

# Rețele neuronale recurente

- Arhitectura
  - Rețele cu interconectivitate totală
  - Rețele cu unități contextuale
  - Rețele celulare
- Dinamica rețelelor recurente
  - Dinamica în timp continuu
  - Dinamica în timp discret
  - Proprietăți de stabilitate
- Rețele celulare
- Alte clase de rețele recurente
- Aplicații

# Rețele neuronale recurente

- Arhitectura
  - Se caracterizează prin prezența conexiunilor inverse (graful suport conține cicluri)
  - In funcție de densitatea conexiunilor inverse există:
    - Rețele total recurente (modelul Hopfield)
    - Rețele parțial recurente:
      - Cu unități contextuale (modelul Elman)
      - Celulare (modelul Chua-Yang)
- Aplicabilitate
  - Memorare asociativă
  - Rezolvarea problemelor de optimizare combinatorială
  - Predicție în serii temporale
  - Prelucrarea imaginilor
  - Modelarea sistemelor dinamice și a fenomenelor haotice

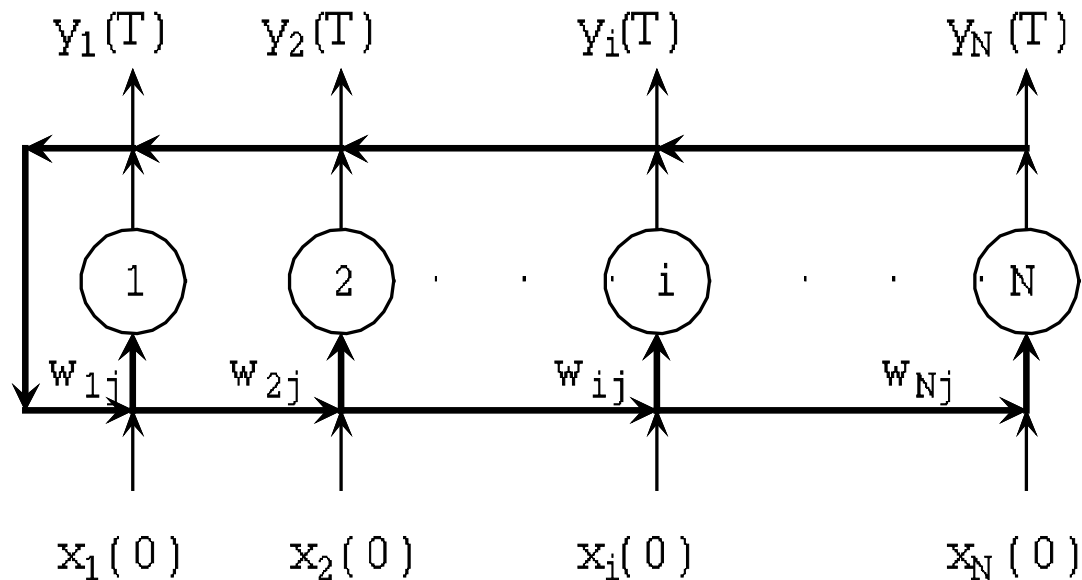
# Rețele de tip Hopfield

Arhitectura: N unități total  
interconectate

Funcții de transfer:  
Signum/Heaviside  
Logistica/Tanh

Parametrii:

matricea cu ponderi a  
conexiunilor



Notatii:  $x_i(t)$  – potențialul (starea) neuronului  $i$  la momentul  $t$   
 $y_i(t)=f(x_i(t))$  – semnalul de ieșire produs de neuronul  $i$  la momentul  $t$   
 $I_i(t)$  – semnalul de intrare primit din partea mediului  
 $w_{ij}$  – ponderea conexiunii dintre neuronul  $j$  și neuronul  $i$

# Rețele de tip Hopfield

- Funcționare:
- semnalul de ieșire este rezultatul unui proces dinamic
  - rețelele de tip Hopfield pot fi asimilate unor sisteme dinamice a căror stare se modifică în timp

## Starea rețelei:

- vectorul stărilor neuronilor  $X(t)=(x_1(t), \dots, x_N(t))$

sau

- vectorul semnalelor de ieșire  $Y(t)=(y_1(t), \dots, y_N(t))$

## Descrierea dinamicii

- Timp discret – relații de recurență (ecuații cu diferențe)
- Timp continuu – ecuații diferențiale

# Rețele de tip Hopfield

Funcționare în timp discret:

starea rețelei la momentul  $t+1$  este descrisă în funcție de starea rețelei la momentul  $t$

Starea rețelei:  $Y(t)$

Variante:

- **Asincronă:** un singur neuron își poate modifica starea la un moment dat
- **Sincronă:** toți neuronii își schimbă starea simultan

Răspunsul rețelei: starea în care se stabilizează rețeaua

# Retele de tip Hopfield

Varianta asincronă:

$$y_{i^*}(t+1) = f\left(\sum_{j=1}^N w_{i^*j} y_j(t) + I_{i^*}(t)\right)$$

$$y_i(t+1) = y_i(t), \quad i \neq i^*$$

Alegerea lui  $i^*$ :

- prin parcurgerea sistematică a mulțimii  $\{1, 2, \dots, N\}$
- aleator (dar astfel încât la fiecare etapă să se modifice starea fiecărui neuron)

Simularea rețelei:

- alegerea unei stări inițiale (în funcție de problema de rezolvat)
- calculul stării următoare până când rețeaua se stabilizează (distanța dintre două stări succesive este mai mică decât o valoare prestabilită,  $\varepsilon$ )

# Retele de tip Hopfield

Varianta sincronă:

$$y_i(t+1) = f\left(\sum_{j=1}^N w_{ij} y_j(t) + I_i(t)\right), \quad i = \overline{1, N}$$

In simulări se utilizează fie funcții de transfer discontinue fie continue

Algoritm de funcționare:

Stabilire stare inițială

REPEAT

    se calculează noua stare pornind de la starea curentă

UNTIL < diferența dintre starea curentă și cea anterioară este  
    suficient de mică >

# Rețele de tip Hopfield

Funcționare in timp continuu:

$$\frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t), \quad i = \overline{1, N}$$

**Simularea rețelei:** rezolvarea numerică a sistemului de ecuații pornind de la o valoare inițială  $x_i(0)$

**Exemplu:** metoda lui Euler explicită

$$\frac{x_i(t+h) - x_i(t)}{h} \cong -x_i(t) + \sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t), \quad i = \overline{1, N}$$

$$x_i(t+h) \cong (1-h)x_i(t) + h \sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t), \quad i = \overline{1, N}$$

Semnal de intrare constant :

$$x_i^{nou} \cong (1-h)x_i^{vechi} + h \left( \sum_{j=1}^N w_{ij} f(x_j^{vechi}) + I_i \right), \quad i = \overline{1, N}$$



