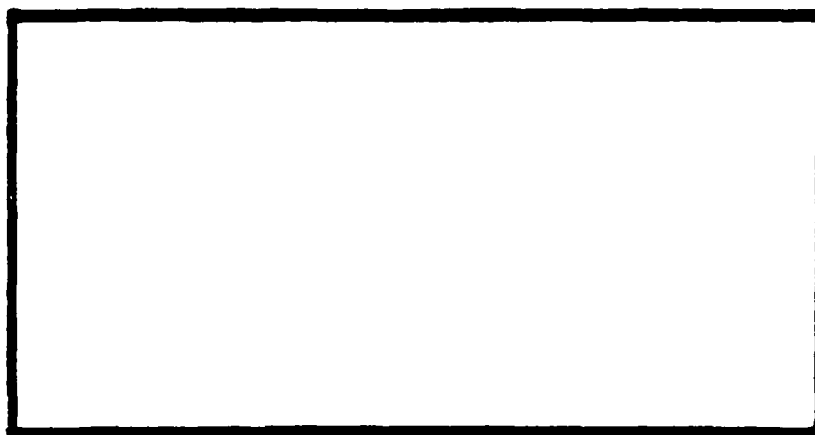
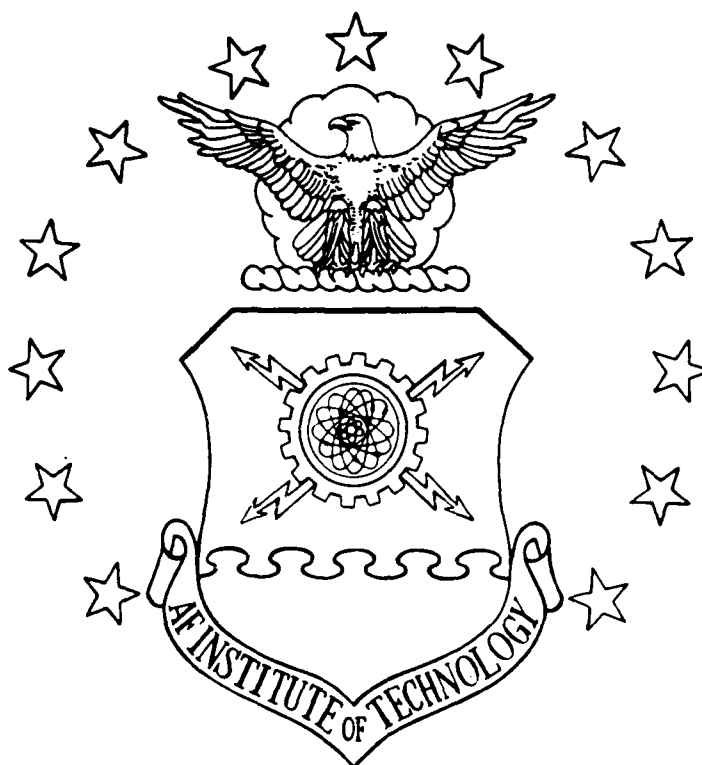


DTIC FILE COPY

①

AD-A229 035



DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

DTIC
S **E** **D**
ELECTE
NOV 30 1990

Wright-Patterson Air Force Base, Ohio

90 11 26 177

AFIT/DS/ENG/90-2

Characterization of Multilayer Perceptrons and their
Application to Multisensor Automatic Target Detection

DISSERTATION

Dennis William Ruck
Captain, USAF

AFIT/DS/ENG/90-2

DTIC
S **ELECTE** **D**
NOV 30 1990
E

Approved for public release; distribution unlimited

AFIT/DS/ENG/90-2

Characterization of Multilayer Perceptrons and their Application to
Multisensor Automatic Target Detection

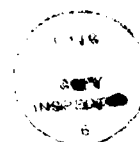
DISSERTATION

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Dennis William Ruck, B.S.E.E, M.S.E.E
Captain, USAF

December 1990



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

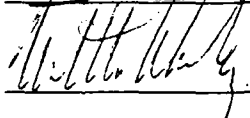
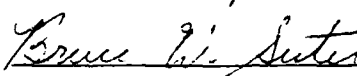
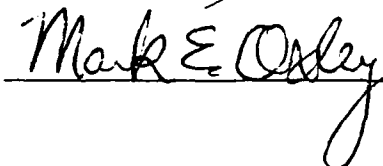
Approved for public release; distribution unlimited

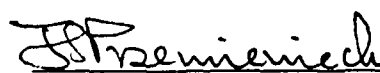
Characterization of Multilayer Perceptrons and their Application to
Multisensor Automatic Target Detection

Dennis William Ruck, B.S.E.E, M.S.E.E

Captain, USAF

Approved:

	<u>4 Oct 90</u>
	<u>4 Oct 90</u>
	<u>4 Oct 90</u>
	<u>4 Oct 90</u>

 11 Oct 90

J. S. Przemieniecki

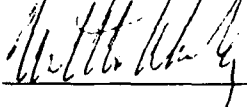
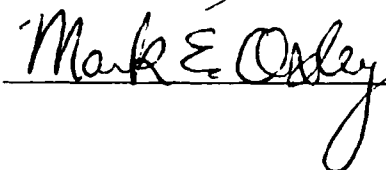
Institute Senior Dean

Characterization of Multilayer Perceptrons and their Application to
Multisensor Automatic Target Detection

Dennis William Ruck, B.S.E.E, M.S.E.E

Captain, USAF

Approved:

	4 Oct 90
	4 Oct 90
Bruce W. Suter	4 Oct 90
	10 Oct 90

 11 Oct 90

J. S. Przemieniecki

Institute Senior Dean

Preface

The goal of this research was to expand understanding of multilayer perceptrons for target detection. This has been accomplished. The primary result is that the multilayer perceptron classifier is really just an approximation to the Bayes optimal discriminant functions. However, the implementation of the multilayer perceptron shows benefits over the traditional statistical classifiers in terms other than accuracy such as speed, compactness, ease of implementation and so forth.

I wish to acknowledge my indebtedness to my committee chairman, Dr. Steven K. Rogers, whose tireless and unfailing support of my work was crucial to my success. Most of the significant results were accomplished at his urging, though I was skeptical at the time of the possible connections he suggested. Also, I would like to thank my committee members both current and former: Dr. Matthew Kabrisky, Dr. Mark E. Oxley, and Dr. James P. Mills whose support and encouragement of my efforts were always welcome. Also, many of the comparisons between learning rules would not have been possible without the necessary computer support for which I thank the system engineers, Dan Zambon and Bruce Clay.

Dennis William Ruck

Table of Contents

	Page
Preface	iii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
List of Symbols	x
Abstract	xii
 I. Introduction	 1
1.1 Historical Background	1
1.2 Problem Statement and Scope	2
1.3 Dissertation Organization	4
 II. Background Material	 5
2.1 Basic Concepts of Target and Pattern Recognition	5
2.2 The Multilayer Perceptron Architecture	10
2.3 Multilayer Perceptron Training Algorithms	13
2.3.1 Backpropagation	14
2.3.2 Extended Kalman Filtering Training Algorithm	16
2.4 Test Methodologies	21
2.4.1 Database Partitioning	21
2.4.2 Confidence Interval Techniques	23

	Page
III. Feature Selection -	
Analyzing the Weights in a Multilayer Perceptron	30
3.1 Introduction	30
3.2 Why Limit Input Features?	30
3.3 Conventional Feature Selection Techniques	31
3.4 The Saliency Metric	32
3.4.1 Introduction	32
3.4.2 Derivation of Saliency Metric	32
3.4.3 Test of the Saliency Metric	38
3.4.4 Application to Learning Rule Evaluation	45
3.5 Conclusion	46
IV. Understanding the Backpropagation	
Training Algorithm	49
4.1 Introduction	49
4.2 Degenerating Kalman to Backpropagation	49
4.3 Implications	52
4.4 Performance Comparisons Between Backpropagation and Extended Kalman Filtering	54
4.5 Conclusion	75
V. The Multilayer Perceptron:	
A Bayes Optimal Discriminant	
Function Approximator	76
5.1 Introduction	76
5.2 Two Class Problem	77
5.3 Multiclass Problem	80
5.4 Discussion	82
5.5 Conclusion	83

	Page
VI. Application to Multisensor Automatic Target Detection	86
6.1 Introduction	86
6.2 Problem Description	86
6.3 The Multisensor Fusion Approach	90
6.4 Conclusion	93
VII. Recommendations and Conclusions	94
7.1 Recommendations	94
7.2 Conclusions	95
Appendix A. Network Sizing Techniques	97
Bibliography	100
Vita	103

List of Figures

Figure	Page
1. Target Recognition Architecture	6
2. Multilayer Perceptron Architecture	11
3. Detail of a Node in the Multilayer Perceptron	12
4. Comparative Performance of Kalman Training to 500,000 Iterations	21
5. Comparative Performance of Kalman Training to 100,000 Iterations	22
6. Comparative Performance of Kalman Training to 10,000 Iterations	23
7. Summary of Equations for Monte Carlo Confidence Interval Technique . .	29
8. Summary of Equations for Classifier Confidence Interval Method	29
9. Multilayer Perceptron Architecture	35
10. Detail of a Node in the Multilayer Perceptron	36
11. Rank Histograms of Saliency for Three Moment Invariants	40
12. Rank Histograms of Saliency for Three FLIR Imagery Features	43
13. Rank Histograms of Saliency Metric Using Kalman Learning	47
14. XOR Classification Problem	56
15. Average Training Accuracy on XOR Data <i>versus</i> Iterations	57
16. Average Training Accuracy on XOR Data <i>versus</i> FLOPS	58
17. Mesh Classification Problem	60
18. Mesh Training Accuracy <i>versus</i> Iterations	61
19. Mesh Training Accuracy <i>versus</i> FLOPS	62
20. Mesh Decision Regions: Kalman <i>versus</i> Backpropagation	63
21. Mesh Decision Regions Using Backpropagation	64
22. Learning Rule Comparisons on Doppler Data: Training Accuracy <i>versus</i> Iterations	66
23. Learning Rule Comparisons on Doppler Data: Testing Accuracy <i>versus</i> Iterations	67

Figure		Page
24.	Learning Rule Comparisons on Doppler Data: Training Accuracy <i>versus</i> FLOPS	68
25.	Learning Rule Comparisons on Doppler Data: Testing Accuracy <i>versus</i> FLOPS	69
26.	Target Detection Training Accuracy in Absolute Range Imagery <i>versus</i> Iterations	71
27.	Target Detection Testing Accuracy in Absolute Range Imagery <i>versus</i> Iterations	72
28.	Target Detection Training Accuracy in Absolute Range Imagery <i>versus</i> FLOPS	73
29.	Target Detection Testing Accuracy in Absolute Range Imagery <i>versus</i> FLOPS	74
30.	Two Class Decision Boundary Example	84
31.	Typical Range Image and its Segmented Version	87
32.	Typical FLIR Image and its Segmented Version	88

ist of Tables

Table		Page
1.	Comparator of Monte Carlo and Classifier Confidence Interval Methods .	28
2.	FLIR Features Evaluated	42
3.	Ranking of FLIR Features	44
4.	Classification Accuracies of Two FLIR Feature Subsets	44
5.	Average Saliency Values on Doppler Recognition Data for Extended Kalman Filtering and Backpropagation Training Algorithms	48
6.	Absolute Range Features Used for Target Detection Problem	70
7.	Features Used for Multisensor Target Detection	89
8.	Correspondence Feature Normalization	91
9.	Multisensor Target Detection Accuracy Results Compared With Roggemann	92
10.	Multisensor Target Detection Accuracy Results Compared With Statistical Classifiers	92

List of Symbols

Symbol		Page
$\mathbf{x}$	Input (Feature) Vector	8
ω_i	Class i	8
N	Number of Outputs (Classes)	8
$P(\omega_i \mathbf{x})$	Conditional Probability of Class i given Input (Feature) Vector .	8
$\mathbf{X}_i$	Set of Training Vectors for Class i	8
P_i	Number of Training Vectors in Class i	8
M	Number of Inputs (Features)	9
σ	Smoothing Factor of Gaussian Parzen Window	9
$\mathbf{X}$	Set of All Training Vectors	14
$\mathbf{d}$	Desired Output Vector	14
$\mathbf{D}$	Set of Desired Output Vectors	14
E_p	Instantaneous Euclidean Error	14
$\mathbf{z}_p$	Actual Output Vector for p th Input Vector	14
E_T	Total Euclidean Error for Training Set	14
P	Total Number of Training Vectors	14
η	Learning Rate	15
α	Momentum Factor	15
$\mathbf{w}$	Vector of All Free Parameters in Network	17
$\mathbf{R}(t_i)$	Strength Matrix of Observation Noise in Kalman Filter	18
$\mathbf{K}(t_i)$	Kalman Gain Matrix	18
$\mathbf{H}(t_i)$	Gradient Matrix in Kalman Filter of Network	19
$\mathbf{P}(t_i)$	Uncertainty Matrix of State Estimates in Kalman Filter	19
ϵ	Inverse of Initial Value of $\mathbf{P}$ Matrix Diagonal in Kalman Filter .	20
q	Initial Value of Driving Noise in Kalman Filter	20

Symbol		Page
S	Number of Weights and Thresholds in Network	31
P_e	Probability of Error	32
x_i^j	Output of Node i in Layer j	33
w_{ij}^k	Weight from Node i in Layer $k - 1$ to Node j in Layer k	33
θ_j^k	Threshold of Node j in Layer k	33
a_i^k	Activation of Node i in Layer k	33
Λ_j	Saliency of Input j	37
D_j	Set of Sample Points for Input (Feature) j	37
$F(\mathbf{x}, \mathbf{w})$	Output of Multilayer Perceptron	77
$\mathcal{X}_i$	Set of All Possible Input (Feature) Vectors for Class i	77
$\mathcal{X}$	Set of All Possible Input (Feature) Vectors	77
$g_0(\mathbf{x})$	Bayes Optimal Discriminant Function for Two Class Problem	77
$E_s(\mathbf{w})$	Sample Data Euclidean Error	78
$E_a(\mathbf{w})$	Average Euclidean Error	78
$p(\mathbf{x} \omega_i)$	Conditional Probability Density Function of Input Vector Given Class i	79
$F_i(\mathbf{x}, \mathbf{w})$	Output i of Multilayer Perceptron	80

Abstract

The multilayer perceptron was extensively analyzed. A technique for analyzing the multilayer perceptron, the saliency measure, was developed which provides a measure of the importance of inputs. The method was compared to the conventional statistical technique of *best features* and shown to provide similar rankings of the input. Using the saliency measure, it is shown that the multilayer perceptron effectively ignores useless inputs and that whether it is trained using backpropagation or extended Kalman filtering, the weighting of the inputs is the same. The backpropagation training algorithm is shown to be a degenerate version of the extended Kalman filter. The extended Kalman algorithm is shown to outperform the backpropagation method in terms of classification accuracy *versus* training presentations; however, in terms of computational complexity, the backpropagation algorithm is shown to be highly efficient. The multilayer perceptron trained using backpropagation for classification is proved to be a minimum mean squared-error approximation to the Bayes optimal discriminant functions. A simple technique for sensor fusion is shown to provide a statistically significant improvement in performance using absolute range and forward looking infrared imagery for target detection over the single sensor case.

Characterization of Multilayer Perceptrons and their Application to Multisensor Automatic Target Detection

1. Introduction

1.1 Historical Background

The problem of automatic target detection and recognition has been an active area of research in the Air Force for over three decades. Several Air Force missions would benefit greatly from the development of effective target detection and recognition machines such as reconnaissance, location of tactical and strategic relocatable targets, and the strategic defense initiative.

Several attempts at designing such pattern recognition machines have produced good results in the laboratory but have failed miserably in the field. To date there is only one example of a machine which is capable of performing target detection and recognition in a variety of conditions, namely, man. He is the existence proof that this problem is solvable.

Since previous attempts to design a recognizer using statistical and artificial intelligence techniques have failed, a new approach toward machine vision is required. Given that the human visual processing system is the only known effective pattern recognizer, recent research has taken inspiration from the human visual system to design a new class of pattern recognition machines. A variety of biologically-inspired architectures have been proposed and tested for pattern recognition problems. However, a single architecture has dominated the field for the past several years. That architecture is known variously as the Multilayer Perceptron, the Back Propagation Network, or simply a multilayer feed-forward network. Hereafter this architecture shall be referred to as the Multilayer Perceptron.

There are many advantages which these *neural networks* claim such as 1) trainability, 2) intrinsically iterable, 3) fast, 4) *brain-like*. The purpose of this dissertation is to examine the most popular neural network, the multilayer perceptron, and its applicability to multisensor automatic target detection and pattern recognition in general.

The multilayer perceptron has previously been shown to be effective as a classifier for target recognition problems (15) (24); however, a great deal is not known regarding appropriate design, training, and performance. Prior to this dissertation, it was not known how a multilayer perceptron would treat spurious features. It was thought that poor features might degrade the classifiers performance when added to a set of useful features. A related question was how to choose the input features for a multilayer perceptron classifier. Also, the backpropagation training algorithm and its relationship to more conventional techniques was not understood. Additionally, the performance of the multilayer perceptron as a classifier was not understood. The relationship to conventional techniques was unknown. It was thought that these classifiers might provide better performance than traditional statistical classifiers. Finally, the multilayer perceptron had not been used in a multisensor target detection system for tactical targets, and whether or not the multilayer perceptron could perform sensor fusion was not known. All these questions are answered in this dissertation.

1.2 Problem Statement and Scope

The multilayer perceptron will be analyzed to determine its usefulness for automatic target detection and pattern recognition in general. The aspects to be examined are 1) selection of features for recognition, 2) relationship of the backpropagation training algorithm to other well-known methods, 3) relationship to conventional statistical classifiers, and 4) applicability to performing multisensor fusion.

Feature Selection. The selection of features for pattern recognition is critical for good recognition performance (6:15-17). A novel measure will be introduced which yields a ranking of the input features for a multilayer perceptron. This technique allows

pruning of the input features required for recognition (27). The desirability of minimizing the number of features is well-known (7:95) (9) (6:187) and will be discussed further in Chapter III.

Understanding Backpropagation. A new result showing that the popular backpropagation training algorithm is actually a degenerate form of the extended Kalman filter algorithm for setting the weights in a multilayer perceptron will be presented in Chapter IV. Understanding the method of determining the weights in a multilayer perceptron is crucial since all the information about the problem is contained in the connectivity of the weights and their values. An understanding of how backpropagation sets the weights can lead to a better understanding of its strengths and weaknesses. This relationship is also presented in (30).

Bayesian Relationship. It will be shown that the multilayer perceptron when trained to perform classification using the backpropagation algorithm leads to a classifier which approximates the Bayesian optimal discriminant functions. This is a previously unknown result, and it shows that the multilayer perceptron classifier is closely related to conventional statistical classifiers which have been used in pattern recognition for over twenty years and whose performance is well understood (29).

Sensor Fusion. A simple but effective method for performing feature level sensor fusion will be presented which provides statistically significant improvements in target detection. Since multiple sensors will be required to perform many of the Air Force missions (2), it is essential that a pattern recognizer be extensible to multiple sensors.

Baseline Comparison. The work of Roggemann (19) will be used as a baseline in this effort. Roggemann used a conventional Bayesian approach to target detection using multiple sensors. The use of the same database as Roggemann will allow comparison of the neural network approach with a traditional statistical method. However, other databases will be used when appropriate.

1.3 Dissertation Organization

The following chapter will review the basic concepts of target and pattern recognition, the multilayer perceptron architecture and training, and test methodologies. The succeeding chapters are organized in a bottom up fashion on the classifier architecture. In Chapter III, the selection of inputs to the classifier will be considered. The selection of features for classification using a multilayer perceptron will be derived. After considering the inputs to the classifier, Chapter IV examines the training of a multilayer perceptron. The backpropagation training algorithm will be analyzed in the context of the extended Kalman filtering algorithm. The fully trained multilayer perceptron classifier will be considered in Chapter V. The relationship between the multilayer perceptron trained with backpropagation and the Bayes optimal discriminant functions will be exposed. Finally, the application of the multilayer perceptron classifier to an Air Force problem, multisensor fusion, will be examined in Chapter VI. A simple but effective technique for performing multisensor fusion will be developed. Chapter VII will discuss recommendations for future research and the conclusions which can be drawn from this work.

II. Background Material

The purpose of this chapter is to review the essential background material necessary to understanding the approaches and results of this dissertation. The first section will review the basic concepts of target and pattern recognition. The two following sections will present the multilayer perceptron architecture and two important training algorithms for it. The last section will review the test methodologies employed in evaluating the training algorithms and the classifiers.

2.1 Basic Concepts of Target and Pattern Recognition

The process of target recognition in its basic form consists of three separate stages: segmentation, feature extraction and classification (see Figure 1). Each of these stages will be reviewed to introduce the essential concepts and nomenclature.

The first stage in target recognition is segmentation. Segmentation is the process of assigning a label to each pixel in an image. For example, the set of labels might be {background,target} or {building,river,grass,...}. In this dissertation, segmentation will refer to assigning labels from the set {background, target}. The purpose of segmentation is to reduce the number of pixels to process further using the feature extraction stage. In most cases, the number of pixels in an image is far too great to process in detail with current computer technology. Another reason for segmentation is to identify multiple targets in an image and their locations. After the image has been segmented into potential targets and background, the contiguous groups of potential target pixels called blobs are further processed by the feature extraction stage. This dissertation does not address the segmentation stage. The segmentation algorithms used were those developed by Roggemann (19) and Ruck (24).

The feature extraction stage computes a number of features for each blob detected in the source image. Some example features are length-width ratio, average temperature (in infrared), complexity (ratio of border pixels to total blob pixels). The features computed

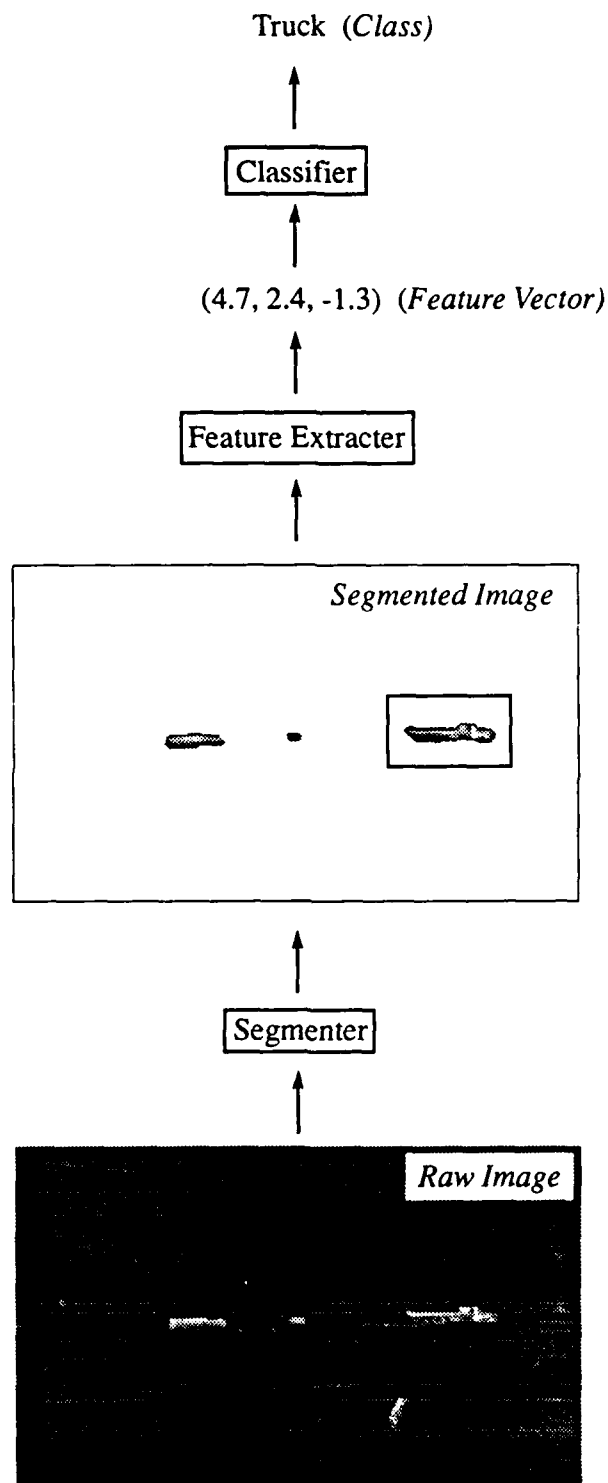


Figure 1. Target Recognition Architecture

for a blob are concatenated into a vector of numbers. This *feature vector* is then sent to the classification stage. These feature vectors lie in the *feature space*. This dissertation will address the issue of selecting the appropriate features to be computed in this stage using a novel technique which ranks the *saliency* of the features.

The final stage in the basic target recognition architecture is the classifier which assigns a label to an input feature vector. The types of labels are {target, non-target} for target *detection* and {tank, truck, jeep, ...} for target *recognition*. This dissertation will extensively evaluate the use of a multilayer perceptron for classification and compare both empirically and theoretically the multilayer perceptron performance to traditional statistical classifiers.

Two popular statistical classifiers will be compared to the multilayer perceptron. The non-parametric Bayesian classifier using both histograms and Parzen windows to estimate the probability density functions and the k -nearest neighbor algorithm were chosen for comparison. Roggemann used the histogram technique for his non-parametric Bayesian classifier. In order to use this method, he assumed the input features were conditionally independent (19:91-92). The non-parametric Bayesian classifier with Parzen windows did not make this independence assumption. These classifiers were chosen because they are *non-parametric*. A non-parametric classifier does not make any assumptions regarding the probability density function controlling the generation of feature vectors. The vector produced by the feature extraction stage is a random variable for a number of reasons. The following conditions cause unpredictable variability in the feature vectors extracted for a given class of objects: 1) noise in the source image due to environmental conditions or sensor limitations, 2) imperfect image segmentation, 3) out-of-plane rotation of the object (a problem when the features extracted are sensitive to such rotations which is usually the case). Thus, the feature vector is a random variable. If the conditions causing the variability are sufficiently understood, it might be possible to determine the type of probability density function that governs the feature vectors. In general, this cannot be determined; hence, either some distribution is assumed, such as Gaussian, or none is. If

the form of the distribution is assumed, then the only remaining task is to determine the parameters of the distribution. In the Gaussian case, these are the mean and variance. A classifier predicated on the assumption of a form of the distribution function is termed a parametric classifier. In target recognition, non-parametric classifiers are favored because it is often impossible to determine the form of the distribution function and the Gaussian assumption is often invalid.

