



# Învățare automată

## 1. Algoritmi de grupare (clustering)

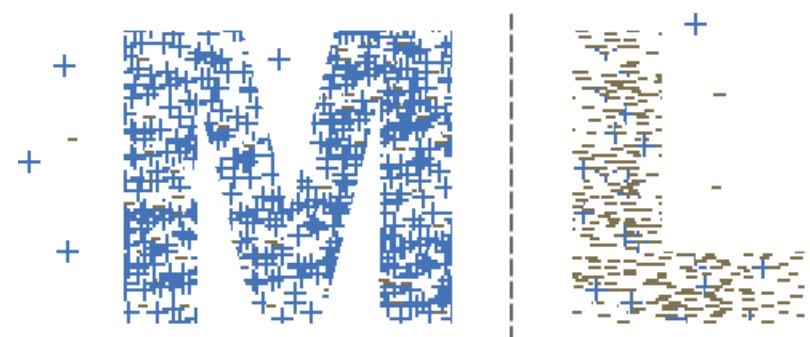
**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)

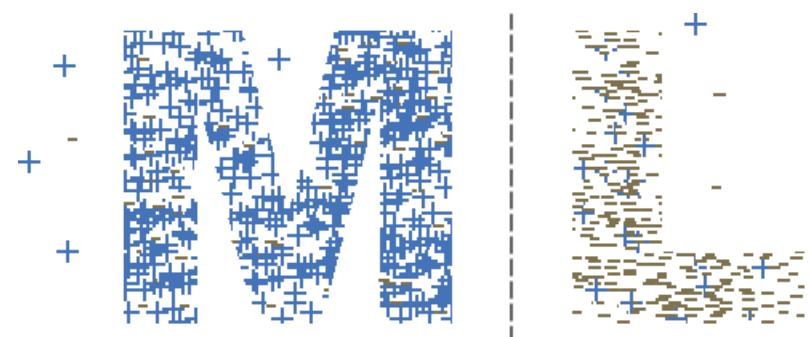
# Algoritmi de grupare (clustering)

1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
3. Algoritmul EM (Expectation-Maximization)
4. Gruparea ierarhică
5. Algoritmul DBSCAN



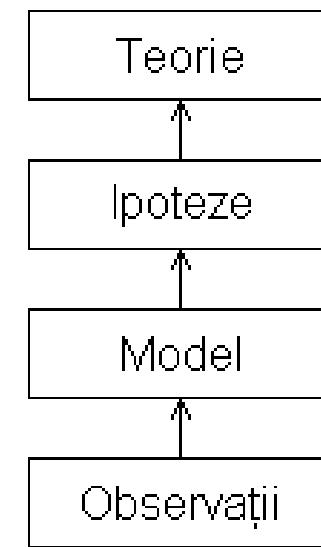
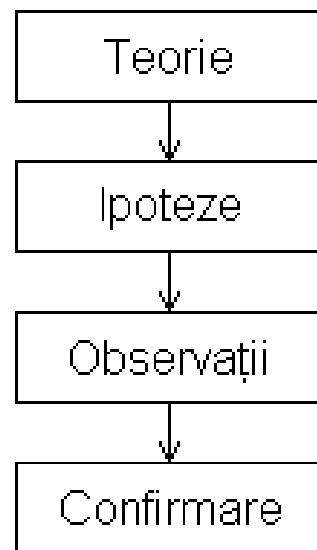
# Algoritmi de grupare (clustering)

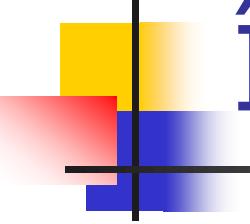
1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
3. Algoritmul EM (Expectation-Maximization)
4. Gruparea ierarhică
5. Algoritmul DBSCAN



# Tipuri de raționament

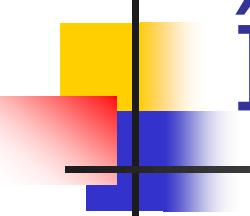
- Raționament deductiv
- Raționament inductiv





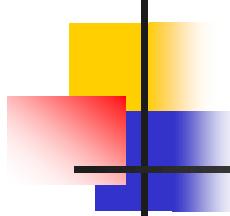
# Învățarea

- Capacitatea de învățare este unul din cele mai importante componente ale comportamentului intelligent
- Un sistem clasic specializat care nu învață:
  - Realizează calcule numeroase pentru rezolvarea unei probleme
  - Nu memorează soluția
  - De fiecare dată, realizează aceeași secvență de calcule complexe



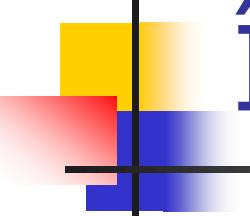
# Învățarea

- Un program învață dacă își îmbunătăște performanțele la îndeplinirea unei sarcini pe baza experienței (Tom Mitchell)
- Învățarea denotă schimbările dintr-un sistem, astfel încât acesta să poată realiza:
  - aceeași sarcină, mai eficient
  - sarcini noi, posibil similare



# Motivație

- Învățarea este esențială pentru mediile necunoscute
  - Nu se pot anticipa toate stările mediului
  - Proiectantul *nu poate* să îi dea agentului toate informațiile
- Învățarea este o alternativă la proiectarea explicită
  - Expune agentul la mediul real în loc să îi spună despre mediu
  - Proiectantul *nu dorește* să îi dea toate informațiile



# Învățarea automată

- engl. “machine learning”
- Își propune găsirea automată a unor modele interesante în date
- Directii principale:
  - Clasificarea și regresia
  - Gruparea (clustering [clăstăring], partitioanare, clusterizare [clasterizáre])
  - Determinarea regulilor de asociere
  - Selectia trăsăturilor

# Clasificarea

- Se dă o **mulțime de antrenare**: o mulțime de **instante** (vectori de antrenare, obiecte)
- Instantele au **attribute**
- Fiecare instanță are attribute cu anumite **valori**
- De obicei, ultimul atribut este **clasa**

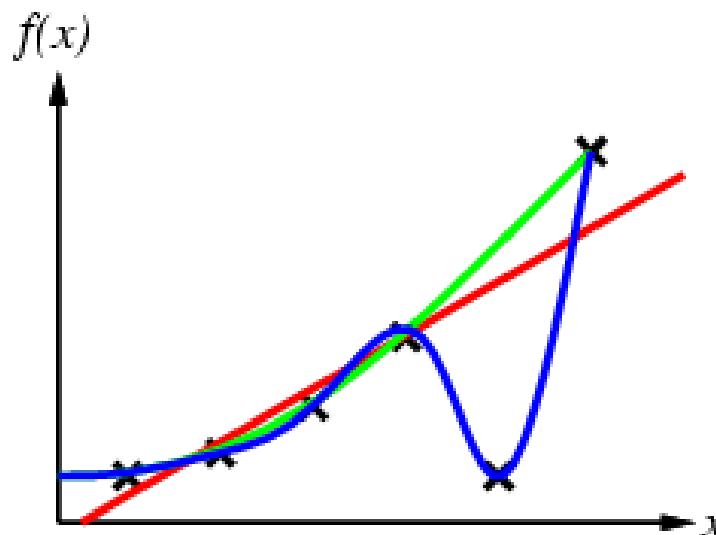
**Atribute**

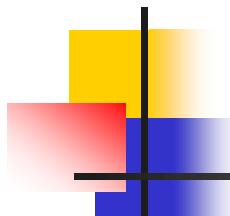
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

**Instante**

# Regresia

- Reprezintă aproximarea unei funcții
  - Orice fel de funcție, nu doar de tipul  $f: \mathbb{R}^n \rightarrow \mathbb{R}$
  - Doar ieșirea este continuă; unele intrări pot fi discrete sau nenumerice



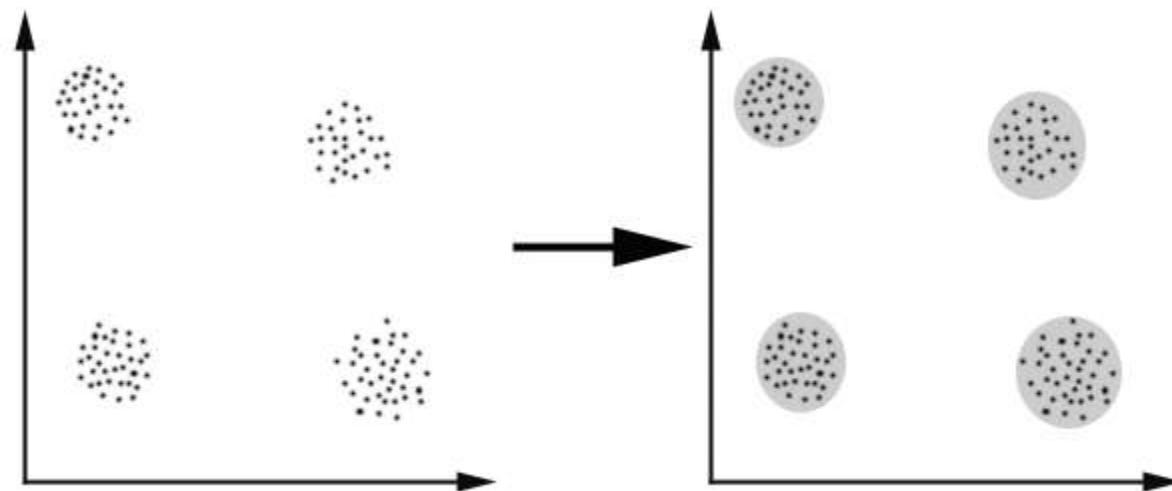


# Clasificarea și regresia

- Aceeași idee de bază: învățarea unei relații între intrări (vectorul  $x$ ) și ieșire ( $y$ ) din date
- Singura diferență între clasificare și regresie este tipul ieșirii: discret, respectiv continuu
- Clasificarea estimează o ieșire discretă, clasa
- Regresia estimează o funcție  $h$  astfel încât  $h(x) \approx y(x)$  cu o anumită precizie
- Pentru fiecare instanță de antrenare, valoarea dorită a ieșirii este dată  $\Rightarrow$  **învățare supervizată**

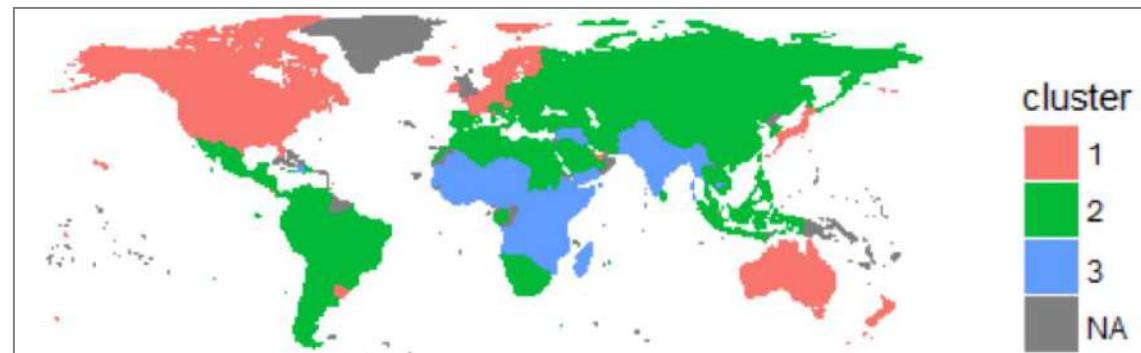
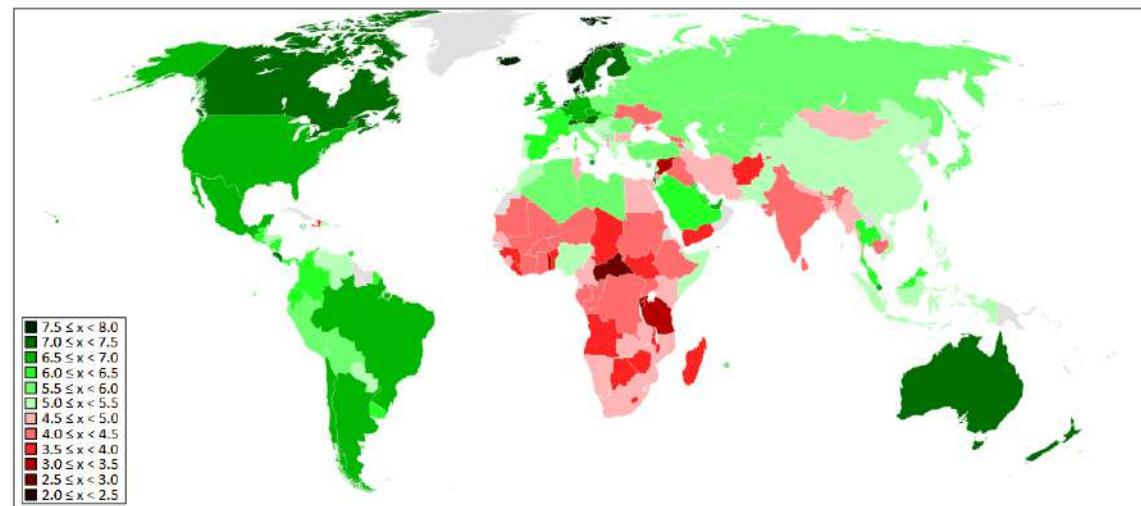
# Gruparea

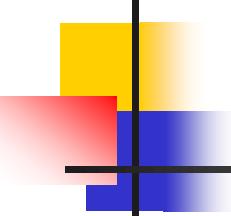
- Are ca scop găsirea unor grupuri astfel încât instanțele din același grup să fie mai asemănătoare între ele decât cu instanțele altor grupuri
- De obicei, este **nesupervizată**



# Gruparea (clusterizarea)

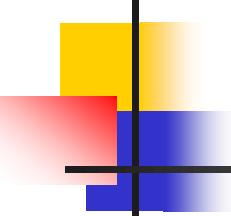
*World  
happiness  
report*





# Exemple de aplicații

- Gruparea rezultatelor de căutare pe web (imagini, documente), înainte de a le clasifica
- Rețele sociale: identificarea comunităților
- Segmentarea imaginilor: identificarea regiunilor având caracteristici similare, pentru detecția obiectelor
- Marketing: gruparea clienților cu comportamente similare
- Asigurări: identificarea unor grupuri de asigurați, identificarea fraudelor
- Biologie: gruparea animalelor și planetelor pe baza trăsăturilor
- Biblioteci: gruparea cărților



# Reguli de asociere

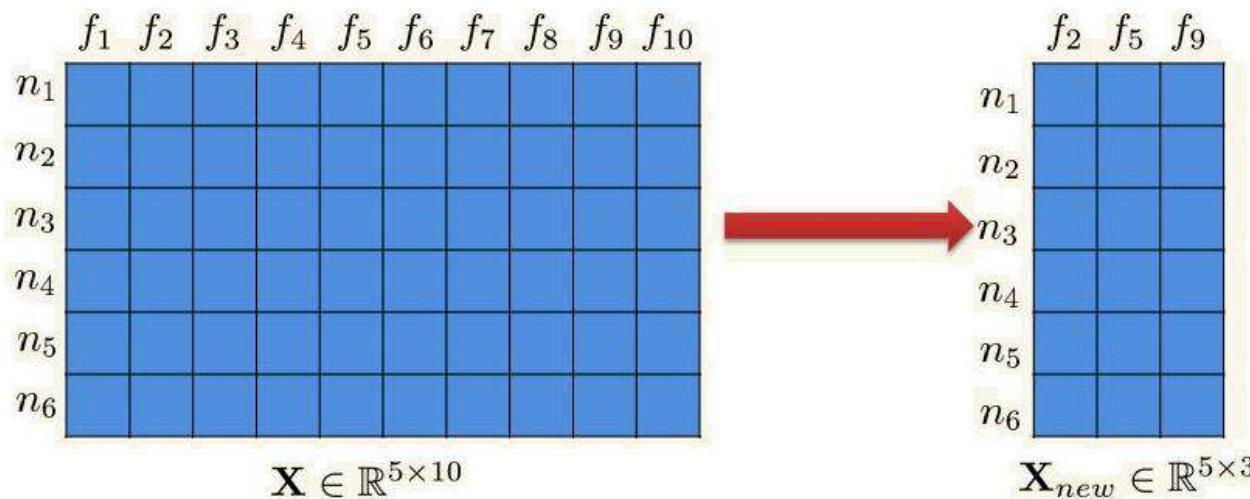
- Regulile de asociere identifică relații interesante între date tranzacționale aparent neînrudite
- Exemplu: analiza coșului de cumpărături

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Beer, Diaper, Bread, Eggs
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Bread, Diaper, Milk

- Persoanele care cumpără lapte și scutece cumpără și bere în 67% din cazuri

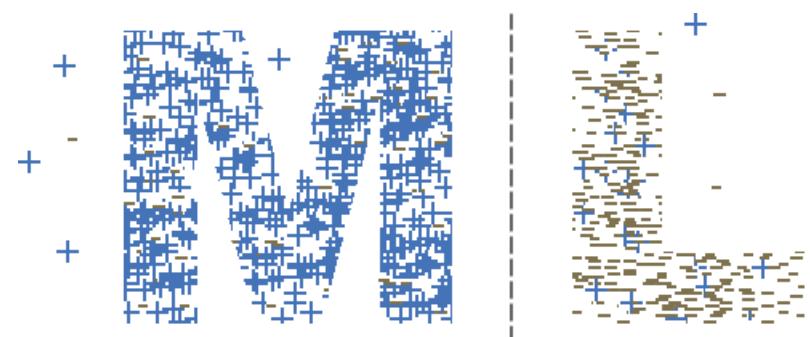
# Selectia trăsăturilor

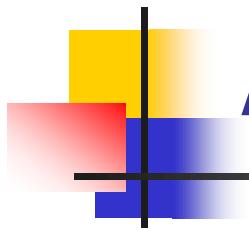
- Reprezintă determinarea unei submulțimi de atrbute relevante din mulțimea de antrenare
- De obicei, este utilizată înainte de aplicarea unor algoritmi de clasificare sau regresie
- Prin scăderea dimensionalității, problemele pot deveni mai ușor de rezolvat



# Algoritmi de grupare (clustering)

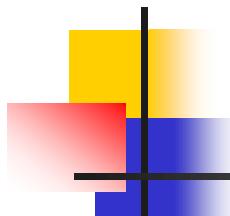
1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
3. Algoritmul EM (Expectation-Maximization)
4. Gruparea ierarhică
5. Algoritmul DBSCAN





# Algoritmul $k$ -medii

- engl. “ $k$ -means”
- Algoritmul partiționează o mulțime de instanțe **numerice** în  $k$  grupuri (clustere)
- $k$  reprezintă numărul de grupuri și este un parametru de intrare ales de utilizator



# Modul de funcționare

- Se initializează aleatoriu cele  $k$  centre inițiale
- Se atribuie fiecărui centru instanțele cele mai apropiate de el

$$C(i) = \operatorname{argmin}_j \|x_i - c_j\|^2$$

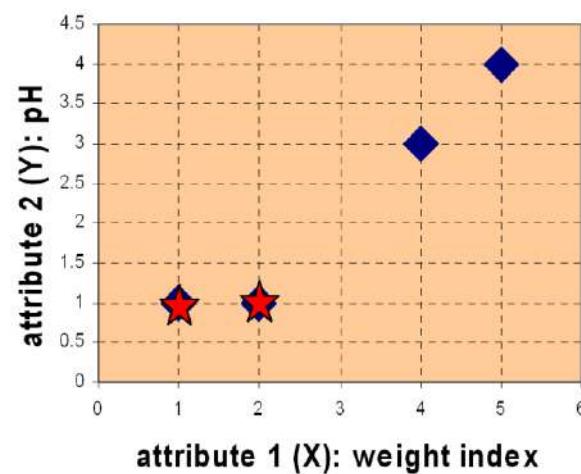
- Se calculează centrul de greutate (media aritmetică) a tuturor instanțelor atribuite unui centru și se actualizează poziția centrului grupului respectiv

$$c_j = \frac{\sum_{i:x_i \in C_j} x_i}{n_j}$$

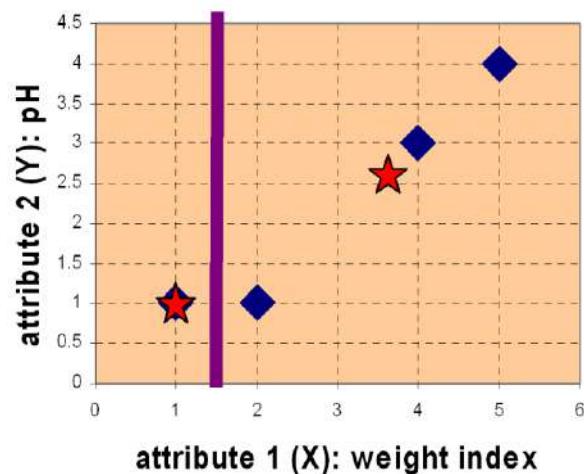
- Se repetă cei doi pași până când nu se mai modifică poziția niciunui centru

# Exemplul 1

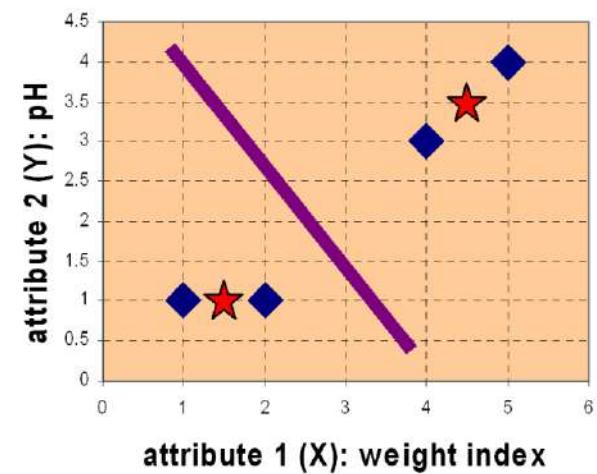
iteration 0



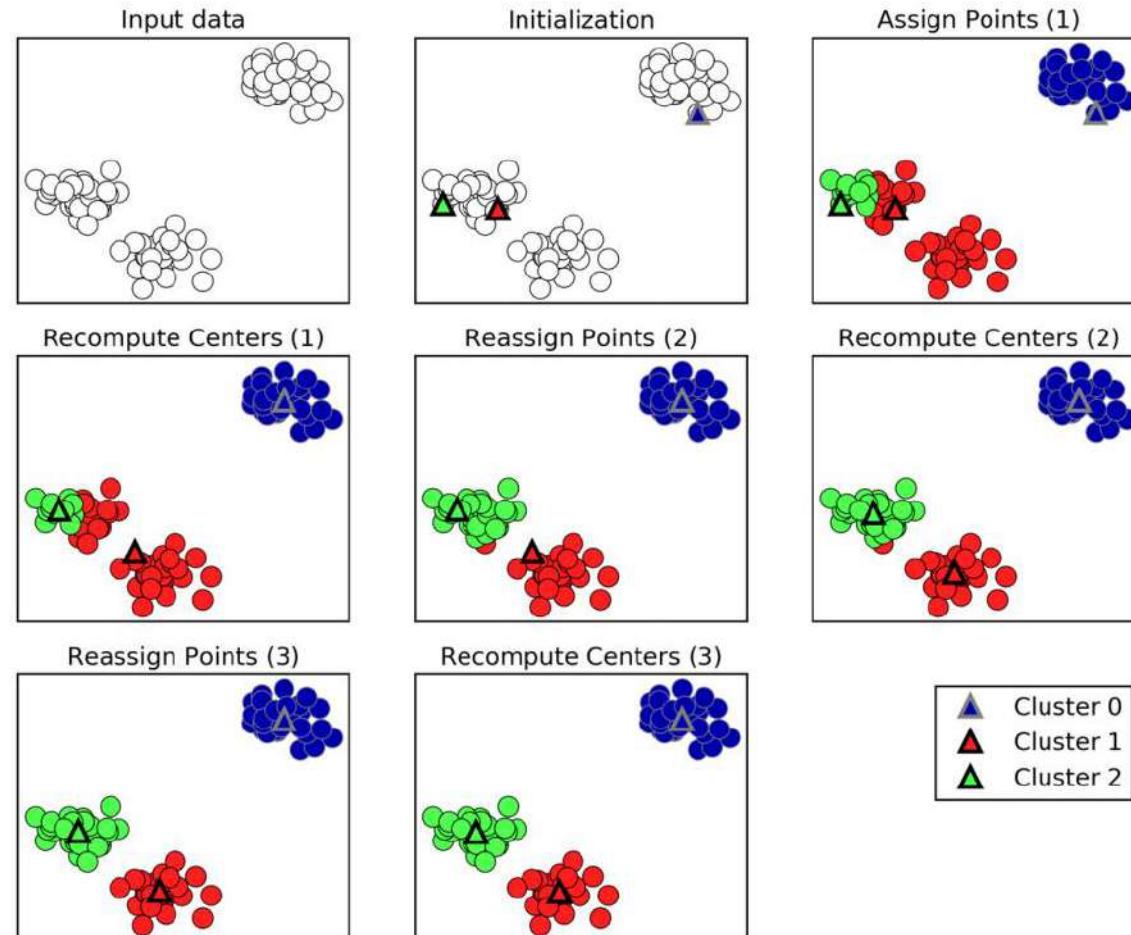
iteration 1

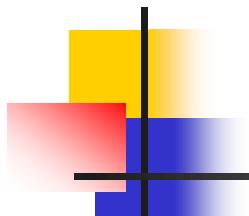


iteration 2

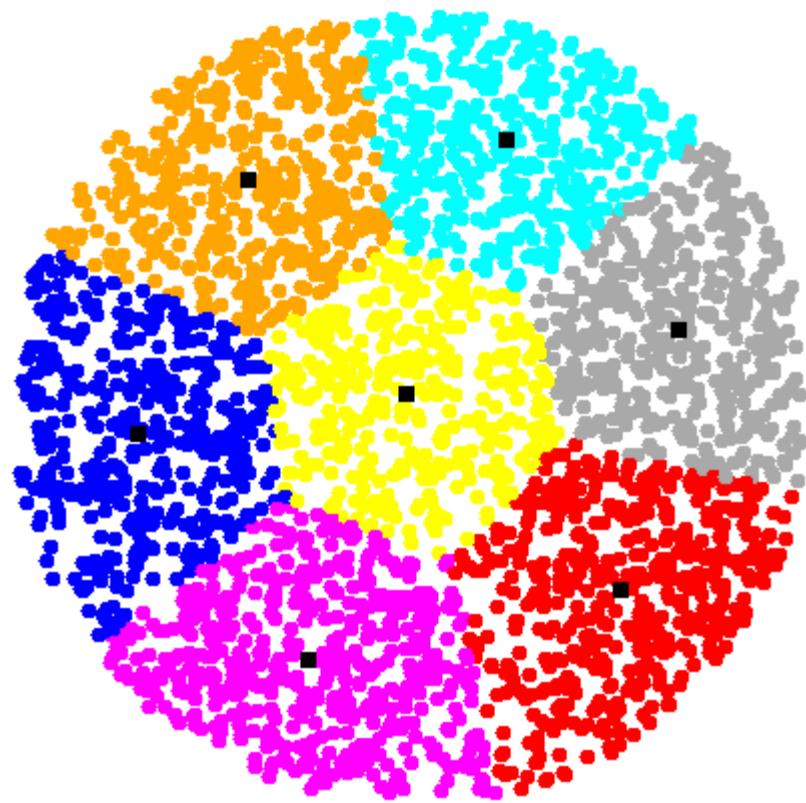
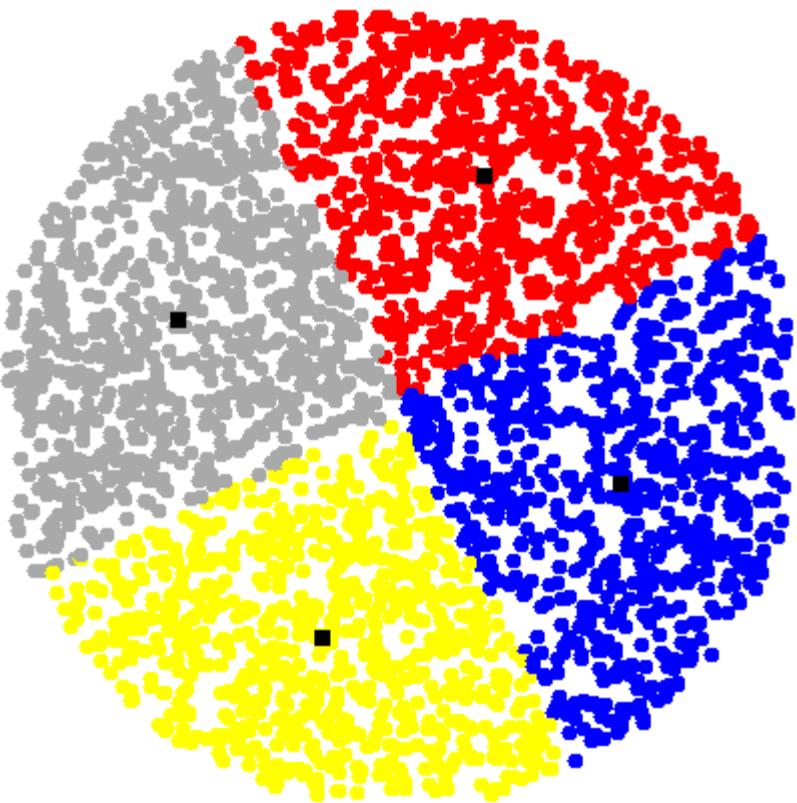


# Exemplul 2





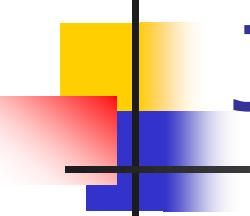
## Exemplul 3



# Exemplul 4: segmentarea imaginilor



- Imaginea din dreapta este rezultatul partitioнării cu trei grupuri, corespunзătoare nivelurilor de gri din imaginea din st ngă



# Justificarea matematică

- Se dorește minimizarea sumei erorilor pătratice

$$SSE = \sum_{j=1}^k \sum_{x_i \in C_j} (x_i - c_j)^2$$

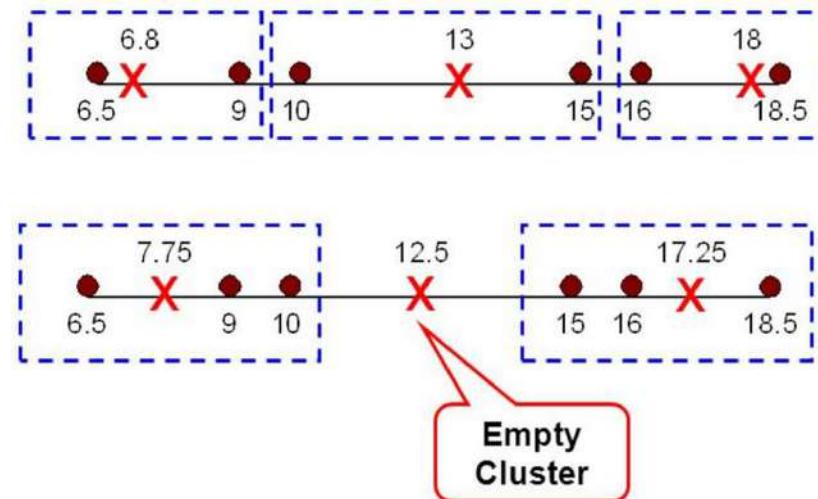
- Trebuie determinate centrele  $c_j$  care minimizează SSE

$$\frac{\partial SSE}{\partial c_j} = 2 \sum_{x_i \in C_j} (x_i - c_j) = 0 \Rightarrow c_j = \frac{1}{n_j} \sum_{x_i \in C_j} x_i$$

- Deci media aritmetică a instanțelor dintr-un grup reprezintă valoarea centrului care minimizează SSE
- Este doar o optimizare locală, la nivel de grup

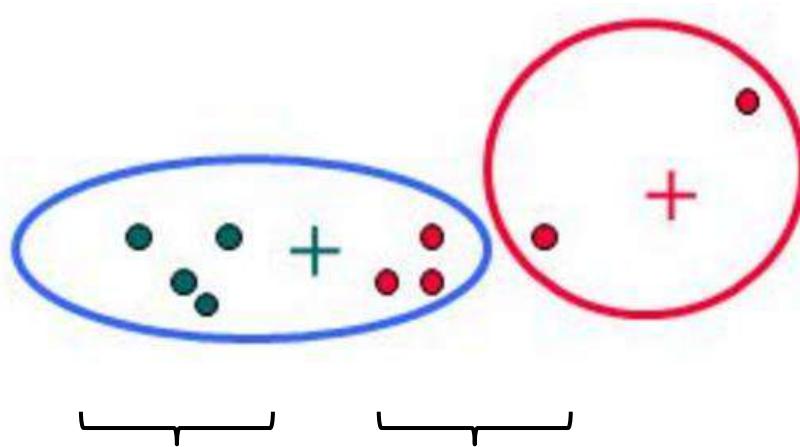
# Grupuri vide

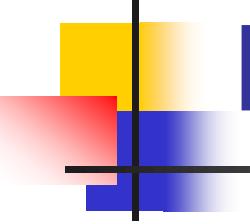
- Se poate întâmpla ca nicio instanță sa nu aparțină unui centru
- Centrele inițiale se pot alege în niște instanțe existente, astfel încât fiecare centru să aibă cel puțin o instanță atribuită
- Dacă există un centru de grup vid, se alege un alt centru în instanța cea mai depărtată de toate celelalte centre



# Valori extreme

- engl. “outliers”
- Valorile extreme pot influența rezultatele grupării
- Înainte de grupare, în faza de pre-procesare a datelor, se pot elimina valorile extreme





# Probleme

- Viteza de căutare

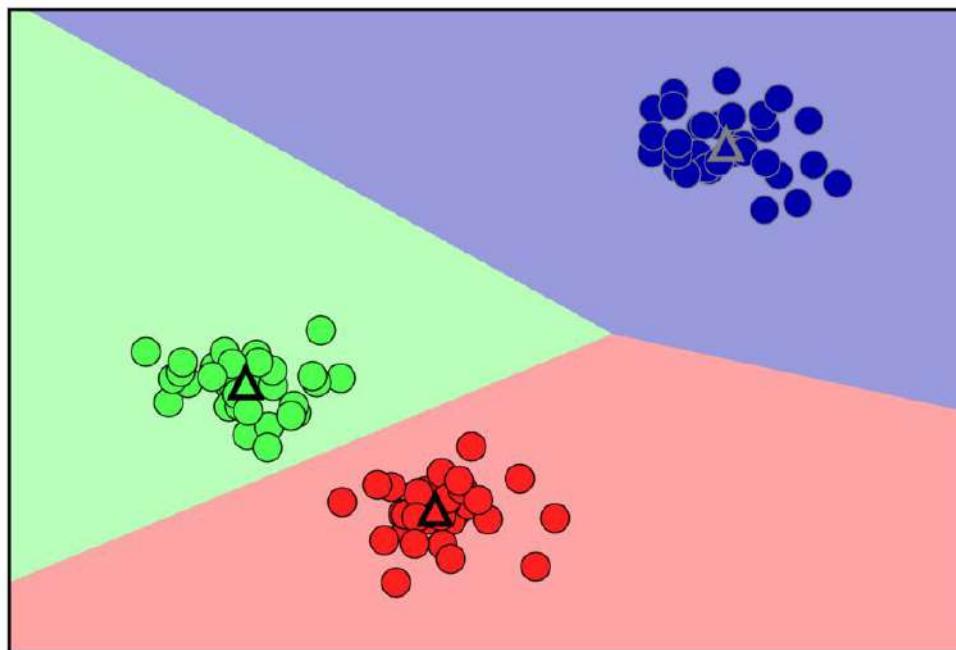
- Pentru determinarea mai rapidă a instanțelor celor mai apropiate de un centru se pot utiliza metodele *kd-tree* sau *ball-tree*
- Metoda *kd-tree* va fi descrisă în cursul 2

- Convergența

- Algoritmul converge, dar găsește de obicei un minim local al funcției de eroare
- În general, se recomandă mai multe rulări și alegerea rezultatului celui mai potrivit

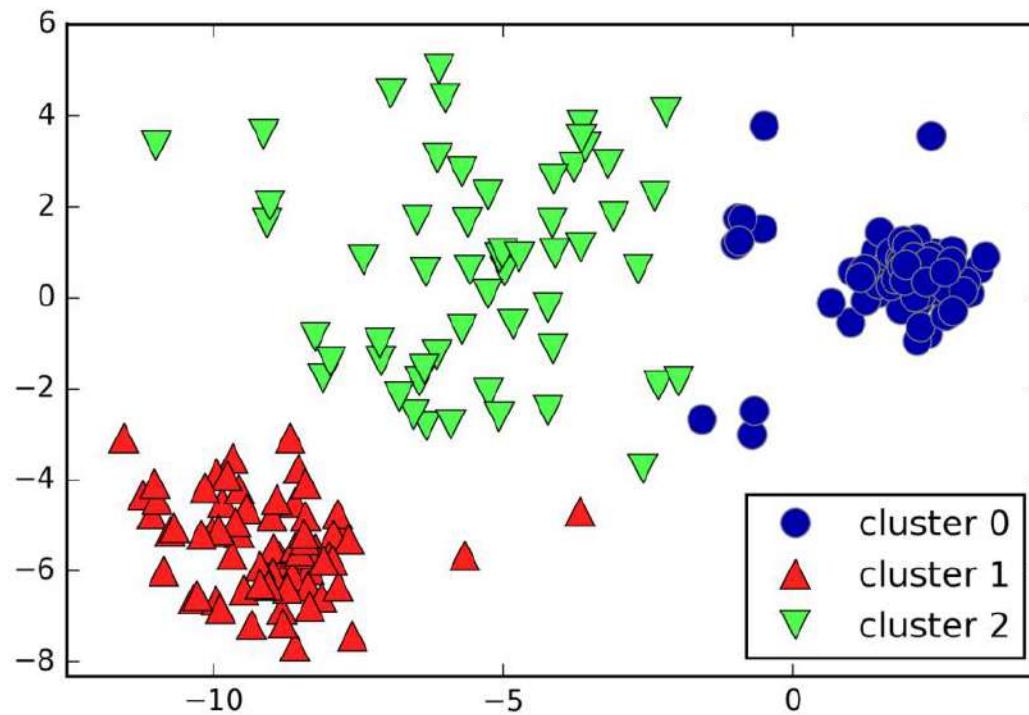
# Densitatea grupurilor

- Algoritmul face ca grupurile să aibă diametre relativ egale, deoarece granițele dintre grupuri sunt la mijlocul distanței dintre centre



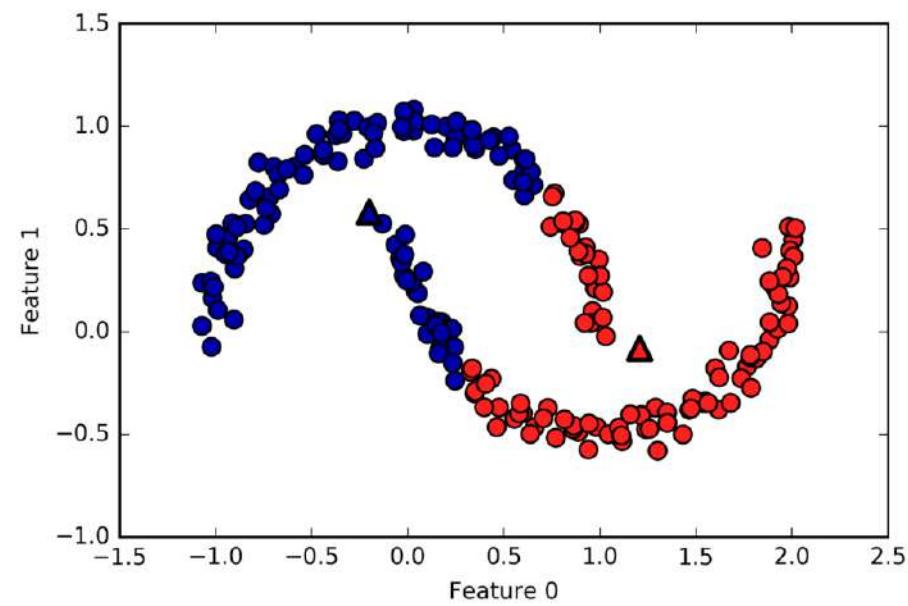
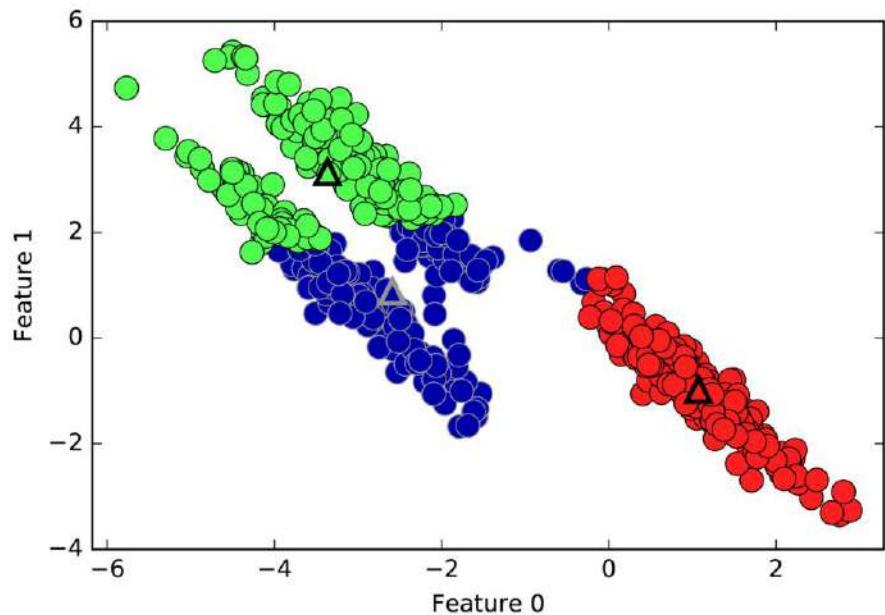
# Densitatea grupurilor

- Dacă grupurile au densități diferite, rezultatele pot fi afectate negativ

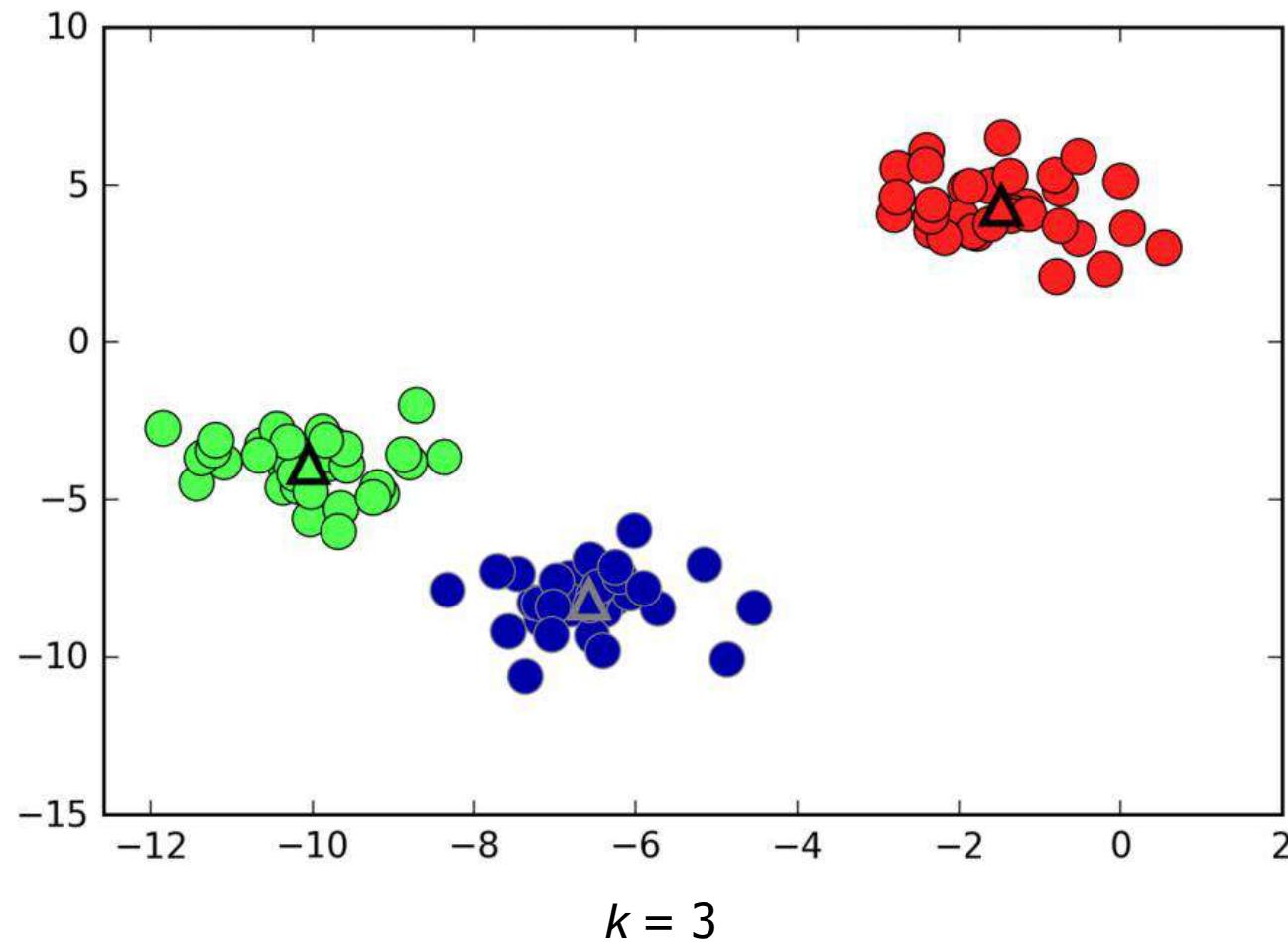


# Forma grupurilor

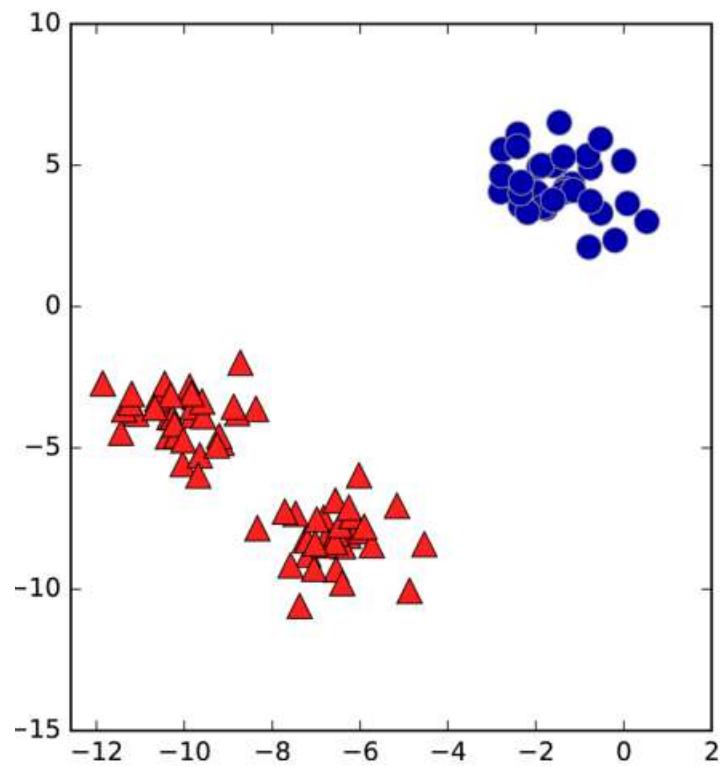
- Algoritmul dă rezultate bune dacă grupurile sunt convexe și mai ales de formă aproximativ sferică



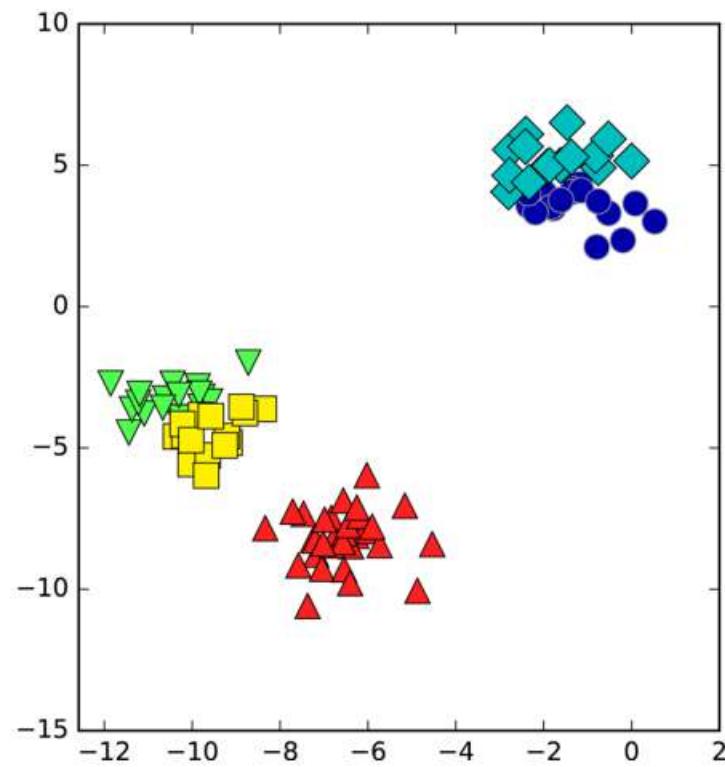
# Numărul de grupuri



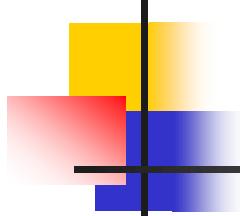
# Numărul de grupuri



$k = 2$



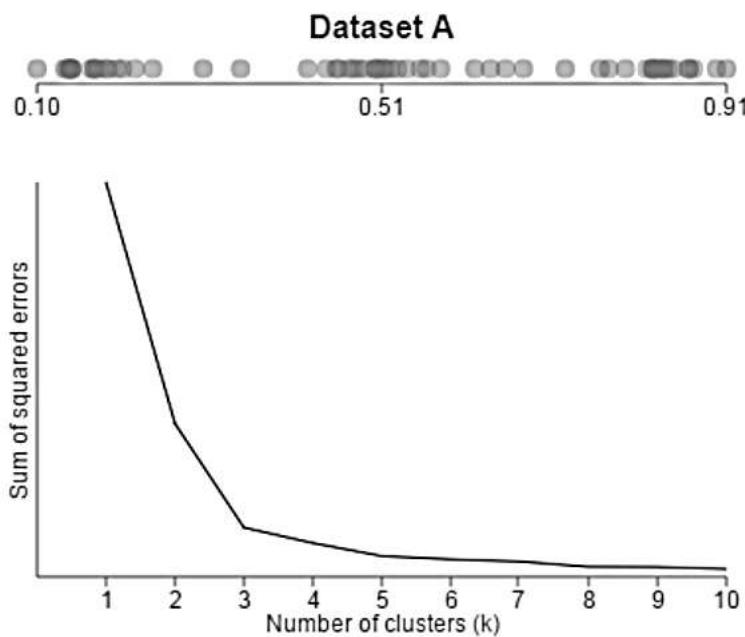
$k = 5$



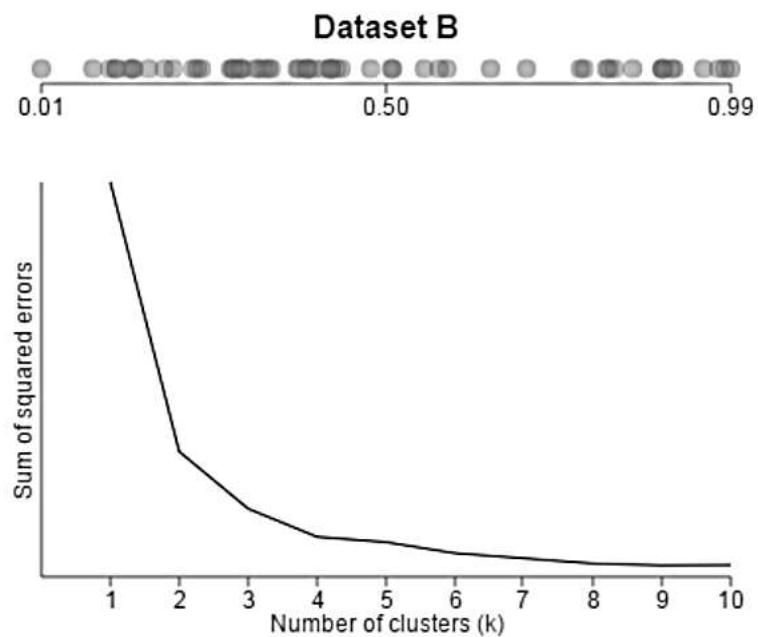
# Determinarea lui $k$

- De obicei, se încearcă mai multe valori pentru numărul de grupuri  $k$  și se alege valoarea care dă cele mai bune rezultate
- O metodă automată de alegere a lui  $k$  este **metoda cotului** (*elbow method*)
- Se alege valoarea lui  $k$  pentru care funcția  $SSE(k)$  nu mai scade prea mult odată cu creșterea lui  $k$

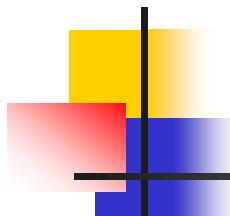
# Metoda cotului



$$k = 3$$

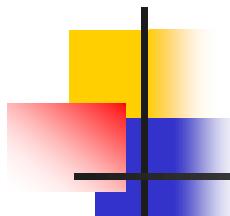


Aici valoarea prag nu este la fel de clară.  
Pot fi necesare metode mai complexe.



# Măsuri de calitate a grupării

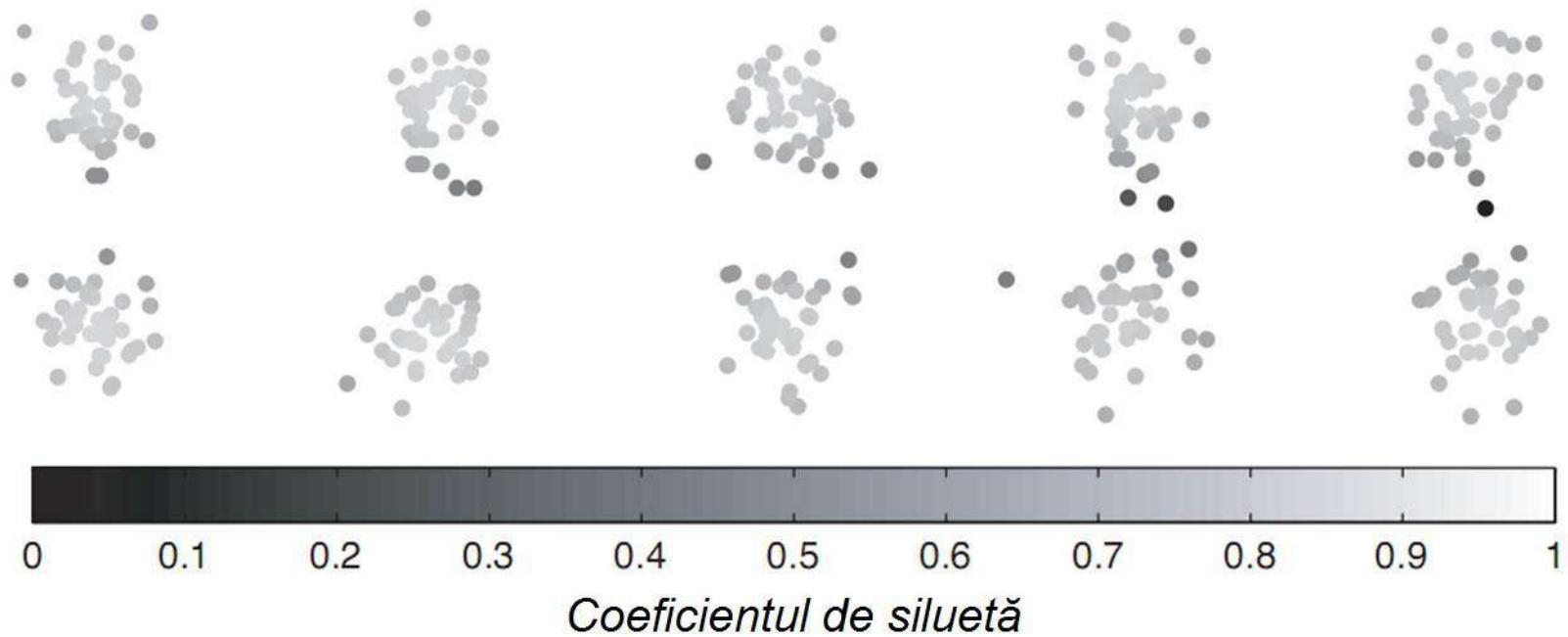
- În funcție de alegerea centrelor inițiale, rezultatele finale pot dифeri mult
- O măsură a calității grupării este **coeficientul de siluetă**
- Scopul său este **maximizarea similarității intra-grup și minimizarea similarității inter-grup**
- Există și alte măsuri, de exemplu, **utilitatea categoriilor**, bazată pe probabilități



# Coeficientul de siluetă

- Pentru o instanță, se calculează:
  - $a_i$  – distanța medie față de instanțele din același grup
  - $b_i$  – distanța minimă față de orice instanță din orice alt grup
- Coeficientul de siluetă al instanței  $i$  este:
  - $s_i = (b_i - a_i) / \max(a_i, b_i) \in [-1, 1]$
- Gruparea este mai bună când  $s_i$  este mai mare, aproape de 1 ( $a_i \ll b_i$ )
- Coeficientul de siluetă al unui grup este media coeficientilor instanțelor

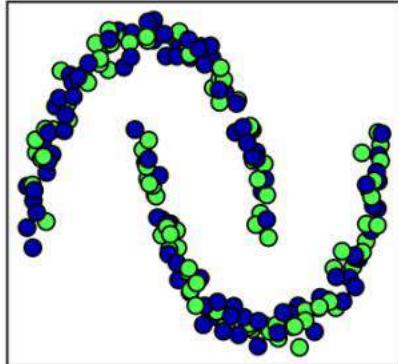
# Exemplu



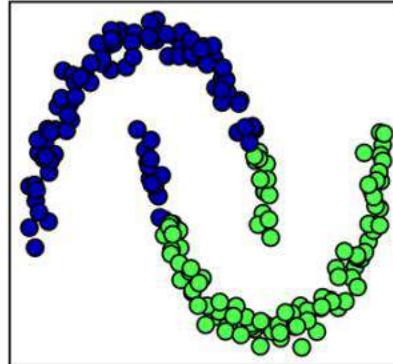
# Probleme

- Metoda coeficientului de siluetă este potrivită pentru algoritmul  $k$ -medii, dar nu reflectă calitatea partiționării și pentru grupuri de forme arbitrare

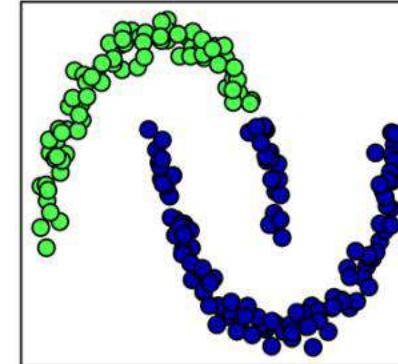
Random assignment: -0.00



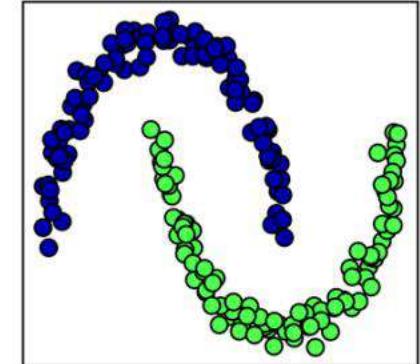
KMeans : 0.49



AgglomerativeClustering : 0.46

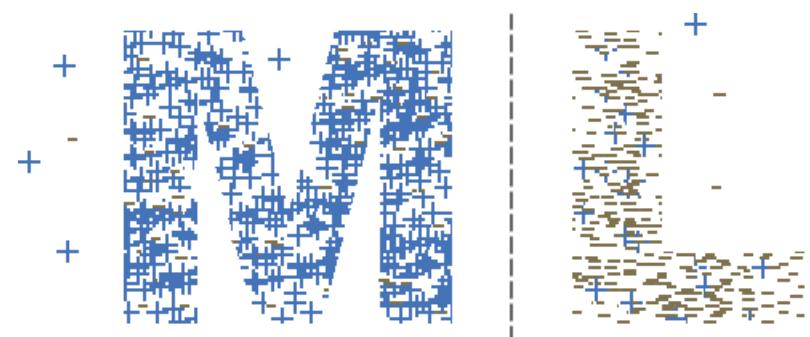


DBSCAN : 0.38



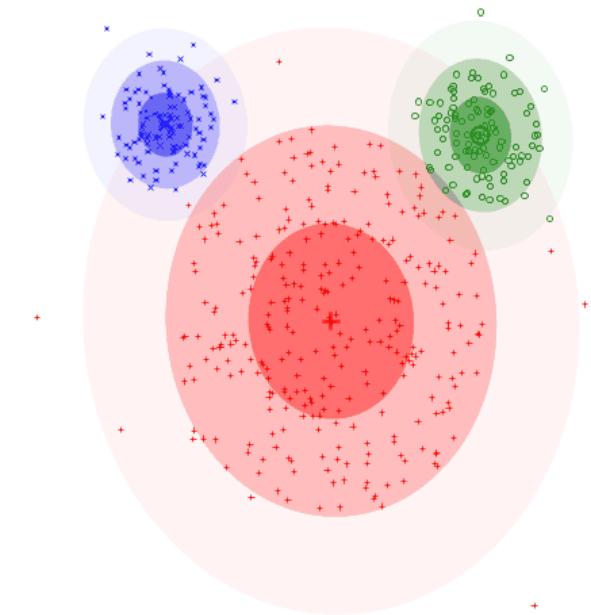
# Algoritmi de grupare (clustering)

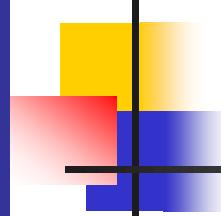
1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
- 3. Algoritmul EM (Expectation-Maximization)**
4. Gruparea ierarhică
5. Algoritmul DBSCAN



# Algoritmul *EM*

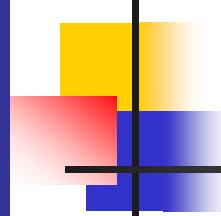
- Algoritmul *EM* (*Expectation-Maximization*, Așteptare-Maximizare) poate fi văzut ca o variantă flexibilă a algoritmului *k-medii*, în care grupurile nu mai sunt clar separate, ci o instanță aparține unui grup cu o anumită probabilitate
- O instanță aparține de fapt tuturor grupurilor, dar în diverse grade
- Dacă se consideră doar probabilitatea cea mai mare, *EM* se reduce la *k-medii*





# Funcție de densitate de probabilitate

- Care este probabilitatea de a ghici un număr natural generat aleatoriu uniform din intervalul [1, 5]?
- $x$  = numărul generat,  $y$  = numărul ghicit
- $P(y = x) = 1/5, \forall x \in \{ 1, 2, 3, 4, 5 \}$
- Care este probabilitatea de a ghici un număr real generat aleatoriu uniform din intervalul [1, 5]?
- Există o infinitate de numere reale în acest interval
- $P(y = x) = 0, \forall x \in [1, 5]$



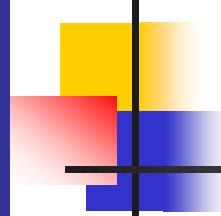
# Funcție de densitate de probabilitate

- Se poate calcula totuși probabilitatea ca  $y$  să aparțină unui subinterval
  - $P(y \in [1, 2]) = 1/4$
  - $P(y \in [1, 5]) = 1$
- În general:

$$P(x \in A) = \int_A f(x)dx$$

unde  $f$  este funcția de densitate de probabilitate, *FDP* (*Probability Density Function, PDF*)

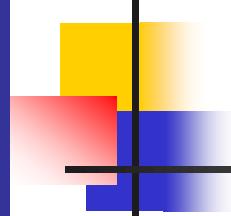
- Fiecare distribuție de probabilitate are o FDP specifică



# Funcție de densitate de probabilitate

- Fie  $\mathbf{y}$  un vector de observații sau eșantioane dintr-o populație (distribuție) necunoscută:  $\mathbf{y} = (y_1, \dots, y_m)$
- $\mathbf{y}$  reprezintă mulțimea de antrenare
- Dorim să estimăm cel mai bun model care ar putea reprezenta această populație
- Modelul are parametri:  $\mathbf{w} = (w_1, \dots, w_k)$
- *FDP* este  $f(\mathbf{y} | \mathbf{w})$ , unde componentele  $y_i$  sunt independente:

$$f(\mathbf{y} = (y_1, \dots, y_m) | \mathbf{w}) = f_1(y_1 | \mathbf{w}) \cdot \dots \cdot f_m(y_m | \mathbf{w})$$



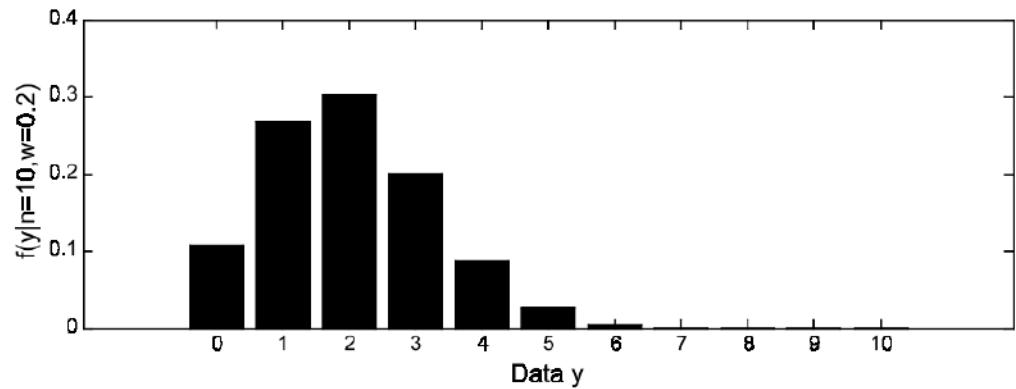
# Exemplu

- Să presupunem că aruncăm un ban de  $n$  ori, iar probabilitatea de succes a fiecărui experiment (de exemplu, obținerea unui „cap”) este  $w$
- Care este probabilitatea să obținem „cap” de  $y$  ori?
- Scenariu modelat de distribuția binomială (Bernoulli)
- Folosim funcția de densitate de probabilitate:

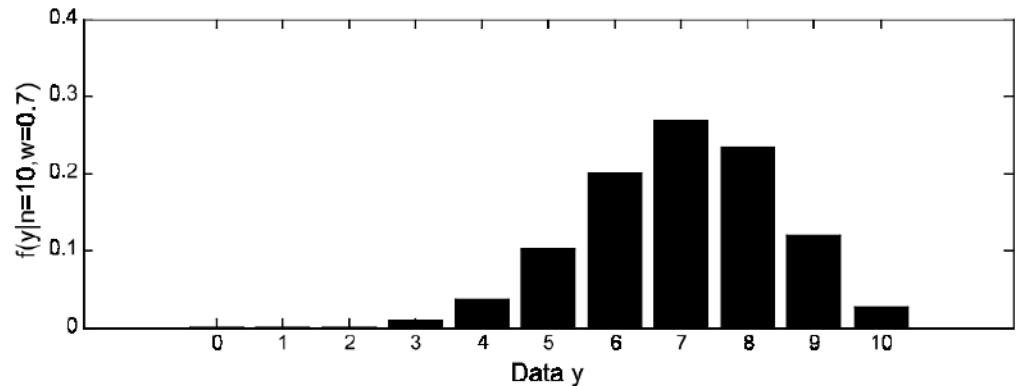
$$f(y|n, w) = \frac{n!}{y!(n-y)!} w^y (1-w)^{n-y}$$
$$(w \in [0, 1], y \in \{0, 1, \dots, n\})$$

# Exemplu

$$f(y|n=10, w=0.2) = \frac{10!}{y!(10-y)!} \cdot 0.2^y \cdot 0.8^{10-y}$$



$$f(y|n=10, w=0.7) = \frac{10!}{y!(10-y)!} \cdot 0.7^y \cdot 0.3^{10-y}$$



$y \in \mathbb{N}$

# Funcție de verosimilitate

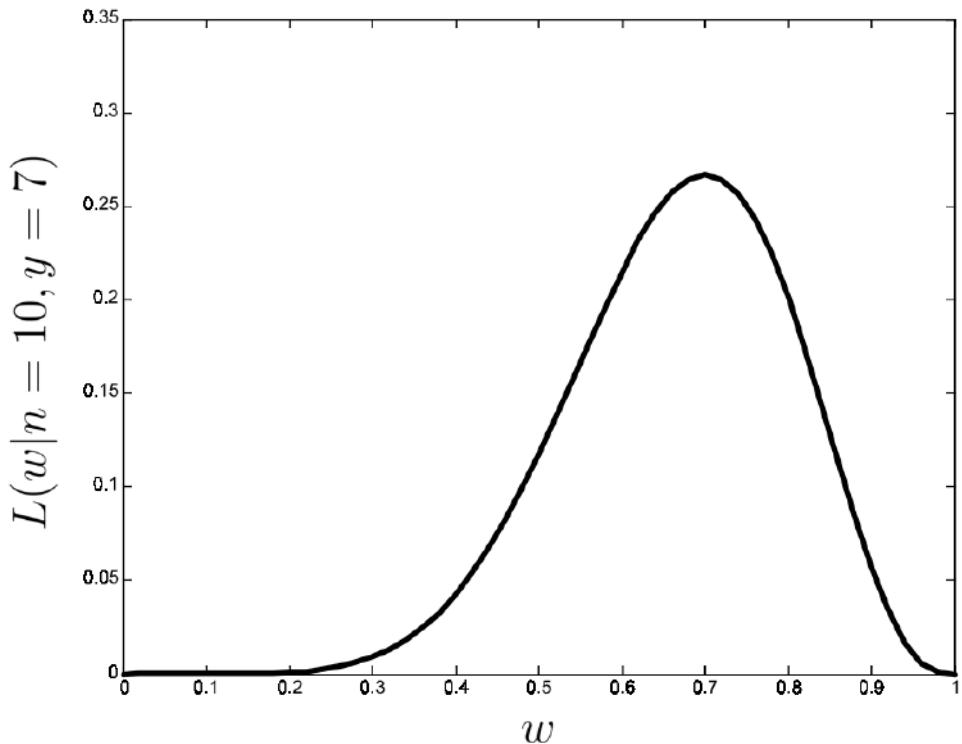
- engl. “likelihood”
- Dacă am obținut de 7 ori „cap” din 10 încercări, care este probabilitatea banului de a cădea „cap”?

$$L(w|y) = f(y|w)$$

$$L(w|n = 10, y = 7) =$$

$$f(y = 7|n = 10, w) =$$

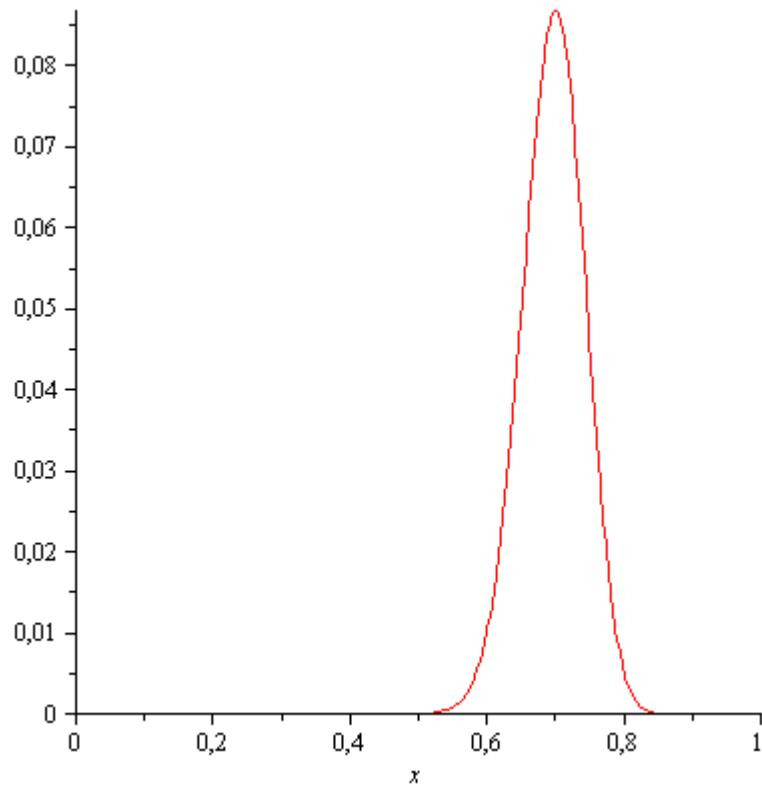
$$\frac{10!}{7! \cdot 3!} w^7 (1-w)^3$$



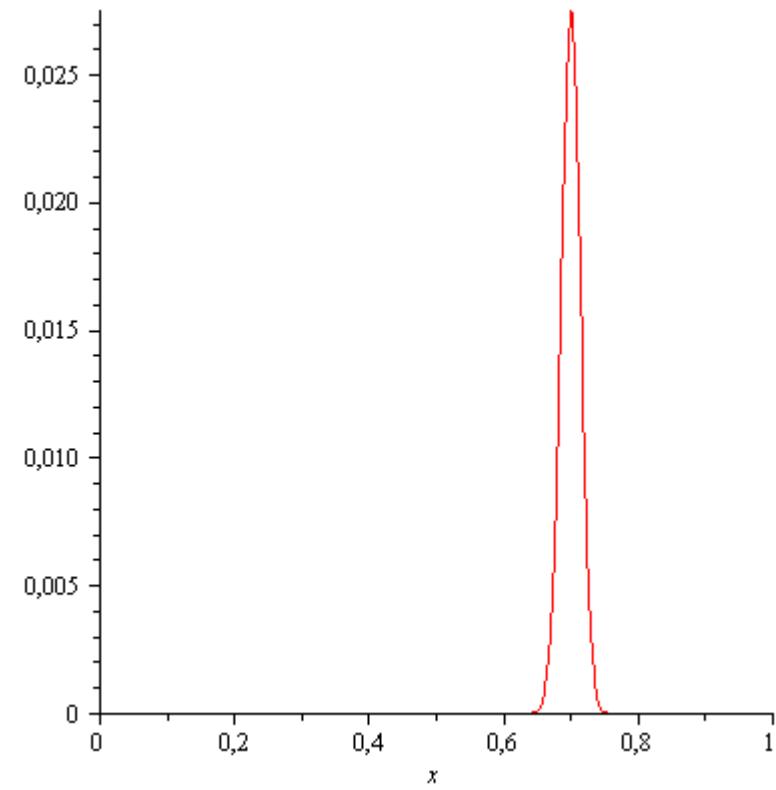
$y \in \mathbb{N}$ , dar  $w \in \mathbb{R}$

Când  $n$  este mare și proporția de apariție a „capului” ( $y/n$ ) este 70%, este foarte puțin probabil ca probabilitatea de succes ( $w$ ) să fie departată de 0.7

# Funcție de verosimilitate



$$n = 100$$



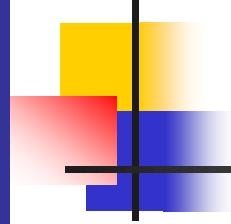
$$n = 1000$$

# Estimarea verosimilității maxime

- engl. “maximum likelihood estimation”, *MLE*
- Deoarece funcția  $L$  este un produs (slide-ul 43), se preferă logaritmarea sa
- Prin logaritmare, produsul devine sumă, dar maximul rămâne același

$$\frac{\partial \ln L(\mathbf{w}|\mathbf{y})}{\partial w_i} = 0 \quad \text{condiția de optim}$$

$$\frac{\partial^2 \ln L(\mathbf{w}|\mathbf{y})}{\partial w_i^2} < 0 \quad \text{condiția ca punctul de optim să fie punct de maxim, nu minim}$$



# Exemplu

$$\ln L(w|n=10, y=7) = \ln \frac{10!}{7! \cdot 3!} + 7 \ln w + 3 \ln(1-w)$$

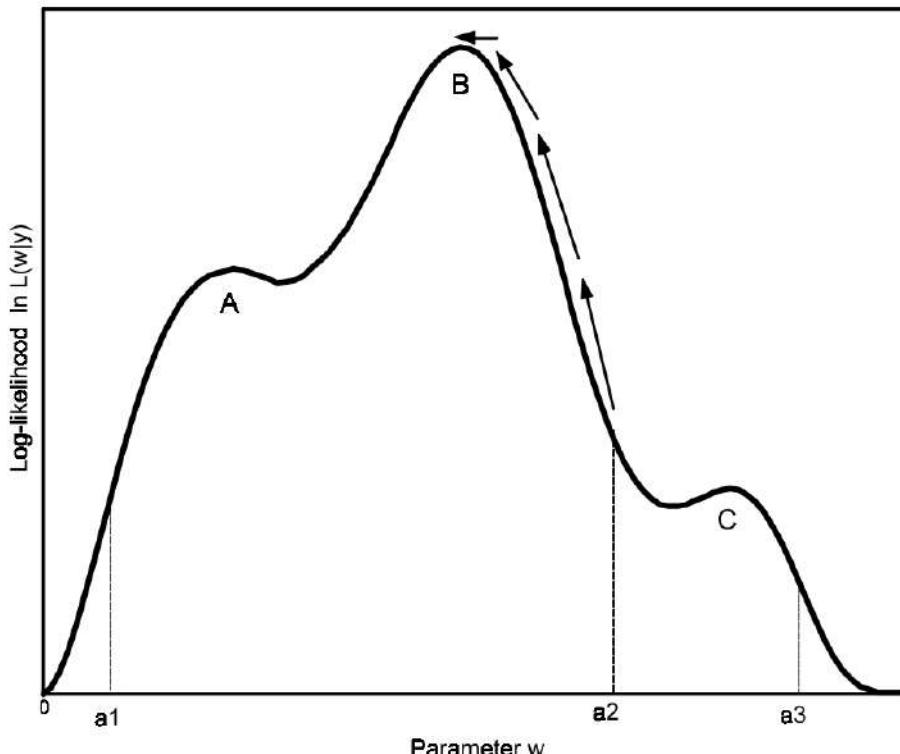
$$\frac{d \ln L(w|n=10, y=7)}{dw} = \frac{7}{w} - \frac{3}{1-w} = \frac{7-10w}{w(1-w)} = 0$$

$$\Rightarrow w_{MLE} = 0.7$$

$$\frac{d^2 \ln L(w|n=10, y=7)}{dw^2} \Big|_{w=w_{MLE}} = -\frac{7}{w^2} - \frac{3}{(1-w)^2} \Big|_{w=w_{MLE}} = -47.62 < 0$$

# Cazul general

- În general, funcția  $L$  are optime locale, iar rezolvarea este iterativă



# De la MLE la EM: MLE

- Să presupunem că facem 5 serii de experimente de aruncat banul cu 2 bani diferenți, A și B, fiecare serie cu 10 aruncări

B  
A  
A  
B  
A

H T T T H H T H T H	
H H H H T H H H H H	
H T H H H H H T H H	
H T H T T T H H T T	
T H H H T H H H T H	

5 sets, 10 tosses per set

Coin A	Coin B
	5 H, 5 T
9 H, 1 T	
8 H, 2 T	
	4 H, 6 T
7 H, 3 T	
24 H, 6 T	9 H, 11 T

$$\hat{\theta}_A = \frac{24}{24 + 6} = 0.80$$

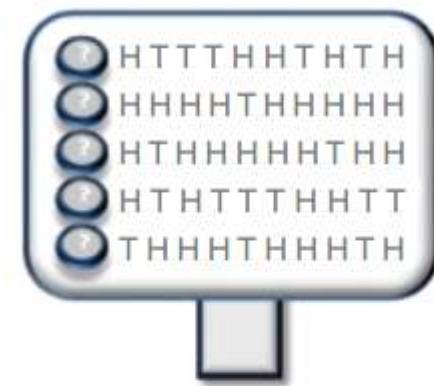
$$\hat{\theta}_B = \frac{9}{9 + 11} = 0.45$$

$$\hat{\theta}_A = \frac{\text{\# of heads using coin A}}{\text{total \# of flips using coin A}}$$

$$\hat{\theta}_B = \frac{\text{\# of heads using coin B}}{\text{total \# of flips using coin B}}$$

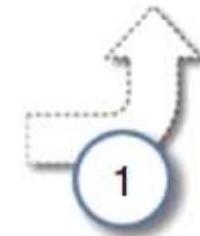
# De la *MLE* la *EM*: *EM*

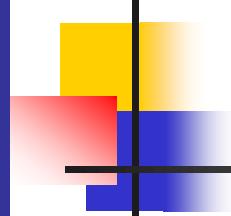
- Să presupunem acum că nu știm ce ban s-a folosit la fiecare serie de experimente
- Nu știm nici banul care a generat o serie, nici probabilitățile de succes  $\theta_j$  (parametrii)
- Pornim cu niște valori aleatorii pentru parametri



$$\hat{\theta}_A^{(0)} = 0.60$$

$$\hat{\theta}_B^{(0)} = 0.50$$





# EM, pasul 2

- Probabilitatea de a da de  $y$  ori „cap” din  $n$  încercări este:

$$f(y|n, w) = \frac{n!}{y!(n-y)!} w^y (1-w)^{n-y}$$
$$(w \in [0, 1], y \in \{0, 1, \dots, n\})$$

- În acest exemplu, avem  $\theta_A$  și  $\theta_B$  în loc de  $w$
- Primul factor (fractia) este identic pentru ambii bani, nu depinde de  $\theta_j$  și va dispărea la normalizare

# EM, pasul 2

$$f(y|n, w) = \frac{n!}{y!(n-y)!} w^y (1-w)^{n-y}$$

- Exemplu de calcul: seria 2

- 9 „cap”, 1 „pajură”
- $\theta_A = 0.6, \theta_B = 0.5$

$$\theta_A^9 (1 - \theta_A)^{10-9} \simeq 0.004$$

$$\frac{0.004}{0.004 + 0.001} = 0.8$$

$$\theta_B^9 (1 - \theta_B)^{10-9} \simeq 0.001$$

$$\frac{0.001}{0.004 + 0.001} = 0.2$$



H H H H T H H H H H

- Sirul observat este foarte debalansat (9H, 1T)
- Cu parametrii curenti, banul A este mai debalansat in favoarea lui H decat banul B, prin urmare este mai probabil ca sirul observat sa fi fost produs de A (cu probabilitatea 80%) decat de B (20%)

# EM, pasul 2

$$f(y|n, w) = \frac{n!}{y!(n-y)!} w^y (1-w)^{n-y}$$

## ■ Exemplu de calcul: seria 2

- 9 „cap”, 1 „pajură”
- $\theta_A = 0.6, \theta_B = 0.5$



H H H H T H H H H H

$$\theta_A^9 (1 - \theta_A)^{10-9} \simeq 0.004$$

$$\theta_B^9 (1 - \theta_B)^{10-9} \simeq 0.001$$

$$\frac{0.004}{0.004 + 0.001} = 0.8$$

$$\frac{0.001}{0.004 + 0.001} = 0.2$$

$$0.80 \times \text{A}$$

$$0.20 \times \text{B}$$

$$0.8 \cdot 9\text{H} = 7.2\text{H}$$

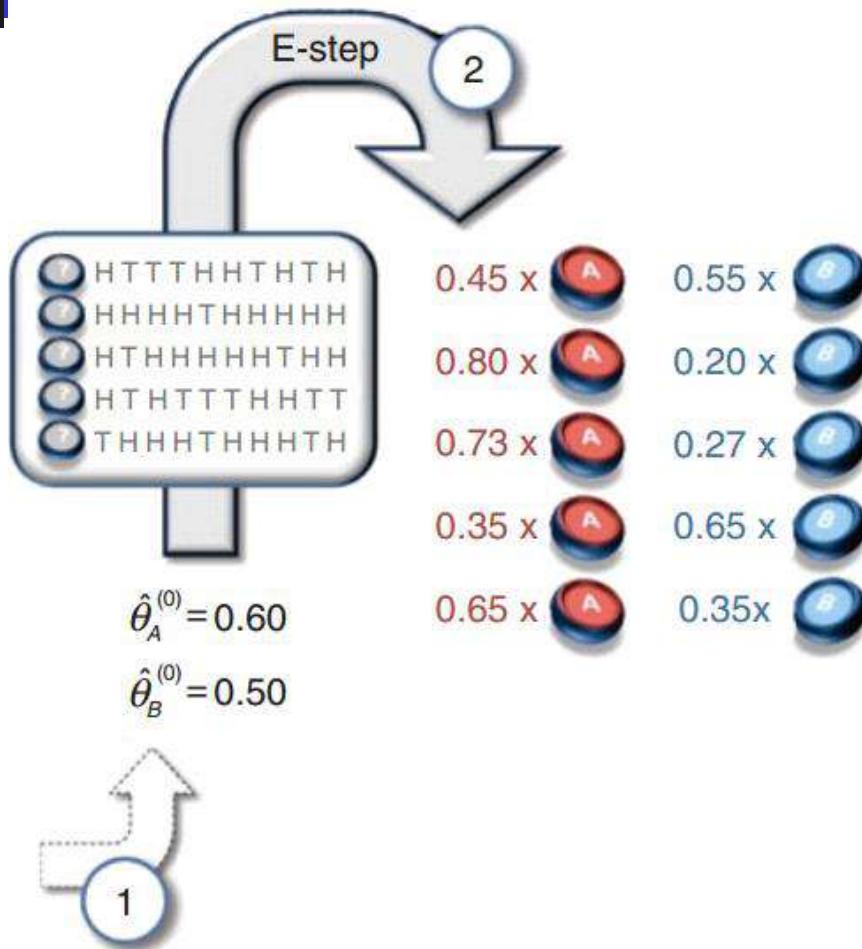
$$0.2 \cdot 1\text{T} = 0.2\text{T}$$

$\approx 7.2\text{ H}, 0.8\text{ T}$

$\approx 1.8\text{ H}, 0.2\text{ T}$

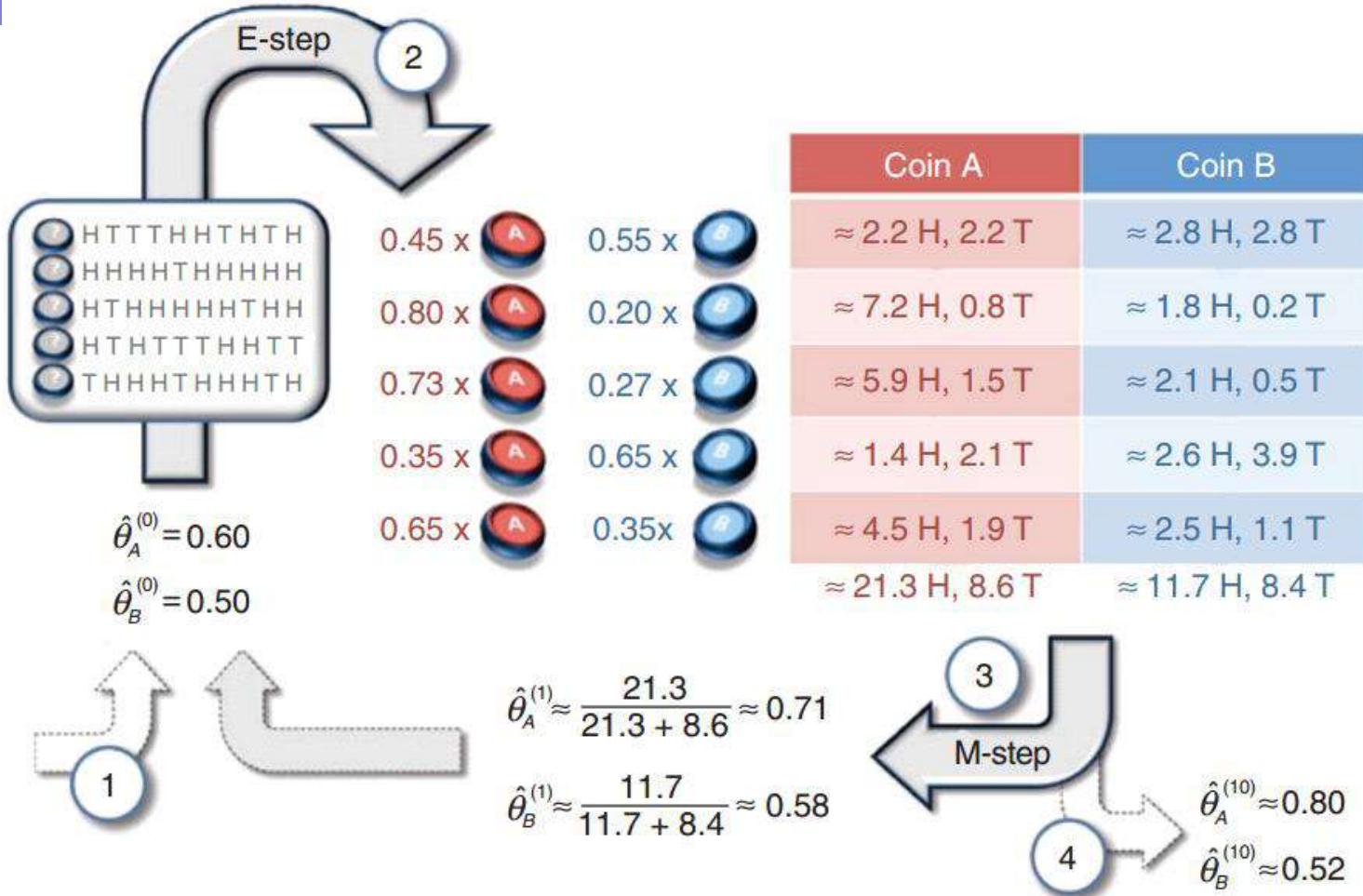
(valori așteptate)

# EM, pasul 2

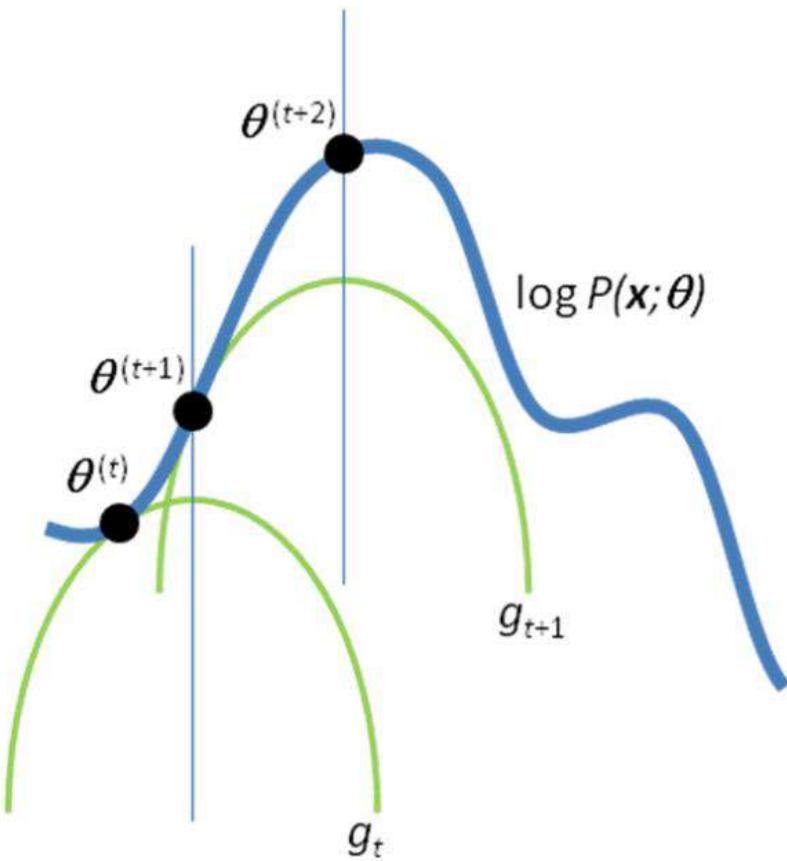


Coin A	Coin B
≈ 2.2 H, 2.2 T	≈ 2.8 H, 2.8 T
≈ 7.2 H, 0.8 T	≈ 1.8 H, 0.2 T
≈ 5.9 H, 1.5 T	≈ 2.1 H, 0.5 T
≈ 1.4 H, 2.1 T	≈ 2.6 H, 3.9 T
≈ 4.5 H, 1.9 T	≈ 2.5 H, 1.1 T
≈ 21.3 H, 8.6 T	≈ 11.7 H, 8.4 T

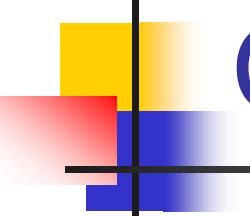
# EM



# Convergența algoritmului EM



- Se pleacă cu  $\theta^{(t)}$
- În pasul  $E$ , se construiește o funcție  $g_t$  care reprezintă o limită inferioară a funcției obiectiv  $\log P(x, \theta)$
- În pasul  $M$ , se calculează  $\theta^{(t+1)}$  care maximizează  $g_t$
- S.a.m.d.