# Proprietăți de stabilitate

## Comportări posibile ale unei rețele:

- $X(t)$  tinde către o stare staționară  $X^*$  (punct fix al dinamicii rețelei)
- $X(t)$  oscilează între două sau mai multe stări posibile
- $X(t)$  are o comportare haotică sau  $\|X(t)\|$  crește pe măsură ce  $t$  crește

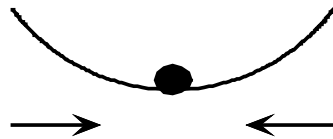
## Comportări utile din punct de vedere practic:

- **Rețeaua tinde către o stare staționară**
  - Există o mulțime de stări staționare: memorare asociativă
  - Stare staționară unică: rezolvarea pb. de optimizare
- **Rețeaua are comportare periodică**
  - Modelarea seriilor temporale

**Obs.** Cea mai convenabilă variantă este cea în care rețeaua tinde către o stare staționară stabilă (perturbații mici în condițiile inițiale nu afectează semnificativ soluția)

# Proprietăți de stabilitate

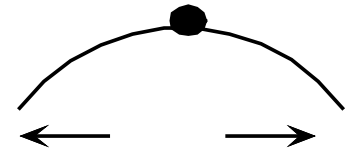
Tipuri de echilibru  
(intuitiv)



Asimptotic stabil



Stabil



Instabil

Formalizare:

$$\frac{dX(t)}{dt} = F(X(t)), \quad X(0) = X_0$$

$$F(X^*) = 0$$

$X^*$  este soluție staționară asimptotic stabilă (în raport cu condițiile inițiale) dacă:

stabilă

atractivă

# Proprietăți de stabilitate

## Stabilitate:

$X^*$  este stabilă dacă pentru orice  $\varepsilon > 0$  există  $\delta(\varepsilon) > 0$  cu proprietățile:  $\|X_0 - X^*\| < \delta(\varepsilon)$  implică  $\|X(t; X_0) - X^*\| < \varepsilon$

## Atractivitate:

$X^*$  este atractivă dacă există  $\delta > 0$  cu proprietatea:  $\|X_0 - X^*\| < \delta$  implică  $X(t; X_0) \rightarrow X^*$

Pentru a studia proprietatea de (asimptotic) stabilitate a soluțiilor în teoria sistemelor dinamice se poate folosi metoda funcției Liapunov

# Proprietati de stabilitate

Funcție Liapunov:

$V : R^N \rightarrow R$ , marginita inferior

$$\frac{dV(X(t))}{dt} < 0, \text{ pentru orice } t > 0$$

- Funcție mărginită inferior care descrește de-a lungul soluțiilor => soluția trebuie să se stabilizeze
- Dacă unui sistem  $i$  se poate asocia o funcție Liapunov atunci soluțiile staționare ale acestuia sunt asimptotic stabile
- Funcțiile Liapunov sunt similare funcțiilor de energie din fizică (sistemele fizice tind în mod natural către stări de energie minimă)
- Stările pentru care funcția Liapunov este minimă corespund stărilor de energie minimă, care sunt stări stabile
- Dacă rețeaua neuronală satisface anumite proprietăți atunci există funcție Liapunov asociată

# Proprietăți de stabilitate

Rezultate de stabilitate pentru rețelele neuronale (cazul continuu):

Dacă:

- matricea ponderilor este simetrică ( $w_{ij}=w_{ji}$ )
- funcția de transfer este strict crescătoare ( $f'(u)>0$ )
- semnalul de intrare este constant ( $I(t)=I$ )

Atunci toate soluțiile staționare asociate rețelei sunt asimptotic stabile

Funcția Liapunov asociată:

$$V(x_1, \dots, x_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij} f(x_i) f(x_j) - \sum_{i=1}^N f(x_i) I_i + \sum_{i=1}^N \int_0^{f(x_i)} f^{-1}(z) dz$$

# Proprietăți de stabilitate

Rezultate de stabilitate pentru rețelele neuronale (cazul discret - asincron):

Dacă:

- matricea ponderilor este simetrică ( $w_{ij}=w_{ji}$ )
- funcția de transfer este signum sau Heaviside
- semnalul de intrare este constant ( $I(t)=I$ )

Atunci toate soluțiile staționare asociate rețelei sunt asimptotic stabile

Funcția Liapunov asociată

$$V(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij} y_i y_j - \sum_{i=1}^N y_i I_i$$

# Proprietăți de stabilitate

Rezultatul anterior sugerează că stările staționare (punctele fixe) ale dinamicii sistemului sunt stabile

Fiecarei stări staționare îi corespunde o regiune de atracție (dacă starea inițială a rețelei se află în regiunea de atracție a unei stări staționare atunci sistemul va tinde către starea staționară corespunzătoare

Această proprietate este utilă în cazul memoriilor asociative

În cazul dinamicii sincrone rezultatul nu mai este valabil, putându-se arăta în schimb că soluțiile periodice de perioadă 2 (rețeaua oscilează între două stări) sunt stabile

# Memorii asociative

**Memorie** = sistem de stocare și regăsire a informației

**Memorie bazată pe adresă:**

- Stocare localizată: toate “elementele” (biții) unei unități de informație (ex: o valoare numerică) sunt stocate în aceeași zonă de memorie identificabilă prin adresă
- Regăsire pe baza adresei

**Memorie asociativă:**

- Informația este stocată distribuit (nu mai există conceptul de adresă)
- Regăsirea se realizează pe baza conținutului (se pornește de la un indiciu privind informația de regăsit – ex: imagine parțială sau alterată de zgomot)



# Memorii asociative

## Proprietăți:

- Robustețe (distrugerea unei componente nu împiedică regăsirea informației)

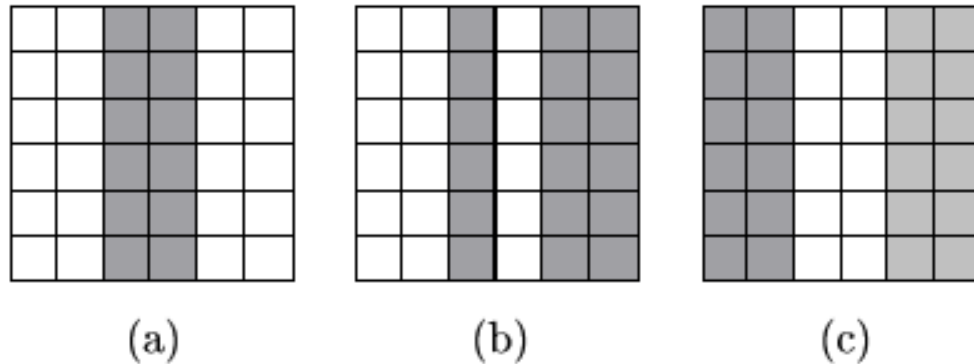
## Modalități de implementare:

- Hardware:
  - Circuite electrice
  - Sisteme optice
- Software:
  - Rețea de tip Hopfield

# Memorii asociative

Simularea software a memoriilor asociative:

- Informațiile sunt codificate binar: vectori cu elemente din  $\{-1,1\}$
- Fiecarei componente a vectorului  $i$  va corespunde o unitate funcțională în cadrul rețelei



