

AD-A243 873



AFIT/DS/ENG/91-01



DTIC
ELECTE
JAN 06 1992
S D D

Multi-Layered Feedforward Neural Networks for Image Segmentation

DISSERTATION

Gregory L. Tarr
Captain, USAF

AFIT/DS/ENG/91-01

92-00046



Approved for public release; distribution unlimited

92 1 2 055

AFIT/DS/ENG/91

Multi-Layered Feedforward Neural Networks for Image Segmentation

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**

Gregory L. Tarr, B.S.E.E, M.S.E.E.

Captain, USAF

December 1991

Approved for public release; distribution unlimited



AFIT/DS/ENG/91

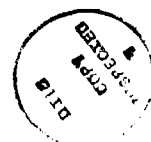
Multi-Layered Feedforward Neural Networks for Image Segmentation

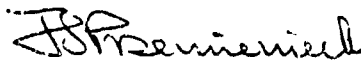
Gregory L. Tarr, B.S.E.E, M.S.E.E.

Captain, USAF

Approved:

	27 Nov 91
	27 Nov 91
Mark E. Osley	27 Nov 91
Dennis W. Ruck	27 NOV 91



 27 Nov. 1991

J. S. Przemieniecki

Dean, School of Engineering

Accession For	
NTIS	CRAB1 <input checked="" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Dist	Avail and/or Spec
A-1	

Abstract

Artificial neural network image segmentation techniques are examined. The biological inspired cortex transform is examined as a means to preprocess images for segmentation and classification. A generalized neural network formalism is presented as a means to produce common pattern recognition processing techniques in a single iterable element. Several feature reduction preprocessing techniques, based on feature saliency, Karhunen-Loève transformation and identity networks are tested and compared. The generalized architecture is applied to a problem in image segmentation, a tracking of high-value fixed tactical targets.

A generalized architecture for neural networks is developed based on the second order terms of the input vector. The relation between several common neural network paradigms is demonstrated using the generalized neural network. The architecture is demonstrated to allow implementation of many feedforward networks and several preprocessing techniques as well.

Because of the limited resources and large feature vectors associated with classification problems, several methods are tested to limit the size of the input feature vector. A feature saliency metric, weight saliency, is developed to assign relative importance to the individual features. The saliency metric is shown to be significantly easier to compute than previous methods. Several neural network implementations of identity networks are tested as a means to reduce the size of the feature vectors presented to classification networks.

Using the generalized approach, a scanning receptive field neural network is developed for image segmentation. The scanning window technique is used to segment sequential images of high value fixed targets (dams, bridges and power plants). The architecture is implemented as a tracking mechanism. The implementation resulted in an improved method for target identification and tracking, using neural networks as a front end.

Table of Contents

	Page
Abstract	iii
Table of Contents	iv
List of Figures	viii
List of Tables	xi
Preface	xii
I. Introduction	1
1.1 Historical Perspective	2
1.1.1 Pattern Recognition	2
1.1.2 Pattern Recognition and Neural Networks.	4
1.1.3 Computer and Biological Vision	6
1.1.4 Artificial Neural Networks for Image Analysis.	7
1.2 Problem Statement and Scope	9
1.2.1 The NeuralGraphics Neural Network Simulation En- vironment	10
1.2.2 Preprocessing Feature Vectors for Classification	10
1.2.3 Image and Feature Vector preprocessing	10
1.2.4 Local Area Transforms for Target Segmentation	11
1.3 Dissertation Organization	11

	Page
II. Cortex Transforms for Pattern Recognition and Image Analysis	12
2.1 Introduction	12
2.2 The Cortex Transform	12
2.3 The Exclusive-Or Problem.	17
2.4 The Mesh Problem	19
2.5 Cortex Transforms for Handwritten Characters.	21
2.6 Conclusion.	23
III. Generalized Neural Network Architectures	25
3.1 A Generalized Neural Network Algorithm	25
3.2 Perceptrons - Hyper-Plane Discrimination	28
3.3 Radial Basis Functions(RBF): Hyper-Ellipse Discrimination	33
3.4 Conic Basis Functions-Second Order Hyper-Surfaces	35
3.5 A Generalized Architecture	36
3.6 Generalized Neural Network Paradigm Implementations	38
3.7 Conclusion	39
IV. Feature Selection in Generalized Neural Networks	41
4.1 Why Limit Input Features?	41
4.2 Computing Feature Saliency	43
4.3 Weights as a Saliency Metric	44
4.4 Karhunen-Loève Networks for Feature Vector Reduction	51
4.5 The Discrete Karhunen-Loève Transformation	52
4.6 A Karhunen-Loève Network.	61
4.7 Identity Networks	73
4.8 Gram-Schmidt Type Networks	77
4.9 Conclusions	81

	Page
V. Generalized Neural Networks for Image Segmentation and Tracking . . .	87
5.1 Introduction.	87
5.2 Image Segmentation.	88
5.3 Segmentation using Self-Organization	90
5.4 Histogram and Gabor Filter Segmentation.	92
5.5 The Scanning Window Architecture.	97
5.5.1 Training the Neural Network.	98
5.5.2 Target Detection and Segmentation.	101
5.6 High Value Fixed Target Tracking using Neural Networks . .	106
5.7 Conclusions	108
VI. Conclusions and Recommendations	115
6.1 Conclusions	115
6.2 Recommendations	116
Appendix A. A General Purpose Environment for Neural Network Simulation	118
A.1 Introduction	118
A.2 Getting Started with the NeuralGraphics Environment	118
A.3 Running the Feedforward Tools	120
A.3.1 The Input Data File.	122
A.3.2 Function Approximation.	125
A.3.3 Interaction With the Network.	126
A.3.4 Statistics and Runtime Files(SG only)).	127
A.3.5 Additional Features.	128
A.3.6 The Setup File.	128
A.3.7 Running Neural Graphics in the Background.	129
A.4 Other Models and Demonstrations	130
A.4.1 Hybrid Propagation.	130

	Page
A.4.2 Kohonen Self-Organizing Maps.	131
A.4.3 The Traveling Salesman Problem.	131
A.4.4 Hopfield Associative Memory.	132
A.5 Conclusion	132
Bibliography	137
Vita	141

List of Figures

Figure	Page
1. Pattern Recognition Processes.	3
2. Neural Network Pattern Recognition.	5
3. The Cortex Transformation.	13
4. Cortex Coordinate Transformation of an Image.	14
5. The Exclusive-Or and the Mesh Problem.	15
6. XOR Problem Screen Displays.	16
7. Training Results for the XOR Problem.	18
8. Mesh Problem Screen Displays.	20
9. Mesh Problem Training Results.	21
10. Handwritten Letters with Cortex Transforms.	22
11. Character Recognition Error Reduction.	22
12. A Generalized Neural Network Input Node	26
13. General form for a Cybenko type network.	29
14. Output of a Perceptron	32
15. Conic Basis Function	37
16. Network Architecture for a Karhunen-Loève Identity Network.	53
17. Network Architecture for a Karhunen-Loève preprocessor.	54
18. Computing the Rotation Matrix.	61
19. Infrared Tactical Target Imagery.(34)	62
20. Gabor Segmented Data.(2)	62
21. Ruck Data using Karhunen-Loève Network	64
22. Roggemann Data using Karhunen-Loève Network.	65
23. Ruck Data using Karhunen-Loève mean Network.	67
24. Roggemann Data using Karhunen-Loève mean Network	68

Figure	Page
25. Ruck Data Using Fisher Linear Network.	69
26. Roggemann Data using Fisher Linear Network.	70
27. Ruck Data using identity networks	75
28. Roggemann Data Using Identity Networks	76
29. Ruck Data Using Gram-Schmidt Identity Network	78
30. Roggemann Data Using Gram-Schmidt Identity Network.	79
31. Ruck Data Using Normalized Identity Network.	82
32. Roggemann Data Using Normalized Identity Network.	83
33. Ruck Data Using Normalized Gram-Schmidt Network.	84
34. Roggemann Data Using Normalized Gram-Schmidt Network.	85
35. Classic Texture Segmentation.	89
36. Gross Segmentation by Radial Basis Function	92
37. Object Segmentation Using Gabor Filter Correlation.	93
38. Histogram Concavity Method.	94
39. Histogram Concavity Segmentation	95
40. Gabor Filter Segmenting Using a Scanning Window.	96
41. Training in the Scanning Window Architecture.	99
42. Scanning Window Architecture Image Propagation.	100
43. Comparison of Methods: Night Vision Lab Imagery.	102
44. Comparison of Methods: Bi-modal Histogram.	104
45. Comparison of Methods: Cluttered Data with Uni-modal Histogram.	105
46. Segmenting Edges Using the Scanning Window.	107
47. Approach on a Hydro-Electric Power Plant(1).	109
48. Approach on a Hydro-Electric Power Plant(2).	110
49. Approach on a Hydro-Electric Power Plant(3)	111
50. Approach on a Hydro-Electric Power Plant(a).	112
51. Approach on a Hydro-Electric Power Plant(b).	113

Figure	Page
52. The Opening Menu of the PC NeuralGraphics Software.	120
53. A Three Layer Network Display on a Silicon Graphics 3120.	121
54. Neural Net O-Scope on a Silicon Graphics	123
55. PC Feed forward models screen display	124
56. Data file formats for class and vector data.	125
57. A Hybrid Propagation Network	134
58. Example Setup File.	134
59. The Kohonen Self-Organizing Map	135
60. The Traveling Salesman Problem	135
61. The Hopfield Associative Memory	136

List of Tables

Table		Page
1.	Exclusive-Or Problems.	19
2.	Exclusive-Or Saliency	47
3.	Node Saliency for the Ruck data using weight vector magnitude.	48
4.	Comparison of Results Between Method for Selecting Saliency.	49
5.	Node Saliency for the Ruck Data using second order terms.	50
6.	Ruck Data Generalization	71
7.	Roggemann Data Generalization	72
8.	Performance of the Gram-Schmidt Network.	80

Preface

WILLIAM SHAKESPEARE 1564-1616

Is this a dagger which I see before me, The handle toward my hand? Come, let me clutch thee: I have thee not, and yet I see thee still. Art thou not, fatal vision, sensible to feeling as to sight? Or art thou but a dagger of the mind, a false creation, proceeding from the heat-oppressed brain?

MACBETH Act Two, Scene One

The difference between mysticism and scholarship can be confused by careless observation of the natural world. For centuries man has observed nature in hopes of solving problems that people can solve but machines cannot. Computer vision is one of these problems.

For example, simple visual classification problems solved by any dog, cat or even pigeon, are far beyond the reach of computer vision systems. We sometimes look to natural objects in hopes of finding solutions to these difficult problems. Mysticism is looking at a natural object and seeing an illusion, while scholarship is looking at an object and understanding what is there.

Investigation of biological systems, in hopes of finding inspiration requires a great leap of faith. Does the inspiration follow careful observation, or does the inspired suffer from *a false creation, proceeding from the heat-oppressed brain?* A working system is not proof that the system is based on a biological model. The difference is the difference between inspiration and understanding.

Novel, new technologies allow the perception to endure that present-day science is beyond mixing mysticism and scholarship. Biological information processing is based on a massively parallel interconnection of small computation elements called neurons.

The sequential processing of information by digital computers probably will never allow real-time processing speech or image information for anything more than toy problems. Understandably, many have turned to biological inspired, parallel models of computation.

The possibility that man will ever understand any more about himself than a machine understands about itself seems unlikely. Still, it may be worth the effort to look into the biological models to find efficient algorithms for information processing. Keep in mind that the inspiration may have little or nothing to do with the biological process. The results of these efforts may simply be new mysticism, with new priests and hi-tech crystal balls.

The point is simply that presently, there appears to be nothing neural about artificial neural nets, but they seem to be useful anyway.

I wish to acknowledge my indebtedness to my committee chairman, Dr. Steven K. Rogers, without whose tireless support and enthusiasm, this work could not have been successful. Also, I would like to thank my committee members: Dr. Matthew Kabrisky, Dr. Mark E. Oxley, and Dr. Paul Morton whose encouragement and support were always welcome. As with any computer intensive project a great deal of the credit goes to those who keep the machines running, for which I thank the system engineers, Dan Zambon and Bruce Clay and David Doak. Also, a special thanks goes to those who have come before, upon which much of this work was based, Dr. Dennis Ruck has been a source of vast technical expertise and inspiration. Capt Kevin Priddy has contributed a great deal to this project for which I am especially indebted.

Gregory L. Tarr

Multi-Layered Feedforward Neural Networks for Image Segmentation

I. Introduction

The United States Air Force and the Department of Defense have long been interested in the problem of autonomous guidance for conventional weapons. Such systems, sometimes called "fire and forget", allow weapons to choose their own target and guide themselves without human intervention. Many technical problems must be solved before such *smart weapons* can be effective. This research looks at some of the fundamental problems associated with autonomous guidance.

Solutions to these problems are dependent upon such fields as computer vision and pattern recognition. Computer vision algorithms use a hierarchical analysis of imagery to convert gray scale pixel information into targeting coordinates. The purpose of this dissertation is to examine a class of pattern recognition problems called image segmentation. The particular problem being studied is to take a series of images, segment out the objects of interests and select a target based on that segmentation.

The difference between this effort and previous efforts is the use of an artificial neural network (ANN) in the front end of the vision system. ANN's allow a targeting computer to be programmed without long sequences of instructions. Instead, endless *if,then,else* constructs are replaced with a massively parallel distributed processing engine. The processing engine learns by examples, much the way humans learn. The ANN is given examples of different classes and learns how to distinguish between them.

This dissertation has resulted in the development of a software environment for neural network research, a deeper understanding on how artificial neural networks make decisions, and a new method for targeting high-valued fixed targets in digital imagery. The software environment accepts a wide range of data sets to be evaluated for neural

network implementations. Building the environment resulted in new information concerning exactly how neural networks make decisions, and what can be done to improve and simplify them. The environment was applied to a specific set of sequential imagery of tactical targets and a neural network solution was found for converting the image into target coordinates.

The next section will discuss the background material which relates target identification and tracking, the focus of this effort, to conventional pattern recognition schemes.

1.1 Historical Perspective

1.1.1 Pattern Recognition Identification and classification of targets from electronic imagery is a difficult problem because the vast amounts of data involved. A single image can contain millions of bits of information (pixels), all of which need to be processed in a systematic way. Processing images for pattern recognition is a threefold problem. First, the targets must be separated from the background or segmented. The first stage is the most difficult part of the problem. Second, the data must be reduced to a manageable size, commonly called vector quantization, or feature extraction. This reduction in data is possible by selecting specific features of a pattern and using only these features for classification. Feature extraction is the process of taking meaningful measurements of the segmented blob. Good pattern recognition requires good features. The final task is classification of the vectors. The output of the classification phase is an identified object with its location. Figure 1 illustrates the process.

Pattern recognition and neural network literature is filled with work describing the use of neural networks for the final stage, or back-end, classification for recognition tasks. The back-end takes the extracted features and makes the classification. This work investigates the use of neural networks for the preprocessing (front-end) step in image understanding.

The first stage is the most difficult part of the problem.

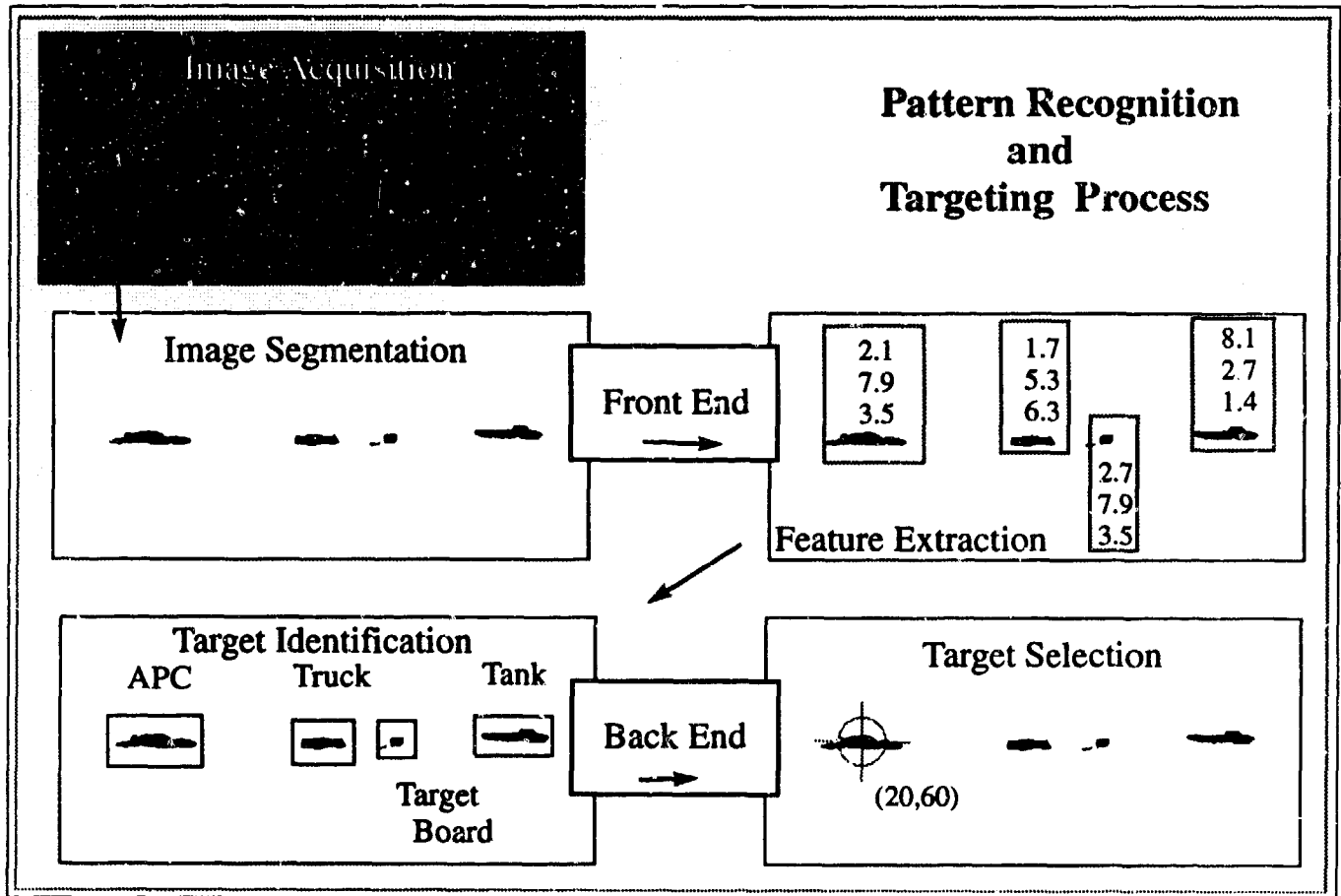


Figure 1. Pattern Recognition Processes.

1.1.2 Pattern Recognition and Neural Networks. Ruck (36) uses neural networks to classify tactical targets taken from infrared and laser range data. He extends Roggemann's work (34) in which he classified similar data using conventional Bayesian techniques in a multi-sensor fusion approach. Neural networks have been used in place of conventional classifiers for sonar return identification(14), speech recognition(26), and radar signal identification (50), to name a few.

Each of these tasks requires separation of exemplar patterns into classes. This type of system, sometimes called, a "neural network back-end", is a substitute for a conventional classifier. Front-end analysis, or segmentation of the target from its background and generating feature vectors, has traditionally depended on heuristic, or rule-based approaches. Algorithms based on neural network techniques may hold some computational advantage over conventional methods.

Neural networks can be considered a blackbox from the designers point. As shown in Figure 2, a feature vector is presented to the input of the neural network and a classification is generated at the output. The literature shows neural network back-ends work relatively well. In fact, Ruck(38) has shown that in training a neural network, using a mean squared error criterion, the algorithms attempt to minimize the difference between a neural network and a conventional Bayes optimal classifier. He shows that neural networks can be used interchangeably with Bayesian classifiers.

Still, a neural net back-end attacks only the least demanding part of the problem. A better approach might be to use neural networks for the front end image analysis. Neural networks could be used not only to classify the feature vectors but compute them from the raw imagery as well.

Using a neural network for the front end is not a new idea. Many have used raw pixel data as feature vectors. Le Cun (6) suggests that front end processing of imagery is not possible yet. His work in classifying handwritten characters suggests that this approach is not extensible beyond the most simple problems. While simple problems such as identification of ships at sea, or aircraft in clear air can be solved with automatic image

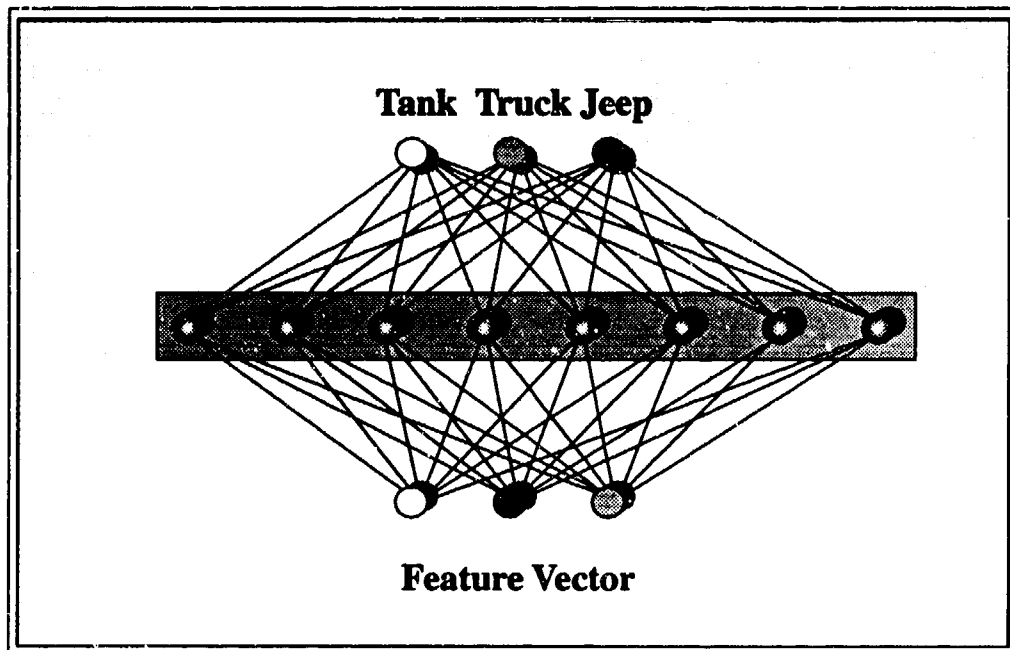


Figure 2. Neural Network Pattern Recognition. The diagram shows the general form of a neural network. The feature vector is entered at the bottom. The lines represent weights connecting the input to the hidden layer(s). The nodes in the hidden layer contain a summation of the input values times the weight followed by the application of a non-linear function. The hidden node values are passed in a similar fashion to the output. The classification is made by selecting the output node with the highest value. The output layer contains one node for each output class.

recognition technology, more complex problems cannot. For example: moving or fixed targets in cluttered backgrounds are still the subject of early exploratory development and probably will require a "man in the loop".

Advances in computer vision systems and sensors have driven the need for increasingly efficient image analysis. Infrared cameras, photo reconnaissance systems, and image guided weapons generate information at rates too explosive to handle with current hardware and algorithms. The problem is compounded by the coming of multi-sensor systems.

This effort begins by examining biological image processing in order to develop and test algorithms that may solve the segmentation problem.

1.1.3 Computer and Biological Vision Computer vision is the process that combines information from one or several sensors, previously acquired models of scene content, and world knowledge to construct a model of the sensor's environment. To be useful, the model must be complete enough to perform a given task.

Often, the task might be as simple as deciding if one view of a scene is different from a previous image, a problem common to overhead reconnaissance systems. Similar analysis can be used to find motion in an image. Short term motion could detect a vehicle moving across the sensor's field of view. Longer term motion, (such as from satellite imagery) might discover increased activity at a military site or detect patterns of growth for environmental monitoring and urban planning.

The goal of computer vision is to take a series of images, and by some means, construct a model of the environment around the sensor. This process begins by decomposing the image into identifiable regions. For examples, an image could be decomposed into target or background, then targets divided into tank, truck, jeep, and so on. Image decomposition is still largely an unsolved problem. How can the separate objects in an image be recognized and classified? Consider the problem of robotic navigation. Given a

preplanned description of the environment where the robot is expected to operate, can the vision system locate itself within the memory representation of that description.

This dissertation will focus on the early aspect of computer vision, the neural network front-end. Pixel information (gray scale intensity) is used to find points of interest, segment out objects and classifying the objects. Associating an object with a coordinate is all that is necessary to place a sensor in its environment. The initial decomposition provides the basis for placing the sensor. This process, sometimes called pre-attentive vision, may be the most difficult part of the problem.

Image analysis begins by picking out the areas of interest for special attention. These areas can be given special attention for picking out specific objects. Segmentation is the process of separating objects of interest from their background. The procedure involves identifying each pixel of an image as belonging to a specific category. For example background/object or target/non-target. The feature extraction process makes meaningful measurements of the object. The classification step uses the features extracted in the first step to identify the objects. A model of the environment is constructed by combining the two dimensional coordinates of the identified objects into an overall three dimensional model.

Solutions to these problems may lie in algorithms and architectures which mimic the biological visual system. Artificial neural networks have already proven effective for tactical target recognition provided the preprocessing step, image segmentation and feature extraction, are performed by conventional methods (36, 34, 48).

The purpose of this effort will be to examine the use of neural networks for the fundamental problems of computer vision, image segmentation and feature extraction, the so called pre-attentive vision problem (21).

1.1.4 Artificial Neural Networks for Image Analysis. Artificial neural networks (ANNs) have been used in pattern recognition systems for some time. Feedforward neural networks have been shown to be effective classifiers provided the features applied to the

network are sufficiently separable in the decision space. Still, selection of the features from digital imagery is usually performed using a heuristic approach.

While little of the biological visual system is understood, its examination may offer algorithmic approaches to machine vision. Biological vision may provide insight on how to solve the most difficult aspects of image understanding. Lateral inhibition in the receptor cells in the retina provides the first steps of automatic gain control. The $\ln(Z)$ transformation of the retinal image to the visual cortex provides scale and rotation invariant processing and provides a means to detect motion with respect to background. Other visual processes provide the solution to generalized object recognition.

To pick an object out of an image, some questions might be asked:

1. Does local spatial information alone characterize an object? How should the information be presented to a neural network? Do local two-dimensional Gabor filters represent an efficient compromise of spatial and spectral data. How are the filter components selected for object identification?
2. Biological vision systems are able, by means of several independent systems, to partition the world into generalized blob type objects. Can machine algorithms be devised to mimic the ability of the visual process to find boundaries between objects, and associate those boundaries with specific texture areas of a scene? Can the process allow for ambiguous illumination and intermittent boundaries?
3. Conventional image segmentation involves classifying each pixel into subpopulations or categories. In general, this is a difficult task as the intensity of the image may not be a good representation of the underlying physical variation of the scene. What transformation of the intensity information could facilitate the pixel classification process?

1.2 *Problem Statement and Scope*

Problem Statement. To investigate and evaluate image segmentation algorithms with regard to design, training and performance of target detection and tracking systems. The result of that research will be applied to the design of an end-to-end automatic target detection and tracking system.

Scope. Source data for the system will include tactical targets, infrared (IR) imagery and sequential target tracking imagery.

1. **Feature Vector Preprocessing Techniques:** Evidence exists to suggest that data processing in biological neural networks is not based strictly on the inputs from the sensors. Sometimes, the processing is performed on some linear (or nonlinear) combination of the data. For example, a Fourier transformation is a simple linear combination of weighted sums of a time delayed, or spatially shifted signal. While there is little reason to believe either signal is better for learning or classification, biological systems use both mechanical transforms, for example the retina to cortex transformation (or $\text{Ln}(Z)$) and computational transforms like Gabor filtering. The $\text{Ln}(Z)$ transform is computed by the physical placement of the receptors in the retina. Gabor filters are computed by taking a weighted sum of adjacent receptors. The first step will be to examine the relationship of simple transforms on neural network classifiers.
2. **Foveal Vision Preprocessing:** A central feature of biological vision processing is the spatial transformation preprocessing provided by the placement of the rods and cones in the retina. The increased density of the receptor cells in the fovea combined with the mapping of the ganglion cell outputs onto the visual cortex, provide an approximate $\text{Ln}(Z)$ transformation of the retina data. The objective is to investigate the use of the foveal transforms as means to reduce the amount of data required for processing.

3. **Machine Implementations of Algorithms:** Evidence that processing of complex imagery can be performed in real time is provided by our own existence and other biological vision systems. Applying biological paradigms to electronic systems will impose constraints some practical and some impossible. The objective is to determine which types of algorithms are suited to hardware implementation.

The research analysis consists of four phases:

1. Develop an image processing and neural network simulation environment to test specific approaches.
2. Investigation of transformations on scene objects for segmentation.
3. Investigation of feature vector preprocessing algorithms.
4. Application of these techniques to a tactical target image processing task.

1.2.1 The NeuralGraphics Neural Network Simulation Environment The Neural Graphics system was developed to test the application of general data sets to neural network solutions. The difference between this system and conventional research software is the additional effort put into the user interface to make the system a general purpose research tool for a variety of user requirements.

1.2.2 Preprocessing Feature Vectors for Classification The investigation will begin with analysis of feedforward neural network classifiers using hard boundary decision region problems. Using two dimensional random data divided into arbitrary disjoint decision regions, biologically inspired algorithms will be tested to determine if the classification process is facilitated. In addition, alternative transforms will be examined to determine if the complexity of a problem can be reduced.

1.2.3 Image and Feature Vector preprocessing The next area of study is the use of preprocessing techniques to reduce the complexity of the classification "back-end". Fourier

techniques have been used for some time. Gabor filters have been used by Daugman (8) and others for compression and image coding, but not for classification. Evidence of Gabor processing in biological vision systems is demonstrated by Jones and Palmer(20).

1.2.4 Local Area Transforms for Target Segmentation The second phase of the investigation will examine local area transformations of sequential imagery. Various transformations will be applied to a small region of a larger image. The transformed values will be used as input to a neural network classifier. The object will be to determine how geometric primitives can be identified from a local area transformation.

1.3 Dissertation Organization

This dissertation will be organized as follows:

- Chapter I. Introduction and overview. Problem statement. Description of the types of problems under consideration.
- Chapter II. Use of vision and coordinate transformations for feature extraction and classification on simple problems.
- Chapter III. Relations between common neural network paradigms. Development and description of generalized neural networks.
- Chapter IV. Feature selection and node saliency. Development of the Karhunen-Loève , and Gram-Schmidt networks, with a comparison to identity nets.
- Chapter V. Application of the generalized net to problems in target segmentation and tracking.
- Chapter VI. Conclusions and Recommendations.

II. Cortex Transforms for Pattern Recognition and Image Analysis

2.1 Introduction

This chapter examines the use of coordinate transforms for image preprocessing to reduce the complexity of the decision space. Two transformations are examined: the cortex transform and the hyperbolic transform. The first image data transform examined is sometimes called a cortex transform or more correctly, the $\text{Ln}(Z)$ transform. Cortex transforms are suggested by the image transform from the mammalian retina to the striate cortex. The hyperbolic transform was suggested by a priori knowledge of the problems. Cortex transforms can be constructed by placement of the photoreceptors in a focal plane array, similar to the retinal placement of photoreceptors. The cortex transform is compared to the hyperbolic transform, which is a two dimensional degeneration of conic basic function presented in Chapter III. Both of these transforms reduce the complexity of the decision space for a wide class of problems.

Ruck (39) showed that neural network models that minimize the mean squared error norm, approximate a Bayes optimal discriminator. For purposes of this experiment, neural networks and Bayes' analysis are considered equivalent. Improving the training times for a neural network problem is assumed to indicate a less complex decision space. If the neural network can partition the decision space faster, it suggests that the preprocessing is beneficial to the classification problem.

Three problems are examined to test the advantages of cortex transforms as well as other data transforms. The first two problems are two dimensional decision region problems. The last problem is a character recognition problem for the handwritten example of the first few letters of the alphabet.

2.2 The Cortex Transform

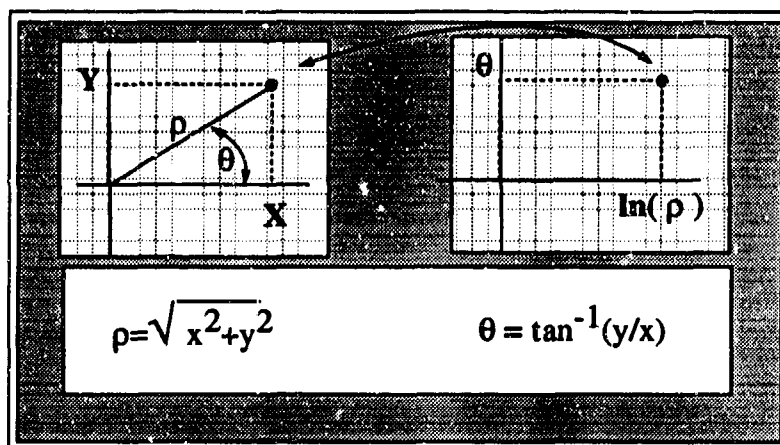


Figure 3. The cortex transform is a two-dimensional conformal mapping which takes the intensity of an image at a point (x,y) and maps that intensity to a point in $(\ln(\rho), \theta)$ space.

To process spatial and temporal information in the cortex, biological vision systems have developed a front-end preprocessing scheme to reduce the computational complexity of the image analysis scheme. After the image falls on the retina, successive transformations occur between the retina, the lateral geniculate nucleus (LGN), and the striate cortex.

The stimuli provided by motion, depth, and color are processed in a parallel and partially independent manner. Of particular interest here are the algorithms used to process the perception of form. A coordinate transformation between the retina and striate cortex provides a cortical representation of information that may allow the extraction of reasonable features. The transformation can be expressed as a coordinate conformal mapping from the x,y plane to a $\log \rho, \theta$ representation. Figure 3 shows the coordinate transform for the x,y plane to the ρ, θ plane. $\rho = \ln(Z + \alpha)$. Commonly called the $\ln(Z)$ ¹ transformation, the cortex transformation is similar, except that the discontinuity at the origin is removed. In software, all that is required, is to ensure the radius is never less than one. This is best accomplished by using a scale such that the width of one pixel is equal to one. The cortex

¹The transformation is similar to the conformal mapping $\ln(Z)$ where, $(Z = x + iy)$, where i is the square root of -1. The mapping is defined as $\rho = |z| = \sqrt{x^2 + y^2}$ and $\theta = \tan^{-1} z = \tan^{-1}(\frac{y}{x})$ (41)



Figure 4. Cortex Coordinate Transformation of an Image.

transform then is expressed as $\text{Ln}(Z+\alpha)$ where α is a small enough number to ensure the argument of the function is never less than one. Consider the images in Figure 4. The image on the left is a standard unscaled and unrotated image. The image on the right uses a coordinate transformation similar to that found in the visual cortex.

Since the visual system uses a coordinate transform to improve the ability to recognize objects, the question might be asked if a coordinate transformation might improve the separability of a general class of hard boundary decision region problems. Two problems were selected, the first is an exclusive-or problem with four disjoint decision regions. The second problem is a mesh problem with three concentric circles forming more complex decision regions as shown in Figure 5.

The NeuralGraphics (44) neural network simulator is used to simulate the problem. When the network is initialized with random weights the total error is measured with respect to every input exemplar. Thereafter at every display cycle, typically 1000 iterations, the total error is measured again and results are plotted. Figure 6 is the first of a number of NeuralGraphics screen displays which presents status information as the network trains. The output map, in the lower right, is a display of the entire input space and how the neural network would classify each area. Each color represents a particular class. The

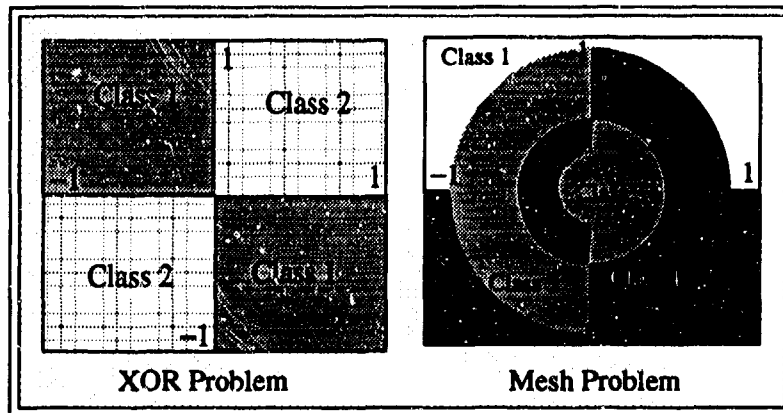


Figure 5. The Exclusive-Or and the Mesh Problem.

map covers the region starting with $(-1,-1)$ in the lower left hand corner up to $(1,1)$ in the upper right.

Node maps, the four square regions on the left (one for each hidden layer node), are similar to the output maps except that the display maps the output of a node in terms of its gray scale set between zero and one. The network topology is displayed showing the number of layers, the number of nodes per layer and the connecting weights. The value of the weights are displayed in gray scale according to the dynamic range scale displayed to the left of the net.

In addition to the error history, general statistics are displayed in the lower right hand corner. A classification of right means that the output of the node corresponding to the correct classification has a value of 0.8 or above and all others are below 0.2. For an exemplar to be classified good, the only criterion is that the node with the highest value corresponds to the correct output class.

The neural network solves a problem by providing a mapping from a set of inputs, or feature vector, to an output vector. The output vector is the classification of the input vector. A coding scheme of some kind is necessary to translate the floating point values of the output nodes into something meaningful to the user. The NeuralGraphics system uses one node for each output class. A value near one indicates that the network has determined the input vector is a member of the class associated with that node. A value near zero

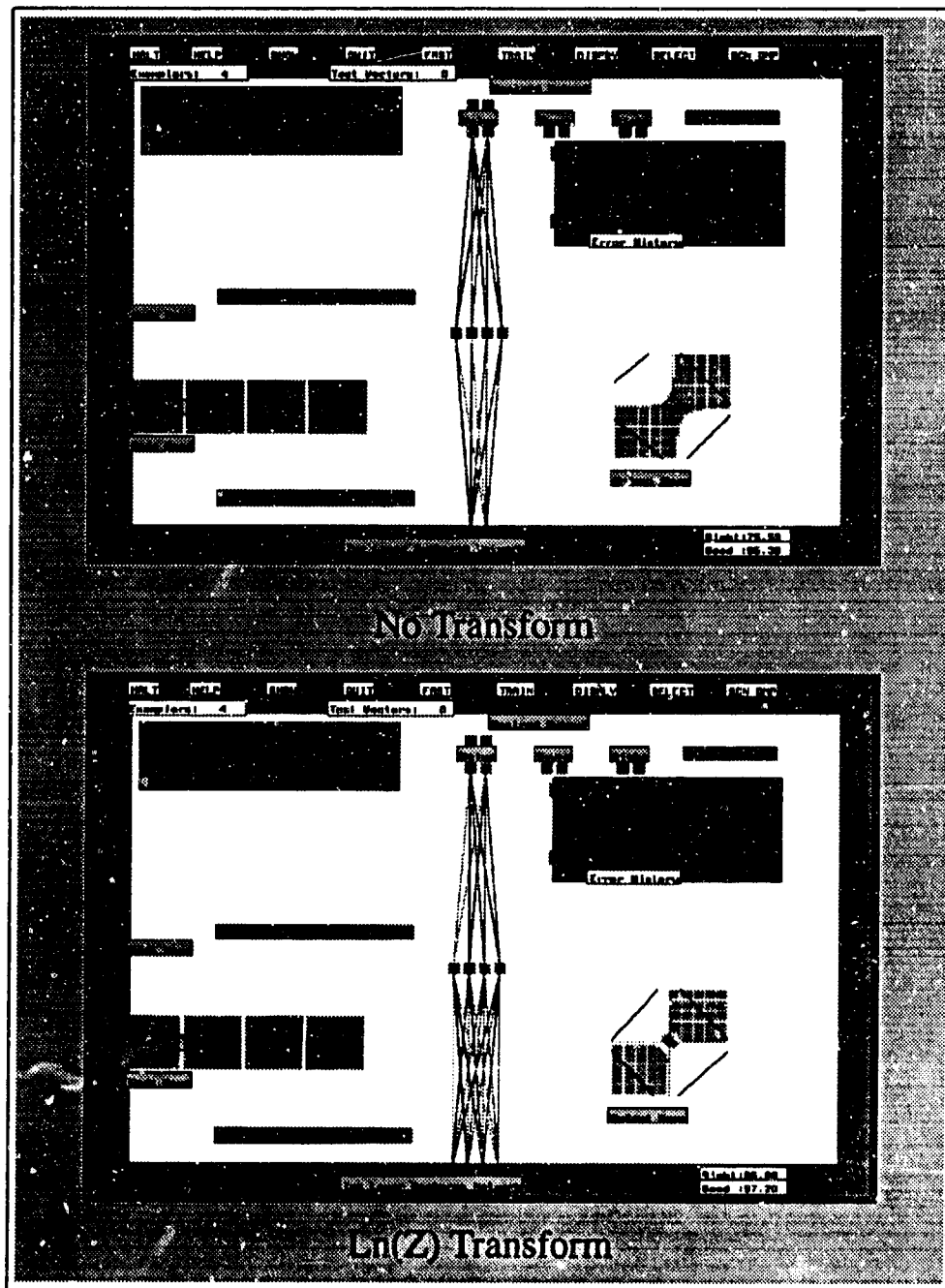


Figure 6. XOR Problem Screen Displays.

indicates a low probability that the input vector is a member of the node's associated class.

