.6885	1.6885	1.6884	1.6883
	0.4	298.5	1.6890	1.6890	1.6889	1.6888	1.6888	1.6887	1.6886	1.6885	1.6884	1.6883
	0.6	299.7	1.6891	1.6891	1.6890	1.6889	1.6889	1.6888	1.6887	1.6886	1.6884	1.6883
	0.8	300.9	1.6893	1.6892	1.6891	1.6890	1.6889	1.6888	1.6887	1.6886	1.6885	1.6884
	1.0	302.1	1.6894	1.6893	1.6892	1.6891	1.6890	1.6889	1.6888	1.6887	1.6885	1.6884

Void size (cm)

			T_2 (K)									
			-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8
			430.9	432.9	434.9	436.9	438.9	440.9	442.9	444.9	446.9	448.9
T_0 (K)	-0.6	292.5	0.0002	0.0002	0.0003	0.0004	0.0005	0.0006	0.0008	0.0010	0.0012	0.0014
	-0.4	293.7	0.0001	0.0002	0.0003	0.0004	0.0005	0.0006	0.0008	0.0009	0.0012	0.0014
	-0.2	294.9	0.0000	0.0001	0.0002	0.0003	0.0004	0.0006	0.0007	0.0009	0.0011	0.0014
	0.0	296.1	0.0000	0.0000	0.0002	0.0003	0.0004	0.0005	0.0007	0.0009	0.0011	0.0014
	0.2	297.3	0.0000	0.0000	0.0001	0.0002	0.0004	0.0005	0.0007	0.0009	0.0011	0.0013
	0.4	298.5	0.0000	0.0000	0.0001	0.0002	0.0003	0.0005	0.0007	0.0008	0.0011	0.0013
	0.6	299.7	0.0000	0.0000	0.0000	0.0002	0.0003	0.0005	0.0006	0.0008	0.0010	0.0013
	0.8	300.9	0.0000	0.0000	0.0000	0.0002	0.0003	0.0004	0.0006	0.0008	0.0010	0.0013
	1.0	302.1	0.0000	0.0000	0.0000	0.0001	0.0003	0.0004	0.0006	0.0008	0.0010	0.0013

There are minimal variations in the composite thickness (±0.0013 cm) over the entire range of initial temperature and second elevated temperature. All composite thicknesses within this operating range are well within the specifications. Therefore, we can choose our process operating conditions based on the void-size responses. The void-size matrix shows that a high initial temperature and a low second elevated temperature are most desirable, yielding a void size approaching 0. Consequently, we set the initial temperature and second elevation temperature at 300 K and 435 K, respectively.

4. Overview of Autoclave-Processing Results

Figure 4.18 shows a graphical representation of the autoclave's operating conditions, where the pressure is 3.8 atm and initial temperature, T_0, is set at 300 K, elevated to 372.5 K at t_1 (predetermined by the composite manufacturing engineer), and further elevated to 435 K at t_3 (predetermined). We set these values based on targeting the composite thickness at 1.69 cm, which is in the lower range of the specifications, $1.675 < L < 1.75$. We selected the target in this range because it will reduce the batch-to-batch variability with respect to raw material properties. Process conditions set for this composite-thickness target, simultaneously minimize the void size.

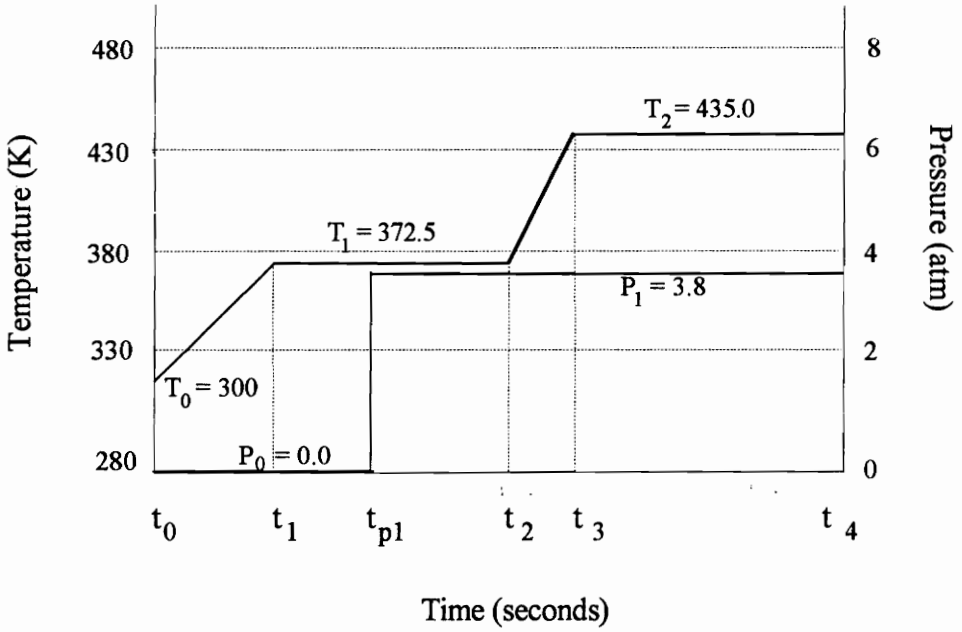


Figure 4.18. The temperature and pressure profiles in the curing cycle for the autoclave process determined using the neural network predictor in conjunction with response-surface analysis.

In summary, this case study demonstrates how to combine neural networks with statistical methods in order to predict and optimize the product qualities for an important manufacturing operation, the autoclave curing process for making composite materials.

