Contract N00014-89-C-0137
Task Final Technical Report
Contract LIN 0001, 0002 (partial)

Application of Polynomial Neural Networks to Classification of Acoustic Warfare Signals

David G. Ward
Roger L. Barron
B. Eugene Parker, Jr., Ph.D.

April 1993

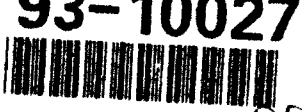
Prepared for:

DEPARTMENT OF THE NAVY
Office of the Chief of Naval Research
Applied Research and Technology Division
800 North Quincy Street
Arlington, Virginia 22217-5000

Prepared by:

BARRON ASSOCIATES, INC.
Route 1, Box 159
Stanardsville, Virginia 22973

(804) 985-4400



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1993	Final Technical, 20 Aug 89-31 Mar 93

4. TITLE AND SUBTITLE
Application of Polynomial Neural Networks to Classification of Acoustic Warfare Signals

5. FUNDING NUMBERS
C: N00014-89-C-0137

6. AUTHOR(S)
David G. Ward
Roger L. Barron
B. Eugene Parker, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Barron Associates, Inc.
Rt. 1, Box 159
Stanardsville, Virginia 22973

8. PERFORMING ORGANIZATION REPORT NUMBER
141-01 FTR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Research Projects Agency
Maritime Systems Technology Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
A. Approved for Public Release. Distribution is Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

For both estimation and classification problems, the benefits of using artificial neural networks include inductive learning, rapid computation, and the ability to handle high-order and/or nonlinear processing. Neural networks reduce the need for simplifying assumptions that use *a priori* statistical models (such as "additive Gaussian noise") or that neglect nonlinear terms, cross-coupling effects, and high-order dynamics. This report demonstrates the usefulness for acoustic warfare applications of an interdisciplinary approach that applies the rigorous theory and algorithms of statistical learning theory to the field of artificial neural networks. In particular, this approach provides two important results: (1) a generalized way of viewing neural modeling in terms of statistical function estimation, and (2) a constrained minimum-logistic-loss polynomial neural network (PNN) classification algorithm. These classification neural networks train rapidly, provide improved discrimination, and use an information-theoretic approach to limit structural complexity and thus avoid overfitting training data. The report documents the successful application of these algorithms for the purpose of discriminating among broadband acoustic warfare signals and makes recommendations concerning further improvement of the algorithms.

14. SUBJECT TERMS			15. NUMBER OF PAGES
Artificial Neural Networks	Acoustic Warfare	Sonar Signal	188
Estimation	Machine Learning	Processing	16. PRICE CODE
Classification	Modeling		

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	NONE

FOREWORD

From September 20, 1989, to March 31, 1993, under the terms of Contract N00014-89-C-0137 with the Office of the Chief of Naval Research (OCNR), Barron Associates, Inc. (BAI) studied the synthesis and use of artificial neural networks for the purposes of detecting and classifying underwater transient acoustic sources using broadband information. Particular emphasis was placed on neural networks trained using a constrained logistic loss function. These are here called minimum-logistic-loss polynomial neural networks (PNNs). This work leading to this report has been supported primarily by the Advanced Research Projects Agency (ARPA) and the Naval Command, Control and Ocean Surveillance Center (NCCOSC) via the above ONR contract.

This is the final technical report on BAI's work under the ARPA Non-Traditional Exploitation System (*DANTES*) program. This report and prior submittals fulfill the requirements of Contract Line Item Nos. 0001 and 0002. Under CLIN 0003, 0004, 0005c and 0006 of the contract, BAI performed research in the area of static and dynamic (or recurrent) polynomial neural networks (with particular emphasis on active control of combustion systems and predictive modeling of synchronous generators); this work was completed and a final technical report thereon was delivered to OCNR on May 30, 1992. Under additional modifications to the contract, BAI continues to investigate the application of static and dynamic neural networks in the areas of acoustic radiation and multivariable control. This ongoing work will be documented in future technical reports for OCNR.

Many individuals in the Government, in industry, and in academia have contributed importantly to the work reported. The OCNR Scientific Officers for this work have been Dr. Robert J. Hansen (now with The Pennsylvania State University), Dr. Eric W. Hendricks (with NCCOSC and OCNR), Mr. James G. Smith, and Cdr. Daniel A. Forkel. This work would not have been accomplished without their guidance, support, and encouragement, for which the authors are deeply appreciative. The support and direction of the ARPA Maritime Systems Technology Office, NCCOSC/NRAD, Planning Research Corporation, and Orincon Corporation are gratefully acknowledged. The ARPA program managers (chronologically) have been Charles E. Stewart, Dr. Paul R. Blasche, and Paul A. Rosenstrach. The NCCOSC/NRAD program manager has been Louis E. Griffith, Code 7304. Thomas J. Martin of the Planning Research Corporation ASW Division (ARPA contractor) has been closely involved in program administration. Many at Orincon Corporation (ARPA/NCCOSC contractor) have contributed, including Dr. Vivek Samant, Dr. Thomas Brotherton, Michael Kurnow, and Dr. Avi Krieger. Paul Hess (Consultant) participated closely in behalf of BAI during the work that was performed at Orincon.

Within BAI, the work of Dean W. Abbott, Richard L. Cellucci, and Paul R. Jordan, III, as well as other members of the BAI staff has been of considerable benefit to earlier phases of this research. Dr. Andrew R. Barron, Professor of Statistics at

Yale University, has provided invaluable advice on information-theoretic aspects of neural network synthesis. Furthermore, the authors express their appreciation to Susan L. Woodson of the BAI staff for the word processing involved in preparation of this report.

The evaluation of the classification algorithms on *Short-Net* data was funded, in part, by an Orincon Corporation subcontract (Orincon purchase order S04365).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND SUMMARY

As sonar systems become increasingly complex and sensitive, sonar displays may become cluttered with hundreds of possible target signatures. Traditionally, sonar operators have manually classified each target signature; however, as an increasing number of signatures appear, timely manual classification of all potential threats is nearly impossible; this problem becomes even more pronounced if, as is often the case, a threat is operating quietly in an attempt to avoid detection. Additionally, due to the relatively infrequent occurrence of hostile targets, a human operator may be required to observe cluttered sonar displays for long periods of time without ever sighting a potential threat, leading to inattention, operator fatigue, and an increased risk of missed detections.

The DARPA Non-Traditional Exploitation System (*DANTES*) addresses these problems in part by combining advanced passive sonar devices with digital signal processing to automatically detect, classify, and track targets of interest. The primary technical objective of the research conducted by Barron Associates, Inc. under the *DANTES* program was to apply dynamic and static polynomial neural network synthesis algorithms to the modeling of functions that detect and classify a broad range of acoustic signal classes, with particular emphasis on transient and other broadband acoustic sources. The work combined state-of-the-art techniques in neural networks, statistical inference, and computer science, and had two main goals:

(1) Develop a design methodology and specific design embodiments to provide accurate pre-classification and classification of signals received by an acoustic surveillance system.

(2) Develop on-line learning capabilities to incorporate new signal classes into the system design.

The ability of neural networks to derive inductively (from the data) complex nonlinear system models has made them good candidates for the automation of detection and classification processes. However, most neural network approaches employed to date, including those using multi-layer perceptrons trained via Backpropagation, suffer from a number of disadvantages: (1) use of a squared-error loss function is not optimal for multi-class data; (2) training time is prohibitively long; (3) training data for which limited numbers of training exemplars are available are prone to be overfitted, resulting in poor performance on unseen data; (4) outputs require post-processing before decisions can be made; and (5) complex structures, often with tens of thousands of coefficients, require a computationally-intensive on-line interrogation process.

In addressing these problems, the authors took an interdisciplinary approach that applied rigorous statistical learning theory to the field of artificial neural networks. This approach yielded two important results:

(1) *A generalized way of viewing neural modeling in terms of statistical function estimation.* A number of traditional neural modeling techniques can be recast in terms of this generalized function estimation technique; as a result, the strengths and weaknesses of particular neural modeling algorithms become more readily apparent, and potential algorithmic improvements often suggest themselves.

(2) *A minimum-logistic-loss polynomial neural network (PNN) classification algorithm.* This algorithm retains the benefits of traditional neural-network-based approaches while overcoming many of the difficulties mentioned above. Some of the most important characteristics of minimum-logistic-loss PNN classifiers are:

- Minimization of the logistic loss function, resulting in optimal (maximum likelihood) classification of data having a multinomial probability distribution. (Estimators of functions with categorical output values are more likely to have a multinomial error distribution than a Gaussian distribution [7].)
- A Levenberg-Marquardt optimization algorithm for rapid on-line and off-line network training.
- Ability to provide nonlinear classification having a degree of complexity (i.e., classification power) commensurate with the quantity and representativeness of the training database.
- Outputs that are estimates of the *a posteriori* probabilities of class membership. These are particularly useful when these outputs are used by higher-level decision-making processes.
- Simple network structures that do not overfit the training data and that can be interrogated rapidly on-line.

Section 2 of this report presents a neural modeling perspective based upon the theory of statistical estimation of functions. Two aspects of neural modeling are covered in detail; these are: (1) generalized network structures capable of implementing a variety of current and proposed neural paradigms, and (2) fast learning algorithms capable of optimizing a variety of nonlinear network structures. Section 2 concludes with a discussion of some of the more popular neural network paradigms in light of the function estimation principles discussed; these principles form the basis of the classificatory neural network paradigms investigated under the *DANTES* program.

Section 3 presents the details of the polynomial neural network synthesis algorithms used by the authors and outlines directions for further algorithmic research. Particular attention is given to the *Algorithm for Synthesis of Polynomial Neural Networks for Classification (CLASS).* This algorithm has provided excellent

results in the classification of acoustic transients and has also proven highly useful in other application domains including image processing, financial analysis, guidance, and control.

Section 4 documents the successful application of PNN classification algorithms on actual sonar data, and Section 5 describes two PNN systems delivered for incorporation in *DANTES*.

2. PRINCIPLES OF FUNCTION ESTIMATION USING ARTIFICIAL NEURAL NETWORKS

2.1 Introduction

To construct an effective neural network for any purpose, including classification of acoustic transients, one must make several decisions regarding the fundamental structure of the network and the algorithms that will be used for network generation. To make these decisions properly it is helpful to understand network structure and network training in the broader context of generalized function estimation. This section is intended to bring unity to a variety of neural network paradigms including polynomial neural networks (PNNs). The section begins with a discussion of a generic neural network structure, proceeds to discuss methods for optimizing both the network coefficients *and* structure, and concludes with a discussion of the relationship between the method presented and a variety of other commonly used neural-network and statistical function-estimation techniques.

Some notes concerning terminology are in order. The authors emphasize the difference between *estimation* and *classification* neural networks, the former being suited best for function estimation, filtering, control, smoothing, and prediction tasks, and the latter being most appropriate for data discrimination tasks. In this section, however, *classification* networks are viewed as networks trained to provide the best estimates of discrimination functions between classes of data. And so, in this context, *classification* networks are viewed as particular instantiations of function estimators.

2.2 Network Structure

An artificial neural network is typically composed of nodal elements that perform a learned transformation between input and output data vectors. Sets of nodal elements are connected in a specific way to comprise layers; the layers in turn are connected to create the entire network. Fig. 2.1 shows the structural hierarchy there exists, at least in principle, within a neural network:

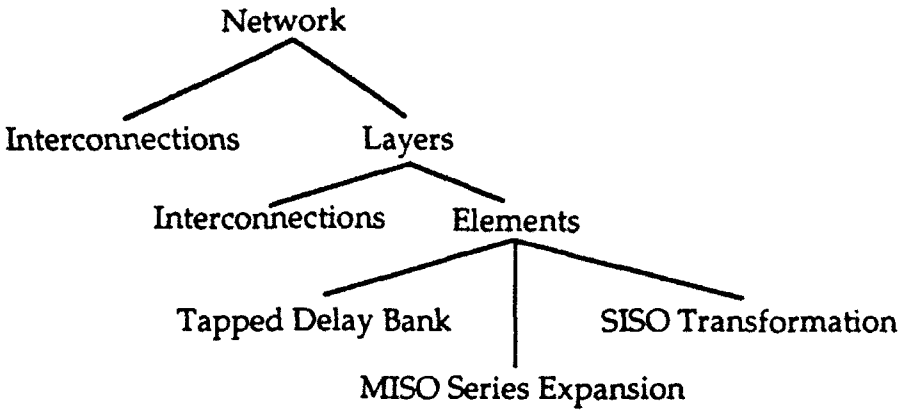


Figure 2.1: Artificial Neural Network Structural Hierarchy

2.2.1 Network Inputs and Outputs

On the highest level, an artificial neural network is a transformation which, when interrogated, produces an output vector, $\underline{s}$, in response to a given input vector, $\underline{x}$. In the case of static networks, the output vector is a single-point transformation of the input data:

$$\underline{s}_i = f(\underline{x}_i, \underline{\theta}) \qquad 2:1$$

where $\underline{\theta}$ is the set of network parameters. Dynamic networks contain internal feedbacks and time delays, and produce a transformation of the form

$$\underline{s}_i = g[\underline{x}_i, \ldots, \underline{x}_{i-m}, \underline{s}_{i-1}, \ldots, \underline{s}_{i-m}, \underline{\theta}] \qquad 2:2$$

Neural networks are typically imbedded in systems and are trained to produce a desired output or effect on the system response. Training involves batch or recursive fitting of a numerical database; we define the training database as:

$$(\underline{x}_i, \underline{y}_i); \qquad i = 1, 2, \ldots, N \qquad 2:3$$

where N is the number of data vectors in the training database and $\underline{x}_i$ and $\underline{y}_i$ are the *measured* inputs and desired outputs or system responses for the i^{th} observation. If the training is *unsupervised*, then there is no knowledge of the desired outputs, $\underline{y}_i$, and only $\underline{x}_i$ is used for training. In one sense, the distinction between *supervised* and *unsupervised* learning is not necessary, since even in *unsupervised* learning, networks are trained to perform some desired transformation on the input data, and the means for determining success or failure are always provided by the analyst *a priori*. In this sense, all learning is supervised.

Often the network output is written as $\underline{\hat{y}}$ instead of $\underline{s}$; however, this invokes the interpretation that the network output is an estimate of the system response

recorded in the training database. For system identification, inverse modeling, and classification such is certainly the case, but there are other instances in which the network output is not intended to be the best estimate of the database response vector.

In certain control applications, for example, it is not the network output, but a transformation of the network output, that is fitted to the response values recorded in the training database. Fig. 2.2 illustrates a multiple-input, multiple-output (MIMO) network controller. In this figure, the network is adapted on-line because the network itself is part of the overall input-output transfer function. The desired network response is the one which, when passed as input into the plant, produces over time the minimum absolute error between plant output and the reference signal.

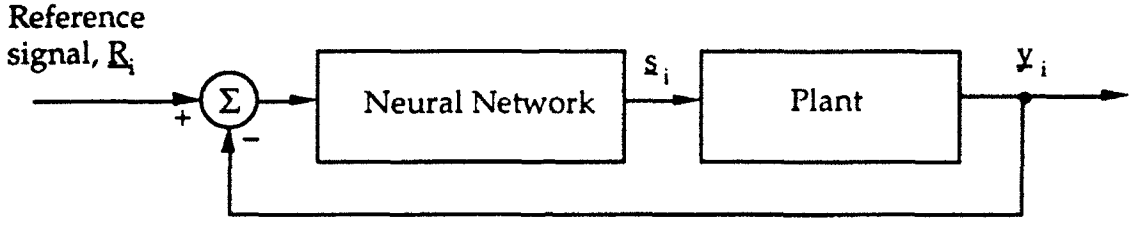
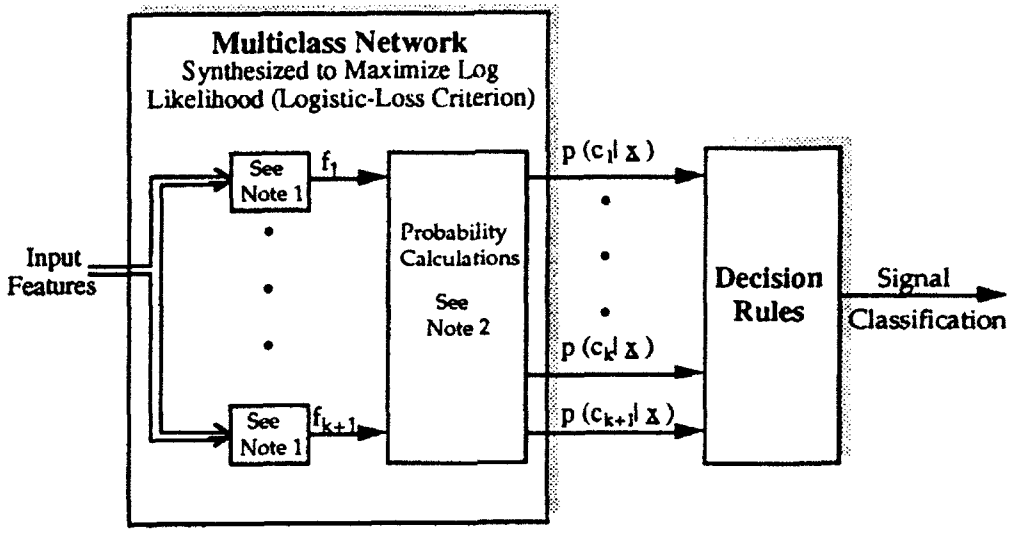


Figure 2.2: MIMO Network Controller

In some network applications, the desired network output is neither the best estimate of the database response values nor a best control signal, but is operated on by an additional transformation. In many classification tasks, for instance, the training database response vector, $\underline{y}_i$, is assigned an integer scalar representing the class of the $\underline{x}_i$ vector. The desired network output, however, is a vector of estimated class probabilities (or log-odds) given that the input state is $\underline{x}_i$. This output vector may then be fed into appropriate decision logic to determine the signal classification (Fig. 2.3). These decision rules may be as simple as assigning the signal to the class corresponding to the network output with the highest probability.

2.2.2 Element Definitions

Most artificial neural networks are comprised of fundamental building blocks called nodes, elements, or nodal elements; a generalized nodal element is shown in Fig. 2.4. This generalized nodal element may be built upon an algebraic or other series expansion, sometimes called the core transformation. The expansion is often composed with a fixed post-transformation function, $h(\cdot)$, that may be linear or nonlinear. In addition, the inputs to a nodal element may be passed through shift registers or delay banks to allow the series expansion to have access to prior input values.



Note 1: Multi-Input, Single Output (MISO Network)

Note 2: The output probabilities are calculated using the formula:

$$p(j) = \frac{e^{f_j}}{1 + \sum_i e^{f_i}} \qquad \begin{matrix} \text{where } i = 1, \ldots, k \\ j = 1, \ldots, k+1 \end{matrix}$$

Figure 2.3: Neural Network Used for Data Classification

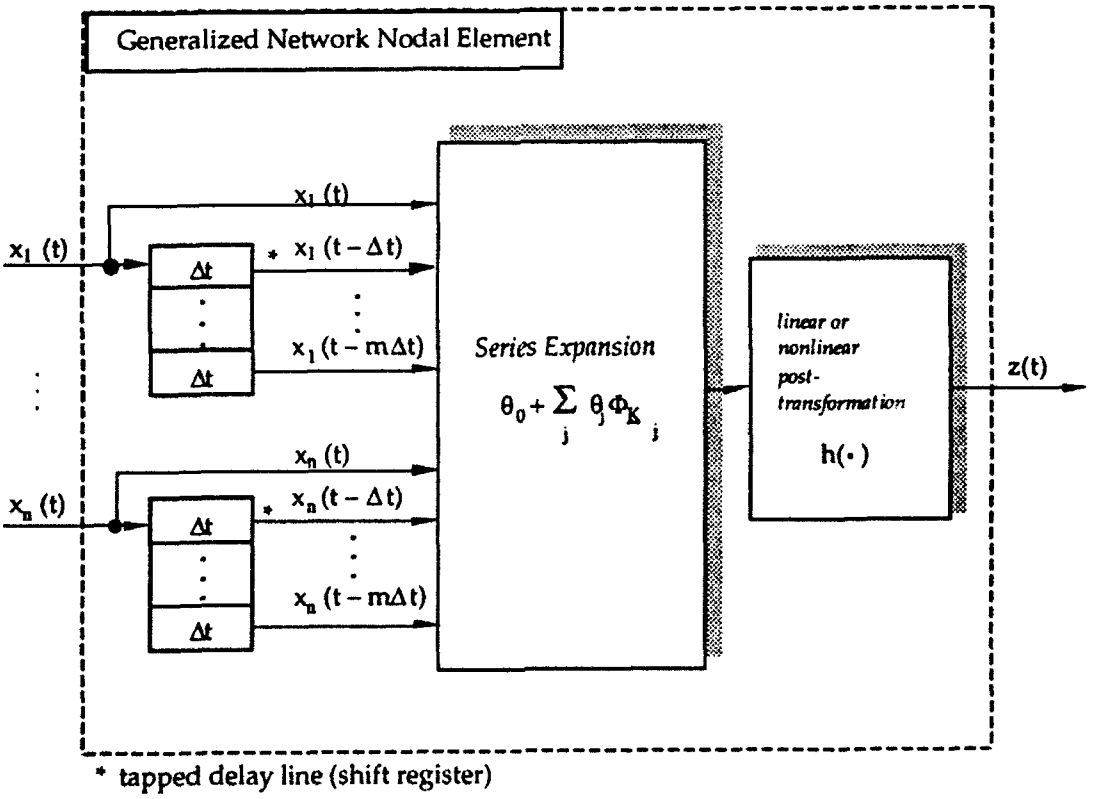


Figure 2.4: Generalized Network Nodal Element

2.2.2.1 *Basis Functions and Series Expansions*

The series expansion of Fig. 2.4 is of the form

$$z = \sum_{j=0}^{J} \theta_j \Phi(\underline{k}_j, \underline{x}) \qquad 2:4$$

where $\underline{\theta}$ is the vector of element coefficients, J is the total number of non-constant terms in the expansion, $\underline{k}_j$ is a vector of integers. A bias term is ensured by requiring that $\Phi(0, \underline{x}) = 1$. The series expansion within a neural network element has the same form as traditional series expansion techniques; however, with network function estimation, it is desirable that the total number of terms in any given element be kept as small as possible. This point will be elaborated on shortly.

The inclusion of $\underline{k}_j$, sometimes called the set of indices or multi-indices, in Eq. 2:4 allows the series expansion to handle both univariate and multivariate cases. For the multivariate case, each $\Phi(\underline{k}_j, \underline{x})$ is a product of functions of scalars. $\underline{k}_j$ is usually taken to be a vector of integers with each element of $\underline{k}_j$ corresponding to one of the variables in the $\underline{x}$ vector. Using this notation, the jth term in the series expansion may be written as:

$$\Phi(\underline{k}_j, \underline{x}) = \Phi(k_{j1}, x_1) \cdot \Phi(k_{j2}, x_2) \cdot \ldots \cdot \Phi(k_{jD}, x_D) \qquad 2:5$$

where D is the total number of inputs to the series expansion.

The notation introduced above (and thus the nodal element) is sufficiently general to implement a variety of basis functions. Table 2.1 gives examples of how the function $\Phi(k_{jd}, x)$ may be chosen to implement some basis functions commonly used in function estimation (note that in Table 2.1 the subscripts have been dropped from k where the basis function does not depend on them).

For the polynomial basis function (Eq. 2:6), the $\underline{k}_j$ vector is used to determine the powers to which the input variables are raised in the jth term of the expansion. The same is true for the spline basis function (Eq. 2:7); however, the degree of the function is never allowed to exceed r; thus r = 3 results in the commonly used cubic spline.

Note that in both the spline and the wavelet cases an additional set of multi-indices, α_{jd}, must be specified. The parameter α_{jd} in Eq. 2:7 is sometimes called the "knot" and is the value about which the approximation takes place. In some cases, such as uniformly spaced knots, the knot set can be obtained automatically, eliminating the need to pre-specify the additional set of multi-indices.

Table 2.1: Some Basis Functions Commonly Used for Function Estimation

polynomial	$\Phi(k, x) = x^k$	2:6
spline	$\Phi(k_{jd}, x) = \begin{cases} x^k & \text{if } k < r \\ (x-\alpha_{jd})_+^r & \text{if } k \geq r \end{cases}$	2:7
orthonormal wavelet	$\Phi(k_{jd}, x) = \begin{cases} 2^{-k/2}\,\Psi(2^{-k}x - \alpha_{jd}) & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases}$	2:8
trigonometric	$\Phi(k, x) = \begin{cases} \sin\left(2\pi \frac{k+1}{2L} x\right) & \text{if } k \text{ is odd} \\ \cos\left(2\pi \frac{k}{2L} x\right) & \text{if } k \text{ is even} \end{cases}$	2:9

In the orthonormal wavelet basis function, $\Psi(\cdot)$ is termed the "mother wavelet" and must satisfy a number of specific conditions, including that it be continuous, integrate to zero, and be non-zero in a very specific limited range [21]. One such function is the Littlewood-Paley basis function:

$$\Psi(x) = \frac{\sin 2\pi x - \sin \pi x}{\pi x} \qquad 2:10$$

In the trigonometric basis function (Eq. 2:9), L represents the fundamental period of the expansion and depends on the sampling rate.

From Eqs. 2:4 – 2:9 it can be seen that the core expansion may be fully specified via a univariate basis function (of the form in Table 2.1) and a J × D matrix $\underline{\underline{K}}$, where each row of $\underline{\underline{K}}$ is the vector of integers $\underline{k}_j$ as defined above. We will illustrate this using two examples:

Example 1: Consider the "Full Double" element of Fig. 2.5. Because this element has no input delays and no post-transformation h(·), it is completely specified by the series expansion of Eq. 2:11

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^3 + \theta_7 x_2^3 + \theta_8 x_1^2 x_2 + \theta_9 x_1 x_2^2 \qquad 2:11$$

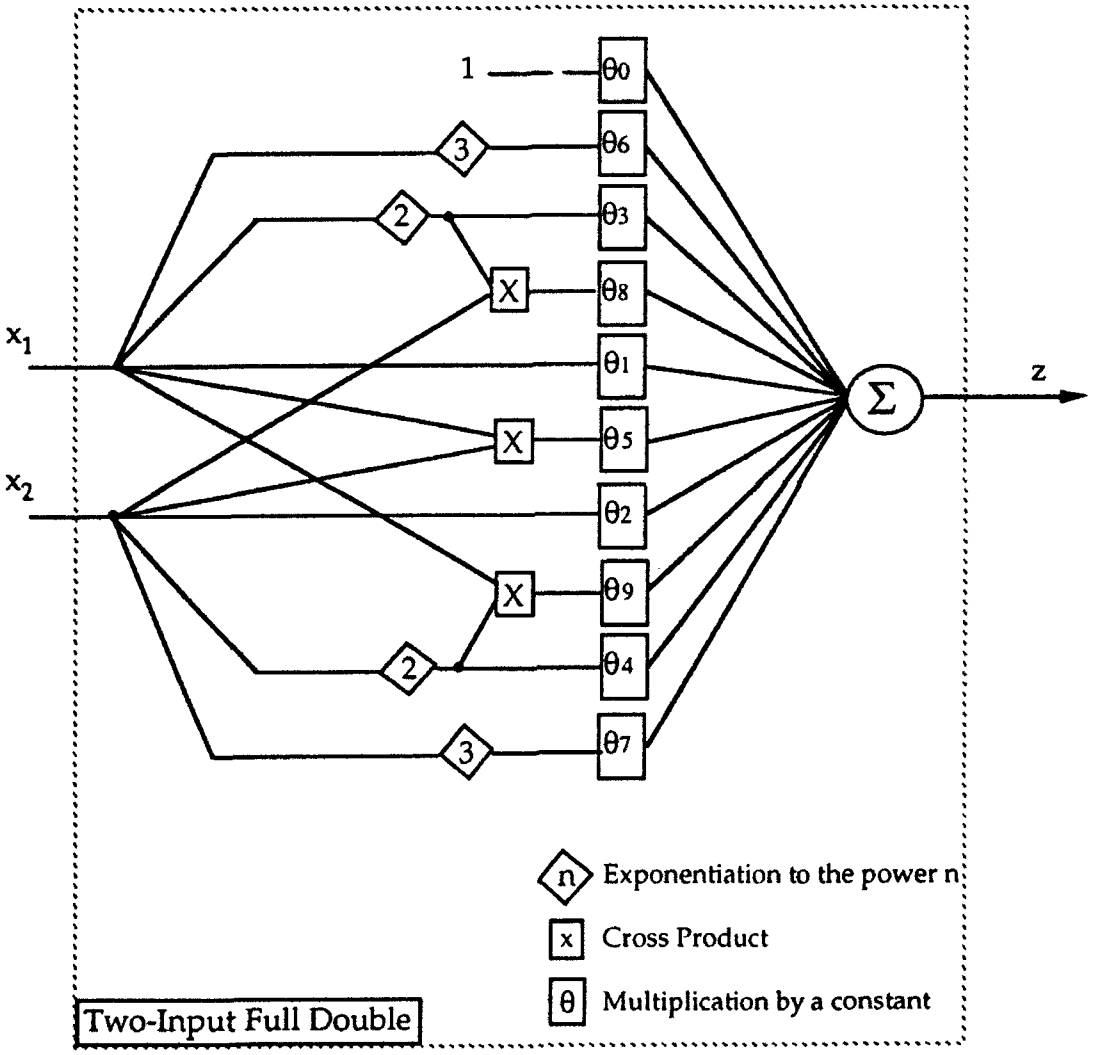


Figure 2.5: Example of a "Full Double" Network Element

If one chooses a polynomial basis function (Eq. 2:6), the J × D matrix $\underline{\underline{K}}$ corresponding to the expansion in Eq. 2:11 is

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \\ 3 & 0 \\ 0 & 3 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad 2:12$$

Note that because the basis functions of Table 2:1 were defined so that the value of any basis function at k = 0 is unity, the θ_0 coefficient is represented by an additional row of zeroes in the $\underline{\underline{K}}$ matrix.

Example 2: Consider the trigonometric series expansion

$$\theta_0 + \theta_1 \sin\left(2\pi \frac{3x_1}{L} x\right) + \theta_2 \sin\left(2\pi \frac{4x_1}{L} x\right) \cos\left(2\pi \frac{2x_2}{L} x\right) \qquad 2:13$$

If we choose a trigonometric basis function (Eq. 2:9), then the J × D matrix, $\underline{\underline{K}}$, that will yield the series in Eq. 2:13 is given by

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 5 & 0 \\ 7 & 4 \end{bmatrix} \qquad 2:14$$

The above $\underline{\underline{K}}$ matrix found as follows: The first term in the expansion of 2:13 is the bias term corresponding to a row of zeroes in the $\underline{\underline{K}}$ matrix (since $\Phi(0,\underline{x}) = 1$). The second term in the series expansion contains a single sin() term. Comparing this first term with the sin( ) expansion of 2:13, one finds that (k+1)/2 = 3, or k=5; since x_2 does not appear in the second term, the second column of the $\underline{\underline{K}}$ matrix (corresponding to x_2) contains a zero. The third term contains both an x_1 and an x_2 term. Solving (k+1)/2 = 4 and k/2 = 2 for the sin( ) and cos( ) terms respectively, one obtains the final row of the $\underline{\underline{K}}$ matrix.

2.2.2.2 *Limiting Series Expansion Complexity*

While the generalized nodal element is capable of implementing many commonly used series-expansion basis functions, neural network function estimation is fundamentally different from traditional series and nonparametric estimation techniques in the following ways:

- Each network element implements only a limited subset of the terms that would make up a complete series expansion; thus element complexity is kept low.
- Network interconnections allow a set of relatively simple network elements to be combined so that they can implement complex transformations; thus the network connections do a great deal of the "work" involved in the estimation problem.
- As the number of inputs to the function increases, the error bounds for network estimation can be shown to be more favorable than that of traditional function estimation techniques [11].

There are five factors, discussed below, that determine the number of coefficients (i.e., complexity) that will be needed in a given series expansion; by limiting one or more of these factors, the complexity of the nodal element may be kept relatively low.

Maximum Degree (R) and Number of Inputs (D): The degree, R, of any given basis function is the maximum value of the sum of the elements in a row of the $\underline{\underline{K}}$ matrix. Thus, for polynomial basis functions, the degree of a given term corresponds to the sum of the powers of the variables in the term. Thus, the $\underline{\underline{K}}$ matrix corresponding to a series expansion with three inputs (D=3) and maximum degree two (R=2) is:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \qquad 2:15$$

And, the $\underline{\underline{K}}$ matrix corresponding to a series expansion with two inputs (D=2) and maximum degree three (R=3) is:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 3 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 3 & 0 \end{bmatrix} \qquad 2:16$$

These expansions are referred to as *complete* expansions of degree R, and it can be shown that the number of terms in a complete series expansion is a function of the number of inputs to the expansion and the maximum degree of the expansion.

$$J = \frac{(R + D)!}{R!D!} \qquad 2:17$$

Thus, the first step in limiting the number of terms (or coefficients) in a series expansion is to limit either the maximum degree, R, of the expansion or the number of inputs, D, to the expansion. The maximum degree of the expansion is completely controllable by the analyst. And, while the number of inputs to the network is largely determined by the application, it is possible to limit the number of inputs to individual elements internal to the network (the GMDH algorithm described later in this section is an example where the number of inputs to any given nodal element is limited to two.)

Maximum Coordinate Degree (P): By placing restrictions on the maximum coordinate degree, P, the number of terms in the series expansion may be further reduced. The coordinate degree of any given series expansion is the maximum value of *any* integer in the $\underline{\underline{K}}$ matrix. For a polynomial basis function, this corresponds to limiting the power to which any given input may be raised. Thus, a three-input (D=3), second-degree (R=2), series expansion with maximum coordinate degree of one (P = 1) would have the following $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad 2:18$$

Maximum Interaction Order (Q): In multivariate function estimation, the interaction order, Q, corresponds to the maximum number of different input variables that may appear at the same time in a given term. Thus, in Eq. 2:13 above the interaction order is two because both x_1 and x_2 appear in the last term of the series. High numbers of interactions result in a combinatorial explosion in the number of terms needed for the complete series expansion, so a limit on the total number of interactions is one of the most important restrictions that can be placed on the nodal element series expansion. A cap on the maximum interaction order can be thought of as limiting the total number of non-zero elements in each $\underline{k}_j$ vector. Thus, the $\underline{\underline{K}}$ matrix for a three-input (D=3) second-degree (R=2) expansion with a maximum coordinate degree of one (P=1) and maximum interaction order of one (Q=1) is

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \qquad 2:19$$

Note that for any series expansion having D inputs, degree R, coordinate degree P, and interaction order Q, the J×D $\underline{\underline{K}}$ matrix may be obtained by "counting" from zero in a base-P system; each row in the matrix represents one number in the series. Once the sequence of numbers is generated, all rows containing more than Q non-zero terms are removed, and all rows with sums greater than R are removed.

Expansion Density: Even after the number of inputs, degree, coordinate degree, and interaction order for a given series expansion are limited, one may choose to remove some terms to obtain a sparse or low-density expansion. Eq. 2:13 and the corresponding $\underline{\underline{K}}$ matrix in Eq. 2:14 exemplify a sparse expansion. While this series has an interaction order of two, a maximum degree of 11, and two inputs, there exist other terms that meet the interaction order and degree constraints and yet are not included in the series expansion. Often a sparse series expansion is obtained by "carving" away any terms in the expansion that have little or no effect on the desired network response. Details of this carving technique are described in Sect. 2.3.3.

Because it is desirable to keep the total number of network coefficients small, more emphasis is placed on determining an appropriate, efficient network structure, and less on problems associated with extremely high-dimensional nonlinear optimization. In general, this approach proves to be more parsimonious in its use of computing resources and also leads to more robust models that do not have an excessive number of internal degrees of freedom.

2.2.2.3 *Linear and Nonlinear Post-Transformations*

The linear or nonlinear fixed *post-transformation*, h(·), of Fig. 2.3, in conjunction with the basis function selected, allows the element specification to be sufficiently general to encompass most neural network nodal elements currently in use. The transformation may be used to introduce helpful nonlinearities into the network, especially when there are few or no nonlinearities in the core transformation. Additionally, the transformation may be used to "clip" the output of the core transformation, which often improves the stability of the network (in the bounded-input, bounded-output sense). This may be especially important when a polynomial core transformation is evaluated near or outside the boundaries of its training region. Fig. 2.6 shows the role of the post-transformation for the popular sigmoidal element used in multi-layer perceptron (MLP) neural networks.

The core transformation of the element shown in Fig. 2.6 also has no time delays and implements a series expansion of the form

$$\theta_0 + \sum_{j=1}^{D} \theta_j x_j \qquad 2:20$$

Following the same method outlined above, this series expansion can be represented by choosing the polynomial basis function of Eq. 2:6 and letting $\underline{\underline{K}}$ be a $D \times D$ identity matrix. Because $\underline{\underline{K}}$ contains only first-order interactions and has a maximum power of one, the number of terms in the series expansion is kept low.

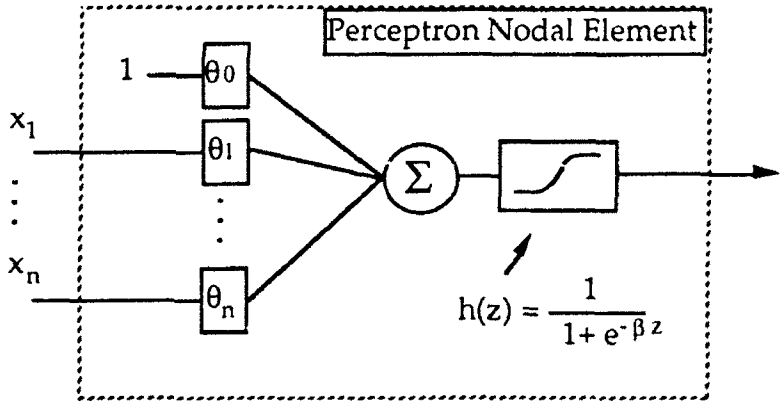


Figure 2.6: An MLP Network Element

The post-transformation, h(·), of Fig. 2.6 is a sigmoidal transformation and has the formula given in the figure. Due to the nonlinear post-transformation, the MLP nodal element is nonlinear in its parameters.

Another use for the post-transformation, h(·), is to allow the generalized nodal element to implement other types of function approximations that are not

simple series expansions. Suppose, for instance, one wants a trigonometric function of the form

$$z = \sin(\theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n) \qquad 2:21$$

In this case, the series expansion is a first-order polynomial expansion, while the post-transformation is the sin(·) function.

2.2.3 Layer Definition

A layer is a set of elements whose inputs are selected from the same set of candidates. It is important to define a layer as a distinct unit within the network for the following reasons:

First, when determining network structure, it is often convenient to build a unit of network structure and then freeze it while building other units of the structure. The network layer is this unit of structure. This is analogous to constructing a building one floor at a time; each subsequent floor is built upon the floors below it, and construction on a new floor cannot begin until a sufficient portion of the lower floors has been constructed.

Second, elements on a given layer are often trained to "work together" as a group to produce the desired network response (Section 2.3.3).

In addition to the internal layers, a network will often contain two special-purpose layers. The first receives inputs, normalizes them, and passes the normalized values to subsequent layers. Often if the inputs are normalized, the network is trained on normalized outputs as well. When this is the case, a second special-purpose layer is required to unitize (or un-normalize) the network outputs. By normalizing and unitizing, each network input is allowed to contribute equally to the solution of the problem, and the magnitudes of the network coefficients become a more accurate reflection of the relative importance of a given term.

2.2.4 Network Interconnections

Fig. 2.7 shows the two types of network interconnections: feedforward and feedback. Intra-layer connections consist of making the inputs to each layer available to every element in the layer. The individual elements are then free to choose which subsets of the available inputs to use. Element outputs are then passed along as layer outputs; the layer outputs may be described by a vector containing scalar values corresponding to the element outputs. Network inputs are available as element inputs at successive layers.

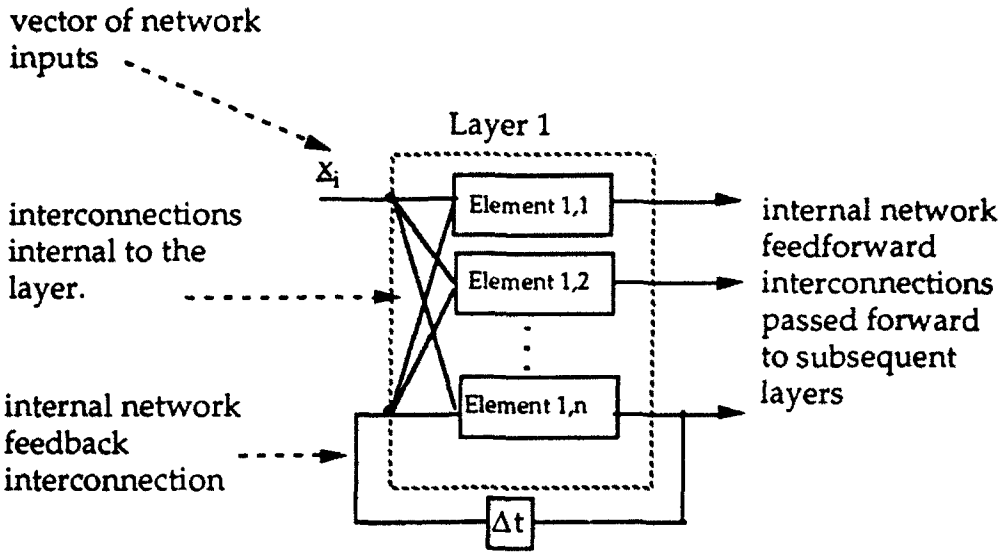


Figure 2.7: Network Interconnections

It is important to note that in the function estimation techniqe presented here, we do not allow feedback connections internal to the layer or within elements. This restriction allows the same layer definition to serve for both feedforward and feedback networks. Connections between layers, however, may be passed forward as inputs to subsequent layers (feedforward networks) or may be passed back as inputs into the given layer and/or previous layers (feedback networks). Thus, for the generalized network structure outlined in Section 2.2, the only difference between dynamic and static networks is the type of inter-layer interconnections allowed.

2.3 Network Training

Often networks are trained using a gradient-based search technique to find the coefficients of a pre-structured network; the popular backpropagation algorithm [62] is an example of this type of training, where the specific optimization algorithm is a form of least mean squares (LMS). The recommended approach, however, is to allow for structural variations by including in the training algorithm(s) methods for determining a network structure suitable for the task at hand. Thus, building the network structure and optimizing coefficients are inter-twined processes used to create more robust networks with less training effort and time.

2.3.1 The Loss Function

For a given network structure the optimal coefficients are those which minimize the sum of a loss function evaluated at every observation in a training database:

$$\min\left(\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i)\right) \qquad 2:22$$

where:

N is the number of observations in the training database

$\underline{y}_i$ is the i^{th} output vector in the training database

$\underline{s}_i$ is the i^{th} output vector of the network; $\underline{s}_i = f(\underline{x}_i, \underline{\theta})$ for feedforward networks (see Eq. 2:1)

$d(\cdot)$ is the loss or *distortion* function.

Because the goal of the network training algorithm is to minimize the output error as quantified by the loss function, it is helpful if the loss function is a convex, twice-differentiable function with respect to the coordinates of $\underline{s}_i$. By imposing these constraints on the loss function, one guarantees that, if the function being fitted is linear in its parameters, the fitting algorithm will be able to find the set of coefficients that globally minimizes the network error. Even if the network function is nonlinear in the parameters, a convex, twice-differentiable loss function will still result in the best performance possible for the optimization algorithm. Depending on the nature of the application, a variety of loss functions may be used effectively.

2.3.1.1 *Squared-Error Loss Function*

The squared-error loss function is by far the most commonly used and can be expressed as

$$d(\underline{y}_i, \underline{s}_i) = |\underline{y}_i - \underline{s}_i|^2 \qquad 2:23$$

In this case, the vector norm $|\cdot|^2$ is defined as the sum of the squares of the differences between the coordinates of $\underline{y}_i$ and $\underline{s}_i$. This loss function is most suited to create networks whose outputs estimate the dependent variable(s) in the training database as closely as possible.

One problem with the squared-error loss function is that data outliers tend to have a greater-than-desirable impact on the coefficient optimization. A number of *robust* loss functions have been suggested which reduce or nullify the effect of outlying data. One such function is Huber's loss function

$$d(\underline{y}_i,\underline{s}_i) = \begin{cases} |\underline{y}_i - \underline{s}_i|^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 \le A \\ 2A|\underline{y}_i - \underline{s}_i| - A^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 > A \end{cases} \qquad 2:24$$

where A is the distance at which outliers begin to have less effect. When $|\underline{y}_i - \underline{s}_i| > A$, $d(\cdot)$ becomes a 1-norm. Thus, this loss function has the advantages of a 1-norm; however, by using a 2-norm near the origin, the function is everywhere continuous in the first and second derivatives, which is not the case with a 1-norm loss function. Note that in Eq. 2:24 it may be desirable to shape each coordinate of the output norms differently by using an N-dimensional vector of values for A.

2.3.1.2 *Logistic-Loss Function*

Optimization of Eq. 2:22 for the squared-error distortion function of Eq. 2:23 corresponds to the maximum likelihood rule in the case of a Gaussian probability model for the distribution of the errors [44]. However, for multi-class classification problems with categorical output variables, a multinomial probability model in regular exponential form is more suitable than the Gaussian model. In this case, the network functions should be used to model the log-odds associated with the conditional probability of each class given the observed inputs. In this setting, the maximum likelihood rule corresponds to the choice of the *logistic* loss function,

$$d(\underline{y}_i, \underline{s}_i) = -\underline{y}_i \bullet \underline{s}_i + \text{Ln}\left(\sum_{j=1}^{C} e^{s_{i,j}}\right) \qquad 2:25$$

where C is the number of outputs (or classes); $s_{i,j}$ is the jth element of the $\underline{s}_i$ vector; and $\underline{y}_i$ is a vector with the coordinate of the observed class equal to one, and all other coordinates equal to zero (i.e., the observed conditional probabilities given $\underline{x}_i$). In this context, the likelihood associated with observation i is

$$p(\underline{y}_i | \underline{x}_i) = \frac{e^{\underline{y}_i \bullet \underline{s}_i}}{\sum_{j=1}^{C} e^{s_{i,j}}} \qquad 2:26$$

and Eq. 2:25 expresses the minus log-likelihood $d(\cdot) = -\log p(\underline{y}_i | \underline{x}_i)$. In this way, it is possible to compute estimates of the probability that an observation is a member of class k, given that the input state is $\underline{x}_i$:

$$p(k | \underline{x}_i) = \frac{e^{s_{i,k}}}{\sum_{j=1}^{C} e^{s_{i,j}}} \qquad 2:27$$

2.3.1.3 *Likelihood-Based Loss Function*

Likelihood-based loss functions, such as the logistic loss function described above, can also be helpful for density estimation and clustering of input data. For instance, the loss function may take the form

$$d(s_i) = -\text{Ln}\, s_i \qquad 2:28$$

where $s_i = f(\underline{x}_i, \underline{\theta})$ and $f(\cdot)$ is the estimated probability density function. In that case, the network output would need to satisfy

$$\int f(\underline{x}, \underline{\theta})dx = 1 \qquad 2:29$$

and

$$f(\underline{x}, \underline{\theta}) \geq 0 \quad \forall\, \underline{x} \qquad 2:30$$

If the network function output, $f(\underline{x}, \underline{\theta})$, does not satisfy the integrability requirement of Eq. 2:29, this condition can be reflected in the choice of the loss function by setting it equal to

$$-\text{Ln}\, s_i + \text{Ln} \int f(\underline{x}, \underline{\theta})dx \qquad 2:31$$

where the second term plays the role of normalizing the network output.

If the network does not satisfy the positivity requirement of 2:30, one can use the network function to model the log-density, and take the density function to be

$$p(\underline{x}_i, \underline{\theta}) = \frac{e^{f(\underline{x}_i, \underline{\theta})}}{\int e^{f(\underline{x}, \underline{\theta})}dx} \qquad 2:32$$

and the minus log-likelihood to be

$$-\log\left(p(\underline{x}_i, \underline{\theta})\right) = -s_i + \text{Ln}\left(\int e^{f(\underline{x}, \underline{\theta})}d\underline{x}\right) \qquad 2:33$$

where $s_i = f(\underline{x}_i, \underline{\theta})$.

2.3.1.4 *Additional Penalty Terms*

Additional penalty terms may be incorporated to improve the ability of the network to interpolate between unseen data points. The most important of these is the complexity penalty, discussed below. However, there are a number of functions

of the network coefficients that may be added to any of the above loss functions to "smooth" the network output; these are often called *roughness* penalties.

In addition to improving the ability to interpolate, a roughness penalty can also improve network input-output stability, such that small variations in network input produce small variations in network output over the entire range of operating conditions. Any of the following, for example, may be used as a roughness penalty:

- Sum of squares of coefficient magnitudes
- Sum of squares of network gradients with respect to the inputs
- Minus the log of the prior density function of the network parameters

Details concerning the implementation of some specific roughness penalties are discussed in Sect. 2.3.4.2.

2.3.2 Model Selection Criterion

A.R. Barron [11] has given general conditions such that the minimum mean integrated squared error for an MLP neural network with one hidden layer will be bounded by

$$O\left(\frac{1}{n}\right) + O\left(\frac{nd}{N}\log N\right) \qquad 2:34$$

where O( ) represents "order of ( )," n is the number of elements, d is the dimensionality (number of coefficients per node), and N is the sample size (number of training exemplars); nd, therefore, is the number of coefficients in the network. The first term in Eq. 2:34 bounds the approximation error, which decreases as network size increases. The second term in Eq. 2:34 bounds the estimation error, which represents the error that will be encountered on unseen data due to overfitting of the training database; it is caused by the error in estimating the coefficients. Estimation error, unlike approximation error, increases as network size increases.

Pre-structured networks, because they often have excessive internal degrees of freedom, are prone to overfit training data, resulting in poor performance on unseen data. Additionally, because of a large number of network coefficients, optimization of pre-structured networks tends to be a slow and computationally intensive process. Without algorithms that learn the structure, the analyst often must resort to guesswork or trial and error if network complexity is to be reduced.

Improvements in network performance on unseen data can be made if one incorporates into the optimization algorithm modeling criteria that allow the

network structure to grow to a just-sufficient level of complexity. Although this technique requires additional effort to search for an optimal structure, the overall network generation time is, in general, greatly reduced due to the reduction in the number of coefficients.