Exemplu (a)

$(-1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1)$

# Memorii asociative

Proiectarea rețelelor pentru simularea software a memoriilor asociative:

- Rețea total interconectată cu  $N$  unități de tip signum ( $N$  este dimensiunea șabloanelor ce urmează a fi memorate)

Stocarea informației:

- Stabilirea ponderilor (elementele matricii  $W$ ) astfel încât șabloanele de memorat să devină puncte fixe (stări staționare) ale dinamicii rețelei

Regăsirea informației:

- Inițializarea stării rețelei cu indiciul de start și simularea dinamicii rețelei până la identificarea stării staționare corespunzătoare; starea staționară va reprezenta informația regăsită

# Memorii asociative

Set de vectori de memorat:  $\{X^1, \dots, X^L\}$ ,  $X^l$  din  $\{-1, 1\}^N$

Metode de stocare:

- Regula Hebb
- Metoda pseudo-inversei (algoritmul Diedierich – Opper)

Regula Hebb:

- Se bazează pe principiul enunțat de D. Hebb: “permeabilitatea sinaptică a doi neuroni activați simultan crește”

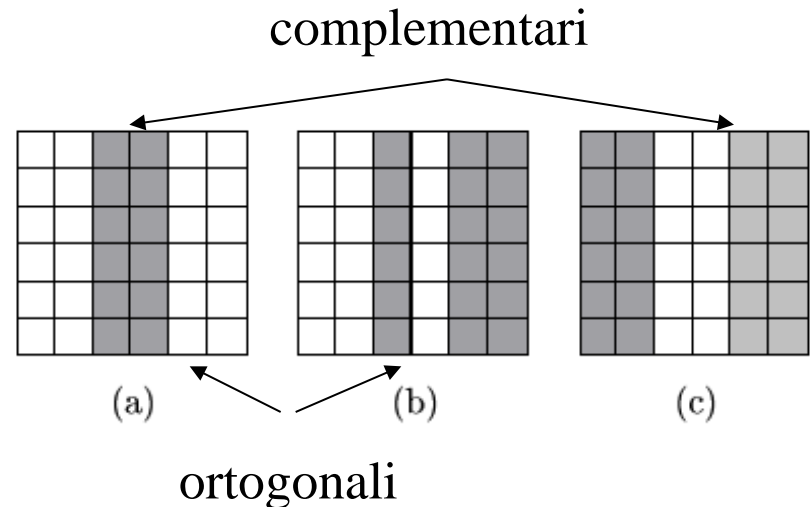
$$w_{ij} = \frac{1}{N} \sum_{l=1}^L x_i^l x_j^l$$

# Memorii asociative

$$w_{ij} = \frac{1}{N} \sum_{l=1}^L x_i^l x_j^l$$

## Proprietăți ale regulii Hebb:

- Dacă vectorii din setul de memorat sunt **ortogonali doi câte doi** (din punct de vedere statistic ar însemna că sunt necorelați) atunci toți sunt puncte fixe ale dinamicii rețelei
- O dată cu stocarea vectorului  $X$  se stochează și opusul său:  $-X$
- O variantă care elimină condiția de ortogonalitate este cea bazată pe metoda pseudo-inversei



# Memorii asociative

Metoda pseudo-inversei:

$$w_{ij} = \frac{1}{N} \sum_{l,k} x_i^l (Q^{-1})_{lk} x_j^l$$

$$Q_{lk} = \frac{1}{N} \sum_{i=1}^N x_i^l x_i^k$$

- Se poate arăta că dacă matricea  $Q$  este inversabilă atunci toate elementele lui  $\{X^1, \dots, X^L\}$  sunt puncte fixe ale dinamicii rețelei
- Pentru a evita calculul inversei matricii  $Q$  se poate folosi un algoritm iterativ de ajustare a ponderilor

# Memorii asociative

Algoritmul Diederich-Opper:

---

$t := 0$

Se inițializează matricea  $W(0)$  folosind regula Hebb

REPEAT

FOR  $l := 1, L$  DO

$$y_i^l := \sum_{j=1}^N w_{ij}(t) x_j^l, \quad i = \overline{1, N}$$

$$w_{ij}(t+1) := w_{ij}(t) + \frac{1}{N} (x_i^l - y_i^l) x_j^l, \quad i = \overline{1, N}, j = \overline{1, N}$$

$t := t + 1$

UNTIL  $\|W(t) - W(t-1)\| < \epsilon$

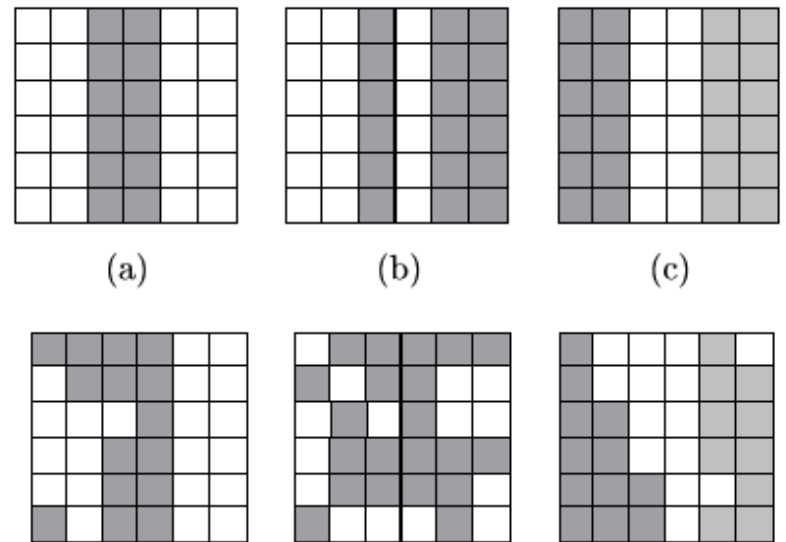
---

# Memorii asociative

## Regăsirea informației:

- Inițializarea stării rețelei cu indiciul de la care se pornește
- Simularea funcționării rețelei până la atingerea stării staționare

## Vectori stocați



## Vectori alterați de zgomot (25%, 50%, 75%)



# Memorii asociative

## Capacitatea de stocare:

- numărul de vectori care pot fi stocați și regăsiți (în limita unei anumite erori)
- Regăsire exactă: capacitate =  $N/(4\ln N)$
- Regăsire aproximativă ( $\text{prob}(\text{eroare})=0.005$ ): capacitate =  $0.15*N$

## Atratori paraziți:

- Stări staționare ale rețelei care nu au fost explicit stocate ci au rezultat ca urmare a stocării altor vectori

## Evitare atratori paraziți

- Modificarea metodei de stocare
- Introducerea de perturbații aleatoare în dinamica rețelei

