



US Army Corps
of Engineers

TECHNICAL REPORT O-91-1

2

ANALYSIS OF ACOUSTIC DEPTH SOUNDER SIGNALS WITH ARTIFICIAL NEURAL NETWORKS

by

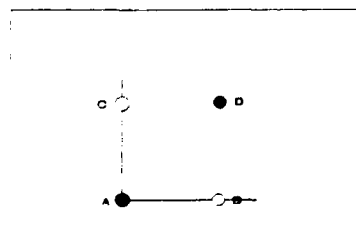
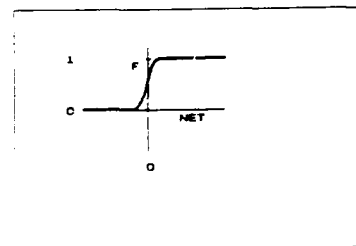
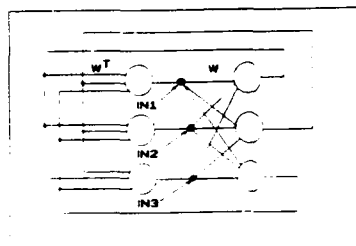
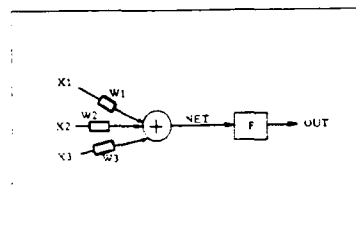
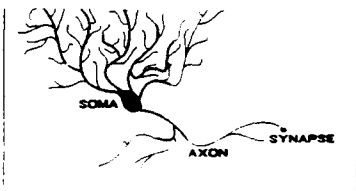
Barry W. McCleave

Instrumentation Services Division

DEPARTMENT OF THE ARMY

Waterways Experiment Station, Corps of Engineers
3909 Halls Ferry Road, Vicksburg, Mississippi 39180-6199

AD-A236 845



DTIC
S
JUL 1991



April 1991
Final Report

Approved For Public Release; Distribution Unlimited

Accession for	
DTIC	☒
DTIC	☐
Unpublished	☐
Justification	☐
By	
Distribution	
Availability	
Dist	Availability
A-1	special

91-02462



91 6 18 051

Destroy this report when no longer needed. Do not return
it to the originator.

The findings in this report are not to be construed as an official
Department of the Army position unless so designated
by other authorized documents.

The contents of this report are not to be used for
advertising, publication, or promotional purposes.
Citation of trade names does not constitute an
official endorsement or approval of the use of
such commercial products.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
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 other 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 (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE April 1991	3. REPORT TYPE AND DATES COVERED Final report		
4. TITLE AND SUBTITLE Analysis of Acoustic Depth Sounder Signals with Artificial Neural Networks		5. FUNDING NUMBERS		
6. AUTHOR(S) Barry W. McCleave				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USAE Waterways Experiment Station, Instrumentation Services Division, 3909 Halls Ferry Road, Vicksburg, MS 39180-6199		8. PERFORMING ORGANIZATION REPORT NUMBER Technical Report O-91-1		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Army Corps of Engineers Washington, DC 20314-1000		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) This dissertation documents research in which 3 problems involving the analysis of acoustic depth sounder returns that concern hydrographic surveyors are addressed using artificial neural networks. The first problem is the detection of a suspended layer of material called fluff that lies above the top layer of the bottom and poses no obstruction to navigation, but appears to the conventional depth sounder as the hard bottom. The second problem involves classifying the top layer of material as to hard silty sand, hard silty clay, or soft clay. The third problem involves classifying the density of the top layer of material of the bottom. Neural network models based on the back propagation learning method are designed and tested. Hardware is proposed for the implementation of these models. The accuracy of the models developed is compared with a conventional mathematical method.				
14. SUBJECT TERMS Acoustic signals Dredging Back propagation Neural networks Depth sounder		15. NUMBER OF PAGES 352		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT	

PREFACE

This report was prepared in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Department of Electrical Engineering, Mississippi State University (MSU), by Barry W. McCleave, Instrumentation Services Division (ISD), US Army Engineer Waterways Experiment Station (WES), Vicksburg, MS.

Sincere appreciation is extended to Dr. John K. Owens, major adviser, MSU. The encouragement and assistance provided by Drs. Ronald J. Classen, Jimmy L. Dodd, James C. Harden, Frank M. Ingels, and Jerry W. Rogers is also gratefully acknowledged.

Appreciation is extended to WES and Dr. Robert W. Whalin, the Technical Director, for use of the computer facilities and acoustic data. Additional thanks are given to George P. Bonner, Chief, ISD, and Richard G. McGee, Hydraulics Laboratory, WES.

The software described herein is available from the Engineering Computer Programs Library, Information Technology Laboratory, WES (program number 723-S8-R0101).

COL Larry B. Fulton, EN, was Commander and Director of WES during publication of this report. Dr. Robert W. Whalin was Technical Director.

TABLE OF CONTENTS

	Page
PREFACE	i
LIST OF TABLES	vi
LIST OF FIGURES	ix
 CHAPTER	
I. INTRODUCTION	1
II. NEURAL NETWORK BACKGROUND	5
2.1 Biological Neurons	5
2.2 Artificial Neurons	6
2.3 The Hopfield Neuron	7
2.4 Artificial Intelligence	8
2.5 Hebbian Learning	9
2.6 Classifications of Networks	10
2.7 Bidirectional Associative Memory	10
2.8 Content Addressable Memory	12
2.9 Adaptive Resonance	12
2.10 ADELIN and MADELINE	13
2.11 Perceptron	15
2.11.1 Limitations of a Single Layer	15
2.11.2 Hidden Layers	16
2.12 Back Propagation	18
2.12.1 Derivation of Back Propagation Equations	20
2.12.2 Derivative of Sigmoidal Function	22
2.12.3 Methods of Accelerating Learning	23
2.12.4 Abstraction	24
2.12.5 Initialization	25
2.13 Kohonen	25
2.14 Combined Networks	27
III. HYDROGRAPHY	28
3.1 Acoustic Signals	28
3.2 Depth Sounders	29
3.2.1 Velocity of Sound	30
3.3 Navigable Bottom	31
3.4 Dredging	32

CHAPTER	Page
3.5 Material Type Determined by Bottom Sampling .	32
3.6 Density Measured with Nuclear Probes	34
3.7 Acoustic Modeling	35
3.7.1 Theoretical Modeling	38
3.7.2 Statistical Modeling	40
3.7.3 Noise	41
3.7.4 Signal Abstraction	41
IV. SOFTWARE	42
4.1 Code Verification	43
4.2 Back Propagation Model Development	44
4.3 Preneural Signal Processing	45
4.4 Results Processing	47
V. FLUFF DETECTION	48
5.1 Physical Data	48
5.2 Time Domain Model	52
5.2.1 Input Generation	52
5.2.2 Model Derived	54
5.3 Frequency Domain Model	66
5.3.1 Input Generation	67
5.3.2 Model Derived	69
VI. MATERIAL CLASSIFICATION	78
6.1 Physical Data	78
6.2 Time Domain Model	80
6.2.1 Input Generation	80
6.2.2 Model Derived	82
6.3 Frequency Domain Model	88
6.3.1 Input Generation	88
6.3.2 Model Derived	90
VII. DENSITY CLASSIFICATION	95
7.1 Physical Data	96
7.2 Time Domain Model	96
7.2.1 Input Generation	97
7.2.2 Model Derived	97
7.3 Frequency Domain Model	102
7.3.1 Input Generation	103
7.3.2 Model Derived	103
VIII. RESULTS	106
8.1 Fluff Detection	106
8.1.1 Previous Attempts	107
8.1.2 Neural Network Model	107

CHAPTER	Page
8.2 Material Classification	108
8.2.1 Conventional Method	108
8.2.2 Neural Networks	109
8.2.3 Comparison with Conventional	110
8.3 Density Classification	110
8.3.1 Conventional Method	111
8.3.2 Neural Networks	112
8.3.3 Comparison with Conventional	112
IX. PROPOSED HARDWARE IMPLEMENTATION	114
9.1 Returned Envelope Circuitry	115
9.2 Sync Circuitry	117
9.3 Hardware Artificial Neurons	119
X. SUMMARY AND CONCLUSIONS	123
APPENDIX	
A. PRENEURA	130
A.1 PRENEURA Example Run 1	137
A.2 PRENEURA Example Run 2	145
A.3 Listing of PRENEURA	153
B. LETTERBP	217
B.1 LETTERBP Example Run	219
B.2 Listing of LETTERBP	223
C. BACKPROP	241
C.1 BACKPROP Example Run	243
C.2 Listing of BACKPROP	245
D. RESULTS	304
D.1 RESULTS Example Run	305
D.2 Listing of RESULTS	307
REFERENCES	333

LIST OF TABLES

Table	Page
1. Ampl 64 Inputs-Iterations Vs. Learn Rate	56
2. Ampl 64 Inputs-Uncertainty Vs. Learn Rate	57
3. Ampl 64 Inputs-Correct Vs. Learn Rate	57
4. Ampl 64 Inputs-Iterations Vs. Momentum	58
5. Ampl 64 Inputs-Uncertainty Vs. Momentum	58
6. Ampl 64 Inputs-Correct Vs. Momentum	59
7. Ampl 32 Inputs-Iterations Vs. Learn Rate	59
8. Ampl 32 Inputs-Uncertainty Vs. Learn Rate	60
9. Ampl 32 Inputs-Correct Vs. Learn Rate	60
10. Ampl 32 Inputs-Iterations Vs. Momentum	61
11. Ampl 32 Inputs-Uncertainty Vs. Momentum	61
12. Ampl 32 Inputs-Correct Vs. Momentum	62
13. Ampl 16 Inputs-Iterations Vs. Learn Rate	62
14. Ampl 16 Inputs-Uncertainty Vs. Learn Rate	63
15. Ampl 16 Inputs-Correct Vs. Learn Rate	63
16. Ampl 16 Inputs-Iterations Vs. Momentum	64
17. Ampl 16 Inputs-Uncertainty Vs. Momentum	64
18. Ampl 16 Inputs-Correct Vs. Momentum	65
19. T/F 64 Inputs-Iterations Vs. Learn Rate	70
20. T/F 64 Inputs-Uncertainty Vs. Learn Rate	70

Table	Page
21. T/F 64 Inputs-Correct Vs. Learn Rate	71
22. T/F 64 Inputs-Iterations Vs. Momentum	71
23. T/F 64 Inputs-Uncertainty Vs. Momentum	72
24. T/F 64 Inputs-Correct Vs. Momentum	72
25. T/F 32 Inputs-Iterations Vs. Learn Rate	73
26. T/F 32 Inputs-Uncertainty Vs. Learn Rate	73
27. T/F 32 Inputs-Correct Vs. Learn Rate	74
28. T/F 32 Inputs-Iterations Vs. Momentum	74
29. T/F 32 Inputs-Uncertainty Vs. Momentum	75
30. T/F 32 Inputs-Correct Vs. Momentum	75
31. Ampl Material-Iterations Vs. Learn Rate	83
32. Ampl Material-Uncertainty Vs. Learn Rate	84
33. Ampl Material-Correct Vs. Learn Rate	84
34. Ampl Material-Iterations Vs. Momentum	85
35. Ampl Material-Uncertainty Vs. Momentum	85
36. Ampl Material-Correct Vs. Momentum	86
37. T/F Material-Iterations Vs. Learn Rate	90
38. T/F Material-Uncertainty Vs. Learn Rate	91
39. T/F Material-Correct Vs. Learn Rate	91
40. T/F Material-Iterations Vs. Momentum	92
41. T/F Material-Uncertainty Vs. Momentum	92
42. T/F Material-Correct Vs. Momentum	92
43. Ampl Dens 64-Iterations Vs. Learn Rate	98
44. Ampl Dens 64-Uncertainty Vs. Learn Rate	98

Table	Page
45. Ampl Dens 64-Correct Vs. Learn Rate	99
46. Ampl Dens 32-Iterations Vs. Learn Rate	100
47. Ampl Dens 32-Uncertainty Vs. Learn Rate	100
48. Ampl Dens 32-Correct Vs. Learn Rate	101
49. T/F Dens 64-Iterations Vs. Learn Rate	103
50. T/F Dens 64-Uncertainty Vs. Learn Rate	104
51. T/F Dens 64-Correct Vs. Learn Rate	104
52. Fluff Detection Results	108
53. Material Classification Results	110
54. Density Classification Results	113

LIST OF FIGURES

Figure	Page
1. Biological Neuron	6
2. Hopfield Neuron	7
3. Sigmoidal Function	8
4. Classification Structure Chart	10
5. Bidirectional Associative Memory	11
6. Adaptive Filter	14
7. XOR Problem	16
8. Multiple Layers	17
9. Back Propagation Model	18
10. Kohonen Feature Map	26
11. Active Sonar	29
12. Depth Sounding	30
13. Grab Samplers	33
14. Phleger Corer	34
15. Nuclear Density Probe	35
16. Returns from Wide Beam	37
17. Depth Sounder's Reading	37
18. Return with Surface Reflection	50
19. Hard Bottom Return	51
20. Fluff Return	51

Figure	Page
21. Amplitude Processed Hard Bottom	53
22. Amplitude Processed Fluff	54
23. Time-Frequency of Hard Bottom	68
24. Time-Frequency Processed Fluff	69
25. Return from Silty Sand Bottom	79
26. Return from Soft Clay Bottom	79
27. Return from Hard Silty Clay	80
28. Amplitude Processed Silty Sand	81
29. Amplitude Processed Soft Clay	81
30. Amplitude Processed Silty Clay	82
31. T/F Processed Silty Sand	89
32. T/F Processed Soft Clay	89
33. T/F Processed Silty Clay	90
34. Envelope Circuitry	115
35. Uncertainty Detect Circuitry	116
36. Sync Pulse Logic	118
37. 80170NW	120
38. Op Amp/Comparator Neuron	122
39. Initial File Entry	137
40. Raw Data	138
41. Half-wave Rectified	138
42. Form Envelope	139
43. Filtered Data	139
44. Cursor Advanced	140

Figure	Page
45. Maximum Value 1.0	140
46. Jump to Threshold	141
47. Clip to 256 Points	141
48. Decimate by 2	142
49. FFT of Data	142
50. Scale to Unity	143
51. Square Root of Data	143
52. Entering File List	145
53. Initial Data Plot	145
54. First Help Screen	146
55. Second Help Screen	146
56. Advance 128 Points	147
57. Scale Between 0 and 1	147
58. Filtered and Clipped	148
59. Fourier Transform	149
60. Mark Beginning	149
61. Mark End	150
62. Clip to Power of Two	150
63. Append to Queue	151
64. Recall Temporary Buffer	151
65. Queue Replaces Data	152
66. Save Data	152
67. Initial LETTERBP Screen	220
68. Restored Problem	220

Figure	Page
69. Sample Case	221
70. Sample Pattern	221
71. First New Case	222
72. Second New Case	222
73. Example of BACKPROP	244
74. First Part of RESULTS Sample Output	306
75. Second Part of RESULTS Output File	307

CHAPTER I

INTRODUCTION

Hydrographic surveying techniques are used to produce the maps that pilots use when traversing the world's waterways. The maps show position versus depth for the channels which serve as watery highways. These depths are usually measured by acoustic depth sounders which send a burst of sound wave to the bottom and receive the reflected return; the transit time is proportional to the depth. However, this depth may correspond to the hard bottom, or it may correspond to a layer of suspended sediment that poses no threat to navigation. The signal burst is modulated by the media through which it passes and contains information which may be used to determine the type and density of the bottom material. This information takes the form of wavelets which have maximum amplitudes at maximum impedance media interfaces.

This study involves using transformed waveform envelopes as inputs into neural network models. These models are fashioned after the human nervous system and brain and attempt to simulate the human learning process. Therefore, neural networks can classify the material causing the reflection by learning identifying characteristics of

the material's acoustic return. These characteristics include the shape and relative amplitudes with time of the wavelets produced by transmission media. The media consists of a layer of water, a possible layer of suspended sediment, and a layer of saturated bottom material (e.g., silty sand, silty clay, or soft clay mud).

The purpose of this research was to advance the field of hydrographic surveying by applying neural networks in the analysis of acoustic returns obtained from conventional depth sounders. Three problems were modeled: (1) fluff detection, (2) bottom material classification, and (3) density classification. This dissertation is organized into ten chapters. The first chapter is the introduction.

Chapter II provides background material on neural networks. A number of types of networks currently in use are presented and compared. The back propagation method is described in detail as this is the method used to obtain the research models.

Chapter III provides background on depth measuring devices used in obtaining the acoustic data and the ground truth (bottom samples). The conventional approach to acoustic modeling is discussed along with the problems encountered when modeling a non-Gaussian media.

Chapter IV discusses the neural network software developed in conjunction with this research. Two programs for preparing input patterns for modeling are discussed in

overview: PRENEURA and LETTERBP. A program called BACKPROP designs and tests the neural network using back propagation learning. A program called RESULTS generates statistics and weight maps from the output of BACKPROP.

The procedure for classifying the signal inputs into categories involves linearly scaling the inputs to between 0.0 and 1.0. The output of a neuron is compressed to an analog value between 0.0 and 1.0 by a sigmoidal function; therefore, this analog value must be converted to a digital value indicating true or false for each possible classification. Thus the threshold between true and false is arbitrarily set at the midpoint of the output range (i.e., 0.5). For example, if the output neuron for fluff had a n activity level of greater than 0.5 and the hard bottom output neuron had an activity level less than 0.5, the network was said to have made the determination that fluff was present.

Chapter V deals with the fluff presence determination problem. It presents the time and time-frequency domain models developed and their performance on the test set. Time-frequency involves taking a series of Fast Fourier Transforms at chronologically advanced segments of the waveform.

Chapter VI deals with the bottom material classification problem. Models for classifying silty sand, silty clay, and soft clay are presented.

Chapter VII deals with the models developed to classify density into ranges and their relative successes. Four ranges are classified.

Chapter VIII summarizes the results of the three modeling problems. The accuracy of the neural network models is compared to the accuracy of a conventional mathematical acoustic modeling method.

Chapter IX describes the proposed hardware implementation of the real-time acoustic analysis depth sounder neural processor add on equipment. The circuitry used for preconditioning the signal as well as the neural network are presented.

Chapter X is a summary of the accomplishments and results of this research. It provides recommendations for future research in this field.

Following Chapter X is a series of appendices that expand the discussion of the software developed in conjunction with this research. The program listings and sample runs are included. The reference section follows the appendices.

CHAPTER II

NEURAL NETWORK BACKGROUND

Man has always sought to understand how nature works so he might duplicate its mechanics to better his own lifestyle. One of nature's best kept secrets has been how the human mind processes sensory inputs (i.e., how we learn). Fundamental to the learning process is the body's brain and nervous system, composed of more than 100 billion neurons.

2.1 Biological Neurons

Biological neurons carry or block transmission of information to and from the brain. They carry sensory inputs to the brain and motor responses from the brain in the form of electrical pulses through a complex network of interconnected nodes [REF 23]. A biological neuron (Figure 1) is composed of dendrites, a cell body, an axon, and synapses. The output of a biological neuron's axon stimulate the dendrites of other neurons through a biochemical reaction at a connection point called a synapse. The cell body, called the soma, performs a weighted summing of its dendrite inputs [REF 25].

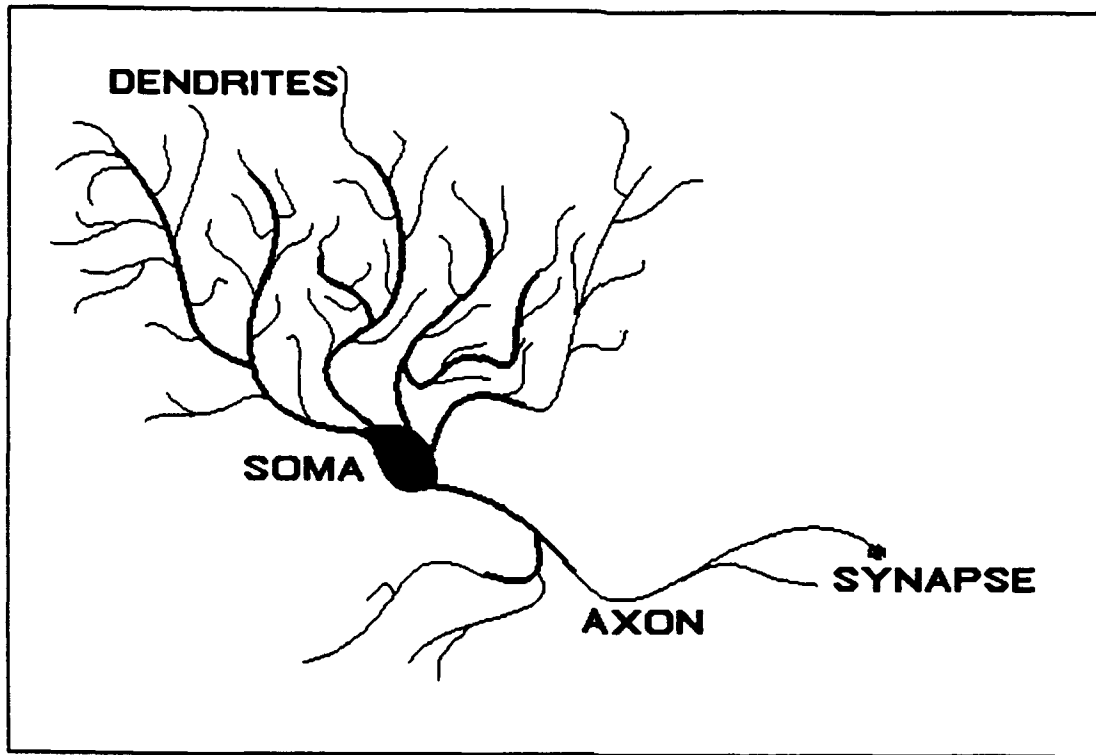


Figure 1. Biological Neuron

2.2 Artificial Neurons

Artificial neurons are modeled after biological neurons. Artificial neurons are similar to operational amplifier summers with problem specific, learned value resistor inputs. The output of the amplifier oftentimes connects to a nonlinear stage, called the activation function. Thus, neural networks are typically a form of nonlinear weighted summing [REF 4]. The output of the summer, called NET, is given by the equation:

$$NET = \sum_{i=1}^{num_{in}} w_i x_i \quad , \quad (2.1)$$

where w_i is the weight of the input multiplier and x_i is the corresponding input of the summer [REF 11].

2.3 The Hopfield Neuron

The basic Hopfield neuron (Figure 2) has two cascaded elements: (1) a summer of both inhibitory and excitatory

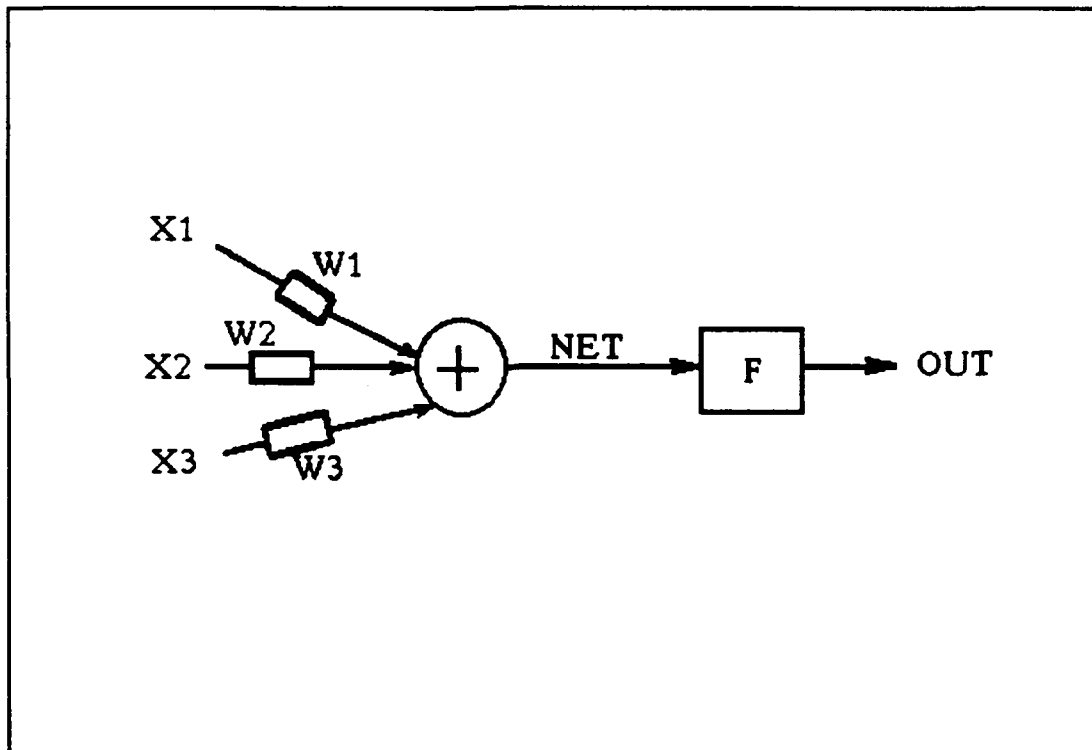


Figure 2. Hopfield Neuron

inputs and (2) a nonlinear activation function of the form

$$F(NET) = \frac{1}{1 + e^{-(G)(NET)}} \quad , \quad (2.2)$$

where G affects the slope (Figure 3) of the nonlinearity [REF 8].

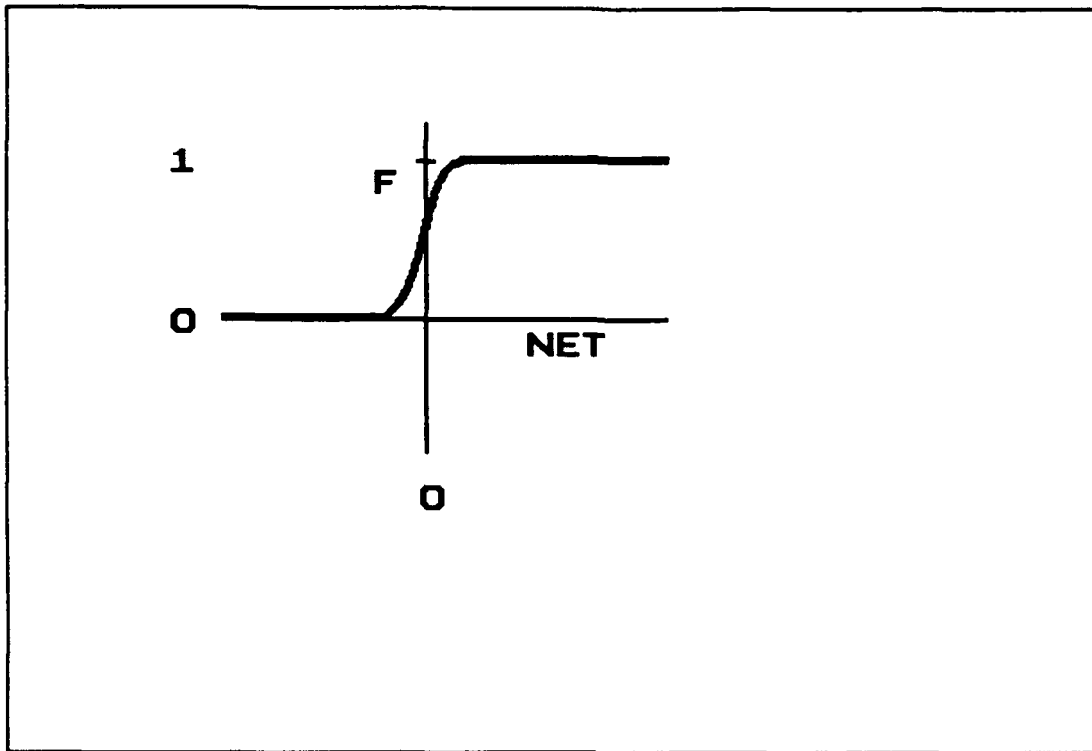


Figure 3. Sigmoidal Function

The nonlinearity gives neurons a wider dynamic range and a higher degree of compliancy. Neural nets become an adaptive mapping table from an input domain to an output domain. They can derive a original and adaptive solution to a problem.

2.4 Artificial Intelligence

The ability to learn in unsupervised mode gives some types of neural networks true artificial intelligence. Expert systems, on the other hand, simply traverse a hierarchical set of fixed rules to arrive at a conclusion. The set of rules is based on the knowledge of a human expert who accomplished the learning and transferred human-acquired

knowledge to the machine by defining the rules. The rules may work backward as well as forward (In MYCIN for example, input symptoms cause the program to request that certain tests be run to provide more inputs).

Neural networks are a form of parallel computing. Several of these artificial neurons are arranged in networks that process a layer of information in parallel. This may be accomplished by software on a general purpose computer with array processing subsystems or by specialized hardware (e.g., neural chips). Parallelism can be simulated on a sequential computer using arrays and loops.

2.5 Hebbian Learning

Donald Hebb, in 1949, theorized [REF 3] that if pairs of adjacent neuron inputs (called synapses) become active simultaneously, the connections between the neurons are strengthened [by a chemical reaction]. This biological phenomena gave way to Hebb's rule for Hebbian learning, the foundation of most neural net models. Hebb's rule is to adjust the strength of the connection between two units proportional to the product of their activation levels [REF 12]. In equation form,

$$\Delta W_{ij} = B(OUT_i)(OUT_j) \quad , \quad (2.3)$$

where w_{ij} is the weight connecting neurons i and j , OUT is the respective neuron output, and B is the learning rate.

2.6 Classifications of Networks

Neural networks may be classified as to their characteristics, such as, method of learning and structure.

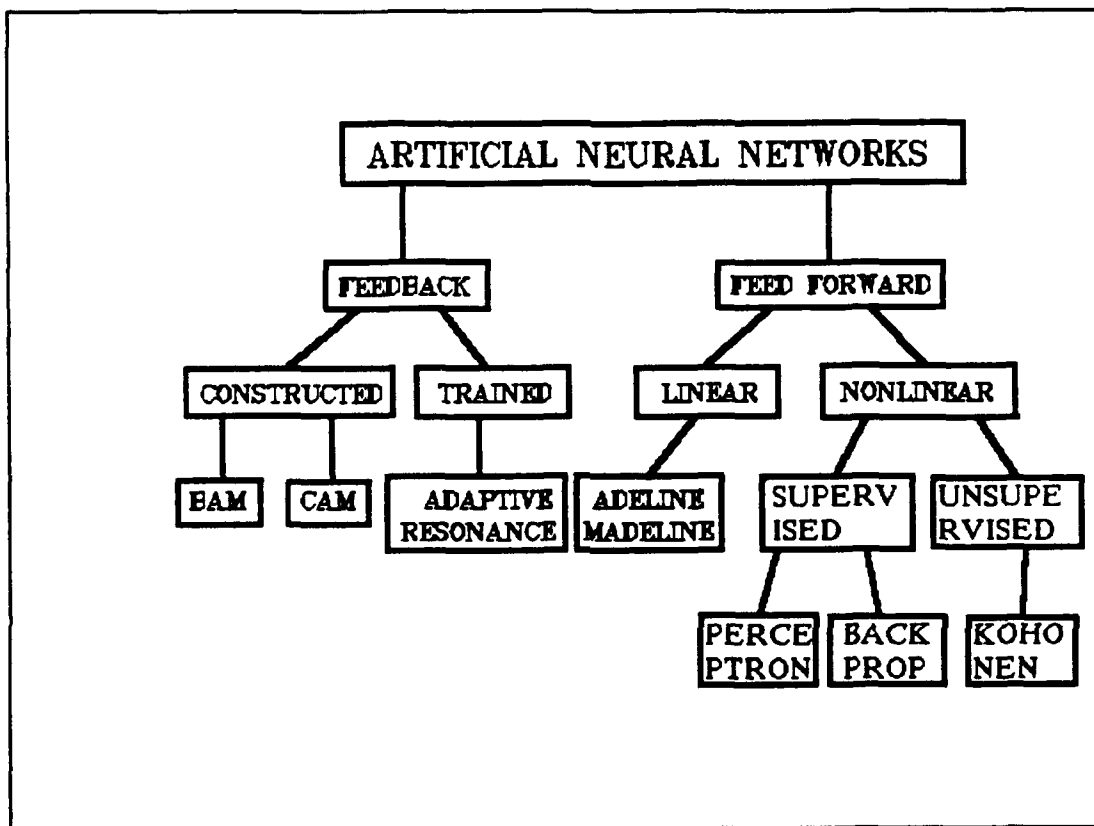


Figure 4. Classification Structure Chart

The classification structure chart of Figure 4 illustrates some of the more classic models [REF 4].

2.7 Bidirectional Associative Memory

The mind remembers things by a series of associations. Attempts have been successful in mimicking this process using Bidirectional Associative Memories, BAM. There are at least two layers of neurons and two groups of weights connected such that the outputs of one layer of neurons is

connected to the inputs of the others (Figure 5).

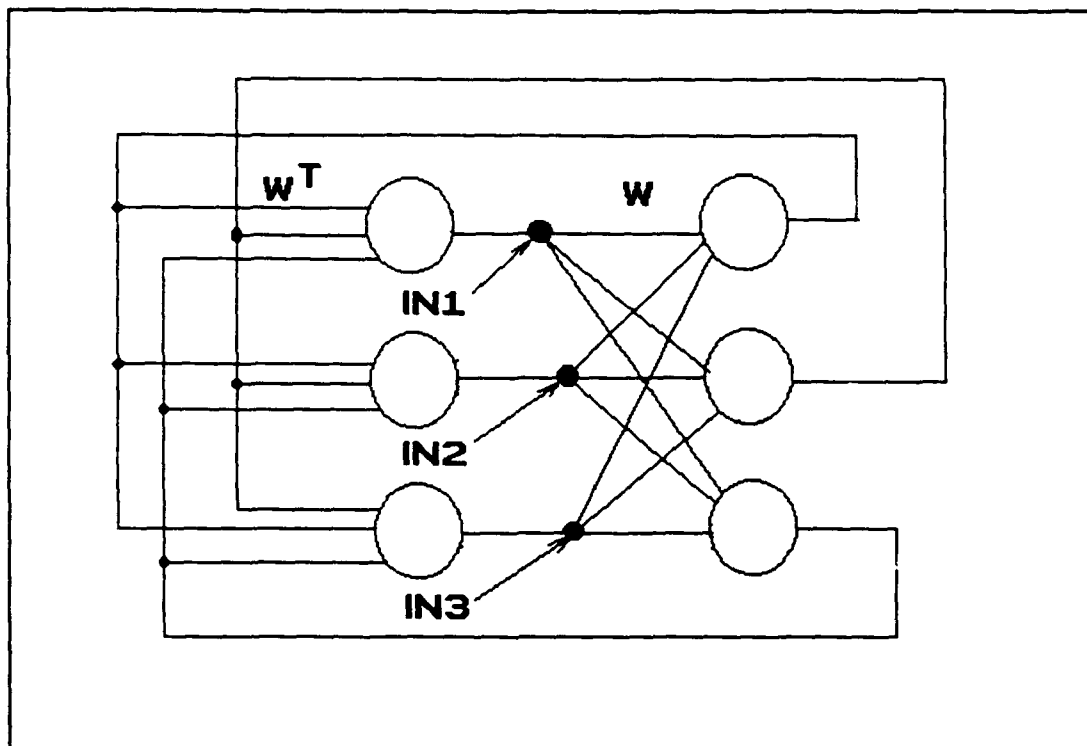


Figure 5. Bidirectional Associative Memory

Inputs are applied to the W matrix for the second layer, whose outputs are applied to the first layer through the transpose W^T of the weight matrix. The output of the first layer should then approximate the learned inputs; the output of the second should approximate the learned outputs [REF 11]. This type network is useful for creating full patterns from corrupted ones. It is trained with pairs of inputs; when it sees the first pattern of the pair, it should recall the second pattern of the pair. Wang has developed a method of insuring recall by generating dummy elements when needed, called dummy augmentation, which is discussed in the noted reference [REF 36].

2.8 Content Addressable Memory

A Hopfield Content-Addressable-Memory, CAM, network is a single large layer of Hopfield neurons with total interconnectivity. The outputs of all neurons are fed back as inputs to all neurons. These networks are used to form associative memories, which recall all of some stored information when given partial data [REF 4]. The networks usually have a binary output (+1 on, -1 off) and the connection weights are also binary (+1, -1). The network iterates until it converges. Hopfield has shown convergence is assured when the weights of two neurons, i and j , are symmetric ($w_{ij}=w_{ji}$). The learning iterative process is given by the following equation [REF 15]:

$$OUT_j(t+1) = NONLINEARITY \left(\sum_{i=0}^{n-1} (w_{ij}) (OUT_i(t)) \right) . \quad (2.4)$$

An interesting application of CAM networks has been in solving Nonpolynomial Hard problems (i.e., problems whose solution convergence rates cannot be described by a polynomial), such as the Traveling Salesman problem [REF 27] and the Map and Graph Coloring Problem [REF 33].

2.9 Adaptive Resonance

Adaptive Resonant Theory (ART) models feed outputs back to inputs in a cyclic fashion. Thus, each successive output better conforms to the input stimulus pattern. ARTs can

thus generalize to extract the ideal from a set of noisy examples. This requires bidirectional elements (e.g., Bidirectional Associative Memory--BAM--models developed by Steven Grossberg in 1982). Software uses a transfer function matrix in the forward direction and the transpose of the transform matrix to go from output back to input. This process is repeated until stability is reached [REF 13,28]. Models of this nature include the neocognitron, which is able to recognize stimulus patterns in the presence of shifting or other distortions [REF 34].

2.10 ADELINe and MADELINE

ADELINe (Adaptive Linear Element) is an adaptive model developed by Bernard Widrow and associates. In this model the basic neuron does not have a nonlinear stage but is otherwise the same as a Hopfield neuron; Widrow refers to this linear implementation as an Adaptive Linear Combiner (ALC). When a threshold element is added as in the Perceptron, the neuron is called an Adaptive Threshold Element (thus the ADELINe can be classified as either linear or nonlinear depending on the absence or presence of the threshold stage). The inputs are often taken from a single waveform at different increments of time (via delay circuits, similar to sampling with an analog to digital converter) to make an adaptive filter (Figure 6). It uses the error from comparing the actual output with the desired output to modify the weights in a direction to

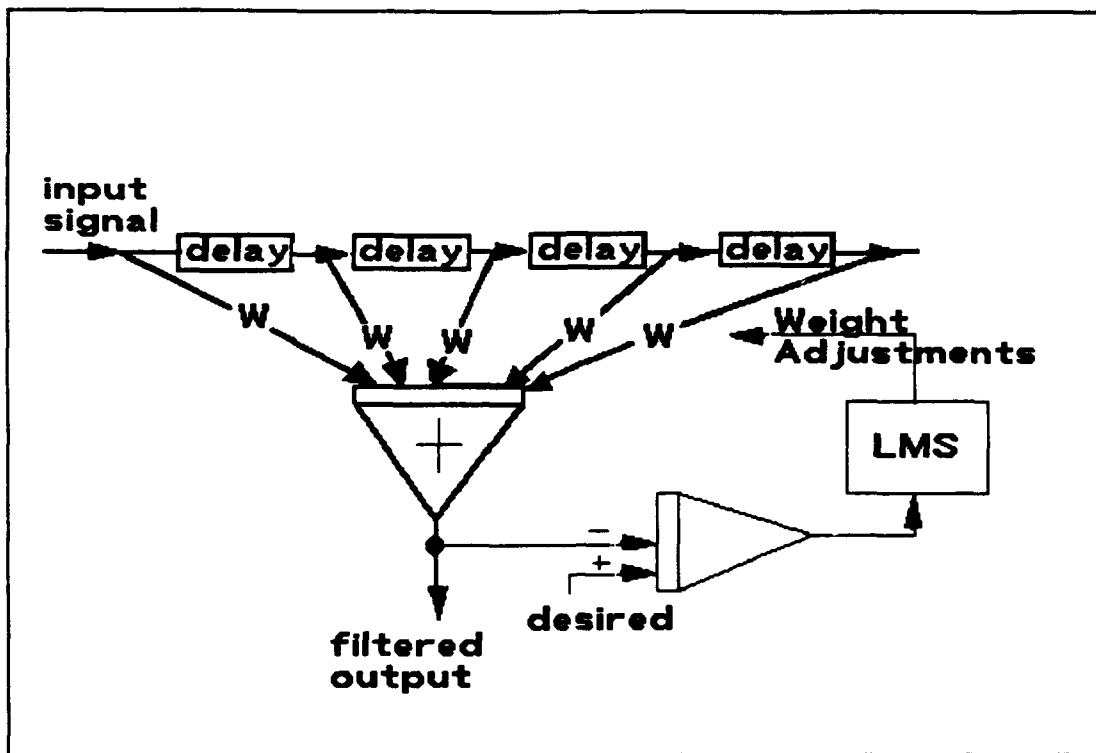


Figure 6. Adaptive Filter

reduce the error. MADELINE is an extension to include many ADELINES. Learning is by the delta rule (or LMS--Least Mean Squares--rule to use Widrow's terminology) that states

$$w_{ij} = B(d_i(t) - a_i(t))X_{ij}(t) \quad , \quad (2.5)$$

where t is time, B is the learning rate, X_{ij} is the j 'th input to neuron i , a_i is neuron i 's activation level (i.e., the output level of neuron i), and w_{ij} is the j 'th input weight to neuron i , and d_i is the desired target output of neuron i . Thus the error from the desired target is used to modify the weights at some learning rate B . The learning rate can be used to control how much effect an individual input pattern has on changes to the network weights.

Smaller values of B cause smaller changes in the weights [REF 13, 14, 30, 35].

2.11 Perceptron

The Perceptron attempts to model the biological sensory model. It was developed by Frank Rosenblatt in 1962. The learning rule, based on Hebb's rule, is that weight changes are proportional to the product of sending and receiving neuron activity levels, but uses a threshold detector instead of the sigmoid function to modify the sum of weighted inputs. When the threshold is exceeded, the output changes in a binary manner from the ambient level to the active level [REF 24].

2.11.1 Limitations of a Single Layer

The limitations of what could be learned by such networks were explored by Marvin Minsky and Seymour Papert in 1969. They mathematically showed that linear neurons were very limited in their problem solving ability; their research proved a major setback to further neural network research for several years [REF 4]. Some functions cannot be replicated in a single layer model (e. g. the EXCLUSIVE-OR, XOR). A perceptron's threshold function adds a nonlinearity which separates the space by a line, plane, or hyperplane (depending on the number of inputs) into two regions. For the XOR problem, space is two-dimensional and is separated into two areas by a line defined by the linear

portion and the threshold level:

$$X_1W_1 + X_2W_2 = C \quad . \quad (2.6)$$

There is no way to define a line that will separate the one output area from the zero output area for the EXCLUSIVE-OR

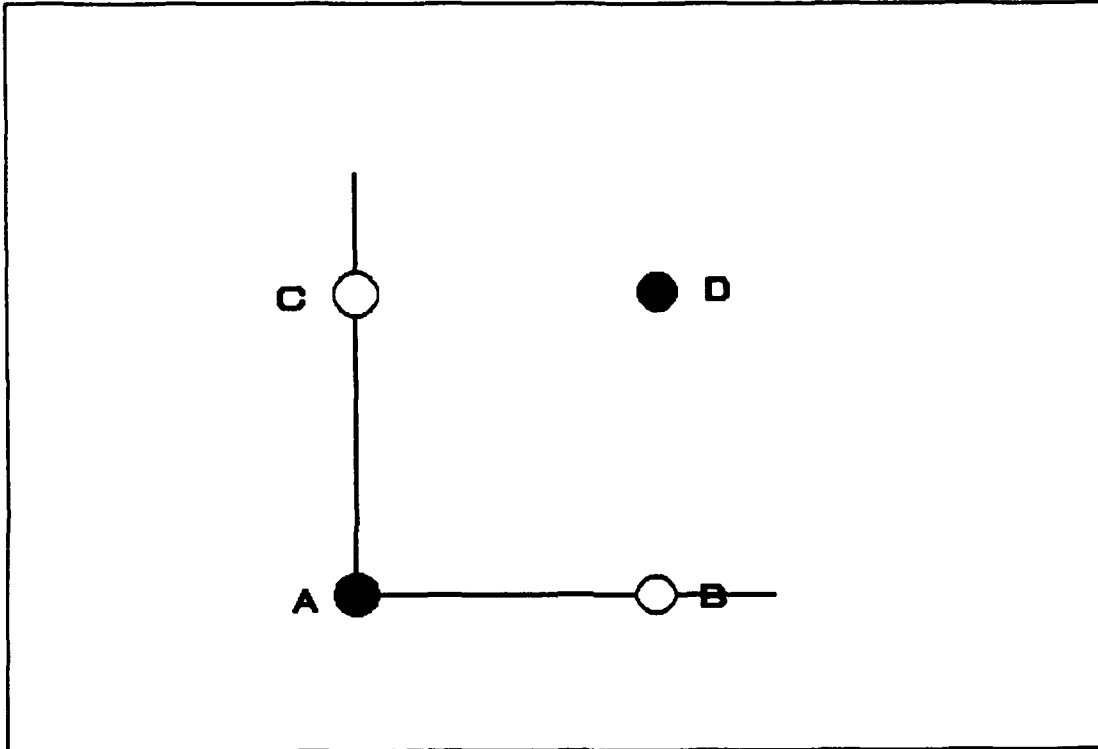


Figure 7. XOR Problem

function. This is obvious by examining Figure 7, where the patterns are A=00, B=01, C=10, and D=11. A and D must fall on one side of the line and B and C on the other.

2.11.2 Hidden Layers

To solve general problems, the idea of hidden layers was introduced. So there is an input layer and an output layer and one or more in between layers. Layers offer the ability to do recurrent simulations, such as shift registers

[REF 9]. Layered networks can be designed to be invariant to rotation, translation and scaling of patterns [REF 31].

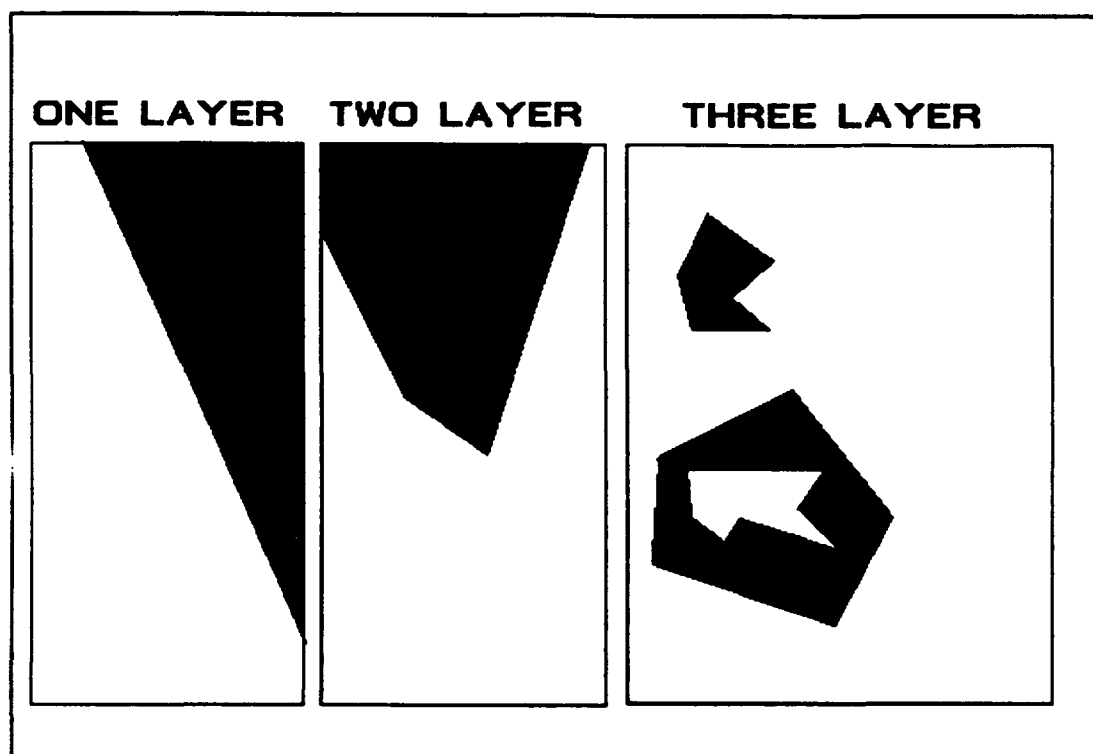


Figure 8. Multiple Layers

Figure 8 shows how adding additional layers of neurons allows formation of convex separation regions. A single layer of neurons can only divide two regions with a line [REF 24]; it can perform any of the sixteen basic 2-input logic forms except EXCLUSIVE-OR and EXCLUSIVE-NOR [REF 11]. Adding a second layer allows taking the intersection of the half-planes from the first layer to form a convex open or closed region of decision. The third layer allows several hypercubes from the second layer to be combined so that any arbitrary decision region may be created; it may be thought of as the OR function [REF 22].

2.12 Back Propagation

The back propagation model uses hidden layers and expands the delta rule to propagate the difference between

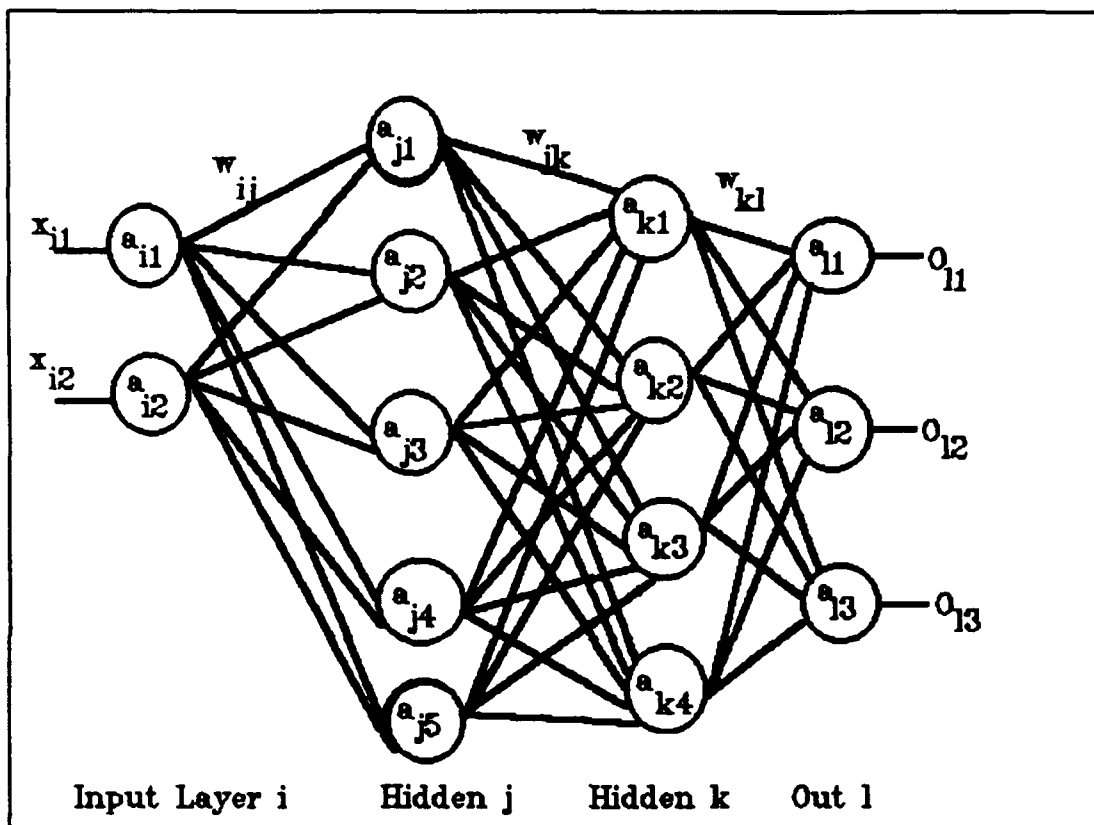


Figure 9. Back Propagation Model

the actual and desired outputs back through the additional layers. The generalized back propagation model [REF 1] is shown in Figure 9. The a 's represent the activation levels of a neuron and the w 's represent the input weights.

The activation and learning equations for the output layer 1 are

$$a_{l1} = \left(\sum_{r=1}^{num_k} (w_{rs}) (o_{rs}) \right) + bias_s, \quad (2.7)$$

$$O_{ls} = F(a_{ls}) \quad , \quad (2.8)$$

$$e_{ls} = F'(a_{ls}) (d_{ls} - O_{ls}) \quad , \quad (2.9)$$

$$\Delta w_{rs} = (B) (O_{ks}) (e_{ls}) \quad , \quad (2.10)$$

and

$$\Delta bias_{ls} = (B) (e_{ls}) \quad , \quad (2.11)$$

where B is the learning rate, bias is a constant added to NET (the sum of the weights times the inputs), F is the nonlinearity function, F' is the derivative of the nonlinearity function, a is the activation output level prior to the nonlinearity, r is the neuron index in the k layer, s is the neuron index in the l layer, num is the number of neurons in the layer, d is the desired target output, o is the output level after the nonlinearity, x is the input to the network, and

$$e_{ls} = \frac{-\partial (d_{ls} - O_{ls})}{\partial a_{ls}} \quad . \quad (2.12)$$

The equations for the hidden layers are similar with the indices changed to correspond to the relative layers except that equation (2.9) is modified, for hidden layer k, to

$$e_{kr} = F'(a_{kr}) \left(\sum_{s=1}^{num_l} (w_{rs}) (e_{ls}) \right) \quad (2.13)$$

and similarly (with index changes) for hidden layer j [REF 1].

2.12.1 Derivation of Back Propagation Equations

In the following paragraphs, the derivation of the back propagation equations will be presented. Although the error terms do not actually propagate back through the network, it is convenient to think of back propagation as a three step cycle: (1) a forward pass of the inputs through the network, (2) a backward pass of the error terms through the network, and (3) an updating of the weight and bias terms.

