Copyright © 1990, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

MULTIPLE LAYER CELLULAR NEURAL NETWORKS: A TUTORIAL

by

Leon O. Chua and Bertram E. Shi

Memorandum No. UCB/ERL M90/113

3 December 1990

MULTIPLE LAYER CELLULAR NEURAL NETWORKS: A TUTORIAL

by

Leon O. Chua and Bertram E. Shi

Memorandum No. UCB/ERL M90/113

3 December 1990

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Multiple Layer Cellular Neural Networks: A Tutorial [1]

Leon O. Chua Bertram E. Shi [2]

December 3, 1990

[1]This paper will appear in *Algorithms and Parallel VLSI Architectures*, ed. by E. F. Deprettere, Elsevier Science Publishers B.V., Amsterdam.

[2]This work is supported in part by the Office of Naval Research under Grant N000014-89-J-1402 and the National Science Foundation under Grant MIP-86-14000 and by a National Science Foundation Graduate Research Fellowship

Cellular neural networks (CNNs) [1, 2] are a new parallel analog circuit architecture for image processing. Although they discuss the extension of cellular neural networks to include cells with a vector of state variables (multiple layers), the original papers by Chua and Yang introducing CNNs deal mainly with networks composed of a two dimensional layer of cells with a scalar state variable. This tutorial further develops the extension to multiple layers. Starting by studying a simple subclass of single layer CNN, we gain insight into how the continuous time/continuous state dynamics of the CNN perform some basic image processing tasks. This insight aids in the design of several multi-layer CNNs. The design of these CNNs exploits architectural similarities between CNNs and cellular automata (CA), a parallel digital circuit architecture useful in image processing. In fact, any single iteration operation possible on a cellular automata (CA) as described in [3, 4] can be performed with a (possibly multi-layered) CNN [5]. After discussing the relationship with CA, this chapter concludes by describing the operations of several binary image processing CNNs which were designed using the results presented here.

1. INTRODUCTION

Cellular neural networks (CNNs) have been introduced [1, 2] as a parallel analog circuit architecture for image processing. They are composed of a two dimensional array of analog processors or 'cells.' Each pixel in the image plane has a cell associated with it. Each cell contains a set of capacitors, the voltage across which will be referred to as the 'state' of the cell. A 'layer' refers to the two dimensional array of capacitors formed by taking one capacitor from each cell. Thus, a single layer CNN has one capacitor per cell. Each cell also has an input voltage associated with it. The currents through the capacitors in a cell are functions of their voltages, the cell's input voltage and the voltages of the capacitors and inputs of the nearest neighbor cells. Typically, the voltages of some of the capacitors of a cell and/or the input associated with it are initialized to the value of the associated pixel in the image to be processed. The circuit is then allowed to settle. The steady state outputs of the cells represent the result of the 'computation'

performed by the CNN.

Theoretically, due to the analog nature of the processors, CNNs can process gray level images of arbitrary precision. However, in practice uncertainties associated with VLSI fabrication and the input and output of analog voltages limit the precision obtainable. The CNN can also be used for binary image processing as it can be designed to ensure that the steady state output of each cell is either ± 1. The key advantages of the analog approach are asynchronous fast operation and a small cell size for VLSI design. Currently, work is being done in fabricating CNN chips [6].

In this chapter, we restrict the discussion to the implementation of binary image processing operations. Our results explain how the dynamics of the cells perform the processing of some of the examples presented in [2]. These results provide a foundation for the design of more complex applications by exploiting the possibility of using multiple layer CNNs. To simplify the discussion of the basic results, we begin by discussing single layer cellular neural networks before extending our results to the multiple layer networks. Section 2 describes single layer cellular neural networks as introduced in [1]. Section 3 discusses some dynamical properties of these networks. Section 4 extends the previous results to multiple layer neural networks. Section 5 introduces cellular automata and shows how to use the results of Sections 3 and 4 to map certain cellular automata operations onto a CNN architecture. This mapping motivates the design of the CNNs of the next section. Section 6 provides numerous design examples before Section 7 summarizes the main results of this chapter.

Many of the ideas presented here have been presented in other contexts associated with artificial neural networks, and will be familiar to those readers familiar the neural network literature. We will identify these as they arise. However, since this chapter is meant to be of a tutorial nature, our treatment assumes no prior knowledge of neural networks.

2. SINGLE LAYER CELLULAR NEURAL NETWORKS

Since each pixel in the image plane has a cell associated with it, an L layer cellular neural network designed to process an M by N pixel image is composed of a two dimensional M by N array of cells. Each cell of this CNN contains L capacitors and the state is an L-dimensional vector. Thus, each cell of a single layer CNN contains a single capacitor and the state is a scalar variable. The rest of this section assumes a single layer CNN. The extension to multiple layers will be made explicit in Section 4.

Each cell of a single layer CNN will be denoted $C(i,j)$ where $1 \leq i \leq M$ and $1 \leq j \leq N$. The output $y_{i,j}$ of $C(i,j)$ is a piecewise-linear function of its state $v_{i,j}$:

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

This function restricts the output of each cell to lie in the interval $[-1, 1]$. See Figure 1. The *state of the CNN*, $\mathbf{v}$, is defined to be the vector of the states of all the cells in the network. Similarly, the *output of the CNN*, $\mathbf{y}$, is defined to be the vector of the output of the cells. Each cell also has an associated input voltage. The *input to the CNN*, $\mathbf{u}$, is defined to be the vector of the inputs to the cells. For binary image processing, $+1$ values of input and output are associated with image pixels and -1 values with background pixels. When discussing boolean functions implemented

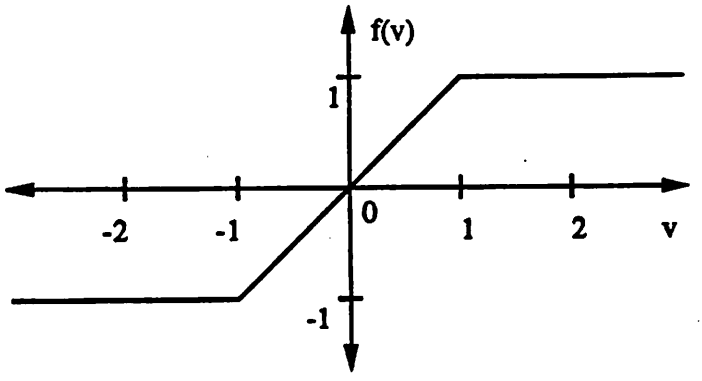


Figure 1: The piecewise-linear sigmoid function is the only non-linearity in each cell of a CNN. This non-linearity maps the state to the interval [-1,1].

with CNNs, $+1$ is associated with the boolean value 'true' and -1 with 'false'.

The dynamical behavior of the CNN which determines the processing performed is provided by allowing the current entering the capacitors of the cells to vary with the state of the array. We impose two restrictions on this interaction. First, the current entering a cell's capacitor is an affine function of the cell's state, its input and output, as well as the input and output associated with its nearest neighbors. The restriction to affine functions allows us to implement the current using only linear voltage controlled current sources. The 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 two dimensional network, such as the Hopfield neural network [7], is practically impossible to build with today's VLSI technology.

To clarify the term *nearest*, define the distance between $C(i,j)$ and $C(m,n)$ by

$$d(i,j;m,n) = \max(|m-i|, |n-j|).$$

Using this metric, the *r-neighborhood* of cell $C(i,j)$ is defined as

$$N_r(i,j) = \{C(m,n) | d(i,j;m,n) \leq r; 1 \leq m \leq M; 1 \leq n \leq N\}.$$

For example, the 1-neighborhood of $C(i,j)$ is a three by three square of nine cells centered at $C(i,j)$. Define r_0 to be the minimum r such that for all $i,j \in \{1,\ldots M; 1,\ldots N\}$, the cells whose outputs affect the current through the capacitor of $C(i,j)$ are in the r_0 neighborhood of $C(i,j)$. The restriction to nearest neighbor interconnections requires r_0 to be much smaller than both M and N. For many applications, r_0 will be 1 or 2.

Second, since image processing operations should often be invariant under translation of the image, the interaction between each cell and its nearest neighbors must be uniform over the entire array. Thus, the function determining the current through the capacitor of one cell of the CNN uniquely determines the functions determining the currents through the capacitors of all the cells in the entire array. Of course, the cells on the boundary of the array must be treated separately. We discuss how to deal with these cells at the end of this section.

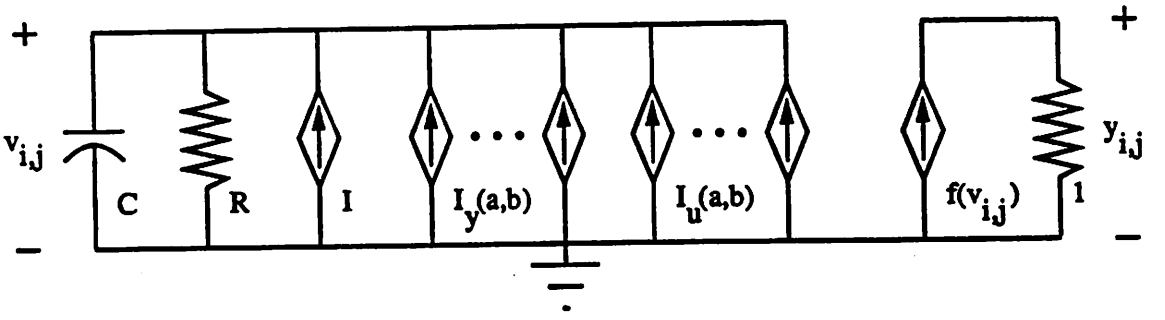


Figure 2: The state equation of each cell of a single layer CNN can be implemented with the above circuit consisting only of a capacitor, resistors, and voltage controlled current sources. $I_y(a,b) = A(a,b)y_{i+a,j+b}$ and $I_u(a,b) = B(a,b)u_{i+a,j+b}$.