# Formulara neuronală a unei probleme de optimizare combinatorială

- Prima abordare: Hopfield & Tank (1985)
  - Propun utilizarea unei rețele total interconectate pentru rezolvarea problemei comis-voiajorului
  - Ideea constă în a proiecta o rețea a cărei **funcție de energie** să fie similară funcției obiectiv a problemei de rezolvat (lungimea unui circuit hamiltonian) și în a o lăsa să **“evolueze”** în mod natural către **starea de energie minimă**; aceasta ar reprezenta soluția problemei

# Formulara neuronală a unei probleme de optimizare combinatorială

## Problema de optimizare:

determină  $(y_1, \dots, y_N)$  cu proprietățile:

minimizează o funcție cost  $C: \mathbb{R}^N \rightarrow \mathbb{R}$

satisface restricții de forma  $R_k(y_1, \dots, y_N) = 0$  cu  $R_k$  funcții nenegative

## Etape:

- Transformarea problemei într-una de optimizare fără restricții (metoda penalizării)
- Reorganizarea funcției cost ca o funcție Liapunov
- Identificarea valorilor parametrilor și a semnalelor de intrare pornind de la forma funcției Liapunov
- Simularea rețelei

# Formulara neuronală a unei probleme de optimizare combinatorială

Transformarea problemei într-una de optimizare fără restricții  
(metoda penalizării)

$$C^*(y_1, \dots, y_N) = aC(y_1, \dots, y_N) + \sum_{k=1}^r b_k R_k(y_1, \dots, y_N)$$

$$a, b_k > 0$$

Valorile coeficienților  $a$  și  $b_k$  se aleg astfel încât să reflecte importanța relativă a funcției obiectiv și a restricțiilor

# Formulara neuronală a unei probleme de optimizare combinatorială

Reorganizarea funcției cost ca o funcție Liapunov

$$C(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij}^{obj} y_i y_j - \sum_{i=1}^N I_i^{obj} y_i$$

$$R_k(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij}^k y_i y_j - \sum_{i=1}^N I_i^k y_i, \quad k = \overline{1, r}$$

**Observatie:** Ideea funcționează doar dacă funcția obiectiv și restricțiile sunt de grad cel mult 2

# Formulara neuronală a unei probleme de optimizare combinatorială

Identificarea valorilor parametrilor si a semnalelor de intrare pornind de la forma funcției Liapunov

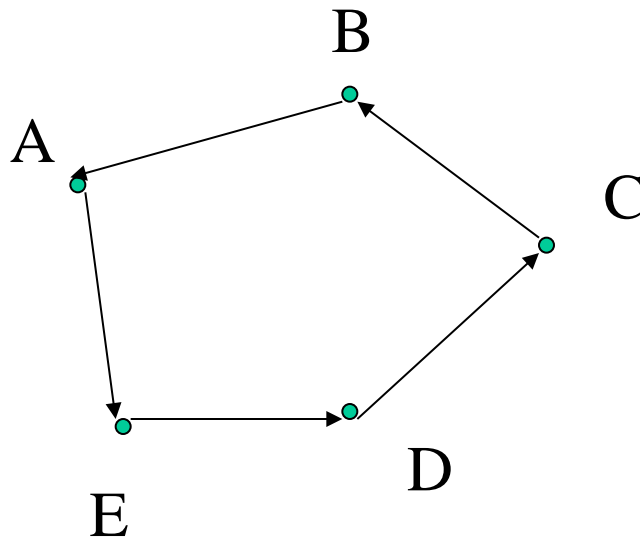
$$w_{ij} = aw_{ij}^{obj} + \sum_{k=1}^r b_k w_{ij}^k, \quad i, j = \overline{1, N}$$

$$I_i = aI_i^{obj} + \sum_{k=1}^r b_k I_i^k, \quad i = \overline{1, N}$$

# Formulară neuronală a unei probleme de optimizare combinatorială

Exemplu: problema comis voiajorului

Se consideră un set de  $n$  orașe și se caută un traseu care trece o singură dată prin fiecare oraș și are costul minim



# Formulară neuronală a unei probleme de optimizare combinatorială

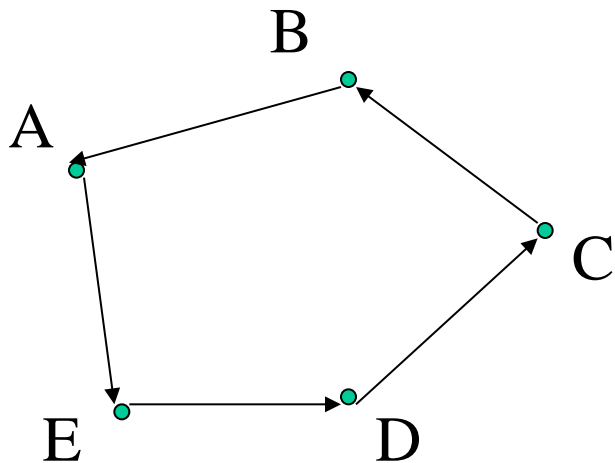
Proiectarea rețelei neuronale:

$N = n * n$  neuroni

Starea neuronului (i,j):

1 - la etapa j este vizitat orașul i

0 - altfel



	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

AEDCB



# Formulara neuronală a unei probleme de optimizare combinatorială

Descrierea problemei:

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

Restricții:

- la un moment dat este vizitat un singur oraș (fiecare coloană conține un singur 1)
- fiecare oraș este vizitat o singură dată (fiecare linie conține un singur 1)

Criteriu de optim:

costul traseului este minim

# Formulară neuronală a unei probleme de optimizare combinatorială

Descrierea problemei:

$$\sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 = 0$$

$$\sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 = 0$$

$$C(Y) = \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1})$$

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

# Rescriere ca problemă de optimizare fără restricții

$$\sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 = 0$$

$$\sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 = 0$$

$$C(Y) = \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1})$$

$$C^*(Y) = \frac{a}{2} \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1}) + \frac{b}{2} \left( \sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 + \sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 \right)$$

# Identificarea cu fctia Liapunov

$$V(Y) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n w_{ij,kl} y_{ij} y_{kl} - \sum_{i=1}^n \sum_{j=1}^n y_{ij} I_{ij}$$

$$C^*(Y) = \frac{a}{2} \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1}) +$$
$$\frac{b}{2} \left( \sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 + \sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 \right)$$

Valorile parametrilor:

$$w_{ij,kl} = -ac_{ik} (\delta_{l,j-1} + \delta_{l,j+1}) - b(\delta_{ik} + \delta_{jl} + \delta_{ik} \delta_{jl})$$
$$w_{ij,ij} = 0$$
$$I_{ij} = 2b$$

# Serii temporale

- Serie temporală = serie de date înregistrate la momente succesive de timp (corespunzătoare unor mărimi ce evoluează în timp)
- Exemple:
  - Evolutia cursului valutar
  - Evolutia pretului unei actiuni
  - Evolutia temperaturii
  - Semnale biologice (electrocardiograma –EKG)
- Scopul analizei unei serii temporale: efectuarea de predicții

