DETECTION OF ARTIFACTS IN BATHYMETRIC SWATH DATA USING ARTIFICIAL NEURAL NETWORKS

by

John Pinto Cardoso

TR90-052, September 1990

This is an unaltered version of the author's
M.Sc.(C.S.) Thesis

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B. E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

DETECTION OF ARTIFACTS IN BATHYMETRIC SWATH DATA USING ARTIFICIAL NEURAL NETWORKS

by

John Pinto Cardoso

E.E.T., Inst. Ind. Lisbon, 1961

B.Sc.(C.S.), U.N.B., 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Computer Science

in the Faculty

of

Computer Science

This thesis is accepted.

.....

Dean of Graduate Studies and Research

THE UNIVERSITY OF NEW BRUNSWICK

September, 1990

ABSTRACT

Due to the existence of new and more effective learning algorithms, artificial neural networks (ANNs) are now being used in many different fields such as in pattern recognition and classification. A study of the feasibility of using ANNs to detect certain types of artifacts (peaks) that occur in bathymetric swath surveying and to localize user-specified areas of ocean floor, was done.

A peak detection network (PDNet) was designed and defined in terms of ANNs. This method of peak detection was labelled as the Neighborhood Averaging Method (NA). The PDNet is a parallel distributed processing and adaptive network that is able to continually monitor its inputs and detect data values outside some expected range (peaks). When some peak is detected the network is able to suggest what the "correct" value should be. Auxiliary average and variance modules (subnetworks) are defined in network terms.

The Lateral Inhibition Method (LI), also biologically inspired, was tried for peak detection. This method, implemented as a "narrow" filter, was particularly effective in detecting peaks composed of single data points.

Both the NA and the LI methods are data-profile oriented for execution speed. Both methods were tested on swath data considering different values of thresholds and neighborhood sizes. The results of these test were then visually verified using "Swath Viewer", an already existent data visualization tool. Interface software tools were created to facilitate result verification. One concludes that both the NA and LI methods are valuable methods for peak detection in swath data.

ANNs were trained using the back-propagation (BP) learning algorithm to detect areas of ocean floor containing certain user-specified features. The BP equations are described. Preliminary tests were made on an IBM PC using profile data to define simple floor types such as ramps and hills. After these tests, a two-hidden layer BP ANN was implemented and tested on swath data. Auxiliary software tools were created to facilitate learning tasks and the handling of data files. One concludes that although ANNs are computational tools with some valuable features such as generalization and learning abilities, when used in swath data applications, such as in feature detection, their usefulness is of questionable value.

Table of Contents

List of Tables and Figures

Chapter 4

Dedication

This work is dedicated to the memory of my father Horacio, for his hope and trust , and to my wife Deolinda for her love and her patience.

Acknowledgments

I want to thank Mike Mikaelian for his constant support during these many years, Brad Nickerson for his attentive and concerned supervision, Lev Goldfarb for many heated but enlightening discussions, and Colin Ware for some helpful comments.

I also must thank Ken Doucet and Hubert Newman for their help and Unix expertise, and Murray Linton for his help and "Mac-Wisdom".

Many thanks also to John Webster, Betty MaClure and Vickie MacLeod from the CPIT for letting me use their Macs.

Thank you Luis for proofreading and editing some of these texts; you did it with grace.

CHAPTER 1

Introduction

1.1 Historical Overview of Connectionism

Connectionism, a computational system that uses the brain as model, started in the early 1940s. McCulloch and Pitts' M-P neuron[2] and Hebb's Learning Law may be regarded as the foundations for subsequent artificial neural network research. In the 1970s, connectionist research almost stoped due to the negative criticism of some influential voices. In the 1980s, because of new advances in electronic hardware technology and new algorithms and network architectures, connectionism is the dominant approach to simulating intelligence.

1.1.1 Early beginnings

Back in ancient Greece, Plato (427-347 B.C.) and Aristotle (384-322 B.C.) wrote the first known theories about mind and brain processes, but to find the beginnings of contemporary neurocomputing and connectionism one has only to go back to the 1940s[1].

In 1943, W. McCulloch and W. Pitts formulated one of the first neural models: the "M-P neuron". The M-P neuron consisted of a finite number of excitatory and inhibitory weighted inputs, a threshold and an all-or-none output. McCulloch and Pitts demonstrated that their simple model could compute logical functions[2].

Later in 1949, Donald Hebb, a psychologist at McGill University, proposed an important neurophysiological postulate in his book "Organization of Behavior" (1949)[3]:

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

That postulate, known today as Hebb's Learning Law, together with the M-P neuron laid the basis for modern artificial neural network (ANN) research.

In 1951, Marvin Minsky, then a graduate student at Princeton, went to Harvard to build *Snark*, a learning machine, and to write his Ph.D. thesis on a learning related subject[4]. Minsky later joined Warren McCulloch and Oliver Selfridge at M.I.T., and in association with John McCarthy founded the Artificial Intelligence Project. McCarthy went on to Stanford to start an artificial intelligence (AI) lab, soon followed by another at the Stanford Research Lab. Around that same time, Allen Newell and Herbert Simon founded the AI lab at Carnegie-Mellon. Those were the beginnings of modern AI. This active scientific movement was based on the premise that the essence of brain processes could be captured by computer programs capable of manipulating signs and symbols. This was the computationalist movement.

Another approach to the study of brain and mind was also emerging and gaining momentum -- connectionism. Headed by Frank Rosenblatt, the inventor of the "Perceptron", a simple computational learning device, connectionism appealed to many because of its biological similarities[4].
Rosenblatt wrote in his book Principles of Neurodynamics (1962):

> Perceptrons are not intended to serve as detailed copies of any actual nervous system. They're simplified networks, designed to permit the study of lawful relationships between the organization of a nerve net, the organization of its environment, and the 'psychological' performances of which it is capable.

Rosenblatt was very critical of the computationalist idea that the processes of the mind could be captured by sign-and-symbol manipulation programs and very enthusiastic about the perceptron's "spontaneous learning" ability.

During the 1960s Marvin Minsky and Seymor Papert wrote a detailed and careful mathematical analysis of the perceptron showing shortcomings of connectionism. Because of Minsky's prominence in the AI area, this book - Perceptrons (1969)[6] - killed research grants, causing most ANN researchers to stop their efforts and abandon the connectionist field. Curiously, shortly after, Frank Rosenblatt was found dead in a boat accident which may have been a suicide[4].

Despite the negative criticism of the late sixties, some connectionist researchers continued their work, later to prove that although the analysis of

Minsky and Papert was correct for single-layer perceptrons, it did not apply to the more promising multi-layer nets using new learning algorithms.

1.1.2 Later Years

ANN research progressed through the 1970s, leading to new insights into neural net properties and competitive-learning machines. Steven Grossberg started an important body of work proposing a biological intelligence theory that would unify psychology, neurology, biophysics and computer science. J. A. Anderson promoted distributed versus local knowledge representation. David Marr and T. Poggio developed a model for stereoscopic depth perception agreeable to the connectionist philosophy. John Hopfield, a physicist at California Institute of Technology[5] , started studying the similarities and differences between biological and electronic computation. In 1982 J. A. Feldman and H. Ballard first used the term "connectionism" to define their work and to emphasize the topology and highly parallel architecture of their models. Also making important contributions to connectionism were Teuvo Kohonen, Bernard Widrow, Terrence Sejnowsky, Kunihito Fukushima, Sun-ichi Amari, Carver Mead, Leon Cooper and many other still active researchers.

In the 1980s, stimulated by significant advances in hardware and new architectures and algorithms, connectionism occupies a prominent position and it is the dominant approach to simulating intelligence. Under another name, Parallel Distributed Processing (PDP), is now engaging many brilliant researchers, such as David E. Rumelhart, James L. McClelland, Geoffrey E. Hinton and others, from many different scientific fields. In contrast to traditional AI, that models learning by using high-level procedures to manipulate symbols and signs, PDP uses the brain as a model: units to represent the neurons, connections between units to represent the axons and the dendrites and connection-weights to represent synaptic processes. PDP models, like brains, are highly parallel, fault-tolerant, and redundant.

1.2 General Observations

Living animals, especially mammals, are equipped with an intricate behavior controlling "hardware": the nervous system[7]. The nervous system provides them with the mechanisms to sense, learn and adapt to their environments. Without this specialized hardware survival would not be possible.

The human nervous system is one the most sophisticated and complex on earth. As an information processing system, it receives, processes and transmits information. One can find some loose analogies between artificial information systems, i.e. digital computers, and the human nervous system, such as memory and information storage and retrieval.

The nervous system plays three vital basic functions: input, processing and output[7] . 1) Receptor cells in the sense organs receive information from the outside world. 2) The new incoming information is processed and analyzed by the system, using its present state (the knowledge acquired and produced by previously acquired information), and some decisions are made. 3) Finally, the system sends out messages to the places where some action must take place, generally the motor cells. Neurons are needed to conduct signals between the different parts of the body.

1.2.1 The Neuron

Neurons are the building blocks of the nervous system, which is a complex information processing system that receives, processes and transmits information. External stimulation produces a change in the resting firing rate of a neuron and information is generally encoded by a frequency modulation mechanism.

Although neurons vary greatly in size and shape according to their types, they all seem to have the same basic configuration and all appear to operate in the same way.

A neuron has three distinct parts: the cell body or soma, the dendrites and the axon[7] .

The cell body holds the nucleus of the neuron which controls the cell vital functions, such as oxygen utilization and energy production. Nerve cells die, but unlike other body cells, do not reproduce.

The dendrites are fibres extending from the cell body. Their function is to receive signals or stimulation from other neurons. A neuron may have from just a few to several hundred dendrites.

The axon is typically a single long fibre extending from the cell body. Axons vary in length from less than a millimetre to several feet and end in terminal branchings called axon terminals. The electrical signals produced by the neuron travel along the axon to these end terminals, and from there, after some transformation, to the dendrites and soma of the next neuron.

The axon terminals of one neuron do not physically touch the dendrites of other neurons. The region where an axon terminal approaches a dendrite or the soma of the next neuron is called a synapse. Nerve signals must travel to the next neuron through a synaptic gap, the space between the bulb or button of the axon terminal of one cell and the dendrites or soma of the next.

1.2.2 How Neurons Work. A Simplified View

Each neuron is enclosed by a semipermeable membrane that separates the fluids on the outside of the cell from those on the inside. Both fluids contain ions of sodium, potassium and chloride[7] .

When a neuron is in its resting state, there are about ten times as many potassium ions inside the neuron as there are on the outside; conversely, the concentration of sodium ions is ten times greater on the outside than on the inside. This imbalance produces both a negative electrical charge inside the neuron and the polarization of its membrane necessary to keep the neuron in a state of readiness. This electrical polarization of the neuron is called the resting potential and its value is about -70 millivolts[7] .

Incoming signals are produced by a chemical action on specialized membrane sites of the dendrites and of the soma, altering momentarily the membrane's resting potential at those locations. These chemical actions can have either an excitatory or an inhibitory action upon the cell. An excitatory action allows sodium ions to enter the neuron causing a depolarization of the membrane, for instance from -70 mV to -60 mV. An inhibitory effect, on the other hand, allows potassium ions to escape producing a hyperpolarization, say from -70 mV to -80 mV. These potentials are called graded potentials. The larger the incoming signals, the larger the changes in the graded potentials[7] .

Thousands of incoming signals from neighbouring neurons may arrive simultaneously to the cell, some exciting it, others inhibiting it. The soma acts as an integrator and the net result is the sum of all these potentials. If the total is above a certain threshold, the membrane around the hillock of the axon is depolarized and sodium floods into the cell, causing it to fire. The resting potential of the cell collapses -- an all-or-none effect -- resulting in the formation of an action potential, a strong electrical impulse always of equal magnitude for a given cell, about +40 mV. This impulse travels along the axon with a constant speed, until it reaches the terminal buttons producing an output for other cells.

Since the amplitude of nerve impulses is always constant, neurons respond to stimulation, or encode information, by changing their rate of firing. Their output is a frequency modulated signal, with stronger stimulations producing greater frequency changes[7].

Because neurons are not in physical contact with each other and the fluid in the synaptic region acts as an insulator, the electrical nerve impulse can't travel to other cells when it reaches the axon terminals. At this time, the electrical conduction changes into a chemical action. When the action potential reaches the axon terminals, chemical transmitters, also known as neuro-transmitters, located in the vesicles of the axon terminals, are released into the synaptic gap and interact with receptor molecules on the membrane of the post-synaptic neuron, thus completing the inter-neuron communication process.

There are two types of synapses: excitatory and inhibitory. In the excitatory synapses the neuro-transmitter released causes the post-synaptic neuron to increase its firing rate; conversely, in the inhibitory synapses the firing rate of the post-synaptic neuron is reduced[7].

The time duration of an action potential is about one millisecond. After firing, it takes also about one millisecond for a nerve cell to recover, return to its resting potential, and get itself ready for further action. This interval is known as the refractory period[7] .

1.2.3 Physiological Constraints

Nerve cells die, but with the exception of some cells in the olfactory system, do not reproduce. Primates are born with a maximum number of neurons. In early infancy and as part of normal brain development a selective massive death of nerve cells occur, and between 15 and 85 percent of the original nerve cells die[8]. It is estimated that humans may loose as many as 10,000 neurons a day[7].

The human brain is estimated to contain 10^{12} to 10^{14} neurons[7] . The favoured role of the brain in relation to other body parts is demonstrated by the fact that when the body is at rest, the brain alone consumes twenty per cent of the body's oxygen supply, although it accounts only for two per cent of the body's total mass[8] .

Biological neural networks are highly and densely interconnected and each neuron has many synapses. It is estimated that some nerve cells may have up to 100,000 synapses with other cells.

Neurons are slow devices operating in the range of one to the tens of milliseconds. The speed of nerve impulses is also slow. Depending upon the type of neuron, nerve impulses travel at speeds that vary from 10 centimetres per second to 200 meters per second.

1.2.4 Modelling Neurons

The M-P neuron, formulated by McCulloch and Pitts in 1943, was one of the first artificial neurons modelled after biological nerve cells. The examination of this very simple model will serve as a basis to understand the foundations of artificial neural networks[2] .

The M-P neuron is characterized by a finite set of inputs, x_i (i=1,2,...,n), a threshold level L, and an output y. Each of these inputs has a weight, w_i, associated with it. Weights may have values of +1 or -1. If a weight equals -1 its associated input is said to be an inhibitory input. Excitatory inputs have weights of +1.

The inputs and the output of the M-P neuron are binary valued, that is, their values are either 0 or 1, and the threshold level can take any positive integer value.

The output value of the M-P neuron may be computed using the following expression[2]:

$$y = g(p) \qquad \text{where} \begin{cases} y=0 & \text{for } p<0 \\ y=1 & \text{for } p\geq 0 \end{cases} \qquad (1.2.1)$$

$$\text{and where } p = \left(\sum_{i=1}^{n} w_i x_i - L\right) \qquad (1.2.2)$$

The inputs of this model correspond to the inhibitory and excitatory synapses of a neuron. Stimulation arriving through inhibitory synapses contributes towards the hyperpolarization of the cell and will move it away from its firing point, while stimulation from excitatory synapses will depolarize the nerve cell and will bring it closer to firing.

The neuron sums or integrates all of its inputs. If the result of this integration is above a certain threshold L, the neuron will fire, and produces an output signal represented by the value y = 1; otherwise the neuron will not fire and y = 0.

Although this model is very simplified, it exhibits some computational ability. For instance, a neuron with two binary inputs X_1 and X_2, weights $W_1 = W_2 = +1$, a threshold L = 2, and an output Y, will be able to compute the AND logic function. To compute more complex functions, individual neurons may be combined into groups to produce networks that will have the necessary architecture to solve specific problems[2] .

1.3. Formalizing Connectionist Systems

To understand and work with artificial neural networks, a formal mathematical system is necessary, and the basic components of such a system must be identified.

1.3.1 A Connectionism System

Rumelhart and McCelland[9] , propose a general framework that isolates the main components of a connectionist or parallel distributed processing

system (PDP). They identify eight main components that are implicitly or explicitly present in PDP systems:

1) A set of processing units.
2) A state of activation.
3) An output function for each unit.
4) A pattern of connectivity among units.
5) A propagation rule.
6) An activation rule.
7) A learning rule.
8) A working environment.

In a connectionist system, units are used to represent some object or concept. There are two schemes of representation that one may choose from: local and distributed. In the local representation scheme, one object is represented by one single unit (one-to-one representation). In the distributed representation, one object or concept is represented by a group of units (one-to-many representation). If N is the total number of units in the system, the ith may be designated by u_i.

There are three different types of units, named according to their function: input, hidden and output. Input units receive the system inputs. Output units send the signal out of the system. Both input and output units are accessible from the outside world. Hidden units are internal units, unaccessible or "invisible" from the outside world.

Typically, the pattern of activation of a PDP system is represented at any given instant t by a vector of N real numbers, $\mathbf{a}(t)$. Each element $a_i(t)$ of this vector, stands for the activation value of unit u_i at time t. At any given time, the system's state of activation stands for whatever the system is representing at that instant.

Unit activation values may be continuous or discrete, depending on the model. If continuous, the values are generally, but not necessarily, bounded to some closed interval, say [0,1]. Discrete values are often binary values (0 and 1), usually used to indicate if the unit is active or inactive.

Associated with every unit u_i there is an output function $f_i(a_i(t))$ which maps the state of activation to an output signal $o_i(t)$. The set of all the output values of the system is then represented by $\mathbf{o}(t)$. Often, the mapping function is

some type of threshold function; other times, the mapping function is the identity function, and the output value of a unit is equal to its activation value ($a_i(t) = o_i(t)$).

The pattern of connectivity is a very important ingredient of a PDP system. It determines what each unit represents, what the system "knows", and how it responds to a given input. Usually, the pattern of connectivity of the system is specified by a set of weights. If the output of some unit j goes to some other unit i, this connection is represented by the weight w_{ij}.

Weights may be positive, negative or zero. Positive weights represent excitatory connections, negative weights identify inhibitory connections, and zero weights mean the absence of connection. The absolute value of a weight indicates the "strength" of its connection. The pattern of connectivity of the system may then be represented by the weight matrix **W**, in which each entry w_{ij} represents the strength and type of connection from unit u_j to unit u_i.

In more complex connectivity patterns -- when units do not merely combine by some additive operation, but by some more complex rule -- a given unit may receive inputs of different kinds. In these cases, it is convenient to have each type of connection represented by a different connectivity matrix $\mathbf{W}_i$.

The rule of propagation is the rule that specifies how the output vector, **o**(t), combines with the connectivity matrices to produce a net-input for each type of input into a unit. If there are different inputs of type i, into unit u_j, then net_{ij} is the net input of type i into unit j. Sometimes, the propagation rule may be complex.

The activation rule is the rule that specifies how the net inputs of each type arriving into a unit, combine with each other and with the present state of the unit, to produce the new state of activation for that unit. There is an activation function associated with the activation rule. This function may be a threshold function, in which case the net input must exceed a certain threshold value if it is to contribute to the new state of activation. The activation function may also be deterministic, stochastic or of some other type[9].

Learning is accomplished by modifying the pattern of connectivity of the system. This is usually implemented by modifying the strength of the existing connections. Rules that are frequently used are the Hebb's rule and the Delta rule[9].

The environment in which a model is to exist must be clearly specified. In PDP models the environment is represented as a time-varying stochastic

function over the space of input patterns. Assuming one can list the set of all possible inputs to the system, numbering them from 1 to M, the environment is then characterized by a set of probabilities, p_i, for i=1,..M. Because each input pattern can be represented as a vector, it is convenient to characterize those patterns with non-zero probabilities as orthogonal or linearly independent sets of vectors. Some connectionist models are restricted in the kinds of patterns that they can learn. For instance, some nets can only give the correct answer if the input vectors form an orthogonal set[9].

CHAPTER 2

Peak Detection in Swath Data, Using Biologically Inspired Mechanisms

2.1 Motivation

Despite their accuracy and efficiency, multibeam bathymetric survey systems (MBSS) may generate inaccurate data. When these data are plotted they create some typical classes of undesirable artifacts in swath charts, such as "Bull's Eyes".

