WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS
AND INFORMATION SCIENCE

Neural Networks

Lecture 6

Solving of Optimization Problems

Optimization problems

The ability for parallel computation yields to the possibility to work with big amount of data. It allows neural networks to solve complex and time consuming optimization problems.

Optimization problems

The basic problem is to replace the task to the problem of minimization of an energy function describing the recurrent net treated as the *minimization network.*

Optimization problems

<u>**The following problems have to be solved**</u>

1. To define the problem by use of a neural network. Its final (stable) stage should determine the optimization problem
2. The energy function minimum value has to be equivalent of the solution of optimization problem

Optimization problems

3. The net structure, connection weights, threshold and out signals.
4. The elements dynamic have to assure the minimization of energy function
5. The definition of the initial values of elements

Optimization problems

- For typical combinatorial optimization problems an energy function has a form

$$E = \sum_i A_i(V_i) + B * F$$

where V_i - *is* the measure of an *i-th* constraint

F - is the objective function

A_i - and *B* are the coefficients

The TSP Optimization Problem

The „**Travelling Salesman Problem**" (TSP) is a classic of difficult optimization.

Goal:

The set of N cities *A, B, ...* have (pairwise) distance separation d_{AB}, d_{AC}, ..., d_{BC}, ...

The problem is to find a closed tour which visits each city once, returns to the starting city, and has a short (or minimum) total part length.

The TSP Optimization Problem

- There are $\frac{N!}{2N} = \frac{(N-1)!}{2}$ distinct paths for closed TSP routes and the problem is NP-hard (complete).

To describe the N neurons in the TSP network to compute a solution to the problem, the network must be described by an energy function in which the lowest energy state (the network stable state) corresponds to the best path.

The TSP Optimization Problem

The Energy Function must be determined
in such a way that its minima correspond
to solutions of the problem considered.
The Hopfield energy function may contain several
energy terms, which may be roughly classified
into constraint terms and the objective function.

To solve by Hopfield net we need to decide the
architecture:

- How many neurons?
- What are the weights?

The TSP Optimization Problem

For ***N*** cities in computer simulation the network was represented by a matrix ***N*** x ***N***.
The system is characterize using a table of cities (rows) and steps (columns).
An entry $\boldsymbol{v_{Xi}}$ is equal to **1** if city ***X*** is visited at step *i*, and **0** otherwise.

At the end of a simulation test which converged to a solution each element representing output potential of neuron $\boldsymbol{v_{Xi}}$, was equal to either zero or one.

The TSP Optimization Problem

The Hopfield network is fully interconnected, that is, all neurons are connected to all other neurons (there are no layers). The weights are symmetric.

There are three main types of weights:

a) *Each neuron has its own positive bias weight.*
b) *Each neuron has a negative (inhibitory) weight to each of the other neurons in its row and its column (α).*
c) *Each neuron has a negative weight to other possible cities just before it and just after it on the tour (d_{XY}). This weight is equal to Euclidean distance. Since the tour makes a loop, the final column is connected to the first column by distance weights.*

The TSP Optimization Problem

The energy function

$$E = \frac{A}{2}\sum_{X}\sum_{Y}\sum_{i,i\neq j} v_{Xi}v_{Xj} + \frac{\mathrm{B}}{2}\sum_{\mathrm{i}}\sum_{X}\sum_{Y,Y\neq X} v_{Xi}v_{Yi}$$

$$+\frac{\mathrm{C}}{2}\Big(\sum_{\mathrm{X}}\sum_{\mathrm{i}} v_{Xi} - N\Big)^2 +$$

$$+\frac{\mathrm{D}}{2}\sum_{\mathrm{X}}\sum_{\mathrm{Y,Y\neq X}}\sum_{\mathrm{i}} d_{XY}v_{Xi}(v_{Y,i+1}+v_{Y,i-1})$$

X, Y – denote cities;

i,j – denote the tour stage

v_{Xi} – is the activation for each neuron, v_{Xi}=1 denotes the fact that city *X* is the i-th city visited in the tour

d_{XY} – denotes the distance between cities *X* and *Y*

The TSP Optimization Problem

Row constraint (first term) in the energy function is zero if and only if there is only one „1” in each order column; thus it takes care that no two or more cities are in the same travel order i.e. no two cities are visited simultaneously.

Column constraint (second term) is zero if and only if there is only one city appears in each order column; thus it takes care that each city is visited only once.

These terms represent the constraint of the problem.

The TSP Optimization Problem

Total number of „1" constraint (third term) is zero if and only if there are only N number of „1" appearing in the whole N*N matrix; thus is takes into care that all cities are visited.

That term represents the constraint of the problem.

The **fourth term** measures the tour length corresponding to a given tour, where the two terms inside the parenthesis stand for two neighboring visiting cities if V_{Xi} implying the tour length is calculated twice.

That term represents the objective function.

The TSP Optimization Problem

The fourth term mmeasures the total distance by adding intercity distances $d_{X,Y}$ for each pair of adjacent cities.

Here **A**, **B** and **C** are positive integers, the setting of these constants are critical for the performance of Hopfield network.

The TSP Optimization Problem

The set solving the TSP problem for N = 4.