4.4 Chapter Summary

- Using 2- and 3-hidden-layer neural networks, rather than single-hidden-layer ones, significantly improves the prediction capability. An effective initial configuration uses 30 nodes in the first hidden layer and 15 nodes in the second hidden layer.
- Neural networks have been very effective in predicting and optimizing performance data from processes and analytical instrumentation that are complex and ill-defined by first principles.
- The first case study shows how neural networks function as an effective alternative to nonlinear regression models. Specifically, we predict the reaction rate for the isomerization of n-pentane from the partial pressures of hydrogen, n-pentane, and isopentane.
- The second case study shows how neural networks function in the development of intelligent process sensors, in serving as "software sensors" (i.e., "soft sensors") to facilitate reliable and accurate interpretations of sensor signals. We use an example that quantitatively predicts product compositions from the fluorescent spectra of a binary mixture of two amino acids, tyrosine and tryptophan.
- The final case study combines neural networks with statistical methods to predict and optimize performance variables in the manufacture of composite materials by the autoclave curing process.

Nomenclature

D_b	:	the diameter of gas bubble which is assumed spherical (void size).
$f(x,\theta)$	:	expectation function in the nonlinear regression model
I_i	:	intensity of fluorescent spectra at wavelength i.
L	:	final composite thickness.
p_i	:	partial pressure of component i.
P_0	:	vacuum pressure in the autoclave process.
P_1	:	applied pressure in the autoclave process.
r_i	:	reaction rate of component i.
T_0	:	initial autoclave temperature.
T_1	:	first elevated temperature in the autoclave process.
T_2	:	second elevated temperature in the autoclave process.
t_i	:	predetermined times for temperature setpoint changes in autoclave process.
t_{pl}	:	time at which pressure is applied in the autoclave process.
W_m	:	final resin weight fraction in the autoclave process.
x_i	:	the 3 independent variables of the nonlinear regression model.
x_{resin}	:	initial resin weight fraction in laminate (autoclave curing process).
$x_{tyrosine}$	:	mole fraction of tyrosine.
α	:	the conversion rate of reaction (extent of curing) in the autoclave process.
θ_i	:	the adjustable parameters used to fit the nonlinear regression model.

Practice Problems

(4.1) Generate a neural network model for prediction of the kinematic viscosity of a lubricant as a function of temperature and pressure, data obtained from Bates and Watts (1988) and Linssen (1975). The proposed nonlinear model for this problem is:

$$f(x,\theta) = \frac{\theta_1}{\theta_2 + x_1} + \theta_3 x_2 + \theta_4 x_2^2 + \theta_5 x_2^3 + \left(\theta_6 + \theta_7 x_2^2\right) x_2 \exp\left[\frac{-x_1}{\theta_8 + \theta_9 x_2^2}\right]$$

In the equation, x_1 and x_2 represent the temperature and pressure, respectively; θ_j (j = 1 to 9) denote nine empirical parameters to be determined by nonlinear regression analysis. $f(x,\theta)$ is the kinematic viscosity.

The data set for the prediction of kinematic viscosity is given in Table 4.P1 and the training data are in file *visc.nna*.

(a) Train and test a backpropagation network (specifications in Table 4.4) for several one- and two hidden-layer configurations using file *visc.nna*. Generate a table of average error as a function of the number of nodes in the hidden layers similar to Table 4.5. Identify the optimal network structure for recall of training data only.

(b) Generate a learning curve for both the best 1- and 2-hidden-layer networks (see Table 4.6 and Figure 4.4). Compare the generalization of network with both 1 and 2 hidden layers.

Table 4.P1. Data for the prediction of lubricant viscosity in practice problem 4.1 (file: *visc.nna*).

T = 0 °C		T = 25 °C	
Pressure (atm)	ln[viscosity]	Pressure (atm)	ln[viscosity]
1.0	5.106	1.0	4.542
740.8	6.387	805.0	5.825
1407.5	7.385	1505.9	6.705
363.2	5.791	2340.0	7.716
1.0	5.107	422.9	5.298
805.5	6.361	1168.4	6.226
1868.1	7.973	2237.3	7.574
3285.1	10.473	4216.9	10.354
3907.5	11.927	5064.3	11.984
4125.5	12.426	5280.9	12.444
2572.0	9.156	3647.3	9.523
		2813.9	8.345

T = 37.8 °C		T = 98.9 °C	
Pressure (atm)	ln[viscosity]	Pressure (atm)	ln[viscosity]
516.8	5.173	1.0	3.381
1738.0	6.650	686.0	4.458
1008.7	5.807	1423.6	5.207
2749.2	7.741	2791.4	6.291
1375.8	6.232	4213.4	7.327
191.1	4.661	2103.7	5.770
1.0	4.298	402.2	4.088
2922.9	7.967	1.0	3.374
4044.6	9.342	2219.7	5.839
4849.8	10.511	3534.8	6.726
5605.8	11.822	4937.7	7.768
6273.9	13.068	6344.2	8.914
3636.7	8.804	7469.4	9.983
1949.0	6.855	5640.9	8.323
1298.5	6.119	4107.9	7.132

(4.2) Develop a neural network to model the output concentration of two continuous stirred-tank reactors (CSTRs) in series (Bhagat, 1990). The dimensionless outlet concentration profile (represented as 15 discrete concentrations at specified times) depends on the flowrate through the CSTRs. In this problem, we use 15 concentrations to predict a corresponding flowrate. Table 4.P2 lists the four patterns for training (file: *conctrn.nna*) and two patterns (file: *conctst.nna*) for testing the network. Figure 4.P1 shows the graphical representation of the training and testing data sets.