# Gruparea cu algoritmul EM

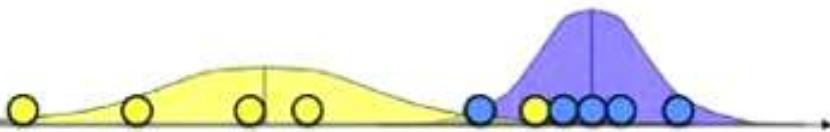
- În continuare, pentru prezentarea algoritmului de grupare EM, vom utiliza distribuția normală în locul distribuției binomiale

# Modele mixte 1D

- engl. "mixture models"
- Instanțe / observații:  $x_1, \dots, x_n$
- Exemplu: două gaussiene cu parametrii  $\mu_j$  și  $\sigma_j$  necunoscuți
- Dacă am ști cărei gaussiene îi aparține fiecare instanță, am putea calcula ușor  $\mu_j$  și  $\sigma_j$

$$\mu_j = \frac{x_1 + \dots + x_{n_j}}{n_j}$$

$$\sigma_j^2 = \frac{(x_1 - \mu_j)^2 + \dots + (x_{n_j} - \mu_j)^2}{n_j}$$

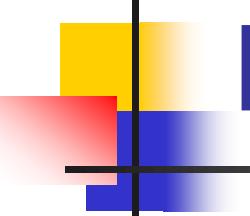


- Dacă am ști parametrii gaussienelor, am putea calcula probabilitățile de apartenență ale instanțelor la ele

$$P(C_j|x_i) = \frac{P(x_i|C_j) \cdot P(C_j)}{P(x_i)} = \frac{P(x_i|C_j) \cdot P(C_j)}{\sum_k P(x_i|C_k) \cdot P(C_k)}$$



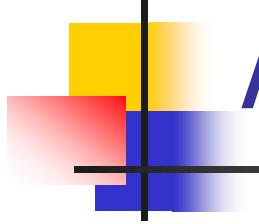
$$P(x_i|C_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_i - \mu_j)^2}{2\sigma_j^2}\right)$$


$$P(C_j|x_i) = \frac{P(x_i|C_j) \cdot P(C_j)}{P(x_i)} = \frac{P(x_i|C_j) \cdot P(C_j)}{\sum_k P(x_i|C_k) \cdot P(C_k)}$$

$$P(x_i|C_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_i - \mu_j)^2}{2\sigma_j^2}\right)$$

# Precizare

- În aceste formule, se folosește notația  $P(x_i | C_j)$  pentru verosimilitate, în conformitate cu teorema lui Bayes
- Dar ecuația 2 reprezintă funcția de densitate de probabilitate, care *nu este o probabilitate*. De exemplu, dacă  $\sigma$  este foarte mică, valoarea părții din dreapta poate fi mai mare decât 1!
- Probabilitatea reală  $P(x_i | C_j)$  este *proporțională* cu această valoare și rezultă prin normalizare, când sunt considerate toate grupurile



# Algoritmul *EM*: cazul 1D

- Gaussienele corespund grupurilor
- Se pornește cu valori aleatorii pentru parametrii grupurilor:  $\mu_j$  și  $\sigma_j$
- Pentru fiecare instanță  $x_i$  se calculează probabilitatea de a apartine grupului  $C_j$ :  $P(C_j | x_i)$
- Se ajustează  $\mu_j$  și  $\sigma_j$  pe baza instanțelor  $x_i$
- Se repetă până la convergența criteriului:

$$L = \sum_i \ln \left( \sum_j P(C_j) P(x_i | C_j) \right)$$

# Distribuție gaussiană 1D (univariată)

- Medie, deviație standard, varianță:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

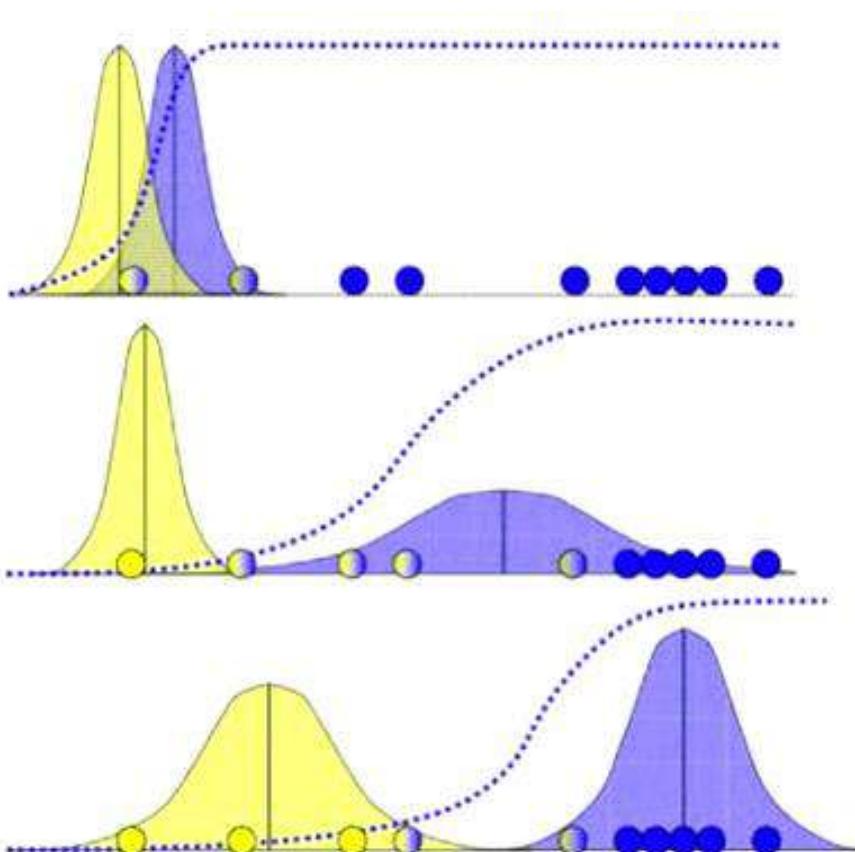
$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{n}}$$

/  $n$  → “biased”  
/  $(n - 1)$  → “unbiased”

$$var(\mathbf{x}) = \sigma^2 = \frac{\sum (x_i - \mu)^2}{n}$$

Din punct de vedere  
practic, diferență  
nesemnificativă

# Exemplu 1D



$$P(x_i|C_1) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x_i - \mu_1)^2}{2\sigma_1^2}\right)$$

$$w_i^1 = P(C_1|x_i) = \frac{P(x_i|C_1) \cdot P(C_1)}{P(x_i|C_1) \cdot P(C_1) + P(x_i|C_2) \cdot P(C_2)}$$

$$w_i^2 = P(C_2|x_i) = 1 - P(C_1|x_i)$$

$$\mu_j = \frac{w_1^j \cdot x_1 + \dots + w_n^j \cdot x_n}{w_1^j + \dots + w_n^j}$$

$$\sigma_j^2 = \frac{w_1^j \cdot (x_1 - \mu_j)^2 + \dots + w_n^j \cdot (x_n - \mu_j)^2}{w_1^j + \dots + w_n^j}$$

$$P(C_1) = \frac{w_1^1 + \dots + w_n^1}{n}$$

$$P(C_2) = 1 - P(C_1)$$

# Exemplu numeric 1D: $x = [1, 2, 4, 5]$

## Prima iteratie, pasul E

Initializare:  $\mu_1 = 1, \mu_2 = 2, \sigma_1 = 1, \sigma_2 = 1, P(c_1) = 0.5, P(c_2) = 0.5$

$$P(c_1|x_1) = \alpha_1 \cdot P(c_1) \cdot P(x_1|c_1) = \alpha_1 \cdot 0.5000 \cdot 0.3989 = \alpha_1 \cdot 0.1995$$

$$P(c_1|x_2) = \alpha_2 \cdot P(c_1) \cdot P(x_2|c_1) = \alpha_2 \cdot 0.5000 \cdot 0.2420 = \alpha_2 \cdot 0.1210$$

$$P(c_1|x_3) = \alpha_3 \cdot P(c_1) \cdot P(x_3|c_1) = \alpha_3 \cdot 0.5000 \cdot 0.0044 = \alpha_3 \cdot 0.0022$$

$$P(c_1|x_4) = \alpha_4 \cdot P(c_1) \cdot P(x_4|c_1) = \alpha_4 \cdot 0.5000 \cdot 0.0001 = \alpha_4 \cdot 0.0001$$

$$P(c_2|x_1) = \alpha_1 \cdot P(c_2) \cdot P(x_1|c_2) = \alpha_1 \cdot 0.5000 \cdot 0.2420 = \alpha_1 \cdot 0.1210$$

$$P(c_2|x_2) = \alpha_2 \cdot P(c_2) \cdot P(x_2|c_2) = \alpha_2 \cdot 0.5000 \cdot 0.3989 = \alpha_2 \cdot 0.1995$$

$$P(c_2|x_3) = \alpha_3 \cdot P(c_2) \cdot P(x_3|c_2) = \alpha_3 \cdot 0.5000 \cdot 0.0540 = \alpha_3 \cdot 0.0270$$

$$P(c_2|x_4) = \alpha_4 \cdot P(c_2) \cdot P(x_4|c_2) = \alpha_4 \cdot 0.5000 \cdot 0.0044 = \alpha_4 \cdot 0.0022$$

$$P(c_1|x_1) = 0.1995 / (0.1995 + 0.1210) = 0.6225$$

$$P(c_1|x_2) = 0.1210 / (0.1210 + 0.1995) = 0.3775$$

$$P(c_1|x_3) = 0.0022 / (0.0022 + 0.0270) = 0.0759$$

$$P(c_1|x_4) = 0.0001 / (0.0001 + 0.0022) = 0.0293$$

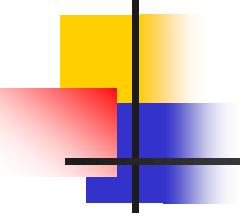
$$P(c_2|x_1) = 0.1210 / (0.1995 + 0.1210) = 0.3775$$

$$P(c_2|x_2) = 0.1995 / (0.1210 + 0.1995) = 0.6225$$

$$P(c_2|x_3) = 0.0270 / (0.0022 + 0.0270) = 0.9241$$

$$P(c_2|x_4) = 0.0022 / (0.0001 + 0.0022) = 0.9707$$

$$L = -11.891549$$



# Exemplu numeric 1D: $x = [1, 2, 4, 5]$

## Prima iteratie, pasul M

$$P(c_1) = (0.6225 + 0.3775 + 0.0759 + 0.0293)/4 = 1.1052/4 = 0.2763$$

$$P(c_2) = (0.3775 + 0.6225 + 0.9241 + 0.9707)/4 = 2.8948/4 = 0.7237$$

$$\mu_1 = (1 \cdot 0.6225 + 2 \cdot 0.3775 + 4 \cdot 0.0759 + 5 \cdot 0.0293)/1.1052 = 1.6536$$

$$\mu_2 = (1 \cdot 0.3775 + 2 \cdot 0.6225 + 4 \cdot 0.9241 + 5 \cdot 0.9707)/2.8948 = 3.5140$$

$$\sigma_1^2 = (0.6225 \cdot (1 - 1.6536)^2 + 0.3775 \cdot (2 - 1.6536)^2 + 0.0759 \cdot (4 - 1.6536)^2 + 0.0293 \cdot (5 - 1.6536)^2)/1.1052 = 0.9565$$

$$\sigma_2^2 = (0.3775 \cdot (1 - 3.5140)^2 + 0.6225 \cdot (2 - 3.5140)^2 + 0.9241 \cdot (4 - 3.5140)^2 + 0.9707 \cdot (5 - 3.5140)^2)/2.8948 = 2.1330$$

$$\sigma_1 = 0.9780$$

$$\sigma_2 = 1.4605$$

# Exemplu numeric 1D: $x = [1, 2, 4, 5]$

## Ultima iterare (9), pasul E

$$P(c_1|x_1) = \alpha_1 \cdot P(c_1) \cdot P(x_1|c_1) = \alpha_1 \cdot 0.5000 \cdot 0.4839 = \alpha_1 \cdot 0.2420$$

$$P(c_1|x_2) = \alpha_2 \cdot P(c_1) \cdot P(x_2|c_1) = \alpha_2 \cdot 0.5000 \cdot 0.4839 = \alpha_2 \cdot 0.2420$$

$$P(c_1|x_3) = \alpha_3 \cdot P(c_1) \cdot P(x_3|c_1) = \alpha_3 \cdot 0.5000 \cdot 0.0000 = \alpha_3 \cdot 0.0000$$

$$P(c_1|x_4) = \alpha_4 \cdot P(c_1) \cdot P(x_4|c_1) = \alpha_4 \cdot 0.5000 \cdot 0.0000 = \alpha_4 \cdot 0.0000$$

$$P(c_2|x_1) = \alpha_1 \cdot P(c_2) \cdot P(x_1|c_2) = \alpha_1 \cdot 0.5000 \cdot 0.0000 = \alpha_1 \cdot 0.0000$$

$$P(c_2|x_2) = \alpha_2 \cdot P(c_2) \cdot P(x_2|c_2) = \alpha_2 \cdot 0.5000 \cdot 0.0000 = \alpha_2 \cdot 0.0000$$

$$P(c_2|x_3) = \alpha_3 \cdot P(c_2) \cdot P(x_3|c_2) = \alpha_3 \cdot 0.5000 \cdot 0.4840 = \alpha_3 \cdot 0.2420$$

$$P(c_2|x_4) = \alpha_4 \cdot P(c_2) \cdot P(x_4|c_2) = \alpha_4 \cdot 0.5000 \cdot 0.4839 = \alpha_4 \cdot 0.2420$$

$$P(c_1|x_1) = 0.2420 / (0.2420 + 0.0000) = 1.0000$$

$$P(c_1|x_2) = 0.2420 / (0.2420 + 0.0000) = 1.0000$$

$$P(c_1|x_3) = 0.0000 / (0.0000 + 0.2420) = 0.0000$$

$$P(c_1|x_4) = 0.0000 / (0.0000 + 0.2420) = 0.0000$$

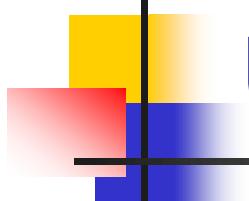
$$P(c_2|x_1) = 0.0000 / (0.2420 + 0.0000) = 0.0000$$

$$P(c_2|x_2) = 0.0000 / (0.2420 + 0.0000) = 0.0000$$

$$P(c_2|x_3) = 0.2420 / (0.0000 + 0.2420) = 1.0000$$

$$P(c_2|x_4) = 0.2420 / (0.0000 + 0.2420) = 1.0000$$

$$L = -5.675742$$



# Exemplu numeric 1D: $x = [1, 2, 4, 5]$

## Ultima iterare (9), pasul M

$$P(c_1) = (1 + 1 + 0 + 0)/4 = 2/4 = 0.5$$

$$P(c_2) = (0 + 0 + 1 + 1)/4 = 2/4 = 0.5$$

$$\mu_1 = (1 \cdot 1 + 2 \cdot 1 + 4 \cdot 0 + 5 \cdot 0)/2 = 1.5$$

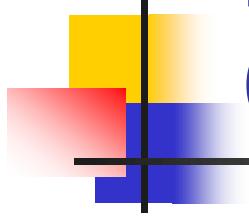
$$\mu_2 = (1 \cdot 0 + 2 \cdot 0 + 4 \cdot 1 + 5 \cdot 1)/2 = 4.5$$

$$\sigma_1^2 = (1 \cdot (1 - 1.5)^2 + 1 \cdot (2 - 1.5)^2 + 0 \cdot (4 - 1.5)^2 + 0 \cdot (5 - 1.5)^2)/2 = 0.25$$

$$\sigma_2^2 = (0 \cdot (1 - 4.5)^2 + 0 \cdot (2 - 4.5)^2 + 1 \cdot (4 - 4.5)^2 + 1 \cdot (5 - 4.5)^2)/2 = 0.25$$

$$\sigma_1 = 0.5$$

$$\sigma_2 = 0.5$$



# Distribuție gaussiană multidimensională (multivariată): matricea de covarianță

$$var(\mathbf{x}) = \frac{\sum (x_i - \mu)^2}{n}$$

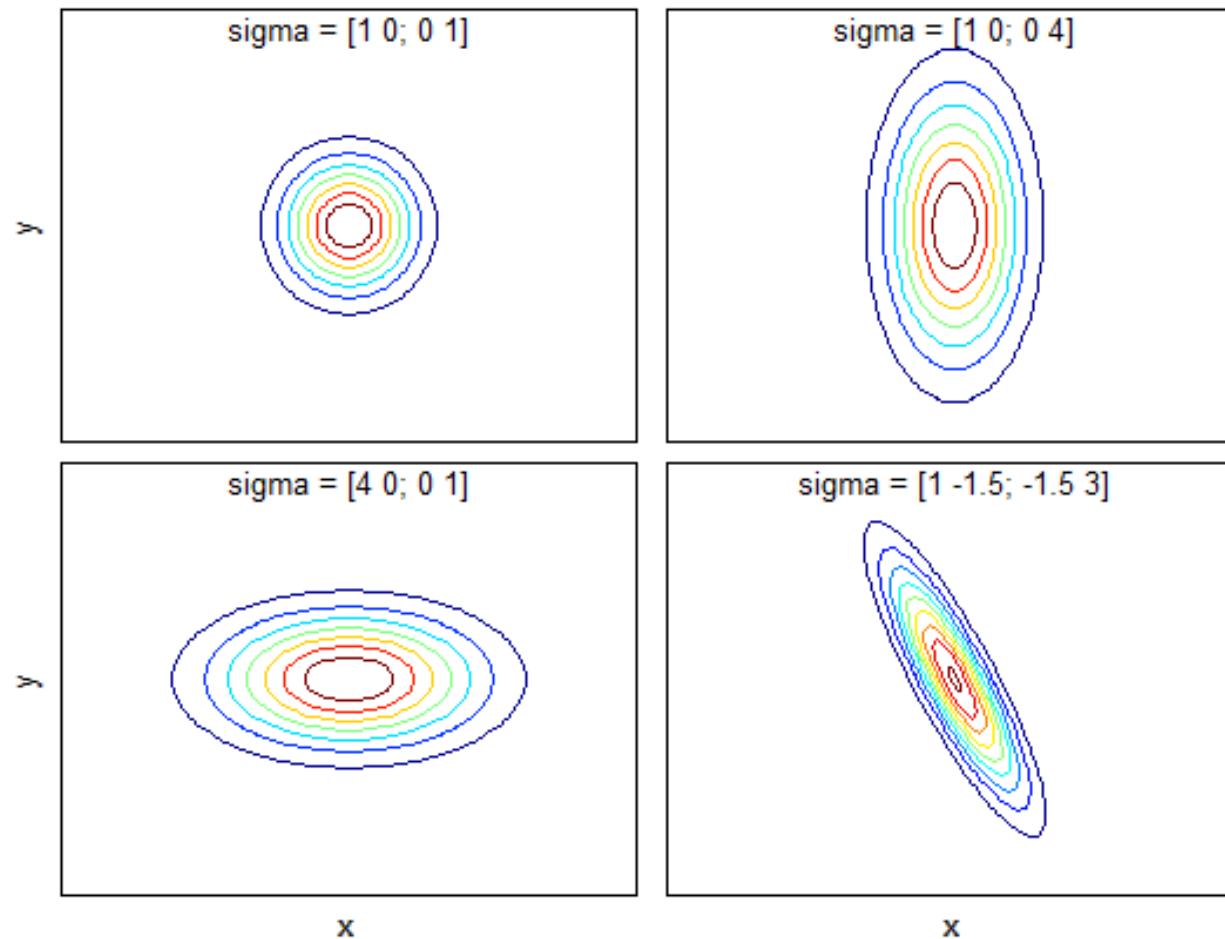
$$cov(\mathbf{x}, \mathbf{y}) = \frac{\sum (x_i - \mu_x)(y_i - \mu_y)}{n}$$

$C^{n \times n} = (c_{i,j}, \ c_{i,j} = cov(Dim_i, Dim_j)),$

$$C = \begin{pmatrix} cov(x, x) & cov(x, y) & cov(x, z) \\ cov(y, x) & cov(y, y) & cov(y, z) \\ cov(z, x) & cov(z, y) & cov(z, z) \end{pmatrix}$$

$cov(a, b) = cov(b, a)$   
(matricea este simetrică)

# Matrice de covarianță 2D: exemple



# Distribuție gaussiană multidimensională

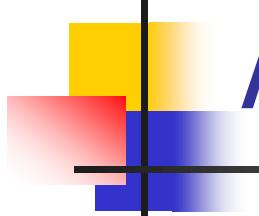
- Funcția de densitate de probabilitate 1D:

$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

- Funcția de densitate de probabilitate  $n$ D:

$$f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

$\mathbf{x}$  și  $\boldsymbol{\mu}$  sunt vectori,  $|\boldsymbol{\Sigma}|$  este determinantul matricei de covarianță



# Algoritmul *EM* general

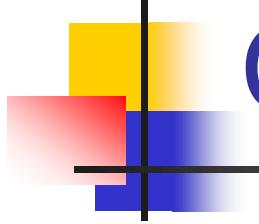
- **Pasul E:**  $P(C_j|x_i) = \frac{P(C_j) \cdot P(x_i|C_j)}{\sum_k P(C_k) \cdot P(x_i|C_k)}$

- **Pasul M:**  $P(C_j) = \frac{1}{n} \cdot \sum_{i=1}^n P(C_j|x_i)$

$$\mu_j = \frac{\sum_i P(C_j|x_i) \cdot x_i}{\sum_i P(C_j|x_i)}$$

$$\Sigma_j = \frac{\sum_i P(C_j|x_i) \cdot [(x_i - \mu_j) \cdot (x_i - \mu_j)^T]}{\sum_i P(C_j|x_i)}$$

Probabilitățile  $P(C_j | x_i)$  acționează ca niște ponderi



# Considerente practice

- Trebuie evitată situația în care  $|\Sigma| = 0$ 
  - Se pot adăuga niște numere mici la diagonala principală sau la elementele egale cu 0
- Pentru a evita un număr mare de parametri, dimensiunile se pot considera independente
  - Matricea  $\Sigma$  va fi o matrice diagonală

# Exemplu numeric 2D: $x = [(1,1), (2,1), (4,3), (5,4)]$

## Ultima iterare (5)

$$P(c_1) = (1 + 1 + 0 + 0)/4 = 2/4 = 0.5$$

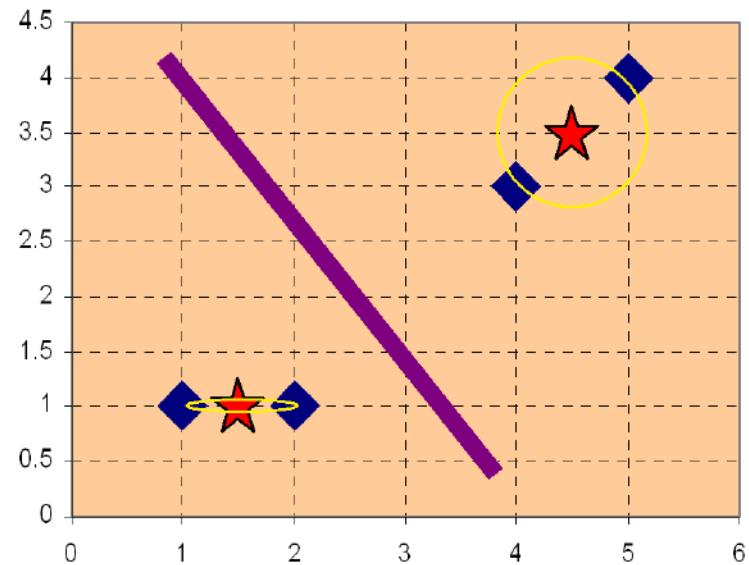
$$P(c_2) = (0 + 0 + 1 + 1)/4 = 2/4 = 0.5$$

$$\mu_1 = ((1,1) \cdot 1 + (2,1) \cdot 1 + (4,3) \cdot 0 + (5,4) \cdot 0)/2 = (1.5, 1)$$

$$\mu_2 = ((1,1) \cdot 0 + (2,1) \cdot 0 + (4,3) \cdot 1 + (5,4) \cdot 1)/2 = (4.5, 3.5)$$

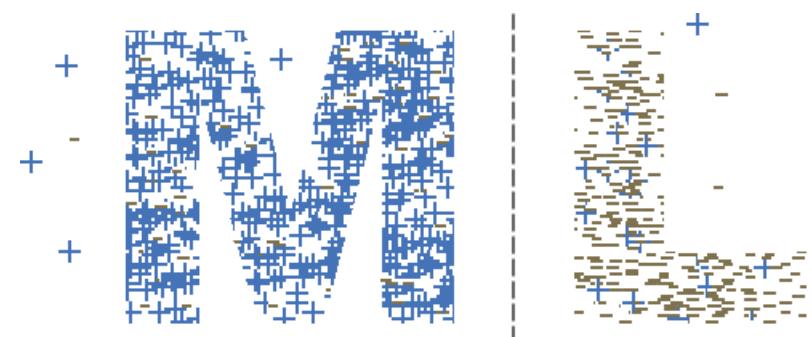
$$\Sigma_1 = (0.2500, 0.0001)$$

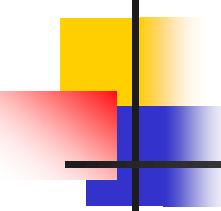
$$\Sigma_2 = (0.2500, 0.2500)$$



# Algoritmi de grupare (clustering)

1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
3. Algoritmul EM (Expectation-Maximization)
- 4. Gruparea ierarhică**
5. Algoritmul DBSCAN

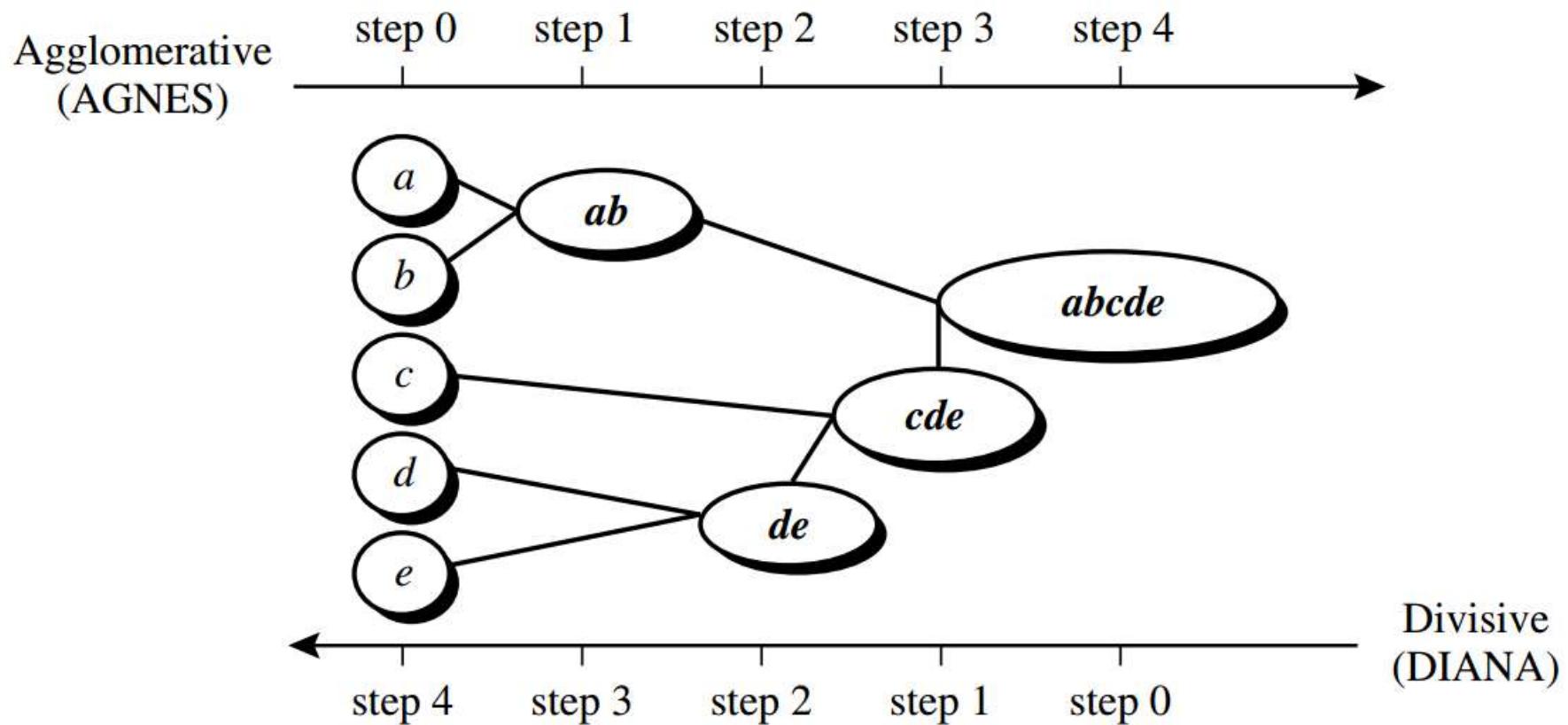




# Gruparea ierarhică

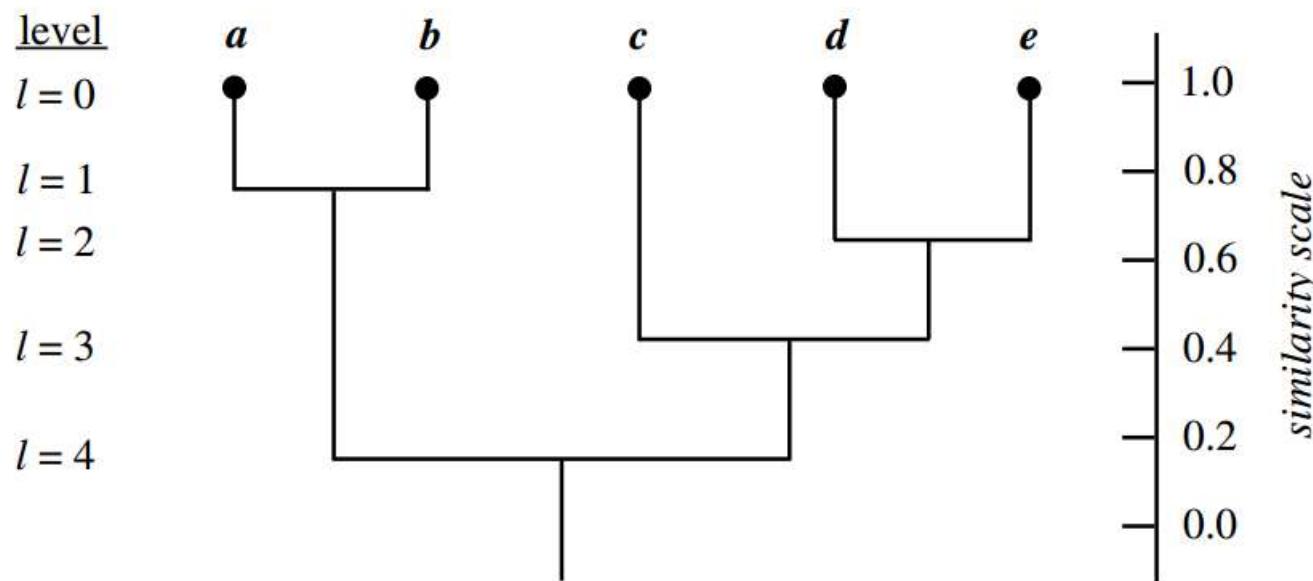
- Grupare aglomerativă
  - Strategie *bottom-up*: inițial fiecare punct reprezintă un grup și apoi se combină acestea în grupuri din ce în ce mai mari
  - De exemplu: Agglomerative Nesting, AGNES
  - Cele mai multe metode ierarhice sunt de acest tip
- Grupare divizivă
  - Strategie *top-down*: inițial toate punctele aparțin unui singur grup și acesta este partit ionat treptat în grupuri mai mici
  - De exemplu: Divisive Analysis, DIANA
  - Sau aplicarea repetată a algoritmului *k*-medii cu  $k = 2$
- Utilizatorul trebuie să specifice un criteriu de terminare, de exemplu numărul de grupuri dorit, un prag impus asupra diametrelor grupurilor etc.

# Gruparea ierarhică



# Dendrograma

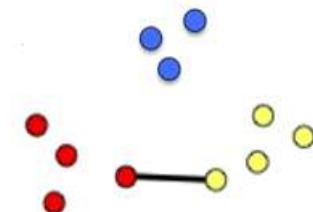
- Indică succesiunea de combinări sau partitioнări din gruparea ierarhică
- Axele pot fi și inverseate, cu similaritatea pe axa x



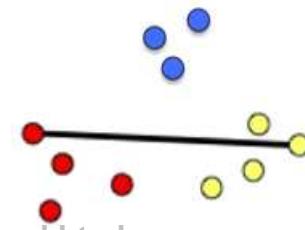
# Metode de combinare pentru gruparea aglomerativă

- La un moment dat, se unesc două grupuri care **minimizează** unul din următoarele criterii:
  - Legătură simplă (*single link*): distanța minimă între oricare două instanțe. Produce „lanțuri” lungi de grupuri
  - Legătură completă (*complete link*): distanța maximă între oricare două instanțe. Produce grupuri sferice

$$D(C_1, C_2) = \min_{x_1 \in C_1, x_2 \in C_2} D(x_1, x_2)$$



$$D(C_1, C_2) = \max_{x_1 \in C_1, x_2 \in C_2} D(x_1, x_2)$$

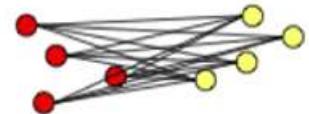


# Metode de combinare pentru gruparea aglomerativă

- Criterii (continuare):

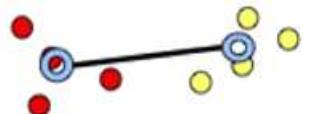
- Legătură medie (*average link*): media tuturor perechilor de distanțe. Scade sensibilitatea la valori extreme

$$D(C_1, C_2) = \frac{1}{n_1 n_2} \sum_{x_1 \in C_1} \sum_{x_2 \in C_2} D(x_1, x_2)$$



- Centroizi: distanța între centrele grupurilor

$$D(C_1, C_2) = D \left( \left( \frac{1}{n_1} \sum_{x_1 \in C_1} x_1 \right), \left( \frac{1}{n_2} \sum_{x_2 \in C_2} x_2 \right) \right)$$



# Metode de combinare pentru gruparea aglomerativă

- Criterii (continuare):

- Metoda Ward: costul de combinare  $\Delta$  pentru două grupuri

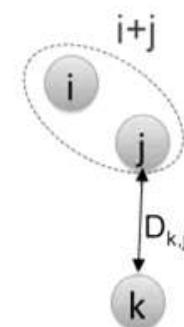
$$\begin{aligned}\Delta(A, B) &= \sum_{i \in A \cup B} \|\vec{x}_i - \vec{m}_{A \cup B}\|^2 - \sum_{i \in A} \|\vec{x}_i - \vec{m}_A\|^2 - \sum_{i \in B} \|\vec{x}_i - \vec{m}_B\|^2 \\ &= \frac{n_A n_B}{n_A + n_B} \|\vec{m}_A - \vec{m}_B\|^2\end{aligned}$$

- $x_i$  instanțele,  $m_j$  centrele,  $n_j$  numărul de puncte din grup
  - În gruparea ierarhică aglomerativă, SSE este initial 0 și apoi crește pe măsură ce se unesc grupurile. Metoda Ward încearcă să minimizeze această creștere
  - Dintre două perechi de grupuri cu centrele egal depărtate, metoda Ward combină grupurile mai mici

# Algoritmul Lance-Williams

- $D = \{D_{i,j} : \text{distance between } x_i \text{ and } x_j \text{ for } i,j=1..N\}$
- for N iterations:
  - i,j = **arg min**  $D_{i,j}$  ... pair of closest clusters
  - add cluster: i+j, delete clusters i, j
  - for each remaining cluster k:

$$D_{k,i+j} = \alpha_i D_{k,i} + \alpha_j D_{k,j} + \beta D_{i,j} + \gamma |D_{k,i} - D_{k,j}|$$



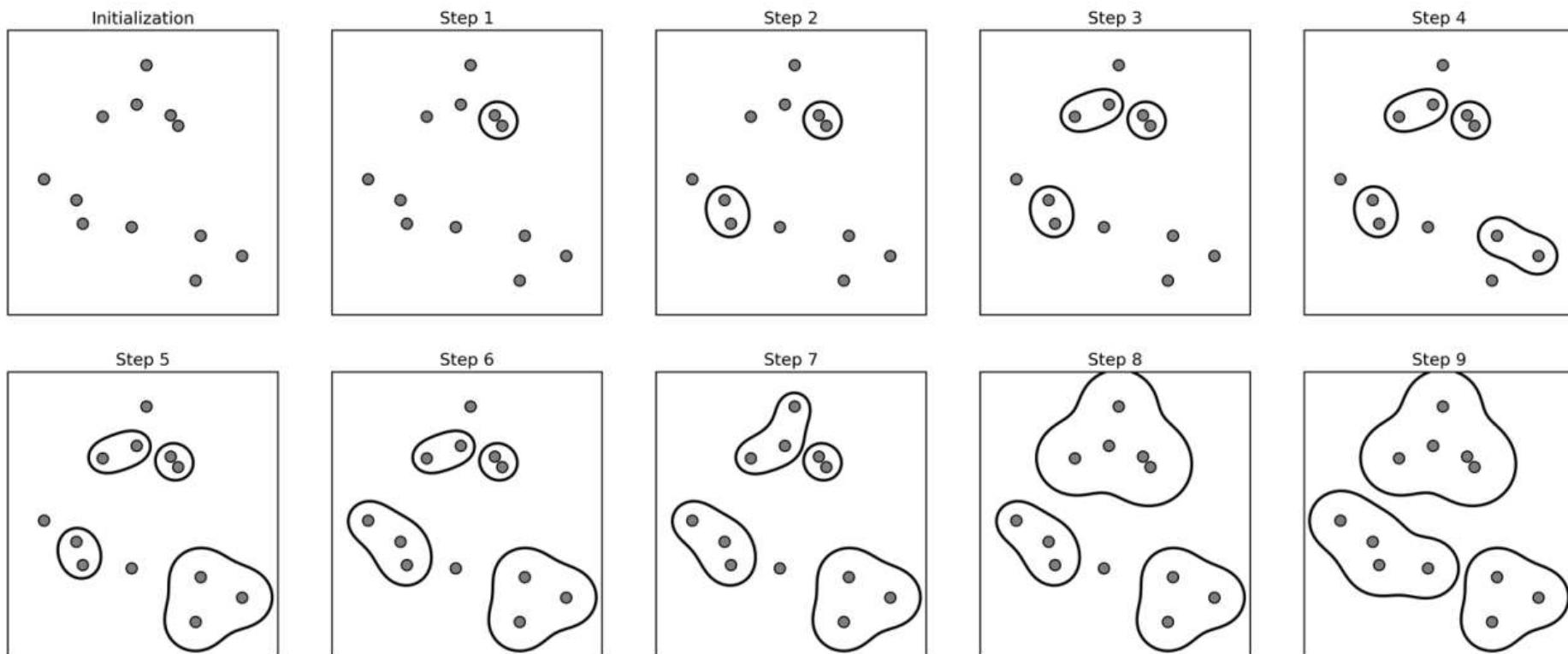
Method	$\alpha_i$	$\alpha_j$	$\beta$	$\gamma$
Single linkage	0.5	0.5	0	-0.5
Complete linkage	0.5	0.5	0	0.5
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Weighted group average	0.5	0.5	0	0
Centroid	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward	$\frac{n_i}{(n_i+n_j+n_k)}$	$\frac{n_j}{(n_i+n_j+n_k)}$	$\frac{-n_k}{(n_i+n_j+n_k)}$	0

**Single link:**

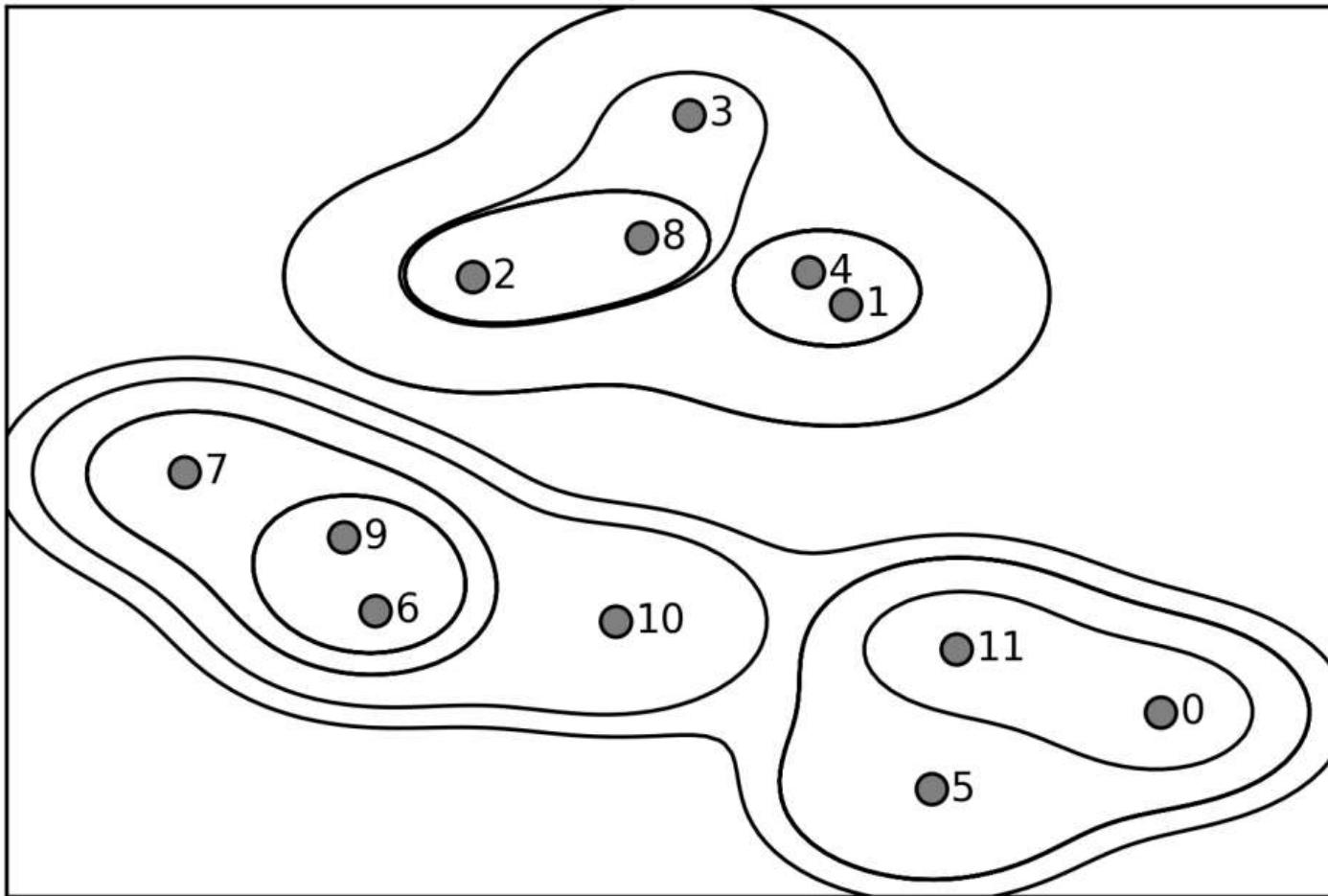
$$\begin{aligned} D_{k,i+j} &= \frac{1}{2} (D_{ki} + D_{kj} - |D_{ki} - D_{kj}|) \\ &= \min \{D_{ki}, D_{kj}\} \end{aligned}$$

$$\min(a, b) = \frac{(a+b) - |a-b|}{2}$$

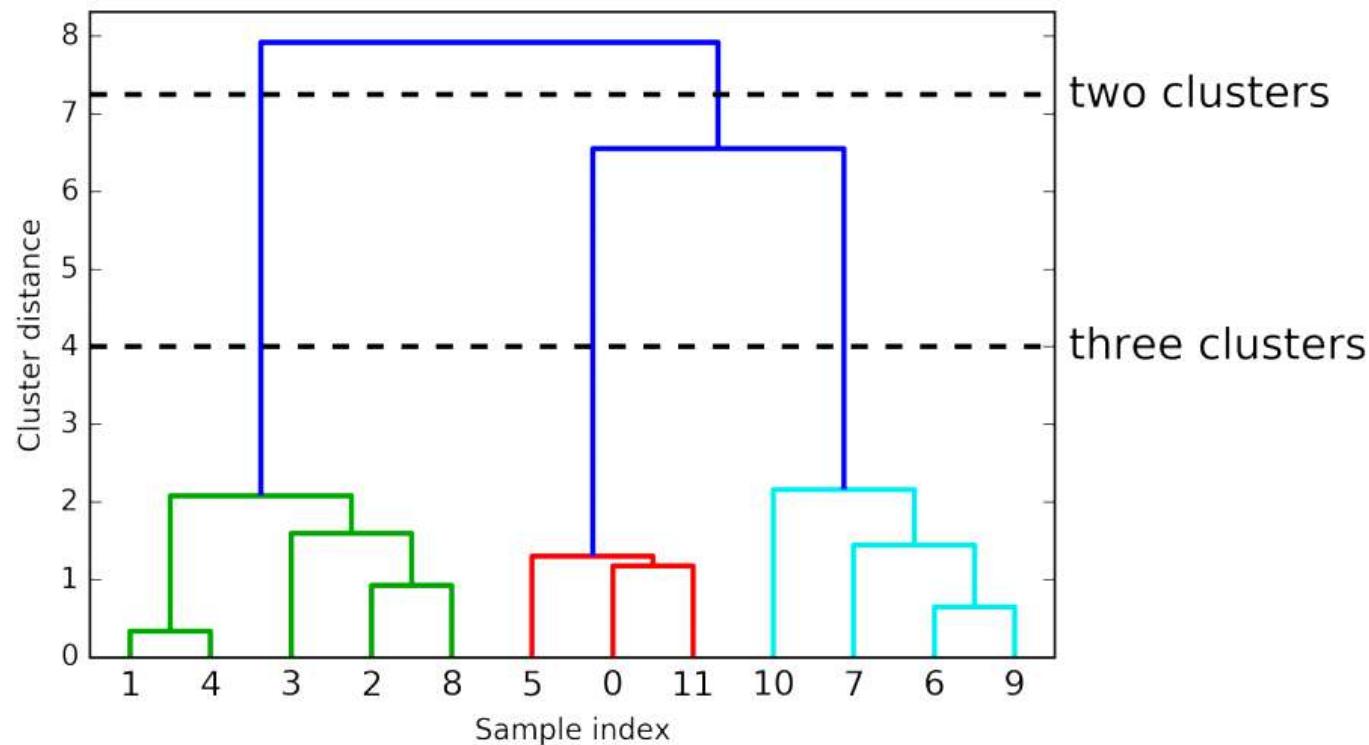
# Exemplu: grupare aglomerativă



# Exemplu: grupare aglomerativă



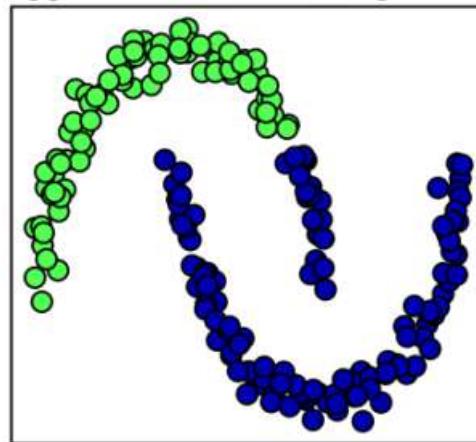
# Exemplu: dendrograma



Lungimea fiecărei ramuri pe axa y arată distanța dintre grupurile combinate. Pentru a ajunge de la 3 la 2 grupuri, s-au unit niște puncte foarte depărtate.

# Probleme

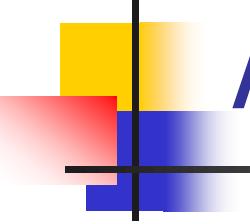
- Deciziile de combinare sunt critice deoarece procesul este *greedy* – nu permite revizuirea unor decizii deja luate
- Această metodă de grupare aglomerativă nu scalează bine, deoarece presupune analiza unui număr mare de instanțe sau grupuri
- Metoda nu reușește să trateze corect unele probleme complexe



# Algoritmi de grupare (clustering)

1. Învățarea și învățarea automată
2. Algoritmul k-medii (k-means)
3. Algoritmul EM (Expectation-Maximization)
4. Gruparea ierarhică
5. **Algoritmul DBSCAN**





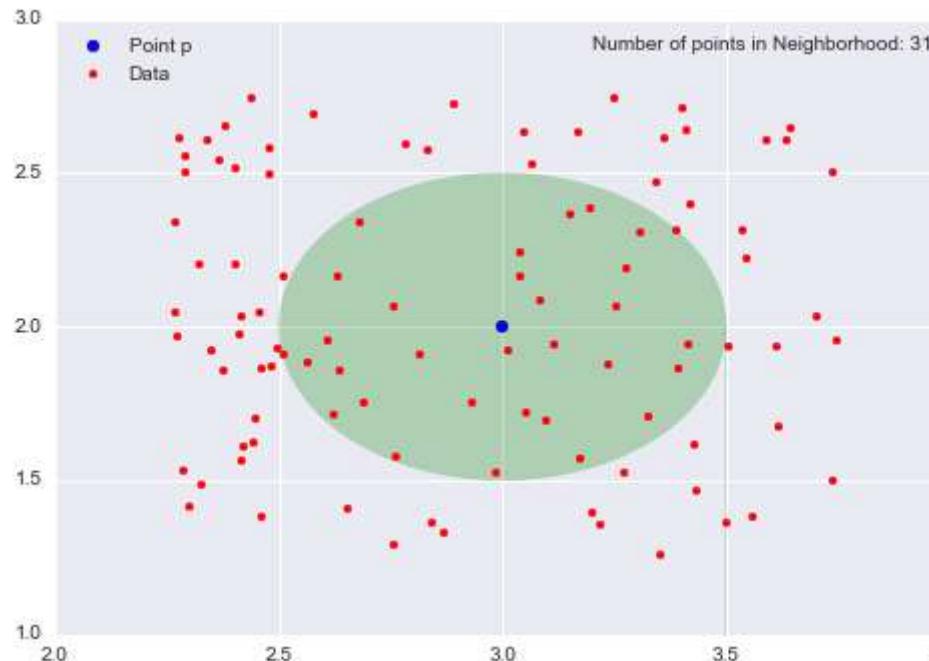
# Algoritmul DBSCAN

- engl. “Density-Based Spatial Clustering of Applications with Noise”
- Parametri:
  - $Eps$ : raza maximă a vecinătății
  - $MinPts$ : numărul minim de puncte din vecinătatea  $Eps$  a unui punct

# Vecinătate *Eps*

- Vecinătatea *Eps* a unui punct  $p$  este mulțimea punctelor cuprinse într-o hipersferă de rază *Eps* în jurul lui  $p$

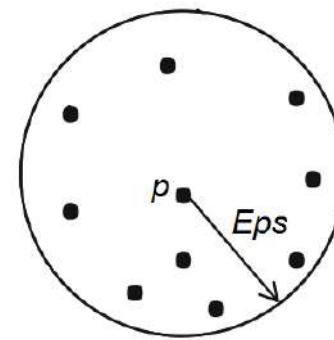
$$N_{Eps}(p) = \{q \in D | dist(p, q) \leq Eps\}$$



$D$  este mulțimea de antrenare

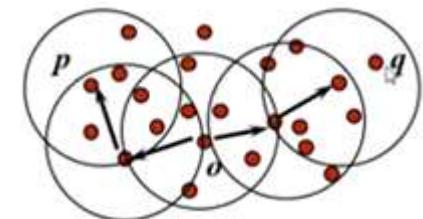
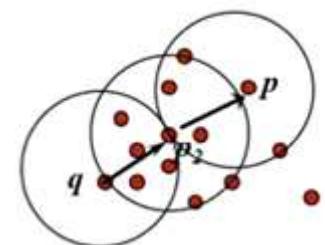
# Tipuri de puncte

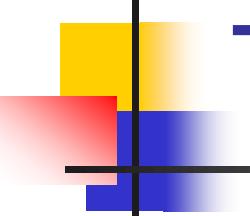
- Punct esențial (*core point*)
  - Un punct  $p$  este esențial dacă vecinătatea  $Eps$  a sa conține cel puțin  $MinPts$  puncte
$$|N_{Eps}(p)| \geq MinPts$$
- Aceste puncte sunt situate în regiunile dense ale spațiului



# Accesibilitate și conectare prin densitate

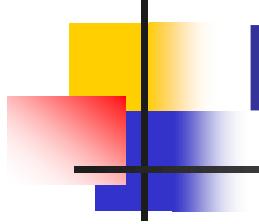
- Un punct  $p$  este **direct accesibil prin densitate** din punctul  $q$  dacă  $p \in N_{Eps}(q)$  și  $q$  este punct esențial
- Un punct  $p$  este **accesibil prin densitate** din punctul  $q$  dacă există un lanț de puncte  $p_1 = q, p_2, \dots, p_{n-1}, p_n = p$  astfel încât  $p_{i+1}$  este direct accesibil prin densitate din punctul  $p_i$
- Un punct  $p$  este **conectat prin densitate** cu punctul  $q$  dacă există un punct  $o$  astfel încât atât  $p$  cât și  $q$  sunt accesibile prin densitate din  $o$





# Tipuri de puncte

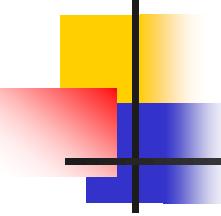
- Punct de graniță (*border point*)
  - Un punct este de graniță dacă nu este esențial, dar este accesibil prin densitate dintr-un punct esențial
  - Aceste puncte sunt situate în regiunile mai puțin dense
- Valoare extremă (*outlier, noise*)
  - Un punct care nu este esențial și nici de graniță
  - Aceste puncte nu sunt introduse în niciun grup!
- Conform algoritmului DBSCAN, un grup este mulțimea cea mai mare de instanțe (puncte) conectate prin densitate



# Pseudocod

**Dbscan( $d, \text{eps}, \text{minPts}$ ):**

```
for each unvisited point  $p$  in dataset  $d$ 
    mark  $p$  as visited
     $\text{neighborhood} = \text{Region}(p, \text{eps})$ 
    if  $\text{count}(\text{neighborhood}) < \text{minPts}$ 
        ignore  $p$ 
    else
         $c = \text{new cluster}$ 
         $\text{ExpandCluster}(p, \text{neighborhood}, c, \text{eps}, \text{minPts})$ 
```



# Pseudocod

**Region( $p, \text{eps}$ ):**

return all points within the  $n$ -dimensional sphere centered at  $p$  with radius  $\text{eps}$ ,  
including  $p$

**ExpandCluster( $p, \text{neighborhood}, c, \text{eps}, \text{minPts}$ ):**

add point  $p$  to cluster  $c$

for each point  $p'$  in  $\text{neighborhood}$

if  $p'$  is not visited

mark  $p'$  as visited

$\text{neighborhood}' = \text{Region}(p', \text{eps})$

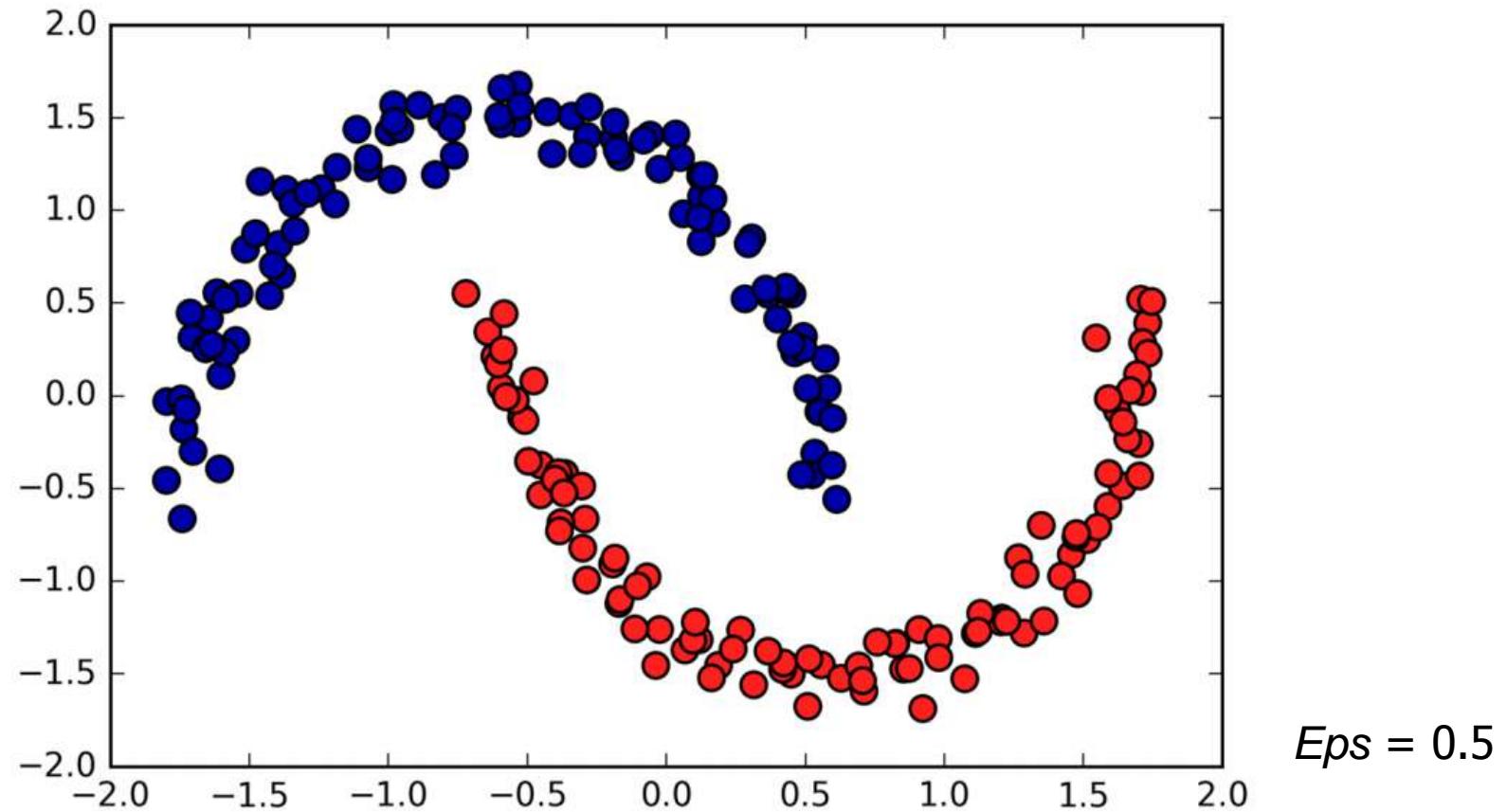
if  $\text{count}(\text{neighborhood}') \geq \text{minPts}$

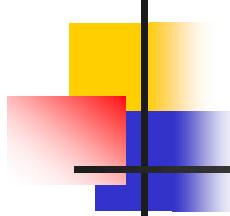
$\text{neighborhood} = \text{union}(\text{neighborhood}, \text{neighborhood}')$

if  $p'$  is not a member of any cluster

add  $p'$  to cluster  $c$

# Rezultate pentru forme complexe

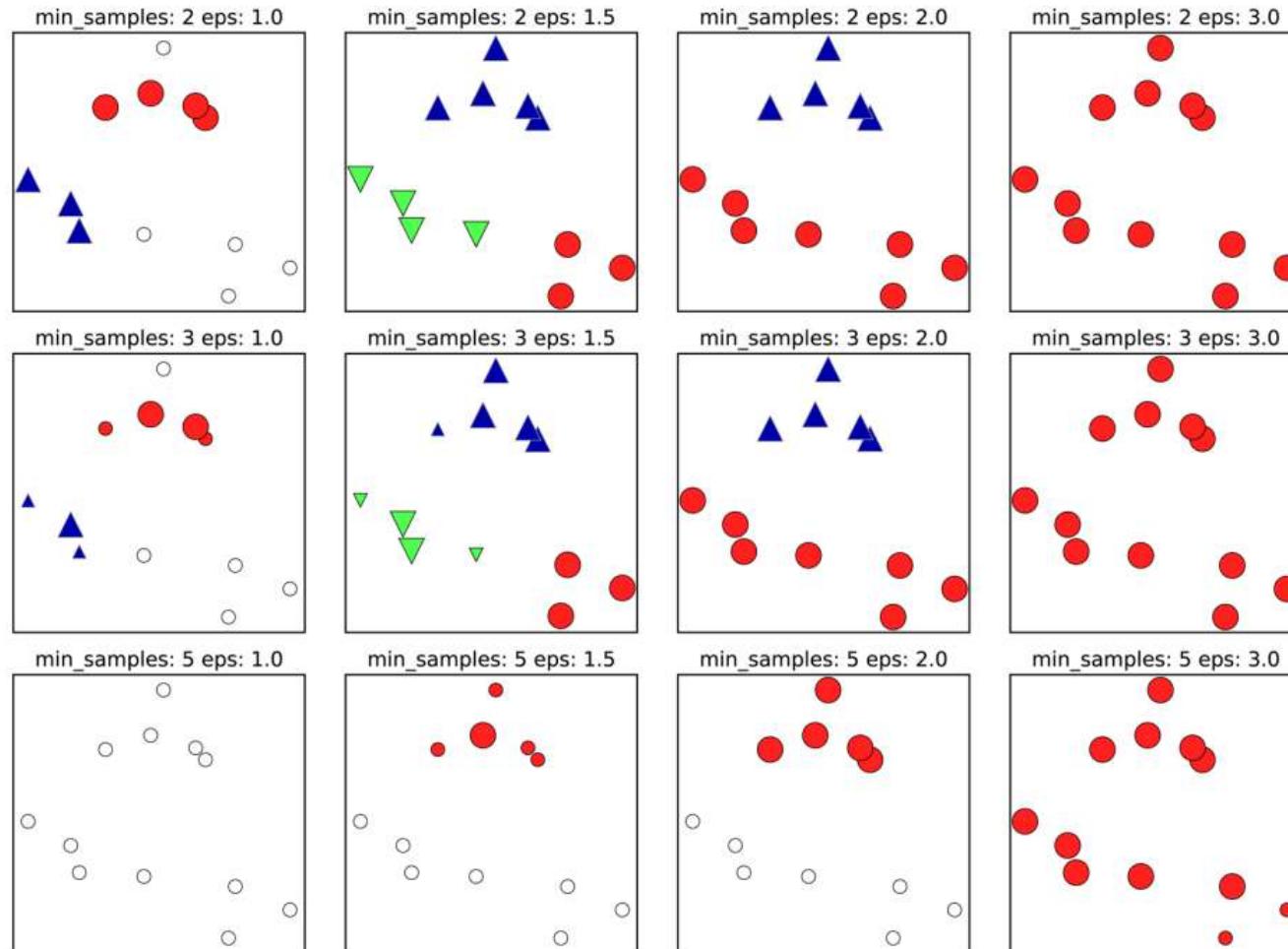


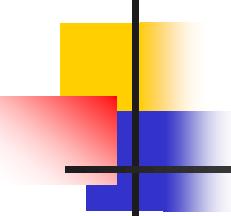


# Parametrii

- *Eps*
  - Dacă *Eps* este mai mare, atunci vor fi incluse mai multe puncte într-un grup
  - Controlează numărul de grupuri generate
- *MinPts*
  - Dacă *MinPts* este mai mare, atunci vor exista mai puține puncte esențiale și mai multe valori extreme
  - Controlează dacă punctele din regiuni mai puțin dense vor fi incluse în grupuri sau vor fi considerate valori extreme

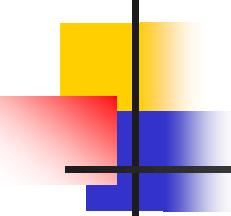
# Rezultate cu diferite valori pentru parametri





# Caracteristici

- Numărul de grupuri nu trebuie cunoscut *a priori*. Implicit, acesta este controlat de parametrii *Eps* și *MinPts*
- Poate trata grupuri de forme arbitrară
- Identifică valorile extreme
- Punctele esențiale situate la distanță mai mică decât *Eps* unele de altele sunt incluse în același grup
- La rulări repetitive, gruparea punctelor esențiale este întotdeauna aceeași, dar punctele de graniță pot apartine unor grupuri diferite
- Complexitatea în mod normal este  $O(n^2)$ . Dacă se folosește o metodă de indexare spațială potrivită, complexitatea se poate reduce la  $O(n \log n)$



# Concluzii

- **Gruparea (clusterizarea)** are ca scop găsirea unor grupuri astfel încât instanțele din același grup să fie mai asemănătoare între ele decât cu instanțele altor grupuri
- **Algoritmul *k-medii*** este probabil cel mai simplu algoritm de grupare și care se comportă bine când grupurile au formă aproximativ sferică iar numărul de grupuri este cunoscut
- **Algoritmul *EM*** este o extensie stochastică a algoritmului *k-medii*
- **Algoritmii ierarhici** încearcă să rezolve problema determinării numărului optim de grupuri prin unirea sau divizarea iterativă a instanțelor sau grupurilor
- **Algoritmul *DBSCAN*** se bazează pe densitatea punctelor și poate determina grupuri de forme arbitrarе



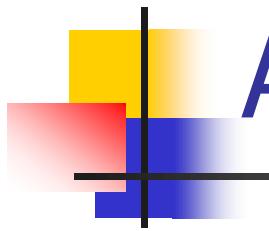
# Învățare automată

## 2. Algoritmi de clasificare

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

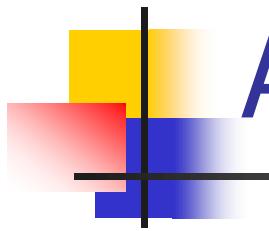
[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)



# Algoritmi de clasificare

1. Clasificarea
2. Arbori de decizie
3. Metoda Bayes naivă
4. Învățarea bazată pe instanțe





# Algoritmi de clasificare

1. Clasificarea
2. Arbori de decizie
3. Metoda Bayes naivă
4. Învățarea bazată pe instanțe



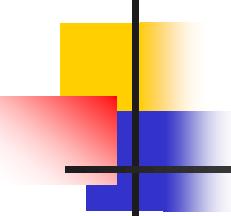
# Problema de clasificare

- Se dă o mulțime de **instante** (obiecte)
  - **Mulțimea de antrenare**
- Instantele au **attribute**
- Fiecare instantă are attribute cu anumite **valori**
- De obicei, ultimul atribut este **clasa**

**Atribute**

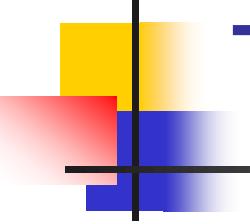
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

**Instante**



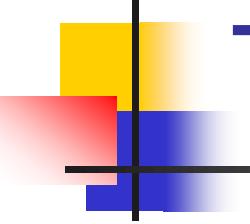
# Exemple de aplicații

- Recunoașterea imaginilor și vorbirii
- Clasificarea email-urilor în *spam* și *ham*
- Clasificarea știrilor în categorii precum politică, meteo, sport etc.
- Clasificarea plăștilor electronice ca legitime sau frauduloase
- Învățarea tratamentelor optime din înregistrările medicale
- Clasificarea celulelor din tumori ca benigne sau maligne pe baza radiografiilor
- Clasificarea structurilor secundare a proteinelor: predictia proprietășilor pe baza componentelor structurale



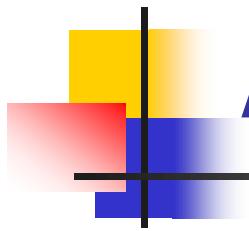
# Tipuri de atrbute

- Discrete (simbolice)
  - Nominale
    - Culoarea ochilor, nume, sex, CNP ca obiect, nu număr
  - Ordinale
    - Înălțime (mică, medie, mare), ranguri, calificative
- Continue (numerice)
  - De tip rational
    - Există un „element neutru”, de exemplu, 0
    - Lungime, distanță, prețuri
  - De tip interval
    - Temperatura în °C, date calendaristice



# Tipuri de atribute și algoritmi

- Unii algoritmi tratează în mod natural atributele **numerice** (de exemplu, rețelele neuronale, cei mai apropiati  $k$  vecini), alții tratează în mod natural atributele **simbolice** (de exemplu, clasificatorul bayesian naiv)
- Dacă avem atribute simbolice pentru algoritmi cu precădere numerici, se creează câte o intrare sau ieșire pentru fiecare valoare discretă (codarea *one-hot*)
- Dacă avem atribute numerice pentru algoritmi cu precădere simbolici, acestea se discretizează



# Algoritmi de clasificare

1. Clasificarea
2. Arbori de decizie
3. Metoda Bayes naivă
4. Învățarea bazată pe instanțe



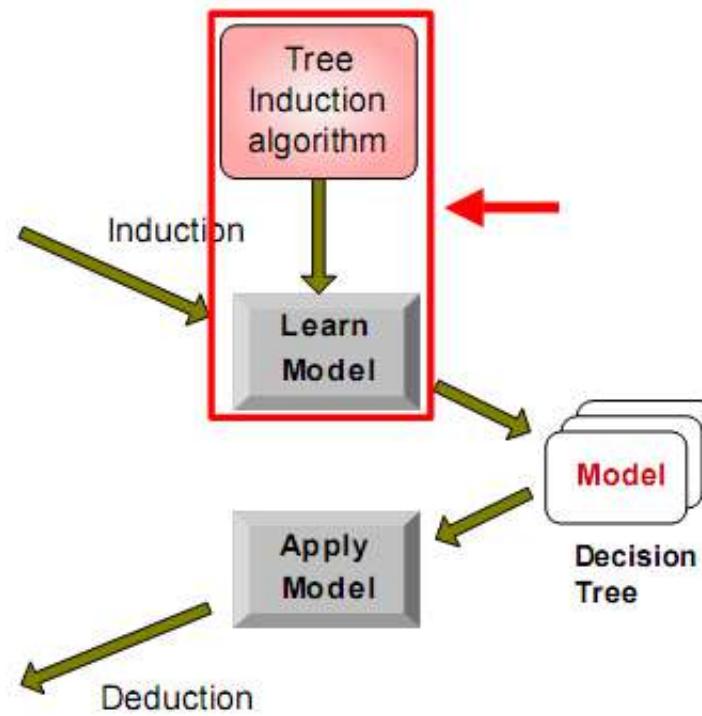
# Clasificarea cu arbori de decizie

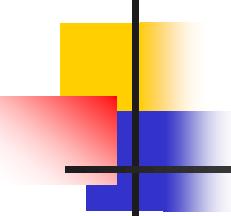
TId	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	80K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

TId	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	57K	?

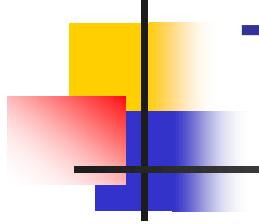
Test Set





# Inducția unui arbore de decizie

- Fie  $D_n$  mulțimea instanțelor de antrenare care ajung la un nod  $n$
- **Algoritmul lui Hunt** (procedura generală):
  - Dacă  $D_n$  conține numai instanțe din aceeași clasă  $y_n$ , atunci  $n$  este o frunză etichetată  $y_n$
  - Dacă  $D_n$  este mulțimea vidă, atunci  $n$  este o frunză etichetată cu clasa implicită (*default*)  $y_d$
  - Dacă  $D_n$  conține instanțe care aparțin mai multor clase, se utilizează un **test de atribut** pentru a parta datele în mulțimi mai mici
  - Se aplică recursiv procedura pentru fiecare submulțime

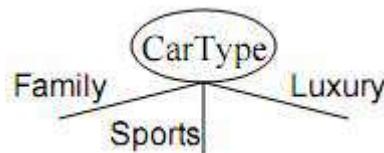


# Testul de atrbute

- Urmează o strategie *greedy*: se partitioanează multimea de instanțe cu un test care maximizează un anumit criteriu
- Depinde de tipul atributului: nominal, ordinal sau continuu
- Depinde de numărul de posibilități de partitioare: binar sau multiplu

# Atribute nominale

- Partiționarea multiplă
  - Numărul de partiții = numărul de valori distincte

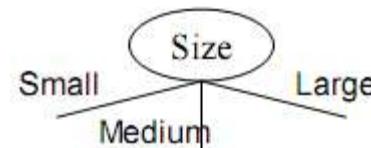


- Partiționarea binară
  - Se împart valorile în două submultimi
  - Trebuie descoperită partiționarea optimă

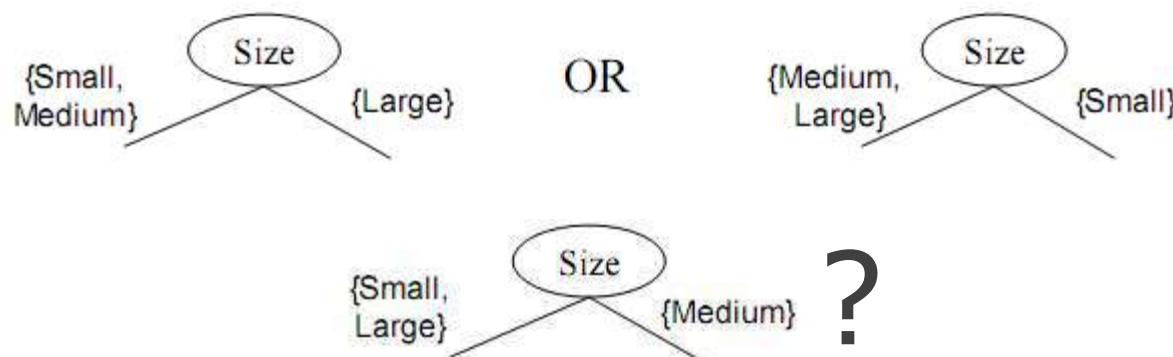


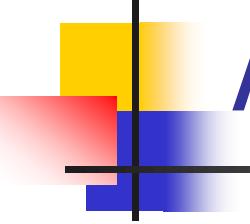
# Atribute ordinale

- Partiționarea multiplă
  - Numărul de partiții = numărul de valori distincte



- Partiționarea binară
  - Se divid valorile în două submulțimi
  - Trebuie descoperită partaționarea optimă





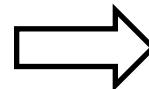
# Atribute continue

- Se discretizează datele pentru a le transforma în atribute ordinale
  - Cu interval egal: histograma
  - Cu frecvență egală: multimi cu numere egale de instanțe
  - Grupare (*clustering*)
- Decizie binară:  $(A_i \leq v)$  sau  $(A_i > v)$ 
  - Trebuie considerate toate partitioнările posibile
  - Necesară un efort de calcul mai mare

# Discretizarea

- Cu interval egal – de exemplu, 3 intervale
  - [65, 75], (75, 85], (85, 95]

Umiditate	Joc
65	Da
70	Da
72	Da
75	Da
80	Da
85	Da
86	Nu
90	Nu
90	Nu
91	Nu
93	Nu
95	Nu



Umiditate-Dis1	Joc
Mică	Da
Medie	Da
Medie	Da
Mare	Nu

# Discretizarea

- Cu frecvență egală – de exemplu, 3 intervale

Umiditate	Joc
65	Da
70	Da
72	Da
75	Da
80	Da
85	Da
86	Nu
90	Nu
90	Nu
91	Nu
93	Nu
95	Nu

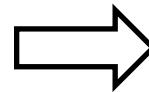


Umiditate-Dis2	Joc
Mică	Da
Medie	Da
Medie	Da
Medie	Nu
Medie	Nu
Mare	Nu

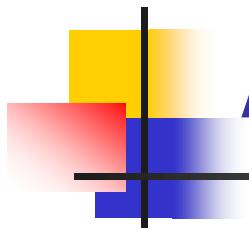
# Discretizarea

- **Binară**, cu o valoare de referință – de exemplu, 85

Umiditate	Joc
65	Da
70	Da
72	Da
75	Da
80	Da
85	Da
86	Nu
90	Nu
90	Nu
91	Nu
93	Nu
95	Nu

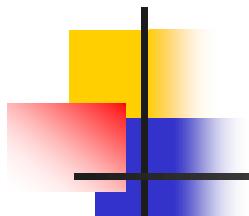


Umiditate	Umiditate-Bin	Joc
65	Da	Da
70	Da	Da
72	Da	Da
75	Da	Da
80	Da	Da
85	Da	Da
86	Nu	Nu
90	Nu	Nu
90	Nu	Nu
91	Nu	Nu
93	Nu	Nu
95	Nu	Nu



# Algoritmul descris

- În continuare, vom prezenta un algoritm sintetizat pe baza ideilor algoritmilor ID3 și C4.5 și folosind măsura de impuritate din algoritmul CART
  - ID3 și C4.5 folosesc entropia
  - CART folosește indexul Gini



# Partiționarea optimă

- Euristică: se preferă nodurile cu **cea mai omogenă** distribuție de clase
- Necesită o măsură a „impurității” nodurilor

C0: 5
C1: 5

Ne-omogene  
Grad mare de impuritate

C0: 9
C1: 1

Omogene  
Grad mic de impuritate

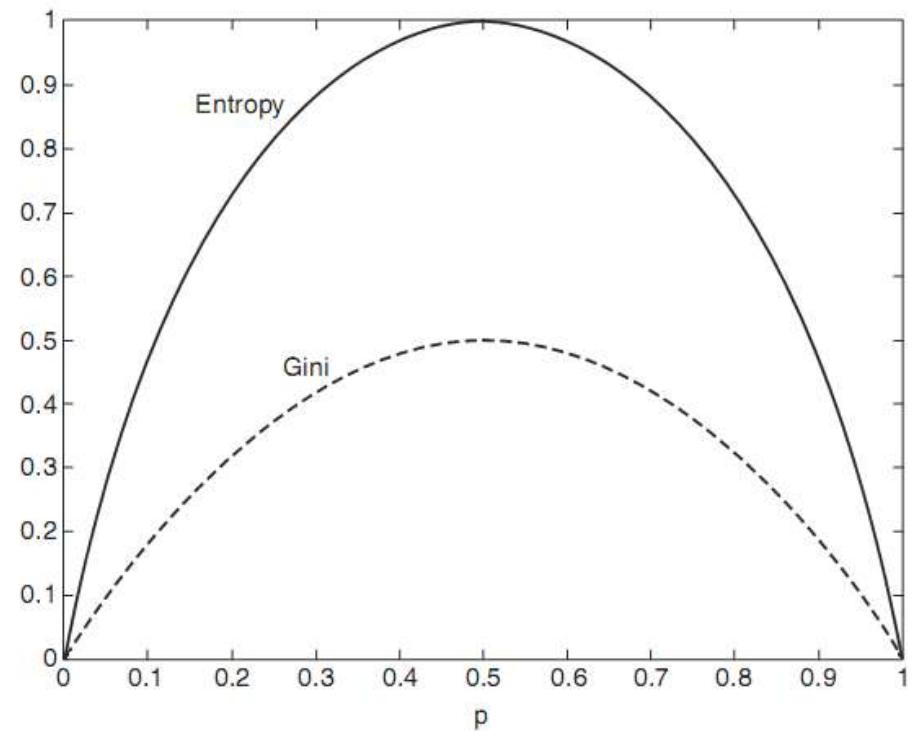
# Măsuri de impuritate

$$\text{Entropy}(t) = - \sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$$

Convenție:  $0 \cdot \log_2 0 = 0$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2$$

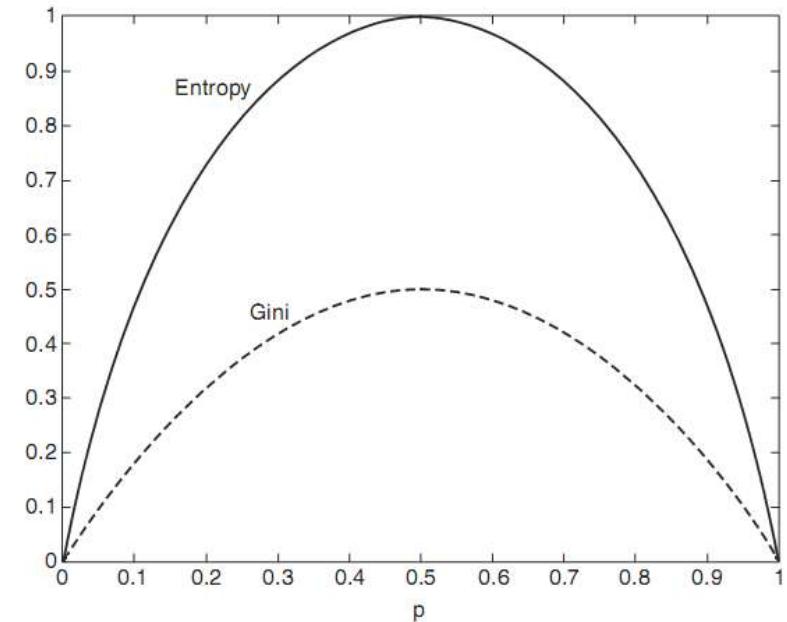
Pentru o problemă cu două clase:



# Măsuri de impuritate



- Valoarea maximă:  
instanțele sunt  
distribuite egal între  
clase
- Valoarea minimă (0):  
toate instanțele  
apartin unei singure  
clase



# Exemple

$$\text{Entropy}(t) = - \sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2$$

Node $N_1$	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$

$$\text{Entropy} = -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$$

Node $N_2$	Count
Class=0	1
Class=1	5

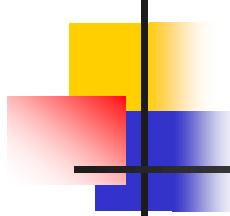
$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

$$\text{Entropy} = -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$$

Node $N_3$	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$

$$\text{Entropy} = -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$$

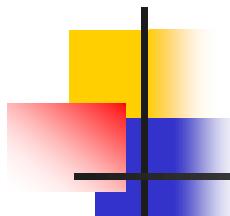


# Partitionarea

- Când un nod părinte  $p$  este partitionat în  $k$  fii, calitatea partitionării (de exemplu, indexul Gini  $G$ ) se calculează astfel:

$$G_{split} = \sum_{i=1}^k \frac{n_i}{n} G_i$$

- unde  $n_i$  este numărul de instanțe din nodul fiu  $i$ , iar  $n$  este numărul de instanțe din nodul  $p$
- Formulă similară pentru entropie



# Câştigul informațional

- Calitatea unei partitioñări este determinată de creșterea omogenităñii submultimilor rezultate
- Trebuie maximizat câştigul informañional:  
$$\Delta = I(\text{părinte}) - \sum_i (n_i / n \cdot I(\text{fiu}_i))$$
- Deoarece  $I(\text{părinte})$  este acelañi pentru toñi fiili, se preferă valoarea minimă pentru  $\sum_i (n_i / n \cdot I(\text{fiu}_i))$
- Termenul de „câştig informañional” se utilizează când se foloseñte entropia ca măsură de impuritate, dar principiul este acelañi pentru indexul Gini sau orice altă măsură de impuritate

# Exemplu: construirea unui AD

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Se calculează câștigul informational pentru fiecare atribut (Refund, Status, Income)
- Refund
  - Refund = Yes → 3 instanțe
    - Cheat = Yes → 0
    - Cheat = No → 3
      - Gini = 0
  - Refund = No → 7 instanțe
    - Cheat = Yes → 3
    - Cheat = No → 4
      - Gini =  $1 - (3/7)^2 - (4/7)^2 = 0.49$
  - $Gini_{Refund} = (3/10) \cdot 0 + (7/10) \cdot 0.49 = 0.343$

# Exemplu: construirea unui AD

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

## ■ Marital Status

- Status = Divorced → 2 instanțe
  - Cheat = Yes → 1
  - Cheat = No → 1
    - Gini =  $1 - (1/2)^2 - (1/2)^2 = 0.5$
- Status = Married → 4 instanțe
  - Cheat = Yes → 0
  - Cheat = No → 4
    - Gini =  $1 - (0/4)^2 - (4/4)^2 = 0$
- Status = Single → 4 instanțe
  - Cheat = Yes → 2
  - Cheat = No → 2
    - Gini =  $1 - (2/4)^2 - (2/4)^2 = 0.5$
- $Gini_{Status} = (2/10) \cdot 0.5 + (4/10) \cdot 0 + (4/10) \cdot 0.5 = 0.3$

# Construirea unui AD: atribute continue

- Pentru eficientizarea calculelor, pentru fiecare atribut:
  - Se sortează valorile
  - Se parcurează liniar valorile, actualizându-se numărarea instanțelor și calculându-se indexul Gini
  - Se alege poziția de partiționare cu indexul Gini minim

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No	
Taxable Income											
Valori sortate	60	70	75	85	90	95	100	120	125	220	
Pozitii de part.	59.9	65	72.5	80	87.5	92.5	97.5	110	122.5	172.5	220.1
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	
Gini	0.420	0.400	0.375	0.343	0.417	0.400	0.300	0.343	0.375	0.400	

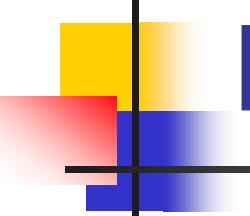
# Construirea unui AD: attribute continue

- Optimizare: se calculează indexul Gini doar pentru pozițiile unde se schimbă valoarea clasei
  - 2 partaționări candidat în loc de 11



Valori sortate →  
Poziții de part. →

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No	
Taxable Income											
	60	70	75	85	90	95	100	120	125	220	
	59.9	65	72.5	80	87.5	92.5	97.5	110	122.5	172.5	220.1
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	0.300	0.343	0.375	0.400	0.420

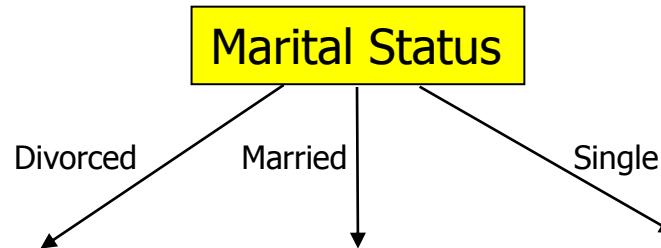


# Prima partitōnare

- $Gini_{Refund} = 0.343$
- $Gini_{Status} = 0.3$
- $Gini_{Income} = 0.3$
  
- Partiționările după *Status* și *Income* sunt egal posibile, dar rezultatele pot fi foarte diferite!

# Procedura recursivă

- Să considerăm *Status* pentru prima partitōnare



Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

# AI doilea nivel

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Atribute rămase
  - Refund, Income
- Status = Divorced
  - Refund = No → Cheat = Yes
  - Refund = Yes → Cheat = No
  - Gini = 0, partitioanare după *Refund*

# AI doilea nivel

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Status = Married  $\Rightarrow$  Cheat = No
- Nu mai sunt necesare alte partitioňări

# AI doilea nivel

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Status = Single
  - Refund = Yes → 1 instanță
    - Cheat = Yes → 0
    - Cheat = No → 1
      - Gini = 0
  - Refund = No → 3 instanțe
    - Cheat = Yes → 2
    - Cheat = No → 1
      - Gini =  $1 - (2/3)^2 - (1/3)^2 = 0.444$
  - $\text{Gini}_{\text{Refund}} = 0 + (3/4) \cdot 0.444 = 0.333$

# AI doilea nivel

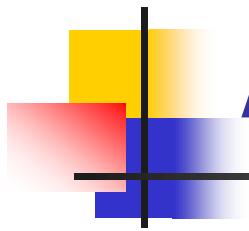
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

## ■ Status = Single

Cheat	No		Yes		Yes		No	
	Taxable Income							
	70		85		90		125	
	69.1		77.5		87.5		107.5	125.1
	<=	>	<=	>	<=	>	<=	>
Yes	0	2	0	2			2	0
No	0	2	1	1			1	1
Gini	0.5		0.333				0.333	0.5



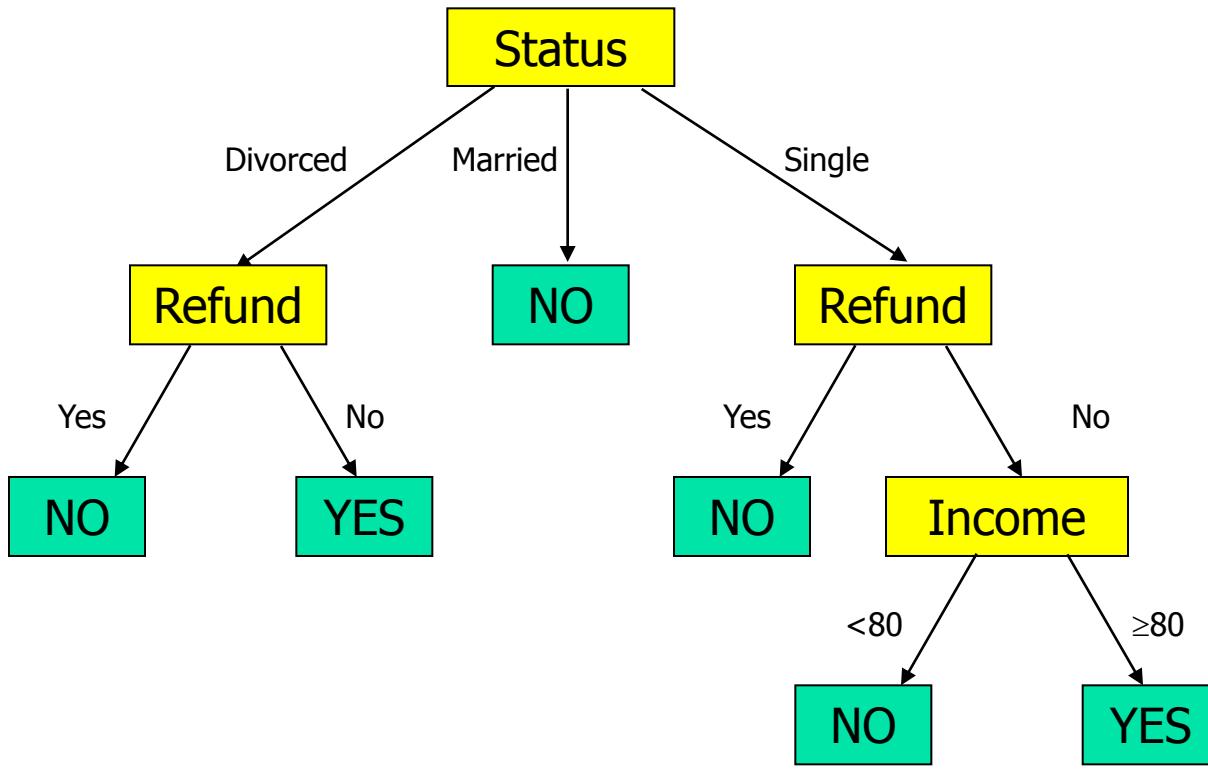
Valoarea clasei neschimbată



# AI doilea nivel

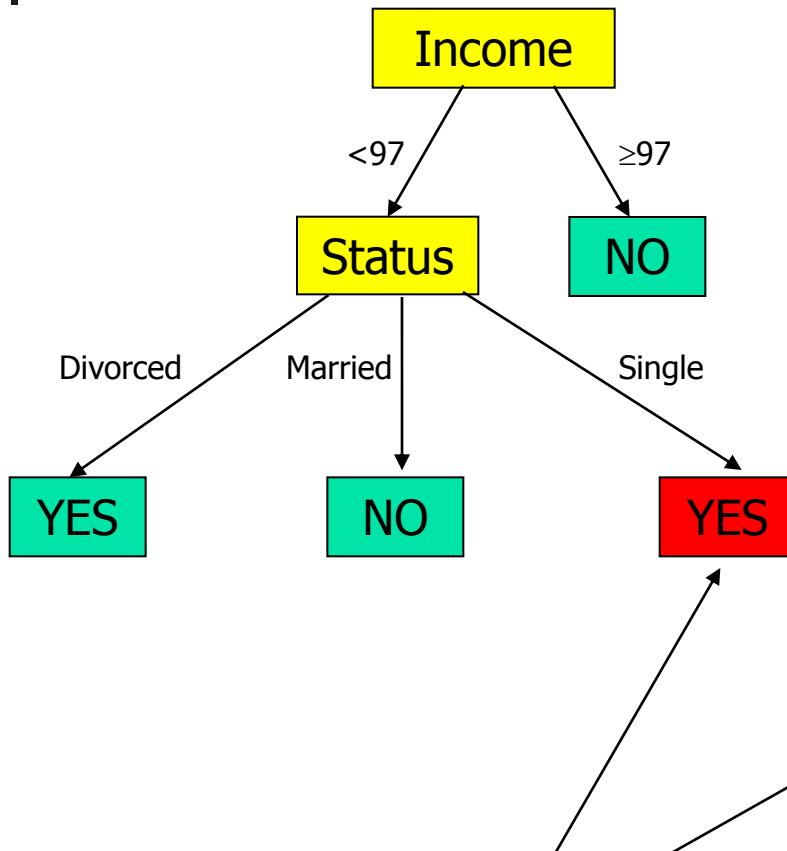
- $\text{Gini}_{\text{Refund}} = 0.333$
- $\text{Gini}_{\text{Income}} = 0.333$
- Partitionările după *Refund* și *Income* sunt egale posibile
- Să considerăm *Refund*

# Arborele final



Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

# Decizia alternativă



Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Arborele de decizie are o **eroare** chiar pentru **multimea de antrenare!**

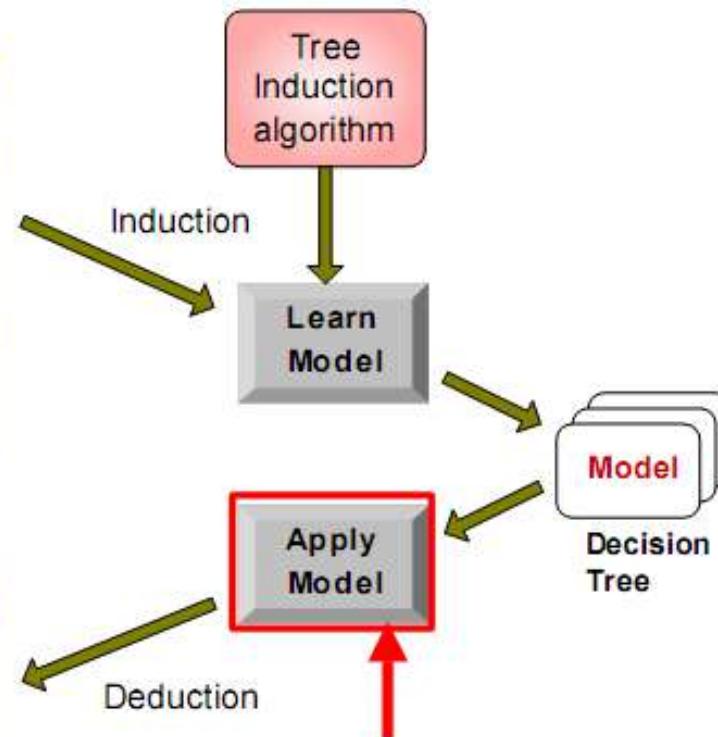
# Clasificarea cu arbori de decizie

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	80K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

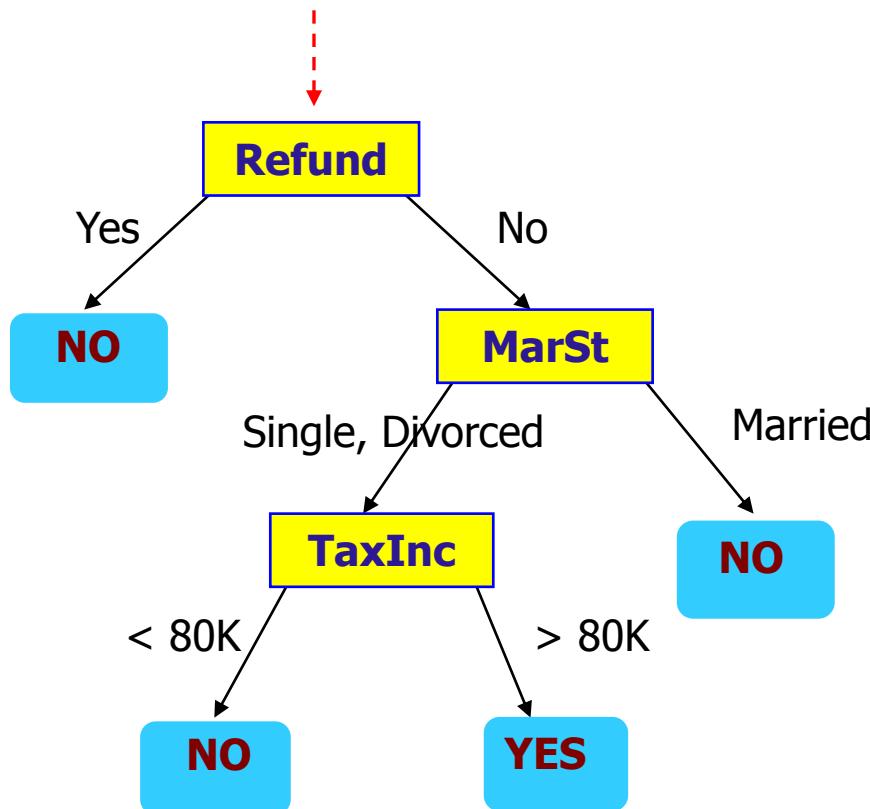
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	57K	?