"Bull's Eyes" are peak artifacts that show up in contoured swath bathymetry charts as sharp peaks. For instance, in the case of the Sea Beam MBSS, de Moustier and Kleinrock [16] have found that these peaks were caused by interferences from external sound sources.

The amplitude of "Bull's Eyes" is typically in the range of tens to hundreds of meters, and the more pronounced peaks are usually spurious, very sharp and steep. With such distinct characteristics, this type of artifact is easily recognized by an experienced chart observer, but as the amplitude of the peaks decreases it becomes harder to identify. Also, more than one peak may occur simultaneously in different parts of the profile (on one or on both sides of the ship's track). When creating charts, it is desirable to have an automated procedure to identify, and conceivably correct, these artifacts.

2.2 PDNet -- An Adaptive Network for Peak Detection

The purpose of the Peak Detector Network (PDNet) is to recognize the presence of one or more peaks in the incoming stream of bathymetric data, and identify (flag) their location. The PDNet also has the ability to correct, or at least improve, inaccurate data points. The most important feature of the PDNet is its adaptive mechanism.

The PDNet can be defined in terms of artificial neural network principles, and was inspired by the signal averaging mechanisms often performed by groups of neighboring neurons, hence the method upon which the PDNet is based is labelled here as the Neighborhood Averaging method (NA).

2.3 Network Architecture

In most cases, the nature of peak detection is simple and well defined. The outstanding characteristic of peaks is that their amplitudes are well above or below the average value of the surrounding amplitudes. The design of an original peak detector based upon some general artificial neural network principles -- PDNet -- is suggested. Because the nature of peak detection is well known, the selection of a "hard net" design seems appropriate.

"Hard net" is a label given in this document to a networks whose weights are pre-set and constant. Domain-specific knowledge -- the knowledge necessary to solve a specific and restricted type of problem (i.e. peak-detection) -- is "hard-wired" into the net by selecting an appropriate set of fixed weights.

The PDNet architecture is shown in Fig. 2.1.

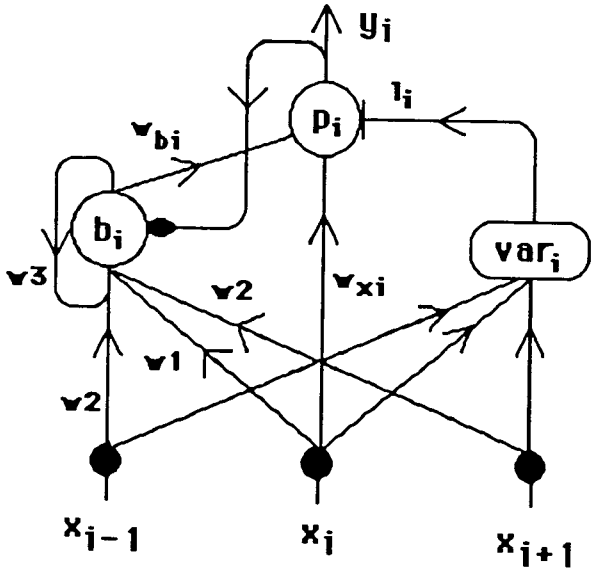


Fig. 2.1 A PDNet unit with two neighbors

The size (the number of units) of the PDNet is determined by the number of its inputs. If there are N different inputs from the environment, the PDNet will have N units and can be conveniently described in vector form. At any given instant the PDNet produces an output vector **y** in response to an input vector **x**. Each unit and its corresponding input and output may then be denoted respectively by u_i, x_i and y_i, for i = 1,..,N.

Each peak detecting unit u_i, is composed of two neurons, namely p_i, b_i, and subnet var_i. The p_i neuron is the peak-detecting neuron and produces the

binary output (y_i) for unit u_i. If unit u_i detects a peak whose magnitude is above some dynamically adjustable threshold level l_i, its p neuron will fire ($p_i = y_i = 1$), otherwise p will remain in its resting state ($p_i = y_i = 0$).

Each p neuron receives two inputs: one directly from input x_i, and the other from its biasing neuron b_i. Let the synaptic weight between neuron p_i and input x_i be called w_{xi}, and the synaptic weight between neurons p_i and b_i be called w_{bi}; then the net input of neuron p_i is

$$net_i = w_{xi} \cdot x_i + w_{bi} \cdot b_i \qquad (2.3.1)$$

$$\text{where} \quad b_i = f(g_i) \qquad (2.3.2)$$

$$\text{and} \quad g_i = 1 \quad \text{for } p_i^{(t)} = 1 \text{ and } p_i^{(t-1)} = 0 \qquad (2.3.3)$$
$$\text{else} \quad g_i = 0$$

where the value of the bias unit b_i is dependent on the variable g_i, which models the refractory period of the unit u_i, and $p_i^{(t)}$ is the output of neuron p_i at time t (see eq. (2.4.1)). The output of a p neuron may then be computed using the following activation function:

$$p_i = y_i = \begin{cases} 1 & \text{for } net_i > l_i \\ 0 & \text{for } net_i <= l_i \end{cases} \qquad (2.3.4)$$

if $w_{xi} = w_{bi} = 1$ then

$$p_i = y_i = \begin{cases} 1 & \text{for } x_i + b_i > l_i \\ 0 & \text{for } x_i + b_i <= l_i \end{cases} \qquad (2.3.5)$$

With the exception of the first and the last, each b neuron receives four inputs: one from input x_i, one from each of its neighbors x_{i-1} and x_{i+1}, and the last input from itself (the output of the neuron is fed back into itself.) Because b_1 and b_N have only one single neighbor, b_2 and b_{N-1} respectively, they must be treated differently. These connections allow each unit to "learn" and "remember" about the most recent past input activity in its surrounding neighborhood.

For every new input pattern that is presented to the PDNet, the new values of the input vector **x** will be used to update the bias b of a neuron p. The refractory period of a biological neuron is generally understood to be necessary, after firing, for a neuron to regain its resting potential. During the refractory period a neuron can't fire even if it receives excitation from another cells. In the PDNet, a mechanism in some way similar to the refractory period of a neuron also exists and indeed it plays a very important role in the operation of the net. When the p neuron fires due to the excitation it received from its inputs, its bias is not allowed to be updated by its inputs unless two consecutive peaks are detected (some time has passed since last peak was detected). The bias unit retains its former activation value for the next operational cycle.

2.4 PDNet Implementation

An artificial neural network to detect peak artifacts can be easily simulated in a digital computer. The algorithm used to implement PDNet is described below.

A functional peak detection neural network should be able to monitor a stream of incoming data, decide whether or not it contains potentially invalid data points, "flag" their location, and perhaps offer its "opinion" on the most likely correct values. PDNet has all of these capabilities. A hardware implementation of a peak detection network, in a swath survey type of application, would also have the advantage of real time computation.

As previously suggested, each unit of the PDNet monitors the environment changes in its neighborhood (neighborhood averaging). Assuming that the unit-neighborhood size is set to 3, this is done by allowing the unit's biasing neuron to "see" (to connect) both its own unit's input and the inputs of its two immediate (left and right) neighboring units. The weights of these connections will determine the degree of contribution of each of these inputs. For a unit to "remember" the activity of its own past inputs, the output of its biasing neuron must be fed back into the biasing neuron itself. As in the previous cases the weight of this connection will determine the contribution of the biasing neuron towards its own new activation value.

When a p neuron receives its excitatory (+) input x_i from the environment, it compares it with the inhibitory (-) input it receives from its biasing neuron. If the absolute value of the sum of theses two inputs is greater than the unit's

current threshold level then p will fire, generating an output (y_i) which is used to signal an observer that input x_i falls outside its expected range and may be an undesirable peak. When a p neuron fires it disallows its biasing neuron from updating. This is a very useful mechanism because it prevents the units' bias from being changed by an incorrect datum value. While preventing the bias from changing, inputs are always allowed to change their units' dynamic threshold; this allows the network to adapt, in time, to novel inputs.

Considering the above factors and assuming an input-neighborhood size of 3, the output of a bias neuron b_i (i=2,..,N-1) at time t may be defined as:

$$\begin{aligned} b_i^{(t)} = \; & w1 \cdot ((1 - g_i) \cdot x_i + g_i \cdot b_i^{(t-1)}) + \\ & w2 \cdot ((1 - g_{i-1}) \cdot x_{i-1} + g_{i-1} \cdot b_{i-1}^{(ti-1)} + \\ & w2 \cdot ((1 - g_{i+1}) \cdot x_{i+1} + \\ & w3 \cdot b_i^{(t-1)} \end{aligned} \qquad (2.4.1)$$

$$\text{with } w1 + 2 \cdot w2 + w3 = 1.0$$

where w1 is the weight of the ith input connections, w2 the weight of the neighboring input connections (i+1 and i-1), w3 the weight of the feedback connection from a bias neuron b_i into itself, and g is defined in eq. (2.3.3).

An exception has to be found for the first and last units because these units only have one single neighbor. In these particular cases:

$$\begin{aligned} b_1^{(t)} = \; & w1 \cdot ((1 - g_1) \cdot x_1 + g_1 \cdot b_1^{(t-1)}) + \\ & w4 \cdot ((1 - g_2) \cdot x_2 + g_2 \cdot b_2^{(ti-1)} + \\ & w3 \cdot b_1^{(t-1)} \end{aligned} \qquad (2.4.2)$$

$$\begin{aligned} b_N^{(t)} = \; & w1 \cdot ((1 - g_N) \cdot x_N + g_N \cdot b_N^{(t-1)}) + \\ & w4 \cdot ((1 - g_{N-1}) \cdot x_{N-1} + g_{N-1} \cdot b_{N-1}^{(ti-1)} + \\ & w3 \cdot b_N^{(t-1)} \end{aligned} \qquad (2.4.3)$$

$$\text{with } w1 + w4 + w3 = 1.0$$

where w1 and w3 are defined as above, and w4 is the weight of the connection of the unit's b neuron to its single neighbor's input. If $w4 = 2 \cdot w2$ one may

interpret this condition as if the first and last neurons would also have two (equal) neighbors.

2.4.1 Networks to Compute Variance and Average

The adaptive mechanism of the PDNet is perhaps its single most important feature. This mechanism allows each unit to adjust to its surrounding environment. This is done by allowing each unit's threshold level l_i to change with the PDNet inputs.

The value of l_i is produced by a variance net. A variance net is a network that is able to compute the statistical variance of its inputs. To define such a network is a simple task. Fig. 2.2 shows the topology of a variance network.

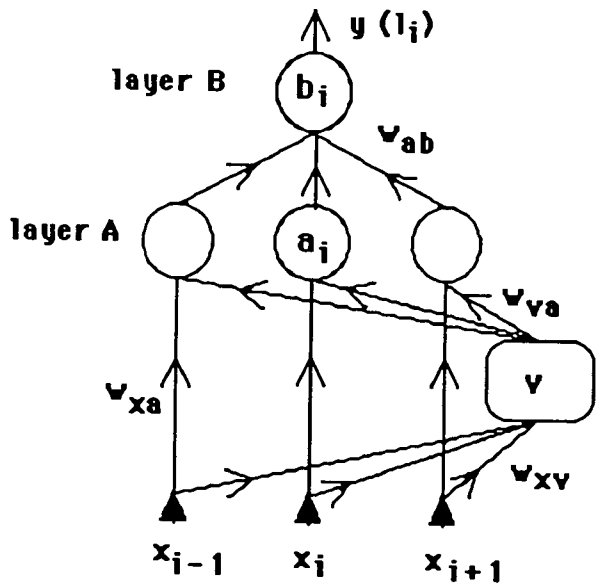


Fig. 2.2 A variance net with 3 inputs.

The statistical variance s^2 of n terms can be computed using the following mathematical expression:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \underline{x})^2}{n - 1} \qquad (2.4.4)$$

where $\underline{x}$ is the average of the n terms given by

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad (2.4.5)$$

As shown in Fig. 2.2 the variance network contains a subnet to compute the average of n terms. Such a subnet can be implemented by a single neuron v with n inputs x_i (i=1,..,n), where the weight of each input connection $w_{vx} = 1/n$, and whose activation value is defined as

$$o_v = \sum_{i=1}^{n} (\frac{1}{n} x_i) \qquad (2.4.6)$$

The variance net uses the output from its average subnet, o_v, to compute its activation value. The variance net can be topologically described as composed of two layers A and B, and a subnet v, working in a feed-forward configuration, . The first layer (layer A) is made of n neurons a_i (i=1,..,n), and the second layer (layer B) contains a single neuron b.

Each neuron in layer A receives two inputs: one from its x_i input and the other from the output of the average subnet , o_v. The connections between each input x_i and the first-layer neurons are excitatory connections with unit weights (w_{xa}= +1). The connections between the output of the average-net and the neurons of the first layer also have unit weights but they are inhibitory connections (w_{va} = -1). The net input net_a, of neurons in layer a is

$$net_{ai} = w_{xa} x_i + w_{va} o_v \qquad (2.4.7)$$

First-layer neurons produce an output o_{ai} whose value is equal to the square of their net input,

$$o_{ai} = net_{ai}^{2}. \qquad (2.4.8)$$

The connections between the single neuron of layer B (the output neuron) and the output of every neuron in layer A, w_{ab}, are excitatory with weight equal to 1/(n-1). The net input value of the output neuron b, is then

$$net_b = \sum_{i=1}^{n}(w_{ab}\, o_{ai}) = \sum_{i=1}^{n}(\frac{1}{n-1}\, o_{ai}) \qquad (2.4.9)$$

and its output y is given by the activation function

$$y = o_b = net_b \qquad (2.4.10)$$

In the context of the PDNet, the output y of a variance net provides the dynamic threshold l_i for a peak unit u_i. The above description clearly shows that certain well known computational tasks, such as the variance and average, may also be defined using the vocabulary of artificial neural networks.

2.4.2 PDNet Algorithm

Fig. 2.3 shows the pseudo-code algorithm that may be used to implement a peak detector network. Notice that in this particular implementation the unit-neighborhood size is 3. The algorithm can easily be modified to accommodate other unit-neighborhood sizes.

- Initialize network variables:
  - set fixed-weights w1,w2,w3,w4
  - get first set of inputs **x**
  - set each p_i to 0 (resting state)
  - set each $p_i^{(t-1)}$ to 0 (no peak at last cycle)
  - set each unit $bias_i^{(t-1)} = bias_i^{(t)} = x_i$
  - select a constant threshold value T
  - set each unit's threshold l_i to T

- Repeat until finished:
  - compute p units' output y
  - compute new biases $b_i^{(t)}$
  - compute the new firing threshold value l_i for each unit
  - get next input **x**.

Fig. 2.3 The PDNet Algorithm.

2.5 Using Lateral Inhibition Mechanisms for Peak Detection in Swath Data

Computer models of other biologically inspired mechanisms can be used successfully to detect artifacts such as peaks (or valleys) in swath bathymetric data.

Biological systems are generally very good at performing their particular tasks. Many centuries of evolution have tuned them for the efficiency and speed needed for survival. The visual system of vertebrates is a good example of a very useful and efficient biological mechanism.

Computer vision has many times based its methods in natural mechanisms aiming to develop better and more efficient computational methods.

David Marr's work on computational vision, such as his study of the architecture of low-level vision (Primal Sketch)[15], is well known, and although some of his work is controversial today, it is also of much value because it raises many important questions. Marr's Primal Sketch is a two-dimensional representation of the grey-level changes in the image. Light reflected from outside world objects is received by the retina. The grey-level is the illuminance of some arbitrary area of the retina at a particular time. According to Marr, the representation of an object can subsequently be obtained using a finite alphabet of primitive symbols such as "edge", "bar" or "corner". Some of the procedures used for edge-detection in the visual system, seem particularly well suited to use in the detection of isolated or single peaks in bathymetric swath data (profile structured data, see 2.5.1).

Natural visual systems have mechanisms that makes them particularly good at responding to sudden grey-level changes. Mechanisms such as lateral inhibition (LI), are used in the retina to further enhance grey-level changes between neighboring neurons, allowing the nervous system to respond more efficiently to these changes. This sort of mechanism is also known to exist in many parallel networks of neurons to enhance contrast and spread information across neural networks. It has been found that some cells in the visual cortex respond particularly well to some features of the environment making them particularly good "edge", "length", "angle" or movement detectors[15].

2.5.1 Peak Detection in Swath Data

Swath data is organized in "profiles". A profile is a given number of data points usually collected simultaneously. For systems such as the *Navitronics* a profile is generally composed of 33 data points, and for practical purposes the data is stored and organized in the same manner.

Fig. 2.4 shows a typical example of a "profile" of depth data points defining the side view of a particular portion of ocean floor at some location.

Fig. 2.4 An example of a profile of ocean floor.

The same graph could as well be interpreted as representing the grey-level changes of some image falling upon a row of receptors in the retina. The analogy of these representations suggests that one can perhaps use the same types of computational principles to study both of these areas.

The nature of peak detection has already been briefly discussed (see 2.1). Peak artifacts are generally isolated and caused by erroneous data. Peak detection may be easily done by measuring the gradients of the slopes and the changes in these gradients. Mathematically, the gradient of a point on a curve can be determined by computing the first derivative of the curve at that point, and the second derivative will compute the change of gradients.

In practice, when finding edges, one can use simple computational methods to determine gradients and gradient changes. The measure of the gradient at some point can simply be determined by subtracting the depths of its two nearest neighbors (left and right neighbors). One can apply the spatial operator (template) -1/+1 centered at the point. To apply this operator the value of the point's left neighbor is multiplied by -1 and the value of its right neighbor is multiplied by +1. The gradient of the slope at the point is the sum of these two

products. The convolution of this template with the curve will determine the gradients for all points in the curve.

When applying a spatial operator such as -1/+1, one must understand the local effects of the operator. Each component of the operator can be applyed to a single point (small operator) or one may consider the average depth of a neighborhood as a substitute (large operator). Large operators are effective for shallow smooth ramps, while small operators are good at determining sharp and narrow changes in the slope.

While the -1/+1 operator is effective in determining gradients, the -1/+2/-1 operator is used to compute changes in gradients. It is interesting to notice that biological visual systems seem to be equipped with both "on-center unit" , the equivalent to the -1/+2/-1 operator, and "off-center unit", the equivalent of the +1/-2/+1 operator[12]. Fig. 2.5 shows the effects of the application of these operators.

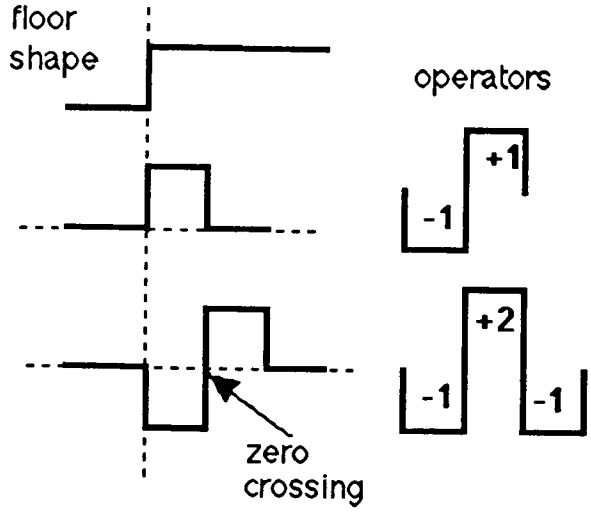


Fig. 2.5 Results of applying the -1/+1 and -1/+2/-1 spatial operators to an elevation change.

The results of applying the -1/+2/-1 spatial operator results in a zero crossing curve which can be used to indicate the detection of a peak in swath data. The implementation of an equivalent method is done by the function *edge()* which code is shown in appendix B.

2.5.2 Advantages and Disadvantages of Zero Crossing Methods

Zero crossing (ZC) techniques that use the type of operators mentioned above offer some advantages for peak detection. They are easy to implement, have fast execution, and under certain conditions are complete processes, that is, given the convolution profile it is possible to recover the original profile (i.e. for filter bandwidths of about 1 octave it is often possible to reconstitute original data with all the important information preserved[15].)