2.3 *The Exclusive-Or Problem.*

Evaluation of a neural network classification system usually begins with the analysis of a well understood problem. Here, an exclusive-or decision space with hard boundaries (no undefined space between classes) is used for the initial tests. The hard boundary exclusive-or problem is useful because it is a non-trivial, yet manageable problem. Also, it contains many problems associated with real data classification. For example, the problem contains multiple disjoint decision regions as well as an ambiguous region along the boundary between classes. The region is ambiguous because at the boundary, an exemplar could be either class. Ambiguous regions (or data points) prevent 100.00 percent accuracy from being achieved. The problem was implemented on the neural network simulator by creating four input data points, (1,1), (-1,1), (-1,-1), and (1,-1), two for each class. Each time a data point is presented to the network, a random value was added to each feature. For this problem the random value is a random number with a uniform distribution on the interval (-1,1). Given enough samples, the random value allows every point in the decision space to be classified into one of the two classes.

In Figure 6, the upper screen display was created with no transform of the input data. The window in the upper right corner of the screen displays the reduction of total error over time as the network trains. Total error is the total of the mean squared error values for each output node for the entire data set. The total error has fallen from 7.1 to 3.149 while most of the inputs are being correctly classified, the boundaries are soft as indicated by the value for **right** 75.0 percent correct of 1000 tries.

On the lower screen, a cortex transform is applied to the data. The total error falls off a little faster as shown in the error history. Also the total accuracy is greater, and the boundaries are sharper as indicated by the greater accuracy (86.8 percent **right**).

To investigate the general utility of transforming the data, another transformation is also considered. In addition to the x,y components, another term, the product of x and y

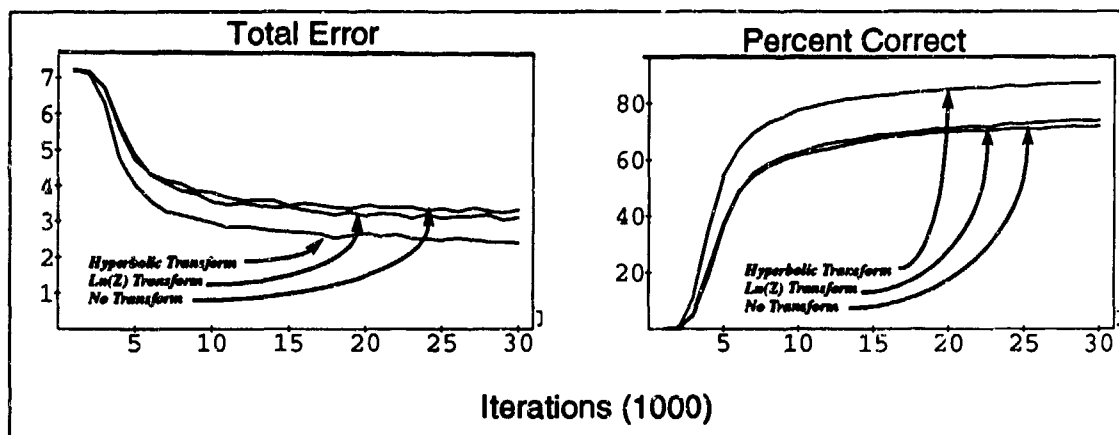


Figure 7. Training Results for the XOR Problem.

is added as a third element of the feature vector, forming a type of hyperbolic transform. As discussed in Chapter III this transformation turns out to be a special case of the conic basis function network. The improvement in training times and accuracy is significant.

Each screen display represents only one example of the search down the error surface. Such evidence is anecdotal at best, so a statistical approach was taken to ensure accurate results.

The exclusive-or problem was run 100 times for each of the data transforms. At every display cycle, (1000 training cycles) the total error was measured. The error measurement was taken as the sum of the total error for 1000 random samples of the decision region. The results are shown in Figure 7.

Notice that both the slope of the error, and the final error is better for each of the transforms. The reduction was small for the cortex transform and significant for the hyperbolic transform. The significant reduction for the hyperbolic transform can be explained in the truth table of the data shown in Table 1. The hyperbolic approach reduced the complexity of the problem enough that it could be solved using only a single layer network. The data transformation allows for the separation of the two classes in a decision space with a single hyperplane. The decision space for the three dimensional problem is actually less complicated than the original two dimensional space problem as shown

in Table 1. The two dimensional space problem contains four disjoint learning regions, which can only be partitioned with (at least) two hyperplanes. Table 1 shows the product of x and y is positive for class one and negative for class two. Thus one hyperplane can separate the data. The equation of that hyper plane would be $xy = 0$.

Sign of Input			
x	y	x·y	Class
+	+	+	1
+	-	-	2
-	+	-	2
-	-	+	1

Table 1. Exclusive-Or Problems. The table indicates the disjointness of the learning regions. Class one is defined as the case where x and y have the same sign. Class two is where x and y have different signs. These constraints can be met in two ways for both cases. But by taking the product of x and y , only one possible region exists for each class. Class one is the case where xy is positive and class two is the case where xy is negative.

2.4 The Mesh Problem

The mesh problem is another hard boundary test problem used to test the complexity of the decision region with the augmented feature vector. The decision region consists of three concentric circles each with the right and left half of the circle representing one of four decision regions. The training data was generated by randomly sampling the decision space 1000 times, then classifying by hand each point (See Figure 5). The network was trained on those points. The network was tested by taking uniformly spaced samples of the decision space and plotting the color coded response of the network in the lower left window of the screen display. (See Figure 8).

Using the cortex transform, the experiment is conducted using first the untransformed data. The results are shown in Figure 9. Next the net is trained using the transformed data.

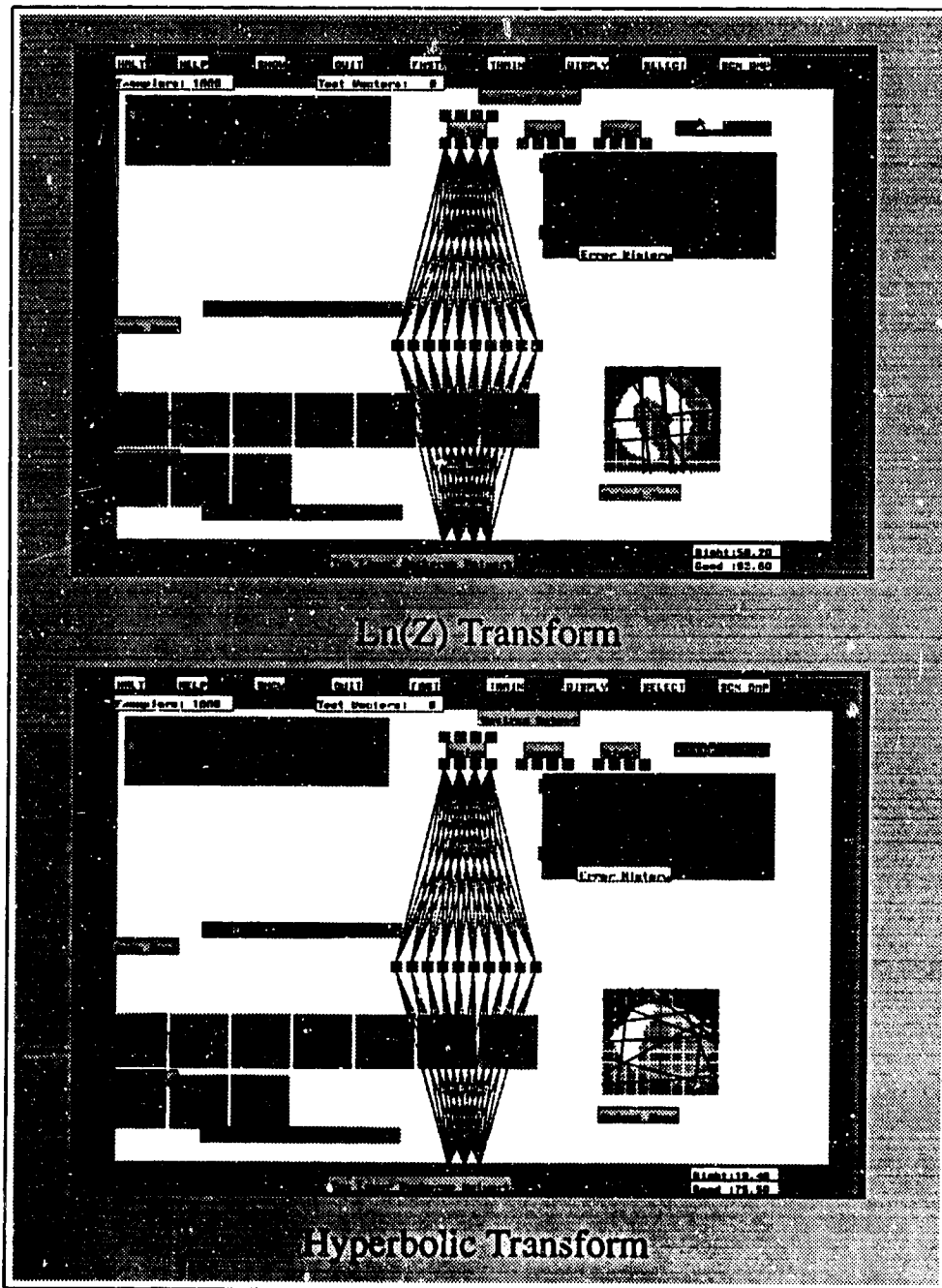


Figure 8. Mesh Problem Screen Displays. The graph at the upper right shows the reduction of error over time during training. The square windows below the hidden nodes are plots of the response of the hidden nodes for the entire input space. Light values indicate a low response, dark values indicate a high response. The map in the lower right is the networks best estimate of the output class for that region of the input space.

One run is made for the cortex transformed data, and one for the hyperbolic transformed data. As shown in Figure 9, the improvement is significant using the cortex transform.

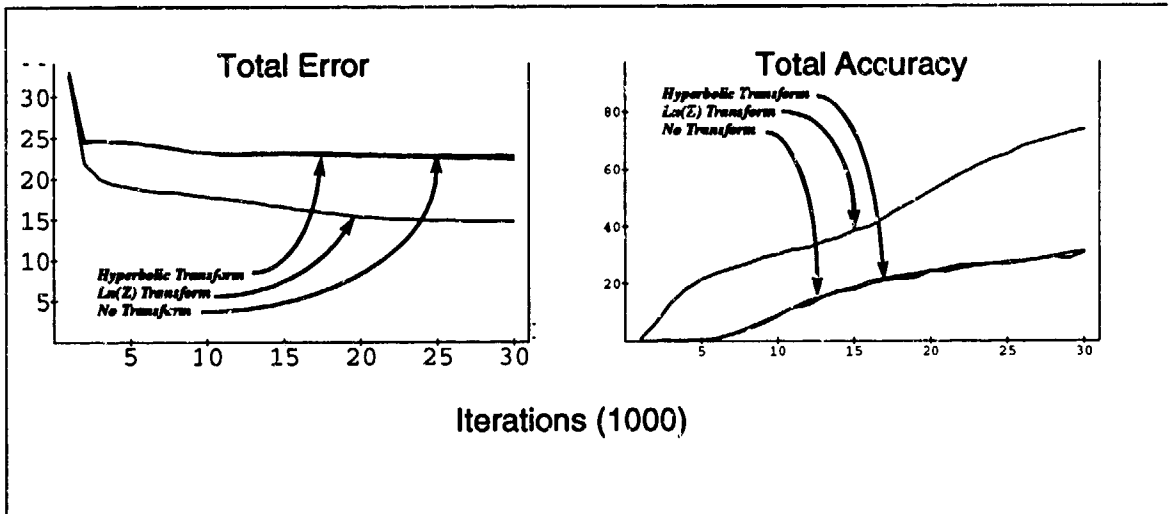


Figure 9. Mesh problem training results. For the mesh problem the cortex transform performed better than either the actual values or the hyperbolic transform.

The hyperbolic transform improved training for the exclusive-or problem and the cortex transform improved training for the mesh problem. The results appear problem dependent. The next section will examine the use of the cortex transform for a problem using real data.

2.5 Cortex Transforms for Handwritten Characters.

No improvement in neural network paradigms is of any significance unless shown to be useful in general applications. The application chosen here is that of identifying single handwritten characters from several subjects.

The characters were written using a mouse as the input device. The figure above shows the characters and their cortex transforms. Most work has shown that pixels do not make good feature vectors, and for that reason the low order Fourier components were selected instead (29, 22). The feature vector then is composed of the 25 low order

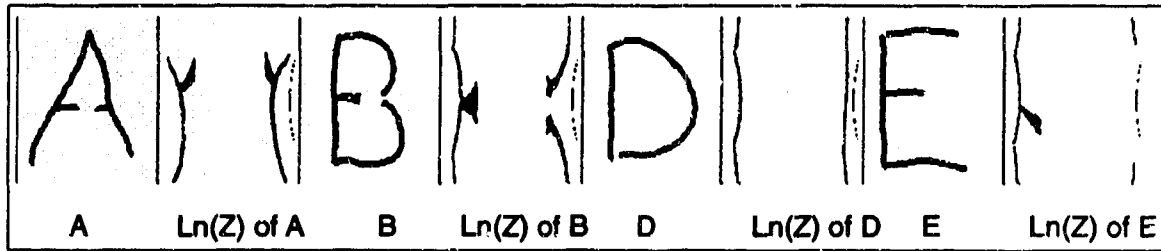


Figure 10. Handwritten letters with cortex transforms. Letters were scanned on a 64 by 64 pixel window. The image was transformed with a small circle at the origin removed. 25 low order Fourier coefficients were used for neural network classification.

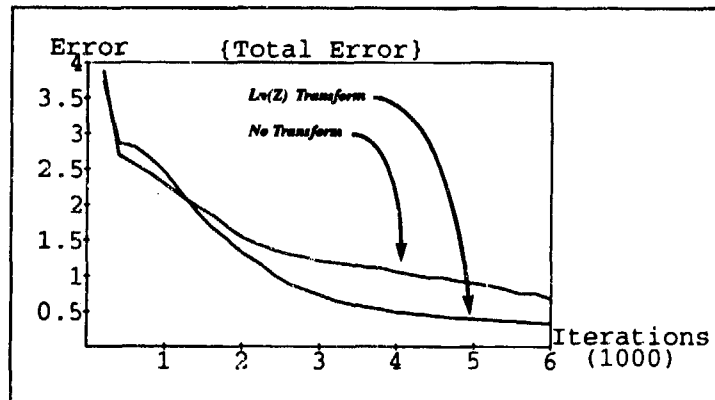


Figure 11. Character recognition plot of error reduction. The neural network learns faster using the Fourier transform of the Ln(Z)(cortex transform) image of the letters.

magnitude components of the two dimensional transform of the cortex transformed image. Ten samples of each of five characters were created for a training set of 50 exemplars.

The graphs in Figure 11 show the reduction of total error over time. The lower graph represents the training set with a cortex transform on the input scene. This graph shows faster training for the transformed data. Better training times indicate that the decision space in the transform domain is less complex than in the original image. The experiment with the hand written characters showed improved performance for this small problem. The character reconstruction highlighted some of the problems inherent to this approach. The problems came from the construction of the cortex transform. The discontinuity at the origin caused a number of problems.

2.6 Conclusion.

This chapter discusses the test of three data sets: the exclusive-or problem, the mesh problem and the handwritten character recognition problem. Each data set was approached using three preprocessing techniques: no input transformation, hyperbolic transformation, and cortex transformation. In each case the preprocessing improved the training times, indicating a reduction in the complexity of the decision space.

The cortex transform, while improving training for the test cases, included a number of special problems. In practice, the discontinuity at the origin causes thickening for lines near the center of the field of view. In Figure 10 the pixels within a radius of five of the center were set to zero otherwise the transformed image may or may not become mostly black based on whether the line passed within five pixels of the origin. If the line happened to miss the origin (like in the *A*, or *B*) half the image is black if not white.

Those pixels at the origin of the original image, account for 50 percent of the transformed image. Biological systems overcome this weakness by increasing the density of the photoreceptors in the center of the field of view. Implementing cortex transforms in computer systems would require a complex aspheric lens, or non-uniform sampling of the focal plane. Holographic lenses may provide a solution to this implementation problem.

The human brain contains as many as about 10,000,000,000 neurons. Ten percent of them are thought to be dedicated to the visual system (23). The cortex transform may require more interconnected computational units than would be practical in a machine vision system.

More immediate results were derived from the hard boundary preprocessing experiments. The first transform discussed, the hyperbolic transform, is a simple case of the conic basis function network presented in Chapter III. The steeper slope in the learning curve shown in Figure 7 indicates that adding the *right* additional terms can improve training. Determining transforms which improve the performance is generally a matter of heuristics. In practice, input transformations are a matter of adding a priori knowledge to the classification problem.

The following chapter will adapt the hyperbolic transform to higher dimensional data, which is a means of avoiding the search through all possible combinations of input transformation functions by providing a general set of input transformations and letting the neural network select which ones to use.

As shown by the graphs in Figure 7, different representations of the same information to a neural network do not represent the same level of decision region complexity. This section showed the complexity of a problem can be reduced by changing the representation of the data without changing information content.

The next chapter discusses a generalized input transform of the input feature vector to reduce the complexity of the learning space.

III. Generalized Neural Network Architectures

This chapter presents a formalism for generalized feedforward neural networks. A generalized architecture is necessary to allow a number of the techniques used in Chapter V to be implemented using a single iterable device. The previous chapter showed how higher order terms can facilitate training times. This chapter will show how higher order terms, and other preprocessing techniques can be generated in an artificial neural network using a standard perceptron-like element.

The architecture is capable of mapping many common neural network paradigms into a single architecture. Using an intrinsically iterable element, neural networks can be used to compute common preprocessing techniques including Karhunen-Loève reduction, Fourier and Gabor spectral decomposition and some wavelet techniques in addition to common discriminant functions. By generalizing neural networks into a common framework, the relationships between various neural network paradigms can be explained in a straightforward manner, as well as implemented in hardware.

The next section will introduce the generalized neural network node and describe the network in terms of vector equations using a Cybenko type network. The nature of discriminant functions used by backpropagation networks will be examined. The difference between the discriminant functions provided by common neural network paradigms will be compared in terms of the generalized neural network node.

3.1 A Generalized Neural Network Algorithm

Figure 12 illustrates the neural network implementation of the Equation 1. A generalized feedforward neural networks node can be represented by the matrix equation:

$$z_k = f_h(\vec{X}^T \cdot \vec{A}_k \cdot \vec{X} + \vec{W}^T \cdot \vec{X} + \theta_k) \quad (1)$$

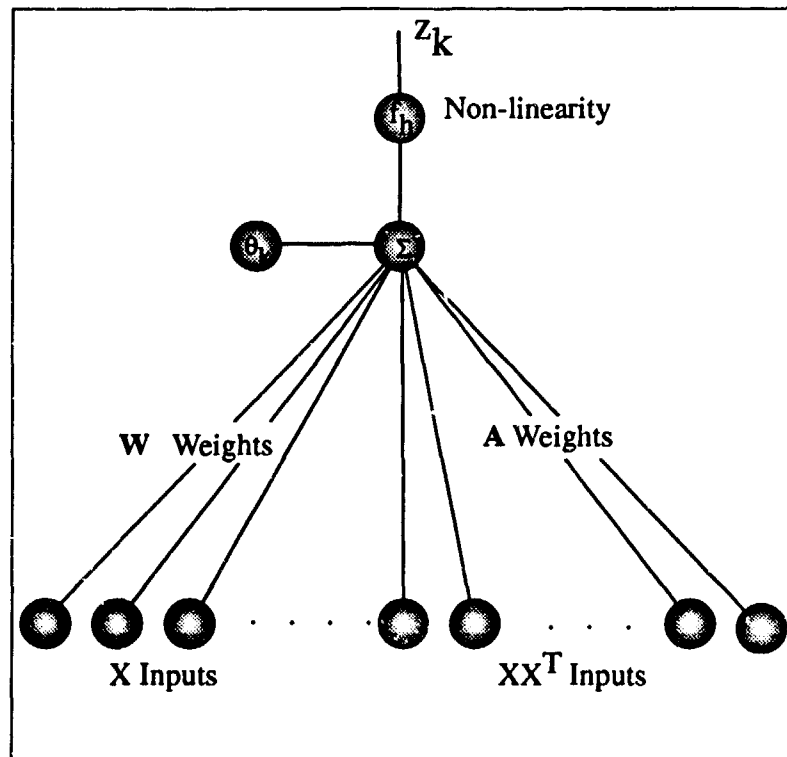


Figure 12. A Generalized Neural Network Input Node

\vec{W} represents the first order weights. \underline{A}_k represents a weight vector connected to second order inputs. The \underline{A}_k are weights which connect the second order input terms to the first hidden, or output layer nodes. θ_k is a single weight. z_k is the output of a single node, where k is the index on the output nodes.

This simple node structure allows implementation of many common neural network paradigms, as well as many common preprocessing techniques which include Karhunen-Loève reduction, Fourier and Gabor spectral decomposition in a connectionist network architecture¹.

Cybenko showed that one hidden layer of nodes (two layers of weights) is sufficient for any multivariate function approximation problem (7). Suter and Oxley proved that the output of the hidden layers can be, not only a sigmoid non-linearity, but also a negative exponential(42). While Cybenko (7) and Kolmogorov (27) have shown that convergence is possible, the resource requirements are never addressed. In other words, a large number of hidden layer nodes may be needed to solve any generalized functional mapping. This observation may have lead to the anonymous entry on the ARPAnet:

While Cybenko showed that a neural network can solve any input-output mapping with a single hidden layer of non-linear nodes, the result may be like proving that a monkey could write Hamlet. Certainly it could be done, you just need a lot of monkeys, a lot of typewriters, and a lot of time.(1)

The problem of resource requirements remains. Using a neural network efficiently requires making the problem as easy as possible for the net. The easier the problems, the fewer resources required. Problems can be made easier by careful selection of the feature space.

The previous chapter demonstrates how appropriate preprocessing of the data causes learning to be speeded up. Faster training, by only changing the input features, indicates a less complicated problem. The first layer of an artificial neural network provides a

¹Finding the correct values for the weight matrix is regarded as a separate problem.

set of basis functions used to construct an output. These nodes can be described as basis functions because the output is made up of weighted combinations of the output of these nodes. Selection of appropriate basis functions at the hidden layers can reduce the complexity of the problem. Since neural network nodes calculate the discriminant function, improving and generalizing the node structure may ease the requirement for large numbers of nodes.

Pao (32) suggests improvements on the node structures to reduce the complexity of the decision space. He uses additional terms in the input, as part of the feature vector. Pao refers to these additional terms as functional links. Nilsson(30) suggested the inclusion of second order terms in a general classification architecture. These include all the product terms up to second order. Pao suggests additional that function-like terms be included as well, and uses the example of $\sin(x_i)$ and $\cos(x_i)$. In practice the advantage of using functional links is dependent on the nature of the decision space. If the functions are related to the nature of the input data, the improvement is dramatic. Unfortunately, when using neural network techniques, a priori knowledge of the decision space is usually not assumed.

The next section will describe the discriminate function provided by a simple Cybenko type network as show in Figure 13.

3.2 Perceptrons - Hyper-Plane Discrimination

If f_h is a sigmoid function,

$$f_h(\alpha) = \frac{1}{1 + e^{-\alpha}} \quad (2)$$

the expression in Equation 1 describes the common perceptron allowing all first and second order terms. Note that if the matrix A is the zero matrix, then we have,

$$y_j = f_h(\vec{W}_j^T \cdot \vec{X} + \theta_j)$$

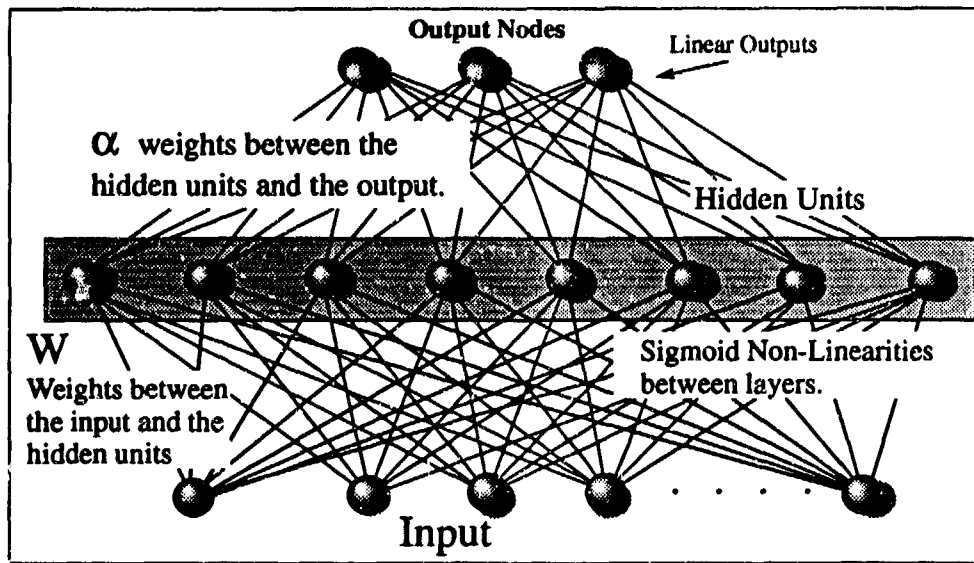


Figure 13. General form for a Cybenko type network.

a simple first order perceptron, and for each node,

$$y_j = f_h\left(\sum_i^n w_{ij} x_i + \theta_j\right)$$

where n is the number of input nodes. If f_h is the sigmoid function from Equation 2, the output of

$$y_j = \frac{1}{1 + e^{-(\sum_{i=1}^n w_{ij} x_i + \theta_j)}}$$

then the output layer can be linear nodes (no sigmoid squashing). That is:

$$z_k = \sum_{j=1}^J \alpha_{jk} \cdot y_j + \phi_k \text{ (sum over hidden units).}$$

The outputs of the network for a two layer net can be described in terms of the inputs:

$$z_k = \sum_{j=1}^J \alpha_{jk} f_h\left(\sum_{i=1}^n w_{ij} x_i + \theta_j\right) + \phi_k,$$

where z_k is the output, j is the index over the hidden layers, and i is the index over the inputs. Cybenko showed that this architecture is rich enough to approximate any functional approximation (7).

Equation 1 defines the elements which make up the generalized neural network. The following development will show how these functional elements can be used to generate the common neural network paradigms. For these definitions the vector A_k will be redefined to be a matrix such that the index of each element a_{lm} corresponds to the weight connected to the second order input of \vec{X} , $x_l x_m$.

Note that if A is a diagonal matrix,

$$A = \begin{pmatrix} a_{11} & . & \dots & 0 \\ . & a_{22} & \dots & . \\ \vdots & \vdots & \ddots & . \\ 0 & . & . & a_{nn} \end{pmatrix},$$

then an additional set of weights, associated with each of the second order individual features is defined. That is, there are the usual $w_{ij}x_i$ terms and also a set of $a_{ii}x_i^2$ terms. Thus,

$$z_k = \sum_{j=1}^J \alpha_{jk} \cdot f_h \left(\sum_{i=1}^n a_{ii}x_i^2 + \sum_{i=1}^n w_{ij}x_i + \theta_j \right) + \phi_k. \quad (3)$$

If A is not diagonal then cross-terms exist and the output of the network can be described by a similar equation. First a new index for A is introduced to distinguish between which cross-term of X are connected to which hidden layer nodes. Use of the higher order terms requires n terms for the first order nodes, and n^2 for the second order terms. But since multiplication is commutative, many of the terms are redundant. The following equation uses two new indices: (l,m) when the cross products exist. The index extends only to m less than or equal to l because terms beyond that are duplications of previous terms.

$$z_k = \sum_{j=1}^J \alpha_j \cdot f_h \left(\sum_{l=1}^n \sum_{\substack{m=1 \\ m < l}}^l a_{lm} x_l x_m + \sum_i^n w_i x_i + \theta_j \right) + \phi_k. \quad (4)$$

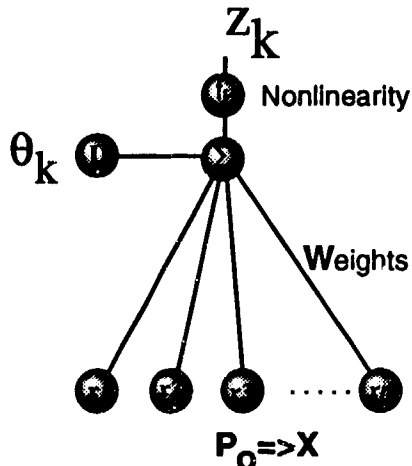
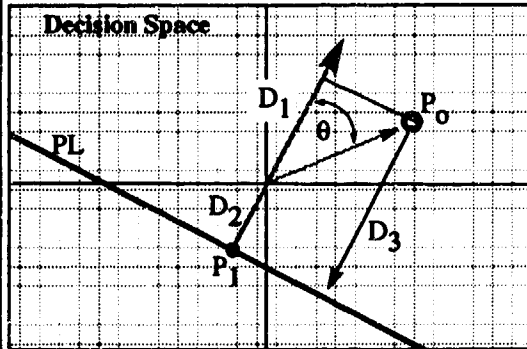
Lippmann(28) demonstrated how simple perceptrons perform hyper-plane separations of the input data. By setting the argument of the sigmoid to zero the resulting expression is the equation of a hyper-plane. This formulation of a neural network node will allow separation of classes by appropriate combinations of hyper-surfaces in the form of n dimensional conic sections. Note that each exemplar (a feature vector and an associated output class) is defined as a point in the decision space. The argument of the sigmoid function is the equation of a hyper-plane. Class distinction is made by determining if the argument of the sigmoid is positive or negative.

Positive arguments are the collection of points on one side of the hyper-plane, negative arguments are on the other. It's interesting to note from the development in Figure 14, the output of a perceptron supplies a distance metric, multiplied by the magnitude of the weights. By adding second order inputs, the expression described by setting the argument of the f_h to zero results in discriminant functions that are no longer linear in X , but quadratic in X . The decision surfaces are not hyper-planes, but hyper-conic sections. Adding second order terms reduces the number of nodes required to approximate a decision boundary or function. Generally, combinations of a number of hyper-planes are required to approximate curved decision surfaces, while the use of higher order terms (conic section) alleviates the need for large numbers of hidden layer nodes. For example, in the exclusive-or problem described in Section 2.3 use of the second order term allowed the network to solve the problem without hidden units.

Third order terms offer even more complex decision surfaces, but in practice, the benefit is outweighed by the cost of the exponential increase of input terms. Barron (3) uses higher order combinations of input terms. By precalculating the relative saliency (value) of the terms, he is able to retain only those terms which contribute to the problem's solution.

What is the output of a perceptron ?

Consider an n dimensional space.



Equation of the Hyper-Plane (PL) is $Ax_1+Bx_2+Cx_3 \dots +Ex_n = D$. The plane coefficients can be represented as a vector W , and the points in space as a vector P with features x_1 . Thus the plane (PL) can be represented as the set of points such that $W \cdot P = D$. Where P is any arbitrary point and D is a constant.
 $A\hat{i}+B\hat{j}+C\hat{k} \dots$ are the direction vectors for the line perpendicular to PL. Thus a normal vector is also W .

The distance D_3 is given by:

$$D_3 = D_1 + D_2$$

First find D_1 .

D_1 is the projection of P_0 on the normal (W)

$$W \cdot P_0 = \|W\| \cdot \|P_0\| \cdot \cos(\theta)$$

$$W \cdot P_0 / \|W\| = \|P_0\| \cdot \cos(\theta)$$

$$D_1 = \|P_0\| \cdot \cos(\theta)$$

$$D_1 = W \cdot P_0 / \|W\|$$

Next find D_2

D_2 is the projection of the normal (W) onto P_1 .

$$W \cdot P_1 = \|W\| \cdot \|P_1\| \cdot \cos(\theta)$$

but since the vector from the origin to P_1 and the normal (W) are parallel, $\theta = 0$. Also, for all points on the plane, $W \cdot P = D$, so $W \cdot P_1 = D$. and since $D_2 = \|P_1\|$:

$$D = \|W\| \cdot D_2$$

$$D_2 = D / \|W\|$$

$$D_3 = W \cdot P_0 / \|W\| + D / \|W\| = (W \cdot P_0 + D) / \|W\| \Rightarrow \text{Distance from the point to the hyper-plane.}$$

The output of a perceptron is

$$f_h (W \cdot P_0 + D)$$

The normal distance of the point in decision space to the hyper-plane is :

$$\text{distance} = (W \cdot P_0 + D) / \|W\|.$$

so,

$$\text{distance} \cdot \|W\| = W \cdot P_0 + D$$

In other words, the output of a node is the sigmoid of the normal euclidian distance of the exemplar to the discriminant function, times the magnitude of the weight vector.

Figure 14. Output of a Perceptron

Most neural network architectures can be classed as one of two types: those which use hyper-ellipses to partition the decision space and those which use hyper-planes. The first type is sometimes referred to as radial basis functions network. Both types are contained in the generalized neural network architecture by an appropriate setting of the weight matrix. The next section will show how both types are special cases of a similar architecture.

3.3 Radial Basis Functions(RBF): Hyper-Ellipse Discrimination

While backpropagation performs logical computations much like the child would identify objects based on the answers to twenty yes/no questions, radial basis functions provide a different view of the decision regions. Every node in the hidden layer identifies a specific region in the decision space. A radial basis node might be considered a computer implementation of a grandmother cell², that is a single node to identify a single concept.

Using the notation as in Equation 1, the output of a radial basis node is given by the following:

$$z_j = f_h\left(\sum_{i=1}^n \frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}\right) \quad (5)$$

where c_{ij} is the center of the j^{th} receptive field and σ_{ij}^2 is the variance. Notice that the output of the RBF is a metric, or value related to the distance between the input vector and the weight vector c . The variance term adds flexibility to the formulation which allows a cost function in the direction of a particular feature vector term. Expanding out the squared term,

$$z_j = f_h\left(\sum_{i=1}^n \frac{(x_i^2 - 2x_i c_{ij} + c_{ij}^2)}{\sigma_{ij}^2}\right)$$

²A grandmother cell is a concept that a single neuron is used to identify a single, complete concept. In other words, one neuron will fire upon recognition of an object, for example a grandmother.

$$z_j = f_h \left(\sum_{i=1}^n \frac{1}{\sigma_{ij}^2} x_i^2 + \sum_{i=1}^n \frac{-2c_{ij}}{\sigma_{ij}^2} x_i + \sum_{i=1}^n \frac{c_{ij}^2}{\sigma_{ij}^2} \right) \quad (6)$$

Because the variance is a multiplicative constant, it can be assumed to be one without a loss of generality. The effect on non-unity variance is to alter the size of the receptive field for a particular node. Thus if the input vector \mathbf{X} and weights \mathbf{C} have been normalized then

$$\sum_{i=1}^n x_i^2 = 1 \text{ and } \sum_{i=1}^n c_{ij}^2 = 1$$

$$z_j = f_h(1 - 2 \sum_{i=1}^n c_{ij} x_i + 1).$$

Two methods are possible for using radial basis functions for classification. The first method is to assign a class type to each node. The process, sometimes called calibrating the network, is possible by using one node for each training exemplar. If the training set is too large, self-organization methods allow assigning a known class to each node. The second method, more common to function approximation techniques, uses a least squares approximation of the output of the nodes for each exemplar. Since the output information of the node is related to distance, constants can be ignored for classification purposes. The only effect of including the first and third terms in Equation 6 is to increase the output of each node by a constant amount.

Normalizing the weights and input vectors allows a distance metric to be computed by taking the dot product of the input with the weights. The output of the node is a number between minus one and one.

Since two terms (Equation 6) are constants they can be ignored in computing a distance metric. Consequently, the only term of significance is $x_i \cdot c_{ij}$. A number of clustering neural network paradigms use this approach to compute distance from pattern templates, ART II, Kohonen Maps, and Counterpropagation to name a few. Normalization

has the disadvantage of forcing all exemplars to the unit hypersphere.³ This disadvantage can be mitigated by a number of approaches. Carpenter suggests one approach is to include a magnitude term as an additional feature element and renormalizing.

Weights attached to radial basis nodes become like the training data. Each weight array is trained in such a way as to represent a cluster of data, hence the analogy to the grandmother cell.

But the structure presented here can go beyond hyper-sphere, and hyper-plane partitioning of data. Note that the argument of Equation 5 could be implemented without input subtractions and divisions, but in the standard format of input times a weight, summed in a node. This is especially important since normalization need not be assumed. This led to the idea of using conic discriminate functions which include not only planes and spheres, but hyperbolas and parabolas.

The following section shows how RBF's can be implemented as a special case of the generalized neural network, yet trained using ordinary backpropagation.

3.4 Conic Basis Functions-Second Order Hyper-Surfaces

Perceptrons use combinations of hyperplanes for discrimination. Radial basis functions use hyper-ellipses for discrimination. Both can be implemented as single elements as shown in figure 12. A radial basis function computes squared distance and divides by the variance. The same calculation is possible using the second order inputs.

For any general radial basis receptive field, the A and \vec{W} weights associated with a particular input node would be (using the same indexing as in Equation 4):

$$\begin{aligned} \text{for } (l = m) \ a_{lm} &= \frac{1}{\sigma_{ij}^2} \\ \text{otherwise } a_{lm} &= 0 \end{aligned} \quad (7)$$

³In other words, vectors (1,1) and (2,2) appear to the network as $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$.

$$\begin{aligned}
 w_i &= -\frac{2c_{ij}}{\sigma_{ij}^2} \\
 \theta_i &= \frac{c_{ij}^2}{\sigma_{ij}^2}
 \end{aligned}
 \tag{8}$$

If the A matrix is zero, the system implements standard perceptrons. If the A matrix is diagonal with weights as in Equation 8 radial basis functions are produced. Additional discriminate forms are possible if cross-term weights are non-zero. The shape is dependent on the particular weights.

The generalized approach has a number of advantages. The output of the nodes (before the non-linearity) is exactly the squared distance, if the variance is one. Radial basis functions are part of the general architecture.

The equation for the node output could be called conic basis functions, because the output equation is the same as used to describe conic sections. The difference between conic basis functions and radial basis functions is simple. Radial basis functions discriminate between points in the decision space which are either inside or outside a radial or elliptical region. Conic basis functions include an additional class of discriminate functions: those which are hyper-parabolic. Figure 15 shows some of the types of discriminant functions possible using second-order combinations of two dimensional data.

3.5 *A Generalized Architecture*

Previous sections have shown that using a generalized neural network, a variety of common neural network paradigms can be computed. These functions were shown to be generalized in the sense that the output of a second order node is of the same general form as the equations for a conic section.

The generalization doesn't end there because, by turning off or changing the non-linearity, several common filter techniques are possible.

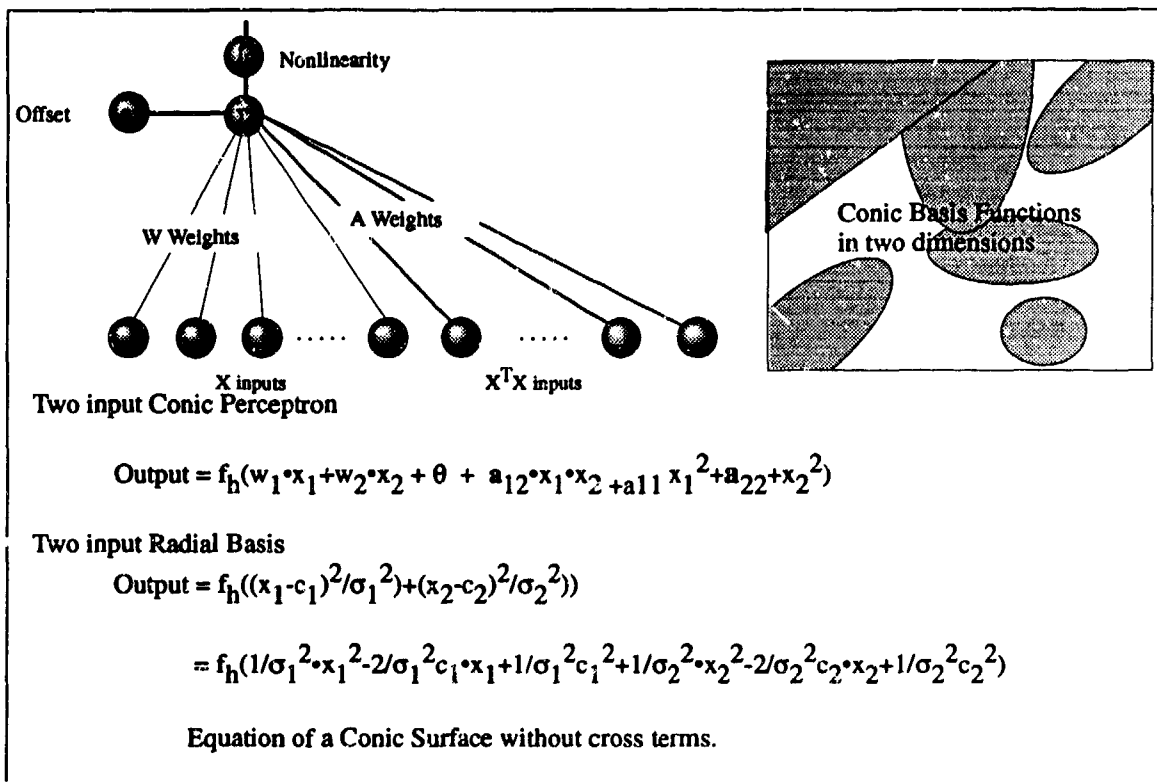


Figure 15. Conic Basis Function. The additional weights and inputs allow the architecture to provide discriminant functions with curved surfaces of a conic sections.

