Counterpropagation neural network detection of visual primitives

Cynthia Lynn Johnson
cynthiajohnson@knights.ucf.edu

COUNTERPROPAGATION NEURAL NETWORK DETECTION OF VISUAL PRIMITIVES

BY

CYNTHIA LYNN JOHNSON
B.S.E.E., University of Miami, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1990

ABSTRACT

Psychological testing has shown that there is an early preattentive stage in the human visual system. At this level, simple features and properties of objects known as visual primitives are detected spatially in parallel by groupings of cells in the visual cortex known as feature maps. In order to study this preattentive stage in a machine vision system, the biologically inspired, highly parallel architecture of the artificial neural network shows great promise. This paper describes how the unique architecture of the counterpropagation neural network was used to simulate the feature maps which detect visual primitives in the human visual system. The results of the research showed that artificial neural networks are able to reproduce the function of the feature maps with accuracy. The counterpropagation network was able to reproduce the feature maps as theorized, however, future research might investigate the abilities of other neural network algorithms in this area. Development of a method for combining the results of feature maps in a simulation of full scale early vision is also a topic for future research that would benefit from the results reported here.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
PREATTENTIVE VISION

The concept of a machine that can identify and recognize objects as quickly and easily as humans can is an ambitious goal, but not an impossible one. The human visual system is one of the most complex neural assemblies known. Although it is not yet completely understood, psychologists and physiologists have obtained a great deal of insight into the structure and function of human vision. In all likelihood, the theories and principles of the biological vision system will provide the best starting point in the development of a machine visual system capable of equal performance. In this research, an attempt was made to understand a portion of the biological vision system and reproduce its function on a machine in an effort to develop a more thorough understanding of complex visual mechanisms.

From biological studies, it is clear that an image is registered by cells within the eye and transmitted to the brain by the optical nerve. In a computer based system, a camera can digitize a scene and convert it into a collection of signals which are transmitted to the computer. The next step is analyzing and accurately recognizing the component objects within the scene. The human recognition system seems to have a striking dichotomy at this point. Many discriminations appear to be made automatically without attention and spatially in parallel, while other discriminations require focused attention or scrutiny. These two types of processing were originally attributed to different levels by Neisser (Neisser 1967). He identified an early or preattentive stage where simple features were registered determining texture segmentation and figure ground groupings. This preattentive stage is separate from a second, attentive stage where focused attention recognizes specific objects within a complex grouping of objects. The understanding of and recreation of a portion of the early visual stage was the primary goal of the research presented here.

There are two types of psychological evidence that support the concept of preattentive processing. One is textural segmentation and the other is visual search. Textural segmentation is the division of an image into segments based on the texture of its component parts. Figure 1 shows an image which is easily partitioned into two segments. The zeros constitute one segment and the other is comprised of ones. Julesz (Julesz 1981) has proposed that texture segmentation is preattentively processed using simple features called textons. When effortless texture segregation occurs, it is because the two segments do not contain the same type of texton. He defines the texton classes as color, elongated blobs of specific widths, orientation and aspect ratios, and the terminators of these blobs.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 1. An Example of Texture Segmentation.

A second source of evidence that supports the early vision theory is visual search. When subjects are asked to identify a target object in displays containing a varying number of distracters, the target appears to "pop-out" of the image when the target is defined by a simple visual feature (Treisman 1985). Figure 2 illustrates this "pop-out" phenomenon. The circular zero among the distracters of ones appears to "pop-out" at the observer. The speed of target detection in these cases suggests spatially parallel processing at the stage prior to attentive vision. This conclusion supports the theory of early vision. Treisman and Gelade have developed an explanation known as feature integration theory (Treisman and Gelade 1980). This theory states that primitive elements are directly sensed by specialized populations of detectors called feature maps. Each feature map responds to a particular feature, and all maps operate in parallel. It is only when attention is focused on the results of the feature maps that location and identity of an object is obtained. This explains why an object that differs from its surrounding distracters by only one simple feature will appear to "pop-out". Attention is focused on the results of that one particular feature map, and when activity in that map signals the presence of the object, there is no need to combine the results of one map. An example is the "pop-out" of a red circle among blue dis tracters. The circle would be the only object causing activity on the "red" feature map making identification easy. "Pop-out" does not occur when more than one feature map is involved. For example, the red circle would not "pop-out" of a background of red squares and blue circles.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 2. An Example of Pop-out Phenomenon.

Although the feature integration theory is not meant to equate feature detectors with single neural units (Treisman and Patterson 1984), there is biological evidence to support the theory of neural networks in the brain devoted to the parallel detection of features. The projection of the retina has been plotted to several areas within the brain of cats and monkeys, and electrical measurements of brain activity have shown similar results in humans (Cowey 1979). Although the exact number of visual areas in the human brain is not known, thirteen have been found in the cat (Tusa, Palmer, and Rosenquist 1975) and it is unlikely that the human brain has fewer areas. In experiments on live cats and monkeys, Hubel and Wiesel (Hubel 1988) were able to map activity in neural cells to a particular line orientation . These biological facts give much credence to the theory of maps that detect simple features in parallel whose results are combined for object recognition.

Once we accept that preattentive vision exists, some questions arise. The most obvious is the question that asks which features and properties constitute the visual primitives detected by feature maps. There are some features which are generally agreed to be visual primitives (Beck and Ambler 1972, Beck, Prazdny, and Rosenfield 1983, Treisman and Patterson 1984) These include line orientation, color, curvature, and blobness or closure. Other candidates which have had some doubt cast upon them include intersection, juncture, number and connectedness (Treisman 1985). It is also interesting to note that new perceptual units may be established through extensive practice so that, for example, an arbitrary set of letters can come to be detected automatically in search (Schneider and Sheffrin 1977). However, there is probably some built-in neural constraints making some physical properties or conjunction of properties difficult to detect preattentively and in parallel (Treisman and Gelade 1980).

The question of how to implement and use feature maps using a computer arises. Traditional computer architectures are serial and deterministic in nature. A single complex processor fetches and executes instructions from memory locations. Data is also stored in memory locations. The same data can be applied to the same program and the same results achieved. Pattern recognition tasks often take a considerable number of time steps to execute. Obviously, the traditional Von Neumann architecture is unsuited to a parallel task such as the detection of visual primitives. However, it will be shown that the biologically inspired architecture of artificial neural nets are a good candidate for this task.

CHAPTER 2
ARTIFICIAL NEURAL NETWORKS

Artificial neural systems are an area in which recent breakthroughs in algorithms and VLSI technology have enabled renewed interest. These systems, also called neural networks, connectionist systems, and neurocomputers, are composed of many simple processing elements that do little more than take a weighted sum of their inputs. In a neural system, a large number of elements are highly interconnected giving neural networks a parallel structure very unlike serial computers.

The architecture of artificial neural networks (ANN) is biologically inspired. The processing elements perform in a fashion similar to the elementary functions of the biological neuron that is the computing element of the cognitive systems of higher life forms. The elements of an ANN are connected in a manner that may or may not be related to the anatomy of the brain. The processing power of this architecture is a result of its massive parallelism and interconnections. Although the actual "intelligence" of the most sophisticated neural network is extremely limited, they do exhibit a surprising number of the characteristics of biological neural systems. It has been shown that ANN's can learn from experience, generalize from previous examples to new ones, and extract essential characteristics from inputs containing irrelevant data. An example of this is the Hopfield net(Hopfield 1982). In this work, the network was shown to be capable of learning the visual patterns of numerical digits during a training session. The network was then able to reproduce the digit when presented with a corrupted version. Neural networks have proved enormously useful in solving problems in many areas that have traditionally proved overwhelming. These tend to be the type of problems humans solve easily. Like humans, they also have limitations. Both neural networks and many humans seem to have difficulty in performing unaided precise numerical calculations. Therefore, it seems obvious that the role of artificial neural systems is not to eclipse traditional computer systems, but rather to work with them. The power of these two systems working together should prove greater than the sum of the two individual systems.

There are many different types of artificial neural networks, and their differences are determined by the type of node algorithm they process. The concept behind artificial neural networks is the biological neural system. The nervous system is built of cells called neurons and is extremely complex. In humans, the nervous system contains an estimated 10^{15} transmission paths known as interconnections. The neurons are similar to each other,

