EXPLOITING CELLULAR AUTOMATA IN THE DESIGN OF CELLULAR NEURAL NETWORKS FOR BINARY IMAGE PROCESSING

by

L. O. Chua and B. E. Shi

Memorandum No. UCB/ERL M89/130

15 November 1989

EXPLOITING CELLULAR AUTOMATA IN THE DESIGN OF CELLULAR NEURAL NETWORKS FOR BINARY IMAGE PROCESSING

by

L. O. Chua and B. E. Shi

Memorandum No. UCB/ERL M89/130

15 November 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Exploiting Cellular Automata in the Design of Cellular Neural Networks for Binary Image Processing[+]

L.O. Chua and B.E. Shi[++]

ABSTRACT

We examine similarities between cellular neural networks (CNNs) [1,2] and cellular automata (CA) [3,4] to derive rules which are useful in converting CA binary image processing applications to a CNN implementation. We show that any single iteration operation possible on a CA as described in [3] can be executed with a CNN. We also give some results and discussion regarding implementing multiple iteration CA operations on CNNs. We conclude with examples which were designed using the results presented here.

I. INTRODUCTION

Both *cellular neural networks (CNNs)* and *cellular automata (CA)* have applications in image processing [1,2,3,4]. The architectures of cellular neural networks and cellular automata are very similar. Both are composed of arrays of cells, each having an internal state whose dynamics are driven by the states of its nearest neighbors and an external input. The main difference between the two architectures is that while the states of cells of CA evolve in a discrete state space in discrete time, the states of cells of CNNs evolve in a continuous state space in continuous time.

Preston and Duff describe the theory and applications of cellular automata in [3] with an emphasis upon image processing. Much of their discussion assumes an architecture based on the CLIP4 machine described by Duff and Fountain [4]. In this machine, each pixel in the image plane has a boolean processing element (cell) associated with it. Although CLIP4 can process a 96 by 96 pixel image with 64 levels of gray scale, the gray scale values must be held in memory external to the processing array. In fact, Preston and Duff discuss primarily operations on binary images. In this case,

+ This work is supported in part by the Office of Naval Research under Grant N00014-89-J-1402 and the National Science Foundation under Grant MIP-86 14000 and by a National Science Foundation Graduate Research Fellowship
++ The authors are with the University of California at Berkeley Electronics Research Laboratory

the image can be held in registers internal to the processor array. We shall assume this to be the case in the following.

At each discrete time iteration, a supervising controller determines the computation to be performed by the processing array. Each iteration consists of two phases: a processing phase preceded by a propagation phase. In the processing phase, all the cells' states are updated simultaneously. The value of the next state of each cell is a boolean function of its current value and the inputs from its nearest neighbors. This operation is referred to as a cellular logic transform. The propagation phase establishes the input values. In the classical definition of a cellular automata, each cell propagates its value to its neighbors. Preston and Duff refer to this as local propagation. Preston and Duff also allow for global propagation. In this case, a cell will propagate a signal based on the value of its own state and whether it has received a signal from one of its neighbors. For example, assume that the state of cells associated with background pixels is 0 and the state of cells associated with image pixels is 1. If a cell propagates a signal if it receives a propagation signal and its own value is 1, then a signal will be propagated throughout a connected component in the image. This propagation phase allows data to be passed over the entire array, overcoming many of the restrictions of the local interconnectivity of the processing elements.

On the other hand, in the CNN described by Chua and Yang each pixel has an analog processor associated with it. Theoretically, due to the analog nature of the processors, a gray level image of arbitrary precision can be held internally in the processing array. However, the CNN can be designed to ensure that the output state of each cell will settle to either ± 1. We shall discuss CNNs in more detail in Section II.

The natural question which arises is what types of operations can be accomplished by both the CNN and CA. Clearly, designing applications for CA is much easier than designing applications for CNNs. The synchronous discrete time/state dynamics of CA is much easier to understand than the asynchronous continuous time/state dynamics of the CNN. In addition, the processors of a CNN are hard wired together so that there is no external control. We simply initialize the state and input and allow the circuit to settle. However, we expect the CNN to have advantages in processing gray level images. Moreover, CNNs are ideally suited for ***VLSI implementation.*** Since the cells of a CNN operate asynchronously, there is no need for a clock signal to be routed to each cell. This considerably simplifies the VLSI layout especially for large arrays of cells. It also eliminates problems associated with clock skew when a clock signal is propagated for long distances on a chip. For example, the CLIP4 processing array alone consists of 1152 integrated circuits. On the other hand, a CNN designed to perform noise removal on a 20 by 20 pixel binary image has been implemented on a single chip!

In this paper, we restrict ourselves to binary image processing operations and examine the intersection of the applications of CNNs and CA in this area. Our approach is to exploit the similarities between CNNs and CA in order to convert image processing operations implemented on CA to a CNN format. We shall see that the two phases of a CA iteration defined above can be replicated by a CNN. Therefore, *any single iteration operation possible on a CA can also be accomplished on a CNN.* Our results are not as neat for operations requiring multiple iterations.

Some of the ideas we present here have been presented in other contexts associated with artificial neural networks, and will be familiar to those readers who have been following the neural network literature. We will identify these as they arise. This paper combines these results into useful rules which assist in the design of CNNs for image processing (and possibly other) applications. In Section II, we define a cellular neural network and slightly extend the definition of multiple layer CNNs given in [1] to encompass some new results in CNNs. In Section III, we define a class of CNN cell array layer for which the map from input and initial conditions to output steady state has a simple closed form. Using layers of this class, we show that any single iteration of local or global propagation followed by a cellular logic transform can be accomplished by a CNN. In Section IV, we discuss implementing multiple iteration operations of CA on CNNs. We then restrict the discussion to CA applications involving infinitely iterated operations based on local propagation and present some results in this area. For example, we give a necessary and sufficient condition for a vector of binary output states to correspond to an asymptotically stable equilibrium point of a multiple layer CNN with asymmetric coefficients. This condition is expressed in terms similar to the analogous condition for CA. Section V uses the results of the Section III, in the design of a CNN which removes the corners of a binary image corrupted by noise. In addition, this section presents simulation results of other image processing CNNs designed with the techniques presented here. Finally in Section VI, we provide an extension of the discussion in Section IV with an example of a CNN which finds one projection of the Radon Transform of a binary image.

II. CELLULAR NEURAL NETWORKS

As presented in [1], a cellular neural network is composed of an array of cells each having an associated state. For image processing operations each pixel in the image plane has an associated cell. Since we will only discuss image processing operations, for the remainder of this paper we shall assume that the image plane is divided into M by N pixels. For a single layer CNN, the state of each cell is a scalar variable. For an L layer CNN, the state is an L-dimensional vector. A completely equivalent way to view a multiple layer CNN is as an L by M by N array of single layer cells. We shall adopt the latter interpretation as it simplifies the discussion and is intuitively appealing. Thus, each cell will be denoted $C(k,i,j)$ where $1 \le k \le L$, $1 \le i \le M$ and $1 \le j \le N$. The *output* $y_{k,i,j}$ of $C(k,i,j)$ is a piecewise-linear function of its state:

$$y_{k,i,j} = f(v_{k,i,j}) = \tfrac{1}{2}|v_{k,i,j}+1| + \tfrac{1}{2}|v_{k,i,j}-1| .$$

This restricts the output of each cell to lie in the interval $[-1,1]$. See Figure 1. Typically, we associate +1 values of input and output with image pixels and -1 values with background pixels.

The *state* v *of the CNN* is defined to be the vector of the states of all the cells in the network. The *input* u *to the CNN* is defined to be the vector of the inputs to the cells. Similarly, the *output* y *of the CNN* is defined to be the vector of the output of the cells. It will also be useful to define the *kth cell array layer* as the set of cells in the kth layer of the CNN and denote the associated state by $\mathbf{v}_k$.

The state of each cell can assume a continuum of values. Its time derivative is an affine function of its current state and the outputs of itself and its nearest neighbors. This restriction to nearest neighbor interactions is imposed to limit the number of interconnections between cells. Due to the complexity and number of interconnections required, a reasonably sized fully interconnected network, such as the Hopfield[5], is practically impossible to build with today's VLSI technology.

To clarify the term *nearest*, we define the distance between $C(k,i,j)$ and $C(l,m,n)$ to be $d(k,i,j;l,m,n) = \max\{\ |m-i|, |n-j|\ \}$. Note that this distance is independent of the layer. Using this metric, we define the *r-neighborhood* of a cell $C(k,i,j)$ to be

$$N_r(k,i,j) = \{\ C(l,m,n)\ |\ d(k,i,j;l,m,n) \le r;\ 1 \le l \le L;\ 1 \le m \le M;\ 1 \le n \le N\ \}$$

For $C(k,i,j)$, we define the smallest r-neighborhood which contains all the cells which determine the cell's dynamics as the *active neighborhood* of $C(k,i,j)$. The restriction to nearest neighbor interconnections dictates that the size of the maximum active neighborhood of the CNN is much smaller than both M and N. For many applications, the active neighborhood will be the cell's 1 or 2-neighborhood.

We often wish image processing operations to be invariant under translation of the image. This condition requires that the nature of the interaction between the cells be uniform over the entire array. Of course, the cells on the boundary must be treated separately. Usually this is done by creating imaginary cells outside of the boundary whose values are constant. For these CNNs, the state equation of one cell from each layer is enough to specify the equations for the entire array. Since these state equations are affine functions, the state equations are uniquely specified by the vectors of coefficients of the affine relations. We will define these vectors of coefficients to be the CNN's *cloning template*.

For this paper, the differential equations governing the state of each cell are similar to the equations presented in [1]. The state $v_{k,i,j}$ of $C(k,i,j)$ satisfies the following differential equation:

$$C_k \frac{dv_{k,i,j}}{dt} = -R_k^{-1} v_{k,i,j} + \sum_{\gamma=1}^{L} \sum_{\alpha=-r}^{r} \sum_{\beta=-r}^{r} A_{k,\gamma}(\alpha,\beta) \cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha=-r}^{r} \sum_{\beta=-r}^{r} B_k(\alpha,\beta) \cdot u_{i+\alpha,j+\beta} + I_k \qquad (1)$$

where r is the size of the active neighborhood and C_k, R_k are positive constants for all k. A circuit realization [1] of this equation is shown in Figure 2, along with a diagram illustrating the connectivity pattern of a single cell in layer 2 of a three layer network. To ensure that we have ± 1 steady state outputs we should have $A_{k,k}(0,0) > R_k^{-1}$ for all k.

The coefficients of $A_{k,\gamma}$ weight the effect of the output of the γth cell array layer on the derivative of the state of the kth cell array layer. The coefficients of B_k weight the effect of the input on the derivatives of the state of the kth cell array layer. As discussed above, $A_{k,\gamma}$, B_k and I_k are independent of i and j. The coefficients of the $A_{k,\gamma}$, B_k and I_k for $1 \le k,\gamma \le L$ form the cloning template for this

CNN. Because of the two dimensional and local nature of the interactions, it is convenient to express the coefficients of $A_{k,\gamma}$ and B_k as $2r+1$ by $2r+1$ matrices where the center element corresponds to the coefficient which weights the cell's own output or input. See Figure 8 for an example with $r = 1$.