Test Set



# Aplicarea modelului

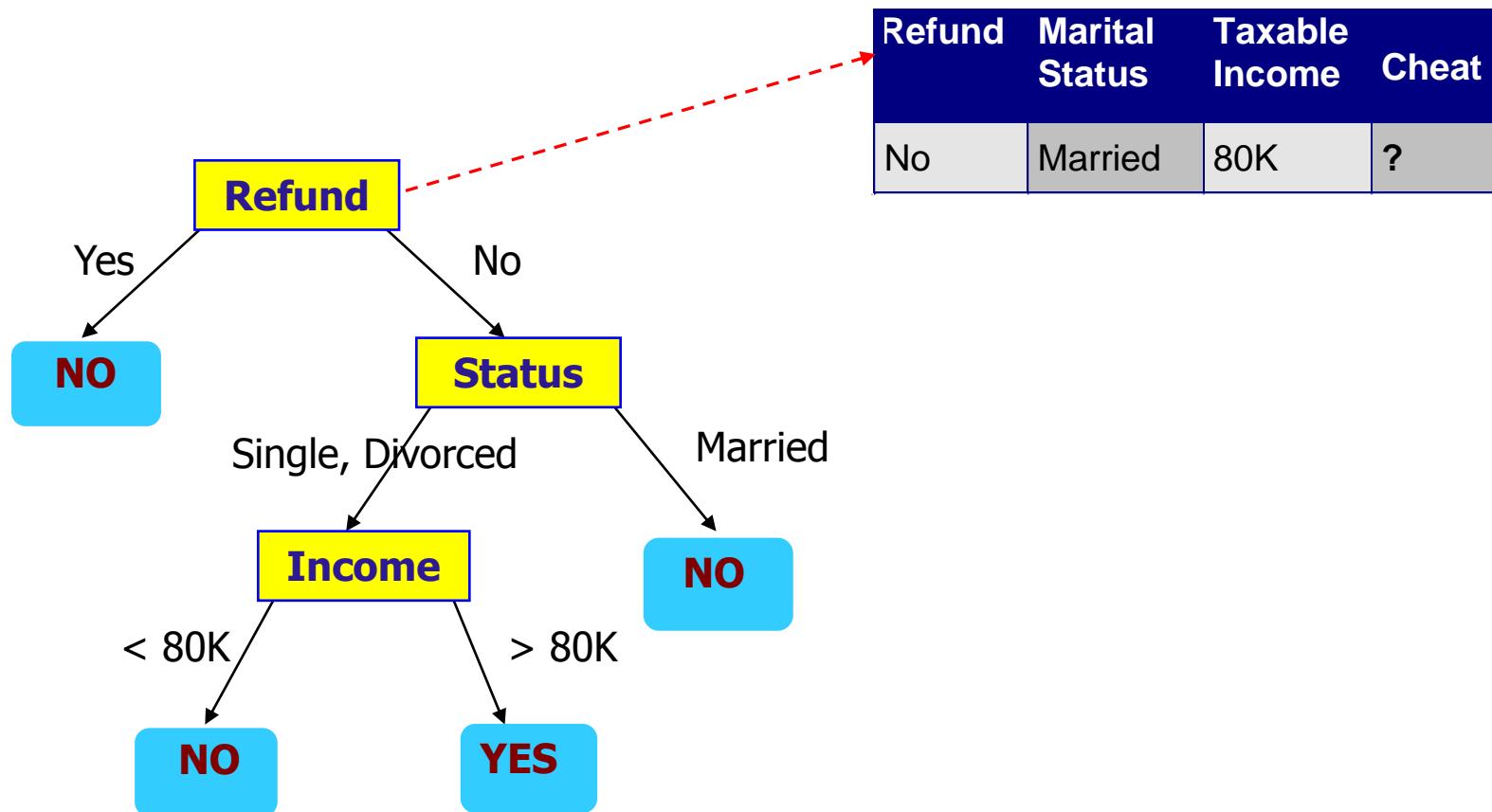
Se începe din rădăcină



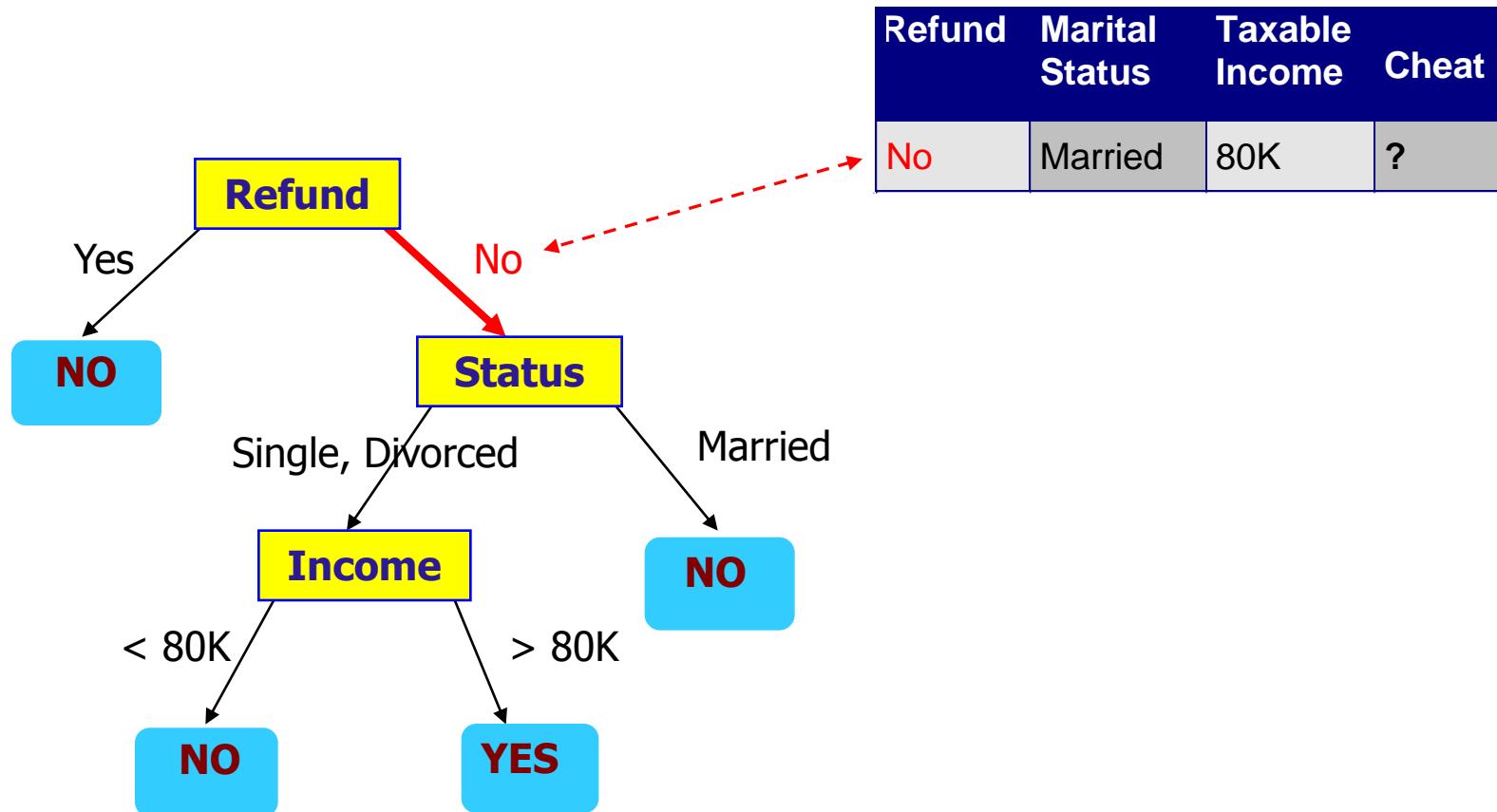
## Instanța de test

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

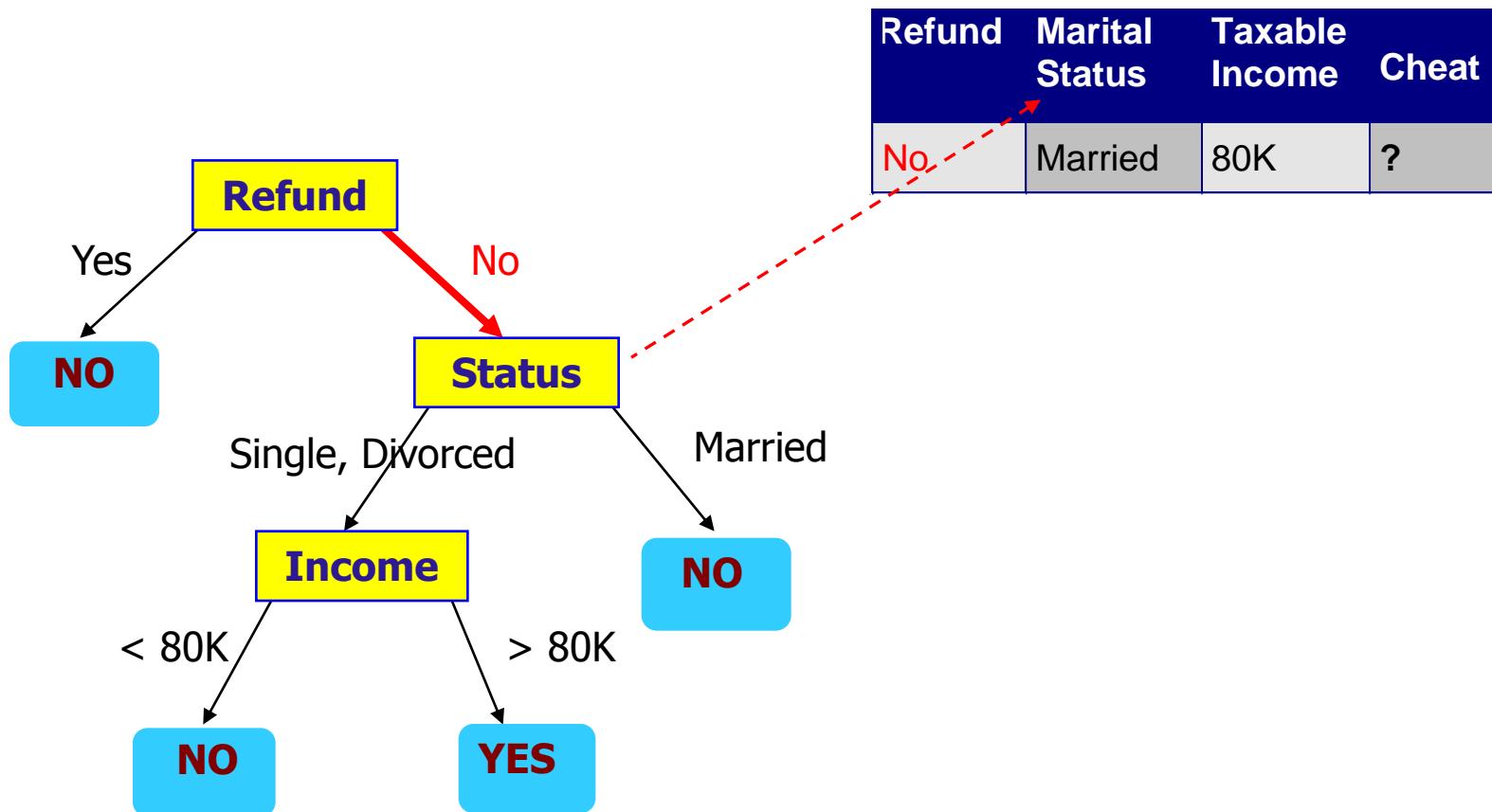
# Aplicarea modelului



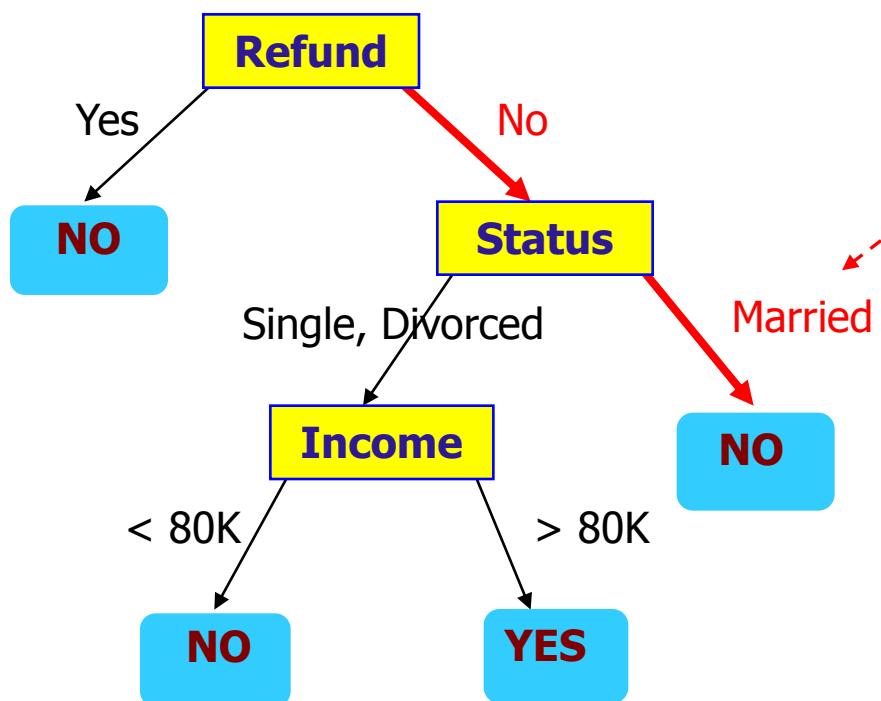
# Aplicarea modelului



# Aplicarea modelului

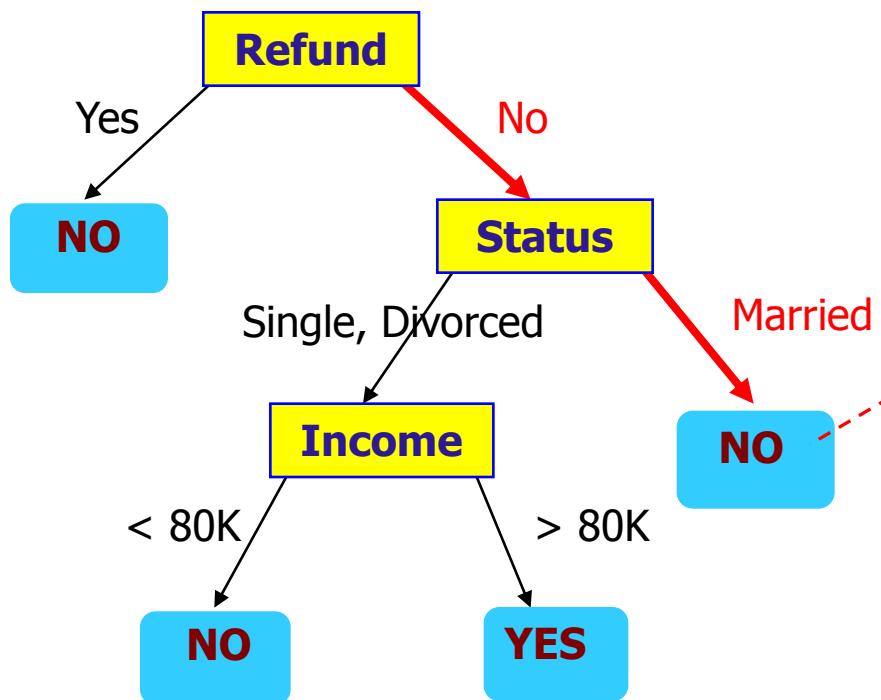


# Aplicarea modelului



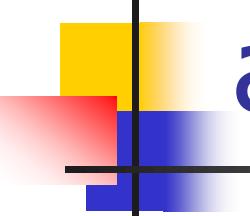
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

# Aplicarea modelului



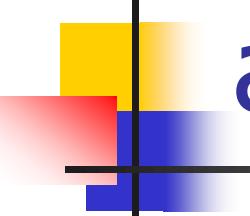
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Se clasifică instanța:  
Cheat = No



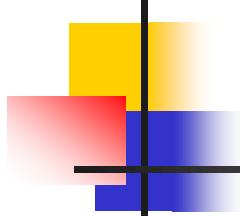
# Algoritmi pentru arbori de decizie

- **ID3** (*Iterative Dichotomiser 3*)
  - Operează numai pe atrbute discrete
- **C4.5**
  - Extensie a algoritmului *ID3*
  - Permite și atrbute continue
  - Permite partiționarea de mai multe ori după același atrbut
  - Permite retezarea (*pruning*) arborelui generat pentru a crește capacitatea de generalizare
- Algoritmii *ID3* și *C4.5* folosesc entropia ca măsură de impuritate, ceea ce conduce la construirea unor arbori de dimensiuni cât mai reduse



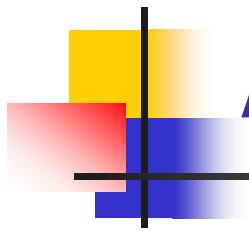
# Algoritmi pentru arbori de decizie

- **Arbori aleatorii (*Random Tree, Random Forest*)**
  - Se stabilesc atributele pentru partitioare în mod aleatoriu
  - Se generează arbori de dimensiuni mai mari
  - Eroarea pe mulțimea de antrenare este foarte mică, dar capacitatea de generalizare poate fi mai redusă
- **Toate aceste metode sunt euristice**
  - Nu se garantează dimensiunea minimă a arborelui și nici performanțele
  - Există mai mulți arbori posibili pentru o problemă dată



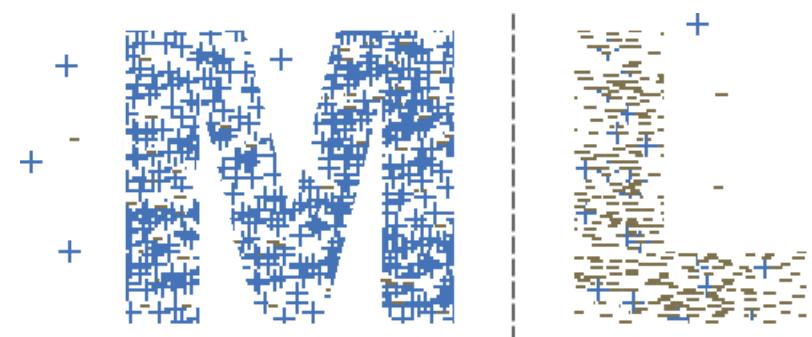
# Clasificarea cu arbori de decizie

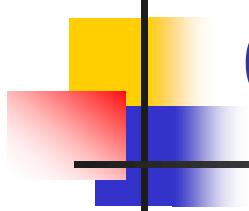
- Dificultate medie de implementare
  - Necesită o serie de calcule
  - Presupune partiționarea recursivă
- Arborii sunt rapizi la clasificarea instanțelor necunoscute
- Ușor de interpretat, mai ales pentru arbori de dimensiuni mici
- Un arbore de decizie poate fi interpretat ca o mulțime de reguli, de exemplu:
  - „Dacă Marital Status este Divorced și Refund este Yes atunci Cheat este No”



# Algoritmi de clasificare

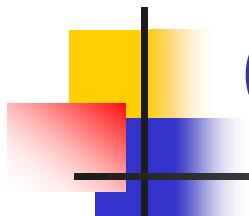
1. Clasificarea
2. Arbori de decizie
3. Metoda Bayes naivă
4. Învățarea bazată pe instanțe





# Clasificarea bayesiană

- Se consideră fiecare atribut și clasa ca fiind variabile aleatorii urmând anumite distribuții
- Se dă o instanță cu attributele  $(A_1, A_2, \dots, A_n)$
- Trebuie determinată valoarea clasei  $C$  care maximizează  $P(C | A_1, A_2, \dots, A_n)$
- $P(C | A_1, A_2, \dots, A_n)$  trebuie estimată direct din date

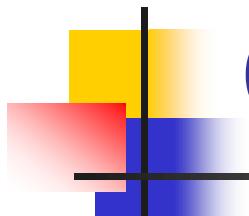


# Clasificarea bayesiană

- Conform teoremei lui Bayes:

$$P(C|A_1A_2...A_n) = \frac{P(A_1A_2...A_n|C) \cdot P(C)}{P(A_1A_2...A_n)}$$

- Se alege valoarea clasei  $C$  care maximizează  $P(C | A_1, A_2, \dots, A_n)$
- Pentru o instanță,  $P(A_1, A_2, \dots, A_n)$  este aceeași, deci se alege valoarea clasei  $C$  care maximizează  $P(A_1, A_2, \dots, A_n | C) \cdot P(C)$
- $P(C)$  este ușor de calculat
- Mai trebuie estimată  $P(A_1, A_2, \dots, A_n | C)$



# Clasificatorul bayesian naiv

- engl. “Naïve Bayes”
- Această metodă de clasificare presupune că atributele  $A_i$  sunt independente dată fiind clasa  $C_j$
- $P(A_1, A_2, \dots, A_n | C_j) = P(A_1 | C_j) P(A_2 | C_j) \dots P(A_n | C_j)$
- Putem estima  $P(A_i | C_j)$  pentru toate  $A_i$  și  $C_j$
- Trebuie găsită valoarea  $C_j$  astfel încât produsul  $P(C_j) \prod_i P(A_i | C_j)$  să fie maxim

# Exemplu de clasificare

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- $x_q = (\text{No}, \text{Married}, 80\text{K})$
- Calculăm separat probabilitățile pentru  
 $C_{\text{Yes}} = (\text{Cheat} = \text{Yes})$  și  
 $C_{\text{No}} = (\text{Cheat} = \text{No})$

# Exemplu de clasificare

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- $x_q = (\text{No}, \text{Married}, 80\text{K})$
- $\operatorname{argmax}_C P(C) \prod_i P(A_i | C)$
- $C_{\text{Yes}} = (\text{Cheat} = \text{Yes})$
- $P(\text{Cheat} = \text{Yes}) = 3 / 10$
- $P(\text{Refund} = \text{No} | \text{Cheat} = \text{Yes}) = 3 / 3$
- $P(\text{Status} = \text{Married} | \text{Cheat} = \text{Yes}) = 0 / 3$

# Considerente practice

- Dacă una din probabilitățile condiționate este 0, atunci tot produsul devine 0

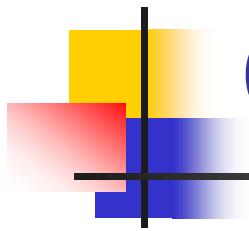
Starea vremii	Umiditate	Joc
Înnorat	Mare	Da
Ploaie	Mare	Da
Înnorat	Mare	Da
Soare	Mare	Nu
Soare	Mare	Nu
Ploaie	Normală	Nu

$$\begin{aligned}P(J_D) &= \frac{3}{6} & P(J_N) &= \frac{3}{6} \\P(S_I|J_D) &= \frac{2}{3} & P(S_I|J_N) &= \frac{0}{3} \\P(U_N|J_D) &= \frac{0}{3} & P(U_N|J_N) &= \frac{1}{3}.\end{aligned}$$

$x_q = (\hat{\text{Innorat}}, \text{Normală}).$

$$P(J_D) \cdot P(x_q|J_D) = \frac{3}{6} \cdot \frac{2}{3} \cdot \frac{0}{3} = 0,$$

$$P(J_N) \cdot P(x_q|J_N) = \frac{3}{6} \cdot \frac{0}{3} \cdot \frac{1}{3} = 0.$$



# Considerente practice

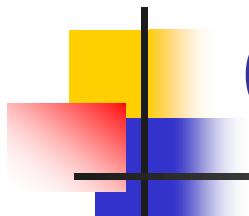
- Soluție: corecția Laplace
- Se estimează probabilitățile *a priori* ale fiecărei valori
- Inițial:

$$P(A_i|C_j) = \frac{n_{ij}}{n_j},$$

- Fie  $c$  numărul de clase
- Corecția Laplace:

$$P(A_i|C_j) = \frac{n_{ij} + 1}{n_j + c}.$$

- Foarte utilă, de exemplu, la clasificarea textelor

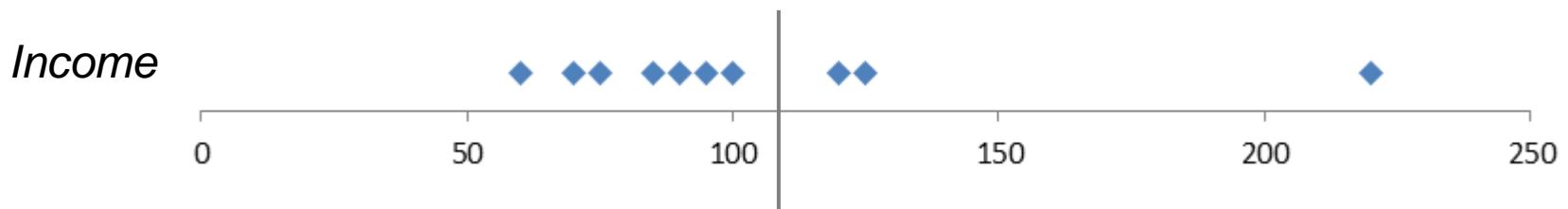


# Considerente practice

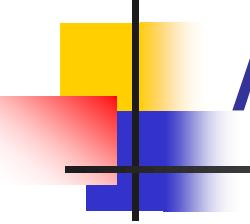
- Precizia calculelor poate fi afectată de înmulțirea probabilităților (valori mici)
  - Soluție: **folosirea logaritmilor**
  - Produsul de probabilități este înlocuit cu suma logaritmilor de probabilități

# Atribute continue

- Abordarea 1. Discretizarea în atribute ordinale
- Abordarea 2. Partiționarea binară: se aplică unul din teste (A<sub>i</sub> ≤ v) sau (A<sub>i</sub> > v) și rezultă valori de tip „Da” sau „Nu”



- Abordarea 3. Discretizarea cu ajutorul unui algoritm de grupare (clusterizare)



# Atribute continue

- Abordarea 4, folosită mai rar. Estimarea distribuției de probabilitate
  - Se presupune că valorile atributului respectă o distribuție (de exemplu, cea normală)
  - Parametrii distribuției (de exemplu, media și deviația standard) se estimează din date, pentru fiecare valoare a clasei
  - Odată cunoscută distribuția de probabilitate, aceasta poate fi utilizată pentru a calcula probabilitățile condiționate  $P(A_i | C_j)$  cu ajutorul funcției de densitate de probabilitate a distribuției utilizate

# Exemplu de clasificare

Tid	Refund	Marital Status	Taxable Income ( $v \leq 100$ )	Cheat
1	Yes	Single	125K $\Rightarrow$ Big	No
2	No	Married	100K $\Rightarrow$ Small	No
3	No	Single	70K $\Rightarrow$ Small	No
4	Yes	Married	120K $\Rightarrow$ Big	No
5	No	Divorced	95K $\Rightarrow$ Small	Yes
6	No	Married	60K $\Rightarrow$ Small	No
7	Yes	Divorced	220K $\Rightarrow$ Big	No
8	No	Single	85K $\Rightarrow$ Small	Yes
9	No	Married	75K $\Rightarrow$ Small	No
10	No	Single	90K $\Rightarrow$ Small	Yes

- $x_q = (\text{No, Married, } 80\text{K} \equiv \text{Small})$
- $\operatorname{argmax}_C P(C) \prod_i P(A_i | C)$
- $C_{\text{Yes}} = (\text{Cheat} = \text{Yes})$
- $P(\text{Cheat} = \text{Yes}) = 3 / 10$
- $P(\text{Refund} = \text{No} | \text{Cheat} = \text{Yes}) = 3 / 3$
- $P(\text{Status} = \text{Married} | \text{Cheat} = \text{Yes}) = 0 / 3$
- $P(\text{Income} = \text{Small} | \text{Cheat} = \text{Yes}) = 3 / 3$

$$P(x_q | C_{\text{Yes}}) = \frac{3}{10} \cdot \frac{3+1}{3+2} \cdot \frac{0+1}{3+2} \cdot \frac{3+1}{3+2} = 0.0384$$

# Exemplu de clasificare

Tid	Refund	Marital Status	Taxable Income ( $v \leq 100$ )	Cheat
1	Yes	Single	125K $\Rightarrow$ Big	No
2	No	Married	100K $\Rightarrow$ Small	No
3	No	Single	70K $\Rightarrow$ Small	No
4	Yes	Married	120K $\Rightarrow$ Big	No
5	No	Divorced	95K $\Rightarrow$ Small	Yes
6	No	Married	60K $\Rightarrow$ Small	No
7	Yes	Divorced	220K $\Rightarrow$ Big	No
8	No	Single	85K $\Rightarrow$ Small	Yes
9	No	Married	75K $\Rightarrow$ Small	No
10	No	Single	90K $\Rightarrow$ Small	Yes

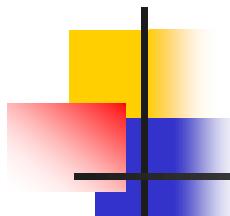
- $x_q = (\text{No}, \text{Married}, 80K \equiv \text{Small})$
- $\operatorname{argmax}_C P(C) \prod_i P(A_i | C)$
- $C_{\text{No}} = (\text{Cheat} = \text{No})$
- $P(\text{Cheat} = \text{No}) = 7 / 10$
- $P(\text{Refund} = \text{No} | \text{Cheat} = \text{No}) = 4 / 7$
- $P(\text{Status} = \text{Married} | \text{Cheat} = \text{No}) = 4 / 7$
- $P(\text{Income} = \text{Small} | \text{Cheat} = \text{No}) = 4 / 7$

$$P(x_q | C_{\text{No}}) = \frac{7}{10} \cdot \frac{4+1}{7+2} \cdot \frac{4+1}{7+2} \cdot \frac{4+1}{7+2} = 0.12$$

# Exemplu de clasificare

Tid	Refund	Marital Status	Taxable Income ( $v \leq 100$ )	Cheat
1	Yes	Single	125K $\Rightarrow$ Big	No
2	No	Married	100K $\Rightarrow$ Small	No
3	No	Single	70K $\Rightarrow$ Small	No
4	Yes	Married	120K $\Rightarrow$ Big	No
5	No	Divorced	95K $\Rightarrow$ Small	Yes
6	No	Married	60K $\Rightarrow$ Small	No
7	Yes	Divorced	220K $\Rightarrow$ Big	No
8	No	Single	85K $\Rightarrow$ Small	Yes
9	No	Married	75K $\Rightarrow$ Small	No
10	No	Single	90K $\Rightarrow$ Small	Yes

- $x_q = (\text{No}, \text{Married}, 80K \equiv \text{Small})$
- $P(x_q | C_{\text{Yes}}) = 0.0384$
- $P(x_q | C_{\text{No}}) = 0.12$
- $\Rightarrow \text{argmax}_C P(x_q | C) = C_{\text{No}}$



# Clasificarea Bayes naivă

- Avantaje
  - Calcule simple
  - Robustete la zgomot și attribute irelevante
- Aplicabilitate
  - Multimi de antrenare medii sau mari
  - Attribute independente condițional. Presupunerea este deseori infirmată în realitate, dar metoda poate funcționa surprinzător de bine. Extensie: rețele bayesiene
- Aplicații de succes
  - Diagnoză
  - Clasificarea documentelor text
  - Detectia spam-ului

# Algoritmi de clasificare

1. Clasificarea
2. Arbori de decizie
3. Metoda Bayes naivă
4. Învățarea bazată pe instanțe



# Clasificarea bazată pe instanțe

Instanțele memorate

Atr1	.....	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

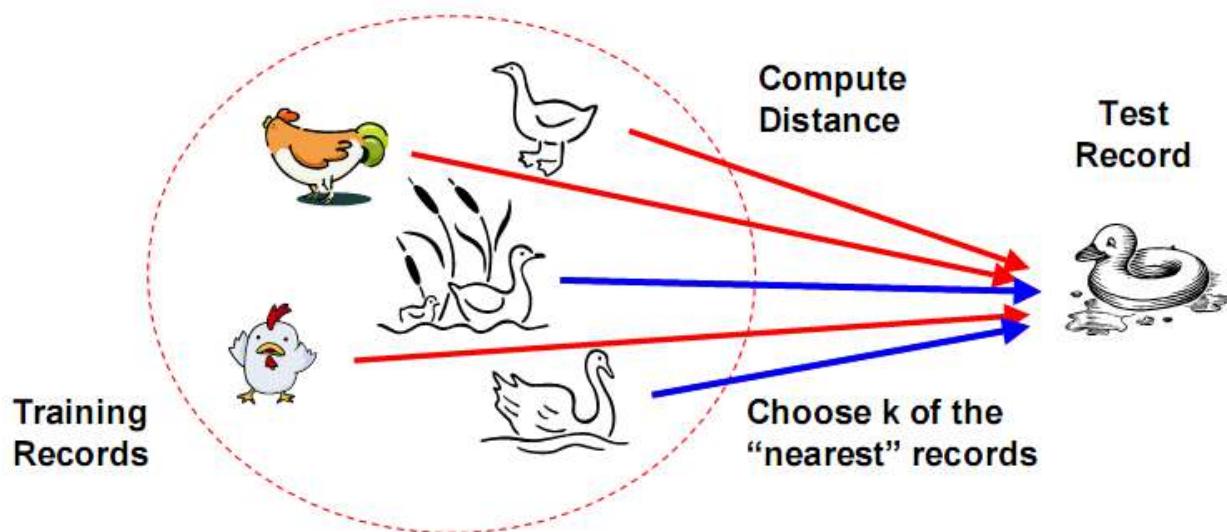
- Se memorează efectiv instanțele de antrenare și se folosesc pentru a prezice clasele instanțelor noi

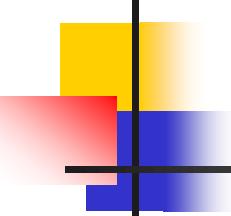
Instanță nouă

Atr1	.....	AtrN

# Cei mai apropiati $k$ vecini

- engl. “*k*-Nearest Neighbor”, *k*-NN
- Se folosesc cele mai apropiate  $k$  instanțe pentru a realiza clasificarea
- „*Dacă merge ca o rață și măcăne ca o rață, atunci este probabil o rață*”



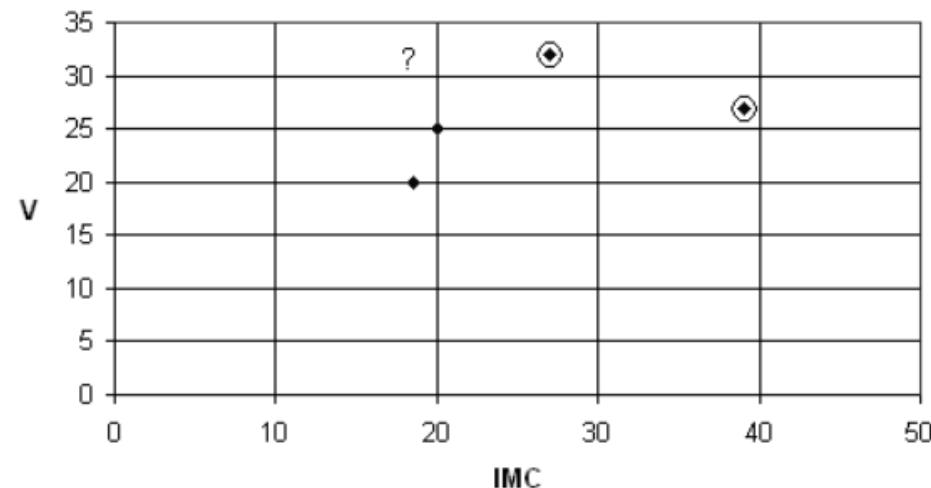


# Clasificarea $k$ -NN

- Necesită:
  - Multimea instanțelor de antrenare
  - O metrică de distanță pentru a calcula distanța dintre instanțe
  - Valoarea lui  $k$ , numărul de vecini apropiati,  $k \geq 1$
- Pentru a clasifica o instanță nouă:
  - Se calculează distanța față de instanțele de antrenare
  - Se identifică cei mai apropiati  $k$  vecini
  - Se folosesc clasele acestor vecini pentru a determina clasa noii instanțe, de exemplu, prin vot majoritar

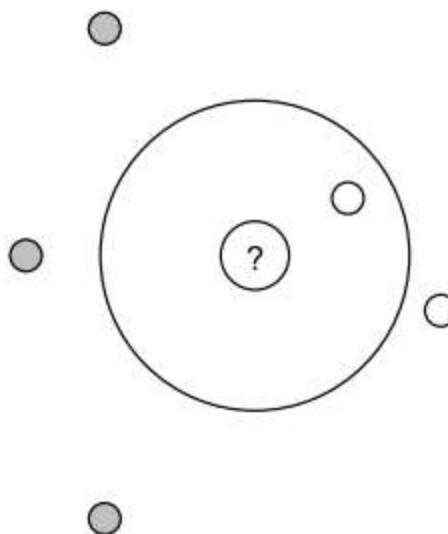
# Reprezentarea instanțelor

- Pentru  $n$  atribute, instanțele pot fi văzute ca puncte într-un spațiu  $n$ -dimensional
- De exemplu, clasificarea riscului unor pacienți
- Atribute:
  - Indicele masei corporale  $IMC (= G / l^2)$
  - Vârstă  $V$
- Instanțe:
  - $IMC = 18.5, V = 20$
  - $IMC = 27, V = 32$
  - $IMC = 39, V = 27$
  - $IMC = 20, V = 25$

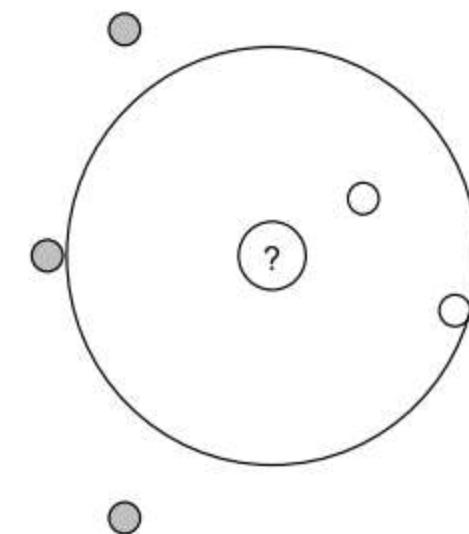


# Cei mai apropiati vecini

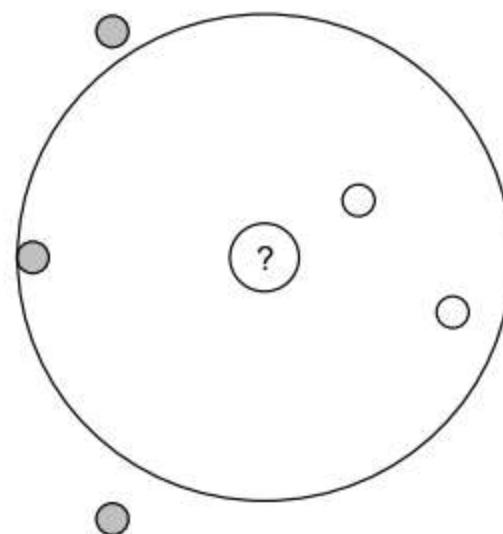
- Cei mai apropiati  $k$  vecini ai unei instanțe  $x$  sunt punctele cu distanțele cele mai mici față de  $x$



$k = 1$



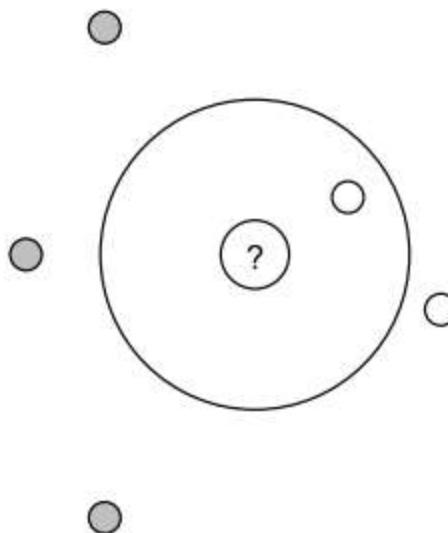
$k = 2$



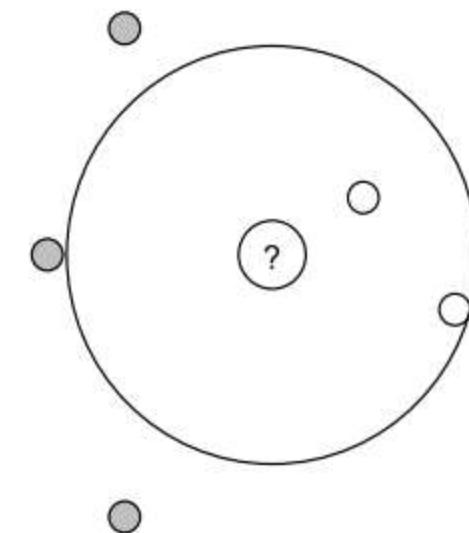
$k = 3$

# Numărul de vecini

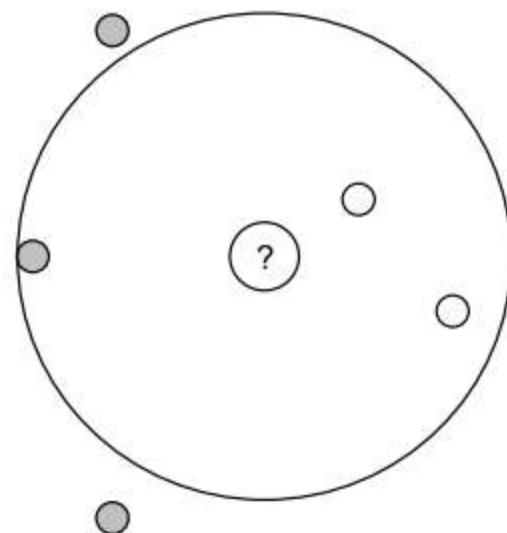
- Când  $k$  este prea mic, clasificarea poate fi afectată de zgomot
- Când  $k$  este prea mare, vecinătatea poate include puncte din alte clase



$k = 1$

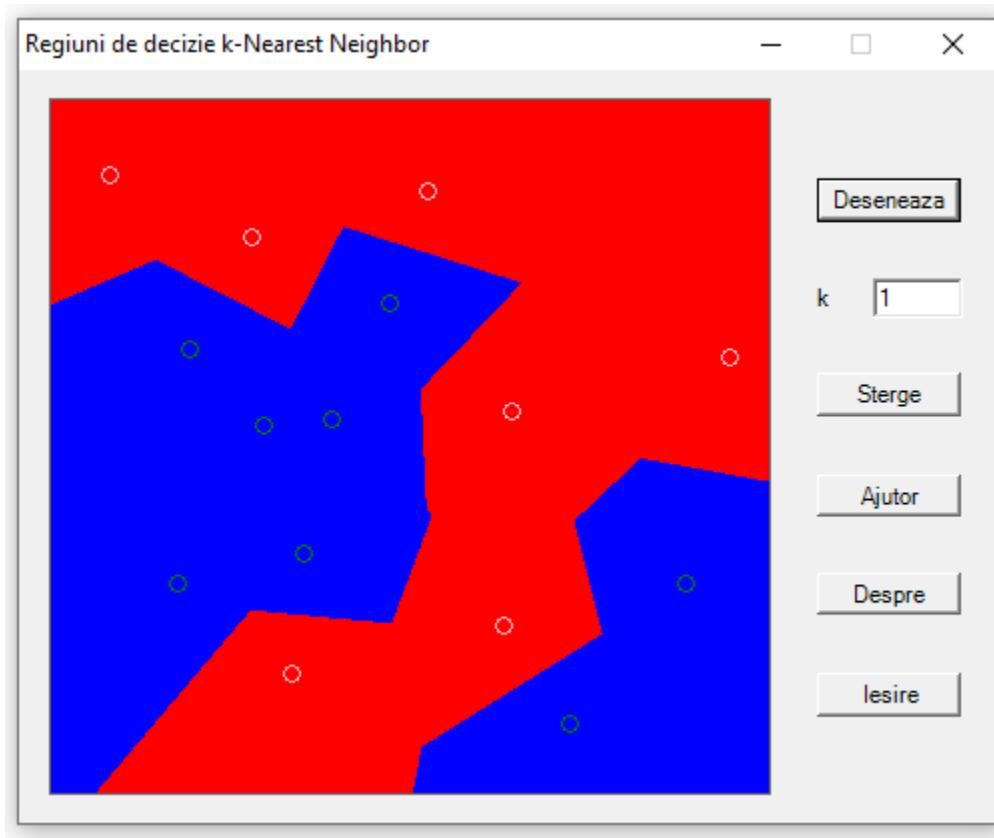


$k = 2$



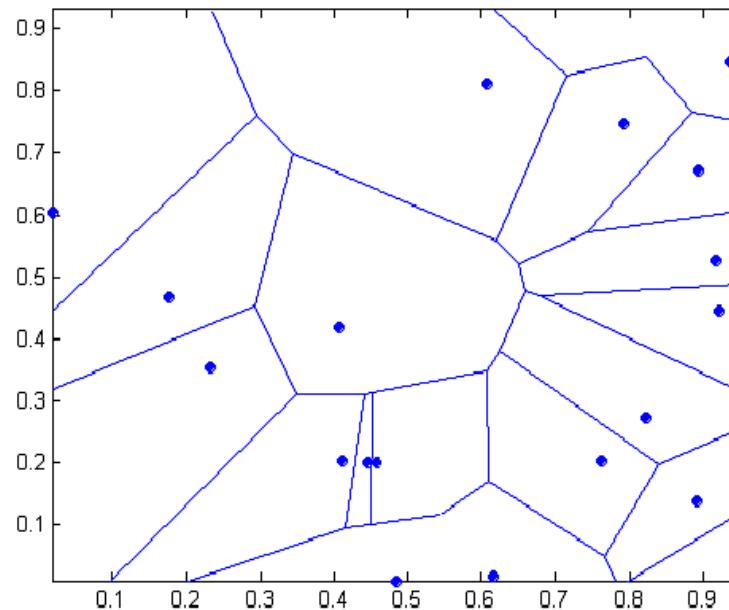
$k = 3$

# Regiuni de decizie



# 1-NN: diagrama Voronoi

- În interiorul unei celule, clasificarea tuturor noilor instanțe este determinată de instanța de antrenare corespunzătoare
- Instanțele noi din afara unei celule sunt mai apropiate de alte instanțe de antrenare

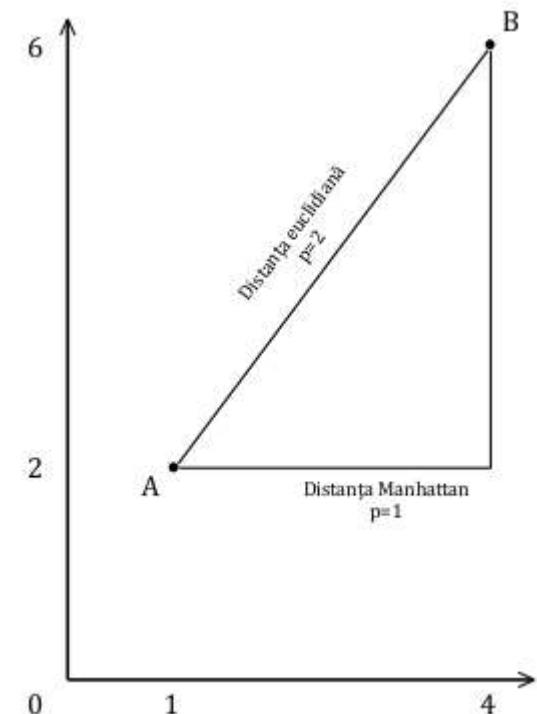


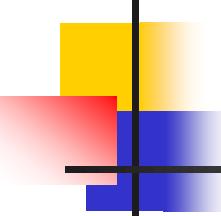
# Metrici de distanță

- Se folosesc în general particularizări ale distanței Minkowski

$$d(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

- Cele mai folosite metrici sunt:
  - Distanța euclidiană:  $p = 2$
  - Distanța Manhattan:  $p = 1$

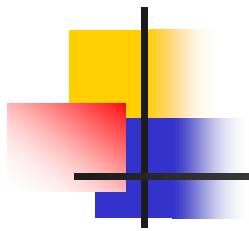




# Scalarea

- Se recomandă scalarea atributelor pentru a preveni dominarea măsurii de distanță de către un anumit atribut
- De exemplu:
  - Înălțimea unei persoane  $\in [1.5, 2.1]$  m
  - Greutatea unei persoane  $\in [50, 120]$  kg
  - Venitul unei persoane  $\in [25\,000, 1\,000\,000]$  lei/an
- Valorile atributelor sunt normalize:

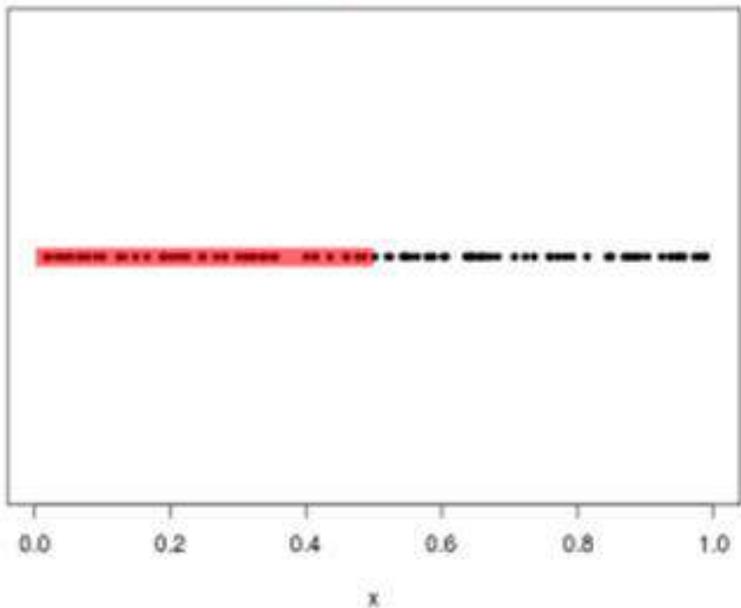
$$x'_i = \frac{(x_i - min_i)}{(max_i - min_i)} \in [0, 1]$$



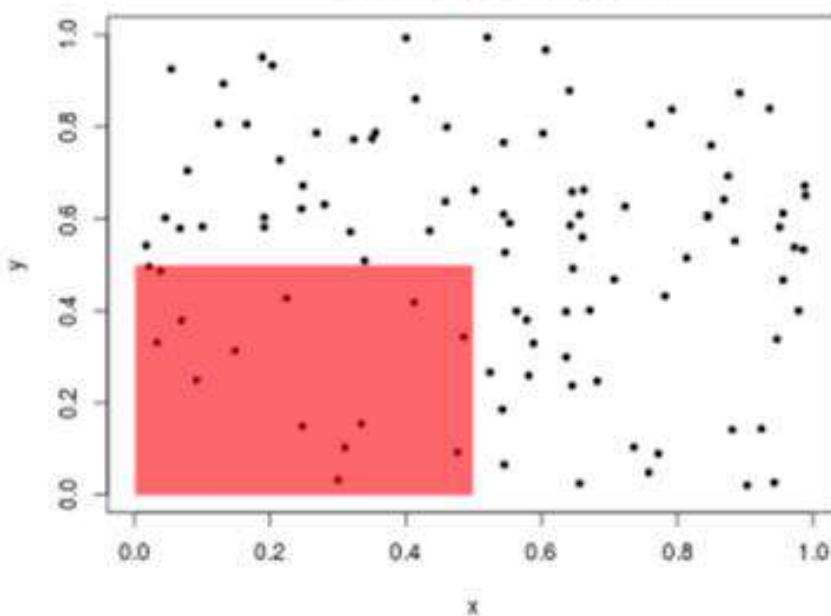
# Blestemul dimensionalității

- engl. “curse of dimensionality”
- Datele devin mai rare în spațiile multidimensionale
- Dacă numărul de atrbute este mare, este nevoie de mai multe instanțe de antrenare pentru a forma un model corect

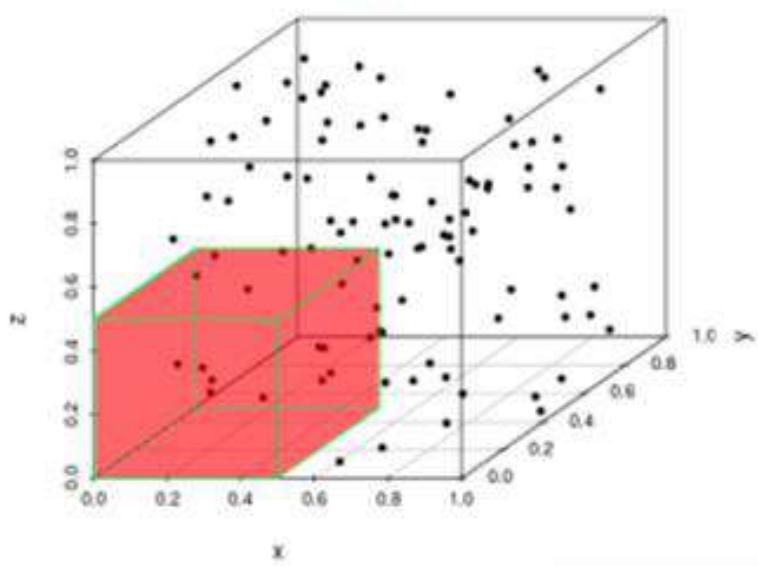
1-D: 42% of data captured.



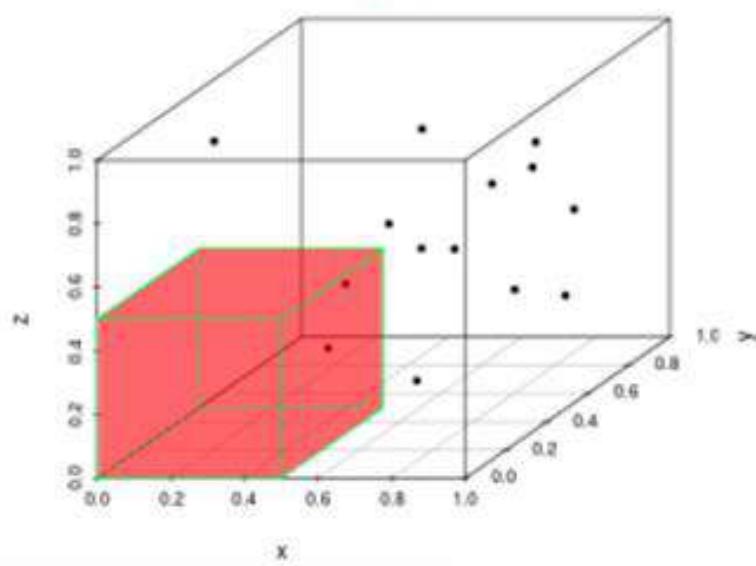
2-D: 14% of data captured.

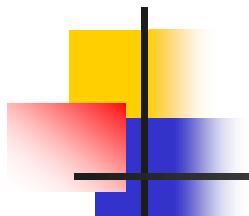


3-D: 7% of data captured.



4-D: 3% of data captured.





# Ponderarea instanțelor

- Vecinii mai apropiati au o influență mai mare la stabilirea clasei
- Influența fiecărui vecin poate fi ponderată pe baza distanței:
  - $w_i = 1 / d(x_q, x_i)^2$
  - Dacă  $d(x_q, x_i) = 0$ , atunci  $f(x_q) = f(x_i)$ , adică se returnează clasa instanței de antrenare din acel punct

# Exemplu: calcularea distanțelor

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Noua instanță:  $x_q = (\text{No}, \text{Married}, 80\text{K})$
- De exemplu:  $d(x_q, x_4) = \dots$ 
  - $d_{\text{Refund}} = 1$  ( $\text{No} \neq \text{Yes}$ )
  - $d_{\text{Status}} = 0$  ( $\text{Married} = \text{Married}$ )
  - $d_{\text{Income}} = \dots$ 
    - $I_q = 80$ , normalizat între 60 și 220  $\rightarrow 0.125$
    - $I_4 = 120$ , normalizat  $\rightarrow 0.375$
    - $d_{\text{Income}} = 0.25$
  - $d(x_q, x_4) = \sqrt{1^2 + 0^2 + 0.25^2} = 1.031$

# Exemplu: calcularea distanțelor

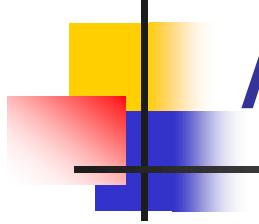
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Noua instantă:  $x_q = (\text{No}, \text{Married}, 80\text{K})$
- $d(x_q, x_1) = 1.442$
- $d(x_q, x_2) = 0.125$
- $d(x_q, x_3) = 1.002$
- $d(x_q, x_4) = 1.031$
- $d(x_q, x_5) = 1.004$
- $d(x_q, x_6) = 0.125$
- $d(x_q, x_7) = 1.663$
- $d(x_q, x_8) = 1.000$
- $d(x_q, x_9) = 0.031$
- $d(x_q, x_{10}) = 1.002$
- Să presupunem  $k = 3$
- Cele mai apropiate 3 instanțe sunt:  $x_9$  (Cheat=No),  $x_2$  (No) și  $x_6$  (No)
- Deci:  $f(x_q) = \text{No}$

# Exemplu: ponderarea

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Noua instanță:  
 $x_q = (\text{No}, \text{Married}, 80\text{K})$ 
  - $w(x_1) = 1 / 1.442^2 = 0.481$  (No)
  - $w(x_2) = 1 / 0.125^2 = 64$  (No)
  - $w(x_3) = 0.996$  (No)
  - $w(x_4) = 0.941$  (No)
  - $w(x_5) = 0.992$  (Yes)
  - $w(x_6) = 64$  (No)
  - $w(x_7) = 0.362$  (No)
  - $w(x_8) = 1.000$  (Yes)
  - $w(x_9) = 1040.583$  (No)
  - $w(x_{10}) = 0.996$  (Yes)
- $k = 10$
- Se pot lua în calcul toate instanțele de antrenare
- Se transformă dintr-o metodă locală într-una globală
- Decizia: suma
- Voturi No: 1171.363
- Voturi Yes: 2.988
- Deci:  $f(x_q) = \text{No}$



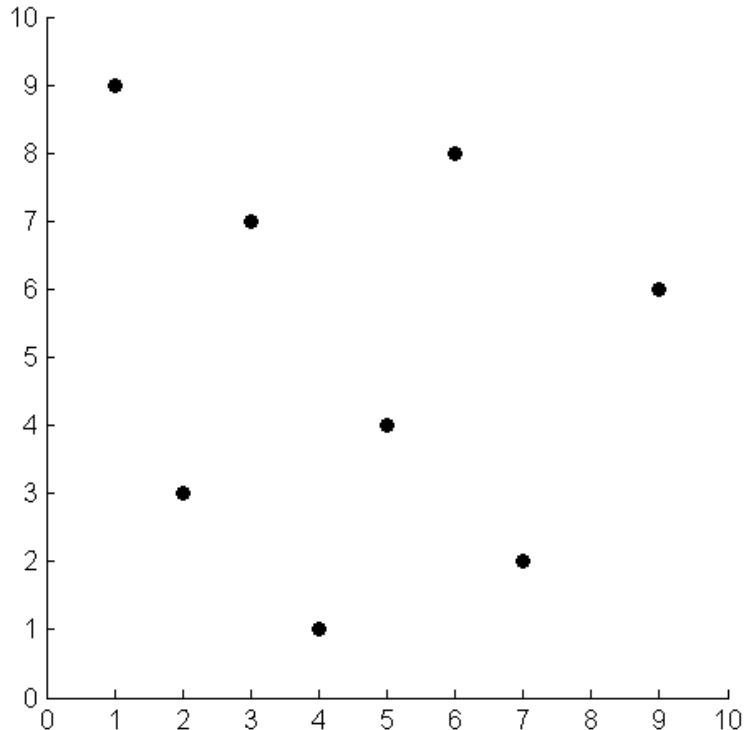
# Arbore kd

- engl. “*kd-trees*” (*k-dimensional trees*)
- Optimizează căutarea celui mai apropiat vecin sau a celor mai apropiati  $k$  vecini
- Fiecare mulțime de noduri se partionează după o dimensiune (atribut), de obicei în mod succesiv
- Pentru o anumită dimensiune, se partionează instanțele după valoarea mediană

# Construirea arborelui

- Exemplu:

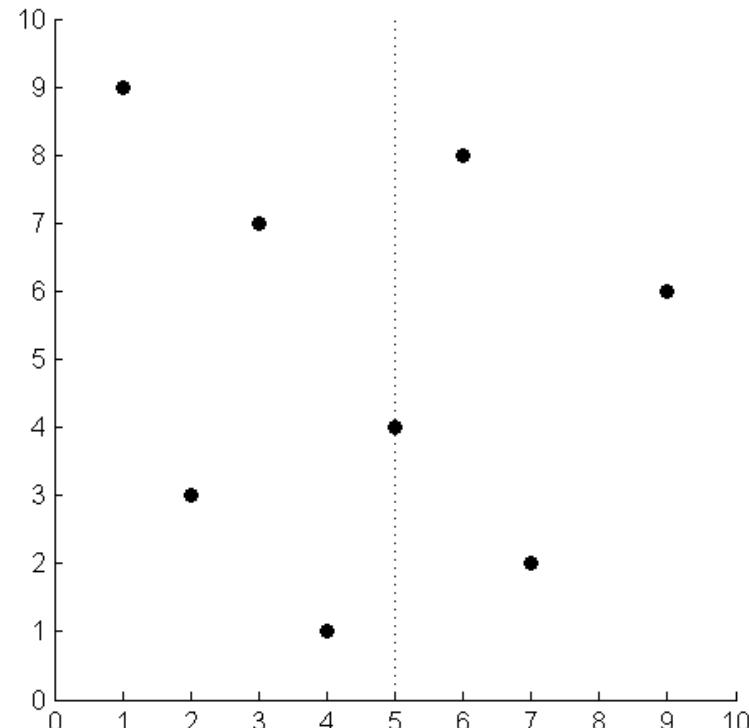
- 1,9
- 2,3
- 3,7
- 4,1
- 5,4
- 6,8
- 7,2
- 9,6



# Construirea arborelui

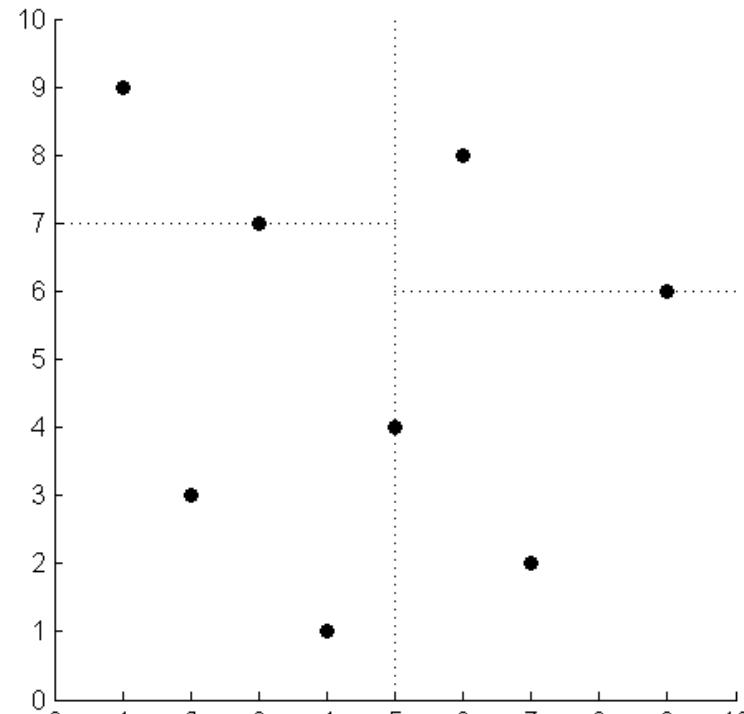
- Partiționare după primul atribut,  $x$
- Se sortează instanțele după  $x$  și se alege medianul: 5

- 1,9
- 2,3
- 3,7
- 4,1
- 5,4
- 6,8
- 7,2
- 9,6



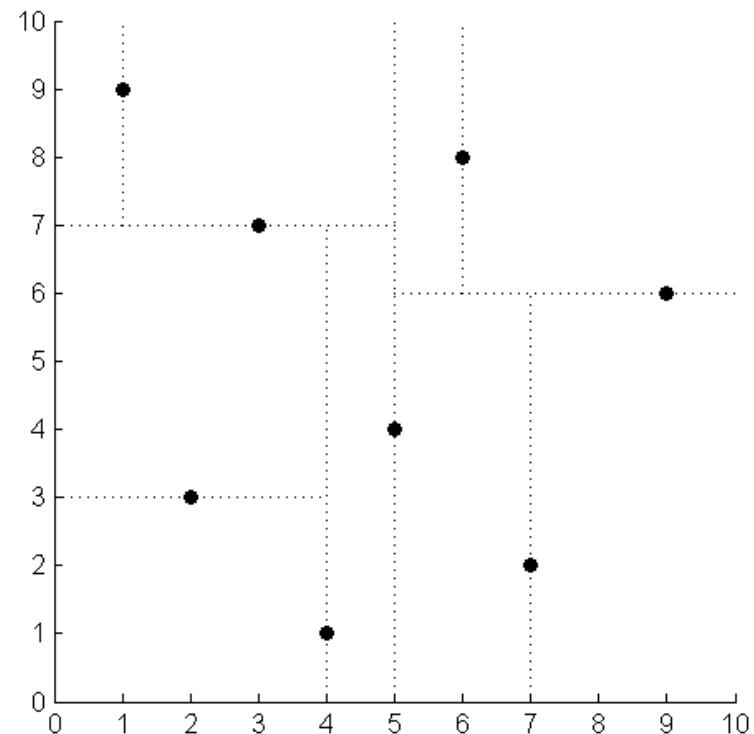
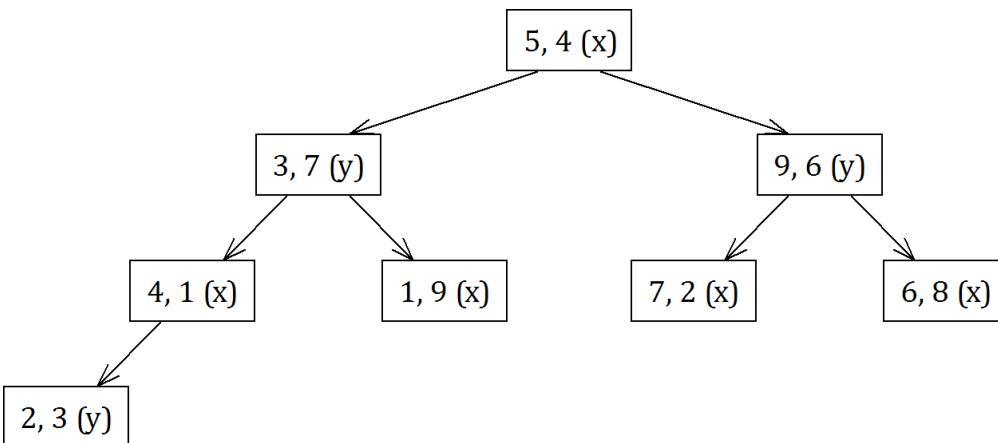
# Construirea arborelui

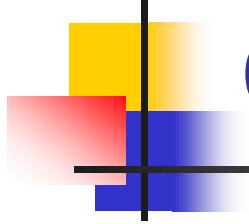
- Partiționarea submulțimilor rezultate (fiii din stânga și dreapta) după al doilea atribut,  $y$ 
  - 4,1
  - 2,3
  - 3,7
  - 1,9
  - 7,2
  - 9,6
  - 6,8



# Construirea arborelui

- Si aşa mai departe, considerând succesiv atributele  $x$  și  $y$

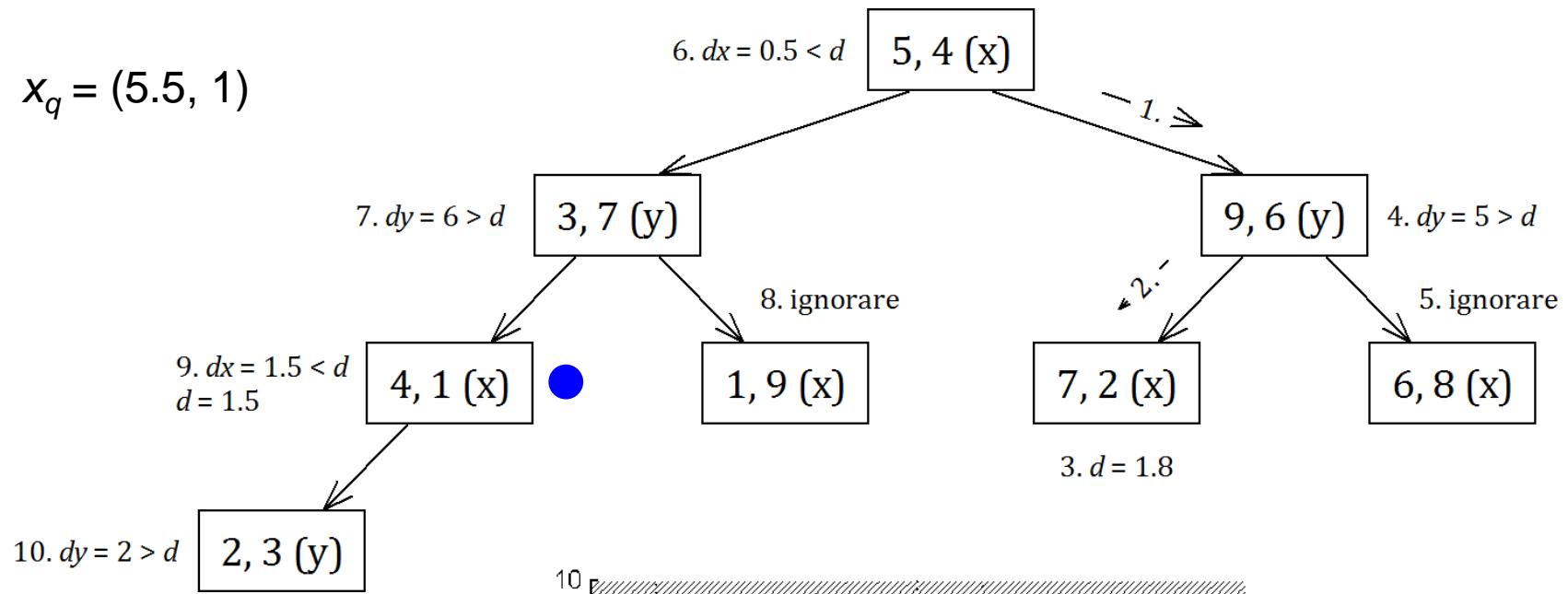




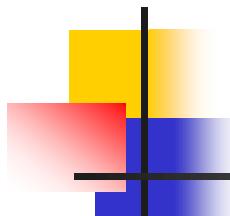
# Căutarea celui mai apropiat vecin

- $x_q = (5.5, 1)$

$$x_q = (5.5, 1)$$



Regiunile hasurate  
nu mai sunt vizitate

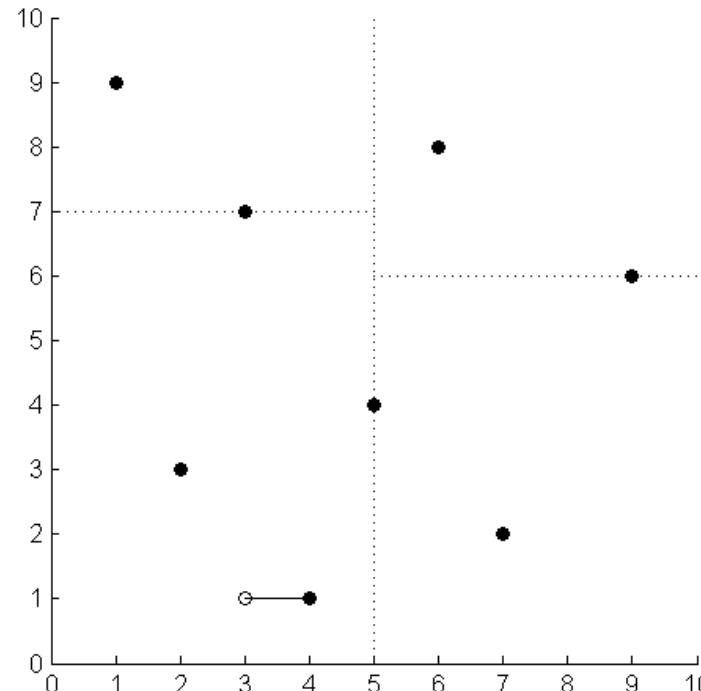


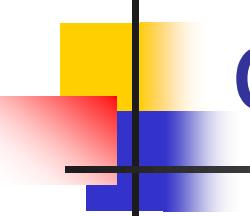
# Complexitate

- $d$  dimensiuni,  $n$  instanțe,  $k$  vecini
- Construcția:  $O(n \log n)$
- Căutarea:  $O\left(n^{1-\frac{1}{d}} + k\right)$
- Dacă  $d$  este mare, complexitatea se apropiie de  $O(n)$ , căutarea liniară
- Pentru arbori echilibrați și instanțe distribuite uniform aleatoriu, în cazul mediu, cel mai apropiat vecin este găsit în  $O(\log n)$

# Variantă aproximativă

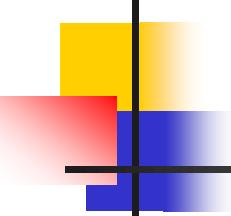
- Se construiește arborele, posibil pe un număr limitat de niveluri
- Se găsește cel mai apropiat vecin doar în mulțimea de instanțe a nodului de care aparține punctul țintă





# Clasificarea bazată pe cei mai apropiati vecini

- Clasificatorii  $k$ -NN sunt **clasificatori pasivi** (*lazy learners*)
  - Modelul nu este construit explicit
  - Efortul de calcul se face la clasificarea instanțelor noi
- Arborii de decizie sunt **clasificatori activi** (*eager learners*)
  - Modelul este construit explicit și aici apare efortul de calcul
  - Clasificarea instanțelor noi este rapidă
- Pentru  $k$ -NN, rata de eroare **la antrenare** este mică; dacă mulțimea de antrenare nu este afectată de zgomot, atunci rata de eroare este 0
  - Dar acest lucru nu garantează și o capacitate bună de generalizare



# Concluzii

- Metodele de clasificare prezentate aparțin unor paradigmăe diferite
- **Arborii de decizie** partitioanează recursiv mulțimea de instanțe cu ajutorul unui test de atribută
- **Metoda Bayes naivă** se bazează pe teorema lui Bayes, dar cu presupunerea că valorile atributelor instanțelor sunt independente dată fiind clasa
- **Clasificarea bazată pe cei mai apropiati vecini** presupune memorarea instanțelor de antrenare și folosirea acestora împreună cu o metrică de distanță pentru a prezice clasele instanțelor noi



# Învățare automată

## 3. Clasificarea bazată pe ansambluri. Selectia trăsăturilor

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)

Florin Leon, Invatare automata, [http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)

# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

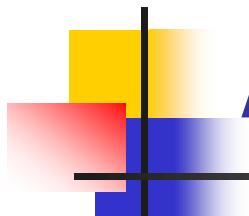
1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
3. Agregarea în stivă
4. Selectia trăsăturilor



# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

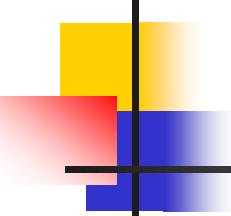
1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
3. Agregarea în stivă
4. Selectia trăsăturilor





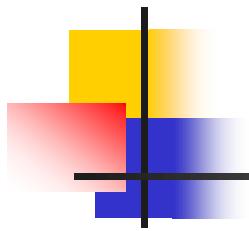
# Ansambluri simple

- Se pot rula separat mai mulți algoritmi pe o mulțime de date
  - Arbori de decizie, metoda Bayes naivă, cei mai apropiati  $k$  vecini, rețele neuronale etc.
- Pentru o instanță de test, fiecare algoritm dă un vot privind apartenența la o clasă
- Clasa cu cele mai multe voturi este răspunsul ansamblului
- Pentru regresie, se face media rezultatelor individuale
- Justificare: este puțin probabil ca toți algoritmii să facă aceleasi erori
- Din păcate, erorile algoritmilor sunt deseori corelate

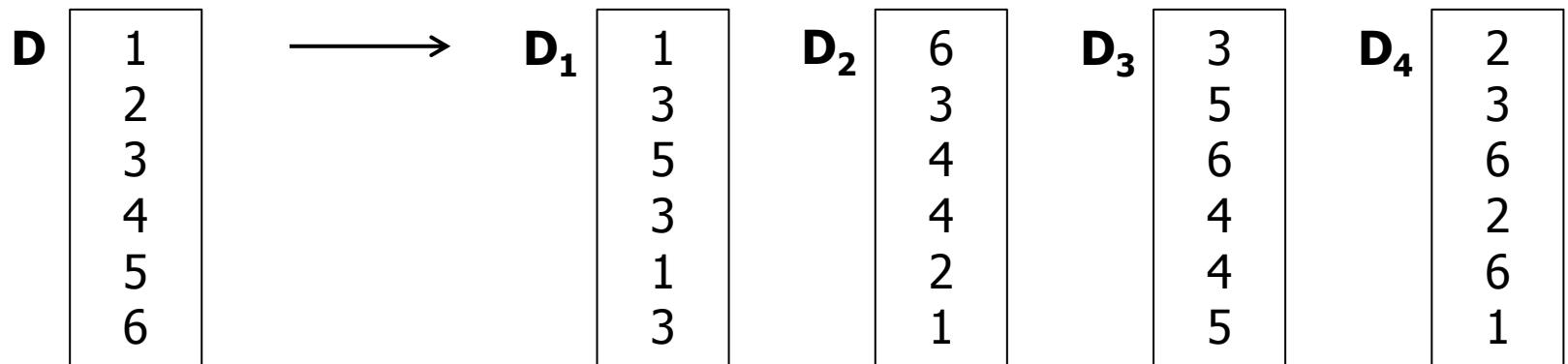


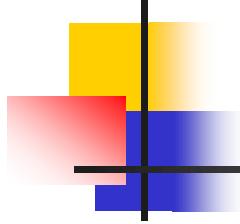
# Bagging

- engl. “bootstrap aggregating”
- Dintr-o mulțime de date  $D$  cu  $n$  instanțe se creează  $m$  mulțimi de date *bootstrapped*
- Fiecare mulțime este compusă tot din  $n$  instanțe, **eșantionate uniform cu înlocuire** din mulțimea  $D$ 
  - Pentru fiecare element dintr-o mulțime  $D_i$ , care trebuie generată, se alege o instanță aleatorie din  $D$
  - Procesul se repetă de  $n$  ori, până se completează toată mulțimea  $D_i$
  - Aceeași instanță poate fi aleasă de mai multe ori



# Exemplu

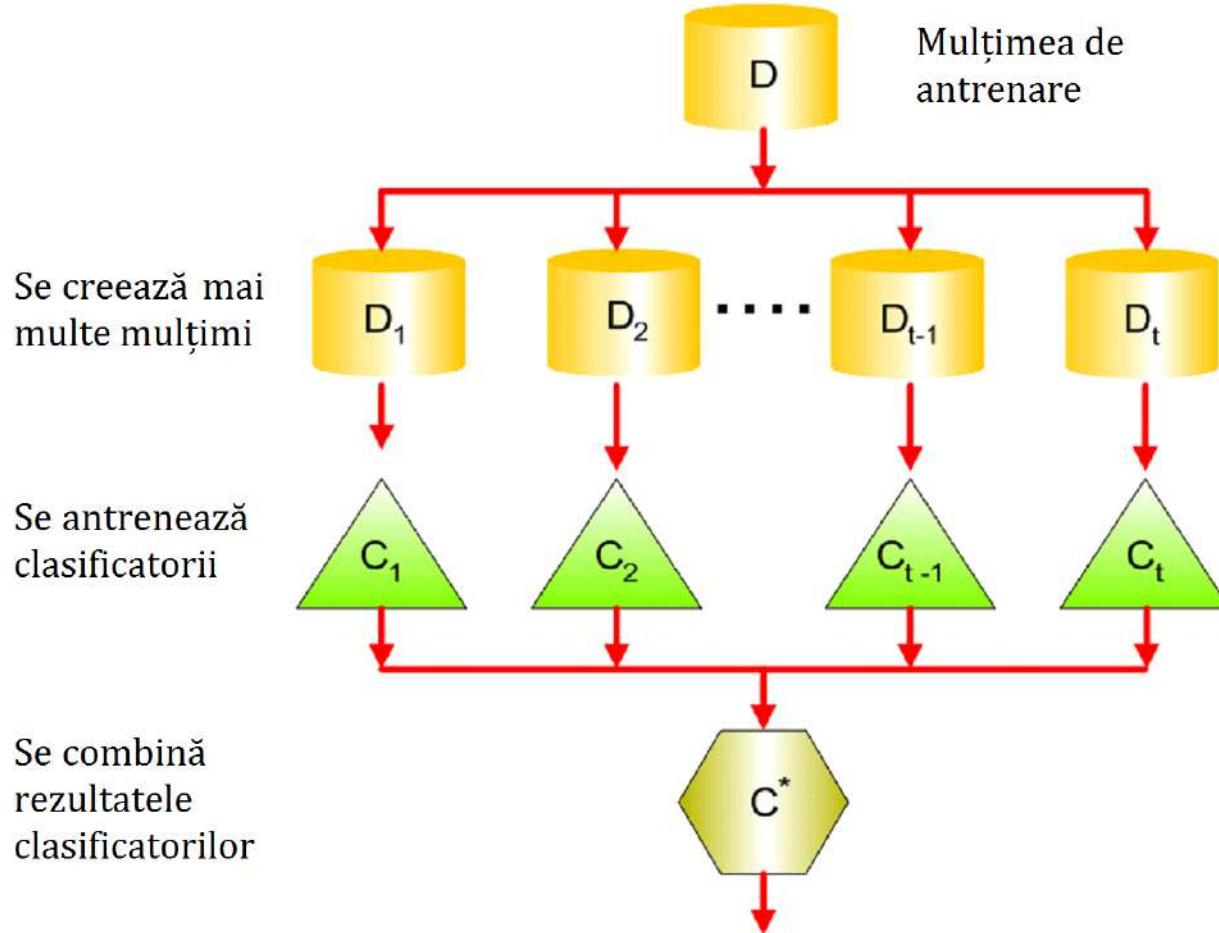


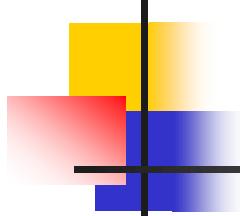


# Bagging

- Același algoritm de clasificare se aplică pentru fiecare mulțime
- Rezultă  $m$  modele
- Pentru o instanță de test, fiecare model dă un vot privind apartenența la o clasă
- Clasa cu cele mai multe voturi este răspunsul ansamblului

# Bagging





# Caracteristici

- Multimile rezultate sunt similare, dar nu foarte similare
- Probabilitatea ca o instanță să *nu* fie selectată la o eșantionare este  $P_1 = 1 - 1/n$
- Probabilitatea ca o instanță să nu fie selectată deloc într-o mulțime cu  $n$  instanțe este  $P_n = (1 - 1/n)^n$
- Când  $n \rightarrow \infty$ ,  $P_n \rightarrow 1/e \approx 0.368$
- Numai 63% din instanțe vor fi prezente într-o anumită mulțime *bootstrapped*

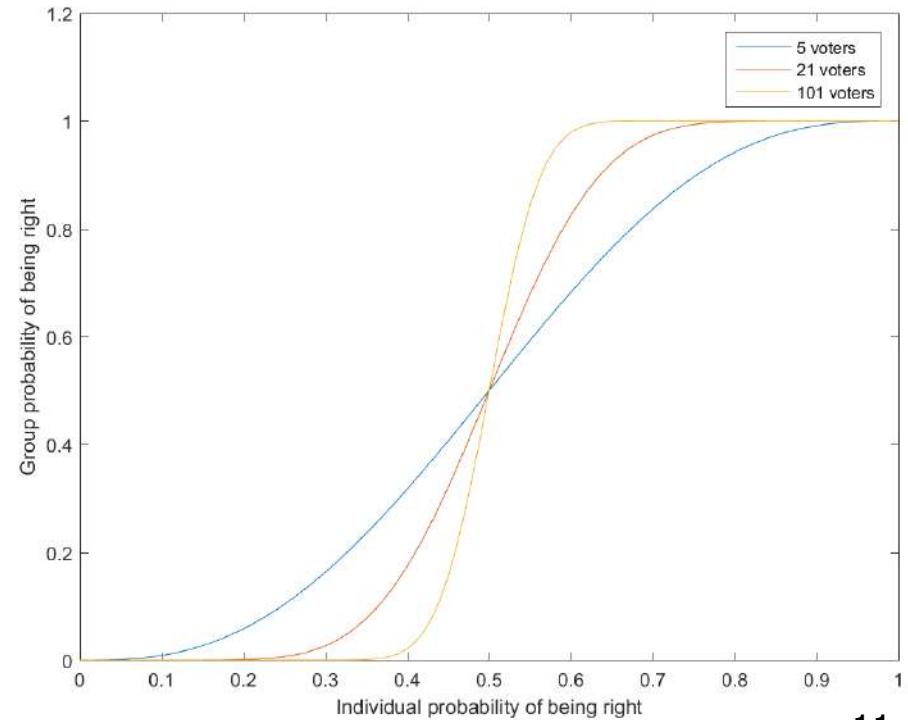
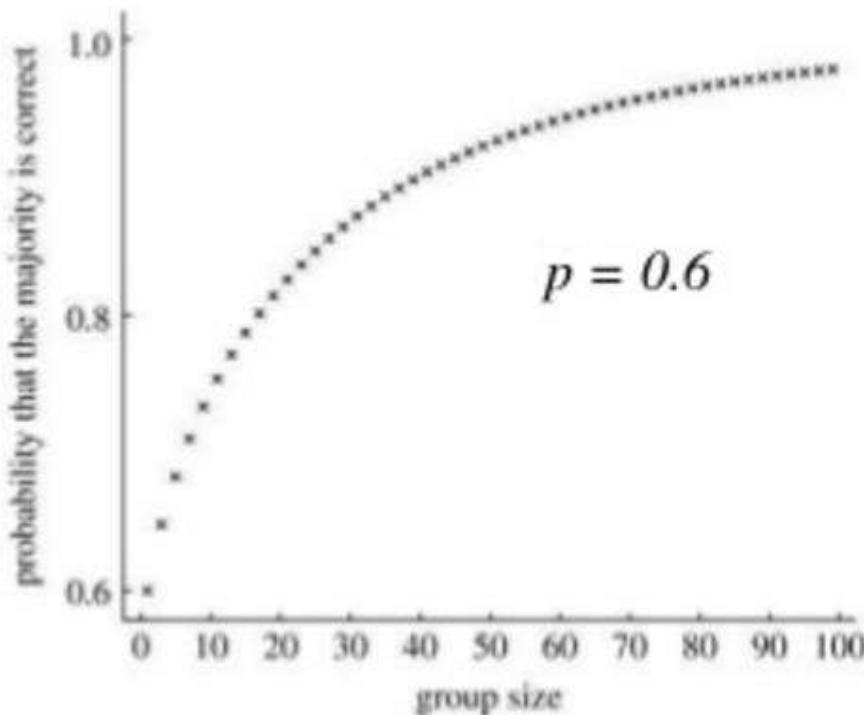
# Teorema juriului a lui Condorcet

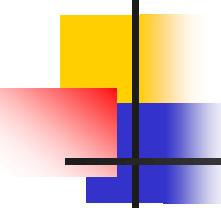
- Fie  $m$  votanți care trebuie să ia o decizie prin vot majoritar (decizia care întrunește jumătate + 1 din voturi)
- Dacă fiecare votant are o probabilitate  $p > 0.5$  de a lua decizia corectă, adăugarea de noi votanți crește probabilitatea ca ansamblul lor să ia decizia corectă
- Dacă  $p < 0.5$ , adăugarea de noi votanți scade calitatea deciziei

$$p_m = \sum_{i=\lceil m/2 \rceil}^m \left( \frac{m!}{(m-i)! \cdot i!} \right) \cdot p^i \cdot (1-p)^{m-i}$$

# Teorema juriului a lui Condorcet

- Dacă  $p > 0.5$ , atunci  $p_m > p$
- Dacă  $p > 0.5$  și  $m \rightarrow \infty$ , atunci  $p_m \rightarrow 1$





# Pseudocode *bagging*

**Input:**

- $D$ , a set of  $d$  training tuples;
- $k$ , the number of models in the ensemble;
- a learning scheme (e.g., decision tree algorithm, backpropagation, etc.)

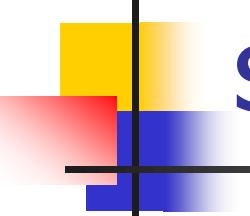
**Output:** A composite model,  $M^*$ .

**Method:**

- (1) **for**  $i = 1$  to  $k$  **do** // create  $k$  models:
- (2)     create bootstrap sample,  $D_i$ , by sampling  $D$  with replacement;
- (3)     use  $D_i$  to derive a model,  $M_i$ ;
- (4) **endfor**

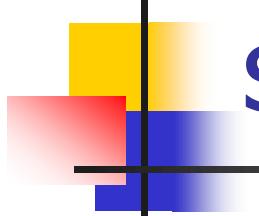
**To use the composite model on a tuple,  $X$ :**

- (1) **if** classification **then**
- (2)     let each of the  $k$  models classify  $X$  and return the majority vote;
- (3) **if** prediction **then**
- (4)     let each of the  $k$  models predict a value for  $X$  and return the average predicted value;



# Descompunerea subiectivitate-varianță

- Eroarea totală =  
    subiectivitate + varianță + zgomot
- Subiectivitatea (*bias*)
  - Reprezintă rata de eroare a unui model pentru o anumită problemă
  - Poate fi diferită de 0 chiar dacă există un număr infinit de mulțimi de antrenare sau modele independente
  - Apare deoarece un model nu se potrivește suficient de bine cu problema



# Descompunerea subiectivitate-varianță

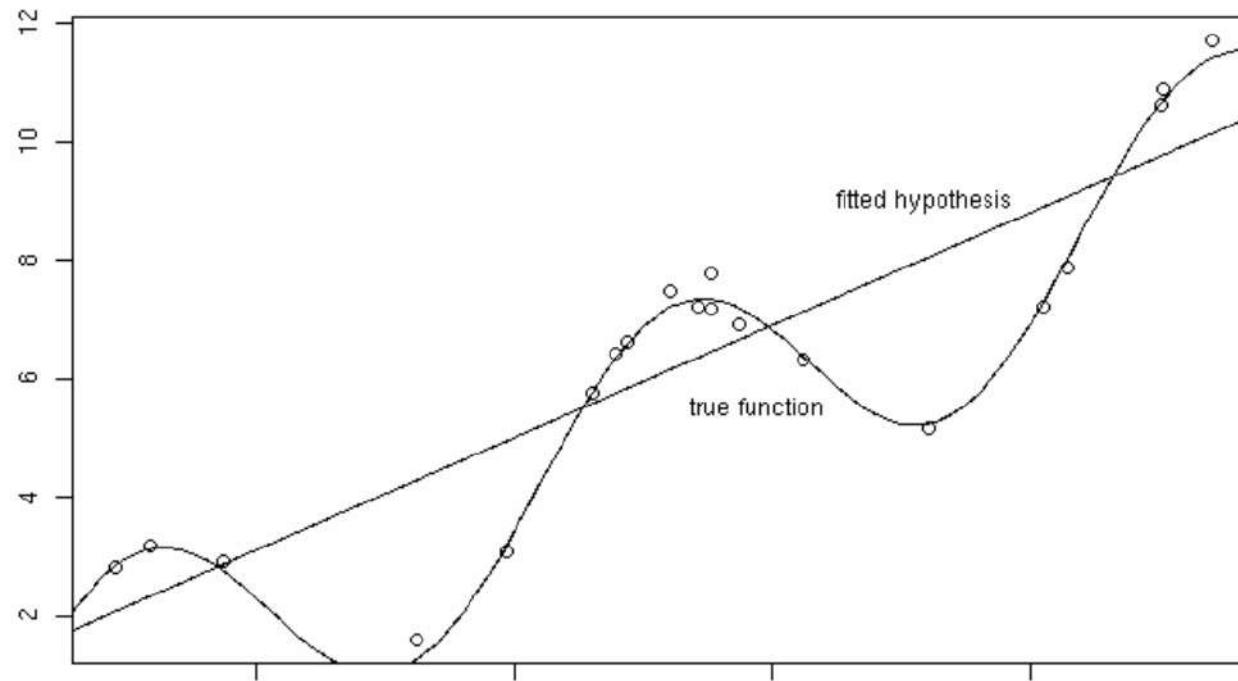
## ■ Varianță

- În practică, multimile de antrenare sunt finite și deci nu reprezintă perfect distribuția instanțelor de antrenare
- Varianța este valoarea așteptată a erorii cauzată de acest fapt
- Când multimile sunt mai mari, varianța este mai mică
- Când modelul este mai complex, varianța este mai mare
- *Bagging*-ul reduce varianța

# Exemplu

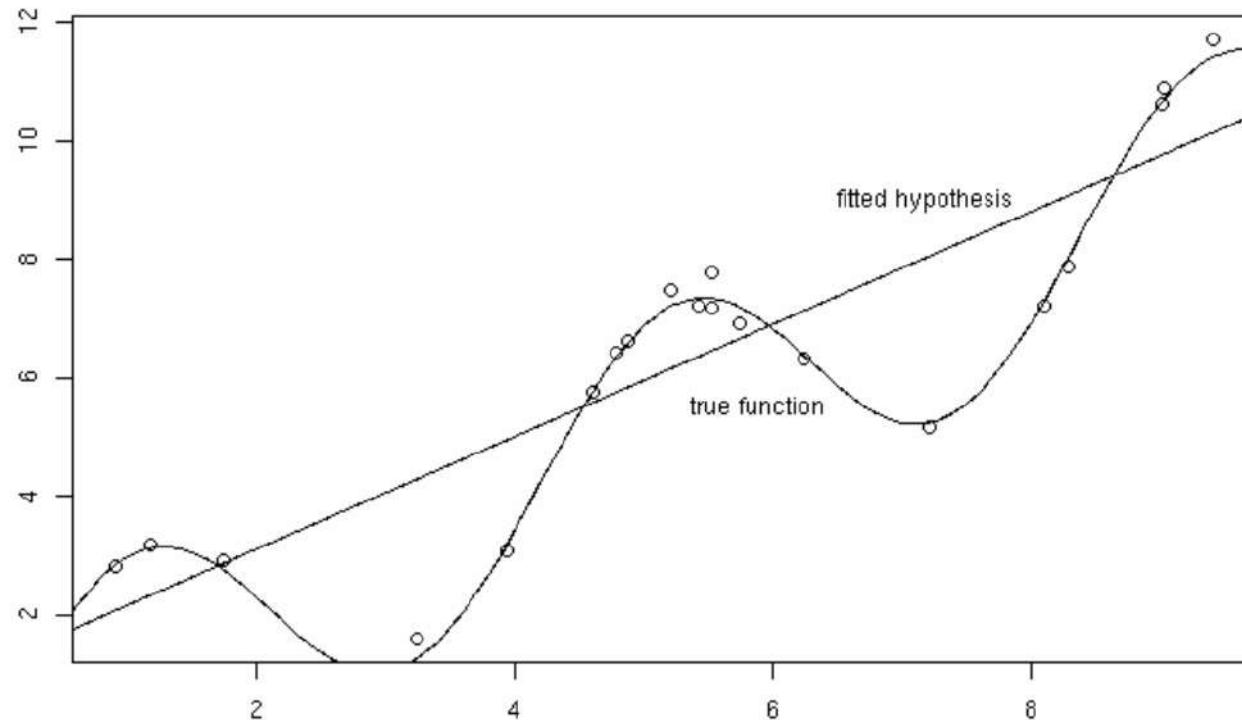
$N(0, 0.2)$  reprezintă numere aleatorii generate dintr-o distribuție normală cu  $\mu = 0$  și  $\sigma = 0.2$

- Avem  $n = 20$  de puncte eșantionate, afectate de zgomot, din funcția:  
 $f(x) = x + 2 \sin(1.5x) + N(0, 0.2)$
- Folosim un model liniar



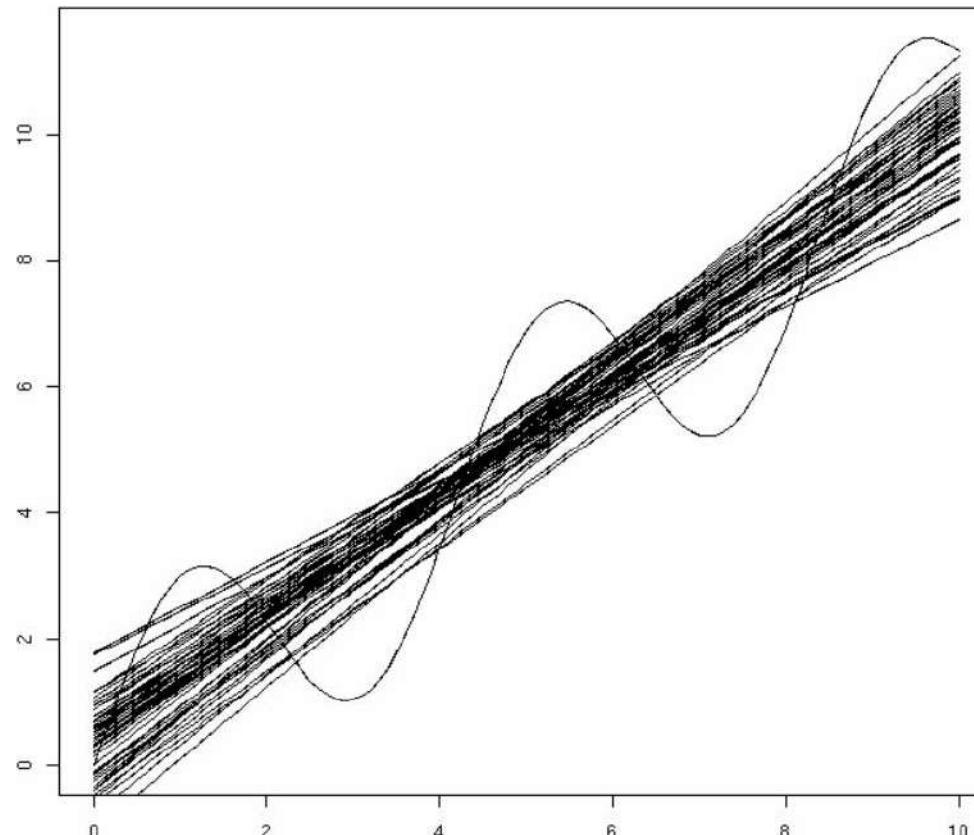
# Subiectivitatea

- Modelul liniar nu poate învăța funcția sinusoidală

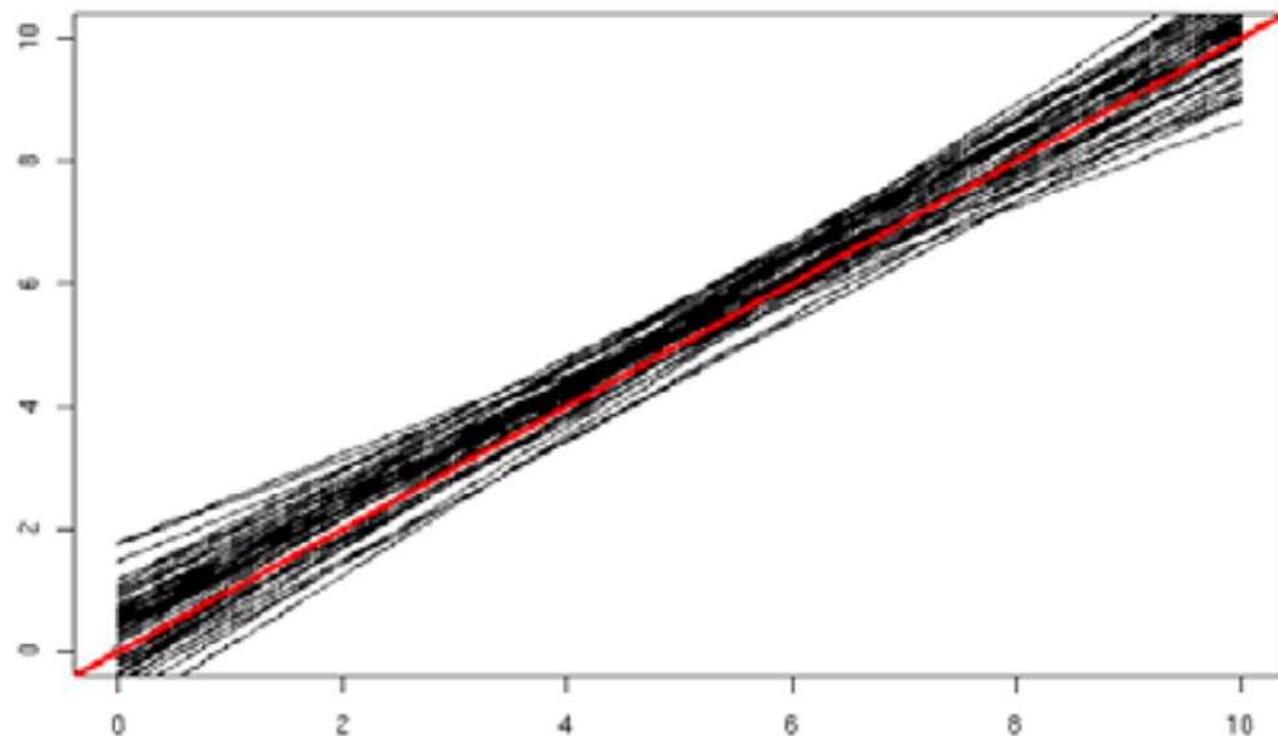


# Bagging

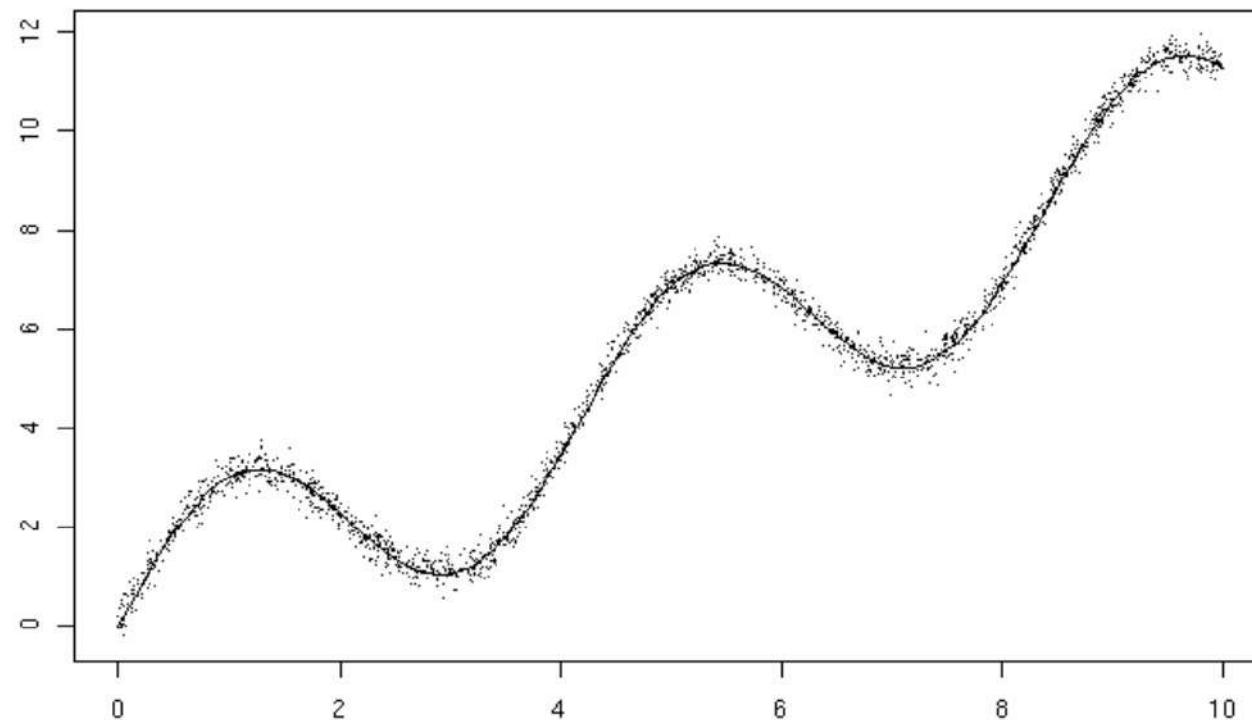
- Bagging cu  $m = 50$  de multimi, fiecare cu  $n = 20$  de instanțe (puncte)

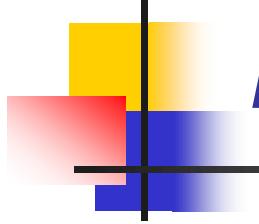


# Varianță



# Zgomotul





# *Random Tree*

- Algoritm asemănător cu ID3/C4.5, dar:
  - Într-un nod, se selectează aleatoriu  $d'$  atrbute din cele  $d$  ale datelor, cu  $1 \leq d' \leq d$ , și se face împărțirea după cel mai bun dintre acestea
  - $d'$  poate fi  $\lceil \log_2(d) \rceil$
  - Un atrbut poate fi folosit de mai multe ori
  - Partiționarea continuă până când eroarea de clasificare ajunge la 0
  - Mai rar, se poate da ca parametru de intrare adâncimea maximă a arborelui generat

# Exemple

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

status = married : no

status = single

- | refund = yes : no

- | refund = no

- | | income < 77.5 : no

- | | income >= 77.5 : yes

status = divorced

- | refund = yes : no

- | refund = no : yes

status = married : no

status = single

- | income < 77.5 : no

- | income >= 77.5

- | | refund = yes : no

- | | refund = no : yes

status = divorced

- | refund = yes : no

- | refund = no : yes

refund = yes : no

refund = no

- | status = married : no

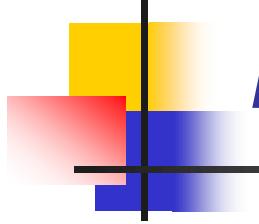
- | status = single

- | | income < 77.5 : no

- | | income >= 77.5 : yes

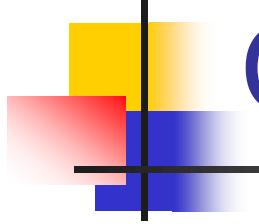
- | status = divorced : yes

Toți acești arbori au eroare 0 pe multimea de antrenare



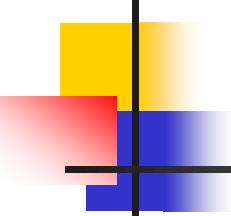
# Random Forest

- Se bazează pe votul a  $m$  arbori aleatorii, unde  $m$  este un parametru de intrare
- Se generează  $m$  mulțimi de antrenare *bootstrapped*
- Se antrenează câte un arbore aleatoriu pentru fiecare mulțime
- Pentru o instanță de test, fiecare arbore dă un vot privind apartenența la o clasă
- Clasa cu cele mai multe voturi este răspunsul pădurii aleatorii



# Caracteristici

- *Random Forest* este una din cele mai bune metode de clasificare „clasice”
- Subiectivitatea scade deoarece arborii componenti pot crește foarte mult, până capturează toate detaliile problemei
- Varianța scade datorită mulțimilor *bootstrapped* și medierii rezultatelor individuale ale arborilor componenti



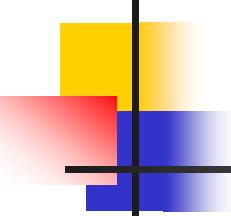
# Caracteristici

- Clasificatorii cu un grad ridicat de neliniaritate, cum ar fi arborii de decizie, beneficiază cel mai mult de avantajele *bagging*-ului
- Arborii de decizie sunt modele instabile: o modificare mică în multimea de antrenare poate cauza o modificare mare în arborele generat
- Modelele liniare sunt mai stabile și nu beneficiază prea mult de *bagging*
- Stabilitatea crește cu numărul de instanțe de antrenare și scade cu dimensionalitatea problemei

# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

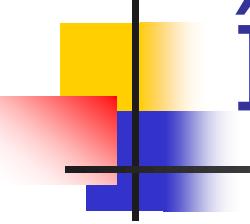
1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
  - 2.1. Detectorul facial Viola-Jones
3. Agregarea în stivă
4. Selectia trăsăturilor





# Învățarea PAC

- Învățarea **probabil aproximativ corectă** (*Probably Approximately Correct*, PAC)
- Se presupune că datele de antrenare și testare aparțin unei distribuții de probabilitate  $D$  definită pe domeniul  $X = \{0, 1\}^n$
- $f$  = funcția care trebuie învățată
- $h$  = ipoteza care aproximează  $f$
- $\varepsilon$  = eroarea maximă admisă în aproximarea lui  $f$
- $\delta$  = probabilitatea maximă de a greși, adică de a depăși această eroare maximă admisă

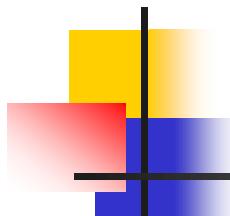


# Învățarea PAC

Fie  $H$  o clasă de funcții binare  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ . Spunem că  $H$  este învățabil PAC dacă există un algoritm  $L$  astfel încât  $\forall f \in H, \forall D$  o distribuție de probabilitate,  $\forall \varepsilon \in [0, 1/2)$ ,  $\forall \delta \in [0, 1/2)$  și o mulțime de instanțe de antrenare aleatorii alese din  $D$ ,  $L$  găsește ipoteza  $h$  care respectă relația:

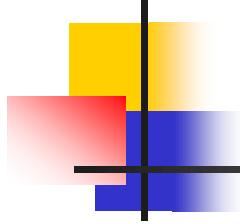
$$P(e(h, f) \leq \varepsilon) \geq 1 - \delta.$$

Prin urmare, problema PAC este de a găsi o ipoteză  $h$  care aproximează conceptul  $f$  cu o probabilitate de cel puțin  $1 - \delta$  ca eroarea dintre  $h$  și  $f$  să fie cel mult  $\varepsilon$ .