The forward pass applies a pattern (p) of inputs (X_{ip} 's), to the first hidden layer of neurons through weights (W_{ip}). The equation for output of the summer for one neuron is

$$NET_p = \sum_{i=1}^{num_{in}} X_{ip} W_{ip} \quad . \quad (2.14)$$

The output of the neuron, F , is a function of NET . For the sigmoidal activation function the output of one neuron is

$$OUT_p = F(NET_p) = \frac{1}{1 + e^{-NET_p}} \quad . \quad (2.15)$$

The outputs from the first hidden layer of neurons become inputs to the second neuron layer and so on. The outputs of the final, output layer are compared with the desired outputs and an error term is developed,

$$E_{jp} = t_{jp} - o_{jp} \quad . \quad (2.16)$$

Then the error from a particular pattern is effectively propagated backwards, through the neuron, to

adjust the weights of the neuron in a direction so as to reduce the error. The error at each input is taken as the error of the previous stage's output and is then propagated back through all hidden layers in turn. The method used to compensate for the error is least-squares, whereby, the sum of the squares of the error terms is minimized. A multiple of 1/2 is used to simplify the math and the error term for pattern p across all outputs (j) of a layer is

$$E_p = \frac{1}{2} \sum_{j=1}^{num_{out}} (t_{jp} - o_{jp})^2 \quad . \quad (2.17)$$

The partial of error with respect to weight is used to determine the amount to modify a weight to better model the current pattern:

$$\frac{\partial E_p}{\partial w_{ijp}} = \frac{\partial E_p}{\partial NET_{jp}} \frac{\partial NET_{jp}}{\partial w_{ijp}} = \frac{\partial E_p}{\partial o_{jp}} \frac{\partial o_{jp}}{\partial NET_{jp}} \frac{\partial NET_{jp}}{\partial w_{ijp}} \quad . \quad (2.18)$$

Since the output is $F(NET)$,

$$\frac{\partial o_{jp}}{\partial NET_{jp}} = F'(NET) \quad . \quad (2.19)$$

Since error is defined in terms of output,

$$\frac{\partial E_p}{\partial o_{jp}} = \frac{1}{2} (2) (t_{jp} - o_{jp}) (-1) = -(t_{jp} - o_{jp}) \quad . \quad (2.20)$$

The equation for NET is

$$NET_{jp} = \sum_{k=1}^{num_{in}} W_{kjp} O_{kp} \quad , \quad (2.21)$$

where the summation goes from k equals 1 to the number of neurons in the k layer. So,

$$\frac{\partial NET_{jp}}{\partial W_{ijp}} = \frac{\partial \sum_{k=1}^{num_{in}} W_{kjp} O_{kp}}{\partial W_{ijp}} = O_{ip} \quad , \quad (2.22)$$

since the only element in the summation with a nonzero partial is $W_{ip} O_{ip}$. Combining the three parts yields

$$\frac{\partial E_p}{\partial W_{ijp}} = - (t_{jp} - O_{jp}) F'(NET_{jp}) O_{ip} \quad . \quad (2.23)$$

2.12.2 Derivative of Sigmoidal Function

For the sigmoidal activation function $F(NET)$, $F'(NET)$ is easily calculated, since

$$F'(NET) = \frac{\partial (1 + e^{-NET})^{-1}}{\partial NET} = -1 (1 + e^{-NET})^{-2} (-e^{-NET}) \quad , \quad (2.24)$$

$$F'(NET) = \frac{1}{(1 + e^{-NET})} \frac{e^{-NET}}{(1 + e^{-NET})} \quad , \quad (2.25)$$

or

$$F'(NET) = \frac{1}{(1 + e^{-NET})} \left(1 - \frac{1}{(1 + e^{-NET})} \right) \quad . \quad (2.26)$$

Thus,

$$F'(NET) = (F(NET)) (1 - F(NET)) = OUT(1 - OUT) \quad . \quad (2.27)$$

2.12.3 Methods of Accelerating Learning

Although a trained back propagation neural network responds nearly instantaneously to a change in input, the time required to train the weights for a particular problem may be quite lengthy consisting of thousands of iterations of forward passes, backwards passes, and weight adjustments. The number of iterations required to converge to a solution exhibiting a predetermined error rate may be reduced by accelerating techniques.

One technique, called momentum, remembers the previous weight change and applies a portion of its value to the next calculated weight change. This tends to force the weights to follow the narrow gullies in a deep error surface seeking a global minimum. The formula is

$$\Delta W_{ij}(n+1) = \Delta W_{ij}(n) + \alpha \Delta W_{ij}(n) \quad . \quad (2.28)$$

A similar method proposed by Sejnowsky and Rosenberg is

$$W_{ij}(n+1) = W_{ij}(n) + B(\Delta W_{ij}[n]) + \alpha (\Delta W_{ij}[n+1]) \quad . \quad (2.29)$$

In both cases a portion of the old delta weight adds to a portion of the newly calculated delta weight to produce a new delta weight [REF 11].

Other methods for improving acceleration involve using

a different activation function, $F(NET)$. One method proposed by Parker, called second-order back propagation, uses second-order derivatives to compute a more accurate estimate of the proper weight change [REF 32]. Stornetta and Huberman proposed replacing the sigmoidal function with

$$F(NET) = -\frac{1}{2} + \frac{1}{1 + e^{-NET}} \quad (2.30)$$

and scaling the inputs between -0.5 and $+0.5$, i.e., centering the transitions about zero. This causes greater magnitudes of weight modifications for zero output patterns, and speeds convergence by thirty to fifty percent [REF 11].

2.12.4 Abstraction

An important aspect of neural networks is their ability to form an abstract solution to a problem. That is, the network should not only perform well on classifying the patterns used to train the network but also on a new set of patterns the network has never seen. The two sets are referred to as a training set and a test set in this paper. The test set of patterns should have similar identifying characteristics to the training set; abstraction is measured by how well the network performs on the test set.

The training set may be classified with a much lower percent of error than the test set. This loss of abstraction is referred to as overtraining, as the network has learned the test set too well. Overtraining can result from several conditions: (1) too many neurons in the hidden

layer(s), (2) too many hidden layers, and (3) too many training iterations [REF 1,2].

2.12.5 Initialization

If all weights are initialized to the same value, e.g. zero, and the solution requires the development of unequal weights, the network will never learn. The error propagated backwards is proportional to the weights and all hidden layer outputs connected to the output layer will receive the same error signal and make the same adjustment. If the weights are initialized too large, the nonlinearities may saturate and clip the outputs. To avoid this dilemma, it is best to initialize the weights to small, random values [REF 12]. Since the proper distribution of weights is not known until the neural network has learned the problem, a uniform distribution is usually used for initialization. A topic for further study would be if Gaussian, Rayleigh, or some other distribution would be a better choice for most problems.

2.13 Kohonen

Kohonen's feature map combines an input layer with a competitive layer and is trained by unsupervised learning. It classifies patterns into a graphical organization of pattern relationships: the second layer being a n -dimensional hypercube (where n is the number of entries in the input pattern). The weights u_{ij} are initialized to a

uniform distribution of small random numbers (typically between 0 and 1). Error terms between the inputs, e_{ij} , and weights are computed for each competitive layer neuron by summing differences squared; that is,

$$ERROR_i = \sqrt{\sum_{j=1}^{num_{in}} (e_j - u_{ij})^2} \quad . \quad (2.31)$$

The neuron with the lowest error is declared the winner (Figure 10).

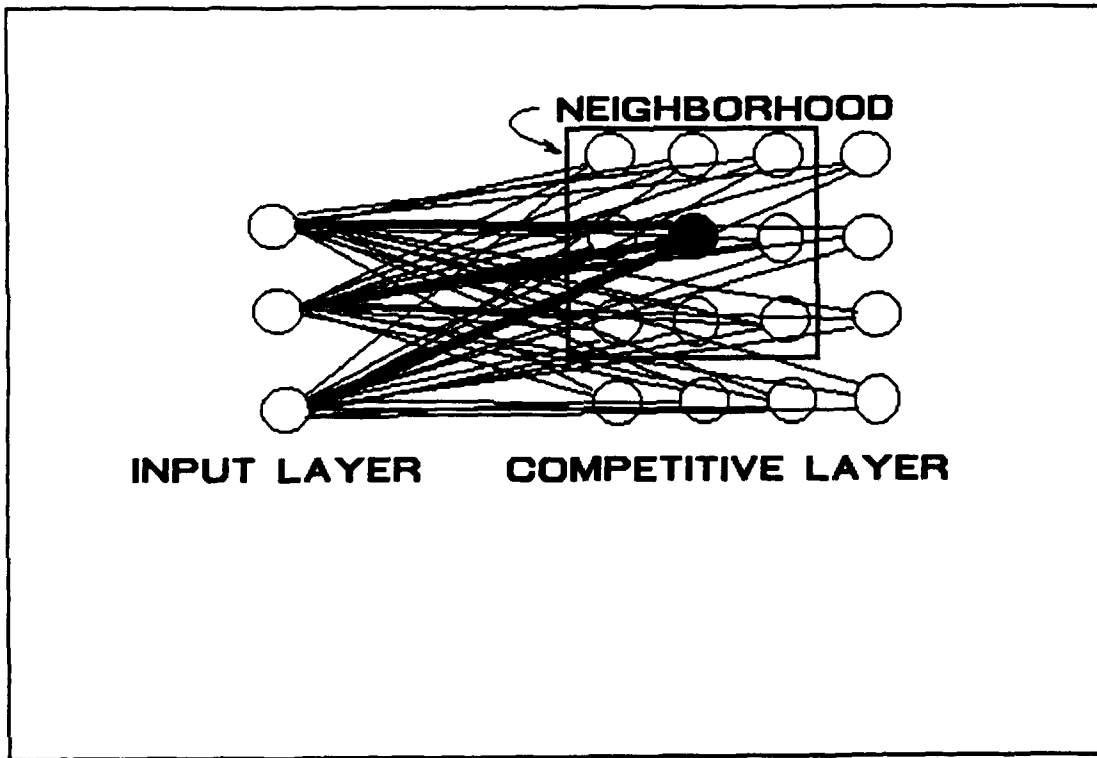


Figure 10. Kohonen Feature Map

Then the winning neuron and all neurons in its neighborhood have their weights updated by

$$\Delta u_{ij} = B(e_j - u_{ij}) \quad . \quad (2.32)$$

The value of B , the learning rate, is initially set to a

small value, such as .35, and decreased as training proceeds by the equation

$$B[n] = (B[n-1]) \left(1 - \frac{n}{T}\right) , \quad (2.33)$$

where n is the current iteration and T is the total number of iterations to be run. The size of the neighborhood is initially large and is decreased by the same relationship as training proceeds [REF 26]. Kohonen learning only works well when patterns do not overlap; if this is not the case, the weight vectors tend to get stuck in isolated regions [REF 29].

2.14 Combined Networks

Some success has been attained in combining networks, such as, a nonsupervised network followed by a supervised one. One of the most common combinations is called counterpropagation, wherein, a Kohonen layer is followed by a Grossberg Outstar layer (Hopfield neurons). The Kohonen layer extracts the statistical properties of the problem and the Grossberg layer maps these properties to the desired outputs. This combination forms a good statistical model but is inferior to back propagation (which has greater accuracy) for most mapping network applications [REF 11, 29].

CHAPTER III

HYDROGRAPHY

Travel by water has provided man with an economical way to exchange commodities with distant neighbors. It has allowed man to venture to foreign regions in search of adventure and knowledge. To sail farther and in more comfort carrying large cargos, huge ships with deep drafts were constructed. Ship wrecks pointed to the need to detect submerged objects and the depth of the bottom.

3.1 Acoustic Signals

Acoustic signals have been used by sailors for years to determine the relative position of objects with respect to current location. The equipment used is called sonar, an acronym for "sound navigation and ranging," which is not capitalized by convention. Sonar may be passive (whereby noises, such as, boat propellers are detected) or active. Active sonars generate a low frequency signal (generally below 3 kHz) that reflects off objects and provides an echo signal for analysis (Figure 11).

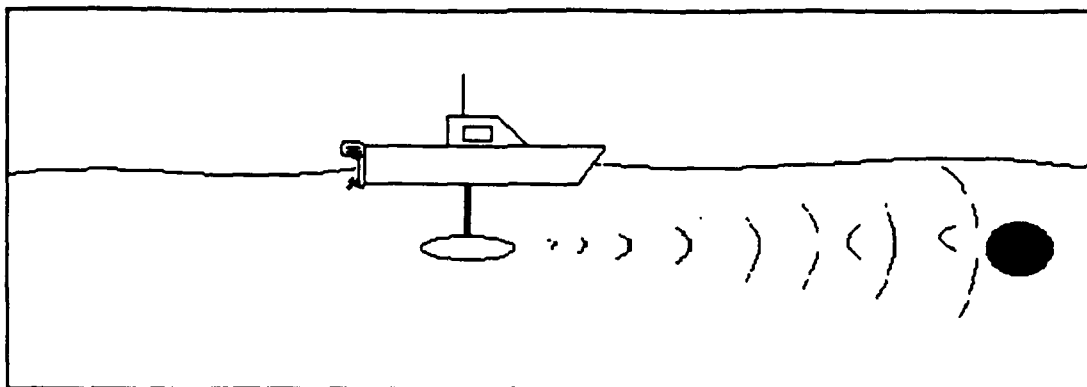


Figure 11. Active Sonar

The transmitter is sometimes referred to as a projector and the receiver is sometimes called a hydrophone. Either one separately or both combined in a single unit are referred to as a transducer, since electrical energy is converted to a pressure wave, i.e., a sound wave. Electromagnetic waves are quickly attenuated when traveling through water, so pressure waves are the most suitable media for information transmission. Transmitters are often electromagnetically driven vibrating pistons which generate about 3 kilowatts of sound; hydrophones often use magnetostrictive materials (e.g., ammonium dihydrogen phosphate) to detect pressure changes [REF 11,37].

3.2 Depth Sounders

Devices similar to sonars, called depth sounders, are used to measure the distance from the water surface to the bottom of the water basin. In a conventional depth sounder, sound is emitted from a point source transmitter, strikes a

surface and is partially reflected and partially absorbed (Figure 12).

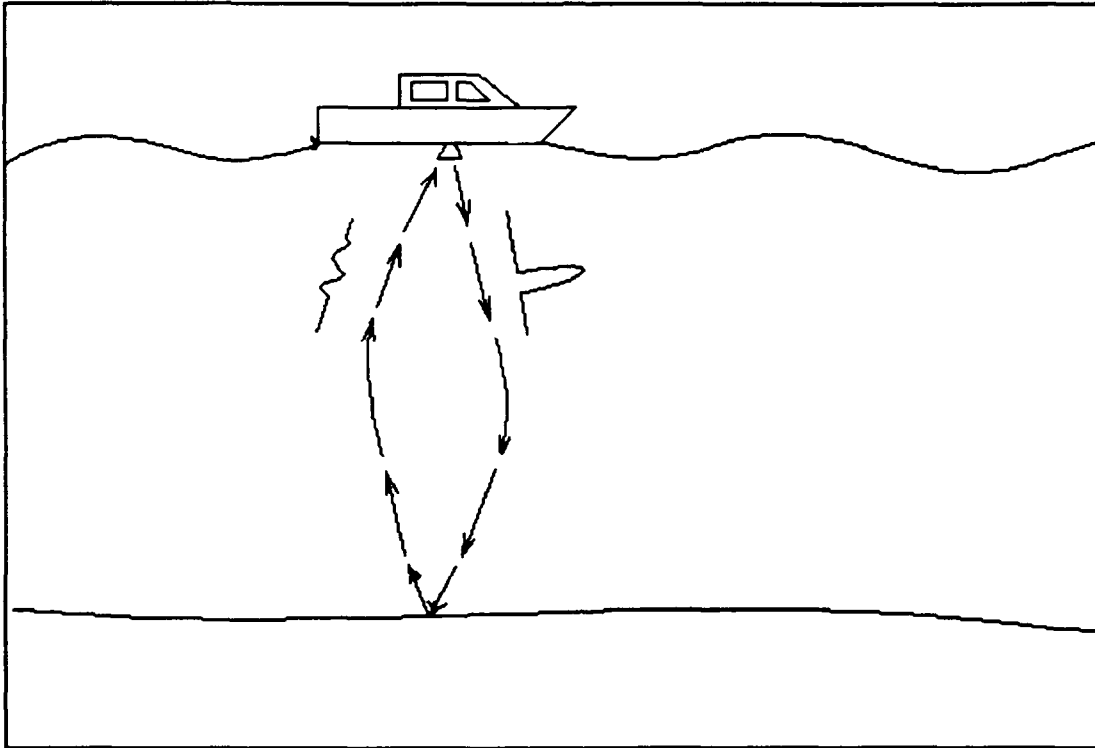


Figure 12. Depth Sounding

Some of the reflected energy returns to the source and is analyzed. The receiver transducer uses a magnetostrictive material, e.g., lead zirconate titanate [REF 38]. The transmitter element generates about 500 watts of sound using a gated oscillator (to reduce the tapering effect at the start and end of the pulse). The transit time of this pulse is determined by the depth and the velocity of sound.

3.2.1 Velocity of Sound

In hydrographic surveying the transit time is used to determine depth, i.e.,

$$depth = \frac{time_{transit}}{2} (velocity_{sound}) \quad . \quad (3.1)$$

Sound velocity varies according to density, salinity, and temperature. Fluids have a shear modulus of zero and thus no transverse wave is transmitted. The speed of the longitudinal (compressional) wave is

$$v = \sqrt{\frac{M_B}{\rho}} \quad , \quad (3.2)$$

where v is the velocity, M_B is the adiabatic bulk modulus, and ρ is the density; it is on the order of 1500 m/s for sea water or 1461 m/s for fresh water [REF 40]. The velocity of sound may be measured directly by an instrument called a velocity profiler or may be calibrated by using a bar suspended at a known depth from the transducer. The time from the transmission of the pulse to the first amplitude peak of the return is the transit time used in depth calculations [REF 17].

3.3 Navigable Bottom

Depth sounders are normally used in conjunction with positioning systems so that maps of the bottom may be made for navigational purposes. Navigation requires unobstructed water-filled channels. The definition of what constitutes a channel is somewhat dependent on the vessel. However, the majority of the world's vessels can navigate easily through suspended sediment. The limiting factor is the density

level of the suspended sediment that causes the pilot to lose control of the vessel's direction [REF 10]. The exact definition of the density is still under discussion, but this density will probably be set somewhere between 1.1 and 1.2 grams/cm³ [REF 6].

3.4 Dredging

To aid navigation, channels are established and maintained on often traveled routes. These channels must be maintained at project depth along their entire course. When an area becomes too shallow, material is removed by dredging to restore proper channel depth.

The type of material--sand, gravel, mud, suspended material, etc.--is of much interest to the dredger. Most dredges work on the principle of a cutter head (similar to a giant eggbeater) followed by a hydraulic vacuum (cleaner). The cutter does well in sand but has problems in gooey mud or rock. Surveys are conducted prior to dredging and afterward so that the amount of material removed from the channel may be calculated for payment purposes. Payment rate per volume is based partially on the type and the density of material removed.

3.5 Material Type Determined by Bottom Sampling

The most common and most reliable method of determining the material type and density composing the bottom is physical sampling. Therefore, bottom material samples are

typically collected in containers and taken to a laboratory for analysis. One type of sample is a surface sample, normally collected with a grabber type sampler (Figure 13).

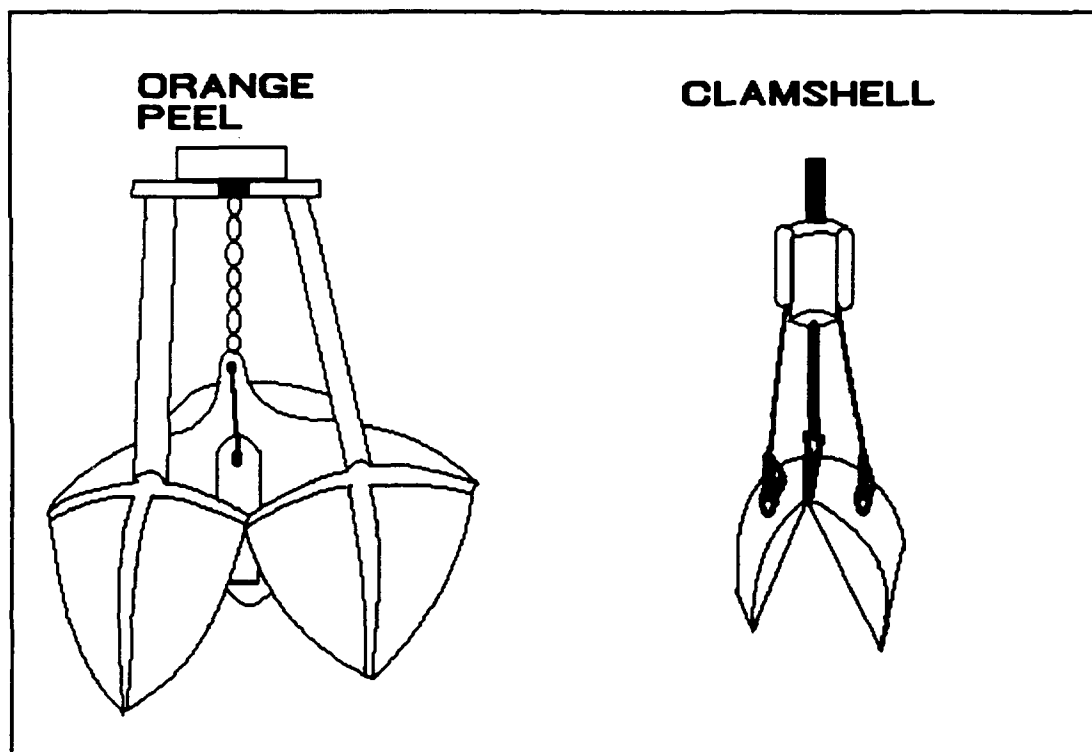


Figure 13. Grab Samplers

This type sampler disturbs the sample and does not delineate by depth of material. To get samples at subsurface depths cores are taken with a pipe. Two common methods are the Phleger or piston sampler of Figure 14 which is good to about 4 feet and the driven pipe type sample, which may go 30 or more feet into the bottom [REF 39].

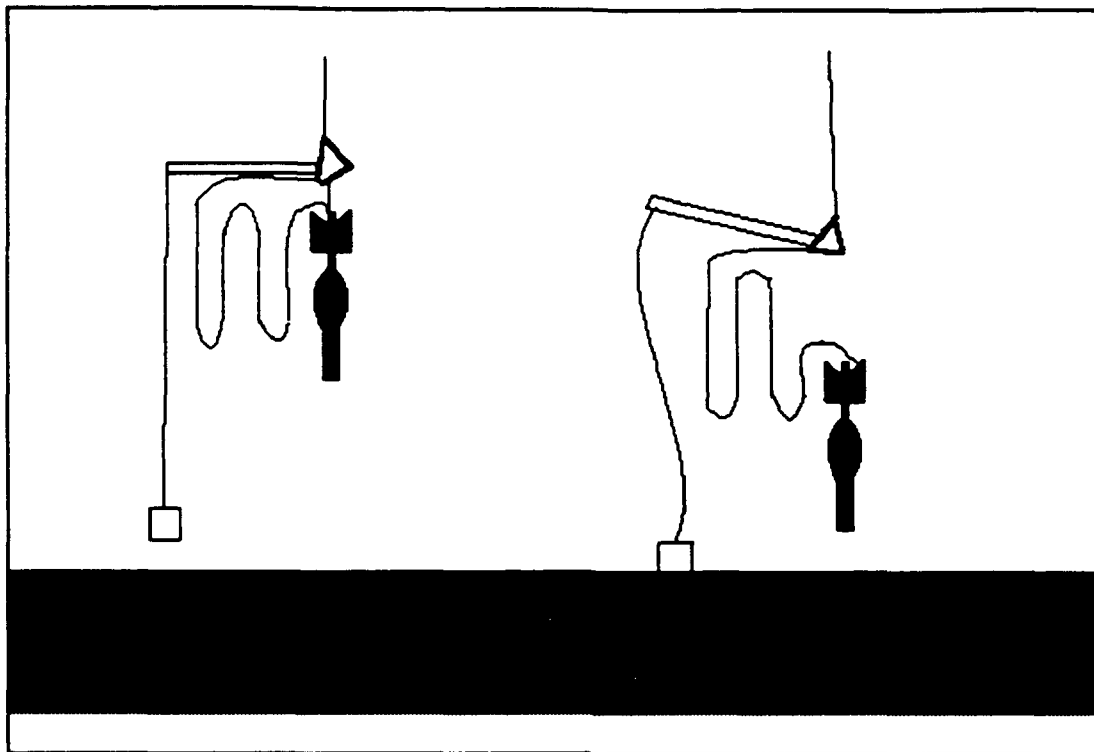


Figure 14. Phleger Corer

3.6 Density Measured with Nuclear Probes

Currently, the only instruments which are capable of measuring density directly are nuclear density probes [REF 5]. Since these contain nuclear capsules, they require special care and numerous permits making them unacceptable for general use in the United States. A typical design is shown in Figure 15. A nuclear density probe consists of two parts: (1) an electronic module housing the battery pack, processor, and mode switches and (2) a stainless steel shaft 1 meter long and 27 millimeters in diameter, containing 8 milliCurie of Cesium-137 as the radioactive source, a gamma ray (photon) detector, and an electronic amplifier. The

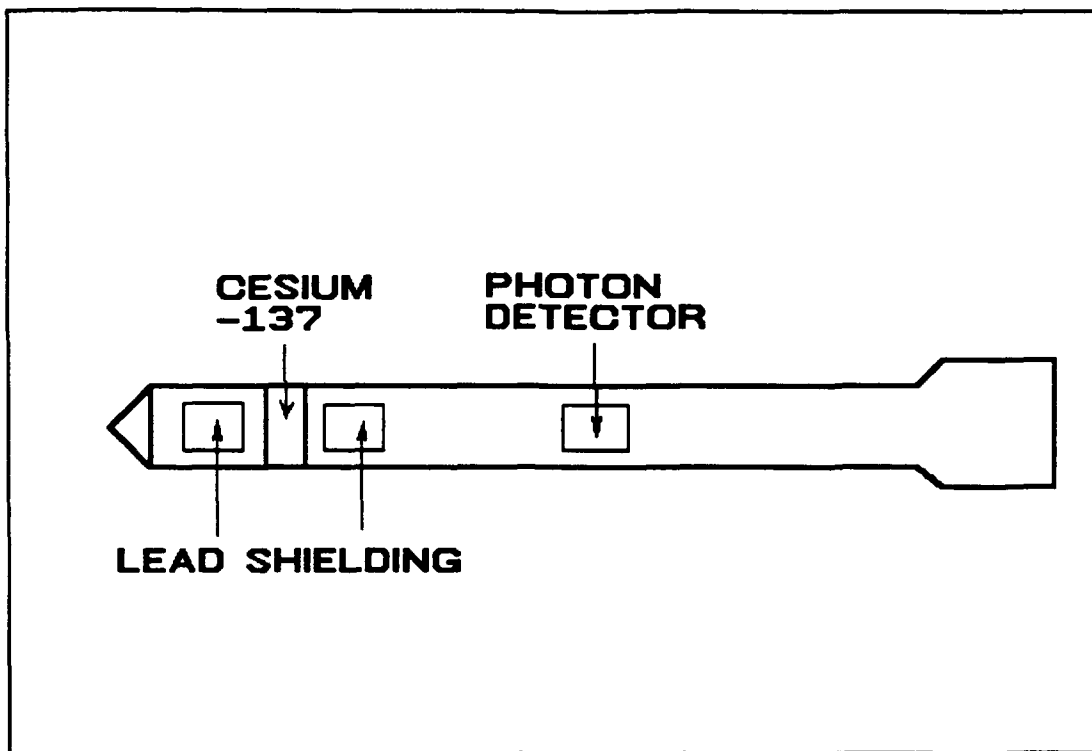


Figure 15. Nuclear Density Probe

instrument is calibrated for the 800 to 2200 kg/m³ range. The backscatter geometry's relationship to count rate and density is given by

$$RATE_{count} = Ae^{-BXD} - C \quad , \quad (3.3)$$

where A, B, and C are constants and D is the density [REF 41].

3.7 Acoustic Modeling

Material classification and density determination could be greatly enhanced by improvements to current depth sounder design. There is an enormous amount of information in the returned signal which is not used in conventional depth sounders due to lack of real-time tools for interpreting the

information. The shape and frequency content of the signal are usually ignored. Research has shown that the shape of the envelope and frequency content of the return are related to the density and material type of the surface causing the return [REF 18]. The amount of returned energy from a particular density of material is related to the frequency of the incident wave. Low frequencies penetrate better than high ones and thus less energy is returned [REF 16].

The most commonly used frequency is in the range of 200-220 kHz. This frequency is used because it works well on hard sand bottoms found in most of the United States and requires a small transducer (3-4 inches in diameter). The reason a small transducer is desirable is that it transmits a narrow (2-8 degree) beam angle. The smaller the beam angle the smaller the beam's footprint on the bottom (also the shallower the depth the smaller the footprint). A small footprint is important because the first return is usually the one conventional depth sounders use for determining depth; therefore, the highest peak within the footprint is chosen as the depth. Thus, a wide footprint reduces resolution by stretching peaks across a larger area than they truly occupy. Figure 16 shows the effect of a 40 degree beam angle. The closest point to the transducer that is covered by the beam is used to determine depth. When the bottom is rough, the effects of wide beam angle have an extremely adverse effect on mapping.

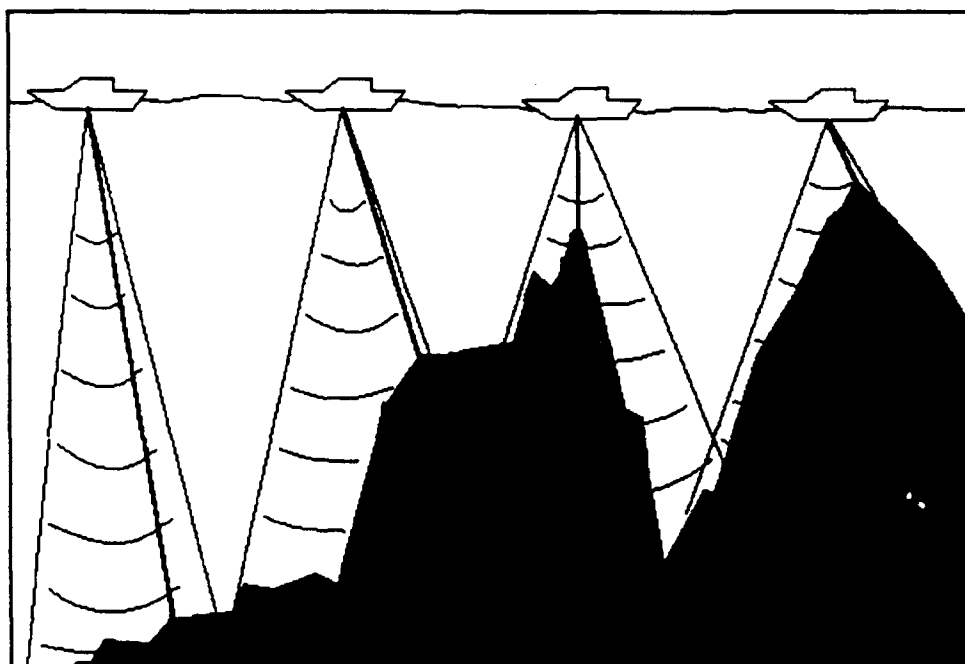


Figure 16. Returns from Wide Beam

Figure 17 shows the bottom as measured by the depth sounder

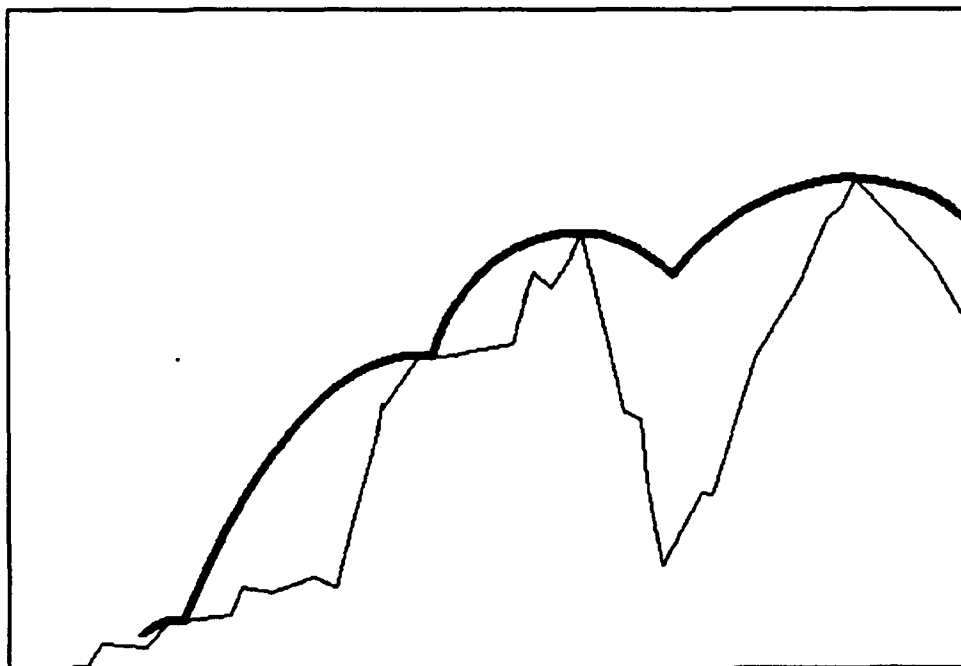


Figure 17. Depth Sounder's Reading

superimposed over the actual bottom. Also low frequency

transducers are much larger (about a foot across) and require a larger vessel and more elaborate mounting. When possible, dual frequency transponders are used (with a 200 kHz transponder used mostly and a 20 kHz used in suspended sediment areas). The 20 kHz unit has a much larger footprint (20-40 degree beam angle) and gives poor resolution of channel side slopes; however, it has the desirable effect of penetrating suspended sediment and measuring the "true" depth [REF 17, 38]. Lower frequencies, 1 to 7 kHz, are used for subbottom profiling, to penetrate 30 feet or so into the bottom itself with echoes at each interface between material types.

There are 2 distinct approaches to acoustic modeling: theoretical and statistical. For theoretical modeling certain assumptions are made and a mathematical model is devised. For statistical modeling a multivariate model is devised using coefficients determined by the maximum-likelihood principle [REF 42].

3.7.1 Theoretical Modeling

The bottom of most waterways consists of a water-saturated porous media. For small stresses (such as, sound waves) water-saturated media respond elastically with attenuation (i.e., they are viscoelastic) [REF 43].

In saturated media, porosity is the volume of voids (pore space) occupied by water ratioed to the total volume of the media. The density of a media has two components:

mineral grains and water. If porosity is represented by eta and density by rho (subscripted by w for water and s for solids), the equation is

$$\rho_{media} = \eta \rho_w + (1 - \eta) \rho_s \quad (3.4)$$

for gas-free media. The characteristic impedance of a media to the transmission of acoustic waves is density (rho) times compressional velocity (V_p) or

$$Impedance = Z = \rho V_p \quad (3.5)$$

The characteristic impedance, Z , determines the amount of sound energy reflected at the interface of two media; thus, depth sounders trigger on an impedance mismatch. The Rayleigh reflection coefficient, R , may be expressed as

$$R = \frac{Z_2 - Z_1}{Z_2 + Z_1} = \frac{\rho_2 V_2 - \rho_1 V_1}{\rho_2 V_2 + \rho_1 V_1} \quad (3.6)$$

where subscript 1 refers to the first medium and subscript 2 to the second [REF 44].

A number of elegant acoustic models have been developed but most have made invalid simplifying assumptions about saturated media that have negated their usefulness. They treat mineral particles as spheres (which they are not) and they ignore shear waves (which are transmitted). It is beyond the scope of this dissertation to examine the different models; however, many of the newer ones are based on the equation (for compressional and shear waves),

$$\frac{1}{Q} = \frac{aV}{\pi f - \frac{a^2 V^2}{4\pi f}} , \quad (3.7)$$

where $1/Q$ is the specific attenuation factor, a is the attenuation coefficient, V is the wave velocity, and f is the frequency. Hamilton believes that internal friction (due to intergrain movements) is the dominant attenuation factor in water-saturated media [REF 45]. Values which have been obtained by physical measurements are usually used in models for particular material types (density, porosity, velocity, and attenuation). More recent models, proposed by Turgut and Yamamoto [REF 49], apply Biot theory (compressional waves are excited and there is mode conversion at the interfaces). These models use spectral ratio calculations to model material interfaces (e.g., fluid-sediment, sediment-fluid, and sediment-sediment).

3.7.2 Statistical Modeling

Statistical models ignore the physical significance of the parameters and simply determine the numerical values of the coefficients (betas). An example model for velocity, V , is

$$V = \beta_0 + \beta_1 \rho + \beta_2 \rho^2 + \beta_3 D + \beta_4 D^2 + \beta_5 \phi + \beta_6 \phi^2 + \beta_7 D \rho_2 + \beta_8 \rho_2^2 , \quad (3.8)$$

where ρ is the bulk density, ρ_2 is the grain density, D is the median diameter of grain size, and ϕ is the grain size deviation. Regression equations are fitted to empirical data to determine the beta's [REF 42].

3.7.3 Noise

How to model noise in water basins is a problem with all modeling efforts. If noise is characterized by a known probability density function (e.g., Gaussian), there is a large body of statistical information available.

Unfortunately, when the noise becomes non-Gaussian (as it is in this case) the problem is greatly compounded.

Nondeterministic noise sources include the following: ice break-up, drilling, boats, atmospheric disturbances, and fish. One approach involves using Bayesian methods (assigning priors to the unknown parameters of the noise power distribution function); however, these require multidimensional integration which is not generally practical [REF 45, 46, 47].

3.7.4 Signal Abstraction

Most models start with idealized mathematical models and make heuristic modifications. Another approach is to view a signal at a higher level of abstraction than just a function of one or more independent variables. Signal abstraction views a signal as a group of conceptual entities, such as, peaks, valleys, wavelets of various shapes, etc. [REF 50]. Using this approach, features of the signal can be used for analysis without concern for the underlying phenomena generating the features. This approach is explored in this research.

CHAPTER IV

SOFTWARE

Hardware neural networks are parallel processors that deal with all the inputs simultaneously. However, simulation software may operate in a sequential manner by effectively freezing time while each input is stepped through sequentially. When this software is run on a computer with parallel processing capabilities (such as, the Cray Y-MP 8/6128 used for running the BACKPROP program described below) a high degree of vectorization is possible and the computer is able to process the inputs concurrently.

The first step in the research was to develop some software tools for conducting the research. Commercial programs for neural network implementation were available, but were "black box" in nature and did not allow control of many network parameters or monitoring of the values of weights determined for the network. Thus, programs were written in FORTRAN to implement the networks used in the research in software. Software is easier to modify than hardware and is the most practical initial implementation methodology of any neural network design.

The dangers of using virgin software to venture into uncharted regions of acoustic signal processing were not

treated lightly. The neural network software developed in conjunction with this research was thoroughly tested. Test cases were used as inputs after all coding changes and the outputs of the code-under-test was compared with the outputs of commercial software with any differences resolved before proceeding.

4.1 Code Verification

A program, called LETTERBP, was developed to provide character patterns suitable for the tests. The patterns were output in two formats: (1) compatible with the research neural network programs and (2) compatible with a commercial program. LETTERBP is discussed in detail in Appendix B. A few of its salient features are (1) it displays a 7x9 checkerboard on the screen, (2) the arrow keys are used to move around to different nib positions, (3) the current nib may be toggled on or off, (4) an output neuron is assigned to the pattern, and (5) patterns may be of either the learning or test type. Numerous learning and test case patterns are specified and saved for later input into neural network programs. In test cases using BACKPROP with one hidden layer (output layer initialized to small random numbers) and running enough iterations to get approximately the same error levels (.001 within one percent) the number of iterations required for the code-under-test and the commercial program NEUROSHELL (available from Ward Systems Group) were within three percent. The differences were

attributed to the fact that certain features of the commercial program were not documented (e.g., network initialization and stopping criteria equations) and may have been slightly different. Neurons in each level, learning rate, and momentum were the same in both cases.

4.2 Back Propagation Model Development

Since there are no significant advantages to using more than two hidden layers (only that the total number of neurons for a problem might be less) a program, BACKPROP, that supports no, one, or two hidden layers was coded. The listing of BACKPROP is included in Appendix C.

BACKPROP allows external inputs of many of the model parameters through predefined file contents. If a file for the parameter of interest exists, its contents are used; otherwise, the program's default value is used. The contents of these files define: (1) number of hidden layer neurons, (2) learning rate, (3) momentum, (4) termination error level, (5) termination iteration count, and (6) number of iterations before automatic network parameter save. The programs may run for several hours on a microcomputer and, thus, an automatic save to disk function was incorporated in case of power outage so that the program could be resumed at the last save point. Not only are the current network values saved but also all momentum terms so that upon resumption the exact same results should be obtained. The program may also be interrupted by depressing the escape key

(microcomputer version) and resumed later.

The program has three modes of operation. The first is to start from scratch on a new problem and to develop and test a network model. The second is to resume learning after an interruption and develop and test a network model. The third is to allow a set of new test patterns to be entered into an existing network model (this mode emulates the way hardware would function). At the beginning of the program, the operator is asked to select an option and the pertinent files are read in from disk. The program may then be left to operate in unsupervised mode; it displays current error level and iteration count on the screen periodically so that program progress may be observed.

4.3 Preneural Signal Processing

Initial tests showed that proper network design, relied heavily on the preconditioning techniques that are applied to the data. Basically, this research uses neural networks as clustering type pattern classifiers; for best performance as classifiers, they must have carefully selected training inputs which allow them to develop classification criteria. To this end, a program, called PRENEURA, was developed.

PRENEURA allows a number of signal processing operations to be applied to a single signal and recorded; then, the sequence may be "played back" and applied to all signals. This creates a set of input patterns which have been modified in the same manner. Most operations which are

available have easily implementable hardware counterparts. Therefore, once the proper sequence of preprocessing operations for a particular problem is determined, operations may be transformed to real time hardware. For example, jump a certain number of points forward, translates to a hardware delay circuit.

PRENEURA also has a type of subroutine (or submodule) feature. It may also record a subset of keystrokes; such that, a keystroke in the main keystroke sequence will cause a subsequence of keystrokes to be "played back." This allows some operations which require more sophisticated hardware, e.g., time-frequency data, to be emulated. To do this type of operation, a subsequence of keystrokes that took the FFT of a group of points after filtering, placed the output in a queue, and restored the original data might be defined. Then the main sequence of keystrokes would advance a fixed number of points, execute the subsequence, advance again, execute the subsequence again, etc., until the proper amount of data was stored in the queue. Finally the original data would be overwritten by that in the queue and the resulting, processed data would be saved.

To precisely define what operations were applied to the data to condition it for input to a neural network, the keystroke sequence applied will be given. The possible operations and their corresponding keystrokes are defined in Appendix A.

4.4 Results Processing

In order to consolidate the outputs of the BACKPROP program, a program called RESULTS was written. RESULTS converts the numeric weight and bias values to alphabetic characters with each successive letter corresponding to a larger weight range of values. RESULTS also counts the number of correctly identified, incorrectly identified, and unidentified learning and test cases and computes percentages of the total. It computes maximum, minimum, and average error for correctly identified cases. A listing of the program RESULTS is given in Appendix D.

The outputs of this program were used to construct tables of the number of iterations required to reach the desired error level of 0.1 percent, tables of the average uncertainty for correctly identified signals, and tables of the percent of cases correctly identified. These tables are based on the test case set as the learning sets were always learned to 100 percent correct classification (i.e., combinations of learning rate and momentum that did not result in complete learning were discarded).

CHAPTER V

FLUFF DETECTION

Suspended sediment, called fluff, causes reflections at shallower depths than the hard bottom. When suspended material is present in a channel that must be maintained at a project depth for navigation purposes, conventional depth sounders often give incorrect readings. An instrument which would collaborate the correctness or incorrectness of the reading by indicating when fluff was present would be a great asset to hydrographic surveyors, as it would signal when lead line depth determination was needed. The purpose of the fluff study was to determine if a neural network could delineate between a fluff signal and a hard bottom signal.

5.1 Physical Data

Depth sounder data from Alabama, Georgia, Mississippi, and California provided input for this study. The data from Savannah, Georgia, was collected on analog tape and digitized as part of this study; the remainder of the data was digitized directly from the depth sounder. In all cases, two channels of analog data provided input: (1) the

raw analog return and (2) the timing pulse that signals the transmission of the initial pulse. The envelope of the return is used by the depth sounder to determine the point of the return for depth determination. All depth sounders had circuits to rectify and filter the return to obtain the envelope. Unfortunately this signal was recorded on a bad tape channel in many tests and was unusable, and software preprocessing was used in the PRENEURA program to simulate the effects of envelope extraction circuits.

The only criteria used for discarding data were (1) no ground truth (i.e., the bottom corresponding to the signal was unknown), (2) clipped signals due to gain set too high, (3) anomalous signals. The first criteria accounted for most of the discards; the third criteria accounted for only a few discards. The anomalous signals discarded involved a reflection at a shallower depth than other signals in the same area and are believed to have been caused by fish. There were only two anomalous signals and their exclusion should not be detrimental to the solution; the occurrence of large schools of fish and submerged objects is rare in ship channels. The effect on the neural model would be either misidentification or no identification of anomalies as there is no output neuron for anomaly classification (reducing the accuracy by at most two percent).

Hard bottoms are characterized by the first return being much larger in amplitude and shorter in duration than

the following returns. A typical hard bottom return obtained in the area around Savannah, Georgia, is shown in Figure 18. It shows a complete signal including (1) the surface reflections, (2) the lack of signal during the water media propagation, and (3) the bottom reflections. Depth sounders use a fixed hardware time delay to disable processing during the surface reflection period.

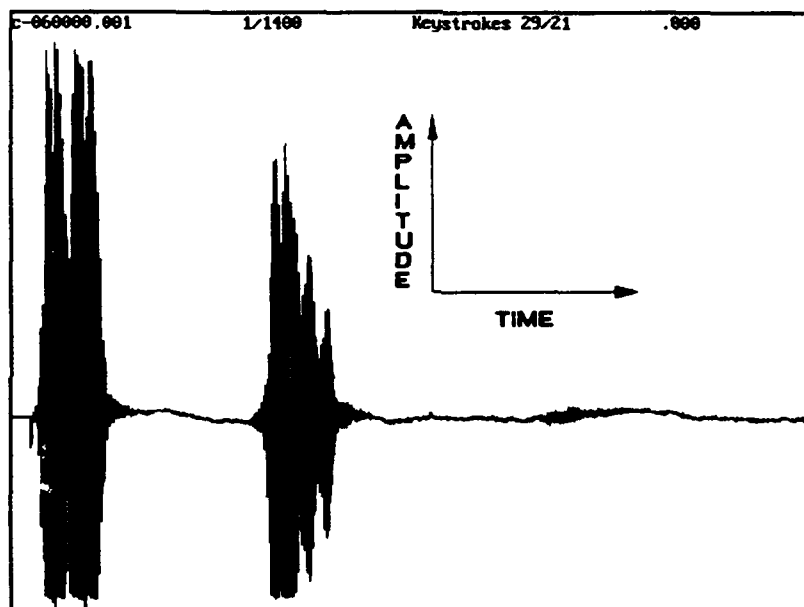


Figure 18. Return with Surface Reflection

Figure 19 shows the same signal advanced to the beginning of the first bottom reflection, thus allowing visual correlation with the processed signals that follow.

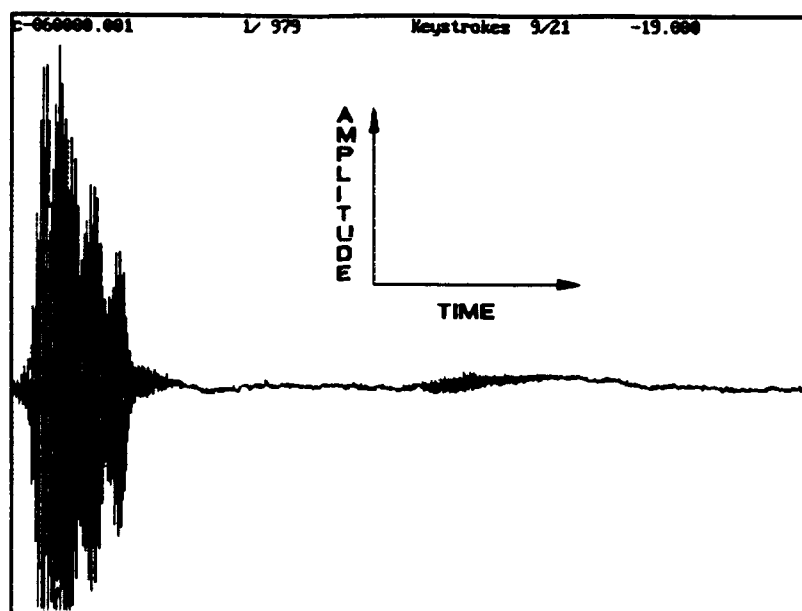


Figure 19. Hard Bottom Return

The signals that follow will be likewise advanced to the beginning of the first bottom reflection. Fluff returns are

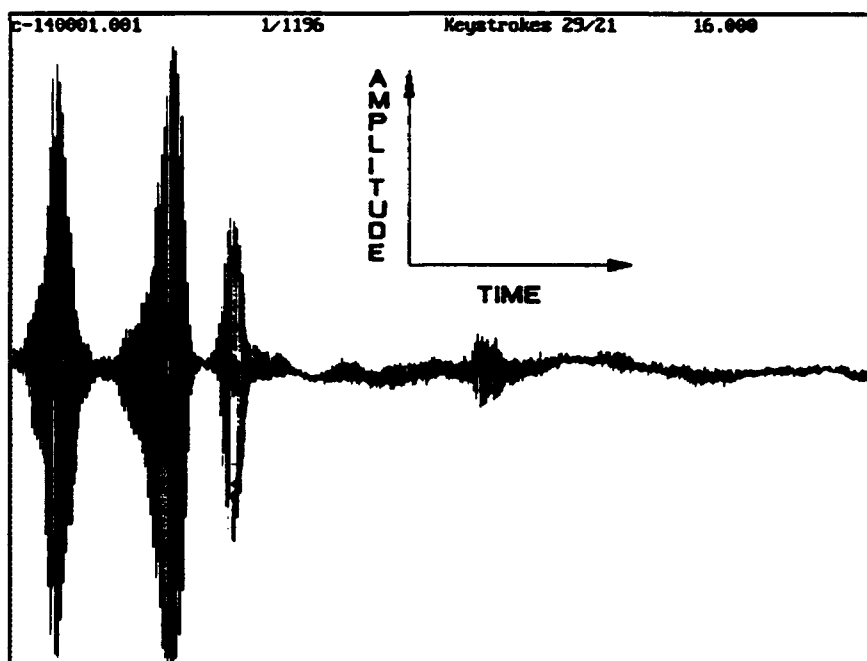


Figure 20. Fluff Return

marked by secondary wavelets that are significant in

amplitude when compared to the first return. A typical fluff return obtained in the area around Savannah, Georgia, is shown in Figure 20. The first wavelet is the fluff return; the wavelets that follow correspond to the bottom.

5.2 Time Domain Model

Data representing a reflected signal is sampled at evenly spaced increments of time and thus represents the signal in the time domain. If an instrument could be constructed that operated in the time domain, the circuitry would be simpler, more compact, and cheaper than a solution that existed in the frequency domain.

5.2.1 Input Generation

Absolute signal amplitude is not a suitable method for making the classification, because amplitude varies due to angle of incident and bottom roughness more than it does due to difference in material. It is not uncommon, for consecutive signals to vary by as much as ten times in amplitude. Therefore, in all processing the signal segment of interest was normalized between zero and one, corresponding to the respective minimum and maximum values. Thus, producing processed data that reflected relative wavelet amplitude was the goal of preprocessing.

The keystrokes used in preprocessing the neural data were

i5.jrtordddank .

A description of the meaning of these strokes accompanies the discussion of the PRENEURA program in Appendix A. The raw return signal was first rectified about the mean and envelope detected to simulate the front end circuitry commonly used in depth sounders (since this data channel was corrupted during recording in many cases). The jump distance file, JUMPDIST, was set to 200 in order to advance past the surface reflection before beginning the threshold search for the first return. The threshold level file, THRESHOLD, was set to .05 to catch the beginning of the first return. Then the data was clipped and decimated to provide 64 points for input to the neural network.

The example hard bottom signal is shown after amplitude (AMPL) processing in Figure 21.

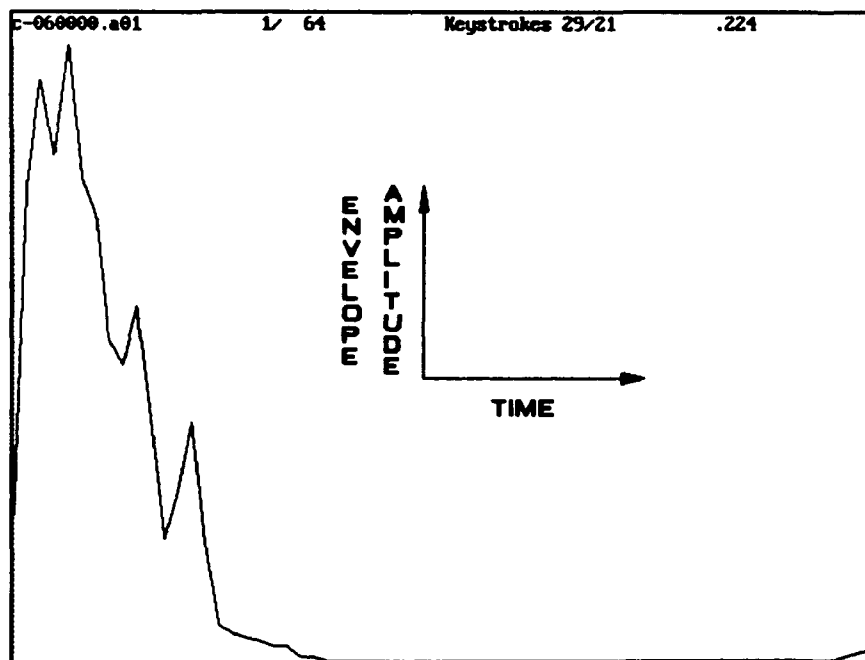


Figure 21. Amplitude Processed Hard Bottom

The example fluff signal is shown after amplitude processing in Figure 22.