# Serii temporale

Predicția se bazează pe un model asociat seriei care exprimă dependența dintre valoarea viitoare și valoarea curentă, respectiv valori anterioare

$$x(t + 1) = F(x(t), x(t - 1), \dots, x(t - p + 1), \varphi_1, \varphi_2, \dots, \varphi_s)$$

ordinul modelului



Parametrii ce modelează factorii externi

# Serii temporale

Modelul asociat unei serii temporale poate fi:

- Liniar
- Neliniar
- Determinist
- Aleator

Exemplu: modelul auto-regresiv (AR(p))

$$x(t+1) = w_0x(t) + w_1x(t-1) + \dots + w_{p-1}x(t-p+1) + zgomot$$

Model zgomot (variabila  
aleatoare din  $N(0,1)$ )

# Serii temporale

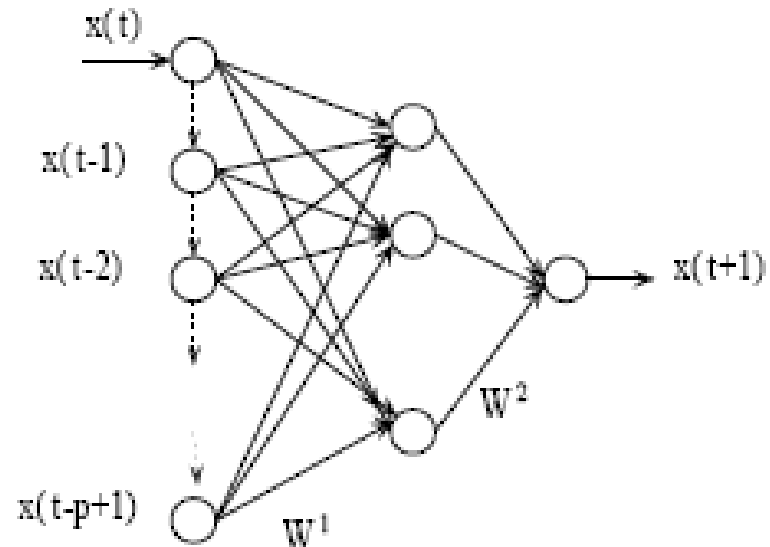
## Modelare cu rețele neuronale

- Ordinul modelului este cunoscut
  - Rețea feedforward cu ferestre temporale (p unitati de intrare)
- Ordinul modelului este necunoscut
  - Retea cu unități contextuale (retea Elman)



# Rețele cu ferestre temporale

Arhitectura:



Functionare:

$$y = \sum_{k=1}^K w_k^2 f\left(\sum_{j=0}^{p-1} w_{kj}^1 x(t-j)\right)$$

# Rețele cu ferestre temporale

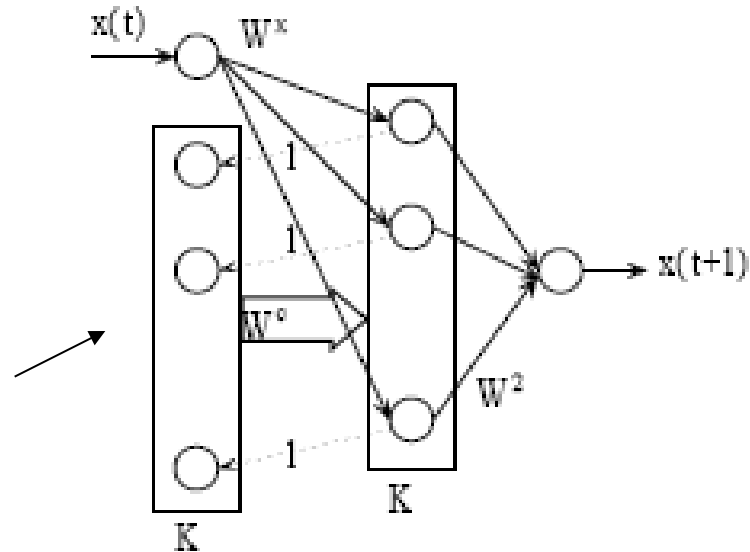
## Antrenare:

- Set antrenare:  $\{((x_l, x_{l-1}, \dots, x_{l-p+1}), x_{l+1})\}_{l=1..L}$
- Algoritm de antrenare: BackPropagation
- Dezavantaj: presupune cunoașterea lui  $p$

# Rețele cu unități contextuale

Arhitectura (retea Elman):

Unități contextuale



Funcționare:

$$y = \sum_{k=1}^K w_k^2 h_k(t)$$

$$h_k(t) = f \left( w_k^x x(t) + \sum_{j=1}^K w_{kj}^c h_j(t-1) \right) \quad h_j(0) = 0.$$

**Obs:** unitățile contextuale conțin copii ale stărilor unităților ascunse

# Rețele cu unități contextuale

## Antrenare

Set antrenare:  $\{(x(1),x(2)),(x(2),x(3)),\dots,(x(t-1),x(t))\}$

Ponderi:

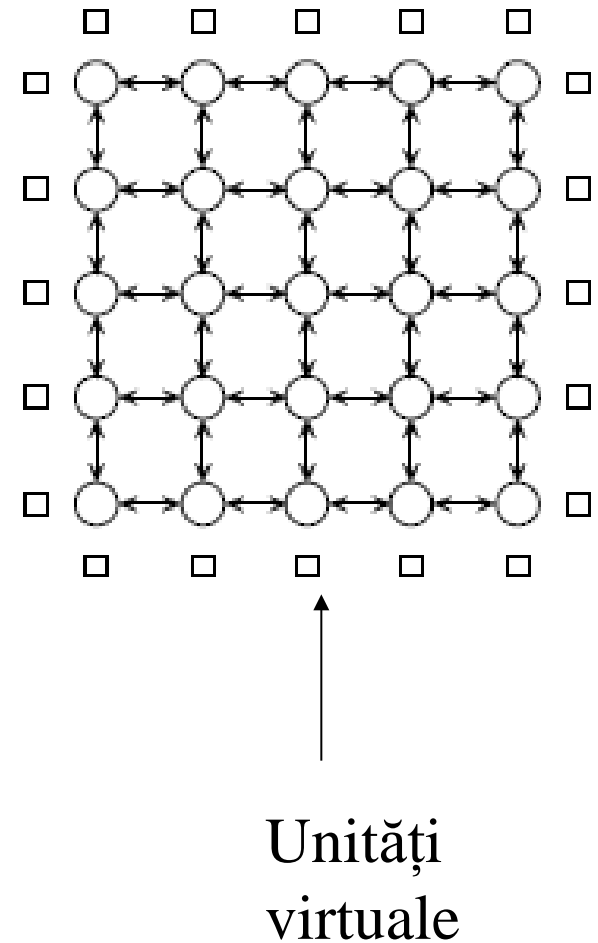
- Ajustabile:  $W^x$ ,  $W^c$  și  $W^2$
- Neajustabile: ponderile conexiunilor dintre nivelul ascuns și cel contextual