# Clasificatoare puternice și slabe

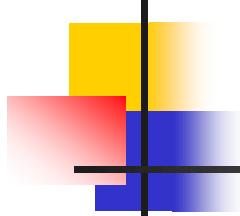
- **Clasificator puternic:** o metodă de clasificare ce respectă relația din slide-ul anterior pentru *toți*  $\varepsilon$  ( $\forall \varepsilon$ )
  - Adică o metodă cu eroare arbitrar de mică
- **Clasificator slab:** o metodă de clasificare ce respectă relația din slide-ul anterior pentru *cel puțin un*  $\varepsilon$  ( $\exists \varepsilon$ )
  - La limită,  $\varepsilon$  poate fi aproape 1/2
  - Este suficient ca metoda să fie *puțin* mai bună decât o decizie aleatorie ( $\varepsilon$  și  $\delta$  trebuie să fie mai mici decât 1/2 pentru orice distribuție)



# Boosting

- Aceste două tipuri de clasificatoare sunt echivalente
  - Un clasificator slab poate fi transformat într-unul puternic
- Adaboost (“adaptive boosting”) este algoritmul care a demonstrat acest lucru, prin construcție
- Adaboost construiește un clasificator puternic  $H$  ca o combinație liniară de clasificatori slabii  $h_t$ :

$$H(\mathbf{x}) = \operatorname{sgn} \left( \sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right) \quad H, h_t \in \{-1, 1\}$$



# Clasificatorul slab

- Se poate folosi orice clasificator, cu condiția ca rata sa de eroare să fie mai mică decât 50%
- De obicei, se folosește algoritmul *Decision Stump*, un arbore de decizie cu un singur nivel, în care criteriul de selecție al atributelor și pragurilor este numărul de erori rezultat

$$h_t = \arg \min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$$

- Se poate folosi și perceptronul cu un singur strat, sau algoritmi mai puternici precum *Naïve Bayes* sau *k-Nearest Neighbor*

# Decision Stump binar

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

în acest caz, orice atribut poate fi folosit pentru partitōnare

- Refund
  - = Yes  $\rightarrow$  Cheat = No (3/3)
  - $\neq$  Yes  $\rightarrow$  Cheat = No (4/7)
  - 3 erori (la No)
- Status
  - = Married  $\rightarrow$  Cheat = No (4/4)
  - $\neq$  Married  $\rightarrow$  Cheat = Yes (3/6)
  - 3 erori (la Yes)
- Income
  - $\leq 97.5 \rightarrow$  Cheat = Yes (3/6)
  - $> 97.5 \rightarrow$  Cheat = No (4/4)
  - 3 erori (la Yes)

# Decision Stump cu valori multiple

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

În acest caz, orice atribut poate fi folosit pentru partitōnare

- Refund
  - = Yes  $\rightarrow$  Cheat = No (3/3)
  - = No  $\rightarrow$  Cheat = No (4/7)
  - 3 erori
- Status
  - = Single  $\rightarrow$  Cheat = No (2/4)
  - = Married  $\rightarrow$  Cheat = No (4/4)
  - = Divorced  $\rightarrow$  Cheat = Yes (1/2)
  - 3 erori
- Income
  - $\leq 97.5$   $\rightarrow$  Cheat = Yes (3/6)
  - $> 97.5$   $\rightarrow$  Cheat = No (4/4)
  - 3 erori

# Pseudocod Adaboost

Given:  $(x_1, y_1), \dots, (x_m, y_m); x_i \in \mathcal{X}, y_i \in \{-1, 1\}$

Initialize weights  $D_1(i) = 1/m$

For  $t = 1, \dots, T$ :

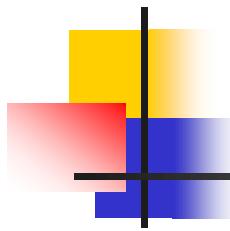
1. (Call *WeakLearn*), which returns the weak classifier  $h_t : \mathcal{X} \rightarrow \{-1, 1\}$  with minimum error w.r.t. distribution  $D_t$ ;
2. Choose  $\alpha_t \in R$ ,
3. Update

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where  $Z_t$  is a normalization factor chosen so that  $D_{t+1}$  is a distribution

Output the strong classifier:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

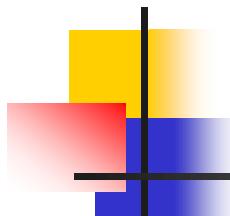


# Coeficientii $\alpha_t$

- Expresia pentru coeficientul  $\alpha_t$  în funcție de eroarea  $\varepsilon_t$  se obține în mod diferențial, prin minimizarea lui  $Z_t$
- Algoritmul minimizează o funcție de cost exponențială
  - Diferite tipuri de funcții de cost vor fi prezentate în cursul 4
- În funcție de varianta de algoritm Adaboost, există mai multe expresii:

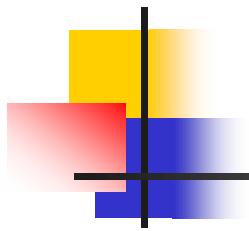
$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \quad \alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{1 + \varepsilon_t} \quad \text{și altele...}$$

- Dacă la un moment dat  $\varepsilon_t = 0$ , înseamnă că s-a ajuns la o clasificare perfectă și nu mai sunt necesare alte iterării

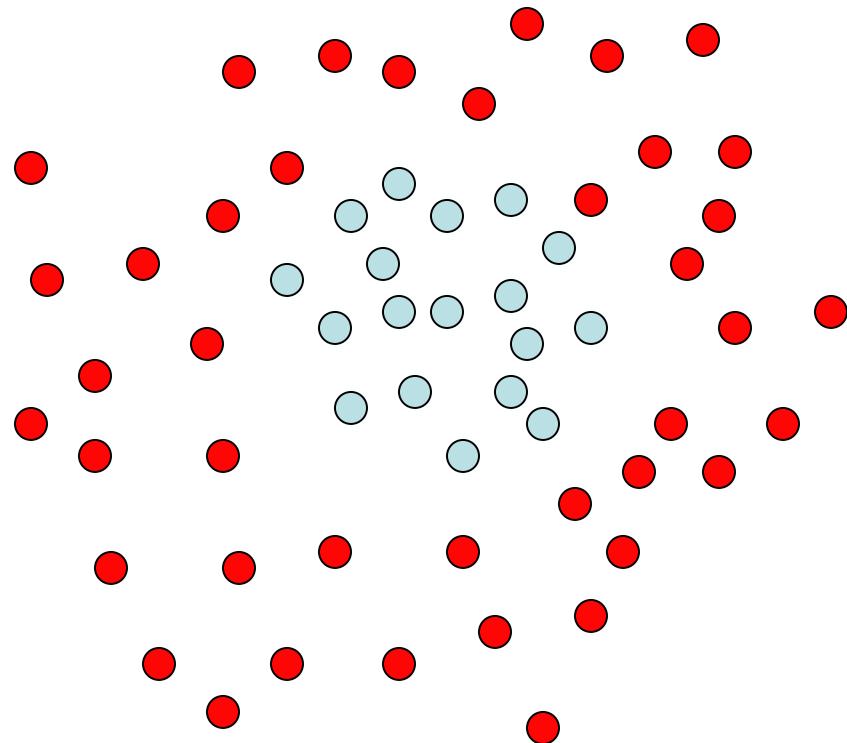


# Re-ponderarea instanțelor

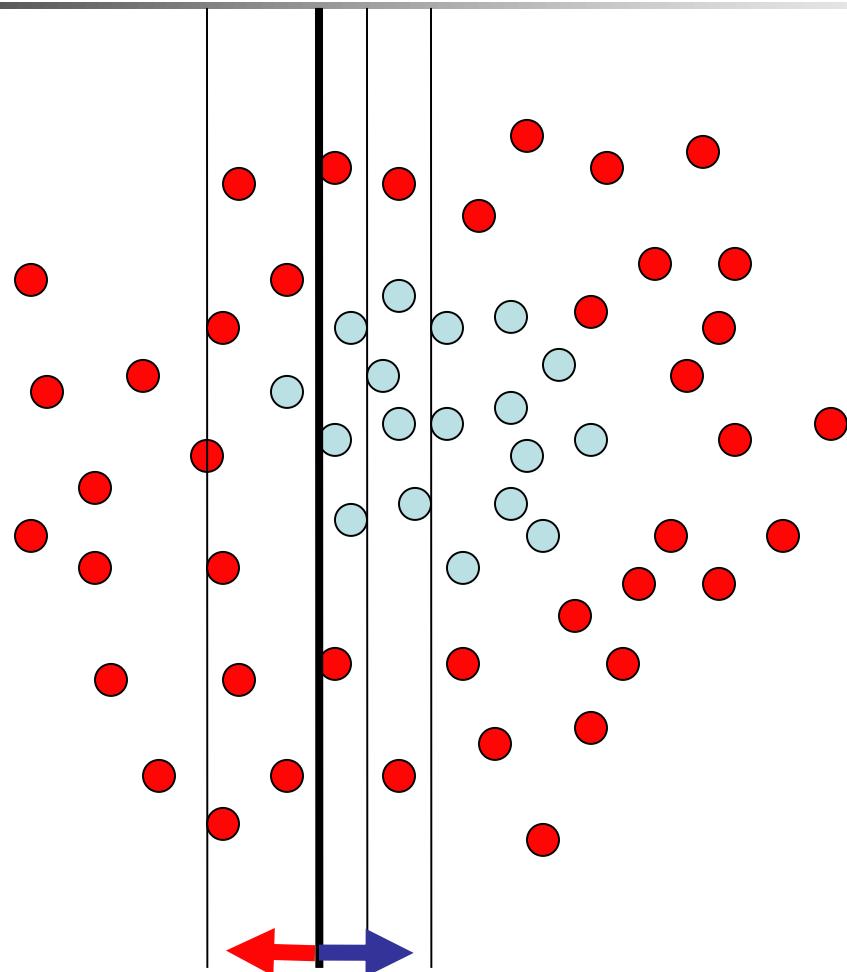
- În următoarea iterație:
  - Instanțele clasificate greșit vor avea o pondere mai mare
  - Instanțele clasificate corect vor avea o pondere mai mică

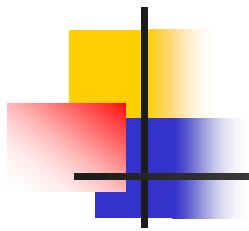


# Modul de funcționare

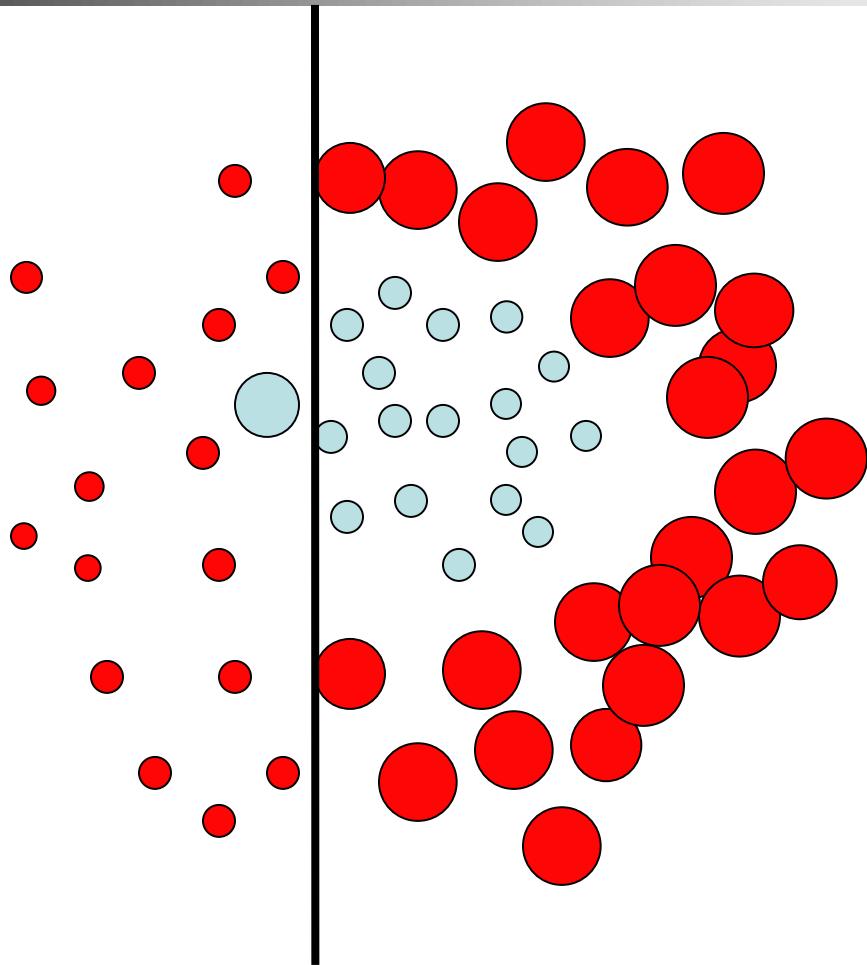


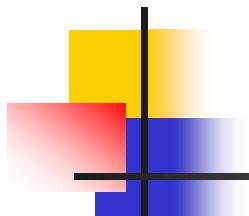
# Modul de funcționare



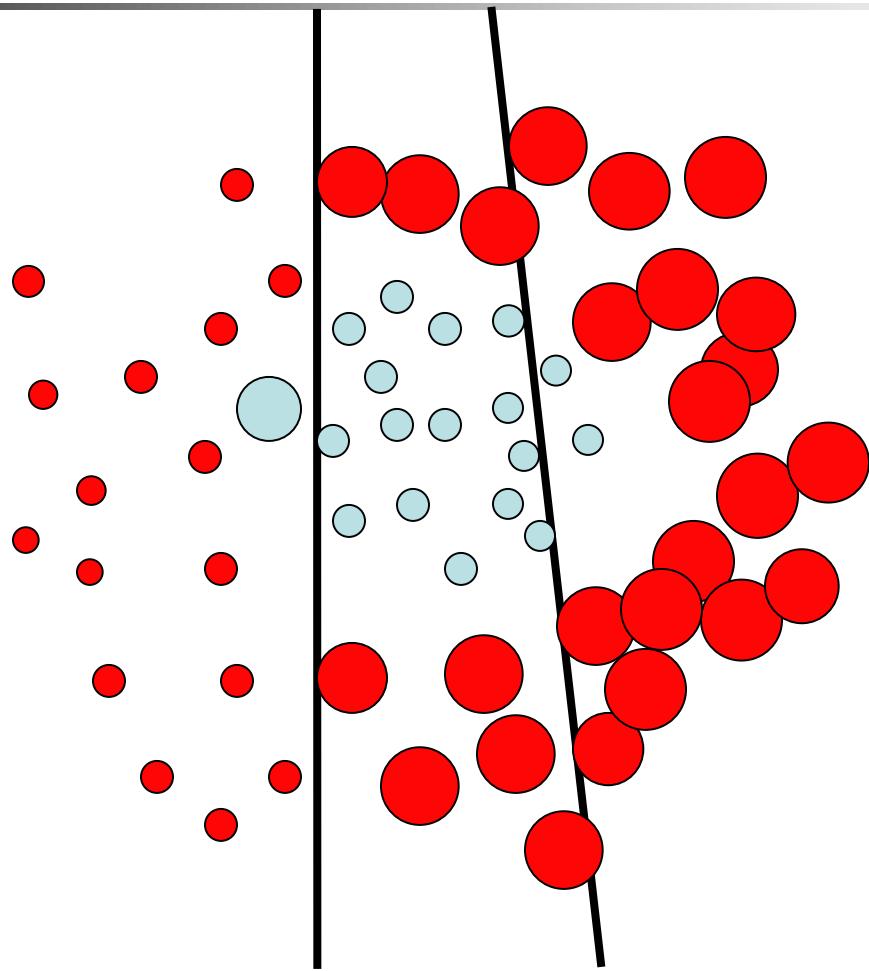


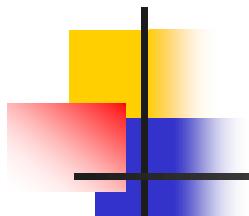
# Modul de funcționare



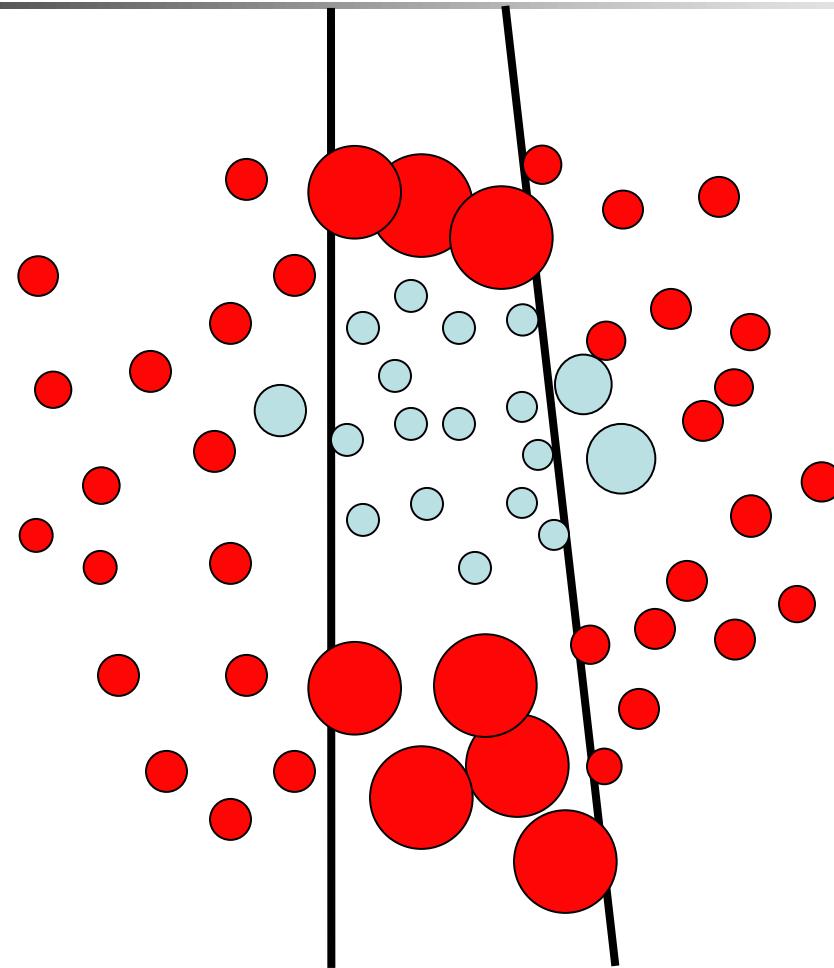


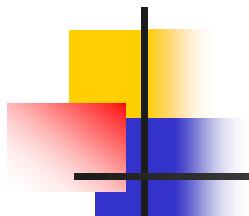
# Modul de funcționare



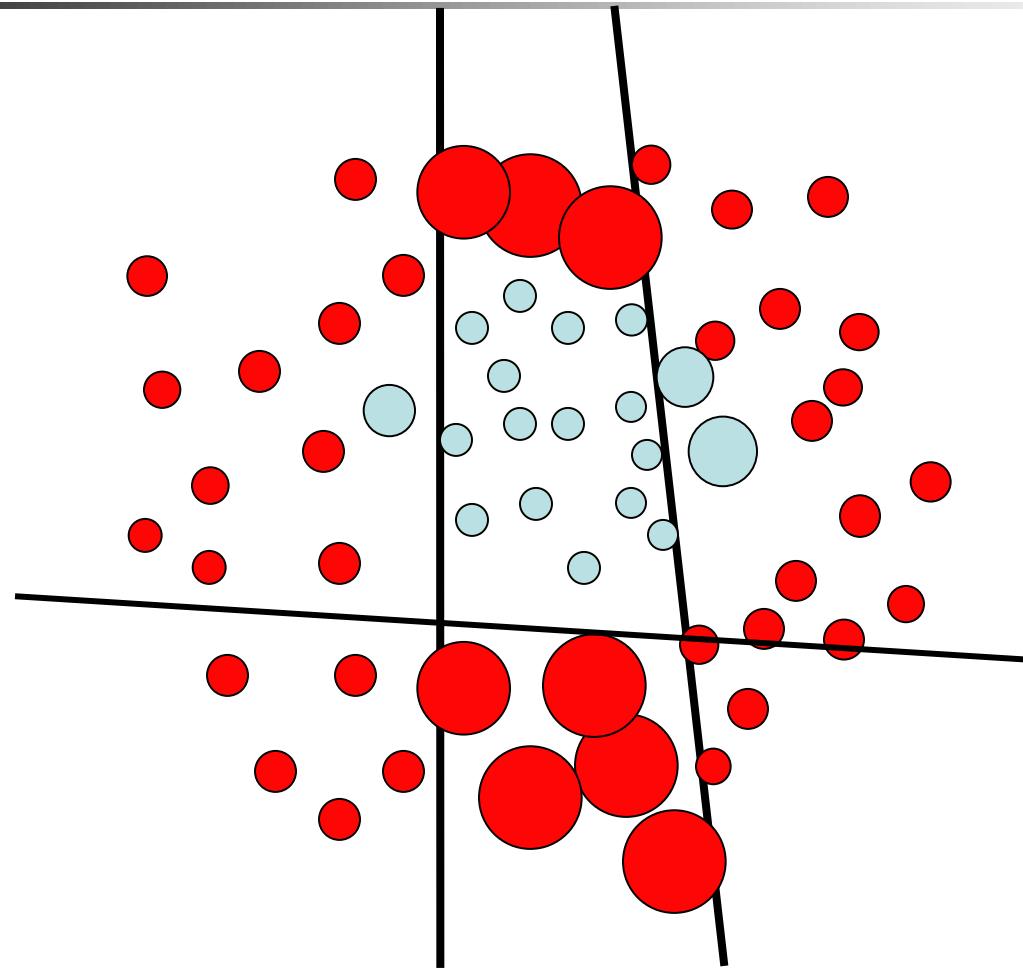


# Modul de funcționare

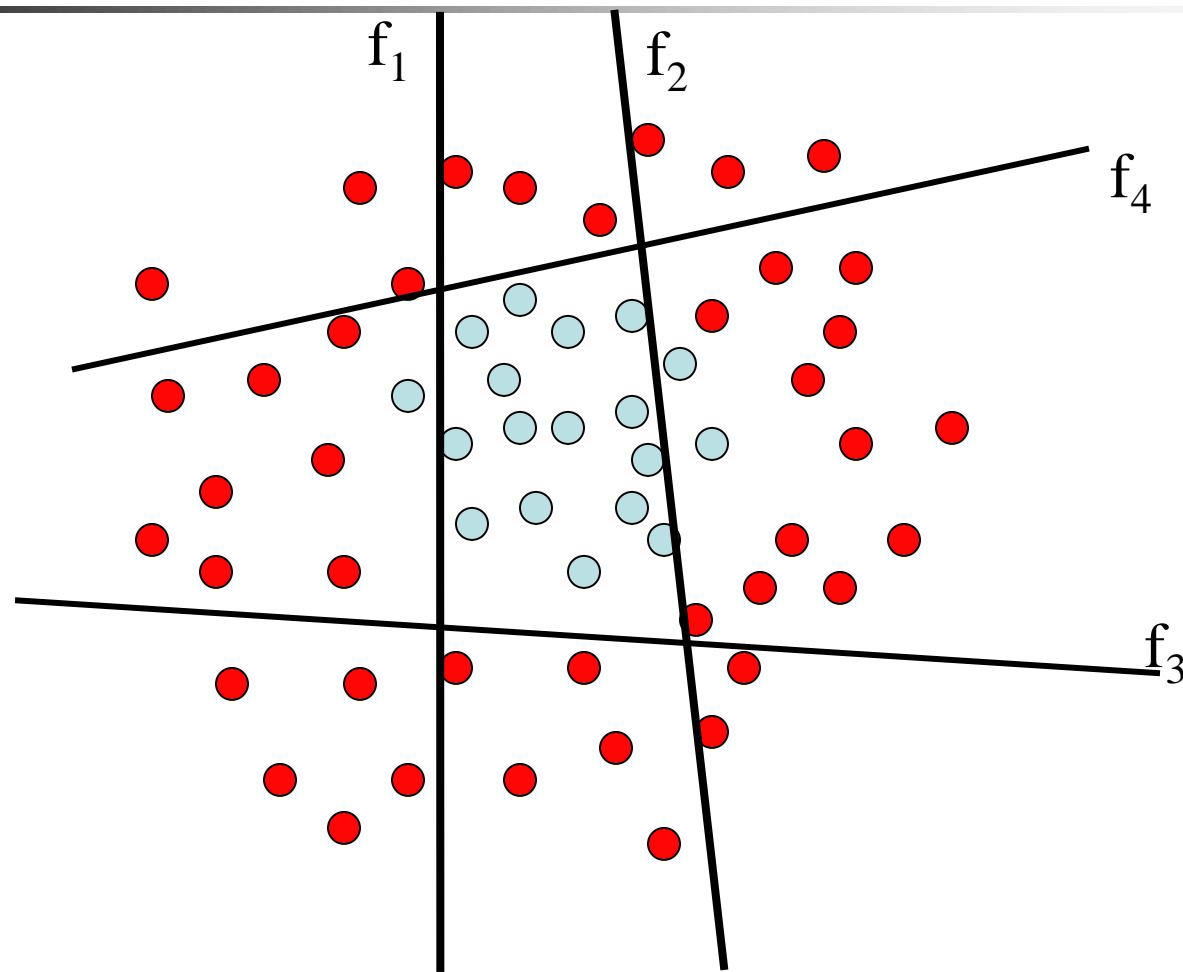


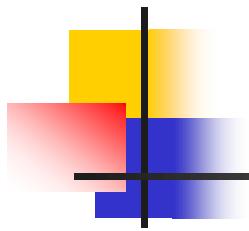


# Modul de funcționare



# Modul de funcționare

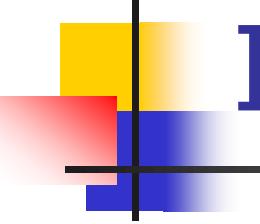




# Exemplu

Nr. instanță	x	y
1	1	1
2	2	-1
3	3	1

$$D_0 = (0.333, 0.333, 0.333)$$



# Iterația 1

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Decision Stump:

$$\begin{cases} x \leq 1.5 \Rightarrow h = 1 \\ x > 1.5 \Rightarrow h = -1 \end{cases}$$

Nr. instanță	x	y	h1
1	1	1	1
2	2	-1	-1
3	3	1	-1

Modelul are 1 eroare. Ar fi fost la fel pentru orice alt prag.

$$\varepsilon_t = 0.333$$

$$\alpha_1 = \frac{1}{2} \ln \frac{1 - 0.333}{0.333} = 0.347$$

Valorile nenormalizate:

$$\begin{aligned} D_1^n(1) &= 0.333 \cdot e^{-0.347 \cdot 1 \cdot 1} = 0.236 \\ D_1^n(2) &= 0.333 \cdot e^{-0.347 \cdot (-1) \cdot (-1)} = 0.236 \\ D_1^n(3) &= 0.333 \cdot e^{-0.347 \cdot 1 \cdot (-1)} = 0.471 \end{aligned}$$

Valorile normalize:

$$\begin{aligned} D_1(1) &= 0.25 \\ D_1(2) &= 0.25 \\ D_1(3) &= 0.5 \end{aligned}$$

# Iterația 2

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Decision Stump:

$$\begin{cases} x \leq 2.5 \Rightarrow h = -1 \\ x > 2.5 \Rightarrow h = 1 \end{cases}$$

Nr. instanță	x	y	D1	h2
1	1	1	0.25	-1
2	2	-1	0.25	-1
3	3	1	0.5	1

Instanța 3 este acum mai importantă.

$$\varepsilon_t = 0.25$$

$$\alpha_2 = \frac{1}{2} \ln \frac{1 - 0.25}{0.25} = 0.549$$

Valorile nenormalizate:

$$D_2^n(1) = 0.25 \cdot e^{-0.549 \cdot 1 \cdot (-1)} = 0.433$$

$$D_2^n(2) = 0.25 \cdot e^{-0.549 \cdot (-1) \cdot (-1)} = 0.144$$

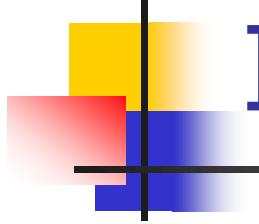
$$D_2^n(3) = 0.5 \cdot e^{-0.549 \cdot 1 \cdot 1} = 0.289$$

Valorile normalize:

$$D_2(1) = 0.5$$

$$D_2(2) = 0.167$$

$$D_2(3) = 0.333$$



# Iterația 3

Decision Stump:

$$\begin{cases} x \leq 1.5 \Rightarrow h = 1 \\ x > 1.5 \Rightarrow h = -1 \end{cases}$$

Nr. instanță	x	y	D2	h3
1	1	1	0.5	1
2	2	-1	0.167	1
3	3	1	0.333	1

$$\varepsilon_t = 0.167$$

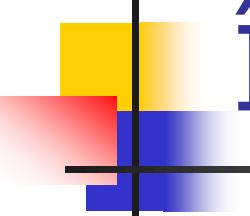
$$\alpha_3 = \frac{1}{2} \ln \frac{1 - 0.167}{0.167} = 0.805$$

Calcularea ieșirii compuse se face după fiecare iterație, iar algoritmul se oprește dacă toate instanțele sunt clasificate corect:

$$H(\mathbf{x}) = \operatorname{sgn} \left( \sum_t \alpha_t h_t(\mathbf{x}) \right)$$

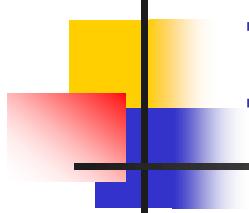
$$\alpha_1 = 0.347, \alpha_2 = 0.549, \alpha_3 = 0.805$$

Nr. instanță	x	y	h1	h2	h3	suma	H
1	1	1	1	-1	1	$0.347 \cdot 1 + 0.549 \cdot (-1) + 0.805 \cdot 1 = 0.603$	1
2	2	-1	-1	-1	1	$0.347 \cdot (-1) + 0.549 \cdot (-1) + 0.805 \cdot 1 = -0.091$	-1
3	3	1	-1	1	1	$0.347 \cdot (-1) + 0.549 \cdot 1 + 0.805 \cdot 1 = 1.007$	1



# Încrederea

- Iterațiile pot continua și după clasificarea corectă, pentru creșterea încrederei
  - Echivalent cu mărirea marginii de separare între clase
  - De exemplu, suma pentru instanța 2 va lua valori tot mai mici, chiar dacă semnul său rămâne -1



# Impossibilitatea învățării

## ■ Problema XOR

Nr. instanță	x1	x2	y
1	0	0	-1
2	0	1	1
3	1	0	1
4	1	1	-1

$$D_0 = (0.25, 0.25, 0.25, 0.25)$$

Decision Stump:

$$\begin{cases} x_1 \leq 0.5 \Rightarrow h = -1 \\ x_1 > 0.5 \Rightarrow h = 1 \end{cases}$$

Nr. instanță	x1	x2	y	h
1	0	0	-1	-1
2	0	1	1	-1
3	1	0	1	1
4	1	1	-1	1

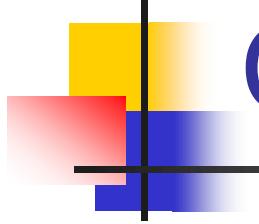
$$\varepsilon_t = 0.5$$

$$\alpha_1 = \frac{1}{2} \ln \frac{1 - 0.5}{0.5} = 0$$

Valorile nenormalizate:

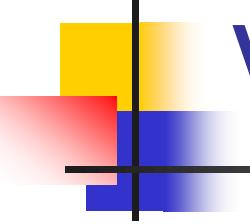
$$D_1^n(1) = 0.25 \cdot e^{-0 \cdot (-1) \cdot (-1)} = 0.25$$

Nu poate învăța funcția XOR pentru că eroarea clasificatorului slab nu este mai mică de 50%. În general, Adaboost poate învăța funcții neliniare.



# Caracteristici

- Avantaje
  - Ușor de implementat
  - Asigură implicit selecția trăsăturilor
  - Generalizează bine
- Dezavantaje
  - Se bazează pe o abordare *greedy*, care poate determina soluții suboptime
  - Zgomotul din date îi poate afecta performanțele



# Variante

- Există numeroase variante de Adaboost:
  - Cu valori discrete
  - Cu clase multiple: Adaboost.M1, Adaboost.M2
  - Cu valori reale: Adaboost.R
  - Cu predicția ratei de încredere
  - Adaboost.MR, Adaboost.MH (cu formule diferite)
  - Totally Corrective Adaboost
  - Cascaded Adaboost
  - etc.

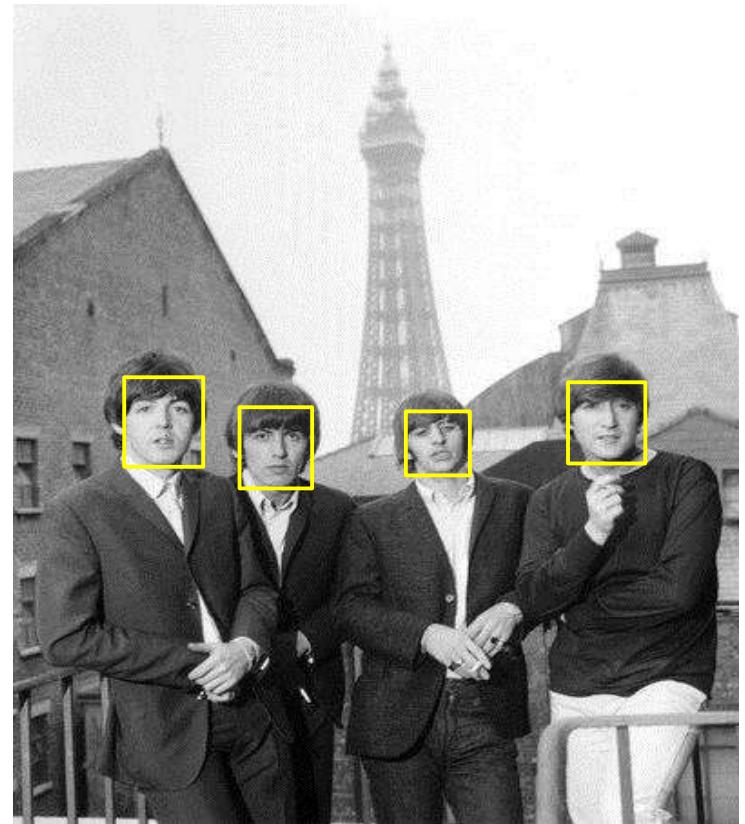
# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
  - 2.1. Detectorul facial Viola-Jones
3. Agregarea în stivă
4. Selectia trăsăturilor



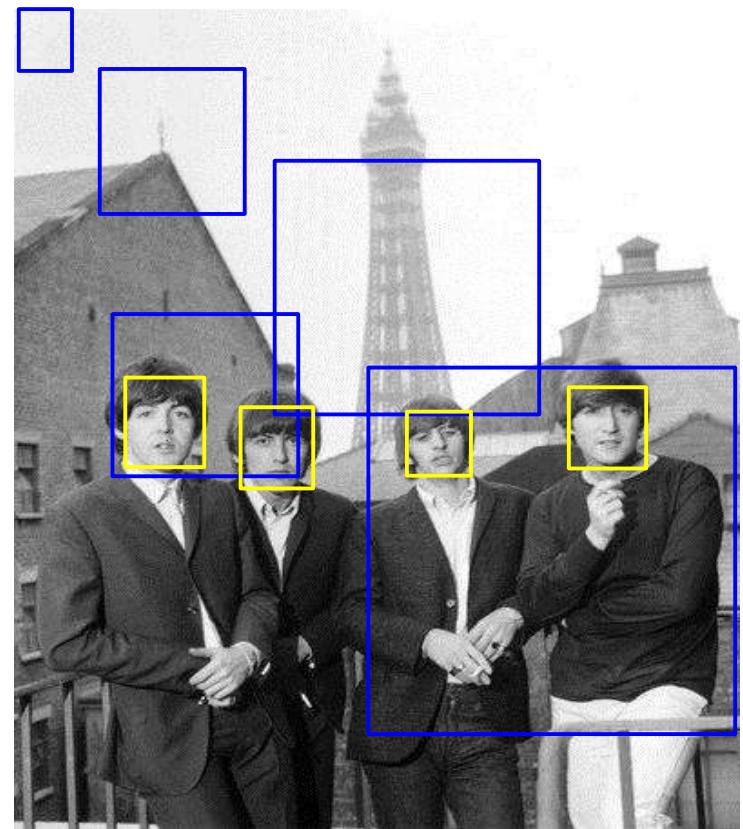
# Adaboost pentru detecția fețelor

- A fost aplicația care i-a adus popularitate algoritmului Adaboost (Viola & Jones, 2001)
- Metoda este implementată în OpenCV: cvHaarDetectObjects



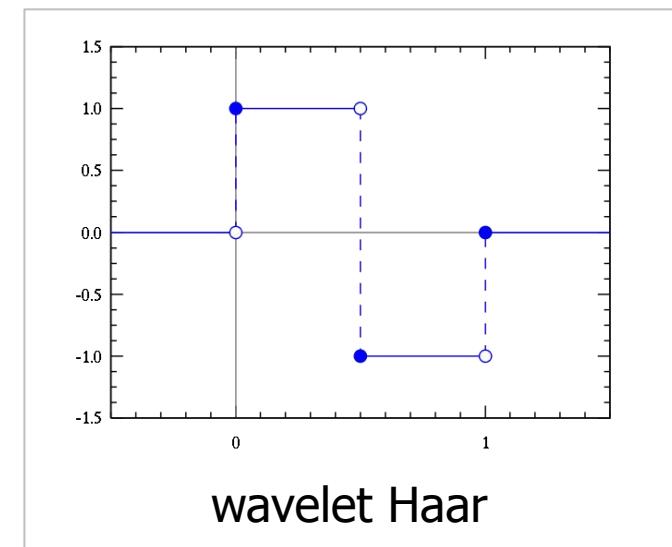
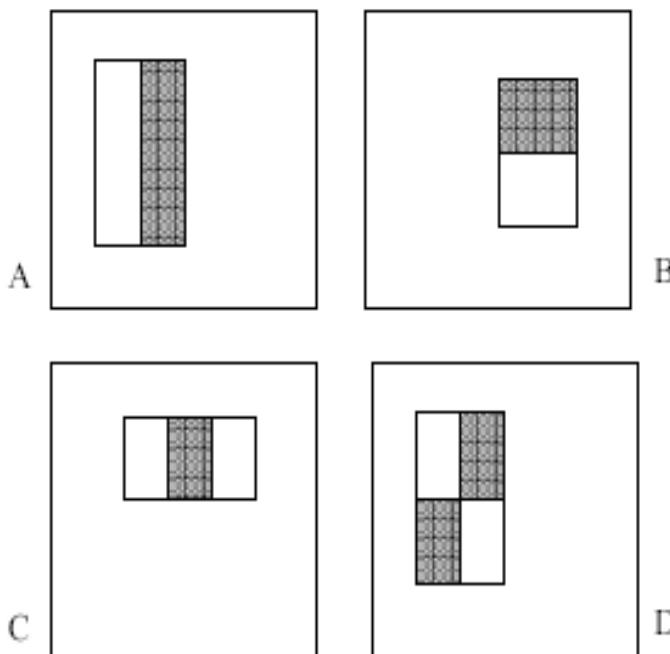
# Adaboost pentru detecția fetelor

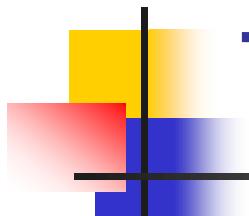
- Se testează un mare număr de dreptunghiuri, cu poziții diferite și de diferite dimensiuni
- Un dreptunghi se clasifică drept „față” sau „fundal”
- Dreptunghiurile de fundal sunt de mii de ori mai numeroase



# Trăsături ale imaginilor

- 4 tipuri de filtre dreptunghiulare, asemănătoare *wavelet-urilor Haar*





# Trăsături ale imaginilor



trăsătură asemănătoare  
regiunii nasului

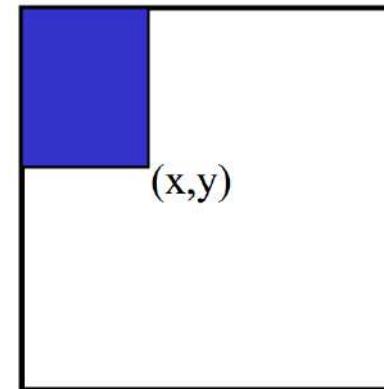


trăsătură asemănătoare  
regiunii ochilor

Fie  $g(\mathbf{x}) = \sum (\text{pixeli în zona albă}) - \sum (\text{pixeli în zona neagră})$

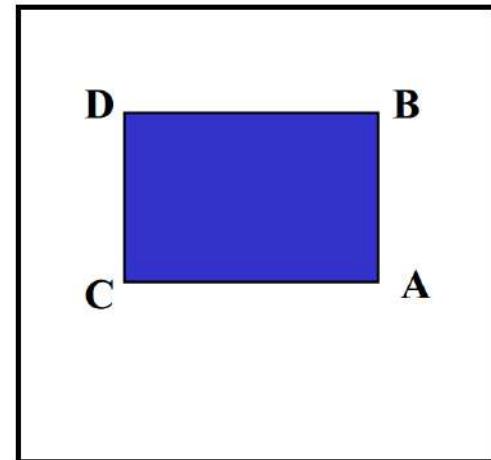
# Imaginea integrală

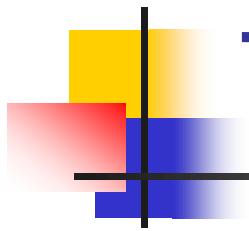
- Pentru fiecare pixel  $(x, y)$  se calculează suma valorilor pixelilor din regiunea aflată în stânga-sus față de el



- Pentru un dreptunghi definit de vîrfurile  $A, B, C, D$ , suma valorilor din interior este:

$$S = A - B - C + D$$





# Trăsături ale imaginilor

- Trăsături:

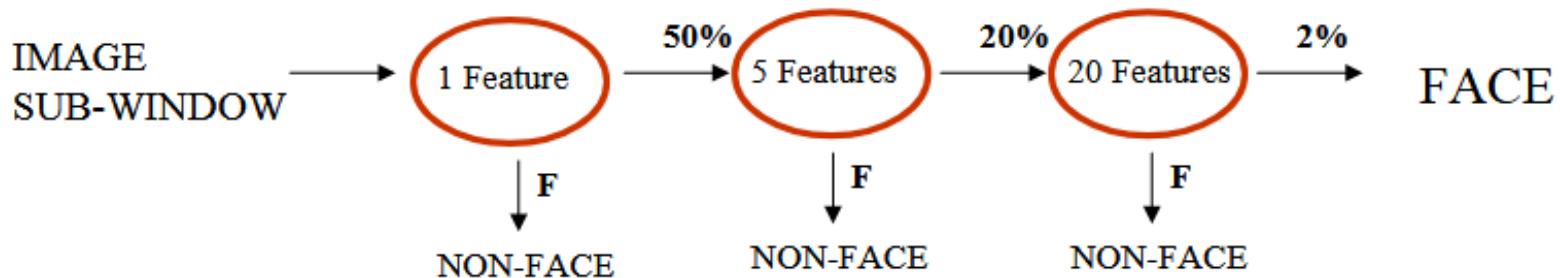
$$h_i(\mathbf{x}) = \begin{cases} 1 & \text{dacă } g_i(\mathbf{x}) > \theta_i \\ -1 & \text{altfel} \end{cases}$$

- Pragurile  $\theta$  sunt alese astfel încât să minimizeze numărul de erori la antrenare pentru fiecare trăsătură
- Funcția de detectie este:

$$H(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \dots$$

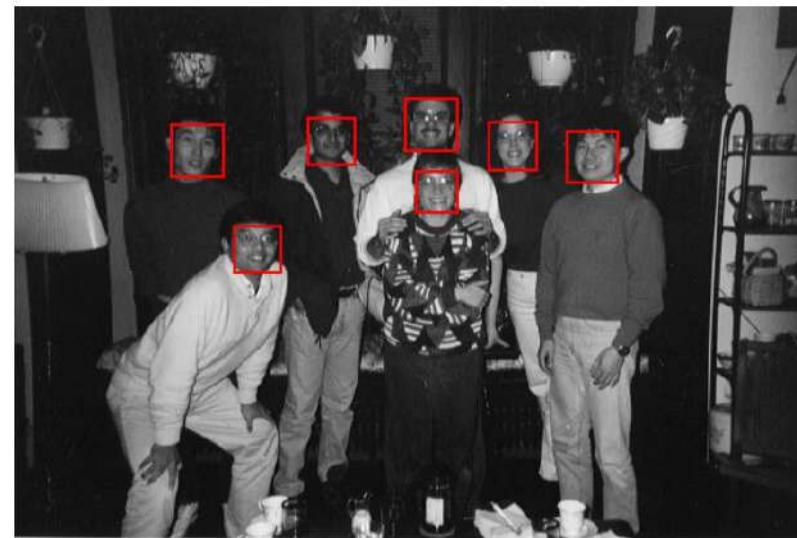
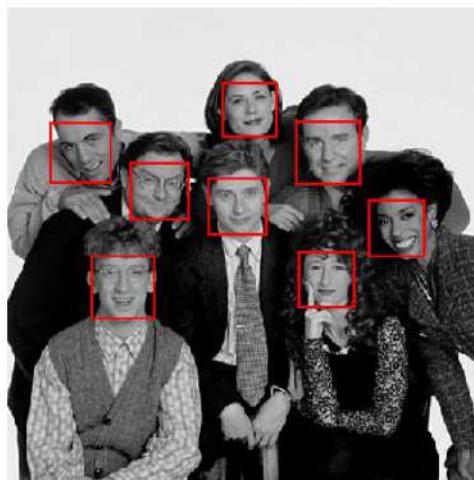
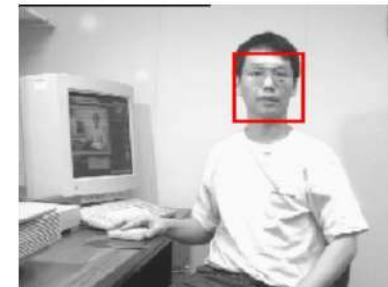
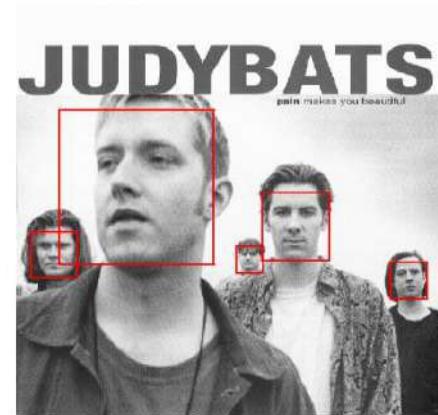
- Ponderile  $\alpha$  sunt calculate de Adaboost
- Există aproximativ 60 000 de trăsături
- Se pot folosi numai trăsăturile cu ponderile cele mai mari

# Detectie în cascadă



- Un clasificator cu 1 trăsătură are o rată de detectie de 100% și 50% rată de fals pozitive
- Un clasificator cu 5 trăsături are o rată de detectie de 100% și 40% rată de fals pozitive (cumulat, 20%)
- Un clasificator cu 20 trăsături are o rată de detectie de 100% și 10% rată de fals pozitive (cumulat, 2%)

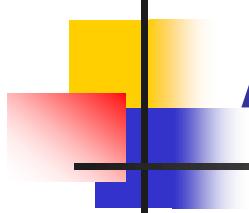
# Rezultate



# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
3. Agregarea în stivă
4. Selectia trăsăturilor



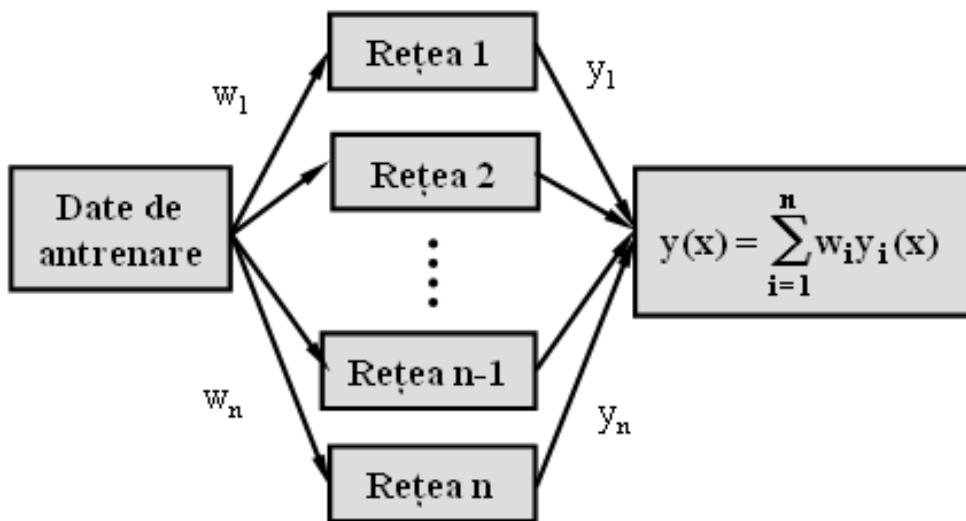


# Agregarea în stivă

- engl. “stacking”
- Presupune ponderarea ieșirilor mai multor algoritmi numerici
- Se folosește de obicei pentru probleme de regresie, în timp ce pentru clasificare se recurge la votare

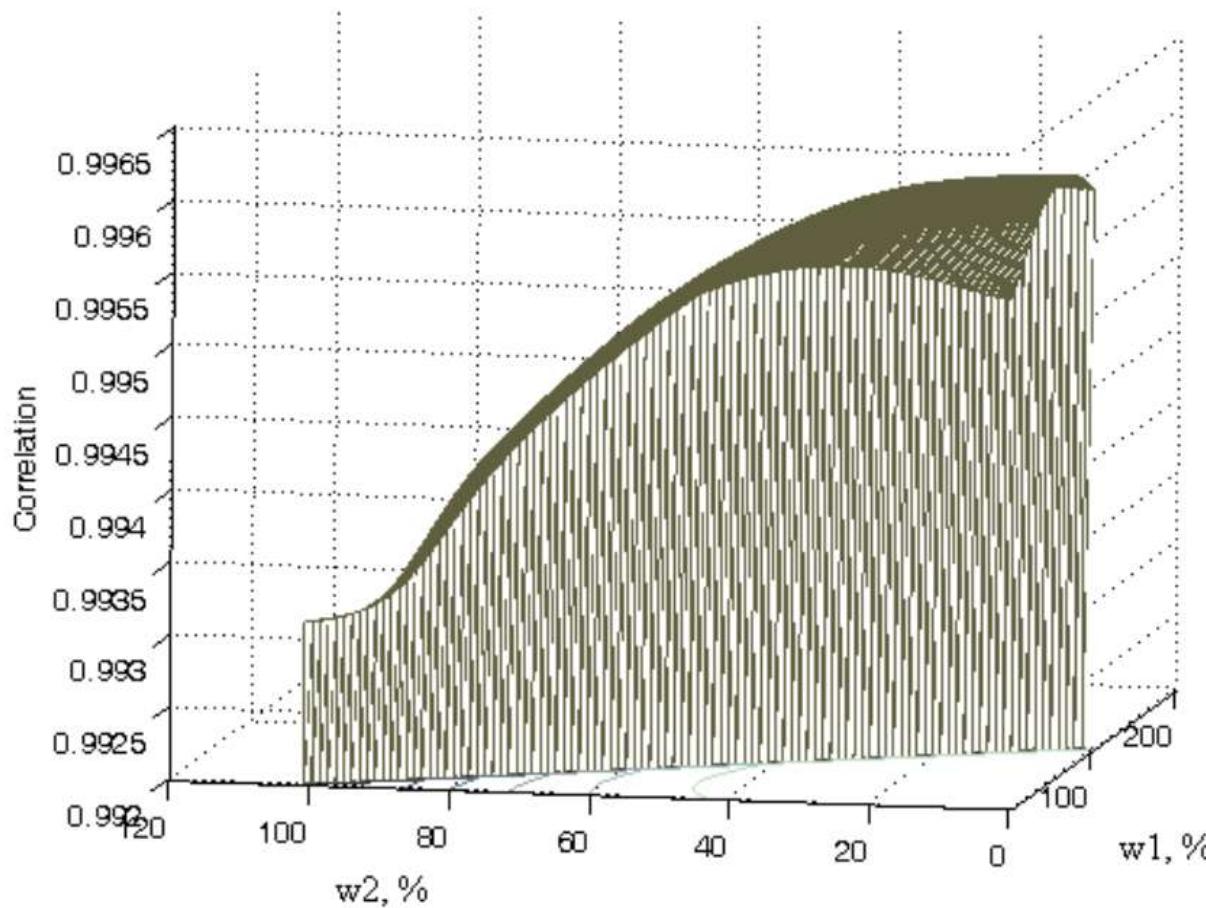
# Rețele stivă

- Prin sumarea ponderată a ieșirilor mai multor rețele individuale, pot crește performanțele de generalizare ale rețelei stivă aggregate



- Rețele omogene
  - Funcție de activare sigmoidă bipolară
- Rețele neomogene
  - Ieșiri: sigmoidă bipolară, exponentială

# Exemplu: performanțe pe mulțimea de test

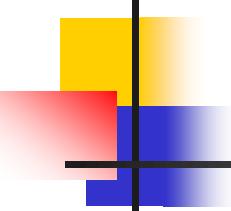


Descompunerea  
fotocatalitică  
eterogenă a  
triclopirului

# Clasificarea bazată pe ansambluri. Selectia trăsăturilor

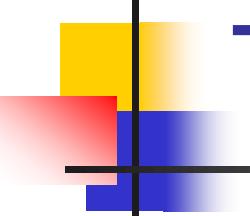
1. Bagging. Algoritmul Random Forest
2. Boosting. Algoritmul Adaboost
3. Agregarea în stivă
4. Selectia trăsăturilor





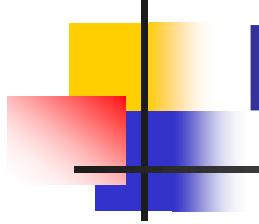
# Selectia trăsăturilor

- Atributele multimii de antrenare pot avea grade diferite de relevanță
  - De exemplu, sexul unei persoane este mai puțin relevant pentru predicția diabetului decât vârsta
- Unele atrbute pot fi complet irelevante
  - Zgomot, afectează clasificarea
- Selectia trăsăturilor încearcă să identifice atrbutele care pot contribui cel mai mult la predicția clasei



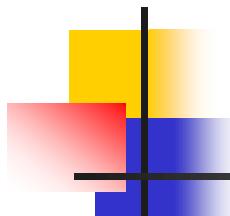
# Tipuri de metode

- Metode de filtrare (*filter*)
  - Folosesc un criteriu matematic pentru a evalua calitatea fiecărui atribut sau grup de attribute
- Metode de acoperire (*wrapper*)
  - Folosesc un algoritm de clasificare pentru a determina importanța atributelor
- Metode încorporate (*embedded*)
  - Învață calitatea atributelor în timp ce este creat modelul
  - În general, aici intră metodele de regularizare, care vor fi descrise în cursul 4



# Metode de filtrare

- Metode statistice:
  - Indexul Gini
  - Entropia
  - Scorul Fisher
- Algoritmul Relief



# Indexul Gini

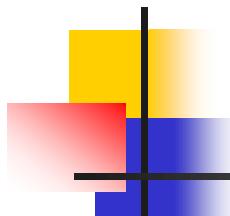
- Importanța unei valori de atribut

$$G(v_i) = 1 - \sum_{j=1}^k p_j^2$$

- Importanța unui atribut

$$G = \sum_{i=1}^r n_i G(v_i)/n$$

- $v_i$  = valorile posibile ale atributului considerat
- $p_j$  = fractiunea de instanțe care conțin valoarea  $v_i$  și care aparțin clasei  $j$
- $n$  = numărul total de instanțe
- $n_i$  = numărul de instanțe care conțin valoarea  $v_i$



# Entropia

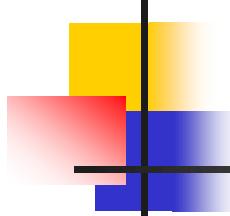
- Importanța unei valori de atribut

$$E(v_i) = - \sum_{j=1}^k p_j \log_2(p_j)$$

- Importanța unui atribut

$$E = \sum_{i=1}^r n_i E(v_i)/n$$

- $v_i$  = valorile posibile ale atributului considerat
- $p_j$  = fractiunea de instanțe care conțin valoarea  $v_i$  și care aparțin clasei  $j$
- $n$  = numărul total de instanțe
- $n_i$  = numărul de instanțe care conțin valoarea  $v_i$



# Scorul Fisher

- Indexul Gini și entropia se aplică pentru attribute simbolice
- Scorul Fisher se aplică pentru attribute numerice
- Măsoară raportul dintre separarea medie inter-clasă și separarea medie intra-clasă

$$F = \frac{\sum_{j=1}^k p_j (\mu_j - \mu)^2}{\sum_{j=1}^k p_j \sigma_j^2}$$

- $\mu_j, \sigma_j, p_j$  = media, deviația standard și fractiunea de instanțe care aparțin clasei  $j$
- $\mu$  = media globală a tuturor instanțelor pentru atributul considerat

$$F = \frac{\sum_{j=1}^k p_j (\mu_j - \mu)^2}{\sum_{j=1}^k p_j \sigma_j^2}$$

# Exemplu: Income

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Instante în clasa No: 125, 100, 70, 120, 60, 220, 75

Instante în clasa Yes: 95, 85, 90

$$\mu_{No} = 110, \sigma_{No} = 50.5, p_{No} = 0.7$$

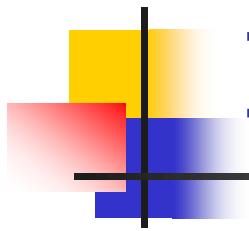
$$\mu_{Yes} = 90, \sigma_{Yes} = 4.1, p_{Yes} = 0.3$$

$$\mu = 104$$

$$F_{Income} = \frac{p_{No} (\mu_{No} - \mu)^2 + p_{Yes} (\mu_{Yes} - \mu)^2}{p_{No} \cdot \sigma_{No}^2 + p_{Yes} \cdot \sigma_{Yes}^2}$$

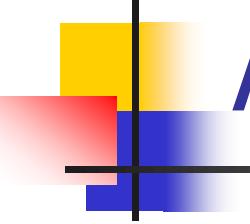
$$F_{Income} = \frac{0.7 \cdot (110 - 104)^2 + 0.3 \cdot (90 - 104)^2}{0.7 \cdot 50.5^2 + 0.3 \cdot 4.1^2} = 0.047$$

Domeniul valorilor pentru Yes (85-95) este o submulțime a celui pentru No (60-220)  $\Rightarrow$  atributul nu este bun pentru a separa valorile clasei



# Importanța atributelor

- Un atribut este cu atât mai important cu cât **indexul Gini** sau **entropia** au valori mai **mici**
  - Valoarea 0 înseamnă că atributul poate rezolva singur problema de clasificare
- Un atribut este cu atât mai important cu cât **scorul Fisher** are valori mai **mari**



# Algoritmul Relief

- Se inițializează ponderile atributelor cu 0
- Pentru fiecare instanță  $x_q$  din mulțimea de antrenare sau pentru o instanță  $x_q$  selectată aleatoriu
  - Se găsește cea mai apropiată instanță din aceeași clasă (*hit*) și cea mai apropiată instanță dintr-o clasă diferită (*miss*)
  - Se actualizează ponderile fiecărui atribut pe baza acestor distanțe
- Un atribut este mai important dacă clasele sunt mai grupate pe dimensiunea lui ( $d_{hit}$  mică,  $d_{miss}$  mare)

# Pseudocod Relief

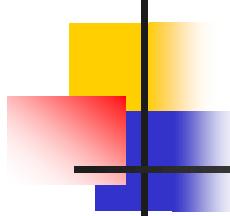
## *Algorithm* Relief

*Input:* for each training instance a vector of attribute values and the class value

*Output:* the vector W of estimations of the qualities of attributes

1. set all weights  $W[A] := 0.0$ ;
2. **for**  $i := 1$  **to**  $m$  **do begin**
3.     randomly select an instance  $R_i$ ;
4.     find nearest hit  $H$  and nearest miss  $M$ ;
5.     **for**  $A := 1$  **to**  $a$  **do**
6.          $W[A] := W[A] - \text{diff}(A, R_i, H) / m + \text{diff}(A, R_i, M) / m$ ;
7.     **end;**

un atribut este mai important dacă pe dimensiunea sa instanțele sunt mai depărtate de instanțele din alte clase și mai apropiate de cele din aceeași clasă



# Diferență

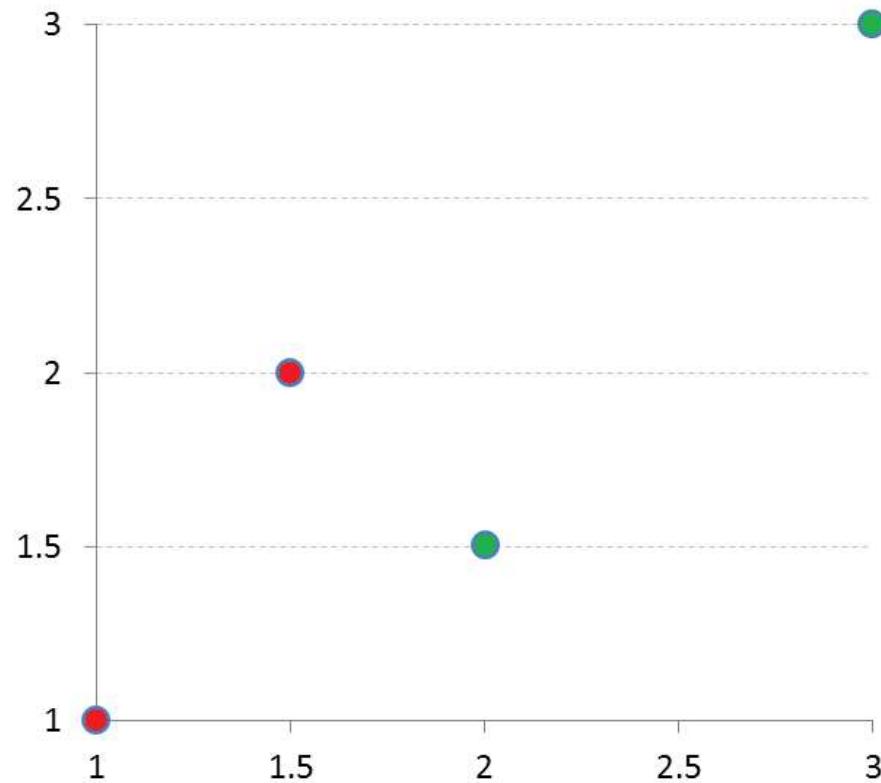
- Pentru atribute nominale:

$$\text{diff}(A, I_1, I_2) = \begin{cases} 0 & ; \text{value}(A, I_1) = \text{value}(A, I_2) \\ 1 & ; \text{otherwise} \end{cases}$$

- Pentru atribute numerice:

$$\text{diff}(A, I_1, I_2) = \frac{|\text{value}(A, I_1) - \text{value}(A, I_2)|}{\max(A) - \min(A)}$$

# Exemplu

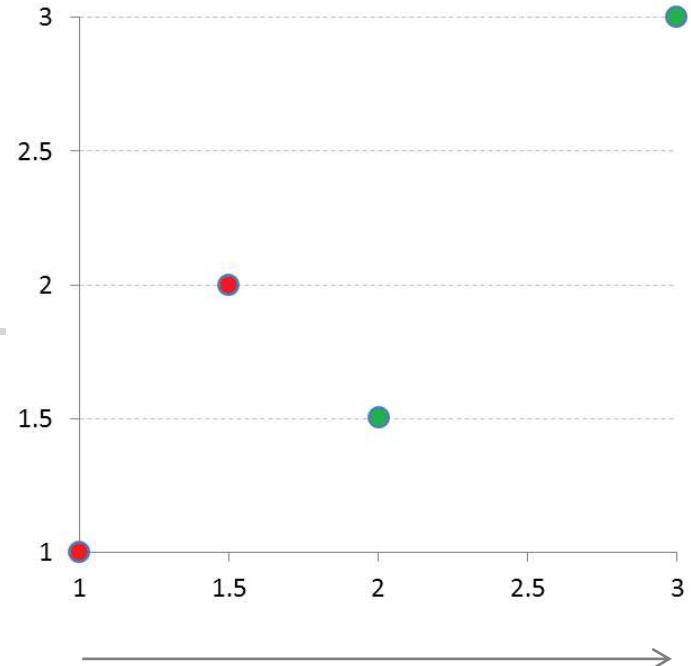


1	1	N
1.5	2	N
2	1.5	Y
3	3	Y

Puncte bidimensionale,  
clasă binară (Y / N)

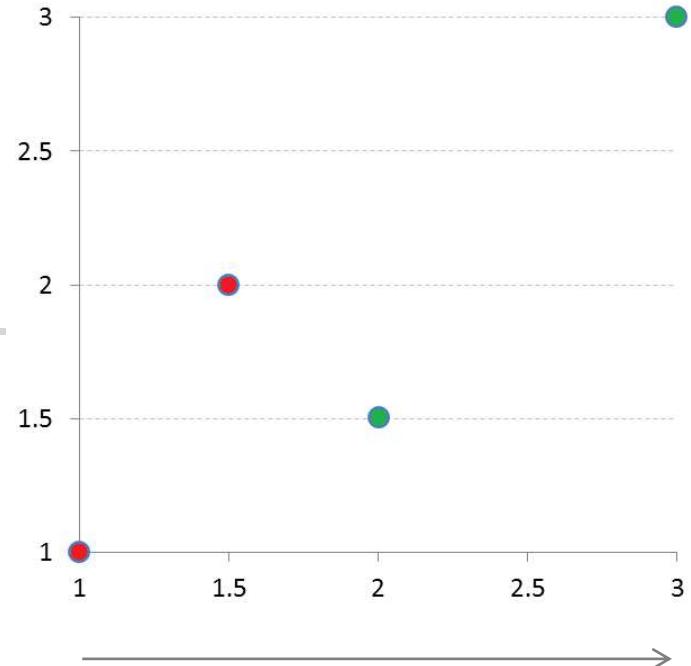
# Exemplu

- Atributul 1
  - $x_{11} = 1, C = N$
  - $x_{21} = 1.5, C = N$
  - $x_{31} = 2, C = Y$
  - $x_{41} = 3, C = Y$
- În acest exemplu vom considera toate instanțele, dar în algoritmul standard instanțele sunt selectate aleatoriu
- Punct selectat:  $x_{11}$ 
  - *hit*:  $x_{21}$ , *miss*:  $x_{31}$
  - $\text{diff}(hit) = 1.5 - 1 = 0.5$
  - $\text{diff}(miss) = 2 - 1 = 1$
  - Pentru normalizare, diferența se împarte la:  $3 - 1 = 2$
  - $\Rightarrow \Delta w_1 = (1 - 0.5) / 2 = 0.25 \Rightarrow w_1 = 0.25$



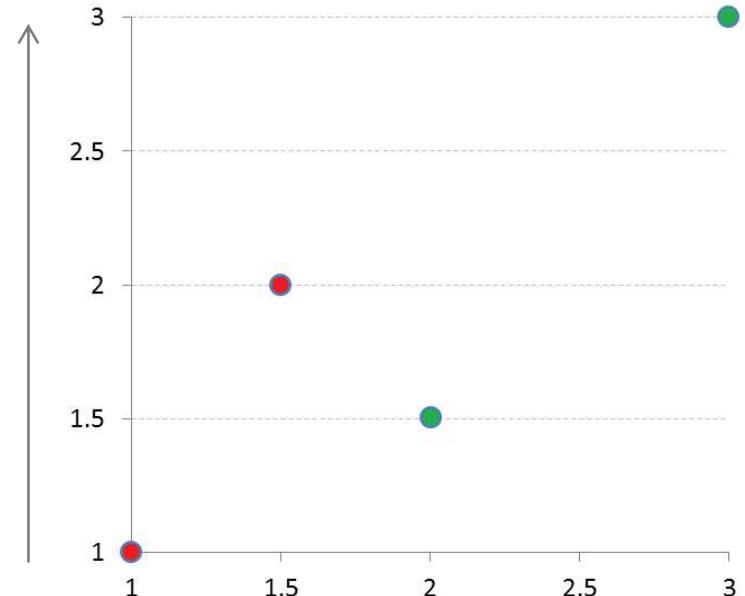
# Exemplu

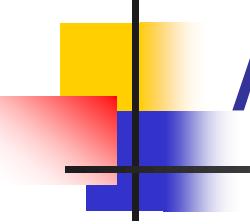
- Atributul 1
  - $x_{11} = 1, C = N$
  - $x_{21} = 1.5, C = N$
  - $x_{31} = 2, C = Y$
  - $x_{41} = 3, C = Y$
- Iterațiile, cu normalizare
  - $x_{11}$ : hit  $x_{21}$ , miss  $x_{31}$ ,  $\Delta w_1 = (1 - 0.5) / 2 = 0.25$
  - $x_{21}$ : hit  $x_{11}$ , miss  $x_{31}$ ,  $\Delta w_1 = (0.5 - 0.5) / 2 = 0$
  - $x_{31}$ : hit  $x_{41}$ , miss  $x_{21}$ ,  $\Delta w_1 = (0.5 - 1) / 2 = -0.25$
  - $x_{41}$ : hit  $x_{31}$ , miss  $x_{21}$ ,  $\Delta w_1 = (1.5 - 1) / 2 = 0.25$
- În această situație,  $w_1 = (0.25 + 0 - 0.25 + 0.25) / 4 = 0.0625$



# Exemplu

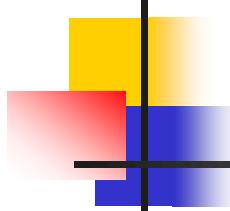
- Atributul 2
  - $x_{12} = 1, C = N$
  - $x_{22} = 1.5, C = Y$
  - $x_{32} = 2, C = N$
  - $x_{42} = 3, C = Y$
- Iterațiile:
  - $x_{12}$ : hit  $x_{32}$ , miss  $x_{22}$ ,  $\Delta w_2 = (0.5 - 1) / 2 = -0.25$
  - $x_{22}$ : hit  $x_{42}$ , miss  $x_{12}$ ,  $\Delta w_2 = (0.5 - 1.5) / 2 = -0.5$
  - $x_{32}$ : hit  $x_{12}$ , miss  $x_{22}$ ,  $\Delta w_2 = (0.5 - 1) / 2 = -0.25$
  - $x_{42}$ : hit  $x_{22}$ , miss  $x_{32}$ ,  $\Delta w_2 = (1 - 1.5) / 2 = -0.25$
- În această situație,  $w_2 = -1.25 / 4 = -0.3125$
- Atributul 1 este mai important deoarece pe axa lui instanțele din cele două clase sunt separabile





# Algoritmul ReliefF

- Extensie a algoritmului Relief care se poate aplica pentru probleme cu mai multe clase, date incomplete (instante cu valori lipsă, tratate probabilistic) și date afectate de zgomot
- În loc de cele mai apropiate instante *hit* și *miss*, se aleg  $k$  instante din fiecare clasă
- Formulele de actualizare a ponderilor urmează același principiu ca la Relief, dar sunt ușor diferite

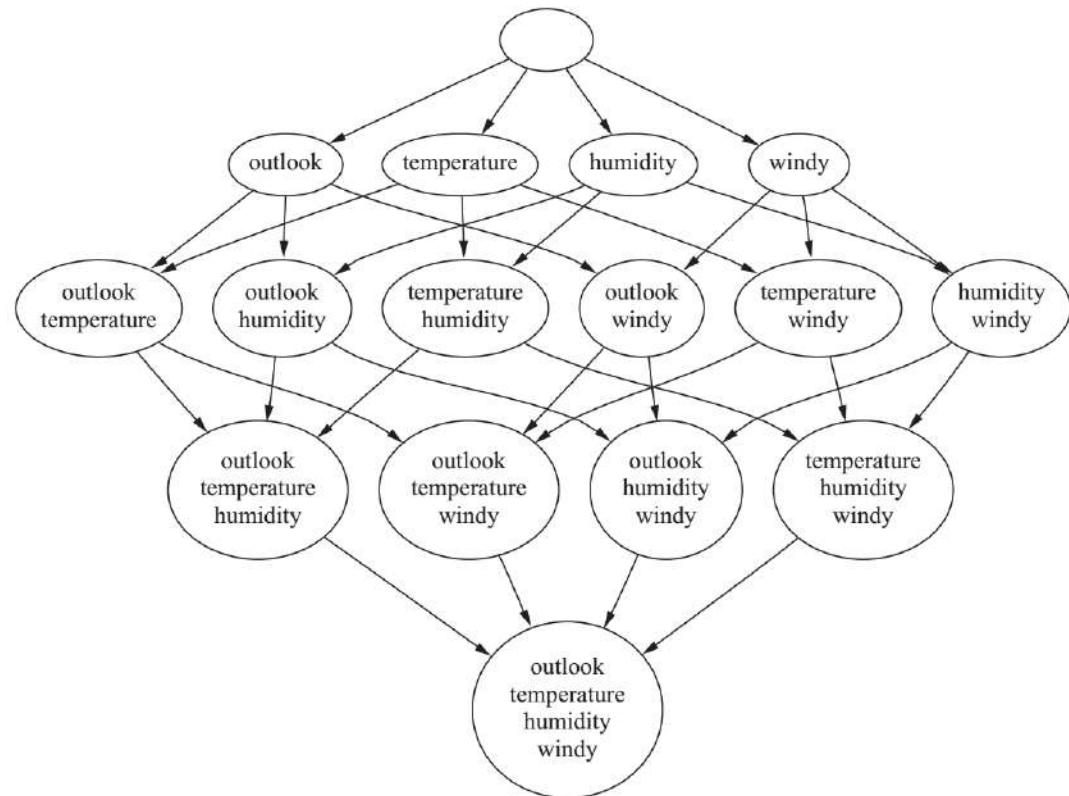


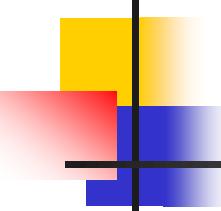
# Căutarea în spațiul atributelor

- Este o metodă de acoperire (*wrapper*)
- Se stabilește algoritmul de clasificare folosit
- Se caută în mod *greedy* în spațiul atributelor
- **Selectie înainte (prin progresie)**: de la niciun atribut, adăugând câte un atribut o dată
- **Eliminare înapoi (prin regresie)**: de la toate atributele, eliminând câte un atribut o dată

# Căutarea în spațiul atributelor

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no





# Căutarea în spațiul atributelor

- La fiecare pas, este evaluat efectul adăugării sau eliminării unui atribut prin validare încrucișată, cu algoritmul de clasificare considerat
  - Validarea încrucișată a fost descrisă în cursul 12 de Inteligență artificială și va fi reamintită în cursul 4
- Când niciun atribut nu mai produce o îmbunătățire a rezultatelor, căutarea se oprește
- Pentru a favoriza multimile mai mici de attribute, la selecția înainte se poate impune condiția ca procentul de erori să scadă cu o anumită valoare minimă (similar pentru eliminarea înapoi)

# Weka



- Colecție open-source de algoritmi de învățare automată  
<http://www.cs.waikato.ac.nz/ml/weka/>

The screenshot shows the Weka interface with two windows:

- Lister Window:** Shows the contents of the "weather.arff" file. The file contains the following attributes and data:

```
@relation weather
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85, FALSE,no
sunny,80,90, TRUE,no
overcast,83,86, FALSE,yes
rainy,78,96, FALSE,yes
rainy,68,80, FALSE,yes
rainy,65,70, TRUE,no
overcast,64,65, TRUE,yes
sunny,72,95, FALSE,no
sunny,69,70, FALSE,yes
rainy,75,80, FALSE,yes
sunny,75,70, TRUE,yes
overcast,72,90, TRUE,yes
overcast,81,75, FALSE,yes
rainy,71,91, TRUE,no
```

- Weka Explorer Window:** Shows the results of an ID3 classification run on the "weather.arff" dataset.

  - Classifier Output:** Displays the generated decision tree:

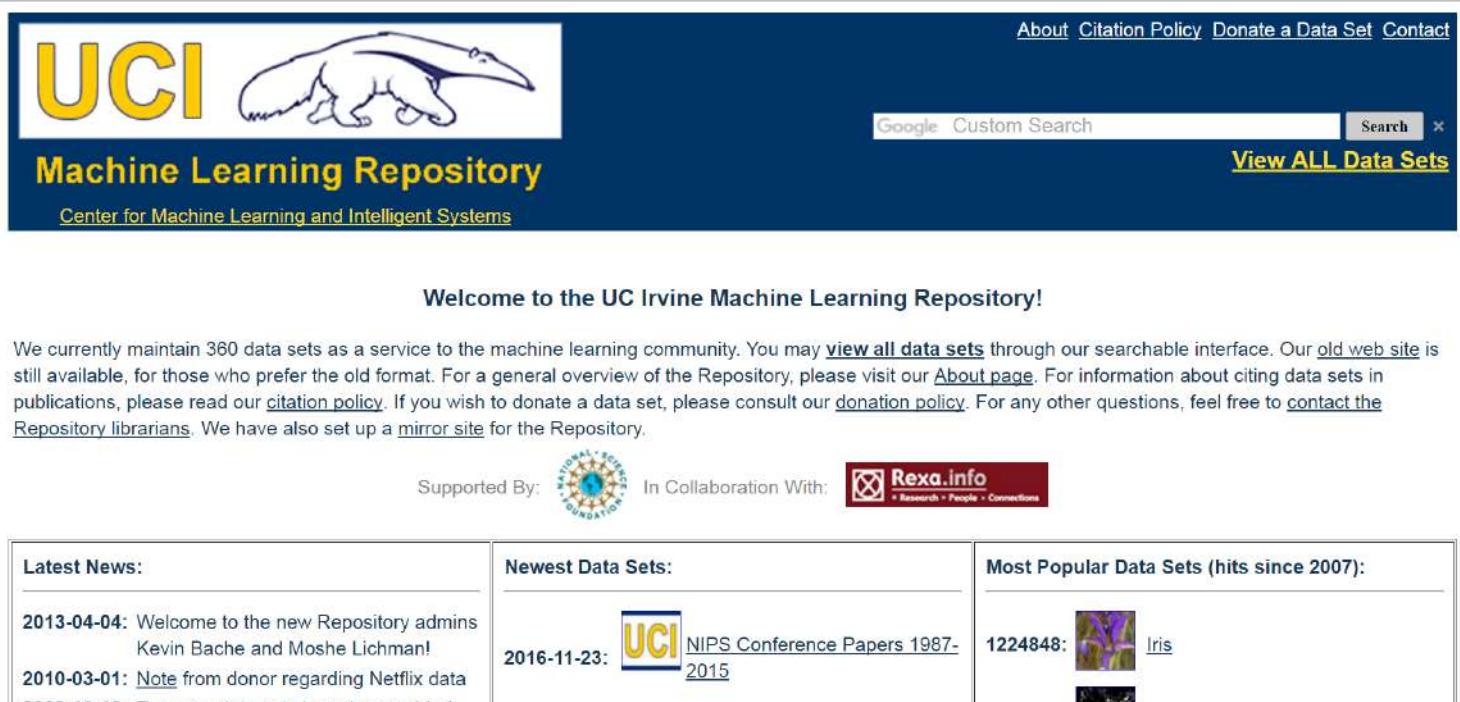
```
outlook = sunny
| humidity = high: no
| humidity = normal: yes
outlook = overcast: yes
outlook = rainy
| windy = TRUE: no
| windy = FALSE: yes
```

  - Result List:** Shows the result of the ID3 classifier.
  - Evaluation Metrics:** Summary statistics for the model's performance:

Correctly Classified Instances	14	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0		
Root mean squared error	0		
Relative absolute error	0		
Root relative squared error	0		%
Total Number of Instances	14		

# UCI Repository

- University of California, Irvine, Machine Learning Repository,  
<http://archive.ics.uci.edu/ml/>
- 585 de multimi de date



The screenshot shows the homepage of the UCI Machine Learning Repository. At the top left is the UCI logo with a stylized antechinus illustration. To its right are links for About, Citation Policy, Donate a Data Set, and Contact. Below the logo is a search bar with a Google Custom Search button and a 'Search' button. A yellow link 'View ALL Data Sets' is also present. The main title 'Machine Learning Repository' is in large yellow letters, with 'Center for Machine Learning and Intelligent Systems' in smaller text below it. A 'Welcome to the UC Irvine Machine Learning Repository!' message is centered on the page. Below this, a paragraph explains the repository's purpose and links to various policies. At the bottom, there are sections for 'Latest News', 'Newest Data Sets', and 'Most Popular Data Sets (hits since 2007)'. The 'Latest News' section lists recent announcements. The 'Newest Data Sets' section shows the most recent datasets added, including 'NIPS Conference Papers 1987-2015'. The 'Most Popular Data Sets' section shows the Iris dataset as the most popular, with a small image of a flower.

Latest News:

2013-04-04: Welcome to the new Repository admins Kevin Bache and Moshe Lichman!

2010-03-01: Note from donor regarding Netflix data

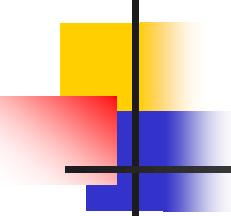
2009-10-16: Two new data sets have been added

Newest Data Sets:

2016-11-23:  NIPS Conference Papers 1987-2015

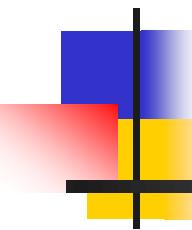
Most Popular Data Sets (hits since 2007):

1224848:  Iris



# Concluzii

- *Bagging-ul* reprezintă generarea unor multimi de date oarecum diferite prin eșantionare uniformă cu înlocuire dintr-o multime de date inițială. Pe aceste multimi se pot antrena clasificatori diferenți, ale căror ieșiri sunt aggregate
- *Boosting-ul* reprezintă transformarea unui clasificator slab într-unul puternic
- *Agregarea în stivă* reprezintă ponderarea ieșirilor mai multor algoritmi numerici
- *Selectia trăsăturilor* identifică atrbutele care pot contribui cel mai mult la predicția clasei
- *Weka* este o colecție foarte populară de algoritmi de învățare automată



# Învățare automată

## 4. Regresie liniară, logistică, softmax

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)

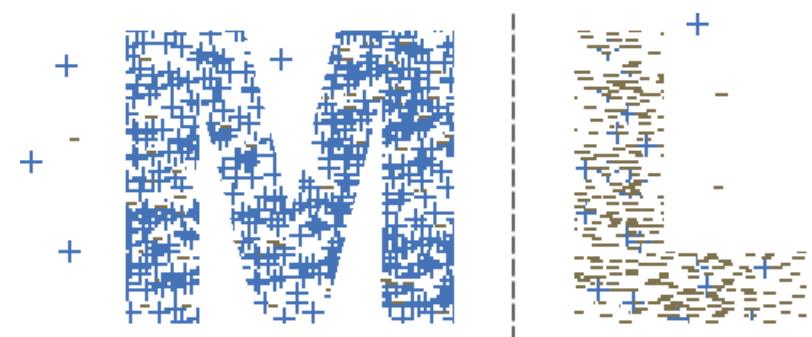
# Regresie liniară, logistică, softmax

1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
4. Regresia softmax
5. Metrici de eroare



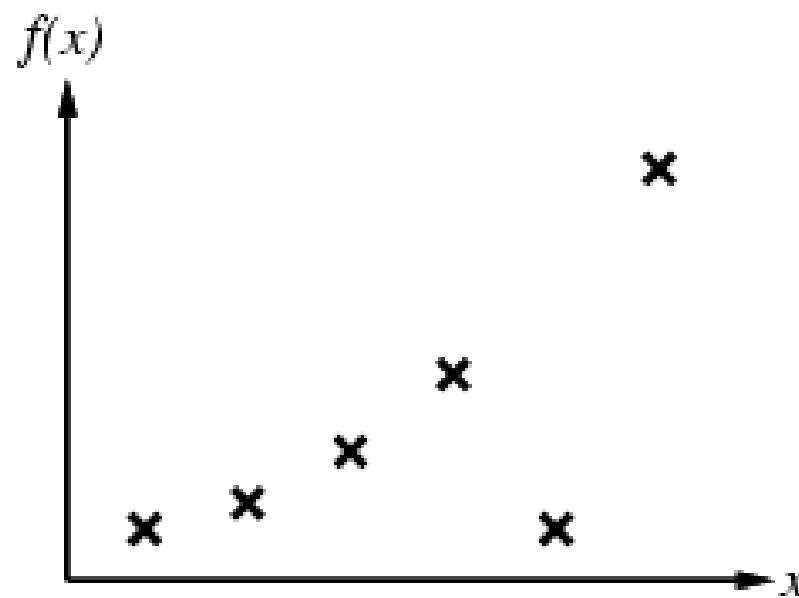
# Regresie liniară, logistică, softmax

1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
4. Regresia softmax
5. Metrici de eroare

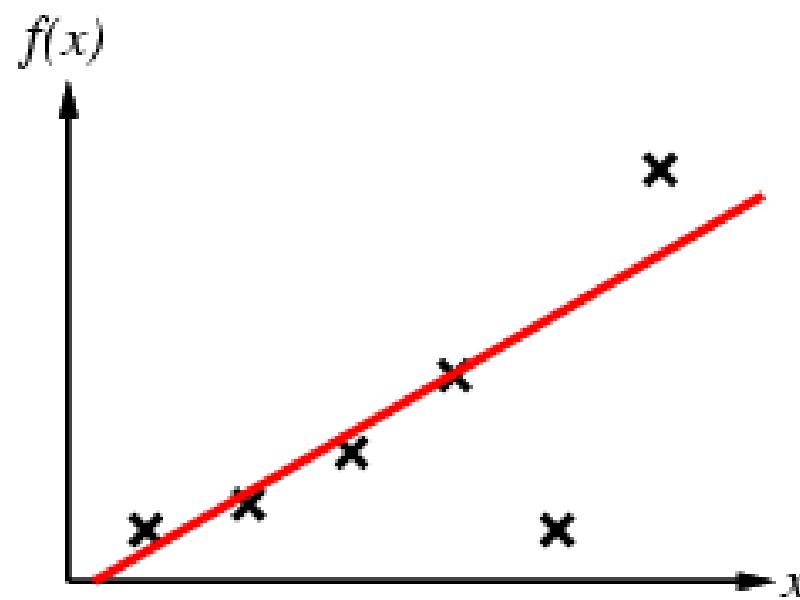


# Aproximarea unei funcții

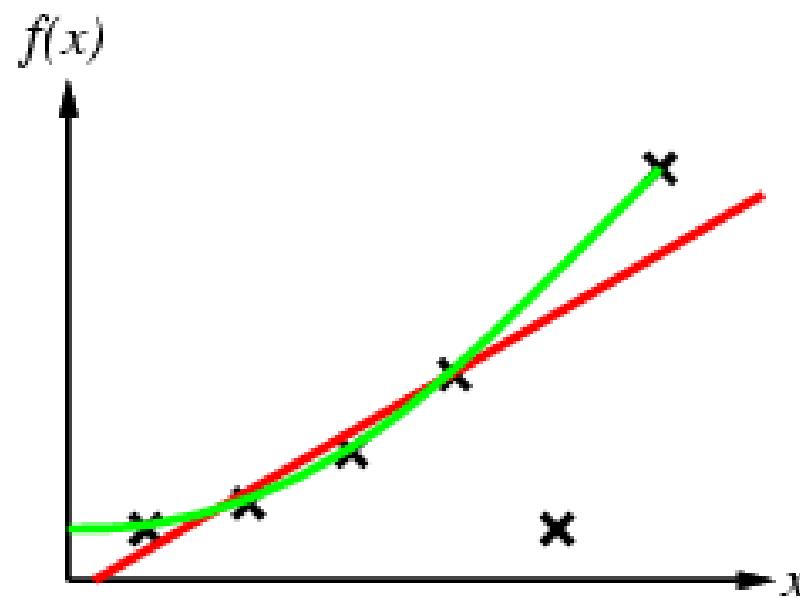
- Să se determine funcția care trece prin punctele de mai jos



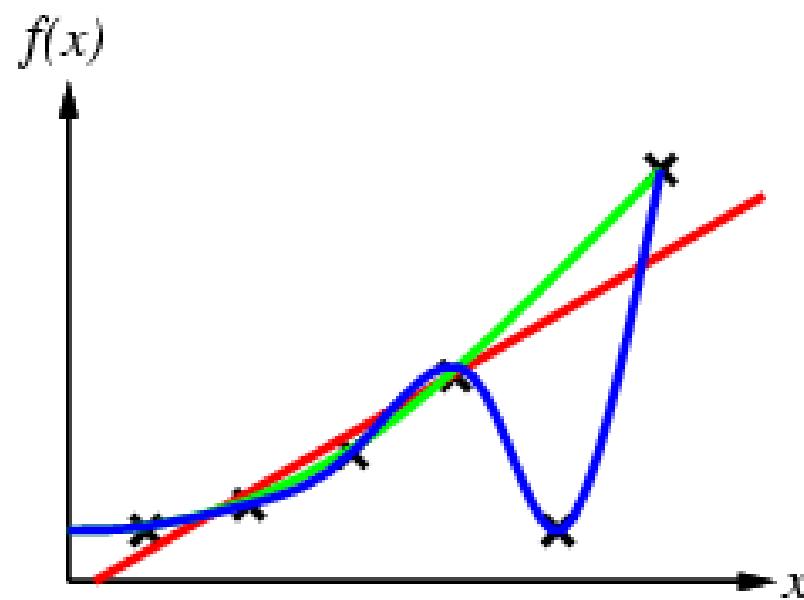
# Aproximarea unei funcții



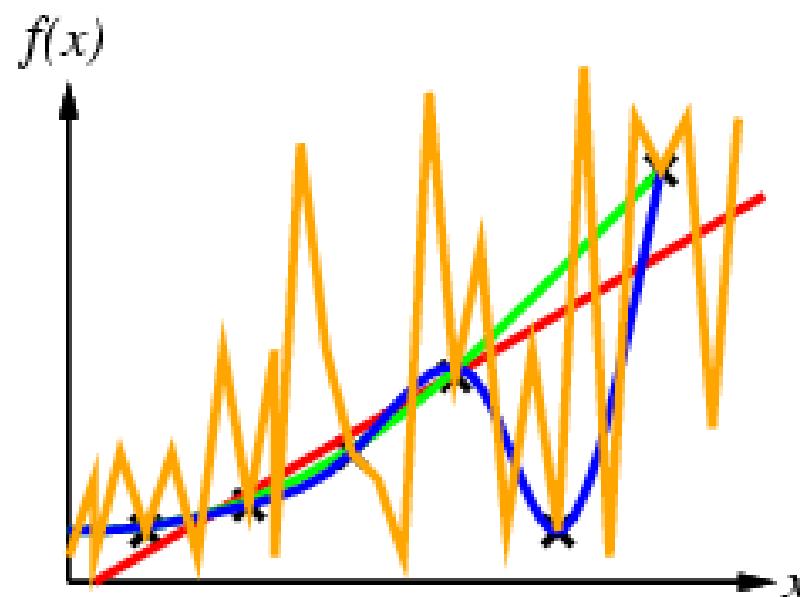
# Aproximarea unei funcții



# Aproximarea unei funcții

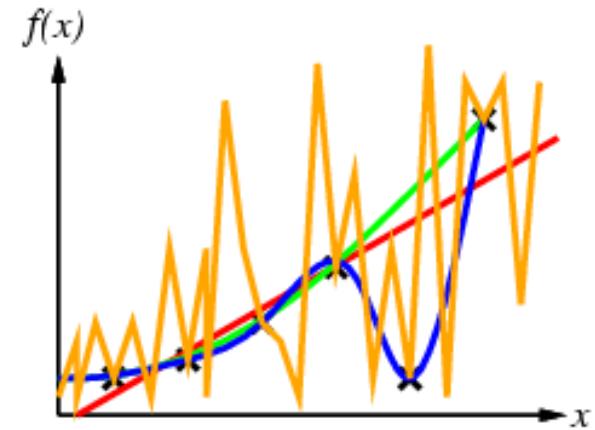


# Aproximarea unei funcții

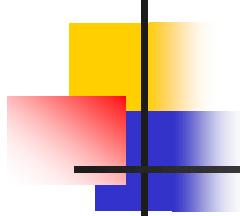


# Briciul lui Occam

- Trebuie preferată cea mai simplă ipoteză consistentă cu datele
- La egalitate, modelele mai simple tind să generalizeze mai bine decât cele complexe

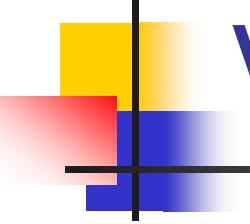


**Lex parsimoniae:** *entia non sunt multiplicanda praeter necessitatem*



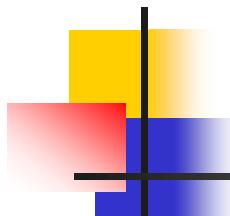
# Generalizarea

- Scopul este găsirea unui model pentru determinarea valorii clasei în funcție de valorile celorlalte atrbute cu eroare cât mai mică
- Modelul trebuie să aibă **capacitate de generalizare**, care arată cât de bun este modelul pentru **date noi**
- De obicei există o **multime de antrenare**, pentru crearea modelului și o **multime de test** pentru verificarea capacității de generalizare



# Validarea

- Uneori, se folosește și o **multime de validare**, pentru a decide când să se oprească antrenarea sau pentru a alege arhitectura cea mai potrivită a modelului
  - Multimea de validare nu este folosită pentru determinarea parametrilor modelului, ci pentru estimarea calității sale
- De exemplu:
  - Antrenăm o rețea neuronală doar cu multimea de antrenare
  - Observăm eroarea pe multimea de validare și oprim antrenarea când aceasta începe să crească
  - Observăm erorile pentru un număr diferit de neuroni în stratul ascuns și alegem numărul cel mai potrivit



# Generarea multimii de test

- Împărțirea 2/3 – 1/3
  - 2/3 pentru antrenare, 1/3 pentru testare
- Validarea încrucișată (*cross-validation*)
  - $n$  grupuri,  $n - 1$  pentru antrenare, al  $n$ -lea pentru testare, se repetă de  $n$  ori
  - De obicei,  $n = 10$
- *Leave one out*
  - $n - 1$  instanțe pentru antrenare, a  $n$ -a pentru testare, se repetă de  $n$  ori

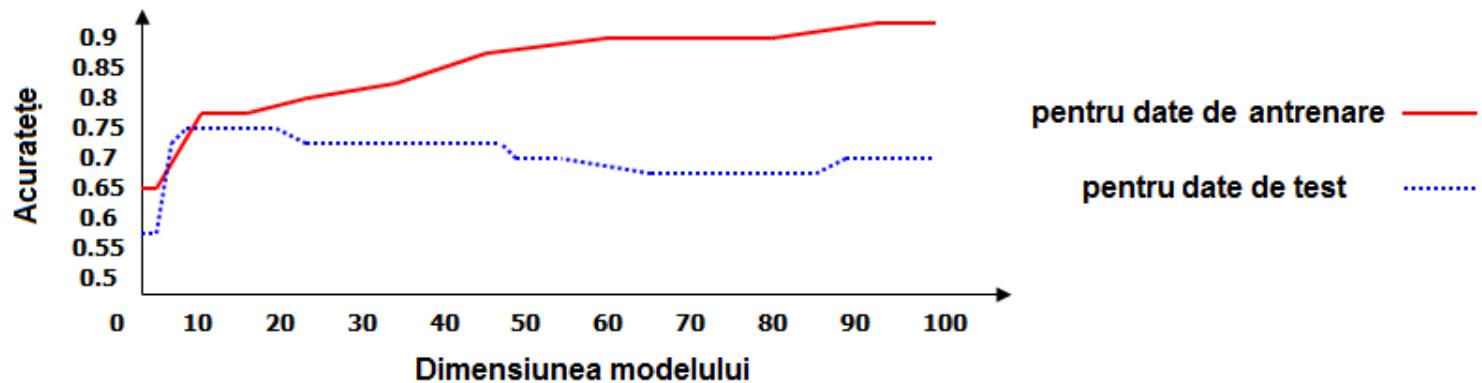
# Exemplu: validarea încrucișată

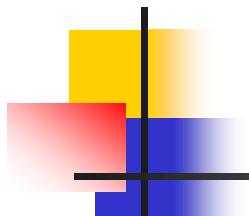
$n = 5$



# Generalitatea unui model

- **Subpotrivirea** (*underfitting*): modelul este prea simplu și nu poate învăța distribuția datelor
- **Suprapotrivirea** (*overfitting*): modelul este prea complex și poate fi influențat de zgomot și date irelevante
- Un model suprapotrivit are performanțe foarte bune pe mulțimea de antrenare, dar performanțe slabe pe mulțimea de validare/test



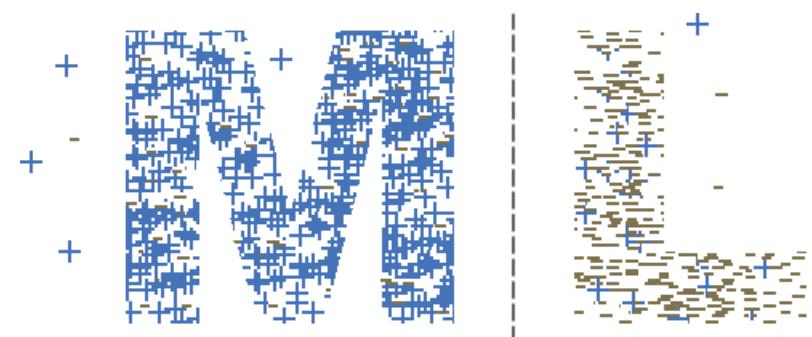


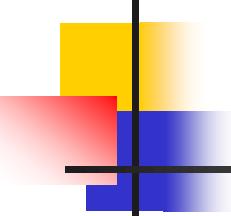
# Briciul lui Occam și Adaboost

- Continuarea iterațiilor pare să nu afecteze generalizarea
- Un model mai complex pare să dea rezultate mai bune
- Mai multe iterații cresc marginea de separare
- Este o consecință a mecanismului de votare

# Regresie liniară, logistică, softmax

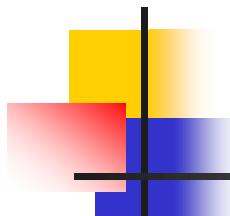
1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
4. Regresia softmax
5. Metrici de eroare





# Regresia liniară

- Multe procese din lumea reală pot fi approximate cu modele liniare
- Regresia liniară apare deseori în modulele unor sisteme mai complexe (perceptron, Adaboost etc.)
- Problemele linare pot fi rezolvate analitic
- Metodologia aplicată aici poate fi aplicată și pentru alte modele (de exemplu, regresie polinomială etc.)
- Pe lângă aproximarea datelor de antrenare, ne interesează și capacitatea de generalizare



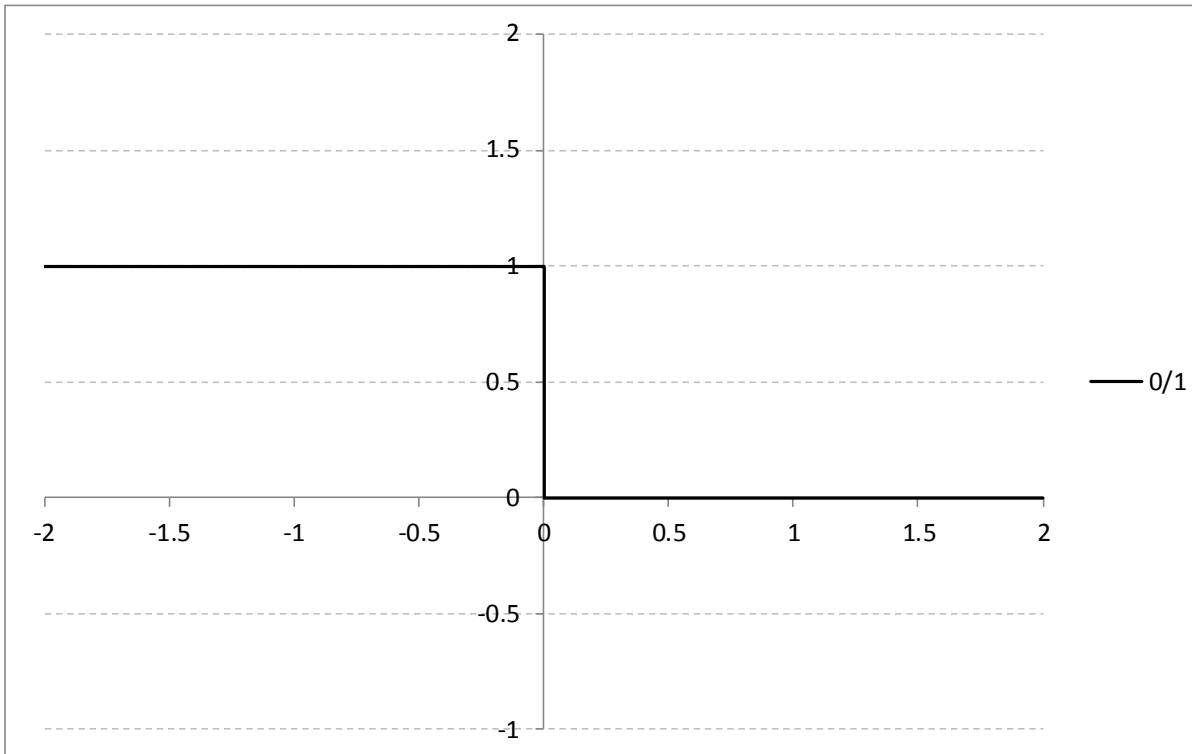
# Numărul de erori

- Un model liniar definește un hiperplan de separare
- Dacă problema este liniar separabilă, poate fi învățată
- Dacă nu, ne interesează un model care face cât mai puține erori