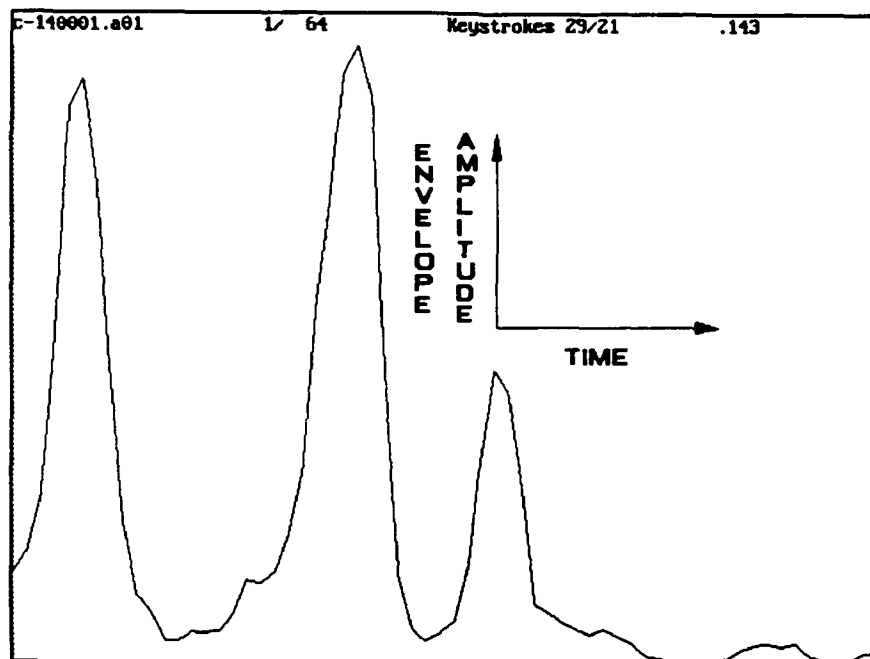


Figure 22. Amplitude Processed Fluff

5.2.2 Model Derived

Models for hundreds of combinations of hidden layer sizes, learning rates, and momentum were developed and tested. The results were too numerous to list; however, charts of some of the more interesting results are included. The number of hidden layers that worked best was a number which caused a uniform fanout or fanin between input and output elements. For example, if there were 64 inputs, 2 outputs, and 1 hidden layer, the hidden layer should have about 12 elements. The fanin between the input and hidden layer would then be $64/12$ or about 6 and the fanin between

the hidden layer and output layer would be 12/2 or 6. For 2 hidden layers 20 and 6 might be good choices giving a fanin of about 3 for each layer. Too many hidden elements (32 in the test cases) caused the network not to converge to a small error value. Too few hidden elements negated the effect of the hidden layer as it tended to simply replace the output layer (with results from the hidden layer passed through to the outputs).

One of the aims of the research was to produce abstract solutions. Abstraction allows networks to be trained on data acquired at one site and used to classify data from another site. The most abstract solutions (i.e., the ones that performed best on data from another site) were obtained when learning rate and momentum were chosen so as to minimize the number of learning iterations. This minimum value is obtained by running numerous combinations and counting the iterations for each combination.

Tables of results are presented in groups of 3. The result tables list: (1) the iterations required to learn the learning cases set, (2) the average percent of uncertainty on the test cases, and (3) the percent of test cases correctly identified. The tables show the results of the models developed having (1) 0 hidden layers (often referred to as 2 layer models), (2) 1 hidden layer (often referred to as 3 layer models), and (3) 2 hidden layers (often referred to as 4 layer models). The tables illustrate the results of

varying learning rate or momentum. Tables 1 through 6 show the results for 64 input elements. Tables 7 through 12 list the results for 32 input elements. Tables 13 through 18 display the 16 input element results.

First momentum was set to 0.0 and learning rate was varied. Table 1 gives the number of iterations required to converge to an error rate of less than 0.1 percent on the learning cases for 64 input elements.

Table 1.--Ampl 64 Inputs-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.6	548	173	548
1	336	101	44
2	173	47	24
4	91	28	14
8	31	14	8
16	9	5	166
32	5	335	No Conv

Table 2 gives the average percent uncertainty for the correctly identified cases.

Table 2.--Ampl 64 Inputs-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layers % Uncertain	2 Hidden Layers % Uncertain
0.6	7.86	4.93	4.07
1	7.65	4.84	3.95
2	7.33	4.87	3.69
4	6.87	4.21	3.05
8	4.32	2.86	2.46
16	0.87	2.09	1.63
32	0.02	1.58	No Conv

Table 3 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 3.--Ampl 64 Inputs-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.6	100	100	100
1	100	100	100
2	100	100	100
4	100	100	100
8	100	100	100
16	100	100	100
32	100	100	No Conv

The next group of 3 tables show the results when the learning rate was held constant at 0.6 while momentum rate was varied. The number of iterations required for

convergence to a 0.1 percent error rate for 64 input elements is given in Table 4.

Table 4.--Ampl 64 Inputs-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layers # Passes	2 Hidden Layers # Passes
0	548	173	76
0.3	390	119	52
0.6	231	70	32
0.9	489	7	35

Table 5 gives the average uncertainty versus momentum for 64 input elements.

Table 5.--Ampl 64 Inputs-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layers % Uncertain	2 Hidden Layers % Uncertain
0	7.86	4.93	4.07
0.3	7.68	5.05	4.1
0.6	7.19	4.6	3.54
0.9	0.95	3.28	6.37

Table 6 gives the percent of test cases correctly identified as a function of momentum for 64 input elements.

Table 6.--Ampl 64 Inputs-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layers % Correct	2 Hidden Layers % Correct
0	100	100	100
0.3	100	100	100
0.6	100	100	100
0.9	100	99.58	99.58

To determine the effect of using fewer input elements, every other data point was deleted, leaving 32 inputs. The first tests determined the effect of using no momentum and varying learning rate. Table 7 shows the number of iterations required to reach an error rate of 0.1 percent.

Table 7.--Ampl 32 Inputs-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.6	1151	428	117
1	702	256	67
2	361	124	35
4	187	64	18
8	100	22	8
16	59	8	5
32	29	14	810

Table 8 gives the average uncertainty percent for the test cases correctly identified versus learning rate for 32 input

elements.

Table 8.--Ampl 32 Inputs-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.6	7.8	3.93	3.62
1	7.68	3.84	3.53
2	7.36	3.95	3.48
4	7.1	3.85	3.25
8	6.64	2.48	2.74
16	6.13	1.86	1.23
32	1.64	1.37	0.3

Table 9 gives the percentage of test cases correctly identified versus learning rate for 32 input elements.

Table 9.--Ampl 32 Inputs-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.6	99.16	100	100
1	99.16	100	100
2	99.16	100	100
4	99.16	100	100
8	99.16	100	100
16	99.16	100	100
32	100	92.89	98.33

Next learning rate was held constant at 0.6 and momentum was

varied for the 32 input elements as shown in the next group of 3 tables. Table 10 shows number of iterations required to reach an error rate of 0.1 percent.

Table 10.--Ampl 32 Inputs-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layers # Passes	2 Hidden Layers # Passes
0	1151	428	117
0.3	815	300	79
0.6	477	167	41
0.9	127	20	94

Table 11 shows the average uncertainty percentage for correctly identified test cases versus momentum.

Table 11.--Ampl 32 Inputs-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0	7.8	3.93	3.62
0.3	7.69	3.98	3.64
0.6	7.41	3.9	3.6
0.9	4.05	1.7	1.27

Table 12 shows the percent of test cases correctly classified versus momentum for 32 input elements.

Table 12.--Ampl 32 Inputs-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0	99.16	100	100
0.3	99.16	100	100
0.6	99.16	100	100
0.9	100	100	100

Finally every other input element was discarded again leaving 16 input elements. First the learning rate was varied with no momentum as shown in the following 3 tables. Table 13 gives the number of iterations required to reach an error rate of 0.1 percent with 16 input elements.

Table 13.--Ampl 16 Inputs-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.6	3800	622	163
1	2283	366	98
2	1145	175	49
4	573	81	27
8	284	40	7
16	138	50	7
32	78	14	10

Table 14 gives the average uncertainty percentage versus learning rate for 16 inputs.

Table 14.--Ampl 16 Inputs-Uncertainty Vs. Learn Rate

63

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.6	7.1	4.03	3.41
1	7	3.96	3.32
2	6.78	3.75	3.09
4	6.47	3.54	2.84
8	6.16	2.71	1.65
16	5.91	2.5	1.8
32	4.12	2.37	0.95

Table 15 gives the percent of the test cases correctly identified versus learning rate for 16 inputs.

Table 15.--Ampl 16 Inputs-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.6	97.91	99.16	99.58
1	97.91	99.16	99.58
2	97.91	99.16	99.58
4	97.91	99.16	99.58
8	97.91	99.58	99.58
16	98.33	99.16	99.58
32	98.33	99.16	99.16

Next learning rate was held constant at 0.6 and momentum was varied for the 16 inputs as shown in the next 3 tables.

Table 16 gives the number of iterations required to reach 0.1 percent error rate versus learning rate.

Table 16.--Ampl 16 Inputs-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0	3800	622	163
0.3	2663	428	115
0.6	1526	234	70
0.8	772	110	23
0.9	425	27	13

Table 17 gives the average uncertainty percentage for the test cases versus learning rate for 16 inputs.

Table 17.--Ampl 16 Inputs-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0	7.1	4.03	3.41
0.3	7.04	3.92	3.33
0.6	6.87	3.78	2.97
0.8	6.52	2.97	2.17
0.9	5.8	0.89	1.12

Table 18 gives percent correctly identified versus momentum for 16 inputs.

Table 18.--Ampl 16 Inputs-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0	97.91	99.16	99.58
0.3	97.91	99.16	99.58
0.6	97.91	99.16	99.16
0.8	97.91	99.16	99.58
0.9	98.33	99.16	98.33

From the average uncertainty and percent correct tables, the most abstract model was obtained by setting learning rate to 32 and momentum to 0 with no hidden layers. The weights for the 64 inputs to the output neuron which is on for hard bottoms and off for fluff are

(IN01-IN32) FZZZZZRZZZZZQHAtnlprxzzzzzzzzuqq

and

(IN33-IN64) zzzzzzzzzzpxyzzzzzzzzvzmrsrottqzz.

The weights for the 64 inputs to the output neuron which is on for fluff and off for hard bottoms are

(IN01-IN32) fzzzzzzrzzzzzzqhaTNLPRXZZZZZZZZUQQ

and

(IN33-IN64) ZZZZZZZZZZPXYYZZZZZZZZVZMRSROTTQZZ.

The notation for weights is that 0 corresponds to a 0.0 weight and the letters of the alphabet increment magnitude by 0.1 in order, with Z representing any values greater than Y's maximum 2.5. Upper case letters correspond to exciting

weights and lower case letters correspond to inhibiting weights. Thus A corresponds to weights between 0.0 and 0.1, B corresponds to weights between 0.1 and 0.2, a corresponds to weights between 0.0 and -0.1, b corresponds to weights between -0.1 and -0.2, etc. From the weights it can be seen that the fluff output is strongly inhibited by the first 15 inputs and excited by the remaining inputs, and the hard bottom output is the opposite. This is as would be expected as the network has learned an important identifying characteristic of fluff and hard bottoms. Hard bottoms have a strong first wavelet; while fluff has multiple weaker wavelets. Thus, if multiple equal amplitude wavelets are present the signal is likely to be fluff. The biases are

zZ.

Thus the hard bottom output neuron has a strong negative bias and the fluff output neuron has a strong positive bias. These strong biases are necessary to counteract the magnitude predominance of the first 15 input values.

5.3 Frequency Domain Model

The first attempt at frequency domain modeling involved taking the FFT of 128 consecutive data points to obtain 64 data points of magnitude spectrum. Both amplitude and power spectrums were input to the BACKPROP program with no success. The problem with frequency processing is that only the relative value of each frequency component is used

regardless of where in the sample record the component was active. Thus a uniform small amplitude frequency across the entire record might map into the frequency domain as the same amplitude as a high amplitude pulse of the frequency occurring over a small segment of the record. This was the case with the hard bottom and fluff signals and the frequency domain representation of both appeared visually identical. To circumvent this problem, the concept of wavelets of energy is often used in the processing of acoustic records.

5.3.1 Input Generation

To allow the energy of the wavelets to be applied to different neural network inputs, time-frequency (T/F) preprocessing was applied to the signal. A number of combinations of different segment length and overlap were tested. The most successful combination was to take sixteen eight sample segments, overlapped fifty percent, along the sample record. The four higher frequency components of each FFT were discarded (thus providing for low pass filtering). The keystrokes used to produce the 64 neural network inputs were

i5.jrtpr9;3b)))ecddfb>>>ec,7));=====rank .

Refer to Appendix A for the meaning of these keystrokes. In short, a subset of keystrokes is used to perform the 16 FFT's and advance the pointer. The 4 data points of each of the 16 passes of subset keystroke processing are placed on a

data queue and the contents of the queue used as 64 inputs to the neural network.

The example of hard bottom signal after time-frequency processing by PRENEURA is shown in Figure 23.

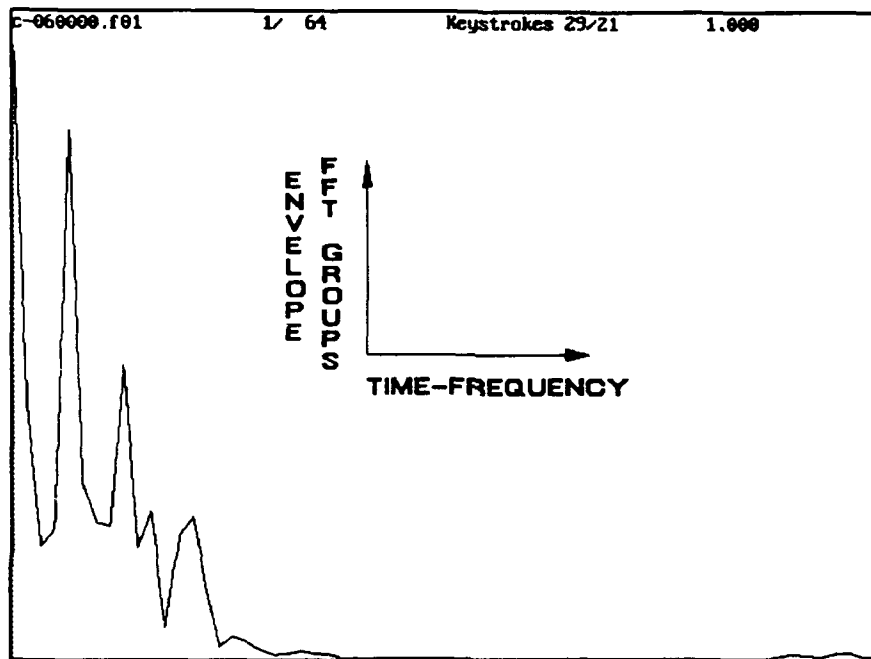


Figure 23. Time-Frequency of Hard Bottom

The example of fluff after time-frequency preprocessing is shown in Figure 24.

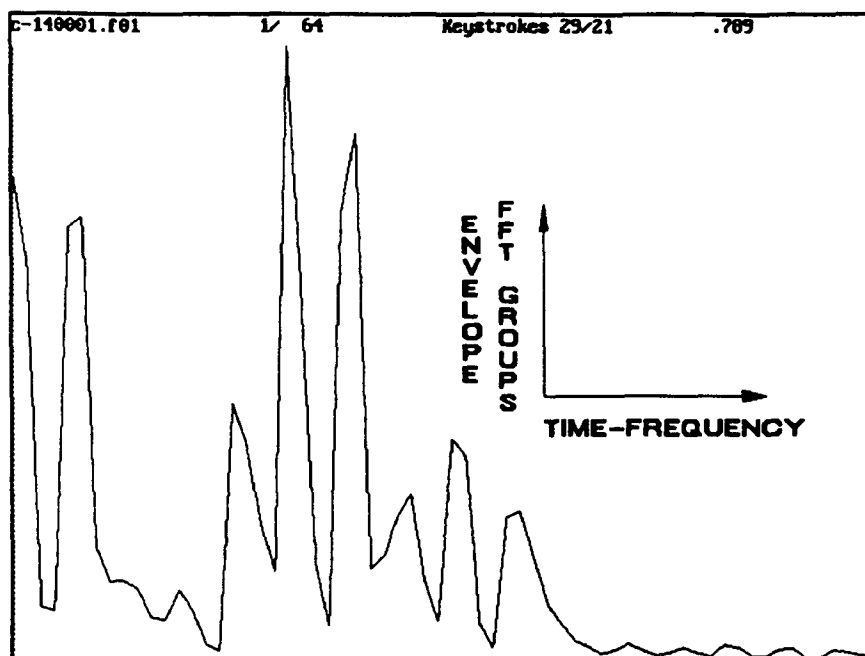


Figure 24. Time-Frequency Processed Fluff

5.3.2 Model Derived

In an approach similar to the one used for amplitude processing, a number of time-frequency models were developed. Tables 19 through 24 list the results for 64 input elements. Tables 25 through 30 list the results for 32 input elements. First momentum was set to 0.0 and learning rate was varied as shown in the following 3 tables.

Table 19.--T/F 64 Inputs-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.6	844	285	116
1	506	171	70
2	251	84	35
4	124	37	18
8	60	21	10
16	9	6	3
32	7	9	2676

Table 19 gives the number of iterations required to converge to an error rate of less than 0.1 percent on the learn cases for 64 input elements. Table 20 gives the average percent uncertainty for the correctly identified cases.

Table 20.--T/F 64 Inputs-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.6	14.9	10	7.98
1	14.73	10.04	7.81
2	14.45	9.33	7.18
4	14.33	8.49	6.27
8	13.26	6.77	4.56
16	2.25	1.35	3.36
32	3.57	0.06	0.06

Table 21 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 21.--T/F 64 Inputs-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.6	97.07	99.16	99.16
1	97.07	99.16	99.16
2	97.07	98.58	98.58
4	97.91	99.58	99.58
8	97.91	100	100
16	100	100	100
32	100	100	100

Next learning rate was held constant at 0.6 while momentum rate was varied as shown in the next 3 tables. The number of iterations required for convergence to a 0.1 percent error rate for 64 input elements is given in Table 22.

Table 22.--T/F 64 Inputs-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0	844	285	116
0.3	590	199	81
0.6	336	110	46
0.9	82	9	10

Table 23 gives the average uncertainty versus momentum for 64 input elements.

Table 23.--T/F 64 Inputs-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0	14.9	10	7.98
0.3	14.75	9.87	7.88
0.6	14.49	10.02	7.79
0.9	5.99	2.49	2.06

Table 24 gives the percent of test cases correctly identified as a function of momentum for 64 input elements.

Table 24.--T/F 64 Inputs-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0	97.07	99.16	99.16
0.3	97.07	99.58	99.58
0.6	97.49	99.16	99.58
0.9	100	100	100

To determine the effect of using fewer input elements, every other data point was deleted, leaving 32 inputs.

Two combinations of learning rate and momentum did not converge: (1) learning rate equals 0.6 and momentum equals 0 and (2) learning rate equals 0.6 and momentum equals 0.3.

The first tests determined the effect of using no momentum

and varying learning rate. Table 25 shows the number of iterations required to reach an error rate of 0.1 percent.

Table 25.--T/F 32 Inputs-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
1	1051	201	70
2	521	102	34
4	258	53	18
8	130	25	6
16	56	9	4

Table 26 gives the average uncertainty percent for the test cases correctly identified versus learning rate for 32 input elements.

Table 26.--T/F 32 Inputs-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
1	14.04	12.97	11.35
2	14.02	12.26	10.72
4	14.35	11.81	10.08
8	14.79	8.06	8.96
16	14.71	5.88	1.55

Table 27 gives the percentage of test cases correctly identified versus learning rate for 32 input elements.

Table 27.--T/F 32 Inputs-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
1	72.8	84.52	84.1
2	73.22	84.94	85.77
4	74.48	90.79	88.7
8	77.82	99.16	90.38
16	78.24	84.1	99.58

Next learning rate was held constant at 0.6 and momentum was varied for the 32 input elements as shown in the next 3 tables. Table 28 shows number of iterations required to reach an error rate of 0.1 percent.

Table 28.--T/F 32 Inputs-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.6	695	133	48
0.9	85	23	34

Table 29 shows the average uncertainty percentage for correctly identified test cases versus momentum.

Table 29.--T/F 32 Inputs-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.6	14.15	12.11	10.45
0.9	11.22	7.56	2.86

Table 30 shows the percent of test cases correctly classified versus momentum for 32 input elements.

Table 30.--T/F 32 Inputs-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.6	73.22	86.19	84.52
0.9	85.77	90.79	96.65

From the average uncertainty and percent correct tables, the most abstract model was obtained by setting learning rate to 32 and momentum to 0 with no hidden layers. The weights for the 64 inputs to the 16 hidden neurons are

(In01.01-In01.32) ZRHJZJLKUFNAAEDbnledumgbszoizztf,
(In01.33-In01.64) ztogwzjghgddkdbciecchjhezdzgzh,
(In02.01-In02.32) UCCDZDFEMCHabaBbkkedokfbljebmlgd,
(In02.33-In02.64) smebgleeliecoecclfdcdfcbgecbwtii,
(In03.01-In03.32) bbAAaaaabaaccaaddbbfdbadecaeecc,
(In03.33-In03.64) febadbbcbbaadbaacbaabbaadcaaecc,
(In04.01-In04.32) cbaabbaabbaaccbaddbbbedbadecaedca,

(In04.33-In04.64) fdbadcbbbbaacaaacbaabaaadcaagecb,
(In05.01-In05.32) ZJEGZEJIFfKenIAfqphfumgbsxnhzzqe,
(In05.33-In05.64) zrmfvzjhqoigzjegxnhhmnpjfzzdfzqhh,
(In06.01-In06.32) KBBBNCCCCGAEabaAbggccjgdbhgcbhgdb,
(In06.33-In06.64) khcaigccgfcbjdbbheccdcbaafdbbolff,
(In07.01-In07.32) JBBBMBCCFADabbAbggcbjgdaggdbhgdb,
(In07.33-In07.64) kgcahgccgecbidbbhdccdcbaafdbbnjee,
(In08.01-In08.32) AaaaBaAAaaAaccaaeebbgecaefcbeecb,
(In08.33-In08.64) gebadbbcbbadbaacbaabbaaddabhfcc,
(In09.01-In09.32) ZDCEZDFFPCIacbBcnmfeglgnkebonhd,
(In09.33-In09.64) vofbsnfenjfdqfcdmgddgecbhecczyl1,
(In10.01-In10.32) ecabebbbbedbaeebbccbaaaaabbbaedba,
(In10.33-In10.64) ecbaccaaccbaebaadcbbcbbadbaafcba,
(In11.01-In11.32) PCBCTCDDKBFabaAbiidcliebihdbjiec,
(In11.33-In11.64) njdblidcigdcldbcjeccedbbfddbtqhh,
(In12.01-In12.32) zddezgignAkCECbDIHEDJFDBJHDBSQJE,
(In12.33-In12.64) ZSHDXRGFSOIEYIDFTKGFJHFCMLCDYNGF,
(In13.01-In13.32) eeabccAakjadnmedefdcbbaadcaaffcb,
(In13.33-In13.64) ecaadcbbihecpfcdnheefedbdcaadbaa,
(In14.01-In14.32) cdabbcAaihadlkddfgdcdbaeebaffcb,
(In14.33-In14.64) fdbadcbbhgdcobclgdddeecbdaaeabaa,
(In15.01-In15.32) bbaaaaaabcaadbbddbbccbacbaedca,
(In15.33-In15.64) fdbadcbadcbaebaadbbbcbbaddaahecb,
(In16.01-In16.32) fcabebbbbgecbffcbbbbbaAAAabaaaedba,
and
(In16.33-In16.64) ebbaccaadbbbgcabfcbccbaeeaaafbaa.

The weights for the output layer are

(hard bottom) YZAaZMMCZhSzpjci

and

(fluff) twljzuskvkyZzznk.

The biases for the hidden layer are

pnlmzjjlpnlNtsmo.

The biases for the output layer are

Zz.

CHAPTER VI

MATERIAL CLASSIFICATION

The type of material of which the bottom is composed is of interest to dredgers as it affects the time it takes to remove the material and the amount of maintenance and repair of the cutter head needed. To advance the state of the art of hydrographic surveying, the second problem considered is determination of bottom material by neural network classification.

6.1 Physical Data

Data from California was used for the bottom material classification problem; this was the only area that had material ground truth obtained by use of piston samplers. Three types of bottom material were classified: (1) hard silty sand, (2) soft clay, and (3) hard silty clay. Some hard silty clay and hard silty sand returns were almost identical in appearance and provided a test of neural networks ability to classify signals that had no visually discriminable classifying characteristics.

To help align the raw and processed signals the raw signals are advanced to the beginning of the first wavelet beyond the surface reflection. Figure 25 shows a raw silty sand signal.

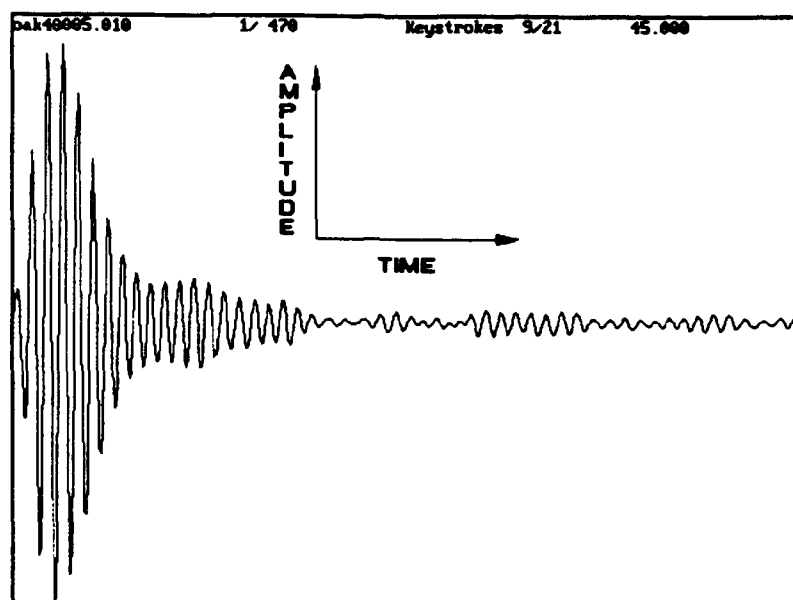


Figure 25. Return from Silty Sand Bottom

Figure 26 shows a typical soft clay reflected signal.

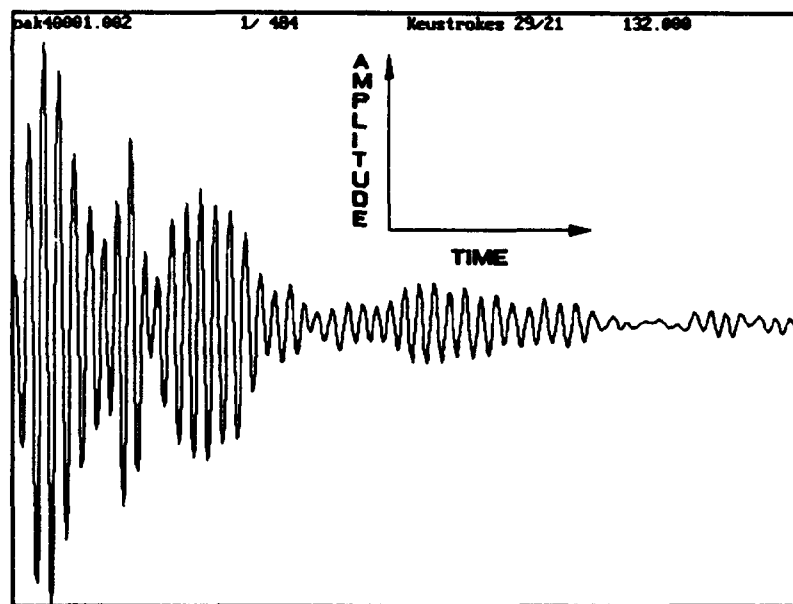


Figure 26. Return from Soft Clay Bottom

Figure 27 shows a typical hard silty clay reflected signal.

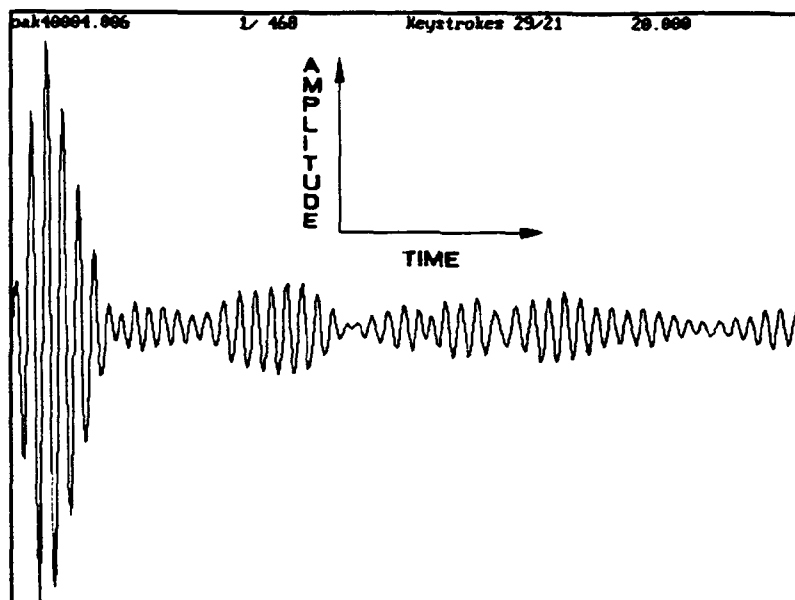


Figure 27. Return from Hard Silty Clay

6.2 Time Domain Model

As with the fluff problem a time domain solution would be desirable because of ease of hardware implementation. As before, relative rather than absolute amplitude was used for the neural network input signal preprocessing.

6.2.1 Input Generation

The same keystrokes that were used for fluff amplitude processing were used for general classification processing. The amplitude processed hard silty sand signal is shown in Figure 28.

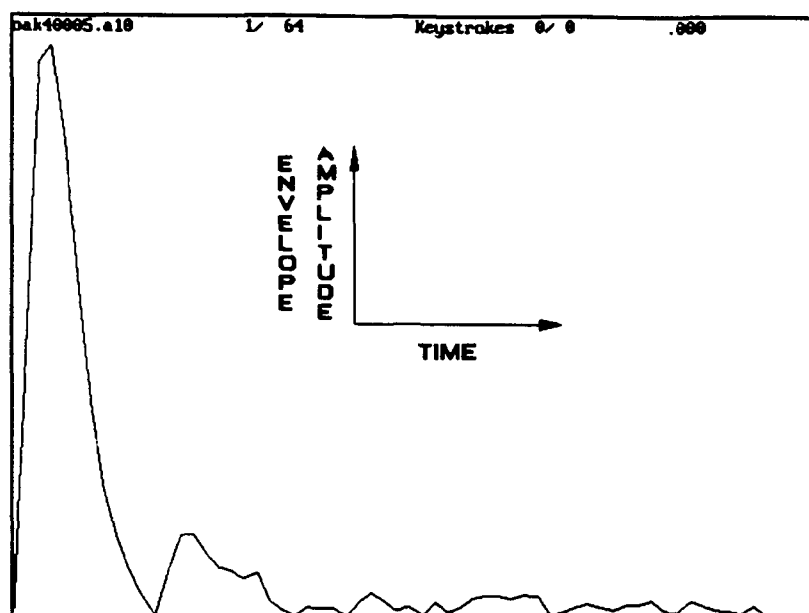


Figure 28. Amplitude Processed Silty Sand

The amplitude processed soft clay signal is shown in Figure 29.

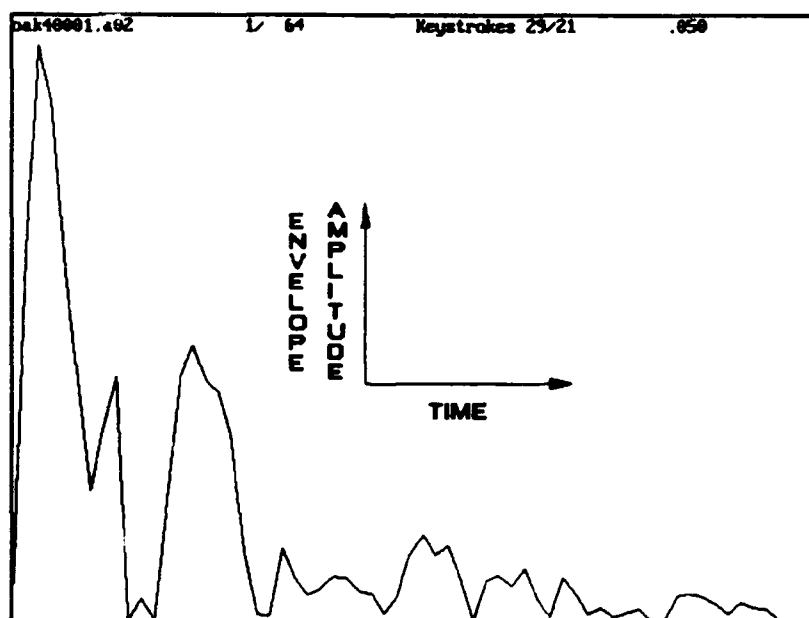


Figure 29. Amplitude Processed Soft Clay

The amplitude processed silty clay signal is shown in Figure 30.

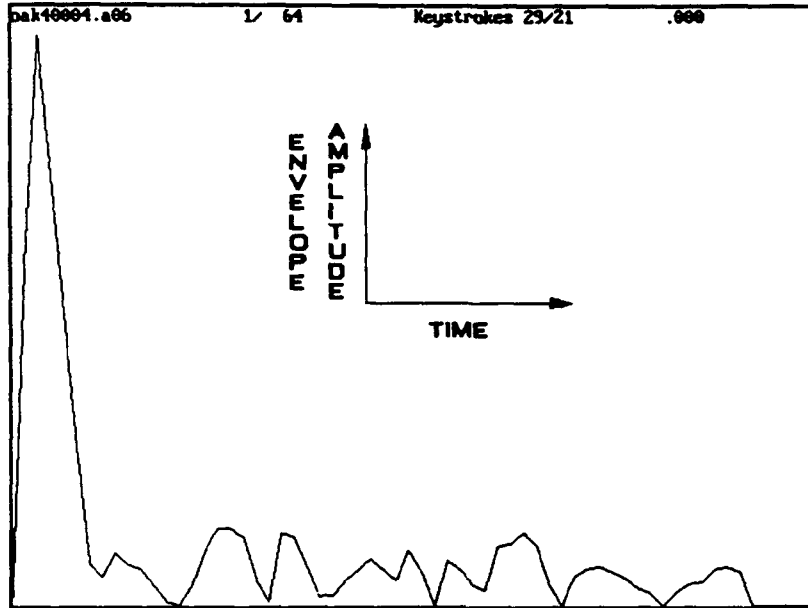


Figure 30. Amplitude Processed Silty Clay

6.2.2 Model Derived

The general classification problem proved to be much harder to model than the fluff problem. When momentum was set to zero, solutions could only be found for 0 hidden layers, no matter what learning rate was used. Too many hidden layer elements also prevented convergence. Using one hidden layer of 16 elements or two hidden layers of 20 and 6 elements, respectively, resulted in no convergence. When momentum was set to 0.3 and learning rate was set to 1.0, 1.2, or 1.5 then convergence was obtained for a single hidden layer of 10 elements or two hidden layers of 12 and

5, respectively. However, reducing the number of elements of the single hidden layer model to 8 and the two hidden layer model to 9 and 5, respectively, resulted in more abstract solutions. Therefore, tables for the 8 element single layer and the 9 and 5 neuron two layer models are presented. Tables 31 through 36 show the results of using 64 input elements.

First momentum was set to 0.3 and learning rate was varied as shown in the next 3 tables. Table 31 gives the number of iterations required to converge to an error rate of less than 0.1 percent on the learning cases for 64 input elements. The maximum number of iterations was limited to 50000 and in some cases the 0 hidden layer model terminated at error rates between 0.1 percent and 0.25 percent.

Table 31.--Ampl Material-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.4	50000	1568	831
0.7	50000	948	5802
1	47049	5255	791
1.5	31282	525	362

Table 32 gives the average percent uncertainty for the correctly identified cases.

Table 32.--Ampl Material-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.4	1.6	1.06	0.5
0.7	1.38	0.81	0.54
1	1.33	0.62	0.56
1.5	1.32	0.55	0.74

Table 33 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 33.--Ampl Material-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.4	77.85	87.25	91.28
0.7	77.18	91.95	91.95
1	77.18	92.62	93.29
1.5	77.18	91.28	93.29

Next learning rate was held constant at 0.4 while momentum rate was varied as shown in the next 3 tables. The number of iterations required for convergence to a 0.1 percent error rate for 64 input elements is given in Table 34.

Table 34.--Ampl Material-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.4	50000	1568	831
0.5	50000	1198	607
0.6	50000	950	781
0.7	50000	821	1027
0.9	16826	411	No Conv

Table 35 gives the average uncertainty versus momentum for 64 input elements.

Table 35.--Ampl Material-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.4	1.6	1.05	0.5
0.5	1.42	1.25	0.79
0.6	1.39	0.71	0.82
0.7	1.35	0.58	0.67
0.9	1.27	0.36	No Conv

Table 36 gives the percent of test cases correctly identified as a function of momentum for 64 input elements.

Table 36.--Ampl Material-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.4	77.85	87.25	91.28
0.5	77.18	85.91	93.29
0.6	77.18	91.28	93.29
0.7	77.18	90.6	92.62
0.9	76.51	91.95	No Conv

To determine the effect of using fewer input elements, every other data point was deleted, leaving 32 inputs.

Convergence could only be obtained for 0 hidden layers and the error rate was greater than 30 percent.

The most abstract model had 2 hidden layers with a learning rate of 1 and a momentum of 0.3. The first hidden layer had 9 elements and the second hidden layer had 5. The model was taught on one set of 39 signals and tested on a separate set of 149 signals. 93.29 percent of the signals (139) were correctly classified. No signals were incorrectly classified. Ten signals were unclassified (i.e., 2 output neurons were active or no output neurons were active).

The weights for the first hidden layer were

(IN1.01-IN1.32) BErxdIhGPDFAPZVPNXQIIVGAAGDJHfaE,

(IN1.33-IN1.64) KBKhibkneiijaahbbeghffebAbbIDAAAA,

(IN2.01-IN2.32) ATOzAZtNBhgjAkverngHgoFLhmofgnrg,

(IN2.33-IN2.64) fecyogtxwxujishdaikjrkecfiFcaaaa,
 (IN3.01-IN3.32) BONaADfEfopmeoqlmkgabgdbdecbbddfc,
 (IN3.33-IN3.64) caaebbcdeeecebecbabbbadcbbbcacaAAA,
 (IN4.01-IN4.32) GzkZgzZzzjpczzAwKfgpFfrtRSYaCZZC,
 (IN4.33-IN4.64) hDoZZTZZZZZUUZTIHZZZZZODLRubaaaa,
 (IN5.01-IN5.32) GixAhoBjCcCGPSZKQVPLSXciFMJCDHKD,
 (IN5.33-IN5.64) IGINDEJIMKGFGIDAdBBDJGEBBFBDAAA,
 (IN6.01-IN6.32) Acgieafdcbcgheacecdedbeeededcbcc,
 (IN6.33-IN6.64) baaccbccddcbaccbbbbbabbbaabbAaaaaa,
 (IN7.01-IN7.32) RBMZpzcnyejizzkxezzqzznbakdvnZTA,
 (IN7.33-IN7.64) qdtZZPZZZZZRNZNOPSRNXSLEIJugcbbb,
 (IN8.01-IN8.32) bicGBbCihaacjkggfhfedddcbccfdCBb,
 (IN8.33-IN8.64) dbdEEBDEBDEAADAABCBCBAAAAdbaaaa,
 (IN9.01-IN9.32) abddcbaaaabbabbbbbbaaaaaaaaaaaaaa,

and

(IN9.33-IN9.64) aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAA.

The weights for the second hidden layer were

(IN1.1-IN1.9) hziZdaZCd,
 (IN2.1-IN2.9) MabnMayca,
 (IN3.1-IN3.9) qSNmsAOBC,

and

(IN4.1-IN4.9) WtpEWatdc.

The weights for the output layer were

(hard silty sand) zZZz,
 (soft clay) iZZZ,

and

(hard silty clay) ZzZy.

The biases for the first hidden layer were

gHGmpndjd.

The biases for the second hidden layer were

ubNj.

The biases for the output layer were

zzz.

There is no clearly discernable pattern to the weights or biases in this case. It is interesting to note that the weights for each of the 3 output neurons are balanced with 2 inhibiting and 2 exciting weights each.

6.3 Frequency Domain Model

The next set of inputs were generated in the time-frequency (T/F) domain. That is, FFT's were taken at multiple points along the time history of the reflected signal.

6.3.1 Input Generation

The same keystrokes that were used for fluff time-frequency (T/F) processing were used for general material classification processing. The time-frequency processed hard silty sand signal is shown in Figure 31.

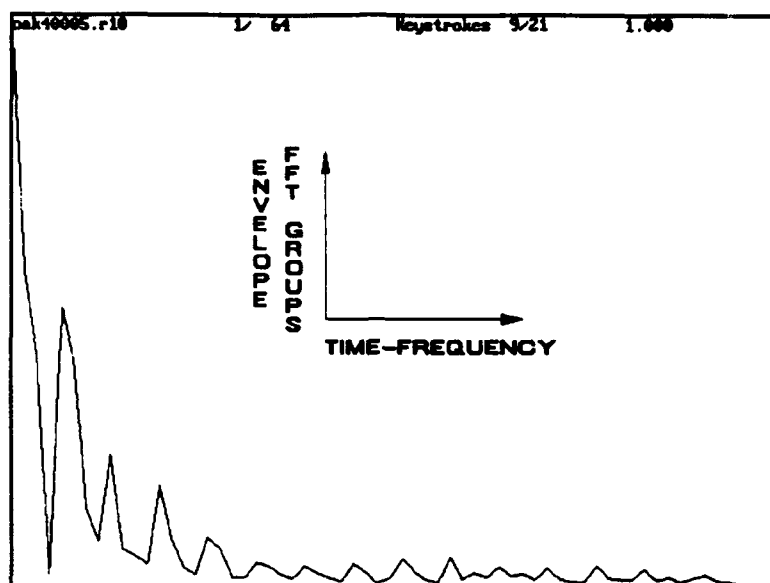


Figure 31. T/F Processed Silty Sand

The time-frequency processed soft clay signal is shown in Figure 32.

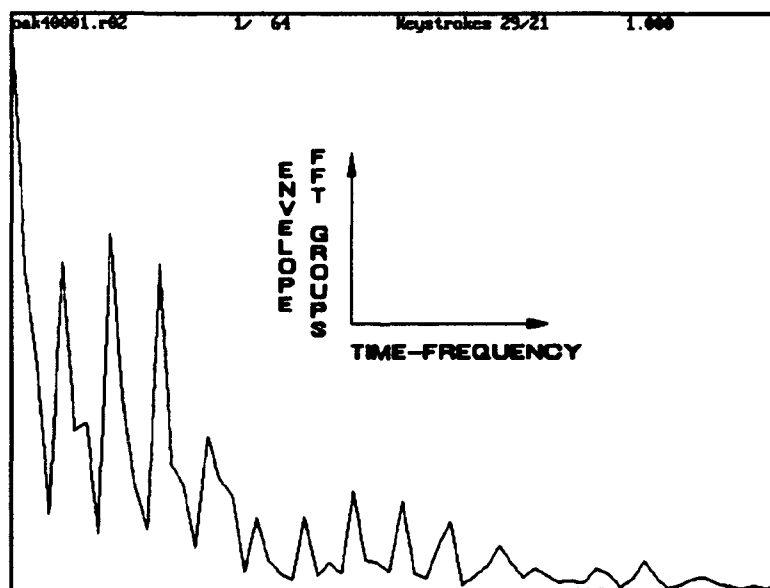


Figure 32. T/F Processed Soft Clay

The time-frequency processed silty clay signal is shown in Figure 33.

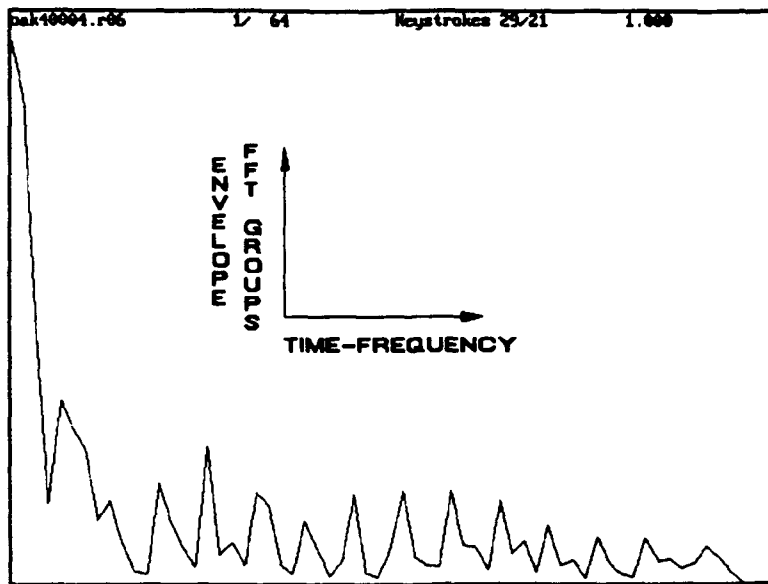


Figure 33. T/F Processed Silty Clay

6.3.2 Model Derived

In an approach similar to the one used for amplitude processing, a number of time-frequency models were developed. Tables 37 through 42 show the results of using 64 input elements. First momentum was set to 0.3 and learning rate was varied as shown in the following 3 tables.

Table 37.--T/F Material-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.4	50000	6526	No Conv
0.7	50000	4489	No Conv
1	50000	4988	No Conv
1.5	50000	No Conv	No Conv

Table 37 gives the number of iterations required to converge to an error rate of less than 0.1 percent on the learn cases for 64 input elements. Table 38 gives the average percent uncertainty for the correctly identified cases.

Table 38.--T/F Material-Uncertainty Vs. Learn Rate

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layer % Uncertain
0.4	1.56	1.44	No Conv
0.7	1.48	0.83	No Conv
1	1.44	0.85	No Conv
1.5	1.26	No Conv	No Conv

Table 39 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 39.--T/F Material-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.4	84.56	95.3	No Conv
0.7	84.56	95.97	No Conv
1	84.56	95.97	No Conv
1.5	83.89	No Conv	No Conv

Next learning rate was held constant at 0.4 while momentum rate was varied as shown in the next group of 3 tables. Table 40 gives the number of iterations required for convergence to an error rate of 0.1 percent for 64 input

elements.

Table 40.--T/F Material-Iterations Vs. Momentum

Momentum Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.3	50000	6526	No Conv
0.6	50000	4715	No Conv

Table 41 gives the average uncertainty versus momentum for 64 input elements.

Table 41.--T/F Material-Uncertainty Vs. Momentum

Momentum Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.3	1.56	1.44	No Conv
0.6	1.48	1.04	No Conv

Table 42 gives the percent of test cases correctly identified as a function of momentum for 64 input elements.

Table 42.--T/F Material-Correct Vs. Momentum

Momentum Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.3	84.56	95.3	No Conv
0.6	84.56	96.64	No Conv

The most abstract solution for the time-frequency inputs was an 8 element single hidden layer model with learning rate set to 0.4 and momentum set to 0.6. This model had correctly classified 96.64 percent (144) of the 149 input cases. None were incorrectly classified. 5 signals were not classified.

The weights for the hidden layer were

(IN1.01-IN1.32) ojkkhcigeeEffhibmikbgebcBABacDac,
 (IN1.33-IN1.64) JIbEPFbEKEDCLBFBMDCbGICCbACbaaaa,
 (IN2.01-IN2.32) qCAmGGbMMKfMPgBDFcaaDGaDDJaDIDc,
 (IN2.33-IN2.64) QMaIYIaISKDDOAHCSFEaMKHDBCEaAAAA,
 (IN3.01-IN3.32) CQczgVhlfcWuhlzcuzerlaiJIQbdTBj,
 (IN3.33-IN3.64) ZZaTZZaWZWLIzCUFZOLeZZOKbDKdbaaa,
 (IN4.01-IN4.32) omhfgfgddbAbaacadacabaAaAAAacAab,
 (IN4.33-IN4.64) bbbAAabABAAABaBACAAaBBAAAAaAAAA,
 (IN5.01-IN5.32) JBalFObfAAIfBciaiglaabAcJFGABIBc,
 (IN5.33-IN5.64) ROAGZLAIQHEDQBHBSEEOJLEDaBDbaaaa,
 (IN6.01-IN6.32) DJBphKdgrbLnrjvexmuemhBfCEKbaLCf,
 (IN6.33-IN6.64) ZXAKZTaOZPGEYBLDZIHcQTLGaCHbaaaa,
 (IN7.01-IN7.32) xzdKDjKETNjOZZUEZZZDPMEGccdBFjAE,
 (IN7.33-IN7.64) zzalzubmzkifzdmczigCovhfCbfcCBAB,
 (IN8.01-IN8.32) oDeogLacNLQiSPkCbEjABDHbNIOaFOEe,

and

(IN8.33-IN8.64) ZSaLZNanZOFGXALDZHGBsQJGBDGBAAAA.

The weights for the output layer were

(hard silty sand) ZzzZZzzz,

(soft clay) zzzZrzZZ,

and

(hard silty clay) ZBZUZZzz.

The biases for the hidden layer were

zmzsbqFq.

The biases for the output layer were

Zzz.

There is no obvious pattern to the weights or biases. The output weight for hard silty clay are mostly all exciting, while the other two materials are mostly inhibiting. The significance of this is that the neural network has developed a solution to the problem that probably could not have been obtained by any conventional approaches.

CHAPTER VII

DENSITY CLASSIFICATION

The third problem considered in advancing the state-of-the-art of Hydrographic Surveying was density classification. Neural networks were used to determine the density of the top layer of bottom material.

Various density ranges were tested, with an output neuron trained to turn on if the input density fell within its specified range. Attempts were made to delineate densities at 0.1 gm/cm^3 intervals. This technique proved successful for the lower density ranges, but resulted in nonconvergence of models when higher densities were modeled at this small interval. Since delineation of the lower densities is the major concern to surveyors, the ranges used in modeling were (1) 1.1 to 1.2 g/cm^3 , (2) 1.2 to 1.3 g/cm^3 , (3) 1.3 to 1.4 g/cm^3 , and (4) 1.4 or more g/cm^3 . The range from 1.0 to 1.1 corresponds to water and is indicated by failure of the threshold detector to trigger.

Two architectural approaches to the problem were tried: (1) developing 1 overall model with 4 output neurons and 1 set of hidden neurons, and (2) developing 4 independent models each with 1 output neuron and its own independent weight set. The first approach had a much higher success

rate and is the approach documented in this chapter. The best solution using 4 separate models had an 85.7 percent success rate and misclassified 6.5 percent of the cases. The best single model solution had 89.5 percent success rate and did not misclassify any of the cases. The back propagation learning developed a better solution when presented the whole problem rather than 4 separate problems.

7.1 Physical Data

Depth sounder data from Alabama, Georgia, Mississippi, and California provided input for this study. There was a large amount of data available, but very little of it had density ground truth. Ground truth was obtained by nuclear density probes (densities between 1.1 and 1.4) and piston samplers (densities above 1.3). The criteria used for discarding data were (1) no ground truth, (2) clipped signals due to improper amplifier gain, (3) anomalous signals. The anomalous signals discarded involved a reflection at a shallower depth than other signals in the same area and are believed to have been caused by fish. Since there are no typical shapes of the envelope associated with a particular density as there are with materials, typical waveform plots are not included.

7.2 Time Domain Model

As with the fluff and material problems, a time domain solution would be desirable because of ease of hardware

implementation. Relative rather than absolute amplitude was used for the neural network input signal preprocessing due to the variation in amplitude of returns from the same area.

7.2.1 Input Generation

The same keystrokes that were used for fluff and material amplitude processing were used for density classification processing. The processing included using a threshold detector to identify the start of the first return as before.

7.2.2 Model Derived

The density classification problem proved similar to the material classification in many ways. Specific materials exhibit ranges of density which overlap those of other materials; so this density determination is a separate but similar problem. When momentum was set to zero, solutions could only be found for 0 hidden layers, no matter what learning rate was used. Tables for the 16 element single layer and the 17 and 9 neuron two layer models are presented. Tables 43 through 45 list the 64 input element results. Tables 46 through 48 show the 32 input element results.

First momentum was set to 0.3 and learning rate was varied as shown in the next 3 tables. Table 43 gives the number of iterations required to converge to an error rate of less than 0.1 percent on the learning cases for 64 input

elements. The maximum number of iterations was limited to 50000 and in some cases the 0 hidden layer model terminated at error rates between 0.15 percent and 0.35 percent.

Table 43.--Ampl Dens 64-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.1	26889	3879	2589
0.2	13435	1953	1293
0.4	6708	1008	662
0.6	4466	698	453
0.8	3345	539	347
1	2673	445	278

Table 44 gives the average percent uncertainty for the correctly identified cases.

Table 44.--Ampl Dens 64-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.1	3.38	3.43	2.68
0.2	3.44	3.4	2.66
0.4	3.34	3.41	2.6
0.6	3.34	3.34	2.57
0.8	3.33	3.29	2.49
1	3.32	3.14	2.48

Table 45 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 45.--Ampl Dens 64-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.1	84.14	87.63	86.02
0.2	84.41	87.63	86.02
0.4	83.87	88.17	86.02
0.6	83.87	88.17	86.02
0.8	83.87	88.17	85.75
1	83.87	87.9	85.75

The weight maps showed that the last 30 weights of the first layer were very small in magnitude. This implied that the last 30 points of the input were not aiding the classification effort. Therefore, the last 32 (the nearest power of two) were truncated and the momentum held at 0.3 as shown in the next 3 tables. Table 46 gives the number of iterations required for convergence to 0.1 percent error for 32 input neurons.

Table 46.--Ampl Dens 32-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.4	7131	1059	No Conv
0.8	3558	543	40295
1	2844	449	7470
1.3	2184	408	1800
1.4	2027	380	2598
1.5	1891	377	13623
1.6	1771	300	No Conv
1.7	1666	282	No Conv

Table 47 gives the average uncertainty versus learning rate for 32 input elements.

Table 47.--Ampl Dens 32-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layers % Uncertain	2 Hidden Layers % Uncertain
0.4	3.26	4.18	No Conv
0.8	3.24	4.32	3.02
1	3.23	4.22	3
1.3	3.22	4.08	2.66
1.4	4.81	4.22	2.82
1.5	4.81	3.96	3.12
1.6	4.78	4.57	No Conv
1.7	4.78	3.42	No Conv

Table 48 gives the percent of test cases correctly identified as learning rate is varied for 32 input elements.

Table 48.--Ampl Dens 32-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.4	77.7	87.1	No Conv
0.8	77.7	87.9	89
1	77.7	87.9	89
1.3	77.7	89	89
1.4	86.6	89	89.3
1.5	86.6	89.5	88.2
1.6	86.3	88.2	No Conv
1.7	86.3	79.6	No Conv

The most abstract model had 1 hidden layer at a learning rate of 1.5; varying momentum from 0.3 did not improve the solution (however, fine tuning the momentum did enable the 2 hidden layer network to perform as well as the 1 hidden layer network). The hidden layer had 12 elements. The model was taught on one set of 20 signals and tested on a separate set of 372 signals. 89.5 percent of the signals (333) were correctly classified. No signals were incorrectly classified. 39 signals were unclassified (i.e., two output neurons were active or no output neurons were active).