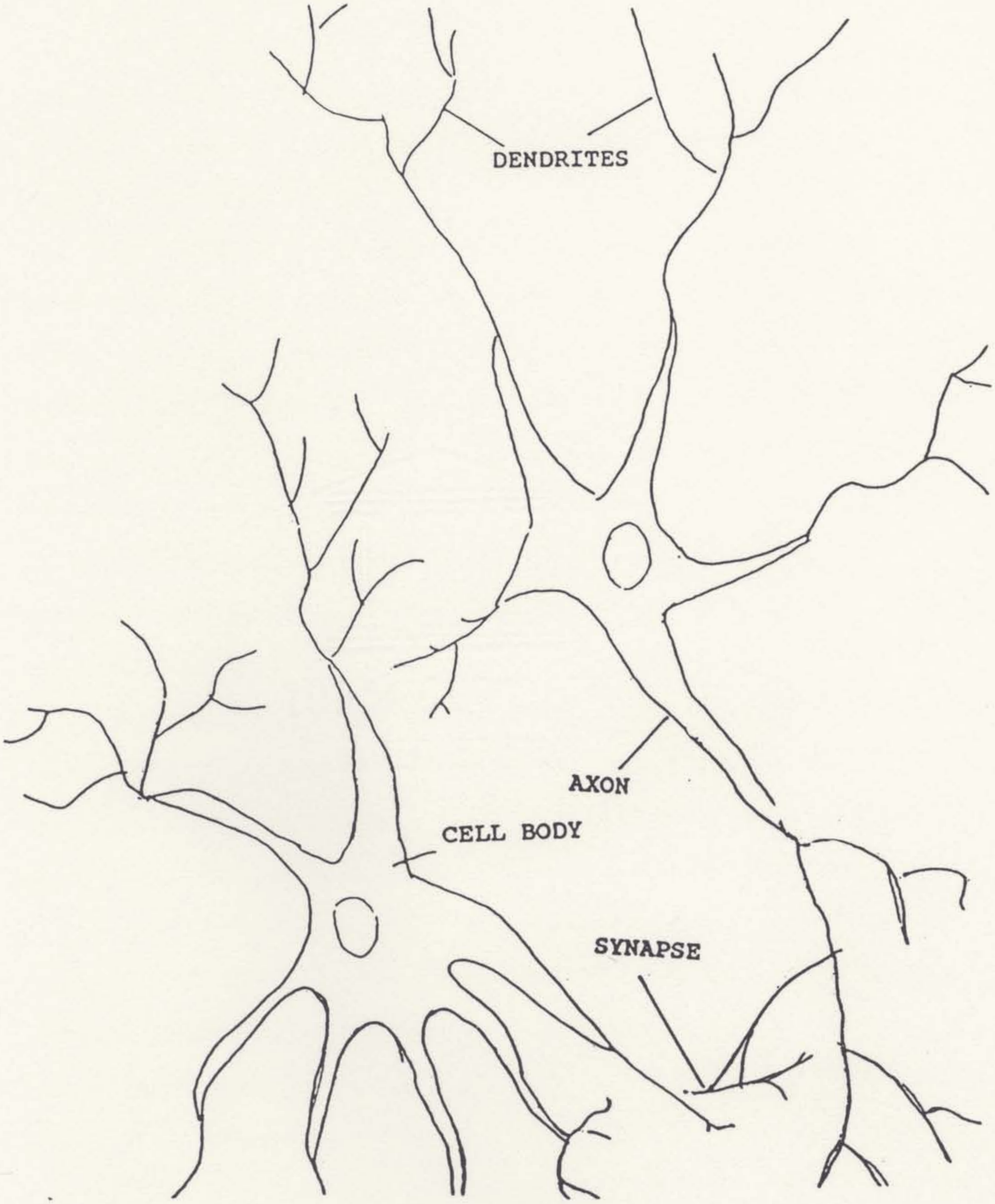

Figure 3. The Biological Neuron.

but each has its own unique capabilities to receive, process, and transmit electrochemical signals over the neural interconnections.

A typical biological neuron is shown in Figure 3. Dendrites connect neurons to each other. They receive input signals at a point known as a synapse and transmit them to the cell body. There the signals are weighted and summed. When the sum of the inputs exceed the threshold, the neuron fires, sending as signal down the axon to other neurons. This is a simplified explanation of the function, but most artificial neural systems model only these simple characteristics.

The neuron or node used in artificial neural networks is typically nonlinear, analog, and may be slow compared to modern digital circuitry. The simplest node sums N weighted inputs and passes the results through a nonlinearity. Figure 4 shows three common types of nonlinearities; hard limiters, threshold logic elements and sigmoidal nonlinearities. More complex nodes may use temporal integration and other types of time dependencies and more complex mathematical operations than summation.

Artificial neural networks can be differentiated by the type of node processing or algorithm used in the network. Neural nets are also distinguished by whether they accept binary or continuous valued inputs. They can also be

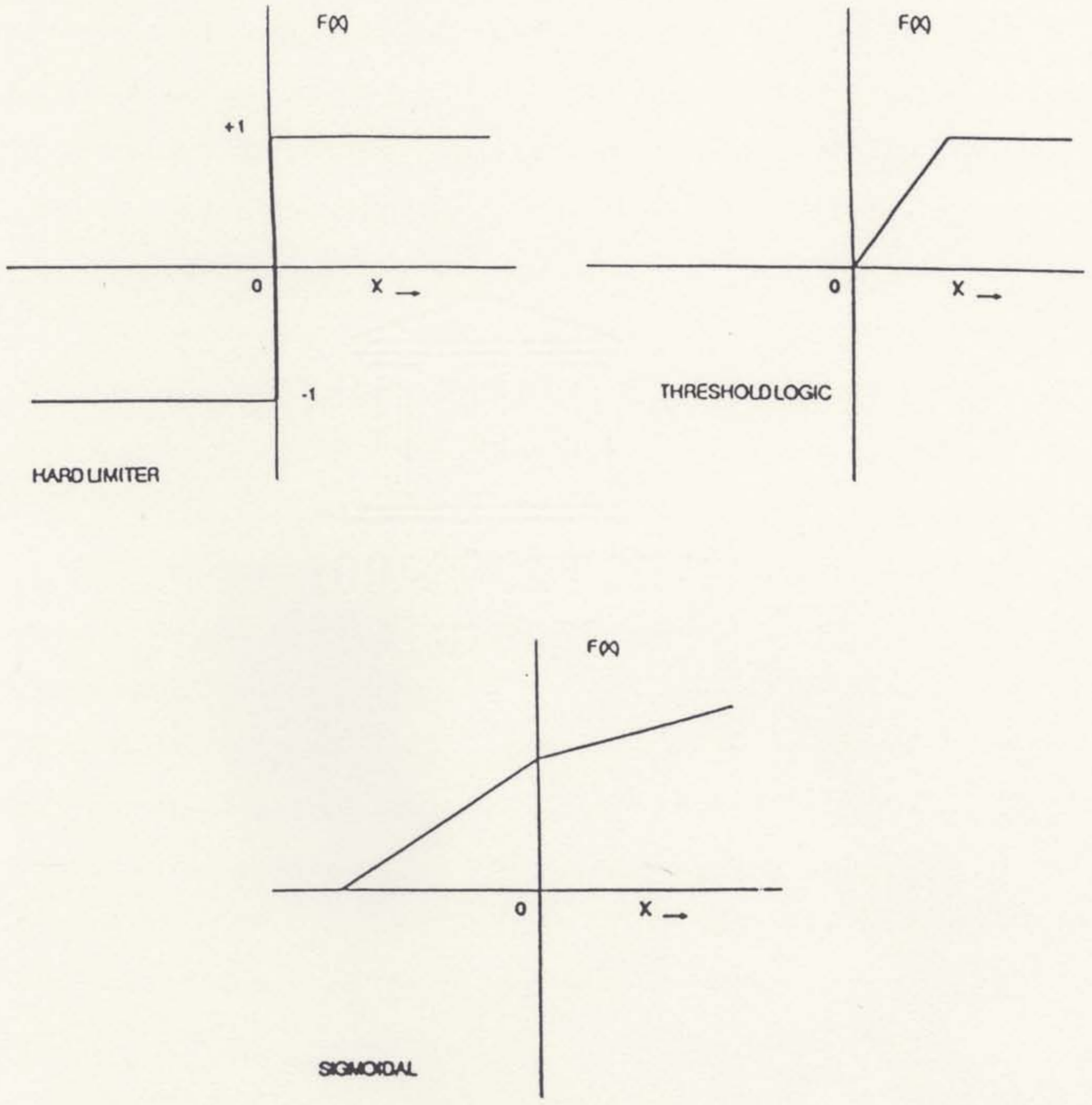

Figure 4. Common Types of Non-linearities Used in Neural Networks.

separated between those trained with or without supervision. Those trained with supervision have input and the desired output presented at training time. These types of nets are most often used as associative memory or classifiers. Nets trained without supervision have no information about the correct class provided at training time. Unsupervised nets are generally used as vector quantizers or to form clusters. Nets can be further categorized by whether or not they support adaptive training. Most nets do support adaptive training, but others use fixed weights during operation.

One of the earliest types of neural networks developed is the perceptron (Minsky and Papart 1969). This is a single layer network which can accept either binary or continuous valued inputs. Figure 5 shows a model of a perceptron network. It generated much interest when developed because of its ability to learn simple patterns. The perceptron accepts weighted inputs, sums them, and compares them to a threshold. If the sum exceeds the threshold, the node sets its output high. Otherwise, the output is set low.