The non-parametric Bayesian classifier using Parzen windows was one of the two statistical classifiers used in this dissertation to provide a baseline comparison with the multilayer perceptron. A Bayesian classifier is optimal when the distributions are known in the sense of minimizing either the probability of error or some other cost function when different type errors have different associated costs (7:17). Let $\mathbf{x}$ denote the M -dimensional vector to be classified and let $\omega_i, i = 1, \dots, N$ represent the N classes. The Bayesian decision rule is

$$\text{Decide } \omega_i \text{ if } P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}) \text{ for all } j \neq i$$

where $P(\omega_i|\mathbf{x})$ is the conditional probability of class ω_i given input vector $\mathbf{x}$. Since the conditional probability distributions are not known, this classifier is not optimal; nonetheless, it is widely used. The classifier can use Parzen windows to estimate the class conditional density functions of the features from a set of *training data*. The training data consists of a set of feature vectors whose true classification is known. A Parzen window estimate of the density function places a window function centered at each of the points in the feature space where a training vector is present for a given class (7:88-95). Then the window functions are summed together. Let $\mathbf{X}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \dots, \mathbf{x}_{i,P_i}\}$ be the set of training vectors for class ω_i where P_i is the number of training vectors from class ω_i . Then the Parzen window estimate of $p(\mathbf{x}|\omega_i)$ using Gaussian windows is given by

$$\hat{p}(\mathbf{x}|\omega_i) \equiv \frac{1}{P_i} \cdot \left(\frac{P_i}{2\pi}\right)^{\frac{M}{2}} \cdot \frac{1}{\sigma^M} \sum_{j=1}^{P_i} \exp\left(-\frac{(\mathbf{x} - \mathbf{x}_{i,j})^T(\mathbf{x} - \mathbf{x}_{i,j})}{\frac{2\sigma^2}{P_i}}\right)$$

where $\hat{p}(\mathbf{x}|\omega_i)$ is the Parzen window estimate of the true density function, M is the dimensionality of the feature vector, $\mathbf{x}$, and σ is the smoothing factor or window width. The width of the window function is proportional to the smoothing factor, σ . Low values of σ result in less averaging of the individual data points while high values of σ cause many data points to be averaged together to obtain the estimate of the probability density function at a given point. It has been shown that the above approximation to the conditional density function converges in the mean-square metric to the true density function under certain non-restrictive conditions (7:89-90). The resulting density function can then be easily converted to an *a posteriori* probability using Bayes' Rule (23:51-57) and the *a priori* probabilities of occurrence of the classes. In the experiments where the non-parametric Bayesian classifier with Parzen windows is used, a range of smoothing factors is tried and the best smoothing factor in that range is listed. The smoothing factors used are from 0.1 to 1.5 by increments of 0.2.

The other non-parametric classifier compared to the multilayer perceptron is the k -nearest neighbor algorithm (7:103-105) (13:120). This algorithm also requires no assumptions regarding the underlying distributions controlling the feature vectors. The algorithm computes the distance between an unclassified feature vector and all the feature vectors in the training set (known as exemplars). These distances are then ranked from smallest to largest. The class assigned to the unknown vector is the class with majority representation in the k nearest exemplars. The parameter k is normally chosen to be odd so that for the two class decision case, there is always a clear majority. In the multiclass case, though, k odd does not guarantee a single class will have a majority. In this event, the class assigned is selected randomly from the tying classes. It should be noted that this algorithm is also performing a type of density function estimation (7:104), so it is closely related to the non-parametric Bayesian classifier using Parzen windows. When the k -nearest neighbor algorithm is used in the following experiments, k is set to all odd values between 1 and 9, inclusive. The value of k yielding the best classification accuracy from these values is reported.

These two common statistical classifiers in addition to Roggemann's approach will be compared to the multilayer perceptron classifier. The next two sections will review the development of the multilayer perceptron architecture and its training algorithms.

2.2 *The Multilayer Perceptron Architecture*

The multilayer perceptron architecture is an outgrowth of the Perceptron which was first studied by Rosenblatt in the mid 1950s (22). The term *Perceptron* was coined by Rosenblatt to cover a variety of architectures designed by him while trying to model the human brain. Today, the use of the term perceptron generally refers to a single node. The term multilayer perceptron means more than one layer of nodes with nodes fully interconnected between layers. This dissertation will deal strictly with multilayer perceptrons.

Figure 2 shows the structure of a multilayer perceptron. Each of the circles in the diagram represents a *node* which performs a weighted sum of the inputs and applies a nonlinearity (see Figure 3). The network shown has two hidden layers, that is, two layers which are neither input nor output. The network has M inputs, H^1 nodes in the first hidden layer, H^2 nodes in the second hidden layer, and N outputs. A short hand notation for describing this architecture is $M - H^1 - H^2 - N$. A single hidden layer network would be simply $M - H^1 - N$. The superscripts are used to indicate the layer the variable is associated with. The layers are numbered from the first layer of nodes performing the nonlinear function of the weighted sum of the inputs. In other words, the inputs to the network are not counted as a layer. The first hidden layer of the network is Layer 1. A two hidden layer network is also called a three layer network because there are three layers of weights, and a single hidden layer network is often called a two layer network. Other nonlinearities can be used. The most popular one, used in this research, is shown in Figure 3.

A remaining problem with application of multilayer perceptrons to various problems lies in determination of the architecture. To date, no techniques have been accepted which

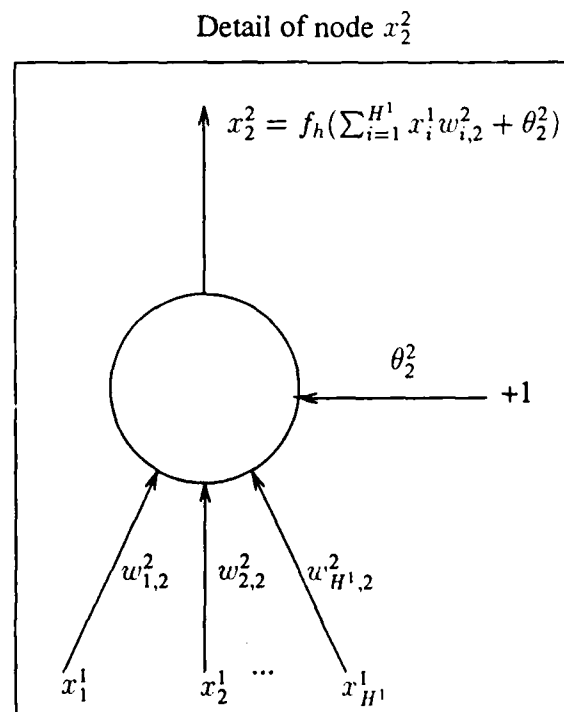
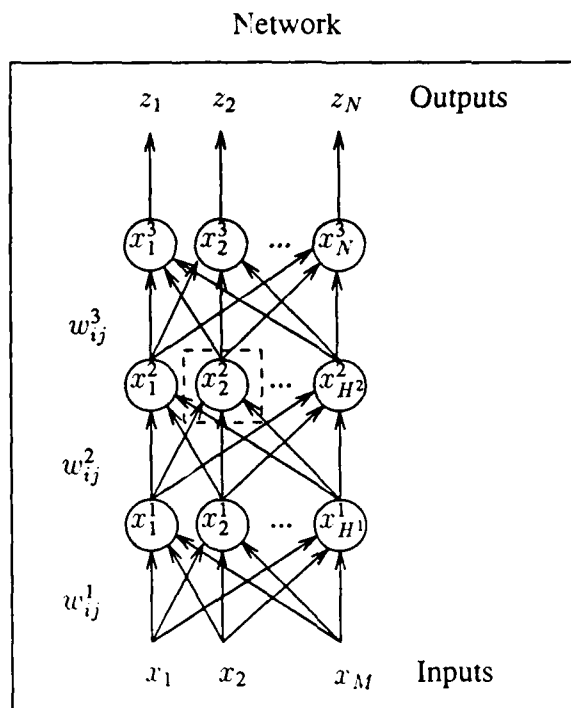


Figure 2. Multilayer Perceptron Architecture

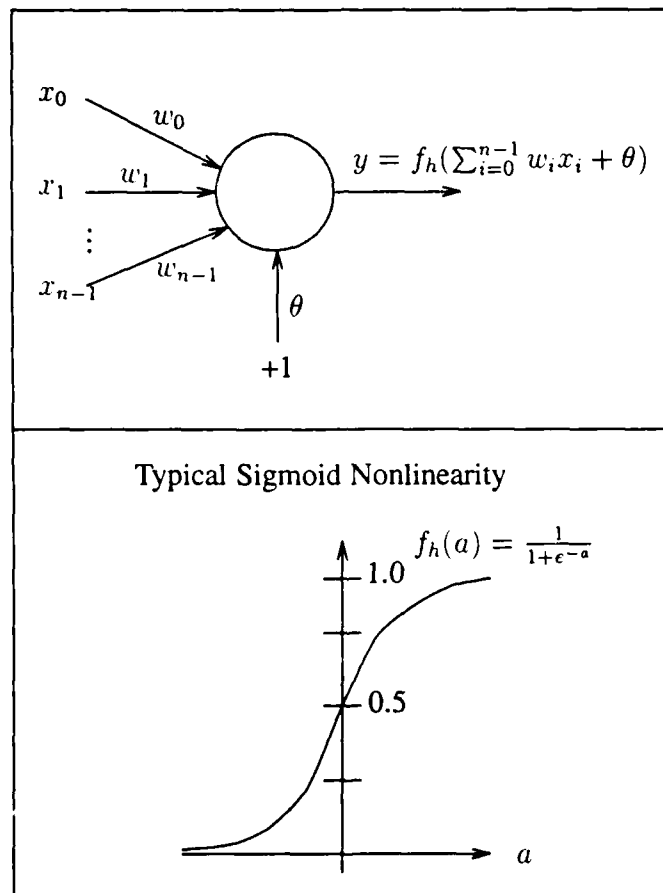


Figure 3. Detail of a Node in the Multilayer Perceptron

specify the number of hidden layers or nodes in each hidden layer based on the training data or other *a priori* information. Techniques have been developed recently which may allow automatically pruning the size of a network (4) (33). The combination of these techniques with some basic rules of thumb for sizing a network should provide effective methods for automatically architecting a multilayer perceptron. This dissertation will not deal with the automatic sizing issue; however, procedures and guidelines for sizing a network are provided in Appendix A.

In order to make such networks useful, a method for determining the interconnection weights is required. Algorithms for setting the weights are called *learning* rules or *training* algorithms and will be discussed in the next section.

2.3 Multilayer Perceptron Training Algorithms

Many researchers worked in the area of adaptive systems during the 1960s using perceptrons. The single node perceptron was a popular architecture for which the learning rule had been shown to converge when a solution exists (13:82-87). However, in 1969 Minsky and Papert published a now famous book which showed that a single node perceptron could not perform a simple Boolean function, namely, the exclusive-OR problem (12). This book discouraged many researchers from further work in the area. From that time until the early 1980s neural network research was not vigorously pursued. It can be easily shown that a simple two layer network can perform the exclusive-OR problem (31:64). In fact, Rosenblatt had developed some algorithms which could train multilayer networks (22) although convergence of these training procedures could not be proved. The lack of an effective training rule for multilayer networks has been cited by many researchers as the primary reason for the demise of neural network research in the 1970s (12:231-232). Effective training rules now exist for the multilayer perceptron and will be discussed in the next two sections.

2.3.1 *Backpropagation* There are now many training algorithms available for multilayer perceptrons. Some algorithms have been developed for multilayer networks where the nodes have hard limiter nonlinearities (e.g., MRL by Widrow (37)). However, the most popular architecture uses sigmoidal nonlinearities on the nodes. The sigmoid is differentiable which makes possible weight update rules based on the gradient of the *error* with respect to the weights in the network. The most popular rule for training the weights in a multilayer perceptron is the backpropagation training algorithm. This technique was made popular by Rumelhart *et al* in their book *Parallel Distributed Processing* (31) in 1986 although it was first derived by Werbos in 1974 (35) and rederived by Parker in 1982 (14). Recently, it has been suggested by White (36) that the stochastic approximation techniques developed by Robbins and Monro (16) in 1951 subsume backpropagation.

Backpropagation is a gradient descent method for training the weights in a multilayer perceptron. For a given problem, there is a set of training vectors $\mathbf{X}$ such that for every vector $\mathbf{x} \in \mathbf{X}$ there is an associated desired output vector $\mathbf{d} \in \mathbf{D}$ where $\mathbf{D}$ is the set of desired outputs associated with the training vectors in $\mathbf{X}$. Let the instantaneous error E_p be defined as

$$\begin{aligned} E_p &= \frac{1}{2}(\mathbf{d}_p - \mathbf{z}_p)^T(\mathbf{d}_p - \mathbf{z}_p) \\ &= \frac{1}{2} \sum_{k=1}^N (d_{k,p} - z_{k,p})^2 \end{aligned} \quad (1)$$

where $d_{k,p}$ is the k th component of the p th desired output vector, $\mathbf{d}_p$, and $z_{k,p}$ is the k th component of the actual output vector, $\mathbf{z}_p$, when the p th training exemplar, $\mathbf{x}_p$, is input to the multilayer perceptron. Let the total error E_T be defined as follows:

$$E_T = \sum_{p=1}^P E_p$$

where P is the cardinality of $\mathbf{X}$, in other words, the total number of training vectors. Note that E_T is a function of both the training set and the weights in the network. The

backpropagation learning rule is defined as follows:

$$w^+ = w^- - \eta \frac{\partial E_p}{\partial w}(w^-) + \alpha(w^- - w^{--}) \quad (2)$$

where η , the learning rate, is some small positive number, α , the momentum factor, is also a small positive number, and w represents any single weight in the network. In the above equation, w^+ indicates the value of the weight after update, w^- is the current weight value, and w^{--} is the previous weight. When the momentum term is used ($\alpha \neq 0$), the training rule is called the Momentum Method; otherwise, it is the Backpropagation Method. The algorithm (Equation 2 with $\alpha = 0$) is often termed *instantaneous backpropagation* because it computes the gradient based on a single training vector. Another variation is *batch backpropagation* which computes the weight update using the gradient based on the total error E_T . Informal experiments indicate that the instantaneous method works better than the batch mode; hence, when training results are presented, instantaneous backpropagation is the method used.

To use the multilayer perceptron for target recognition, the input to the network is the feature vector produced by the feature extraction stage, and the desired output is some vector that indicates the class of the input. Often the input vectors must be normalized in some fashion, so that no one feature dominates the classification process.

The following normalization algorithm has proved to be highly effective on a wide variety of recognition data. A normalization transformation is first computed from the training set of feature vectors. For each feature, the mean and standard deviation are computed over the training set irrespective of class. Then each feature of each vector is normalized by subtracting the mean and dividing by the standard deviation of the feature. Let μ_i be the mean of feature i over the training set and σ_i be the standard deviation. Note that this σ_i is different from the smoothing factor or window width, σ , associated with the Parzen windows in the non-parametric Bayesian classifier. Then for every vector in both

the training set and test set, the normalized i th component is

$$x'_i = \frac{x_i - \mu_i}{\sigma_i} \quad (3)$$

where x'_i is the normalized value. This procedure will be called Gaussian Normalization.

Generally, the desired output is a block code such that there is one output for each potential class of inputs. When a vector of a given class is inputted, the desired output of the node corresponding to that class is high and all other nodes are low. In Chapter V, it will be shown that this particular choice of desired output allows the outputs of the network to be interpreted as *a posteriori* probabilities. Other researchers have chosen to encode the output, for example a binary code of high and low outputs so that 2^N classes can be represented by N outputs (32). Additionally, it has been suggested that pseudo-random output vectors or Hadamard like codes be used to insure that each output has equal likelihood of being high as well as low in order to speed-up learning (17).

2.3.2 Extended Kalman Filtering

Training Algorithm Another significant method for training the weights in a multilayer perceptron is the extended Kalman filter algorithm. Singhal *et al* suggested the idea of using an extended Kalman filter to train a multilayer perceptron (34). Basically, a Kalman filter attempts to estimate the state of a system which can be modeled as a linear system driven by additive white Gaussian noise and where the measurements available are linear combinations of the system states corrupted by additive white Gaussian noise (10). The weights of the multilayer perceptron are the states which the Kalman filter attempts to estimate, and the output of the network is the measurement used by the Kalman filter. Since the multilayer perceptron is not a linear system, the usual Kalman filter cannot be used. However, if the model of the system is linearized, then the Kalman filter algorithm can be applied. The resulting filter is called an *extended Kalman filter*.

The Kalman filter approach to training a multilayer perceptron considers the optimal weights in the network to represent the state of the system which is to be estimated.

The optimal weights are those that minimize the total error, E_T . Since the underlying distributions which control the feature vector, $\mathbf{x}$, are assumed to be time-invariant, the state of the network does not change. Hence, this is a static estimation problem. If the goal were to track some changing distributions, then a dynamics model would be required for the state estimation. A possible system model is

$$\dot{\mathbf{w}}(t) = \mathbf{0}$$

where $\mathbf{w}$ is a vector consisting of all the weights and thresholds in the network. (Note that this notation differs from the usual Kalman filter literature. Compatibility with that literature has been maintained wherever possible. In the Kalman filter literature, the states being estimated are represented by $\mathbf{x}(t)$ and the driving noise is $\mathbf{w}(t)$. Here the states being estimated are $\mathbf{w}(t)$, the network inputs are $\mathbf{x}(t)$, and the driving noise is $\mathbf{n}(t)$.) However, problems can arise from using this system model in practice due to the operation of the Kalman filter. As the Kalman filter obtains more observations of the system behavior, its estimated error in the states approaches zero. If some change occurs in the system, the filter will not track the change because it believes its estimates are correct. In the jargon of the Kalman filter literature, the filter gain goes to zero. Thus, it is often advantageous to add some noise to the system model which prevents the gain from going to zero and forces the filter to continually adjust the state estimates. In this case the system model is (11:44)

$$\dot{\mathbf{w}}(t) = \mathbf{G}(t)\mathbf{n}(t)$$

where $\mathbf{n}(t)$ is a zero-mean white Gaussian noise process of arbitrary dimension n and $\mathbf{G}(t)$ is a matrix of dimension S , the number of weights and thresholds in the network, by n that provides flexibility in characterizing the driving noise in the system model. The covariance kernel of $\mathbf{n}(t)$ is

$$E\{\mathbf{n}(t)\mathbf{n}^T(t + \tau)\} = \mathbf{Q}(t)\delta(t)$$

where $Q(t)$ is termed the strength matrix of the driving noise, $n(t)$. This notation differs somewhat from the traditional Kalman filter literature where the states being estimated are $x(t)$ and the noise vector is $w(t)$. Initially, several different system models were tested to determine which provided the best performance in training the multilayer perceptron with the extended Kalman filter algorithm. These variations will be discussed after the Kalman technique has been presented.

In the extended Kalman filter algorithm, the measurements of the system are assumed to be some nonlinear function of the states corrupted by zero-mean white Gaussian noise. Thus the observations of the system are modeled as follows (11:44):

$$z(t_i) = h[w(t_i), t_i] + v(t_i)$$

where $z(t_i)$ is the output of the system at observation time t_i , $h[\cdot, \cdot]$ is the nonlinear function mapping the states to the observations, and $v(t_i)$ is a zero-mean white Gaussian noise sequence. The covariance kernel of $v(t_i)$ is

$$E\{v(t_i)v^T(t_j)\} = \begin{cases} R(t_i) & t_i = t_j \\ 0 & t_i \neq t_j \end{cases}$$

where $R(t_i)$ is called the strength matrix of the observation noise, $v(t_i)$.

Using the preceding models for the system and the observations, a method for updating the estimate of the system state can be derived. The update equations are (11:44)

$$\hat{w}(t_i^+) = \hat{w}(t_i^-) + K(t_i)[d(t_i) - z(t_i)] \quad (4)$$

$$K(t_i) = P(t_i^-)H^T(t_i)[H(t_i)P(t_i^-)H^T(t_i) + R(t_i)]^{-1} \quad (5)$$

$$P(t_i^+) = P(t_i^-) - K(t_i)H(t_i)P(t_i^-) \quad (6)$$

where $\hat{w}(t_i)$ is the Kalman filter estimate of the system state at time t_i , $K(t_i)$ is called the Kalman gain and is an S by N matrix (N is the number of outputs), $d(t_i)$ is the desired

output, $\mathbf{H}(t_i)$ is the gradient matrix which results from linearizing the network, and $\mathbf{P}(t_i)$ is a matrix representing the uncertainty in the estimates of the states of the system. The symbol t_i^- represents the time just prior to performing the updates, and t_i^+ represents the time just after the updates. The factor, $[\mathbf{d}(t_i) - \mathbf{z}(t_i)]$, in Equation 4 is termed the residual because it represents the difference between what was expected and the actual observation. This difference is what drives the correction to the weights in the network. The entries in the gradient matrix are (11:44)

$$H_{ij} = \frac{\partial z_i}{\partial w_j} \quad (7)$$

A usual part of the Kalman filter is the propagation phase. For a dynamic system which is changing between the times when observations are taken, the estimate of the state of the system at the new time, t_{i+1} , must be based on both the estimate of the system at the previous observation time, t_i , and the dynamics of the system. For a static system model, which is being used here, the estimate of $\mathbf{w}(t_{i+1})$ is the same as at the previous time. When the system is driven by white Gaussian noise, the uncertainty in the estimate of the states increases in proportion to the noise. Hence, the propagation of the system state from t_i to t_{i+1} is given by integrating the following equations (11:44)

$$\dot{\hat{\mathbf{w}}}(t|t_i) = \mathbf{0} \quad (8)$$

$$\dot{\mathbf{P}}(t|t_i) = \mathbf{G}(t)\mathbf{Q}(t)\mathbf{G}^T(t) \quad (9)$$

In the above equations, the dot notation indicates derivative with respect to time and the $(t|t_i)$ means the estimate of the function at time t given its value at time t_i . Assuming that $\mathbf{G}(t) = \mathbf{I}$ ($\mathbf{I}$ is the S by S identity matrix) and $\mathbf{Q}(t) = \mathbf{I}q(t)$ where $q(t)$ is slowly changing in relation to the propagation time, the integrated propagation equations are

$$\hat{\mathbf{w}}(t_{i+1}^-) = \hat{\mathbf{w}}(t_i^+) \quad (10)$$

$$\mathbf{P}(t_{i+1}^-) = \mathbf{P}(t_i^+) + \mathbf{I}q(t) \quad (11)$$

When actually implementing the extended Kalman filter for determining the weights, initial conditions must be specified, namely, $\mathbf{w}(t_0)$ and $\mathbf{P}(t_0)$. In the absence of any *a priori* information, the initial state vector can be set randomly, and the $\mathbf{P}$ matrix can be set to some multiple of the identity matrix. The following initial conditions were used in this implementation of the Kalman algorithm

$$\begin{aligned}\mathbf{P}(t_0) &= \frac{1}{\epsilon} \mathbf{I} \\ \mathbf{w}(t_0) &= N[0, \mathbf{P}(t_0)]\end{aligned}$$

where ϵ is some small positive number and $N[0, \mathbf{P}(t_0)]$ denotes a zero-mean Gaussian distribution with covariance matrix $\mathbf{P}(t_0)$.

As stated earlier, several different system models were evaluated for training a multilayer perceptron using the extended Kalman algorithm. The models differed in whether or not driving noise was present and if so how it changed with time. The following models were tested

1. No driving noise: $q(t) = 0$.
2. Fixed driving noise: $q(t) = q$ where q is a fixed small positive number.
3. Decaying driving noise: $q(t) = q \exp(-k/50)$ where k is the sweep number, that is, the number of passes through the complete training set at time t in the training.

The performance of these three models for training is shown in Figures 4 - 6 on the Exclusive-OR (XOR) problem (this problem is fully defined in Section 4.4 on p. 55). For comparison purposes, the performance of backpropagation and momentum is also shown. The network trained was a single hidden layer network with four nodes in the hidden layer. For each of the Kalman trained networks, the $\mathbf{P}$ matrix was initialized using $\epsilon = 1.0$. When driving noise (Q-noise) was used, q was set to 0.05. A learning rate of $\eta = 0.3$ was used for backpropagation; and when momentum was used, it was set to $\alpha = 0.8$. Each network started with the same random weights. From the figures, it can be seen that the algorithm which used decaying driving noise is far superior to the other Kalman system

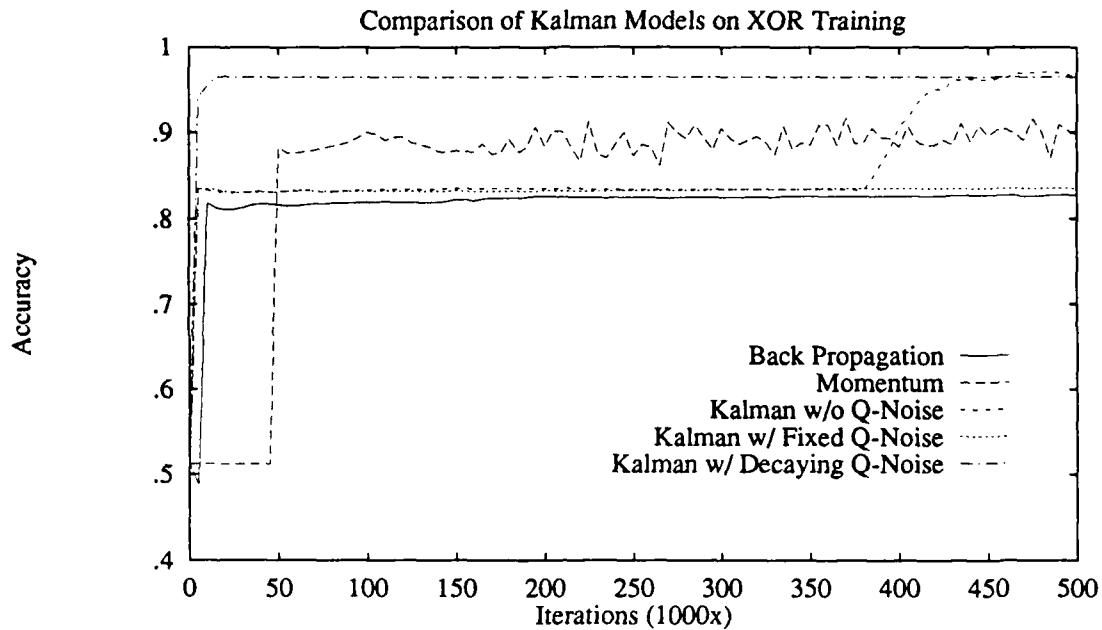


Figure 4. Comparative Performance of Extended Kalman Training on the Exclusive-OR Problem using Different Driving Noise Models up to 500,000 Iterations

models. Different initial conditions would yield somewhat different results; however, for all the runs observed the decaying driving noise model yielded noticeably superior results. Therefore, it was decided to use the decaying driving noise model for all Kalman training of multilayer perceptrons in this research. In fact, even the parameters were fixed at $\epsilon = 1.0$ and $q = 0.05$ for all training runs.