Artificial neural networks perform a mapping from a multivariate input feature vector, to an output mapping related to the structure of the data and the problem being solved, either classification, or function estimation. The generalized view of neural networks allows function estimation to include Fourier decomposition and Gabor filtering.

3.6 Generalized Neural Network Paradigm Implementations

Many common neural network paradigms and preprocessing techniques can be implemented by using different values of the weight A . The following examples show modifications which can alter the function of a generalized neural network.

- **Backpropagation.** For vanilla backpropagation the A matrix is zero. Most of the variations of backpropagation differ in the training algorithm.
- **Radial Basis Functions.** For hyper-elliptical receptive fields, the matrix A is diagonal. Variance values of other than one allow for elliptical receptive fields, unit values imply spherical receptive fields. In the conventional format the non-linearity is changed to a simple exponential, although a sigmoid can be used without loss of generalization.
- **Counterpropagation.** Counterpropagation is a variation on radial basis functions. The inverse mapping property of counterpropagation is not considered here.
- **Kohonen Mapping.** Kohonen maps are radial basis function networks where the output nodes are arranged as a two dimensional array.
- **Digital Fourier Analysis.** Neural networks can perform Fourier analysis by setting the weights according to the Fourier kernel, which is $e^{-\frac{j2\pi kt}{N}}$ where k is the number of the output node, N is the number of input nodes and t is the number of the particular input node. The fact that weights are complex can be overcome by allocating one weight for the imaginary part, and one for the real part.

- **Gabor Functions.** Gabor function analysis is performed by correlating the signal with a Gaussian times a complex exponential. For Gabor filtering, an input signal can be pre-multiplied with the appropriate Gaussian, then use the Fourier net on top.
- **Karhunen-Loève Feature Reduction.** Karhunen-Loève transforms, or eigen transformations, transform the input data into a feature space with a smaller dimensionality. The weights are calculated by taking the eigenvectors of the data covariance matrix and using one node for each eigenvector. Karhunen-Loève transforms are a means to self-organize the data based on the interdependence between the features. A Karhunen-Loève network is performed by taking the eigenvectors of the data covariance matrix and using each eigenvector as a weight vector. The derivation is shown in Section 4.4
- **Statistical Normalization.** Statistical normalization is important to neural network training because it relieves the backpropagation algorithm from learning wide differences in dynamic range between feature vectors. The effect is to stabilize the learning rate enough so that convergence can occur in finite time for a wide range of learning rates. Most important is that it allows all learning rates to be equal. Statistical normalization layers have only one connect between the input and the layer above, the weight is the inverse of the standard deviation. The threshold is the mean divided by the standard deviation.

3.7 Conclusion

While many neural network paradigms have gained popularity over the years, most paradigms can be simplified into one of two classes of discriminant functions, hyper-surface classifiers or hyper-ellipse classifiers, that are either perceptrons or radial basis functions. Both of these constructs can be implemented in a single class of neural networks called a conic basis function classifier. In addition, eliminating the non-linearity between layers allows for computation of Gabor filters or Fourier analysis. Generalized neural networks still suffer from the exponential growth of the input layer for large problems.

In order to reduce the size of the input layer for the image segmentation problem, the next chapter will evaluate several methods for finding a minimum sized input feature vector.

IV. Feature Selection in Generalized Neural Networks

This chapter will investigate the use of several novel techniques to reduce the size of the input feature vector and the size of the network architecture. This is important considering the generalized neural network architecture requires $\frac{n^2+n}{2}$ inputs. The first method is a straight forward procedure to compute the relative usefulness of a feature based on the magnitude of the weight vector. This simple method is shown to be statistically equivalent to more sophisticated methods. Next, a method for data preprocessing using a variation on the method of principal components is illustrated. Finally, a means of computing the principal components recursively is evaluated for use in segmentation problems.

These methods are used to (at least partially) overcome the *curse of dimensionality* referred to by Duda and Hart (9). The first method is derived from Ruck's *saliency metric* (35:32). The second method uses a novel neural network implementation of the method of principal components. The third method uses a self-organization technique called *identity mapping* to construct an alternate input feature vector.

4.1 Why Limit Input Features?

Any classification device will have limited resources to compute its output. In constructing a classification device, one of the first considerations is what measurements will be used for inputs. Increasing the size of the input vector increases resource requirements in a number of ways.

The size of the input feature vector controls the size of the interconnection matrix (weights). Also, larger input vectors require additional training data to ensure generalization. Foley showed(13) that as the size of the input feature vector grows, the size of the training data set should also grow to prevent simple memorization. Foley's rule suggests that the number of training samples per class should be greater than three times the number of features. Under this condition, the error rate on the training set should be close to the

error rate on an independent test set. Foley's rule requires certain statistical criteria must be met. Samples must be taken from Gaussian distributions, which in practice is usually ensured by the Central Limit Theorem. Cover, who assumes a Gaussian distribution, proved that a data set can be memorized if the total number of training samples is less than twice the number of features. He shows, that under these conditions, a two class data set can be partitioned by a single hyperplane, with a probability of greater than 50 percent, even if the data comes from the same distribution(5).

With image data, obtaining large data sets is not difficult, though preparation of the data for training can be. The difficulty lies in the large input feature vector generated by a scanning window architecture. For the segmentation problem, grouping pixels into classes, is based on local pixels that surround the pixel being classified. These pixels are (usually) taken from eight by eight, or sixteen by sixteen windows around the pixel in question. Looking at an area only eight pixels on either side of the source pixel, requires a 256 wide input vector. The only solution is to preprocess the data in some way.

The most obvious way is to eliminate redundant or useless features. This introduces the concept of saliency, or usefulness, of a feature. Unfortunately, the saliency of a feature cannot be determined individually. Classification is determined by taking combinations of features. Some features may be useful only in the presence of other features, while useless on their own. Consequently, saliency computations involve all network parameters simultaneously.

The following section will show a simple method to determine feature saliency using a trained neural network. Under certain conditions, network training allows saliency to be determined by examining the weights attached to a feature¹.

¹Saliency can be determined in a similar fashion for hidden nodes. A feature is defined as an input node.

4.2 Computing Feature Saliency

This section will present an empirical method to calculate saliency based on the weight vectors. Previous work by Ruck and others suggest several methods for finding feature saliency. The brute force approach to calculating saliency requires examination of the change in output of the network with respect to changes in the input. For example, if several points were selected over the range of each feature value the entire input space could be characterized. Such a characterization of the feature space would require a long time. Even if only ten points were examined for each feature of an eight by eight window, 64^{10} data point (exemplars) would have to be propagated through the network.

A alternate approach would be to examine the input space only in the regions where sample points exist. This method is called sensitivity analysis. Klimasauskas (24) suggests that a feature's saliency can be determined by holding the data sample constant and adjusting the feature in question over its range. Sensitivity analysis requires examination of the error over the entire range of the input features. The saliency is computed by looking at the change in error over the data set. Ruck et al (37, 10) suggest an examination of the feature space across the range of each input around each training point to determine the change in the total error for the classification problem. The difference here is that instead of looking at only the change in the output, the derivative of the error is summed for the entire data set, over a range of points for each input sample. The second method uses the derivative of the error directly while the first method attempts to approximate the derivative of the error.

Priddy(33) describes the calculation of the saliency metric below:

The saliency (Ω_i) can be calculated as follows: Take each training vector (\vec{x}) in the training set S. For the i^{th} feature, sample at various locations over the expected range of the feature while holding all other features in \vec{x} constant. Next compute the magnitude of the partial derivative of the output z_k for the sample values. Sum the magnitude of the partial derivative of P_{error} over the outputs (j), the sampled values of x_i (D_i), and the training set S.

The calculation of the saliency metric is described by the expression:

$$\Omega_i = \sum_j \sum_{\vec{x} \in S} \sum_{x_i \in D_i} \left| \frac{\partial P_{error}(j, \vec{x})}{\partial x_i} \right|$$

which can be rewritten with some restrictions as:

$$\Omega_i = \sum_j \sum_{\vec{x} \in S} \sum_{x_i \in D_i} \sum_{k \neq j} \left| \frac{\partial z_k}{\partial x_i} \right| \quad (9)$$

where

$j \equiv$ Output nodes index

$k \equiv$ Output nodes index (other than node j)

$D_i =$ Set of sample points for feature x_i

Ruck uses the expression:

$$\tilde{\Lambda}_i = \sum_{\vec{x} \in S} \sum_k \sum_{x_i \in D_i} \left| \frac{\partial z_k}{\partial x_i} \right| \quad (10)$$

Priddy showed that the difference between Equation 9 and Equation 10 is a simple multiplicative constant $(n - 1)$ (33). The following section describes a simpler way to compute saliency.

4.3 Weights as a Saliency Metric

Considering how weights in a neural network are updated, the weights can be used to calculate the saliency. When a weight is updated, the network moves the weight a small amount based on the error. Given that a particular feature is relevant to the problem solution, the weight would be moved in a constant direction until a solution with no error

is reached. If the error term is consistent, the direction of the movement of the weight vector, which forms a hyper-plane decision boundary, will also be consistent. A consistent error term is the result of all points in a local region of the decision space belong to the same output class. If the error term is not consistent, which can be the case on a single feature out of the input vector, the movement of the weight attached to that node will also be inconsistent. In a similar fashion, if the feature did not contribute to a solution, the weight updates would be random. In other words, useful features would cause the weights to grow, while weights attached to non-salient features simply fluctuate around zero. Consequently, the magnitude of the weight vector serves as a reasonable saliency metric, and can be calculated as shown in Equation 11, where i is the index of the feature and k is the index of the next layer node the weight is connected.

$$\tilde{\Lambda}_i = \sum_k w_{ik}^2 \quad (11)$$

Several conditions apply which are met using normal training procedures. Each feature should have about the same dynamic range, often obtained using statistical normalization, to ensure the relative value of feature saliency metrics are consistent.

To test this hypothesis, several data sets were processed with the NeuralGraphics software. Using both methods, the saliency of several input sets was generated and compared with the saliency metric as suggested by Ruck and LeCun, called here the second order method(37). Tables 2 through 5 show comparisons of the second order method and weight magnitude method for computing feature saliency.

In order to show that the magnitude of the weights could serve as an alternative to the expression in equation 10, the exclusive-or problem (Section 2.3) is run again. Because the result shown in Table 1 indicated that adding the first cross term improved training, the saliency for that term should be greater than the two first order inputs. An additional (redundant) term was added to the feature vector equal to two times the xy term. This has the effect of doubling the variance, which should increase it's saliency. If the weight

method is an accurate predictor of saliency, it can be expected that the xy terms would have a higher saliency than the x and y inputs.

The test was run and the results are shown in Table 2. The results were close to what was predicted. The results showed that the $2xy$ term was the most important, followed by the xy term with the x and y terms splitting third and fourth. The second order method worked as well, but the results were not as close to the predicted values. For the weight method, the saliency for xy was half the saliency for the $2xy$ term. Using second order saliency, the results were similar, yet obviously not as consistent. Even though features two and three were set up to be not as important as the xy terms, the second order saliency method ranked them higher on two out of twenty runs.

Since the weight saliency performed as expected for test data, the next step is to calculate saliency for real data. Ruck calculated saliency for a set of data based on laser range imagery and reported the results in (35). The data set is a 22 input, four output class problem. The data set was used to train a neural network twenty times. At the end of each run the saliency was calculated as before. The results for the weights saliency are shown in Table 3.

Notice in Table 3 that the net identifies 13,2,19, and 0 as the four most important features and 15,7,6, and 9 as the least important. Testing the prediction is possible by running the network, turning off the training, then eliminating nodes. Eliminating any of the nodes identified as the most important results in a severe loss of training accuracy while eliminating the least important nodes results in no change in total error or accuracy. The same test was tried for the second order saliency metrics shown in Table 5. The net selected 13,18,0,2 as the most important and 17,15,7 and 6 as the least important.

The statistics presented in Tables 2 through 5 were calculated using a two layer net with fifteen hidden nodes, for twenty runs. Ruck reported in his test that the most important features were 13,0,19 and 18 and the least important were 6,9,7,17(37). To compare the consistency of both methods, all the features selected as important by both nets are shown in Table 4.

Weight Method					
Feature	Rank				
	0	0	1	2	3
	0	0	0	12	8
	1	0	0	8	12
	2	0	20	0	0
3	20	0	0	0	

Saliency Histogram				
Feature	Rank	Value	Feature	Rank
0:	2.40	0.03	3:	0.00
1:	2.60	0.03	2:	1.00
2:	1.00	0.49	0:	2.40
3:	0.00	1.00	1:	2.60

Second Order Method					
Feature	Rank				
	0	0	1	2	3
	0	0	1	6	13
	1	0	1	12	7
	2	0	18	2	0
3	20	0	0	0	

Saliency Histogram				
Feature	Rank	Value	Feature	Rank
0:	2.60	0.00	3:	0.00
1:	2.30	0.00	2:	1.10
2:	1.10	0.07	1:	2.30
3:	0.00	1.00	0:	2.60

Table 2. Exclusive-Or Saliency.Node Saliency for the XOR Data. The square matrix illustrates 20 runs of the same data using different initial conditions. The rows indicate the number of times a particular feature ranked at that position. The columns indicate the feature. Row 3 of column 0, shows that feature 3 was the most important 20 times out of twenty. Column 2 indicates that feature 2 was in the number one place 20 times. Below the histogram, the saliency tables show the average rank of a feature first by number, then rank ordered. The value column shows the actual calculated saliency scaled by the highest value for the final run.

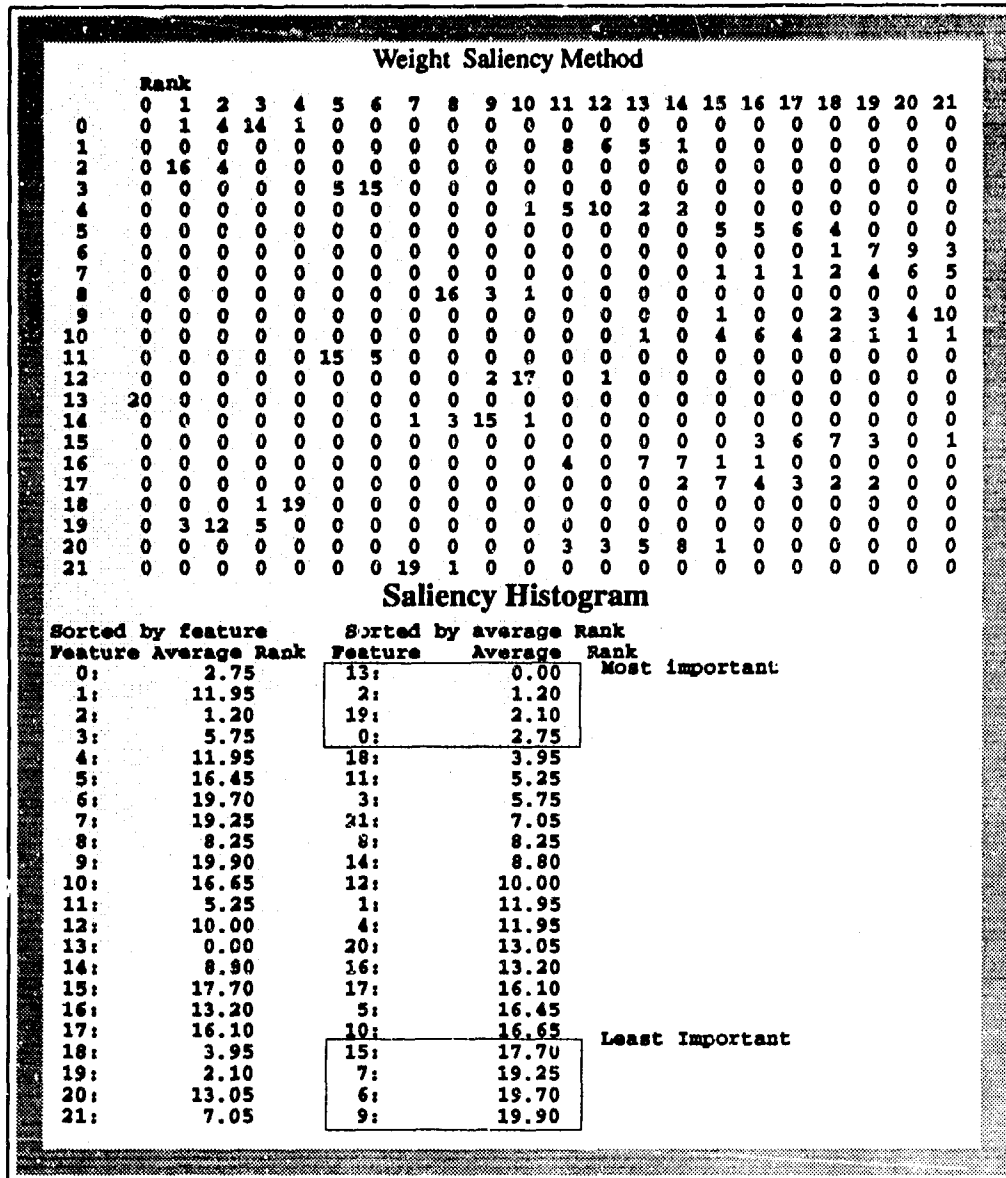


Table 3. Node Saliency for the Ruck Data using weight vector magnitude. The square matrix illustrates 20 runs of the same data using different initial conditions. The rows indicate the number of times a particular feature ranked at that position. The columns indicate the feature. Row 13 of column 0, shows that feature 13 was the most important 20 times out of twenty. Column 2 indicates that feature 2 was in the number one place 16 times and number 2 four times. Below the histogram, the saliency tables show the average rank of a feature first by number, then rank ordered.

Comparison of Saliency Calculation Methods			
Feature	Weight	2nd Order (Tarr)	2nd Order (Ruck)
13	0	0	0
2	1	3	5
19	2	5	2
0	3	1	1
18	4	2	3
21	7	4	4

Table 4. This table shows how the features selected as most important compared for each saliency method. Notice, that even though the rank ordering was not exactly the same, the discrepancies were not more than a few places different. The first column was calculated as shown in Equation 11. The second column shows the results as calculated by Equation 10. The third column is taken from the Ruck[ruck:fsel].

As Table 4 shows, the ranking for the top few nodes never differed by more than a few places. Using saliency to determine which features to select was demonstrated by Ruck. Calculation of saliency as demonstrated by LeCun and Ruck, requires second order networks² and considerable computational effort is involved. Calculation of saliency using the weight method requires fewer calculations and is possible on an ordinary backpropagation network.

The results presented in this section show that the weights can be used to compute the saliency of a feature. The specific results for the exclusive-or problems indicate that saliency doesn't take into account redundant features. Fortunately, a mechanism for elimination of redundant, or correlated features exist. Karhunen-Loève transformations provide a means of eliminating redundancy. The next section will discuss principal component analysis for preprocessing data in a neural network.

²The nets are not second order because they use second order inputs, instead they used the second order derivative for calculating the weight updates. The second derivative of the error is also used to calculate saliency.

Second Order Saliency Method

	Rank	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	6	2	3	1	3	1	1	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	2	4	2	4	1	3	2	1	0	0	1
2	0	6	2	5	3	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	3	3	2	4	7	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	1	4	6	5	2	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	1	3	3	6	0	2	1	1	0	3	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	3	2	6	7
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	3	6	5	2
8	0	0	0	0	0	1	0	1	2	4	5	4	3	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	3	3	2	3	2	2
10	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	3	3	5	1	1	3
11	0	0	0	1	3	2	2	5	6	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	1	0	0	4	6	4	1	1	3	0	0	0	0	0
13	13	2	2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	1	2	5	8	4	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2	4	4	1	3
16	0	0	0	1	2	4	3	0	3	3	3	1	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	3	3	2	4	2	2	3
18	0	7	7	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	2	2	2	2	4	3	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	2	1	8	5	1	1	0	0	2	0
21	1	1	4	4	3	2	0	1	1	2	1	0	0	0	0	0	0	0	0	0	2	0

Saliency Histogram

Feature	Rank	Feature	Rank
0:	2.95	13:	0.90
1:	14.35	18:	2.05
2:	3.10	0:	2.95
3:	8.60	2:	3.10
4:	6.15	21:	4.30
5:	11.85	19:	4.70
6:	19.40	4:	6.15
7:	18.85	11:	6.45
8:	9.70	16:	7.05
9:	17.45	3:	8.60
10:	17.35	8:	9.70
11:	6.45	5:	11.85
12:	12.65	14:	12.60
13:	0.90	12:	12.65
14:	12.60	1:	14.35
15:	17.90	20:	14.85
16:	7.05	10:	17.35
17:	17.80	9:	17.45
18:	2.05	17:	17.80
19:	4.70	15:	17.90
20:	14.85	7:	18.85
21:	4.30	6:	19.40

Most important

Least important

Table 5. Node Saliency for the Ruck Data using second order terms.

4.4 Karhunen-Loève Networks for Feature Vector Reduction

The following section describes a neural network architecture for eliminating redundancy in the input feature vector by adding a preprocessing layer. This preprocessing layer coupled to a conventional backpropagation network is called a Karhunen-Loève neural network. Karhunen-Loève networks can be used to facilitate the classical pattern recognition process.

Classical pattern recognition requires a series of data reduction steps followed by application of a discriminant function. Data reduction begins by decomposing a real world event into a tangible set of measurements. Tangible measurements include photos, radar cross-section returns, and laser range data, to name a few. The next step, segmentation, requires that the events, or objects, are separated from its background and unrelated information.

Feature extraction requires that meaningful measurements be made from the separated or segmented objects. The measurements or features are grouped together and are the sole representation of the real world event to the classification engine. Sometimes in an effort to find the best set of meaningful features, multiple features are selected which convey the same information.

Classification can be handled several ways, using artificial intelligence techniques, statistical classification such as K-nearest neighbor, Bayesian, or artificial neural networks. Ruck has shown that statistical methods and neural networks are only different in their implementation (39). Because statistical techniques do not lend themselves easily to hardware implementations, neural networks offer an alternate approach. Although neural networks are fast on execution, training time can be excessive. Excessive training times are usually caused by large numbers of interconnections, and more commonly, training data that is not easily partitioned in the decision space. The two problems, complexity in the architecture and complexity in the decision space, could be reduced if the size of the input vector were reduced to the minimum number of elements required to partition the decision space.

A common method used to reduce the dimensionality of the feature vector in statistical classification is the method of principal components. The number of feature components can be reduced by taking linear combinations of the individual features. This procedure is known by several names, the method of principal components, Hotelling transform, or eigenvector transform(16). This section presents a neural network implementation, related to the discrete Karhunen-Loève transform. Using a Karhunen-Loève transform layer offers another partial solution to the *curse of dimensionality*(9) problem associated with large feature vector classification.

The following sections will describe three methods for computing the weights in a Karhunen-Loève neural network. The weight matrix is computed from the data covariance matrix. Computing the data covariance matrix can be done in several ways. Each method for computing the covariance matrix provides an alternate method for finding the Karhunen-Loève weights. This section will compare the accuracy of each method.

The first, simply called Karhunen-Loève, uses all training vectors to compute the covariance matrix. The second, the Karhunen-Loève mean uses only the in-class mean data vectors, (one for each class) to compute a covariance matrix. The third method, Fisher linear discriminants, is used to maximize the ratio between the between-class covariance and the inter-class covariance(9). The algorithms are tested against two data sets consisting of features extracted from infrared (IR) and laser range imagery for classification of tactical targets.

The following sections show how the system can be implemented as part of a neural network architecture.

4.5 The Discrete Karhunen-Loève Transformation

This section will develop a means of computing a set of weights which have Karhunen-Loève characteristics. The Karhunen-Loève weight values are generated by first constructing a two layer network which calculates an estimate of itself as the output. The network uses backpropagation to train the weights. The output of that layer, Y ,

shown in Figure 16 is used as a preprocessing layer. The output of the nodes in the Y layer will be used to replace the input vector X . If the number of nodes in the Y layer is the same as the number of input features, the input vector can be reproduced exactly. Karhunen-Loève provides a means to eliminate nodes of the Y and degenerate the estimate of X in systematic way. Figure 17 shows a Karhunen-Loève network which uses the first layer of weights from the Karhunen-Loève identity network as an input to a conventional backpropagation network.

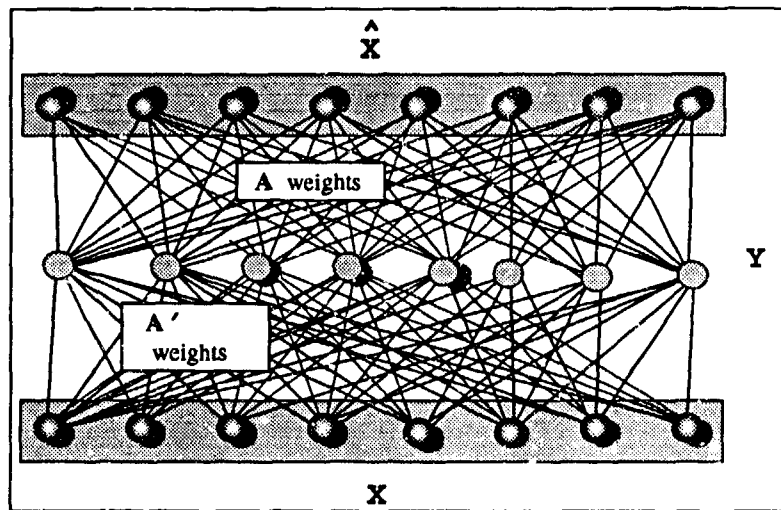


Figure 16. Network Architecture for a Karhunen-Loève Identity Network.

Y is a set of transformed input vectors(X). Since the data set Y can be used to reproduce X or at least a reasonable estimate of X (\hat{X}), Y can be used to make classifications. If an input vector can be passed through a layer of hidden nodes that is less in number than the size of the input vector, then the network has found a more compact representation of the input data. The weight matrix A' (the transpose of A) is calculated using eigenvectors of the covariance matrix of X as will be shown below.

Figure 17 shows how the network is organized. A set of input vectors X is propagated through the net to the first hidden layer of nodes Y . Propagation is the result of a matrix multiplication of the input vector X times the weight matrix A' .

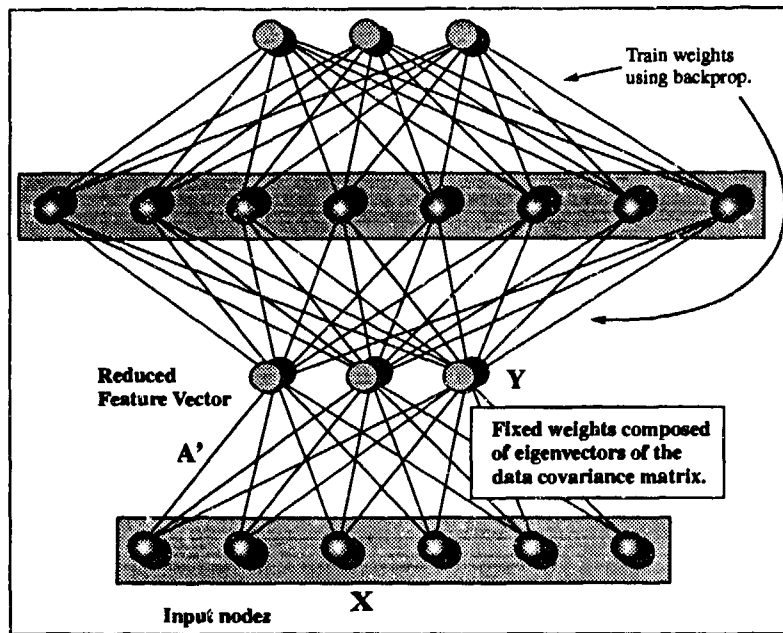


Figure 17. Network Architecture for a Karhunen-Loève preprocessor.

In matrix notation, a random vector X , could be represented as a column vector.

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \end{pmatrix} \quad (12)$$

X , the feature vector, could be represented without error, by a transformation of the vector by matrix multiplication with A such that:

$$X = AY$$

$$x_i = \sum_{i=1}^n y_i A_{ij}$$

and

$$A = [A_1 \ A_2 \ A_3 \ \cdots \ A_n] \text{ and } |A| \neq 0$$

The matrix A represents the weight matrix, and the A'_i 's are single column vectors of the weight matrix. The A'_i 's represents the weights connected to a single hidden node. Here the transformation matrix A does a simple rotation of the vector Y to the vector X .

If A is selected such that the columns of A are orthonormal and form n linearly independent basis vectors then:

$$A'A = I \text{ and } A^{-1} = A' \tag{13}$$

Under these conditions the new feature vector Y can be expressed as

$$Y = A'X$$
$$y_j = \sum_{i=1}^n A'_{ij}x_i \tag{14}$$

$$\tag{15}$$

Consequently, one can go between Y and X interchangeably. If a transformation matrix A could be found such that, for a n -dimensional vector X , transformed to a m -dimensional vector Y whose $m < n$ components were either constant or zero for all X , then those extra components of the feature vector could be eliminated from the learning machine. They contribute no extra information.

However, the object is not to reproduce X exactly, but to provide an estimate of X under the constraint that the dimensionality of Y is less than X . The procedure would be to replace those components y_i , $m < i < n$, with their mean and create an estimate, \hat{X} . If the dimensionality of Y is the same as X , then X could be reproduced exactly.

But, if the dimensionality of Y is less than X only an estimate is possible. If the estimate is *good enough*, the estimate can be used for classification. Since Y contains the same information as \hat{X} , Y could be used instead.

If, for every X propagated through the network, some of the Y nodes were either very small, zero or constant, no matter which X vector is used, they are not necessary to reproduce X . If a node is constantly zero, or near zero, it can be eliminated. If it were constant, a constant (b_i) times the particular value of A could be added to the X component. From a neural network stand point, it becomes part of an offset value added to each component. That element of Y could be eliminated. Equation 16 shows replacement of Y nodes with their average value. The b_i coefficients are constant for all X , while the y_i coefficients are variable for each X .

$$\hat{X}(m) = \sum_{i=1}^m y_i A_i + \sum_{i=m+1}^n b_i A_i \quad (16)$$

As will be shown, b_i coefficients are the mean outputs of the deleted components for the Y_i . If the deleted components are replaced by their mean value, the expected error in the transformation will be minimum. The error represented by the transformation is:

$$\vec{\epsilon} = X - \hat{X}(m) = \sum_{i=m+1}^n (y_i - b_i) A_i. \quad (17)$$

Where ϵ is a vector of differences between X and \hat{X} . Components of Y are selected for deletion to minimize the error. The error will be random unless it can be minimized. The expected value of the mean squared error measures how effective the transformation from Y to \hat{X} actually is.

$$\begin{aligned}\overline{\varepsilon^2(m)} &= E\{\varepsilon'\varepsilon\} \\ &= E\left\{\sum_{i=m+1}^n \sum_{j=m+1}^n (y_i - b_i)(y_j - b_j)A'_i A_j\right\} \quad (18)\end{aligned}$$

$$= \sum_{i=m+1}^n E(y_i - b_i)^2 \quad (19)$$

where $E(x)$ is the expected value of x . By taking the partial derivative of equation 19 the error can be minimized and the optimum value for the replacement values for y_i 's can be determined.

$$\frac{\partial}{\partial b_i} \overline{\varepsilon^2(m)} = -2E\{(y_i - b_i)\} = 0.$$

The best choice for the constants would be

$$\begin{aligned}b_i &= E\{y_i\} \\ &= \bar{y}_i \\ &= A'_i E\{X\}.\end{aligned}$$

From equation 18, the error also can be expressed as:

$$\begin{aligned}\overline{\varepsilon^2(m)} &= E\left\{\sum_{i=m+1}^n (y_i - b_i)(y_i - b_i)\right\} \\ &= E\left\{\sum_{i=m+1}^n (y_i - \bar{y}_i)(y_i - \bar{y}_i)\right\} \\ &= E\left\{\sum_{i=m+1}^n A'_i (X - \bar{X})(X - \bar{X})' A_i\right\} \\ &= \sum_{i=m+1}^n A'_i \Sigma_X A_i.\end{aligned}$$

Where Σ_X is the covariance matrix of X . By selecting the A_i basis vectors, as the eigenvectors of Σ_X :

$$\Sigma_X A_i = \lambda_i A_i$$

and since $A_i' A_i = 1$,

$$\lambda_i = A_i' \Sigma_X A_i$$

which in turn defines the mean squared error as:

$$\overline{\varepsilon^2(m)} = \sum_{i=m+1}^n \lambda_i \quad (20)$$

Next the eigenvalues of Σ_X are computed. Each eigenvalue provides an eigenvector which is used as the weights connecting the input vector X with y_i . The increase in error is dependent on each eigenvalue. If the λ_i 's are rank ordered, highest to lowest, the components (y_i 's) of the transformed vectors Y will also be rank ordered according to its variance over all of X . The amount of mean squared error for a particular selection of m is equal to the sum of the $m - n$ eigenvalues remaining in the rank-ordered set of eigenvalues for Σ_X .

In simpler terms, the eigenvalues λ_i 's are equal to the variances of the y_i 's. Replacing those components of Y with their mean has the least effect if the variance is small. Since the objective is to classify rather than to reproduce X , the mean value (b_i) can be ignored altogether. Consequently the new vector Y can be presented to the network instead of X for training and classification.

Some modifications remain before the neural network can train efficiently on the transformed data. Ruck (40) has shown that the backpropagation algorithm is a degenerate

form of the Kalman filter. One result of the degeneration is the lack of an adaptive weight learning rate. Every weight update must use the same learning rate.

Lack of an adaptive learning rate forces some constraints on the network. These constraints can ensure convergence in a reasonable amount of time. When a vastly different dynamic range exists between input nodes, the strongest node (the one with the greatest dynamic range) tends to dominate the training. Over time the weights would converge if the hardware could perform exact arithmetic, but hardware is never perfect. Training under these conditions makes convergence more susceptible to errors introduced by round-off errors and other hardware constraints. To make training easier, each weight update value should be about the same for each node. This is possible only when the general range of the data for each node is about equal. The best way to ensure this is to statistically normalize the data after the network Karhunen-Loève transform.

A consequence of rank ordering the eigenvectors is that the first feature of the Y matrix has the largest dynamic range, followed by the next and so on. Rank ordering is necessary to decide how many features to keep. After the decision has been made, the data needs to be normalized to ensure the network training algorithm doesn't have to work too hard to overcome the natural differences in dynamic ranges of the features. Normalization, as used here, does not refer to the conventional energy normalization. Energy normalization adjusts the total squared energy in a given vector to one. Here statistical normalization is used. Statistical normalization adjusts each feature component to have a mean of zero and a standard deviation of one. In effect, the feature value of every vector is replaced by its statistical Z-score

$$x_{i_{new}} = \frac{x_i - \bar{x}_i}{\sigma_{x_i}} \quad (21)$$

computed as shown in Equation 21. The statistics are taken over the training data set only. Then, the new feature values are computed for the entire data set, using the statistics of the training set. The process lends itself to neural network implementation as a single layer

with equal inputs and outputs with only one weight connected to the next layer. Statistical normalization ensures that the network training can converge in a reasonable amount of time.

The following tests use three methods to calculate the covariance matrix of \mathbf{X} . The first method, Karhunen-Loève networks, uses a covariance matrix based on all the data exemplars. The method works well but the neural network system can be improved if the effect of outliers and incorrectly labeled data points can be reduced. The Karhunen-Loève mean and Fisher linear networks do reduce the effect of bad data points. For Karhunen-Loève mean, instead of using all data points to calculate the covariance matrix, only one exemplar for each class is used. That exemplar is computed by taking the mean vectors of all input vectors which are members of the particular class. Duda and Hart (9) suggest an additional modification to the process which actually pushes the separate classes apart in the decision space. They suggest building a covariance matrix from both the between-class means and another from the within-class or inter-class means. By taking the ratio of the two covariance matrices, the separation of classes should be optimum.

The computation is a variation on the standard rules for computing a covariance matrix. Let $\bar{\mathbf{X}}_{IC}$ denote the intra-class mean vector and $\bar{\mathbf{X}}_{OC}$ denote the out-of-class mean vector. The matrix Σ_{FL} is computed by taking the inverse of the intra-class (IC) covariance times the between class covariance to compute the eigenvectors.

$$\Sigma_{X_{FL}} = E [(\mathbf{X} - \bar{\mathbf{X}}_{IC})(\mathbf{X} - \bar{\mathbf{X}}_{IC})'] \cdot E [(\mathbf{X} - \bar{\mathbf{X}}_{OC})(\mathbf{X} - \bar{\mathbf{X}}_{OC})']^{-1} \quad (22)$$

This method is called Fisher linear discriminants.

Figure 18 illustrates the process for computing the weights in the Karhunen-Loève layer. Starting with a set of random training vectors, a covariance matrix is computed. Next the eigenvalues are computed and rank ordered, eliminating the smallest. For the following

data sets, the data was rotated by multiplying the entire data set by the eigenvectors. Finally the data set is normalized using the statistics of the training set. The next section compares a Karhunen-Loève network with standard backpropagation.

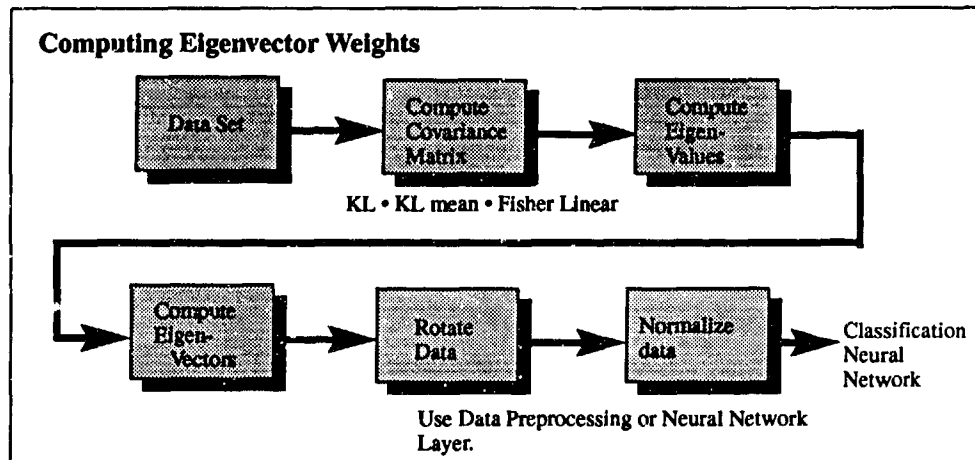


Figure 18. Computing the Rotation Matrix. The weights for the Karhunen-Loève layer are computed as illustrated. First a covariance matrix is computed from the data set. The covariance matrix can be computed in one of three ways. The data is rotated into the new decision space. Data rotation can be implemented as preprocessing of the input vectors or as a neural network layer.

4.6 A Karhunen-Loève Network.

This section will present the results of a test of the Karhunen-Loève preprocessing layer. A system for Karhunen-Loève transforms preprocessing was implemented as a menu item to the NeuralGraphics neural network simulator(44). The three types of Karhunen-Loève networks were tested. The routines are part of the data normalization and preprocessing menu. Data normalization and preprocessing can be toggled between no normalization, statistical normalization, energy normalization, Karhunen-Loève, Karhunen-Loève mean, and Fisher Linear Discriminants.

To test the efficiency of the transformation, several test sets were processed. A set of data consisting of some imagery taken from infrared and laser radar range data was used.

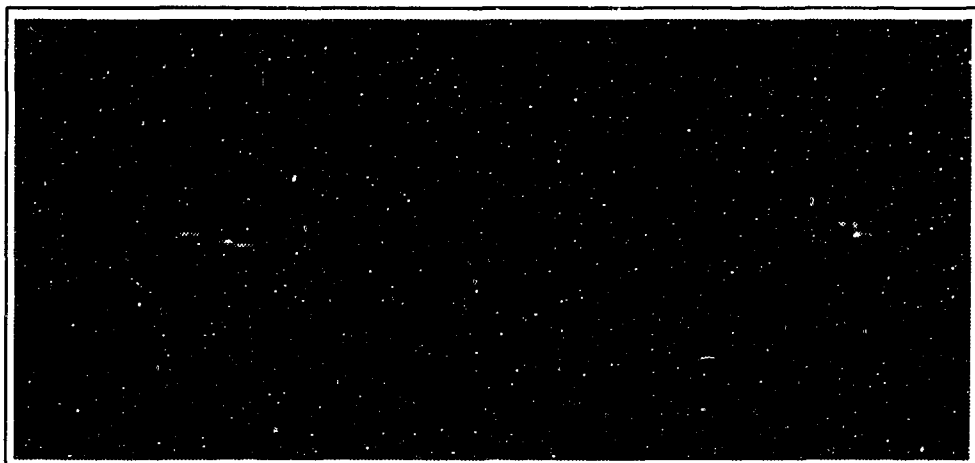


Figure 19. Infrared Tactical Target Imagery.(34)

The image in Figure 19 shows infrared tactical targets. The object of the segmentation function is to label each pixel as part of a target or part of the background. The segmentation is performed by thresholding a Gabor filtered version of the image as shown in Figure 20. A feature vector was constructed from the segmented blobs, based on shape characteristics.