The route: 3-2-4-1

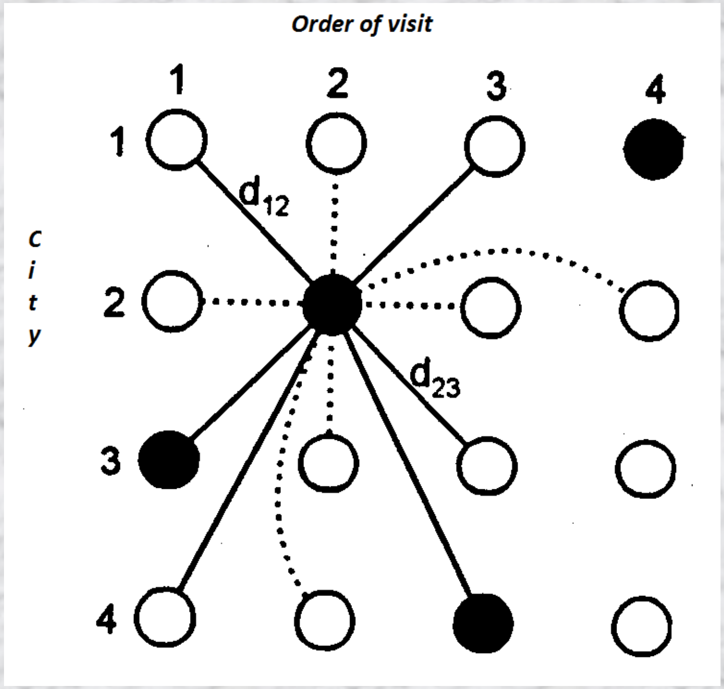
● tern on element

tern off element

connections for V_{22}

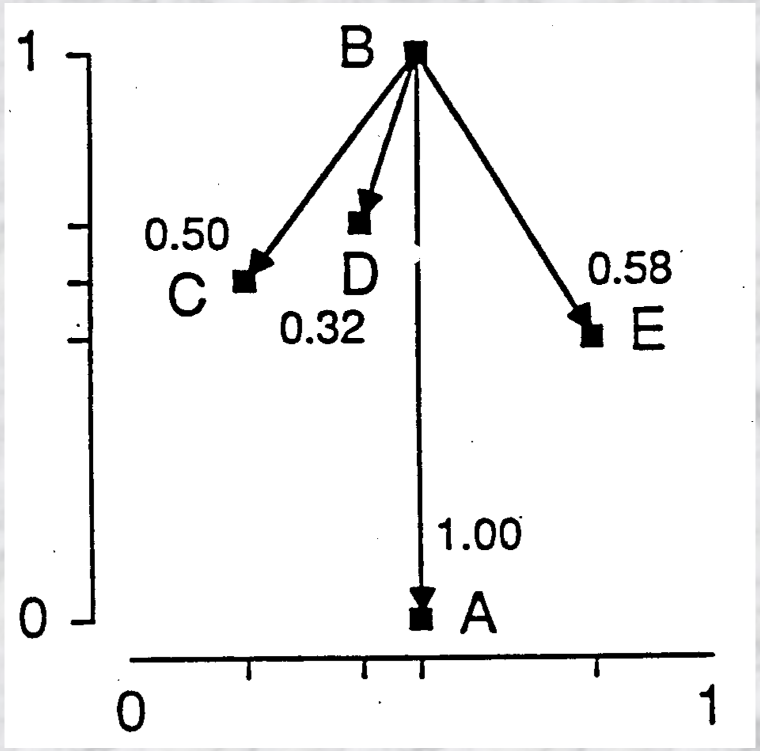
inhibitory, weight d_{XY}

·····inhibitory, weight α



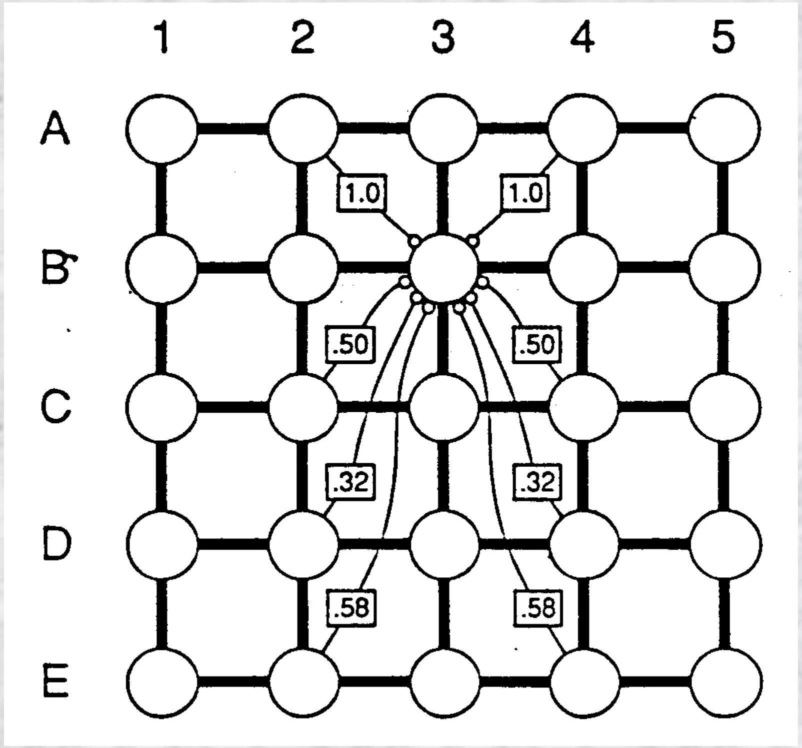
The TSP Optimization Problem

The example of normalized distances from the city B to other cities.



The TSP Optimization Problem

- Inhibitory connections of the V_{B3} element with the rest of cities



Neural Networks for Matrix Algebra Problems

Neural Networks for Matrix Algebra Problems

The feedforward neural networks for solving (in real time) a large variety of important matrix algebra problems such as:

- matrix inversion,
- matrix multiplication
- LU decomposition,
- the eigenvalue problem

Neural Networks for Matrix Algebra Problems

These algorithms basing on the massively parallel data transformation assure the high speed (µsek) in practice – in the real-time.

For a given problem define the error (energy) function and proper multilayer network and during learning phase find the minimum of the error function

Neural Networks for Matrix Algebra Problems

Matrix inversion

Let **A** be a nonsingular square

Goal:

To find the neural network calculating the matrix $\mathbf{B} = \mathbf{A}^{-1}$. matrix **B** fulfill the relation

$$\mathbf{BA} = \mathbf{I}$$

Neural Networks for Matrix Algebra Problems

Multiplying both sides by arbitrary non-zero vector **x**=[$x_1,x_2,\ldots,x_n$] we get

$$\mathbf{BAx - x = 0} \qquad \mathbf{(1)}$$

The energy (error) function can be defined by

$$E = \frac{1}{2}\left\| BAx - x \right\| \qquad (2)$$

Neural Networks for Matrix Algebra Problems

Solving of equation (1) can be replaced by the minimization of the function (2).

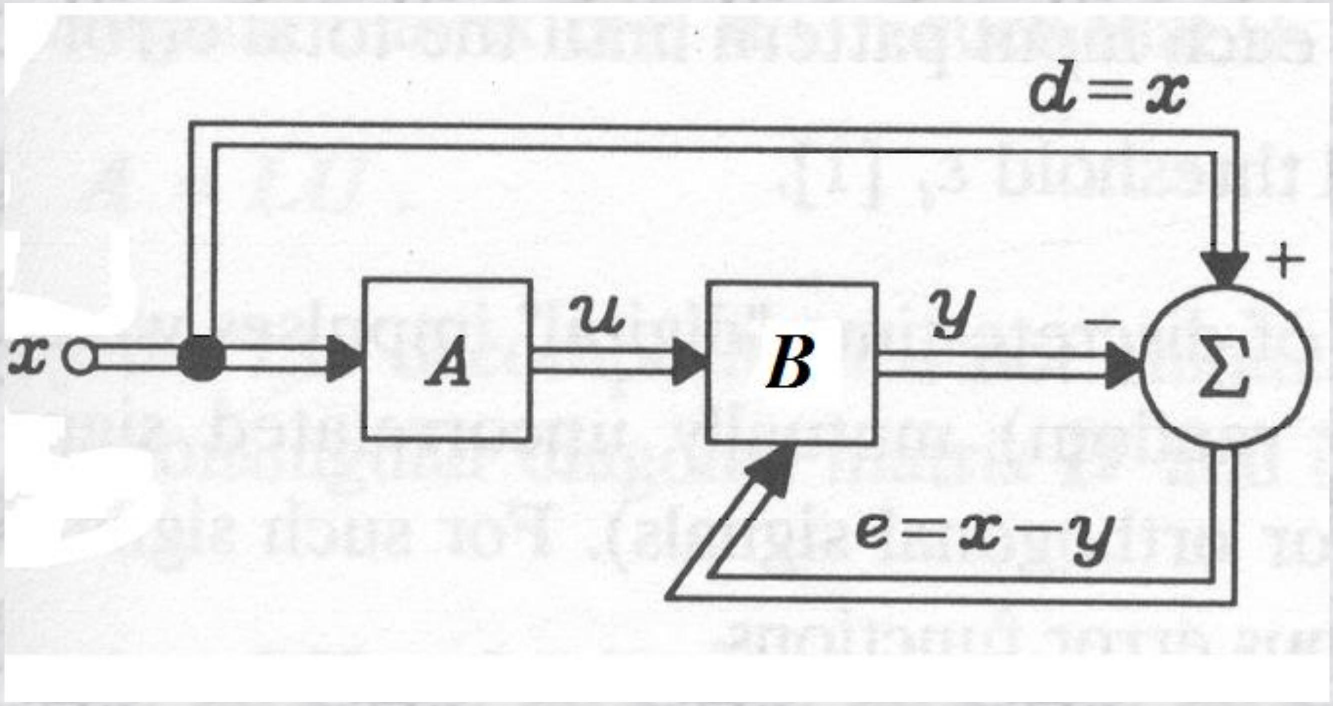
Vector **x** plays double role:

- is the learning signal (network input signal),
- is the desired output (target) signal

i. e. **it is the autoassociative network**

Neural Networks for Matrix Algebra Problems

A simplified block diagram



Neural Networks for Matrix Algebra Problems

u = Ax,   y = Bu   or

y = Bu = BAx = Ix = x

It means that the output vector signal **y** must be equal to the input vector signal **x** – i.e. the network should learn the identity map **y** = **x.**

The fundamental question for the training phase:

what kind of input signals x should be applied in order to obtain the desired solution?

Neural Networks for Matrix Algebra Problems

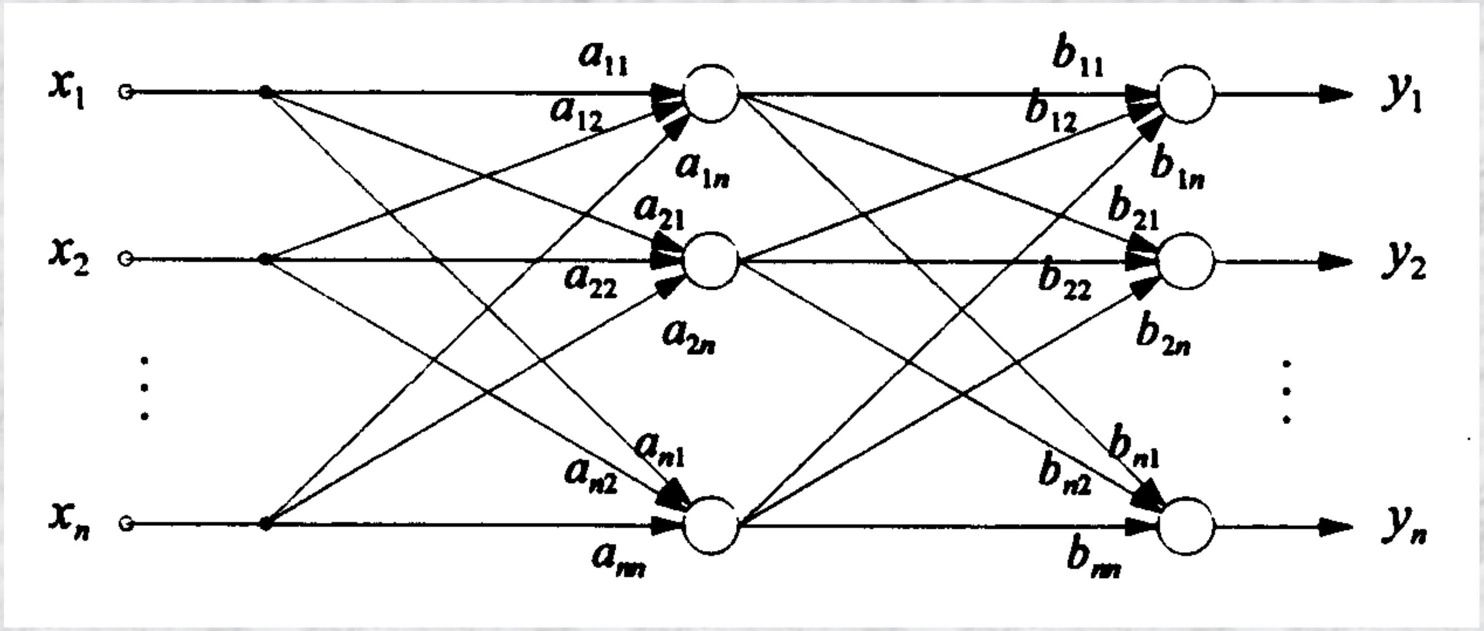
One of the simplest input patterns can be chosen as:
$\mathbf{x}^{(1)}=[1,0,0,\ldots,0]^T$, $\mathbf{x}^{(2)}=[0,1,0,\ldots,0]^T,\ldots,$ $\mathbf{x}^{(n)}=[0,0,0,\ldots,1]^T$.