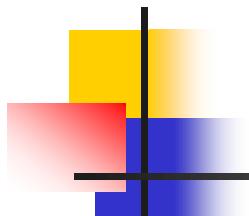
$$\min_{w,b} \sum_i \mathbf{1}[y_i \cdot (w \cdot x_i + b) < 0]$$

- Această funcție este **rata de eroare** sau **funcția de cost 0/1**
- $\mathbf{1}[\cdot]$  = funcția indicator:  $\mathbf{1}[adevărat] = 1$  și  $\mathbf{1}[fals] = 0$
- Aici, funcția indicator este folosită pentru a număra instanțele clasificate greșit
- $y_i$  = **ieșirea dorită** pentru instanța  $i$ ,  $y_i \in \{-1, 1\}$
- $(w \cdot x_i + b)$  = **ieșirea modelului**  $\in \mathbb{R}$
- $y_i \cdot (w \cdot x_i + b)$  = **marginea**

# Funcția de cost 0/1

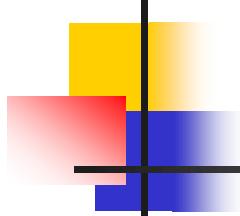


Pe axa  $x$  este marginea:  $y_i \cdot (w \cdot x_i + b)$



# Problema de optimizare

- Problema de optimizare este NP dificilă atât de rezolvat exact cât și de aproximată cu o precizie arbitrară
- Motivul este că o schimbare mică în parametri poate determina o schimbare mare în clasificare – între 0 și 1

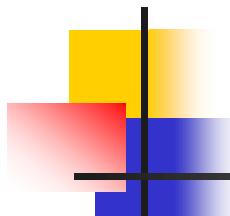


# Regularizarea

- Problema de optimizare anterioară se referă la minimizarea numărului de erori pe mulțimea de antrenare
- Nu se referă la performanțele pentru date noi
- În acest scop, se introduce un termen de regularizare:

$$\min_{w,b} \sum_i \mathbf{1}[y_i \cdot (w \cdot x_i + b) < 0] + \lambda R(w, b)$$

- Primul termen dorește minimizarea numărului de erori de antrenare
- Al doilea termen dorește determinarea unui model cât mai simplu, care să generalizeze cât mai bine



# Funcții de cost

- Funcțiile de cost (*loss functions*) definesc limite superioare ale ratei de eroare
- Prin minimizarea lor, se minimizează și rata de eroare

Zero/one:

$$\ell^{(0/1)}(y, \hat{y}) = \mathbf{1}[y\hat{y} \leq 0]$$

$$\hat{y} = w \cdot x + b.$$

Hinge:

$$\ell^{(\text{hin})}(y, \hat{y}) = \max\{0, 1 - y\hat{y}\}$$

Logistic:

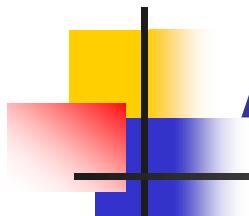
$$\ell^{(\log)}(y, \hat{y}) = \frac{1}{\log 2} \log (1 + \exp[-y\hat{y}])$$

Exponential:

$$\ell^{(\exp)}(y, \hat{y}) = \exp[-y\hat{y}]$$

Squared:

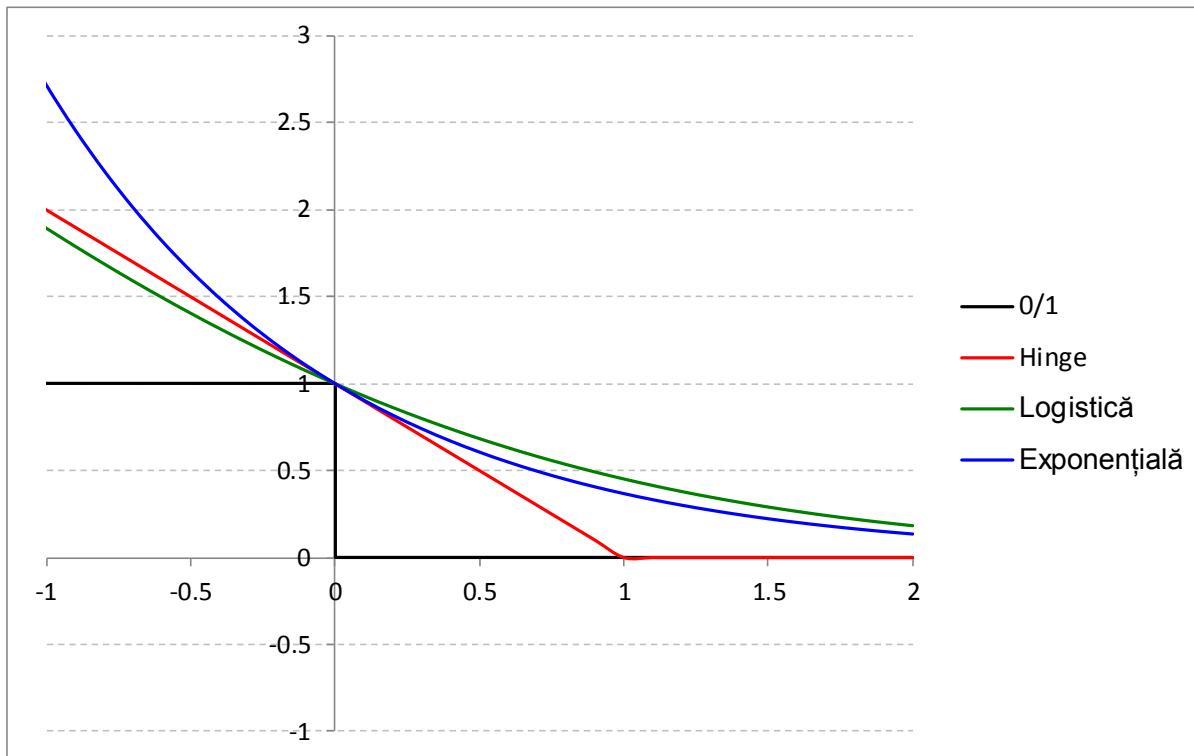
$$\ell^{(\text{sqr})}(y, \hat{y}) = (y - \hat{y})^2$$



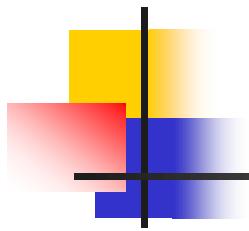
# Algoritmi și funcții de cost

- *Decision stump* folosește funcția de cost 0/1
- Perceptronul multistrat folosește funcția de cost pătratică
- Adaboost folosește funcția de cost exponentială

# Reprezentare grafică

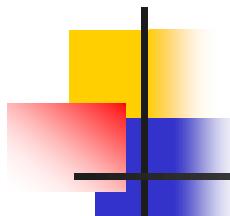


Pe axa  $x$  este marginea:  $y_i \cdot (w \cdot x_i + b)$



# Functii de cost

- Predicții eronate
  - Funcția *hinge*: creștere liniară
  - Functiile pătratică, exponentială: creștere supra-liniară, preferă câteva instanțe puțin greșite față de una singură complet eronată
- Predicții corecte
  - Funcția *hinge* nu mai face diferență
  - Functiile pătratică, exponentială pot optimiza în continuare, pentru o mai mare încredere



# Regularizarea

- Problema de optimizare este similară, dar folosește orice funcție de cost  $l$  și un termen suplimentar  $R$

$$\min_{w,b} \sum_i l(y_i, w \cdot x_i + b) + \underline{\lambda R(w, b)}$$

$$R^{(\text{norm})}(w, b) = \|w\| = \sqrt{\sum_d w_d^2}.$$

$$R^{(\text{sqr})}(w, b) = \|w\|^2 = \sum_d w_d^2$$

$$R^{(\text{abs})}(w, b) = \sum_d |w_d|.$$

# Regresia liniară cu gradient descendent

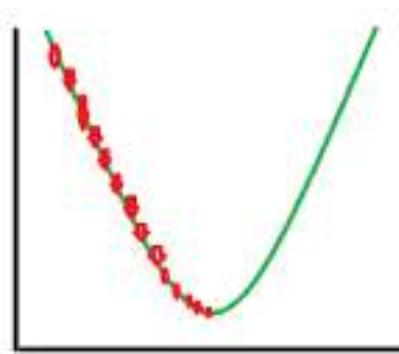
---

**Algorithm**     $\text{GRADIENTDESCENT}(\mathcal{F}, K, \eta_1, \dots)$

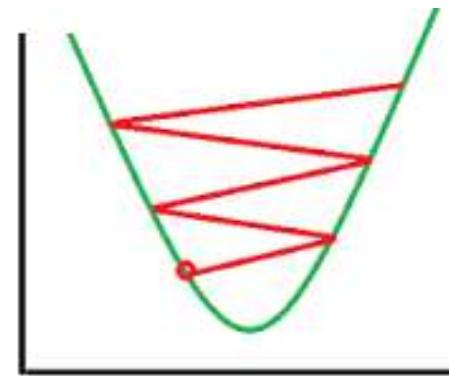
---

```
1:  $z^{(0)} \leftarrow \langle o, o, \dots, o \rangle$  // initialize variable we are optimizing
2: for  $k = 1 \dots K$  do
3:    $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$  // compute gradient at current location
4:    $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$  // take a step down the gradient
5: end for
6: return  $z^{(K)}$ 
```

---



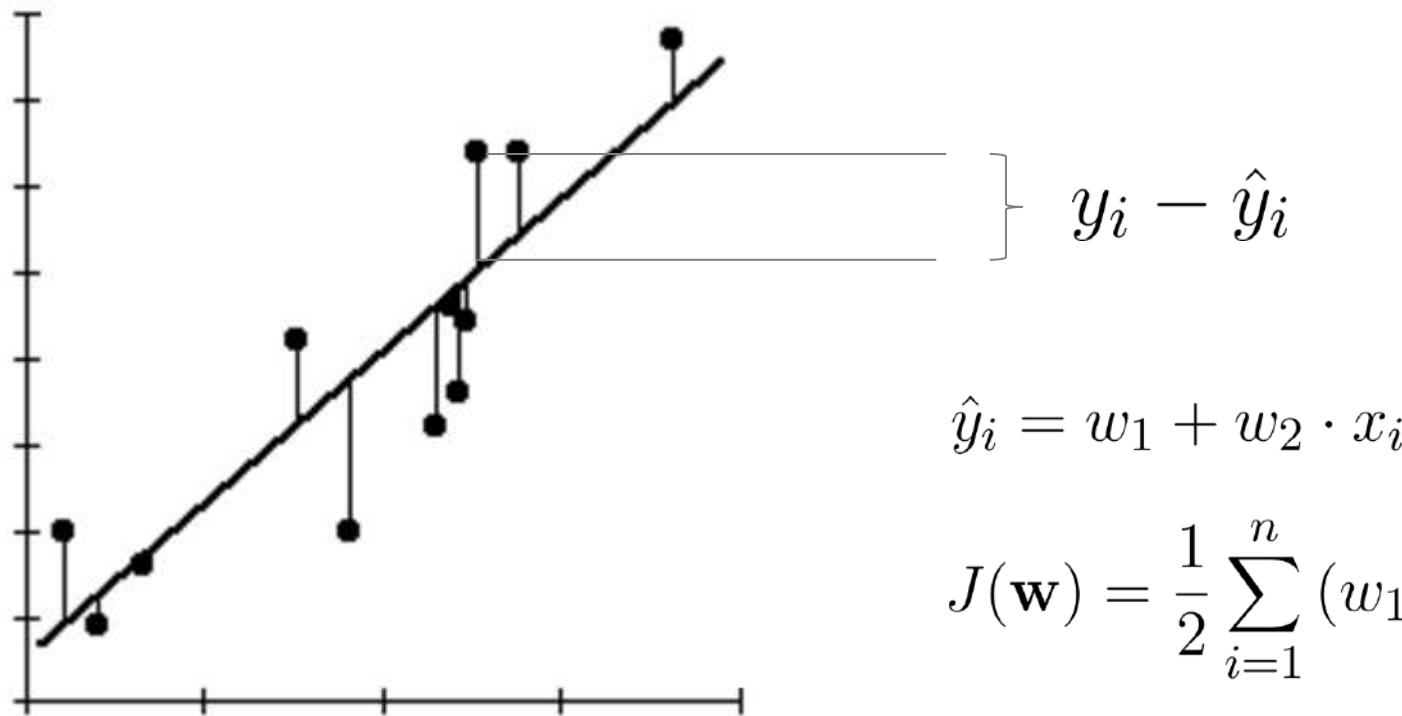
pas mic  
algoritmul converge



pas mare  
algoritmul diverge

# Funcția de cost pătratică

- În general, regresia liniară minimizează eroarea medie pătratică (MSE)
- w este un vector  $n$ -dimensional



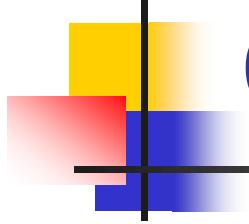
# Calculul gradientilor (cu regularizare)

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (w_1 + w_2 \cdot x_i - y_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad \|\mathbf{w}\|^2 = w_1^2 + w_2^2$$

$$g_1 = \frac{\partial J(\mathbf{w})}{\partial w_1} = \left( \sum_{i=1}^n (\hat{y}_i - y_i) \right) + \lambda \cdot w_1 = \sum_{i=1}^n e_i + \lambda \cdot w_1$$

$$g_2 = \frac{\partial J(\mathbf{w})}{\partial w_2} = \left( \sum_{i=1}^n (\hat{y}_i - y_i) \cdot x_i \right) + \lambda \cdot w_2 = \sum_{i=1}^n e_i \cdot x_i + \lambda \cdot w_2$$

$$w_i \leftarrow w_i - \alpha \cdot g_i$$



# Criterii de terminare

- Iterațiile se pot termina când:
  - A fost atins un număr maxim prestabilit de iterării
  - Funcția obiectiv nu se mai modifică mult
  - Parametrii nu se mai modifică mult

# Exemplul 1. Regresie liniară

```
var x = new double[] { 1, 2, 3 };
var y = new double[] { 1, 2, 4 };
var w = new double[] { 1, 1 }; // estimari initiale

int noInstances = 3;
var yo = new double[noInstances]; // iesirea modelului
double alpha = 0.1; // rata de invatare
double mse = 1;

bool over = false;
while (!over)
{
    double prevMse = mse;
    mse = 0;
    var dw = new double[2];

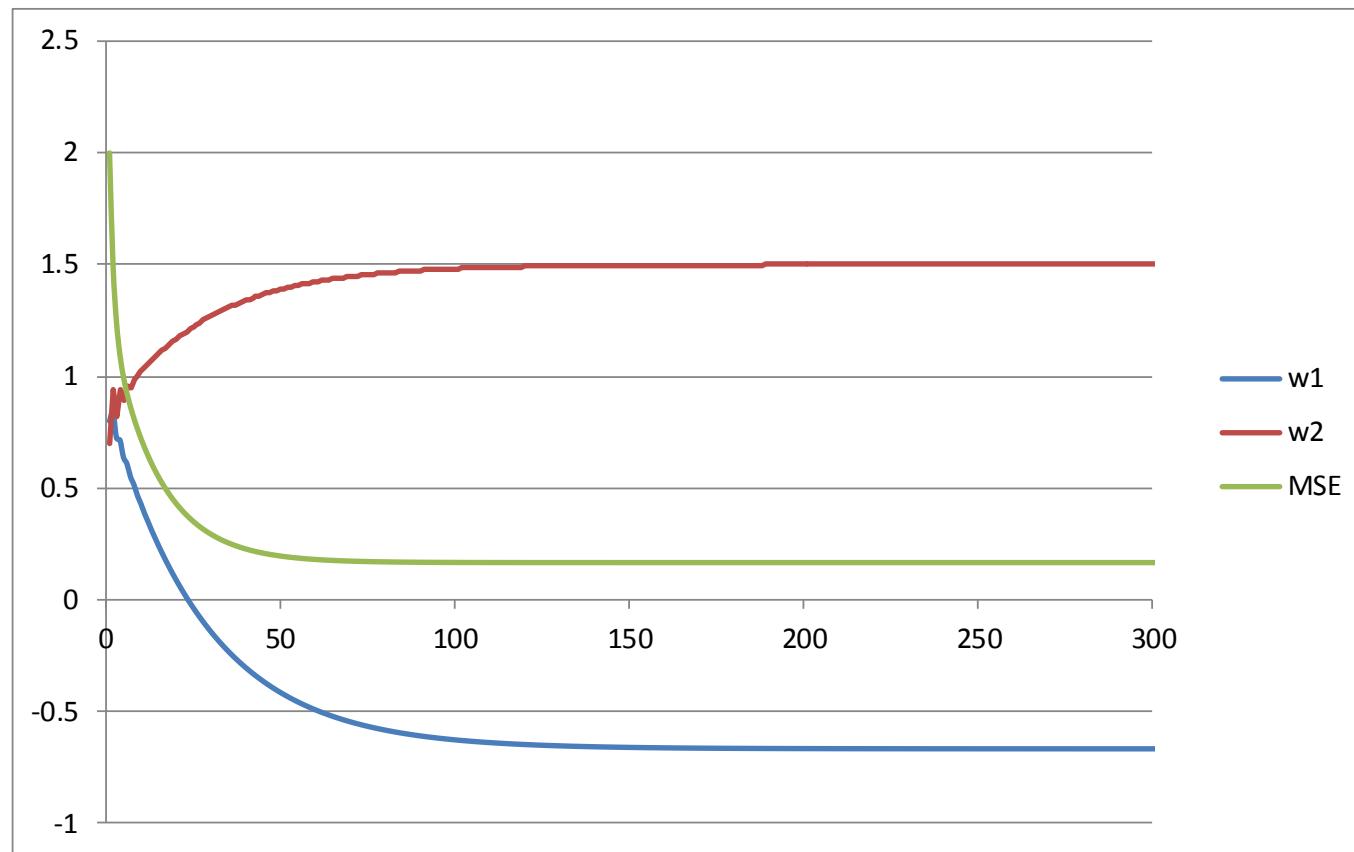
    for (int i = 0; i < noInstances; i++)
    {
        yo[i] = w[0] + w[1] * x[i];
        double err = yo[i] - y[i];
        mse += err * err;
        dw[0] += err;
        dw[1] += x[i] * err;
    }

    w[0] -= alpha * dw[0];
    w[1] -= alpha * dw[1];

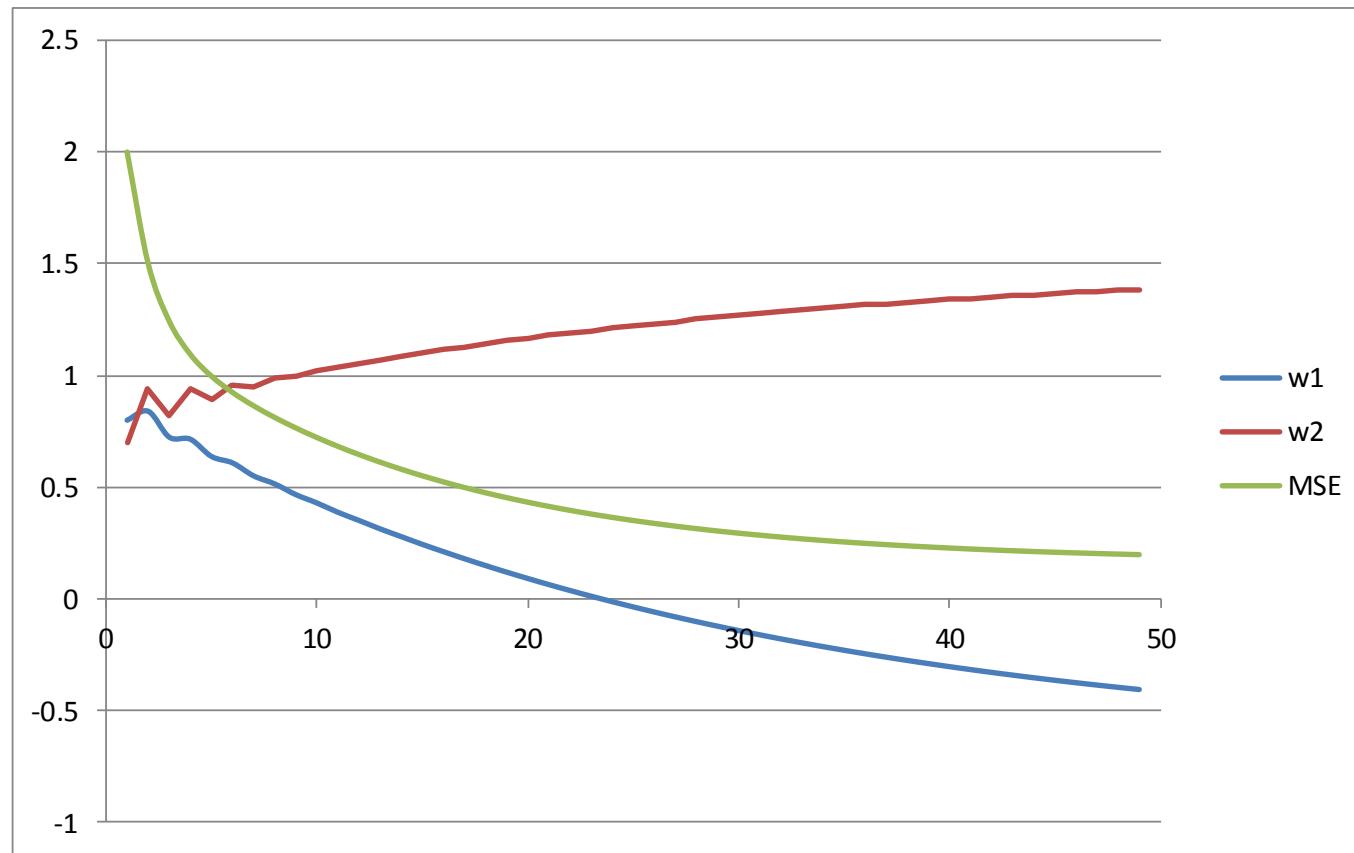
    if (Math.Abs(mse - prevMse) < 1e-10)
        over = true;
}
```

Rezultate:  
w[0] = -0.667  
w[1] = 1.5

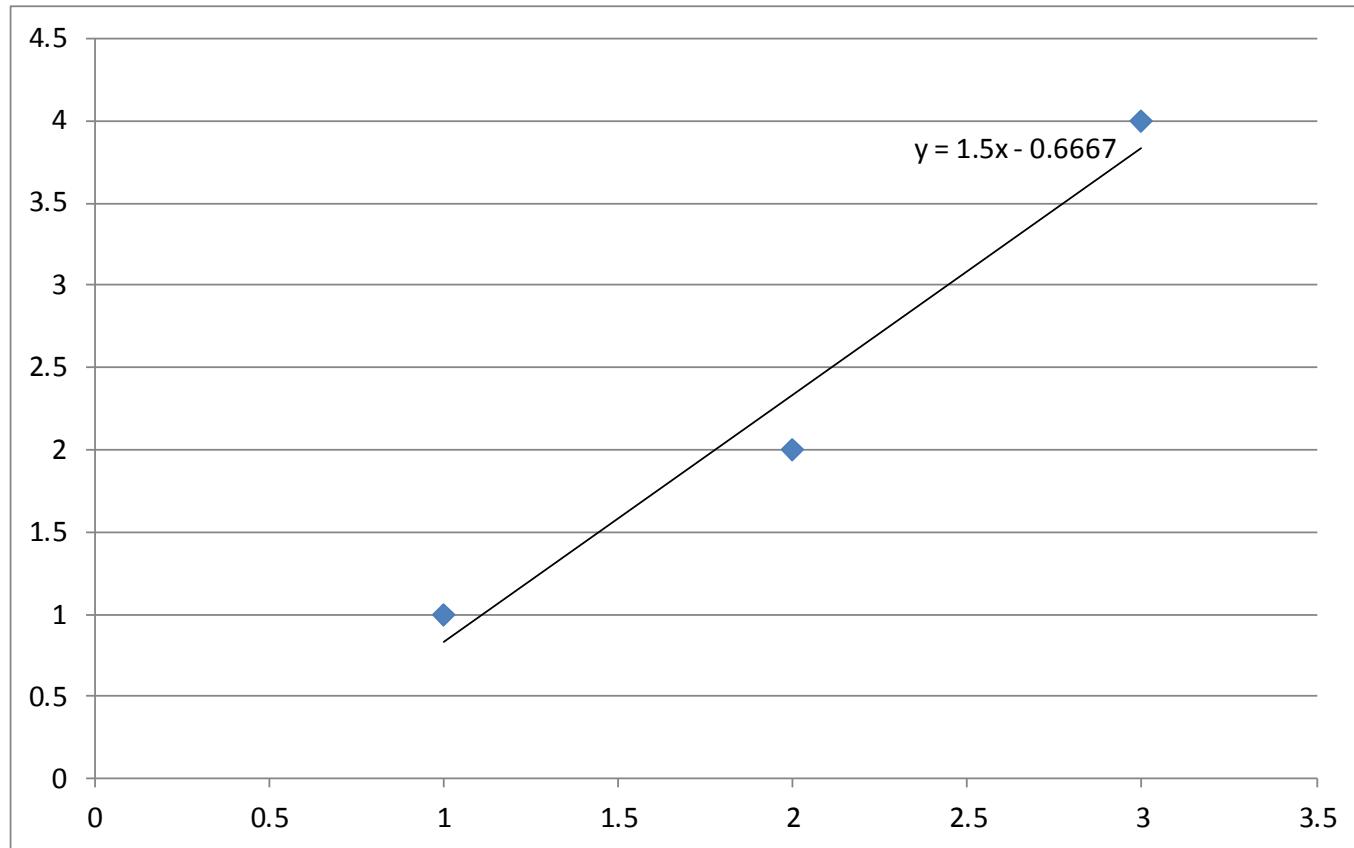
# Convergență (300 de iterații)



# Convergență (50 de iterări)



# Rezultat



## Exemplul 2. Regresie liniară cu regularizare

```
var x = new double[] { 1, 2, 3, 4, 5 };
var y = new double[] { 1, 2, 3, 4, 20 };
var w = new double[] { 1, 1 }; // estimari initiale

int noInstances = 5;
var yo = new double[noInstances]; // iesirea modelului
double alpha = 0.01; // rata de invatare
double lambda = 100; // 0, 1, 10, 100 (daca e foarte mare, trebuie miscorata alpha)
int epoch = 0; double mse = 1;

while (epoch < 1000)
{
    mse = 0; epoch++;
    var dw = new double[2];

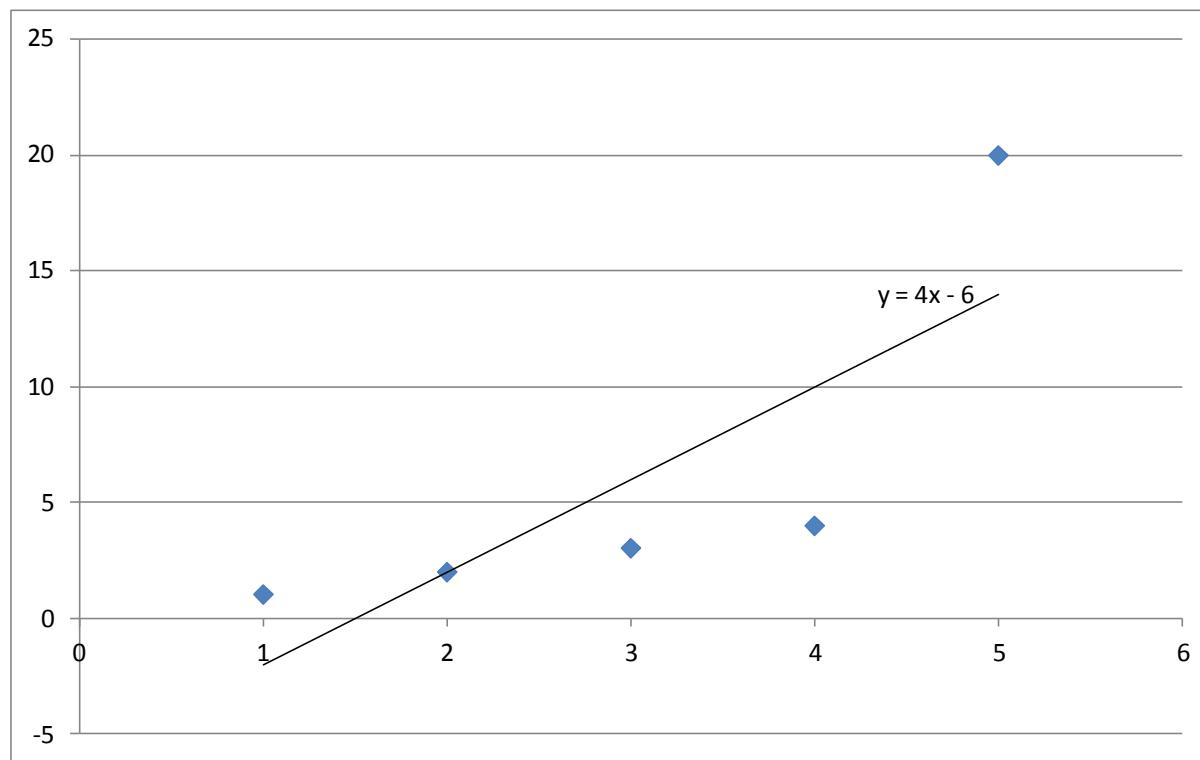
    for (int i = 0; i < noInstances; i++)
    {
        yo[i] = w[0] + w[1] * x[i];
        double err = yo[i] - y[i];
        mse += err * err;
        dw[0] += err;
        dw[1] += x[i] * err;
    }

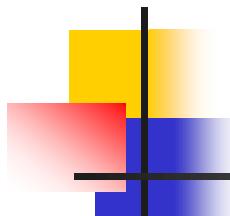
    dw[0] += lambda * w[0];
    dw[1] += lambda * w[1];

    w[0] -= alpha * dw[0];
    w[1] -= alpha * dw[1];
}
```

# Exemplul 2

- $\lambda = 0 \Rightarrow w_1 = -6, w_2 = 4$





## Exemplul 2

- $\lambda = 0 \Rightarrow w_1 = -6, w_2 = 4$
- $\lambda = 1 \Rightarrow w_1 = -2.432, w_2 = 2.973$
- $\lambda = 10 \Rightarrow w_1 = 0, w_2 = 2$
- $\lambda = 100 \Rightarrow w_1 = 0.168, w_2 = 0.822$
- $\lambda = 1000 \Rightarrow w_1 = 0.028, w_2 = 0.123$
  
- Dacă  $\lambda$  este prea mic, modelul poate fi afectat de valori extreme sau erori
- Dacă  $\lambda$  este prea mare, modelul nu mai aproximează bine deoarece parametrii săi tind la 0

# Optimizare diferențială pentru funcții de cost nediferențiable

- Se utilizează subgradienți
- De exemplu, pentru funcția *hinge*

$$\begin{aligned}\partial_w \max\{0, 1 - y_n(w \cdot x_n + b)\} \\ &= \partial_w \begin{cases} 0 & \text{if } y_n(w \cdot x_n + b) > 1 \\ y_n(w \cdot x_n + b) & \text{otherwise} \end{cases} \\ &= \begin{cases} \partial_w 0 & \text{if } y_n(w \cdot x_n + b) > 1 \\ \partial_w y_n(w \cdot x_n + b) & \text{otherwise} \end{cases} \\ &= \begin{cases} 0 & \text{if } y_n(w \cdot x_n + b) > 1 \\ y_n x_n & \text{otherwise} \end{cases}\end{aligned}$$

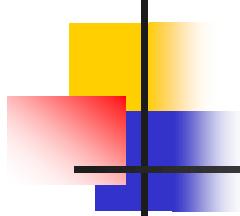
# Regresia liniară rezolvată matriceal, exact

$$\hat{\mathbf{Y}} = \mathbf{X} \cdot \mathbf{W}$$

$$\begin{pmatrix} \hat{y}_1 \\ \dots \\ \hat{y}_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1d} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{nd} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \dots \\ w_n \end{pmatrix}$$

$$\mathbf{W} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y}$$

$$\mathbf{W} = (\mathbf{X}^T \cdot \mathbf{X} + \underline{\lambda \mathbf{I}_D})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y} \quad \longleftarrow \text{cu regularizare}$$



# Exemplu

$$\mathbf{X} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \quad \mathbf{Y} = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$$

$$\mathbf{W} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y} = \begin{pmatrix} -2/3 \\ 3/2 \end{pmatrix}$$

---

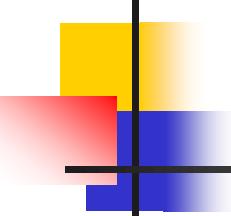
(2x3)·(3x2)=(2x2)

---

(2x2)·(2x3)=(2x3)

---

(2x3)·(3x1)=(2x1)



# Complexitate

- $n$  instanțe,  $d$  dimensiuni
  - De fapt, este  $d+1$ , nu  $d$ , din cauza coloanei suplimentare cu valori de 1
- Metoda gradientului descendente:  $O(knd)$ ,  $k$  iterații
- Metoda matriceală:  $O(d^3 + d^2n)$ , dacă  $n > d \Rightarrow O(d^2n)$

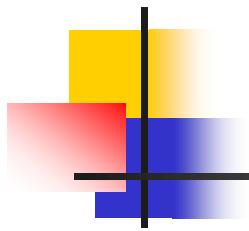
$$R_1 = X^T \cdot X : (dn, nd \rightarrow dd) \Rightarrow O(d^2n)$$

$$R_2 = R_1^{-1} : (dd) \Rightarrow O(d^3)$$

$$R_3 = R_2 \cdot X^T : (dd, dn \rightarrow dn) \Rightarrow O(d^2n)$$

$$R_4 = R_3 \cdot Y : (dn, n1 \rightarrow d1) \Rightarrow O(dn)$$

- Pentru un număr mic și mediu de dimensiuni (de exemplu, mai mic de 100), este preferabilă metoda matriceală
- Pentru un număr mare, se preferă metoda diferențială



# Funcția de cost bazată pe MLE

- Estimarea verosimilității maxime (*maximum likelihood estimation*), a se revedea cursul 1

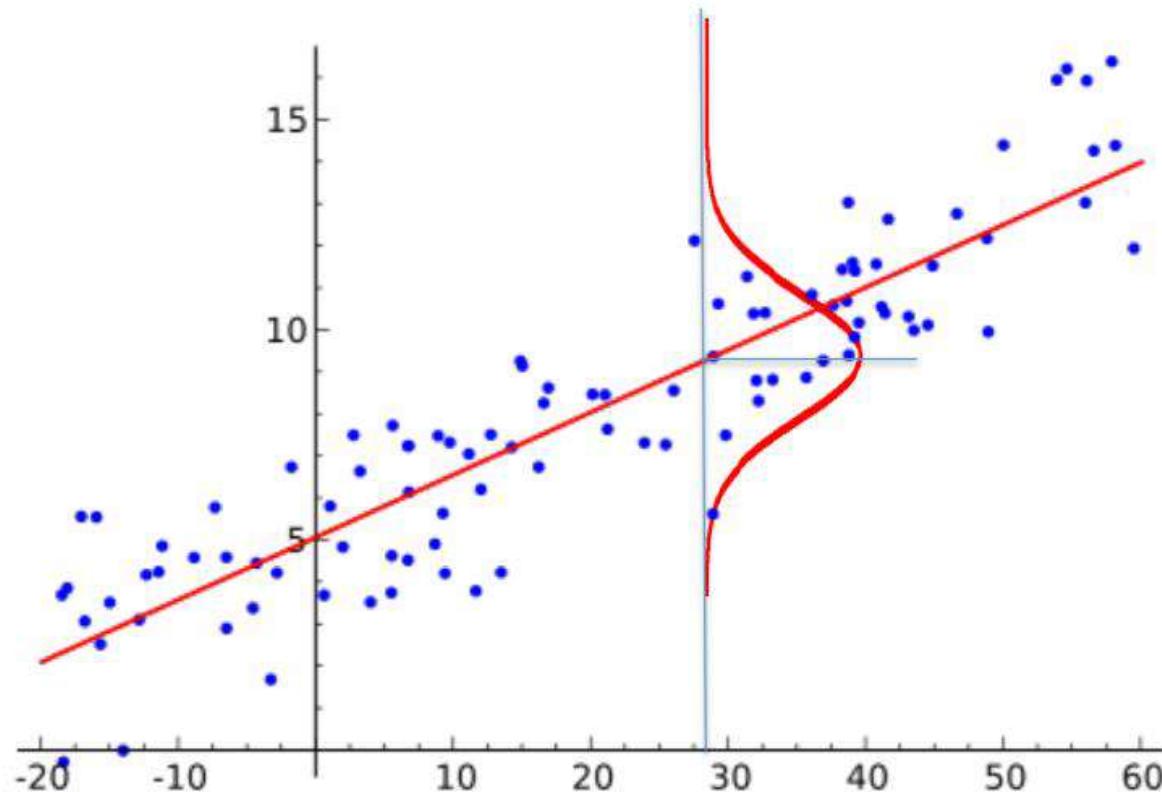
$$\mathbf{w}_{ML} = \arg \max_{\mathbf{w}} P(\mathbf{Y}|\mathbf{X}, \mathbf{w})$$

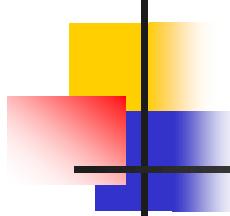
$$\mathbf{w}_{ML} = \arg \max_{\mathbf{w}} \sum_i \log P(y_i|x_i, \mathbf{w})$$

- Se presupune că erorile aparțin unei distribuții gaussiene și că instanțele sunt independente și identic distribuite (*i.i.d.*)

$$\hat{y}_i = w_1 + w_2 \cdot x_i$$

$$P(y_i|x_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2} (w_1 + w_2 \cdot x_i - y_i)^2\right)$$





# MLE vs. MSE

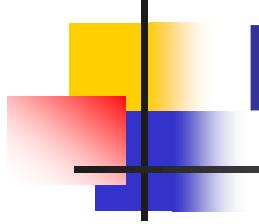
- MLE:

$$\sum_{i=1}^n \log P(y_i|x_i, \mathbf{w}) = -n \log \sigma - \frac{n}{2} \log (2\pi) - \sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{2\sigma^2} \quad \leftarrow$$

- MSE:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad \leftarrow$$

- Pentru regresia liniară, optimizarea după MSE dă același rezultat cu optimizarea după MLE
  - Dacă datele sunt i.i.d. și urmează o distribuție gaussiană cu varianță constantă

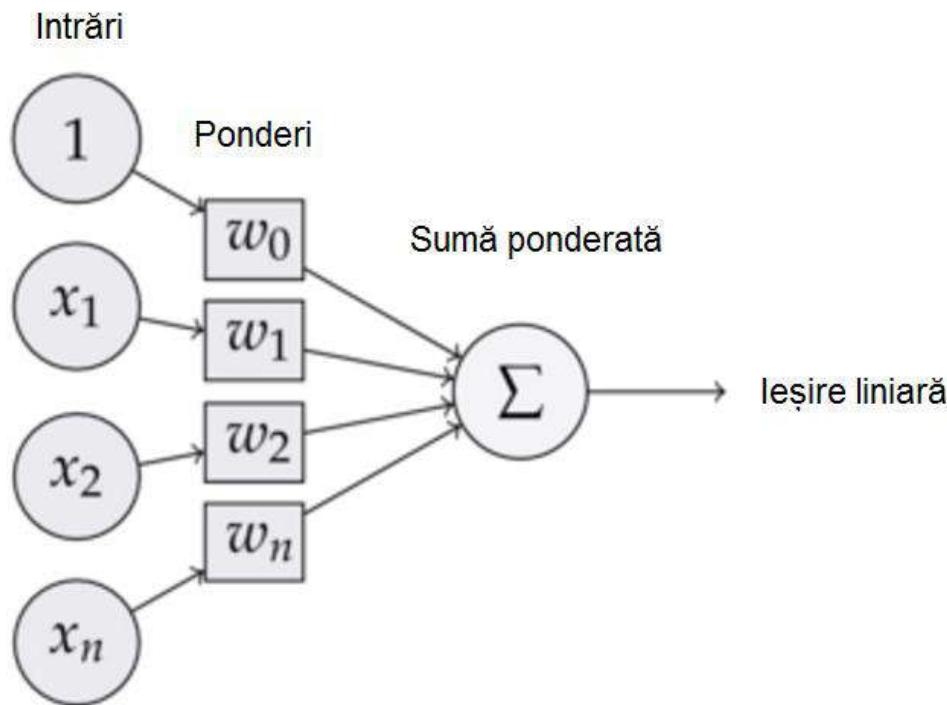


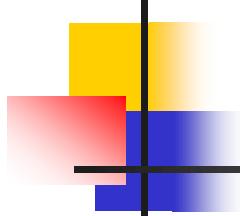
# MLE vs. MSE

- Optimizarea după MLE este în general preferată pentru metodele de învățare automată recente
- Proprietăți ale MLE:
  - Consistență: când numărul de instanțe tinde la infinit, optimizarea după MLE tinde la parametrii reali (MLE este cel mai bun estimator asimptotic)
  - Eficiență statistică: mai puține erori la generalizare
- Când soluțiile MSE și MLE sunt diferite, ar trebui preferată soluția MLE
- MLE se poate folosi împreună cu metodele de regularizare

# Regresia liniară

- Modelul de regresie liniară are aceeași structură ca un perceptron cu funcție de activare liniară (adaline)

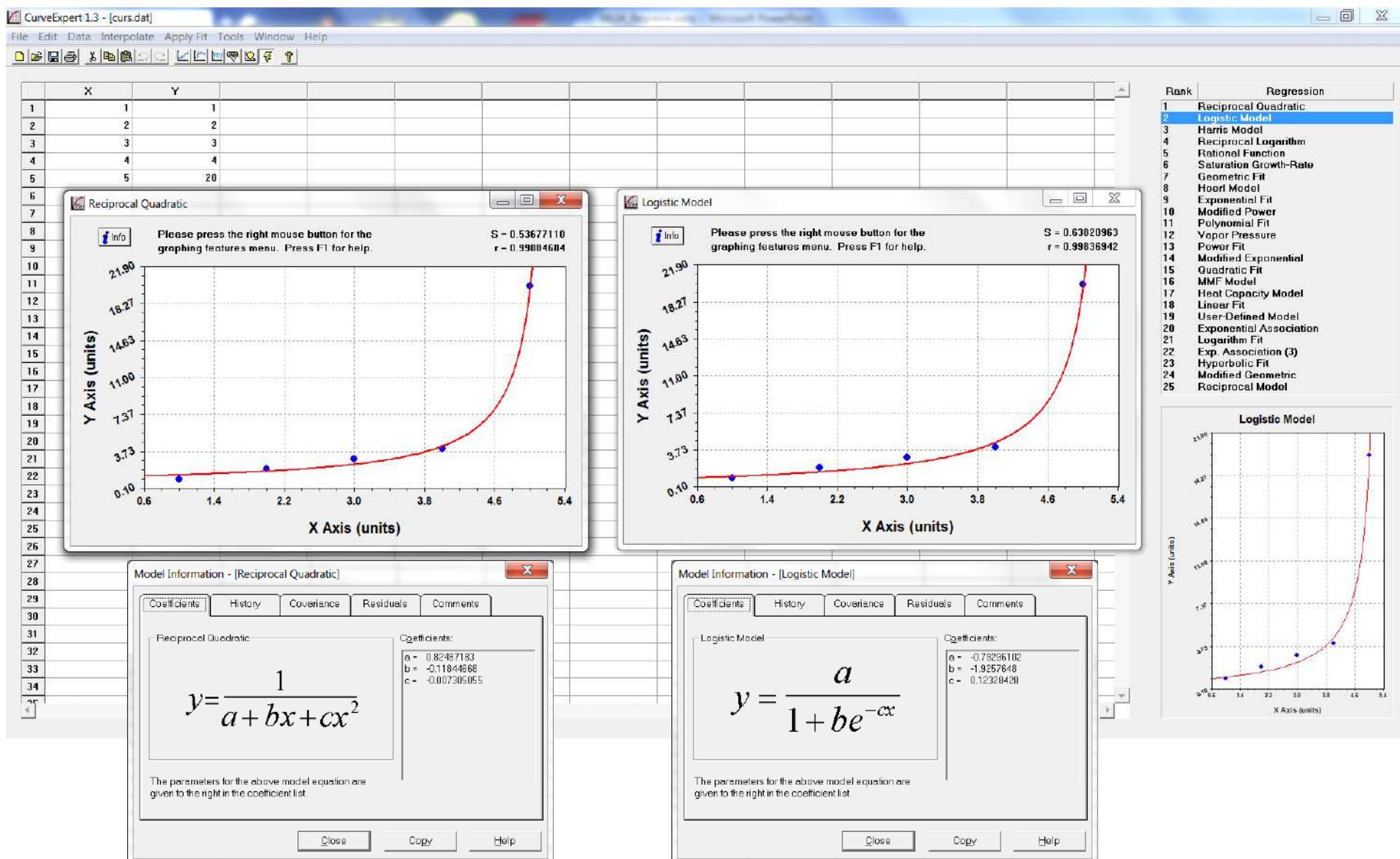




# Discuție

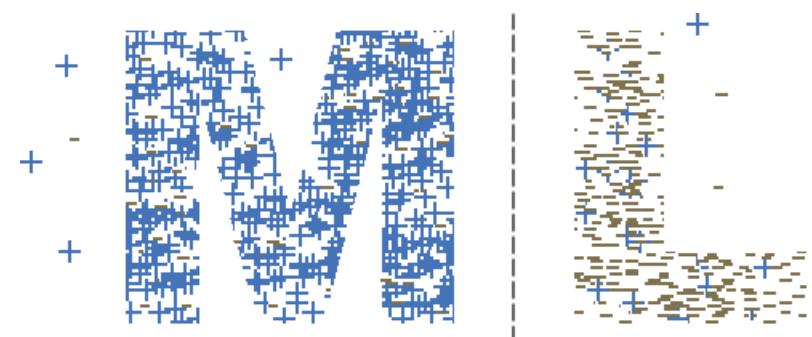
- Cu aceleasi metode se pot aproxima orice modele de regresie
- Prin metoda diferențială trebuie formată funcția obiectiv și trebuie calculați gradientii parametrilor
- Exemplu de program de regresie cu o mare varietate de modele: *CurveExpert*
  - <https://www.curveexpert.net>

# Curve Expert



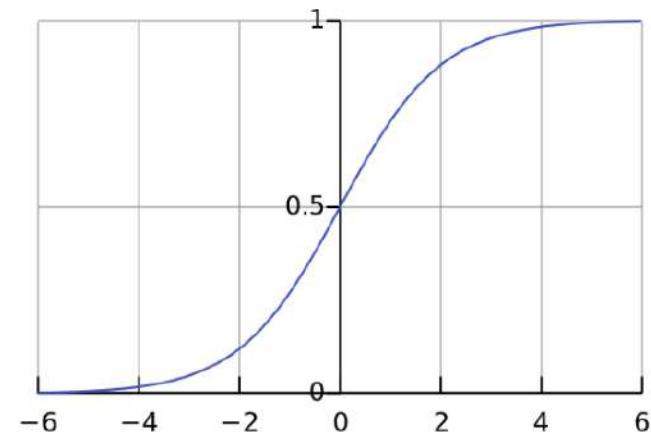
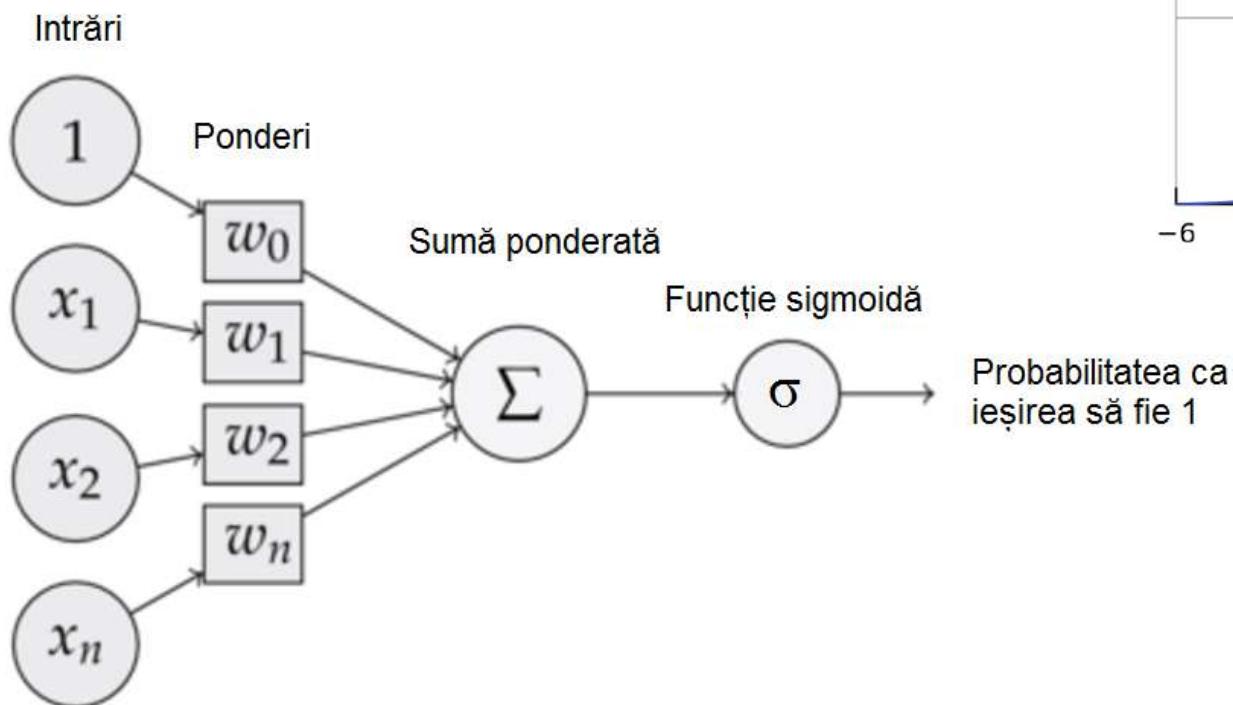
# Regresie liniară, logistică, softmax

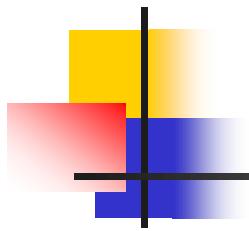
1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
4. Regresia softmax
5. Metrici de eroare



# Regresia logistică

## ■ Structură



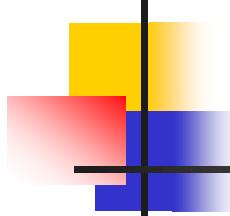


# Regresia logistică

- Ieșirea modelului reprezintă probabilitatea de apartenență la o clasă

$$P(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x}_i)} = \sigma(\mathbf{w}^T \cdot \mathbf{x}_i) = p_i$$

$$P(y_i = 0 | \mathbf{x}_i, \mathbf{w}) = 1 - P(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \cdot \mathbf{x}_i) = 1 - p_i$$



# Funcția de cost

- Se bazează pe MLE
- Regresia logistică tratează probleme cu două clase
- Un experiment stochastic cu două rezultate posibile este modelat de distribuția Bernoulli

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \prod_{i=1}^n P(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^n \text{Ber}(y_i|\sigma(\mathbf{x}_i \cdot \mathbf{w})) = \\ \prod_{i=1}^n \left( \frac{1}{1+\exp(-\mathbf{x}_i \cdot \mathbf{w})} \right)^{y_i} \left( 1 - \frac{1}{1+\exp(-\mathbf{x}_i \cdot \mathbf{w})} \right)^{1-y_i}$$

- Optimizarea este mai simplă după logaritmare

$$J(\mathbf{w}) = - \sum_i (y_i \cdot \log p_i + (1 - y_i) \cdot \log(1 - p_i))$$

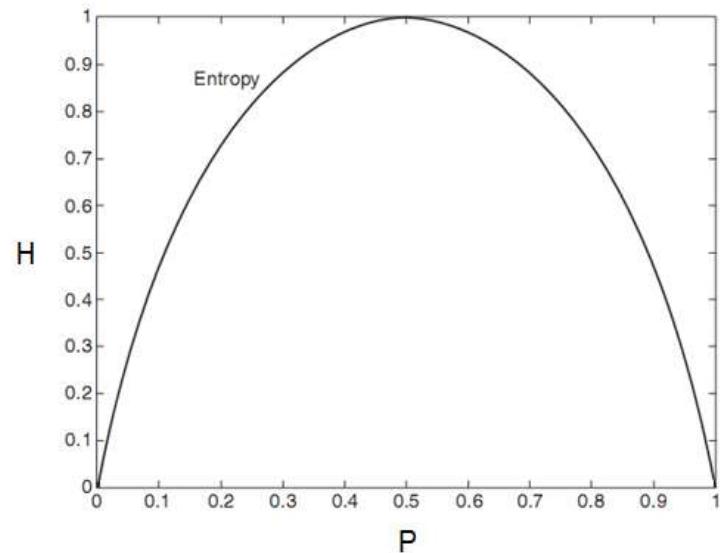
# Entropia

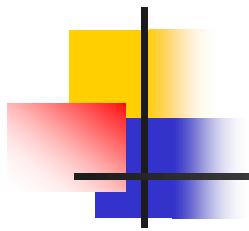
- Entropia este o măsură a incertitudinii asociate cu o variabilă aleatorie

$$H_X = - \sum_x p_x \log p_x$$

- Pentru o variabilă binară este:

$$H_X = - [p_x \log p_x + (1 - p_x) \log (1 - p_x)]$$



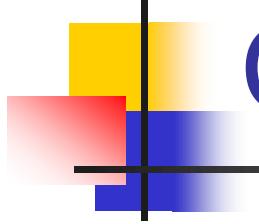


# Funcția de cost *cross-entropy*

- Expresia lui  $J$  se mai numește *cross-entropy*
  - Se referă atât la datele de antrenare ( $y_i$ ), cât și la ieșirile modelului ( $p_i$ )
  - Scopul este ca probabilitățile modelului să fie cât mai apropiate de datele de antrenare

$$J(\mathbf{w}) = - \sum_i (y_i \cdot \log p_i + (1 - y_i) \cdot \log(1 - p_i))$$

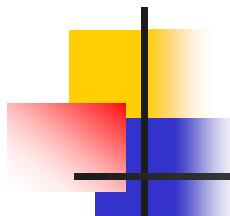
$$H_{\mathbf{X}} = - [p_x \log p_x + (1 - p_x) \log (1 - p_x)]$$



# Gradientii

$$J(\mathbf{w}) = - \sum_i (y_i \cdot \log p_i + (1 - y_i) \cdot \log(1 - p_i))$$

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_{i=1}^n x_{ij} (p_i - y_i)$$



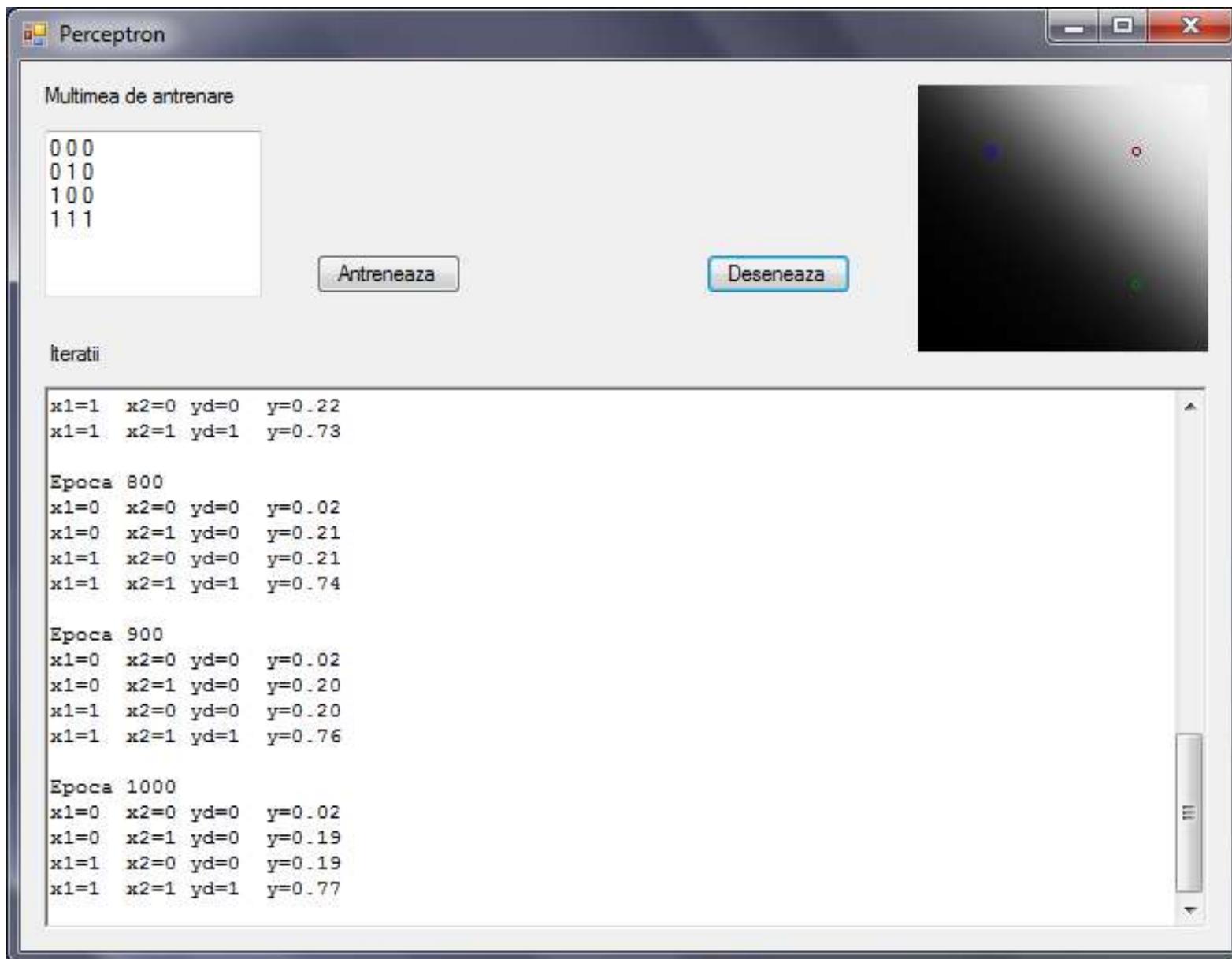
# Reguli de antrenare

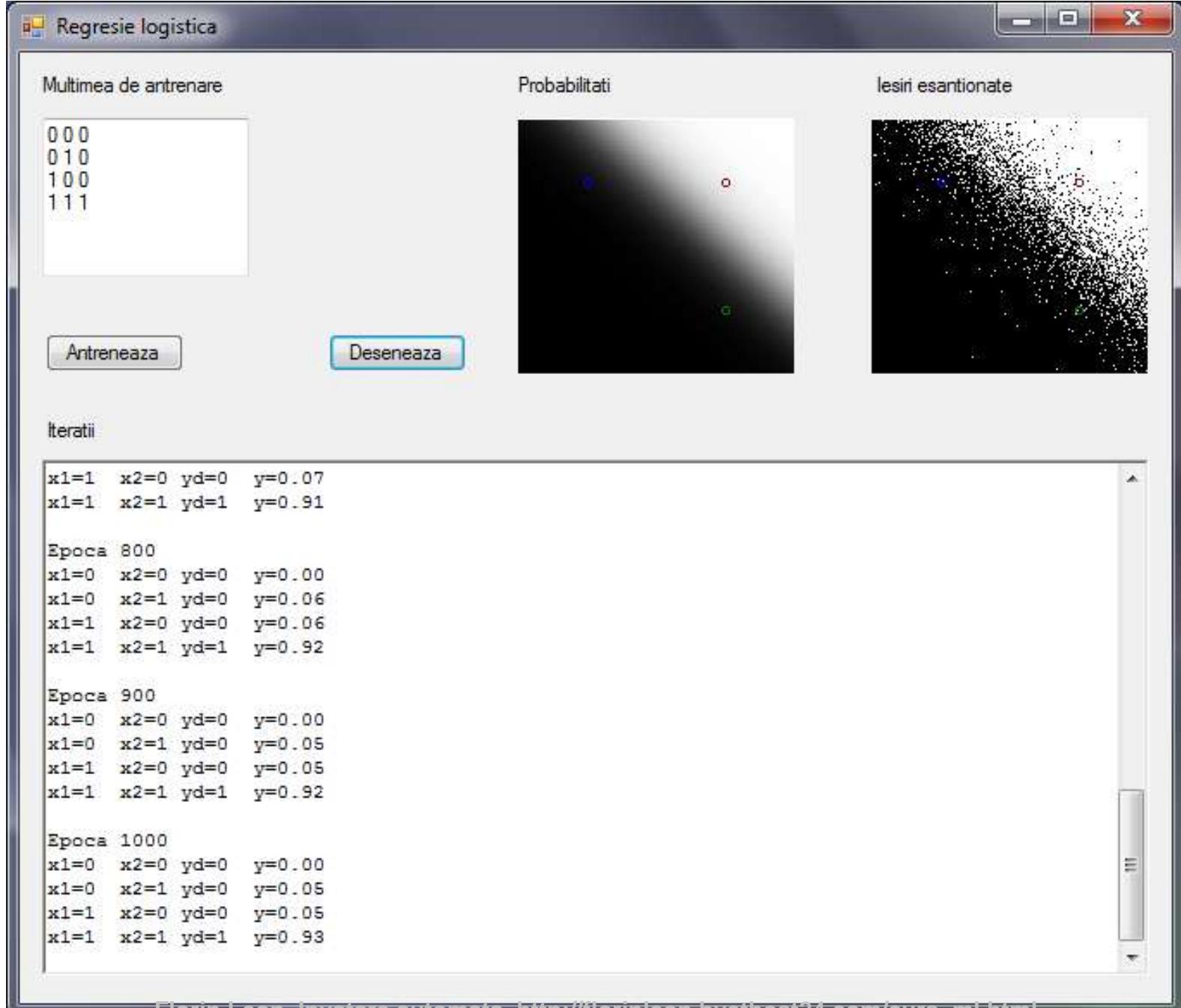
- Cu regula delta (MSE)

$$w_j = w_j - \alpha \sum_{i=1}^n x_{ij} (\hat{y}_i - y_i) \cdot \frac{(\hat{y}_i (1 - \hat{y}_i))}{\sigma'}$$

- Cu formula *cross-entropy* (MLE)

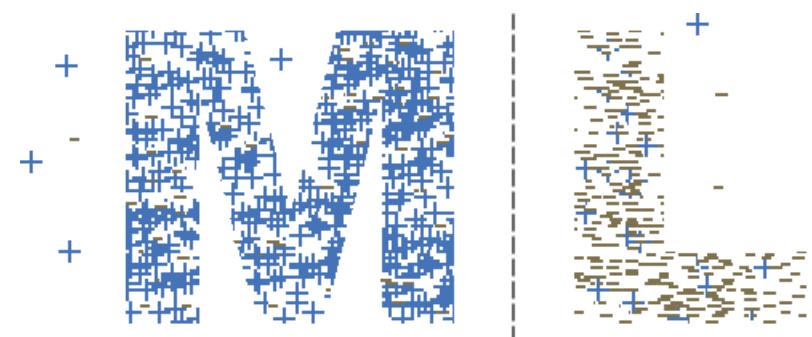
$$w_j = w_j - \alpha \sum_{i=1}^n x_{ij} (p_i - y_i)$$

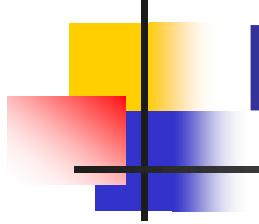




# Regresie liniară, logistică, softmax

1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
- 4. Regresia softmax**
5. Metrici de eroare





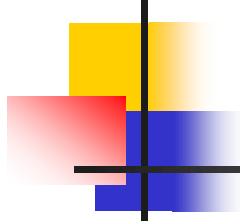
# Funcția softmax

- Funcția softmax generalizează expresia regresiei logistice la situația cu mai multe clase

$$s : \mathbb{R}^k \rightarrow \mathbb{R}^k$$

$$s(a_i) = \frac{\exp(a_i)}{\sum_{j=1}^k \exp(a_j)}$$

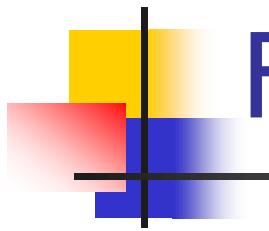
- Rezultatul funcției reprezintă probabilitățile ca o instanță să aparțină fiecărei clase



# Exemplu

- $a_1 = 2, a_2 = 0, a_3 = 5, a_4 = 4$
- Abordarea deterministă:
  - max  $\Rightarrow a_3$
- Abordarea stochastică:
  - $\exp(a_1) = 7.4, \exp(a_2) = 1, \exp(a_3) = 148.4, \exp(a_4) = 54.6$
  - suma = 211.4
  - $\Rightarrow P(a_1) = 7.4 / 211.4 = 3.5\%$
  - Analog,  $P(a_2) = 0.5\%, P(a_3) = 70.2\%, P(a_4) = 25.8\%$
- Dacă valoarea maximă este mult mai mare decât celelalte, probabilitatea să se apropie de 1

$$s(a_i) = \frac{\exp(a_i)}{\sum_{j=1}^k \exp(a_j)}$$



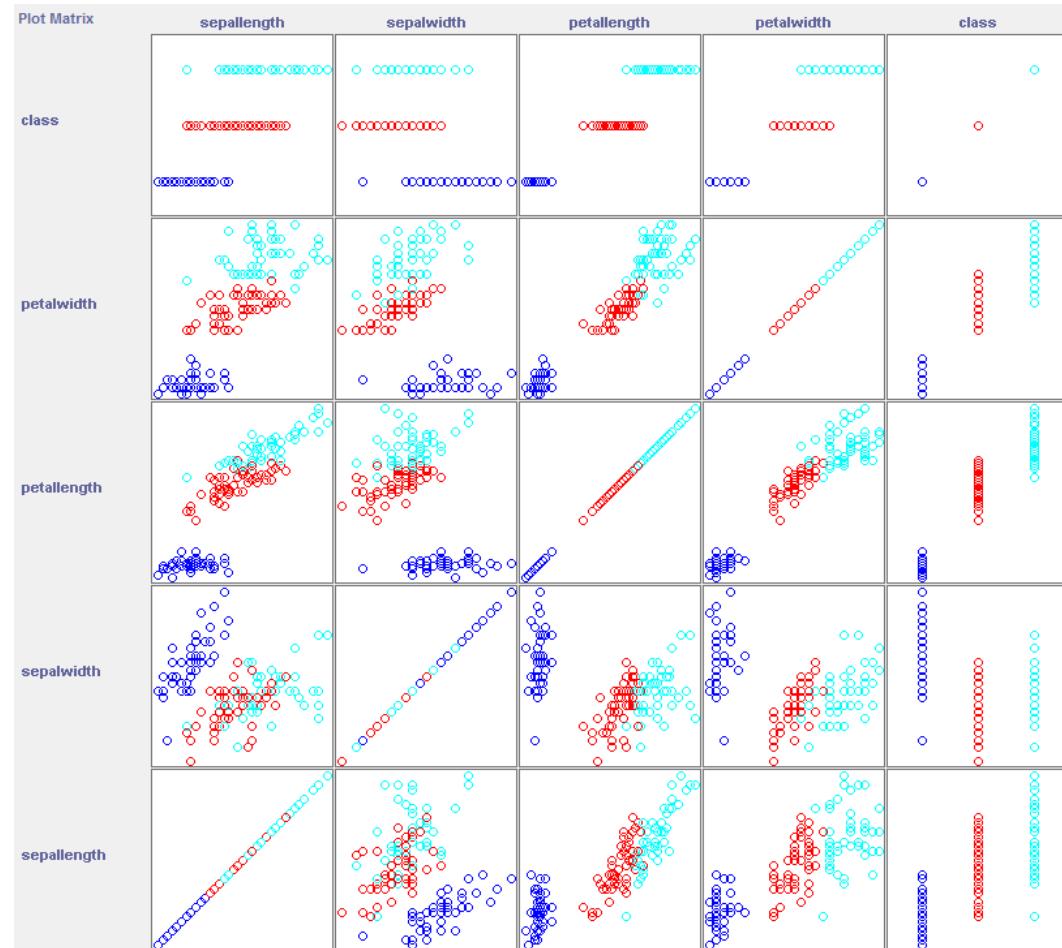
# Funcția de cost și gradienții

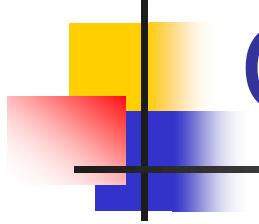
$$J(\mathbf{w}) = -\frac{1}{n} \left( \sum_{i=1}^n \sum_{j=1}^k \mathbf{1}[y_i = j] \log \frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{l=1}^k \exp(\mathbf{w}_l^T \mathbf{x}_i)} \right)$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_j} = -\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i (\mathbf{1}[y_i = j] - P(y_i = j | \mathbf{x}_i, \mathbf{w})))$$

# Exemplu: problema florilor de iris

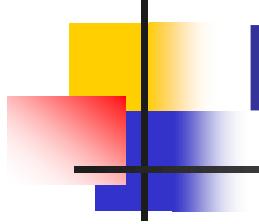
- 4 atribute: lungimea și lățimea petalelor și sepalelor
- 3 clase: setosa, versicolor, virginica
- 150 de instanțe, câte 50 din fiecare clasă





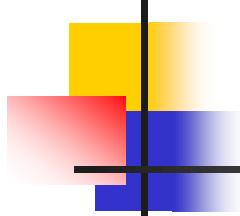
# Clasificarea cu softmax

- Multimea de antrenare:
    - 4 intrări
    - 3 ieșiri (*one-hot*, câte una pentru fiecare clasă)
- |                     |         |
|---------------------|---------|
| 5.1, 3.5, 1.4, 0.2, | 1, 0, 0 |
| 4.9, 3.0, 1.4, 0.2, | 1, 0, 0 |
| ⋮                   | ⋮       |
| 7.0, 3.2, 4.7, 1.4, | 0, 1, 0 |
| 6.4, 3.2, 4.5, 1.5, | 0, 1, 0 |
| ⋮                   | ⋮       |
| 6.3, 3.3, 6.0, 2.5, | 0, 0, 1 |
| 5.8, 2.7, 5.1, 1.9, | 0, 0, 1 |



# Notări

- Numărul de instanțe:  $n = 150$
  - Numărul de attribute (intrări):  $d = 4$
  - Numărul de clase (ieșiri):  $k = 3$
- 
- **X**[ $n, d+1$ ] (+ 1 pentru termenul liber)
  - **Y**[ $n, k$ ]
  - **W**[ $d+1, k$ ]



# Pseudocod

- Pentru un număr prestabilit de epoci se repetă următorii pași:
  - $\mathbf{Z} = \mathbf{X} \cdot \mathbf{W}$
  - Pentru  $u \in \{1, \dots, n\}$ :  $\mathbf{P}_u = \text{softmax}(\mathbf{Z}_u)$
  - Pentru  $i \in \{1, \dots, d+1\}$  și  $j \in \{1, \dots, k\}$ :
    - $$g_{ij} = -\frac{1}{n} \sum_{u=1}^n x_{ui} \cdot (y_{uj} - p_{uj})$$
    - $w_{ij} = w_{ij} - \alpha \cdot g_{ij}$

$\mathbf{g}$  sunt gradienții  
 $\alpha$  este rata de învățare

## Varianta iterativă

```
private void InitModel()
{
    string fileName = "iris.csv";
    _noInputs = 4;
    _noOutputs = 3;
    _noInstances = 150;

    _inputs = new double[_noInstances, _noInputs + 1]; // + 1 pentru prag
    _outputs = new double[_noInstances, _noOutputs];

    StreamReader sr = new StreamReader(fileName);
    for (int u = 0; u < _noInstances; u++)
    {
        string[] toks = sr.ReadLine().Split(", ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
        _inputs[u, 0] = 1;
        for (int i = 0; i < _noInputs; i++)
            _inputs[u, i + 1] = Convert.ToDouble(toks[i]);
        for (int j = 0; j < _noOutputs; j++)
            _outputs[u, j] = Convert.ToDouble(toks[_noInputs + j]);
    }
    sr.Close();

    _weights = new double[_noInputs + 1, _noOutputs];

    for (int i = 0; i < _noInputs + 1; i++)
        for (int j = 0; j < _noOutputs; j++)
            _weights[i, j] = _rand.NextDouble() / 5.0 - 0.1;
}
```

## Varianta vectorială

```
public SoftmaxRegression()
{
    string fileName = "iris.csv";

    noInputs = 4;
    noOutputs = 3;
    noInstances = 150;

    var sr = new StreamReader(fileName);
    var toks = sr.ReadToEnd().Split(", \t\r\n".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
    sr.Close();

    var data = new DenseMatrix(1, 150 * 7, toks.Select(n => Convert.ToDouble(n)).ToArray()).Reshape(150, 7);

    inputs = data.SubMatrix("all", "first 4").InsertColumn(0, DenseVector.Create(150, 1)) as DenseMatrix;
        // noInstances x (noInputs + 1), + 1 pentru prag
    outputs = data.SubMatrix("all", "last 3"); // noInstances x noOutputs

    rand = new Random();
    weights = DenseMatrix.Create(noInputs + 1, noOutputs, (x, y) => rand.NextDouble() / 5.0 - 0.1);
}
```

## Varianta iterativă

```
private double[,] Softmax(double[,] z)
{
    double[,] p = new double[_noInstances, _noOutputs];

    for (int u = 0; u < _noInstances; u++)
    {
        double sum = 0;

        for (int j = 0; j < _noOutputs; j++)
            sum += Math.Exp(z[u, j]);

        for (int j = 0; j < _noOutputs; j++)
            p[u, j] = Math.Exp(z[u, j]) / sum;
    }

    return p;
}
```

## Varianta vectorială

```
private DenseVector Softmax(DenseVector z)
{
    var z1 = z - z.Max() as DenseVector; // (z - (max z)) este o optimizare pentru a evita erorile atunci cand z este foarte mare
    return z1.Exp() / z1.Exp().Sum();
}
```



$$\frac{e^x}{e^x + e^y} = \frac{e^x}{e^x + e^y} \cdot \frac{e^{-y}}{e^{-y}} = \frac{e^{x-y}}{e^{x-y} + e^{y-y}}$$

$$x \leq y \Rightarrow e^{x-y} \leq 1$$

## Varianta iterativă

```
private void Train()
{
    int epoch = 1;
    int maxEpochs = 1000;
    _alpha = 0.1;

    while (epoch <= maxEpochs)
    {
        double[,] z = new double[_noInstances, _noOutputs];

        for (int u = 0; u < _noInstances; u++)
            for (int j = 0; j < _noOutputs; j++)
                for (int i = 0; i < _noInputs + 1; i++)
                    z[u, j] += _inputs[u, i] * _weights[i, j];

        double[,] p = Softmax(z);

        for (int i = 0; i < _noInputs + 1; i++)
            for (int j = 0; j < _noOutputs; j++)
            {
                double dw = 0;

                for (int u = 0; u < _noInstances; u++)
                    dw += _inputs[u, i] * (_outputs[u, j] - p[u, j]);

                _weights[i, j] -= _alpha * (-1.0 / _noInstances) * dw;
            }

        epoch++;
    }
    ... // programul complet este in arhiva cu materiale demonstrative
}
```

## Varianta vectorială

```
public void Train()
{
    int maxEpochs = 1000;
    double alpha = 0.1;

    p = new DenseMatrix(noInstances, noOutputs);

    for (int epoch = 0; epoch < maxEpochs; epoch++)
    {
        var z = inputs * weights; // Z = X(150x5) * W(5x3) -> 150x3

        for (int u = 0; u < noInstances; u++)
            p.SetRow(u, Softmax(z.GetRow(u)));

        var dw = inputs.Transpose() * (outputs - p) as DenseMatrix; // gradientii dw(5x3)
        weights -= alpha * (-1.0 / noInstances) * dw;

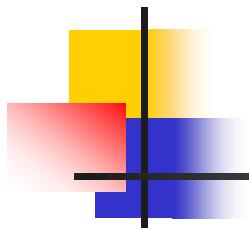
        epoch++;
    }
}
```

# Rezultate

Ieșiri model	Ieșiri dorite	Comparatie
0.991 0.009 0.000	1.000 0.000 0.000	0 - 0
0.972 0.028 0.000	1.000 0.000 0.000	0 - 0
...		
0.009 0.970 0.021	0.000 1.000 0.000	1 - 1
0.013 0.924 0.064	0.000 1.000 0.000	1 - 1
...		
0.002 0.374 0.624	0.000 1.000 0.000	2 - 1 *
...		
0.000 0.214 0.786	0.000 1.000 0.000	2 - 1 *
...		
0.000 0.004 0.996	0.000 0.000 1.000	2 - 2
0.000 0.054 0.946	0.000 0.000 1.000	2 - 2

Ponderile (o linie pentru fiecare atribut și o coloană pentru fiecare ieșire)

Setosa	Versicolor	Virginica	
0.439	0.610	-1.018	Termenul liber (pragul)
0.853	0.712	-1.627	Lungimea sepalelor
2.102	-0.168	-1.842	Lățimea sepalelor
-2.770	-0.093	2.970	Lungimea petalelor
-1.338	-1.178	2.491	Lățimea petalelor



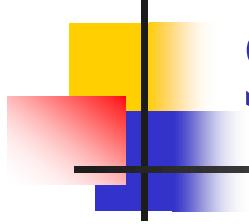
# Regularizarea

- La expresia lui  $J$  se adaugă termenul:

$$\frac{\lambda}{2} \sum_{i=0}^n \sum_{j=1}^k w_{ij}^2$$

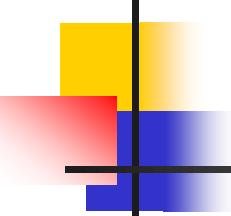
- La expresia gradientilor  $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_j}$  se adaugă termenul:

$$\lambda \mathbf{w}_j$$



# Softmax vs. $k$ clasificatori binari

- Când clasele sunt mutual exclusive, se recomandă regresia softmax
  - De exemplu: muzică clasică, country, rock, jazz
  - Pe lângă cele patru clase, se mai poate adăuga una: „alte genuri muzicale”
- Altfel, se recomandă  $k$  clasificatori binari, precum regresia logistică, iar fiecare decide dacă o instanță aparține sau nu unei clase
  - De exemplu: cu solist vocal, coloană sonoră, dance



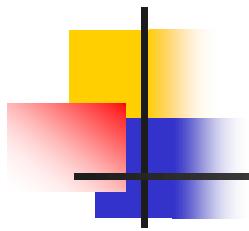
# Discuție

- Chiar dacă toate cele trei metode prezentate au în denumire cuvântul „regresie”, regresia liniară este de fapt singurul algoritm de regresie propriu-zisă, în care ieșirea este continuă
- Regresia logistică și regresia softmax sunt de fapt algoritmi de clasificare, cu două, respectiv oricâte clase
- Se poate arăta că, particularizând formulele de la regresia softmax pentru două clase, se obțin formulele de la regresia logistică

# Regresie liniară, logistică, softmax

1. Generalizarea
2. Regresia liniară. Regularizarea
3. Regresia logistică
4. Regresia softmax
5. Metrici de eroare





# Matricea de confuzii

- Fiecare element al matricei arată numărul de instanțe pentru care clasa dorită este linia și clasa prezisă de model este coloana
- Rezultate bune se obțin atunci când diagonala principală are numere mari

		Predicted Class			
		a	b	c	Total
Actual Class	a	88	10	2	100
	b	14	40	6	60
	c	18	10	12	40
	Total	120	60	20	

Aici, rata de eroare este:  
 $1 - (88+40+12) / 200 = 0.3$

# Metrici de eroare pentru modele numerice

Mean-squared error

$$\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}$$

$a_i$  sunt valorile de antrenare  
 $p_i$  sunt valorile prezise

Root mean-squared error

$$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}}$$

Mean-absolute error

$$\frac{|p_1 - a_1| + \dots + |p_n - a_n|}{n}$$

Relative-squared error\*

$$\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}$$

Root relative-squared error\*

$$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}}$$

Relative-absolute error\*

$$\frac{|p_1 - a_1| + \dots + |p_n - a_n|}{|a_1 - \bar{a}| + \dots + |a_n - \bar{a}|}$$

Correlation coefficient\*\*

$$\frac{S_{PA}}{\sqrt{S_P S_A}}, \text{ where } S_{PA} = \frac{\sum_i (p_i - \bar{p})(a_i - \bar{a})}{n-1},$$

$$S_P = \frac{\sum_i (p_i - \bar{p})^2}{n-1}, \quad S_A = \frac{\sum_i (a_i - \bar{a})^2}{n-1}$$

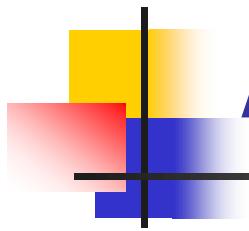
\*Here,  $\bar{a}$  is the mean value over the training data.

\*\*Here,  $\bar{a}$  is the mean value over the test data.

# Exemplu: compararea modelelor

- În figura de mai jos, se prezintă performanțele a patru modele la validare încrucișată
- Se observă că modelul *D* este cel mai bun

	A	B	C	D
Root mean-squared error	67.8	91.7	63.3	57.4
Mean absolute error	41.3	38.5	33.4	29.2
Root relative squared error	42.2%	57.2%	39.4%	35.8%
Relative absolute error	43.1%	40.1%	34.8%	30.4%
Correlation coefficient	0.88	0.88	0.89	0.91

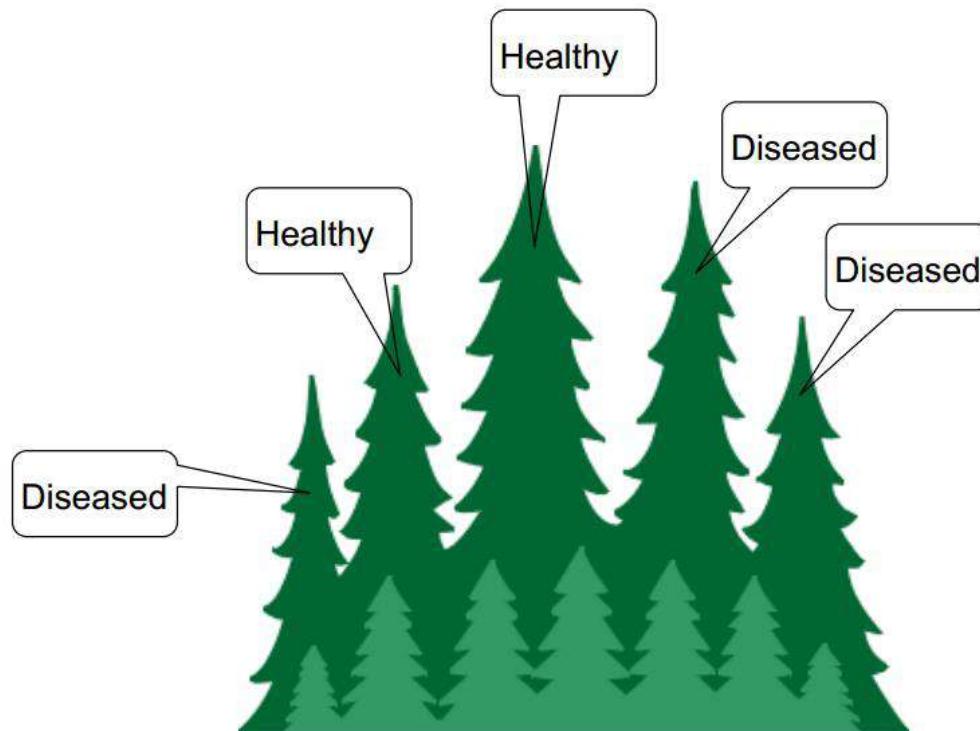


# Alte metrici

- Există foarte multe alte măsuri statistice de eroare:
  - Măsuri bazate pe  $TP$ ,  $TN$ ,  $FN$ ,  $FP$  (*true positive*, *true negative*, *false negative*, *false positive*)
  - *Precision*, *Recall*, *F-measure*
  - *Specificity*, *Sensitivity*
  - Curba ROC (*Receiver Operating Characteristic curve*)

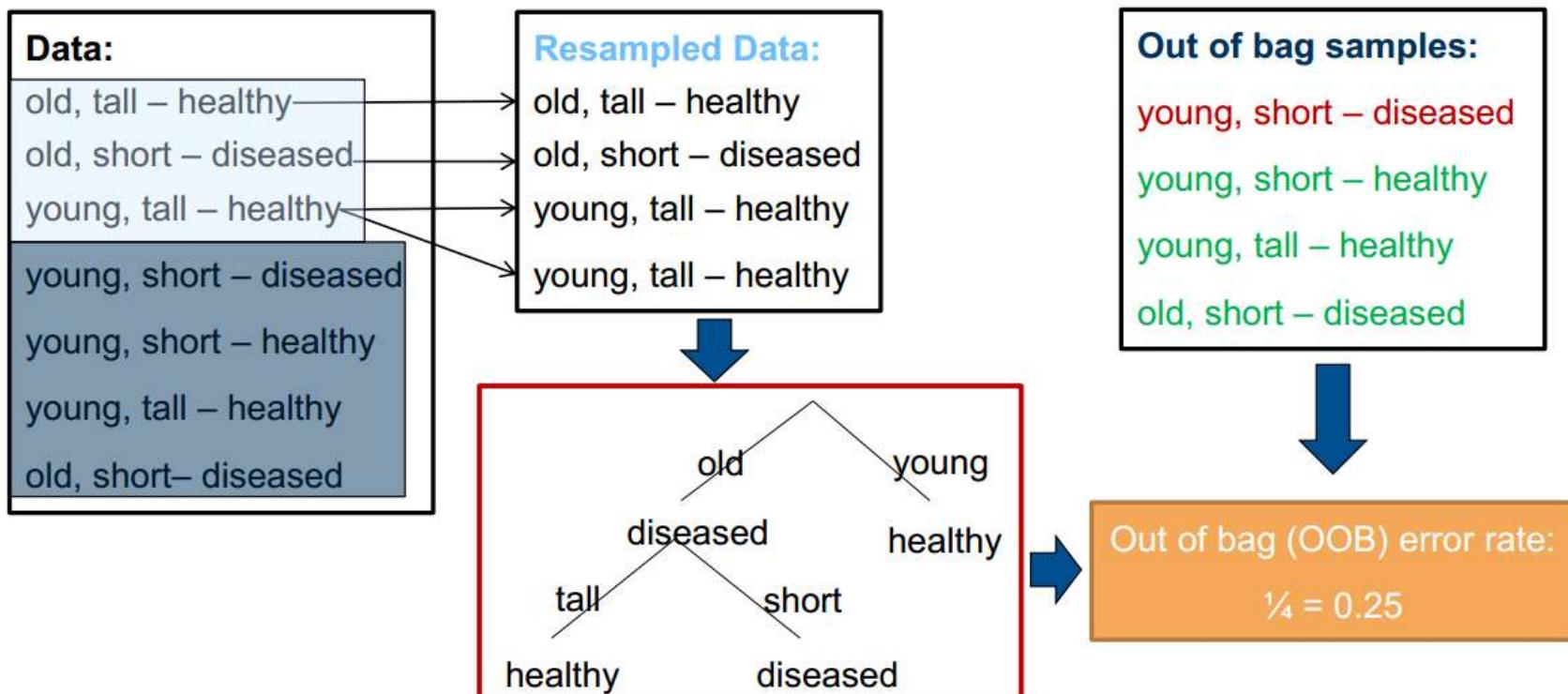
# Eroarea *out-of-bag* (pentru *bagging*)

- Reprezintă eroarea medie pentru instanțele care **nu** au fost incluse în mulțimea *bootstrapped*



# Eroarea *out-of-bag* (pentru *bagging*)

- Reprezintă eroarea medie pentru instanțele care **nu** au fost incluse în mulțimea *bootstrapped*



# Exemplu de metrici: Weka, clasificare

Iris, Naïve Bayes

==== Confusion Matrix ===

a	b	c	<-- classified as
50	0	0	a = Iris-setosa
0	48	2	b = Iris-versicolor
0	4	46	c = Iris-virginica

Correctly Classified Instances 144 96 %

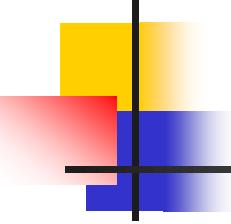
# Exemplu de metrici: Weka, regresie

1 1  
2 2  
3 4

Linear Regression Model

$$y = 1.5 * x + -0.6667$$

Correlation coefficient	0.982
Mean absolute error	0.2222
Root mean squared error	0.2357
Relative absolute error	20 %
Root relative squared error	18.8982 %
Total Number of Instances	3



# Concluzii

- Modelele învățate trebuie să aibă o bună **capacitate de generalizare**, adică să aibă performanțe bune pentru date noi, care nu au fost folosite la antrenare
- **Regresia liniară** este unul din cei mai simpli algoritmi de regresie, dar cu toate acestea este des folosit, iar metoda descrisă pentru determinarea parametrilor poate fi aplicată și pentru alte modele
- **Regresia logistică** și **regresia softmax** sunt de fapt algoritmi de clasificare stochastică, foarte des întâlniți în metodele de învățare profundă (*deep learning*)
- Există multe **măsuri de eroare**, cele mai des utilizate fiind eroarea medie pătratică și coeficientul de corelație



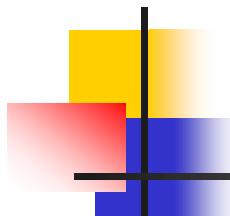
# Învățare automată

## 5. Rețele neuronale profunde

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

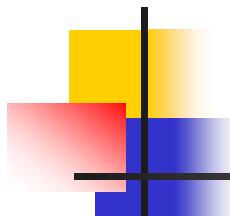
[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)



# Rețele neuronale profunde

1. Învățarea profundă: scurt istoric
2. Rețele profunde cu propagare înainte
3. Autoencodere
4. Biblioteci de învățare profundă





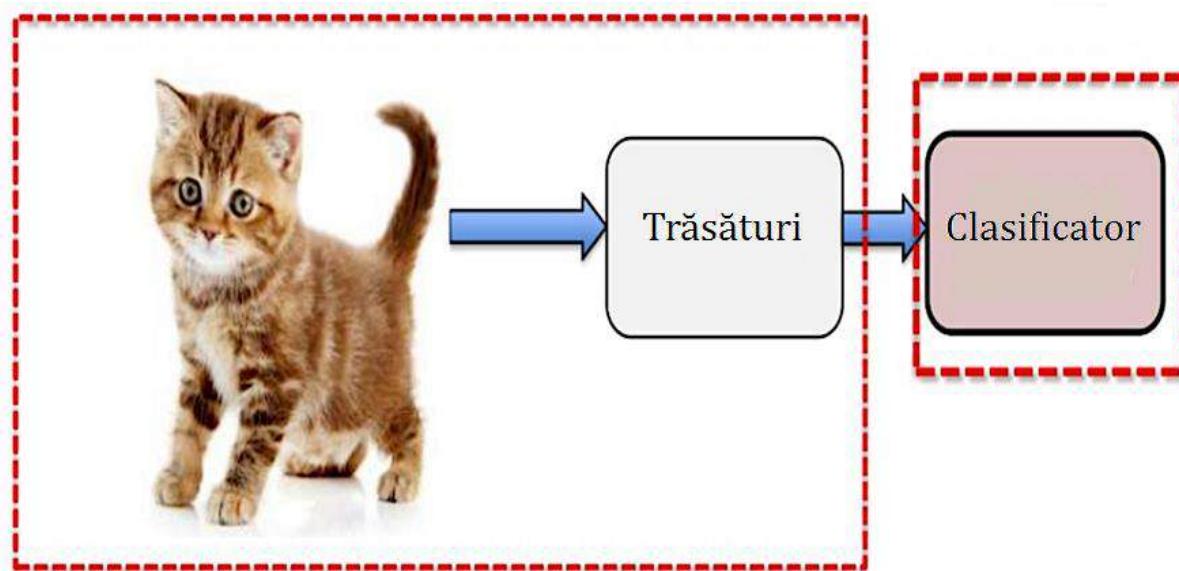
# Rețele neuronale profunde

1. Învățarea profundă: scurt istoric
2. Rețele profunde cu propagare înainte
3. Autoencodere
4. Biblioteci de învățare profundă



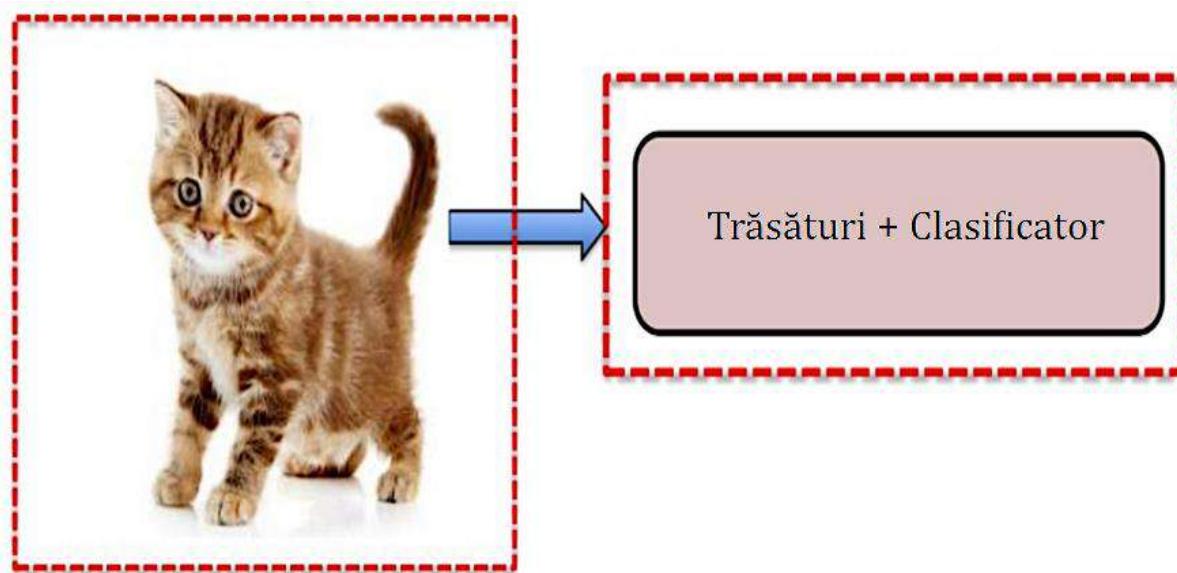
# Învățare automată tradițională

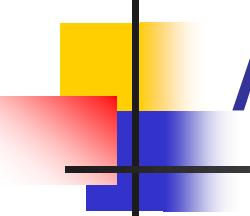
- În general, tehniciile tradiționale de învățare automată nu pot prelucra date brute
- Necesită un proces de extragere a trăsăturilor și transformare într-o reprezentare accesibilă pentru algoritmii utilizati



# Învățare profundă

- Tehnicile de învățare profundă pot acționa direct asupra datelor brute pentru a descoperi în mod automat reprezentările utile
- Proces facilitat de existența unui mare volum de date disponibile pentru antrenare: imagini, documente etc.



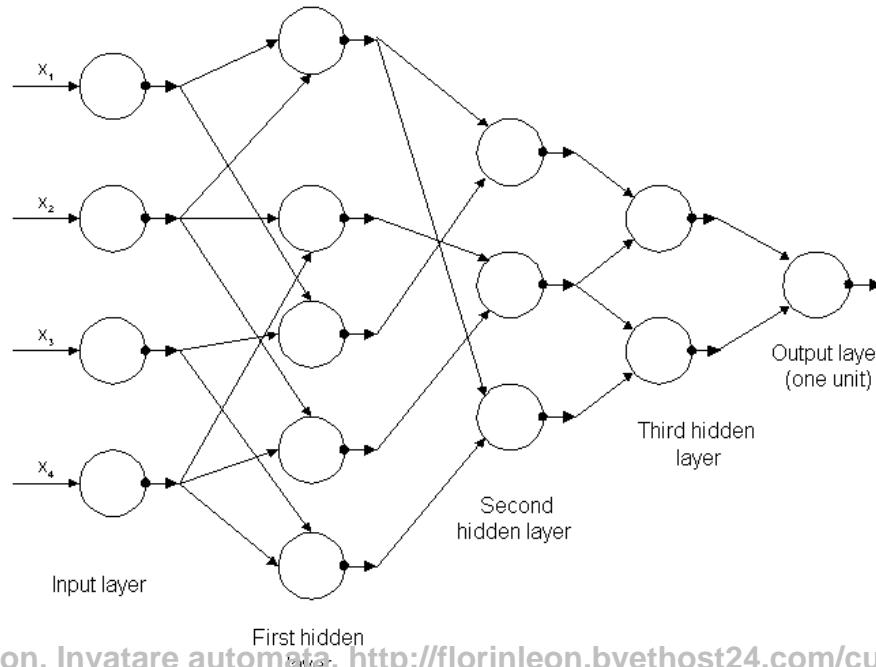


# Aplicații

- Învățarea profundă a adus rezultate superioare metodelor tradiționale în multe domenii:
  - Recunoașterea imaginilor
  - Înțelegerea limbajului natural
  - Clasificarea subiectelor, analiza sentimentelor, răspunsul la întrebări, traducerea automată
  - Analiza efectelor medicamentelor
  - Analiza exprimării genelor și a impactului asupra bolilor
  - Analiza circuitelor neuronale biologice
  - Analiza datelor din acceleratoarele de particule

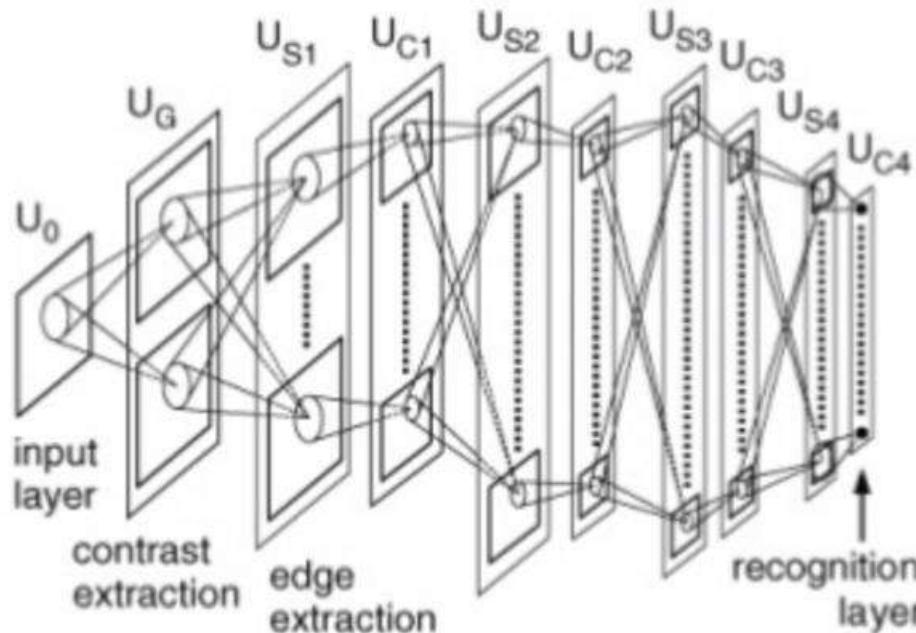
# Primele arhitecturi

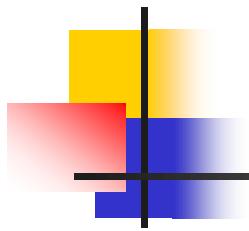
- 1965, Ivahnenko & Lapa, prima arhitectură de rețea cu mai multe straturi, dar puțini neuroni
- Funcții de activare polinomiale, antrenare cu metode statistice, nu diferențiale



# Primele arhitecturi

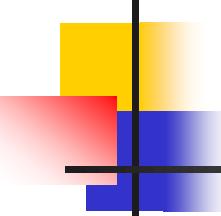
- 1980, Fukushima, neocognitronul, prima rețea de convecție, inspirat din modul de funcționare al cortexului vizual





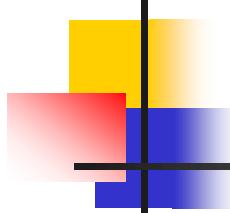
# Backpropagation

- Cauchy, 1847, metoda gradientului descendente
- Bryson & Ho, 1969
- Werbos, 1974
- Rumelhart, Hinton & Williams: *Learning representations by back-propagating errors*, 1986



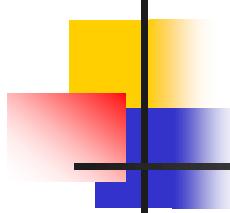
# Primele realizări „moderne”

- 1989, LeCun et al., aplicarea algoritmului *backpropagation* standard pentru recunoașterea cifrelor scrise de mână din codurile poștale
- Tot LeCun propune ideea de autoencoder pentru reducerea dimensionalității
- Hochreiter & Schmidhuber, *Long short-term memory* (LSTM), 1997, un tip de rețea recurrentă, utilizată pentru prelucrarea limbajului natural



# Impunerea Învățării profunde

- 2006, Hinton et al., mașini Boltzmann restricționate, rețele probabilistice, cu antrenare nesupervizată
- *Canadian Institute For Advanced Research, Learning in Machines & Brains*: Bengio, LeCun et al.
- Ng, 2011, proiect de învățare profundă la Google, inițial pentru comenzi vocale pe telefoanele Android și etichetarea imaginilor pe rețeaua socială Google+
- 2013, Hinton vine la Google
- 2014, Google cumpără DeepMind cu 500 de milioane de dolari



# Impunerea Învățării profunde

- Facebook folosește metode de învățare profundă pentru:
  - Înțelegerea textului (*DeepText*)
  - Recunoaștere facială (*DeepFace*, 97% rata de succes la deosebirea a două persoane după figură, oamenii au 96% rata de succes)
  - Direcționarea publicității

# Jocuri Atari

- DeepMind,  
jocuri Atari,  
învățare cu  
întărire  
profundă



# Jocuri Atari



Enduro, Atari 2600

Jucător expert: 368 puncte

Deep Learning: 661 puncte

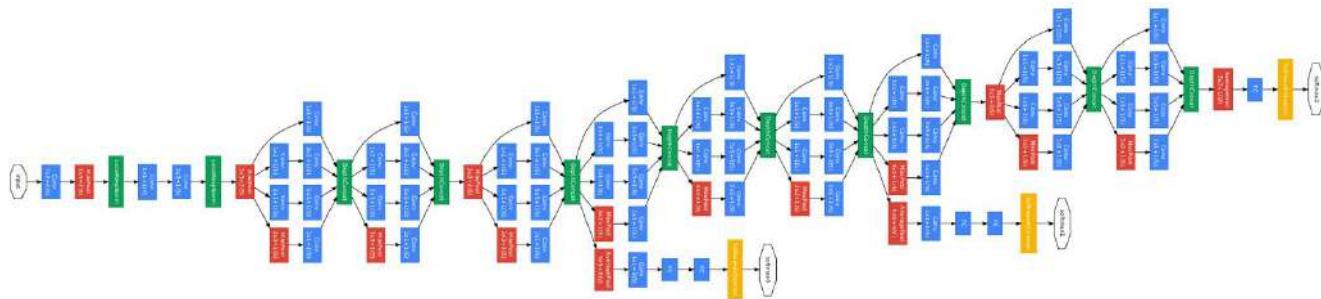
# AlphaGo

- AlphaGo nu a fost preprogramat să joace Go
- Folosește un algoritm general care interpretează modelele de joc ale adversarului



# AutoML

- Proiectarea unor rețele neuronale cu performanțe foarte bune necesită foarte mult efort
  - Exemplu: arhitectura *GoogleNet*

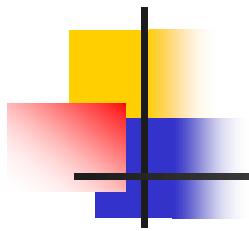


- O rețea cu 10 straturi poate avea  $10^{10}$  variante posibile
- În AutoML, o rețea controller propune o rețea-fiu, aceasta este antrenată, iar performanțele sale sunt luate în calcul la următoarele propuneri
- Controller-ul învăță în timp, după mii de propuneri, ce părți din arhitectură sunt mai promițătoare

# ImageNet

Krizhevsky et al., NIPS 2012.