This equation differs from that presented in [1], in that we include contributions to the dynamics of layer k from layers of higher index than k. This assumption was taken in [1] to ensure stability for multiple layer CNNs with symmetric coefficients within layers. By symmetric coefficients within layers, we mean that $A_{k,k}(\alpha,\beta) = A_{k,k}(-\alpha,-\beta)$. However, recently a connected component detecting CNN[6] and a thinning CNN[7] have been developed using asymmetric coefficients. In addition, the thinning CNN is a multiple layer CNN whose cells' state equations have the form (1). Unfortunately, not all CNNs of this form will be stable.

Those familiar with the neural network literature will recognize the above equations as being similar to those of the Hopfield type neural networks. However, the key differences are that we restrict ourselves to local interconnects and use a piecewise linear sigmoid nonlinearity which allows for ± 1 steady state outputs without the assumption of infinite gain in the sigmoid non-linearity. We also allow for the self feedback term from the cell's output to its input. In addition, although a multiple-layer Hopfield network is encompassed in the form of its dynamical equation, we are not aware of any research explicitly relating Hopfield networks and multiple-layer neural networks[8].

III. SINGLE ITERATION CELLULAR AUTOMATA OPERATIONS

In general, we do not know if a CNN is stable nor can we easily predict the state trajectories short of simulating them. We now examine a class of CNN cell array layer for which, under certain conditions, CNNs made exclusively of this type of layer are stable regardless of the symmetry of the coefficients. Under these conditions, there exists a simple closed form expression for the map from input and initial conditions to steady state output. These types of layers can be used to implement the processing phase of a CA iteration.

Definition: A CNN cell array layer, k, is said to be in the *linear threshold class* if and only if $A_{k,k}(i,j) = 0$ for all $(i,j) \neq (0,0)$.

Essentially, this definition prohibits any interconnections among the cells within a linear threshold layer. The dynamics of the cells depend only upon their own states, the states of the other layers, and the external input.

To see the rationale behind this definition, consider a CNN composed of a single linear threshold layer driven by a constant input. If we partition the right hand side of eqn. (1) into a part, $h(v_{1,i,j})$, which depends upon $v_{1,i,j}$ and a part, g, which does not, we can rewrite the state equation of $v_{1,i,j}$ as

$$C_1 \frac{dv_{1,i,j}}{dt} = h(v_{1,i,j}) + g \qquad (2a)$$

where

$$h(v_{1,i,j}) = -R_1^{-1} v_{1,i,j} + A_{1,1}(0,0) \cdot f(v_{1,i,j}) \qquad (2b)$$

and

$$g = \sum_{\alpha}\sum_{\beta} B_1(\alpha,\beta) \cdot u_{i+\alpha,j+\beta} + I_1 \qquad (2c)$$

Figure 3 shows the graph of $h(v)$.

In this case, $g = \sum_{\alpha}\sum_{\beta} B_1(\alpha,\beta) \cdot u_{i+\alpha,j+\beta} + I_1$ is constant. Figure 4 depicts the dynamic routes [9, 10] for three characteristic g values. If $|g| > A_{1,1}(0,0) - R_1^{-1}$, then there exists only one equilibrium point for the cell. Since this equilibrium point is globally asymptotically stable, the state of the cell settles to this point. Thus, if g is positive (resp. negative), the output of the cell is +1 (resp. -1). If $|g| < A_{1,1}(0,0) - R_1^{-1}$, then there exist two stable equilibrium points and one unstable equilibrium point. The state eventually settles to one of the stable equilibrium points depending upon the initial condition of the cell. If we assume binary initial conditions so that $y_{1,i,j}(0) = v_{1,i,j}(0)$, then the state will settle to a value which maps to an output value equal to the output value of the initial condition. The case of $g = A_{1,1}(0,0) - R_1^{-1}$ is somewhat problematic. In this case, there are two equilibrium points. One is stable while the other is unstable. Theoretically, assuming binary initial conditions, this case is identical to the case where $|g| < A_{1,1}(0,0) - R_1^{-1}$. However, in practice due to inevitable thermal noise, the trajectories will eventually approach the stable equilibrium point. Fortunately, we will see that for many applications, this case is not encountered. Thus, we can summarize the above results with the following equation:

$$\lim_{t\to\infty} y_{1,i,j}(t) = \mathrm{sgn}\left[(A_{1,1}(0,0) - R_1^{-1})\cdot y_{1,i,j}(0) + \sum_\alpha\sum_\beta B_1(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_1\right] \quad (3)$$

Note that if $A_{1,1}(0,0) = R_1^{-1}$, then the final state of the output of the cell depends solely on the external input, unless $g = 0$.

The left hand side of equation (3) is a signum function of an affine combination of input and output variables. This type of function is commonly referred to as a linear threshold function (see appendix). Any boolean function which can be expressed in this form must be linearly separable by a hyperplane in the variable space. In the appendix we show that any boolean function which is linearly separable can be implemented with a CNN such that the signum function in (3) is well defined (i.e., the quantity inside the signum is never zero). This fact implies that *any linearly separable boolean function of the input and initial condition of the state can be implemented using a CNN composed of a single linear threshold layer.* In particular, we can implement the logical convolution and binary mask matching operations of [3] with a single linear threshold layer. To do this, we need only set the B_1 matrix equal to the mask or the convolution kernel, set the current I_k equal to the corresponding threshold, set $A_{1,1}(0,0) = R_k^{-1}$ and set the input equal to the image to be transformed.

Minsky and Papert studied linear threshold functions in their book *Perceptrons[11]*. However, the above analysis does not imply that CNNs suffer from the same restrictions as perceptrons. Most of Minsky and Papert's results showed the limitations of perceptrons as pattern classifiers. Here we are not interested in CNNs as pattern classifiers, except perhaps as pre-processors which extract features for input to a pattern classifier. Our applications are in the area of image processing involving transformations of the image which extract useful information. As we shall see below, the continuous time dynamics and multiple layer capability of CNNs extend the possible applications of CNNs beyond those of simple perceptrons. In addition, we stress that this type of layer is only a small subset of all possible CNN cell array layers.

The operations possible by these templates would be quite limited indeed if the only boolean functions which could be implemented are those which are linearly separable. Fortunately, using *multiple layer feedforward* CNNs removes this restriction.

Definition: A CNN is a *feedforward cellular neural network* if and only if $A_{i,j} = 0$ for all $i < j$.

This definition is equivalent to the definition of a multiple layer CNN given in [1]. If all the layers of a feedforward CNN are of the linear threshold class and $A_{k,k}(0,0) = R_k^{-1}$ for all $k > 1$, then the steady state of each layer above the first depends only upon the steady state outputs of the previous layers and the input. The steady state of the first layer depends only upon the input and its initial conditions. Thus, the outputs of all the cells of the CNN depend only upon the initial conditions of layer 1 and the input to the CNN. We have ensured stability by ruling out any feedback paths between cells in our definition of a feedforward CNN and a linear threshold layer. In fact, we have implemented a two dimensional array of feedforward hidden layer neural networks [8]. It is well known that a feedforward hidden layer neural network with enough hidden layers can implement any boolean function of its inputs. In this case, these inputs are the inputs of the nearest neighbor cells. Therefore, if we simply take the input of the CNN to be the image to be transformed (the current state in the case of CA), *any single iteration CA operation involving only local propagation can also be accomplished with a CNN composed solely of linear threshold layers.*

We can extend this result to include global propagation. Recall that a CA cell will propagate a signal based upon two factors: the value of its current state and whether it has received a propagation signal from one of its neighbors. Clearly, a propagation rule in which a cell will propagate a signal if it has not received a propagation signal will not lead to global propagation of data. In fact, a rule of this type essentially implements local propagation. Global propagation over the array can only be accomplished by propagation rules in which a cell propagates a signal if it has received a signal from one of its neighbors coupled with some condition on the value of the current state. Thus, we consider only rules of this type.

In order to realize global propagation in a CNN, we add an additional layer which performs the global propagation. If we design layer 1 of a CNN so that its cells' outputs are +1 if the cell is propagating a signal and -1 otherwise, we can add a set of feedforward linear threshold layers satisfying $A_{k,k}(0,0) = R_k^{-1}$ on top of layer 1 to perform a cellular logic transform. Under these conditions, the outputs of the linear threshold layers will depend solely upon the steady state of layer 1 (analogous to the propagation signal from each cell) and the input (analogous to the current state of the CA for the case of single iteration operations). This is clearly equivalent to global propagation of data in CA.

Since the other cases are quite similar, we discuss only the case where a cell will propagate a signal if it has received a signal and the value of its input is +1. Consider the single layer CNN defined by the following template:

$$A_{1,1} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \qquad B_1 = [\,3.0\,] \qquad I_1 = -1.0$$

We assume that $R_1 = C_1 = 1.0$ since we are only interested in the dynamics.

The analysis of the state trajectories for this CNN turns out to be somewhat similar to the analysis for a linear threshold layer. The state equation of each cell is also given by (2a). However, in this case, g depends also on other cells in layer 1.

$$g = 0.25 \sum_{\alpha,\beta \neq 0,0}\sum y_{1,i+\alpha,j+\beta} + 3.0u_{i,j} - 1.0$$

Note that $A_{1,1}(0,0) - R_k^{-1} = 0.25$.

Consider cell $C(1,i,j)$. For $u_{i,j} = -1$, $g < -0.25$ for all combinations of $y_{1,i+\alpha,j+\beta}$. Thus, in steady state $y_{i,j} = -1$. For $u_{i,j} = 1$, the quantity in the signum will be greater than 0.25 whenever at least one $y_{1,i+\alpha,j+\beta}$ equals one. This provides the propagation. There is one subtle point here. Even though a cell in the background directly adjacent to an image component which is initialized to +1 has -1 output in steady state, it could start propagation through the adjacent image component due the finite time required for the transition of its output from +1 to -1. However, we have designed the CNN so that this transition is essentially instantaneous compared to the time required to start propagation. Thus, a cell will have +1 output if and only if it is associated with an image component and was either initialized to +1 or received a propagation signal from a nearest neighbor.

Another discussion of this global propagation is contained in [12]. Since we have now implemented global propagation on a CNN, it follows that *any single iteration CA operation consisting of a propagation phase (local or global) and a cellular logic transform can be accomplished with a CNN.*

IV. MULTIPLE ITERATION CELLULAR AUTOMATA OPERATIONS

Before we proceed with some design examples which utilize the above results, we digress to discuss CNN implementation of multiple iteration CA operations. Although the CNN can accomplish any single iteration CA operation, multiple iteration operations are more difficult. If the operation requires a small number of iterations, we can cascade CNNs designed to do each iteration into one large CNN. To ensure that the iterations are carried out sequentially, we could design the layers so that the layers corresponding to the first iteration settle much faster than those corresponding to the second and so on. We have used this technique in some of the results presented in Section V. However, this is limited to a operations with small number of iterates since in practice the capacitance and resistance values must be scaled to slow down the dynamics of progressive layers, leading quickly to impossibly large capacitors and resistors for a VLSI chip.