Table 4.P2. Training and testing data (files: *conctrn.nna and conctst.nna*) for the prediction of flowrate that corresponds to a dimensionless outlet concentration profile of two CSTRs in series (Bhagat, 1990).

	Training data				Testing data	
Time	pattern 1	pattern 2	pattern 3	pattern 4	pattern 5	pattern 6
15	0.011	0.040	0.133	0.189	0.073	0.257
20	0.019	0.065	0.204	0.283	0.056	0.267
25	0.028	0.095	0.279	0.376	0.282	0.548
30	0.038	0.127	0.354	0.463	0.144	0.562
40	0.063	0.198	0.492	0.615	0.408	0.735
50	0.093	0.272	0.610	0.731	0.457	0.820
60	0.125	0.345	0.706	0.816	0.499	0.971
70	0.159	0.417	0.782	0.876	0.546	0.903
80	0.195	0.484	0.840	0.917	0.711	0.957
90	0.231	0.546	0.883	0.946	0.745	1.054
100	0.268	0.602	0.916	0.965	0.792	0.936
125	0.359	0.720	0.964	0.988	0.937	1.022
150	0.446	0.807	0.985	0.996	0.999	1.025
200	0.598	0.912	0.997	1.000	0.923	0.922
250	0.716	0.962	1.000	1.000	0.970	0.917
Ouput flowrate (m^3/s)	0.01	0.02	0.04	0.05	0.03	0.06

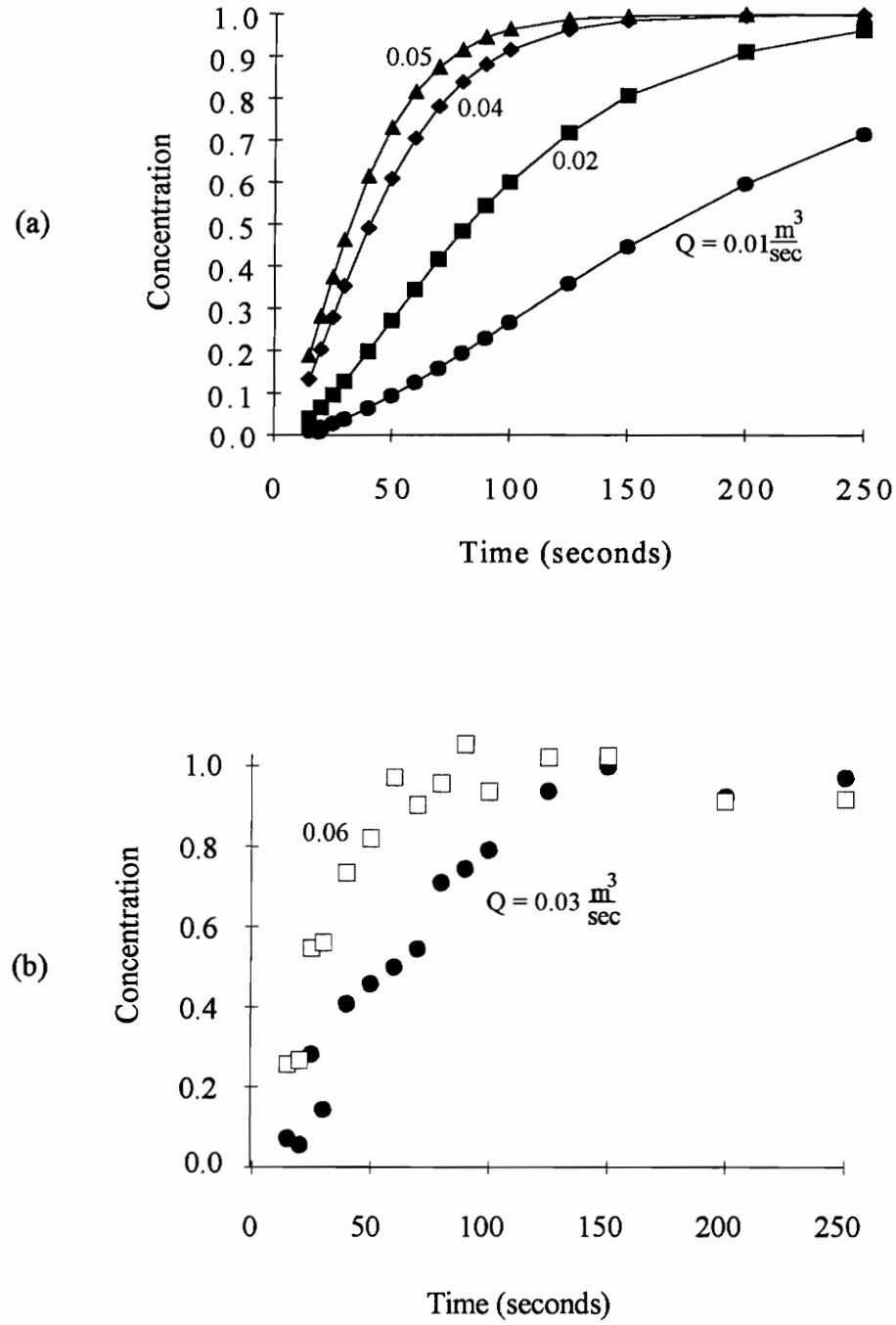


Figure 4.P1. Graphical representations of (a) the training data set and (b) the testing data set for the prediction of flowrate that corresponds to a dimensionless outlet concentration profile of two CSTRs in series (Bhagat, 1990).

(a) Train the neural network, using the four patterns in file *conctrn.nna*, for a network that has 7 nodes in the hidden layer.