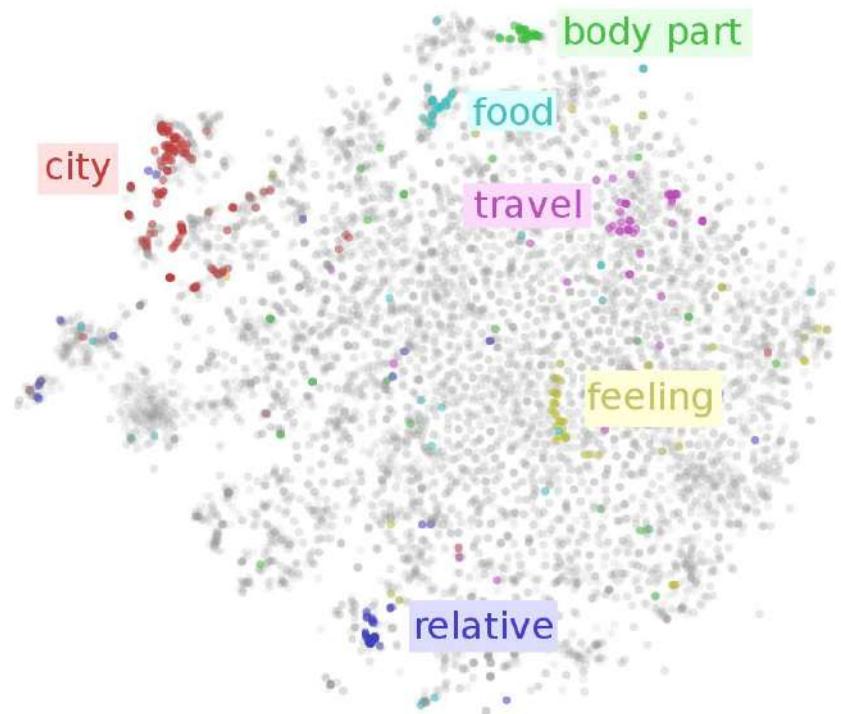


# Recunoașterea imaginilor

- *Large Scale Visual Image Challenge*, 2014:  
cea mai bună rată de eroare la recunoașterea  
imaginilor era de 5%, mai mică decât a unui  
om

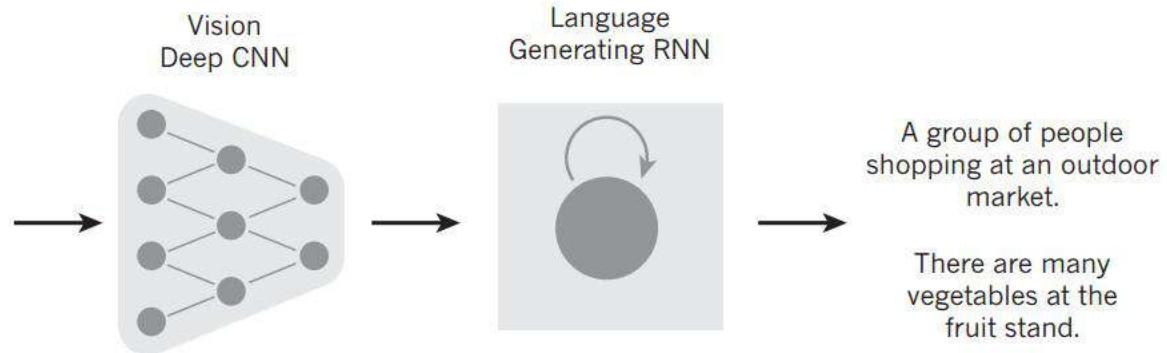
# Reprezentarea vectorială a cuvintelor

- Cuvintele apropiate ca sens se regăsesc în aceeași regiune a spațiului
- Poate fi folosită pentru traduceri între diferite limbi sau reprezentări



# Înțelegerea imaginilor

- Rețele de conoluție pentru identificarea imaginii și rețele recurente pentru generarea textului



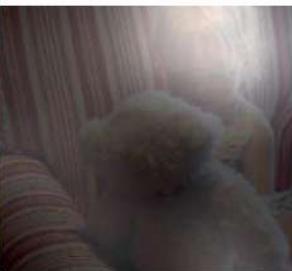
# Înțelegerea imaginilor



A **woman** is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

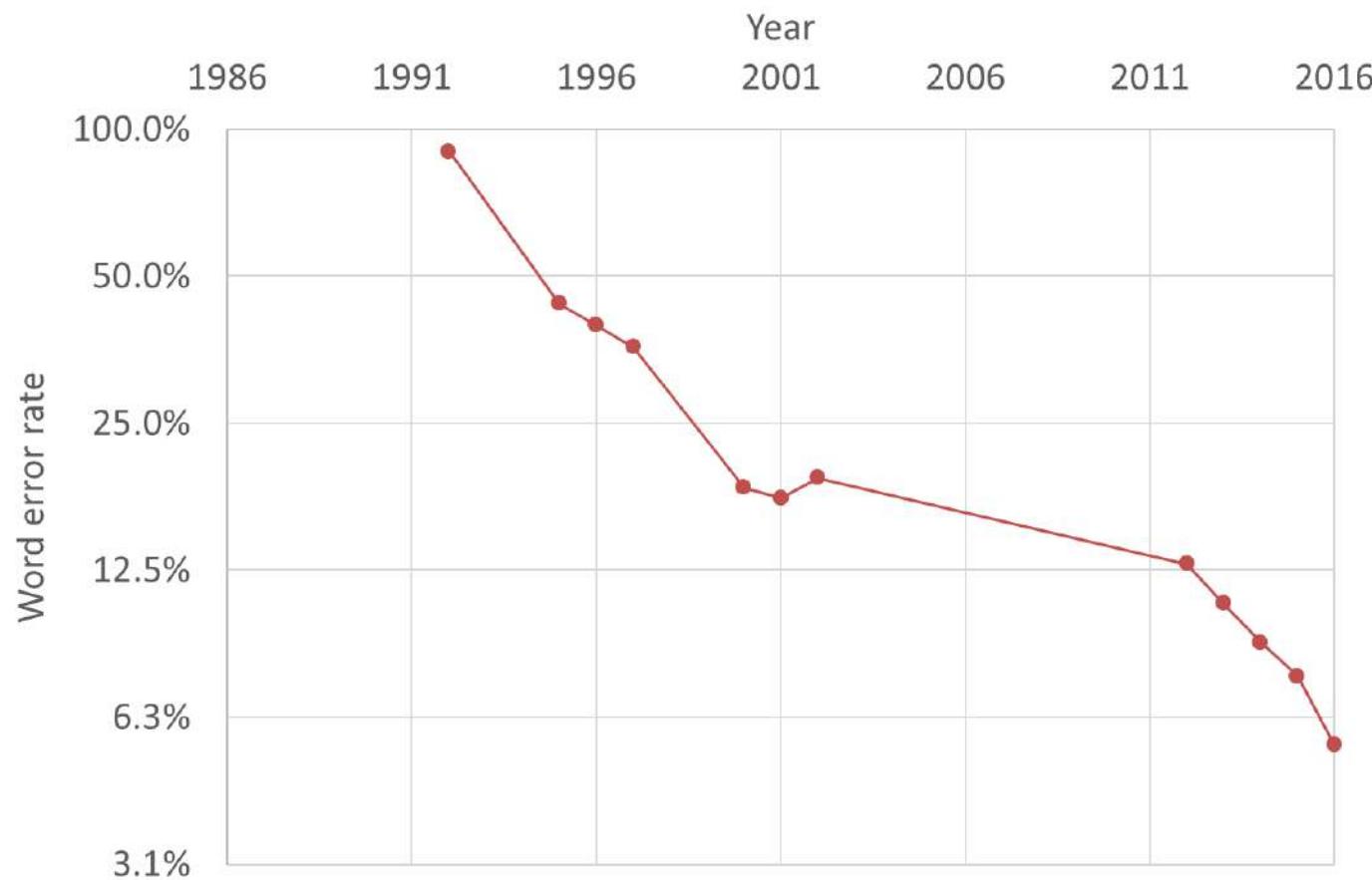


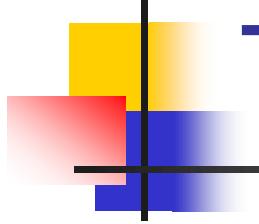
A little **girl** sitting on a bed with a **teddy bear**.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

# Recunoașterea limbajului în conversații

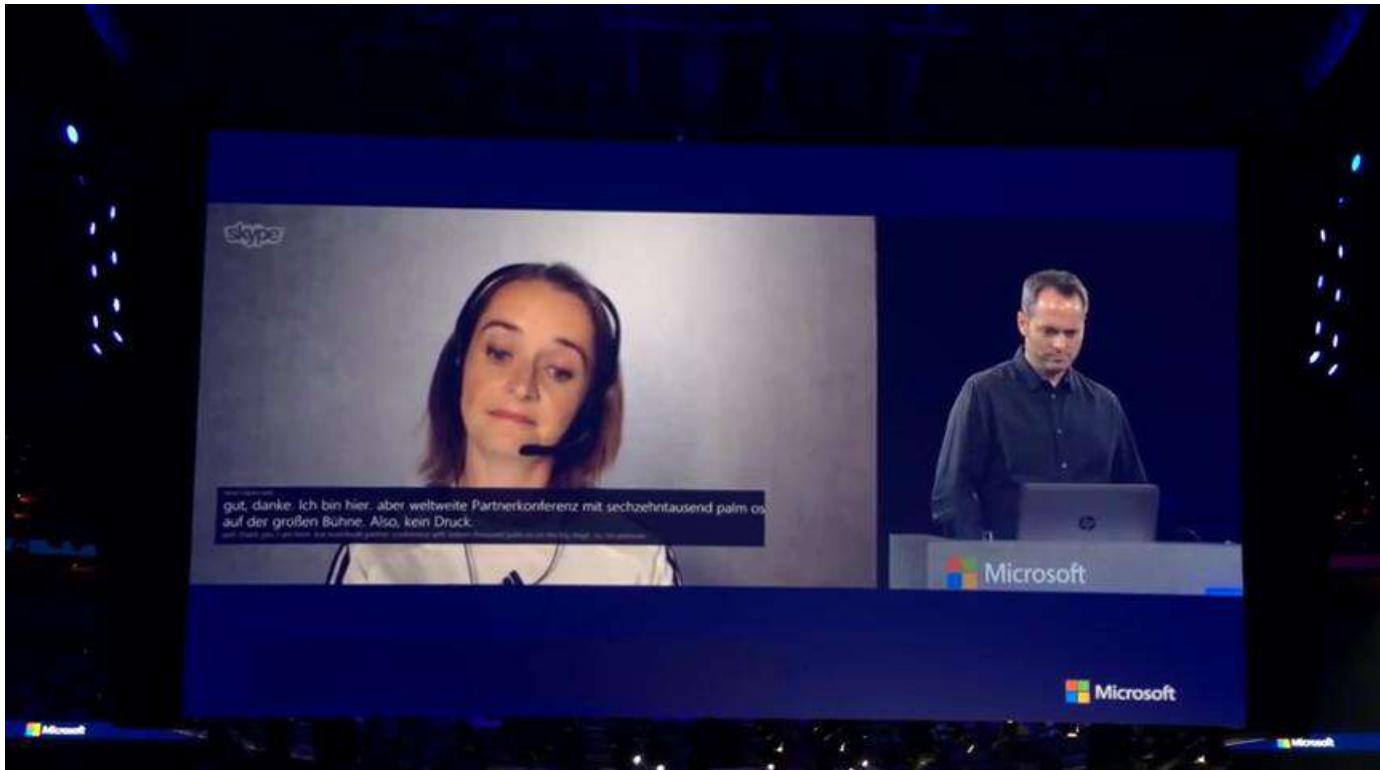




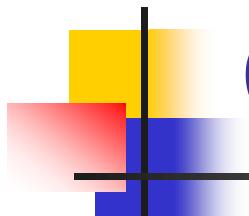
# Traduceri automate

- 2016: Folosirea rețelelor profunde pentru *Google Translate* a scăzut rata de eroare cu 60%
- 2016: *Microsoft Translator*, folosind tot învățare profundă

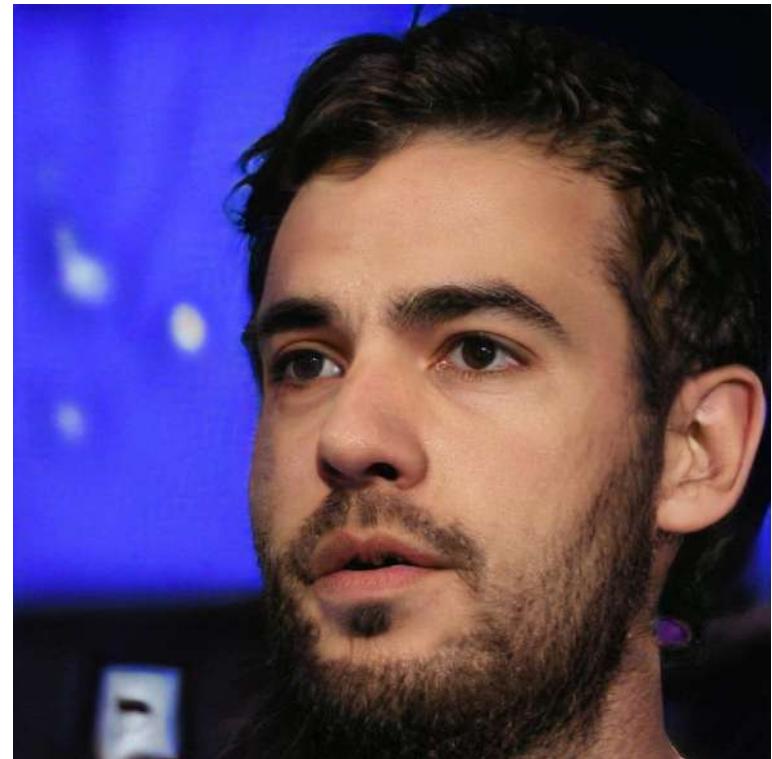
# Traducere în timp real



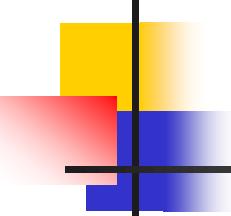
<https://www.youtube.com/watch?v=Mr2uK3cy8wE>



# Generare de imagini

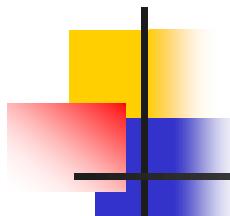


<https://thispersondoesnotexist.com> (2019)



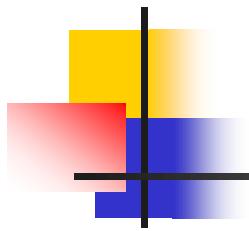
# Google DeepMind

- MuZero (2019)
  - Un algoritm superior AlphaZero
  - A învățat să joace săh, shogi, go și 57 de jocuri Atari fără să știe inițial regulile
  - Este un algoritm de învățare cu întărire bazat pe model
- Alphastar (2019)
  - A bătut 99.8% din oameni la jocul Starcraft II
- OpenAI Five (2019)
  - Jocul Dota 2: antrenare echivalentă a 10000 de ani de joc cu el însuși, a bătut echipa umană campion mondial



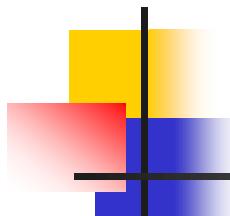
# Rețele neuronale profunde

1. Învățarea profundă: scurt istoric
2. Rețele profunde cu propagare înainte
  - 2.1. Funcții de activare (ReLU)
  - 2.2. Funcții de cost (cross-entropy)
  - 2.3. Algoritmi de antrenare (SGD, RMSProp, Adam)
  - 2.4. Inițializarea ponderilor (Xavier, Kaiming)
  - 2.5. Regularizare (Dropout)
3. Autoencodere
4. Biblioteci de învățare profundă



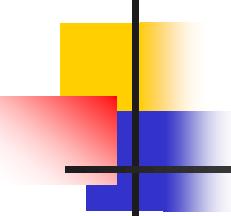
# Rețele neuronale clasice

- Pentru a înțelege conceptele prezentate în continuare, se recomandă a se revedea cursul 11 de la Inteligență artificială (Rețele neuronale I)



# Rețele clasice și rețele profunde

- Rețele clasice
  - 1-2 straturi
  - Funcții de activare sigmoide
  - Funcții de cost bazate pe MSE
  - Algoritmi de antrenare: Backpropagation, RProp, Levenberg-Marquardt etc.
- Rețele profunde
  - Mai multe straturi
  - Funcții de activare mai simple: ReLU
  - Funcții de cost bazate pe MLE
  - Algoritmi de antrenare: SGD, RMSProp, Adam etc.
  - Alte metode de inițializare a ponderilor, regularizare, pre-antrenare
- În afară de numărul de straturi, diferențele nu sunt stricte!



# Gradienti instabili

- Primele straturi ale unui perceptron multistrat clasic cu un număr mai mare de straturi ascunse nu se antrenează bine
- Când ponderile sunt mici sau funcțiile de activare sigmoide sunt saturate (gradienti mici), corecțiile ponderilor  $\Delta w$  sunt mici, iar antrenarea este foarte lentă
- De asemenea, ultimele straturi, cele apropiate de ieșire, pot învăța problema „destul de bine” și deci semnalul de eroare trimis către primele straturi devine și mai mic
- Sunt necesare alte metode de antrenare pentru a pune în valoare primele straturi

# Functia de activare ReLU

- *ReLU (Rectified Linear Unit)*

$$f(x) = \max(0, x)$$

- *Leaky ReLU*

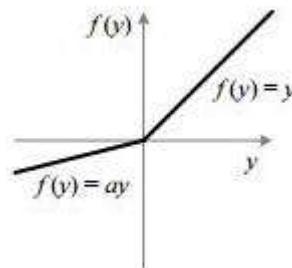
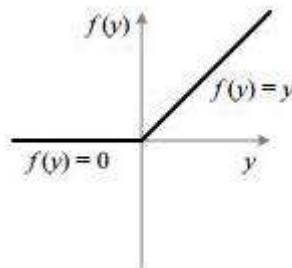
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

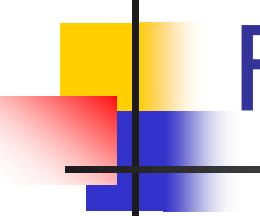
- *Parametric ReLU (PReLU)*

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$



a poate fi învățat





# Funcția de activare ReLU

- ReLU este funcția de activare recomandată pentru majoritatea rețelelor profunde
- Este neliniară
  - O combinație liniară de funcții liniare este tot o funcție liniară
  - O rețea cu funcții de activare liniare este echivalentă cu un perceptron cu un singur strat
- Este ușor de optimizat prin metode diferențiale
  - Chiar dacă în partea negativă gradientul este 0, în practică funcționează bine
  - S-a observat că învățarea este de circa 6 ori mai rapidă față de funcțiile sigmoide

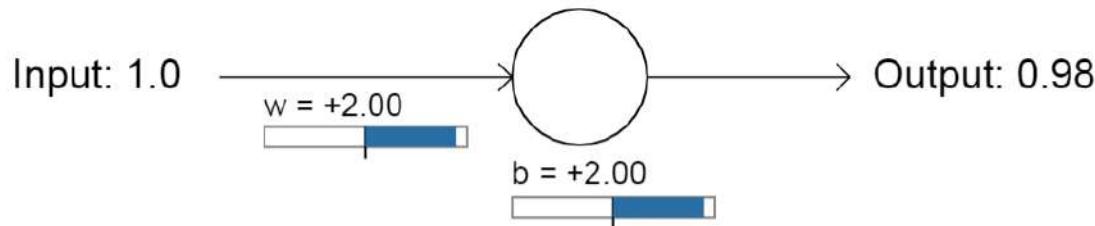
$$\frac{df}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

# Functia de cost

- Exemplu: un neuron care aproximează funcția *NOT* (primește intrarea 1 și trebuie să producă ieșirea 0)

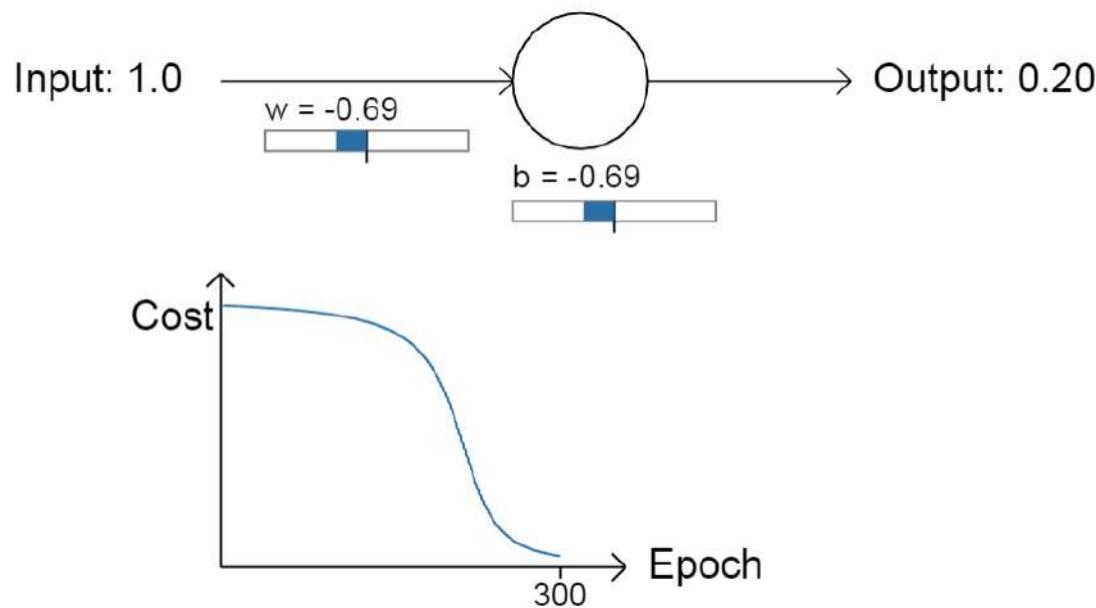


- Starea inițială



# Antrenarea

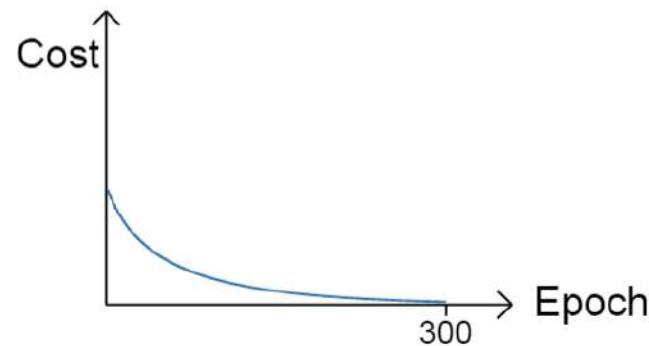
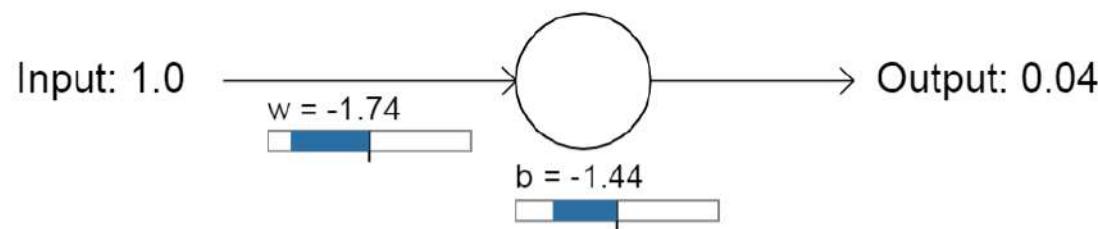
- Cu funcția de cost MSE



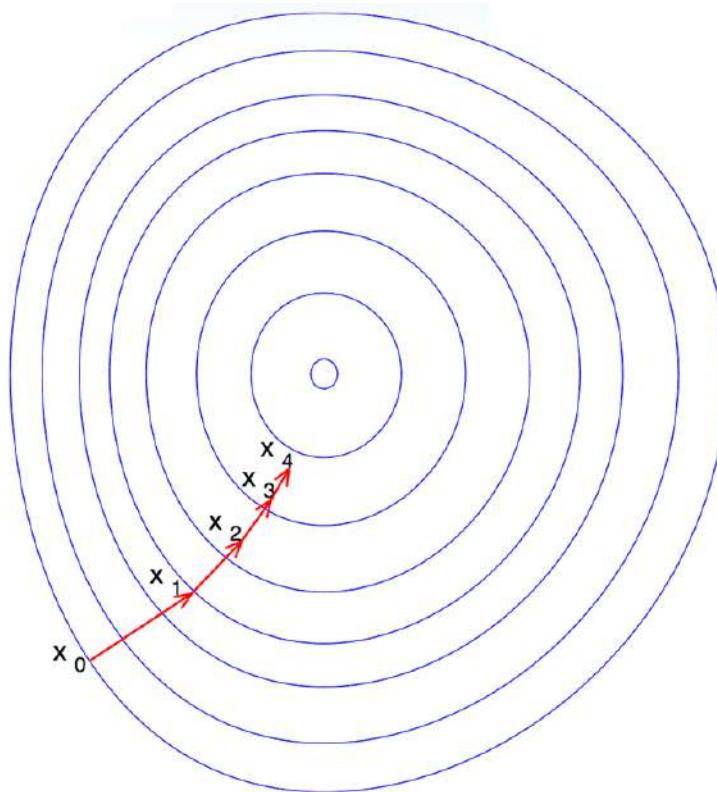
# Antrenarea

$$w_j = w_j - \alpha \sum_{i=1}^n x_{ij} (p_i - y_i)$$

- Cu funcția de cost *cross-entropy* (cursul 4)



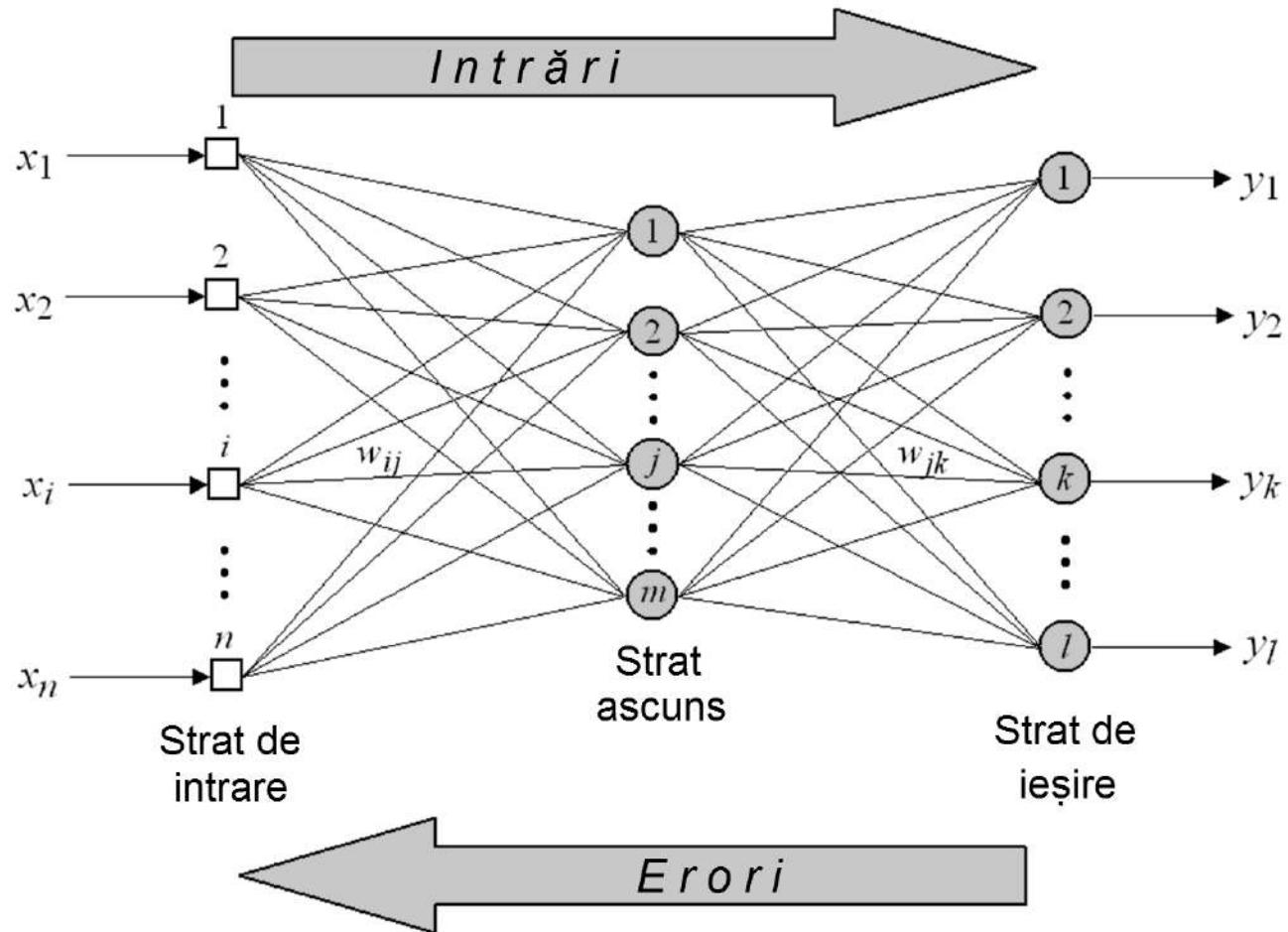
# Gradientul descendente

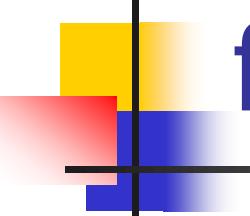


$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \nabla F(\mathbf{x}_n)$$

$\alpha_n$  este rata de învățare

# Backpropagation





# *Backpropagation standard cu funcție de activare sigmoidă*

$$o_j = \varphi(\text{net}_j) = \varphi \left( \sum_{k=1}^n w_{kj} o_k \right)$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{l \in L} \delta_l w_{jl})o_j(1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

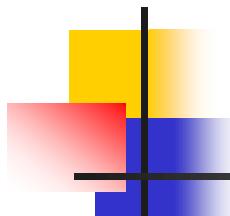
$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} = \begin{cases} -\alpha o_i(o_j - t_j)o_j(1 - o_j) & \text{if } j \text{ is an output neuron,} \\ -\alpha o_i(\sum_{l \in L} \delta_l w_{jl})o_j(1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

$o_j$  – ieșirea neuronului  $j$

$o_i$  – ieșirea neuronului  $i$  = intrarea neuronului  $j$

$t_j$  – ieșirea dorită pentru neuronul  $j$

$\varphi$  – funcția de activare



# Gradient Descent

- Este ideea algoritmului *backpropagation*
- Pentru clasificare se folosesc acum funcții de cost bazate pe MLE

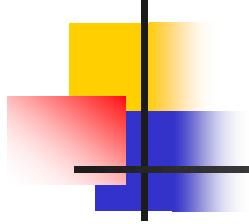
$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\log p(y | \mathbf{x}; \boldsymbol{\theta}).$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

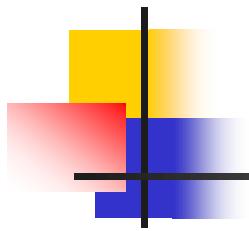
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g,$$

- $J$  – funcția obiectiv,  $\boldsymbol{\theta}$  – parametrii (ponderile),  $g$  – gradienții,  $\epsilon$  – rata de învățare



# Stochastic Gradient Descent

- Pentru problemele actuale, numărul instanțelor de antrenare poate fi foarte mare
- SGD lucrează asemănător cu gradientul DESCENDENT, dar nu ia în calcul toate instanțele la calcularea gradientilor, ci doar un **mini-lot** (*minibatch*)
- La fiecare iterație, se aleg aleatoriu alte instanțe
- Gradientii sunt doar o APPROXIMAȚIE a gradientilor reali
- COMPLEXITATEA de timp poate fi CONTROLATĂ prin dimensiunea mini-lotului



# Stochastic Gradient Descent

---

**Algorithm**      Stochastic gradient descent (SGD) update at training iteration  $k$

---

**Require:** Learning rate  $\epsilon_k$

**Require:** Initial parameter  $\theta$

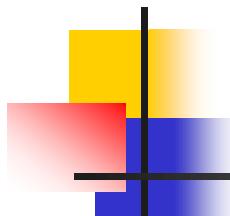
**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient estimate:  $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Apply update:  $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

**end while**



# Rata de învățare

- Pentru a asigura convergență:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty,$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

- În practică, până la iterată  $\tau$ :

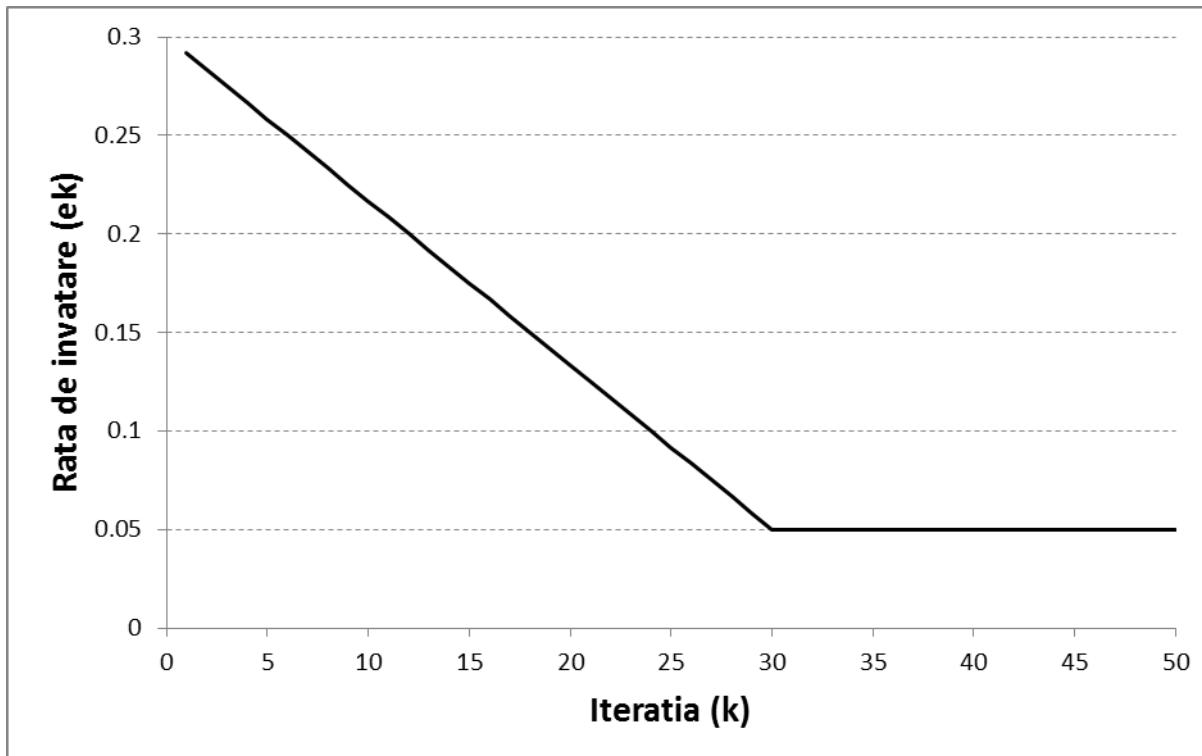
$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_{\tau} \quad \alpha = \frac{k}{\tau}$$

iar după iterată  $\tau$ ,  $\epsilon_k$  rămâne constant

- $\epsilon_0, \epsilon_{\tau}, \tau$  sunt parametri aleși de utilizator

# Exemplu

- $\varepsilon_0 = 0.3, \varepsilon_\tau = 0.05, \tau = 30$



# Momentul (inertia)

- Accelerează învățarea bazată pe gradient descendent
- Parametrul „viteză” acumulează gradientii:

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_{\theta} \left( \frac{1}{m} \sum_{i=1}^m L(\mathbf{f}(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \right)$$

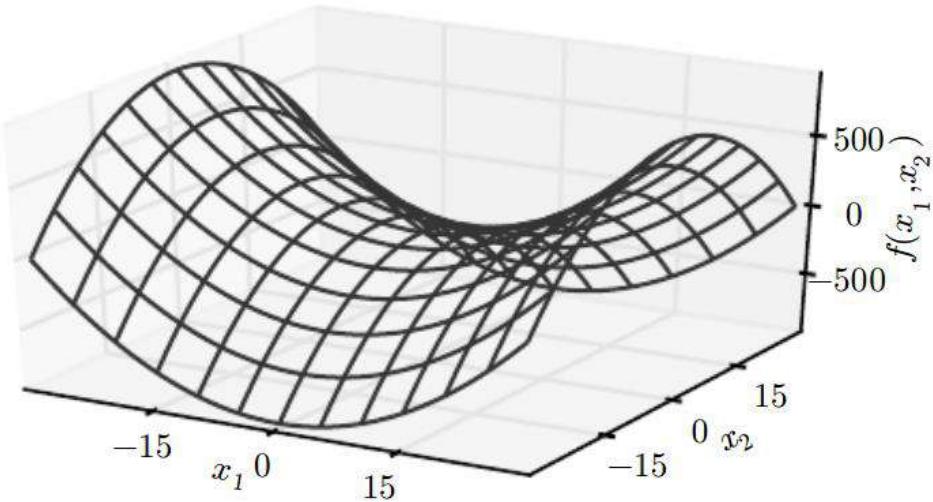
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}.$$

- Algoritmul accelerează în direcția  $-\mathbf{g}$ , până când dimensiunea unui pas este:

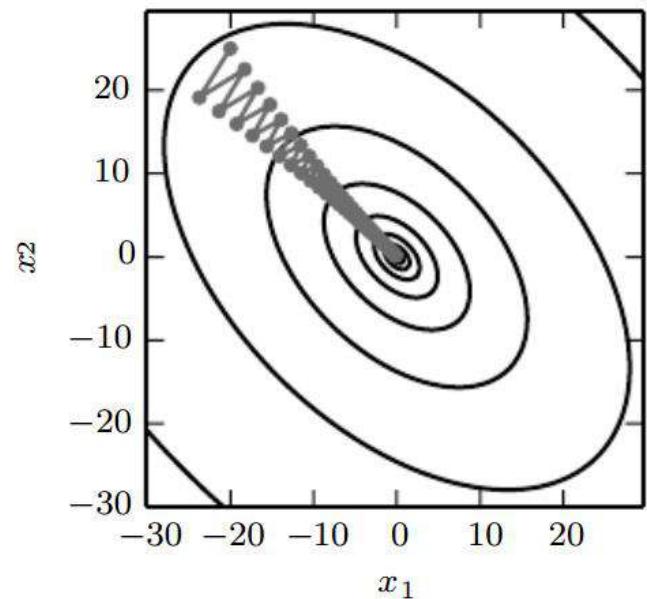
$$\frac{\epsilon \|\mathbf{g}\|}{1 - \alpha}.$$

- De exemplu,  $\alpha = 0.9$  înseamnă o viteză maximă de 10 ori mai mare decât în cazul gradientului descendent simplu

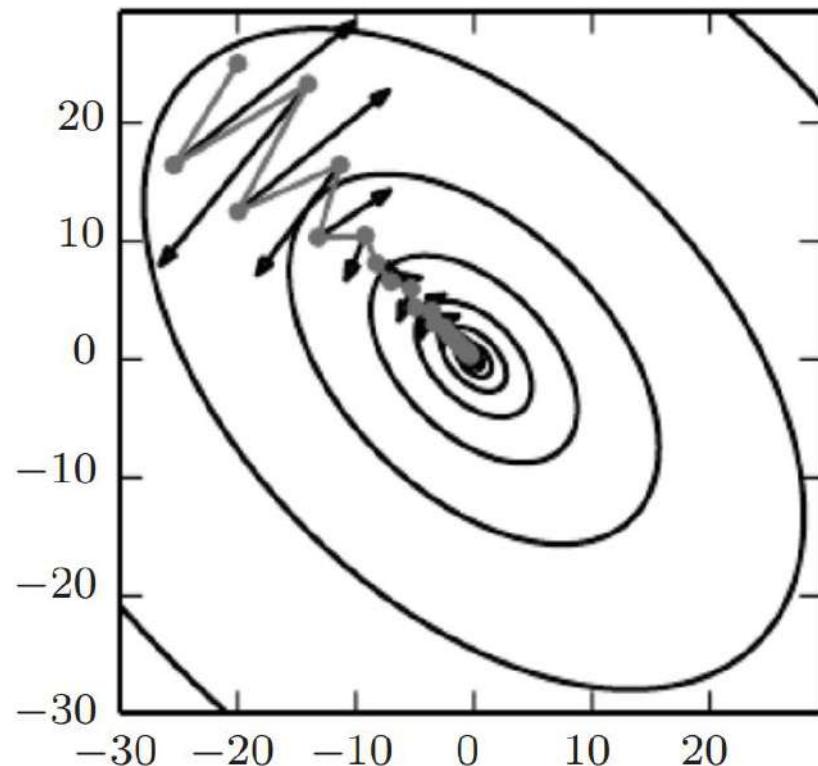
# Exemplu: GD



$$f(\mathbf{x}) = x_1^2 - x_2^2$$



# Exemplu: GD cu moment



cu gri: GD+M

cu negru: ce pas ar face GD

## Algorithm      Stochastic gradient descent (SGD) with momentum

---

**Require:** Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .

**Require:** Initial parameter  $\theta$ , initial velocity  $v$ .

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

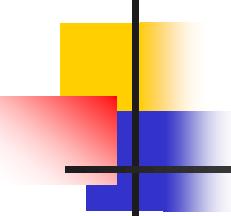
    Compute gradient estimate:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Compute velocity update:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

    Apply update:  $\theta \leftarrow \theta + \mathbf{v}$

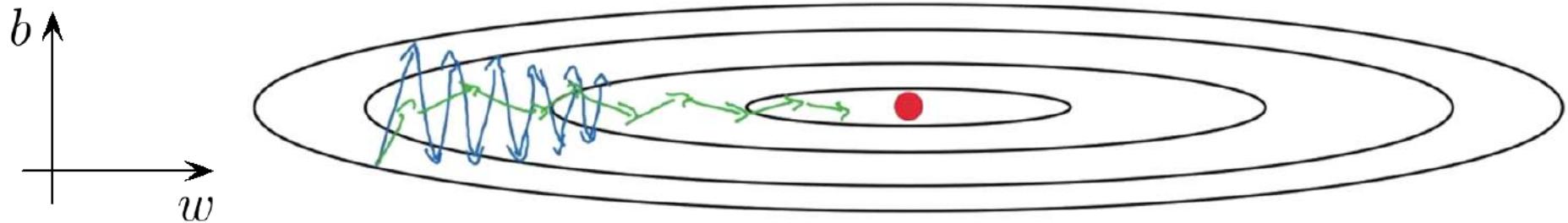
**end while**

Demonstrație și explicații suplimentare: <https://distill.pub/2017/momentum/>



# RMSProp

- *Root Mean Square Propagation*
- Se bazează pe ideea adaptării ratei de învățare invers proporțional cu suma pătratelor gradientilor
- Rata de învățare scade rapid pentru parametrii cu gradienti mari și mai încet pentru cei cu gradienti mici
- Nu se folosește direct suma istorică, ci o medie glisantă (*moving average*), care elimină efectele trecutului îndepărtat
- Altfel, prin traversarea unor zone neconvexe ale spațiului problemei, rata de învățare ar putea deveni prea mică înainte ca algoritmul să ajungă într-o zonă convexă local, unde se găsește soluția
- La fel ca metoda momentului, are ca efect atenuarea oscilațiilor și permite mărirea ratei de învățare, ceea ce accelerează învățarea



Problema: gradienții sunt prea mari în direcția  $b$  și prea mici în direcția  $w$

$$r_{dw} = \rho \cdot r_{dw} + (1 - \rho) \cdot dw^2$$

$$r_{db} = \rho \cdot r_{db} + (1 - \rho) \cdot db^2$$

$$\underline{w = w - \epsilon \cdot \frac{1}{\sqrt{r_{dw} + \delta}} \cdot dw}$$

$$\underline{b = b - \epsilon \cdot \frac{1}{\sqrt{r_{db} + \delta}} \cdot db}$$

Notări:

$dw, db$  = gradienții

$r$  = acumularea gradienților pătratici

$\rho$  = rata de descompunere (*decay rate*)

$\epsilon$  = rata de învățare

$\delta$  = o constantă mică (de exemplu,  $10^{-6}$ )

$dw$  este mic,  $db$  este mare

$r_{dw}$  este mai mic,  $r_{db}$  este mai mare

$w$  se modifică mai mult, iar  $b$  mai puțin

---

**Algorithm**      The RMSProp algorithm

---

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$ .

**Require:** Initial parameter  $\theta$

**Require:** Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers.

Initialize accumulation variables  $r = 0$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

    Compute parameter update:  $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{\delta + r}}$  applied element-wise)

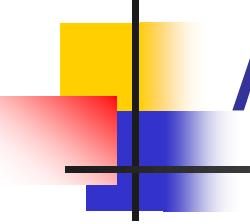
    Apply update:  $\theta \leftarrow \theta + \Delta \theta$

**end while**

---

$\mathbf{g} \odot \mathbf{g}$  este produsul element cu element

$$\mathbf{a} \odot \mathbf{b} = (a_1, \dots, a_n) \odot (b_1, \dots, b_n) = (a_1 \cdot b_1, \dots, a_n \cdot b_n)$$



# Adam

- *Adaptive Moment*
- Poate fi văzut ca o combinație între metoda momentului și algoritmul RMSProp
- Adaugă niște corecții termenilor calculați pentru a scădea subiectivitatea (cursul 3)
- În general, funcționează bine cu valorile implicate ale parametrilor
- Uneori trebuie ajustată rata de învățare

## Algorithm The Adam algorithm

---

**Require:** Step size  $\epsilon$  (Suggested default: 0.001)

**Require:** Exponential decay rates for moment estimates,  $\rho_1$  and  $\rho_2$  in  $[0, 1]$ .  
(Suggested defaults: 0.9 and 0.999 respectively)

**Require:** Small constant  $\delta$  used for numerical stabilization. (Suggested default:  
 $10^{-8}$ )

**Require:** Initial parameters  $\theta$

Initialize 1st and 2nd moment variables  $s = 0, r = 0$

Initialize time step  $t = 0$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with  
    corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

    Update biased first moment estimate:  $s \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$

    Update biased second moment estimate:  $r \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

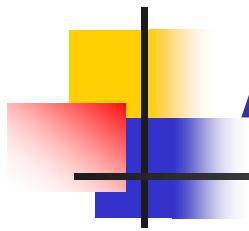
    Correct bias in first moment:  $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$

    Correct bias in second moment:  $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

    Compute update:  $\Delta \theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$  (operations applied element-wise)

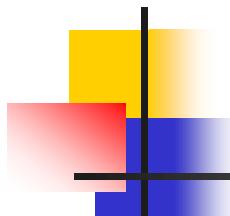
    Apply update:  $\theta \leftarrow \theta + \Delta \theta$

**end while**



# Alți algoritmi de antrenare

- RMSProp cu moment Nesterov
- AdaDelta
- AdaGrad
- AdaMax
- Nadam (Nesterov-Adam)



# Inițializarea ponderilor

- Ideea de bază: ponderile trebuie să aibă media 0 și o variantă specificată

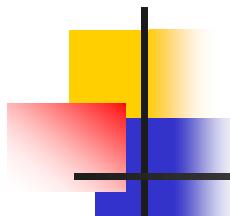
$$\text{Var}(X) = \mathbb{E}[(X - \mu)^2]$$

- De exemplu, pentru distribuția normală

$$\text{Var}(X) = \sigma^2$$

- La propagarea înainte într-o rețea profundă, se dorește ca varianta semnalului să rămână constantă

$$\frac{\text{Var}(\mathbf{w}^T \mathbf{x})}{\text{Var}(X)} = \frac{\sum_n^N \text{Var}(w_n x_n)}{\text{Var}(X)} = \frac{n \text{Var}(W) \text{Var}(X)}{\text{Var}(X)} = n \text{Var}(W) = 1$$



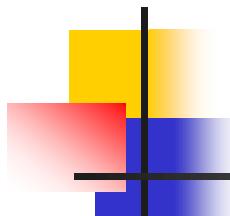
# Inițializarea ponderilor

## ■ Inițializarea Xavier (Glorot)

- $n_i$  este numărul de neuroni cu care este conectat la intrare neuronul considerat (*fan-in*), care are ponderile  $W_i$ , iar  $n_{i+1}$  este numărul de neuroni cu care este conectat la ieșire (*fan-out*)
- $N$  reprezintă distribuția normală, iar  $U$  cea uniformă

$$W_i \sim N\left(0, \sqrt{\frac{2}{n_i + n_{i+1}}}\right)$$

$$W_i \sim U\left(-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}\right)$$

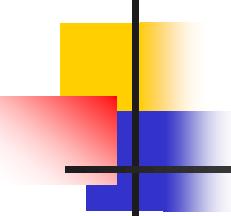


# Inițializarea ponderilor

## ■ Inițializarea Kaiming (He)

$$W_i \sim N \left( 0, \sqrt{\frac{2}{(1 + a^2) n_i}} \right)$$

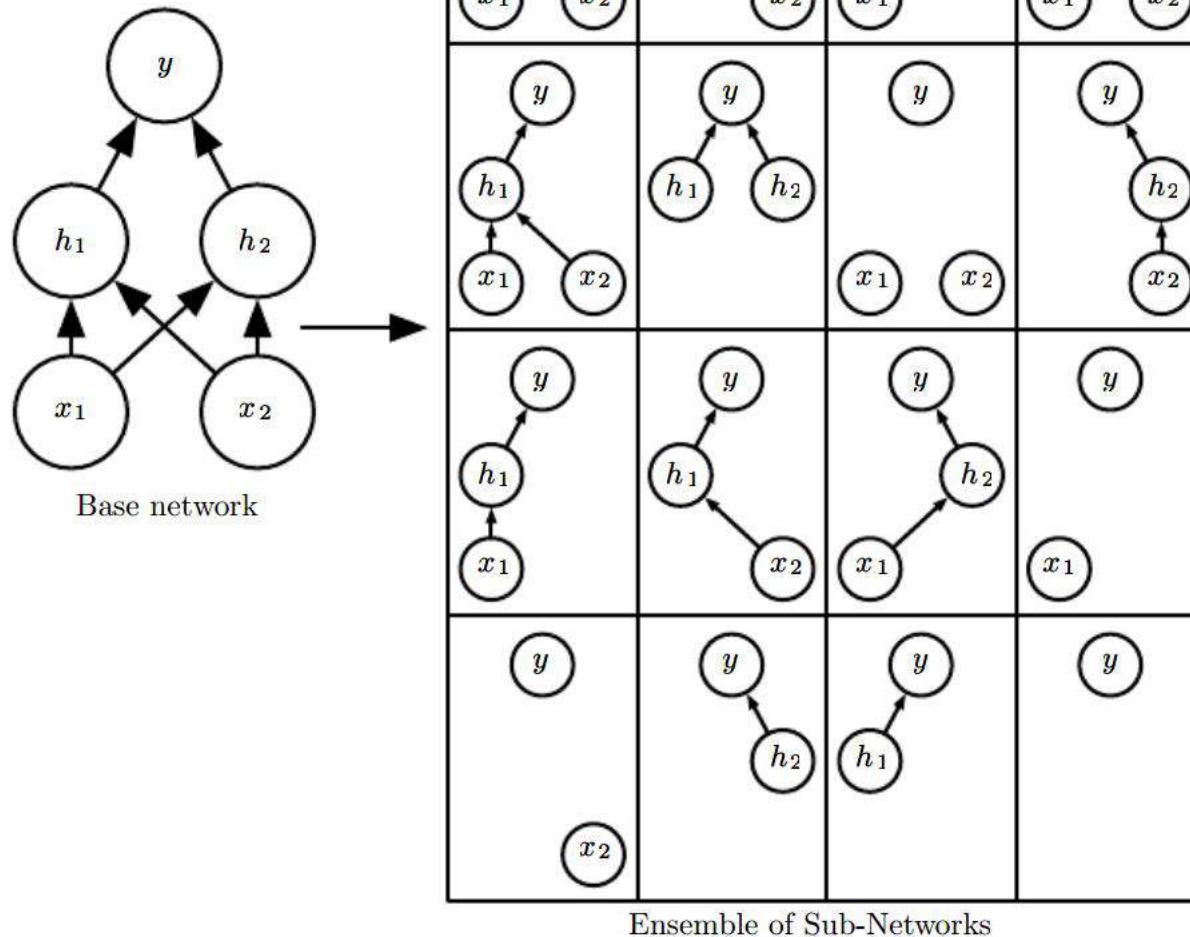
- Pentru funcții de activare de tip ReLU
- $a$  este parametrul portiunii negative din PReLU
- Pentru ReLU simplu,  $a = 0$

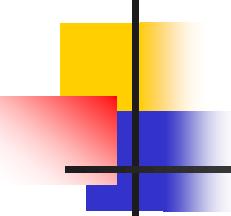


# Dropout

- *Dropout* este o formă de regularizare
- Lucrează asemănător cu ideea de la *bagging*
- Simulează antrenarea unor rețele diferite prin dezactivarea aleatorie a unor neuroni în timpul antrenării
- Aceste subrețele sunt parte a rețelei originare, iar ponderile sunt comune
- *Dropout-ul previne suprapotrivirea (overfitting)*
- În mod ideal, fiecare neuron din rețea ar trebui să detecteze trăsături în mod independent. Dacă mai mulți neuroni detectează aceleasi trăsături în mod repetat, fenomenul se numește **co-adaptare**
- *Dropout-ul previne și co-adaptarea*

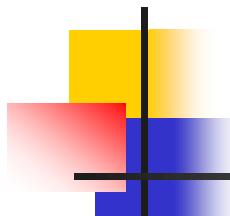
# Dropout





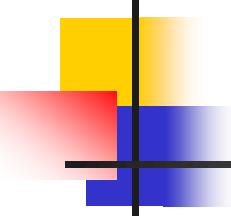
# Implementare

- Din punct de vedere practic, la fiecare iterație din timpul antrenării, se generează o mască de biți care indică includerea sau excluderea unui neuron din rețea
- Probabilitățile cu care se generează masca reprezintă un hiper-parametru
- De obicei, un neuron de intrare este păstrat cu probabilitatea 0.8, iar un neuron ascuns, cu probabilitatea 0.5
- *Dropout-ul* se aplică doar în faza de propagare înainte, după care ajustarea ponderilor se face în mod normal



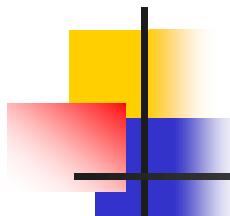
# Rescalarea ponderilor

- Dacă unii neuroni sunt excluși, intrările neuronilor sunt mai mici, iar ponderile se măresc ca să compenseze
- După antrenare, ponderile trebuie scalate pentru ca rețeaua să funcționeze corect pentru predicție, când toți neuronii sunt activi
- Ponderile se înmulțesc cu rata de *dropout*
  - De exemplu, dacă  $p = 0.5$ , ponderile trebuie împărțite la 2



# Discuție

- S-a observat că *dropout*-ul conduce la performanțe mai bune decât alte tipuri de regularizare, precum scăderea ponderilor (*weight decay*)
- Eliminarea unor neuroni de intrare are efecte foarte puternice. În lipsa unor intrări, rețeaua trebuie să compenseze prin găsirea altor trăsături importante pentru clasificare
- De exemplu, pentru recunoașterea unei fețe, dacă din imagine se elimină nasul, rețeaua trebuie să fie capabilă să găsească alte indicii, de exemplu ochii sau gura



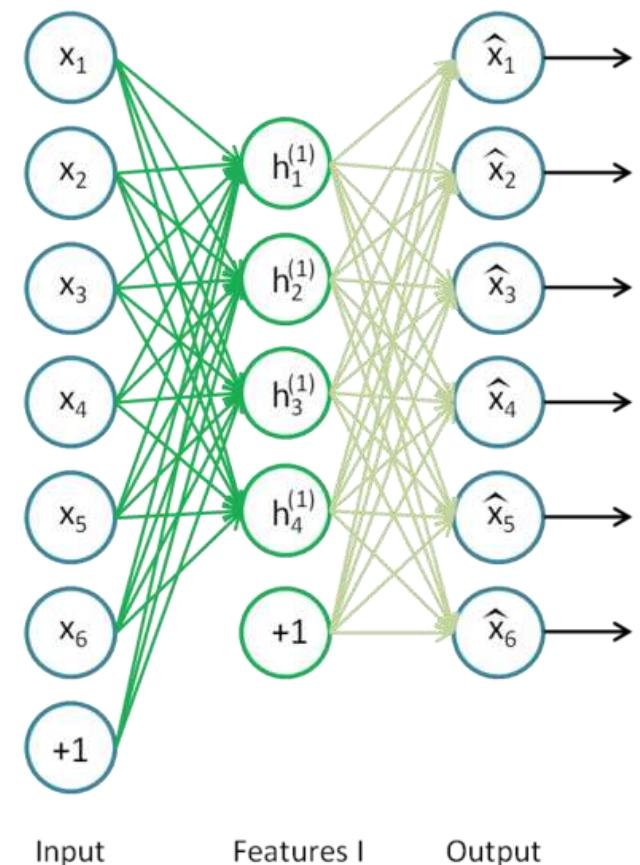
# Rețele neuronale profunde

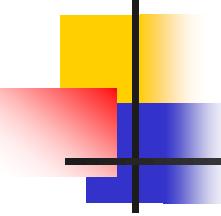
1. Învățarea profundă: scurt istoric
2. Rețele profunde cu propagare înainte
3. Autoencodere
  - 3.1. *Denoising autoencoders*
  - 3.2. *Stacked denoising autoencoders*
  - 3.3. *Variational autoencoders*
4. Biblioteci de învățare profundă

# Autoencodere

- Un **autoencoder** învață funcția identitate:  $h(\mathbf{x}) = \mathbf{x}$

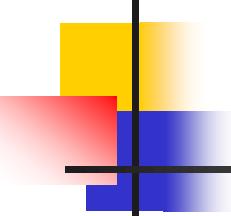
Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001





# Autoencodere

- Autoencodere subcomplete (*undercomplete*)
  - $|\mathbf{h}| < |\mathbf{x}|$
  - Stratul ascuns este o **versiune comprimată** a intrării
  - Reprezintă o metodă de reducere a dimensionalității problemei
  - De fapt, majoritatea modelelor de învățare presupun reținerea trăsăturilor esențiale
- Autoencodere supracomplete (*overcomplete*)
  - $|\mathbf{h}| > |\mathbf{x}|$
  - Creșterea dimensionalității este un proces asemănător cu aplicarea unui nucleu la SVM
  - Necesită regularizare, pentru a nu copia pur și simplu intrarea



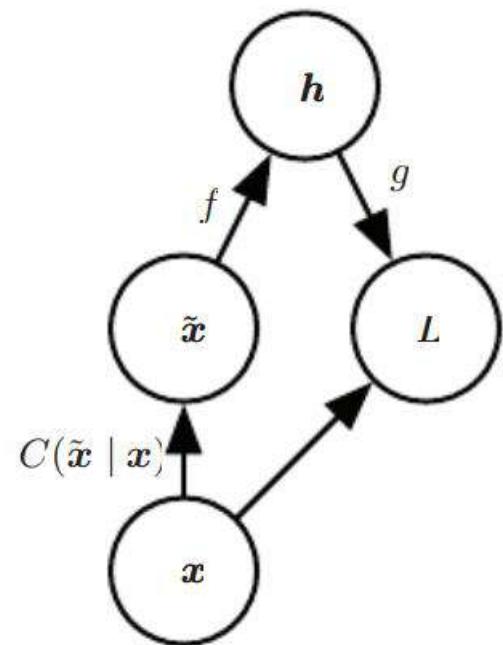
# Denoising Autoencoders

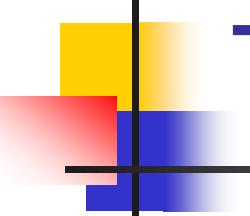
- O reprezentare bună:
  - Este robustă în prezența zgomotului
  - Se poate obține din intrări afectate de zgomot
  - Poate fi folosită pentru a corecta acele intrări
- *Denoising autoencoders* sunt antrenate cu intrări corupte, pe care încearcă să le corecteze (*de-noise*)



# Procedura

- Intrările ( $x$ ) sunt corupte ( $\tilde{x}$ )
- Intrările corupte intră în autoencoder ( $f$ )
- Funcția de cost ( $L$ ) se calculează prin compararea ieșirii autoencoderului ( $g$ ) cu intrările necorupte ( $x$ )





# Tipuri de zgomot

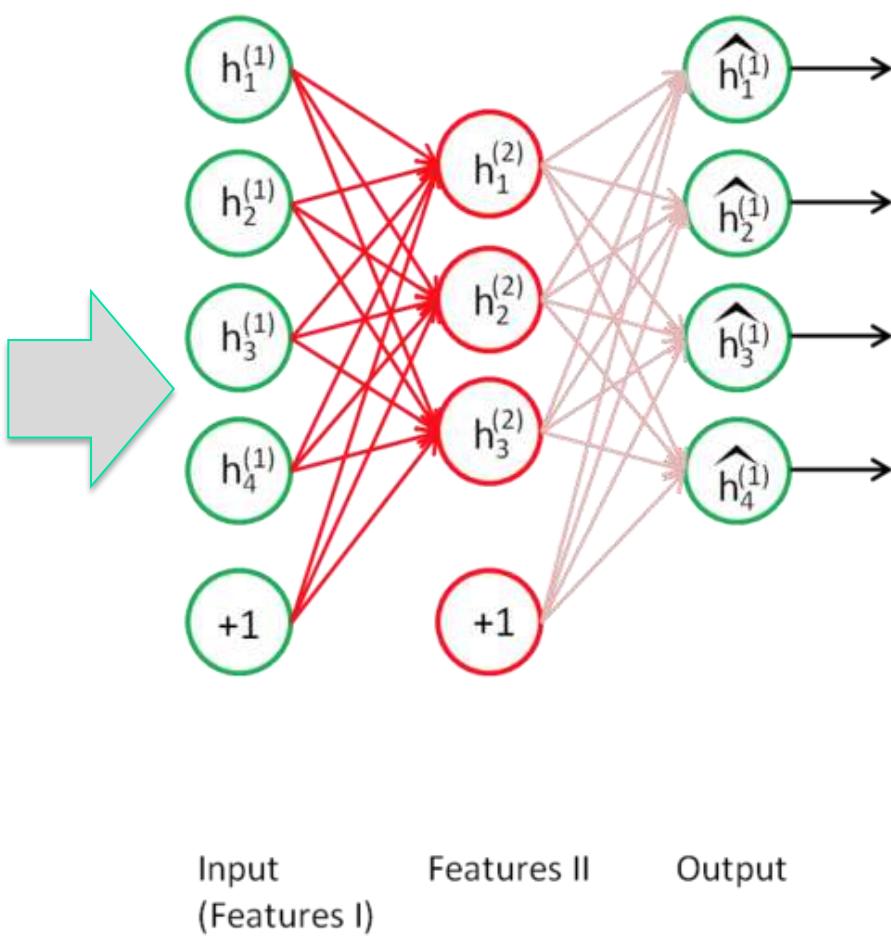
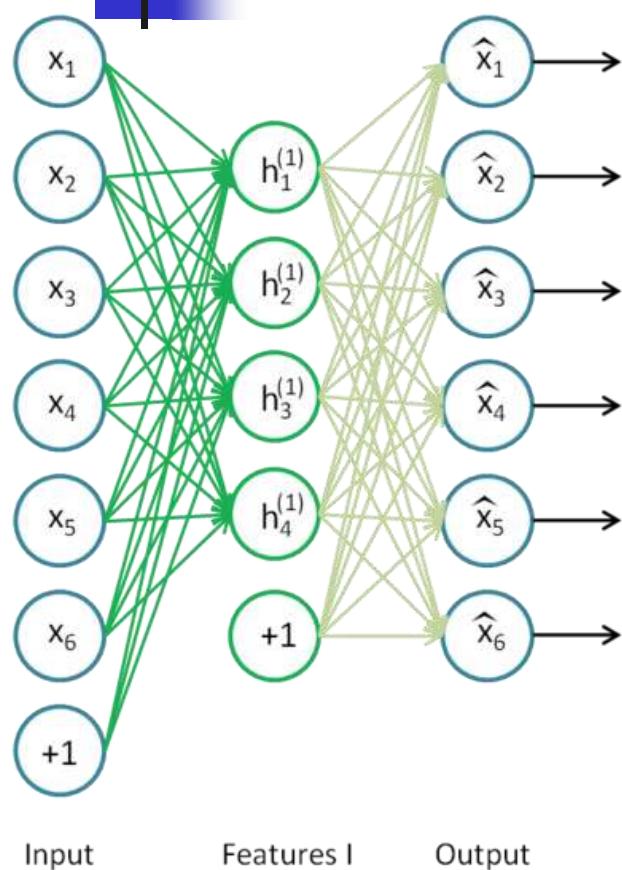
## ■ Zgomot de mascare

- O fracțiune din intrări, alese la întâmplare pentru fiecare instanță, sunt puse pe 0
- Fracțiunea de intrări anulate este de obicei 25%
- Este cel mai folosit tip de zgomot

## ■ Zgomot de tip sare și piper

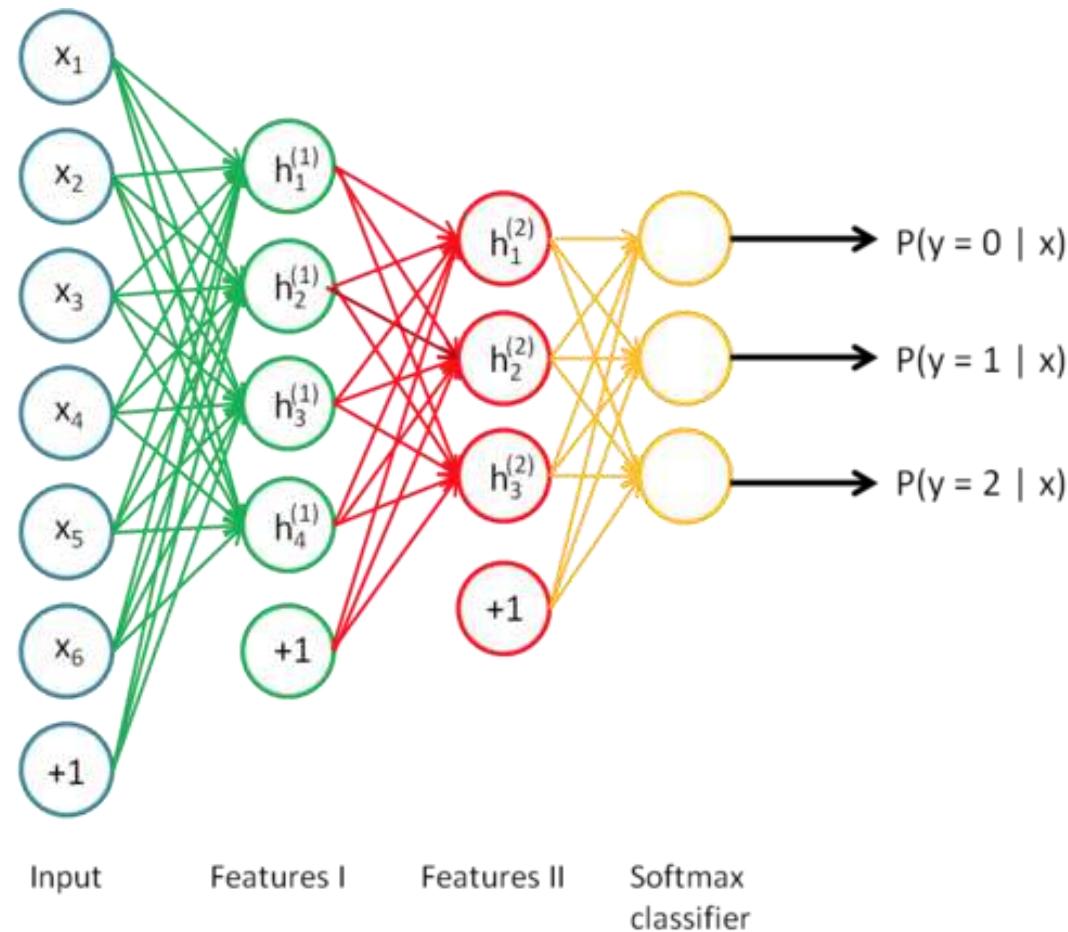
- O fracțiune din intrări, alese la întâmplare pentru fiecare instanță, sunt setate la valorile minime sau maxime posibile (de exemplu, 0 sau 1) în mod aleatoriu

# Autoencodere aggregate în stivă



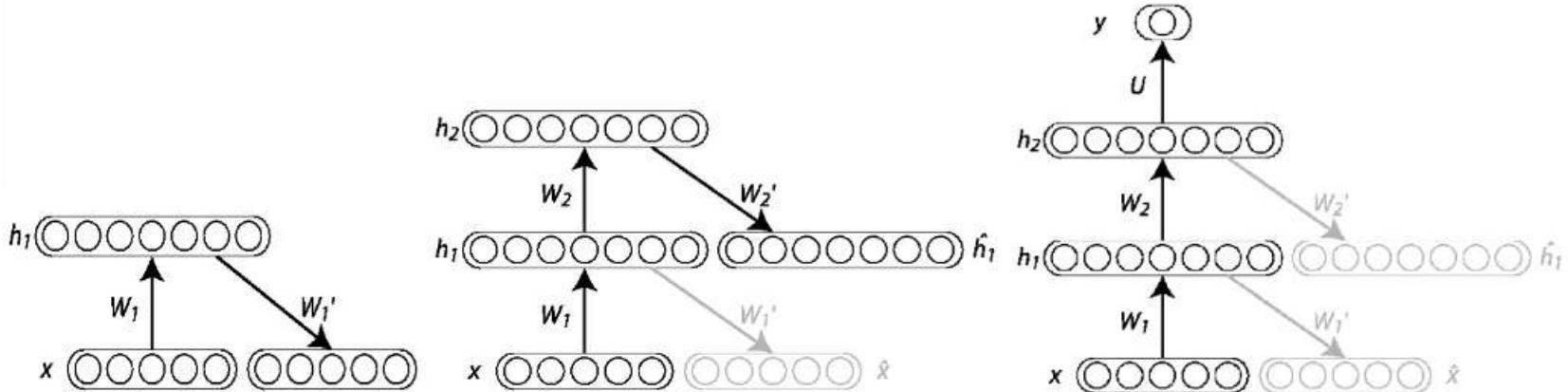
o nouă  
multime de  
antrenare  
pentru  
următorul  
nivel

# Autoencodere agregate în stivă



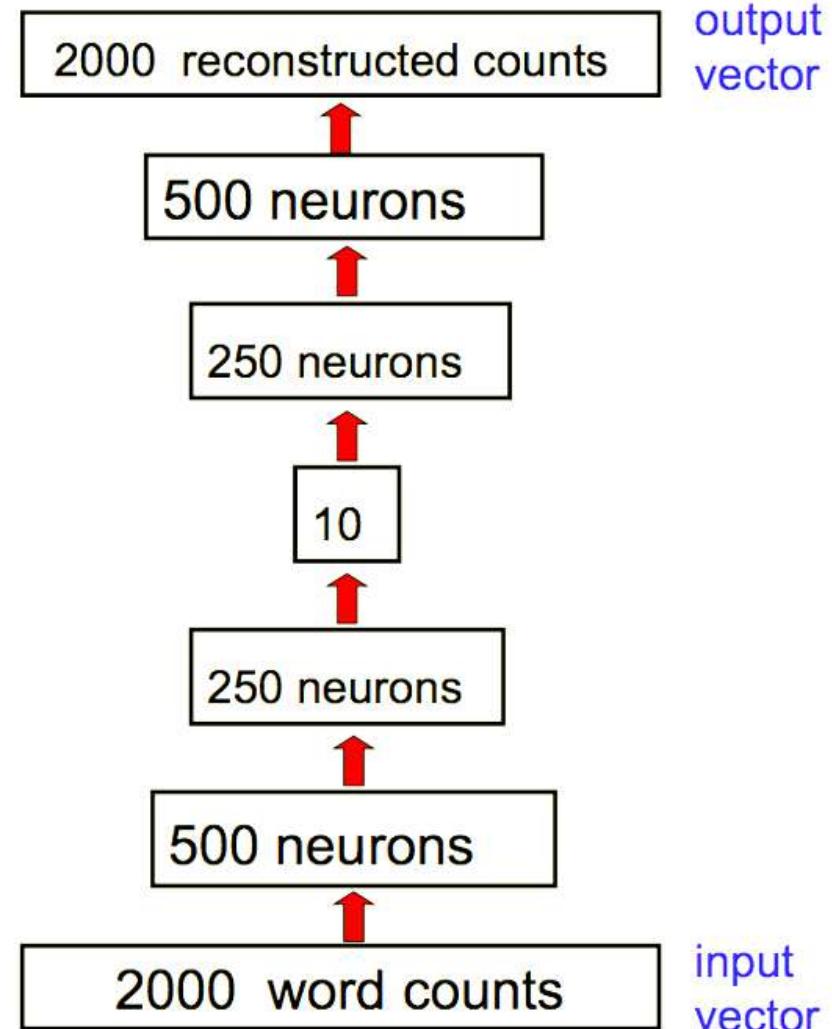
# Procedura de antrenare

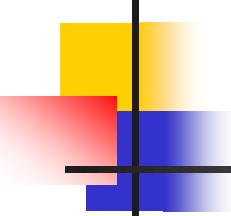
- Antrenare *greedy* nivel cu nivel
- Fiecare nivel se antrenează pe baza codurilor intermediare ale nivelului precedent
- Ultimul nivel are de obicei o ieșire, de exemplu softmax, care realizează clasificarea pe baza trăsăturilor detectate
- În final, se folosește algoritmul *backpropagation* pentru reglarea supervizată a ponderilor (*fine tuning*)



# Exemplul 1

- Hinton & Salakhutdinov: *Reducing the Dimensionality of Data with Neural Networks*, 2006
- Clasificarea documentelor din corpusul *Reuters RCV1-v2* (804 414 documente) pe baza celor mai frecvente 2000 de cuvinte
- Trăsăturile caracteristice fiecărui document sunt comprimate într-un vector de 10 elemente pe nivelul superior
- Acești vectori se folosesc pentru a clasifica documentele

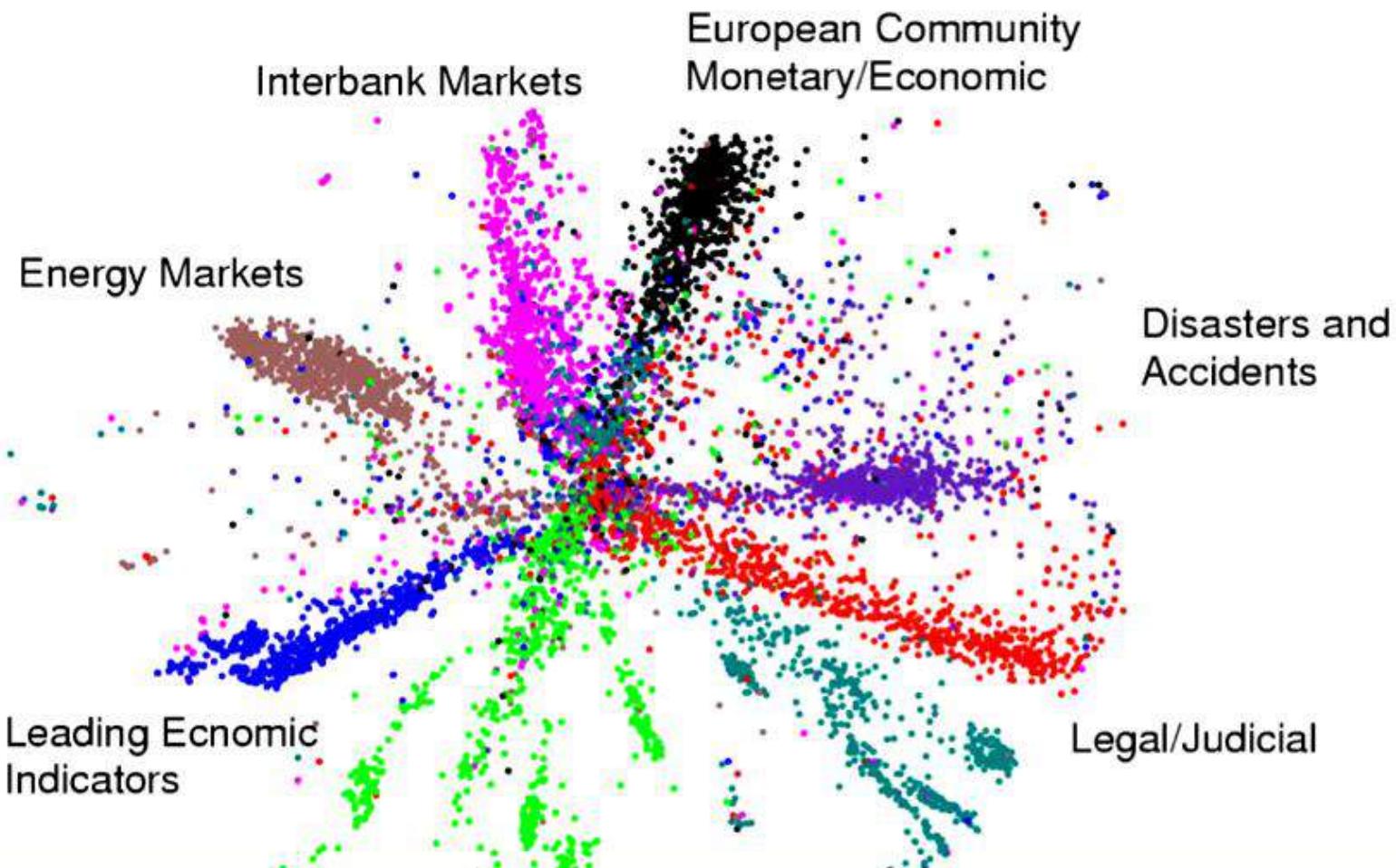


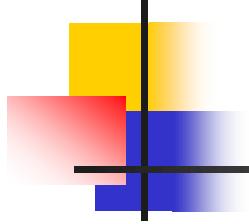


# Exemplul 1

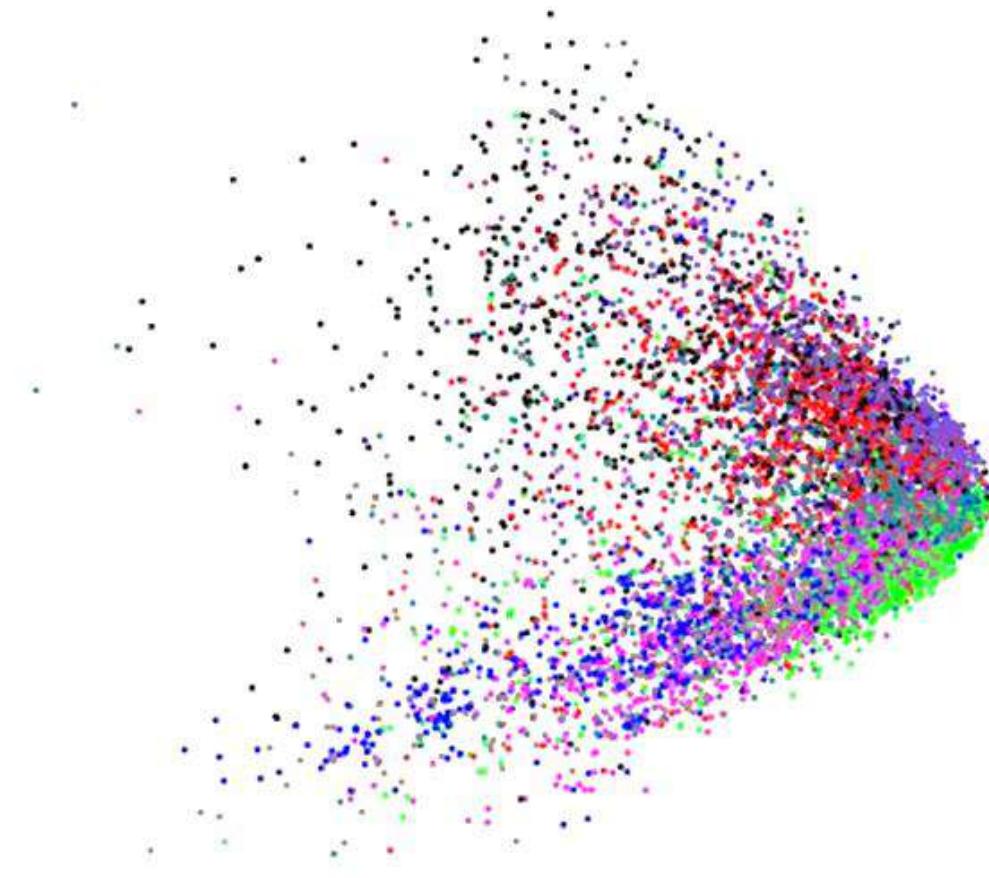
- Trăsăturile fiecărui document pot fi comprimate suplimentar într-un vector cu 2 elemente, folosit pentru vizualizare
- Slide-urile următoare prezintă rezultatele comprimării cu un autoencoder, în comparație cu cele obținute prin LSA (*Latent Semantic Analysis*), metodă asemănătoare cu PCA (*Principal Component Analysis*)
- Culoarea reprezintă categoria documentelor
- Rezultatele obținute cu autoencoderul sunt mult mai clare

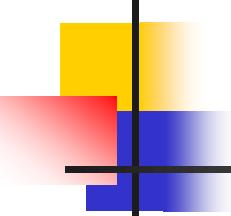
# Rezultate autoencoder





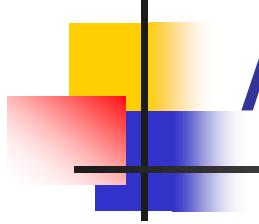
# Rezultate LSA





## Exemplul 2

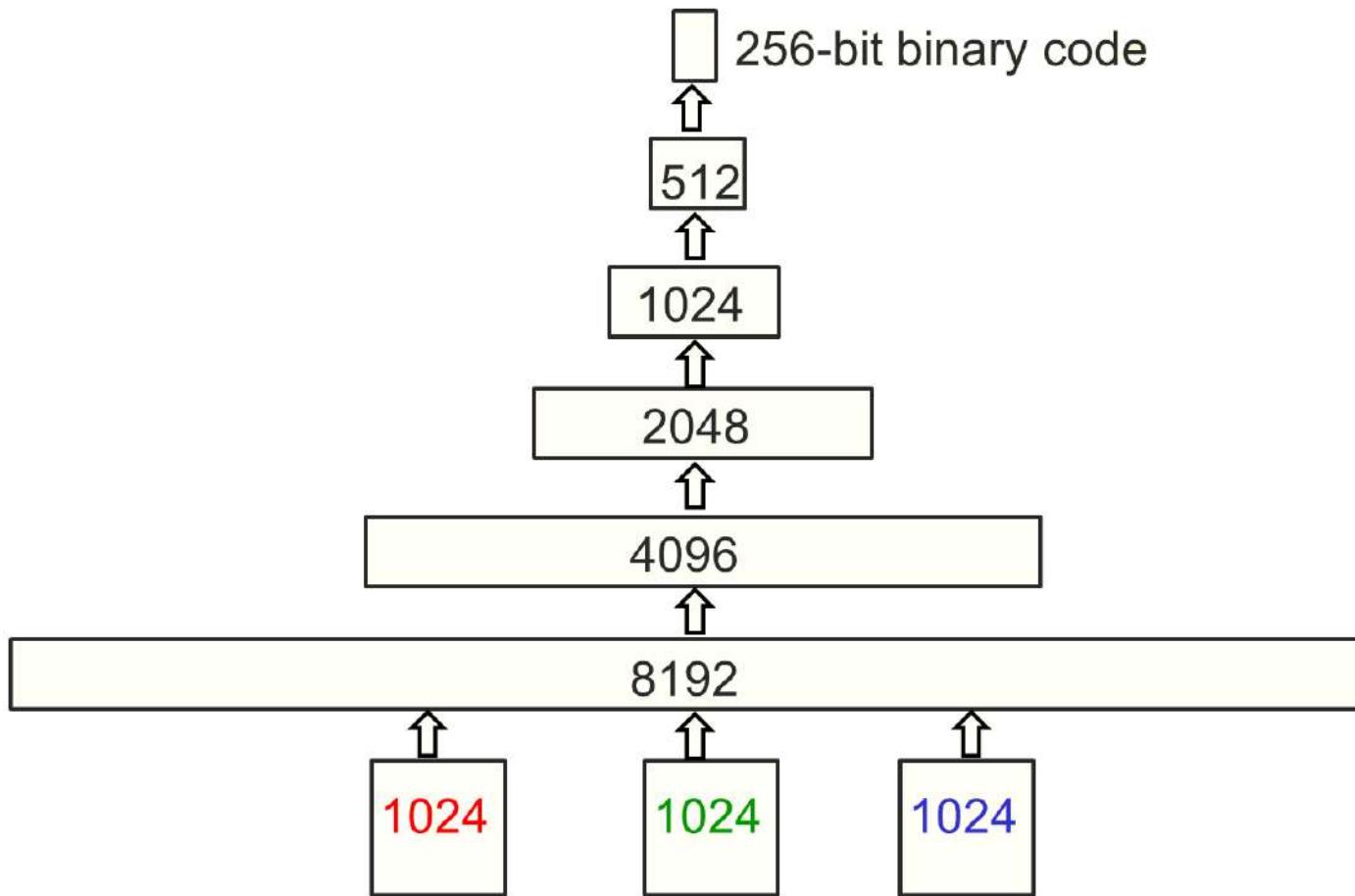
- Krizhevsky, Sutskever & Hinton: *ImageNet Classification with Deep Convolutional Neural Networks*, 2012
- Regăsirea de imagini asemănătoare cu o imagine dată
- Multime de antrenare de 1.6 milioane de imagini
- Metode de calcul al similarității:
  - Autoencoder profund ( $A$ )
  - Cod spectral de 256 de biți ( $S$ )
  - Distanța euclidiană ( $E$ )

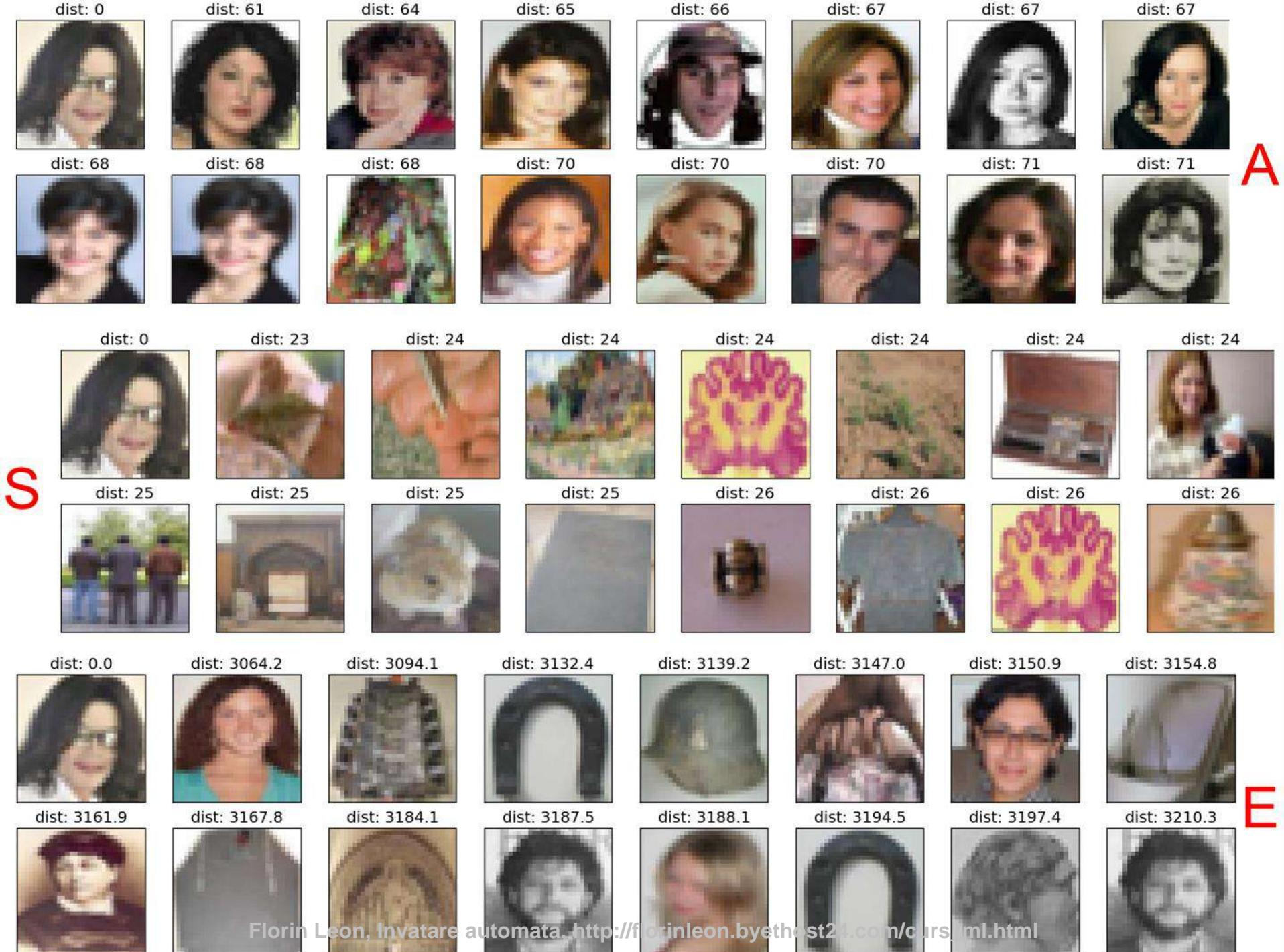


# Autoencoderul

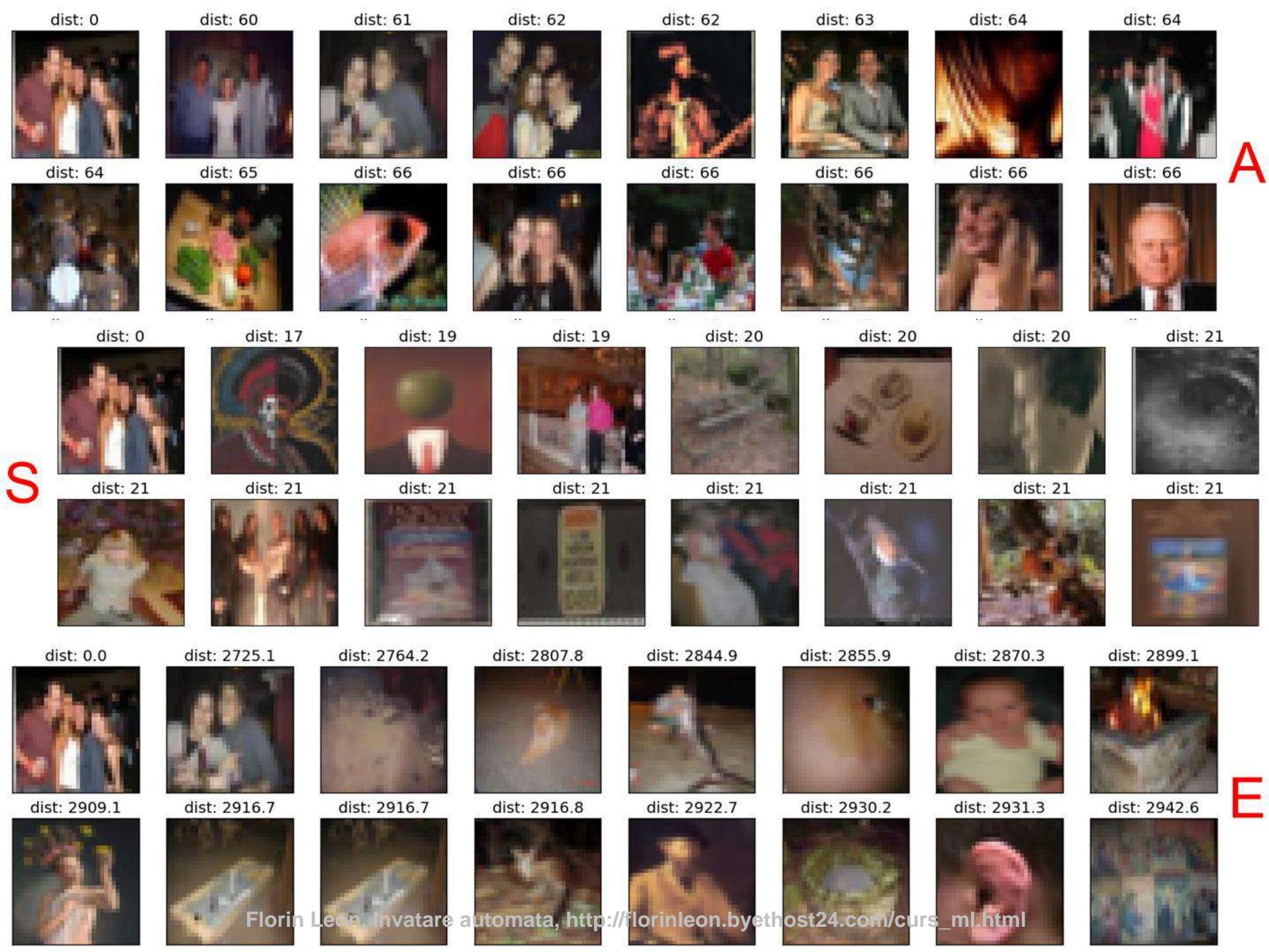
- Preia imagini cu 3 canale (RGB) și produce coduri binare de 256 de biți
- Are aproximativ 67 de milioane de parametri
- A necesitat câteva zile de antrenare pe o mulțime de GPU GTX 285
- Rezultatele obținute de autoencoder, prezentate în slide-urile următoare, sunt cele mai bune