The better convergence speed can be obtained by changing the input patterns randomly on each time step from the set
$\mathbf{x}^{(1)}=[1,-1,\ldots,-1]^T$, $\mathbf{x}^{(2)}=[-1,1,-1,\ldots,-1]^T,\ldots,$
$\mathbf{x}^{(n)}=[-1,-1,\ldots,1]^T$.

In this two-layer network the first layer has fixed connection weights a_{ij}, while in the second layer weights are unknown, and are described by the unknown matrix **B** = **A**^{1}**.**

Neural Networks for Matrix Algebra Problems

The network architecture



Neural Networks for Matrix Algebra Problems

In order to minimize the local error function ***E***

$$E = \frac{1}{2}\sum_{j=1}^{n} e_j^2 = \frac{1}{2}\sum_{j=1}^{n}(x_j - y_j)^2$$

for a single pattern

y_i is the actual output signal
x_i is the desired output signal

we can apply a standard steepest-descent approach

$$\Delta B_{ij} = -\mu(y_i - x_i)u_j$$

Neural Networks for Matrix Algebra Problems

Matrix multiplication

If matrix **C** is equal the product of matrices **A** and **B** it fulfills the equation

$$\mathbf{C} = \mathbf{AB}$$

Neural Networks for Matrix Algebra Problems

To construct a proper neural network able to solve the problem it is necessary to define the error (energy) function whose minimization leads to the desired solution. Multiplying both sides by arbitrary non-zero vector $\mathbf{x}=[x_1,x_2,\ldots,x_n]$ we get

$$\mathbf{ABx} - \mathbf{Cx} = \mathbf{0}$$

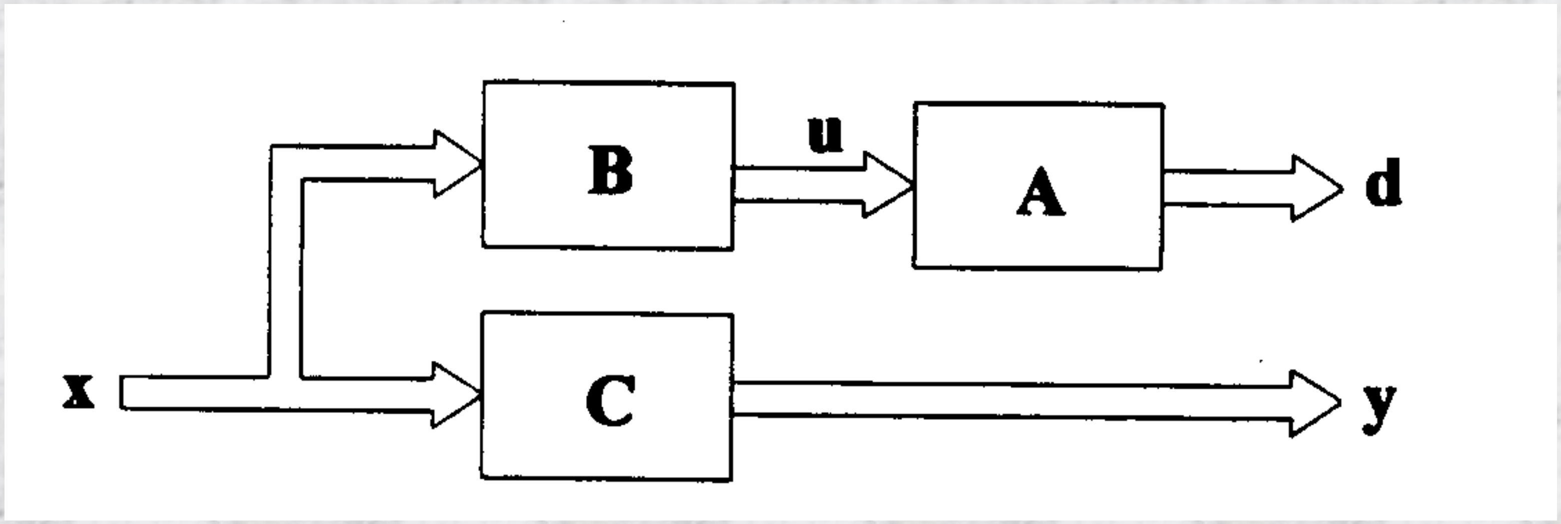
Neural Networks for Matrix Algebra Problems

On the basis of this equation we can define the error (energy) function

$$E = \frac{1}{2}(\|ABx - Cx\|_2)^2$$

Neural Networks for Matrix Algebra Problems

A simplified block diagram for matrix multiplication. In real it is one-layer network in spite that on the diagram there are three layers



Neural Networks for Matrix Algebra Problems

Only one out of these three layers responsible for matrix **C** is the subject of a learning procedure – realizing the equation

$$y = \mathbf{C}\mathbf{x}$$

After the learning process the network has to fulfill the equation **C** = **AB** in the diagram there are two additional layers with constant weights (the elements of matrices **A** and **B** respectively).

Neural Networks for Matrix Algebra Problems

These layers are used to compute the vector **d**, according to

$$\mathbf{d} = \mathbf{Au} = \mathbf{ABx}$$

Again we can apply a standard steepest-descent algorithm. The adaptation rule has the form $c_{ij}(t+1) = c_{ij}(t) + \eta(d_{ij} - y_{ip})x_{jp}$

where *p* is the number of a learning pattern.

Neural Networks for Matrix Algebra Problems

LU decomposition

The standard **LU** decomposition of a square matrix **A** into: lower-triangular matrix **L** and upper-triangular matrix **U** such that:

$$\mathbf{A = LU}$$

generally the **LU** decomposition is not unique. However, if the **LU** is factorization for a lower-triangular matrix **L** with unit diagonal elements factorization is unique.

Neural Networks for Matrix Algebra Problems

Multiplying both sides by arbitrary non-zero vector $\mathbf{x}=[x_1,x_2,\ldots,x_n]$ and after some further transformation we get the energy function

$$E=\frac{1}{2}(\|LUx-Ax\|_2)^2$$

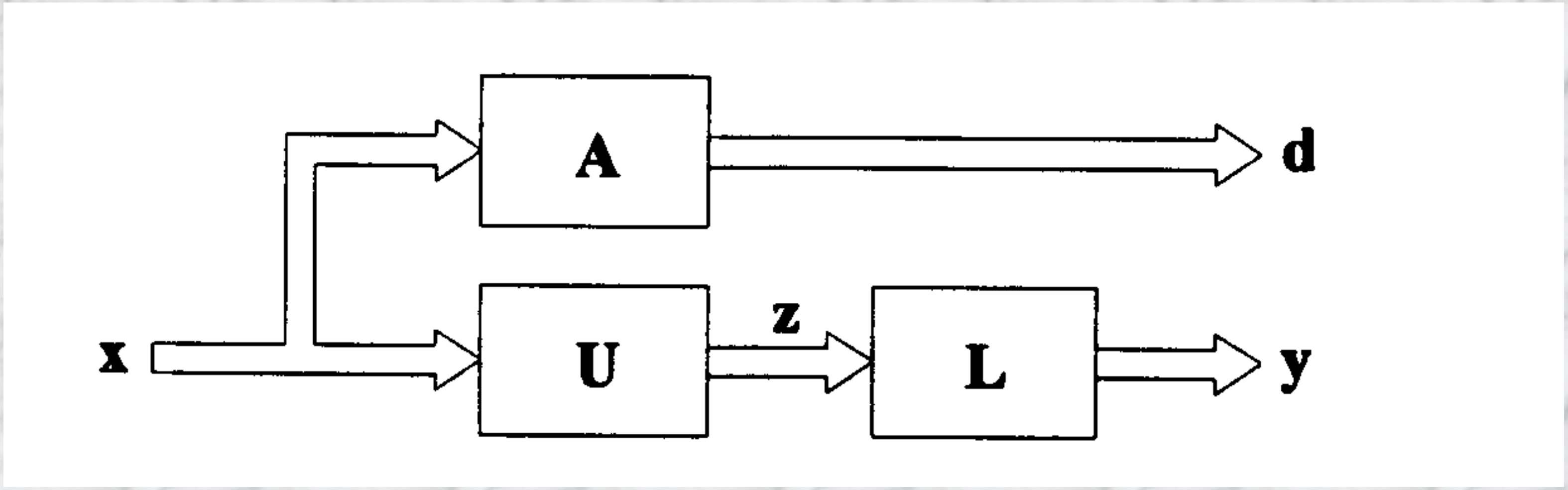
Neural Networks for Matrix Algebra Problems

The two-layer linear network is more complicated than the network for the matrix inversion or multiplication.

