(1)

DYNAMIC ANALYSIS OF FEEDFORWARD NEURAL NETWORKS USING SIMULATED AND MEASURED DATA

THESIS

Gregory L. Tarr
Captain, USAF

AFIT/GE/ENG/88D-54

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/88D-54

DYNAMIC ANALYSIS OF FEEDFORWARD NEURAL NETWORKS USING SIMULATED AND MEASURED DATA

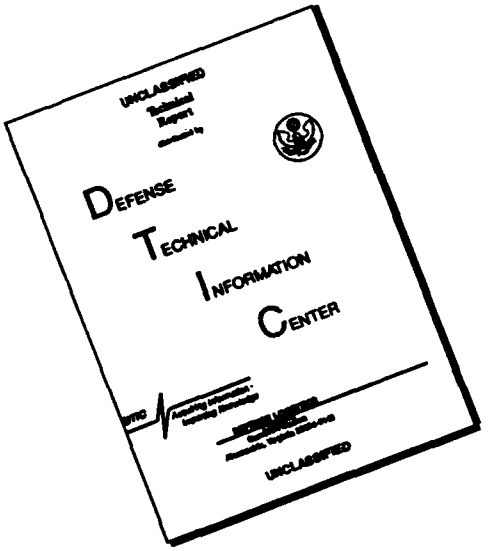
THESIS

Gregory L. Tarr
Captain, USAF

AFIT/GE/ENG/88D-54

Approved for public release; distribution unlimited

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GE/ENG/88D-54

DYNAMIC ANALYSIS OF FEEDFORWARD NEURAL NETWORKS USING SIMULATED AND MEASURED DATA

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Gregory L. Tarr, B.S.E.E
Captain, USAF

December 1988

Preface

The days when a single researcher, working alone in a laboratory could make great contributions to his or her field have past. Those days are part of a bygone era or perhaps never existed at all. Great accomplishment today is the result of team efforts, each individual striving to make a contribution by fitting a small piece into a larger puzzle. This work contains the small piece of the puzzle I have worked on. Such work would not be meaningful or useful without strong leadership and direction from real giants. Dr. Steven K. Rogers, Dr. Matthew Kabrisky and Major Phil Amburn provided that leadership. I would like to acknowledge my appreciation for their help and support. Many thanks to my thesis advisor, Dr. Steven K. Rogers, for his encouragement and enthusiasm. I would also like to thank Dr. Matthew Kabrisky and Dr. Phil Amburn for their assistance and advice throughout the research effort.

Sir Isaac Newton once said "If I have seen farther than other men it is because I have stood on the shoulders of giants". If I have seen at all, it is because I peered over the shoulders of giants.

Gregory L. Tarr

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/88D-54

Abstract

An environment is developed for the study of dynamic changes in patterns of weight and node values for artificial neural networks. Graphic representations of neural network internal states are displayed using a high resolution video terminal. Patterns of node firings and changes in weight vectors are displayed to provide insight during training. Four pattern recognition problems are applied to four types of artificial neural networks. Using simulated data, a simple disjoint region classification problem is developed and examined using a Kohonen net and a multilayer feedforward back propagation (MFB) network.

A MFB neural network is also used to simulate a Fourier filter. Using a Kohonen net, a MFB, a counterpropagation and a hybrid network, data measured from infrared and laser radar imagery of military vehicles is analyzed. The accuracy and training times for a MFB net and a Hybrid net are compared using an ambiguous decision region problem. Each classification problem is examined and compared to classical, nearest neighbor pattern recognition techniques. Using dynamic analysis, neural network pruning is used to determine optimum node configurations. A hybrid neural network is developed using Kohonen training rules for the first hidden layer followed by one or two hidden layers using standard back propagation rules for training. Advantage of the hybrid network is shown for classification problems involving anomalies characteristic of measured data. The Hybrid network requires less training and fewer interconnections than MFB when classifications involves ambiguous decision regions.

DYNAMIC ANALYSIS OF FEEDFORWARD NEURAL NETWORKS USING SIMULATED AND MEASURED DATA

I. Introduction

Autonomous military target detection and classification from electronic imagery is a topic of great importance to the Department of Defense. The solution to the problem may lie in one of several implementations of artificial neural networks. Several topologies for neural networks have been proposed, each of which provide a solution for a narrow class of pattern recognition problems. Some researchers (Huang,1987) feel that combinations of more than one type of neural network may result in a more dynamic and robust system.

Identification and classification of targets from electronic imagery is a difficult problem due to the vast amounts of data involved. A single image can contain millions of bits of information, all of which need to be processed. Processing images for pattern recognition is a threefold problem. First, the targets must be separated from the background or segmented. Second, the data must be reduced to a manageable size, commonly called vector quantization. This reduction in data can be accomplished by selecting specific features of a pattern and using only these features for classification. Good pattern recognition requires good features. The final task is classification of the vectors.

Determining which features of an image form the best description of an object is, in itself, a difficult problem. This study will examine several sets of data collected from a variety of sensors, including laser radar and passive infrared imagery. Baseline

classification analysis will be made using simple feedforward neural networks then extended to new forms of combinations of the feedforward networks.

One of the difficulties encountered when testing neural networks is the lack of good test data. Training a pattern recognition system requires thousands of teaching cycles. Although, one would prefer very large training sets using real imagery, in practice unfortunately, training sets rarely exceed more than a few hundred images due to the difficulty in segmentation and vector quantization. When exploring or testing a particular neural net algorithm, the data problem can be avoided by computer generation of the input vectors, based on the problem description.

Success of a particular classification problem depends on a number of factors. First, consider the validity of the segmentation of the data. Has the actual target been separated from the background data and noise ? Next, is the feature extraction legitimate. Do the features selected for the input vector represent a good description of the target ? Once the target has been extracted from the background, is the vector quantized description unique enough to allow classification ? Finally, is the neural network topology sufficient for the size of the decision region and can it accurately classify input patterns ? Special tools may be needed to answer these questions.

An environment to examine internal operation of the neural networks as they train could help determine the efficiency, and accuracy of different topologies. Evaluation of the internal constants and variables as the network trains may offer insight into which values may be best suited for a particular set of circumstances. Although the primary area of investigation considered here is network topologies, the software package generated as a result of this effort will be a generalized research tool to study segmentation or vector quantization algorithms as they apply to neural network classification problems.

1.1 *Background*

The ability of machines to interpret visual images remains an unsolved problem. Military planners have long been interested in developing means to automatically detect and classify military targets using conventional sensors. Although these sensors, television, infrared scanners or multifunction laser radars, provide enough information for a human operator to find a target, the extension to automatic detection and classification is still impractical using current computer architectures. The computational effort to classify or detect tactical targets from mission sensor data is too demanding to be performed in real-time.

These problems may be resolved by a computer technology called artificial neural networks. Artificial neural networks are computer structures or architectures that attempt to mimic some of the known characteristics of biological brains. Neural networks may provide relatively fast, approximate solutions to problems, as opposed to the slow, exhaustive (but exact) solutions provided by conventional computer architectures. Artificial neural networks may be arranged in a variety of interconnections of data input and outputs along with intermediate levels. These different arrangements characterize the topology of the network. Of the myriad of topologies of neural networks studied, the most directly applicable to the target identification problem, are the multilayer feedforward networks (MFB) using backward error propagation (sometimes called multilayer perceptrons) and the Kohonen maps. These two classes of neural networks have an advantage over many other networks, as both accept continuous data as input (Lippman,1987). Many other network configurations accept only binary data. Although binary data processing is sufficient for some classes of problems such as text recognition, to pick an object out of an image, the system must be able to process continuous valued inputs. The analog data processed by the network may be in the form of correlation peaks, statistical moments, or some other vector quantization of distinguishing characteristics calculated from the original image.

While both the multilayer perceptron and the Kohonen maps are suited to the pattern recognition problem there are drawbacks. The MFB model requires hundreds of thousands of training cycles. Complete training may require several days on a high speed computer. Also, while the number of nodes at the input and output is defined by the problem, optimization of the inner node parameters has not been fully addressed. The number of input nodes is fixed by the size of the input vector and the number at the output relates to the number of classes. Currently, the correct number of hidden nodes is determined by experimentation. Some feel that the number of hidden nodes in the first layer should be three times the number in the input layer. Others feel the configuration of the inner layers should be related to the input data (Baum,1986).

Analysis of data using a Kohonen map only partially solves the problem. The output of a Kohonen map is a two dimensional mapping of a multi-dimensional decision space. By mapping one class into one area of the map and another class into another region of the map, the complexity of the decision regions are reduced. Unfortunately, for a complete solution the map must be interpreted. Currently, that is done by examination. The output of a Kohonen map is simply another type of pattern recognition problem. Huang believes that a Kohonen map may be able to function as a preprocessor for some other class of neural network (Huang,1987). The combination of the Kohonen map and the multilayer perceptron may offer a solution to the weaknesses of both. Using a Kohonen map to organize the data into a two dimensional grid, then feeding the output of the grid nodes to the input layer of a multilayer perceptron provides a mean of interpreting the Kohonen map. The Kohonen map may also reduce the complexity of the decision space for the perceptron. Training time should be reduced since the time to train the weight values for a perceptron is related to the complexity of the decision regions.

1.2 Problem

The largest part of the effort will be devoted to developing a method to display the internal values of the network using a graphic representation that allows insight into it's operation.

By analysis of the dynamic nature of the network, it is hypothesized that methods for optimizing the topology of the network may be devised, and a better understanding of the artificial neural network paradigms would be gained. The problem of general interest is: can two types of neural network topologies be combined to get a synergistic effect greater than the sum of the two separately? The MFB provides a solution to the disjoint decision region problem while the Kohonen map provides a method to organize unclassified data. This thesis effort will examine several methods of combining networks to get improved performance and reduced training time.

1.3 Approach

The final product of the study is a graphics intensive environment for dynamic analysis of artificial neural networks. This software package will allow the user to explore other neural network problems using common problem definition format. By creating a file listing of a number of exemplars patterns (vector quantizations), along with a specified classification, any general classification problem can be feed into the package for analysis.

The package will consist of four types of neural networks accessed from a common menu. The networks are a Kohonen map, a multilayer perceptron, a counter propagation net and a hybrid net. In addition, an error surface demonstration will be included in the menu for tutorial purposes.

Validation and testing of the package is accomplished by using the software to study four neural net problems. The first problem considered is a decision regiou

problem, using both simple and disjoint decision regions. The second problem will consider neural networks as a Fourier filter. The last two problems will use measured image data from the Ruck (Ruck,1987) and Roggemann (Roggemann,1988) data sets.

Using the environment tools, efforts will be made to optimize the number of nodes to the type of data being tested. Each network will be tested for the number of training cycles to convergence, and accuracy. Test sets will be analyzed using each of the conventional nets, as well as classical nearest neighbor classifiers.

The development of a hybrid neural network will be the final step in a process to examine several types of neural networks. The process will include finding ways to display information about the dynamic processes inside the networks.

Chapter II provides background information concerning artificial neural network. Neural networks are discussed in context of the type of problems each topology is intended to solve.

Chapter III discusses the software engineering aspect of the neural network environment. The organization of the software is explained in terms of the various functions and operation of the major modules.

Chapter IV provides the analysis and testing of the software environment. Several common classification problems are analyzed using the system as well as two specific data sets. Each problem or data set is analyzed using one of several common neural network topologies.

Chapter V is an overview of the results of the experiments including general observations applicable to many types of neural network problems.

II. Literature Review

Real-time target recognition for intelligent weapons systems is too computationally intensive to be practical using current computer technology. Picking a legitimate military target from video sensor information requires so many calculations that identification and classification may requires hours while only seconds are available. The brain, a biological computing engine, can solve these type problems quickly. A novel computer architecture based on the way the brain is thought to function, may provide a solution to classification problems. This architecture, called artificial neural networks, may provide an alternate approach to conventional computers and artificial intelligence paradigms for target recognition tasks.

This section will discuss the origins of the artificial neural networks, and the relation between the biological inspiration and the computer implementations. Starting with the basic building blocks of neural computers, the node, the difference between natural and artificial intelligence will be discussed in terms of abstract reasoning verses brute force calculations. Several topologies of node interconnections will be discussed to demonstrate the diversity of programming or training techniques. Finally, this chapter will discuss the characteristic of problems, in terms of disjoint and ambiguous decision regions, which affect selection of network topology and training rules.

2.1 Historical Perspective

Greek philosophers, including Plato and Aristotle, offered theoretical explanations of the brain and thinking process a few thousand years ago. Heron the Alexandrian built a hydraulic automata around 100 B.C. Abstract, conceptual information processing operations have been performed by mechanical devices for a few centuries, for example the slide rule.

The first theorists to conceive of a computer based psychological models and neurophysiological research were W.S. McCulloch and W.A. Pitts in the early 1940's (Kohonen,1987). Although early research efforts paralleled the development of the modern computer (the Von Neumann Machine), the concept of a cybernetic machine has been popular since ancient times. These types of computers were first called connectionist machines and later artificial neural networks.

Artificial neural networks have been studied for several decades. At first they were studied in connection with psychological theories and neurophysiological research. By 1960, many implementations of "neural computers" had been developed. Today, neural computers may offer a solution to the growing need for a machine that will perform not only calculations, but actually make judgments. A fundamental limitation of conventional computer architectures is the inability for abstract reasoning. Based on the biological model, neural network computers attempts to overcome that weakness.

2.2 Natural vs Artificial Intelligence

Although it may be impossible to determine exactly how the brain works, observation of its behavior may be enough to determine why humans are good at solving types of problems that cause computers great difficulty. Identifying a face in a crowd, or navigating through a room without bumping into the furniture, requires an effort beyond the capability of modern computers. Yet, even children learn to identify their parents after only a few months and the smallest insect is capable of solving navigation problems beyond the capabilities of advanced robotic systems.

Bruce D. Shriver noted:

> ...digital computers are extremely good at executing sequences of instructions that have been precisely formulated for them with stored programs representing the processing steps that need to be done. The human brain, on the other hand, performs well at such tasks as vision, speech, information retrieval, and complex spatial and temporal pattern recog-

nition in the presence of noisy and distorted data-tasks that are very difficult for sequential digital computers to do at all (Caudill, 1987:48).

Artificial intelligence is a sequential process which involves collecting all available data and systematically processing each piece until the solution is established by completing an algorithm. Natural intelligence differs from artificial intelligence by using a system to extract relevant information from the available data, then extrapolate an approximate solution. In terms of the target classification problem, the process would organize the data (self-organization), extract relevant information (feature extraction), and compute a solution (classification).

The basic building blocks used by the brain to perform these calculations are called *neurons*.

2.3 The Neuron

. Neurons are the brain's electrochemical processing elements which allow individuals to "store, represent, retrieve, and manipulate data such as images, smells, sensations and thoughts" (Caudill, 1987:48). Neurons are slow by digital electronic standards. Response times are measured in hundreds of milliseconds as opposed to the nanosecond response time of today's integrated circuits. Still, the massive number of neurons may make up for the lack of speed. The brain is estimated to use between 10^{10} and 10^{11} neurons (Kohonen,1987:227) in an intricate, interconnected structure. Many neurons are connected to thousands of other neurons. Each neuron reaching out to other neurons by means of a single output, the axon, a fiber-like structure that attaches to other neurons by a synaptic terminal (Kohonen,1987,:210-240). This connection between the neurons performs a small, imprecise multiplication of the signal transmitted down the axon to the synaptic terminal. These signals are accumulated by the receiving neuron. When the sum of these signals reach a sufficient level, the neuron fires a weak electrical signal down its own axon to the next neuron in the chain.

The process causing the neuron to fire is the most powerful aspect of neural computing. Each of the axons provide an adaptable weight which can inhibit/excite the signal between neurons. The axons are similar to the memory unit in a digital computer with one exception; they are able to adapt and change with time. The mechanism which modifies these weights enables learning. By duplicating this process artificially, we may be able to build computers that learn by example, rather than loading a stored program. The Japanese, in their announcement of the fifth generation computers, "coined the term natural vs artificial intelligence" (Caudill,1987:46).

What kind of process or how the process allows people to think, walk, or recognize objects, is not understood. That this process contains the solution to the pattern recognition problem is demonstrated by our own ability. Whether a computer can reproduce this ability is the subject of many research efforts. Using a neural computer architecture, the computer may be able to discover, on its own, the underlying structures in a given set of input data which makes recognition of specific patterns possible (Kohonen, 1981:214-215).

The biological neuron is the basic computation unit of the brain and provides the inspiration for the neural network node. The node is the basic computational element of artificial neural networks.

2.4 The Neural Network Node

Advances in many fields have made possible faster and more powerful computers than ever before. Unfortunately, even the most sophisticated computers are tied to a structure which allows them to perform only a single digital operation at a time. These register functions, multiplication of two numbers, storing results or getting more data, must be performed sequentially, one register at a time. Although the speed of these simple computations is limited only by the speed of the material and design technology, there is only a single path for the data flow.

This limitation is known as the "Von Neumann Bottleneck" (Hamacher,1986).

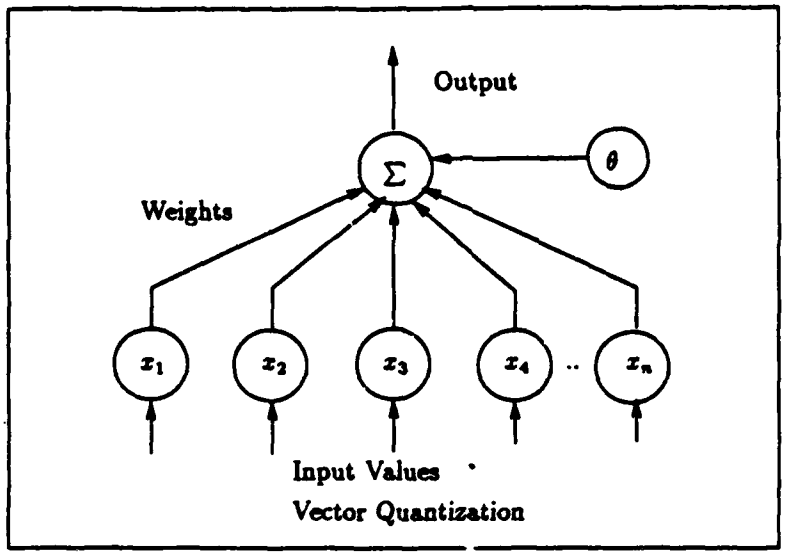


Figure 1. A Neural Network Node

Because each calculation must pass through a single point, the central processor, data flow is restricted, like a bottleneck. This obstacle may soon be overcome by an architecture which uses a massively parallel approach to problem solving. These structures, called Neural Networks or Connectionist Machines, are based at least partially, on the way the human brain is thought to function. The simplest processing element of a neural network is the node. Schematically, a node is composed of a multitude of inputs and one output.

In early neural networks, the output node value was either zero or one, based on thresholding the dot product of the input vector and the weight vector. A neural computer is composed of from several to a few thousand nodes.

Dr. Robert Hecht-Nielsen describes the first commercial neurocomputer as

> "a computing system made up of a number of simple, highly interconnected processing elements, which processes information by its dynamic state response to external inputs" (Caudill,1987).

By connecting the output of some nodes to the inputs of other nodes, large arrays or networks are formed which seem to be capable of computing abstract decisions. Each processing element computes, in parallel with all other nodes, a simple *true/false* decision based on the weighted sum of the inputs. These basic decisions form a basis for broader abstract reasoning.

Analysis of the differing methods to arrange the interconnection between nodes together with the rules to establish the interconnection weights makeup a large part of the study of artificial neural networks.

2.5 Artificial Neural Networks

Study of neural network models is usually conducted on standard digital computers. Several of Kohonen's models were first written in Turbo Pascal for the IBM personal computer (Kohonen, 1987:14). These models take the form of a computer program. The various parts of the neuron are modeled as data structures in digital memory. Although many of the advantages of using neural nets are lost by using a "Von Neumann" machine, the research process may develop algorithms later suitable for dedicated hardware.

Developing a neural net is a three step process. First, the data structures are organized in memory, either as arrays or link lists. There are two types of data structures, the nodes and weights. The nodes are simple accumulators which occupy the number of memory locations or bytes required for one floating point number. Nodes can be considered the "neurons" of a neural network. The other data structure, the weights, are analogous to the axons. They occupy an array of floating point memory locations, usually a two dimensional matrix of floating point numbers. The second step is to determine the adaptive weight values using a training algorithm. These can be either supervised, that is, a priori knowledge of the input pattern, or unsupervised, using unclassified random inputs. Finally, using a black box approach, data is presented to the inputs, propagated through the system, then

read at the outputs (Kononen,1987:13-17).

Like the neuron in biological systems, the node is the principle unit of artificial systems. A node value is formed by mathematically manipulati the input value and the connection weights. Exactly how, is dependent on a particular neural network topology. Every element of the input vector is connected to every node. Many arrangements or taxonomies of nodes and weights have been developed by Hopfield and others. See Lippman's overview (Lippman, 1987:7-13).

Each type of neural network is applicable to a specific class of recognition problems. Only two of these networks relate to the problem under investigation, Kohonen maps and the multilayer feedforward backward error propagation nets (MFB), sometimes called the Multilayer Perceptron net. These two neural network topologies are suitable to the combination network problem because they allow continuous data to be used as inputs. Most other types of networks accept only binary data (Lippmann, 1987:7-13). The Kohonen map was selected for its ability to organize data into rational decision regions, while the MFB was selected for its ability to classify data contained in complex decision regions. Classification efforts could be reduced if relevant features which differentiate the exemplars, could be extracted from the entire data set first.

This is the principle of self-organization. Self-organization is a form of unsupervised learning. In unsupervised learning, the classification of a particular training exemplar is not used in training process. Comparison of the actual output with a desired output, essential to backward error propagation, is not made. Self-organization on compares each exemplar with all the other exemplars. Unsupervised learning provides an organization of the data which reduces the complexity of the classification problem.

2.6 *Unsupervised Learning*

Kohonen felt that the pattern recognition problem could be simplified if the algorithm could somehow automatically extract relevant features from the input data (Kohonen, 1981). Kohonen's work differed from earlier work, because he was looking for an organization of neuron-like units which could automatically detect common characteristics of the input data without regard to a specific classification. As he was trying to do speech recognition, his data set consisted of many unlabeled examples. Kohonen was using input vectors made by spectral decomposition of speech to train his mapping networks. These examples were presented to the recognizer in hopes that the mapping algorithm could sort out the various vowel and consonant sounds. The Kohonen map tries to construct a map or "hierarchical clustering" of patterns with the same characteristics. He called this feature extraction the "first step to all perception"(Kohonen 1981:2). To do this, he developed an unsupervised learning scheme which pushes one class of data to one part of the map while another class would appear on another part of the map. These "feature maps" are a spatial clustering of input samples with similar characteristics.

2.7 *Kohonen Feature Maps*

A Kohonen map has a unique structure. A layer of at least two input nodes is required. Weights connect the inputs to a layer of output nodes. Every input node is connected to every output node. This output layer is constructed in the form of a grid. Fifteen by fifteen nodes is a common arrangement (Barmore,1988).

The map is trained by using distance measurements between the input data and the weights. The difference is fed back to the system to reduce the distance for inputs with similar characteristics. When data is presented to the input nodes, the Euclidian distance between an input and a output is calculated. This distance is calculated by considering each output node in turn. First, calculate the sum of the square of the differences between each input node and the weight. After considering

each output node, one distance will be lower than any of the others. This node is considered the "winner." The weights connected to this node are adjusted slightly so the distance is a little smaller. Also, each node in a small neighborhood around the "winner" is adjusted so that any similar data will appear in this region. As the training continues, the neighborhood around the winner is reduced until only the single node is updated. By the end of the training process, under non-pathological conditions, similar exemplars will cause a node to fire in small region of the map. Classification is a matter of determining which region of the map is activated when data is presented to the inputs.

Notice that the map is never told what pattern it was training on. The net simply organized dissimilar data into different regions of the map. Kohonen classification is based on visual inspection of the feature map. To automate the classification process, a new type of learning process is required.

2.8 Supervised Learning

Another class of training algorithms uses a supervised training technique. With this method, a pattern is presented to the input of a neural net, then the difference between the actual output and the expected output is calculated. This measure of error can be used to adjust the weights. This is called "error correction" or "back propagation". Back propagation is the principle behind the single and multilayer feedforward (MFB) model neural network.

2.9 The Single Layer Perceptron

While the Kohonen model uses unsupervised learning to organize data, the specific classification of an exemplar input is important to the training of the MFB model. The back propagation scheme trains the neural network by example. This algorithm begins with the connection weights set at small random values. Data are presented to the input and the error at the output is measured. The gradient of the

error is calculated, and the weights are adjusted a small amount to reduce the total error. New data are presented over and over again, correcting the connection weights a little each time. Eventually the weights, under sufficient conditions, will converge to a solution. When data are presented to the input, a classification value will appear at the output. By arranging one set of input nodes to feed many output nodes, many classes of patterns can be identified by detecting either a zero or a one at the output. Unfortunately, the work of Sun (Sun, 1986) and others (Lippman, 1987) have shown that the Single Layer Perceptron can only make correct classifications in very simple decision regions. A number of complications can affect the performance of a single layer perceptron. Different classes of data being meshed in a small decision region are one example. The next section discusses a class of problems which cannot be solved by a single layer perceptron: the disjoint region problem, sometimes called a disjunctive learning region.

2.10 The Disjunctive Learning Region Problem

A common example of a disjoint region problem is shown in figure 2.10. This example is sometimes called the *exclusive – or* problem.

Assume a specific pattern to be in a bounded region in the x-y plane. The input vector can be generated by selecting one of the regions, and adding random noise. The output vector, or desired classification of the input vector is determined by noting the location of the x-y pair in the plane. The example can be extended to include any number of disjoint regions in either two-space as described above, or up to *n-space* with *n* elements of the input vector.

Additionally the output vector can describe any number of classes. The simplest example could be to relate one region to one class. The disjoint region problem concerns the case where two regions in the decision space make up a single class. Large, even infinite data sets can be used when the particular problem being investigated allows computer generated input vectors such as decision regions in space.

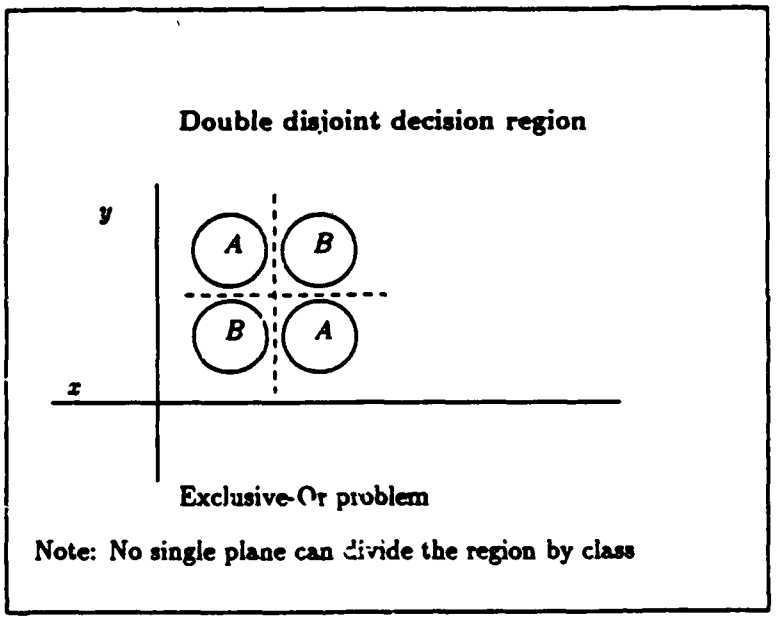


Figure 2. The Disjoint Region Problem

The disjunctive learning region problem stems from a pattern recognition situation in which the decision space may be split into several regions. Even though two or more regions in the decision space may be widely separated, the regions may represent the same class of patterns. When training a net under these conditions, the convergence time is greatly increased and may not be possible at all. Valiant notes:

> "Simple rules of thumb may be hard to learn. There is evidence that certain significant ... classes [may be] rendered computationally intractable" (Valiant,1983)

He was referring to the fact that generally "rule of thumb" type decisions fall into this class and contain disjunctive learning regions. Although, in some cases the network can learn the disjunction in the decision space, the training time may be unacceptable.

The performance of the perceptron can be improved by adding additional layers. Single layer perceptrons cannot solve disjunctive learning region problems. By adding an additional hidden layer the perceptron model can be use to solve disjunctive learning regions. The next section explains the multilayer perceptron.

2.11 Multilayer Perceptron

Adding layers to the perceptron model can allow for an increase in the complexity of the decision regions. The input to a perceptron can be considered as a point in a decision space with one input node for each dimension of the decision space. The weight vector defines a plane (or hyper-plane for n-dimensional space) dividing the decision space into two regions (Lippmann,1987:16). Inputs whose hyperspace representation appear on one side of the plane are considered to be a member of the class or in-class and inputs on the other side of the plane are considered out-of-class. Additional layers in the perceptron model allow for more complex decision regions by partitioning off the space into intersections of several in-class regions. The network

topology is changed by adding one or two hidden layers of nodes between the input and output layer. Minsky and Pappert showed that a single layer perceptron could not be used to solve the exclusive-or problem (Minsky,1961). Huang and Lippmann showed that two and three layer nets can form arbitrary decision regions as well as solve disjoint decision region problems (Haung,1987:1-2).

A problem may arise when the input data is not sufficiently separable. Rosenblatt demonstrated (Lippmann, 1987:14) that if the data was separable, and a boundary could be placed between the two decision regions, the training algorithm would converge to the correct solution. However, if the data is not sufficiently separable, then the convergence procedure might oscillate, moving the decision boundary between overlaying data points. A modification to the training algorithm, using a least-means squares solution was suggested by Widrow and Hoff (Lippmann, 1987:14). Unfortunately, the algorithm is not as efficient for distinctly separable data. This type of problem is caused by ambiguous decision regions. When two classes of data are very close to each other in decision space, perceptron models do not converge well as will be shown in Chapter four.

Another solution, recommended by Huang, suggests using a hybrid combination of a Kohonen layer on the input whose output nodes would feed the inputs of a MFB network (Haung, 1987:1-10). In his approach, the Kohonen map could be used to organize data which would feed a MFB model input layer. This approach could prevent having to train the MFB to classify data in complex decision spaces and significantly reduce training convergence time.

To explore the possibility of combining different types of neural networks, the sonar classification work of Sejnowski should be considered (Sejnowski,1987:75-89). His work is important because it is one of the first efforts, along with Huang, to understand exactly what is happening in the hidden weights and nodes of the multilayer MFB. In his experiment, Sejnowski trained a neural network to differentiate between the sonar return of an underwater metal cylinder and a rock with a similar shape.

The network uses two layers. The classification was based on the low order Fourier components of the return signal. His goal was to determine the internal strategy used by the network to make a classification. A complete description of the result is beyond the scope of this discussion, refer to the Sejnowski paper (Sejnowski,1987) for a complete analysis. He did note that the overall strategy of the net was to default to a cylinder response and to detect the presence of rock characteristics. The scheme caused deactivation of all nodes, while the activation of only a single node caused the firing of the rock node. His work is significant, not only for the results of the experiment, but the tools and procedures he developed to characterize the inner workings of the hidden layers.

Huang, with a simple constructive proof, showed for several cases that disjoint decision regions could be formed, not only with a two hidden layer arrangement, but also with a single hidden layer. He went on to show that although these types of complex decision regions could be constructed on paper, in practice back propagation training would not converge in any reasonable amount of time without the second hidden layer.

With this analysis, Huang proposed three alternate classifiers, a fixed weight classifier, a hypercube classifier and a feature map classifier. Fixed weight classifiers attempt to reduce training time by only adapting weights between the upper layers of the network. By fixing the weights at some arbitrary values, hyperplanes formed at the input may be sufficient for the upper layer to classify without additional training of the lower levels. The weights were fixed and tested in two manners, first randomly set weights between negative 0.5 and positive 0.5, and second, using statistics of the input data to form grid lines. This solution proved inadequate, as the possibility of convergence depended on the starting values for the randomly selected weights.

The second proposed solution was to fix both of the lower level weights at specific hypercubes calculated from the statistics of the input data set. The upper layer of weights are trained using back propagation or some similar degeneration of

back propagation based on the relaxed requirements. The results were good, but the computational effort of estimating the initial hypercubes was significant. Better results were achieved using the feature map approach.

The feature map classifier is a simplification of the combination neural network. The first layer of the network forms a feature map using a self-organizing clustering algorithm as described by Kohonen (Kohonen,1987). Weights to the Kohonen map are trained unsupervised allowing first layer feature nodes to sample the input space with a node density proportional to the combined probability density of all classes. The first layer feature map nodes perform a function similar to that of second layer hypercube nodes from the previous example, except the results are more general. Each node will go high for a general region of the feature map. The upper layer will do the necessary "and-ing" and "or-ing" of the decision space for proper classification.

Huang used quantized speech vectors for his data set. For image target classification, the decision regions may be complex enough to require the additional layers of a MFB. Huang uses only a single layer above the feature map.

The next section discusses an alternate approach to classification called the counterpropagation network. Counterpropagation is used as the starting point and benchmark for development of the hybrid network.

2.12 Counterpropagation

The counterpropagation network is an architecture which combines the self-organization feature of Kohonen with the outstar structure suggested by Grossberg.

The combination network is a multilayer feedforward network with a Kohonen organizer on the first layer and a Grossberg outstar on the second. It has two additional layers used to train the network which will not be discussed, except to note that a counterpropagation net will allow regeneration of a input vector by specifying an output, hence the name counterpropagation.

Donald Woods (Woods,1987:473), noting two aberrations to the learning, suggested that a conscience be to added to counterpropagation. The first problem with the Hecht-Nielsen algorithm is that the input vector must be normalized. limiting all exemplars to the unit hypersphere. Consequently, no differentiation can be made between vectors which are related by a scaling factor. The problem can be overcome somewhat by a training rule which tries to minimize the distance measurement instead of maximizing correlations. In this way both (1,1), and (2,2) will be mapped uniquely in the decision space.

The result obtained by using Kohonen type mapping as an input another type of network is the generation of an intermediate, unclassified representation of the input. The complexity of the data to the output layer is greatly reduced and the construction of the output net is better understood. For backpropagation using the three to one first hidden layer nodes to input nodes, will probably converge, and network pruning can reduce complexity even more.

The difficulty of this approach is that the intermediate representation of the data may not be a good vector quantization of the input. Convergence is guaranteed under the simple criterion that the number of elements in the Kohonen layer is greater than the number of distinct decision regions in the input data, again something unknown a priori. The number is bounded by the number of exemplar vectors used for training. As the number of Kohonen units approaches that number, the solution will degenerate to a simple table look-up without generalization.

On the other hand, as the number of nodes is reduced, each node represents the average of a number of exemplars. Classification is simply a matter of mapping the Kohonen map to an output map. For something as simple as the counterpropagation rule, the Grossberg outstar format is sufficient. The output is simply a binary word one bit per class.

2.13 Summary

Computation based on neurophysiological models has been considered for a few thousand years, though actual neural computers have only existed for only a few decades. While the von Neumann computer remains the most widely used computer architecture today, neural computers may someday perform tasks which today's computer cannot.

Neural computers do not find solutions to problems based on systematic, exact calculations, but make judgments or estimates using a massively parallel approach. The approach uses from several to thousands of individual processing elements.

The processing elements, or nodes, are based partially on how the brain is thought to function. While the nodes may in some ways resemble the architecture of a neuron, how this architecture with interconnections between hundreds of nodes is able to solve real problems, remains a mystery.

Many different interconnection schemes have been suggested to transform a number of nodes into a computing machine capable of making judgments. Each of these schemes or topologies reveal their own strengths or weaknesses depending on the type of problem being considered.

The statistical makeup of input data seems to determines which type of neural network is best suited to solve a particular problem. For simple decision region problems with no disconnected or disjoint learning regions, neural network training is easy and only a few nodes are required. As the number of disjoint or ambiguous regions increase, the size of the network required for convergence seems to increase dramatically.