Two decades of research have gone into this topic. In Ukraine, Ivakhnenko [41] introduced the *Group Method of Data Handling* (GMDH). With GMDH, the loss function is squared-error, and overfitting is kept under control by means of cross-validation testing that employs independent subsets (groups) of the database for fitting and selection. GMDH is a satisfactory approach when sufficient data are available. Usually, however, the quantity and variety of the available data are limited by operational considerations, and it is desirable to use all of the data in the fitting process. In Japan, Akaike [4] introduced an information theoretic criterion (AIC) that uses all of the data and incorporates a penalty term for overfit control. Akaike's criterion is one of several that take the form

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) \; + \; C\frac{K}{N} \qquad 2:35$$

where K in this context is the number of non-zero coefficients in the model, N is the number of data vectors in the database, and C is a constant. Since the second term does not depend on the network coefficient values, model selection criteria of the form shown in Eq. 2:35 are often optimized one term at a time.

Akaike's information criterion and subsequent criteria introduced by Schwarz [63] and Rissanen [59] require the loss functions to take the form of a minus log-likelihood. When the loss function takes this form, the AIC is given by Eq. 2:35 with C = 1, and the simplest forms of Schwarz's information criterion "B" (BIC) and Rissanen's minimum description length (MDL) criteria are given by Eq. 2:35 with $C = \frac{1}{2}\log N$. Note that these criteria are applicable to both squared-error loss (for function estimation with a Gaussian error model) and logistic loss (for class probability estimation).

The AIC, BIC, and MDL criteria depend explicitly on an assumed probability model to yield the likelihood expressions. However, other criteria of the form of Eq. 2:35 can be justified by the principle of *predicted squared error* (PSE) [5] [45], defined below, or the principle of complexity regularization [9].

To use the AIC or MDL criteria in the squared-error case, the loss function is recast in the form of a minus log-likelihood for a Gaussian model. This may be written for the single-input case as

$$d(y_i, s_i) = \frac{| y_i - s_i |^2}{2\sigma^2} + \frac{1}{2} \log 2\pi\sigma^2 \qquad 2:36$$

where σ^2 is a constant that may be regarded as the variance of the error in the Gaussian model.† The constant σ^2 could be replaced with its maximum likelihood estimate

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} | y_i - s_i |^2 \qquad 2:37$$

which leads to a criterion of the form

$$\frac{1}{2N} + \frac{1}{2} \log \hat{\sigma}^2 + \frac{1}{2} \log 2\pi + C\frac{K}{N} \qquad 2:38$$

with $C = 1$ or $C = \frac{1}{2} \log N$ for the AIC and MDL, respectively. Notice that the first and third term are constants and do not depend on the network structure or coefficients; these constant terms can be removed from the equation to yield a criterion of the form

$$\frac{1}{2} \log \hat{\sigma}^2 + C\frac{K}{N} \qquad 2:39$$

A different $\hat{\sigma}^2$ is obtained for each candidate network model. However, choices for $\hat{\sigma}^2$ which depend on the candidate model have two serious drawbacks: (1) these criteria depend explicitly on the assumed family of the error distribution (Gaussian), and (2) they do not account for potential error due to differences between the model structure and the actual system; thus, they tend to favor complicated overfit models.

As an alternative, it is better to use a prior estimate, σ_p^2, of the model error variance that does not depend on the candidate models. Barron showed that even when a prior estimate σ_p^2 is not extremely accurate, the criterion can still prove

---

† Eq. 2:31 may be extended for multiple outputs as follows:

$$d(\underline{y}_i, \underline{s}_i) = \sum_{j=1}^{J} \frac{| y_{ij} - s_{ij} |^2}{2\sigma_j^2} + \frac{1}{2} \log 2\pi\sigma_j^2$$

where σ_j^2 is a constant that may be regarded as the variance of the error of output j.

useful [5].† By inserting Eq. 2:36 into the criterion of Eq. 2:35, multiplying by $2\sigma_p^2$ and ignoring a constant, it can be seen that minimizing Eq. 2:35 is the same as minimizing

$$\frac{1}{N}\sum_{i=1}^{N} | y_i - s_i |^2 + 2\sigma_p^2 C \frac{K}{N} \qquad 2:40$$

With the value of C = 1, this represents the PSE criterion. This criterion, unlike the general AIC, is appropriate even when the error distributions are non-Gaussian [5]. For $C = \frac{1}{2} \log N$, the terms in Eq. 2:40 become the leading terms of the complexity regularization criterion derived by Barron [9].

For the classification or conditional-probability estimation problem, one may use the AIC or MDL criterion of Eq. 2:35 with the logistic-loss function. Since it has been shown already that the logistic-loss function takes the form of a minus log-likelihood, no modification to Eq. 2:35 is required, and the task becomes one of minimizing

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) + C \frac{K}{N} \qquad 2:41$$

where the distortion function, d(·), in Eq. 2:39, is the logistic-loss function of Eq. 2:25.

By minimizing the constrained loss functions, Eqs. 2:40 and 2:41, instead of their unconstrained predecessors, Eqs. 2:23 and 2:25, network complexity can be appropriately penalized so that overfit is avoided. One may follow the same steps to modify a variety of objective functions.

The $C \frac{K}{N}$ term in the model selection criterion is called the *complexity penalty* and can be thought of as an additional term added to the loss function. The complexity penalty allows the loss function to account for both estimation error and approximation error. By adding the *roughness penalty* to the loss function, i.e.

Loss = *distortion function* + *complexity penalty* + *roughness penalty* 2:42

---

† If no value of σ_p^2 is known *a priori*, one can use, for instance, the conservative nearest-neighbor estimate of σ_p^2. The nearest-neighbor approximation consists of assuming that the output for each given data vector is to be estimated using the output value of the data vector closest to it in the data space; σ_p^2 may then be set equal to the variance of these estimates. After modeling, the predicted error of the model can be checked to verify that it is less than or equal to σ_p.

one has all that is needed to create a robust objective function that not only takes into account estimation and approximation error, but also function smoothness and input-output stability. It is important to note, however, that it is not possible to compute a gradient of the complexity penalty with respect to the network coefficients. Thus, the optimization strategy must use a heuristic search method while traversing the space of potential network structures.

2.3.3 Optimization Strategy

Having defined the structural building blocks for a generic artificial neural network and an appropriate objective function, we next turn to consideration of an efficient search strategy that will find the network structure and optimize the coefficients of that structure.

The optimization strategy proposed here is distinctive in two ways. First, only small subsets of network coefficients are optimized at a given time, thus reducing the dimensionality of the search space and improving the performance of the search algorithm. In most cases, it is sufficient to optimize only the coefficients of a single element while holding all other elements fixed. Ivakhnenko [42] was the first to propose this type of network construction. In his scheme, the coefficients of each element are optimized in such a way that *each element* attempts to solve the *entire* input-output mapping problem.

While Ivakhnenko's method is powerful, it can be improved upon. A second major distinction of the proposed optimization strategy consists of training the elements on a given layer *so that they work in linear combination with other elements in that layer* to minimize the objective function. This is accomplished using a technique inspired by the projection-pursuit algorithm of Friedman *et al.* [20] [27] [28] In this strategy, an additional set of "dummy" coefficients, $\beta_1, \ldots, \beta_k$, multiply the outputs of the n elements on a given layer (Fig. 2.8):

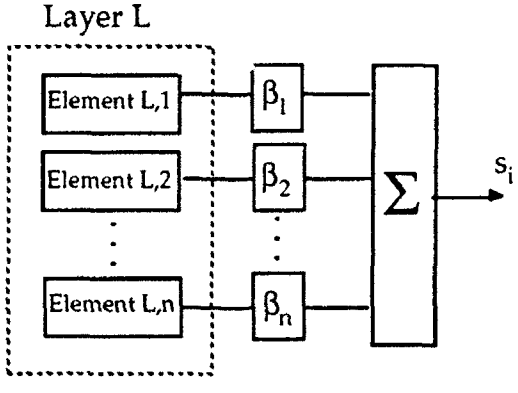


Figure 2.8: Projection-Pursuit Optimization Strategy

The coefficients of the node under consideration, along with the additional dummy coefficients, may be optimized together so that the weighted sum of element outputs minimizes the objective function. This has the effect of training each new element to work well in combination with the existing elements of a given layer. Additional nodes are added to a layer only as long as their additional complexity is justified.

Additionally, coefficients within the new elements may be built up or "carved" away using an objective function that contains a complexity penalty; the complexity penalty allows only terms which contribute significantly to network performance to be retained.

Entire layers may be optimized following a strategy originally used by Adaptronics, Inc. in the 1970s and most recently suggested by Breiman and Friedman [20]. In this strategy, which is called "backfitting," each coefficient subset is improved by iterating the search algorithm a few steps while holding the rest of the coefficients fixed. This method is then repeated for another subset of network coefficients, etc. For neural-network-based estimation, the nodal elements become the logical choice for the coefficient subsets to be optimized, and a layer may be optimized by successively recursing through each nodal element, iterating the optimization algorithm a few times for each element. Breiman and Friedman showed that under appropriate conditions this method will yield the same coefficient values as are obtained via a successful global optimization of the same structure. Practical implementation of the backfitting strategy has an advantage in that only a small set of linear equations needs to be solved at any given time.

An example of the way in which backfitting may be applied can be illustrated using a network as defined in Fig. 2.8. Once the structure of the layer has been determined, the coefficients of element L,1 and the dummy coefficients, $\underline{\beta}$, are adjusted using one iteration of the search routine (see Section 2.3.4). Next the coefficients of element L,2 and $\underline{\beta}$ are adjusted using one iteration of the search routine. This process continues n times until the coefficients of element L,n have been adjusted. At this point, the process begins again with element L,1. The optimization routine continues until the optimization no longer improves performance significantly.

Another way backfitting can be used is during the search for network structure. Elements may be backfitted each time a new element is added, and the new element can be scored based on its performance in conjunction with the backfitted prior elements. In general, backfitting will increase training time, but it is a technique which can be used as often or as seldom as desired. Even when used to a small extent, backfitting can be a highly efficient way of optimizing larger sets of coefficients so that they work well together.

Once the structure of a given layer is determined, subsequent layers have the option of combining the layer outputs linearly using the $\underline{\beta}$ coefficients chosen above,

or they may go on and recombine the outputs in more complex ways if the improved performance justifies the additional complexity. Layers are added one at a time in this fashion until overall network growth stops. The stopping rule is that the constrained fitting criterion has reached a minimum.

2.3.4 Optimization Method

An iterative least-squares (ILS) method for optimizing the types of nonlinear networks described in this section will now be derived. The algorithm is iterative in the sense that multiple passes through the data are usually required to achieve convergence. It is a least-squares method in the sense that it minimizes a local quadratic approximation of the objective function; it does not, however, require that a squared-error distortion function be used or that the network equations be linear in the parameters.

2.3.4.1 *The ILS Algorithm*

The ILS algorithm consists of finding the local least-squares solution to a linearized version of the network (or other) function at each consecutive operating point (e.g., $\underline{\theta}$). Because the optimization strategy described in Section 2.3.3 consists of optimizing subsets of the coefficients, in particular those contained in a single network element, the entire network optimization task can be reduced to a series of single-element optimization tasks.

Let $\nabla \underline{f}(\underline{x}_i, \underline{\theta}_0)$ be the $J \times C$ gradient of the $C \times 1$ network output vector, $\underline{f}$, with respect to the $J \times 1$ element coefficient vector, $\underline{\theta}$, evaluated at $\underline{\theta}_0$ and abbreviated $\nabla \underline{f}_{\underline{\theta}_0}$.

$$\nabla \underline{f}_{\underline{\theta}_0} = \left[\begin{array}{cccc} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_1} & \cdots & \frac{\partial f_C}{\partial \theta_1} \\ \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_C}{\partial \theta_2} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial f_1}{\partial \theta_J} & \frac{\partial f_2}{\partial \theta_J} & \cdots & \frac{\partial f_C}{\partial \theta_J} \end{array} \right] \Bigg|_{\underline{\theta} = \underline{\theta}_0}$$

This gradient is used to make a local linear approximation of the network function about $\underline{\theta}_0$:

$$\underline{f}(\underline{x}_i, \underline{\theta}) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla \underline{f}_{\underline{\theta}_0})^T (\underline{\theta} - \underline{\theta}_0) \qquad 2:43$$

Becaise the general form of the method is to be iterative, we wish to find a $\Delta\underline{\theta}$ such that the iteration

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \mu\,\Delta\underline{\theta} \qquad 2.44$$

produces a minimum of the loss linearized about $\underline{\theta}_{old}$. If μ, the parameter that controls the step size, is taken to be unity, then $\Delta\underline{\theta} = \underline{\theta}_{new} - \underline{\theta}_{old}$. Taking $\underline{\theta}_0$ as $\underline{\theta}_{old}$, $\underline{\theta}$ as $\underline{\theta}_{new}$, Eq. 2:43 may be rewritten

$$\underline{f}(\underline{x}_i, \underline{\theta}) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla \underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta}) \qquad 2:45$$

Now, let $\nabla d(\underline{y}_i, \underline{f}_{i,0})$ and $\nabla^2 d(\underline{y}_i, \underline{f}_{i,0})$ be the $C \times 1$ gradient and $C \times C$ Hessian, respectively, of the distortion function with respect to the $C \times i$ vector of network outputs, $\underline{f}_i$, at observation, i, and evaluated at $\underline{f}_i = \underline{f}_{i,0}$. These are abbreviated ∇d_{f_0} and $\nabla^2 d_{f_0}$, respectively:

$$\nabla d_{f_0} = \left[\begin{array}{c} \frac{\partial d}{\partial f_1} \\ \frac{\partial d}{\partial f_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial d}{\partial f_C} \end{array}\right] \Bigg|_{\underline{f} = \underline{f}_0}$$

$$\nabla^2 d_{f_0} = \left[\begin{array}{cccc} \frac{\partial^2 d}{\partial f_1 \partial f_1} & \frac{\partial^2 d}{\partial f_1 \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_1 \partial f_C} \\ \frac{\partial^2 d}{\partial f_2 \partial f_1} & \frac{\partial^2 d}{\partial f_2 \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_2 \partial f_C} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial^2 d}{\partial f_C \partial f_1} & \frac{\partial^2 d}{\partial f_C \partial f_2} & \cdots & \frac{\partial^2 d}{\partial f_C \partial f_C} \end{array}\right] \Bigg|_{\underline{f} = \underline{f}_0}$$

Where C is the number of network outputs.

Because restrictions are put on the objective function such that it is convex and everywhere twice-differentiable, the gradient and Hessian are known everywhere and can be used to make a local quadratic approximation of the loss function in the vicinity of the current network output, $\underline{s}_0$:

$$d(\underline{y}_i, \underline{f}_i) \equiv d(\underline{y}_i, \underline{f}_0) + (\nabla d_{\underline{f}_0})^T(\underline{f}_i - \underline{f}_0) + \frac{1}{2}(\underline{f}_i - \underline{f}_0)^T(\nabla^2 d_{\underline{f}_0})(\underline{f}_i - \underline{f}_0) \qquad 2:46$$

Eq. 2:45 may be substituted into Eq. 2:46 to yield an approximation to the i^{th} component of the objective function in terms of $\Delta\underline{\theta}$:

$$d(\underline{y}_i, \underline{f}_i) \equiv$$

$$d(\underline{y}_i, \underline{f}_0) + (\nabla d_{\underline{f}_0})^T(\nabla \underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T\left((\nabla \underline{f}_{\underline{\theta}_0})(\nabla^2 d_{\underline{s}_0})(\nabla \underline{f}_{\underline{\theta}_0})^T\right)(\Delta\underline{\theta}) \qquad 2:47$$

The total empirical loss, J, may then be calculated by summing the approximation of the distortion function over all observations:

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{f}_0) + \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{f}_0})^T(\nabla \underline{f}_{\underline{\theta}_0})^T(\Delta\underline{\theta})$$

$$+ \frac{1}{2N}\sum_{i=1}^{N} (\Delta\underline{\theta})^T\left((\nabla \underline{f}_{\underline{\theta}_0})(\nabla^2 d_{\underline{f}_0})(\nabla \underline{f}_{\underline{\theta}_0})^T\right)(\Delta\underline{\theta}) \qquad 2:48$$

or

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}^T(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T\underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad 2:49$$

where

$$\underline{\underline{A}} = \frac{1}{N}\sum_{i=1}^{N} (\nabla \underline{f}_{\underline{\theta}})(\nabla^2 d_{\underline{f}})(\nabla \underline{f}_{\underline{\theta}})^T \qquad 2:50$$

and

$$\underline{b} = \frac{1}{N}\sum_{i=1}^{N} (\nabla \underline{f}_{\underline{\theta}})(\nabla d_{\underline{f}}) \qquad 2:51$$

It is now possible to calculate the gradient of the empirical loss function with respect to the coefficient vector $\underline{\theta}$:

$$\nabla J_{\underline{\theta}} = \underline{b} + \underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad 2:52$$

Because the loss function is required to be convex, the minimum is found at the point where the gradient is zero. Thus, Eq. 2:52 may be solved for $\Delta\underline{\theta}$ by the choice

$$(\Delta\underline{\theta}) = -\underline{\underline{A}}^{-1}\,\underline{b} \qquad 2:53$$

Thus

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\,\underline{\underline{A}}^{-1}\,\underline{b} \qquad 2:54$$

is the desired iteration.

Recall that each element is the composition of a transformation h(z), with a linearly parameterized expansion (Fig. 2.4):

$$f(\underline{x},\underline{\theta}) = h\left(\sum_{j=0}^{J} \theta_j\,\Phi\,(\underline{k}_j,\underline{x})\right) \qquad 2:55$$

where $\Phi\,(\underline{k}_0,\underline{x})$ is unity by definition. Eq. 2:55 may be rewritten for clarity as

$$f(\underline{x},\underline{\theta}) = h(z) \qquad 2:56$$

where

$$z = \sum_{j=0}^{J} \theta_j\,\Phi\,(\underline{k}_j,\underline{x}) \qquad 2:57$$

And the partial $\nabla f_{\underline{\theta}}$ may be computed via the chain rule as follows:

$$\frac{\partial f}{\partial \underline{\theta}} = \frac{\partial h}{\partial \underline{\theta}} = \frac{dh}{dz}\,\frac{\partial z}{\partial \underline{\theta}} = \frac{dh}{dz}\,\Phi_j(\underline{k}_j,\underline{x}) \qquad 2:58$$

Thus, to use the ILS optimization technique, the analyst must provide the following:

- Analytic forms of the first and second partials of the objective function with respect to the network outputs, ∇d_f and $\nabla^2 d_f$.
- An analytic form for the first derivative of the post-transformation h(z).

Given these three pieces of information, Eq. 2:58 may be used to compute $\nabla \underline{f}_{\underline{\theta}}$, and Eqs. 2:50 - 2:54 may be used to compute the ILS update.

Table 2.2 gives gives a summary of the variables that are used in the solution of the ILS equations.

Table 2.2: Summary of ILS Variables

VARIABLE	DESCRIPTION	DIMENSION
$\Delta\underline{\theta}$	Coefficient update vector.	$J \times 1$
$\underline{f}$ or $\underline{s}$	Network output vector.	$C \times 1$
d_i	The distortion function calculated at observation i.	Scalar
J	The objective function; the sum of the distortion function over all observations.	Scalar
$\nabla \underline{f}_{\underline{\theta}}$	Gradient of the network output vector, $\underline{f}$, with respect to the coefficient vector, $\underline{\theta}$.	$J \times C$
∇d_f	Gradient of the distortion function with respect to the network output vector, $\underline{f}$.	$C \times 1$
$\nabla^2 d_f$	Hessian of the distortion function with respect to the network output vector, $\underline{f}$.	$C \times C$
$\underline{b}$	Computed gradient of the objective function with respect to the coefficients.	$J \times 1$
$\underline{\underline{A}}$	Pseudo-Hessian of the objective function with respect to the coefficients.	$J \times J$

2.3.4.2 *Incorporation of Additional Penalty Terms*

Often it is desirable to incorporate additional penalty terms in the objective function (Sections 2.3.1 and 2.3.2). These additional penalty terms may be divided into three categories:

(1) functions of the network structure and database size (complexity penalty),

(2) additional functions of the network output, and

(3) functions of the network coefficients.

As mentioned above, the first type of penalty term does not involve the coefficients of the network; therefore, partial derivatives cannot be computed and the penalty term must be minimized by a heuristic search of the space of potential structure.

Penalty terms that are functions of the network output may be handled by incorporating these functions in the computation of the $\underline{\underline{A}}$ and $\underline{b}$ matrices. This is possible since the partial derivatives, $\nabla d_{\underline{s}}$ and $\nabla^2 d_{\underline{s}}$, exist for this type of function. This method may be demonstrated by the following example.

Assume that the network is interrogated with time-series data, and that the output is required to match a desired response for only a portion, M, of the samples. Also assume that for subsequent samples, there is no deterministic network response that is desired; and it is important that, in any event, the network response does not become excessively large during subsequent samples. This situation is shown in Figure 2.9:

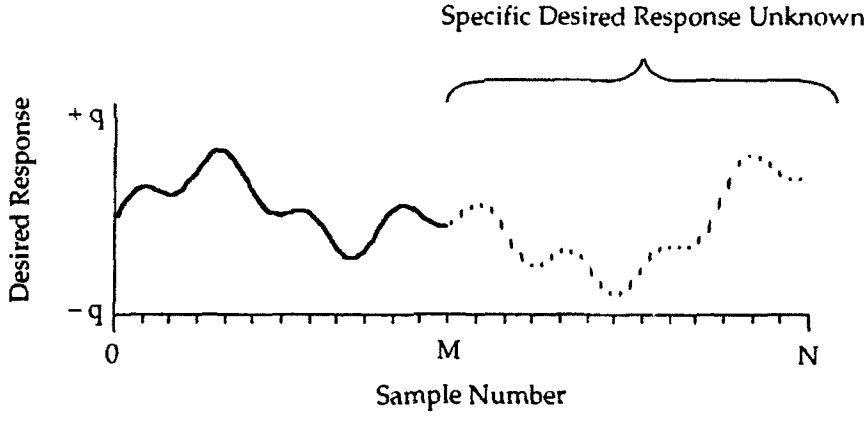


Figure 2.9: A Desired Network Response that Requires an Additional Penalty Term

If the network is trained using only the first M samples, the response of the network subsequent to sample M may grow without bound, because the response is not constrained in this interval. However, if the network is trained to provide a response of zero between samples M and N, its performance in the region 0 to M will be degraded due to the severe requirement placed on the fit in the region M to N.

One method for handling this case is to divide the objective function into two parts. The squared-error distortion function may be used prior to sample N:

$$d(\underline{y}_i, \underline{s}_i) = |\underline{y}_i - s_i|^2 \qquad 0 \le i \le M \qquad 2:59$$

In the region where the specific desired response is unknown ($M < i \leq N$), however, the squared-error distortion function is not appropriate. A modified squared-error distortion function may be used instead to penalize the network only when its response falls outside some range $\pm q$:

$$d(\underline{y}_i, \underline{s}_i) = \begin{cases} 0 & \text{if } |\underline{y}_i - s_i| \leq q \\ K\,|\underline{y}_i - s_i - q|^2 & \text{if } |\underline{y}_i - \underline{s}_i| > q \end{cases} \qquad M < i \leq N \qquad 2:60$$

K is a user-specified constant that controls the "rigidity" of the $\pm q$ boundary. Thus, a large K will heavily penalize any excursion of the network response beyond the boundaries, whereas a small K allows the network response to exceed the boundaries by small amounts without significant penalty.

Note that the distortion function of Eq. 2:60 is convex and everywhere twice differentiable. Therefore, the $\underline{\underline{A}}$ and $\underline{b}$ matrices (Eqs. 2:50 - 2:51) may be computed as before, keeping in mind that after sample number M in the summation, the alternative forms of the objective function should be used.

The above example illustrates the incorporation of alternative or additional penalty terms that are functions of the network output. If, however, the penalty terms are direct functions of the network coefficients, a slightly different approach must be used. Assume

$$d'(\underline{y}_i, s_i) = d(\underline{y}_i, s_i) + g(\underline{\theta}) \qquad 2:61$$

where $g(\bullet)$ is convex and twice differentiable everywhere. Since $g(\bullet)$ is not a function of the network output, Eqs. 2:50 - 2:51 cannot be used to compute $\underline{\underline{A}}$ and $\underline{b}$ directly as before. Instead, the equations for $\underline{\underline{A}}$ and $\underline{b}$ must be modified to account for the additional term.

It has already been shown that the portion of the objective function corresponding to the $d(\bullet)$ term may be represented by a second-order Taylor series expansion

$$J_1(\underline{\theta}) = J_1(\underline{\theta}_0) + \underline{b}(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T \underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad 2:62$$

with $\underline{\underline{A}}$ and $\underline{b}$ matrices defined in Eqs. 2:50 - 2:61. The portion of the objective function, $J_2(\underline{\theta})$, corresponding to the additional term, $g(\underline{\theta})$, may also be expanded in a similar manner

$$J_2(\underline{\theta}) = J_2(\underline{\theta}_0) + \nabla_{\underline{\varepsilon}_{\underline{\theta}}}(\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T (\nabla^2_{\underline{\varepsilon}_{\underline{\theta}}})(\Delta\underline{\theta}) \qquad 2:63$$

where $\nabla_{g_{\underline{\theta}}}$ and $\nabla^2_{g_{\underline{\theta}}}$ are the gradient and Hessian matrices of the additional term, $g(\bullet)$, with respect to the coefficient vector, $\underline{\theta}$. Combining Eqs. 2:62 and 2:63 to obtain the sum of the two objective functions, one may obtain new $\underline{\underline{A}}$ and $\underline{b}$ matrices:

$$\underline{\underline{A}}' = \underline{\underline{A}} + \nabla^2_{g_{\underline{\theta}}} \qquad 2:64$$

$$\underline{b}' = \underline{b} + \nabla_{g_{\underline{\theta}}} \qquad 2:65$$

The ILS update may now proceed as before with these new matrices. Note that this type of additional penalty term requires that the analyst provide an analytic form of the first and second partial derivatives of the additional objective function term with respect to the network coefficients, $\nabla_{g_{\underline{\theta}}}$ and $\nabla^2_{g_{\underline{\theta}}}$.

The following example illustrates the use of a penalty term that is a function of the coefficients. Suppose one wanted to put a constraint on the magnitudes of the network coefficients; one way of accomplishing this would be to use the following distortion function

$$d'(\underline{y}_i, s_i) = d(\underline{y}_i, s_i) + K\,\underline{\theta}\,\underline{\theta}^T \qquad 2:66$$

where K is a user-defined constant associated with the amount of penalty to be applied to the coefficient magnitude term. In this case, following Eqs. 2:64 and 2:65, the ILS update should make use of $\underline{\underline{A}}'$ and $\underline{b}'$ matrices defined by

$$\underline{\underline{A}}' = \underline{\underline{A}} + 2K\underline{\underline{I}} \qquad 2:67$$

$$\underline{b}' = \underline{b} + 2K\underline{\theta} \qquad 2:68$$

Note that, in this example, summation over the observations is not required for the computation of the additional term in the distortion function, because $K\underline{\theta}\underline{\theta}^T$ is independent of the observation number, i.

2.3.4.3 *Relationship to Other Optimization Techniques*

ILS is closely related to a number of other optimization techniques. First, consider the special case of a linear nodal element, a quadratic objective function, and $\underline{\theta}_{old}$ set to zero. In this case, Eq. 2:54 becomes

$$\underline{\theta}_{new} = \mu\,\underline{\underline{R}}^{-1}\,\underline{p} \qquad 2:69$$

where $\underline{\underline{R}}$ is the correlation matrix of the input vector, $\underline{x}$, and $\underline{p}$ is the cross-correlation vector between the input vector, $\underline{x}$, and the desired response, $\underline{y}$. If $\mu = 1$,

then Eq. 2:69 forms the Wiener-Hopf equations [38] and provides an optimal least-squares solution in a single step.

If the nodal element is nonlinear in its coefficients, and the distortion function is squared-error, and $\mu = 1$, these equations then correspond to the Gauss-Newton optimization method; in fact, Gauss' fundamental contribution to Newton's method was to simplify the Hessian of the objective function by using a linear approximation of the function being optimized (Eq. 2:43). A full Newton method, on the other hand, would require the calculation of a complete Hessian via the incorporation of terms related to second partial derivative of the element output with respect to the coefficients.

Gradient-descent algorithms, including least-mean-squares (LMS), are also very similar to the ILS algorithm except that they ignore second derivative information altogether. If the quadratic term in Eq. 2:46 is dropped, Eq. 2:48 becomes,

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_0) + \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{s}_0})^T (\nabla_{f_{\underline{\theta}}})^T (\Delta\underline{\theta}) \qquad 2:70$$

or

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}^T(\Delta\underline{\theta}) \qquad 2:71$$

where $\underline{b}$ is defined in Eq. 2:51.

We can now calculate the gradient of the cost function, J, with respect to the coefficient vector, $\underline{\theta}$.

$$\nabla J_{\underline{\theta}} = \frac{\partial J}{\partial \underline{\theta}} = \underline{b} = \frac{1}{N}\sum_{i=1}^{N} (\nabla_{f_{\underline{\theta}}})(\nabla d_{\underline{s}_0}) \qquad 2:72$$

From Eq. 2:72, it can be seen that the network coefficients may be adjusted in the direction of steepest descent by,

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\underline{b} \qquad 2:73$$

where μ is the size of the step at each iteration. For a squared-error cost function,

$$\nabla d_{\underline{s}_i} = \frac{\partial}{\partial \underline{s}_i} | \underline{y}_i - \underline{s}_i |^2 = -2(\underline{y}_i - \underline{s}_i) \qquad 2:74$$

and for a linear filter,

$$\nabla_{f_{\underline{\theta}}} = \underline{x} \qquad 2:75$$

So the coefficient update in Eq. 2:64 becomes:

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \mu \frac{1}{N} \sum_{i=1}^{N} 2 (\underline{y}_i - \underline{s}_i)^T \underline{x} \qquad 2:76$$

Comparing Eq. 2:73 to Eq. 2:54 one sees that by ignoring the distortion function curvature information, the term $\underline{\underline{A}}^{-1}$ is dropped from the weight update. While this simplification greatly reduces the number of computations required to compute each iteration, the convergence rates for gradient-descent algorithms is typically very slow.

2.3.4.4 *Regularization*

Experience has shown that, for non-quadratic objective functions, Newton methods may be unreliable, especially if the coefficients are initialized far from the minimum. This is because techniques for solving the system of equations in 2:53 break down when the pseudo-Hessian matrix, $\underline{\underline{A}}$, becomes singular or nearly singular. Regularization techniques are methods that can be used to ensure that $\underline{\underline{A}}$ is positive-definite. Many of these techniques can accomplish this and still provide an iteration that is only slightly different than the optimal Newton direction. One such technique is the Levenberg-Marquardt (LM) method [43] [48]. LM can be incorporated into the ILS algorithm in a straightforward fashion..

Because the matrix $\underline{\underline{A}}$, as defined by Eq. 2:50, is square, it is also (by definition) positive-semi-definite. Thus, one way of ensuring that $\underline{\underline{A}}$ is positive-definite is simply to add some small positive values to the diagonals. Thus, at each iteration, $\underline{\underline{A}}$ may be modified using one of the following methods [44] [57] [58]:

$$\underline{\underline{A}}' = \underline{\underline{A}} + \lambda \underline{\underline{I}} \qquad 2:77$$

or,

$$\underline{\underline{A}}' = \underline{\underline{A}} + \lambda \, \mathrm{diag}(\underline{\underline{A}}) \qquad 2:78$$

where λ is positive constant, $\underline{\underline{I}}$ is the identity matrix and $\mathrm{diag}(\underline{\underline{A}})$ denotes the matrix $\underline{\underline{A}}$ with all but its diagonal elements set to zero. When λ is large, the second term in the above equations dominates, and the iteration steps along the gradient (Eq. 2:64). When λ is small or zero, the first term in the above equations dominates, and the iteration becomes a Gauss-Newton iteration. There are a number of heuristic schemes for varying λ during the course of the search so that $\underline{\underline{A}}'$ remains positive-definite and the algorithm converges rapidly.

2.3.4.5 *Global Optimization*

Up to this point, we have only considered the optimization of single network elements. While, for many GMDH-based neural network paradigms this is all that is required, at times it may be desirable, once the network is constructed, to optimize globally all of the network coefficients. Global optimization can be accomplished by using the chain rule to propagate the gradient information through all the network elements.

Consider the generic "hidden" nodal as shown in Fig. 2.10, where the time delays have been dropped for notational convenience:

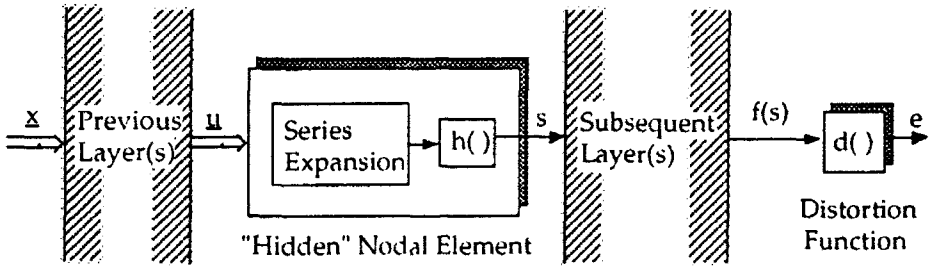


Figure 2.10: A "Hidden" Nodal Element

To compute the gradient of the network output, f, with respect to the coefficients, $\underline{\theta}$, of the hidden element, the chain rule must be used:

$$\frac{\partial f}{\partial \underline{\theta}} = \frac{\partial f}{\partial s}\frac{\partial s}{\partial \underline{\theta}} \qquad 2:79$$

The partial derivative, $\frac{\partial s}{\partial \underline{\theta}}$, may be computed by summing the partials of all the paths that the variable, s, may take through the subsequent layers. To illustrate this, assume the structure of the subsequent layers is as shown in Fig. 2.11. In the figure, $\underline{s}$ is the input vector to the penultimate network layer, and f(s) is the final network output. Notice that there may be other inputs to layer L–1; however, it is not necessary to know the nature of these inputs to compute the gradient of f(s) with respect to the single intermediate variable, s. Summing up the paths that s takes through the subsequent layers, the gradient may be calculated as a sum of chain-rule terms:

$$\frac{\partial f}{\partial s} = \sum_{i=1}^{n} \frac{\partial f}{\partial l_n}\frac{\partial l_n}{\partial s} \qquad 2:80$$

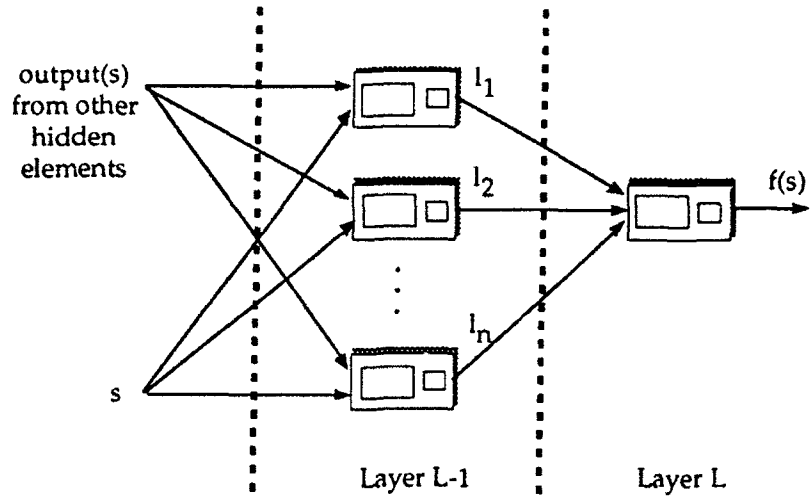


Figure 2.11: Interconnections on Final Network Layers

Thus, three analytic expressions are now required to perform ILS with global optimization:

- an analytic form of the first and second partials of the objective function with respect to the network outputs, ∇d_{s_0} and $\nabla^2 d_{s_0}$.
- an analytic form for the first derivative of the post-transformation h(z), and
- an analytic form for the gradient of an element output with respect to its input vector, $\nabla f_{\underline{x}}$.

Once this information is known, the gradient of the network output with respect to all coefficients, $\nabla_{f_{\underline{\theta}}}$, may be calculated using Eq. 2:79, and the ILS update step of Eq. 2:54 may be computed as before. It should be noted that $\nabla f_{\underline{x}}$ may also be used to compute some forms of the roughness penalty as described in Section 2.3.2.

2.3.4.6 *Elements with Feedback*

If the network contains feedback, calculating the derivatives becomes more complicated, because the inputs to a given nodal element may, in fact, depend on prior values of the outputs of the same element. Hence, the inputs are functions of the parameters of the element, and the core transformation is no longer linear in the parameters. In this case we must add an additional chain-rule term to the derivative calculation of Eq. 2:58. Thus, for a single input element, Eq. 2:58 becomes

$$\frac{\partial h}{\partial \theta_j} = \frac{dh}{dz}\,\Phi_j(\underline{k}_j, \underline{x}) + \sum_{j=1}^{J} \theta_j \Phi_j(\underline{k}_j, \underline{x}) \sum_{d=1}^{D} \frac{\Phi_j'(\underline{k}_{j,d}, \underline{x}_d)}{\Phi_j(\underline{k}_{j,d}, \underline{x}_d)} \frac{\partial x_d}{\partial \theta_j} \qquad 2:81$$

where $\Phi'_j(\underline{k}_{j,d}, \underline{x}_d)$ is the partial of the j,d term of the series expansion with respect to the input x_d. The derivatives of the inputs are readily calculated because the inputs to the current nodal element are outputs from another element, and we have provided an algorithm for calculating the derivatives for the output of an element. Notice that the same information is required to compute Eq. 2:81 as is required to perform global optimization.

In some cases, analytic forms of the gradients of the network or the objective function may not be available (Fig. 2.2). In these cases, the ILS method cannot be used and a direct search method such as simulated annealing, Powell search, or *GR/GARS* must be used. A description of *GR/GARS* is provided in Appendix B.

2.3.4.7 *Recursive Forms*

It is possible to update the network coefficients recursively (at each sample interval) using a recursive iterative least-squares (RILS) technique. Updating the network at each sample interval has a number of advantages including: (1) computationally efficient recursive coefficient updates may be more suitable for on-line network training; and (2) in some contexts, the process being modeled by the network may be non-stationary; therefore, it may be desirable for the network parameters to be adapted over time [72].†

The key to the development of a recursive ILS is to approximate the $\underline{\underline{A}}$ and $\underline{b}$ matrices in a way that does not require the summation over the observations (Eqs. 2:50 and 2:51). This can be accomplished as follows:

$$\tilde{\underline{\underline{A}}}_N = (1-\gamma)\,\tilde{\underline{\underline{A}}}_{N-1} + \gamma(\nabla f_{\underline{\theta}_N})(\nabla^2 d_{f_N})(\nabla f_{\underline{\theta}_N})^T \qquad 2:82$$

and

$$\tilde{\underline{b}}_N = (1-\gamma)\,\tilde{\underline{b}}_{N-1} + \gamma(\nabla f_{\underline{\theta}_N})(\nabla d_{f_N}) \qquad 2:83$$

where N denotes the number of iterations.

---

† It is important to note that apparent time-varying characteristics of a process often are caused by unmodeled nonlinearities. If the neural networks are originally trained to model these nonlinearities properly, then on-line adaptation may not be required. A *learning* system (a system that includes a modeled process, a neural network, and a network synthesis algorithm) consists generally of a functional form capable of representing a complex process response over a wide range of operating conditions, whereas an *adaptive* system is typically less capable of a wide range of representation until it modifies its parameters. The line between an *adaptive* system and a *learning* system is fine, and there are potential situations where a fixed network cannot be trained to model a process over the entire operating range. (Note that "adaptation" has sometimes been associated with knowledge gained while in operation; however, current usage also employs "on-line learning" to make this distinction when a neural network is designed to be trained on-line.)

Note that if the time-varying gain, γ, is chosen to be 1/N at each sample, and if the network coefficients are *not* updated during the process of recursively computing Eqs. 2:82 and 2:83, then after N iterations the resulting $\tilde{\underline{\underline{A}}}$ and $\tilde{\underline{b}}$ matrices are identical to the $\underline{\underline{A}}$ and $\underline{b}$ matrices of Eqs. 2:50 and 2:51. If, however, the network coefficients are updated at each observation, then Eqs. 2:82 and 2:83 are not equivalent to Eqs. 2:50 and 2:51 because the matrices contain gradients that were computed using previous versions of the network coefficients.

One problem with Eqs. 2:82 and 2:83 is that older gradient and Hessian information is allowed to contribute equally in the computation of the $\underline{\underline{A}}$ and $\underline{b}$ matrices. This is a problem because (1) the estimated system parameters are improving with each iteration, and therefore the more recent gradient estimates are more accurate, and 2) if the system is varying with time, and the observations occur in chronological order, prior gradient and Hessian information may be obsolete.

One way to place more emphasis on recent observations is to choose γ to be greater than 1/N. A more computationally convenient method involves introducing a *forgetting factor*, λ, directly into the computations of the $\underline{\underline{A}}$ and $\underline{b}$ matrices. In this case, Eqs. 2:50 and 2:51 become

$$\underline{\underline{A}} = \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla^2 d_{f_i})(\nabla \underline{f}_{\underline{\theta}_i})^T \qquad 2:84$$

and

$$\underline{b} = \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla d_{f_i}) \qquad 2:85$$

Now the $\underline{\underline{A}}$ and $\underline{b}$ matrices can be updated recursively as follows:

$$\tilde{\underline{\underline{A}}}_N = \lambda \tilde{\underline{\underline{A}}}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N})(\nabla^2 d_{f_N})(\nabla \underline{f}_{\underline{\theta}_N})^T \qquad 2:86$$

and

$$\tilde{\underline{b}}_N = \lambda \tilde{\underline{b}}_{N-1} + (\nabla \underline{f}_{\underline{\theta}_N})(\nabla d_{f_N}) \qquad 2:87$$

With RILS, the new set of coefficients is found by solving the same set of equations used to compute the ILS coefficients (Eq. 2:53):

$$\underline{\theta}_N = \underline{\theta}_{N-1} - \tilde{\underline{\underline{A}}}_N^{-1} \tilde{\underline{b}}_N \qquad 2:88$$

Computation of Eq. 2:88 can be sped up significantly if Eq. 2:86 is modified to yield a direct recursive relationship between $\underline{\underline{\tilde{A}}}_N^{-1}$ and $\underline{\underline{\tilde{A}}}_{N-1}^{-1}$. To obtain this recursion, first make the following assignments:

$$P_{N-1} = \underline{\underline{\tilde{A}}}_{N-1}^{-1} \qquad 2:89a$$

$$X = (\nabla f_{\underline{\theta}_N}) \qquad 2:89b$$

$$Y = (\nabla^2 d_{f_N}) \qquad 2:89c$$

$$Z = (\nabla f_{\underline{\theta}_N})^T \qquad 2:89d$$

Inverting both sides of Eq. 2:86, one obtains

$$P_N = \underline{\underline{\tilde{A}}}_N^{-1} = [\lambda P_{N-1}^{-1} + XYZ]^{-1} \qquad 2:90$$

The right-hand side of Eq. 2:90 can be rearranged according to the *matrix inversion lemma* which is reproduced below without proof:

$$[W^{-1} + XYZ]^{-1} = W - WX\left(ZWX + Y^{-1}\right)^{-1}ZW \qquad 2:91$$

By noting that $W = \frac{1}{\lambda}P$, $P = P^T$, and $X = Z^T$, the matrix inversion lemma may be used to rewrite Eq. 2:90 as follows

$$P_N = \frac{1}{\lambda}P_{N-1} - \frac{1}{\lambda^2}P_{N-1}X\left(\frac{X^TPX}{\lambda} + Y^{-1}\right)^{-1}X^TP^T \qquad 2:92$$

or

$$P_N = \frac{1}{\lambda}\left(P_{N-1} - QS^{-1}Q^T\right) \qquad 2:93$$

where

$$Q = P_{N-1}(\nabla f_{\underline{\theta}_N}) \qquad 2:94$$

and

$$S = (\nabla f_{\underline{\theta}_N})^T P_{N-1} (\nabla f_{\underline{\theta}_N}) + \lambda I (\nabla^2 d_{f_N})^{-1} \qquad 2:95$$

and I is the identity matrix. Substituting Eq. 2:89a into Eq. 2:88 one obtains the update equations

$$\underline{\theta}_N = \underline{\theta}_{N-1} - P_N \tilde{\underline{b}}_N \qquad 2:96$$

where P_N is computed from Eq. 2:93. Note that the whereas the update formerly required the inversion of the $J \times J$ $\underline{\underline{A}}$ matrix, it now requires the inversion of two $C \times C$ matrices, S and $\nabla^2 d_{f_N}$, where C is the number of system outputs. Typically, the number of outputs will be significantly smaller than the number of parameters, J. Additionally, the inversion of $\nabla^2 d_{f_N}$ is frequently trivial since it is often either a diagonal or an inverted matrix.†

As might be expected, RILS reduces to the popular recursive least-squares (RLS) algorithm if the distortion function is squared-error:

$$d_i = \frac{1}{2}(\underline{y}_i - \underline{f}_i)^T(\underline{y}_i - \underline{f}_i) \qquad 2:97$$

and, the model being optimized is linear in the parameters:

$$f_i = (\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{i-1} \qquad 2:98$$

For this special case, Eq. 2:85 may be rewritten as

$$\begin{aligned}
\tilde{\underline{b}}_N &= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(y_i - f_i) \\
&= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\left(\underline{y}_i - (\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{N-1}\right) \\
&= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})\, \underline{y}_i \;+\; \frac{1}{N}\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla \underline{f}_{\underline{\theta}_i})(\nabla \underline{f}_{\underline{\theta}_i})^T \underline{\theta}_{N-1}
\end{aligned}$$

---

† A common squared-error criterion for multiple outputs is $d = \frac{1}{2}(\underline{y} - \underline{f})^T \Lambda^{-1} (\underline{y} - \underline{f})$, where Λ is the covariance matrix of the estimation errors. Note, that when this is the case, $(\nabla^2 d_f)^{-1} = \Lambda$ and no inversion is required.

$$= -\sum_{i=1}^{N} \lambda^{(N-i)} (\nabla f_{\underline{\theta}_i}) \underline{y}_i + \underline{\underline{\tilde{A}}}_N \underline{\theta}_{N-1} \qquad 2:99$$

Substituting Eq. 2:99 into Eq. 2:88 one obtains

$$\underline{\theta}_N = \underline{\theta}_{N-1} + \underline{\underline{\tilde{A}}}_N^{-1} \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla f_{\underline{\theta}_i}) \underline{y}_i + \underline{\theta}_{N-1}$$

$$= \underline{\underline{\tilde{A}}}_N^{-1} \sum_{i=1}^{N} \lambda^{(N-i)} (\nabla f_{\underline{\theta}_i}) \underline{y}_i$$

$$= \underline{\underline{\tilde{A}}}_N^{-1} \left(\lambda \sum_{i=1}^{N-1} \lambda^{(N-1-i)} (\nabla f_{\underline{\theta}_i}) \underline{y}_i + (\nabla f_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \underline{\underline{\tilde{A}}}_N^{-1} \left(\lambda \underline{\underline{\tilde{A}}}_{N-1} \underline{\theta}_{N-1} + (\nabla f_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \underline{\underline{\tilde{A}}}_N^{-1} \left(\left(\underline{\underline{\tilde{A}}}_N - (\nabla f_{\underline{\theta}_N})(\nabla^2 d_{f_N})(\nabla f_{\underline{\theta}_N})^T \right) \underline{\theta}_{N-1} + (\nabla f_{\underline{\theta}_N}) \underline{y}_N \right)$$

$$= \underline{\theta}_{N-1} + \underline{\underline{\tilde{A}}}_N^{-1} (\nabla f_{\underline{\theta}_N})(\nabla^2 d_{f_N}) \left(\underline{y}_N - (\nabla f_{\underline{\theta}_N})^T \underline{\theta}_{N-1} \right) \qquad 2:100$$

Note that for a linear system, $\nabla f_{\underline{\theta}_N}$ may be rewritten as a vector of system inputs, $\underline{\varphi}(N)$. Using Eqs. 2:93 - 2:95, $\underline{\underline{\tilde{A}}}_N^{-1}$ (= P_N) can be updated recursively.

$$P_N = \frac{1}{\lambda} \left(P_{N-1} - \frac{P_{N-1} \underline{\varphi}(N) \underline{\varphi}(N)^T P_{N-1}}{\lambda + \underline{\varphi}(N)^T P_{N-1} \underline{\varphi}(N)} \right) \qquad 2:101$$

Eqs. 2:100 and 2:201 are the RLS equations when a forgetting factor is used to control the gain sequence. Thus, for a linear system and a quadratic distortion function, the RILS equations (Eqs. 2:92-2:96) are equivalent to the RLS equations. In similar fashion, it can be shown that RILS is closely related to a number of other recursive Gauss-Newton algorithms including the Kalman filter, sequential regression, and stochastic–Newton optimization methods [44] [74]. RILS has the advantage, however, in that it is not restricted to a specific model structure and, therefore, is suitable for neural network function estimation where the system structure may not be known *a priori*.

2.4 Relationship to Other Neural Network and Statistical Modeling Paradigms

Many commonly used neural network and statistical function estimation techniques are subsumed by the methods presented here. A description of the most commonly used neural-network paradigms using the terminology presented in this section offers the following advantages:

(1) By understanding the relationships among popular neural network paradigms, the appropriate paradigm may be selected for the modeling task at hand.

(2) By understanding to what extent specific paradigms, including polynomial neural networks (PNNs), implement the general function estimation techniques presented here, one may readily see where improvements might be made to existing paradigms.

(3) By observing the close relationships among a variety of paradigms, one can make more efficient use of software and hardware development resources. For instance, it may be possible to implement a particular paradigm in special-purpose neural network hardware that has been designed to implement a different, but related, paradigm.

This section uses the terminology of generalized neural-based function estimation to describe some of the most common neural network and statistical modeling paradigms. Emphasis is given to paradigms that are designed to map data from a continuous-valued input space to a continuous-valued output space, although some "unsupervised" paradigms (i.e., techniques that find natural groupings in the input data space) will be mentioned.

2.4.1 Group Method of Data Handling (GMDH)

As already discussed, the Group Method of Data Handling (GMDH) was introduced in the late 1960's by A.G. Ivakhnenko, a Ukrainian cyberneticist. Ivakhnenko found that in the modeling of complex systems it is often very difficult, if not impossible, to develop a mathematical model and find all its parameters, and, even if the models and parameters could be obtained, very often, as the models begin to get sufficiently complex, they also begin to overfit the available data. GMDH solves this problem by "growing" a model from zero complexity to just-sufficient complexity [24].