The perceptron learns in a supervised mode and its learning procedure is fairly simple. First, the weights and threshold are initialized to small random non-zero values, then an input vector is presented along with the desired

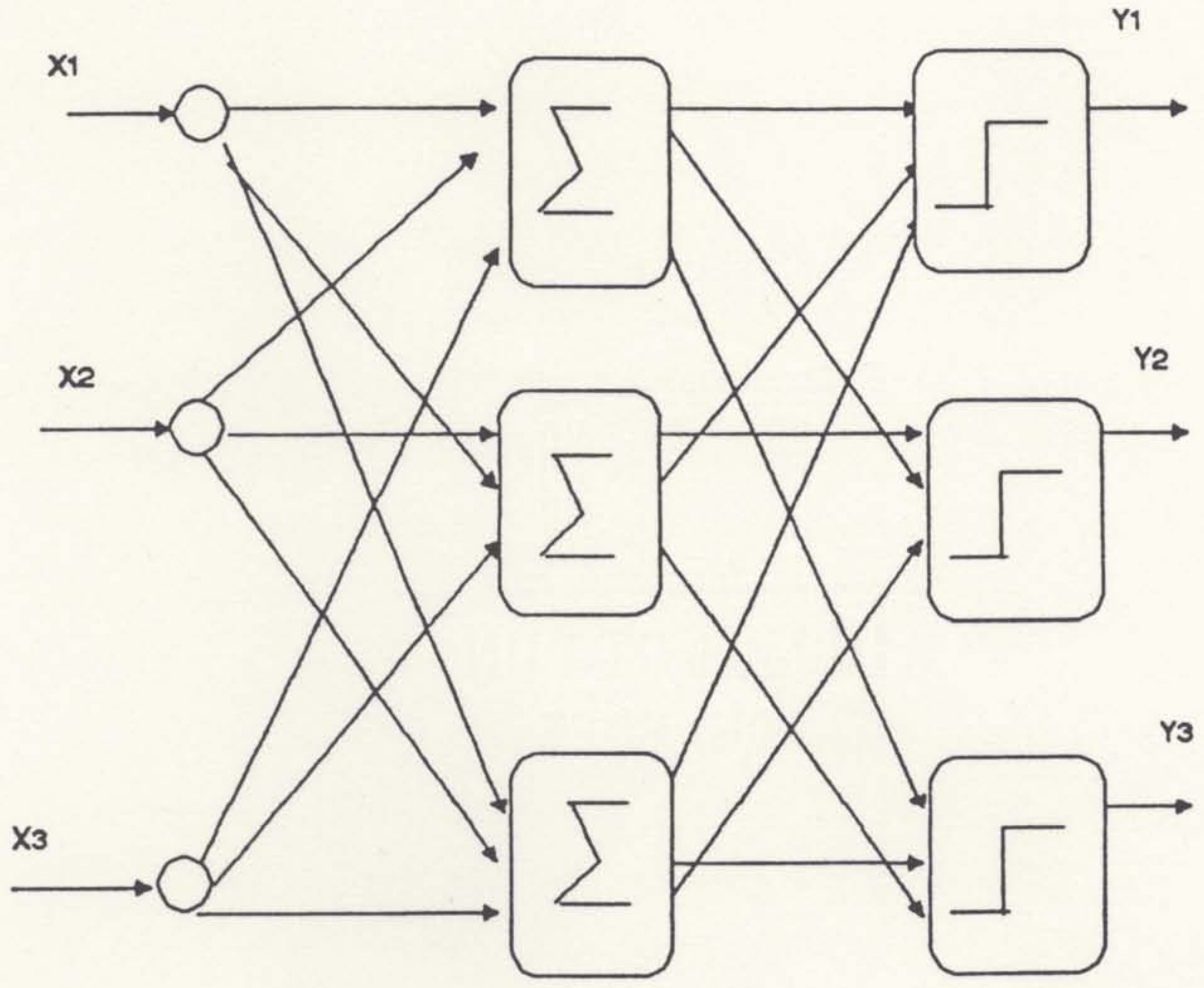

Figure 5. Perceptron Network.

output. The actual output is then calculated by multiplying each input by its associated weight and summing the results of the multiplication. The threshold values are then subtracted from the results giving the output. The weights are then adapted until the desired output is achieved. Then a new output and input can be presented and the weights again adapted until all training sets have been presented. A number of different methods have been developed to adapt the weights allowing the perceptron to converge to the proper output. One important algorithm that can be used for binary or continuous valued inputs is the delta rule. The weights are adjusted using the equation:

$$w_i(t+1)=w_i(t)+B(d(t)-y(t))x_i(t)$$

where $w_i(t+1)$ is the weight associated with input x_i after adjustment, $w_i(t)$ is the current weight associated with input x_i, $d(t)$ is the desired output of the node, $y(t)$ is the actual output of the node, and $x_i(t)$ is the ith input of vector X. Modifications to this algorithm can be made to develop a Least Mean Squared (LMS) solution or a Gaussian classifier.

Although the perceptron was innovative when introduced, the limitations of the single layer design nearly ended research into artificial neural networks. Research activity

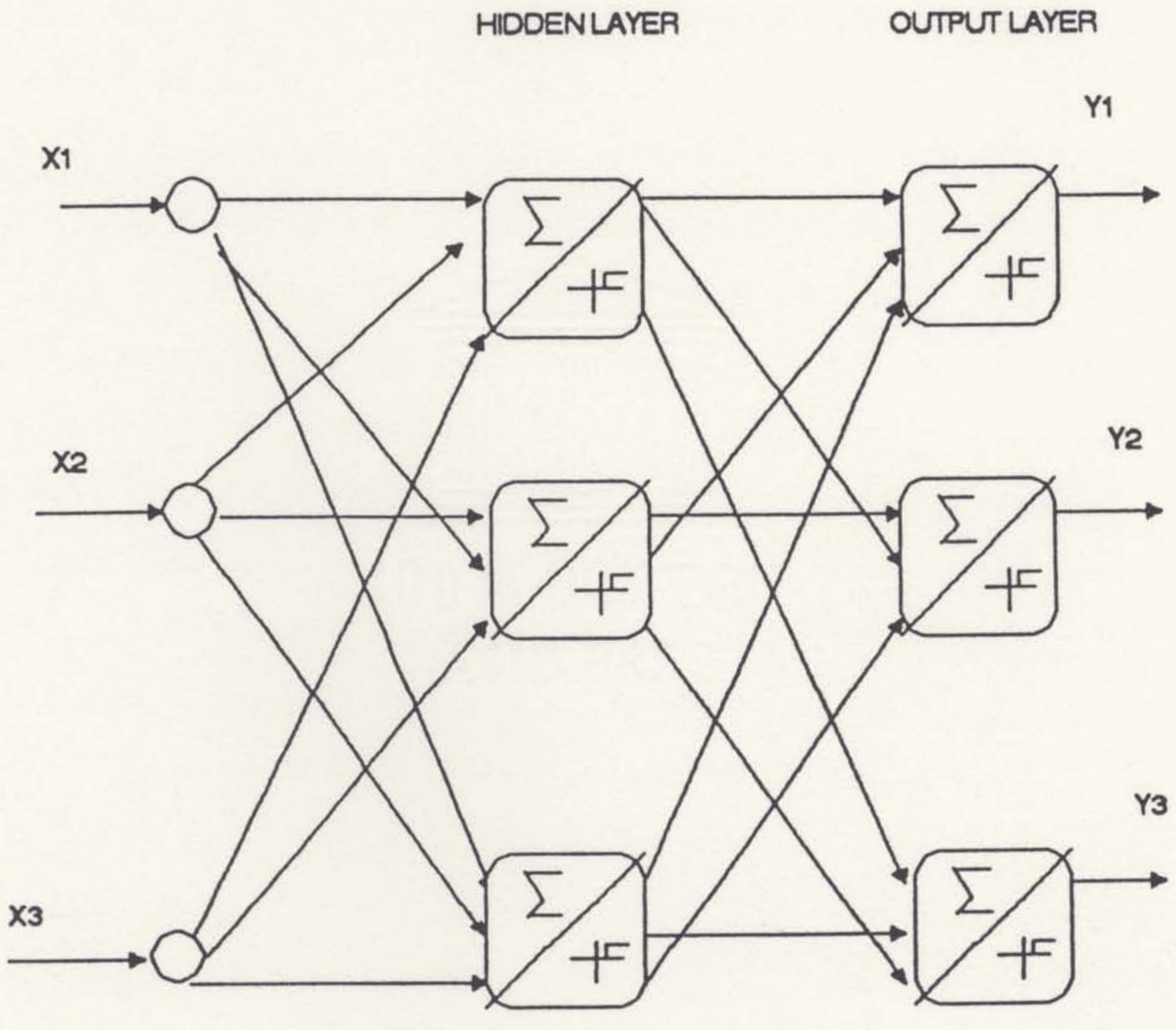

Figure 6. Two Layer Backpropagation Network.

increased only when training algorithms for multilayer nets were developed.

The development of the backpropagation algorithm (Rumelhart, Hinton, and Williams 1986) played a large part in the resurgence of interest in artificial neural networks. The algorithm is a generalization of the LMS algorithm of the perceptron. It uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and actual net outputs. Figure 6 shows an example of a two layer network. Although backpropagation works on any number of layers, only two are needed to demonstrate the algorithm. Like perceptrons, backpropagation networks learn in the supervised mode.

The weights of the neurons are initially set to small random non-zero values. The network is fully connected, that is , each neuron has a connection to each node or input in the previous and subsequent layer. Training begins by applying the input vector to the network. The network then calculates its output. This is compared to the desired output provided. Then the weights of the network are adjusted. First, the output layer weights are adjusted using the following equation:

$$w_{ij}(t+1)=w_{ij}(t)+NS_jx_i$$

where w_{ij} is the weight from hidden node i to node j at time t, x_i is the output of node i, N is a learning coefficient which can be any value in the range $0 < N < 1.0$, S_j is an error term. For output nodes, S_j is calculated by the following equation:

$$S_j=y_j(1-y_j)(d_j-y_j)$$

where y_j is the actual output and d_j is the desired output of node j. Upon completion of the weight calculations for the output nodes, the connecting hidden layer nodes' weights are updated. This is done using the first equation with the S_j term begin calculated with the following equation:

$$S_j=x_j(1-x_j)\text{Summation }_k s_k{*}w_{jk}$$

where k is over all nodes in the layer above node j. The error is propagated back through the network in this manner which led to the development of the name backpropagation.