Classification problems would be simpler if the input data could be mapped into a decision space which lacks these disjoint regions. Data self-organization as suggested by Kohonen, may provide such a mapping. Combining Kohonen self-

organization with back propagation of error may reduce the complexity of the decision regions and consequently reduce training times.

In order to construct a combination MFB and self-organizing neural net, an environment to analyze the dynamic nature of neural nets is developed for use on a color graphics workstation. The environment will allow the study of several types of neural nets with application to both measured data (the Ruck and Roggerman sets) and calculated data. The next section will discuss the organization of the software environment use to develop the Hybrid network.

III. The Neural Network Analysis Environment

In order to study the dynamic nature of neural networks, an environment was devised to present the vast amount of data in such a way that would allow insight into the internal workings of the net. A system was designed for the Silicon Graphics IRIS workstation and written in the "C" programming language. The IRIS was selected over similar work stations because of the larger color table available than on the Sun or GPX workstations.

The design method was iterative. A number of working prototypes were developed, adjusting displays and menus as needed. As more was learned about each topology and algorithm, new requirements were added. Consequently, the requirements analysis phase lasted through most of the project. One of the goals of the project was to study different network topologies. Only as the models were implemented and tested did it become apparent what type of information needed to be displayed. After a number of prototypes were developed, patterns began to emerge as to which software modules were problem specific and which could be reused to create more varied and powerful modules.

Special features and displays were added to the environment as research into the problems continued. The simple models showed what might be useful. The final product took the basic designs and fit the pieces into general purpose reusable components for construction of new topologies.

The *NeuralGraphics* software package developed as part of the research, consists of independent programs run from a common menu. The programs are selected from a shell or script program which calls the individual programs. Each program is independent and is run as an execution file from the script program.

Although each program is independent, they all share a common software design. This section will discuss the modules which make up each of the four types of neural networks which make up the NeuralGraphics package.

3.1 Software Design

Each of the neural networks is composed of a number of independent software modules. Changing the topology required only small local changes within a particular module to change the old network to include new training rules or display features. In addition to the basic modules, a graphics tool box is provided, which is specific to the Silicon Graphics IRIS video display.

At the highest level of abstraction, a neural network will consist of: a routine to compose input vectors, a propagation algorithm to feed the vector through the network, a training routine for modification of weights, a graphic display package and an analysis routine for periodic testing. Important lower level modules include the initialization procedure and an event driven menu to control the training and operation of the network.

3.2 Initialization Module

The initialization routine has three functions. The most important is the establishment of the network in memory. In addition, the weights and thresholds must be filled either by a random number generator, or a stored file from a previously trained net. The second feature is to check which software switches are being invoked. These include display options, and training parameters. The third function is the equipment check to determine the nature of the graphic displays and to initialize the video drivers as necessary. The size of the screen will determine exactly how much data can actually be displayed.

3.2.1 Initialization and Declaration of the Net A "C" data structure is used to define the network. Each type of net uses a similar structure. An array of numbers for the output and input form the basic net. Between layers lie similar arrays for the various hidden nodes, if any. In addition to the nodes, there are the weights connecting the nodes. These are represented as two dimensional matrices. Another element that makes up a neural net is called a threshold value. Although similar to a weight, these values are stored as a single dimension array associated with a particular node array.

These data structures, along with a definition for input, output and hidden layer array lengths are stored in a single program module (a header file for C programmers). The data structure is declared and is one of the few global variables for the whole program. The net weights are initialized in one of two manners: either by randomly selected weights, or retrieved from a file. After the data structures are initialize the video hardware must be initialized.

3.2.2 Initialization of the Video Display The current version of the package initializes the hardware for a Silicon Graphic IRIS workstation. This is a Motorola 68020 based computer with an extended graphics capability. Wherever possible, hardware specific commands are isolated to the graphics module. Porting the code to other types of hardware requires only a substitution of the gl.h and device.h header files. These device specific files are provided by the IRIS compiler package. A substitute file would include macro redefinitions for the hardware specific commands. Once initialized, a framework is established for solving a classification problem. The main program loop continues the process by propagating the input to the output, adjusting the weights, and checking the progress. The next section discusses the structure of that process.

3.3 The Main Program Loop

The main program loop consists of the MAKEINPUT module, the PROPAGATE module, the TRAINNET module, the TESTNET and the DISPLAY and SHOW modules. A counter is used to inhibit the calling of some modules through the loop to reduce computations effort. For example the screen may be updated only on every tenth cycle. However, each pass through the loop represents one training cycle, so MAKEINPUT, PROPAGATE, and TRAINNET will always be called.

3.3.1 Makeinput Essential to the training of a neural net is a set of input patterns and a defined classification, in other words, an input vector and a desired output (doft). The function will fill the net data structure with a randomly selected exemplar from the exemplar or test set. The module allows for calculating the vectors or taking them from a stored data set read in during initialization. Also, the initialization data preprocessor may add gaussian noise to vectors. The work of Sietsma and Dow indicates that adding noise distributed across the actual statistics of the data seems to improve the performance of the network.(Sietsma,1988:325).

3.3.2 File Input of Exemplar Sets Most problems can be described in terms of a set of input vectors and a prescribed classification. The *NeuralGraphics* software requests a file name on initialization which is used to fill a pool of exemplars. An exemplar is selected randomly from the pool whenever the makeinput routine is called. Also, the routine ensures that in addition to random selection on exemplar number, there is also random selection based on class type. This prevents excessive training on a single class, when the classes are not evenly distributed in the input file.

3.3.3 Measured Data With measured data, the data is contained in a data file. These files can contain any type of information. In general these input files contain a data set which exercise a particular area of investigation or data that has

been measured by a real system. The Ruck data is a collection of Zernike moments (Born and Wolf,1964) calculated from laser radar images of tanks and trucks. The file could just as easily have contained data from frequency analysis of speech or spatial frequencies of character data. A neural net should treat any type data in a similar manner. The file also must contain header information relating to the length of the input and output vectors and number of exemplars and classes. The data is composed of an exemplar number, the vector quantizations and finally the class identification.

3.3.4 General Purpose Functions The MAKEINPUT package contains other modules necessary for handling the data files. The three principle functions contained in the package allow for initializing the data storage, returning a random exemplar to the main routine, and statistical processing of the data. The statistical function is a stand alone program. By calling this program from the menu, the analysis is completed and the program terminates. The statistical package performs a K nearest neighbor analysis of the data and displays the result.

3.3.5 The Fourier filter problem A special subroutine for the backpropagtion network demonstrates mathematically calculated input vectors. This routine is contained in the Makeinput package. It is not intended to be part of the software package, but serves as a template for problems that require calculated exemplars rather than those taken from a file.

The MAKEINPUT routine for this problem computes an input vector based on the sum of three sampled sine waves with an amplitude of one. The length of the input vector is arbitrary, but once defined in the header information, represents the Nyquist sampling limit. The output vector has the same length as the input vector. The output vector represents the desired output in terms of harmonic components. For example: if the length of the two vectors is chosen to be eight, the input vector is formed by taking eight equally spaced samples of any of three possible sine waves,

$sin(0), sin(t), sin(2t) \ldots sin(nt)$. See figure 3.

There are n outputs, one for each possible frequency component. The output vector represents the presences or absence of a specific fourier component.

In addition, a random phase is added to each of the three components. The output is specified by the three selected sine waves. If a particular sine wave is selected, that output value is one, if not selected as one of the three, its output node value is zero. The hypothesis that a mapping from the input vector to a Fourier analysis at the output seems reasonable considering that the sine and cosine functions, together form a basis set of orthogonal vectors.

These types of problems can be used to test neural networks. One advantage of neural networks is an amazing fault tolerance. Experience has shown that this fault tolerance extends to the software as well. Consequently, many software errors only show up as a degradation in training efficiency. Before a new neural network can be used for measurements, it should be tested against one of these types of standard problems. After validation of the net against a standard problem, the net could be used to analyze measured data.

3.3.6 PROPAGATE - Using the net With the organization of the net in memory, the net can begin to learn and classify targets. To use the net, the rules for propagating the data from the input to the output must be specified. This is the purpose of the Propagate package.

Due to the experimental nature of the presentation environment, a design goal was to reduce the amount of software an experimentor would have to understand in order to alter the code to suit a new experiment. Both PROPAGATE and TRAINNET are the two modules which must be the shortest, cleanest and most precise from a software engineering point of view. Altering the screen displays could require a fair amount of expertise in graphics and program interfacing, but one would not expect to have to change them very often. The PROPAGATE and TRAINNET

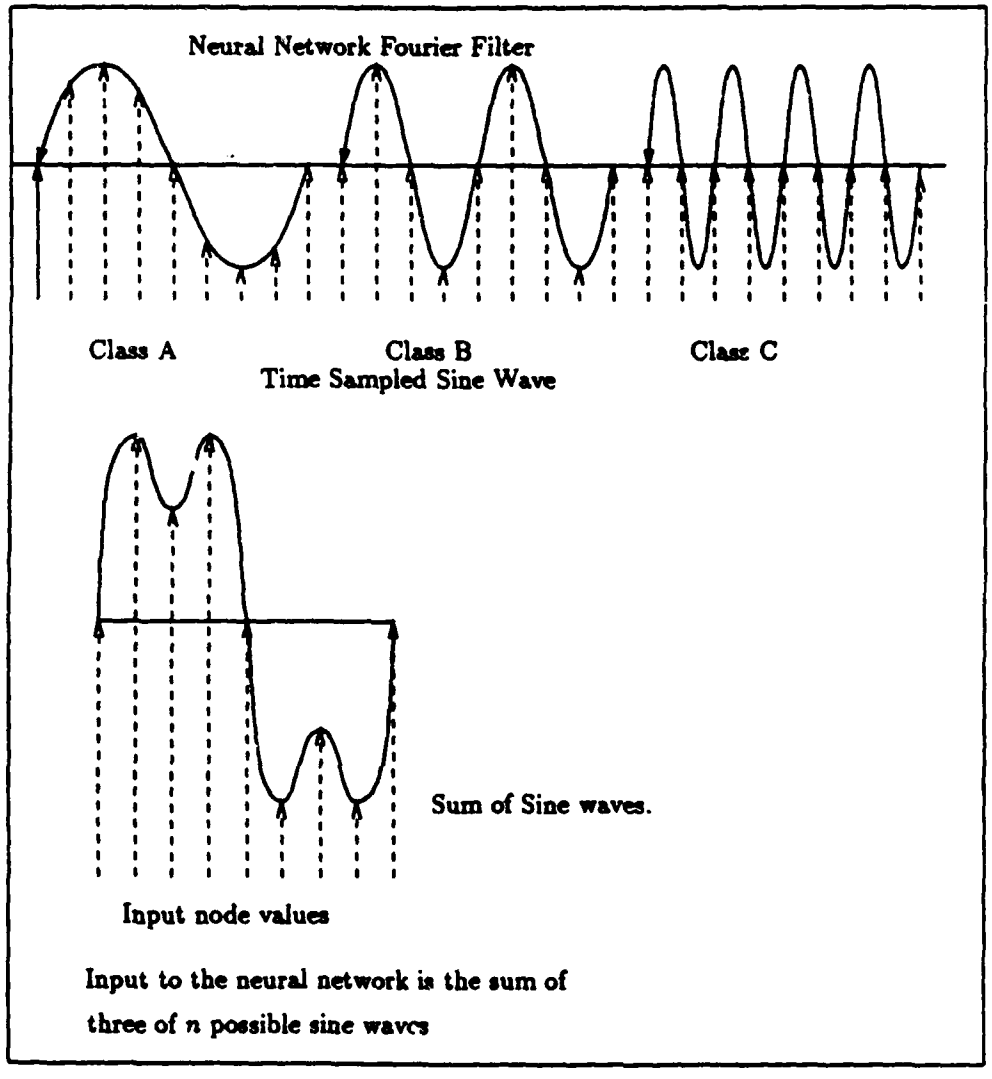


Figure 3. Neural Network Fourier Filter

Note: Class A,B and C represent three of n possible sine waves.

packages would be expected to change with each new experiment, so a reasonable effort is made to enhance the simplicity and documentation of the code.

3.3.7 TRAINNET - Teaching the net. The TRAINNET module specifies the training algorithm for the network. Four versions of the trainnet were written, one for each type of training rule. The package includes the backpropagation rule developed by Paul Werbos (Werbos, 1974), as well as Kohonen type rules for unsupervised training, counterpropagation by Hecht-Neilsen and hybrid Propagation developed as a result of this thesis effort.

3.3.8 CHECKERRORS - Checking performance of the net. Network performance is evaluated in two ways. First, training is periodically stopped and a test set is evaluated. The second method checks the performance after every training cycle against the current training vector. For a general evaluation of the neural net performance, a set of training data is run through the net without training cycles between tests. For a more specific analysis, using a data set different than the one used for training, can show the validity of the feature set used to classify the targets. The CHECKERRORS routine allows this type of checking mid-process by running a quick test set through the net, then measuring the performance. The actual test set is specified in the initialization routine. When the data is read into memory during initialization, the first line of the file specifies the number of training exemplars followed by the number of test exemplars. This partitioning of the data allows the CHECKERRORS routine to test the net with a set of vectors the net hasn't seen before.

3.4 Program control

The program uses two devices for program control: the mouse and the keyboard. The mouse is used where specification of a particular node is required and the keyboard is used for larger program control functions like reading or writing weights.

3.4.1 Event Driven Menu An event driven menu is provided to control housekeeping functions of the network. Event driven routines are more efficient than using a keyboard poll. Event driven menus require a hardware event to call the menu subroutine. No device polling is necessary. The event, in this case, is typing a control C on the keyboard. A control C activates a hardware interrupt to the kill address vector. The program has substituted the normal kill address vector with the menu address vector.

While the main purpose of the menu is to allow the user to save and restore weights, the menu also allows control parameters to be changed while training is in progress. The menu display in the textport offers a series of selections. Selection of a particular item will then prompt the user of the parameters associated with the particular call.

3.4.2 DISPLAYNET The display routines refer only to the graphics portions of the display. These include such functions as drawing the network weights, drawing the nodes, setting the colors, finding data ranges, and drawing color bars. The package is split into two types of graphic routines. The basic set contains those functions which are machine dependent. In general, a macro is used when possible to allow for redefinition to other machines. Those which are not machine dependent are combinations of the basic routines that are machine dependent. An example would be the color bar routine. These two types of routines are included in the graphics package. The group of routines which are specific to a particular topology are contained in the display package. In general, when the problem under consideration changes from something like a Kohonen map to a counterpropagation model. The entire display package is replaced. The graphics package is used as an include file so the process of building a new display is somewhat reduced.

3.4.3 SHOWNET The SHOWNET package includes all the textport printing and display routines. This package includes routines for display of a set of weights,

output and input values printed in ascii format in the text port. Except for test results, the subroutines are not intended for use in general operation of the network. These routines are intended for use in trouble shooting when the program has been altered. The exception is the showoutput function. Periodic presentation of both the desired and actual output is useful in tracking the progress of the net. In addition, the error information is displayed in the same module. Adding more information during the *SHOW* subroutine is a matter of changing the "l" codes in line 10 of the SHOW subroutine.

3.5 Summary

The material in this chapter outlined the major software modules which make up the NeuralGraphics environment. Although the package is self sufficient and can be applied to almost any segmentation or vector quantization problem without code modifications, the nature of software dictates that sometime changes will be required. The information presented, together with the code and comments, should be sufficient for making changes to suit a particular problem.

The next section chapter two purposes: to document validations and testing the system and more important, is to answer the types of questions the system was developed to investigate.

The NeuralGraphics study environment is designed to function as a window into the internal structures of neural networks. The following chapter will describe the observations which led to the development of a new type of neural network, the hybrid net.

IV. Data Analysis

4.1 Introduction

This chapter has two purposes: first, to document the validation of the software and second, to show how the graphics environment was used to construct a new paradigm for training artificial neural networks. The hybrid network is the result of examining learning characteristic for several types of neural networks, then amending the learning rules to correct several weakness discovered. The hybrid network is shown to be more efficient than MFB for ambiguous decision regions.

Several types of data are analyzed to validate the study environment. To investigate the learning characteristic of MFB, error surface analysis software was developed. Using a three-dimensional display, patterns of neuron weight adaptations are presented for simple neural network problems. A Fourier filter problem demonstrated the use of the environment with calculated exemplar sets. The last two problems, consider two sets of data extracted from digital imagery. The Ruck and Roggemann data sets are used to evaluate the features selected for neural network analysis. The analysis of these data sets is used to contrast numbers of training cycles and accuracy for Kohonen maps, the multilayer perceptron, counterpropagation, and hybrid propagation.

4.2 Error Surface Analysis

This section will discuss an experiment in error surface analysis used to examine the adaptation of neural network weights while training. Two methods of adapting weights are examined and compared.

A neural network node is the fundamental unit of an artificial neural network. It is composed of an input vector, a single output, and a set of interconnecting weights. A node also includes a rule for propagating the input to the output, and a

rule for training the weights. Three types of nodes can process analog input data. These three types of nodes are the back propagation nodes, the Kohonen nodes, and similar to the Kohonen nodes are the counterpropagation nodes.

To investigate the process of error reduction by adjusting the weight vector, each type of node will be considered in turn.

4.2.1 Error Reduction by Back Propagation of Errors Back propagation of errors was developed by Paul Werbos (Werbos,1974). In recent work, a modification to back propagation has been suggested by David Parker (Parker,1986) called second order learning. To understand how these algorithms reduce error, a visualization tool is included with the package to track the error as the weights adapt toward a solution.

4.2.2 A Simple Classification Problem The first artificial neural network system considered is the simplest system possible. The simplest problem associated with a neural network would be that of a single neuron with two weights attached. The classification problem is to differentiate between two points in the decision space. Use of only two weights will allow plotting of the system error as a function of two values, W_1 and W_2. A value of one is used for θ, the threshold.

Two points are arbitrarily selected: (2,1) for class one and (1,1) for class zero. Now, the amount of error for any two arbitrary points can be determined by the relation:

$$error^2 = (1 - f(x_1 * w_1 + y_1 * w_2))^2 + (0 - f(x_2 * w_1 + y_2 * w_2))^2 \qquad (1)$$

The function $f(w_1, w_2)$ is a limiting function as described by Lippmann (Lippmann,1987). This demonstration uses a sigmoid function to ensure differentiability as required for the back propagation rule. This function is sometime called a squashing function, because the output is squashed between zero and one. The program is run a number of times with the results shown in figure 5.

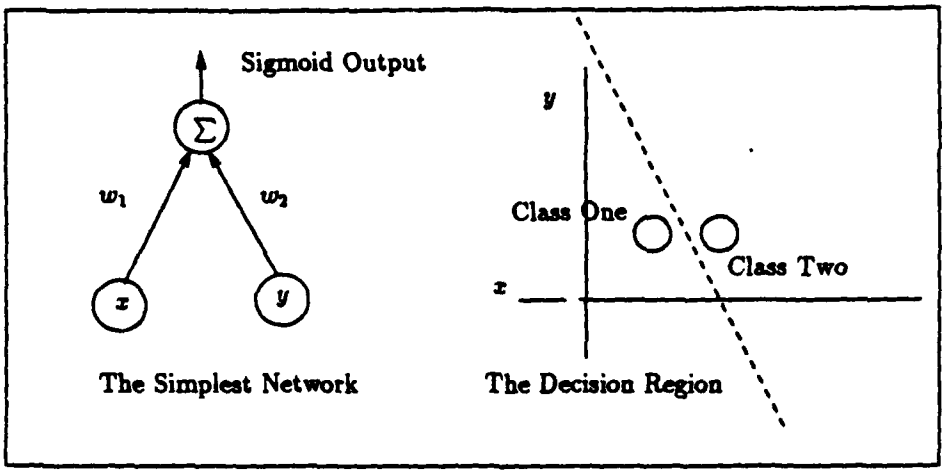


Figure 4. Simple Neuron Problem

In Figure 5 the red and yellow tracks show the change in weights from the random starting point to the final convergence point. Red areas on the plot indicates those areas with the greatest amount of error and blue represents the lowest amount of error. Weights are plotted with w_1 on the x-axis and w_2 and the y-axis. The three dimensional plot shows height of the error surface on the z-axis. The graph in the upper right hand corner exhibits the line found by the neural network which divides the two decision regions.

Two point problems are of little interest because of the simplicity of the error surface. The problem is made arbitrarily complex by adding additional points. The example in Figure 8 considers six points for classification. The equation of the error surface can be generalized to include an arbitrary number of points.

$$error(w_1, w_2)^2 = \sum_{i=1}^{n} (doft_i - f(x_i * w_1 + y_i * w_2))^2 \qquad (2)$$

Considering weight values between positive and negative two, an error surface is constructed to show the relative error for different values of w_1 and $w2$. The results are shown in Figure 8.

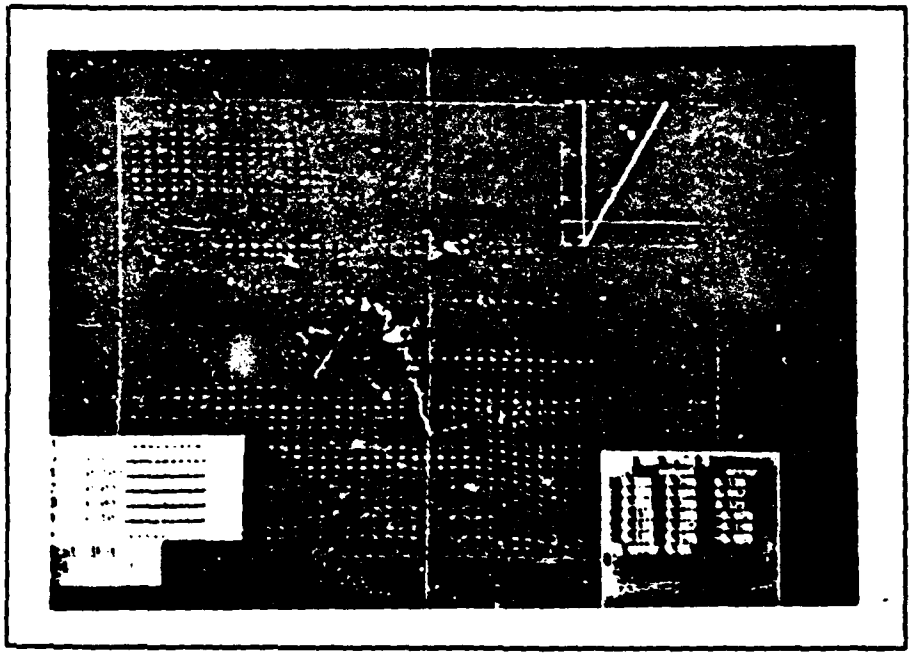


Figure 5. Six Point Error Surface: Low Eta

Note: No matter where the weighs start, they move toward the lowest point on the error surface. Note the small step size in areas of low error.
Eta = 0.3 Momentum = 0.0

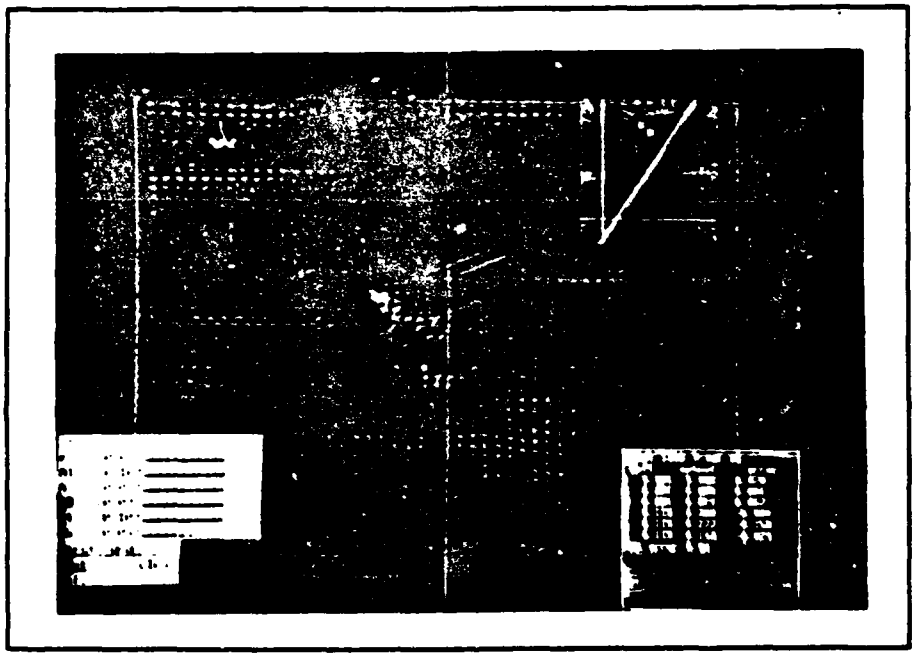


Figure 6. Six Point Error Surface: Large Eta

Note: For larger eta the step size seems larger.
Eta = 0.8 Momentum = 0.0

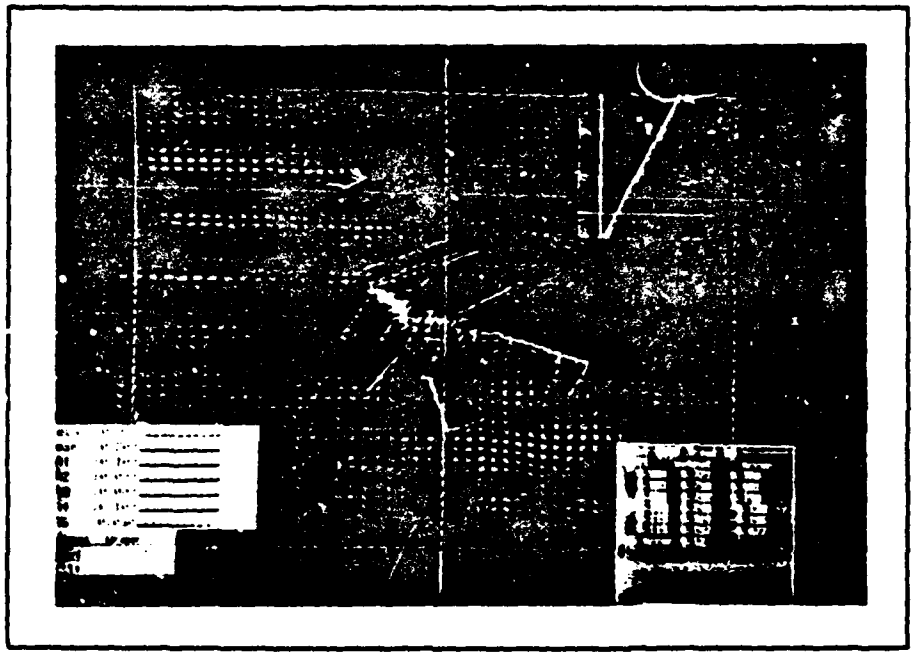


Figure 7. Six Point Error Surface: Low Momentum

Note: Momentum seems to round the edges of the path.
Eta = 0.3 Momentum = 0.2

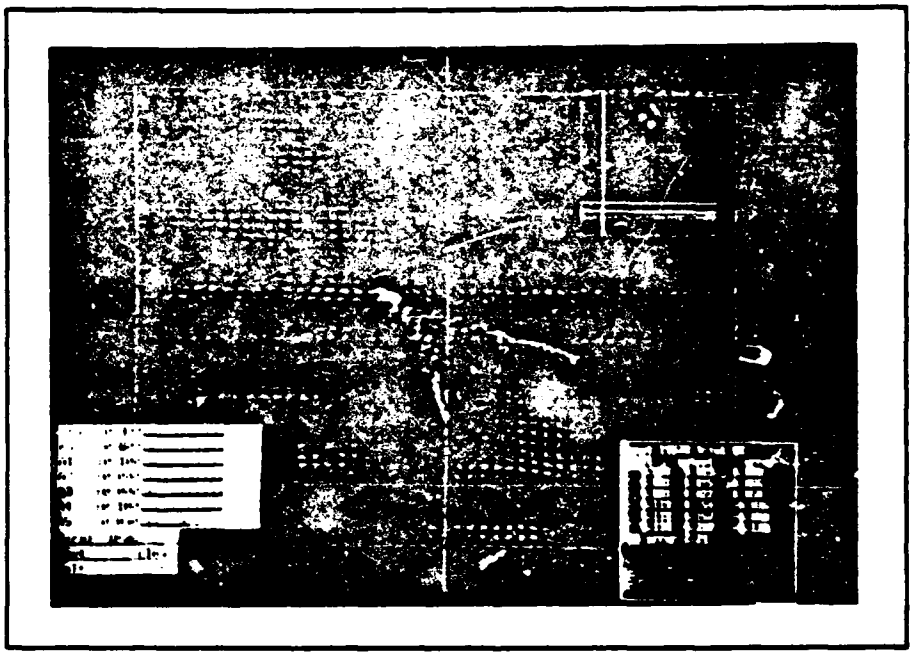


Figure 8. Six Point Error Surface: Large Momentum

Note: The values offered the best compromise between good movement in the low error slope regions and yet lack the wild swings of the high slope regions.
Eta = 0.3 Momentum = 0.8

The experiment was repeated several times using different clusters of the exemplar points and randomly selected initial weight setting. Each initial weight setting relates to a new starting point on the error surface. The yellow tracks in Figures 8 and 9 show the weight values adjusting from higher values (red and green) into the blue regions that indicate a the lower error. Figure 8 shows the reduction of error for a Werbos back propagation rule, while Figure 9 shows the decent down the error surface of a second order algorithm.

The error surface routine demonstrates how the weights adapt from the random starting point, to the region with the lowest error. While the second order algorithm

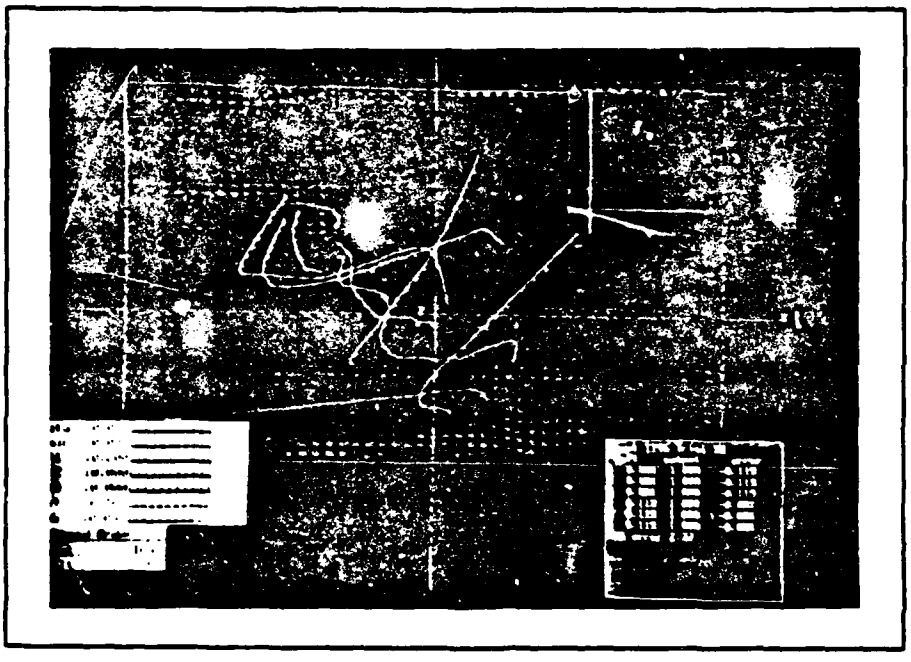


Figure 9. Second Order Algorithm

Note: The error reduction is smoother, but sometimes misses altogether as noted where the line goes off the screen.
A1 = 0.2 A2 = 0.0 A3 = 0.0 A4 = 0.1 A5 = 0.05

showed a smoother flow across the error surface, the first order seems to find the error minima with fewer training algorithms. Second order adaptation requires adjustment for four coefficients. (Piazza, 1988). Examination of the error surface ensures that use of first order algorithm is sufficient for the following problems. Of the five coefficients necessary for second order back propagation, two are also required for the first order algorithm. One is the standard η, or training rate, and another is a term which Lippman calls momentum (Lippman,1987).

Momentum sometimes has the effect of decreasing training iterations, while at the same time may avoid local minima.

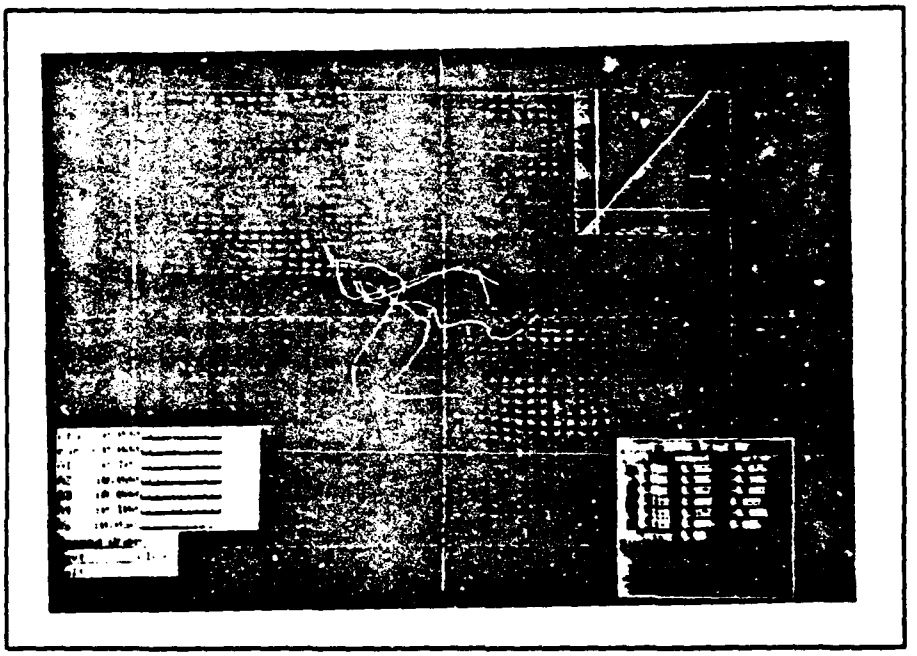


Figure 10. Second Order Algorithm

Note: Reducing the value of A1 to 0.1 improved the performance.
A1 = 0.1 A2 = 0.0 A3 = 0.0 A4 = 0.1 A5 = 0.05

4.2.3 Self-Organizing Nodes No graphic tools were developed to analyze the Kohonen and counterpropagation node. Their learning paradigms is fairly straight forward, with the weight vectors moving toward the input vectors. Neither of these types of nodes is capable of classification on its own, each of these nodes performs a simple correlation with the input vector.

In the case of the Kohonen node, the correlation is a measure of the Euclidian distance between the weight vector and input vector. The counterpropagation node uses a dot product multiplication to correlate between the two. In either case, the weight update rules cause the weight vector to move toward the input vector.

The output of a Kohonen node is usually based on the values of adjacent

nodes. For example, only the node with the lowest distance or highest correlation of a layer of nodes is passed to the output or next layer. The rest are sometimes set to zero. The learning process in self-organizing nodes specifies that only one or several adjacent nodes are adjusted with each training iteration. The effect is to allow one node to *capture* a particular region in the decision space. The weight vector for a particular node will be equal to the average of all the input patterns for which it has been updated.

The counterpropagation node is in most ways equivalent to the Kohonen node. The difference is related to the type of data which is being classified. The counterpropagation node expects normalized data, otherwise the dot product will be scaled and the correlation of the weight vector with the input vector would not be valid. All input vectors must lie on the unit hypersphere. When the data is derived from transducers or instruments, the measurements for similar patterns my be uniformly scaled. Normalization removes the effects of this scaling.

4.2.4 Results First order back propagation of errors seems sufficient for any simple classification problem. Using a value of 0.3 for η and a value of 0.8 for momentum seem sufficient for training. Higher values didn't seem to make any difference in training times.