(b) Test the network with file *conctst.nna* that has random noise added (up to $\pm$ 0.1). Compare the results to the expected flowrates of 0.03 and 0.06 m^3/sec for patterns 5 and 6, respectively. How well does the network filter the noise?

(4.3) Develop neural network models for the prediction of a multireponse system that predicts foaming properties of whey protein concentrates (Khuri and Cornell, 1987). The product quality parameters (output responses) are Y_1 = whipping time (min) (the total elapsed time required to produce peaks of foam formed during whipping of a liquid sample), Y_2 = maximum overrun (%) [overrun is determined by weighting 5 oz. paper cups of foam and unwhipped liquid sample and calculating the expression: % overrun = 100*(weight of liquid-weight of foam)/weight of foam], and Y_3 = percent soluble protein (%). The five input variables are X_1 = heating temperature (°C), X_2 = pH level, X_3 = redox potential (volt), X_4 = sodium oxalate (molar), and X_5 = sodium lauryl sulfate (%).

Table 4.P3 lists the data (file: *foam.nna*) obtained using a central composite design, a common experimental design in response-surface modeling. We define the normalized input and output variables, x_i and y_j, using the zero-mean normalization method (see Section 2.3.B) with the following equations:

$$x_1 = \frac{X_1 - 75}{10} \quad x_2 = \frac{X_2 - 6}{2} \quad x_3 = \frac{X_3 - 0.175}{0.2} \quad x_4 = \frac{X_4 - 0.025}{0.025} \quad x_5 = \frac{X_5 - 0.10}{0.10}$$

and

$$y_1 = \frac{Y_1 - 5}{5} \qquad y_2 = \frac{Y_2 - 1050}{350} \qquad y_3 = \frac{Y_3 - 85}{35}$$

Table 4.P3. The data (file: *foam.nna*) for predicting foam properties of whey protein concentrates (Khuri and Cornell, 1987).

Input variables					Normalized input variables					Output			Normalized output		
X_1	X_2	X_3	X_4	X_5	x_1	x_2	x_3	x_4	x_5	Y_1	Y_2	Y_3	y_1	y_2	y_3
75	6	0.180	0.0250	0.10	0.0	0.0	0.0	0.0	0.0	3.50	1179	104	-0.300	0.369	0.543
75	6	0.175	0.0250	0.10	0.0	0.0	0.0	0.0	0.0	3.50	1183	107	-0.300	0.380	0.629
75	6	0.175	0.0250	0.10	0.0	0.0	0.0	0.0	0.0	4.00	1120	104	-0.200	0.200	0.543
75	6	0.175	0.0250	0.10	0.0	0.0	0.0	0.0	0.0	3.50	1180	101	-0.300	0.371	0.457
75	6	0.175	0.0250	0.10	0.0	0.0	0.0	0.0	0.0	3.00	1195	103	-0.400	0.414	0.514
70	5	0.075	0.0125	0.15	-0.5	-0.5	-0.5	-0.5	0.5	4.75	1082	81.4	-0.050	0.091	-0.103
80	5	0.075	0.0125	0.05	0.5	-0.5	-0.5	-0.5	-0.5	4.00	824	69.6	-0.200	-0.646	-0.440
70	7	0.075	0.0125	0.05	-0.5	0.5	-0.5	-0.5	-0.5	5.00	953	105	0.000	-0.277	0.571
80	7	0.075	0.0125	0.15	0.5	0.5	-0.5	-0.5	0.5	9.50	759	81.2	0.900	-0.831	-0.109
70	5	0.275	0.0125	0.05	-0.5	-0.5	0.5	-0.5	-0.5	4.00	1163	80.8	-0.200	0.323	-0.120
80	5	0.275	0.0125	0.15	0.5	-0.5	0.5	-0.5	0.5	5.00	839	76.3	0.000	-0.603	-0.249
70	7	0.275	0.0125	0.15	-0.5	0.5	0.5	-0.5	0.5	3.00	1343	103	-0.400	0.837	0.514
80	7	0.275	0.0125	0.05	0.5	0.5	0.5	-0.5	-0.5	7.00	736	76.9	0.400	-0.897	-0.231
70	5	0.075	0.0375	0.05	-0.5	-0.5	-0.5	0.5	-0.5	5.25	1027	87.2	0.050	-0.066	0.063
80	5	0.075	0.0375	0.15	0.5	-0.5	-0.5	0.5	0.5	5.00	836	74	0.000	-0.611	-0.314
70	7	0.075	0.0375	0.15	-0.5	0.5	-0.5	0.5	0.5	3.00	1272	98.5	-0.400	0.634	0.386
80	7	0.075	0.0375	0.05	0.5	0.5	-0.5	0.5	-0.5	6.50	825	94.1	0.300	-0.643	0.260
70	5	0.275	0.0375	0.15	-0.5	-0.5	0.5	0.5	0.5	3.25	1363	95.9	-0.350	0.894	0.311
80	5	0.275	0.0375	0.05	0.5	-0.5	0.5	0.5	-0.5	5.00	855	76.8	0.000	-0.557	-0.234
70	7	0.275	0.0375	0.05	-0.5	0.5	0.5	0.5	-0.5	2.75	1284	100	-0.450	0.669	0.429
80	7	0.275	0.0375	0.15	0.5	0.5	0.5	0.5	0.5	5.00	851	104	0.000	-0.569	0.543
65	6	0.175	0.0250	0.10	-1.0	0.0	0.0	0.0	0.0	3.75	1283	100	-0.250	0.666	0.429
85	6	0.175	0.0250	0.10	1.0	0.0	0.0	0.0	0.0	11.00	651	50.5	1.200	-1.140	-0.986
75	4	0.175	0.0250	0.10	0.0	-1.0	0.0	0.0	0.0	4.50	1217	71.2	-0.100	0.477	-0.394
75	8	0.175	0.0250	0.10	0.0	1.0	0.0	0.0	0.0	4.00	982	101	-0.200	-0.194	0.457
75	6	-0.025	0.0250	0.10	0.0	0.0	-1.0	0.0	0.0	5.00	884	85.8	0.000	-0.474	0.023
75	6	0.375	0.0250	0.10	0.0	0.0	1.0	0.0	0.0	3.75	1147	103	-0.250	0.277	0.514
75	6	0.175	0.0000	0.10	0.0	0.0	0.0	-1.0	0.0	3.75	1081	104	-0.250	0.089	0.543
75	6	0.175	0.0500	0.10	0.0	0.0	0.0	1.0	0.0	4.75	1036	89.4	-0.050	-0.040	0.126
75	6	0.175	0.0250	0.00	0.0	0.0	0.0	0.0	-1.0	4.00	1213	105	-0.200	0.466	0.571
75	6	0.175	0.0250	0.20	0.0	0.0	0.0	0.0	1.0	3.50	1103	113	-0.300	0.151	0.800