Most early GMDH work employed the following quadratic multinomial in two inputs as the fundamental model building block:

$$z = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_i^2 + \theta_4 x_j^2 + \theta_5 x_i x_j \qquad 2:102$$

Eq. 2:102 provides in (m−1)/2 potential structural models (elements) for z, where m is the number of input variables. While this function is nonlinear in the inputs, it is linear in its parameters, and for each pair of input variables, x_i and x_j, the coefficients of Eq. 2:102 may be determined by linear regression. After finding all the candidate tw-input elements using the input/output data, those that are best able to estimate y are retained, the outputs of these elements become candidate inputs for subsequent layers of processing, and the regression continues. Note that as each layer is added, the degree of the resulting model increases by two. Any complete polynomial of any degree can be realized by suitable combinations of Eq. 2:70.

GMDH model construction continues until a level of optimal complexity is reached. To determine when to stop model construction, Ivakhnenko suggested using cross-validation, i.e., dividing the data into separate training (fitting) and evaluation data sets. Coefficients are determined by performing a linear regression on the training data; however, at each step, the resulting model is evaluated against the independent set of observations. When the model performance on the independent data ceases to improve, model evolution ceases. Many current practitioners of neural-based modeling continue to employ this method of determining when to terminate network training.

To implement GMDH using the principles described in this section, one begins with nodal elements that implement Eq. 2:102; such elements will have no time delays and no nonlinear post-transformation, h(·). The outputs of these elements will only feed forward (no feedback), and their basis function will be a polynomial (Eq. 2:6) with the following set of multi-indices

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} \qquad 2:103$$

GMDH, because it uses unconstrained linear regression, employs the squared-error distortion function of Eq. 2:23 without additional penalty terms; thus, at each stage in network construction, the coefficients may be found with a single ILS step (Eq. 2:54). While GMDH builds the network structure one element at a time, it does not use the projection pursuit strategy. As mentioned above, GMDH stops model construction when the network performance on independent data ceases to improve.

GMDH has been criticized because of the "enormous number of large matrix calculations [that] must be carried out" [39]. Although it is true that a large number of matrix calculations is required, it is also true that at any given step only six

parameters need to be determined. Thus, the "curse of dimensionality," which is usually the prime cause of long training times, is essentially avoided. The number of computations required to fit parameters optimally is $O(mn^2) + O(n^3)$, where m is the number of independent observations, and n is the number of coefficients [37]. Clearly, for any sizeable number of inputs and models, GMDH is more efficient computationally than other regression techniques.

Consider an example: Assume one wants to fit a fourth-order polynomial (multinomial) with ten inputs. This high-order model will contain over 1000 terms and require $O(10^9)$ iterations to arrive at a unique solution. A two-layer (fourth-order) GMDH network, on the other hand, requires only $O(10^5)$ computations if ten elements from the first layer are retained for use in subsequent layers or, at the most, $O(10^6)$ computations if all 45 elements from the first layer are retained. Even in the worst case, GMDH is over three orders of magnitude faster than brute-force high-order modeling. This numerical example is confirmed by the authors' experience, in which *GMDH algorithms typically have been found to be orders of magnitude faster than MLPs.*

An additional advantage of GMDH is that the performance surface for a single nodal element is always quadratic; thus, the coefficients on any nodal element can be globally optimized, in the least-squares sense, in a single iteration.

A disadvantage to GMDH is the fact that network construction is "heuristic" in tthat a definitive statistical theory of GMDH does not yet exist; however, there is general agreement that GMDH function estimation generally yields accurate and reasonably robust results [39]. In many other neural network paradigms (e.g., MLP), however, network structure is assigned arbitrarily by the analyst, or refined by the analyst using a trial and error approach. Another disadvantage sometimes cited is the fact that polynomials can lead to erratic fits *outside* the training region. One can reduce these effects, however, by incorporating a post-transformation, $h(\cdot)$, that limits the range of element outputs. A final disadvantage to the GMDH algorithm is the "corruption" of the independent test set by using it as part of the training procedure; A preferable method for determining when a network has reached a level of just-sufficient complexity is to add an information-theoretic complexity penalty term to the distortion function (Section 2.3.2).

2.4.2 Multi-Layer Perceptron (MLP)

The multi-layer perceptron (MLP) trained via the backward-error propagation (BP) technique is currently the most commonly used neural network paradigm. As such, there are many variants on the algorithm (fully or partially connected, Reduced Coloumb Energy (RCE) optimization, etc.); however, here, we shall deal only with the standard form of the algorithm.

The fundamental nodal element for the MLP was originally proposed by Rosenblatt in 1958 [o1]. The perceptron element implements the following nonlinear transformation:

$$z = h\left(\theta_0 + \sum_{i=1}^{i=D} \theta_i x_i \right) \qquad 2:104$$

where D is the total number of inputs to the nodal element, and h(·) is the nonlinear post transformation. Rosenblatt's original work used the following step nonlinearity for this post transformation:

$$h(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases} \qquad 2:105$$

With the emergence of gradient-based optimization techniques in the 1960s [73], however, the hard limiter, with a discontinuity at z=0, could not be used. Therefore, researchers substituted a continuously-differentiable approximation of Eq. 2:85. The most popular choice was the sigmoidal function.

$$h(z) = \frac{1}{1 + e^{-\gamma z}} \qquad 2:106$$

where γ, the sigmoid gain, determines the steepness of the transition region; as the magnitude of γ increases, Eq. 2:106 approaches the hard limiter of Eq. 2:105. Often, γ is set to unity.

The sigmoidal element of Eq. 2:104 can be implemented by a polynomial basis function (Eq. 2:6) composed with the sigmoidal nonlinearity, h(z), of Eq. 2:106. Since the polynomial basis function is linear, the $\underline{\underline{K}}$ matrix (J × D) is:

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & 1 \end{bmatrix} \qquad 2:107$$

If the network is fully connected, the number of inputs to the node, D, is the same as the number of outputs from the previous layer, and the polynomial expansion has D+1 terms. Note that whereas GMDH reduces its complexity by limiting the number of inputs to any given element, MLP reduces its complexity by limiting the order of the polynomial expansion to one (i.e., it is linear). Note that

the generalized network nodal element of Fig. 2.4 can handle either of these scenarios.

By far the most common method for training MLPs is the backward-error propagation (BP) algorithm, traceable to Robbins and Monroe [60]. The first complete description of BP was provided by Werbos [71]; however BP was not popularized as a useful procedure until 1983 [62]. BP is an iterative, gradient-based, least-mean squares (LMS) technique that tunes all the network weights simultaneously in an attempt to minimize the mean-squared error of the network output.

It can be shown that BP is the special case of the ILS optimization technique when the squared-error distortion function is used, all second-derivative information is ignored (Section 2.3.4.2), the network structure is fixed, and all coefficients are globally optimized from some randomly initialized starting point.

The first derivative of the squared-error distortion function (Eq. 2:23) with respect to the current network output, s, is given by

$$\nabla d_{s_0} = \frac{\partial d}{\partial s} = -2(y - s) = -2e \qquad 2:108$$

where y is the desired output and e is the error between the desired output and the network output, s. Substituting Eq. 2:76 into Eqs. 2:62 - 2:64, one obtains

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}(\Delta\underline{\theta}) \qquad 2:109$$

where

$$\underline{b} = -\frac{2}{N}\sum_{i=1}^{N} e_i \nabla_{f_{\underline{\theta}}} \qquad 2:110$$

and $\nabla_{f_{\underline{\theta}}}$ is the gradient of the network output with respect to the coefficient vector, $\underline{\theta}$. And

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu\underline{b} \qquad 2:111$$

where μ is the size of the step at each iteration.

As mentioned above, the MLP neural network assumes a fixed, pre-determined network structure; however, once this structure is fixed, BP optimizes the MLP by using techniques similar to those described for ILS global optimization of a neural network (Section 2.3.4.5). In fact, the global optimization algorithm described in 2.3.4.5 "backpropagates" gradient information through the network and could be described as a backward-error-propagation algorithm. Such terminology

was intentionally avoided in Section 2.3.4.5, because BP usually refers to the specific case of linear polynomial expansions, sigmoidal post-transformations, and gradient-based LMS optimization of the squared-error distortion function, whereas the ILS global optimization strategy is not restricted in any of these areas.

2.4.3 Radial Basis Function (RBF) Networks

After MLPs, radial basis function networks (RBFs) are one of the most popular and successful neural network paradigms [40]. The RBF network contains two layers (not counting the input layer). The hidden-layer elements implement a transformation that produces an output only when the input vector falls within a specific region of the data space. The term *basis function* in the paradigm name refers to this transformation. The output layer consists of a single element that constructs a weighted sum of the hidden-layer outputs.

The most commonly used hidden-layer transformation is the Gaussian kernel function of the form:

$$z = \exp\left(-\frac{(\underline{x} - \underline{w})^T(\underline{x} - \underline{w})}{2\sigma^2} \right) \qquad 2{:}112$$

where $\underline{w}$ and σ are the parameters of the node (we use the notation $\underline{w}$ and σ for now because alternative coefficients, $\underline{\theta}$, will be specified later). Note that the node outputs are in the range from zero to one, and the closer the input vector is to the center of the Gaussian function (as defined by $\underline{w}$) the larger the response of the node. The radial symmetry of Eq. 2:112 is what gives RBFs their name.

The RBF output layer is simply a linear combination of the outputs of the RBF nodal elements on the hidden layer.

$$y = \underline{\theta}^T \underline{z} \qquad 2{:}113$$

Eq. 2:113 may be implemented using a polynomial basis function and the same $\underline{\underline{K}}$ matrix that is used for MLPs (Eq. 2:107).

The most straightforward way to implement the hidden RBF elements (Eq. 2:112) using generalized nodal elements is to use a small network for each hidden-layer transformation. Such a network is shown in Fig. 2.12.

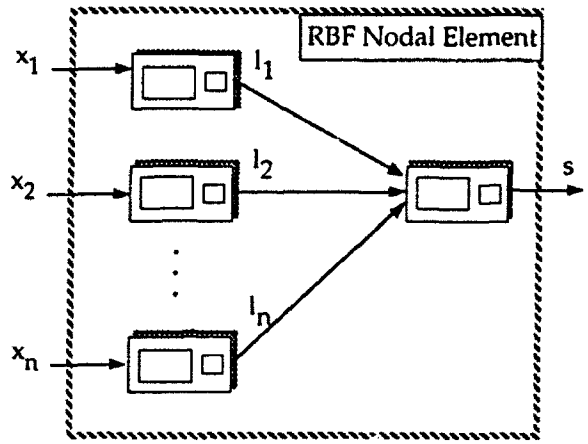


Figure 2.12: A Gaussian Kernel Implemented using Multiple Generalized Nodal Elements

In Fig. 2.12, the nodes on the first layer, $l_1 \ldots l_n$, implement a transformation of the form:

$$z = \theta_0 + \theta_1 x \qquad 2:114$$

which can be accomplished using a polynomial basis function (Eq. 2:6) and the following $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad 2:115$$

These first-layer of nodes have no nonlinear post-transformation, h(·). The second-layer node, s, in Fig. 2.12 then performs the following transformation on its input vector, $\underline{l}$:

$$s = \exp\left(- \frac{\theta_1 l_1^2 + \theta_2 l_2^2 + \ldots + \theta_n l_n^2}{2\sigma^2} \right) \qquad 2:116$$

Eq. 2:116 may be implemented using a nodal element that contains a polynomial power series expansion with a $\underline{\underline{K}}$ matrix

$$\underline{\underline{K}} = \begin{bmatrix} 2 & 0 & \ldots & 0 \\ 0 & 2 & \ldots & 0 \\ . & . & . & . \\ 0 & 0 & \ldots & 2 \end{bmatrix} \qquad 2:117$$

and a nonlinear post-transformation:

$$h(z) = \exp\left(-\frac{z}{2\sigma^2}\right) \qquad 2:118$$

where z is the output of the series expansion implemented by Eq. 2:117. Note that for these nodes to implement Eq. 2:112 exactly, the following restrictions should be placed on the nodal elements:

Layer 1

$$\theta_0 = -w$$
$$\theta_1 = 1.0 \qquad 2:119$$

Layer 2

$$\theta_i = 1.0 \quad \text{for all i}$$

Also note that the post transformation Eq. 2:118 contains a parameter, σ, and on-line supervised updating will require the derivative of h(z) with respect to σ. The parameter, σ, may be removed from the post-transformation by redefining the coefficients on the first layer:

Layer 1

$$\theta_0 = -\frac{w}{\sigma} \qquad 2:120$$
$$\theta_1 = \frac{1.0}{\sigma}$$

so that

$$h(z) = \exp\left(-\frac{z}{2}\right) \qquad 2:121$$

Now all the coefficients have been moved to the first layer of Fig. 2.12, and it is easier to interpret the tasks of the specific nodal elements. In short, the first layer computes distance measures between an input exemplar, $\underline{x}$, and some pre-determined vector, where each nodal element measures the distance along a single axis. The second layer of Fig. 2.12 (Eq. 2:121) converts the distances along each axis into a single Euclidean distance measure (L_2 norm) between the input vector and the pre-determined vector, $\underline{w}$. The nonlinearity of Eq. 2:121 then converts this distance measure into a probability of membership in a Gaussian cluster (i.e., high values when the vectors are close).

Dividing the RBF into two layers of generalized nodal elements helps give insight into the nature of the RBF. Often, regardless of the network paradigm, it is

useful to normalize all the input data; this can be done using an input layer consisting of nodes having the same structure as the input nodes of Fig. 2.12 using

$$\theta_0 = -\frac{\mu_i}{\sigma_i} \qquad 2:122$$

and

$$\theta_1 = \frac{1.0}{\sigma_i} \qquad 2:123$$

where i corresponds to a particular input variable, x_i ; and μ_i and σ_i are the mean and standard deviation of that input variable. Thus, a normalizing node outputs a measure of the distance between the input data point, x_i, and its mean measured in units of standard deviation. Note that these coefficients are determined prior to network training and are based solely on the statistical nature of the training database.

In an RBF network, the parameters on the input layer serve the same purpose; only, instead of measuring the distance between an input data point, x_i, and the mean of the entire input data set, they measure the distance between the input data point and the mean of some cluster in the data space. This distance is measured in units of standard deviation of the cluster. Note that to perform normalization, n input nodes are needed for the n input features, but to compute cluster distances for m clusters, n × m input nodes are required because each cluster has a different set of statistics.

As with normalizing input nodes, the parameters of the RBF input nodes are determined prior to network training. Typically, this is accomplished using an unsupervised clustering algorithm, such as *K*-means, to determine the statistics of the naturally occurring clusters in the data.

This interpretation of the RBF hidden layers suggests some potential improvements. As noted above, RBF networks are radially symmetric; thus, for a given Gaussian kernel, the value of σ is fixed for all axes. However, if the Gaussian kernel is intended to describe naturally occurring clusters in the data space, it is conceivable, and indeed probable, that these clusters will have different standard deviations along each feature axis. In this case, a more accurate distance measure will be one that measures the distances along each axis in units of the standard deviation of the cluster along that axis.

Fig. 2.13 illustrates the distinction between radial and elliptical distance measures. If the shaded region of the figure represents a naturally occurring cluster, then the circle in Fig. 2.13 with radius σ_0 represents the one-sigma boundary for a radial cluster in the data space. It is obvious from the figure that the point, A, lies

within this boundary. If however, σ_1 and σ_2 are used to define the one-sigma boundary for an elliptical cluster, then the point, A, lies well outside the cluster. The latter representation is more accurate.

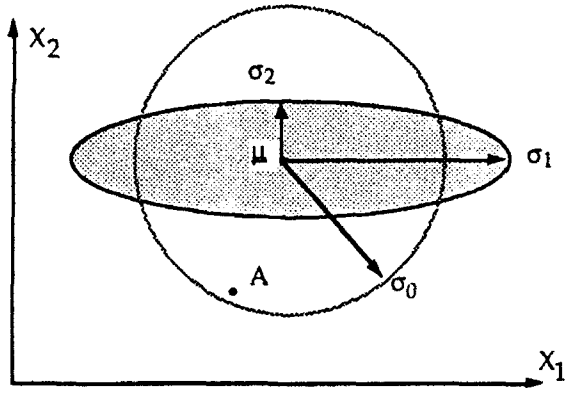


Figure 2.13: Measuring Cluster Distances

Generalized nodal elements can easily implement elliptical data clusters if the constraints in Eq. 2:120 are relaxed, and each input node of Fig. 2.12 is allowed to use a different value of σ. The resulting network is an elliptical basis function (EBF) network and it implements the following transformation:

$$z = \exp\left(-(\underline{x} - \underline{w})^T \underline{\underline{\rho}}^{-1} (\underline{x} - \underline{w}) \right) \qquad 2:124$$

where

$$\underline{\underline{\rho}} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix} \qquad 2:125$$

The distance measures as computed by the input nodes are known as standardized Euclidean distances or "Karl Pearson distances." Where desirable, other distance measures may be used, such as the *Mahalanobis* distance, which results in an EBF kernel of the form

$$z = \exp\left(-(\underline{x} - \underline{w})^T \underline{\underline{\Sigma}}^{-1} (\underline{x} - \underline{w}) \right) \qquad 2:126$$

where $\underline{\underline{\Sigma}}^{-1}$ is the normalization matrix for the kernel, and $\underline{\underline{\Sigma}}$ is the covariance matrix for the input data. Note that the expression in Eq. 2:126 contains the generalized Fisher linear discriminant function

$$(\underline{x}-\underline{w})^T\underline{\underline{\Sigma}}^{-1}(\underline{x}-\underline{w}) \qquad 2:127$$

which itself constitutes a classical measure used in linear classification.

We have already mentioned that the coefficients on the input layer correspond to the statistics of naturally occurring data clusters and are found off-line via data analysis. Once the RBF kernels have been set, the tuning of the output layer consists of finding the best linear mapping between the kernel outputs and the desired network output(s). Because the output layer is linear, a single ILS search step will globally optimize the output layer coefficients. However, despite its inferiority, LMS is frequently used.

Once the coefficients on the output layer have been determined, the RBF network may be further enhanced by globally optimizing all layers of the network using ILS. Thus, when global optimization is used, the hidden-layer parameters are allowed to vary from their initialized values to form new "data clusters" that serve even better as basis functions for the classification task at hand. This global optimization method is often referred to as adaptive kernel classification (AKC) [34].

2.4.4 Pi-Sigma and Other Higher-Order Networks

Higher-order networks are networks that utilize polynomial series expansions of higher order than the linear expansions used by MLPs. In this regard, GMDH is often considered a higher-order network, because each nodal element implements the second-degree polynomial expansion of Eq. 2:83. However, as mentioned in Section 2.4.1, GMDH compensates for the higher-order series expansion by limiting the number of inputs to any given nodal element and by limiting the total number of nodal elements.

Pi-Sigma networks (PSNs) are another higher-order network paradigm in current use [64]. PSNs get their name from the fact that the network output is a *product of sums* of the input variables. Typically, a PSN contains two layers (not counting the input layer). Each element in the single hidden layer implements the following transformation:

$$z = \theta_0 + \sum_{i=1}^{D} \theta_i x_i \qquad 2:128$$

where D is the number of inputs to the network. Generalized nodal elements can implement Eq. 2:118 with a linear polynomial expansion (Eq. 2:108) and no post-transformation, $h(\cdot)$.

The output layer of a PSN computes

$$s = h\left(\prod_{i=1}^{l} z_i\right) \qquad 2:129$$

where l is the number of nodal elements on layer one, and h(·) is the sigmoidal transformation given by Eq. 2:106. Once again, this element may be implemented by using a polynomial expansion; however, in this case, the polynomial consists of a single cross term and can be implemented by the following $1 \times l$ $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = [1\ 1\ 1 \ldots\ldots\ 1] \qquad 2:130$$

A hybrid GMDH/LMS approach is usually used to train PSNs. The network structure is fixed with a small number (one or two) hidden-layer elements and tuned using the LMS global optimization strategy described in 2.4.2 (backward-error propagation when there are no hidden layers). However, once the coefficients have converged, an additional element is added to the hidden layer and the coefficients of each element are re-tuned in an asynchronous fashion (i.e., only the parameters of a single element are optimized at a given time; this tends to yield more favorable results than a global optimization). At each step, performance is tested on independent data (as with GMDH) and network growth is stopped when overfitting begins to occurs. The *order* of the PSN is equal to the number of elements on the hidden layer.

2.5 Summary

This section has provided a way of viewing generalized function estimation in a neural network context. The intent is to provide a paradigm that is sufficiently general to cover many estimation techniques currently in use, including GMDH, MLPs, RBFs, static and dynamic polynomial neural networks, and many of the estimation techniques popular within the statistics community. What follows is a discussion of specific polynomial neural network (PNN) algorithms that the authors have implemented, with an explanation of how they fit into the overall function estimation paradigm described above. The hope is that the general and comprehensive paradigm in this section will help the reader understand the PNN algorithms, the relationships between various techniques, and the nature of suggested improvements to the network generation algorithms.

3. POLYNOMIAL NEURAL NETWORK (PNN) SYNTHESIS ALGORITHMS

3.1 Introduction

In the previous section, a way of approaching neural (and non-neural) function estimation was presented and a variety of common neural modeling techniques were recast into the terminology of generalized function estimation. This highlighted the similarities and differences among various artificial neural network (ANN) paradigms. In this section, we present three neural network algorithms that have been developed by Barron Associates, Inc., and, as in the previous section, describe these algorithms in terms of the approach to generalized function estimation from the previous chapter. The three neural network algorithms discussed in this section are:

(1) *Algorithm for Synthesis of Polynomial Neural Networks for Classification (CLASS)*

(2) *Algorithm for Synthesis of Polynomial Neural Networks for Estimation (ASPN)*

(3) *Algorithm for Synthesis of Dynamic Polynomial Neural Networks for Estimation (DynNet)*

These algorithms primarily make use of polynomial basis functions (Eq. 2:6), with no nonlinear post-transformations, h(z) (Fig. 2.4).

Table 3.1 outlines the differences between the above PNN algorithms. To understand the differences, it is important to understand the contexts in which these algorithms are likely to be used and the terminology associated with those contexts.

Models of complex systems and processes can be created from physical observations or from simulations. *Direct* models are used to predict future outcomes or infer present situations that are dependent on observable antecedent conditions, whereas *inverse* models infer the antecedent conditions that have brought about observed outcomes or infer the present actions that will bring about desired future events.

Estimation models can be direct or inverse estimators and are used to calculate future, present, or past values of the parameters of a system or process. The forms of these parameters are limitless, and may include the values of observable or unobservable state variables, feedback gains, control settings, vehicle or plant parameters, levels of impairment, Lagrange multipliers, etc. *Classification* models, which can be direct or inverse classifiers, also perform estimations, but

specifically of the class or classes of future, present, or past input data vectors or the probabilities of these classes.

Batch syntheses of models proceed from recorded databases of real or simulated observations and the corresponding true (i.e., desired) model outputs. (Note that "truth" is sometimes an uncertain observable.) Batch syntheses may be performed either off- or on-line; when used on-line, batch syntheses require periodic updating of the database. Batch syntheses are often referred to as "supervised" training. *Recursive* syntheses usually do not employ explicit databases. Instead, the memories of recursive systems generally reside in their structure and coefficients. Recursive syntheses are most often used for on-line learning.

Table 3.1: Differences between Several Polynomial Neural Network Synthesis Algorithms

Attribute	ALGORITHM		
	CLASS (Static Classification)	*ASPN* (Static Estimation)	*DynNet* (Dynamic Estimation)
Purpose	Estimate probabilities of past, present, or future *classes* of input observation vectors	Estimate past, present, or future values of variables dependent on input observation vectors (FIR filtering)	Estimate past, present, or future values of variables dependent on input observation vectors (IIR filtering)
Structure	Feedforward network of polynomial nodal elements in first layer, with logistic transformation in second layer	Feedforward network with an analyst-specified number of layers of polynomial and transcendental nodal elements	Network with internal memory and/or feedback and an analyst-specified number of layers of polynomial nodal elements
Criterion of Optimality	Constrained Minimum Logistic Loss	Predicted Squared Error or Minimum Description Length	Predicted Squared Error
Method of Synthesis	Levenberg-Marquardt fitting algorithm used with pre-structured nodes in first layer of network	Least-squares fitting algorithm embedded in combinatorial search among alternative network structures	Least-squares initialization and Levenberg-Marquardt optimization of elements with feedback

Static models contain no internal feedback paths and no internal time delays or other forms of internal memory. Static models are often ambiguously described as using "feedforward" structure. Note, however, that "feedforward" models may incorporate internal memory, in which case they may become *dynamic* with or without the use of internal feedback. Models containing time delays and/or internal feedback are properly called "dynamic." Dynamic networks that contain internal feedback are also properly called *recurrent* or *reverberatory* models, and are capable of producing oscillatory and aperiodically varying time responses even when their inputs are constants. In filter terminology, static networks provide finite impulse response (FIR) transformations, while recurrent networks provide infinite impulse response (IIR) filtering.

3.2 Algorithm for Synthesis of Polynomial Classification Neural Networks *(CLASS)*

The *CLASS* algorithm automates the batch synthesis of static polynomial neural networks suitable for use as direct or inverse classification models. Although it is possible to use estimation software to perform classifications (often by discretizing or thresholding the output values and sometimes with majority-rule voting), improved performance is obtained using algorithms specifically derived for the task of classification. *CLASS* offers the following advantages:

- Minimization of the logistic loss function (Eq. 2:20), resulting in optimal (maximum likelihood) classification of data having a multinomial probability distribution.
- A regularized nonlinear Gauss-Newton optimization algorithm for rapid on- and off-line network training.
- Ability to provide nonlinear classification having a degree of complexity (i.e., classification power) commensurate with the quantity and quality of the training data base.
- Outputs that are estimates of the *a posteriori* probabilities of class membership. These are particularly useful when these outputs are used by higher-level decision-making processes.
- Simple network structures that do not overfit the training database and that can be interrogated rapidly on-line.

3.2.1 Network Structure

CLASS makes use of a pre-structured, fully-connected, feedforward network structure, as shown in Fig. 3.1. This network has an input layer, a hidden layer, and an output layer that together compute the estimates of the conditional probabilities of class membership. The last layer is required for interrogation of the network.

During network training, however, the transformation implemented by the output layer is incorporated in the distortion function, and the hidden layer elements are treated as network outputs.

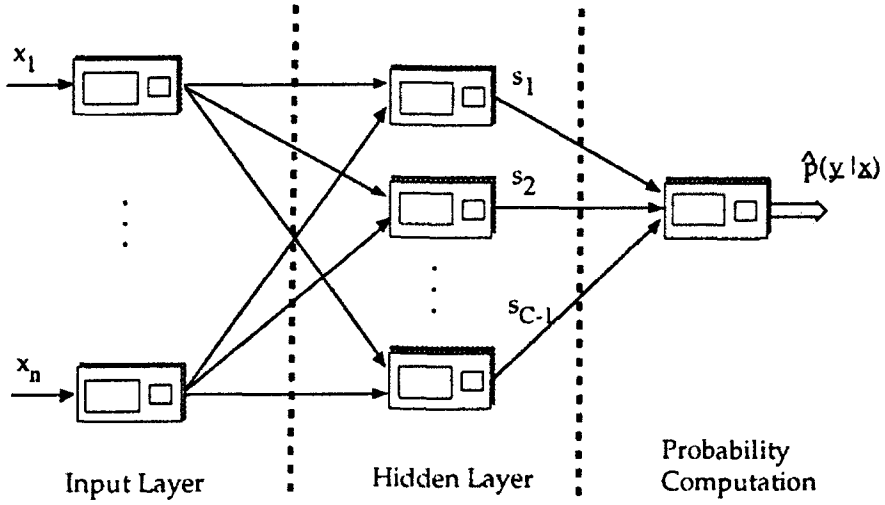


Figure 3.1: *CLASS* **Network Structure**

The input layer of a *CLASS* network may contain coefficients that unit-normalize each input vector. These coefficients are optional, and are determined prior to network training based on the statistics of the database. If normalization is desired, the input layer nodes use a polynomial basis function with no nonlinear post-transformation:

$$z = \theta_0 + \theta_1 x \qquad 3:1$$

which can be accomplished using a polynomial basis function (Eq. 2:6) and the following $\underline{\underline{K}}$ matrix:

$$\underline{\underline{K}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad 3:2$$

The coefficients have a direct relationship to the mean, μ, and standard deviation, σ, of each input variable

$$(\theta_0, \theta_1)^T = \frac{1}{\sigma}(-\mu, 1.0)^T \qquad 3:3$$

Note that the results obtained with input normalization will, in general, differ from those obtained without normalization.

In Section 2.3.1.2, we showed that for multi-class classification problems having categorical output variables, a multinomial probability model in regular exponential form results in maximum-likelihood estimation. In this context, the probability that an observation is a member of class k is given by

$$p(k \mid \underline{x}) = \frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}} \qquad 3:4$$

and the distortion function is minus the log of the likelihood

$$d(k, \underline{s}) = -s_k + \ln\left(\sum_{j=1}^{C} e^{s_j}\right) \qquad 3:5$$

Because the outputs of the probability computation sum to unity, only C–1 hidden nodal elements are required to solve a C-class problem (since the Cth probability is one minus the sum of the C–1 probabilities). Each of the C–1 hidden nodes (sub-networks) returns essentially unbounded values (∞ to $-\infty$) which, after the static logistic transformation, indicate the probabilities of membership in their respective classes. In *CLASS*, the output of the C^{th} node is defined to be zero. (This is not required by the logistic model, but is done for convenience. With C (rather than C–1) sub-nodes for a C-class problem, the class probabilities will still sum to one.) Because the probability computation (Eq. 3:4) does not contain any coefficients to be optimized, and because the distortion function (Eq. 3:5) is a function of the outputs of the hidden layer, $\underline{s}$, one may treat the C–1 hidden-layer outputs, $\underline{s}$, as network outputs during the training process.

Each of the C–1 hidden elements in a *CLASS* network receives all of the inputs (i.e., the network is fully connected). The series expansion, or core transformation, chosen for the hidden nodal elements is the Kolmogorov-Gabor (KG) multinomial [5], which is an algebraic sum of terms:

$$z = \theta_0 + \sum_i \theta_i x_i + \sum_{ij} \theta_{ij} x_i x_j + \sum_{ijk} \theta_{ijk} x_i x_j x_k + \ldots \qquad 3:6$$

The KG multinomial can model any analytic single-valued transformation [42]; therefore it is a good choice as a basis function, but, in principal, many kinds of building-block elements could be used in modeling by induction (see Section 2.2.2). The attention to algebraic elements derives from the pioneering work in the 1940s of Kolmogorov and, working independently, Gabor [31] [32]. They demonstrated the near universality of multinomials in representing physical processes, including dynamic systems. In fact, recent developments in statistics, information theory, computational methods, and approximation theory suggest that a multinomial description of the network learning process highlights some of the similarities (as

well as important differences) among network syntheses and modern statistical inference methods [7]. Eq. 3:6 explicitly shows the first four KG multinomial summations, corresponding to all degree 0, 1, 2, and 3 terms.

As was shown in Section 2.2.2.2, the number of terms (coefficients), J, for any series expansion, including the complete KG multinomial of Eq. 3:6, is

$$J = \frac{(R + D)!}{R!D!} \qquad 3:7$$

where R is the *degree* of the multinomial (i.e., the maximum sum of the exponents for any given term). D is the number of inputs to the core transformation. As can be seen from Eqs. 3:6 and 3:7, the number of terms in the complete KG multinomial can become very large for even small values of D and R. *CLASS* allows the analyst to limit the number of terms in the series expansion in one of three ways:

(1) Limit the maximum degree, R, of the series expansion. The resulting polynomial expression is called a *complete* polynomial of degree R. For a polynomial basis function, limiting the degree is equivalent to eliminating any terms for which the sum of the powers on the inputs exceeds R. Thus, the number of summation terms in Eq. 3:6 is limited, and the sum of the rows of the J × D matrix of indices, $\underline{\underline{K}}$, cannot exceed R. A two-input (D = 2), third-degree (R = 3) complete polynomial is

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_2 + \theta_5 x_2 x_1 + \theta_6 x_2 x_1^2 + \theta_7 x_2^2 + \theta_8 x_2^2 x_1 + \theta_9 x_2^3 \qquad 3:8$$

(2) Limit the maximum coordinate degree, P, of the expansion to one, and limit the maximum degree, R, of the expansion. The resulting polynomial expression is called a *multilinear* multinomial of degree R. For a polynomial basis function, limiting the coordinate degree is equivalent to eliminating any terms in the expansion that contain inputs raised to a power greater than P. Thus, the size of any integer in the $\underline{\underline{K}}$ matrix can never exceed P. Experience has shown that in many applications just the cross terms, $x_i x_j$, represent very effective nonlinearities. Eliminating all terms where the maximum coordinate degree is greater than one reduces the complexity of the element, yet retains its power to introduce nonlinearities when such are needed to produce a desired output. A three-input (D = 3), second-degree (R = 2), multilinear (P = 1) element is

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1 x_2 + \theta_5 x_1 x_3 + \theta_6 x_2 x_3 \qquad 3:9$$

Notice that the degree of a *multilinear* multinomial corresponds to the number of inputs allowed to appear together in any given cross term.

(3) Limit the interaction order, Q, to one, and limit the maximum degree, R, of the expansion. The resulting polynomial expression is called an *additive* multinomial of degree R. For a polynomial basis function, limiting the coordinate degree is equivalent to eliminating any terms in the expansion that contain more than R inputs. Thus, the rows of the $\underline{\underline{K}}$ matrix may not contain more than R non-zero terms. A three-input (D = 3) second-degree (R = 2), additive (Q = 1) expansion is

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1^2 + \theta_5 x_2^2 + \theta_6 x_3^2 \qquad 3:10$$

Thus, although P, Q, and R could, in general, be specified independently (see Section 2.2.2.2), *CLASS restricts* the user to specifying R and optionally selecting a single additional restriction: P = 1 (multilinear) or Q = 1 (additive). Without P and Q restrictions, a complete polynomial is selected. Note that regardless of any additional restrictions, R = 1 will always result in a linear polynomial.

In addition to the above restrictions, *CLASS* allows the analyst to specify other multinomials by creating a custom $\underline{\underline{K}}$ matrix and saving it in a named node file. Because the network is fully connected, and all input data components ($\underline{x}$) are submitted to each node, it is important for the analyst to keep the degree of the specified polynomial reasonably low to prevent the number of coefficients from increasing exponentially.

Once the specific terms in the KG-multinomial are specified by one of the above methods, *CLASS* creates C–1 structurally identical hidden-layer nodal elements; this uniformity is a matter of convenience. Additionally, to use C–1 nodal elements to compute C probabilities, one defines an additional hidden-layer output, s_C, that is always zero by definition. Once again, *this restriction is not necessary*, but it limits the complexity of the network and leads to the following interpretation of the hidden-layer outputs: the output sub-network, k, is the natural log of the ratio of the probability of membership in class k to the probability of membership in some baseline class, C (with *CLASS*, the baseline class should be the "all-other" class; in the logistic model, however, class C can be any of the classes):

$$s_k = \ln\left[\frac{p(k|\underline{x})}{p(C|\underline{x})}\right] \qquad 3:11$$

To arrive at Eq. 3:11, define $F(\underline{s})$ as follows:

$$F(\underline{s}) = \sum_{j=1}^{C} e^{s_j} \qquad 3:12$$

Therefore, Eq. 3:4, may be rewritten for class C as

$$p(C \mid \underline{x}) = \frac{1}{F(\underline{s})} \qquad 3:13$$

Additionally, Eq. 3:4 may be solved for s_k as follows:

$$s_k = \ln[\, F(\underline{s})] + \ln[\, p(k \mid \underline{x})] = \ln[\, F(\underline{s})\, p(k \mid \underline{x})] \qquad 3:14$$

Substituting Eq. 3:13 into Eq. 3:14, one obtains the result in Eq. 3:11.

The C–1 sub-networks are trained *simultaneously* as the numerical search seeks to minimize the loss function of Eq. 3:5. Each sub-network computes the log-odds of a particular class vs. the baseline class. (The choice of the baseline class is arbitrary.) The sub-network results are processed by an output layer as given by Eq. 3:4 to obtain the class probabilities.

Minimization of Eq. 3:5 produces the maximum log-likelihood that the classification probabilities are correct. Suppose the polynomials in the C–1 branches of the minimum-logistic-loss classifier are linear functions of the form

$$s_k = \theta_{0.k} + \sum_{j=1}^{N} \theta_{j.k}\, x_j \qquad 3:15$$

If we first examine a two-input, two-class minimum-logistic-loss network using a *linear* node, we see that the network performs a linear separation between the two classes. This linear node defines the discrimination boundary between classes (Fig. 3.2). The example can be extended to more than two inputs and more than two classes. When more than two classes are involved, use of linear nodes with the minimum-logistic-loss criterion creates an optimal family of discrimination lines (or hyperplanes) dictated by the distributions of the synthesis data populations for the various classes. This family is found using a simultaneous search as interactions arise between classes when locating multiple discriminant functions.

Considering, next, a two-input second-degree additive polynomial as the nodal polynomial form, and assuming only two classes are to be discriminated, the minimum-logistic-loss classifier finds the best (at least locally) quadratic separator. In general, the coefficients of the quadratic polynomial will be different from those found using traditional clustering techniques unless the statistics of the features are Gaussian. (Indeed, closed-decision boundaries are not guaranteed with *CLASS*, but the decision surface geometry an be determined by examination of the relationships of the nodal parameters within the sub-nodes.) The advantage, therefore, of using the minimum-logistic-loss network is that the coefficients are more general and reflect the distributions found in the data. The network can only do better (at least on the training data) than traditional clustering techniques. In fact, given suitable values of the coefficients, a second-degree multilinear polynomial

can describe a point, a line, a circle, an ellipse, a hyperbola, or a pair of intersecting lines.

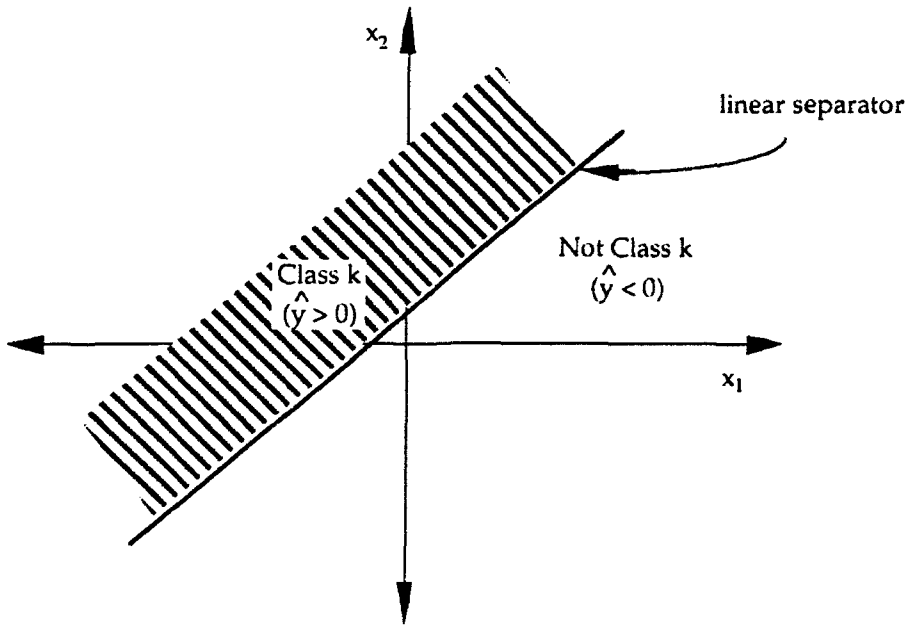


Figure 3.2 Minimum-Logistic-Loss Classifier with Linear Nodes

Further, one may implement even more general forms of data clustering with minimum-logistic-loss networks by using a two-input second-degree complete polynomial. Note that the only difference mathematically is the inclusion of a cross-product term in the complete polynomial. Given this form, not only can any conic section form of separation be performed, but the separation can be rotated with respect to the coordinate system defined by the features. The advantage here is that the minimum-logistic-loss network will automatically perform a linear rotation of the coordinates if it provides better separation than the principle axes of the features. With traditional clustering techniques, this would have to be performed in a separate step before clustering, e.g., by using the Karhunen-Loève transform or singular value decomposition. (See Section 4.5.1.6 for further details.)

The constrained minimum-logistic-loss criterion, which is explicitly designed for classification problems, provides performance superior to classifiers fitted using estimation criteria. Estimation networks place emphasis on estimation accuracy; minimum-logistic-loss networks instead place emphasis on maximizing the likelihood of correct class discrimination. Whereas true probabilities are always between 0 and 1, estimation networks are unbounded; in contrast, the logistic-loss criterion correctly maps the network outputs onto {0, 1}. A significant advantage of the minimum-logistic-loss classifier is that the nodes are fitted simultaneously, giving the system a complete view of the problem at hand, with the coefficients in all nodes fitted simultaneously to the entire synthesis data set, instead of using

separate fitting of partitioned data sets. This property also forces the trained nodes to be consistent with each other.

Often the output of a *CLASS* network is post-processed by one or more decision rules. If all of the classes of interest are included in one classifier, the final decision rules can be simple. An obvious decision rule is to select the class having the highest probability. However, one may impose further requirements. For example, one may require that for a class to be selected, its probability must exceed a threshold.

If it is known that class occurrences are independent events (or nearly so) and if the quantity of synthesis data is very limited, it may be appropriate to use *CLASS* to create C (C – 1)/2 *pairwise* (one vs. all) sub-classifiers. All of the available data may be used in synthesizing each sub-classifier, and the probability for class k may then be calculated by

$$\begin{aligned} p(k \mid \underline{x}) = \; & p(k \text{ vs. class } 1 \mid \underline{x}) * \\ & p(k \text{ vs. class } 2 \mid \underline{x}) * \\ & \ldots * \\ & p(k \text{ vs. class } C\text{–}1 \mid \underline{x}) * \\ & p(k \text{ vs. class } C \mid \underline{x}) \end{aligned} \qquad 3:16$$

Where p(k vs. k | $\underline{x}$) is unity.

3.2.2 Levenberg-Marquardt Optimization

The underlying statistical criterion for classifier synthesis using the *CLASS* algorithm is of the form given in Eq. 2:35

$$J = \frac{1}{N} \sum_{i=1}^{N} d(k_i, \underline{s}_i) + \kappa \frac{\kappa}{N} \qquad 3:17$$

where:

N = number of synthesis-data input vectors (exemplars)

K = number of non-zero degrees of freedom (weights) in neural network

$d(k_i, \underline{s}_i)$ = loss (distortion) function for the i^{th} input vector

k_i = Correct classification number for the i^{th} input vector. Recall from Section 2.3.1.2 that the desired output may also be represented as a desired output vector, $\underline{y}_i$, where each component of this vector is a member of the set {0, 1}; if class k is the actual class, the kth component of $\underline{y}_i$ is unity and the other components are zero

$\underline{s}_i$ = binary vector of actual classifications by the neural network for the i^{th} input vector; each component of this vector is a member of the set {0, 1}; if class k is the estimated class, the kth component of $\underline{s}_i$ is unity and the other components are zero

κ = A constant that can be taken to be unity for the present discussion

Node output k within the network is denoted

$$s_{i,k}(\underline{x}_i ; \underline{\theta}_k) \; ; \quad i = 1, \ldots, n; \quad k = 1, \ldots, C-1$$

in which the index i represents the ID number of a given one of the input-data vectors, the index k represents the ID number of a given one of the network nodes (each node corresponding to an output of the classifier prior to the logistic transformation), $\underline{x}_i$ is the i^{th} input vector, $\underline{\theta}_k$ is a vector of parameter values internal to node k, and C is the number of different classes that the network is trained to recognize. From this point on, the subscript, i, will be dropped from all equations.

The distortion function for logistic-loss classifiers is minus the log of the likelihood of class membership (Eqs. 2:25 and 3:5) and is reproduced here for convenience

$$d(k, \underline{s}) = -\ln[(p(k|\underline{x})] \quad = -s_k - \ln[\, F(\underline{s})\,] \qquad 3:18$$

where $F(\underline{s})$ is defined by

$$F(\underline{s}) = \sum_{j=1}^{C} e^{s_j} \qquad 3:19$$

The logistic-loss criterion is to be minimized over the synthesis database during classifier design by adjusting K degrees of freedom in a network having the C–1 outputs $s_1, \ldots, s_{C-1}$, as shown in Fig. 3.1.

Because minimization of logistic loss involves a nonlinear transformation between the classifier weights and the estimated probabilities for the different modes of impairment, a linear regression method of adjusting the weights is inappropriate. Instead, the Levenberg-Marquardt (LM) algorithm is be employed. LM is a nonlinear regression technique that exploits the derivative information that is known analytically. LM combines Gauss-Newton and gradient descent methods and is subsumed by the ILS network training technique (Section 2.3.4.4).

Recall that the ILS global optimization requires the analyst to provide the following information:

- an analytic form of the first and second partials of the objective function with respect to the network outputs, $\nabla d_{\underline{s}}$ and $\nabla^2 d_{\underline{s}}$
- an analytic form for the first derivative of the post-transformation h(z)
- an analytic form for the gradient of an element output with respect to its input vector, $\nabla f_{\underline{x}}$

Because *CLASS* has only one layer to optimize, and the elements on that layer do not make use of the post-transformation h(z), only the partials of the objective function need to be specified.

The partial derivative of Eq. 3:18 with respect to the m^{th} network outputs, s_m, may be calculated as follows:

$$\frac{\partial d}{\partial s_m} = \frac{1}{F(\underline{s})} \frac{\partial F}{\partial F s_m} - \begin{cases} 1 & \text{if } m = k \\ 0 & \text{if } m \neq k \end{cases} \qquad 3:21$$

Eq. 3:21 may be simplified by using the m^{th} element of the desired output vector, y_m. Thus

$$\frac{\partial d}{\partial s_m} = \frac{e^{s_l}}{F(\underline{s})} - y_m \qquad 3:22$$

The second partial derivative of Eq. 3:18 with respect to the m^{th} and n^{th} network outputs is given by

$$\frac{\partial^2 d}{\partial s_m \partial s_n} = \begin{cases} -\dfrac{e^{s_m} e^{s_n}}{F^2(\underline{s})} & \text{for } m \neq n \\ \dfrac{e^{s_m}}{F(\underline{s})} - \dfrac{e^{2s_m}}{F^2(\underline{s})} & \text{for } m = n \end{cases} \qquad 3:23$$

The only other information required to find the gradient of the objective function with respect to the network coefficients is the vector of partial derivatives of the output of the series expansion with respect to the coefficients. For a series expansion of the form

$$z = \sum_{j=0}^{J} \theta_j \Phi(\underline{k}_j, \underline{x}) \qquad 3:24$$

with no post-transformation, h(·), the vector of partials with respect to the coefficients is

$$\nabla s_{\underline{\theta}} = \begin{bmatrix} \frac{\partial s}{\partial \theta_1} \\ \frac{\partial s}{\partial \theta_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial s}{\partial \theta_J} \end{bmatrix} = \begin{bmatrix} \Phi(\underline{k}_1, \underline{x}) \\ \Phi(\underline{k}_2, \underline{x}) \\ \cdot \\ \cdot \\ \cdot \\ \Phi(\underline{k}_J, \underline{x}) \end{bmatrix} \qquad 3:25$$

where

$$\Phi(\underline{k}_j, \underline{x}) = \Phi(k_{j1}, x_1) \bullet \Phi(k_{j2}, x_2) \bullet \ldots \bullet \Phi(k_{jD}, x_D) \qquad 3:26$$

and k_{jk} is an element in the J × D matrix of indices, $\underline{\underline{K}}$. For the polynomial basis functions used in a *CLASS* network

$$\Phi(k_{jk}, x) = x^{k_{jk}} \qquad 3:27$$

Note that for C outputs, the gradient of the vector output $\underline{s}$ with respect to the coefficient vector $\underline{\theta}$, is a matrix

$$\underline{\nabla s}_{\underline{\theta}} = [\nabla s_{1\underline{\theta}}, \nabla s_{2\underline{\theta}}, \ldots, \nabla s_{C-1\underline{\theta}}] \qquad 3:28$$

Because each element contains only a small subset of the overall network coefficients, the gradient matrix of Eq. 3:28 is block-diagonal, and block optimizations may be used to compute the matrix efficiently.

The $\underline{\underline{A}}$ and $\underline{b}$ required for ILS optimization may be calculated as was described in Section 2.3.4.1:

$$\underline{\underline{A}} = \frac{1}{N} \sum_{i=1}^{N} (\nabla_{f_{\underline{\theta}}})(\nabla^2 d_{s_0})(\nabla_{f_{\underline{\theta}}})^T \qquad 3:29$$

and

$$\underline{b} = \frac{1}{N} \sum_{i=1}^{N} (\nabla_{f_{\underline{\theta}}})(\nabla d_{s_0}) \qquad 3:30$$

where $\underline{s} = \underline{f}(\underline{x}, \underline{\theta})$ and so $\nabla_{f_{\underline{\theta}}} = \underline{\nabla}\, \underline{s}_{\underline{\theta}}$. Normally, the ILS update is given by

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu \underline{\underline{A}}^{-1} \underline{b} \qquad 3:31$$

However, for nonquadratic objective functions, it is desirable to incorporate regularization as described in Section 2.3.4.4. With regularization, Eq. 3:31 becomes

$$\underline{\theta}_{new} = \underline{\theta}_{old} - \mu \left[\underline{\underline{A}} + \lambda \, \mathrm{diag}(\underline{\underline{A}})\right]^{-1} \underline{b} \qquad 3:32$$

If $\mu = 1$, Eq. 3:32 becomes the Levenberg-Marquardt (LM) optimization algorithm, and this is the technique used in *CLASS* to optimize the polynomial elements. The heuristics for adjusting λ and performing LM optimization are shown in Figure 3.3.

3.2.3 Complexity Penalty and Building Terms

CLASS can fit the entire network simultaneously, can build the parameters one at a time, or can build the parameters in groups. *Fitting all of the parameters* simultaneously is just that: the search is applied to all of the parameters. However, to build parameters, a criterion that trades off complexity and fitting error must be employed. As shown previously, for the logistic-loss function, a complexity penalty may be added as follows:

$$\frac{1}{N} \sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) + \kappa \frac{K}{N} \qquad 3:33$$

where d_i is the multiclass logistic-loss function evaluated at observation i, κ is a constant complexity multiplier, K is the number of non-zero parameters, and N is the number of observations in the database. Theoretically, the correct value for κ is one [7]; however, *CLASS* allows the analyst to increase or decrease the complexity penalty term *a priori*.

The procedure for building parameters is a forward stepwise greedy algorithm. At each step, given a set of parameters to add to the model, the one selected is that which produces the smallest J when the model is refitted with that parameter enabled. This process is repeated until no improvement in J is obtained, or until all of the parameters have been added.

Building the network parameters in groups can save computing time. There are Q^* groups in the network (Q^* is the number of parameters per node). Because the structure of each node is identical, corresponding parameters in each node may be grouped together, i.e., enabled and disabled simultaneously. For binary classification problems, grouping does not exist (since there is only one node). Problems with more than two classes, however, can benefit greatly from parameter grouping, since the time required to add one parameter at a time could be prohibitive.