There are certain characteristics of these methods that must be considered when these operators are used. For instance, although zero crossing techniques are well suited for detecting narrow or single-point peaks, they are very poor at detecting wide peaks (peaks made of a group of contiguous data points.) In such cases, only the leading and trailing edges are detected.

One must also consider the fact that the selection of a threshold value is necessary in determining when a feature must be accepted or rejected. Global thresholds are generally very poor performers; one must consider, at least, selecting a threshold value for each individual profile.

Another factor to have in mind is the size of the operator. A conflict arises here: to detect gentle slopes one must use larger operator sizes, but large operators are ineffective in detecting sharp glitches that may occur in the slope. A sharp operator, with the unit size, was chosen for this implementation.

2.6 Detecting Peaks in Swath Data

Both the PDNet, based on the neighborhood averaging (NA) method, and the ZC Detector, based on the lateral inhibition (LI) method, were tested for execution times and number of rejected points on real bathymetric data from the Louisbourg sw1988000230.plt data file. The results of these tests are discussed below.

2.6.1 The Neighborhood Averaging Method

The peak detection capability of SWANN -- SWANN is the name of the software implementation (see appendix A) and the abbreviation for Swath Artificial Neural Networks -- was tested using the bathymetric swath data file sw1988000230.plt. This file contains 119889 data points organized in 3633 profiles of 33 data points each, and it was chosen to test the PDNet because it had already been visually checked by human operators and some 'bad' data

points had escaped detection. It was suggested that it would therefore be a good data set to test the peak detection ability of the systems.

Both in the Neighborhood Averaging (NA) and in Lateral Inhibition (LI) methods, the number of data points rejected as peak artifacts depended primarily on the values chosen for system thresholds. These values are unit independent, that is, they are always expressed in whatever depth-units the data file uses (generally centimeters). In these tests, the values chosen for the thresholds ranged from 20 to 100 depth-units.

Both the PDNet and the ZC Detector are profile oriented systems. They check every point in each data profile. If the depth-value of a point is found to fall outside its expected range, then both the PDNet and the ZC Detector call upon a traditional "average/standard-deviation" procedure to check the point more thoroughly, considering all points within some previously specified neighborhood. The size of this neighborhood (NS) is specified by the user. This last checking process is the mechanism that finally rejects or accepts the point. Pre-testing each point, contributes towards greater execution speeds.

The NS parameter defines a square matrix centered on the point in question. Boundary conditions are resolved by clipping the size of the matrix and considering only the points located within bounds. For instance, for the first point of the first profile, and a neighborhood size of 5 (defining a matrix of 25 points), only 8 points (the point is excluded) contribute towards computing the "expected" value.

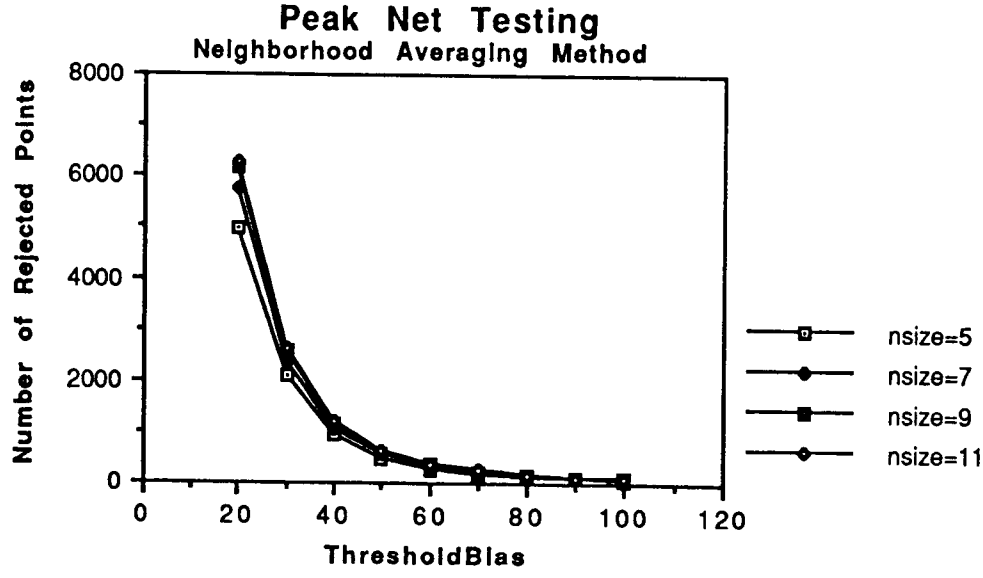
Tables 2.6.1 and 2.6.2 list the number of rejected data points and the execution times, in seconds, obtained for the tests run on the PDNet (NA method) and on the ZC Detector (LI method), for Threshold Bias (TBias) or Threshold values between 20 and 100 depth-units (in this particular case centimeters), and for Neighborhood Sizes (NS) varying from 5 to 11 data points (square-area neighborhoods containing 25 to 121 points).

Table 2.6.1

Neighborhood Averaging Method

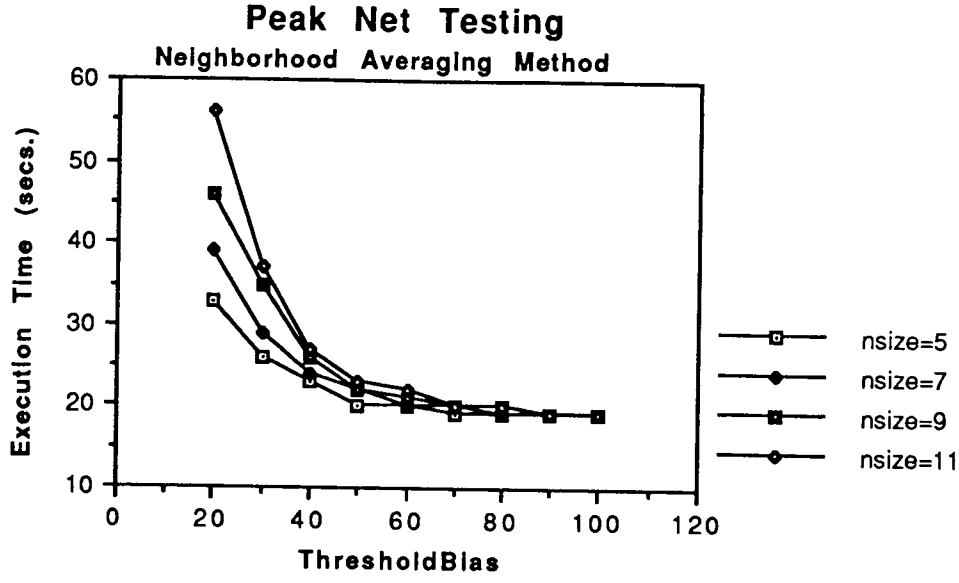
TBias	100	90	80	70	60	50	40	30	20
NS = 5									
Rejected	69	81	109	175	277	481	948	2111	4970
Time	19	19	19	19	20	20	23	26	33
NS = 7									
Rejected	84	97	143	214	338	556	1046	2366	5777
Time	19	19	19	20	20	22	24	29	39
NS = 9									
Rejected	88	106	149	228	367	595	1160	2566	6177
Time	19	19	20	20	21	22	26	35	46
NS = 11									
Rejected	98	118	162	239	388	620	1223	2620	6267
Time	19	19	20	20	22	23	27	37	56

Using the values from table 2.6.1, graph 2.6.1 shows the curves obtained by plotting the number of rejected points as a function of the threshold bias and the neighborhood size.



Graph 2.6.1 Results of testing the PDNet, show number of data points rejected by the PDNet as a function of the Threshold Bias value and the Neighborhood Size.

Note the nearly exponential curves. The 'bad' data points contained in the file and mentioned above were among the 69 points caught in the first performed test, for TBias = 100 and a Neighborhood Size = 5 units (square of 5x5 = 25 points). The PDNet took 19 seconds to perform the test. The execution times of the PDNet for the tests on the sw1988000230.plt file are shown in graph 2.6.2



Graph 2.6.2 Results of testing the PDNet. The curves show the execution times of the PDNet as a function of the Threshold Bias value and the Neighborhood Size.

In the worst condition tested, TBias = 20 and a Neighborhood Size = 11, it took the PDNet 56 seconds to reject 6267 points.

2.6.2 The Lateral Inhibition Method

Similar tests were performed using the LI method (ZC detector) of peak detection. As observed before, due to its nature, this biologically inspired process is particularly efficient at detecting isolated peaks or gradient-changes in the data.

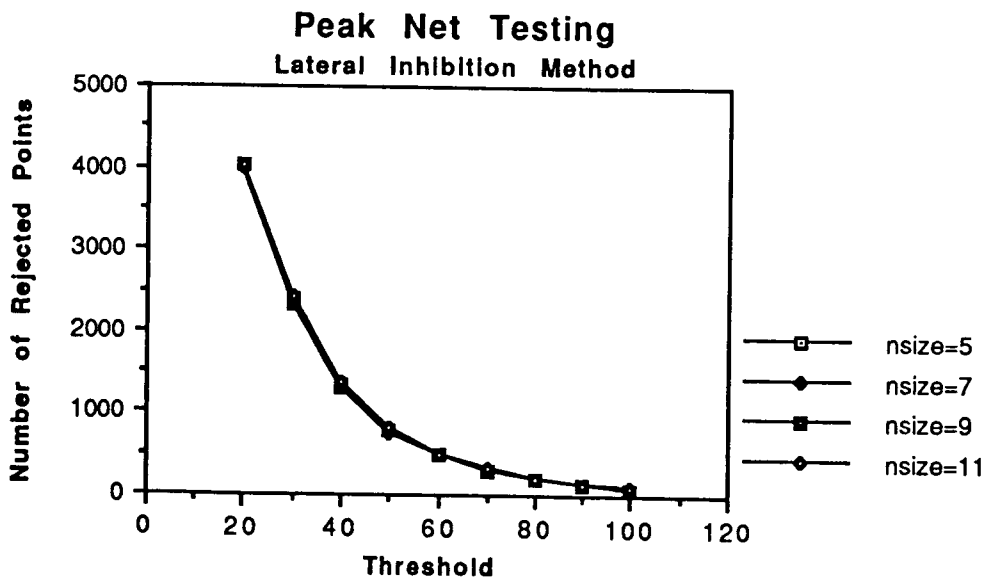
Table 2.6.2 lists the results of the tests using ZC (LI method) procedures, showing both the number of rejected points and the execution times obtained also as a function of the neighborhood size and the threshold of the network.

Table 2.6.2

Lateral Inhibition Method

Thrshld	100	90	80	70	60	50	40	30	20
NS = 5									
Rejected	81	132	202	307	488	770	1321	2309	4022
Time	4	4	4	5	6	6	8	11	13
NS = 7									
Rejected	76	129	197	297	477	765	1312	2312	3985
Time	4	5	5	6	7	7	9	12	16
NS = 9									
Rejected	80	130	198	307	484	777	1322	2380	4026
Time	5	5	6	6	8	9	11	15	19
NS = 11									
Rejected	87	133	211	313	498	804	1387	2431	4025
Time	4	5	6	7	8	11	14	18	24

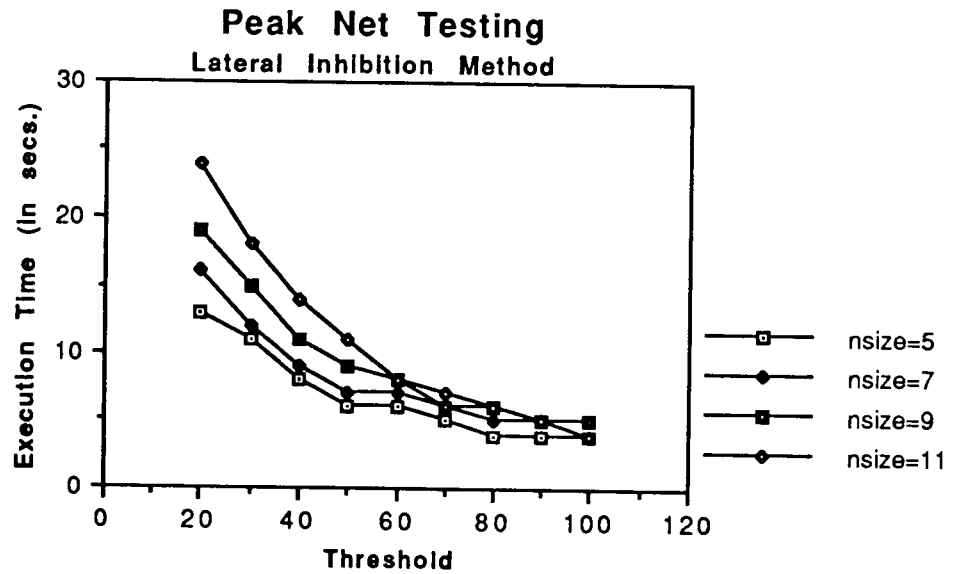
The curves obtained from the number of rejected points by the LI method are displayed in the graph 2.6.3 below.



Graph 2.6.3 Results of Lateral Inhibition method of peak detection. The curves show the number of rejected points as a function of the Threshold value and the Neighborhood Size.

Note the similarity between the four curves shown in graph 6.2.3. The number of rejected points by the LI method seem to be, for most practical purposes, independent of the neighborhood-size values used .

The test results of the execution-time as a function of the threshold and the neighborhood size are shown in graph 2.6.4. In the worst tested condition, for a threshold of 20 and a neighborhood size of 11, the execution time of 24 seconds is six times greater than the best case (4 seconds) for a threshold of 100 and neighborhood size of 5.



Graph 2.6.4 Results of Lateral Inhibition method of peak detection. The curves show the execution times as a function of the Threshold value and the Neighborhood Size.

All the above tests were performed on a SUN 4/150, running under the SunView environment.

Good practical results were also obtained when the two methods were combined. This was done by using the output file produced by the NA method as the input file for the LI method. The result is equivalent to ORing the meaningful bit (bit 7) of the status bytes produced by both tests.

The points rejected, by both methods, were visually evaluated with the aid of data visualization tools, to see if their rejection was warranted; this was confirmed.

Lower threshold-biases led to points close to the neighborhood average being rejected. These biases represent a limit within which no points are rejected if the variance of the surrounding points (determined by the neighborhood size) is zero. This could be matched to the resolution of depth measurements.

An attempt was made to compare the results obtained by the PDNet to those obtained by other methods, on a Louisbourg dataset with 170,000 depth points. Initially this was thought possible by comparing the response of the PDNet with bit 4 of the status byte associated with each depth data point (the Smith data processing software uses bit 4 to indicate "depth deleted" conditions).

Counters were set to register the number of times that the results of the PDNet agreed or disagreed with the setting of bit 4. For a Tbias = 27, the results agreed on 156,648 and disagreed on 13,352 points, a ratio of approximately 8.5%.

Unfortunately, it was found, this comparison was not possible since depth points may be flagged "deleted" by reasons other than the point being a peak. When the results were carefully checked it was found that some points had been flagged as "depth-deleted points" although their depth values were perfectly within the expected ranges.

CHAPTER 3

Testing the Ability of Back-Propagation to Identify Simple Topographic Features

3.1 Overview

The ability to identify basic topographic shapes such as slopes, planes, hills and valleys is a valuable feature when analyzing and processing bathymetric data.

This chapter attempts to study the feasibility of using Artificial Neural Networks (ANNs), in particular the Back-Propagation (BP) algorithm using the Generalized Delta Rule, for learning and recognition of simple topographic shapes of small area.

3.2 Objectives of Machine Pattern Recognition

Pattern recognition can be understood as a mapping from pattern space into class-membership space. This mapping is generally "opaque" when performed by biological systems, but when performed by machine it must become "transparent", that is, it must be precisely and unambiguously described. In computerized pattern recognition both the transparent mapping and the appropriate choice of the essential and sufficient features are of major importance. The choice of the sufficient and necessary pattern features is not always a simple task.

Although mapping transparency seems to be an essential characteristic of computerized pattern recognition, this is not often the case when using artificial neural networks (ANNs). ANNs do have well defined algorithms, but the computational details are not always clearly identifiable due to the massively parallel style of computation and the use of nonlinear functions.

3.3 Back-Propagation with the Generalized Delta Rule

Due to the natural constraints built into nervous systems, in particular the low velocity of neural signal transmission, it seems reasonable to infer that, when performing learning and classification, biological neural networks make use of concurrent parallel distributed processing mechanisms.

The idea that simple computational elements, connected in ways that resemble natural nerve networks, are able to perform autonomous pattern learning and classification has recently motivated and contributed towards the development of various ANN algorithms.

The Back-Propagation (BP) Network using the Generalized Delta Rule of Rumelhart, Hinton and Williams[9] is an ANN that typically performs well in pattern learning and classification, overcoming the limitation of early networks which could only solve linear classification problems[10].

The nodes of ANNs are organized in well defined layers. A BP net is a multilayer network made up of one input layer, at least one middle or hidden layer, and one output layer. The output signal of a node is passed to every node in the next layer (fully connected) through links or connections that amplify, attenuate or inhibit that signal through the use of weighting factors or "weights". The activation value of a node -- the value of its output at any given instant -- depends upon its inputs, its activation function and its bias. During testing, information flows only in one direction, from the input to the output layer. During learning information is also propagated backwards through the network and it is used to update the connection weights. The BP net gets its name from this mechanism. If, during learning, the network does not produce some desired or target output value, the responsibility for the error (the difference between the target and the obtained value) is assigned to every node in the network by propagating the error backwards (output to input direction) through the links of previous layers and correcting the weights of these connections.

3.4 Description of the Operation of a Back-Propagation Network

Detailed mathematical descriptions of the Back-Propagation Network and its Algorithm may be found in different sources such as Rumelhart McClelland and the PDP Research Group[9] and Pao[10]. The following is a description of the BP operation to provide a basic understanding of the network, based upon Pao's description[10].

Consider the three layer BP network of Fig. 3.1. This net has an input layer i, a hidden layer j, and an output layer k. When a neuron or node in the input layer receives an input, it passes it unaltered to every node in the middle or hidden layer.

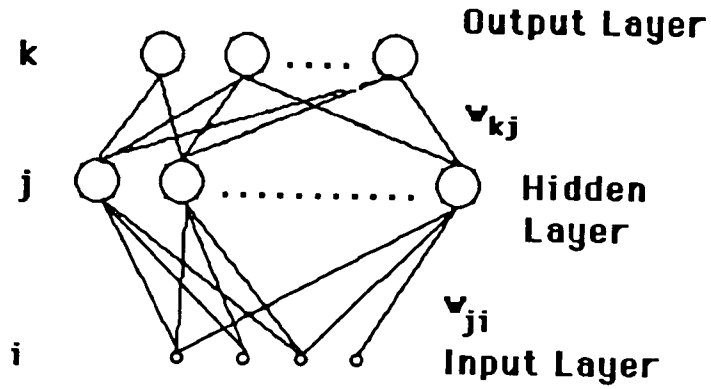


Fig.3.1 A Back-Propagation Network.

Each node j in the hidden layer receives a net input net_j that is defined as

$$net_j = \sum_i (w_{ji}\, o_i) \qquad (3.1)$$

where w_{ji} is the weight of the connection between a hidden node j and an input node i, and o_i is the real-valued magnitude of the input node i.

After receiving its net input, each node in the hidden layer computes its activation by applying a nonlinear function, generally a sigmoid (s shaped) function, to its net input net_j. The output of each hidden node is $o_j = f(net_j)$, where f is the sigmoid function. The output o_j is defined as

$$o_j = \frac{1}{1+e^{-(net_j+\theta_j)/\theta_o}} \qquad (3.2)$$

where θ_j is a bias or threshold value and e is the exponential constant.