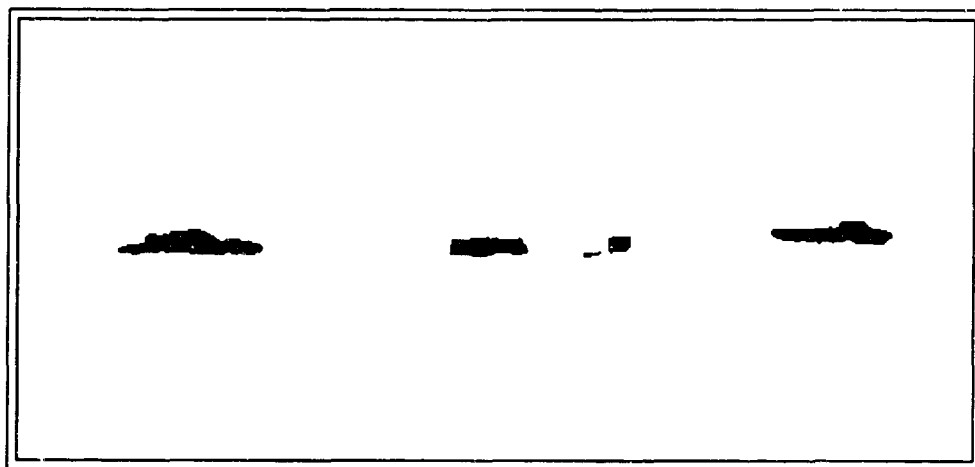


Figure 20. Gabor Segmented Data.(2)

The range data was collected as part of a study to fuse information from a multisensor suite for target identification. The procedure was tested against two data sets: the Ruck

(35) data set and the Roggemann set (34). The Ruck set contains 78 exemplars with 22 input features. The Roggemann set consisted of 400 exemplars with 12 input features.

The first set tested was the Ruck data set taken from Avionics Laboratory laser range data. Shapes were segmented, using a histogram based technique, from the infrared images. Feature extraction provided the first 11 Zernike moments from the silhouette, and 11 moments taken from the border of the object, making 22 separate features. The set was trained for 100,000 training iterations using a random presentation order of the data set. The set consisted of 78 exemplars with 52 for training and 26 for testing. To establish a base line, the network was trained from 50 initial random conditions.

The error for a particular count or training exemplar was averaged over each training cycle for each data set. Each data set was trained using three, six, nine or twelve nodes in the Karhunen-Loève layer. Only the training error reduction is plotted because reduction of error for the test set over time provides no information about the simplification of the decision space. Still, it is important that simplification of the decision space does not reduce the capacity to generalize. Statistics for mean squared error and accuracy are summarized in Tables 6 and 7. The graphs which follow show the results for training the networks for one hundred thousand iterations.

The error plot in Figure 21 shows that same total error can be maintained with as few as six of the 22 nodes. Significant improvement is possible with as few as nine of the 22 nodes. In Figure 21 comparable accuracy was possible by reducing the input vector from 22 nodes to only six nodes. Note that the accuracy is maintained with as few as six of the 22 input nodes.

Figure 21 shows the total error for a random set of weights is about ten and the final error is about two. This corresponds to an initial classification rate of about 50 percent with a final classification rate of about 98 percent after training.

One of the advantages of Karhunen-Loève preprocessing is that it can be used to not only reduce the complexity of the problem, but to simplify the network architecture as

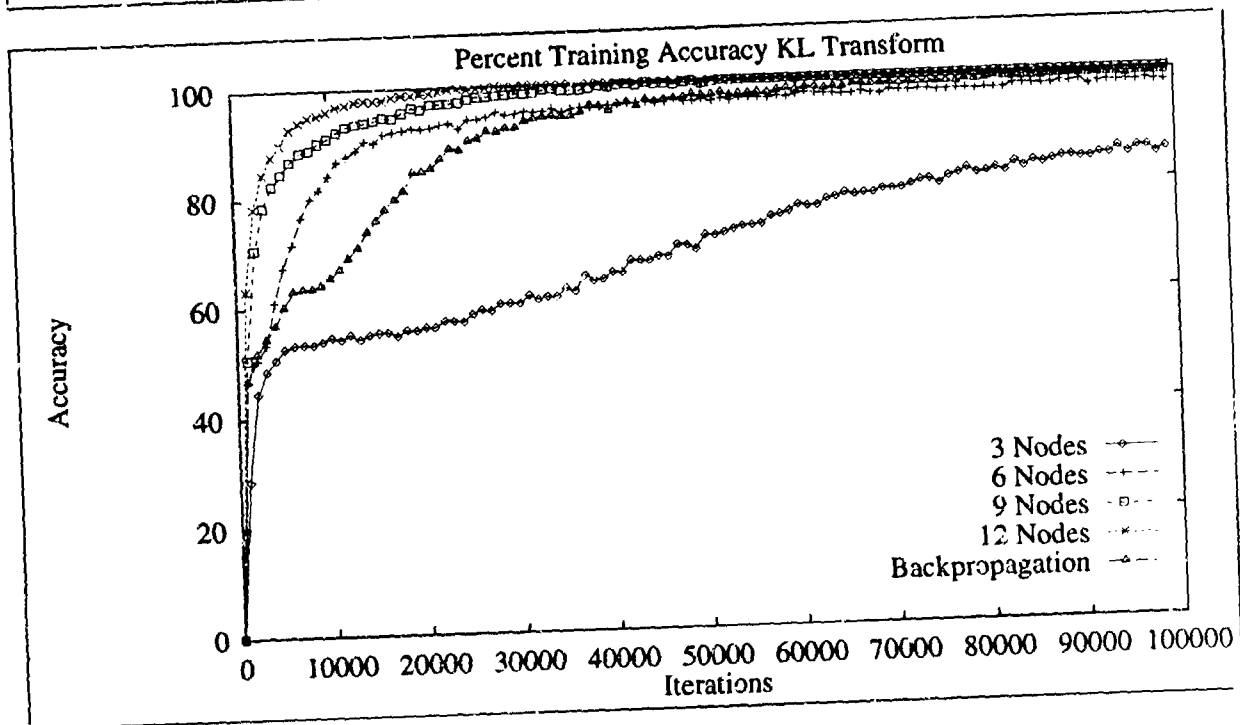
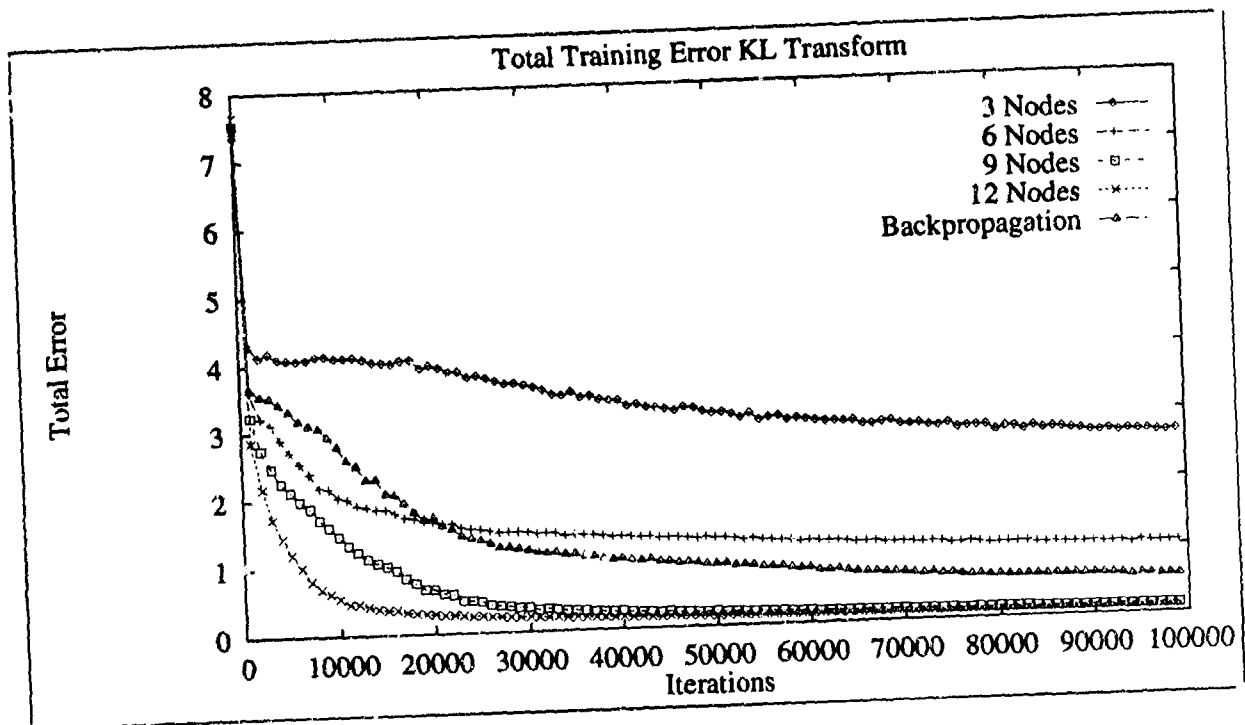


Figure 21. Ruck Data Using Karhunen-Loève Transform. Note that using only nine and twelve input nodes, the Karhunen-Loève transform net performed better than a backpropagation network. Equal performance was obtained using only six of 22 input nodes. Similar levels of accuracy are maintained for 6,9,and 12 of 22 inputs.

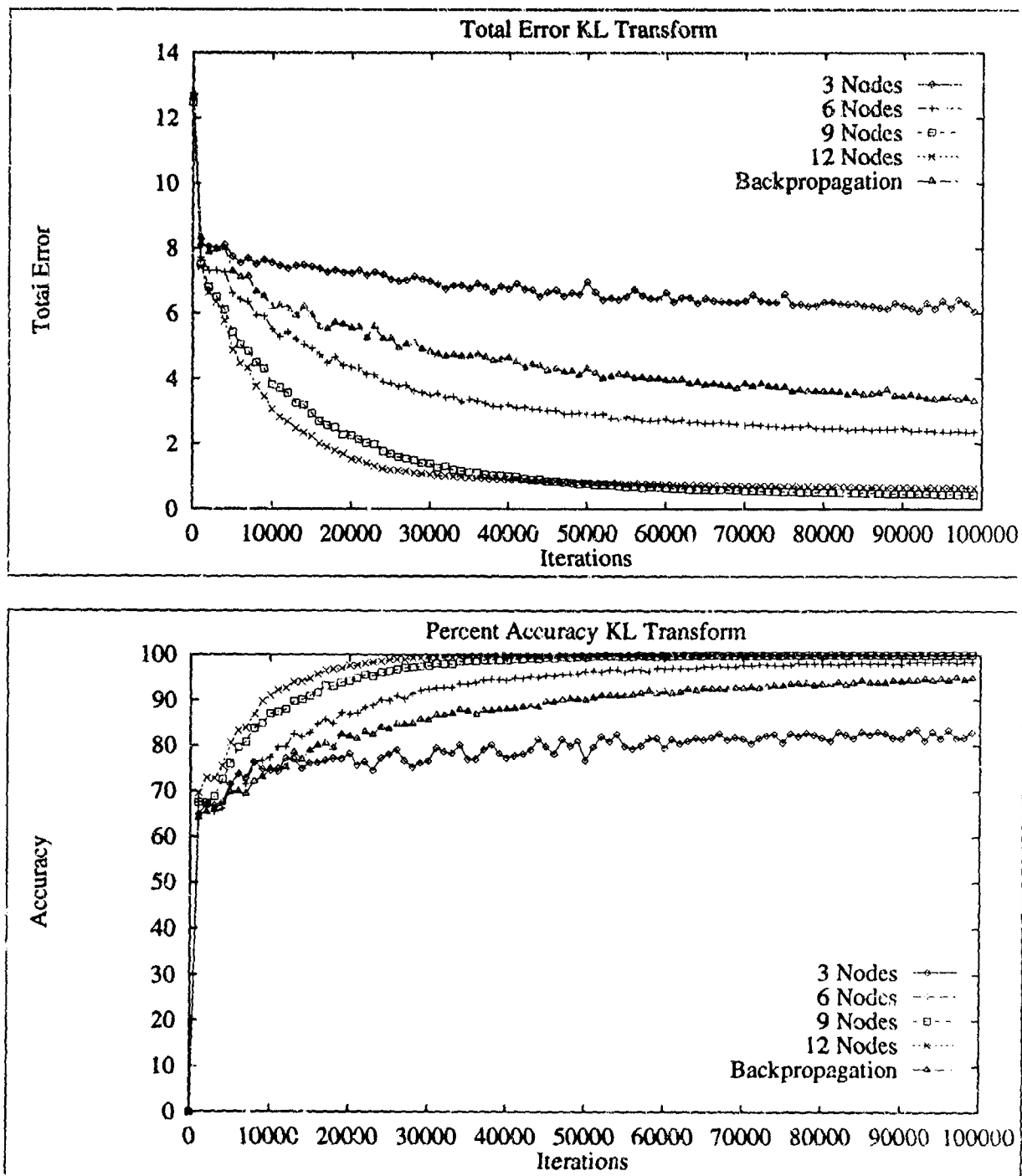


Figure 22. Roggemann Data Using Karhunen-Loève Transform. The Roggemann data contains only twelve features. Transforming all twelve, nine and six nodes results in better performance than using backpropagation alone. Similar classification performance is possible using only half the number of input nodes.

well. Each graph in Figure 21 shows the error history and accuracy over time for a network trained with three, six, nine and twelve input nodes. Determining which nodes can be deleted is simplified because the ordered eigenvector transform places the most important nodes first, and the least important nodes last. One of the problems encountered was that outliers in the training set could significantly influence the eigenvalues. To alleviate this problem a change was made to the algorithm to reduce the effects of outliers in the data set. The Karhunen-Loève mean and the Fisher linear networks were the result of an effort to reduce the effect of outliers on the classification. The results are shown in Figures 22 through 26.

Although reducing training time is useful, there is no advantage if the capability to generalize is lost. Generalization can be tested by presenting the network with data it hasn't seen before. By partitioning the data into one-third test and two-thirds training data the following results were demonstrated. In the previous tests, the test data was examined after training the net. The results are shown in Tables 2 through 7. To compare each of the feature reduction methods, one more category was added to the table. Noted as *most significant* features were selected as those chosen by the weight saliency test as the most significant. The net was trained using only these, the most significant features. No Karhunen-Loève transformation was performed.

The tables show that Fisher linear discriminants, Karhunen-Loève and Karhunen-Loève mean all performed about as well. A fourth case (*most significant*) was added for comparison. The most significant entries indicate a neural network was trained using only the number of input features in the Nodes column, the features selected was determined by the saliency metric. Karhunen-Loève results were superior to using saliency alone.

The best results came from the Karhunen-Loève mean algorithm and the worst from Fisher linear. The difference between the three was statistically insignificant. For the Ruck set, the classification accuracy was only reduced by a few percent using only six input nodes (with a Karhunen-Loève mean transform on the input). This means that the performance was similar to backpropagation using only one-quarter of the input nodes.

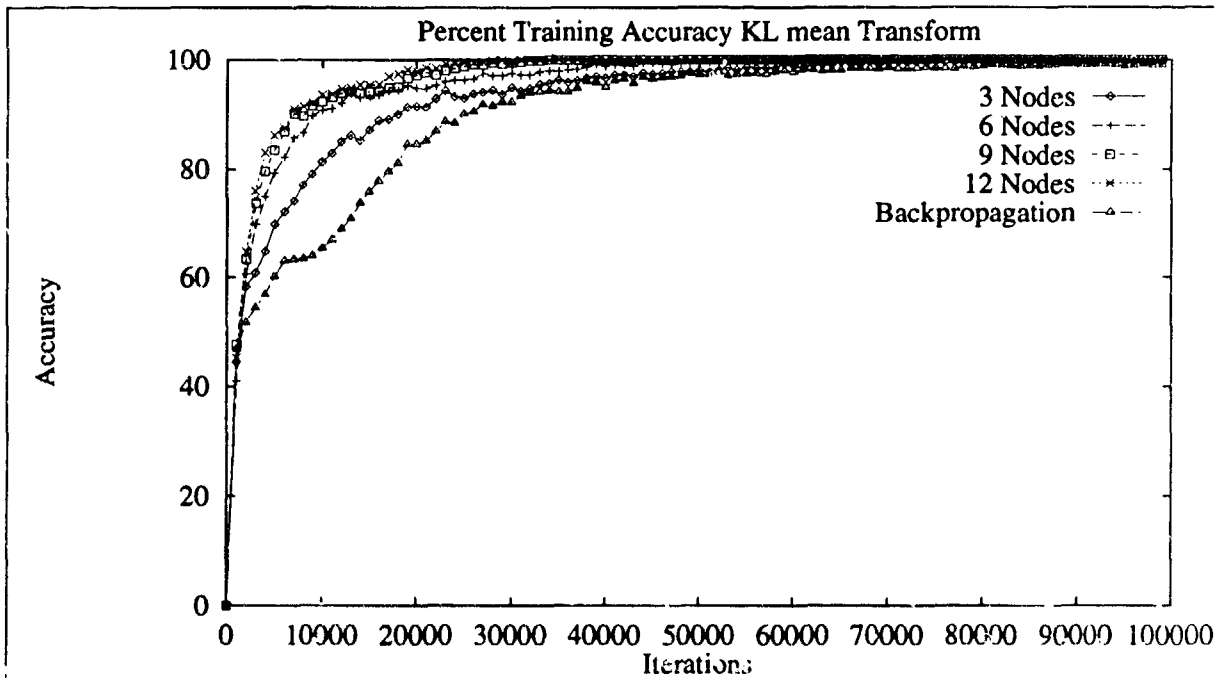
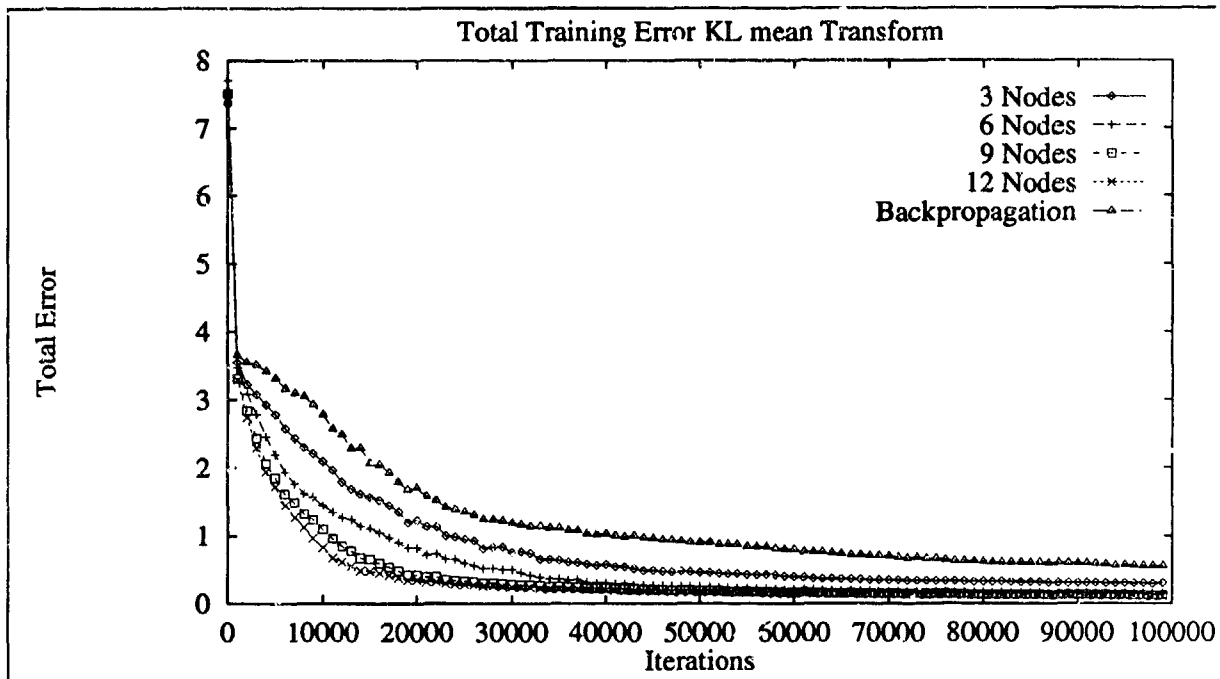


Figure 23. Ruck Data Using Karhunen-Loève mean Network. The Karhunen-Loève mean transforms performed better than Karhunen-Loève on this problem. Total reduction of error increased, and accuracy was better than backpropagation in every case.

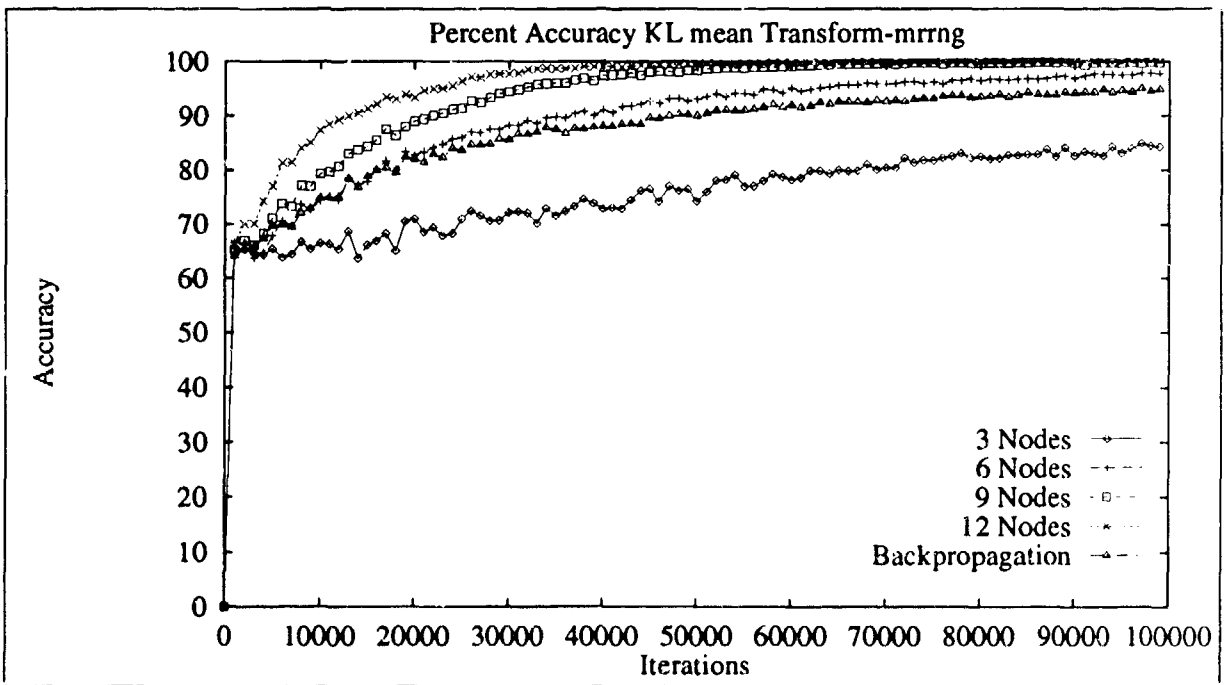
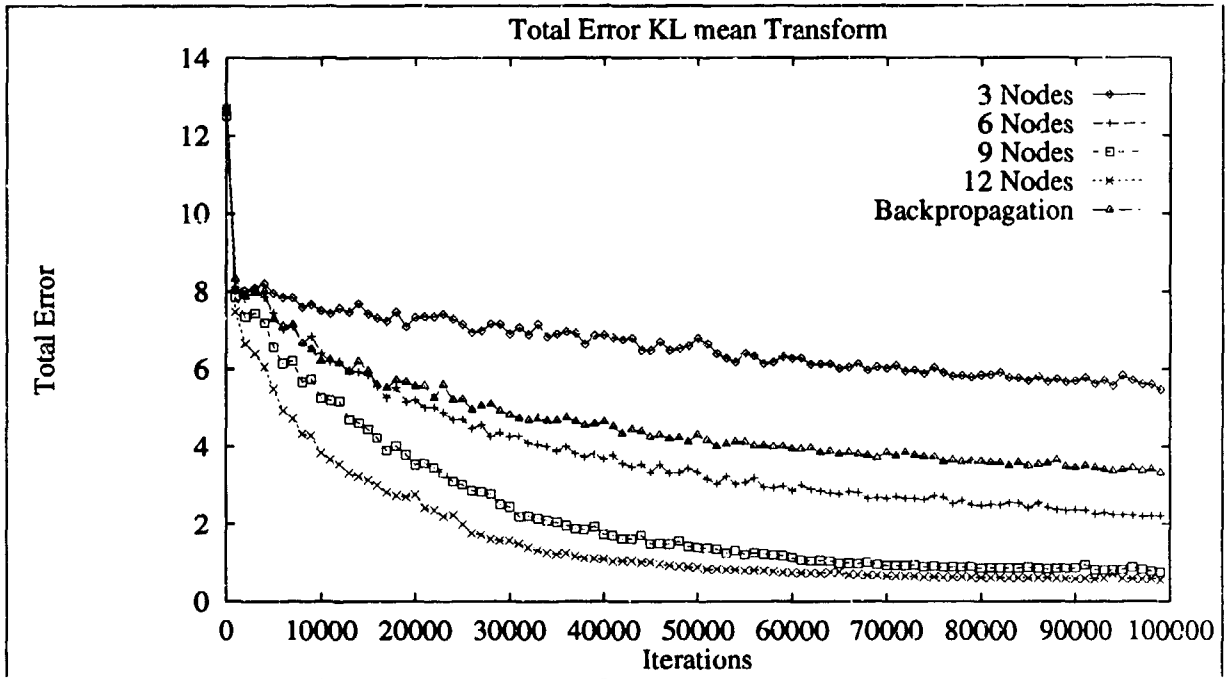


Figure 24. Roggemann Data Using Karhunen-Loève mean Network. Performance as indicated by faster reduction of error is better: for a Karhunen-Loève mean network than for the Karhunen-Loève network.

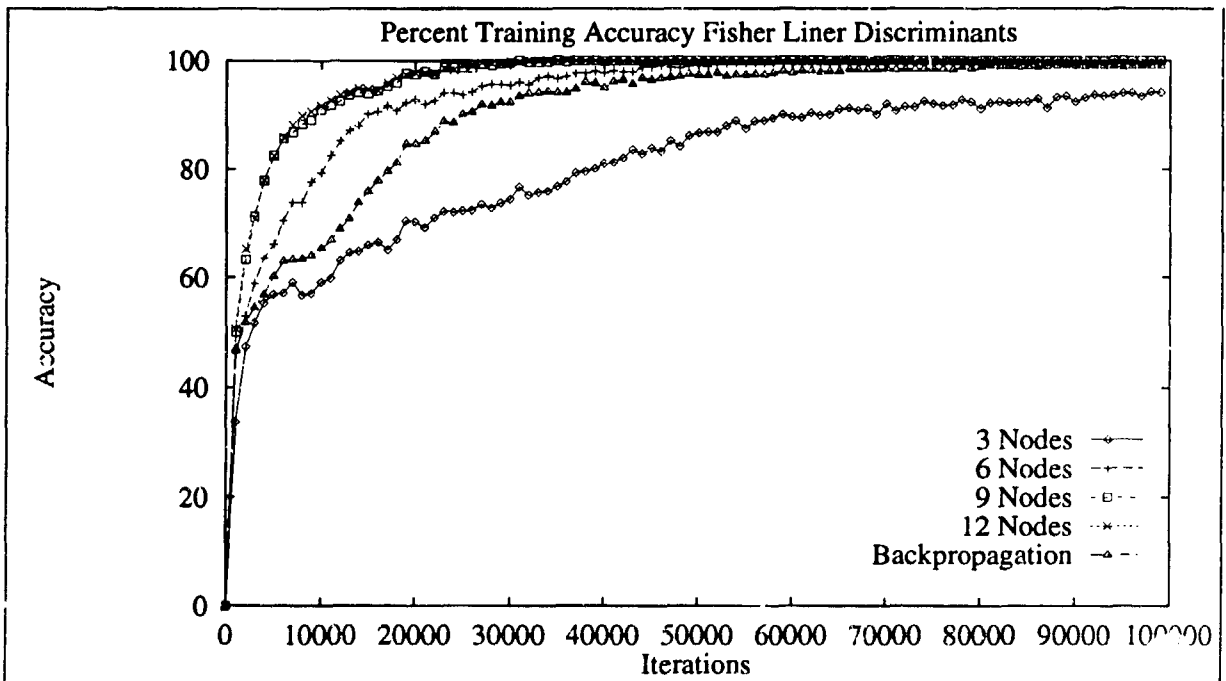
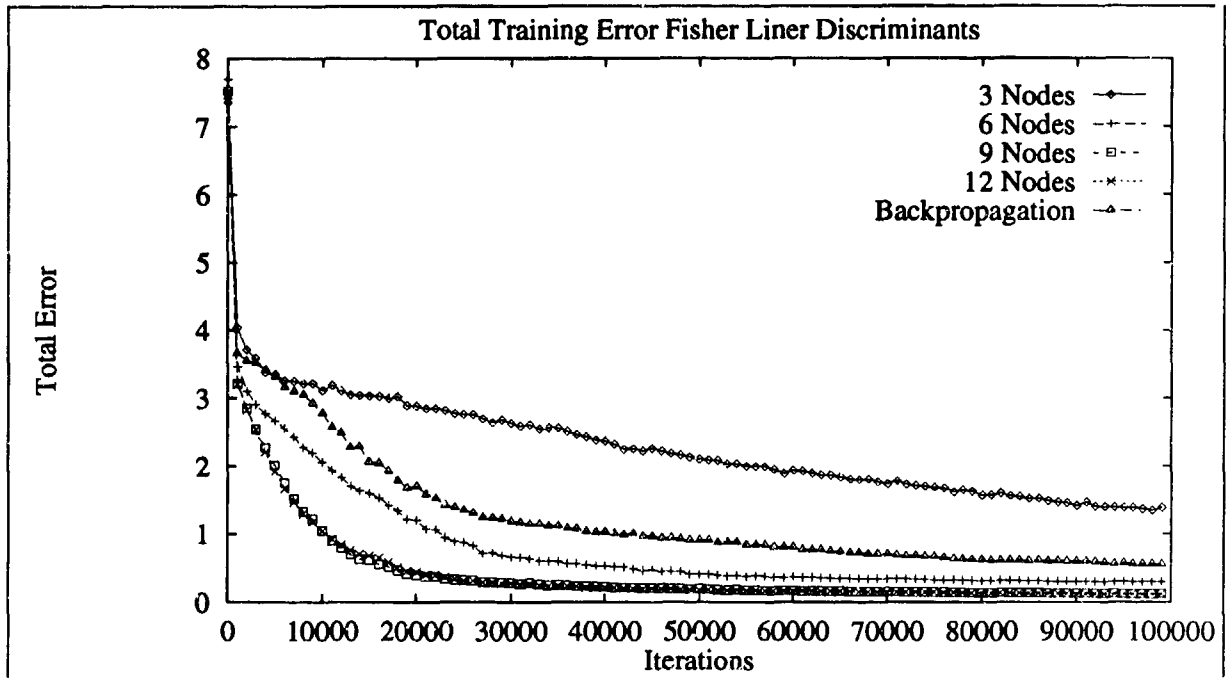


Figure 25. Ruck Data Using Fisher Linear Network. Training times were improved over Karhunen-Loève but Karhunen-Loève mean performed better.

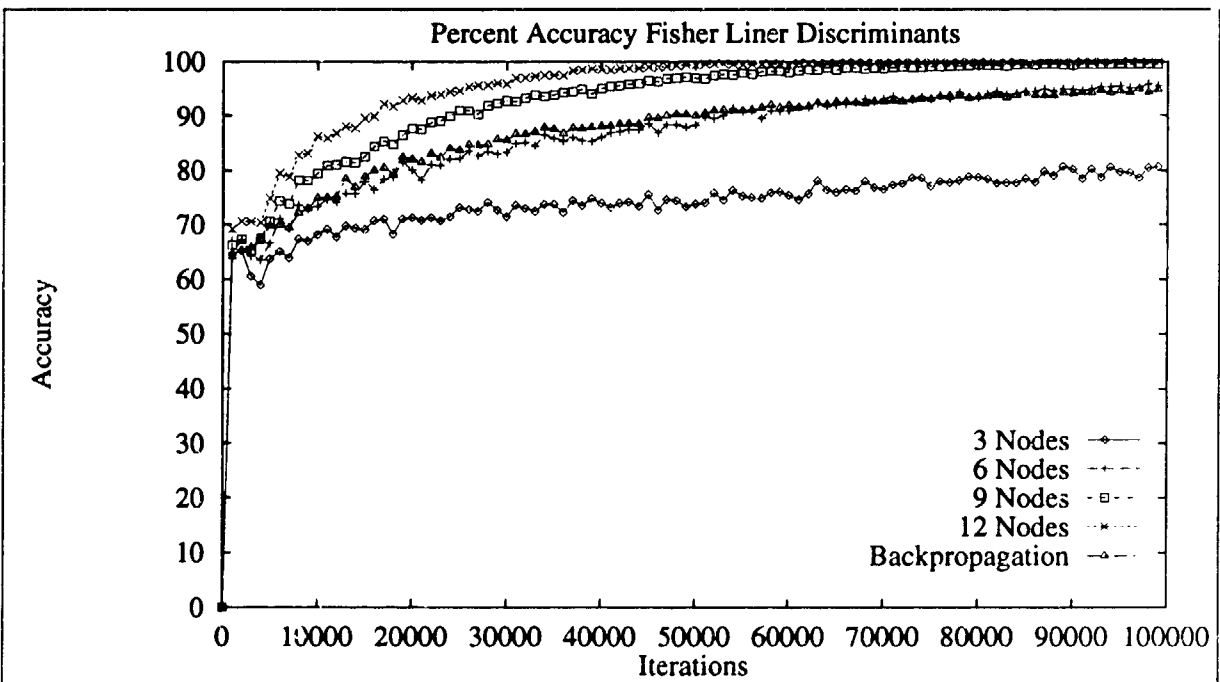
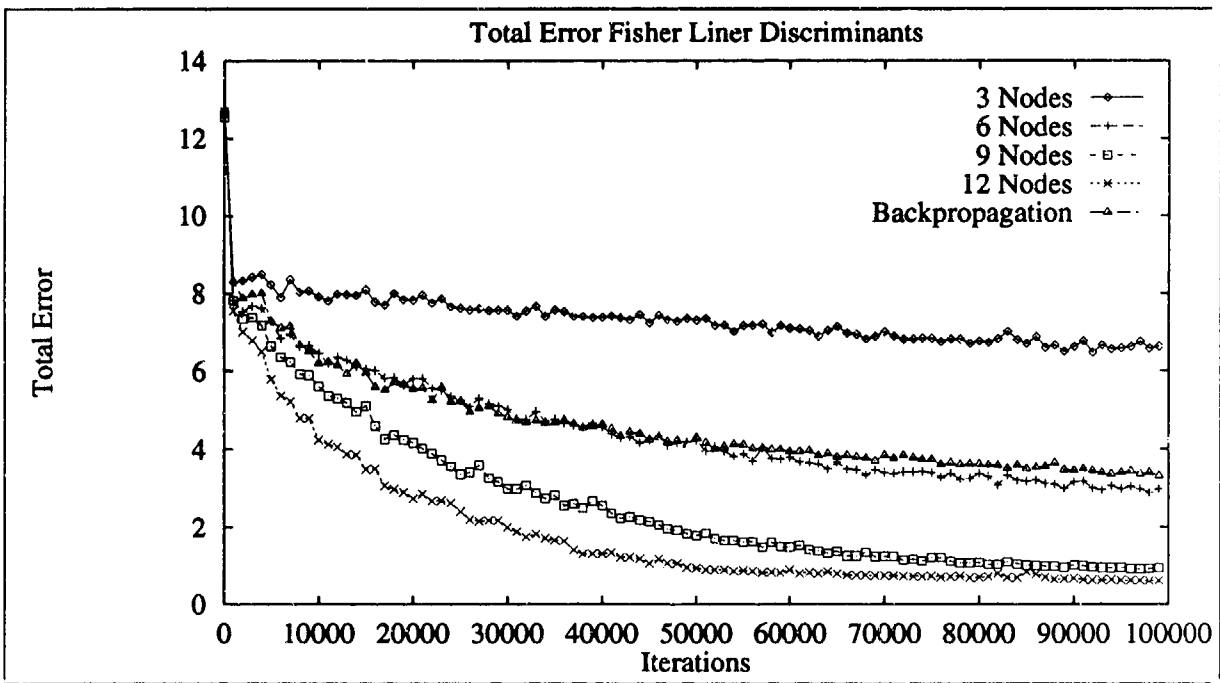


Figure 26. Roggemann Data Using Fisher Linear. Fisher Linear discriminants did not perform as well as either Karhunen-Loève, or Karhunen-Loève mean.

Comparison Results for Ruck Data				
File	Algorithm	Nodes	Train	Test
Ruck	Backprop	22	100.0	74.4
	Karhunen-Loève	3	86.0	51.9
	Karhunen-Loève mean	3	100.0	72.9
	Fisher linear	3	100.0	74.4
	Most Significant	3	98.2	68.1
	Karhunen-Loève	6	98.0	69.2
	Karhunen-Loève mean	6	100.0	73.0
	Fisher linear	6	100.0	76.4
	Most Significant	6	100.0	69.8
	Karhunen-Loève	9	100.0	63.0
	Karhunen-Loève mean	9	100.0	74.4
	Fisher linear	9	100.0	74.1
	Most Significant	9	100.0	68.1
	Karhunen-Loève	12	100.0	66.3
	Karhunen-Loève mean	12	100.0	77.8
	Fisher linear	12	100.0	70.4
	Most Significant	12	100.0	66.4

Table 6. Ruck Data Generalization. This table compares the results for the three types of Karhunen-Loève transformation to standard backpropagation. For 50 runs to 100K iterations, best generalization was possible with 12 of 22 nodes, using Karhunen-Loève mean. Karhunen-Loève mean with only 9 nodes performed as well as standard backpropagation.

Comparison Results for Roggemann Data				
File	Algorithm	Nodes	Train	Test
Roggemann	Backprop	12	100.	84.7
	Karhunen-Loève	3	82.0	74.2
	Karhunen-Loève mean	3	91.4	74.1
	Fisher linear	3	90.3	72.9
	Most Significant	3	98.2	75.0
	Karhunen-Loève	6	95.2	81.3
	Karhunen-Loève mean	6	99.2	83.7
	Fisher linear	6	94.5	79.4
	Most Significant	6	99.4	79.4
	Karhunen-Loève	9	99.6	81.3
	Karhunen-Loève mean	9	100.0	83.9
	Fisher linear	9	99.4	82.7
	Most Significant	9	100.0	84.5
	Karhunen-Loève	12	99.8	91.2
	Karhunen-Loève mean	12	100.0	86.2
	Fisher linear	12	98.2	84.3
	Most Significant	12	100.0	84.7

Table 7. Roggemann Data Generalization. Generalization was better for Karhunen-Loève and Karhunen-Loève mean when the same number of input nodes were used.

For the Roggemann set, the Karhunen-Loève mean generalized better than the standard Karhunen-Loève nets, again the results were mixed.

The results show that a single layer composed of the elements of the eigenvector rotation matrix can reduce the complexity of the decision space, increase generalization and improve performance. The full Karhunen-Loève transform seems to be best for memorization of data with good generalization, while the Karhunen-Loève mean transform reduces the number of nodes required for similar but sub-optimal performance compared with the Karhunen-Loève .

Fisher linear discriminants performance is similar to that of the Karhunen-Loève mean, but their use is complicated by the necessity of taking the inverse of the intra-class matrix. Sometimes the inverse does not exist. The additional complexity over the

Karhunen-Loève mean, plus the possibility of total failure for non-invertible matrices suggests a limited application except with problems involving pristine data sets.

4.7 Identity Networks

This section will describe the use of a generalized identity network to compute an input transformation with some of the characteristics of Karhunen-Loève networks. The architecture of the identity network is similar to the network shown in Figure 16. The previous section showed that an input layer which rotates the feature vector into a better decision space can improve training times with fewer backpropagation nodes. The Karhunen-Loève nets, which were based on eigenvectors, do have a few problems which might be solved using neural network training techniques as well. The major problem with the Karhunen-Loève networks is that the computation of eigenvectors requires the inverse of what can be a large matrix. Neural network techniques allow iterative learning to replace the matrix inversion. Two methods will be presented: Identity(12, 31) networks and Gram-Schmidt networks. Both methods work the same as before. The only difference is the method used to determine the weight matrix A . Both networks are referred to as identity networks because training the weights begins by forcing the network to learn a lower dimensional representation of the data.

Identity classification networks train in stages. First a multilayer network is trained with the input also being the desired output. A single hidden layer is normally used. Once the network can faithfully reproduce the input vectors, the output layer of weights are discarded. The input weights are then used as a preprocessing layer to feed another type of network. This provides an input layer to a classification network (like backprop) with a smaller feature vector.