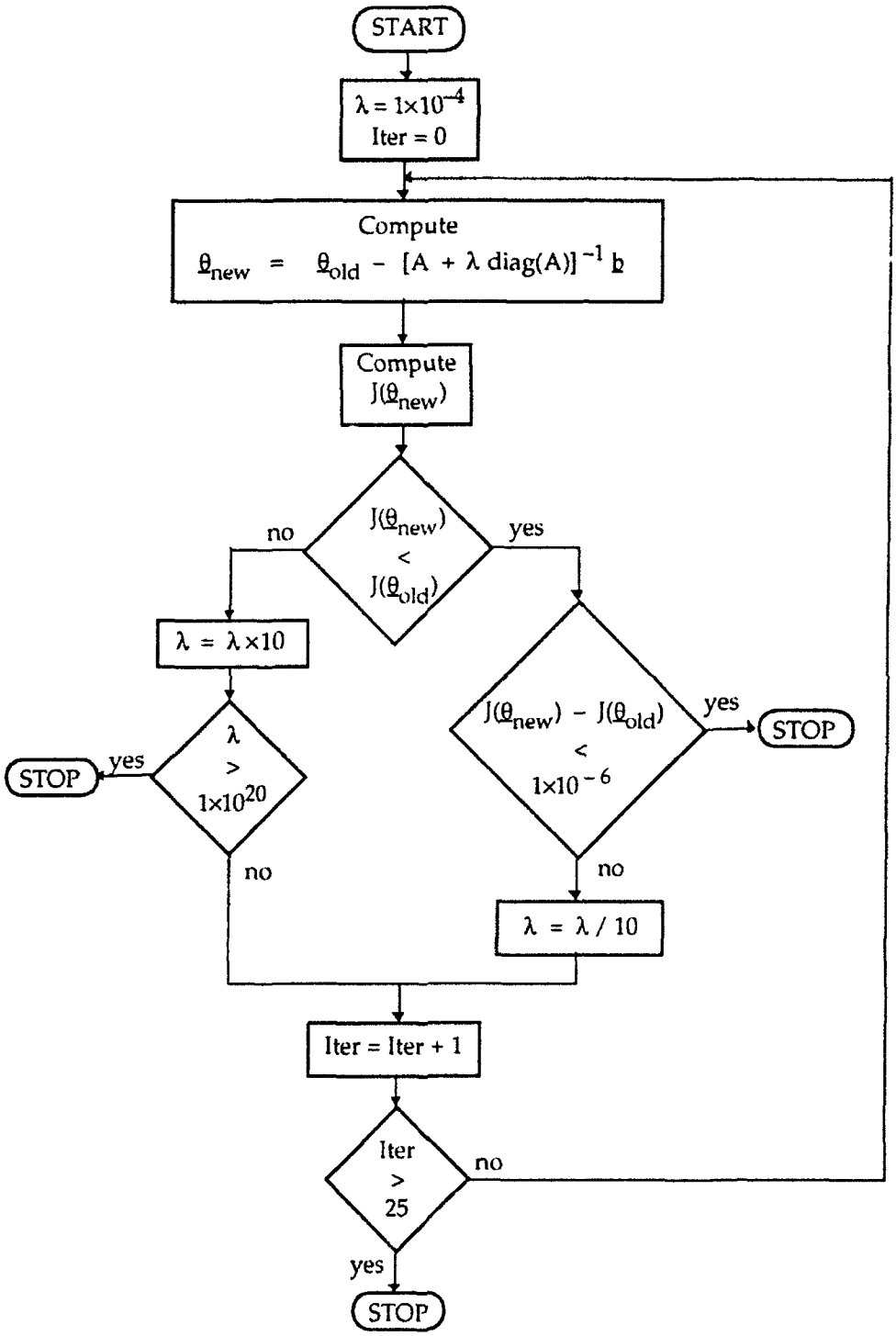


Figure 3.3: Levenberg-Marquardt Optimization Algorithm

Fig. 3.4 presents a pseudo-code definition of the *CLASS* algorithm with parameter building.

```
Set all parameters to zero and disable all of them from fitting.
Set the best overall score to infinity.
LOOP — Enable the next best parameter one parameter at a time.

        Set the best local score to the best overall score.
        Save the parameter values.
        LOOP — Find the next best (disabled) parameter.

                IF there is none left, exit this inner loop.
                Enable this trial parameter and fit the network.
                        Only the enabled parameters will be fitted.
                IF the current score is less than the best local score, set the
                        best local score to the current score and save the
                        best parameter values.
                Disable this trial parameter and reset the parameters to
                        their previous values.

        END
        IF the best local score is no better than the best overall score, there
                has been no improvement, so exit this outer loop.
        ELSE set the best overall score to the best local score and save the
                parameters.

END
Set the network parameters to the best parameters.
```

Figure 3.4: Pseudo-Code for *CLASS* Algorithm with Parameter Building

3.2.4 *CLASS* Convergence

The *CLASS* algorithm converges rapidly for two reasons:

1) *CLASS* makes use of curvature (second-derivative) information during optimization (Eq. 3:32 $\underline{\underline{A}}$ matrix). This curvature allows the ILS search to proceed in a Newton direction rather than just a gradient-descent direction.

2) *CLASS* does not use sigmoidal nonlinearities. These nonlinearities have very weak gradients far from the origin (see Fig. 2.6). Many iterations are required to traverse the objective function in the region of these near-zero gradient values; attempts to speed up this portion of the search via the introduction of a larger learning rate, μ, present difficulties when the search enters the steep region of the sigmoidal nonlinearity.

Figs. 3.5 and 3.6 show *CLASS* learning curves for networks trained to discriminate between three species of iris [25].

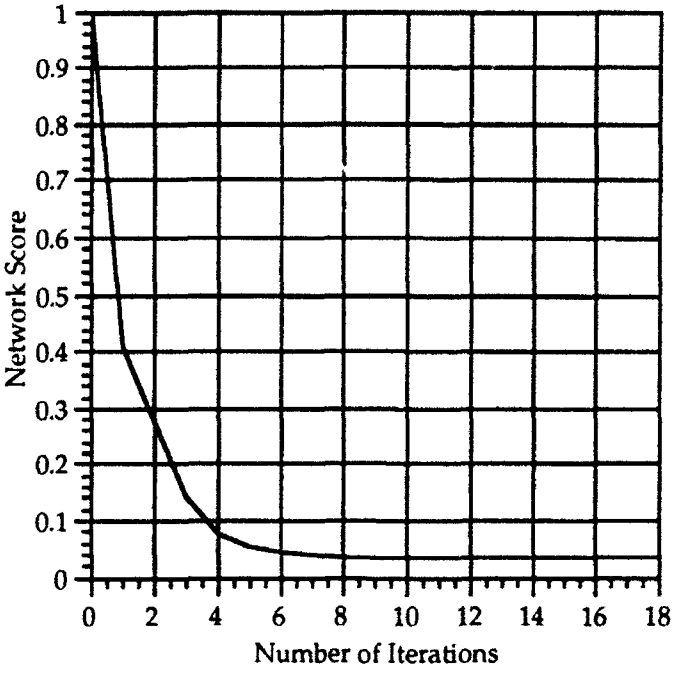


Figure 3.5: **Learning Curve for Three-Class Network Using LM Algorithm**

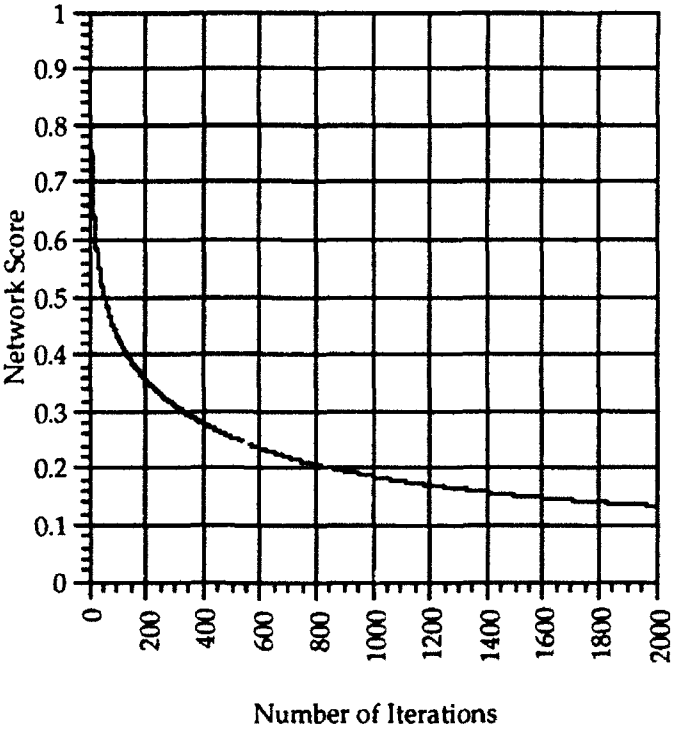


Figure 3.6: **Learning Curve for Three-Class Network Using LMS Algorithm, with $\mu = 0.1$**

The structures of the LM and LMS *CLASS* networks with learning curves shown in Figs. 3.5 and 3.6 were identical: the input vector contained four features, and the network was pre-structured with first-degree nodal elements. Both learning curves were obtained by performing multiple training runs with network weights randomly initialized between -0.5 and 0.5; however, the shape of the learning curves proved to be independent from the initial coefficient values, so only one learning curve is shown.

As can be seen from the figures, the LM-trained network (Fig. 3.5) had nearly converged by the sixth iteration, providing a 98.7% classification accuracy. The LMS-trained network (Fig. 3.6), on the other hand, had an accuracy of 97.4% and had still not converged at 2000 iterations. Additionally, multiple trial-and-error runs of the LMS network had to be performed to determine the learning rate, μ, of 0.1 that provided the most rapid convergence.

3.3 A Rapid Structure-Learning Classification Algorithm

As discussed in Section 3.2 above, the *CLASS* algorithm makes use of a fixed network structure, with terms that may be built as needed. It has been shown that when term building is not used, *CLASS* converges on a solution very rapidly (Section 3.2.3). Building the terms, however, places an increased computational burden on the algorithm for any large classification task, because a nonlinear Gauss-Newton optimization must be performed for each candidate coefficient subset. If, in addition to building terms, were the network structure allowed to grow from zero complexity to a level of optimal complexity (as implemented in *ASPN*), this computational burden would increase dramatically.

ASPN (and its predecessor, GMDH) can build estimation network structures rapidly because, at each stage of network construction, linear regression is used to find optimal coefficients rapidly in a single iteration. A way that a *classification* network structure of just-sufficient complexity can be found rapidly will now be described. Once the network structure is fixed, the coefficients may be globally optimized using an appropriate (non-quadratic) distortion function.

Recall that the probability that the i^{th} observation, $\underline{x}_i$, is a member of class k, is given by

$$p(\underline{y}_i \mid \underline{x}_i) = \frac{e^{\underline{y}_i \cdot \underline{s}_i}}{\sum_{k=1}^{C} e^{s_{i,k}}} \qquad 3:34$$

where $\underline{y}_i$ is the vector containing a value of one in the k^{th} position and zeroes everywhere else, C is the number of classes, and $\underline{s}_i$ is the vector of outputs from the C polynomial nodal elements.

Also recall that the optimal distortion function, in a maximum-likelihood sense, is given by

$$d(\underline{y}_i, \underline{s}_i) = -\ln\left[p(\underline{y}_i \mid \underline{x}_i)\right] = -\underline{y}_i \cdot \underline{s}_i + \ln\left(\sum_{k=1}^{C} e^{s_{i,k}}\right) \qquad 3:35$$

For the *CLASS* algorithm, $s_{i,C}$ is zero by definition, because only C–1 values are required to compute C probabilities. However, neither Eqs. 3:34 nor 3:35 require this restriction, so for now it will be dropped. Additionally, the subscript, i, will also be dropped for notational convenience.

If all C elements in the $\underline{s}$ vector are allowed to take on any real value, then it can be seen that for the vector of estimated probabilities, $p(\underline{y} \mid \underline{x})$, to take on the same values as the vector of known probabilities, $\underline{y}$, then the ideal output vector, $\underline{s}_I$, should contain a value of $+\infty$ in the k^{th} position and a value of $-\infty$ in all other positions. This is shown in Eq. 3:36 below.

$$\underline{s}_I = \begin{bmatrix} -\infty \\ \cdot \\ \cdot \\ -\infty \\ +\infty \\ -\infty \\ \cdot \\ \cdot \\ -\infty \end{bmatrix} \xrightarrow{\text{Eq. 3:34}} \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = p(\underline{y} \mid \underline{x}) = \underline{y} \qquad 3:36$$

Thus, if a network would output the appropriate $\underline{s}$ vector for all observations, containing $+\infty$ in the desired location, k, and $-\infty$ in all other locations, that network would provide perfect probability estimates. The remaining question, therefore, is what objective function will efficiently optimize the structure and coefficients of such a network.

For computational reasons, the first step in defining such an objective function involves approximating ∞ with a large number, Ω. How large Ω must be will be addressed later. This step results in a desired output vector

$$\underline{s}_d = \begin{bmatrix} -\Omega \\ \cdot \\ \cdot \\ -\Omega \\ +\Omega \\ -\Omega \\ \cdot \\ \cdot \\ -\Omega \end{bmatrix} \qquad 3:37$$

Note that $\underline{s}_d$ and $\underline{y}$ are related to each other by

$$\underline{s}_d = 2\Omega\,(\underline{y} - 0.5) \qquad 3:38$$

since $\underline{y}$ is a vector of binary numbers; thus, for $y_j = 1$, $s_j = \Omega$ and for $y_j = 0$, $s_j = -\Omega$.

During construction of a network designed to estimate $\underline{s}_d$, it is desirable not penalize the network for having outputs that are "closer" to ∞ than Ω, because for those cases the outputs are closer to the ideal, $\underline{s}_I$, than our approximation, $\underline{s}_d$. Additionally, if rapid optimization is to be possible at each step in the structure generation process, then the distortion function must be quadratic in the network output, $\underline{s}$. The following distortion function meets these criteria and may be used to optimize each element in the $\underline{s}$ vector

$$d(k, s_j) = \begin{cases} (\Omega - s_j)^2 & \underline{x} \in \text{class k } \textit{and } s_j < \Omega \\ (-\Omega - s_j)^2 & \underline{x} \notin \text{class k } \textit{and } s_j > -\Omega \\ 0 & \text{otherwise} \end{cases} \qquad 3:39$$

Notice that this function, because it is quadratic, is convex and everywhere twice differentiable. Fig. 3.7 illustrates such an objective function for the output of the j^{th} network, s_j. If the input vector, $\underline{x}$, is a member of class k, then the network output, $\underline{s}$, is driven toward Ω; whereas if $\underline{x}$ is not a member of class k, then $\underline{s}$ is driven toward $-\Omega$. The distortion function for the overall network is the sum of the distortion functions for each network output

$$d(k, \underline{s}) = \sum_{j=1}^{C} d(k, s_j) \qquad 3:40$$

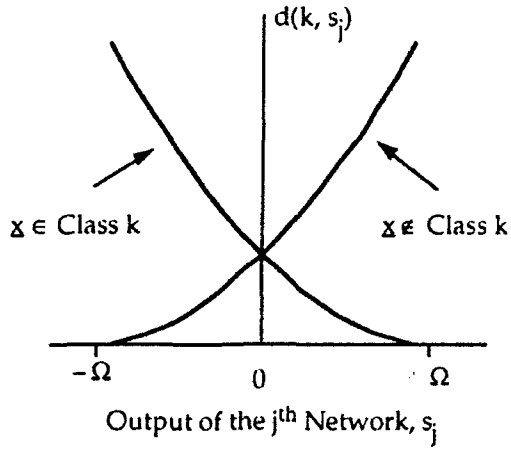


Figure 3.7: An Objective Function for Classification

If the coefficients are initialized to zero, the network outputs will be zero; thus, if Ω is non-zero, the first iteration of Eq. 3:40 will yield the same results as a simple squared-error distortion function for a vector output:

$$d(k, \underline{s}) = (\underline{s}_d - \underline{s})^T(\underline{s}_d - \underline{s}) \qquad 3:41$$

where $\underline{s}_d$ is the desired output vector having $+\Omega$ and $-\Omega$ elements the (positions of which depend on the true class, k). If, however, the minimization of the distortion function of Eq. 3:40 is iterated a second time from the operating point found in the first iteration, network outputs that are "too large" in the correct direction will not be penalized. Once again, this is desirable because these outputs are closer to ∞ than Ω. The performance improvement obtained via more than a single iteration of the optimization algorithm is still a research issue; nevertheless, sub-optimal classification can be achieved rapidly using one or two iterations of the above objective function.

We now turn to the question of how large Ω is required to be. Consider a C-class network that produces a desired response, $\underline{s}_d$, when interrogated with an input vector, $\underline{x}$, that is a member of class k. The estimated probability that $\underline{x}$ is a member of class k may be computed using Eq. 3:34

$$p(k \mid \underline{x}) = \frac{e^{\Omega}}{e^{\Omega} + (C-1)e^{-\Omega}} = p_d \qquad 3:42$$

Eq. 3:40 may be solved to find Ω as a function of p_d

$$\Omega = \frac{1}{2} \ln \left[\frac{p_d (C-1)}{(1 - p_d)} \right] \qquad 3:43$$

Eq. 3:42 confirms the result, presented above, that to achieve a perfect probability estimate of unity, Ω would have to equal ∞. However, values of Ω far less than infinity yield very adequate probability estimates, as shown in Fig. 3.8.

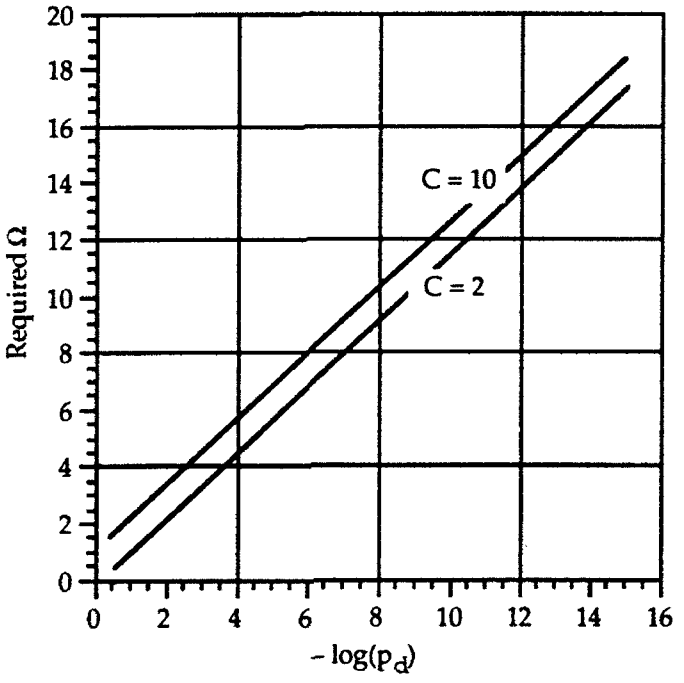


Figure 3.8: Ω vs. Desired Probability, p_d

Note that, in Fig. 3.8, an abscissa value of 2.0 corresponds to p_d of 0.99; an abscissa value of 4.0 corresponds to a p_d of 0.9999, etc. As can be seen from the figure above, Ω need not be very large to yield probabilities quite close to the ideal (unity). Additionally, optimal classification is not required at this stage of network construction; therefore a value of Ω = 10 should be more than adequate for most purposes when learning the classifier structure.

The discussion above implies that a squared-error loss function may indeed be used to determine rapidly the structure of a classification network. Although the structure and coefficients may constitute sub-optimal classifiers, they do approximate optimal classifiers, and are surely better than analyst guessing. Because Eq. 3:39 is a quadratic distortion function and converges rapidly for network elements that are linear in their coefficients, the method of growing network structures from zero complexity to just-sufficient complexity (Section 2.3.3) may be applied efficiently. All that is required is an additional complexity penalty, as shown in Eq. 3:33 above.

In Section 3.2.1, we showed that for a C-class network, only C–1 nodal elements are *required*. C nodal elements may, of course, be used. This led to the

interpretation that the outputs of the nodal elements are natural logarithms of the ratio of the conditional probabilities, and class C, the class for which there was no corresponding nodal element, was considered a baseline class. For pre-structured *CLASS* networks, the choice of C is arbitrary; however, if data is available for a background, no-fault, or "all other" class, this is often the best choice for class C. With the structure-building approach presented here, the baseline class, C, may be chosen to be the class for which network performance is the poorest. In this way, rather than estimating the probability of membership in class C from a poorly performing sub-network, one may infer that probability from the probabilities of membership in the other C–1 classes, and those probabilities are estimated more accurately than the probabilitiy of class C. Thus, if one of the C networks is performing poorly in relation to the other C–1 networks, then that sub-network output may be assigned identically to zero. When to make this decision and the performance advantages resulting from it are current topics of research.

Once the C (or C–1) network structures have been determined, the squared-error objective function of Eq. 3:31 may be replaced with the more appropriate (for classification) logistic-loss objective function of Eq. 3:5, and the network optimized globally. Global ILS optimization of a multi-layer network, as described in Section 2.3.4.4, requires analytic forms for the following:

- analytic forms of the first and second partials of the objective function with respect to the network outputs, ∇d_{s_0} and $\nabla^2 d_{s_0}$,
- an analytic form for the first derivative of the post-transformation, h(z), and
- an analytic form for the gradient of an element output with respect to its input vector, $\nabla f_{\underline{x}}$.

Analytic forms of the gradient and Hessian of the objective function were derived above in Section 3.2.2, and there is no h(z) in the proposed polynomial neural network. Thus, all that remains is to define the gradient of an element output with respect to its input vector. For polynomial nodal elements of the form described in Eqs 3:24 - 3:27, the gradient may be defined in terms of the J × D matrix of indices, $\underline{\underline{K}}$

$$\frac{\partial z}{\partial x_i} = \sum_{m=1}^{J} k_{mi}\theta_m \prod_{n=1}^{D} x_n^{k'_{mn}} \qquad 3:45$$

where J is the number of terms in the expansion, D is the number of inputs to the expansion, and k' is defined for each input x_i as

$$k'_{mn}(i) = \begin{cases} k_{mn} - 1 & \text{if } n = i \\ k_{mn} & \text{otherwise} \end{cases} \qquad 3:46$$

Notice that the non-zero terms in Eq. 3:45 correspond to the rows of the $\underline{\underline{K}}$ matrix that have non-zero elements in column i.

3.4 Algorithm for Synthesis of Polynomial Neural Networks for Estimation (*ASPN*)

The *ASPN Facility* synthesizes static *estimation* neural networks having no internal feedback paths, no internal time delays, no post-transformations, and (in general) a polynomial basis function (time-dealyed input or output data may, of course, be provided to *ASPN*). The main function of *ASPN* is to create a neural network that estimates the value(s) of a dependent variable or variables when interrogated with an input observation vector. In synthesizing a network, *ASPN* selects the most relevant inputs from a list of candidates, determines the most appropriate structure (connectivity) of the network, determines the best algebraic function to use at each node, and optimizes the weights in the network. With *ASPN*, the network structure evolves from the simplest form (a single input connected to the output) to a feedforward network having just-sufficient complexity for the database under consideration. Because *ASPN* synthesizes a static network, the network output vector is a single-point transformation of the input data.

3.4.1 Network Structure

Each *ASPN* network is a combination of nodal elements, where each nodal element may contain a series expansion made up of terms that are a subset of the complete Kolmogorov-Gabor (KG) multinomial (either pre-defined or analyst defined). Just as *CLASS*, with its "building" option enabled, selects the best inputs to use in network synthesis, so does *ASPN*. For the most part, *ASPN* nodal elements do not have more than three inputs.

Whereas GMDH used one basic element type (Section 2.4.1), *ASPN* makes use of a number of subsets of the complete KG multinomial (Eq. 3:6). The pre-defined polynomial elements used by *ASPN* are listed below, with their corresponding $\underline{\underline{K}}$ matrices:

Single:

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3 \qquad 3:47$$

$$\underline{\underline{K}} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \qquad 3:48$$

Double:

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_i x_j + \theta_4 x_i^2 + \theta_5 x_j^2 + \theta_6 x_i^3 + \theta_7 x_j^3 \quad 3:49$$

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 3 \\ 1 & 0 \\ 2 & 0 \\ 3 & 0 \\ 1 & 1 \end{bmatrix} \quad 3:50$$

Triple:

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_k + \theta_4 x_i x_j + \theta_5 x_i x_k + \theta_6 x_j x_k + \theta_7 x_i^2 + \theta_8 x_j^2 + \theta_9 x_k^2 + \theta_{10} x_i^3 + \theta_{11} x_j^3 + \theta_{12} x_k^3 + \theta_{13} x_i x_j x_k \quad 3:51$$

$$\underline{\underline{K}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 3 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad 3:52$$

Linear:

$$y = \theta_0 + \sum_{i=1}^{D} \theta_i x_i \quad 3:53$$

Note that whereas the "single" is a one-input, third-degree, additive polynomial, the double and triple cannot be described completely by simply placing limits on the degree, coordinate degree, or interaction order. In this sense, the double and triple have a low expansion density and may be described as *sparse* expansions (Section 2.2.2.2). *ASPN* also allows the analyst to define a custom polynomial nodal element simply by specifying its $\underline{\underline{K}}$ matrix; however, this is seldom necessary.

In addition to the polynomial elements described above, the following transcendental nodal elements are sometimes desirable.

Exponential:

$$y = \theta_0 + \theta_1 e^{(\theta_2 x_i)} \qquad 3:54$$

Cube-Root:

$$y = \theta_0 + \theta_1 (x_i - \theta_2)^{1/3} \qquad 3:55$$

Sigmoidal:

$$z = h\left(\theta_0 + \sum_{i=1}^{i=D} \theta_i x_i \right) \qquad 3:56$$

$$\text{where } h(z) = \frac{1}{1 + e^{-\gamma z}} \qquad 3:57$$

ASPN currently implements the *exponential* or *cube-root* element, and the *sigmoidal* element may be incorporated readily into the algorithm. However, experience has shown that these functions are rarely selected by *ASPN* during model syntheses.

Using *ASPN*, the model for each dependent variable usually consists of a network of polynomial elements arranged in layers; such a network is shown in Figure 3.9. In this figure, y_1 and y_2, the two outputs of the example network, are simultaneous estimates for two dependent variables. The sub-network for y_1 is three layers deep and employs six elements and six original inputs. The y_2 sub-network consists of only two elements, one on each of its two layers, and employs four original inputs. Note that there is cross-coupling, or sharing of results, between the sub-networks, and only six of the original 12 or more candidate inputs are used. As with all *ASPN*-synthesized networks, the inputs pass through a mean-sigma normalization stage (N) before introduction into the network, and the dependent variable estimates are re-scaled, i.e., unitized (U) at their outputs.

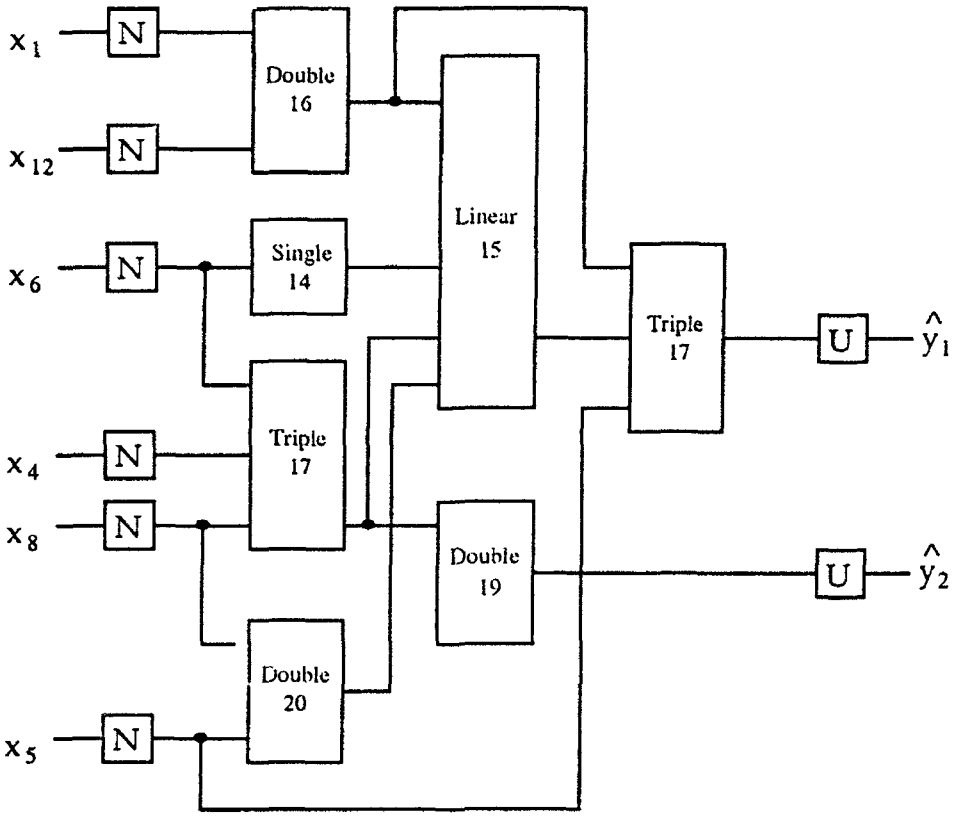


Figure 3.9: Sample Polynomial Network

3.4.2 Optimization via Linear Regression

For each layer of the *ASPN*-synthesized network, a succession of the various nodal functions, with different combinations of inputs, is fitted and scored. Fitting consists of computing the optimum values for the candidate element coefficients using a batch least-squares technique. The candidate element is fitted in such a way that it attempts to solve the entire input-output mapping problem by itself [42]. A candidate element is scored using a model selection criterion that considers a loss function and a complexity penalty (see Section 2.3.2). The model selection criterion used by *ASPN* is either the predicted squared error, PSE, minimum description length, MDL, or predicted classification error, PCE.

In its simplest form, PSE is written

$$PSE = FSE + 2K \sigma_p^2 / N \qquad 3:58$$

in which σ_p^2 is a prior estimate of the true error variance that does not depend on the model being considered, K is the number of parameters within the model, N is the number of exemplars in the training database, and FSE is the fitting squared error (i.e., the mean squared error)

$$FSE = \frac{1}{N}\sum_{i=1}^{N} | y_i - s_i |^2 \qquad 3:59$$

where $s_i = f(\underline{x}_i, \hat{\underline{\theta}})$ is the network output.

PSE is a very conservative criterion provided the chosen σ_p^2 exceeds the true error variance, a condition readily verified upon completion of model synthesis. This conservatism is important in engineering and predictive estimation applications. PSE is *not* dependent on the shape of the error distribution.

In addition to PSE, *ASPN* provides two other distortion functions. The MDL criterion is based on Shannon coding theory and follows the equation

$$MDL = \text{Ceiling}\left[BN + \frac{N}{2}\log(2\pi e\, FSE\, N_c)\right] \text{ bits,} \qquad 3:60$$

where B is the number of bits that would be used to encode each observation error (say, 16), N_c is N raised to the (K+1)/N power, and the Ceiling function refers to use of the next-highest integer. Use of the MDL criterion is best where the "shortest explanation" of the existing data is believed to be appropriate.

The final model selection criterion available in *ASPN* is PCE:

$$PCE = \left[1.0 - \frac{1}{N}\sum_{i=1}^{N} H\left(|y_i - \hat{y}_i|, C_{Tol}\right)\right] + C_{Mult}\frac{K}{N} \qquad 3:70$$

where:

$$H \equiv \begin{cases} 1 \text{ if } |y_i - \hat{y}_i| < C_{Tol} \\ 0 \text{ if } |y_i - \hat{y}_i| \geq C_{Tol} \end{cases} \qquad 3:71$$

y_i is the true output given the i^{th} input vector i. (For PCE, true outputs must be positive.)

$\hat{y}_i$ is the candidate model output given input vector i

C_{Tol} is the classification tolerance (default value 0.5)

C_{Mult} is the complexity penalty multiplier (nominally 1.0).

Once an element has been fit, the analyst has the option of instructing the algorithm to *carve* away terms that are not statistically justified. If an element has a polynomial expansion with J terms, the carving algorithm tries all possible

combinations of J–1 terms. If one of the new elements has an improved score, the original element (J terms) is replaced by the best new element (J–1) (i.e., the term that was carved was not statistically justified). The carving process continues, removing one term at a time, until the removal of terms no longer improves yields an improvement in the score. This method of carving is referred to as a "greedy" carving algorithm in that once a term has been removed, it can never be considered again. A "complete" or "exhaustive" carving algorithm, on the other hand, would first consider all J–1 combinations of the original element, then all J–2 combinations, etc. While the exhaustive carving algorithm is guaranteed to find the optimal coefficient subset, experience has shown that the additional computational burden is rarely justified.

When all possible candidate elements have been fitted and scored by *ASPN*, a given layer is constructed by selecting the "best" elements, as gauged by the model selection criterion. The maximum number of "best" elements allowed in each layer and the maximum number of layers are specified by the analyst. To improve performance (minimize the modeling criteria), elements may be joined in series and in parallel, creating a feedforward network. The inputs for the next layer include the previous layer outputs and the network inputs. Since only the best elements or nodes are retained in each layer, successive layers can only improve or maintain the performance of the previous layer. The growth of the network is halted when the model ceases to improve with the addition of new elements, or the maximum number of layers has been reached. Candidate elements (and their inputs) not required for the final network output(s) are then eliminated.

3.4.3 *ASPN* Convergence

ASPN networks synthesize rapidly for the following reasons:

1) The PSE criterion is quadratic in the network output, and the polynomial nodal elements are linear in their parameters; therefore, during network construction, the coefficients of each nodal element may be optimized in a single ILS update step.

2) The polynomial elements in an *ASPN* network contain relatively few coefficients; therefore, the size of the system of linear equations to be solved at each step is relatively small.

3) The correlation information (the $\underline{\underline{A}}$ matrix in Eq. 2:50) from each linear regression step may be saved and reused on subsequent linear regression steps for other candidate network structures.

4) *ASPN* employs a set of heuristics to limit the number of candidate inputs to layers other than the input layer. These heuristics make available only the "most promising" original inputs and prior layer outputs to a layer

that is being constructed, greatly reducing the number of potential combinations of layer inputs.

Because at each stage in network construction, coefficients may be optimized rapidly and globally in a single step, ample computational resources remain to solve the network construction problem. It is not relevant to discuss *ASPN* convergence in terms of numbers of iterations of a search algorithm; but experience has shown, as with *CLASS*, that *ASPN* trains much more rapidly than the LMS-based network training algorithms, such as backpropagation.

3.5 Algorithm for Synthesis of Dynamic Polynomial Neural Networks for Estimation (*DynNet*)

Recently, the authors have been investigating two specific aspects of the synthesis and use of neural networks for estimation. The first concerns the use of tapped delay lines (i.e., memory) within the network elements, whereas the second pertains to the use of feedback connections internal to the network. By including delays (which are usually time delays in applications, although are not necessarily so) and feedback connections, dynamic networks can:

- compute time-varying transformations given static inputs
- perform infinite impulse response (IIR) filtering operations along with finite impulse response (FIR) filtering operations
- provide a phase-shift operator between time-varying inputs
- provide better modeling accuracy with fewer internal degrees of freedom
- handle time-series data more naturally than feedforward networks

The core transformation of the networks described here uses the polynomial basis function given in Eq. 2:6. Because the networks described in this section usually incorporate internal time delays and feedback paths, they will sometimes be referred to as *dynamic polynomial neural networks* (DPNNs). Although DPNNs have been used primarily for function estimation, it is conceivable that they may be adapted to allow for classification of spatio-temporal input patterns. Additionally, these networks may be used to generate features for classification purposes (see Section 4.3.1).

3.5.1 Network Structure

The candidate nodal elements for a DPNN are exactly the same as those available for a *CLASS* network: additive, multilinear, and complete polynomials with no post-transformation, h(z). DPNNs, however, unlike static networks incorporate delay banks internal to the elements. Thus, when the analyst specifies an element structure, the numbers and values for the time delays must also be

specified. Additionally, DPNN networks allow the output of a given layer to feed back upon itself, or, the output of the entire network to feed back upon itself.

There are two approaches to adaptive IIR filtering that may also be applied to dynamic neural networks; these are known as the *equation-error* and *output-error* methods. In the equation-error method, the output estimate is found using previous output *measurements* obtained from the system itself or from the training database. Fig. 3.10 shows the application of an equation-error network to a system identification problem:

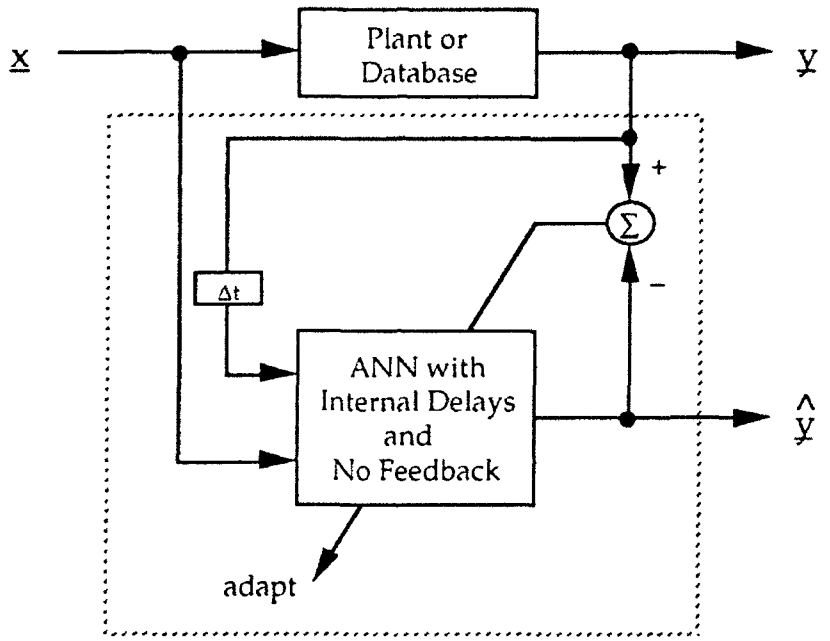


Figure 3.10: Equation-Error System Identification

Because an equation-error network need not contain internal feedbacks, one can treat the equation-error formulation as a feedforward network. As such, the rapid batch ILS techniques used for the synthesis of static feedforward polynomial neural networks can be applied to the synthesis of equation-error dynamic networks.

It is not, however, always possible (nor necessarily desirable) to use the equation-error formulation. In an on-line situation, prior measurements of the system output(s) may not always be available for use as inputs to the network. Even if prior measurements are available, it has been shown that the equation-error formulation can lead to biased coefficient estimates if the underlying system dynamics truly contain feedback dynamics; especially in cases where the measured responses contain additive noise [66].

The output-error formulation feeds back previous *estimates* of the output values. Fig. 3.11 shows the application of an output-error network to the same system identification problem that was shown in Fig. 3.10:

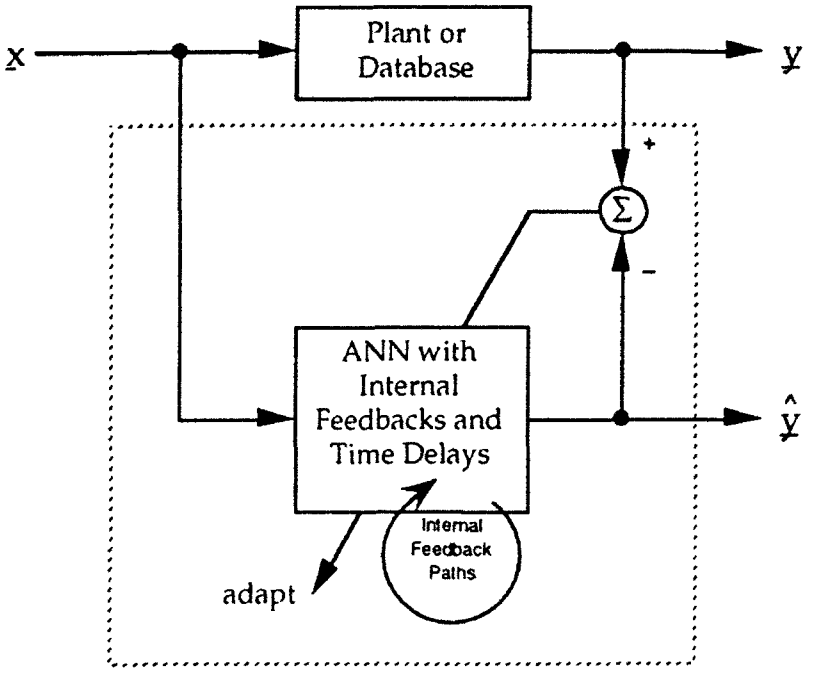


Figure 3.11: Output-Error System Identification

The output-error network has both internal time delays and internal feedback paths, and as such has all of the advantages of dynamic networks that were listed above.

The *DynNet* algorithm can synthesize both equation-error and output-error dynamic neural networks. Since internal delays are the only aspect of equation-error networks that distinguish them from static feedforward networks, this section will instead concentrate on the creation and optimization of output-error networks. However, it will be seen that even in the synthesis of output-error networks, a number of equation-error networks may be created as intermediate steps.

3.5.2 Network Training

The potentially improved performance of output-error dynamic networks comes at a price. Because of network feedbacks, the nodal elements of these networks can become nonlinear in their coefficients. Fig. 3.12 shows a simple example of a network containing feedback and delays. Note that Δt represents a delay of one sample step.

In Fig. 3.12, the output, z_A, of nodal element A depends on the input vector $\underline{x}_i$ and on the coefficients, $\underline{\theta}_A$, of element A, i.e.

$$z_A = f(\underline{x}_i, \underline{\theta}_A) \qquad 3:72$$

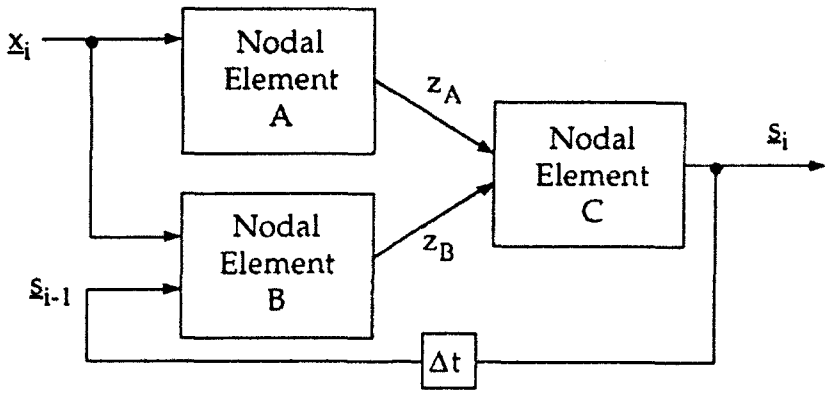


Figure 3.12: Sample Dynamic Network

Thus, if f(·) is linear in its coefficients, the coefficients of A may be optimized using a batch least-squares technique. For nodal element B, however, the situation is different:

$$z_B = f(\underline{x}_i, \underline{s}_{i-1}, \underline{\theta}_B) \qquad 3:73$$

But $\underline{s}_{i-1}$ is also some function, $g(\underline{\theta}_B)$, involving $\underline{\theta}_B$; thus

$$z_B = f(\underline{x}_i, g(\underline{\theta}_B), \underline{\theta}_B) \qquad 3:74$$

which is usually not linear in the coefficients $\underline{\theta}_B$. Any nodal element that is involved in a feedback loop is nonlinear in its coefficients, and the batch least-squares techniques that are useful in synthesizing static feedforward networks (see Section 3.4) cannot be used to fit these nodal elements. Instead, the optimization of dynamic networks and dynamic nodal elements generally involves one of the more computationally intensive iterative techniques described in Section 2.3.4.

Feedback connections also introduce a further difficulty concerning network optimization. Parameter nonlinearities tend to distort the surface of the objective function and often introduce multiple minima, making it impossible to guarantee global optimization in all cases. Because of these multiple minima, great care must be taken when initializing the parameters for an iterative search; additionally, it may be desirable to use a multi-modal search method that is better suited to finding the global minimum of a surface having numerous local minima.

Another cost of dynamic networks is the initialization required, which results from the use of sample delays. System input and/or output values must be used to fill all delay banks before the element core transformation can generate an output value. This does not pose a problem if one can afford to wait M observations before generating the network output, where M is the maximum delay value. In other applications, however, such as tracking and control, output estimates must be generated beginning at the first observation. In these cases, one may need to create a static network to provide the network output or to initialize the values of the delay banks contained within the dynamic network.

To overcome some of the problems associated with DPNNs, *DynNet* extends a principal applied frequently in the generation of adaptive IIR filters: *the equation-error structure that best solves the mapping problem is close to the desired output-error structure* [74]. Thus, when determining the initial structure of a DPNN, one may replace the feedback values with the known values of the output. This is true even for layer feedbacks, since the output of each layer in a PNN is an estimate of the true system output. Once this assumption is made, the DPNN structure may be determined in exactly the same way the *ASPN* structure is determined, by treating prior output values as candidate inputs.

Fig. 3.13 shows the general algorithm for constructing either an output-feedback or intra-layer-feedback MISO DPNN. This algorithm may be easily extended to the MIMO case as in *DynNet;* however, the MISO case is shown here for simplicity.

The *DynNet* algorithm shown in Fig. 3.13 may be summarized as follows:

1. Find the M best MISO equation-error *Dyn3* networks for each output, with the element complexities limited as described above. If a database output is used as an input variable, make sure it passes through a one-step sample delay.

2. To account for additional error that might be introduced when the true equation-error (database) values of the network output are replaced by feedbacks of the output estimates, add an additional PSE penalty term proportional to the prior estimate of the error variance of the variable that will be fed back.

3. Recombine the elements from Step 1 so that there are M potential MIMO layers, with each layer containing one element corresponding to a network output.

4. If intra-layer feedback is desired, replace the *equation-error* network inputs which are prior values of the output columns of the training database, with feedbacks of the output estimates, thus creating M potential *output-error* first layers.

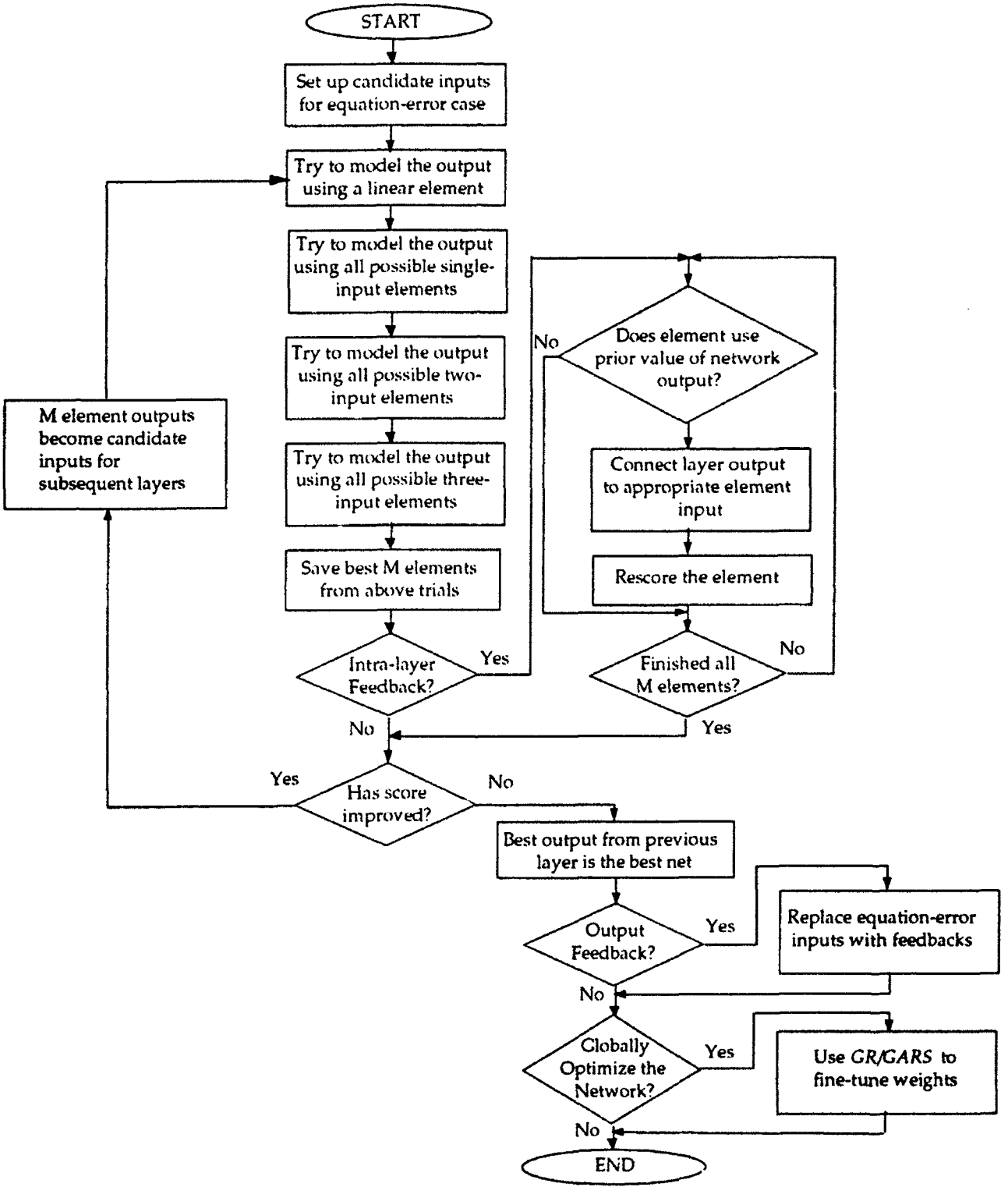


Figure 3.13: *DynNet* **Algorithm for Constructing a DPNN**

5. Optimize the coefficients of each layer using a *GR/GARS* search technique. (See Section 3.5.3.)

6. Continue adding layers until the overall network score ceases to improve. Subsequent layers are built in the same way as the first layer, except that in addition to system inputs and outputs, every element on subsequent layers must have at least one input which comes from the immediately preceding layer.

The above algorithm may be tuned and modified in a number of ways depending on the application.

3.5.3 Random Global Optimization Techniques

In principle, DPNNs could be optimized using a combination of the ILS optimization techniques described in Sections 2.3.4.5 and 2.3.4.6, respectively. Currently, however, the *DynNet* algorithm uses random optimization techniques that do not depend on any analytic gradient information. Presently, the authors make use of one of three techniques: (1) *Guided Random Search (GR)*, (2) *Guided Accelerated Random Search (GARS)*, and (3) The *Gambit* Search Method.