For the rest of this section, we restrict the discussion to implementing CA which continually iterate the same cellular logic transform with only local propagation. For a CA of this type, an output vector will be a equilibrium point if it is a fixed point of the boolean map governing the state transitions. Due to the local interconnectivity of the CA, this condition is equivalent to a local condition which must be satisfied at all cells in the array. For example, say the state transition rule for cell i,j is $x_{i,j}(t+1) = s(x_{i-1,j}(t), x_{i,j}(t), x_{i+1,j}(t))$, where s is a boolean function. Then an output vector $\{ x_{i,j} \}$ is an equilibrium point of the CA if and only if $x_{i,j} = s(x_{i-1,j}, x_{i,j}, x_{i+1,j})$ is satisfied at each cell of the array. The following results shows that a similar case holds for CNNs in binary image processing.

Claim 1: *If (4) is well defined (i.e., the quantity inside the signum is never zero) for all binary vectors* $\{ \hat{y}_{k,i,j} \}$, *then a binary output state,* $\hat{\mathbf{y}} = \{ \hat{y}_{k,i,j} \}$ *satisfies the implicit equation*

$$\hat{y}_{k,i,j} = \mathrm{sgn}\left[(A_{k,k}(0,0) - R_k^{-1}) \cdot \hat{y}_{k,i,j} + \sum_{\gamma,\alpha,\beta,\neq k,0,0}\sum\sum A_{k,\gamma}(\alpha,\beta) \cdot \hat{y}_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta) \cdot u_{i+\alpha,j+\beta} + I_k \right] \quad (4)$$

for all k,i,j, *if and only if there exists an equilibrium point* $\hat{\mathbf{v}} = \{ \hat{v}_{k,i,j} \}$ *of the CNN which maps to* $\hat{\mathbf{y}}$ *through the output non-linearity. Furthermore, this equilibrium point is asymptotically stable and its basin of attraction contains the neighborhood in which the states of all the cells are operating in the saturated region of the piecewise-linear output non-linearity.*

The proof of this claim is contained in the appendix. We have noted above that for many applications the assumption of the claim is satisfied. Essentially, Claim 1 states that if a CNN's output satisfies (4) for all k,i,j, the output of the CNN will no longer change and the state converges to a stable equilibrium point. Due to the local nature of the interactions between the cells of a CNN, this necessary and sufficient condition for a stable output equilibrium point is also expressed in terms of a local condition which must be satisfied for all cells in the array. To emphasize the similarity with the CA case, we can consider the signum function in (4) to be a boolean function since it maps $\{-1,1\}^n$ to $\{-1,1\}$. This equation provides a necessary condition for a CNN to execute a CA operation since the same output vectors must be stable in both cases. Thus, (4) provides a convenient starting point for converting CA applications to CNN applications.

What can we say if (4) is not satisfied? For a cellular automata we can predict the next state of the network based upon the boolean rule governing the updates. For a CNN, the continuous time/state dynamics make the situation more complex. This is where the difficulty in designing cloning templates for CNN's arises. However, we can obtain a weaker result.

Claim 2: *If the output vector $\hat{\mathbf{y}}$ corresponding to a state vector $\hat{\mathbf{x}}$ does not satisfy (4) for all k,i,j, then the state of at least one the cells which does not satisfy (4) will enter the linear region of the piecewise-linear sigmoid output function (Fig. 1).*

This claim is also proved in the appendix. Thus, the output states which do not satisfy (4) are unstable. In fact, the proof of Claim 2 shows that for some time the outputs of cells which do not satisfy (4) progress towards the output state which are dictated by the signum function of (4). Interestingly, a CA implementation of the corresponding signum function of the connected component detector[6] does settle to the desired steady state. However, because of the complex dynamical behavior associated with the continuous time/state dynamics of the CNN, in general we cannot view this signum function as an approximate boolean function which the CNN is implementing. In particular, although (4) is in the form of a linear threshold logic function, we will show by the example of the Radon Transforming CNN that this does not limit the CNN to execute only CA operations described by a linearly separable boolean state transition function.

V. EXAMPLES

We now present a simple example which uses the above results to design a two layer CNN which extracts the corners from a noisy image, along with simulation results of other CNNs also designed using the above results.

Consider the corner detecting CNN presented in [2]. Although the CNN easily detects the corners of a noise free image, noise often results in false corner detection. For example, Figure 5(d) shows the steady state output of the corner detecting CNN introduced in [2] when presented with an image of the letter "a" corrupted by Gaussian noise of variance 0.3. We see that the noise causes some corner pixels to be missed and other pixels to be misclassified as corner pixels. It would be helpful if we could somehow cascade a corner detection CNN with the noise removal CNN to create a two layer CNN which filters the image before extracting the corners.

We could use the noise removal CNN to remove the noise from the input image and then initialize the corner detecting CNN to the steady state output of the noise removal CNN. However, this operation requires an intermediate step in which the output must be read from one chip and passed to another. Not only will this intermediate step take extra time, new noise will probably be added to the image during this step and we have the same problem. A CNN which *simultaneously* removes noise and extracts the corners would be more desirable.

If we proceeded rather naively, we might cascade the noise removal CNN and corner detection CNN presented in [2] by connecting the output of the noise removal layer to the input of the corner detection layer and initializing both layers to the same (noisy) image. The result of this computation is shown in Figure 5(e). In this case, there are no false positives, but most of the corners are missed. These corners are missed because the steady state of the corner detection layer is affected by the transient response of the noise removal layer and its own noisy initial conditions.

Let us use the results above to analyze the operation of the corner detection CNN of [2]. The cloning template of this CNN is:

$$A_{1,1}(0,0) = 2.0 \qquad B_1 = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2.0 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad I_1 = -3.0.$$

Here we have assumed that $R_1 = C_1 = 1.0$ since we are only interested in the dynamics and not in actual implementation. The input and initial conditions of the CNN are the input image.

Since this CNN consists of a single linear threshold layer with constant input, we can use (3) to explicitly characterize the map from input and initial conditions to the output steady state:

$$\lim_{t\to\infty} y_{1,i,j}(t) = \text{sgn}\left[y_{1,i,j}(0)) + 2.0u_{i,j} + 0.25n_{-1} - 0.25n_1 - 3.0\right]$$

where n_{-1} and n_1 are the numbers of neighbors in the 1-neighborhood whose inputs are -1 and +1 respectively. Since $n_{-1} + n_1 = 8$ and $y_{1,i,j}(0) = u_{i,j}$, we can further simplify the above equation to

$$\lim_{t\to\infty} y_{1,i,j}(t) = \text{sgn}\left[(3.0u_{i,j} - 3.0) + (2.0 - 0.5n_1)\right]$$

The above equation shows that if $u_{i,j} = -1$, then in steady state $y_{1,i,j} = -1$. If $u_{i,j} = +1$, then $y_{1,i,j}=1$ if and only if less than five of $C(1,i,j)$'s neighbors' inputs are high.

Thus, we can redesign corner detection CNN so that the steady state of its output depends solely on its input. The resulting layer is a linear threshold layer with the following cloning template:

$$A_{1,1}(0,0) = 1.0 \qquad B_1 = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2.0 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad I_1 = -1.75.$$

Again we have assumed that $R_1 = C_1 = 1.0$. Given identical input images, the two CNNs will settle to the exact same steady state. However, because the CNN presented here is independent of its initial conditions, cascading it with the noise removal layer presented in [2] produces a CNN which settles to a steady state very close to the desired steady state (Figure 5(f)). In fact, the pictures differ only by the pixel associated with the tip of seraph of the "a". That pixel has been removed by the noise removal operation! This combined CNN has the additional advantage that only the noise removal layer need be initialized. The output is independent of the initial state of the corner detection layer.

Formulas (3), (4) and the propagation layer have been used to translate many CA applications presented in [3] to a CNN format, such as:

1. Extract the holes in an image
2. Extract the objects in the image which contain holes
3. Extract objects larger than a 3x3 square
4. Extract the connected component containing a specified image pixel
5. Find the minimal circumscribing octagonal convex hull.

See Figure 6 for sample input and output images. The appendix contains the cloning templates and a description of each of these CNNs.

Up to this point, we have used the linear threshold class as a means of working around the effects of the continuous time/state dynamics. We now present an example which uses linear threshold layers to produce a CNN whose operation depends critically upon the effects of the continuous dynamics of the CNN.

VI. A RADON TRANSFORMING CELLULAR NEURAL NETWORK

Introduction

The CNN described here computes one projection of the Radon Transform of a binary image. For any given image, define $I(x,y)$ to be the image intensity at the point (x,y) in the image plane. The Radon Transform of the image is a function $\Psi(\rho,\theta)$, where ρ and θ are polar coordinate variables. For specific values of ρ_0 and θ_0, $\Psi(\rho_0,\theta_0)$ is the integral of $I(x,y)$ along the line which is perpendicular to the line $\theta = \theta_0$ and passes through the point (ρ_0,θ_0). Thus, for fixed θ_0, the resulting one dimensional function $\psi(\rho) = \Psi(\theta_0,\rho)$ is the projection of the image intensity onto the line $\theta = \theta_0$. See Figure 7. For binary images, the horizontal, vertical and diagonal projections can be used to find the position and orientation of an object in the image plane [13].

The CNN template presented here finds the value of $g(\rho,\frac{\pi}{2})$ for a binary image. In other words, the CNN integrates the image intensity along the horizontal rows of the image plane. To obtain the full radon transformation, the image input can be rotated through all desired angles. The output $g(\rho,\frac{\pi}{2})$ is presented as a histogram along the right hand side of the image plane.

An Infinitely Iterated Cellular Logic Transform

In order to utilize the ideas presented in this paper, we first consider one way to compute one projection of the Radon Transform of a discretized binary image using an infinitely iterated cellular logic transform. Since the projection operates on each horizontal line independently, to simplify the discussion we consider only one horizontal line. For each pixel in the image plane, assign the value 1 to that pixel if it is in the image and 0 otherwise. At each time step, image pixels which have a background pixel on their right shift right. In other words, the value of each pixel at time $t+1$ is determined by

the following *boolean* function of the values at time t of the pixel (c_t) and its left (l_t) and right (r_t) neighbor pixels:

$$c_{t+1} = c_t r_t + l_t \overline{c}_t . \tag{5}$$

where $\overline{c}_t$ is the complement of c_t. Equation (5) is shown graphically in Figure A2. The boundary conditions should be set up so that the pixels on the left hand side of the image plane have $l_t = 0$ for all t and the right hand side cells have $r_t = 1$ for all t. As time progresses, pixels continue shifting until they reach either the end of the row in the image plane or another image pixel. In steady state, the pixels have all "piled up" on the right hand side of the image plane. Since the number of pixels in each row is preserved, the resulting logical steady state is a histogram representation of the projection operation.

The CNN Cloning Template

The CNN cloning template is shown in Figure 8. We have again set $R_k = C_k = 1.0$ since we are only interested in the dynamics. Scaling the values to implementable values only changes the time scale of the dynamics, but not the state trajectories. The CNN consists of three layers. The cells of the layers 1 and 3 are initialized to +1 or -1 depending upon whether the corresponding pixel is in the image or in the background. The cells of layer 2 are all initialized to -1. When the circuit finally settles, the output states of layer 1 and layer 3 each contain to the output histogram.

To see the similarity between the operation of the CNN circuit and the iterated cellular logic transform, assume for the moment the state of the cells in layer 1 is constant. We examine the steady state of layers 2 and 3 which results from this condition.

Essentially, layers 2 and 3 compute the next logical state of the iterated cellular logic transform. Notice that all of the templates are of the linear threshold class. Since the boolean function (5) is not linearly separable in the (l,c,r) boolean variable space (Figure A2), one linear threshold layer is not sufficient to implement this function.