Using a network as described in Figure 16, identity networks relax the requirement on the A matrix (weights) presented in equations 13 and 15. No requirement is made that A be invertible, or orthogonal. The only requirement is that:

$$\begin{aligned}
 Y &= AX \\
 f_h(Y)B &= \hat{X}.
 \end{aligned}
 \tag{23}$$

Equation 23 says that the first layer of weights (A) times a random X generates an activation vector Y . Next, a sigmoid of Y is multiplied times a matrix B and an estimate of X is generated.

If the dimensionality of Y is less than X , and X can be reconstructed to the desired accuracy, then the network has discovered a lower dimensional representation for X . Because A and B are weight matrices and Y is the output of the first layer, A , B , and Y can be determined by training a network using backpropagation such that X is both the input and the desired output.

The training for an identity network is performed in two phases. First the network is trained to reproduce the original training vectors. After the input vectors can be reconstructed to the desired accuracy, the first layer training is turned off and the output is used for the input to another feedforward network. After the first network is trained, training is stopped on the first layer. The output of the first hidden layer (Y) is fed to a (different) two layer backpropagation (or any feedforward) network. The procedure was tested against both the Ruck and Roggemann data sets, under the same conditions as before, in section 4.6. Training two networks doubles the number of network parameters. One question is how long should the identity layer be trained? A number of stopping points were tried, from very short (5000 iterations) to very long(100,000 iterations). For the result reported in the next section, a relatively short training time was selected. Very little change was noticed in the overall speed of convergence and the identity error after a few thousand iterations. For that reason, the identity layer of the network was trained for only 5000 iterations.

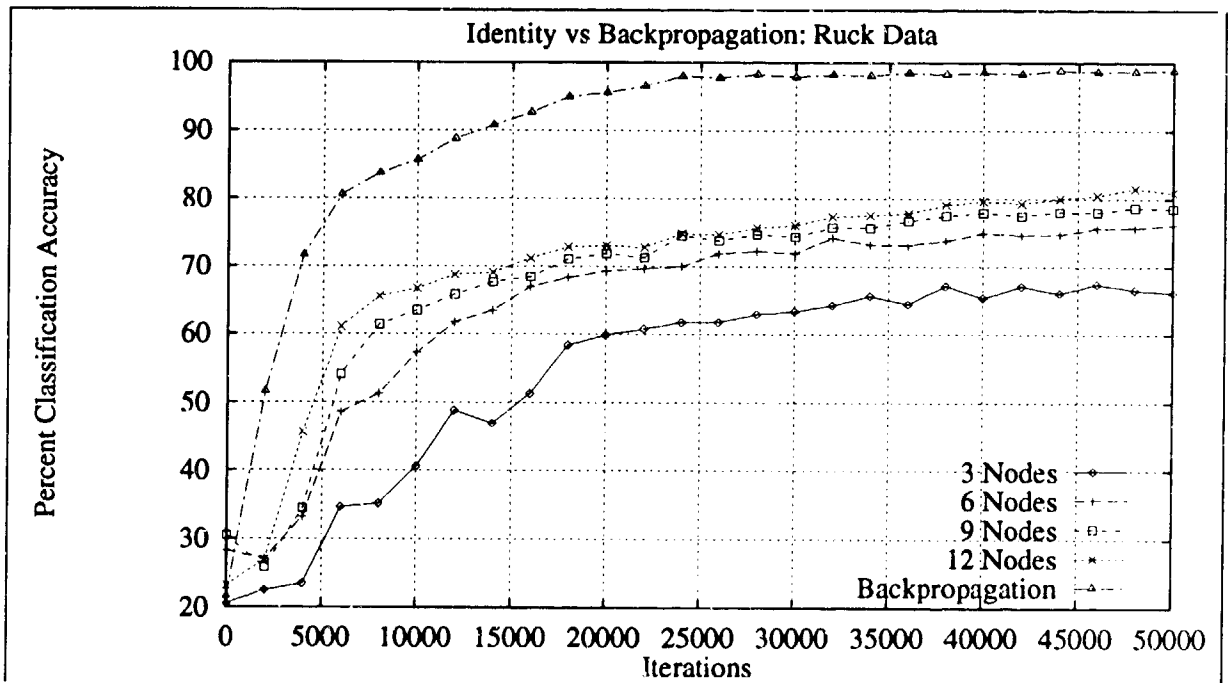
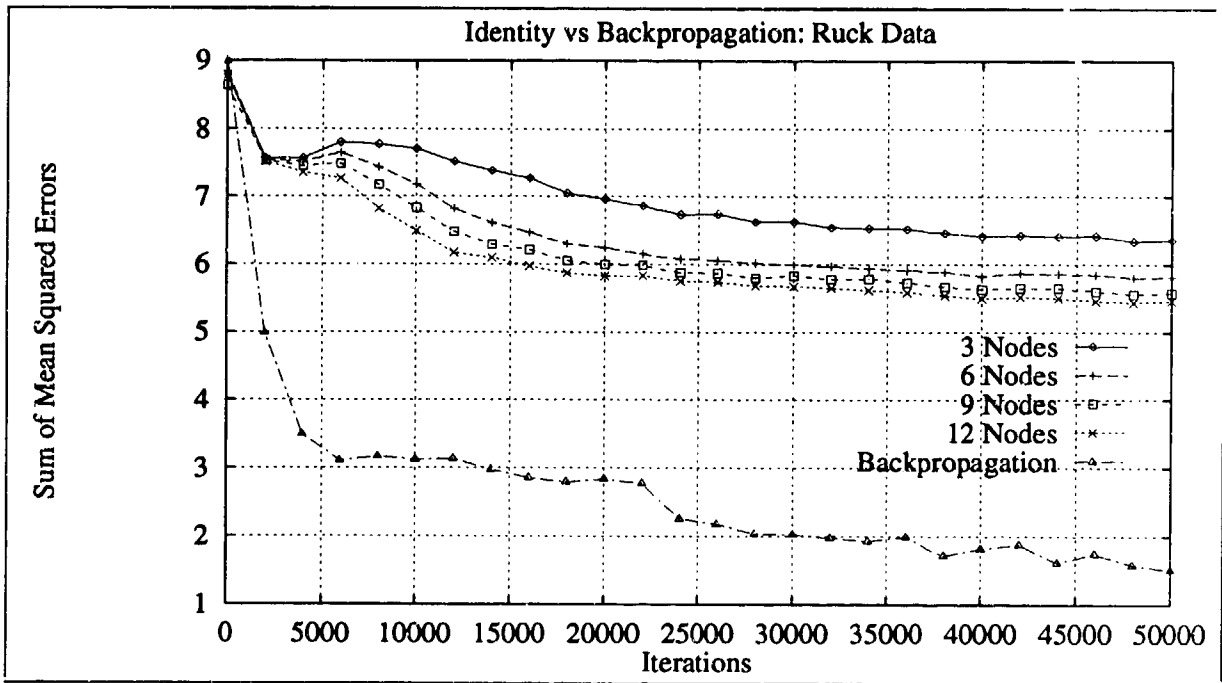


Figure 27. Ruck Data Using Identity Networks. The networks were unable to achieve the accuracy of the Karhunen-Loève networks. Still the results trained up to 80 percent with only six of 22 nodes.

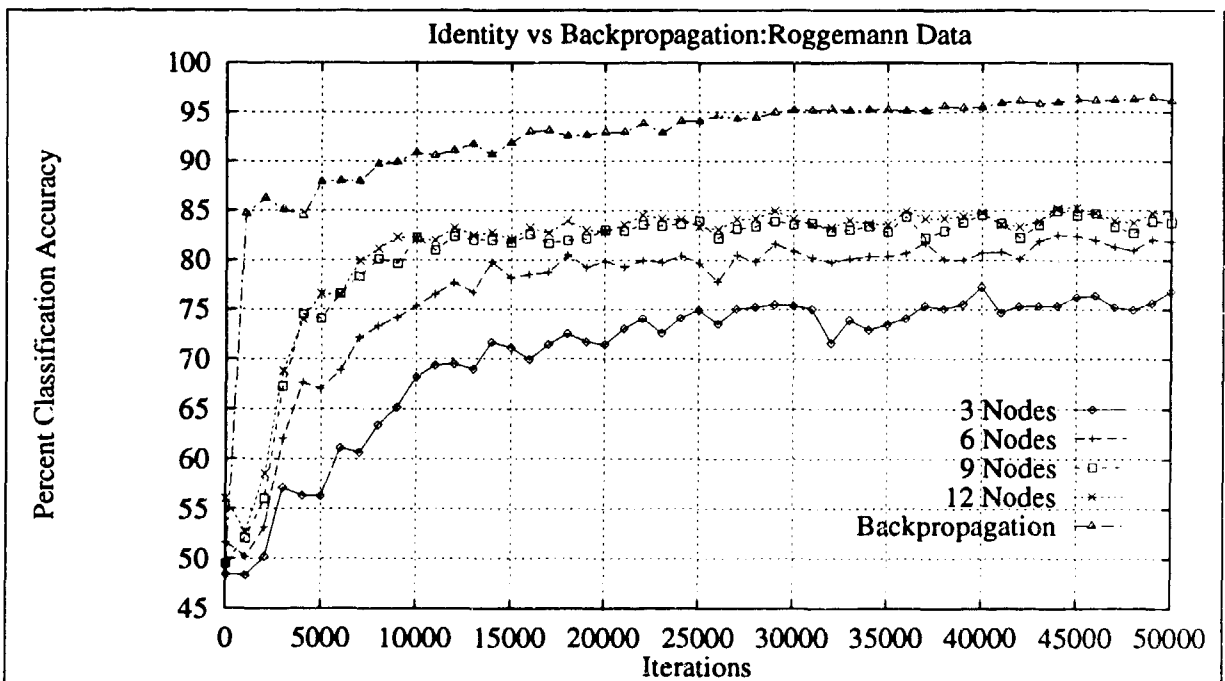
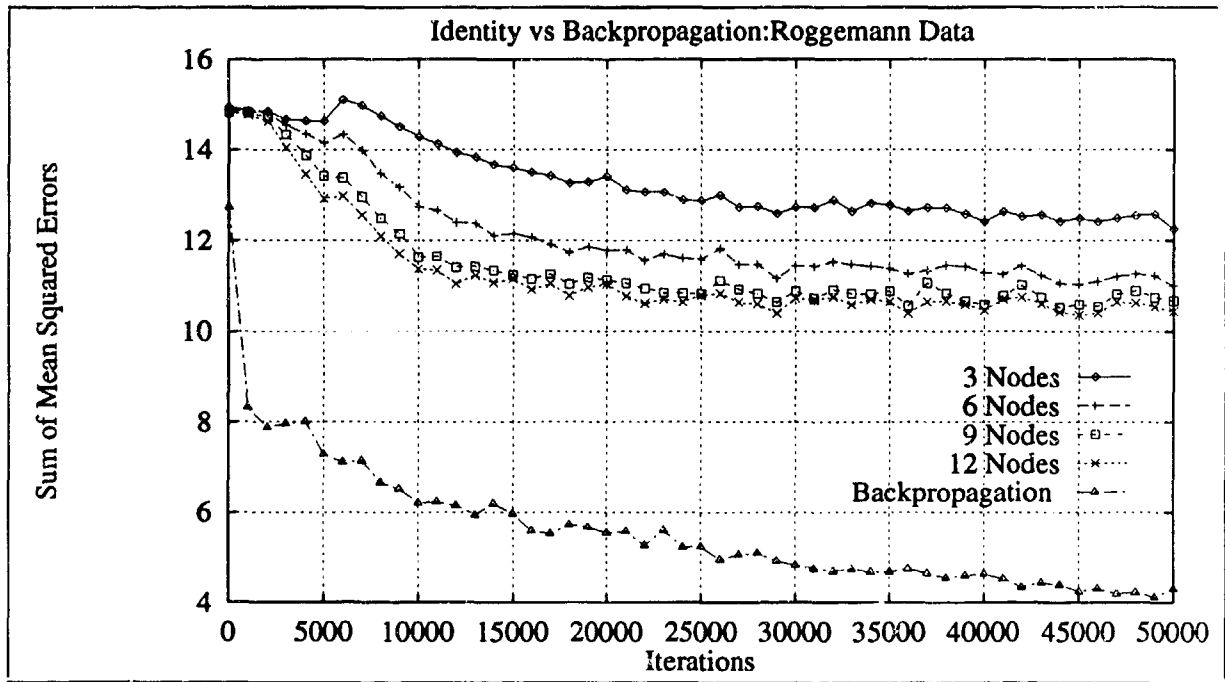


Figure 28. Roggemann Data using Identity Networks. Again the identity networks did not perform as well as the Karhunen-Loève networks where the weights are calculated directly.

4.8 Gram-Schmidt Type Networks

The Karhunen-Loève networks suggest that training is faster if the input features are orthogonal. This section will present the use of a Gram-Schmidt type network, which computes an approximately orthogonal set of weights between the input and the backpropagation network. The output of the Gram-Schmidt layer provides a projection of each input vector onto a weight vector. Propagation of the input to the output of the Gram-Schmidt layer is a vector dot product of the input vector and the Gram-Schmidt weight vector.

The modification to the identity networks is simple. When the weight update is made to the first hidden node, an amount is subtracted from the rest of the nodes' weights to ensure that every weight update is perpendicular to the direction from every other nodes weight vector. Finding the correct amount to subtract is a result of the Gram-Schmidt process. The process requires three steps. First compute the weight updates ΔW_{ij} (momentum matrix) using the identity network. Next orthogonalize and normalize the momentum matrix using the Gram-Schmidt process ((19)) described by Equation 24. Add the orthonormal weight updates to the previous weight vectors, then repeat the process with the weight matrix. To make the computation, let the weight and momentum matrix W be represented as a set of vectors $\beta_1, \beta_2, \beta_3, \dots, \beta_n$. Let the orthogonal vectors associated with β_k the weights matrix and momentum matrix, α_k be computed by

$$\alpha_{m+1} = \beta_{m+1} - \sum_{k=1}^m \frac{\beta_{m+1} \cdot \alpha_k}{\|\alpha_k\|^2} \cdot \alpha_k \quad (24)$$

and $1 \leq m \leq n$. Because the vector computed from 24 are orthogonal but not orthonormal, then each of the individual weight vectors (α_i) must be normalized.

The results are shown in Figures 29 and 30.

Because the Identity network, as presented by Cottrell and Oja, didn't perform as well as expected, a modification was made to the architecture. The modification brought the performance of the Gram-Schmidt network level of performance up to the level of the

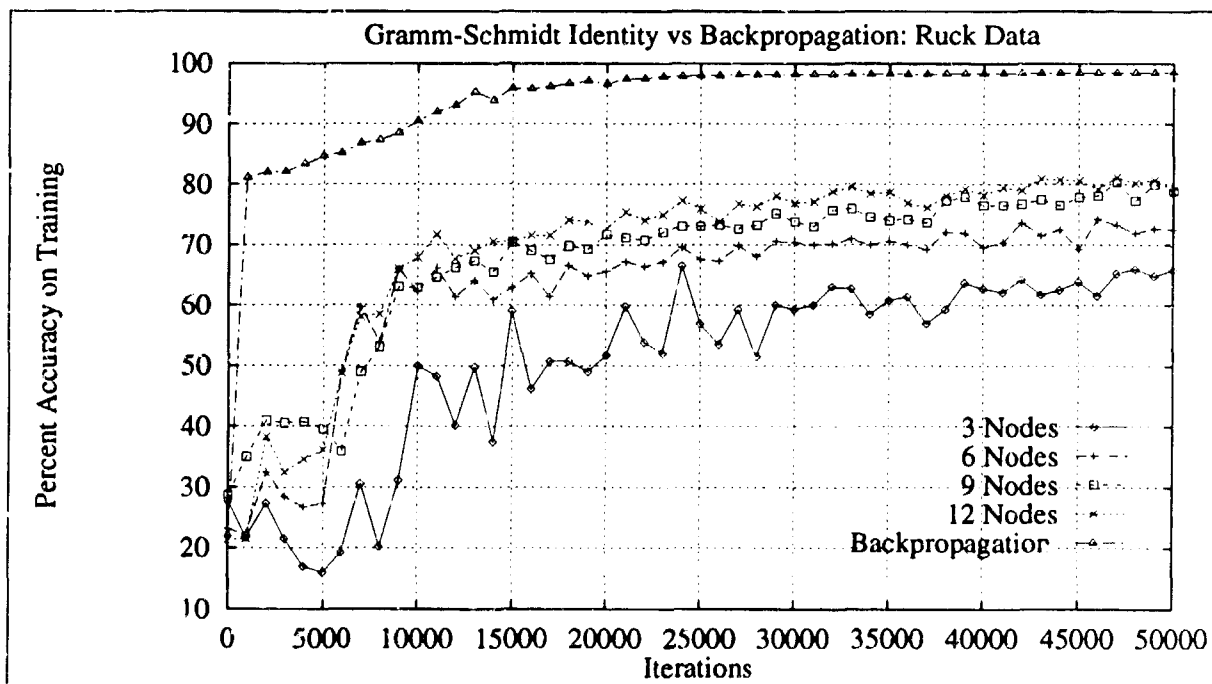
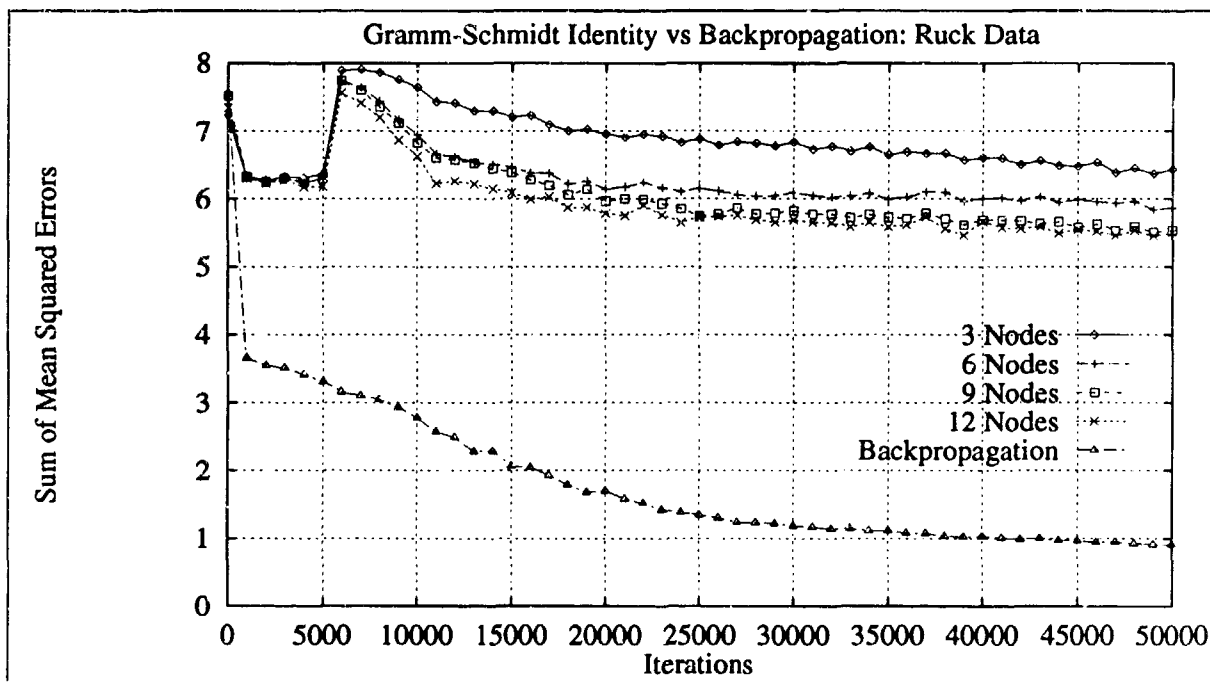


Figure 29. Ruck Data Using Gram-Schmidt Identity Network. The Normalized Gram-Schmidt networks performed slightly better than any of the previous methods.

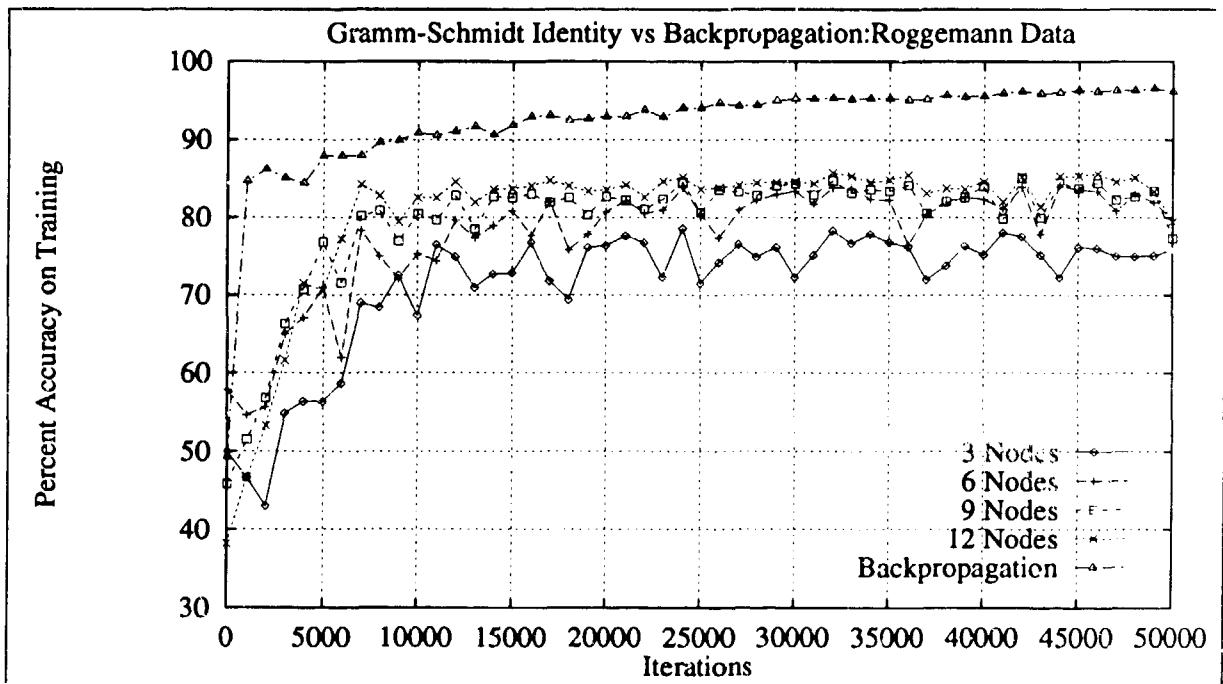
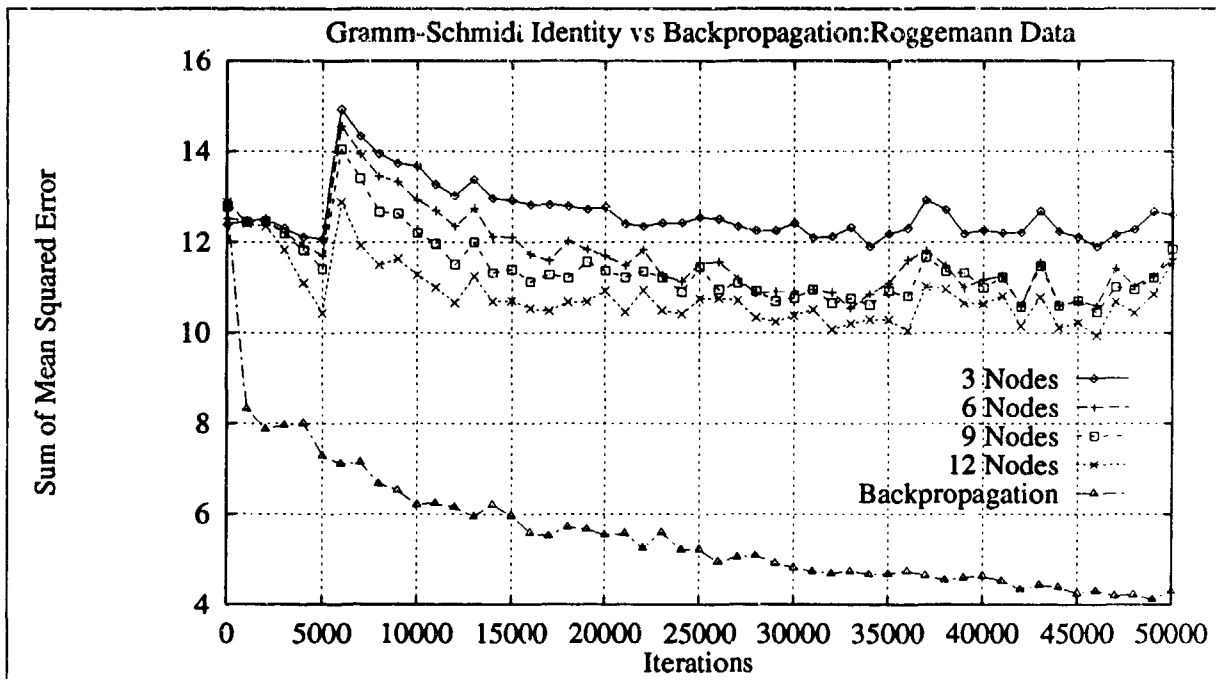


Figure 30. Roggemann Data Using Gram-Schmidt Identity Network.

Comparison of Results				
File	Algorithm	Nodes	Train	Test
Ruck	Backprop	12	100	84.7
	Identity	3	66.2	43.6
	Gram-Schmidt	3	63.8	43.6
	Normalized Gram-Schmidt	3	83.3	55.7
	Identity	6	76.2	43.6
	Gram-Schmidt	6	75.6	43.6
	Normalized Gram-Schmidt	6	93.4	53.7
	Identity	9	78.4	44.1
	Gram-Schmidt	9	78.2	44.1
	Normalized Gram-Schmidt	9	94.5	50.3
	Identity	12	80.9	42.0
	Gram-Schmidt	12	81.4	42.0
	Normalized Gram-Schmidt	12	94.7	47.6
	Roggemann	Backprop	12	100.0
Identity		3	76.7	65.5
Gram-Schmidt		3	75.7	63.2
Normalized Gram-Schmidt		3	86.0	62.9
Identity		6	81.6	68.5
Gram-Schmidt		6	83.3	71.8
Normalized Gram-Schmidt		6	94.3	68.1
Identity		9	83.7	69.9
Gram-Schmidt		9	83.4	67.1
Normalized Gram-Schmidt		9	96.7	66.2
Identity		12	84.7	70.1
Gram-Schmidt		12	85.6	67.1
Normalized Gram-Schmidt		12	97.2	98.6

Table 8. Performance of the Gram-Schmidt network was similar to the Identity network.

Karhunen-Loève networks. The change was to add a processing layer between the identity layer and the backpropagation layer. That layer performed a statistical normalization of the input to the backpropagation network. The results are shown in Figures 31 and 33.

4.9 Conclusions

This chapter has discussed the use of several techniques to reduce the size of the input feature vector. First an empirical metric was presented to determine which input features were most useful in determining classification. The tests showed that the saliency of a node or feature can be determined by comparing the magnitude of the attached weight vectors, instead of an exhaustive search of the weight space.

Next a means to reduce the size of the input space using linear combinations of the input features was developed using eigenvectors and the discrete Karhunen-Loève transform. The Karhunen-Loève transform was implemented in a neural network architecture. Three means were used to determine the best rotation matrix (weights): Karhunen-Loève, Karhunen-Loève mean and Fisher linear discriminants. Each of the three methods held advantages under different conditions. Karhunen-Loève mean worked the best for data sets with inconsistent data. Inconsistent data, data which possibly contains incorrectly labeled exemplars, can increase entries of the covariance matrix. The Karhunen-Loève layer rotates the data in a direction of maximum variance. By reducing the variance contribution of inconsistent exemplars, the Karhunen-Loève mean produced slightly better results, for the data under consideration. Fisher linear discriminant networks did not show enough performance improvement to justify the additional computation expense.

Using an identity network, improvements on the Karhunen-Loève architecture were developed which used ANN learning techniques to set the weights. First attempts to implement these networks did not perform as well as directly calculating the weight matrix using the eigenvalues. The performance was improved by adding an additional layer of processing to statistically normalize the output of the identity layer.

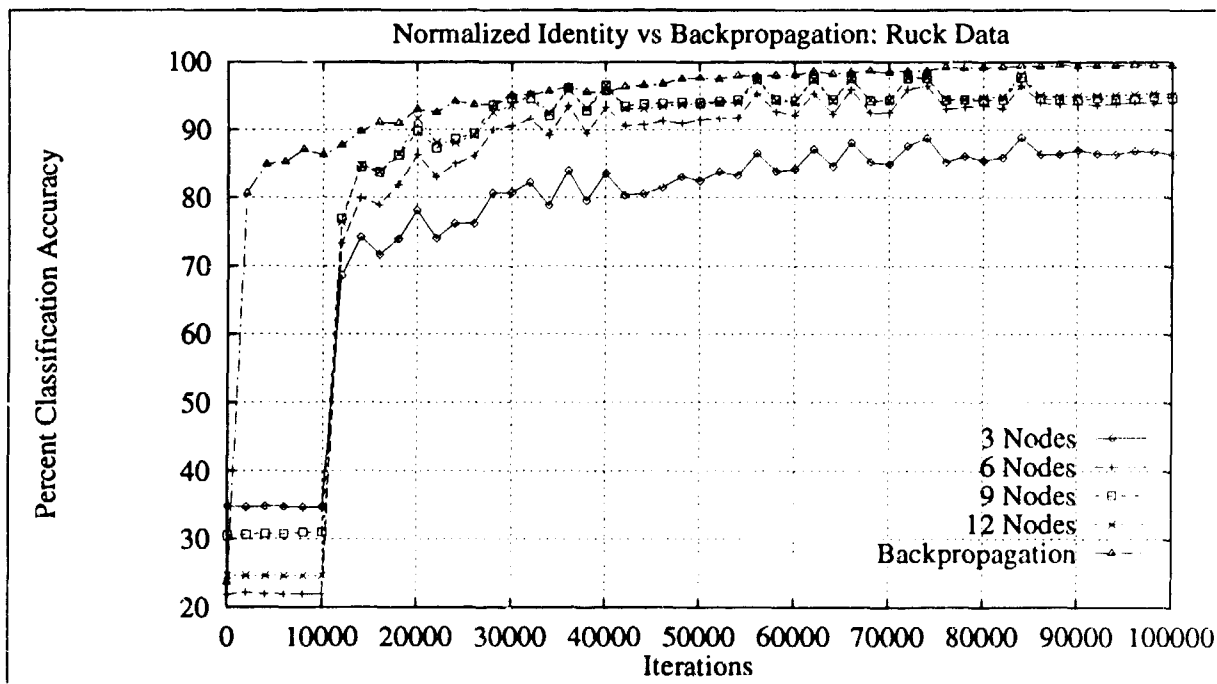
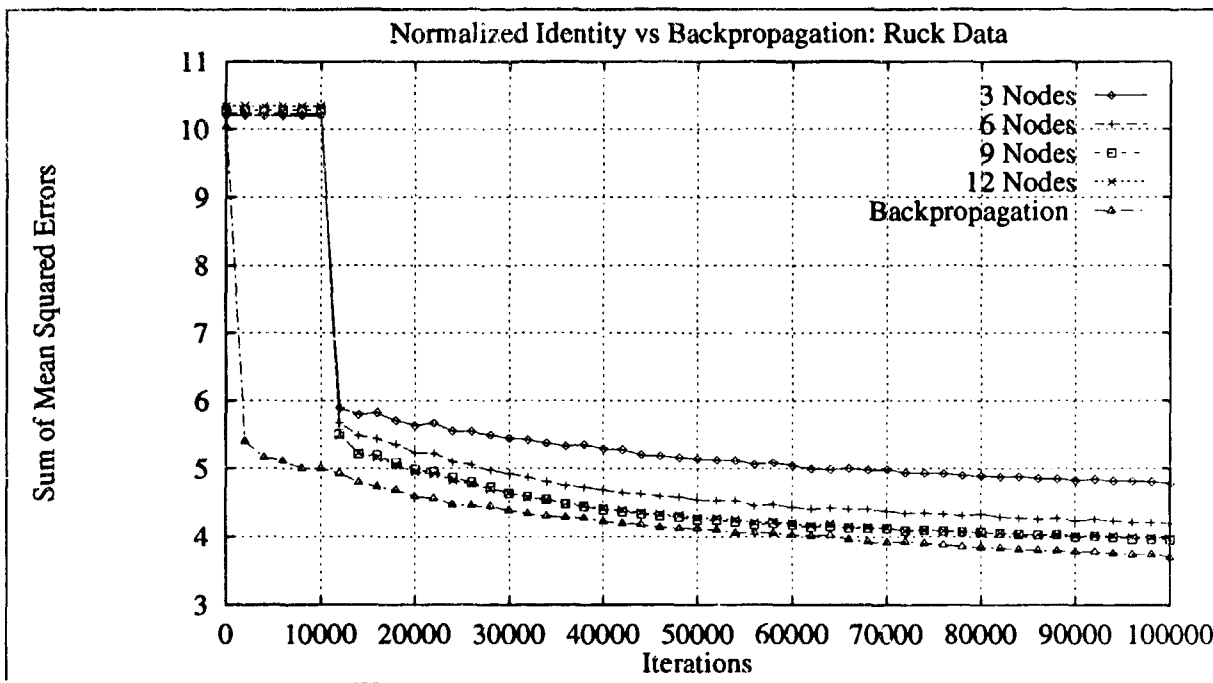


Figure 31. Ruck Data Using Normalized Identity Network. Normalization of the output of the Identity network significantly improves performance of the network.

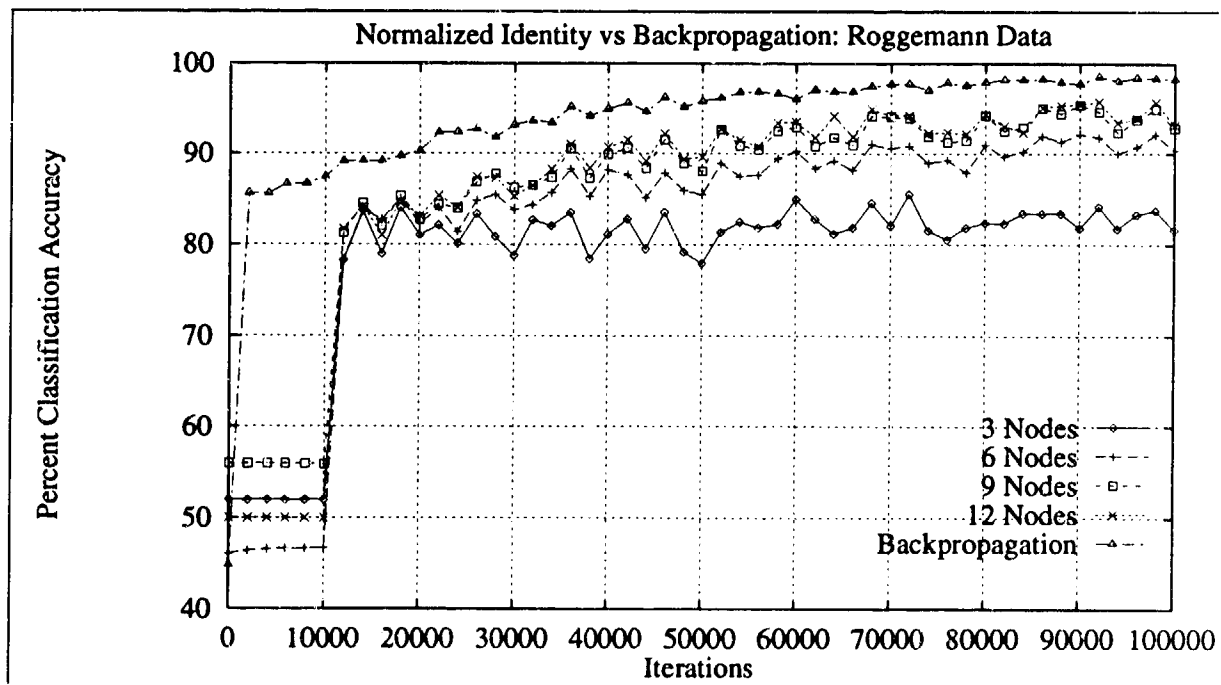
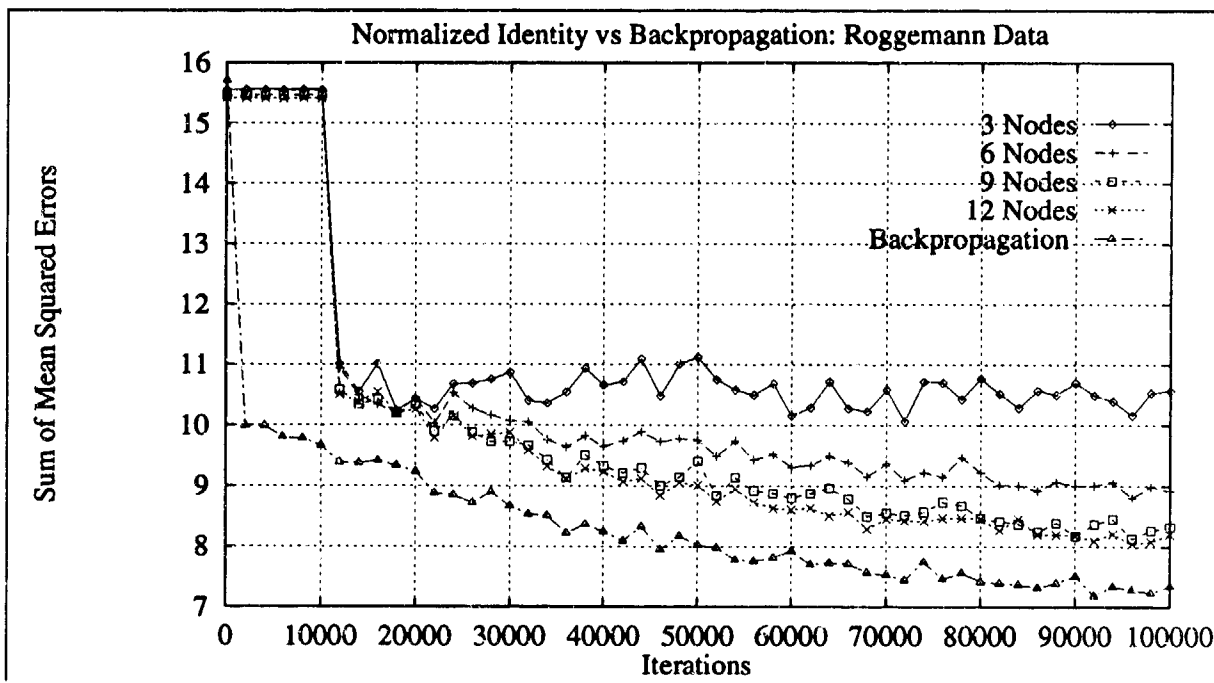


Figure 32. Roggemann Data Using Normalized Identity Network.