3.5.3.1 *Guided Random Search* (*GR*) [19]

The *Guided Random* (GR) *Search* algorithm is a random optimization procedure intended primarily for the initial and final stages of numerical searches. It has multimodal search capabilities, but these are not as powerful as those of the *GARS* algorithm, which is described in Section 3.5.3.2. The basic *GR* algorithm is quite simple, yet it embodies many qualities of other, more sophisticated, search algorithms.

GR uses an "amoeba," which is a set of search points, typically five, comprising the current information to be used by the search algorithm. The amoeba moves through the search space by adding and removing points from its search set in the following manner:

1. Remove worst point from the amoeba.

2. Add most recent point to the amoeba.

3. Recompute mean and standard deviation of the amoeba for each search dimension.

4. Determine the next search point to test using the following vector equation:

$$\underline{\mu}_g = \underline{X}_n^* + speed \cdot (\underline{X}_n^* - \underline{\mu}_n) \qquad 3:75$$

$$\underline{X}_{n+1} = \underline{\mu}_g + \textit{dispersion} \cdot N[\underline{\mu}_g, \underline{\sigma}_n] \qquad 3:76$$

where n denotes the iteration number and:

$\underline{\mu}_g$ is the mean of the goal region,

$\underline{X}_n^*$ is the location of the best point in the amoeba,

$\underline{\mu}_n$ is the vector mean of the five amoeba points,

$\underline{X}_{n+1}$ is the next search point to be tested,

$\underline{\sigma}_n$ is the vector standard deviation of the five amoeba points,

$N[\underline{\mu}_g, \underline{\sigma}_n]$ is a Gaussian random vector of mean $\underline{\mu}_g$ and standard deviation $\underline{\sigma}_n$, and

speed and *dispersion* are search parameters (constants) set by the analyst, each typically to unity.

Thus, the strategy of the amoeba search is very simple: move from the centroid of the amoeba beyond the point with the best score. The distance beyond the current best point to use, the "goal" region, is proportional to the distance between the best point and the amoeba centroid. With *speed* set to unity, the mean of the goal region for the next search point will be collinear with $\underline{X}_n^*$ and $\underline{\mu}_n$, at a distance $(\underline{X}_n^* - \underline{\mu}_n)$ beyond $\underline{X}_n^*$. The shape of the goal region is determined by the standard deviations of the current amoeba.

If an amoeba repeatedly receives good points in a particular direction it will stretch out along that direction and thus begin to accelerate. (See Fig. 3.14.) If an amoeba comes across an area representing a minima, it will surround that region and shrink, causing it to focus on that region or, when necessary, "thread the needle" and re-expand after getting through that portion of the search space.

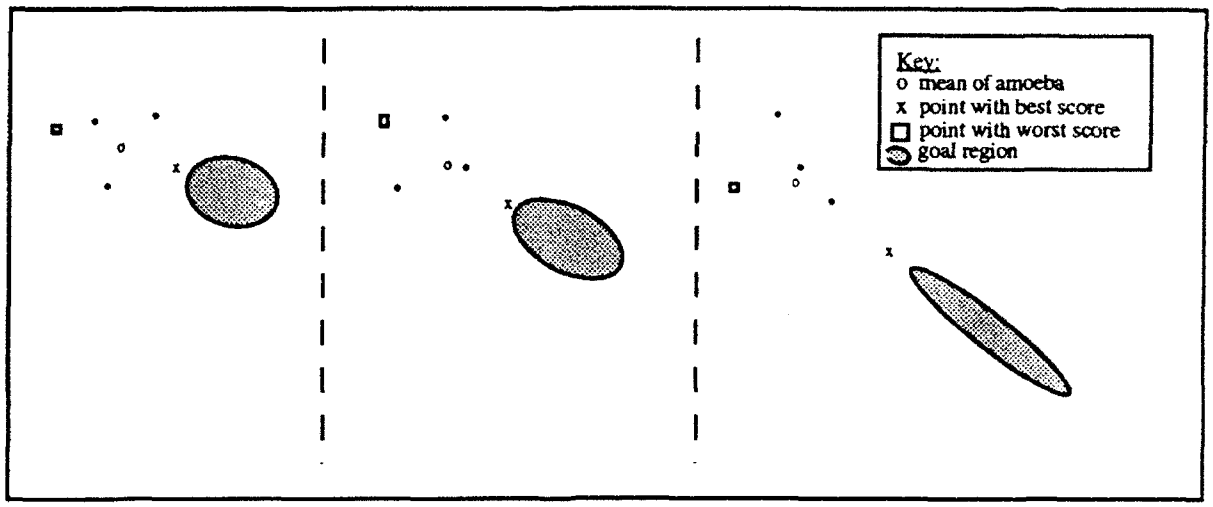


Figure 3.14: Sample *GR* Amoeba Acceleration

To initialize the *GR* search, initial values and standard deviations for each search dimension are specified.

3.5.3.2 *Guided Accelerated Random Search (GARS)* [19]

The *Guided Accelerated Random Search (GARS)* algorithm is a random optimization search intended primarily for the mid-game stage of numerical searches. *GARS* is a powerful multimodal search technique that avoids trapping in local minima of the performance surface. It is also effective when there is a shallow score slope to be traversed, in which case the *GARS* acceleration/deceleration capabilities quickly find the local minimum.

To govern random experiments, the *GARS* algorithm uses the following vector equation:

$$\underline{X}_{n+1} = \underline{X}_n^* + dispersion \cdot N[0, \underline{\sigma}_n] \qquad 3:77$$

where $\underline{X}_n^*$ is the location of the best-to-date trial, $N[0, \underline{\sigma}_n]$ is a Gaussian random vector of mean zero and standard deviation $\underline{\sigma}_n$, and *dispersion* is a search parameter (constant) set by the analyst, typically to unity. Note the similarity of Eq. 3:77 for *GARS* to Eq. 3:76 used in the *GR* search. Whereas *GR* conducts its random explorations in a region removed from $\underline{X}_n^*$, *GARS* performs its random trials in a region centered at $\underline{X}_n^*$.

Whereas *GR* establishes $\underline{\sigma}_n$ from the standard deviation of the five amoeba points, *GARS* determines $\underline{\sigma}_n$ from the score of the best-to-date trial. Thus, for *GARS*

$$\underline{\sigma}_n = \frac{J_n^*}{J_0^*} \cdot \underline{\sigma}_0 \qquad 3:78$$

where:

$\underline{\sigma}_n$ is the standard deviation vector used in computing $\underline{X}_{n+1}$

$\underline{\sigma}_0$ is the initial standard deviation vector

J_n^* is the best-to-date score as of iteration n

J_0^* is the initial best-to-date score.

J_0^* can be the value determined from an opening *GR* search. $\underline{\sigma}_0$ is usually specified by the analyst using *GARS*, but may be determined from an opening search with *GR*.

Using Eq. 3:78, the standard deviations of the random trial components shrink to zero as the score J_n^* approaches zero (perfection). However, even when the standard deviations become small, the search can still move quickly to remote regions of the search space, propelled by a deterministic acceleration heuristic that will now be described.

Once $\underline{X}_{n+1}$ is selected, the new point and the corresponding score are computed. If a new best score is not found at this new point, another random trial is selected and another new point is tested. If a new best score is found, the step $\Delta\underline{X} = \underline{X}_{n+1} - \underline{X}_n$ is multiplied by two to establish the next trial. As long as consecutive new best scores are found, $\Delta\underline{X}$ is repeatedly doubled, causing exponential acceleration of the search. Once a new best is not found, the search decelerates (it has gone too far) and tries again. If deceleration fails to improve the score, a final point is attempted on the opposite side of the best point found so far from the decelerated attempt, and the search resets, finding a new random $\Delta\underline{X}$ with acceleration factor of one.

Fig. 3.15 presents details of the *GARS* algorithm.

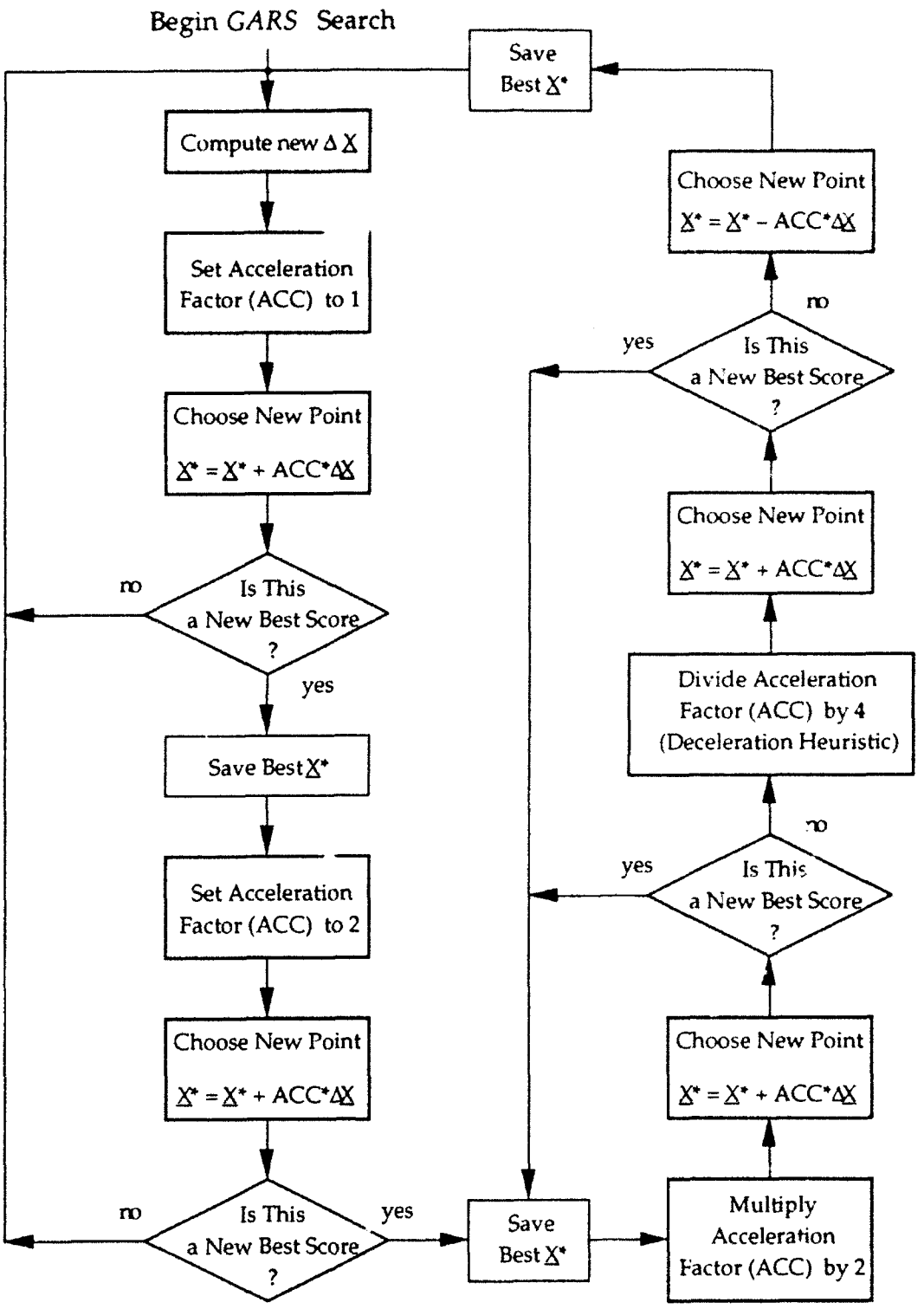


Figure 3.15: *GARS* Algorithm Block Diagram

3.5.3.3 *Combined GR/GARS Search* [19]

Care must be taken in initializing *GARS*. If *GARS* is initialized with standard deviations that are too large, the random phase may not readily find a new best score from which to begin the acceleration process, thus increasing search time. However, initial standard deviations that are too small may prolong the search time by requiring very large accelerations to provide meaningful improvement of the best score. Therefore, *GARS* is most effective when preceded by a search algorithm such as *GR*. The flow chart for a combination of *GR* and *GARS* (called *GR/GARS*) is shown in Fig. 3.16. The *GR* startup finds a good region for *GARS* and, because *GR* sets its own standard deviations based upon the four best scores found and the most recent point tested, an estimate of standard deviations is made available for use by *GARS*.

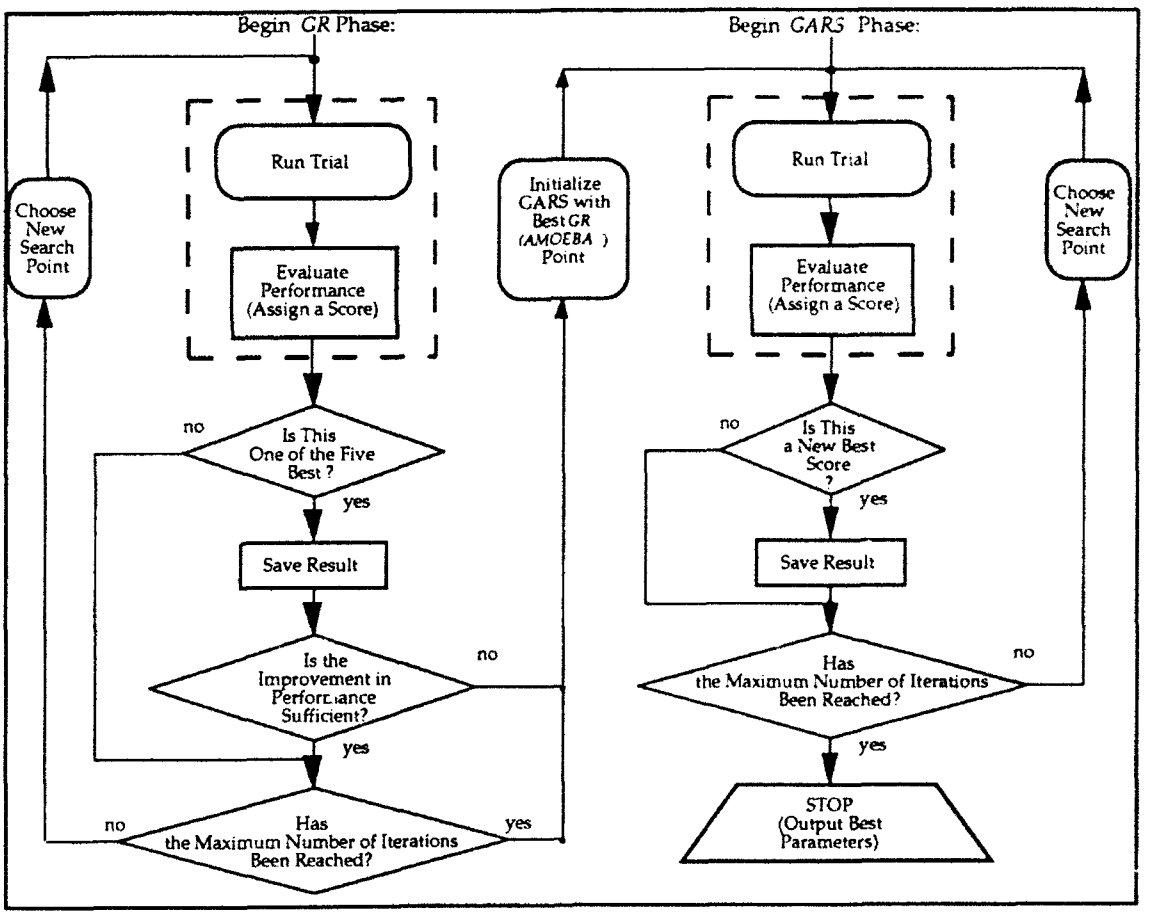


Figure 3.16: Combined *GR/GARS* Search

Fig. 3.17 shows a representative search convergence (learning) curve for *GR/GARS*. The abscissa of Fig. 3.17 is the iteration number, and the ordinate is the relative penalty assigned by the utility function. Note the rapid acceleration of search convergence several times during the search.

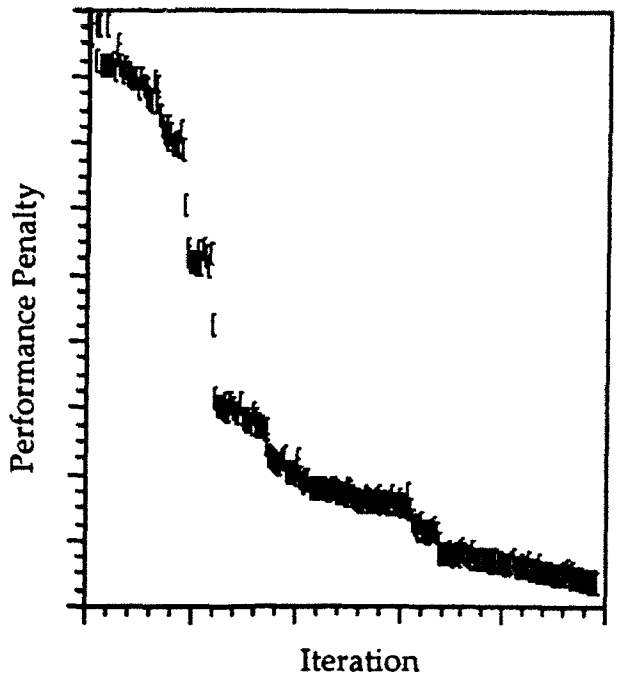


Figure 3.17: Example Learning Curve for *GR/GARS* Search

3.5.3.4 *GAMBIT Search*

The *GAMBIT* search is an alternative means of combining *GR* and *GARS* searches. *GAMBIT* incorporates two amoebas and a mid-game *GARS* search within a search "cycle." A cycle consists of the following four stages:

1. Perform an analyst-specified number of gambits: random trials normally distributed as $N[\underline{X}^*_n, \textit{dispersion} \cdot \underline{\sigma}_o]$, used to fill a ten-point "global" amoeba excluding $\underline{X}^*_n$
2. Perform a "global" amoeba search until:
   (a) amoeba is filled with better points, or
   (b) search exceeds an analyst-specified maximum number of consecutive unsuccessful trials.
3. Perform *GARS* search using fixed standard deviations, computed by "local" or "global" amoeba, until maximum number of consecutive unsuccessful trials is exceeded, where an improvement in score below *tolerance* also counts as an unsuccessful trial.
4. Perform "local" amoeba search, where the amoeba is populated to include $\underline{X}^*_n$, as determined by *GARS*, and four points distributed as $N[\underline{X}^*_n, \textit{dispersion} \cdot \underline{\sigma}_n]$ until:
   (a) Two times the maximum consecutive unsuccessful trials is exceeded, at which point the termination condition is tested:

i. If true, the *GAMBIT* search is finished.

ii. If false, the "local" amoeba hands off $\underline{X}_n^*$ to the "global" amoeba, and one full cycle has been completed, or

(b) Five times the size of the "local" amoeba better points is found, at which point *GARS* is started using $\underline{\sigma}_n$, as computed by the "local" amoeba.

When initializing the search, the analyst sets $\underline{X}_0^*$, the initial best point, and $\underline{\sigma}_0$, the "global" standard deviations, which play an important role in the powerful multi-modal capabilities of *GAMBIT*. The global amoeba is populated after a handoff from the "local" amoeba by choosing points normally distributed about the current best point, $\underline{X}_n^*$, but not including it. The relatively large "global" standard deviations set by the analyst allow the search to jump out and escape a local minimum and converge to better solutions elsewhere in the search space. If the search reaches the same point, within some analyst-specified tolerance, on three consecutive cycles, *GAMBIT* assumes that the point is a global minimum and sets the termination condition to "true."

GAMBIT incorporates the best characteristics of *GR* and *GARS*. *GR* gives the *GAMBIT* search the ability to maneuver through difficult search spaces, and is particularly useful in the end-game stage of a search as the amoeba shrinks and lowers its standard deviations to look near its best point. *GARS* allows the *GAMBIT* search rapidly to cover portions of the search space with shallow slopes. *GARS* exhibits more of a tendency to avoid local "trapping" minima, by accelerating over holes, or jumping out of them, provided its standard deviations are set correctly.

While the *GAMBIT* search algorithm is powerful and can successfully avoid many local minima, it does so at the expense of numerous iterations. A typical cycle can require 1,000 to 10,000 or more iterations, where other search techniques may converge in fewer iterations. However for difficult search problems, where gradient information is not available and other techniques fail, the *GAMBIT* search algorithm can find better solutions when given sufficient computing time.

4. APPLICATION OF POLYNOMIAL NEURAL NETWORKS TO ACOUSTIC WARFARE SIGNAL PROCESSING

4.1 Introduction

In Section 2 of this report, a way of viewing neural networks in the context of general function estimation principles was presented. Additionally, classification tasks were viewed as a particular type of estimation problem that involved finding a function that mapped an input vector of features into a vector of conditional probabilities of class membership. In Section 3, some specific polynomial neural network (PNN) structures and learning algorithms were presented in light of the principles discussed in Section 2. Because, under the scope of this project, the authors were tasked with applying PNN techniques to the detection and *classification* of acoustic warfare (AcW) signals, Section 3 emphasized logistic-loss PNN classifiers.

One reason for discussing a number of popular neural network paradigms using the common terminology of Section 2 was to allow these paradigms to be readily compared. Section 3.2 discussed a number of advantages of the minimum-logistic-loss PNN classification algorithm (*CLASS*); these advantages are summarized here:

- Whereas most neural network paradigms minimize a squared-error loss function (Eq. 2:23), resulting in a maximum-likelihood classification of data having a Gaussian probability distribution, *CLASS* minimizes the logistic loss function (Eq. 2:20), resulting in maximum-likelihood classification of data having a multinomial probability distribution, which is more suited to discrete multi-class problems.

- Whereas most neural network paradigms use gradient information only during the optimization process, *CLASS* uses a regularized nonlinear Gauss-Newton optimization algorithm (ILS) that can also use curvature information for more effective training iterations. This algorithm provides rapid on-line and off-line network training.

- Whereas most neural network paradigms use arbitrarily complex pre-structured networks, *CLASS* can build its structure to provide nonlinear classifiers having a degree of complexity (i.e., classification power) commensurate with the quantity and representativeness of the training database.

- Whereas most neural network classifiers output confidence measures for a particular class, *CLASS* outputs estimates of the *a posteriori* probabilities of class membership. These are particularly useful when these outputs are used by higher-level decision-making processes.

- Whereas many pre-structured neural networks require thousands of coefficients and interconnections, *CLASS* uses simple network structures that do not overfit the training data; these structures are particularly well-suited for rapid and efficient on-line interrogation.

Section 3.2 also discussed some limitations of the *CLASS* algorithm as currently implemented; however, it is important to note that these limitations arise primarily from the pre-structured nature of the networks created by the current algorithm and are shared by all pre-structured neural network paradigms. Limitations of the present *CLASS* algorithm include: (1) full interconnectivity that may lead to overly complex networks and (2) the restriction that all nodal elements have identical structure. Additionally, *CLASS* is currently limited to one hidden layer of nodal elements and the potentially sub-optimal arbitrary assignment of a *baseline* class against which all other classes are compared. Section 3.3 outlined algorithmic enhancements that can potentially overcome these limitations by building a just-sufficiently complex *CLASS* network structure. However, even without these algorithmic enhancements, the advantages of *CLASS* outweigh its limitations; especially considering that these limitations are shared by most alternative paradigms.[†]

This section focuses on the specific design of an AcW classification system that makes use of the polynomial neural network (PNN) technology discussed in previous sections. The material presented below is organized topically, with a major subsection devoted to each stage of the classification process. Classification results are presented along the way where they help illustrate the utility of the technique being discussed. BAI worked primarily with two datasets: (1) *Dataset B*[*] and (2) the *Rangex* data.[**] For the most part, *Dataset B* contained a greater quantity of cleaner signals resulting in better classification performance; all of the 20-class confusion matrices presented were taken from this dataset. The *Rangex* data, on the other hand, proved more challenging to classify for the following reasons: (1) there were numerous low SNR exemplars, (2) the exemplars within a given class tended to be quite diverse, (3) many exemplars from differing classes tended to be quite

---

[†] The *Algorithm for Synthesis of Polynomial Neural Networks (ASPN)* is an alternative algorithm that does *not* have the limitations of a pre-structured network; however, comparisons between *ASPN* and *CLASS* have shown that *CLASS*, using the logistic loss function, achieves better multi-class classification performance than *ASPN* using the squared-error loss function.

[*] These data were provided by Orincon Corp. in the second quarter of 1990.

[**] These data were provided by Orincon Corp. in December 1990.

similar,† and (4) the supply of quality exemplars for a number of the shorter (and more difficult) classes was very limited.

Appendices A and B give a complete listing of these datasets and the use made of them in this project. Section 5.2 contains a more detailed analysis of some of the shorter classes found in the *Rangex* dataset.

4.2 AcW Classification Signal Processing Overview

Whereas the research conducted by Barron Associates, Inc. has focused on data classification, the classification task is only one element of an entire acoustic warfare signal classification system, and the synthesis of a discrimination function is not isolated from the overall context of the signal processing flow. Fig. 4.1 shows more of the steps required for proper classification of unknown input signals.

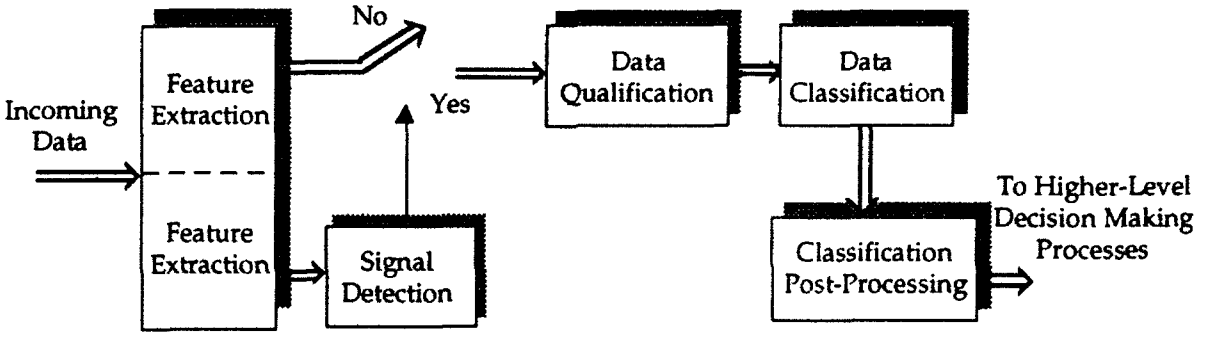


Figure 4.1: Processing Chain for a General Classification System

4.2.1 Scope of Research Conducted by Barron Associates, Inc.

Because the synthesis of improved discrimination (classification) functions is not isolated from the other aspects of AcW signal processing, BAI investigated *data qualification* and *post-processing* as steps that could potentially improve the classification performance. Additionally, because the information-theoretic neural network approach used by BAI stresses the generation of networks with limited complexity commensurate with the amount and diversity of the training data, research was conducted in the area of reduced-dimension *feature extraction*, since one method employed to limit the network complexity is to limit the size of the input feature vector. *Signal detection*, as discussed below, was considered to be

---

† For example, both BAI's and Orincon's networks had difficulty discriminating between Classes 3 and 4 (DW and SD); time- and frequency-domain analysis also showed numerous exemplars from these two classes that looked and sounded alike. Therefore, when possible, these two similar classes were combined by the authors into one *superclass* to achieve improved performance.

generally outside the scope of the present research effort; however, a type of detector was used in the *data qualification* stage to determine if the incoming feature vector fell inside the region for which the classification networks had been trained.

4.2.2 Detection

Detection is the initial step of separating signals of interest from background noise. This step may also include pre-processing operations, such as use of a whitening filter to decorrelate the signals of interest from background noise, interference, and sensor characteristics. Accurate performance of the detection task can greatly enhance subsequent signal classification tasks by freeing these algorithms from spurious detections.

The signals of interest in AcW signal processing are generated by complex, possibly nonlinear, vibrations. Such signals may exhibit non-Gaussian statistical behavior, as evidenced experimentally by the successes of kurtotic features (fourth-order statistical moments) in the classification process (see Section 4.4.7). For AcW signatures with random characteristics, deterministic signal analyses based on matched-filtering (for the coherent case) or banks of matched filters (for the incoherent case) generally do not apply. In many cases, these signals can be detected adequately using conventional radiometric techniques, which do not exploit the distributional characteristics of the signals. Radiometry assumes that there is no structure to either the AcW signal or the noise (i.e., both are assumed white) and that both the signal and the noise have Gaussian amplitude distributions. It involves comparing the received normalized energy to a threshold, which may be set and adapted as a function of operating conditions. The nominal threshold can be learned on line, for example, by continually reducing it to a level where many false detections are observed (internally only). Then, the actual threshold used is set to some function of the recurring false-detection threshold. Radiometry, however, may be sub-optimal for this application, and it may be possible to improve significantly upon it by exploiting the more detailed statistical behavior of the signals of interest [33]. For example, the use of higher-order spectral information can potentially result in significant improvements in the early detection capabilities of subtle AcW transients. More sophisticated detectors or Gabor and wavelet representations may also be useful in detecting transient AcW signals when the signal is non-white [26]. Such improvements may be particularly crucial for the early detection of subtle but important AcW signals.

Along with the detection of signals that have previously been characterized, an important task is the detection of signals that have not been seen previously by the detection system. This problem is essentially one of hypothesis testing, in which the null hypothesis consists of those statistical models, arising both from target and non-target signals, that *have* been previously characterized. If sufficient evidence arises to reject this hypothesis, it can reasonably be assumed that a new signal class has arisen, since the range of normal conditions can be characterized *a priori*. To test such an hypothesis, it is necessary to operate on a set of observables that is

sufficiently rich to capture the behavior of the possible signals of interest. Rejection of the entire class can then be based on a chi-square or related statistic.

Since a suitable set of observables will already be available from the detection and feature extraction stages of the system, the *implementation* of algorithms for detecting novelty should involve relatively modest additional complexity. The *development* of suitable algorithms for the purpose of novelty detection is, however, a separate task. Initially, these algorithms are constrained to operate on the observables generated by other stages of the system. Within this constraint, the efficacy of traditional statistical tests, such as the aforementioned chi-square test, should be examined. If such tests do not perform sufficiently well, then tests that use more detailed information, such as Kolmogorov-Smirnov-type tests [56], should be considered. If these more sophisticated tests do not suffice, then additional features should be customized for novelty detection by examining subspaces orthogonal to the other features extracted by the system.

4.2.3 Feature Extraction

Feature extraction involves preliminary processing of sonar time-series data to obtain suitable parameters that, in linear and/or nonlinear combination, allow the subsequent neural networks to discriminate between various classes of signal. Some relevant feature extraction techniques include:

- Spectral analysis
- Time-frequency and scale analysis (e.g., Wigner distribution, wavelets, etc.)
- Recursive estimation methods (linear and nonlinear predictive coding, etc.)
- Dynamic neural network prediction approaches
- Prony's method
- Moment analysis
- Auto/cross bi-coherence (bi-spectrum), tri-spectrum, etc.
- Higher moments (e.g., kurtosis)
- Cumulants
- Generalized hypercoherence
- Hypercoherence filtering
- Phase-domain averaging
- Nonstationary analysis
- Extrema signal processing
- Heuristic features

All of the above techniques, and others, have potential merit for acoustic waveform signal feature extraction. Parameters extracted from these representations can be used as input vectors to detection and classification neural networks. Spectral and recursive techniques make the assumption that the signal is not white but that the noise is. Nonlinear and higher-order techniques (e.g.,

cumulants, polyspectra, etc.) are especially valuable when the acoustic signal and/or noise are non-Gaussian. In such cases techniques based on second-order statistics (e.g., coherence, spectral analysis, etc.) and linear (e.g., wavelet) and bilinear (e.g., Wigner) transformations are suboptimal. All of the above approaches may be considered in a three-dimensional context (e.g., amplitude vs. frequency vs. time) and (where practicable) combined with spatial selectivity derived from implicit or explicit multi-channel sensor beamforming, to provide distinct and representative spatio-temporal patterns. In this regard, classification of acoustic signals is somewhat analogous to the "cocktail party problem" in human audition, whereby people attending a large gathering are able to isolate a single conversation occurring at a particular spatial location from all other spatially-distributed conversations. This is achieved, in part, by taking advantage of auditory spatial processing capabilities derived from the use of multiple sensing channels (i.e., one for each ear); different perception by each ear of the phase and amplitude of stimuli enables humans to determine the angle of arrival of sound sources. (Although the ear pinnae are used to distinguish sounds arriving in front from sounds arriving from behind, three or more vibration sensors will provide a more precise isolation capability.) This is a form of adaptive beamforming.

Features considered for use with the logistic-loss PNN classifier are discussed in Section 4.3.

4.2.4 Data Qualification

Data qualification is a pre-classification stage to determine if an acoustic signature is one for which prior information is available and for which classification should be attempted. Pre-classification is best done using closed decision boundaries in feature space and may be implemented using hyperellipsoidal clustering techniques or suitable polynomial neural networks to form hyperspheres (normalized hyperellipsoids) in feature space, each of which encloses a hypervolume corresponding to a different AcW class or portion of a class. All space outside of the hyperspheres then corresponds to *background* or *unknown* cases. Alternatively, the features derived from background sounds may form one or more hyperspheres, in which case exemplars falling outside of these hyperspheres represent signals of interest.

For pre-classification purposes, the closed decision boundaries defined by the hyperellipoids represent a significant improvement over hyperplane class separators. This is because an extreme value in one or more of the feature vector components will cause classifiers based on hyperplanes to see the vector in one of the boundary classes, even if the vector is displaced far from any feature vector on which the classifier was trained. The advantage of using PNNs to form closed decision boundaries is that no assumptions regarding the Gaussianity of the signal or noise are necessary, which is *not* true for Bayesian (i.e., Gaussian feature) classifiers. If the acoustic signals *are* Gaussian, the PNN closed-boundary decision functions will automatically adapt to the Gaussian decision properties. Any

boundary selected by the PNN can be checked analytically to verify that it forms a closed region.

4.2.5 Data Classification

Data classification infers the specific classes of detected acoustic signals. As mentioned above, classification neural networks should be trained using a constrained logistic-loss fitting criterion, rather than a least-squares fitting criterion, as the former are demonstrably superior. Multi-class networks based on the logistic-loss criterion attempt to separate classes by maximizing the probabilities of correct class membership for all observations in the database. Thus, the outputs of these networks are estimates of the true probabilities of class memberships, rather than estimates of arbitrary class output numbers.

4.2.6 Classification Post-Processing

Classification post-processing represents any additional operations that are performed subsequent to any of the steps outlined above. Subsequent to both the detection and classification stages, for example, post-processing operations may be used to reduce false-alarm rates via multiple-look strategies. Here the post-processor accumulates a number of single-look decisions before actually making final detection/classification decisions. Additionally, a passive sonar system may incorporate current and historical target track information to assist in its post-processing. Syntactical decision rules may also be used, based on the computed probabilities and knowledge of the AcW context.

4.3 Database Design

Databases used for neural network syntheses must take into account (1) the conditions under which the system must operate, (2) the means of observing the physical process, and (3) the nature of the physical system itself. Each of these should have a strong bearing on database design.

The conditions under which the system must eventually operate should determine the conditions for which data are obtained and the quantity of data. The properties of the data space should determine how the data in the database are distributed. Consider a physical system that is definable in terms of the equivalent of F descriptors or features. The *space* of these F features has the following properties:

$$N_c = 2^F \qquad 4:1$$

$$N_e = F2^{F-1} \qquad 4:2$$

$$N_b = 2F \qquad 4:3$$

where N_c, N_e, and N_b, are the numbers of corners, edges, and spatial boundaries, respectively.

The corners (vertices) are the points at which all independent variables have limiting values, and are of particular concern. When F is large, most of the volume of the space is crowded near the corners. For the neural network to provide good discrimination, in most applications, it must perform acceptably in these corner regions of the input feature space as well as in the interior. Thus, when F is large, considerably more samples are needed remote from the center of the space, near the edges and particularly near the corners, than are required near the center of the space. To achieve model quality with a nonlinear network at the corners of the data space comparable to the quality at the center, each corner region ideally should be represented by 2^F *times* the density of samples used to represent the interior of the data space. In practice, this may not be practicable and the user should be cautioned that the neural network is less reliable in the corner (and other fringe) regions than in the middle of the space.

For AcW classification, a number of measures may be taken to ensure adequate performance of the neural networks in spaces involving many features:

(1) Use as many data vectors in the corner regions as are reasonably possible.

(2) Keep the dimension, F, of the feature vector as low as possible.

(3) Fit the networks to an artificially enlarged data space, so that the fitting quality is improved within the space of actual variation.

(4) Use nonlinear terms in the networks sparingly.

(5) Carefully test the networks in the corner regions before judging their performance to be acceptable.

(6) When interrogating the networks operationally, verify that each unknown data vector is within the region of accurate representation by the networks.

In sizing the database, allowance should also be made for reserving a significant fraction for design evaluation, which must be performed on an independent (and statistically representative) subset of the data.

Because, in this work, emphasis was placed on testing and evaluating classification methods on real-world data, and because the collection and truthing of acoustic data are costly, the number of exemplars in the database was often extremely limited. When limited training data are available, great care must be taken to achieve a quality network model capable of adequate generalization that does not overfit the training database; a specific example and detailed analysis of these tradeoffs is found in Section 5.2.

4.4 Feature Generation

Barron Associates, Inc. (BAI) considered a number of different features for use with the minimum-logistic-loss PNN classification algorithm, *CLASS*. The primary focus in all of the feature research was to reduce the dimensionality of the input vector to the neural network for the reasons cited above, and all feature research was performed on the data provided to BAI by Orincon Corporation.

4.4.1 PNN Predictors as Feature Generators

One method for generating input features for data classification is to use a bank of polynomial neural networks to predict the future values of the time-series data input; this type of feature generation is illustrated in Fig. 4.2.

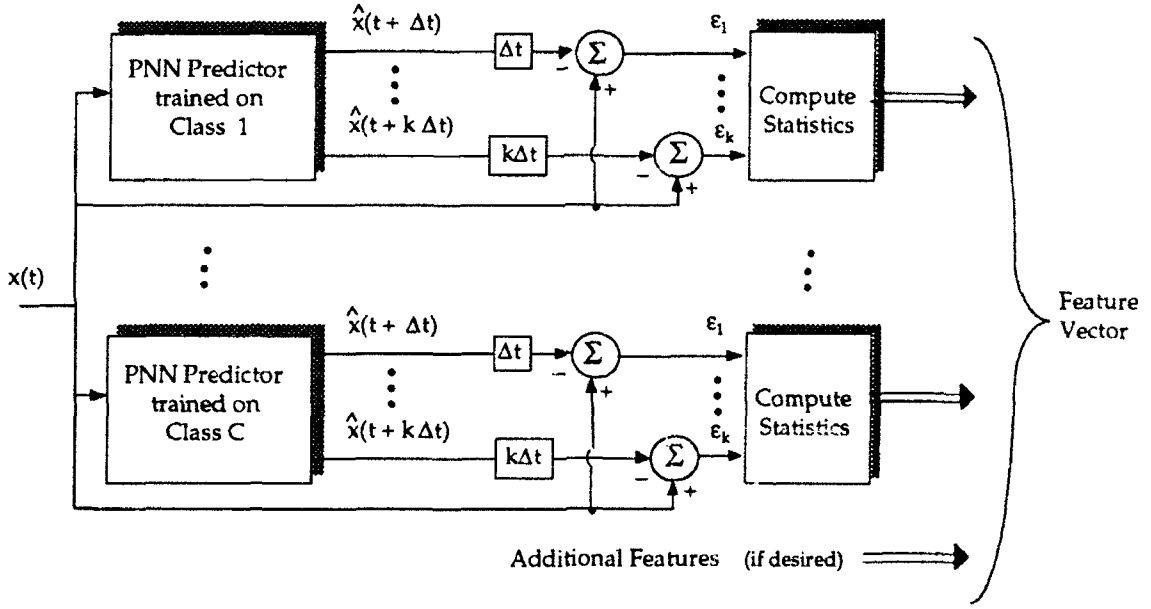


Figure 4.2: PNN Predictors as Feature Generators

To use PNN predictors as feature generators, multiple PNN predictors (PNPs) are trained to predict future values of the incoming data stream, x(t), using data from a different class of signal for each PNP. If the incoming data represents class k, then predictor k should provide the best prediction. In this sense, the PNPs could themselves serve as a form of classifier; however, better performance can be obtained by using the statistics of the prediction errors as features in a further classification process.

Either static estimation networks (*ASPN*: Section 3.4) or dynamic estimation networks (*DynNet*: Section 3.5) may be used as predictors. Figs. 4.3 and 4.4 show static and dynamic PNPs, respectively.

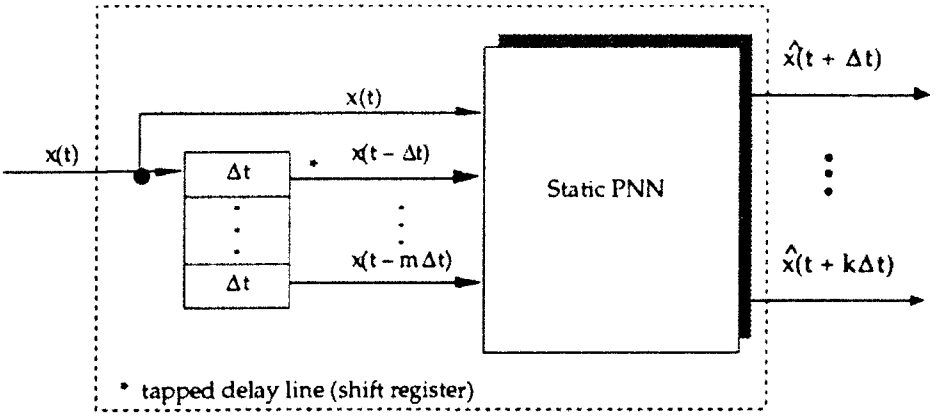


Figure 4.3: Static Polynomial Neural Network Predictor (PNP)

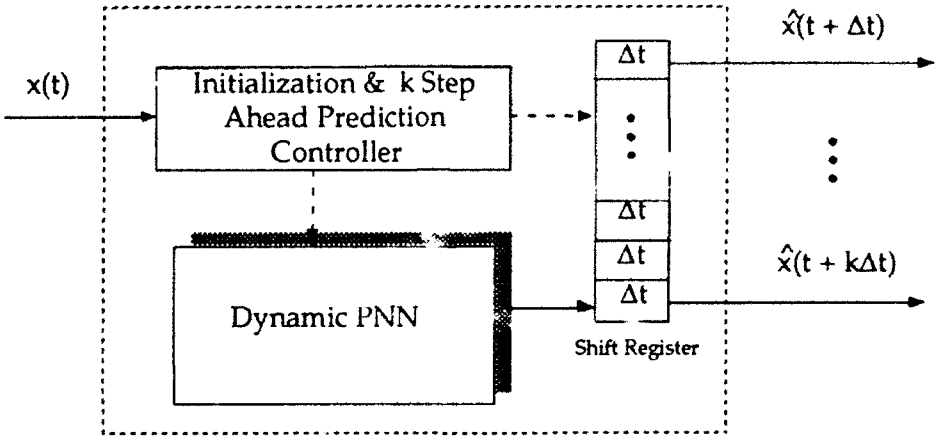


Figure 4.4: Dynamic Polynomial Neural Network Predictor (PNP)

In Fig. 4.3, the shift registers outside the network are used to provide current and historical values to the network as inputs. Then, a multi-input multi-output (MIMO) static transformation maps these inputs into expected future values of the time series, x(t). In Fig. 4.4, the shift registers are contained inside the dynamic PNN (see Section 3.5 and Fig. 2.4). These shift registers are initialized, and the network is allowed to "run ahead" for k samples to predict future values of the time series. The output shift register serves to convert the DPNN series output into a vector output as required by the system shown in Fig. 4.2.

In work conducted on the AcW data, static PNPs were used to predict one sample into the future, and the statistical computation of Fig. 4.2 consisted of computing a time average of the absolute errors and, for one class, the squared-errors (see Section 4.4.5); no additional features were used. These statistics were

then passed on to *ASPN* for classification.† The four-class network created by *ASPN* is shown in Fig. 4.5 (element definitions are given in Section 3.4.1, Eqs. 3:47 - 3:53).

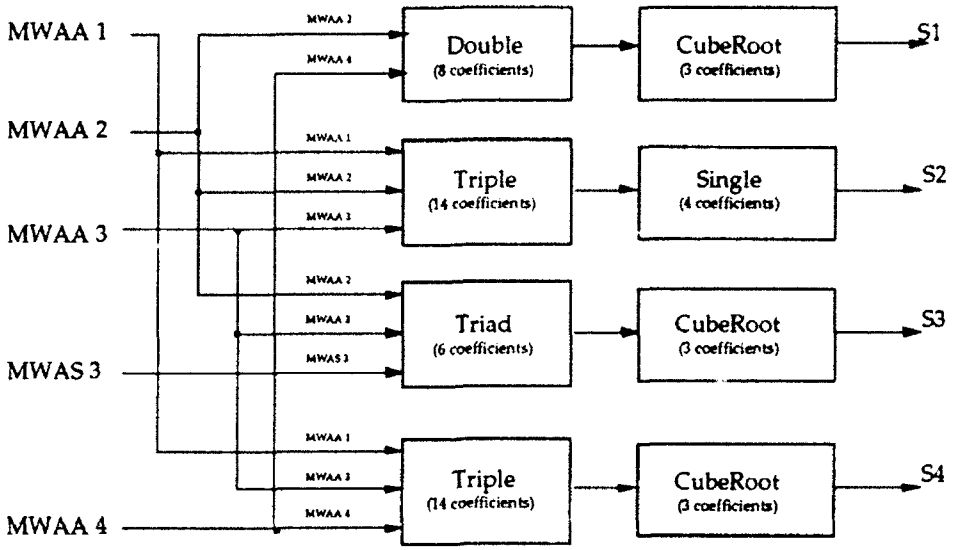


Figure 4.5: ***ASPN* Classification Network for PNP Features**

In Fig. 4.5, the input values (MWs) represent moving-window statistics of the prediction error as described above and defined in Section 4.4.5. The network output, $\underline{s}$, contains an estimate of class membership for each of the four classes. The results of the network shown in Fig. 4.5 on the limited amount of data initially available are shown in Table 4.1 [18].

Table 4.1: **PNP Classification Results (Four-Classes)‡**

True Class	System Decision				
	Class 1	Class 2	Class 3	Class 4	Unknown
Class 1	4/4	0	0	0	0
Class 2	0	1/1	0	0	0
Class 3	0	0	4	0	0
Class 4	0	0	0	4/4	0
Unknown 1	0	0	0	0	1/1
Unknown 2	1/1	0	0	0	0

† At this stage in the work effort, the *CLASS* algorithm had not been completed.

‡ An unclassified key to class numbers can be found in Appendix A.

To demonstrate the utility of PNPs on a larger and more difficult problem, BAI later extracted exemplars for 20 data classes [1]. These classes were then manually grouped into four families (see Section 4.5.1.2) as follows:

Table 4.2: Signal Family Groupings

Group	*Signals*
Family 1	1, 2, 3, 4, 11, 16
Family 2	5, 6, 8, 10, 13, 15
Family 3	12, 17, 18, 20
Family 4	7, 9, 14, 19

Tables 4.3 - 4.6 present the results of PNP predictors when used in conjunction with a *CLASS* network on this dataset.

Table 4.3: PNP Classification Results (Family One)

True Class	System Decision					
	Class 1	Class 2	Class 3	Class 4	Class 11	Class 16
Class 1	0.92	0.01	0	0	0.05	0.03
Class 2	0.01	0.93	0.05	0	0	0.01
Class 3	0	0.04	0.93	0	0	0.03
Class 4	0	0	0	1.0	0	0
Class 11	0.02	0	0	0.01	0.98	0
Class 16	0.02	0.01	0.03	0	0	0.92

Table 4.4: PNP Classification Results (Family Two)

True Class	System Decision					
	Class 5	Class 6	Class 8	Class 10	Class 13	Class 15
Class 5	0.87	0.02	0	0	0.11	0
Class 6	0.01	0.98	0	0	0.02	0
Class 8	0	0	1.00	0	0	0
Class 10	0	0	0	1.00	0	0
Class 13	0.06	0.01	0	0	0.94	0
Class 15	0	0	0	0	0	1.00

Table 4.5: PNP Classification Results (Family Three)

True Class	System Decision			
	Class 12	Class 17	Class 18	Class 20
Class 12	1.00	0	0	0
Class 17	0	0.79	0.11	0.10
Class 18	0	0.36	0.45	0.19
Class 20	0	0.21	0.17	0.62

Table 4.6: PNP Classification Results (Family Four)

True Class	System Decision			
	Class 7	Class 9	Class 14	Class 19
Class 7	1.00	0	0	0
Class 9	0	1.00	0	0
Class 14	0	0	1.00	0
Class 19	0	0	0.02	0.98

An alternative way of using PNPs as feature generators is to adapt a single predictor on-line, and use the coefficients of the predictor as features for the classifier; this approach is shown in Fig. 4.6:

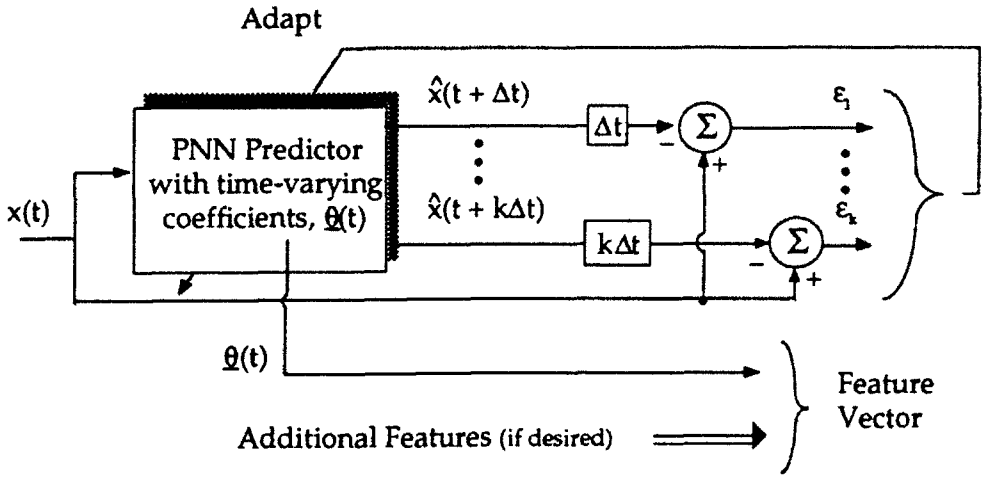


Figure 4.6: PNP Coefficients as Classification Features

Use of PNP coefficients as features is closely related to linear and nonlinear predictive coding. Modern spectral analysis techniques have shown that these coefficients contain information concerning the power spectrum of the input time series [47].

One method of improving the performance of the PNP technique described above is to add a bank of passband filters prior to the prediction process. Fig. 4.7 illustrates the use of passband filters and additional data scaling in conjunction with a one-step ahead PNP.