Chua and Yang [1] derived the state equation of each cell of the CNN as the dynamical equation governing the circuit shown in Figure 2. Each capacitor is shunted to ground by a resistance R. It is also driven by a bias current and linear voltage controlled current sources which are controlled by the inputs and outputs of the cell and its nearest neighbors. The piecewise-linear function mapping the state of each cell to its output is the only non-linearity in the cell. Making the local interconnections and translation invariance explicit, the state equation of of $C(i,j)$ presented in [1] reduces to:

$$C\frac{dv_{i,j}}{dt} = -\frac{v_{i,j}}{R} + \sum_{\alpha=-r_0}^{r_0}\sum_{\beta=-r_0}^{r_0} A(\alpha,\beta)y_{i+\alpha,j+\beta} + \sum_{\alpha=-r_0}^{r_0}\sum_{\beta=-r_0}^{r_0} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I \qquad (1)$$

where r_0 is defined above and C and R are positive constants. The coefficient $A(\alpha,\beta)$ is the gain of the voltage controlled current source linking the output of $C(i+\alpha,j+\beta)$ to the current entering $C(i,j)$. Note that it is independent of i and j. Similarly $B(\alpha,\beta)$, the gain of the voltage controlled current source linking the input voltage associated with $C(i+\alpha,j+\beta)$ with the current entering $C(i,j)$, is also independent of i and j. I is a bias current applied uniformly to each cell.

Since this state equation is an affine function, it is uniquely specified by the coefficients of the affine relation. These coefficients are defined to be the CNN's *cloning template*. Thus, the values of $A(\cdot,\cdot)$ $B(\cdot,\cdot)$, I, C and R are 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 $A(\cdot,\cdot)$ and $B(\cdot,\cdot)$ as $2r_0+1$ by $2r_0+1$ matrices where the center element corresponds to the coefficient which weights the effect of the cell's own output or input upon its state's derivative. See (8) for an example with $r_0 = 1$.

Those familiar with the neural network literature will recognize the above equation as being similar to that governing the dynamics of the Hopfield neural networks. However, the key differences are that we restrict ourselves to local interconnections and use a piecewise linear sigmoid nonlinearity. This non-linearity, in conjunction with the self feedback term from the cell's output to its input current, allows for ± 1 steady state outputs without the assumption of infinite gain in the sigmoid non-linearity. Chua and Yang show show that the condition $A(0,0) > R^{-1}$ ensures that each cell has a ± 1 steady state output. Since we discuss exclusively

binary image processing applications, unless specified otherwise we shall assume this condition holds.

We made the assumption that the nearest neighbor interaction of each cell is uniform over the entire array. In other words, the relationship between the derivative of each cell's state and outputs of the cell's neighbors is the same for every cell. However, since in practice the CNN array has a finite size, there is a problem with cells which are close to the boundary of the array. These cells may not have the full complement of nearest neighbors that cells in the interior of the array have. Thus, the dynamics of the cells cannot be uniform over the array. Improperly set boundary conditions may lead to strange effects propagating in from the edges of the array. In the following, the cells which do not have the full complement of nearest neighbors are referred to as edge cells.

To compensate for the effect of finite array size, we can imagine that there are cells outside of the array which complete the neighborhood of each edge cell. There are various choices for the output states of these imaginary boundary cells. In the hole finding CNN discussed below, the imaginary boundary cells output a constant $+1$ voltage. This boundary condition provides a 'source' which drives the output voltages of the edge cells to $+1$. These cells in turn drive the voltages of their neighbors to $+1$, and so on. This creates a 'wave' of cells with $+1$ output voltage propagating into the array. The hole finding CNN stops this propagation at the outside edges of the image objects with an appropriate choice of B. On the other hand, the boundary cells of a layer can also be set to output a constant -1. In this case, the edge cells always see the imaginary boundary cells as part of the background. Although there are many other possible boundary conditions, only these two boundary conditions are used in the examples of this chapter.

From the standpoint of VLSI implementation, the imaginary boundary cells do not add to the number of cells which must be put on the chip. Instead of actually building boundary cells outside of the CNN array to input to the edge cells, the boundary cell effects can be incorporated into the dynamics of each edge cell. In fact, for the cases mentioned above, the change can be made by simply altering the bias current of each edge cell.

3. DYNAMICAL PROPERTIES OF SINGLE LAYER CELLULAR NEURAL NETWORKS

This section begins with a discussion of a necessary and sufficient condition for an output vector to have an associated stable equilibrium point. Due to the local nature of the interconnections, this condition is expressed as an implicit equation of a cell's input and output and the inputs and outputs of its nearest neighbors which must be satisfied at each cell. Under certain circumstances, the right hand side of this equation is the expression for the map from input and initial conditions to the steady state output.

A CNN has *symmetric coefficients* if and only if $A(\alpha,\beta) = A(-\alpha,-\beta)$. In this case, the dynamics of the CNN operate to minimize the following energy functional:

$$E = \sum_{i,j} y_{i,j} \left(-\frac{1}{2} \sum_{\alpha,\beta} A(\alpha,\beta) y_{i+\alpha,j+\beta} + \frac{1}{2R} y_{i,j} - \sum_{\alpha,\beta} B(\alpha,\beta) u_{i+\alpha,j+\beta} - I \right)$$

subject to the constraint that $|y_{i,j}| \leq 1$ for all i, j. The existence of this functional ensures the stability of all CNNs of this type as well as a means of characterizing the dynamics of the CNN as a gradient descent system. In the case of asymmetric coefficients, we have no such energy functional. However, recently a connected component detecting CNN [8] and a image thinning CNN [9] have been developed using asymmetric coefficients. Not all CNNs with asymmetric coefficients will be stable, but Chua and Roska have recently discovered some stability results for CNNs with asymmetric coefficients [10].

It can easily be shown that any minimum $\{\hat{y}_{i,j}\}$ of the above energy functional such that $\hat{y}_{i,j} = \pm 1$ for all i, j must obey the following condition: For all i, j

$$\hat{y}_{i,j} = \text{sgn}\left[(A(0,0) - \frac{1}{R})\hat{y}_{i,j} + \sum_{\alpha,\beta \neq 0,0} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I\right] \quad (2)$$

In fact, this condition is a necessary and sufficient condition for an output vector to be a stable equilibrium point of the output dynamics, even in the case of asymmetric coefficients.

Claim 1 *Assume the right hand side of (2) is well defined (i.e., the quantity inside the signum is never zero) for all binary* (± 1) *combinations of the input, u, and output, y, variables. A binary output state,* $\hat{y} = \{\hat{y}_{i,j}\}$ *satisfies the implicit equation (2) for all* i, j*, if and only if there exists an equilibrium point* $\hat{v} = \{\hat{v}_{i,j}\}$ *of the CNN which maps to* $\hat{y}$ *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 appropriate saturated region of the piecewise-linear output non-linearity.*

The proof of this claim is contained in the appendix. Essentially, Claim 1 states that if a CNN's output satisfies (2) for all i, j, the output of the CNN will no longer change and the state converges to a stable equilibrium point, and vice versa. Due to the uniform local nature of the interactions between the cells of a CNN, this necessary and sufficient condition is a single local condition which must be satisfied for all cells in the array.

In general, even if there exists a binary output vector which satisfies (2), a CNN may not be stable for all initial conditions nor can the state trajectories be easily predicted, short of simulating them. However, there exists a class of single layer CNN which is stable regardless of the symmetry of the coefficients. In addition, for this class there is a simple explicit expression for the map from input and initial conditions to steady state output.

Definition 1 *A single layer CNN is said to be in the* linear threshold class *if and only if*

$$A(\alpha,\beta) = 0 \qquad \forall (\alpha,\beta) \neq (0,0).$$

This definition prohibits any interconnections between the cells of a linear threshold single layer CNN, thereby considerably simplifying its dynamics. The state trajectories of the cells depend only upon their own states and the external input. The corner detecting and edge detecting CNNs of [2] are both in the linear threshold class.

For a linear threshold single layer CNN, (2) reduces to:

$$\hat{y}_{i,j} = \text{sgn}\left[(A(0,0) - \frac{1}{R})\hat{y}_{i,j} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I\right] \quad (3)$$

Analysis similar to that in [1] shows that assuming the input is constant, the right hand side

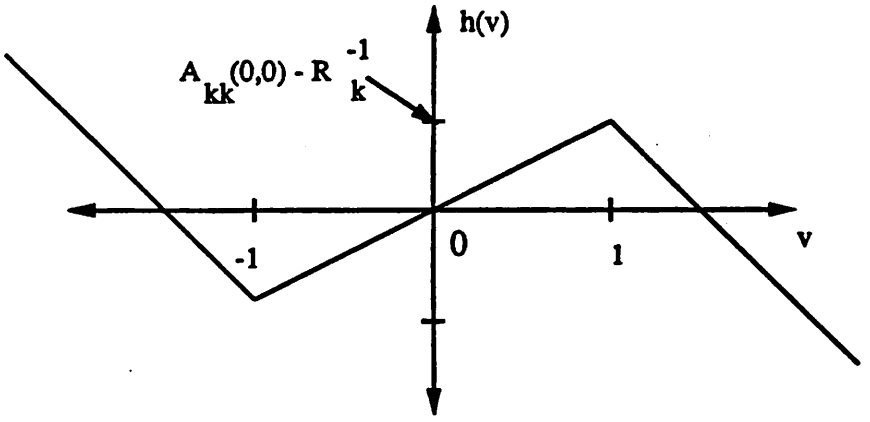


Figure 3: The function $h(v)$ determines the contribution from the state of a linear threshold single layer CNN to its derivative.

of this equation is exactly the map from the input and initial conditions to steady state output. Partitioning the right hand side of (1) into a part, $h(v_{i,j})$, which depends upon $v_{i,j}$ and a part, g, which does not, the state equation of $v_{i,j}$ can be rewritten as

$$C\frac{dv_{i,j}}{dt} = h(v_{i,j}) + g \tag{4}$$

where

$$h(v_{i,j}) = -\frac{v_{i,j}}{R} + A(0,0)f(v_{i,j})$$

and

$$g = \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I$$

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

In this case, g is constant. Figure 4 depicts the dynamic routes [11, 12] for three characteristic g values. If $|g| > A(0,0) - R^{-1} > 0$, 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. If g is positive, then the output of the cell is $+1$. In addition, since $A(0,0) - R^{-1} > 0$ by assumption, $(A(0,0)-R^{-1})+g > -(A(0,0)-R^{-1})+g > 0$. On the other hand, if g is negative, then the output of the cell is -1 and $-(A(0,0) - R^{-1}) + g < (A(0,0) - R^{-1}) + g < 0$. If $|g| < A(0,0)-R^{-1}$, then there exist two stable equilibrium points and one unstable equilibrium point. If $v_{i,j} < \frac{-g}{A(0,0)-R^{-1}}$, then the cell will settle to the stable equilibrium point which maps to -1. If $v_{i,j} > \frac{-g}{A(0,0)-R^{-1}}$, then the cell will settle to the stable equilibrium point which maps to 1. If $v_{i,j} = \frac{-g}{A(0,0)-R^{-1}}$, then the initial condition lies exactly on the unstable equilibrium point and theoretically, the output is equal to the initial condition for all time. However, in practice, thermal noise will cause the the cell to eventually settle at one of the stable equilibrium points, although we cannot tell which in advance. The case of $|g| = A(0,0) - R^{-1}$ is also somewhat problematic. In this case, there are two equilibrium points. One is stable while the other is unstable. Theoretically, this case is identical to the case where $|g| < A(0,0) - R^{-1}$.

