ESD-TR-88-164

Technical Report
810

Implementing Artificial Neural Networks in Integrated Circuitry: A Design Proposal for Back-Propagation

S.L. Gilbert

18 November 1988

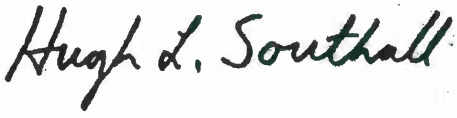
Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-85-C-0002.

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-85-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Hugh L. Southall, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS IN INTEGRATED CIRCUITRY: A DESIGN PROPOSAL FOR BACK-PROPAGATION*

S.L. GILBERT
Group 23

TECHNICAL REPORT 810

18 NOVEMBER 1988

*This report was a thesis submitted to the Department of Electrical Engineering and Computer Science, MIT, on May 2, 1988 in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

LEXINGTON MASSACHUSETTS

ABSTRACT

In an attempt to develop CMOS circuitry (both analog and digital) for the implementation of artificial neural networks, the back-propagation learning algorithm was examined in detail. Simulations were performed to determine the robustness of this algorithm to anticipated implementation artifacts such as quantization and weight-range limitation. Circuitry which computes with analog signals and digitally encoded weights was then designed to implement the algorithm within the tolerances determined by the simulations.

The architecture of an alternative, fully digital design was also defined and its performance compared with that of both the analog/digital design and a fully analog design based on circuitry that has been proposed in the literature.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS IN INTEGRATED CIRCUITRY: A DESIGN PROPOSAL FOR BACK-PROPAGATION

1. INTRODUCTION

Recently, a great deal of work has been done in an attempt to define computational architectures that capture some of the properties of biological neural systems. This work is motivated by the hope that these artificial systems will also demonstrate some of the properties of the biological systems including learning, massive parallelism, and fault tolerance. Although many of the results look promising, most of this work has been done by implementing software routines on conventional digital computers that model these highly distributed architectures. As a result, the simulations are often extremely slow and researchers are often limited in the size of the networks that can be practically simulated. Although most researchers recognize that implementation in custom hardware is an eventual goal, the progress toward this goal has been relatively slow.

One of the artificial neural learning algorithms that has been studied most intensively is the back-propagation algorithm. This algorithm has been applied to problems in character recognition [Burr, 86], speech recognition [Burr, 86] [Huang, 87] [Lippmann, 87] [Peeling, 86], text-to-speech processing [Sejnowski, 87], signal prediction [Lapedes, 87], protein structure analysis [Qian, 88] [Levin, 88], and a variety of other difficult problems. All these applications, however, have in common the fact that they have been implemented on conventional computers running software simulators.

This project attempts to define an architecture and circuitry which can implement the back-propagation algorithm in hardware that is specifically designed to take advantage of the distributed nature of artificial neural-network algorithms. First, the pertinent issues involved in back-propagation are discussed, and a general architecture described that can be used to implement the algorithm. Section 2 attempts to abstract the essential features of the problems to which back-propagation is typically applied, and Section 3 presents the specifics of how the back-propagation algorithm solves these problems. A specific architecture is defined in Section 4, and the results of simulations that were performed to model this architecture are presented in Section 5. Circuits with analog signaling and digital control that can be used to implement this architecture are described in Section 6. Finally, in Sections 7 and 8, alternative methods of implementing different portions of the architecture are presented and compared in terms of performance, cost, and feasibility.

The goals of this report are to present the issues that are critical to the design of hardware that implements the back-propagation algorithm, and to define a particular design which satisfies these constraints. Additionally, a variety of implementation techniques that have been applied to other artificial neural networks in the literature are discussed with respect to their relevance to the implementation of the back-propagation algorithm.

2. WHAT IS BACK-PROPAGATION USED FOR?

"Artificial neural network" is a generic term that is used rather loosely to describe systems that are characterized by a large number of relatively simple processors (neurons) that are interconnected to form a complex system. Unlike digital systems in which the processors are **programmed** to allow the system to perform a given task, in these systems it is the modification of the interconnection of the processors which is used to program the system. The pattern of interconnection thus stores the information. In the sense that biological neural systems are also thought to store information in a distributed fashion and not locally, artificial neural networks are "biologically inspired." This, however, is arguably where the biological inspiration ends for systems like the back-propagation networks that will be described in this report. The significance of such systems is thus not necessarily the biological inspiration so much as the fact that they represent an approach to computation that is radically different from that of more conventional digital systems.

There are a host of different designs that fall under the heading of artificial neural networks, but these designs can generally be characterized by the types of processors used, the way in which these processors are interconnected, and the method used to program or "learn" the interconnections pattern for a given task. In this context, the term back-propagation describes a particular learning algorithm (also known as the generalized delta rule) but it also implies a specific type of network architecture and a processor of a given characteristic. The specific learning algorithm was proposed by [Rumelhart, 86], but the general architecture and processor type is an adaptation of the perceptron architecture that was proposed by [Rosenblatt, 62] and was itself inspired by the work of McCulloch and Pitts in the 1940's.

The detailed specifications of the architecture and processors will be discussed in Section 4, but it is important to understand what these systems are expected to do before discussing how they are expected to do it. For now, the network will be thought of as a black-box with n inputs and m outputs as in Figure 2-1 (temporarily glossing over the fact that the box may actually be full of a large number of highly interconnected processors). The inputs and outputs of the network will be thought of as n- and m-dimensional vectors, respectively, with each component of each vector being a signal that can take on any value in a specified range of values. In this report, that range will generally be 0 to 1 unless otherwise noted. (This is a rather arbitrary choice, and consistency in using this range is more important than the actual range itself.)

The back-propagation algorithm has been applied to a wide variety of diverse problems including character recognition [Burr, 86], speech recognition [Burr, 86] [Huang, 87] [Lippmann, 87] [Peeling, 86], text-to-speech processing [Sejnowski, 87], signal prediction [Lapedes, 87] and a variety of other difficult problems. In the context of this research, however, it is most important to abstract the fundamental properties of these problems and attempt to define the essential characteristics of a back-propagation problem in an abstract context. I hope that this will lead to a design which is useful for a wide variety of problems, instead of being useful for only one particular application.

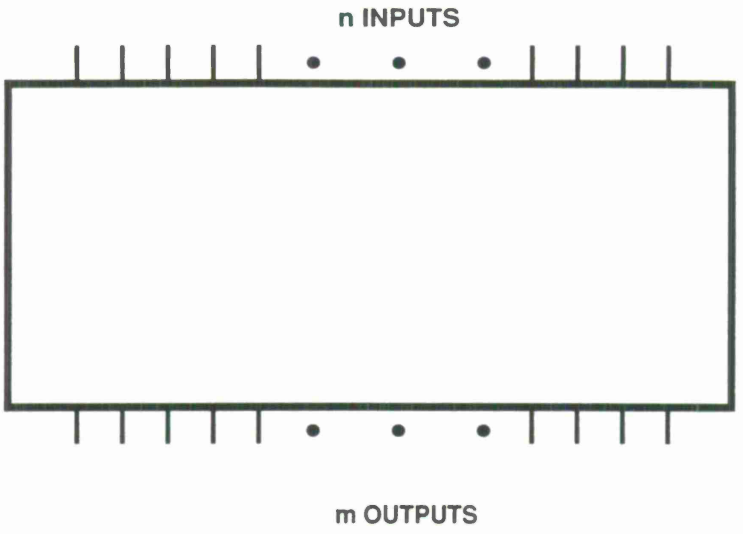


Figure 2-1. Black-box view of back-propagation network.

Pattern **association** is one problem to which back-propagation networks are commonly applied [Sejnowski, 87] [Lapedes, 87]. In such a problem there is a set of input and output vector *pairs*. The network is "programmed" with a particular weight pattern so that, when one of the input vectors is applied, the network produces the corresponding output vector. The power of such a network as a pattern associator is basically twofold. First of all, unlike conventional digital systems that are programmed by a person, a back-propagation network "learns" the correct weight pattern from experience. As will be discussed in much greater detail in Section 3, the weight pattern that programs the network is found by a training process that involves sequentially applying the input vectors while at the same time supplying the network with the desired output. At each pass through the training database, the network uses this information to modify its weights until it learns the appropriate weight pattern. The second significant ability of a back-propagation network when used as a pattern associator is its ability to generalize. That is, if a noisy version of one of the input vectors is applied to the network, the network should still produce an output that is similar to the one corresponding to the noise-free input vector. If an input vector is halfway between two training vectors, then the output should be a combination of the outputs that correspond to those two training vectors.

Pattern association can be thought of abstractly by thinking of the network as being used to define a smooth m-dimensional function in n-dimensional input space. The network is trained by simply giving it information about the function at a certain limited number of points in n-space, and the network is then used to interpolate values of the function at previously unencountered points.

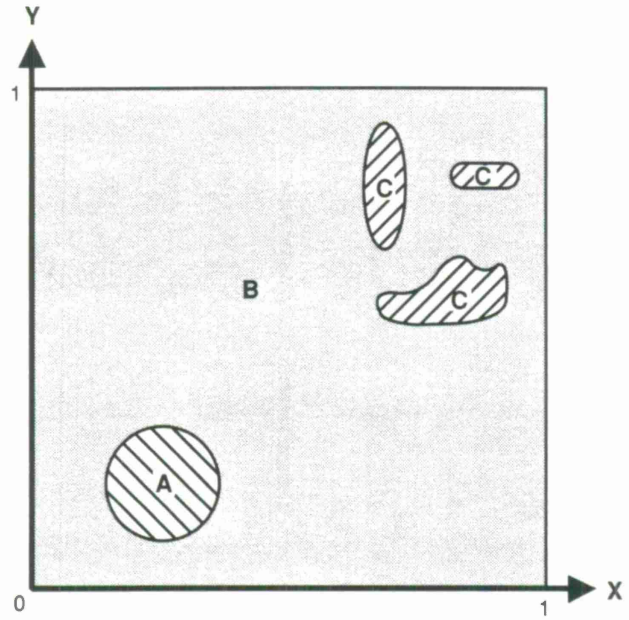
Another common application of back-propagation networks is pattern **classification** [Burr, 86], [Huang, 87], [Rumelhart, 86]. Although this application has many things in common with the pattern **association** application discussed above, there are significant differences between

the two. In the context of pattern **classification**, the input vector should be thought of as a point in n-space. The objective is to define n-dimensional regions in n-space and have the network determine which region the input "point" is in. The output of such a network would be a code that indicates which region the input vector is in. Thus, the network is associating input vectors with output codes instead of the more complicated output vectors in the pattern **association** application. The important difference between a pattern **classifier** and a pattern **associator** is the performance on input vectors that are halfway between two of the training vectors. Whereas the response of the pattern **associator** would be interpreted as an interpolation between the two trained outputs, the response of the pattern **classifier** would be to choose the best matched output code.

This report will concentrate on back-propagation networks used as pattern **classifiers**. Although it is important that any hardware implementation be capable of performing the actual back-propagation algorithm for a variety of applications, pattern classification was chosen to evaluate design trade-offs in this study and was simulated extensively in order to predict the performance of the design. In these networks, there will be one output signal dedicated to each "class" of inputs. When an input vector is applied to the network, all of the output signals will be observed and the class corresponding to the largest network output will be considered the network's "classification" of that input. Although each component of the output vector will be able to take on any value in the 0 to 1 range, the component associated with the correct class will be trained to be 1, and all of the others will be trained to be 0.

Pattern classification problems can have varying levels of complexity and difficulty as measured by several parameters including connectedness, convexity, linear separability, and granularity. All these metrics describe characteristics that are best understood when the pattern classification problem is seen as defining regions in n-dimensional space and classifying input vectors according to the region in which they are located. A class region is *connected* if it is possible to move from any one point in the region to any other point in the region without ever leaving the region. Regions that are not connected will be referred to as *disjoint*. *Convexity* is determined for any region if no pair of points exists such that both points are in the region but a line segment connecting the two points includes points that are not in that region. In the simple 2-dimensional example shown in Figure 2-2, region A is convex and connected, region C is not connected, and region B is connected but not convex.

Unlike convexity and connectedness, the other parameters are more characteristic of a group of regions than any one particular region. Linear separability means that it is possible to separate the regions with a straight line in the 2-dimensional case, and with a single hyperplane in higher dimensional situations. Granularity is a qualitative metric which describes the fineness of detail required to describe the regions as compared with the 0 to 1 scale. While regions A and B in Figure 2-2 are not linearly separable, regions A and C are. The regions in Figure 2-2 exhibit low granularity, while those in Figure 2-3 have high granularity.



102480-2

Figure 2-2. Classification regions in 2-dimensional input space.

Finally, there is a very significant distinction to be made between problems in which class regions abut, as in Figure 2-2, and those in which there is *unclassified* space between adjacent class regions as in Figure 2-3. The difference between these two types of problems is that in the latter there are points in the domain space of the problem for which no class is defined; no one cares how these points get classified. It should be intuitive that when class regions abut, there is no room for error in defining the dividing line between the regions. When there is space between the regions, however, the division can be made with less precision and still satisfy the constraints of the problem. It will be shown in the simulation results that for back-propagation it does require less precision to define the class regions for a problem with space between the class regions.

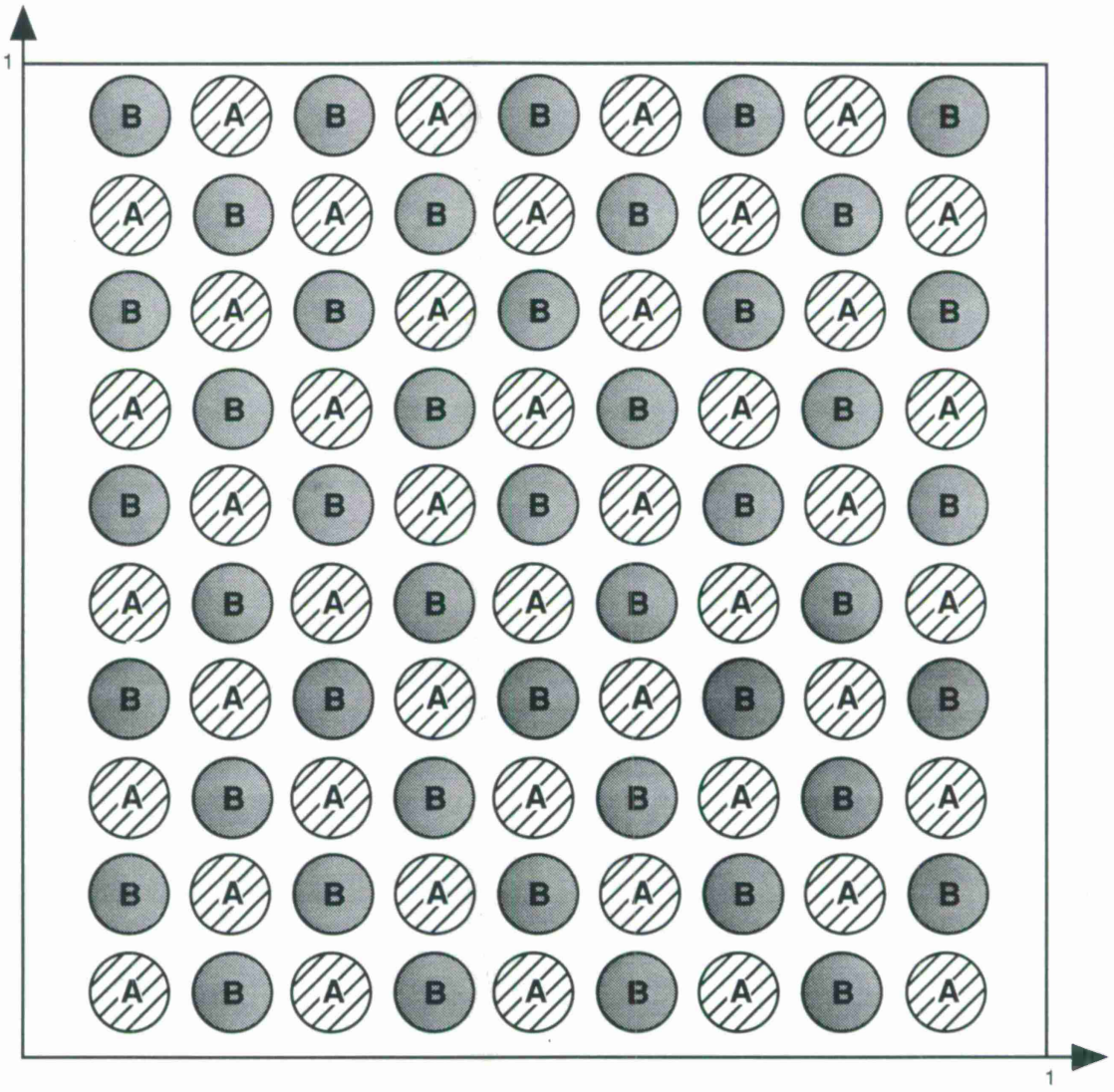


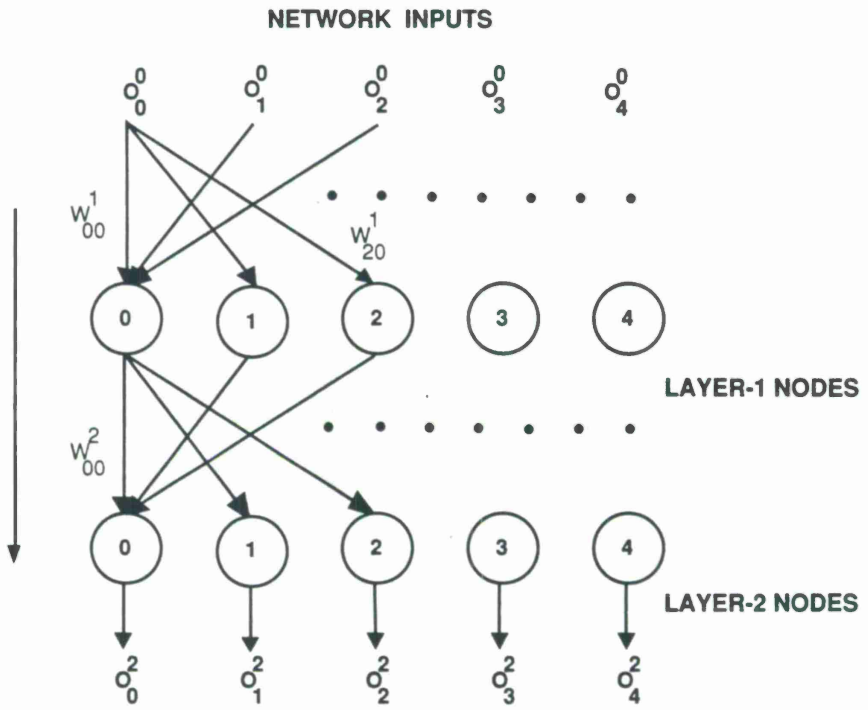
Figure 2-3. High granularity classification regions.

3. THEORY OF BACK-PROPAGATION ALGORITHM

Section A below is a review of the work presented in [Rumelhart, 86]. It is presented here because a detailed understanding of the back-propagation algorithm is essential for understanding critical implementation issues that will be emphasized in the following discussion. Section B describes a conceptual framework that provides some insight into how a back-propagation network functions. This insight will be valuable when discussing the results of the simulations that appear in Section 4.

A. THE GENERALIZED DELTA RULE

As discussed earlier, the term "back-propagation," although descriptive of the learning algorithm used in these networks, also implies a specific network architecture and a particular processor with certain characteristics. As shown in Figure 3-1, the underlying network architecture is composed of ordered layers of processors which are also referred to as "nodes" in the network. Nodes in one particular layer ℓ (ℓ = 1, 2, 3, . . . , L) receive input signals from nodes in the previous layer (ℓ – 1), and pass their outputs on to the nodes in the next layer (ℓ + 1). As a notational convention, the output signal from node j in layer ℓ is referred to as O_j^{ℓ} (following this convention, the i^{th} network input will be O_i^0). The connections between processors in different layers are characterized by strengths or *weights*.



102480-4

Figure 3-1. 2-layer network architecture.

The notation for the weight connecting node i in layer $\ell - 1$ to node j in layer ℓ is W_{ji}^{ℓ}. The action of a processor is to accept as input the weighted sum of the output signals from the previous layer, and produce an output by passing this weighted sum through an activation or transfer function such as the one shown in Figure 3-2. It should be noted in this figure that, because the strength of a connection can be positive, negative, or zero, the weighted sum which is the input to any node can be positive, negative, or zero.

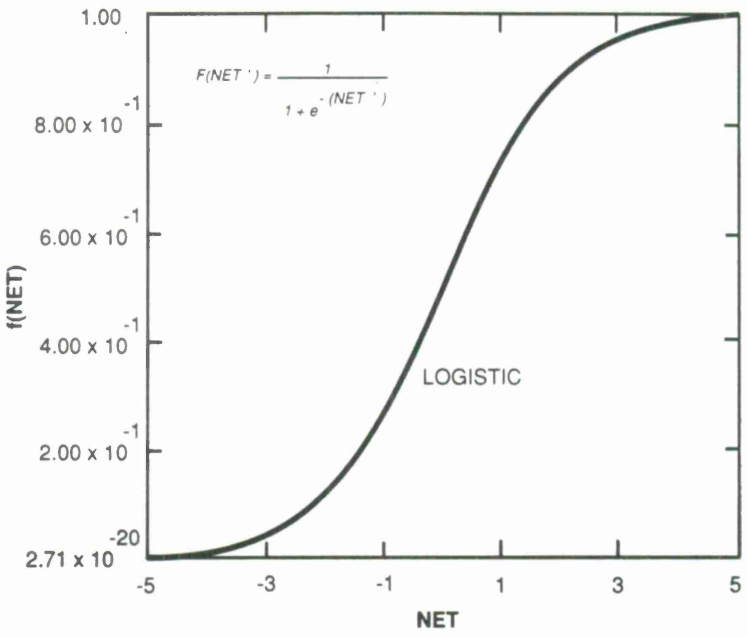


Figure 3-2. Logistic activation function.

Note that the superscript, which usually refers to exponentiation, will be used here to indicate association with a particular layer; exponentiation will be indicated by a circumflex:

x squared = x^ 2

The output of the j^{th} node in the 2^{nd} layer = O_j^2 .

A one-layer network would have one layer of nodes each with its own distinct weighted connection to each of the network's input signals. A two-layer network would have a second layer of nodes that calculate their inputs as the weighted sum of the outputs of the first layer. Networks of more than two layers are constructed by adding additional layers in this same fashion. In a multi-layer network, all layers except for the output layer are called *hidden* layers, and nodes in these layers are called *hidden* nodes.