Algoritm antrenare: BackPropagation

# Rețele celulare

## Arhitectura:

- Toate unitățile au dublu rol: unități de intrare și unități de ieșire
- Unitățile sunt plasate în nodurile unei grile bidimensionale (notată cu  $L$ )
- Fiecare unitate este identificată prin poziția ei  $p=(i,j)$  în cadrul grilei
- Fiecare unitate este conectată doar cu unitățile aflate în vecinătatea sa (vecinătățile se definesc pe baza unei distanțe; cel mai frecvent se folosește  $d((i,j),(k,l))=\max\{|i-k|,|j-l|\}$ )

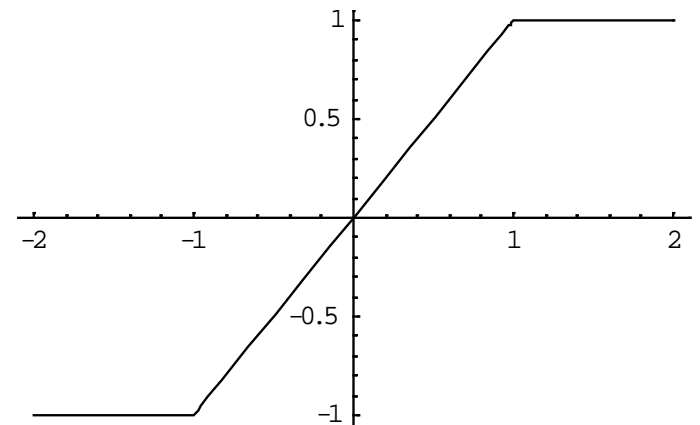


# Rețele celulare

Funcție de activare: rampă

$$f(u) = (|u + 1| - |u - 1|)/2$$

Notatii



- $X(t) = \{x_p(t) | p \in \mathcal{L}\}$  reprezintă ansamblul semnalelor de intrare în unitățile funcționale;
- $Y(t) = \{y_p(t) | p \in \mathcal{L}\}$  reprezintă ansamblul semnalelor de ieșire,  $y_p(t) = f(x_p(t))$ ;
- $U(t) = \{u_p(t) | p \in \mathcal{L}\}$  reprezintă ansamblul semnalelor de control transmise unităților;
- $I(t) = \{i_p(t) | p \in \mathcal{L}\}$  reprezintă ansamblul pragurilor corespunzătoare unităților;
- $a_{pq}$  este ponderea conexiunii dintre unitatea  $q$  și unitatea  $p$ ;
- $b_{pq}$  este ponderea conexiunii de la componenta  $q$  a semnalului de control la unitatea funcțională  $p$ ;

# Rețele celulare

Funcționare:

Semnal produs de  
alte unități

Semnal  
de control (sau de intrare)

$$\frac{dx_p(t)}{dt} = -x_p(t) + \sum_{q \in V(p)} a_{pq} y_q(t) + \sum_{q \in V(p)} b_{pq} u_q(t) + i_p(t), \quad p \in \mathcal{L}$$

Semnal de intrare  
(similar unui prag)

Obs:

- Grila asociată rețelei este bordată cu celule virtuale care produc o valoare specifică (in funcție de condițiile la frontieră)
- **Caz particular:** ponderile conexiunilor dintre unitățile vecine nu depind de poziția acestor unități

**Exemplu:** dacă  $p=(i,j)$ ,  $q=(i-1,j)$ ,  $p'=(i',j')$ ,  $q'=(i'-1,j')$  atunci

$$a_{pq} = a_{p'q'} = a_{-1,0}$$

# Rețele celulare

Condițiile la frontieră definesc valorile produse de către celulele virtuale

Variante de condiții la frontieră:

- Celulele virtuale produc o valoare constantă (aceeași pentru toate sau diferite în funcție de poziția lor) – condiții Dirichlet
- Celulele virtuale preiau valoarea unității funcționale vecine (de exemplu:  $z_{i,0}=z_{i,1}$ ,  $z_{i,n+1}=z_{i,n}$ ,  $z_{0,j}=z_{1,j}$ ,  $z_{n+1,j}=z_{n,j}$ ) – condiții Neumann
- Celulele virtuale preiau valoarea unității funcționale de pe frontiera opusă (de exemplu:  $z_{i,0}=z_{i,n}$ ,  $z_{i,n+1}=z_{i,1}$ ,  $z_{0j}=z_{nj}$ ,  $z_{n+1,j}=z_{1,j}$ ) – condiții periodice la frontieră



# Retele celulare

Retelele ce folosesc ponderi care depind doar de poziția relativă a unităților se numesc **rețele cu șabloane independente de poziție**

Exemplu:

$$V_1(i, j) = \{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$$

$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \quad B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

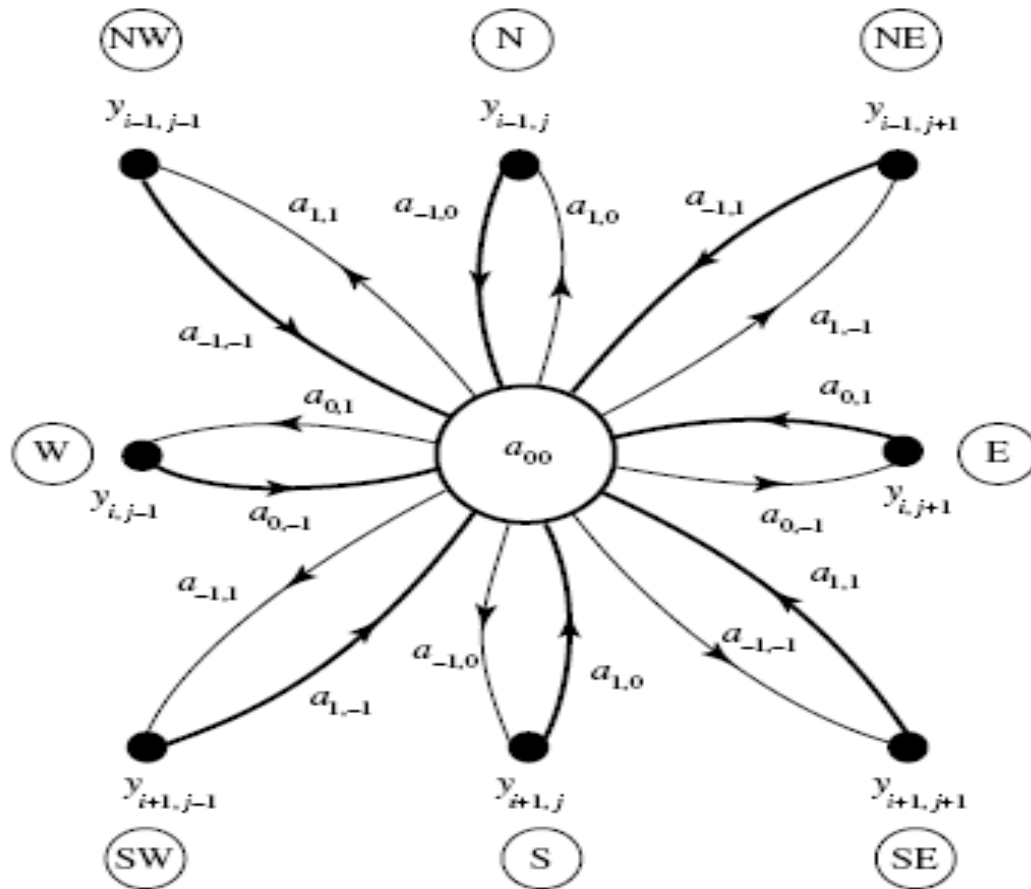
$$\frac{dx_{i,j}(t)}{dt} = -x_{i,j}(t) + \sum_{(k,l) \in V^*} a_{k,l} y_{i+k,j+l}(t) + \sum_{(k,l) \in V^*} b_{k,l} u_{i+k,j+l}(t) + I, \quad i, j \in \{1, \dots, n\}$$