The weights for the hidden layer were

(IN01.01-IN01.32) FaplgCHLjIAhCaFAaCCHHCCGEceeeccCA,
 (IN02.01-IN02.32) gfGCAdowEkeJDHCGLHMIHIIFDDECCBBA,
 (IN03.01-IN03.32) deceedgjbbebbecdADBbgfcdfaFGFDBAc,

(IN04.01-IN04.32) dfdfgceiECJMBHEGIGIEDFFCBCDAaaaa,
 (IN05.01-IN05.32) CHOPKDbdiptochhgfefdddddcbaaCCCCB,
 (IN06.01-IN06.32) cbCBabeIBdaDACABDCEDCDDCAABAAaAA,
 (IN07.01-IN07.32) gfHECCmqIgAJBFABCaCacCCacBDCBBCA,
 (IN08.01-IN08.32) cfcddfiijcgfccbdaBAdgfcdeaFGGECBB,
 (IN09.01-IN09.32) deacdejlcfdeceADBdjhdegAHJIFDBC,
 (IN10.01-IN10.32) hFZZQEpuMwjKDBjdcgcgiabggCGHFFGB,
 (IN11.01-IN11.32) cgiieACIMRTIdabbfekllhiifABabaac,

and

(IN12.01-IN12.32) hiAdcdljQKQPAFbBacgmmegjfEGECBBb.

The weights for the output layer were

(IN1.01-IN1.12) ZXqNvHBPzvzz,
 (IN2.01-IN2.12) zZQMzCXZYZNZ,
 (IN3.01-IN3.12) PzpqqnmCzzZS,

and

(IN4.01-IN4.12) LczzZzQZzXhI.

The biases for the hidden layer are

CikdcejlkCe.

The biases for the output layer are

jzdl.

7.3 Frequency Domain Model

The next set of inputs were generated in the time-frequency (T/F) domain. This approach is good in that it combines the best of the time and frequency domains. Frequency gives energy content without concern as to where in the sample ensemble the energy was concentrated. Time

separates the samples into bins according to depth.

7.3.1 Input Generation

The same keystrokes used for fluff time-frequency (T/F) processing were used for the density classification processing.

7.3.2 Model Derived

Tables for the 16 element single layer and the 17 and 9 neuron 2 layer models are presented. Tables 49 through 51 are the group of tables for the 64 input element neural network model results. First momentum was set to 0.3 and learning rate was varied. Table 49 gives the number of iterations required to converge to a learning case error rate of less than 0.1 percent for 64 input elements. The maximum number of iterations was limited to 50000 as before.

Table 49.--T/F Dens 64-Iterations Vs. Learn Rate

Learning Rate	0 Hidden Layers # Passes	1 Hidden Layer # Passes	2 Hidden Layers # Passes
0.2	27803	2653	1408
0.4	13897	1342	723
0.6	9263	893	503
0.8	6948	664	374
1	5560	525	311
2	2791	282	484

Table 50 gives the average percent uncertainty for the

correctly identified cases.

Table 50.--T/F Dens 64-Uncertainty Vs. Learn Rate

Learning Rate	0 Hidden Layers % Uncertain	1 Hidden Layer % Uncertain	2 Hidden Layers % Uncertain
0.2	2.64	3.34	2.84
0.4	2.63	3.36	2.89
0.6	2.62	3.3	2.82
0.8	2.62	3.19	2.72
1	2.61	3.09	2.8
2	2.51	2.52	2.75

Table 51 gives the percent of the test cases that were correctly identified for 64 input elements.

Table 51.--T/F Dens 64-Correct Vs. Learn Rate

Learning Rate	0 Hidden Layers % Correct	1 Hidden Layer % Correct	2 Hidden Layers % Correct
0.2	83.6	79.3	83.87
0.4	83.6	79.3	84.14
0.6	83.6	78.76	83.87
0.8	83.6	77.69	83.6
1	83.6	77.15	84.68
2	83.06	69.62	80.65

The number of inputs was truncated to 32 as with the amplitude model, but the results were not as good as with the 64 input model. The best 32 input solution was 83.87

percent, less than the 64 input solution.

Next a set of 64 inputs composed of combinations of the 32 truncated amplitude and the 32 truncated frequency inputs were modeled. The best solution for the combined set was 84.14 percent correct.

The best frequency solution exhibited an 84.7 percent correct classification, as contrasted with an 89.5 percent for the amplitude model. This case occurred with the 2 hidden layer model developed with learning rate set to 1.0 and momentum set to 0.3. Since this model was vastly inferior to the amplitude model, its weights are not listed. The superiority of the amplitude model may possibly be attributed to the ability of neural networks to develop their own transforms, which may provide a form of mapping that is superior to time-frequency for this class of problems.

CHAPTER VIII

RESULTS

To demonstrate that this research has, in fact, advanced the state-of-the-art of hydrographic surveying the accuracy of the neural network approach is compared to the accuracy of the conventional approach, when the problem has been addressed with success by conventional methods. Numerous attempts by conventional methods have been made at developing a system that would interpret whether a layer of fluff overlays the bottom, but all have resorted to human interpretation to make the decision. A number of conventional, and some rather unconventional, material and density classification systems have been demonstrated to the author of this dissertation but only 1 mathematical solution has proven successful.

8.1 Fluff Detection

Fluff, also called suspended sediment, poses no obstacle to navigation, but reflects the depth sounder signal causing the depth sounder to indicate the bottom as the top of the fluff layer. This material does not need to be dredged; thus, determination of areas which need to be sounded by nonacoustic means is of economic concern.

8.1.1 Previous Attempts

Research has been conducted for more than 20 years in the field of acoustic signal analysis by the U.S. Army Engineer Waterways Experiment Station. During this period a number of hardware devices were constructed and tested. Density hardware tested was considered too complicated and too inaccurate for use. Currently a fluff detection system is being tested that basically displays the envelope of the return signal and leaves the determination of whether the signal was fluff or a hard bottom to a human observer. This device is a step in the right direction, but does not meet the desired goal of indicating presence or absence of fluff. There are no plans to develop the device beyond its current display of relative envelope display. The only fluff detection units currently under investigation thus rely on the judgement of a human being based on less than conclusive graphical presentations.

8.1.2 Neural Network Model

A neural network model with 2 output neurons which indicate presence or absence of fluff was developed. Since the outputs were complimentary, one neuron would have sufficed, but using 2 gave redundancy (if both were on or both were off the results would be inconclusive). The best model had 64 inputs, a hidden layer of 16 neurons, and an output layer of 2 neurons. All of the test cases (obtained from a different state than the training cases) were

correctly classified. The weights and biases obtained are given at the end of Chapter 5. A fluff or hard bottom breakdown and the overall results are shown in Table 52.

Table 52.--Fluff Detection Results

Material	% Correct
Fluff	100
Bottom	100
Overall	100

8.2 Material Classification

The majority of material which must be dredged is in the form of clays and sands. Hard clays and sands wear on the cutting edge; soft clays tend to coat the cutter blades and clog them. The type of material is of interest to dredgers in planning the method of dredging.

8.2.1 Conventional Method

The only conventional bottom classification system which has been successfully demonstrated to the author relies on the fact that the most commonly encountered bottom materials in this country fall into certain density ranges; thus if the density of the bottom is determined mathematically then a table lookup would yield the most likely material type. The tables used for relating density to material were primarily developed by the research of

Hamilton [REF 43-45]. This method relies on proper density determination using mathematical modeling. The density determination method relies on taking a group of samples to gain more degrees of freedom; therefore, the results of this method cannot be given on an individual sample basis for comparison purposes. The same test set that was used in testing the neural network model was used to test the mathematical model. The soft clay samples were classified as silty clay, the silty clay samples were classified as silty clay, and the silty sand samples were classified as sandy silt. The first group of samples should have been classified as clay by the notation used by the program. The parameters for the mathematical method for these cases were determined by the mathematical model's developers. Thus, the mathematical method was 66.67 percent correct on the test set. It was 100 percent correct on the harder bottom types.

8.2.2 Neural Networks

The neural model was developed using one test set from California and tested on a separate set from California. The best model tested used 64 time/frequency inputs and correctly classified 96.64 percent of the cases. The model had 1 hidden layer of 9 neurons and 3 output neurons corresponding to (1) soft clay, (2) silty clay, and (3) silty sand.

8.2.3 Comparison with Conventional

For the test set the neural network approach had a 96.64 percent success rate as opposed to 66.67 percent for the mathematical approach. This may be due to the fact that particular materials exhibit particular shapes of return envelopes (REF 18). Also, material classification is a recent addition to the mathematical method and may be improved upon in future releases of the mathematical software. Table 53 shows a breakdown by material type and the overall results. The fact that the mathematical model averages a number of signals to get its classification resulted in an all correct or all incorrect material determination on the test case set.

Table 53.--Material Classification Results

Material	Neural Network % Correct	Math Model % Correct
Silty Clay	90.5	100
Silty Sand	96.3	100
Clay	99	0
Overall	96.64	66.67

8.3 Density Classification

The determination of density of material composing the bottom is of interest both to dredgers and to the oil

industry. The density of the material is related to rate at which material may be pumped by the dredge and may indicate the possible presence of oil.

8.3.1 Conventional Method

Numerous attempts have been made to determine the density of the bottom from acoustic returns. One major modeling problem is that there are 2 unknowns: density and the velocity of the acoustic propagation. The exact equations used for modeling the underwater saturated media are not known to the author since the software used is a commercial product. However, the steps used in processing the acoustic return are (1) computation of relative total energy, (2) estimating signal to noise ratio, (3) calculating reflectivity, (4) calculating acoustic impedance, (5) estimating the acoustic velocity, and (6) calculating density. The program starts with the velocity of sound in water and adjusts this velocity based on the density it calculates for the material at each interface [REF 20]. The program averages a number of signals in the frequency domain to gain degrees of freedom; therefore, results cannot be compared on a signal by signal basis.

The same set of data used for testing the neural network models was used for testing the conventional method software. For the 1.1 to 1.2 gm/cm³, the program classified 11.1 percent correctly, classified none incorrectly, and did not classify 88.9 percent because of poor signal to noise

ratio. The 1.2-1.3 gm/cm³ set was identified as 1.39 gm/cm³. None of the 1.3-1.4 gm/cm³ set were classified due to signal to noise ratio. 80 percent of the 1.4 and up gm/cm³ signals were classified correctly; the other 20 percent were not classified due to signal to noise ratio. Overall 41 percent of the signals were correctly classified, 9 percent were incorrectly classified, and 50 percent were not classified due to signal to noise ratio. The 9 percent that were incorrectly classified were within the 0.15 gm/cm³ error band of the software. Thus, all signals that were classified were classified within the typical error band of the software. Every effort was made to see that the mathematical model was given the proper parameters; however, since the parameters were not selected by the mathematical model's developer (in some cases), it is possible that the results could be improved by better refinement of the parameters.

8.3.2 Neural Networks

The best neural network model classified 89.5 percent of the test cases correctly and failed to classify 10.5 percent of the test cases. This network used 32 amplitude inputs with 1 hidden layer of 12 neurons and a 4 neuron output layer.

8.3.3 Comparison with Conventional

Both methods correctly classified all test cases that

were classified within a small error band. The neural network classified more of the cases correctly than the conventional method. Therefore, for the test set used, the neural method outperformed the conventional method. A breakdown of the percent correctly identified by density range and overall for the neural network model and the math model (conventional method) is shown in Table 54. There were more test cases in the range above 1.4 gm/cm³; so, the overall success of the math model is higher than it would have been if equal percentages were included for each range. The upper range had more cases because it is an unbounded band and is thus wider than the others. The conventional method involved more calculations than the neural method and took at least an order of magnitude more time to complete the calculations (i.e., between 20 and 100 seconds).

Table 54.--Density Classification Results

Density (gm/cm ³)	Neural Network % Correct	Math Model % Correct
1.1-1.2	98.7	11.1
1.2-1.3	82.4	0
1.3-1.4	90.5	0
>1.4	83.8	80
Overall	89.5	41

CHAPTER IX

PROPOSED HARDWARE IMPLEMENTATION

One goal of this research was to devise a method which could be implemented in hardware real-time. Rather than treat each problem's hardware independently, since the only difference between the models is the set of weight and bias parameters used, hardware suitable for all three problems will be presented. The hardware presented will address the time domain solution, since it was the best solution for 2 of the 3 problems and was only slightly inferior to the time-frequency solution for the third. To implement the time-frequency solution would involve adding digital signal processing chips prior to the neural network inputs.

The hardware would attach to a conventional depth sounder at 2 points: (1) the returned envelope signal output and (2) the synchronization pulse output. The returned envelope corresponds to the rectified and filtered returned signal, which has the shape of the envelope of the reflected energy. The synchronization pulse is a TTL level pulse which corresponds in time to the start of the emitted burst of acoustic energy.

9.1 Returned Envelope Circuitry

The returned envelope circuitry is basically analog and is shown in overview in Figure 34. The circuit is basically a sampled data system, but is analog rather than digital.

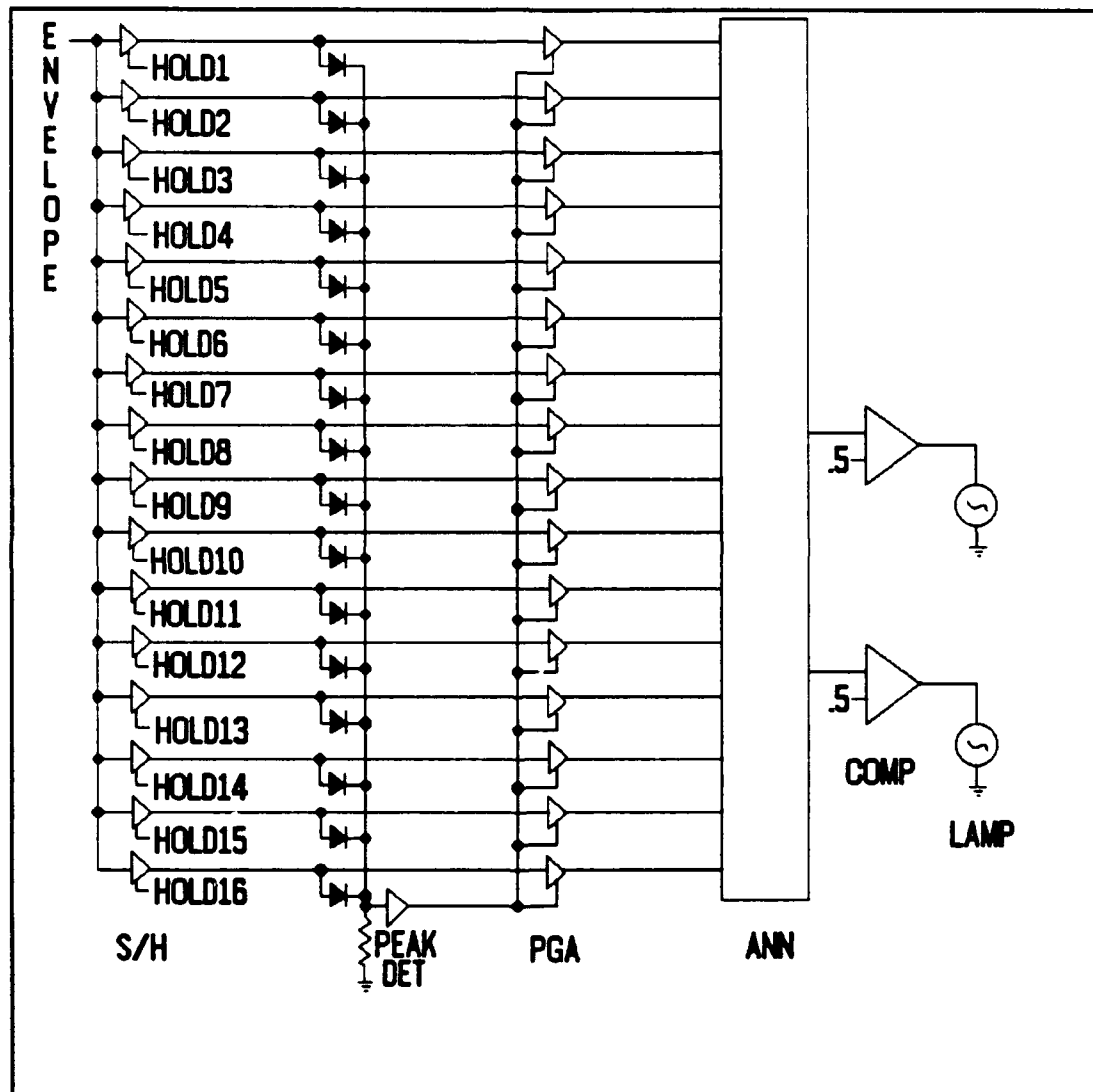


Figure 34. Envelope Circuitry

There is a sample and hold amplifier for each of the neural network inputs. A 16 input system is presented in Figure 34; for a 32 or 64 input system the circuits are replicated

for the additional elements. The holds are driven sequentially at fixed intervals controlled by the synch pulse circuitry. Thus, the first stage of the circuitry captures the analog values of the waveform at 16 points in time. The second stage detects the peak value of these points. Since all points are positive, the peak detection circuit is a simple diode circuit that allows the largest held value to pass through to the resistor, while the remaining values are blocked by reverse bias. The output of the resistor is buffered by a unity gain amplifier that is biased to remove the effect of the active diode's forward drop potential. This output voltage is used to scale all the inputs, such that, their values fall between 0.0 and 1.0 volts. This is accomplished by using a multiplying, programmable gain amplifier (PGA) for each input to scale the input by ratioing it to the output of the peak detector.

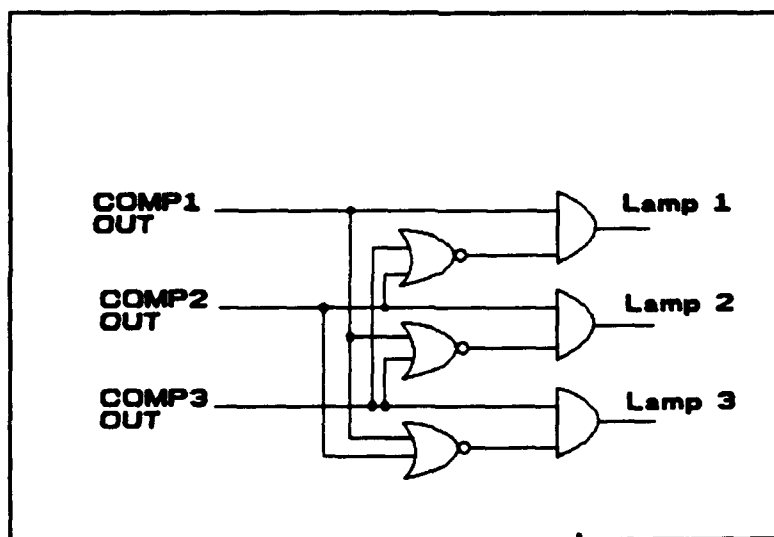


Figure 35. Uncertainty Detect Circuitry

If it is desired to have the indicator lamps not illuminate when the results are indeterminate, the logic of Figure 35 may be added. This circuit allows the indicator lamp associated with the output of a particular neuron to illuminate only if no other output neuron is simultaneously active.

9.2 Sync Circuitry

The sync pulse circuitry is basically digital. It is shown in Figure 36. The circuit consists mainly of a sophisticated gated counter. The counter begins counting at the first instant that the envelope signal exceeds a predetermined threshold voltage, after a delay to allow the surface return to dissipate. The counter counts for 16 counts at a rate set by the oscillator (OSC). Then the circuit is disabled until the next cycle begins.

A cycle begins when the depth sounder pulses the sync line indicating the sending of a pulse of energy. A 74221 one-shot provides a fixed delay of duration set by a potentiometer/capacitor combination to correspond to the time recommended by the depth sounder vendor to allow the surface reflections to pass.

The output of this one-shot indicates cycle start. At this time the output of the center most D flip flop (FF) is low disabling the gated clock AND gate (it was set low by the sync pulse). A small delay provided by the second one-shot in the 74221 is provided to prevent this flip flop from

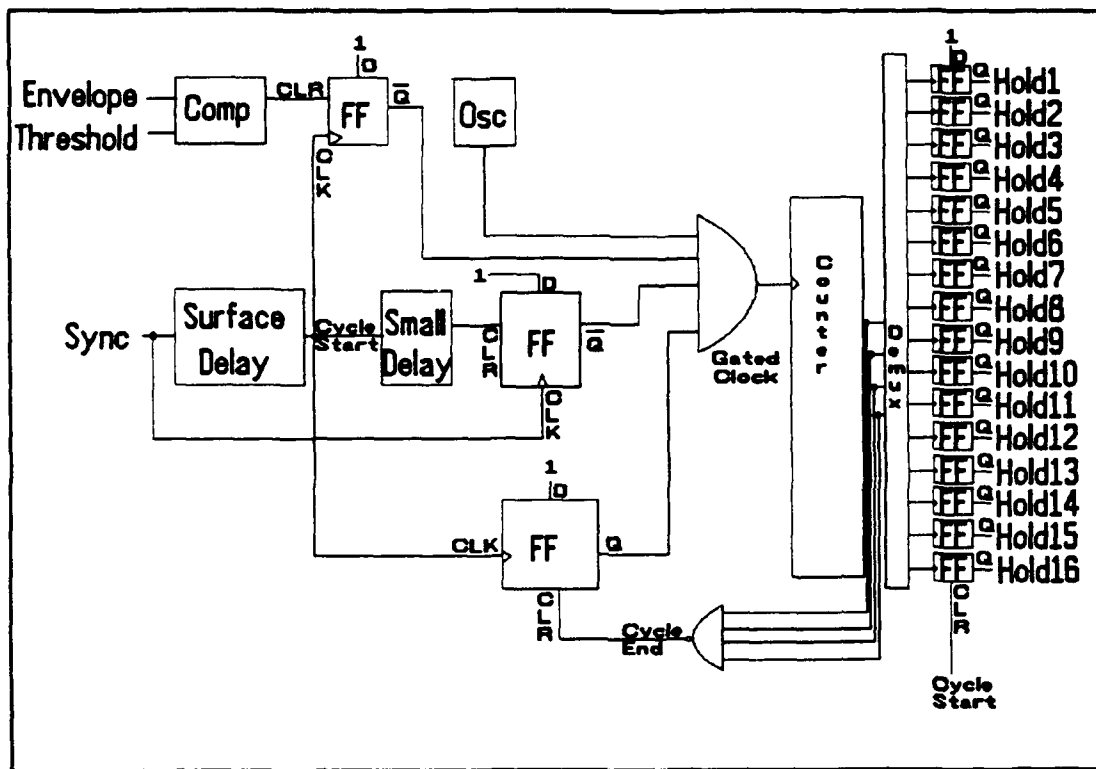


Figure 36. Sync Pulse Logic

re-enabling while the other 2 flip flops are being set by the cycle start command. The cycle start command also resets the output flip flops enabling them for the next cycle. At the end of the small delay the outputs of the lower 2 flip flops are high; so only the upper flip flop blocks the clock. Whenever the envelope signal exceeds the predetermined threshold, the output of the upper flip flop goes high enabling the clock to pass through the AND gate. The counter counts 16 times before the count of 15 disables the gate circuit by resetting the lower flip flop, so that, it blocks the gate. The output of the counter is demultiplexed, so that, each output goes low momentarily clocking the 16 output flip flops on 1 at a time,

sequentially. Thus, the upper most flip flop turns the count on and the lower most flip flop turns the count off.

9.3 Hardware Artificial Neurons

The proposed artificial neural network implementation device is the 80170NW made by Intel Corporation [REF 51]. The device layout is shown in Figure 37. The 80170NW has been preprogrammed to the desired weights and biases and is operated in the forward pass mode. The only pins of concern in this mode are the 64 analog inputs and the 64 analog outputs. The reason all 64 of these pins are shown is that this diagram was produced by Tango Schematic (trademark of Accel Corporation) and is functionally complete for the purpose of producing the wire list which Tango Route (trademark of Accel Corporation) uses to autoroute the printed circuit card. The outputs are fed back as inputs so that one chip may provide two layers of neurons. Use of a single chip to serve as two layers requires time-multiplexed operation. That is, the chip first serves as the hidden layer with inputs to the neurons tied to the chip inputs; then the outputs are latched to provide inputs to the next pass, where the neurons serve as the output layer. During the second pass, the inputs of the neurons are connected to the latched outputs from the first pass and disconnected from the chip inputs. In order that the outputs not be time-multiplexed for networks with two layers, two chips are used; therefore, the outputs are always dedicated (i.e., the

outputs of the first chip are the outputs of the hidden layer and the outputs of the second chip are the outputs of the output layer). Thus, the input and output buffers are always enabled and the feedback buffers are always disabled. Used in this manner, 64 analog inputs are connected through 4096 synapses to 64 neurons providing 64 dedicated outputs. The input synapses are set to the weights designed by the computer modeling. The input biases are combined to yield the biases designed by the computer modeling. The feedback synapses and biases are set to zero. The sigmoids closely

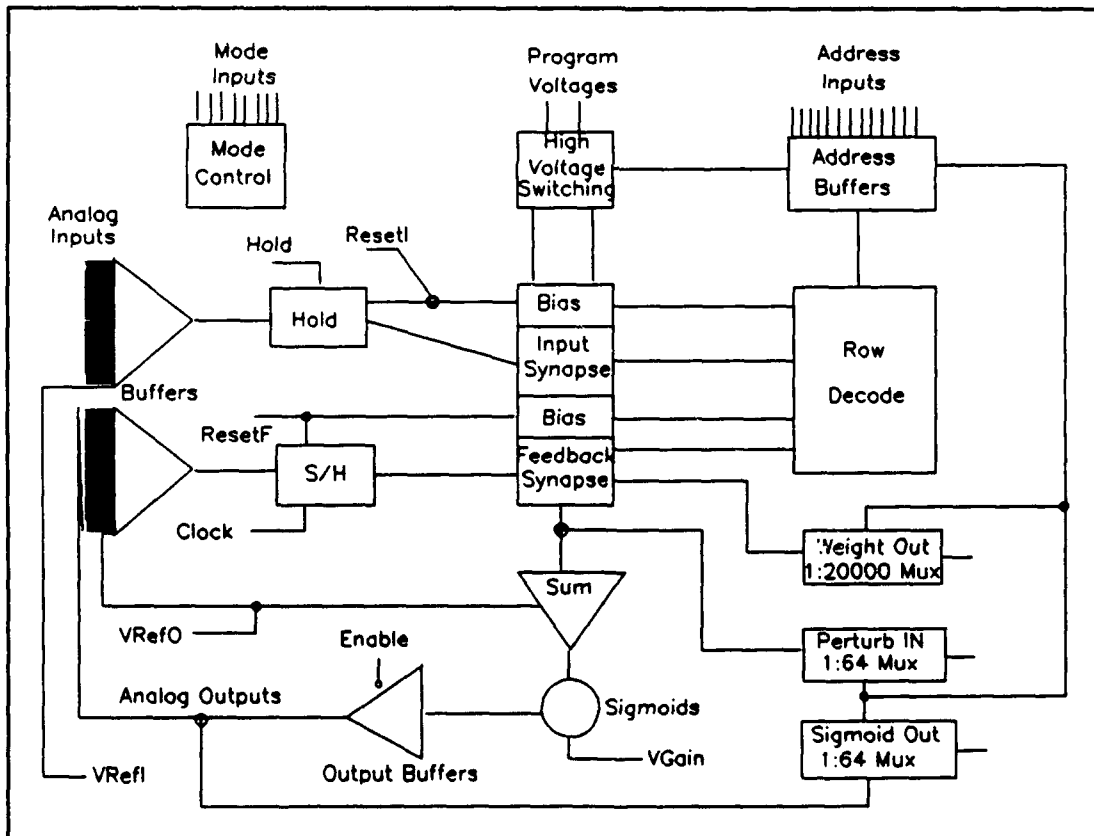


Figure 37. 80170NW

mimic a true sigmoid, but to account for their imperfection,

Intel recommends that its software be used to determine weight and bias values (as it takes the sigmoid imperfections into consideration). The chip currently sells for \$2000 with \$100 per chip charged for programming the weights and biases. A complete development system is available at a cost of approximately \$18000.

Although, the Intel 80170NW chip is the device proposed for the prototype implementation, there are other neural chips which are currently available or will be available shortly. One such chip will be the N1000 designed by Nestor Incorporated (Providence, Rhode Island) under a \$1.2 million contract from Defense Advanced Research Projects Agency [REF 52]. Another chip, the Angle-Modulated Exponential Operator Neural Network (AXON), is a 40 neuron chip being developed by General Dynamics [REF 53]. Neural network chips basically come in 2 types: analog and digital. Analog chips can potentially pack more neurons per chip, since the interconnection problem is lesser with analog implementations. An analog signal requires 1 wire, while a digital signal requires a separate wire for each bit of the digital word representing the value. Digital circuits can be more accurate as they are not subject to drift and other problems that shift analog values. Japan is very active in neural network research and is marketing or developing chips of both types. Digital chips range from a 7-bit, 1 neuron chip from Ricoh to an 18-bit, 576 neuron chip from Hitachi.

Analog chips range from a 1 neuron chip from Fujitso to a 125 neuron chip from Mitsubishi. There is even a 90 neuron, optical chip under development by Mitsubishi (REF 54).

Neural networks can also be constructed from commonly available components. They have been constructed using digital signal processing chips or transputers. A simple implementation of a perceptron neuron is shown in Figure 38. This approach involves using analog operational amplifiers (OP AMPs) followed by comparators.

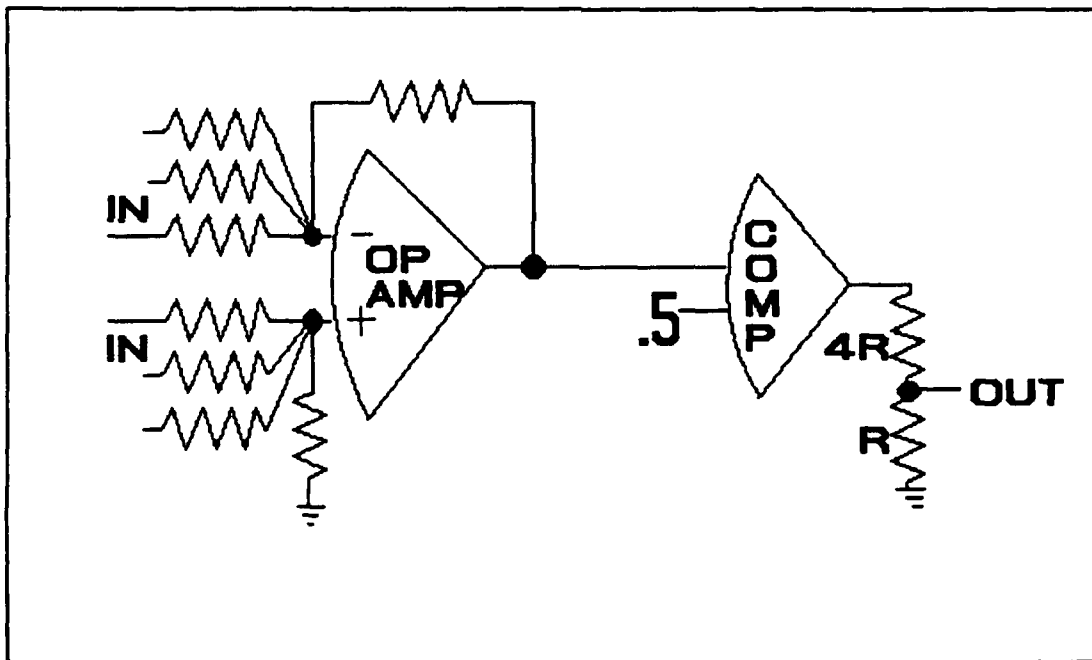


Figure 38. Op Amp/Comparator Neuron

The plus summing junction accepts exciting inputs and the minus summing junction accepts inhibiting inputs. The comparator converts the output to digital. Since comparator outputs usually range from 0 to 5 volts, a resistor divider on the output is used to change the digital values to 0 and 1.

CHAPTER X

SUMMARY AND CONCLUSIONS

To advance the state-of-the-art of hydrographic surveying, 3 problems that have accompanied the use of acoustic depth sounders for hydrographic surveying were addressed with neural network modeling. The problems considered were (1) fluff detection, (2) material classification, and (3) density classification. For use as classifiers the input signals were scaled to between 0.0 and 1.0, and the output neuron associated with each classification was considered active if its value exceeded 0.5. For example, if the output neuron for silty sand had an activity level of greater than 0.5 and all other output neurons had activity levels less than 0.5, the network was said to have classified the material as silty sand.

In investigating the use of neural networks to solve these problems, models for hundreds of combinations of hidden layer sizes, learning rates, and momentum were developed and tested. The number of hidden layers that worked best was a number which caused a uniform fanout or fanin between input and output elements; difficult problems required reducing the number of elements in the hidden layers to obtain convergence. A single model with multiple

output neurons gave better results than multiple models with single output neurons on the same test sets. The error of each multiple model was less than the overall error of the single model; however, when the test cases for all models were considered as a whole (as they should be), the single model correctly classified more cases.

One of the aims of the research was to produce abstract solutions. Abstraction allows networks to be trained on data acquired at one site and used to classify data from another site. For the problems considered, the most abstract solutions (i.e., the ones that performed best on a test set composed of data from another site) were obtained when learning rate and momentum were chosen so as to form a nearly critically damped solution. If the minimum number of iterations required to reach a certain level of error on the learning set is considered the critically damped point, then slightly increasing either learning rate or momentum caused the number of iterations to increase due to the overcorrecting for the error (underdamped). Conversely, slightly decreasing the learning rate or momentum caused the number of iterations to increase due to undercorrecting for error (overdamped). The implication of this observation is that there is no need to examine combinations of learning rate and momentum that have extremely long learning times. Further, when identifying a test set with unknown characteristics, choosing the critically damped case of the

learning set would be a reasonable selection criteria for expecting good results on the test case set.

The first problem considered was fluff (suspended sediment) layer detection. Both amplitude and time-frequency models proved 100 percent successful on identifying the test case set. The learning set of 85 cases was constructed of data collected in Georgia of both hard bottom and fluff signals with a 20 kHz transducer. The test set of 239 samples was constructed of data collected in Mississippi and Alabama and consisted of data collected with both 7 and 20 kHz transducers. The model was abstract enough to not only identify data from other sites, but also other transducers. Since both time (amplitude) and time-frequency models exhibited 100 percent success, the time model with no hidden layers was chosen as having the best hardware potential because of the reduced complexity. There is no similar conventional method to use for comparison; so the development of a fluff detection method which can be implemented real-time in hardware greatly advances the state-of-the art of hydrographic surveying in regions where fluff may be a present in ship channels.

The second problem considered was classification of bottom materials: (1) hard silty sand, (2) soft clay, or (3) hard silty clay. Models in both the time (amplitude) and time-frequency domains were developed. The most abstract model developed had 1 hidden layer with a learning rate of

0.4 and a momentum of 0.6. The 64 inputs were from the time-frequency domain and consisted of 16 sets of overlapped, lower frequency 4 amplitudes (from an 8 amplitude FFT). The hidden layer had 8 neurons. The only data with known bottom material was from California from a 7 kHz transducer. The model was taught on one set of 39 signals and tested on a separate set of 149 signals. A total of 96.64 percent of the signals (144) were correctly classified. No signals were incorrectly classified. Five signals were unclassified (i.e., 2 output neurons were active or no output neurons were active). The neural network method outperformed the conventional method which correctly classified 66.7 percent and incorrectly classified 33.3 percent of the cases.

The third problem considered was classification of bottom material by density range: (1) 1.1 to 1.2 g/cm³, (2) 1.2 to 1.3 g/cm³, (3) 1.3 to 1.4 g/cm³, and (4) 1.4 and up g/cm³. Data from California and Georgia were used to train the network and data from California, Georgia, Alabama, and Mississippi were used to test the network. Signals for a particular density range of the training set were taken from a single site; signals for a particular density range of the test set were taken from several sites. The most abstract model had 1 hidden layer at a learning rate of 1.5. The hidden layer had 12 elements. The model was taught on one set of 20 signals and tested on a separate set of 372

signals. A total of 89.5 percent of the signals (333) were correctly classified. No signals were incorrectly classified; however, 39 signals were unclassified. This was significantly better than the conventional method that classified 41 percent correctly, classified 9 percent incorrectly, and did not classify 50 percent.

Neither the neural network approach nor the conventional mathematical modeling approach (for the particular set of test set data and neural network models developed in this research) classified all signals. The neural network did not classify signals of which it was uncertain (i.e., no single output neuron was excited above the 0.5 threshold); the mathematical model either classified the material as water or did not attempt a classification when signal to noise level was poor. The neural network method did not exhibit an error band, whereas the mathematical model typically exhibited an error band of 0.15 gm/cm^3 . The mathematical approach gave continuous density predictions even in the higher density ranges. A combined approach would improve the predictions over either method alone. The neural network method could be used for classifying densities below 1.4 gm/cm^3 and the conventional mathematical model used for classifying densities above 1.4 gm/cm^3 .

Although it was not the sole aim of this research to advance the state-of-the-art of neural network research, an

interesting behavior was uncovered which deserves further investigation in a future study. It was found that if the 1 hidden layer network learning cycle was initialized with the weights and biases for the output layer of the no hidden layer solution (and the hidden layer initialized to small random numbers), convergence could be obtained more than 20 times as rapidly (in most cases) than if the output layer weights were initially set to small random numbers. The quality of the solution obtained was as good or better (90 percent correct on the test case set instead of 50 percent correct for some learning rates and momentum combinations), and convergence could be obtained in many cases where initialization with small random numbers resulted in the learning stagnating in a local minima. This approach works well when extended to additional hidden layers; a 7 neuron single hidden layer model could provide an excellent starting point for a 16 first and 7 second neuron 2 hidden layer model. The delta learning rule is more exact than the back propagation learning rule and apparently pretrains the weights to superior starting values. This behavior is peculiar in that the weight arrangement matrix for differing layers of hidden layers is quite different. In fact, reuse would not have been possible had not the weights been treated as a 1 dimensional rather than a 2 dimensional array in the BACKPROP program. Therefore, this phenomena is presented solely as a topic for future investigation.

Additional neural network observations for this class of problem include: (1) adding additional layers beyond 1 may not improve the solution, (2) the small random numbers used for initialization of weights and biases should be very small (0.0000001 was the maximum for this research) for convergence, (3) the number of neurons in the hidden layers should be kept small and provide a hierarchical topology for convergence.

Recommendations for further research in the neural network field include investigation of the critical damping criteria for selecting learning rate and momentum in other classes of problems. Also pretraining the output layer using no hidden layers as a starting point for developing single hidden layer models needs further investigation. Recommendations for further research in the hydrographic surveying field include implementing and testing the hardware designs. Further research might also include a network to identify the type and quantity of fish below a boat based on acoustic returns. Also, this research used a conventional depth sounder; wide band signal sources should also be investigated as they have a potential of providing better accuracy.

APPENDIX A
PRENEURA

The PRENEURA program is a program used to condition waveforms prior to input to the neural network program, BACKPROP. It displays the current waveform at all times. The initial waveform may be modified and reduced in number of points by the following commands (a key which is depressed to initiate a command is italicized:

A auto save in file named 'INFILE'.A??

If a file named 'OUTDIGIT' exists, the letter A above in the output file name is replaced by the first character in the file 'OUTDIGIT'. This allows creating different sets of inputs for the BACKPROP program without renaming output files. The file 'OUTDIGIT' is built using a test editor, an example contents would be

Z

to set the filename to 'INFILE'.Z??

B mark the beginning limit

B, E, and C work together. You first mark the limits with B and E and then clip to them with C

C clip to the marked limits

D decimate by 2

E mark the ending limit

F take the Fourier amplitude transform

If you want the Power Spectral Density follow *F* with *W*

G get a new file

Generally using file lists is more convenient, but

G allows doing one file independently

H for help on key functions

I initiate keystroke save or terminate save

The first time you strike *I* the program goes into

record keystroke mode and records all keystrokes

until either *I* is struck again or *K* is struck.

If the sequence is terminated with *K* a recursive pattern is created.

Embedded *H* and terminating *I* are not recorded.

J jumps relative to the current point by predetermined

amount. Jump distance in points is input from a

file called "JUMPDIST"; the distance may be either positive or negative and clips to the screen limits.

It is often used following the threshold command to back up a set number of points. An example file contents might be

-20

K do predetermined Keystrokes or terminate keystroke save with *K*.

This is a powerful means of applying the same key sequence to a number of files automatically. If a list of files is input by the *L* command, a keystroke sequence may be ended by the strokes *A*, *N*, *K*.

When executed with the 1st file on the screen, the *K* command will cause all files in list to be sequentially processed by the keystroke string and saved automatically by the *A* command. *N* advances file to next one and *K* causes the recursive processing. When the last file has been processed the sequence ends. Everything is visible on the screen, so you can sit back and visually observe the effects on your data as the screens fly by.

L get a list of files for sequential processing

This is used to read a file containing a list of files, e.g. a test set. Then the *M* and *N* keys can be used to advance and backup through the list.

M Previous sequential file

N Next sequential file

O Clip to number of points specified in file "CLIPPNTS"

Note this works from the current cursor position

The file "CLIPPNTS" is built using a test editor, an example contents would be

50

to set the clip value to 50 points i.e. present to present+49

P clip to power of two points from current point

Note this works from the current cursor position

and is often used to follow the *T* command

Q quit

R rectify and scale to between 0. and 1.

Values below 0 are set to 0 (half wave rectify) and

then data is scaled to range from 0. to 1.

S save current data in a file

Used in conjunction with the *G* command. Normally *A*

is used to save files loaded with the *L* command.

T jump to the next point above the threshold

The threshold is set in a data statement (currently

.05) and is normally used after the *R* command to

jump to the first scaled point => the threshold.

If a file named "THRESHOL" exists, the ASCII

floating point value in this file is used for the

threshold thus the threshold can be changed without

recompiling. The file "THRESHOL" is built using a

text editor, an example contents would be

0.5

to set the threshold to .5

U unzoom

V square root of data

W square of data

X rectify to mean full-wave

Y scale between 0 & 1 with original mean = .5

Z zoom in on marked limits

. Form envelope of signal via peak detection

0 filter data with coefficients from file "FILTER"

The filter may be either recursive or nonrecursive

The first character of the file "R" or "N" determines

type. The next lines of the file are the

coefficients starting at the present point.

Nonrecursive are assumed balanced and only the center

and one side are entered. For example, the contents

of "FILTER" for a .25 .5 .25 nonrecursive filter

would be

N

0.5

0.25

1 interpolate data spaced at delt to deltnew

If a file named "DELTAS" exists, the ASCII floating

point values in this file is used for delta

and deltnew; deltas can be changed without

recompiling. The file "DELTAS" is built using a text

editor, an example contents would be

0.5

0.2

3 save buffer

5 half wave rectify about the mean

7 restore saved buffer .

9 Clear queue

, append to queue
 / get queue
 - invert data about zero
 ; start/stop recording of subkeys

The first time you strike ; the program goes into record subkeystroke mode and records all keystrokes until either ; is struck again or = is struck. Embedded "H" and terminating ";" are not recorded. The keystrokes are saved in a file called "SUBSTROK."

= playback sub keys

This is a powerful means of applying the same key sequence to a number of files automatically. This is used like a subroutine to playback a set of keystrokes that will be applied repeatedly to the same waveform. When used with the queue feature then time/frequency neural net inputs may be created by doing FFT's along the time axis. Everything is visible on the screen, so you can sit back and visually observe the effects on your data as the screens fly by.

RIGHT ARROW moves cursor right along data.

UP ARROW moves cursor left along data.

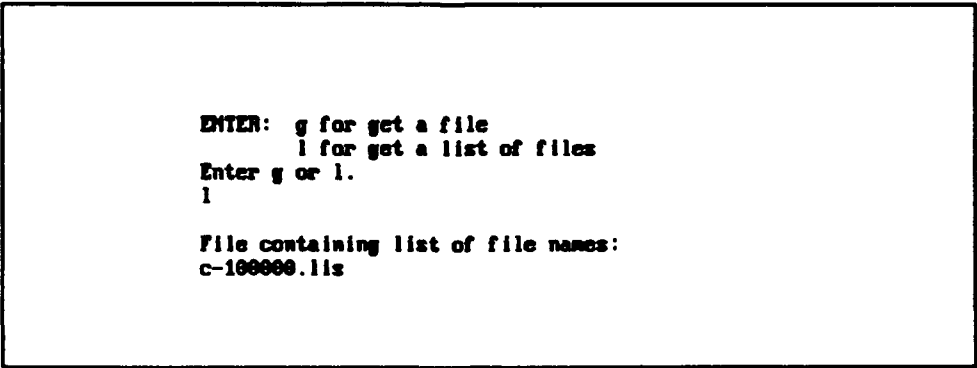
SPACE BAR redraws the screen.

UP ARROW increments the jump size (in powers of 2) that is used by the left and right arrow commands.

DOWN ARROW increments the jump size (in powers of 2) that is used by the left and right arrow commands.

A.1 PRENEURA Example Run 1

The use of keystroke sequences to condition data for the neural networks program might best be illustrated by an example. The following is raw data which is clipped on one side and unrectified. Suppose we wished to transform it into the square root of the unity FFT of the envelope of the unclipped side. In this example the unclipped side is the positive side; if it were the negative side the waveform would be inverted by the - keystroke as the first processing step. The program is started by typing "PRENEURA." At this point the user is asked to enter either a single data file name or the name of a file containing a list of data files. In this case the list option was used (Figure 39).



```
ENTER: g for get a file
       l for get a list of files
Enter g or l.
l

File containing list of file names:
c-100000.lis
```

Figure 39. Initial File Entry

Once the file list is input, the program displays the

contents of the first data file in the list (Figure 40).

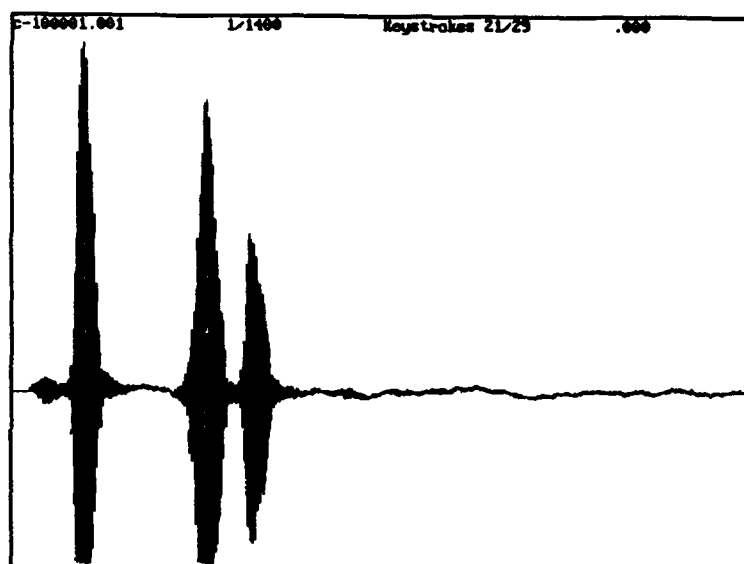


Figure 40. Raw Data

Striking any key will return to the data screen. Entering 5 will half wave rectify the wave about its mean (Figure 41).

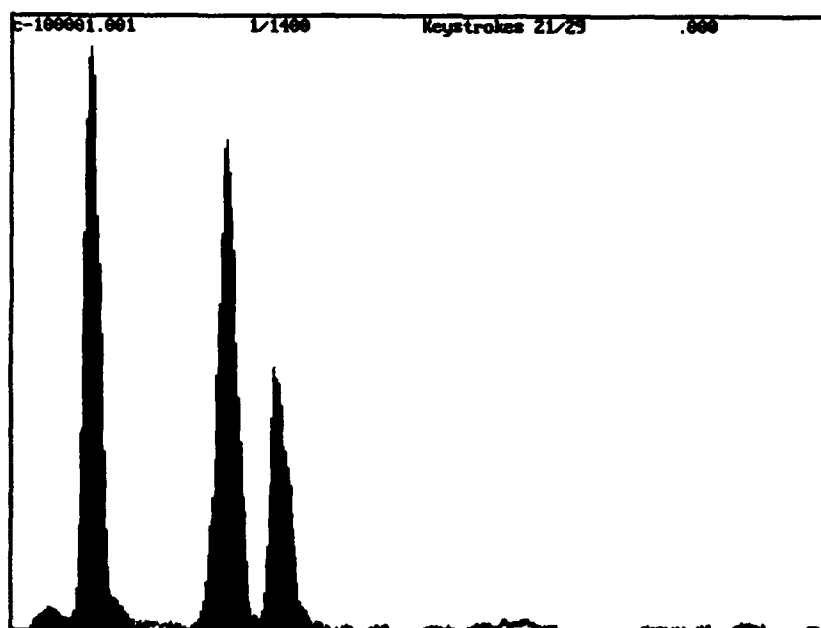


Figure 41. Half-wave Rectified

Striking . will form the envelope, Figure 42.

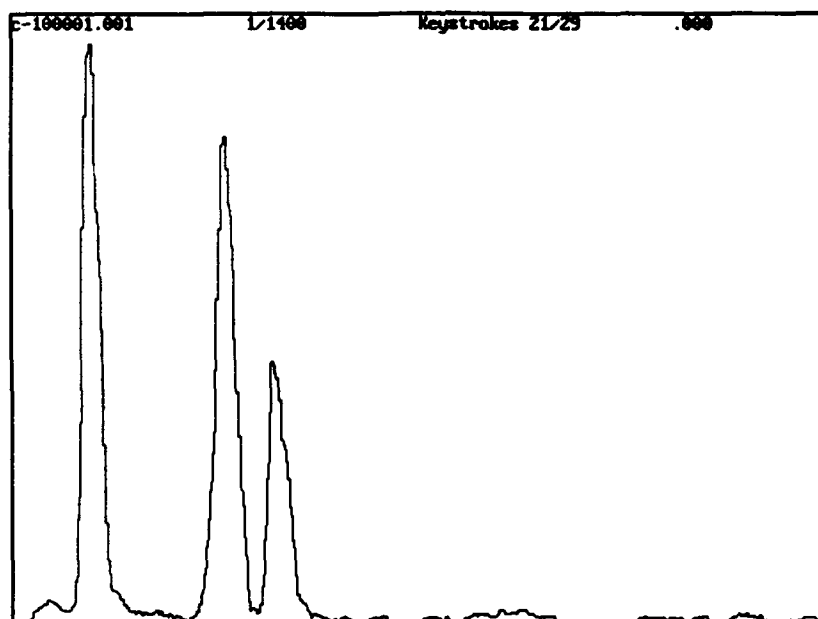


Figure 42. Form Envelope

Striking 0 will filter the envelope, Figure 43.

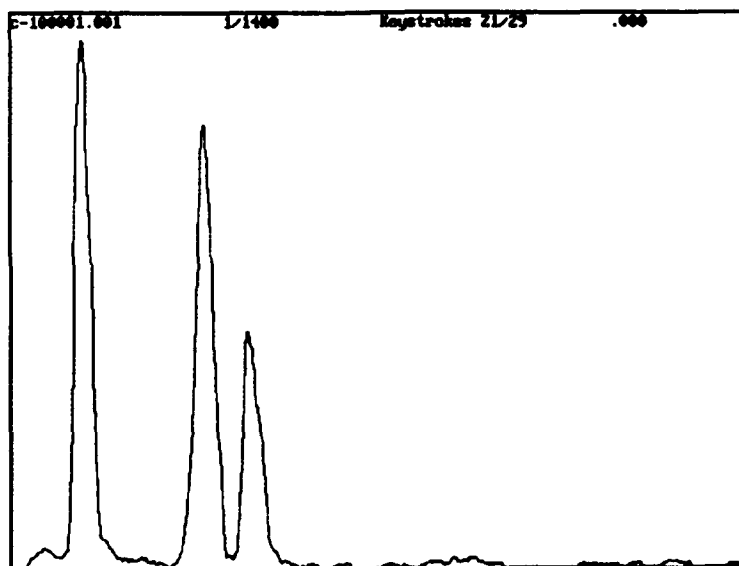


Figure 43. Filtered Data

The cursor may be advanced past the surface reflection

data simulating a delay circuit, by either combinations of arrow keys or by the jump key *J*. The resulting cursor position is shown in Figure 44.

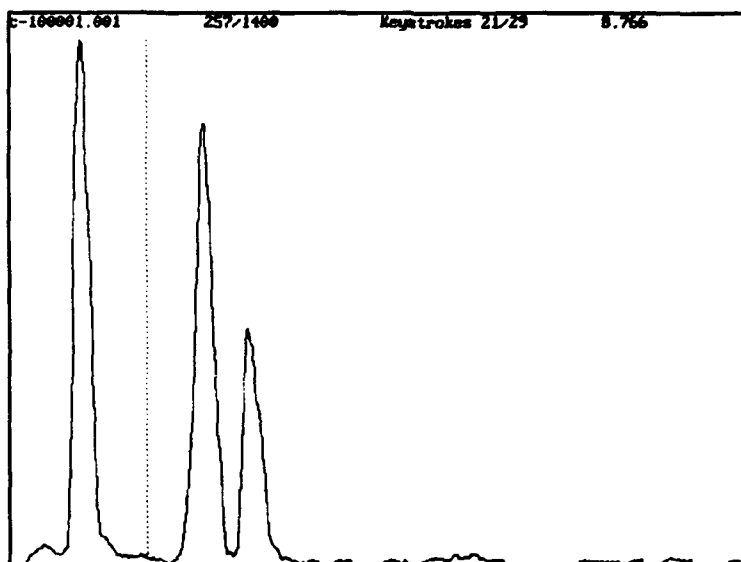


Figure 44. Cursor Advanced

The rectify and set to unity maximum value key *R* causes the peak value to be one, Figure 45.

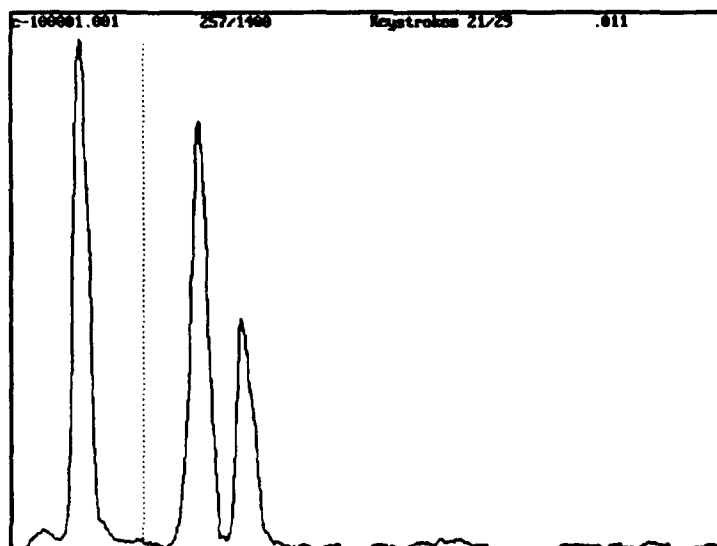


Figure 45. Maximum Value 1.0

The threshold command *T* is used to jump to the first point higher than the set threshold (in this case .1), Figure 46.