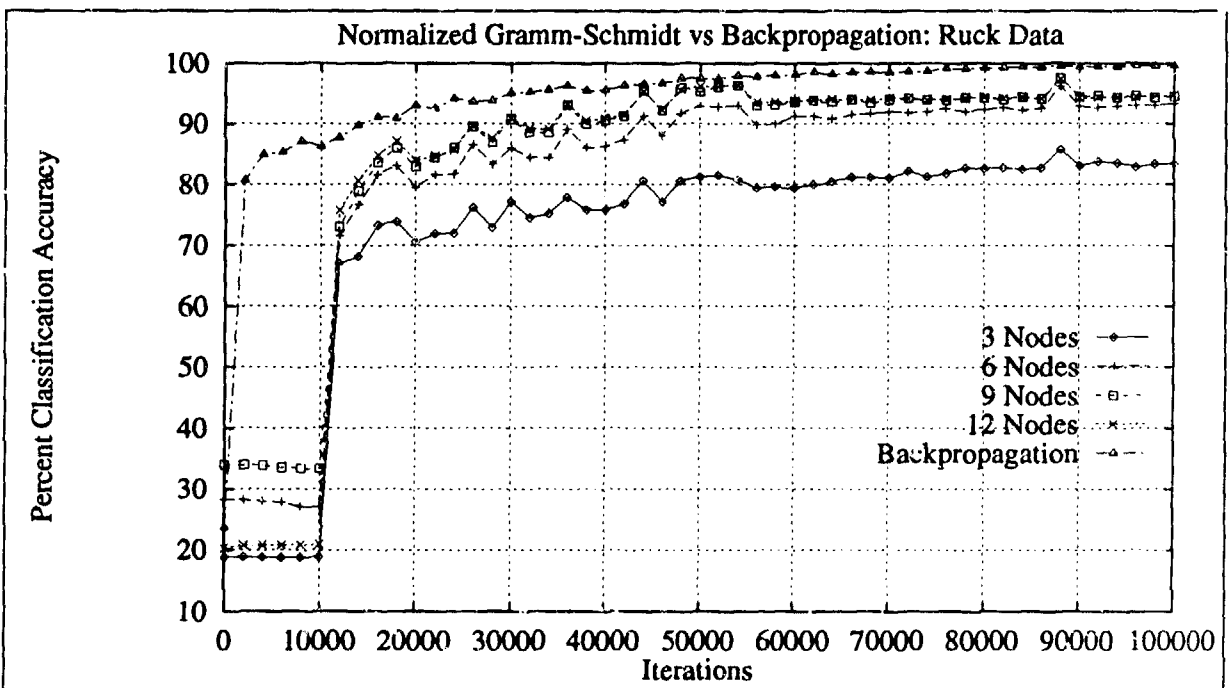
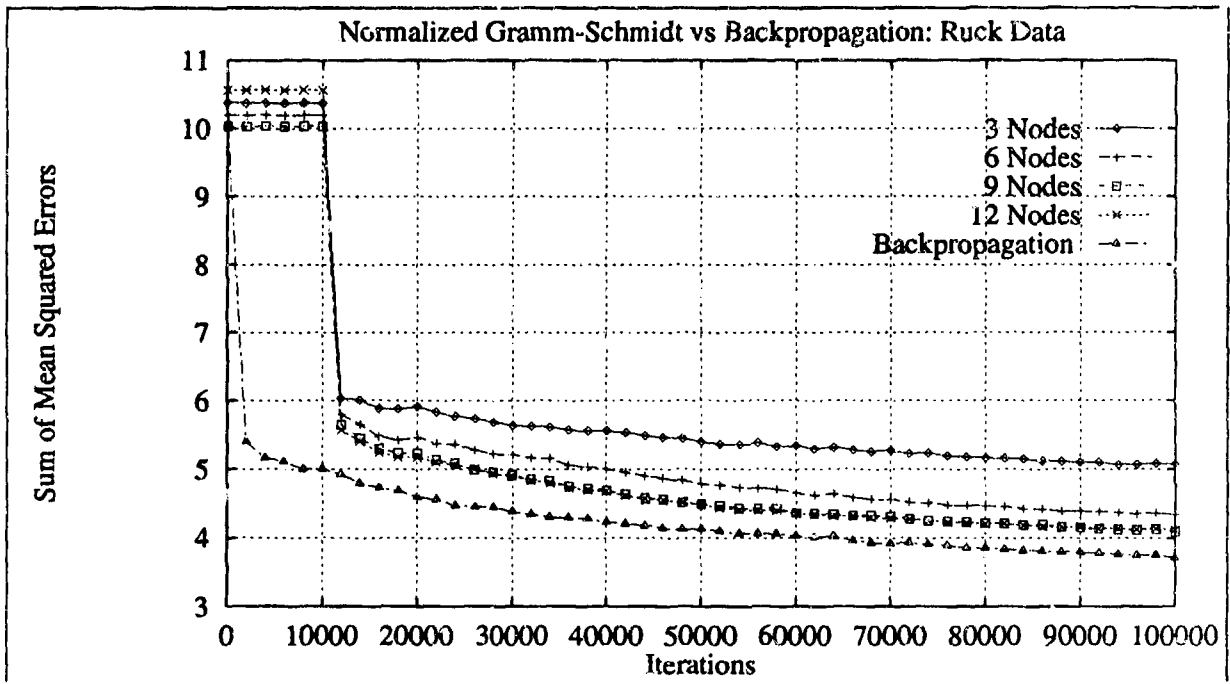


Figure 33. Ruck Data Using Normalized Gram-Schmidt Network.

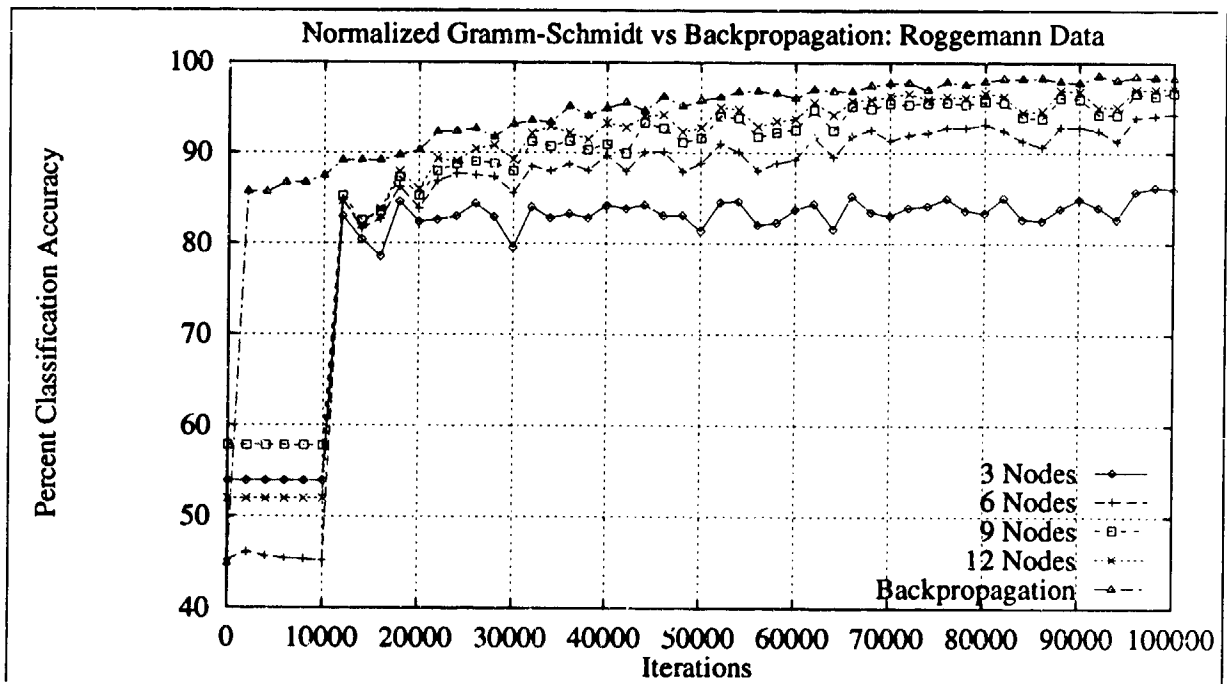
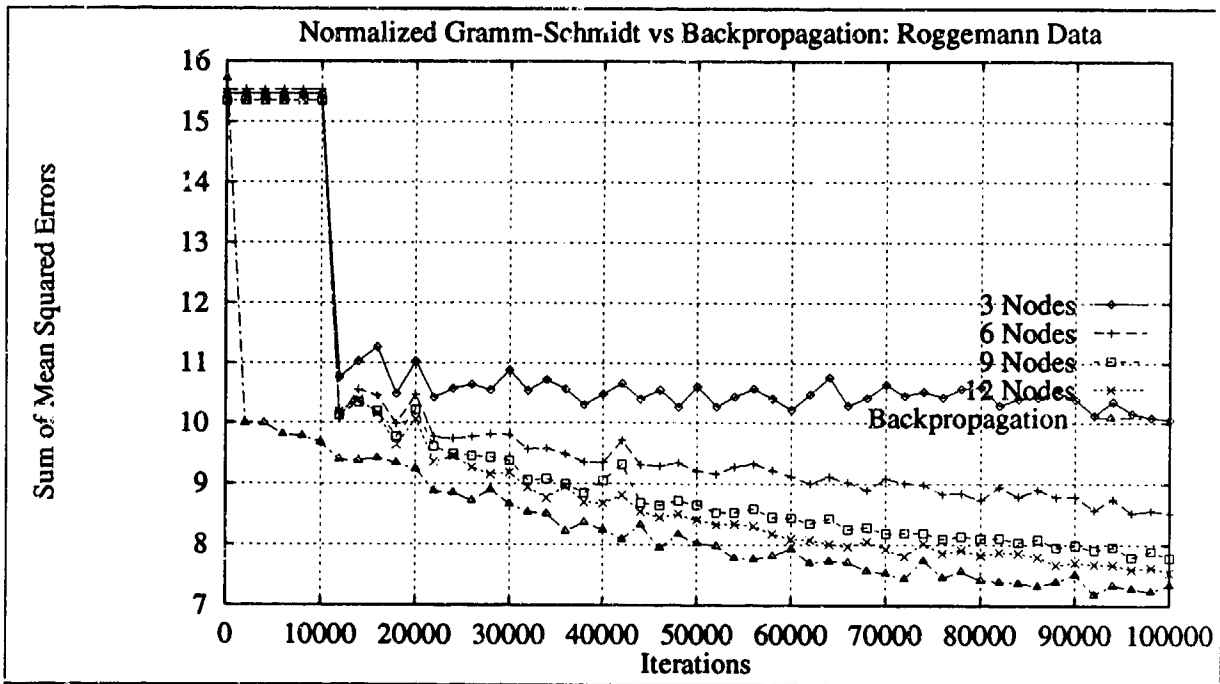


Figure 34. Roggemann Data Using Normalized Gram-Schmidt Network.

The next chapter discusses an implementation of the generalized neural network to perform image segmentation on a set of sequential images for target tracking.

V. Generalized Neural Networks for Image Segmentation and Tracking

5.1 Introduction.

This chapter will discuss use of generalized neural networks for an image segmentation and tracking problem. The tracking problem, finding high-valued fixed targets in a cluttered background, can be solved using a scanning window architecture with a neural network classifier.

The scanning receptive field method is compared with classical segmentation techniques on two types of infrared imagery. The data sets include sequential imagery of high-valued fixed targets like dams, runways and power plants, and multiple targets of tanks, trucks and jeeps in a cluttered background. This architecture is presented as a novel means to perform the first level of hierarchical vision analysis. The scanning window method is compared to conventional techniques for segmenting infrared imagery, such as the histogram concavity methods, and Gabor filter thresholding(34, 2).

To enable neural network analysis for imagery, the NeuralGraphics system was augmented to include imagery input and output. Although the user interface was modified to meet the requirement for building exemplar sets and importing sequential imagery, the underlying software uses the same learning and propagation modules as the NeuralGraphics package described in Appendix A.

This chapter will introduce a classical segmentation problem to illustrate the problems of cluttered data segmentation. Following that is a brief review of previous methods for segmentation. Next, a comparison of techniques will be presented on several problems to show the advantages of each. Finally, the solution to a problem in segmentation and tracking will be given which uses a simulation of the scanning window architecture.

5.2 Image Segmentation.

Segmentation is an intermediate step between detection and classification. The exact scope of the process is not well defined. Providing a complete definition for *segmentation* is difficult as the definition is dependent on the problem's final objective.

Subjective comparison of methods is also difficult for the same reason. Specifying a segmented image as good or bad is related to the type of targets and final objective. In practice, good segmentation is *good enough* when the segmented image can be passed along on to the next stage of the process, usually classification. For that reason, the problem selected to test the segmentation process for this experiment includes a conclusion or measurable objective.

The objective is to maintain an aim point on the target in a reasonable place.

The objective of the segmentation phase is to label each pixel into one of several classes. Consider the images in Figure 35. The segmentations shown are produced by the scanning window method. The method (presented in detail in Section 5.5) partitions each image pixel into one of two classes.

The problems presented in this section require segmented images for target tracking. For tracking and targeting purposes, two classes are sufficient. The two classes for this problem are target and non-target¹. Each image in the figure represents a classic segmentation problem. The background is represented by one type of texture, while the target pattern is represented by another. In the figure, there are three textures shown on the left. The first texture is produced by varying the density of the black dots. The second overlays a region of circles on a background of crosses. The final example uses crosses to represent the background, and circles to represent the targets. The image on the right is the segmented image. None of these textures are easily discriminated by pixel intensity.

The holes and ragged edges of the segmentation demonstrate how many of the regions associated with the target class have characteristics of the background class. These

¹The NeuralGraphics system software limit is 256 separate classes, which is enough for most problems.



Figure 35. Classic Texture Segmentation. The images on the left are images composed of two types of textures. The first was constructed by placing random dots over the image. In one area of the image the density of dots was doubled. The second image overlays a distribution of crosses with a region of circles. The final image has a region of randomly distributed circles surrounded by a distribution of crosses. What makes these images difficult to segment is the fact that all pixels have only one of two values for intensity. Single pixel intensity provides no information to the segmenter. The images were segmented using the techniques described in Section 5.5 and displayed on the right.

holes demonstrate one of the greatest difficulties in segmenting cluttered images, *illusory contours*. The segmentation isn't wrong, but in fact is more correct than our perception. The perceived boundary of the region is an illusory contour. Illusory contours are probably the most difficult problem in cluttered background object segmentation, because they don't really exist in the image intensity information.

Biological image processing systems perform additional computations to associate these regions with the target class. Computing sharp edges for the textures, when no sharp edges exist, is difficult in machine vision systems.²

The targeting method presented here avoids searching for illusory contours. Instead of finding fixed edges, the scanning window method finds regions or clusters with a high density of target pixels. For targeting, an aim point is associated with the centroid of a region, eliminating the need to find fixed edges. Since the segmented image is made up of either zeros or ones, crude blob finders are easily implemented, as presented in Section 5.5.

The following sections will discuss previous efforts at image segmentation using both histogram and Gabor filter methods for gray scale imagery. First, several images from the Night Vision Laboratories are segmented using each of the three techniques mentioned above. Next, imagery collected for the Autonomously Guided Conventional Weapons (AGCW) program is segmented with the histogram method and the scanning window method.

5.3 Segmentation using Self-Organization

Learning paradigms can be divided into two broad categories, tutored and self-organizing. Tutored learning, like backpropagation, assumes that each piece of learning data has a label associated with it. The label is used to calculate the error terms for

²Placing an object boundary across an illusory contour has been the subject of many studies. In particular, some of the Grossberg Boundary Contour System (BCS)(15) work shows promise for filling in illusory contours. Still, these processes are computationally intensive. By limiting the area of interest the scanning window techniques allow conventional boundary filling methods to be implemented over much smaller regions in the scene.

training the weights. Self-organization doesn't use labels for the learning process but assumes that the data contains an underlying structure which the network can discover. The learning procedure attempts to cluster the data into a finite number of classes. The following experiment attempts to cluster different regions of an image into a small number of classes.

In Figure 37 the segmented image shows the result of a correlation of the image with a single point in the image. This technique is the basis for correlation tracking. The point with the highest correlation is used for tracking. The correlation image is a measure of how close (in decision space) every point in the image is to the target window. A threshold was set to show only pixels above specific values. By selecting a point that contains cultural items, the hope was to threshold out regions which contained other cultural items. The segmentation did do that, but a more accurate interpretation might be that the procedure segmented objects with vertical and horizontal edges. The same procedure could be implemented by selecting several examples or templates and establishing the class of a pixel by finding the closest template and assigning the pixel to that class. Kohonen et al (25, 18) demonstrated a self-organization method which allows the network to find its own best templates. This technique was combined with the scanning receptive field method to implement a segmentation algorithm.

Self-organization was tested against the image shown in Figure 36. The objective was to organize the data into seven different types of regions using RBF's. The process starts by selecting 100 arbitrary points in the image and creates composite spectral filters using Fourier and Gabor components. Using a Kohonen like neural network, the image is segmented into the seven best spectral representations of the image at single points.

These methods can be used to classify large regions in the image which share common characteristics such as fields, forests, bodies of water, etc. The object is to identify regions with cultural objects. Cultural items tend to show up in areas with erratic and diverse spectral content.

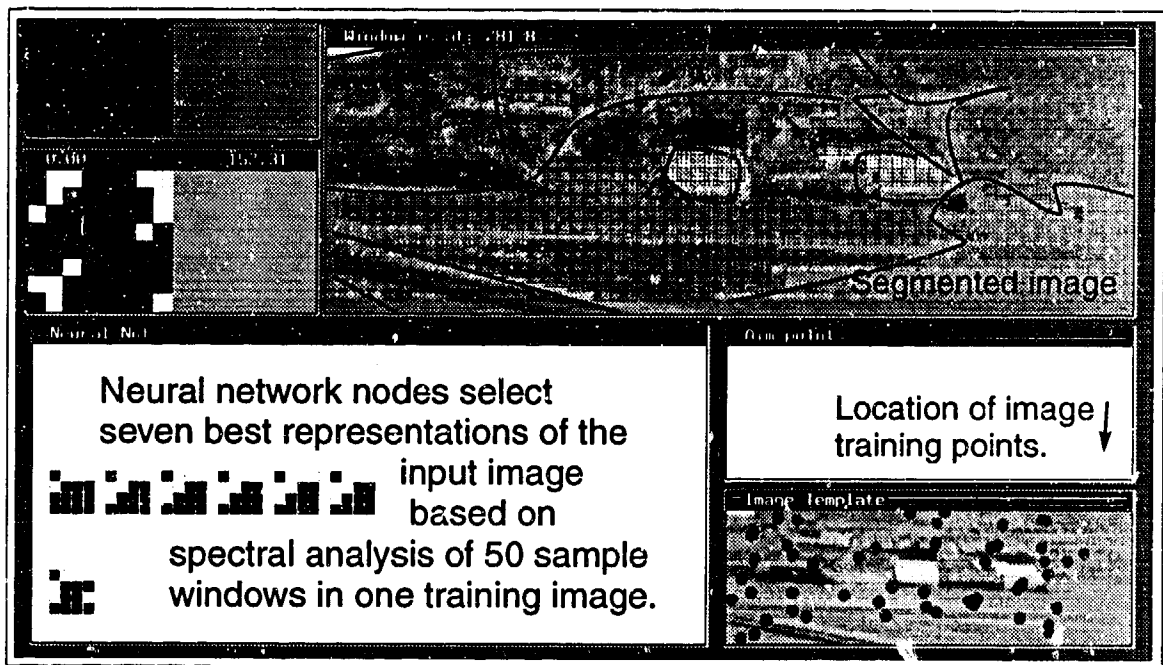


Figure 36. Gross segmentation by Radial Basis Function. The image in the upper right has been segmented into the seven best classes of texture using RBFs. The image in the lower right shows the sample points used to train the image.

5.4 Histogram and Gabor Filter Segmentation.

Most target segmentation and detection methods depend upon heuristic algorithms. Heuristic methods are based on using a set of filters and thresholds. Threshold criterion and filter components are determined by trial and error or a priori knowledge of the data set. Two examples of heuristic segmentation techniques include histogram concavity analysis and Gabor filter thresholding.

Figure 38 illustrates the histogram concavity method. By placing the classification threshold between the two peaks of the histogram, each pixel in the image can be classified as being above the threshold or below the threshold. Being above or below the threshold discriminates between class one or class two. Figure 38 shows the histogram used to segment the image in Figure 44 which illustrates intensity thresholding for an image of a runway.

Previous infrared segmentation(34, 2) efforts have used thresholding and filtering

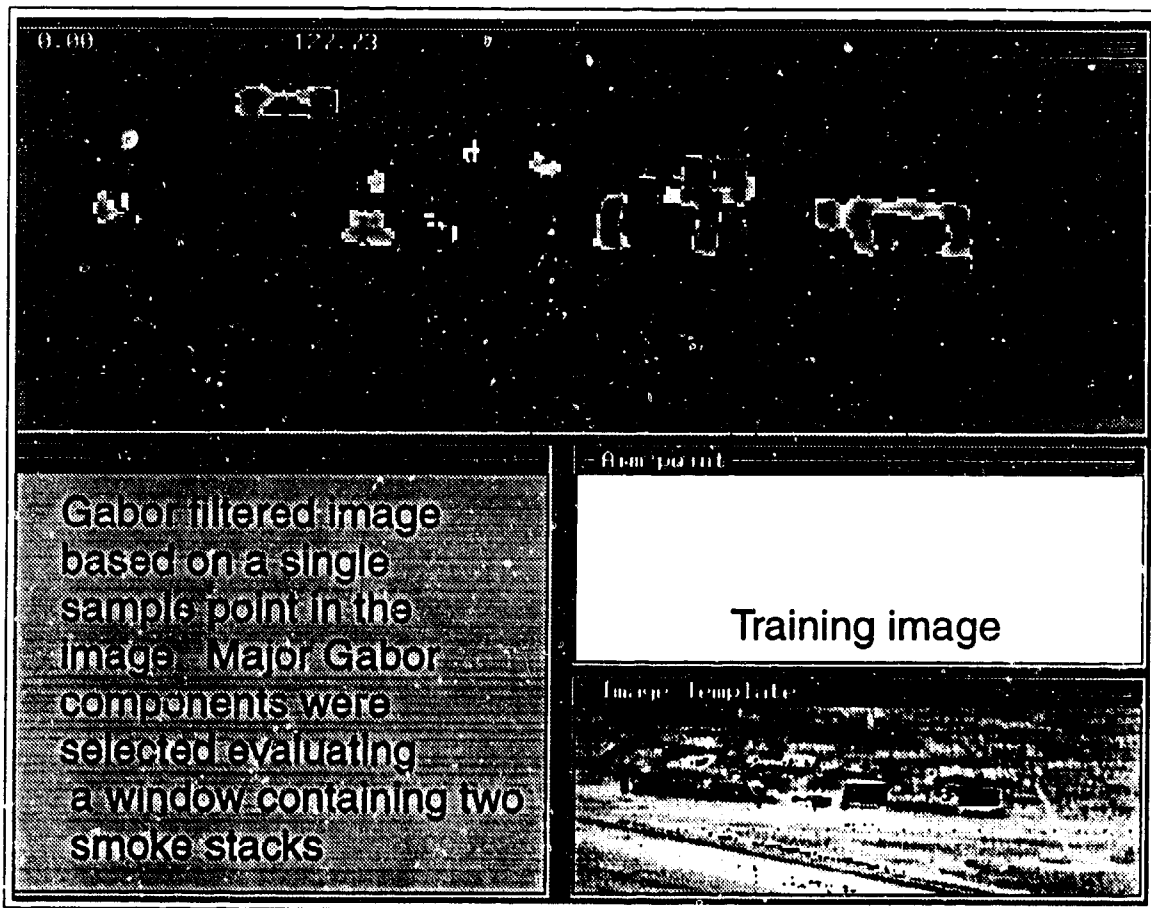


Figure 37. Object segmentation using Gabor filter correlation. A single window is correlated with the image. The filtered image is thresholded and displayed in the top window. The image in the lower right is the original.

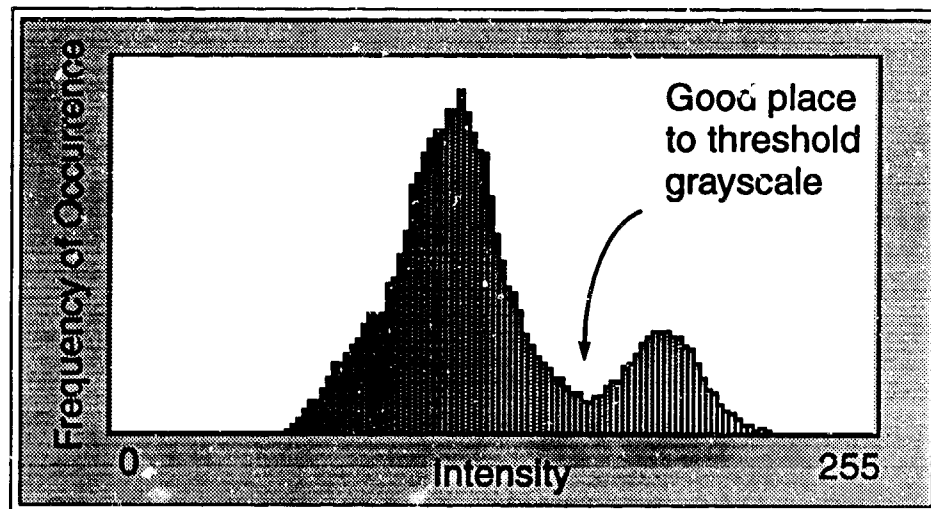


Figure 38. Histogram Concavity Method.

techniques to partition pixels into the two classes. Roggemann (34) was able to segment mobile targets from infrared imagery by examining the histogram of the pixel data. He suggests that the histogram for the class of targets under consideration is bi-modal. Consider the histogram of the image in Figure 39, the histogram window shows a weakly bi-modal distribution. To segment the image, the two intensity peaks are found and a threshold is set between them. The three segmented images indicate three possible threshold levels between the two peaks. Notice that the total target energy is small compared to the total background energy. Each level is a legitimate choice for the threshold. The first uses the left inside peak. The next centers the threshold between the peaks. The final image uses the point right inside of the peak.

These images illustrate an important point about thresholding. *Thresholding is an intelligent process.* It requires a decision based on what makes the final results look the best. You can't tell what will look the best until after it has been tried.

Segmenting an image using Gabor filtering is similar to the scanning window technique. The image is filtered with several spectral filters. The user must select the filters. The output of the filters are summed and a threshold is applied. Figure 40 illustrates how Gabor filters could be implemented as a scanning window architecture. Ayer used FFT

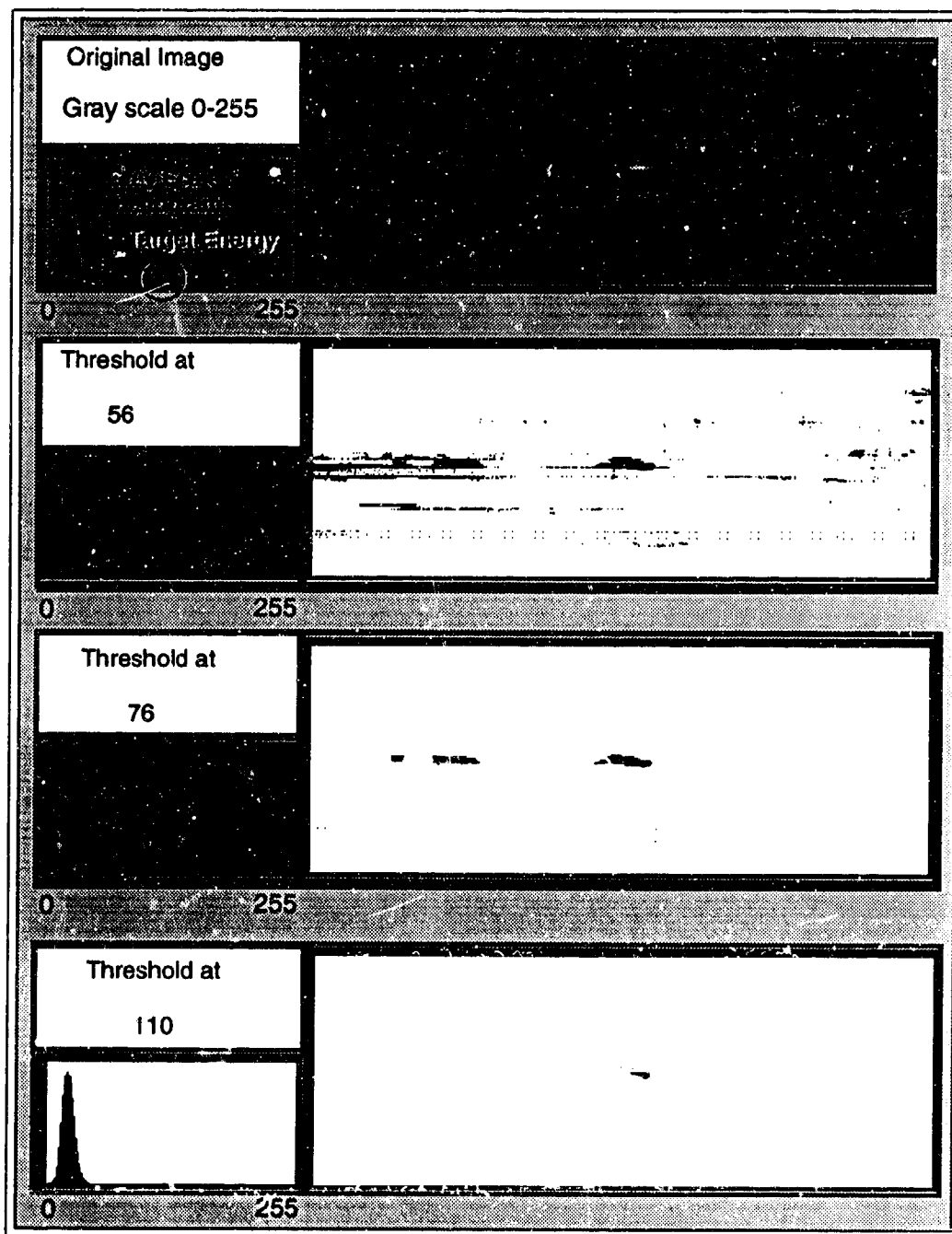


Figure 39. Histogram Concavity Segmentation. Segmentation by histogram is an adaptive thresholding process. The figure shows the three thresholds used to segment the image. Setting the threshold at 56 was selected by finding the first positive increase moving across the histogram right to left between the two energy peaks. Setting the threshold at 110 was determined by finding the first positive increase moving left to right. A threshold of 76 was determined by splitting the difference between the two peaks.

convolutions to construct the output image, which is computationally equivalent(2).

Ayer was able to segment the targets from the background using a tailored Gabor filter. The components of the filter were determined by trial and error. After filtering, the resulting image was put through a threshold filter to separate between the target and non-target. The threshold was selected by visual inspection.

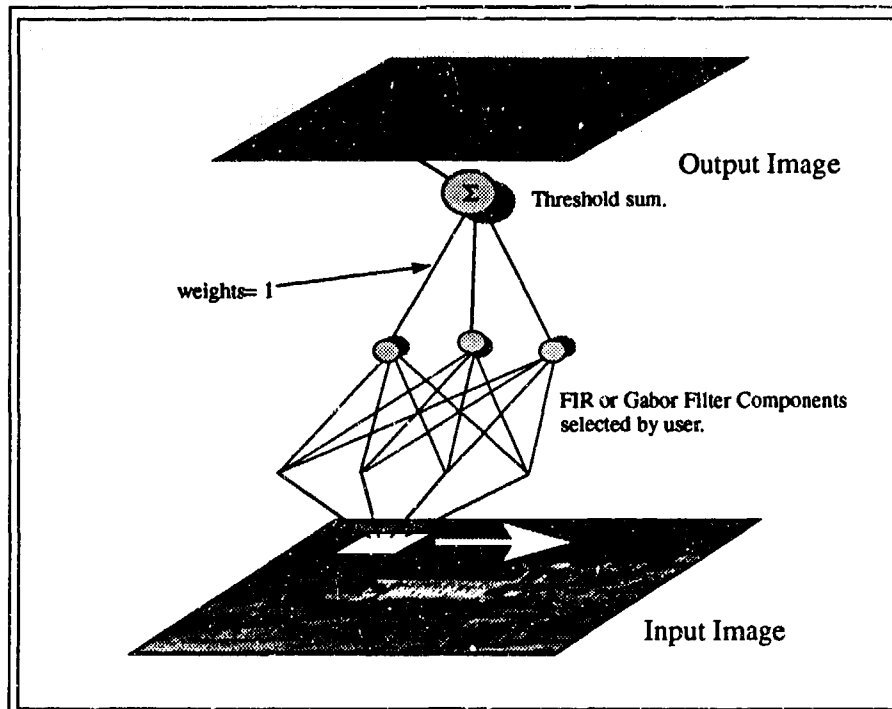


Figure 40. Gabor Filter Segmenting Using a Scanning Window.

Roggermann's result is significant because the method is not heuristic. The drawback is that the method is limited to a narrow class of images: those with bi-modal histograms. The Ayer method doesn't require a bi-modal intensity distribution and should be able to segment cluttered images³ where the image target pixels demonstrate uniform texture over the target regions. Unfortunately, finding the correct thresholds and filter spectral components must be heuristic without some type of self-learning system. Adding a neural

³Ayer uses the same data set as Roggermann.

network front end allows the thresholds and filter components to be determined from the training data.

These algorithms are limited by characteristics of the image, in particular the disjointness of the region to be learned. Neural network segmentation may overcome some of these problems, using a scanning window architecture.

The scanning window approach goes beyond the previous methods by allowing the system to select the appropriate filters and threshold values based on the nature of the training samples. The training samples are selected by the system operator. The operator builds a training set based on representative samples (windows) of the various classes. The samples are processed through a bank of spectral filters. The output is fed to a neural network model and trained to respond to the classes designated by the training set. The system is in fact building custom textures(21) to distinguish between textures based on spectral correlation. Also classification is made differentiating between combinations of textures.

5.5 The Scanning Window Architecture.

The scanning window architecture for image segmentation is based on the fact that classification cannot be determined from a single pixel, but is based on the pixels in the *local* neighborhood around the pixel in question. Histogram and threshold/filter techniques usually compare each pixel to every other pixel in the image.

Two sets of data are considered in this section: mobile tactical targets taken by the Night Vision Laboratories(NVL) (34) and sequential target approach imagery from the Autonomously Guided Conventional Weapons Program(AGCW). For the NVL laboratory data, the actual contours are important, as the next process will be to extract features. For the AGCW data the silhouette is less important than the centroid. For that reason exact distinction between detection, segmentation and classification is sometimes fuzzy.

Pixels for purposes of target identification are either background or target. Further classification, (for example, is the target a tank, truck or jeep) need only consider pixels

classified as part of the target.

The NVL data are a set of single images of tanks, trucks, jeeps and target boards against a slightly cluttered background. The AGCW data are sequential images of high-value fixed tactical targets. The imagery was collected during simulated bombing runs against high-valued fixed targets. Examples of these targets include bridges, dams, airport runways, and power stations, etc.

Using high-value fixed targets reduces the complexity of the problem a little, because (usually) only one target exists in the field of view. The solution to the tracking problem lies in following the centroid of the target pixels.

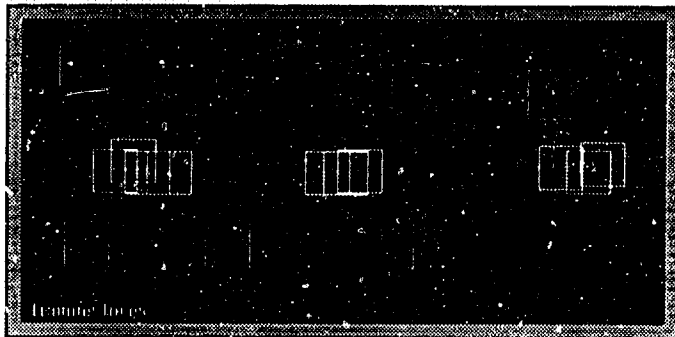
5.5.1 Training the Neural Network. Figure 41 illustrates the scanning window segmentation method. The method is based on the technique of comparing the characteristic of one class of objects (targets) with another (background).

The first step in the process is to select representative windows from an image (or set of images) together with a defined class assignment. A mouse device is used to select points in the image. As the operator selects representative samples, the designated class is stored with the sample.

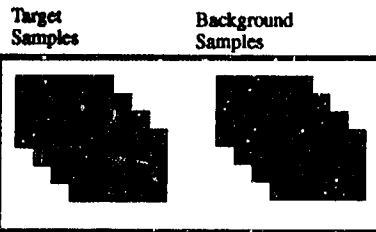
The next steps may be considered data preprocessing but as shown in Figure 41 can be implemented in a neural network architecture. The neural network acts like a blackbox, which when presented with a window (of pixels) centered at a specific pixel, assigns an output class to that pixel.

The preprocessing begins with a spectral decomposition of the input window. To preserve shift invariance, only the magnitude of the spectral component is used in the feature vector. Also, since the feature vector is to be propagated through a neural network, each component should have approximately the same dynamic range. For that reason, the feature vectors were normalized across each spectral component. This adjustment, called statistical normalization, prevents one single component from dominating the training. Training on typical images tends to be dominated by the D.C. intensity. For segmenting

Scanning Window Architecture



Step One: Collect window samples for training.



Class Information for Training Neural Net

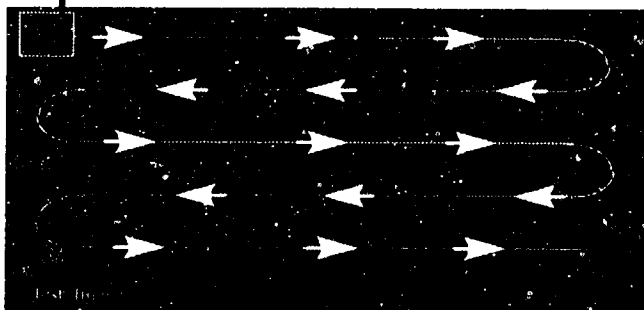
Step Two: Process sample windows for spectral components



Spectral Transform
Replace pixel data with FFT or Gabor Magnitude Coefficients

Statistical Transform
Replace each Spectral component with a statistical Z-score equivalent

New Feature Vectors



Neural Network

Max Picker

Step Four: Systematically move roving window one pixel at a time. Process pixel information through the neural net and estimate class: Target or Background



Figure 41. Training in the Scanning Window Architecture.

images, the D.C. value is certainly important, but the D.C. magnitude can be several orders of magnitude above the next highest components. Statistical normalization allows each spectral component to affect the weight updates equally. After extracting statistically normalized magnitude components for each exemplar window, the network is ready for training.

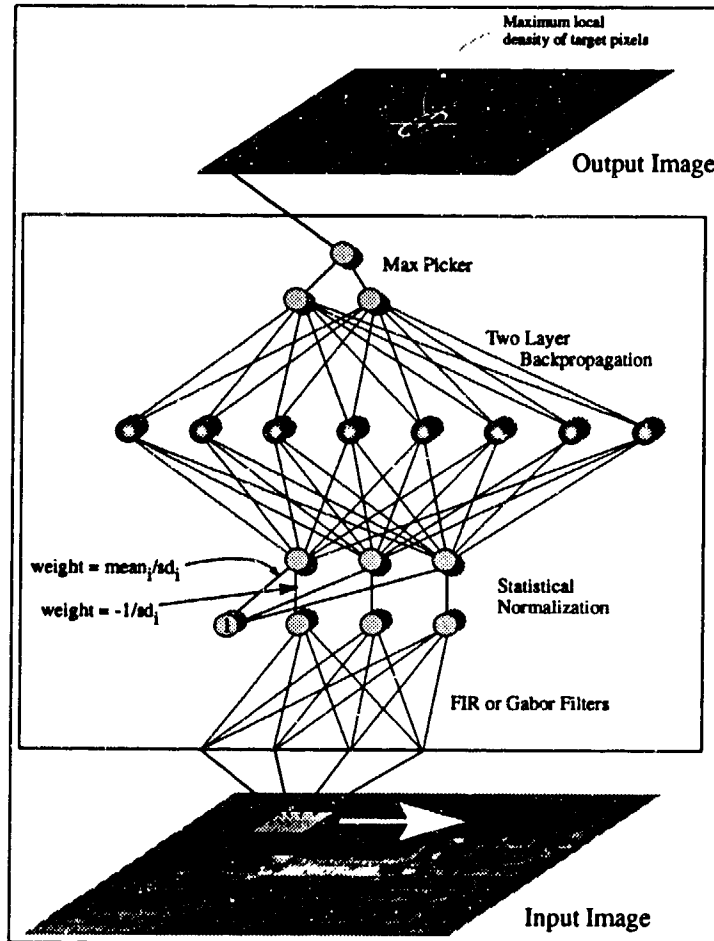


Figure 42. Scanning Window Architecture Image Propagation.

Using the training samples, the neural network is trained using the specified class associated with each training vector. A number of network methods were tried, all with about equal success. The scanning receptor field was very robust to variation in the network architecture. For each image approximately 200 training samples were selected. A net

with 15 nodes in the hidden layer was used and the training algorithm was backpropagation with momentum.

The net was trained until between 98 and 100 percent accuracy was obtained on the training data. This usually took about 10,000 training iterations. One iteration means to present one exemplar to the input and backpropagation of the error through the network. The results are shown in Figures 47 through 49.

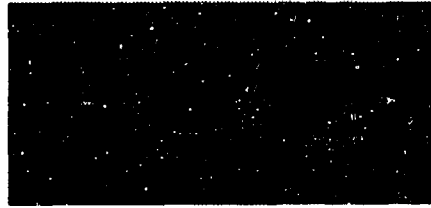
The flexibility of the method comes from the neural networks ability to naturally overcome what Valiant(49) calls the disjoint learning regions problem. Disjoint learning regions are defined to occur when two distinctly different textures in an image represent the same class of object. Consider an image with a tri-modal histogram, if one class were represented by the two outside peaks, with another class represented by the center peak, the classification would always be at least half wrong, using the histogram concavity method. Using an intensity based classification scheme, the scanning window with a neural network classifier would estimate a threshold for intensity, then fix a discriminant function between the estimates.

The layers perform preprocessing, output function mapping, and class estimation. The two preprocessing layers perform a spectral decomposition, and a statistical adjustment. The remainder of the architecture consists of a traditional neural network.

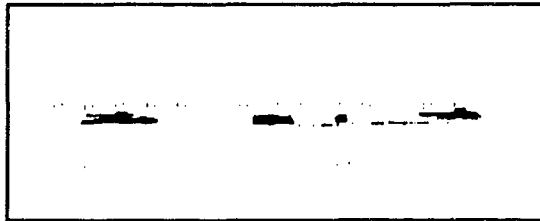
5.5.2 Target Detection and Segmentation. Figure 43 shows several mobile targets in a slightly cluttered background. The imagery is taken from an infrared sensor. Because of the engine heat, they appear hotter than their background; so segmentation based on histogram techniques alone is straightforward. Unfortunately, the second intensity peak in the histogram provides a broad choice for the intensity threshold level.