Based on the current state of layer 1, the cells whose outputs are +1 in layer 2 are those which correspond to pixels which should shift right in the next step of the iterated cellular logic transform. The outputs of the other cells of layer 2 are -1. If the output of a cell in layer 2 is +1, then the output of its corresponding cell in layer 3 is -1 and the output of the cell to its right is +1. Otherwise the outputs of the cells of layer 3 remain equal to their initial values. Thus, the output of layer 3 corresponds to the next logical state of the iterated cellular logic transform. Note that the output of layer 2 and the initial conditions of layer 3 are sufficient to determine the next output of layer 3.

In actual operation, the state of layer 1 is not constant. The design of the A_{13} coefficients ensures that the outputs of layer 1 track the outputs of layer 3 with some delay due to the time constants of the dynamics. In the circuit implementation, this delay is associated with the charging and discharging of the capacitors of layer 1. Heuristically, based upon the state of layer 1, layer 2 settles to the data required to update level 3. Then layer 3 settles to the next logical state. Then layer 1 settles so that it reflects the current state of layer 3 and the process repeats.

This explanation might suggest a clocked operation. In fact, the layers operate asynchronously. As one layer updates the next, it will have some effect on its own state since the network is no longer "feedforward". However, each layer is "insulated" from its effect on the next layer by a third layer. Each layer has the same time constant. The circuit works in a somewhat clock-like manner due to the delays induced by the dynamics at each layer. These finite delays are essential to the correct operation of this circuit.

Simulations of the circuit indicate that the circuit does settle to the desired steady state. See Figure 9. It seems that passing the update information through a sequence of 3 layers enables the circuit to preserve the total number of image pixels in the image. Although the logic described above could be implemented with only two layers, simulations run with only 2 layers did not converge due to the reasoning above. However, a careful examination of the simulation results indicates that the actual operation of the circuit cannot be approximated by a clocked iterated cellular logic transform. Although the operation does appear to mimic the operation of a CA at the beginning of the transient, toward the middle and end of the transient the pixels do not shift synchronously, even in a single row. See Figure 10.

Since the shifting occurs only at the rightmost end of each horizontal connected component, synchronous shifting is not essential to the proper operation of the circuit.

VI. CONCLUSION

In this paper, we have explored similarities between binary image processing operations using cellular automata and using cellular neural networks. We define a class of CNN cell array layer for which the map from binary input and initial conditions to output is given by a simple closed form expression. Using this expression and layer which executes global propagation, we show that any single iteration operation possible by a CA is also possible on a CNN. We then discuss the extension of these results to multiple iteration CA operations. In particular, we give a necessary and sufficient condition for a binary output vector to have an associated stable state equilibrium point, even in the case of asymmetric coefficients. This condition is expressed in terms similar to that for the analogous condition for CA. The results presented here simplify the design of many templates, as evidenced by the numerous templates presented in Section V. Although the continuous time/state dynamics and lack of external controller make some CA applications difficult to convert to a CNN format, we show by the example of the Radon Transforming CNN, that these results can also be used in the design of CNNs which execute infinitely iterated cellular logic transforms. This paper has only described a small subset of the possible CNNs, and thus only a subset of the possible applications. However, we see that this subset is already quite large. We expect that the continuous time-continuous state dynamics will enable the CNN to be applied to a much richer class of problems than possible by a CA, notably gray level problems.

APPENDIX

A1. Linearly Separable Boolean Functions

For our purposes, we can view a boolean function of n variables, s, as a mapping from $\{0,1\}^n$ to $\{0,1\}$. In other words, s assigns a value of either zero or one to each vector of zeros and ones. If we interpret $\{0,1\}^n$ as a subset of $\mathbb{R}^n$, s is linearly separable if and only if there exists a vector $a \in \mathbb{R}^n$ and a constant $b \in \mathbb{R}$ such that the following two inequalities are satisfied:

$$\langle a,x \rangle \geq b \text{ for } \textit{all } x \in \{ x \in \{0,1\}^n \mid s(x) = 1 \} \tag{A1.1}$$

$$\langle a,x \rangle < b \text{ for } \textit{all } x \in \{ x \in \{0,1\}^n \mid s(x) = 0 \}. \tag{A1.2}$$

Geometrically, a linearly separable boolean function is a function for which all the points which map to 1 lie on or to one side of an $n-1$ dimensional hyperplane in $\mathbb{R}^n$ and all the points which map to -1 lie on the other side of the hyperplane. For example, in two dimensions the function $A + B$ (A or B is true) is linearly separable while the function $A = B$ is not. See Figure A1. In particular, examination of Figure A2 reveals that the boolean function (6) is not linearly separable.

Note that if s is linearly separable, we can always find a vector $\hat{a}$ and a constant $\hat{b}$, such that the inequalities in (A1.1) and (A1.2) are strict. To see this, assume that a and b satisfy (A1.1) and (A1.2). Take $\delta = \min\{ b - \langle a,x \rangle \mid x \in \{0,1\}, s(x) = 1 \}$. This minimum is greater than zero since it is the minimum of a finite set of positive numbers. The vector $\hat{a} = a$ and constant $\hat{b} = b - \frac{1}{2}\delta$ satisfy (A1.1) and (A1.2) with strict inequality. In other words, for any $x \in \{0,1\}^n$, $\langle \hat{a},x \rangle - \hat{b} \neq 0$. Thus, any boolean function which is linearly separable can be implemented in the signum functions of equations (3) and (4).

A2. Proofs of Claims in Section VI

Proof of Claim 1:

First assume that (4) is satisfied for all k,i,j. Define for each k,i,j

$$\hat{v}_{k,i,j} = R_k \cdot \left[\sum_{\gamma,\alpha,\beta}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_k \right] \tag{A2.1}$$

Clearly, $\hat{v} = \{ \hat{v}_{k,i,j} \}$ will be an equilibrium point if $\hat{y}_{k,i,j} = f(\hat{v}_{k,i,j})$ for all k,i,j. Consider the case where $y_{k,i,j} = 1$. Then from (4),

$$(A_{k,k}(0,0) - R_k^{-1})\cdot\hat{y}_{k,i,j} + \sum_{\gamma,\alpha,\beta\neq k,0,0}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_k > 0$$

Adding $R_k^{-1}\cdot\hat{y}_{k,i,j}$ to both sides yields

$$\sum_{\gamma,\alpha,\beta}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot \mathrm{u}_{i+\alpha,j+\beta} + I_k > R_k^{-1} y_{k,i,j}$$

Multiplying by R_k^{-1} and noting that $y_{k,i,j} = 1$, we see that $v_{k,i,j} > 1$, i.e. $f(v_{k,i,j}) = y_{k,i,j}$. The case of $y_{k,i,j} = -1$ is proved similarly. Thus, $f(v_{k,i,j}) = y_{k,i,j}$ is true for all k,i,j and $\hat{v}$ is the associated equilibrium point.

Now take any binary output vector, $\hat{y}$, associated with an equilibrium point v of the CNN. Then the following equation must be satisfied for all k,i,j.

$$0 = -R_k^{-1} v_{k,i,j} + [\sum_{\gamma,\alpha,\beta}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_k]$$

Assume that $\hat{y}_{k,i,j} = 1$. In this case, $v_{k,i,j} \geq 1$, which implies

$$0 \geq -R_k^{-1}\hat{y}_{k,i,j} + \sum_{\gamma,\alpha,\beta}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_k$$

The same statement holds for $y(v_{k,i,j}) = -1$ with the inequality reversed. By assumption the inequality is strict and the combination of the two statements results in (4).

To show asymptotic stability we must show that for each $\hat{v} = \{ \hat{v}_{k,i,j} \}$ as defined in (A2.1), there exists a $\rho > 0$ such that any trajectory starting at some $v_o \in B(\hat{v},\rho) = \{ v \mid |v-\hat{v}| < \rho \}$ approaches $\hat{v}$ asymptotically. For this proof, we define $|v| = \max_{k,i,j} |v_{k,i,j}|$. This definition is not restrictive, since all norms on a finite dimensional Euclidean space are topologically equivalent. Set $\rho = \min_{k,i,j} \{ |\hat{v}_{k,i,j} - \hat{y}_{k,i,j}| \}$. This is greater than zero as it is the minimum over a finite set of numbers which are greater than zero since the quantity in (4) is well defined. Essentially, we have chosen our neighborhood such that all the states are operating in the saturated region of output nonlinearity. Now take any $v_o \in B(\hat{v},\rho)$ and consider the state trajectory starting at v_o. The state of cell $C(k,i,j)$ evolves according to the following ordinary differential equation:

$$C_k \frac{dv_{kij}}{dt} = -R_k^{-1} v_{kij} + \left\{ \sum_{\gamma,\alpha,\beta}\sum\sum A_{k,\gamma}(\alpha,\beta)\cdot y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha}\sum_{\beta} B_k(\alpha,\beta)\cdot u_{i+\alpha,j+\beta} + I_k \right\} \qquad \text{(A2.2)}$$

The quantity in braces is the constant $R_k^{-1}\hat{v}_{kij}$. Thus v_{kij} approaches $\hat{v}_{kij}$ asymptotically for all k,i,j, implying that v approaches $\hat{v}$ asymptotically. To see that the basin of attraction contains the neighborhood in which all the states are operating in the saturation region of the piecewise-linear sigmoid non-linearity, observe that in that case the state equation of each cell is also given by (A2.2).

Proof of Claim 2:

Since (4) is not satisfied for all k,i,j, there exists a set $\{ k_\eta,i_\eta,j_\eta \}_{\eta=1}^{U}$ for which (4) is not satisfied. U is the number of states for which (4) is not satisfied. As long as all the cells of the CNN are operating in the saturation region of the sigmoid non-linearity, the cells of the network evolve according to (A2.2). However, for $k,i,j \in \{ k_\eta,i_\eta,j_\eta \}_{\eta=1}^{U}$, the quantity in braces is less than minus one (resp. greater than one) when $\hat{v}_{kij}$ is greater or equal to one (resp. less than or equal to minus one). Thus, in some finite time at least one of the cells in $k,i,j \in \{ k_\eta,i_\eta,j_\eta \}_{\eta=1}^{U}$ will enter the linear region. It will not immediately return to the saturated region since the quantity in braces is a continuous function of the states of the network and therefore will remain close to its original value. Thus, the cells which do not satisfy (4) tend toward the output state which are given by the signum function in (4). However, due to the continuous time dynamics they do so at varying rates and in the linear region the dynamics become much more complex.

A3. Templates of Example CNNs

1. Hole finder

$$C_1 = R_1 = 1.0 \quad I_1 = -0.1 \quad A_{1,1} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.25 & 1.0 & 0.25 \\ 0.0 & 0.25 & 0.0 \end{bmatrix} \quad B_1 = -1.0$$

$$C_2 = R_2 = 1.0 \quad I_2 = -0.5 \quad A_{2,2} = 1.0 \quad A_{2,1} = -0.5 \quad B_2 = -0.5$$

The initial conditions of layer 1 and 2 are all -1. The image is presented at the input of the array. The imaginary cells outside the boundary of the array should be +1 for layer 1 and -1 for layer 2. The output of layer 2 contains the holes in the image. Layer 1 fills in the background of the image which is connected to the edge of the array. Layer 2 finds the pixels associated with cells whose input and layer 1 outputs are both -1, i.e., the pixels associated with the holes.