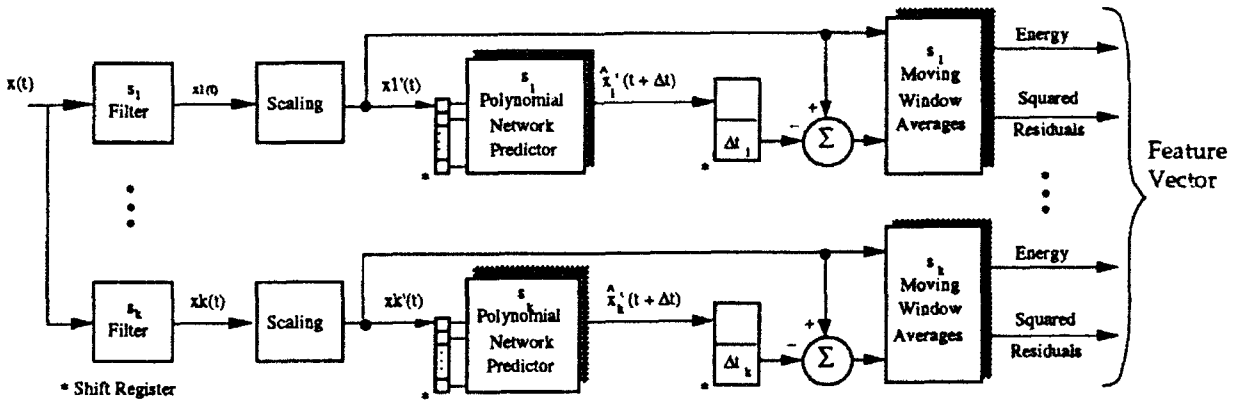


Figure 4.7: Time-Domain Pre-Processing for PNP Feature Generation

The scaling in Fig. 4.7 ensures that the input to the network is in the range ±1. The passband filters in the figure provide both noise suppression and discrimination capabilities for each class. Additionally, for a given class, i, no energy

outside the frequency of interest is put into the classification system. The passband filter may be represented mathematically in the frequency domain as

$$\gamma_i(f) = \frac{\varphi_i(f)}{\varphi_i(f) + \frac{1}{C-1}\sum_{\substack{n=1 \\ n \neq i}}^{C} \varphi_n(f)} \qquad 4:4$$

where $\varphi_i(f)$ is the spectral magnitude of the signal corresponding to class i at frequency f, $\gamma_i(f)$ is the gain of the filter at frequency f, and C is the number of classes considered by the classifier. Sample filters for a three-class classifier are shown in Fig. 4.8.

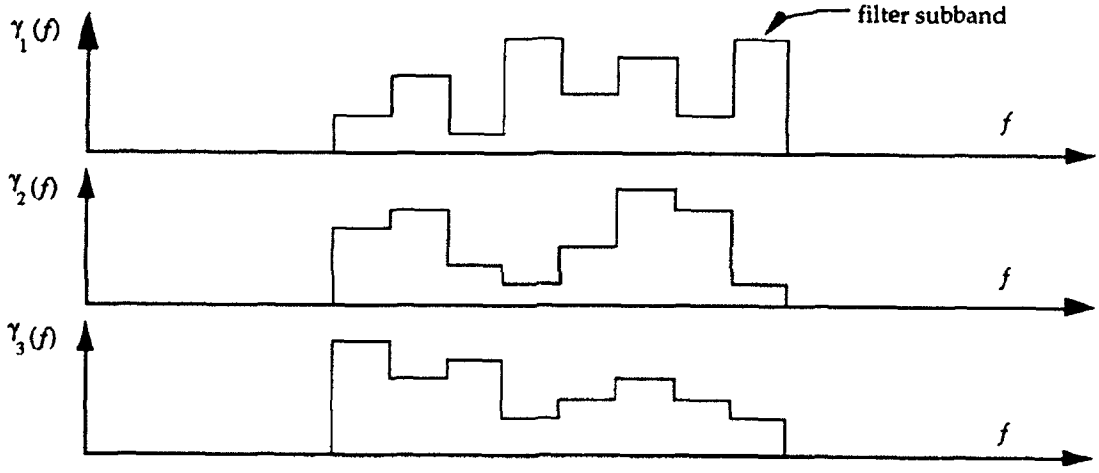


Figure 4.8: Sample Discrimination Filters for a Three-Class Network

4.4.2 Additional Time-Domain Features

Higher-order statistics are an additional type of time-domain information that can be helpful for AcW classification if the signals of interest are non-Gaussian. Two statistical measures investigated were the *skewness* or third moment, and the *kurtosis* or fourth moment. Skewness, which characterizes the degree of asymmetry of a distribution around its mean, is defined as

$$\text{Skew}(\underline{x}) = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{x_j - \bar{x}}{\sigma}\right)^3 \qquad 4:5$$

where $\bar{x}$ is the mean of the input data vector, and σ is the standard deviation. Kurtosis, which measures the relative peakedness or flatness of a distribution, is defined as

$$\text{Kurt}(\underline{x}) = \left[\frac{1}{N} \sum_{j=1}^{N} \left(\frac{x_j - \bar{x}}{\sigma} \right)^4 \right] - 3 \qquad 4:6$$

Experiments with the *Rangex* data set showed that the Kurtosis of the time-series data may be particularly useful in distinguishing man-made signals from biological waveforms. In fact, on the particularly difficult Orincon "short-net" database [52], the authors were able to achieve perfect discrimination between biological and man-made signals on both training and evaluation data sets using a minimum-logistic-loss PNN classifier and kurtosis as the sole network input.

4.4.3 Frequency Domain Features

In addition to the time-domain features presented above, various means were investigated for reducing the dimensionality of frequency-domain features for logistic-loss PNN classification. Fig. 4.90 shows a typical lofargram display.

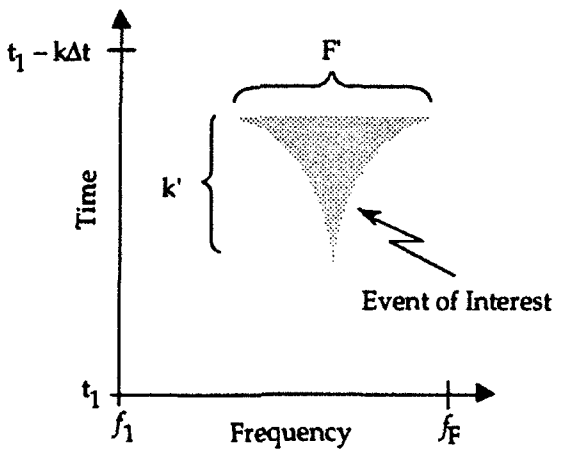


Figure 4.9: A Typical Lofargram Display

The F × k lofargram will often contain hundreds of frequency bins and historical time scans, resulting in $O(10^5)$ pixels in the overall display. This would present an unreasonably large number of inputs to a neural network, especially given the limited amount of training data available (Section 4.3).

As Fig. 4.9 shows, for many types of signals it may not be necessary for the network to see the entire lofargram. If the events of interest to a particular network have an expected frequency range, F', and an expected duration, k', then the number

of pixels required to display the event (shaded region) can be reduced. However, the new F' × k' *retina* may still contain an unacceptably large number of inputs for a neural network trained on a limited number of signals. One popular method of reducing the dimensionality of the retina is to average the bins, scans, or both. While this technique may be able cut the number of inputs in half, it does not significantly affect the order-of-magnitude of the retinal area.

Two alternative methods to reduce the dimensionality of the frequency-domain features were used: (1) heuristic features derived from the FFT information, and (2) principal component analysis of the input features. The latter technique is discussed in Section 4.4.6 since it is not limited to frequency-domain features.

In a first attempt to reduce the dimensionality of the input feature vector, the following twenty-one heuristic features were chosen based on their ability to characterize the information in a single scan† of *Rangex* data. Previous experience with similar features had also yielded favorable results.

1. Total spectral energy between 100Hz and 1000Hz (E_1).
2. Total spectral energy between 1000Hz and 3000Hz (E_2).
3. Total spectral energy between 3000Hz and 5000Hz (E_3).
4. The ratio $\frac{E_1}{1-E_1}$.
5. The ratio $\frac{E_1+E_2}{1-E_1-E_2}$.
6. The ratio $\frac{E_2}{1-E_2}$.
7. The frequency at which the largest spectral peak occurs (F_1).
8. The frequency at which the second largest peak occurs (F_2).
9. The frequency at which the third largest peak occurs (F_3).
10. The magnitude of the largest spectral peak (M_1).
11. The magnitude of the second largest spectral peak (M_2).
12. The ratio $\frac{M_1}{M_2}$. *(a measure of coherence)*

---

† A single scan constituted one row of the lofargram display shown in Fig. 4.9. This represented a subset of the frequency bins resulting from a 1024-point FFT. Adjacent scans overlap each other by 50% (i.e., the FFT window slides by 512 points in the time-domain between each scan).

13. The ratio $\frac{M_1}{M_3}$. *(Where M_3 is the magnitude of the third largest peak)*
14. Frequency of 10% cumulative power (f_{10}). *(i.e.,10% energy below f_{10})*
15. Frequency of 25% cumulative power (f_{25}).
16. Frequency of 45% cumulative power (f_{45}).
17. Frequency of 70% cumulative power (f_{70}).
18. The ratio $\frac{0.10 - 0.00}{f_{10} - f_0}$. *(Where f_0 is 0.0.)*
19. The ratio $\frac{0.25 - 0.10}{f_{25} - f_{10}}$.
20. The ratio $\frac{0.45 - 0.25}{f_{45} - f_{25}}$.
21. The ratio $\frac{0.70 - 0.45}{f_{70} - f_{45}}$.

Before the FFT is taken, the time-domain data are preprocessed as shown in Fig. 4.10.

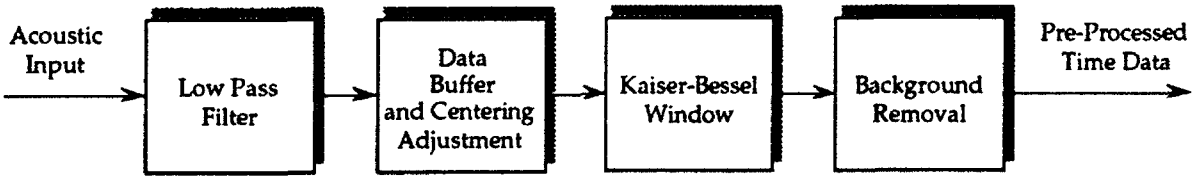


Figure 4.10: Time-Domain Preprocessing for FFT Feature Generation

In Fig. 4.10, the data buffer and centering adjustment is described in Section 4.4.4, and the Kaiser-Bessel (KB) window is defined by:

$$w(n) = \frac{I_0\left[\pi\alpha\sqrt{1.0 - \left(\frac{n}{N/2}\right)^2}\right]}{I_0[\pi\alpha]}, \qquad 0 \le |n| \le \frac{N}{2} \qquad 4:7$$

where

$$I_0(x) = \sum_{k=0}^{\infty}\left[\frac{(x/2)^k}{k!}\right]^2 \qquad 4:8$$

These single-scan features were tested on five classes from the *Rangex* data set. The classes were divided into four *families* using hyperellipsoidal clustering (see Section 4.5.1), and a classifier was generated for each family. The classification accuracies obtained on independent evaluation data are presented in Tables 4.7 - 4.10 [3].

Table 4.7: FFT-Based Classification Results (Family One)

True Class	System Decision		
	Class 6	Class 11	Other
Class 6	6/6	0	0
Class 11	0	3/3	0
Other	0	0	68/68

Table 4.8: FFT-Based Classification Results (Family Two)

True Class	System Decision	
	Class 3 or 4	Other
Class 3 or 4	45/45	0
Other	2/34	32/34

Table 4.9: FFT-Based Classification Results (Family Three)

True Class	System Decision	
	Class 8	Other
Class 8	6/6	0
Other	0	111/111

Table 4.10: FFT-Based Classification Results (Family Four)

True Class	System Decision	
	Class 2	Other
Class 2	15/19	4/19
Other	1/35	34/35

All results presented in Tables 4.7 - 4.10 included the use of hyperellipsoidal cluster detectors, and did *not* include any multi-look post-processing (Section 4.6).

4.4.4 Automatic Window Centering

The location of an AcW transient within a window of data can affect the nature of the power spectrum. This is especially true if the transient is brief as compared to the window size and the classifier is using a single-scan feature vector. Fig. 4.11 shows a technique that can be used to ensure that the AcW transient is automatically centered in a given FFT window.

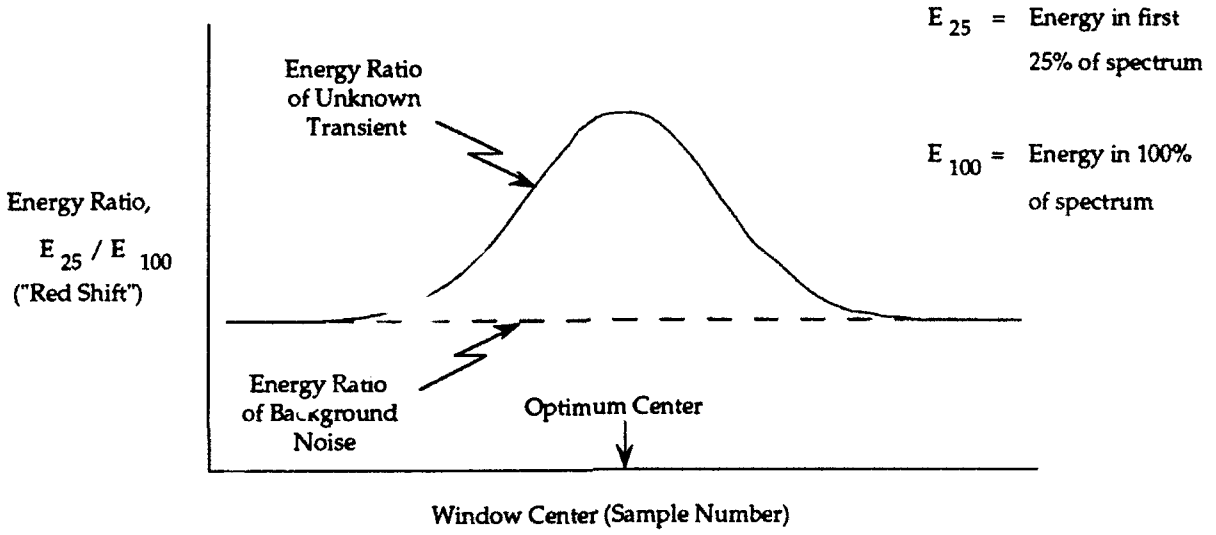


Figure 4.11: Self-Centered Spectral Windowing via *Red Shift*

As the sampled-data window approaches its optimum center position relative to the buffer-stored transient acoustic energy, the spectral power distribution undergoes its maximum *red shift* to low frequencies. The red shift is defined as follows:

$$S_R = \frac{E_p}{E_{100}} \qquad 4:9$$

where E_p is the energy in the first p percent of the power spectrum bins.

4.4.5 Moving Window Feature Calculations

The frequency-domain features described above were all calculated for a single FFT scan. Often a single scan is all that is needed to classify an AcW signal correctly; especially if that scan is derived from a window that is consistently

centered on the onset of the signal.† However, it is often desirable to give the network access to the temporal characteristics of the transient that correspond to multiple scans (see Fig. 4.9).

The most common method of incorporating temporal information is to give the network access to historical feature vectors. However, once again, doing so tends to result in an overly large number of network inputs given the size of the training data. Just as heuristics can be used to reduce the dimensionality of the frequency (spatial) axis of the lofargram shown in Fig. 4.9, similar heuristics can be used to reduce the dimensionality of the time (temporal) axis. These are known as moving-window (MW) features, and some commonly used MW features are given below. Note that in Eqs. 4:10 - 4:17, the vector, $\underline{x}$, here represents current and historical values of a *single* feature rather than current values of multiple features.

Average:

$$\mathrm{MWA}(\underline{x}, N) = \frac{1}{N-1}\sum_{i=0}^{N} x(t-i) \qquad 4:10$$

Standard Deviation:

$$\mathrm{MWSD}(\underline{x}, N) = \sqrt{\frac{1}{N-2}\sum_{i=0}^{N} \left[x(t-i) - \bar{x} \right]^2} \qquad 4:11$$

Average Energy:

$$\mathrm{MWAE}(\underline{x}, N) = \frac{1}{N-1}\sum_{i=0}^{N} \left[x(t-i) \right]^2 \qquad 4:12$$

Average Absolute:

$$\mathrm{MWAA}(\underline{x}, N) = \frac{1}{N-1}\sum_{i=0}^{N} \left| x(t-i) \right| \qquad 4:13$$

Historical Value:

$$\mathrm{MWV}(\underline{x}, N) = x(t-N) \qquad 4:14$$

---

† For instance, the ShortNet test results (Section 5.2), obtained using a single FFT scan for each event, compare favorably with other short-net networks that made use of multiple scans.

Span:

$$MWS(\underline{x}, N) = | x(t) - x(t-N) | \qquad 4:15$$

Average Derivative:

$$MWD(\underline{x}, N) = \frac{1}{N-1} \sum_{i=1}^{N} [x(t) - x(t-i)] \qquad 4:16$$

Point of P_E Power:

$$MWP(\underline{x}, N, P_E) = j; \qquad 4:17a$$

where

$$P_E = 100 * \sum_{i=0}^{j} [x(t-i)]^2 \Big/ \sum_{i=0}^{N-1} [x(t-i)]^2 \qquad 4:17b$$

Notice that these moving-window functions calculate first and second moments for a window of temporal data (Eqs. 4:10 and 4:11). The temporal data may be raw time-domain data, as in the case of the PNN prediction errors described in Section 4.4.1 and shown in Figs. 4.2 and 4.9. Or, the data may be current and historical values of the frequency-domain features described in Section 4.4.3. This type of spatio-temporal approach was used for the Build 2 sea trial and is discussed in Section 5.1.

4.4.6 Principal Component Analysis

Principal component analysis (PCA) is a non-heuristic technique for reducing the dimensionality of an input feature vector. The principal component transformation, also known as the Karhunen-Loève transformation, uses a standardized linear combination (singular value decomposition) of the input data to maximize the variance of the transformation output data. The idea behind maximizing the variance is to "maximally separate" the input feature vector, thereby easing consideration of differences between feature vectors. The coefficients that weight the old input features, in forming the linear combination, are scaled so that the sum of their squares is equal to one.

To transform an input feature vector, $\underline{x}$, into a new feature vector, $\underline{x}'$, the following principal component transformation is used:

$$\underline{x}' = \underline{\underline{A}}\,\underline{x} \qquad 4:18$$

where $\underline{\underline{A}}$ is the transformation matrix with rows that contain the eigenvectors of the sample data covariance matrix [46]. The result of the principal component transformation is that the new set of $\underline{x}'$ vectors will have linearly independent columns (features). Additionally, if the $\underline{\underline{A}}$ matrix is arranged so that the eigenvectors are sorted according to the magnitude of their corresponding eigenvalues, then the features of $\underline{x}'$ will be sorted according to their ability to account for variance in the original feature database.

A useful property of eigenvalue analysis is that the sum of any N eigenvalues divided by the sum of all the eigenvalues represents the proportion of the total variation explained by these N eigenvalues. The "total variation," which is the sum of all of the eigenvalues, is equal to the trace of the data covariance matrix. Values on the diagonal of the data covariance matrix can also be divided by the trace of the covariance matrix to compute the proportion of the total variation explained by the original variables. Fig. 4.12 illustrates this for a typical feature database containing 84 original features.

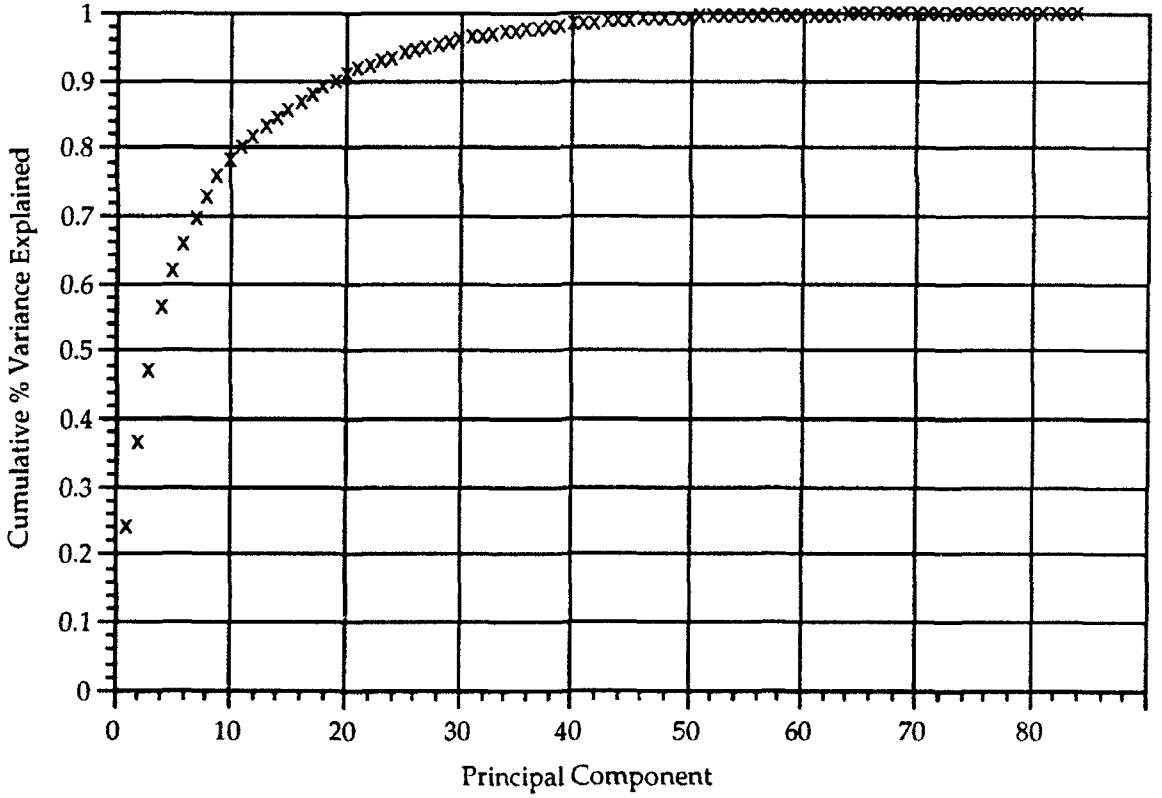


Figure 4.12: Cumulative Percent Variance Explained by Principal Components

As can be seen from the figure, almost 80% of the variance in the original database was explained by the first ten elements in the principal component vector, $\underline{x}'$. Thus, for this example, the size of the input feature vector could be reduced by

88% with only a 20% loss of information. Twenty principal components achieve a 76% reduction in the dimension of the feature vector with only a 10% information loss.

Clearly, for this example, the feature vector could be reduced in size by one-half with virtually no loss of information; however, the effects of further reductions in the size of the input feature vector on classification performance cannot be determined *a priori*.† Experience has shown that in many cases the size of the input vector can be reduced so that χ' accounts for 60% to 80% of the variance in the original data with little adverse effect on classification performance. Section 5.2 discusses the use of PCA on AcW data and the corresponding classification results obtained with the *CLASS* algorithm.

4.5 Data Qualification

Data qualification, or detection, address the question of whether an input vector of observables represents a class or classes from among C classes on which a classifier is trained. Classification is not performed on input vectors that fail the qualification criteria. The idea behind data qualification is to provide a means for the identification of signals (and their corresponding feature vectors), for which the classifier has not been trained; it is often desirable to classify these signals by other means (see Section 4.7) and then use them to retrain the AcW classifier so that in the future it can respond to this type of input.‡

4.5.1 Supervised Hyperellipsoidal Clustering (HEC)

4.5.1.1 *The HEC Methodology*

Physically, the process of representing data vectors as points in feature space can define hyperellipses, based on the statistics of the data features. Data qualification may be performed by computing normalized distances from hyperellipsoidal centroids and comparing each distance with a threshold. The interrogation process is shown in Fig. 4.13.

---

† Note that a structure-learning classification algorithm, such as that proposed in Section 3.3, is capable of automatically selecting suitable inputs from among the principal components, eliminating the need for analyst involvement in the reduction of the size of the input vector. One might then ask why one would use PCA with a structure-learning algorithm at all; PCA can improve the speed of network learning by eliminating candidate inputs and reducing the number of input combinations that need to be tried.

‡ The data qualification stage may also correctly be called a "detection" stage of the AcW classification process, in the sense that it "detects" whether or not the incoming signal is one for which the classifier has been trained. This type of "detector" should not be confused with the detector in Fig. 4.1. That detector is responsible for determining the presence of a signal of interest among the background noise and is not concerned with the region for which the classifier has been trained.

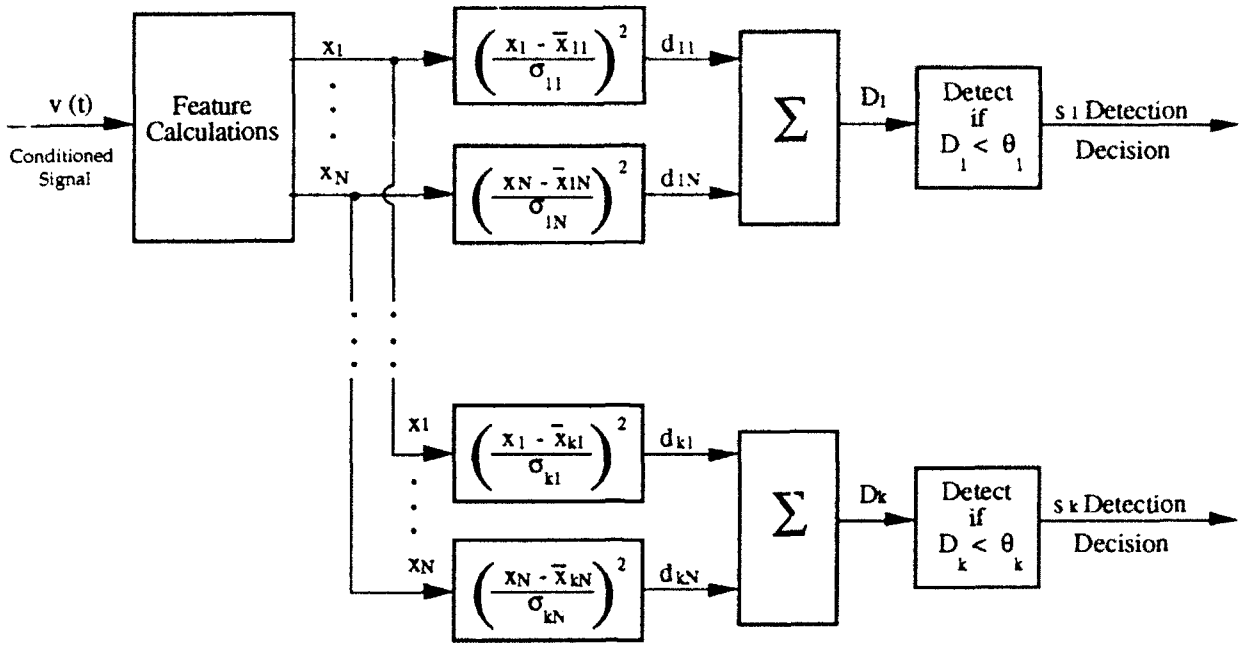


Figure 4.13: Data Qualification Using Ellipsoidal Cluster Tests

In Fig. 4.13:

$\bar{x}_{jk}$ ≡ the mean of feature j for class k

σ_{jk} ≡ the standard deviation of feature j for class k

i=1, . . . , n; n is the number of input data vectors

j=1, . . . , N; N is the number of features (dimensions) in the input vector

k=1 . . . , C; C is the number of known classes

θ_k ≡ the detection threshold for class k.

During the synthesis of the data qualification stage, N-dimensional ellipsoidal clusters (one dimension for each feature) are formed with closed boundaries; within each cluster it is likely that the input vectors for a given class will appear. These n-dimensional boundaries are learned *from the data* in the form of ellipsoids, nominally at some multiplicative constant, θ_k, times the N-dimensional input feature standard deviation vector. Closed decision boundaries are important because they provide greater robustness than linear separators; without closed decision boundaries, input vectors unlike anything seen during training might pass through the data qualification stage.

Note that for data qualification, the ellipsoids are chosen to define crude class boundaries. While it would be possible to define a single ellipsoid that enclosed the data seen by a network for *all* classes, it is preferable to cluster by classes.

Consider a two-class problem with data distributed as shown in Fig. 4.14. If qualification of the data in the figure is accomplished by computing the statistics for

all the data (classes 1 and 2) the decision boundary will enclose a large amount of space for which training data did not exist (the shaded region). If, on the other hand, a separate cluster is formed for each data class, the resulting decision boundaries will exclude these regions, resulting in improved (i.e., more discriminating) data qualification.

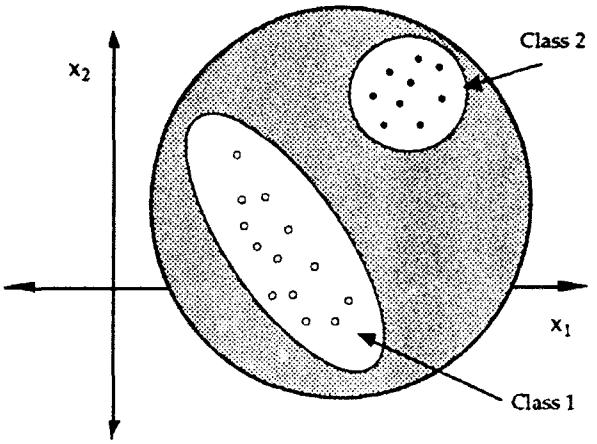


Figure 4.14: HEC For a Two-Class Problem

For the special case of two input features, the equation for the distance measure for a single class as computed in Fig. 4.13 is given by

$$D_1 = \frac{(x_1 - \bar{x}_{11})^2}{\sigma_{11}^2} + \frac{(x_2 - \bar{x}_{12})^2}{\sigma_{12}^2} \qquad 4:19$$

which may be expanded into the general polynomial form

$$D_1 = \hat{y} = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_2 + a_4 x_2^2 \qquad 4:20$$

where:

$$a_0 = (\bar{x}_{11} / \sigma_{11})^2 + (\bar{x}_{12} / \sigma_{12})^2$$

$$a_1 = -2\,\bar{x}_{11} / \sigma_{11}^2$$

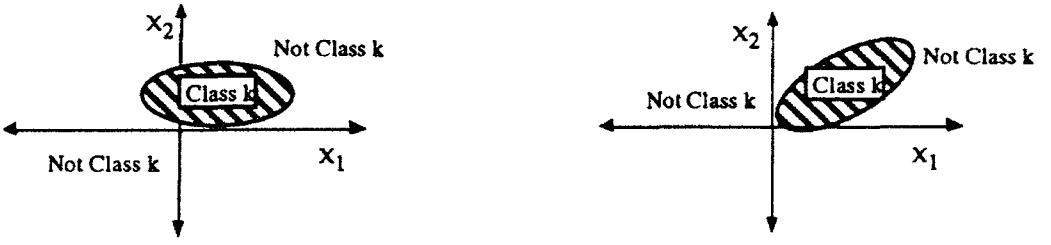
$$a_2 = 1 / \sigma_{11}^2 \qquad 4:21$$

$$a_3 = -2\,\bar{x}_{12} / \sigma_{12}^2$$

$$a_4 = 1 / \sigma_{12}^2$$

Each coefficient value is fixed based upon the mean, $\bar{x}_{ij}$, and the standard deviation, σ_{ij} (assuming a Gaussian distribution) of each of the features.

A graphical representation of the ellipsoidal clustering process using two features, x_1 and x_2, is shown in Fig. 4.15(a).



(a) Principal Axes Collinear with Features (b) Rotated Principal Axes

Figure 4.15: Graphical Representation of Ellipsoidal Clustering

Everything in the above figure that is not in the shaded region denoted "Class k" is, by definition, not in Class k. Note that the principle axes of the ellipse represented by Eq. 4:20 are collinear with the principal axes of the feature space, as in Fig. 4.15(a). The HEC transformation, Eq. 4:20, cannot represent a cluster that is rotated on its principal axes as in Fig 4.15(b). However, Eq. 4:20 may be modified to incorporate the *Mahalanobis* distance measure (Section 2.4.3, Eq. 2:126). This distance measure accounts for cross-correlations between the input features and is related to principal component analysis (Section 4.4.6). An alternative to the Mahalanobis distance is to use PNNs with nodal elements that contain cross terms to represent the clusters. In addition to being able to rotate the principal axes of the clusters, the general polynomial network approach has the advantage that *no* a priori *structure is imposed on the data other than the nodal element model, which may be expanded into any general form* (see Section 4.5.1.4).

In supervised hyperellipsoidal clustering the class membership of each item is determined analytically by the user. The cluster algorithm then determines the size and location of each cluster by computing the mean and standard deviation along each dimension of all points which belong to that cluster. The following statistics are accumulated for each cluster:

1. Standard deviation along each dimension of the database.

2. Mean along each dimension of the database.

During interrogation the normalized distance from a data point $\underline{X}_i = [x_{i1}, \ldots, x_{ij}, \ldots, x_{iN}]^T$ to the centroid of a cluster k is given by

$$D(\underline{X}_i) = \sqrt{\sum_{j=1}^{N_k}\left(\frac{x_{ij} - \bar{x}_{ij}}{\sigma_{ij}}\right)^2} \qquad 4:22$$

where N_k is the total dimensionality of cluster k. The point $\underline{X}_i$ can then be classified as being most similar to that cluster for which it has the smallest normalized distance. An inverse measure of relative likelihood for membership in cluster k is

$$e^{-D(\underline{X}_i)} \qquad 4:23$$

As the distance to the center of the cluster goes to zero, the likelihood of membership goes to unity. By computing Eq. 4:23 for k=1, . . . , C, one may establish the class of most likely membership of $\underline{X}_i$.†

Two or more of the input features in a cluster may be correlated, causing the cluster to become large in the involved dimensions, creating a large volume within the cluster that is not populated. A point from a different class, such as shown in Fig. 4.16(a), may then more readily appear within the cluster, leading to false classifications.

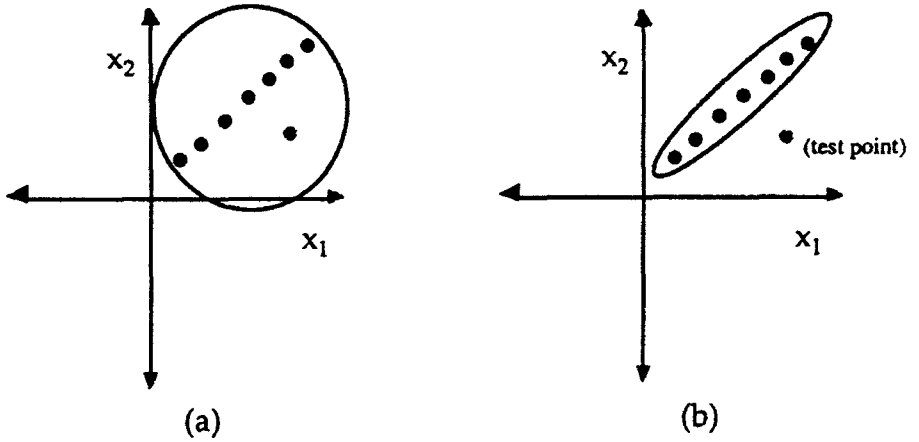


Figure 4.16: (a) A Cluster with Excessive Capture Area (volume); (b) A Preferable Cluster

Two modifications to the supervised clustering technique would address this concern:

1. Provide partially supervised clustering, in which the analyst determines the class membership for each vector in the database, but an

† Note the relationship between the cluster distance measure and the Radial Basis Function (RBF) networks discussed in Section 2.4.3.

unsupervised clustering algorithm operates separately on each class of data to determine sub-clusters within that class.

2. Transform the input variables of each cluster using singular value decomposition. This provides a linear transformation of the inputs by which the principal components become aligned with the cluster axes, resulting in clusters as depicted in Fig. 4.16(b).

4.5.1.2 *HECs as Pre-Classifiers and "Family" Detectors*

In addition to rejecting exemplars outside the training region of the classifiers, HEC data qualification can improve classification performance by allowing the use of multiple PNN classifiers. Often, in feature space, there exist class groupings that may be distinguished from each other using cluster techniques. The classes within each grouping, however, may not be distinguishable using cluster techniques. A natural approach, then, is to create *superclasses* or *families* of classes. A different PNN classifier may then be trained for each family, and pre-classification may be used to determine the family (or families) to which the incoming data belong.

One approach to creating families is to group signals by duration; this is the technique that is currently used by Orincon Corporation in the Build 2 and Build 3 systems (short-net and long-net) [52]. Another approach is to group the AcW classes by frequency range as shown in Fig. 4.17. This approach is especially useful if the classification features are already in the frequency domain.

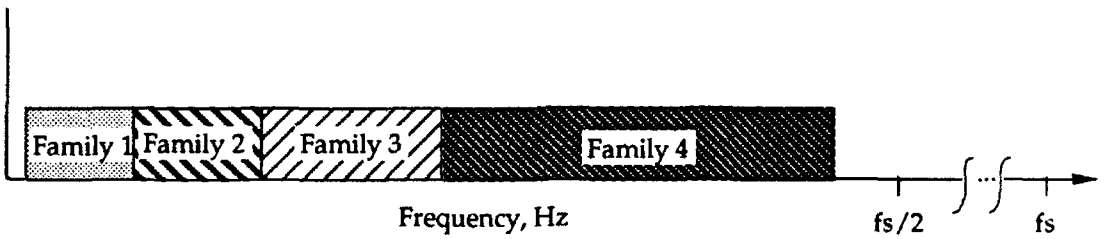


Figure 4.17: Grouping Classes by Frequency

The hyperellipsoidal clustering method was applied to the same 4-family, 20-class data used for the PNP experiments (Section 4.4.1, Tables 4.2 - 4.6). The cluster detection results on independent evaluation data are shown in Table 4.11 [52].

Improved results can be obtained using multi-look post-processing (Section 4.6, Table 4.20). Note that for HEC data qualification, the rows of the confusion matrix (Table 4.11) need not sum to unity because an exemplar may be a member of more than one data cluster (i.e., some clusters overlap).

Table 4.11: Single-Look HEC Pre-Classification Results

Class of Evaluation Signal \ Single-Look Detected Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.86	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	0	0	0	0	0
2	0	0.77	0.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.84	0	0	0	0	0	0	0	0.40	0	0	0	0	0	0	0	0	0
4	0	0	0.80	0.80	0	0	0	0	0	0	1.00	0.11	0	0	0	0	0	0	0	0
5	0	0	0	0	0.86	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0
6	0	0	0	0	0.03	0.80	0	0	0	0	0	0	0	0	0	0	0	0	0.14	0
7	0	0	0	0	0	0	0.90	0	0.04	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.84	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.90	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.90	0	0	0	0	0	0	0	0	0	0
11	0.29	0	0	0	0	0	0	0	0	0	0.86	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.83	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0.86	0	0.86	0	0	0	0	0	0.43	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.74	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0.50	0	0.86	0	0.93	0	0	0	0.29	0
16	0	0	0	0	0	0	0	0	0	0	0.76	0	0	0	0	0.74	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.74	0.14	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0.04	0	0	0	0.2	0.60	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0.60

In general, supervised clustering should not be used as a stand-alone tool for classification because of the inherent assumption that all points in the same class will belong to a *unimodal* Gaussian distribution. However, as can be seen from Table 4.11, for this particular dataset the hyperellipsoidal clusters were adequate class discriminators for pre-qualification.

4.5.1.3 *HECs as Feature Generators*

In addition to data pre-qualification, hyperellipsoidal clustering may also be used to generate features for classification. The most useful features are based on the cluster distance measures, D_k, defined in Eq. 4:22.

The most straightforward way to use the distance measures as features is to tap off the distance measures prior to the thresholding in Fig. 4.13 and send them directly to the classifiers as features. To evaluate the effectiveness of HEC distance measures, *CLASS* networks were trained using only cluster distances as input features. The single-look results on independent evaluation data containing ten known and three unknown classes are shown in Tables 4.12 - 4.15 [2].

Table 4.12: Single-Look HEC-Distance Classification (Family One)

True Class	System Decision						
	Class 1	Class 2	Class 3	Class 4	Class 11	Class 16	Unknown
Class 1	1.00	0	0	0	0	0	0
Class 2	0	1.00	0	0	0	0	0
Class 3	0	0.50	0.13	0	0.38	0	0
Class 11	0	0	0	0	1.00	0	0
Unknowns 1&2	0	0	0	0	0.05	0	0.95
Unknown 4	0	0	0	0	0	0	1.00

Table 4.13: Single-Look HEC-Distance Classification (Family Two)

True Class	System Decision						
	Class 5	Class 6	Class 8	Class 10	Class 13	Class 15	Unknown
Class 5	1.00	0	0	0	0	0	0
Class 6	0	1.00	0	0	0	0	0
Class 10	0	0	0	0	1.00	0	0
Class 13	0	0	0	0	0.67	0	0.33
Unknowns 1&2	0.07	0.01	0	0.08	0	0	0.83
Unknown 4	0	0	0.01	0	0	0	0.99

Table 4.14: Single-Look HEC-Distance Classification (Family Three)

True Class	System Decision				
	Class 12	Class 17	Class 18	Class 20	Unknown
Unknowns 1&2	0.04	0	0	0	0.96
Unknown 4	0.46	0	0	0	0.54

Table 4.15: Single-Look HEC-Distance Classification (Family Four)

True Class	System Decision				
	Class 12	Class 17	Class 18	Class 20	Unknown
Class 7	1.00	0	0	0	0
Class 9	0	1.00	0	0	0
Unknowns 1&2	0.01	0.17	0	0	0.82
Unknown 4	0	0	0	0.35	0.65

Once again, the results presented in Tables 4.12 - 4.15 can be improved upon by using a multi-look post-processing technique (Section 4.6, Tables 4.16 - 1.19).

The fact that HEC distance measures provide useful features is not surprising given the fact that there is a close relationship between their use as PNN inputs and Radial Basis Function (RBF) neural networks (see Section 4.5.1.5).

4.5.1.4 *HECs as Implemented by PNNs*

Suppose the polynomials in the C–1 branches of the minimum-logistic-loss classifier are linear functions of the form

$$\hat{y}_k = a_{0,k} + \sum_{j=1}^{N} a_{j,k}\, x_j \qquad 4:24$$

If we first examine a two-input, two-class minimum-logistic-loss network using a *linear* node, we see that the network performs a linear separation between the two classes: the linear node defines the discrimination boundary between classes (Fig. 4.18). This can be extended to more than two inputs and more than two classes. When more than two classes are involved, the minimum-logistic-loss criterion creates an optimum family of discrimination lines (or hyperplanes) dictated by the distributions of the synthesis data populations for the various classes. This family is found in a simultaneous search because interactions arise between classes when locating multiple discriminant functions.

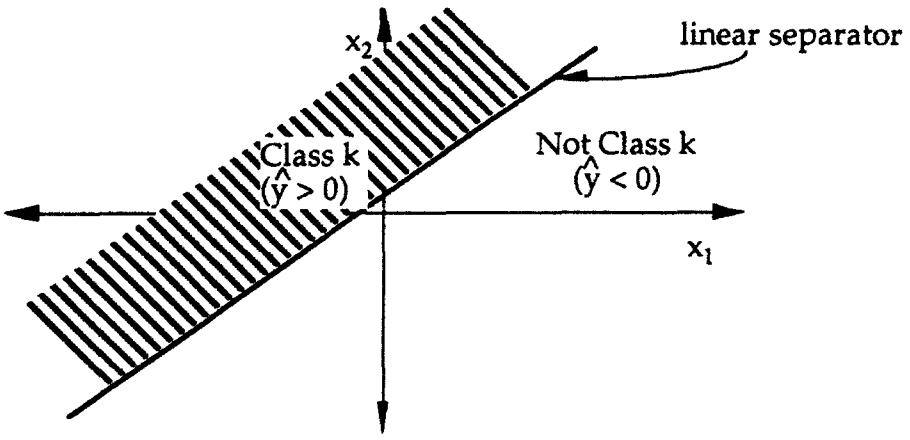


Figure 4.18: Classifier Using Linear Polynomials

It should be noted that a multi-layer perceptron (MLP) nodal element is limited to linear separation boundaries. The polynomial nodal element, however, with its incorporation of nonlinear terms, does not have this limitation.

Consider, next, the two-input second-degree additive polynomial as the branch polynomial form

$$\hat{y} = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_2 + a_4 x_2^2 \qquad 4:25$$

Assuming only two classes are to be discriminated, the minimum-logistic-loss classifier finds the best (at least locally) quadratic separator. Note that Eq. 4:25 has the same form as the hyperellipsoidal cluster equation (Eq. 4:20). While the polynomial form is the same, the coefficients of the quadratic polynomial, in general, will be different from those found using ellipsoidal clustering; they will be the same only if the statistics of the features are Gaussian. The advantage, therefore, of using the minimum-logistic-loss network is that the coefficients are more general and reflect what is found in the data. The network can only do better (at least on the training data) than ellipsoidal clustering techniques. In fact, given suitable values of the coefficients, Eq. 4:25 can describe a point, a line, a circle, an ellipse, a hyperbola, or a pair of intersecting lines.

Further, one may implement even more general forms of ellipsoidal clustering with minimum-logistic-loss networks. If, instead of using the additive quadratic polynomial in Eq. 4:25, a complete quadratic polynomial is used, for the case with two input features and two class outputs, the describing equation is:

$$\hat{y} = a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_2 + a_4 x_2^2 + a_5 x_1 x_2 \qquad 4:26$$

Note that the only difference mathematically is the inclusion of a cross-product term. Given this form, any conic-section form of separation can be performed *and* the separation can be rotated with respect to the coordinate system defined by the

features, as shown in Fig. 4.15(b). The advantage here is that the minimum-logistic-loss network will automatically perform a linear rotation of the coordinates if it provides better separation than the principle axes of the features. With ellipsoidal clustering, this would have to be performed in a separate step before clustering, e.g., by using the Karhunen-Loève transform or singular value decomposition.

In summary, minimum-logistic-loss networks subsume the capabilities of ellipsoidal clustering techniques and should provide superior performance in detection as well as isolation. The minimum-logistic-loss network is a completely general way of reflecting the natural distribution of the data, without imposing an assumed structure on the data. However, when the features have Gaussian distributions and the network used to fit the data is a quadratic additive polynomial, the minimum-logistic-loss classifier boundaries are essentially the same as those obtained with multivariate-Gaussian hyperellipsoidal clustering.

4.5.1.5 *HECs, Radial Basis Functions, and Unsupervised Clustering*

Hyperellipsoidal clustering techniques, especially when used as feature generators (Section 4.5.1.3), are closely related to radial basis function (RBF) neural networks. Recall, from Section 2.4.3, that RBFs contain one hidden and one output layer. The most commonly chosen hidden layer is the Gaussian kernel function

$$z = \exp\left(-\frac{(\underline{x}-\underline{w})^T(\underline{x}-\underline{w})}{2\sigma^2} \right) \qquad 4:27$$

and the output layer is a linear combination of the input layer nodal outputs.

$$y = \underline{\theta}^T \underline{z} \qquad 4:28$$

Rewriting the exponent in Eq. 4:27 as

$$\sum_{j=1}^{D} \frac{(x_j - w_j)^2}{2\sigma^2} \qquad 4:29$$

results in an equation of nearly the same form as the HEC distance measure, D_k, shown in Fig. 4.13, and Eq. 4.19, the difference being that in Eq. 4:29 the standard deviation, σ, is constant in each dimension. This constant σ is what makes RBFs "radial." If instead of a constant parameter, σ, a vector of standard deviations is used, one for each dimension of the input database, the *radial* basis function neural network becomes an *elliptical* basis function neural network (EBF), and Eq. 4:29 will take exactly the same form as the HEC distance measures. Fig. 4.19a shows the structure of an EBF network.

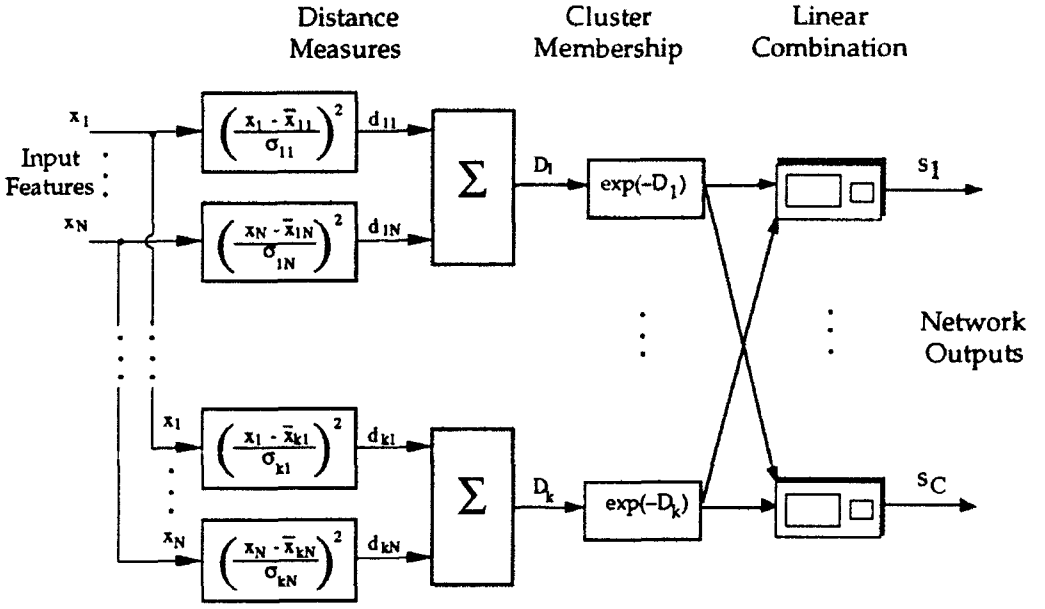


Figure 4.19a: RBF/EBF Classification Network Structure

Note that the *CLASS* network can readily implement the transformation shown in Eq. 4:28 as well as more complex nonlinear combinations of the hidden layer outputs. Fig. 4.19b shows the structure of a *CLASS* network with HEC distance measures as input features.