The backpropagation network is currently one of the most popular networks in use, because of its versatility. It has been proved effective in a number of application from exclusive-OR to speech synthesis and recognition. However, it does have the disadvantage that in some cases the number of presentations of training data has been large (more than 100 passes through all the training data). Although algorithms have been developed to help speed convergence, it seems unlikely that training can be speeded up significantly. As mentioned before, the backpropagation algorithm is a gradient descent algorithm with the associated pitfall that it is quite possible for the algorithm to fail to provide a correct solution to the problem.

The learning algorithm of the perceptron and backpropagation networks is a mathematical one and only loosely based on actual biological systems. Many researchers are looking for more biologically based learning systems. This leads to the discussion of some unsupervised learning algorithms, such as the Kohonen self-organizing map.(Kohonen, 1988) Self-organization refers to the ability of the network to learn without being supplied the correct answer. Self-organizing networks are generally closely modeled after neurobiological systems and often are the result of researchers attempting to understand how the brain works.

When initially compared to the networks already discussed, the Kohonen network is deceptively simple. It is not a heirarchial system and consists of a single layer of nodes. However, it contains interconnections among the nodes within the single layer which do not exist in supervised learning networks. Each node contains a weighted connection to all of the other nodes within the network. Also, the initial weight vectors of a Kohonen network must be normalized to a constant fixed length, usually one, so that the weight vectors are normalized vectors pointing in random directions about the unit circle.

When inputs are applied to the Kohonen layer, each node computes the dot product of its weight vector with the

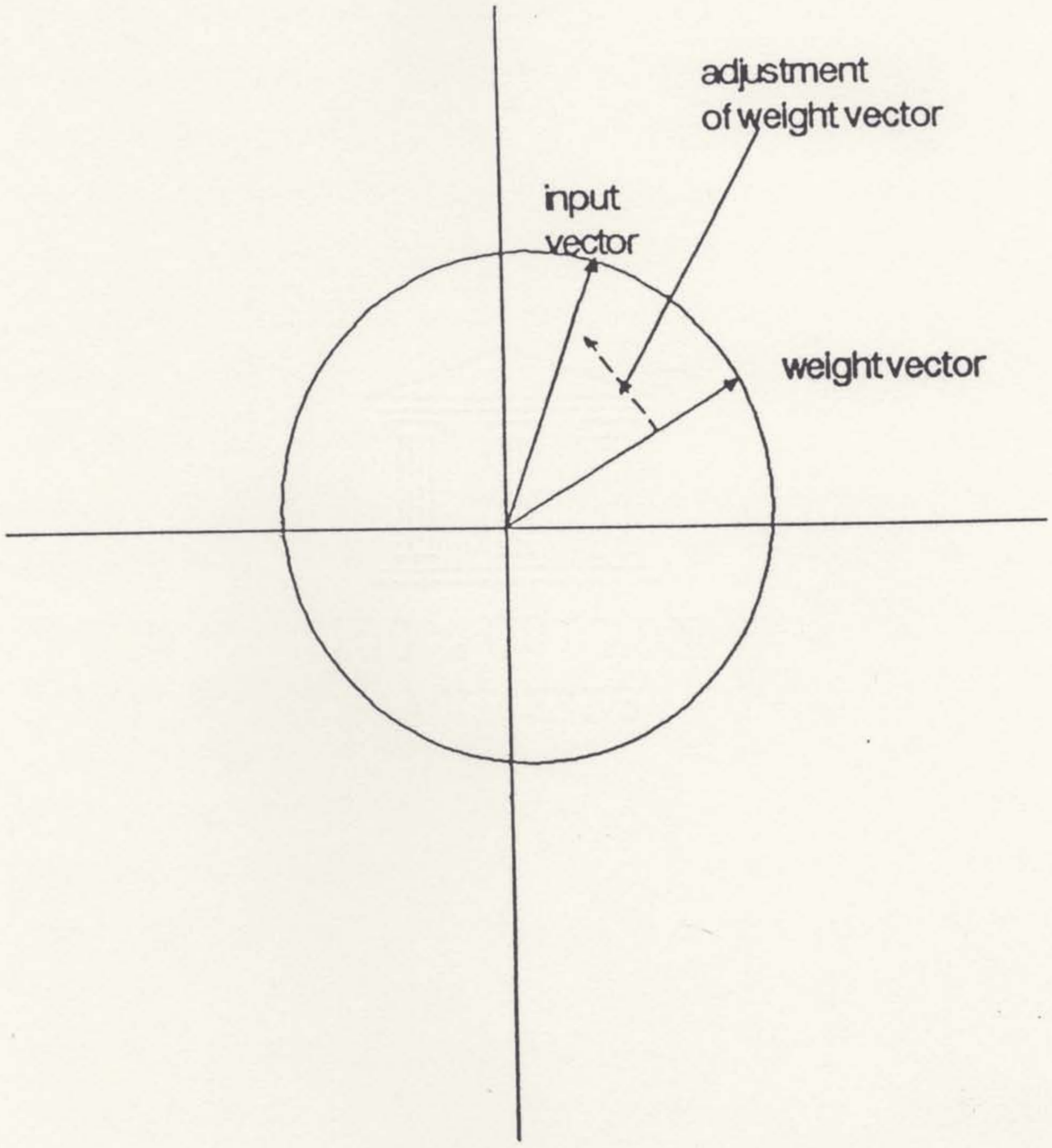

Figure 7. An Example of Kohonen Learning on the Unit Circle.

input vector. The dot product is the relative distance between the weight vector and the input vector. Once the dot product is calculated, the nodes compete to see who has the largest dot product. Only the winning node is allowed to output, and only the winner and his neighbors are allowed to adjust their weights. This is known as competitive learning. Figure 7 shows how a weight vector will adjust itself closer to the input vector on the unit circle. Neighbors are generally defined as the physically closest nodes. The exact size of a neighborhood allowed to output is variable. It generally shrinks in size during the training process. The Kohonen learning rule for adjusting weights is actually quite simple. It says:

$$w_{new}=w_{old}+A(x-w_{old})$$

where w_{new} is the new weight, w_{old} is the old weight, A is the learning constant, and x is the input vector. As you can see , the weights of the vector are adjusted to approximate the input. Experiments have shown this system to be effective for associative classifier type applications such as associative memory and pattern recognition. It also shows a high resistance to noise in the input.

The instar and outstar networks are also networks developed during studies of the brain.(Grossberg, 1982) The first is the instar. Each node is the center of a large number of inputs. These can be drawn as a star shape with the arrows pointing to the node. The second is the out star. Just as each node has a large number of inputs, its output goes to a large number of other nodes or outputs. This can be drawn with the node at the center of a large outwardly pointing star. Every network is comprised of a interwoven mesh of instars and outstars. The operation of the network depends on the interaction of the instars and outstars.

The learning algorithm for the instar is an unsupervised algorithm. The equation is:

$$\frac{dy_j(t)}{dt} = -Ay_j(t)+I_j(t)+đ_i\ w_i(y_i(t-t_o)-T).$$

The result of the equation is the change in response of the node, $y_j(t)$ is the current activation of the node j, $I_j(t)$ is the input to the node from outside the system, w_i is the weight for the input from node i within the system, $y_i(t-t_o)$ is the activation of node i in the previous time frame, T is a threshold value below which no response is desired, and A is a forgetting constant. From the equation, it can be seen that the activity of the node is based on a percentage of its previous activity, external stimuli, and stimulation of the other nodes within the system. The weights of the node are adjusted by using the equation:

$$\frac{dw_{ij}(t)}{dt} = -Fy_j(t)+Gy_j(t)[y_i(t-t_o)-T]$$

where y_j is the activation of the node j, $y_i(t-t_o)$ is the activation of the node i in the previous time frame, T is a threshold value to filter out noise, F is a forgetting constant and G is a learning constant.

The result of the equations is that the node acts in a manner similar to Pavlov's dogs. In psychological experiments, Pavlov observed that dogs salivate when presented with food. He began ringing a bell just prior to presenting the dogs with food. Eventually the dogs were conditioned to salivate when hearing the bell. In the instar equations, activation occurs when the input I_j occurs. I_j is like the food in the Pavlov experiments. At the same time, the node is receiving a pattern of stimulation from other sources in the network. This stimulation can be thought of as the bell. Eventually, the node will learn to give the same response to the stimulation pattern as it does to the input I_j. This is a powerful tool because it gives insight into actual biological function, as well as increasing our knowledge about artificial neural networks. Instars and outstars are rarely used as stand alone neural networks, but they have enormous potential for application.

The application potential of all artificial neural networks is enormous and generally untapped. The similarity of the architecture to the biological neural systems of higher organisms suggests that the applications that the ANN's are best suited to are those performed by the biological neural system. It is for this reason that an artificial neural system was chosen to duplicate the functions of early vision. The purpose is not prove the unsuitability of a more traditional computer architecture, but rather to prove the suitability of the artificial neural network in this type of application.