The smooth rounded tracks of the second order algorithm indicate excellent error correction. However, due to the computational complexity of computing each correction, the first order algorithm was selected over the second order algorithm. Piazza's work with larger network showed that although the second order network does out perform first order networks, the improvement is small and may not justify the additional computations (*Piazza,1988*).

With selection of a training algorithm, the next section discusses an application of the training routine to another simple problem, identification of cosines in a random signal.

4.3 Feedforward Networks as Fourier Filters

This section will describe an experiment in which a back propagation network is used to identify the presence of specific components in a harmonic signal. The experiment was devised as a simple prototype on which to build the graphic displays. Although the experiment was successful, the purpose of the program was to identify which elements of the network should be displayed in graphics format.

The equation describing the calculation of the digital Fourier transform looks like it could be represented as a description of weights in a neural network.

$$X_d(k) = 1/N \sum_{n=0}^{N-1} x_n * cos2\pi kn/N - jx_n sin2\pi kn/N \tag{3}$$

The output of a specific node is given by $X_d(k)$ the inputs are the x_n values. With k output nodes and n input nodes a system could be implemented to generate Fourier transforms by setting the weights using the equation:

$$Weight_{kn} = cos2\pi kn/N + jsin2\pi kn/N. \tag{4}$$

To adapt Fourier analysis to a neural network, the problem is modified to take advantage of the strengths of neural networks.

To reduce the problem to a manageable size, only the in-phase (cosine) components will be considered, and not the quadrature components(sine). Further, only the actual presence of a component will be detected and not the magnitude related to that component. Although it appears conceivable to calculate coefficients, it would not be a good pattern recognition problem. A problem consistent with the intent of pattern recognition would simply indicate the presence of a Fourier component in sufficient strength to fire an output node.

A similar problem, calculating quick approximations of periodic data is discussed in detail in the thesis work of James Straight. His work pays particular attention to time projections based on the past history of the input data. (Straight,1988)

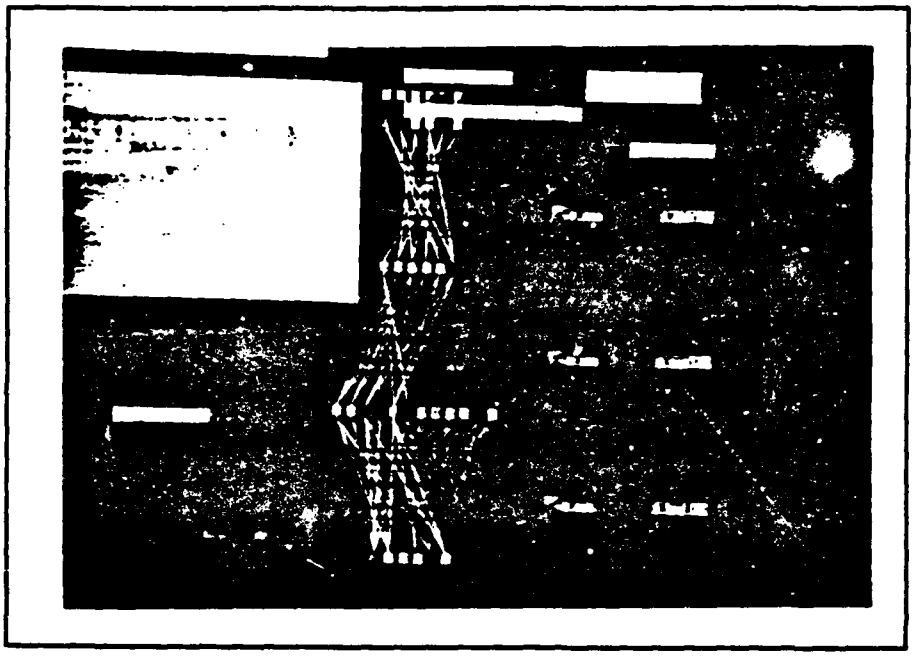


Figure 11. Fourier Filter Neural Network

Note: The position of the three red output nodes indicate the input pattern contained the sum of $sin2t, sin3t, and sin5t$ with t between zero and 2π.

The back propagation algorithm requires the output to be in the form of a differentiable squashing function. The most commonly used is the sigmoid (Lippmann,1987). To generate magnitudes at the output would require using a mapping function to recover the desired output from a sigmoid function. This complication will be avoided by using only unit valued Fourier components in the input, and training to only desired outputs of one or zero. To train the network to generate actual components of the Fourier transform is beyond the scope of this exercise.

4.3.1 Making the Input and Output Vectors The network consists of an arbitrary number of input nodes with an equal number of output nodes. Several variations in the size of two hidden layers were tried. The elements of the input vec-

tor are formed by taking time samples of the sum of three sine wave. For example: if the network topology consisted of n nodes, three values of n were selected randomly. The three sine wave are added together. The sine wave were selected from: sin t, sin 2t, sin 3t ... sin nt. Time samples of the composite waves are used as the input vectors.

Before time sampling an arbitrary phase is added to each of the samples. Arbitrary phase is added to each signal to ensure the internal representation of the input vector is related to the Fourier decomposition. An early experiment neglected adding a phase component. Consequently, the training effort was greatly reduced, which indicates that a simpler internal representation was detected by the network.

The resulting signal (input vector) is fed as input into the neural network. The output is calculated and compared to the desired output. In this limited problem three of the output nodes should contain one, and N-3 will contain zero. Evaluation of the system consisted of considering one of the fundamental problems of neural networks, finding the optimum number of nodes.

It was noted during analysis that the greater the number of connections, the more complex of a problem it could solve. For example, with no hidden layers, the neural network would converge as long as the phase component remained zero for all components. By adding two hidden layers, random phase could be added to the signals.

4.3.2 Results A number of measurements were made with the results shown in Table 1.

The experiment demonstrated that neural networks can learn Fourier analysis, although no evidence exists to show that the internal representation discovered by the network has anything to do with Fourier analysis. The data set for these initial classification experiments, was statistically homogenous because each of the n possible Fourier components was selected with equal probability. Also, the amount of

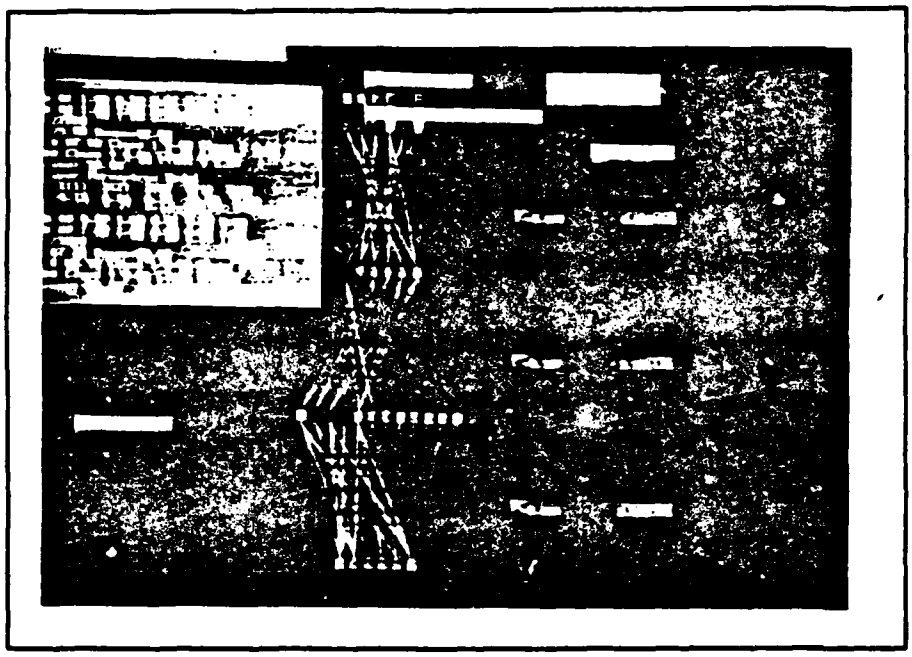


Figure 12. Neural Network Fourier Filter

Note: This topology was used to represent a six input Fourier filter. The input is the time sample sum of three sine waves with random phase. The three red dots at the output show which components are being detected.

phase added to each sine wave component was selected randomly using a constant probability. Consequently, an infinite number of possible combinations of random phase and components was possible forming an infinite training set.

Real problems usually lack this advantage. Measured data tend to have data point clustered together, with indistinct class separation boundaries. The next section introduces some of the problems inherent to measured data, and an approach to solving these types of problems. This approach, called self-organization, attempts to map the input data into a more logical decision space reducing the complexity of the classification problem.

Input	First	Second	Outputs	Training Cycles	Correct
4	4	4	4	20,000	.22
4	8	8	4	20,000	.82
4	16	16	4	20,000	.95
8	16	16	8	20,000	.97
10	20	20	10	10,000	.95
16	32	32	16	10,000	.95

Table 1. Fourier Filter Nodes vs Training Time

4.4 Self-Organization of Data

Lippmann and others have shown that the multilayer perceptron approach to problem solving is to the divide the decision region with hyperplanes in the first layer then appropriately combine these regions with *and-ing* and *or-ing* functions to generate class distinctions in the upper layers. Self-organization has been used for some time in image processing data reduction (Kuczdwski, 1987). Kuczdwski suggests using MFB as a self-organization structure. His algorithm uses a five layer feedforward network which attempts to find a smaller internal representation of the input by using the input vector as the desired output then trains on itself.

If there are fewer nodes in the hidden layers than at the output, the internal nodes are forced to learn a set of lower dimensional features than were given at the input. The reduction should be a more efficient pattern classification where the reduced features can be considered as some sort of basis function set. If the net converges, these features preserve the distance metrics and can be used to reconstruct the original pattern.

When the network is presented a pattern which contains more information than the storage capability of the net, the best average match is selected internally and used to generate the output. Kuczewski reports that the reduction in dimensionality has no consequences until noise is added to the input vectors in which case misclassifications go up significantly.

For application to the classification problem, the bottom two layers may be extracted as self-organizing input layers, then used as an input to another network.

To test this means of self-organization, a five-layer network was constructed and tested using a double disjoint region test set. After several hundred thousand training iterations, the network had still not converged to a solution. The network had difficulty finding a solution because of the weight update algorithm used for lower levels. As more layers are added the backward error propagation becomes more diluted with each level removed from the true error measured at the output. As more and more hidden layers are added, training time increases significantly. When the reason for the failure was understood, this line of investigation was dropped.

Perhaps a network which trained from both ends toward the center could overcome the dilution of the error. Such a network could use self-organizing based training at the input and backward error propagation at the output.

Although the application of MFB as a self-organization tool has been demonstrated by Kuczdwski, application implies greatly increased training times which is an unacceptable liability. Use of Kohonen maps to set the weights for an MFB network may offer a better solution.

4.4.1 Kohonen Self-organization Lippmann suggested that back propagation of errors works by using the input layer to construct hyperplanes to separate classes. The upper levels appropriately combine regions formed by the hyperplanes to allow classification. To increase the efficiency of each input node, it may be possible to construct these hyperplanes based on the statistics of the data. To explore this hypothesis, this section examines Kohonen self-organization.

4.4.2 Kohonen Self-Organization and Simulated Data Using an early outline of Kohonen's self-organizing algorithm (Kohonen, 1987), a display was designed to evaluate different distributions of iuput data. First a two dimensional input

is constructed, then a three dimensional, then finally an n-dimensional input. To investigate this self organization of data with respect to the Ruck and Roggermann data sets, a Kohonen map is constructed which accepts file data as inputs.

An important aspect of biological neural function is the ability to organize neurons in response to stimulus. Kandel and Schwartz believe that the placement of neurons is orderly and reflects characteristics of the data being sensed (Kandel,1983). Kohonen maps attempt to dupiicate this ordering of data without respect to any particular classification of data with the self-organizing feature map.

The environment includes a Kohonen map demonstration designed for a high resolution graphic display. The first display demonstrates self organization of randomly selected two dimensional data. The presentation allows for exhibition of the weight vectors, a decision region map, and a display of the actual data point being presented to the neural net.

The user may select any of a number of statistically distribution patterns for training the network including gaussian, random, chi-squared. In addition, the data may be distributed over limited regions such as, squares, triangles and crosses. The net was trained a number of times to verify that the nodes is distributed over statistical optimum regions as predicted by Kohonen (Kohonen, 1986).

In Figure 13, a square bounds a region of gaussian distributed data. The input distribution box shows a plot of the points as they are presented to the net for training. The yellow Kohonen decision region map shows how the Kohonen output nodes are distributed across the input data. Each intersection of lines represents a Kohonen node. The position of the node is set according to the corresponding position of the weight vector in space. Since only two weights are connected to each node, the node can be regarded as a position in the decision region. Notice how the nodes reflect a higher concentration in the central region where there has been more data presented. To facilitate tracking data, nodes in one corner are colored white, while the remaining nodes are colored magenta (purple to everyone

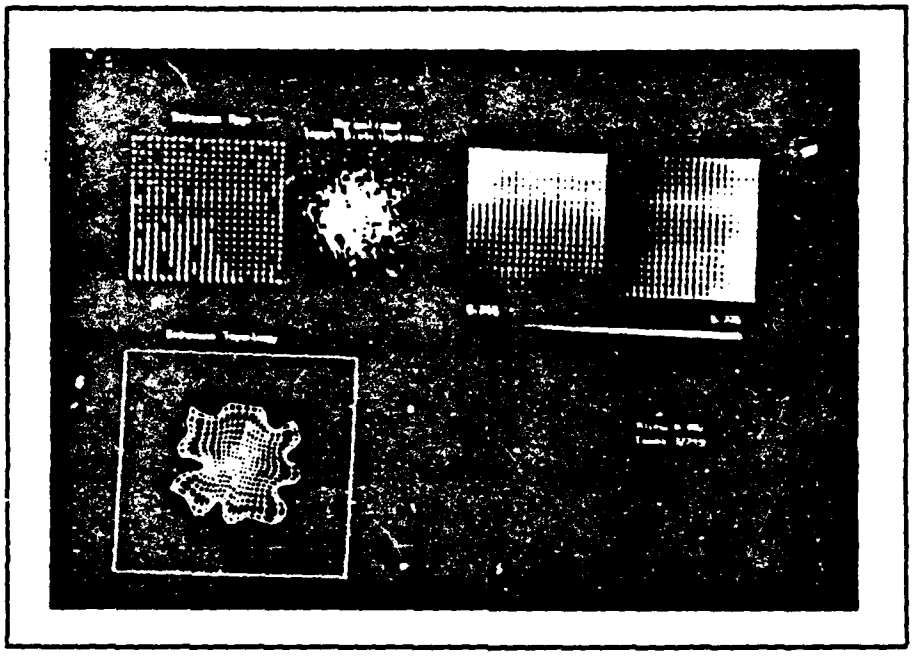


Figure 13. Gaussian Distribution over a Square
Note: Grid line are closer together in the center where more data has been presented.

but computer graphics specialists). The color code represents an arbitrary partition into two classes. In every distribution, the white samples will represent 25 percent of the input data.

As you can see from Figure 13, even though the white dots occupy only a small region of the input vectors plot, on convergence, they occupy one quarter of the Kohonen map nodes.

This is always the case (under ideal conditions). The number of nodes per class is proportional to the distribution of the classes within the entire data sample.

Looking at the Kohonen decision region map, the yellow grid lines are closer near the center than on the fringes. This represents how the decision region allocates

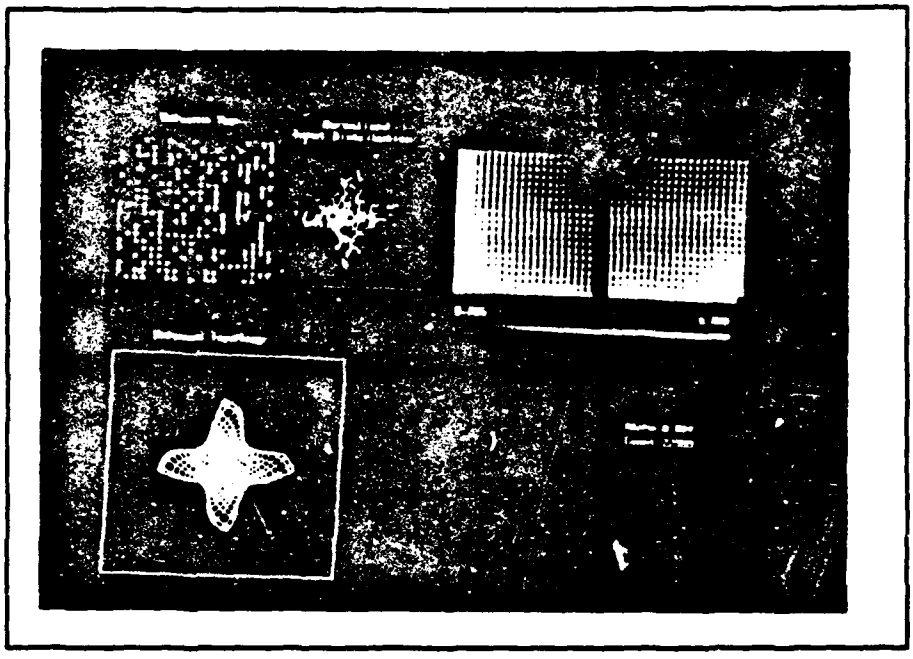


Figure 14. Gaussian Distribution over a Cross

Note: Number of white nodes is about 25 percent just as white input data is about 25 percent.

more hyperplanes to the regions where there is a higher concentration of data. Each intersection of a grid line represents a Kohonen node. This self organization of data used as an input layer to another type of network may improve the performance of a network with synergism between the two layers.

4.4.3 Kohonen Self-Organization and Measured Data When the two dimension problem is extended to three dimensions the results began to break down. If a Kohonen preprocess of the input data is expected to be useful, then each node must become a clear winner for a particular class. With three inputs to the Kohonen network the results were mixed.

Although, clustering of the nodes to a distinct class was exhibited, a number of nodes were isolated away from the central group. Further, dynamic analysis of the net revealed that under training, nodes were changing from one class to another.

When the Ruck data was used to train the net, the effects were more severe: isolated nodes, disjoint groupings, and inconsistent winners. Obviously, self-organization, though proven with ideal data sets, will require some modification to the training algorithm for use with less than perfect data. These deficiencies are also reported by Robert Hecht-Neilsen (Hecht-Neilsen,1987) and Dan DeSieno (DeSieno,1987) and others,who propose several improvement of the training method. The modifications were included in later models using Kohonen self-organization as part of the network. However, these improvements were not added to this particular demonstration.

A later section will discuss implementation of these improvements. The next section will return to MFB. While, self organization is proposed as a method to optimize neural network topologies and training complexity, self-organization lack a fundamental capability necessary for pattern recognition.

The next section evaluates MFB with respect to measured data, and the following section can discuss combination networks.

4.5 Back Propagation of Error

Kohonen maps cannot perform classifications, only create a self-organized clustering of the input data. Additional effort is required to relate these clusters to a particular classification. In the next section back propagation, used with the Ruck and Roggermann data, is consider for the classification mechanism. Back propagation will be used to bridge the gap between organized data and classified data. This section will discuss two critical issues related to neural networks: numbers of interconnections, and training difficulty.

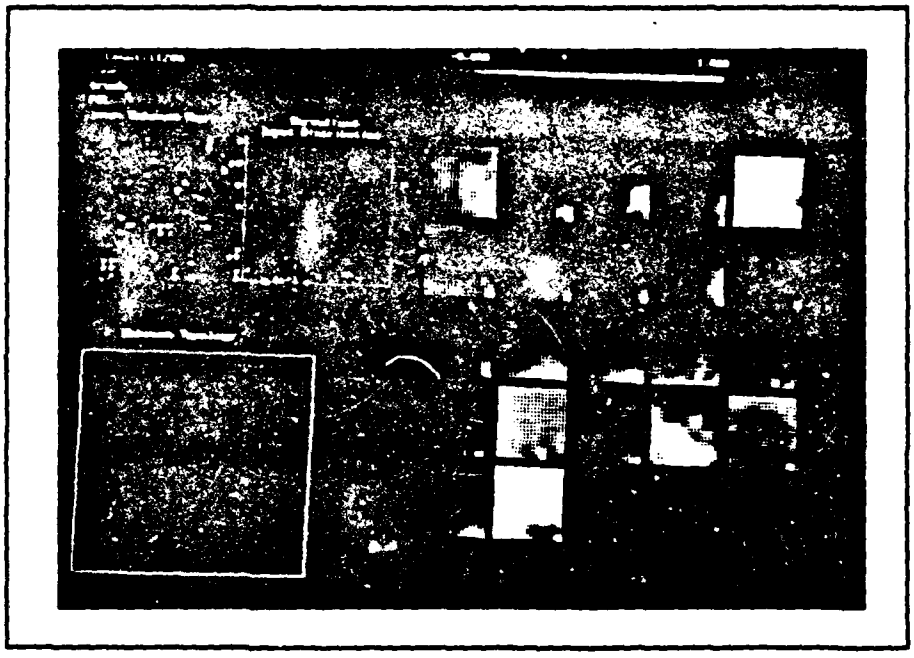


Figure 15. Kohonen Map of Ruck Data

Note: Data of similar classes indicated by color does not cluster and many node remain unused.

Using artificial neural networks to solve pattern recognition problems requires a tradeoff analysis between system requirements. For an optimum system, the computational complexity must be minimized while at the same time maximizing the classification accuracy. Not only are the two requirements usually mutually exclusive, the relation between these goals is not well understood and are complicated even more by real world considerations i.e. multiplicative and additive noise.

4.5.1 Classical Analysis for Pattern Recognition The method of k-nearest neighbors is used for the initial evaluation of the Ruck and Roggermann Data. Classical methods for pattern recognition dictate using an Euclidian distance measurement to the nearest neighbor exemplar pattern. These distance measurements are compared to the k nearest vectors around the test vector in hyperspace. If the majority of the neighbors are a specific class, the exemplar is classified accordingly. The Ruck and Roggermann sets were analyzed using this criterion. The results are a measure of the internal consistency of the data. Table 4.5.1 shows the results.

Neighbors	Roggermann Target	Roggermann Image	Ruck Image
1	73	82	83
3	77	76	75
5	78	71	67
7	78	73	71
9	77	76	73

Table 2. Nearest Neighbors Percent Accurate Classification

Table 4.5.1 shows, for the three data sets considered, what percent of the exemplars would have been classified correctly if the decision criterion was the class of the majority of neighbors. Column one indicates the number of nearest neighbors considered.

Nearest neighbor analysis shows that the data lacks internal consistency. For any classification scheme to work well, the data should be clustered in some manner.

The table measurements demonstrate that at least about 25 percent of the time, for all three data sets, the nearest neighbor represents a different class. This certainly can affect neural network training. It means that at least one quarter of the time the exemplar pattern conflicts with previous training. In effect the net is being lied to. The result of such lying is that training time is increased, and the net size must be increased. To accommodate such irrationality in the data, the network must memorize the irrational exemplars on a one by one basis.

To test this hypothesis a double disjoint test set was constructed and used to train a MFB network. The test set consisted of four, three-input vectors divided into two output classes. The network could solve this problem with a 2-3 network in about 80 training iterations. The notation 2-3 indicates two nodes in the second hidden layer and three in the first. The notation relates to the graphic displays, with the number of nodes nearest the output, at the top of the screen coming first.

The number in the other layers, input and output is fixed by the problem. By adding a fifth vector which was very close to the forth vector and defining the class as different, it takes a 20-30 network with 30,000 training iterations. By adding only one training vector, the complexity of the problem was increased by a several orders of magnitude. This is a constructed example of an ambiguous decision region. Ambiguous decision regions are the weak point of back propagation. Any improvement to back propagation would need to address this problem.

For these reasons, to determine the size network for a particular pattern recognition problem, consideration must be given to the structure of the input data. The size of the net is related to the number of separate clusters of data in the decision region. If the structure is unknown, the only alternative is trial and error.

Analysis of the Ruck and Roggermann data sets partially demonstrate this hypothesis. Simple back propagation networks were used for analysis with similar results. The dynamic analysis tool was used to identify exemplars which the network had trouble learning. When these exemplars were removed from the training set,

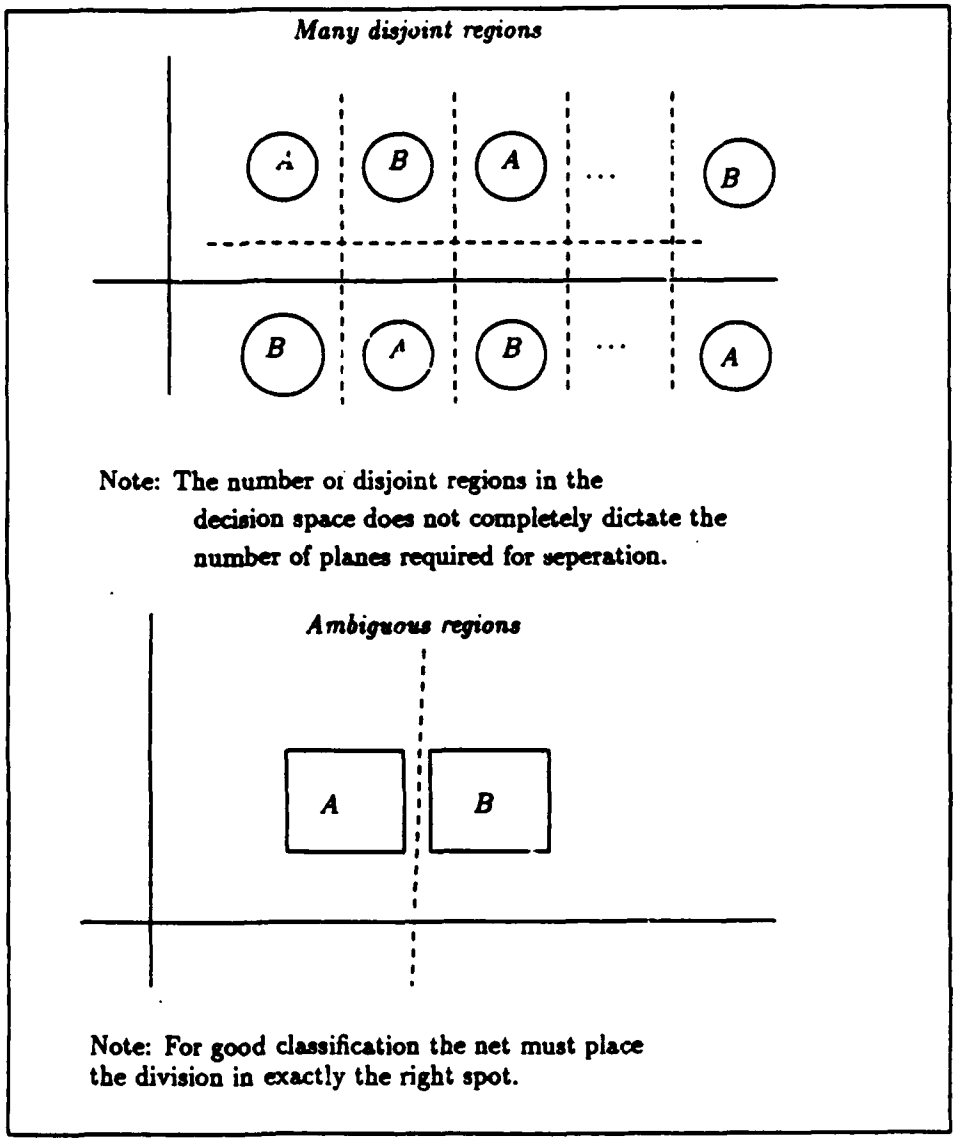


Figure 16. Disjoint and Ambiguous Decision Regions

Note: Multiple disjoint decision regions problems are easier to solve than ambiguous decision region.

the percentage of correct classification became near one hundred percent.

For the details of the Ruck and Roggermann data analysis, see the appendix A and B.

The number of independent decision regions could be determined using self-organization methods. A single layer Kohonen offers a sort of nearest neighbor classification where each output node classifies a particular decision region. Each node, in fact, measures the distance between the average of a number of exemplars just like a k nearest neighbor classifier.

The result of using Kohonen type mapping as an input to another type of network is an intermediate, unclassified representation of the input. The complexity of the data to the output layer is greatly reduced and the requirements for construction of the classifying layers are better understood.

The principle drawback to the generalized self-organizing net is that the unsupervised reduction of the decision region may not result in a distance preserving mapping. Such is the case when the input data is greatly interspersed. It seems that examining the number of nodes required for showing clear winners, that is all nodes are not winning for one class then later winning for a different class, would give insight into the number required for a supervised multilayer perceptron model. *Ambiguous input data is the greatest hinderence to MFB neural network training.*

The next section will discuss a method to reduce the network size for back propagation networks. This method was suggested by Siestma (Siestma,1987). The technique is called network pruning.

4.5.2 Neural Network Pruning The initial weight setting affects delta rule training as well. When using random numbers for the initial settings, sometimes the network will train quickly and sometimes, not so quickly. J. Sietsma has suggested there are additional consequences of poor initial settings. She feels, and experiments have demonstrated, that sometimes more nodes are required to find a solution using

a given weight update algorithm, to a mapping problem than are actually required for a minimum node solution.

As a feedforward back propagation network trains, it appears that some nodes contribute early on, pushing all the weights in the general direction of a solution, but in the end become unnecessary.

As weight values approach a solution, one of several things can happen to a node. First all the weights connected could converge to zero. In another situation, weights will line up with another set of weights causing redundancy. This occurrence has a reciprocal as well; a set of weights leaving a node can duplicate the compliment of another set of weights.

Analysis of the weights for redundancy and impotence would be difficult to automate, but the consequence of these things could be determined by dynamic analysis of node firings. Candidates for pruning could be determined by observing their action under training.

The easiest to spot is a stuck node. Any node that fires all the time regardless of input, can be eliminated by adding, individually, the value of weights to the threshold of the node above it. The second way is to identify tandem nodes. If two nodes are acting in tandem, one can be eliminated. The converse is also true, if two node are acting in compliment to each other, that is, one is high while the other is low, one can be eliminated.

Experiments with node pruning have shown that a large node configuration can be reduced to a very small configuration. Using the Ruck data, a two hidden layer 20-26 net could consistently be reduced to a 6-5 net. While consistent convergence could only be obtained on the larger net, occasionally convergence could be obtained with as small as a 5-5 net and still retain 100 percent accuracy on the training set and 74 percent accuracy on the test set.

Node pruning demonstrates that fewer nodes are needed for an implementa-

tion of a neural net problem than are actually necessary for training. Still while pruning is useful for back propagation network optimization, it left unanswered the basic question. How many interconnections are require to solve a particular problem? Kung (Kung,1987) tries to answer this question by using algebraic projection analysis.

The number of hidden units per layer dictate the space partitioning separability of the network. The more units per layer, the finer the partition of the decision region. The work of Kaczmarz suggests that the number of nodes required can be calculated (Kung,1987). Unfortunately, a priori knowledge about the decision space is required. This restriction is not acceptable in most real problems as decision region topology cannot be determined without statistical analysis of the data.

The solution may lie in self-organization techniques which map the input data into well behaved functions. These functions could take advantage of the Kaczmarz Algebraic Projection technique by forcing the data into a well behave decision space. A candidate is the Kohonen learning rule.

Unfortunately, as shown previously, the Kohonen decision regions are not always well behaved. A first effort to study the integration of two system is the counterpropagation network. The next section discusses counterpropagation which may be modified to make the decision regions better behaved.

4.6 Counterpropagation

Counterpropagation networks (CPN) are described in an article by Robert Hecht-Nielsen (Hecht-Nielsen,1987). A CPN network was implemented in the lab without good results, except for simple problems.

Counterpropagation is a multilayer feedforward network with a Kohonen organizer on the first layer with a Grossberg outstar on the second. It has two additional layers use to train the network.

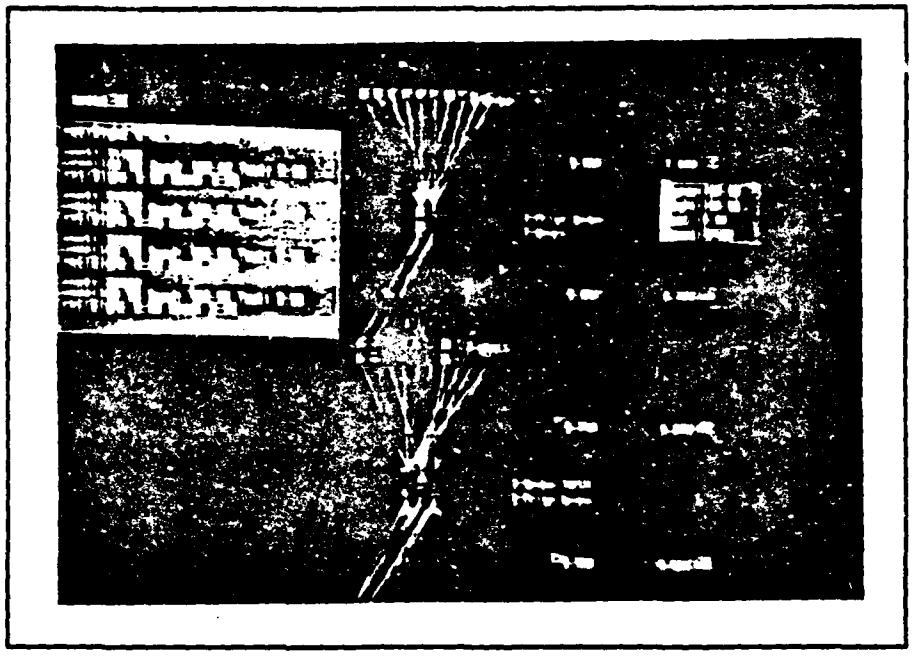


Figure 17. Counterpropagation

The counterpropagation algorithm was implemented exactly as suggested by Hecht-Neilsen and tested against the Ruck and Roggermann Data. Classification accuracy was significantly less that the accuracy from a MFB. On the Ruck data, while 100 percent classification was a achieved on the training set, only 50 percent accuracy was noted on the test set. The MFB classified at 100 percent training, and 74 percent test. See appendix A and B.

As suggested by Donald Woods et al (Woods, 1988) two modifications were made to the Kohonen layer of the counterpropagation. Woods, Hecht-Neilsen and Deseino all suggested conscience be added to counterpropagation and Kohonen maps. Deseino suggested an implementation. (Deseino, 1988). A Kohonen layer with a conscience is the first step to the hybrid network.

4.6.1 Conscience Experiments related to the thesis effort, indicate that Kohonen works well for data sets calculated from some types of probability functions. For example, a two dimensional input based on a variety of probability density functions, including constants, Gaussian, and chi-squared were tested. By defining a region in space to be a particular class, then color coding the classes, data clustering could be tracked by noting winning regions on the Kohonen map. Classes seemed to separate well for calculated data. As expected, the number of nodes which won for a particular class was related to number of times the class was used as an exemplar. The results were mixed for higher dimensional data. Three and four inputs were tried, and completely fell apart for measured data like the Ruck and Roggemann sets.

Because of the mixed results, modifications to the Kohonen training were considered. A few suggestions were found in the literature. Rumelhart and Zipser suggested two things. The first was to add a distance bias to each processing element. If a node wins a competition the threshold is increased so the chance of winning is decreased. The second was to update not only the winning weights and neighbors, but the losing weights as well just not as much.

Both fixes would tend to force losers, as well as winners, toward the centroid of the input data vectors. Duane DeSieno suggested an elegant algorithm for implementing these concepts. He called it a conscience.

The process requires adding a data structure to the Kohonen net. The structure can be implemented as an array equal in size to the array of Kohonen nodes. For discussion purposes, these are referred to as probability bias nodes.

The algorithm impacts the training rule in two places. First in calculation of the probability bias and second in the calculation of the winning node for the Kohonen competition. The actual update of the weights remains the same.