Here, both layers are the subject of learning procedure. The connection weights of the first layer are described by the matrix **U** and the second layer by the matrix **L**.

Neural Networks for Matrix Algebra Problems

A simplified block diagram



Neural Networks for Matrix Algebra Problems

The first layer performs a simple linear transformation **z** = **Ux**, where **x** is a given input vector. The second layer performs transformation **y = Lz = LUx.**

The parallel layer with weights defined by the matrix A elements is used to calculate the desired (target) output **d = Ax.**

Neural Networks for Matrix Algebra Problems

The weights l_{ii} are fixed and equal to unity, and proper elements of the matrices **L** and **U** are equal to zero.
To minimize the error function we will apply the simplified back-propagation algorithm.

Neural Networks for Matrix Algebra Problems

We get

$$l_{ij}(t+1) = l_{ij}(t) + \eta e_{ip} z_{ip}$$

for i > j, and

$$u_{ij}(t+1) = u_{ij}(t) + \eta \left[\sum_{h=1}^{n} l_{hi}(t+1) e_{hp} \right] x_{jp}$$

dla $i \leq j$

Neural Networks for Matrix Algebra Problems

where

$e_{ip} = d_{ip} - y_{ip}$ is the actual error of *i*-th output element for *p*-th pattern $\mathbf{x}_p$

$z_{ip} = \sum_{j=1}^{n} u_{ij} x_{jp}$ is the actual output of *i*-th element of the first layer for the same *p*-th pattern $\mathbf{x}_p$

and $y_{ip} = \sum_{j=1}^{i} l_{ij} z_{jp}$ $d_{ip} = \sum_{j=1}^{n} a_{ij} x_{jp}$

We’ll take a 5-minute break now

Cellular Neural Networks

Cellular Neural Networks

Cellular Neural Networks (CNN) are
a parallel computing paradigm similar to neural networks, with the difference that communication is allowed between neighboring units only.
CNN can be viewed as a special case of
a continuous-time Hopfield network.
It differs from the analog Hopfield network in its *local connectivity* property

Cellural Neural Networks – CNN

Cellural Neural Networks – CNN are built from identical nonlinear units called cells. It is a multi-input, dynamical system, and the behavior of the overall system is driven primarily through the weights of the processing unit's linear interconnect.

Cellular Neural Networks

From an architecture standpoint, CNN processors are a system of a finite, fixed-number, fixed-location, fixed-topology, locally interconnected, multiple-input, single-output, nonlinear processing units.
Cells are defined in a normed space, commonly a two-dimensional Euclidean geometry, like a grid.

Cellural Neural Networks – CNN

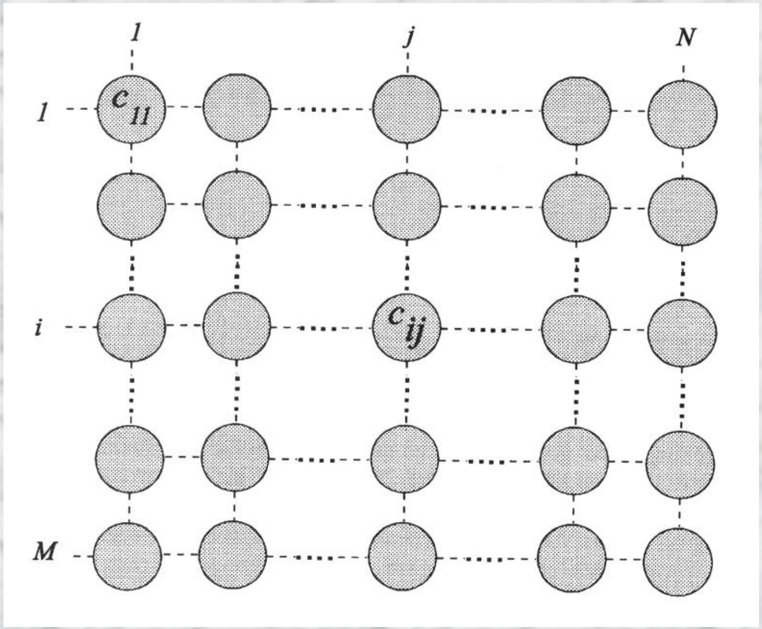
Cells are defined in a normed space, commonly a two-dimensional geometry, like a grid. The cells are not limited to two-dimensional spaces however; they can be defined in an arbitrary number of dimensions and can be square, triangle, hexagonal, or any other spatially invariant arrangement. Topologically, cells can be arranged on an infinite plane or on a toroidal space. Cell interconnect is local, meaning that all connections between cells are within a specified radius (with distance measured topologically). Connections can also be time-delayed to allow for processing in the temporal domain.

Network topology

Typically the two-dimensional network is organized in an eight-neighbour rectangular grid

Each cell c_{ij} situated in *i-th* row and *j-th* column interacts directly only with the cells within its *radius of neighbourhood r*.

When r = 1, which is a common assumption, the neighbourhood includes the cell itself and its eight nearest neighbouring cells



Network topology

A cell $c_{i+k,j+l}$ situated in $i+k$ row and $j+l$ column belongs to the cell neighbourhood, when

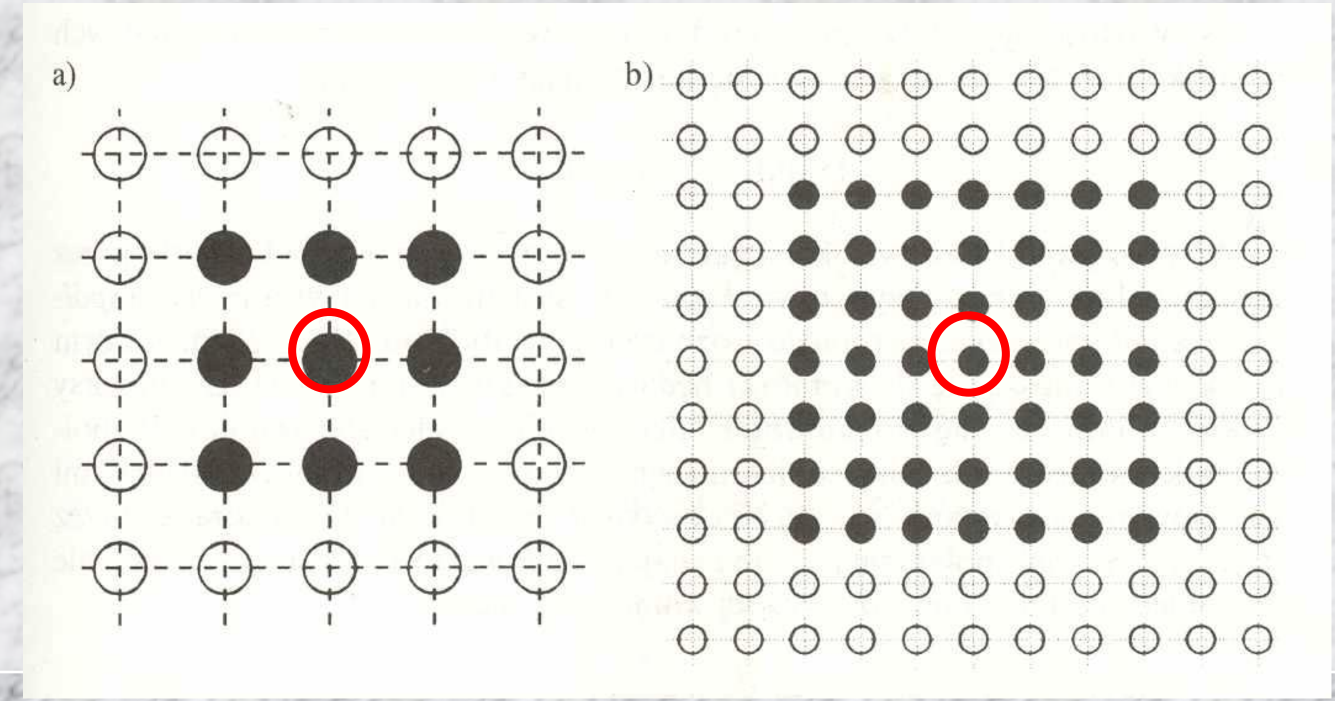
$$c_{i+k,\, j+l} \in N_r(i,\, j) \Leftrightarrow |k| \le r, |l| \le r$$

where r jest natural number called *radius of neighbourhood,* $N_r(i,j)$ denotes neighbourhood of the c_{ij} of the radius r.