cu

$$V^* = \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$$

# Retele celulare

Utilizarea elementelor din șabloane independente de poziție:



# Rețele celulare

Simulare software = rezolvare numerică a sistemului de ecuații diferențiale completat cu o condiție inițială

Se poate folosi metoda explicită a lui Euler

$$x_p(t+1) = (1-h)x_p(t) + h\left(\sum_{(k,l) \in V^+} a_{k,l}y_{i+k,j+l}(t) + \sum_{(k,l) \in V^+} b_{k,l}u_{i+k,j+l}(t) + I\right), \quad i,j \in \{1, \dots, n\}$$

p=(i,j)

## Aplicabilitate:

- Prelucrarea imaginilor alb-negru sau pe nivele de gri
- Fiecare pixel este asociat cu o unitate a rețelei
- Nivelul de gri al unui pixel este codificat în  $[-1, 1]$

# Rețele celulare

## Prelucrarea imaginilor:

- In functie de alegerea matricilor șablon, a semnalului de control ( $u$ ), a condiției inițiale ( $x(0)$ ) și a condițiilor pe frontieră ( $z$ ) se obțin diferite tipuri de prelucrări:
  - Binarizare (imagine pe nivele de gri  $\rightarrow$  imagine binară)
  - Umplerea golurilor în imagini binare
  - Eliminarea zgomotului (pixeli izolați) din imagini binare
  - Detectarea contururilor în imagini binare

**Observație:** imaginile alb negru sunt codificate în  $\{-1,1\}$ :

Alb: -1

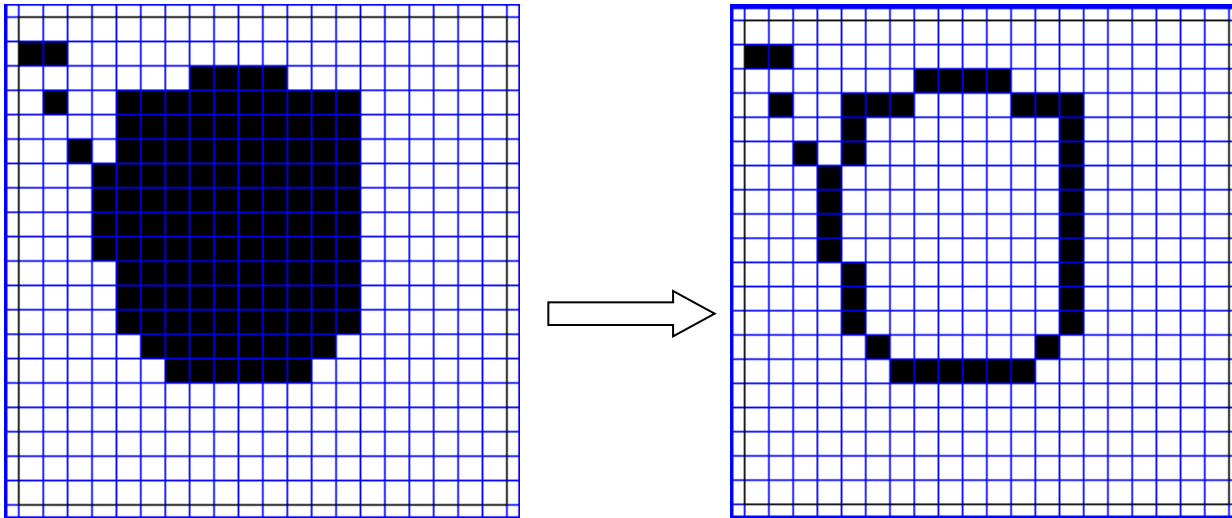
Negru: +1

# Rețele celulare

Exemplu 1: identificarea contururilor  
 $z=-1$ ,  $U$ = imaginea de prelucrat,  
 $h=0.1$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$I = -1, X(0) = U$$



[http://www.isiweb.ee.ethz.ch/haenggi/CNN\\_web/CNNsim\\_adv.html](http://www.isiweb.ee.ethz.ch/haenggi/CNN_web/CNNsim_adv.html)

# Rețele celulare

Exemplu 2: umplerea golurilor

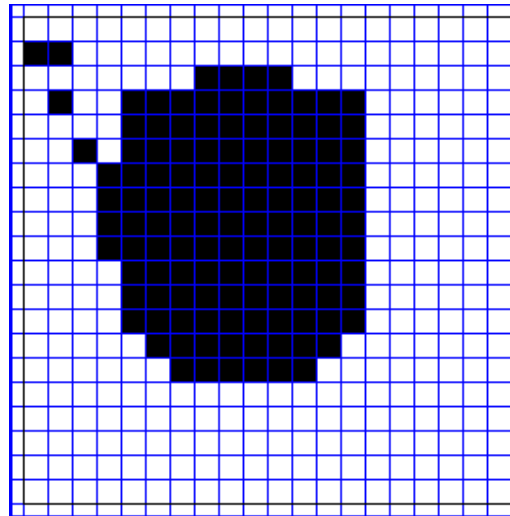
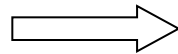
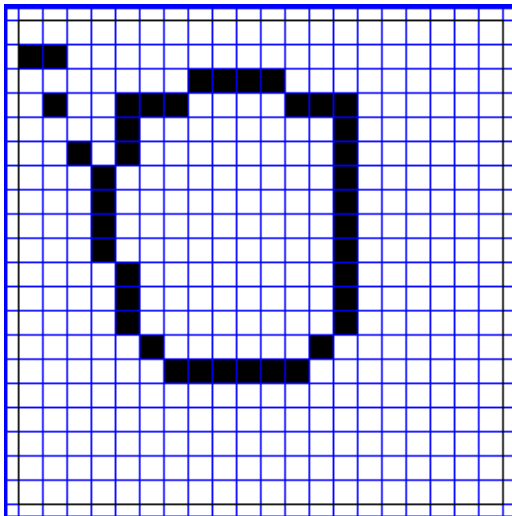
$z=-1$ ,