The probability bias array is filled according to the formula:

$$p_i^{new} = p_i^{old} + B(y_i - p_i^{old}) \tag{5}$$

The constant B is an adjustment parameter. DeSieno suggests 0.0001. C is a constant which controls the influence of the conscience. The larger C, is the more probable that all nodes would win the competition an equal number of times. A value of one was used for these experiments. The rule to select the winner of the competition is modified to add a bias term to the distance before selecting the winner. The bias term is:

$$b_i = C(1/N - p_i) \tag{6}$$

where N is the number of nodes in the competitive or Kohonen layer. The conscience rules were implemented on a hybrid Kohonen-perceptron net with good results. After implementation, it becomes obvious why many Kohonen nets don't work well for measured data. All the data points are closer to each other than any of the randomly set initial values. A handful of nodes will dominate the training. The rest will never be adjusted.

The experiment offered insight into several elements of Kohonen training. One consideration is the relation of the initial weight settings. If the initial weight settings are orders of magnitude away from the input vectors, the first vectors adjusted will dominate the training. In other words, if the initial neighborhood size doesn't cover the entire map, any vector not adjusted the first time through will probably never win a competition again. If the magnitude of the training vectors are all very large and the initial weights are set small, on the first pass one weight will win and there after, always win.

Conscience inserts a probability bias into the distance comparison before the winner is selected. Consequently, all nodes should have an equal probability of firing.

A difficulty was encountered selecting the parameters B, and C. Work with the hybrid model suggests that the value suggested by DeSieno is not generally applicable

to all problems. Using a value ten times his value yielded better results. The other value C should be fixed at one. DeSieno actually uses two adjustments where only one is needed. Either of the two equations could be used for conscience alone. A more intuitive approach would be to set the probability bias to:

$$p_i = count_i/COUNT_{tot} - 1/N \tag{7}$$

$count_i$ is the number of times a node has won. Barmore arrived at a similar equation independently (Barmore, 1988), he uses a win rate disparity of 1.5 before the conscience inhibits a node from winning. When B is eliminated, and C in the second equation becomes a relation between the excess wins, a small number between (-1/N) and (1/N - 1), and a weighing factor which allows the conscience to kick in sooner. The difference between the DeSieno method and that used here is that if p_i is greater than an arbitrary small value the node is remove from the competition completely. The DeSeino equation has the effect of a weighted, conditional removal based on the number of times a node has won.

Both equations work well, but the second is a little easier to adjust to different ranges of data. The results demonstrate why a counterpropagation model has little chance of working outside of well behaved input vectors and initial weights set to represent a distribution similar to the training set.

4.6.2 Weaknesses of Counterpropagation The counterpropagation model has three primary weakness. It lack conscience; usually, only a few nodes win the distance competition for most real data sets. Second, with normalized inputs all vectors are mapped to the unit hypersphere. Finally, counterpropagation isn't able to solve the disjoint region problem well.

Since the input vectors must be normalized, exemplars are limited to the unit hypersphere, no differentiation can be made between vectors which are related by a scaling factor. The problem can be overcome somewhat by minimizing the distance

measurement instead of maximizing correlations. In this way both (1,1), and (2,2) will be mapped uniquely in the decision space. This difficulty is overcome by replacing the dot product of the input with the weight vector, with a simple distance measurement.

A third weakness is partially a result of using the Grossberg outstar for classifying the disjoint regions. To solve a disjoint region problem the output to the classifier should have an unambiguous one or zero at the output node. If a Kohonen node sometimes wins for class *a* and sometimes for class *b*, the outstar weights will oscillate between the two. Convergence is not possible. Using conscience can ensure a more efficient Kohonen layer to prevent ambiguous nodes, that is nodes which fire for either class.

4.7 The Hybrid Network

By taking the weaknesses out of the counterpropagation network a more powerful network is established. A conscience is added to ensure maximum efficiency of the Kohonen layer. A multilayer perceptron replaces the outstar layer. The final step in constructing a hybrid net is to improve the interface between the Kohonen and perceptron layers. The counterpropagation model passes along only one piece of information to the upper layers, the Kohonen competition winner. The winner specifies which set of weights most closely reflects the input vector. More information may assist the upper layer in making a classification. Hecht- Neilsen suggests passing more information to the upper layer. He suggests passing not only the winner, but second winner, third etc. The hybrid network inherits the same weakness, only one node at a time can pass information to the multilayer perceptron model. To improve the performance of the hybrid net a different interface should be used to pass more information to the perceptron layers.

The obvious solution, to pass the distance directly, was tried with only partial success. For small toy problems, convergence was found easily. The model was

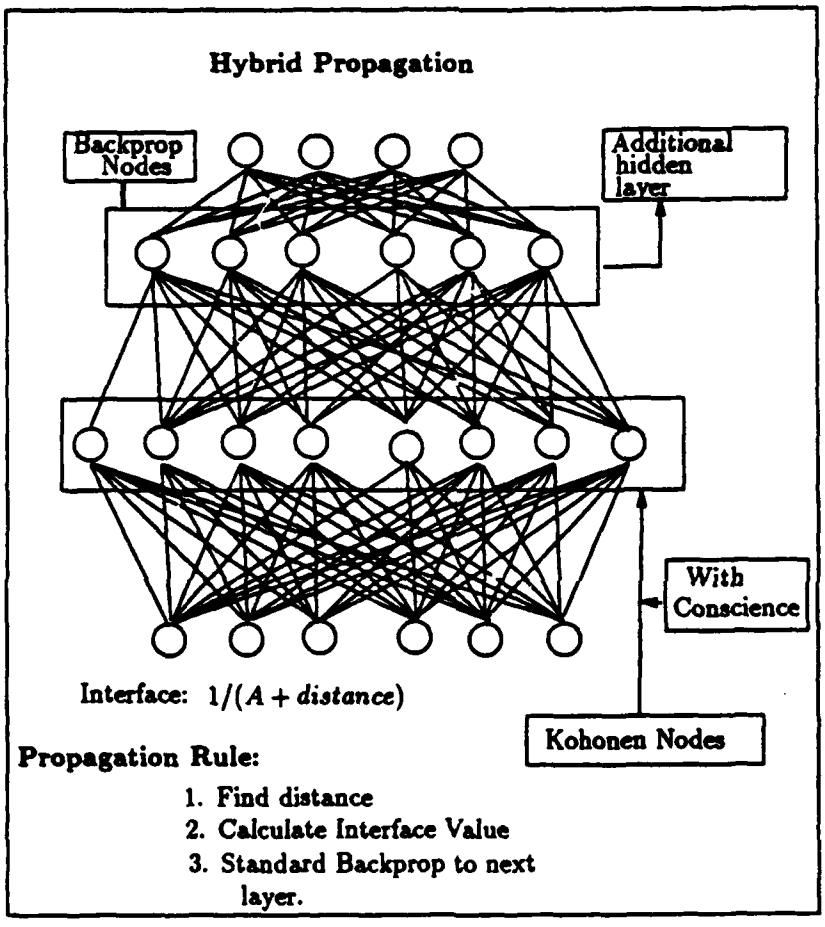


Figure 18. The Hybrid Network

Note: First hidden layer uses kohonen update rules for weight adjustments and Second hidden layer used back propagation.

constructed and tested. For noisier data, like the Roggemann set, convergence is not possible for more than a few samples.

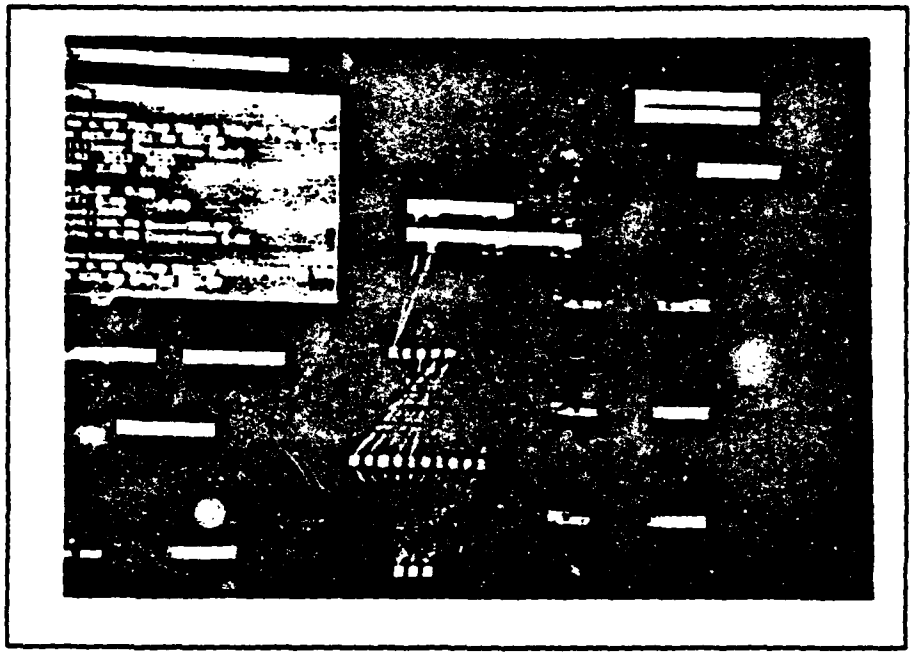


Figure 19. Hybrid Propagation Environment

The solution is based on sending a weighted composite of the winners. In other words, a high number is sent for the winner, a sightly lower number for second and so on. A convenient function that performs this conversion is similar to the sigmoid function.

$$x_i = 1/(A + k_i) \tag{8}$$

The value x_i is a distance measurement for a particular node. A is a scaling constant less than one. A typical value for A is 0.05. This value inversely scales all distances to a number between zero and twenty. The improvement in the performance of the net is dramatic.

The result of using Kohonen type mapping as an input to another type of network, is a new intermediate, unclassified representation of the input. The complexity of the data to the output layer is greatly reduced and the construction of the output net is much better understood. Using a rule of thumb, three first hidden layer nodes for every Kohonen node, the network always converged in few thousand training iterations. Network pruning seems to reduce the number of interconnections even more.

The difficulty of this approach is the intermediate representation of the data may not be a good vector quantization of the input. Because the weight vector associated with each Kohonen node tends towards the average of a number of close exemplar, convergence is guaranteed if the number of elements in the Kohonen layer is greater than the number of distinct decision regions in the input data. That is, if all the nodes are used. This is something unknown a priori. The number is bounded by the number of exemplar vectors used for training. As the number of Kohonen units approaches the number of exemplars, the solution degenerates to a simple table look-up without generalization. To test the hybrid network, it is important to demonstrate exactly what type of problems are better solved using a new type of network. At first it was considered that the hybrid net would scale better for larger numbers of disjoint regions. This hypothesis proved incorrect. Back propagation scales very well. The reason, of course, is obvious. As the number of disjoint region increases, each new region can take advantage of previous hyperplanes to partition the decision region. By appropriately placing the decision regions, sometimes no more hyperplanes are needed to make the distinction. The real advantage was unexpected. By creating ambiguous decision regions, that is placing two exemplars of dissimilar classes close together, hybrid propagation performed better.

The first experiment involves using four exemplars with a fifth used as a spoiler. The fifth exemplar was placed a small distance away from the fourth exemplar as indicated by the distance column.

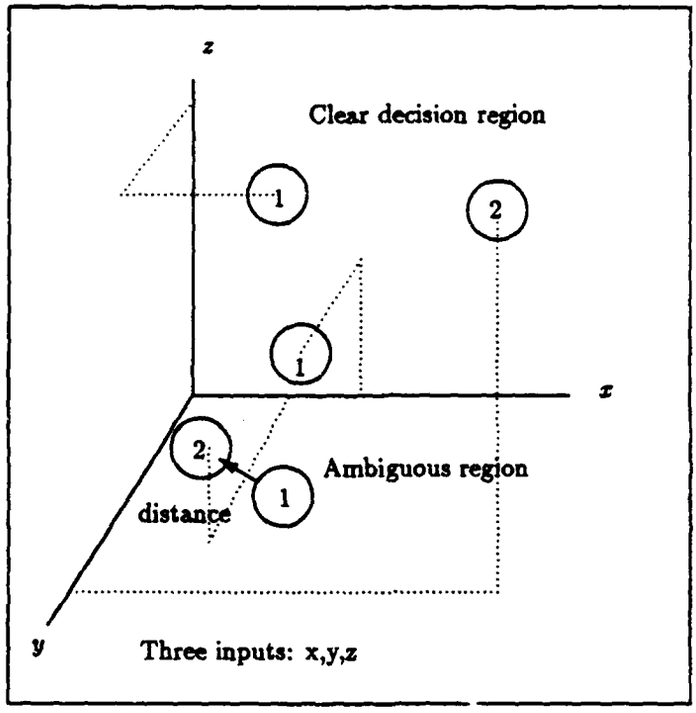


Figure 20. The Hybrid Network Test Problem

These measurements are shown as five regions. In the second test the number of spoilers was increased to four, consequently every decision region was made ambiguous.

			MFB		Hybrid	
Net Size	Regions	Distance	Cycles	Accuracy	Cycles	Accuracy
2-2	5	1	800	100	1000	100
3-5	5	0.5	10,000	100	1500	100
5-7	5	0.4	4700	100	1500	100
5-7	5	0.2	14,000	78	6000	100
5-10	5	0.2	20,000	87	6000	100
10-20	5	0.2	20,000	85	5000	100
5-10	5	0.1	50,000	86	10,000	100
5-10	8	0.5	18,000	100	10,000	100
8-15	8	0.3	26,000	100	10,000	100
10-20	8	0.3	18,000	100	10,000	100
5-10	8	0.1	50,000	0	2,500	100
10-20	8	0.1	50,000	0	5,500	100
15-30	8	0.1	50,000	0	5,500	100

Table 3. Back propagation vs Hybrid Net

Note: In every category hybrid propagation out performs Back propagation for this class of problem. The problem under consideration is three input, two output disjoint region problem. The first case uses five exemplars. Four are distinctly separable with one ambiguous exemplar. The ambiguous exemplar is the same as a one of the other exemplars, but moved an arbitrary distance away and the class is changed. This distance is noted in the distance column. The second case increased the number of ambiguous decision region to four, each exemplar has a very close compliment of the opposite class.

4.8 Summary

The hybrid net seems to work similar to a nearest neighbor classifier. The operator selects a number of Kohonen nodes. The number is an estimate of the number of decision regions. In practice a few more are required due to inefficiencies in the conscience. The Kohonen layer training moves the weight vectors attached to these nodes toward the average of a group of exemplars. For example: if five

Kohonen nodes are selected, then each of the five nodes would represent an average of the group of exemplars in a region around them.

The experiment is based on the assumption that only a limited number of data clusters exist in the decision space. If the training vectors are consistent then each should lie within a reasonable distance of these data clusters. Given that the number of data clusters cannot exceed the number of training vectors, it seems reasonable to assume that the complexity can be controlled by limiting the number of vectors used for training.

The question becomes, how well does this limited set of training vectors relate to the entire ensemble of input vectors. If the training set were perfect, the data clusters would be identical to the number of classes. The worst case would be where there is no clustering of data at all. In this case the number of independent decision regions is equal to the number of exemplars. A typical case is some number in between.

The hybrid net expects one Kohonen node for each data cluster. Because of the conscience training algorithm, in practice more nodes are required to ensure good separation between nodes. The back propagation network stacked above the net assumed simple decision regions, and was set to twice the Kohonen layer for the first hidden and equal to the Kohonen for the second hidden layer. Experience has shown that for the simple, binary type output of the Kohonen layer, these values are sufficient for convergence. Actually, using a multilayer back propagation net for the output of the Kohonen layer is more than is necessary. For this simple classification problem Hopfield or Grossberg outstar would be more efficient and train faster. The back propagation method will allow continuous data as input vectors. This flexibility allows the use of more complicated functions as the interface between the Kohonen layer and the back propagation layer.

The hybrid Network offers a solution to two difficult problems encounter in other networks. The multilayer perceptron requires long training times when ambi-

guities exist in the decision regions. The counterpropagation network requires less training time but is unable to solve the disjoint region problems without efficient use of the Kohonen layer. The hybrid net offers a compromise. The hybrid net can obtain k nearest neighbor accuracy in a few tens of thousands of training iteration.

V. Recommendation and Conclusions

Developing the hybrid neural network embodied four separate aspects or phases. The major effort involved the development of an environment for dynamic analysis of neural networks. Using these tools, the next phase involved the analysis of several commonly used neural network topologies: Kohonen maps, back propagation, and counterpropagation. After gaining insight into how the networks function, several data sets were analyzed. The final step in the process was to take the information gained in the first three parts of the study and develop a better learning process. The results of these efforts are the Hybrid network and the NeuralGraphics software package.

5.1 Graphics as an Analysis Tool

Graphical analysis of a neural network under training is useful in several respects. The effectiveness of a particular node can be observed, tested or the node can even be removed. The pruning function of the study environment was used to demonstrate this. Weaknesses in a particular training method could sometimes be detected. For example, graphic displays demonstrated that the Hecht-Nielsen Kohonen layer update rules resulted in only a portion of the nodes being used at all. Once detected, conscience was added and the efficiency of the Kohonen layers for both counter and hybrid propagation was improved.

Error surface analysis allowed two different weight update rules to be compared. Progress of the error reduction as a function of training cycles could be tracked from the initial conditions to the final solution. So much could be observed in the displays, that it wasn't possible to investigate but a portion of the anomalies noticed during observation.

Spotting weaknesses in training methods is a useful application of graphics to neural networks. The fact that the Kohonen nodes were not all contributing to

a solution, was obvious with a graphic display. Early versions of Kohonen based classifiers showed that only a small portion of the nodes were consistently winning the competition. This lead to the inclusion of conscience in the training algorithm.

Examination of the ranges in the weights lead to a better understanding of "pathological networks", network no longer capable of learning. When the weights are starting to grow without bound, no amount of training will bring the network back toward convergence, and training could be terminated. Other anomalies were noted as well. For example, it was noted that generally the majority of the weights were zero. The first major contribution of this research was the graphics environment which allowed detection of network deficiencies. Discovery of these deficiencies allowed new neural network training paradigms to be designed.

5.2 Criterion for measuring error

A difficulty was encountered in establishing a criterion for measuring the performance of a neural network. When comparing the results reflected in this effort with results of similar studies, different error criterion prevented exact comparison. This study uses three methods to measure error. Each of the three methods varies from a very strict criterion to a very loose measure.

For comparisons of one data set to another, it seemed most convenient to use a very strict measure of error. A classification of correct for a given input vector indicated that the difference between the actual output and the desired output was below a specified threshold for every output node. For a non-ambiguous classification a value of 0.2 was used. For a correct, but less definitive classification, a value of 0.5 is used.

Unfortunately, as the width of the output vector grew, as with the sine wave problem, the less likely it becomes for all nodes to classify correctly simultaneously. Consequently, measured success of the network appears to go down. For the Fourier filter problem, a minimal criterion is used. Percentage of error is based on each

independent decision. Changing the error criterion for the Fourier problem was necessary to prevent misinterpretation of the results. Under the former method the networks would show only 80 percent accuracy, while identifying each individual sine wave 98 percent of the time.

Using and understanding different error criterion is important for two reasons. First, for comparisons with other work, the methods used must be at lease relatable. Second, unlike most problem solving techniques, the goal is not to determine a 100 percent accurate solution. The goal is to arrive at a reasonable guess as quickly as possible. For these reasons, understanding the error criterion becomes central to neural network study. Comparison of different techniques is a matter of evaluating training times and classification effort for a solution that is arbitrarily close to 100 percent. If the completely accurate solutions are necessary, some technique other than neural networks may be more appropriate. In the context of problems where absolute accuracy is necessary, neural networks may be used to limit the search for non-neural network solutions.

After establishing an error criterion appropriate for a particular problem, the next step was to construct a network with the minimum computational effort. The computation effort required for a given problem is related to the size of the network and the number of interconnections.

5.3 Determining Network Size and Training Time

The questions: how long should the net be trained, and how many nodes are required for a particular problem ? occupied many hours in this effort. The author's conclusion is that the answer these questions depends almost entirely on the data. For each problem considered here, the number of disjoint and ambiguous decision regions in the data set dictated the size of the network required. Also, the results seems to indicate that it takes a lot fewer than previously thought. A single node seems very powerful. As pointed out by the Ruck and Roggermann data, training

until no changes are detected in the test set accuracy may be good enough for real problems. It appears that any reasonable problem can be solved in a few thousand iterations. Training after that point is an act of memorizing the special cases. That is, when a tank exemplar is presented to the net for training and nearest neighbors (in the decision space) are jeeps, the net must memorize what makes that tank different from all the jeeps around it. Such memorization does not apply to real world problems. Generally, classification will be made on unique data taken directly from a sensor system. Not only is this memorization very expensive in training time, but requires additional nodes to handle the memorization of particular exemplars. If the function of the network is table look-up, the additional resource may be justified. But, if the data set is infinite, as data coming off the backplane of a camera would be, a net only needs to learn generalities. Two factors were noticed which affected the smallest possible network, momentum and network pruning. A first effort to establish a minimum node size tended to overshoot the optimum. Overestimation may be a natural consequence of using a biological model. With the human brain using hundreds of billions of neurons to solve problems, the temptation is ever present to use more than necessary.

Network pruning is a good example. The first published analysis on the Ruck data used a 100-200 network (Ruck,1987). Subsequent efforts here reduced that number to 10-10. Network pruning reduced that number to 6-8. Using the momentum term, reasonable results were obtained with a 3-5 net. With this information the Roggermann data resulted in a 20-50 net which pruned to an 18-47. The authors conclusion is that pruning is most useful when too many nodes are present. When starting with a near optimum number of nodes pruning is only superficially useful. The momentum parameter may be just as important as pruning.

Most of the first neural network models did not include a momentum factor in back propagation algorithm. The computational effort to include momentum did not seem a fair trade off for decreased training cycles. By saving all previous weights

each time, computational effort was predicted to increase, at a rate greater than the expected reduction in training time.

Late in this study, the Piazza work (Piazza,1988) suggested that momentum carried additional benefits, benefits both in accuracy, and the ability to jump over local minima by smoothing out the error surface. With the inclusion of momentum, net size on the Ruck data reduced from a 10-10 net to a 3-5 net with only a tiny reduction in accuracy (5 percent training data, none for test data). No additional benefit could be gained from pruning. Going from 200 nodes to 8 suggests that each hidden node may be more powerful than previously thought.

Each node can be considered more powerful still, if a node is not expected to memorize specific exemplars, but only general trends in all the exemplars. The Hybrid network is an attempt to exploit this observation.

The second major contribution of this effort was the study of how network training and node requirements relates to the problem under consideration. This research effort showed what types of problems are difficult and how these difficulties can be avoided.

5.4 Application of the Hybrid Network

The hybrid network is an extension of Kohonen mapping and counter propagation. Using a simple distance metric, hybrid propagation maps the exemplar patterns into a new decision space. This new decision space seems to reduce the burden on the back propagation classification layer.

The Hybrid network required an additional layer of hidden nodes, so the training time increased some to allow the Kohonen layer to organize. Also, the first hidden layer usually needed to be larger than the first hidden layer in back propagation alone. However, discounting the time to organize the Kohonen layers, training times were reduced by a third to a half and the network showed a greater ability

to memorize. This ability was noted in analysis of the Roggermann target data analysis.

In general, the Hybrid net works better than a multilayer perceptron for problems that include ambiguous decision regions. For small clearly distinct decision regions, the multilayer perceptron seems to be more efficient more efficient.

The third major contribution of this research was the hybrid network which outperformed other networks configuration for specific types of problems.

5.4.1 Summary Of the four neural networks considered, each had its own strength and weakness. The Kohonen maps cannot solve classification problem without some interface to a classification network. Both Kohonen and counter-propagation were inefficient without augmenting the weight update rules to include a conscience. Even with a conscience, the nodes were not one hundred percent efficient, mapping one node to a one decision region. With a Multilayer perceptron, as the number of disconnected disjoint decision regions increased, the number of nodes required to solve the problem increased in a non-linear fashion.

The Hybrid network shows promise of being able to solve the the ambiguous decision region problem of the back propagation algorithm. As dissimilar data points moved closer together, the Hybrid network the number of nodes required seems to increase linearly. Also, the Hybrid network is a useful tool for data analysis. Still, more work could be done to improve the efficiency of the Kohonen layer. The conscience rule doesn't seems to map exactly from one decision region to one kohonen node. This may be possible by using some other paradigm to train and set the size of the Kohonen layer.

5.5 Recommendations

The Neural-Graphic study environment is like a window into a complex mathematical process. For every question, answered several more were raised. For ex-

ample, it was noted that generally the majority of the weights were zero. Most of the questions associated with these types of observations went unanswered. Future investigation may want to consider the effects of removing near zero valued weights.

In implementing neural network in dedicated processors, real world constraints should be considered. An important consideration for making integrated circuits would be the constraints on the size and accuracy of the weights. In a chip these weights could be implemented as resistors. Further investigations should try hard limiting the weights to different ranges. Also, the dynamic ranges requirements could be explored by adding varying degrees of noise to the weight values. This would provide insight into how accurate the weight values would have to be.

The NeuralGraphic software package allows rapid study of segmentation and vector quantization paradigms. As many government organizations are purchasing commercial software to perform these types of task, making this package available could save the Air Force thousands of dollars. To make this package of commercial quality would only require porting the source code to several types of machines. Although NeuralGraphics was written for a Silicon Graphic IRIS, with only small changes it could be made to run on a Sun workstation, a Micro Vax III or even a mainframe Vax (without the graphics).

Appendix A. *Ruck Data Analysis*

The Ruck data set is a collection of 52 exemplar vectors and 27 test vectors extracted from laser radar range imagery. The input vector features are base on Zerneki moments which offer position, shift, and scale invariant to the exemplar pattern.

The network configuration consists of 22 inputs and four outputs. The four output indicate classes related to tanks, jeeps,trucks and POL tanker vehicles. The number of hidden units is indicated by the column Netsize in each of the tables. The first algorithm used to evaluate the data set is a multilayer feedforward backpropagation network, without theta training and momentum equal 0.7. An η of 0.3 was used. Several values between 0.3 and 0.1 were tested each without any significant difference in training times or accuracy. The only difference noted was that using smaller values of η caused the training to converge in a smoother fashion.

The training statistics are based on 1000 random samples pulled from the training set. The test statistic are calculated by sequential classification of a separate test set and calculating error statistics.

A classification of *right* means that for every in-class vectors a value of 0.8 and above was found at the output together with a value of 0.2 and below for each out of class node. A guess is an indication of an ambiguous output, but still above 0.5 for the in-class node and below for the out of class node.

A.1 Backpropagation Rules

For a base line estimation of the number of nodes required, several net configurations were to 50,000 training cycles with the results shown in table 4.

Considering that the Ruck classification problem can be solved with a 3-5 system (three in the first hidden layer and five in the second), improvement would

Netsize Lower	Upper	Training Right	Guess	Test Right	Guess	Iterations	Converging
3	5	59.1	74.7	89.5	86.1	50,000	No
5	5	59.1	74.7	95.2	97.8	50,000	Yes
10	5	66.6	75.0	94.9	97.4	50,000	Yes
10	10	68.2	74.7	100.0	100.0	50,000	Yes
15	15	69.2	74.2	100.0	100.0	50,000	Yes
26	20	74.4	76.2	100.0	100.0	50,000	Yes

Table 4. Percent Accuracy vs Net Size

Note: reasonable good results were obtained for the 3-5. Nets larger than 10-10 did not improve accuracy.

be difficult. Convergence with such a small system indicates that the data is well behaved with only a handful of disjoint regions and no ambigous regions. If this is the case, improvement can't be expected for either a hybrid network or a counter propagation network.

A.2 Counter Propagation

Counter propagation allows convergence in only a few thousand training cycles, but the generalization properties are poor. Convergence here means 100 percent accuracy on the training data. Although the training set classifies at 100 percent, the test set shows only 50 percent correct classification. The baseline for the test data is 74 percent.

Probably the poor performance cannot be attributed directly to the counter propagation paradigm. The Hecht-Neilsen model for CPN does not use the Kohonen nodes effectively because a conscience mechanism is not included.

A.3 Hybrid Propagation

Table 6 shows the same tests using a hybrid network.

Kohonen Nodes	Training Right	Test Right	Guess	Cycles
20	56.3	9.4	13.8	15,000
30	56.8	11.2	18.1	15,000
60	100.0	29.9	50.0	15,000

Table 5. Counter Propagation:Kohonen Node vs accuracy

Note: Counter propagation demonstrates very poor generalization with good convergence.

Net Size Kohonen	First	Second	Training Right	Guess	Test	Cycles
10	5	0	66.2	80.2	74.7	25,000
15	10	0	70.3	85.1	70.4	25,000
20	10	0	73.3	88.4	70.4	25,000
30	15	0	94.3	98.2	74.7	25,000
60	30	0	100.0	100.0	74.0	25,000

Table 6. Hybrid Net: Kohonen Nodes vs Accuracy

Note: The Hybrid net compared to the BPN seems to trade training time for number of nodes while retaining same accuracy.

A.4 Summary

As expected, the simple backpropagation rule outperformed all both counter propagation and hybrid propagation. While the CPN did converge, the poor performance on the test set demonstrated an inability of the network to generalize. The Hybird net classified data equally as well as the BPN, yet more nodes were required to obtain the same performance. The advantage of the Hybrid net was a reduced training time.

```
/*****************************************************************************
*
*       DATE:  3 Oct 1988
*       VERSION: 1.0
*
*       NAME: General Neural Net-Main Loop
*       MODULE NUMBER: 1.1
*       DESCRIPTION: General Purpose Neural Network with Node Pruning
*       ALGORITHM: Werbos BackPropagatio
*       FILES READ: Weights , data files
*       FILES WRITTEN: Log file, Weights Stored
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: Main Program Shell
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/

#include "net.h"

struct neural_net net;

#define wait(A) if((count%(A))==0)

extern int menu();
int stopit=1000000;
int count=0,right,good,guess,test,display = 100,show_weights=100;
extern int decisions,fast;
extern int exam_test,examplars;

main()

{

/* For initialization only */

   hide_one = 10;

   hide_two = 10;

   INITIALIZE();

   DISPLAY_NET();

   while (count < stopit) {

/*  Reading the mouse allows magnification and node pruning  */

       check_mouse();
```

```
        MAKE_INPUT(net.inp,net.doft,-1);

        FEED_FORWARD();

        CHECK_ERRORS(net.outp,net.doft);

        TRAIN_NET();


        wait(display){ SHOW();
                       if(exam_test!=0)DO_TEST();
                        calculate_error();
                        make_graph(600.,470.);
                        DISPLAY_NET();

                     }
        if(fast == 1 ){DISPLAY_NET();}
        count++;
        display_count();
                              }

/*  Save weights on termination  */

write_std_weights();

}
```

```
/*************************************************************************
*
*       DATE: 11 Aug 1988
*       VERSION:2.1
*
*       NAME: Definitons.h
*       MODULE NUMBER: 1.2
*       DESCRIPTION: Standard Definitons and Macros
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/

#define GRAPHICS TRUE

#define TABLE 64
#define Bytes_color 256                                                  30

#define loopi(ON) for(i=0;i<ON;i++)

#define loopj(ON) for(j=0;j<ON;j++)

#define loopk(ON) for(k=0;k<ON;k++)

#define loopij(ONE,TWO) for(i=0;i<ONE;i++)for(j=0;j<TWO;j++)

#define HARD_ON  RED                                                     40

#define INDETERM (TABLE/2)

#define HARD_OFF BLUE

#if FALSE

#define TRUE 1

#define FALSE 0                                                          50

#define output 4
```

```
#define input 22
#define hide_one 20
#define hide_two 28
#endif
#define line printf("\n")
#define disply 100
#define datafile "..\data\ruck.data"
#define spcing 200
#define weight_s 20
#define video 1.00
```

```
/*****************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 1.0
*
*       NAME:  Net.h
*       MODULE NUMBER: 1.3
*       DESCRIPTION:  Structure Definition for the Neural Net
*                                                                          10
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/

#define size 100

/* Critical Definition . memory allocated at compile time */

struct neural_net {                                                        20

  float outp[size];

  float doft[size];

  float w2[size][size];
  float w2_mom[size][size];

  float aw2[size][size];

  float t2[size];                                                          30

  float outp_mask[size];

  float y2[size];

  float y2_mask[size];

  float y2_dt[size];

  float w1[size][size];                                                    40
  float w1_mom[size][size];

  float aw1[size][size];

  float t1[size];

  float y1[size];

  float y1_mask[size];                                                     50

  float y1_dt[size];
```

```
float v0[size][size];
float w0_mom[size][size];

float aw0[size][size];

float t0[size];

float inp[size];

float inp_mask[size];

 int class_test[size];
 int class_count[size];

} ;

int output,hide_one,hide_two,input;
```

```
/*****************************************************************************
*
*       DATE: 3 August 1988
*       VERSION: 1
*
*       NAME: Initialization Module
*       MODULE NUMBER: 1.4
*       DESCRIPTION: Initialization of the display hardware,
*                    any input test data, and net data structure.
*
*       ALGORITHM: None
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Net
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/

#include "definitions.h"

#include"gl.h"

#include"device.h"

#include <time.h>

#include <stdio.h>

#include <signal.h>

#include "net.h"

extern int count;

extern struct neural_net net;

float random();


extern int menu();

INITIALIZE()
```

INITIALIZE

```
{
    signal(SIGINT,menu);
    init_screen();
    menu();
    init_net();
}

init_net()
{
  int i,j;
  new_net();
  count = 0;
  loopij(size,size) net.w2[i][j] = random();
  loopi(size) net.t2[i] = random();
  loopij(size,size)
      net.w1[i][j] = random();
  loopi(size)net.t1[i] = random();
  loopij(size,size)
      net.w0[i][j] = random();
  loopi(size)net.t0[i] = random();
  loopi(size) net.y2_mask[i] = TRUE;
  loopi(size) net.y1_mask[i] = TRUE;
  loopi(size) net.inp_mask[i] = TRUE;
  loopi(size) net.outp_mask[i] = TRUE;

  }
/* Generate a random number between -0.5 and + 0.5 */
float random ()
{ float x;
  int y;
```

init_net

Appendix B. *Roggemann Data Analysis*

The Roggeman data set consists of two problems. The first is a simple target/non-target classification based on statistical relations of an input scene.

The second problem takes a target identified from the first part and using an expanded vector quantization of the scene make a specific classification of vehicle type.

B.1 Target/Non-Target Classification

The input data is a vector quantization of three statistical features of the image data. The first value of the input vector is related to the ratio of the height to width of a *blob* segmented from an infrared image. The second feature is a ratio of the energy in the *blob* to the energy in the background. The third number represents the standard deviation of the *blob* pixel values.

The network configuration consists of three inputs and two outputs. The two outputs indicate target (1,0) or a non-target (0,1) with one going high while the other goes low. The number of hidden units is indicated by the column Netsize in table 7 for two hidden layers. The first algorithm tested uses a multilayer feedforward back propagation network, without theta training and without a momentum term. An η of 0.3 was used. Several values between 0.3 and 0.1 were tested each without any significant difference in training times or accuracy. The only difference noted was that using smaller values of η caused the training to converge in a smoother fashion.

The training set consisted of 600 samples. The training statistics are based on 1000 random samples pulled from the training set. The test statistic are calculated by sequential classification of a separate test set and calculating error statistics.

A classification of right means that for every in-class vectors a value of 0.8 and above was found at the output together with a value of 0.2 and below for each out

Netsize Lower	Upper	Training Right	Guess	Test Right	Guess	Converging
10	10	3.7	52.1	2.1	60.3	No
20	20	30.1	61.0	21.0	81.0	No
20	10	40.0	72.0	48.0	52.0	No
30	10	46.7	76.3	54.0	73.0	Nc
30	30	53.4	63.10	63.0	81.0	No
40	10	48.5	75.2	43.0	75.0	Maybe
40	40	65.4	71.7	64.0	74.0	Maybe
50	10	49.9	72.8	56.0	72.0	Maybe
50	50	70.1	70.6	75.0	76.0	Maybe
100	10	55.2	65.2	55.0	60.2	No
100	20	66.4	81.3	51.0	68.2	No
100	30	68.9	77.7	62.0	67.2	No
50	20	58.7	82.2	50.0	72.0	Yes

Table 7. MFB: Percent Accuracy vs Net Size

Note: The 20-50 net seems to be the most efficient and accurate.

of class node. A guess is an indication of an ambiguous output, but still above 0.5 for in class node and below for out of class node.