2. Object with hole detector

$$C_1 = 10^{-10} \quad R_1 = 1.0 \quad I_1 = -0.1 \quad A_{1,1} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.25 & 1.0 & 0.25 \\ 0.0 & 0.25 & 0.0 \end{bmatrix} \quad B_1 = -1.0$$

$$C_2 = R_2 = 1.0 \quad I_2 = -0.1 \quad A_{2,2} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad A_{2,1} = \begin{bmatrix} 0.0 & -0.2 & 0.0 \\ -0.2 & 0.0 & -0.2 \\ 0.0 & -0.2 & 0.0 \end{bmatrix} \quad B_2 = \begin{bmatrix} 0.0 & -0.2 & 0.0 \\ -0.2 & 2.0 & -0.2 \\ 0.0 & -0.2 & 0.0 \end{bmatrix}$$

Initial conditions of cells in both layers are -1. Image is presented at input. The imaginary cells outside the boundary of the array should be +1 for layer 1, -1 for layer 2. The output of layer 2 contains the objects in the image which contain holes in their interior. Layer 1 performs the same function as in the hole finding CNN, except its settling time is much faster than that of layer 2. Layer 2 propagates a signal of +1 through image pixels associated with images next to holes.

3. Objects larger than 3x3 square detector

$$C_1 = 10^{-10} \quad R_1 = 1.0 \quad I_1 = -0.1 \quad A_{1,1} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.25 & 1.0 & 0.25 \\ 0.0 & 0.25 & 0.0 \end{bmatrix} \quad B_1 = -1.0$$

$$C_2 = R_2 = 1.0 \quad I_2 = -1.2 \quad A_{2,2} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.0 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad A_{2,1} = \begin{bmatrix} -0.025 & -0.25 & -0.25 \\ -0.025 & -3.025 & -0.025 \\ -0.025 & -0.025 & -0.025 \end{bmatrix}$$

$$C_3 = R_3 = 1.0 \quad I_3 = -1.5 \quad A_{3,3} = 1.0 \quad A_{3,2} = 1.0 \quad B_3 = 1.0$$

Initial conditions of all layers of all cells are -1. Image is presented to input. The imaginary cells outside boundary of the array are +1 for layer 1 and -1 otherwise. The output of layer 3 contains the objects in the image which are larger than a 3x3 pixel square. Layer 1 fills in background adjacent to edge. Layer 2 fills in "holes" bigger than a 3x3 pixel square in layer 1 output. Layer 3 outputs those image pixel contained in the regions highlighted by output of layer 2.

4. Connected image component containing specified pixels

$$C_1 = R_1 = 1.0 \quad I_1 = -1.0 \quad A_{1,1} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.4 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad A_{1,2} = -3.0$$

$$C_2 = 10^{-10} \quad R_2 = 1.0 \quad I_2 = -0.1 \quad A_{2,2} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.25 & 1.0 & 0.25 \\ 0.0 & 0.25 & 0.0 \end{bmatrix} \quad B_2 = -1.0$$

$$C_3 = R_3 = 1.0 \quad I_3 = -1.5 \quad A_{3,3} = 1.0 \quad A_{3,1} = 1.0 \quad B_3 = 1.0$$

Initial conditions of layers 2 and 3 are -1. Layer 1 contains the chosen image pixels. Image presented to input of CNN. The imaginary cells outside the boundary of the array should be +1 for layer 2 and -1 otherwise. The output of layer 3 contains objects of image which contain the pixels specified in layer 1. Layer 2 fills in the edge connected background of the image with a faster settling time than the other layers. Layer 1 fills in the holes of layer 2 which contain a specified image pixel. Layer 3 selects those image pixels contained inside the regions highlighted by layer 1.

5. Minimal octagonal convex hull finder

$$C_1 = R_1 = 1.0 \quad I_1 = 1.1 \quad A_{1,1} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.25 & 2.0 & 0.25 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$C_2 = R_2 = 1.0 \quad I_2 = 1.1 \quad A_{2,2} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.25 & 0.0 \end{bmatrix}$$

$$C_3 = R_3 = 1.0 \quad I_3 = 1.1 \quad A_{3,3} = \begin{bmatrix} 0.25 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.25 \end{bmatrix}$$

$$C_4 = R_4 = 1.0 \quad I_4 = 1.1 \quad A_{4,4} = \begin{bmatrix} 0.0 & 0.0 & 0.25 \\ 0.0 & 2.0 & 0.0 \\ 0.25 & 0.0 & 0.0 \end{bmatrix}$$

$$C_5 = R_5 = 1.0 \quad I_5 = -0.85 \quad A_{5,1} = 0.25 \quad A_{5,2} = 0.25 \quad A_{5,3} = 0.25 \quad A_{5,4} = 0.25 \quad A_{5,4} = 1.0$$

Initial conditions of layers 1 through 4 are the image. Initial conditions of layer 5 are -1. The imaginary cells outside boundary of the array should all be -1. The output of layer 5 is the minimum circumscribing convex set whose boundaries are restricted to lie parallel or 45 degrees to the coordinate axes. Layers 1 through 4 propagate a signal from the image in the four directions associated with the octagonal hull. Layer 5 selects those pixels common to all four layers.

References

1. L.O. Chua and L. Yang, , "Cellular Neural Networks: Theory," *IEEE Trans. Circuits and Systems*, vol. CAS-32, Oct. 1988.
2. L.O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Trans. Circuits and Systems*, vol. CAS-32, Oct. 1988.
3. K. Preston, Jr. and M.J.B. Duff, *Modern Cellular Automata: Theory and Applications*, Plenum Press, New York, 1984.
4. M.J.B. Duff and T.J. Fountain, *Cellular Logic Image Processing*, Academic Press, New York , 1986.
5. J.J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci. USA*, vol. 81.
6. T. Matsumoto, L.O. Chua, and H. Suzuki, "CNN Cloning Template: Connected Component Detector," *U.C. Berkeley Electronics Research Laboratory Memorandum*, UCB/ERL M89/65, 25 May 1989..
7. T. Matsumoto, L.O. Chua , and T. Yokohama, "Image Thinning with a Cellular Neural Network," *U.C. Berkeley Electronics Research Laboratory Memorandum*, UCB/ERL M89/86, 19 July 1989.
8. D.E. Rumelhart, J.L. McClelland, and The PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, The MIT Press, Cambridge, MA, 1986.
9. L.O. Chua, C.A. Desoer, and E.S. Kuh, *Linear and Nonlinear Circuits*, McGraw-Hill, New York, 1987.
10. L.O. Chua, *Introduction to Nonlinear Network Theory*, McGraw-Hill, New York, 1969.
11. M. Minsky and S. Papert, *Perceptrons*, The MIT Press, Cambridge, MA, 1969.
12. T. Matsumoto, L.O. Chua, and R. Furukawa, "CNN Cloning Template: Hole-Filler," *U.C. Berkeley Electronics Research Laboratory Memorandum*, UCB/ERL M89/66, 25 May 1989.
13. B.K.P. Horn, *Robot Vision*, The MIT Press, Cambridge MA, 1986.

Figure Captions

1: The piecewise-linear sigmoid function is the only non-linearity in each cell of a CNN. This non-linearity maps the state to the interval [-1,1].

2: (a) The state equation of each cell of the CNN can be implemented with the above circuit consisting only of a capacitor, resistors, and voltage controlled current sources. (b) The interconnections of a CNN are restricted to nearest neighbor couplings. This figure illustrates the interconnections for a cell in layer 2 of a three layer CNN.

3: The function h(v) determines the contribution to the derivative of a linear threshold layer cell's state from the current value of its state.

4: The value of g determines the dynamic routes for a cell in a linear threshold layer. (a) $g < -(A_{1,1}(0,0) - R_1^{-1})$. (b) $|g| < (A_{1,1}(0,0) - R_1^{-1})$. (c) $g > (A_{1,1}(0,0) - R_1^{-1})$.

5: The CNN presented in [2] must be redesigned to detect the corners of a noisy image. (a) The original noise free image. (b) The original image corrupted by Gaussian noise of variance 0.3. (c) The corners detecting using the corner detector of [2] on (a). (d) The corners detected using the corner detector of [2] on (b). (e) The corners detected using the "naively" cascaded CNNs of [2] on (b). (f) The corners detected using the redesigned CNN on (b).

6: The results of this paper have been used to design CNNs which: (a) detect the holes in an image, (b) detect the objects which contain holes, (c) detect objects larger than a 3 by 3 pixel square, (d) detect objects containing specified image pixels, and (e) compute the minimal octagonal convex hull. See appendix for details.

7: (a) For fixed values of ρ_o and θ_o the Radon Transform $\Psi(\rho_o,\theta_o)$ of an image $I(x,y)$ is the line integral of $I(x,y)$ along line L. (b) For fixed θ_o, the function $\Psi(\rho,\theta_o)$ is the projection of the image onto the line $\theta = \theta_o$.

8: The cloning template of the Radon Transforming CNN shows that it is composed solely of linear threshold layers.

9: (a) Simulations of the CNN indicate that the Radon Transforming CNN projects the image intensity onto the vertical axis. (b) To obtain other projections of the Radon Transform, the input image can be rotated. Here the image has been rotated by 90 degrees to obtain the projection onto the horizontal axis.

10: (a) Although the Radon Transforming CNN has been designed using ideas from cellular automata, the CNN does not operate like a CA. Although the pixels shift synchronously at the start of the transient (a), as time progresses, the pixels shift asynchronously (b).

A1: $A + B$ (A or B is equal to 1) is linearly separable, while $A = B$ is not.

A2: The values which map to one and zero by the boolean function (5) are not linearly separable by a two-dimensional hyper-plane in the (l,c,r) variable space.