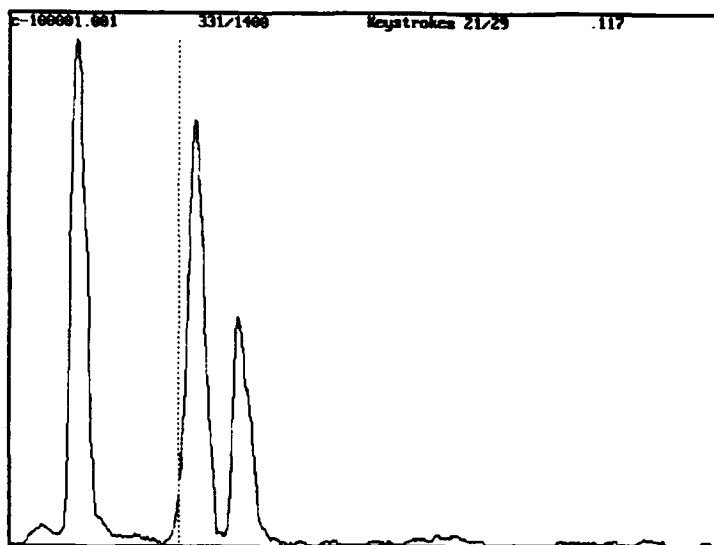


Figure 46. Jump to Threshold

The *O* keystroke causes the waveform to be clipped to a predetermined number of points beyond the cursor (e.g., 256 points--input from the file "CLIPPNTS"), Figure 47.

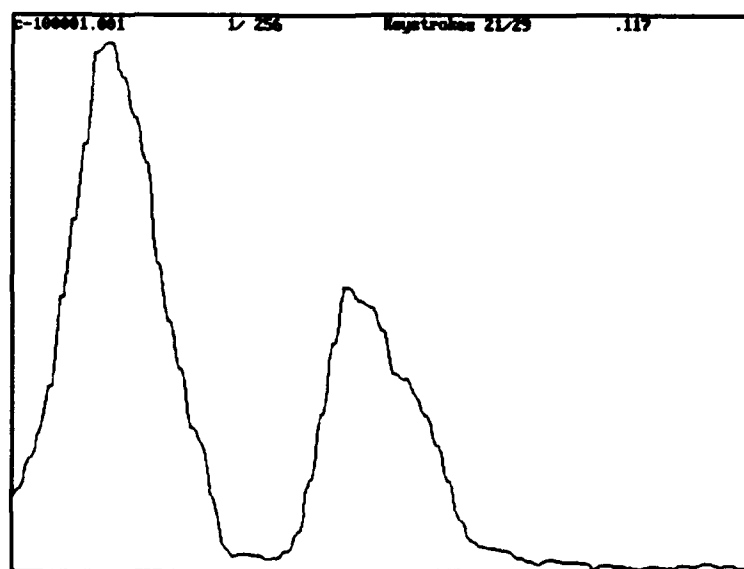


Figure 47. Clip to 256 Points

The *D* key causes decimation by a factor of 2 to 128 points, Figure 48.

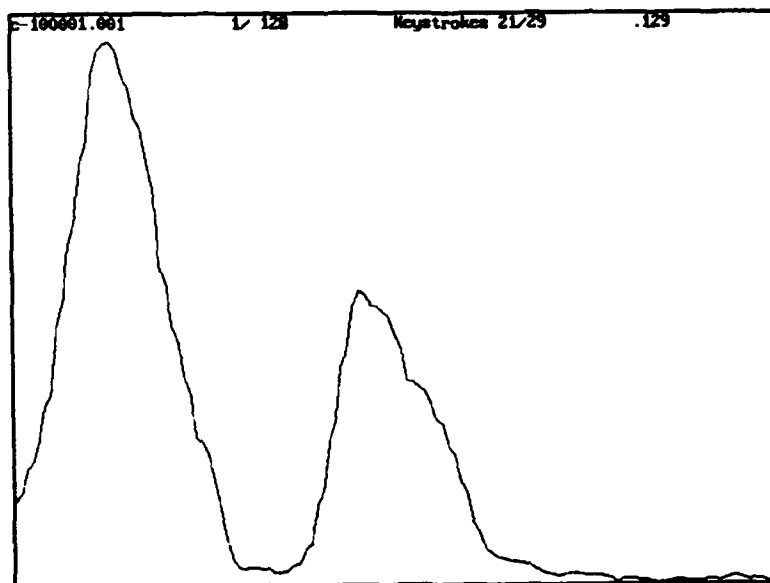


Figure 48. Decimate by 2

The *F* key causes a magnitude FFT to be performed, Figure 49.

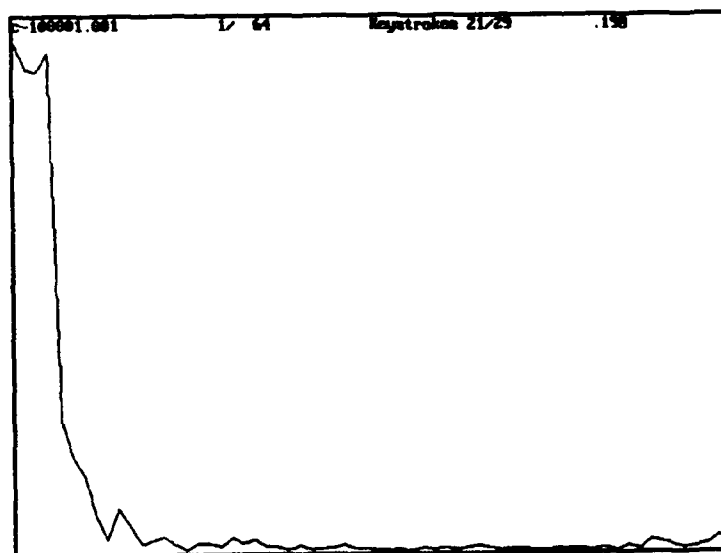


Figure 49. FFT of Data

One way to increase the effect of the higher frequency

components on the neural model would be to scale the FFT to unity and then take the square root of the values. R is used for the scaling, Figure 50.

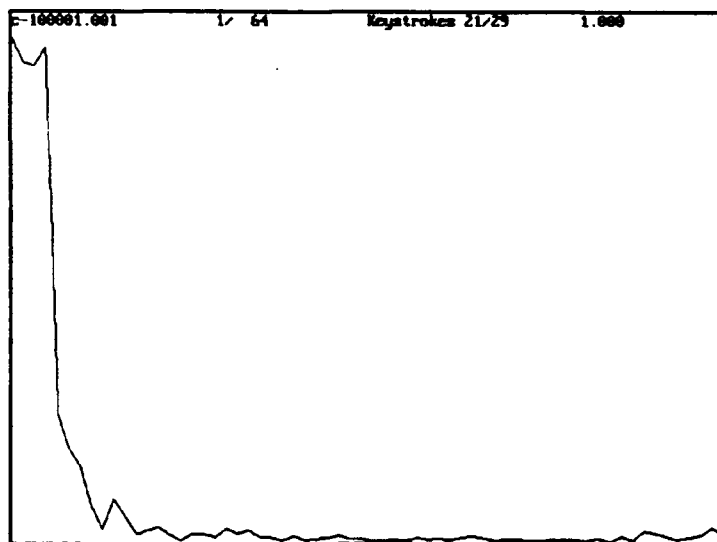


Figure 50. Scale to Unity

V is used to take the square root, Figure 51.

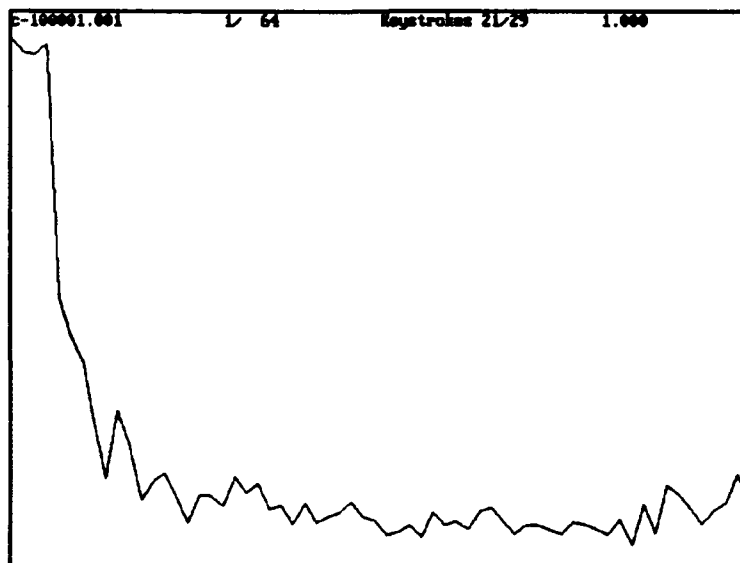


Figure 51. Square Root of Data

The data could be saved by entering A. The next file

in the list could be brought up by entering *N*. If this entire series of keystrokes was to be applied to every file in the list, the keystrokes could be saved and replayed recursively. To do this, after the first data file was displayed, the following keystrokes would be entered:

i5.0jrtodfrvank

At this point the keystrokes are saved in a file called "KEYSTROK" and the second data file in the list is displayed. Entering *K* will cause all remaining files in the list to be processed and saved automatically. When the final file is processed, "PRENEURA" will terminate.

Before running "PRENEURA" the contents of the following files should be defined using a text editor, if keys associated with the files are to be used in processing: (1) OUTDIGIT, (2) JUMPDIST, (3) CLIPPNTS, (4) THRESHOL, (5) FILTER, and (6) DELTAS. The use of these files and sample contents are given at the beginning of this document.

The value of the current point under the cursor, the current data file name, the current cursor position, and the current number of keystrokes saved is displayed at the top of the screen. Note that the arrow keys on a 101 key keyboard will cause two characters to be generated and the keystroke count will increment by 2 rather than one (the keypad arrow keys in numlock give 1 character).

A.2 PRENEURA Example Run 2

Another example will illustrate the power of subroutine style keystrokes and the queue for building a time-frequency set of inputs. Each record will have four FFT's performed at different, overlapping increments of time along the signal. The first thing that must be entered is the file list (Figure 52).

```
ENTER: g for get a file
       l for get a list of files
Enter g or l.
l

File containing list of file names:
oak20004.lis
```

Figure 52. Entering File List

The program then retrieves and displays the first data file's signal (Figure 53).

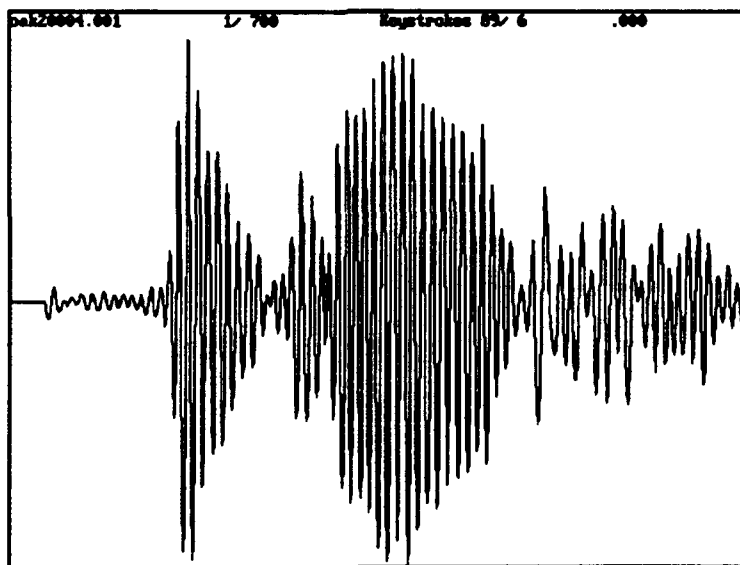


Figure 53. Initial Data Plot

When *H* is input, two screens listing the possible keystrokes are displayed. The first is shown in Figure 54.

```
A autosave in file named INFILE.A??
B mark the beginning limit
C clip to the marked limits
D decimate by 2
E mark the ending limit
F take the Fourier amplitude transform
G get a new file
H or ? for help on key functions
I initiate keystroke save or terminate save
J relative to current point
K do predetermined Keystrokes or end keystroke save
L get a list of files for sequential processing
M Previous sequential file
N Next sequential file
O clip to points from CLIPPNTS from current point
P clip to power of two points from current point
Q quit
R rectify and scale to between 0. and 1.
S save current data in a file
T jump to the next point above the threshold
U unzoom
V square root of data
W square of data
X full-wave rectify around mean value of data
Y scale between 0 & 1 with original mean value .5
Z zoom in on marked limits
```

Figure 54. First Help Screen

Depressing any key will cause the next screen to be displayed (Figure 55).

```
> move forward 1 point
< move backward 1 point
) move forward 16 points
( move backward 16 points
) move forward 256 points
( move backward 256 points
@ filter data
1 interpolate data
3 temporary save
5 half-wave rectify around mean value of data
7 temporary restore
9 clear queue
. append to queue
/ get queue
. form data envelope
right arrow move right along data
left arrow move left along data
up arrow double movement rate
down arrow halve movement rate
space bar redraw screen display
- invert data
. find peak of data
: Start/stop recording of subkeys
= playback sub keys or end sub keystroke save
```

Figure 55. Second Help Screen

Inputting *19))))))* will cause the screen of Figure 56 to

be displayed. I will initiate the saving of major keystrokes, 9 will clear the queue, and each) will advance the pointer 16 points, for 128 points total, to point 129.

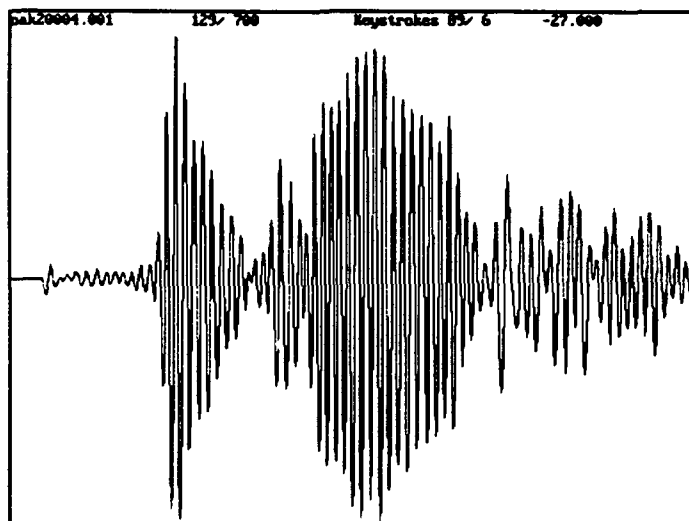


Figure 56. Advance 128 Points

Figure 57 is the result of depressing Y; which causes the data to be scaled between 0 and 1 and the ; causes initialization of subkeystroke saving.

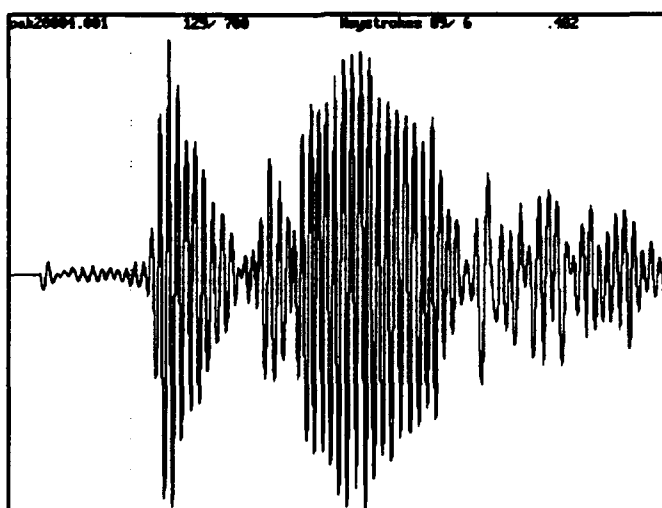


Figure 57. Scale Between 0 and 1

The next set of keystrokes 300 cause the display to change to that of Figure 58. 3 causes the current contents of the buffer to be saved in a temporary buffer, 0 causes the data to be filtered according to the filter specified in the file "FILTER", and 0 causes the data to be clipped timewise from the current point to the number of points specified in the file "CLIPPNTS."

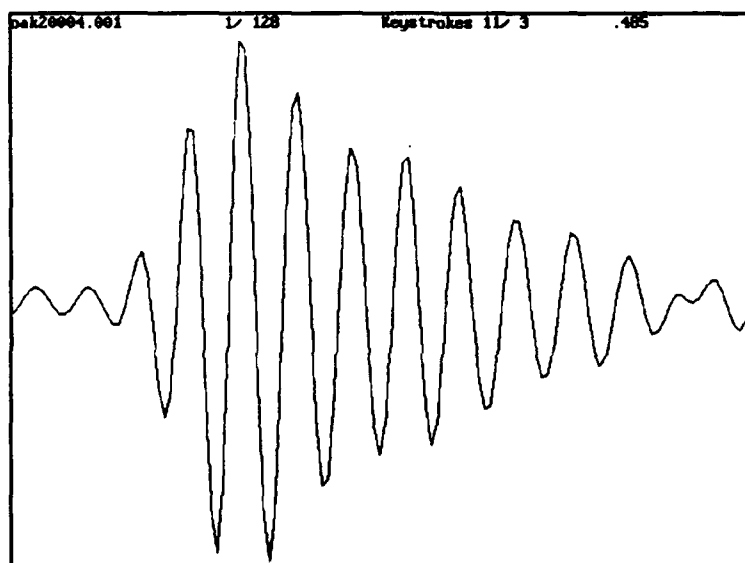


Figure 58. Filtered and Clipped

Then *F* is entered to take the Fourier transform of the data on the screen (Figure 59).

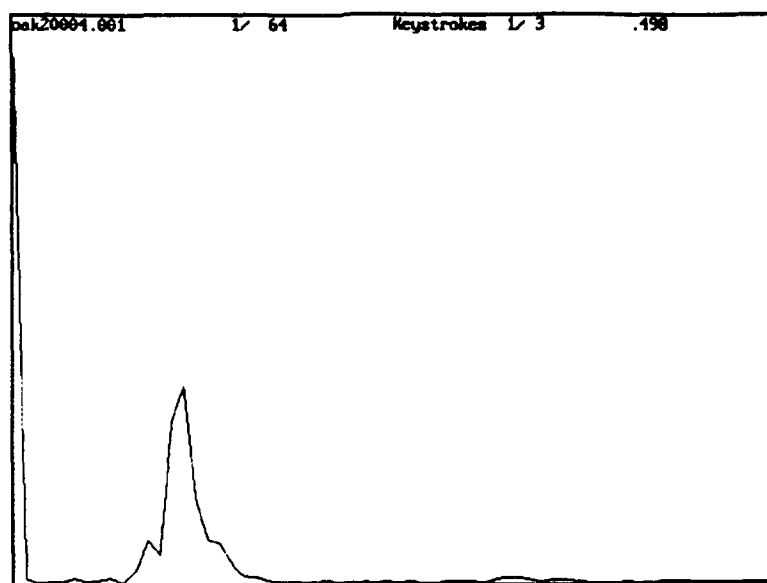


Figure 59. Fourier Transform

Next eight right arrow depressions are followed by depressing B which causes the pointer to advance eight data points and mark the point as the beginning (Figure 60).

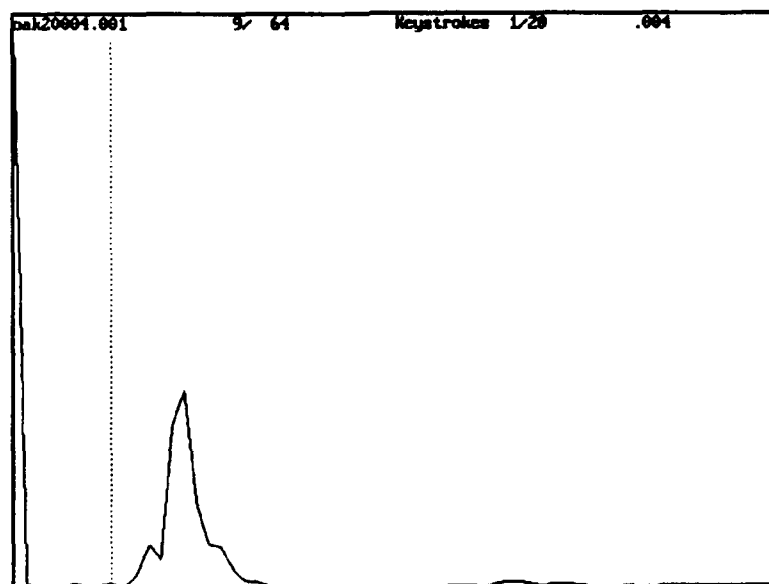


Figure 60. Mark Beginning

150

The next keys entered are)e which cause the pointer to advance 16 points and the point to be marked as the end (Figure 61).

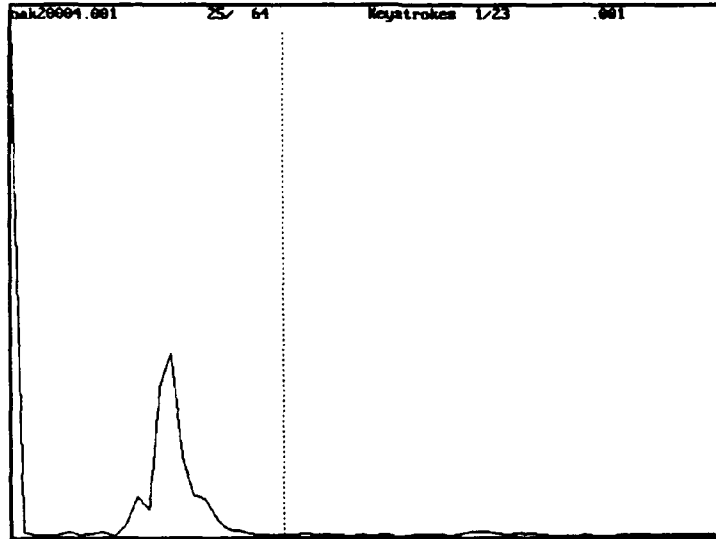


Figure 61. Mark End

Then CP are entered to clip the data to 17 points and then clip it to the largest power of two less than 17 (i.e., 16 points). Figure 62 shows the results.

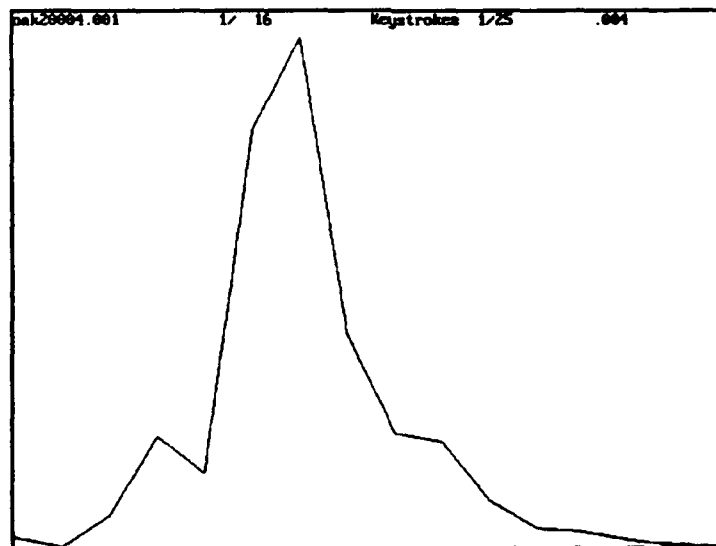


Figure 62. Clip to Power of Two

Next comma is entered to append the points to the queue (Figure 63).

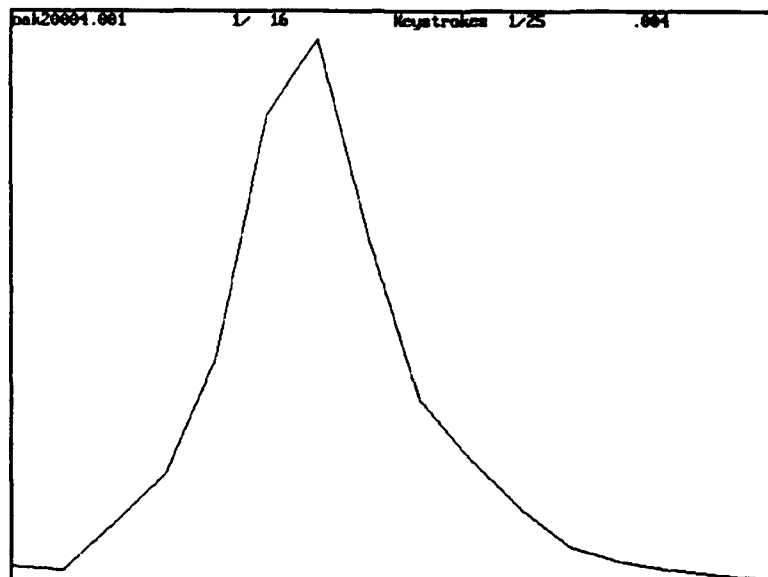


Figure 63. Append to Queue

Then 7); causes the data stored in the temporary buffer to be recalled and the pointer to be advances 32 points (Figure 64). The semicolon causes the saving of subkeys to end.

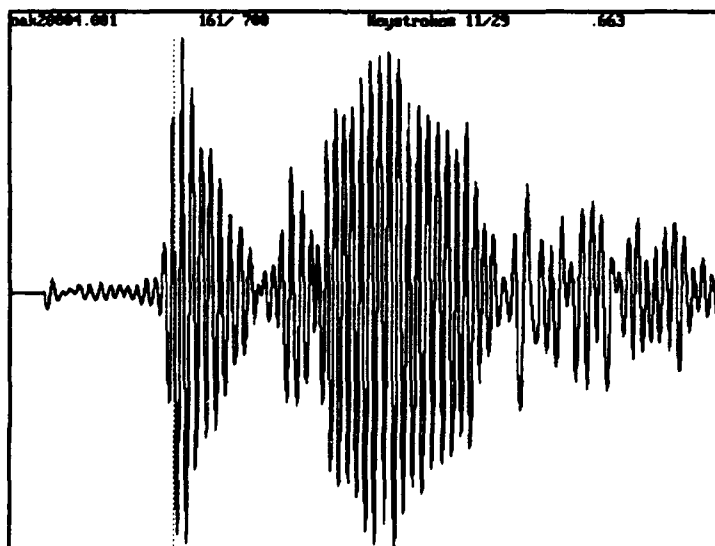


Figure 64. Recall Temporary Buffer

Then inputting three equals causes the subkeys to be played back three times and / causes the contents of the queue to replace the data (Figure 65).

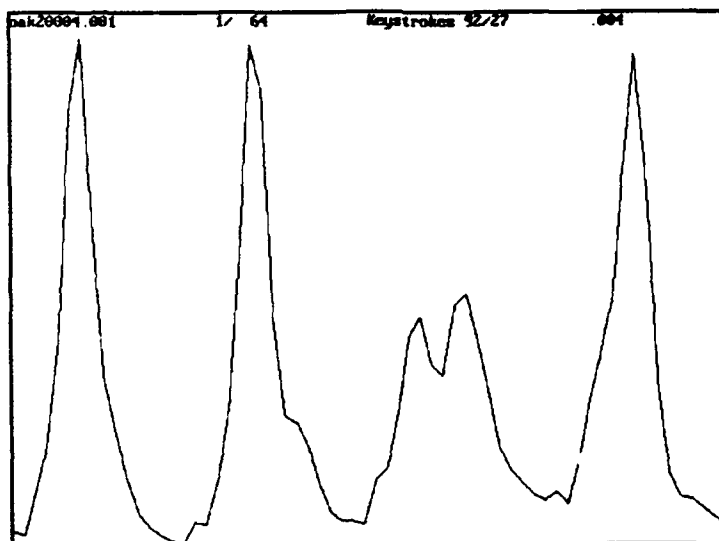


Figure 65. Queue Replaces Data

Finally ANK causes the data to be saved in a file, the next data file to be read in and displayed, and the recording of major keystrokes to end (Figure 66).

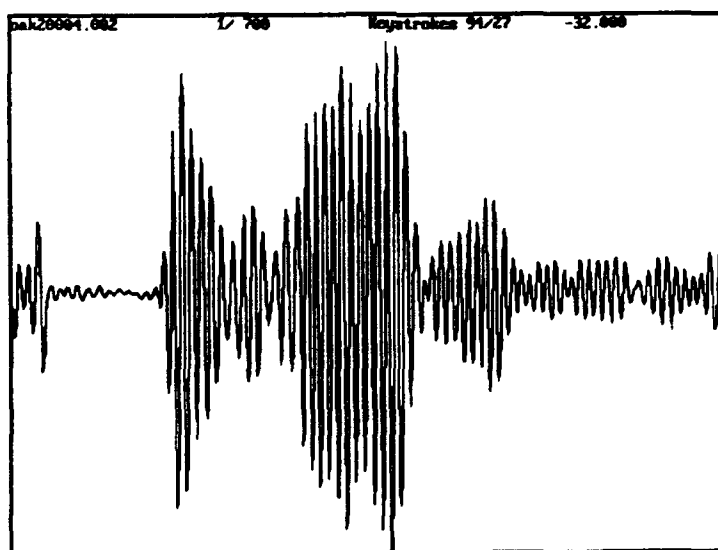


Figure 66. Save Data

Then pressing *K* will cause all remaining data files in the list to be processed by the played back keystrokes. The strokes will be recursively played back; so that all remaining files are processed and saved. Therefore, the keystrokes entered were

```
i9}}}}}}Y;66666666b)ecp,7));===/ank
```

where 6 may be used for right arrow. This results in the saving of a major keystroke file containing

```
i9)))))Y===/ank
```

and a subkeystroke file containing

```
30of66666666)ecp,7))
```

where the subkeystroke sequence was replaced by an equals sign in the major keystroke sequence, insuring that processing of subsequent file lists will yield the same results.

A.3 Listing of PRENEURA

C KEY FUNCTIONS:

C A autosave in file named 'INFILE'.A??

C If a file named 'OUTDIGIT' exists, the letter A
C above in the output file name is replaced by the
C first character in the file 'OUTDIGIT'. This
C allows creating different sets of inputs for
C the BACKPROP program without renaming output
C files. The file "OUTDIGIT" is built using a
C text editor, an example contents would be

C Z

C to set the filename to 'INFILE'.Z??

C B mark the beginning limit

C "B", "E", and "C" work together. You first mark

C the limits with "B" and "E" and then clip to them

C with "C"

C clip to the marked limits

C D decimate by 2

C E mark the ending limit

C F take the Fourier amplitude transform

C If you want the Power Spectral Density follow "F"

C with "W"

C G get a new file

C Generally using file lists is more convenient, but

C G allows doing one file independently

C H or ? for help on key functions

C Displays part of the above help

C I initiate keystroke save or terminate save

C The first time you strike "i" the program goes into

C record keystroke mode and records all keystrokes

C until either "i" is struck again or "k" is struck.

C If the sequence is terminated with "k" a recursive

C pattern is created (see k). Embedded "H" and

C terminating "I" are not recorded. The keystrokes

C are saved in a file called "KEYSTROK."

C J jumps relative to the current point by predetermined

C amount. Jump distance in points is input from a file
C called "JUMPDIST"; the distance may be either pos.
C or negative and clips to the screen limits. It is
C often used following the threshold command to backup
C a set number of points. An example file contents
C might be

C -20

C K do predetermined Keystrokes or terminate keystroke
C save with k. This is a powerful means of applying
C the same key sequence to a number of files
C automatically.

C If a list of files is input by the L command, a
C keystroke sequence may be terminated by the strokes
C "A", "N", "K". When executed with the first file on
C screen, the K command will cause files in the list
C to be sequentially processed by the keystroke
C string and saved automatically by the "A" command.
C "N" advances the file to the next one and "K"
C causes the recursive processing. When the last
C file has been processed the sequence terminates.
C Everything is visible on the screen, so you can
C sit back and visually observe
C the effects on your data as the screens fly by.

C L get a list of files for sequential processing

C This is used to read a file containing a list of
C files, generally a test set. Then the "M" and "N"

C keys can be used to advance and backup through the
C list.

C M Previous sequential file

C N Next sequential file

C O Clip to number of points specified in file CLIPPNTS

C Note this works from the current cursor position

C The file "CLIPPNTS" is built using a test editor,

C an example contents would be

C 50

C to set the clip value to 50 points i.e. present to

C present+49

C P clip to power of two points from current point

C Note this works from the current cursor position

C and is often used to follow the "T" command

C Q quit

C R rectify and scale to between 0. and 1.

C Values below 0 are set to 0 (half wave rectify)

C and then the data is scaled to range from 0. to 1.

C S save current data in a file

C Used in conjunction with the "G" command. Normally

C "A" is used to save files loaded with the "L"

C command.

C T jump to the next point above the threshold

C The threshold is set in a data statement

C (currently .05) and is normally used after the "R"

C command to jump to the first scaled point => the

C threshold. If a file named
C "THRESHOL" exists, the ASCII floating point value
C in this file is used for the threshold thus the
C threshold can be changed without recompiling. The
C file "THRESHOL" is built using a test editor, an
C example contents would be
C 0.5
C to set the threshold to .5
C U unzoom
C V square root of data
C W square of data
C X rectify to mean full-wave
C Y scale between 0 & 1 with original data's mean = .5
C Z zoom in on marked limits
C . Form envelope of signal via peak detection
C 0 filter data with coefficients from file "FILTER"
C The filter may be either recursive or nonrecursive
C The first character of the file "R" or "N"
C determines type. The next lines of the file are
C the coefficients starting at the present point.
C Nonrecursive are assumed balanced and only the
C center and one side are entered. For example, the
C contents of "FILTER" for a .25 .5 .25 nonrecursive
C filter would be
C N
C 0.5

158

```
C          0.25
C      1 interpolate data spaced at delt to deltnew
C          If a file named "DELTAS" exists, the
C      ASCII floating point values in this file is used for
C      delta and deltnew thus deltas can be changed without
C      recompiling. The file "DELTAS" is built using a test
C      editor, an example contents would be
C          0.5
C          0.2
C      3 save buffer
C      5 half wave rectify about the mean
C      7 restore saved buffer
C      9 Clear queue
C      , append to queue
C      / get queue
C      - invert data about zero
C      ; start/stop recording of subkeys
C      The first time you strike ";" the program goes into
C      record subkeystroke mode and records all keystrokes
C      until either ";" is struck again or "=" is struck.
C      Embedded "H" and terminating ";" are not recorded.
C      The keystrokes are saved in a file called SUBSTROK
C      = playback sub keys
C      This is a powerful means of applying the same key
C      sequence to a number of files automatically. This
C      is used like a subroutine to playback a set of
```

C keystrokes that will be applied repeatedly to the
C same waveform. When used with the queue feature then
C time/frequency neural net inputs may be created by
C doing FFT's along the time axis. Everything is
C visible on the screen, so you can sit back and
C visually observe
C the effects on your data as the screens fly by.

C

C -> move right along data

C <- move left along data

C

C The following movement rate commands increment the jump
C size that is used by the left and right arrow commands
C above. On initialization the jump rate is one. Every
C time one of the above keys is pressed the cursor advances
C (back up) one data point. Pressing up arrow increases
C the rate in powers of 2, down arrow decreases in
C powers of 2. Thus, if up arrow is pressed four times
C the cursor jumps 16 data points each time one of the
C above (2) keys is pressed.

C

C ^

C | double movement rate

C

C | halve movement rate

C v

160

C

C space bar redraw screen

C

CC PRENEURA.FOR - PreNeural Network program.

 INCLUDE 'FGRAPH.FI'

 INCLUDE 'FGRAPH.FD'

 LOGICAL go_graphics

 EXTERNAL go_graphics

 LOGICAL mode_of_screen

 IF(go_graphics(mode_of_screen)) THEN

 CALL pre_neural(mode_of_screen)

 ELSE

 WRITE (*,*) ' This program requires a CGA, EGA, '

+ , ' or',

+ ' VGA graphics card.'

 END IF

 END

C Additional functions defined below

CC Function to enter graphics mode

 LOGICAL FUNCTION go_graphics(mode_of_screen)

 INCLUDE 'FGRAPH.FD'

 INTEGER*2 dummy

 LOGICAL mode_of_screen

 RECORD /videoconfig/ screen

 COMMON screen

C

C Set to maximum number of available colors.

C

```

CALL getvideoconfig( screen )
mode_of_screen=.FALSE.
SELECT CASE( screen.adapter )
  CASE( $CGA, $OCGA )
    dummy = setvideomode( $MRES4COLOR )
    mode_of_screen=.TRUE.
  CASE( $EGA, $OEGA )
    dummy = setvideomode( $ERESCOLOR )
  CASE( $VGA, $OVGA )
    dummy = setvideomode( $VRES16COLOR )
  CASE DEFAULT
    dummy = 0
END SELECT
CALL getvideoconfig( screen )
go_graphics = .TRUE.
IF( dummy .EQ. 0 ) go_graphics = .FALSE.
END

```

CC Pre_neural allows editing and transforming data files

```

SUBROUTINE pre_neural(mode_of_screen)
  INCLUDE 'FGRAPH.FD'
  LOGICAL mode_of_screen
  INTEGER*2          dummy

```

INTEGER*2 xwidth, yheight, cols, rows,x,y
COMMON screen
RECORD /videoconfig/ screen
RECORD /rccoord/ curpos
CHARACTER*9 text
CHARACTER*16 text2
CHARACTER*10 keymsg
CHARACTER*80 a
CHARACTER*36 h1
CHARACTER*27 h2
CHARACTER*28 h3
CHARACTER*16 h4
CHARACTER*24 h5
CHARACTER*39 h6
CHARACTER*17 h7
CHARACTER*33 h8
CHARACTER*44 h9
CHARACTER*28 h10
CHARACTER*52 h11
CHARACTER*48 h12
CHARACTER*27 h13
CHARACTER*23 h14
CHARACTER*50 h14a
CHARACTER*49 h15
CHARACTER*7 h16
CHARACTER*41 h17

CHARACTER*23 h17a
CHARACTER*24 h17b
CHARACTER*25 h17c
CHARACTER*26 h17d
CHARACTER*26 h17e
CHARACTER*27 h17f
CHARACTER*30 h21
CHARACTER*45 h22
CHARACTER*9 h23
CHARACTER*22 h24
CHARACTER*17 h25
CHARACTER*47 h25a
CHARACTER*50 h25b
CHARACTER*27 h26
CHARACTER*17 h26a
CHARACTER*20 h26b
CHARACTER*20 h26c
CHARACTER*47 h26d
CHARACTER*21 h26e
CHARACTER*15 h26f
CHARACTER*19 h26g
CHARACTER*13 h26h
CHARACTER*22 h26i
CHARACTER*35 h27
CHARACTER*34 h28
CHARACTER*32 h29

```

CHARACTER*33 h30
CHARACTER*33 h31
CHARACTER*16 h32
CHARACTER*20 h33
CHARACTER*34 h34
CHARACTER*46 h35
CHARACTER*12 filenames(500),INFILE,OUTFILE,FILE
INTEGER*1 keydep,sqq,lqq,space,four,five,six,plus
INTEGER*1 keydep2,eight,two,lrr,srr,lpp,spp,minus
INTEGER*1 lcaret,rcaret,lparen,rparen,lbrace,rbrace
INTEGER*1 lss,sss,la,da,ra,ua,lvv,svv,lww,sww
INTEGER*1 lcc,scc,lee,see,lbb,sbb,ldd,sdd,one
INTEGER*1 laa,saa,lkk,skk,ifile(12),type_filter
INTEGER*1 lii,sii,ljj,sjj,outchar,outch,zero,dot
INTEGER*1 lhh,shh,question_mark,loo,soo,lxx,sxx
INTEGER*1 ltt,stit,lff,sff,lgg,sgg,lastkey,null
INTEGER*1 luu,suu,lzz,szz,lmm,smm,lnn,snn,lll,sl1
INTEGER*1 three,seven,nine,comma,slash,
1 semicolon,equals
INTEGER*1 lyy,syy
INTEGER powr2(13)
LOGICAL keysave,subkeysave
real ybuf(4000),ybuf1(4000),coef_filter(100),xbuf(201)
real ybuf2(4000),ybuf3(4000)
EQUIVALENCE (FILE,IFILE(1))
integer*1 keystrokes(499),substrokes(99)

```

```

COMMON /yb/ybuf1,ybuf2,ybuf3

DATA numstrokes/0/,outchar/'A'/,numsubstrokes/0/

DATA keymsg/'Keystrokes'/,threshold/0.05/

DATA powr2/2,4,8,16,32,64,128,256,512,1024,
1 2048,4096,8192/

DATA lqq/'Q'//,sqq/'q'//,space/' '/,dot/'.'/

DATA lll/'L'//,sll/'l'//,null/0//,zero/'0'/

DATA lrr/'R'//,srr/'r'//,lpp/'P'//,spp/'p'/

DATA laa/'A'//,saa/'a'//,lkk/'K'//,skk/'k'/

DATA lvv/'V'//,svv/'v'//,lww/'W'//,sww/'w'/

DATA lmm/'M'//,smm/'m'//,lnn/'N'//,snn/'n'/

DATA lbb/'B'//,sbb/'b'//,lee/'E'//,see/'e'/

DATA ldd/'D'//,sdd/'d'//,lcc/'C'//,scc/'c'/

DATA luu/'U'//,suu/'u'//,lzz/'Z'//,szz/'z'/

DATA lll/'I'//,sll/'i'//,ljj/'J'//,sjj/'j'/

DATA lss/'S'//,sss/'s'//,left_text1/41//,left_text2/61/

DATA left_text0/21//,lyy/'Y'//,syy/'y'/

DATA ltt/'T'//,stt/'t'//,loo/'O'//,soo/'o'/

DATA lff/'F'//,sff/'f'//,lgg/'G'//,sgg/'g'/

DATA lhh/'H'//,shh/'h'//,question_mark/'?'/

DATA lxx/'X'//,sxx/'x'//,coef_filter/.5,.25,98*0./

DATA type_filter/'N'//,num_coef/2//,one/'1'/

DATA la/75//,da/80//,ra/77//,ua/72//,num_to_clip/50/

DATA four/'4'//,five/'5'//,six/'6'//,eight/'8'//,
1 equals/'='/

DATA nine/'9'//,comma/','//,slash/'/'//,semicolon/';'/

```



```

DATA minus/'-'/,plus/'+'/,three/'3'/,seven/'7'/
DATA lcaret/'<'/,rcaret/'>'/,lparen/'('/
DATA rparen/')'/,lbrace/'{'/,rbrace/'}'/
DATA two/'2'/,filenames/500*'      '/
DATA h1/' A autosave in file named INFILE.A??'/
DATA h2/' B mark the beginning limit'/
DATA h3/' C clip to the marked limits'/
DATA h4/' D decimate by 2'/
DATA h5/' E mark the ending limit'/
DATA h6/' F take the Fourier amplitude transform'/
DATA h7/' G get a new file'/
DATA h8/' H or ? for help on key functions'/
DATA h9/' I initiate keystroke save or terminate',
1  ' save'/
DATA h10/' J relative to current point'/
DATA h11/' K do predetermined Keystrokes or end',
1  ' keystroke save'/
DATA h12/' L get a list of files for sequential',
1  ' processing'/
DATA h13/' M Previous sequential file'/
DATA h14/' N Next sequential file'/
DATA h14a/' O clip to points from CLIPPNTS from',
1  ' current point'/
DATA h15/' P clip to power of two points from current'
1  ' point'/
DATA h16/' Q quit'/

```

```
DATA h17/' R rectify and scale to between 0. and 1.'/
DATA h17a/' > move forward 1 point'/
DATA h17b/' < move backward 1 point'/
DATA h17c/' ) move forward 16 points'/
DATA h17d/' ( move backward 16 points'/
DATA h17e/' } move forward 256 points'/
DATA h17f/' { move backward 256 points'/
DATA h21/' S save current data in a file'/
DATA h22/' T jump to the next point above the',
1   ' threshold'/
DATA h23/' U unzoom'/
DATA h24/' V square root of data'/
DATA h25/' W square of data'/
DATA h25a/' X full-wave rectify around mean value of',
1   ' data'/
DATA h25b/' Y scale between 0 & 1 with original mean',
1   ' value .5'/
DATA h26/' Z zoom in on marked limits'/
DATA h26a/' 0 filter data  '/
DATA h26b/' 1 interpolate data'/
DATA h26c/' 3 temporary save  '/
DATA h26d/' 5 half-wave rectify around mean value of',
1   ' data'/
DATA h26e/' 7 temporary restore'/
DATA h26f/' 9 clear queue'/
DATA h26g/' , append to queue'/
```

```

DATA h26h/' / get queue'/
DATA h26i/' . form data envelope'/
DATA h27/' right arrow  move right along data'/
DATA h28/' left arrow   move left along data'/
DATA h29/' up arrow     double movement rate'/
DATA h30/' down arrow   halve movement rate'/
DATA h31/' space bar   redraw screen display'/
DATA h32/' -           invert data'/
DATA h33/' + Find peak of data'/
DATA h34/' ; Start/stop recording of subkeys'/
DATA h35/' = playback sub keys or end sub keystroke',
1    ' save'/

DATA delt/20./,deltnew/20./,jump_relative/0/

iupdate=0
ifileptr=0
nqueue=0
numstrokes=1

OPEN(3,file='KEYSTROK',status='OLD',
1   form='BINARY',err=114)

115  read(3,err=113,end=113) keystrokes(numstrokes)
      numstrokes=numstrokes+1
      go to 115

113  CLOSE(3)

114  numstrokes=numstrokes-1
      istrokes=numstrokes+1
      numsubstrokes=1

```

```
OPEN(3,file='SUBSTROK',status='OLD',
1   form='BINARY',err=174)
175  read(3,err=173,end=173) substrokes(numsubstrokes)
    numsubstrokes=numsubstrokes+1
    go to 175
173  CLOSE(3)
174  numsubstrokes=numsubstrokes-1
    isubstrokes=numsubstrokes+1
C    Get new autosave first extension character if
C    available
OPEN(3,file='OUTDIGIT',status='OLD',err=764)
    read(3,765,err=763,end=763) outch
765  format(A1)
    outchar=outch
763  CLOSE(3)
C    Get number of points for clip if available
764  OPEN(3,file='JUMPDIST',status='OLD',err=364)
    read(3,965,err=363,end=363) nclip
    jump_relative=nclip
363  CLOSE(3)
C    Get new interpolation deltas
364  OPEN(3,file='DELTAS',status='OLD',err=264)
    read(3,865,err=263,end=263) thresh
    delt=thresh
    read(3,865,err=263,end=263) thresh
    deltnew=thresh
```

```
170
263  CLOSE(3)
C    Get new trigger threshold if available
264  OPEN(3,file='THRESHOL',status='OLD',err=864)
      read(3,865,err=863,end=863) thresh
865      format(F20.0)
      threshold=thresh
863  CLOSE(3)
C    Get number of points for clip if available
864  OPEN(3,file='CLIPPNTS',status='OLD',err=964)
      read(3,965,err=963,end=963) nclip
965      format(I6)
      num_to_clip=nclip
963  CLOSE(3)
C    Get filter coefficients
964  OPEN(3,file='FILTER',status='OLD',err=1067)
      read(3,765,err=1063,end=1063) type_filter
      num_coef=1
1066  read(3,865,err=1063,end=1063) coef_filter(num_coef)
      num_coef=num_coef+1
      go to 1066
1063  CLOSE(3)
1064  num_coef=num_coef-1
1067  maxfiles=0
      keysave=.FALSE.
      subkeysave=.FALSE.
      isn=1
```

```

CALL clearscreen( $GCLEARSCREEN )
xwidth  = screen.numxpixels
yheight = screen.numypixels
cols    = screen.numtextcols
rows    = screen.numtextrows
ipat=1
ijump=0
C      If screen is CGA move letter start position to left
C      since we can only fit 40 characters across the screen
      IF(mode_of_screen) then
          left_text0=9
          left_text1=19
          left_text2=30
      ENDIF
C      Input first file name or name of file list file
196 write(6,191)
191 format(' ENTER:  g for get a file'/
1      '          l for get a list of files'/
2      ' Enter g or l.')
```

```

      read(5,199,err=196,end=999) keydep
199 format(a1)
C      The following shows what happens when you patch a
C      program.  It skips down to execute the code which
C      would be executed if "L" or "G" were entered while
C      processing, purists may
C      want to make L and G subroutines to avoid this go to

```

```

      if (keydep.eq.sll.or.keydep.eq.lll.or.keydep.eq.sgg
1 .or.keydep.eq.lgg) go to 344

```

C The following go to is necessary to implement the
 C While command not available in FORTRAN; in PASCAL
 C statement 196 would be a "WHILE" and the following
 C would be the "END" for the while block. This
 C construct is used at various points in the program to
 C implement the "WHILE" statement

```

      Go to 196

```

C Beginning of the While input key not "Q" construct
 C This is the point of return for commands that require
 C that the screen be totally redrawn after execution

```

1 CALL setviewport( 0, 0, xwidth - 1, yheight - 1 )
  CALL settextwindow( 1, 1, rows , cols )
  yminn=ymin-(ymax-ymin)/20.
  dummy = setwindow( .FALSE., xmin, yminn, xmax, ymax )
  CALL clearscreen( $GCLEARSCREEN )
  call displayone(ybuf,ymin,ymax,npoint1,npoint2)

```

C This is the point of return for commands that require
 C only the cursor and current screen display values be
 C updated

```

4 call cursor(ybuf,ymin,ymax,ipoint,
1 npoint1,npoint2)

```

C To write to the screen in graphics mode requires
 C setting the
 C start of text position [with (1,1) being the upper

```
C    left hand corner of the screen] and then calling
C    outtext to write to the screen.  Since outtext can
C    only handle ASCII strings, formatted
C    writes must be done to a memory buffer "a" and then
C    ascii reads done from the buffer to accomplish the
C    conversion to ASCII
C    The following displays the info in the upper right
C    hand corner of the screen

x=1
y=1
call settextposition(y,x,curpos)
call outtext(infile)
x=left_text0
y=1
call settextposition(y,x,curpos)
dummy=setcolor(2)
write(a,19) ipoint,maxpoints
19  format(i4,'/',i4)
    read (a,20) text
    call outtext(text)
x=left_text1
y=1
call settextposition(y,x,curpos)
write(a,771) keymsg,numstrokes,slash,numsubstrokes
771 format(a10,i3,a1,i2)
    read (a,772) text2
```



```

174
772  format(a16)
      call outtext(text2)
      x=left_text2
      y=1
      call settextposition(y,x,curpos)
      write(a,21) ybuf(ipoint)
21   format(f9.3)
      read (a,20) text
      call outtext(text)
20   format(a9)
141  lastkey=keydep
14   call getkey(keydep,ierr)
      if (ierr.eq.0.and.(keysave.or.istrokes.gt.numstrokes)
1    .and.(subkeysave.or.isubstrokes.gt.numsubstrokes))
2    go to 14
      if((.not.subkeysave).and.(isubstrokes.le.
1    numsubstrokes)) then
          keydep=substrokes(isubstrokes)
          isubstrokes=isubstrokes+1
      else
          if((.not.keysave).and.(istrokes.le.numstrokes))
1      then
          keydep=keystrokes(istrokes)
          istrokes=istrokes+1
      endif
  endif
endif

```

```

if(subkeysave) then
    if(.not.(keydep.eq.lhh.or.keydep.eq.shh
1    .or.keydep.eq.question_mark)) then
        numsubstrokes=numsubstrokes+1
        substrokes(numsubstrokes)=keydep
    endif
else
    if(keysave.and.(isubstrokes.gt.numsubstrokes)) then
        if(.not.(keydep.eq.lhh.or.keydep.eq.shh
1    .or.keydep.eq.question_mark)) then
            numstrokes=numstrokes+1
            keystrokes(numstrokes)=keydep
        endif
    endif
endif

C
C    space bar  redraw screen
C
344  if(keydep.eq.space) then
        go to 1

C
C    left arrow  move left along data
C

    else if(keydep.eq.four.or.(lastkey.eq.null
1        .and.keydep.eq.la)) then
        ipoint=ipoint-2**ijump

```

176

```
    if(ipoint.lt.npoint1) then
```

```
        ipoint=npoint1
```

```
    endif
```

C

C down arrow halve movement rate

C

```
    else if(keydep.eq.two.or.(lastkey.eq.null.and.
```

```
1 keydep.eq.da)) then
```

```
        ijump=ijump-1
```

```
        if(ijump.lt.0) ijump=0
```

```
        go to 14
```

C

C up arrow double movement rate

C

```
    else if(keydep.eq.eight.or.(lastkey.eq.null.and.
```

```
1 keydep.eq.ua)) then
```

```
        ijump=ijump+1
```

```
        if(ijump.gt.10) ijump=10
```

```
        go to 14
```

C

C right arrow move right along data

C

```
    else if(keydep.eq.six.or.(lastkey.eq.null.and.
```

```
1 keydep.eq.ra)) then
```

```
        ipoint=ipoint+2**ijump
```

```
        if(ipoint.gt.npoint2) then
```

```
        ipoint=npoint2
    endif

C
C    left caret <  move left 1 point along data
C

    else if(keydep.eq.lcaret) then
        ipoint=ipoint-1
        if(ipoint.lt.npoint1) then
            ipoint=npoint1
        endif

C
C    left parenthesis (  move left 16 points along data
C

    else if(keydep.eq.lparen) then
        ipoint=ipoint-16
        if(ipoint.lt.npoint1) then
            ipoint=npoint1
        endif

C
C    left brace {  move left 256 points along data
C

    else if(keydep.eq.lbrace) then
        ipoint=ipoint-256
        if(ipoint.lt.npoint1) then
            ipoint=npoint1
        endif
```

178

C

C right caret > move right 1 point along data

C

 else if(keydep.eq.rcaret) then

 ipoint=ipoint+1

 if(ipoint.gt.npoint2) then

 ipoint=npoint2

 endif

C

C right parenthesis) move right 16 points along data

C

 else if(keydep.eq.rparen) then

 ipoint=ipoint+16

 if(ipoint.gt.npoint2) then

 ipoint=npoint2

 endif

C

C right brace) move right 256 points along data

C

 else if(keydep.eq.rbrace) then

 ipoint=ipoint+256

 if(ipoint.gt.npoint2) then

 ipoint=npoint2

 endif

C

C H or ? for help on key functions

C

```
else if(keydep.eq.lhh.or.keydep.eq.shh
1 .or.keydep.eq.question_mark) then
CALL clearscreen( $GCLEARSCREEN )
x=1
y=1
call settextposition(y,x,curpos)
call outtext(h1)
y=2
call settextposition(y,x,curpos)
call outtext(h2)
y=3
call settextposition(y,x,curpos)
call outtext(h3)
y=4
call settextposition(y,x,curpos)
call outtext(h4)
y=5
call settextposition(y,x,curpos)
call outtext(h5)
y=6
call settextposition(y,x,curpos)
call outtext(h6)
y=7
call settextposition(y,x,curpos)
call outtext(h7)
```

```
y=8
call settextposition(y,x,curpos)
call outtext(h8)

y=9
call settextposition(y,x,curpos)
call outtext(h9)

y=10
call settextposition(y,x,curpos)
call outtext(h10)

y=11
call settextposition(y,x,curpos)
call outtext(h11)

y=12
call settextposition(y,x,curpos)
call outtext(h12)

y=13
call settextposition(y,x,curpos)
call outtext(h13)

y=14
call settextposition(y,x,curpos)
call outtext(h14)

y=15
call settextposition(y,x,curpos)
call outtext(h14a)

y=16
call settextposition(y,x,curpos)
```

```
call outtext(h15)
y=17
call settextposition(y,x,curpos)
call outtext(h16)
y=18
call settextposition(y,x,curpos)
call outtext(h17)
y=19
call settextposition(y,x,curpos)
call outtext(h21)
y=20
call settextposition(y,x,curpos)
call outtext(h22)
y=21
call settextposition(y,x,curpos)
call outtext(h23)
y=22
call settextposition(y,x,curpos)
call outtext(h24)
y=23
call settextposition(y,x,curpos)
call outtext(h25)
y=24
call settextposition(y,x,curpos)
call outtext(h25a)
y=25
```



```
    call settextposition(y,x,curpos)
    call outtext(h25b)
    y=26
    call settextposition(y,x,curpos)
    call outtext(h26)
C    Wait for the user to depress any key before displaying
C    second screen of help info
94   call getkey(keydep2,ierr)
      if (ierr.eq.0) go to 94
      CALL clearscreen( $GCLEARSCREEN )
      x=1
      y=1
      call settextposition(y,x,curpos)
      call outtext(h17a)
      y=2
      call settextposition(y,x,curpos)
      call outtext(h17b)
      y=3
      call settextposition(y,x,curpos)
      call outtext(h17c)
      y=4
      call settextposition(y,x,curpos)
      call outtext(h17d)
      y=5
      call settextposition(y,x,curpos)
      call outtext(h17e)
```

```
y=6
call settextposition(y,x,curpos)
call outtext(h17f)

y=7
call settextposition(y,x,curpos)
call outtext(h26a)

y=8
call settextposition(y,x,curpos)
call outtext(h26b)

y=9
call settextposition(y,x,curpos)
call outtext(h26c)

y=10
call settextposition(y,x,curpos)
call outtext(h26d)

y=11
call settextposition(y,x,curpos)
call outtext(h26e)

y=12
call settextposition(y,x,curpos)
call outtext(h26f)

y=13
call settextposition(y,x,curpos)
call outtext(h26g)

y=14
call settextposition(y,x,curpos)
```

```
call outtext(h26h)
y=15
call settextposition(y,x,curpos)
call outtext(h26i)
y=16
call settextposition(y,x,curpos)
call outtext(h27)
y=17
call settextposition(y,x,curpos)
call outtext(h28)
y=18
call settextposition(y,x,curpos)
call outtext(h29)
y=19
call settextposition(y,x,curpos)
call outtext(h30)
y=20
call settextposition(y,x,curpos)
call outtext(h31)
y=21
call settextposition(y,x,curpos)
call outtext(h32)
y=22
call settextposition(y,x,curpos)
call outtext(h33)
y=23
```

```
call settextposition(y,x,curpos)
call outtext(h34)
y=24
call settextposition(y,x,curpos)
call outtext(h35)
C    Wait for the user to depress any key before displaying
data screen
95   call getkey(keydep2,ierr)
      if (ierr.eq.0) go to 95
      GO TO 1
C
C    ; initiate keystroke save or terminate save
C
      else if(keydep.eq.semicolon) then
        if(subkeysave) then
          numsubstrokes=numsubstrokes-1
          subkeysave=.FALSE.
          IF (numsubstrokes.GT.0) THEN
            OPEN(3,file='SUBSTROK',form='BINARY')
            DO 499 j=1,numsubstrokes
              WRITE(3) substrokes(j)
499          CONTINUE
            CLOSE(3)
          ENDIF
          isubstrokes=numsubstrokes+1
          if(keysave) then
```

```
        keystrokes(numstrokes)=equals
    endif
else
    numsubstrokes=0
    subkeysave=.TRUE.
endif

C
C   I initiate keystroke save or terminate save
C
else if(keydep.eq.lii.or.keydep.eq.sii) then
    if(keysave) then
        numstrokes=numstrokes-1
        keysave=.FALSE.
        IF (numstrokes.GT.0) THEN
            OPEN(3,file='KEYSTROK',form='BINARY')
            DO 399 j=1,numstrokes
                WRITE(3) keystrokes(j)
399          CONTINUE
            CLOSE(3)
        ENDIF
        istrokes=numstrokes+1
    else
        numstrokes=0
        keysave=.TRUE.
    endif
C
```

```
C      J jump relative to current point predetermined amount
C
      else if(keydep.eq.ljj.or.keydep.eq.sjj) then
          ipoint=ipoint+jump_relative
          if(ipoint.lt.npoint1) ipoint=npoint1
          if(ipoint.gt.npoint2) ipoint=npoint2
C
C      = do predetermined Substrokes or terminate substroke
C      save
C
      else if(keydep.eq.equals) then
          if(subkeysave) then
              subkeysave=.FALSE.
              IF (numsubstrokes.GT.0) THEN
                  OPEN(3,file='SUBSTROK',form='BINARY')
                  DO 1389 j=1,numsubstrokes
                      WRITE(3) substrokes(j)
1389                  CONTINUE
                  CLOSE(3)
              ENDIF
              isubstrokes=numsubstrokes+1
              if(keysave) then
                  keystrokes(numstrokes)=equals
              endif
          else
              isubstrokes=1
```

188

endif

C

C K do predetermined Keystrokes or terminate keystroke

C save with k

C

else if(keydep.eq.lkk.or.keydep.eq.skk) then

if(keysave) then

keysave=.FALSE.