2.4 Test Methodologies

This section will review the test methodologies used to evaluate the performance of the classifiers designed. First, the methods of partitioning the database between training and test will be explained followed by a review of the techniques of confidence intervals.

2.4.1 Database Partitioning Two methods were used for partitioning the data between training and testing subsets. The first method is called the *hold out* technique. The hold out technique randomly selects a fraction of the database for use in testing. The

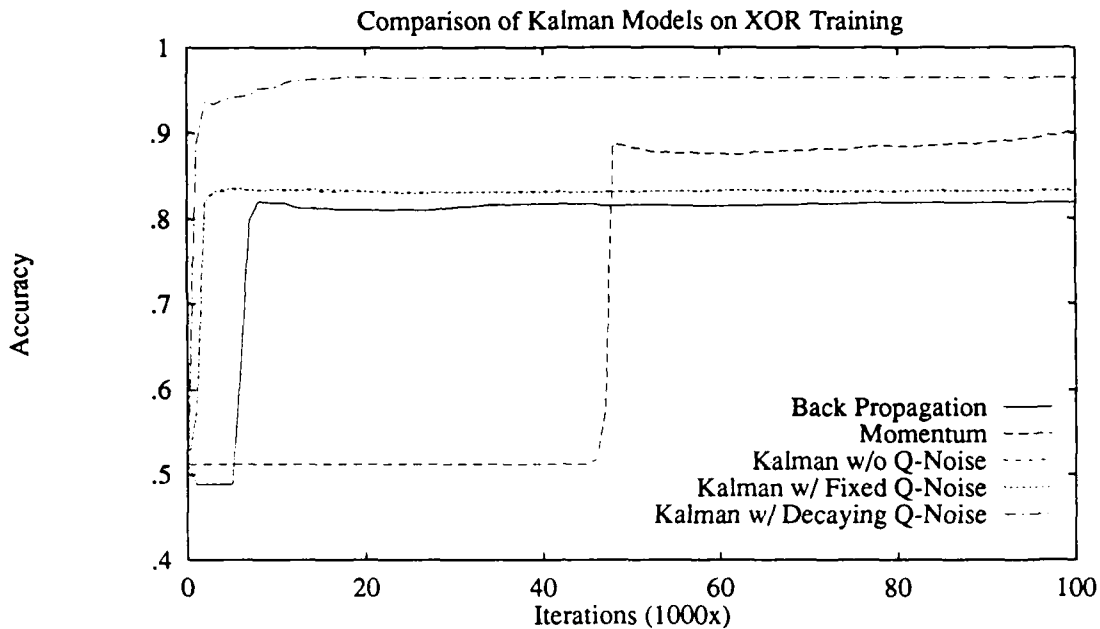


Figure 5. Comparative Performance of Extended Kalman Training on the Exclusive-OR Problem using Different Driving Noise Models up to 100,000 Iterations

remaining feature vectors in the database are used in designing the classifier. After the classifier has been designed, the vectors selected for testing are presented to the classifier and its responses are recorded, but the classifier is **not** redesigned using the testing data. The hold out method does not efficiently use the available data since presumably a better classifier could be designed using all the vectors in the database. In fact, the hold out method in general gives pessimistic results regarding classifier performance (6:355).

A second method for partitioning the data is the *hold one out* method (6:356-357). In the hold one out method, the classifier is designed using all the data except for one vector which is then tested on the classifier. Then another vector is held back and the classifier is redesigned and tested. This process is repeated for every vector in the database. The error rate is then computed from the number of vectors which were misclassified. For a multilayer perceptron, this means performing the training of a network for every vector in the database. While computationally intensive for large databases, this method

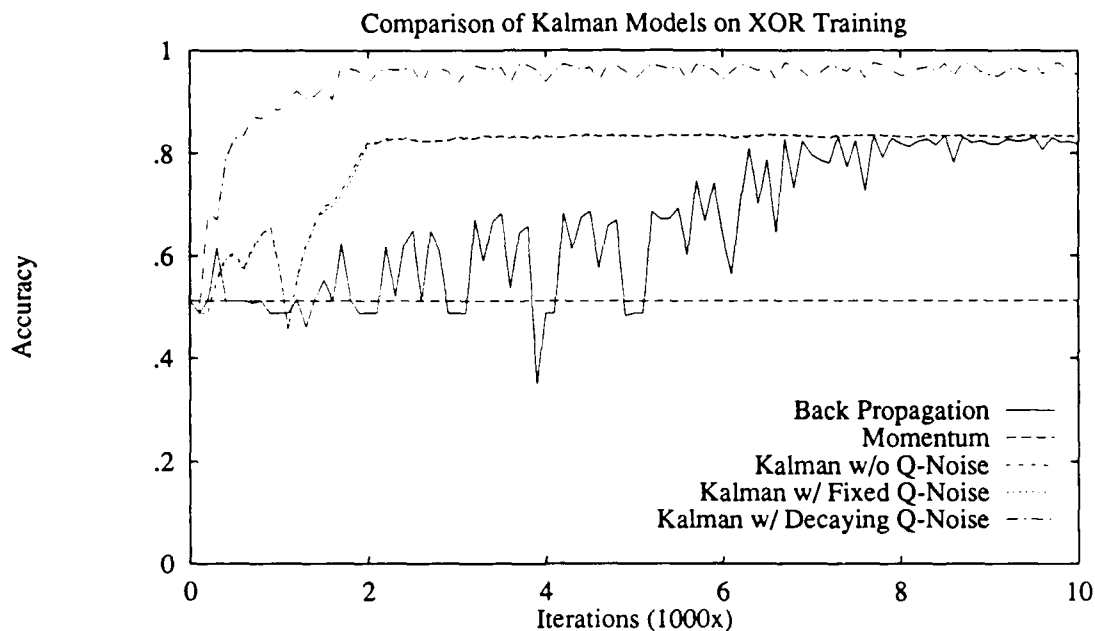


Figure 6. Comparative Performance of Extended Kalman Training on the Exclusive-OR Problem using Different Driving Noise Models up to 10,000 Iterations

gives better estimates of the error rate (or classification accuracy). This method uses the available data more efficiently than the hold out technique. Also, the hold one out method provides unbiased estimates of the error rate (or classification accuracy) (6:356). The hold one out method is preferred where practical; however, the requirement to train a multilayer perceptron once for each vector in the database can be computationally prohibitive.

2.4.2 Confidence Interval Techniques The method of confidence intervals provides information as to the reliability of the performance figures reported. Commonly, figures of merit are reported as a single number. For example, one might say the accuracy of a classifier is 85 percent. This is called a point estimate. Obviously, since the classifier is tested on a finite data set, the true accuracy is not 85 percent; rather it is probably in some interval centered about the point estimate. A confidence interval for the accuracy of a classifier specifies the probability that the true accuracy lies within an interval. In the preceding example, the 95 percent confidence interval might be 80 to 90 percent meaning

that with 95 percent probability the true accuracy of the classifier is between 80 and 90 percent where *true accuracy* is the accuracy when tested on all possible inputs. This type of confidence interval will be termed the *Classifier Confidence Interval*.

A second type of confidence interval will also be used in this dissertation which will be termed the *Monte Carlo Confidence Interval*. Suppose some experiment is performed, such as training a multilayer perceptron, and that some figure of merit produced by the experiment is a random variable. In the multilayer perceptron, it might be the total error, E_T , after training. The total error is a random variable because its value is influenced by the initial weights that are randomly chosen and the order of presentation of training vectors which is random. Since the figure of merit is random, it is desirable to quantify its variability. Several experiments are performed and the figure of merit is recorded for each experiment. For the multilayer perceptron, several training sessions are performed on the same architecture network using different random weights and order of presentation of training vectors for each session. It would be possible to produce a point estimate for the figure of merit by simply averaging its value for many experiments, but this estimate would not give any indication of its reliability. Again, a confidence interval on the average value of the figure of merit will provide an indication of its reliability. In the multilayer perceptron case, suppose the average total error is 0.02 for 100 training sessions and the 95 percent confidence interval is 0.01 to 0.03. (A slight digression is required here. Actually, the total error, E_T would probably not be used since it is dependent on the size of the training database. As the number of vectors in the database increases so will, necessarily the total error; hence, a better figure of merit would be total error per training vector, E_T/P . This value is independent of the size of the database and gives a better indication of how well the network has learned the database.) The confidence interval for the average figure of merit will usually shrink as the number of experiments is performed. For the multilayer perceptron, the confidence interval for the total average error indicates how well the network can learn the database on average. Note that the *average* total error is being characterized not the total error for one single training session. Note that the Monte

Carlo technique is not restricted to classifiers but can be applied to any experiment where the outcome is a random variable. The following will review the mathematics applicable to both of these confidence interval techniques.

Consider first the Classifier Confidence Interval technique. Suppose a classifier is designed and then tested with P test vectors. Let the true error rate, E , be the error rate that would be measured by testing every possible input vector. Note that E represents an error rate while E_T is the total squared error difference between the desired output and the actual output of the network. Supposing also that L vectors are misclassified, then true error rate, E , for the classifier might be estimated using

$$\hat{E} = \frac{L}{P} \quad (12)$$

Now when the number of test vectors, P , is large, there is greater confidence that the measured error, $\hat{E}$, is closer to the true error rate, E than when P is small. Assuming that the test set is a random sample of the feature vector distribution, then the measured error, $\hat{E}$, is a random variable. An expression for the measured error rate in terms of the sum of Bernoulli random variables can be written as (6:346-347)

$$\hat{E} = \frac{1}{P} \sum_{i=1}^P \zeta(\mathbf{x}_i)$$

where

$$\zeta(\mathbf{x}_i) = \begin{cases} 0 & \text{if } \mathbf{x}_i \text{ correctly classified} \\ 1 & \text{if } \mathbf{x}_i \text{ incorrectly classified} \end{cases}$$

To a high degree, $\zeta(\mathbf{x})$ can be approximated as a Bernoulli random variable which makes $\hat{E}$ a binomial random variable. Assuming that $\hat{E}$ is a binomial random variable, it can be shown that Equation 12 gives the maximum likelihood estimate for E , the true error rate. Also, the mean of $\hat{E}$ is E and the variance is $E(1 - E)/P$ (6:347). Thus the estimator is unbiased and consistent. Using the Gaussian approximation to the binomial distribution

of $\hat{E}$ and approximating the variance by $\hat{E}(1 - \hat{E})/P$, it can be shown that

$$\text{Prob}\{\hat{E} - z\hat{\sigma}\{\hat{E}\} < E < \hat{E} + z\hat{\sigma}\{\hat{E}\}\} \approx \gamma$$

where $\hat{\sigma}\{\hat{E}\}$ is the square root of the estimated variance of $\hat{E}$, and z and γ are controlled by the standard Gaussian distribution on $\hat{E}$. For a confidence level of 95%, $z = 1.96$ and $\gamma = 0.95$ while for a confidence level of 90%, $z = 1.64$ and $\gamma = 0.90$.

The Monte Carlo Confidence Interval technique has a slightly different basis than the Classifier Confidence Interval technique. The goal here is to characterize the performance of a multilayer perceptron as a function of the initialization of the weights and the order of presentation of training vectors. In this case, the error rate measured from a single training session (or run) on the multilayer perceptron is considered a random variable. Let $\mathcal{E}$ represent the measured error rate for a trained multilayer perceptron. Thus, $\mathcal{E}$ is a random variable depending on the initial weights and presentation order for a network. The distribution of $\mathcal{E}$ is unknown. Let $\mathcal{E}_i$, $i = 1, \dots, n$, be the realization of $\mathcal{E}$ for the i th training session. It is assumed that the $\mathcal{E}_i$ are independent and identically distributed since the weights were randomly initialized using a different and randomly chosen seed for the pseudorandom number generator and likewise the presentation order was random and different for each run. Also, the distribution of $\mathcal{E}_i$ is assumed to be the same as $\mathcal{E}$. It is desired to characterize the error rate, $\mathcal{E}$. The following technique will provide an interval estimate of the mean of $\mathcal{E}$. That is, the expected error rate will be estimated. Let the mean and variance of $\mathcal{E}$ be as follows:

$$\begin{aligned} \mathbf{E}\{\mathcal{E}\} &= m \\ \sigma^2\{\mathcal{E}\} &= b^2 \end{aligned}$$

where $E\{\cdot\}$ is the expectation operator and $\sigma^2\{\cdot\}$ is the variance of the argument. The goal is to find an interval estimate for m . Let the estimate for m be

$$\hat{m} = \frac{1}{n} \sum_{i=1}^n \mathcal{E}_i$$

which is unbiased and consistent. Now consider the following interval estimate for the mean:

$$\text{Prob}\{a < \hat{m} - m < b\} \approx \gamma$$

where a , b , and γ are determined by the distribution of $\hat{m} - m$. It is easily shown that $\hat{m} - m$ is approximately a Gaussian random variable with zero mean and variance of b^2/n . The actual variance of $\mathcal{E}$, b^2 can be estimated as follows:

$$\hat{b}^2 = \frac{1}{n-1} \sum_{i=1}^n (\mathcal{E}_i - \hat{m})^2$$

which is unbiased (10:130). Now the interval estimate can be restated as follows:

$$\text{Prob}\{\hat{m} - z\hat{b} < m < \hat{m} + z\hat{b}\} \approx \gamma$$

where z and γ are determined by the standard Gaussian distribution. As with the Classifier Confidence Interval, a 95 percent confidence is given by $z = 1.96$ and $\gamma = 0.95$, and the 90 percent confidence interval implies $z = 1.64$ and $\gamma = 0.90$.

These two confidence interval techniques serve different functions. The Monte Carlo method is used primarily to determine the average performance which can be expected from a multilayer perceptron when trained on a given problem. Its purpose is to provide a way to discount the variability of performance results as a function of the initial weights and the presentation order. The Classifier method on the other hand is used strictly for classification performance and specifies an interval within which the true error rate lies with some given probability. In this dissertation, the Monte Carlo Confidence Interval method is used to compare training algorithms on a multilayer perceptron classifier. By

Table 1. Comparison of Monte Carlo and Classifier Confidence Interval Methods

Monte Carlo	Classifier
Attempts to characterize performance as a function of the initial weights and presentation order.	Attempts to determine the true error rate for the classifier.
Used to compare training algorithms for multilayer perceptrons.	Used to compare the multilayer perceptron classifier with traditional classifiers.
Applies to any random variable of the training process.	Applies only to classification accuracy or error rate.
Width of interval primarily controlled by the number of training runs.	Width of interval primarily controlled by the number of test vectors.
Results based on the application of the Central Limit Theorem to the random variable measured.	Results based on approximation of the measured error rate by a binomial random variable.

using this technique to compare learning algorithms, it is possible to factor out differences that occur due to different initial weights and presentation order. Generally, the algorithms are compared on synthetic data for which the solution is known *a priori*. On the other hand, the Classifier Confidence Interval method is applied when bounds on the true error rate is desired for comparison of the multilayer perceptron classifier with other classifiers. In this case, the data is normally sensor data for which the ideal solution and minimum error rate are not known. Table 1 summarizes the differences between these two confidence interval techniques. Figures 7 and 8 summarize the pertinent equations.

Figure 7. Summary of Equations for Monte Carlo Confidence Interval Technique

The mean and variance of the error rate are

$$\begin{aligned} E\{\mathcal{E}\} &= m \\ \sigma^2\{\mathcal{E}\} &= b^2 \end{aligned}$$

The mean and variance are estimated by

$$\begin{aligned} \hat{m} &= \frac{1}{n} \sum_{i=1}^n E_i \\ \hat{b}^2 &= \frac{1}{n-1} \sum_{i=1}^n (E_i - \hat{m})^2 \end{aligned}$$

where n is the number of training sessions. The interval estimate is

$$\text{Prob}\{\hat{m} - z\hat{b} < m < \hat{m} + z\hat{b}\} \approx \gamma$$

where z and γ are determined by the standard Gaussian distribution.

Figure 8. Summary of Equations for Classifier Confidence Interval Method

The measured error rate is given by

$$\hat{E} = \frac{L}{P}$$

where P is the total number of test vectors and L is the number misclassified. The mean and variance of the measured error rate are

$$\begin{aligned} E\{\hat{E}\} &= E \\ \sigma^2\{\hat{E}\} &= E(1 - E)/P \end{aligned}$$

where E is the true error rate. The variance of the measured error rate is estimated as

$$\hat{\sigma}^2\{\hat{E}\} = \hat{E}(1 - \hat{E})/P$$

The interval estimate on the true error rate is

$$\text{Prob}\{\hat{E} - z\hat{\sigma}\{\hat{E}\} < E < \hat{E} + z\hat{\sigma}\{\hat{E}\}\} \approx \gamma$$

where $\hat{\sigma}\{\hat{E}\}$ is the square root of the estimated variance of $\hat{E}$, and z and γ are controlled by the standard Gaussian distribution.

III. Feature Selection - Analyzing the Weights in a Multilayer Perceptron

3.1 Introduction

In this chapter, a novel technique for analyzing a trained multilayer perceptron will be presented. This technique, called the *saliency metric*, allows the determination of how the inputs affect the classification performance of the network. This provides a method for selecting which input features to use in the final classifier design as well as a vehicle for the comparison of learning rules. Through the examination of the weights in two identical networks trained with different learning algorithms, it can be seen how the learning rules result in different networks or similar ones. The following section will explain why the input features to a classifier should be limited. In Section 3.3, the conventional techniques for feature selection will be reviewed. After that the saliency metric will be introduced and shown to be consistent and useful.

3.2 Why Limit Input Features?

The first consideration when designing a classifier is what features to use as inputs. The initial temptation is to use a large number of features since each feature hopefully provides some information regarding classification. However, the number of features must be limited for several reasons. The primary reason for limiting the number of input features is known as the *curse of dimensionality* (6:187) (7:95) which requires that as the number of features grows the size of the training set required to characterize the distribution of the vectors grows exponentially. Since it is very expensive to collect labeled data for classification, training sets are by their nature small.

Foley's Rule (9) provides some guidelines as to the minimum number of training samples required for accurate classification as a function of the input features. Foley showed empirically that the *number of training samples per class* should be greater than

three times the number of features. When this condition is satisfied, the error rate on the training set will be close to the error rate on an independent test set. His results are limited to two class discrimination and a multivariate Gaussian distribution of the input features, but the results could be more general since two key variables in his derivations were sums of independent, identically distributed variables (9:623). Thus, for most practical distributions the variables will approach Gaussian distributions by the Central Limit Theorem. Also, Cover has shown that if the *total number of training samples* is less than twice the number of features, then there exists a hyperplane which can separate the training data perfectly even if the two classes are generated by the same distribution (3)! Hence, an absolute minimum number of training vectors per class is equal to the number of input features, but greater than three times the number of features per class is desirable.

Another reason for reducing the number of input features is to minimize the complexity of the classifier. In general, as the number of input features increases, the time to classify an input vector increases unless special purpose parallel hardware is used. Also, the number of free parameters in the classifier may increase with the number of features. A large number of free parameters in the classifier may allow overfitting to the training data with resultant poor performance on independent test data. The work of Baum and Haussler (1) provides some guidelines for the number of training vectors required as a function of the number of weights in the network and the desired error rate. They suggest that the number of training vectors, P , be greater than S/ϵ where S is the number of weights and thresholds in the network and ϵ is the desired error rate on an independent test set. Hence, if an accuracy rate of 90% is desired on the test set, then the number of training vectors in the test set should be at least 10 times the number of weights and thresholds in the network.

3.3 *Conventional Feature Selection Techniques*

It is clear that it is critical to be able to determine from a set of potential features which provide the best discrimination and which are worthless. The optimal technique

for feature selection requires implicit examination of every subset of the set of features under consideration (6:207-214). This requirement leads to a combinatoric explosion in the computations required and is, therefore, impractical. Several suboptimal methods are available for selecting a subset of features (6:214-223). The method used by Roggemann in his dissertation (19:71-76) will provide the baseline for comparing the results of the multilayer perceptron-based solution presented here.

Roggemann used the best features or *single probability of error criterion* approach. This technique computes the probability of error on the training set using each feature individually in succession. These error rates are then rank ordered and the M features with the lowest probability of error (P_e) are selected (6:215-216). This method has the advantage over the optimal method in that the numerical complexity grows linearly with the number of features; however, it fails to consider the relationships between the input features. Two features with a fixed relationship between them and each giving a low P_e would both be selected using this method although the second feature gives no additional information.

3.4 The Saliency Metric

3.4.1 Introduction A novel technique will be presented here to determine which features are useful for classification using a multilayer perceptron and which features are useless. The technique also provides a way to compare the performance of different learning rules and a way to analyze the behavior of multilayer perceptrons. The saliency metric is also described by Ruck *et al* in (27). The next section will present a derivation of the saliency metric followed by a test of the consistency and utility of the metric. The final section of this chapter shows the results of comparing learning rules for multilayer perceptrons using the saliency metric.

3.4.2 Derivation of Saliency Metric The network analysis technique developed examines the responsiveness of the network outputs. The sensitivity of the network

outputs to its inputs is used to rank the input features' usefulness or *saliency*. First, an expression for the derivative of an output with respect to a given input will be derived; then it will be shown how this can be used to create a measure of the sensitivity of a trained multilayer perceptron to each input feature.

The notation to be used throughout is as follows. Superscripts always represent a layer index and not a quantity raised to a power unless otherwise indicated. The layers are counted from the first layer of nodes which compute the sigmoid of a weighted sum. Thus, Layer 1 is the first layer of hidden nodes and not the inputs. The output of node i in Layer j is denoted by x_i^j . Input i is represented by x_i with no superscript, and output i is represented by z_i . For the weights, the first subscript denotes the source node, and the second denotes the destination. The superscript on weights represents the layer of the destination node. Hence, w_{ij}^k is the weight connecting node i in layer $k - 1$ to node j in layer k , and θ_j^k is the threshold associated with node j in layer k .

Consider again the network in Figure 2 which is repeated here for convenience as Figure 9. It is desired to calculate the derivative of output z_i with respect to input x_j . Suppose each of the nodes in the network performs a weighted sum of its inputs plus a threshold term and puts the result through a sigmoid as shown in Figure 10. To compute the desired derivative, all that is required is some simple calculus. Using the fact that the derivative of that sigmoid is the output times one minus the output yields,

$$\frac{\partial z_i}{\partial x_j} = z_i(1 - z_i) \frac{\partial}{\partial x_j}(a_i^3)$$

where a_i^3 is the activation of node i in layer 3. Activation is the weighted sum of the inputs plus the node threshold. The activation of node i in layer k is denoted by a_i^k . Substituting the expression for the activation gives

$$\frac{\partial z_i}{\partial x_j} = z_i(1 - z_i) \frac{\partial}{\partial x_j} \left(\sum_m w_{mi}^3 x_m^2 + \theta_i^3 \right)$$

The summation is over all nodes in layer 2. Applying the derivative to this expression for the activation produces

$$\frac{\partial z_i}{\partial x_j} = z_i(1 - z_i) \sum_m w_{mi}^3 x_m^2 (1 - x_m^2) \frac{\partial}{\partial x_j} (a_m^2)$$

Let $\delta_i^3 = z_i(1 - z_i)$. Then

$$\frac{\partial z_i}{\partial x_j} = \delta_i^3 \sum_m w_{mi}^3 x_m^2 (1 - x_m^2) \frac{\partial}{\partial x_j} (a_m^2)$$

Continuing the process through two more layers yields

$$\frac{\partial z_i}{\partial x_j} = \delta_i^3 \sum_m w_{mi}^3 \delta_m^2 \sum_n w_{nm}^2 \delta_n^1 w_{jn}^1 \quad (13)$$

where

$$\delta_m^2 = x_m^2 (1 - x_m^2)$$

$$\delta_n^1 = x_n^1 (1 - x_n^1)$$

Note that the derivative of the output with respect to the input (Equation 13) depends not only on the weights in the network, the w_{ij} , but also on the current outputs of the nodes in the network, the x_i^k and the network outputs, z_i . Thus, the derivative depends on the current input to the network as well as the network weights.