Training was extended for network configurations that appeared to have the best chance of converging. Training was stopped when accuracy rates appeared stable or no longer converging.

B.2 Backward Propagation Rules

For a base line estimation of the number of nodes required, several net configurations were to 50,000 training cycles with the results shown in table 7.

Since the problem consisted of over 800 exemplars, as expected, a larger net is required. The large size of the data set indicates a greater probability several disjoint regions. The fact that the data never did better than a about 80 percent indicated ambiguous decision regions. As expected, training time also increases and to bring the network up to the maximum performance requires over 500,000 iterations.

Netsize Lower	Upper	Training Right	Guess	Test Right	Guess	Training Cycles
50	10	59.2	78.7	57.0	70.0	100,000
40	20	79.0	80.7	53.0	58.0	100,000
50	20	67.7	78.7	53.0	72.0	500,000

Table 8. MFB: Extended Training of Converging Topologies

Note: There is no significant improvement between 50,000 and 500,000 training cycles.

Using the weights save from the 20-50 network, neural network pruning techniques were employed see if any improvement could be made in the efficiency or accuracy.

Netsize Lower	Upper	Training Right	Guess	Test Right	Guess	Training Cycles
50	20	67.2	78.7	57.0	70.0	500,000
47	12	69.0	75.7	73.0	77.0	500,000
41	12	81.7	88.7	69.0	76.0	500,000
41	11	74.0	82.7	76.0	76.0	500,000
41	11	74.0	82.7	76.0	76.0	500,000
38	7	59.0	62.0	11.0	52.0	500,000

Table 9. Pruning to Find the Optimum Size Network

Note: Pruning the network found a better solution with an 11-41 configuration.

B.5 Counter Propagation

Because of the long training times and large net size require for convergence the Counter Propagation net was used to try and get a measure how many complex the decision regions are present. counterpropagation fail to classify with any reasonable numbers of nodes.

The experiment was halted after training 30 samples on to 20 kohonen nodes. The intent of the experiment is to find a less complex system than the multilayer

Samples	Kohonen Nodes	Training Right	Test Right	Guess
8	12	100.0	27.0	48.2
12	12	100.0	27.0	48.2
20	12	89.2	17.0	55.6
20	16	100.0	30.7	71.0
30	20	100.0	29.9	50.0

Table 10. Counter Propagation: Kohonen Node vs Samples

Note: When any more than 30 exemplars were used, 100 percent accuracy couldn't be reached.

perceptron alone. With the 20-40-20 net, the number of interconnection for only 30 samples is greater than the back propagation model.

B.4 Hybrid Propagation

As expected the Hybrid net trained faster than the back propagation network. the net size was greater than expected.

which is most probable due to inefficiencies in the Kohonen layer. By reducing the Kohonen layer to partition fewer decision regions, the back propagation layer can be reduced as well.

Samples	Net Size Kohonen	First	Second	Training Right	Test Right	Guess
100	12	20	12	93.0	61.0	72.0
200	16	24	12	81.0	53.0	66.0
300	20	20	12	92.0	56.0	72.0
652	60	60	24	91.0	63.0	74.0

Table 11. Hybrid Propagation: Kohonen Nodes vs Samples

B.5 Target Identification Data

The target identification data is a subset of the previous data. Those blobs identified as target were subjection to addition analysis. Eighteen statistical moments were calculated for each target. The targets could be either tanks, trucks or jeeps. Using an approach similar to that used for the target/non-target data. Differing net sized gave the results shown in Table 12

Netsize Lower	Upper	Training Right	Guess	Test Right	Guess	Converging
5	5	66.3	81.4	60.0	70.0	No
10	10	58.3	79.2	55.0	63.3	No
15	15	63.3	81.4	58.2	61.1	No
40	20	63.2	78.4	52.5	71.2	No

Table 12. Back Propagation: Accuracy vs Net Size

Using the Hybrid Propagation Network Table 13 shows the results.

Netsize Kohonen	Lower	Upper	Training Right	Guess	Test Right	Guess	Cycles
10	10	10	3.7	52.1	2.1	60.3	25,000
20	20	10	45.1	76.5	21.6	46.7	25,000
30	20	20	65.2	100.0	55.2	74.1	25,000

Table 13. Hybrid Propagation: Accuracy vs Net Size

Appendix C. *Computer Source Code*

This section include the computer software for several programs.

```
y=ra  .l() % 1 '0;
x=((float)y/100.0-0.5);
return x;
}
```

```
/*************************************************************************
*
*       DATE:  3 Oct 1988
*       VERSION:  1
*
*       NAME:  Makeinput.c
*       MODULE NUMBER: 1.4
*       DESCRIPTION: Provide for file input of a data set.
*       ALGORITHM: None
*       PASSED VARIABLES: None                                          10
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: input data
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: main()
*       CALLING MODULES: None
*                                                                       20
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/

#include "definitions.h"

#include "net.h"
#include <stdio.h>
#include <math.h>                                                       30

int examplars ;
int vector;
int exam_test;
int classes;

#define TRUE 1
#define FALSE 0

#define maxexamplars 800                                                40
#define maxexam_test 200
#define maxvector 40

float data[maxexamplars+maxexam_test][maxvector];
float data_test[maxexam_test][maxvector];

int class[maxexamplars];
int class_test[maxexam_test];

float add_noise=0.0;                                                    50
float data_stats[maxexamplars][maxvector];

extern float random();
```

```
char filename[30];

 static char sinwave[7] = {'s','i','n','w','a','v','e'};

int do_sin = TRUE;

FILE *fst;                                                          60
init_data()                                                  init_dat.
{

 int i,j,x;

 float y,z;

 FILE *fp,*fc;

 char target[4];                                                    70

          printf("\nData Filename:\n ");

          scanf("%s",filename);

loopi(7){if (sinwave[i] != filename[i]) {do_sin = FALSE;break; }}

if(do_sin == TRUE) {init_sin();
                    return;}
                                                                    80
  fst=fopen("data_stats","w");
  fp=fopen(filename,"r");

  if(fp==NULL){ printf("\nnnnn File not Found nnnn\n");exit(0);}

  printf("\nData Installed\n");

        fscanf(fp,"%d %d %d %d",&examplars,&exam_test,&vector,&classes);

        output = classes;                                           90

        input = vector;

        printf("%d %d %d %d\n",examplars,exam_test,input,output);

        for (i=0;i<examplars+exam_test;i++){

                fscanf(fp,"%d ",&x);

                for (j=0;j<vector;j++){                             100

                        fscanf(fp,"%g ",&data[i][j]);

                }

                        fscanf(fp,"%s",target);
```

```
                    if (target[0] == 'T') class[i]=0;
                    if (target[1] == 'A') class[i]=1;
                    if (target[0] == 'S') class[i]=2;

                    if (target[0] == '0') class[i]=0;
                    if (target[0] == '1') class[i]=0;
                    if (target[0] == '2') class[i]=1;
                    if (target[0] == '3') class[i]=2;
                    if (target[0] == '4') class[i]=3;
                    if (target[0] == '5') class[i]=4;
                    if (target[0] == '6') class[i]=5;
                    }

        fclose(fp);

        for (i=0;i<exam_test;i++){

            for (j=0;j<vector;j++){

                    data_test[i][j]= data[i+examplars][j];}

            class_test[i]=class[i+examplars];}

/*calculate_error();
*/
}

int sample,ctype;

MAKE_INPUT(x,doft,randx)                                    MAKE_INPUT

float x[];
float doft[];
int randx;

{ int i;

if(do_sin == TRUE) {MAKE_SIN(x,doft);return;}

if(randx < 0 ){

      ctype = rand() % output ;

      sample = rand() % examplars;}

else {sample = randx;
      ctype = class[sample];}


   while(class[sample] != ctype)
   sample= rand() % examplars;
```

```
/*    printf("%d %d \n",class[sample].ctype) ; */
      for(i=0;i<vector;i++)x[i]= data[sample][i];
      for(i=0;i<vector;i++) x[i] += x[i] * random()* add_noise;
      for(i=0;i<output;i++)
                  if(class[sample]==i) doit[i]=TRUE;
                  else doit[i]=FALSE;
   }
MAKE_TEST(x,doit,which)                                          MAKE_TEST
float x[],doit[];
int which;
{  int i;
   sample = which;
   ctype = class_test[sample];
   for(i=0;i<vector;i++)x[i]=data_test[which][i];
   for(i=0;i<output;i++)
       if(class_test[which]==i) doit[i]=TRUE;
   else doit[i]=FALSE;
}
#define allexam (examplars+exam_test)
float mean[size][maxexamplars+maxexam_test],
                stddev[size][maxexamplars+maxexam_test];
data_stats_function()                                            data_stats_function
{  int nn,i,j,k,l,small,right,wrong,totright;
   int class_accums[size];
   float dist[maxexamplars],temp;
   FILE *fnig;
   textport(0,1000,0,700);
   fnig = fopen("BearHeigh","w");
```

```
    system("clear");
    printf("File: %s \n",filename);
    fprintf(fnig,"File: %s \n",filename);
    printf("Number of Each Class: \n\n");
/*  Calculate Mean of each vector  */
    loopi(output) class_accums[i]=0;
    loopi(allexam) class_accums[class[i]]++;
    loopi(output){ printf("Class %d: %d    ",i,class_accums[i]);
                   fprintf(fnig,"Class %d: %d    ",i,class_accums[i]);}
    line:fprintf(fnig,"\n");

    loopj(input)
            {loopk(output)
                    mean[k][j] = 0.0;
                    loopi(allexam){
                    mean[class[i]][j] = mean[class[i]][j]+data[i][j];}

              loopk(output){
                        mean[k][j]=mean[k][j]/class_accums[k]; }

        }

#if FALSE
/*  Now calculate standard deviation   */
    loopj(input){loopk(output) stddev[k][j] = 0.0;
        loopi(allexam){
            printf("%d %d %d \n",j,k,i);
                stddev[class[i]][j] +=
                                mean[class[i]][j]-data[i][j]*
                                mean[class[i]][j]-data[i][j];}
              loopk(output) {stddev[k][j]=stddev[k][j]/class_accums[k];
        }
```

```
        } line;

#endif
/*  Using the mean for each class. calculate distance from
    each class     */                                                    270

right = 0;

loopi(allexam){loopk(output) dist[k] = 0.0;

                    loopj(input)

                        { loopk(output){ temp = data[i][j]-mean[k][j];

                                dist[k] += temp * temp ;}}                 280
                small=0 ;

                temp=dist[0];

                /* printf("Smp:%d ".i); */

                loopk(output) { /*  printf("%3.3f ".dist[k]); */

                                if (temp > dist[k])
                                                                           290
                                    { small = k;

                                      temp=dist[k];}
                              }

              if (class[small] == class[i]) right++;

            /* printf( " Class %d Nearest Class:%d ".class[i],small);line:
                */
                                                                           300
              } printf("Estimate using class means\n");

                fprintf(fnig,"Estimate using means\n");

                printf(" %d right/%d \n".right,allexam);

                fprintf(fnig," %d right/%d \n".right,allexam);

for (nn= 1;nn<10;nn++){
                                                                           310
    printf("\n %d nearest Neighbors  ".nn);

    fprintf(fnig,"\n %d nearest Neighbors   ".nn);

    totright = 0;

    loopi(examplars)
            {loopk(examplars) dist[k] = 0.0;
```

```
                loopj(input)
                                                                        320
                  { loopk(examplars){ temp = data[i][j]-data[k][j];
                                      dist[k] += temp * temp ;}
                  }
                 small=0 ;

                 temp=dist[0];

/*                 printf("Sample:%d class:%d neighbors ",i,class[i]);
                   fprintf(fnig, Sample:%d class:%d neighbors ",i,class[i]);
*/                                                                      330
          right = 0;wrong = 0;

          for(l= 0;l<nn;l++){

                 small=0 ;

                 temp=dist[0];

                 loopk(examplars) {
                                                                        340
                                  if ((temp > dist[k]) && (i != k))
                                       { small = k;
                                         temp=dist[k];
                                         }
                                  }
                        dist[small]=1000;

               /*         printf( "%3d ",class[small]);
                          fprintf(fnig, "%3d ",class[small]);
                */                                                      350
       if (class[small] == class[i]) right++;
                        else wrong++;
                             }
                       if(right>wrong) totright++;
                 }
printf("Total right %d %3.2f",totright,(float)totright/(float)examplars);

fprintf(fnig,"Total right %d %3.2f %"
                ,totright,(float)totright/(float)examplars*100.);
                                                                        360
}

fclose(fnig);
exit(0);
}

normalize_data()                                          normalize_data

{ int i,j=0;
  float mag;                                                            370
loopj(allexam){
```

```
        mag = 0.0;
        loopi(input) mag += data[j][i] * data[j][i];
                     mag = sqrt(mag);
        loopi(input) data[j][i]= data[j][i]/mag;
}printf("Data Normalized\n");

}
#define PI 3.1415

                                                                                    390
MAKE_SIN(x,doft)                                                                MAKE_SIN
 float x[];
 float doft[];
{ float y,z,add1,phase;
  long now;
  int i,j,k,which,which2,which3,which4;
   srand(time(&now) % 37 );
   which = rand() % input;
   phase = random() * PI;
   loopi(input){ x[i]= cos(((float)i*PI*(float)which)/input+ phase);}                 390
   loopi(output) doft[i] = 0.0;doft[which]=1.0;

   phase = random() * PI;
   which2 = rand() % input;
   loopi(input) x[i] = x[i]+ cos(((float)i*PI*(float)which2)/input+phase);
   doft[which2] = 1.0;

   phase = random() * PI;
   which3 = rand() % input;
   loopi(input)  x[i] = x[i] + sin(((float)i*PI*(float)which3)/iuput+phase);          400
   doft[which3] = 1.0;

       }
init_sin()                                                                      init_sin
{
printf("How many Input and Output Nodes ?");
scanf("%d",&output);
input = output;
examplars = 0;
exam_test = 0;                                                                      410
}
```

```
/*****************************************************************************
*                                                                            *
*       DATE:  29 Sept 1988
*       VERSION: 1
*
*       NAME: Feedforward
*       MODULE NUMBER: 1.5
*       DESCRIPTION: Propagates data from the input to the output
*       ALGORITHM: Standard feedforward rules                                 10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                                 20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/

#include "definitions.h"

#include "net.h"
                                                                              30
#include <math.h>

extern struct neural_net net;

float calcy();

FEED_FORWARD()                                          FEED_FORWARD

{ int i,j;
                                                                              40
   loopi(input) if (net.inp_mask[i] == 0.0) net.inp[i]=0.0;

   loopi(hide_one)
                  {net.y1[i]=   calcy(net.inp,net.w0,net.t0,&i,input);
                   }

   loopi(hide_one) if (net.y1_mask[i] == 0.0) net.y1[i]=0.0;

   loopi(hide_two)
               {net.y2[i]=   calcy(net.y1,net.w1,net.t1,&i,hide_one);}        50

   loopi(hide_two) if (net.y2_mask[i] == 0.0) net.y2[i]=0.0;
```

```
    loopi(output)
                {net.outp[i]=calcy(net.y2,net.w2,net.t2,&i,hide_two);}

    loopij(hide_one,input)

        { if(net.y2_mask[j]==1.0) net.aw0[i][j] = net.w0[i][j] * net.inp[j];}

    loopij(hide_two,hide_one)

        { if(net.y1_mask[j]==1.0) net.aw1[i][j] = net.w1[i][j] * net.y1[j];}

    loopij(output,hide_two)

        { if(net.inp_mask[j]==1.0) net.aw2[i][j] = net.w2[i][j] * net.y2[j];}

}

float fixy();

float calcy(x,w,theta,index,lower)

 float x[],w[][size],theta[];

 int *index;

 int lower;

 { int i,k,number;

   float y;

   y = 0.0;

   number = *index;

   loopi(lower){ y = y + x[i] * w[number][i];}

   y = y - theta[number];

   return fixy(y,2.0);}

float fixy(y,hardness)

 float y,hardness;

{
  return( 1.0/(1.0 + (float)exp(-(double)(hardness * y))));
}
findnode(xs,ys)

int xs,ys;
```

findnode

```
{int i,layer,the_node,temp;
        layer = (ys/spcing);                                          110
      the_node = (int)(xz / weight_s);
      temp = (1024/weight_s);
      clear_screen();
      switch(layer) {
        case 0: the_node -= (temp-input)/2;                           120
                if(net.inp_mask[the_node] == 0.0)
                 { net.inp_mask[the_node] = 1.0;
                   loopi(hide_one) net.t0[i] += net.aw0[i][the_node]; }
                else
                 {  net.inp_mask[the_node] = 0.0;                     130
                    loopi(hide_one) net.t0[i] -= net.aw0[i][the_node]; }
                break;
        case 1: the_node -= (temp-hide_one)/2;
                if(net.y1_mask[the_node] == 0.0)
                   {net.y1_mask[the_node] = 1.0;                      140
                   loopi(hide_two) net.t1[i] += net.aw1[i][the_node]; }
                else
                {   net.y1_mask[the_node] = 0.0;
                   loopi(hide_two) net.t1[i] -= net.aw1[i][the_node]; }
                break;                                                150
        case 2: the_node -= (temp-hide_two)/2;
                if(net.y2_mask[the_node] == 0.0)
                 {  net.y2_mask[the_node] = 1.0;
                    }
                else
```

```
    {   net.y2_mask[the_node] = 0.0;                                    160
        }
    break;
        }

}
```

```
/*****************************************************************************
*
*       DATE: 28 July 1988
*       VERSION: 1
*
*       NAME: Trainnet
*       MODULE NUMBER: 1.6
*       DESCRIPTION: Using output, adjusts wights reduce error.
*       ALGORITHM: Werbos Multilayer Perceptron Backpropagation.
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Weight Vectors
*       GLOBAL VARIABLES CHANGED: Weight Vectors
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: Main Loop
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/


#define THETA TRUE

#include "definitions.h"

#include "net.h"

extern struct neural_net net;

#define neta 0.3

float delx();

float dely();

#define momentum  0.7
TRAIN_NET()                                                    TRAIN_NET

{int i,j,k,n;
float del3[size],del2[size],del1[size];

 /*  output */

    loopj(output) { del3[j] = dely(net.outp[j],net.doft[j]);

      loopi(hide_two){
            net.w2_mom[j][i] = (neta * del3[j] * net.y2[i]+
```

```
    {   net.y2_mask[the_node] = 0.0;
        }
    break;
        }

}
```

```
            momentum * net.w2_mom[j][i])
            * net.y2_mask[i];
            net.w2[j][i] += net.w2_mom[j][i];

          }}

/* Second Hidden */

    loopj(hide_two)

          {  del2[j]= delx(net.y2[j],del3,net.w2,j,output);

             /* net.t1[j] += neta * del2[j] ;*/

              loopi(hide_one) {
              net.w1_mom[j][i] = (neta * del2[j] * net.y1[i]
              + momentum * net.w1_mom[j][i])
              * net.y1_mask[i];
              net.w1[j][i] += net.w1_mom[j][i];

             }}

/* First Hidden */

    loopj(hide_one) { del1[j]=delx(net.y1[j],del2,net.w1,j,hide_two);

          /* net.t0[j] += neta * del1[j] ; */

          loopi(input)      {

           net.w0_mom[j][i] =
             ( neta * del1[j] * net.inp[i] + momentum * net.w0_mom[j][i])
                * net.inp_mask[i];
           net.w0[j][i] += net.w0_mom[j][i];

                  }   }

}

float dely(y,doft)

 float y;

float doft;

 { float del=0.0;;

   del = y*(1.0-y)*(doft-y);

   return del;

   }
```

```
float delx(x,del,w,n,upper)
   float x,del[],w[][size];                                          110
   int n,upper;
 { float delta,sum;
   int i,j;
   sum = 0.0;
   loopi(upper) sum = sum + del[i] * w[i][n];                        120
   sum = x*(1.0-x) * sum;
   return sum;

 }
```

```
/*****************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME: Show.c
*       MODULE NUMBER: 1.7
*       DESCRIPTION: Used to display internal values in the textport
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                              20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/

#include "net.h"

#include <stdio.h>
                                                                           30
#include "definitions.h"
#define TRUE 1
#define FALSE 0
extern int display.do_sin;

extern struct neural_net net;
extern float accuracy;
extern int count,right,guess,good,test,decisions;

int l0 = 0;                                                                40
int l1 = 0;
int l1w = 0;
int l2 = 0;
int l3 = 0;
int l4 = 1;

SHOW()                                                                  SHOW
{   if(l0) showinput(net.inp) ;
    if(l1) shownode(net.y1.hide_one);
    if(l1w) showweights(net.w0.net.t0.1.hide_one.input);                   50
    if(l2) shownode(net.y2.hide_two);
    if(l4) showoutput(net.outp.net.doft);
}
```

showinput

```
showinput(x)
 float x[];
 {   int j,i;
     printf("Multilayer Perceptron Model\n");
     loopi(input) printf(" I(%d) ",i);line;
     loopi(input) printf("%1.2f  ",x[i]);line;line;
}
```

showoutput

```
showoutput(y,doft)
   float y[];
   float doft[];
   { int i,j;
    float error;
    printf("Out :");
    loopi(output) printf("%2.2f ",y[i]);line;
    printf("DofT:");
    loopi(output) printf("%2.2f ",doft[i]); line;
    printf("Count:%d \nRight:%2.2f Guess:%2.2f\n",count,
                              (float)right/(float)display*100.0,
                              (float)good/(float)display*100.0);
if(do_sin != TRUE)
    loopi(output) printf("Class %d %2.2f \n",i,
            (float)net.class_test[i]/
            (float)net.class_count[i]*100.0);
    line;
accuracy = (float)decisions/(float)display/(float)output * 100.;
printf("Decisions: %3.2f \n",(float)decisions/
                              (float)display/(float)output * 100.0);
clear_test();
   }
```

shownode

```
    float y[];
    int n;
    { int i,j;                                                    110
      float max,min;
      line;
      printf ("Node Values:");line;
     loopi(n)  printf("%2.4f  ",y[i]);line;
     }                                                            120
show:eights(w,theta,layer,upper,lower)                    showweights
 float w[][size],theta[];
 int layer,upper,lower;
 {   int j,i;
     float max,min;                                               130
    line;
    loopi(upper)  printf("W1(%d,%d) ",layer,i);line;
    loopi(lower) {
    loopj(upper) { if (w[j][i] < 0.0)
                   printf("%1.3f  ",w[j][i]);
           else                                                   140
                   printf(" %1.3f  ",w[j][i]);}
    line;}
    printf( "\nThetas: ");
    loopi(upper){ printf( " %3.6f ",theta[i]);}
    line;
}                                                                 150
```

```
/*************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME: Menu.c
*       MODULE NUMBER: 1.8
*       DESCRIPTION: Provides interactive menu functions
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
extern int dis_class,sample,exam_test;
extern float add_noise;
extern int activ,show_weights,display;
extern float threshold,add_noise;
                                                                         30
#include <signal.h>
#include <stdio.h>
#include "definitions.h"
#include "net.h"
extern int mistakes,stopit;
extern struct neural_net net;
extern int count,right,good,guess,test;
#define TRUE 1
#define FALSE 0
int menu()                                                               40
{  char select,filename[20];
   int nodenumber,selector=TRUE;


   system("clear");

   printf (" Menu \n");
   printf (" 1) Initialize System");line;
   printf (" 2) Save Weights");line;
   printf (" 3) Read Weights ");line;                                    50
   printf (" 4) Toggle Act/Weights").line;
   printf (" 5) Add  Noise");line;
   printf (" 6) Display Intervals");line;
```

```
    printf (" 7) Toggle Errors ");line;
    printf (" 8) New Net Topology");line;
    printf (" 9) Data Statistics");line;
    printf (" a) Set Stop");line;
    printf (" e) EXIT \n");
    printf ("          SELECTION: ? ");
    select = getchar();                                                    60
      switch(select) {
          case 'e':
                    gexit();
                    system("gclear");
                    exit(0);
                    break;
          case  '1': init_data();
                       :break;
          case  '2': write_weights ();
                                :break;                                    70
          case  '3': printf("\nEnter Filename: \n ? ");
                     scanf("%s",filename);
                     read_weights (filename);
                     printf("\n %s Installed\n\n",filename);
                      break;

          case '4': activ  = TRUE;
                    break;

                                                                           80
          case  '5': printf("How much noise: ?");
                     scanf("%f",&add_noise);
                     break;

          case  '6': printf("\nCount between Screen Update ?\n");
                     scanf("%d",&show_weights);
                     printf("\nCount between tests ?\n");
                     scanf("%d",&display);

                     break;                                                90
          case  '7': mistakes  = TRUE;
                     break;
          case  '8': system("clear");
                     init_net();
                     break;
          case  '9': init_data();
                     data_stats_function();
                     break;
}
                                                                          100
          signal(SIGINT,menu);}
new_net(){                                                          new_net
                     printf("New Network Topology\n");
                     printf("Number in First Layer\n");
                     scanf("%d",&hide_one);
                     printf("Number in Second Layer\n");
```

```
                    scanf("%d",&hide_two);
          };
read_weights (filename)                                          read_weights
 char filename[];                                                        110
{   FILE *fp;
    float x;
    int i,j,k;
      fp = fopen (filename,"r");
      if (fp==NULL)
      {printf("\n ***File Error*** \n");return;}
      fscanf (fp,"%d %d %d %d",&output,&hide_two,&hide_one,&input);

        loopij(output,hide_two){
            fscanf(fp,"%f",&x);                                          120
            net.w2[i][j]=x;}
        loopij(hide_two,hide_one){
            fscanf(fp,"%f",&x);
            net.w1[i][j]=x;}
        loopij(hide_one,input){
            fscanf(fp,"%f",&x);
            net.w0[i][j]=x;}

            loopi(output)    {fscanf(fp,"%f",net.t2+i);}
            loopi(hide_two) {fscanf(fp,"%f",net.t1+i);}                  130
            loopi(hide_one) {fscanf(fp,"%f",net.t0+i);}
            fscanf(fp,"%d",&count);
            loopi(hide_two) {fscanf(fp,"%f",net.y2_mask+i);}
            loopi(hide_one) {fscanf(fp,"%f",net.y1_mask+i);}
            loopi(input) {fscanf(fp,"%f",net.inp_mask+i);}
            fclose(fp);}

write_weights ()                                                 write_weights

  { FILE *fp;                                                            140
    int i,j,k;
     char filename[20];
     printf("\nEnter filename: ");
     scanf("%s",filename);
     fp = fopen (filename,"w");
    fprintf (fp,"%d %d %d %d \n",output,hide_two,hide_one,input);
    loopij(output,hide_two)
          fprintf(fp,"%f ",net.w2[i][j],i,j);
    loopij(hide_two,hide_one)
          fprintf(fp,"%f ",net.w1[i][j],i,j);                            150
    loopij(hide_one,input)
          fprintf(fp,"%f ",net.w0[i][j],i,j);
          loopi(output)    {fprintf(fp,"%f \n",net.t2[i]);}
          loopi(hide_two) {fprintf(fp,"%f \n",net.t1[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.t0[i]);}
          fprintf(fp,"%d \n",count);
          loopi(hide_two) {fprintf(fp,"%f \n",net.y2_mask[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.y1_mask[i]);}
          loopi(input) {fprintf(fp,"%f \n",net.inp_mask[i]);}
```

```
        fclose(fp);printf("\n Weights Stored\n");
        }

write_std_weights ()                                                    write_std_weights

  { FILE *fp;
    int i,j,k;
     char filename[20];
     fp = fopen ("standard.wei","w");

    fprintf (fp,"%d %d %d %d \n",output,hide_two,hide_one,input);
    loopij(output,hide_two)
          fprintf(fp,"%f ",net.w2[i][j],i,j);
    loopij(hide_two,hide_one)
          fprintf(fp,"%f ",net.w1[i][j],i,j);
    loopij(hide_one,input)
          fprintf(fp,"%f ",net.w0[i][j],i,j);
          loopi(output)    {fprintf(fp,"%f \n",net.t2[i]);}
          loopi(hide_two) {fprintf(fp,"%f \n",net.t1[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.t0[i]);}
          fprintf(fp,"%d \n",count);
          loopi(hide_two) {fprintf(fp,"%f \n",net.y2_mask[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.y1_mask[i]);}
          loopi(input) {fprintf(fp,"%f \n",net.inp_mask[i]);}
          fclose(fp);printf("\n Weights Stored\n");
          }

write_string(x,y,l,title)                                               write_string
int x,y,l;
char title[20];

{ char number[20];
  l = (int)((float)l/video);
  color(8);
  rectfi(x-5,y-5,x+l,y+15);
  color(4);
  linewidth(1);
  recti(x-5,y-5,x+l,y+15);
  cmov2i(x,y-2);
  charstr(title);
   }

write_float(x,y,l,title,ft,a_color)                                     write_float
int x,y,l;
char title[20];
float ft;
int a_color;

{ char number[20];
  l = (int)((float)l / video);
  sprintf(number,"%3.3f",ft);
  color(a_color);
```

```
    rectfi(x-5,y-5,x+1,y+15);
    color(4);
    linewidth(1);
    cmov2i(x-2,y);
    charstr(title);
    charstr(number); }

write_int(x,y,title,ft)
int x,y;
char title[20];
int ft;

{ char number[20];
  sprintf(number,"%d",ft);
  cmov2i(x,y);
  charstr(title);
  charstr(number); }
```

write_int

220

230

```
/*****************************************************************************
*
*       DATE:10 August 1988
*       VERSION:
*
*       NAME: DISPLAY_NET
*       MODULE NUMBER: 1.9
*       DESCRIPTION: Displays the Network using Graphics calls
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Net
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr                                            20
*       HISTORY:
*
*****************************************************************************/
#include "net.h"

#include "definitions.h"

#include "gl.h"

extern int count,right,good,test,do_sin;                                   30

extern struct neural_net net;

int activ = FALSE;

DISPLAY_NET()                                                  DISPLAY_NET


{
write_string(480,652,50,"Value");                                          40
write_string(580,652,50,"Guess");
write_string(680,652,50,"Right");

write_string(520,700,130,"Desired Output");

write_string(30,730,350,"Feedforward Backpropagation Network");

do_screen();
if (! activ) {
                                                                           50
    plotnode (0,net.inp,net.inp_mask,net.y1_mask,net.w0,input,hide_one);

    plotnode (1,net.y1 ,net.y1_mask,net.y2_mask,net.w1,hide_one,hide_two);
```

```
        plotnode (2,net.y2 ,net.y2_mask,net.outp_mask,net.w2,hide_two,output);
        write_string(100,220,120,"Weights");
    }
    else {
        plotnode (0,net.inp,net.inp_mask,net.y1_mask,net.sw0,input,hide_one);
        plotnode (1,net.y1 ,net.y1_mask,net.y2_mask,net.sw1,hide_one,hide_two);
        plotnode (2,net.y2 ,net.y2_mask,net.outp_mask,net.sw2,hide_two,output);
        write_string(100,220,120,"Activation");}
        plotnodei(3,net.outp,output);
        plotnoded(3.0,net.doft,output);
    if(do_sin == FALSE){
        plotnoded(3,100,net.doft,output);
        plotnoded(3,200,net.doft,output);}
        /*write_error(); */
        check_mouse();
}

float threshold= 0.0;

plotnodei(x,node,lower)                                          plotnodei
int x;
float node[];
int     r;
{ int i,j,k,y,x2,y2;

   y=spcing*x+30;
   x=(1024-weight_s*lower)/2;
   loopi(lower) {
        set_color(1.1,0.01,node[i]);
        big_plot(i*weight_s+x,y,weight_s/4*3);
```

```
if(do_sin == FALSE){
        color_of(0.5,0.5,node[i]);
                                                                    110
        big_plot(i*weight_s+x+100,y,weight_s/4*3);

        color_of(0.9,0.1,node[i]);

        big_plot(i*weight_s+x+200,y,weight_s/4*3);
}
}


                                                                    120
}

plotnoded(x,y2,node,lower)                                  plotnoded

int x,y2;

float node[];

int lower;
                                                                    130
{ int i,j,k,x2,y;

   y=spcing*x+30;

   x=(1024-weight_s*lower)/2;

    loopj(lower){

       loopi(lower) {
                                                                    140
         color_of(0.9,0.1,(float)node[i]);

         big_plot(i*weight_s+x+y2,
                  y+weight_s*2,weight_s/4*3);
}}}

plotnodex(x,y2,node,lower)                                  plotnodex

int x,y2;
                                                                    150
int node[];

int lower;

{ int i,j,k,x2,y;

   y=spcing*x+30;

   x=(1024-weight_s*lower)/2;
```

```
    loopj(lower)

    loopi(lower) {

            set_color(1.0,0.0,(float)node[i]);

            big_plot(i*weight_s+x+y2,
                     y+weight_s*2,weight_s/4*3);
}}


#if FALSE
plotnode(x,node,array,lower,upper)                                  plotnode
int x;
float node[],array[][size];
int lower,upper;
{ int i,j,k,y,x1,x2,y2;
  float max,min,temp,temp2;
   y=spcing*x+30;
   x1=(1024-weight_s*lower)/2;
   x2=(1024-weight_s*upper)/2;
  linewidth(2);

  findmax(array,&max,&min,upper,lower);
  loopi(lower) {

          color_of(0.9,0.1,node[i]);
          big_plot(i*weight_s+x1,y,weight_s/4*3);


  loopj(upper) { temp = array[i][j];
                  set_color(max,min,temp);
                  drawit(x1+weight_s*i,y+weight_s*3/4,
                    x2+weight_s*j,y+spcing);
                 }


                 }
     colorbar(1024-256,y+10,max,min);
}
#endif

plotnode(x,node,mask,mask_up,array,lower,upper)                     plotnode
int x;
float node[],mask[],mask_up[],array[][size];
int lower,upper;
{ int i,j,k,y,x1,x2,y2;
  float max,min,temp,temp2;
  cursoff();
   y=spcing*x+30;
   x1=(1024-weight_s*lower)/2;
```

```
    x2=(1024-weight_s*upper)/2;
    linewidth(2);
    findmax(array,&max,&min,upper,lower);
    loopi(lower) {
            if(mask[i]==1.){
                    color_of(0.9,0.1,node[i]);
                    big_plot(i*weight_s+x1,y,weight_s/4*3);

      loopj(upper) {
              if(mask_up[j] == 1.0){
                    set_color(max,min,array[i][j]);
                    drawit(x1+weight_s*i,y+weight_s*3/4,
                    x2+weight_s*j,y+spcing);}
                      }
                    }


                    }
      colorbar(1024-256,y+10 ,max,min);
      cursor();}


findmax(array,max,min,outs,ins)                                    findmax

float array[][size],*max,*min;
int outs,ins;
{ int i,j,k;
  int maxi=0,maxj=0;
  int mini=0,minj=0;
 *min = array[0][0];
 *max = array[0][0];
loopi(ins){
  loopj(outs){     if (array[j][i]< *min) *min=array[j][i];

                   if (array[j][i]> *max) *max=array[j][i];
              }
             }}


display_count()                                                    display_count
 {  int where = 650;
     linewidth(1);
     color(7);
     rectfi(800,where,923,where+20);
     color(500);
     recti(800,where,923,where+20);
     color(1);
     write_int(805,where+2,"",count);}
```

```
/*****************************************************************************
*
*       DATE: 9 Oct 1988
*       VERSION:
*
*       NAME: Graphic Package
*       MODULE NUMBER: 1.10
*       DESCRIPTION: Graphic Routine for the Silicon Graphic IRIS 3130
*       ALGORITHM: None                                                      10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                                20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
#include"gl.h"

#include"device.h"

#include "definitions.h"                                                     30
extern float history[];
extern int count,display,do_sin;
int fast = 0;
check_mouse()                                                    check_mouse

{ short val;

  float temp,xtemp,ytemp;

  Screencoord xs,ys;                                                         40
  int whichx,whichy;
  char message[20];

  if (qtest()!=0){

                        xs=getvaluator(MOUSEX);
                        ys=getvaluator(MOUSEY);

                        xs = (int)((float)xs / video);
                        ys = (int)((float)ys / video);                       50
                       whichx = (xs-30)/100;
                       whichy = xs/50;
```

```
switch (qread(&val)){
        case MIDDLEMOUSE: if(val==1){
                                                                        60
                          blowup(xs,ys);}
                          else {
                          restore_screen();
                          clear_screen();}
                          break;
        case LEFTMOUSE: if(val==1){                                     70
                          findnode(xs,ys);
                          break;  }

        case RIGHTMOUSE: if(val==1){

          if (ys > 695){                                                80
             switch(whichx){ .
                   case 4:fast  = 1:break;
                   case 3:exit(0);break;
                   case 2:menu();break;
                   case 1:make_graph():break;
                   case 0:while(val==1) qread(&val);
                          while(val!=1) qread(&val);
                          break;
                            }
                        }                                               90
                          break;}
                    }

DISPLAY_NET();
  }
}


blowup(xs,ys)                                                           blowup
                                                                        101
Screencoord xs,ys;

{Screencoord xc,yc;
  cursoff();
```

```
                        xc=getvaluator(MOUSEX);
                        yc=getvaluator(MOUSEY);

                                                                        110
    viewport(xc-100,xc+100,yc-100,yc+100);

    linewidth(2);color(512);

    recti(xs-95,ys-95,xs+95,ys+95);

    color(0);clear();

    ortho2((Coord)xs-20.,(Coord)xs+20.,(Coord)ys-20.,(Coord)ys+20.);
                                                                        120

}


restore_screen()                                                restore_screen
{
  viewport(0,(int)(1023. * video), 0 ,(int)(767.0 * video));
  ortho2(0.,1023.,0.,767.);
}
                                                                        130
color_of(hi,lo,value)                                           color_of

float hi,lo,value;

{ color(INDETERM);

  if(value > hi) color(HARD_ON);

  if(value < lo) color(HARD_OFF);
  }                                                                     140
/* 600.900,470.620 */

make_graph(x,y)                                                 make_graph
 float x,y;
{ int i,index;
  char numb[5];
  cursoff();
  if (count < display) return;

  viewport((int)(x * video),(int)((x + 300.) * video),                  150
           (int)(y * video),(int)((y+150.0) * video));

if(count < 10000)
  ortho2(0.,(float)count, -0.5,history[0]);
else
  ortho2((float)count - 10000.,(float)count, -0.5,history[0]);
```

```
  color(1000);
  clear();
  color(1001);
  index = count/display;

  loopi(index){ /* horizontal */
  move2((float)(i*display),-0.5);
  draw2((float)(i*display),history[0]);}

   if(do_sin != TRUE)

{    index = (int)history[0];
     loopi( (index) ){ /* verticle */
     move2(-0.5,(float)i);
     draw2((float)count,(float)i);} }

  color(YELLOW);

  index = count/display;
  move2(0,history[0]);
  loopi(index+1)
  draw2((float)(i*display),history[i]);

  restore_screen();

  write_float((int)x,(int)y+130,50,"",history[0],1000);
  write_float((int)x,(int)y+30,50,"",history[index],1000);
  write_string2((int)x +50,(int)y,130,"Error History");
cursor();

}
```