CHAPTER 3
COUNTERPROPAGATION NETWORK

It is also possible to create networks from a combination of different types of node algorithms. The counterpropagation network is the first developed in this manner (Hecht-Nielson 1987a), and it consists of a combination of both Kohonen learning and Grossberg learning methods. Figure 8 shows a graphical model of the network. The resulting network is capable of many of the same types of functions as the backpropagation networks, but it is able to train at a much faster rate. This makes it ideal for tasks where a lengthy training time is undesirable. In fact, because of the ability to train quickly, the counterpropagation algorithm can be used to test the feasibility of a neural network solution when the end product may be a backpropagation or other similar network. It is for this reason that a counterpropagation network was selected to test the feasibility of a neural network solution to the detection of visual primitives.

As shown in Figure 8, pairs of example vectors(x,y) are presented to the network at layers 1 and 5. These vectors propagate through the network in a counterflow pattern to yield x' and y' at layers 2 and 4 (Hecht-Nielson, 1987).

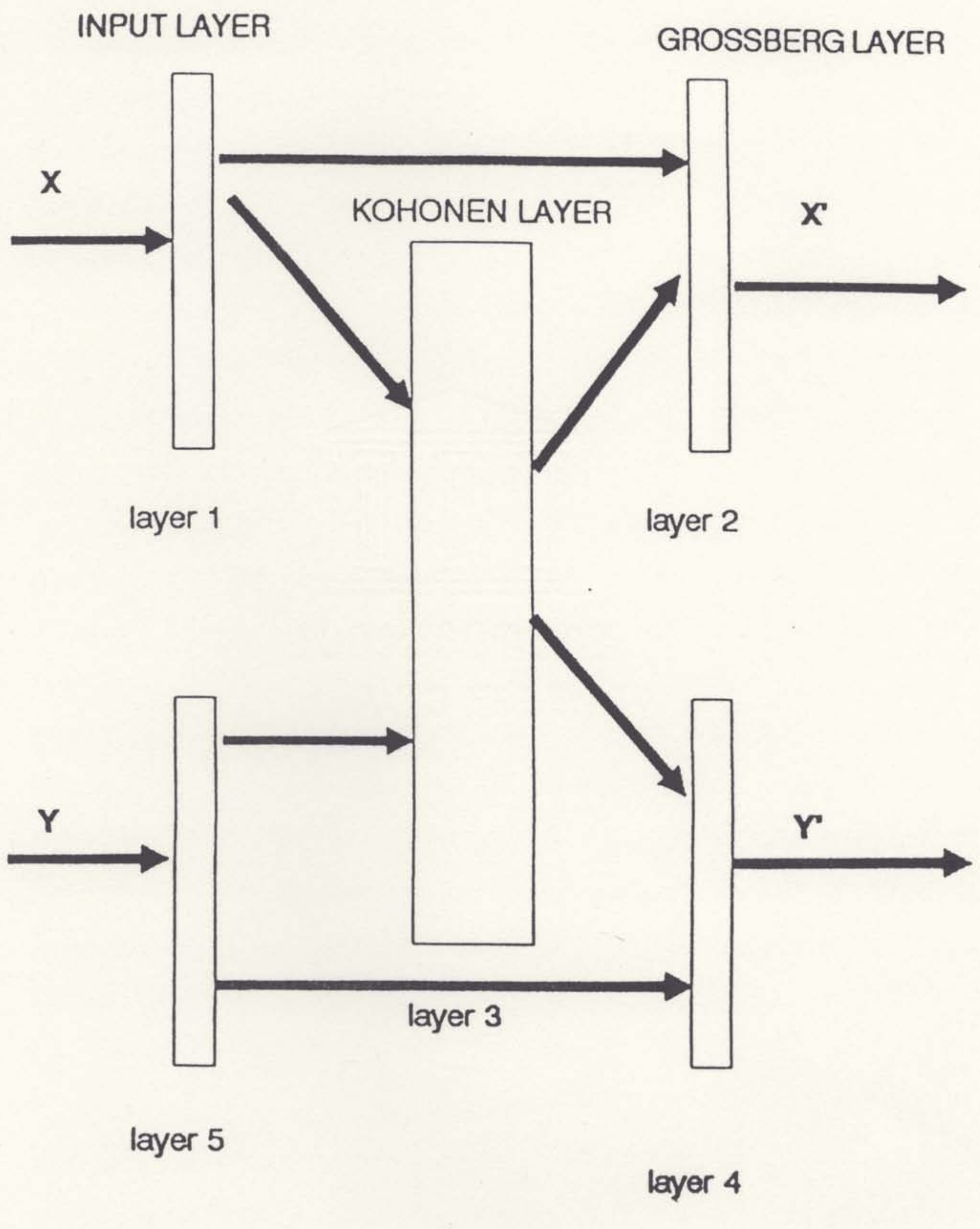

Figure 8. Counterpropagation Network.

For this application, the forward-only version of the counterpropagation network will be used. Figure 9 shows this version of the network. This version consists of three layers. An input layer consisting of N fanout units is the first layer. The middle layer contains M Kohomen nodes with one output each which connect to the third layer of Grossberg outstars. The output of the outstars is the output of the network. This version of the network trains in the supervised mode with the desired output presented to the output layer at training time.

The inputs into the network are fanned out to the Kohonen layer. In this layer, the nodes sum of the product of each input with its associated weight. Then, the nodes compete to see which has the highest sum. This node is designated the winning node and is the only node allowed to output for that frame. It is also the only node allowed to adjust its weights. The layer will self-organize in response to the inputs so that the weights will be organized as more of less statistically optimal sets of the possible input vectors. The learning equation for the winning node is

$$w_{new}=w_{old}+A(I-w_{old})$$

where A is a learning constant between 0 and 1, and I is the associated input into the node. This Kohonen learning law moves the closest matching weight vector towards the latest

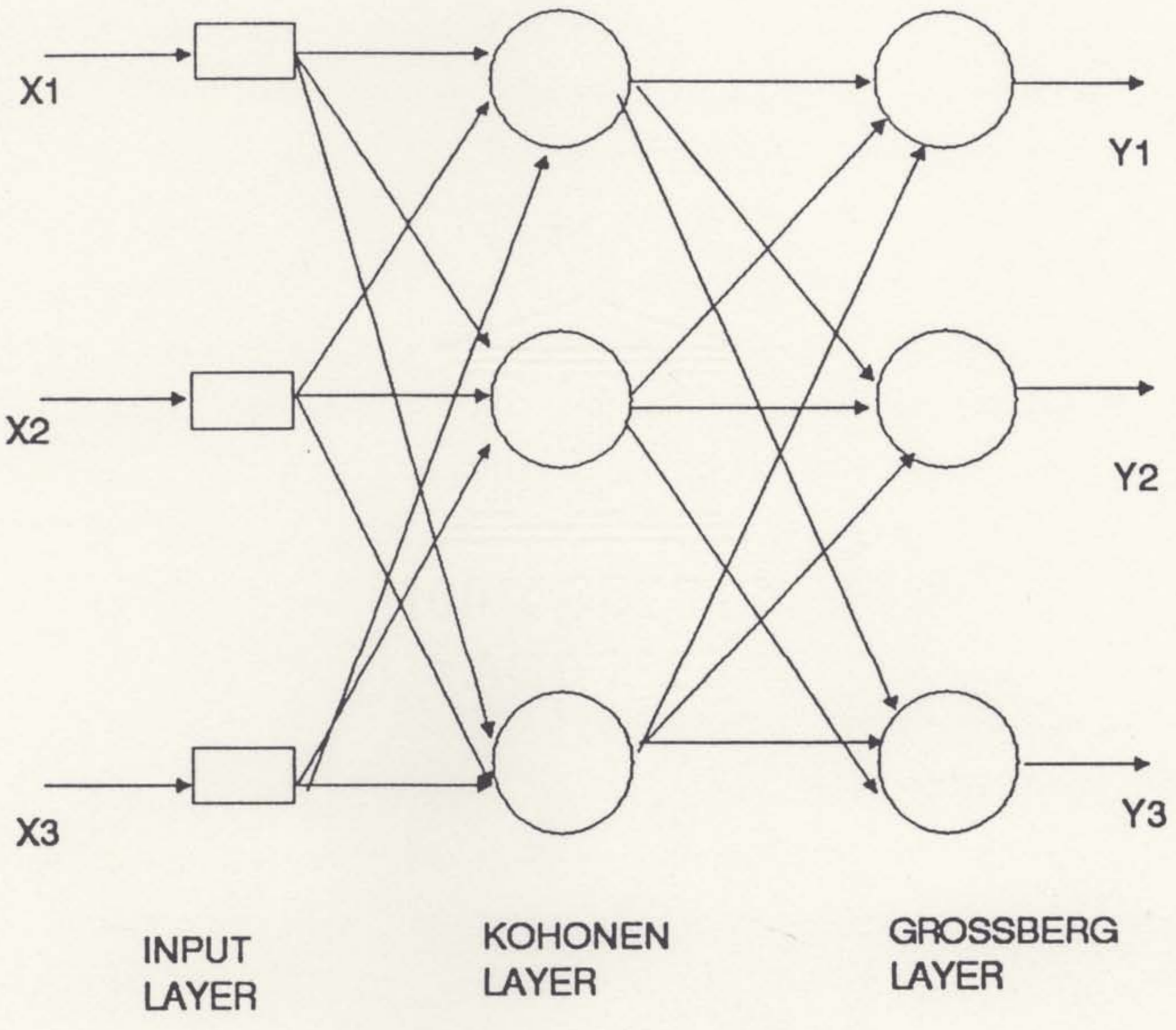

Figure 9. Feed Forward Counterpropagation Network.

input vector at a rate determined by the learning constant, A. A large value causes the network to converge more quickly than a smaller value, but some statistical information may be lost with a too large value.