The dependence of the derivative of the outputs with respect to the inputs on the current input to the network forces the evaluation of the derivative at a set of points in the input space. Ideally, each input would be independently sampled over its expected range of values. Suppose R points were used for each input. Then the total number of points the derivatives would be evaluated at would be R^M where M is the number of inputs. As an example, suppose 22 features were being evaluated, and 20 points in the feature space were sampled over the expected range of each feature. In this case, $R = 20$ and $M = 22$, so the number of points at which the derivative must be evaluated is approximately 10^{28} . If 10^6 derivative evaluations could be performed each second, it would take about 10^{15}

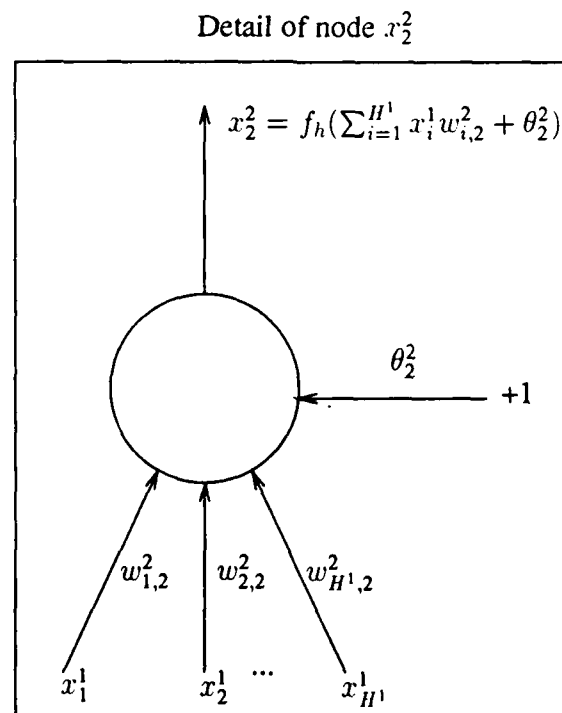
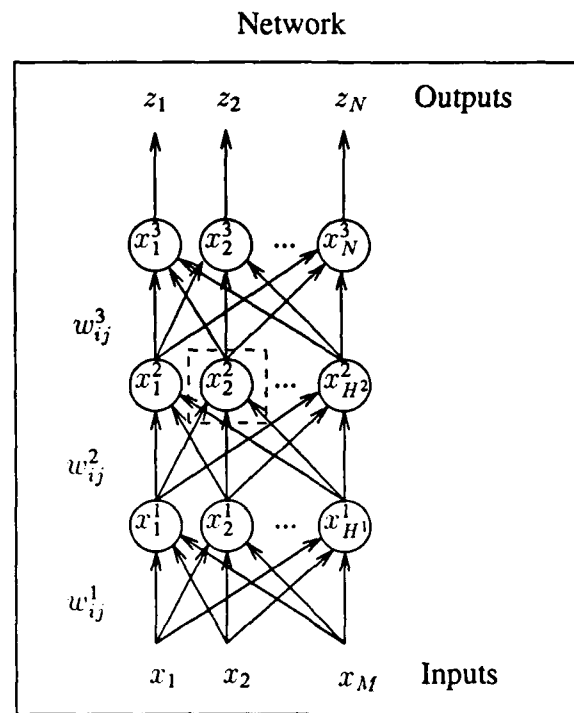


Figure 9. Multilayer Perceptron Architecture

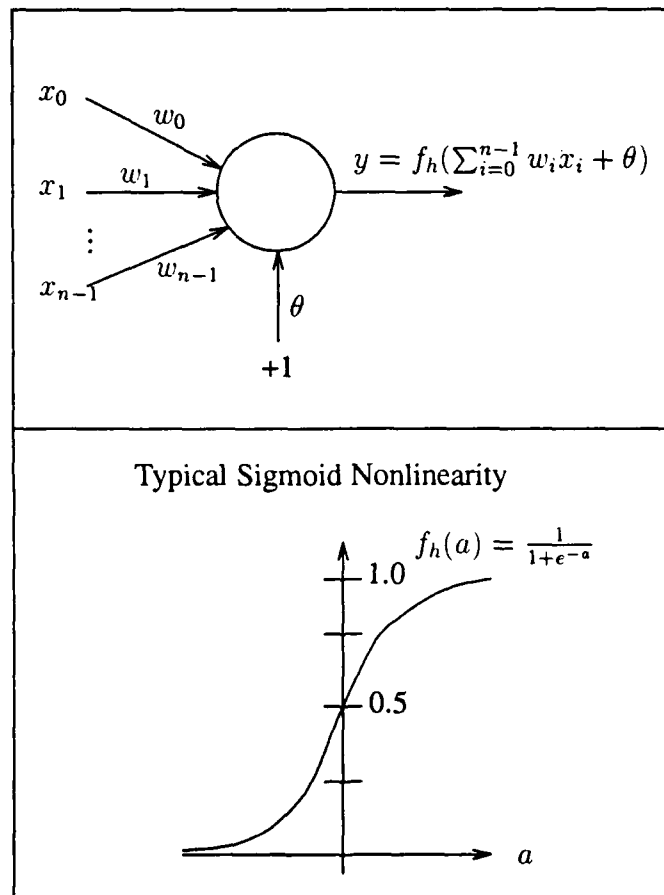


Figure 10. Detail of a Node in the Multilayer Perceptron

years (or about one million universe times) to evaluate all the derivatives. Even for this small example the number of computations is intractable. Obviously, as the number of inputs grows the number of derivative evaluations increases tremendously. In fact, this is an NP-complete problem. Since the input space cannot be sampled uniformly, it is desired to sample it at the *important* points. The points of greatest importance in the input space are those where training data exists; hence, the training vectors are used as starting points to sample the input space. For every training vector, each input is sampled over its range while the other inputs are determined by the training vector currently being evaluated. Suppose there are P training vectors, then the number of derivative evaluations is PMR . Continuing the above example, suppose 500 training vectors are available. The number of derivative evaluations is 220,000 which would take about one-fifth of a second to compute. Now the number of evaluations increases linearly with the number of inputs, M , which is computationally tractable.

A measure of the *saliency* of an input can now be formulated as follows. Let Λ_j , the saliency of input j , be defined by

$$\Lambda_j \equiv \sum_{\mathbf{x} \in \mathbf{X}} \sum_i \sum_{x_j \in D_j} \left| \frac{\partial z_i}{\partial x_j}(\mathbf{x}, \mathbf{w}) \right| \quad (14)$$

where $\mathbf{x}$ indicates the M -dimensional vector inputs, $\mathbf{X}$ is the set of P training vectors, $\mathbf{w}$ represents the weights in the network, D_j represents the set of R points for input x_j which will be sampled. Normally, D_j is a set of uniformly spaced points covering the expected range of input x_j . Using this method, the derivatives are not computed at the exact training points, but can be computed arbitrarily closely by increasing the number of points in D_j . It is believed that in a high dimensionality input space, having all inputs except one set to the values of a given training point results in evaluation of derivatives sufficiently close to the the actual training vectors for the purposes of the saliency metric. Note that the dependence of the derivative of the output, z_i , with respect to the input, x_j , on the weights, $\mathbf{w}$, and the input to the network, $\mathbf{x}$, is explicitly shown in Equation 14.

Two questions now need to be asked. Is this metric consistent with existing statistical techniques and independent of network initial conditions? Specifically, does the ranking of the Λ_j actually correspond to the importance of the inputs as determined by the single feature probability of error method? This is the utility question. Second, will the ranking of the Λ_j be independent of the initial network conditions? In other words, is the saliency metric consistent? The next section will address these questions.

3.4.3 Test of the Saliency Metric In this section, the consistency of the saliency metric developed in the previous section will be examined followed by a test of the actual utility of the metric. The first test consisted of training 100 networks on a set of image recognition data. The classification problem consisted of using a set of 22 moment invariants computed from objects segmented from doppler imagery to determine the type of object. These features were chosen for this recognition task because they have previously been shown to be effective for PSRI aircraft recognition using conventional techniques (8). There were four object classes: tank, jeep, 2.5-ton truck, and oil tanker truck. The p_q ordinary moment of an object is an integral over the object of the factor $x^p y^q$. That is,

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) x^p y^q dx dy$$

where M_{pq} is the pq ordinary moment of the function $g(x, y)$. In this case, the function $g(x, y)$ simply represents the silhouette of the segmented objects. These moments are then scaled to provide invariance to changes in either the position or size of the object. *Moment invariants* are created by combining individual scaled moments so that the resulting feature is invariant to changes in rotation of the object in the viewing plane. Thus, the moment invariants are unchanged by differences in position, scale, or in-plane rotation of an object. See (24) (28) for more information on this recognition problem.

Each network was started with a different set of random weights. The order of presentation of the training data was random and different for each training session. The momentum method was used for training with $\eta = 0.3$ and $\alpha = 0.8$. After each network

was trained, the saliency of each input, Λ_j , was calculated. For each network, the rank of every input was computed from the sorted ordering of the saliency metric. The input with the lowest Λ_j , indicating the least sensitivity of the output to that input, was ranked zero; and the input with the highest was ranked number 21. Next a histogram was computed for each input using the 100 different networks indicating the number of times the input had a given rank. Ideally, the rank of a given input would be independent of the network's initial weights and presentation of the training data. In that case, the histogram for an input would be a single spike at the rank of the input. Figure 11 shows the histograms for three of the 22 input moment invariants. The three features selected are representative of the types of distributions observed for all the features. The distribution about rank 21 represents a feature that was consistently determined to be important using Equation 14. The distribution spread across the middle rank values represents a feature which does not provide significantly more information than approximately 10 of the other features because the median of the distribution is near rank 11. Finally, the distribution that is dominated by low values represents a feature that is consistently found to be unimportant independent of the network initial conditions. It is possible from the histograms to produce an overall ranking of the saliency of the inputs. Alternatively, the saliency metric can be averaged for a given feature over all networks trained. The average saliencies can then be ranked to yield an overall ranking for the input features. This approach yields nearly identical results to the histogram method. As a preliminary test of the utility of the saliency metric, the three features ranked most important using saliency and their associated weights were eliminated from a trained network and the classification accuracy dropped from 85 percent to below 50 percent. Also, the three least significant features according to the saliency metric and their weights were eliminated from a trained network which resulted in no degradation of classification accuracy.

In summary, the saliency metric is a consistent measure of the ranking of the input features. As the histograms indicate, there is some variation in the ranking as a function of the initial weights of the network. Those features of mid-level importance have the widest

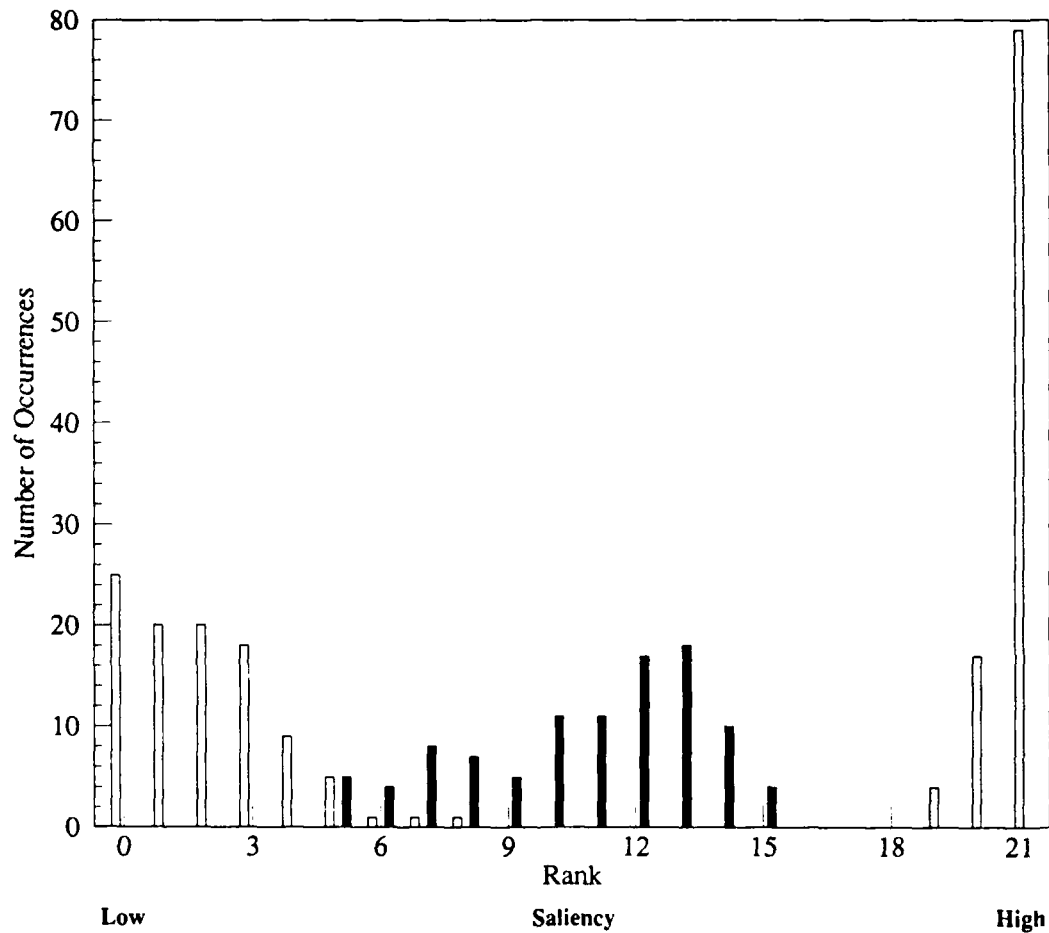


Figure 11. Rank Histograms of Saliency Metric for Three Moment Invariants. Each feature is displayed using a different fill pattern. The moment invariant feature represented by the unshaded bars ranging in rank from 0 to 8 shows very little saliency, meaning that this feature provided little information in the pattern classification process. The darkly shaded bars ranging in rank from 5 to 15 represent a moment invariant feature which in most cases provided less information for classification than approximately ten other input features. The feature represented by the medium shaded bars ranging from rank 19 to 21 has a great deal of saliency and provided much information in classifying the input patterns.

variation indicating that the network can choose from several of the mid-level features for discrimination.

The next question to be answered is whether or not this saliency metric is useful. As previously stated, a preliminary test indicated that it would be useful for determining which features to select; hence, a more rigorous test was devised. In this test, the problem was to identify targets from non-targets in forward looking infrared (FLIR) images where targets consisted of tanks, trucks and armored personnel carriers (APCs). A set of nine features were extracted for this purpose. The features chosen have previously been shown to be effective using conventional techniques (19) (20) (21). The features are listed in Table 2. These features were ranked using the probability of error criterion. The feature with the lowest probability of error was given the highest rank and *vice versa*. This ranking provided the baseline for evaluation of the utility of the saliency metric developed. Note that these features and the probability of error criterion were used by Roggemann; hence, his work provides a baseline for this neural network approach.

As with the consistency test, 100 networks were trained on the same data with different random initial weights and a different random order of presentation of the training data. Histograms of the ranking of the input features were constructed. The histograms of three representative features distributions are shown in Figure 12. The ranking of the features by the probability of error criterion and the saliency metric is shown in Table 3. Note that the two methods of ranking the data are in agreement for the top two and bottom two features. As a further test of the utility of the saliency metric, the accuracy of classifiers designed using the top three features from each ranking were compared. Table 4 shows the results when the classifiers are tested using the hold one out method. The classifier confidence interval technique was used to produce the estimates of the true classification accuracy for each classifier. Note that the accuracy rate differences between the classifiers using the top three features of either ranking are negligible. Hence, the saliency metric does provide a useful measure of the significance of input features.

Table 2. FLIR Features Evaluated

Feature	Description
Complexity	Ratio of border pixels to total object pixels
Length/Width	Ratio of object length to width
Mean Contrast	Contrast ratio of object's mean to local background mean
Maximum Brightness	Maximum brightness on object
Contrast Ratio	Contrast ratio of object's highest pixel to its lowest
Difference of Means	Difference of object and local background means
Standard Deviation	Standard deviation of pixel values on object
Ratio Bright Pixels/Total Pixels	Ratio of number of pixels on object within 10% of maximum brightness to total object pixels
Compactness	Ratio of number of pixels on object to number of pixels in rectangle which bounds object

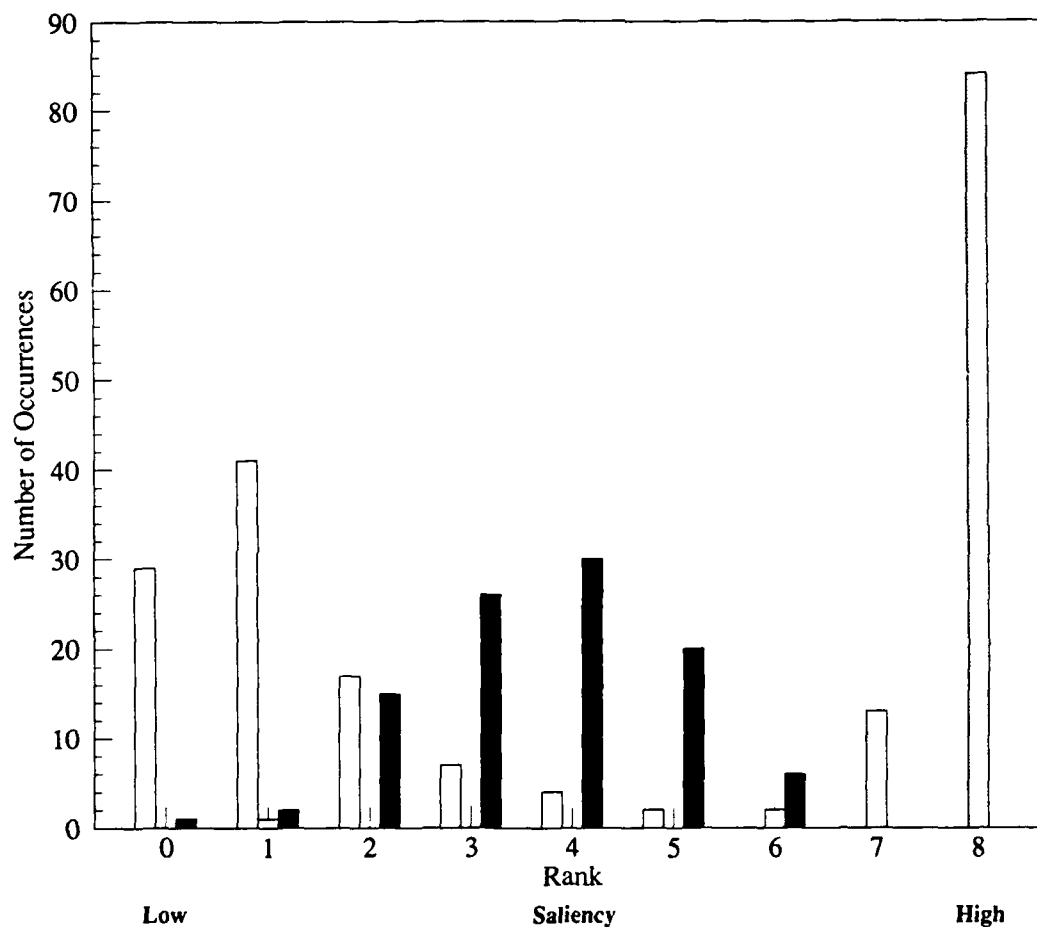


Figure 12. Rank Histograms for Three FLIR Imagery Features. Each feature is plotted using a different fill pattern. The FLIR feature represented by the unfilled bars shows little saliency and, thus, did not provide much information for classifying the input patterns. The darkly shaded bars represent a FLIR feature with a moderate degree of saliency providing some information for classification. The feature shown with the lightly shaded bars has a large saliency and provided much information for classifying the FLIR objects.

Table 3. Ranking of FLIR Features

Feature	P_e Criterion Rank	P_e	Saliency Metric Rank	Average Saliency
Complexity	8	0.210	8	0.473
Length/Width	7	0.244	7	0.403
Difference of Means	3	0.311	6	0.323
Standard Deviation	2	0.317	5	0.300
Mean Contrast	6	0.259	4	0.275
Contrast Ratio	4	0.296	3	0.272
Maximum Brightness	5	0.296	2	0.223
Compactness	0	0.357	1	0.199
Ratio Bright Pixels/Total Pixels	1	0.350	0	0.192

Table 4. Estimated True Classification Accuracies of Two Feature Subsets (95% confidence interval is shown)

Classifier	Best 3 P_e Features	Best 3 Saliency Metric Features
Bayesian w/ Parzen Windows ($\sigma = 0.1$)	90.3-94.7%	91.3-95.5%
k -Nearest Neighbor ($k = 1$)	90.5-94.9%	90.3-94.7%
Multilayer Perceptron (3-5-2, $\eta = 0.3, \alpha = 0.8$)	88.3-93.1%	87.7-92.7%

3.4.4 Application to Learning Rule Evaluation In this section, the saliency metric will be used to compare two learning rules for multilayer perceptrons. As stated in Section 2.3.1, there are several learning rules available for determining the weights in a multilayer perceptron. Two will be compared here. The traditional backpropagation and the extended Kalman filtering approaches will be considered. The question to be considered is whether or not the Kalman approach would use the features in a different way than the backpropagation approach. That is, would the importance of the inputs for backpropagation differ significantly from those for extended Kalman filtering? The Kalman filter (not the extended one) is optimal in several metrics of optimality. The estimates produced by the Kalman filter are the mean, mode, and median of the random variable being estimated. It is also the maximum likelihood estimate as well as the minimum mean squared error estimate (10:231-236). These traits do not carry over to the extended Kalman filter; nonetheless, it is a powerful estimation algorithm. A disadvantage of the extended Kalman filter algorithm is its computational complexity, $\mathcal{O}(S^3)$, where S is the number of weights and thresholds in the system while backpropagation with or without momentum is only $\mathcal{O}(S)$. The computational complexity is the number of arithmetic operations required to perform a single update of the weights in the network. It is hoped that backpropagation would find a similar solution to the classification problem, in terms of weighting the importance of input features, as the extended Kalman filter algorithm at a much lower computational cost.

The doppler image data with the four classes (p. 38), which was used to test the consistency of the saliency metric, provided the test data. One hundred networks were trained with the extended Kalman filtering approach using the same data, initial weight values, and presentation order of training vectors as the 100 which were trained using backpropagation. Thus, the only difference between the two sets of 100 networks was the training method employed. Figure 13 shows the histograms of the ranking of the same three input features used as examples in Figure 11 (p. 40). Compare these with the histograms in Figure 11. Note that while there are some differences between the two, the

ranking of the features is mostly the same between the two training methods. Hence, the two training rules place equal relative emphases on the input features. Table 5 shows the average saliency values computed for all twenty-two features for both training methods.

3.5 Conclusion

In this chapter, a new technique for ranking the importance of features for a multilayer perceptron has been developed. The saliency metric has been shown empirically to be both consistent and useful. When compared with the traditional method of ranking input features by the probability of error criterion, it performed as well and resulted in a similar ranking for the input features. Also, backpropagation was compared to extended Kalman filtering using the saliency metric. It was found that both methods for setting the weights in a multilayer perceptron place equal relative emphases on the features. The saliency metric gives a clear indication that multilayer perceptrons are able to effectively ignore useless inputs as indicated by the test results. This fact gives insight into the behavior of multilayer perceptrons with respect to sensitivity to features.

One extension to the saliency metric is its application to the hidden nodes in a multilayer perceptron. The outputs of the hidden nodes are inputs to the following layer; hence, the saliency of the hidden nodes can also be computed. Thus, it should be possible to eliminate unnecessary hidden nodes using this technique. Also, conceptually the idea of the saliency of weights in a network is possible. This method should, hence, also be extensible to computing the saliency of weights in the network yielding a method for pruning the weights in a network. Future research into this area will, hopefully, provide a method for automatically sizing the hidden layers in a multilayer perceptron.

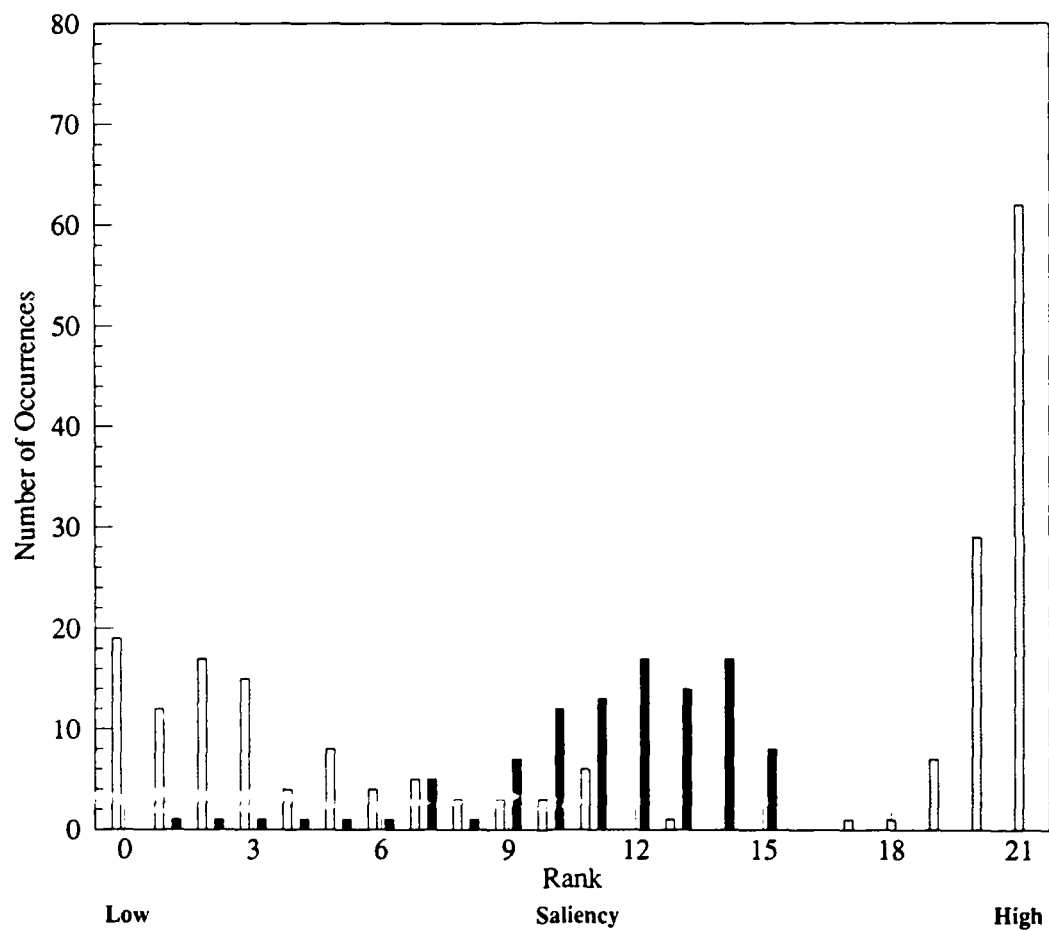


Figure 13. Rank Histograms for Three Moment Invariants Using Kalman Approach. The same three features are displayed as those in Figure 11 using the same fill patterns. Comparing each of the three histograms with those in Figure 11 shows that the Kalman approach and backpropagation place equal relative emphases on the input features.