(a) Train the network as one network with three output nodes (file: *foam.nna*) and as three independent networks, having a network to predict whipping time (y_1), maximum overrun (y_2), and percent soluble protein (y_3) independently (files: *foama.nna*, *foamb.nna*, and *foamc.nna*). Compare the prediction capability of both network architectures over a range of one- and two-hidden-layer configurations. Identify the optimal hidden-layer configuration.

(b) Identify the two input variables that have the greatest effect on each of the three response output variables independently. To generate a matrix or response-surface plot of an output variable as a function of its two primary input variables, create a data file that spans the entire operating range of the both input variables (e.g., x_1 and x_2 both ranging from -1 to 1 with increments of 0.25). Present this data file to the network and record the accompanying network predictions. Use these three prediction matrices to generate a response-surface plot of each output variable with respect to their two primary input variables.

(4.4) Develop a neural network to evaluate washing treatments for quality improvement of minced mullet flesh (Khuri and Cornell, 1987). The effects of three input variables are analyzed: X_1 = washing temperature ($^{\circ}C$), X_2 = washing time (min), and washing ratio (X_3). There are four response variables we wish to optimize:

Y_1 = springiness (mm), Y_2 = thiobarbituric acid (TBA) number, Y_3 = percent cooking loss (%), and Y_4 = whiteness index.

Table 4.P4 lists the data (file: *wash.nna*) for training the network. We normalized the input and output variables, x_i and y_j, using the zero-mean normalization method (see Section 2.3.B) with the following equations:

$$x_1 = \frac{X_1 - 33}{11.8} \qquad x_2 = \frac{X_2 - 5.5}{4.5} \qquad x_3 = \frac{X_3 - 22.5}{7.6}$$

and

$$y_1 = \frac{Y_1 - 1.65}{0.3} \qquad y_2 = \frac{Y_2 - 35}{15} \qquad y_3 = \frac{Y_3 - 22.5}{7.5} \qquad y_4 = \frac{Y_4 - 55}{15}$$

(a) Develop an effective neural network model for each of the four output responses as a function of the three input variables.

(b) Determine the optimal operating conditions of the input variables to simultaneously maximize springiness (Y_1) and whiteness index (Y_4), while minimizing TBA (Y_2) and percent cooking loss (Y_3). Use the neural network models to generate prediction matrices and response-surface plots to support your conclusions.

Table 4.P4. The data (file: *wash.nna*) for evaluating wash treatments of minced mullet flesh (Khuri and Cornell, 1987).

Input variables			Normalized input variables			Output variables				Normalized output variables			
X_1	X_2	X_3	x_1	x_2	x_3	Y_1	Y_2	Y_3	Y_4	y_1	y_2	y_3	y_4
26	2.8	18	-0.6	-0.6	-0.6	1.83	29.31	29.50	50.36	0.600	-0.379	0.933	0.357
40	2.8	18	0.6	-0.6	-0.6	1.73	39.32	19.40	48.16	0.267	0.288	-0.413	0.211
26	8.2	18	-0.6	0.6	-0.6	1.85	25.16	25.70	50.72	0.667	-0.656	0.427	0.381
40	8.2	18	0.6	0.6	-0.6	1.67	40.81	27.10	49.69	0.067	0.387	0.613	0.313
26	2.8	27	-0.6	-0.6	0.6	1.86	29.82	21.40	50.09	0.700	-0.345	-0.147	0.339
40	2.8	27	0.6	-0.6	0.6	1.77	32.20	24.00	50.61	0.400	-0.187	0.200	0.374
26	8.2	27	-0.6	0.6	0.6	1.88	22.01	19.60	50.36	0.767	-0.866	-0.387	0.357
40	8.2	27	0.6	0.6	0.6	1.66	40.02	25.10	50.42	0.033	0.335	0.347	0.361
21.2	5.5	22.5	-1.0	0.0	0.0	1.81	33.00	24.20	29.31	0.533	-0.133	0.227	-1.046
44.8	5.5	22.5	1.0	0.0	0.0	1.37	51.59	30.60	50.67	-0.933	1.106	1.080	0.378
33	1	22.5	0.0	-1.0	0.0	1.85	20.35	20.90	48.75	0.667	-0.977	-0.213	0.250
33	10	22.5	0.0	1.0	0.0	1.92	20.53	18.90	52.70	0.900	-0.965	-0.480	0.513
33	5.5	14.9	0.0	0.0	-1.0	1.88	23.85	23.00	50.19	0.767	-0.743	0.067	0.346
33	5.5	30.1	0.0	0.0	1.0	1.90	20.16	21.20	50.86	0.833	-0.989	-0.173	0.391
33	5.5	22.5	0.0	0.0	0.0	1.89	21.72	18.50	50.84	0.800	-0.885	-0.533	0.389
33	5.5	22.5	0.0	0.0	0.0	1.88	21.21	18.60	50.93	0.767	-0.919	-0.520	0.395
33	5.5	22.5	0.0	0.0	0.0	1.87	21.55	16.80	50.98	0.733	-0.897	-0.760	0.399