An alternative variant of this layer is to operate in an interpolative mode, as opposed to the accretive mode discussed above. In this mode, more than one node is allowed to win the competition and produce an output. The outputs are adjusted so that they sum to one (the former single output signal), and all are allowed to adjust their weights. This interpolation process may lead to an increase in mapping approximation accuracy without increasing the network size, however, there is no conclusive evidence to proving this theory.

The outputs of the Kohonen layer are connected to the Grossberg layer. The Grossberg layer learns the average output vector when each of the Kohonen processing elements wins the closeness competition. Each Grossberg element sums the product of the Kohonen output with its associated weight. The output for a continuous valued network is this value. In a binary network, the summation value is compared to a threshold value and set equal to one if greater than the threshold , or zero if less than the threshold. The layer then adjusts its weights appropriately. Only the weights associated with a non-zero input are allowed to adjust using the following equation:

$$w_{new}=w_{old}+B(d-w_{old})$$

where d is the desired output for that element and B is a learning constant similar to A for the Kohonen nodes.

The CPN functions very much like a lookup table. The Kohonen layer learns the characteristics of the input data, and the Grossberg layer learns the desired output for that particular set of data. Because of this simplistic function, counterpropagation is inferior to backpropagation for many applications. However, it forms a good statistical model of its input vectors and works well for applications such as data compression, pattern recognition, function approximation, and statistical analysis(Hecht_Nielson 1988). The network shares the high resistance to noise demonstrated by the Kohonen self-organizing network. Also like the Kohonen network, a large number of Kohonen nodes are often necessary to obtain good performance.

CHAPTER 4
DETECTION OF VISUAL PRIMITIVES

For purposes of implementing the feature map function, a simulation of the feed forward counterpropagation network was developed in the C language. Listings of the program are included in Appendix A. The next step was to discover the limits on the size of the network and determine how large a visual model was feasible. After experimentation, the maximum size was found to be a 30 by 30 input image with 35 Kohonen nodes. The Grossberg layer was designed to have either two or three output nodes. The limitations on the size was due to the capability of the computer chosen for the implementation, an Intel 80386 based personal computer with one megabyte of RAM.

The simulation is capable of running in either training or operational mode. In training mode, an input file is expected to hold both inputs and desired outputs of the network. In operational mode, only input data is expected in the input file. Two weight files are used; one contains the weights for the Kohonen layer and the second the weights for the Grossberg layer. The design of the network simulation in operational mode is fairly simple. The input vector is normalized and the sum of each Kohonen node is calculated. After the winning Kohonen node is determined, the outputs to the Grossberg layer are set with the winning node outputting a one and the other nodes outputting zero. The summation of each Grossberg node is performed and the result is compared to a threshold value of 0.8. If the summation is greater than the threshold, the output of the node is set to one, otherwise, it is set to zero. The three outputs are then examined to determine if the feature is present, not present, or undetermined (Wasserman 1989).

The training mode is similar with the addition of steps to adjust the weights of the winning Kohonen node, and the Grossberg nodes' weights. At the beginning of a training session, all weight vectors are normalized. By normalizing the input vector as well, the Kohonen weight vectors will remain normalized vectors after adjustment. The weight adjustment will merely rotate the weight vector closer to the input vector on the unit circle. In many neural networks it is customary to randomize the initial weight vectors. However, in Kohonen learning, this can cause serious training problems as it will uniformly distribute the weight vectors about the unit circle. If the input vectors are not evenly distributed about the unit circle, some nodes will never win the competition and be allowed to adjust their weights. This wastes those nodes and effectively decreases the size of the network. This can be disastrous if a mapping of two similar inputs to different outputs is desired. In order to avoid this pitfall, all of the Kohonen weight vectors were set to equal coincident values, and a training technique known as the convex combination method was used to train the Kohonen layer.

In the convex combination method, all weights are initially set to $1/(N)^{1/2}$, where N is the number of inputs. During training the input vector **X** is given the value:

$$X=A*X+(1/(N)^{1/2}*(1-A))$$

where A is initially the small value of 0.1 and is increased gradually until it reaches a value of 1.0. This allows the input vectors to gradually separate and assume their true values. Each node's weight vector will follow one or a group of the input vectors. This slows the training process, but makes sure that no Kohonen nodes are wasted (Wasserman 1989).

The Grossberg layer trains much faster than the Kohonen layer. Therefore, the simulation does not adjust the Grossberg weights until the Kohonen layer has had a chance to converge. Five passes are made with a learning constant of 0.7 before any adjustments are made to the Grossberg layer. On the sixth pass, the Kohonen layer has converged and the Grossberg layer can be gradually trained to match the desired output. At this point, the learning constant is gradually decreased to a minimum value of 0.1 until the network converges.

After the simulation was judged to be robust, visual primitives were selected for recognition. Since the network used a binary image, color was ruled out as a candidate. Line orientation was selected due to its general acceptance as a primitive (Treisman 1985), as well as the biological evidence of cells that detect line orientation provided by Hubel and Weisel (Hubel 1988). Training sets were developed in three categories: sets containing the line, sets without the line, and sets containing more than one line. For the first orientation, which was horizontal, 38 training sets were developed. Thirty of these contained a single horizontal line, four contained a variety of non-horizontal lines, and four contained a number of horizontal lines. Initially, a network with only two outputs was used, one for feature present and one for feature not present. It became obvious during training that another output was needed. Some of the sets from the category without a horizontal line and from the category with many horizontal lines were mapping to the same Kohonen node. When this occurred, the network would not converge. When these similar sets were mapped to a third output called "undetermined" , the network converged quickly in only nine passes through the training sets. This result is understandable when some of the psychological test results are examined. Treisman and Schmidt found that illusory conjunctions can occur when attention is diverted

```
1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 10. A Training Image Mapped to "Undetermined".

or overloaded (Treisman and Schmidt 1982). The images that mapped to "undetermined" all contained a large number of lines or curves which would place a strain on attention. Figure 10 shows an example of an image which mapped to "undetermined".

Similar training sets were developed for different line orientation. Between 38 and 40 training images were developed for each. When the images containing a large concentration of features were mapped to the "undetermined" category, the network converged in eight or nine passes through the training set. As discussed in Chapter 3, the ability to train quickly is an advantage of the using the counterpropagation neural network. A disadvantage is that the Kohonen layer training is unsupervised and if two data sets meant to be mapped to different outputs end up mapped to the same Kohonen node, the network will never converge. This can be avoided by using a larger network, but if size is an issue, the counterpropagation network is a bad choice. Despite the constrained environment used for this research, the network was able to accurately map images containing different line orientations into the correct category. The most interesting result was the inability of the network to immediately identify whether or not a feature was present in "noisy" images. This trait is shared by the feature maps in the human visual system, although it was not intentionally trained into the artificial or biological network.

CHAPTER 5
RESULTS AND CONCLUSIONS

After the counterpropagation network was trained to recognize the selected visual primitive, a series of test data similar to that used by Treisman in her experiments on human recognition of visual primitives (Treisman 1985) were passed through the network. Figures 11 and 12 show an examples of the images run through the network trained to recognize horizontal lines. Figure 11 shows an image which gave a result of feature present, and the image in Figure 12 gave a result of feature not present. Other test data was passed through and the results were very accurate.

The next step was to attempt to train a different visual primitive. Curvature was chosen as a good candidate for training. The network was trained using a series of images containing circles mapped to feature present, a series of lines in different orientations mapped to feature not present, and a few noisy images similar to those in used in line orientation mapped to undetermined. After the network was successfully trained to recognize curvature, a series of test images were run through the network. These included images in which the circle was bisected with a horizontal line, and others with only horizontal lines. Figure 13 shows a test image which mapped to feature present

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
```