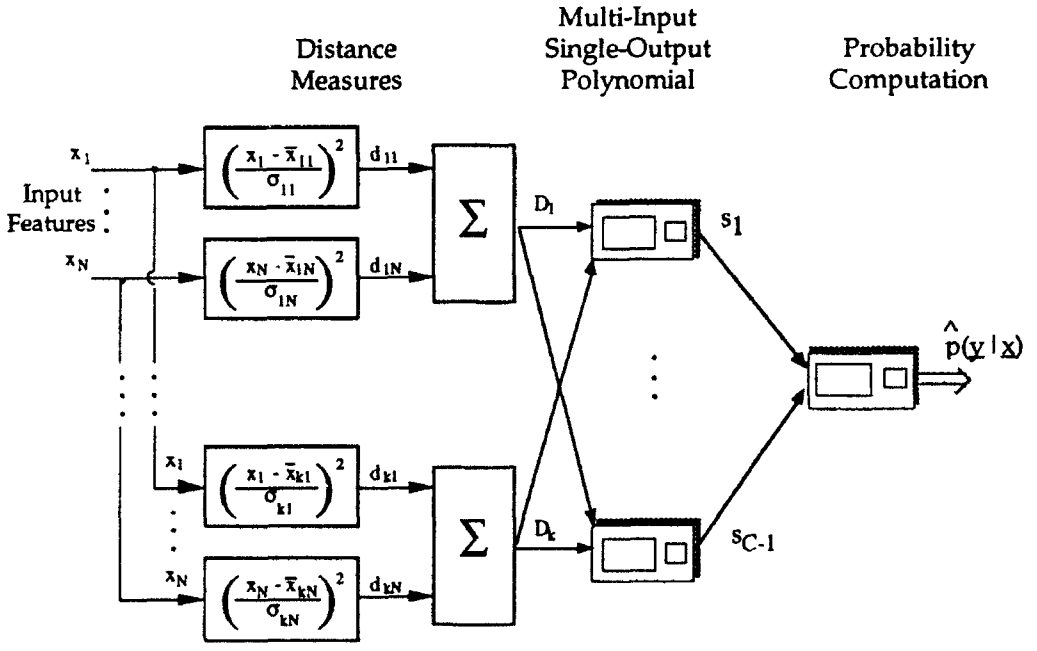


Figure 4.19b: HEC PNN Classification Network Structure

Figs. 4.19a and 4.19b illustrate the structural differences between RBF/EBF networks and PNN classification networks that use HEC distance measures as features. One difference is the presence of the membership calculation (Eq. 4:27) in the RBF/EBF network. This function serves to convert distance measures into a probability of membership in a Gaussian cluster and could have been used as part of the feature calculations for the PNN network of Fig. 4.19b (Eq. 4.23); however, this transformation is not essential because it does not add additional information. While HEC PNN classifiers can be used to implement RBF/EBF networks, the reverse is not the case. *CLASS* networks have two *structural* advantages over RBF/EBF networks: (1) they can combine the distance measures nonlinearly, and (2) they output estimates of actual *a posteriori* probabilities as a result of the logistic-loss training method.

In addition to the structural advantages mentioned above, *CLASS* networks have one significant *algorithmic* advantage over RBF/EBF networks: they optimize the layer of MISO polynomial nodal elements using a logistic loss function as opposed to a squared-error loss function.

Another significant algorithmic difference is the method in which the cluster statistics are determined. Supervised HEC assigns one cluster to each data class and computes the cluster coefficients by finding the distributions of the class observations. The RBF/EBF approach, however, typically uses an *unsupervised* technique such as *K-means* [22], to determine an analyst-specified number of natural data groupings (clusters). The advantages of the supervised approach over the unsupervised approach is an issue of ongoing research. In the current work, all cluster coefficients were determined using the supervised technique. However, unsupervised clustering could be valuable in finding the natural distributions of data points in multidimensional spaces with minimal reliance on *a priori* assumptions. One alternative to the *K-means* technique is BAI's *F-Cluster* program for unsupervised data clustering. The strength of *F-Cluster* is that it generates all possible valid clusters of the data set and evaluates them using multidimensional F-tests (significance tests) to identify the partitions that best describe the true geometric subjects of the data.

F-Cluster produces its final answer by generating a provisional cluster starting from each point in the observation database. As each of these provisional clusters is generated, it is compared to clusters that were found earlier to see if the new cluster is essentially the same as something the program already knows about. When this happens, the new provisional cluster is merged with the composite cluster. This allows a parsimonious description of the data space while still ensuring that the boundaries of each of the final clusters have been well explored. One benefit of this approach is that the analyst does not need to provide, *a priori*, the number of natural data groupings.

4.5.1.6 *HECs as Probability Density Function Estimators (Bayes Approach)*

While HECs are used primarily in a data qualification context, they may also be used indirectly as part of the classification process. As mentioned above, ellipsoidal clustering essentially involves computing the statistics of a Gaussian class distribution. Given these statistics, it is possible to generate a Bayes' classifier using decision features generated by polynomial neural networks and multimodal probability density functions generated by HECs. Fig. 4.20 illustrates this approach for a five-class problem.

BAI considered the implementation of the Bayes' classifier approach outside the scope of the current work effort.

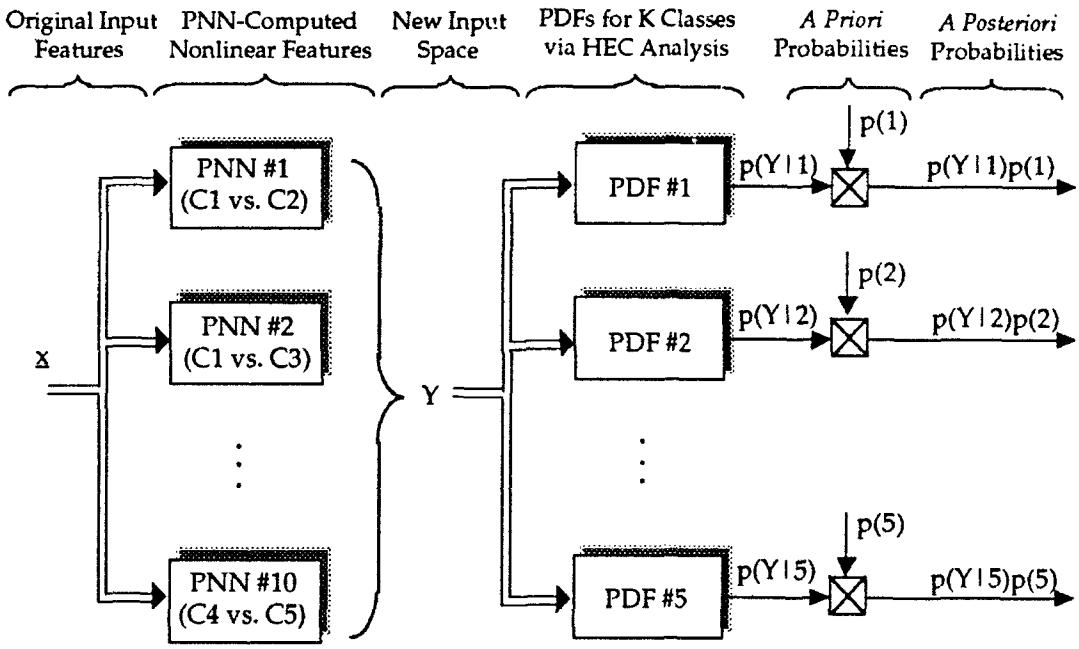


Figure 4.20: Five-Class Bayes' Classifier Using PNNs and HECs

4.5.2 Coherent Signal Processing

In addition to hyperellipsoidal clustering, a study was made of the use of *coherent* signal detectors as data qualifiers to remove and classify steady-state narrow-band classes (e.g., an RMS pulse). This method is shown in Fig. 4.21.

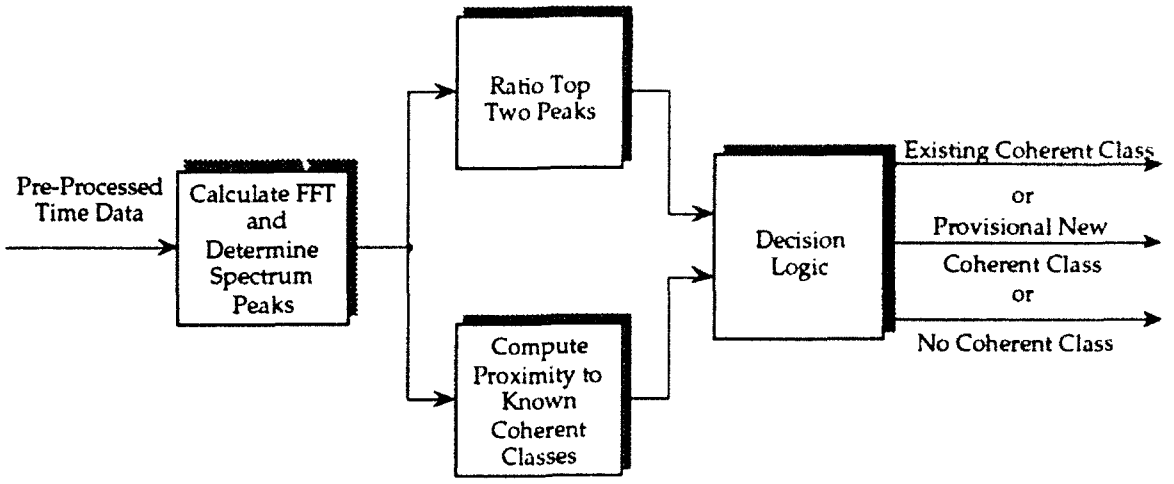


Figure 4.21: Data Qualification for Narrow-Band Steady-State (Coherent) Signals

The following steps were used in the discrimination of coherent signals:

(1) Locate the highest and second-highest spectral peaks in the signal, and label them P_1 and P_2 respectively.

(2) Determine the frequency, f_k, of P_1.

(3) If P_1/P_2 is smaller than a pre-defined threshold, the signal is deemed not coherent and is submitted to PNN classification.

(4) If the signal is coherent, the signal is classified by comparing the frequency f_k with a database of known coherent signals.

This method obviously can be extended to multiple-tone coherent signals.

4.6 Classification Post-Processing

Classification post-processing consists of a set of decision rules that map current and historical classification output probabilities into a class decision. The simplest decision rule, *maximum select*, involves selecting the class corresponding to the largest classifier output probability. The decision rule may include a requirement that the maximum probability be above a class-specific threshold. If decisions are based upon one interrogation interval only, they are called *single-look* decisions. Single-look decision rules provide good benchmarks against which other schemes can be measured.

Classification performance can be improved with *multi-look* classification post-processing. Figs. 4.22 and 4.23 show two multi-look post-processing schemes:

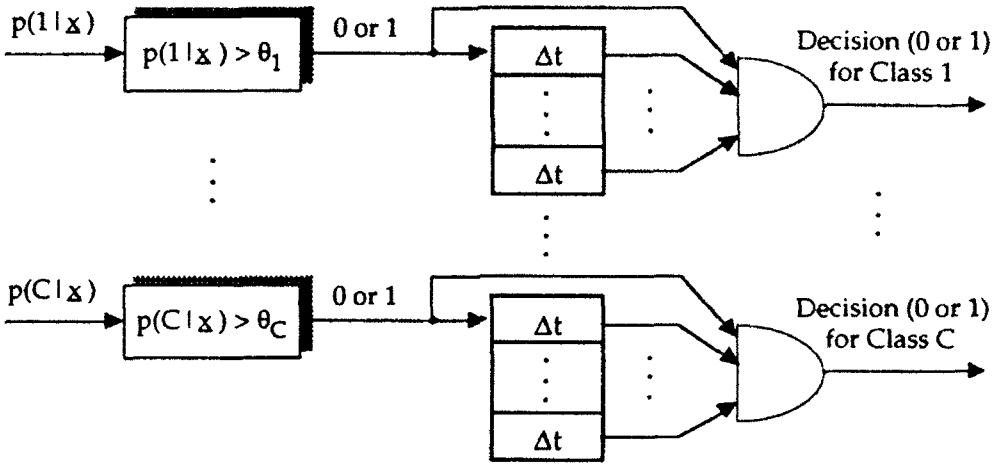


Figure 4.22: Multi-Look Post-Processing (Decision Accumulation)

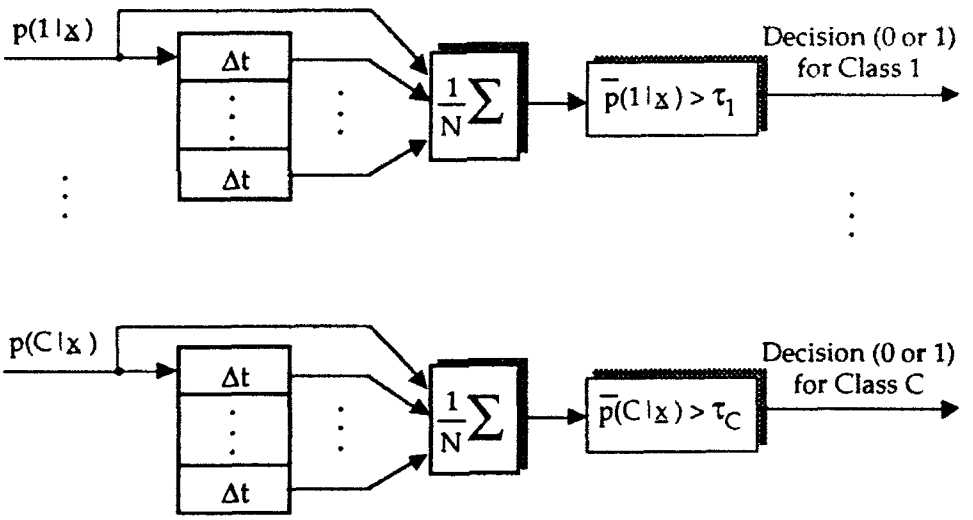


Figure 4.23: Multi-Look Post-Processing (Probability Averaging)

In Fig. 4.22, the decision logic declares class k if the some of the last N single-look classification probabilities exceeds some threshold, θ_k. In Fig. 4.23, the decision logic declares class k if the average of the last N probabilities exceeds some threshold, τ_k. Note that there is a unique threshold corresponding to each class; additionally, the shift registers for each class may be different lengths. There are a number of variations on these two post-processing methods: for instance, in the method of Fig. 4.22, instead of requiring all N single-look probabilities for a class to exceed a threshold (shown by the *and* gate in Fig. 4.23), one could relax the requirement so that only K of N single-look values are required for classification.

While the choice of a specific multi-look post-processing method and its corresponding parameters must be determined experimentally, the authors' experience has shown that even minimal, and perhaps sub-optimal, post-processing improves classification performance and should be used. To illustrate, the multi-look post-processing technique shown in Fig. 4.22 was used with the data from Section 4.5.1.3 (Tables 4.12 - 4.15) to obtain the improved results shown below in Tables 4.16 - 4.19:

Table 4.16: Multi-Look HEC-Distance Classification (Family One)

True Class	System Decision						
	Class 1	Class 2	Class 3	Class 11	Class 11	Class 16	Unknown
Class 1	1.00	0	0	0	0	0	0
Class 2	0	1.00	0	0	0	0	0
Class 3	0	1.00	0	0	0	0	0
Class 11	0	0	0	0	1.00	0	0
Unknowns 1&2	0	0	0	0	0	0	1.00
Unknown 4	0	0	0	0	0	0	1.00

Table 4.17: Multi-Look HEC-Distance Classification (Family Two)

True Class	System Decision						
	Class 5	Class 6	Class 8	Class 10	Class 13	Class 15	Unknown
Class 5	1.00	0	0	0	0	0	0
Class 6	0	1.00	0	0	0	0	0
Class 10	0	0	0	1.00	0	0	0
Class 13	0	0	0	0	1.00	0	0
Unknowns 1&2	0	0	0	0	0	0	1.00
Unknown 4	0	0	0	0	0	0	1.00

Table 4.18: Multi-Look HEC-Distance Classification (Family Three)

True Class	System Decision				
	Class 12	Class 17	Class 18	Class 20	Unknown
Unknowns 1&2	0	0	0	0	1.00
Unknown 4	0	0	0	0	1.00

Table 4.19: Multi-Look HEC-Distance Classification (Family Four)

True Class	System Decision				
	Class 7	Class 9	Class 14	Class 19	Unknown
Class 7	1.00	0	0	0	0
Class 9	0	100	0	0	0
Unknowns 1&2	0	0	0	0	1.00
Unknown 4	0	0	0	0	1.00

Table 4.20: Multi-Look HEC Pre-Classification Results

Class of Evaluation Signal	Multi-Look Detected Class																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1.00	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1.00	0	0	0	0	0	0	0.33	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0.67	0	1.00	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0.50	0	0.50	0	1.00	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	1.00	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00

4.7 On-Line Updating

The success of any neural-network-based application depends not only on the neural network paradigm employed but also the quality of the data used to optimize the network coefficients and/or structure. However, time and cost constraints often place limits on the effort that can be spent preparing a large and representative network database or fine-tuning network structures. Additionally, it is highly unlikely that all signal classes will be anticipated in the initial design phase, and subsequent off-line data collection and network training can prove costly.

To account for unforeseen circumstances, it is desirable to create an AcW classification system that can be easily and rapidly retrained *at sea*. Such a system presents two challenges: (1) the traditionally labor-intensive process of manually collecting and truthing new training data, and (2) the computationally intensive process of neural network training. The key to overcoming the first challenge, data preparation, is found in the use of unsupervised clustering techniques to preprocess raw data and assist the analyst in assigning a "truth" class to large numbers of exemplars simultaneously. Such automation could potentially result in a drastic reduction in the amount of analyst time required to prepare and update training databases. The key to overcoming the second challenge, network training, is found in the use of logistic-loss PNN classification networks. As discussed in Section 3.2.4 and demonstrated at Orincon Corporation,[†] the *CLASS* algorithm is capable of very rapid optimization of AcW classification networks. A proposed system capable of rapid on-line updating is shown in Fig. 4.24.

In Fig. 4.24, the training database is continuously updated with signals of interest (i.e., signals that passed through the detector in Fig. 4.1) that either fail to pass the data qualification stage or fail to be classified with a high degree of confidence (additionally, the analyst or operator may choose to include any signals that he knows have been misclassified). While one could also update the training database with exemplars that are classified correctly with a high degree of confidence, this, in general, is not necessary since a high confidence decision indicates that the classification algorithm is already performing properly in the relevant region of feature space. If, on the other hand, the classifier cannot classify an incoming event with a high degree of confidence, the classifier should be retrained so that, in the future, similar incoming exemplars will be classified correctly.

[†] Rapid On-Line retraining was demonstrated at Orincon Corp. at the following QPRs: July 1991, October 1991, and October 1992.

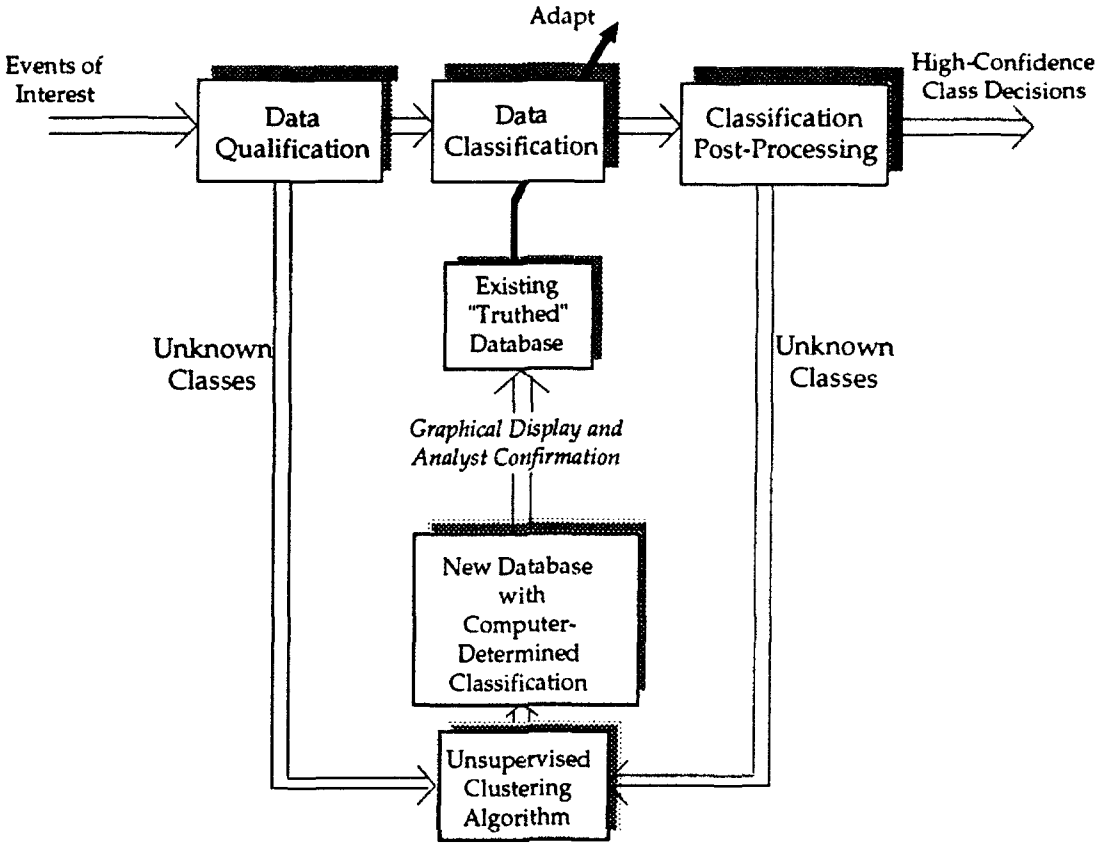


Figure 4.24: Automated Database Preparation and Classifier Retraining

Because the classification algorithm is a supervised learning technique (i.e., the algorithm is trained using a database that contains *a priori* information concerning exemplar classes), each new exemplar must first be assigned to a class before the network can be retrained. As already discussed, manually truthing new data is analyst-intensive and costly. Therefore, Fig. 4.24 proposes the incorporation of a *feedback loop* in the classification process. This feedback loop would use an unsupervised clustering algorithm to determine the natural groupings (i.e., classes) of novel data. The initial assumption is that detected unknown events represent new classes for which the classifier has not been trained. As each event is added to the database, a clustering algorithm would be used to determine if the event is "similar" in feature space to any other detected but unclassified events. After the clustering algorithm has completed its calculations, an analyst would be presented with a graphical display of the identified data clusters. By manually examining a few events within each cluster, an entire cluster of like events, perhaps containing hundreds or thousands of exemplars, may be automatically assigned to the same class of signals.

The development of techniques to implement the system shown in Fig. 4.24 is a logical choice for continuing research in the area of cost-effective on-line, near-real-time retraining of neural-based automated classification systems.

5. PNN SYSTEMS DELIVERED

During the course of investigating logistic-loss PNN techniques for the classification of AcW signals, Barron Associates, Inc. (BAI) delivered the following PNN-based AcW classification systems to the Government via Orincon Corporation:

(1) In October of 1991, BAI shipped software for a stand-alone system to be incorporated into the Build 2 application. This system, described in Section 5.1, included time-domain pre-processing, feature extraction, data qualification via hyperellipsoidal clustering, logistic-loss PNN classification, and multi-look post-processing. This system performed well on *Dataset B*, *Rangex*, and Build 2 Sea Trial data.†

(2) In January of 1993, BAI shipped software for a PNN classification system to be incorporated into the Build3 system. This system, described in Section 5.2, included principal component analysis routines and logistic-loss PNN classifiers. It was designed to make use of the Orincon Corporation pre-processing, feature extraction, and post-processing routines. This system performed well on *Rangex* Short-Net evaluation data [52].

The architecture and performance of these systems are described in detail below.

5.1 PNN Stand-Alone System for Build 2

5.1.1 Introduction

A PNN classification system was developed for incorporation in the Build 2 system for the July 1992 sea trial. The software was written using the ANSI standard "C" programming language and was tested at BAI as a stand alone system using signals from both *Dataset B*, and the *Rangex* data.

Orincon Corporation was responsible for the data interfaces, user interfaces, and any modifications required to take advantage of the I860 digital signal processors used in the PRISM hardware chassis. At the request of the Government, BAI sent engineers to Orincon prior to the sea trial to assist with the implementation of the core of the system (data pre-processing, single-scan feature extraction, and logistic-loss PNN classifiers).

---

† See Appendix B for a description of these datasets.

Subsequent to the sea trial, at the request of Orincon, BAI sent engineers to Orincon to assist in the implementation of further portions of the PNN system (moving-window feature calculations and multi-look post-processing). This version of the PNN software was trained on data collected during the sea trial and demonstrated by BAI at the October, 1992, Quarterly Progress Review meeting in San Diego, CA.

5.1.2 PNN Software Processing Flow

The standalone PNN system is designed to allow easy database management and network retraining to account for acoustic events that represent either new classes or new examples of an existing class. The software has three distinct processing chains: *Update, Resynthesize,* and *Interrogate* (Fig. 5.1).

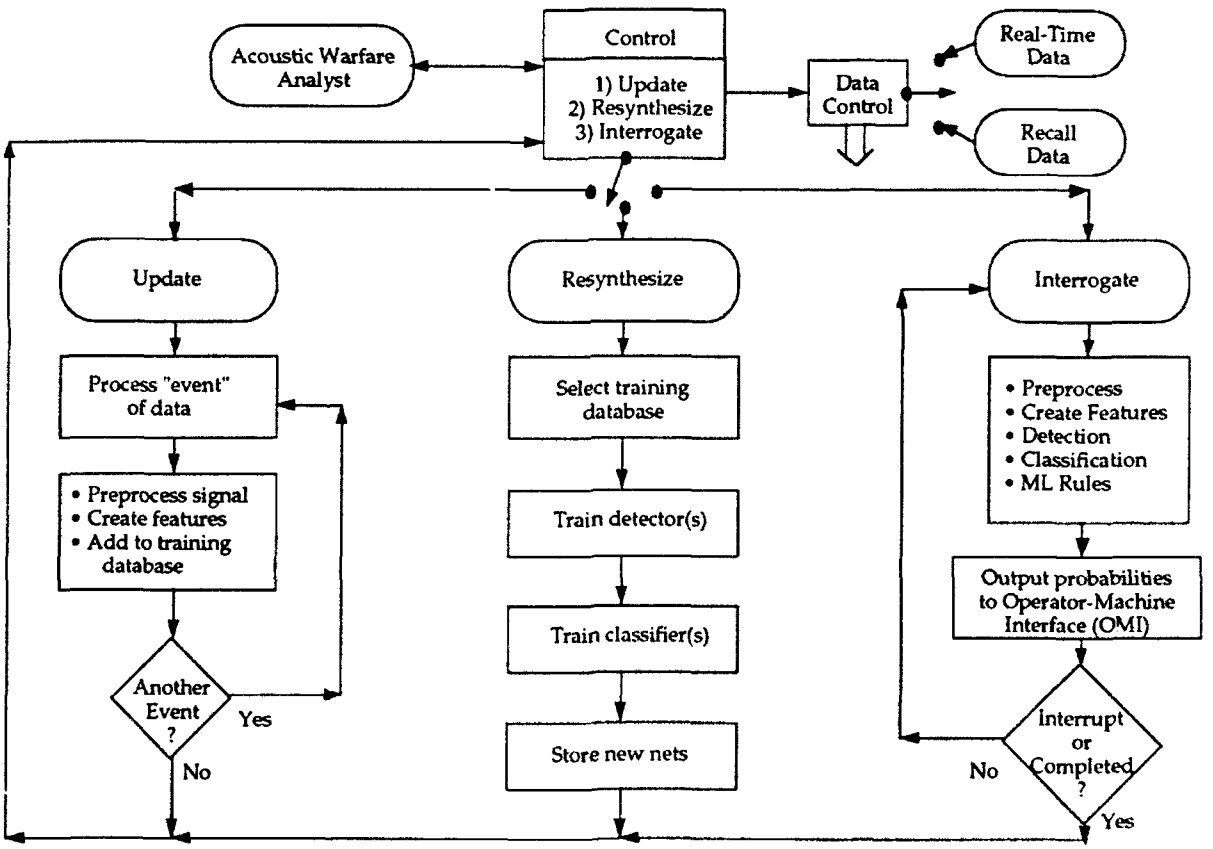


Figure 5.1: Build 2 PNN Software Processing Chains

The *update* chain updates the training databases to incorporate new classes or new exemplars for an existing class. This chain consists of data pre-processing and frequency-domain feature extraction. The features, along with a corresponding class number, are appended to the training database. If the class is new, it is assigned to

the network family that is closest in data space, and the PNN structure and corresponding model files are updated.

The *resynthesize* chain uses flags set by the *update* chain to determine whether the detection network or any of the classification networks needs retraining. If so, the information in the training database is used to modify the statistical information contained in the detection network. This information is also used to adjust the coefficients of a classification network. If a class has been added, then the structure of the classification network responsible for that family must be modified, and the network is completely retrained. The new detection and classification network information is stored in corresponding network files.

The *interrogate* mode preprocesses and extracts features in the same way as the *update* mode. The incoming data may be either real-time data or previously stored time-series data recalled for interrogation. If one or more classes are detected, appropriate classification networks are interrogated to determine the probabilities that the incoming data correspond to given classes. These probabilities are averaged over time and reported to the acoustic warfare analyst via the operator interface.

Fig. 5.2 shows the preprocessing and feature extraction signal processing used in both the *update* and *interrogate* processing chains, and Fig. 5.3 shows the data qualification and classification signal processing used in the *interrogate* processing chain.

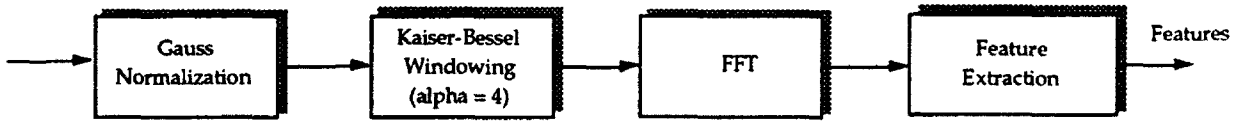


Figure 5.2: Build 2 PNN Preprocessing and Feature Extraction

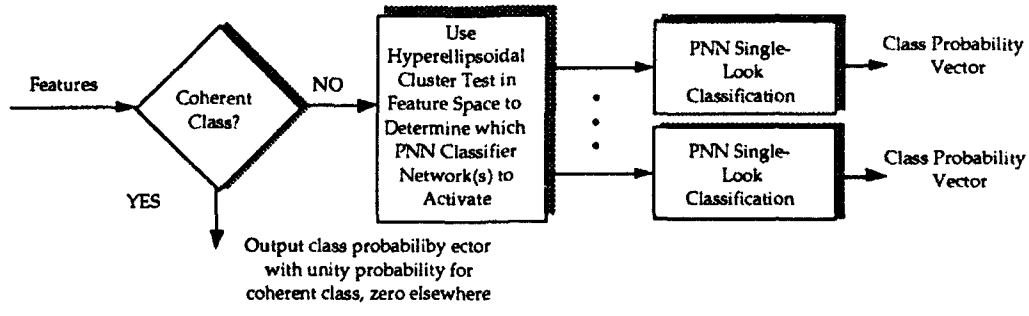


Figure 5.3: Build 2 PNN Data Qualification and Classification

The features chosen for the Build 2 PNN system were the 21 FFT-based features described in Section 4.4.3 and reproduced below for convenience:

1. Total spectral energy between 100Hz and 1000Hz (E_1).
2. Total spectral energy between 1000Hz and 3000Hz (E_2).
3. Total spectral energy between 3000Hz and 5000Hz (E_3).
4. The ratio $\frac{E_1}{1-E_1}$.
5. The ratio $\frac{E_1+E_2}{1-E_1-E_2}$.
6. The ratio $\frac{E_2}{1-E_2}$.
7. The frequency at which the largest spectral peak occurs (F_1).
8. The frequency at which the second largest peak occurs (F_2).
9. The frequency at which the third largest peak occurs (F_3).
10. The magnitude of the largest spectral peak (M_1).
11. The magnitude of the second largest spectral peak (M_2).
12. The ratio $\frac{M_1}{M_2}$. *(a measure of coherence)*
13. The ratio $\frac{M_1}{M_3}$. *(Where M_3 is the magnitude of the third largest peak)*
14. Frequency of 10% cumulative power (f_{10}). *(i.e.,10% energy below f_{10})*
15. Frequency of 25% cumulative power (f_{25}).
16. Frequency of 45% cumulative power (f_{45}).
17. Frequency of 70% cumulative power (f_{70}).
18. The ratio $\frac{0.10-0.00}{f_{10}-f_0}$. *(Where f_0 is 0.0.)*
19. The ratio $\frac{0.25-0.10}{f_{25}-f_{10}}$.
20. The ratio $\frac{0.45-0.25}{f_{45}-f_{25}}$.
21. The ratio $\frac{0.70-0.45}{f_{70}-f_{45}}$.

During database updating, the hyperellipsoidal clusters (HECs) used for data qualification are also used to assign automatically new classes to an appropriate family. As exemplars for a new class are added to the database, the update process

computes HEC statistics for the new class. These statistics are then used to add the class to the family containing the nearest existing HEC. If no existing HEC is near enough, or all families in the neighborhood contain the maximum allowable number of classes, then a new family is created, and the new class is assigned to it. The process of determining the family assignment of a new class is a type of unsupervised nearest-neighbor clustering in which new classes are assigned, if possible, to the HECs nearest to them in feature space. The processing logic for cluster updating is shown below in Fig. 5.4.

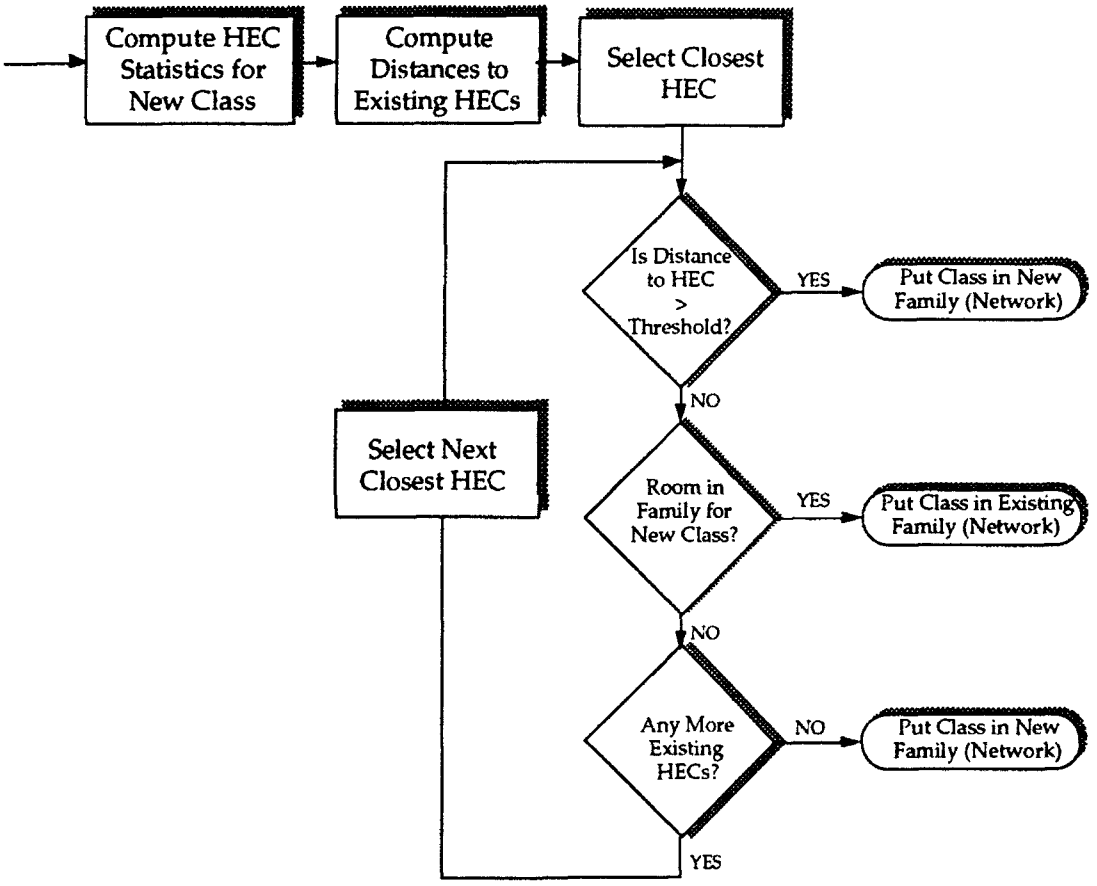


Figure 5.4: Using HECs to Assign a New Class to the Nearest Family

Every neural network (family) in the Build 2 PNN system contains a class for background noise, and this background class is assigned to the *baseline* class, C, as described in 3.4.1. Therefore, before a new network can be created or an existing network can be retrained, background exemplars must be present in the training database. This class is *not* used for the generation of the data qualification HECs.

Fig. 5.5 shows the postprocessing incorporated into the Build 2 PNN system.

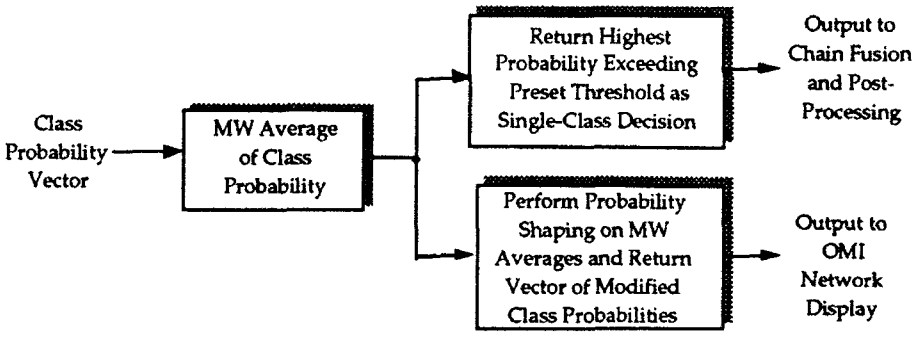


Figure 5.5: Build 2 PNN Classification Postprocessing

The single-look class probabilities are first averaged over a predetermined number of samples as shown in Eq. 4:10 (the number of samples that make up the average may be different for different classes). If the highest average probability exceeds a predetermined threshold, then the classifier declares an occurrence of the event corresponding to that probability. As discussed in Section 4.6, this method of post-processing improves results considerably.

For the PRISM implementation of the system, an additional probability-shaping post-processing step was required. The Build 2 operator-machine interface (OMI) was designed to display the non-probabilistic outputs of multi-layer-perceptrons (MLPs). To produce a color mapping that displays probabilistic network outputs well, a further transformation was required.

For a C-class problem, the probability shaping used was:

$$y_{OMI} = \begin{cases} 0 & \text{if } \hat{y} \le \frac{1}{C+1} \\ Ty'^2 & \text{if } \frac{1}{C+1} < \hat{y} \le T \\ \hat{y} & \text{if } \hat{y} > T \end{cases} \qquad 5:1$$

To ensure continuity, y' must be a function of the network output probability, $\hat{y}$, that takes on a value of zero at $\hat{y} = 1/(C+1)$ and a value of one at $\hat{y} = T$. The linear equation that meets these constraints is

$$y' = \frac{C+1}{T(C+1)-1}\left(\hat{y} - \frac{1}{C+1}\right) \qquad 5:2$$

It was found that setting T = 0.7 resulted in an improved OMI display; Fig. 5:6 shows this probability shaping function for T = 0.7 and C = 3.

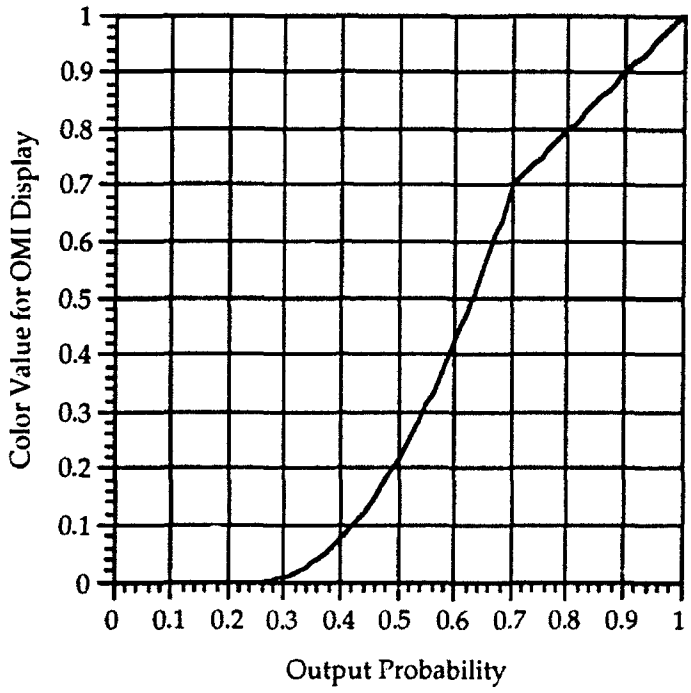


Figure 5.6: Output Probability Shaping

Prior to delivery of this system, the processing chain was evaluated on the two most complete datasets available: *Dataset B*, and the *Rangex* data.

The first evaluation was conducted on the 20-class problem using data taken from *Dataset B*. The 20 classes in this dataset were manually divided into four families as follows:

Table 5.1: Signal Family Groupings

Family	*Signals*
1	1, 2, 3, 4, 11, 16
2	5, 6, 8, 10, 13, 15
3	12, 17, 18, 20
4	7, 9, 14, 19

Tables 5.2 – 5.5 present the multi-look results of the Build 2 PNN processing chain evaluated on independent evaluation data from *Dataset B*..

Table 5.2: **Build 2 PNN Classification Results (Family One) for *Dataset B***

True Class	System Decision						
	Class 1	Class 2	Class 3	Class 4	Class 11	Class 16	Unknown
Class 1	1.0	0	0	0	0	0	0
Class 2	0	0.6	0.2	0	0	0	0.2
Class 3	0	0.4	0.6	0	0	0	0
Class 4	0	0.2	0	0.8	0	0	0
Class 11	0	0	0	0	1.0	0	0
Class 16	0	0	0	0	0	1.0	0
Unknown	0	0	0	0	0	0	1.0

Table 5.3: **Build 2 PNN Classification Results (Family Two) for *Dataset B***

True Class	System Decision						
	Class 5	Class 6	Class 8	Class 10	Class 13	Class 15	Unknown
Class 5	1.0	0	0	0	0	0	0
Class 6	0	1.0	0	0	0	0	0
Class 8	0	0	1.0	0	0	0	0
Class 10	0	0	0	1.0	0	0	0
Class 13	0	0	0	0	1.0	0	0
Class 15	0	0	0	0	0	0.8	0.2
Unknown	0	0	0	0	0	0	1.0

Table 5.4: Build 2 PNN Classification Results (Family Three) for *Dataset B*

True Class	System Decision				
	Class 12	Class 17	Class 18	Class 20	Unknown
Class 12	1.0	0	0	0	0
Class 17	0	0.5	0	0	0.5
Class 18	0	0	1.0	0	0
Class 20	0	0	0	1.0	0
Unknown	0	0	0	0	1.0

Table 5.5: Build 2 PNN Classification Results (Family Four) for *Dataset B*

True Class	System Decision				
	Class 7	Class 9	Class 14	Class 19	Unknown
Class 7	1.0	0	0	0	0
Class 9	0	1.0	0	0	0
Class 14	0	0	0.8	0	0.2
Class 19	0	0	0	1.0	0
Unknown	0	0	0	0	1.0

Results of the Build 2 PNN processing chain on the *Rangex* dataset have already been presented in Section 4.4.3, Tables 4.7 - 4.10. These tables are reproduced here for convenience (note that these results do not include the multi-look post-processing).

Table 5.6: **Build 2 PNN Classification Results (Family One) for *Rangex Data***

True Class	System Decision		
	Class 6	Class 11	Other
Class 6	1.0	0	0
Class 11	0	1.0	0
Other	0	0	1.0

Table 5.7: **Build 2 PNN Classification Results (Family Two) for *Rangex Data***

True Class	System Decision	
	Class 3 or 4	Other
Class 3 or 4	1.0	0
Other	0.06	0.94

Table 5.8: **Build 2 PNN Classification Results (Family Three) for *Rangex Data***

True Class	System Decision	
	Class 8	Other
Class 8	1.0	0
Other	0	1.0

Table 5.9: **Build 2 PNN Classification Results (Family Four) for *Rangex Data***

True Class	System Decision	
	Class 2	Other
Class 2	0.79	0.21
Other	0.03	0.97

5.1.3 *DANTES* Implementation

The PNN software as described above was originally delivered to the Government via Orincon Corporation on September 30, 1991. During the period July 6-19, 1982, BAI sent a consultant to Orincon Corporation at the request of the Government to assist with the installation of a version of the Build 2 PNN software on the *DANTES* system. Because access to the *DANTES* hardware was severely limited due to conflicting priorities related to preparation for the sea trial, the software installed in the *DANTES* system had somewhat reduced capabilities from the PNN software as originally delivered. Nevertheless, the work produced significant benefits for the over-all *DANTES* program, because: (1) valuable technical insights, working relationships, and know-how were acquired by both BAI and Orincon during installation of the PNN software in Build 2, and (2) the experience gained by BAI while at Orincon provided a basis for subsequent evaluation of the PNN algorithms on the *DANTES* hardware with the Build 2 sea-trial data.

The following PNN capabilities were integrated in the *DANTES* system for the Build 2 sea trial:

- *Real-Time Interrogation*: The PNN networks can be interrogated in real-time, providing class probabilities and classifier decisions to the PRISM OMI interface.

- *Collection of Exemplar Data*: The operator can collect exemplar data and generate the features required by the PNN using the PRISM OMI interface. The PNN does not require a fixed retina size, so the operator can adjust the retina size depending on the duration of signals being captured.

- *On-Line PNN Training* (X-Window and command-line interfaces): An existing PNN can be updated, or a new network can be created created by using an operator-selected subset of the exemplar data that have been collected.

- *Database Management*: The operator can incrementally build a training database by selecting various exemplar files. Utilities are provided for examining and editing the class composition of the exemplar files.

- *Configuration Management*: The operator can store and retrieve various trained networks using keyword identification. For example, if the operator wishes to retrieve a network that was saved under the keyword "Jul30", all files associated with that network can be automatically retrieved and installed.

The following capabilities of the original Build 2 PNN software were not implemented, although they should be restored in subsequent implementations of the PNN classification software:

- *Family Processing:* The multi-family capability of the PNN system should be activated and tested. This will allow for automatic assignment of new classes to particular detection clusters and their corresponding neural networks. In conjunction with this effort, an interface screen should be designed which will allow the operator to see which classes have been assigned to which PNNs and to override manually any automatic class assignments.

- *Code Optimization*: The PNN code should be reviewed and optimized for the PRISM architecture. As part of this optimization, the code should be tested for both functionality and statistical performance on the Build 2 sea trial data, and the algorithm should be tuned to provide accurate, robust answers.

5.1.4 Build 2 Sea Trial Test Plan

The following recommendations were made by BAI concerning the testing and evaluation of the PNN software:

5.1.4.1 *Interrogation*

The PNNs were trained on classes for which laboratory data were available. The time available for pretraining the networks at Orincon was limited due to the heavy use of the system hardware for other purposes in the weeks immediately preceding the sea trial. Nevertheless, it was recommended that the pretrained networks should be interrogated with at-sea data that contain one or more of the transients for which for which the network was pretrained. Classifier and detection performance should be noted.

When the BAI PNN software was connected to the Orincon OMI, there was a great deal of clutter in the network output display. Two factors contributed to this clutter: (1) The neural networks were designed to output estimates of the true conditional probabilities of the classes and background noise. As with any set of probabilities, these values summed to unity. In most applications decisions are made by comparing the probability values. The OMI, however, displayed each "raw" class probability value. Thus, "unlikely" events which had some low probability assigned to them showed up as color on the display. The probability shaping described above represented an attempt at filtering these probabilities to remove "unlikely" outputs from the display and make the visual evaluation of the PNN software easier. (2) When the PNNs were applied to short-duration transients, a class output could be high for one or two pixels only; these pixels may or may not have been visually detectable by the operator; looking at the detector

output may help. BAI recommended that the operator be aware of these interface issues and comment on any interface problems that could be readied by additional post-processing.

5.1.4.2 *Update and Resynthesis*

Because the PNN can be trained very rapidly, it will be possible to perform more thorough network training at sea. One anticipated use for the PNN is in the very rapid learning and tracking of new transients; especially when there is not time to train the MLPs. BAI recommended that network retraining be tested by gathering a few exemplars for background noise and some known transient signals at the beginning of a data collection run, retraining the PNNs *immediately*, and then interrogating the networks on the live data before completing the data collection run. It would be especially important to record the number of classes assigned to the PNN, whether the retraining was partial or complete. (Did retraining include data from the laboratory or previous data collection runs, or were all the data new?) In this test, complete retraining would be more desirable. It would also be helpful to record how much time was required for the network training.

Because retraining is not computationally expensive, it was recommended that the PNN be trained a number of times. Experiments are desirable concerning the effectiveness of the PNN when trained on laboratory data alone, on a mixture of laboratory data and at-sea data, and when trained on at-sea data alone.

5.1.4 Build 2 Evaluation

Due to difficulties encountered by Orincon Corporation during the sea trial, none of the tests outlined above was conducted. Therefore, Orincon requested that BAI send personnel to Orincon to train and demonstrate the PNN system on data recorded during the Build 2 sea trial.

Once again, due to limited access to the *DANTES* hardware, BAI was unable to incorporate the multi-family processing capabilities as originally desired. However, BAI was able to include moving window features (see Section 4.4.5) to improve classification performance on longer signals. The following moving-window features were added:

(1) MW averages of all 21 single-scan features.

(2) MW standard deviations of all 21 single-scan features.

DANTES OMI was used to create a training database for the neural networks. This proved extremely difficult, especially for the shorter signals, due to the low-resolution of the FFT/Feature display. Nevertheless, BAI was able to extract data as shown in Table 5.10.

Table 5.10: Build 2 Sea Trial Data Used for PNN Demonstration

Class	Total Events	Events Used for Training	Number of Training Exemplars Used†
Background	N/A	N/A	111
Class 1	5	4	109
Class 2	3	1	22
Class 3	6	3	15
Class 4	1	1	32
Class 5	10	3	9
Class 6	1	1	20
TOTAL	26	13	318

A total of 318 exemplars were extracted from 13 of the 26 "truth" events, and an eight-class PNN was trained in 29 minutes on a Sparc 1E (equivalent to less than one minute on the Mercury processor). While it was difficult to create a truly independent evaluation set, and the amount of training data used was small, judging from the OMI display during real-time interrogation, the minimum-logistic-loss PNN classifier performed well on events that were withheld during training. Quantitative results, however, were unavailable due to the limited capabilities of the Build 2 OMI in the areas of exemplar extraction and database management (especially for the shorter signals).

In light of the qualitatively successful demonstration of a streamlined version of the PNN algorithm on data collected during the sea trial, the following recommendations were made:

- *Training Speed:* Port the PNN training algorithm to the Mercury processors. The 40-fold increase in floating-point processing speed would allow for very rapid at-sea retraining. (As mentioned above, *the demonstration network would have trained in less than a minute on the Mercury processor*).