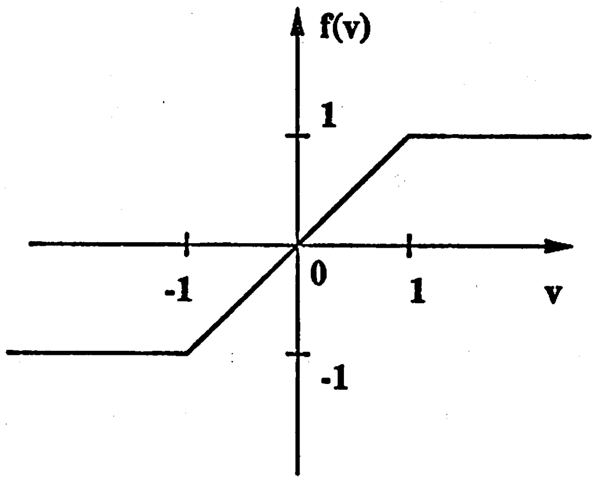


Figure 1

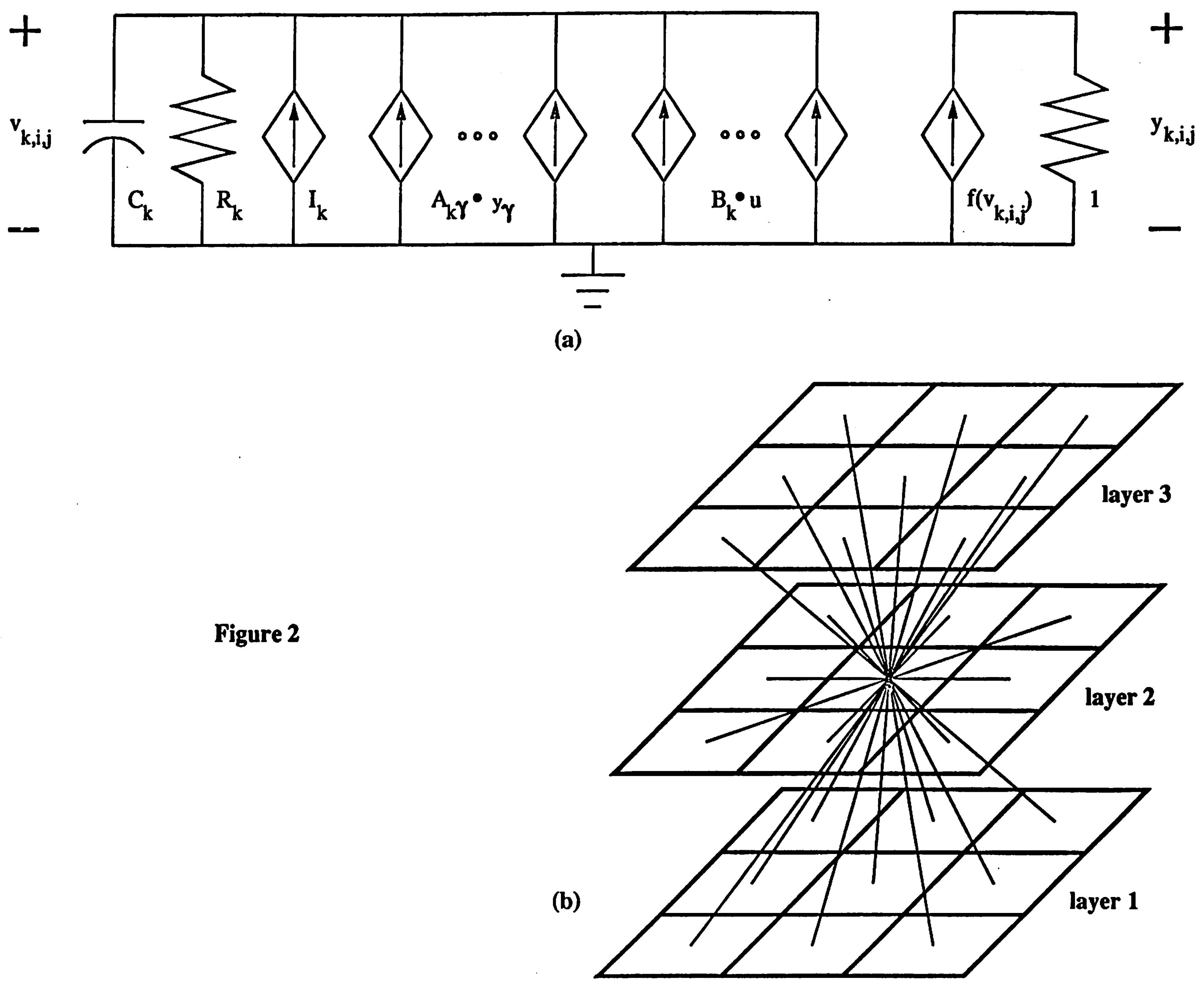

+
$v_{k,i,j}$
–
C_k
R_k
I_k
$A_{k\gamma} \bullet y_\gamma$
$B_k \bullet u$
$f(v_{k,i,j})$
1
+
$y_{k,i,j}$
–
(a)
layer 3
layer 2
layer 1
(b)