Back-propagation defines an algorithm for modifying weights in order to solve a particular problem. The algorithm starts with a network with random weights. An input vector is applied to

the network while the outputs of the network are observed. The magnitude of the difference between the actual output vector of the network and the desired output vector for that particular input is used to generate an error function that is a measure of how well the network has been trained. Back-propagation then specifies how to calculate the weight changes required to do a **gradient descent** of this error function.

To be more precise, back-propagation defines an error function which measures the closeness of the actual outputs of the network (O_j^L) and the target outputs (T_j) for all of the patterns in the training data set:

$$E = \sum_p \sum_j \frac{1}{2} (T_j - O_j^L)^2 \Big|_p \tag{3-1}$$

where p is used to index the patterns in the training set. For the sake of clarity, the subscript "p" will only be made explicit when there is some ambiguity. In general, it should be assumed that all variables represent signals that are specific to the particular input pattern.

The idea is to choose weight changes which will reduce this error. This effect is maximized by selecting weight changes in the direction opposite to the gradient of this error function, thus performing a **gradient descent** of the error function. That is:

$$-\Delta W_{ji}^{\ell} \propto \frac{\partial E}{\partial W_{ji}^{\ell}} \quad . \tag{3-2}$$

The first simplification is to notice that the derivative of a sum is the sum of the derivatives and, hence, the effects of each pattern in the training set can be observed independently:

$$\frac{\partial E}{\partial W_{ji}^{\ell}} = \sum_p \frac{\partial E_p}{\partial W_{ji}^{\ell}} \quad . \tag{3-3}$$

To calculate these partial gradients, then, the error function must be represented as a function of the W_{ji}'s. To do this, first define the input to the j^{th} node in layer ℓ, which is a weighted sum of the outputs of the previous layer, to be NET_j^{ℓ}:

$$NET_j^{\ell} = \sum_i O_i^{\ell-1} * W_{ji}^{\ell} \quad . \tag{3-4}$$

Furthermore, the activation function which relates the output of a node to its input will be referred to as f(). In order to calculate the gradient, this must be a differentiable function [$\partial f()/\partial NET = f'()$]. With this notation, the gradient can be calculated by repeatedly applying the chain rule. First, for weights in the output layer (layer L):

$$\frac{\partial E_p}{\partial W_{ji}^{L}} = \frac{\partial E_p}{\partial NET_j^{L}} * \frac{\partial NET_j^{L}}{\partial W_{ji}^{L}} \quad . \tag{3-5}$$

From Equation (3-4), the second term can be calculated as $\partial NET_j^L / \partial W_{ji}^L = O_i^{L-1}$. For notational convenience, define $\delta_j^L = \partial E_p / \partial NET_j^L$ [the first term in Equation (3-5)], so that:

$$\frac{\partial E_p}{\partial W_{ji}^L} = \delta_j^L O_i^{L-1} \quad . \tag{3-6}$$

Additional application of the chain rule shows that δ_j^L is:

$$\delta_j^L = \frac{\partial E_p}{\partial NET_j^L} = \frac{\partial E_p}{\partial O_j^L} * \frac{\partial O_j^L}{\partial NET_j^L} \quad . \tag{3-7}$$

Using the fact that $O_j^L = f(NET_j^L)$ and the definition of the error function in Equation (3-1), this equation simplifies to:

$$\delta_j^L = 2 * \frac{1}{2} (O_j^L - T_j) * f'(NET_j^L) \quad . \tag{3-8}$$

Finally, substituting this result back into the equation for calculating $\Delta_p W_{ji}^L$ (3-2) yields:

$$\frac{\partial E_p}{\partial W_{ji}^L} = \delta_j^L * O_j^{L-1} = (O_j^L - T_j) * f'(NET_j^L) * O_i^{L-1} \quad . \tag{3-9}$$

This computes the gradient of the error function with respect to weights in the output layer. Calculating the gradient for weights in the previous layers, however, is a bit more complicated. Again, the chain rule shows that:

$$\frac{\partial E_p}{\partial W_{ji}^{L-1}} = \frac{\partial E_p}{\partial NET_j^{L-1}} * \frac{\partial NET_j^{L-1}}{\partial W_{ji}^{L-1}} \quad . \tag{3-10}$$

Just as in the case of weights in the output layer, the second term is:

$$\frac{\partial NET_j^{L-1}}{\partial W_{ji}^{L-1}} = O_i^{L-2} \quad . \tag{3-11}$$

Consistent with the earlier definition of $\delta_j^L = \partial E_p / \partial NET_j^L$, the first term in Equation (3-10) is $\delta_j^{L-1} = \partial E_p / \partial NET_j^{L-1}$. Thus, Equation (3-10) can be written as:

$$\frac{\partial E_p}{\partial W_{ji}^{L-1}} = \delta_j^{L-1} * O_i^{L-2} \quad . \tag{3-12}$$

Note the similarity between this equation and Equation (3-6). The difference is that here, because the outputs of the nodes in this layer do not appear explicitly in the error function, calculating δ_j^{L-1} requires further application of the chain rule. Doing this yields:

$$\delta_j^{L-1} = \frac{\partial E_p}{\partial NET_j^{L-1}} = \frac{\partial E_p}{\partial O_j^{L-1}} * \frac{\partial O_j^{L-1}}{\partial NET_j^{L-1}} = \sum_k \frac{\partial NET_k^L}{\partial O_j^{L-1}} * \frac{\partial E_p}{\partial NET_k^L} * f'(NET_j^{L-1}) \quad . \tag{3-13}$$

Again, it follows from Equation (3-4) that:

$$\frac{\partial NET_k^L}{\partial O_j^{L-1}} = W_{kj}^L \quad . \tag{3-14}$$

Finally, because $\partial E_p / \partial NET_k^L = \delta_k^L$, Equation (3-13) thus simplifies to:

$$\delta_j^{L-1} = \sum_k W_{kj}^L * \delta_k^L * f'(NET_j^{L-1}) \quad . \tag{3-15}$$

This is a recursive algorithm in that it defines a way to calculate $\delta^\ell = \partial E_p / \partial NET$ (referred to as the δ signal) at any node; simply take a weighted sum of the δ signals of all nodes in the next layer and multiply by the derivative of the activation function. Once the δ_j^ℓ is calculated according to this algorithm for all of the nodes in the network, the $\Delta_p W_{ji}^\ell$ is found to be simply:

$$-\Delta_p W_{ji}^\ell \propto \frac{\partial E_p}{\partial W_{ji}^\ell} = \delta_j^\ell * O_i^{\ell-1} \tag{3-16}$$

where the negative sign is to emphasize that it is the **descent** and not **ascent** of the error function that is to be performed. The derivation is summarized in Table 3-1.

The keys to this result are its simplicity and its generality. No matter how many layers are in the network or how many nodes are in each layer, this gives a simple algorithm for adjusting every single weight so that the error at the output of the network is reduced. The steps in performing the algorithm would then be as follows:

(1) Apply the first training vector to the inputs of the network.

(2) Calculate the δ's associated with the nodes in the output layer according to Equation (3-8):

$$\delta_j^L = (O_j^L - T_j) * f'(NET_j^L) \quad . \tag{3-17}$$

(3) Use these delta values to calculate the delta values of nodes in the previous layer according to Equation (3-15):

$$\delta_j^{L-1} = \sum W_{kj}^L * f'(NET_j^{L-1}) \quad . \tag{3-18}$$

(4) Continue calculating the δ's for previous layers of the network in this way until δ signals have been calculated for all of the nodes in the network.

(5) Calculate the weight change for this particular input vector according to:

$$-\Delta_p W_{ji}^\ell \propto \frac{\partial E_p}{\partial W_{ji}^\ell} = \delta_j^\ell * O_i^{\ell-1} \quad . \tag{3-19}$$

(6) Repeat this procedure for each pattern vector in the training set.

TABLE 3-1

Summary of Back-Propagation Derivation

$$\frac{\partial E_p}{\partial W_{ji}^{\ell}} = \overbrace{\left[\frac{\partial E_p}{\partial NET_j^{\ell}}\right]}^{\delta_j^{\ell}} * \left[\frac{\partial NET_j^{\ell}}{\partial W_{ji}^{\ell}}\right]$$

$$\left[\frac{\partial E_p}{\partial O_j^{\ell}} * \frac{\partial O_j^{\ell}}{\partial NET_j^{\ell}}\right] * \left[O_i^{\ell-1}\right]$$

$$\left[\left(\sum_k \frac{\partial E_p}{\partial NET_k^{\ell+1}} * \frac{\partial NET_k^{\ell+1}}{\partial O_j^{\ell}}\right) * f'(NET_j^{\ell})\right] * \left[O_i^{\ell-1}\right]$$

$$\left[\left(\sum_k \delta_k^{\ell+1} * W_{kj}^{\ell+1}\right) * f'(NET_j^{\ell})\right] * \left[O_i^{\ell-1}\right]$$

Thus:

$$\frac{\partial E_p}{\partial W_{ji}^{\ell}} = \delta_j^{\ell} * O_i^{\ell-1}$$

where

$$\delta_j^{\ell} = \left[\left(\sum_k \delta_k^{\ell+1} * W_{kj}^{\ell+1}\right)\right] * f'(NET_j^{\ell}) \triangleq \frac{\partial E_p}{\partial NET_j^{\ell}} .$$

(7) Update all of the weights in the network by an amount equal to:

$$\Delta W_{ji}^{\ell} = \sum_p \frac{\partial E_p}{\partial W_{ji}^{\ell}} = \sum_p \Delta_p W_{ji}^{\ell} \quad . \tag{3-20}$$

To accelerate performance, the algorithm is often modified so that the weights are adjusted after each training vector application. Because each vector application only yields partial information regarding the gradient of the error function, this method of weight adjustment is inherently inaccurate. To compensate for the limited amount of information available from the application

of just one vector, the weight update will take into account the weight adjustment done during the application of the previous vector. In this case, steps (6) and (7) would be replaced by:

(6) Update all of the weights in the network according to:

$$\Delta W_{ji}^{\ell}(p) = \Delta_p W_{ji}^{\ell} + \alpha * \Delta W_{ji}^{\ell}(p-1) \quad . \tag{3-21}$$

(7) Repeat this procedure for each pattern vector in the training set.

The idea is to change each weight in the direction specified by the present state of the network, but influenced by the direction that the weight was changed when the last vector was being applied. The influence of the previous weight change is determined by the constant α.

B. VECTOR SPACE INTERPRETATION

In order to develop an intuitive understanding of how a back-propagation network represents a problem, consider the pattern classification application discussed in Section A. A good example would be a one-layer network with two inputs and one output. If the two-dimensional input space is mapped to the x-y Cartesian plane and the output of the network is mapped to the z dimension, then a geometric picture of network operation emerges. Because all inputs are limited to the range between 0 and 1, as discussed earlier, the domain of the mapping is a square of unit area in the first quadrant. Referring to the inputs O_0^0 and O_1^0 as x and y, respectively, the value of the network output for any input x and y is $f(x * W_{00}^1 + y * W_{01}^1)$ [where f() is the activation function of the node as discussed in Section A above]. An interesting contour to examine is the intersection of the z = 0.5 plane and the network output mapped into input space as described above. The equation for this contour is the line $x * W_{00}^1 + y * W_{01}^1 = 0$, with z = 0.5. When projected into the x-y plane, this is a line with slope $-(W_{00}^1/W_{01}^1)$. Figure 3-3 shows an example of the mapping that was just described and the intersection of this function with the z = 0.5 plane.

Returning to the pattern classification problem that was discussed in Section 1, this network should be thought of as dividing the input space into two half-spaces for which the output of the one node would be greater than 0.5 in one half-space, and less than 0.5 in the other half-space. The value of the network output for any point in input space depends upon which half-space the point is in, how far the point is from the 0.5 *dividing line*, and the magnitude of W_{01}^1 and W_{00}^1. The magnitudes of the weights determine the steepness of the "step," while the relative values of the two weights determine the orientation of the "step."

Figure 3-3 shows an example where the *dividing line* does not pass through the origin. In order to allow this possibility, an offset is added to the activation function. This offset can be implemented as a weight to a fixed third input with a value of 1.0. The equation for the dividing line is then $x * W_{00}^1 + y * W_{01}^1 + 1.0 * W_{02}^1 = 0$. This is now the equation for a line with slope of $-(W_{00}^1/W_{01}^1)$ and an intercept of $-(W_{02}^1/W_{01}^1)$. The advantage of implementing the threshold as an extra node is that its weight value can be modified in the same way as any other weights in the network. Thus, the threshold of each node can be trained without changing the basic back-propagation algorithm. In all networks in the rest of this report, it will be assumed that each layer (except for the output layer) has an extra *dummy* node with an output fixed at 1.0 which allows the nodes in the next layer to establish nonzero thresholds.

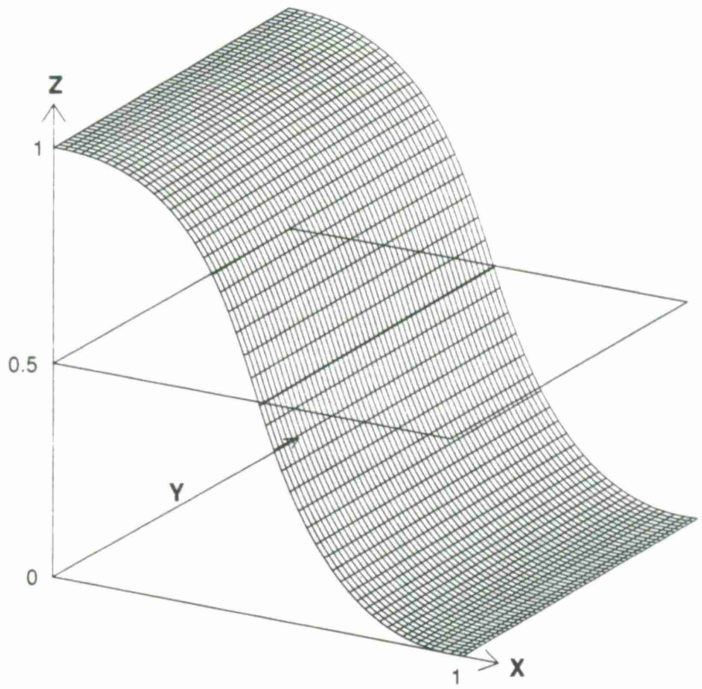


Figure 3-3. Activation function mapped into 2-dimensional input space.

Building on this geometric interpretation of a simple one-layer, *one*-node two-input network, a one-layer two-input network with more than one node would have several dividing lines in input space, one associated with each node. Such a network could be used for some pattern classification problems; however, a network in which each output can divide input space into half planes is useful in defining only the simplest of regions and is not even capable of defining regions as simple as those shown previously in Figures 2-2 and 2-3.

In order to define more complex regions, another layer must be added to the network. The effect is that the second layer can form combinations of half planes. The types of regions that the network can map out in input space are now greatly expanded. The basic principle of examining the 0.5 "dividing line" remains. The nodes in the first layer still define these smooth steps in input space that can be represented by the dividing line, and a measure of how steep the step is. A node in the second layer maps out a function that is dependent on a weighted sum of these smooth steps. As the weights become large, the *smooth* steps become steep steps and in the limit, the *dividing line* represents the transition from the output being 1 and the output being 0. In this limit, the node in the second layer will be able to define any convex polygon region in input space by forming a weighted sum of these dividing lines. In the practical case of smaller weights, and smooth steps in input space, a 2-layer node will be capable of defining a much broader class of convex regions in the same fashion.

Similarly, a three-layer net expands even further the types of regions that can be represented in input space. As a two-layer network can form convex regions in input space, a three-layer net forms combinations of convex regions. With such a net, regions which are connected or unconnected, convex or nonconvex, can be created. Such a network is capable of representing a very broad class of, if not all possible, regions.

Adding more inputs to the network expands the dimensionality of the input space making pictures impossible, but the geometric principles remain. In n-dimensional input space, a one-layer network defines n – 1 dimension hyperplanes that divide the space into two half-spaces. The orientation of the hyperplanes is determined by the relative values of the weights in the network, and the steepness of the division is determined by the absolute magnitude of the weights. Similarly, a two-layer network forms combinations of half-spaces, and a node in the second layer of a two-layer network can define any n-dimensional convex region in input space. Finally, any arbitrary n-dimensional regions can be represented by an n-input three-layer network.

The way that back-propagation networks perform pattern classification can thus be viewed as choosing weight patterns that define hyperplanes in input space, thereby defining the correct regions for the pattern classification problem. It should be clear that in order to construct convex regions a two-layer network is required. It should also be apparent that a minimum number of nodes also are required in the first layer because first-layer nodes can only represent one step in input space and there is no way that the output nodes could use only one half-plane separation to define a closed region like a square.

4. ARCHITECTURAL OVERVIEW OF DESIGN

With an understanding of the back-propagation algorithm and the problems it can solve, it is possible to consider the implementation of the algorithm in dedicated custom hardware. Although specific circuit designs will not be discussed at this point, it is important to define the major blocks that will be required in the implementation and consider the implementation options. This discussion will provide questions, rather than answers, to be addressed by the simulation results that are presented in Section 5. After the basic architecture has been proposed here, the performance of the algorithm will be evaluated in the presence of approximations and other artifacts that will be introduced by the implementation. Once this is done, specific circuits that meet the restrictions revealed by the simulations will be discussed.

It is important to define the major design constraints before defining the architecture. The first design constraint and the most significant is the technology that is to be used. Although there have been proposals in the literature to use many more exotic technologies such as CCD arrays [Agranat, 87] [Chiang, 87] [Sage, 86] and optics [Abu-Mostafa, 87] to implement neural network structures (some of these will be discussed in Section 7), the goal of this project is to use a technology that is readily available, inexpensive, and well understood. Additionally, this project is a part of a larger project, the goal of which is to implement large-scale artificial neural architectures in wafer-scale VLSI. For all these reasons, this design is to be implemented in a standard digital CMOS VLSI process technology as available through the MOSIS foundry. Thus, all circuits considered here will be implementable in such a technology.

Within this technology constraint, the possible architectures range from a fully digital systolic array processor to a fully analog circuit in which the physics of the MOS devices is used to implement the various operations required by the back-propagation algorithm. Each of these approaches has its strengths and weaknesses, and many of them will be discussed in much greater detail in Section 8 which attempts to compare these various implementation strategies. The goal of this design project, however, is to attempt to use the physics of the devices wherever possible to implement the computations. For this reason, a fully digital approach will be used mainly as a reference for comparison with the other circuit approaches. Additionally, it is important that the final design be flexible enough and well enough controlled to make it useful in the investigation of how the back-propagation algorithm works on various problems.

The first block that must be constructed is the basic multi-layered perceptron network that will be used to propagate the input signals forward. This network is composed of blocks that form the weighted sum of many input signals and produce an output after passing through an "activation" function. It is important that the weights are easily modifiable if the network is to learn new weight patterns, and it would be attractive if the network could be initialized from the external world.

The simplest way to implement such a block is with resistors, switches, and operational amplifiers. Such a circuit is shown in Figure 4-1. In such an implementation, the signals are analog voltages that produce currents through the weighting resistors. The size of the resistors

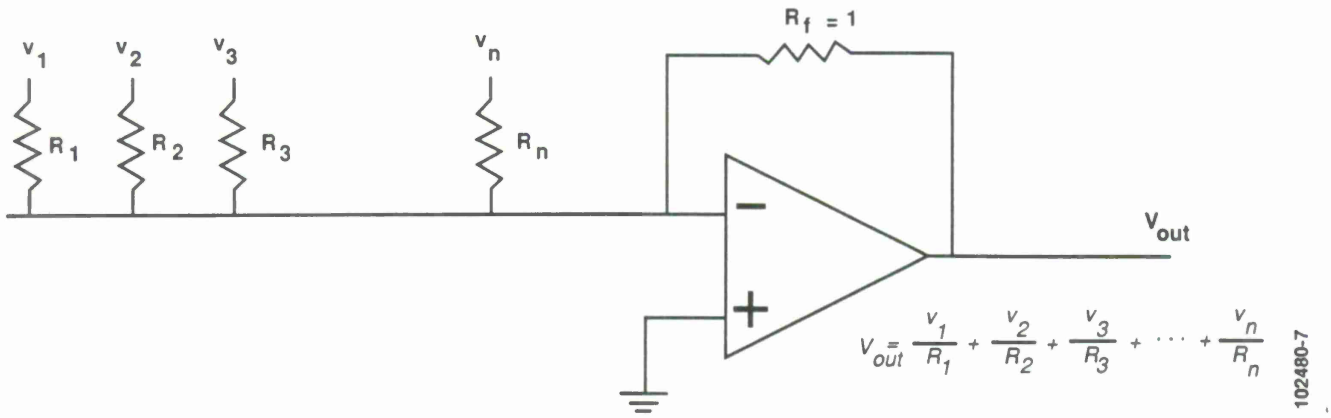


Figure 4-1. Simple Op-Amp and resistor circuit.

determines the strength of the weight in this circuit but the computation in the network is performed in continuous analog signals. In a CMOS process, resistors with programmable weights are not readily available but there are several transistor circuits that closely approximate the resistor's characteristics. Although the specifics of the possible circuit implementation will be discussed in Section 6, the transistor circuits that can be used as programmable resistors can basically be divided into two categories.

The first category of circuits is that in which the value of the resistive connection is digitally controlled. These circuits work on the principle of parallel resistor networks, with each "resistor" circuit having a standard resistance. The value of the weight is determined by how many of the resistor circuits are connected in parallel at a given connection. In these circuits there is generally a digital register associated with each weight and the number stored in that register controls the number of parallel devices at that connection. The advantages of this type of circuit are that it is easy to determine what all the weights in the network are at any time by simply reading these digital registers, and it is easy to set all the weights to a certain weight pattern simply by writing to all the registers. The disadvantage of encoding the weights digitally is that the weight resolution is limited by the size of the registers, and increasing the size of the registers quickly leads to unacceptably large circuits.

The second category of circuits has weights that are controlled by continuous analog signals. In these circuits the resistance is changed by changing the operating region of the circuit that is used to implement the resistor. The obvious advantage of this type of circuitry is that it appears to eliminate the resolution issue that is introduced by digital quantization. Additionally, since small "resistors" don't require many devices operating in parallel, but rather one device operating in a different region, these circuits tend to be significantly smaller than the digital circuits. The disadvantage of such circuits, however, is that it is more difficult to set the weights in the network by writing a stored analog value and it is also quite difficult to read the state at any one point in time. This circuitry is further complicated by the fact that the circuits that control the

values of the weights store the analog signal in the form of charge. This tends to be a dynamic storage in the sense that a value cannot be held indefinitely when the circuitry is implemented in standard CMOS circuitry, so the system must constantly refresh these values and bring them back to their desired levels.