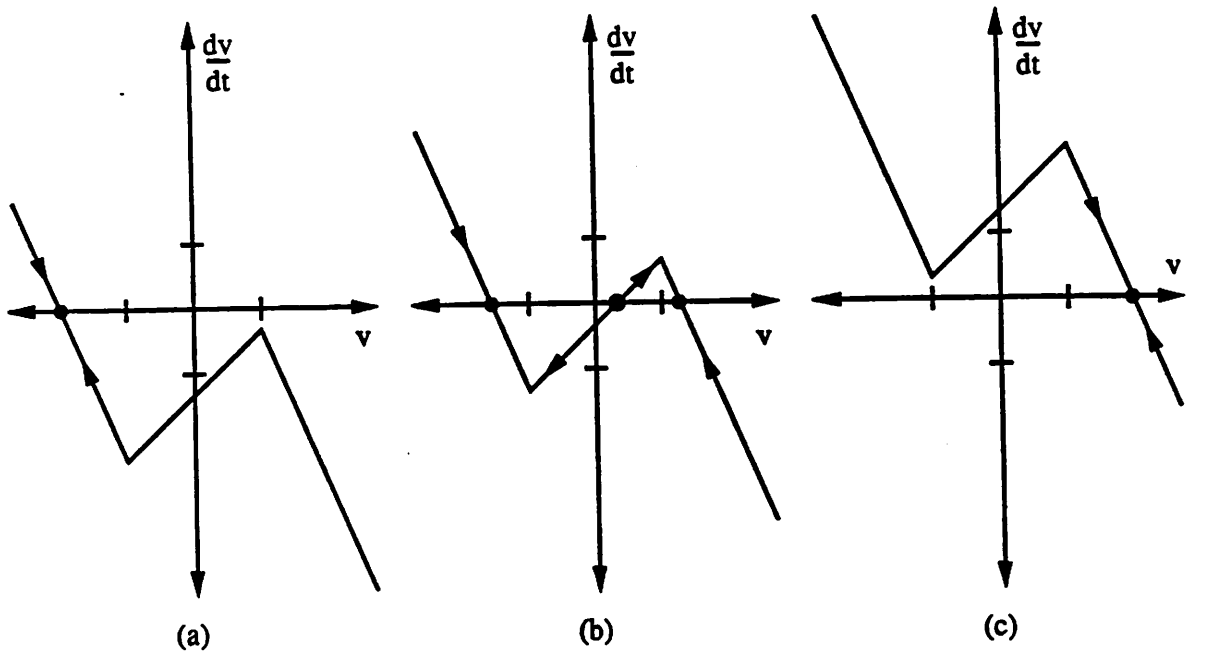


Figure 4: The value of g determines the dynamic routes for a cell in the linear threshold layer. (a) $g < -(A(0,0) - R_1^{-1})$. (b) $|g| < (A(0,0) - R_1^{-1})$. (c) $g > (A(0,0) - R_1^{-1})$.

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 in binary image processing, these two cases can be avoided. Ignoring these last two cases, the following equation summarizes the above results

$$\lim_{t\to\infty} y_{i,j}(t) = \text{sgn}\left[(A(0,0) - R^{-1})y_{i,j}(0) + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I\right] \tag{5}$$

where $|y_{i,j}| \leq 1$ and $y_{i,j}(0) = v_{i,j}(0)$ if $|y_{i,j}| < 1$. The two problematic cases correspond to situations in which the quantity inside the signum function is exactly zero and thus the right hand side is undefined. Assuming $g \neq 0$, the analysis above also holds even if $A(0,0) = R^{-1}$. In this case, the $y_{i,j}(0)$ term drops out and the steady state output of the cell depends solely on the external input,

The left hand side of (5) is a signum function of an affine combination of the input and the initial condition of the output. This type of function is commonly referred to as a linear threshold function. Any boolean function which can be expressed in this form must be linearly separable by a hyperplane in the variable space. Conversely, any boolean function *of the input* 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). See appendix. This fact implies that *any linearly separable boolean function of the input can be implemented using a CNN composed of a single linear threshold layer.* However, we cannot implement all linearly separable boolean functions of the input and initial condition of the state as the coefficient $A(0,0) - R^{-1}$ is constrained to be greater than or equal to zero.

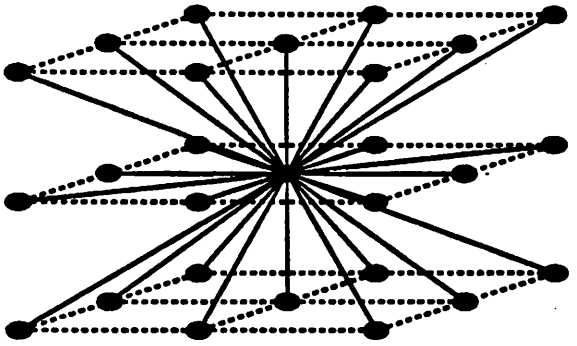


Figure 5: The interconnections of a CNN are restricted to nearest neighbor couplings. This figure illustrates the interconnections for a cell in layer two of a three layer CNN.

In fact, the map (5) is exactly the same as the input/output map of the single layer perceptron [13]. The limitations of the single layer perceptron were extensively studied in Minsky and Papert's *Perceptrons* [14]. However, the continuous time dynamics and multiple layer capability of CNNs extend the possible applications of CNNs beyond those of simple perceptrons. In addition, this type of single layer CNN is only a small subset of all possible single layer CNNs.

4. THE EXTENSION TO MULTIPLE LAYER CELLULAR NEURAL NETWORKS

As stated above, each cell of an L layer cellular neural network contains L capacitors and has an associated L dimensional state vector. The output of the cell is also an L dimensional vector where each component is the previously given piecewise-linear function of the corresponding component of the state. The input can also be a vector, although for most applications so far it has been a scalar. Thus, the coefficients C and R are L by L diagonal matrices of positive coefficients. Each $A(\alpha,\beta)$ is also an L by L matrix, although not necessarily diagonal. $B(\alpha,\beta)$ and I are L dimensional vectors. The treatment of the edge cells is completely analogous to the single layer case, except that the imaginary boundary cells have vector valued output where each element can be specified independently. This approach to multiple layer CNNs taken in [2].

A completely equivalent way to view a multiple layer CNN is as an L by M by N array of cells with scalar state variables. We adopt the latter interpretation as it simplifies the discussion and is intuitively appealing as it emphasizes the grouping of the state variables into L layers of M by N cells. Figure 5 illustrates the connectivity pattern of a single cell in layer 2 of a three layer network with $r_0 = 1$. Often each layer will perform a different processing task. In this interpretation, imaginary boundary cells with scalar valued output are associated with each layer. When dealing with multiple layer CNNs, cells and their states and outputs will be denoted by $C_k(i,j)$, $v_{k,i,j}$ and $y_{k,i,j}$, where $k \in \{1,\ldots L\}$ is the layer number. We will generally drop the subscript k for simplicity when dealing with single layer CNNs.

$A_{k,l}(\alpha,\beta)$ denotes the k,l-th element of the matrix $A(\alpha,\beta)$ which is the gain of the voltage controlled current source linking the output of $C_l(i+\alpha,j+\beta)$ to $C_k(i,j)$. C_k, R_k, and I_k are the capacitance, resistance and bias current associated with each layer. $B_k(\alpha,\beta)$ denotes the

k-th element of the vector $B(\alpha,\beta)$, which is the gain of the voltage controlled current source linking the input associated with $C(i+\alpha,j+\beta)$ to $C_k(i,j)$. Using the new notation, the state equation governing the state of $C_k(i,j)$ is:

$$C_k\frac{dv_{k,i,j}}{dt} = -\frac{v_{k,i,j}}{R_k} + \sum_{\gamma=1}^{L}\sum_{\alpha,\beta} A_{k,\gamma}(\alpha,\beta)y_{\gamma,i+\alpha,j+\beta} + \sum_{\alpha,\beta} B_k(\alpha,\beta)u_{i+\alpha,j+\beta} + I_k \qquad (6)$$

This equation differs from that presented in [1] in that layers of higher index than k can affect the dynamics of layer k. To ensure stability for multiple layer CNNs, Chua and Yang assumed that layer k was connected only to the output of lower layers and that the coefficients were symmetric within each layer. We change the definition to encompass the Radon Transforming CNN presented here and the image thinning CNN.

With this new notation, the previous results extend trivially to the multiple layer case. Claim 1 can be applied to multiple layer CNNs by replacing (2) with (7),

$$\begin{aligned}\hat{y}_{k,i,j} &= \mathrm{sgn}[(A_{k,k}(0,0) - \frac{1}{R_k})\hat{y}_{k,i,j} + \sum_{\gamma,\alpha,\beta\neq k,0,0} A_{k,\gamma}(\alpha,\beta)\hat{y}_{\gamma,i+\alpha,j+\beta} \\ &\quad + \sum_{\alpha,\beta} B_k(\alpha,\beta)u_{i+\alpha,j+\beta} + I_k], \end{aligned} \qquad (7)$$

for all k,i,j.

Although a multiple layer CNN cannot be in the linear threshold class, any layer of a multiple layer CNN can be.

Definition 2 *The k-th layer of a multiple layer CNN is in the* linear threshold class *if and only if*

$$A_{k,k}(\alpha,\beta) = 0 \qquad \forall(\alpha,\beta) \neq (0,0).$$

This definition reduces to the previous definition in the single layer case.