Figure 2

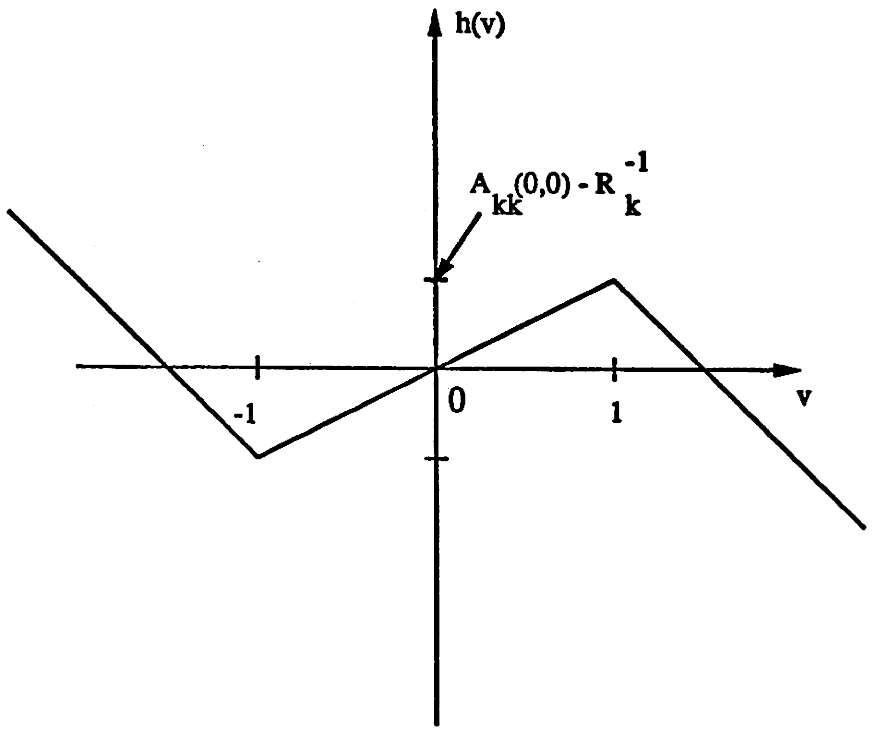


Figure 3

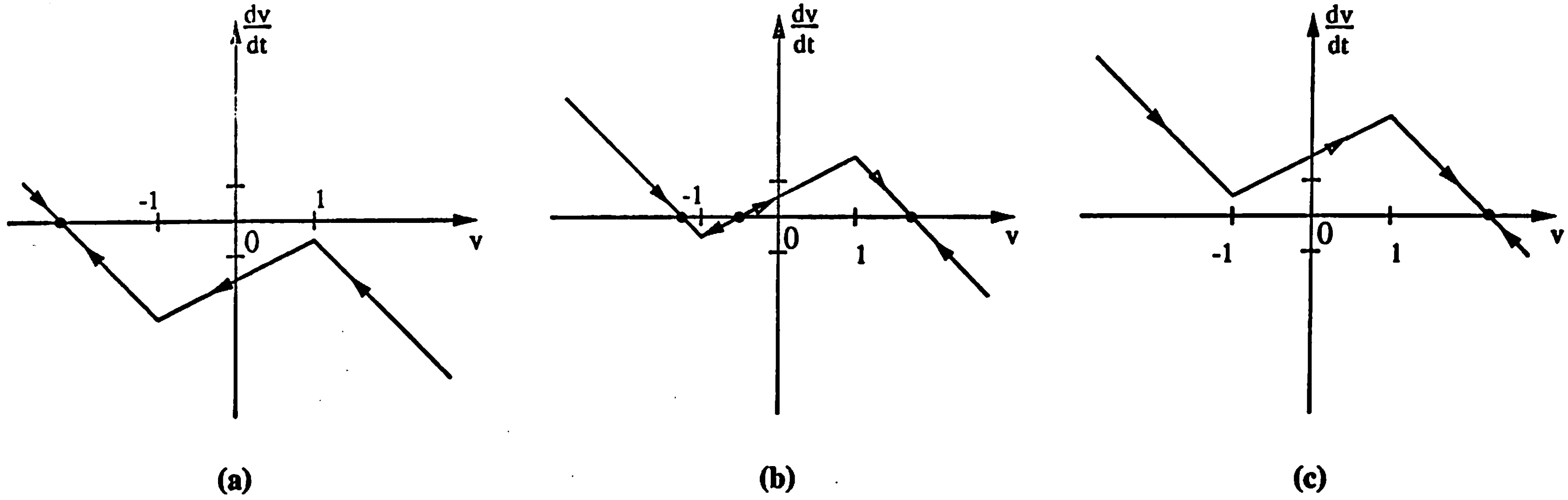


Figure 4

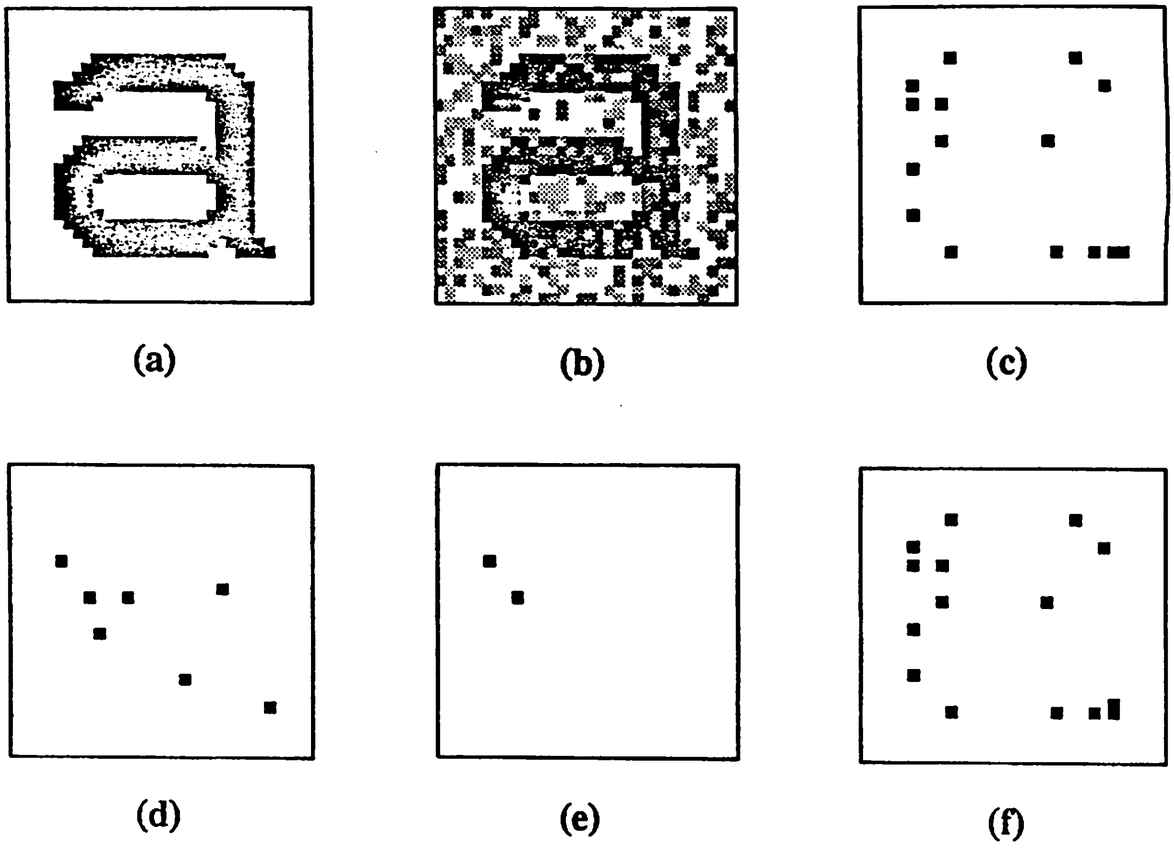


Figure 5

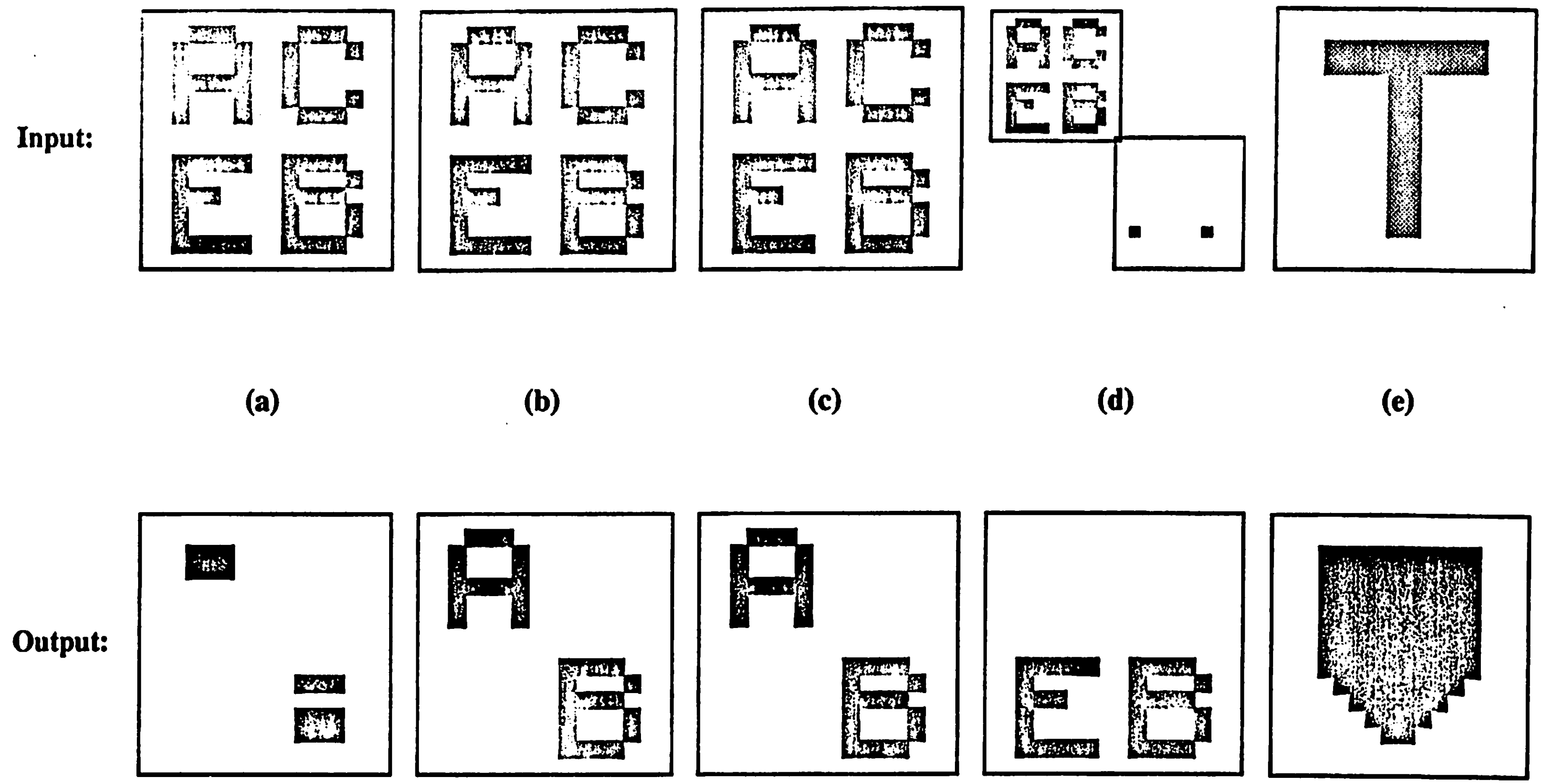
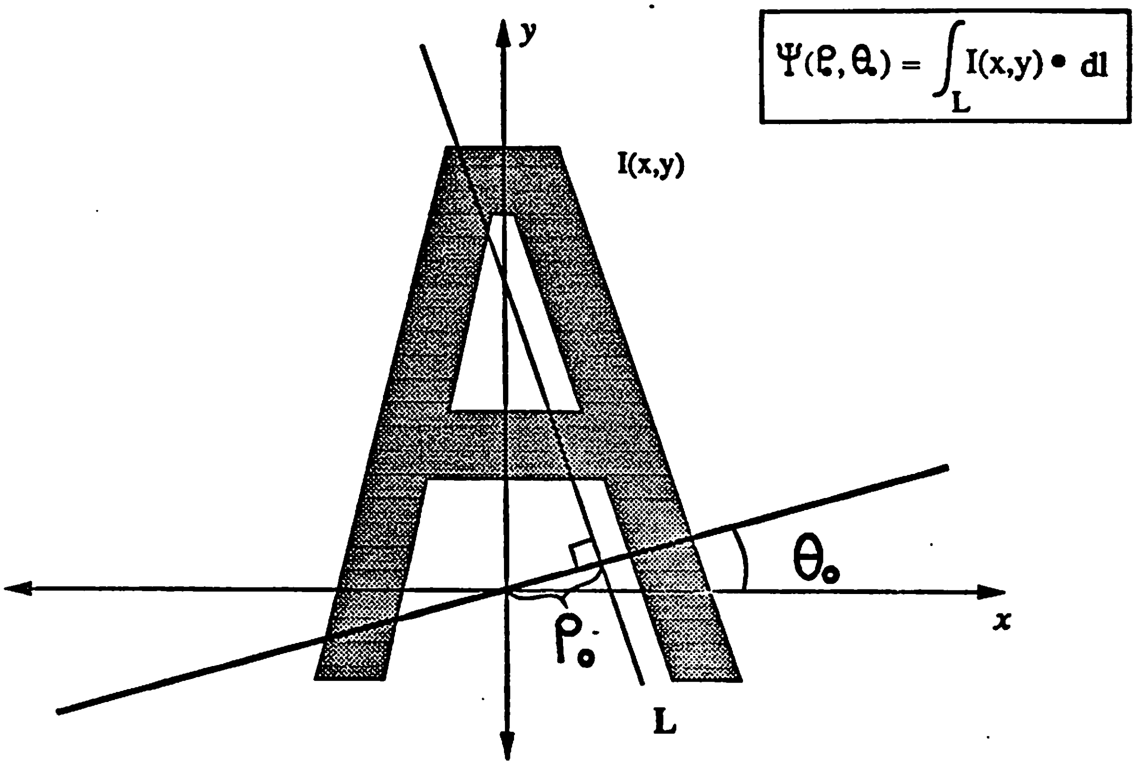
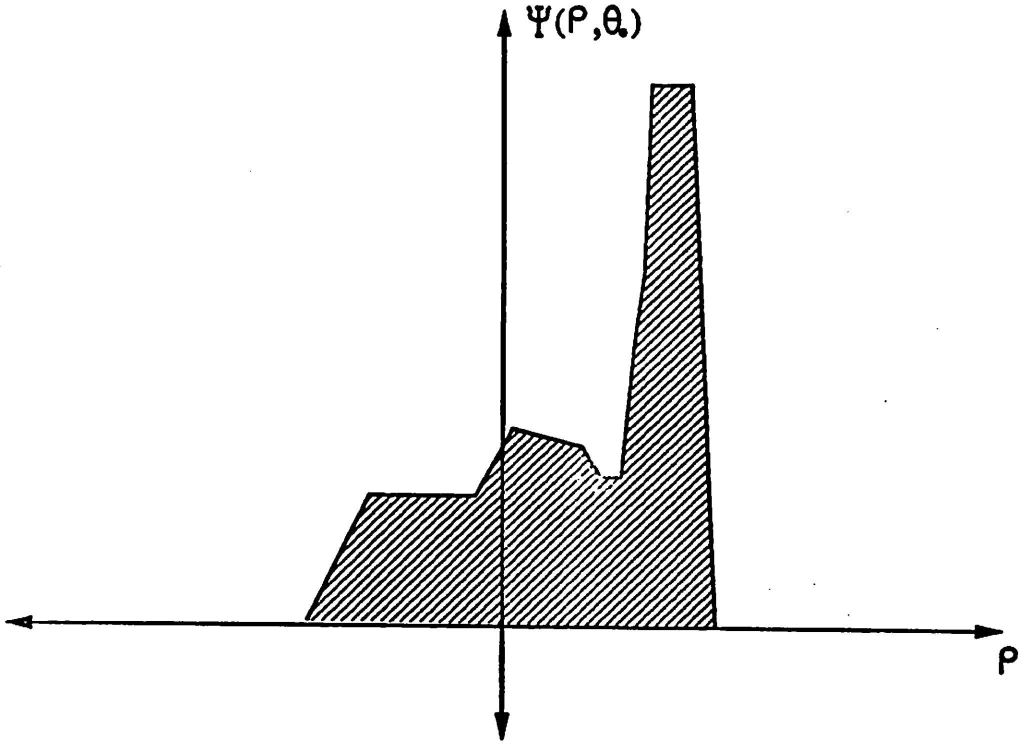


Figure 6


$\Psi(\rho_o, \theta_o) = \int_L I(x,y) \bullet dl$
y
I(x,y)
θ_o
ρ_o
x
L

(a)


$\Psi(\rho, \theta_o)$
ρ

(b)

Figure 7

0.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	0.0

A_{11}

0.0	0.0	0.0
0.0	1.0	-1.0
0.0	0.0	0.0

A_{21}

0.0	0.0	0.0
1.0	-1.0	0.0
0.0	0.0	0.0

A_{32}

0.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	0.0

A_{13}

0.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	0.0

A_{22}

0.0	0.0	0.0
0.0	1.25	0.0
0.0	0.0	0.0

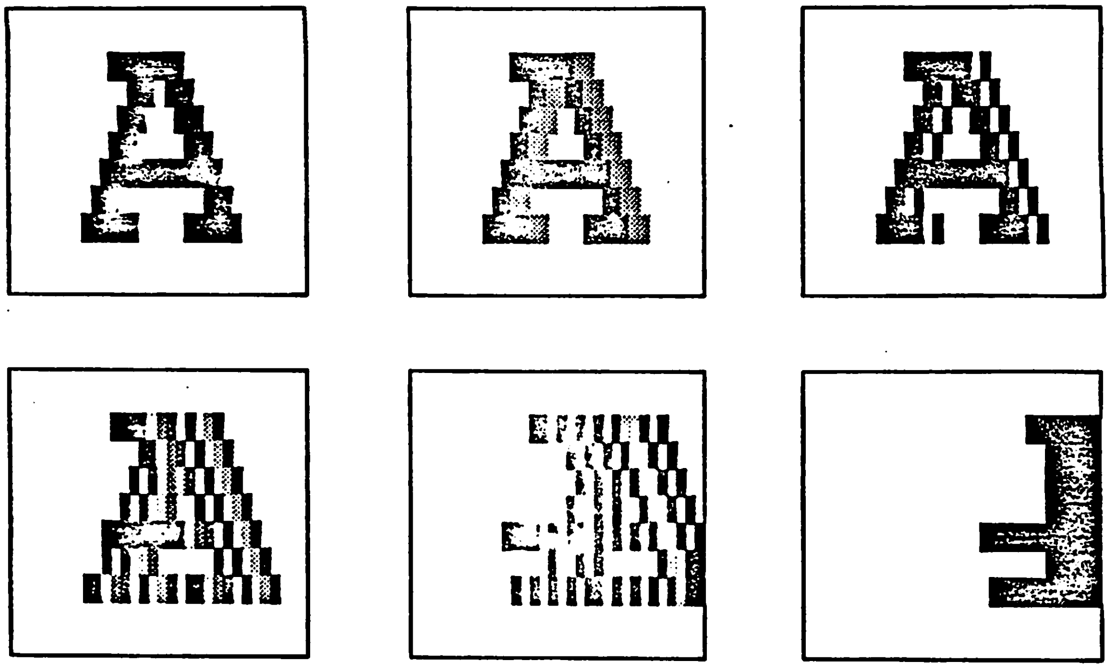
A_{33}

$I_1 = 0.0$ $I_2 = -1.1$ $I_3 = 0.0$

$C_1 = 1.0$ $C_2 = 1.0$ $C_3 = 1.0$

$R_1 = 1.0$ $R_2 = 1.0$ $R_3 = 1.0$

Figure 8


(a)