Although it is certainly not clear which of these types of circuits is best since they both seem to have drawbacks, the digitally encoded weighting scheme will be proposed for use in this design. Although the fully analog design approach yields more compact designs (as will be discussed in Section 8), the complex nature of storing analog signals in CMOS technology might compromise the reliability of a back-propagation system.

Choice of word length is the most critical design decision arising from the use of digitally encoded weights. How many bits must the digital register contain in order to represent the weights with sufficient precision to implement the back-propagation algorithm, and how can a circuit be constructed that will have the required precision? This is the principal question to be addressed extensively in Section 5 on simulations and in later sections.

The other architectural issue requiring careful consideration is the technique for implementing the back-propagation learning algorithm. The issue is how to generate the signals for each weight in the network that indicate how to change its value during learning. The learning algorithm requires two signals to be multiplied together in order to determine how to change W_{ji}^{ℓ}. These signals, as shown in Equation (3-19), are $O_i^{\ell-1}$ (the output of the node from which the weight emanates) and δ_j^{ℓ} (the δ signal associated with the node which the weight is entering). The key observation is that the δ signals in one layer are found by simply taking the weighted sum of the δ signals in the next layer and multiplying this weighted sum by $f'(O_j^{\ell})$. What emerges is a network that is propagating the error signal "backward" from the output of the network. Additionally, it is important to notice that the weights used for propagating the error signal backward have the same value as those that were used to propagate the original signal forward.

Now it is possible to understand the basic structure of the back-propagation architecture as presented in Figure 4-2 which shows one layer of a back-propagation network. This structure is modular in the sense that these modules can be connected directly together to form a multi-layer network. The basic structure as shown in Figure 4-2 contains a forward network that is propagating the signals in from the left and down to the bottom of the figure (drawn with thin lines), and a backward network intertwined with the forward structure that is propagating the error signals back from the outputs at the bottom and out to the left of the figure (drawn with thick lines). The nodes of this layer are at the bottom of the picture, and the weights are arranged in an array with the columns containing all the forward weights that go to a particular node. Equivalently, a row contains all the weights that go from a particular input. At each location in the weight array is a weight structure that is composed of three sub-blocks. Taking advantage of the fact that the corresponding weights in the forward and backward networks always have the same value, the weight structure has one digital register to establish the size of the weight, and two other blocks — one presents this weight to the forward network, the other to the backward network.

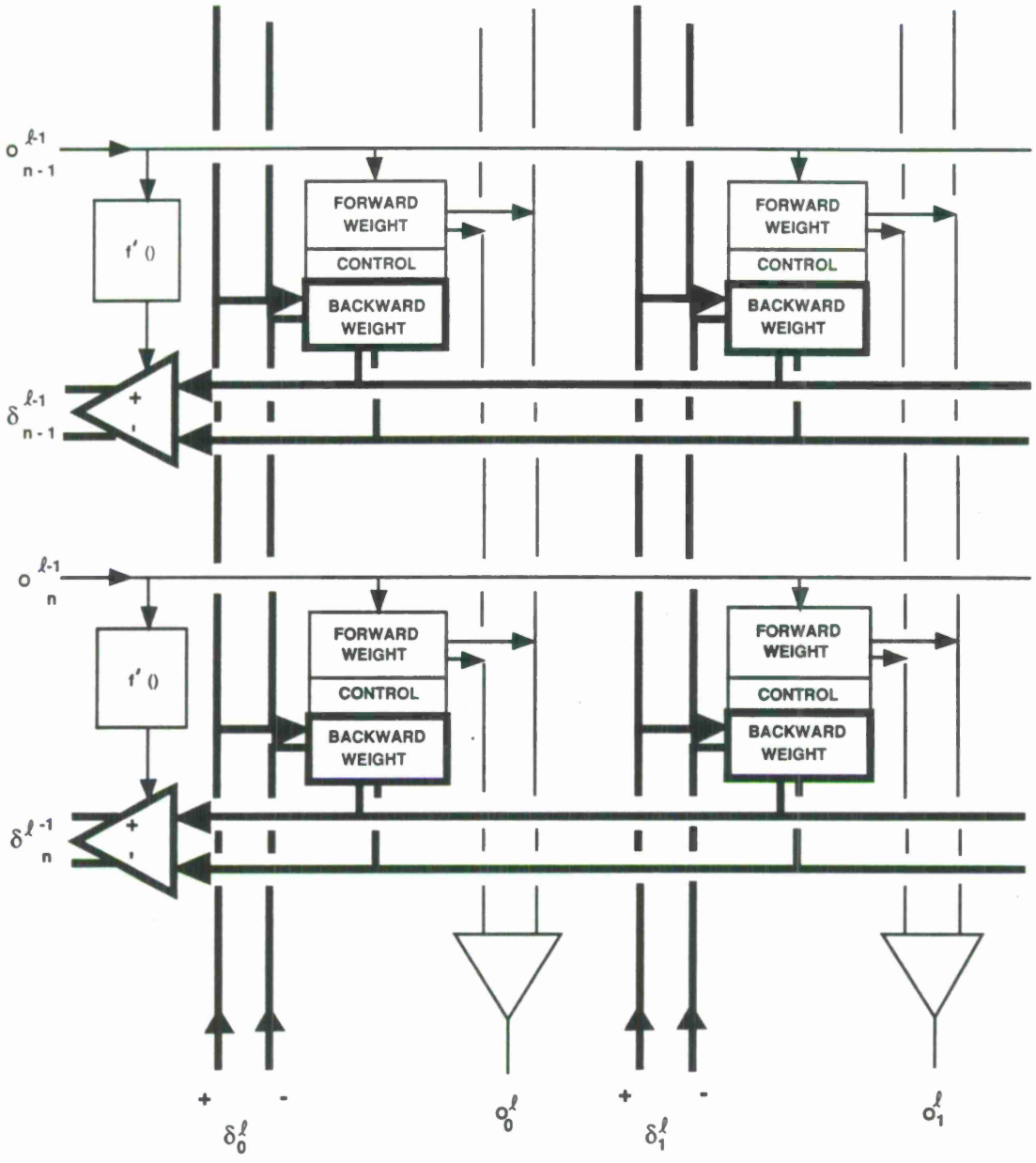


Figure 4-2. Block architecture of design.

Another important concept behind this architecture is that all the signals required by the back-propagation algorithm in order to perform learning are always available at the appropriate weight structures in the form of analog signals. The signal that is feeding W_{ji} in the forward direction is $O_i^{\ell-1}$, and the signal that is fed through this weight in the backward direction is δ_j^{ℓ}. Thus, the calculation required to determine the weight change involves only signals that are already available at the weight structure.

A node in this structure will be constructed from an amplifier that sums the input signals until they exceed a certain threshold, after which the amplifier saturates. This will produce the desired activation-function characteristic at the nodes. The back-propagation algorithm specifies that the δ signal must be multiplied by the derivative of this activation function, evaluated at the operating point of the amplifier. This multiplication is represented by a box in the block diagram labeled f'( ) but, because the activation function may not be an analytic function, an important question to be addressed in the simulations is how closely the output of this box must match the derivative of the activation function of the amplifier in the network.

5. SIMULATIONS

The general architecture proposed in Section 4 raised the critical questions of weight resolution and accuracy of approximation of the derivative of the activation function. To investigate the influence of these factors on the back-propagation algorithm performance, a simulation program, originally written by William Huang of Lincoln Laboratory, was rewritten to simulate the effects of the implementational artifacts. The simulator was written in C and run on a VAX 11780 and the outline of these simulations and the results will be presented in this section. A more detailed description of these simulations is presented in Appendix A.

The goal of these simulations is to determine how certain artifacts influence the performance of these networks on prototypical problems and to determine when these effects become unacceptably severe. Thus, the problems that were simulated were chosen using two basic criteria:

(a) They were simple enough that the effects which were being examined could be observed and understood.

(b) They were thought to contain the essential features of more complicated and intrinsically interesting problems.

A. PROBLEM DEFINITION

Within the broad class of pattern classification which characterizes all of the simulated problems, there are finer distinctions to be made. The first distinction among pattern classification problems is between those that have binary (0 or 1) input signals, and those that have continuous (any value between 0 and 1) input signals. The distinction between these two problems is seen when the pattern classification problem is observed in the theoretical "vector space" framework as discussed in Section 3. The simplest example would be a network that had only two inputs. The input space for such a problem, when represented as a plot in Cartesian coordinates, is a unit square in the first quadrant. A problem that had binary inputs would only be able to specify the class of each of the corners of the square [(0,0) (0,1) (1,1) ...], whereas a problem with analog inputs could specify any arbitrary regions in this input space (such as those shown in Figure 2-2).

The significance of this distinction is the "resolution" of the class regions. In a problem that has digital inputs, there is considerable room in the input space between the points of interest. If each node in the first layer of a network divides the space into two half-planes (as in Section 3), then it can be seen that when there is considerable space between the points in two different regions, the "dividing line" can move to many different places and still have the desired effect. If, however, the inputs are analog, every point in input space may have a well-defined desired classification and any movement of the "dividing line" may cause some points to be reclassified.

Another important distinction between different types of pattern classification problems arises from differences in the desired outputs. Although, in Section 2, pattern classification problems were defined generically to be those that have an output code determined by the input class, the coding used has ramifications for the performance of the network. As mentioned in Section 2,

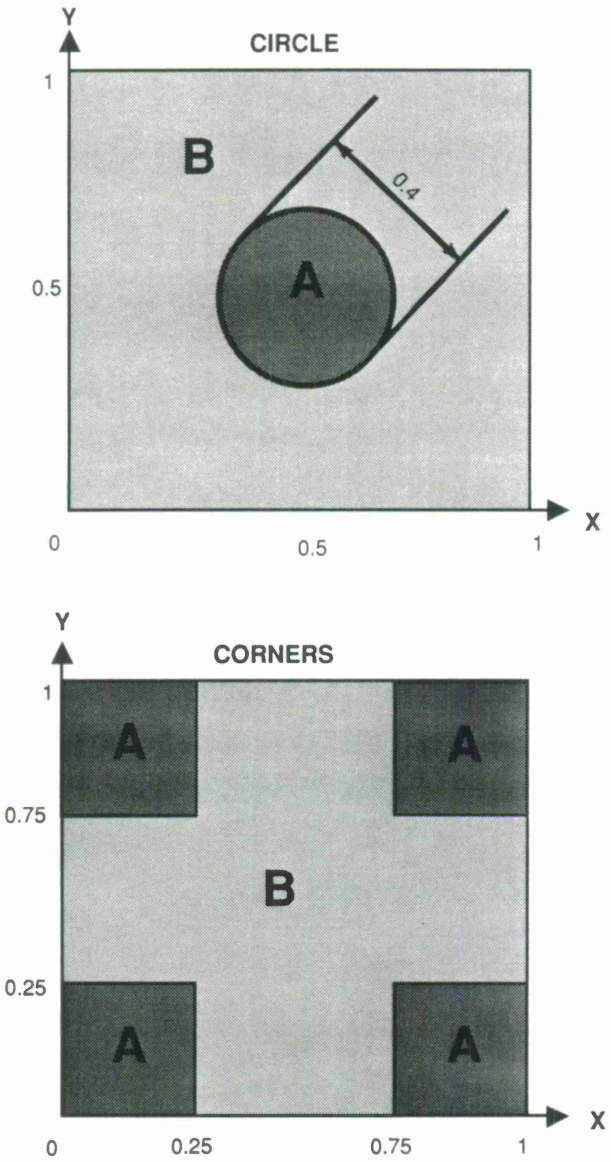


Figure 5-1. Classification regions in input space for simulated problems.

the simulations done in this project assume that there is a separate output node for each class and, although the desired output code has one output node with a value of 1 and all the others at 0, the pattern will be classified by observing which of the output nodes has the largest value. Another coding scheme that has been used by some researchers encodes each class with a binary code, of which each network output is a bit. In this coding scheme, a threshold of 0.5 is generally used so that if the output is above the threshold then it is considered to be a 1, and if it is below the threshold then it is considered to be a 0. Although these two methods of coding seem to accomplish the same goal, the classification results will be different for points that lie near region boundaries.

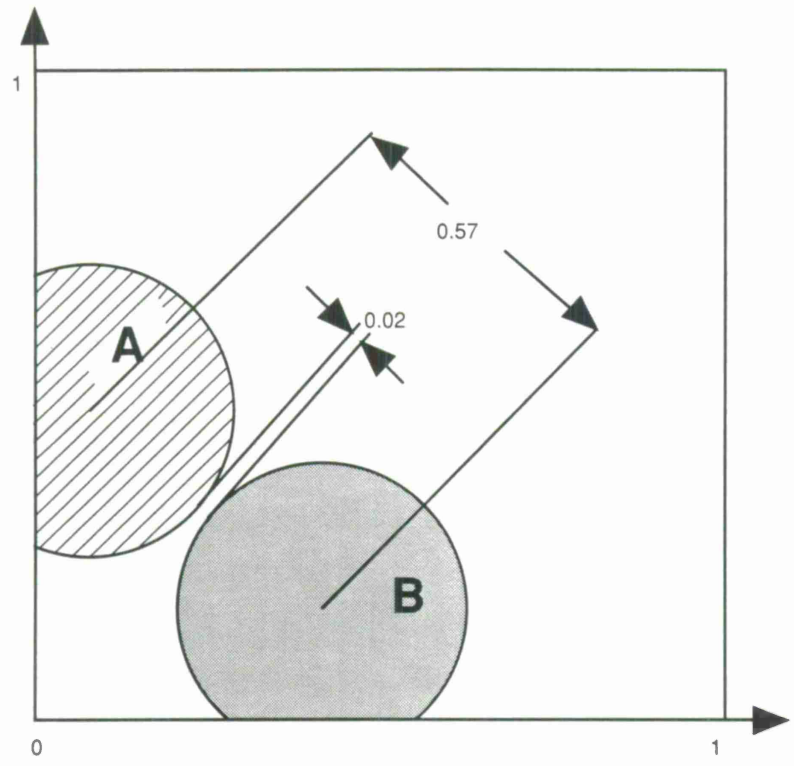
Three problems were simulated extensively, and the results of these simulations were used to determine the capability of particular hardware designs. One of the most important goals of the simulations was to determine how the weights in the network change during learning, and what effect restricting these changes in different ways has on the performance of the back-propagation algorithm. Two of the simulated problems had only two inputs so that the input space could be mapped graphically to the x-y coordinate system to facilitate a conceptual understanding of the network performance. The inputs to these two problems were analog values that ranged between 0 and 1, and the class regions for these two problems are shown in Figure 5-1. The third problem that was simulated had ten input dimensions, and five classes composed of balls in 10-space (all inputs are still analog and confined to the 0-1 input space). The centers of the five hyperspheres were equidistant while the space between any two regions was small. Figure 5-2 shows graphically what a 2-dimensional equivalent to this problem would be with only two class regions (this figure is only to help the reader conceptualize the shape of the actual 10-dimensional problem). The training data for each of these problems consisted of an equal number of data points chosen randomly from each of the class regions in the problem. It is important to note that, in addition to the fact that the third problem has 10 inputs while the others have 2, there is another significant difference between them. While the class regions in the "circle" and "corner" problems abut, there is space between the defined class regions of the 10-dimensional problem for which no class is defined. As mentioned in Section 2, this undefined space puts a much less precise constraint on the placement of the class boundaries and this will be reflected in the simulation results.

B. THE QUESTIONS AND RESULTS

The goal of the simulations was to discover what effects the anticipated hardware artifacts would have on the performance of the back-propagation algorithm. The artifacts that were considered should first be separated into two categories: those that affect representation, and those that affect learning. Although some artifacts affect both representation and learning, these effects will be viewed independently.

1. Quantized Representation

The effects of weight quantization on **representation** were investigated first, and these results provide a frame of reference for the investigation of the weight quantization effects on **learning**.

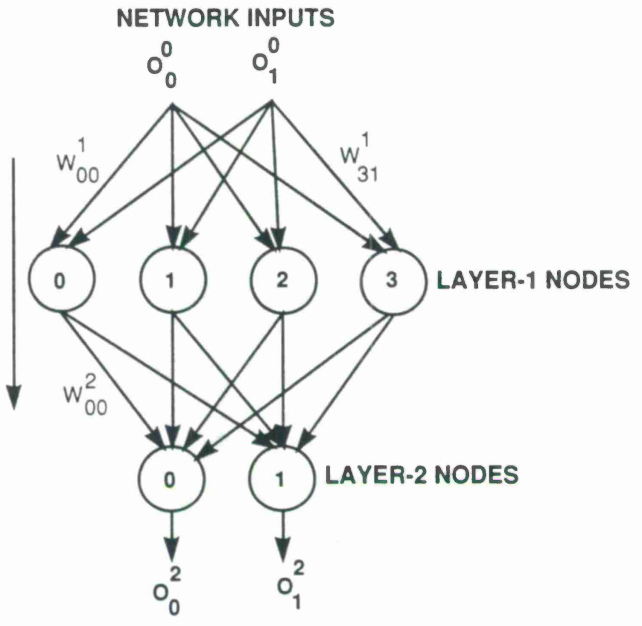


102480-10

Figure 5-2. 2-dimensional equivalent of 10-dimensional problem.

To investigate this question, back-propagation was used to learn a weight pattern for a particular network and problem with high-precision weights and then the network performance was measured as a function of weight quantization. The network architecture used in these experiments had two layers, and the number of nodes in the hidden layer was an experimental parameter. A typical architecture with four nodes in the hidden layer is shown in Figure 5-3.

This question brings up a subtle, but significant, issue that is related to the idea of weight quantization. It is important to recognize that, in a back-propagation network, specifying the number of bits of quantization is not sufficient; it is also essential to specify the **range** of weight values to be represented. The reason that this is an essential point is that for a given problem, regardless of the number of bits used in the representation, the problem cannot be solved with that system if the weights cannot represent large enough values. On the other hand, if the weights can take on a wide range of values, but have only a very small number of bits of resolution, then the net will still be unable to represent many problems because the error in weight representation will be too large. In order to emphasize the significance of this point, quantization will be measured in this section in terms of both the number of bits used (including sign) and the magnitude of largest representable value.



102480-11

Figure 5-3. 2-layer network (2 inputs, 2 outputs, and 4 hidden nodes).

In the simulations, the range of the quantized weights used for a particular problem was determined by the largest weight in the high-precision weight set. In fact, each layer of the network was quantized to a different range as determined by the largest weight in that layer (it was assumed that the same number of bits was available to represent the weights in each layer). The training data were then applied to a network with the quantized weight pattern and the number of points in the training set that had been misclassified by the quantization was used to measure the effect of the weight quantization. Additionally, in the problems that had two input dimensions, the entire weight space was scanned with very fine resolution and the percentage of the input space that was classified differently by the quantized and the unquantized weight patterns was used as a more precise measure of the effects of weight quantization. The results are summarized in Tables 5-1(a) through (c).

There are a number of things worth noting in these tables. First of all, it should be noticed that the "10-Dimensional" problem had the lowest misclassification rate with and without quantization due to the fact that the class regions in this problem do not abut. The result is that, even when the weights were quantized (restricting the location of dividing lines to discrete orientations and locations), the network was able to define regions of high-enough resolution to correctly classify all the training vectors. In the "Circle" and "Corner" problems, however, the regions covered the entire space so the boundaries had to be defined in exactly the right place in order to correctly classify all the training patterns.

The initial weight patterns for this experiment were learned with very high precision; however, in the "Circle" and "Corner" problems, it was difficult to decide when the learning process should be stopped. Although the first few passes through the training data caused the weights to change rather significantly until a rough outline of the decision regions had been formed, the learning process then continued at a progressively slower rate. Later, learning was generally characterized by little movement of the "dividing" lines in input space, but a steady increase in the magnitude of the weights, and a slight refining of the positions of the "dividing" lines.

In these experiments, learning was continued until two objectives were met. The first was that the percentage of misclassified training vectors fell below 5 percent, and the second was that the rate of change of this misclassification percentage from one pass to the next approached zero. Continued learning would have improved performance further; however, for the purposes of this experiment, it was the difference between the quantized-weight and high-precision-weight performance that was most significant and the absolute performance of the high-precision-weight pattern was less important. The fifth column of Tables 5-1(a) through (c) indicates the performance of the weight pattern before quantization.

TABLE 5-1(a)

Percentage Reclassification After Quantization for *"Circle" Problem*

Number of Hidden Nodes	No. of Bits* Used	Layer-1 Maximum Weight	Layer-2 Maximum Weight	Training Vectors Misclassified with High-Precision Weights (percent)	Training Vectors Misclassified with Quantization (percent)	Percent Input Field Reclassified
4	4	18.2	9.8	2.2	26.8	9
4	5				6.2	3
4	6				1.8	2
4	8				2.0	<1
8	4	17.8†	9.9	2.4	6.6	5
8	5				6.0	6
8	6				2.8	2
8	8				3.4	<1
12	4	17.6	10.1	3.2	30.6	9
12	5				3.8	3
12	6				3.2	1
12	8				2.6	<1

* Including sign bit.

† The entire weight pattern for this run is shown in Appendix A.

TABLE 5-1(b)

Percentage Reclassification After Quantization for *"Corner" Problem*

Number of Hidden Nodes	No. of Bits* Used	Layer-1 Maximum Weight	Layer-2 Maximum Weight	Training Vectors Misclassified with High-Precision Weights (percent)	Training Vectors Misclassified with Quantized Weights (percent)	Percent Input Space Reclassified
4	4	26.2	14.4	2.6	8.6	12
4	5				4.2	4
4	6				2.6	2
4	8				2.8	1
8	4	26.2†	14.2	2.6	6.0	7
8	5				6.0	5
8	6				3.0	3
8	8				2.8	1
12	4	26.2	14.1	2.6	6.8	8
12	5				5.2	4
12	6				6.8	2
12	8				2.8	1

* Including sign bit.

† The entire weight pattern for this run is shown in Appendix A.

TABLE 5-1(c)

Percentage Reclassification After Quantization for the *"10-Dimensional" Problem*

Number of Hidden Nodes	No. of Bits* Used	Layer-1 Maximum Weight	Layer-2 Maximum Weight	Training Vectors Misclassified with High-Precision Weights (percent)	Training Vectors Misclassified with Quantized Weights (percent)
4	4	10.1	8.3	0	1.5
4	5				0
4	6				0
4	8				0
8	4	6.3	7.1	0	0.6
8	5				0
8	6				0
8	8				0
12	4	6.6	6.3	0	0
12	5				0
12	6				0
12	8				0

* Including sign bit.