To compare histogram concavity, Gabor segmentation and the scanning window method, the same image was segmented three times. In Figure 43 the first image segmented with histogram concavity would probably be considered the best. The image was segmented visually to give the best presentation. The last image in Figure 39

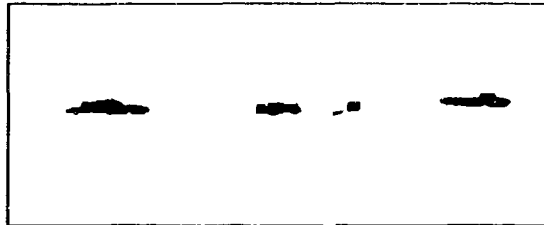
**Night Vision Lab
Image Set**



Roggemann Histogram: Method



Ayer Gabor Filter and Threshold Method



Scanning Window Neural Network Segmentation

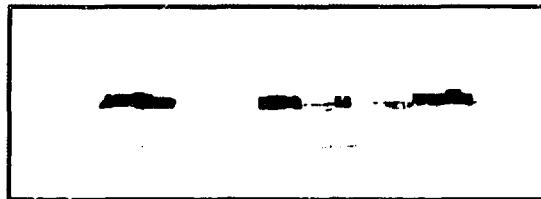


Figure 43. Comparison of Methods: Night Vision Lab Imagery.

could also be considered a valid segmentation. The next segmentation, generated by Ayer with a Gabor filter, was also pretty good. The final segmentation was generated with the scanning window. The results were still pretty good although the boundaries were not as clearly defined. However, the scanning window method has an important advantage over the histogram and Gabor filter methods. The scanning window segmentor learned from the data, and didn't require intervention to select a threshold or which Gabor filters to use.

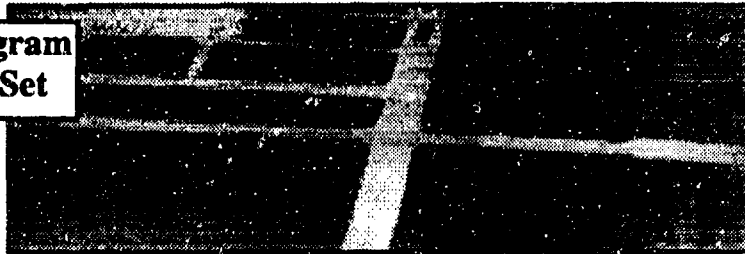
The object of the second test is to find the runway in a series of images approaching an airport. The process begins by training a neural network using examples of the runway, and the surrounding areas. The net is trained with 300 window samples of the image, 150 from each class. After training is complete the image is processed window by window on other images in the sequence. The result is shown in Figure 44. Notice that the histogram segmentation method is not as clean as the scanning window method. In the lower left hand corner of the image, a section of grass had about the same gray scale intensity as the runway and was misinterpreted. This occurred even though the histogram of the image was clearly bi-modal.

Images with purely bi-modal histograms usually segment well using histogram based techniques. However, segmentation performance for the histogram method is not as flexible as the scanning window neural network method. The reason is that the neural network takes advantage of the histogram concavity information; it's contained in the average intensity information presented to the neural network. In addition to the intensity information, the neural network incorporates other information as well.

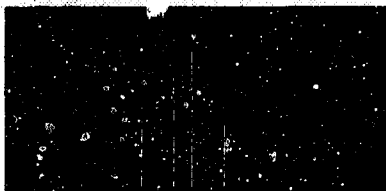
The next example demonstrates what happens when there is no clear dip in the histogram curve to indicate how to threshold the image. The threshold shown in Figure 45 was generated by inspection to produce the best results and ensure a fair comparison.

Images with a uni-modal histogram cannot be segmented easily with histogram concavity techniques. The scanning window method is more flexible in a number of ways. Not only can it pick out regions with similar intensity, it can focus in on other types of textures as well. In Figure 46 the network is trained to recognize edges. The architecture

**AGCW Program
Image Data Set**



Simple Case



The histogram is clearly bimodal. With threshold set between the peaks, the segmented image is not as accurate as scanning window method.

Histogram Method



Scanning Window Neural Network Segmentation

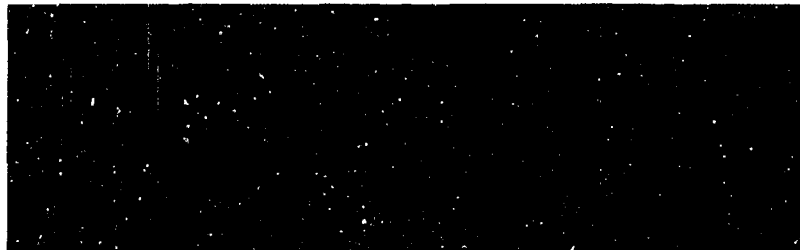
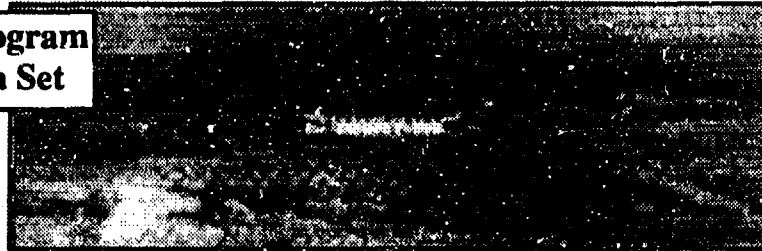


Figure 44. Comparison of Methods: Bi-modal Histogram.

**AGCW Program
Image Data Set**



Cluttered Data



The histogram has only one peak, so machine estimation of the threshold would be difficult. Even with optimum threshold, by visual inspection, the results are poor.

Histogram Method



Scanning Window Neural Network Segmentation

Here, the dam contains the highest density of target pixels.

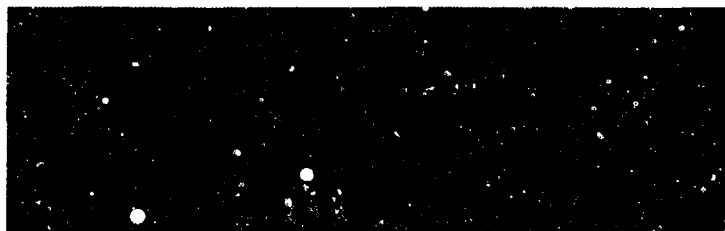


Figure 45. Comparison of Methods: Cluttered Data with Uni-modal Histogram.

suggested by the scanning window approach is versatile enough to segment on a variety of textures as well as intensity.

5.6 High Value Fixed Target Tracking using Neural Networks

The final section will discuss an implementation of the system to track a fixed target from a mobile platform. A series of images was taken on an aerial approach to a high-value fixed target. Figures 47 and 48 show representative images from a tracking sequence. The sequence is trained by selecting 200 randomly spaced windows from the first image. Two hundred samples of the target were taken as well and trained on a two layer backpropagation network with spectral and statistical preprocessing. The network is used to process the sequences shown in Figures 47 through 49. The tracking results are shown for several samples from each approach. The figures show four of an eight image sequence. The network was trained only on the first image.

The tracking points are selected by passing a scanning window over the segmented image. The program calculates the density of target pixels in the window, and the geometric mean within the window. Forty by forty (pixel) windows were scanned across the image (360 by 120 pixels) ten pixels at a time. This results in 98 windows with corresponding densities and target coordinates. The targets were selected as the four windows with the highest density of target pixels.

The first sensor approach sequence contains eight images, starting with the dam being barely visible. In the final image, the dam is shown with four targets indicating the four areas with the highest density of target pixels. The second sensor approach is similar. The third sensor approach is more difficult, as much of the clutter resembled the dam at the window size chosen.

Figures 50 and 51 show a much longer sequence of images. The complete sequence is made up of 439 images. The target is the power plant near the front right of the building. The sequence is trained on two images, one at the beginning and one at a midpoint. Because the building behind the power plant is initially easier to detect, it is used as the

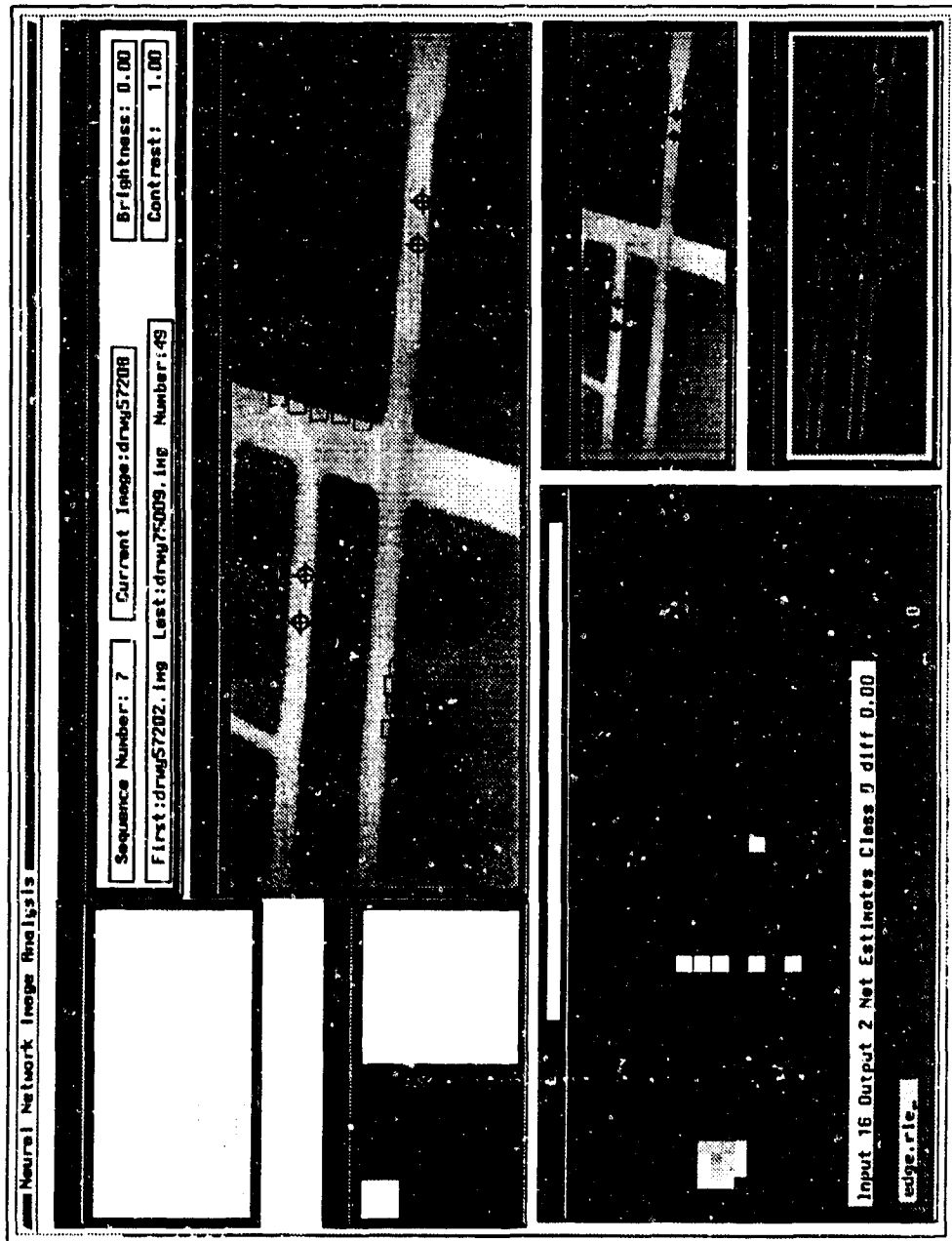


Figure 46. Segmenting Edges Using the Scanning Window.

initial aim point. After the power plant comes into view, the focus of the target is changed to the other building.

5.7 Conclusions

The goal of this experiment was to design a targeting system for an autonomously guided missile system. That objective was achieved. The system requires little technical sophistication for mission planning. With only two or three images of the target along the approach path, the system is able to lock-on the target. The scanning window architecture offers promise for image segmentation and tracking in machine vision systems. Although the system was implemented only in simulation, real data is used through-out. The approach is modeled to take advantage of specialized parallel processing integrated circuits. The architecture described above could be implemented using neural network integrated circuits such as INTEL's ETANN device for real time computation of what has been only simulated here.

The scanning window method is an improvement over previous methods for several reasons. First, the neural network segmentation is based on multiple criteria. The neural network combines information from a number of correlation planes to make a discrimination. Because the neural network combines disjoint learning regions, segmentation can be based on combinations of features. For example, dark and light regions can be combined into a single classification with medium regions in another. In addition, higher order harmonic content can be included in the criteria. High second harmonic values from the scanning window filters are strong indicators of edges. This type of filtering allows a segmentation to mimic heuristic rule-based segmentation, without the required analysis. For example, segmentation can be set to pick out edges of a specific intensity and regions of a specific intensity range.

Most important is the fact that the process is self-tutoring. The operator need only select representative windows of each class. The neural network builds up the rules which define the segmentation scheme. There are few parameters which need to be set by the

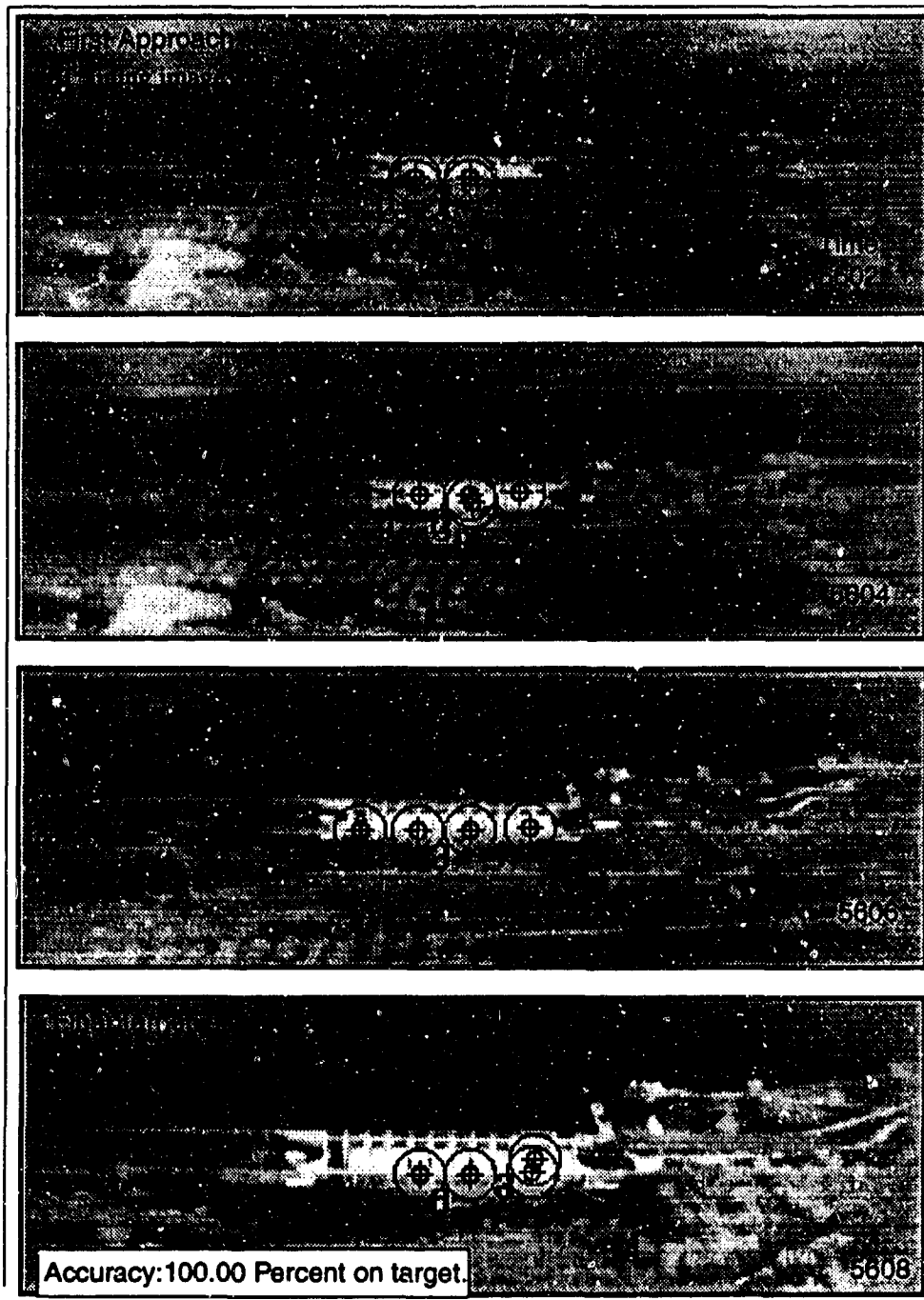


Figure 47. Approach on a Hydro-Electric Power Plant(1). The tracker is trained on the first image, using texture exemplars selected by the user. The figure above shows the subsequent images and the selected target points. Four images out of a tracking sequence of eight are shown.

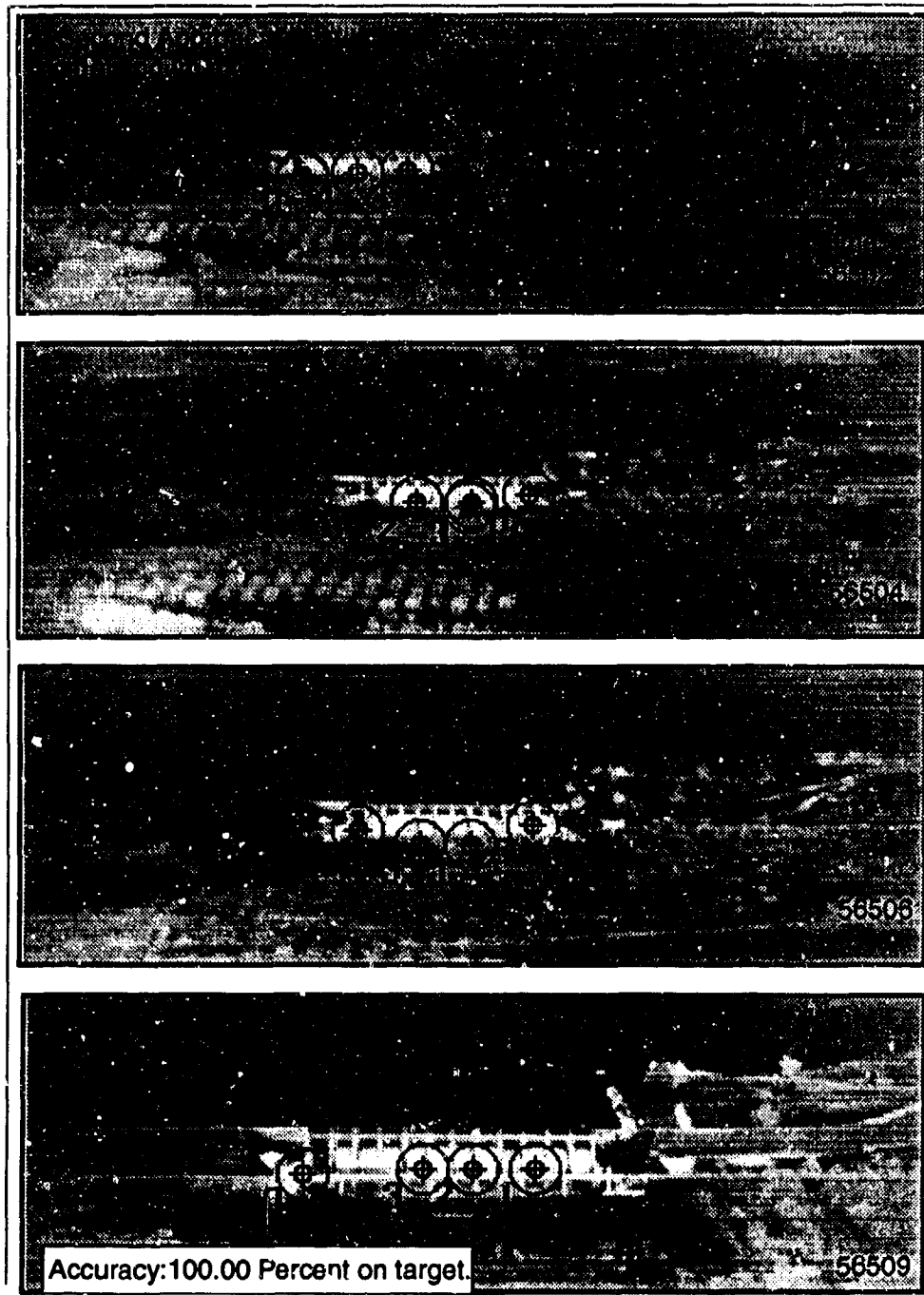


Figure 48. Approach on a Hydro-Electric Power Plant(2). Trained as before, user selecting sample textures. Four images shown out of a tracking sequence of eight images.

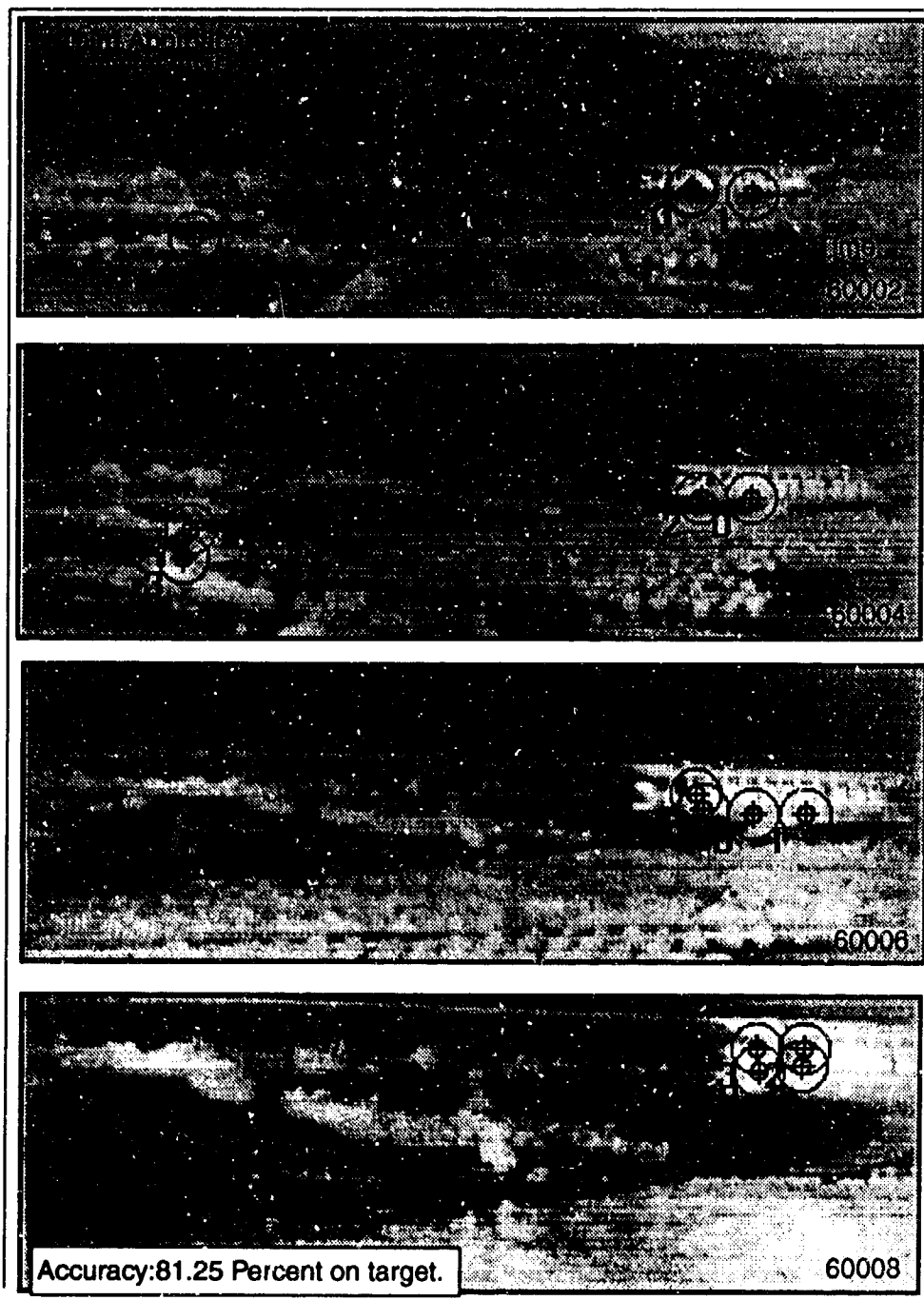


Figure 49. Approach on a Hydro-Electric Power Plant(3). In these images three targets out of 16 were not placed on target, still the tracking held to the final image.

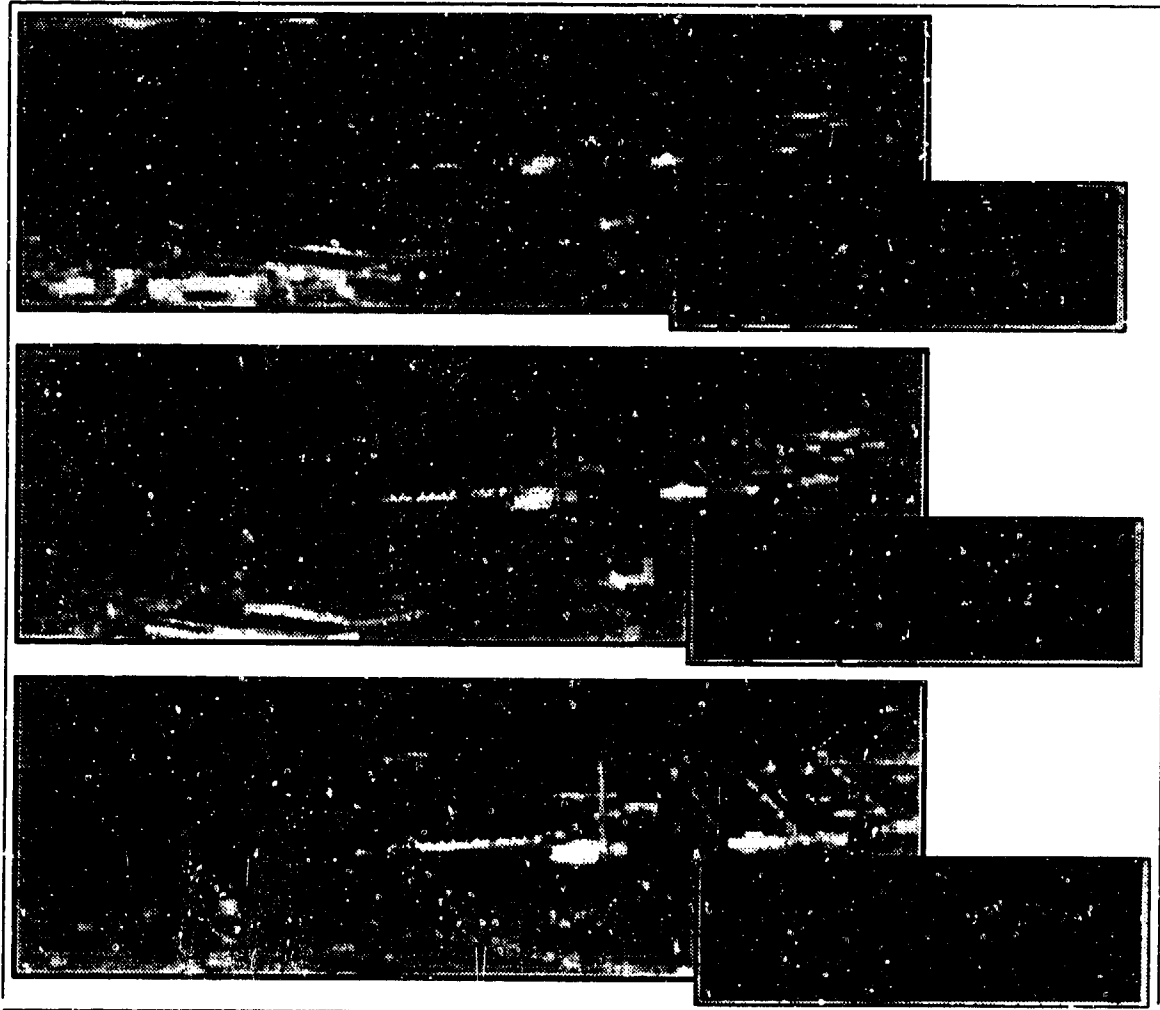


Figure 50. Approach on a Electric Power Plant. These images are part of a longer sequence of approaching a electric power plant. The image is trained on only the first image. The tracking algorithm begins by selecting the roof as the point of focus until a better target comes into view.

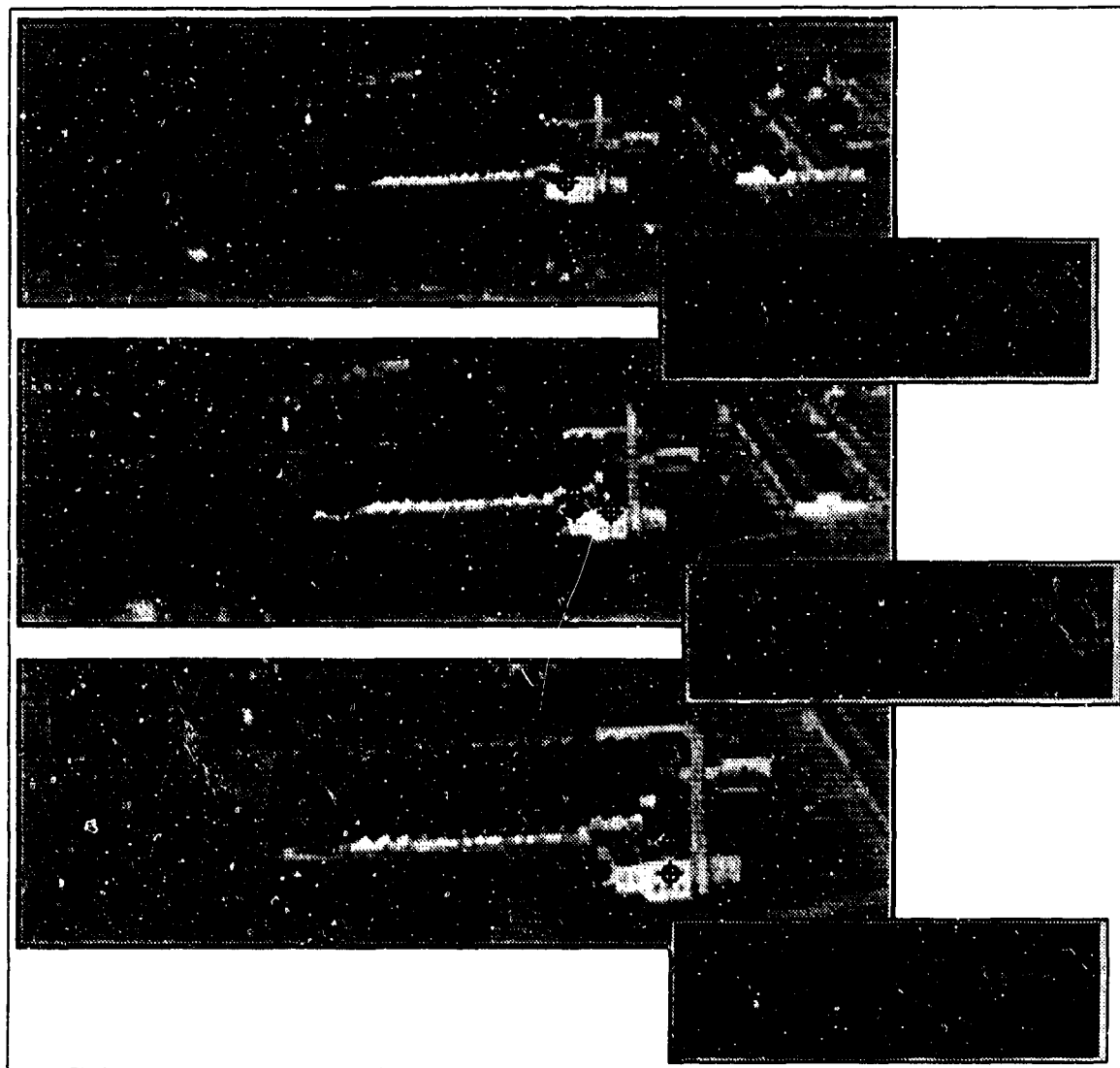


Figure 51. Approach on a Electric Power Plant. The top image shows the second training image. Once the primary target has enough detail to allow segmentation, the focus is changed to center on the smaller building in front. Using two images for training, then switching the neural network allows selecting a more precise target.

operator.⁴ The resulting autonomous targeting system requires a minimal amount of expert system rule based analysis. The approach presented here learns to recognize targets much the same way people recognize targets, by seeing examples and being told what they are.

⁴Except for the normal neural network architecture and learning rates. However, the systems were robust over a wide range of values.

VI. Conclusions and Recommendations

6.1 Conclusions

A difficulty in working on computer vision problems is that the results are often compared with science fiction or biological systems. In fiction, anything is possible; vision technology is taken for granted. Biological vision systems perform so well that machine vision pales in comparison. The problems which can be solved as a result of this work may not be impressive until one considers the magnitude of the problem. The system described here attempted to do with a few hundred interconnected weights what requires billions of interconnected weights in a biological vision system.

Three contributions are presented in this dissertation. First, an environment for testing neural network paradigms and problems was developed for general use. Second, a generalized structure for neural network problem solving was developed and tested against a number of classification problems. Finally, a segmenting and targeting scheme was developed using a generalized neural network to track high-value fixed targets through a sequence of approach images.

The NeuralGraphics system has proven to be a useful and flexible tool for neural networks analysis on a variety of problems not presented here. The user interface allows interactive control and monitoring of the network training. Development of the system provided a number of novel programming techniques for simulating artificial neural networks. The techniques are discussed in (47).

The generalized neural network window architecture provides an attractive way to segment targets from cluttered data. The use of Gabor and Fourier spectral filters allows discrimination on any of the represented correlation planes, whether the correlations come from Fourier, Gabor or any Wavelet like filter. Use of the spectral magnitude makes the recognition shift invariant. Adding the neural network allows the spectral filter coefficients to be combined in a way best suited to discriminate between the background and targets.

The research included finding ways to reduce the size of the input vectors using saliency, principal component analysis and identity networks. The weight saliency metric allowed a user to select the best combination of input features. Examination of the Karhunen-Loève and Identity network showed a means to implement neural networks under real hardware constraints. Current hardware will often limit the size of the input feature vector. Many integrated circuits limit the input vector to 64 features. Using Karhunen-Loève networks, the feature vector was reduced by a factor of three and retained the same performance. Identity networks eliminated the need for off-line preprocessing the feature vector. The identity nets were improved upon by adding additional processing layers for statistical normalization, and also orthogonalization with Gram-Schmidt techniques.

The generalized architecture and the reduced feature vector analysis allowed implementation of a scanning receptive field system for segmenting target from non-target pixels in sequential data. The tracking mechanism added to the neural network segmenter was simple but effective. The rough approximation to a blob finder was able to solve the problem, without the complexity of finding contiguous regions in the segmented images. Future work might consider adding the element of time to sequential image processing. Superior performance could have been obtained if the processing rejected transient spatial noise. Implementation would required only that the targeting mechanism be trained to recognize that a target pixel would persist over several frames. The algorithms developed here could be implemented in real time using integrated circuits specifically designed for neural network architectures. This resulted in an end-to-end design for a neural network tracker. The tracker uses neural network architectures not only in the back-end as in previous work, but for the front-end: segmentation and feature extraction as well.

6.2 Recommendations

The NeuralGraphics development environment was written entirely in the C programming language. Since the software was written, C++ has become more popular and supported as a programming environment. C++ offers constructs which are well suited

to neural network simulation. Porting the routines which perform the internal network computations would be a simple matter and would facilitate software maintenance. Since the programs were written more workstations have incorporated X-window based graphics using the Motiff widget set. Changing the graphics modules from the SGI based libraries to X would enhance the portability.

The targeting algorithm was tested using pre-recorded video data. To further examine neural network for targeting, the video source should be a camera on a movable gimbal to allow the neural network to provide steering commands. Such a system could be implemented using a workstation with a video digitizer. A number of systems provide vector processing in the forms of digital signal processors as part of the computer system.

Appendix A. *A General Purpose Environment for Neural Network*

Simulation

A.1 Introduction

This appendix will describe the software environment used to run the neural network training experiments presented in Chapters III through V. The program, although developed as a part of the dissertation effort, goes beyond its original purpose. The program can be found on computer bulletin boards from Finland to Australia. The software has been used for classroom instruction from Florida to Oklahoma.

This chapter is organized as follows. The first section discusses the purpose of the environment, lists hardware requirements, and start-up procedures. The next section tells how to run a network problem, descriptions of how to prepare the input data, controlling the training and so on. The last section describes several of the demonstrations included in the program, hybrid nets, Kohonen maps, the traveling salesman problem and a Hopfield associative memory.

A.2 Getting Started with the NeuralGraphics Environment

In order to test and evaluate a number of paradigms and techniques, a neural network simulator with a graphical interface was constructed. While the system was used as a test bed for the image segmentation problems, it was constructed with sufficient flexibility for general research. The **NeuralGraphics** system is a collection of software tools and demonstrations that provide graphical displays and allow users to interact with the network during training.¹

The programs run on a Silicon Graphics 3120, 4D, Personal Iris and IBM PC compatible computers (with EGA and a math coprocessor). Information specific to the IBM

¹The only difference between a demonstration and a tool in this package, is that a demonstration uses a fixed data set.

version is contained in the on-line help facility and is documented in the NeuralGraphics User's Guide (46). The NeuralGraphics workstation system is made up of a neural network simulator, and an image processing tool. The simulator code allows networks to be tested based on a user supplied data set, using a variety of paradigms.

Image processing software allows feature vector data sets to be constructed from gray scales images. Several data sets are supplied with the code. Most of the imagery is taken from the Autonomously Guided Conventional Weapon program.

The system will come up by typing the program name "net" in the appropriate directory. Older Silicon Graphics (SG) machines need to be in the window server (type "mex").² The first thing required by the system is setting up the network parameters. A good set of defaults is provided by the program, so in most cases a carriage return will suffice (or clicking "done").

For the SG, pick the configure item using the mouse and the right button. The default values in the boxes can be changed by clicking on the small white circles. Clicking on an item opens a dialogue window to type in the new values.

The system contains several models and demonstrations. The primary systems used to evaluate user data sets are contained in the feedforward models. These include several flavors of multi-layer backpropagation and a hybrid³ propagation network. Figure 52 shows the opening menu for the IBM version of the code.

The demonstrations include Kohonen mapping, a neural net solution for the traveling salesman problem and a Hopfield associative memory.

While all models are not supported for both implementation, (i.e. Silicon Graphics and IBM) the user interface works about the same for each. Currently, the Silicon Graph-

²Older SG machines allow for running programs with a single window environment. For multiple windows, the program mex must be run before the net program.

³The present convention in neural network literature is to define a hybrid network as a mix of electronic and optical technologies. The term is sometimes used to indicate a mix of self-organizing and tutored training paradigms. The hybrid neural network discussed here refers to the second meaning, a mix of Kohonen and backpropagation learning.

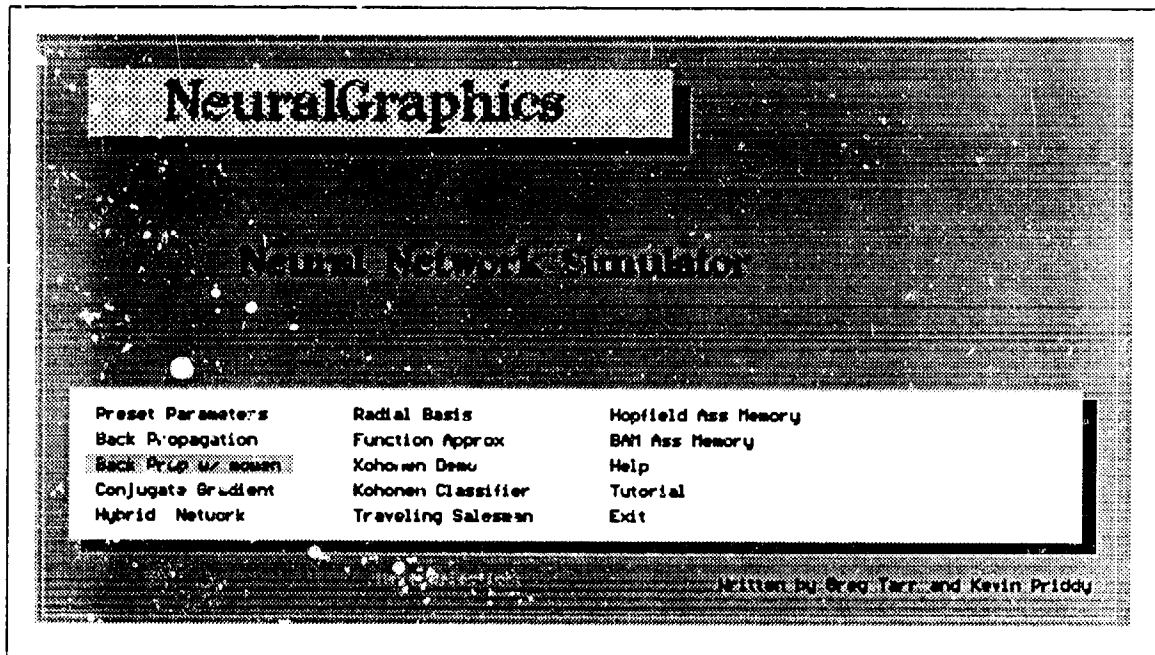


Figure 52. The Opening Menu of the PC NeuralGraphics Software.

ics tool does not support Hopfield demonstrations and the IBM does not support Error Analysis.