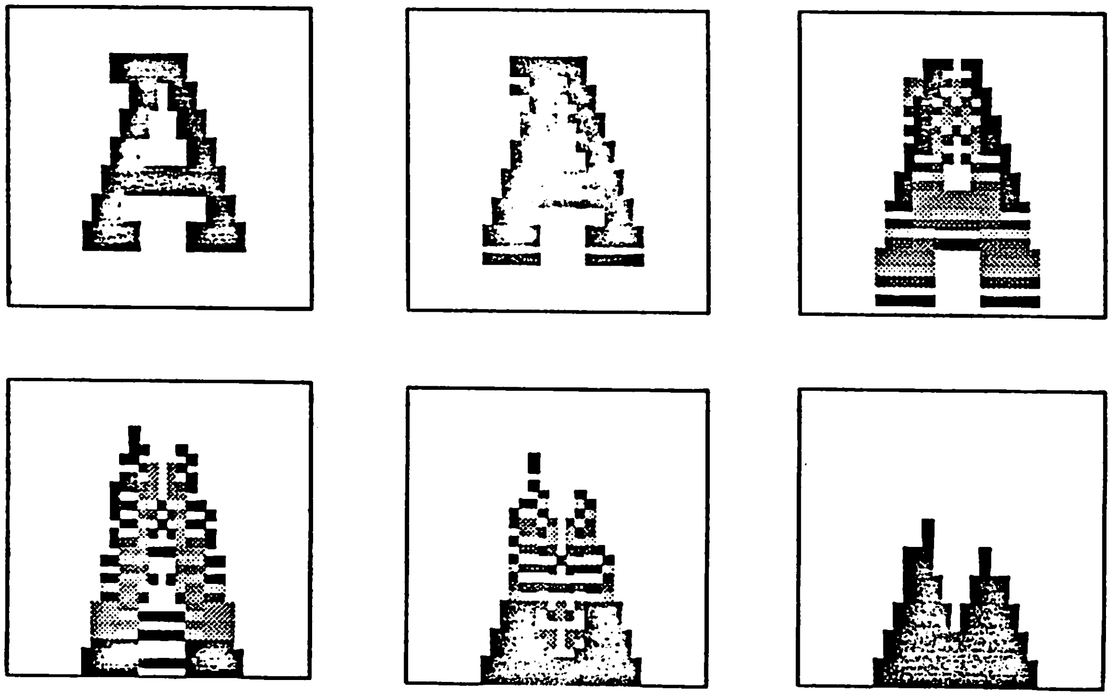
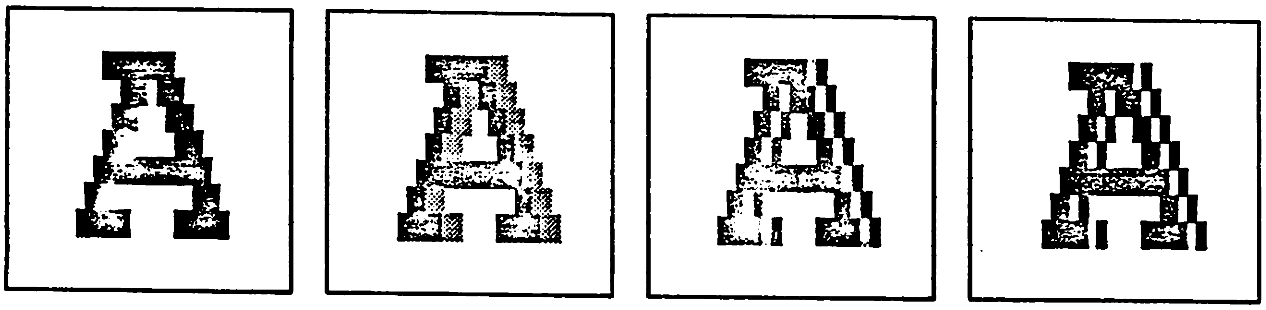
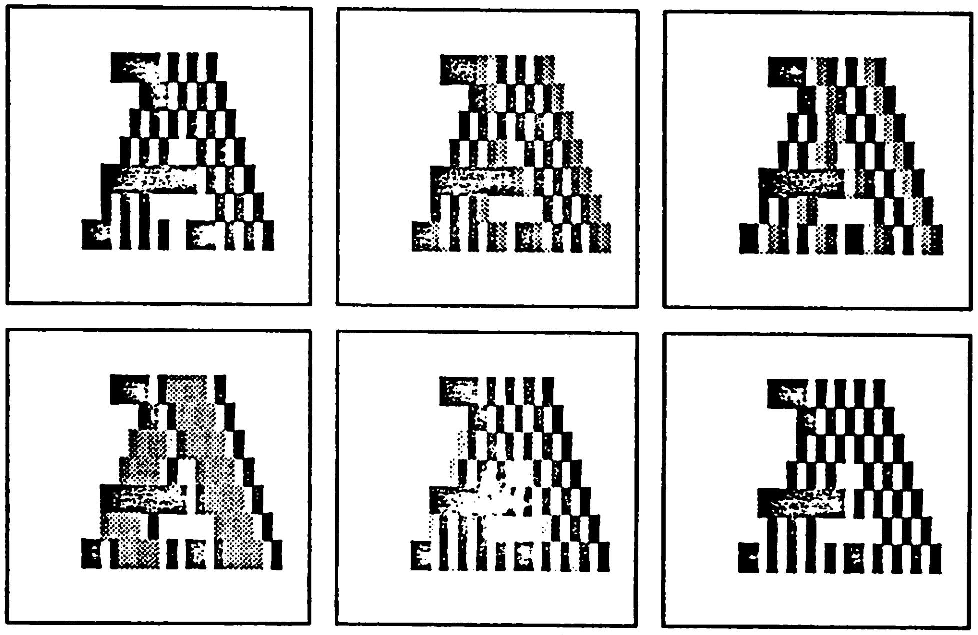

(b)

Figure 9



(a)



(b)

Figure 10

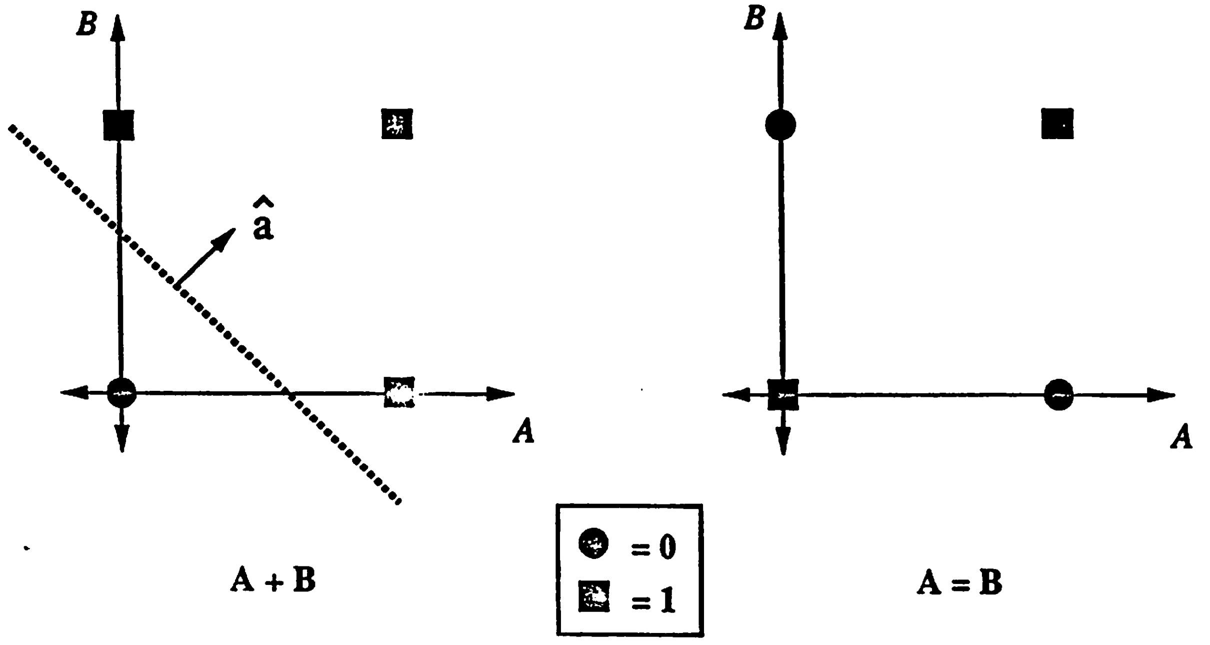


Figure A1

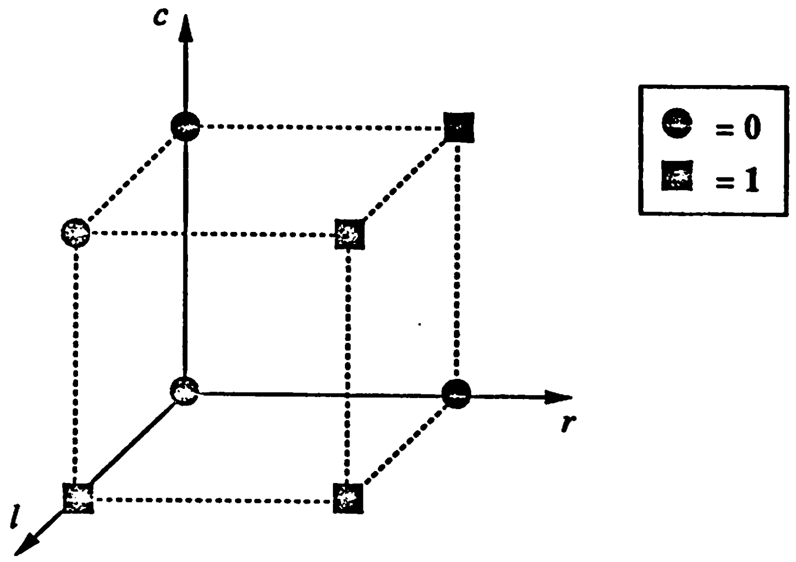


Figure A2