Using multiple layer *feedforward* CNNs constructed of linear threshold layers removes the restriction to linearly separable boolean functions encountered by linear threshold single layer CNNs.

Definition 3 *A CNN is* feedforward *if and only if $A_{k,l} = 0$ for all $k < l$.*

All multiple layer CNNs defined in [1] are feedforward.

Assume all the layers of a feedforward CNN with constant input are of the linear threshold class such that $A_{k,k}(0,0) = R_k^{-1}$ for all $k > 1$. The output of layer 1 settles to a constant given by (5) since the input is constant. Once layer 1 has settled, by the feedforward assumption the input to layer 2 is constant. Any initial conditions setup by the initial conditions of layer 2 and the effect of the transient response of layer 1 are irrelevant to the final output of layer 2 since $A_{2,2}(0,0) = R_2^{-1}$ by assumption. Iterating this argument for each successive layer, we see that the steady state of each layer depends only upon the steady state outputs of the previous layers and the input. Thus, the outputs of all the cells of the CNN depend only upon the input to the CNN and the initial condition of layer 1. Stability is guaranteed since the definitions of a feedforward CNN and a linear threshold layer rule out any feedback paths between the cells.

If in addition $A_{1,1} = R_1^{-1}$, this type of CNN is equivalent to a two dimensional array of feedforward hidden layer neural networks [15] operating on the input to the CNN. A feedforward

hidden layer neural network with enough layers can implement any boolean function of its inputs. Any boolean function of n variables can be decomposed into a sum of products of the variables and their complements. The sums correspond to the boolean relation 'or' and the products correspond to the boolean relation 'and'. It can easily be verified that each of the products can be implemented using a single perceptron. The output of all of these perceptrons can be input to another perceptron which performs the summation.

5. THE RELATIONSHIP WITH CELLULAR AUTOMATA

5.1. Cellular Automata

Preston and Duff describe the theory and applications of cellular automata applied to image processing in [3]. Much of their discussion assumes an architecture based on the CLIP4 machine described by Duff and Fountain [4]. In CLIP4, each pixel in the image plane has a boolean processing element (cell) associated with it. Each processing element is connected only to its nearest neighbors. 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. Preston and Duff discuss primarily operations on binary images. In this case, the image can be held in registers internal to the processor array. We discuss only the binary image processing case in the following.

At each discrete time iteration, a supervising controller determines the computation to be performed by the processing array. Aside from steps for input and output of data, each iteration consists essentially 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. The term, propagation, as used in [3, 4] refers to the propagation of information over the array. In the traditional definition of a cellular automata, each cell outputs its current value to its neighbors. Preston and Duff refer to this as local propagation, as information is only shared locally among nearest neighbor processors. Preston and Duff also allow for global propagation. In this case, a cell will output a second signal to its neighbors based on the value of its own state and whether it has received a similar signal from one of its neighbors. Information is 'propagated' globally throughout the array as the signal is passed from neighbor to neighbor until a steady state is reached. 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. Directional sensitivity can also be added to the cell's input and output of the global propagation signal. This propagation phase allows data to be passed over the entire array, overcoming some of the restrictions of the local interconnectivity of the processing elements while preserving the nearest neighbor interaction.

The architecture of CLIP4 is very similar to that of the CNN. Both are composed of a two dimensional array of simple processors connected only with nearest neighbors. In both cases, the interaction between the neighboring elements is assumed to be uniform over the array. The chief differences between CNNs and CA are that CNNs operate in continuous time and in a

continuous state space, while CA operate in discrete time and in a discrete state space. However, they can both operate on binary images with binary input and output and processing involving only nearest neighbor interactions. The natural question which arises is what types of operations can be accomplished by both the CNN and CA.

5.2. Single Iteration Operations

Given enough layers, a CNN composed solely of linear threshold layers can implement any boolean function of its input. Therefore, by taking 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 by a CNN composed solely of linear threshold layers.* In particular, the logical convolution and binary mask matching operations of [3] can be implemented with a linear threshold single layer CNN. To do this, set the B matrix equal to the mask or the convolution kernel, set the current I equal to the negative of the corresponding threshold, set $A(0,0) = R^{-1}$ and set the input equal to the image to be transformed.

This result can be extended to include global propagation. A CA cell will propagate a global propagation signal based upon two factors: whether its current state is 1 or 0 and whether it has received a propagation signal from one of its neighbors or not. 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 and some condition on the value of the current state is satisfied. Thus, in this section we consider only rules of this type.

In order to realize global propagation in a CNN, an additional layer is added to perform the global propagation. The outputs of this layer are $+1$ if the cell is propagating a signal and -1 otherwise. The cellular logic transform is performed by a set of feedforward linear threshold layers satisfying $A_{k,k}(0,0) = R_k^{-1}$. 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:

$$A = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.25 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 20.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad I = -18.0$$

The coefficients have been normalized so that $R = C = 1.0$.

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 (4). However, in this case, g depends also on the outputs of other cells.

$$g = 0.25 \sum_{\alpha,\beta \neq 0,0} y_{i+\alpha,j+\beta} + 20.0 u_{i,j} - 18.0$$

Note that $A(0,0) - R^{-1} = 0.25$.

Consider cell $C(i,j)$. If $u_{i,j} = -1$, $g \leq -35.75 < -(A(0,0) - R^{-1})$ for all combinations of nearest neighbor outputs. Thus, in steady state $y_{i,j} = -1$. For $u_{i,j} = 1$, $g > A(0,0) - R^{-1}$ if the output of at least one of $C(i,j)$'s nearest neighbors is $+1$. If all of the outputs of $C(i,j)$'s nearest neighbors are -1 then $g = 0$, and the output of the CNN is equal to the initial output. If a cell with an associated $+1$ input is initialized to $+1$, then the outputs of its neighbors which also have an associated $+1$ input will be driven towards $+1$. In turn, the outputs of their neighbors with associated $+1$ input will also be driven to $+1$. This provides the propagation.

There is one subtle point here. Even though a cell in the background directly adjacent to an image component has -1 output in steady state, if initialized to $+1$ it might start propagation through the adjacent image component due to the finite time required for the transition of its output from $+1$ to -1. However, the CNN has been designed so that this transition is essentially instantaneous compared to the time required to start propagation. In the worst case, a cell with $+1$ input and -1 initial condition is surrounded by background cells with -1 input and $+1$ initial condition. The time derivative of the cell with $+1$ input decreases from 3.25 to a negative value as the outputs of cells around it decrease from $+1$ to -1. The time derivatives of the cells with -1 input in the 1-neighborhood of the cell with $+1$ input are always less than -36.25. Thus, the outputs of the cells with -1 input decrease to -1 before the output cell with $+1$ input can reach $+1$ and begin propagation. Thus, a cell will have $+1$ steady state output if and only if it is associated with an image component and was either initialized to $+1$ or received a valid propagation signal from a nearest neighbor.

This type of global propagation has been used in the examples of Section 6, as well as in a 'hole filler' CNN [16] and a 'shadow detector' CNN [17]. One of the examples of Section 6 shows how to incorporate directional sensitivity to the global propagation. Since both local and global propagation can be implemented on a CNN, *any single iteration CA operation consisting of a propagation phase (local or global) and a cellular logic transform can be accomplished with a CNN.*

5.3. Multiple Iteration 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, CNN layers designed to do each iteration can be cascaded into one large CNN on a single chip. To ensure that the iterations are carried out sequentially, the capacitances of each layer are chosen so that the layers corresponding to the first iteration settle much faster than those corresponding to the second and so on. Some of the examples presented in Section 6 use this technique. However, this is limited to operations with a small number of iterates since the capacitance and resistance values must be scaled to slow down the dynamics of progressive layers, leading quickly to impractically large capacitors and resistors for a VLSI chip. Alternatively, a digital controller could start the layers associated with each step of the CNN with the appropriate input and initial conditions once the layers associated with the previous step have settled. In this case, the layers associated with each step can be separated onto different chips.

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, a state

vector will be an 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 $s(\cdot)$ is a boolean function. A state 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. Claim 1 showed that a similar condition for binary output vectors holds for CNNs.

To emphasize the similarity with the CA case, consider the signum function in (7) 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, (7) provides a convenient starting point for converting CA applications to CNN applications.

What if (7) is not satisfied for all k,i,j? For a cellular automata the next state of the network is given a boolean function of the current state. 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 (7) for all k,i,j, then the state of at least one the cells which does not satisfy (7) will enter the linear region of the piecewise-linear sigmoid output function.*

For a proof of this claim, see the appendix. Thus, the output states which do not satisfy (7) for all k,i,j are unstable. In fact, the proof of Claim 2 shows that for some time the states of all the cells which do not satisfy (7) progress towards the linear region. Interestingly, a CA implementation of the corresponding signum function of the connected component detector [8] does settle to same steady state as the CNN for identical initial conditions. However, because of the complex dynamical behavior associated with the continuous time/state dynamics of the CNN, in general this signum function is not an approximate boolean function which the CNN is implementing.

6. DESIGN EXAMPLES

6.1. Corner Extraction From a Noisy Image

This simple example uses the above results to design a two layer CNN which extracts the corners from a noisy image. In the process, we see exactly how the dynamical behavior of the corner detecting CNN presented in [2] performs the desired processing, and how to use this understanding the redesign the corner detecting CNN so that it can be incorporated into a more complex two layer CNN.

Consider the corner detecting CNN presented in [2]. Although the CNN easily detects the corners of a noise free image, additive noise can result in false corner detection. For example, Figure 6 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. The noise causes some corner pixels to be missed and other pixels to be misclassified as corner pixels. It would be helpful if the corner detection CNN could be combined the noise removal CNN also presented in [2] to create a two layer CNN which filters the image before extracting the corners.