Table 5. Average Saliency Values on Doppler Recognition Data for Extended Kalman Filtering and Backpropagation Training Algorithms

Backpropagation		Kalman		Feature Index
Rank	Average Saliency	Rank	Average Saliency	
0	0.13	1	0.26	6
1	0.14	2	0.26	9
2	0.15	0	0.24	7
3	0.17	3	0.26	17
4	0.21	4	0.27	10
5	0.22	6	0.33	4
6	0.33	5	0.33	20
7	0.34	15	0.58	16
8	0.35	11	0.47	5
9	0.38	9	0.38	8
10	0.42	7	0.36	11
11	0.43	8	0.38	1
12	0.44	10	0.41	12
13	0.44	12	0.48	14
14	0.53	14	0.48	15
15	0.56	13	0.48	3
16	0.76	16	0.69	2
17	0.80	19	0.82	21
18	0.82	18	0.74	18
19	0.91	20	0.95	19
20	0.96	17	0.73	0
21	1.16	21	1.00	13

IV. Understanding the Backpropagation Training Algorithm

4.1 Introduction

The previous chapter described a technique for deciding which features to use for classification in a multilayer perceptron. After the set of features has been determined, the network must then be trained using some training algorithm. This chapter will analyze the popular backpropagation training algorithm in the context of the well-known and understood technique of extended Kalman filtering. It has already been shown that backpropagation and extended Kalman filtering place equal relative emphases on the input features. This suggests that the techniques may be related somehow. It will be shown that backpropagation is actually a degenerate form of extended Kalman filtering for training the weights. Several examples using both synthetic data and sensor data will highlight the differences and similarities between the training algorithms.

4.2 Degenerating Kalman to Backpropagation

It has been known for some time that backpropagation is a gradient descent procedure for setting the weights in a multilayer perceptron when performed in the batch mode (14) (31) (35). However, instantaneous backpropagation is only approximately a gradient descent procedure. That is, the gradient is computed for the error surface defined by the immediate training vector only and not the ensemble of training vectors. An alternate interpretation of backpropagation is proposed here, namely, as a degenerate form of the extended Kalman filter.

In this section, it will be shown how the extended Kalman filter approach to setting the weights in a multilayer perceptron can be degenerated into the backpropagation algorithm. The following section will consider the implications of this result and the assumptions required to obtain it.

First, recall the equation for instantaneous backpropagation (Equation 2, p. 15):

$$w^+ = w^- - \eta \frac{\partial E_p}{\partial w}$$

which can be rewritten for the weight w_{ij}^l as

$$\Delta w_{ij}^l = -\eta \frac{\partial E_p}{\partial w_{ij}^l}$$

Now simplify using Equation 1 (p. 14):

$$\Delta w_{ij}^l = \eta \sum_{k=1}^N (d_{k,p} - z_{k,p}) \frac{\partial z_k}{\partial w_{ij}^l} \quad (15)$$

Recall that the p subscript refers to the current training vector input and N is the number of outputs. Equation 15 is simply another way of stating the backpropagation training rule.

Now, consider the update to $\hat{w}_i$, the i th component of $\hat{\mathbf{w}}$, the Kalman filter estimate of the optimal states of the network. From Equation 4, the change in $\hat{w}_i$ is

$$\Delta \hat{w}_i = \mathbf{K}_i (\mathbf{d} - \mathbf{z})$$

where $\mathbf{K}_i$ is the i th row of $\mathbf{K}$ and $i = 1, \dots, S$ with S the total number of weights and thresholds in the network. Recall the equation for the Kalman gain matrix (Equation 5, p. 18):

$$\mathbf{K} = \mathbf{P}\mathbf{H}^T [\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}]^{-1}$$

where the time dependence has been suppressed for notational convenience. Let the inverted quantity be denoted by $\mathbf{A}$. That is, let

$$\mathbf{A} = [\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}]^{-1}$$

Then

$$\mathbf{K} = \mathbf{P}\mathbf{H}^T\mathbf{A}$$

Now suppose that $\mathbf{A}$ is of the form

$$\mathbf{A} = a\mathbf{I}$$

where $\mathbf{I}$ is the N by N identity matrix and a is a scalar. Recall N is the number of outputs of the network. Then

$$\mathbf{K} = a\mathbf{P}\mathbf{H}^T$$

So

$$\mathbf{K}_i = a\mathbf{P}_i\mathbf{H}^T$$

where $\mathbf{P}_i$ and $\mathbf{K}_i$ are the i th rows of $\mathbf{P}$ and $\mathbf{K}$, respectively. Now compute K_{ij} .

$$K_{ij} = a \sum_{k=1}^S P_{ik} H_{kj}^T = a \sum_{k=1}^S P_{ik} H_{jk}$$

With K_{ij} , the update for $\hat{w}_i$ can be determined.

$$\begin{aligned} \Delta \hat{w}_i &= \mathbf{K}_i(\mathbf{d} - \mathbf{z}) \\ &= \sum_{j=1}^N K_{ij}(d_j - z_j) \\ &= \sum_{j=1}^N \sum_{k=1}^S a P_{ik} H_{jk}(d_j - z_j) \end{aligned} \tag{16}$$

Further assume that $\mathbf{P}$ is diagonal. Then

$$\Delta \hat{w}_i = \sum_{j=1}^N a P_{ii} H_{ji}(d_j - z_j)$$

To relate this to the previous backpropagation weight update equation, use Equation 7 (p. 19) to obtain

$$\Delta w_i = \sum_{j=1}^N (d_j - z_j) a P_{ii} \frac{\partial z_j}{\partial w_i}$$

where the hat notation has been dropped simply to elucidate the similarities between the backpropagation training rule and the degenerated Kalman training rule. One final assumption will produce the backpropagation update rule. Suppose that $\mathbf{P} = p\mathbf{I}$. Then

$$\Delta w_i = ap \sum_{j=1}^N (d_j - z_j) \frac{\partial z_j}{\partial w_i}$$

which is the backpropagation rule (Equation 15) with $\eta = ap$.

The assumptions required to obtain this result were

1. $[\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}]^{-1} = a\mathbf{I}$
2. $\mathbf{P}(t_i) = p\mathbf{I}$

In the next section, the significance of the assumptions will be considered.

4.3 Implications

The previous section demonstrated that backpropagation can be considered a degenerate form of the extended Kalman filter. This section will examine the assumptions required to obtain this result and consider its implications. Consider how the extended Kalman filter would operate when implementing backpropagation. Only two modifications to the usual algorithm are needed. First, the matrix inversion required by Equation 5 is replaced by using a constant matrix equal to $a\mathbf{I}$ in its place. Second, instead of propagating the $\mathbf{P}$ matrix, its value is held constant at $p\mathbf{I}$. Thus, the update equations become

$$\begin{aligned} \hat{\mathbf{w}}(t_i^+) &= \hat{\mathbf{w}}(t_i^-) + \mathbf{K}(t_i)[\mathbf{d}(t_i) - \mathbf{z}(t_i)] \\ \mathbf{K}(t_i) &= \mathbf{P}(t_i^-)\mathbf{H}^T(t_i)a \\ \mathbf{P}(t_i^+) &= \mathbf{P}(t_i^-) \end{aligned} \tag{17}$$

and the propagation equations are

$$\hat{\mathbf{w}}(t_{i+1}^-) = \hat{\mathbf{w}}(t_i^+) \tag{18}$$

$$\mathbf{P}(t_{i+1}^-) = \mathbf{P}(t_i^+) \quad (19)$$

with the initial conditions of $\mathbf{w}(t_0)$ and $\mathbf{P}(t_0) = p\mathbf{I}$. Also, no driving noise is being used in the system model. Compare these update and propagation equations with those presented in Section 2.3.2 (Equations 4 - 6 and 10 - 11).

Now consider the significance of the assumptions required. Assumption 2 is easier to understand, so consider that first. The assumption is that $\mathbf{P} = p\mathbf{I}$. From the propagation equation (19), it is apparent that if $\mathbf{P}$ starts in this form, it will remain in that form because $\mathbf{P}$ is not altered during propagation. The assumption that $\mathbf{P}$ starts diagonal is not significant. It merely states that there is no knowledge regarding the cross correlation of initial errors in the weights which is a reasonable assumption. However, the lack of propagation of the $\mathbf{P}$ matrix is significant. It states that the filter (backpropagation) is not using any information to update its knowledge of the errors in the weights as training proceeds. Thus, as training progresses the weights are updated the same amount even though previous training should decrease the errors in the weights; and, hence, the weights should be changed less based on the current residual (the residual is the difference between the desired output and the actual output, *i.e.*, $\mathbf{d}(t_i) - \mathbf{z}(t_i)$). As a side note, this aspect of Kalman training provides the motivation for decreasing the training rate, η , over time. Also, by maintaining the diagonal form of $\mathbf{P}$, backpropagation is assuming that the errors in the weights are uncorrelated which is not the case. Since the relationships between the weights are known, namely, the weights in the upper layers contribute to the derivative of the outputs with respect to weights in the lower layers, then there is useful information in the off-diagonal terms of $\mathbf{P}$. Because backpropagation discards the off-diagonal terms, it does not use all the information which is available to it.

Assumption 1 is more difficult to interpret. It says that the result of inverting the matrix in Equation 5 is a constant of value $a\mathbf{I}$. Assuming that $\mathbf{R}(t_i) = \epsilon\mathbf{I}$ and using

Assumption 2, we obtain

$$[\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}]^{-1} = [p\mathbf{H}\mathbf{H}^T + \epsilon\mathbf{I}]^{-1}$$

The assumption that $\mathbf{R}$ is proportional to the identity matrix is reasonable since $\mathbf{R}$ represents the uncertainty in the measurements from the system (*i.e.*, the multilayer perceptron's outputs) and there is no reason to believe the noise in the outputs is correlated because the weights going into the final output nodes are independent of each other. In order for the inverse to be $a\mathbf{I}$ requires that $\mathbf{H}\mathbf{H}^T = b\mathbf{I}$ for $b > 0$. This condition will be true when the rows of $\mathbf{H}$ are proportional to an orthonormal set, *i.e.*, the rows must be mutually orthogonal and the length of each row vector of $\mathbf{H}$ must be $\sqrt{b}$. In general, these conditions are not satisfied. Particularly, since the row entries of $\mathbf{H}$ are derivatives of a fixed output with respect to all the weights in the network, there is no guarantee that all the row vectors will have the same length. However, when these conditions are satisfied,

$$[\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}]^{-1} = \frac{1}{pb + \epsilon} \mathbf{I}$$

Since these conditions are not, in general, satisfied, it is clear that backpropagation throws away information that would otherwise be useful in updating the weights.

The following section compares the performance of backpropagation to Kalman filtering on several different examples. These examples will demonstrate that Kalman filtering outperforms backpropagation in most cases.

4.4 Performance Comparisons Between

Backpropagation and Extended Kalman Filtering

In the previous section, it was demonstrated that backpropagation can be considered a degenerate form of extended Kalman filtering. In the process of degenerating the Kalman filter, information was discarded which is useful in setting the weights. This section will examine the comparative performance of backpropagation and Kalman filtering on four

example problems.

The first problem is known as the exclusive-OR (XOR) problem. Suppose that a classification problem is given such that vectors composed of two features must be classified into two categories. Let $\mathbf{x}$ be the feature vector and ω_1 and ω_2 be the two categories. Further assume that the vectors can be classified as shown in Figure 14. A set of 1000 training vectors was generated from a uniform distribution on the region $[0, 1] \times [0, 1]$. Each of the training vectors was classified using Figure 14. For a vector in class ω_i the associated desired output was a vector with all entries of 0.1 except for the entry i whose value was 0.9. These values were chosen because the sigmoidal function of the individual nodes ($f_h(a)$ in Figure 3, p. 12) could achieve values of zero and one only for infinite inputs. Attempting to train to zero and one will push the weights towards large values. A single hidden layer network was trained on this set of vectors using backpropagation, momentum, and extended Kalman filtering. The number of hidden layer nodes was chosen to be four because Singhal claimed the Kalman algorithm was able to find a correct solution ninety percent of the time with four nodes (34). Thus, there were two input nodes; four hidden layer nodes; and two output nodes (one for each class). This network is described by the notation 2-4-2 where the first number is the number of inputs, the second is the number of hidden layer nodes, and the last is the number of outputs.

The network's performance was tested during training on the set of vectors. Each vector was input to the network and the output was observed. The network made a correct classification when the maximum output corresponded to the class of the input vector. Since the solution obtained by any of the training algorithms employed is dependent on the initial state of the network, one hundred different initial states were chosen; and the results were averaged over all one hundred training runs. Figure 15 shows the estimated average classification accuracy of the network trained using three different learning rules: extended Kalman filtering, backpropagation, and momentum as a function of the number of training vector presentations (iterations). The 95 percent confidence interval of the estimated average accuracy *versus* the number of training iterations has been plotted using

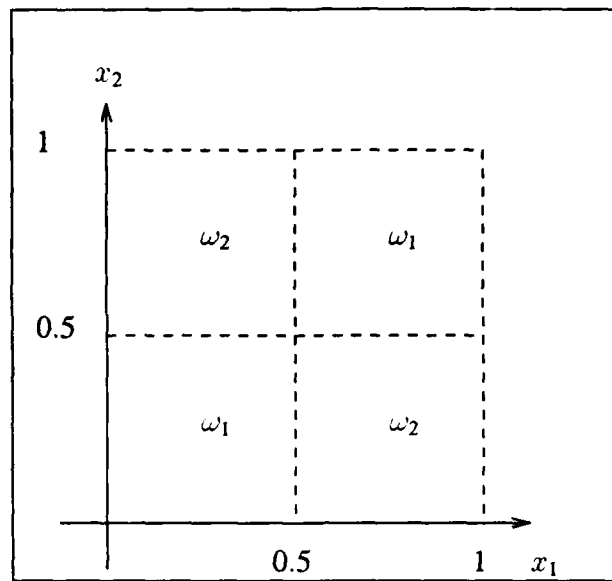


Figure 14. XOR Classification Problem

the Monte Carlo technique. From the figure, it is readily apparent that Kalman filtering achieves higher performance more quickly than either backpropagation or the momentum method when compared on the basis of iterations. Figure 16 shows the same data where the computational complexity of the algorithms has been taken into consideration. That is, the estimated average accuracy has been plotted as a function of the number of floating point operations (FLOPS) required. Now it is seen that the Kalman algorithm requires an order of magnitude more FLOPS to achieve the same accuracy as either backpropagation or momentum.

The next example uses a more complex configuration of two dimensional decision regions. Consider a four class decision problem where the feature vectors are two dimensional and can be classified as shown in Figure 17. A set of 1000 training vectors was uniformly distributed on the decision space. A network with two hidden layers of ten nodes each was used for this example. This network is described as a 2-10-10-4 network since there are two inputs, four outputs, and two hidden layers of ten nodes

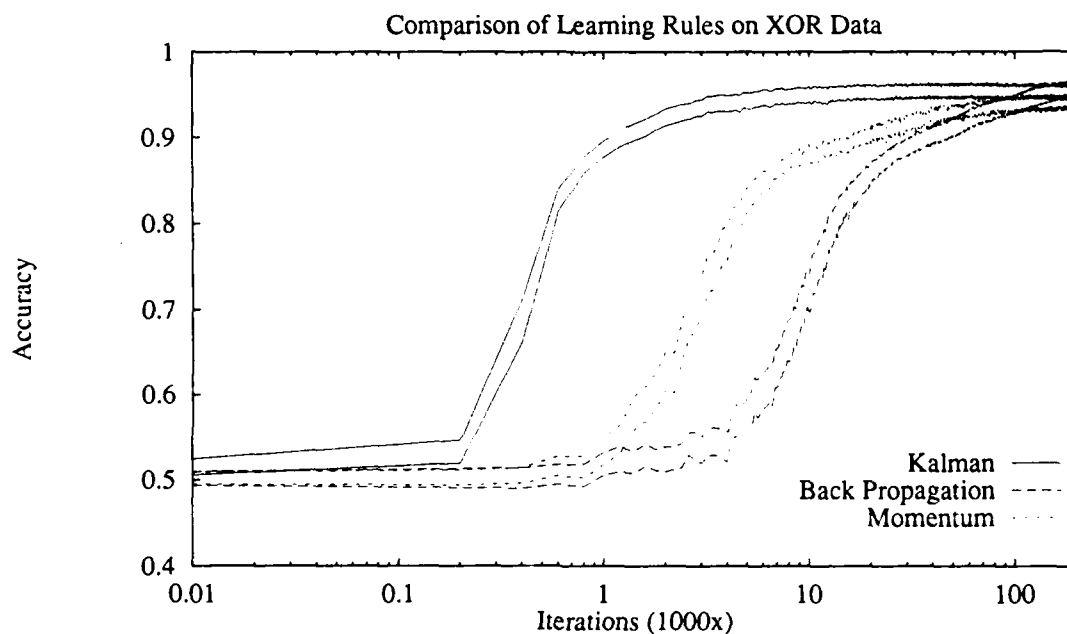


Figure 15. Average Training Accuracy *versus* Iterations for Kalman, Backpropagation, and Momentum Training on the XOR Data (2-4-2 network). Kalman parameters are $\epsilon = 1.0$ and $q = 0.05$. Backpropagation learning rate is $\eta = 0.3$ and the momentum factor is $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is shown for each training method.

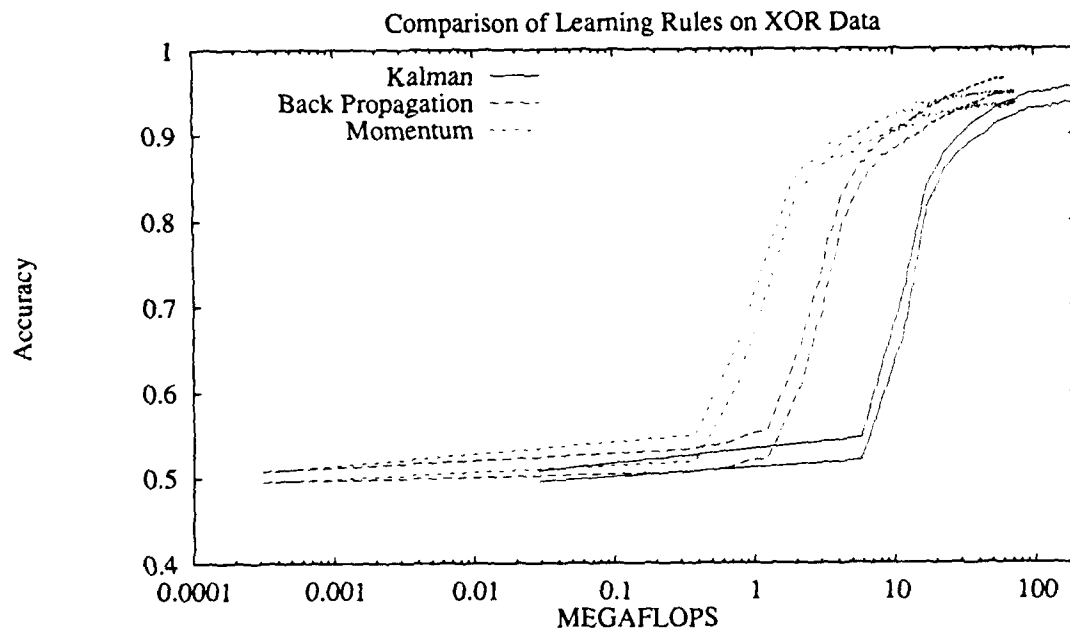


Figure 16. Average Training Accuracy *versus* Floating Point Operations (FLOPS) for Kalman, Backpropagation, and Momentum Training on the XOR Data (2-4-2 network). Kalman parameters are $\epsilon = 1.0$ and $q = 0.05$. Backpropagation learning rate is $\eta = 0.3$ and the momentum factor is $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is shown for each training method.

each. The training classification performance for the three methods is shown in Figure 18 *versus* iterations. For the backpropagation and momentum learning rules the average accuracy is plotted with 95 percent confidence using the Monte Carlo technique. A single run of the Kalman learning rule on the problem is shown. The computational complexity of the Kalman learning algorithm prohibited using the Monte Carlo confidence interval technique. It can be seen from the figure that the Kalman method achieves higher classification accuracy more quickly than either backpropagation or the momentum method as a function of the number of training iterations. However, when the computational complexity is accounted for, the Kalman algorithm requires three orders of magnitude more computations than either backpropagation or momentum as shown in Figure 19. Again, the 95 percent confidence interval is shown for backpropagation and momentum while a single run is shown for the Kalman algorithm. As demonstrated in the figure, the 100 runs each of backpropagation and momentum learning rules required an order of magnitude fewer computations than a single run of the Kalman algorithm. Insufficient computer resources thus prevented averaging multiple runs using the Kalman algorithm since a single training session required approximately 100 billion floating point operations.

It is also possible to compare the performance of the training algorithms by examining the decision regions formed by the networks at various points in the training. Since the input is two-dimensional, the decision regions are easy to present graphically. The decision regions formed by both the extended Kalman filter and backpropagation are shown in Figure 20 after 26,600 iterations. Note that at this point the Kalman trained network decision regions are very close to the actual class boundaries with the exception of the small bump for class ω_1 near the center. The backpropagation trained network on the other hand does not closely approximate the true decision regions. Figure 21 shows the decision regions formed at two other points in the training of the network with backpropagation. The decision regions formed at 140,000 iterations are reasonably close to the true boundaries except for the small bump in the center again, and at 2 million iterations the backpropagation trained network accurately approximates the true decision

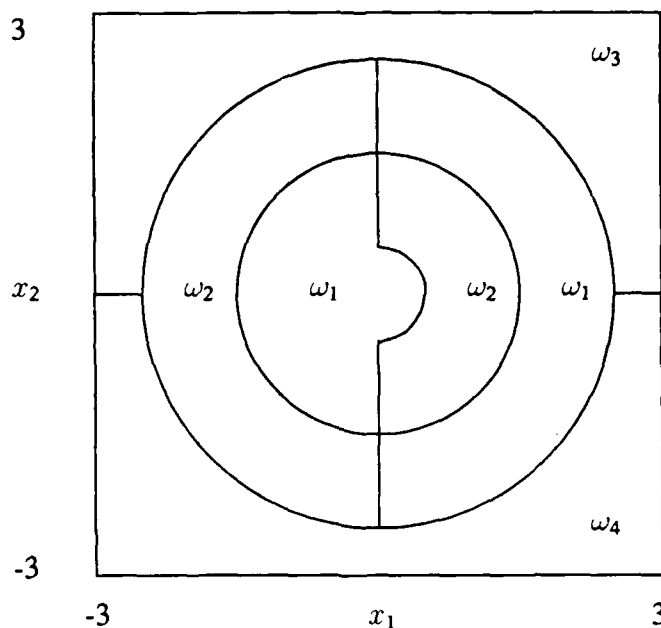


Figure 17. Mesh Classification Problem

regions including the small bump. It should be noted that the training data included only 10 vectors in that small bump region for class ω_1 out of a total of 1,000 training vectors. Also, the number of floating point operations for 2 million iterations on the backpropagation trained network is approximately equal to 225 extended Kalman filter training iterations.

The previous results have shown that the extended Kalman filtering algorithm outperforms backpropagation in terms of training accuracy *versus* iterations but not in terms of training accuracy *versus* floating point operations. The next two examples consider the performance of the training methods on sensor data.

In the next example, the doppler tactical target data (p. 38) was used for comparing the training algorithms. A set of 58 exemplars was used to train a 22-10-6-4 network. The training classification accuracy *versus* iterations is shown in Figure 22 for the three training methods. For the backpropagation and momentum techniques, a set of 100 runs were averaged. The figure shows the 95 percent confidence interval using the Monte

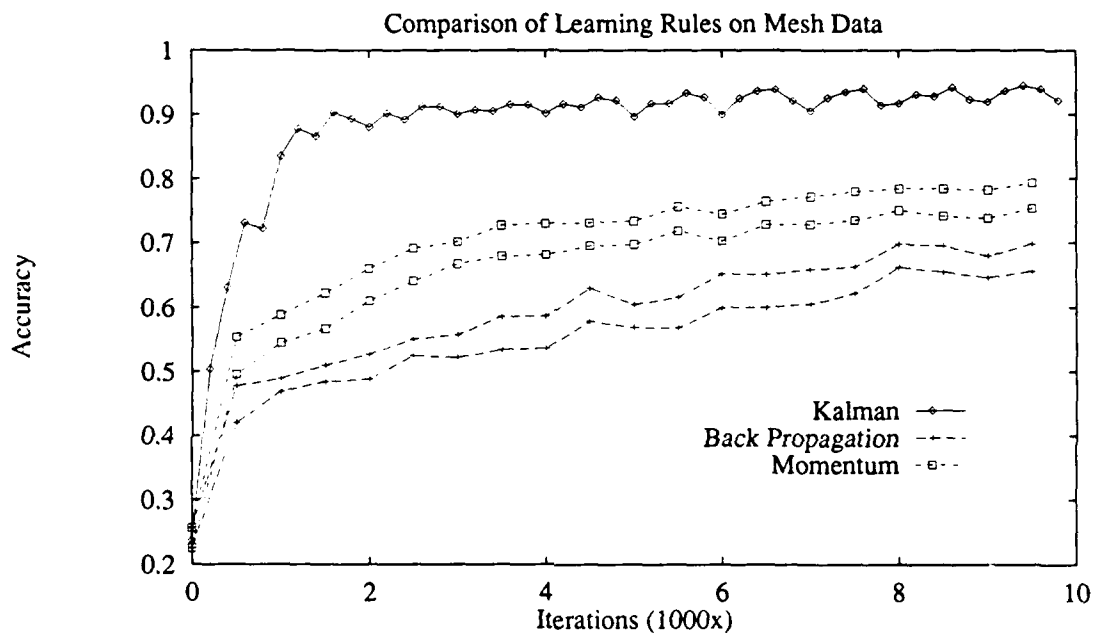


Figure 18. Mesh Training Accuracy *versus* Iterations. A 2-10-10-4 network was trained with the following parameters: Kalman ($\epsilon = 1.0$, $q = 0.05$), Backpropagation ($\eta = 0.1$), Momentum ($\eta = 0.1$, $\alpha = 0.6$). The 95 percent Monte Carlo confidence interval on the average training accuracy is shown for backpropagation and momentum. A single Kalman run is plotted.