colorbar

```
colorbar(x,y,max,min)
int x,y;

float max,min;

{ int i;

   char maxstring[20],minstring[20];

   x -= 80;

   for (i=8;i<TABLE;i++,i++){

   color(i);

   big_plot(x+i/2*(512/TABLE)-20,y,18,(512/TABLE));}

   sprintf(maxstring,"%3.3f",max);
```

```
    sprintf(minstring,"%3.2f",min);
    color(YELLOW);
    cmov2i(x+20,y);
    charstr(minstring);
    cmov2i(x+160,y);                                                    220
    charstr(maxstring);
}

set_color(max,min,value)                                                set_color
float max,min,value;
{ float percent;
                                                                        230
    int colx;
    if(value > max) value = max;
    if(value < min) value = min;
    percent = (value-min)/(max-min) * (float)(TABLE-9);
    colx = (int)percent + 8;
                                                                        240
    color(colx);
}

#if FALSE

big_plot(x,y,dot)                                                       big_plot
int x,y,dot;
{
/*  rectf((float)(x-dot/2),(float)(y),                                  250
        (float)(x+dot/2),(float)(y+dot));
  drawit(x,y,x,y+size_of_dot);
*/
  }
#endif

big_plot(x,y,ww,hh)                                                     big_plot
                                                                        260
int x,y,ww,hh;
{
```

```
rectf((float)(x-ww/2),(float)(y),
        (float)(x+ww/2),(float)(y+ww));
 }

drawit(xstart,ystart,xend,yend)

int xstart,ystart,xend,yend;
{
        move2((float)xstart,(float)ystart);
        draw2((float)xend,(float)yend);
}

drawiti(xstart,ystart,xend,yend)
int xstart,ystart,xend,yend;

{
        move2i((float)xstart,(float)ystart);
        draw2i((float)xend,(float)yend);
}
drawit2(xstart,ystart,xend,yend)

float xstart,ystart,xend,yend;

{
        move2(xstart,ystart);
        draw2(xend,yend);
}

clear_screen()
{
  cursoff();
  color(0);
  clear();
  textpt1();
  curson();}

  textpt1() /* Used for menu */
 {
    textport(20,(int)(480.*video),(int)(400.*video),(int)(700.*video));
    pagecolor(1000);
    textcolor(CYAN);
    system("clear");}
```

drawit

drawiti

drawit2

clear_screen

textpt1

init_dav

```
init_dav()
{
    gbegin();
    restore_screen() ;
    gconfig();
    textpt1();
}
```

init_screen

```
init_screen()

{ int i,j,k,red,green,blue;

    gbegin();

    restore_screen() ;

    gconfig();

    mapcolor(1000,100,100,100);
    mapcolor(1001,50,50,50);

    textpt1();

    clear_screen();

/*
    cmov2i(430,650);

    charstr("Actual");

    cmov2i(530,650);

    charstr("Guess");

    cmov2i(630,650);

    charstr("Right");
*/
    qdevice(MIDDLEMOUSE);
    qdevice(LEFTMOUSE);
    qdevice(RIGHTMOUSE);

    for(i=TABLE,k=0,j=8;j<.TABLE;j++){
        if (j<(TABLE/2))
                    {green=(int)((256./TABLE.) * (float)j)*2;
                     blue = 256-green ;
                     mapcolor(j,0,green/3,blue);}
              else {
                    red =(int)((256./TABLE.) * (float)(j-TABLE/2))*2;
                    green=(256-blue);
                    mapcolor(j,red,green/3.0);}

}
do_screen();
```

```
}
do_screen()
{
write_string2(30,700,40,"HALT");
write_string2(130,700,50,"GRAPH");
write_string2(230,700,40,"MENU");
write_string2(330,700,40,"Quit");
write_string2(430,700,40,"Fast");}
```

do_screen
371

```
write_string2(x,y,l,title)
int x,y,l;
char title[20];

{ char number[20];
  l = (int)((float)l/video);

  color(5);
  rectfi(x-5,y-5,x+l,y+15);
  color(1);
  linewidth(1);
  recti(x-5,y-5,x+l,y+15);
  cmov2i(x,y-2);
  charstr(title);
   }
```

write_string2
380

390

```
/*****************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME:test.c
*       MODULE NUMBER:1.11
*       DESCRIPTION:Provides for system test and evaluation during training
*       ALGORITHM: None                                                      10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                                20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
extern int count,right,good,test,exam_test,examplars,sample,ctype;
extern int display;
extern struct neural_net net;
int decisions;
float accuracy;                                                              30
#include "definitions.h"
#include "net.h"
#include <math.h>
int mistakes = 0;
float history[500];
     float error[size];
extern int output,input,hide_one,hide_zero;
CHECK_ERRORS(y,doft)                                          CHECK_ERRORS
   float y[];
   float doft[];                                                             40
   { int i,j;
     int correct_right=0;
     int correct_good=0;
      ++net.class_count[ctype];
      loopi(output) { error[i] = (float)doft[i]- y[i];
                      if (error[i] < 0)    error[i] = -error[i];

                      if (error[i] < 0.5) {correct_good++;
                                           decisions++;}
                      if (error[i] < 0.2) correct_right++;                   50
                      }
               if (correct_good  == (output)){ good++;
                             ++net.class_test[ctype]; }
```

```
                else
                 if (mistakes) printf("Sample %2d Type %d\n",sample,ctype);

               if (correct_right == (output)) right++;
   }
float err,olderr=0.0,derverr=0.0;                                          60
                                                                          CHECK
CHECK(y,doft)
   float y[];
   float doft[];
   { float missed_by=0;
    int i,j;
    loopi(output)  {
                        missed_by = (float)doft[i]- y[i];
                         err += missed_by * missed_by;
                  /*       printf("doft %3.1f y %3.1f "
                                 ,doft[i],y[i]);    */                     70

   }
missed_by = 0.;          }

                                                                          clear_test
clear_test()
{int i;
     loopi(output) {net.class_test[i]=0;
                    net.class_count[i] = 0;}
                right = 0;good = 0;decisious = 0;
   }
                                                                          DO_TEST
DO_TEST()                                                                  81
{  int i;
   clear_test();
   loopi(exam_test){
      MAKE_TEST(net.inp,net.doft,i);
      FEED_FORWARD();
      CHECK_ERRORS(net.outp,net.doft);
   }
   printf("Test: %3.2f          %3.2f \n",(float)right/(float)exam_test*100.,
                   (float)good/(float)exam_test*100.);
   clear_test();                                                           90

}

                                                                          calculate_error
calculate_error()
{    int i,j,index;
     err= 0.;

    clear_test();                                                         100
     loopi(examplars){
       MAKE_INPUT(net.inp,net.doft,i);
       FEED_FORWARD();
       CHECK(net.outp,net.doft);
    }
   err = (float)sqrt((double)err);
```

```
  derverr = (err - olderr);
  olderr = err;

if(examplars == 0)  err = 100. - accuracy;
      index=count/display;                                          110
       history[index] = err;
  err = 0;
}


CHECK2(y,doft)                                                      CHECK2
   float y[];
   float doft[];
   { float missed_by=0;
    int i,j;                                                        120
    loopi(output)  {
                        missed_by = (float)doft[i]- y[i];
                         err += missed_by * missed_by;
   }

missed_by = 0.;          }
```

```
/*****************************************************************************
*
*       DATE: 17 Aug 1988
*       VERSION: 1.0
*       NAME: Counter Propagation Neural Net--Main Loop
*       MODULE NUMBER: 2.0
*       DESCRIPTION: Counterprop main loop
*       ALGORITHM: None
*       PASSED VARIABLES: None                                          10
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:                                                        20
*
*****************************************************************************/

#include "net.h"

#include "definitions.h"

#define wait(A) if((count%(A))==0)

struct neural_net net;                                                  30

int input,output,hide_one=30;

extern sample;

int display = 500;

extern int menu();

int count=0,right,good,guess,test;                                      40

main()                                                                  main
{

   INITIALIZE();

   normalize_data();

   while (count < 100000) {
                                                                        50
      /* with counter propagation */

       MAKE_INPUT(net.inp,net.yout,-1);
```

```
        FEED_FORWARD(TRAIN);
        TRAIN_NET();

        /* remove counter propagation to test */

        MAKE_INPUT(net.inp,net.yout,-1);
        FEED_FORWARD(CHECKIT);                                          60
        CHECK_ERRORS(net.yp,net.yout);
        wait(display){ SHOW();
                       DO_TEST();
                       line; }


        wait(display) DISPLAY_NET();

        count++;
        display_count();                                                70

}}
```

```
/*****************************************************************************
*
*       DATE: 11 Aug 1988
*       VERSION:2.1
*
*       NAME: Defineitons.h
*       MODULE NUMBER:
*       DESCRIPTION:
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                              20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
#define GRAPHICS TRUE
#define video 0.60
#define loopi(ON) for(i=0;i<ON;i++)
#define loopj(ON) for(j=0;j<ON;j++)
#define loopk(ON) for(k=0;k<ON;k++)                                        30

#define loopij(ONE,TWO) for(i=0;i<ONE;i++)for(j=0;j<TWO;j++)

#define TABLE 512
#define HARD_ON  RED
#define INDETERM GREEN
#define HARD_OFF BLUE

#define TRUE 1
#define FALSE 0                                                            40

#define TRAIN 0
#define CHECKIT 1


#define line printf("\n")
#define disply 100
```

```
#define size 200

struct neural_net {


  float yout[size]; /* NODE Y output fanout  */
  float yp[size];       /*NODE Y'*/
  float yz[size][size]; /* WEIGHTS*/
  float zyp[size][size];

  float z[size];        /* NODE Z*/
  float z_con[size];
  float xp[size];     /* NODE X'  */
  float xz[size][size];
  float zxp[size][size];
  float inp[size];    /*NODE X*/


} ;
```

```
/*************************************************************************
*
*       DATE: 3 August 1988
*       VERSION:
*
*       NAME: Initialization Module
*       MODULE NUMBER: 2.3
*       DESCRIPTION: Initialization of the display hardware.
*                    any input test data, and net data structure.
*
*       ALGORITHM: None
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Net
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#include "definitions.h"

#include"gl.h"

#include"device.h"

#include <time.h>
#include <stdio.h>
#include <signal.h>
#include <math.h>
#include "net.h"

extern struct neural_net net;
extern int input,output,hide_one;

float random();

#define TABLE 512

extern int menu();

INITIALIZE()
{
/* signal allows the menu to be called by a control c */
```

INITIALIZE

```
    signal(SIGINT,menu);

    init_screen();
    init_data();
    init_net();
}

init_screenx()                                                    init_screenx

{ int i,j,k;

    winopen("net");
    ginit();
    gconfig();
    color(BLACK);
    clear();
    linewidth(1);

    color(YELLOW);
    cmov2i(40,600);
    charstr("Counter Propagation");

    cmov2i(40,550);
    charstr("Neural Network");

    mapcolor(GREEN,50,50,50);

    for(i=TABLE/2,k=0,j=8;j<TABLE;j++)
        { if (j<264) {i--; mapcolor(j,0,128-i/2,i);}
          if (j>264) {k++; mapcolor(j,k,128-k/2,0);}
}
}

init_net()                                                        init_net
{

    int i,j;
    loopij(output,hide_one)
          net.zyp[i][j] = 1.0;
          normalize_weights(net.zyp,output,hide_one);

    loopij(output,hide_one)
         net.yz[i][j]= random();
          normalize_weights(net.yz,output,hide_one);

    loopij(hide_one,input)
          net.xz[i][j] = 1.0;
          normalize_weights(net.xz,hide_one,input);

    loopij(hide_one,input)
          net.zxp[i][j] = 1.0;
    loopi(hide_one) net.z_con[i] = 0.0;
```

```
        normalize_weights(net.zxp,hide_one,input);

printf("Weights Normalized\n");
}

normalize_weights(weights,upper,lower)

float weights[];
int upper,lower;

{ int i,j=0;
  float mag;
  loopj(upper){
       mag = 0.0;
       loopi(lower) mag += weights[i+j*size] * weights[i+j*size];
                    mag = sqrt(mag);
       loopi(lower) weights[i+j*size]= weights[i+j*size]/mag;
              }
}


float random ()

{ float x;
  int y;
  y=(rand() % 1000) ;
  x=((float)y/1000.0--0.5);
  return x;

  }
findnode()
{}
big_plot2()
{}
```

normalize_weights

findnode

big_plot2

```
/*****************************************************************************
*       DATE:
*       VERSION:
*
*       NAME: Feedforward
*       MODULE NUMBER:
*       DESCRIPTION:
*       ALGORITHM: None
*       PASSED VARIABLES: None
*       RETURNS: None                                                    10
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr                                          20
*       HISTORY:
*
*****************************************************************************/
#include "definitions.h"
#include "net.h"
#include <math.h>

extern struct neural_net net;
extern int input,output,hide_one,count;
                                                                         30
int max_index;

FEED_FORWARD(tst)                                          FEED_FORWARD
  int tst;
{
  int i,j;
  float max,temp1,temp2,bi;

/*  calculate z's   */
                                                                         40
/*  Note that the feed forward algorithm
    works a little differently when actually testing
    the system as opposed to teaching.

    For teaching the yout value are counter propagated
    through the system.  For testing yout value are assumed
    to be zero.

    The test condition statement in the double loop then
   accounts for this difference.                                         50

*/
    loopi(hide_one) {temp1 = 0.0;temp2=0;
```

```
        loopj(input)   temp1 +=  net.inp[j] * net.xz[i][j];
        if(tst != TRUE ){
        loopj(output)  temp2 +=  net.yout[j] * net.yz[j][i];}
         net.z[i]= temp1 + temp2;
        }

/*  Now find max of z's  */

    max = 0;max_index = 0;
    loopi(hide_one) {  bi = net.z_con[i]/(float)count - 1.0/(float)hide_one;
                       if (tst == TRUE) bi = 0.0;
                       if (bi < 0.02) {
                       if (net.z[i] > max){
                       max_index=i;
                       max = net.z[i];} }}
   if(tst != TRUE)     net.z_con[max_index] += 1.0;


/* by using on a > symbol default is to the lowest index
   as recommended by Hecht_Nielsen                        */


    loopi(hide_one) { if (i==max_index)
                                    net.z[i]=1.0;

                                else
                                    net.z[i]=0.0;}

    loopi(output) {net.yp[i]=0.0;
       loopj(hide_one)
            { net.yp[i] += net.zyp[j][i] * net.z[j];}}

    loopi(input) {net.xp[i]= 0;
       loopj(hide_one)
           { net.xp[i] += net.zxp[i][j] * net.z[j];}}
                }
```

```
/*****************************************************************************
*
*       VERSION:
*       NAME: train net.c
*       MODULE NUMBER: 2.4
*       DESCRIPTION:
*       ALGORITHM:
*       PASSED VARIABLES:
*       RETURNS: None                                                    10
*       GLOBAL VARIABLES USED: Weight Vectors
*       GLOBAL VARIABLES CHANGED: Weight Vectors
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: Main Loop
*
*       AUTHOR: Gregory L. Tarr                                          20
*       HISTORY:
*
*****************************************************************************/


#define THETA FALSE

#include "definitions.h"
#include "net.h"
extern struct neural_net net;                                            30
extern int input,output,hide_one;
#define alpha 0.01
#define beta  0.01
extern int max_index;


TRAIN_NET()                                                  TRAIN_NET
{int i,j,k,n;

/*  adjust the weights between the y prime and the z layer  */           40
    loopi(output)
    net.zyp[max_index][i] +=
              (-alpha * net.zyp[max_index][i] + beta * net.yout[i]);

/* adjust the weights between the x prime and the z layer    */
       loopi(input)
          net.zxp[max_index][i] +=
                 (-alpha * net.zxp[max_index][i] + beta * net.inp[i]);

/* adjust weights between z and x input  */                              50
    loopi(input)
       net.xz[max_index][i] += alpha * (net.inp[i] -  net.xz[max_index][i]) ;
```

```
/* adjust weights between z and y prime  */

        loopi(hide_one)
         net.yz[max_index][i] += beta * (net.yout[i] -  net.yz[max_index][i]);

}
```

```
/*************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME: Show.c
*       MODULE NUMBER:  2.5
*       DESCRIPTION:
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#include "net.h"
#include <stdio.h>
#include "definitions.h"
extern int display;
extern struct neural_net net;                                            30
extern int input,output,hide_one,right,good;

float DO_CHECK();

extern int count,right,guess,good,test;
SHOW()                                                                   SHOW
{
    showoutput(net.yout,net.yp);
/*
    showweights(net.yz,1,output,hide_one);                               40
    shownode(net.z,hide_one);
    showweights(net.xz,1,hide_one,input);
    showinput(net.inp);
    shownode(net.y2,hide_two);
    nodeinfo(800,600);
*/}

showinput(x)                                                             showinput
 float x[];
 {   int j,i;                                                            50
    printf("Counter Propagation Model\n");
    loopi(input) printf(" X(%d) ",i);line;
```

```
    loopi(input) printf("%1.2f  ",x[i]);line;line;
}

showoutput(yout,yp)                                                    showoutput
   float yout[];
   float yp[];
   { int i,j;                                                                  60
     float error;
/*
printf("\nY-out      ");
    loopi(output) printf("%2.2f  ",yout[i]);line;

printf("Y-prime    ");
    loopi(output) printf("%2.2f  ",yp[i]); line;line;
*/

 printf("Count:%6d \nGuess:%1.2f Good: %1.2f Test: %1.2f\n",count,           70
                              (float)right/display*100.0,
                              (float)good/display*100.0,
                              0.0);
header();
/* fprintf(stderr,"Count:%6d  Guess:%3.2f   Good:  %3.2f    Test: %3.2f\n",count,
                              (float)right/display*100.0,
                              (float)good/display*100.0,
                              0.0);
*/   }
                                                                           80

shownode(y,n,title)                                                    shownode

   float y[];
   int n;
   char title[20];
   {. int i,j;line;

    printf ("%s ",title);line;
    loopi(n)  printf("%2.4f  ",y[i]);line;                                  90

    }

showweights(w,layer,upper,lower)                                       showweights

 float w[][size];
 int layer,upper,lower;
 {   int j,i;
    line;
    loopi(upper)  printf("Wi(%d,%d) ",layer,i);line;                         100

    loopi(lower) {
    loopj(upper) { if (w[j][i] < 0.0)
                     printf("%1.3f  ",w[j][i]);
            else
```

```
                printf(" %1.3f  ",w[j][i]);}
        line:}
```



```
        line: }
```

```
/*************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME:menu.c
*       MODULE NUMBER:2.6
*       DESCRIPTION: Provides interactive menu options.
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
extern int is_class;
extern int activ;
extern float threshold;

#include <signal.h>                                                      30
#include <stdio.h>
#include "definitions.h"
#include "net.h"
#include <gl.h>
extern struct neural_net net;
extern int count,right,good,guess,test,display;

int menu()
{  char select,filename[20];
   printf (" Menu \n");                                                  40
   printf (" 1) Save Weights");line;
   printf (" 2) Display Weights");line;
   printf (" 3) Read Weights ");line;
   printf (" 4) Change Act Display");line;
   printf (" 5) Show Activation Levels");line;
   printf (" 6) Show Weight Levels");line;
   printf (" 7) Change Display Threshold");line;
   printf (" e) EXIT \n");
   printf ("          SELECTION: ? ");
   select = getchar();                                                   50
     switch(select) {
         case 'e':
                  gexit();
```

```
                exit(1);
                break;
        case  '1': write_weights ();
                                ;break;
        case  '2': printf("\nWeights Displayed \n");
                   pause();                                          60
                   break;
        case  '3': printf("\nEnter Filename: ");
                   scanf("%s",filename);
                   read_weights (filename);
                   printf("\n %s Weights Installed\n",filename);
                    break;
        case  '4': printf ("\nEnter Class:");
                   break;

        case  '5': activ = TRUE;                                     70
                   nodeinfo(800,600);
                   break;

        case  '6': activ = FALSE;
                   nodeinfo(800,600);
                   break;
        case  '7': printf ("\nEnter New Threshold:");
                   scanf("%f",&threshold);
                   break;
}                                                                    80
        signal(SIGINT,menu);}
```

read_weights

```
read_weights (filename)
 char filename[];
{  FILE *fp;
   float x;
#if FALSE
   int i,j,k;
   int foutput,finput,fhide_one,fhide_two;
     fp = fopen (filename,"r");
     if (fp==NULL)                                                   90
     {printf("\n ***File Error*** \n");exit(1);}
     fscanf (fp,"%d %d %d %d",&foutput,&fhide_two,&fhide_one,&finput);
     if((foutput>output)|
        (fhide_one>hide_one)|
        (fhide_two>hide_two)|
        (finput>input))

             {printf("\n ***File Too Large*** \n");exit(1);}
        loopij(foutput,fhide_two){
           fscanf(fp,"%f",&x);                                       100
           net.w2[i][j]=x;}
        loopij(fhide_two,fhide_one){
           fscanf(fp,"%f",&x);
           net.w1[i][j]=x;}
        loopij(hide_one,finput){
           fscanf(fp,"%f",&x);
           net.w0[i][j]=x;}
```

```
        loopi(foutput)    {fscanf(fp,"%f",net.t2+i);}
        loopi(fhide_two)  {fscanf(fp,"%f",net.t1+i);}
        loopi(fhide_one)  {fscanf(fp,"%f",net.t0+i);}
        fclose(fp);
#endif
}

write_weights ()                                                    write_weights

  { FILE *fp;
    int i,j,k;
     char filename[20];
#if FALSE
     printf("\nEnter filename: ");
     scanf("%s",filename);
     fp = fopen (filename,"w");
    fprintf (fp,"%d %d %d %d \n",output,hide_one,input);
    loopij(output,hide_two)
        fprintf(fp,"%f \n",net.w2[i][j],i,j);
    loopij(hide_two,hide_one)
        fprintf(fp,"%f \n",net.w1[i][j],i,j);
    loopij(hide_one,input)
        fprintf(fp,"%f \n",net.w0[i][j],i,j);
        loopi(output)    {fprintf(fp,"%f \n",net.t2[i]);}
        loopi(hide_two)  {fprintf(fp,"%f \n",net.t1[i]);}
        loopi(hide_one)  {fprintf(fp,"%f \n",net.t0[i]);}
        fclose(fp);printf("\n Weights Stored\n");
#endif
        }



header()                                                            header
{ int offx=780,offy=600;
color(1000);
linewidth(YELLOW);
rectfi(offx,offy,180+offx,(int)(70./video)+offy);
color(1);
recti(offx,offy,180+offx,(int)(70./video)+offy);
color(YELLOW);
write_int(offx+10,offy+10,  "Count:",count);
write_float(offx+10,offy+50,"Right:",(float)good/display*100);
write_float(offx+10,offy+70,"Guess:",(float)right/display*100);
write_float(offx+10,offy+30,"Test :",test);
}
```

nodeinfo

```
nodeinfo(offx,offy)
int offx,offy;
{
color(6);

rectfi(0+offx,0+offy,150+offx,80+offy);
color(1);
recti(0+offx,0+offy,150+offx,80+offy);
color(4);
  cmov2i(10+offx,55+offy);
  charstr("Current Display");
  cmov2i(10+offx,40+offy);
  charstr("Shows");
  cmov2i(10+offx,25+offy);
if (activ)
  charstr("Activation");
else
  charstr("Weight Levels");
}
```

write_float

```
write_float(x,y,title,ft)
int x,y;
char title[20];
float ft;

{ char number[20];

  sprintf(number,"%3.2f",ft);
  cmov2i(x,y);
  charstr(title);
  charstr(number); }
```

write_int

```
write_int(x,y,title,ft)
int x,y;
char title[20];
int ft;

{ char number[20];
  sprintf(number,"%d",ft);
  cmov2i(x,y);
  charstr(title);
  charstr(number); }
```

```
/*************************************************************************
*
*       DATE:10 August 1988
*       VERSION:
*
*       NAME: DISPLAY_NET
*       MODULE NUMBER: 2.7
*       DESCRIPTION: Provides display routines of counterprop
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                              20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#include "net.h"
#include "definitions.h"
#include "gl.h"
extern int count,right,good,test;
extern struct neural_net net;                                              30
extern int input,output,hide_one;

int activ = FALSE;

DISPLAY_NET()                                                  DISPLAY_NET
{
if (! activ) {

plotnodef(0,net.z,hide_one,"Z-NODES");

                                                                           40
plotnodef(1,net.inp,input,"X-Nodes:INPUT");

plotnodef(1,net.xp,input,"X-Prime Nodes");

plotnodef(2,net.z,hide_one,"Z-NODES");

plotnodef(2,net.z,hide_one," ");

plotnodef(3,net.yp,output,"Y-Prime Nodes");

                                                                           50
plotnodef(4,net.z,hide_one,"Z-Nodes");

plotnodeh(1,net.inp,input);
```

```
plotnodel(3,net.yout,output,"Y-Nodes");

plotnode (0,1,net.zxp,input,hide_one);

plotnode (1,2,net.xz,hide_one,input);

plotnode (2,3,net.yz,output,hide_one);

plotnode (3,4,net.zyp,hide_one,output);

}

}

#define weight_s 20
/* #define spcing 185 */
#define spcing 155

float threshold= 0.0;

plotnodei(x,node,lower)                                          plotnodei
int x,lower;
float node[];

{ int i,j,k,y,x2,y2;
   y=spcing*x+20;
   x=(1024-weight_s*lower)/2;
  loopi(lower) {

        set_color(1.0,0.0,node[i]);
        big_plot(i*weight_s+x,y,weight_s/4*3);

        color_of(0.5,0.5,node[i]);
        big_plot(i*weight_s+x+100,y,weight_s/4*3);

        color_of(0.9,0.1,node[i]);
        big_plot(i*weight_s+x+200,y,weight_s/4*3);

}}

plotnodef(x,node,lower,title)                                    plotnodef

int x;
float node[];
int lower;
char title[];
```

```
{ int i,j,k,y,x2,y2;
  y=spcing*x+20;;
  x=(1024-weight_s*lower)/2;

  loopi(lower) {

        set_color(1.05,0.0,0.node[i]);
        big_plot(i*weight_s+x,y,weight_s/4*3);

}

    color(6);
    cmov2i(x+150,y);
    charstr(title);

}

plotnodel(x,node,lower,title'
int x;
float node[];
int lower;
char title[];

{ int i,j,k,y,x2,y2;

  y=spcing*(x);
  x=(1024-weight_s*lower)/2;
  loopi(lower) {
        set_color(1.05,0.0,0.node[i]);
        big_plot(i*weight_s+x,y,weight_s/4*3);

}

    color(6);
    cmov2i(x+150,y);
    charstr(title);

}

plotnodeh(x,node,lower)
int x;
float node[];
int lower;

{ int i,j,k,y,x2,y2;
  y=spcing*(x)+15;
  x=(1024-weight_s*lower)/2;

  loopi(lower) {
        set_color(1.0,0.0,0.node[i]);
        big_plot2(i*weight_s+x,y,4.5);

}}
```

plotnodel

plotnodeh

plotnoded

```
plotnoded(x,node,lower)
                                                                    161
int x,node[],lower;

{ int i,j,k,x2,y;

   y=spcing*(x)+20;
   x=(1024-weight_s*lower)/2;
   loopi(lower) {
         color_of(0.9,0.1,(float)node[i]);
         big_plot(i*weight_s+x,                                     170
                  y+weight_s*2,weight_s/4*3);

}}
```

plotnode

```
plotnode(x,other,array,upper,lower)

int x,other;
float array[][size];
int upper,lower;
                                                                    180
{  int i,j,k,y,x1,x2,y2,yup;
   float max,min,temp,temp2;
   y=spcing*x+20;
   x1=(1024-weight_s*upper)/2;
   x2=(1024-weight_s*lower)/2;
   linewidth(2);

   findmax(array,&max,&min,upper,lower);

   loopi(upper){                                                    190
            loopj(lower) {
                        set_color(max,min,array[i][j]);
                        drawit(x2+weight_s*j,y+weight_s*3/4,
                                 x1+weight_s*i,other*spcing);
                       }}


   colorbar(1024-256,y+80,max,min);
}
                                                                    200
```

findmax

```
findmax(array,max,min,outs,ins)

float array[][size],*max,*min;
int outs,ins;

{ int i,j,k;

 *min = array[0][0];
 *max = array[0][0];
                                                                    210
loopij(outs,ins){     if (array[i][j]< *min) *min=array[i][j];
                         if (array[i][j]> *max) *max=array[i][j];
```

```
        }
}

display_count()                                                    display_count

  { float offx=45.*video,offy=820.*video;                                  220

      color(1000);
      rectf(offx,offy-3.,offx+80.,offy+16.);
      color(YELLOW);
      write_int((int)offx,(int)offy," ",count);}
```

```
#include "definitions.h"
#include "net.h"
float history[size];
extern int output,input,hide_one;

extern int right,good,examplars,sample,test,exam_test;

extern struct neural_net net;

CHECK_ERRORS(y,doft)
   float y[],doft[];
   { int i,j;
     float error[size];
     int correct_right=0;
     int correct_good=0;
     loopi(output) { error[i] = doft[i] - y[i];
                     if (error[i] < 0)   error[i] = -error[i];
                     if (error[i] < 0.5) correct_right++;
                     if (error[i] < 0.2) correct_good++;
                     }
               if (correct_right == (output)) right++;
               if (correct_good  == (output)) good++;
   }

float DO_CHECK()
{  int i,j,tright,tgood;
   tright = right;tgood = good;
   right=0;good=0;;
   loopi(examplars){
      MAKE_INPUT(net.inp,net.yout,i);
      FEED_FORWARD(CHECKIT);
      CHECK_ERRORS(net.yp,net.yout);
   }
   right=tright;good=tright;
   test = (float)right/(float)examplars * 100.;
   return test;}

DO_TEST()
{  int i,j;
   right=0;good=0;;
   loopi(exam_test){
      MAKE_INPUT(net.inp,net.yout,i+examplars);
      FEED_FORWARD(CHECKIT);
      CHECK_ERRORS(net.yp,net.yout);
   }
   printf("Test:Right %3.2f Good: %3.2f \n",
                  (float)right/(float)exam_test*100.,
                  (float)good/(float)exam_test*100.);
   right=0;good=0;}
```

CHECK_ERRORS

DO_TEST

```
/*************************************************************************
*
*       DATE:
*       VERSION:
*
*       NAME: Hybird Neural Net--Main Loop
*       MODULE NUMBER: 3.0
*       DESCRIPTION: Hybrid net main loop
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#include "net.h"
#include "definitions.h"
#include <stdio.h>
struct neural_net net;
extern int menu();                                                       30
int count=0,right,good,guess,test,display = 500,show_weights=500;
extern int exam_test;
float nhoodf;
extern int fast;

extern FILE *fst;
main()                                                                   main
{  hide_zero = 40 ;
   hide_one = 20;
   hide_two = 13;                                                        40
   nhoodf = (float)hide_zero;

   INITIALIZE();
   DISPLAY_NET();
   while (count < 100000) {

       check_mouse();
       MAKE_INPUT(net,inp,net,doft,-1);
       FEED_FORWARD();
       CHECK_ERRORS(net,outp,net,doft);                                  50

       BACK_PROP();
     /*    wait(100){perterb();
```

```
                FIXUP():  } */

        wait(display)SHOW();

        wait(show_weights)DISPLAY_NET();

        wait(display) { calculate_error();                                    60
                        if(exam_test!=0) DO_TEST();
                        make_graph(500.,560.);
                      }

        if(fast ==1) DISPLAY_NET();
        count++;
        display_count();
    }
fclose(fst);
save_weights();                                                               70
}
```

```
/*************************************************************************
*
*       DATE: 11 Aug 1988
*       VERSION:2.1
*
*       NAME: Defineitons.h
*       MODULE NUMBER:3.1
*       DESCRIPTION:
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#define GRAPHICS TRUE
#define TABLE 512
#define loopi(ON) for(i=0;i<ON;i++)
#define loopj(ON) for(j=0;j<ON;j++)
#define loopk(ON) for(k=0;k<ON;k++)                                      30

#define loopij(ONE,TWO) for(i=0;i<ONE;i++)for(j=0;j<TWO;j++)

#define HARD_ON  RED
#define INDETERM GREEN
#define HARD_OFF BLUE

#if FALSE
#define TRUE 1
#define FALSE 0                                                          40
#define output 4
#define input 22
#define hide_one 20
#define hide_two 26
#endif

#define line printf("\n")
#define disply 100
#define datafile "..\data\ruck.data"
#define spring 150                                                       50
#define weight_s 20
#define wait(A) if((count%(A))==0)
#define square(A) ((A)*(A))
```

```
#define NETA 0.3
#define eud_koh 10000
```

```
#define size 100

struct neural_net {

  float outp[size];
  float doft[size];
  float w2[size][size];
  float aw2[size][size];
  float t2[size];
  float outp_mask[size];

  float y2[size];
  float y2_mask[size];
  float y2_dt[size];

  float w1[size][size];
  float aw1[size][size];
  float t1[size];

  float y1[size];
  float y1_mask[size];
  float y1_dt[size];


  float w0[size][size];
  float aw0[size][size];
  float t0[size];

  float wk[size][size];
  float awk[size][size];
  float tk[size];

  float k1[size];
  float k1_mask[size];
  float k1_dt[size];
  float k1_con[size];
  int k1_claim[size];
  float inp[size];
  float inp_mask[size];

   int class_test[size];
   int class_count[size];
} ;
int input,hide_zero,hide_one,hide_two,output;
```

```
/*************************************************************************
*
*       DATE: 3 August 1988
*       VERSION:
*
*       NAME: Initialization Module
*       MODULE NUMBER: 3.2
*       DESCRIPTION: Initialization of the display hardware.
*                    any input test data, and net data structure.
*
*       ALGORITHM: None
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Net
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
#include "definitions.h"
#include"gl.h"
#include"device.h"

#include <time.h>
#include <stdio.h>
#include <signal.h>

#include "net.h"

extern struct neural_net net;

float random();

#define TABLE 512

extern int menu();

extern float conscience;

INITIALIZE()

{

    signal(SIGINT,menu);
    init_screen();
```