Figure 11. A Test Image Mapped to Feature Present.

```
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 12. A Test Image Mapped to Feature Not Present.

and Figure 14 shows a test image which mapped to feature not present.

The results obtained in training the two types of visual primitive were excellent. The counterpropagation showed a great deal of resistance to noise and was able to pick out the desired primitive among a surprisingly large number of distractors. Obviously, there is a limit on the noise resistance as shown by the necessity of including the undetermined output. However, this limit is shared by the human visual system and is an accurate recreation of feature maps in the human visual system, and that is the goal of the research presented here.

Since such good results were received in training the network to recognize line orientation and curvature, it was decided to attempt to train a more complex property. The property of juncture is a possible visual primitive. There have been test results which show that it is probably not a primitive shared by most humans (Treisman 1985). However, since there are results showing that some individuals can be trained to detect new perceptual units automatically in search (Schneider and Shiffrin 1977), the attempt to train the counterpropagation network to detect a corner was made. A corner angled at -90 degrees was chosen as the feature to detect. While developing training data, it became obvious that a larger network would be necessary to get a good

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 13. Image mapped to Curvature Feature Present.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 14. Image mapped to Curvature Feature Not Present.

result. By decreasing the size of the input image to a 25 by 30 array,the network became proportionally larger. Even with the larger network and a proportionally larger training set, the network is still only capable of recognizing corners in a limited area within the image.

Interestingly enough, this result was not unexpected. Treisman and Patterson conducted experiments to detect whether similar features known as emergent features were actually primitives or not. Their results were inconclusive (Treisman and Patterson 1980). Emergent features are defined as combinations of simple elements which appear to generate new interaction of relation properties. It is theorized that some emergent features may also be detected by independent feature maps and thus qualify as visual primitives by themselves. As stated earlier, experiments to show that emergent features are primitives neither proved nor disproved the theory, just as my attempt to train the network was successful in the sense that some corners are detected and unsuccessful in that all corners cannot be detected by the network.

The fact that both the training attempt and the experiments were inconclusive suggests that emergent features such as corners may be trained as primitives in certain individuals, but not in everyone. If a large enough network were used and enough time given, a network could be trained to recognize all corners or other emergent feature.

Just as a human will train his or herself to recognize corners automatically if there is a need, a machine visual system could be trained to recognize such features if there is a need. The decision to invest the time and resources to do train a network versus recognizing the features at a secondary level would be a system application sensitive decision.

The research presented here was very successful. The counterpropagation network trained and detected visual primitives in a manner not unlike that of human early vision. The network accurately mapped simple visual primitives such as line orientation and curvature quickly and easily. The network was able to map the more complex feature of juncture as well, although not as quickly and easily. Although the simulation enviroment used was constrained, the network enviroment could easily be changed to a larger , more powerful computer and more complex mappings could be achieved. The counterpropagation network served its purpose well by converging in relatively few training attempts. However, it required a larger network than a more complex network paradigm such as backpropagation would due to the simplistic nature of the counterpropagation network. For this reason alone, it would worthwhile to explore the abilities of a different network pardigm in detecting visual primitives. On the other hand, if only the simpler primitives are to be detected, the counterpropagation model is ideal as shown by the results presented here. The counterpropagation model was easily trained to recognize the simple visual primitives even in a cluttered enviroment. If a complete early visual system were developed, the counterpropagation network would work well as a feature detector for the simple visual primitives such as line orientation. If the need to detect a more complex feature was desired, the counterpropagation could be used to test the feasibility of a neural network solution and a more complex paradigm could be used for the final product. The next step in this research would be to train separate artificial neural networks to detect a selected number of visual primitives and devise a means to use the output of these neural networks to simulate the entire process of early vision. Once that is accomplished, the next step is simulating the attentive stage of the visual process and combining them for a complete machine visual system.

APPENDICES

APPENDIX A

COUNTERPROPAGATION NETWORK SIMULATION SOURCE CODE

```
#include <stdio.h>
#include <alloc.h>
#include <math.h>
main()
{
  int c,itemp;
  int tie,k2,k3;
  FILE *weights, *idata,*kweight;
  unsigned char *input,*GROS,*OUTPUT ;
  float  *GWT;
  float  *temp,*FANIN, *KOH;
  float  huge *KWT[45];
  float tryout,learn, ftotal, alpha;
  int I,J,N, M, kwin,pass;
  int i,j,k,l,m,sets,curset,total;
  long n;
  short int train,rep,cont;
  unsigned long memory;
  double square;

  n=farcoreleft();
  /*get number of nodes per layer*/
   N=750; /*Number of inputs*/
   M=3; /*Number of outputs*/
   J=35; /*Number of Kohonen cells*/
   alpha=0.1;

   input=(unsigned char *)calloc(N,sizeof(unsigned char));
    if (input==NULL)
      printf("Memory allocation error on input\n");
   GROS=(unsigned char *)calloc(M,sizeof(unsigned char));
    if (GROS==NULL)
      printf("Memory allocation error on GROS\n");
   OUTPUT=(unsigned char *)calloc(M,sizeof(unsigned char));
     if (OUTPUT==NULL)
       printf("Memory allocation error on OUTPUT\n");
   temp=(float *)calloc(J,sizeof(float));
     if(temp==NULL)
       printf("Memory allocation error on temp\n");
    FANIN=(float *)calloc(N,sizeof(float));
      if (FANIN==NULL)
        printf("memory allocation error on temp\n");
    GWT=(float *)calloc((M*J),sizeof(float));
      if (GWT==NULL)
        printf("Memory allocation error on GWT\n");
     KOH=(float *)calloc(J,sizeof(float));

       if(KOH==NULL)
        printf("Memory allocation error on KOH\n");
     for (i=0;i<J;i++)
      {
       KWT[i]=(float *)farcalloc(N,sizeof(float));
       if (KWT[i]==NULL)
         printf("Error in allocating KWT\n");
        printf("i=%d\n",i);
      }

  printf("Is this a training run-Y or N?\n");
  c=getchar();
  if(c=='Y' || c=='y')
  {
 /*Open files for reading*/
   kweight=fopen("kweight.dat","r+");
   if (kweight==0)
    printf("error in opening weights file\n");

  /*read in weights from file*/
  /*Kohonen weights*/
  for (i=0;i<J;i++)
   {
   for(j=0;j<N;j++)
    {
      fscanf(kweight,"%f",&tryout);
      KWT[i][j]=tryout;
     }
   }
    close(kweight);
   weights=fopen("weights.dat","r");
  /*Grossberg weights*/

  for (i=0;i<M*J;i++)
   {
     fscanf(weights,"%f",&tryout);
     GWT[i]=tryout;
   }

 /*Close weights file*/
  fclose(weights);
/* net is training*/
     train=1;
     idata=fopen("train3.dat","r");
     if (idata==0)
       printf("error opening input file\n");

 if (train==1)
 {
 /*Determine number of sets of training data*/
 total=0;
 pass=1;
 sets=0;
 cont=0;
 while (cont==0)
```

```
{
 sets++;
 for (i=0;i<N;i++)
 { fscanf(idata,"%d",&itemp);
  cont=feof(idata);
 }
 if (cont!=0)
   sets--;
 for(i=0;i<M;i++)
  fscanf(idata,"%d",&itemp);
 cont=feof(idata);
}
fclose(idata);
idata=fopen("train3.dat","r");

/*Begin a training session*/
curset=1;
learn=0.7;
rep=1;
while(rep==1)
 {
  /*read in training data*/
  for(i=0;i<N;i++)
    {
     fscanf(idata,"%d",&itemp);
     input[i]=itemp;
    }
  for(i=0;i<M;i++)
    {
     fscanf(idata,"%d",&itemp);
     OUTPUT[i]=itemp;
    }
 /*Normalize input vector*/
 square=0;
 for(i=0;i<N;i++)
  square=square+(input[i]*input[i]);
 if (square!=0.0)
 {
  square=sqrt(square);
 for(i=0;i<N;i++)
 { FANIN[i]=input[i]/square;
   FANIN[i]=alpha*FANIN[i]+(0.03125*(1-alpha));
 }
 }


/*KOHONEN CALCULATIONS*/

/*Calculate temp[i] = KWT[i][0..N]*FANIN[0..N]*/
 for(i=0;i<J;i++)
  {
    temp[i]=0;
    for(j=0;j<N;j++)
      {
        temp[i]=temp[i]+(FANIN[j]*KWT[i][j]);
       }
    }

 /*Find winning Kohonen node*/
     kwin=0;
   for(i=1;i<J;i++)
    {
     if(temp[i]>=temp[kwin])
        kwin=i;
    }


/*Adjust weights of winning Kohonen node*/
    /*Read in node weights*/


   for(i=0;i<N;i++)
     {
      KWT[kwin][i]=KWT[kwin][i]+learn*(FANIN[i]-
KWT[kwin][i]);
      }


/*Set output for Kohonen nodes*/
   for(i=0;i<J;i++)
    {
     KOH[i]=0;
     if(i==kwin)
      {
       KOH[i]=1.0;
      }
    }

/*GROSSBERG CALCULATIONS*/

/*Clear out temp locations*/
 for(i=0;i<j;i++)
  temp[i]=0.0;


    if (pass>=6)
    {
/*Calculate     value     of     Grossberg     node-
temp[i]=GWT[i][0..J]*KOH[0..J]*/

  for(i=0;i<M;i++)
   {
     for(j=0;j<J;j++)
      {
        temp[i]=temp[i]+(KOH[j]*GWT[(i*J)+j]);
      }
    }

/*Set output of Grossberg layer*/

  for(i=0;i<M;i++)
    {
     if(temp[i]>0.8)
      GROS[i]=1;
     else
      GROS[i]=0;
    }

/*Adjust weights of Grossberg layer*/
  for(i=0;i<M;i++)
    {
     for(j=0;j<J;j++)
      {
        if(KOH[j]>0)
         {
          GWT[(i*J)+j]=GWT[(i*J)+j]+learn*(OUTPUT[i]-GWT[(i*J)+j])
;
         }
      }
    }
  /*Determime accuracy of result*/
    j=0;
    for(i=0;i<M;i++)
     {
      if (GROS[i]!=OUTPUT[i])
       j++;
     }
     total=total+j;
    printf("Training  set  %d,  Incorrect  outputs=
%d\n",curset,j);
     }
     else
      { total=1;
        learn=0.7;
      }
    /*Determine if complete one pass training */
    if (curset!=sets)
     curset++;
    else
     {
     if (total==0)
       rep=0;
      else
```