Network topology

Part of the cellural neural network with the radius of neighbourhood

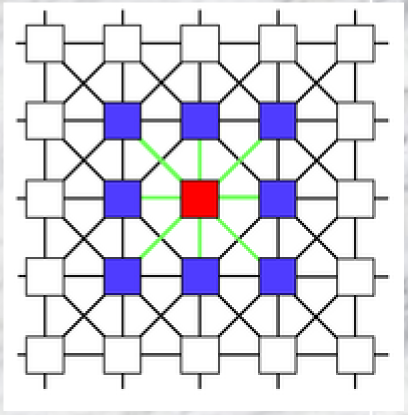
r = 1 r = 3



Network topology

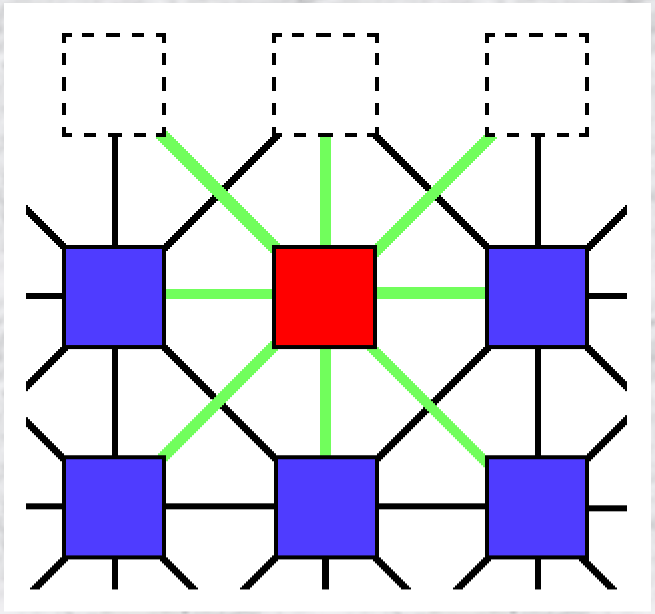
Part of the cellular neural network with the radius of neighbourhood   r = 1

The figure shows the emphasized cell (red) connected to the nearest neighbours (blue). The cells marked in grey represent the neighbourhood cells of the black cell. The neighbourhood includes the black cell itself.



Network topology

In order to calculate the state of the cells on the boundary, it is necessary to define the boundary conditions of the network, as shown in the figure.



Local connections of an *edge cell*. Observe that three of its neighbors are *boun-dary cells* (dashed).

Network topology

In the standard model, the boundary conditions can be:

- Fixed (or Dirichlet), if the value of the *boundary cells* is a prescribed constant;
- Zero-flux (or Neumann), if the value of the *boundary cells* is the same as the *edge cells*;
- Periodic (or toroidal), if the value of the *boundary cells* is the same as the *edge cells* on the opposite side (e.g., top *boundary cells* have the value of bottom *edge cells*).

Network operation

All cells are transforming signals the same way, generating **output signal**

The output signal y_{ij} depends from **cells' state** x_{ij}.

Cell state is determined by the integration of the sum of cell **control signals** multiplied by the proper coefficients

Network operation

Control signals:

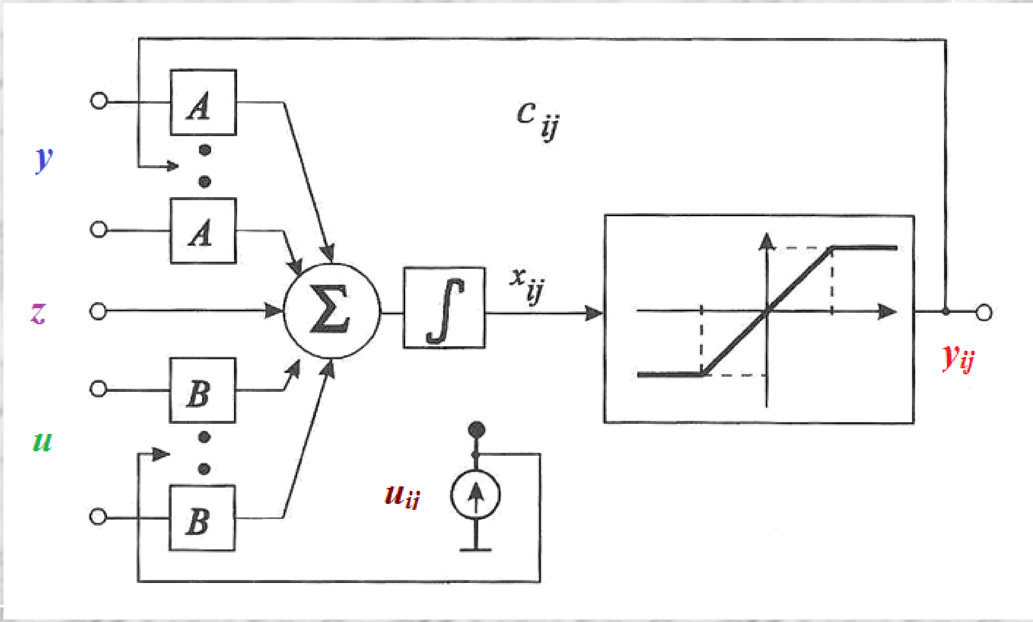
a) output signals $y_{i+k,j+l}$ the cells in the neighborhood

b) own output signal of the c_{ij} cell y_{ij}

c) input signals $u_{i+k,j+l}$ the cells in the neighborhood

d) owninput signal of the cell c_{ij}

e) bias z



Network operation

For input signals u_{ij} and initial conditions of a cell state x_{ij} for $t=0$, following conditions are imposed

$$\left|x_{ij}(0)\right| \le 1, \qquad \left|u_{ij}(t)\right| \le 1$$

Network operation

Weights of the output signals are called feedback operators and denoted

$$A_{i,j;i+k,j+l}$$

subscripts i,j refer to the cell c_{ij} being controlled
subscripts $i+k,j+l$ refer to the cells controlling the cell c_{ij}

Network operation

Weights of input signals are called control operators and denoted

$$B_{i,j;i+k,j+l}$$

where

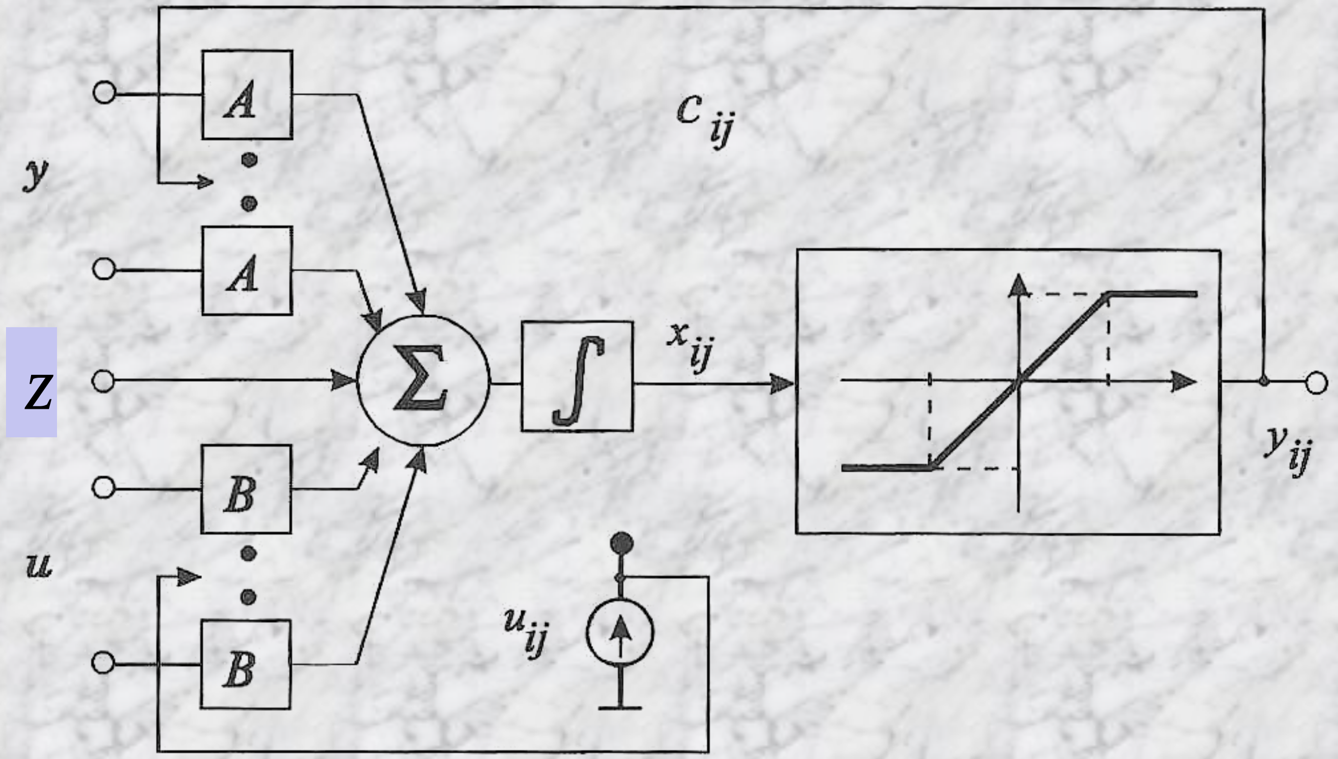
subscripts i,j refer to the cell c_{ij} being controlled

subscripts $i+k,j+l$ refer to the cells controlling the cell c_{ij}.