Although, as discussed above, word length is not a sufficient metric for determining solvability, it is an important factor for determining implementability. As Tables 5-1 indicate, all the test problems exhibited less than 5-percent reclassification when the weights were quantized with 6 bits over a range that included the largest unquantized weight. Because the largest weights in the simulated problems tended to be about 16, this translates into a minimum step size of about 0.5. It is worth noting that these results are best-case in the sense that the weight range was known in advance, and the representable range was set to match the required range exactly. Additionally, in these simulations the weight range was set separately for each layer. In order to achieve comparable performance in a circuit implementation, the range that each layer can represent would have to be adjustable.

2. Range-Limited Learning

As discussed in the previous section, there are two effects that result from quantizing the weights in a network. The first is that the weights can only be changed by discrete values of a given size, and the second is that the range of weights that can be represented is limited. In an attempt to separate the influence of these two effects, the first learning simulations involved range limiting with unquantized weights.

In these simulations, different results were obtained with different problems, and when different architectures were used to implement the same problem. In some cases, restricting the range seemed to cause learning to stop as soon as one of the weights reached the range limit, and sometimes learning continued until several weights hit the range limit, and then stopped. In these cases, the final weight pattern appeared to be very much like the pattern that was learned without range limitations but with all the large weights clipped at the range limit.

Surprisingly, limiting the range of the weights sometimes caused substantial differences in the final weight pattern. When some of the weights hit the limit in the network that was learning with range limits, all the weights would not just stop increasing; rather, the network would learn a weight pattern that was not like the pattern learned without range limitations. A particular instance of the learning in a new direction occurred when the circle pattern was learned by a network that had 12 nodes in the first layer of a two-layer network. From the perspective of dividing lines in input space, the network learned a weight pattern that only relied on the dividing lines introduced by four of the hidden nodes until some of the weights reached the range limit. The network that had no range limit continued to refine the position of these dividing lines and increase the steepness of the step that they introduced, but the network that had range limitations began to recruit more of the dividing lines introduced by other hidden nodes.

This adaptation to range limitation did not occur in all cases, and it was sometimes observed that limiting the range simply stopped the learning process. This will make it very difficult for any hardware network to learn these problems. It does seem, however, that by increasing the number of hidden nodes in a network, the network can (in some cases) compensate for range restrictions on the network weights.

The most severe difficulty that results from the fact that different problems require different weight ranges is the fact that no method has been developed to determine, in advance, the range of weights required for a particular problem. In the simulations this was always determined empirically by simulating with no range limitation first. The implication for implementation is that a robust system will require that there be some mechanism for adjusting the range of representable weights for the given number of bits of resolution.

3. Fully Quantized Learning

Learning weight patterns in a network with quantized **and** range-limited weights was the most complicated task that was simulated. In this situation, it was found that the size of the "unit increment" determined the success of the simulation. Once again, it was a trade-off between range and resolution. If the weights were quantized to a small range, then the unit increment was usually acceptably small so that the network could begin learning correctly; however, the weights would soon reach the range limit and this would restrict how well the network could learn the pattern. If the range were larger, then the unit increment that was associated with a given resolution might be too large for learning to proceed at all. The successful simulations thus required a careful trade-off between these two factors.

In simulations that began with very coarse quantization or very large ranges, the learning coefficient had to be quite large before any of the weights would change at all. When they did change, however, the steps were often too large and the result was that the network could not learn a solution to the problem and the weights just continued to fluctuate (generally only a small subset of the weights was changing and only over a very limited range).

More successful simulations would approach a solution to a point where only a few percent of the training vectors were not classified correctly. At this point the weights would continue to change, but they would oscillate about a specific value, never really getting much larger or much smaller, and never really improving or degrading the performance of the network any more. This generally occurred after some of the weights had reached the range limit.

Tables 5-2(a) through (c) show some of the convergence data which indicate how quickly the network was able to converge to a solution to a particular problem with a specific range/resolution and how good the eventual solution was. Although "convergence" to a solution is a subjective measure of performance, it was found that there was a very clear distinction between networks that had "converged" to a partial solution to a problem, and those that simply had not solved the problem. Problems that "converged" to a solution formed regions in space that roughly matched the class regions in the training data. These problems did not necessarily find the exact regions, but examination of the regions that they did form revealed that they were of the same basic shape and location as those in the training data. Problems that did not converge often just classified the entire input space as one class; other times, regions were formed that simply did not match those in the input data. With the high-dimensional problem in which the class regions did not span the input space, the network was able to converge to a solution in which all data points were classified correctly. In the problems in which the regions did span the entire input space, a 100-percent correct solution was never reached.

TABLE 5-2(a)

Quantized Learning with the *"Circle" Problem*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Observed Weight	Layer-2 Maximum Observed Weight	No. of Training Vectors Applied	Percent Misclassified
2-8-2	No* quantization	20	11	15,000	2.4
2-8-2	8	8†	8†	15,000	2.2
2-8-2	6	8†	8†	20,000	7.2

* 32-bit precision of computer.

† Maximum weight limited to 12.

TABLE 5-2(b)

Quantized Learning with the *"Corner" Problem*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Observed Weight	Layer-2 Maximum Observed Weight	No. of Training Vectors Applied	Percent Misclassified
2-8-2	No* quantization	23.3	13.1	30,000	2.8
2-8-2	8	12†	11.3†	20,000	8.0
2-8-2	6	No convergence			

* 32-bit precision of computer.

† Maximum weight limited to 12.

TABLE 5-2(c)

Quantized Learning with the *"10-Dimensional" Problem*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Observed Weight	Layer-2 Maximum Observed Weight	No. of Training Vectors Applied	Percent Misclassified
10-8-5	No* quantization	4.25	4.89	2,000	0
10-8-5	8	4	4	10,000	0
10-8-5	6	No convergence			

* 32-bit precision of computer.

In learning experiments, a large number of parameters that can be adjusted include: the number of hidden nodes, the learning rate, the order of training vector application, the number of network layers, and (when the weights are being quantized) the number of bits used and the weight range. Determining the optimum setting of all these parameters is quite complicated, although a set of values was determined which seemed to work acceptably on most problems (these values are described in Appendix A). On some problems, however, these standard settings did not work, and they were adjusted in an attempt to optimize performance. A table entry of "No Convergence" indicates that no solution was learned with any combination of these parameters that was tried.

These results show that, although 6 bits were sufficient to represent a pattern that had already been learned (see Section 5-B-1), it did not provide sufficient resolution to perform the learning. Even in the one case where 6 bits did result in a solution to the "Circle" problem, this was only achieved after a careful adjustment of the parameters. Although 8 bits were sufficient to allow the network to converge to a solution in most cases, for the "Corner" problem the range had to be limited to ± 12 in order to find a solution with only 8 bits. With this range limitation, the best solution that was learned misclassified 8 percent of the training vectors. Further simulation revealed that, even when high-precision weights were used, this range limitation prevents the problem from being solved to better than 5-percent misclassification.

To summarize, fewer than 8 bits prevented the network from learning weight patterns in almost all cases. To learn most of the test problems, 8 bits provided sufficient resolution; however, more bits might be needed to learn problems that required large weights such as the "Corner" problem. The most important figure seems to be the unit step size, and in these problems it was found that this had to be about 0.1 or smaller to permit learning.

4. Slope Approximation

One of the anticipated problems with an analog implementation of the back-propagation algorithm is the determination of the derivative of the activation function of the nodes. In anticipation of this being an extremely difficult calculation to perform in analog hardware, simulations were done that attempted to examine the effects of approximating this value.

The activation function that was used in these simulations was the one described in [Rumelhart, 86] and shown in Figure 3-2. The derivative is shown in Figure 5-4 along with a function that approximates this derivative with a pair of threshold functions. The approximation can take on only one of two possible values. If the output of a node is very large or very small, then the derivative will be approximated by a small value; but, if the output is in the middle of the range, then the derivative will be approximated with a larger value. This very simple approximation can easily be implemented in hardware, and Figure 5-5 shows that the activation function which is described by the derivative approximation is very similar to the sigmoidal activation function that was used in these simulations. The simulation results are shown in Tables 5-3(a) through (c).

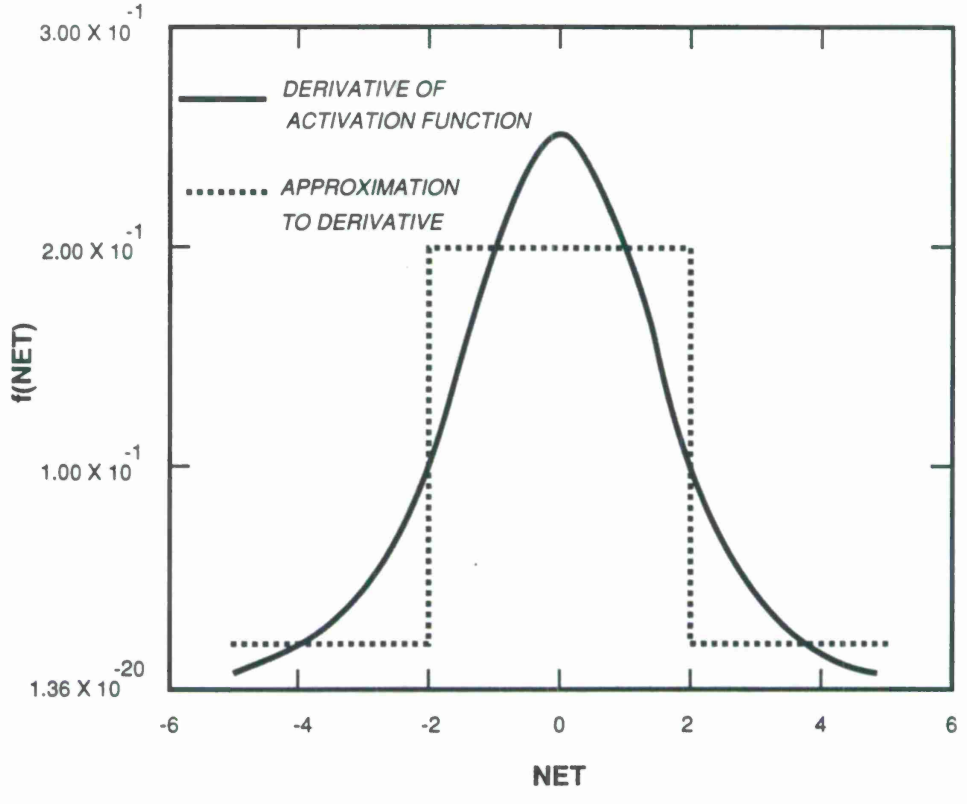


Figure 5-4. Derivative of activation function and approximation.

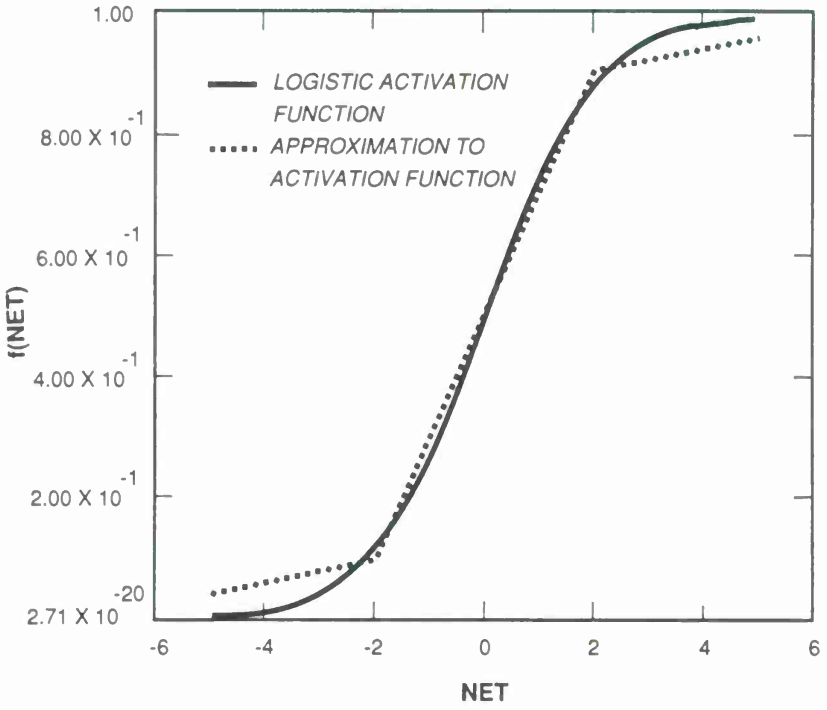


Figure 5-5. Comparison of activation function and approximation.

TABLE 5-3(a)

Quantized Learning with *Slope Approximation ("Circle" Problem)*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Weight	Layer-2 Maximum Weight	No. of Training Vectors Applied	Percent Misclassified
2-8-2	No* quantization	14.8	9.7	15,000	2.0
2-8-2	8	8	8	15,000	4.0
2-8-2	6	No convergence			

* 32-bit precision of computer.

TABLE 5-3(b)

Quantized Learning with *Slope Approximation ("Corner" Problem)*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Weight	Layer-2 Maximum Weight	No. of Training Vectors Applied	Percent Misclassified
2-8-2	No* quantization	27.5	11.2	30,000	3.4
2-8-2	8	12.0	6.8	20,000	9.8
2-8-2	6	No convergence			

* 32-bit precision of computer.

TABLE 5-3(c)

Quantized Learning with *Slope Approximation ("10-Dimensional" Problem)*

Architecture of Network	No. of Bits Used	Layer-1 Maximum Observed Weight	Layer-2 Maximum Observed Weight	No. of Training Vectors Applied	Percent Misclassified
10-8-5	No* quantization	5.6	5.7	2,000	0
10-8-5	8	4	4	10,000	0
10-8-5	6	No convergence			

* 32-bit precision of computer.

These results should be compared with the data in Tables 5-2(a) through (c). This comparison reveals that, although the slope approximation seems to result in a performance degradation as measured in terms of the percentage of training vectors that are misclassified after a given amount of learning, the networks are still capable of learning the experimental problems with 8 bits of weight quantization. One noticeable difference in the two sets of results is that the network was no longer able to find a solution to the "Circle" problem with only 6 bits of resolution when the slope approximation was used.

5. Nonrandom Initialization

Most of the simulation work in this project was done to determine the effects of hardware artifacts on the back-propagation algorithm. During these simulations, the issue of weight initialization arose and a new weight initialization algorithm was developed. Conventionally, the weights are initialized to random values that are usually restricted to some range. Although this randomness seems to start the network in a "blank" state, a nonrandom algorithm which takes advantage of known restrictions on the input data might be more effective.

In all the problems that were simulated here, all input variables were restricted to values between 0 and 1 and the networks had two layers. As discussed in Section 3, the weights in the first layer of the network determine the orientation and steepness of the "smooth step" output function of a first-layer node as mapped into input space. Additionally, the half-way point of this output function maps into a hyperplane that divides the input space into two half-spaces. When the weights are initialized randomly, this hyperplane is not guaranteed to even pass through the region of input space to which all of the input data have been restricted. With this in mind, a nonrandom weight initialization algorithm was developed which requires all these dividing hyperplanes to pass through the middle of the input space to which all input data points are restricted.

Specifically, consider a network with n inputs. A given node in the first layer will have n + 1 weights associated with it: one to each of the inputs, and one to the "always one" node that is used to generate an offset. The weights from the inputs define the orientation of a dividing hyperplane which passes through the origin in the input n-space. These weights can be thought of as an n-dimensional vector that is normal to this hyperplane, and the magnitude of this vector will determine the steepness of the output function of the node as mapped into this input space. The nonrandom weight initialization algorithm allows these n weights to be selected randomly, but they are then scaled uniformly so that this normal vector has a predetermined magnitude. The weight to the offset node can then be chosen such that the hyperplane passes through the point (0.5,0.5,...,0.5). Once all weights in the first layer are initialized in this way, the weights in the second layer are initialized uniformly to a number that is $1/m$, where m is the number of nodes in the first layer. This prevents the second-layer nodes from receiving extremely large positive or negative inputs at first so that they can most readily learn to become sensitive to the nodes in the first layer that have the desired orientations.

In the simulations of the 2-dimensional problems (where there are two inputs to the network), the dividing hyperplanes are lines, and the algorithm that was used generated weights so

that these dividing lines not only passed through the center of the input space, but so that the orientations of the dividing lines introduced by the first-layer nodes actually spanned 360°. This nonrandom initialization method was not used for the "10-Dimensional" problem because it was developed after those simulations had been completed.

6. Simulation Conclusions

To summarize, the simulations yielded several important results. First of all it became clear that because some problems require weights with a large dynamic range, the ability to adjust the dynamic range, at the expense of resolution, would improve the versatility of any implementation. Still greater versatility is achievable if this can be done on a layer-by-layer basis.

A second result was the observation that there is a disparity between the resolution required for weight representation during learning and that required to represent a previously learned pattern. It was found that a minimum step size of about 0.5, which translates to 6 bits including the sign bit in the simulated problems, gave performance that was considered to be acceptable for representing previously learned weight patterns. A finer resolution of about 0.1, which corresponds to 8 bits in the simulated problems, was required during learning.

Lastly, the simulations also demonstrated that a very simple slope approximation technique degraded network performance. The effect, however, seems relatively minor.

A final observation that resulted from the simulations was that, in order to have certain networks perform correctly, several learning parameters had to be carefully adjusted. The implication is that a successful implementation will require that these parameters be adjustable.

6. SPECIFIC SUB-CIRCUITS

A. PROCESSOR

The basic circuit required to implement a back-propagation network forms the weighted sum of a number of inputs and passes this through a nonlinear activation function. Two approaches to the design of this circuitry have been investigated, both of which send analog signals through digitally controlled weighting circuitry. These circuits should be thought of as Multiplying Digital to Analog Converters (or MDACs) because they convert digital weights to an analog conductance, and multiply this value by the incoming analog signal.

1. Voltage-Driven MDACs

a. Circuit Overview

The first circuit that was examined was proposed by Jack Raffel at Lincoln Laboratory [Raffel, 87]. This circuit represents the O_j^{ℓ} signals as voltages which drive the sources of MOS devices operating in the linear region that act as weights. Figure 6-1 shows a block diagram of this circuit. A similar circuit has been designed for implementing a Boltzmann Machine [Alspector, 87]. Although this figure depicts only one node with two weights for illustration, the fan-in can be expanded by simply adding more weight circuits.

In this design, signed weights are implemented by developing two summing currents, one excitory (I_e) and one inhibitory (I_i) and taking their difference. A positive weight develops a current in the excitation sum, and a negative weight develops a current in the inhibition sum. The current on the excitation line is subtracted from the current of the inhibition summing line with the first Op-Amp in Figure 6-1, and this net current is input to a transimpedance amplifier. The output voltage signal of this amplifier is proportional to the difference between the excitation and inhibition currents. This circuit allows 2-quadrant operation in the sense that the input voltage, which has to be positive, can be multiplied by either a positive or negative weight.

A schematic of the weight circuitry used in this design is shown in Figure 6-2. The conductance of the weight is modulated by digitally programming the effective width of the MOS device. In this particular design, the weight is digitally encoded with a 5-bit (including sign) register. Because the gate voltage is to be driven by a 5-V dc digital signal, it is important that the drain voltage of the active devices stay well below this level to assure that the devices remain in the linear region of operation. Thus, the 0 to 1 range of the signals must be mapped to something like a 0- to 1-V range in the voltages used to represent these signals.

b. Performance

This design was implemented by Jim Mann of the Digital Integrated Circuits Group at Lincoln Laboratory and fabricated through the MOSIS foundry. The performance of this design was investigated as a part of this research project by implementing a simple Hamming network with the circuit. That evaluation and the results that came from it were reported in [Raffel, 87]. Those results are summarized here.

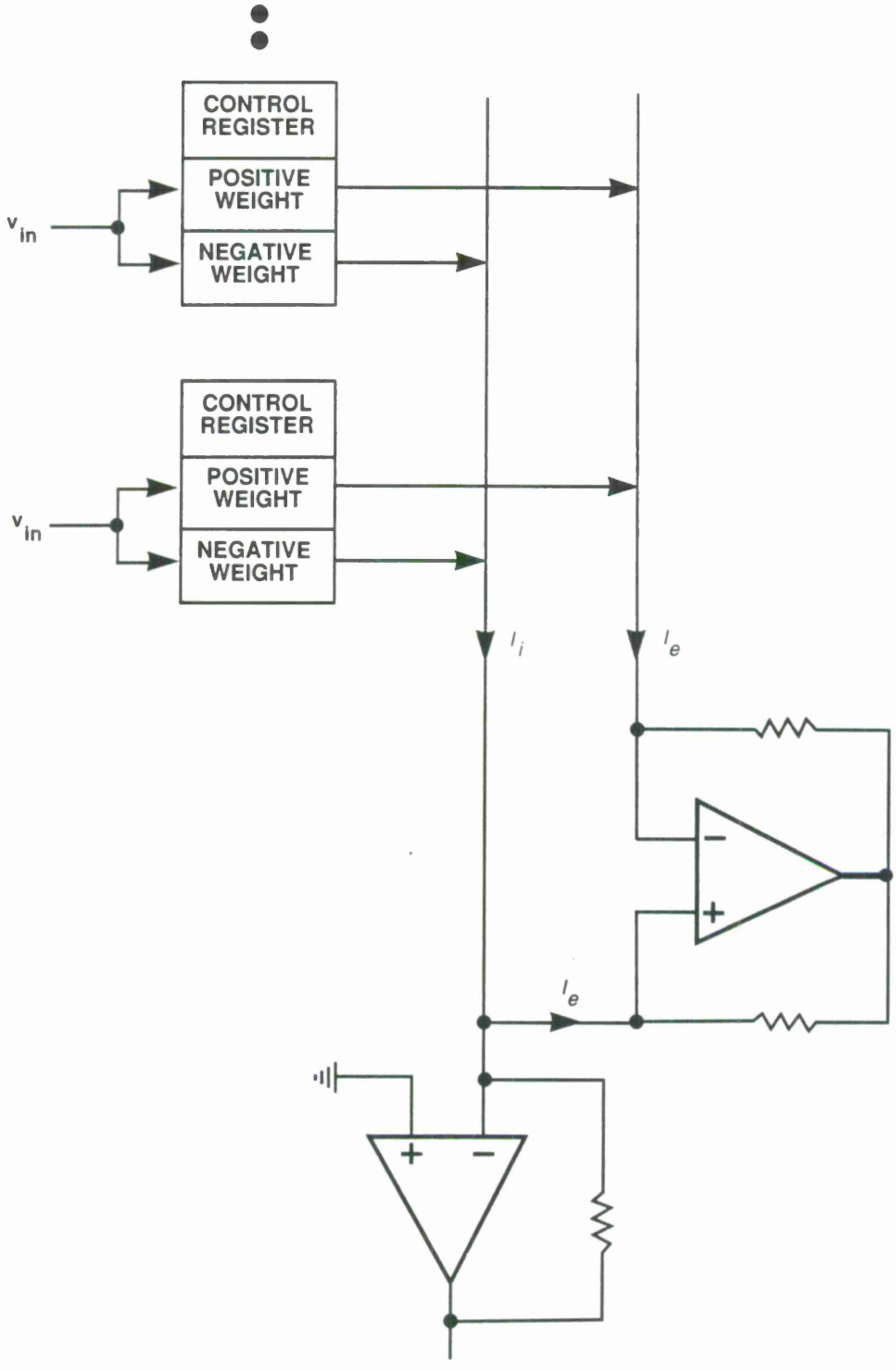


Figure 6-1. Block diagram of voltage-driven MDAC circuit.