$U$ = imaginea de prelucrat,

$h=0.1$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1.5 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$I = 0.5$ ,  $x_{ij}(0) = 1$  (toti pixelii sunt egali cu 1)



# Rețele celulare

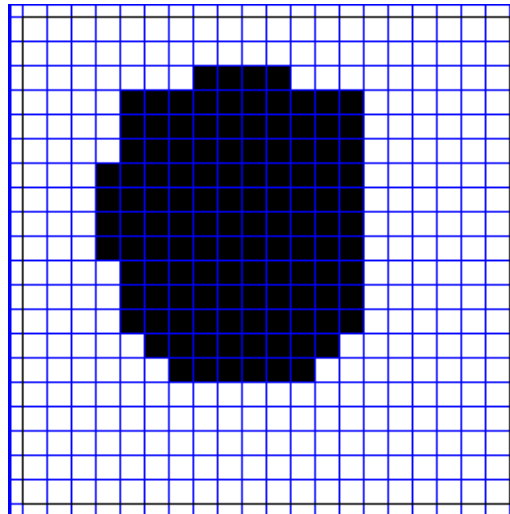
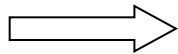
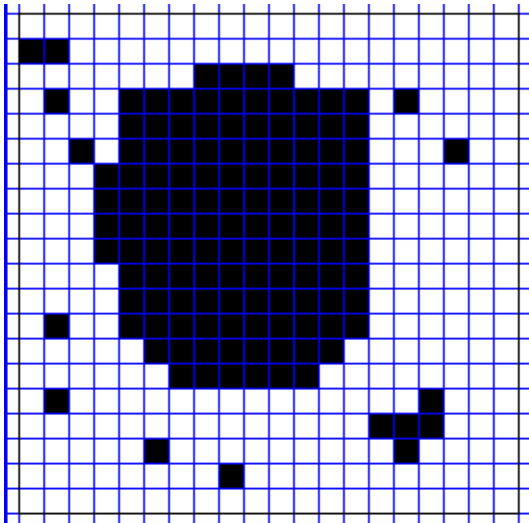
Exemplu 3: eliminarea zgomotului

$z=-1$ ,  $U$ =imaginea de prelucrat,

$h=0.1$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = 0, X(0) = U$$



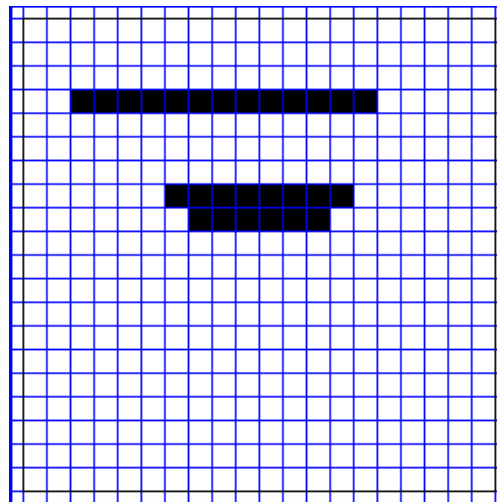
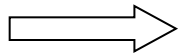
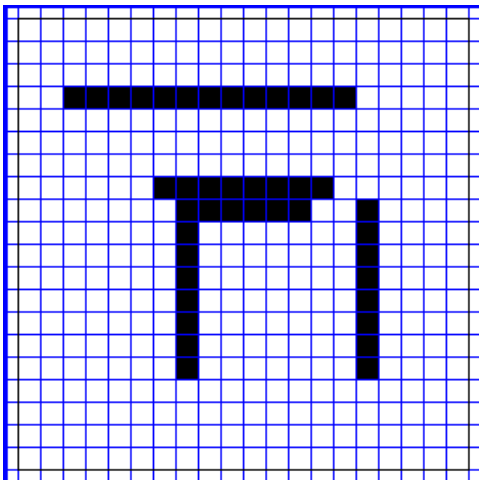
# Rețele celulare

## Exemplu 4: detectare linii orizontale

$z=-1$ ,  $U$ = imaginea de prelucrat,  $h=0.1$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = -1, X(0) = U$$





# Alte clase de rețele recurente

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

Specific:

- Utilizează un set de unități ascunse (numit rezervor) conectate arbitrar (ponderile conexiunilor sunt stabilite aleator); fiecare dintre aceste unități realizează o transformare neliniară a semnalelor preluate de către unitățile de intrare
- Semnalele de ieșire sunt determinate prin combinarea liniară a semnalelor produse de unitățile din rezervor (și a celor de intrare)
- Doar ponderile conexiunilor către unitățile de ieșire sunt antrenate

# Alte clase de rețele recurente

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

Variante:

- *Temporal Recurrent Neural Network* (Dominey 1995)
- *Liquid State Machines* (Natschläger, Maass and Markram 2002)
- *Echo State Networks* (Jaeger 2001)
- *Decorrelation-Backpropagation Learning* (Steil 2004)

# Alte clase de rețele recurente

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

Echo State Networks:

$U(t)$  = vector cu semnale de intrare

$X(t)$  = vector de stare corespunzător unităților din rezervor

$Z(t)=[U(t);X(t)]$  = vector intrare concatenat cu vector de stare

$Y(t)$  = vector cu semnalul de ieșire

$$X(t)=(1-a)x(t-1)+a \tanh(W^{\text{in}} U(t)+W x(t-1))$$

$$Y(t)=W^{\text{out}} Z(t)$$

$W^{\text{in}}$ ,  $W$  – stabilite aleator ( $W$  scalata a.i. raza spectrală = valoarea proprie maximă să aibă o valoare prestabilită);

$W^{\text{out}}$  determinat prin antrenare

# Alte clase de rețele recurente

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

Aplicații:

- recunoașterea vorbirii
- Recunoașterea scrisului de mână
- Controlul roboților
- Predicția datelor financiare
- Predicția în timp real a atacurilor de epilepsie

# Alte clase de rețele recurente

Deep learning (<http://deeplearning.net/>)

Specific:

- Arhitectura cu multe nivele (scop: extragerea ierarhica a caracteristicilor datelor);
- Pre-antrenare nesupervizata bazata pe RBM (Restricted Boltzmann Machines) urmata de antrenare supervizata (ex: Backpropagation)

Obs:

- Boltzmann Machines = rețele recurente cu unitati cu functionare aleatoare
- Restricted BM = rețele recurente organizate pe nivele cu conexiuni complete doar intre unitatile aflate pe nivele diferite
- Unele rețele cu structura “adâncă” sunt rețele feed-forward (ex: Convolutional Neural Networks)

# Alte clase de rețele recurente

Deep learning (<http://deeplearning.net/>)

## Aplicatii:

- Clasificare imagini si identificare obiecte (ex: recunoastere fete – Deep Face)
- Recunoasterea vorbirii (Google Brain, Siri)
- Indexare semantica (ex: word2vec) si traducere automata
- Simulare vise (<http://npcontemplation.blogspot.ca/2012/02/machine-that-can-dream.html>)