Bias signal z has usually constant value but not necessary identical for every cell in the network.

Cell

Block diagram of a single cell



CNN dynamics

The CNN dynamics is described by a system of nonlinear differential equations. Using the simplest first-order cell dynamics and linear interactions, the state equation of a cell in position *(i,j)* is as follows

$$C\frac{dx_{ij}}{dt} = -\frac{x_{ij}}{R_x} + \sum_{k=-r}^{r}\sum_{l=-r}^{r} A_{ij;i+k,j+l}\, y_{i+k,j+l} +$$

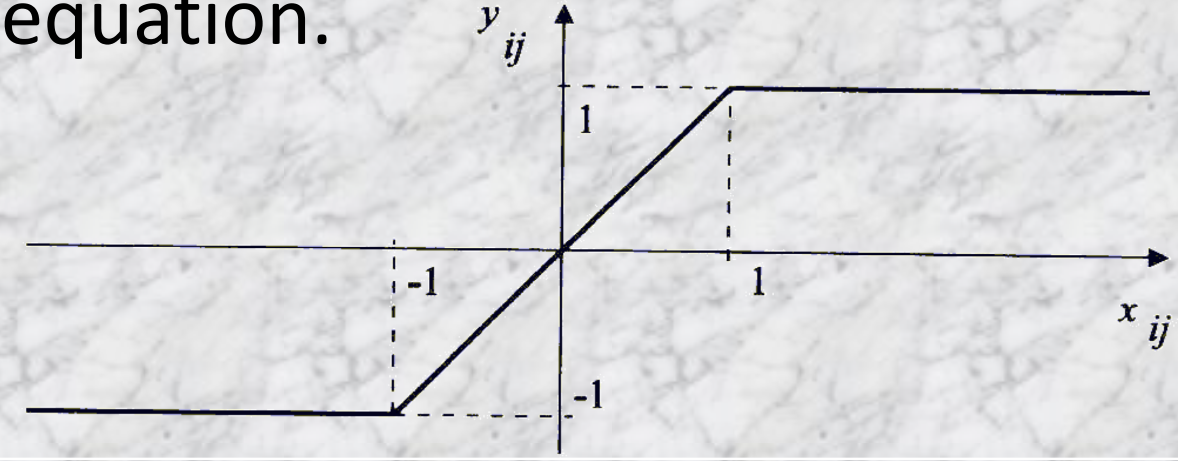
$$+ \sum_{k=-r}^{r}\sum_{l=-r}^{r} B_{ij;i+k,j+l}\, u_{i+k,j+l} + z$$

CNN dynamics

The expression for the output y_{ij} defined by the piece-wise linear function is

$$y_{ij} = \mathrm{f}(x_{ij}) = 0{,}5 * (|x_{ij} + 1| - |x_{ij} - 1|)$$

Where f(*) is the standard nonlinearity for the output equation.



Network operation

The set of matrices *{A,B}*, which contains the weights of the network, is called the *cloning template* and it defines the operation performed by the network. When the values of the *cloning template* do not depend on the position of the cell, the CNN is called *space-invariant*.

$A_{ij;i+k,j+l} = A_{IJ;I+k,J+l}$ and $B_{ij;i+k,j+l} = B_{IJ;I+k,J+l}$

where $0 \leq i, I, i+k, I+k < M$

$0 \leq j, J, j+l, J+l < N$

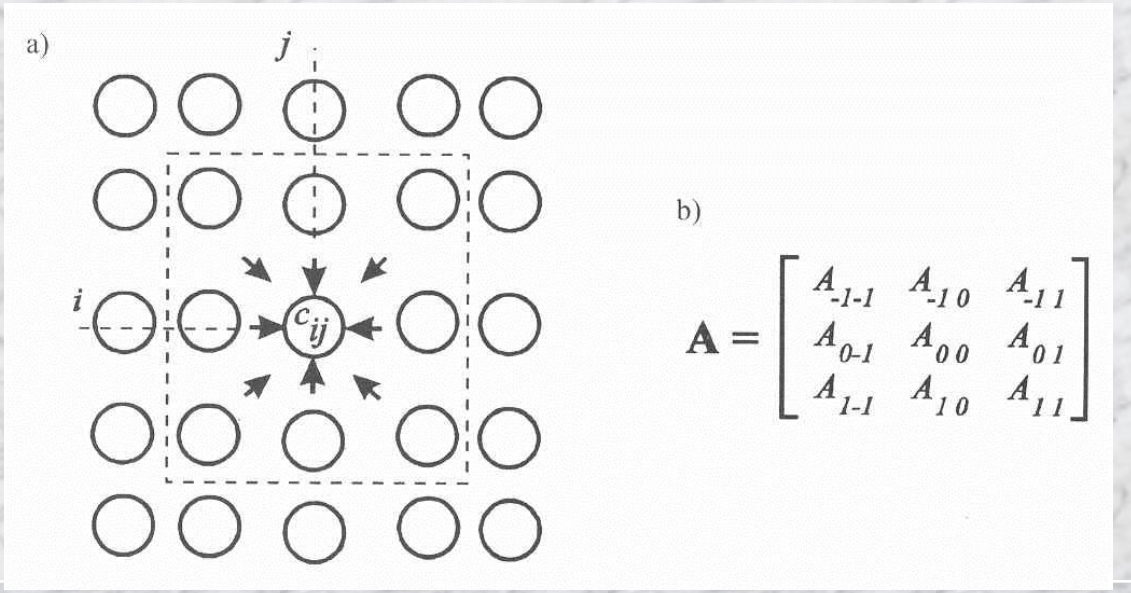
Network operation

We can neglect subscripts ij and IJ leaving only kl, where $-r \leq k,l \leq r$.
Weights A_{kl} i B_{kl} are usually represented in a matrix form **A** (**feedback operator**) and **B** (**control operator**) with dimension $(2r+1)x(2r+1)$.
The central element of neighborhood has the indexes $k = 0$ and $l = 0$, and A_{00} denotes power of the cell c_{ij} self-control.

Network operation

A_{kl} defines the power of control by a cell distant by l columns and k rows.

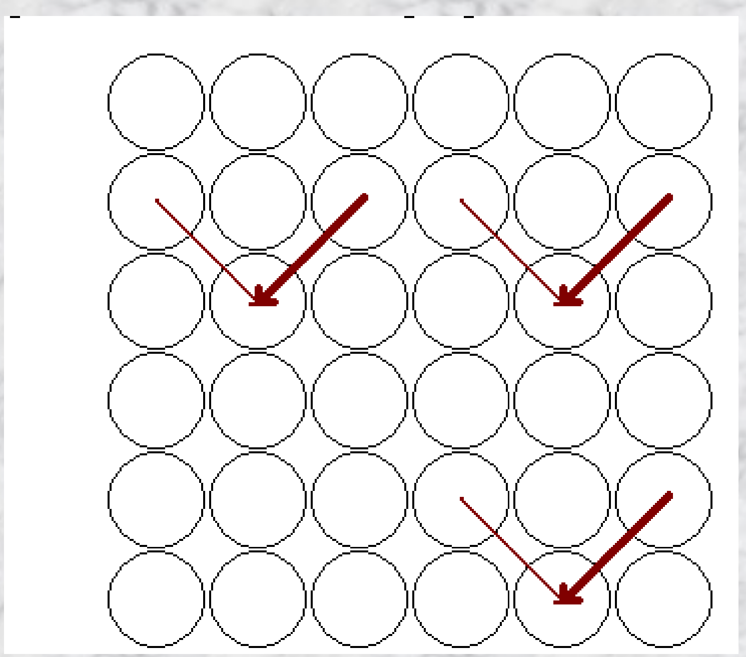
Network fragment for $r = 1$ and matrix **A**

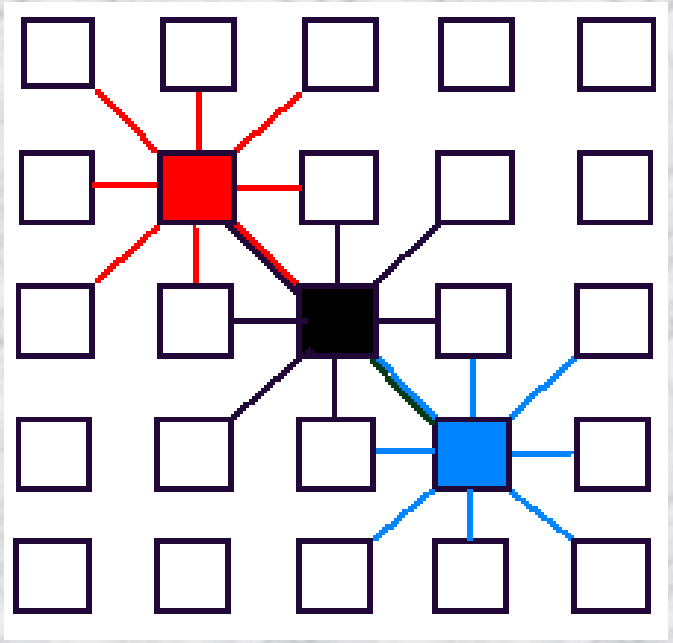


$$\mathbf{A} = \begin{bmatrix} A_{-1-1} & A_{-1\,0} & A_{-1\,1} \\ A_{0-1} & A_{0\,0} & A_{0\,1} \\ A_{1-1} & A_{1\,0} & A_{1\,1} \end{bmatrix}$$