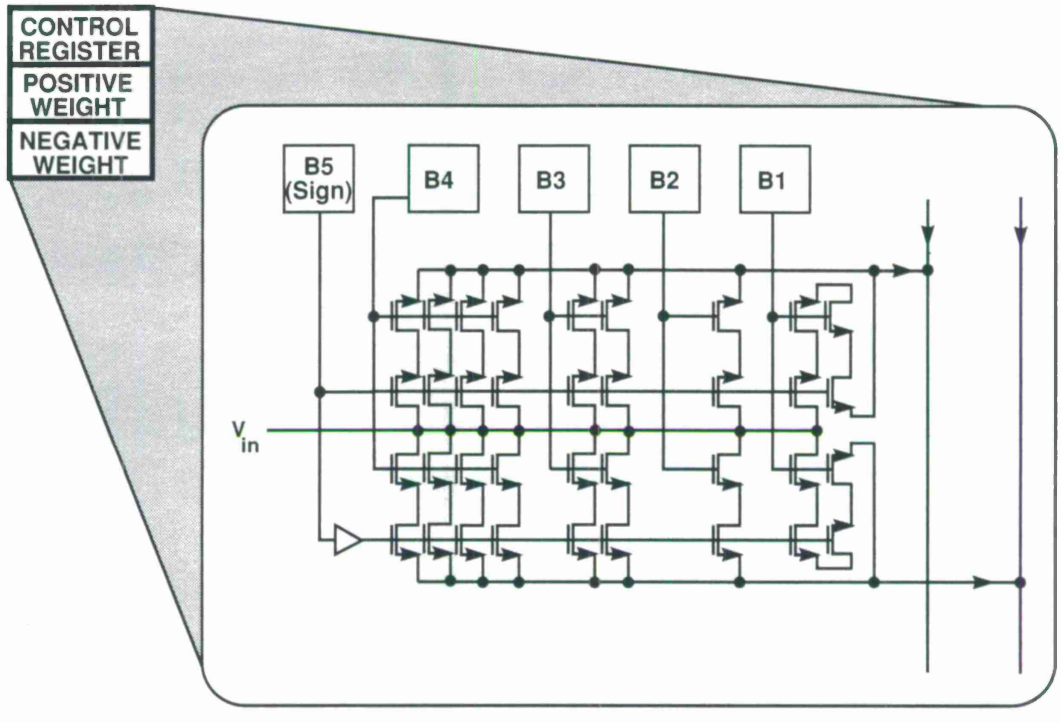


Figure 6-2. Weight circuitry for voltage-driven MDAC.

The circuit was implemented in a 3-μm p-well CMOS process. The silicon area required was 28 mm^2 and included 512 2-quadrant weights each of which had 5-bit resolution (including sign). With a unit weight, as the input voltage varied from 0 to 1 V, the current through the weight ranged from 0 to 25 μA. Thus, a fully on weight with a 1-V input voltage would source 400 μA. In the Hamming Net application, the MDAC array was evaluated in terms of its ability to perform the weighted sum computations that would be required in the back-propagation algorithm. In this situation, 22-bit digital input vectors which represent speech data were applied to a 1-layer network in which each node computed the distance between a particular template vector and the applied input.

This experiment demonstrated that the voltage-driven MDAC circuit was capable of performing the weighted sum computation, but it also illustrated some of the potential pitfalls of this design. One of the artifacts that was revealed by these experiments was that the currents driven by the input voltage signals caused clearly observable parasitic voltage drops due to the resistance of interconnect on the chip, thus distorting signals.

Another important issue was power consumption. In this experiment, a 1-V input generated 25 μA. In a large back-propagation network, where the average magnitude of all the weights tends to increase as learning proceeds, currents of this level could be a problem. The power demand on the nodes could change dramatically, dependent upon the particular weight pattern

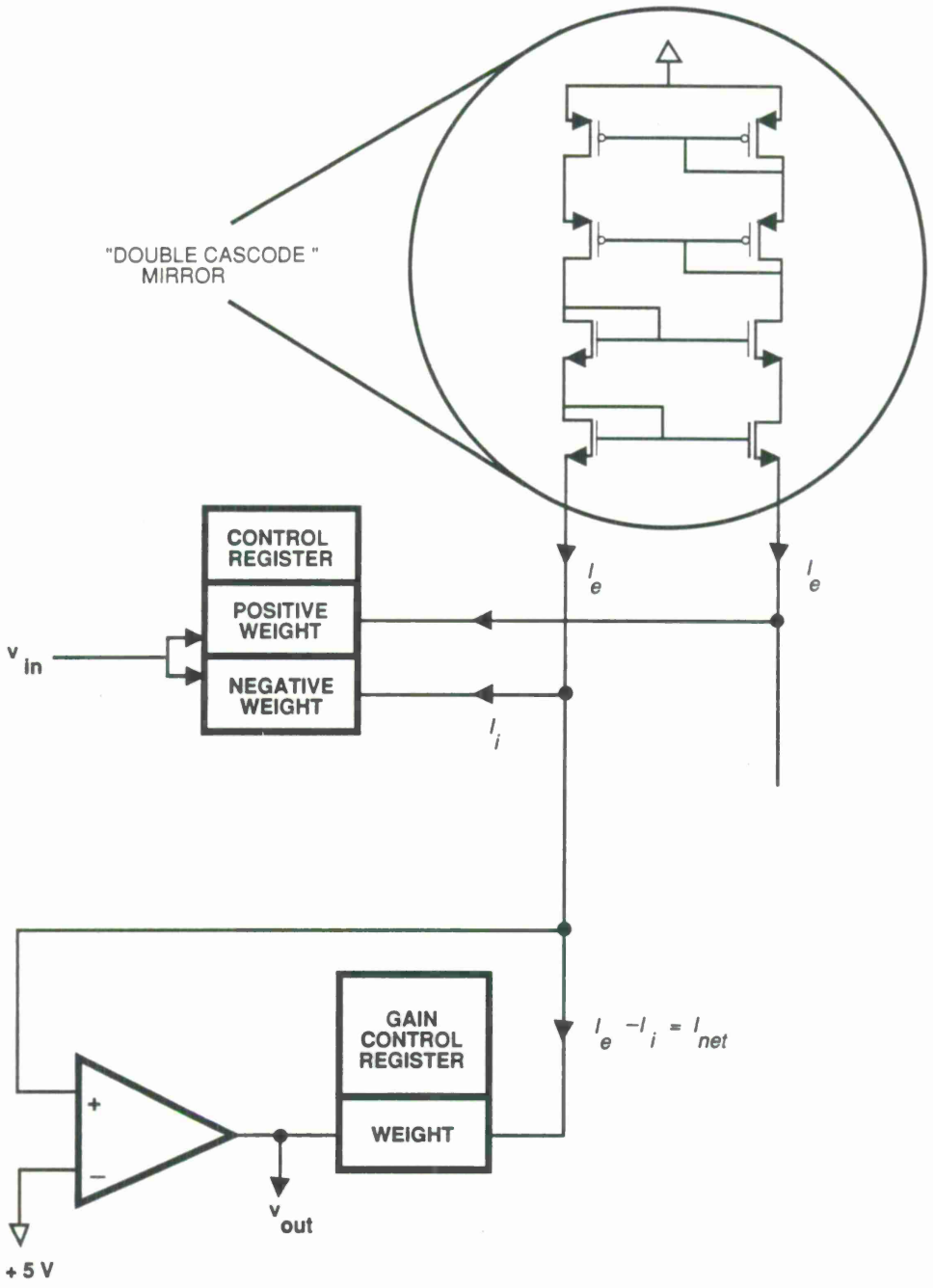


Figure 6-3. Block diagram of current-driven circuit.

that is stored in the network. A node that had to drive 50 half-on weights would be required to source up to 10 mA, a very large amount of current. Of course, this number could be reduced by decreasing the transconductance of a unit weight, but the fact that this current demand is greatly dependent upon the particular weight configuration cannot be avoided with this design.

These weight circuits require two operational amplifiers per node in order to perform the current summing as shown in Figure 6-1. The chip that was fabricated contained only the weight circuitry, and discrete Op-Amps were used to implement the current summing off chip. In the implementation of a back-propagation network, of course, the amplifiers would have to be on the chip and would need to be able to source several milliamps for moderate-sized application. Additionally, they would have to have low offset. Layout area would also be a consideration, and it is estimated that appropriate amplifiers would occupy as much as 0.2 mm^2 for each node.

2. Current Mirror MDAC

a. Circuit Overview

An alternative design approach, proposed by Professor Harry Lee of MIT to implement the processor, involves extensive use of current mirrors. In this scheme, the input signals are voltages but, unlike the previous design, these voltages drive only the gates of MOS devices which operate in saturation. Again, the effective widths of the MOS devices determine the weight of the connection, and these widths are controlled by a digital register which switches in combinations of parallel transistors. Thus, the current through a weight is linearly dependent upon the effective width of the device, and nonlinearly related to the gate voltage by the following equation:

$$I = \mu C_{ox} \frac{W}{2L} (V_{gs} - V_{th})\hat{}\,2 \quad . \tag{6-1}$$

The block diagram in Figure 6-3, which includes the circuitry for one node with one input weight (of course the fan-in could be expanded by simply adding more copies of the weight circuitry), shows that this design has some similarity to the previous one. As in the previous design, there are two currents developed, one that is the sum of the currents from the positive weights (I_e), and one that is the sum of the currents from the negative weights (I_i). These currents are subtracted to form the net current. Additionally, as in the previous design, each weight is actually composed of two identical circuits: one connected to the inhibition line, and one to the excitation line. Although both these circuits are driven by the same input, only one is activated, depending upon the sign of the weight.

Unlike the previous design, these weights sink instead of source current, and the current differencing is done at the inhibition node. All "negative" weights sink current out of this node, while the mirror serves to source a current into this node that is equal to the current being sunk by the "positive" weights in the excitation branch. The feedback circuitry at the bottom of Figure 6-3 will sink the difference current between that sourced by the mirror I_e and that sunk by the "negative" weights I_i.

The output of the circuit is taken from the output of the feedback amplifier which automatically adjusts the gate voltage to cause the weighting circuit to sink $I_e - I_i$ so, if it is used to drive a weight in the next layer, it will produce a current that is proportional to this difference current. Additionally, the effective gain of this circuit can be adjusted by adjusting the effective width of the feedback weight. Since this adjustment can be done with a digital register, this feature can be used to increase or decrease the range of the weights during operation.

Figure 6-4 shows a detailed schematic of the actual weight circuitry. In this diagram there is a 5-bit digital register at the top which encodes the value of the weight in a one's-complement representation. The bits in this register determine the effective width of the "transistor" that is being driven by the input voltage by selecting the number of transistors that are being driven in parallel. As more and more devices are added in parallel, the current that flows into a weight increases.

The configuration of the current summing circuitry requires that the weight be represented with the one's-complement format. If, for instance, the weights were represented in sign/magnitude format and all the weights were negative, then all the weights would be attempting to source current from the inhibition node, but there would be no current to bias the current mirror.

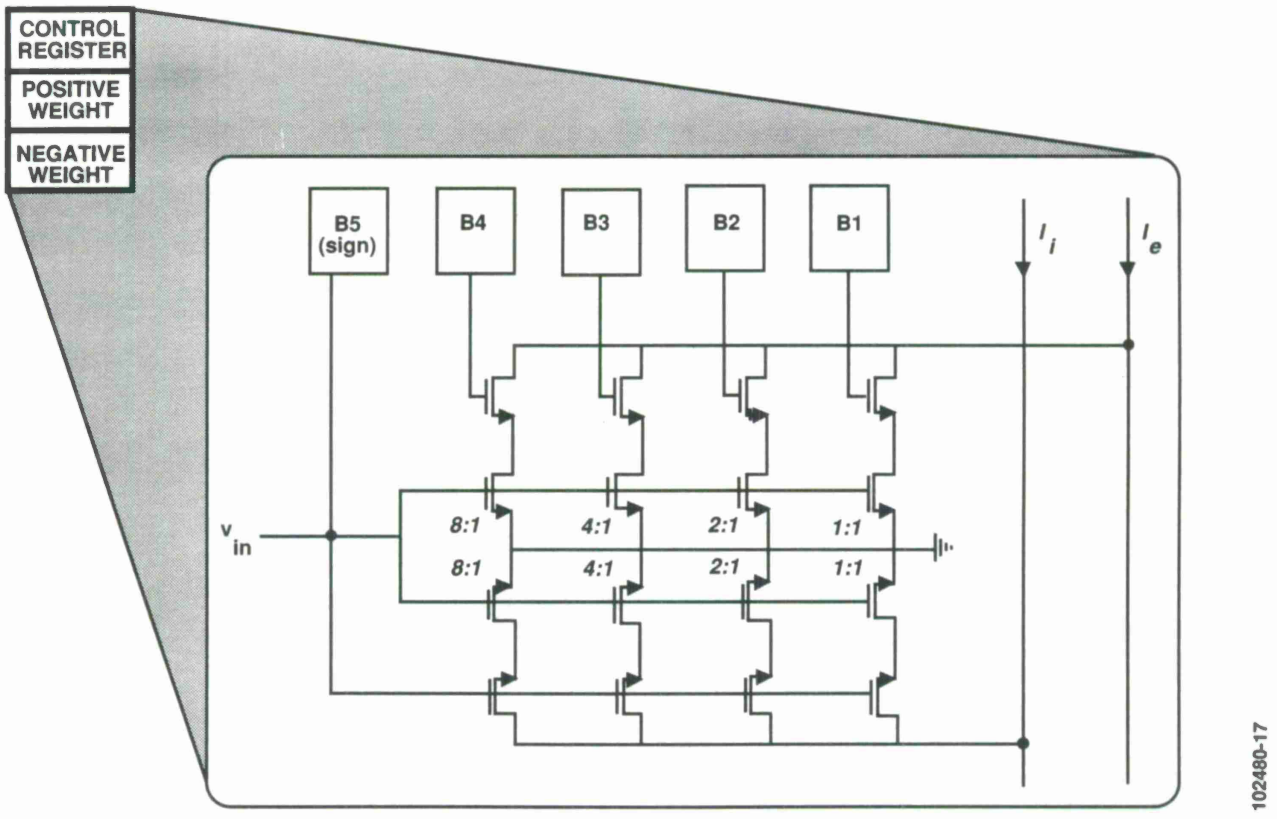


Figure 6-4. Weight circuitry for current-driven MDAC.

The mirror circuit performs two important functions in this design. As mentioned previously, it reflects the I_e into the inhibition line, but it also reflects the voltage from the inhibition line back onto the excitation line. The feedback amplifier monitors the voltage at the inhibition line and maintains a voltage at the input to the feedback weight such that the weight will sink the difference in current between what is sourced by the mirror I_e and what is sunk by the active negative weights I_i. Thus, the drain and source of the active transistors in all weight circuits are held at 5 and 0 V, respectively, by the feedback in the node circuitry, and the output voltage of the feedback amplifier is the voltage that, when applied to a weight circuit, will cause it to sink a current equal to $I_e - I_i$.

b. 4-Quadrant MDAC Design

One difficulty of both designs, as proposed above, is that they only work with positive input signals. While this 2-quadrant operation is not a problem for the forward net which only has to deal with signals that range from 0 to +1, the backward net which has the δ signals as inputs must be able to accept both positive and negative input signals and generate outputs that are positive or negative ($I_i > I_e$).

Figure 6-5 shows how the current mirror MDAC design can be modified to allow it to perform 4-quadrant operation. The design requires a second feedback weight in the excitation current summing node, and a fully differential amplifier with zero output offset voltage. The input and output of this circuit would now be composed of two analog signals that are equal in magnitude, but opposite in sign.

Where the 2-quadrant circuit discussed above required two weight circuits for each weight (one connected to the inhibition current summing node, and one to the excitation current summing node), the 4-quadrant circuit would require four of these same circuits and they would be connected as shown in Figure 6-5. At any one time, one of the input voltages will be negative and will thus cause no current to flow through the weight circuits that it is driving. Of the four weighting circuits, two would represent positive weights and two would be used to represent negative weights. In the block diagram, W1 is presenting a positive weight to the noninverted input signal because driving it increases I_e, while W3 presents a positive weight to the inverted input signal because driving it will increase the inhibition current (just the desired effect when a "negative" signal is to be multiplied by a positive weight). Likewise, W2 is a negative weight for the positive signal and W4 is the negative weight for the negative signal.

The feedback circuitry is the key to how this circuit can generate the appropriate output signals. Because the feedback amplifier is fully differential and has a 0-V output offset, only one of the feedback weights will ever be active at any one time. The other one will have a negative gate voltage, and thus be shut off. When the weights connected to the excitation summing node are conducting more current than those connected to the inhibition summing node, feedback weight FW1 will be active; but, when the weights connected to the inhibition node are conducting more current, feedback weight FW2 will be active.

This 4-quadrant design allows the construction of the back-propagation network necessary to propagate delta signals back from the output toward the input.

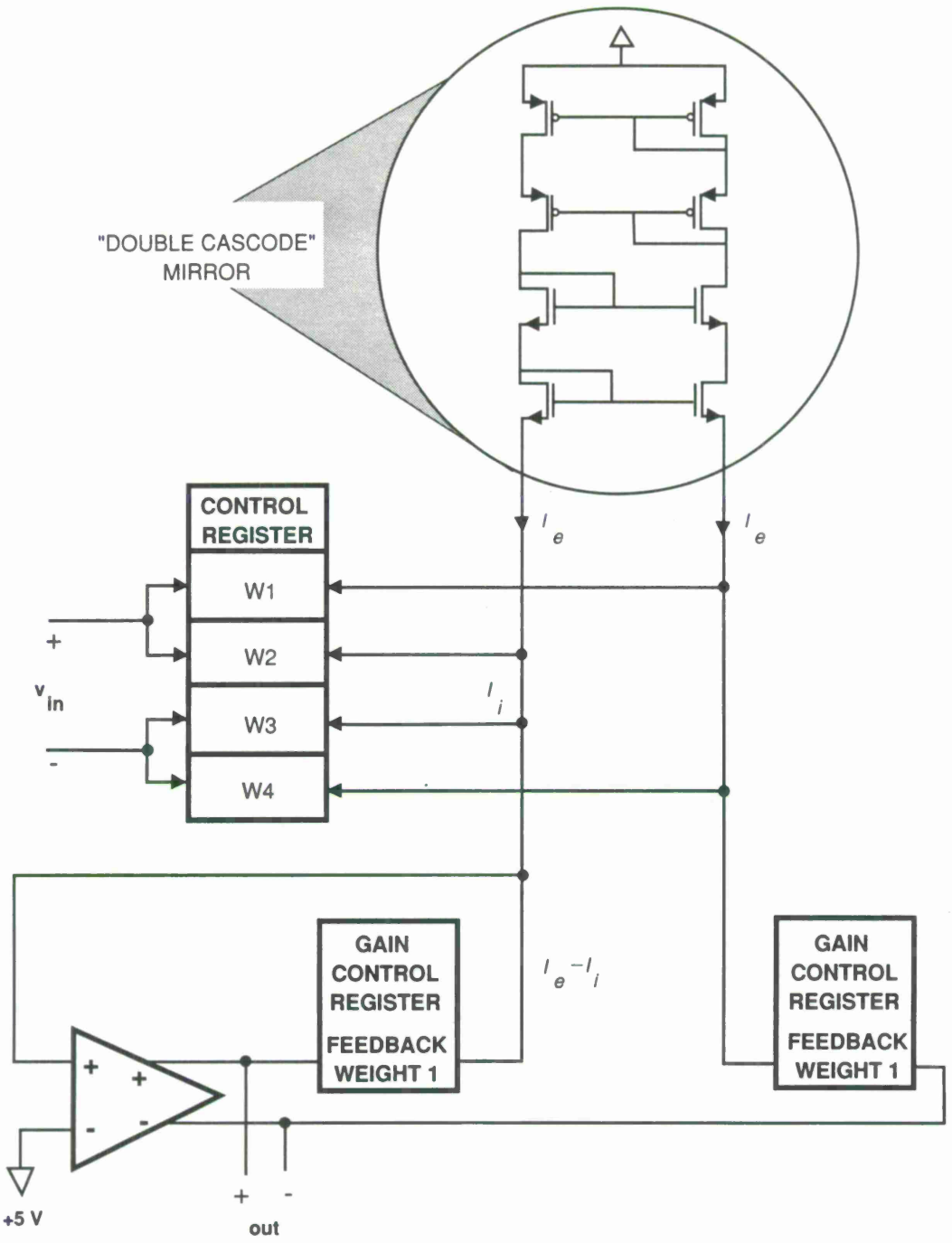


Figure 6-5. Block diagram of 4-quadrant MDAC.

c. Resolution of the Current Mirror Weight Circuitry

One significant problem with the weight circuit that was proposed in the previous section is that increasing the resolution by 1 bit effectively doubles the size of the circuit. In order to construct weights of higher resolution without this large area cost, the circuit in Figure 6-6 was proposed by myself and Professor John Wyatt of MIT. This circuit is a two-stage design that works on the principle of increasing resolution by dividing the smallest weight, instead of adding larger ones. In this circuit, transistor M1 is of "unit" width and its gate is driven by the input voltage to the weight. Only a fraction of the current that is conducted by this device, however, is drawn from the current in the current summing node; the rest of the current is drawn from a separate power supply. The second-stage circuitry determines what fraction of this current is taken from the summing node, and thus determines the effective value of the least-significant bits.