- *Family Processing:* Incorporate the multi-family capabilities of the PNN code as originally intended. This would allow for automatic assignment of new classes to a particular network. The results would be improved training time and improved classification accuracy.

---

† For the longer-duration events, it is possible to extract more than one training exemplar from a single event.

- *Reduction in Signal Features:* Incorporate principal components of the (now 63) features to reduce the number of features, allow use of greater PNN nonlinearities, and further reduce the training time.

- *DANTES Pre- and Post-Processing:* Fully incorporate the PNN algorithms into the *DANTES* processing chain to allow the algorithm to take advantage of Orincon's detection procedures and normalized FFTs. This would also allow the IP and MHT to take advantage of the PNN classifier.

5.2 Short-Net System for Build 3

5.2.1 Introduction

Subsequent to the Build 2 sea trial, Orincon requested that BAI assist in the inclusion of the PNN technology into the Build 3 system. To conduct an evaluation of the proposed classifier processing chains, Orincon required quantitative results of PNN algorithm performance on the *Rangex* "Short-Net" data. This section presents these quantitative results. Additionally, some important comments are made regarding interpretation of Short-Net evaluation results for both the multi-layer Perceptron (MLP) and the PNN given the nature of the *Rangex* database.† ‡

The quantities of training data originally given to BAI are shown in Table 5.11

Table 5.1: Short-Net Training Database

Class	Unclassified Abbreviation	Number of Occurrences
S1	BIO	35
S2	DW	11
S3	P	37
S4	SD	9
S5	WSR	7

† Most of this work was funded by Orincon Corporation under P.O. S04365, and these results were first reported to Orincon in a letter to Mr. Mike Kurnow dated January 8, 1993.

‡ BAI was also given data for five transients of relatively long duration. However, BAI concentrated on the short-net classes for the following reasons: (1) the short-net classes are the most difficult to classify correctly; (2) in the Build 2 work, Orincon encouraged BAI to concentrate on these classes; and (3) BAI's time and resources for this experiment were limited.

BAI later learned that some of the events contained in the training and evaluation data sets were not used in the training and evaluation of Orincon's networks. Not knowing which events to leave out, BAI worked with all the data.†

5.2.2 Processing Flow

The PNN algorithm, as originally delivered by BAI to the Government via Orincon, used multiple minimum-logistic-loss PNNs and a hyperellipsoidal detector to achieve improved classification. For purposes of the Short-Net comparison, however, the detector was removed and all five classes were assigned to a single PNN classifier; BAI believes that this configuration yielded acceptable but sub-optimal results. An additional modification was made in that, whereas all the original PNN classifiers contained a "background" class, no "background" class was used for this evaluation. The re-inclusion of a "background" or "other" class in the actual implementation of the classifier processing chains is strongly encouraged, because it is highly unlikely that every event detected will be a class for which a network has been trained.

The small numbers of *Rangex* Short-Net training data made available for classes S2, S4, and S5 require that the following precautions be taken during network training and evaluation:

1. Interpret cautionsly any statistical measure on a data set comprised of a few exemplars. (For class S5, a 14% improvement was obtained by classifying a single additional evaluation exemplar correctly.)

2. Given a class with only seven exemplars, network complexity should be restricted to six degrees of freedom if overfitting is to be avoided. In the case of minimum-logistic-loss PNN classifiers having linear input layers, this means limiting the number of inputs.

Because of the restrictions on network complexity, BAI was unable to use the 63 FFT-based features described in Section 5.1. Instead, (1) only a single retinal scan of the Short_Net1 retina was used (historical feature data was ignored), and (2) a principal component analysis was performed on the single scan; the principal components that explained the largest amount of variance in the data were used as inputs to the network. Once again, were more data available, these techniques would be sub-optimal; however, for this limited training data set, a network was obtained with a statistically justified complexity that could be expected to perform comparably on unseen data from the same population as the training data.

---

† In the case of S3, BAI removed four events from the training database for which the SNR was so low as to make the event hardly distinguishable from background noise.

One advantage to using principal components is that as more training data become available, more principal components may be used, and optimum logistic-loss networks can be rapidly retrained to employ the greater complexity allowed by the additional data. Additionally, if the PNN algorithm is implemented as originally envisioned, of multiple networks (families) may be used with more complex networks handling the classes for which more data exist.

Orincon requested that no post-processing be included in the PNN processing chain. Fig. 5.7 shows the PNN processing flow for the Short-Net software:

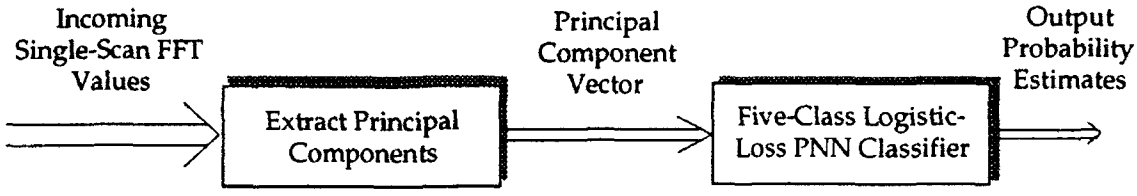


Figure 5.7: PNN Short-Net Processing Flow

5.2.3 Database Issues and Discussion

Orincon Corporation properly withheld from BAI the evaluation portion of the Short-Net database, and there were not enough events in the training data base to allow its further partitioning into training and evaluation sets. Therefore, except for Table 5.19 which represents results on Orincon's independent test set, all of BAI's estimates of network performance were obtained via a *jackknife* approach.

In the jackknife approach, a single training exemplar is removed from the database, and a network is trained on the remaining exemplars. This network is then evaluated on the exemplar that was removed. If there are N training exemplars, this technique is performed N times, and the statistics of the resulting N evaluations are combined to provide an estimated performance matrix. This technique provides a statistically sound estimate of how a network will perform on unseen data, provided that (1) the unseen data are drawn from the same statistical population as the training data, and (2) the classes in the unseen data base are represented in the same proportions as those in the training data base (i.e., if the training data containes twice as many exemplars for class 1 as for class 2, the evaluation data should also containe approximately twice as many exemplars for class 1 as for class 2.)

A class for which a significant number of training exemplars existed is S1. The jackknife statistics for a one-vs.-all minimum-logistic-loss classifier for class S1

using ten principal components are presented in Table 5.12. Note that the S1 network, while incorrectly dismissing 8.6% of the data, had no false alarms.†

Table 5.12: PNN Short-Net One-vs.-All Performance for S1

True Class	System Decision	
	Class S1	Other
Class S1	0.91	0.86
Other	0.00	1.00

The other populous class was S3. The jackknife statistics for a one-vs.-all minimum-logistic-loss classifier for class S3 using ten principal components are presented in Table 5.13. The numbers in parentheses represent the network performance obtained when the "other" class did not include class S1.

Table 5.13: PNN Short-Net One-vs.-All Performance for S3

True Class	System Decision	
	Class S3	Other
Class S3	0.76 (0.90)	0.24 (0.10)
Other	0.12 (0.22)	0.89 (0.78)

For the three remaining classes, for which the size of the training set was extremely small, a less-complex network had to be used to achieve a robust design. The jackknife statistics for a three-class minimum-logistic-loss network with linear input layers, using five principal components, are shown in Table 5.14:

† BAI found that use of a single additional feature, the kurtosis of the time series, resulted in perfect classification of this event. In general, the kurtosis may prove very useful in discriminating between man-made and biological signals; however, BAI decided that for the purposes of the current experiment, this feature was too "data dependent" and might lead to misplaced optimism.

Table 5.14: PNN Short-Net Three-Class Network for Less Populous Classes

True Class	System Decision		
	Class S2	Class S4	Class S5
Class S2	0.64	0.18	0.29
Class S4	0.11	0.67	0.14
Class S5	0.29	0.14	0.57

Network accuracies when an "other" class was added to the above three-class network are shown in Table 5.15:

Table 5.15: PNN Short-Net Three-Class Network for Less Populous Classes and "Other" Class

True Class	System Decision			
	Class S2	Class S4	Class S5	Other
Class S2	0.37	0.18	0.09	0.36
Class S4	0.11	0.56	0.11	0.22
Class S5	0.29	0.14	0.57	0.0
Other	0.03	0.03	0.02	0.92

Based on the statistics presented in the above tables, it is seen that network accuracy improved greatly when more data per class were available. This behavior is presented graphically in Fig. 5.8:

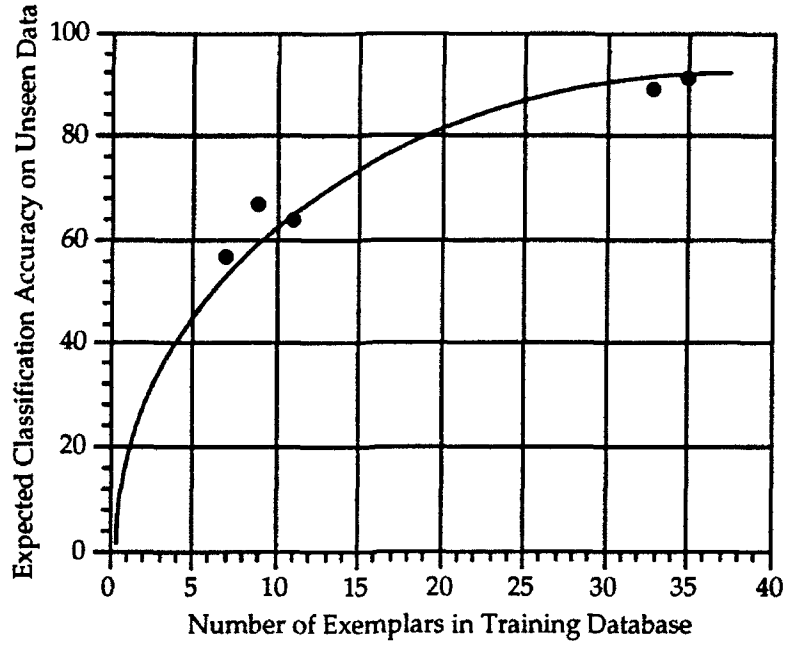


Figure 5.8: PNN Short-Net Network Accuracy vs. Number of Training Exemplars per Class

Presumably, differences in accuracy between the classes were partly due to some classes being more difficult to discriminate than others (e.g., classes S1 and S3); however, for this data set, three of the classes did not contain enough data to train a robust network and accurately predict performance on unseen data. This can be seen in the October '92 QPR matrices given for Orincon's Short_Net1 and Short_Net2, reproduced in Table 5.16 for convenience:

Table 5.16: Percentage Correct for Two Orincon Networks

Network	Short Signals				
	Class S1	Class S2	Class S3	Class S4	Class S5
Short_Net1	0.91	0.84	0.80	0.53	0.50
Short_Net2	0.82	0.40	0.77	0.78	0.33

While the two networks of Table 5.16, using frequency-domain features, achieved somewhat comparable results on classes S1 and S3, there was a great deal of variance in the results for the other three classes. Table 5.17 gives a summary of the statistics of all the Short-Net classifiers.

Table 5.17: Performance Summary for all Short-Net Classifiers

Statistics	Short Signals				
	Class S1	Class S2	Class S3	Class S4	Class S5
Number of Exemplars	35	11	37	9	7
Mean Performance	0.91	0.58	0.80	0.52	0.47
Performance Variance	0.07	0.19	0.08	0.30	0.18

As can be seen from Table 5.17 (and Fig. 5.8), the mean classification accuracy for all Short-Net classifiers improved with the number of training exemplars. Additionally, there was a very large variance among the Short-Net classifiers for the classes with limited numbers of independent training exemplars (S2, S4, S5). These statistics imply that the different classification results for the other classes may not have been entirely due to the features used or network structures used, but may also have been evidence of the dangers involved when networks are trained on very small data sets (even when data set sizes were artificially increased via addition of noisified exemplars).

BAI also noted that even with a single classification technology (i.e., minimum-logistic-loss PNN classifiers), slight changes in the features (e.g., the number of principal components used) or the network structure (e.g., linear vs. additive polynomial nodal elements) resulted in large differences in performance on classes for which limited training data existed. These indicators suggest that *all of the Short-Net classification networks may have been overfitting the data for those classes having few training exemplars.* This overfitting may account for many of the performance differences between the various Short-Net classifier chains.

For the five-class *Rangex* data set in question, therefore, three options were and are available concerning the creation of classification networks:

1. Subdivide the data to allow multiple networks of differing complexities to handle the classification problem. This approach will give statistically optimal classifiers; however, it requires the use of multiple neural networks for different sets (families) of classes.

2. Use a five-class network, but limit network complexity so as not to overfit the class for which the fewest number of exemplars exists. This will result in a sub-optimal five-class network that does not overfit any of the classes.

3. Use a five-class network, but give the network sufficient complexity so as to perform well on the classes for which significant training data do exist. This will result in a five-class network that overfits the classes for which

limited quantities of independent training data exist. This *must* be taken into account in the decision fusion process. With subsequent additions to the database, the network can be retrained to eliminate its overfitting.

BAI presented classification results using the first option above; however, for purposes of comparison with Orincon's short nets, BAI followed Option 3 and trained a five-class network allowing sufficient complexity to perform well on classes S1 and S3.

5.2.4 Short-Net Classification Results

Using ten principal components and a second-degree additive network polynomial form (see Section 3.2.1) the jackknife statistics shown in Table 5.18 were obtained:

Table 5.18: Five-Class PNN Short-Net "Jackknife" Classification Results (Option 3)

True Class	System Decision				
	Class S1	Class S2	Class S3	Class S4	Class S5
Class S1	0.80	0.09	0.03	0.03	0.05
Class S2	0.09	0.64	0.27	0	0
Class S3	0.03	0.12	0.73	0.06	0.06
Class S4	0	0.11	0.22	0.56	0.11
Class S5	0	0.14	0.14	0.29	0.43

This network had a figure of merit (FOM) of 63% (the average of the diagonals); however, once again, it should be emphasized that any contribution to the FOM made by classes S2, S4, and S5 should be taken cautiously.

A five-class network was then trained using all the training data. The learning curve for this network is shown in Fig. 5.10:

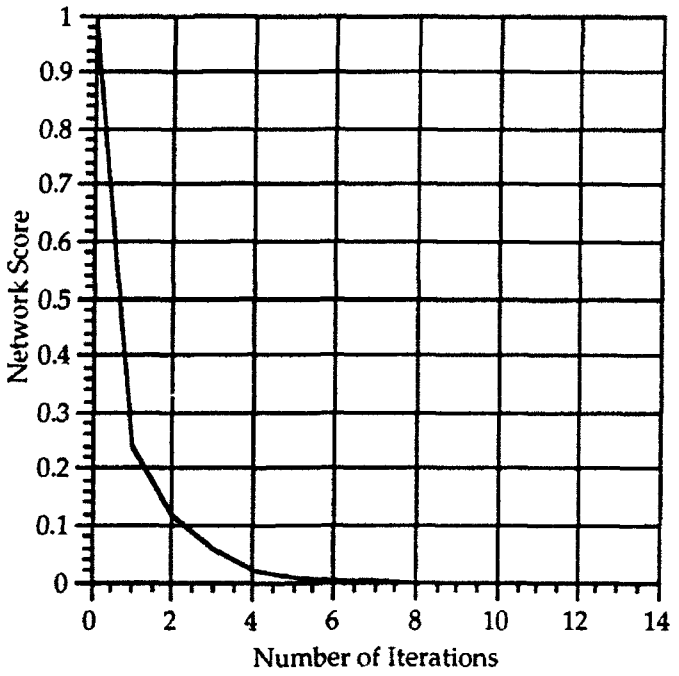


Figure 5.10: Learning Curve for a Five-Class PNN Short-Net Classifier Using Ten Input Features and 81 Total Network Coefficients

The five-class classifier was evaluated at Orincon Corp. using independent evaluation data; the results of this evaluation are shown in Table 5.19.†

These results bear out the caveat stated in Option 3 above: Even though the FOM (60%) is comparable to that predicted by the jackknife statistics, there is a significant variance in the performance on the classes for which limited quantities of independent data exist. This confirms the earlier assertion that the network statistics for these classes are less reliable than the statistics for the other classes (for which larger quantities of independent evaluation data exist). Since these less reliable statistics are allowed to contribute equally in the overall FOM calculation, the resulting FOM may also have been an unreliable indicator (possibly either optimistic or pessimistic) of future network performance.

---

† A single FFT scan was extracted from each exemplar. BAI used the same bins, 16-163, as Orincon. BAI's FFT routines were used for feature generation and did NOT remove background noise by normalizing the FFT; however, it is believed that the different FFT normalization was not a signfiicant factor in the results.

Table 5.19: Five-Class PNN Short-Net Classification Results on Evaluation Data (Option 3)

True Class	System Decision				
	Class S1	Class S2	Class S3	Class S4	Class S5
Class S1	0.91	0.03	0	0	0.06
Class S2	0.08	0.54	0.15	0	0.23
Class S3	0.03	0.09	0.71	0.03	0.14
Class S4	0.10	0.40	0.20	0.10	0.20
Class S5	0	0.29	0	0	0.71

In conclusion, the Short-Net experiments show that for this particular limited data set, the PNNs and MLPs exhibited similar performance. However, the PNN classifiers trained considerably faster than the MLPs (which used backward-error propagation), and the PNNs were considerably less complex. In fact, information theory implies that networks with a large degree of complexity are not justified when very limited quantities of training data are available. As additional and more representative data become available at sea, more complex networks will be statistically justified. Using PNN techniques, statistically justified networks can be synthesized on-line to obtain improved performance.

6. CONCLUSIONS AND RECOMMENDATIONS

For both estimation and classification problems, the benefits of using artificial neural networks include inductive learning, rapid computation, and the ability to handle high-order and/or nonlinear processing. Neural networks reduce the need for simplifying assumptions that use *a priori* statistical models (such as "additive Gaussian noise") or that neglect nonlinear terms, cross-coupling effects, and high-order dynamics.

This report demonstrates the usefulness of an interdisciplinary approach that applies the rigorous theory and algorithms of statistical learning theory to the field of artificial neural networks. In particular, this approach provides two important results:

(1) A generalized way of viewing neural modeling in terms of statistical function estimation.

(2) A constrained minimum-logistic-loss polynomial neural network (PNN) classification algorithm.

The successful application of the PNN classification algorithm is demonstrated for transient acoustic warfare (AcW) signals. Additionally, a number of algorithmic improvements are suggested to allow the rapid automatic learning of multi-layered PNN classification network structures and coefficients.

One of the significant problems facing the design of neural-network-based AcW signal processing systems is the lack of significant numbers of exemplars for off-line training. The proposed algorithms address this problem in two ways: (1) they make use of information-theoretic criteria to create networks with a statistically-justified level of complexity that do not overfit the training data and generalize well on unseen data, and (2) they make use of rapid Gauss-Newton optimization techniques that are suitable for rapid on-line retraining of classification networks as more data become available.

However, neural network algorithmic enhancements can only partially make up for deficiencies in the quantity and diversity of the training database; the work reported underscores the need for further development of systems for the automated collection and management of large amounts of AcW data. Only when such data are available and fully exploited will the complete benefits of neural network techniques be realized.

7. REFERENCES

[1] Abbott, D.W., P.R. Jordan, R.L. Barron, *Application of Polynomial Networks to Detection and Classification of Non-Steady-State Signals*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., June, 1990.

[2] Abbott, D.W., P.R. Jordan, R.L. Barron, *Application of Polynomial Networks to Detection and Classification of Non-Steady-State Signals*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., Nov., 1990.

[3] Abbott, D.W., P.R. Jordan, P.R., R.L. Barron, Dr. B. E. Parker, *et al., Application of Polynomial Networks to Detection and Classification of Transient AcW Signals*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., Mar., 1991.

[4] Akaike, H., "Information theory and an extension of the maximum likelihood principle," *Proc. Second Int'l. Symp. on Information Theory*, B.N. Petrov and F. Csaki (Eds.), Akadémiai Kiadó, Budapest, 1972.

[5] Barron, A.R., "Predicted Squared Error: A Criterion for Automatic Model Selection," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 4, 1984.

[6] Barron, A.R., *Logically Smooth Density Estimation*, Ph.D. Dissertation, Dept. of E.E., Stanford University, Palo Alto, CA, 1985.

[7] Barron, A.R. and R.L. Barron, "Statistical learning networks: A unifying view," *Proc. 20th Symposium on the Interface: Computing Science and Statistics*, Reston, VA, Apr., 1988.

[8] Barron, A.R., "Statistical properties of artificial neural networks," *Proc. IEEE 1989 Conf. on Decision and Control*, Tampa, FL, Dec., 1989.

[9] Barron, A.R., "Complexity regularization with applications to artificial neural networks," *Proc. NATO ASI on Nonparametric Functional Estimation*, Spetses, Greece, G. Roussas, Ed., Kluwer Academic Publishers, Dordrecht, Netherlands, Aug., 1990.

[10] Barron, A.R. and X. Xiao, "Discussion on multivariate adaptive regression," *Annals of Statistics*, Vol 19, No. 1, 1991.

[11] Barron, A.R., "Approximation and estimation bounds for artificial neural networks," *Computational Learning Theory: Proc. of 4th Ann. Workshop*, Morgan Kaufman, 1991.

[12] Barron, R.L., "Adaptive transformation networks for modeling, prediction, and control," *Proc. Joint Nat'l. Conf. on Major Systems*, IEEE/ORSA, Oct., 1971.

[13] Barron, R.L., "Guided accelerated random search as applied to adaptive array AMTI radar," *Proc. Adaptive Antenna Systems Workshop*, Naval Research Laboratory, Washington DC, 11-13 Mar., 1974.

[14] Barron, R.L., "Theory and application of cybernetic systems: An overview," *Proc. 1974 NAECON*, May, 1974.

[15] Barron, R.L., A.N. Mucciardi, F.J. Cook, J.N. Craig, and A.R. Barron, "Adaptive Learning Networks: Development and Application in the United States of Algorithms Related to GMDH," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 2, 1984.

[16] Barron, R.L., D.W. Abbott, *Application of Polynomial Neural Networks to Transient Signal Analysis*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., July, 1989.

[17] Barron, R.L., D.G. Ward, *Algorithms for Synthesis of Dynamic Polynomial Neural Networks for Estimation and Classification*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., July, 1989.

[18] Barron, R.L., D.W. Abbott, R.L. Cellucci, P.R. Jordan, *Application of Polynomial Networks to Signal Analysis*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., Jan., 1990.

[19] Barron, R.L., P. Hess, P.R. Jordan, III, and C.M. Hawes, *Diagnostic Abductive and Inductive Reasoning for Flight Control Effector FDIE and Reconfiguration, Part I: Synthesis Algorithms and Results for Pre-Trained FDIE, Part II: Recursive Estimation of Aircraft Parameters Using Neural Network Calculation of Impairment Probability*, Barron Associates, Inc. Final Technical Report for Flight Dynamics Directorate, Wright Laboratory (AFSC), under Contract F33615-88-C-3615, WL-TR-91-3108, Mar., 1992.

[20] Breiman, L. and J.H. Friedman, "Estimating optimal transformations for multiple regression and correlation," *J. Amer. Statist. Assoc.*, Vol. 80, 1985.

[21] Daubechies, I., *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics, Capital City Press, Montpelier, VT, 1992.

[22] Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[23] Farley, B.G. and W.A. Clark, "Simulation of self-organizing systems by digital computers," *IRE Trans. on Inform. Theory*, Vol. PGIT-4, 1954.

[24] Farlow, J. (Ed.) "The GMDH Algorithm," *Self-Organizing Methods in Modeling: GMDH Type Algorithms,* Marcel Dekker, Inc., New York, Chap. 1, 1984.

[25] Fisher, R.A., "The use of multiple measurements in axonomic problems," *Annals of Eugenics,* Vol. 7, 1936.

[26] Friedlander, B. and B. Porat, "Detection of transient signals by the Gabor representation," *IEEE Trans. of ASSP,* Vol. 37, No. 2, Feb., 1989.

[27] Friedman, J.H. and J.W. Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Trans. on Computers,* Vol. 23, 1974.

[28] Friedman, J.H. and W. Stuetzle, "Projection pursuit regression," *J. Amer. Stat. Assoc.,* Vol. 76, 1981.

[29] Friedman, J.H., "Fitting functions to noisy scattered data in high dimensions," *Proc. 20th Symposium on the Interface: Computing Science and Statistics,* Reston, VA, Apr., 1988.

[30] Friedman, J.H., "Multivariate adaptive regression splines," *Annals of Statistics,* Vol 19, No. 1, 1991.

[31] Gabor, D., "Communication theory and cybernetics," *Trans. of IRE,* Vol. CT-1, No. 4, 1954.

[32] Gabor, D., P.L. Wilby, and R. Woodcock, "A universal non-linear filter, predictor and simulator which optimizes itself by a learning process," *J. IEE,* paper received Oct. 17, 1959.

[33] Garth, L.M. and H.V. Poor, "Detection Techniques for Acoustic Non-Gaussian Signals," *Technical Report NOSC TD 1855,* Naval Ocean Systems Center, San Diego, May, 1990.

[34] Ghosh, J., L. Deuser, and S. Beck, "A neural network based hybrid system for detection, characterization and classification of short-duration oceanic signals," *IEEE J. of Oceanic Engineering,* Vol. 17, No. 4, Oct., 1992.

[35] Giles, C.L. and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics,* Vol. 26, No. 23, Dec., 1988.

[36] Gilstrap, L.O. Jr., "An adaptive approach to smoothing, filtering and prediction," *Proc. 1969 NAECON,* 1969.

[37] Golub, G.H. and C.F. Van Loan, *Matrix Computations,* Johns Hopkins University Press, Baltimore, 1983.

[38] Haykin, S. *Adaptive Filter Theory,* Prentice Hall, Englewood Cliffs, NJ, 1986.

[39] Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley Publ. Co., Reading, MA, 1989.

[40] Hush D.R., and B.G. Horne, "Progress in supervised neural networks: What's new since Lippmann?," *IEEE Signal Processing Magazine*, Jan., 1993.

[41] Ivakhnenko, A.G., "The group method of data handling — A rival of stochastic approximation," *Soviet Automatic Control*, Vol. 1, 1968.

[42] Ivakhnenko, A.G., "Polynomial theory of complex systems," *IEEE Trans. on Systems, Man, & Cybernetics*, Vol. SMC-1, No. 4, Oct., 1971.

[43] Levenberg, K., "A method for the solution of certain nonlinear problems in least squares," *Quart. Appl. Math.*, vol. 2, 1944.

[44] Ljung, L., and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983.

[45] Mallows, C.L., "Some comments on Cp," *Technometrics*, Vol. 15, 1973.

[46] Mardia, K.V., J.T. Kent, and J.M. Bibby, *Multivariate Analysis*, Academic Press, London, 1979.

[47] Marple, S.L. , *Digital Spectral Analysis with Applications*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[48] Marquardt, D.W., "An algorithm for least-squares estimation of non-linear parameters," *Journal SIAM*, vol. 11, 1963.

[49] McCulloch, W.S. and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, Vol. 5, 1943.

[50] Minsky, M.L. and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, M.I.T. Press, Cambridge, MA, 1969.

[51] Moddes, R.E.J., R.j. Brown, L.O. Gilstrap, Jr., R.L. Barron, et al., *Study of Neurotron Networks in Learning Automata*, Adaptronics, Inc., AFAL-TR-65-9, Feb. 6, 1965.

[52] Orincon Corporation, "DARPA Non-Traditional Exploitation System Quarterly Review." Presentation to personnel of DARPA, NOSC, and Barron Associates, Oct. 1992.

[53] Parker, B.E., W.A. Patterson, D.W. Abbott, R.L. Barron, *Application of Polynomial Networks to Detection and Classification of Transient AcW Signals*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., July, 1991.

[54] Parker, B.E., W.A. Patterson, D.W. Abbott, R.L. Barron, *Application of Polynomial Networks to Detection and Classification of Transient AcW Signals for BUILD 2*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., Oct., 1991.

[55] Parker, B.E., T.M. Nigro, M.P. Carley, R.L. Barron, D.G. Ward, H.V. Poor, D. Rock, T.A. DuBois, "Helicopter gearbox diagnostics and prognostics using vibration signature analysis," *SPIE Int'l. Symp. on Optical Engineering and Photonics in Aerospace and Remote Sensing*, Vol. 1965, Paper No. 39, Orlando FL, Apr., 1993.

[56] Poor, H.V., *An Introduction to Signal Detection and Estimation*, Springer-Verlag, 1988.

[57] Press, W.H., Brian P. Flannery, *et al., Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, NY, 1986.

[58] Reklaitis, G.V., A. Ravindran, and K.M. Ragsdell, *Engineering Optimization Methods and Applications*, John Wiley & Sons, 1983.

[59] Rissanen, J., "A universal prior for integers and estimation by minimum description length," *Annals of Statistics*, Vol. 11, No. 2, 1983.

[60] Robbins, H. and Monro, S., "A stochastic approximation method," *Annals of Math. Stat.*, 22, 1951.

[61] Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958.

[62] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,Vol. 1: Foundations*, M.I.T. Press, Cambridge, Massachusetts, 1986.

[63] Schwarz, G., "Estimating the dimension of a model," *Ann. Stat.*, Vol. 6, No. 2, 1977.

[64] Shin, Y. and J. Ghosh, "The pi-sigma network: An efficient higher-order network for pattern classification and function approximation," *Proc. IEEE Joint Conf. Neural Networks*, July, 1991.

[65] Shrier, S., R.L. Barron, and L.O. Gilstrap, "Polynomial and Neural Networks: Analogies and Engineering Applications," *Proc. IEEE First Int'l. Conf. on Neural Networks*, Vol. II, June, 1987.

[66] Shynk, J.J., "Adaptive IIR filtering," *IEEE ASSP Magazine*, Vol. 6, No. 2, Apr. 1989.

[67] Specht, D.F., "Generation of polynomial discriminant functions for pattern recognition," *IEEE Trans. on Electronic Computers*, Vol. EC-16, No. 3, pp. 308-319, June, 1967.

[68] Ward, D.G. and R.S. Bahiti, "Parallel Algorithms and Architectures for Kalman Filters," *IEEE Conference on Decision and Control*, Jan., 1987.

[69a] Ward, D.G., B.E. Parker, Jr., and R.L. Barron, *Active Control of Complex Systems Via Dynamic (Recurrent) Neural Networks*, Barron Associates, Inc. Task Final Technical Report for ONR, Contract N00014-89-C-0137, May 1992.

[69b] Ward, D.G., B.E. Parker, Jr., C.M. Hawes, and R.L. Barron, *Active Control of a Multivariable System Via Polynomial Neural Networks*, Barron Associates, Inc. Final Technical Report for ONR, Aug., 1992.

[70] Ward, D.G., R.L. Barron, P. Hess, *PNN Classifier Results on Range-X Short-Net Data*, Presentation to personnel of DARPA, NOSC, and Orincon Corp., Jan., 1993.

[71] Werbos, P.J., "Backpropagation: Past and future," *Proc. of the Intl. Conf. on Neural Networks*, I, 3430353, IEEE Press, New York, July, 1988.

[72] White, D. and D. Sofage, *Handbook of Intelligent Control*, Van Nostrand Reinhold, New York, 1992.

[73] Widrow, B. and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Conv. Record*, New York, 1960.

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

APPENDIX A: UNCLASSIFIED CLASS ABBREVIATIONS

Table 4.1

Data Received from Orincon Corp. Prior to January, 1990

Class 1:	P__
Class 2:	Pile Driver
Class 3:	Noise from P__
Class 4:	Noise from Pile Driver
Unknown 1:	Ice
Unknown 2:	Seal Bomb

Tables 4.2 - 4.6; 4.11; 4.12 - 4.15; 4.16 - 4.19; 4.20; 5.2 - 5.5

Dataset B: Received from Orincon Corp. Prior to June, 1990

Class 1:	T__ F__
Class 2:	T__ St__
Class 3:	T__ R__
Class 4:	T__ Sp__
Class 5:	V__ St__
Class 6:	V__ H__
Class 7:	Fl__ R__
Class 8:	St__sl__
Class 9:	Ping #1
Class 10:	Active Sonar
Class 11:	Pile Driver
Class 12:	Killer Whale
Class 13:	Seal Bomb
Class 14:	Snapping Shrimp
Class 15:	Ice
Class 16:	Rain
Class 17:	W S
Class 18:	2nd Propeller Moter
Class 19:	Ping #2
Class 20:	Triple Ping

Tables 4.7 - 4.10; 5.6 - 5.9.

Rangex Data: Received from Orincon Corp. Prior to December 1990

Class 1:	Catss Pulse
Class 2:	Catss BB
Class 3:	Slam W/T Door

Class 4: D__ W__
Class 5: RMS Pulse
Class 6: T__ WS
Class 7: (not used)
Class 8: Sig__ Ej__ WS
Class 9: (not used)
Class 10: (not used)
Class 11: SPM

Tables 5.12 - 5.15; 5.16 - 5.18; 5.19

Rangex Data: Received from Orincon Corp. December, 1992

Class 1: Biologic
Class 2: D__ W__
Class 3: P__
Class 4: S__ D__
Class 5: WS R.

APPENDIX B: CHRONOLOGY OF BAI WORK EFFORT

		System Description			
QPR Date	Data Date[1]	Pre- Processing	Features	Classification Method	Post-Processing
1/90	1/90	–	PNPs	*ASPN*	Multi-Look
6/90	6/90	KB Window HECs	HEC Distances	*CLASS*	Single-Look Multi-Look
6/90	6/90	Passbank Filter Scaling	PNPs	*CLASS*	–
11/90[2]	6/90	KB Window HECs	HEC Distances FFT-Based	*CLASS*	Single-Look Multi-Look
3/91	12/90	LPF Auto-Center KB Window	FFT-Based	*CLASS*	–
7/91[3] 10/91	12/90	KB Window HECs	FFT-Based	*CLASS*	Multi-Look
10/92[4]	Build2	KB Window	FFT-Based Historical	*CLASS*	Multi-Look
2/93[5]	12/92	KB Window	PCAs of FFT	*CLASS*	–

[1]See Appendix A.

[2]Interrogation on Rangex data demonstrated at QPR.

[3]On-line retraining on Rangex data demonstrated at 7/91 QPR; Demonstration of BAI PNN software based on this processing chain presented at 10/91 QPR. This software formed the bases for the Build 2 implementation.

[4]Training and interrogation demonstrated on PRISM hardware using data collected from Build 2 sea trial.

[5]Results of Rangex short-net evaluation funded in part by Orincon Corporation.

APPENDIX C: BUILD 2 PNN SOFTWARE DOCUMENTATION†

1. Algorithm Summary

The PNN algorithm is summarized in Section 5.1.

2. PNN Procedures

2.1 Introduction

This appendix describes the configuration and procedures for operation of the PNN Build 2 software. If there are any questions about using the PNN software, please do not hesitate to contact us using the numbers shown below.

Paul Hess:	(800) 323-8790	(24 hrs)
David G. Ward:	(804) 985-4401	(Barron Associates Office)
Ross Mahtafar:	(619) 455-5530, x414	(Orincon Office)

2.2 Real Time Version

A stand alone version of the real time PNN software resides on the Dantes workstation in the following directory:

/home1/ross/prism2/Scripts

There is also a version of the real time PNN integrated into the overall Build 2 software. Please check with Ken Stanwood (Orincon) to find out the current location and configuration of these files.

Pre-trained PNN networks are available in the directory:

/home1/dantesInt/dantes/pnnTrainer

Install the pre-trained network by copying the files PNN.model, class.net, detect.net, and training to the directory from which you will run the PSC program. Refer to the file */home1/dantesInt/dantes/pnnTrainer/bck/pre_test.readme* for information concerning the event names and class numbers for this pre trained network.

PNN has been linked under the existing "SVD" menu item. **To invoke PNN, select the "SVD" menu item for display while running the PRISM interface.**

---

† Hess, P., Barron, R.L., Ward, D.G., and Mahtafar, R. *Polynomial Neural Network (PNN) Acoustic Warfare Software: Configuration and Test Procedures for the Build 2 DANTES System Sea Trial,* Report prepared by Barron Associates, Inc. for Orincon Corporation, July, 1992.

Background noise will not appear in the PNN output display as a separate class during build 2, but will have been used as part of the computation of the probabilities of the other classes.

2.3 Collecting Exemplar Data

There is a single PNN network with a maximum of 11 signal classes. You may use various retina sizes for collecting exemplar data. To configure the retina size, edit the exemplar_rows and pixel_rows settings in the "dantes.ocfg" file and restart the OMI interface.

exemplar_rows	This is the actual size of the retina which will be captured by the PNN. Set this to any value from 1 through 8 (4 is a good value for most short and medium duration signals).
pixel_rows	This is the size of the yellow capture window which will be displayed on the screen. The best setting for pixel_rows is two greater than exemplar_rows. The effect will be that the selection box will not cover up and hide the actual pixels that are being captured.

Figure 2-1 illustrate an example setting for these parameters.

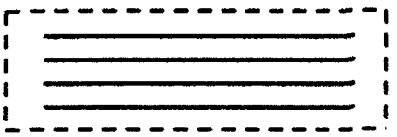


Fig 2-1: In this example, the retina is set up for an exemplar size of 4 to be captured by the PNN. The pixel_rows setting is 6, which allows all four rows of the retina data to fit inside yellow capture window.

When selecting exemplars from a signal, be very sure not to include background noise or other classes of signals within the capture window. If it is difficult to position the window to include only the signal of interest, then reduce the retina size.

Since the PNN Training software allows you to quickly and easily select multiple data files for training, it is a good idea to output your data into small manageable units. During data collection at Orincon we would typically create data files of 10-30 exemplars. It is a good idea to collect separate files containing background noise, since background exemplars must be included during all training sessions.

> Background noise must be specified as class 12.

2.4 PNN Trainer and Utilities

The PNN neural network training and database management takes place in the directory called */home1/dantesINT/dantes/pnnTrainer*. Figure 2-2 illustrates the subdirectory directory structure for the trainer and utilities.

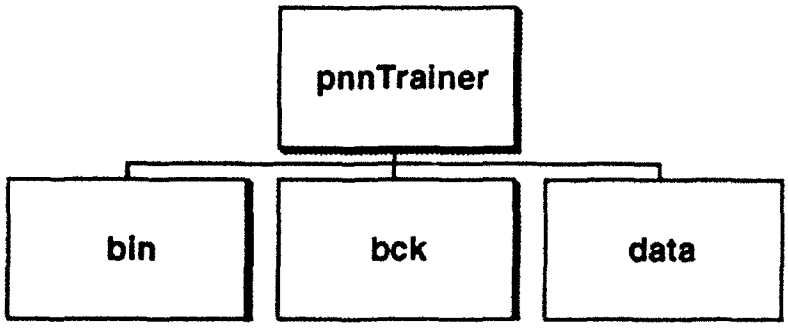


Figure 2-2: Directory structure of pnnTrainer

The "bin" subdirectory contains all of the programs and utilities available:

PNNTrainX	-	PNN Trainer (X-Windows version) [sect. 2.4.1]
PNNTrain	-	PNN Trainer (command line version) [sect. 2.4.2]
LookClass	-	Examines ".sel4" exemplar files [sect. 2.4.3]
ChangeClass	-	Edits ".sel4" exemplar files [sect. 2.4.4]
PNNBck	-	Backs up a current PNN network for later use [sect. 2.4.5]
PNNGet	-	Recalls previously backed up PNN network [sect. 2.4.6]

It is recommended that the operator include this "PNNTrainer/bin" directory as part of the UNIX path environment variable. If this is not done, the operator will have to specify the bin pathname each time the PNN Trainer and utilities are used.

The "bck" subdirectory contains archived PNN networks are stored and retrieved using the PNNBup and PNNUnBup commands. See sections 2.4.5 and 2.4.6 for more information.

The "data" subdirectory contains the original exemplar data files extracted at Orincon from the Sound Designer data. The operator is encourage to create additional directories and subdirectories to manage the data and networks generated during the sea trial.

2.4.1 PNNTrainX

PNNTrainX is the X-Window's version of the PNN trainer program. The PNNTrainX develops a new PNN network or updates an existing PNN network in the directory that the program is started from. The PNN trainer will automatically determine whether to update or completely resynthesize the network files.

The four files associated with the PNN network are: PNN.model, class.net, detect.net, and training. The three buttons on the PNNTrainX main window are initialize, file, and synthesize.

Initialize: Pressing the initialize button will clear the PNN.model, class.net, detect.net, and training files so that you may begin creating a new PNN network from scratch.

File: Pressing the file button will open a file selection window which allows you to specify training data to add to the PNN working database (which is accumulated in the local training file). Use the file selection window to go to the directories where the raw exemplar data (*.sel4 files) are stored. Select a file to add by

clicking on its name and press the load button. The command window will display an itemization of the exemplars and classes that were contained in the new file, and the summary window will show a history of exemplars that have been added so far.

Synthesize: Synthesize will create (or update) a PNN model using the data accumulated in the training file. Watch the command window for the progress of training. First the PNN detector will be trained, then the classifier. Training the classifier is an iterative process (much like the Fishnet training process). Each iteration you will see the current and best scores displayed in the command window. Training will take anywhere between 1 and 25 iterations.

> Remember, unless you select initialize, there is no need to load data that has been loaded in prior sessions. Use the file button to add new data only. If you press the initialize button, then all data must be re-added.

> IMPORTANT: IN ORDER TO SYNTHESIZE A NETWORK, YOUR TRAINING FILE MUST CONTAIN BACKGROUND NOISE EXEMPLARS. THERE MUST ALSO BE NO MISSING CLASS NUMBERS. FOR EXAMPLE, IT WOULD BE ILLEGAL TO USE EXEMPLARS FROM CLASSES 1 AND 3, BUT NOT FROM CLASS 2. USE THE ChangeClass UTILITY (section 2.4.4) TO MODIFY THE CLASS NUMBERS OF THE EXEMPLAR FILES TO COMPLY WITH THIS RULE.

When training is complete, you may install the new networks by copying the files into the proper script directory. The best way to do this is to create a named backup using the PNNbck command (sect. 2.4.5), changing to your script directory, and then retrieving the backup with the PNNget command (sect. 2.4.6).

Once a network is created, it is useful to keep a worksheet describing the classes which the network has been trained for and the data files which were used for training.

2.4.2 PNNTrain

PNNTrain is a command-line version of the PNNTrainX program. Run the program by typing "PNNTrain" along with any command line arguments. Typing PNNTrain by itself will display instructions on how to use the program. There are three modes of running the program:

PNNTrain init	Clears the PNN.model, class.net, detect.net, and training files so that you may begin creating a new PNN network from scratch.
PNNTrain filenames...	Adds the exemplar files to the training database. UNIX wildcard characters may be used for convenience. For example, you may type "PNNTrain p*.sel4" to add all exemplar files whose name begins with p. The program may be run multiple times in this mode to add all of the data for training.
PNNTrain go	This mode will create (or update) a PNN model using the data accumulated in the training file. First the PNN detector

will be trained, then the classifier. The PNN trainer will automatically determine whether to update or completely resynthesize the network files. Training the classifier is an iterative process (much like the Fishnet training process). Each iteration will be displayed along with the current and best scores. Training will take anywhere between 1 and 25 iterations.

Remember, unless you use init mode, there is no need to add data that has been loaded in prior sessions. Add new data only. If you use the initialize mode, then all data must be re-added.

IMPORTANT: IN ORDER TO SYNTHESIZE A NETWORK, YOUR TRAINING FILE MUST CONTAIN BACKGROUND NOISE EXEMPLARS. THERE MUST ALSO BE NO MISSING CLASS NUMBERS. FOR EXAMPLE, IT WOULD BE ILLEGAL TO USE EXEMPLARS FROM CLASSES 1 AND 3, BUT NOT FROM CLASS 2. USE THE ChangeClass UTILITY (section 2.4.4) TO MODIFY THE CLASS NUMBERS OF THE EXEMPLAR FILES TO COMPLY WITH THIS RULE.

When training is complete, you may install the new networks by copying the files into the proper script directory. The best way to do this is to create a named backup using the PNNbck command (sect. 2.4.5), changing to your script directory, and then retrieving the backup with the PNNget command (sect. 2.4.6).

2.4.3 LookClass

This is a useful utility which can be used for both PNN and Fishnet exemplar files. Run the program by typing "LookClass <filename>" where <filename> is the name of an exemplar (.sel4) file.LookClass will display summary information about the file such as number of exemplars, number of cases per exemplar, and total number of cases in the file. It will then list each exemplar and print the class that the exemplar belongs to.

Once a network is created, it is useful to keep a worksheet describing the classes which the network has been trained for and the data files which were used for training.

2.4.4 ChangeClass

This is a useful utility which can be used for both PNN and Fishnet exemplar files. The ChangeClass utility will automatically edit an exemplar file to change selected class numbers. For example, you may wish to change all class 5's into class 2's. Other classes within the same file will remain unchanged.

The syntax for using the ChangeClass utility is as follows:

ChangeClass infilename outfilename classnum newclassnum

Infilename is the name of the exemplar (.sel4) file to be edited. Outfilename is the name of the new, edited exemplar file. Classnum is the number of the class to be changed. Newclassnum is the number of the class which Classnum is to be changed into.

2.4.5 PNNBck

PNNBck is a backup utility which will create an archive of the PNN.model, class.net, detect.net, and training files. The files will be stored in the *pnnTrainer/bck* directory under a user supplied keyword. The syntax for the PNNBck command is:

PNNBck keyword

2.4.6 PNNGet

PNNGet is a backup utility which will retrieve an archive of the PNN.model, class.net, detect.net, and training files. The files will be retrieved from the *pnnTrainer/bck* directory and stored in the directory from which you run the command. The syntax for the PNNGet command is:

PNNGet keyword

Example:

You have just finished synthesizing a PNN network which you may wish to use. Archive the network using the command:

PNNBck Jul19

Where Jul19 is the keyword to store the network files under. When you wish to use the files, go to the location that you want the files restored to and type:

PNNGet Jul19

2.5 Development Directories

Following is a list of the development directories for editing and running the standalone versions of the PNN software. To run the PNN in an integrated fashion, changes to this source code must be checked into the PRISM configuration control and rebuilt.

1. */home1/ross/prism2/pnn/tool.d*	Type 'make' to build, then build the mcp.d (see 4)
2. */home1/ross/prism2/omi.d*	Type 'make' to build, then build the mcp.d (see 4)
3. */home1/ross/prism2/pnn/train*	Type 'make' to build text version, then move the file PNNTrain into /home1/dantesINT/dantes/PNNTrainer/bin
	Type 'make -f pnn_gui.make' for GUI,
then move the file PNNTrainX into	/home1/dantesINT/dantes/PNNTrainer/bin
4. */home1/ross/prism2/mcp.d*	Type 'make' to build, then type 'mv *.860 ../S*' to install.

5. */home1/ross/prism2/pnn/backup* Backup versions of the tool and train directories.

IMPORTANT: Prior to Sea Trials please check in (to co. figuration management) all of the source code in the tool.d directory and rebuild the software. See Ken Stanwood for more details. The pre-trained networks must also be installed in the proper script directory (use the PNNGet command, sect. 2.4.6).

3. Build 2 Sea Trial Test Plan

3.1 Introduction

The three PNN modes, *update, resynthesize, and interrogate,* are to be tested in the DANTES Build 2 sea trial. Other than the Background class, the AcW classes used for laboratory pretraining of the PNN detection and classification software are described in the *.readme file in the directory */home1/dantesINT/dantes/pnnTrainer/bck.*

Bear in mind that the scope and amount of PNN pretraining have been extremely limited.

The PNN software is presently configured for use with up to eleven AcW waveform classes plus the Background class, and the PNN training algorithm provides capabilities for very rapid updating and resynthesis of the PNN at sea. The *updating* capability is to be exercised extensively during the sea trial to augment the pretrained PNN and to evaluate the speed, accuracy, and utility of on-line training. The *interrogate* mode is to be exercised throughout the PNN sea trial.

Updating and interrogation of the PNN during the Build 2 sea trial should emphasize work with Acoustic Classes and Background signals of *short-to-medium* duration.

The recommended test plan follows. This plan is not intended to be rigid. For instance, it may be valuable to update the PNN relatively early if exemplars of a novel acoustic class become available at the opening of the trial, or if the background noise is characteristically different from that which was used during training.

3.2 Test Interrogation of Pre-Trained PNN

Prior to any updating, at least a few basic interrogation tests of the pretrained PNN should be performed. The steps are:

(1) Familiarize personnel with the PNN operator-machine interface. (Be aware that, for short-duration signals, a class output may be high in color value for one or two pixels only. If the high-color-coded pixels are not readily detectable by the operator, looking at the detector output as a cue might be helpful.) The operating personnel are requested to log their comments regarding OMI issues so that these can be resolved in Build 3.

(2) Test the PNN interrogation process when the input waveforms are those for the ambient Background. Log the detection and classification performance results for this test and the comments of operators.

(3) Test the PNN interrogation responses for representative accoustic waveforms, particularly for accoustic waveforms in the classes for which the network has been pre-trained. Log the detection and classification performance results and the comments of operators.

3.3 Test PNN Updating Procedure and Algorithm

To evaluate the existing Build 2 PNN software and help Orincon Corporation and Barron Associates in coming work on the Build 3 system, it will be important to test the PNN updating procedure and algorithm. The steps are:

(1) Following the protocol in Section 2, update the pretrained PNN exemplars for *one* additional AcW class. Log the CPU time required for the updating and log the comments of the operators.

(2) Test the PNN interrogation responses for specifically the added AcW class, preferably using new exemplars of this class rather than the exemplars used for updating. Log the detection and classification performance results and the comments of operators.

(3) Test the PNN interrogation responses for one or more of the other AcW classes that were represented in the PNN pretraining. Test the PNN on Background. Log performance results and operator comments.

3.4 Continue PNN Updating and Training with Further Accoustic Classes

Continue as in Section 3.3, adding further accoustic classes. Also, attempt to create a completely new PNN network using only Background and accoustic class exemplars obtained during the sea trial.

4. Comments on Scope of Build 2 Software

Due to the tight availability of resources leading up to the sea trial, a subset of the total PNN algorithm was chosen for integration into the Build 2 sea trial.

The primary capability being tested is the core PNN classification algorithm and the rapid retraining capability. Analysis of results from the PNN classifier should be cognizant of the fact that the neural network receives a limited input vector of 21 features not based on historical data. This performance will provide a baseline which may improve dramatically for certain longer term signals with the addition of historical feature processing.

Future systems will also contain integration of a family structure for the classes. Family structures allow the PNN to group accoustic signals which are statistically similar using a partially unsupervised learning technique. Also, because the number of output nodes for any given classifier of the PNN will be reduced, training time will grow only gradually as the system expands to handle many more classes. Analysis of data and results from this sea trial of the single family PNN will assist in determining the specific choice of algorithms which will be used.

Due to the limited availability of resources, limited verification and validation has been performed on the build 2 software.