IF (numstrokes.GT.0) THEN

OPEN(3,file='KEYSTROK',form='BINARY')

DO 389 j=1,numstrokes

WRITE(3) keystrokes(j)

389 CONTINUE

CLOSE(3)

ENDIF

istrokes=numstrokes+1

else

istrokes=1

endif

C

C F take the Fourier amplitude transform

C

else if(keydep.eq.lff.or.keydep.eq.sff) then

do 171 j=13,1,-1

if (maxpoints.le.powr2(j)) ipwr=j

171 continue

```

      call ISDFFT(ybuf,ipwr)
      i=1
      do 172 j=1,maxpoints,2
          jj=j+1
          ybuf(i)=sqrt(ybuf(j)*ybuf(j)+ybuf(jj)*ybuf(jj))
          i=i+1
172      continue
      maxpoints=maxpoints/2
      npoint1=1
      npoint2=maxpoints
      call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
      goto 1

C
C      V square root of data
C
      else if(keydep.eq.lvv.or.keydep.eq.svv) then
          do 185 j=1,maxpoints
              ybuf(j)=sqrt(ybuf(j))
185      continue
      go to 1

C
C      3 temporary save
C
      else if(keydep.eq.three) then
          do 742 j=1,maxpoints

```


190

ybuf2(j)=ybuf(j)

742

continue

maxpoints_old=maxpoints

npoint1_old=npoint1

npoint2_old=npoint2

ipoint_old=ipoint

C

C 7 temporary restore

C

else if(keydep.eq.seven) then

maxpoints=maxpoints_old

do 743 j=1,maxpoints

ybuf(j)=ybuf2(j)

743

continue

npoint1=npoint1_old

npoint2=npoint2_old

ipoint=ipoint_old

call scale(ybuf,xmin,xmax,ymin,ymax,

1 npoint1,npoint2,ierr)

go to 1

C

C 9 clear queue

C

else if(keydep.eq.nine) then

nqueue=0

C

```

C      , append to queue
C
      else if(keydep.eq.comma) then
          do 745 j=1,maxpoints
              nqueue=nqueue+1
              ybuf3(nqueue)=ybuf(j)
745      continue
C
C      / get queue
C
      else if(keydep.eq.slash) then
          do 746 j=1,nqueue
              ybuf(j)=ybuf3(j)
746      continue
          maxpoints=nqueue
          npoint1=1
          npoint2=maxpoints
          ipoint=1
          call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
          go to 1
C
C      M Previous sequential file
C
      else if(keydep.eq.lmm.or.keydep.eq.smm) then
          if(maxfiles.gt.0) then

```

```

        ifileptr=ifileptr-1
        if(ifileptr.lt.1) go to 999
        INFILE=filenames(ifileptr)
        call readone(infile,ybuf,maxpoints,xmin,
1          xmax,ymin,ymax,
1          ipoint,npoint1,npoint2,ierr)
        go to 1
    endif

```

C

C N Next sequential file

C

```

    else if(keydep.eq.lnn.or.keydep.eq.snn) then
        if(maxfiles.gt.0) then
            ifileptr=ifileptr+1
            if(ifileptr.gt.maxfiles) go to 999
            INFILE=filenames(ifileptr)
            call readone(infile,ybuf,maxpoints,xmin,
1              xmax,ymin,ymax,
1              ipoint,npoint1,npoint2,ierr)
            go to 1
        endif
    endif

```

C

C W square of data

C

```

    else if(keydep.eq.lww.or.keydep.eq.sww) then
        do 186 j=1,maxpoints

```

```

        ybuf(j)=ybuf(j)*ybuf(j)
186      continue
        go to 1
C
C      . envelope of points
C
      else if(keydep.eq.dot.and.maxpoints.gt.4) then
        do 681 j=1,maxpoints-4
          if(ybuf(j).lt.ybuf(j+1)) ybuf(j)=ybuf(j+1)
          if(ybuf(j).lt.ybuf(j+2)) ybuf(j)=ybuf(j+2)
          if(ybuf(j).lt.ybuf(j+3)) ybuf(j)=ybuf(j+3)
          if(ybuf(j).lt.ybuf(j+4)) ybuf(j)=ybuf(j+4)
681      continue
        go to 1
C
C      + find peak of data & jump to there
C
      else if(keydep.eq.plus) then
        ypeak=-1.0E+14
        do 911 j=npoint1,npoint2
          If(ybuf(j).gt.ypeak) then
            ipoint=j
            ypeak=ybuf(j)
          endif
911      continue
C

```

194

C - invert data

C

 else if(keydep.eq.minus) then

 do 912 j=1,maxpoints

 ybuf(j)=-ybuf(j)

912 continue

 call scale(ybuf,xmin,xmax,ymin,ymax,

1 npoint1,npoint2,ierr)

 goto 1

C

C U unzoom

C

 else if(keydep.eq.luu.or.keydep.eq.suu) then

 npoint1=1

 npoint2=maxpoints

 call scale(ybuf,xmin,xmax,ymin,ymax,

1 npoint1,npoint2,ierr)

 goto 1

C

C Z zoom in on marked limits

C

 else if(keydep.eq.lzz.or.keydep.eq.szz) then

 npoint1=ibeg

 npoint2=iend

 ipoint=npoint1

 call scale(ybuf,xmin,xmax,ymin,ymax,

```

1      npoint1,npoint2,ierr)
      go to 1

C
C      clip to the marked limits
C

      else if(keydep.eq.lcc.or.keydep.eq.scc) then
          i=1
          do 8 j=ibeg,iend
              ybuf(i)=ybuf(j)
              i=i+1
8      continue
          maxpoints=iend-ibeg+1
          ipoint=1
          npoint1=1
          npoint2=maxpoints
          call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
          goto 1

C
C      0 to filter
C

      else if(keydep.eq.zero) then
C          NONRECURSIVE
          If((type_filter.eq.lnn.or.type_filter.eq.snn).and.
1      maxpoints.gt.2*num_coef-1) then
              do 306 j=1,num_coef

```

196

 xbuf(j)=ybuf(j)

306

 continue

do 307 j=num_coef,maxpoints-num_coef+1

 r=0.

 rtemp=ybuf(j)

do 304 jj=2,num_coef

 r=xbuf(num_coef-jj+2)*coef_filter(jj)+r

 r=ybuf(j+jj-1)*coef_filter(jj)+r

304

 continue

 ybuf(j)=rtemp*coef_filter(1)+r

do 305 jj=1,num_coef-1

 jjj=jj+1

 xbuf(jj)=xbuf(jjj)

305

 continue

 xbuf(num_coef)=rtemp

307

 continue

endif

C

RECURSIVE

If((type_filter.eq.lrr.or.type_filter.eq.srr).and.

1 maxpoints.gt.num_coef) then

do 301 j=num_coef,maxpoints

 r=0.

do 302 jj=1,num_coef

 r=ybuf(j-jj+1)*coef_filter(jj)+r

302

 continue

 ybuf(j)=r

```
301      continue
      endif
      npoint1=1
      npoint2=maxpoints
      call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
      goto 1

C
C      D decimate by 2
C
      else if(keydep.eq.ldd.or.keydep.eq.sdd) then
          mpoints=maxpoints/2
          do 7 j=1,mpoints
              jj=j*2
              ybuf(j)=ybuf(jj)
7      continue
          maxpoints=mpoints
          ipoint=1
          npoint1=1
          npoint2=maxpoints
          call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
          goto 1

C
C      1 third order interpolate to new sample rate
C
```


198

```
else if(keydep.eq.one) then
```

```
    mpoints=maxpoints*delt/deltnew
```

```
    do 752 j=2,mpoints-2
```

```
        time=(j-1)*deltnew
```

```
        ybuf1(j)=terpol(ybuf,time,delt)
```

```
752    continue
```

```
    do 753 j=2,mpoints-2
```

```
        ybuf(j)=ybuf1(j)
```

```
753    continue
```

```
    maxpoints=mpoints
```

```
    ipoint=1
```

```
    npoint1=1
```

```
    npoint2=maxpoints
```

```
    call scale(ybuf,xmin,xmax,ymin,ymax,
```

```
1    npoint1,npoint2,ierr)
```

```
    goto 1
```

C

C 0 clip num_to_clip of points from current point

C

```
else if(keydep.eq.100.or.keydep.eq.soo) then
```

```
    npoint1=ipoint
```

```
    maxpoints=num_to_clip
```

```
    npoint2=maxpoints
```

```
    jj=npoint1
```

```
    do 1191 j=1,maxpoints
```

```
        ybuf(j)=ybuf(jj)
```

```

        jj=jj+1
1191    continue
        ipoint=1
        npoint1=1
        call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
        go to 1

C
C      P clip to power of two points from current point
C
      else if(keydep.eq.lpp.or.keydep.eq.spp) then
        npoint1=ipoint
        maxpoints=npoint2-npoint1+1
        do 71 j=13,1,-1
          if (maxpoints.lt.powr2(j)) ipwr=j
71      continue
        maxpoints=powr2(ipwr-1)
        npoint2=maxpoints
        jj=npoint1
        do 90 j=1,maxpoints
          ybuf(j)=ybuf(jj)
          jj=jj+1
90      continue
        ipoint=1
        npoint1=1
        call scale(ybuf,xmin,xmax,ymin,ymax,

```

200

1 npoint1,npoint2,ierr)

 go to 1

C

C T jump to the next point above the threshold

C

 else if(keydep.eq.ltt.or.keydep.eq.stt) then

 j=ipoint-1

73 j=j+1

 if(j.le.npoint2.and.ybuf(j).lt.threshold) go to 73

 ipoint=j

C

C E mark the ending limit

C

 else if(keydep.eq.lee.or.keydep.eq.see) then

 iend=ipoint

C

C B mark the beginning limit

C

 else if(keydep.eq.lbb.or.keydep.eq.sbb) then

 ibeg=ipoint

C

C Q quit

C

 else if(keydep.eq.lqq.or.keydep.eq.sqq) then

 go to 999

C

C A autosave in file named 'INFILE'.A??

C

 else if(keydep.eq.laa.or.keydep.eq.saa) then

 file=infile

 icount=0

 my_dot_at=0

 do 677 j=1,12

 if(ifile(j).ne.space) icount=icount+1

 if(ifile(j).eq.dot) my_dot_at=j

677 continue

 if(my_dot_at.ne.0) then

 ifile(my_dot_at+1)=outchar

 else

 ifile(icount+1)=dot

 ifile(icount+2)=outchar

 endif

 icount=0

 do 678 j=1,12

 if(ifile(j).ne.space) then

 icount=icount+1

 ifile(icount)=ifile(j)

 endif

678 continue

 if(icount.ne.12) then

 do 679 j=icount+1,12

 ifile(j)=space

202

679 continue

 endif

 outfile=file

 call writeone(outfile,ybuf,maxpoints,ierr)

C

C S save current data in a file

C

 else if(keydep.eq.lss.or.keydep.eq.sss) then

96 write(6,61)

 read(5,99,err=96,end=4) OUTFILE

61 format(' Output File Name:')

 call writeone(outfile,ybuf,maxpoints,ierr)

 goto 1

C

C X rectify to mean value full wave

C

 else if(keydep.eq.lxx.or.keydep.eq.sxx) then

 rmean=0.

 do 270 j=1,maxpoints

 rmean=ybuf(j)+rmean

270 continue

 rmean=rmean/maxpoints

 do 271 j=1,maxpoints

 ybuf(j)=ybuf(j)-rmean

 if(ybuf(j).lt.0) ybuf(j)=-ybuf(j)

271 continue

```

        call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
        go to 1

C
C      Y scale between 0 and 1 about mean value
C
      else if(keydep.eq.lyy.or.keydep.eq.syy) then
        rmean=0.
        rmin=1.E15
        rmax=-1.E15
        do 2270 j=1,maxpoints
          if(rmin.gt.ybuf(j)) rmin=ybuf(j)
          if(rmax.lt.ybuf(j)) rmax=ybuf(j)
          rmean=ybuf(j)+rmean
2270      continue
        rmean=rmean/maxpoints
        scaletemp=rmax-rmean
        scalet=rmean-rmin
        if(scalet.gt.scaletemp) scaletemp=scalet
        scaletemp=scaletemp*2.
        do 2271 j=1,maxpoints
          ybuf(j)=(ybuf(j)-rmean)/scaletemp+.5
2271      continue
        call scale(ybuf,xmin,xmax,ymin,ymax,
1      npoint1,npoint2,ierr)
        go to 1

```

204

C

C 5 rectify to mean value half wave

C

 else if(keydep.eq.five) then

 rmean=0.

 do 277 j=1,maxpoints

 rmean=ybuf(j)+rmean

277 continue

 rmean=rmean/maxpoints

 do 278 j=1,maxpoints

 ybuf(j)=ybuf(j)-rmean

 if(ybuf(j).lt.0) ybuf(j)=0.

278 continue

 call scale(ybuf,xmin,xmax,ymin,ymax,

1 npoint1,npoint2,ierr)

 go to 1

C

C R rectify and scale to between 0. and 1.

C

 else if(keydep.eq.lrr.or.keydep.eq.srr) then

 do 70 j=1,maxpoints

 ybuf(j)=ybuf(j)/ymax

 if(ybuf(j).lt.0) ybuf(j)=0.

70 continue

 call scale(ybuf,xmin,xmax,ymin,ymax,

1 npoint1,npoint2,ierr)

```

        go to 1

C
C      L get a list of files for sequential processing
C
      else if(keydep.eq.l11.or.keydep.eq.s11) then
46      write(6,41)
          read(5,99,err=46,end=4) INFILE
41      format(/' File containing list of file names:')
          maxfiles=0
          ifileptr=1
          open(4,file=infile,status='OLD',err=4)
42      maxfiles=maxfiles+1
          read(4,99,err=43,end=43) filenames(maxfiles)
          go to 42
43      maxfiles=maxfiles-1
          if (maxfiles.lt.1) go to 1
          INFILE=filenames(1)
          call readone(infile,ybuf,maxpoints,xmin,
1      xmax,ymin,ymax,
1      ipoint,npoint1,npoint2,ierr)
          go to 1

C
C      G get a new file
C
      else if(keydep.eq.lgg.or.keydep.eq.sgg) then
          outfile=infile

```


206

```
296      write(6,91)
      read(5,99,err=296,end=4) INFILE
99      format(a12)
91      format(/' File containing one data record:')
      call readone(infile,ybuf,maxpoints,
1      xmin,xmax,ymin,ymax,
1      ipoint,npoint1,npoint2,ierr)
      if(ierr.ne.0) INFILE=OUTFILE
      maxfiles=0
      go to 1
    else
      goto 141
    endif
    goto 4
999  dummy = setvideomode( $DEFAULTMODE )
END
```

```
SUBROUTINE displayone(ybuf,ymin,ymax,npoint1,npoint2)
INCLUDE  'FGRAPH.FD'
INTEGER*2          dummy
real ybuf(1)
DOUBLE PRECISION    x,y
RECORD /videoconfig/ screen
RECORD /wxycoord/    wxy
COMMON              screen
```

C

```
x=npoint1
y=ymax-ybuf(npoin1)+ymin
dummy=setcolor(14)
call moveto_w(x,y,wxy)
do 2 j=npoin1+1,npoin2
    x=j
    y=ymax-ybuf(j)+ymin
    dummy=lineto_w(x,y)
2 continue
RETURN
END

SUBROUTINE scale(ybuf,xmin,xmax,ymin,ymax,
1 npoin1,npoin2,ierr)
    real ybuf(1)
    ierr=0
    ymin=1.0E+15
    ymax=-1.0E+15
    do 6 j=npoin1,npoin2
        if(ybuf(j).lt.ymin) ymin=ybuf(j)
        if(ybuf(j).gt.ymax) ymax=ybuf(j)
6 continue
    xmin=npoin1
    xmax=npoin2
    RETURN
    END
```

```

FUNCTION terpol(ybuf,time,delt)
real ybuf(1)
I=TIME/DELT
IF(I.LT.1) TERPOL=ybuf(1)
IF(I.LT.1) RETURN
DIF1=(YBUF(I+2)-YBUF(I))*0.5
DIF2=(YBUF(I+3)-YBUF(I+1))*0.5
E=YBUF(I+2)-(YBUF(I+1)+DIF1)
D=DIF2-(DIF1+2.*E)
DT=TIME-FLOAT(I)*DELT
TAU=DT/DELT
TERPOL=((D*TAU+(E-D))*TAU+DIF1)*TAU+YBUF(I+1)
RETURN
END

```

```

SUBROUTINE readone(infile,ybuf,maxpoints,
1 xmin,xmax,ymin,
1 ymax,ipoint,npoint1,npoint2,ierr)
real ybuf(1)
character*12 infile
ipoint=1
npoint1=1
ierr=1
open(4,file=INFILE,form='BINARY',status='OLD',err=6)
ierr=0
maxpoints=1

```

```

ymin=1.0E+15
ymax=-1.0E+15
2  read(4,end=5,err=5) ybuf(maxpoints)
   if(ybuf(maxpoints).lt.ymin) ymin=ybuf(maxpoints)
   if(ybuf(maxpoints).gt.ymax) ymax=ybuf(maxpoints)
   maxpoints=maxpoints+1
   goto 2
5  close (4)
   maxpoints=maxpoints-1
   npoint2=maxpoints
   xmin=1.
   xmax=maxpoints
7  format(f20.0)
6  RETURN
   END

```

```

SUBROUTINE cursor(ybuf,ymin,ymax,ipoint,
1 npoint1,npoint2)
   real ybuf(1)
   INCLUDE 'FGRAPH.FD'

```

```

INTEGER*2          dummy
DOUBLE PRECISION   x,y
RECORD /videoconfig/ screen
RECORD /wxycoord/   wxy
COMMON             screen

```

210

data ipoint0/0/

C

dummy=setcolor(0)

if(ipoint0.ne.0) then

 x=ipoint0

 y=ymin

 call moveto_w(x,y,wxy)

 y=ymax

 dummy=lineto_w(x,y)

endif

ires=(npoint2-npoint1)/600+1

ip0=ipoint0-ires

ip8=ipoint0+ires

if(ip0.lt.npoint1) ip0=npoint1

if(ip8.gt.npoint2) ip8=npoint2

dummy=setcolor(14)

call displayone(ybuf,ymin,ymax,ip0,ip8)

dummy=setcolor(3)

x=ipoint

y=ymin

call moveto_w(x,y,wxy)

y=ymax

dummy=lineto_w(x,y)

dummy=setcolor(14)

ipoint0=ipoint

RETURN

END

SUBROUTINE writeone(outfile,ybuf,maxpoints,ierr)

real ybuf(1)

character*12 outfile

ierr=1

open(4,file=OUTFILE,form='BINARY',err=6)

ierr=0

do 8 j=1,maxpoints

write(4,err=5) ybuf(j)

8 continue

5 close (4)

7 format(f20.8)

6 RETURN

END

SUBROUTINE ISDFFT(VR,N)

REAL VR(1)

IS=-1

CALL RFFT(VR,N,IS)

NUM=2**N

XNUM=NUM

DO 10 I=1,NUM

VR(I)=VR(I)/XNUM

10 CONTINUE

RETURN

END

SUBROUTINE INVFFT(VR,N)

REAL VR(1)

IS=1

CALL RFFT(VR,N,IS)

NUM=2**N

DO 10 I=1,NUM

VR(I)=VR(I)/2.

10 CONTINUE

RETURN

END

SUBROUTINE RFFT(VR,M,ISIGN)

DIMENSION VR(1)

N=2**M

NTW=N+4

NN=N/2

CALL TRIG(N,1,XK,SM1,SM2,CM1,CM2)

S=XK*SM1-SM2

C=XK*CM1-CM2

SM2=SM1

CM2=CM1

SM1=S

CM1=C

IF(ISIGN.LT.0) GO TO 5

```
4      DO 11 K2=4,NN,2
        S=XK*SM1-SM2
        C=XK*CM1-CM2
        SM2=SM1
        CM2=CM1
        SM1=S
        CM1=C
        IF (ISIGN.GT.0) C=-C
        N2=NTW-K2
        BK1=VR(K2-1)+VR(N2-1)
        BK2=VR(K2)-VR(N2)
        BN1=VR(K2)+VR(N2)
        BN2=VR(K2-1)-VR(N2-1)
        XBN1=C*BN1-S*BN2
        XBN2=-C*BN2-S*BN1
        VR(K2-1)=BK1+XBN1
        VR(K2)=BK2+XBN2
        VR(N2-1)=BK1-XBN1
        VR(N2)=-BK2+XBN2
11     CONTINUE
        VR(NN+2)=-VR(NN+2)
        IF (ISIGN.GT.0) GO TO 3
        VR(N+1)=VR(1)-VR(2)
        VR(N+2)=0.
        VR(1)=VR(1)+VR(2)
        VR(2)=0.
```


214

RETURN

3 VR(2)=VR(1)-VR(N+1)

VR(1)=VR(1)+VR(N+1)

5 K2=M-1

CALL CPXFFT(VR,K2,ISIGN)

IF(ISIGN.LT.0) GO TO 4

RETURN

END

SUBROUTINE CPXFFT(VR,M,ISIGN)

DIMENSION VR(1)

N=2**M

K=N

N2=N+N

IA=1

1 IB=0

K2=K

K=K/2

CALL TRIG(N,IA,XK,SM1,SM2,CM1,CM2)

DO 65 J=1,K

S=XK*SM1-SM2

C=XK*CM1-CM2

SM2=SM1

CM2=CM1

SM1=S

CM1=C

```
      IF (ISIGN.LT.0) S=-S
      DO 66 I=J,N,K2
          I1=I+I
          I1M=I1-1
          I2=I1+K2
          I2M=I2-1
          VI1=VR(I1)
          VI1M=VR(I1M)
          VI2=VR(I2)
          VI2M=VR(I2M)
          X=VI1M-VI2M
          Y=VI1-VI2
          VR(I2)=S*X+C*Y
          VR(I2M)=C*X-S*Y
          VR(I1M)=VI1M+VI2M
          VR(I1)=VI1+VI2
66      CONTINUE
65      CONTINUE
          IA=IA+IA
          IF (K.GT.1) GO TO 1
          NV2=N/2
          J=1
          DO 30 I=1,N-1
              IF (I.GE.J) GO TO 25
              NI=(I-1)*2+1
              NJ=(J-1)*2+1
```

216

T=VR(NJ)

VR(NJ)=VR(NI)

VR(NI)=T

T=VR(NJ+1)

VR(NJ+1)=VR(NI+1)

VR(NI+1)=T

25 K=NV2

26 IF(K.GE.J) GO TO 31

J=J-K

K=K/2

GO TO 26

31 J=J+K

30 CONTINUE

RETURN

END

SUBROUTINE TRIG(M, IA, XK, SM1, SM2, CM1, CM2)

DATA PI2/6.283185307/

ANG=PI2*FLOAT(IA)/FLOAT(M)

X=COS(ANG)

XK=X*2.

CM1=X

CM2=XK*X-1.

SM1=-SIN(ANG)

SM2=XK*SM1

RETURN

END

APPENDIX B

LETTERBP

LETTERBP requires an IBM compatible computer with CGA, EGA, or VGA graphics adapter, a hard disk, and a floating point coprocessor. It was written primarily to verify the correct operation of BACKPROP by allowing creation of neural network test cases that could be input to both BACKPROP and to a commercial back propagation based program called NEUROSHELL (available from Ward Systems Group). This allowed checking the number of conversions the two programs took to converge to the same level of error given the same learning cases and checking the error percentages on the same test cases. The program allows user friendly definition of 7X9 nib character patterns and corresponding active output. Then distorted versions of the patterns may be input as test cases. The back propagation programs accept these patterns as inputs and classify them as to what percentage of certainty there is that it is each of the possible outputs. Thus the effects of distortion may be studied.

The LETTERBP program creates a series of data files (one for each learning case and one for each test case). A

file containing a list of the learning data files called PATTERNL.LIS and a file containing a list of the test data files called PATTERNL.LIS is automatically created by LETTERBP. When BACKPROP is run it will ask for the file containing the list of learning cases, enter PATTERNL.LIS

Then the name of the file containing a list of test cases will be asked for, enter PATTERNL.LIS.

The programs will then learn the learn cases and process the test cases; see the documentation about the BACKPROP programs.

An example is presented below using LETTERBP to define a problem and NEUROSHELL's Binary to solve it.

B.1 LETTERBP Example Run

Start LETTERBP running by typing Letterbp. A screen similar to Figure 67 should be displayed.

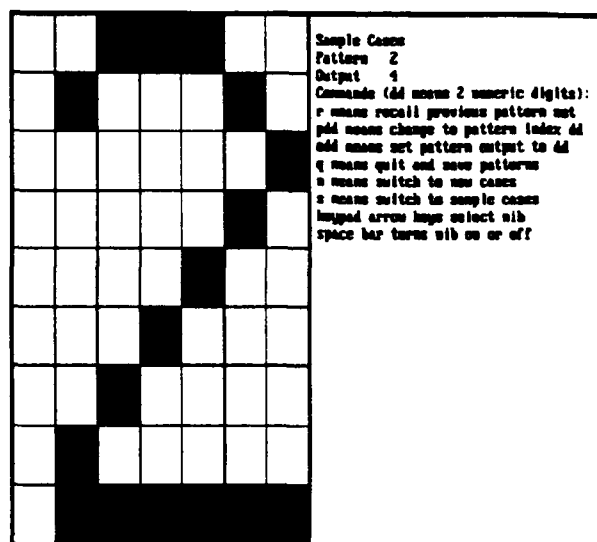


Figure 69. Sample Case

Entering p03 will bring up a blank pattern. Use the arrow keys and space bar to create the pattern of Figure 70 and enter o03 to change the output characteristic to number 3.

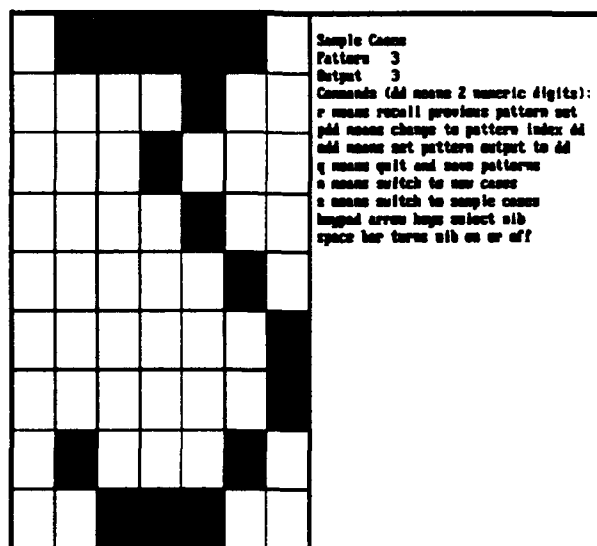


Figure 70. Sample Pattern

Entering n will bring up the first new case as shown in Figure 71. Note that output is defined for error analysis

in BACKPROP.

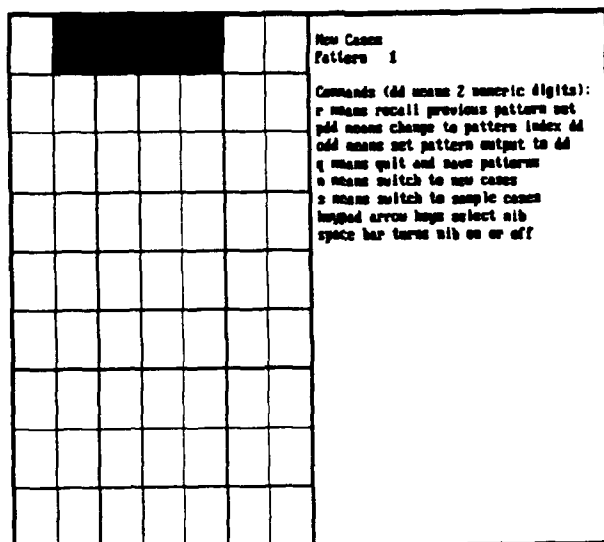


Figure 71. First New Case

Entering p02 will bring up the second new case. Use the arrow keys and space bar to modify the pattern to that of Figure 72.

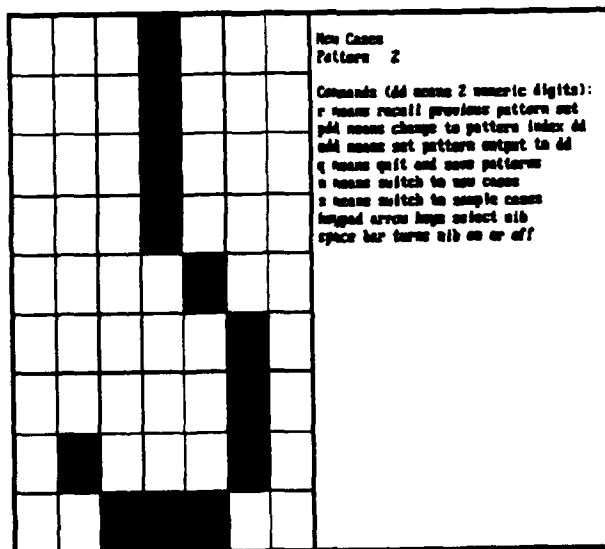


Figure 72. Second New Case

Then enter q to quit.

B.2 Listing of LETTERBP

CC LETTERBP.FOR - Pre neural network program.

INCLUDE 'FGRAPH.FI'

INCLUDE 'FGRAPH.FD'

LOGICAL fourcolors

EXTERNAL fourcolors

IF(fourcolors()) THEN

CALL show_nibs()

ELSE

WRITE (*,*) ' This program requires a CGA,',

+ ' EGA, or',

+ ' VGA graphics card.'

END IF

END

C Additional functions defined below

CC FOURCOLORS - Function to enter graphics mode for REALG.

LOGICAL FUNCTION fourcolors()

INCLUDE 'FGRAPH.FD'

INTEGER*2 dummy

RECORD /videoconfig/ screen

COMMON screen

C Set to maximum number of available colors.

CALL getvideoconfig(screen)

SELECT CASE(screen.adapter)

CASE(\$CGA, \$OCGA)

```

        dummy = setvideomode( $MRES4COLOR )
CASE( $EGA, $OEGA )
        dummy = setvideomode( $ERESCOLOR )
CASE( $VGA, $OVGA )
        dummy = setvideomode( $VRES16COLOR )
CASE DEFAULT
        dummy = 0
END SELECT
CALL getvideoconfig( screen )
fourcolors = .TRUE.
IF( dummy .EQ. 0 ) fourcolors = .FALSE.
END

```

CC SHOW_NIBS - This subroutine outlines the nibs.

```

SUBROUTINE show_nibs()
INCLUDE 'FGRAPH.FD'
INTEGER*2          dummy, halfx, halfy
INTEGER*2          xwidth, yheight, cols, rows,x,y
COMMON             screen
RECORD /videoconfig/ screen
RECORD /rccoord/   curpos
CHARACTER*3 text
CHARACTER*80 a
INTEGER*1 nibon(7,9,50,2),out(50,2)
INTEGER*1 keydep,sqq,lqq,space,four,five,six
INTEGER*1 eight,two,zero,lrr,srr,lpp,spp

```

```
INTEGER*1 loo,soo,lss,sss,lnn,snn,la,da,ra,ua
```

```
DOUBLE PRECISION x1,x2,y1,y2
```

```
c    LOGICAL keydown
```

```
integer*2 ierr
```

```
integer*1 iout(50)
```

```
INTEGER*1 LEARNNAME(12),TESTNAME(12)
```

```
CHARACTER*12 LEARNN,TESTN
```

```
EQUIVALENCE (LEARNN,LEARNNAME(1)),(TESTN,TESTNAME(1))
```

```
DATA lqq/'Q'//,sqq/'q'//,space/' '/
```

```
DATA lrr/'R'//,srr/'r'//,lpp/'P'//,spp/'p'//
```

```
DATA loo/'O'//,soo/'o'//,lss/'S'//,sss/'s'//
```

```
DATA lnn/'N'//,snn/'n'//,la/75//,da/80//,ra/77//,ua/72//
```

```
DATA nibon/6300*0//,four/'4'//,five/'5'//,six/'6'//,
```

```
1 eight/'8'//
```

```
DATA two/'2'//,zero/'0'//,out/100*1//
```

```
DATA LEARNN/'PATTERNL.000'//,TESTN/'PATTERNNT.000'//
```

```
iupdate=0
```

```
isn=1
```

```
CALL clearscreen( $GCLEARSCREEN )
```

```
xwidth = screen.numxpixels
```

```
yheight = screen.numypixels
```

```
cols = screen.numtextcols
```

```
rows = screen.numtextrows
```

```
halfx = xwidth / 2
```

```
halfy = (yheight / rows) * (rows / 2)
```

```
c    Second window
```

```

      ipat=1
1      CALL setviewport( halfx, 0, xwidth - 1, yheight - 1 )
      CALL settextwindow( 1, (cols / 2) + 1, rows , cols )
      dummy = setwindow( .FALSE., 0.0, 0.0, 10.0, 10.0 )
      dummy = rectangle_w( $GBORDER, 0.0, 0.0, 10.0, 10.0 )
C      First window
      CALL setviewport( 0, 0, halfx - 1, yheight - 1 )
      CALL settextwindow( 1, 1, rows, cols / 2 )
      dummy = setwindow( .FALSE., 0.0, 0.0, 7.0, 9.0 )
      CALL grid()
      dummy = rectangle($GBORDER,0,0,halfx - 1,yheight - 1 )
      irow=1
      icol=1
          x2=icol
          y2=irow
          x1=x2-1.
          y1=y2-1.
20      dummy = setcolor(2)
          dummy = rectangle_w( $GBORDER, x1, y1, x2, y2 )
          dummy = setcolor(4)
      CALL setviewport( halfx, 0, xwidth - 1, yheight - 1 )
      CALL settextwindow( 1, (cols / 2) + 1, rows , cols )
      dummy = setwindow( .FALSE., 0.0, 0.0, 10.0, 10.0 )
      x=2
      y=2
      call settextposition(y,x,curpos)

```

```
IF(isn.eq.1) THEN
    call outtext('Sample Cases')
ELSE
    call outtext('New Cases  ')
ENDIF
x=2
y=3
call settextposition(y,x,curpos)
call outtext('Pattern')
x=10
call settextposition(y,x,curpos)
write(a,18) ipat
18 format(i3)
read(a,19) text
19 format(a3)
call outtext(text)
x=2
y=4
call settextposition(y,x,curpos)
    call outtext('Output')
    x=10
    call settextposition(y,x,curpos)
    write(a,18) out(ipat,isn)
    read(a,19) text
    call outtext(text)
x=2
```

```
y=5
call settextposition(y,x,curpos)
call outtext('Commands (dd means 2 numeric digits):')
x=2
y=6
call settextposition(y,x,curpos)
call outtext('r means recall previous pattern set')
x=2
y=7
call settextposition(y,x,curpos)
call outtext('pdd means change to pattern index dd')
x=2
y=8
call settextposition(y,x,curpos)
call outtext('odd means set pattern output to dd')
x=2
y=9
call settextposition(y,x,curpos)
call outtext('q means quit and save patterns')
x=2
y=10
call settextposition(y,x,curpos)
call outtext('n means switch to new cases')
x=2
y=11
call settextposition(y,x,curpos)
```

```

call outtext('s means switch to sample cases')
x=2
y=12
call settextposition(y,x,curpos)
call outtext('keypad arrow keys select nib')
x=2
y=13
call settextposition(y,x,curpos)
call outtext('space bar turns nib on or off')
CALL setviewport( 0, 0, halfx - 1, yheight - 1 )
CALL settextwindow( 1, 1, rows, cols / 2 )
dummy = setwindow( .FALSE., 0.0, 0.0, 7.0, 9.0 )
    x2=icol
    y2=irow
    x1=x2-1.
    y1=y2-1.
    dummy = rectangle_w( $GBORDER, x1, y1, x2, y2 )
    dummy = setcolor( 2 )
if(iupdate.eq.1) then
    do 8 j=1,7
        do 8 k=1,9
            if(nibon(j,k,ipat,isn).eq.1) then
                call on(j,k)
            else
                call off(j,k)
            endif
        enddo
    enddo

```


230

```
8      continue
      iupdate=0
      CALL grid()
      dummy = setcolor(4)
      dummy = rectangle_w( $GBORDER, x1, y1, x2, y2 )
      dummy = setcolor(2)
endif

4  call getkey(keydep,ierr) ! Wait for key to be pressed
   IF (IERR.EQ.0) GO TO 4
   CALL setviewport( 0, 0, halfx - 1, yheight - 1 )
   CALL settextwindow( 1, 1, rows, cols / 2 )
   dummy = setwindow( .FALSE., 0.0, 0.0, 7.0, 9.0 )
   if(keydep.eq.space.or.keydep.eq.five) then
      if(nibon(icol,irow,ipat,isn).eq.0) then
         call on(icol,irow)
         nibon(icol,irow,ipat,isn)=1
      else
         call off(icol,irow)
         nibon(icol,irow,ipat,isn)=0
      endif
   else if(keydep.eq.four.or.keydep.eq.1a) then
      icol=icol-1
      if(icol.lt.1) then
         icol=7
         irow=irow-1
         if(irow.lt.1) irow=9
      endif
   endif
```

```
endif
else if(keydep.eq.two.or.keydep.eq.da) then
    irow=irow+1
    if(irow.gt.9) then
        irow=1
    endif
else if(keydep.eq.eight.or.keydep.eq.ua) then
    irow=irow-1
    if(irow.lt.1) then
        irow=9
    endif
else if(keydep.eq.six.or.keydep.eq.ra) then
    icol=icol+1
    if(icol.gt.7) then
        icol=1
        irow=irow+1
        if(irow.gt.9) irow=1
    endif
else if(keydep.eq.lqq.or.keydep.eq.sqq) then
    go to 999
else if(keydep.eq.lss.or.keydep.eq.sss) then
    isn=1
    iupdate=1
    ipat=1
    goto 1
else if(keydep.eq.lnn.or.keydep.eq.snn) then
```

```
    isn=2
    iupdate=1
    ipat=1
    goto 1
else if(keydep.eq.lrr.or.keydep.eq.srr) then
    go to 998
else if(keydep.eq.lpp.or.keydep.eq.spp) then
3   call getkey(keydep,ierr)      ! Wait for ENTER key
    IF (IERR.EQ.0) GO TO 3
    i1=keydep-zero
2   call getkey(keydep,ierr)      ! Wait for ENTER key
    IF (IERR.EQ.0) GO TO 2
    i2=keydep-zero
    ipat=10*i1+i2
    if(ipat.gt.50) ipat=50
    if(ipat.lt.1) ipat=1
    iupdate=1
    go to 1
else if(keydep.eq.loo.or.keydep.eq.soo) then
7   call getkey(keydep,ierr)      ! Wait for ENTER key
    IF (IERR.EQ.0) GO TO 7
    i1=keydep-zero
88  call getkey(keydep,ierr)      ! Wait for ENTER key
    IF (IERR.EQ.0) GO TO 88
    i2=keydep-zero
    out(ipat,isn)=10*i1+i2
```

```
        if(out(ipat,isn).gt.50) out(ipat,isn)=50
        if(out(ipat,isn).lt.1) out(ipat,isn)=1
        iupdate=1
        go to 1
    endif
    go to 20
998  open(4,file='outfile',status='OLD')
    isn=1
    do 15 k=1,50
        read(4,11) ((nibon(i,j,k,isn),i=1,7),j=1,9)
        read(4,11) out(k,isn)
15  continue
    isn=2
    do 25 k=1,50
        read(4,11) ((nibon(i,j,k,isn),i=1,7),j=1,9)
        read(4,11) out(k,isn)
25  continue
    close (4)
    iupdate=1
    isn=1
    ipat=1
    go to 1
11  format(63i2)
999  open(4,file='outfile')
    isn=1
    ndef=0
```

```
nclass=0
nnew=0
do 16 k=1,50
    write(4,11) ((nibon(i,j,k,isn),i=1,7),j=1,9)
    do 500 i=1,7
        do 500 j=1,9
            if(nibon(i,j,k,isn).ne.0) ndef=k
500    continue
        write(4,11) out(k,isn)
        if(out(k,isn).gt.nclass) nclass=out(k,isn)
16 continue
    isn=2
    do 26 k=1,50
        write(4,11) ((nibon(i,j,k,isn),i=1,7),j=1,9)
        do 505 i=1,7
            do 505 j=1,9
                if(nibon(i,j,k,isn).ne.0) nnew=k
505    continue
        write(4,11) out(k,isn)
26 continue
    close (4)
    nhhidden=0
    open (4,file='PATTERNS.FIG')
    write(4,100)
100    format('binary')
    write (4,101) 63,nhhidden,nclass
```

```
101  format(3I3)
      write (4,102)
102  format('100 .5 .6 .9 .0001 1')
      do 103 j=1,63
          if(j.le.9) then
              write(4,104) j
          else
              write(4,204) j
          endif
103  continue
104  format('nib_',I1)
204  format('nib_',I2)
304  format('output_',I1)
404  format('output_',I2)
      write(4,105)
105  format('-----')
      do 106 j=1,nclass
          if(j.le.9) then
              write(4,304) j
          else
              write(4,404) j
          endif
106  continue
      close (4)
      open(4,file='PATTERNS.CHR')
      write(4,700) ndef
```

236

700 format(I2)

isn=1

do 512 k=1,ndef

write(4,511) ((nibon(i,j,k,isn),i=1,7),j=1,9)

511 format(63I1)

do 513 l=1,nclass

if (out(k,isn).eq.1) then

iout(l)=1

else

iout(l)=0

endif

513 continue

write(4,514) (iout(l),l=1,nclass)

514 format(50I1)

512 continue

close(4)

open(4,file='PATTERNS.CLS')

write(4,700) nnew

isn=2

do 712 k=1,nnew

write(4,511) ((nibon(i,j,k,isn),i=1,7),j=1,9)

do 713 l=1,nclass

if (out(k,isn).eq.1) then

iout(l)=1

else

iout(l)=0

```
endif
713 continue
      write(4,514) (iout(l),l=1,nclass)
712 continue
      close(4)
      OPEN(4,FILE='PATTERNL.LIS')
      WRITE(4,641) NCLASS
      ISN=1
      DO 640 K=1,NDEF
        WRITE(A,641) K
641      FORMAT(I3)
        READ(A,644) (LEARNNAME(J),J=10,12)
644      FORMAT(3A1)
        IF(LEARNNAME(11).EQ.SPACE) LEARNNAME(11)=ZERO
        IF(LEARNNAME(10).EQ.SPACE) LEARNNAME(10)=ZERO
        WRITE(4,642) LEARNN,OUT(K,ISN)
642      FORMAT(A12,I2)
        OPEN(3,FILE=LEARNN,form='BINARY')
        DO 652 J=1,9
          DO 651 I=1,7
            R=NIBON(I,J,K,ISN)
            WRITE(3) R
651          CONTINUE
652        CONTINUE
        CLOSE(3)
640      CONTINUE
```



```

CLOSE(4)
OPEN(4,FILE='PATTERNT.LIS')
ISN=2
WRITE(4,641) NCLASS
DO 645 K=1,NNEW
    WRITE(A,641) K
    READ(A,644) (TESTNAME(J),J=10,12)
    IF(TESTNAME(11).EQ.SPACE) TESTNAME(11)=ZERO
    IF(TESTNAME(10).EQ.SPACE) TESTNAME(10)=ZERO
    WRITE(4,642) TESTN,OUT(K,ISN)
    OPEN(3,FILE=TESTN,form='BINARY')
    DO 654 J=1,9
        DO 653 I=1,7
            R=NIBON(I,J,K,ISN)
            WRITE(3) R
653          CONTINUE
654          CONTINUE
        CLOSE(3)
645      CONTINUE
    CLOSE(4)
    dummy = setvideomode( $DEFAULTMODE )
END

SUBROUTINE on(icol,irow)
INCLUDE 'FGRAPH.FD'
INTEGER*2          dummy

```

```

DOUBLE PRECISION      x1,y1,x2,y2
RECORD /videoconfig/ screen
COMMON                screen

    dummy = setcolor( 14)
    x2=icol
    y2=irow
    x1=x2-1.
    y1=y2-1.
    dummy = rectangle_w($GFILLINTERIOR,x1,y1,x2,y2)
    dummy = setcolor( 2 )

RETURN

FND

SUBROUTINE off(icol,irow)
INCLUDE  'FGRAPH.FD'
INTEGER*2          dummy
DOUBLE PRECISION    x1,y1,x2,y2
RECORD /videoconfig/ screen
COMMON              screen

    x2=icol
    y2=irow
    x1=x2-1.
    y1=y2-1.
    dummy = setcolor( 0)
    dummy = rectangle_w( $GFILLINTERIOR,x1,y1,x2,y2 )
    dummy = setcolor( 2)
    dummy = rectangle_w( $GBORDER, x1, y1, x2, y2 )

```

240

```
        dummy = setcolor( 2 )
RETURN
END
CC  GRID - This subroutine plots nib grid.
      SUBROUTINE grid()
      INCLUDE 'FGRAPH.FD'
      INTEGER*2          dummy,  i
      DOUBLE PRECISION    x,y
      RECORD /videoconfig/ screen
      RECORD /wxycoord/   wxy
      COMMON              screen
C      Plot the grid
      DO i = 1, 6
        dummy = setcolor( 2 )
        x=i
        CALL  moveto_w( x, 0.0, wxy )
        dummy = lineto_w( x, 9.0 )
      END DO
      DO i = 1, 8
        dummy = setcolor( 2 )
        y=i
        CALL  moveto_w(0.0, y, wxy )
        dummy = lineto_w(7.0, y )
      END DO
      RETURN
      END
```

APPENDIX C

BACKPROP

BACKPROP is used to develop no, one, and two hidden layer models. Before running the program several files should be created using a text editor: (1) STOP_ERR, (2) AUTOSAVE, (3) RATES, (4) HIDDEN1 (for one layer), (5) HIDDEN2 (for two layers), (6) PASSES, (7) a file containing a list of learn case files, and (8) a file containing a list of test case files.

The contents of all the above files are ASCII and can be built with a text editor. STOP_ERR contains a value, such as, 0.0001 meaning to stop after the error reaches this limit. AUTOSAVE contains a value, such as, 500 meaning to automatically save the model every 500 iterations. RATES contains two values, one per line, such as 0.6 and 0.9. The first value is the learning rate and the second is the momentum factor. HIDDEN1 contains a value, such as, 16 to declare 16 neurons for the hidden layer (0 means calculate based on input and output neuron count). HIDDEN2 has two entries, one per line, such as, 16 and 8, to indicate the number of neurons in each hidden layer. PASSES contains the maximum allowed number of iterations on a model before termination. If STOP_ERR contains 0.001 and PASSES

contains 50000, the program terminates at an error level of 0.001 (0.1 percent) if this error level is reached before 50000 iterations or at 50000 iterations, otherwise.

The learn case file and test case file are identical in format. The first line has the number of output classifications (neurons) in fields 1-6, free field, such as, 4. The next lines each have two entries: (1) a file name in columns 1-12 and (2) the output neuron to be active (have a one rather than 0 output) in columns 11-12.

The weights and biases are initialized to small random numbers. However if option 4 (do 0, 1, and 2 hidden layers) is selected then each network builds on the previous one's weights and biases, this can speed convergence dramatically. For example, if HIDDEN1 contains 7 and HIDDEN2 contains 16 and 7, then the output weights from the no hidden layer model will be the starting point for the one layer model and the output and hidden layer weights from the one layer model will be the starting point for the two layer model.

C.1 BACKPROP Example Run

If the learning cases are in INLEARN and the test cases are in INTEST, the one hidden level model would be run by typing

BACKPROP,

followed by one for the one hidden layer model

1,

followed by a one for initial running (new model)

1,
 followed by a learning case file name of
 INLEARN,
 followed by s or r for sequential or random learning case
 order
 r,
 and activated by a test case file name of
 INTEST.