INITIALIZE

```
    menu();
}


init_net()                                                          init_net
{

   int i,j;
   loopij(size,size)
        net.w2[i][j] = random();
        loopi(size) net.t2[i] = random();

   loopij(size,size)
        net.w1[i][j] = random();
        loopi(size)net.t1[i] = random();

   loopij(size,size)
        net.w0[i][j] = random();
        loopi(size) net.t0[i] = random();

   loopij(size,size)
        net.wk[i][j] = 0.0;
        loopi(size) net.tk[i] = random();
        loopi (size)net.k1_cou[i] = 0.0;

   loopi(size) net.k1_claim[i] = -1;

   loopi(size) net.y2_mask[i] = TRUE;
   loopi(size) net.y1_mask[i] = TRUE;
   loopi(size) net.k1_mask[i] = TRUE;
   loopi(size) net.inp_mask[i] = TRUE;
   loopi(size) net.outp_mask[i] = TRUE;


   }

float random ()

{ float x;
  int y;
  y=(rand() % 100) + (rand() % 100) + (rand() % 100);
  x=((float)y/300.0-0.5);
  return x;

  }
```

```
/*****************************************************************************
*
*       DATE: 13 Sept 1984
*       VERSION:
*
*       NAME: Feedforward
*       MODULE NUMBER:3.4
*       DESCRIPTION: Provides forward propagation of the input signal.
*       ALGORITHM: feedforward                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                                   20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
#include "definitions.h"
#include "net.h"
#include <math.h>
extern int minimum,nhood,ctype;
extern float alpha;                                                             30
extern struct neural_net net;
extern int count ;
float calcy();
float calck();
int kclass[size];
float min_dist;
float conscience;
FEED_FORWARD()                                                  FEED_FORWARD

{ int i,j,winner;                                                               40

   loopi(input) if (net.inp_mask[i] == 0.0) net.inp[i]=0.0;

/*  Kohonen Layer   */

   loopi(hide_zero)
                  {net.k1[i]=  calck(net.inp,net.wk,net.tk,&i,input);}

   find_min_node(&winner); minimum = winner;
                                                                                50

/* Regular feedforward with mask for node wacker  */
```

```
    loopi(hide_one)
                    {net.y1[i]=  calcy(net.k ,net.w0,net.t0,&i,hide_zero);}

    loopi(hide_one) if (net.y1_mask[i] == 0.0) net.y1[i]=0.0;

                                                                          60
    loopi(output)
                    {net.outp[i]=calcy(net.y1,net.w1,net.t1,&i,hide_one);}


#if FALSE

    loopij(hide_zero,input)
          { if(net.inp_mask[j]==1.0) net.awk[i][j] = net.wk[i][j] * net.inp[j];}
    loopij(hide_one,hide_zero)
          { if(net.k1_mask[j]==1.0) net.aw0[i][j] = net.w0[i][j] * net.k1[j];}      70
    loopij(hide_two,hide_one)
          { if(net.y1_mask[j]==1.0) net.aw1[i][j] = net.w1[i][j] * net.y1[j];}
    loopij(output,hide_two)
          { if(net.y2_mask[j]==1.0) net.aw2[i][j] = net.w2[i][j] * net.y2[j];}
#endif

}

float fixy();
float calcy(x,w,theta,index,lower)                                            80
 float x[],w[][size],theta[];
 int *index;
 int lower;

 { int i,k,number;
   float y;
   y = 0.0;
   number = *index;
   loopi(lower){ y = y + x[i] * w[number][i];}
   y = y - theta[number];                                                     90
   return fixy(y,1.0);}


float fixy(y,hardness)
 float y,hardness;
{
  return( 1.0/(1.0 + (float)exp(-(double)(hardness * y))));
}

float calck(x,w,theta,index,lower)                                           100
 float x[],w[][size],theta[];
 int *index;
 int lower;

 { int i,k,number;
   float y;
```

```
   y = 0.0;
   number = *index;
   loopi(lower){ y = y + square(x[i] - w[number][i]);
   /*       printf("x[i] %3.2f w[i] %3.2f xsqr %3.2f y %3.2f \n",
                    x[i],w[number][i],square(x[i]-w[number][i]),y);*/}
   return (float)sqrt((double)y);}

float vigilence = 0.1;

find_min_node(winner)                                          find_min_node
int *winner;

{ int i;
  float min,bi,n,distance;
  int min_node;
  min = 100000000.0;
  min_node = 0;
 loopi(hide_zero){ bi = net.k1_con[i]/(float)count - 1.0/(float)hide_zero;

                   if(bi> conscience) if(alpha > 0.0) continue;
                      if(net.k1[i] < vigilence) { min_node = i; break;}
                     if(net.k1[i] > min) continue;
                     min = net.k1[i]; min_node = i;}

net.k1_con[min_node] += 1.0;

net.k1_claim[min_node] = ctype;

*winner = min_node;

kclass[min_node]=ctype;

min_dist = min;

loopi(hide_zero){
            net.k1[i] = 1.0/(0.10 + net.k1[i]);}
net.k1[min_node] *= 2.0;
}



findnode(xs,ys)                                                findnode
int xs,ys;
{int i,layer,the_node,temp;
        layer = (ys/spcing);
     the_node = (int)(xs / weight_s);
     temp = (1024/weight_s);
     clear_screen();
     switch(layer) {

       case 0: the_node -= (temp-input)/2;
               if(net.inp_mask[the_node] == 0.0)
```

```
                net.inp_mask[the_node] = 1.0;
              else
                net.inp_mask[the_node] = 0.0;
              break;

        case 1: the_node -= (temp-hide_zero)/2;
              if(net.k1_mask[the_node] == 0.0)

               { net.k1_mask[the_node] = 1.0;
                 loopi(hide_one) net.t0[i] += net.aw0[i][the_node]; }
              else
               {  net.k1_mask[the_node] = 0.0;
                  loopi(hide_one) net.t0[i] -= net.aw0[i][the_node]; }
              break;

        case 2: the_node -= (temp-hide_one)/2;
              if(net.y1_mask[the_node] == 0.0)

                 {net.y1_mask[the_node] = 1.0;
                 loopi(hide_two) net.t1[i] += net.aw1[i][the_node]; }

              else

              {   net.y1_mask[the_node] = 0.0;
                 loopi(hide_two) net.t1[i] -= net.aw1[i][the_node]; }
              break;

        case 3: the_node -= (temp-hide_two)/2;

              if(net.y2_mask[the_node] == 0.0)

               {  net.y2_mask[the_node] = 1.0;
                  }
              else

               {  net.y2_mask[the_node] = 0.0;
                  }
              break;
                 }

}
```

```
/*****************************************************************
*
*       DATE: 28 July 1988
*       VERSION:
*
*       NAME: Backprop
*       MODULE NUMBER:
*       DESCRIPTION: Using output, adjusts wights reduce error.
*       ALGORITHM: Werbos Multilayer Perceptron Backpropagation.
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: Weight Vectors
*       GLOBAL VARIABLES CHANGED: Weight Vectors
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: Main Loop
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************/

#define THETA TRUE

#include "definitions.h"
#include "net.h"
extern struct neural_net net;
extern int count,display;
float neta = 0.3;
float alpha = 0.3;
float delx();
float dely();

int minimum,nhood;
float nhoodf;
BACK_PROP()
{int i,j,k,n,mini;
 float min;
 float del3[size],del2[size],del1[size],del0[size];


 if (count > end_koh) {

 /*  output */
```

BACK_PROP

```
    loopj(output) { del3[j] = dely(net.outp[j],net.doft[j]);
      loopi(hide_two){
            net.w2[j][i] += (neta * del3[j] * net.y2[i]) * net.y2_mask[i];
          }}

/*  Second  Hidden  */

    loopj(hide_two)
          {  del2[j]= delx(net.y2[j],del3,net.w2,j,output);
             net.t1[j] +=  neta * del2[j] ;
        loopi(hide_one) {
                  net.w1[j][i] += neta * del2[j] * net.y1[i]* net.y1_mask[i];
              }}


/*  First Hidden */

    loopj(hide_one)
          { del1[j]=delx(net.y1[j],del2,net.w1,j,hide_two);
            net.t0[j] +=  neta * del1[j] ;
       loopi(hide_zero)      {
           net.w0[j][i] += neta * del1[j] * net.k1[i] * net.k1_mask[i];
                    }    }
}


/* end_koh is the point where kohonen training is turned off          */

 if(count < end_koh )   {
/* adjust neighborhood every few  counts  */
wait(200){  nhoodf = nhoodf * 0.80;
            alpha = alpha * 0.95;
            color(7);
            rectfi(45,75,200,95);
            nhood = (int)nhoodf;
            write_int(50,80,"Neighborhood:",nhood);
            }
if(nhood == 0) nhood = 1;
if(count<400) nhood = hide_zero;

/* find min distance node  */

mini = minimum;

/* adjust neighborhood weights  */

for(i= mini- nhood;i<mini+nhood;i++,k=neighbors(i,nhood,hide_zero))
{
loopj(input) net.wk[k][j] += alpha * ( net.inp[j]- net.wk[k][j]);}
```

```
}

}

neighbors(i,neigh,layer)

 int i,neigh,layer;
{ int x;
  if(i<0) x=layer+i;
  else    x=i % layer;
  return x;

}

float dely(y,doft)
 float y;
 int doft;
 { float del=0.0;;

   del = y*(1-y)*((float)doft-y);
   return del;
   }

float delx(x,del,w,n,upper)
   float x,del[],w[][size];
   int n,upper;
 { float delta,sum;
   int i,j;
   sum = 0.0;
   loopi(upper) sum = sum + del[i] * w[i][n];
   sum = x*(1-x) * sum;
   return sum;


   }
```

neighbors

```
/*****************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME:show.c
*       MODULE NUMBER: 2.5
*       DESCRIPTION: Displays internal values in the textport
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                              20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
#include "net.h"
#include <stdio.h>
#include "definitions.h"
extern struct neural_net net;
extern int count,right,guess,good,test,display;                            30
extern float nhoodf;
extern float alpha,conscience;
int l0 = 1;
int l1 = 0;
int l2 = 0;
int l3 = 0;
int l4 = 1;
extern FILE *fst;
SHOW()                                                                     SHOW
{   if(l0) showinput(net.inp) ;                                            40
    if(l4) showoutput(net.outp,net.doft);
    printf("Alpha = %3.2f Conscience %3.2f\n",alpha,conscience);
shownode(net.k1_con.hide_zero,"Conscience");

/*      showweights(net.wk,net.k1_con.1.hide_zero.input);
*/}

showinput(x)                                                               showinput
 float x[];
 {   int j,i;                                                              50
     printf("Multilayer Perceptron Model\n");
     loopi(input) printf(" X(%d) ",i);line;
```

```
    loopi(input) printf("%1.2f  ",x[i]);line;line;
}

float pright,pgood;
showoutput(y,doft)                                              showoutput
   float y[];
   float doft[];                                                        60
   { int i,j;
     float error;
 printf("Out:");

    loopi(output) printf("%2.2f  ",y[i]);line;

printf("DofT:");
    loopi(output) printf(" %3.2f     ",doft[i]); line;

    pright = (float)right/display * 100.0;                              70
    pgood  = (float)good/display * 100.0;

    printf("Count:%d \nRight:%2.2f Guess:%2.2f \n",count,
                                pright,pgood);
    fprintf(fst,"Count:%d Right:%2.2f Guess:%2.2f ",count,
                                pright,pgood);
    fflush(fst);
right=0;
good=0;
                                                                    80
   }


shownode(y,n,mess)                                              shownode

   float y[];
   int n;
   char mess[];
   { int i,j;
     float max,min;                                                 90
    line;
    printf ("%s ",mess);line;
    loopi(n)  printf("%2.2f ",y[i]);line;

    }

showweights(w,theta,layer,upper,lower)                          showweights

 float w[][size],theta[];
 int layer,upper,lower;                                             100
 {   int j,i;
     float max,min;
    line;
    loopi(lower) {
    loopj(upper) { if (w[j][i] < 0.0)
```

```
                printf("%1.1f ",w[j][i]);
            else
                printf(" %1.1f ",w[j][i]);}

    line;}
    printf( "\nThetas: \n");
    loopi(upper){ printf( "%1.1f ",theta[i]);}


    line;
}
```

```
/*************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME:menu.c
*       MODULE NUMBER: 2.6
*       DESCRIPTION: Provides of interactive menu options
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/
extern int dis_class,sample,exam_test;
extern float uhoodf;
extern float add_noise,alpha;
extern int activ,show_weights,display;
extern float threshold,add_noise,conscience;                             30

#include <signal.h>
#include <stdio.h>
#include <gl.h>
#include "../wacker/definitions.h"
#include "net.h"

extern int mistakes;
extern struct neural_net net;
extern int count,right,good,guess,test;                                  40
extern int hide_one,hide_zero,hide_two,input,output;

int menu()

{   char select,filename[20];
    int nodenumber,selector=TRUE;


    system("clear");
                                                                         50
    printf (" Menu \n");
    printf (" 1) Initialize System");line;
    printf (" 2) Save Weights");line;
```

```
printf (" 3) Read Weights ");line;
printf (" 4) Toggle Act/Weights");line;
printf (" 5) Add  Noise");line;
printf (" 6) Display Intervals");line;
printf (" 7) Toggle Errors ");line;
printf (" 8) New Net Topology");line;
printf (" e) EXIT \n");                                          60
printf ("          SELECTION: ? ");
fflush(stdin);
select = getchar();
  switch(select) {
      case 'e':
                gexit();
                exit(0);
                break;
      case  '1': init_data();
                 printf("Enter alpha,conscience,first,second:\n");   70

          /*     alpha = 1.0;conscience = 0.03;hide_zero = 10;
                 hide_one = 5; */

               scanf("%f %f %d %d",&alpha,&conscience,&hide_zero,&hide_one);



               printf ("\n%3.2f %3.2f %d %d\n",alpha,conscience,
                        hide_zero,hide_one);                     80
              init_net();

                    ;break;
      case  '2': write_weights ():
                          ;break;
      case  '3': printf("\nEnter Filename: \n ? ");
                 scanf("%s",filename);
                 read_weights (filename);
                 printf("\n %s Installed\n\n",filename);
                  break;                                         90

      case '4': activ  = TRUE;
                break;


      case  '5': printf("How much noise: ?");
                 scanf("%f",&add_noise);
                 break;

      case  '6': printf("\nCount between Screen Update ?\n");     100
                 scanf("%d",&show_weights);
                 printf("\nCount between tests ?\n");
                 scanf("%d",&display);

                 break;
      case  '7': mistakes  = TRUE;
```

```
                break;
        case  '8': system("clear");
                printf("New Network Topology\n");
                printf("Number in Kohonen Layer\n");
                scanf("%d",&hide_zero);
                printf("Number in First Layer\n");
                scanf("%d",&hide_one);
                printf("Number in Second Layer\n");
                scanf("%d",&hide_two);
                init_net();
                nhoodf = (float)hide_zero;
                alpha = 0.5;
                nhoodf = (float)hide_zero;
                printf("alpha:%1.3f nhood:%.3.2f ",alpha,nhoodf);
                break;
}

        signal(SIGINT,menu);}

read_weights (filename)
 char filename[];
{  FILE *fp;
   float x;
   int i,j,k;
     fp = fopen (filename,"r");
     if (fp==NULL)
     {printf("\n ***File Error*** \n");return;}
     fscanf (fp,"%d %d %d %d %d ",
           &output,&hide_two,&hide_one,&hide_zero,&input);

       loopij(output,hide_two){
           fscanf(fp,"%f",&x);
           net.w2[i][j]=x;}
       loopij(hide_two,hide_one){
           fscanf(fp,"%f",&x);
           net.w1[i][j]=x;}
       loopij(hide_one,hide_zero){
           fscanf(fp,"%f",&x);
           net.w0[i][j]=x;}
       loopij(hide_zero,input){
           fscanf(fp,"%f",&x);
           net.wk[i][j]=x;}

           fscanf(fp,"%f %f ",&nhoodf,&alpha);

           loopi(output)    {fscanf(fp,"%f",net.t2+i);}
           loopi(hide_two)  {fscanf(fp,"%f",net.t1+i);}
           loopi(hide_one)  {fscanf(fp,"%f",net.t0+i);}
           fscanf(fp,"%d",&count);
           loopi(hide_two)  {fscanf(fp,"%f",net.y2_mask+i);}
           loopi(hide_one)  {fscanf(fp,"%f",net.y1_mask+i);}
           loopi(input) {fscanf(fp,"%f",net.inp_mask+i);}
           fclose(fp);}
```

read_weights

write_weights

```
write_weights ()

  { FILE *fp;
    int i,j,k;
     char filename[20];
     printf("\nEnter filename: ");
     scanf("%s",filename);
     fp = fopen (filename,"w");
    fprintf (fp,"%d %d %d %d %d \n",
            output.hide_two,hide_one,hide_zero,input);
    loopij(output.hide_two){ {
          fprintf(fp,"%f ",net.w2[i][j],i,j);} fprintf(fp,"\n");}
    loopij(hide_two.hide_one)
          fprintf(fp,"%f ",net.w1[i][j],i,j);
    loopij(hide_one.hide_zero)
          fprintf(fp,"%f ",net.w0[i][j],i,j);

    loopij(hide_zero.input)
          fprintf(fp,"%f ",net.wk[i][j],i,j);

          fprintf(fp,"%f %f ",nhoodf.alpha);


          loopi(output)   {fprintf(fp,"%f \n",net.t2[i]);}
          loopi(hide_two) {fprintf(fp,"%f \n",net.t1[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.t0[i]);}
          fprintf(fp,"%d \n",count);
          loopi(hide_two) {fprintf(fp,"%f \n",net.y2_mask[i]);}
          loopi(hide_one) {fprintf(fp,"%f \n",net.y1_mask[i]);}
          loopi(input) {fprintf(fp,"%f \n",net.inp_mask[i]);}
          fclose(fp);printf("\n Weights Stored\n");
          }
```

save_weights

```
save_weights ()

  { FILE *fp;
    int i,j,k;
     char filename[20];
    fp = fopen ("hybrid_net.w","w");
    fprintf (fp,"%d %d %d %d %d \n",
            output.hide_two,hide_one,hide_zero,input);
    loopij(output.hide_two){ {
          fprintf(fp,"%f ",net.w2[i][j],i,j);} fprintf(fp,"\n");}
    loopij(hide_two.hide_one)
          fprintf(fp,"%f ",net.w1[i][j],i,j);
    loopij(hide_one.hide_zero)
          fprintf(fp,"%f ",net.w0[i][j],i,j);

    loopij(hide_zero.input)
          fprintf(fp,"%f ",net.wk[i][j],i,j);

          fprintf(fp,"%f %f ",nhoodf.alpha);
```

```
loopi(output)    {fprintf(fp,"%f \n",net.t2[i]);}
loopi(hide_two) {fprintf(fp,"%f \n",net.t1[i]);}
loopi(hide_one) {fprintf(fp,"%f \n",net.t0[i]);}
fprintf(fp,"%d \n",count);
loopi(hide_two) {fprintf(fp,"%f \n",net.y2_mask[i]);}
loopi(hide_one) {fprintf(fp,"%f \n",net.y1_mask[i]);}
loopi(input) {fprintf(fp,"%f \n",net.inp_mask[i]);}
fclose(fp);printf("\n Weights Stored\n");
}
```

write_string

```
write_string(x,y,l,title)
int x,y,l;
char title[20];

{ char number[20];
  l = (int)((float)l/video);
  color(6);
  rectfi(x-5,y-5,x+l,y+15);
  color(4);
  linewidth(1);
  recti(x-5,y-5,x+l,y+15);
  cmov2i(x,y-2);
  charstr(title);
   }
```

write_float

```
write_float(x,y,l,title,ft,a_color)
int x,y,l;
char title[20];
float ft;
int a_color;

{ char number[20];
  l = (int)((float)l/video);

  sprintf(number,"%3.3f",ft);
  color(a_color);
  rectfi(x-5,y-5,x+l,y+15);
  color(YELLOW);
  linewidth(1);
  cmov2i(x,y-2);
  charstr(title);
  charstr(number); }
```

write_int

```
write_int(x,y,title,ft)
int x,y;
char title[20];
int ft;

{ char number[20];
```

```
  color(YELLOW);
  sprintf(number,"%d",ft);
  cmov2i(x,y);
  color (BLUE);
  charstr(title);
  charstr(number); }                                                    270


write_an_int(x,y,ft)                                                write_an_int
int x,y;
int ft;

{ char number[20];
  sprintf(number,"%d",ft);
  cmov2i(x,y);
  cnarstr(number); }                                                    280
```

```
/*****************************************************************************
*
*       DATE:10 August 1988
*       VERSION:
*
*       NAME: DISPLAY_NET
*       MODULE NUMBER: 2.7
*       DESCRIPTION: Displays internal values in a graphic format
*       ALGORITHM: None                                                    10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                              20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************************/
#include "net.h"
#include "definitions.h"
#include "gl.h"
extern int count,right,good,test,minimum,ctype;
extern struct neural_net net;                                              30
extern kclass[];
extern float pright,pgood;
int activ = FALSE;

DISPLAY_NET()                                                  DISPLAY_NET
{ int i,j,k;
  char cks[4];
do_screen();
write_string(480,500,250,"Value     Guess      Right");
write_string(480,540,150,"Desired Output");                                40
write_string(10,730,300,"Hybrid Propagation Network");


if (! activ) {

plotnode (0,net.inp,net.inp_mask,net.k1_mask,net.wk,input,hide_zero);
plotnodek (1,net.k1,net.k1_mask,net.y1_mask,net.w0,hide_zero,hide_one);
plotnode (2,net.y1 ,net.y1_mask,net.outp_mask,net.w1,hide_one,output);
write_string(100,220,120,"Weights");}

                                                                           50
else {
```

```
plotnode (0,net.inp,net.inp_mask,net.k1_mask,net.awk,input,hide_zero);
plotnodek (1,net.k1,net.k1_mask,net.y1_mask,net.aw0,hide_zero,hide_one);
plotnode (2,net.y1 ,net.y1_mask,net.outp_mask,net.aw1,hide_one,output);
write_string(100,220,120,"Activation");}

plotnodei(3,net.outp,output);
plotnoded(3.0,net.doft,output);                                          60
plotnoded(3.100,net.doft,output);
plotnoded(3,200,net.doft,output);
plotnoder(0.7,0,net.k1,hide_zero);
plotstats();

check_mouse();
}

plotstats()                                                              plotstats
{                                                                        70

  write_float(10,320,130,"Right:",pright,1000);
  write_float(180,320,130,"Guess:",pgood,1000);}


float threshold = 0.0;

plotnodei(x,node,lower)                                                  plotnodei
int x;
float node[];                                                            80
int lower;
{ int i,j,k,y,x2,y2;


   y=spcing*x+30;
   x=(1024-weight_s*lower)/2;
   loopi(lower) {
         set_color(1.1,0.01,node[i]);
         big_plot(i*weight_s+x,y,weight_s/4*3);
         color_of(0.5,0.5,node[i]);                                      90
         big_plot(i*weight_s+x+100,y,weight_s/4*3);
         color_of(0.9,0.1,node[i]);
         big_plot(i*weight_s+x+200,y,weight_s/4*3);

}

}

plotnoded(x,y2,node,lower)                                               plotnoded
int x,y2;                                                                100
int node[];
int lower;
{ int i,j,k,x2,y;
   y=spcing*x+30;
   x=(1024-weight_s*lower)/2;
```

```
loopj(lower){
  loopi(lower) {
        color_of(0.9,0.1,(float)node[i]);
        big_plot(i*weight_s+x+y2,
                y+weight_s*2,weight_s/4*3);
}}}

plotnoder(x2,y2,node,lower)
float x2,node[];
int y2;
int lower;
{ int i,j,k,x,y;
  float maxr,minr;
  char temp[3];
   y=(int)(spcing*x2+35);
   x=(1024-weight_s*lower)/2;

  findmaxnode(node,&maxr,&minr,lower);
  colorbar(100,50,maxr,minr);
  loopi(lower) {

        set_color(maxr,minr,node[i]);
        big_plot(i*weight_s+x+y2,
                y+weight_s*2,weight_s/4*3+1);

    color(YELLOW);
        write_an_int(i*weight_s+x+y2-3,
                y+weight_s*2+4,kclass[i]);

}}


plotnode(x,node,mask,mask_up,array,lower,upper)
int x;
float node[],mask[],mask_up[],array[][size];
int lower,upper;
{ int i,j,k,y,x1,x2,y2;
  float max,min,temp,temp2;
  cursoff();
   y=spcing*x+30;
   x1=(1024-weight_s*lower)/2;
   x2=(1024-weight_s*upper)/2;
  linewidth(2);
  findmax(array,&max,&min,upper,lower);
  loopi(lower) {
        if(mask[i]==1.){
                color_of(0.9,0.1,node[i]);
                big_plot(i*weight_s+x1,y,weight_s/4*3);


    loopj(upper) {
        if(mask_up[j] == 1.0){
                set_color(max,min,array[i][j]);
```

plotnoder

plotnode

```
                drawit(x1+weight_s*i,y+weight_s*3/4,
                x2+weight_s*j,y+spcing);}
                  }
                }


                }
    colorbar(1024-256,y + 80,max,min);
    curson();}


plotnodek(x,node,mask,mask_up,array,lower,upper)
int x;
float node[],mask[],mask_up[],array[][size];
int lower,upper;
{ int i,j,k,y,x1,x2,y2;
  float max,min,temp,temp2;
  curson();
   y=spcing*x+30;
   x1=(1024-weight_s*lower)/2;
   x2=(1024-weight_s*upper)/2;
  linewidth(2);
  findmax(array,&max,&min,upper,lower);
  loopi(lower) {
          if(mask[i]==1.){
     loopj(upper) {
            if(mask_up[j] == 1.0){
                set_color(max,min,array[i][j]);
                drawit(x1+weight_s*i,y+weight_s*3/4,
                x2+weight_s*j,y+spcing);}
                  }
                }


                }
     colorbar(1024-256,y + 80,max,min);
     curson();}

findmax(array,max,min,outs,ins)

float array[][size],*max,*min;
int outs,ins;
{ int i,j,k;
  int maxi=0,maxj=0;
  int mini=0,minj=0;
 *min = array[0][0];
 *max = array[0][0];
loopi(ins){
  loopj(outs){     if (array[j][i]< *min) *min=array[j][i];

                   if (array[j][i]> *max) *max=array[j][i];
                }
               }}
```

plotnodek

findmax

findmaxnode

```
findmaxnode(array,max,min,lower)

float array[],*max,*min;
int lower;
{ int i,j,k;
  int maxi=0,maxj=0;
  int mini=0,minj=0;
 *min = array[0];
 *max = array[0];
  for(j=lower){     if (array[j]< *min) *min=array[j];

                    if (array[j]> *max) *max=array[j];
                }
               }
```

display_count

```
display_count()
 {
      linewidth(1);
      color(7);
      rectfi(800,600,1023,620);
      color(500);
      recti(800,600,1023,620);
      color(4);
      write_int(805,603," ",count);}
```

```
extern int count,right,good,test,exam_test,examplars,sample,ctype;
extern struct neural_net net;
#include "definitions.h"
#include "net.h"
#include <math.h>
#include <stdio.h>
int mistakes = 0;
extern FILE *fst;
CHECK_ERRORS(y,doft)                                          CHECK_ERRORS
   float y[size];                                                          10
   float doft[size];
   { int i,j;
     float error[size];
     int correct_right=0;
     int correct_good=0;
    loopi(output) { error[i] =doft[i]- y[i];
                    if (error[i] < 0)    error[i] = -error[i];
                    if (error[i] < 0.5) correct_good++;
                    if (error[i] < 0.2) correct_right++;
                    }                                                      20
              if (correct_good  == (output)) good++;
               else
                if (mistakes) printf("Sample %2d Type %d\n",sample,ctype);

              if (correct_right == (output)) right++;
   }
float err,olderr=0.0,derverr=0.0;
CHECK(y,doft)                                                 CHECK
   float y[size];
   float doft[size];                                                       30
   { float missed_by;
    int i,j;
    loopi(output)  {
                    missed_by = doft[i]- y[i];
                    err += missed_by * missed_by;

   }              }
DO_TEST()                                                     DO_TEST
{  int i;
   right=0;good=0;;                                                        40
   loopi(exam_test){
      MAKE_TEST(net.inp,net.doft,i);
      FEED_FORWARD();
      CHECK_ERRORS(net.outp,net.doft);
   }
   printf("Test: %3.2f          %3.2f \n",(float)right/(float)exam_test*100.,
                  (float)good/(float)exam_test*100.);
   fprintf(fst,"Test: %3.2f          %3.2f \n",(float)right/(float)exam_test*100.,
                  (float)good/(float)exam_test*100.);
   fflush(...);                                                            50
   right=0;good=0;
}
```

perterb

```
perterb()
{   int i;
    err= 0.;

    loopi(examplars){
      MAKE_INPUT(net.inp,net.doft,i);
      FEED_FORWARD();
      CHECK(net.outp,net.doft,&err);
    }

  err = (float)sqrt((double)err);
  derverr = (err - olderr);
  olderr = err;


}
write_error()
{
  write_float(810,680,180,"Change: " derverr,3);
  write_float(810,705,180,"Error : ",err,3);
}
```

write_error

```
/*************************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME: Two Dimensional Kohonen Map
*       MODULE NUMBER: 4.0
*       DESCRIPTION:
*       ALGORITHM: None                                                  10
*       PASSED VARIABLES: None
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None                                            20
*
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/

#define kohonen 1

/*  I tried to keep as close to the original article as possible */
                                                                         30
#include <math.h>
#include "../graph.c"

#define side 20    /* define size of one side */

#define imax side*side  /* number of neurons */
#define jmax 2     /* no of inputs */
#define a0 0.3     /* starting gain */
#define g 0.2      /* initialization factor */
#define ci 1       /* color use in plotting points */                    40
#define update 1000
enum densityfunctions {square, triangle, cross};
int ik,t1,t,t2,w0,w,h1,h,v1,v,i,j,c;
float a,a1,a2,miny,m[imax][jmax],x[jmax],n[imax],y[imax];


/*  a                   current alpha value
    a0                  starting alpha value
    a1
    a2                                                                   50
    miny
    m[imax][jmax]       weights for ---  to ----
    x[jmax]             x[0] = x value x[1] = y value
```

```
    n[imax]
    y[imax]

*/

float random().gauss();
int plotcolor();
enum densityfunctions densityfunction;

main()                                                          main
{
/*   get type of function   */


   initialize();
     srand(2.1);   /* set seed */
     a = a0; a1 = a; w0 = side / 2; t1 = update; t2 = 10; t = 0; tk = 0;

   for (i=0; i<imax; i++)  {
      n[i] = 0.0;
      for (j=0; j<jmax; j++)  {
         m[i][j] =  (0.5-g/2.0) + g*((float)random());

         n[i] = n[i] + m[i][j]*m[i][j];
      }   /* end j loop */
      n[i] = n[i] / 2.0;
   }   /* end i loop */

   update_screen();



/* ************************************************************ */


   while (a != 0.0)
{
      for (t=1; t<=t1; t++,tk=tk+1)  {

   /*     tk = tk + 1; */

         readinput();
         miny = n[0];
         for (i=0; i<imax; i++)  {
            y[i] = n[i];
            for (j=0; j<jmax; j++) y[i] = y[i] - m[i][j] * x[j];
            if (y[i] <= miny)  {
               miny = y[i];
               c = i;
            }   /* end if */
         }   /* end for i loop */

         a1 = a*(1.0 - (float)t/(float)t1);  a2 = 1.0 - a1;
```

```
        h1 = c % side;
        v1 = c / side;
        w = (int)((float)w0 * (1.0 - (float)t/(float)t1)) + 1;

        for (h=max(0,h1-w); h<=min(side-1,h1+w); h++)  {
           for (v=max(0,v1-w); v<=min(side-1,v1+w); v++)  {
              i = side*v +h; n[i] = 0.0;
              for (j=0; j<jmax; j++)  {
                 m[i][j] = a1*x[j] + a2*m[i][j];
                 n[i] = n[i] + m[i][j]*m[i][j];
              }  /* end for j loop */
              n[i] = n[i] / 2.0;
           }  /* end for v loop */
        }  /* end for h loop */
                              if ((x[0] < 0.5) &&( x[1]<0.5))
                                  color(WHITE);
                              else color(MAGENTA);
                              big_plot(h1*10+2.400+v1*10.5);
                              plotinput();

        if ((t % t2) == 1) update_screen();

     }  /* end for t loop */

     a = 0.2*a; w0=0; t1=5*t1; t2=5*t2;

  }  /* end while loop */

}  /* end main program */


readinput()                                                    readinput
{
  int inside;

  inside = 0;
   while (inside == 0)  {

     makeinput();
     switch (densityfunction)  {
        case square   : inside = 1;
                        break;
        case triangle : if (x[1] > 2.0*(abs(x[0] - 0.5)) inside = 1;
                        break;
        case cross    : if ((fabs(x[0]-0.5) <= 0.1) ||
                            (fabs(x[1]-0.5) <= 0.1)) inside = 1;
                        break;
     }  /* end of case */

  }  /* end while loop */

}  /* end readinput */
```

```
#if FALSE                                                          160

big_plot(x2,y2,si)                                          big_plot
int x2,y2,si;


{ linewidth(si);
  drawit(x2,y2,x2,y2+si);
  linewidth(1);
}                                                                  170

#endif
plotinput()                                                 plotinput
{ int ax,by;
  ax=(int)(200.0 + x[0] * 200.0);
  by=(int)(400.0 + x[1] * 200.0);
  big_plot(ax,by,3);}

makeinput()                                                 makeinput
{                                                                  180
   x[0]=gauss();
   x[1]=gauss(); }

float gauss()
{ int i,times=5;
  float x1=0;
  for(i=0;i<times;i++)
    x1 += random();
    x1 = x1/times;return x1;}
                                                                   190


drawdistribution()                                          drawdistribution
{
#define xw 300  /* define x width of plotting window */
#define yw 300  /* define y width of plotting window */
#define xbase 0  /* where x=0 is on screen */
#define ybase 0   /* where y=0 is on screen */

   int x1,y1,x2,y2;                                                200
      /* end of case */

      x1 = xbase;
      y1 =  ybase;
      x2 = xw +  xbase;
      y2 = yw + ybase;
color(YELLOW);
cmov2i(50,320);
charstr("Kohonen Topology");
                                                                   210
color(WHITE);
```

```
        recti(x1,y1,x2,y2);

color(CYAN);
cmov2i(50,620);
charstr("Kohonen Map");

color(BLUE);
        recti(0,400,200,600);                                          220

color(CYAN);
cmov2i(250,620);
charstr("Normalized");
cmov2i(220,605);
charstr("Input Distribution");

color(BLUE);
        recti(201,400,400,600);
color(YELLOW);                                                         230

    for (h=0; h<side; h++)  {

        x1 = (xw * m[h][0]) + xbase;
        y1 = (yw * m[h][1]) + ybase;
        x2 = (xw * m[side*h][0]) + xbase;
        y2 = (yw * m[side*h][1]) + ybase;
        drawline(&x1,&y1,h,side); drawline(&x2,&y2,side*h,1);

        for (v=1; v<side-1; v++)  {                                    240
            drawline(&x1,&y1,side*v+h,0);
            drawline(&x1,&y1,side*v+h,side);
            drawline(&x2,&y2,side*h+v,0);
            drawline(&x2,&y2,side*h+v,1);
        }   /* end for v loop */

        drawline(&x1,&y1,side*(side-1)+h,0);
        drawline(&x2,&y2,side*h+side-1,0);

    }   /* end for h loop */                                           250

    }   /* end drawdistribution */
```