References and Further Reading

Section 4.1 provides representative references for applications of neural networks to a variety of prediction and optimization problems in bioprocessing, chemical engineering, analytical chemistry, materials processing, and mineral and metallurgical engineering, etc. Those of particular interest include Cherian et al. (1991), Himmel and May (1993), Gill and Shutt (1992), Joseph and Wang (1993), Long et al. (1990), May (1994), McAvoy et al. (1992), Meyers et al. (1991), Piovoso and Owen (1991), Qian and Seijnowski (1990). For additional background on statistical methods that are useful for optimization applications in conjunction with neural network predictors, we recommend the texts by Himmelblau (1970) and Box et al. (1978).

Agamenroni, O., C. Chessari, J. A. Romagnoli, G. W. Barton and K. Bourke, "A Neural-Network-Based Prediction Scheme for an Industrial Propathene Reactor," *World Congress of Neural Networks*, San Diego, CA, June (1994).

Anker, L. S. and P. C. Jurs, "Prediction of Carbon-13 Nuclear Magnetic Resonance Chemical Shifts by Artificial Neural Networks," *Anal. Chem.*, **64**, 1157 (1992).

Bates, D. M. and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*, Wiley Interscience, New York (1988).

Bhagat, P., "An Introduction to Neural Nets," *Chemical Engineering Progress*, **86**. No.8, 55, August (1990).

Bhat, N. V., P. A. Minderman, Jr., T. J. McAvoy and N. S. Wang, "Modeling Chemical Process Systems via Neural Computation," *IEEE Control Systems Magazine*, **10,** 24, April (1990).

Bhat, N. V. and T. J. McAvoy, "Determining Model Structure Using Neural Models by Network Stripping," *Comput. Chem. Eng.*, **16**, 271 (1992).

Bodor, N., A. Harget, and M. Huang, "Neural Network Studies. 1. Estimation of the Aqueous Solubility of Organic Compounds," *J. Amer. Chem. Soc.*, **113**, 9480-9483 (1991).

Box, G. E., W. G. Hunter and J. S. Hunter, *Statistics for Experiments*, Wiley, New York (1978).

Bulsari, A. B. and H. Saxen, "Artificial Neural Networks for Predicting Silicon Content in Raw Iron from Blast Furnaces," pp. 642-644, in *Lecture Notes in Computer Science*, Vol. 497, Springer-Verlag, New York (1991).

Bulsari, A. B., A. Krastawski and H. Saxen, "Implementing a Fuzzy Expert System in an Artificial Neural Network," *Comput. Chem. Eng.*, **16**, s405 (1992).

Campmajó, C., M. Poch, J. Robusté, F. Valero and J. Lafuente, "Evaluation of Artificial Neural Networks for Modeling Enzymatic Glucose Determination by Flow Injection Analysis," *Analysis*, **20**, 127 (1992).

Carr, N. L., "Kinetics of Catalytic Isomerization of n-Pentane," *Industrial and Engineering Chemistry*, **52**, 391 (1960).

Cherian, S. S., T. J. Anderson and S. A. Svoronos, "Application of a Neural Network Approach to the Sequential Optimization of MOCVD Indian Phosphate Growth," *AIChE Annual Meeting*, Los Angeles, CA, Nov. (1991).

Cheng, T. F. and J. I. Elsey, "Building Empirical Models fo Process Plant Data by Regression or Neural Network," *Proceedings of Amer. Control Conf.*, p. 1922, Chicago, IL, June (1992).

Depeyre, D., A. Isambert, P. H. Prevost, C. Donadille and R. Perisse, "Simulation of Steel Transformation Curves with a Neural Network," *Comput. Chem. Eng.*, **16**, 5417 (1992).

Dong, D. and T. J. McAvoy, "Nonlinear Principal Component Analysis Based on Principal Curves and Neural Networks," *Comput. Chem. Eng.*, in press (1994).

Geladi, P. and B. R. Kowalski, "Partial Least-Squares Regression: A Tutorial," *Analyt. Chem. Acta.*, **185**, 1 (1986a).

Geladi, P. and B. R. Kowalski, "An Example of Two-Block Predictive Partial Least-Squares Regression with Simulated Data," *Analyt. Chem. Acta.*, **185**, 19 (1986b).

Gemperline, F. J., J. R. Long and V. G. Gregoriou, "Nonlinear Multivariate Calibration Using Principal Component Regression and Artificial Neural Networks," *Anal. Chem.*, **63**, 2313 (1991).

Gill, T. and J. Shutt, "Optimizing Product Formulations Using Neural Networks," *Scientific Computation and Automation*, Sept. (1992).

Haaland, D. M. and E. V. Thomas, "Partial Least-Squares Methods for Spectral Analysis. 1. Relation to Other Quantitative Calibration Methods and the Extraction of Qualitative Information," *Anal. Chem.*, **60**, 1193 (1988).

Himmel, C. D. and G. S. May, "Advantages of Plasma Etch Modeling Using Neural Networks over Statistical Techniques," *IEEE Trans. Semicond. Manufact.*, **6**, 103 (1993).