An example run is shown in Figure 73.

```

C:\NEURAL> backprop
Enter number of hidden layers 0, 1, or 2
Enter 4 to do all 3 models)
4
Option 2 is not valid when doing all models
Option:
1 for new learning
2 to resume learning after interruption
3 for test cases only
Enter 1, 2, or 3.
1
File containing list of learning case files
savannah.lrn
ORDER OF CASES DURING LEARNING
ENTER 2 FOR RANDOM OR
3 FOR SEQUENTIAL
3
File containing list of test case files
gulfport.tst

```

Figure 73. Example of BACKPROP

The program may be stopped at any time by typing escape and restarted later by entering 2 rather than 1 in answer to the first question. When new test cases are be applied to an old model the 3 option is entered. The models are saved in NET.0H and MOMENTUM.0H, in NET.1H and MOMENTUM.1H, and in NET.2H and MOMENTUM.2H, respectively. If a new set of learn cases is to be input, these files should be renamed or saved to floppy as they will be overwritten.

C.2 Listing of BACKPROP

C BACKPROP WITH 2 HIDDEN LAYERS (4 LAYER NETWORK INCL IN)

C

```

-
|I|
|N| _____|_____|_____|_____|_____|_____|
|P| /          |Hidden|      |HIDDEN|      |Out   |
|U|<  _____|Layer1|_____|Layer2|_____|Layer |---OUT
|T| ' /          |N(1)  |- /   |N(1)  |- /   |N(1)  | (1)
=  '/          |_____| '/   |_____| '/   |_____|
|L| '/'          |_____| '/'   |_____| '/'   |_____|
|A| / '          |_____| / '   |_____| / '   |_____|
|Y|<  '---| N(NUM-| '---| N(NUM-| '---| N(NUM-|
|E| \_____|HIDDEN1|      |HIDDEN2|      |OUT) |---OUT
|R|          |      )|_____|      )|_____|      | (NUMOUT)
-          |_____|_____|_____|_____|_____|
WEIGHT_          WEIGHT_          WEIGHT_
HIDDEN1          HIDDEN2          OUTLAYER
(INLAYER,          (NHIDDEN1,          (NHIDDEN2,
NHIDDEN1)          NHIDDEN2)          NOUTLAYER)

```

C

C INLAYER={1..NUMIN}

C NHIDDEN1={1..NUMHIDDEN1}

C NHIDDEN2={1..NUMHIDDEN2}

C NOUTLAYER={1..NUMOUT}

C

C Additionally there is a bias input to each neuron

C BIAS_HIDDEN1(NHIDDEN1) for hidden layer 1

C BIAS_HIDDEN2(NHIDDEN2) for hidden layer 2

C BIAS_OUTLAYER(NOUTLAYER) for out layer

C

C BACKPROP WITH ONE HIDDEN LAYER (3 LAYER NETWORK w/ IN)

C

C

```

C      -
C      |I| _____|_____
C      |N| _____|_____|_____
C      |P| / _____| Hidden| _____| Out
C IN(1)-----|U|< _____| Layer | _____| Layer |---OUT(1)
C      |T| ' / _____| N(1) | -- / _____| N(1)
C      = ' / _____| ' / _____|
C      |L| /' _____| /' _____|
C      |A| / ' _____| / ' _____|
C IN(NUMIN)-|Y|< '---| N(NUM-| '---| N(NUM-|
C      |E| \_____|HIDDEN)| _____| OUT) |---OUT(N)
C      |R| _____|_____|_____
C      - _____|_____|_____

```

C

WEIGHT_

WEIGHT_

C

HIDDEN

OUTLAYER

C

(INLAYER,

(NHIDDEN,

C

NHIDDEN)

NOUTLAYER)

C

C INLAYER={1..NUMIN}

```

C  NHIDDEN={1..NUMHIDDEN}
C  NOUTLAYER={1..NUMOUT}
C
C  Additionally there is a bias input to each neuron
C  BIAS_HIDDEN(NHIDDEN) for hidden layer
C  BIAS_OUTLAYER(NOUTLAYER) for out layer
C
C  NO HIDDEN LAYER (LMS) (2 LAYER NETWORK INCL IN)
C
C      -
C      |I| _____
C      |N| _____|      |
C      |P| /          | out  |
C  IN(1)-----|U|<  _____| Layer |---OUT(1)
C      |T| ' /          | N(1)  |
C      =  '/          | _____|
C      |L| /'          | _____|
C      |A|      '      |      |
C  IN(NUMIN)-|Y|<  '----| N(NUM-|
C      |E| \_____ | OUT) |---OUT(NUMOUT)
C      |R|          |      |
C      -          | _____|
C
C      WEIGHT_
C      OUTLAYER
C      (INLAYER,
C      NOUTLAYER)

```

248

C

C INLAYER={1..NUMIN}

C NOUTLAYER={1..NUMOUT}

C

C Additionally there is a bias input to each neuron

C BIAS_OUTLAYER(NOUTLAYER) for out layer

C

C

C LEARNING_RATE is multiplied by the newly calculated

C Delta_Weight

C MOMENTUM is multiplied by the previous instigated

Delta_Weight

C both range from {0..1}

C

C Delta value means the change from the previous value

C CASE_IN is the learning set inputs

C CASE_OUT is the desired output

C ERROR is the difference between the desired and obtained

C NET is the sum of the inputs by their respective weights

C SIGMOID is the nonlinear function

C

C Note: The following are maximum values

C The actual may be less than these numbers

C Max number of input neurons

PARAMETER (NUMIN1=64)

C Max number of first hidden layer neurons

```

PARAMETER (NUMHIDDEN11=100)
C    Max number of second hidden layer neurons
PARAMETER (NUMHIDDEN21=100)
C    Max number of output layer neurons
PARAMETER (NUMOUT1=10)
C    Max number of learning or test cases
PARAMETER (NUMCASES1=500)
C    Buffer sizes
PARAMETER (NUMWEIGHTHIDDEN1=NUMIN1*NUMHIDDEN11)
PARAMETER (NUMWEIGHTHIDDEN2=NUMHIDDEN11*NUMHIDDEN21)
PARAMETER (NUMWEIGHTOUT=NUMHIDDEN21*NUMOUT1)
PARAMETER (NUMCASIN1=NUMIN1*NUMCASES1)
PARAMETER (NUMCASOUT1=NUMOUT1*NUMCASES1)
C    The network is saved in case of power outage every
C    auto_save1 iterations; may be restarted when resumed
PARAMETER (auto_save1=50)
C
common/i7/ weight_hidden1,delta_weight_hidden1
common/i8/ weight_hidden2,delta_weight_hidden2
common/i9/ weight_outlayer,delta_weight_outlayer
CHARACTER*12 INLEARN,INTEST,INFILE
C*****
C    The MSDOS version allows an escape and save option
C    This is necessary since PC's are slower than Cray's
C    The following line is needed for the MSDOS version
C    Remove the c for comment for MSDOS

```

```
c      Integer*1 dummy_read,keydep,escape
```

```
C*****
```

```

character*1 iis,nns,iir,nnr,case_order
REAL OUT_HIDDEN1(NUMHIDDEN1),
1 OUT_HIDDEN2(NUMHIDDEN2),
1 OUT(NUMOUT1),WEIGHT_HIDDEN1(NUMWEIGHTHIDDEN1),
1 WEIGHT_HIDDEN2(NUMWEIGHTHIDDEN2),
1 WEIGHT_OUTLAYER(NUMWEIGHTOUT),
2 BIAS_HIDDEN1(NUMHIDDEN1),BIAS_OUTLAYER(NUMOUT1),
2 BIAS_HIDDEN2(NUMHIDDEN2),
3 ERROR(NUMOUT1),DELTA_OUT(NUMOUT1),
4 DELTA_BIAS_OUTLAYER(NUMOUT1),
4 DELTA_WEIGHT_OUTLAYER(NUMWEIGHTOUT),
5 DELTA_WEIGHT_HIDDEN1(NUMWEIGHTHIDDEN1),
5 DELTA_WEIGHT_HIDDEN2(NUMWEIGHTHIDDEN2),
6 DELTA_BIAS_HIDDEN1(NUMHIDDEN1),
6 DELTA_HIDDEN1(NUMHIDDEN1),
7 DELTA_BIAS_HIDDEN2(NUMHIDDEN2),
7 DELTA_HIDDEN2(NUMHIDDEN2),
8 CASE_IN(NUMCASIN1),CASE_OUT(NUMCASOUT1)
REAL LEARNING_RATE,MOMENTUM
integer iin(numcases1),auto_save
INTEGER*2 ORDER(NUMCASES1)
data iis/'S'/,iir/'R'/,nns/'s'/,nnr/'r'/
DATA NUMIN/NUMIN1/,NUMHIDDEN1/numhidden1/,
1 NUMOUT/NUMOUT1/,

```

```

1 ERROR_MAX/.001/,NUMCASES/NUMCASES1/,
1 NUMHIDDEN2/numhidden21/
C*****
C      The following line is needed for the MSDOS version
C      Remove c for comment for MSDOS
c      data escape/27/
C*****
      data auto_save/auto_save1/,max_iterations/1000000/
      DATA LEARNING_RATE/.6/,MOMENTUM/.9/
C      Number of hidden layers is entered from keyboard
721 write(6,724)
724 format(' Enter number of hidden layers 0, 1, or 2'/
1 '(Enter 4 to do all 3 models)')
      read(5,102,err=721,end=999) numlayers
      numlay=numlayers
      if(numlayers.gt.4.or.numlayers.lt.0) go to 721
      if(numlayers.eq.4) then
          numlayers=0
c      write(6,725)
c725      format(' Option 2 is not valid when doing all'
c      1 ' models')
      endif
C      New, Resume, or Test is entered from keyboard
100 write(6,101)
101 format(' Option: '/
1 ' 1 for new learning'/

```

252

```
2  ' 2 to resume learning after interruption'/
3  ' 3 for test cases only'/
4  ' Enter 1, 2, or 3.')
    read(5,102) iopt
102  format(i6)
    if(iopt.lt.1.or.iopt.gt.3) go to 100
    if(iopt.eq.1.or.iopt.eq.2) then
C    file name for learning cases -> INLEARN
        write(6,200)
200    format(' File containing list of learning case'
1    ' files')
        read(5,201) INLEARN
C    Use order of signals in INLEARN for learning passes or
C    randomly determine a new order for each pass
292    WRITE(6,290)
290    FORMAT(' ORDER OF CASES DURING LEARNING'/
1    ' ENTER R FOR RANDOM OR'/
2    '          S FOR SEQUENTIAL')
        READ(5,291) CASE_ORDER
291    format(A1)
        IF(CASE_ORDER.EQ.NNS) CASE_ORDER=IIS
        IF(CASE_ORDER.EQ.NNR) CASE_ORDER=IIR
        IF(CASE_ORDER.NE.IIS.AND.CASE_ORDER.NE.IIR) GO TO
292
        endif
C    file name for test cases -> INTEST
```

```
write(6,205)
205  format(' File containing list of test case files')
      read(5,201) INTEST
201  format(A12)
C    Initialize weights and biases to small random numbers
C    If option 4 is taken each network builds on the
C    previous one's weights and biases, this can speed
C    convergence dramatically.
C    For example, if HIDDEN1 contains 7 and HIDDEN2 16 & 7
C    then the output weights from the no hidden layer model
C    will be the starting point for the one layer model and
C    the output and hidden layer weights from the one layer
C    model will be the starting point for the two layer
C    model.
111  numhidden1=0
      numhidden2=0
C    PASSES contains the maximum number of iterations
1    OPEN(3,FILE='PASSES',STATUS='OLD',ERR=564)
      READ(3,965,ERR=563,END=563) NUMH
      max_iterations=NUMH
563  CLOSE(3)
C    STOP_ERR contains the target error level
564  OPEN(3,FILE='STOP_ERR',STATUS='OLD',ERR=664)
      READ(3,865,ERR=663,END=663) RATE
      ERROR_MAX=RATE
663  CLOSE(3)
```


254

C AUTOSAVE contains the number of iterations betw saves

664 OPEN(3,FILE='AUTOSAVE',STATUS='OLD',ERR=764)

 READ(3,965,ERR=763,END=763) NUMH

 auto_save=NUMH

763 CLOSE(3)

C RATES contains the learning rate and momentum

764 OPEN(3,FILE='RATES',STATUS='OLD',ERR=864)

 READ(3,865,ERR=863,END=863) RATE

865 FORMAT(F20.0)

 LEARNING_RATE=RATE

 READ(3,865,ERR=863,END=863) RATE

 MOMENTUM=RATE

863 CLOSE(3)

C HIDDEN2 contains the number of neurons in the 1st &

C 2nd hidden layers

864 if(numlayers.eq.2) then

 OPEN(3,FILE='HIDDEN2',STATUS='OLD',ERR=964)

 READ(3,965,ERR=963,END=963) NUMH

965 FORMAT(I6)

 NUMHIDDEN1=NUMH

 READ(3,965,ERR=963,END=963) NUMH

 NUMHIDDEN2=NUMH

963 CLOSE(3)

C HIDDEN1 contains the number of neurons in hidden layer

else if(numlayers.eq.1) then

 OPEN(3,FILE='HIDDEN1',STATUS='OLD',ERR=964)

```

      READ(3,965,ERR=962,END=962) NUMH
      NUMHIDDEN1=NUMH
962      CLOSE(3)
      endif
964      isave=0
      if(iopt.eq.3) isave=1
      ITERATIONS=0
      INDEX=0
C      Initialize number of hidden layers
      if (numhidden1.eq.0) then
          rn=numin+numout
          numhidden1=sqrt(rn)
          numhidden1=numhidden1*2
          numhidden2=numhidden1
      endif
c      Initialize biases and weights to small random numbers
      if(numlay.eq.4) then
          if(numlayers.eq.0)
1          CALL INITIALIZE_NETWORK_2(WEIGHT_HIDDEN1,
1          WEIGHT_OUTLAYER,
1          DELTA_WEIGHT_HIDDEN1,DELTA_WEIGHT_OUTLAYER,
2          NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,
2          BIAS_OUTLAYER,
3          DELTA_BIAS_HIDDEN1,DELTA_BIAS_OUTLAYER,
4          WEIGHT_HIDDEN2,DELTA_WEIGHT_HIDDEN2,NUMHIDDEN2,
5          BIAS_HIDDEN2,DELTA_BIAS_HIDDEN2)

```

```

else
    if(numlayers.eq.1) then
        CALL INITIALIZE_NETWORK_1(WEIGHT_HIDDEN1,
1      WEIGHT_OUTLAYER,
1      DELTA_WEIGHT_HIDDEN1, DELTA_WEIGHT_OUTLAYER,
2      NUMIN, NUMHIDDEN1, NUMOUT, BIAS_HIDDEN1,
2      BIAS_OUTLAYER,
3      DELTA_BIAS_HIDDEN1, DELTA_BIAS_OUTLAYER)
    else if(numlayers.eq.2) then
        CALL INITIALIZE_NETWORK_2(WEIGHT_HIDDEN1,
1      WEIGHT_OUTLAYER,
1      DELTA_WEIGHT_HIDDEN1, DELTA_WEIGHT_OUTLAYER,
2      NUMIN, NUMHIDDEN1, NUMOUT, BIAS_HIDDEN1,
2      BIAS_OUTLAYER,
3      DELTA_BIAS_HIDDEN1, DELTA_BIAS_OUTLAYER,
4      WEIGHT_HIDDEN2, DELTA_WEIGHT_HIDDEN2, NUMHIDDEN2,
5      BIAS_HIDDEN2, DELTA_BIAS_HIDDEN2)
    else
        call INITIALIZE_NETWORK_0(WEIGHT_OUTLAYER,
1      DELTA_WEIGHT_OUTLAYER, NUMIN, NUMOUT,
2      BIAS_OUTLAYER, DELTA_BIAS_OUTLAYER)
    endif
endif

if(numlayers.eq.2) write(6,254) numhidden1,numhidden2
if(numlayers.eq.1) write(6,9254) numhidden1
254 format(' numhidden1=',i3,'          numhidden2=',i3)

```

```
9254  format(' numhidden1=',i3)
      if(iopt.ne.3) then
          inumcases=1
          numin=0
c      Get list of learning files
          open(3,file=INLEARN)
C      number of output neurons -> numout
          read(3,216) numout
          write(6,266) numout
266    format(' numout=',i3)
216    format(i4)
C      Read in name of signal file and its output neuron
250    read(3,217,end=257) infile,iin(inumcases)
217    format(a12,i2)
          numcases=inumcases
          inumcases=inumcases+1
          inumin=1
          open(4,file=INFILE)
C      Read in signal
251    jjcase=(numcases-1)*numin+inumin
C      count number of cases while reading to EOF
          read(4,252,end=253) case_in(jjcase)
252    format(f20.0)
          if(numcases.eq.1) then
              numin=inumin
          endif
```

258

```
if(numcases.eq.2.and.inumin.eq.1) then
```

```
write(6,256) numin
```

256 format(' numin=',i3)

```
endif
```

```
inumin=inumin+1
```

```
go to 251
```

253 close(4)

```
go to 250
```

257 close(3)

```
write(6,259) numcases
```

259 format(' numcases=',i3)

C Set output neuron desired outputs to 1. if desired

C active; 0. if not

```
do 104 j=1,numcases
```

```
ORDER(J)=J
```

```
do 103 jj=1,numout
```

```
jjj=(j-1)*numout+jj
```

```
if(iin(j).eq.jj) then
```

```
case_out(jjj)=1.
```

```
else
```

```
case_out(jjj)=0.
```

```
endif
```

103 continue

104 continue

```
endif
```

C

```

C      Two Hidden Layers
C
      if(numlayers.eq.2) then
        if(iopt.eq.2.or.iopt.eq.3) then
C      Read in iterations, weights, biases if continuing
          OPEN(4,FILE='NET.2H')
          READ(4,211) ITERATIONS
211      FORMAT(11X,I7)
          READ(4,218) NUMIN
218      format(i4)
219      FORMAT(A1)
          READ(4,218) NUMHIDDEN1
          DO 2231 NHIDDEN1=1,NUMHIDDEN1
            NUMROW=(NHIDDEN1-1)*NUMIN
            DO 2231 INLAYER=1,NUMIN
              INDEX_WEIGHT=NUMROW+INLAYER
              READ(4,20) WEIGHT_HIDDEN1(INDEX_WEIGHT)
2231      CONTINUE
          READ(4,218) NUMHIDDEN2
          DO 2232 NHIDDEN2=1,NUMHIDDEN2
            NUMROW=(NHIDDEN2-1)*NUMHIDDEN1
            DO 2232 NHIDDEN1=1,NUMHIDDEN1
              INDEX_WEIGHT=NUMROW+NHIDDEN1
              READ(4,20) WEIGHT_HIDDEN2(INDEX_WEIGHT)
2232      CONTINUE
          READ(4,218) NUMOUT

```

```
DO 2241 NOUTLAYER=1,NUMOUT
    NUMROW=(NOUTLAYER-1)*NUMHIDDEN2
    DO 2241 NHIDDEN2=1,NUMHIDDEN2
        INDEX_WEIGHT=NUMROW+NHIDDEN2
        READ(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)
2241    CONTINUE
    READ(4,219) dummy_read
    READ(4,20) (ERROR(I),I=1,NUMOUT)
    READ(4,219) dummy_read
    READ(4,20) (BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
    READ(4,219) dummy_read
    READ(4,20) (BIAS_HIDDEN2(J),J=1,NUMHIDDEN2)
    READ(4,219) dummy_read
    READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)
    CLOSE(4)
C    Read in delta_weights, delta_biases if continuing
    OPEN(4,FILE='MOMENTUM.2H')
    READ(4,219) dummy_read
    DO 2431 NHIDDEN1=1,NUMHIDDEN1
        NUMROW=(NHIDDEN1-1)*NUMIN
        DO 2431 INLAYER=1,NUMIN
            INDEX_WEIGHT=NUMROW+INLAYER
            READ(4,20) DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
2431    CONTINUE
    READ(4,219) dummy_read
    DO 2433 NHIDDEN2=1,NUMHIDDEN2
```

```

      NUMROW=(NHIDDEN2-1)*NUMHIDDEN1
      DO 2433 NHIDDEN1=1,NUMHIDDEN1
        INDEX_WEIGHT=NUMROW+NHIDDEN1
        READ(4,20) DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)
2433    CONTINUE
      READ(4,219) dummy_read
      DO 2441 NOUTLAYER=1,NUMOUT
        NUMROW=(NOUTLAYER-1)*NUMHIDDEN2
        DO 2441 NHIDDEN2=1,NUMHIDDEN2
          INDEX_WEIGHT=NUMROW+NHIDDEN2
          READ(4,20) DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
2441    CONTINUE
      READ(4,219) dummy_read
      READ(4,20) (DELTA_BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
      READ(4,219) dummy_read
      READ(4,20) (DELTA_BIAS_HIDDEN2(J),J=1,NUMHIDDEN2)
      READ(4,219) dummy_read
      READ(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)
      CLOSE(4)
    endif
2071  if(iopt.ne.3) then
2001    ERR_MAX=0.0
C      Scramble order of input of learning cases
      IF(CASE_ORDER.EQ.IIR) CALL
1      RANDOMIZE(ORDER,NUMCASES)
C      Do one major pass through network of all learn cases

```



```

DO 2004 i=1,numcases
  INCASE=(ORDER(I)-1)*numin+1
  ioutcase=(ORDER(i)-1)*numout+1
  CALL FORWARD_PASS_2(CASE_IN(INCASE),
1    OUT_HIDDEN1,OUT,
1    WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,
2    NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,
2    BIAS_OUTLAYER,
3    NUMHIDDEN2,BIAS_HIDDEN2,WEIGHT_HIDDEN2)
  CALL BACKWARD_PASS_2
1    (CASE_IN(INCASE),OUT_HIDDEN1,OUT,
1    WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,
2    NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,
2    BIAS_OUTLAYER,
3    CASE_OUT(IOUTCASE),ERROR,DELTA_OUT,
3    DELTA_BIAS_OUTLAYER,
4    DELTA_WEIGHT_OUTLAYER,DELTA_HIDDEN1,
4    DELTA_BIAS_HIDDEN1,
5    DELTA_WEIGHT_HIDDEN1,LEARNING_RATE,MOMENTUM,
6    WEIGHT_HIDDEN2,NUMHIDDEN2,
6    BIAS_HIDDEN2,DELTA_HIDDEN2,
7    DELTA_BIAS_HIDDEN2,DELTA_WEIGHT_HIDDEN2)
  ERR=0.
C    Test for error limit reached termination
      DO 2010 J=1,NUMOUT
        ERR=ERR+ERROR(J)*ERROR(J)

```

```

2010      CONTINUE

          IF(ERR.GT.ERR_MAX) ERR_MAX=ERR

2004      CONTINUE

          ITERATIONS=ITERATIONS+1

          INDEX=INDEX+1

C        Display iterations and error level every 100'th pass
          IF(INDEX.EQ.100) THEN

              INDEX=0

              WRITE(6,11) ITERATIONS,ERR_MAX

          ENDIF

11        FORMAT(' ITERATION:',I7,'          ERROR:',F10.5)

          isave=isave +1

C        Is it time to auto_save?

          if(isave.eq.auto_save) then

              isave=0

          endif

          if(isave.eq.0) goto 2014

C        Has number of iterations exceeded limit?

          if(iterations.gt.max_iterations) go to 2014

C*****

C        The following 3 lines are necessary for MSDOS

c        call getkey(keydep,iierr)

c        if(iierr.eq.0) go to 2012

c        if(keydep.eq.escape) go to 2014

C*****

2012      IF(ERR_MAX.GT.ERROR_MAX) GO TO 2001

```

264

C Write current network iterations, weights, biases, etc.

2014 OPEN(4, FILE='NET.2H')

WRITE(4, 11) ITERATIONS, ERR_MAX

WRITE(4, 621) NUMIN

621 FORMAT(I4, ' INPUTS')

WRITE(4, 19) NUMHIDDEN1

19 FORMAT(I4, ' HIDDEN WEIGHTS FIRST LAYER')

DO 2031 NHIDDEN1=1, NUMHIDDEN1

NUMROW=(NHIDDEN1-1)*NUMIN

DO 2031 INLAYER=1, NUMIN

INDEX_WEIGHT=NUMROW+INLAYER

WRITE(4, 20) WEIGHT_HIDDEN1(INDEX_WEIGHT)

2031 CONTINUE

WRITE(4, 719) NUMHIDDEN2

719 FORMAT(I4, ' HIDDEN WEIGHTS SECOND LAYER')

DO 2032 NHIDDEN2=1, NUMHIDDEN2

NUMROW=(NHIDDEN2-1)*NUMHIDDEN1

DO 2032 NHIDDEN1=1, NUMHIDDEN1

INDEX_WEIGHT=NUMROW+NHIDDEN1

WRITE(4, 20) WEIGHT_HIDDEN2(INDEX_WEIGHT)

2032 CONTINUE

20 FORMAT(F16.10)

WRITE(4, 21) NUMOUT

21 FORMAT(I4, ' OUTPUT WEIGHTS')

DO 2041 NOUTLAYER=1, NUMOUT

NUMROW=(NOUTLAYER-1)*NUMHIDDEN2

```
DO 2041 NHIDDEN2=1, NUMHIDDEN2
    INDEX_WEIGHT=NUMROW+NHIDDEN2
    WRITE(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)
2041 CONTINUE
    WRITE(4,22)
22    FORMAT(' ERROR')
    WRITE(4,20) (ERROR(I), I=1, NUMOUT)
    WRITE(4,23)
23    FORMAT(' HIDDEN BIAS FIRST LAYER')
    WRITE(4,20) (BIAS_HIDDEN1(J), J=1, NUMHIDDEN1)
    WRITE(4,723)
723    FORMAT(' HIDDEN BIAS SECOND LAYER')
    WRITE(4,20) (BIAS_HIDDEN2(J), J=1, NUMHIDDEN2)
    WRITE(4,24)
24    FORMAT(' OUT BIAS')
    WRITE(4,20) (BIAS_OUTLAYER(J), J=1, NUMOUT)
    CLOSE(4)
C    Write out success rate on learning cases
    OPEN(4, FILE='LEARNCAS.2H')
    WRITE(4,25)
25    FORMAT(' LEARN CASE ERRORS')
    DO 2040 I=1, NUMCASES
        WRITE(4,26) I
26    FORMAT(' CASE ', I2)
        incase=(i-1)*numin+1
        ioutcase=(i-1)*numout
```

```

        CALL FORWARD_PASS_2(CASE_IN(INCASE),
1      OUT_HIDDEN1,OUT,
1      WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,
2      NUMIN,NUMHIDDEN1,NUMOUT,
2      BIAS_HIDDEN1,BIAS_OUTLAYER,
3      NUMHIDDEN2,BIAS_HIDDEN2,WEIGHT_HIDDEN2)
        WRITE(4,27) (case_out(ioutcase+j),OUT(J),
1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)
27      FORMAT(3F10.6)
2040    CONTINUE
        CLOSE(4)
C      Write out delta_weights & delta_biases
        OPEN(4,FILE='MOMENTUM.2H')
        WRITE(4,459)
459    FORMAT(' Delta_Weight_Hidden Values First Layer:')
        DO 2461 NHIDDEN1=1,NUMHIDDEN1
            NUMROW=(NHIDDEN1-1)*NUMIN
            DO 2461 INLAYER=1,NUMIN
                INDEX_WEIGHT=NUMROW+INLAYER
                WRITE(4,20)
1      DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
2461    CONTINUE
        WRITE(4,479)
479    FORMAT(' Delta_Weight_Hidden Values Second Layer:')
        DO 2761 NHIDDEN2=1,NUMHIDDEN2
            NUMROW=(NHIDDEN2-1)*NUMHIDDEN1

```

```

DO 2761 NHIDDEN1=1,NUMHIDDEN1
    INDEX_WEIGHT=NUMROW+NHIDDEN1
    WRITE(4,20)
1        DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)
2761    CONTINUE
        WRITE(4,467)
467    FORMAT(' Delta_Weight_Outlayer Values:')
        DO 2471 NOUTLAYER=1,NUMOUT
            NUMROW=(NOUTLAYER-1)*NUMHIDDEN2
            DO 2471 NHIDDEN2=1,NUMHIDDEN2
                INDEX_WEIGHT=NUMROW+NHIDDEN2
                WRITE(4,20)
1            DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
2471    CONTINUE
        WRITE(4,468)
468    FORMAT(' Delta_Bias_Hidden Values First Layer:')
        WRITE(4,20) (DELTA_BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
        WRITE(4,418)
418    FORMAT(' Delta_Bias_Hidden Values Second Layer:')
        WRITE(4,20) (DELTA_BIAS_HIDDEN2(J),J=1,NUMHIDDEN2)
        WRITE(4,469)
469    FORMAT(' Delta_Bias_Outlayer Values:')
        WRITE(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)
        CLOSE(4)
endif
C    If not auto_save then quit

```

268

```
      if(isave.eq.0) go to 2071

C
C   1 Hidden Layer
C

      else if (numlayers.eq.1) then
C   Read in old network weights & biases on resuming
      if(iopt.eq.2.or.iopt.eq.3) then
        OPEN(4,FILE='NET.1H')
        READ(4,211) ITERATIONS
        READ(4,218) NUMIN
        READ(4,218) NUMHIDDEN1
        DO 1231 NHIDDEN=1,NUMHIDDEN1
          NUMROW=(NHIDDEN-1)*NUMIN
          DO 1231 INLAYER=1,NUMIN
            INDEX_WEIGHT=NUMROW+INLAYER
            read(4,20) WEIGHT_HIDDEN1(INDEX_WEIGHT)
1231    CONTINUE
          READ(4,218) NUMOUT
          DO 1241 NOUTLAYER=1,NUMOUT
            NUMROW=(NOUTLAYER-1)*NUMHIDDEN1
            DO 1241 NHIDDEN=1,NUMHIDDEN1
              INDEX_WEIGHT=NUMROW+NHIDDEN
              READ(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)
1241    CONTINUE
            READ(4,219) dummy_read
            READ(4,20) (ERROR(I),I=1,NUMOUT)
```

```

      READ(4,219) dummy_read
      READ(4,20) (BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
      READ(4,219) dummy_read
      READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)
      CLOSE(4)
      OPEN(4,FILE='MOMENTUM.1H')
C      Read in old delta_biases & delta_weights on resume
      READ(4,219) dummy_read
      DO 1431 NHIDDEN=1,NUMHIDDEN1
          NUMROW=(NHIDDEN-1)*NUMIN
          DO 1431 INLAYER=1,NUMIN
              INDEX_WEIGHT=NUMROW+INLAYER
              READ(4,20) DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
1431      CONTINUE
      READ(4,219) dummy_read
      DO 1441 NOUTLAYER=1,NUMOUT
          NUMROW=(NOUTLAYER-1)*NUMHIDDEN1
          DO 1441 NHIDDEN=1,NUMHIDDEN1
              INDEX_WEIGHT=NUMROW+NHIDDEN
              READ(4,20)
1              DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
1441      CONTINUE
      READ(4,219) dummy_read
      READ(4,20) (DELTA_BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
      READ(4,219) dummy_read
      READ(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)

```



```
        CLOSE(4)

    endif

1071 if(iopt.ne.3) then
1001     ERR_MAX=0.0
C       Scramble order of inputs for next pass
        IF(CASE_ORDER.EQ.IIR) CALL
1       RANDOMIZE(ORDER,NUMCASES)
C       Do one overall pass of all learning cases
        DO 1004 i=1,numcases
            INCASE=(ORDER(I)-1)*numin+1
            ioutcase=(ORDER(i)-1)*numout+1
            CALL FORWARD_PASS_1(CASE_IN(INCASE),
1            OUT_HIDDEN1,OUT,
1            WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,
2            NUMIN,NUMHIDDEN1,NUMOUT,
2            BIAS_HIDDEN1,BIAS_OUTLAYER)
            CALL BACKWARD_PASS_1(CASE_IN(INCASE),
1            OUT_HIDDEN1,OUT,
1            WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,
2            NUMIN,NUMHIDDEN1,NUMOUT,
2            BIAS_HIDDEN1,BIAS_OUTLAYER,
3            CASE_OUT(IOUTCASE),ERROR,
3            DELTA_OUT,DELTA_BIAS_OUTLAYER,
4            DELTA_WEIGHT_OUTLAYER,DELTA_HIDDEN1,
4            DELTA_BIAS_HIDDEN1,
5            DELTA_WEIGHT_HIDDEN1,LEARNING_RATE,MOMENTUM)
```

```

      ERR=0.

C      Check for error limit termination?

      DO 1010 J=1,NUMOUT

      ERR=ERR+ERROR(J)*ERROR(J)

1010      CONTINUE

      IF(ERR.GT.ERR_MAX) ERR_MAX=ERR

1004      CONTINUE

      ITERATIONS=ITERATIONS+1

      INDEX=INDEX+1

      IF(INDEX.EQ.100) THEN

      INDEX=0

      WRITE(6,11) ITERATIONS,ERR_MAX

      ENDIF

      isave=isave +1

      if(isave.eq.auto_save) then

      isave=0

      endif

      if(isave.eq.0) goto 1014

C      Check for maximum allowed iterations termination?

      if(iterations.gt.max_iterations) go to 1014

C*****

C      The following 3 lines are necessary for MSDOS

c      call getkey(keydep,iierr)

c      if(iierr.eq.0) go to 1012

c      if(keydep.eq.escape) go to 1014

C*****

```

272

1012 IF(ERR_MAX.GT.ERROR_MAX) GO TO 1001

C Save network weights and biases

1014 OPEN(4,FILE='NET.1H')

WRITE(4,11) ITERATIONS,ERR_MAX

WRITE(4,621) NUMIN

WRITE(4,19) NUMHIDDEN1

DO 1031 NHIDDEN=1,NUMHIDDEN1

NUMROW=(NHIDDEN-1)*NUMIN

DO 1031 INLAYER=1,NUMIN

INDEX_WEIGHT=NUMROW+INLAYER

WRITE(4,20) WEIGHT_HIDDEN1(INDEX_WEIGHT)

1031 CONTINUE

WRITE(4,21) NUMOUT

DO 1041 NOUTLAYER=1,NUMOUT

NUMROW=(NOUTLAYER-1)*NUMHIDDEN1

DO 1041 NHIDDEN=1,NUMHIDDEN1

INDEX_WEIGHT=NUMROW+NHIDDEN

WRITE(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)

1041 CONTINUE

WRITE(4,22)

WRITE(4,20) (ERROR(I),I=1,NUMOUT)

WRITE(4,23)

WRITE(4,20) (BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)

WRITE(4,24)

WRITE(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)

CLOSE(4)

```

OPEN(4,FILE='LEARNCAS.1H')
WRITE(4,25)
DO 1040 I=1,NUMCASES
    WRITE(4,26) I
    incase=(i-1)*numin+1
    ioutcase=(i-1)*numout
    CALL FORWARD_PASS_1(CASE_IN(incase),
1      OUT_HIDDEN1,OUT,
1      WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,
2      NUMIN,NUMHIDDEN1,NUMOUT,
2      BIAS_HIDDEN1,BIAS_OUTLAYER)
    WRITE(4,27) (case_out(ioutcase+j),OUT(J),
1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)
1040 CONTINUE
    CLOSE(4)

C    Save network delta_weights and delta_biases
    OPEN(4,FILE='MOMENTUM.1H')
    WRITE(4,459)
    DO 1461 NHIDDEN=1,NUMHIDDEN1
        NUMROW=(NHIDDEN-1)*NUMIN
        DO 1461 INLAYER=1,NUMIN
            INDEX_WEIGHT=NUMROW+INLAYER
            WRITE(4,20)
1          DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
1461 CONTINUE
        WRITE(4,467)

```

```

DO 1471 NOUTLAYER=1,NUMOUT
    NUMROW=(NOUTLAYER-1)*NUMHIDDEN1
    DO 1471 NHIDDEN=1,NUMHIDDEN1
        INDEX_WEIGHT=NUMROW+NHIDDEN
        WRITE(4,20)
1          DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
1471    CONTINUE
        WRITE(4,468)
        WRITE(4,20) (DELTA_BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
        WRITE(4,469)
        WRITE(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)
        CLOSE(4)
endif
if(isave.eq.0) go to 1071

C
C    0 Hidden Layers
C
    else if (numlayers.eq.0) then
C    Read in iterations, weights, biases, etc. on resuming
    if(iopt.eq.2.or.iopt.eq.3) then
        OPEN(4,FILE='NET.OH')
        READ(4,211) ITERATIONS
        READ(4,218) NUMIN
        READ(4,218) NUMOUT
        DO 241 NOUTLAYER=1,NUMOUT
            NUMROW=(NOUTLAYER-1)*NUMIN

```

```

DO 241 INLAYER=1,NUMIN
    INDEX_WEIGHT=NUMROW+INLAYER
    READ(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)
241  CONTINUE
    READ(4,219) dummy_read
    READ(4,20) (ERROR(I),I=1,NUMOUT)
    READ(4,219) dummy_read
    READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)
    CLOSE(4)
C    Read in delta_weights, delta_biases on resuming
    OPEN(4,FILE='MOMENTUM.OH')
    READ(4,219) dummy_read
    DO 441 NOUTLAYER=1,NUMOUT
        NUMROW=(NOUTLAYER-1)*NUMIN
        DO 441 INLAYER=1,NUMIN
            INDEX_WEIGHT=NUMROW+INLAYER
            READ(4,20)
1            DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
441  CONTINUE
        READ(4,219) dummy_read
        READ(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)
        CLOSE(4)
    endif
71  if(iopt.ne.3) then
3001  ERR_MAX=0.0
C    Scramble order of input of learning cases

```

```

        IF(CASE_ORDER.EQ.IIR) CALL
1      RANDOMIZE(ORDER,NUMCASES)
C      Do one major pass of the learning cases
        DO 4 i=1,numcases
            INCASE=(ORDER(I)-1)*numin+1
            ioutcase=(ORDER(i)-1)*numout+1
            CALL FORWARD_PASS_0(CASE_IN(INCASE),OUT,
1          WEIGHT_OUTLAYER,
2          NUMIN,NUMOUT,BIAS_OUTLAYER)
            CALL BACKWARD_PASS_0(CASE_IN(INCASE),OUT,
1          WEIGHT_OUTLAYER,
2          NUMIN,NUMOUT,BIAS_OUTLAYER,
3          CASE_OUT(IOUTCASE),ERROR,
3          DELTA_OUT,DELTA_BIAS_OUTLAYER,
4          DELTA_WEIGHT_OUTLAYER,
5          LEARNING_RATE,MOMENTUM)
            ERR=0.
C      Test for error limit termination
        DO 10 J=1,NUMOUT
            ERR=ERR+ERROR(J)*ERROR(J)
10         CONTINUE
            IF(ERR.GT.ERR_MAX) ERR_MAX=ERR
4         CONTINUE
        ITERATIONS=ITERATIONS+1
        INDEX=INDEX+1
        IF(INDEX.EQ.100) THEN

```

```

        INDEX=0

        WRITE(6,11) ITERATIONS,ERR_MAX
    ENDIF

    isave=isave +1

    if(isave.eq.auto_save) then
        isave=0
    endif

    if(isave.eq.0) goto 14
C      Test for maximum iteration count exceeded termination
        if(iterations.gt.max_iterations) go to 14
C*****
C      The following 3 lines are necessary for MSDOS
C      call getkey(keydep,iierr)
C      if(iierr.eq.0) go to 12
C      if(keydep.eq.escape) go to 14
C*****
12      IF(ERR_MAX.GT.ERROR_MAX) GO TO 3001
C      Save the current weights and biases
14      OPEN(4,FILE='NET.0H')

        WRITE(4,11) ITERATIONS,ERR_MAX

        WRITE(4,621) NUMIN

        WRITE(4,21) NUMOUT

        DO 41 NOUTLAYER=1,NUMOUT
            NUMROW=(NOUTLAYER-1)*NUMIN
            DO 41 INLAYER=1,NUMIN
                INDEX_WEIGHT=NUMROW+INLAYER

```



```

WRITE(4,20) WEIGHT_OUTLAYER(INDEX_WEIGHT)

41  CONTINUE

WRITE(4,22)

WRITE(4,20) (ERROR(I),I=1,NUMOUT)

WRITE(4,24)

WRITE(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)

CLOSE(4)

OPEN(4,FILE='LEARNCAS.OH')

WRITE(4,25)

C    Save the success rate of the learning cases

DO 40 I=1,NUMCASES

    WRITE(4,26) I

    incase=(i-1)*numin+1

    ioutcase=(i-1)*numout

    CALL FORWARD_PASS_0(CASE_IN(incase),OUT,

1      WEIGHT_OUTLAYER,

2      NUMIN,NUMOUT,BIAS_OUTLAYER)

    WRITE(4,27) (case_out(ioutcase+j),OUT(J),

1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)

40  CONTINUE

CLOSE(4)

C    Save the current delta_weights and delta_biases

OPEN(4,FILE='MOMENTUM.OH')

WRITE(4,467)

DO 471 NOUTLAYER=1,NUMOUT

    NUMROW=(NOUTLAYER-1)*NUMIN

```

```

DO 471 INLAYER=1,NUMIN
    INDEX_WEIGHT=NUMROW+INLAYER
    WRITE(4,20)
1      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
471    CONTINUE
        WRITE(4,469)
        WRITE(4,20) (DELTA_BIAS_OUTLAYER(J),J=1,NUMOUT)
        CLOSE(4)
    endif
    if(isave.eq.0) go to 71
    endif

C
C    Run Test Cases
C
    inumcases=1
C    Read in the Test cases
        open(3,file=INTEST)
C    The following read is a dummy for compatibility
C    So that INLEARN & INTEST files will be interchangeable
        read(3,216) numo
350    read(3,217,end=357) infile,iin(inumcases)
        numcases=inumcases
        inumcases=inumcases+1
        inumin=1
C    Read in one test signal
        open(4,file=INFILE)

```

280

```
351  jjcase=(numcases-1)*numin+inumin
      read(4,252,end=353) case_in(jjcase)
      inumin=inumin+1
      go to 351
353  close(4)
      go to 350
357  close(3)
      write(6,259) numcases
      do 304 j=1,numcases
        do 303 jj=1,numout
          jjj=(j-1)*numout+jj
          if(iin(j).eq.jj) then
            case_out(jjj)=1.
          else
            case_out(jjj)=0.
          endif
303    continue
304  continue
C
C    Two Hidden Layers
C
      If (numlayers.eq.2) then
C    Save the success rate of the test cases
      OPEN(4,FILE='TESTCASE.2H')
      WRITE(4,325)
325  FORMAT(' TEST CASE ERRORS')
```

```

DO 2340 I=1,NUMCASES
    WRITE(4,26) I
    incase=(i-1)*numin+1
    ioutcase=(i-1)*numout
    CALL FORWARD_PASS_2(CASE_IN(INCASE),
1      OUT_HIDDEN1,OUT,
1      WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,
2      NUMIN,NUMHIDDEN1,NUMOUT,
2      BIAS_HIDDEN1,BIAS_OUTLAYER,
3      NUMHIDDEN2,BIAS_HIDDEN2,WEIGHT_HIDDEN2)
    WRITE(4,27) (case_out(ioutcase+j),OUT(J),
1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)
2340 CONTINUE
    CLOSE(4)

C
C      1 Hidden Layer
C
    else if (numlayers.eq.1) then
C      Save the success rate of the test cases
    OPEN(4,FILE='TESTCASE.1H')
    WRITE(4,325)
    DO 1340 I=1,NUMCASES
        WRITE(4,26) I
        incase=(i-1)*numin+1
        ioutcase=(i-1)*numout
        CALL FORWARD_PASS_1(CASE_IN(incase),

```

282

```
1      OUT_HIDDEN1,OUT,
1      WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,
2      NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,
2      BIAS_OUTLAYER)
      WRITE(4,27) (case_out(ioutcase+j),OUT(J),
1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)
1340  CONTINUE
      CLOSE(4)

C
C      0 Hidden layers
C
      else if(numlayers.eq.0) then
C      Save the success rate of the test cases
      OPEN(4,FILE='TESTCASE.OH')
      WRITE(4,325)
      DO 340 I=1,NUMCASES
        WRITE(4,26) I
        incase=(i-1)*numin+1
        ioutcase=(i-1)*numout
        CALL FORWARD_PASS_0(CASE_IN(incase),OUT,
1      WEIGHT_OUTLAYER,
2      NUMIN,NUMOUT,BIAS_OUTLAYER)
        WRITE(4,27) (case_out(ioutcase+j),OUT(J),
1      (case_out(ioutcase+j)-OUT(J)),J=1,NUMOUT)
340  CONTINUE
      CLOSE(4)
```

```

endif

If(numlay.ne.4.or.numlayers.eq.2) go to 999
numlayers=numlayers+1
go to 111
999 stop
END

C
C Subroutine to initialize 2 hidden layer network
C

SUBROUTINE INITIALIZE_NETWORK_2(WEIGHT_HIDDEN1,
1 WEIGHT_OUTLAYER,
1 DELTA_WEIGHT_HIDDEN1,DELTA_WEIGHT_OUTLAYER,
2 NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,BIAS_OUTLAYER,
3 DELTA_BIAS_HIDDEN1,DELTA_BIAS_OUTLAYER,
4 WEIGHT_HIDDEN2,DELTA_WEIGHT_HIDDEN2,NUMHIDDEN2,
5 BIAS_HIDDEN2,DELTA_BIAS_HIDDEN2)
REAL WEIGHT_HIDDEN1(1),WEIGHT_OUTLAYER(1),
1 DELTA_WEIGHT_HIDDEN1(1),DELTA_WEIGHT_OUTLAYER(1),
1 DELTA_WEIGHT_HIDDEN2(1),WEIGHT_HIDDEN2(1),
2 BIAS_HIDDEN1(1),BIAS_OUTLAYER(1),
3 DELTA_BIAS_HIDDEN1(1),DELTA_BIAS_OUTLAYER(1),
3 DELTA_BIAS_HIDDEN2(1),BIAS_HIDDEN2(1)
DATA WEIGHT_MAX/0.0000001/,BIAS_MAX/0.0000001/
INDEX_WEIGHT=0
w=weight_max/2.
b=bias_max/2.

```

```

DO 2 NHIDDEN1=1, NUMHIDDEN1
  DO 1 INLAYER=1, NUMIN
    INDEX_WEIGHT=INDEX_WEIGHT+1
    WEIGHT_HIDDEN1(INDEX_WEIGHT)=
1      RANDOM(WEIGHT_MAX) -w
    IF (WEIGHT_HIDDEN1(INDEX_WEIGHT).EQ.0.)
1      WEIGHT_HIDDEN1(INDEX_WEIGHT)=WEIGHT_MAX/2.
    DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)=0.0
1  CONTINUE
    BIAS_HIDDEN1(NHIDDEN1)=RANDOM(BIAS_MAX) -b
    DELTA_BIAS_HIDDEN1(NHIDDEN1)=0.0
2  CONTINUE
    INDEX_WEIGHT=0
    DO 4 NHIDDEN2=1, NUMHIDDEN2
      DO 3 NHIDDEN1=1, NUMHIDDEN1
        INDEX_WEIGHT=INDEX_WEIGHT+1
        WEIGHT_HIDDEN2(INDEX_WEIGHT)=
1      RANDOM(WEIGHT_MAX) -w
        IF (WEIGHT_HIDDEN2(INDEX_WEIGHT).EQ.0.)
1      WEIGHT_HIDDEN2(INDEX_WEIGHT)=WEIGHT_MAX/2.
        DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)=0.0
3      CONTINUE
        BIAS_HIDDEN2(NHIDDEN2)=RANDOM(BIAS_MAX) -b
        DELTA_BIAS_HIDDEN2(NHIDDEN2)=0.0
4  CONTINUE
    INDEX_WEIGHT=0

```

```

DO 6 NOUTLAYER=1,NUMOUT
    DO 5 NHIDDEN2=1,NUMHIDDEN2
        INDEX_WEIGHT=INDEX_WEIGHT+1
        WEIGHT_OUTLAYER(INDEX_WEIGHT)=
1        RANDOM(WEIGHT_MAX) -w
        IF(WEIGHT_OUTLAYER(INDEX_WEIGHT).EQ.0.)
1        WEIGHT_OUTLAYER(INDEX_WEIGHT)=WEIGHT_MAX/2.
        DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=0.0
5    CONTINUE
        BIAS_OUTLAYER(NOUTLAYER)=RANDOM(BIAS_MAX) -b
        DELTA_BIAS_OUTLAYER(NOUTLAYER)=0.0
6 CONTINUE
    RETURN
END

```

C

C Subroutine to do forward pass through 2 Hidden Layer

C

```

SUBROUTINE FORWARD_PASS_2(IN,OUT_HIDDEN1,OUT,
1 WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,
2 NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,BIAS_OUTLAYER,
3 NUMHIDDEN2,BIAS_HIDDEN2,WEIGHT_HIDDEN2)
    REAL IN(1),OUT_HIDDEN1(1),OUT(1),
1 WEIGHT_HIDDEN1(1),WEIGHT_OUTLAYER(1),
2 BIAS_HIDDEN1(1),BIAS_OUTLAYER(1),NET,
3 WEIGHT_HIDDEN2(1),OUT_HIDDEN2(1),
4 BIAS_HIDDEN2(1)

```


C*****

C FIRST HIDDEN LAYER FORWARD PASS

INDEX_WEIGHT=0

DO 2 NHIDDEN1=1, NUMHIDDEN1

NET=0.

DO 1 INLAYER=1, NUMIN

INDEX_WEIGHT=INDEX_WEIGHT+1

NET=NET+IN(INLAYER)*WEIGHT_HIDDEN1(INDEX_WEIGHT)

1 CONTINUE

NET=NET+BIAS_HIDDEN1(NHIDDEN1)

OUT_HIDDEN1(NHIDDEN1)=1.0/(1.0+EXP(-NET))

2 CONTINUE

C*****

C SECOND HIDDEN LAYER FORWARD PASS

INDEX_WEIGHT=0

DO 4 NHIDDEN2=1, NUMHIDDEN2

NET=0.

DO 3 NHIDDEN1=1, NUMHIDDEN1

INDEX_WEIGHT=INDEX_WEIGHT+1

NET=NET+OUT_HIDDEN1(NHIDDEN1)*

1 WEIGHT_HIDDEN2(INDEX_WEIGHT)

3 CONTINUE

NET=NET+BIAS_HIDDEN2(NHIDDEN2)

OUT_HIDDEN2(NHIDDEN2)=1.0/(1.0+EXP(-NET))

4 CONTINUE

C*****

C OUTPUT LAYER FORWARD PASS

INDEX_WEIGHT=0

DO 6 NOUTLAYER=1,NUMOUT

NET=0.