```
          {
           fclose(idata);
           idata=fopen("train3.dat","r");
           curset=1;
           learn=learn-0.005;
           if (learn<0.1)
             learn=0.1;
           alpha=alpha+0.1;
           if (alpha>1.0)
            alpha=1.0;
           total=0;
           pass++;
           printf("Training pass=%d\n",pass);
          }
      }
     }
      fclose(idata);
      printf("Total number of training passes=%d\n",pass);
      /*write weights out to file*/

      kweight=fopen("nkweights3.dat","w");
      /*Kohonen weights*/
      for (i=0;i<J;i++)
       {
        for(j=0;j<N;j++)
         {
          fprintf(kweight,"%f ",KWT[i][j]);
         }
       }
      weights=fopen("nweights3.dat","w");
      /*Grossberg weights*/
      for (i=0;i<M*J;i++)
       {
         fprintf(weights,"%f\n",GWT[i]);
       }
     /*Close weights file*/
     fclose(weights);
}    }
    else
    {

     /*Open files for reading*/
       kweight=fopen("nkweight2.dat","r+");
       if (kweight==0)
        printf("error in opening weights file\n");

      /*read in weights from file*/
      /*Kohonen weights*/
      for (i=0;i<J;i++)
       {
       for(j=0;j<N;j++)
    {
      fscanf(kweight,"%f",&tryout);
      KWT[i][j]=tryout;
     }
   }
    close(kweight);
   weights=fopen("nweights2.dat","r");
  /*Grossberg weights*/

  for (i=0;i<M*J;i++)
   {
     fscanf(weights,"%f",&tryout);
     GWT[i]=tryout;
   }

 /*Close weights file*/
  fclose(weights);
     idata=fopen("input.dat","r");
     if (idata==0)
       printf("error opening input file\n");

  cont=feof(idata);
  while (cont==0)
  {


   /*read in input data*/
   for(i=0;i<N;i++)
    {
     fscanf(idata,"%d",&itemp);
     input[i]=itemp;
     cont=feof(idata);
    }
  if (cont==0)
  {
  /*Normalize input vector*/
  square=0;
  for(i=0;i<N;i++)
   square=square+(input[i]*input[i]);
  if (square!=0.0)
  {
   square=sqrt(square);
  for(i=0;i<N;i++)
  { FANIN[i]=input[i]/square;
  }
  }
    /*Display input data to screen*/
  for (i=0;i<30;i++)
   {
   for(j=0;j<30;j++)
    {
      printf("%d ",input[(i*30)+j]);
    }
    printf("\n");
   }

 /*KOHONEN CALCULATIONS*/

 /*Calculate temp[i] = KWT[i][0..N]*FANIN[0..N]*/
  for(i=0;i<J;i++)
   {
    temp[i]=0;
    for(j=0;j<N;j++)
      {
        temp[i]=temp[i]+(FANIN[j]*KWT[i][j]);
       }
    }

 /*Find winning Kohonen node*/
     kwin=0;
   for(i=1;i<J;i++)
    {
     if(temp[i]>=temp[kwin])
        kwin=i;
    }


/*Set output for Kohonen nodes*/
   for(i=0;i<J;i++)
    {
     KOH[i]=0;
     if(i==kwin)
      {
       KOH[i]=1.0;
      }
    }

/*GROSSBERG CALCULATIONS*/

/*Clear out temp locations*/
 for(i=0;i<j;i++)
  temp[i]=0.0;


/*Calculate     value     of     Grossberg     node-
temp[i]=GWT[i][0..J]*KOH[0..J]*/

  for(i=0;i<M;i++)
   {
    for(j=0;j<J;j++)
     {
      temp[i]=temp[i]+(KOH[j]*GWT[(i*J)+j]);
      }
    }

/*Set output of Grossberg layer*/

  for(i=0;i<M;i++)
    {
      if(temp[i]>0.8)
       GROS[i]=1;
      else
       GROS[i]=0;
    }
    if (GROS[0]==1)
     printf("Feature present\n");
    else
    {
    if(GROS[1]==1)
     printf("Feature not present\n");
    else
     printf("Feature possibly present\n");
      }
     cont=feof(idata);
   }
  }
  fclose(idata);

}
}
```

APPENDIX B

SAMPLE TRAINING DATA

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0
1
0
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
0
0
```

Figure 15. Sample Training Data for Horizontal Lines.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
1
0
```

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1
0
0
```

Figure 16. Sample Training Data for Slanted Lines.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1
0
0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
1
0
```

Figure 17. Sample Training Data for Corners.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 18. Sample Training Data for Curvature

BIBLIOGRAPHY

Beck, J. and Ambler, B. 1972. Discrminability of differences line slope and in line arrangement as a function of mask delay, Perception and Psychophysics 12(1A): 33-38.

Beck, Jacob, K. Prazdny , and Azriel Rosenfield. 1983. A Theory of Textural Segmentation, In Human and Machine Vision, ed. J. Beck, B. Hope and A. Rosenfeld. Orlando,FL: Academic Press, Inc.

Cowey, A. 1979. Cortical Maps and Visual Perception. Quarterly Journal of Experimental Psychology, 31:1-17.

Feldman, J.A. and D.H. Ballard. 1983. Computing with Connections. In Human and Machine Vision. ed. J. Beck, B. Hope and A. Rosenfeld. Orlando,FL:Academic Press, Inc.

Grossberg, Stephen. 1968. Some Physiological and Biochemical Consequences of Psychological Postulates, Applied Mathematics. 60:758-765.

Grossberg, Stephen. 1982. Studies of mind and brain. Boston: Reidel.

Grossberg, Stephen and E. Mingolla. 1986. Computer Simulation of Neural Networks for Perceptual Psychology, Behavior Research Methods, Instruments and Computers. 18(6):601-607.

Hecht-Nielson, Robert. 1987a. Counterpropagation Networks. Applied Optics. 26:4979-4984.

Hecht-Nielson, Robert. 1987b. Counterpropagation Networks. In Proceedings 1987 IEEE Conference on Neural Networks. ed. M. Caudill. New York: IEEE Press.

Hecht-Nielson, Robert. 1988a. "Applications of Counterpropagation Networks", Neural Networks, 1: 131-139.

Hecht-Nielson, Robert. 1988b. Neurocomputing:picking the human brain. _IEEE Spectrum_. 25(March):.36-41.

Hopfield, J.J. 1982. Neural Networks and Physical Systems with Emergent Collective Abilities. _Proceedings of the National Academy of Science, USA_. 79(April):2554-2558.

Hubel, D.H. and T.N. Wiesel. 1968. Receptive Fields and functional characteristics of monkey striate cortex. _Journal of Physiology_. 195: 215-243.

Hubel, D.H. 1988. _Eye, Brain and Vision_. New York: Scientific American Library.

Julesz, Bela. 1981. Textons, the elements of texture perception, and the interactions. _Nature_. 290: 91-97.

Kohonen, Tuevo. 1977. _Associative Memory:A System-Theoretical Approach_. New York: Springer-Verlag.

Kohonen, Tuevo. 1987. Adaptive, associative, and self-organizing functions in neural computing. _Applied Optics_. 26:4910-4918.

Kohonen, Tuevo. 1988. _Self-Organization and associative memory_. New York: Springer-Verlag.

Lippman, R.P. 1987. An Introduction to Computing with Neural Nets". _IEEE ASSP Magazine_. April: 4-22.

Minsky, M.L. and S. Papert. 1969. _Perceptrons_. Cambridge, MA: MIT Press.

Neisser, U. 1967. _Cognitive Psychology._ New York: Appleton-Century-Crofts.

Posner, M.I. 1978. _Chornometric Explorations of the Mind._ Hillsdale, NJ: Erlbaum.

Posner, M.I. and Henik, A. 1983. Isolating Representational Systems. In _Human and Machine Vision_ ed. J. Beck, B. Hope, and A. Rosenfeld. Orlando,FL: Academic Press.

Rumelhart, D., G. Hinton, G., and Williams. 1986. Learning Internal Representations by Error Propagation. In _Parallel distributed processing"_(Vol. 1). ed. D. Rumelhart and J.L. McCLelland. Cambridge, MA: MIT Press.

Rumelhart, D. and McClelland, J.L. 1986. Parallel distributed processing: Explorations in the micro-structure of cognition(vols I and II). Cambridge, MA: MIT Press.

Schnieder W. and R.M. Shiffrin. 1977. Controlled and Automatic Human Information Processing:I. Detection, Search, and Attention. Psychological Review. 84: 1-66.

Treisman, A. and G. Gelade. 1980. A Feature-Integration Theory of Attention. Cognitive Psychology. 12: 97-136.

Treisman, A. 1982. Perceptual grouping and attention in visual search for features and for objects. Journal of Experimental Psychology:Human Perception and Performance. 8: 194-214.

Treisman, A. and R. Patterson. 1984. Emergent Features, Attention, and Object Perception. Journal of Experimental Psychology: Human Perception and Performance. 10(1): 12-31.

Treisman, A. 1985. Preattentive Processing in Vision. In Human and Machine Vision II, ed. A. Rosenfeld. Orlando, FL: Academic Press.

Treisman, A. 1986. Properties, Parts, and Objects. In Handbook of Perception and Human Performance . ed. K.R.Boff, L. Kaufman and J.P. Thomas. New York: John Wiley & Sons.

Tusa, R.J., L.A. Palmer, and A.C. Rosenquist. 1975. The Retionatopic Organization of the Visual Cortex in the Cat. Neuroscience Abstracts. I: 52.

Wasserman, Philip D. 1989. Neural Computing: Theory and Practice. New York: Van Nostrand Reinhold.