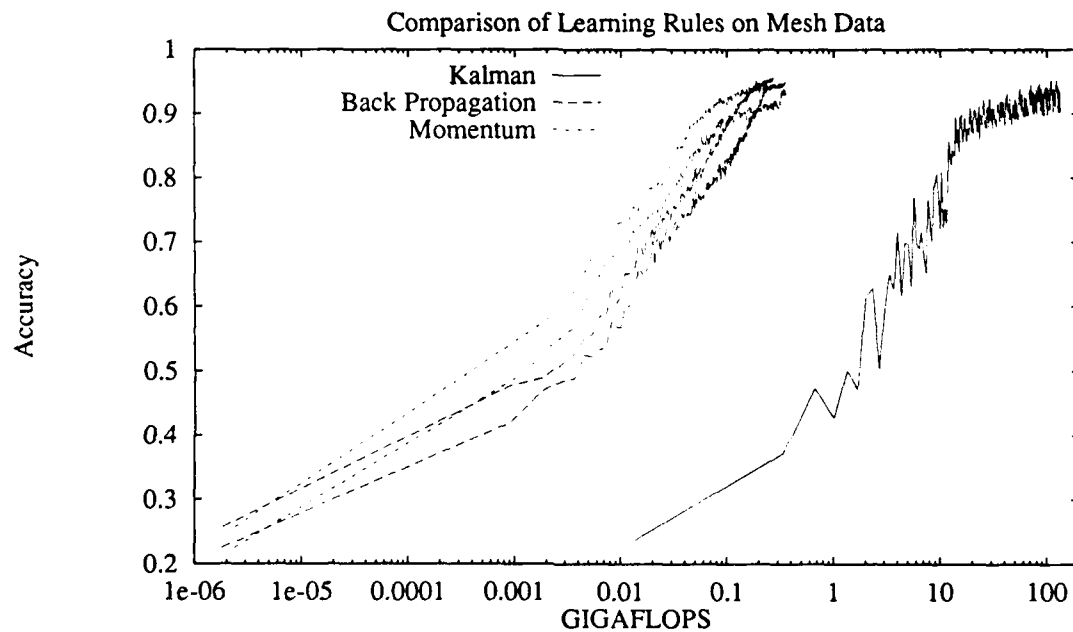
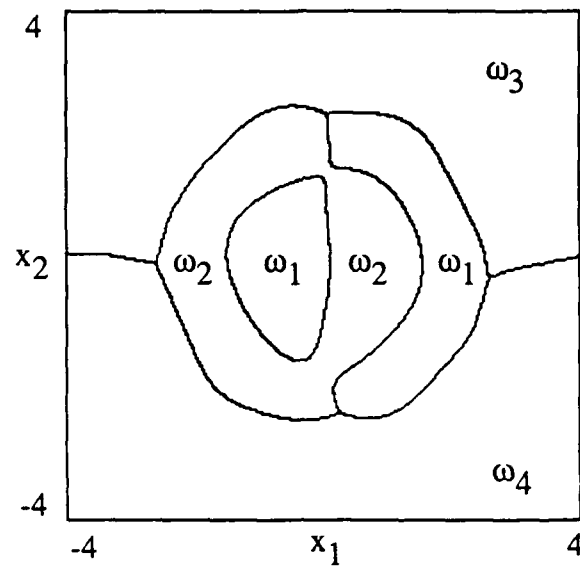
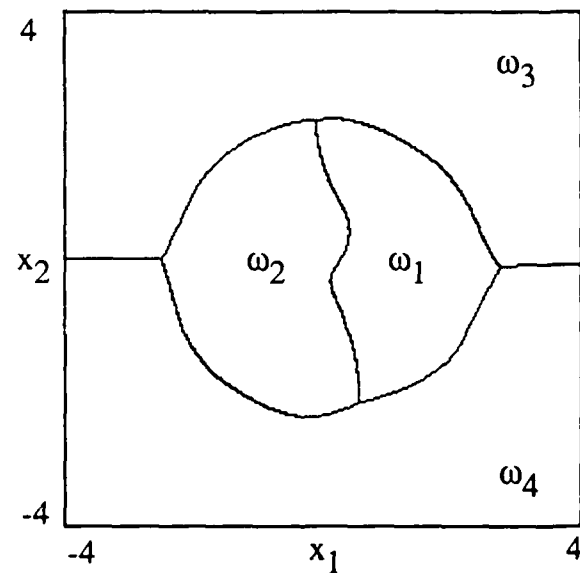


Figure 19. Mesh Training Accuracy *versus* FLOPS. A 2-10-10-4 network was trained with the following parameters: Kalman ($\epsilon = 1.0$, $q = 0.05$), Backpropagation ($\eta = 0.1$), Momentum ($\eta = 0.1$, $\alpha = 0.6$). The 95 percent Monte Carlo confidence interval on the average training accuracy is shown for backpropagation and momentum. A single Kalman run is plotted.



(a)



(b)

Figure 20. Mesh Decision Regions formed by the multilayer perceptron (2-10-10-4) trained with the extended Kalman filter algorithm (a) and backpropagation (b) at 26,600 iterations. Kalman parameters: $\epsilon = 1.0$ and $q = 0.05$. Backpropagation learning rate: $\eta = 0.1$.

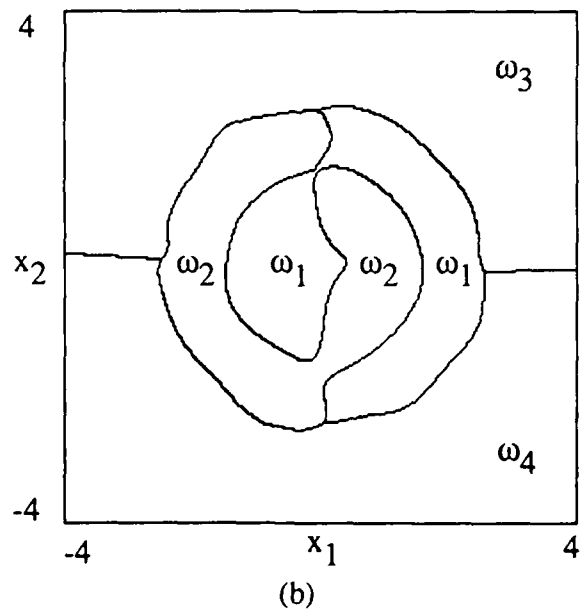
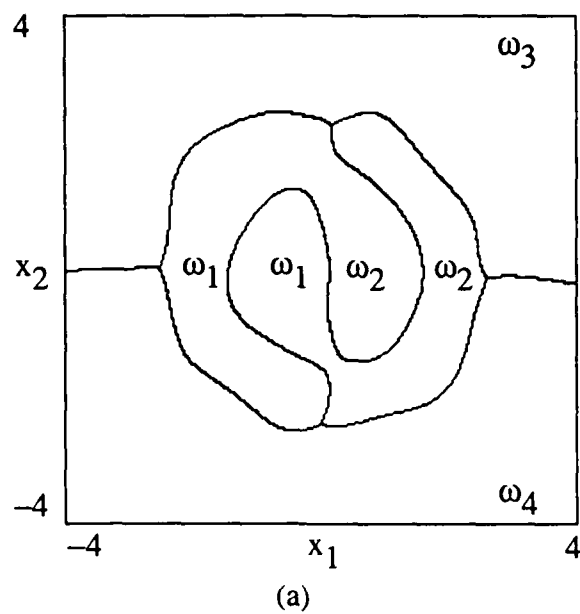


Figure 21. Mesh Decision Regions formed by the multilayer perceptron (2-10-10-4) trained with backpropagation at 140,000 iterations (a) and 2 million iterations (b). Learning rate $\eta = 0.1$.

Carlo method for the average training accuracy. The Kalman technique was not averaged for this problem due to the computational complexity; thus, the accuracy *versus* iterations for a single training session is plotted. Again it is apparent that the Kalman method achieves higher classification accuracy more quickly on the basis of iterations. The trained networks were tested on a set of twenty-three features vectors independent of the training database. The test results are shown in Figure 23. For the backpropagation and momentum techniques, the test accuracy as a function of training iterations is shown using the 95 percent Monte Carlo confidence interval. For the single Kalman training session, only the terminal test accuracy was measured at 1675 iterations. Thus, only one point for the Kalman run is plotted. From the plot, it is apparent that the single Kalman run achieved a higher test set accuracy (82%) than either backpropagation or momentum which achieved average test set accuracies of 72-74% and 74-76%, respectively (95 percent Monte Carlo confidence interval). When the network is trained using either backpropagation or the momentum for 10,000 iterations, an average classification accuracy on the test data of 78-80% (95 percent Monte Carlo confidence interval) is achieved. Figure 24 compares the learning rules on the basis of floating point operations for training accuracy. Again, the Kalman algorithm requires three orders of magnitude more FLOPS to achieve the same accuracy. The test accuracy as a function of FLOPS is shown in Figure 25. Lack of computer resources prevented averaging multiple runs of the Kalman learning rule since a single Kalman training session consumed over 100 billion floating point operations.

The final comparison of the learning rules used the Roggemann absolute range imagery. The problem was to classify feature vectors extracted from absolute range data as either target or non-target (target detection). A total of 276 non-targets and 124 targets were used. The hold out technique for testing was used in this example. Two-thirds of the vectors from each class were used for training and one-third were reserved for testing the trained networks. Eight features from the range data were used. They are shown in Table 6. The results of training a two-layer network with five hidden nodes are shown in Figures 26 and 27. The 95 percent Monte Carlo confidence interval of the estimated

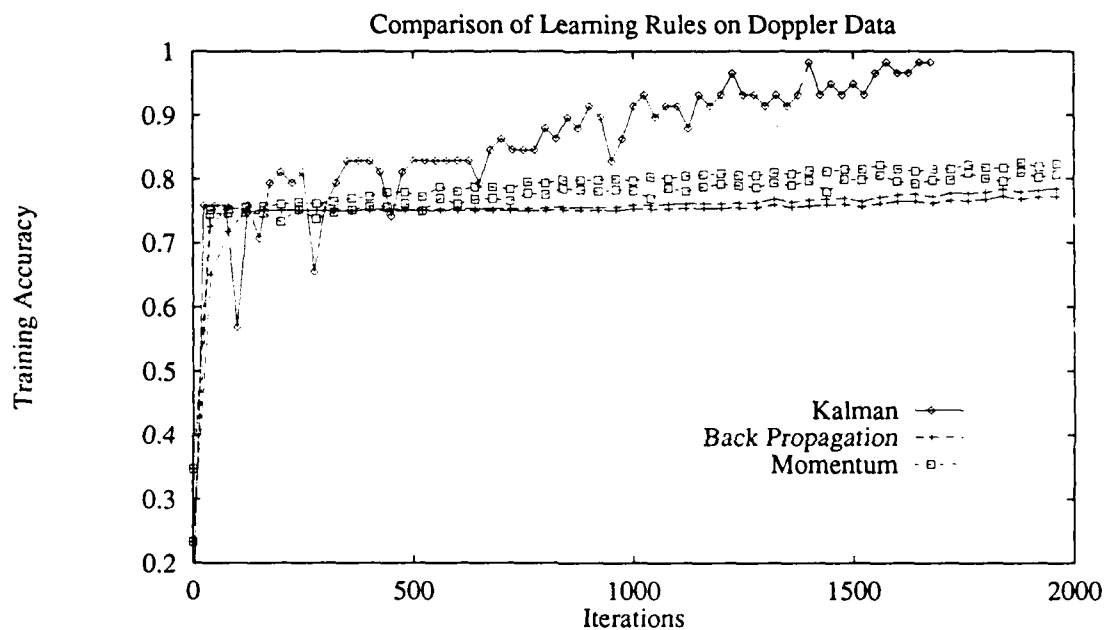


Figure 22. Doppler Data Training Classification Accuracy vs Iterations. A multilayer perceptron (22-10-6-4) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent confidence interval on the average accuracy is shown for backpropagation and momentum. A single Kalman run is plotted.

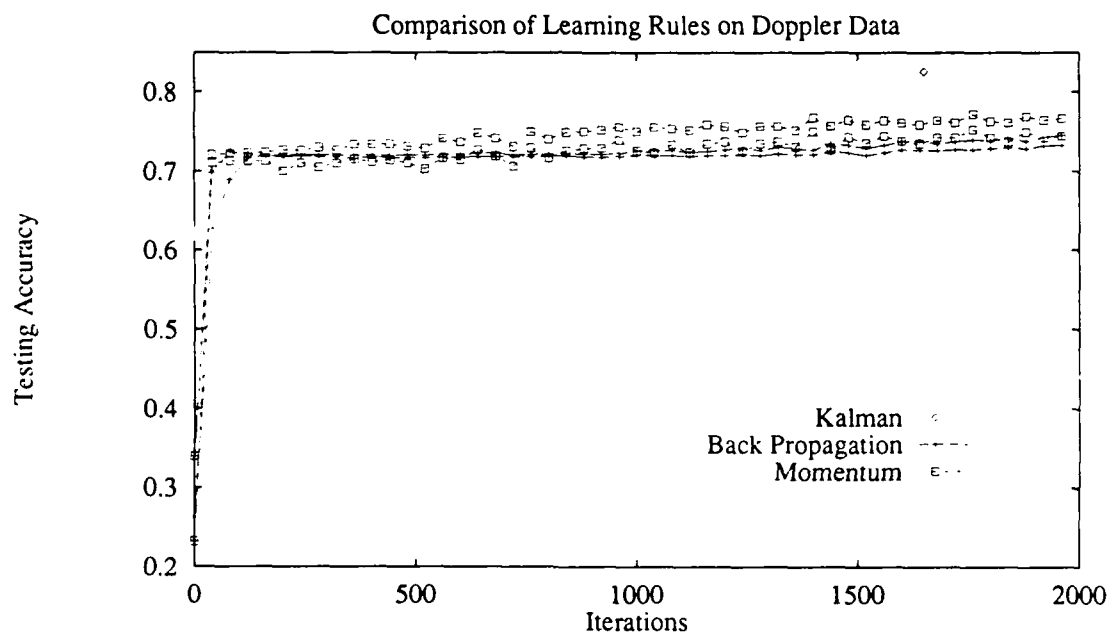


Figure 23. Doppler Data Testing Classification Accuracy vs Iterations. A multilayer perceptron (22-10-6-4) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent confidence interval on the average accuracy is shown for backpropagation and momentum. The terminal test accuracy for a single Kalman run is plotted.

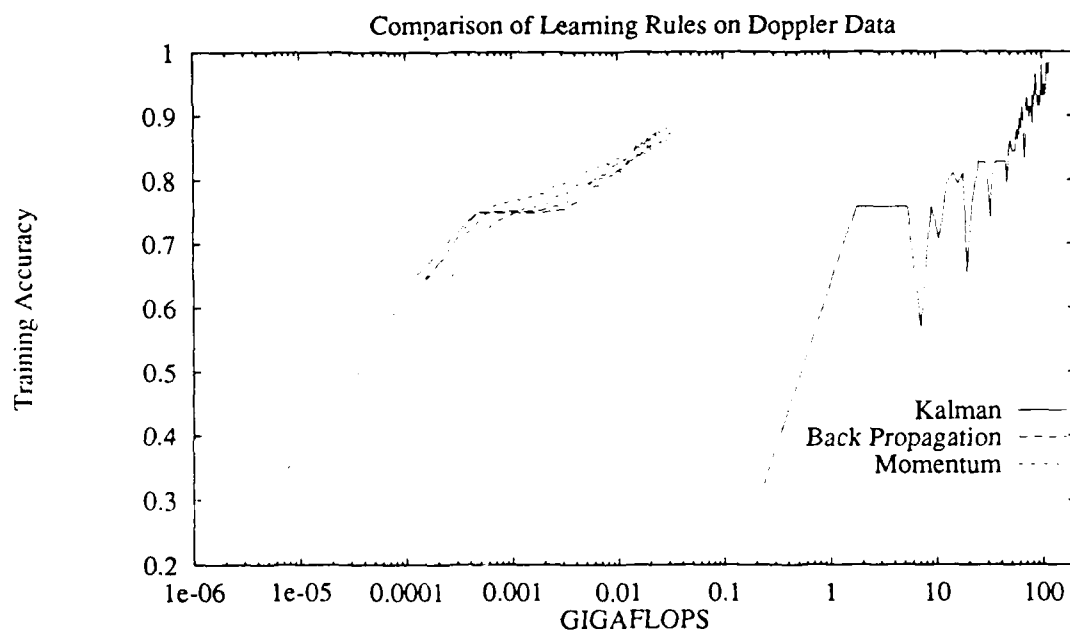


Figure 24. Learning Rule Comparisons on Doppler Data: Training Accuracy vs FLOPS. A multilayer perceptron (22-10-6-4) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. Kalman parameters are $\epsilon = 1.0$ and $q = 0.05$. The 95 percent confidence interval on the average accuracy is shown for backpropagation and momentum. A single Kalman run is plotted.

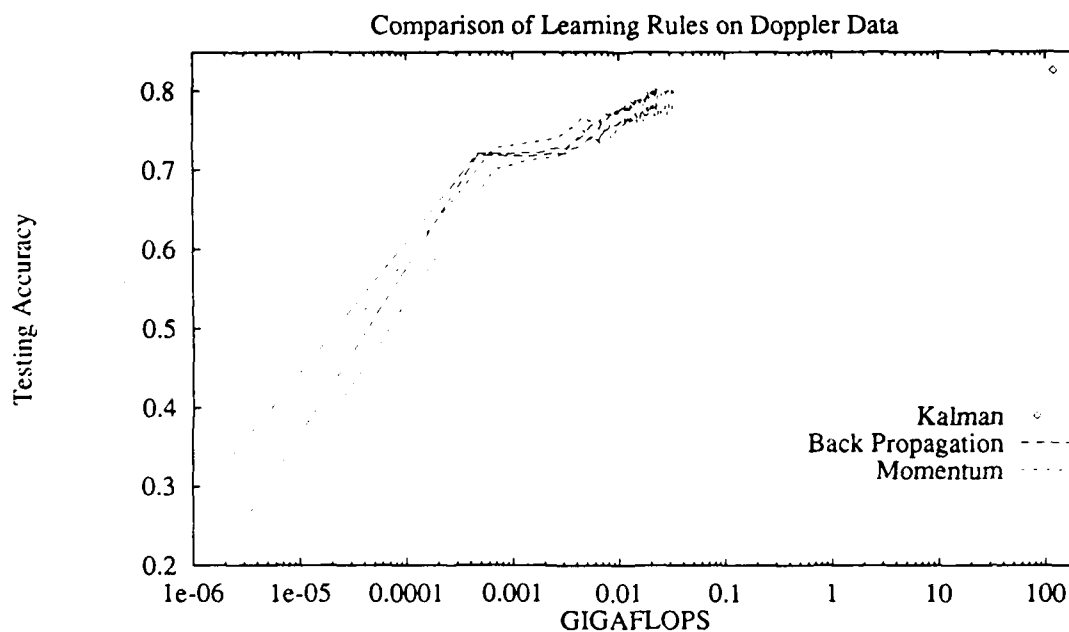


Figure 25. Learning Rule Comparisons on Doppler Data: Testing Accuracy vs FLOPS. A multilayer perceptron (22-10-6-4) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. Kalman parameters are $\epsilon = 1.0$ and $q = 0.05$. The 95 percent confidence interval on the average accuracy is shown for backpropagation and momentum. The terminal test accuracy for a single Kalman run is plotted.

Table 6. Absolute Range Features Used for Target Detection Problem

Feature	Description
Length/Width	Ratio of object length to width
Absolute Value of Difference of Standard Deviations	Absolute value of the difference between the standard deviation of the object pixel values and the standard deviation of the local background pixels
Complexity	Ratio of border pixels to total object pixels
Blob Standard Deviation	Standard deviation of the object pixel values
Absolute Difference of Means	Absolute value of the difference between the mean of the object pixel values and the mean of the local background pixels
Blob Length	Self-explanatory
Blob Height	Self-explanatory
Compactness	Ratio of number of pixels on object to number of pixels in rectangle which bounds object

average accuracy is plotted for the both the training and test data as training progresses. The full database of 400 feature vectors was randomly partitioned between training and test sets insuring proportionate representation in both sets for each of the 100 runs. In this case, the Kalman algorithm shows no benefit over the momentum technique on the test data. The situation is even worse for the Kalman algorithm when computational complexity is included as shown in Figures 28 and 29. Again, the Kalman algorithm uses three orders of magnitude more FLOPS to achieve somewhat poorer performance than either backpropagation or momentum.

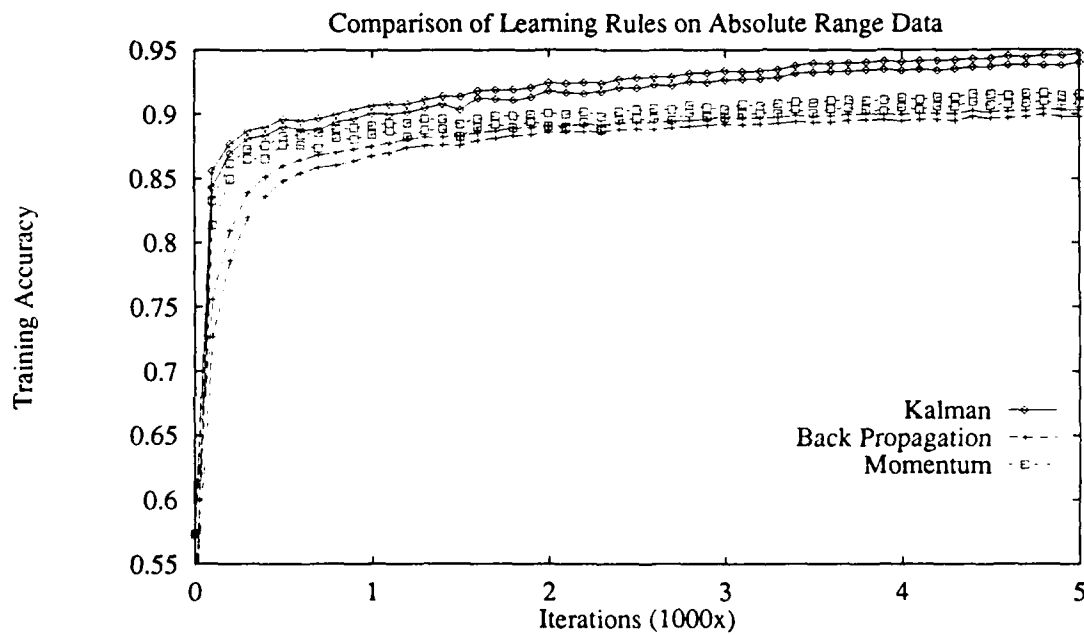


Figure 26. Target Detection Training Accuracy in Absolute Range Imagery *versus* Iterations. A multilayer perceptron (8-5-2) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is plotted for the average training accuracy.

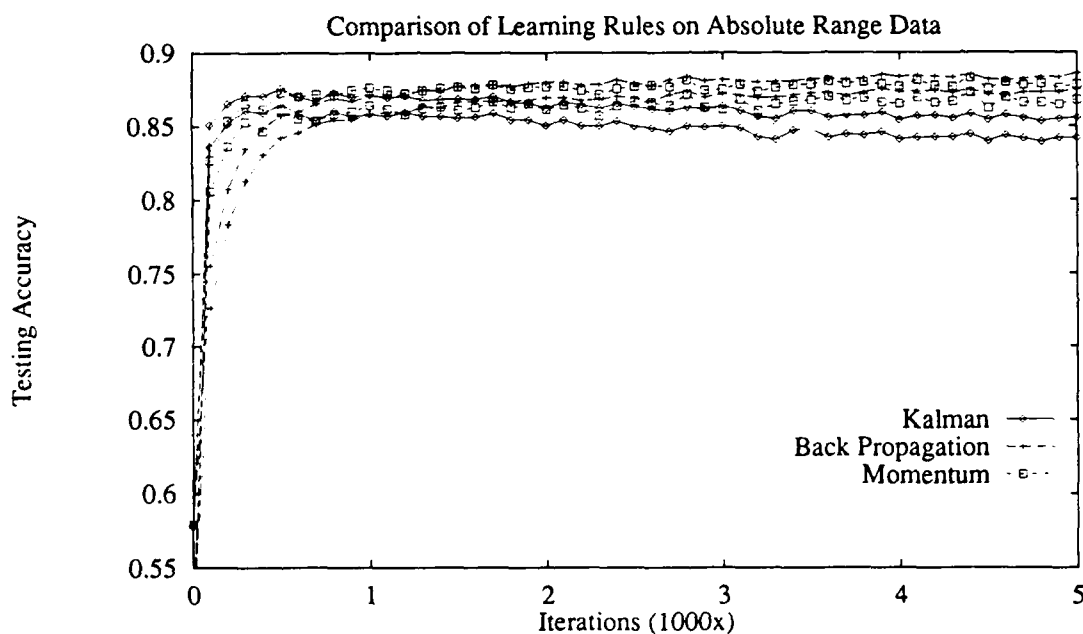


Figure 27. Target Detection Testing Accuracy in Absolute Range Imagery *versus* Iterations. A multilayer perceptron (8-5-2) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is plotted for the average test accuracy.

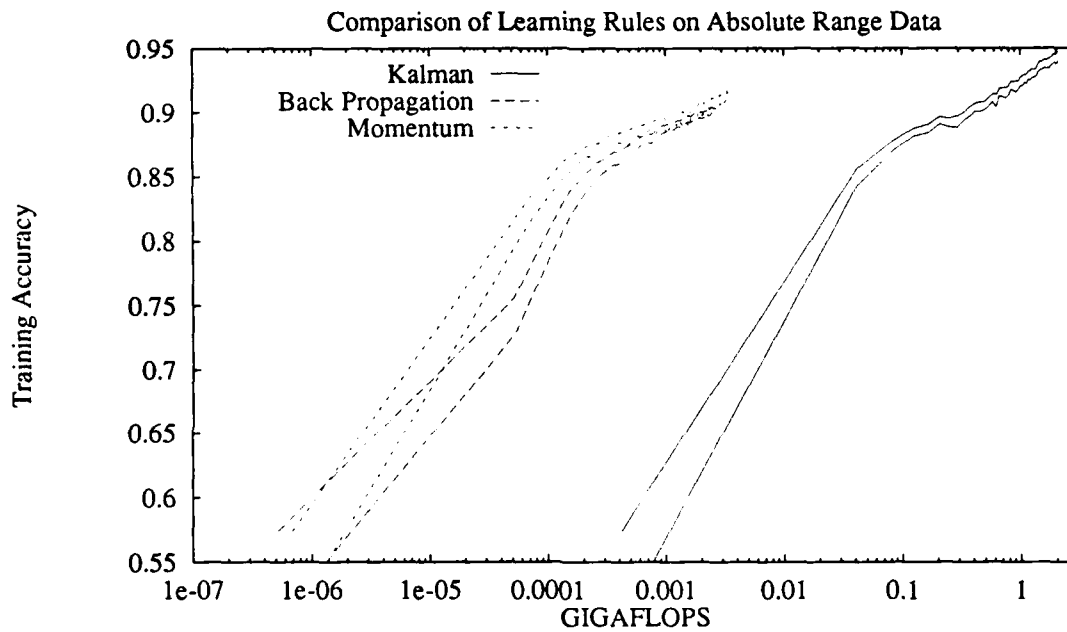


Figure 28. Target Detection Training Accuracy in Absolute Range Imagery *versus* FLOPS. A multilayer perceptron (8-5-2) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is plotted for the estimated average training accuracy.