The sigmoidal function has a squashing effect, to keep the activation values of the nodes between 0.0 and 1.0. The role of θ_o is to change the shape of the sigmoid curve. If the value of θ_o is small, the shape of the sigmoid function approaches the shape of a threshold-logic unit function; a large value of θ_o produces a shallower slope. A change in the value of the threshold θ_j results in a shift of the curve along the x axis. Fig. 3.2 shows the s-shaped function and the effect of changing θ_o (in Fig. 3.2 θ_o is represented as t0).

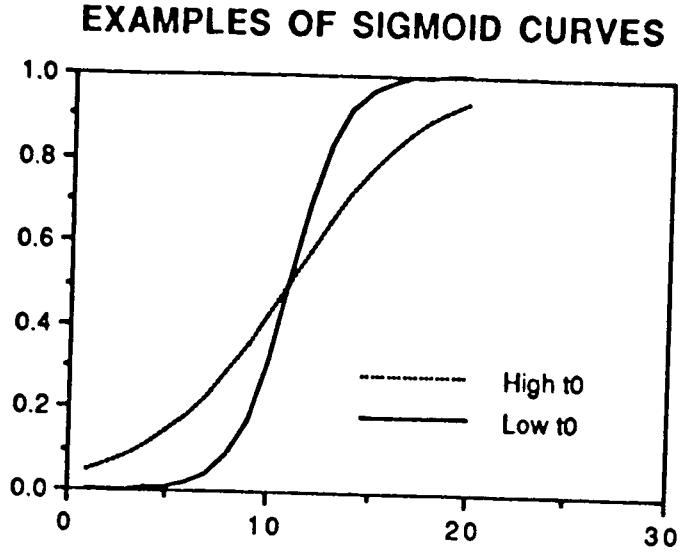


Fig.3.2 A sigmoid function

Once every node in the middle layer computes its activation, the process is repeated for the nodes in the output layer ($o_k = f(net_k)$), with the middle layer now supplying the inputs. This is the only process involved in the test or recall mode of a BP network.

In the learning mode, the BP net has two distinct phases: a forward pass and a backward pass. Learning starts with the forward pass. This process is exactly identical to the recall process described above.

Once the activations of the output nodes are computed, one must now correct the output of the net to provide the desired or target values. This initiates the backward propagation phase.

First, the error of each output node in the network must be determined by subtracting its desired output from its actual output. After the presentation of a pattern p, the squared error is

$$E = \frac{1}{2}\sum_k (t_k - o_k)^2 \qquad (3.3)$$

where t_k is the target value, o_k is the actual output value and the factor of 1/2 is used only for computational convenience.

The values of weights and thresholds may then be improved, to decrease the error, by making incremental changes Δw_{kj} proportional to $-\partial E/\partial w_{kj}$, where η is known as the learning rate

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \tag{3.4}$$

Note that error is expressed in terms of the output o_k, and $o_k = f(net_k)$, where net_k is the input of a node in layer k and it is defined as,

$$net_k = \sum w_{kj} o_j \tag{3.5}$$

The partial derivative in eq. (3.4) can be computed using the chain rule

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} \tag{3.6}$$

and since

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum w_{kj} o_j = o_j \tag{3.7}$$

one may define a new parameter δ, called delta, such that

$$\delta_k = - \frac{\partial E}{\partial net_k} \tag{3.8}$$

then the incremental changes in the weights become

$$\Delta w_{kj} = \eta \, \delta_k \, o_j \tag{3.9}$$

To obtain δ_k in terms of the rate of change of the of the error of a node k with respect to its output, and of the rate of change of the same output with respect to its input, one must compute

$$\delta k = - \frac{\partial E}{\partial net_k} = - \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k} \tag{3.10}$$

since,

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{3.11}$$

and

$$\frac{\partial o_k}{\partial net_k} = f'k(net_k) \tag{3.12}$$

then

$$\delta_k = (t_k - o_k)\, f'(net_k) \tag{3.13}$$

One can finally define Δw_{kj} for a link between a node k in the output layer and a node j in the last hidden layer in terms of δ_k and o_j as follows

$$\Delta w_{kj} = \eta(t_k - o_k)\, f'(net_k) o_j = \eta \delta_k o_j \tag{3.14}$$

To adjust the weights between the middle and the input layers one must use a different process since the desired target values for the nodes in the middle layer are not known. One must find δ_j in terms of previously known quantities, that is in terms of the deltas in an upper layer.

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= -\eta \frac{\partial E}{\partial net_j} o_i = \eta \left(-\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j}\right) o_i \tag{3.15}$$

$$= \eta \left(-\frac{\partial E}{\partial o_j}\right) f'(net_j)\, o_i = \eta\, \delta_j\, o_i$$

Since $\partial E/\partial o_j$ can't be evaluated directly one must find a way to express it in terms of known quantities

$$-\frac{\partial E}{\partial o_j} = -\sum_k \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial o_j}$$

$$= \sum_k \left(-\frac{\partial E}{\partial net_k}\right) \frac{\partial}{\partial o_j} \sum_m w_{km}\, o_m$$

$$= \sum_k \left(-\frac{\partial E}{\partial net_k}\right) w_{kj} = \sum_k \partial_k\, w_{kj} \tag{3.16}$$

Then in the case of intermediate nodes

$$\delta_j = f'_j (net_j) \sum_k \delta_k \, w_{kj} \qquad (3.17)$$

Starting at the upper layer the errors are propagated backwards and the weights corrected accordingly. One can then write, using the subscript p to refer to a given pattern,

$$\Delta_p \, w_{ji} = \eta \, \delta_{pj} \, o_{pi} \qquad (3.18)$$

Two cases develop now; if the j nodes are nodes in the output layer one can compute

$$\delta_{pj} = (t_{pj} - o_{pj}) \, f' (net_{pj}) \qquad (3.19)$$

but if the j nodes are nodes of an hidden layer, then

$$\delta_{pj} = f'_j (net_{pj}) \sum_k \delta_{pk} \, w_{kj} \qquad (3.20)$$

Assuming that

$$o_j = \frac{1}{1 + \exp[-(\sum_i w_{ji} o_i + \theta_j)]} \qquad (3.21)$$

then

$$\frac{\partial o_j}{\partial net_j} = o_j (1 - o_j) \qquad (3.22)$$

and finally the deltas for the output and hidden layer units may be computed using the expressions:

$$\delta_{pk} = (t_{pk} - o_{pk}) \, o_{pk} (1 - o_{pk}) \qquad (3.23)$$

$$\delta_{pj} = o_{pj} (1 - o_{pj}) \sum_k \delta_{pk} \, w_{kj} \qquad (3.24)$$

The thresholds θ are treated as weights and are adjusted as if they were the weights from a node with a constant output value of unity.

For more computational details on the above steps see Pao[10].

3.5 The BP Algorithm

The training of the BP net is a straightforward process. First, the weights of the network are initialized to small real random numbers, say between -0.5 and 0.5. Then, the network is presented with repeated sets of input patterns and target patterns, and the delta rule is used to adjust the weights for each pattern, until the desired network responses are obtained.

One summarized version of the algorithm to implement a BP network based upon Lippman's description [2], is shown below in fig 3.3.

1- Initialize the Weights and Offsets.

Initialize weights and offsets to small random values.

2- Present Input and Desired Outputs.

Present a continuous-valued input vector $x_0, x_1, \ldots, x_n$ and specify the desired outputs $o_0, o_1, \ldots, o_m$. (Generally the same samples are presented repeatedly during learning until weights stabilize.)

3- Calculate Actual Outputs.

Use equations (3.1) and (3.2), repeatedly at every layer level, to calculate the output $o_0, o_1, \ldots o_m$.

4- Adapt Weights.

Recursively start at the output nodes and work back toward the first hidden layer. Adjust weights so that $w_{ji}(t+1) = w_{ji}(t) + q d_j x'_j$, where $w_{ij}(t)$ is the weight at time t, x'_j is either the output from a node j or an input, q is a gain term, and d_j is an error term for node j.

a) If node j is an output node then

$$d_j = (1-o_j)(t_j-o_j),$$

where t_j is the desired or target output for node j, and o_j is its actual output.

b) If node j is an hidden node, then

$$d_j = x'_j (1-x'_j) \sum_k d_k w_{kj},$$

where k is over all nodes in the layers above node j.

Internal node thresholds are adapted in a similar manner. Convergence is sometimes faster if a momentum term is added and weight changes are smoothed by $w_{ji}(t+1) = w_{ji}(t) + m d_j x'_i + a(w_{ji}(t) - w_{ji}(t-1))$ and where $0<a<1$.

Fig. 3.3 Back Propagation Algorithm

3.6 Testing Back-Propagation Learning

Initial tests of the BP net were performed to study the learning and recall abilities of the network. The net was trained to recognize two simple topographic profile shapes, respectively: a +45 and a -45 degree smooth ramps. Each of these ramps was defined as an eight element depth-vector; the +45 degree ramp was defined by the vector $\mathbf{x}_1$= (0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0) while a -45 degree ramp was defined by $\mathbf{x}_2$= (0.0, -1.0,-2.0,-3.0,-4.0,-5.0,-6.0,-7.0). In a depth-vector each element represents the depth of a topographic profile at a given point of the profile.

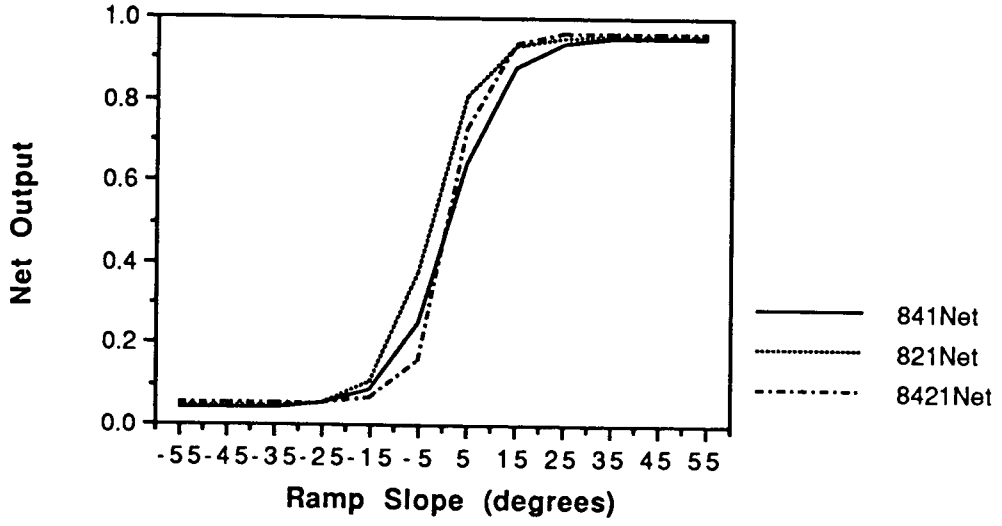
Three different networks were tested; the first net, here referred to as an 8-2-1 net, had an input layer with eight nodes, one hidden layer with two nodes and an output layer with a single node. An 8-4-1, and a two-hidden-layer 8-4-2-1 nets were also tested.

The nets were trained to produce an output of 1.0 when presented with a +45 degree ramp and an output of 0.0 when presented with a -45 degree ramp. The momentum (see step 4 of Fig.3.3) and the learning-rate parameters were kept constant and equal respectively to 0.9 and 0.7 and the networks were, in most cases, asked to perform with a total-system-error better than 0.001 and an individual-node error better than 0.0001.

To complete their training the networks required different number of iterations, an iteration being defined as a learning pass through the whole training set. The 8-2-1 net learned in 97 iterations, the 8-4-1 net learned in 52 iterations, and the 8-4-2-1 net required 171 iterations to learn the two patterns. Although the number of iterations taken by the 8-4-1 net was close to half the number taken by the 8-2-1 net, the learning times were not significantly different. The nets 8-4-1 and 8-2-1 took respectively 90 and 100 seconds to complete their learning. (These tests were performed on an IBM PC XT with a 4.7 MHz clock and without a math co-processor.) This can be easily understood because although the number of iterations was about half for the first net, its number of units in the hidden layer was double. It took the 8-4-2-1 net over 300 seconds to complete its learning.

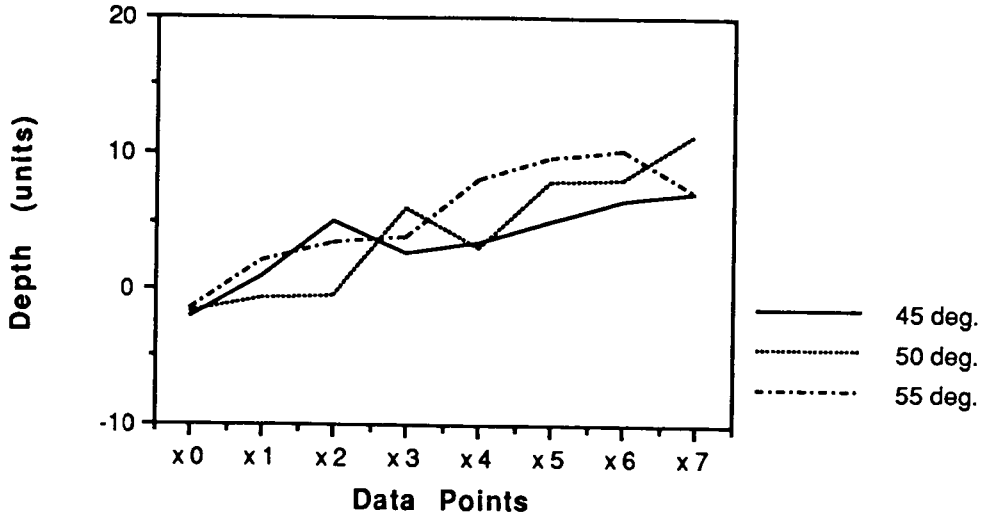
After learning the two sample patterns, the nets were tested using a set of twenty-three patterns representing smooth-ramps between -55 and +55 degrees at 5 degree intervals. The output of the networks were plotted in Graph 3.1. There were no major differences found in the performance of the networks.

All nets were able to generalize well and classify correctly the input patterns. Note that since the characteristics of the output curve are smooth and regular, one is able to obtain a reasonable measure of the slope of the tested ramps although this is not the main purpose of the classification.



Graph 3.1 Back Propagation Test. Training set=2 (-45, +45), m=0.9, g=0.7, error < 0.001, iter(821)=97, iter(841)=52, iter(8421)=171. Test set=23 (-55 to +55, every 5 degrees).

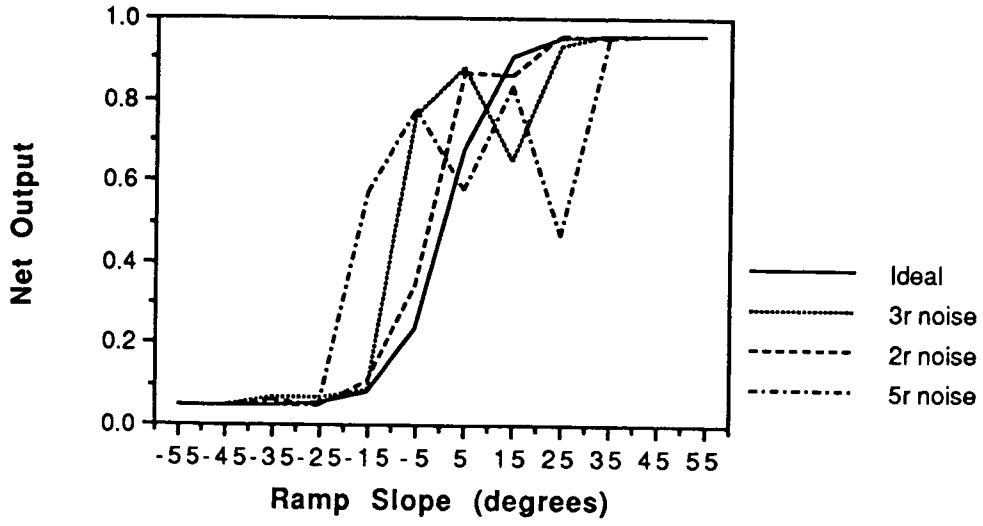
The next test consisted of presenting the networks with corrupted patterns **x'**. The twenty-three test samples were corrupted with noise. The corrupting operation consisted of adding randomly to each of the eight elements of each ramp ±3.0 noise units, such that any point x'_j in a test sample would be $(x_j-3.0) < x'_j < (x_j+3.0)$. Some examples of the distorted ramps are presented in Graph 3.2. The graph shows the +45, +50 and +55 degree ramps corrupted with ±3.0 units of random noise.



Graph 3.2 Examples of 3 "noise-corrupted" ramps. Ramps were obtained by adding to each data point of an "ideal" ramp, a random number between -3.0 and 3.0.

The results of the classification of some noise-corrupted patterns by the 8-2-1 network are plotted in Graph 3.3. The graph shows the results of the tests on the 23 sample-patterns with ±2.0, ±3.0 and ±5.0 units of random noise, described respectively as 2r, 3r, and 5r noise, versus the set of ideal ramps. The network was still able to generalize well and classify correctly its corrupted-inputs when these inputs are close to the learning samples, that is, about +45 and -45 degrees. The response of the network to patterns in the intermediate region -- horizontal flat profiles -- becomes unpredictable, with the output of the net swinging abruptly from pattern to pattern.

One concludes, at least for the tested cases, that the net seems to be able to "grasp" the basic differences between positive and negative ramps regardless of the amount of noise introduced (2r, 3r and 5r noise). This performance has both advantages and disadvantages. If one wants generalization in the classification, the net performs very well both in the case of smooth ramps and of noise-corrupted ramps. If one wants some measure of the slope of the input ramp, then the output is useful only when classifying smooth-ramps; when classifying distorted ramps the net output is meaningless.



Graph 3.3 Back Propagation Test. 821 Net. Train set = 2 (-45, +45),m=0.9, g=0.7, error<0.001. Test set=23 (-55 to +55). Tested ramps were ideal ramps and ramps corrupted with 2r, 3r and 5r noise.

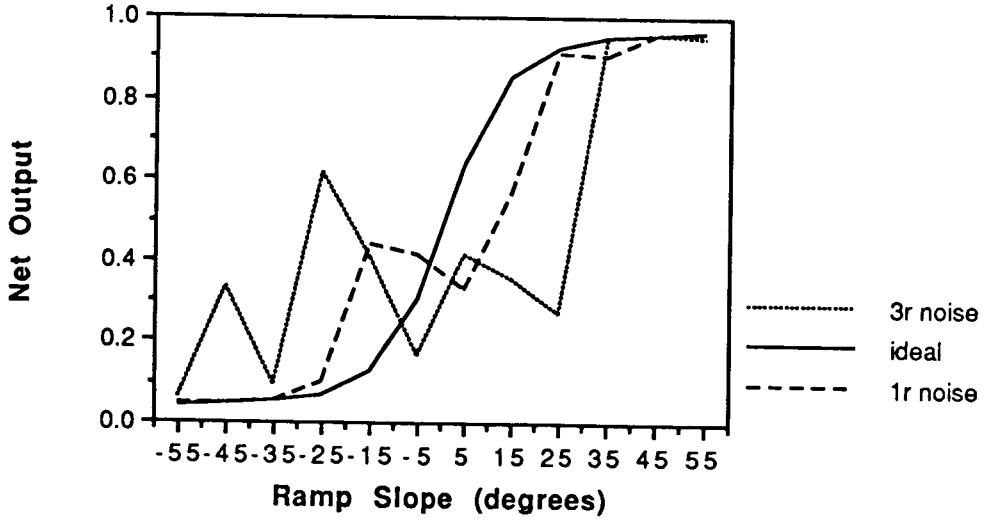
It is well known, in Pattern Recognition, that the choice of representational features is a very important step towards obtaining the desired results. Choosing the wrong feature space may turn a simple task into one of impossible solution.