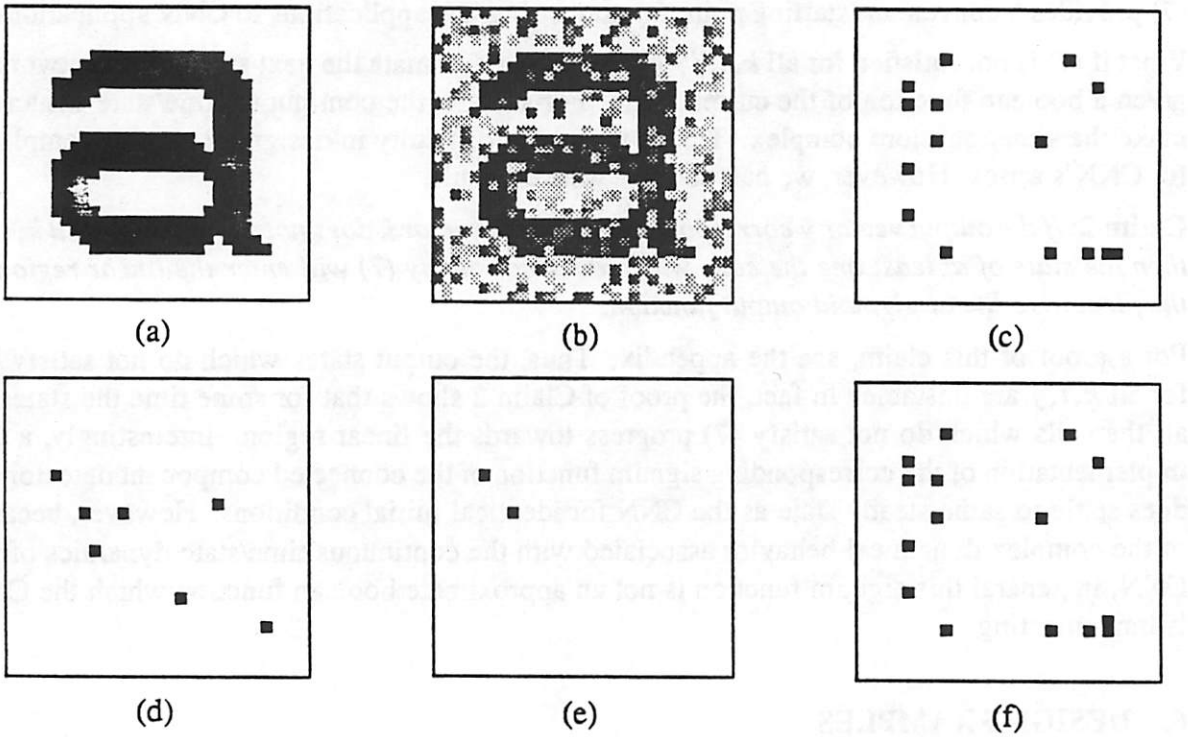


Figure 6: The CNN presented in [2] must be redesigned to detect the corners of a noisy image. (a) The original noise free image. (b) The original image corrupted by zero mean Gaussian noise of variance 0.3. (c) The corners detected using the corner detector of [2] on (a). (d) The corners detected using 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).

One alternative is to 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 may 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 6. This CNN detects only the same true corners that the previous one did, except for the corner associated with the tip of the serif of the 'a'. Adding the noise removal layer has eliminated the false positives, but has not solved the problem of the missed corners.

The state equation of the corner detection cells sheds light on why the two CNNs cannot simply be cascaded as above. The cloning template of the corner detecting CNN is:

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2.0 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \quad I = -3.0. \tag{8}$$

where $R = C = 1.0$. This CNN consists of a single linear threshold layer. Substituting the actual parameters into (1) yields:

$$C\frac{dv_{i,j}}{dt} = v_{i,j} + 2.0f(v_{i,j}) + 2.0u_{i,j} + 0.25 \sum_{\alpha,\beta \neq 0,0} u_{i+\alpha,j+\beta} - 3.0. \tag{9}$$

Since in the previous example the input of the CNN was connected to the output of the noise removal layer, the input to the CNN is in fact time varying. Thus, the state trajectory of the corner detection layer depends not only upon its own (noisy) initial conditions, but also upon the transient response of the noise removal layer.

Ideally, the corner detection layer should operate on the *steady state* output of the noise removal layer. The results of Sections 3 and 4 can completely characterize the processing taking place and be used to redesign the CNN so that it will work properly when cascaded with the noise removal CNN. When working in isolation, the corner detecting CNN is a linear threshold single layer CNN with constant input. The input and initial conditions of the CNN are set equal to the input image, which is denoted by $\{I_{i,j}\}$. Assume binary initial conditions and input. The map from the input and the initial conditions to the output steady state is explicitly given by (5):

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

where n_{-1} and n_1 are the numbers of neighbors in the 1-neighborhood whose inputs are -1 and $+1$ respectively. Using the fact that $I_{i,j} = y_{i,j}(0) = u_{i,j}$ and $n_{-1} + n_1 = 8$ simplifies the above equation to

$$\lim_{t\to\infty} y_{i,j}(t) = \text{sgn}[(3.0I_{i,j} - 3.0) + (2.0 - 0.5n)]\,.$$

where n is the number of neighbors of $I_{i,j}$ which are $+1$. If $I_{i,j} = -1$, then in steady state $y_{i,j} = -1$. If $I_{i,j} = +1$, then the steady state output of the CNN depends upon the sign of $(2.0 - 0.5n_1)$. If $n < 4$, then $\lim_{t\to\infty} y_{i,j}(t) = 1$ since $(2.0 - 0.5n) > 0$. If $n_1 > 4$, then

$\lim_{t\to\infty} y_{i,j}(t) = -1$ since $(2.0 - 0.5n) < 0$. If $n = 4$, then $(2.0 - 0.5n) = 0$ and the signum function is not defined. The state equation (9) shows that this is exactly one of the problematic cases mentioned in Section 3. In this case, due to the inevitable thermal noise, $\lim_{t\to\infty} y_{i,j}(t) = -1$. Summarizing, $\lim_{t\to\infty} y_{i,j}(t) = 1$ if and only if $I_{i,j} = +1$ and less than five of $I_{i,j}$'s neighbors are $+1$, otherwise $\lim_{t\to\infty} y_{i,j}(t) = -1$. Assuming that the CNN's initial conditions and input are set equal to the image data, the steady state output of the CNN is a simple linearly separable function of the image data.

Using this fact, the corner detection layer can be redesigned so that the output depends only upon the input to the CNN, independently of the initial conditions. Consider the following cloning template.

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2.0 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \quad I = -1.75.$$

Again $R = C = 1.0$. Substituting the above parameters into (5) yields:

$$\lim_{t\to\infty} y_{i,j}(t) = \text{sgn}[2.0u_{i,j} + 0.25n_{-1} - 0.25n_1 - 1.75].$$

All of the dependence upon the initial conditions has been removed by setting $A(0,0) = R^{-1}$. Further manipulating the above equation yields

$$\lim_{t\to\infty} y_{i,j}(t) = \text{sgn}[(2.0u_{i,j} - 2.0) + (2.25 - 0.5n)].$$

It can easily be verified that $\lim_{t\to\infty} y_{i,j}(t) = 1$ if and only if $C(i,j)$'s input is $+1$ and less than five of $C(i,j)$'s neighbors' inputs are $+1$. In other words, given identical images, the two CNNs will settle to the exact same steady state. The redesigned CNN has also eliminated the problematic case where thermal noise was required to ensure the output settled to the desired steady state.

If the new corner detection layer is cascaded with the noise removal layer, the transient response of the noise removal layer will still affect the state trajectory of the corner detection layer. However, in steady state the output of the noise removal layer is constant. After the transient response of the noise removal layer has died out, the corner detection layer has constant input. The output steady state of the corner detection layer is a function only of this constant input, independent of any initial conditions set up by the transient response of the noise removal layer. The new two layer CNN settles to a steady state very close to the desired steady state. In fact, the pictures differ only by the pixel associated with the tip of serif 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. Analysis similar to the above can also be used to explain the processing of the edge detection CNN [2].

6.2. Additional Examples

Equation (5) and the global propagation layer have been used to translate many other CA applications presented in [3, 4] to a CNN architecture. Here we present five examples of CNNs which:

1. Extract the holes in an image

2. Extract objects containing holes

3. Extract objects larger than a 3x3 square

4. Extract objects containing specified image pixels

5. Compute the minimal circumscribing octagonal convex hull

Below, the operation of each of these CNNs is described in detail by explaining the processing performed by each layer. Each of these CNNs contains at least one global propagation layer and possibly a linear threshold layer. The are all feedforward. Each linear threshold layer satisfies $A_{k,k}(0,0) = R_k^{-1}$ so that its steady state depends only upon the steady state values of the previous layers and the input. Unless stated otherwise, the coefficients of the cloning template have been normalized so that $R_k = C_k = 1.0$. Any zero elements of the cloning template have been omitted and when possible, the matrices associated with the $A_{k,\gamma}$ and B_k have been reduced to the lowest dimension possible.

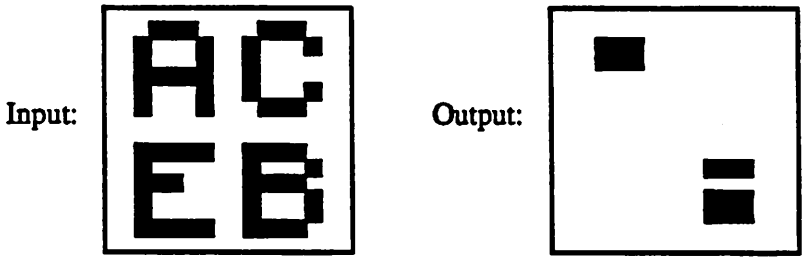
In the following, assume that the background is 4-connected and that objects in the image are 8-connected. A region is 4-connected if each pixel in the region has one face in contact with another pixel in the region. A region is 8-connected if each pixel in the region has one face or corner in contact with another pixel in the region. This distinction between the connectivity of the background and the connectivity of the image objects is necessary to ensure that the Jordan curve theorem is satisfied.

Hole Extraction This CNN finds the holes in an input image. A hole for this application is defined as a 4-connected segment of background which is isolated from the edge of the image plane by an 8-connected image component. This CNN has two layers. The image to be processed is presented as the input to the array. The cells of layer 2 whose steady state output is $+1$ are associated with pixels contained inside the holes in the image. See Figure 7.

Layer 1 fills in the background which is connected to the edge of the array using the global propagation described in Section 5. This layer is essentially the same as the hole filler of Matsumoto, Chua and Furukawa [16]. Cells in this layer are initialized to -1 but are $+1$ in steady state if they have -1 input and share a face with another cell whose output is $+1$. To start propagation from the edge of the array, the imaginary cells outside the boundary of layer 1 output a constant $+1$. Layer 2 is a linear threshold layer which finds the pixels associated with cells whose input and layer 1 output are both -1, i.e., the pixels associated with the holes.