Network operation

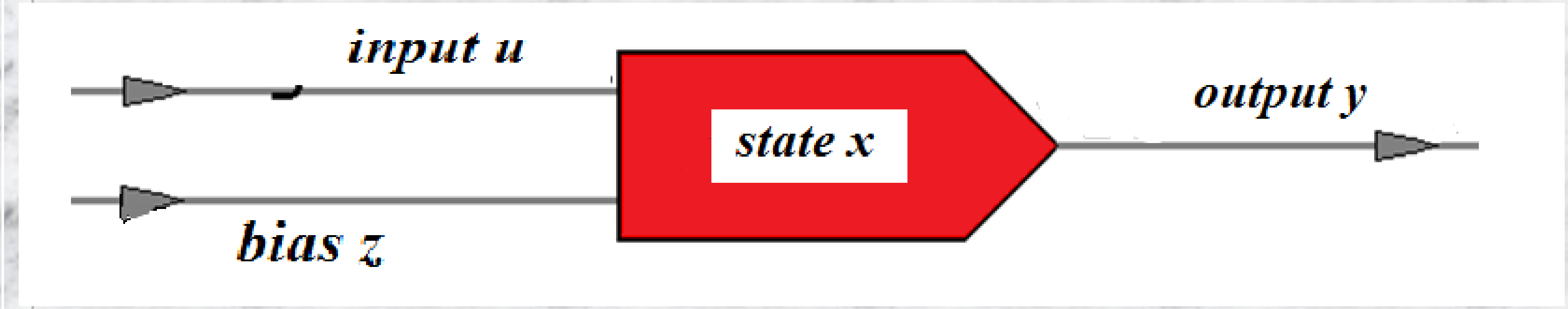
The template cloning is identical for every cell





Network operation

The simplified cell



output $y(t) = \mathrm{f}(x(t), u(t), z)$

state $x(t+1) = \mathrm{g}(x(t), u(t), z)$

$x, y, u, z \in \mathbf{R}$

Network operation

To describe the network structure it is enough to define the cloning template:

- bias term ***z*** (usually identical for each cell),
- control operator **A**, describing the weights in the feedback connections,
- feedback operator **B**, describing the feedforward connections,
- initial conditions.

To start the calculations it is necessary to define:

- initial state ***x*** for each cell,
- input signal ***u*** for each cell.

Network operation

The g function is define by:

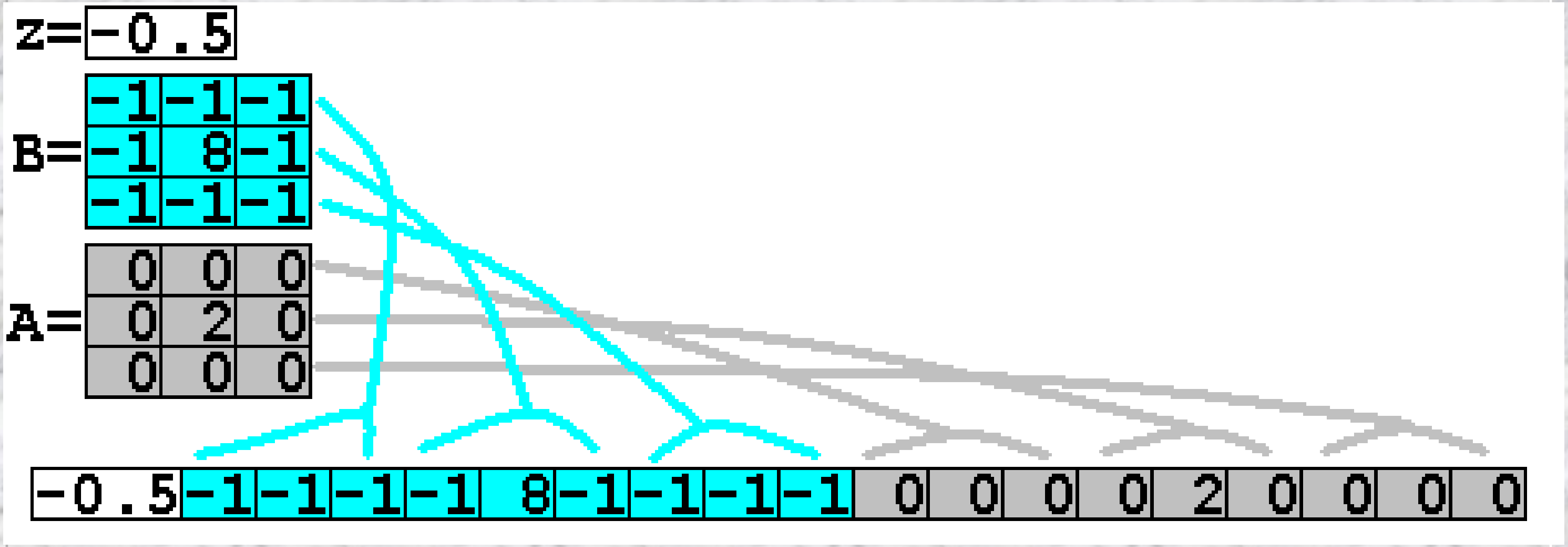
$$x_{ij}(t+1) = x_{ij}(t) + \sum_{k=-r}^{r}\sum_{l=-r}^{r} A_{ij;i+k,j+l}\, y_{i+k,j+l} +$$

$$+ \sum_{k=-r}^{r}\sum_{l=-r}^{r} B_{ij;i+k,j+l}\, u_{i+k,j+l} + z$$

Weights A and B are invariable and their selection depends on the problem to be solved.

Representation

- Matrix and vector (genotype) notation of the weights



$z = -0.5$

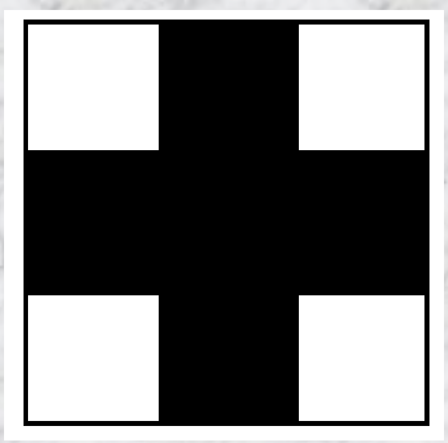
$$B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

-0.5	-1	-1	-1	-1	8	-1	-1	-1	-1	0	0	0	0	2	0	0	0	0

Representation

Detection of composed binary pattern in the input.

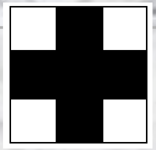


Representation

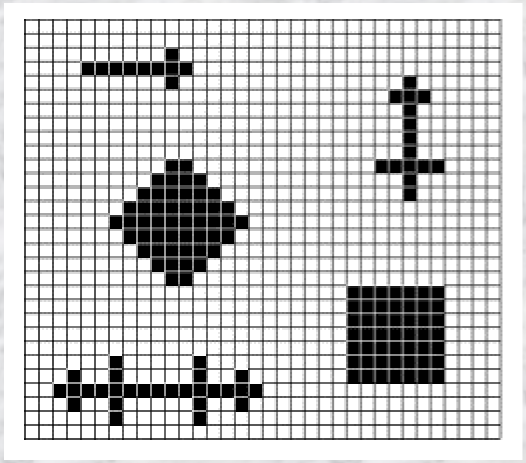
Rectangular CNN grid, weight matrices and bias

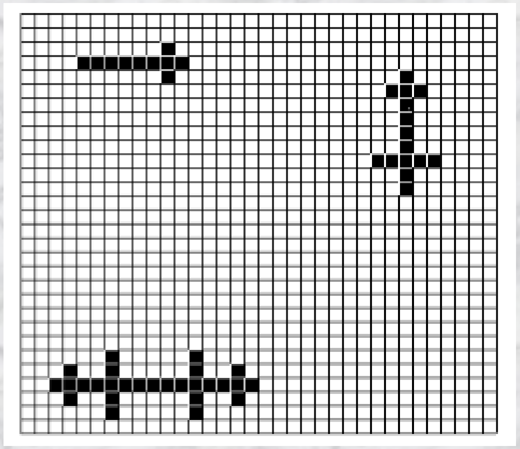
$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3{,}7 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} -0{,}1 & 0{,}1 & -0{,}1 \\ 0{,}1 & 10{,}0 & 0{,}1 \\ -0{,}1 & 0{,}1 & -0{,}1 \end{bmatrix} \qquad z = 0$$

Representation

Extraction of the shapes containing 

Input pattern Output after processing





Applications

Cellular neural networks are used in many areas, but mainly in:

- image processing
- feature extraction
- modeling physical phenomena

Applications

But also:

missile tracking, flash detection, level and gain adjustments, color constancy detection, contrast enhancement, deconvolution, image compression, motion estimation, image encoding, image decoding, image segmentation, orientation preference maps, pattern learning/recognition, multi-target tracking, image stabilization, resolution enhancement, image deformations and mapping, image inpainting, optical flow, contouring, moving object detection, axis of symmetry detection, and image fusion.