The features chosen to represent a pattern must be sufficient and unambiguous so that the recognition system may do its job properly. With this in mind, the learning and testing sample sets used above (and, for consistency, the only ones used throughout these tests), were translated from depth (amplitude) values into angle values, where each vector element is the angle of the line joining two consecutive points of the profile. For instance, a depth vector $\mathbf{x}_d$=(0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0) when translated into an angle vector would become $\mathbf{x}_a$=(45, 45, 45, 45, 45, 45, 45, 45), assuming that the run and the rise of each point of the profile are both equal to 1.0. To maintain the same vector and network sizes, one assumes that there exists an extra imaginary 9th point in the depth-vector (in order that the number of elements of the angle-vector may be also equal to eight.)

The 8-2-1 BP net was then trained on the new translated angle samples of -45 and +45 degree ramps. The net learned and converged in 105

iterations, a number slightly larger than the number of iterations (97) needed for the same net to learn the amplitude or depth vectors.

The test results of learning and classifying the new angle vectors are shown in Graph 3.4, picturing the response of the network to the ideal (no noise) 23 angle-vectors testing set, to the testing set with a 1r noise, and to the testing set with 3r noise. Note that for the sake of consistency, these noisy sets are the same amplitude-vectors used in the previous tests, now simply translated into angle-vectors. As it can be easily observed in the graph, the 8-2-1 BP net is much more sensitive to noise when trained with angle-vectors than when trained with amplitude-vectors. The fact that the slope of the middle portion of the response curve to the ideal or smooth-ramps is not as sharp as in the previous tests with the amplitude vectors, also confirms this observation.



Graph 3.4 Back Propagation Test. 821 Net. Train set = 2 (-45, +45) of angle-vectors,m=0.9, g=0.7, error<0.001. Test set=23 (-55 to +55). Tested ramps were ideal ramps and ramps corrupted with 1r and 3r noise.

3.6.1 Multi-Hidden-Layer Nets

Although single hidden layer nets performed satisfactorily in learning the difference between positive and negative ramps and in classifying the posed patterns, when a third class, a flat horizontal profile, was added to the training

set and presented to an 8-4-3 net -- a net with eight input nodes, four nodes in the hidden layer, and three nodes in the output layer -- the network, even after taking 758 iterations to converge, was unable to classify correctly the twenty three patterns of the testing set.

The poor performance of the 8-4-3 net was no surprise. The samples of the testing set were presented to an unsupervised-learning classifier and after an appropriate threshold was selected so that three clusters were obtained, it was evident that the patterns formed three contiguous and close classes in some eight dimensional space. It is still not clear whether such a network would not be able to classify the testing samples if given different parameters, since single hidden layer nets are indeed capable of classifying and separating convex open or closed regions (see Fig.3.4). When the same training set was presented to an 8-4-4-3 BP network (see Graph 3.5), the network easily learned the correct discriminants in 129 iterations through the 3-sample training set. The network was trained to produce an output of 1.0-0.0-0.0 to a -45 degree ramp, an output of 0.0-1.0-0.0 to a flat pattern (0 degree ramp) and an output of 0.0-0.0-1.0 to a ramp of +45 degrees.

Instead of using a single output node to identify more than one class, the unambiguous identification of a class was indicated when two of the nodes were "off" (close to zero) and the other node was "on" (close to one).

Networks with multi-hidden layers are able to perform complex mappings (see again Fig.3.4).

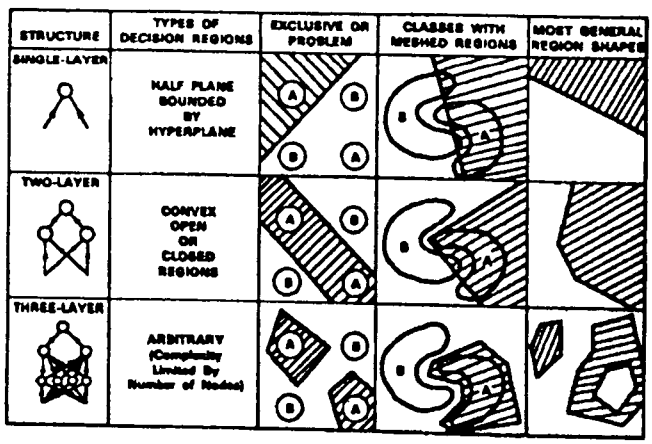
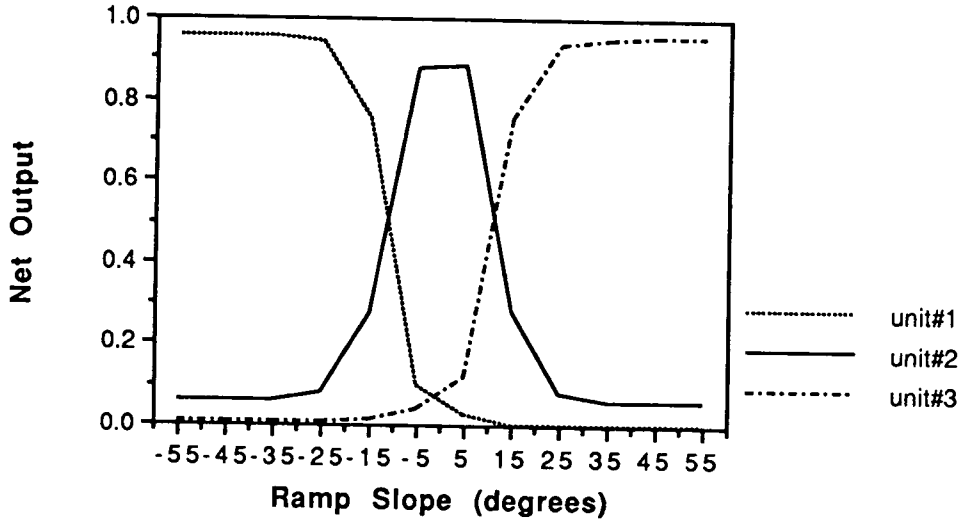
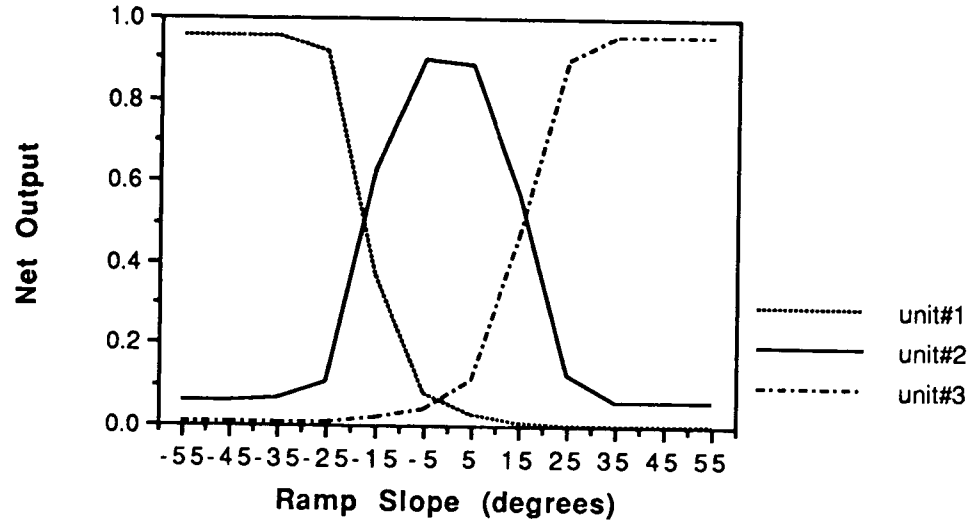


Fig.3.4 Types of decision regions performed by perceptrons. (Figure copied from "An Introduction to Computing with Neural Nets, by R.L.Lippman[2])

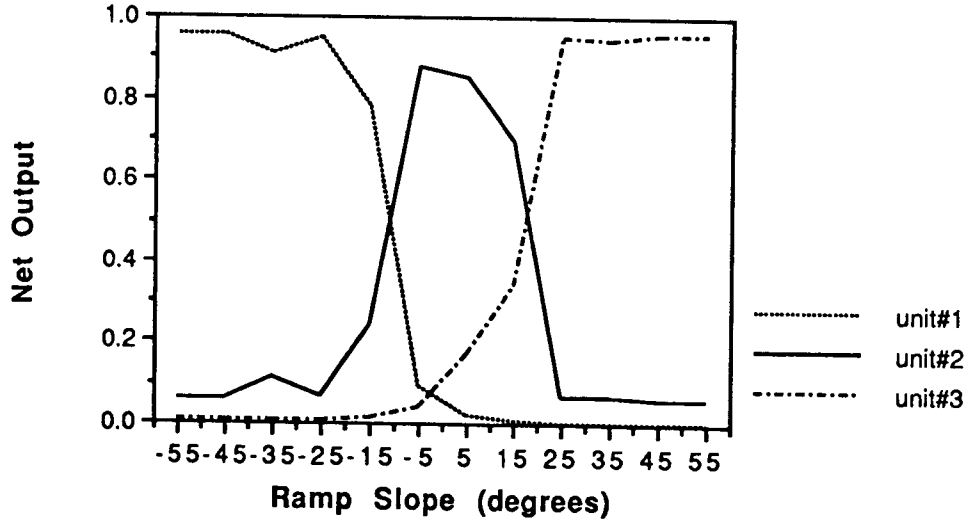
When tested, the 8-4-4-3 BP net performed satisfactorily even with a noise-corrupted set. Graphs 3. 6, 3.7, and 3.8 show the network performance with the 1r, 2r, and 3r unit noise. One must note that the introduction of up -3.0 to +3.0 units of random noise to smooth-ramp patterns can produce a very large distortion of the pattern, mainly in close-to-flat patterns.



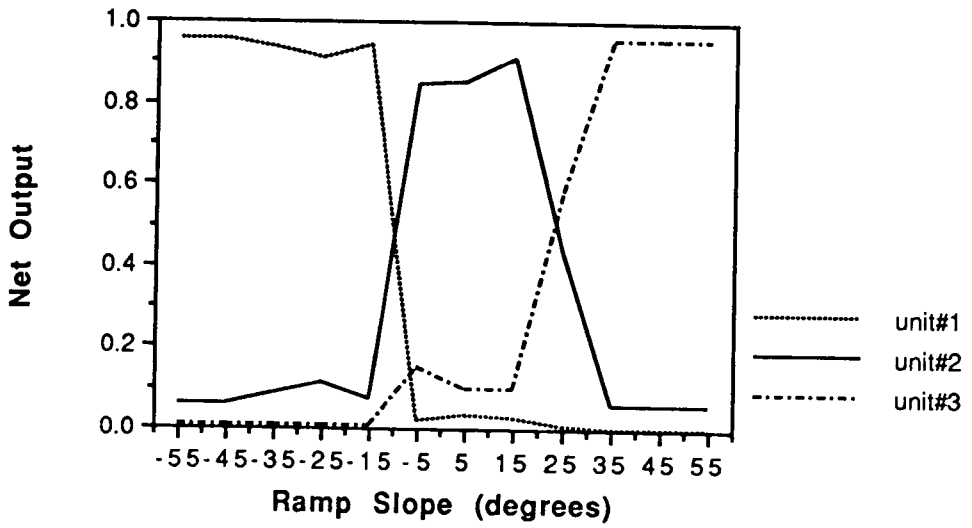
Graph 3.5 Back Propagation Test. 8443 Net. Train set = 3 (-45, 0, +45) of depth-vectors,m=0.9, g=0.7, error<0.001. Iter=129, time=540sec.Test set=23 (-55 to +55). Tested ramps were ideal ramps.



Graph 3.6 Back Propagation Test. 8443 Net. Train set = 3 (-45, 0, +45) of depth-vectors,m=0.9, g=0.7, error<0.001. Test set=23 (-55 to +55). Tested ramps were 1r noise ramps.

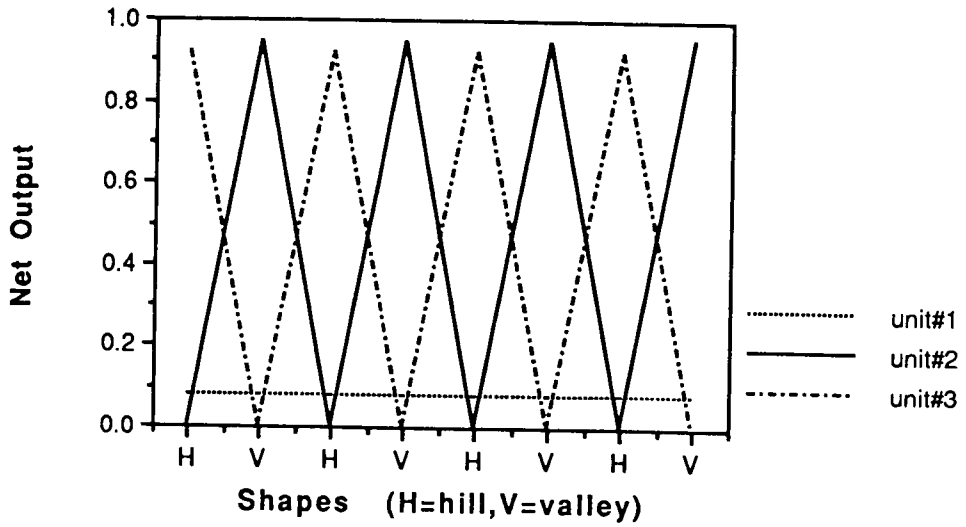


Graph 3.7 Back Propagation Test. 8443 Net. Train set = 3 (-45, 0, +45) of depth-vectors,m=0.9, g=0.7, error<0.001. Test set=23 (-55 to +55). Tested ramps were 2r noise ramps.



Graph 3.8 Back Propagation Test. 8443 Net. Train set = 3 (-45, 0, +45) of depth-vectors,m=0.9, g=0.7, error<0.001. Test set=23 (-55 to +55). Tested ramps were 3r noise ramps.

Testing for other interesting shapes was also important. The following task consisted of training the 8-4-4-3 net to learn to differentiate "hills" and "valleys" from other common shapes. The training set had five profile patterns from 3 classes: one hill (identified by unit #3), one valley (identified by unit #2), and one -45 degree ramp, one +45 degree ramp and one horizontal-flat profile (all identified by unit #1). The hill pattern was made up of two half-ramps with the top at the center of the profile. The valley was an inverted hill. Graph 3.9 shows the results of this test when classifying perfect shapes. The net was presented *only* with instances of perfect hills and valleys and was able to perform correctly.



Graph 3.9 Back Propagation Test. 8443 Net. Train set = 5 (hill,valley,others) of depth-vectors,m=0.9, g=0.7, error<0.001. Test set: ideal hill and ideal valley.

The tests described in this chapter were initial and simple tests performed on an IBM PC XT. Although the dimentionality of the vectors was small and the patterns were simple, the tests provided important clues on the feasibility of using back-propagation multi-layer neural networks to perform detection and recognition of common topographic shapes. Further work with real topographic data is described in the next chapter.

CHAPTER 4

Testing the BP Network on Swath Data

4.1 T1 Tests. Introduction

Encouraged by the results of the tests of the previous chapter, the tests described in this section -- T1 tests -- attempt to investigate the performance of four-layer (one input, two hidden and one output) BP networks when used for feature detection in swath data.

These tests study the importance of the number of units in the hidden layers of the networks. Two hidden-layer networks were chosen, because these nets are capable of forming complex (meshed and non-linear) decision regions in pattern classification space[2].

Since the number of units in the hidden layers of multi-layer neural networks determines how well these nets perform pattern classification[2], it is important to investigate how network size affects the classification of topographic features when dealing with ocean floor swath data.

The study consisted of training different networks on patterns based on two prototype samples extracted from the Louisbourg file SW1989000230.PLT, and then testing the nets on data from the same file.

4.2 The Training Set

Figures 4.2.1 and 4.2.2 show the shapes of the two basic prototype samples that were used in these tests. These samples were obtained with the use of XPAT (eXtract PATtern), a software tool specially designed to ease the task of selecting patterns (portions of ocean floor) from PLT files and storing them in training files. With XPAT this selection can easily be done visually and interactively.

Each prototype sample was made up of 49 data points (a vector in a 49-dimensional space), each point indicating the depth of the ocean floor at some geographical location. For computational convenience, and since the data obtained from many swath data systems, such as the *Navitronics*, is regularly structured (the profile data points lie equidistantly on a straight line), the samples were organized in the form of a 7x7 grid (a square matrix). One assumes that the distance between profiles is always the same.

Prototype #0 was obtained from records 10 to 16 of profiles 350 to 356 and it looks very much like a common ramp.

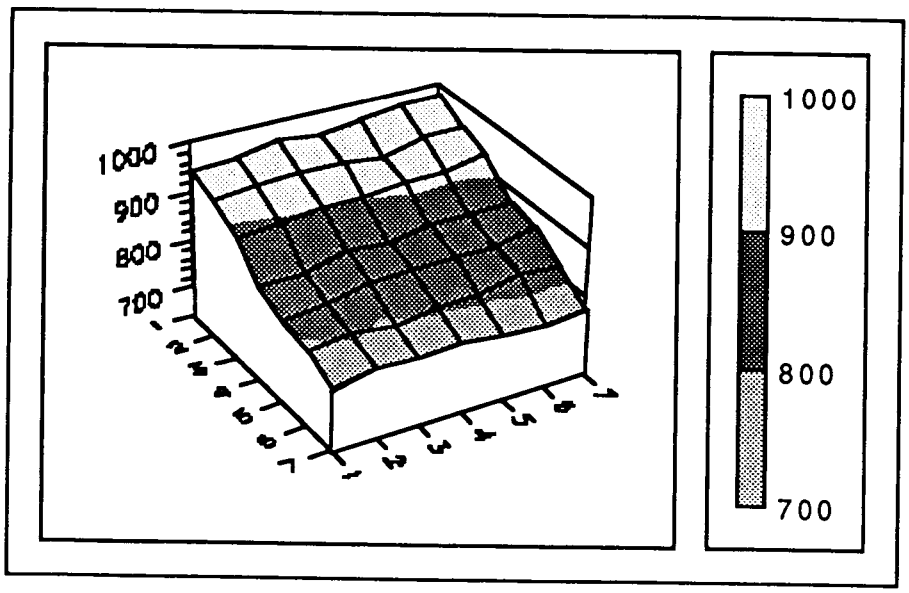


Fig.4.2.1 Prototype #0, from records 10 to 16 of profiles 350 to 356.

Prototype #1 was obtained from records 2 to 8 from profiles 378 to 384. The pattern of Fig.4.2.2 has been rotated clockwise approximately 180 degrees to facilitate the view of the prominent features of the area (this is shown by the numbering of the elements in the axes of graph).

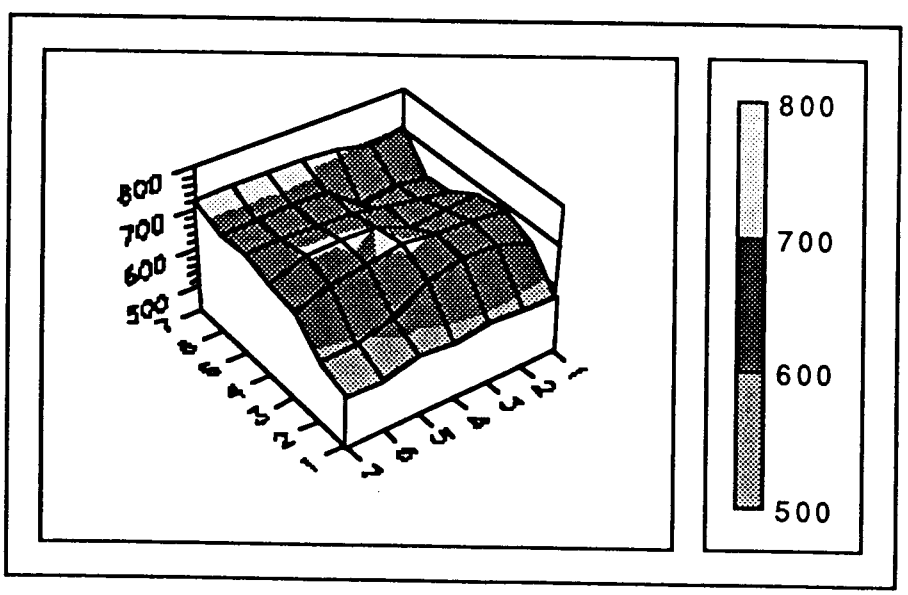


Fig. 4.2.2 Prototype #1, from records 2 to 8 of profiles 378 to 384.