Hole Figure Extraction This CNN is similar to the previous one except it actually extracts the objects which contain the holes in the image. This CNN is also a two layer CNN. The image is presented as the input to the CNN. The output of layer 2 contains the objects in the image which contain holes in their interior. See Figure 8.

In this case, both layers perform global propagation. Layer 1 is exactly the same as above, except the capacitors associated with this layer are much smaller than 1.0 so that its settling time is much faster than that of layer 2. Layer 1 is essentially in steady state for the entire transient response of layer 2. Layer 2 is initialized to -1 and propagates a $+1$ signal through



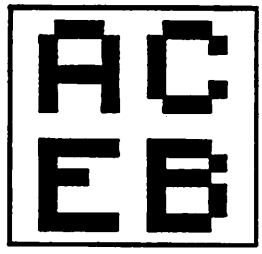
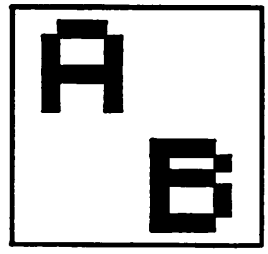
(a)

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

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

(b)

Figure 7: Extraction of the holes in an image. (a) Sample input and output images. (b) The CNN cloning template.

Input:  Output: 

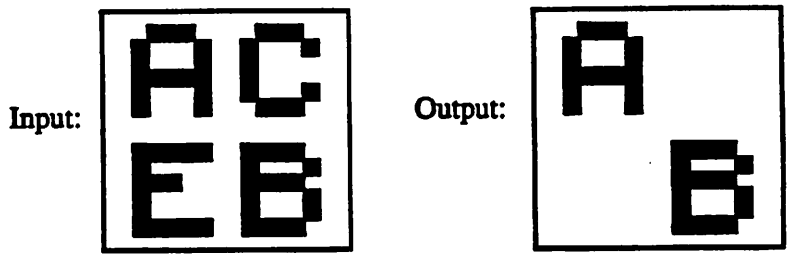
(a)

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

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

(b)

Figure 8: Extraction of objects containing holes. (a) Sample input and output images. (b) The CNN cloning template.



(a)

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

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

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

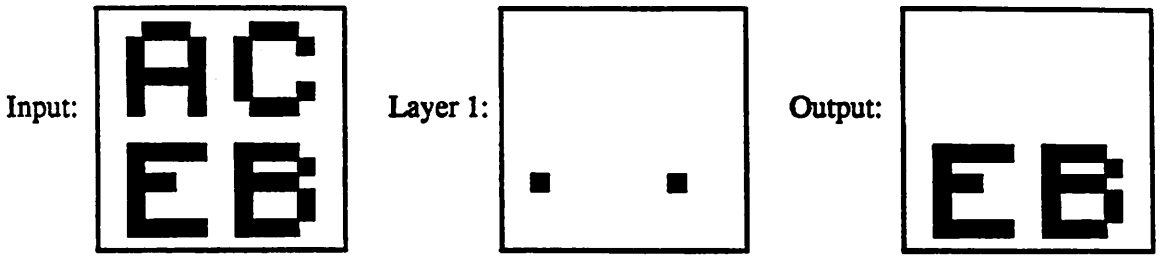
(b)

Figure 9: Extraction of objects larger than a 3x3 pixel square. (a) Sample input and output images. (b) The CNN cloning template.

image pixels associated with objects next to holes. To prevent propagation from the edges, the imaginary cells outside the boundary of layer 2 all output a constant -1.

Large Object Extraction This three layer CNN extracts those figures in the input image whose total area including holes contained in their interior contains a three by three pixel square. The image is presented as the input to the CNN. The output of layer 3 contains the desired figures. See Figure 9.

Layers 1 and 2 are global propagation layers and layer 3 is a linear threshold layer. Again $C_1 \ll 1.0$ so that the cells of layer 1 in the background adjacent to the edge settle to $+1$ steady state output voltage during the initial moments of the transient response of layer 2. Layer 2 is initialized to -1 and its imaginary boundary cells output a constant -1. Propagation in layer 2 is started from pixels at the center of 3x3 pixel squares contained in regions which have not received a propagation signal in layer 1. Propagation is stopped at the boundary of these regions. Layer 3 outputs the image pixels contained the the regions which received the



(a)

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

$$A_{2,2} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1.4 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \quad A_{2,1} = -3.0 \quad I_2 = -1.0$$

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

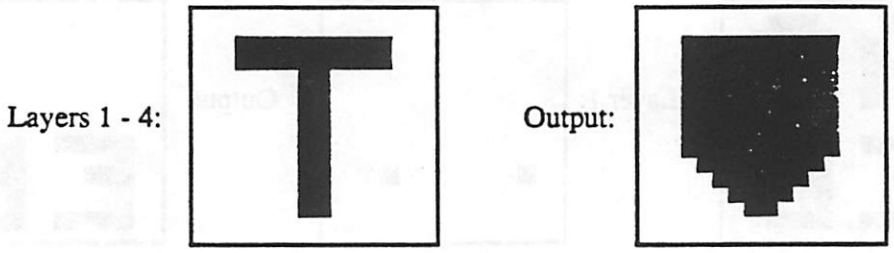
(b)

Figure 10: Extraction of objects containing specified image pixels. (a) Sample input and output images. (b) The CNN cloning template.

propagation signal in layer 2.

Specified Object Extraction Given specified image pixels, layer 3 of this three layer CNN will output the figures in the image whose total area including enclosed holes contains at least one of the specified image pixels. This CNN is also composed of two global propagation layers and a linear threshold layer. However, unlike the previous CNNs, some of the information necessary for processing is contained in the initial conditions of the array. The cells of layer 2 which correspond to the specified image pixels are initialized to $+1$. The output of layer 3 contains objects of the image which contain the pixels specified in layer 2. See Figure 10.

Layer 1 is the same as the layer 1 of the previous examples. Again, $C_1 \ll 1.0$ so that layer 1 settles much faster than layer 2, which subsequently propagates a signal through the unlabeled regions of layer 1 containing a specified image pixel. To prevent propagation from the edges, imaginary boundary cells for layer 2 output a constant -1. Layer 3 is a linear threshold layer which selects those image pixels contained inside the regions receiving a propagation signal in layer 2.



(a)

$$A_{1,1} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.25 & 2.0 & 0.25 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad I_1 = 1.1 \qquad A_{2,2} = \begin{bmatrix} 0.0 & 0.25 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.25 & 0.0 \end{bmatrix} \quad I_2 = 1.1$$

$$A_{3,3} = \begin{bmatrix} 0.25 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.25 \end{bmatrix} \quad I_3 = 1.1 \qquad A_{4,4} = \begin{bmatrix} 0.0 & 0.0 & 0.25 \\ 0.0 & 2.0 & 0.0 \\ 0.25 & 0.0 & 0.0 \end{bmatrix} \quad I_4 = 1.1$$

$$A_{5,1} = 0.25 \quad A_{5,2} = 0.25 \quad A_{5,3} = 0.25 \quad A_{5,4} = 0.25 \quad A_{5,5} = 1.0$$

$$I_5 = -0.85$$

(b)

Figure 11: Computation of the minimal circumscribing octagonal convex hull. (a) Sample input and output images. (b) The CNN cloning template.

Octagonal Hull Computation The output of layer 5 of this five layer CNN is the minimum circumscribing convex set whose boundaries are restricted to lie parallel or at 45 degrees to the coordinate axes. This CNN demonstrates the use of directional sensitivity in the global propagation. See Figure 11.

Initial conditions of layer 1 through 4 are the image. Each of these layers propagates a signal from the image in one of the four directions associated with the borders of the octagonal hull. To prevent signals propagating from the edges, the imaginary cells outside the boundary of the array should all be -1. Layer 5 is a linear threshold layer which selects those pixels in the intersection of the outputs of layers 1 through 4.

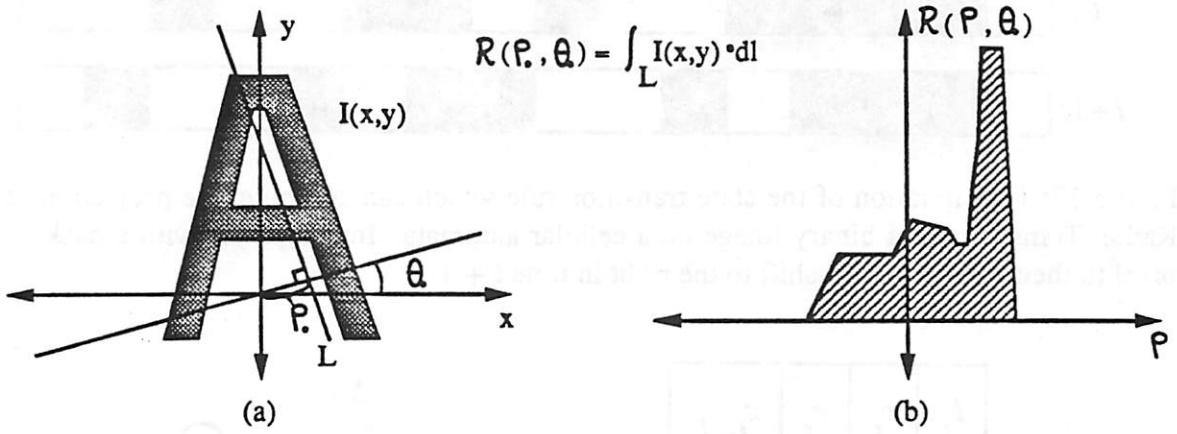


Figure 12: (a) For fixed values of ρ_0 and θ_0 the Radon Transform $\mathcal{R}(\rho_0, \theta_0)$ of an image $I(x, y)$ is the line integral of $I(x, y)$ along line L. (b) For fixed θ_0, the function $\mathcal{R}(\rho, \theta_0)$ is the projection of the image onto the line $\theta = \theta_0$.