Applications

- Are relatively simple for realization (small number of connections).
- Are able to perform parallel and distributed processing which yields to high computational power.

Multi-Layer CNN

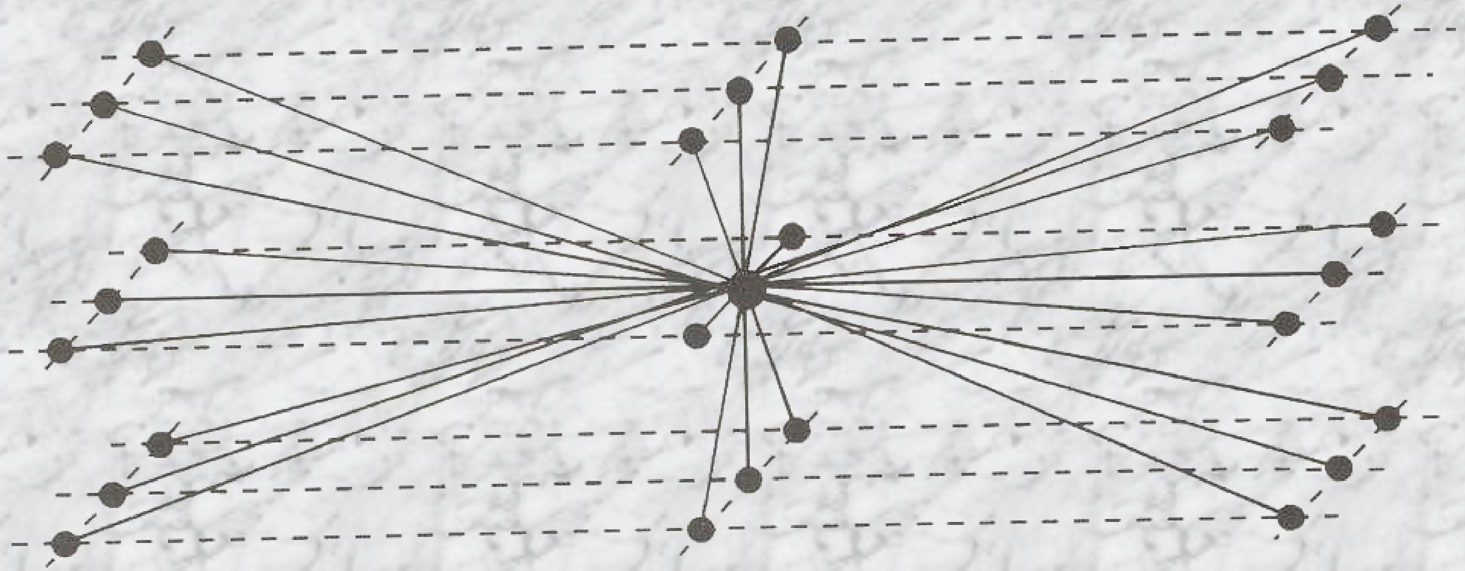
In multi- layer structure instead of scalar values :

- state x_{ij},
- output signal y_{ij},
- input signal u_{ij},

we have vectorial: $\boldsymbol{X}_{ij}$, $\boldsymbol{Y}_{ij}$ i $\boldsymbol{U}_{ij}$.

Multi-Layer CNN

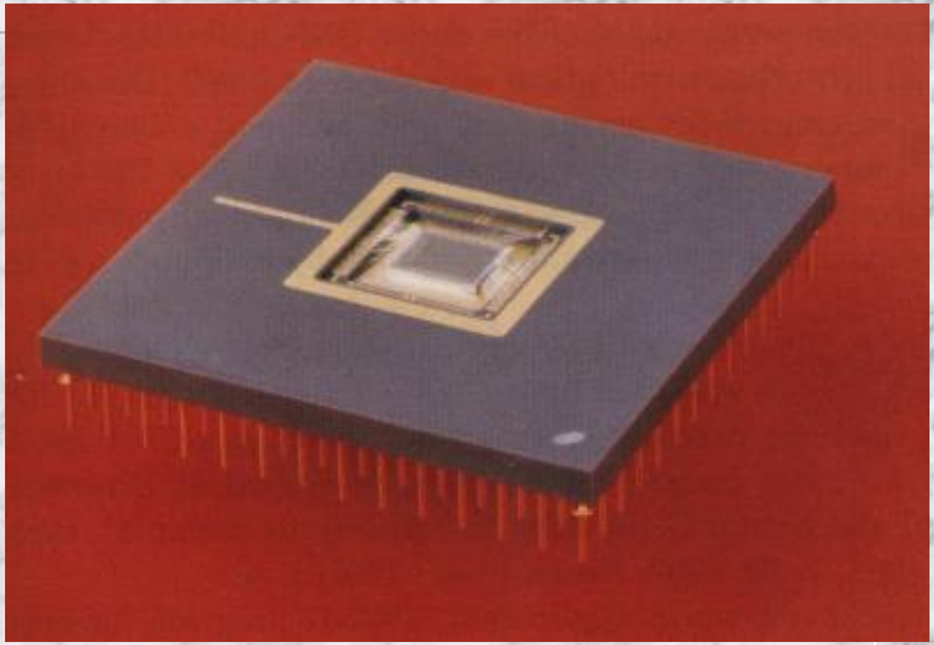
Example of connections of single cell in the middle layer of three layer cellular network of radius r = 1



source: T.Kacprzak, K.Ślot, *Sieci neuronowe komórkowe*, PWN 1995.

Technical realisation

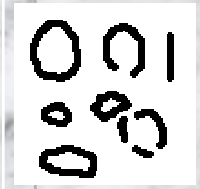
- 5.5 mm x 4.7 mm
- 0.3 W
- 60 000 transistors
- 16 x 16 cells
- Programmable weights



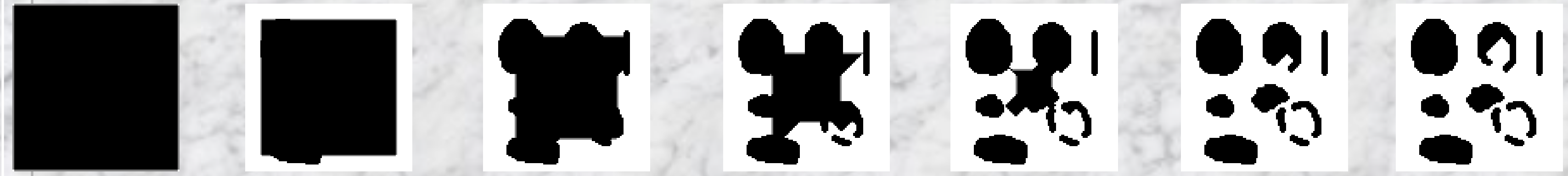
For 100 x 100 cell expected calculation speed **10^{12}** instructions per second

Example

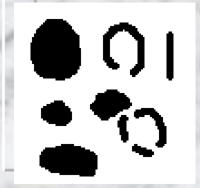
Filling of closed region



input signal (100 x 100)



t = 0 t = 10 t = 20 t = 30 t = 40 t = 50 t = 60



final output signal, t = 71

Potential use

Linear templates B 3x3, A 1x1, binary patterns

image segmentation	separate points removal	one-side edge detection
vertical translation	NOT operation	corner detection
horizontal translation	AND operation	image erosion
diagonal translation	OR operation	image fusion
separate points remaining	edge detection	image compression

Potential use

Linear templates B 3x3, A 3x3, binary patterns

shadow casting in define direction	lines detection parallel to diagonal
vertical gap detection	detection of fully filled objects
diagonal gap detection	filling of closed shapes
separate points remaining	edge detection

Potential use

Linear templates B 3x3, A 3x3, gray-levels patterns

half-toning
inverse half-toning
texture extraction of similar shade

Nonlinear templates B 3x3, A 3x3, binary patterns

histogram creation
pixel-wise parity detection (XOR for neighbor pixels)
row-wise parity detection

Potential use

Nonlinear templates B 3x3, A 3x3, gray-levels patterns

contour detecton
region detection of similar gray-levels

Potential use

Linear templates B 3x3, A 3x3, with time-delay

object movement detection (8 basic directions)
object movement detection with filtering of stationary objects
object movement detection with predefined speed or slower

Linear templates B 5x5, A 5x5

de-blurring

CNNs scientific applications

- feature extraction & classification
- motion detection & estimation
- collision avoidance
- object counting & size estimation
- path tracking
- detecting minima and maxima
- detecting area with gradients that exceed a given threshold
- thermographic maps
- antenna-array images
- medical maps and images

References

- Chua, Leon O. and Yang, Lin (1988). *Cellular neural networks: theory*. IEEE Transactions on Circuits and Systems 35(10): 1257-1272.
- Chua, Leon O. and Roska, Tamas (1993). *The CNN paradigm*. IEEE Transactions on Circuits and Systems 40(3): 147-156.
- L. O. Chua, *CNN: A paradigm for complexity*, World Scientific 1998.