DO 5 NHIDDEN2=1,NUMHIDDEN2

INDEX_WEIGHT=INDEX_WEIGHT+1

NET=NET+OUT_HIDDEN2 (NHIDDEN2) *

1 WEIGHT_OUTLAYER (INDEX_WEIGHT)

5 CONTINUE

NET=NET+BIAS_OUTLAYER (NOUTLAYER)

OUT (NOUTLAYER)=1.0/(1.0+EXP (-NET))

6 CONTINUE

C*****

RETURN

END

C

C Subroutine to do backward pass through 2 Hidden Layer

C

SUBROUTINE BACKWARD_PASS_2 (IN,OUT_HIDDEN1,OUT,

1 WEIGHT_HIDDEN1,WEIGHT_OUTLAYER,OUT_HIDDEN2,

2 NUMIN,NUMHIDDEN1,NUMOUT,BIAS_HIDDEN1,BIAS_OUTLAYER,

3 TARGET,ERROR,DELTA_OUT,DELTA_BIAS_OUTLAYER,

4 DELTA_WEIGHT_OUTLAYER,DELTA_HIDDEN1,

4 DELTA_BIAS_HIDDEN1,

5 DELTA_WEIGHT_HIDDEN1,LEARNING_RATE,MOMENTUM,

6 WEIGHT_HIDDEN2,NUMHIDDEN2,BIAS_HIDDEN2,DELTA_HIDDEN2,

```

7 DELTA_BIAS_HIDDEN2,DELTA_WEIGHT_HIDDEN2)
  REAL IN(1),OUT_HIDDEN1(1),OUT_HIDDEN2(1),OUT(1),
1 WEIGHT_HIDDEN1(1),WEIGHT_OUTLAYER(1),
1 WEIGHT_HIDDEN2(1),BIAS_HIDDEN1(1),
2 BIAS_HIDDEN2(1),BIAS_OUTLAYER(1),
3 TARGET(1),ERROR(1),DELTA_OUT(1),
3 DELTA_BIAS_OUTLAYER(1),
4 DELTA_WEIGHT_OUTLAYER(1),NET,LEARNING_RATE,MOMENTUM,
5 DELTA_HIDDEN1(1),DELTA_BIAS_HIDDEN1(1),
5 DELTA_HIDDEN2(1),DELTA_BIAS_HIDDEN2(1),
6 DELTA_WEIGHT_HIDDEN1(1),DELTA_WEIGHT_HIDDEN2(1)

```

C*****

C OUTPUT LAYER

 INDEX_WEIGHT=0

 DO 2 NOUTLAYER=1,NUMOUT

 SIGMOID=OUT(NOUTLAYER)

 SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)

 ERROR(NOUTLAYER)=TARGET(NOUTLAYER)-OUT(NOUTLAYER)

 DELTA_OUT(NOUTLAYER)=SIGMOID_PRIME*ERROR(NOUTLAYER)

 DELTA_BIAS_OUTLAYER(NOUTLAYER)=LEARNING_RATE*

1 DELTA_OUT(NOUTLAYER)+MOMENTUM*

1 DELTA_BIAS_OUTLAYER(NOUTLAYER)

 DO 1 NHIDDEN2=1,NUMHIDDEN2

 INDEX_WEIGHT=INDEX_WEIGHT+1

 DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=

1 LEARNING_RATE*

```

1      DELTA_OUT(NOUTLAYER) *
1      OUT_HIDDEN2(NHIDDEN2)+MOMENTUM*
2      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
1      CONTINUE
2      CONTINUE

C*****
C      HIDDEN LAYER 2
      INDEX_WEIGHT=0
      DO 5 NHIDDEN2=1, NUMHIDDEN2
          SIGMOID=OUT_HIDDEN2(NHIDDEN2)
          SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)
          NET=0.0
          DO 3 NOUTLAYER=1, NUMOUT
              INDEX_2_WEIGHT=(NOUTLAYER-1)*NUMHIDDEN2+NHIDDEN2
              NET=NET+WEIGHT_OUTLAYER(INDEX_2_WEIGHT) *
1          DELTA_OUT(NOUTLAYER)
3      CONTINUE
          DELTA_HIDDEN2(NHIDDEN2)=SIGMOID_PRIME*NET
          DELTA_BIAS_HIDDEN2(NHIDDEN2)=LEARNING_RATE*
1      DELTA_HIDDEN2(NHIDDEN2)+MOMENTUM*
1      DELTA_BIAS_HIDDEN2(NHIDDEN2)
          DO 4 NHIDDEN1=1, NUMHIDDEN1
              INDEX_WEIGHT=INDEX_WEIGHT+1
              DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)=
1          LEARNING_RATE*
1          DELTA_HIDDEN2(NHIDDEN2) *

```

290

```
1      OUT_HIDDEN1(NHIDDEN1)+MOMENTUM*
2      DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)
4      CONTINUE
5 CONTINUE
```

C*****

```
C      HIDDEN LAYER 1
      INDEX_WEIGHT=0
      DO 15 NHIDDEN1=1,NUMHIDDEN1
          SIGMOID=OUT_HIDDEN1(NHIDDEN1)
          SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)
          NET=0.0
          DO 13 NHIDDEN2=1,NUMHIDDEN2
              INDEX_2_WEIGHT=(NHIDDEN2-1)*NUMHIDDEN1+NHIDDEN1
              NET=NET+WEIGHT_HIDDEN2(INDEX_2_WEIGHT)*
1          DELTA_HIDDEN2(NHIDDEN2)
13      CONTINUE
          DELTA_HIDDEN1(NHIDDEN1)=SIGMOID_PRIME*NET
          DELTA_BIAS_HIDDEN1(NHIDDEN1)=LEARNING_RATE*
1      DELTA_HIDDEN1(NHIDDEN1)+MOMENTUM*
1      DELTA_BIAS_HIDDEN1(NHIDDEN1)
          BIAS_HIDDEN1(NHIDDEN1)=BIAS_HIDDEN1(NHIDDEN1)+
1      DELTA_BIAS_HIDDEN1(NHIDDEN1)
          DO 14 INLAYER=1,NUMIN
              INDEX_WEIGHT=INDEX_WEIGHT+1
              DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)=
1          LEARNING_RATE*
```

```

1      DELTA_HIDDEN1(NHIDDEN1)*IN(INLAYER)+MOMENTUM*
2      DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
      WEIGHT_HIDDEN1(INDEX_WEIGHT)=
1      WEIGHT_HIDDEN1(INDEX_WEIGHT)+
1      DELTA_WEIGHT_HIDDEN1(INDEX_WEIGHT)
14     CONTINUE
15 CONTINUE

C*****
C     HIDDEN LAYER 2
C     HAD TO WAIT TO UPDATE WEIGHT_HIDDEN2
C     SINCE IT WAS USED TO CALCULATE DELTA_HIDDEN1
      INDEX_WEIGHT=0
      DO 17 NHIDDEN2=1,NUMHIDDEN2
          BIAS_HIDDEN2(NHIDDEN2)=BIAS_HIDDEN2(NHIDDEN2)+
1      DELTA_BIAS_HIDDEN2(NHIDDEN2)
          DO 16 NHIDDEN1=1,NUMHIDDEN1
              INDEX_WEIGHT=INDEX_WEIGHT+1
              WEIGHT_HIDDEN2(INDEX_WEIGHT)=
1      WEIGHT_HIDDEN2(INDEX_WEIGHT)+
1      DELTA_WEIGHT_HIDDEN2(INDEX_WEIGHT)
16     CONTINUE
17 CONTINUE

C*****
C     OUTPUT LAYER
C     HAD TO WAIT TO UPDATE WEIGHT_OUTLAYER
C     SINCE IT WAS USED TO CALCULATE DELTA_HIDDEN2

```

```

INDEX_WEIGHT=0
DO 7 NOUTLAYER=1,NUMOUT
    BIAS_OUTLAYER(NOUTLAYER)=BIAS_OUTLAYER(NOUTLAYER)+
1    DELTA_BIAS_OUTLAYER(NOUTLAYER)
    DO 6 NHIDDEN2=1,NUMHIDDEN2
        INDEX_WEIGHT=INDEX_WEIGHT+1
        WEIGHT_OUTLAYER(INDEX_WEIGHT)=
1        WEIGHT_OUTLAYER(INDEX_WEIGHT)+
1        DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
6    CONTINUE
7 CONTINUE

C*****
    RETURN
    END

C
C    Subroutine to generate random no.s between 0 and RANGE
C

FUNCTION RANDOM(RANGE)
INTEGER*2 I1,I(2)
EQUIVALENCE (LONG,I(1))
DATA I1/1001/,I/0,2001/
I(1)=0
LONG=LONG*I1
R=(I(2)+32768)/65536.
RANDOM=R*RANGE
RETURN

```

END

C

C Subroutine to initialize 1 hidden layer network

C

```

SUBROUTINE INITIALIZE_NETWORK_1(WEIGHT_HIDDEN,
1 WEIGHT_OUTLAYER,
1 DELTA_WEIGHT_HIDDEN, DELTA_WEIGHT_OUTLAYER,
2 NUMIN, NUMHIDDEN, NUMOUT, BIAS_HIDDEN, BIAS_OUTLAYER,
3 DELTA_BIAS_HIDDEN, DELTA_BIAS_OUTLAYER)
  REAL WEIGHT_HIDDEN(1), WEIGHT_OUTLAYER(1),
1 DELTA_WEIGHT_HIDDEN(1), DELTA_WEIGHT_OUTLAYER(1),
2 BIAS_HIDDEN(1), BIAS_OUTLAYER(1),
3 DELTA_BIAS_HIDDEN(1), DELTA_BIAS_OUTLAYER(1)
  DATA WEIGHT_MAX/0.0000001/, BIAS_MAX/0.0000001/
  INDEX_WEIGHT=0
  w=weight_max/2.
  b=bias_max/2.
  DO 2 NHIDDEN=1, NUMHIDDEN
    DO 1 INLAYER=1, NUMIN
      INDEX_WEIGHT=INDEX_WEIGHT+1
      WEIGHT_HIDDEN(INDEX_WEIGHT)=RANDOM(WEIGHT_MAX)-w
      IF(WEIGHT_HIDDEN(INDEX_WEIGHT).EQ.0.)
1      WEIGHT_HIDDEN(INDEX_WEIGHT)=WEIGHT_MAX/2.
      DELTA_WEIGHT_HIDDEN(INDEX_WEIGHT)=0.0
1    CONTINUE
      BIAS_HIDDEN(NHIDDEN)=RANDOM(BIAS_MAX)-b

```



```

      DELTA_BIAS_HIDDEN(NHIDDEN)=0.0
2  CONTINUE

      INDEX_WEIGHT=0
      DO 4 NOUTLAYER=1,NUMOUT
        DO 3 NHIDDEN=1,NUMHIDDEN
          INDEX_WEIGHT=INDEX_WEIGHT+1
          WEIGHT_OUTLAYER(INDEX_WEIGHT)=
1          RANDOM(WEIGHT_MAX)-w
          IF(WEIGHT_OUTLAYER(INDEX_WEIGHT).EQ.0.)
1          WEIGHT_OUTLAYER(INDEX_WEIGHT)=WEIGHT_MAX/2.
          DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=0.0
3      CONTINUE
      BIAS_OUTLAYER(NOUTLAYER)=RANDOM(BIAS_MAX)-b
      DELTA_BIAS_OUTLAYER(NOUTLAYER)=0.0
4  CONTINUE

      RETURN

      END

```

C

C Subroutine to do forward pass through 1 Hidden Layer

C

```

      SUBROUTINE FORWARD_PASS_1(IN,OUT_HIDDEN,OUT,
1  WEIGHT_HIDDEN,WEIGHT_OUTLAYER,
2  NUMIN,NUMHIDDEN,NUMOUT,BIAS_HIDDEN,BIAS_OUTLAYER)
      REAL IN(1),OUT_HIDDEN(1),OUT(1),
1  WEIGHT_HIDDEN(1),WEIGHT_OUTLAYER(1),
2  BIAS_HIDDEN(1),BIAS_OUTLAYER(1),NET

```

C*****

C HIDDEN LAYER FORWARD PASS

 INDEX_WEIGHT=0

 DO 2 NHIDDEN=1, NUMHIDDEN

 NET=0.

 DO 1 INLAYER=1, NUMIN

 INDEX_WEIGHT=INDEX_WEIGHT+1

 NET=NET+IN(INLAYER)*WEIGHT_HIDDEN(INDEX_WEIGHT)

 1 CONTINUE

 NET=NET+BIAS_HIDDEN(NHIDDEN)

 OUT_HIDDEN(NHIDDEN)=1.0/(1.0+EXP(-NET))

 2 CONTINUE

C*****

C OUTPUT LAYER FORWARD PASS

 INDEX_WEIGHT=0

 DO 4 NOUTLAYER=1, NUMOUT

 NET=0.

 DO 3 NHIDDEN=1, NUMHIDDEN

 INDEX_WEIGHT=INDEX_WEIGHT+1

 NET=NET+OUT_HIDDEN(NHIDDEN)*

 1 WEIGHT_OUTLAYER(INDEX_WEIGHT)

 3 CONTINUE

 NET=NET+BIAS_OUTLAYER(NOUTLAYER)

 OUT(NOUTLAYER)=1.0/(1.0+EXP(-NET))

 4 CONTINUE

C*****

```
RETURN
```

```
END
```

```
C
```

```
C Subroutine to do backward pass through 1 Hidden Layer
```

```
C
```

```

SUBROUTINE BACKWARD_PASS_1(IN,OUT_HIDDEN,OUT,
1 WEIGHT_HIDDEN,WEIGHT_OUTLAYER,
2 NUMIN,NUMHIDDEN,NUMOUT,BIAS_HIDDEN,BIAS_OUTLAYER,
3 TARGET,ERROR,DELTA_OUT,DELTA_BIAS_OUTLAYER,
4 DELTA_WEIGHT_OUTLAYER,DELTA_HIDDEN,DELTA_BIAS_HIDDEN,
5 DELTA_WEIGHT_HIDDEN,LEARNING_RATE,MOMENTUM)
REAL IN(1),OUT_HIDDEN(1),OUT(1),
1 WEIGHT_HIDDEN(1),WEIGHT_OUTLAYER(1),
2 BIAS_HIDDEN(1),BIAS_OUTLAYER(1),
3 TARGET(1),ERROR(1),DELTA_OUT(1),
3 DELTA_BIAS_OUTLAYER(1),
4 DELTA_WEIGHT_OUTLAYER(1),NET,LEARNING_RATE,MOMENTUM,
5 DELTA_HIDDEN(1),DELTA_BIAS_HIDDEN(1),
6 DELTA_WEIGHT_HIDDEN(1)

```

```
C*****
```

```
C OUTPUT LAYER
```

```
INDEX_WEIGHT=0
```

```
DO 2 NOUTLAYER=1,NUMOUT
```

```
SIGMOID=OUT(NOUTLAYER)
```

```
SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)
```

```
ERROR(NOUTLAYER)=TARGET(NOUTLAYER)-OUT(NOUTLAYER)
```

```

        DELTA_OUT(NOUTLAYER)=SIGMOID_PRIME*ERROR(NOUTLAYER)
        DELTA_BIAS_OUTLAYER(NOUTLAYER)=LEARNING_RATE*
1      DELTA_OUT(NOUTLAYER)+MOMENTUM*
1      DELTA_BIAS_OUTLAYER(NOUTLAYER)
      DO 1 NHIDDEN=1,NUMHIDDEN
        INDEX_WEIGHT=INDEX_WEIGHT+1
        DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=
1      LEARNING_RATE*
1      DELTA_OUT(NOUTLAYER)*
1      OUT_HIDDEN(NHIDDEN)+MOMENTUM*
2      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
1    CONTINUE
2  CONTINUE

C*****
C    HIDDEN LAYER
      INDEX_WEIGHT=0
      DO 5 NHIDDEN=1,NUMHIDDEN
        SIGMOID=OUT_HIDDEN(NHIDDEN)
        SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)
        NET=0.0
        DO 3 NOUTLAYER=1,NUMOUT
          INDEX_2_WEIGHT=(NOUTLAYER-1)*NUMHIDDEN+NHIDDEN
          NET=NET+WEIGHT_OUTLAYER(INDEX_2_WEIGHT)
1        *DELTA_OUT(NOUTLAYER)
3      CONTINUE
      DELTA_HIDDEN(NHIDDEN)=SIGMOID_PRIME*NET

```

```

    DELTA_BIAS_HIDDEN(NHIDDEN)=LEARNING_RATE*
1    DELTA_HIDDEN(NHIDDEN)
1    +MOMENTUM*DELTA_BIAS_HIDDEN(NHIDDEN)
    BIAS_HIDDEN(NHIDDEN)=BIAS_HIDDEN(NHIDDEN)+
1    DELTA_BIAS_HIDDEN(NHIDDEN)
    DO 4 INLAYER=1,NUMIN
        INDEX_WEIGHT=INDEX_WEIGHT+1
        DELTA_WEIGHT_HIDDEN(INDEX_WEIGHT)=LEARNING_RATE*
1        DELTA_HIDDEN(NHIDDEN)*IN(INLAYER)+MOMENTUM*
2        DELTA_WEIGHT_HIDDEN(INDEX_WEIGHT)
        WEIGHT_HIDDEN(INDEX_WEIGHT)=
1        WEIGHT_HIDDEN(INDEX_WEIGHT)+
1        DELTA_WEIGHT_HIDDEN(INDEX_WEIGHT)
4    CONTINUE
5 CONTINUE

C*****
C    OUTPUT LAYER
C    HAD TO WAIT TO UPDATE WEIGHT_OUTLAYER
C    SINCE IT WAS USED TO CALCULATE DELTA_HIDDEN
    INDEX_WEIGHT=0
    DO 7 NOUTLAYER=1,NUMOUT
        BIAS_OUTLAYER(NOUTLAYER)=BIAS_OUTLAYER(NOUTLAYER)+
1    DELTA_BIAS_OUTLAYER(NOUTLAYER)
        DO 6 NHIDDEN=1,NUMHIDDEN
            INDEX_WEIGHT=INDEX_WEIGHT+1
            WEIGHT_OUTLAYER(INDEX_WEIGHT)=

```

```

1      WEIGHT_OUTLAYER(INDEX_WEIGHT)+
1      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)
6      CONTINUE
7      CONTINUE

C*****
      RETURN
      END

C
C      Subroutine to initialize 0 hidden layer network
C
      SUBROUTINE INITIALIZE_NETWORK_0(WEIGHT_OUTLAYER,
1  DELTA_WEIGHT_OUTLAYER,
2  NUMIN,NUMOUT,BIAS_OUTLAYER,
3  DELTA_BIAS_OUTLAYER)
      REAL WEIGHT_OUTLAYER(1),
1  DELTA_WEIGHT_OUTLAYER(1),
2  BIAS_OUTLAYER(1),
3  DELTA_BIAS_OUTLAYER(1)
      DATA WEIGHT_MAX/0.0000001/,BIAS_MAX/0.0000001/
      INDEX_WEIGHT=0
      w=weight_max/2.
      b=bias_max/2.
      INDEX_WEIGHT=0
      DO 4 NOUTLAYER=1,NUMOUT
        DO 1 INLAYER=1,NUMIN
          INDEX_WEIGHT=INDEX_WEIGHT+1

```

```

        WEIGHT_OUTLAYER(INDEX_WEIGHT)=
1      RANDOM(WEIGHT_MAX)-w
        IF(WEIGHT_OUTLAYER(INDEX_WEIGHT).EQ.0.)
1      WEIGHT_OUTLAYER(INDEX_WEIGHT)=WEIGHT_MAX/2.
        DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=0.0
1      CONTINUE
        BIAS_OUTLAYER(NOUTLAYER)=RANDOM(BIAS_MAX)-b
        DELTA_BIAS_OUTLAYER(NOUTLAYER)=0.0
4      CONTINUE
        RETURN
        END

```

C

C Subroutine to do forward pass through 0 Hidden Layer

C

```

        SUBROUTINE FORWARD_PASS_0(IN,OUT,
1  WEIGHT_OUTLAYER,
2  NUMIN,NUMOUT,BIAS_OUTLAYER)
        REAL IN(1),OUT(1),
1  WEIGHT_OUTLAYER(1),
2  BIAS_OUTLAYER(1),NET

```

C*****

C OUTPUT LAYER FORWARD PASS

INDEX_WEIGHT=0

DO 2 NOUTLAYER=1,NUMOUT

NET=0.

DO 1 INLAYER=1,NUMIN

```

        INDEX_WEIGHT=INDEX_WEIGHT+1

        NET=NET+IN(INLAYER)*

1      WEIGHT_OUTLAYER(INDEX_WEIGHT)

1      CONTINUE

        NET=NET+BIAS_OUTLAYER(NOUTLAYER)

        OUT(NOUTLAYER)=1.0/(1.0+EXP(-NET))

2      CONTINUE

C*****

        RETURN

        END

C

C      Subroutine to do backward pass through 0 Hidden Layer

C

        SUBROUTINE BACKWARD_PASS_0(IN,OUT,

1  WEIGHT_OUTLAYER,

2  NUMIN,NUMOUT,BIAS_OUTLAYER,

3  TARGET,ERROR,DELTA_OUT,DELTA_BIAS_OUTLAYER,

4  DELTA_WEIGHT_OUTLAYER,

5  LEARNING_RATE,MOMENTUM)

        REAL IN(1),OUT(1),

1  WEIGHT_OUTLAYER(1),

2  BIAS_OUTLAYER(1),

3  TARGET(1),ERROR(1),DELTA_OUT(1),

4  DELTA_BIAS_OUTLAYER(1),

5  DELTA_WEIGHT_OUTLAYER(1),LEARNING_RATE,MOMENTUM

C*****

```



```

C      OUTPUT LAYER

      INDEX_WEIGHT=0

      DO 2 NOUTLAYER=1, NUMOUT

          SIGMOID=OUT(NOUTLAYER)

          SIGMOID_PRIME=(SIGMOID)*(1.0-SIGMOID)

          ERROR(NOUTLAYER)=TARGET(NOUTLAYER)-OUT(NOUTLAYER)

          DELTA_OUT(NOUTLAYER)=SIGMOID_PRIME*ERROR(NOUTLAYER)

          DELTA_BIAS_OUTLAYER(NOUTLAYER)=LEARNING_RATE*

1      DELTA_OUT(NOUTLAYER)+MOMENTUM*

1      DELTA_BIAS_OUTLAYER(NOUTLAYER)

      BIAS_OUTLAYER(NOUTLAYER)=BIAS_OUTLAYER(NOUTLAYER)+

1      DELTA_BIAS_OUTLAYER(NOUTLAYER)

      DO 1 INLAYER=1, NUMIN

          INDEX_WEIGHT=INDEX_WEIGHT+1

          DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)=

1      LEARNING_RATE*

1      DELTA_OUT(NOUTLAYER)*IN(INLAYER)+MOMENTUM*

2      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)

          WEIGHT_OUTLAYER(INDEX_WEIGHT)=

1      WEIGHT_OUTLAYER(INDEX_WEIGHT)+

1      DELTA_WEIGHT_OUTLAYER(INDEX_WEIGHT)

1      CONTINUE

2      CONTINUE

C*****

      RETURN

      END

```

C

C Subroutine to scramble order of inputs

C

SUBROUTINE RANDOMIZE(ORDER,NUM)

INTEGER*2 ORDER(1),NORDER(150)

DO 3 J=1,NUM

NORDER(J)=J

3 CONTINUE

NN=NUM

DO 4 J=1,NUM

RANGE=NN

II=RANDOM(RANGE)

II=II+1

IF(II.GT.NUM) II=NUM

ORDER(J)=NORDER(II)

DO 2 JJ=II,NN

NORDER(JJ)=NORDER(JJ+1)

2 CONTINUE

NN=NN-1

4 CONTINUE

RETURN

END

APPENDIX D
RESULTS

RESULTS is used to consolidate the results obtained from an all layer pass of BACKPROP (i.e., option 4 for 0, 1, and 2 hidden layers). It produces consolidated weight tables and statistics as to the number of learning and test cases successfully classified.

D.1 RESULTS Example Run

To run the RESULTS program type
results.

The program needs no keyboard inputs as it has no options. It creates several output files containing weights and biases and one file containing the consolidated statistics (called RESULTS). An example RESULTS file is shown in Figures 74 and 75.

```
2 hidden layers took 831 learning iterations;
1 hidden layer took 1568 learning iterations;
0 hidden layers took 50001 learning iterations.
For 2 Hidden layers and 39 learn cases:
  39 were correctly classified
  0 were incorrectly classified
  0 were not classified
  100.00 % were correctly classified
  .00 % were incorrectly classified
  .0046650 was the average amount of uncertainty
  .0293080 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty
For 1 Hidden layers and 39 learn cases:
  39 were correctly classified
  0 were incorrectly classified
  0 were not classified
  100.00 % were correctly classified
  .00 % were incorrectly classified
  .0047615 was the average amount of uncertainty
  .0253820 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty
For 0 Hidden layers and 39 learn cases:
  39 were correctly classified
  0 were incorrectly classified
  0 were not classified
  100.00 % were correctly classified
  .00 % were incorrectly classified
  .0065110 was the average amount of uncertainty
  .0384260 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty
```

Figure 74. First Part of RESULTS Sample Output

```

For 2 Hidden layers and 149 test cases:
  136 were correctly classified
    3 were incorrectly classified
    10 were not classified
  91.28 % were correctly classified
    2.01 % were incorrectly classified
  .0050103 was the average amount of uncertainty
  .2102960 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty
For 1 Hidden layers and 149 test cases:
  130 were correctly classified
    2 were incorrectly classified
    17 were not classified
  87.25 % were correctly classified
    1.34 % were incorrectly classified
  .0105614 was the average amount of uncertainty
  .3167200 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty
For 0 Hidden layers and 149 test cases:
  116 were correctly classified
    3 were incorrectly classified
    30 were not classified
  77.85 % were correctly classified
    2.01 % were incorrectly classified
  .0160266 was the average amount of uncertainty
  .4818220 was the maximum amount of uncertainty
  .0000000 was the minimum amount of uncertainty

```

Figure 75. Second Part of RESULTS Output File

D.2 Listing of RESULTS

C PROGRAM TO MAKE WEIGHT MAPS

C

C Note: The following are maximum values

C The actual may be less than these numbers

C

PARAMETER (NUMIN1=64)

PARAMETER (NUMHIDDEN11=100)

```

PARAMETER (NUMHIDDEN21=100)
PARAMETER (NUMOUT1=10)
PARAMETER (NUMCASES1=500)
PARAMETER (NUMWEIGHTHIDDEN1=NUMIN1*NUMHIDDEN11)
PARAMETER (NUMWEIGHTHIDDEN2=NUMHIDDEN11*NUMHIDDEN21)
PARAMETER (NUMWEIGHTOUT=NUMHIDDEN21*NUMOUT1)
PARAMETER (NUMCASIN1=NUMIN1*NUMCASES1)
PARAMETER (NUMCASOUT1=NUMOUT1*NUMCASES1)
C   The network is saved in case of power outage every
C   auto_savel iterations; may be restarted
PARAMETER (auto_savel=50)
C
common/i7/ weight_hidden1
common/i8/ weight_hidden2
common/i9/ weight_outlayer
INTEGER*1 IIA,NNA,BASE,LETTERS(NUMWEIGHTHIDDEN2)
INTEGER*2 BASECHAR
REAL WEIGHT_HIDDEN1(NUMWEIGHTHIDDEN1),
1 WEIGHT_HIDDEN2(NUMWEIGHTHIDDEN2),
1 WEIGHT_OUTLAYER(NUMWEIGHTOUT),
2 BIAS_HIDDEN1(NUMHIDDEN11),
2 BIAS_OUTLAYER(NUMOUT1),OUT(NUMOUT1),
2 BIAS_HIDDEN2(NUMHIDDEN21),ERROR(NUMOUT1),
2 CASE_OUT(NUMCASOUT1)
EQUIVALENCE(BASE,BASECHAR)
DATA IIA/1HA/,NNA/1Ha/

```

```

DATA WMAX/2./
WINC=WMAX/20.

C      Two Hidden Layers
C
      OPEN(4,FILE='NET.2H')
      OPEN(3,FILE='WEIGHT1.2H')

C      Get number of learning passes required
      READ(4,211) ITERATIONS2
211      FORMAT(11X,I7)

C      Get Number Of Inputs
      READ(4,218) NUMIN
218      format(i4)
219      FORMAT(A1)

C      Get Number Of Hidden Neurons Layer 1
      READ(4,218) NUMHIDDEN1

C      Covert weight values to alphabetic character map
      DO 2231 NHIDDEN1=1,NUMHIDDEN1
          NUMROW=(NHIDDEN1-1)*NUMIN
          DO 2230 INLAYER=1,NUMIN
              INDEX_WEIGHT=NUMROW+INLAYER
              READ(4,20) W
C              W IS WEIGHT_HIDDEN1(INDEX_WEIGHT)
              BASE=IIA
              IF(W.LT.0) THEN
                  BASE=NNA
                  W=-W

```



```

ENDIF
IF (W.EQ.0.) THEN
    BASE='0'
ELSE
    DO J=1,24
        IF(W.GE.WINC) THEN
            BASE=BASE+1
            W=W-WINC
        ENDIF
    ENDDO

    FIND.F
    IF(W.GE.WINC) BASE=BASE+1
    LETTERS(INLAYER)=BASECHAR
2230    CONTINUE
        WRITE(3,7) (LETTERS(JH),JH=1,NUMIN)
7    FORMAT(100A1)
2231    CONTINUE
        CLOSE(3)
        OPEN(3,FILE='WEIGHT2.2H')
C    Get Number Of Neurons Hidden Layer 2
        READ(4,218) NUMHIDDEN2
C    Covert weight values to alphabetic character map
        DO 2232 NHIDDEN2=1,NUMHIDDEN2
            NUMROW=(NHIDDEN2-1)*NUMHIDDEN1
            DO 2225 NHIDDEN1=1,NUMHIDDEN1
                INDEX_WEIGHT=NUMROW+INLAYER

```

```

      READ(4,20) W
C      W=WEIGHT_HIDDEN2 (INDEX_WEIGHT)
      BASE=IIA
      IF(W.LT.0) THEN
          BASE=NNA
          W=-W
      ENDIF
      IF (W.EQ.0.) THEN
          BASE='0'
      ELSE
          DO J=1,24
              IF(W.GE.WINC) THEN
                  BASE=BASE+1
                  W=W-WINC
              ENDIF
          ENDDO
      ENDIF
      IF(W.GE.WINC) BASE=BASE+1
      LETTERS (NHIDDEN1)=BASECHAR
2225      CONTINUE
          WRITE(3,7) (LETTERS (JH) ,JH=1,NUMHIDDEN1)
2232      CONTINUE
          CLOSE(3)
          OPEN(3,FILE='WEIGHT3.2H')
C      Get Number Of Outputs
          READ(4,218) NUMOUT

```

312

```
      if (numout.lt.1) then
        write(6,934)
934      format(' Error in file WEIGHT3.2H,',
1          ' enter number of outputs')
        read(5,935) numout
935      format(I6)
      endif
C      Covert weight values to alphabetic character map
      DO 2241 NOUTLAYER=1,NUMOUT
        NUMROW=(NOUTLAYER-1)*NUMHIDDEN2
        DO 2229 NHIDDEN2=1,NUMHIDDEN2
          INDEX_WEIGHT=NUMROW+NHIDDEN2
          READ(4,20) W
C          W=WEIGHT_OUTLAYER(INDEX_WEIGHT)
          BASE=IIA
          IF(W.LT.0) THEN
            BASE=NNA
            W=-W
          ENDIF
          IF (W.EQ.0.) THEN
            BASE='0'
          ELSE
            DO J=1,24
              IF(W.GE.WINC) THEN
                BASE=BASE+1
                W=W-WINC
```

```

                                ENDIF
                                ENDDO
                                ENDIF
                                IF(W.GE.WINC) BASE=BASE+1
                                LETTERS(NHIDDEN2)=BASECHAR
2229          CONTINUE
                                WRITE(3,7) (LETTERS(JH),JH=1,NUMHIDDEN2)
2241  CONTINUE
                                CLOSE(3)
                                READ(4,219) dummy_read
                                READ(4,20) (ERROR(I),I=1,NUMOUT)
                                OPEN(3,FILE='BIAS1.2H')
                                READ(4,219) dummy_read
C      Covert weight values to alphabetic character map
                                READ(4,20) (BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
                                DO 1730 IJKL=1,NUMHIDDEN1
                                    W=BIAS_HIDDEN1(IJKL)
                                    BASE=IIA
                                    IF(W.LT.0) THEN
                                        BASE=NNA
                                        W=-W
                                    ENDIF
                                    IF (W.EQ.0.) THEN
                                        BASE='0'
                                    ELSE
                                        DO J=1,24

```

```
                IF(W.GE.WINC) THEN
                    BASE=BASE+1
                    W=W-WINC
                ENDIF
            ENDDO
        ENDIF
        IF(W.GE.WINC) BASE=BASE+1
        LETTERS(IJKL)=BASECHAR
1730    CONTINUE
        WRITE(3,7) (LETTERS(JH),JH=1,NUMHIDDEN1)
        CLOSE(3)
        OPEN(3,FILE='BIAS2.2H')
        READ(4,219) dummy_read
C      Covert weight values to alphabetic character map
        READ(4,20) (BIAS_HIDDEN2(J),J=1,NUMHIDDEN2)
        DO 1731 IJKL=1,NUMHIDDEN2
            W=BIAS_HIDDEN2(IJKL)
            BASE=IIA
            IF(W.LT.0) THEN
                BASE=NNA
                W=-W
            ENDIF
            IF (W.EQ.0.) THEN
                BASE='0'
            ELSE
                DO J=1,24
```

```

                                IF(W.GE.WINC) THEN
                                    BASE=BASE+1
                                    W=W-WINC
                                ENDIF
                            ENDDO
                        ENDIF
                    IF(W.GE.WINC) BASE=BASE+1
                    LETTERS(IJKL)=BASECHAR
1731      CONTINUE
                WRITE(3,7) (LETTERS(JH),JH=1,NUMHIDDEN2)
            CLOSE(3)
        OPEN(3,FILE='BIAS3.2H')
        READ(4,219) dummy_read
        READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)
C      Covert weight values to alphabetic character map
        DO 1732 IJKL=1,NUMOUT
            W=BIAS_OUTLAYER(IJKL)
            BASE=IIA
            IF(W.LT.0) THEN
                BASE=NNA
                W=-W
            ENDIF
            IF (W.EQ.0.) THEN
                BASE='0'
            ELSE
                DO J=1,24

```

```

        IF(W.GE.WINC) THEN
            BASE=BASE+1
            W=W-WINC
        ENDIF
    ENDDO

ENDIF

    IF(W.GE.WINC) BASE=BASE+1
    LETTERS(IJKL)=BASECHAR
1732    CONTINUE

        WRITE(3,7) (LETTERS(JH),JH=1,NUMOUT)
    CLOSE(3)
    CLOSE(4)
    OPEN(4,FILE='NET.1H')
    OPEN(3,FILE='WEIGHT1.1H')
C      Get number of learning passes required
        READ(4,211) ITERATIONS1
C      Get Number Of Inputs
        READ(4,218) NUMIN
C      Get Number Of Neurons in Hidden Layer
        READ(4,218) NUMHIDDEN1
C      Covert weight values to alphabetic character map
        DO 1231 NHIDDEN=1,NUMHIDDEN1
            NUMROW=(NHIDDEN-1)*NUMIN
            DO 1228 INLAYER=1,NUMIN
                INDEX_WEIGHT=NUMROW+INLAYER
                read(4,20) W

```

```

C          W=WEIGHT_HIDDEN1(INDEX_WEIGHT)
          BASE=IIA
          IF(W.LT.0) THEN
              BASE=NNA
              W=-W
          ENDIF
          IF (W.EQ.0.) THEN
              BASE='0'
          ELSE
              DO J=1,24
                  IF(W.GE.WINC) THEN
                      BASE=BASE+1
                      W=W-WINC
                  ENDIF
              ENDDO
          ENDIF
          IF(W.GE.WINC) BASE=BASE+1
          LETTERS(INLAYER)=BASECHAR
1228      CONTINUE
          WRITE(3,7) (LETTERS(JH),JH=1,NUMIN)
1231      CONTINUE
          CLOSE(3)
          OPEN(3,FILE='WEIGHT2.1H')
          READ(4,218) NUMOUT
C          Covert weight values to alphabetic character map
          DO 1241 NOUTLAYER=1,NUMOUT

```



```
NUMROW=(NOUTLAYER-1)*NUMHIDDEN1
DO 1227 NHIDDEN=1,NUMHIDDEN1
    INDEX_WEIGHT=NUMROW+NHIDDEN
    READ(4,20) W
C    W=WEIGHT_OUTLAYER(INDEX_WEIGHT)
    BASE=IIA
    IF(W.LT.0) THEN
        BASE=NNA
        W=-W
    ENDIF
    IF (W.EQ.0.) THEN
        BASE='0'
    ELSE
        DO J=1,24
            IF(W.GE.WINC) THEN
                BASE=BASE+1
                W=W-WINC
            ENDIF
        ENDDO
    ENDIF
    IF(W.GE.WINC) BASE=BASE+1
    LETTERS(NHIDDEN)=BASECHAR
1227    CONTINUE
    WRITE(3,7) (LETTERS(JH),JH=1,NUMHIDDEN1)
1241    CONTINUE
    CLOSE(3)
```

```

20      FORMAT(F16.10)
      READ(4,219) dummy_read
      READ(4,20) (ERROR(I),I=1,NUMOUT)
      OPEN(3,FILE='BIAS1.1H')
      READ(4,219) dummy_read
C      Covert weight values to alphabetic character map
      READ(4,20) (BIAS_HIDDEN1(J),J=1,NUMHIDDEN1)
      DO 1733 IJKL=1,NUMHIDDEN1
          W=BIAS_HIDDEN1(IJKL)
          BASE=IIA
          IF(W.LT.0) THEN
              BASE=NNA
              W=-W
          ENDIF
          IF (W.EQ.0.) THEN
              BASE='0'
          ELSE
              DO J=1,24
                  IF(W.GE.WINC) THEN
                      BASE=BASE+1
                      W=W-WINC
                  ENDIF
              ENDDO
          ENDIF
          IF(W.GE.WINC) BASE=BASE+1
          LETTERS(IJKL)=BASECHAR

```

320

1733 CONTINUE

WRITE(3,7) (LETTERS(JH),JH=1,NUMHIDDEN1)

CLOSE(3)

OPEN(3,FILE='BIAS2.1H')

READ(4,219) dummy_read

READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)

C Covert weight values to alphabetic character map

DO 1734 IJKL=1,NUMOUT

W=BIAS_OUTLAYER(IJKL)

BASE=IIA

IF(W.LT.0) THEN

BASE=NNA

W=-W

ENDIF

IF (W.EQ.0.) THEN

BASE='C'

ELSE

DO J=1,24

IF(W.GE.WINC) THEN

BASE=BASE+1

W=W-WINC

ENDIF

ENDDO

ENDIF

IF(W.GE.WINC) BASE=BASE+1

LETTERS(IJKL)=BASECHAR

```
1734      CONTINUE

      WRITE(3,7) (LETTERS(JH),JH=1,NUMOUT)

      CLOSE(4)

      CLOSE(3)

      OPEN(3,FILE='WEIGHT1.0H')

      OPEN(4,FILE='NET.0H')

C      Get number of learning passes required

      READ(4,211) ITERATIONS0

C      Get Number Of Inputs

      READ(4,218) NUMIN

C      Get Number Of Outputs

      READ(4,218) NUMOUT

C      Covert weight values to alphabetic character map

      DO 241 NOUTLAYER=1,NUMOUT

        NUMROW=(NOUTLAYER-1)*NUMIN

        DO 228 INLAYER=1,NUMIN

          INDEX_WEIGHT=NUMROW+INLAYER

          READ(4,20) W

C          W=WEIGHT_OUTLAYER(INDEX_WEIGHT)

          BASE=IIA

          IF(W.LT.0) THEN

            BASE=NNA

            W=-W

          ENDIF

          IF (W.EQ.0.) THEN

            BASE='0'
```

```

ELSE
    DO J=1,24
        IF(W.GE.WINC) THEN
            BASE=BASE+1
            W=W-WINC
        ENDIF
    ENDDO
ENDIF

IF(W.GE.WINC) BASE=BASE+1
LETTERS(INLAYER)=BASECHAR
228     CONTINUE
        WRITE(3,7) (LETTERS(JH),JH=1,NUMIN)
241     CONTINUE
        CLOSE(3)
        READ(4,219) dummy_read
        READ(4,20) (ERROR(I),I=1,NUMOUT)
        OPEN(3,FILE='BIAS1.0H')
        READ(4,219) dummy_read
C      Covert weight values to alphabetic character map
        READ(4,20) (BIAS_OUTLAYER(J),J=1,NUMOUT)
        DO 1735 IJKL=1,NUMOUT
            W=BIAS_OUTLAYER(IJKL)
            BASE=IIA
            IF(W.LT.0) THEN
                BASE=NNA
                W=-W

```

```

ENDIF
IF (W.EQ.0.) THEN
    BASE='0'
ELSE
    DO J=1,24
        IF(W.GE.WINC) THEN
            BASE=BASE+1
            W=W-WINC
        ENDIF
    ENDDO
ENDIF
IF(W.GE.WINC) BASE=BASE+1
LETTERS(IJKL)=BASECHAR
1735 CONTINUE
    WRITE(3,7) (LETTERS(JH),JH=1,NUMOUT)
CLOSE(3)
CLOSE(4)
C    Write our error rates for 0, 1, & 2 hidden layers
OPEN(3,FILE='RESULTS')
    WRITE(3,900) 2,'s',ITERATIONS2,','
    WRITE(3,900) 1,' ',ITERATIONS1,','
    WRITE(3,900) 0,'s',ITERATIONS0,','
900    FORMAT(' ',I1,' hidden layer',A1,' took',I6,
1      ' learning iterations',A1)
C    Initialize case counts to zero
CALL INIT_CALC(ERRMAX,ERRMIN,

```

324

```
1  ERRTOT,NUMCORRECT,NUMINCORRECT,
2  NUMTOTAL,NUMUNIDENTIFIED)
    OPEN(4,FILE='LEARNCAS.2H')
    READ(4,25) keydep
25  FORMAT(A1)
    I=0
4235  READ(4,26,ERR=1039,END=1039) ii
    i=i+1
26  FORMAT(A1)
    ioutcase=(i-1)*numout
    READ(4,27,err=1039,end=1039)
1    (case_out(ioutcase+j),
1    OUT(J),ERROR(J),J=1,NUMOUT)
C    Calculate and write error percentages
    CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
1  ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
2  PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)
27  FORMAT(3F10.6)
    go to 4235
1039 NUMCASES=I
    CLOSE(4)
    WRITE(3,90) 2,NUMTOTAL,' learn',NUMCORRECT,
1  NUMINCORRECT,
1  NUMUNIDENTIFIED,PERCENT_CORRECT,
2  PERCENT_INCORRECT,ERRAVG,
2  ERRMAX,ERRMIN
```

```

90  FORMAT(' For',I2,' Hidden layers and',I4,A6,' cases: '/
1  I7,' were correctly classified'/
1  I7,' were incorrectly classified'/
1  I7,' were not classified'/
1  f10.2,' % were correctly classified'/
1  f10.2,' % were incorrectly classified'/
1  f13.7,' was the average amount of uncertainty'/
1  f13.7,' was the maximum amount of uncertainty'/
1  f13.7,' was the minimum amount of uncertainty')

C    Initialize case counts to zero

      CALL INIT_CALC(ERRMAX,ERRMIN,
1  ERRTOT,NUMCORRECT,NUMINCORRECT,
2  NUMTOTAL,NUMUNIDENTIFIED)

      OPEN(4,FILE='LEARNCAS.1H')

      READ(4,25,end=1040,err=1040) keydep

      I=0

4435      READ(4,26,end=1040,err=1040) II

          i=i+1

          ioutcase=(i-1)*numout

          READ(4,27,end=1040,err=1040)

2          (case_out(ioutcase+j),OUT(J),
1          ERROR(J),J=1,NUMOUT)

C    Calculate and write error percentages

      CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
1  ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
2  PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)

```



```

        go to 4435
1040    NUMCASES=I
        CLOSE(4)
        WRITE(3,90) 1,NUMTOTAL,' learn',NUMCORRECT,
1      NUMINCORRECT,
1      NUMUNIDENTIFIED,PERCENT_CORRECT,
1      PERCENT_INCORRECT,ERRAVG,
2      ERRMAX,ERRMIN
C      Initialize case counts to zero
        CALL INIT_CALC(ERRMAX,ERRMIN,
1 ERRTOT,NUMCORRECT,NUMINCORRECT,
2 NUMTOTAL,NUMUNIDENTIFIED)
        OPEN(4,FILE='LEARNCAS.OH')
        READ(4,25) keydep
        i=0
4135    READ(4,26,err=40,end=40) II
        i=i+1
        ioutcase=(i-1)*numout
        READ(4,27,err=40,end=40)
1      (case_out(ioutcase+j),OUT(J),
1      ERROR(J),J=1,NUMOUT)
C      Calculate and write error percentages
        CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
1 ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
2 PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)
        go to 4135

```

```

40      NUMCASES=I
        CLOSE(4)
        WRITE(3,90) 0,NUMTOTAL,' learn',NUMCORRECT,
1      NUMINCORRECT,
1      NUMUNIDENTIFIED,PERCENT_CORRECT,
1      PERCENT_INCORRECT,ERRAVG,
2      ERRMAX,ERRMIN
C      Initialize case counts to zero
        CALL INIT_CALC(ERRMAX,ERRMIN,
1      ERRTOT,NUMCORRECT,NUMINCORRECT,
1      NUMTOTAL,NUMUNIDENTIFIED)
        OPEN(4,FILE='TESTCASE.2H')
        READ(4,25) keydep
        I=0
2762     READ(4,26,err=2340,end=2340) II
        i=i+1
        ioutcase=(i-1)*numout
        READ(4,27,err=2340,end=2340)
1      (case_out(ioutcase+j),OUT(J),
1      ERROR(J),J=1,NUMOUT)
C      Calculate and write error percentages
        CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
1      ERN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
2      PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)
        goto 2762
2340    CLOSE(4)

```

328

```
      numcases=i
      WRITE(3,90) 2,NUMTOTAL,
1    ' test ',NUMCORRECT,NUMINCORRECT,
1  NUMUNIDENTIFIED,PERCENT_CORRECT,
1  PERCENT_INCORRECT,ERRAVG,
1  ERRMAX,ERRMIN
C
C    1 Hidden Layer
C
C    Initialize case counts to zero
      CALL INIT_CALC(ERRMAX,ERRMIN,
1  ERRTOT,NUMCORRECT,NUMINCORRECT,
1  NUMTOTAL,NUMUNIDENTIFIED)
      OPEN(4,FILE='TESTCASE.1H')
      i=0
      READ(4,25) keydep
2835    READ(4,26,err=1340,end=1340) II
      i=i+1
      ioutcase=(i-1)*numout
      READ(4,27,err=1340,end=1340)
1    (case_out(ioutcase+j),OUT(J),
1    ERROR(J),J=1,NUMOUT)
C    Calculate and write error percentages
      CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
1  ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
2  PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)
```

```

        go to 2835
1340    NUMCASES=I
        CLOSE(4)
        WRITE(3,90) 1,NUMTOTAL,
1      ' test ',NUMCORRECT,NUMINCORRECT,
1 NUMUNIDENTIFIED,PERCENT_CORRECT,
1 PERCENT_INCORRECT,ERRAVG,
1 ERRMAX,ERRMIN
C
C    0 Hidden layers
C
C    Initialize case counts to zero
        CALL INIT_CALC(ERRMAX,ERRMIN,
1 ERRTOT,NUMCORRECT,NUMINCORRECT,
1 NUMTOTAL,NUMUNIDENTIFIED)
        OPEN(4,FILE='TESTCASE.0H')
        READ(4,25) keydep
        i=0
2935    READ(4,26,err=340,end=340) II
        i=i+1
        ioutcase=(i-1)*numout
        READ(4,27,end=340,err=340)
1      (case_out(ioutcase+j),OUT(J),
1      ERROR(J),J=1,NUMOUT)
C    Calculate and write error percentages
        CALL ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,

```

330

```
1 ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,  
2 PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)  
    go to 2935
```

340 NUMCASES=I

```
    CLOSE(4)
```

```
    WRITE(3,90) 0,NUMTOTAL,
```

```
1  ' test ',NUMCORRECT,NUMINCORRECT,
```

```
1  NUMUNIDENTIFIED,PERCENT_CORRECT,
```

```
1  PERCENT_INCORRECT,ERRAVG,
```

```
1  ERRMAX,ERRMIN
```

```
    CLOSE(3)
```

999 stop

```
    END
```

C

C Subroutine to do error calculations

C

```
    SUBROUTINE ERROR_CALC(ERRMAX,ERRMIN,NUMOUT,ERROR,ERRX,
```

```
1  ERRN,ERRTOT,ERRAVG,NUMCORRECT,NUMINCORRECT,NUMTOTAL,
```

```
2  PERCENT_CORRECT,PERCENT_INCORRECT,NUMUNIDENTIFIED)
```

```
    REAL ERROR(1)
```

```
    ERRX=-1.0E10
```

```
    ERRN=1.0E10
```

```
    ERRT=0
```

```
    DO 2 J=1,NUMOUT
```

C Find minimum & maximum errors this case

```
    ERRNX=ERRX
```

```

ER=ABS (ERROR(J) )
IF (ER.LT.ERRN) ERRN=ER
IF (ER.GT.ERRX) ERRX=ER
ERRT=ERRT+ER
2  CONTINUE
NUMTOTAL=NUMTOTAL+1
C  Increment case counts
IF (ERRX.LT.0.5) THEN
C      ***** CORRECT *****
ERRTOT=ERRTOT+ERRT/NUMOUT
NUMCORRECT=NUMCORRECT+1
IF (ERRMAX.LT.ERRX) ERRMAX=ERRX
IF (ERRMIN.GT.ERRN) ERRMIN=ERRN
ERRAVG=ERRTOT/NUMCORRECT
ELSE
C      Use following statement if more than 2 outputs
IF (ERRN.LT.0.5.AND.ERRNX.GT.0.5) THEN
C      Use following statement if only 2 outputs
C      IF (ERRN.LT.0.5) THEN
C      ***** UNIDENTIFIED *****
NUMUNIDENTIFIED=NUMUNIDENTIFIED+1
ELSE
C      ***** INCORRECT *****
NUMINCORRECT=NUMINCORRECT+1
ENDIF
ENDIF

```

```
T=NUMTOTAL  
PERCENT_CORRECT=100.*NUMCORRECT/T  
PERCENT_INCORRECT=100.*NUMINCORRECT/T  
RETURN  
END  
SUBROUTINE INIT_CALC(ERRMAX,ERRMIN,  
1 ERRTOT,NUMCORRECT,NUMINCORRECT,  
1 NUMTOTAL,NUMUNIDENTIFIED)  
ERRMAX=-1.0E10  
ERRMIN=1.0E10  
NUMCORRECT=0  
NUMINCORRECT=0  
NUMUNIDENTIFIED=0  
NUMTOTAL=0  
ERRTOT=0.  
RETURN  
END
```

REFERENCES

1. Mark Jurik, Neuro Tapes, a 16 hour video-cassette tutorial on Neural Networks, 1988.
2. R. Paul Gorman and Terrence J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", pages 75-89, Neural Networks, Vol. 1, No. 1, 1988.
3. Ralph Linsker, "Self-Organization in a Perceptual Network", pages 105-117, Computer, March 1988.
4. Introduction to Neural Networks, California Scientific Software, August 1988.
5. D. J. Bakker, "Density Measurements in Conjunction with Echosounding," The Hydrographic Journal No. 14, April 1979.
6. R. Kirby, W. R. Parker, and W. H. A. van Oostrum, "Definition of the Seabed in Navigation Routes through Mud Areas", paper presented at International Hydrographic Technical Conference, Ottawa, May 1979.
7. D. D. Caulfield and Yung-Chang Yim, "Prediction of Shallow Sub-Bottom Sediment Acoustic Impedance while Estimating Absorption and Other Losses", pages 44-50, Journal of the Canadian Society of Exploration Geophysicists, Vol. 19, No. 1, December 1983.
8. J. Barry Shackelford, "Neural Data Structures: Programming with Neurons", pages 69-78, Hewlett-Packard Journal, June 1989.
9. David S. Touretzky and Dean A. Pomerleau, "What's Hidden in the Hidden Layers?", pages 227-233, Byte, August 1989.
10. "Silt Density Measurement," The Dock and Harbour Authority, January 1981, Westminster Dredging Company Limited.
11. Philip D. Wasserman, Neural Computing: Theory and Practice, Van Nostrand Reinhold, 1989.
12. David E. Rumelhart, James L. McClelland, and the PDP Research Group, Parallel Distributed Processing, Volume 1, MIT Press, 1986.
13. Bernard Widrow and Rodney Winter, "Neural Nets for

Adaptive Filtering and Adaptive Pattern Recognition," pages 25-39, Computer, March 1988.

14. Bernard Widrow and Samuel D. Stearns, Adaptive Signal Processing, Prentice Hall, 1985.

15. Richard P. Lippmann, "An Introduction to Computing with Neural Nets," pages 36-54, Artificial Neural Networks: Theoretical Concepts, The Computer Society of IEEE, 1988.

16. D. D. Caulfield, Acoustic Core System DE-IB-100 Manual Version 4.0, Caulfield Engineering Group, June 1990.

17. Barry W. McCleave and E. Dale Hart, "Small-Boat Survey Systems: Technical Report HL-89-21," pages A2-A3, USAE Waterways Experiment Station.

18. George Downing, Unpublished notes of George Downing relating results of research by Downing, E. G. McLeroy, and A. Deloach under Army Order No. 73-49 "Particle Suspension Acoustics," May 1974, USAE Waterways Experiment Station.

19. A. E. Ingham, Hydrography for the Surveyor and Engineer, pages 67-93, John Wiley and Sons, 1984.

20. Dave Caulfield, Acoustic Core System CE-IB-100 Manual, Caulfield Engineering Group, September, 1989.

21. R. Colin Johnson, "Intel Goes Soft with Neural Nets," Electronic Engineering Times, May 14, 1990.

22. Tarun Khanna, Foundations of Neural Networks, Addison-Wesley Publishing Company, 1990.

23. Christoph von der Malsburg and Wolf Singer, "Principles of Cortical Network Organization," Organization of Neural Networks: Structures and Models, edited by W. von Seelen, G. Shaw, and U. M. Leinhos, Verlagsgesellschaft mbh Publishers, Weinheim (Federal Republic of Germany), 1988, pages 109-126.

24. Marvin Minsky and Seymour Papert, Perceptrons, The MIT Press, 1969.

25. Tom Manuel, "Are Artificial Neural Networks Finally Ready For Market?", Electronics, pages 85-88, August 1988.

26. Judith Dayoff, Neural Network Architectures, Van Nostrand Reinhold, 1990.

27. John J. Hopfield and David W. Tank, "Computing with Neural Circuits: A Model", Science, Volume 233, pages 625-633, August, 1986.

28. Gail C. Carpenter and Stephen Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," IEEE Computer, pages 77-88, March, 1988.
29. Robert Hecht-Nielsen, "Counterpropagation Networks," IEEE First International Conference On Neural Networks, Volume II, pages 19-32, 1987.
30. Bernard Widrow, "ADELINE and MADELINE-1963," IEEE First International Conference On Neural Networks, Volume 1, pages 143-157, 1987.
31. Bernard Widrow, Captain Rodney G. Winter, and Robert A. Baxter, "Learning Phenomena in Layered Neural Networks," IEEE First International Conference On Neural Networks, Volume II, pages 411-429, 1987.
32. David B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," IEEE First International Conference On Neural Networks, Volume II, pages 593-600, 1987.
33. Edward Denning Dahl, "Neural Network Algorithm for and NP-Complete Problem: Map and Graph Coloring," IEEE First International Conference On Neural Networks, Volume III, pages 113-120, 1987.
34. Kuniyiko Fukushima, Sei Miyake, and Takayuki Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition," IEEE Computer Society Press Technology Series, pages 136-144, 1988.
35. Bernard Widrow, Rodney G. Winter, and Robert A. Baxter, "Layered Neural Nets for Pattern Recognition," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 36, Number 7, pages 1109-1117, 1988.
36. Yeou-Fang Wang, Jose B. Cruz, Jr., and James H. Mulligan, Jr., "Two Coding Strategies for Bidirectional Associative Memory," IEEE Transactions on Neural Networks, Volume 1, No. 1, pages 81-92, 1990.
37. Frank Massa, "Sonar Transducers: A History," Sea Technology, pages 39-48, November, 1989.
38. Shipboard Underwater Instruments Systems, Innerspace Technology, Incorporated, 1989.
39. I. Noorany, "Underwater Soil Sampling and Testing--A State-of-the-Art Review," ASTM STP 501, American Society for Testing and Materials, pages 3-41, 1972.

40. George Shortley and Dudley Williams, Elements of Physics, Prentice-Hall, Incorporated, pages 446-449, 1965.
41. Vincent Montante, "The Nuclear Density Probe," paper submitted to World Dredging Conference, 1980.
42. Nabil A. Morgan, "Physical Properties of Marine Sediments As Related to Seismic Velocities," Geophysics, Volume 34, Number 4, pages 529-545, August, 1969.
43. Edwin L. Hamilton, "Elastic Properties of Marine Sediments," Journal of Geophysical Research, Volume 76, Number 2, pages 579-604, January, 1971.
44. Edwin L. Hamilton, "Prediction of In-Situ Acoustic and Elastic Properties of Marine Sediments," Geophysics, Volume 36, Number 2, pages 266-284, April, 1971.
45. Edwin L. Hamilton, "Compressional-Wave Attenuation in Marine Sediments," Geophysics, Volume 37, Number 4, pages 620-646, August, 1972.
46. Debasis Sengupta and Steven Kay, "Efficient Estimation of Parameters for Non-Gaussian Autoregressive Processes," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, Number 6, pages 785-789, June, 1989.
47. J. Pierre Le Cadre, "Parametric Methods for Spatial Signal Processing in the Presence of Unknown Colored Noise Fields," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, Number 7, pages 965-969, July, 1989.
48. Steven M. Kay, "Asymptotically Optimum Detection in Incompletely Characterized Non-Gaussian Noise," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, Number 5, pages 627-633, May, 1989.
49. Altan Turgut and Tokuo Yamamoto, "Synthetic Seismograms for Marine Sediments and Determination of Porosity and Permeability," Geophysics, Volume 53, Number 8, pages 1056-1067, August, 1988.
50. Evangelos E. Milios and S. Hamid Nawab, "Signal Abstractions in Signal Processing Software," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 37, Number 6, pages 913-917, June, 1989.
51. "80170NW Electrically Trainable Analog Neural Network", pages 1-40, Intel Corporation, May 1990.
52. Janet J. Barron, "Chips for the Nineties and Beyond," Byte, pages 342-350, November 1990.

53. Charlotte Adams, "GD Claims Neural Chip Breakthrough," pages 1 and 17, December 1990.

54. Noboyuki Miyazaki, "Neural Networks Go to Work in Japan," Electronic Engineering Times, pages 43 and 46, January 28, 1991.