# Autoencoderul



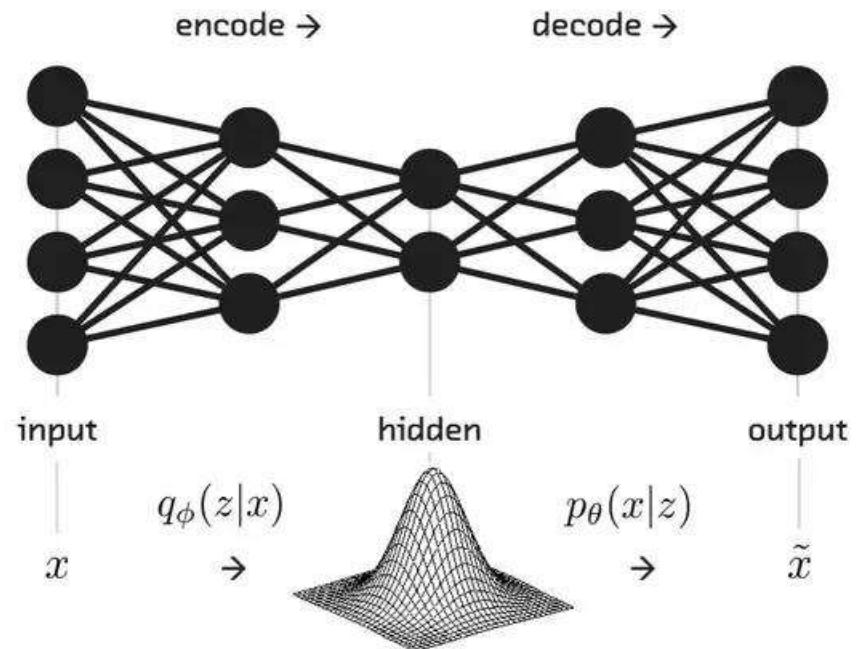




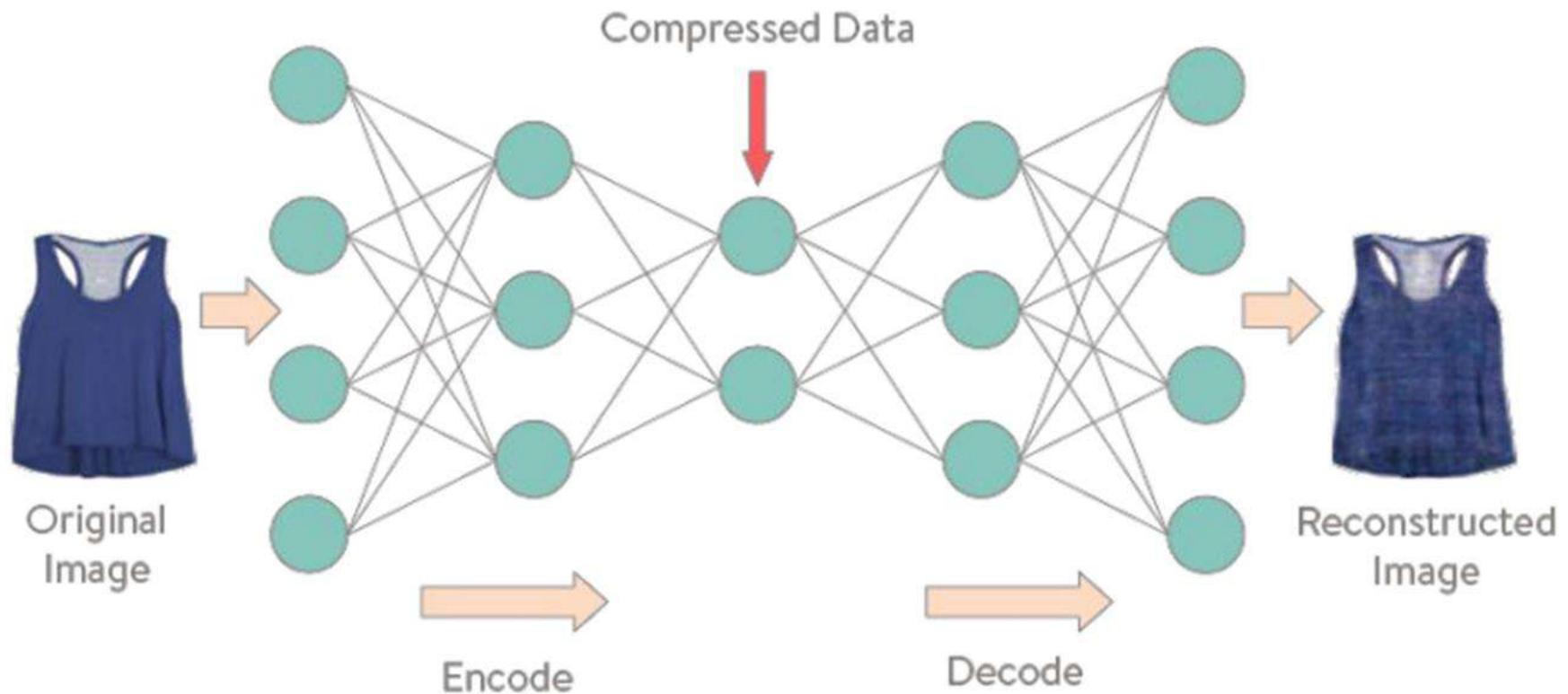


# Variational Autoencoder

- Ideea: se determină o distribuție parametrică, de exemplu gaussiană, care aproximează datele de intrare
- Pot fi eșantionate noi exemple din această distribuție



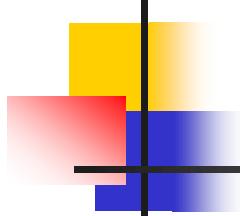
# Variational Autoencoder



# Variational Autoencoder

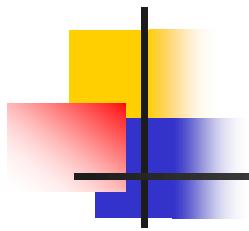


Aceste fețe sunt generate de autoencoder, nu aparțin mulțimii de antrenare



# Discuție

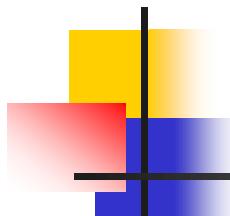
- Deși au apărut în cercetările de învățare profundă după modelele probabilistice generative (care vor fi prezentate în cursul 6), în prezent le-au depășit ca performanță, deși modelele generative au o mai mare asemănare cu mecanismele naturale și biologice
- *Stacked denoising autoencoders* sunt comparabile ca performanță cu *deep belief networks*
- *Variational autoencoders* reprezintă un tip recent de autoencodere, cu performanțe foarte bune



# Rețele neuronale profunde

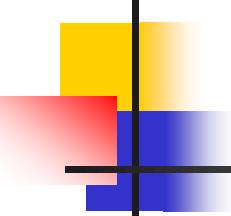
1. Învățarea profundă: scurt istoric
2. Rețele profunde cu propagare înainte
3. Autoencodere
4. Biblioteci de învățare profundă





# Biblioteci de învățare profundă

- PyTorch
- TensorFlow
- Keras
  
- Caffe
- Microsoft Cognitive Toolkit (CNTK)
- Theano
- Deeplearning4j
- Accord.NET



# Concluzii

- Succesul din prezent al inteligenței artificiale se datorează în mare măsură tehnicielor de învățare profundă
- Rețelele profunde au mai multe straturi decât rețelele neuronale clasice și pot avea alte funcții de activare, alte funcții de cost, alte metode de regularizare și alți algoritmi de antrenare
- Autoencoderele învață funcția identitate, dar stratul ascuns reprezintă o versiune comprimată a intrării, care reține trăsăturile esențiale



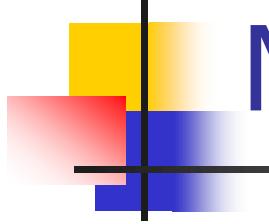
# Învățare automată

## 6. Modele bazate pe energie

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

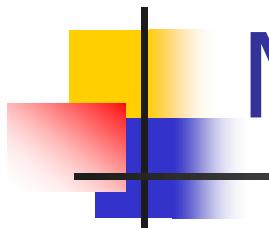
[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)



# Modele bazate pe energie

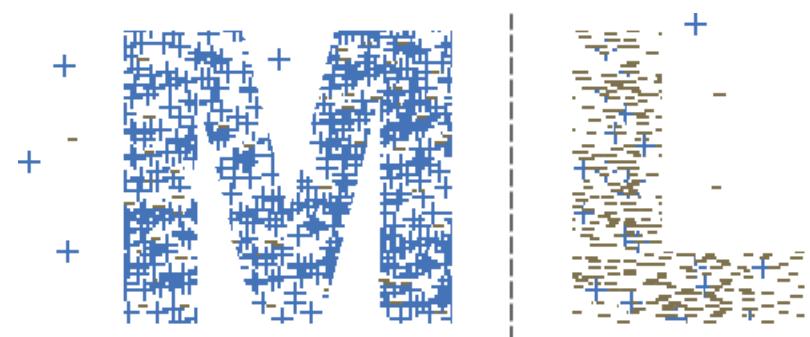
1. Modelul Ising
2. Rețele Hopfield
3. Mașini Boltzmann
4. Mașini Boltzmann restrictionate
5. Rețele de convingeri profunde





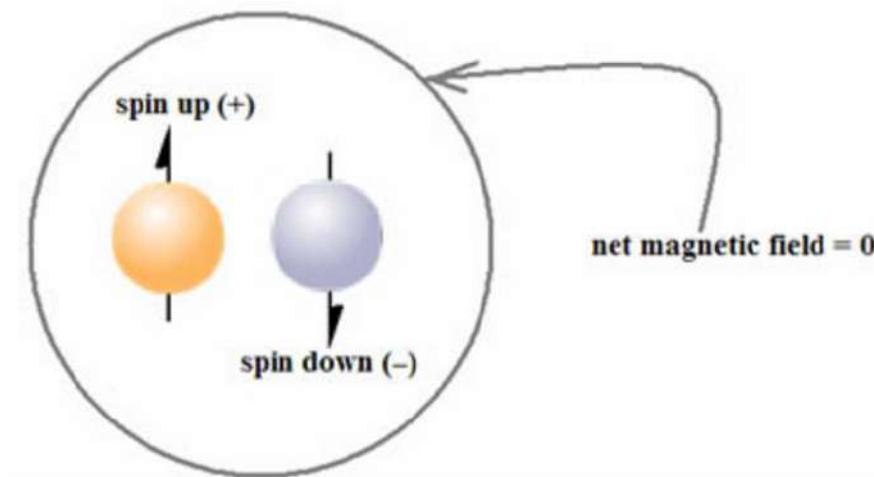
# Modele bazate pe energie

1. Modelul Ising
2. Rețele Hopfield
3. Mașini Boltzmann
4. Mașini Boltzmann restricționate
5. Rețele de convingeri profunde



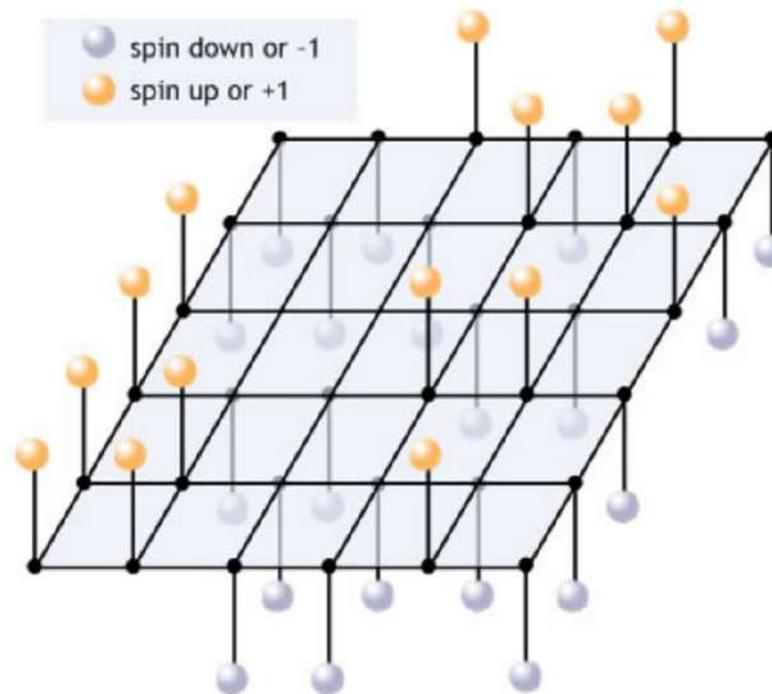
# Modelul Ising

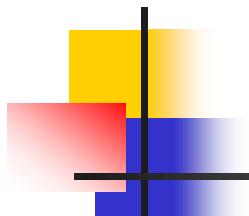
- Modelul Ising (1D, 1924; 2D, 1944) este un model al feromagnetismului, cu variabile discrete care reprezintă momentele magnetice ale spinurilor atomice



# Modelul Ising

- Modelul se bazează pe o latice a momentelor magnetice ale atomilor





# Noțiuni de bază

- Energia unei stări

$$E(s) = - \sum_{(i,j)} J_{ij} s_i s_j - M \sum_i H_i s_i$$

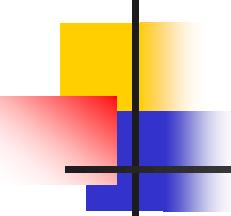
$(i, j)$  reprezintă perechile de atomi vecini

- Probabilitatea unei stări

$$P_T(s) = \frac{e^{-E(s)/(k_B \cdot T)}}{Z}$$

$$Z = \sum_s e^{-E(s)/(k_B \cdot T)}$$

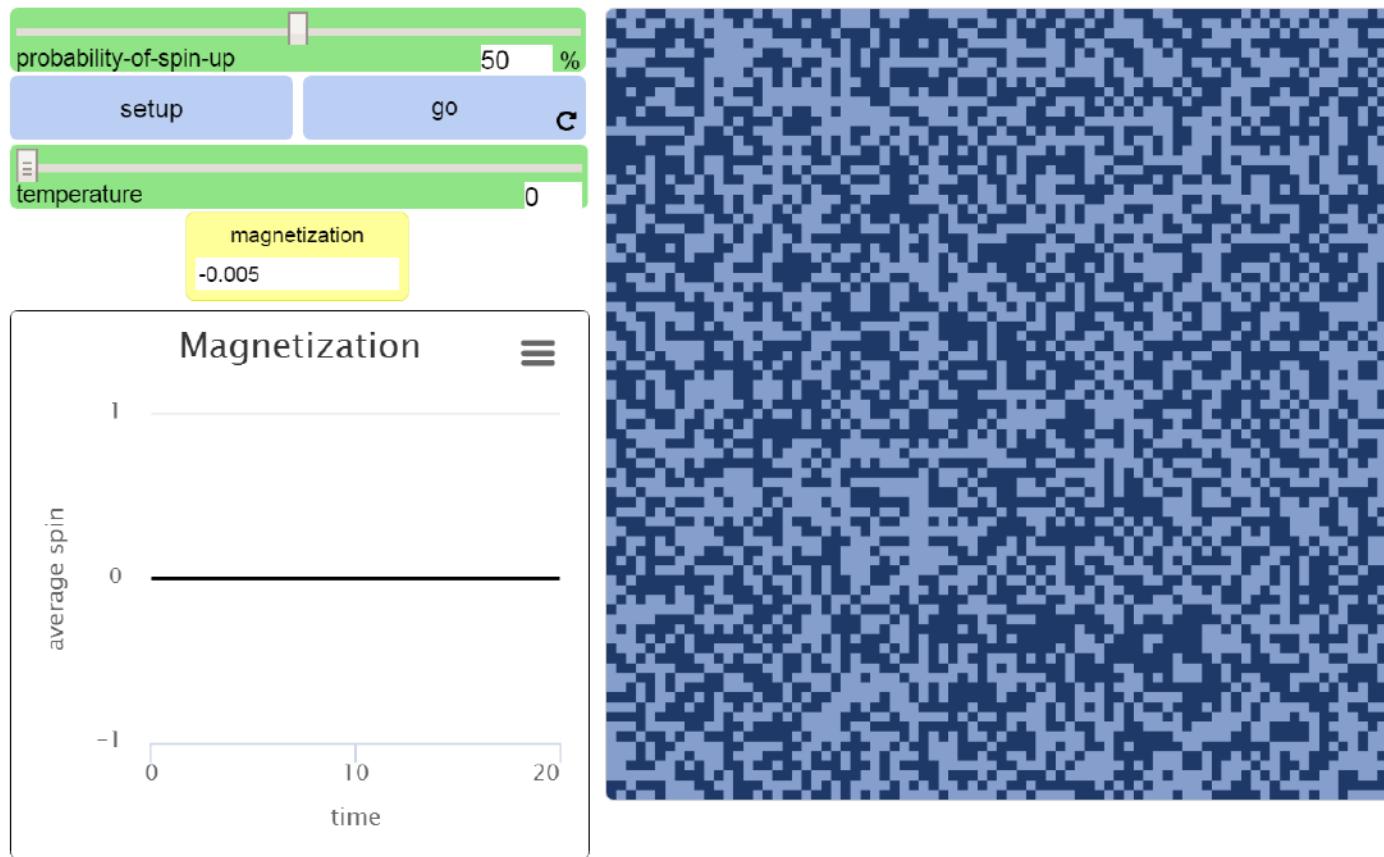
$J_{ij}$ : puterea interacțiunii vecinilor  $(i, j)$   
 $H_i$ : câmp magnetic extern  
 $M$ : momentul magnetic  
 $T$ : temperatura



# Evoluția sistemului

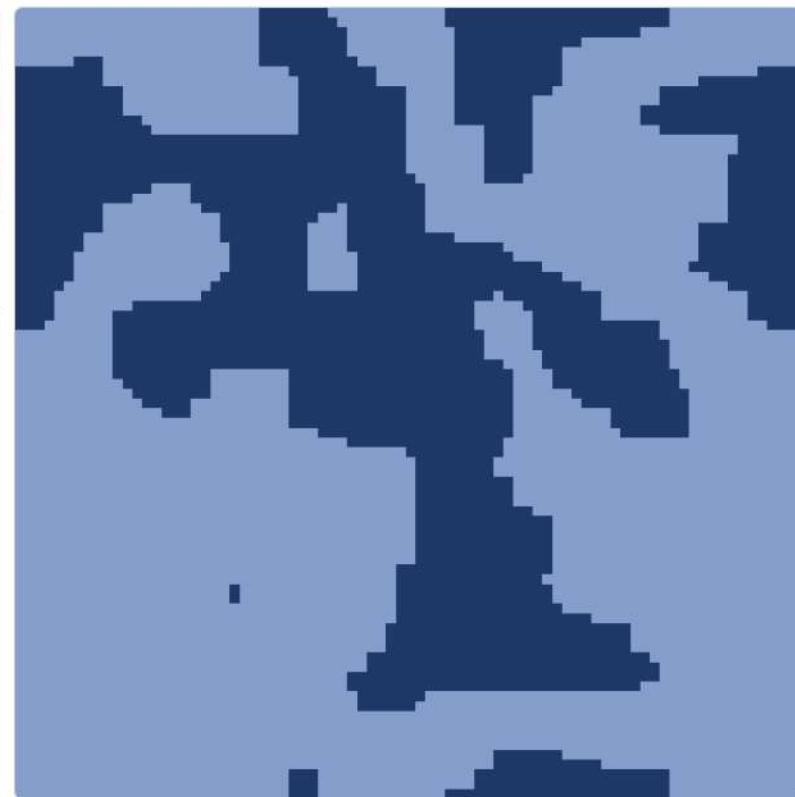
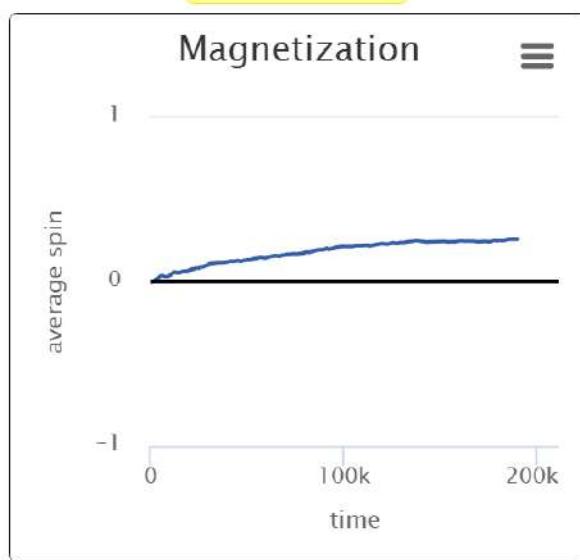
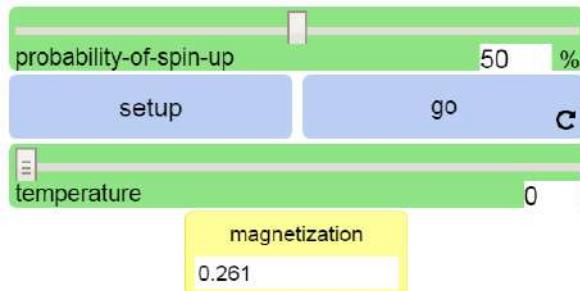
- Sistemul tinde către o stare cu energie minimă
- Atomii vecini au tendința să își modifice orientarea spinului pentru a se alinia cu vecinii, dacă aceasta scade energia sistemului
- Un atom își schimbă întotdeauna orientarea spinului când configurația rezultată are o energie mai mică ( $E_{next} < E_{crt}$ )
- Dar chiar dacă energia este mai mare ( $E_{next} > E_{crt}$ ), își poate schimba orientarea cu probabilitatea  $P = \exp(-\Delta E / T)$ , unde  $\Delta E = E_{next} - E_{crt}$
- Mai multe detalii despre călirea simulată se găsesc în cursul 5 de Inteligență artificială

# Simulare: starea inițială

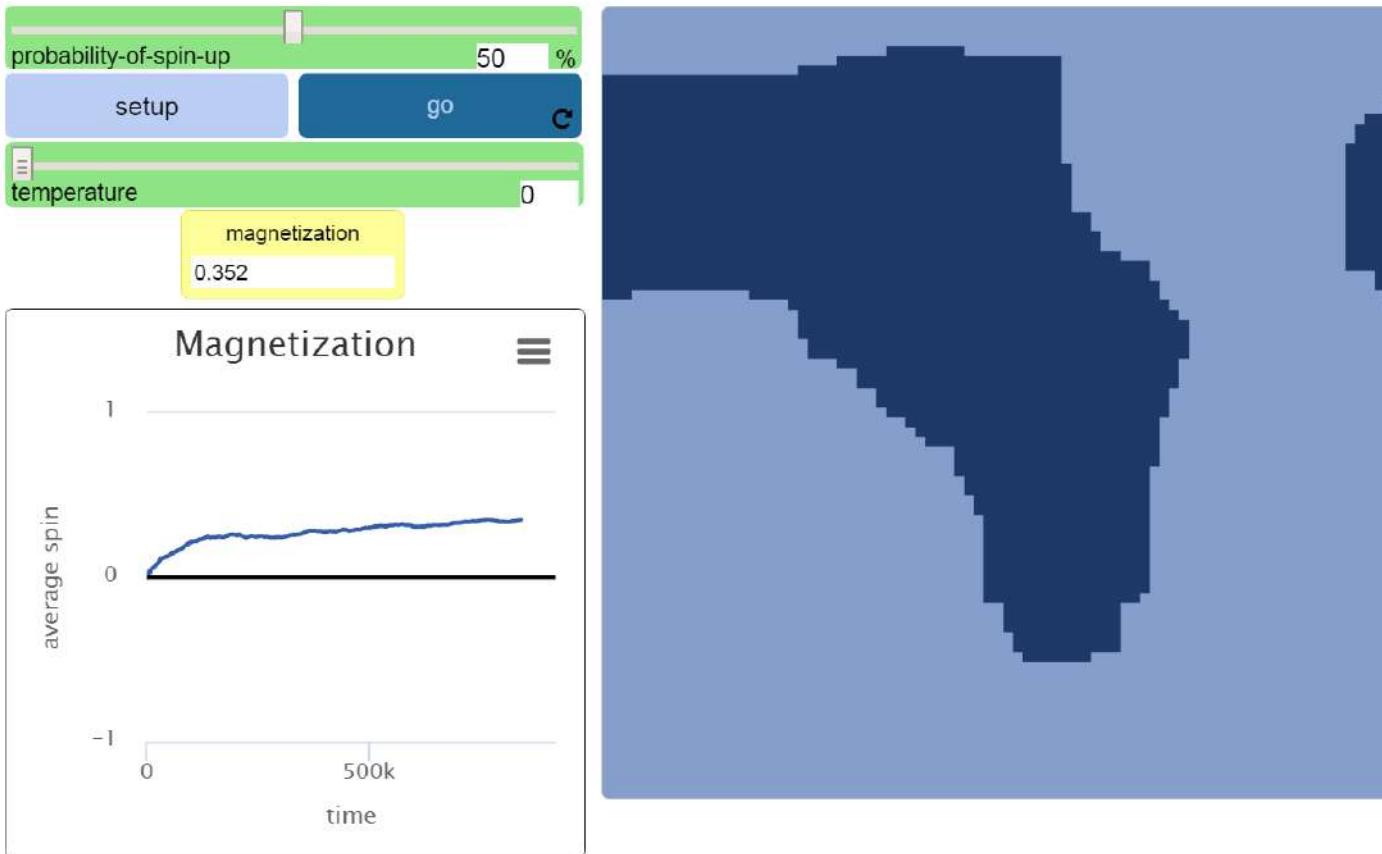


<http://netlogoweb.org/launch#http://netlogoweb.org/assets/modelslib/Sample%20Models/Chemistry%20&%20Physics/Ising.nlogo>

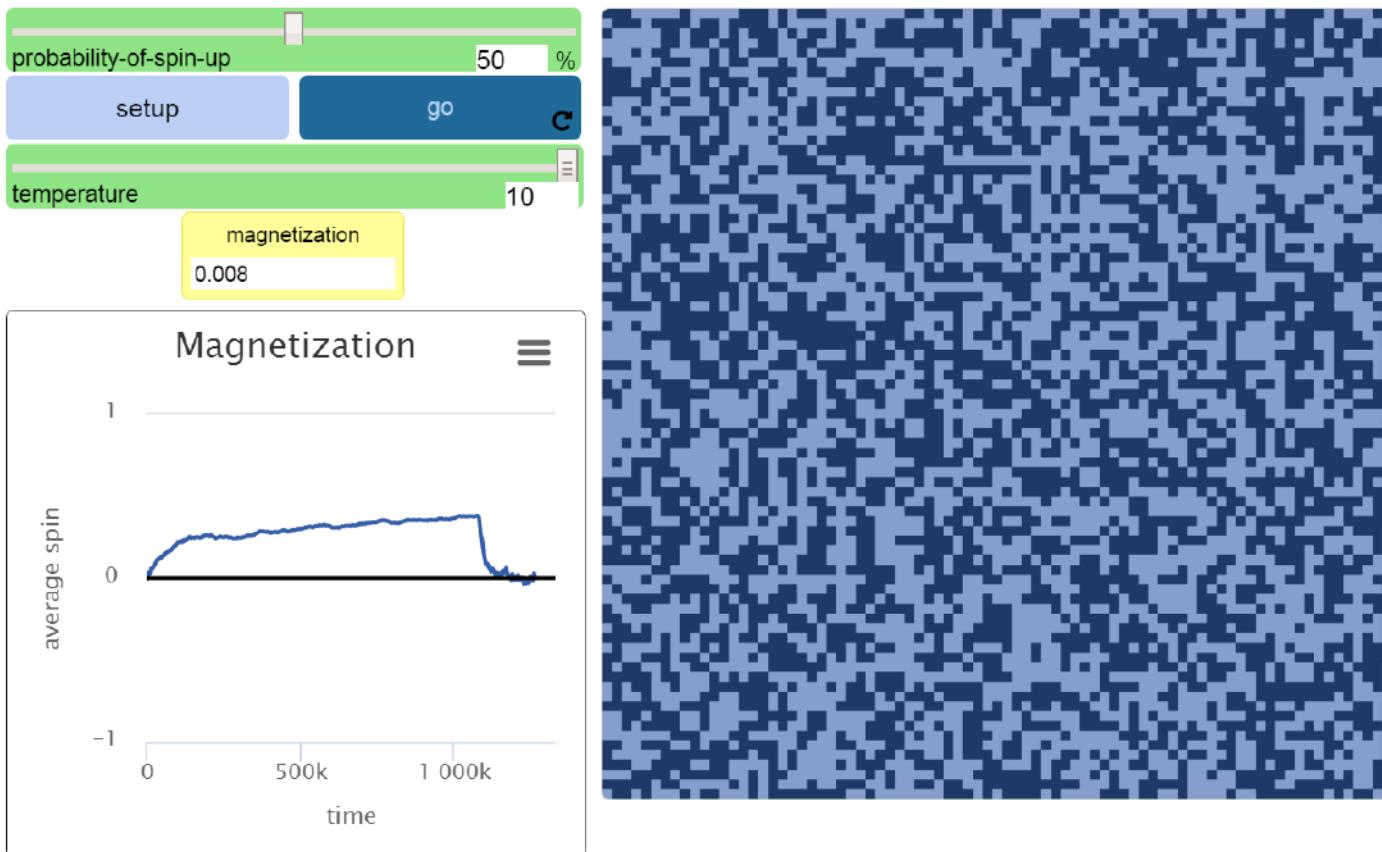
# Temperatură mică



# Temperatură mică: comportament feromagnetic

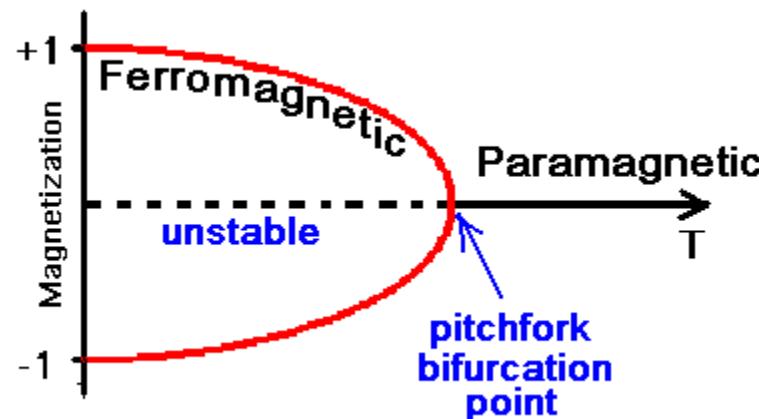


# Temperatură mare: comportament paramagnetic



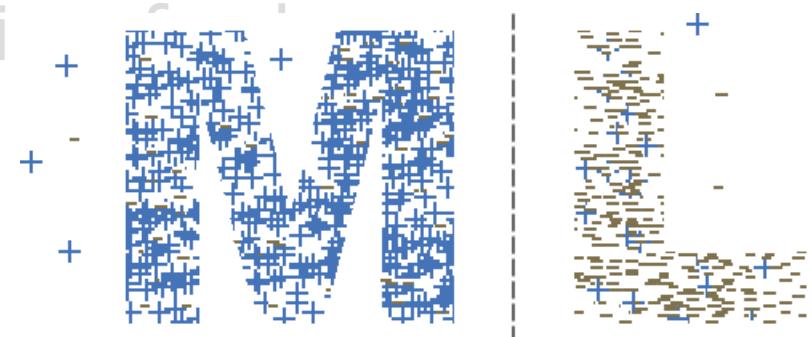
# Tranzitii de fază

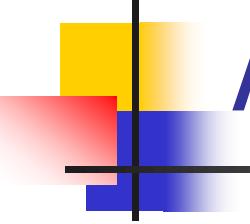
- Modelul 1D nu are tranzitii de fază
- Modelul 2D are
  - Până la temperatura critică, sistemul poate ajunge ori în starea +1, ori în starea -1



# Modele bazate pe energie

1. Modelul Ising
2. Rețele Hopfield
  - 2.1. Procesul de învățare
  - 2.2. Procesul de optimizare
3. Mașini Boltzmann
4. Mașini Boltzmann restricționate
5. Rețele de convingeri



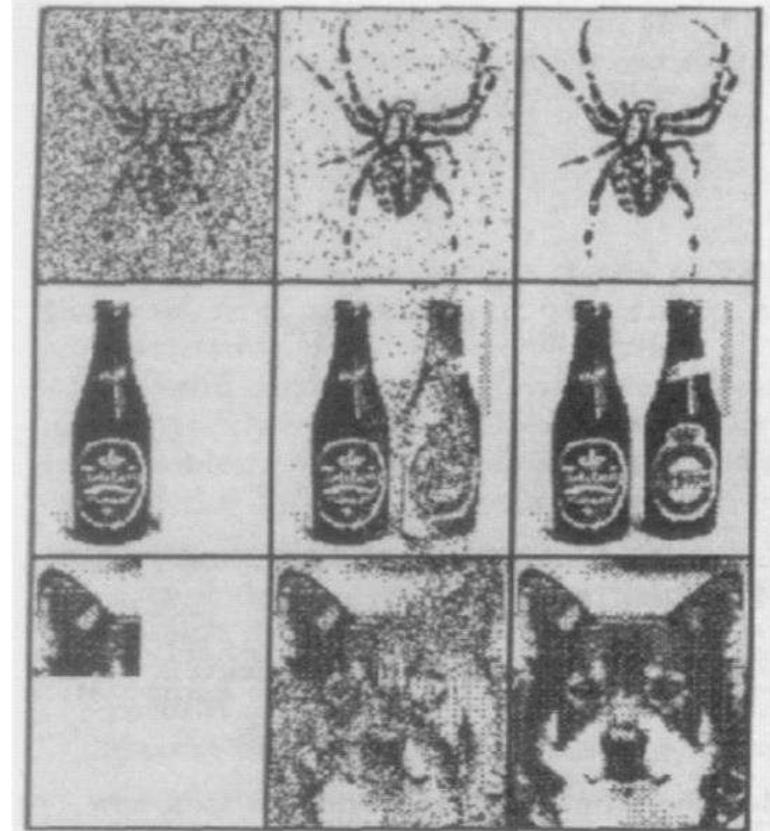


# Asocieri

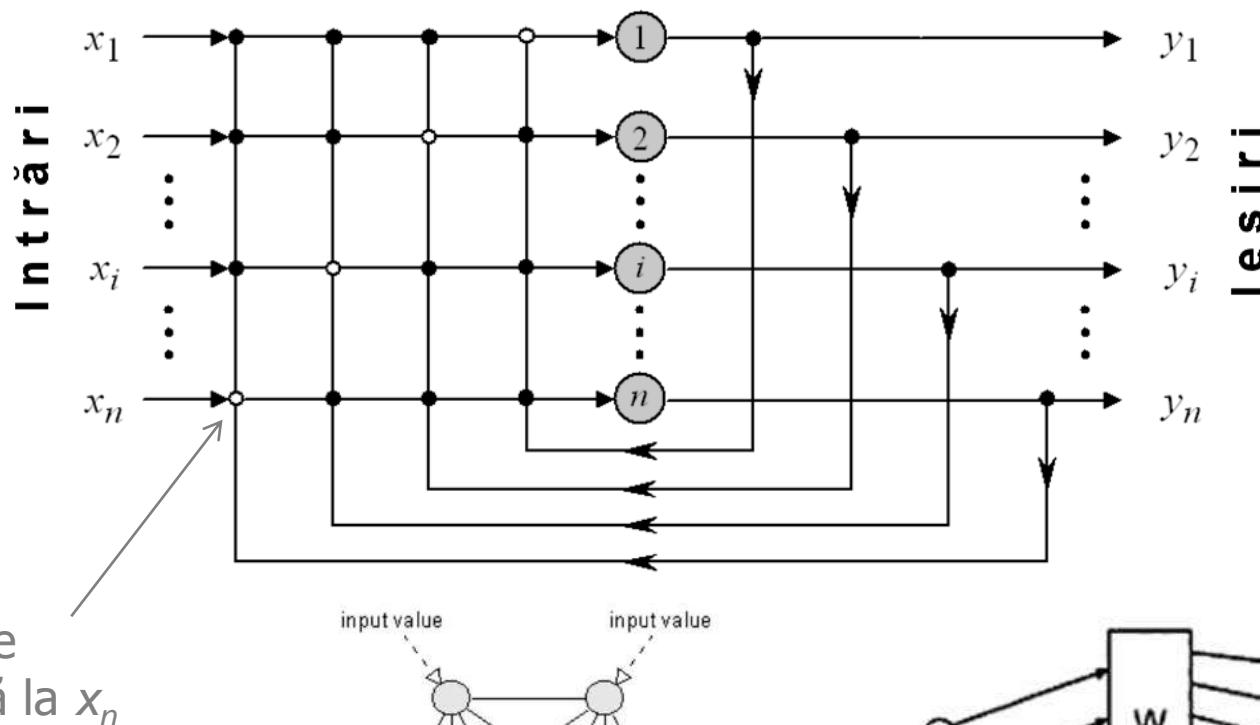
- În creierele biologice, memoria lucrează prin asociere
  - Putem recunoaște o față cunoscută într-un mediu necunoscut în 100-200 ms
  - Putem rememora o întreagă experiență senzorială, vizuală și auditivă, când auzim primele măsuri ale unei melodii
- Perceptronul multistrat nu lucrează prin asociere
- Pentru emularea acestei capacitați, este nevoie de un alt tip de rețea neuronală: o **rețea recurrentă**, care are bucle de reacție de la ieșiri către intrări
- Datorită reacției (*feedback*), stabilitatea rețelei devine o problemă. Aceasta a fost rezolvată abia în 1982, când Hopfield a formulat principiul de stocare a informațiilor într-o rețea stabilă din punct de vedere dinamic

# Asocieri

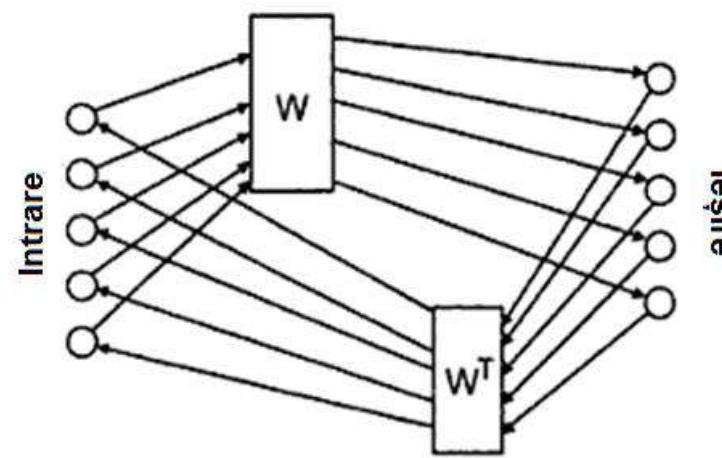
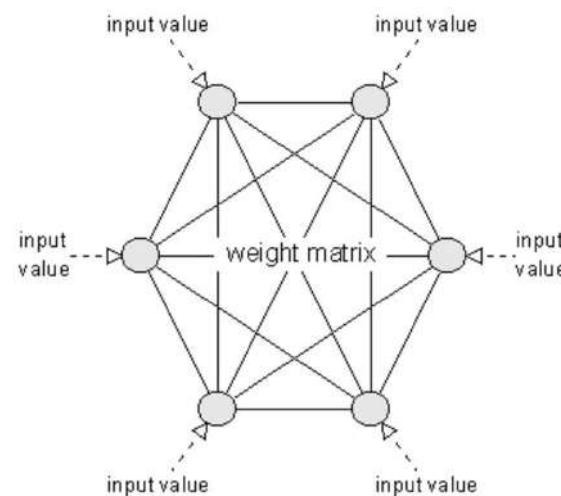
- Rețeaua Hopfield este o memorie auto-asociativă:
  - Își poate aminti modelele stocate
  - Își poate aminti un model dacă primește doar o parte din el
  - Își poate aminti modelele dacă primește versiuni similare, dar nu identice, sau versiuni afectate de zgomot

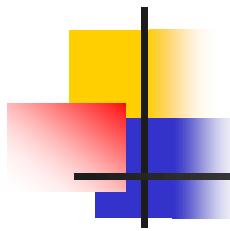


# Rețea Hopfield cu $n$ neuroni



$y_n$  nu este conectată la  $x_n$



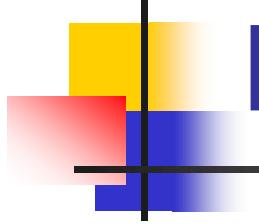


# Caracteristici

- Rețeaua Hopfield folosește neuroni cu **funcție de activare semn**

$$Y^{(t+1)} = \begin{cases} +1, & \text{dacă } X^{(t)} > 0 \\ -1, & \text{dacă } X^{(t)} < 0 \\ Y^{(t)}, & \text{dacă } X^{(t)} = 0 \end{cases}$$

- Starea curentă a rețelei este determinată de ieșirile tuturor neuronilor:  $\mathbf{Y} = (y_1, y_2, \dots, y_n)$
- $\mathbf{Y}$  se numește **vector de stare**



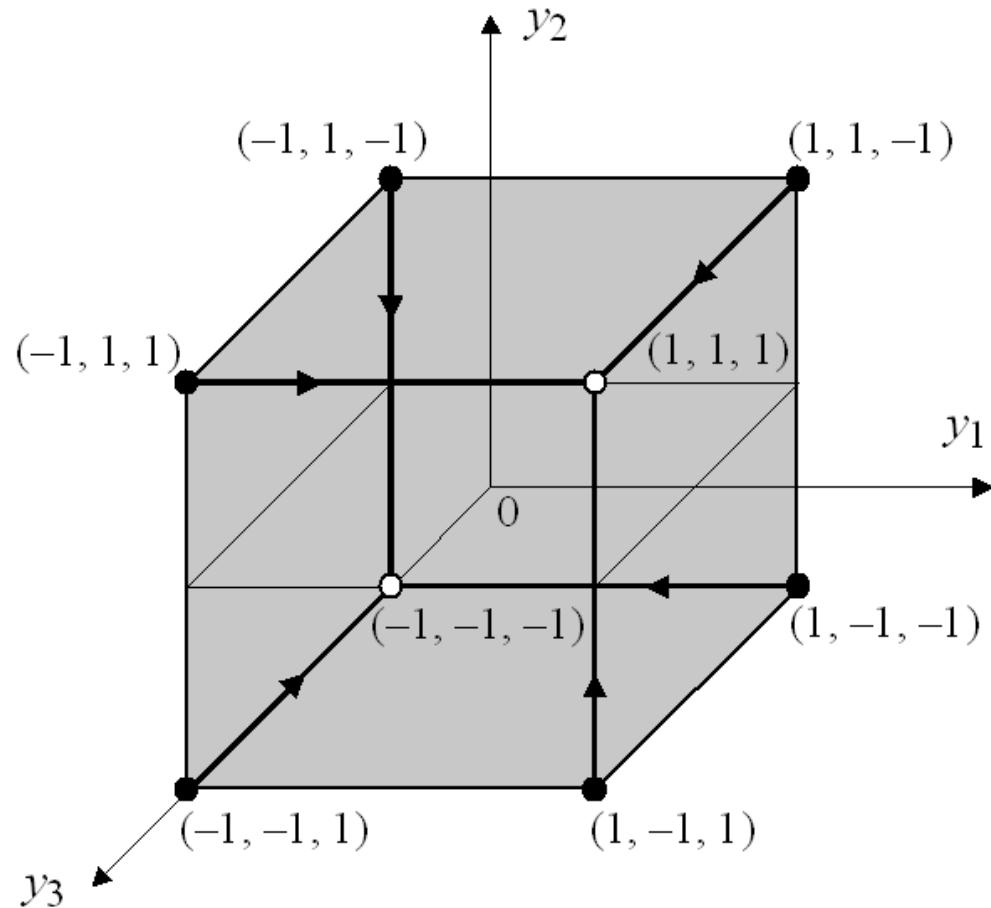
# Ponderile

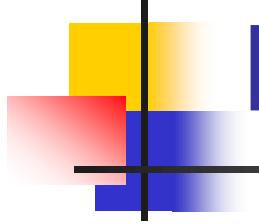
- Ponderile se reprezintă de obicei în formă matriceală:

$$\mathbf{W} = \sum_{m=1}^M \mathbf{Y}_m \mathbf{Y}_m^T - M \mathbf{I}$$

unde  $M$  este numărul de stări care trebuie memorate,  
 $\mathbf{Y}_m$  este un vector de antrenare binar  $n$ -dimensional  
și  $\mathbf{I}$  este matricea identitate de dimensiune  $n \times n$

# Stări posibile pentru o rețea cu trei neuroni

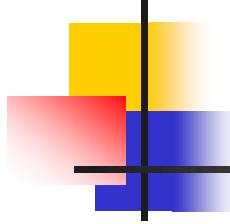




# Exemplu

- Să presupunem că vrem să memorăm două stări: (1, 1, 1) și (-1,-1,-1)

$$\mathbf{Y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$



# Exemplu

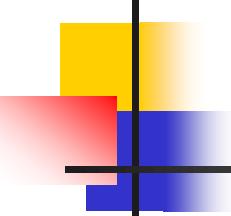
$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [1 \ 1 \ 1] + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} [-1 \ -1 \ -1] - 2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

# Testarea rețelei

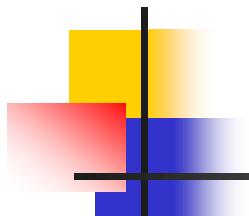
$$\mathbf{Y}_1 = sign \begin{Bmatrix} \left[ \begin{array}{ccc|c} 0 & 2 & 2 & 1 \\ 2 & 0 & 2 & 1 \\ 2 & 2 & 0 & 1 \end{array} \right] \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

$$\mathbf{Y}_2 = sign \begin{Bmatrix} \left[ \begin{array}{ccc|c} 0 & 2 & 2 & -1 \\ 2 & 0 & 2 & -1 \\ 2 & 2 & 0 & -1 \end{array} \right] \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ -1 \end{Bmatrix}$$



# Stări stabile și instabile

- Aceste 2 stări sunt stabile. Celelalte 6 stări rămase sunt instabile
- Stările stabile (numite **amintiri fundamentale**) sunt capabile să atragă stările apropiate
- Amintirea fundamentală  $(1, 1, 1)$  atrage stările instabile  $(-1, 1, 1)$ ,  $(1, -1, 1)$  și  $(1, 1, -1)$
- Amintirea fundamentală  $(-1, -1, -1)$  atrage stările instabile  $(-1, -1, 1)$ ,  $(-1, 1, -1)$  și  $(1, -1, -1)$
- Fiecare din aceste stări are o singură eroare, în comparație cu amintirea fundamentală
- Deci o rețea Hopfield are capacitatea de a **corecta erorile** din stările prezentate



## Exemplu: stări instabile

$$\mathbf{Y} = \text{sign} \left( \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \right) = \text{sign} \left( \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

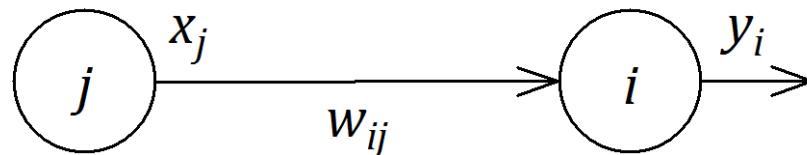
$$\mathbf{Y} = \text{sign} \left( \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right) = \text{sign} \left( \begin{bmatrix} -4 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

când rezultatul este 0, y își păstrează starea anterioară

# Învățarea hebbiană

## ■ Legea lui Hebb

- Dacă doi neuroni conectați sunt activați în același timp, ponderea conexiunii dintre ei crește
- Dacă doi neuroni sunt activați în contratimp, ponderea conexiunii dintre ei scade
- “*Neurons that fire together, wire together*”



$$\Delta w_{ij} = \alpha \cdot x_j \cdot y_i$$

## ■ Antrenarea rețelei Hopfield este o formă de învățare hebbiană

# Alt exemplu

$$\mathbf{X}_1 = (+1, +1, +1, +1, +1)$$

$$\mathbf{X}_2 = (+1, -1, +1, -1, +1)$$

$$\mathbf{X}_3 = (-1, +1, -1, +1, -1)$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 3 & -1 & 3 \\ -1 & 0 & -1 & 3 & -1 \\ 3 & -1 & 0 & -1 & 3 \\ -1 & 3 & -1 & 0 & -1 \\ 3 & -1 & 3 & -1 & 0 \end{bmatrix}$$

Se observă efectul învățării hebbiene:

Intrările 1 și 2 apar cu același semn în  $\mathbf{X}_1$  și cu semn contrar în  $\mathbf{X}_2$  și  $\mathbf{X}_3 \Rightarrow W_{12} = -1 < 0$

Intrările 1 și 3 apar cu același semn în toți vectorii de antrenare  $\Rightarrow W_{13} = 3 > 0$

# Amintiri false

$$\mathbf{X}_1 = (+1, +1, +1, +1, +1)$$

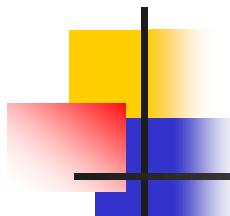
$$\mathbf{X}_2 = (+1, -1, +1, -1, +1)$$

$$\mathbf{X}_3 = (-1, +1, -1, +1, -1)$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 3 & -1 & 3 \\ -1 & 0 & -1 & 3 & -1 \\ 3 & -1 & 0 & -1 & 3 \\ -1 & 3 & -1 & 0 & -1 \\ 3 & -1 & 3 & -1 & 0 \end{bmatrix}$$

$$\mathbf{X} = (+1, +1, -1, +1, +1) \iff \mathbf{X}_3 = (-1, +1, -1, +1, -1)$$

Deoarece  $\mathbf{X}$  este foarte apropiat de  $\mathbf{X}_1$ , ne-am așteptă ca rețeaua să-și amintească  $\mathbf{X}_1$  când i se prezintă  $\mathbf{X}$ . Rețeaua produce însă  $\mathbf{X}_3$ , o amintire falsă.

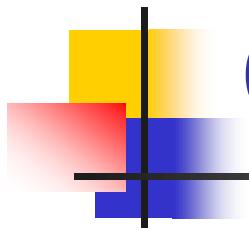


# Energia rețelei Hopfield

- În descrierea anterioară a rețelei Hopfield, nu am luat în calcul pragurile neuronilor
- Dacă includem și pragurile  $\theta$ , **energia** rețelei pentru un vector de stare  $\mathbf{x}$  este:

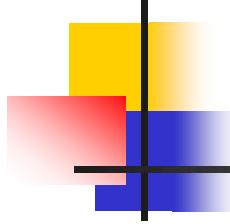
$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}\mathbf{W}\mathbf{x}^T + \theta\mathbf{x}^T$$

$$E(\mathbf{x}) = -\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n w_{ij}x_i x_j + \sum_{i=1}^n \theta_i x_i$$



# Capacitatea de optimizare

- Exemplu: problema plasării turnurilor pe o tablă de sah, astfel încât să nu se atace reciproc
- Fie  $x_{ij}$  starea neuronului corespunzător celulei de pe poziția  $(i, j)$
- $x_{ij}$  este 1 dacă în celulă este un turn și 0 dacă nu
- Condiția  $C_1$ : pe fiecare coloană trebuie să fie doar un turn
- Condiția  $C_2$ : pe fiecare linie trebuie să fie doar un turn



# Rezolvare

- Pentru condiția  $C_1$ , trebuie minimizată funcția:

$$E_1 = \sum_{j=1}^n \left( \left( \sum_{i=1}^n x_{ij} \right) - 1 \right)^2$$

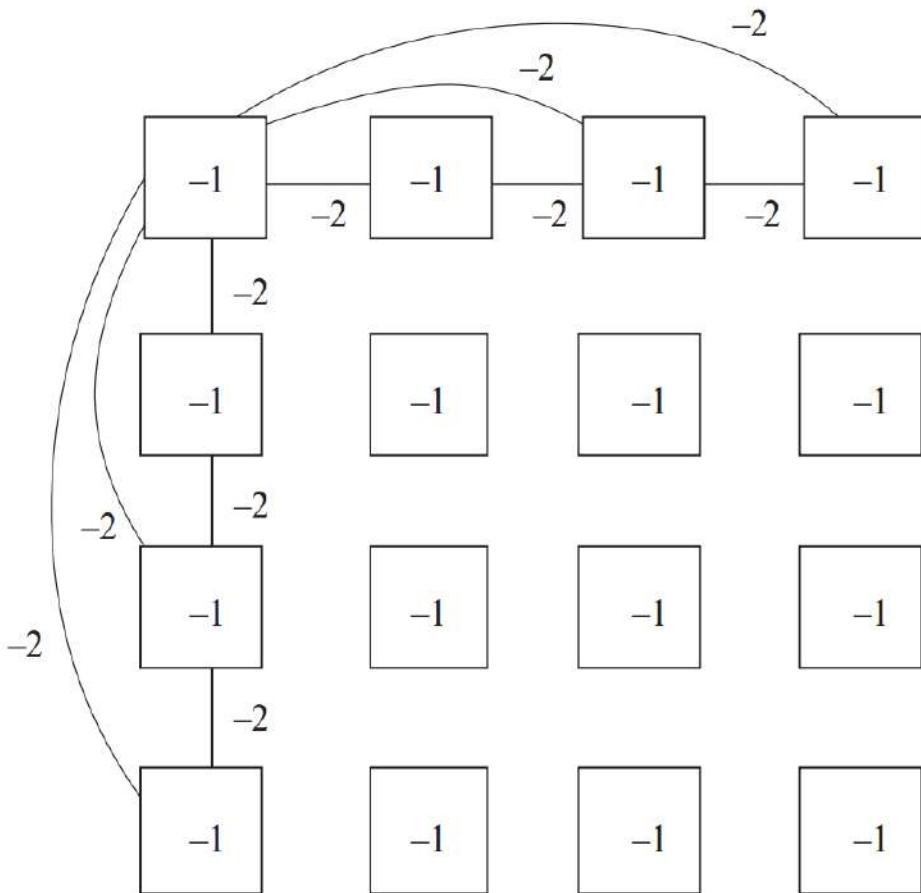
- Pentru condiția  $C_2$ , trebuie minimizată funcția:

$$E_2 = \sum_{i=1}^n \left( \left( \sum_{j=1}^n x_{ij} \right) - 1 \right)^2$$

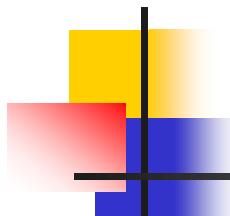
- Trebuie găsite ponderile și pragurile care minimizează energia  $E = E_1 + E_2$

$$E(\mathbf{x}) = -\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n w_{ij} x_i x_j + \sum_{i=1}^n \theta_i x_i$$

# Soluție pentru patru turnuri



- Nu s-au reprezentat toate conexiunile pentru a nu aglomera figura
- Ponderile conexiunilor între neuroni de pe aceeași linie sau coloană sunt -2. Celealte ponderi sunt 0
  - Un neuron „aprins” pe o linie sau coloană îi inhibă pe ceilalți
- Toate pragurile sunt -1
  - Dacă niciun alt neuron nu e aprins pe linie sau coloană, se aprinde neuronul curent



# Rularea rețelei

- Se pornește cu un vector de intrare  $x$  generat aleatoriu, cu elemente de 0 și 1 (în loc de -1 și 1; aşa este definită aici problema)
- Se actualizează în mod repetat starea rețelei (sau până când aceasta nu se mai modifică)
- Pentru a asigura stabilitatea, neuronii trebuie activați secvențial și în ordine aleatorie
- Problema are mai multe soluții
- Rețeaua converge la o soluție, care poate fi diferită la fiecare rulare

```

from random import random, randint
import numpy as np

n = 4

w = np.zeros((n * n, n * n))

for i in range(n * n):
    for j in range(n * n):
        if i != j and (i // n == j // n or i % n == j % n): # împărțire întreagă, respectiv modulo
            w[i, j] = -2

print('w = \n{}'.format(w.astype(int)))

t = np.full((n * n,), -1) # toate pragurile sunt -1

x = np.zeros((n * n,))
for i in range(n * n):
    if random() < 0.5:
        x[i] = 1

print('\nx = \n{}'.format(x.reshape((n,n)).astype(int)))

y = np.zeros((n * n,))

```

```

iter = 1000
for i in range(iter):
    q = randint(0, n * n - 1) # neuronii sunt activați secvențial și în ordine aleatorie
    wq = w[q, :] # linia q din matricea w
    y[q] = np.heaviside(wq.dot(x) - t[q], 0.5) # wq.dot(x) calculează aici produsul scalar
    x = y

print('ny = \n{}'.format(y.reshape((n,n)).astype(int)))

```

w =

```

[[ 0 -2 -2 -2 -2  0  0  0 -2  0  0  0 -2  0  0  0]
 [-2  0 -2 -2  0 -2  0  0  0 -2  0  0  0 -2  0  0]
 [-2 -2  0 -2  0  0 -2  0  0  0 -2  0  0  0 -2  0]
 [-2 -2 -2  0  0  0  0 -2  0  0  0 -2  0  0  0 -2]
 [-2  0  0  0 -2 -2 -2  0  0  0 -2  0  0  0 -2]
 [ 0 -2  0  0 -2  0 -2 -2  0 -2  0  0  0 -2  0  0]
 [ 0  0 -2  0 -2 -2  0 -2  0  0 -2  0  0  0 -2]
 [ 0  0 -2  0 -2 -2  0 -2  0  0 -2  0  0  0 -2]
 [-2  0  0  0 -2  0  0  0 -2 -2 -2 -2  0  0  0]
 [ 0 -2  0  0 -2  0  0 -2  0 -2 -2 -2  0  0  0]
 [ 0  0 -2  0  0 -2  0 -2 -2  0 -2  0  0 -2]
 [ 0  0  0 -2  0  0  0 -2 -2 -2 -2  0  0  0 -2]
 [-2  0  0  0 -2  0  0  0 -2  0  0  0 -2 -2 -2]
 [ 0 -2  0  0  0 -2  0  0  0 -2  0  0 -2  0 -2]
 [ 0  0 -2  0  0  0 -2  0  0  0 -2  0 -2 -2  0]
 [ 0  0  0 -2  0  0  0 -2  0  0  0 -2 -2 -2  0]]

```

x =

```

[[1 1 0 0]
 [0 1 0 0]
 [1 1 1 0]
 [1 0 1 0]]

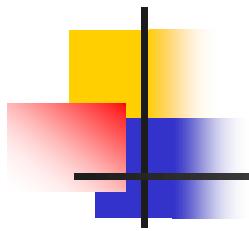
```

y =

```

[[0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]]

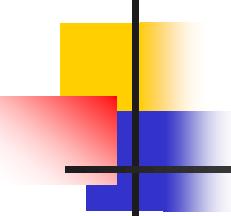
```



# Modele bazate pe energie

1. Modelul Ising
2. Rețele Hopfield
- 3. Mașini Boltzmann**
4. Mașini Boltzmann restricționate
5. Rețele de convingeri profunde



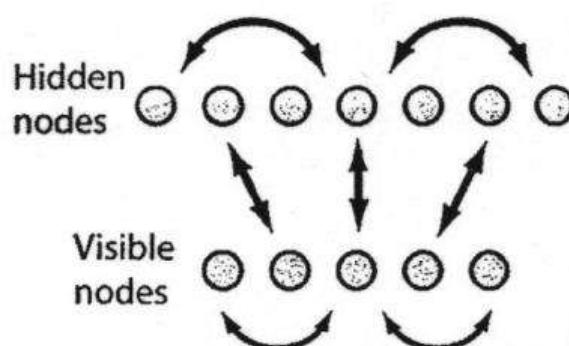
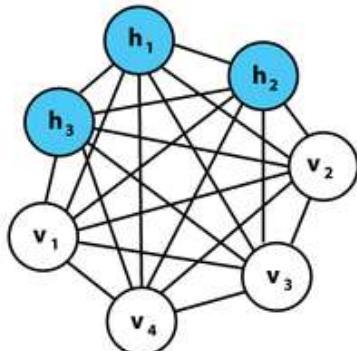


# Ipoteza creierului anticipativ

- Creierul realizează un model generativ probabilistic al lumii, care este folosit pentru a anticipa sau prezice stările mediului
- Inversul modelului poate fi folosit pentru a recunoaște cauzele unor evenimente externe prin evocarea conceptelor interne
- Conceptele interne sunt învățate prin potrivirea ipotezelor generate de creier cu intrările senzoriale din mediu
- Învățarea apare prin testarea activă a ipotezelor prin acțiuni
- Creierul încearcă să potrivească intrările senzoriale externe cu stările generate intern

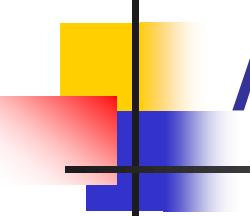
# Mașina Boltzmann

- Asemănătoare cu rețeaua Hopfield, dar:
  - Distinge între **noduri vizibile** (~ intrările senzoriale) și **noduri ascunse** (~ stările generate de creier)
  - Activările neuronilor sunt **stochastic**, nu deterministe



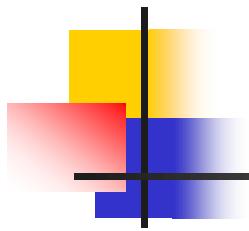
$$z_i = b_i + \sum_j s_j w_{ij}$$

$$\text{prob}(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$



# Antrenarea

- Energia rețelei are aceeași expresie ca la rețeaua Hopfield
- Probabilitatea unei stări este aceeași ca la modelul Ising
- Antrenarea presupune minimizarea energiei, ceea ce conduce la minimizarea diferenței dintre distribuția datelor de intrare și distribuția datelor generate de model pentru acele date de intrare
- Dacă  $T = 0$ , o mașină Boltzmann este echivalentă cu o rețea Hopfield



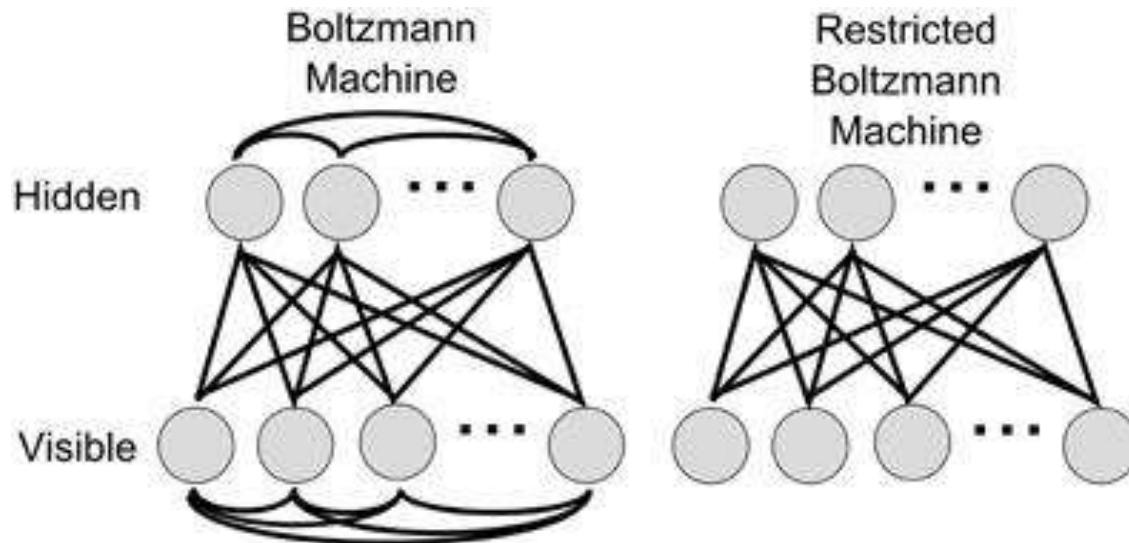
# Modele bazate pe energie

1. Modelul Ising
2. Rețele Hopfield
3. Mașini Boltzmann
- 4. Mașini Boltzmann restrictionate**
5. Rețele de convingeri profunde



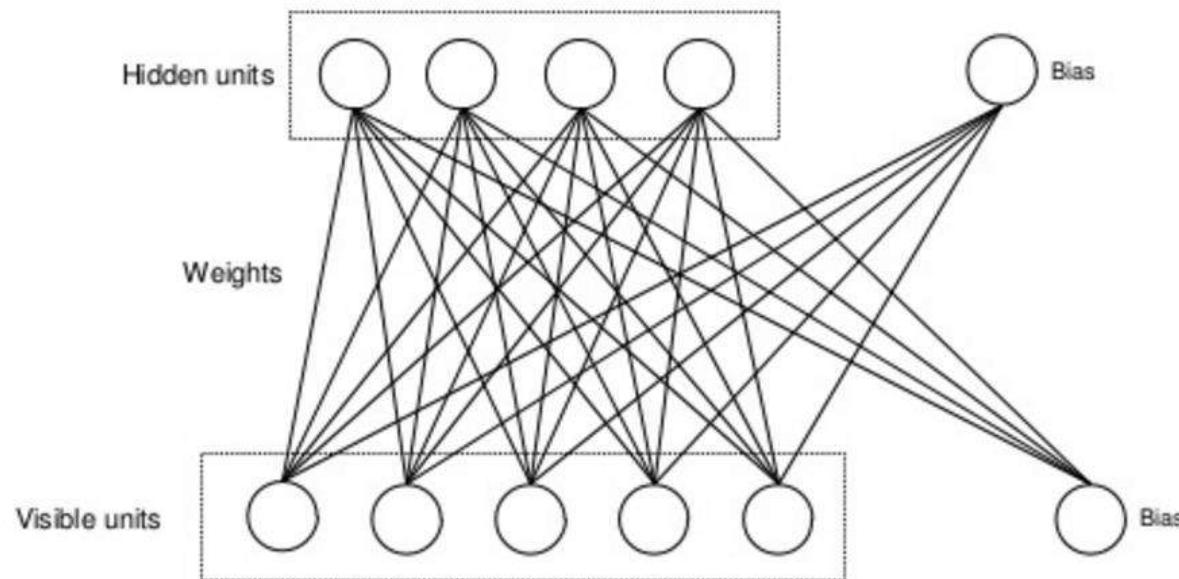
# BM vs. RBM

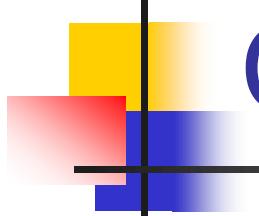
- O mașină Boltzmann restrictionată (*Restricted Boltzmann Machine*, RBM) este un **graf bipartit**: nu există conexiuni laterale între neuronii din același strat (vizibil sau ascuns)



# RBM

- Stratul vizibil:  $v \in \{0,1\}^N$ , stratul ascuns:  $h \in \{0,1\}^M$
- Pragurile neuronilor din stratul vizibil:  $a \in \mathbb{R}^N$
- Pragurile neuronilor din stratul ascuns:  $b \in \mathbb{R}^M$
- Ponderile conexiunilor dintre neuroni:  $W \in \mathbb{R}^{N \times M}$





# Caracteristici

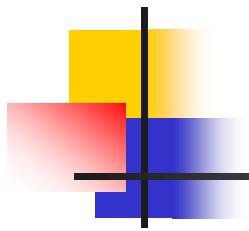
- Energia unei stări  $(v, h)$  este:

$$E(v, h) = -a^T v - b^T h - v^T Wh$$

- Probabilitatea unei stări  $(v, h)$  este:

$$P(v, h) = \frac{1}{Z} \exp\{-E(v, h)\}$$

$$Z = \sum_v \sum_h e^{-E(v, h)}$$



# Probabilitățile de activare

- Pornind de la aceste relații, se deduc probabilitățile de activare ale neuronilor, bazate pe funcția sigmoidă:

$$P(h_j = 1|v) = \frac{1}{1 + e^{-(b_j + \langle W_{\cdot j}, v \rangle)}} \\ \triangleq \sigma(b_j + \langle W_{\cdot j}, v \rangle) \text{ este o sigmoidă}$$

$$P(v_i = 1|h) = \sigma(a_i + \langle W_{i \cdot}, h \rangle)$$

- Pentru RBM, funcția sigmoidă nu se dă, ci este un rezultat al calculelor de probabilități!

# Antrenarea RBM

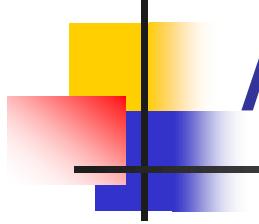
- Se bazează tot pe ideea MLE:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

$$\boxed{\quad = -\frac{\partial E}{\partial w_{ij}} = \frac{\partial (\sum a_i v_i + \sum b_j h_j + \sum v_i w_{ij} h_j)}{\partial w_{ij}} = v_i h_j}$$

$$\Delta w_{ij} = \epsilon (\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$$

- $\langle x \rangle_d$  este valoarea așteptată a lui  $x$  după distribuția  $d$
- $\epsilon$  este rata de învățare



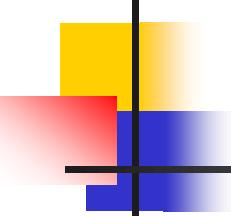
# Antrenarea RBM

- Primul termen, cu distribuția *data*, este ușor de obținut:

$$p(h_j = 1 \mid \mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij})$$

$$p(v_i = 1 \mid \mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij})$$

- Al doilea termen, cu distribuția *model*, este mai greu de obținut și necesită metode statistice



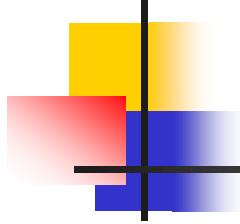
# Metode de eşantionare

- Metodele de eşantionare **Monte Carlo** pentru **lanțuri Markov** (*Markov Chain Monte Carlo, MCMC*) rezolvă problema eşantionării dintr-o distribuție de probabilitate prin construirea unui lanț Markov care are distribuția dorită ca distribuție de echilibru
- Una din metodele MCMC este **eşantionarea Gibbs** (*Gibbs sampling*), pentru probabilități condiționate

# Exemplu

- Să presupunem că starea vremii poate avea una din următoarele valori:
  - (Rain, Sunny, Cloudy)
- Probabilitățile de tranziție sunt de forma:
$$\begin{aligned} P(\text{ Rain tomorrow } | \text{ Rain today }) &= 0.5, \\ P(\text{ Sunny tomorrow } | \text{ Rain today }) &= 0.25, \\ P(\text{ Cloudy tomorrow } | \text{ Rain today }) &= 0.25 \end{aligned}$$
- Matricea completă de tranziții este:

$$\mathbf{P} = \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0.25 & 0.5 \end{pmatrix} \quad \xleftarrow{\text{prima linie}}$$



# Exemplu

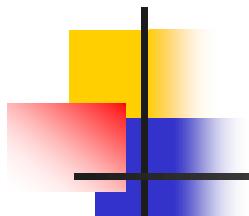
- Dacă astăzi este soare, cum va fi vremea peste 2 zile?

$$\pi(0) = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

$$\pi(2) = \pi(0)\mathbf{P}^2 = \begin{pmatrix} 0.375 & 0.25 & 0.375 \end{pmatrix}$$

- Dar peste 7 zile?

$$\pi(7) = \pi(0)\mathbf{P}^7 = \begin{pmatrix} 0.4 & 0.2 & 0.4 \end{pmatrix}$$



# Distribuția staționară

- Pentru  $n$  mare:

$$\mathbf{P}^n = \begin{pmatrix} 0.4 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.4 \end{pmatrix}$$

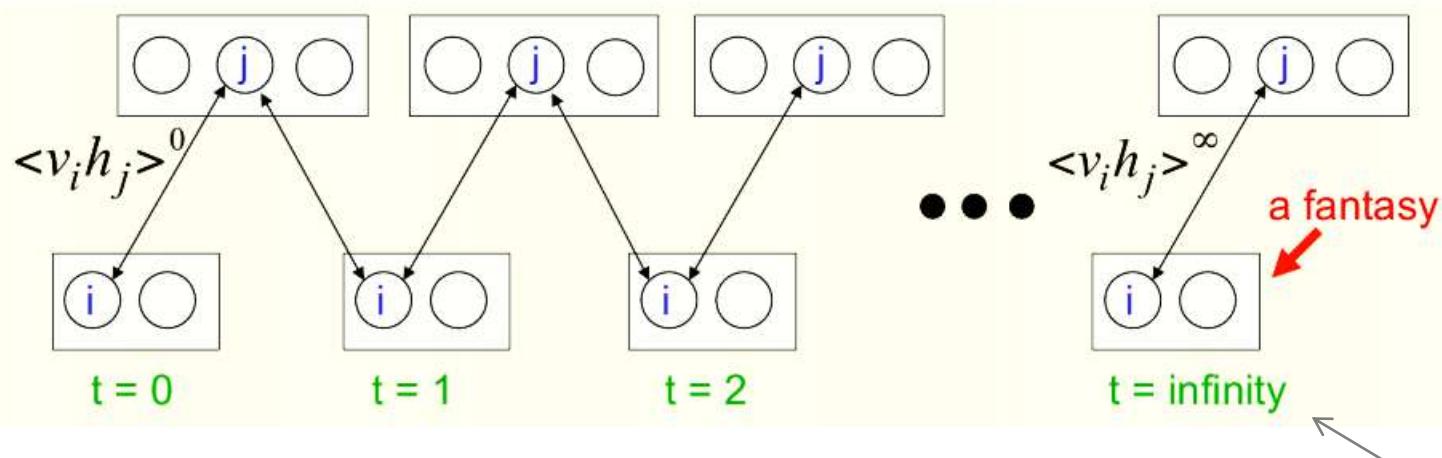
- Oricare ar fi starea inițială  $\pi(0)$ , peste un număr mare de zile, probabilitățile vor fi:

$$\pi(n) = ( 0.4 \quad 0.2 \quad 0.4 )$$

- Se spune că lantul Markov a ajuns la o **distribuție staționară**, în care probabilitățile sunt independente de valorile de start

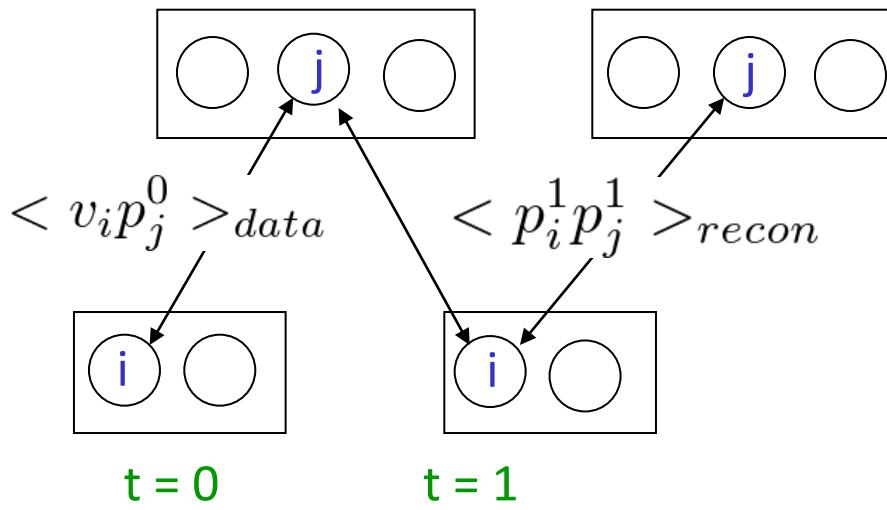
# Antrenarea RBM

- Se folosește eșantionarea Gibbs pentru a estima  $\langle v_i h_j \rangle_{model}$



În distribuția staționară, aici nu mai este vectorul de intrare, ci ce crede rețeaua că e vectorul de intrare!

# Antrenarea RBM



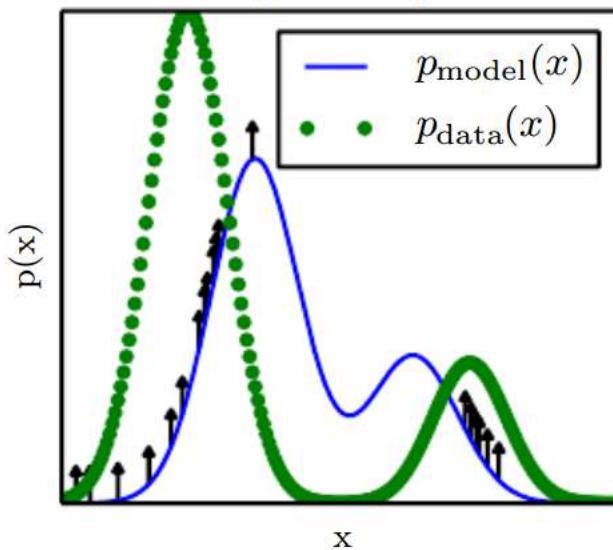
## Faza pozitivă

Propagarea vizibil → ascuns  
Învățarea datelor  
(~ rețeaua este trează)

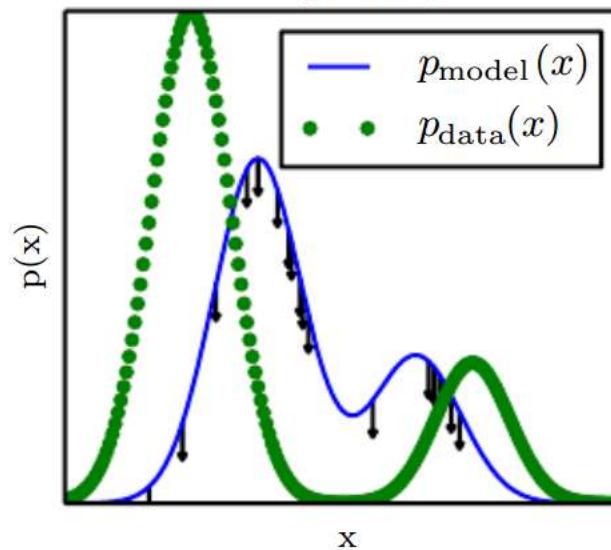
## Faza negativă

Propagarea ascuns → vizibil  
Reconstrucția datelor  
(~ rețeaua visează)

The positive phase



The negative phase



În faza pozitivă, se eșantionează puncte din distribuția datelor și acestea se ridică conform probabilităților lor nenormalizate. Punctele mai probabile din date sunt ridicate mai mult.

În faza negativă, se eșantionează puncte din distribuția modelului și acestea se coboară conform probabilităților lor nenormalizate. Acest fapt contracarează tendința fazei pozitive de a adăuga o constantă mare probabilităților nenormalizate de peste tot. Când distribuția datelor și distribuția modelului sunt egale, faza pozitivă are aceeași sansă de a ridica un punct pe cât are faza negativă să îl coboare. Atunci se termină antrenarea.

# Antrenarea RBM

$$P_j^{(0)} = \sigma \left( \sum_i v_i^{(0)} w_{ij} + b_j \right)$$

$$h_j^{(0)} \sim P_j^{(0)}$$

$$P_i^{(1)} = \sigma \left( \sum_j h_j^{(0)} w_{ij} + a_i \right)$$

$$v_i^{(1)} \sim P_i^{(1)}$$

$$P_j^{(1)} = \sigma \left( \sum_i v_i^{(1)} w_{ij} + b_j \right)$$

$$h_j^{(1)} \sim P_j^{(1)}$$

$$P_i^{(2)} = \sigma \left( \sum_j h_j^{(1)} w_{ij} + a_i \right)$$

$$v_i^{(2)} \sim P_i^{(2)}$$

...

$v^{(0)}$ , sunt datele de antrenare

$h_j \sim P_j$  înseamnă că activările  $h_j$  sunt eșantionate din distribuția de probabilitate  $P_j$ , analog  $v_i \sim P_i$

Autorul metodei, Hinton, recomandă folosirea probabilităților în loc de activările propriu-zise  $v_i$  și  $h_j$  (cu valori 0 sau 1), vezi slide-ul următor

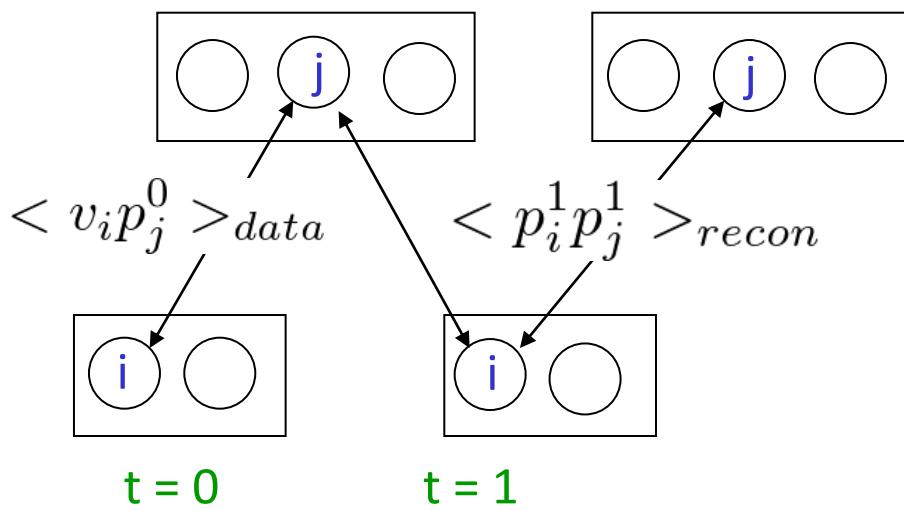
Rata de antrenare  $\epsilon$  poate fi, de exemplu, 0.1

$$\Delta w_{ij} = \epsilon \cdot \left( v_i^{(0)} h_j^{(0)} - v_i^{(n)} h_j^{(n)} \right)$$

$$\Delta a_i = \epsilon \cdot \left( v_i^{(0)} - v_i^{(n)} \right)$$

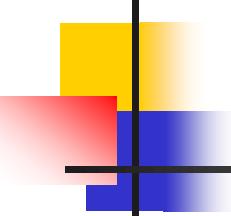
$$\Delta b_j = \epsilon \cdot \left( h_j^{(0)} - h_j^{(n)} \right)$$

# Antrenarea RBM: variantă rapidă



$$P_j^{(0)} = \sigma \left( \sum_i v_i^{(0)} w_{ij} + b_j \right)$$
$$P_i^{(1)} = \sigma \left( \sum_j h_j^{(0)} w_{ij} + a_i \right)$$
$$P_j^{(1)} = \sigma \left( \sum_i P_i^{(1)} w_{ij} + b_j \right)$$

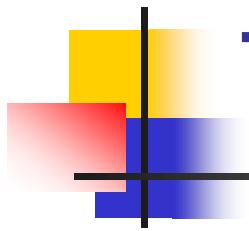
$$\Delta w_{ij} = \varepsilon \cdot \left( v_i P_j^{(0)} - P_i^{(1)} P_j^{(1)} \right)$$



# Divergența contrastivă

- engl. “**Contrastive Divergence**”, CD
- Este algoritmul de antrenare cel mai cunoscut pentru RBM
- Presupune aplicarea formulelor prezentate anterior pentru actualizarea ponderilor și pragurilor, pentru un număr specificat de epoci
- Eșantionarea Gibbs se face în  $k$  pași, dar de cele mai multe ori se consideră  $k = 1$
- CD nu urmează gradientul verosimilității maxime decât dacă  $k = \infty$ . CD minimizează în mod aproximativ **divergența Kullback-Leibler** (KL) care măsoară distanța între două distribuții de probabilitate (aici, între date și model):

$$D_{\text{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx, \quad D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.$$



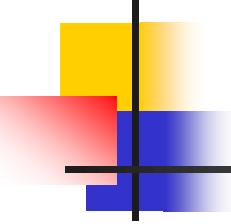
# Tipuri de RBM

- RBM Bernoulli-Bernoulli (intrări și ieșiri discrete)

$$E(\mathbf{v}, \mathbf{h}, \theta) = - \sum_{i=1}^N \sum_{j=1}^M w_{ij} v_i h_j - \sum_{i=1}^N a_i v_i - \sum_{j=1}^M b_j h_j$$

- RBM Gaussian-Bernoulli (intrări continue și ieșiri discrete)

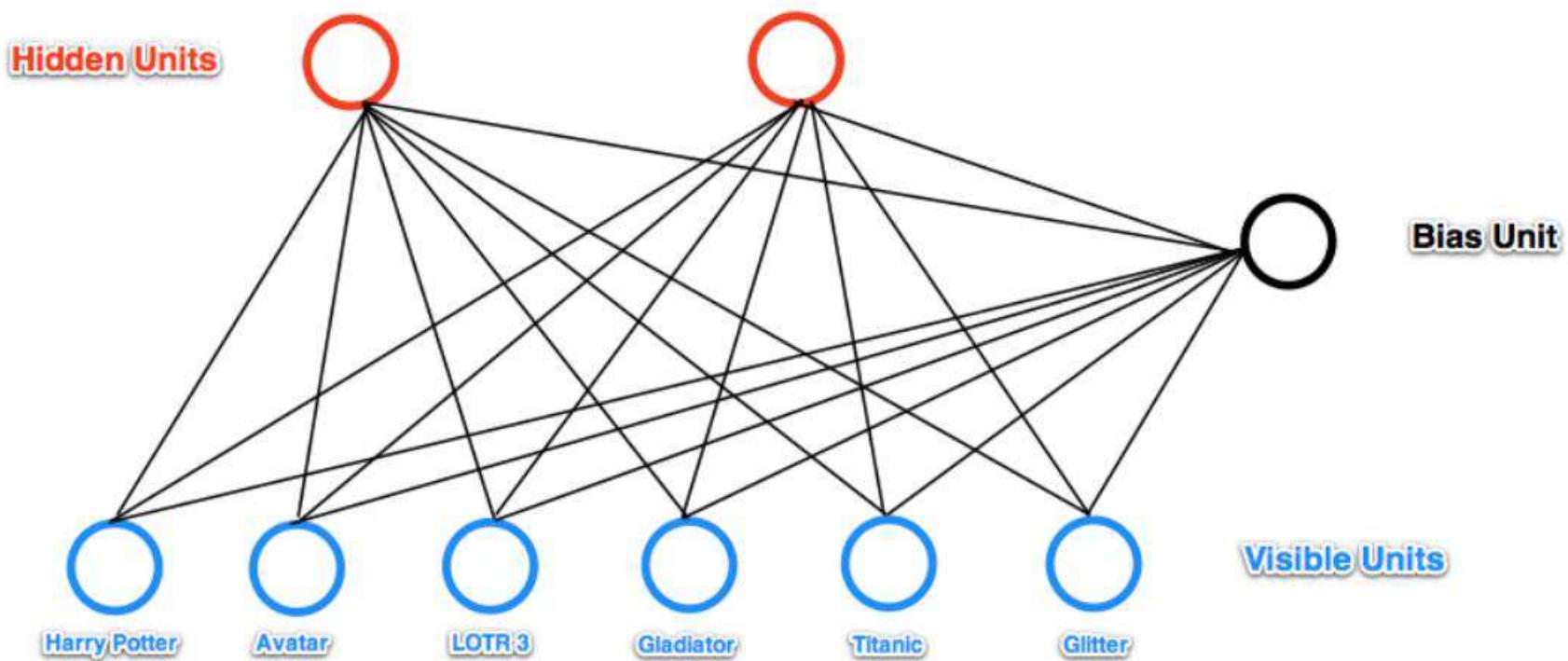
$$E(\mathbf{v}, \mathbf{h}, \theta) = - \sum_{i=1}^N \sum_{j=1}^M w_{ij} v_i h_j + \frac{1}{2} \sum_{i=1}^N (v_i - a_i)^2 - \sum_{j=1}^M b_j h_j$$

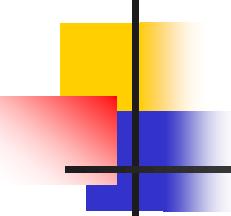


# Exemplu

- Trăsături „vizibile”:
  - Filme: Harry Potter, Avatar, LOTR 3, Gladiator, Titanic, Glitter
- Trăsături „ascunse”:
  - Există două trăsături ascunse, reprezentate de doi neuroni în stratul ascuns
  - Semnificația celor două trăsături nu este cunoscută a priori, ci apare după procesul de învățare
- Instante de antrenare:
  - Alice: (Harry Potter = 1, Avatar = 1, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). Fan SF/fantasy
  - Bob: (Harry Potter = 1, Avatar = 0, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). Fan SF/fantasy, dar nu-i place Avatar
  - Carol: (Harry Potter = 1, Avatar = 1, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). Fan SF/fantasy
  - David: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Fan filme Oscar
  - Eric: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Fan filme Oscar, cu excepția Titanic
  - Fred: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Fan filme Oscar

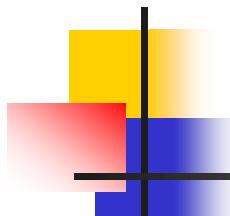
# Exemplu





# Exemplu

- Unitatea ascunsă 1 se activează la filmele câștigătoare ale premiilor Oscar
- Unitatea ascunsă 2 se activează la filmele SF/fantasy
- Pe baza preferințelor pentru cele 6 filme, se poate spune dacă o persoană este fan SF/fantasy și/sau fan filme Oscar
- Dacă se activează una din cele 2 unități ascunse, sau ambele, se generează o „persoană” cu anumite preferințe. De fiecare dată, preferințele generate sunt ușor diferite



# Exemplu

	Bias Unit	Hidden 1	Hidden 2
Bias Unit	-0.08257658	-0.19041546	1.57007782
Harry Potter	-0.82602559	-7.08986885	4.96606654
Avatar	-1.84023877	-5.18354129	2.27197472
LOTR 3	3.92321075	2.51720193	4.11061383
Gladiator	0.10316995	6.74833901	-4.00505343
Titanic	-0.97646029	3.25474524	-5.59606865
Glitter	-4.44685751	-2.81563804	-2.91540988

- Referință exemplu: <http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/>
- Implementări RBM: <https://github.com/echen/restricted-boltzmann-machines>, <https://gist.github.com/yusugomori/4428308>

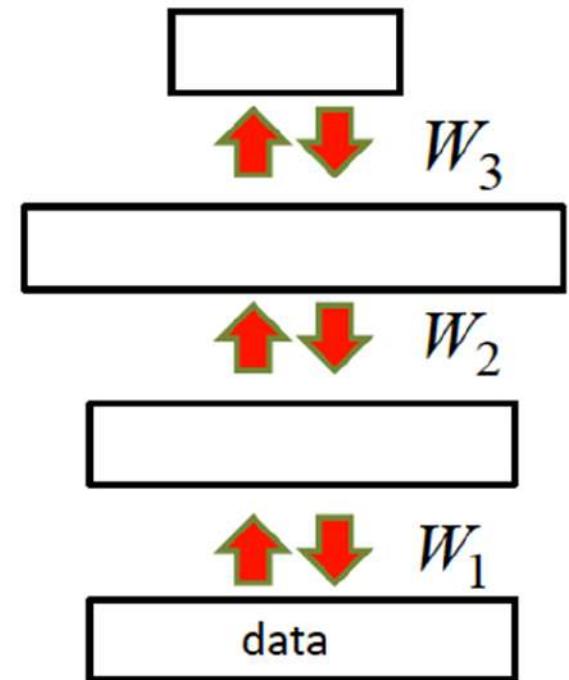
# Modele bazate pe energie

1. Modelul Ising
2. Rețele Hopfield
3. Mașini Boltzmann
4. Mașini Boltzmann restricționate
5. Rețele de convingeri profunde

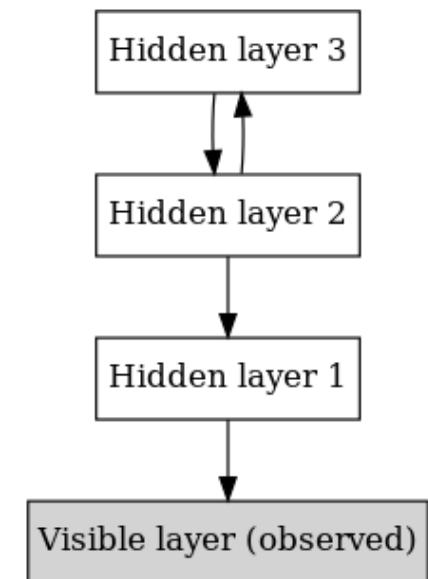
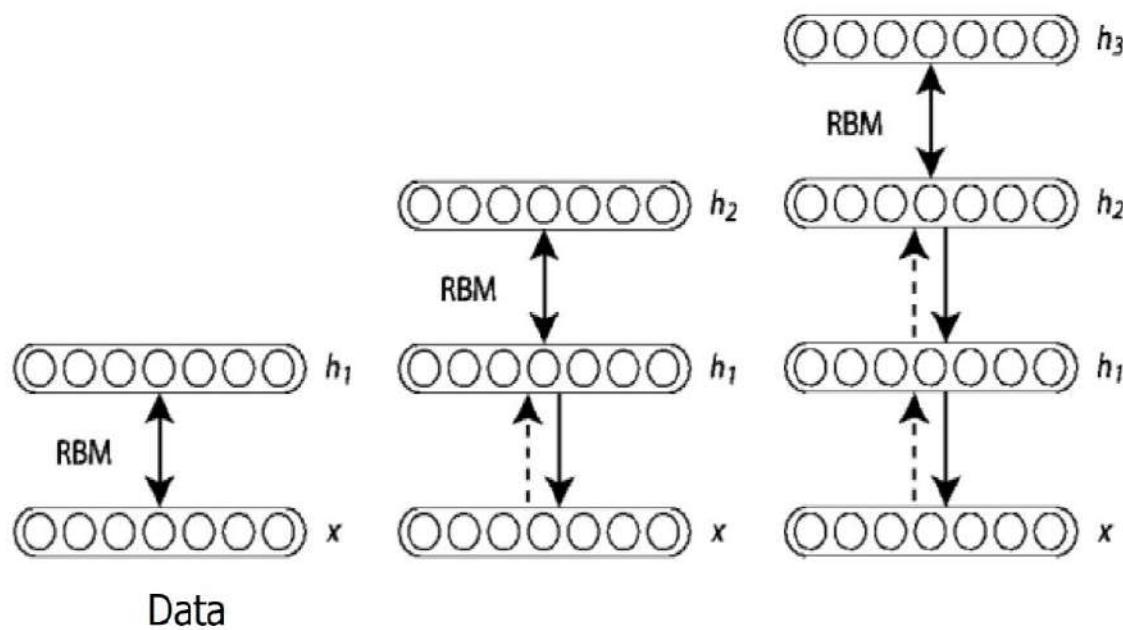


# Rețele de convingeri profunde

- engl. “**Deep Belief Networks**”, DBN
- RBM-urile sunt de obicei agregate în stive
- Antrenarea se face strat cu strat, folosind, de exemplu, algoritmul divergenței contrastive (CD)
- Odată ce un RBM este antrenat, alt RBM se pune în stivă deasupra sa
- Unitățile ascunse devin unități vizibile pentru stratul imediat superior
- Ponderile RBM-urilor antrenate anterior rămân fixe

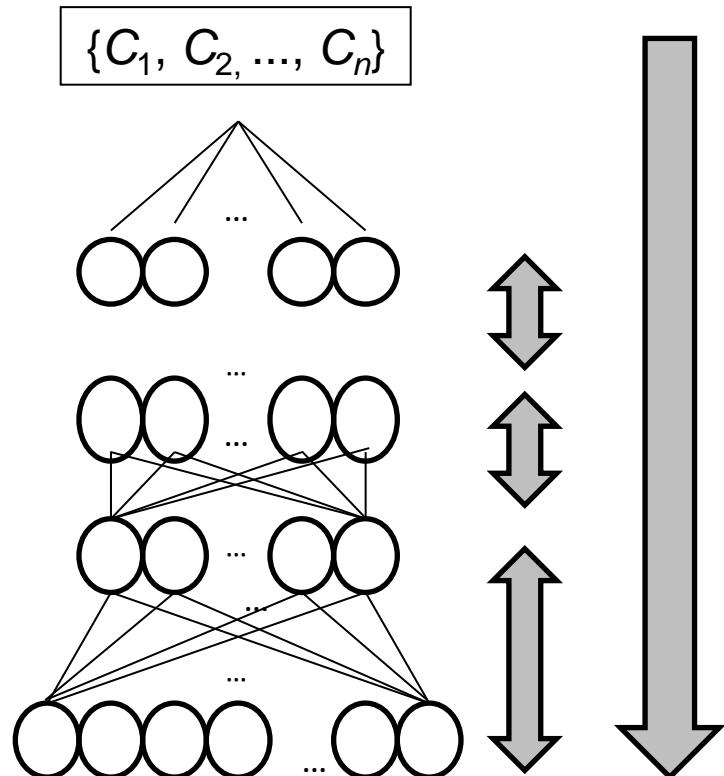


# Arhitectura DBN

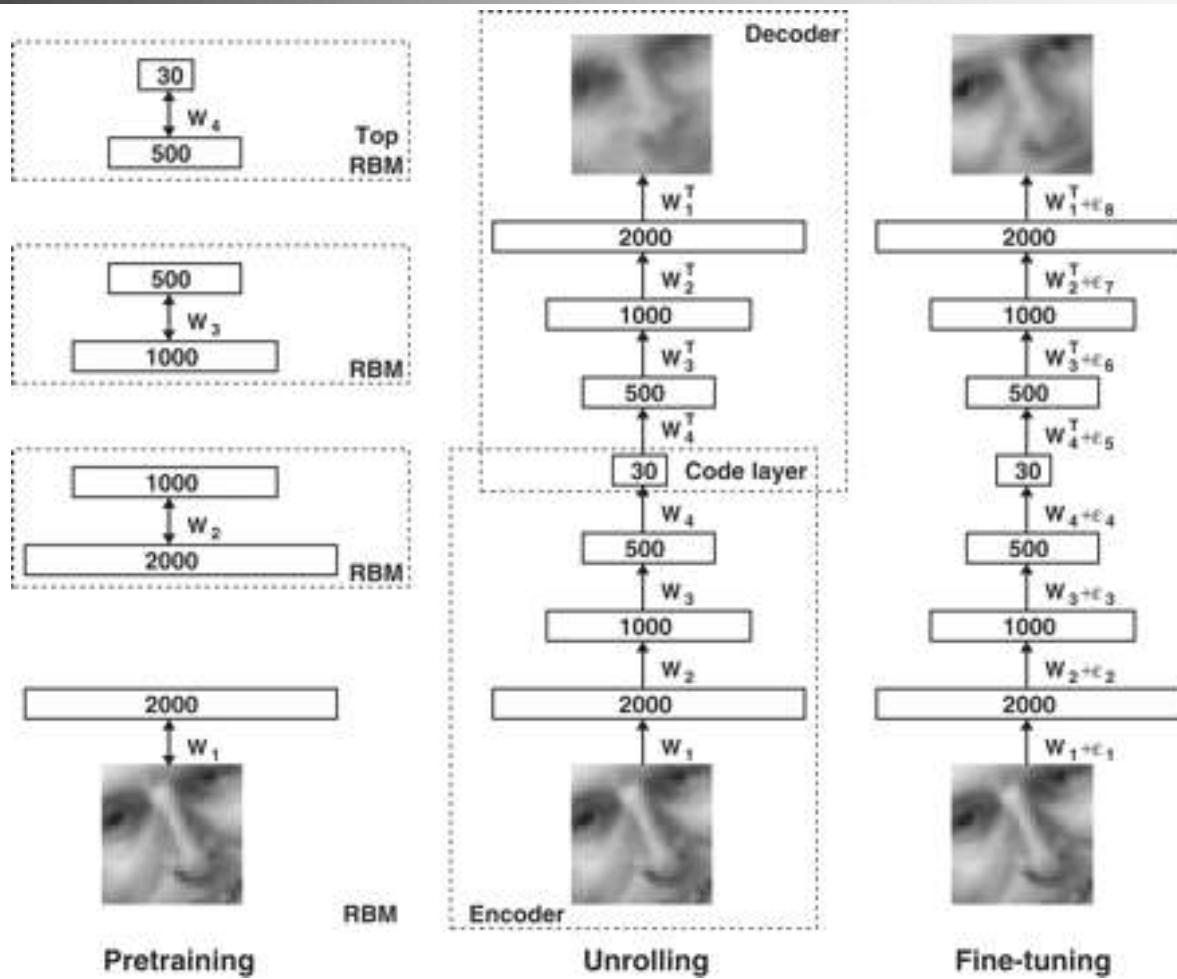


# Antrenarea DBN

- Ponderile se ajustează strat cu strat prin învățare nesupervizată
- Dacă scopul final este clasificarea, se antrenează un clasificator clasic pe nivelul cel mai de sus, prin învățare supervizată
- În final, toate ponderile sunt rafinate (*fine tuned*) cu algoritmul *backpropagation*

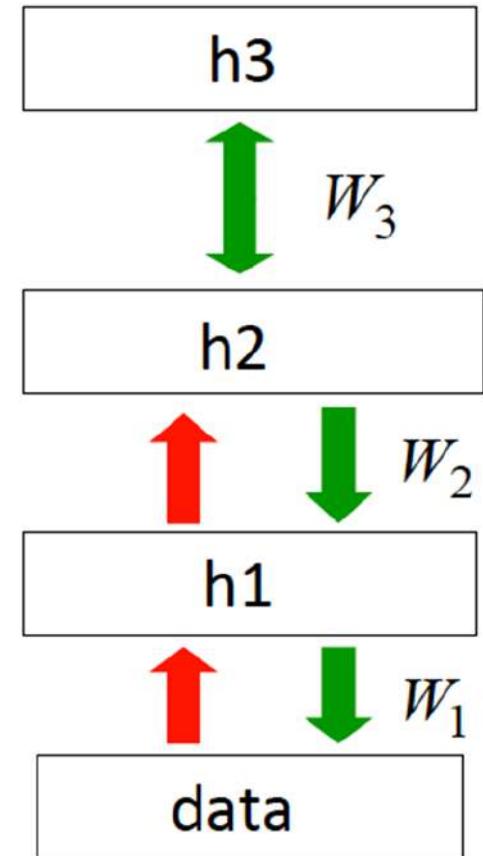


# Transformarea în autoencoder



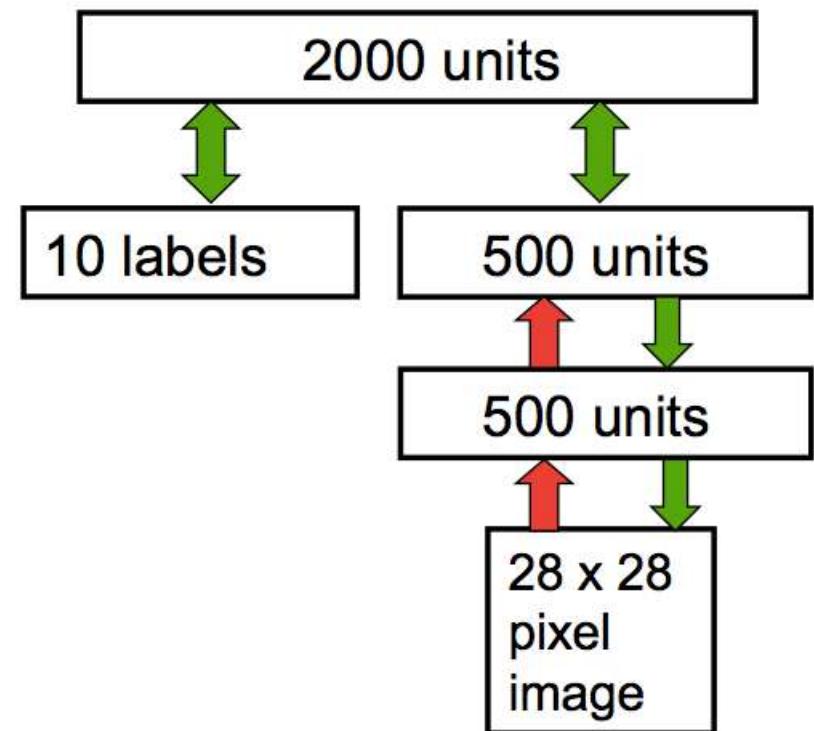
# Generarea din model

- Se obține un eșantion de echilibru din RBM-ul de la nivelul cel mai de sus efectuând eșantionarea Gibbs pentru un număr mare de pași (vezi slide-ul următor)
- Se face traversarea *top-down* cu activarea unităților de pe celelalte straturi
- Conexiunile *bottom-up* de pe nivelurile inferioare (marcate cu roșu) nu sunt folosite pentru generare, ci doar pentru inferență