The advantage of this architecture is that doubling the number of bits only doubles the area, rather than increasing it exponentially as the more conventional design would. For this project, that means that a 10-bit weighting circuit will take about twice the area of a 5-bit design rather than 32X the area.

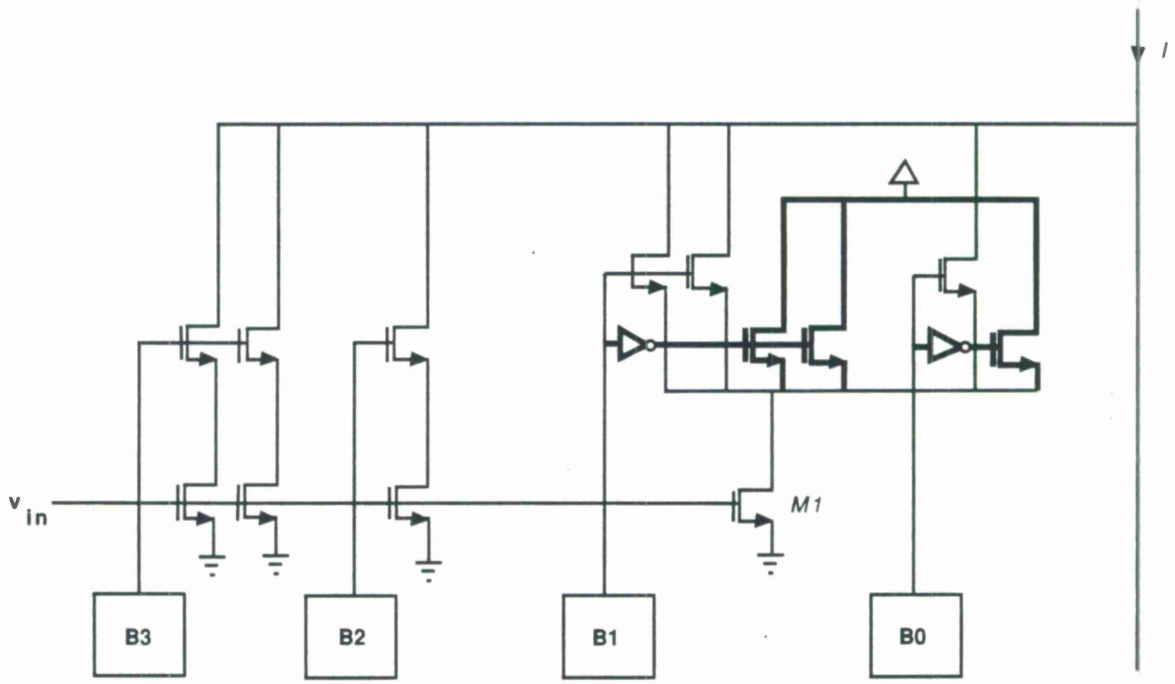


Figure 6-6. High-resolution current mirror weight circuitry.

3. Summary of MDAC Processors

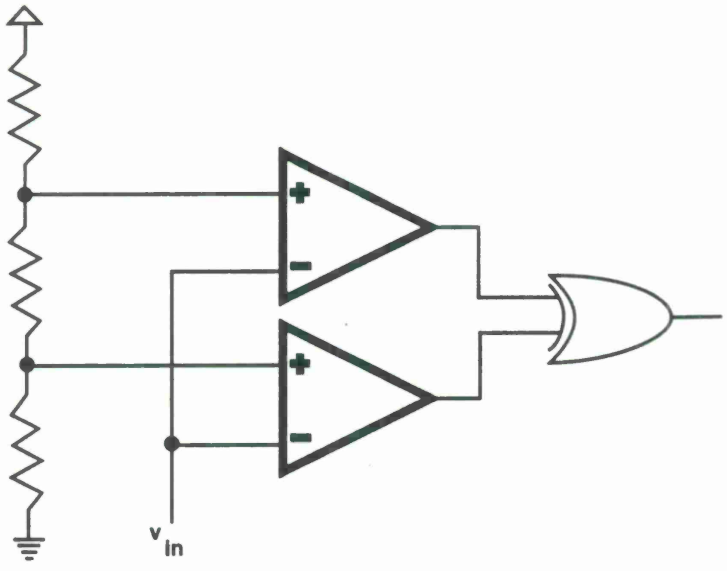
There have been two basic MDAC processor circuits presented here. The first one, referred to as "Voltage-Driven," uses the MOS devices in the linear region as resistors and sums currents with Op-Amps. The second circuit, referred to as "Current-Driven," uses MOS devices in the saturation region as current sources.

Although both designs have been investigated as a part of this research, it is the current-driven design that seems more attractive for the implementation of the back-propagation algorithm. First of all, the voltage-driven design requires the amplifiers to source a good deal of current, and the load on these amplifiers depends upon the particular weight configuration. Additionally, the signal can be distorted by resistive losses across the circuit.

Significantly, a method of implementing a 4-quadrant MDAC has been developed for the current-driven circuit and a technique has been discovered to greatly reduce the area required to implement a weight of a given resolution with this implementation. These features, in addition to the points mentioned above, clearly indicate that the current-driven design is superior for the back-propagation implementation.

B. DERIVATIVE APPROXIMATION

Taking advantage of the observation (Section 5-B-4) that the piecewise linear approximation of the derivative is acceptable for the learning process, a simple circuit like that shown in Figure 6-7 could be used to approximate the derivative function.



102480-20

Figure 6-7. Simple circuit to approximate derivative function.

The circuit works by simply detecting whether the signal voltage is within a certain range. If it is, then the digital output signal from this circuitry is true; but if it isn't, the output signal is false.

In the back-propagation algorithm, the derivative of the activation function is used to calculate the delta signal that is to be associated with a particular node. The delta signal for a node is found by taking the weighted sum of the delta signals from the nodes in the next layer, and multiplying that result by the derivative of the activation function of the node. This can be accomplished by using the output of the derivative approximation circuitry to modulate the gain of the processor that is being used to compute the weighted sum of the deltas.

C. WEIGHT MODIFICATION CIRCUITRY

The processor circuits discussed in the previous section will compute the activation and error signals. Additional circuitry, however, is required to calculate the weight changes from these signals. According to the back-propagation algorithm, adjusting the weight W_{ji}^{ℓ} requires multiplying $O_i^{\ell-1}$ by δ_j^{ℓ}. As Figure 4-2 showed, both these signals are now available at every weight in the network in the form of currents. Because the product of these two **analog** signals will be used to modify a **digitally** encoded weight, digitization must be performed either before or after the multiplication operation has taken place.

Another design consideration is whether the multiplication operation should be done with local circuitry at each weight, with global circuitry that is used to update all the weights sequentially, or with some circuitry that is shared on a somewhat smaller scale. All the methods that have been considered for implementing the update operation would require a good deal of circuitry. Replicating this circuitry at each node thus seems unreasonable, and it seems clear that this circuitry should be some type of global resource. This resource could be shared in a relatively simple way. The weights can be sequentially accessed and, when a weight is accessed, it drives current signals onto a global bus that corresponds to the δ and O signals for that weight. The global circuitry could then observe those signals, calculate the weight change to be applied to that weight, and modify the weight appropriately.

One possibility is to design a 4-quadrant analog multiplier that would multiply the two analog signals, and then a circuit which would quantize that result to a digital code that could be used to increment or decrement the register that controls the value of the weight. This implementation would have the severe drawback, however, that it would be difficult to modify the algorithm to adjust the learning rate, or to include such concepts as momentum that were discussed in Section 3.

In order to allow for the greatest algorithmic flexibility, the signals could be converted immediately to digital codes and then the required weight change could be calculated by a digital circuit (either externally, or on the same chip). The algorithm could then be adjusted by simply modifying the program that the digital computer is running, yet the analog network would still be performing computations required to calculate all the activation and δ signals.

7. ALTERNATIVE DESIGN APPROACHES

The goal of this research is to design circuitry that will implement the back-propagation algorithm. An original design has been presented in previous sections and, although the proposed design has the advantage of well-controlled performance and the use of conventional technology, it has the drawback that it is very area intensive and it would be difficult to imagine using this method to construct extremely large networks. Here, some alternative design approaches will be presented.

The alternative design approaches can be separated into two classes: those that use analog hardware, and those that use digital hardware. The alternative analog designs discussed here store the weight as an analog value instead of digitally, and thus achieve great reduction in terms of weight area. Several approaches to storing these weight values are described and compared in this section.

A purely digital design of a systolic array processor is presented in Section B below. Most of the alternative approaches presented here are based on ideas that have been presented in the literature for implementing other neural networks, and their applicability to back-propagation networks is discussed here. The one exception is the custom digital array processor architecture which is presented for the first time in this report.

A. ANALOG DESIGNS

Analog Weights in Weighted Summing Circuits:— The basic principle that was used in the proposed designs of Section 6 involved constructing variable weight connections by adjusting the number of parallel "unit" weights. An alternative circuit would use only one MOS device, and would change the weight by changing the transconductance of this device by modifying its bias point [Tsividis, 87]. The basic circuit is shown in Figure 7-1 where, if the voltage stored on the capacitor is sufficiently greater than the V_{ds} of the MOS device, the transistor will have a linear

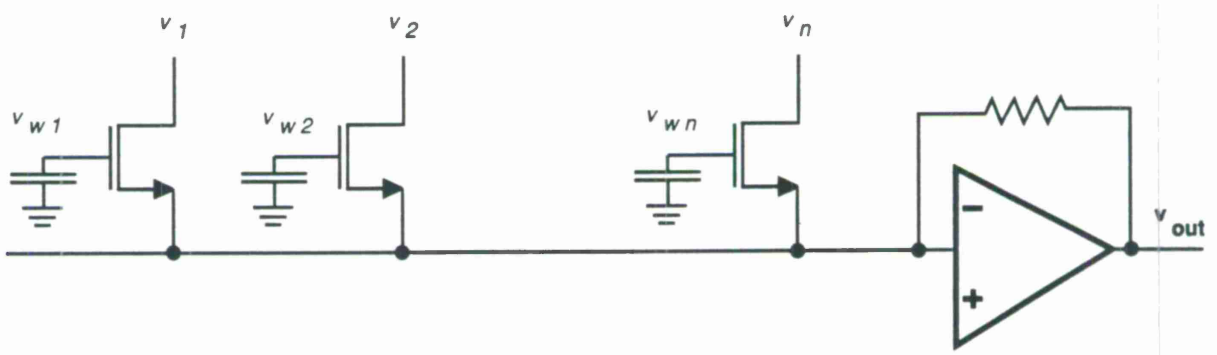


Figure 7-1. Basic fully analog circuit.

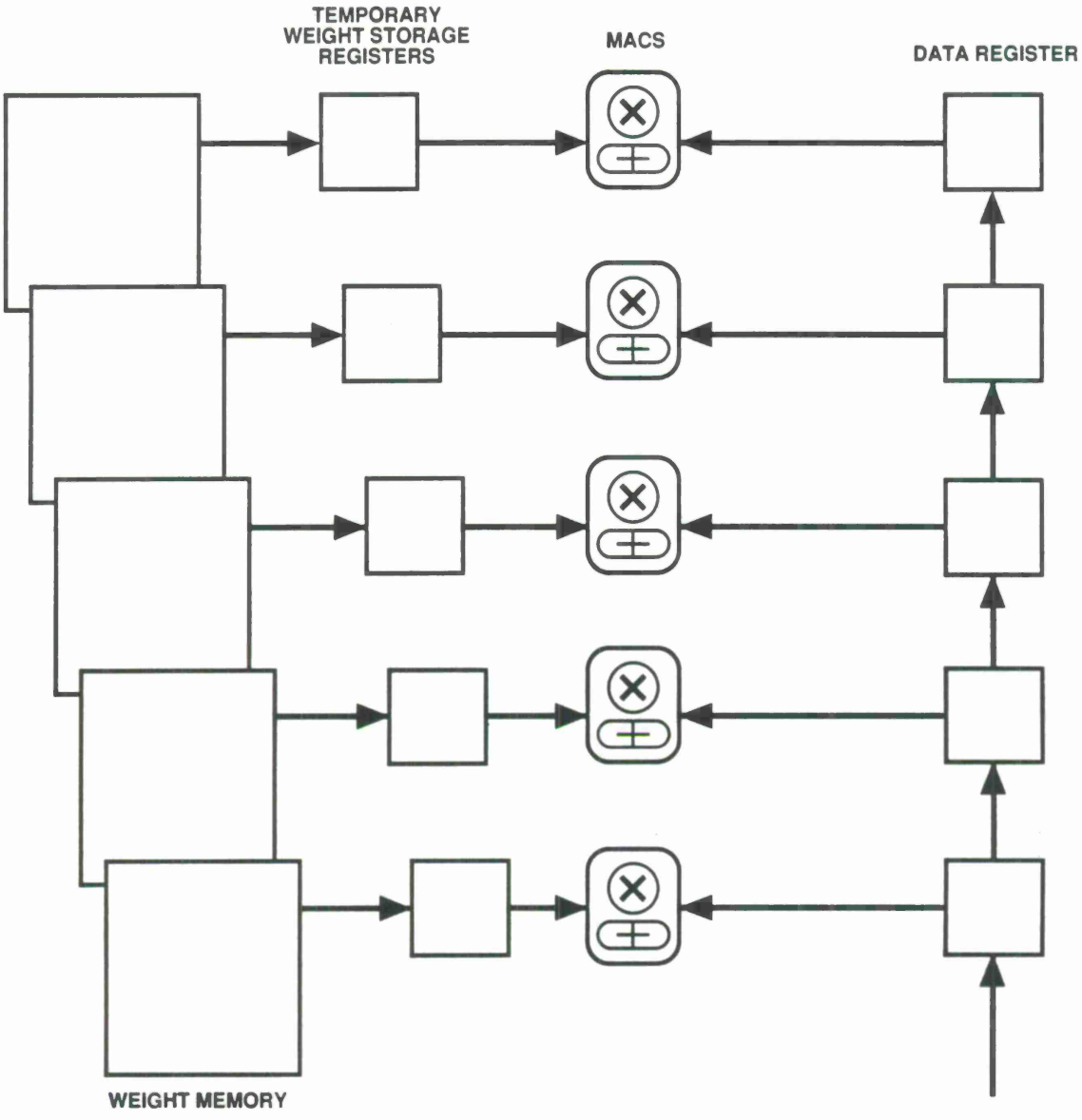


Figure 7-2. Basic architecture of fully digital array processor.

characteristic. In this circuit, the MOS device operates in the linear region just as in the circuit described in Section 6-A-1; however, the bias of the device is now changed in order to vary the transconductance.

The problem with this implementation, as with any implementation which stores analog values in a standard CMOS circuit, is that the charge will leak from the capacitor, and the analog value will drift unless it is continually refreshed. There have been at least two interesting methods presented in the literature for refreshing these voltages.

One method [Brown, 87] suggests the use of a quantized refresh scheme. In this method, the range of allowable gate voltages is broken into a number of steps (256 steps would be equivalent to 8-bit quantization), and a stored voltage is refreshed by determining which sub-range the voltage is in. The refresh circuitry then restores the signal to the upper limit of that particular sub-range. If refresh is performed frequently enough to assure that no value can drift by more than the span of one sub-range, then all voltages will be maintained at a quantized version of their original value.

Another method that has been proposed [Kub, 88] involves maintaining a digital representation of all the weights in the entire network in conventional memory. During a refresh cycle, the digital code for the desired weight value is applied to a D/A converter. The output of the D/A converter is used to restore the capacitor voltage to the desired level. The coarseness of the quantization in this algorithm is limited only by the number of bits of precision available in the D/A converter. Of course, the D/A converter is a shared resource, so the precision used does not affect the size of each weight. Another advantage of this strategy is that a host computer can always have a record of the current state of every weight in the network for the sake of performing weight updates, or just keeping track of the value of a weight during learning.

These proposals are for circuitry that could be constructed in a standard CMOS process. There have been some suggestions, however, for other analog circuit implementations that use slightly more exotic technologies. For example, the use of MNOS technology has been proposed to store the network weights [Sage, 86]. In this scheme, the weights are represented as analog voltages but they are stored in traps in a gate dielectric so that there is essentially no leakage, and thus no need for refreshing.

B. DIGITAL DESIGN

Digital-Array-Processor Architecture:— Another alternative design approach would be to design a fully digital machine that achieved speed and efficiency through parallelism but maintained the degree of precision and control that is available in a modern digital computer. An original architecture is proposed here to illustrate one possible way to implement such a machine. This design will be examined in Section 8 when the performance of the various designs is considered.

Figure 7-2 shows the structure of the proposed architecture. The components are weight memory, a series of multiply-accumulate (MAC) processors, and a shift register. This structure

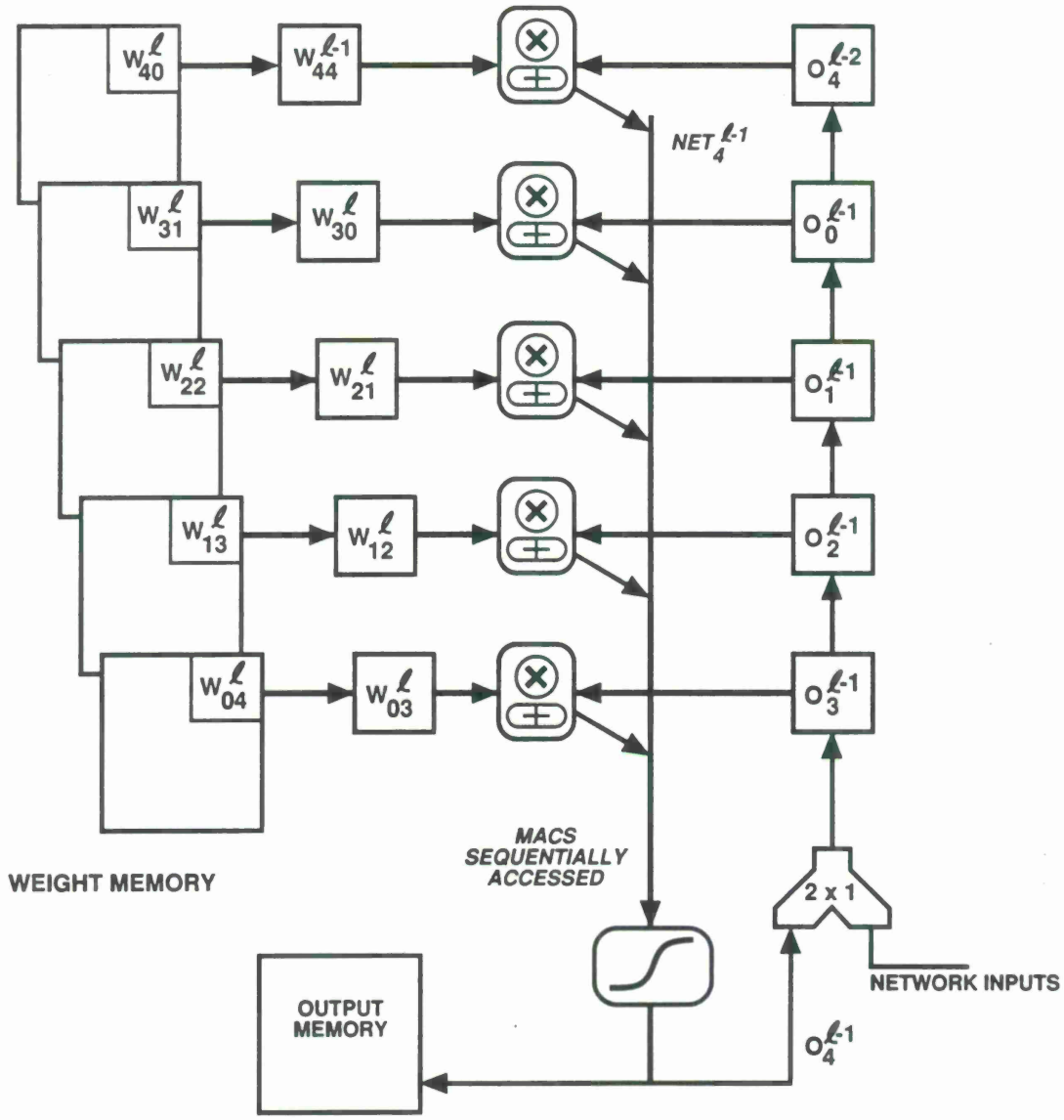


Figure 7-3. Configuration for forward-propagation of activation signals.

could be used to calculate both the weighted sum of products of the activation signals for the forward-propagation and the error signals in the back-propagation. In general, multiple layers timeshare the hardware as calculations are done for one layer at a time, and the results of the calculations from one layer are used (as soon as they are available) as the inputs to the next layer calculations. Although the system is most efficient when the number of MACs is equal to the number of nodes in the widest layer, the design is modular, so that two small circuits could be concatenated to form a larger circuit. The architecture is most efficient when there are the same number of nodes in each layer.

The basic structure will be configured slightly differently when used for the forward and back calculations. The main reason for this is to insure that the memory that is used to store the weights can be used most efficiently, as will be shown in the following description.

When the network is being used to perform the forward calculations, it would be configured as in Figure 7-3. During each clock cycle, a new weight is read into each of the weight holding registers, and a new value is shifted into the first stage of the shift register. In this configuration, each MAC will calculate the weighted sum of products for a particular node (NET_j). When a MAC has completed that calculation the result will be read, and that MAC will immediately begin calculating the weighted sum for the corresponding node in the next layer. Once the "pipe" is full, one of the MACs will produce a complete result after each clock cycle (if there are the same number of nodes in each layer). The result from a MAC will be passed through a processor that calculates the activation function, and the output from that will be both stored in a memory for later use in updating the weight values and fed back into the shift register as an input for the next layer. Although the 2×1 multiplexer will initially let the network inputs fill up the pipe, it will switch as soon as the processors start having complete results so as to use these outputs as the inputs to the next stage.

When used to back-propagate the error signal, the basic architecture would be configured as in Figure 7-4. The biggest change is that now the network "inputs" (the error signals) are stationary relative to the weight memories, and the partial results from the MACs are shifted. In this configuration, during any one period each MAC will calculate one more term that it will add to the partial sum in its accumulator, and will then pass its partial sum on to the next MAC (the accumulator of the first MAC will be loaded with 0). In this case, the results would be shifted out one at a time and the inputs would be loaded by sequential access of the storage registers. The reason for this change in role (between what is shifting and what is stationary) is to allow the weights to come from the same pages of memory and allow them to be accessed in a similar order, independent of whether the forward- or back-propagation is taking place.