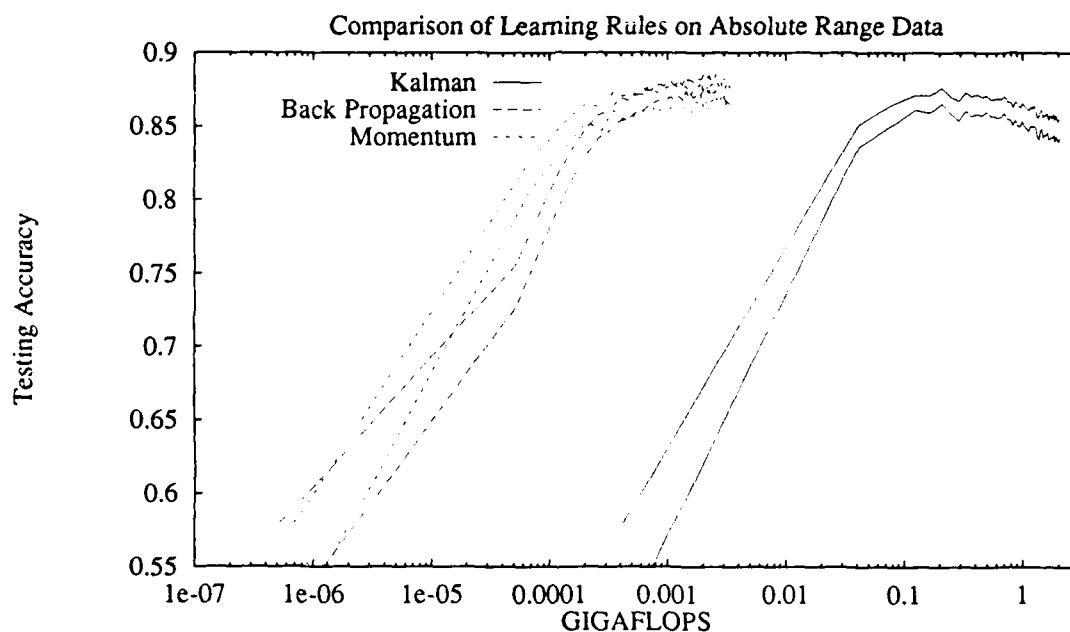


Figure 29. Target Detection Testing Accuracy in Range Imagery *versus* FLOPS. A multilayer perceptron (8-5-2) was trained with the following parameters: Kalman - $\epsilon = 1.0$ and $q = 0.05$; Backpropagation - $\eta = 0.3$; Momentum - $\eta = 0.3$ and $\alpha = 0.8$. The 95 percent Monte Carlo confidence interval is plotted for the average test accuracy.

In this section, four different classification problems were solved using the extended Kalman filtering approach to training a multilayer perceptron as well as using backpropagation and its cousin, the momentum method. On the *toy* problems using the XOR and mesh data as well as the doppler imagery, the extended Kalman filtering approach achieved higher classification accuracies more quickly than either backpropagation or momentum in terms of training iterations. However, on the absolute range imagery, there was no statistical advantage to the extended Kalman filtering technique over backpropagation or momentum in terms of average test accuracy *versus* iterations. In all cases, when the computational complexity was accounted for, the extended Kalman filtering method for training the weights required about three orders of magnitude more computations than either backpropagation or momentum to achieve the same classification accuracy.

4.5 Conclusion

It was previously shown by Singhal and Wu that a multilayer perceptron can be trained using an extended Kalman filter. This chapter has demonstrated a previously unknown result; namely, that backpropagation is a degenerate form of extended Kalman filtering. That is, the backpropagation algorithm results from making certain simplifying assumptions in the extended Kalman filtering approach. It was found that backpropagation sacrifices a great deal of information about the weights that the Kalman filter uses. The examples in Section 4.4 have shown the Kalman filter can achieve higher accuracy, in many cases, after a fixed number of training cycles than backpropagation by using the information which backpropagation discards; however, this is not always the case as demonstrated on the final example with the absolute range data. Although backpropagation discards much information, the result is a learning algorithm which is much more computationally efficient and achieves comparable classification performance.

*V. The Multilayer Perceptron:
A Bayes Optimal Discriminant
Function Approximator*

5.1 Introduction

In chapter III, it was shown how to determine which features to use in a multilayer perceptron classifier. The previous chapter has demonstrated that there is a concrete link between the backpropagation learning rule and the extended Kalman filter algorithm. The question now is "Does a multilayer perceptron classifier have any relationship to other well-known classifiers?" This chapter will demonstrate that the multilayer perceptron when trained for classification using backpropagation is a minimum mean squared-error approximation to the Bayes optimal discriminant functions. This result shows conclusively that the multilayer perceptron is performing classification using the same rules as most traditional classifiers (29). The proof shown here was inspired by the work of Duda and Hart in their book, *Pattern Classification and Scene Analysis* (7).

Previous research in the area of pattern recognition has shown that multilayer perceptron, k -nearest neighbor and conventional non-parametric Bayesian classifiers yield the same classification accuracy, statistically speaking (18, 25, 26). This fact is also demonstrated by the performance levels shown in Table 4 on the FLIR imagery. These results have been empirical and, hence, are dependent on the data sets used. However, the consistently similar performance has led this author to investigate the theoretical connections. This chapter will show how a multilayer perceptron approximates the Bayes optimal discriminant function when used for classification.

The next section will show how a single output multilayer perceptron can be trained to approximate the Bayes optimal discriminant function. In section 5.3, this result will be extended to multiple classes. The following section will discuss some of the implications of these results for neural network architecture and training criteria.

5.2 Two Class Problem

In this section, the two class discrimination problem will be considered and it will be shown that a multilayer perceptron can be trained to approximate, in a mean squared-error sense, the Bayes optimal discriminant function.

Let $\mathbf{x}$ represent the feature vector which is to be classified. Define $F(\mathbf{x}, \mathbf{w})$ to be the output of the multilayer perceptron where $\mathbf{w}$ is the weight vector. Let the two classes be denoted ω_1 and ω_2 . Let $\mathcal{X}_i$ be the set of all possible feature vectors for class ω_i and $\mathcal{X}_1 \cup \mathcal{X}_2 = \mathcal{X}$. The set $\mathcal{X}$ represents the *ensemble* of all possible feature vectors that can be generated. Also, suppose for the two class problem the network is trained to produce $+1$ when the feature vector is from class ω_1 and -1 when the vector is from class ω_2 . These outputs are commonly achieved using the $\tanh^{-1}(\cdot)$ function as the output nonlinearity for the network nodes. The Bayes optimal discriminant function which minimizes probability of error can be written in many forms. Let $g_0(\mathbf{x})$ be the Bayes optimal discriminant function given by

$$g_0(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) \quad (20)$$

where $P(\omega_i|\mathbf{x})$ is the probability that $\mathbf{x}$ belongs to class ω_i . Hence, $g_0(\mathbf{x})$ is positive when $\mathbf{x}$ is most likely from class ω_1 and negative when $\mathbf{x}$ is most likely from class ω_2 . The distribution of feature vectors is governed by

$$p(\mathbf{x}) = p(\mathbf{x}|\omega_1)P(\omega_1) + p(\mathbf{x}|\omega_2)P(\omega_2)$$

where $P(\omega_i)$ is the *a priori* probability of class ω_i and $p(\mathbf{x}|\omega_i)$ is the conditional probability density function of $\mathbf{x}$ given that $\mathbf{x}$ is from class ω_i , so that $p(\mathbf{x})$ is the probability density function governing $\mathbf{x}$. In the following, a lower case $p(\cdot)$ always indicates a probability density function and an upper case $P(\cdot)$ represents a probability function.

It shall be shown that the following error criterion is minimized by backpropagation in the limit as the number of training vectors, P , becomes infinite.

$$\epsilon^2(\mathbf{w}) = \int_{\mathcal{X}} [F(\mathbf{x}, \mathbf{w}) - g_0(\mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x}$$

In other words, backpropagation finds a minimum mean squared-error approximation to the Bayes optimal discriminant function.

Suppose samples are generated by $p(\mathbf{x})$ such that $\mathbf{X}_1 = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{P_1}\} \subset \mathcal{X}_1$ and $\mathbf{X}_2 = \{\mathbf{x}_{P_1+1}, \mathbf{x}_2, \dots, \mathbf{x}_{P_1+P_2}\} \subset \mathcal{X}_2$ where P_i is the number of training vectors in class ω_i . Define the sample data error function, $E_s(\mathbf{w})$, as shown below:

$$E_s(\mathbf{w}) = \sum_{\mathbf{x} \in \mathbf{X}_1} (F(\mathbf{x}, \mathbf{w}) - 1)^2 + \sum_{\mathbf{x} \in \mathbf{X}_2} (F(\mathbf{x}, \mathbf{w}) + 1)^2$$

where $\mathbf{X}_i$ is the set of training vectors from class ω_i . It is well known that backpropagation approximately minimizes E_s with respect to $\mathbf{w}$ (14) (31:318-328) (35). The sample data error function, $E_s(\mathbf{w})$, describes a hyper-dimensional surface which backpropagation traverses seeking a minimum; hence, $E_s(\mathbf{w})$ is called the *sample data error surface*. Now let the average error, $E_a(\mathbf{w})$ be defined as follows:

$$E_a(\mathbf{w}) = \lim_{P \rightarrow \infty} \frac{1}{P} E_s(\mathbf{w})$$

where P is the total number of features vectors for both classes. The average error, $E_a(\mathbf{w})$, is the *ensemble error surface* for the pattern recognition problem since it represents the error surface when all possible feature vectors are included in the computation. When the minimum of the sample data error surface, $E_s(\mathbf{w})$, is sought, it is assumed that $E_s(\mathbf{w})$ represents a reasonable approximation to the ensemble error surface, $E_a(\mathbf{w})$. If, however, the sample data does not accurately represent the underlying probability density functions, then the minimum of $E_s(\mathbf{w})$ will not yield a classifier with comparable performance when tested with feature vectors not used in designing the classifier.

Now rewrite the expression for E_a using the number of vectors in each class.

$$E_a(\mathbf{w}) = \lim_{P \rightarrow \infty} \left[\frac{P_1}{P} \cdot \frac{1}{P_1} \sum_{\mathbf{x} \in X_1} (F(\mathbf{x}, \mathbf{w}) - 1)^2 + \frac{P_2}{P} \cdot \frac{1}{P_2} \sum_{\mathbf{x} \in X_2} (F(\mathbf{x}, \mathbf{w}) + 1)^2 \right]$$

where P_i is the number of feature vectors from class ω_i . As the total number of feature vectors, P , increases without bound, the number of vectors in the individual classes, P_i , becomes infinite for all classes with non-zero *a priori* probability. Thus, by the Strong Law of Large Numbers (23:258-263),

$$E_a(\mathbf{w}) = P(\omega_1) \int_{X'} (F(\mathbf{x}, \mathbf{w}) - 1)^2 p(\mathbf{x}|\omega_1) d\mathbf{x} + P(\omega_2) \int_{X'} (F(\mathbf{x}, \mathbf{w}) + 1)^2 p(\mathbf{x}|\omega_2) d\mathbf{x}$$

where $p(\mathbf{x}|\omega_i)$ is the conditional probability density function of the input $\mathbf{x}$ given that it is from class ω_i . Combining the integrals and gathering terms yields

$$E_a(\mathbf{w}) = \int_{X'} \{ [F^2(\mathbf{x}, \mathbf{w}) + 1] \cdot [p(\mathbf{x}|\omega_1)P(\omega_1) + p(\mathbf{x}|\omega_2)P(\omega_2)] - 2F(\mathbf{x}, \mathbf{w})[p(\mathbf{x}|\omega_1)P(\omega_1) - p(\mathbf{x}|\omega_2)P(\omega_2)] \} d\mathbf{x} \quad (21)$$

Using Bayes Formula, the above expression can be simplified and the optimal discriminant function given by Equation 20 can be introduced. Note that

$$g_0(\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}|\omega_1)P(\omega_1) - p(\mathbf{x}|\omega_2)P(\omega_2)$$

$$p(\mathbf{x}|\omega_1)P(\omega_1) + p(\mathbf{x}|\omega_2)P(\omega_2) = p(\mathbf{x})$$

Thus Equation 21 becomes

$$\begin{aligned} E_a(\mathbf{w}) &= \int_{X'} [F^2(\mathbf{x}, \mathbf{w})p(\mathbf{x}) - 2F(\mathbf{x}, \mathbf{w})g_0(\mathbf{x})p(\mathbf{x})] d\mathbf{x} + 1 \\ &= \int_{X'} [F(\mathbf{x}, \mathbf{w}) - g_0(\mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x} + \left\{ 1 - \int_{X'} g_0^2(\mathbf{x})p(\mathbf{x}) d\mathbf{x} \right\} \\ &= \epsilon^2(\mathbf{w}) + \left\{ 1 - \int_{X'} g_0^2(\mathbf{x})p(\mathbf{x}) d\mathbf{x} \right\} \end{aligned} \quad (22)$$

Now since backpropagation is minimizing E_s with respect to $\mathbf{w}$, it is also minimizing E_a with respect to $\mathbf{w}$. Since the term in braces in Equation 22 is independent of $\mathbf{w}$, minimizing E_a with respect to $\mathbf{w}$ also minimizes ϵ^2 . Hence, backpropagation yields a multilayer perceptron which is a minimum mean squared-error approximation to the Bayes optimal discriminant function.

The next section will extend this result to a multiclass discrimination problem and show that the outputs of the multilayer perceptron also represent the Bayes optimal discriminant functions and that the outputs can be interpreted as *a posteriori* probabilities.

5.3 Multiclass Problem

This section will show that the output functions of a multilayer perceptron approximate the Bayes optimal discriminant functions for a multiclass recognition problem.

Let $F_i(\mathbf{x}, \mathbf{w})$ be the i th output of a multilayer perceptron, $i = 1, \dots, N$ (N class problem). Also, let the desired output be one when $\mathbf{x} \in \mathcal{X}_i$ and zero otherwise. That is,

$$d_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_i \\ 0 & \text{otherwise} \end{cases}$$

The Bayes optimal discriminant functions are given by

$$g_i(\mathbf{x}) = P(\omega_i | \mathbf{x}) \quad \text{for all } i = 1, 2, \dots, N$$

and the decision rule is: Decide $\mathbf{x}$ is from class ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$. The Bayes optimal discriminant functions are optimal in the sense that probability of error is minimized by their use.

Define the sample data error function similarly to that for the two class problem:

$$E_s(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}_1} \left[\sum_{j \neq 1} F_j^2(\mathbf{x}, \mathbf{w}) + (F_1(\mathbf{x}, \mathbf{w}) - 1)^2 \right] + \dots +$$

$$\sum_{x \in \mathbf{X}_N} \left[\sum_{j \neq N} F_j^2(\mathbf{x}, \mathbf{w}) + (F_N(\mathbf{x}, \mathbf{w}) - 1)^2 \right]$$

The average error becomes

$$\begin{aligned} E_a(\mathbf{w}) &= \lim_{P \rightarrow \infty} \frac{1}{P} E_s(\mathbf{w}) \\ &= \lim_{P \rightarrow \infty} \left\{ \frac{P_1}{P} \cdot \frac{1}{P_1} \sum_{x \in \mathbf{X}_1} \left[\sum_{j \neq 1} F_j^2(\mathbf{x}, \mathbf{w}) + (F_1(\mathbf{x}, \mathbf{w}) - 1)^2 \right] + \dots \right\} \end{aligned}$$

Applying the Strong Law of Large Numbers and simplifying yields the following

$$\begin{aligned} E_a(\mathbf{w}) &= P(\omega_1) \left[\int_{\mathcal{X}} (F_1(\mathbf{x}, \mathbf{w}) - 1)^2 p(\mathbf{x}|\omega_1) d\mathbf{x} + \int_{\mathcal{X}} F_2^2(\mathbf{x}, \mathbf{w}) p(\mathbf{x}|\omega_1) d\mathbf{x} + \dots \right. \\ &\quad \left. + \int_{\mathcal{X}} F_N^2(\mathbf{x}, \mathbf{w}) p(\mathbf{x}|\omega_1) d\mathbf{x} \right] + \dots + P(\omega_N) [\dots] \\ &= \sum_{i=1}^N \left[\int_{\mathcal{X}} (F_i(\mathbf{x}, \mathbf{w}) - 1)^2 p(\mathbf{x}|\omega_i) P(\omega_i) d\mathbf{x} + \sum_{j \neq i} \int_{\mathcal{X}} F_j^2(\mathbf{x}, \mathbf{w}) p(\mathbf{x}|\omega_j) P(\omega_j) d\mathbf{x} \right] \\ &= \sum_{i=1}^N \left\{ \int_{\mathcal{X}} F_i^2(\mathbf{x}, \mathbf{w}) \left[\sum_{j=1}^N p(\mathbf{x}|\omega_j) P(\omega_j) \right] d\mathbf{x} - \right. \\ &\quad \left. \int_{\mathcal{X}} [2F_i(\mathbf{x}, \mathbf{w}) - 1] p(\mathbf{x}|\omega_i) P(\omega_i) d\mathbf{x} \right\} \end{aligned} \quad (23)$$

By Bayes Law the following identities hold

$$\sum_{j=1}^N p(\mathbf{x}|\omega_j) P(\omega_j) = p(\mathbf{x})$$

$$\begin{aligned} p(\mathbf{x}|\omega_i) P(\omega_i) &= p(\mathbf{x}, \omega_i) \\ &= P(\omega_i|\mathbf{x}) p(\mathbf{x}) \\ &= g_i(\mathbf{x}) p(\mathbf{x}) \end{aligned}$$

Thus, Equation 23 becomes

$$\begin{aligned} E_a(\mathbf{w}) &= \sum_{i=1}^N \left[\int_{\mathcal{X}} \{F_i^2(\mathbf{x}, \mathbf{w})p(\mathbf{x}) - [2F_i(\mathbf{x}, \mathbf{w}) - 1]g_i(\mathbf{x})p(\mathbf{x})\} d\mathbf{x} \right] \\ &= \sum_{i=1}^N \left[\int_{\mathcal{X}} [F_i(\mathbf{x}, \mathbf{w}) - g_i(\mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x} + \left\{ \int_{\mathcal{X}} g_i(\mathbf{x})(1 - g_i(\mathbf{x}))p(\mathbf{x}) d\mathbf{x} \right\} \right] \quad (24) \end{aligned}$$

Note that the term in braces is independent of $\mathbf{w}$; hence, backpropagation minimizes the following error criterion

$$\epsilon^2(\mathbf{w}) = \sum_{i=1}^N \int_{\mathcal{X}} [F_i(\mathbf{x}, \mathbf{w}) - g_i(\mathbf{x})]^2 p(\mathbf{x}) d\mathbf{x}$$

Thus, the output functions of a multilayer perceptron approximate the Bayes optimal discriminant functions in the minimum mean squared-error sense. The next section will discuss the implications of this result.

5.4 Discussion

It has been shown that backpropagation provides a minimum mean squared-error approximation to the Bayes optimal discriminant functions for both the two class problem and the multiclass problem. There are several implications of this result which will now be discussed.

An important conclusion is that the outputs of the multilayer perceptron when trained as previously described represent *a posteriori* probabilities. That is,

$$F_i(\mathbf{x}, \mathbf{w}) \approx P(\omega_i|\mathbf{x})$$

This fact makes it possible to set sensible decision thresholds for the outputs of a multilayer perceptron. For example, a rejection criterion could be set that specifies the input be rejected if the *a posteriori* probability of the indicated class is below 95 percent. Such criteria can be set not knowing what the outputs represent, but without a proper

interpretation of the outputs the criteria are at best *ad hoc*. Note, however, that how closely the multilayer perceptron approximates the *a posteriori* probabilities depends on the architecture of the network and the functional form of the underlying probability density functions. If there are insufficient units in the hidden layer(s) of the multilayer perceptron to accurately model the *a posteriori* probability functions, then the network's outputs will be poor approximations to the actual probability functions.

The second result of the foregoing is that this proof is not restricted to the multilayer perceptron. The specific architecture of the network was never used in the derivation. Hence, the result applies to any technique which attempts to minimize the mean squared-error and the desired outputs are 0 and 1 (for the multiclass problem).

Finally, recall Equations 22 and 24 for the two class and multiclass problems respectively. In both equations, the error between the Bayes optimal discriminant function and the multilayer perceptron's approximation is weighted by the probability density of the feature vectors, $p(\mathbf{x})$. Hence, the multilayer perceptron's output will more closely approximate the Bayes optimal discriminant function where $p(\mathbf{x})$ is large. If the goal is to minimize the probability of error, the fit between the multilayer perceptron and the Bayes optimal discriminant function should be better where $g_0(\mathbf{x}) = 0$, in the two class problem, and where $g_i(\mathbf{x}) = g_j(\mathbf{x})$ in the multiclass case since these conditions determine the decision boundaries of the classifier. In general, these conditions do not occur where $p(\mathbf{x})$ is large (see Figure 30 for an example).

5.5 Conclusion

This chapter has shown that the multilayer perceptron trained using backpropagation approximates the Bayes optimal discriminant functions for both two class and multiclass recognition problems. Most importantly, it has been shown that the outputs of the multilayer perceptron approximate the *a posteriori* probability functions when trained using backpropagation for the multiclass problem. In fact, the proof does not depend on the architecture of the network and is, hence, applicable to any network that minimizes

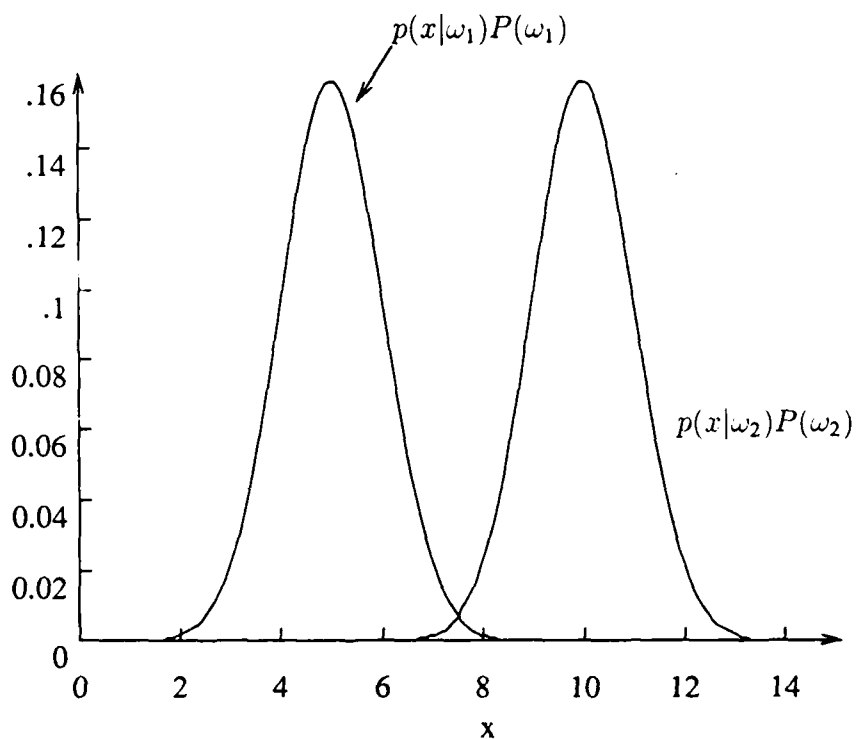


Figure 30. Example showing that the decision boundaries do not necessarily occur where the density function is large. The decision boundary occurs where the weighted density functions crossover; yet the overall density function, $p(x) = p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)$, is greatest at the peaks of the individual density functions.

the mean squared-error measure. The conclusion is that, contrary to popular opinion, these neural networks are simply another method for estimating the probability density function of the input feature vectors. There is really nothing magical or mysterious about these techniques. A multilayer perceptron simply provides a powerful architecture for function approximation. Recently, the work of Cybenko has shown that a multilayer perceptron with a single hidden layer can uniformly approximate any continuous function with support in the unit hypercube when sigmoidal output functions are used in the hidden layer (5). Hence, these networks are reasonable architectures with which to attempt probability density function estimation.

VI. Application to Multisensor Automatic Target Detection

6.1 Introduction

It has now been shown in many examples that the multilayer perceptron can be used effectively as a classifier, that the training method is closely related to the well-known extended Kalman filtering algorithm, and that the classifier approximates the Bayes optimal discriminant functions. In this chapter, the multilayer perceptron classifier will be applied to an Air Force problem, and its performance will be compared to a conventional approach. Roggemann showed that multisensor fusion provides a statistically significant increase in performance using conventional techniques (19). The question is "Can a multilayer perceptron be used for this problem?" This chapter will show that the multilayer perceptron can be used effectively for multisensor fusion and that its performance is statistically the same as Roggemann's conventional Bayesian approach.

This chapter will demonstrate the fusion of information from multiple sensors for the discrimination of targets from non-targets. The need for multiple sensors has been acknowledged as necessary for many military applications (2). Effective algorithms for using the information from several sensors need to be developed.

The following section will provide background information on the problem to be addressed. In Section 6.3, a simple technique for multisensor fusion will be described and applied to real sensor data. It will be shown that statistically significant benefits can be derived from the fusion process.