Himmelblau, D. M., *Process Analysis by Statistical Methods*, Wiley, New York (1970).

Holcomb, T. R. and M. Morari, "PLS/Neural Networks," *Comput. Chem. Eng.*, **16**, 393 (1992).

Huang, B. and A. S. Mujumdar, "Use of Neural Networks to Predict Industrial Dryer Performance," *Drying Technology*, **11**, 525 (1993).

Huang, Y. L., T. F. Edgar, D. M. Himmelblau and I. Tratchtenberg, "Constructing a Reliable Neural Network Model for a Plasma Etching Process Using Limited Experimental Data," *IEEE Trans. Semicond. Manufac.*, in press (1994).

Jackson J. and G. Mudholkar, "Control Procedures for Residuals Associated with Principal Component Analysis," *Technometrics*, **21**, 341 (1979).

Jolliffe, I. T., *Principal Component Analysis*, Springer-Verlag, NY (1986).

Joseph, B., F. H. Wang and D. S. S. Shieh, "Exploratory Data Analysis: A Comparison of Statistical Methods with Artificial Neural Networks," *Comput. Chem. Eng.*, **16**, 413 (1992).

Joseph, B. and F. H. Wang, "Predictive Control of Quality in a Batch Manufacturing Process Using Artificial Neural Network Models," *Ind. Eng. Chem. Res.*, **32**, 1951 (1993).

Khuri, A. I. and J. A. Cornell, *Response Surfaces : Designs and Analysis*, Marcel Dekker, Inc., New York (1987).

Kier, L. B., "A Shape Index from Molecular Graphs," *Quant. Struct.-Act. Relat.*, **4**, 109 (1985).

Kier, L. B., *Molecular Connectivity in Structure-Activity Analysis*, Research Studies Press, Letchworth, England, (1986).

Kito, S., T. Hattori and Y. Murakami, "Estimation of the Acid Strength of Mixed Oxides by a Neural Network," *Ind. Eng. Chem. Res.*, **31**, 979 (1992).

Kramer, M. A., M. L. Thompson and P. M. Bhagat, "Embedding Theoretical Models in Neural Networks," *Proceedings of the Amer. Control Conf.*, p. 475, Chicago, IL, June (1992).

Kresta, J. V., J. F. MacGregor and T. E. Marlin, "Multivariate Statistical Monitoring of Process Operation Performance," *Can. J. Chem. Eng.*, **69**, No. 2, 35 (1991).

Lee, M. J. and J. T. Chen, "Fluid Property Predictions with the Aid of Neural Networks," *Ind. Eng. Chem. Res.*, **32**, 995 (1993).

Leonard, J. A., M. A. Kramer and L. H. Ungar, "A Neural Network Architecture that Computes its Own Reliability," *Comput. Chem. Eng.*, **16**, 819 (1992).

Linssen, H. N., "Nonlinearity Measures: A Case Study," *Statistica Neerlandica*, **29**, 93-99 (1975).

Liu, Q., S. Hirono and I. Moriguchi, "Application of Functional-Link Net in Qualitative Structure-Activity Relationships. 1. QSAR for Activity Data Given by Continuous Variate; 2. QSAR for Activity Data Given by Ratings," *Quant. Struct.-Act. Relat.*, **11**, 135 and 318 (1992a).

Liu, Q., S. Hirono and I. Moriguchi, "Comparison of the Functional-Link Net and the Generalized Delta Rule in Quantitative Structure-Activity Relationship Studies," *Chem. Pharm. Bull.*, **40**, 2962 (1992b).

Long, J. R., V. G. Gregoriou and P. J. Gemperline, "Spectroscopic Calibration and Quantitation Using Artificial Neural Networks," *Anal. Chem.*, **62**, 1791 (1990).

Long, J. R., H. T. Mayfield and M. V. Henley, "Pattern Recognition of Jet Fuel Chromatographic Data by Artificial Neural Networks with Backpropagation of Error," *Anal. Chem.*, **63**, 1256 (1991).

MacGregor, J., T. Marlin, J. Kresta and B. Skagerberg, "Multivariate Statistical Methods in Process Analysis and Control," *Proceedings Chem. Process Control, IV*, Y. Akrun and W. H. Ray, editors, p. 79, CACHE Corporation, Austin, TX (1991).

MacGregor, J. F., C. Jaeckle, C. Kiparissides and M. Koutoudi, "Process Monitoring and Diagnosis by Multiblock PLS Methods," *AIChE J.*, **40**, 826 (1994).

Madron, F., *Process Plant Performance : Measurement and Data Processing for Optimization and Retrofits*, Ellis Horwood Ltd., Chichester, West Sussex, United Kingdom (1992).

May, G. S., "Manufacturing ICs the Neural Way," *IEEE Spectrum*, **31**, No. 9, 47 (1994).

McAvoy, T. J., H. T. Su, N. S. Wang, M. He, J. Horvath and H. Semerjian, "A Comparison of Neural Networks and Partial Least Squares for Deconvoluting Florescent Spectra," *Biotech. and Bioeng.*, **40**, 53 (1992).

Meyers, B., T. Hansen, D. Nute, P. Albersheim, A. Darrill, W. York and J. Sellers, "Identification of the ^{1}H-NMR Spectra of Complex Oligosaccharides with Artificial Neural Networks," *Science*, **251**, 542 (1991).

Mocella, M. T., Bondur, J. A. and T. R. Turner, "Etch Process Characterization Using Neural Network Methodology: A Case Study," *SPIE Proceedings of Process Module Metrology, Control, and Clustering*, **1594**, 232 (1992).