6.3. A Radon Transforming CNN

Introduction Up to this point, linear threshold layers have been used as a means of working around the effects of the continuous time/state dynamics. This example uses linear threshold layers to produce a CNN whose operation depends critically upon the effects of the continuous time dynamics of the CNN. 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 $\mathcal{R}(\rho, \theta)$, where ρ and θ are polar coordinate variables. For specific values of ρ_0 and θ_0, $\mathcal{R}(\rho_0, \theta_0)$ 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 $\mathcal{P}(\rho) = \mathcal{R}(\theta_0, \rho)$ is the projection of the image intensity onto the line $\theta = \theta_0$. See Figure 12. For binary images, the horizontal, vertical and diagonal projections are sufficient to compute the zeroth, first and second moments of a region in a binary image. These moments can be used to find the position and orientation of that object [18].

The CNN template presented here finds the value of $\mathcal{R}(\rho, \pi/2)$ 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 Transform, the image can be rotated through all desired angles. The output $\mathcal{R}(\rho, \pi/2)$ 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 chapter, 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 consider 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. See Figure 13. In other words, the value of each pixel at time $t + 1$ is determined by

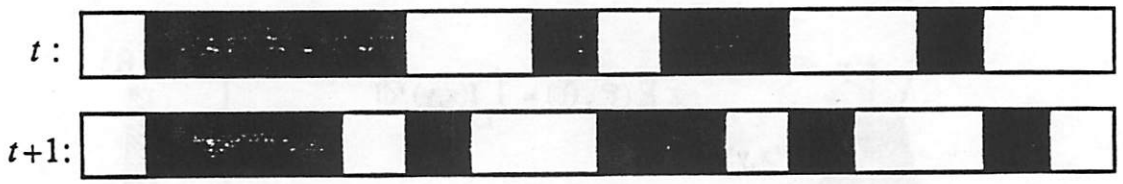


Figure 13: One iteration of the state transition rule which can compute one projection of the Radon Transform of a binary image on a cellular automata. Image pixels with a background pixel to their right at time t shift to the right in time $t+1$.

l_t	c_t	r_t	c_{t+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

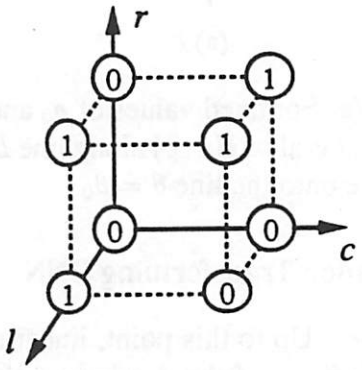


Figure 14: The values which map to zero and one by the boolean function (10) are not separable by a plane in (l, c, r) variable space

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:

$$c_{t+1} = c_t r_t + l_t \bar{c}_t. \tag{10}$$

where $\bar{c}_t$ is the complement of c_t. The values of equation (10) are shown in graphical and tabular form in Figure 14. The boundary conditions should be set up so that the cells on the left hand side of the image plane have $l_t = 0$ for all t and the cells on the right hand side 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 15 where $R_k = C_k = 1.0$ for all k. 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 the output histogram.

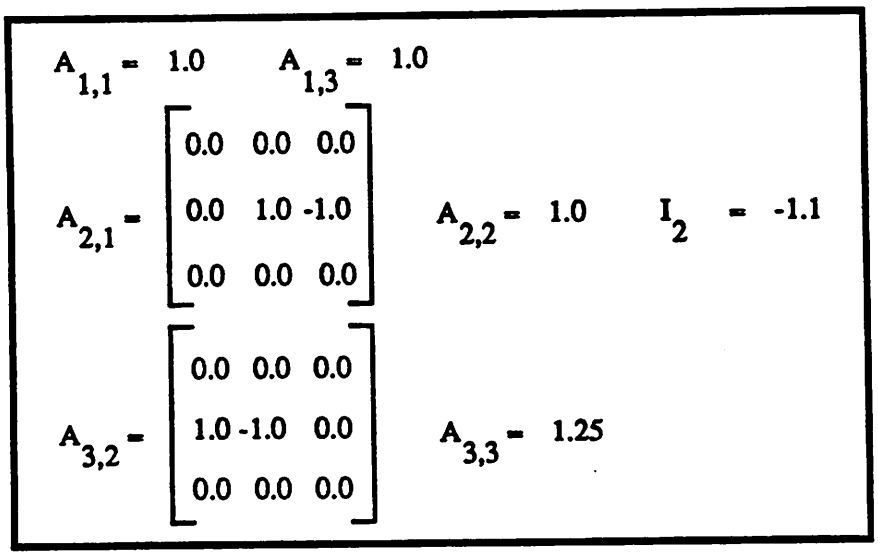
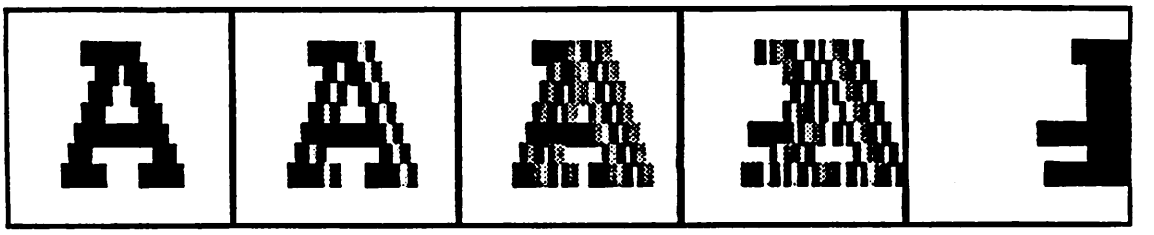


Figure 15: The cloning template of the Radon Transforming CNN shows that it is composed solely of linear threshold layers.

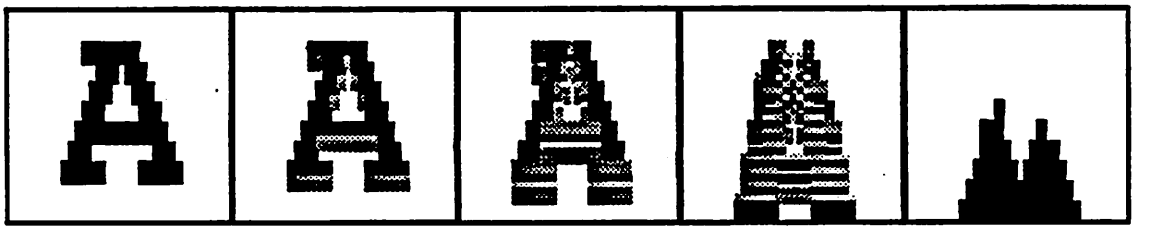
To see the similarity between the operation of the CNN circuit and the iterated cellular logic transform, assume for the moment the state of the cells in layer 1 is constant. Essentially, layers 2 and 3 compute the next logical state of the iterated cellular logic transform. Notice that all of the layers are of the linear threshold class. Since the boolean function (10) is not linearly separable in the (l, c, r) boolean variable space, 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 $A_{1,3}$ 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 output of layer 1, the output of layer 2 evolves toward the data required to update level 3. The output of layer 3 evolves toward the next logical state. Simultaneously, layer 1 evolves so that it reflects the current state of layer 3 and so on.

This heuristic 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 not feedforward. However, each layer is 'insulated' from its own 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



(a)



(b)

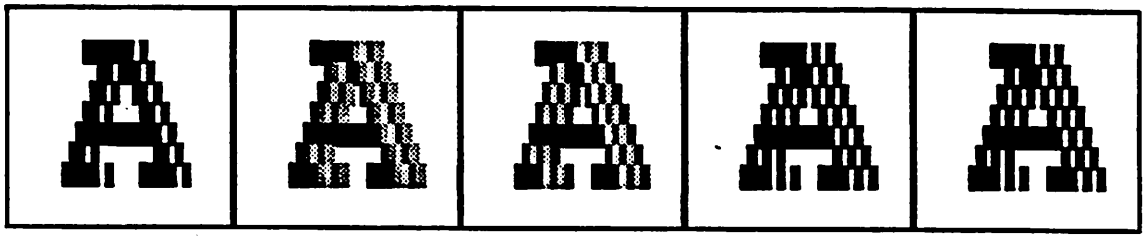
Figure 16: (a) Simulations of the Radon Transforming CNN indicate that it projects the image intensity onto the vertical axis. (b) To obtain other projections of the Radon Transform, the input image can be rotated. Here the image has been rotated by 90 degrees to obtain the projection onto the horizontal axis.

are essential to the correct operation of this circuit.

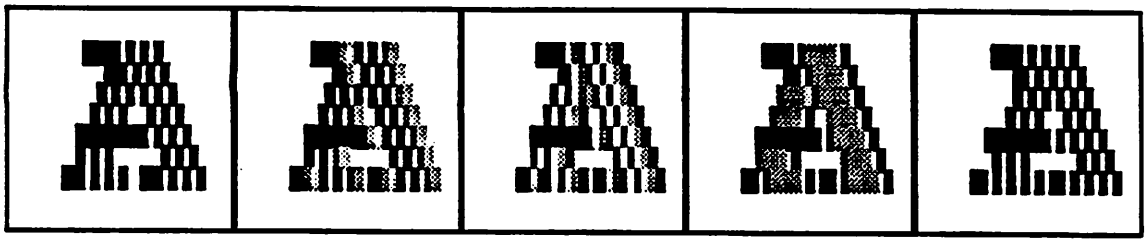
Simulations of the circuit indicate that the circuit does settle to the desired steady state. See Figure 16. It seems that passing the update information through a sequence of three layers enables the circuit to preserve the total number of image pixels in the image. Although the logic described above could be implemented with only two layers, simulations run with only two layers did not converge. 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 17. 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.

7. CONCLUSION

In this chapter, we have discussed a simple subclass of all the possible CNNs. Because the state dynamics of these CNNs are so simple, we were able to derive an explicit map between the input and initial conditions of the CNN to the steady state output. These results explain how the dynamics perform the processing of some of the examples in [2]. This insight into the operation of these CNNs has allowed us to redesign the corner detecting CNN of [2] to



(a)



(b)

Figure 17: Although the Radon Transforming CNN has been developed using ideas from cellular automata, the CNN does not operate like a CA. Although pixels shift synchronously at the start of the transient (a), as time progresses, the pixels shift asynchronously (b).

incorporate it into a more complex multi-layered CNN, as well as to design new binary image processing applications for the CNN. The similarities between the architectures of CNNs and CA have proved quite useful in designing new applications. In fact, we have shown that any single iteration operation possible on a CA is also possible on a CNN. We also discuss the extension of these results to multiple iteration CA operations, although much work remains to be done in this area. This paper has only described a small subset of the possible CNNs, and thus only a subset of the possible applications. Hopefully, the continuous time and continuous state dynamics will enable the CNN to be applied to a much richer class of problems.