Much of the information specific to the IBM implementation is contained in a help tutorial, one of the menu items. To access the help files, select the menu item *Tutorial*.

A.3 Running the Feedforward Tools

Several feedforward models are provided that allow up to five layers of weights and any number of hidden nodes.⁴ Only the display limits the number. The user can select any number of weights and nodes per layer, but only about 50 nodes are displayed. The network uses dynamic allocation of resources, so every selection forces a trade off. Using larger net models mean less memory data storage and so on. Trying to allocate too much

⁴The program allows more, but the display becomes crowded, and the error term is diluted for each additional layer. Theoretically, Cybenko (7) showed that only two layers are ever required, however the generalized model uses more.

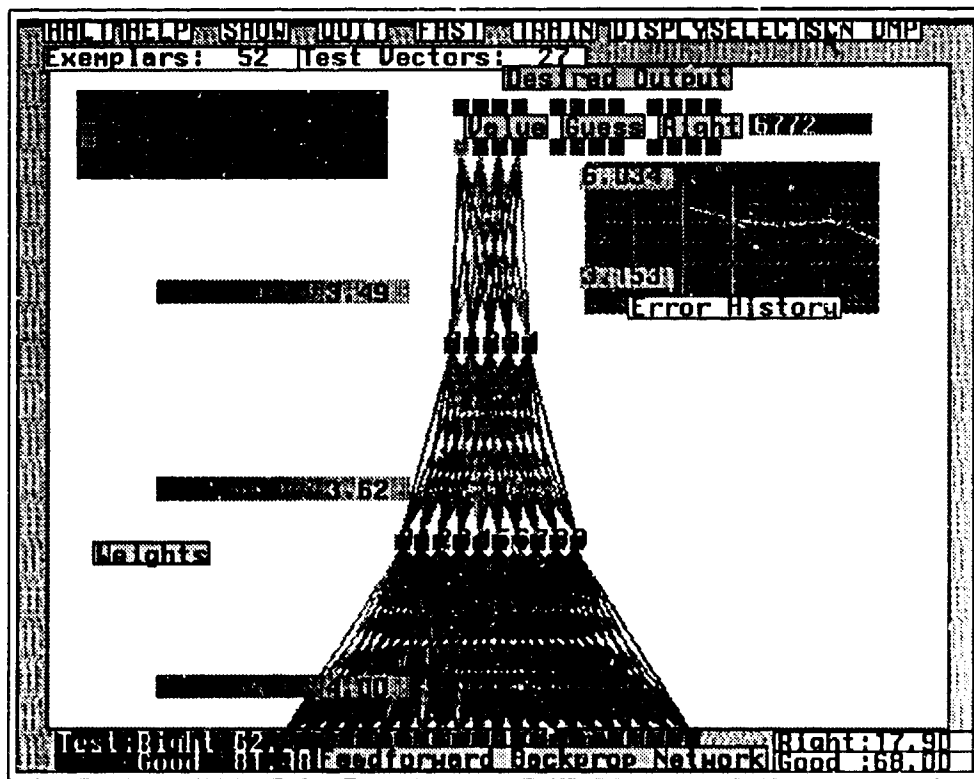


Figure 53. A Three Layer Network Display on a Silicon Graphics 3120.

memory, will cause the program to terminate. An error message will be written to the error channel explaining the problem.

The first step in using the simulator is to set up the desired network. Several types are available, which include Multi-layer Feedforward Back propagation of Errors, mixed propagation rule ⁵, and Kohonen Mapping.

Both systems prompt the user for setup data, the workstation models require a mouse, while the IBM version uses the arrow keys. On start-up, the user selects a paradigm or demonstration with the mouse or arrow keys. Select the net type, then in response to the prompts, type in the number of nodes (computation units) in each hidden layer.

Request for a weight file can be entered at start-up. If there are no previous files, enter "r" for random weights to be generated by the program.

A.3.1 The Input Data File. The training file contains all the data for testing and training the neural network. The training file name is entered at the prompt requesting data. A training file can be created as a standard ascii file using a standard text editor. For quick testing, enter the word "edit" for the data file and a "built-in" lotus style editor allows a file to be created randomly. Sometimes it is easier to modify one of these files than to start from scratch.

The basic format is show in Figure 56. The first line defines the size of the training set, test set and the size (width) of the input and output vectors. After that, each exemplar is listed in order. The first number is an arbitrary integer and is ignored by the program. The only purpose is to identify the exemplar number. Next list each element of the exemplar vector $x_0, x_1, x_2 \dots x_{n-1}$. The last element is the exemplar class type.

Exemplar class types must be sequential, i.e., 1,2,3,... etc. The first class type must be one and no numbers can be skipped. Classes can be randomly mixed within the data file. To allow flexibility, very little error checking is performed on the input file. An

⁵Some training rules mix paradigms between layers, using for example backprop on the output layer and self-organization on the input layer.

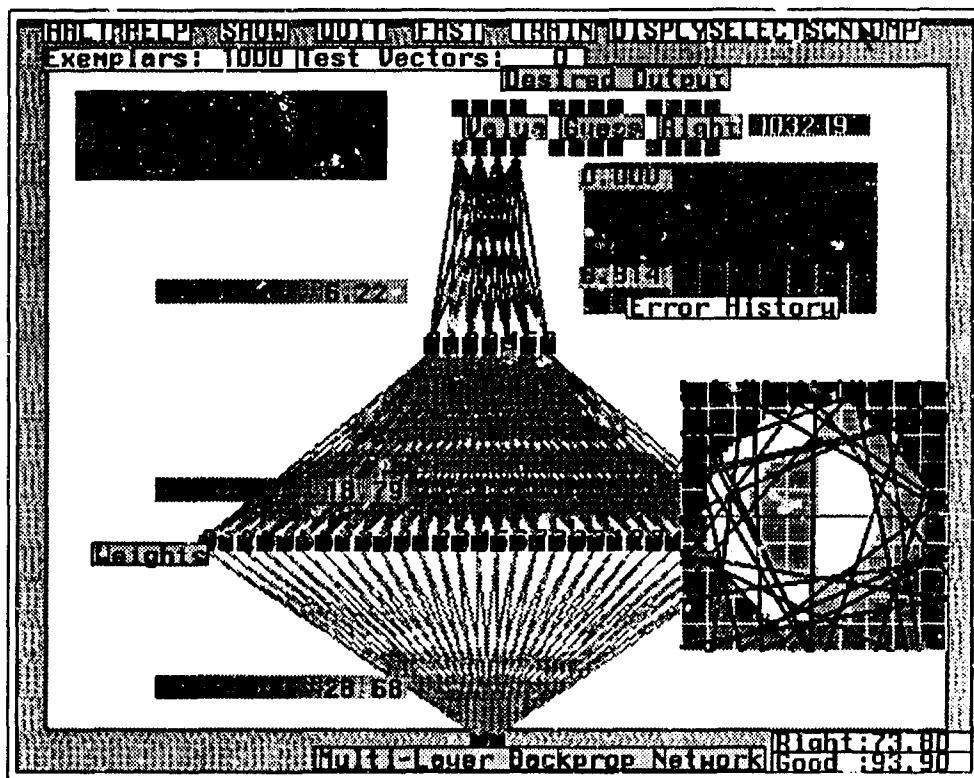
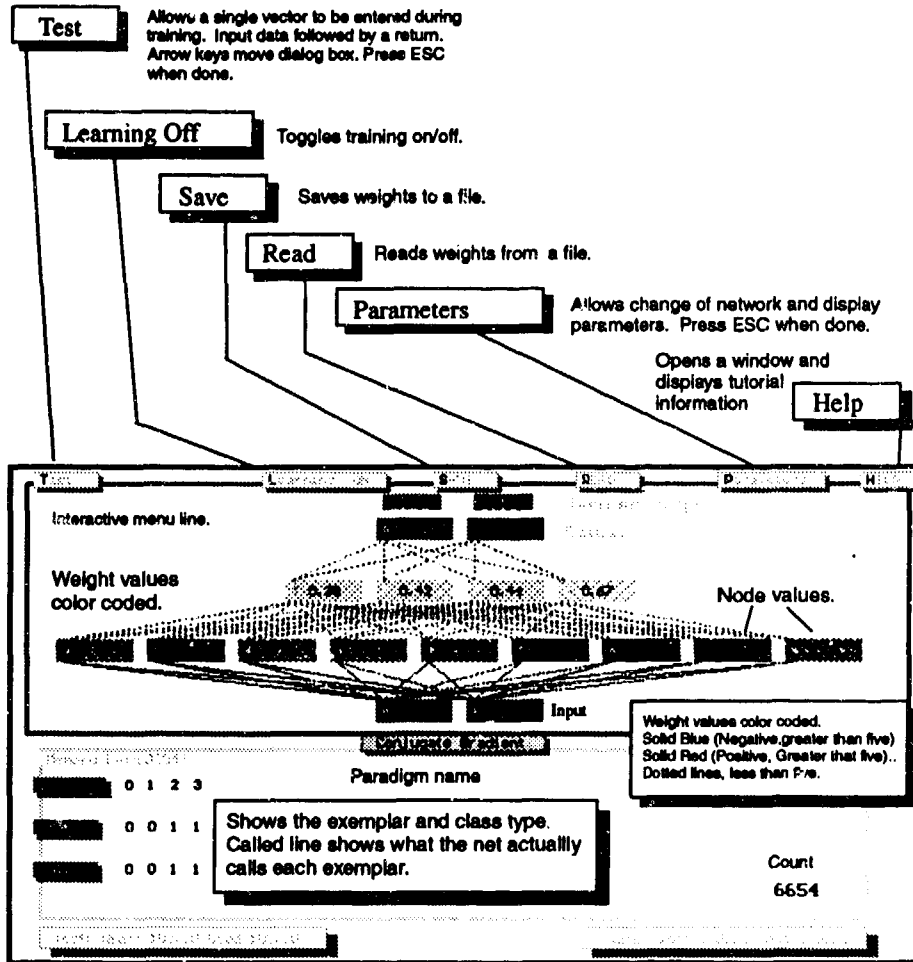


Figure 54. Neural Net O-Scope on a Silicon Graphics

Interactive Control of PC feedforward programs.



Accuracy of test set classification.
Measured at the end of each display cycle.
Right means all outputs within 20%
of desired value. Good means highest
output corresponds to the correct class.

Accuracy of training
data classification, measured
during the the display cycle.

Figure 55. PC Feed forward models screen display

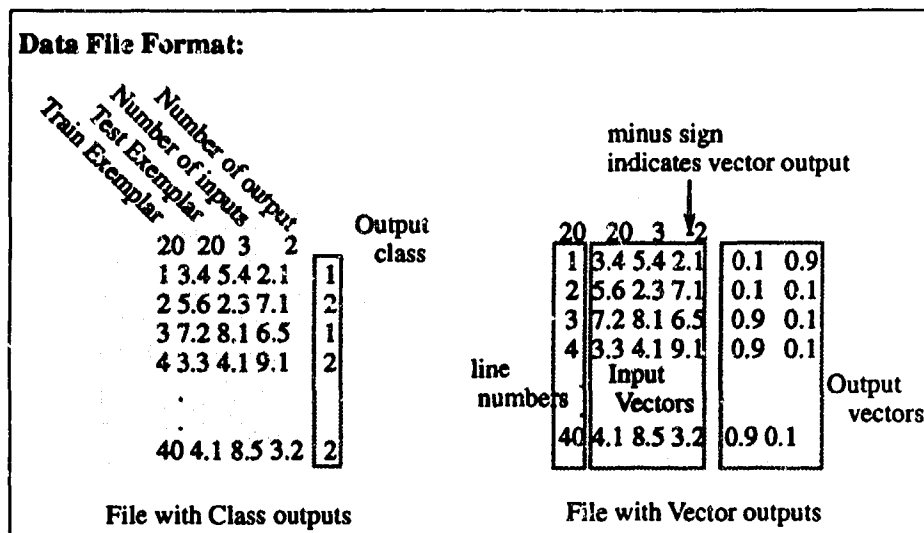


Figure 56. Data file formats for class and vector data.

incorrect file can cause the program to appear dead. If this happens use control-C to get to the extended menu. The *class* data format is shown in Figure 56. The file number of the exemplar line represents class membership. The value can be any number from 1-256, but every class must be represented. For large numbers of classes the output can be binary encoded. The output of the net then will have more than one node high at any one time. Class is assigned by setting the output to one if the node is more than a half, and zero if less than a half. The output is read as a binary code one bit per node.

A.3.2 Function Approximation. For function approximation, the specific desired value of a node can be given. This is called vector output, and specified in the parameter adjustments as one of the output encoding techniques. The value must be between 0.0 and 1.0 unless the function approximation paradigm is used. Function approximation allows greater values and uses the backpropagation with momentum technique. For function approximation the only meaningful output is the mean square error between what is desired and what is actually calculated by the network. In the example files shown in Figure 56 there are 20 training exemplars and 20 test exemplars, three inputs, and two outputs in vector format (the minus sign). The number of hidden units is set elsewhere in

the program (at runtime on in the defaults setup file).

It should be pointed out that the program looks at function approximation in two manners. The first uses any appropriate paradigm. The output is specified using a limited number of output vectors. And, the output vector for specific class is always the same. With this limitation, it makes sense to calculate a percent correct. For the function approximation paradigms, input vectors are not associated with a class. Consequently, the only measure of performance is the total error for the given data set.

Two files for testing and demonstration are included in the IBM package. These are: xor.dat and test.dat. The silicon graphics version offers more in the data directory.

The system uses dynamic memory allocation, so the size of the input and output vectors, and the number of hidden nodes is not strictly limited. The screen display shows the amount of memory left at run time. Should you try to allocated more memory than is available, the program will tell you so, and exit.

A.3.3 Interaction With the Network. The mainframe version interacts with the user in two ways: a mouse for graphical versions, and a pull down menu for the terminal version. The terminal version, usually used for background and batch jobs, allows access to a user control menu by hitting a control -C.

The three mouse buttons are used as follows:

- Right: Selection of top menu items.
- Left : Removes/replaces selected Nodes.
- Middle: Magnification of the screen display.

The screen displays information about the progress of the network as it trains. The top line allows menu selections by clicking the right mouse button as follows:

- HALT: Toggles a halt action (freezes display).

- **HELP:** Displays a help file.
- **SHOW:** Displays node values.
- **QUIT:** Exits program.
- **FAST:** Toggles display update between one to pre-set value.
- **TRAIN:** Toggles training on and off.
- **DISPLAY:** Displays error surfaces for selected nodes. Only works for two input files.
- **MENU:** Pulls up the paradigms setup menu. Only effects items that make sense to change after the network configuration has been fixed, for example the display interval.
- **SCN DMP:** Dumps screen display to a file. (Utah RLE format);

The IBM version menus are activated by pressing the first letter of the menu item listed on the top line of the display. See Figure 55.

A.3.3.1 Error Measurements. Several error measurements are built into the system. The error history is a graph that displays the progress of the network as training progresses. Also, after each display cycle, a tabulation is made to determine the percent correct for both the training set and the test set. If the output of the network is within 20 percent of the desired value for every node, a counter is incremented for the *Right* indicator. If the highest output corresponds to the correct node for that test case the *Good* counter is incremented. These values are displayed for the training and test sets as a percentage correct.

A.3.4 Statistics and Runtime Files(SG only). Statistical and historical data are kept on each run in a file called `data_stats`. This is a print out of the accuracy and error for the training and test set for each display period. When the stopping point is reached, the program re-initializes and starts over. A running average is kept for the number of

runs in a file called `data_stats_old`. The stopping point and the number of runs, are set in the initial configuration menu, or using the setup file (see Section A.3.6) Another useful file is called `item_report`. The test data is computed at each display cycle and the result is shown in a list for every test exemplar. The list shows the exemplar number, the class and the what the net estimated for that item.

A.3.5 Additional Features. The feedforward models offer a few special features.

- **Extended Menu.** The net program, in terminal mode, allows user control with an extended menu. The control-C invokes the user menu where several parameters can be adjusted by using the extended menu. Weight files can be saved or restored, displays can be adjusted and internal values can be displayed.
- **Neural Net Probe.** When the input is exactly two entries wide, a neural network O-scope kicks in which displays the response of a node over the total range of the input.
- **Node Elimination.** Placing the cursor over a node and pressing the Left Mouse button, removes the node and all connected weights from the network.
- **Magnification Window.** Placing the cursor on any point in the screen can blow up that region for examination. This capability is used to examine the value of the weight near the node terminals.
- **Hard Copy.** Network progress is also tracked in a file called `data_stats`. Any orderly termination of the program will allow the file to print training graphs using Mathematica.

A.3.6 The Setup File. The default setup is kept in a special file called `setup.fil`. When the program starts up the file is read and all default values are taken from this file. See Figure 58.

The first line shows the number of layers, followed by the the number of nodes in each hidden layer. The next line is the weight file to be loaded. Normally weights are

randomly set by using "random". `random` is a keyword which sets the weights between -0.5 and 0.5.

The next item listed is the amount of noise added to each vector. The amount of noise added to each feature of the input vector is computed by taking a random variable in the range [-1,1] and multiplying it by this entry each time the input layer is filled.

The next item is the training paradigm number. This number is the paradigms number taken for the paradigm selection list in the menus.

Next the specified data file is entered. This is the name of a data file prepared as illustrated in section A.3.1

The next number specifies the preprocessing method. Sometimes the training is faster if the data is preprocessed. The program provides several preprocessing methods. For the specific numbers, look at the function `normalize()`, in the file `normal.c`. The algorithms are explained there as well. To leave data alone, use zero, to normalize, statistically use one. Other functions were implemented for specific research questions, and include Fisher linear discriminants, and Karhunen-Loève with several variations. The utility of these functions is examined in chapter IV.

The next line tells the program where to store the statistical data. This can be useful when running in the background.

The final few lines are used to run programs in batch or background mode. The first number on the line specifies the display update increment. In the background program mode, this number is used to determine when to do all the error calculations. The next number specifies when to stop, and the last number denotes the number of runs.

A.3.7 Running Neural Graphics in the Background. While trying to run the Neural Graphic software without any graphics or screen displays may be like reading "The Collected Works of Picasso", in braille. It is possible, but the main points are missed. The terminal version of the program allows two command line parameters for this purpose.

The flag `-bg` turns off all graphics. The background mode flag allows the program to be run as a background job in the normal Unix manner. Just use the command line `net -bg & .` The program will run using the configuration specified by the setup file `setup.fil`. If a user wants to run several jobs consecutively, a different `setup.fil` is needed for each run. Consecutive jobs with different configurations can be run by using a script file. All that is necessary, is to use the flag `-setup` followed by a filename. Instead of using the default setup configuration in `setup.fil`, the user supplied name is used. Consequently, the statistical information is saved using the name specified in the setup file. If jobs are running concurrently, each job should be processed in a different directory, otherwise temporary files with the same names will clash.

Also, a popular method for analyzing a data set is called hold-one-out. The network is trained on all data points except one. When training is complete, the one held out is processed through the net. The network is retrained until all exemplars have been processed. The results are stored in a file called `hold_one_out`. To invoke the process in batch mode use a line parameter `"-hold"`.

In the graphics program the option is available under the output type, under data class types in the window menu. Hold one out is a variation on the CLASS data type.

A.4 Other Models and Demonstrations

A.4.1 Hybrid Propagation. From the users point of view, this paradigm works the same as the Backprop model. The difference is in the convergence time for various types of problems. Backprop has difficulty training when decision regions are very close together, but works well for many distinct and disjoint decision regions. The hybrid propagation network works well for several disjoint regions that are spaced closely together as long as the number of first layer nodes is a few more than the number of distinct decision regions. Hybrid propagation uses Kohonen self-organization on the input layer followed by a two layer backprop for classification.

Hybrid propagation is used in this package as a conglomeration of hybrid type

networks such as ART2 (4) and Counterpropagation (18). Hybrid propagation requires only a conscience parameter to be adjusted while ART and Counterpropagation require several parameters to be set exactly. While ART2 is generally considered an unsupervised method, it becomes supervised during the calibration process, where labels are associated with each output node.

A.4.2 Kohonen Self-Organizing Maps. Kohonen maps organize data according to the distribution of the training data. Neural Graphics provides a demonstration that graphically illustrates the distribution of the Kohonen nodes with different distributions of input data.

The first selection allows the training data to be distributed across a square, a triangle or a cross. The second allows the input in the x and y directions to be distributed according to a uniform, Gaussian or a chi-square probability function.

The display has three maps: a map of the input data(upper right), a Kohonen map (lower right) and a graphical display of the distribution of the Kohonen nodes. The input data is colored red and yellow to form two arbitrary input classes. The class information is not used in the training of the net. It is only provided to demonstrate the grouping of the Kohonen nodes for data with similar characteristics.

A.4.3 The Traveling Salesman Problem. The traveling salesman problem is a classic example of an Non-polynomial (NP) complete problem. This demonstration solves a traveling salesman problem from either a randomly generated list of cities or a specified file.

The input file defaults to sales.dat. The file consists of a number of cities, the word 'list' or 'random', a problem number and a list of x,y locations of the cities.

The display shows the current location of the nodes with interconnecting lines, the total length of the tour, the current number of nodes, and the current city being evaluated.

A.4.4 Hopfield Associative Memory. The Hopfield associative memory model accepts a user file. The first line of the file contains three pieces of information. The first value, as with the other models is the number of exemplars. The second and third relate to the display only. These are the width of each exemplar followed by the height respectively. On the next line the patterns follow. First comes an unused pattern number. After that each bit of the exemplar pattern is listed 0 for pixel off, 1 for pixel on.

The Hopfield net treats each exemplar as a single binary word, the height and width are for the benefit of the user only. The package provides a sample input file called "hop3.dat".

The Hopfield network is an associative memory. Several patterns are stored in a set of weights. A pattern is selected, then corrupted with noise randomly inverting bits in the image plane. A Hopfield uses only binary patterns. The image is iterated through the network feeding the output of the net back on itself. The result is hopefully, that the pattern will converge to a noise free pattern of the closest match of the input exemplar. Two things can effect convergence. One is the number of patterns stored in the network and the second is the amount of noise added to the exemplar pattern. A third consideration is the Hamming distance between exemplars, obviously if two exemplars are very close to each other, small amounts of noise can confuse the network. The rule of thumb is .15 stored images per bit in the pattern. These routines seem to work only for much smaller numbers of exemplar sets.

A.5 Conclusion

The NeuralGraphic software package provides a simple interface for testing the viability of using neural networks. Applications and problems that arise in other areas of study may be tested quickly without having to write primitive I/O and display routines. The modular design allows rapid prototyping of novel preprocessing and learning techniques. Neural Graphics is first a graphics program. The graphic design allows simple presentation of a large amount of information. While the software package was written to support only

this dissertation effort, additional effort was put into the user interface to make it more of general purpose tool than strictly research software.

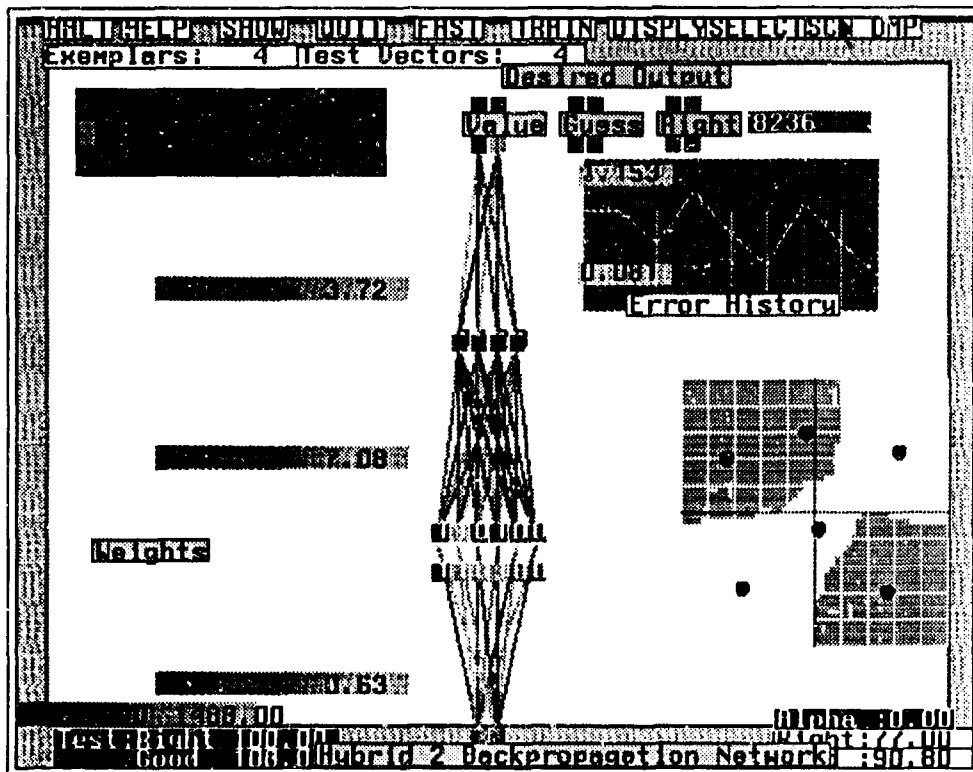


Figure 57. A Hybrid Propagation Network

```

3 20 5      :Number of Layers and Hidden Nodes
random
0.000000   :noise
1          :training rule
lant.dat
1          :Normalization
data_s:ats
1000 20000 1 :display stop point, number of runs
0 0        :saliency, Output data

```

Figure 58. Example Setup File.

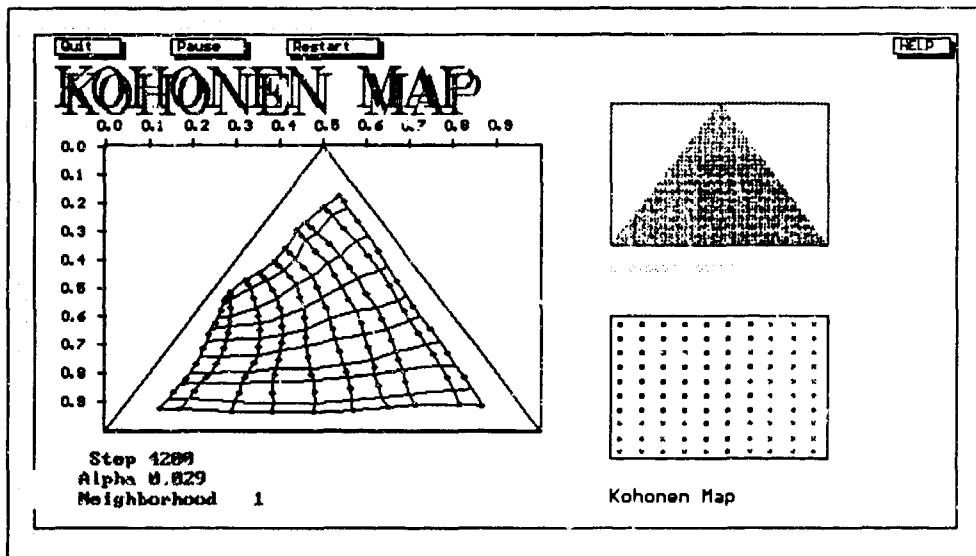


Figure 59. The Kchonen Self-Organizing Map

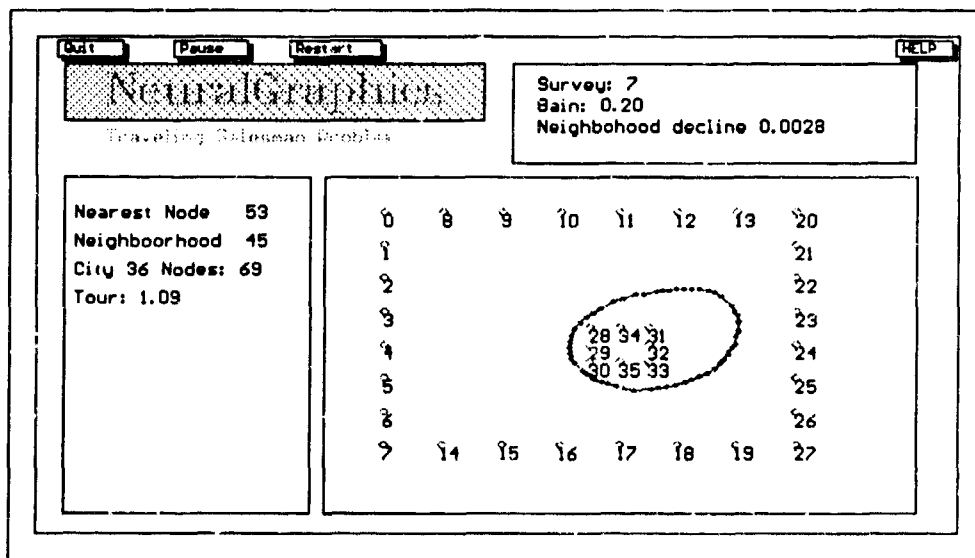


Figure 60. The Traveling Salesman Problem

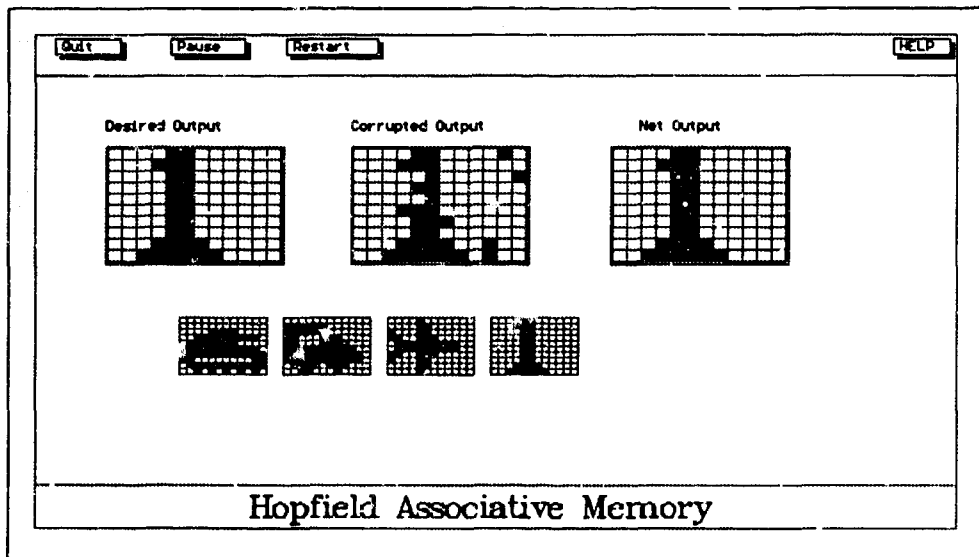


Figure 61. The Hopfield Associative Memory

Bibliography

1. Anonymous. Network conversation. Anonymous conversation between users of the ARPAnet Read News Service, December 1990.
2. Kevin W. Ayer. Gabor transforms for forward looking infrared image segmentation. Master's thesis, Air Force Institute of Technology, 1989.
3. Andrew R. Barron. Statistical properties of artificial neural networks. In *Proceedings of Conference on Decision and Control*, 1989.
4. Gail Carpenter and Stephen Grossberg. Art 2 self organization of stable category recognition codes fo analog input patterns. *Applied Optics*, December 1987.
5. Thomas M. Cover. Geometrical and statistical properties of system of linear inequalities with applications to pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326-334, June 1965.
6. Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541-551, 1989.
7. G. Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Controll, Signals and System*, 2(3):303-314, 1988.
8. John G. Daugman. Complete discrete 2-d gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Accoustics, Speech, and Signal Processing*, 36(7):1169, July 1988.
9. Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.
10. Y. Le Cun et al. Optimal brain damage. *Advances in Neural Information Processing Systems 2*, 1, 1990.
11. Martin A. Fischler and Oscar Firschein. *Readings in Computer Vision: Issues, Problems, Principles and Paradigms*. Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, California, 1987.
12. Michael K. Fleming and Garrison W. Cottrell. Categorization of faces using unsupervised feature extraction. *IEEE International Joint Conference on Neural Networks*, pages 65-70, 1990.
13. Donald H. Foley. Considerations of sample and feature size. *IEEE Transactions on Informaticn Theory*, IT-18:618-626, September 1972.
14. Gorman and Terrence Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75-89, 1988.

15. Stephen Grossberg. The adaptive brain I: Cognition, learning, reinforcement, and rhythm and the adaptive brain II: Vision, speech, language, and motor control. Elsevier/North-Holland, 1986.
16. Ernest L. Hall. *Computer Image Processing and Recognition*. Computer Science and Applied Mathematics. Academic Press, 1979.
17. Naser A. Hamadani. Automatic target cueing in infrared imagery. Master's thesis, Air Force Institute of Technology, 1981.
18. Robert Hecht-Nielsen. Applications of counterpropagation networks. *Neural Networks*, 1:131-139, 1988.
19. Kenneth Hoffman and Ray Kunze. *Linear Algebra*. Prentice Hall, 1971.
20. J. Jones and L. Palmer. An evaluation of the two dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58:1233-1258, 1987.
21. B. Julesz and J.R. Bergen. Textons, the fundamental elements in preattentive vision and perception of textures. In Martin A. Fischler, editor, *Computer Vision*, chapter 2, page 243. Morgan Kaufmann Publishers, Inc., 1987.
22. Matthew Kabrisky. *A Proposed Model for Visual Information Processing in the Human Mind*. University of Illinois Press, Urbana, IL, 1966.
23. Eric R. Kandel. Small systems of neurons. *Scientific American*, 241(3):67-76, September 1979.
24. Kasimir Klimassauskas. Neural nets tell why. *Dr. Dobbs Journal*, page 1025, April 1991.
25. Teuvo Kohonen. An introduction to neural computing. *Neural Networks*, 1(1):3-16, 1988.
26. Teuvo Kohonen, K. Makisara, and Saramaki. Phonotopic maps: Insightful representation of phonological features for speech representation. In *Proceedings of IEEE 7th International Conference on Pattern Recognition*, Montreal, Canada, 1984.
27. A. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk USSR*, 114:953-956, 1957.
28. Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April 1987.
29. Frank A. Maher. A correlation of human and machine pattern discrimination. *NAECON*, 78:260-264, 1970.
30. Nils J. Nilsson. *Learning Machines*. McGraw-Hill Book Company, 1965.
31. E. Oja. A simple model neuron as a principal components analyzer. *Journal of Mathematical Biology*, pages 267-273, 1982.

32. Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley Publishing Company, Inc, 1989.
33. Kevin L. Priddy. Bayesian selection of important features for feedforward neural networks. Submitted for Publication, August 1991.
34. Michael C. Roggemann. *Multiple Sensor Fusion for Detecting Targets in FLIR and Range Images*. PhD thesis, Air Force Institute of Technology, June 1989.
35. Dennis W. Ruck. Multisensor target detection and classification. Master's thesis, Air Force Institute of Technology, 1987.
36. Dennis W. Ruck. Multisensor fusion target classification. In *SPIE Symposium on Optics, Electro-Optics, and Sensors*, 1988.
37. Dennis W. Ruck, Steven K. Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *The Journal of Neural Network Computing*, 2(2):40-48, Fall 1990.
38. Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley, and Bruce W. Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1990.
39. Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley, and Bruce W. Suter. The multilayer perceptron: A Bayes optimal discriminant function approximator. *IEEE Transactions on Neural Networks*, 1, March 1990.
40. Dennis W. Ruck, Steven K. Rogers, Peter S. Maybeck, and Matthew Kabrisky. Back propagation: A degenerate Kalman filter? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, June 1989. Accepted for publication.
41. Richard A. Silverman. *Introductory Complex Analysis*. Dover Publishing Inc., 1972.
42. Bruce W. Suter and Mark E. Oxley. A unified theory of feedforward neural networks. *In Publication*, 1990.
43. Gregory L. Tarr. AFIT neural network development tools and modeling artificial neural networks. In *SPIE Symposium on Applications of Artificial Neural Networks*, 1989.
44. Gregory L. Tarr. Neural network development tools at the Air Force Institute of Technology. In *Proceeding of the 6th Annual AAAIC*. ACM/SIGART, 1989.
45. Gregory L. Tarr. Artificial neural network implementation of the karhunen-loeve transforms for feature vector reduction. In *Proceeding of the 7th Annual AAAIC*. ACM/SIGART, 1990.
45. Gregory L. Tarr. *NeuralGraphics User's Guide*. In publication, 1991.
47. Gregory L. Tarr, Steven K. Rogers, Mark E. Oxley, Matthew Kabrisky, and Kevin L. Priddy. Effective neural network modeling in c. In *Proceedings fo the 1991 International Conference on Artificial Neural Networks*, Espoo, Finland, June 1991.

48. Steven E. Troxel. Position, scale, and rotation invariant target recognition using range imagery. Master's thesis, Air Force Institute of Technology, 1986.
49. L. G. Valiant. A theory of the learnable. *Comm ACM*, 27(11):1134–1142, 1984.
50. Danial Zahirniak. Characterization of radar signals using neural networks. Master's thesis, Air Force Institute of Technology, 1990.

Vita

Captain Gregory L. Tarr was born on September 18, 1952 in Roseburg, Oregon. He graduated from Benson Polytechnic High School in Portland, Oregon in 1970. Capt. Tarr entered the Air Force in December of 1978 as an Aerospace Ground Equipment Technician. After serving two years at Castle AFB, California with the 84th Fighter-Interceptor Squadron, he entered the Airman's Education and Commissioning Program completing a Bachelor of Science in Electrical Engineering at the University of Arizona in December 1983. Capt. Tarr served three years with the Aeronautical Systems Division at Wright-Patterson AFB, Ohio before entering the School of Engineering, Air Force Institute of Technology in June 1987. He is married to Lana (Evenhus) Tarr of Wenatchee, Washington and has one child: Kari Marie.

Permanent address: 10138 N. James
Portland, Oregon 97203

December 1991

Doctoral Dissertation

Multi-Layered Feedforward Neural Networks for Image Segmentation

Gregory L. Tarr, Captain, USAF

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/DS/ENG/91-01

Gregory Ahlquist
RADC/IRR
Griffiss AFB NY 13441

Approved for public release; distribution unlimited

Artificial neural network image segmentation techniques are examined. Biologically inspired transforms are investigated to preprocess images for segmentation and classification. A generalized neural network formalism is presented as a means to produce common pattern recognition processing techniques in a single iterable element. Several feature reduction preprocessing techniques, based on feature saliency, Karhunen-Loève transformation and identity networks are tested and compared. The generalized architecture is applied to a problem in image segmentation and tracking of high-value fixed tactical targets.

A generalized architecture for neural networks is developed based on the second order terms of the input vector. The relation between several common neural network paradigms is demonstrated using the generalized neural network. A feature saliency metric is developed to assign relative importance to individual features. The saliency metric is shown to be significantly easier to compute than previous methods.

A scanning receptive field neural network is developed for image segmentation. The scanning window technique is used to segment sequential images of high value fixed targets (dams, bridges and power plants). The architecture is implemented as a tracking mechanism. The implementation resulted in an improved method for target identification and tracking, using neural networks as a front end.

Neural Networks, Image, segmentation, Karhunen-Loève , Image Processing

141

Unclassified

Unclassified

Unclassified

UL

REPORT DOCUMENTATION PAGE

Form Approved
GSA GEN. REG. NO. 27

Public reporting burden for this collection of information 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 aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, 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 Reduction Project (2030-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1991	3. REPORT TYPE AND DATES COVERED Doctoral Dissertation	
4. TITLE AND SUBTITLE Multi-Layered Feedforward Neural Networks for Image Segmentation		5. FUNDING NUMBERS	
6. AUTHOR(S) Gregory L. Tarr, Captain, USAF		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/DS/ENG/91-01	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583		9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Gregory Ahlquist RADC/IRR Griffiss AFB NY 13441	
11. SUPPLEMENTARY NOTES		10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Artificial neural network image segmentation techniques are examined. Biologically inspired transforms are investigated to preprocess images for segmentation and classification. A generalized neural network formalism is presented as a means to produce common pattern recognition processing techniques in a single iterable element. Several feature reduction preprocessing techniques, based on feature saliency, Karhunen-Loève transformation and identity networks are tested and compared. The generalized architecture is applied to a problem in image segmentation and tracking of high-value fixed tactical targets. A generalized architecture for neural networks is developed based on the second order terms of the input vector. The relation between several common neural network paradigms is demonstrated using the generalized neural network. A feature saliency metric is developed to assign relative importance to individual features. The saliency metric is shown to be significantly easier to compute than previous methods. A scanning receptive field neural network is developed for image segmentation. The scanning window technique is used to segment sequential images of high value fixed targets (dams, bridges and power plants). The architecture is implemented as a tracking mechanism. The implementation resulted in an improved method for target identification and tracking, using neural networks as a front end.			
14. SUBJECT TERMS Neural Networks, Image, segmentation, Karhunen-Loève, Image Processing		15. NUMBER OF PAGES 141	
17. SECURITY CLASSIFICATION OF REPORT Unclassified		16. PRICE CODE	
18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified		19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	
20. DISTRIBUTION STATEMENT OF ABSTRACT UL		21. DISTRIBUTION STATEMENT OF ABSTRACT	

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to *stay within the lines* to meet *optical scanning requirements*.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with ...; Trans. of ...; To be published in ... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."

DOE - See authorities.

NASA - See Handbook NHB 2200.2.

NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.

DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.

NASA - Leave blank.

NTIS - Leave blank.

Block 13. Abstract. Include a brief (*Maximum 200 words*) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (*NTIS only*).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.