Normandin, A., B. P. A. Grandjean and J. Thibault, "PVT Data Analysis Using Neural Network Models," *Ind. Eng. Chem. Res.*, **32**, 970 (1993).

Oishi, K., M. Tominaga, A. Kawato and S. Imayasu, "Analysis of the State Characteristics of Sake Brewing with a Neural Network," *J. Ferm. and Bioeng.*, **73**, 153 (1992).

Peterson, K. L., "Counterpropagation Neural Networks in the Modeling and Prediction of Korats Indices for Substituted Phenols," *Anal. Chem.*, **64**, 379 (1992).

Phillips, J. A., " Spectroscopic Sensors for Bioprocessing," *AIChE Symp. Series*, **85**, No. 267, 67 (1989).

Pillai, V. K., A. N. Beris and P. S. Dharjati, " Implementation of Model-Based Optimal Temperature Profiles for Autoclave Curing of Composites Using a Knowledge-Based System," *Ind. Eng. Chem. Res.*, **32**, 2443 (1994).

Piovoso, M. J. and A. J. Owen, "Sensor Data Analysis Using Artificial Neural Networks," pp. 101-118, in *Chemical Process Control IV*, Y. Arkun and W. H. Ray, editors, CACHE Corporation, Austin, TX (1991).

Piovoso, M., K. Kosanovich and R. Pearson, "Monitoring Process Performance in Real Time," *Proceedings Amer. Control Conf.*, p. 2359, Chicago, IL, June (1992a).

Piovoso, M., K. Kosanovich and J. Yuk, "Process Data Chemometrics," *IEEE Trans. Instru. and Meas.*, **41**, 262 (1992b).

Pollard, J. F., M. R. Broussard, D. B. Garrison and K. Y. Sen, "Process Identification Using Neural Networks," *Comput. Chem. Eng.*, **16**, 263 (1992).

Ponton, J. W. and J. Klemes, "Alternatives to Neural Networks for Inferential Measurement," *Comput. Chem. Eng.*, **17**, 491 (1993).

Peterson, R., A. Fredenslund, and P. Rasmussen, "Artificial Neural Networks as a Predictive Tool for Vapor-Liquid Equilibrium," *Comput. Chem. Eng.*, **18**, 563 (1994).

Prevost, P. H., A. Isambert, D. Depeyre, C. Donadille, and R. Perisse, "Some Practical Insights into Neural Work Implementation in Metallurgical Industry," *Comput. Chem. Eng.*, **18**, 1157 (1994)

Qian, N. and T. J. Sejnowski, "Predicting the Secondary Structure of Globular Proteins Using Neural Network Models," *Mol. Biol.*, **202**, 865 (1988).

Qin, S. J. and T. J. McAvoy, "Nonlinear PLS Modeling Using Neural Networks," *Comput. Chem. Eng.*, **16**, 379 (1992).

Rangwala, S. S. and D. A. Dornfield, "Learning and Optimization of Machining Operations Using Computing Abilities of Neural Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, **19**, 299, March/April (1989).

Reisner, J., M. A. Reuter and J. Krüger, "Modeling of the Mass Transfer in Gas-Sparged Electrolyses with Neural Nets," *Chem. Eng. Sci.*, **48**, 1089 (1993).

Reuter, M. A., J. S. J. van Deventer and T. J. van der Walt, "A Generalized Neural-Net Kinetic Rate Equation," *Comput. Chem. Eng.*, **48**, 1281 (1993).

Robb, E. W. and M. E. Munk, "A Neural Network Approach to Infrared Spectrum Interpretation," *Mikrochim. Acta*, **1**, 131 (1990).

Savkovic-Stevanovic, J., "Neural Networks for Process Analysis and Optimization : Modeling and Applications," *Comput. Chem. Eng.*, **18**, 1149 (1994).

Servais, R. A., C. W. Lee and C. E. Browning, "Intelligent Processing of Composites," *SAMPE J.*, **22**, 5, 14 (1986).

Shao, X., "Synthesis of Waste Treatment Systems", PhD dissertation, chemical engineering, University of Cincinnati, OH (1993).

Shieh, D. S. S., "Automated Knowledge Acquisition from Routine Data for Process and Quality Control," Ph.D. Dissertation, Chemical Engineering, Washington University, St. Louis, MO (1992).

Ulbrecht, J. J., editor, *Processing Sensing and Diagnosis, AIChE Symp. Series*, **85**, No.267 (1989).

Van Der Walt, T. J. and J. S. J. van Deventer, "The Dynamic Modeling of Ill-Defined Processing Operations Using Connectionist Networks," *Chem. Eng. Sci.*, **48**, 1945 (1993).

Wizzard, J. T. and M. G. Fuhrman, "Neural Network Application to Remote Temperature Measurements," *AIChE National Meeting*, Houston, TX, April (1991).

Wu, H. T., "Knowledge-Based Control of Composite Material Manufacturing Processes," Ph.D. Dissertation, Chemical Engineering, Washington University, St. Louis, MO (1990).

Wu, H. T. and B Joseph, "Model-Based Control of Autoclave Curing of Composites," *SAMPE J.*, **26**, No. 6, 39 (1990).

Wythoff, B. J., S. P. Levine and S. A. Tomellini, "Spectral Peak Verification and Recognition Using a Multilayered Neural Networks," *Anal. Chem.*, **62**, 2702 (1990).

Zhou, D. N., V. Cherkassy, T. R. Baldwin and D. E. Olson, "A Neural Network Approach to Job-Shop Scheduling," *IEEE Trans. on Neural Networks*, **2**, 175 (1991).