A training set of thirty pattern samples was generated from the two prototypes. The patterns of the training set were divided into two classes. Each class contained the class prototype plus fourteen other patterns generated from the prototype. Each point in the generated patterns was obtained by adding to the corresponding point in the prototype pattern a positive integer random number between zero and two per cent of its value. For instance, a point in a prototype pattern with a value of 1000 depth-units could produce a sample point with any value between 1000 and 1020 units.

The input patterns were then normalized using the Euclidian norm, to preserve the relationships between the pattern elements. This is an important step because it makes the pattern independent of the absolute values of its individual elements.

The Euclidian norm of a vector **x** with n elements can be defined as

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2} \qquad (4.1)$$

Once the value of the magnitude of the input vector, or norm, is computed, each element in the pattern is then computed by dividing its value by the norm of the vector,

$$x_i = \frac{x_i}{||x||} \qquad (4.2)$$

4.3 Network Learning

During learning, in all tests, the momentum and the learning rate were kept constant and equal to 0.9 and 0.7, respectively, as suggested by Rumelhart and McClelland[9]. Before learning started, the weights of the connections were initialized to small random values between -0.5 to +0.5, and learning was stopped when the normalized system error reached below 0.001 (see eq.(3.3)) and individual unit errors below 0.0001. The sizes of the input and output layers were always constant and equal to 49 (the number of points in the patterns) and 2 (the number of classes), respectively. The only parameters allowed to vary were the sizes of the two hidden layers.

Training a network was done in a supervised mode and consisted of repeatedly presenting the patterns in the training set to the network and noting the resultant output. If the output produced by the units in the output layer differed from the target (desired) results for more than some allowed value, then the back-propagation learning would take place to decrease the error. This process was repeated until the error was acceptable, at which time the learning was stopped.

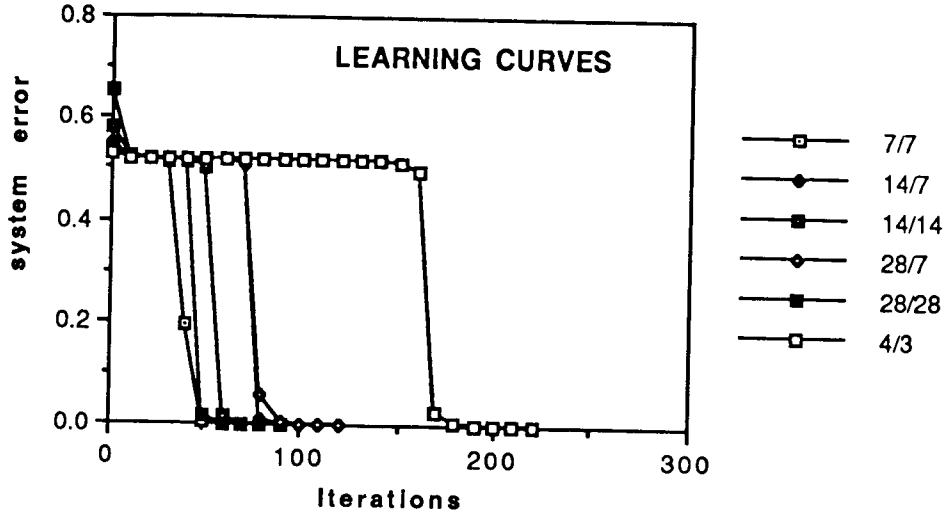
During back-propagation learning, the weights of the connections are adjusted according to the back-propagation algorithm as discussed in section 3.4 of the previous chapter. Once a network has been successful in learning a given task, these weights are stored in a weight data file and can be loaded back into the network at any time when testing (or recall) is needed.

Six cases were initially tested, as shown in Table 4.3.1. The first column of the table identifies the test, the second and third columns show the number of computing nodes in the first and second hidden layers, the fourth column shows the number of connections in the networks (or the number of weights), the fifth column shows the number of iterations (passes over the training set) needed for training, and the sixth and last column shows the training time in seconds.

Table 4.3.1

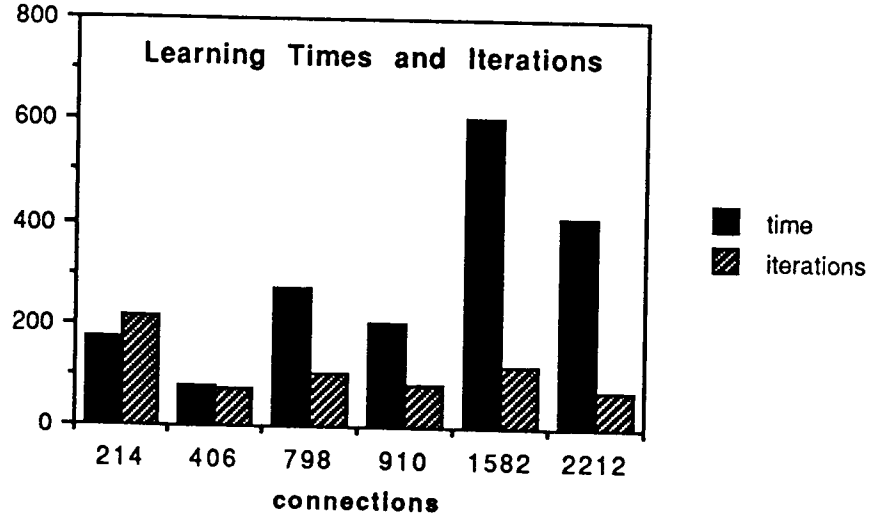
Test ID	1st HLayer	2nd HLayer	Connections	Iterations	Time
1NA	7	7	406	71	76
1NB	14	7	798	106	274
1NC	14	14	910	86	206
1ND	28	7	1582	118	608
1NE	28	28	2212	74	411
1NF	4	3	214	214	171

The analysis of the system error during learning is also important. Graph 4.3.1 shows the behaviour of the networks as learning progresses; after a small initial decrease, the system error seems to stabilize for a while and then suddenly plunges to its final value, below 0.001. The observed behaviour is common to all nets.



Graph 4.3.1 Learning Curves from tests T1.

One can also examine the relationship between the number of the learning iterations used by the different networks and the actual time in seconds as a function of the number of network connections. Figure 4.3.2 illustrates these results. The ratios between the number of seconds and the number of iterations for a given net seem to vary greatly from a 1.07 ratio for the 49/7/7/2 net (406 connections) to a 5.55 ratio for the 49/28/28/2 net (2212 connections). There is an overall increase with the number of connections, although this increase is not monotonic.



Graph 4.3.2 Times and iterations vs. connections.

4.4 Network Recall

Since the networks had been trained with patterns starting at profiles 350 and 378 of the Louisbourg file, the first recall tests consisted of presenting the networks with the data from profiles 300 to 400 (2700 patterns) from the same file.

During the testing phase (or recall), the networks were asked to identify a pattern as an instance of a class if the value of the corresponding output unit was above 0.98, regardless of the value of the other output unit. This was understood to be a loose criteria since during learning both output units were necessary to define the final result (0-1 or 1-0). Nevertheless, this criteria seemed acceptable, at least for initial testing, since it would allow for the inclusion, and visual examination, of a larger number of patterns. The required value of 0.98 seemed to be a reasonable value, since the nets had been trained for a normalized system error smaller than 0.001.

Table 4.4.1 shows the output of testing net 49/14/7/2. The first two columns identify the pattern's location in the data file and the third column shows the pattern class (similarity with prototype #0 or #1). Note that neither of the prototype patterns (350/10 or 378/2) was identified probably because of the two percent random-noise added to the training samples.

Table 4.4.1 Results of tests and similarity measures between patterns.

Profile	Rec	Class	Output	Euclidian	Dir Cos	Tanimoto
351	9	0	0.980655	0.018187	0.999835	0.997710
351	10	0	0.982055	0.018819	0.999823	0.998164
351	11	0	0.982728	0.019874	0.999803	0.998569
351	12	0	0.981583	0.021483	0.999770	0.998703
351	13	0	0.980048	0.021998	0.999758	0.998850
351	15	0	0.980937	0.022421	0.999749	0.999194
352	9	0	0.981452	0.023275	0.999729	0.992472
352	10	0	0.983927	0.024557	0.999699	0.993105
352	11	0	0.984937	0.024732	0.999694	0.994023
352	12	0	0.985257	0.024602	0.999698	0.994633
352	13	0	0.984045	0.025044	0.999687	0.995208
352	14	0	0.984119	0.025256	0.999681	0.995748
352	15	0	0.983226	0.023839	0.999716	0.996319
352	16	0	0.981405	0.022450	0.999748	0.996618
352	17	0	0.980243	0.022363	0.999750	0.996851
352	18	0	0.980183	0.021622	0.999766	0.996871
352	19	0	0.981160	0.023173	0.999732	0.996856
352	20	0	0.981760	0.022479	0.999748	0.996941
352	21	0	0.982332	0.020498	0.999790	0.997006
352	22	0	0.982338	0.019919	0.999802	0.997093
352	23	0	0.982753	0.020167	0.999797	0.997485
352	24	0	0.983335	0.020775	0.999785	0.997704
352	25	0	0.983883	0.019550	0.999809	0.998064
352	26	0	0.983173	0.021244	0.999775	0.998081

Profile	Rec	Class	Output	Euclidian	Dir Cos	Tanimoto
353	10	0	0.980839	0.024363	0.999704	0.984927
353	11	0	0.982744	0.024426	0.999702	0.986248
353	12	0	0.982025	0.024725	0.999695	0.987291
353	13	0	0.982542	0.024231	0.999707	0.988207
353	14	0	0.981905	0.022877	0.999739	0.989428
353	15	0	0.982035	0.022967	0.999737	0.990172
353	16	0	0.982692	0.023466	0.999725	0.990752
353	17	0	0.982845	0.022820	0.999740	0.991341
353	18	0	0.983039	0.024093	0.999710	0.991312
353	19	0	0.983541	0.025927	0.999664	0.991305
353	20	0	0.982405	0.026463	0.999650	0.991350
353	21	0	0.983614	0.023482	0.999725	0.991350
353	22	0	0.983405	0.021540	0.999768	0.991681
353	23	0	0.982530	0.019672	0.999807	0.992373
353	24	0	0.983805	0.020359	0.999793	0.992697
353	25	0	0.983106	0.020033	0.999800	0.993340
353	26	0	0.983188	0.023441	0.999725	0.993304
354	10	0	0.982108	0.025598	0.999673	0.975514
354	11	0	0.981546	0.025307	0.999680	0.977308
354	13	0	0.980001	0.025494	0.999676	0.979458
354	14	0	0.980815	0.022446	0.999748	0.980975
354	15	0	0.980559	0.022614	0.999745	0.981832
354	18	0	0.980935	0.023243	0.999730	0.982988
354	19	0	0.981707	0.025399	0.999678	0.982845
354	23	0	0.981888	0.022822	0.999740	0.983902
354	24	0	0.983068	0.023101	0.999734	0.984380
354	25	0	0.982235	0.020849	0.999783	0.985393
354	26	0	0.981297	0.026294	0.999654	0.985267
378	3	1	0.981762	0.028555	0.999592	0.999172
378	4	1	0.983082	0.051160	0.998691	0.997115
378	5	1	0.980295	0.070751	0.997497	0.993943

During recall, all nets produced results very similar to those displayed in Table 4.4.1. These values are typical for all performed tests, both in the total amount of patterns identified by the networks (approx. 50) and the location of those patterns.

To obtain an idea of how far, in some metric space, the identified patterns were from the associated prototypes (similarity measure), three similarity measuring methods were used: euclidian distances, direction cosines and Tanimoto's distance (see [12]). Initially, only the euclidian distance was used to measure the similarity between the patterns identified by the networks and the corresponding prototypes. The numbers obtained were, at times, difficult to interpret as a measure of visual (perceptual) similarity, so the other two methods were tried; the direction-cosines method, to preserve the relative magnitude of the individual components, and Tanimoto's method. The results of these similarity tests are displayed in columns five to seven. Tanimoto's method seems, empirically, to be the most useful of the three, since it provides

information that corresponds best with the visual perception of the patterns (see table 4.4.1)).

4.5 Further Testing

To test the effect of using minimum size training sets, a 49/4/2/2 network (208 connections) was trained with only two samples, the two prototype samples. The network learned in 1516 iterations, in 73 seconds, for maximum normalized system error under 0.01 and/or maximum unit error below 0.001.

When the same Louisbourg data was tested, the net did not produce any output for target values above 0.95. When the target value was reduced to 0.93, there were 56 patterns, from profiles 351 to 354, identifyed as class #0 patterns, and 5 patterns, from records 2 to 6 of profile 378, identified as class #1 patterns. All of the 56 class #0 patterns were ramps visually similar to the prototype ramp, although the prototype ramp itself (350/10) was not included among these. The prototype pattern for class #1 was among the 5 identified patterns.

Because the results of all T1 tests were very similar, both in terms of the number of outputs produced by the networks and in terms of the identity of the patterns identified, one observes that at least for the type of patterns used in the training set, and the parameters used by the networks, varying the number of units in the two hidden layers does not considerably alter the performance of the networks. This shows that in conditions similar to the ones met in these tests, small networks perform as well as larger ones. This is an important result since swath data files tend to be very large files and small nets offer the advantage of shorter processing (recall) time.

When identifying, or locating, one single topographic shape, and this seems to be the most common case, one may disregard the uninteresting class, as far as some unnecessary processing is concerned (i.e. recall information is not printed to the console or to a file.) This will also shorten recall processing time.

To evaluate network performance when the feature space of the important class is further constrained, that is, increasing the number of samples representative of the class that does not contain the pattern of interest one wants to find, a sampling set was created with patterns from two classes. One class contained 45 patterns generated from nine basic shapes, ramps at 0, +10, -10, +20, -20, +30, -30, +45 and -45 degrees, by introducing five percent of

random noise to each of these ramps. The other class contained only one pattern, the pattern of interest, but repeated 45 times, alternating with the patterns from the other class.

The network selected for this test was a 49/7/3/2 network -- 370 connections --, trained for a maximum unit error of 0.001 and a maximum system error of 0.01. The net learned in 51 iterations over the 90 sample learning set, in 108 seconds.

The network performed well when tested on profiles 300 to 400 of the Louisbourg file, producing 2 patterns (378/2 and 378/3) for a target value greater than 0.90.

Further feature-space constraint was studied in a similar test using a larger training set, with 180 samples generated in the same manner as the previous example. The size of the net, in this case 49/4/4/2, was slightly smaller than for the previous case. Although with a larger training set, the net learned faster, in 16 seconds time, and with only 6 iterations over the training set. When tested on the entire Louisbourg file of 3633 profiles for a target value greater than 0.94, the net identified 11 patterns in 603 seconds. The prototype pattern 378/2 was among these. Four other patterns were ramps not visually similar to the prototype and the remaining six patterns had some zeros in the data (generally originated by inactive sensors) and could be considered as "defective" patterns.

In summary, further testing with small size nets was done with three different networks each trained with a different number of patterns. For this data, small networks trained with smaller training sets detected patterns similar to the prototypes, while increasing the number of training samples, apparently, decreased the success of the network. The results may be misleading since the number of tested patterns was not the same; in the last case the whole file was tested while in the two first cases only 100 profiles were tested.

The major difficulty that was found during the recall tests was the selection of the target values. As recorded above, a small difference in the target value can lead to very different classification results.

CHAPTER 5

Conclusions

5.1 General Remarks

ANNs have enjoyed a fair measure of success in different application areas, such as Pattern Recognition and Classification. Examples of these are: "A System to Classify Sonar Targets"[14] and Snoope a system developed to detect explosives regardless of shape of packaging[13]. It was therefore of interest to investigate the possibility of applying neural nets to the important area of identification and localization of data artifacts and certain topographic shapes in bathymetric swath data. Other biologically inspired methods were also tried.

5.2 Peak Detection

Peak artifacts in swath data are common problems due to faulty or inoperative sensors and other more complex phenomena such as sound interference between adjacent sensors. Two methods were employed for peak detection (see chapter 2). The PDNet is a computational system was inspired by the processes that exist between neighboring neurons in biological systems. The LI method was inspired by the lateral inhibition that occurs between neighboring neurons and makes possible the edge detection mechanisms in visual systems.

Although the PDNet is a useful tool, one must be aware of the limitations that are introduced by the nature of the processes involved. One of these limitations is due to the nature of its most valuable property: its local dynamic adaptability. The PDNet adapts to all features consistently present in the input data. Unexpected data values are rejected by the PDNet, but if these initially unexpected data remains for a while in the input, the net adapts to them and learns to accept them. This well known predicament, common to all adaptive methods, is a hard problem to solve. The PDNet also employs an user-defined threshold , whose value is added to the standard deviation of the neighborhood

values to provide for greater decision control. This means that whether a feature is accepted or rejected is, in some degree, also dependent upon the selected threshold value.

Single rejected peaks are not allowed to affect the adaptive mechanism of the PDNet. This important feature is responsible for maintaining the correct biases and thresholds of the network.

To decrease processing time, points are first subjected to quick local testing using only the closest neighbors in the data-profile (line oriented testing). Any point rejected by this local testing is then thoroughly tested considering all surrounding points (area oriented testing), and may then be accepted or rejected by this final test. Because the number of points (determined by the neighborhood size) involved in this final test is also user defined, one must then expect that the result will vary according to the value chosen for the neighborhood size.

When tested on the Louisbourg sw1988000230.plt file, a file containing 119889 depth points, the PDNet worked satisfactorily and located a few data artifacts which had been overlooked by previous visual inspection . The number of points accepted or rejected changes, as mentioned above, with the value selected for the threshold and number of neighboring points required for the test (see table 2.6.1 and graph 2.6.1).

The processing times of the PDNet are plotted in graph 2.6.2. In the worst case, it took the network 56 seconds to reject 6267 points using a neighborhood area of 11x11 and a threshold bias of 20. In the best case, it took 19 seconds to reject 98 points for a neighborhood area of 11x11 and threshold bias of 100.

To further decrease the computational time, the LI method, inspired by the biological mechanisms of lateral inhibition, was implemented for peak detection (see the results in table 2.6.2). In the worst case, it took this method 24 seconds to reject 4025 points, using a neighborhood area of 11x11 and a threshold of 20. In the best case it took only 4 seconds to reject 87 points, using a neighborhood area of 11x11 and a threshold of 100.

Edge detection principles were applied to swath data and proved successful in peak detection, mainly in identifying isolated peaks. Good practical results were also obtained when the two methods (Neighborhoof Averaging and Lateral Inhibition) were combined sequentially. This was done

by using the output file produced by the PDNet as the input file to the edge detector.

To facilitate experimentation and provide for easy visual checking of the results of the peak detection tests, a software tool was created to link SWANN with SWATH VIEWER (see appendix A).

5.3 Feature Learning and Detection

As explained in chapter 4, a four-layer back-propagation neural network was implemented to detect and locate user-specified patterns. This method was chosen because of its ability to learn by example and its ability to handle nonlinear pattern discrimination. ANNs can extract the dominant statistical features of the data from a collection of training samples representative of some domain, and generate a set of "weights" that will allow the system to later classify patterns from that domain.

The results of the tests run on the Louisbourg sw1988000230.plt file using the BPNet were satisfactory (see chapter 4). Patterns similar to the prototypes learned by the networks were found in all tests and visually displayed along with their location.

It was noticed that, for successful training, it is important that the samples used in the training set represent as closely as possible the domain of interest. That may imply that sometimes a large number of training samples is needed. It has been suggested [13], as a rule of thumb, that for successful learning, the number of training samples should at least equal the number of weights (connections) in the network. In most test run, however, the number of samples used to train the BPNet was well below this suggested number.