Of course, the goal of back-propagating the error signals is to allow for the calculations of the weight modifications. The weight updating could be performed concurrently with the δ calculations with the additional circuitry included in Figure 7-5. In this figure, the outputs that were calculated during the forward pass through the network are sequentially shifted through another shift register. During any given cycle, the output and the δ that are at any given stage are the correct signals to use to update the weight which is at that same stage during that cycle. After

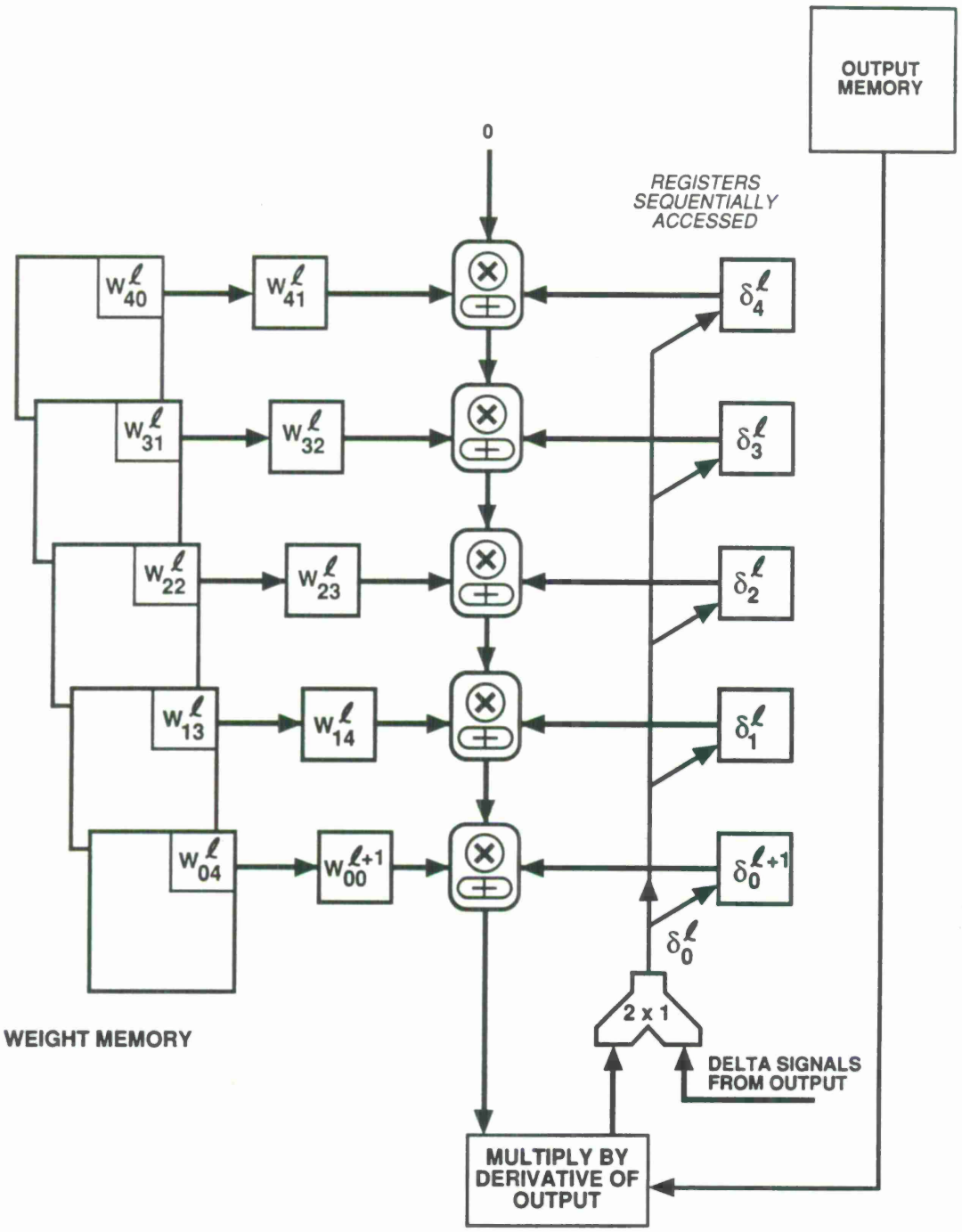


Figure 7-4. Configuration for back-propagation of error signals.

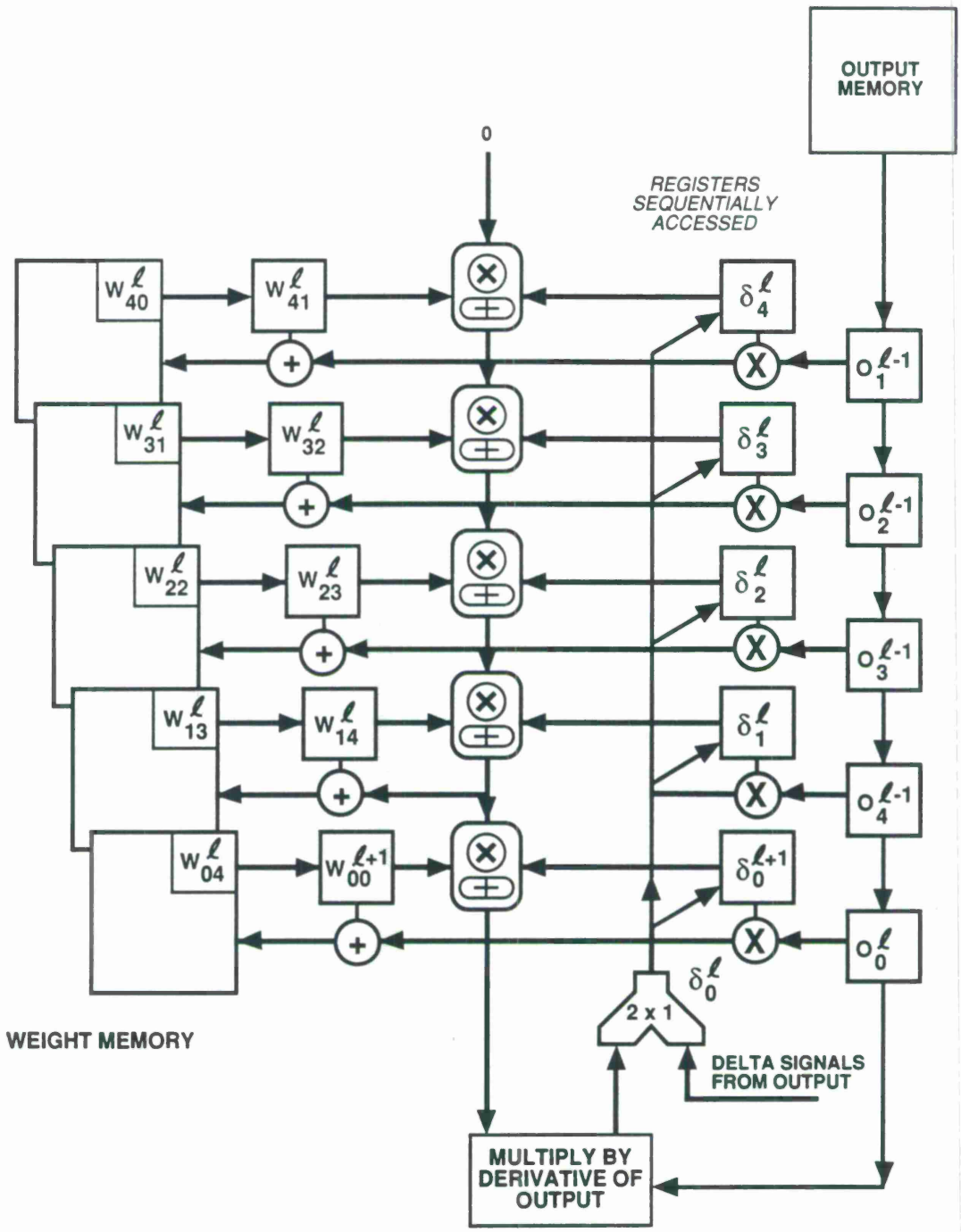


Figure 7-5. Configuration for concurrent weight modification.

the cycle, the calculated weight adjustment is added to the weight and the modified weight value is written back to the weight memory. Additionally, the outputs emerge at the bottom of the shift register at the correct time to be used for the calculation of the derivative of the appropriate activation function.

The multiplier and adder that are used for weight updating in Figure 7-5 represent only the simplest such architecture. Additional memory might be added to the system to facilitate the accumulation of weight-change information for use in variations of the back-propagation algorithm that require the accumulation of information regarding past weight changes. Alternatively, instead of having separate multipliers and adders in the circuit for the purpose of weight-change calculation, the appropriate sections of the MACs could be multiplexed between being used to calculate the activation/error signals, and the calculation of the weight adjustments. This design would save the cost of the additional multipliers and adders at the cost of complexity in the MACs and the controlling circuitry.

C. ALTERNATIVE ALGORITHM

One common feature of all the alternative designs presented so far is that they attempt to implement the back-propagation algorithm. A final alternative, which does not perform the calculations of the back-propagation algorithm but could perform a similar gradient descent, is presented here.

As discussed in Section 3, back-propagation implements a gradient descent of an error function in order to learn a weight pattern. In this context, each weight is changed by an amount proportional to the negative of the gradient of the error function with respect to that weight. Back-propagation simply specifies a method of computing these gradients. The alternative, which was proposed by Professor John Wyatt of MIT, involves measuring (instead of calculating) this gradient.

The system would require a forward network similar to that used in back-propagation, and the learning procedure would still involve stepping through the training vectors and defining an error function based upon the actual outputs of the network. The alternative algorithm would then involve randomly adjusting each weight, one at a time, and determining whether the adjustment increased or decreased the error function. If the weight change increased the error, then a weight change in the opposite direction would be performed. In this way, each weight would be adjusted in a direction that would tend to decrease the error function. The magnitude of the weight change that should be made could be determined by a number of factors including how dramatically the error is affected by a small change in the particular weight, or how much the weight can be adjusted before further adjustments no longer decrease the error function.

This algorithm might be appropriate for analog designs. The advantage would be that the measurement of $\partial E_p / \partial W_{ji}$ would, in principle, correctly account for all sources of imprecision in the network since the actual response of the physical network is being measured. The disadvantage is that, at least in the most straightforward implementations, the measurements required for

each individual weight change would be made sequentially and thus require considerable amounts of time. A more parallel alternative would be to sequentially perturb each neuron's NET input to directly measure each neuron's "δ," and then in parallel perform the 2-quadrant multiplications necessary to update the individual weights.

8. PERFORMANCE COMPARISON

The designs that were presented in Section 7 can be classified as fully analog, fully digital, and a combination of the two with digital weights and analog signals. This section will attempt to compare and contrast these designs in terms of their relative performance and costs. This comparison will be complicated by the fact that, although detailed descriptions of the analog/digital and fully digital design proposals have been presented, a fully analog approach has not been developed in this report (or in any other work that the author is aware of). Although this will limit the level of detail of the performance comparison, the basic advantages and disadvantages of each approach should become apparent.

In order to form a basis of comparison, the performance and cost of each design will be measured in terms of several basic parameters. Cost will include "area of silicon required" which will be computed from the area required for one node and the area required for one weight. In this context, a "node" will include the fixed cost of the circuitry required to sum the activation signals in the forward direction and the error signals during the back-propagation phase. Similarly, a "weight" will include the marginal cost of the circuitry for one weight that will operate in both the forward and backward directions. The alternative design approaches will also be compared in terms of the speed with which they perform certain calculations. In addition to these objective metrics, some comparison also will be done in terms of more subjective, yet equally significant, points such as adaptability, expandability, and versatility.

A. PURELY ANALOG APPROACH

Although there were several different fully analog approaches discussed in Section 7, this performance comparison will only be concerned with those that can be implemented in standard CMOS circuitry. These designs generally have in common the fact that the size of the weight is stored as an analog voltage on a capacitor at the gate of the weighting transistor (Kub, 88) (Tsividis, 87). The issue that determines the area required to implement a weight in these designs is that the capacitor must be sufficiently large so that it will maintain a voltage to the desired precision over the time interval between refreshes. The following assumptions will be made:

Leakage Current	5 pA
Range of Gate Voltages	2 V
Required Precision	8 bits
Capacitance/μm^2	0.3 fF/μm^2
Refresh Period	1 ms

From these assumptions, the following calculations can be made:

(1) The capacitance required to maintain the gate voltage to 8-bit precision:

$$C_{min} = \frac{5 * 10^{-12}\ A * 1 * 10^{-3}\ s}{(2/128^{\dagger})v} = 320\ fF \quad . \tag{8-1}$$

(2) The silicon area required to implement C_{min} in standard CMOS:

$$AREA_{min} = \frac{320\ fF}{0.3 * 10^{-15}\ F/\mu m^2} = 1067\ \mu m^2 = 0.001\ mm^2 \quad . \tag{8-2}$$

Thus, each weight will require a 320-fF capacitor which would require 1067 μm^2 to maintain the required precision for the entire refresh period. Although a weight circuit like those proposed for the fully analog network would only be able to implement a 2-quadrant multiplier, I will assume for the purpose of this analysis that an equivalent 4-quadrant circuit for the back-propagation of the error signal could be developed. Because the size of the weight is determined by the size of the capacitor used to store the weight, I will assume that a 4-quadrant circuit will not add significantly to the area of the weight. Additionally, because the size of the weight is to be the same in both the forward and reverse directions, I will assume that only one capacitor of this size will be needed at each weight and that, when compared with the size of this capacitor, the other circuitry at the node will be of insignificant area.

It is important to note that these calculations are based on estimates of such factors as leakage current and refresh cycle time. More elaborate schemes such as cooling the circuitry with liquid nitrogen or modifying the process in order to minimize leakage current might dramatically reduce the area required for each weight.

The nodes in this circuit will consist of several amplifiers, and the area estimate will be based upon the circuit presented in (Raffel, 87) and which requires two amplifiers for the node in the forward network, and two amplifiers in the reverse direction. The area for this circuitry would be approximately 0.3 mm^2.

Determining the speed of this architecture is a bit more complicated than predicting its size. The speed will be determined by the amount of time required to charge and discharge the various parasitic capacitances in the circuit. In the most general version of this design (Figure 7-1), these time constants will be determined mainly by parasitic capacitances in both the interconnect and the active circuitry, the conductances of the weights, and the characteristics of the summing amplifiers. Although all these factors will be very dependent upon the specifics of the particular analog implementation used, some general interrelationships should be implementation independent.

First of all, the speed will depend upon the architecture. The parasitic capacitance will be affected by the "fan-in" and "fan-out" of each node because each weight circuit will add

† 8 bits includes sign bit, so 128 steps to represent magnitude.

additional parasitic capacitance. Additionally, geometric considerations will require that each additional weight is further away, physically, from the amplifiers, and thus the interconnection parasitics will scale almost linearly with the size of the network.

The speed of this design will be further determined by the operating characteristics of the amplifiers that are used, and the specific conductance ranges that the weights can assume.

B. MIXED ANALOG AND DIGITAL APPROACH

The design proposed in Section 6 involved storing the weights as digital codes, and representing them with MDACs. The performance estimate will be based on the current mirror approach assuming that all weights were implemented as 8-bit values according to the circuit design that was proposed in Section 6-A-2-c. According to this design, the 2-quadrant weights used for forward-propagation require approximately 57,000 μm^2. With the addition of 4-quadrant weights for the back net, the total weight area would be 0.4 mm^2.

In this design, the overhead required for each node would be two amplifiers with feedback weights as shown in Figures 6-3 and 6-5. This circuitry would require approximately 0.15 mm^2.

As with the fully analog design, predicting the speed of this design is a difficult task. Again, the speed will be dependent upon the architecture used for a particular application as well as by the specific characteristics of the feedback amplifier. In this design, the complexity of the feedback path in the current mirror circuit adds more complication because preliminary observations indicate that the bandwidth of the feedback amplifier may need to be limited in this design in order to insure the stability of the feedback arrangement.

C. DIGITAL ARRAY PROCESSOR (PURELY DIGITAL) APPROACH

Implementation of the basic digital architecture proposed in Section 7 requires an array of multiply-accumulate processors, and some storage registers. For the purposes of the performance analysis, it will be assumed that all numbers are 16-bit fixed point, and that the results of all multiplications of two 16-bit values are stored as 32-bit results. The reason for using a higher precision than the 8 bits for comparison to the analog/digital design with 8-bit weights is that in the analog/digital approach the weights are quantized but the signals are not. Simulations should be done in the future to determine exactly what precision is required, but, for the purposes of this performance evaluation, the conservative value of 16 bits will be used.

There are basically two ways to implement the MACs: with serial arithmetic or with parallel arithmetic. With serial arithmetic, each cell would be approximately 3 mm^2 and, if they were to run at 20 MHz, they would perform one 16 $\times$ 16 bit multiply in 80 μs. Although a weighted sum of products involves 50 such operations, it would only take 100 μs for each sum due to the parallel nature of the architecture. A forward and backward cycle through the network would consist of about 250 such calculations, and would thus require about 25 ms.

If the MACs were to be implemented in parallel hardware, then they would be larger (about 7 mm^2) but faster — capable of performing each weighted sum of products in approximately 30 μs. A forward and backward cycle through the network would thus involve approximately 250 of these operations and would thus take about 7.5 ms.

Like the analog and analog/digital implementations, the required computation time in the fully digital architecture is affected by the "fan-in" and "fan-out" of the nodes. Unlike those designs, however, the speed of the digital design would scale linearly with the size of the network, while the speed of the other designs would tend to be more severely affected by network size as discussed above.

Unlike the analog and analog/digital implementations, in the fully digital implementation the weights require very little area because they can be stored as digital numbers in high-density RAM. If the weights were to be stored as 8-bit numbers, then the area required would be approximately 300 μm^2 for each weight.

D. SUMMARY

1. Area Efficiency

The different design approaches can be characterized by the amount of area required. In a network that has n nodes in each layer, there are n^2 weights in each layer, so it should be clear that the area consumed by the weight circuitry is more critical than the area that is consumed by the node circuitry. Table 8-1 compares the size of weights and nodes that would result from the various design approaches. As the table shows, the weight circuitry associated with the combined analog/digital approach is considerably larger than the area associated with the weight circuitry in either of the other approaches.

TABLE 8-1

Summary of Cell Area Requirements

Design Approach	Size	
	Node (mm^2)	Weight (mm^2)
Mixed Analog and Digital	0.15	0.4
Purely Analog	0.3	$1 * 10^{-3}$
Digital Array Processor (serial arithmetic)	3.0	$3 * 10^{-4}$
Digital Array Processor (parallel arithmetic)	7.0	$3 * 10^{-4}$

Table 8-2 compares the different implementation strategies in terms of the size of the network that could be implemented on a 1 × 1 cm die assuming that there were the same number of nodes in each layer.

TABLE 8-2

Maximum Network Size in Nodes/Layer (1 × 1 cm Die)

Design Approach	Number of Layers			
	1	2	3	4
Analog/Digital	14	10	8	7
Analog	175	100	70	50
Digital*	30	30	30	30

*Serial arithmetic

2. Ability to Initialize

A significant drawback of the fully analog approach is the difficulty in initializing the weights in a network to some predetermined values, because the voltages stored on the capacitors are not related in any simple way to the value of the absolute value of the weight that they represent. Although the weights could be initialized to a specific voltage, the mapping from the stored voltage to the weight size is not simple. In both the fully digital and half-digital/half-analog approaches, the weights are represented as digital numbers and these numbers map directly to the weight size. Although this is not catastrophic when the weights are to be initialized randomly and learned by the network, this would prevent the network from being used for only the processing phase of the algorithm (not the learning phase) and would also make it difficult to evaluate the effects of different nonrandom initialization patterns.

3. Versatility

The fully digital approach has many advantages in this regard. In the digital architecture, speed can be traded off against size. Although the architecture works most efficiently when there are as many MACs as there are nodes in the largest layer, this architecture could easily be modified so that any size architecture could implement even the largest of networks. The trade-off

would be that the network would no longer be able to operate at maximum efficiency and would perform more and more slowly on larger and larger networks. Nonetheless, unlike the analog architectures in which the size of the hardware limits the size of networks that can be implemented, there is no such limitation with the fully digital approach. Additionally, chips containing small digital networks could be cascaded to form larger networks; this would be much more difficult, if not impossible, to accomplish with the analog/digital or fully analog approaches.

E. CONCLUSIONS

Three designs have been explored in this report and, although they all have their advantages, the digital design seems to be the most promising. The digital design has the advantages of being more versatile and provides ways to cascade smaller nets to construct larger ones. Although the fully analog approach is more area efficient for very small networks, the digital approach would be very comparable if not smaller for networks that have a large number of weights. While the analog/digital approach makes it easier to store the weight values than the fully analog approach, the weight size is simply too large to make it viable for large networks.

9. CONCLUSIONS AND FUTURE WORK

This research has concentrated on defining the significant issues involved in implementing the back-propagation algorithm in hardware, and designing the required circuitry. This work has led to the determination of the sensitivity of the algorithm to various implementation artifacts, and to some insight into the advantages of the different possible design approaches.

The simulations revealed that weight quantization affects learning and operation differently. Furthermore, it was determined that in both learning and representation, range was as important as the word length used to represent the weights. Specifically, it was determined that in a system that used the logistic activation function which is described in Figure 3-2, the resolution required for representing a problem is a minimum step size of 0.5 which translates into 6 bits for the problems that were examined. For learning, a minimum step size of about 0.1 was required. In the experiments, this translated into 8-bit word lengths.

To meet the goal of a design that performed analog computations, a design which used analog signals and digital weights was proposed. This design was then contrasted with a fully analog approach where it was shown that the fully analog approach was much more area-efficient but was not capable of being accurately initialized. Although the fully analog approach has the additional complication of storing analog voltages, several schemes for accomplishing this were reviewed.

A digital architecture to perform the back-propagation algorithm was also proposed, and this was contrasted with the other two design approaches. It was found that this approach offered many advantages including reliability, expandability, adaptability, and the ability to use small architectures to emulate larger networks by simply trading off speed for size. Furthermore, while the digital design would require more area for smaller networks, on networks with a large number of connections the area required by the digital design would quickly become comparable to that required by the fully analog design.

The most promising direction for future effort is in the detailed design of a fully digital array processor architecture. Although questions remain regarding the number of bits required in this implementation as well as the details of a design that could be readily cascaded and expanded, this design approach seems to offer the most potential.