ACKNOWLEDGEMENT

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.

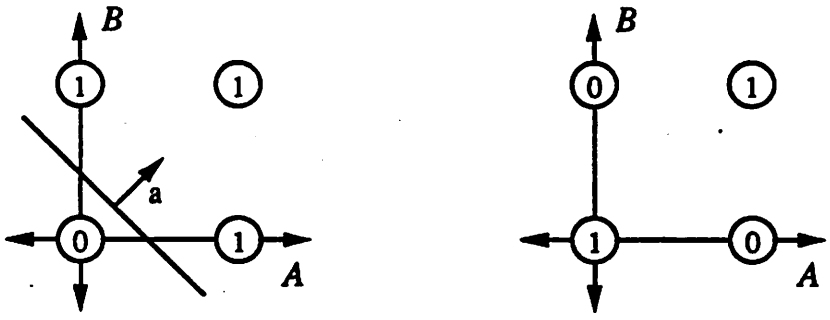


Figure 18: $A + B$ is linearly separable, while $A = B$ is not.

APPENDIX

A Linearly Separable Boolean Functions and Multiple Layer Networks

For our purposes, we can view a boolean function of n variables, f, as a mapping from $\{0,1\}^n$ to $\{0,1\}$. In other words, f 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 $\Re^n$, f is linearly separable if and only if there exists a vector $\mathbf{a} \in \Re^n$ and a constant $b \in \Re$ such that the following two inequalities are satisfied:

$$\langle \mathbf{a}, \mathbf{x} \rangle \geq b \quad \forall \mathbf{x} \in \{\mathbf{x} \in \{0,1\}^n \mid f(\mathbf{x}) = 1\} \tag{11}$$

$$\langle \mathbf{a}, \mathbf{x} \rangle < b \quad \forall \mathbf{x} \in \{\mathbf{x} \in \{0,1\}^n \mid f(\mathbf{x}) = 0\}. \tag{12}$$

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 $\Re^n$ and all the points which map to 0 lie on the other side of the hyperplane. For example, in two dimensions the function $A + B$ is linearly separable while the function $A = B$ is not. See Figure (18). In particular, examination of Figure 14 reveals that the boolean function (10) is not linearly separable.

If f is linearly separable, we can always find a vector $\hat{\mathbf{a}}$ and a constant $\hat{b}$, such that the inequalities in (11) and (12) are strict. To see this, assume that $\mathbf{a}$ and b satisfy (11) and (12). Take $\delta = \min\{b - \langle \mathbf{a}, \mathbf{x} \rangle \mid \mathbf{x} \in \{0,1\}^n, f(\mathbf{x}) = 0\}$. This minimum is greater than zero since it is the minimum of a finite set of positive numbers. The vector $\hat{\mathbf{a}} = \mathbf{a}$ and constant $\hat{b} = b - \frac{1}{2}\delta$ satisfy (11) and (12) with strict inequality. In other words, for any $\mathbf{x} \in \{0,1\}^n$, $\langle \hat{\mathbf{a}}, \mathbf{x} \rangle - \hat{b} \neq 0$. Thus, any boolean function which is linearly separable can be implemented using signum functions like those in equations (2),(5), and (7).

B Proofs of Claims 1 and 2

Proof of Claim 1

First assume that (2) is satisfied for all i, j. Define for each i, j

$$\hat{v}_{i,j} = R \cdot \left[\sum_{\alpha,\beta} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I \right] \tag{13}$$

Clearly, $\hat{\mathbf{v}} = \{\hat{v}_{i,j}\}$ will be an equilibrium point if $\hat{y}_{i,j} = f(\hat{v}_{i,j})$ for all i, j. Consider the case where $\hat{y}_{i,j} = 1$. Then from (2),

$$(A(0,0) - R^{-1})\hat{y}_{i,j} + \sum_{\alpha,\beta \neq 0,0} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I > 0$$

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

$$\sum_{\alpha,\beta} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I > R^{-1}\hat{y}_{i,j}$$

Multiplying through by R and noting that $\hat{y}_{i,j} = 1$, we see that $\hat{v}_{i,j} > 1$, i.e. $f(\hat{v}_{i,j}) = \hat{y}_{i,j}$. The case of $\hat{y}_{i,j} = -1$ is proved similarly. Thus, $f(\hat{v}_{i,j}) = \hat{y}_{i,j}$ is true for all i, j and $\hat{\mathbf{v}}$ is the associated equilibrium point.

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

$$0 = -R^{-1}\hat{v}_{i,j} + \left[\sum_{\alpha,\beta} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I \right]$$

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

$$0 \leq -R^{-1}\hat{y}_{i,j} + \sum_{\alpha,\beta} A(\alpha,\beta)\hat{y}_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I$$

The same statement holds for $\hat{y}_{i,j} = -1$ with the inequality reversed. The equality is strict by assumption and the combination of the two statements results in (2).

To show asymptotic stability we must show that for each $\hat{\mathbf{v}} = \{\hat{v}_{i,j}\}$ as defined in (13), there exists a $\rho > 0$ such that for all $\mathbf{v}_0 \in B(\hat{\mathbf{v}}, \rho) = \{\mathbf{v} \mid \|\mathbf{v} - \hat{\mathbf{v}}\| < \rho\}$ the state trajectory starting at $\mathbf{v}_0$ approaches $\hat{\mathbf{v}}$ asymptotically. For this proof, we define $\|\mathbf{v}\| = \max_{i,j}\{|v_{i,j}|\}$. This definition is not restrictive, since all norms on a finite dimensional Euclidean space are topologically equivalent. Set $\rho = \min_{i,j}\{|\hat{v}_{i,j} - \hat{y}_{i,j}|\}$. This is well defined and greater than zero as it is the minimum over a finite set of of numbers which are greater than zero since the quantity in (2) 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 $\mathbf{v}_0 \in B(\hat{\mathbf{v}}, \rho)$ and consider the state trajectory starting at $\mathbf{v}_0$. The state of cell $C(i, j)$ evolves according to the following ordinary differential equation:

$$C\frac{dv_{i,j}}{dt} = -\frac{v_{i,j}}{R} + \left\{ \sum_{\alpha,\beta} A(\alpha,\beta)y_{i+\alpha,j+\beta} + \sum_{\alpha,\beta} B(\alpha,\beta)u_{i+\alpha,j+\beta} + I \right\} \tag{14}$$

The quantity in braces is the constant $R^{-1}\hat{v}_{i,j}$. Thus $v_{i,j}$ approaches $\hat{v}_{i,j}$ asymptotically for all i, j, implying that $\mathbf{v}$ approaches $\hat{\mathbf{v}}$ asymptotically.

Proof of Claim 2

For consistency of notation with the proof of Claim 1, we prove Claim 2 for the single layer case. The proof carries over to the multiple layer case by a slight change in notation.

Since (2) is not satisfied for all i, j, there exists a set $\{i_\eta, j_\eta\}_{\eta=1}^{U}$ for which (2) is not satisfied. U is the number of states for which (2) is not satisfied. As long as the output state of the network is equal to $\hat{y}$, the cells of the network evolve according to (14). However, for all $(i, j) \in \{i_\eta, j_\eta\}_{\eta=1}^{U}$, the quantity in braces is less than one when $\hat{v}_{i,j}$ is greater than or equal to one (since $\hat{y}_{i,j}$ is equal to one) and greater than -1 when $\hat{v}_{i,j}$ is less than or equal to -1. Thus, in some finite time the state of one of the cells $C(i, j)$ such that $(i, j) \in \{i_\eta, j_\eta\}_{\eta=1}^{U}$ 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 output of the network and thus will remain close to its original value. Once the state of some of the cells have entered the linear region, the dynamics become much more complex.

References

[1] Leon O. Chua and Lin Yang. Cellular Neural Networks: Theory. *IEEE Transactions on Circuits and Systems*, 32, October 1988.

[2] Leon O. Chua and Lin Yang. Cellular Neural Networks: Applications. *IEEE Transactions on Circuits and Systems*, 32, October 1988.

[3] Kendall Preston, Jr. and Michael 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] L. O. Chua and B. E. Shi. Exploiting cellular automata in the design of cellular neural networks for binary image processing. Memorandum UCB/ERL M89/130, University of California at Berkeley Electronics Research Laboratory, November 1989.

[6] L. O. Chua L. Yang and K. R. Krieg. VLSI implementation of cellular neural networks. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2425–2427. IEEE, 1990.

[7] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, 81, May 1984.

[8] T. Matsumoto, L. O. Chua, and H. Suzuki. CNN cloning template: Connected component detector. *IEEE Transactions on Circuits and Systems*, 37:633–635, May 1990.

[9] T. Matsumoto, L.O. Chua, and T. Yokohama. Image thinning with a cellular neural network. *IEEE Transactions on Circuits and Systems*, 37:633–635, May 1990.

[10] Leon O. Chua and Tamás Roska. Stability of a class of nonreciprocal cellular neural networks. IEEE Transactions on Circuits and Systems, to appear.

[11] Leon O. Chua, C.A. Desoer, and E.S. Kuh. *Linear and Nonlinear Circuits*. McGraw-Hill Book Company, New York, 1987.

[12] Leon O. Chua. *Introduction to Nonlinear Network Theory.* McGraw-Hill Book Company, New York, 1969.

[13] R. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1959.

[14] Marvin Minsky and Seymour Papert. *Perceptrons.* The MIT Press, Cambridge, MA, 1969.

[15] David E. Rumelhart, James L. McClelland, and The PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. The MIT Press, Cambridge, MA, 1986.

[16] T. Matsumoto, L.O. Chua, and R. Furukawa. CNN cloning template: Hole-filler. *IEEE Transactions on Circuits and Systems*, 37:635–638, May 1990.

[17] L. O. Chua T. Matsumoto and H. Suzuki. CNN cloning template: Shadow detector. *IEEE Transactions on Circuits and Systems*, 37(8):1070 – 1073, August 1990.

[18] Berthold Klaus Paul Horn. *Robot Vision.* The MIT Press, Cambridge MA, 1986.