It was also noticed during experimentation that the value of the target specified in the testing phase -- the value that determines whether a pattern belongs to some class -- will strongly determine the number of patterns identified by the network. For instance, when trying to detect some shape of interest the network will produce two or three patterns when a target of 0.93 is specified, but will produce about 20 patterns when a target of 0.90 is chosen (these values were selected because the learnig target had been equal to 1.0). This is not necessarily a handicap, since in most cases all the identifyed samples will be similar to the pattern sought, but in other cases totally dissimilar patterns may be identified. It was noticed however, that in most of these latter

instances the data contained "bad" points -- a group of zeros on one side of the profile, probably due to inoperative sensors -- and these "bad" patterns were very easily identified (visually very different). It is not clear at this stage if these deficiencies are due to poor training, inadequate network architecture, or both. These issues are largely unresolved. A method that may be used to avoid this type of error is to check for profiles containing zeros, and preventing these from entering the network for classification.

The overall conclusion from this work is that although ANNs offer some useful computational properties, such as their "heuristic style, generalization characteristics" and learning ability, the difficulty in selecting the recall target values (classification thresholds) raises some questions about their use in swath dat applications. It is suspected that they are most useful when used together with "more traditional and accurate" computational methods. Also, particularly in the case of swath data or similar applications, ANNs may prove to be useful as advisors in interactive applications where human operators will be the final judges in the deciding issue.

5.4 Suggested Future Work

In many instances a swath data set will contain overlapping data samples (obtained from overlapping swaths). In such cases, there is a strong possibility that these depth values, although referent to the same location, will be different. This work did not consider data from overlapping swaths; it only considered data from non-overlapping swaths. It is therefore suggested that further work should be done to address the need created by this common feature of swath surveying systems, and that ANN systems be developed that will be able to handle these cases of multiple and conflicting data.

List of References

[1]. Kohonen, T., "An Introduction to Neural Networks", *Neural Networks*, pp.3-16, 1988.

[2]. Vemuri, V., "Artificial Neural Networks:An Introduction", *Artifiicial Neural Networks:Theorectical Concepts*. Washington,D.C.: The Computer Society of the IEEE, 1988.

[3]. Hebb, D.O., The Organization of Behavior. New York: John Wiley, 1949.

[4]. Pagels, H. R., The Dreams of Reason. New York: Simon & Schuster, 1988.

[5]. Johnson, R. C. and Brown, C., Cognizers. Neural Networks and MAchines that Think. New York: John Wiley and Sons, Inc., 1988.

[6]. Minsky, M. and Papert, S., Perceptrons: Introduction to Coomputational Geometry. Cambridge: MIT Press, Second Edition, 1982.

[7]. Bourne, L. E. and Ekstrand, B. R., Psychology: Its Principles and Meanings. New York: Holt, Rinehart and Winston, Fourth Edition, 1982.

[8]. Churchland, P. S., Neurophilosophy: Towards a Unified Science of Mind/Brain. Cambridge: MIT Press, 1986.

[9]. Rumelhart, D.E., McClelland, J.L., and PDP Research Group, Parallel Distributed Processing. Cambridge: MIT Press, 1986.

[10]. Pao ,Yoh-Han, Adaptive Pattern Recognition and Neural Networks. Addison-Wesley Publising Company, Reading, Massachusetts, 1989.

[11]. Coren, S., Porac, C. and Ward, L.M., Sensation and Perception. Academic Press, Inc., Orlando, Florida, 1984.

[12]. Kohonen, T., Self Organization and Associative Memory. Springer-Verlag, New York, 1988.

[13]. Schwartz, T., "Neural Networks: Capabilities and Applications", The 31st. IEEE Video Conferences Seminars Via Satellite.

[14]. Gorman, R.P. and Sejnowski, T.J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", *Neural Networks*, Vol.1, pp. 75-79.

[15]. O'Shea ,T. and Eisenstadt, M., Artificial Intelligence, Tools, Techniques and Applications, Harper & Row, Publishers, New York, 1984.

[16]. de Moustier, C. and Kleinrock, M., "Bathymetric Artifacts in Sea Beam Data: How to Recognize Them and What Causes Them", *Journal of Geophysical Research*, Vol. 91, No. B3, pp. 3407-3424, March 10, 1986.

APPENDIX A

This appendix contains the description and operating instructions of SWANN (Swath Artificial Neural Networks).

Also included in this appendix are the descriptions of the software tools associated with SWANN, such as XPAT (extract pattern) and DRAWPAT (draw pattern) and others.

APPENDIX A

A.1 Swath Artificial Neural Networks (SWANN)

Swath Artificial Neural Networks (SWANN) is a software system aimed to investigate the feasibility of using Artificial Neural Networks to detect peak artifacts and look for user-specified topographic shapes and profiles in PLT swath data files.

SWANN was developed on a SUN 4/150 computer under the SunView environment. It is recommended that users be familiar with this environment as well as with the terminology used in swath data processing.

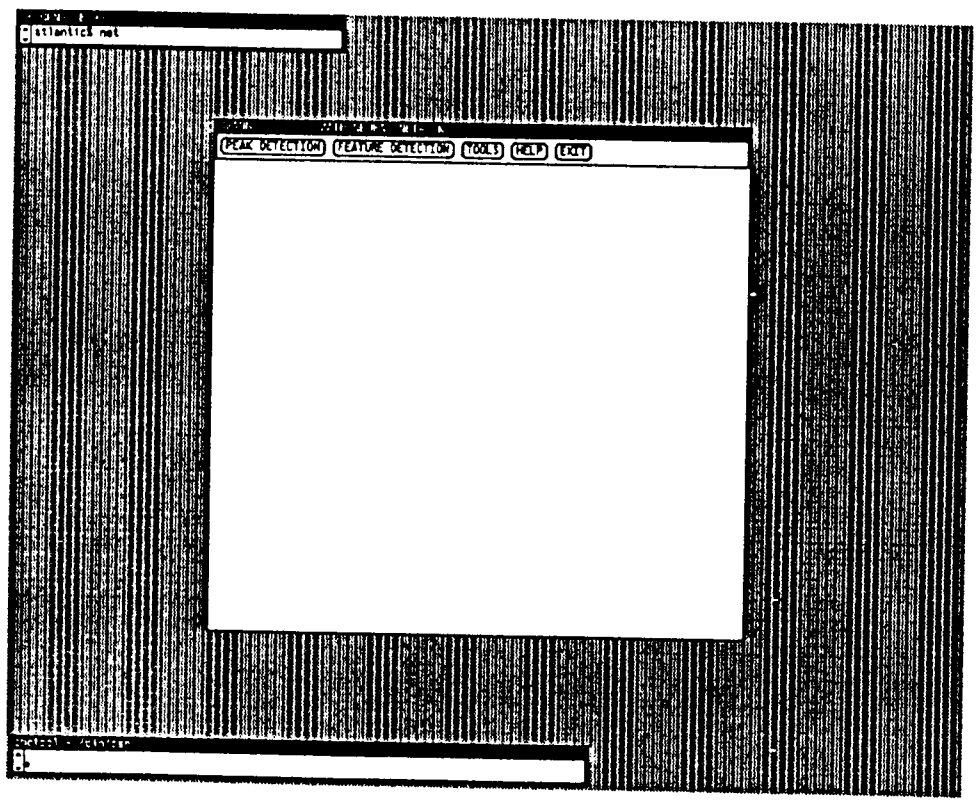


Fig. A1 The base-window of SWANN

A.2 Overall System Description

There are four basic modules in SWANN: Peak Detection, Feature Detection, Tools and a Help utility (see buttons in Fig. A1). Peak detection can be performed by two different methods: Lateral Inhibition and Neighborhood Averaging. The Feature Detection module is a Back-Propagation Neural Network with two hidden layers able to learn complex topographic shapes and search for similar shapes in swath (PLT) data files. The Tools module contains auxiliary tools to help users perform secondary tasks such as shape specification, visualization and extraction of patterns (portions of ocean floor) for network training purposes, and tools that allow for a direct interface between SWANN and other software systems such as SWATH VIEWER. The Help module can be called upon at any time during a work session for assistance on a given task. The operation of any of these modules can be initiated by pointing and clicking (with the mouse) on the appropriate software-button in the top control panel. Fig. A1 the starting (or base) window of SWANN.

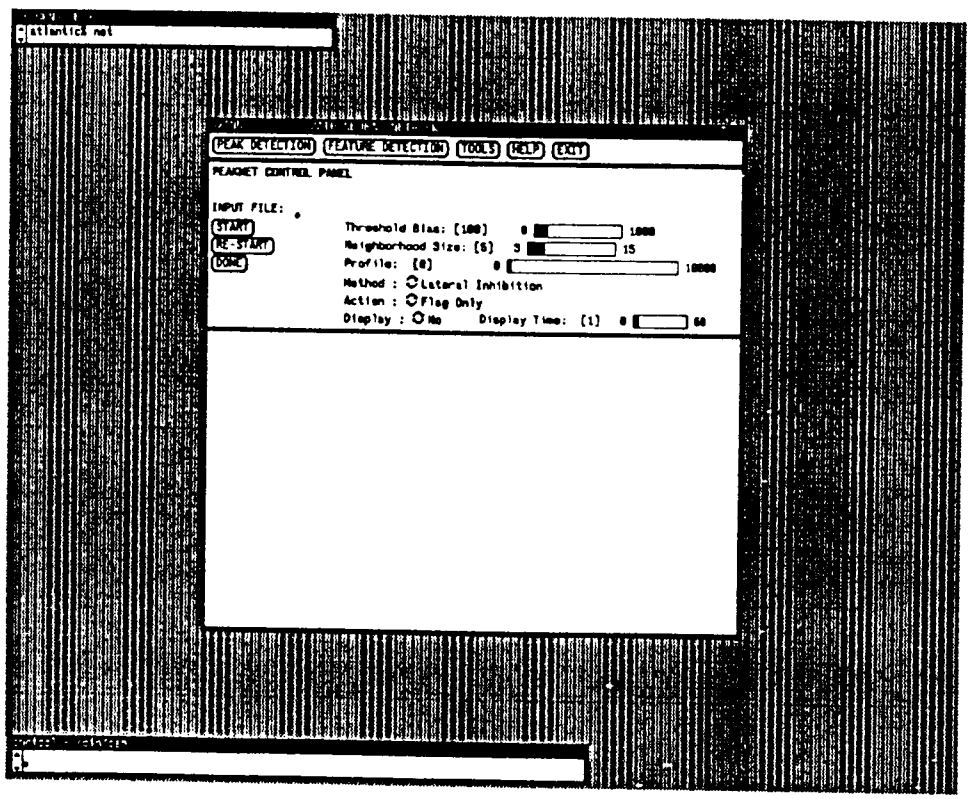


Fig. A2 The Peak Detection Control Panel

A.3 Peak Detection Operation

Clicking on the PEAK DETECTION button, located in the control panel of the base window, displays the PEAK NET CONTROL PANEL

The PEAK-NET CONTROL PANEL (refer to Fig. A2) contains the interface buttons, sliders and switches used to initialize necessary parameters before program execution starts. All of these, but the INPUT FILE button, which must be supplied by the user, have pre-set default values that may be changed, if desired.

The parameters needed for peak detection are the following:

1- The INPUT FILE is the name of the PLT swath data file to be processed.

2- The THRESHOLD BIAS is a constant value that is added to the local bias, dynamically computed by the program, to produce a compound threshold. A point is accepted as a "good" point if its depth is within the point's neighborhood average value plus or minus this compound threshold, and rejected, or flagged as a "bad" point, otherwise.

3- The NEIGHBORHOOD SIZE parameter specifies the size of the side of the square area, centered on the current data point, to be considered to verify if a suspect data point is to be accepted or rejected. The default is 5 (5x5 or 25 points.)

4- The PROFILE slider is used as a counter to monitor the progress of program execution. This slider can not be set.

5- The METHOD switch should be toggled to the desired operational method: Lateral Inhibition or Neighborhood Averaging. The Lateral Inhibition Method is based upon the edge detection and lateral inhibition mechanisms found in many biological systems, this method is particularly efficient in detecting isolated or single peaks and it offers fast execution time. The Neighborhood Averaging method uses the PDNet, a detection system that uses both neural networks principles and more traditional computational methods.

6- The ACTION or desired results of the program may be set to "Flag Only" or to "Flag and Correct. In the "Flag Only" mode, bit 7 of the status byte associated with each data point is set to 1 if the point is a bad data point, and to 0 if the point is accepted. In the "Flag and Correct" mode, both bit 7 of the status byte is set, and the depth data point is replaced with the suggested "correct" value. To preserve the integrity of the raw data the original data file is never modified. The program creates a new data file in the current directory with the extension '.pnt' (for Peak-Net Tested).

a) The TASK FILENAME specifies the name of the file containing the training set (the set of patterns that the network is expected to learn.)
b) The NUMBER OF INPUT UNITS is generally equal to the number of elements in the vectors that represent the patterns.
c) The NUMBER OF OUTPUT UNITS is generally set to the number of classes of patterns in the training set.
d) The NUMBER OF SAMPLES TO LEARN is generally set to the total number of patterns or samples in the task file.
e) The MAX NUMBER OF LEARNING CYCLES is used to stop the learning process when learning is not successful (the error will not decrease) or taking too long (the error decreasing too slowly).
f) A record of the system-error during learning (every 10 cycles) may be obtained if the CREATE ERROR FILE is set to "yes", and a display of the input-data in the task file may be obtained before processing starts if DISPLAY DATA is set correctly.

After all this information has been specified, the user must press the DONE button to return to the BACK-PROP CONTROL PANEL and press the START button to start the Learning Phase.

Learning may be aborted by keeping the mouse's left button pressed until the program halts.

specify the number of units, "neurons", or processing elements for the two hidden layers.

2- The MOMEMTUM or "inertia" of the network during learning must be set together with the LEARNING RATE. For general purpose applications, the default values of 0.9 for the momentum and 0.7 for the learning rate are the recommended values.

3- Also the maximum allowed total system and unit errors must be set. The default values are 0.01 for the system-error and 0.001 for the unit-error. These values are needed to stop the learning process.

4- The PROFILE slider is not used in the learning mode. It can not be set.

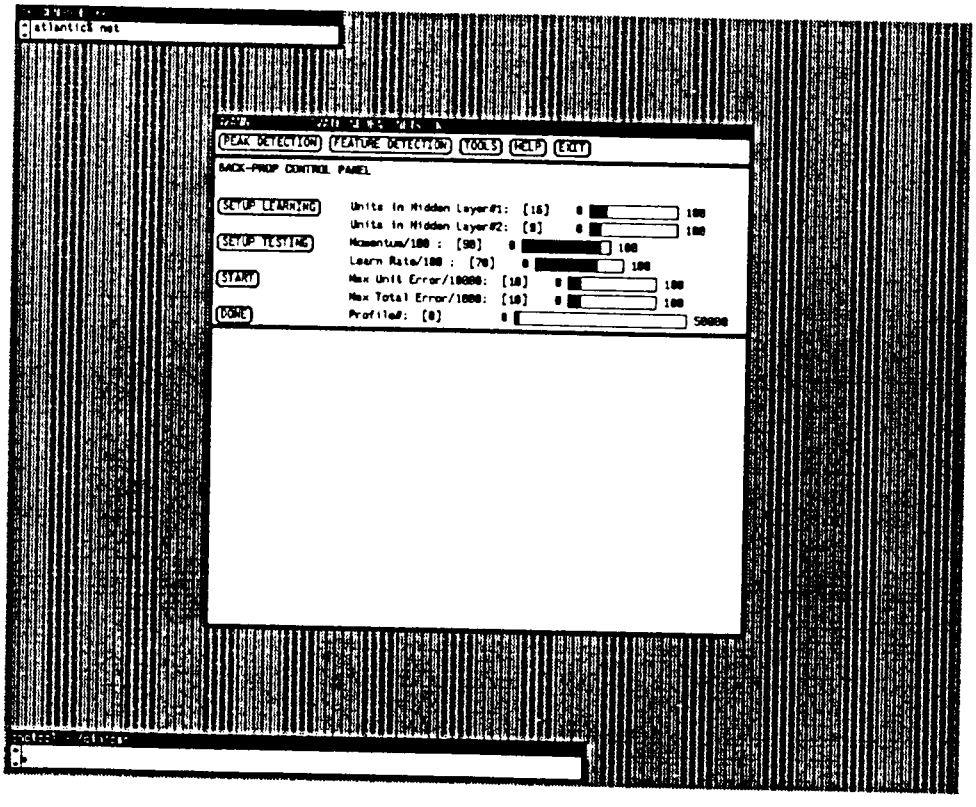


Fig. A3 SWANN's Feature Detection Control Panel

After these values on the BACK-PROP CONTROL PANEL are set, clicking on the SETUP LEARNING button pops a second window asking the user for the parameters which are specific to the "task" to be performed (see Fig. A4).

7- The current profile being processed can be graphically displayed together with the dynamically adjusted thresholds and biases by setting the DISPLAY switch to "yes" and the DISPLAY TIME set to the number of seconds for each individual display. This is possible in Neighborhood Averaging method ONLY!
8- Execution starts when the START button is pressed. To ABORT processing one must keep the mouse's left button pressed until the program halts.
9- The RE-START button should be pressed to re-start processing on a given data file.

At the time of this writing, two files are created as result of Peak Detection processing:
a) A '.flagged' file -- an ASCII file storing the profiles containing bad data points --, and
b) A '.pnt' file -- a binary file identical to the original data file with the exception that bit 7 of the status bytes of bad data points have been set, and the depth of these data points may also have been changed (if the execution was performed with the ACTION switch set to "Flag and Correct".)

A.4 Feature Detection

Topographic Feature Detection is performed by a four layer back-propagation neural network (BPNet).

Before the BPNet is able to search for and find some desired topographic features, prototypes of these features must be repeatedly presented to the net until it has learned to classify them correctly. This is done in the BPNet Learning Phase.

A.4.1 Network Learning

Before Learning starts, the parameters shown on the BACK-PROP CONTROL PANEL must be specified.
1- The architecture of the two hidden layers are set through the two sliders UNITS IN HIDDEN LAYER#1 and UNITS IN HIDDEN LAYER #2. These sliders

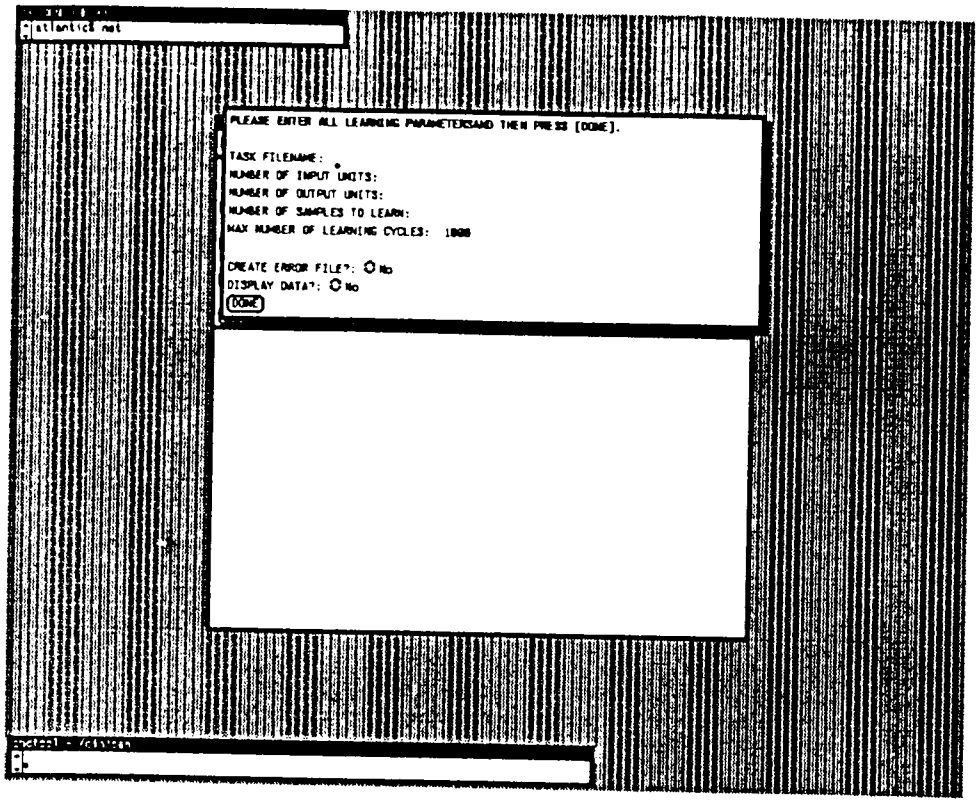


Fig. A4 The Learning Subwindow.