drawline

```
drawline(xb,yb,i2,e)
    int *xb,*yb,i2,e;
{
    int xo,yo;

    xo = *xb; *xb = ((xw/2.0) * (m[i2][0] + m[i2+e][0])) + xbase;      260
    yo = *yb; *yb = ((yw/2.0) * (m[i2][1] + m[i2+e][1])) + ybase;

    drawit(xo,yo,*xb,*yb);
}   /* end drawline */
```

max

```
max (a,b)
int a,b;
{ if (a>b) return a;
  else return b;
}
```

min

```
min (a,b)
int a,b;
{ if (a<b) return a;
  else return b;
}
float  random ()
 { float x;
    x= ((float)rand() / 32768.0);
    return x;
}


#define weight_s 8
```

plotweight

```
plotweight()

{ int i,j,k,which,ix,iy;
  float minweight,maxweight;
     minweight=m[0][which];
     maxweight=m[0][which];
 /*  Get maximum and minimum weights for plotting  */
for(which=0;which<jmax;which++)
   {for (k=0;k<imax;k++){
        if (m[k][which]>maxweight) maxweight=m[k][which];
        if (m[k][which]<minweight) minweight=m[k][which]; }}

/*            Now, plot the weights       */

for(which=0;which<jmax;which++)
   for (k=0;k<imax;k++){
      ix=450+220*which+(weight_s+2)*(k/side);
      iy=400+(weight_s+2)*(k%side);

      color(plotcolor(maxweight,minweight,m[k][which]));

      big_plot(ix,iy,weight_s);

      }

/*  display a color bar to show the range of the weights  */

   colorbar(400,350,maxweight,minweight);

}
```

```
#define TABLE 64
int plotcolor(max,min,value)
float max,min,value;                                                    320
  {
    return 8+(int)(((value-min)/(max-min))*(TABLE-8));}

colorbar(x,y,max,min)                                                   colorbar
int x,y;
float max,min;
{ char maxstring[20],minstring[20];
   for (i=8;i<TABLE;i++){
   color(i);
   big_plot(x+i*7,y,7);}                                                330

    sprintf(maxstring,"%3.3f",max);
    sprintf(minstring,"%3.3f",min);
   cmov2i(x+50,y+20);
   charstr(minstring);
   cmov2i(x+400,y+20);
   charstr(maxstring);
}

write_integer(x,y,title,i)                                              write_integer
 int x,y;                                                               341
 char title[20];
 int i;
{ char number[20];

   sprintf(number,"%d",i);
   cmov2i(x,y);
   charstr(title);
   charstr(number); }

                                                                        350
write_float(x,y,title,ft)                                               write_float
 int x,y;
 char title[20];
 float ft;

{ char number[20];

   sprintf(number,"%3.3f",ft);
   cmov2i(x,y);
   charstr(title);                                                      360
   charstr(number); }

update_screen()                                                         update_screen

        {drawdistribution();
         plotweight();
         color(GREEN);
         write_float  (700,230,"Alpha:",al);
         write_integer(700,200,"Count:",tk);
```

```
swapbuffers();
color(BLACK);
clear();}
```

```
#include"gl.h"
#include"device.h"

#define video 0.65

initialize()                                                    initialize
{ int i;

    winopen("koh");
    doublebuffer() ;                                                10
    gconfig();
    viewport(0,(int)(1047.*video),0,(int)(1047.*video));

    color(BLACK);
    frontbuffer(TRUE);
    clear();
    frontbuffer(FALSE);
    clear();
    linewidth(1);
    name_things();                                                  20
    for (i=8;i<64;i++) mapcolor(i,i*4,i*4,i*4);
    clear();
    qdevice(MIDDLEMOUSE);
}

#if FALSE
check_input()                                                   check_input
{ short val;
  int x,y;
  char message[20];                                                 30
  if (qtest()!=0){
  switch (qread(&val)){

          case MIDDLEMOUSE: /*if(val==1){
                            x=getvaluator(MOUSEX);
                            y=getvaluator(MOUSEY);*/
                            dis_class = (dis_class + 1) % 3;
                            printf("hello(1)");
                            break;
  }}}
#endif                                                              40
#ifdef perceptron
#define TABLE 511
#define HARD_ON   RED
#define INDETERM GREEN
#define HARD_OFF BLUE
initialize()                                                    initialize
{ int i,j,k;

   winopen("net");                                                  50
   ginit();
   gconfig();
```

```
    color(BLACK);
    clear();
    linewidth(1);

    color(YELLOW);
    cmov2i(400,700);
    charstr("Multi-Layer Perceptron");

    color(6);
    cmov2i(10,700);
    charstr("D of T");

    cmov2i(10,680);
    charstr("Output");

    cmov2i(80,660);
    charstr("Actual");

    cmov2i(140,660);
    charstr("Guess");

    cmov2i(220,660);
    charstr("Right");


    cmov2i(10,645);
    charstr("Hide 3");

    cmov2i(10,600);
    charstr("Node Y2");

    cmov2i(10,450);
    charstr("Hide 2");

    cmov2i(10,350);
    charstr("Node Y1");

    cmov2i(10,250);
    charstr("Hide 1");

    cmov2i(10,125);
    charstr("Inputs");
    mapcolor(GREEN,50,50,50);

    for(i=TABLE/2,k=0,j=8;j<TABLE+8;j++)
        { if (j<264) {i--; mapcolor(j,0,70,i);}
          if  (j>263) {k++; mapcolor(j,k,70,0);}
}
}
#endif

name_things()
{
```

name_things

```
color(4);cmov2i(0.680);charstr("tank");
color(5);cmov2i(0.620);charstr("jeep");
color(6);cmov2i(0.640);charstr("POL");
color(7);cmov2i(0.660);charstr("truck");
}
#if perceptron

#define weight_s 8

plotweight(x,y,array,inp,out,maxweight,minweight)
 int x,y;
 float array[];
 int inp,out;
 float maxweight,minweight;

{ int i,j,k,which,ix,iy;

 /*  Get maximum and minimum weights for plotting  */

      findmax(array,&maxweight,&minweight,out,inp);
      colorbar(x,y,maxweight,minweight);
      20; y +=20;
/*            Now. plot the weights   line    */
     for (i=0;i<inp;i++){
           for (j=0;j<out;j++) {
                 set_color(maxweight,minweight,array[i*output+j]);

                 big_plot(j*weight_s+x,i*weight_s+y,weight_s-1);

                      }          }


}
plotnode(x,y,node,leng)
int x,y;
float node[];
int leng;
{ int i,j,k;
  loopi(leng) {
         set_color(1.0,0.0,node[i]);
         big_plot(i*12+x,y,10);
      }
}
plotfire(x,y,hi,lo,node,leng)
int x,y;
float hi,lo,node[];
int leng;
{ int i,j,k;
  loopi(leng) {
         color_of(hi,lo,node[i]);
         big_plot(i*12+x,y,10);
      }
```

plotweight

plotnode

plotfire

```
}
color_of(hi,lo,value)                                                    color_of
float hi,lo,value;
{ color(INDETERM);
  if(value > hi) color(HARD_ON);
  if(value < lo) color(HARD_OFF);
  }

plotnodei(x,y,node,leng)                                                 plotnodei
int x,y;
int node[];
int leng;
{ int i,j,k;
  loopi(leng) {
          set_color(1.0,0.0,(float)node[i]);
          big_plot(i*12+x,y,10);
       }
}
```

```
findmax(array,max,min,outs,ins)                                          findmax

float array[],*max,*min;
int outs,ins;
{ int i,j,k;
 *min = array[0];
 *max = array[0];

loopi(outs*ins){if (array[i]< *min) *min=array[i];
                if (array[i]> *max) *max=array[i];
                  }

}

plot_node();
```

```
colorbar(x,y,max,min)                                                    colorbar
int x,y;
float max,min;
{ int i;
   char maxstring[20],minstring[20];
   for (i=8;i<TABLE+8;i++,i++){
```

```
    color(i);
    big_plot2(x+i/2,y,1,10);}

    sprintf(maxstring,"%3.3f",max);
    sprintf(minstring,"%3.3f",min);
/*    color(BLUE);
    big_plot2(x+30,y+20,80,10);*/
    color(YELLOW);
    cmov2i(x,y);
    charstr(minstring);
/*    color(BLUE);
    big_plot2(x+195,y+20,80,10);
    color(YELLOW);*/
    cmov2i(x+180,y);
    charstr(maxstring);
}

set_color(max,min,value)

float max,min,value;
{ float percent;
    int colx;
    if(value > max) value = max;
    if(value < min) value = min;


    percent = (value-min)/(max-min) * (float)(TABLE-2);
    colx = (int)percent + 8;
    color(colx);
}
#endif

big_plot(x,y,size_of_dot)
int x,y,size_of_dot;
{ linewidth(size_of_dot);
  drawit(x,y,x,y+size_of_dot);
  linewidth(1);
  }

big_plot2(x,y,ww,hh)
int x,y,ww,hh;
{ linewidth(ww);
  drawit(x,y,x,y+hh);
  linewidth(1);
  }


drawit(xstart,ystart,xend,yend)

int xstart,ystart,xend,yend;
```

set_color

big_plot

big_plot2

drawit

```
{
        move2i(xstart,ystart);
        draw2i(xend,yend);
}

drawit2(xstart,ystart,xend,yend)

float xstart,ystart,xend,yend;

{
        move2(xstart,ystart);
        draw2(xend,yend);
}
```

drawit2

270

```
/*************************************************************************
*
*       DATE: 2 Sept 1988
*       VERSION: 1.0
*
*       NAME:  Error Surface Demonstration
*       MODULE NUMBER:5.0
*       DESCRIPTION: Error surface generation main loop
*       ALGORITHM: Feedforward Backpropagation
*       PASSED VARIABLES: None                                          10
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*                                                                       20
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*************************************************************************/

float w1=1.,w2=1.,x,y,output,doft,errors= 1.,target= 0.05;
float olderror;
float random ();
float randomx();
#define video 0.65                                                      30
#include "surface.c"
#include <time.h>
#define GRAY 1000
#define GRAY2 1001
float neta= 0.3;
int wrong=4;
int which=0,which2=0;
int check_mouse();
int count;
float start_w1,start_w2;                                                40
main()                                                                  main
{ int i;
   float oldw1,oldw2;
   long now;
   srand(time(&now) % 37 );
   surface(TRUE);

   init_mouse();
   mapcolor(1000,100,100,100);
   textport(760,1020,0,220);                                            50
   pagecolor(1000);
   textcolor(CYAN);
   system ("clear");
```

```
        menu();
        move(w1,w2,4.);
        color(YELLOW);
                        initialize ();
        for(i=0,count=0;i<200;i++){
        while (errors > target ) {
                        makeinput();
                        feedforward();
                        move(w1,w2,4.0);
                       oldw1 = w1;oldw2 = w2;

                      if (which2 == 0)
                           backpropagate();
                      else  backpropagate2();

          if (which2 == 1) color(YELLOW);
          else  color(RED);
                        draw(w1,w2,4.);
          if (which2 == 1) color(CYAN);
          else  color(YELLOW);
                        pnt(oldw1,oldw2,4.0);
                        if((count % 5) == 0){test(i);
                                             screen_two();
                                             screen_one();
                                             move(w1,w2,4.);}
                        if(check_mouse() == LEFTMOUSE) break;
                        count++;}
                        printf("start %2.2f %2.2f \n end %2.2f %2.2f ",
                        start_w1,start_w2,w1,w2);
                        initialize ();
         errors = 1.0;
}}
```

initialize

```
initialize()
      {w1 = random();
      w2 = random();
      mapcolor(2,0,128,0);
      start_w1=w1;
      start_w2=w2;
 }

/*  Feedforward backpropagation rules  */
```

feedforward

```
feedforward()
{ float temp;
  temp = w1*x + w2 *y + 1.; /* theta is one */
  output = sigmoid(temp);
   }

float deltax = 0.0 ,deltay = 0.0 ;
float olddely = 0.0 ,olddely = 0.0;
float moment= 0.0;
```

backpropagate

```
backpropagate()
    { float del;
              olddelx =  momen * deltax;
              olddely = momen * deltay;
              del = output * (1.0 - output) * (doft - output);
              deltax = neta * del * x;
              deltay = neta * del * y;
              w1 = w1 + deltax + olddelx;
              w2 = w2 + deltay + olddely;
    }

float V,Q,R,errorprime;
float A1 = 0.1, A2 =0.0 ,A3 =0.0 ,A4 = 0.1, A5 = 0.05;
float U,fprime,error,delw1=0.,delw2=0.;
```

backpropagate2

```
backpropagate2()

    {
              U = output*(1.0-output);
              V = x * (A3 * w1 + A5 * delw1) + y * (A3 * w2 + A5 * delw2);
              fprime = U * V;
              error = 2.0 * (doft - output);
              Q=U * error;
              errorprime = -2.0 * fprime;
              R=U * (errorprime+error * (1.0-2.0*output) * V);
              delw1 =  delw1 - A2 * w1
                             - A4 * delw1
                             + (A1 * Q + R )  * x;
              delw2 =  delw2 - A2 * w2
                             - A4 * delw2
                             + (A1 * Q + R )  * y;
              w1 = w1 + delw1;
              w2 = w2 + delw2;
        printf("w1 %3.2f w2 %3.2f  delw1 %3.2f delw2 %3.2 \n",
                        w1,w2,delw1,delw2);


    }
```

test

```
test(i)
int i;
{ float del;
    wrong = 0;
    x = pix;
    y = piy;
    doft = sigmoid(2.0);
    feedforward();
    del = doft - output;errors =square(del);
    system("clear");
   textport(760,1020,0,220);
```

```
    printf ("Count %d Trial %d \n",count,i);
    printf (" doit    output     error \n");
    printf ("1  %3.3f  %3.3f   %3.3f \n",doit,output, del);

    x = p2x;
    y = p2y;
    doit = sigmoid(2.0);
    feedforward();
    del = doit - output;
    printf ("2  %3.3f  %3.3f   %3.3f \n",doit,output, del);
    errors += square(del);

    x = p3x;
    y = p3y;
    doit = sigmoid(2.0);
    feedforward();
    del = doit - output;
    errors += square(del);
    printf ("3  %3.3f  %3.3f   %3.3f \n",doit,output, del);

    x = p4x;
    y = p4y;
    doit = sigmoid(-2.0);
    feedforward();
    del = doit - output;
    errors += square(del);
   printf ("4  %3.3f  %3.3f   %3.3f \n",doit,output, del);

    x = p5x;
    y = p5y;
    doit =sigmoid(-2.0);
    feedforward();
    del = doit - output;
    errors += square(del);
    printf ("5  %3.3f  %3.3f   %3.3f \n",doit,output, del);

    x = p6x;
    y = p6y;
    doit = sigmoid(-2.0);
    feedforward();
    del = doit - output;
    errors += square(del);
    printf ("6  %3.3f  %3.3f   %3.3f \n",doit,output, del);

    printf ("RMS error %3.2f \n",errors);
}

propagate()                                                    propagate
    { float del;
            del = output - doit;
             w1 -= neta * del * x;
             w2 -= neta * del * y;
     }
```

```
makeinput()
{ int which;
  which = rand() % 6;
  switch(which)

    { case 0: x = p1x;
              y = p1y;
              doft = sigmoid(2.0);
              break;
      case 1: x = p2x;
              y = p2y;
              doft = sigmoid(2.0);
              break;

      case 2: x = p3x;
              y = p3y;
              doft = sigmoid(2.0);
              break;

      case 3: x = p4x;
              y = p4y;
              doft = sigmoid(-2.0);
              break;
      case 4: x = p5x;
              y = p5y;
              doft = sigmoid(-2.0);
              break;
     case 5: x = p6x;
              y = p6y;
              doft = sigmoid(-2.0);
              break;}


 }
float random ()

{ float x;
  int y;
  y=(rand() % 4000);
  x=((float)y/4000.0-0.5);
  x = x * 4.0;
  return x;

 }
float randomx()

{ float x;
  int y;
  y=(rand() % 200);
  x=((float)y/200.0 - 0.5);

  return 2.*x;
```

```
}

init_mouse()                                                    init_mouse

{   mapcolor(1000,100,100,100);                                        270
    mapcolor(1001,100,100,200);
    qdevice(MIDDLEMOUSE);
    qdevice(LEFTMOUSE);
    qdevice(RIGHTMOUSE);}


check_mouse()                                                   check_mouse

{ short val;
  int the_return;                                                       280
  float temp,xtemp,ytemp;
  Screencoord xs,ys;
  int select;
  float xvalue;
  char message[20];

  if (qtest()!=0){            cursoff();menu();

                                xs=getvaluator(MOUSEX);
                                ys=getvaluator(MOUSEY);                 290

                              xvalue= (float)(xs-110)*0.02;
                              if (xvalue > 2.0) xvalue = 2.0;
                              if (xvalue < 0.0) xvalue = 0.0;
                              select = (int)((float)ys/20);

  switch (qread(&val)){

          case MIDDLEMOUSE: if(val==1){
                              xs=getvaluator(MOUSEX);                   300
                              ys=getvaluator(MOUSEY);
                              w2 = (float)(xs)/1048.* 2.0;
                              w1 = (float)(ys)/768.* 2.0;
                                    }
                              break;

          case LEFTMOUSE: /*if(val==1){ } */
                            break;

          case RIGHTMOUSE: if(val==1){                                  310

     if(which2 == 0) {

                    switch(select){
          case 6: meta = xvalue;
                  write_float (10.120.110,"    Eta:",meta,CYAN);break;
          case 5: momen = xvalue;
```

```
                write_float (10,100,110,"   Mom :",momen,CYAN);break;
                                    }}
        else {                                                              320
                 switch(select){
            case 0:  A1 = xvalue;
                write_float (10,180,110,"  A1 :",A1,CYAN);break;
            case 8:  A2 = xvalue;
                write_float (10,160,110,"  A2 :",A2,CYAN);break;
            case 7:  A3 = xvalue;
                write_float (10,140,110,"  A3 :",A3,CYAN);break;
            case 6:  A4 = xvalue;
                write_float (10,120,110,"  A4 :",A4,CYAN);break;
            case 5:  A5 = xvalue;                                           330
                write_float (10,100,110,"  A5 :",A5,CYAN);break;
              }}


                 switch(select){
            case 4:  which2  = 1;
                if (which2 == 0) write_string(10,80,110,"First  Order");
                else             write_string(10,80,110,"Second Order");
                menu();break;
            case 3: if(xvalue == 0.0)the_return = LEFTMOUSE;                340
                else {color(BLACK);clear(); surface(FALSE);menu();}

                 break;
            case 2:
                if(xvalue == 0.0){while(val==1) qread(&val);
                                  while(val!=1) qread(&val);}
                 else exit(0);
                        break;

                    } /* end of if */                                       360
                    } /* end of switch */

                    } /* end of if */
}

cursor();
return the_return;
}

write_float(x,y,l,title,ft,a_color)                                write_float
int x,y,l;                                                                  361
char title[20];
float ft;
int a_color;

{ char number[20];
  l = (int)((float)l+100. );
  screen_three();
  sprintf(number,"%3.3f",ft);
  color(a_color);                                                           370
```

```
  rectfi(x-5,y-5,x+1,y+15);
  color(YELLOW);
  sscale(x+50,y);
  linewidth(1);
  cmov2i(x-2,y);
  charstr(title);
  charstr(number); }

write_string(x,y,l,title)                                          write_string
int x,y,l;
char title[20];

{ char number[20];
  l = (int)((float)l/video);
  color(6);
  rectfi(x-5,y-5,x+1,y+15);
  color(4);
  linewidth(1);
  recti(x-5,y-5,x+1,y+15);
  cmov2i(x,y-2);
  charstr(title);
   }

menu()                                                             menu
{ cursoff();
if (which2 == 0){
                        write_float (10,120,110," Eta:",eta,GRAY);
                        write_float (10,100,110," Mom:",momen,GRAY);
                        write_string(10,80,110," First  Order");}
              else   {  write_string(10,80,110," Second Order");
                        write_float (10,180,110," A1 :",A1,GRAY2);
                        write_float (10,160,110," A2 :",A2,GRAY2);
                        write_float (10,140,110," A3 :",A3,GRAY2);
                        write_float (10,120,110," A4 :",A4,GRAY2);
                        write_float (10,100,110," A5 :",A5,GRAY2);}
                        write_string(10,60,110," Next     Clear");
                        write_string(10,40,110," Halt     Quit ");
cursoff();
}

sscale(x,y)                                                        sscale
int x,y;

{ int i;
  x = x + 50;
  linewidth(2);
  color(YELLOW);
  loopi(10){
          move2i(x+i*10,y+5);draw2i(x+i*10,y);draw2i(x+10+i*10,y);}
          draw2i(x+i*10,y+5); }
```

```
/*****************************************************************
*
*       DATE: 3 October 1988
*       VERSION: 2.0
*
*       NAME: Surface.c
*       MODULE NUMBER:
*       DESCRIPTION: Surface Generator for Error_surc
*       ALGORITHM: None
*       PASSED VARIABLES: None                                    10
*       RETURNS: None
*       GLOBAL VARIABLES USED: None
*       GLOBAL VARIABLES CHANGED: None
*       FILES READ: None
*       FILES WRITTEN: None
*       HARDWARE INPUT: None
*       HARDWARE OUTPUT: None
*       MODULES CALLED: None
*       CALLING MODULES: None
*                                                                 20
*       AUTHOR: Gregory L. Tarr
*       HISTORY:
*
*****************************************************************/
/*  Beginning of Second file component error.c  */

#include<gl.h>
#include<device.h>
#include<math.h>
#define size 50                                                   30
#define hello(a)  printf("hello %d\n",a);
#define start (5.0)
#define incre (-2.0* start/size)

float getx();
float sigmoid();
float the_function();
float square();
float max,min;
float array[size][size];                                          40

float p1x,p1y,p2x,p2y,p3x,p3y,p4x,p4y,p5x,p5y,p6x,p6y;


/*
float height(x,y)

  float x,y;
{ float z;
  int i,j;                                                        50
  i = (int)( x / 1.8 - 20.);
  j = (int)( y / 1.8 - 20.);
  z = array[i][j];
```

```
  return 1;
}
*/
#define TABLE 512
#define up (0.50 * incre)

int getcolor();

surface(fast)
int fast;
{   float x,xn,yn,y;
    int i,j,k;
    Coord parray[4][3];
    Colorindex iarray[4];
    cursoff();
if(fast == TRUE){
    fill_points();
    for (i=0;i<size;i++,x=getx(i)){
                        for (j=0;j<size;j++,y=getx(j)){
                                array[i][j]= (the_function(x,y));

                                        }}
findmax(array);

 ginit();
  gconfig();
   for(i=TABLE/2,k=0,j=8;j<TABLE;j++)
       { if (j<256) {i--; mapcolor(j,0,128-i/2,i);}
         if  (j>255) {k++; mapcolor(j,k,128-k/2,0);}}
 color(0);
 clear();

   color(BLACK);
   clear();}
   screen_one();
   zbuffer(TRUE);
   zclear();
   drawfloor();
   for (i=1;i<size-1;i++){x=getx(i);xn=getx(i+1);
                        for (j=1;j<size-1;j++){y=getx(j);yn=getx(j+1);

                                        parray[0][0] = x;
                                        parray[0][1] = y;
                                        parray[0][2] = array[i][j];
                                        iarray[0]=getcolor(array[i][j]);

                                        parray[1][0] = xn-up;
                                        parray[1][1] = y;
                                        parray[1][2] = array[i+1][j];
                                        iarray[1]=getcolor(array[i+1][j]);
```

```
                parray[2][0] = xn-up;
                parray[2][1] = yn-up;
                parray[2][2] = array[i+1][j+1];
                iarray[2]=getcolor(array[i+1][j+1]);


                parray[3][0] = x;
                parray[3][1] = yn-up;
                parray[3][2] = array[i][j+1];
                iarray[3]=getcolor(array[i][j+1]);
                spif(4,parray,iarray);

                color(CYAN);
 }}

 zbuffer(FALSE);
 cursoff();
 }
 fill_points()                                                    fill_points
 {
 p1x=0.5;
 p1y=2.5;

 p2x=p1x + randomx();
 p2y=p1y + randomx();

 p3x=p1x + randomx();
 p3y=p1y + randomx();


 p4x=3.0;
 p4y=1.0;

 p5x=p4x + randomx();
 p5y=p4y + randomx();

 p6x=p4x + randomx();
 p6y=p1y + randomx();


 }
 getcolor(z)                                                      getcolor
 float z;
 {
    Colorindex light;
    light = (Colorindex)((TABLE-20)*(z-min)/(max-min));

    return (light+8);
 }
 normal(x,y,z)                                                    normal

 float *x,*y,*z;
```

```
{ float mag,a,b,c;
     a = *x;                                                    160
     b = *y;
     c = *z;

      mag = a*a+b*b+c*c;
      mag = (float)sqrt((double)mag);
      *x = a / mag;
      *y = b / mag;
      *z = c / mag;
}                                                               170


float getx(i)
      int i;
{ float x;
  x = incre *(float)i;
  x = x + start;

  return x;

}                                                               180
float sigmoid(y)
 float y;
{
  return( 1.0/(1.0 + (float)exp(-(double)(y))));
}
float square (x)
float x;
{ float y;
  y = x * x;
  return y ;}                                                   190
#define a -2.


float the_function(x,y)

float x,y;

{ float err,err1,err2,err3,err4,err5,err6;

   err1 = square(sigmoid(1.) - sigmoid(1.+ p1x * x + p1y * y));  200
   err2 = square(sigmoid(1.) - sigmoid(1.+ p2x * x + p2y * y));
   err3 = square(sigmoid(1.) - sigmoid(1.+ p3x * x + p3y * y));
   err4 = square(sigmoid(-1.) - sigmoid(1.+ p4x * x + p4y * y));
   err5 = square(sigmoid(-1.) - sigmoid(1.+ p5x * x + p5y * y));
   err6 = square(sigmoid(-1.) - sigmoid(1.+ p6x * x + p6y * y));
   err = err1 + err2 + err3 + err4 + err5 + err6;
   return (2 * err);
}
drawfloor()                                                     drawfloor
{  color (BLUE);                                                210
```

```
color(CYAN);
move(-5.,-5.,0.);
draw(-5.,5.,0.);
draw(5.,5.,0.);
draw(5.,-5.,0.);
draw(-5.,-5.,0.);

move(-6.,0.,0.);
draw(6.,0.,0.);                                                    220

move(0.,-6.,0.);
draw(0.,6.,0.);

move(0.,0.,-2.);
draw(0.,0.,5.);

cmov(6.01,0.,0.);charstr("X");
cmov(0.,6.01,0.);charstr("Y");
cmov(0.,0.,6.01);charstr("Z");                                     230
}
findmax(array)                                                     findmax

#define loopi(ON) for(i=0;i<ON;i++)
#define loopj(ON) for(j=0;j<ON;j++)
#define loopk(ON) for(k=0;k<ON;k++)

float array[][size];
{ int i,j,k;
  int maxi=0,maxj=0;                                               240
  int mini=0,minj=0;
 min = array[0][0];
 max = array[0][0];
loopi(size){
  loopj(size){     if (array[j][i]< min) min=array[j][i];

                   if (array[j][i]> max) max=array[j][i];
                }
              } printf ("%f %f \n",max,min);
                                                                   250
}
screen_one()                                                       screen_one
{
  ortho(0.0,(float)XMAXSCREEN,0.0,(float)YMAXSCREEN,-1.0,1.0);
  setdepth(0x000,0xF00);
  viewport(0,(int)(1023.*video),0,(int)(787.*video));
  perspective(150,1.0,2.0,-2.0);
  lookat(2.,0.,50.,0.,0.,0.,0.0);

}                                                                  260
screen_two()                                                       screen_two
{ int i;
  float left,right,top,bottom;
  left= -1.;right = 5.;bottom = -1.;top = 5.;
```

```
    ortho2(left,right,bottom,top);
    cursoff();
    color(1009);
    viewport((int)(700.*video),(int)(900.*video)
                    ,(int)(500.*video),(int)(700.*video));
    clear();                                                          270
    color(YELLOW);
    for(i= -3;i<4;i++)
    move2(left,i);draw2(right,i);
    for(i= -3;i<4;i++)
    move2(i,bottom);draw2(i,top);
color(BLUE);
    cmov2(p1x,p1y);charstr("*");
    cmov2(p2x,p2y);charstr("*");
    cmov2(p3x,p3y);charstr("*");
color(BLACK);                                                         280
    cmov2(p4x,p4y);charstr("*");
    cmov2(p5x,p5y);charstr("*");
    cmov2(p6x,p6y);charstr("*");
color(7);
move2(0.0,-0.5/w2);
draw(5000.,(-(w1/w2*5000.) - 0.5/w2));
move2(0.0,-0.5/w2);
draw(-5000.,((w1/w2*5000.) - 0.5/w2));

curson();                                                             290
}

screen_three()                                                  screen_three
{
    ortho2(0.,1023.*video,0,767.*video);
    viewport(0,(int)(1023.*video),0,(int)(767.*video));

}
```

Bibliography

1. Barmore Gary *Speech Recognition Using Neural Nets and Dynamic Time Warping.* MS thesis, AFIT/GEO/ENG/88D-1. Air Force Institute of Technology (AU), December 1988 (AD-A177598).
2. Baum, Eric B. "On the Capabilities of Multilayer Perceptrons" AIP Conference Proceedings 151: *Neural Networks for Computing* 53-58 (1986)
3. Baum, Eric B. "Supervised Learning of Probability Distributions by Neural Networks" *AIP Conference Proceedings: NeuralNetworks for Computing* 1-8 (1987)
4. Born M., Wolf E. *Principles of Optics* 2nd rev. ed. MacMillan, New York (1964).
5. Caudill, Maureen. "Neural Networks Primer", *AI Expert*, 47-52 (December 1987).
6. DeSieno Duane, *Adding a Conscience to Competitive Learning*, IEEE ICNN (1988)
7. Grossberg, S. *The Adaptive Brain: Cognition, Learning, Reinforcement, and Rhythm.* Amsterdam: Elsevier (1986)
8. Hamacher, Carl V. and others. *Computer Organization.* New York: McGraw-Hill Book Company, (1984)
9. Hopfield J.J. "Computing with Neural Circuits: A Model" *Science*, 233: 625-633 (August 1986)
10. Huang, W. Y. and Lippmann, R. P. "Neural Net and Traditional Classifiers", *Proceeding of the Conference on Neural Information Processing Systems.* Denver (November 1987)
11. Hecht-Nielsen, Robert *Counter Propagation Networks* IEEE ICNN (1987)
12. Kohonen T. *Self-Organization and Associative Memory.* (Second Edition) Berlin: Springer-Verlag Series In Information Sciences, (1987)
13. Kohonen T. "Automatic Formation of Topological Maps of Patterns in a Self-Organizing System" *Proc. 2nd Scand.Conf. on Image Analysis* 214-220 (June 1981).
14. Kuczdwski, Robert M.*Exploration of Backward Error Propagation as a Self-Organization Sturcture* IEEE ICNN (1987)
15. Kung S.Y.,*An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Backpropagation Learning* IEEE ICNN (1987)
16. Lippmann,Richard P. "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine.* 4-22 (April 1987)

17. Ruck, D. W. *Multisensor Target Detection and Classification.* MS thesis, AFIT/GE/ENG/86D-20. Air Force Institute of Technology (AU), December 1986 (AD-A177598).

18. Roggemann, M. Personal Conversation and Papers (August 1988)

19. Se,nowski Terrence J. "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" *Neural Networks*, (January 1988)

20. Sietsma, Jocelyn "Neural Net Pruning - Why and How" *Proceedings of the IEEE International Conference on Neural Networks* (1988)

21. Sun, G.Z. "A Novel Net That Learns Sequential Decision Process" *AIP Proceedings*, (1986).

22. Stright James R. *A Neural Network Implementation of Chaotic Time Series Prediction.* MS thesis, AFIT/GE/ENG/88D-50. Air Force Institute of Technology (AU), December 1988 (AD-A177598).

23. Troxel, Steven E. *Position, Scale, and Rotation Invariant Target Recognition Using Range Imagery.* MS thesis, AFIT/GE/ENG/86D-20. School of Engineering, Air Force Institute of Technology (AU),Wright-Patterson AFB OH, December 1986 (AD- A177598).

24. Valiant L.G. "Learning Disjunctions of Conjunctions" *Proceedings of the Ninth International Conference on Artificial Intelligence.* (1983)

25. Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* Harvard University Dissertation. 1974

26. Woods, D. *Back and Counter propagation Aberrations* IEEE ICNN (1987)

Vita

Captain Gregory L. Tarr [redacted]

Capt Tarr entered the Air Force in December of 1978 as an Aerospace Ground Equipment Technician. After serving two years at Castle AFB, California with the 84th Fighter-Interceptor Squadron, he entered the Airman's Education and Commissioning Program completing a Bachelor of Science in Electrical Engineering at the University of Arizona in December 1983. Capt Tarr served three year with the Aeronautical Systems Division at Wright-Patterson AFB, Ohio before entering the School of Engineering, Air Force Institute of Technology in June 1987. [redacted]

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-54	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Dynamic Analysis of Feedforward Neural Networks Using Simulated and Measured Data

12. PERSONAL AUTHOR(S)
Gregory L. Tarr, B.S.E.E., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ____ TO ____	1988 December	281

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	09		Artificial Neural Networks; Artificial Intelligence; Pattern Recognition

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Steven K. Rogers, Capt, USAF
Associate Professor of Electrical Engineering

Approved for release in Accordance with AFR 190-1
12 Jan. 1989

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Steven K. Rogers, Capt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-6027
	22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

ABSTRACT

An environment is developed for the study of dynamic changes in patterns of weight and node values for artificial neural networks. Graphic representations of neural network internal states are displayed using a high resolution video terminal. Patterns of node firings and changes in weight vectors are displayed to provide insight during training. Four pattern recognition problems are applied to four types of artificial neural networks. Using simulated data, a simple disjoint region classification problem is developed and examined using a Kohonen net and a multilayer feedforward back propagation (MFB) network.

A MFB neural network is also used to simulate a Fourier filter. Using a Kohonen net, a MFB, a counterpropagation and a hybrid network, data measured from infrared and laser radar imagery of military vehicles is analyzed. The accuracy and training times for a MFB net and a Hybrid net are compared using an ambiguous decision region problem. Each classification problem is examined and compared to classical, nearest neighbor pattern recognition techniques. Using dynamic analysis, neural network pruning is used to determine optimum node configurations. A hybrid neural network is developed using Kohonen training rules for the first hidden layer followed by one or two hidden layers using standard back propagation rules for training. Advantage of the hybrid network is shown for classification problems involving anomalies characteristic of measured data. The Hybrid network requires less training and fewer interconnections than MFB when classifications involves ambiguous decision regions.