APPENDIX A
SIMULATION DETAILS

1. THE PROBLEMS

The training data for the simulated problems was found by defining the region boundaries and then selecting data points within each region from a uniform random distribution over that region. Each problem had an equal number of points chosen from each class region, and the points were presented during training interspersed by region (one point from region A, one point from region B, etc.). There were a total of 500 training vectors for the "circle" and "corner" problems (250 from class A and 250 from class B), and there were 1000 training vectors for the 10-dimensional problem (100 from each class).

The class region boundaries for the 10-dimensional problems were hyperspheres. All the centers of the hyperspheres were equidistant (0.57 input-space-units from each other) and the spheres have a radius of 0.275 input-space-units. Thus, all spheres are 0.02 input-space-units from each other at the nearest point. Figure 5-2 shows graphically what a 2-dimensional equivalent of this 10-dimensional problem would be.

2. THE SIMULATIONS

All weights were adjusted after application of each vector according to the modified learning algorithm presented in Section 3. The proportionality constant η which determined what fraction of the calculated weight change was actually applied to the weight, and the "momentum" α were generally set to 0.3, but their values were varied between 0.2 and 1.0 in simulations where acceptable performance was not achieved with the values of 0.3. The effect of changing each of these two constants was very different.

When η was too small, no weight adjustments would take place because all calculated changes were smaller than the resolution of the weights. When η was too large, the network weights would change after each vector was applied and the error would never get very small. When α was too small, the weights would also change considerably without minimizing the error; but, when α was too large, the network would cyclically reduce and increase the error function. First, the network would begin reducing the error function as the network moved toward a solution. When the error was getting small, however, the weights would continue to change and the error function would begin to increase again.

Because the sigmoid function that was used (see Figure 3-2) never reaches 0 or 1, in the simulations, when the output of the network was within 0.1 of the desired output, then the error signal was set to 0 for that output. This was done to prevent the network from attempting to increase weights without limit in an attempt to drive the network outputs all the way to 0 or 1 (something that could only be accomplished with infinite weights because these are asymptotes of the activation function).

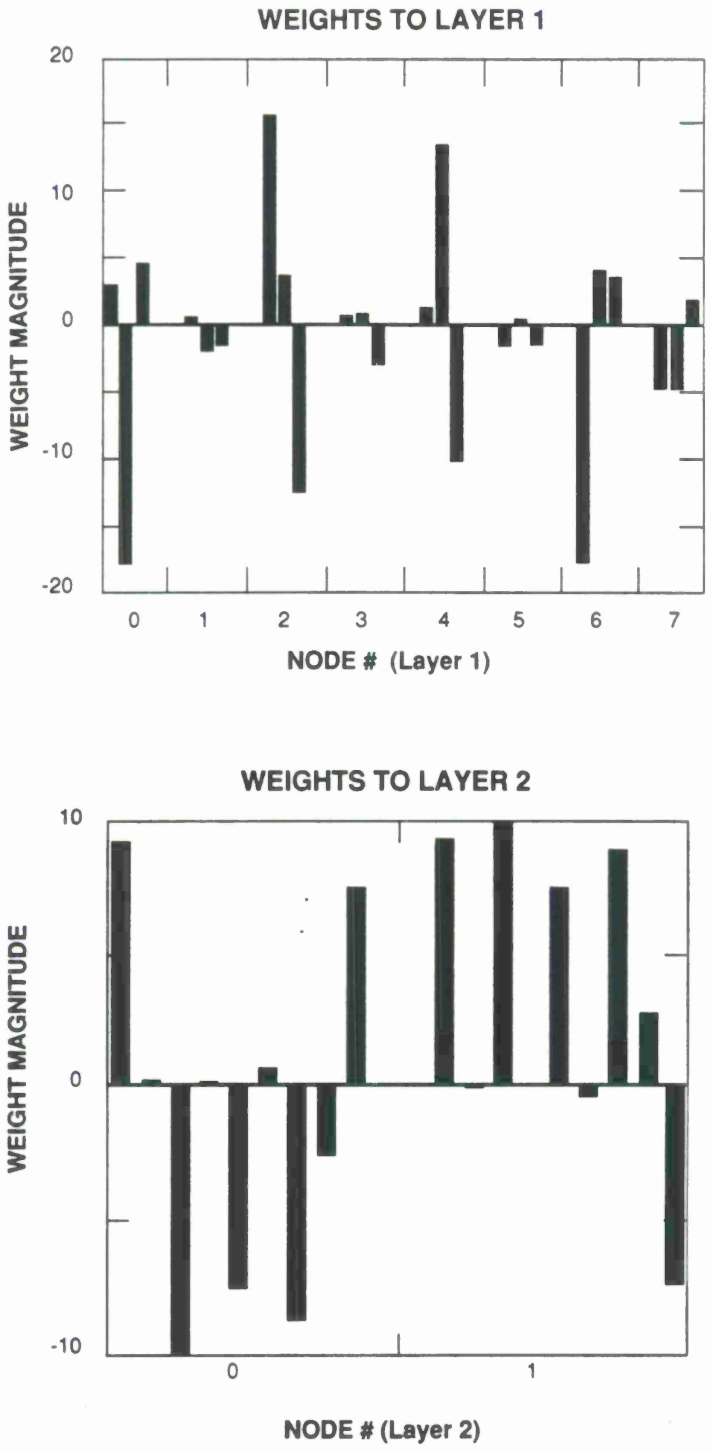


Figure A-1. Typical weight pattern for circle problem.

3. WEIGHT PATTERNS

In an attempt to be more specific about the weight patterns that were learned, Figures A-1 and A-2 show one set of weight patterns that were learned with high-precision weights for the circle and corner problems in a network that had two layers and eight hidden nodes. The performance results from these simulations were shown in Tables 5-1(a) and (b).

Careful examination of these weight patterns which were found in the trained network reveals a great deal of similarity between the layer-1 weights for the circle and corner problems. Figures A-3 through A-6 provide some insight into the reasons for this similarity. The figures consist of straight lines with normal bisecting segments, and curved contours. Each of these figures indicates the response of a single output node in a network to data in the domain space of the network.

To interpret these figures, understand that the straight lines represent the dividing lines that pass through the domain space of the problems as introduced by the first-layer nodes and (indexed by the number of the first layer node that defines the line).† Additionally, attached to each dividing line is a perpendicular segment which indicates how the particular second-layer node being represented by the figure is affected by the first-layer node corresponding to that dividing line. The length of the perpendicular segment indicates how strongly the node is influenced by that *dividing line*, and the direction of the segment indicates on which side of the dividing line the node is positively affected.

Each figure also contains contours which indicate the decision regions that were formed by the network. In the case of the circle problem, the decision region is a free-form closed contour that closely resembles the predefined circular class boundary. For the corner problem, the decision regions form in the four corners of the domain space. The boundary lines indicate the points in space where the network's two outputs are equal. On one side of these lines, output 1 is largest; on the other side, output 0 is largest.

Figures A-3 and A-4 illustrate how nodes 0 and 1, respectively, respond to data points in the input space in the "circle" problem, while Figures A-5 and A-6 indicate this same information for nodes 0 and 1 in the "corners" problem. The first interesting observation based upon these figures is that, although the second-layer weights are very different for the two problems, the first layer weights (as represented by the dividing lines) are similar. This is a result of the fact that both of the weight patterns were learned from the same initial nonrandom weight distribution as described in Section 5-B-5. During learning, several of the first-layer nodes that could separate the two data classes adjusted their position in input space. Because the data in the corner and circle problems have a similar symmetry, the same dividing lines were selected.

---

† Although there are eight nodes in the first layer, the dividing lines not shown do not pass through the domain space.

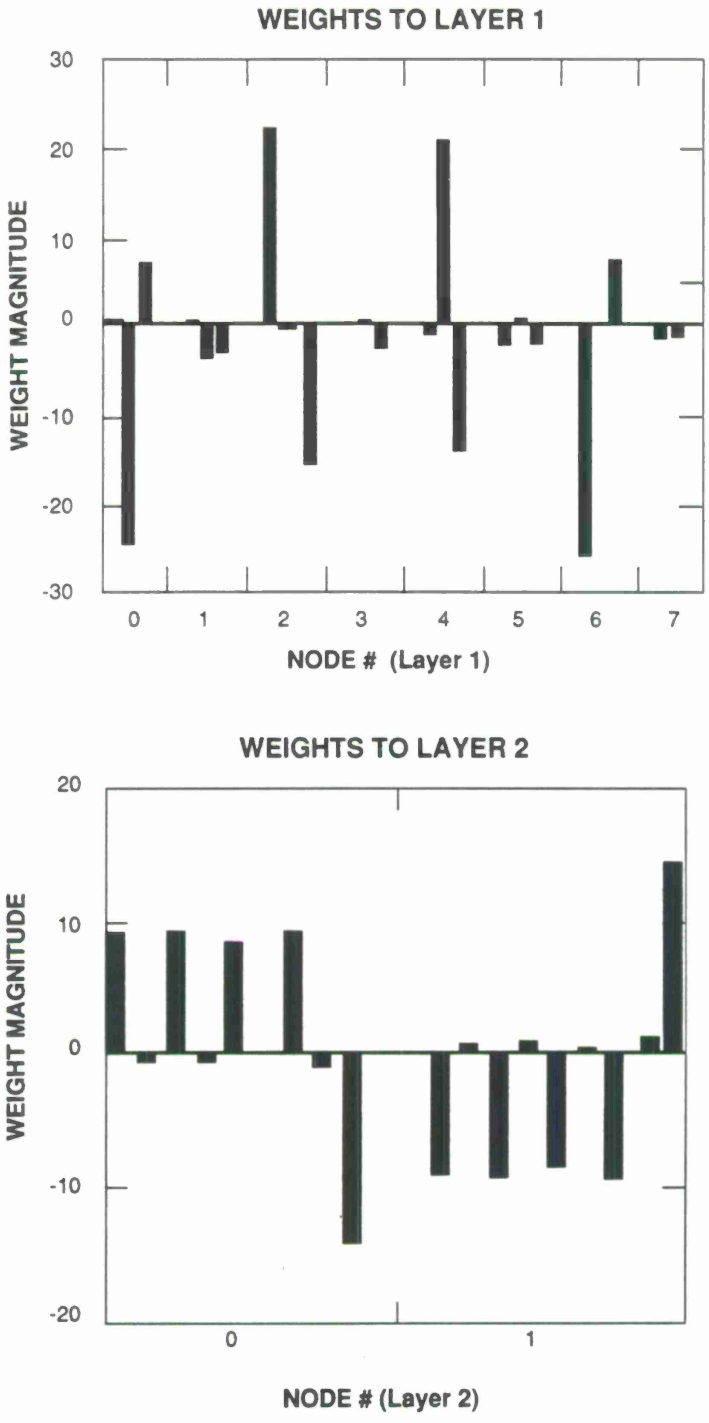
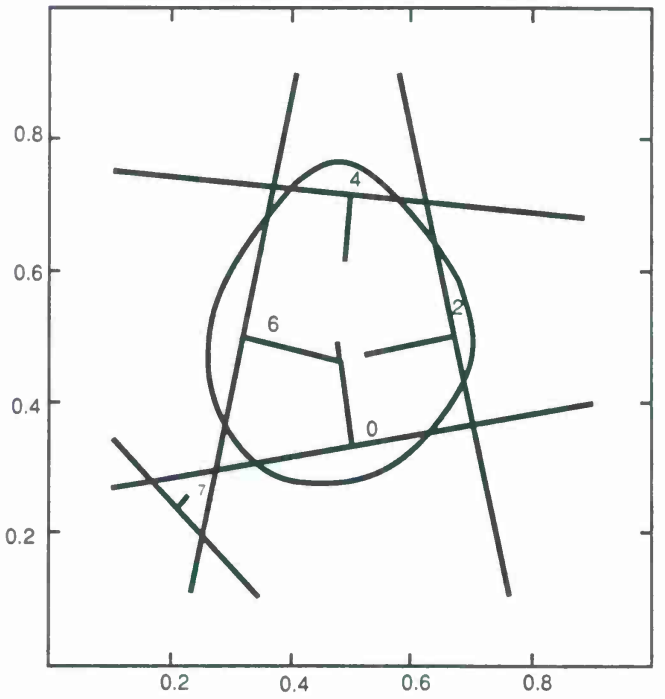
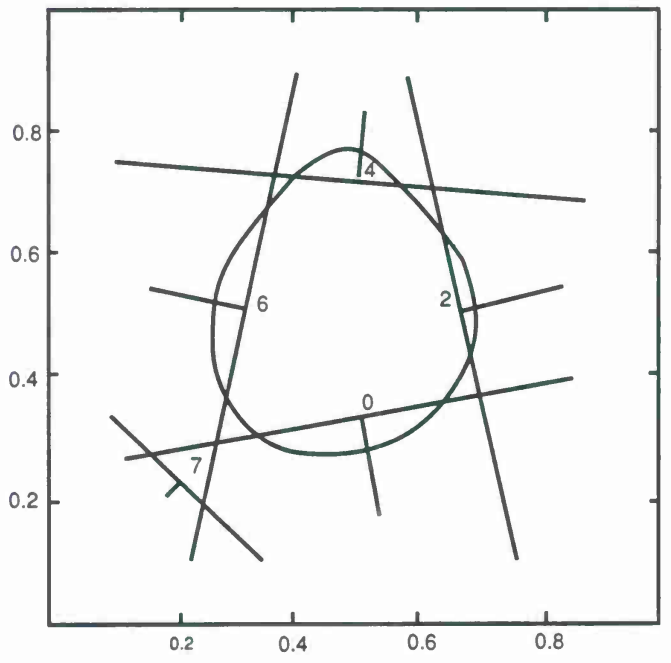


Figure A-2. Typical weight pattern for corner problem.



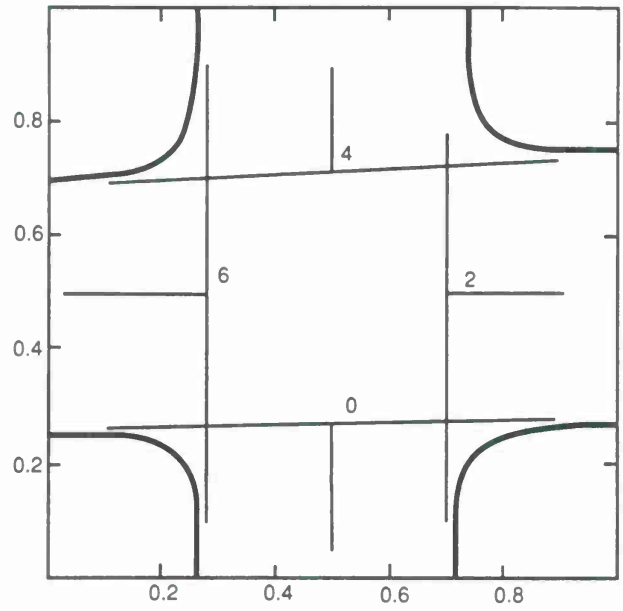
102480-28

Figure A-3. Response of node 0 for the circle problem.



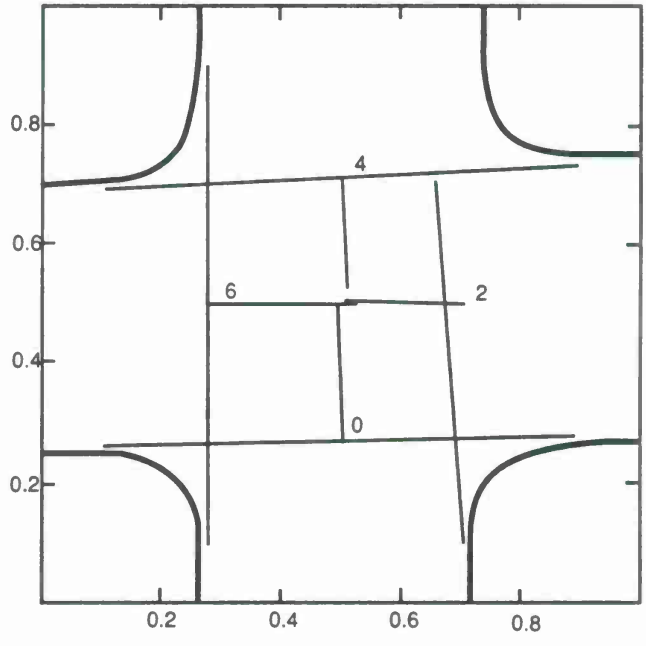
102480-29

Figure A-4. Response of node 1 for the circle problem.



102480-30

Figure A-5. Response of node 0 for the corner problem.



102480-31

Figure A-6. Response of node 1 for the corner problem.

It should also become clear from Figures A-3 through A-6 that the solutions represented by these weight patterns are not globally optimum. In the case of the circle problem, a better definition of the circular boundary would result if all eight dividing lines introduced by the first-layer nodes were arranged to form an octagon about the center of the space. In the case of the corner problem, the four dividing lines that pass through the domain space are in what seem to be ideal positions, but the other dividing lines are not contributing to the solution. A more optimum use of the network would be achieved in this case if all eight of the dividing lines passed through the domain space (perhaps several dividing lines could have the same orientation).

These observations lead to the following suggestion for future investigation. After a certain amount of learning has taken place, the input weights to first-layer nodes that are not contributing to the solution might be reinitialized. This could be done by examining all the weights that leave a particular first-layer node. If they are all below a certain threshold level, then the node is not contributing significantly to the solution, and its input weights should be reinitialized.

APPENDIX B
BIBLIOGRAPHY

Abu-Mostafa, Y.S. and D. Psaltis, "Optical Neural Computers," Sci. Am. **256**, 88-95 (1987).

Agranat, A. and A. Yariv, "A New Architecture for a Microelectronic Implementation of Neural Network Models," *Proceedings of IEEE First Annual International Conference on Neural Networks at San Diego*, Vol. III (SOS Printing, San Diego, CA, 1987), pp. 403-409.

Alspector, J. and R. Allen, "A Neuromorphic VLSI Learning System," *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference* (MIT Press, Cambridge, MA, 1987), pp. 313-349.

Brown, P.B., R. Millecchia, and M. Stinely, "Analog Memory for Continuous-Voltage, Discrete Time Implementation of Neural Networks," *Proceedings of IEEE First Annual International Conference on Neural Networks at San Diego*, Vol. III (SOS Printing, San Diego, CA, 1987), pp. 523-530.

Burr, D.J., "A Neural Network Digit Recognizer," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, October 1986.

Chiang, A.M., "A Real Time CCD Parallel Radar Processor," *Real Time Signal Processing X*, Proc. SPIE **827**, 126-130 (1987).

Huang, W.Y. and R.P. Lippmann, "Comparisons Between Neural Net and Conventional Classifiers," *Proceedings of IEEE First Annual International Conference on Neural Networks at San Diego*, Vol. IV (SOS Printing, San Diego, CA, 1987), pp. 485-494.

Kub, F.J., I.A. Mack, C.T. Yao, and K. Moon, "Programmable Analog Synapses for Microelectronic Neural Networks," presented at *Second Synthetic Microstructures in Biological Research Med. Conference,* Airlie House, VA, 1988.

Lapedes, A. and R. Farber, "Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling," preprint of Los Alamos National Laboratory Report LA-UR-87-2662 (July 1987).

Levin, E. and S. Solla, "Analyzing Protein Structure with Neural Networks," *Neural Networks for Computing Conference: Snowbird*, UT (American Institute of Physics, New York, 1986).

Lippmann, R.P., "An Introduction to Computing with Neural Nets," IEEE ASSP Mag. **4**, 4-22 (1987).

Peeling, S.M., R.K. Moore, and M.J. Tomlinson, "The Multi-Layer Perceptron as a Tool for Speech Pattern Processing Research," *Proceedings IoA Autumn Conference on Speech and Hearing*, 1986.

Qian, N. and T. Sejnowski, "Predicting the Secondary Structure of Globular Proteins Using Neural Network Models," *Neural Networks for Computing Conference: Snowbird, UT* (American Institute of Physics, New York, 1988).

Raffel, J., J. Mann, R. Berger, A. Soares, and S. Gilbert, "A Generic Architecture for Wafer-Scale Neuromorphic Systems," *Proceedings of IEEE First Annual International Conference on Neural Networks at San Diego*, Vol. III (SOS Printing, San Diego, CA, 1987), pp. 501-514.

Rosenblatt, F., *Principles of Neurodynamics* (Spartan, New York, 1962).

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, Vol. 1 (MIT Press, Cambridge, MA, 1986), pp. 318-362.

Sage, J.P., K.E. Thompson, and R.S. Withers, "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles," *Neural Networks for Computing Conference: Snowbird, UT* (American Institute of Physics, New York, 1986), pp. 381-385.

Sejnowski, T.J. and C.R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," Complex Syst. **1**, 145-168 (1987).

Tsividis, T.J., "Analogue Circuits for Variable-Synapse Electronic Neural Networks," Electron. Lett. **23**, 1313 (1987).

REPORT DOCUMENTATION PAGE

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) Technical Report 810	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-88-164
6a. NAME OF PERFORMING ORGANIZATION Lincoln Laboratory, MIT 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Electronic Systems Division
6c. ADDRESS *(City, State, and Zip Code)* P.O. Box 73 Lexington, MA 02173-0073	7b. ADDRESS *(City, State, and Zip Code)* Hanscom AFB, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Systems Command, USAF 8b. OFFICE SYMBOL *(If applicable)* XTKT	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-85-C-0002
8c. ADDRESS *(City, State, and Zip Code)* Andrews Air Force Base Washington, DC 20334	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 63250F; PROJECT NO. 22; TASK NO.; WORK UNIT ACCESSION NO.
11. TITLE *(Include Security Classification)* Implementing Artificial Neural Networks in Integrated Circuitry: A Design Proposal for Back-Propagation	
12. PERSONAL AUTHOR(S) Sheldon L. Gilbert	
13a. TYPE OF REPORT Technical Report 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT *(Year, Month, Day)* 1988, November, 18 15. PAGE COUNT 94
16. SUPPLEMENTARY NOTATION None	

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	neural networks; back-propagation; integrated circuitry; CMOS

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

In an attempt to develop CMOS circuitry (both analog and digital) for the implementation of artificial neural networks, the back-propagation learning algorithm was examined in detail. Simulations were performed to determine the robustness of this algorithm to anticipated implementational artifacts such as quantization and weight-range limitation. Circuitry which computes with analog signals and digitally encoded weights was then designed to implement the algorithm within the tolerances determined by the simulations.

The architecture of an alternative, fully digital, design was also defined and its performance compared with that of both the analog/digital design and a fully analog design based on circuitry that has been proposed in the literature.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Lt. Col. Hugh L. Southall, USAF	22b. TELEPHONE *(Include Area Code)* (617) 981-2330	22c. OFFICE SYMBOL ESD/TML