A.4.2 Network Testing or Recall

Different parameters, mainly related to the task to be performed, must be set before the BPNet is able to start testing data. This is initiated by clicking on the SETUP TESTING button located in the BACK-PROP CONTROL PANEL.

When the SETUP TESTING button is pressed, a secondary window appears asking for the following information (refer to Fig. A5) :

1- The TASK-FILE NAME is the name of the file that was used to train the network for this particular job.

2- The TEST-FILE NAME is the name of the file that contains the patterns to be tested.

3- The NUMBER OF SAMPLES TO TEST is used to specify the number of samples or patterns from the test-file that are to be tested.

4- Through the TEST FILE IS switch one can specify whether the test-file is a swath data file or any other type of file. This flexibility allows the use of BPNet with other tasks besides swath data tasks. If the file specified is a swath data file, a second switch is used to specify whether the swath data file contains single profile patterns or multi-profile patterns.

Once all these parameters have been seleted, pressing the DONE button returns the user to the BACK-PROP CONTROL PANEL, where clicking on the START button will initiate the testing.

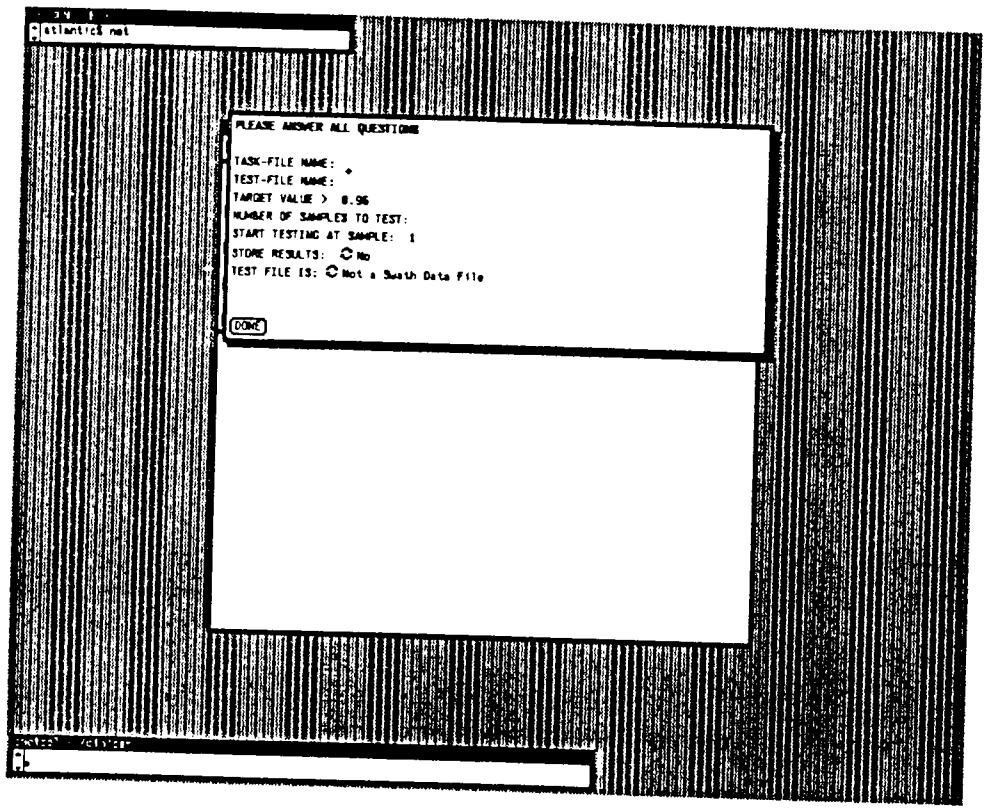


Fig. A5 The Testing Subwindow.

A.5 SWANN Files

A.5.1 Format of Training Files

Training or task files contain a set of samples to be presented to the net during training (or learning). These files must have the extention '.dat' (i.e. 'samples1.dat').

Each pattern-vector in a task file must be followed by the target values -- the values that the net is expected to produce as its ouptut.

Assume the training of anetwork with 5 input nodes and 2 output nodes, and assume a task (training) file containing patterns from two different classes. The patterns are represented by 5-element vectors. The first and third patterns (10 20 39 27 4) and (11 22 37 25 7) belong to one class, and the second and

fourth patterns (20 2 44 2 2) and (19 0 48 1 0) belong to the other class. Assuming that the network is to be trained to produce an output (0 1) and (1 0) to indicate whether a pattern belongs to one or to the other class, then the correct format of the 'samples1.dat' task file would start as follows:

```
10 20 39 27 4
0 1
20 2 44 2 2
1 0
11 22 37 25 7
0 1
19 0 48 1 0
1 0
.....
```

Note that for faster and easier learning, patterns of different classes should alternate position in the file, as in the example above. It is not recommended to have task files where all patterns of a class are stored together.

A.5.2 Format of Test Files

The format of the test or recall files is identical to the fomat of the learning files with the exception that the target values are excluded. Recall files only contain data patterns. Using the same patterns of the example in the previous section, a portion of a recall file could look as follows:

```
.....
20 2 44 2 2
11 22 37 25 7
10 20 37 27 4
19 0 48 1 0
.....
```

A.5.3 Files Created During Learning

Two files are created during learning, which store information later needed by the nework during testing. After learning a task, the program stores the topology of the network in a file with the suffix and extension '_v.dat', and stores the weights obtained during learning in a file with the sufix and extension '_w.dat'.

The format of the '_v.dat' file is as follows:

```
n  no  ni  m  lr  nhl  mxit
n1  n2  n3  ...
nit  err
```

where

n - number of training samples
no - number of output nodes
ni - number of input nodes
m - momentum
lr - learning rate
nhl - number of hidden layers
mxit - max number of iterations
n1...nn - number of nodes in each layer
nit - number of specified iterations
err - specified error

If desired, an error file may also be created. To obtain an error file, the user must toggle the appropriate switch, located in the learning subpanel, to "yes".

The error file stores the normalized system-errors observed every ten iterations during learning, providing a record of the learning progress of the network.

The error file has the name of the task file with the extention '.criter'.

A.5.4 Files Created During Testing

To keep a record of the patterns identified by the network and its performance during classification a file may be created if desired.

A.6 SWANN Tools

To help users perform different tasks with ease, a set of software tools is also available. The tools available to SWANN users are the following: "Draw Profile", "Extract Pattern", "View Map" and "Update Files". The operation of these tools is discussed in the following subsection.

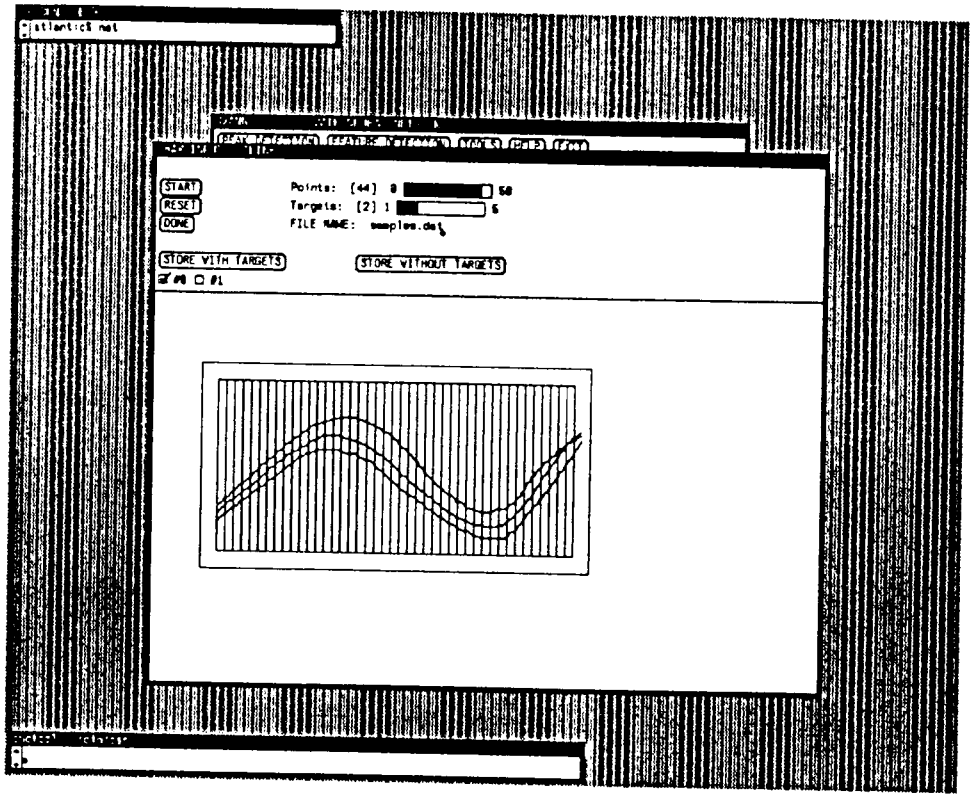


Fig. A6 The Draw-Pattern Tool.

A.6.1 Draw-Profile Tool

Fig. A6 shows the control panel of the Draw-Profile Tool. This tool enables users to easily build both training and testing data files for the BPNet.

To build a file with training patterns:

1- Select a file name.

2- Select the number of data points in a profile using slider POINTS.

3- Select the number of classes using the slider TARGETS.

4- Press the START button to display the canvas where the picture of the profile will be drawn. Note that this canvas contains a number of vertical lines that corresponds to the number of profile data points. The canvas is surrounded by a border.

5- Draw a profile by dragging a the mouse across all the vertical lines, starting at the left side and ending at the right side.
6- If the profile looks satisfactory, it can then be stored together with the class to which it belongs by first selecting the class, and then clicking on the STORE-WITH-TARGETS button. If the profile does not look satisfactory, the canvas may be erased by clicking on the RESET button and a new profile may be drawn on a clean canvas.

This tool allows for the drawing of both single and multi-profile patterns. For example, to draw an area composed of ten profiles, the first nine profiles are drawn and stored without targets and finally the tenth -- the last -- profile is stored with targets.

These steps are repeated until the desired number of patterns is obtained.

The steps to build a file with testing patterns are identical to the steps to build a training file with the exception that all patterns are stored without targets using the STORE-WITHOUT-TARGETS button.

A.6.2 XPAT Tool

Fig. A7 shows the first screen of the XPAT tool. This tool was designed to help users to extract patterns from existing swath files files and create a file with these patterns.

XPAT is started by selecting the EXTRACT PATTERN option from the TOOLS menu. The first screen displayed by XPAT is necessary to determine the filename and its size.

To create a file with training patterns extracted from a swath data file follow the steps below:
1- Select the DATAFILE name.
2- Select the TASKFILE name (the name of the training file that will contain the set of training patterns).
3- Select the PATTERN SIZE.
4- Select the number of TARGETS or pattern classes.
5- Press the START button.

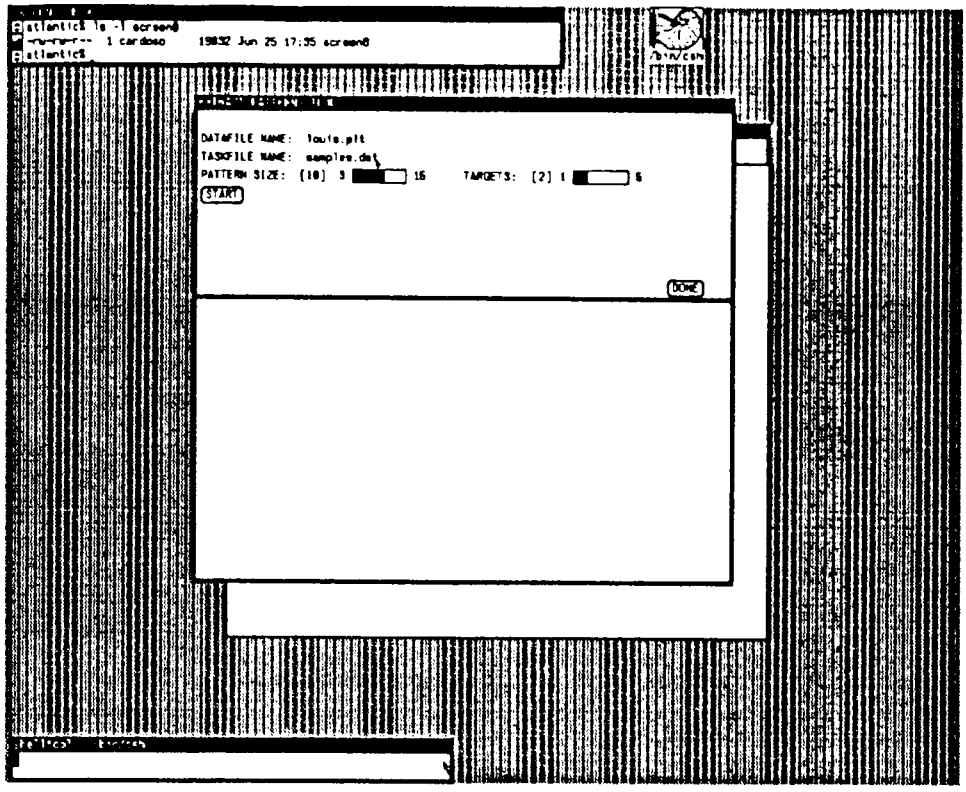


Fig. A7 XPAT Set-Up Screen.

At this stage a second panel appears, with a canvas were the patterns will be displayed (see Fig. A8).

To see a pattern (some area of ocean floor) follow the steps below:

1- Use sliders START PROFILE and ADD TO PROFILE to select the number of the first data profile that stores the pattern. The sum of these two sliders will select the desired data profile.

2- Use the START AT RECORD slider to select the the first record (the first data point) of the profile selected in step 1, to be displayed. At this stage a 3D image of the desired area will be displayed on the canvas.

3- Select the target, or class, for the pattern by ticking the appropriate class #.

4- Press the STORE button to store the pattern in the training file.

5- Repeat steps 1-4 until finished, then press DONE.

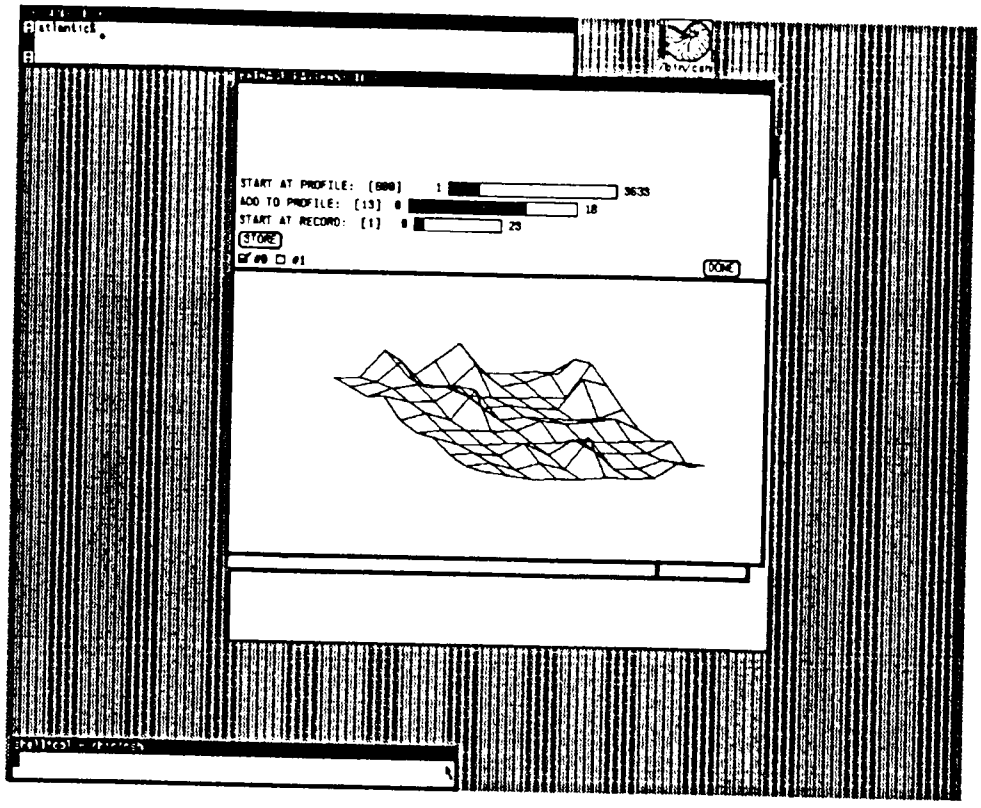


Fig. A8 XPAT Display Canvas and Control Panel

A.6.3 Other Tools

SWANN offers users other software tools to facilitate system operations such as VIWE-MAP, SHOW-PLT-FILES, and SHOW-PROCESSED-FILES.

VIEW-MAP allows for a quick link from SWANN to the SWATH-VIEWER, an external software tool that displays swath data files in a variety of ways. This interface is very useful to check the results of the PDNet.

SHOW-PLT-FILES is used to display a list of all PLT files in the swath files directory. Swath data files have long and strange names; selecting one of the files in the list, using the selection technique of SUNVIEW -- highlighting the string of text representing the file name -- will automatically load the name of the file into the PDNet data filename slot saving time and the possibility of typing errors.

UPDATE-FILES produces the most recent list of PLT files in the swath file directory, and SHOW-PROCESSED-FILES will display on the screen a list of the processed files (or previously tested by the peak detection networks) existent in the current working directory.

APPENDIX B

This appendix contains the SWANN source code used to implement the Edge Detector using the Lateral Inhibition Metho.

Appendix B

```
#define  M  N+2  /*N is the profile size*/
extern int x[N]; /*input data vector*/

int edge(p, tbias)
int *p;      /*used by calling program*/
int tbias;  /*threshold*/
{
int i;
int corrupted;
int u[M];
int z[M];

/*init ializeflag */
corrupted = 0;

/*compute lateral inhibition*/
for(i=0; i<N-1; i++)
        {
        u[i+1] = x[i+1] - x[i];
        }
u[0] = x[0] - x[1];
u[M-2] = x[N-2] - x[N-1];

/*find zero crossings*/
for(i=1; i<M-1; i++)
        {
        z[i] = ((u[i] * u[i-1]) < 0)? 1 : 0;
        }
/*set and return points*/
for(i=1; i<M-1; i++)
        {
        if (u[i] < 0)  u[i] = -u[i];
        *p = z[i] && (u[i] > tbias);
        if(*p)  corrupted = 1;
        p++;
        }
return(corrupted);
}
```

CURRICULUM VITAE

Candidate's Full Name:	John Pinto Cardoso
Place and Date of Birth:	Lisbon, Portugal, Sept. 28th, 1938
Permanent Address:	115 Eglinton St. Fredericton, New Brunswick, Canada E3B 2V9
Schools Attended:	1944-1948 Escola Publica de Azambuja Azambuja, Portugal 1949-1953 Liceu Camoes Lisbon, Portugal 1953-1956 Externato de S. Joao de Brito Lisbon, Portugal
Universities Attended:	1957-1961 Electro-Mechanical Eng. Technologist Instituto Industrial de Lisboa Lisbon, Portugal 1977-1986 Bachelor of Science (C.S.) University of New Brunswick Fredericton, N.B., Canada
Publications:	Learning and Adaptation in Artificial Neural Networks, Proceedings of the 2nd UNB Applications of Artificial Intelligence, Workshop. Hugh John Flemming Forestry Centre Fredericton, N.B., Canada. October 3rd, 1989. pp. 116-125.