6.2 Problem Description

In this section, the problem of target recognition using multiple sensors and the approach to its solution used here will be described. The problem of interest is to discriminate targets from non-targets in absolute range and forward looking infrared (FLIR) imagery. Targets consist of tanks, trucks, and armored personnel carriers (APCs).

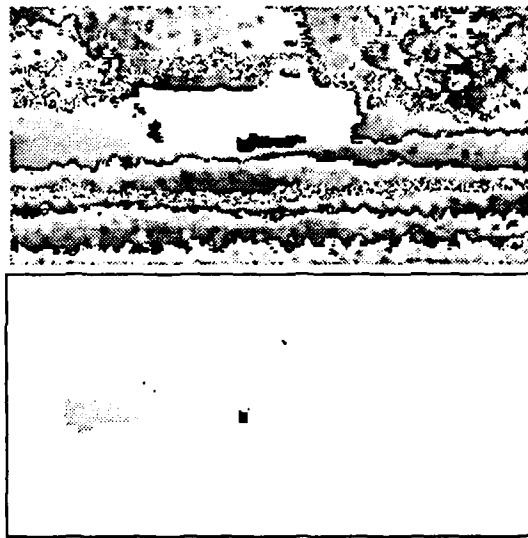


Figure 31. Typical Range Image and its Segmented Version

The absolute range imagery consists of 32 bits per pixel of range information with an angular resolution of 0.05-0.20 milliradians depending on the sensor field-of-view setting. The FLIR imagery has 8 bits per pixel with an angular resolution of 0.186-0.56 milliradians. Targets were viewed at distances from 860 meters to 1700 meters.

The goal is to find a classifier which can fuse the information from multiple sensors. In this case, there are two streams of information, one from the range sensor and one from the FLIR sensor.

The segmenter used for the absolute range imagery was developed by Roggemann (19:42-68). This segmenter uses small scale planarity to separate suspected targets from the background. A typical range image and its segmented version are shown in Figure 31. A histogram-based algorithm provided the FLIR imagery segmentation. This algorithm was also developed by Roggemann (19). Figure 32 shows a typical FLIR image and the output of the segmenter.

A variety of features were initially computed by the feature extractor for each sensor. The features chosen have been shown to be useful in image recognition. The

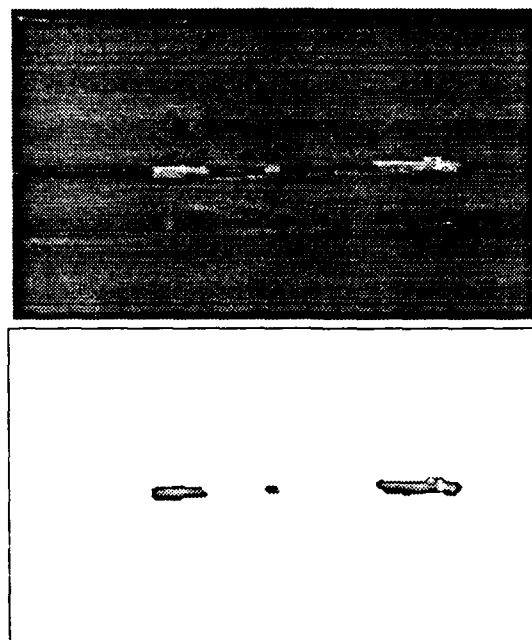


Figure 32. Typical FLIR Image and its Segmented Version

features computed from each sensor are listed in Table 7. These are the features used by Roggemann (19:71-76,101). The correspondence feature listed last in the table was developed by Roggemann (19:77-82). The correspondence feature is discrete valued and is dependent on the spatial correspondence of blobs segmented in the dominant and non-dominant sensors. The dominant sensor provides the initial blobs which are classified as either target or non-target. The non-dominant sensor segmented image is searched in the spatial locations corresponding to segmented blobs in the dominant sensor. If there is a segmented blob in the same spatial location in the non-dominant sensor as the dominant sensor, then features are extracted from the non-dominant sensor blob for use in classifying the dominant sensor blob.

Three different classifiers were evaluated for fusing the information from the sensors. A multilayer perceptron, the k -nearest neighbor algorithm and a non-parametric Bayesian classifier using Parzen windows were applied to the classification problem. The k -nearest neighbor algorithm and non-parametric Bayesian classifier are well understood

Table 7. Features Used for Multisensor Target Detection

FLIR Features	
Feature	Description
Complexity	Ratio of border pixels to total object pixels
Length-to-Width Ratio	Ratio of object length to width
Contrast of Means	Contrast ratio of object mean to local background mean

Absolute Range Features	
Feature	Description
Length-to-Width Ratio	Same as above
Absolute Difference of the Standard Deviations	Absolute difference between standard deviation of object pixels to local background pixels
Complexity	Same as above

Multiple Sensor Feature	
Feature	Description
Correspondence	A multiple sensor feature developed by Roggemann (19:77-82) (see text, p. 88)

statistical classifiers. They were used to provide a baseline comparison with the neural network. The performance of the multilayer perceptron is also compared with the results of Roggemann's conventional non-parametric Bayesian approach using histograms to estimate the density functions (19).

6.3 *The Multisensor Fusion Approach*

In the previous section, the problem at hand was described. In this section, a simple method for feature level fusion will be proposed and tested on a set of range and FLIR imagery.

To perform the fusion, one of the sensors is assumed to be dominant. The dominant sensor provides the initial blobs. The non-dominant sensor is then checked to see if there are any blobs which correspond to the one under consideration in the dominant sensor. For the database used, only the range sensor was used as the dominant sensor because in the FLIR database out of 550 blobs only 176 were located such that the range sensor had a view of that region in space. Of these 176 FLIR blobs, only one was misclassified using the FLIR features alone; hence, there was negligible opportunity for the range sensor to aid the FLIR sensor. On the other hand, all 400 blobs in the range database were positioned so that there was a corresponding view in the FLIR. The range sensor alone was able to classify correctly 85 percent of the blobs. Hence, there was room for improvement with the addition of the FLIR sensor. The need for a dominant sensor is not a severe limitation. It simply means that each sensor will have its own neural network to classify detections found in the sensor with the aid of any non-dominant sensors.

The input to the network is the range and FLIR feature vectors concatenated with the correspondence feature. When no blob is available in the FLIR for a given range blob, the FLIR vector is set to zero. These vectors are then normalized using the Gaussian technique described in Chapter II (p. 16). Note that in the case when a corresponding blob in the FLIR is not found and zeros are substituted, the actual input to the network is non-zero due to the normalization technique applied. However, the correspondence

Table 8. Correspondence Feature Normalization

Value	Network Input
No Correspondence	1/4
Weak-Weak Correspondence	1/2
Weak Correspondence	3/4
Strong Correspondence	1

feature was not normalized using the Gaussian procedure. Instead, the correspondence feature was normalized as shown in Table 8.

For the following results, the hold one out method (6:356-357) was applied to the database which consisted of 400 concatenated feature vectors. Table 9 gives the results of applying the hold one out method to the range dominant database using the multilayer perceptron and Roggemann's approach. The performance with and without the Roggemann correspondence feature is shown for the multilayer perceptron. The multilayer perceptron was trained with a learning rate of $\eta = 0.3$ and a momentum factor of $\alpha = 0.8$. Note that Roggemann used the same database of images and features and also the same test methodology, namely, hold one out. The addition of the FLIR information provides a substantial improvement in classifier performance. The table shows the 95 percent classifier confidence interval estimate of the true accuracy rate. Since for each classifier the confidence interval for the multisensor fusion accuracy is higher than and non-overlapping with the range only accuracy, then the addition of the FLIR sensor provides a statistically significant improvement in performance. Note that the addition of the correspondence feature is only of marginal utility for the multilayer perceptron classifier. More importantly, there is no statistical difference between the multilayer perceptron approach and Roggemann's conventional approach.

The technique applied for fusion of information in a multilayer perceptron is not restricted to that architecture. In fact, the fusion can also be performed using a k -nearest neighbor approach or a non-parametric Bayesian classifier using Parzen windows.

Table 9. Multisensor Target Detection Accuracy Results Compared With Roggemann

Classifier	Range Only Accuracy	Range+FLIR Accuracy
	(95% Confidence Intervals)	
Multilayer Perceptron(6-5-2) without correspondence feature	82.0-89.0%	90.2-95.3%
Multilayer Perceptron(7-5-2) with correspondence feature	82.0-89.0%	92.6-96.9%
Roggemann's Conventional Bayesian Approach (19:102)	83.0-89.8%	91.7-96.3%

Table 10. Multisensor Target Detection Accuracy Results Compared With Statistical Classifiers

Classifier	Range Only Accuracy	Range+FLIR Accuracy
	(95% Confidence Intervals)	
Multilayer Perceptron(6-5-2) ($\eta = 0.3, \alpha = 0.8$)	82.0-89.0%	90.2-95.3%
k -nearest neighbor ($k = 3$)	81.8-88.7%	91.1-95.9%
Non-parametric Bayesian with Parzen Windows ($\sigma = 0.5$)	83.7-90.3%	91.7-96.3%

Table 10 shows the estimated true accuracy rate of the three classifiers on the Roggemann target/non-target database but not using the correspondence feature. As in the previous table, the hold one out method was used and the 95 percent confidence interval is shown. For the k -nearest neighbor algorithm, the best performance on the multiple sensor case was achieved using $k = 3$ out of the tested values of $\{1,3,5,7,9\}$. The width of the Gaussian Parzen window was also varied from 0.1 to 1.5 by increments of 0.2. The best multiple sensor performance was achieved using a width of $\sigma = 0.5$. The table clearly shows that there is no statistically significant difference between the three classifiers which is not surprising since as shown in Chapter V the multilayer perceptron classifier is approximating the Bayes optimal discriminant functions.

6.4 Conclusion

In this chapter, the problem of fusing information from multiple sensors for pattern recognition has been addressed. The discrimination between targets and non-targets in absolute range and forward looking infrared (FLIR) imagery was examined. A novel approach to sensor fusion was developed and was shown to provide a significant improvement in target detection over the single sensor approach. Although the example shown fuses information from only two sensors, the technique is easily extended to any number of additional non-dominant sensors. Also, the performance of the multilayer perceptron is statistically the same as the conventional approach used by Roggemann and other statistical classifiers which further supports the results of Chapter V which showed the equivalence between the multilayer perceptron classifier and the Bayes optimal discriminant functions. Roggemann's correspondence feature was found to be of little use for the multilayer perceptron given that its performance was not significantly degraded without the feature using the confidence intervals technique to define significant.

VII. Recommendations and Conclusions

7.1 Recommendations

There are, of course, several avenues of research remaining which could be followed in the future with worthwhile benefits. First, the feature selection technique developed in Chapter III can be naturally extended to pruning not only the inputs to a multilayer perceptron but also the internal nodes and weights. The problem of selecting the architecture of a network for a given situation continues to elude researchers. To date, no theoretical work has been able to specify how many nodes are needed or when multiple hidden layers would be beneficial. Neither have automated experimental techniques been generally accepted for the determination of network architecture. This area is ripe for further research.

An exhaustive comparative study of the relative merits of statistical and multilayer perceptron classifiers needs to be undertaken. In this dissertation, it has been shown that no significant difference in classification performance can be expected between the statistical classifiers and the multilayer perceptron. However, many questions remain unanswered. These competing designs should be compared based on the number of free parameters, ease of implementation, cost, computational complexity, and generalization as a function of training set size. Only after a study such as this will it be possible to say when a given type classifier should be chosen over another in a real world situation.

The neural network architecture studied here is basically static. The network is trained on an unchanging database and then the weights are fixed. If something occurs after training which changes the distribution of the data, the network will not be able to compensate for the change. Dynamical systems need to be undertaken now that will be continuously learning from the environment and are thus able to follow changes in the input distributions. For example, this capability would allow target recognizers to function effectively even under changing weather conditions.

7.2 Conclusions

A new technique for ranking the importance of features for a multilayer perceptron has been developed. The saliency measure has been shown empirically to be both consistent and useful. When compared with the traditional method of ranking input features by the probability of error criterion, it performed as well and resulted in a similar ranking for the input features. Also, backpropagation was compared to extended Kalman filtering using the saliency measure. It was found that both methods for setting the weights in a multilayer perceptron place equal relative emphases on the features. The saliency measure gives a clear indication that multilayer perceptrons are able to effectively ignore useless inputs as indicated by the test results. This fact gives insight into the behavior of multilayer perceptrons with respect to sensitivity to input features.

For the first time it has been demonstrated that backpropagation is closely related to the well known method of extended Kalman filtering for training the weights in a multilayer perceptron. That is, the backpropagation algorithm results from making certain simplifying assumptions in the extended Kalman filtering approach. It was found that backpropagation sacrifices a great deal of information about the weights that the Kalman filter uses. The examples have shown the Kalman filter can achieve higher accuracy, in many cases, after a fixed number of training cycles than backpropagation by using the information which backpropagation discards; however, this is not always the case as demonstrated on the final example with the absolute range data. Although backpropagation has poorer performance in terms of classification accuracy *versus* number of training iterations, when compared on the basis of numerical complexity, backpropagation is clearly superior to the extended Kalman filtering approach. Thus, the assumptions made by backpropagation are clearly beneficial.

Perhaps, the single most important result of this research is the proof that the multilayer perceptron trained using backpropagation for classification approximates the Bayes optimal discriminant functions. For the first time, it has been shown that this neural network is not doing anything unusual. It is simply another method for approximating

the underlying distribution of the data which is the goal of all classifiers. The result is a complete demystification of the multilayer perceptron classifier. Importantly, it has been shown that the outputs of the multilayer perceptron approximate the *a posteriori* conditional probability distribution functions when trained using backpropagation for the multiclass problem. In fact, the proof does not depend on the architecture of the network and is, hence, applicable to any network that minimizes the mean squared-error measure.

Finally, a novel approach to sensor fusion was developed and was shown to provide a significant improvement in target detection over the single sensor approach. The approach is simple and can be easily extended to more than two sensors. A statistically significant improvement in target detection was demonstrated on the absolute range and FLIR imagery multiple sensor database. The sensor fusion technique can also be applied using other statistical classifiers. It was shown that the classification accuracy differences between the multilayer perceptron classifier and the statistical classifiers is not significant in practice which further bolsters the proof that the multilayer perceptron classifier is merely a minimum mean squared-error approximation to the Bayes optimal discriminant functions.

Appendix A. *Network Sizing Techniques*

This appendix will provide heuristic guidelines for determining the architecture of multilayer perceptron for a specific application. Rigorous mathematical techniques have not been developed to determine the appropriate number of hidden layers or the number of nodes in those layers for a given problem. The result is that the architecture must be selected in some *ad hoc* fashion. This appendix will attempt to describe the techniques used in this research to determine the appropriate architectures for the classification problems dealt with.

Initially, a baseline performance level should be established for the network to achieve. There are two cases to consider. If the data being used is synthetically generated, then the ideal solution to the problem is known. For example, in either the Exclusive-OR or Mesh problems the solution is known, and the optimal performance is known to be 100 percent accuracy. When sensor data is used, the optimal solution is generally not known. In these cases, the baseline performance must be specified using other criteria. Given that the multilayer perceptron classifier approximates the Bayes optimal discriminant functions, it is appropriate to use the performance of a non-parametric Bayesian classifier as the baseline. Optionally, the performance of the k -nearest neighbor classifier could be used as a baseline. If the application problem is not a classification one, then the baseline performance on real world data must be established using other methods.

After establishing a baseline performance level, the networks can be trained using the following approach:

- Start with a single layer network, that is, no hidden layers. In this case there are no parameters which can be varied other than the learning rate, η , and the momentum factor, α . If the problem is linearly separable, or almost so, this network will perform well.

- Train the network using a variety of training rates and momentum factors. Observe training out to approximately 10,000 iterations. If the performance index (accuracy rate or average total euclidean error) is highly erratic, the learning rate and momentum factor should be reduced. If learning is very slow, the learning rate may be increased.
- Evaluate the network. Does the performance level approach the baseline? If the performance level of the network is much below the baseline, increase the size of the network.
- Add a single hidden layer to the network. Start with a small number of nodes on the order of five.
- Train the network as before. Has the performance level been increased over that of the previous network? Does the performance level approach that of the baseline? If performance is still below that of the baseline, increase the number of nodes in the hidden layer. Performance should be monitored on both the training set and an independent test set. If the performance level of the training set is much better than the test set (memorization), then the network should be reduced in size. As the size of the network increases, expect the training to take longer.
- Evaluate the network. If the performance is not approaching that of the baseline with increases in the number of single hidden layer nodes, increase the number of hidden layers to two. Start with approximately five to ten nodes in each hidden layer.
- Train the network as before except training will now take longer. Train networks to at least 20,000 iterations before changing the architecture. Increase the number of nodes in each hidden layer maintaining a fixed number in one layer while increasing the other. Start by increasing the number of nodes in the first hidden layer and slowly increase the number in the second hidden layer.

- Select a few of the most promising architectures and train until performance stabilizes for each. Up to 500,000 iterations may be required to obtain peak performance from a given network.

After a network architecture has been found which provides near to baseline performance, the following criteria should be considered. First, does the number of free parameters in the network exceed the minimum requirements given by Cover? The number of free parameters should not exceed one-half the total number of training vectors (3). Second, does the network conform with the bounds given by Baum and Haussler? According to Baum and Haussler, the number of free parameters should be less than the product of the total number of training vectors and the desired error rate (1). The Baum bound should be considered desirable but not required since his results are based on networks with hard limiters.

Bibliography

1. Baum, Eric and David Haussler. "What Size Net Gives Valid Generalization?." In *Advances in Neural Information Processing Systems 1*, pages 81-90, San Mateo, CA: Morgan Kaufmann, November 1989.
2. Comparato, Vito G. "Fusion - The Key to Tactical Mission Success." In *Proceedings of SPIE Conference on Sensor Fusion*, Volume 931, pages 2-7, Bellingham, WA: Society of Photo-Optical Engineers, 1988.
3. Cover, Thomas M. "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transactions on Electronic Computers*, EC-14:326-334 (June 1965).
4. Cun, Yann Le, et al. "Optimal Brain Damage." In Touretzky, David S., editor, *Advances in Neural Information Processing Systems 2*, pages 598-605, San Mateo, CA: Morgan Kaufmann, 1990.
5. Cybenko, G. "Approximations by Superpositions of Sigmoidal Functions," *Mathematics of Control, Signals, and Systems* (1989). Accepted for publication.
6. Devijver, Pierre A. and Josef Kittler. *Pattern Recognition: A Statistical Approach*. London: Prentice Hall International, 1982.
7. Duda, Richard O. and Peter E. Hart. *Pattern Classification and Scene Analysis*. New York: John Wiley and Sons, 1973.
8. Dudani, S.A., et al. "Aircraft Identification By Moment Invariants," *IEEE Transactions on Computers*, C-26:39-45 (1977).
9. Foley, Donald H. "Considerations of Sample and Feature Size," *IEEE Transactions on Information Theory*, IT-18:618-626 (September 1972).
10. Maybeck, Peter S. *Stochastic Models, Estimation, and Control*, Volume 1. New York: Academic Press, 1979.
11. Maybeck, Peter S. *Stochastic Models, Estimation, and Control*, Volume 2. New York: Academic Press, 1982.
12. Minsky, M. and Papert. *An Introduction to Computational Geometry*. MIT Press, 1969.
13. Nilsson, Nils J. *Learning Machines*. New York: McGraw-Hill, 1965.
14. Parker, David B. *Learning-Logic*. Invention Report 581-64, Stanford, CA: Stanford University, October 1982.
15. Piazza, Charles C. *Modified Backward Error Propagation for Tactical Target Recognition*. MS thesis, Air Force Institute of Technology, 1988.

16. Robbins, H. and S. Monro. "A Stochastic Approximation Method," *Annals of Mathematical Statistics*, 22:400-407 (1951).
17. Roberts, John E. "Influence of Target-Vector Code Selection on the Performance of a Neural-Network Word Recognizer." In *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, page II.628, June 1989.
18. Rogers, Steven K., et al. "Artificial Neural Networks for Automatic Target Recognition." In *Proceedings of SPIE Conference on Applications of Artificial Neural Networks*, 1990.
19. Roggemann, Michael C. *Multiple Sensor Fusion for Detecting Targets in FLIR and Range Images*. PhD dissertation, Air Force Institute of Technology, Wright-Patterson AFB, OH, May 1989.
20. Roggemann, Michael C., et al. "An Approach to Multiple Sensor Target Detection." In *Proceedings of SPIE Conference on Sensor Fusion*, Volume 1100, 1989.
21. Roggemann, Michael C., et al. "Multiple Sensor Tactical Target Detection in FLIR and Range Images." In *Proceedings of Tri-Service Data Fusion Symposium*, May 1989.
22. Rosenblatt. *Principles of Neurodynamics*. New York: Spartan Books, 1959.
23. Ross, Sheldon. *A First Course in Probability*. New York: Macmillan Publishing Co., 1976.
24. Ruck, Dennis W. *Multisensor Target Detection and Classification*. MS thesis, Air Force Institute of Technology, 1987.
25. Ruck, Dennis W., et al. "Tactical Target Recognition: Conventional and Neural Network Approaches." In *Proceedings of 5th Annual Aerospace Applications of Artificial Intelligence Conference*, October 1989.
26. Ruck, Dennis W., et al. "Target Recognition: Conventional and Neural Network Approaches." In *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, 1989. Abstract Only.
27. Ruck, Dennis W., et al. "Feature Selection Using a Multilayer Perceptron," *Journal of Neural Network Computing* (1990). Accepted for publication in fall.
28. Ruck, Dennis W., et al. "Multisensor Fusion Target Classification." In *Proceedings of SPIE Symposium on Optics, Electro-Optics, and Sensors*, pages 14-21, 1988.
29. Ruck, Dennis W., et al. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," *IEEE Transactions on Neural Networks* (1990). Accepted for publication.
30. Ruck, Dennis W., et al. "Back Propagation: A Degenerate Kalman Filter?," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (June 1989). Submitted for publication.

31. Rumelhart, David E., et al. *Parallel Distributed Processing*, Volume 1: Foundations. Cambridge, Mass.: MIT Press, 1986.
32. Simon, Wayne E. and Jeffrey R. Carter. "Learning to Identify Letters with REM Equations." In Caudill, Maureen, editor, *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, pages I.727-730, January 1990.
33. Simon, Wayne E. and Jeffrey R. Carter. "Removing and Adding Network Connections with Recursive Error Minimization (REM) Equations." In *Proceedings of SPIE Conference on Applications of Artificial Neural Networks, Volume 1294*, April 1990.
34. Singhal, Sharad and Lance Wu. "Training Multilayer Perceptrons with the Extended Kalman Algorithm." In Touretzky, David S., editor, *Advances in Neural Information Processing Systems 1*, pages 133-140, Morgan Kaufmann, 1989.
35. Werbos, Paul J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD dissertation, Harvard University, Cambridge, Mass., 1974.
36. White, Halbert. "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, 1:425-464 (1989).
37. Winter, Rodney and Bernard Widrow. "MADALINE RULE II: A Training Rule for Neural Networks." In *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, pages I.401-408, July 1988.

Vita

Captain Dennis W. Ruck [REDACTED]

He was graduated from Beloit Memorial High School in 1978 and attended Beloit College and Northwestern University in Evanston, Illinois. Captain Ruck received the Bachelor of Science in Electrical Engineering degree from Northwestern in 1983. Following graduation, he attended the Air Force Officer Training School and received his commission in September. He was assigned to the Aeronautical Systems Division at Wright-Patterson AFB, Ohio where he worked initially as a Research and Development Engineer and later as a Flight Simulator Design Engineer. He was lead engineer on a program to design, build, and deliver two systems for training operators of the GBU-15, a TV-guided bomb. The systems were successfully deployed to Clark AB, Philippines and Lakenheath RAF in 1986. Captain Ruck entered the Air Force Institute of Technology in May, 1986 and was graduated in December 1987 with the degree Master of Science in Electrical Engineering. Upon completion of the masters program, he entered into the doctoral program at the Air Force Institute of Technology.

R [REDACTED] [REDACTED]

D [REDACTED]

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this report is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Project Room (0704-0188), Washington, DC 20503.				
1 AGENCY USE ONLY (Leave blank)	2 REPORT DATE December 1990	3 REPORT TYPE AND DATES COVERED Doctoral Dissertation		
4 TITLE AND SUBTITLE Characterization of Multilayer Perceptrons and their Application to Multisensor Automatic Target Detection			5 FUNDING NUMBERS	
6 AUTHOR(S) Dennis W. Ruck, Capt, USAF				
7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583			8 PERFORMING ORGANIZATION REPORT NUMBER AFIT/DS/ENG/90-2	
9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10 SPONSORING/MONITORING AGENCY REPORT NUMBER	
11 SUPPLEMENTARY NOTES				
12a DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited.			12b DISTRIBUTION CODE	
13 ABSTRACT (Maximum 200 words) ➤ The multilayer perceptron was extensively analyzed. A technique for analyzing the multilayer perceptron, the saliency measure, was developed which provides a measure of the importance of inputs. The method was compared to the conventional statistical technique of <i>best features</i> and shown to provide similar rankings of the input. Using the saliency measure, it is shown that the multilayer perceptron effectively ignores useless inputs and that whether it is trained using backpropagation or extended Kalman filtering, the weighting of the inputs is the same. The backpropagation training algorithm is shown to be a degenerate version of the extended Kalman filter. The extended Kalman algorithm is shown to outperform the backpropagation method in terms of classification accuracy <i>versus</i> training presentations; however, in terms of computational complexity, the backpropagation algorithm is shown to be highly efficient. The multilayer perceptron trained using backpropagation for classification is proved to be a minimum mean squared-error approximation to the Bayes optimal discriminant functions. A simple technique for sensor fusion is shown to provide a statistically significant improvement in performance using absolute range and forward looking infrared imagery for target detection over the single sensor case.				
14 SUBJECT TERMS ➤ Neural Nets, Pattern Recognition, Target Detection, Multisensors, Statistical Decision Theory, Artificial Intelligence			15 NUMBER OF PAGES 116	
			16 PRICE CODE	
17 SECURITY CLASSIFICATION OF REPORT Unclassified	18 SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19 SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20 LIMITATION OF ABSTRACT UL	