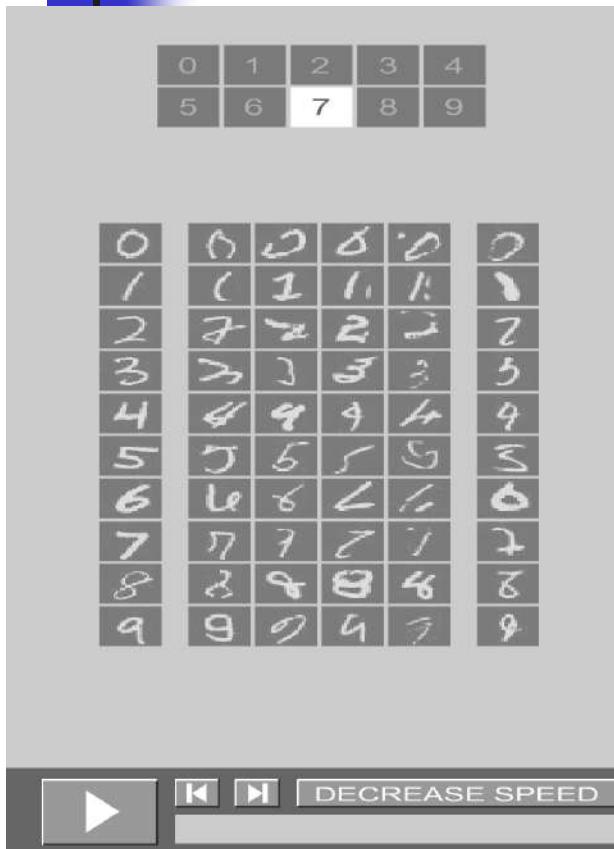


# Exemplu: recunoașterea cifrelor scrise de mâňă (MNIST)

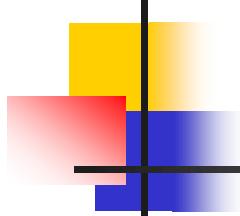
- Ultimele două niveluri superioare formează o memorie asociativă între reprezentările de nivel superior ale imaginilor cifrelor și etichetele (clasele) acestora
- Pentru recunoaștere, se pleacă de la imagini, se propagă activările în sus și se fac câteva iterații în memoria asociativă de la nivelul superior
- Pentru a genera imagini, se activează o unitate care corespunde unei etichete (clase)



# Exemplu: rezultatele DBN

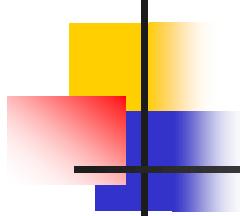


<http://www.cs.toronto.edu/~hinton/adi/index.htm>



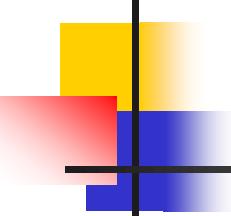
# Exemplu: MNIST

- Rate de eroare pe mulțimea de test
  - *Deep Belief Network*: 1.25%
  - *Support Vector Machine*: 1.4%
  - Perceptron multistrat antrenat cu *backpropagation*, cu 1000 neuroni într-un strat ascuns: ~1.6%
  - Perceptron multistrat antrenat cu *backpropagation*, cu 500, respectiv 300 de neuroni în două straturi ascunse: ~1.6%
  - *k-Nearest Neighbor*: 3.3%



# Discuție

- Autoencoderele simple sau *denoising autoencoders* nu sunt modele generative
- Se pot activa unitățile de pe nivelurile superioare ale ierarhiei pentru a produce rezultate pe nivelul cel mai de jos (corespunzător intrării), dar rezultatele sunt deterministe
- *Deep belief networks* și *variational autoencoders* sunt modele generative: rezultatele generate sunt de fiecare dată altele, deși asemănătoare



# Concluzii

- Rețelele Hopfield sunt memorii auto-asociative
- Mașinile Boltzmann pot fi văzute ca o variantă stochastică a rețelelor Hopfield
- Mașinile Boltzmann restrictionate (RBM) sunt grafuri bipartite, în care nu există conexiuni laterale între neuronii din același strat
- Mașinile Boltzmann restrictionate se antrenează de obicei cu algoritmul divergenței contrastive (CD)
- Rețele de convingeri profunde (DBN) reprezintă mașini Boltzmann restrictionate aggregate în stivă
- Rețele de convingeri profunde sunt modele generative



# Învățare automată

## 7. Aplicații complexe de învățare automată

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

[http://florinleon.byethost24.com/curs\\_ml.html](http://florinleon.byethost24.com/curs_ml.html)

# Aplicații complexe de învățare automată

1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe*
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network*
  - 2.3. *AlphaGo*



# Aplicații complexe de învățare automată

## 1. Vectori de cuvinte (*word embeddings*)

1.1. Modelul *word2vec*

1.2. Descompunerea valorilor singulare

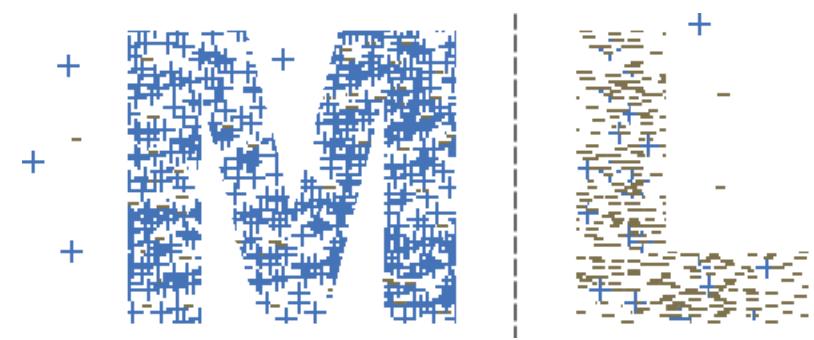
1.3. Modelul *GloVe*

## 2. Învățare cu întărire profundă

2.1. *TD-Gammon*

2.2. *Deep Q-Network*

2.3. *AlphaGo*



# Reprezentarea atomică a cuvintelor

- Fiecare cuvânt este reprezentat de un vector
- Lungimea vectorului este numărul de cuvinte din vocabular
- Vectorul are 1 pe poziția indexului unui cuvânt și 0 în rest

apple [0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0]

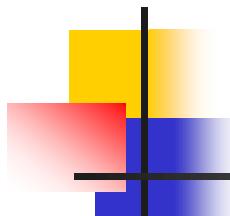
orange [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 ... 0 0 0 0 0]

car [0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0]

- Relațiile semantice sunt greu de determinat, nu există o măsură de similaritate

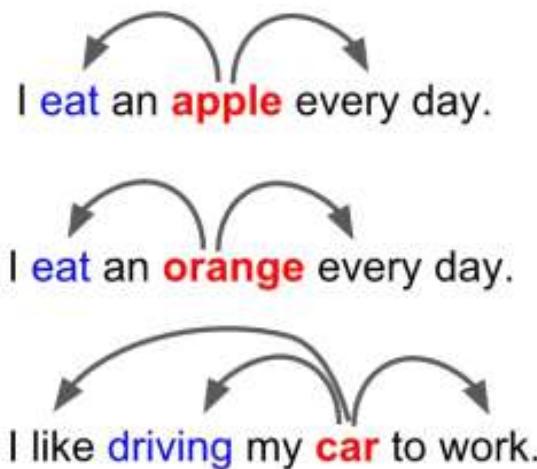
$$\text{apple} [0 \ 1 \ 0 \ \dots] \cdot \text{orange} [0 \ 0 \ 1 \ \dots]^T = 0$$

$$\text{apple} [0 \ 1 \ 0 \ \dots] \cdot \text{car} [1 \ 0 \ 0 \ \dots]^T = 0$$



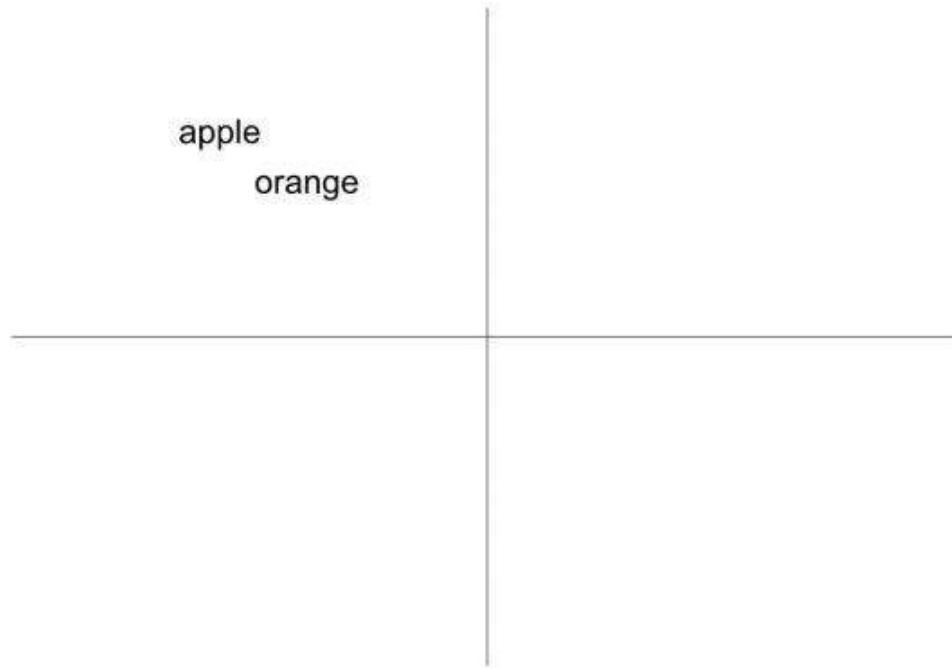
# Reprezentarea contextuală

- Nivelul de similaritate reiese din contextul în care sunt utilizate cuvintele
- Reprezentarea unui cuvânt se bazează pe vecinii săi



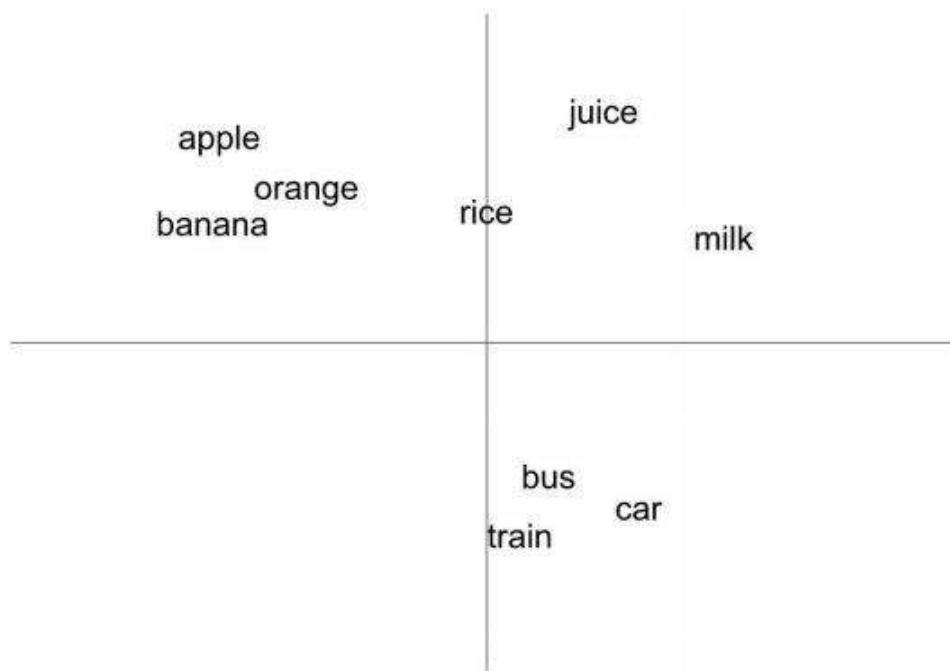
# Vectori de cuvinte (*Word Embeddings*)

- În această reprezentare vectorială și în proiecția ei într-un spațiu cu mai puține dimensiuni (de exemplu, 2D), conceptele similare din punct de vedere semantic trebuie să fie apropiate



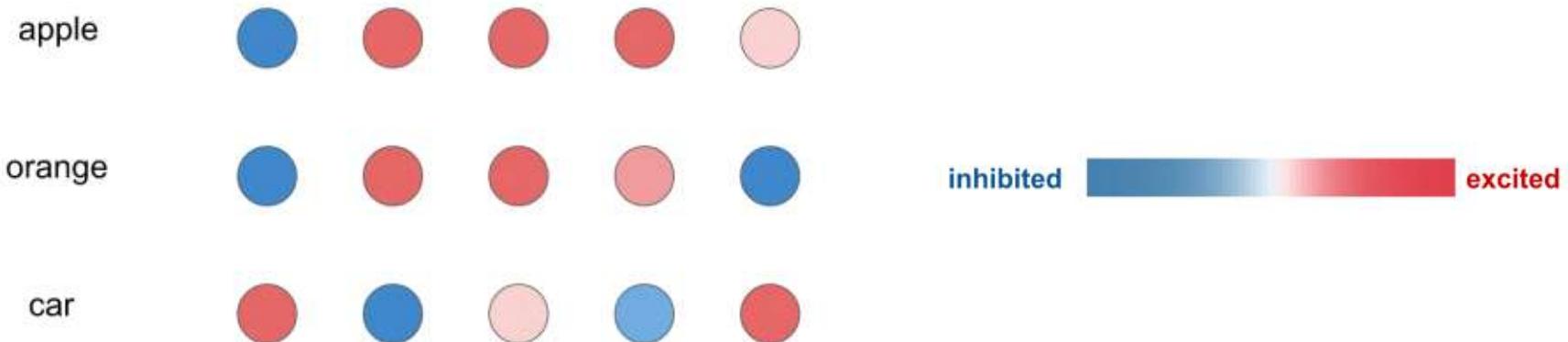
# Vectori de cuvinte (*Word Embeddings*)

- ... iar conceptele diferite din punct de vedere semantic să fie mai depărtate



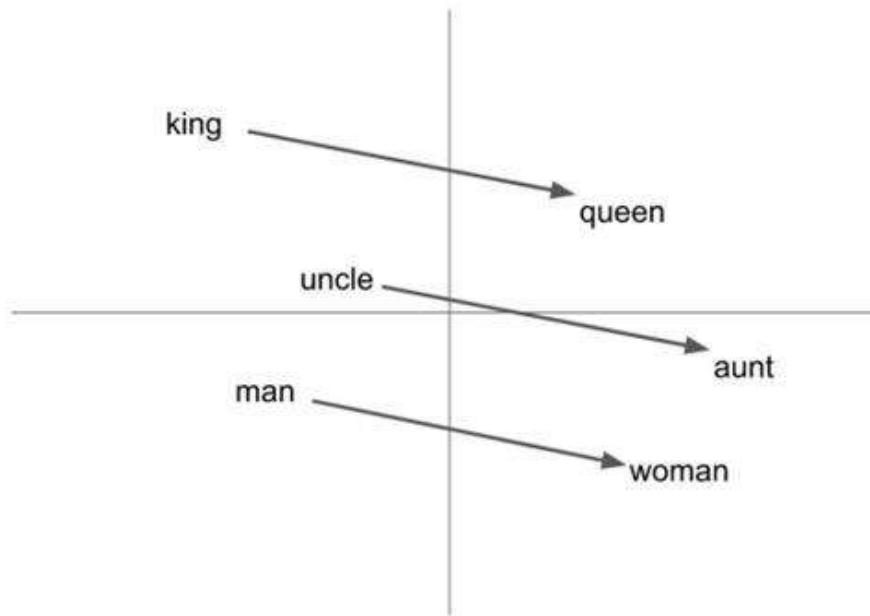
# Interpretarea cognitivă a vectorilor de cuvinte

- Conceptele determină diferite niveluri de activare ale neuronilor din creier
- Conceptele similare dau niveluri de activare asemănătoare
- Conceptele de *măr și portocală* sunt mai asemănătoare decât conceptul de *mașină*
- Un cuvânt poate fi reprezentat ca un vector de niveluri de activare

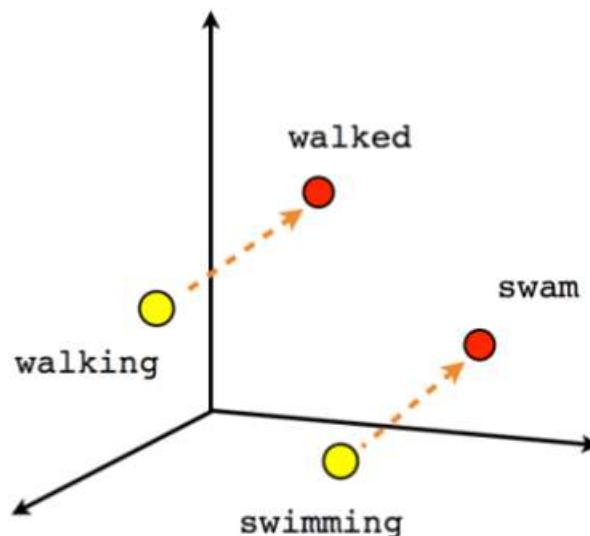


# Analogii

- Modelele recente (de exemplu, *word2vec*) determină vectori pentru cuvinte care încorporează relații semantice
  - $\text{king} - \text{queen} = \text{uncle} - \text{aunt}$
  - $\text{queen} = \text{king} + (\text{woman} - \text{man})$



# Exemple: analogii

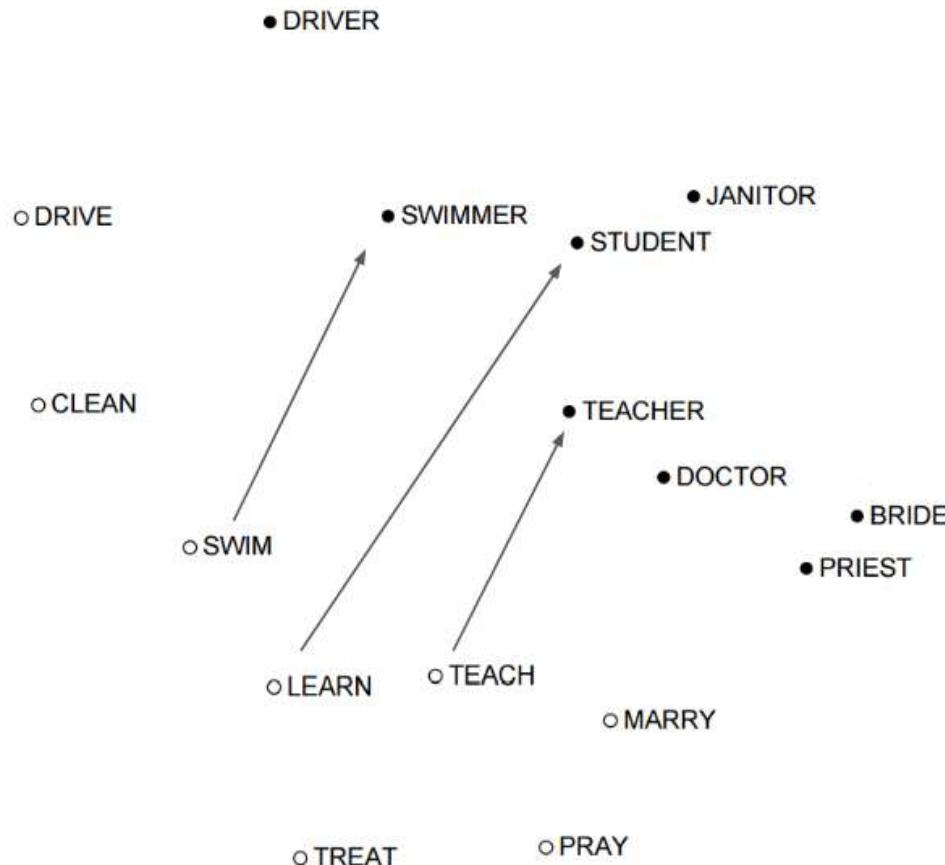


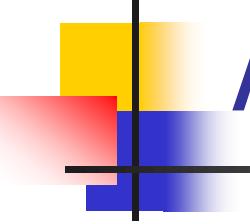
Verb tense

Spain	
Italy	
Germany	
Turkey	
Russia	
Canada	
Japan	
Vietnam	
China	

Country-Capital

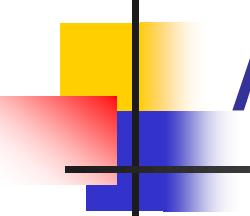
# Exemple: analogii





# Aplicații

- Traducere automată
- Parsare sintactică (*dependency parsing*)
  - Determinarea structurii sintactice a unei propoziții, crearea arborilor sintactici (identificarea subiectului, obiectului, atributelor acestora)
  - Utilizată pentru corectare gramaticală, răspunsuri la întrebări
- Recunoașterea entităților (*named-entity recognition*)
  - Clasificarea entităților dintr-un text în categorii predefinite
  - $[Jim]_{Person} \text{ bought } 300 \text{ shares of } [Acme Corp.]_{Organization} \text{ in } [2006]_{Time}$



# Aplicații

- Analiza sentimentelor
  - Determină atitudinea autorului unui text cu privire la un anumit subiect (de exemplu, pozitivă, negativă, neutră)
  - Se poate aplica direct o metrică de distanță între texte cunoscute și texte necunoscute
- Detectia parafrazelor
  - Determină dacă două propoziții au același înțeles
- Clasificarea documentelor
- Terminarea frazelor (*sentence completion*)

# Aplicații complexe de învățare automată

1. Vectori de cuvinte (*word embeddings*)

1.1. Modelul *word2vec*

1.2. Descompunerea valorilor singulare

1.3. Modelul *GloVe*

2. Învățare cu întărire profundă

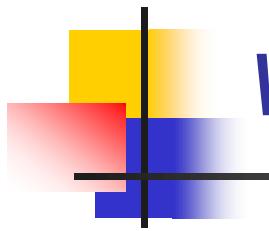
2.1. *TD-Gammon*

2.2. *Watson*

2.3. *Deep Q-Network*

2.4. *AlphaGo*

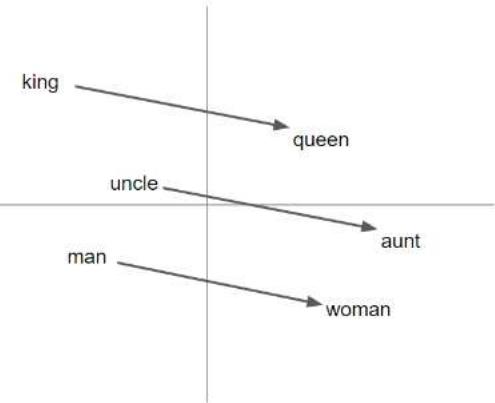
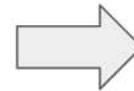
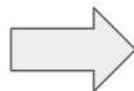




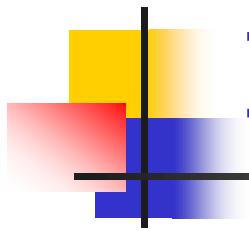
# *word2vec* (Google)



Wikipedia



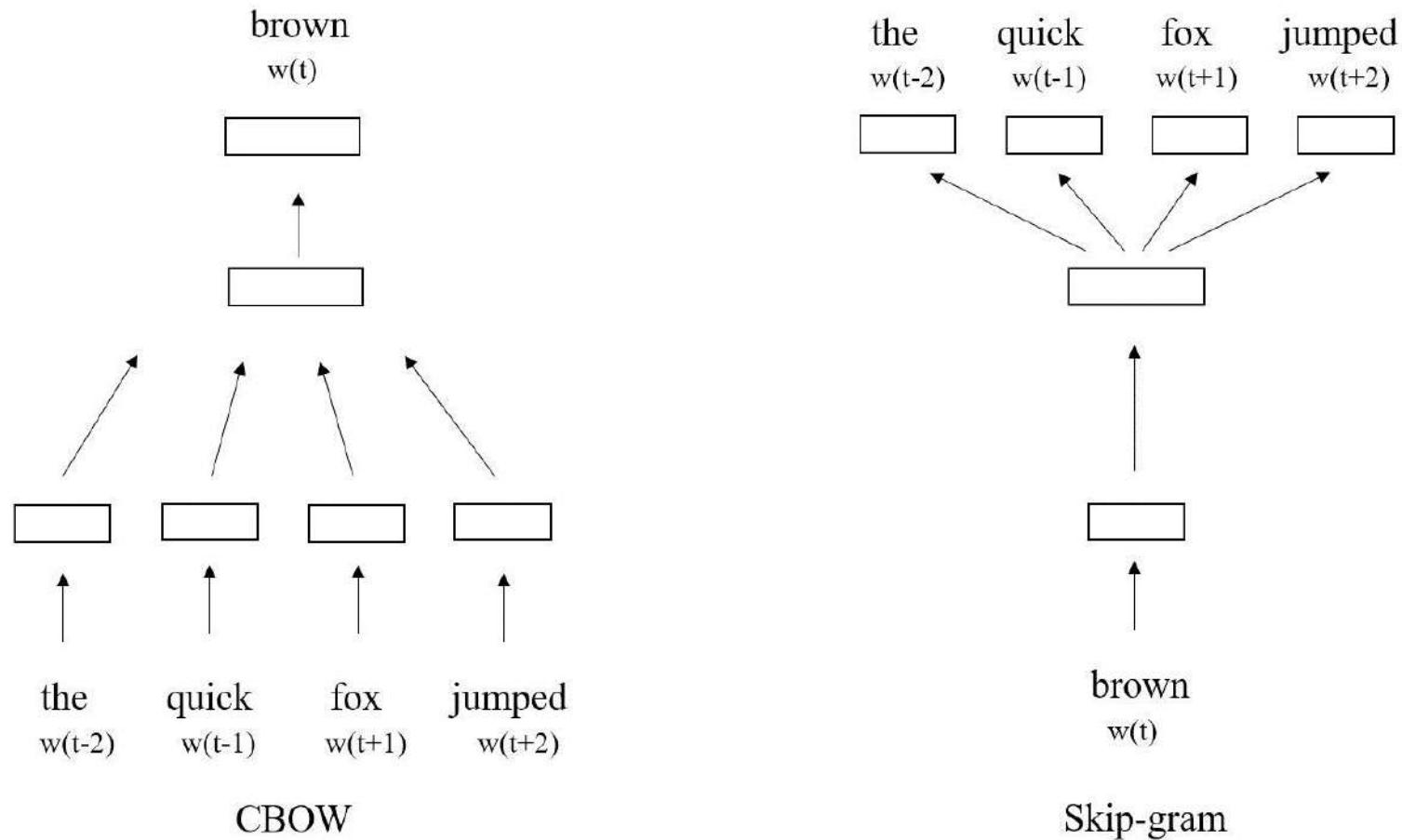
*word2vec* este unul din cele mai populare modele de *word embedding* (alături de *GloVe*, Stanford)



# Ideea de bază

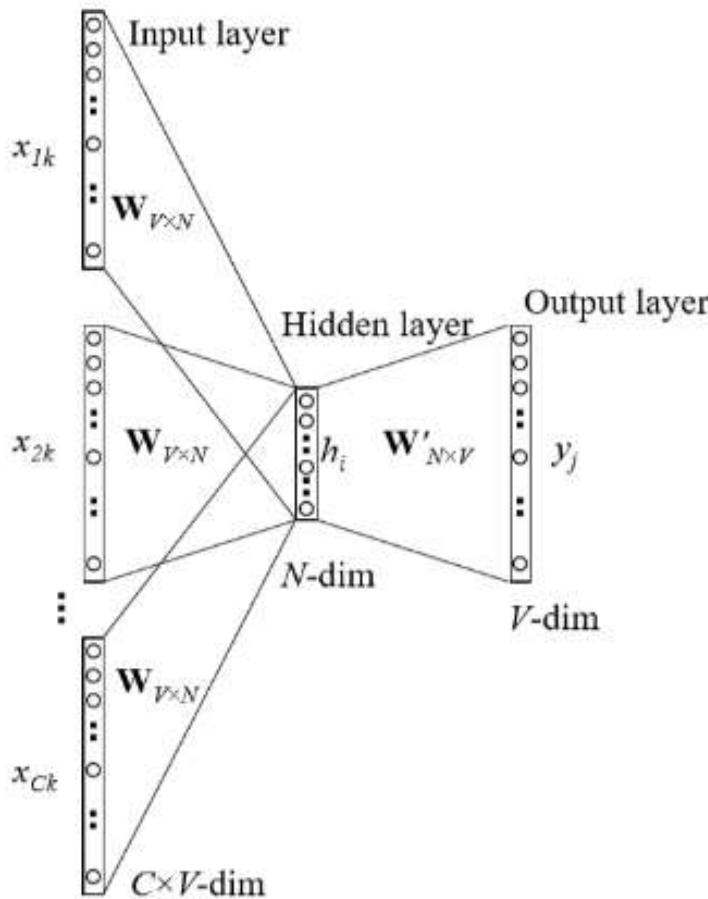
- Se definește un model care încearcă să lege un cuvânt de **contextul** în care apare acesta
  - Să prezică un cuvânt pe baza cuvintelor vecine („contextul”)  
sau
  - Să prezică un număr de cuvinte vecine pe baza unui cuvânt „central”

the quick brown fox jumped over the lazy dog



*Continuous Bag of Words*

# CBOW

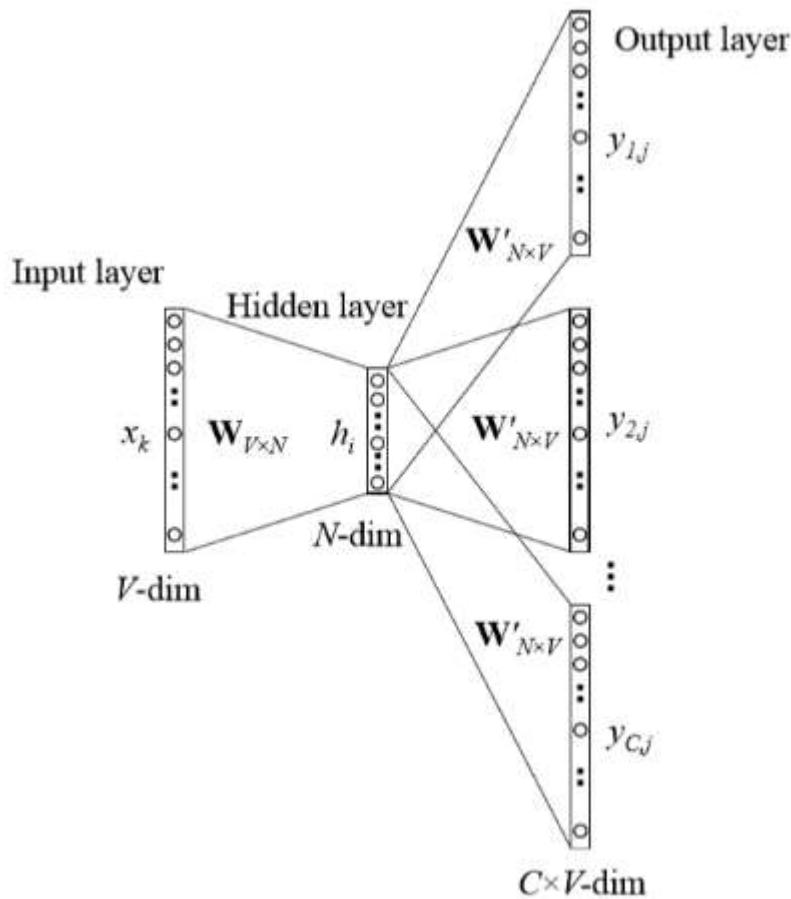


Se folosesc cuvintele care apar înainte și după un cuvânt țintă pentru a-l prezice

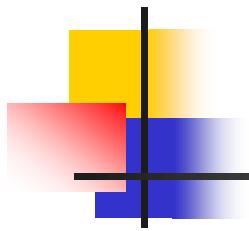
Ordinea cuvintelor nu contează

La intrare și la ieșire, cuvintele sunt reprezentate atomic (*one-hot*)

# Skip-gram



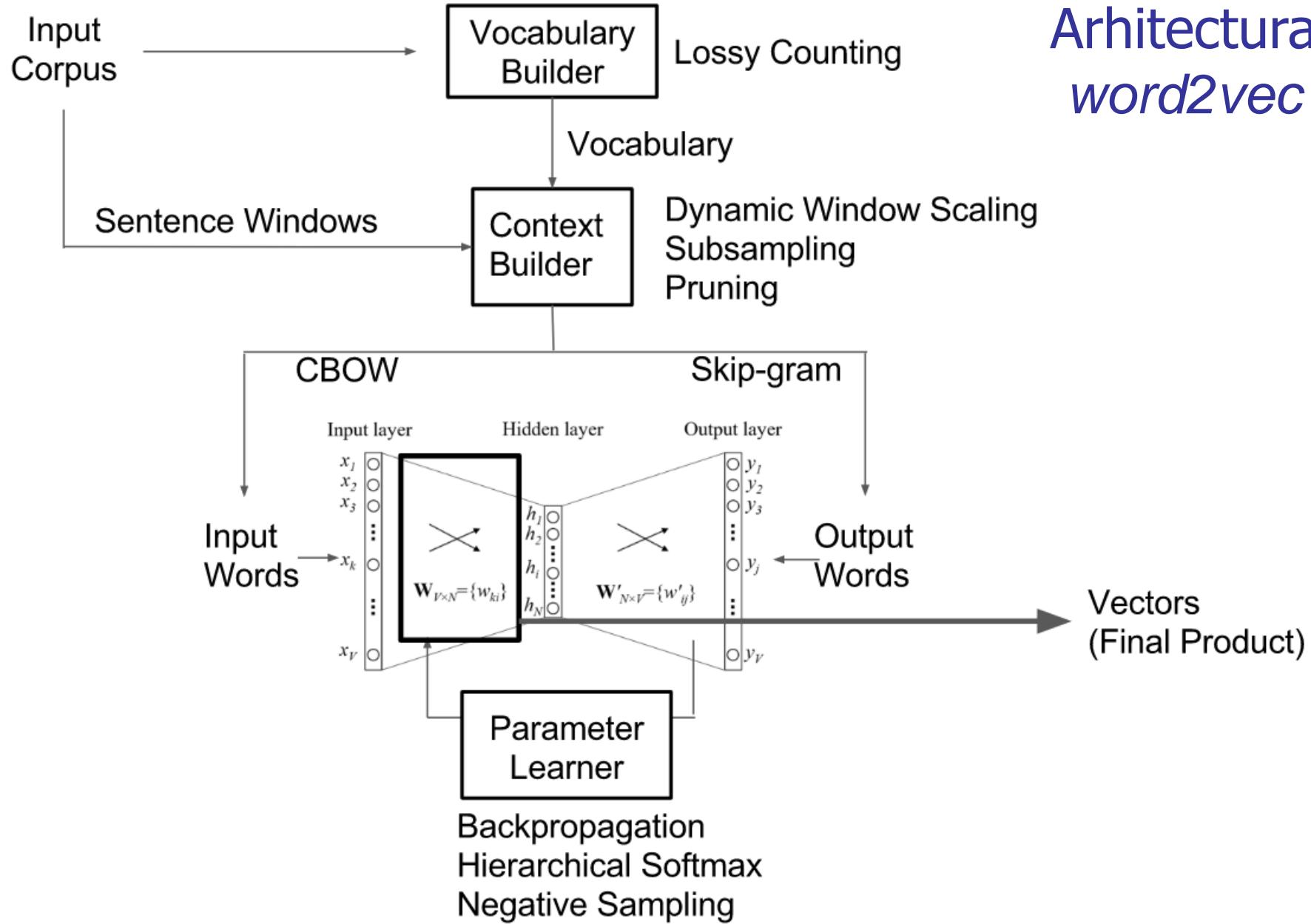
Se folosește cuvântul central  
pentru a prezice cuvintele  
vecine



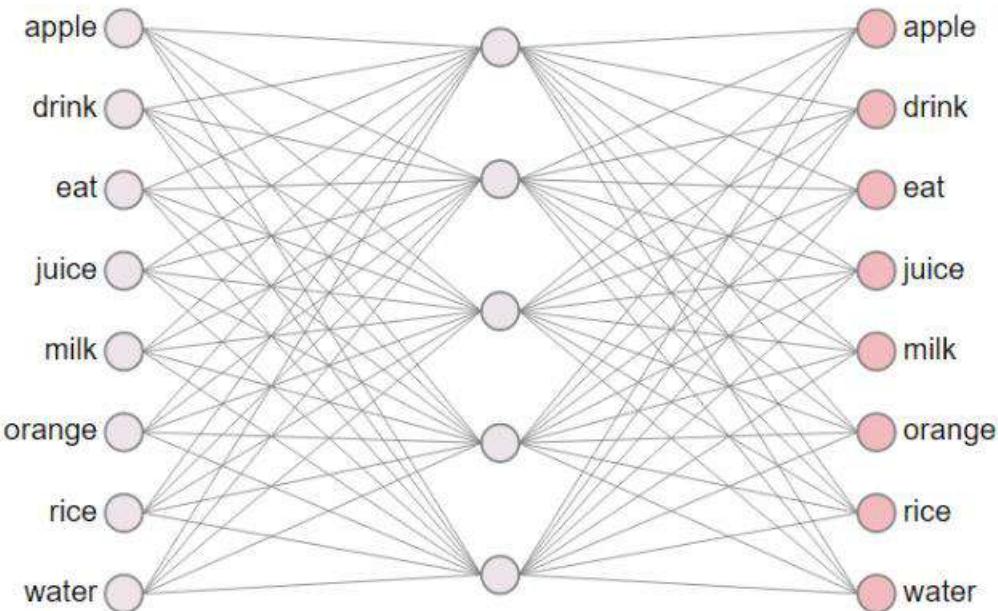
# Modelul *word2vec*

- Poate folosi atât varianta CBOW cât și Skip-gram
- În continuare, vom prezenta varianta Skip-gram
  
- Se dorește maximizarea produsul de probabilități de tipul  $P(w_{context} | w_t)$ , unde  $w_t$  este numit **cuvânt țintă (target)** sau **cuvânt central**
- Cuvintele de antrenare se iau dintr-un corpus, de exemplu, Wikipedia

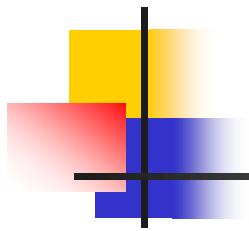
# Arhitectura word2vec



# Rețeaua neuronală



- Stratul de intrare: vectori *one-hot*
- Stratul ascuns: funcții liniare
- Stratul de ieșire: *softmax*

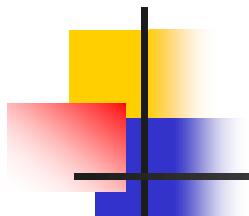


# Funcția obiectiv

- Pentru fiecare cuvânt cu indexul  $t$  din **textul sursă**, se prezic cuvintele vecine
- Funcția obiectiv presupune maximizarea probabilității tuturor cuvintelor de context, dat fiind cuvântul central corespunzător
- $\theta$  reprezintă vectorii cuvintelor pe care dorim să îi determinăm

$$J'(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

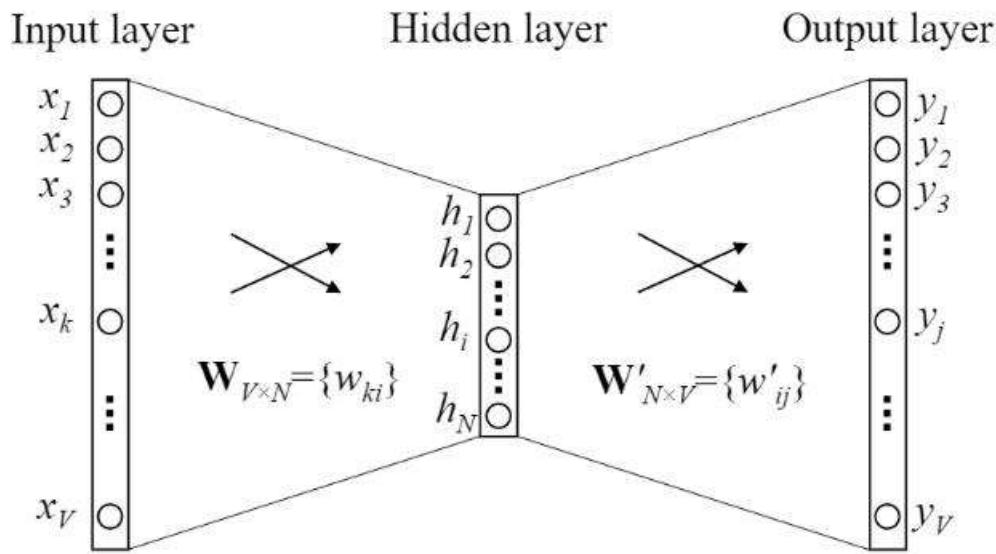


# Probabilitatea condiționată

- Fie  $o$  indexul unui cuvânt de context (*outside*),  $c$  indexul unui cuvânt țintă sau central,  $V$  este dimensiunea **vocabularului** (cuvintele distincte, spre deosebire de textul sursă), iar  $\mathbf{u}_o$  și  $\mathbf{v}_c$  sunt vectorii asociati cuvintelor cu indecsi  $o$ , respectiv  $c$
- Între vectori se realizează produsul scalar, care este o măsură de similaritate
- Expresia probabilității se bazează pe funcția softmax

$$P(w_o | w_c) = \frac{\exp(\mathbf{u}_o \cdot \mathbf{v}_c)}{\sum_{i=1}^V \exp(\mathbf{u}_i \cdot \mathbf{v}_c)}$$

# Rețeaua neuronală

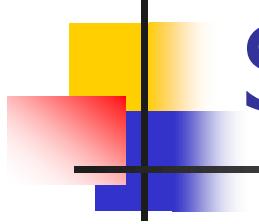


$$\mathbf{h} = \mathbf{W} \cdot \mathbf{x}_c = \mathbf{v}_c$$

$$\mathbf{y}_o = \text{softmax}(\mathbf{W}' \cdot \mathbf{h})$$

Un singur element al vectorului de intrare  $\mathbf{x}_c$  este 1

Fiecare coloană a matricei  $\mathbf{W}$  este o reprezentare a cuvântului de intrare:  $\mathbf{v}_c$

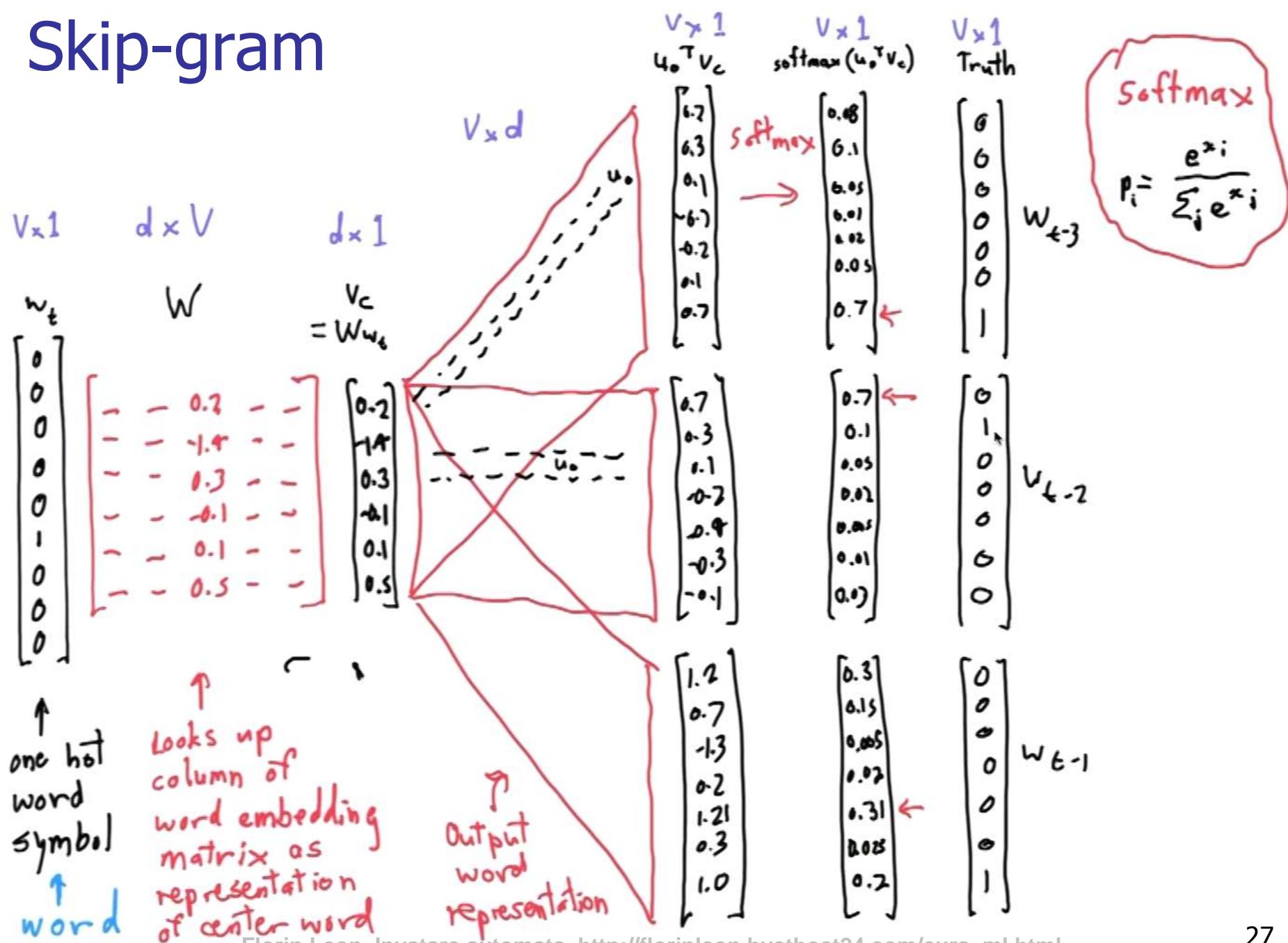


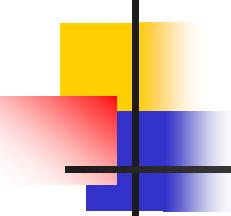
# Stratul ascuns

- Deoarece vectorii de intrare  $\mathbf{x}$  sunt *one-hot*, produsul cu matricea  $\mathbf{W}$  este echivalent cu selectarea unei coloane din matrice

$$\mathbf{W} (d \times V) \quad \mathbf{x} (V \times 1) \quad \mathbf{h} (d \times 1)$$
$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \end{vmatrix} \times \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} b \\ f \end{vmatrix}$$

# Skip-gram

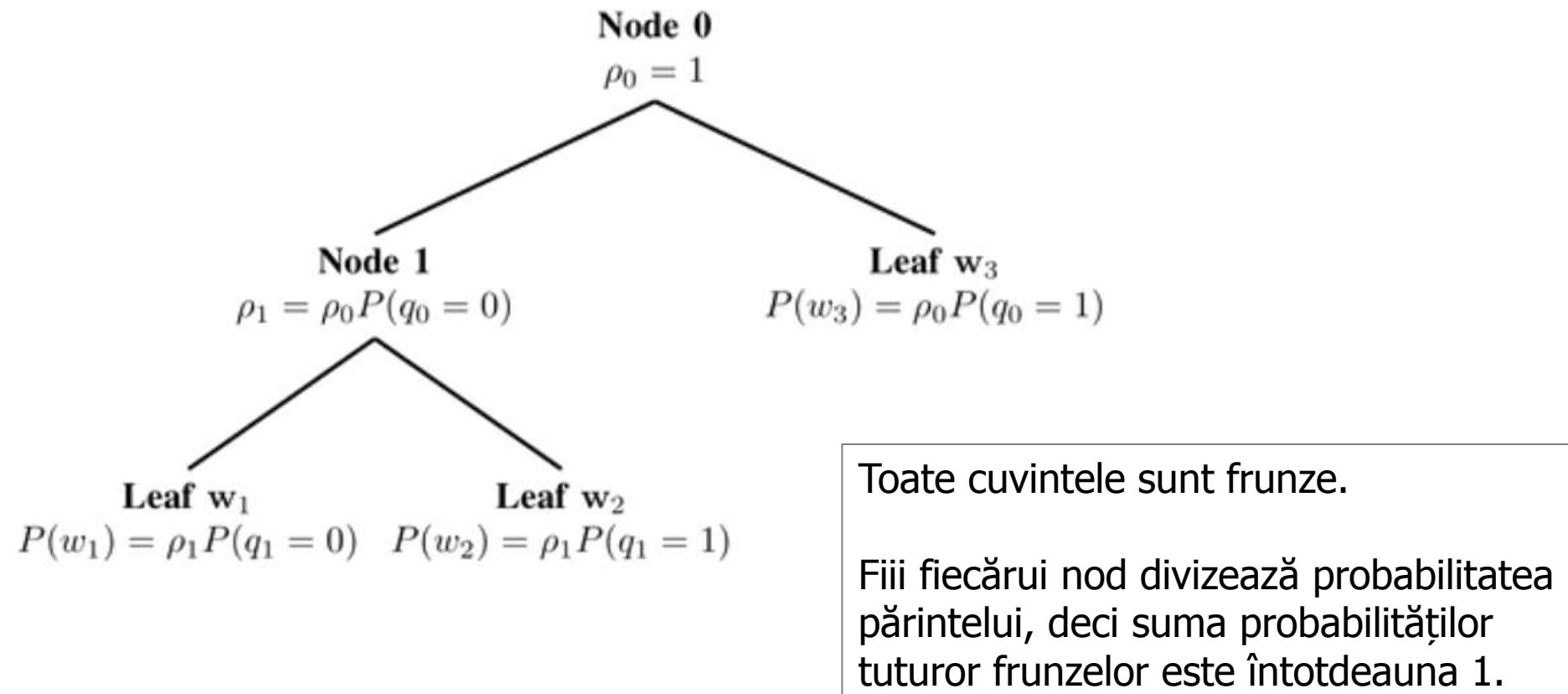


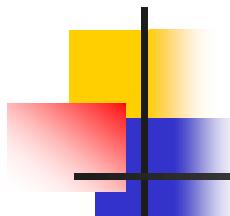


# Softmax ierarhic

- engl. “**hierarchical softmax**”
- Trebuie calculate probabilitățile (normalize) pentru un număr foarte mare de elemente
- Stratul softmax este înlocuit cu un arbore, care evită normalizarea
- Probabilitățile se calculează direct urmărind diferite căi în arbore
- Trebuie evaluate cel mult  $\log_2(|V|)$  noduri, unde  $|V|$  este numărul total de cuvinte din vocabular
- Empiric, s-a constatat o creștere de vitezei de 50 de ori în comparație cu metoda softmax standard

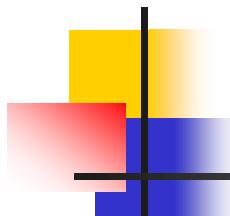
# Exemplu





# Eșantionarea negativă

- engl. “negative sampling”
- Nu toate cuvintele de ieșire contribuie la o actualizare
- La fiecare iteratie, cuvântul țintă este inclus împreună cu un număr mic de cuvinte non-țintă, care reprezintă eșantioanele negative



# Eșantionare suplimentară

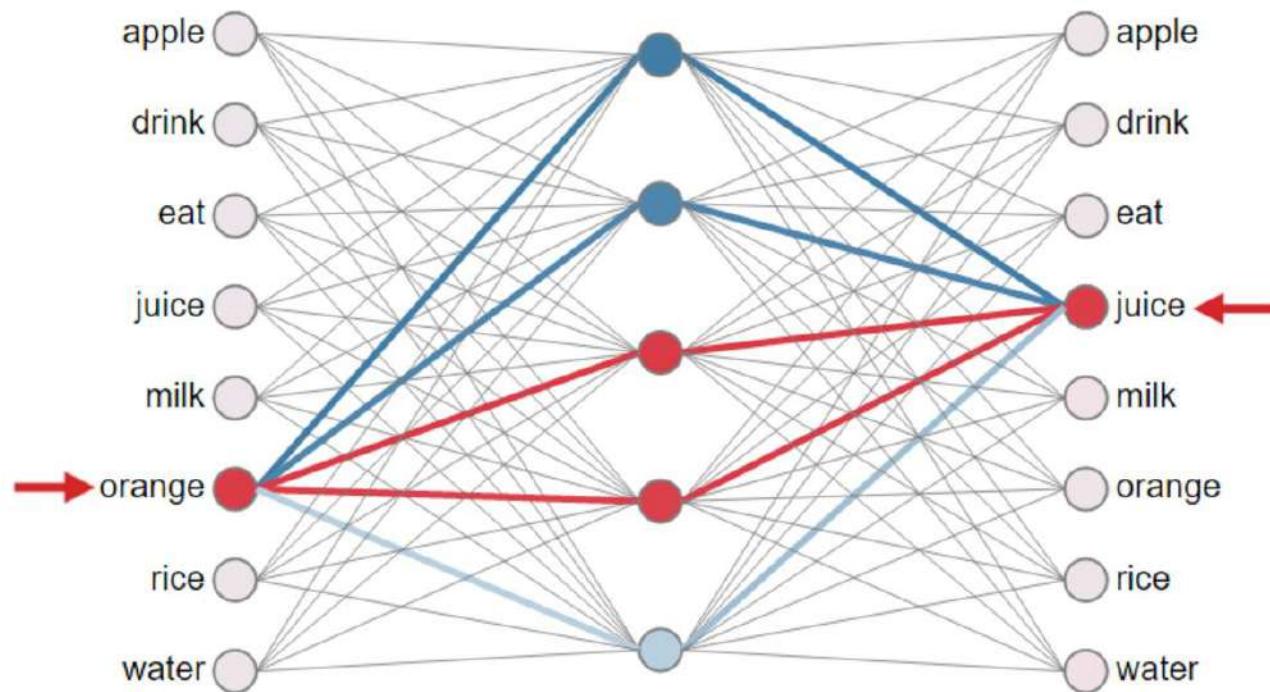
- Se mai folosește o sub-eșantionare pentru a echilibra influența cuvintelor rare și a celor frecvente: fiecare cuvânt  $w_i$  din mulțimea de antrenare este **ignorat** cu probabilitatea:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

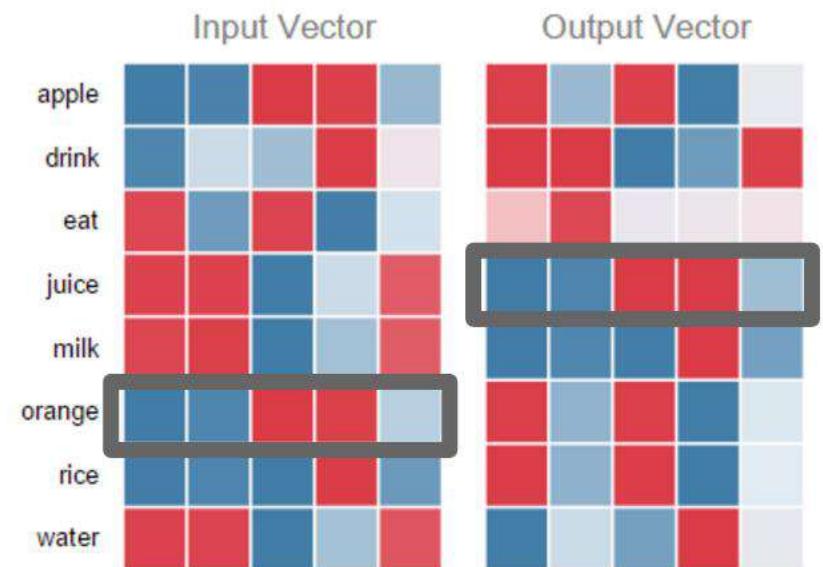
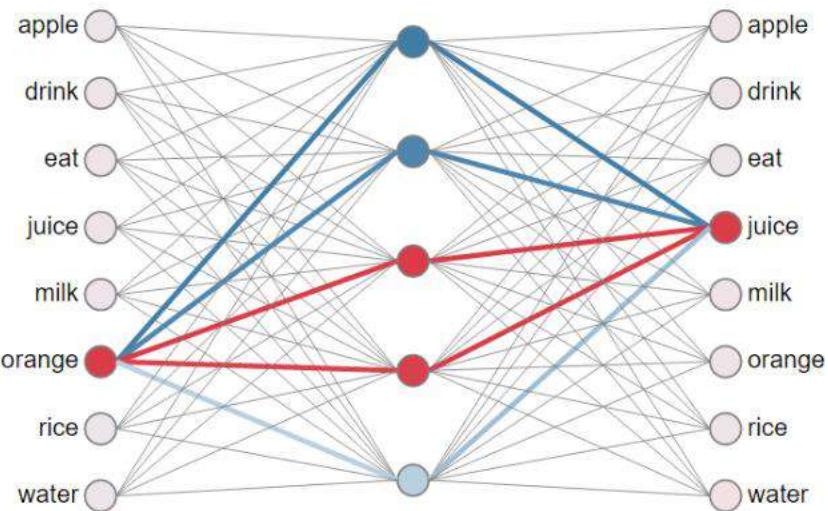
- unde  $t$  este un prag, de exemplu,  $10^{-5}$ , iar  $f(w_i)$  este frecvența cuvântului  $w_i$

# Exemplu

- Cuvântul țintă este “juice”: se actualizează vectorii pentru el și doar alte câteva cuvinte



# Exemplu: ponderile matricelor

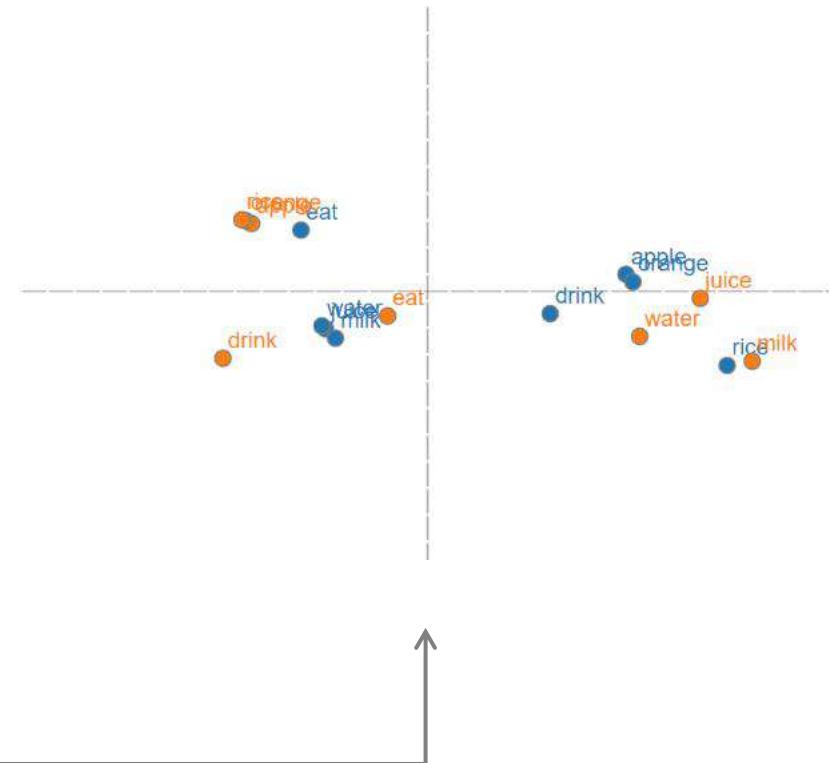
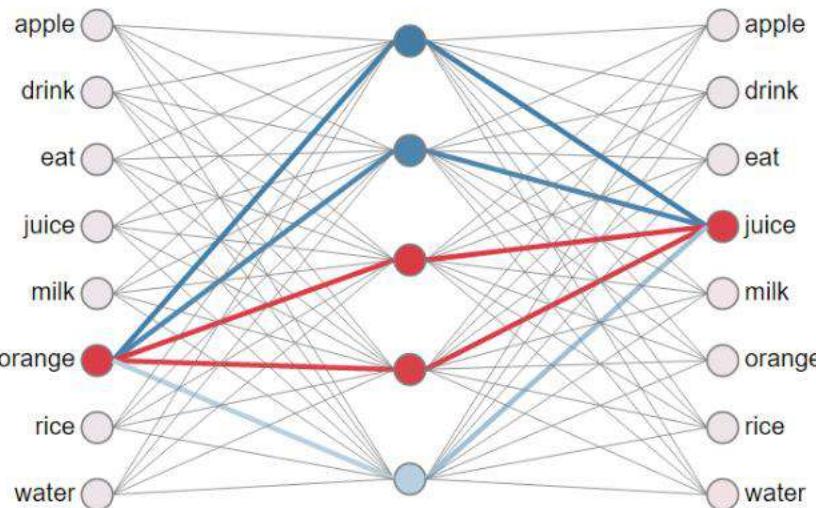


$W$

$W'$

inhibited      excited

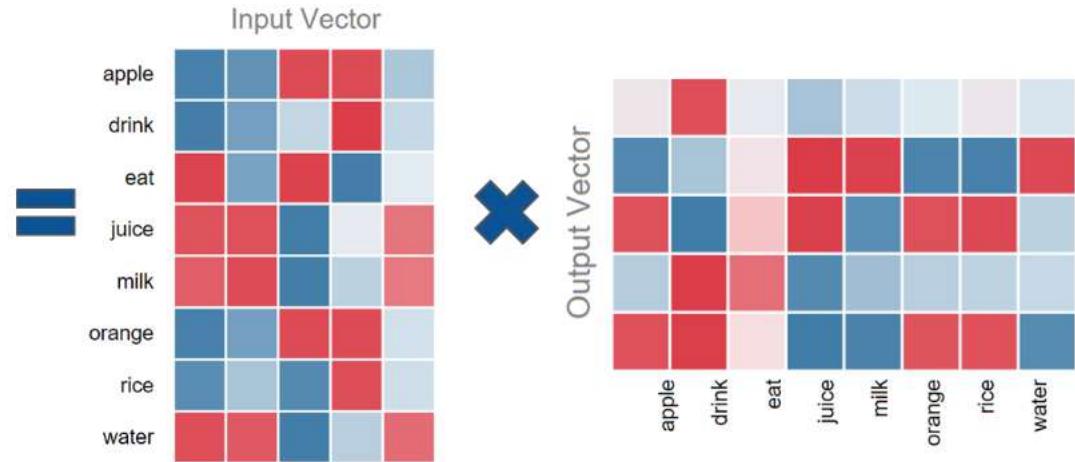
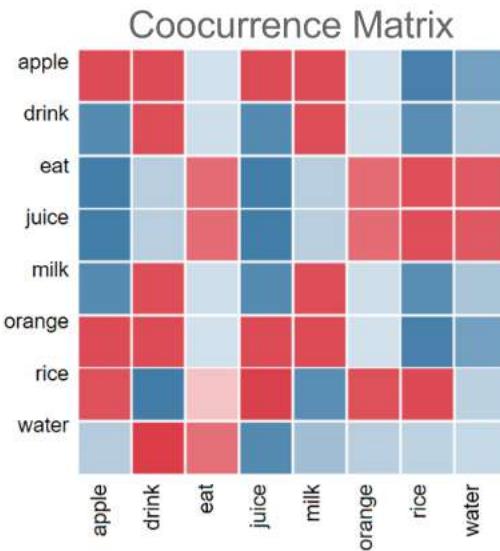
# Exemplu: proiecția 2D a vectorilor de cuvinte



aceste ponderi reprezintă vectorii de cuvinte care apoi se pot proiecta într-un spațiu cu mai puține dimensiuni, de exemplu, 2D

# Interpretarea modelului

- Vectorii de cuvinte pot fi văzuți și ca elementele unei factorizări a **matricei de co-ocurență** (co-apariții), care numără de câte ori apar împreună perechile de cuvinte



# Aplicații complexe de învățare automată

1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe*
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network*
  - 2.3. *AlphaGo*



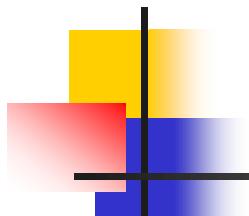
# Exemplu: construirea matricei de co-ocurență

- Contextul este cuvântul anterior și cel următor

Corpus:

I like deep learning. I like NLP. I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0



# Reducerea dimensionalității cu SVD

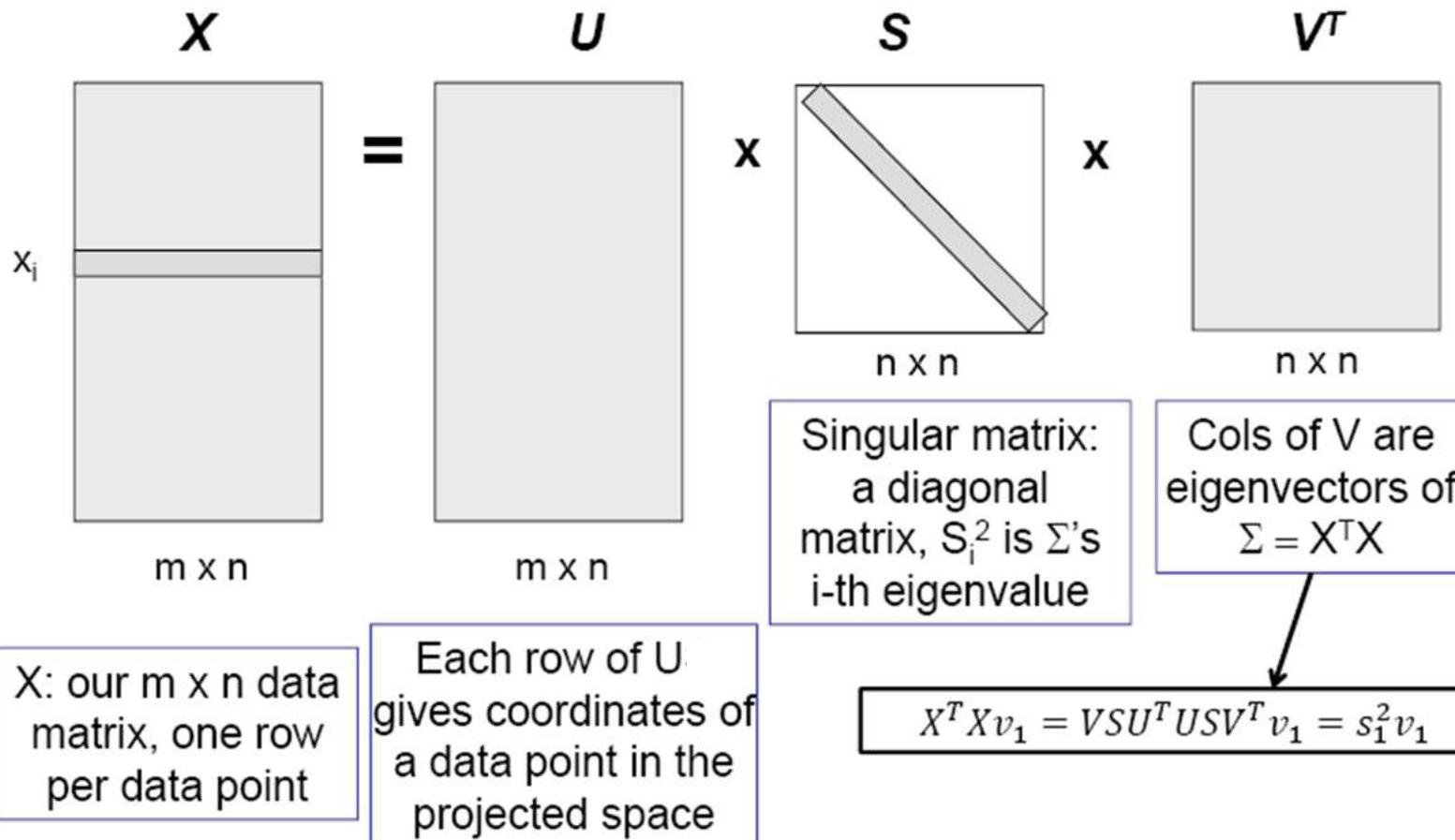
- Descompunerea valorilor singulare (*Singular Value Decomposition*, SVD) este una din cele mai populare metode de reducere a dimensionalității
- În domeniul prelucrării limbajului natural, se poate aplica asupra matricei de co-ocurență

$$\begin{matrix} n \times k \\ \boxed{\phantom{000}} \end{matrix} = \begin{matrix} n \times k \\ \boxed{\phantom{000}} \end{matrix} \begin{matrix} k \times k \\ \boxed{\phantom{000}} \end{matrix} \begin{matrix} k \times k \\ \boxed{\phantom{000}} \end{matrix}$$

Orthogonal matrix      Diagonal matrix      Orthogonal matrix

# Descompunerea valorilor singulare

$$X = U \cdot S \cdot V^T$$



# Exemplu de aproximare

$$\mathbf{X} = \begin{vmatrix} 2 & 8 & 4 \\ 7 & 9 & 2 \\ 4 & 1 & 3 \\ 2 & 9 & 6 \end{vmatrix}$$

$$\mathbf{U} = \begin{vmatrix} -0.50 & -0.33 & 0.11 \\ -0.61 & 0.65 & 0.43 \\ -0.20 & 0.42 & -0.87 \\ -0.59 & -0.53 & -0.24 \end{vmatrix}$$

$$\mathbf{S} = \begin{vmatrix} 18.12 & 0 & 0 \\ 0 & 5.07 & 0 \\ 0 & 0 & 3.31 \end{vmatrix}$$

$$\mathbf{V}^T = \begin{vmatrix} -0.40 & -0.82 & -0.40 \\ 0.89 & -0.24 & -0.39 \\ -0.22 & 0.51 & -0.83 \end{vmatrix}$$

$$\mathbf{S}' = \begin{vmatrix} 18.12 & 0 & 0 \\ 0 & 5.07 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

$$\mathbf{X}' = \mathbf{U} \cdot \mathbf{S}' \cdot \mathbf{V}^T = \begin{vmatrix} 2.13 & 7.83 & 4.28 \\ 7.35 & 8.27 & 3.14 \\ 3.34 & 2.46 & 0.62 \\ 1.88 & 9.41 & 5.32 \end{vmatrix} \approx \mathbf{X}$$

# Exemplu Python

Corpus:

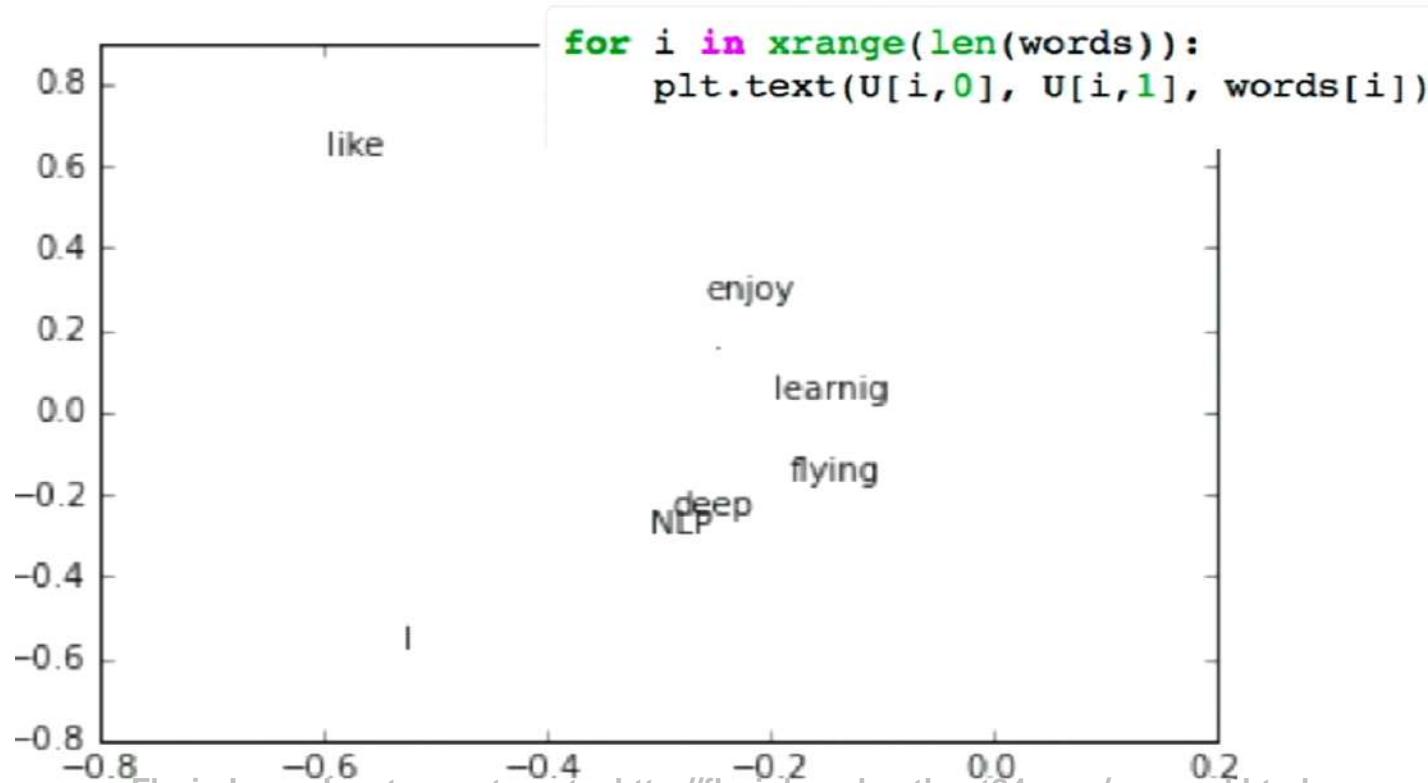
I like deep learning. I like NLP. I enjoy flying.

---

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep", "learnig", "NLP", "flying", ".."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])
U, s, Vh = la.svd(X, full_matrices=False)
```

# Vizualizare

- Se afișează cuvintele în funcție de primele două coloane ale matricei **U**, care corespund celor mai mari două valori singulare



# Aplicații complexe de învățare automată

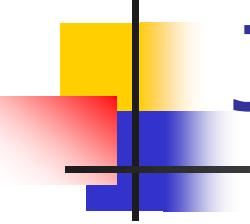
1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe***
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network*
  - 2.3. *AlphaGo*



# GloVe (Global Vector Representations, Stanford)

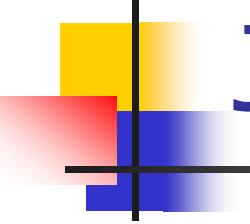
- GloVe încearcă să realizeze în mod explicit ceea ce word2vec face implicit: codarea informațiilor semantice sub formă de vectori de cuvinte
- Folosește fractii de probabilități de co-ocurență, nu direct probabilitățile de co-ocurență

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96



# Justificare

- Relația dintre două cuvinte poate fi examinată comparând relațiile acestora cu diferite „cuvinte sondă” (*probe words*)
- În tabelul anterior:
  - *solid* este înrudit cu *ice*, dar nu și cu *steam* ⇒ valoarea fracției este mare
  - *gas* este înrudit cu *steam*, dar nu și cu *ice* ⇒ valoarea fracției este mică
  - *water* este înrudit cu ambele ⇒ fracția este aproximativ 1
  - *fashion* nu este înrudit cu niciunul ⇒ fracția este tot aproximativ 1



# Justificare

- Un mod natural de a coda fracțiile este sub forma diferențelor folosind logaritmi
- De aici apar semnificațiile semantice ale diferențelor de vectori, ca în analogiile prezentare la început
- Obiectivul antrenării este ca produsul scalar al vectorilor de cuvinte să fie egal cu logaritmul probabilității de co-ocurență a cuvintelor

# Algoritmul GloVe

The GloVe algorithm consists of following steps:

1. Collect word co-occurrence statistics in a form of word co-occurrence matrix  $X$ . Each element  $X_{ij}$  of such matrix represents how often word  $i$  appears in context of word  $j$ . Usually we scan our corpus in the following manner: for each term we look for context terms within some area defined by a *window\_size* before the term and a *window\_size* after the term. Also we give less weight for more distant words, usually using this formula:

$$\text{decay} = 1/\text{offset}$$

2. Define soft constraints for each word pair:

$$w_i^T w_j + b_i + b_j = \log(X_{ij})$$

Here  $w_i$  - vector for the main word,  $w_j$  - vector for the context word,  $b_i, b_j$  are scalar biases for the main and context words.

3. Define a cost function

$$J = \sum_{i=1}^V \sum_{j=1}^V f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

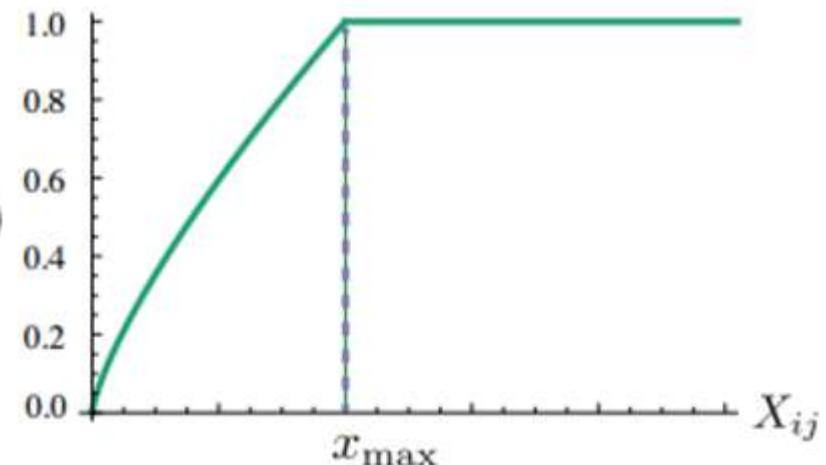
Here  $f$  is a weighting function which help us to prevent learning only from extremely common word pairs. The GloVe authors choose the following function:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^\alpha & \text{if } X_{ij} < X_{MAX} \\ 1 & \text{otherwise} \end{cases}$$

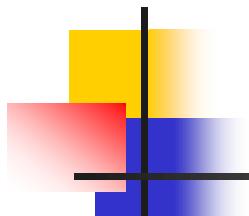
# Co-ocurențele rare

- Co-ocurențele rare dau mai puține informații decât cele frecvente
- Modelul include o funcție de ponderare

$$f(x) = \begin{cases} (x / x_{max})^\alpha & \text{dacă } x < x_{max} \\ 1 & \text{altfel} \end{cases}$$

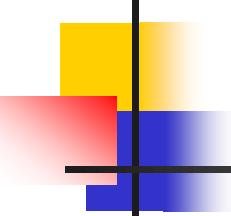


- Empiric, s-a descoperit că  $\alpha = 3/4$  dă rezultate puțin mai bune decât funcția liniară ( $\alpha = 1$ )
- Pragul  $x_{max}$  influențează și el performanțele. Autorii au ales  $x_{max} = 100$



# Metodologie

- S-au folosit, drept corpusuri, printre altele:
  - Wikipedia2014 cu 1,6 miliarde cuvinte
  - Gigaword5 + Wikipedia2014 cu 6 miliarde de cuvinte
- Acestea au fost reduse la cele mai frecvente 400.000 cuvinte
- S-a folosit un context de 10 cuvinte
- Pentru optimizare, s-a folosit algoritmul AdaGrad



# Discuție

- Își *word2vec* și *GloVe* determină vectori de cuvinte, însă *word2vec* este un model „predictiv”, iar *GloVe* este un model „bazat pe numărare”
- Modelele predictive încearcă să prezică cuvintele de context pe baza unui cuvânt țintă
- Modelele bazate pe numărare încearcă să reducă dimensiunea matricei de co-ocurență cu minimizarea erorii de reconstrucție
- Rezultatele *word2vec* și *GloVe* sunt destul de asemănătoare
- Metoda *GloVe* este mai ușor de paralelizat

# Aplicații complexe de învățare automată

## 1. Vectori de cuvinte (*word embeddings*)

1.1. Modelul *word2vec*

1.2. Descompunerea valorilor singulare

1.3. Modelul *GloVe*

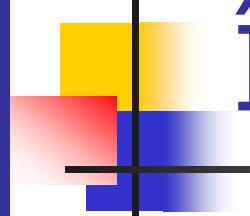
## 2. Învățare cu întărire profundă

2.1. *TD-Gammon*

2.2. *Deep Q-Network*

2.3. *AlphaGo*



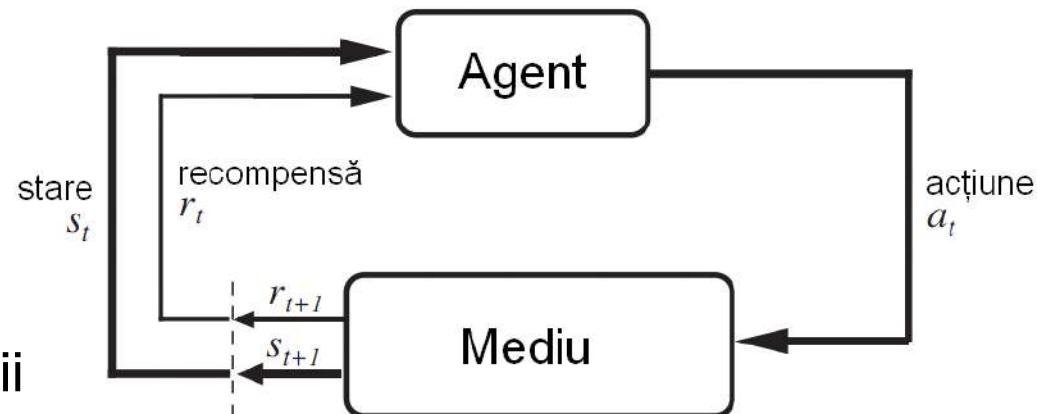


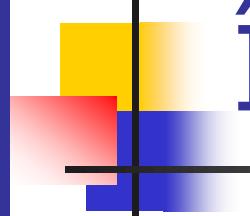
# Învățarea cu Întărire

- engl. “**reinforcement learning**”
- Agentul trebuie să învețe un comportament fără a avea un instructor
  - Agentul are o sarcină de îndeplinit
  - Efectuează niște acțiuni în mediul de execuție
  - Ulterior, primește un *feedback* (reacția mediului) privind cât de bine a acționat în vederea îndeplinirii sarcinii
  - Agentul urmărește îndeplinirea aceleiași sarcini în mod repetat
- Un **agent** este o entitate autonomă (software sau hardware) care acționează fără intervenție umană

# Modelul de interacțiune

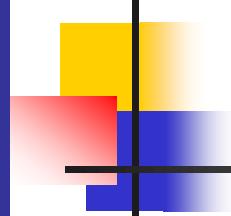
- Agentul
  - Efectuează **acțiuni**
- Mediul
  - Acordă **recompense**
  - Îi prezintă agentului situații numite **stări**
- Agentul primește o recompensă (întărire pozitivă) dacă îndeplinește bine sarcina
- Agentul primește o pedeapsă (întărire negativă) dacă îndeplinește rău sarcina





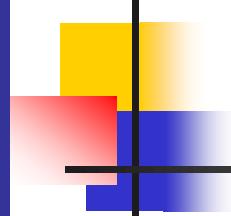
# Învățarea cu întărire

- Scopul este de a determina agentul să acționeze astfel încât să-și maximizeze recompensele totale
- Agentul trebuie să-și dea seama ce se va întâmpla dacă unele acțiuni conduce la îndeplinirea cât mai bună a sarcinii
- Acțiunile de obicei afectează și recompensele ulterioare, nu numai pe cele imediate: au un efect întârziat



# Decizii sevențiale

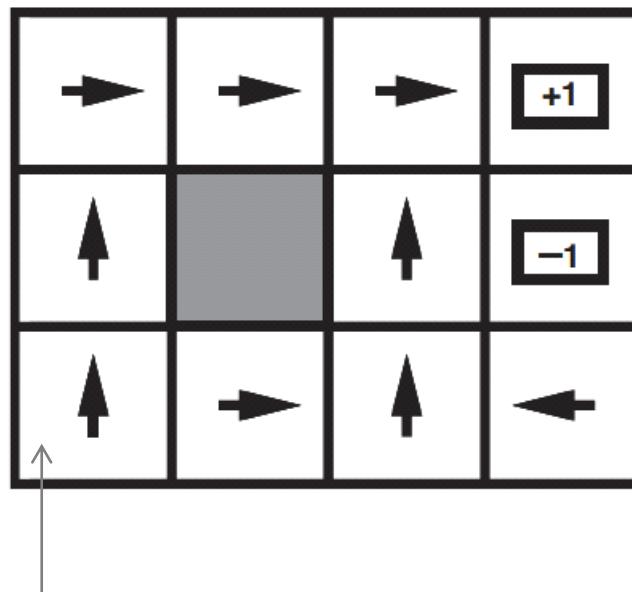
- Agentul pleacă dintr-o stare initială, face o secvență de acțiuni și ajunge într-o stare terminală
- Recompensele din stări nu sunt cunoscute
- Pentru jocuri, acțiunile sunt deterministe (nu luând în considerare aici elementele de șansă: zaruri, cărți), iar uneori recompensele se pot cunoaște doar pentru stările terminale (a câștigat sau a pierdut)
- Agentul execută o serie de încercări (*trials*), adică parcurge mediul în mod repetat din starea initială într-o stare terminală
- În definițiile din continuare, vom prezenta expresiile pentru acțiuni deterministe



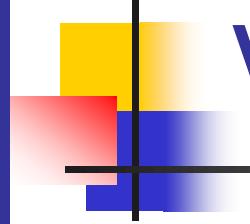
# Definiții

- Soluția unei astfel de probleme de căutare se numește **politică** (*policy*) și se notează  $\pi$
- $\pi(s)$  este acțiunea recomandată în starea  $s$ 
  - Politica este descrierea unui reflex
  - Politica este definită pentru toate stările: ce acțiune trebuie să facă agentul în orice stare s-ar afla
- Se numește **valoare** sau **utilitate** suma recompenselor pentru o secvență de stări
  - Recompensa este câștigul imediat, pe termen scurt
  - Valoarea este câștigul total, pe termen lung
- **Politica optimă**  $\pi^*$  maximizează valoarea

# Exemplu de politică



În această stare, agentul alege acțiunea „sus”



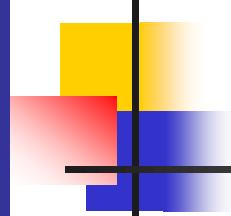
# Valorile stărilor

- Valoarea unei stări s atinsă în pasul  $t$  este valoarea secvenței de stări următoare. Secvența este determinată de  $\pi(s)$

$$V_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

$$V_t = R_{t+1} + \gamma V_{t+1}$$

- $\gamma$  este **factorul de actualizare** (*discount factor*) care arată cât de importante sunt recompensele viitoare în raport cu cele imediate

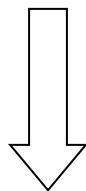


# Politica optimă

- Politica optimă alege acțiunea care conduce în starea cu cea mai mare valoare
- Valoarea unei stări este recompensa imediată pentru acea stare plus valoarea maximă actualizată a stării următoare
- Agentul trebuie să învețe politica optimă

# Ecuăția Bellman

$$V(s_t) = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$



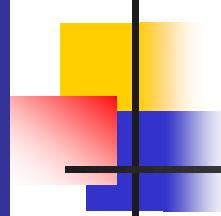
$\delta(s, a)$  returnează starea următoare rezultată prin aplicarea acțiunii  $a$  în starea  $s$



$$V(s_t) = R_{t+1} + \gamma \max_{a \in A(s_t)} V(\delta(s_t, a))$$



Ecuăția Bellman

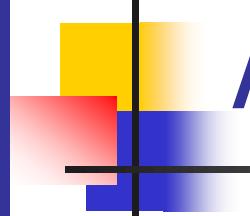


# Diferențele temporale

- engl. “**temporal differences**”
- Metoda diferențelor temporale combină **observațiile directe** cu constrângerile **ecuațiilor Bellman**, care caracterizează modelul mediului:

$$V_t \leftarrow V_t + \alpha (R_{t+1} + \gamma V_{t+1} - V_t)$$

- Rata de învățare  $\alpha$  determină viteza de convergență la utilitatea reală



# Algoritmul Q-Learning

Initialize  $Q(s, a)$ ,  $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

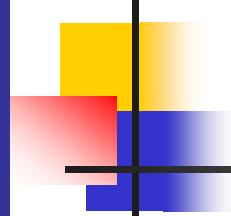
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

    until  $S$  is terminal



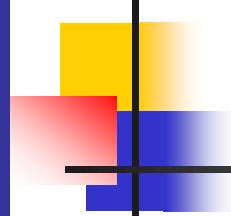
# Explicații

- Valoarea lui  $Q$  (de la *quality*) este suma dintre recompensa imediată și valoarea actualizată corespunzătoare execuției politicii optime din starea următoare în continuare

$$Q(s, a) = R(s, a) + \gamma V^*(\delta(s, a))$$

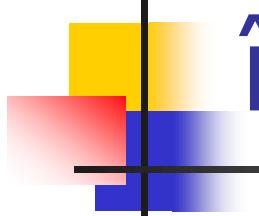
$$V^*(s) = \max_a Q(s, a)$$

$$\pi(s) = \operatorname{argmax}_a Q(s, a)$$



# Explorare vs. exploatare

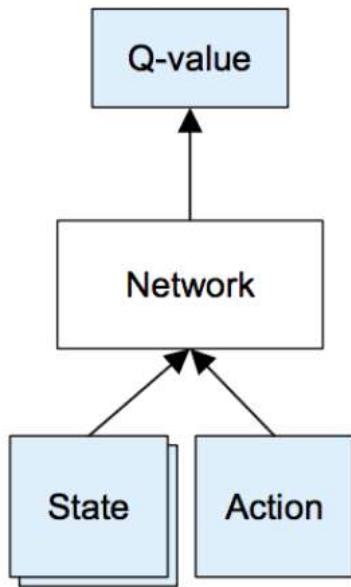
- Strategia  $\varepsilon$ -greedy
  - Explorare: cu probabilitatea  $\varepsilon$ , se selectează o acțiune aleatorie
  - Exploatare: altfel, se selectează în mod greedy acțiunea cea mai bună conform politicii (care are valoarea Q maximă)



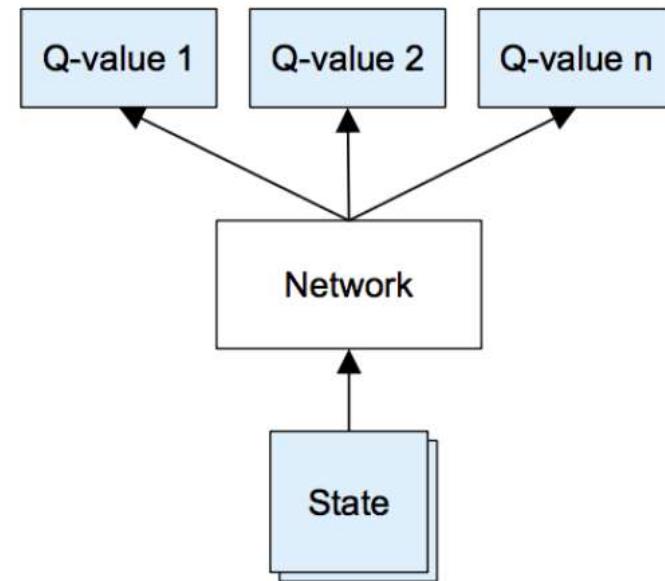
# Rețele neuronale pentru învățare cu întărire

- De obicei:
  - Rețeaua este folosită pentru a aproxima valorile Q
  - Funcția aproximată de rețea este numită  $v$  (de la „valoare”)

# Aproximarea funcției Q



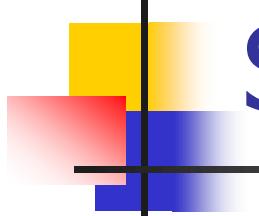
Abordarea directă



**Abordarea optimizată:** ieșirile sunt valorile acțiunilor posibile în starea curentă (toate valorile Q pentru o stare se calculează într-un singur pas)

# Aproximarea funcției Q: algoritmul general

- Se calculează  $Q_t(s_t, a_i)$  folosind propagarea înainte prin rețea neuronală pentru toate acțiunile  $a_i$  posibile în starea  $s_t$
- Se alege o acțiune nouă  $a_t$  și se trece într-o nouă stare  $s_{t+1}$
- Se calculează  $Q_t(s_{t+1}, a_i)$  folosind propagarea înainte prin rețea neuronală pentru toate acțiunile din starea  $s_{t+1}$
- Se setează valoarea  $Q_{t\text{intă}}$  ca fiind:  $r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a)$  doar pentru acțiunea curentă  $a_t$ , respectiv  $Q_{t+1}(s_t, a_i) = Q_t(s_t, a_i)$  pentru celelalte acțiuni
- Se calculează vectorul de eroare  $e = Q_{t\text{intă}} - Q_t = Q_{t+1} - Q_t$
- Se aplică algoritmul *backpropagation* asupra rețelei pentru actualizarea ponderilor

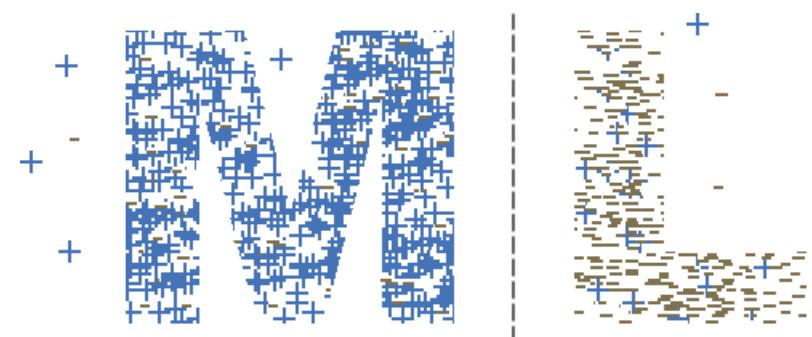


# Studii de caz

- TD-Gammon
  - Tesauro, 1992-2002
- Deep Q-Network
  - Mnih, Google DeepMind, 2015
- AlphaGo
  - Silver et. al, Google DeepMind, 2017

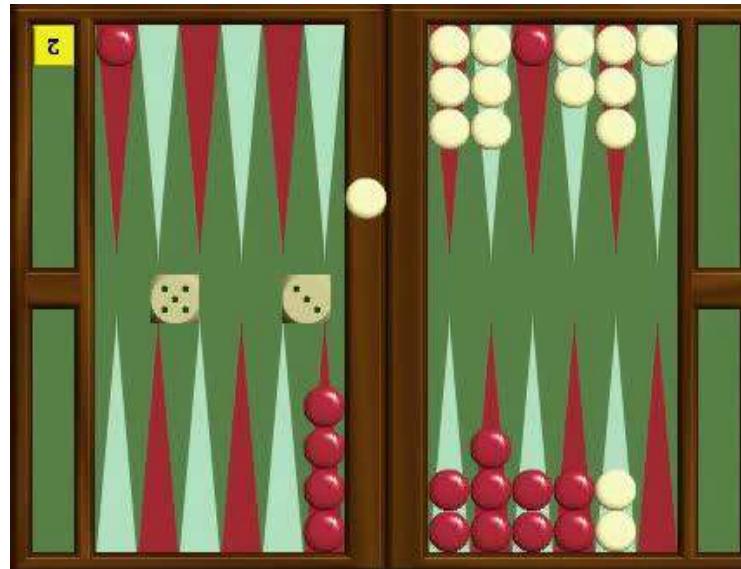
# Aplicații complexe de învățare automată

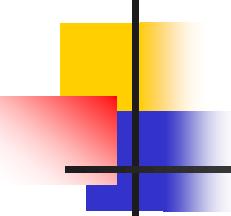
1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe*
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network*
  - 2.3. *AlphaGo*



# TD-Gammon

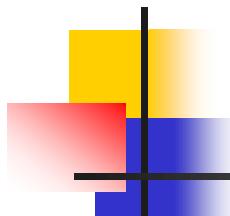
- Program de table (*backgammon*)
- Inițial 15 piese pentru fiecare jucător, 24 locații (+2: bara și în afara tablei), 2 zaruri  $\Rightarrow$  factorul de ramificare  $\approx 400$





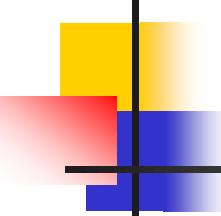
# Modelul TD-Gammon 0.0

- Valoarea estimată a unei stări s este probabilitatea de a câștiga din acea stare
- Valorile sunt estimate de o rețea neuronală
- Intrările rețelei reprezintă configurația tablei la un moment dat
- Pentru fiecare locație sunt 4 intrări ( $I_1 I_2 I_3 I_4$ ) care codează numărul de piese albe
  - Nicio piesă albă  $\Rightarrow I_1 I_2 I_3 I_4 = 0000$
  - O piesă albă (poate fi „mâncată” de adversar)  $\Rightarrow I_1 = 1$
  - Două sau mai multe (nu pot fi „mâncate”)  $\Rightarrow I_2 = 1$
  - Trei piese (o singură rezervă)  $\Rightarrow I_3 = 1$
  - $n$  piese,  $n > 3 \Rightarrow I_4 = (n - 3) / 2$
- Pentru fiecare locație sunt alte 4 intrări care codează numărul de piese negre



# Modelul TD-Gammon 0.0

- Pentru 24 de locații sunt  $24 \cdot 4 \cdot 2 = 192$  intrări
- Alte 6 intrări:
  - O intrare cu valoarea  $n / 2$ , unde  $n$  este numărul de piese albe de pe bară
  - O intrare similară pentru piesele negre
  - O intrare cu valoarea  $n / 15$ , unde  $n$  este numărul de piese albe scoase de pe tablă
  - O intrare similară pentru piesele negre
  - O intrare = 1 dacă e rândul albului să mute
  - O intrare = 1 dacă e rândul negrului să mute
- În total, rețeaua are 198 de intrări



# Urme de eligibilitate

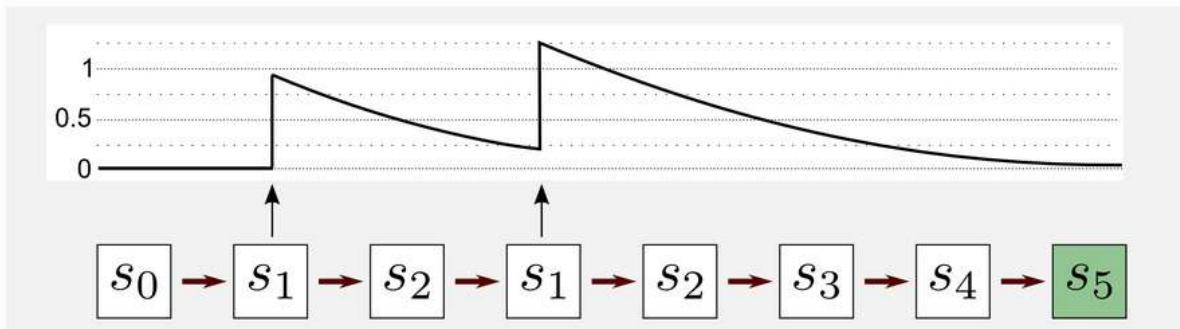
- engl. “eligibility traces”
- Algoritmul Q-Learning prezentat nu ia în calcul stările trecute
- Din starea  $s_{t+1}$  se actualizează numai starea anterioară  $s_t$
- Învățarea poate fi accelerată actualizând și stările din trecut
- Pentru aceasta este necesar un mecanism de **memorie pe termen scurt**, care să arate ce stări au fost vizitate în ultimii pași
- Pentru fiecare stare  $s$  în pasul  $t$  se definește **urma de eligibilitate**  $e_t(s)$ :

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t; \end{cases}$$

- unde  $\gamma$  este factorul de actualizare, iar  $\lambda \in [0, 1]$  se numește *trace-decay* sau *accumulating trace* și definește scăderea în timp a urmei de eligibilitate

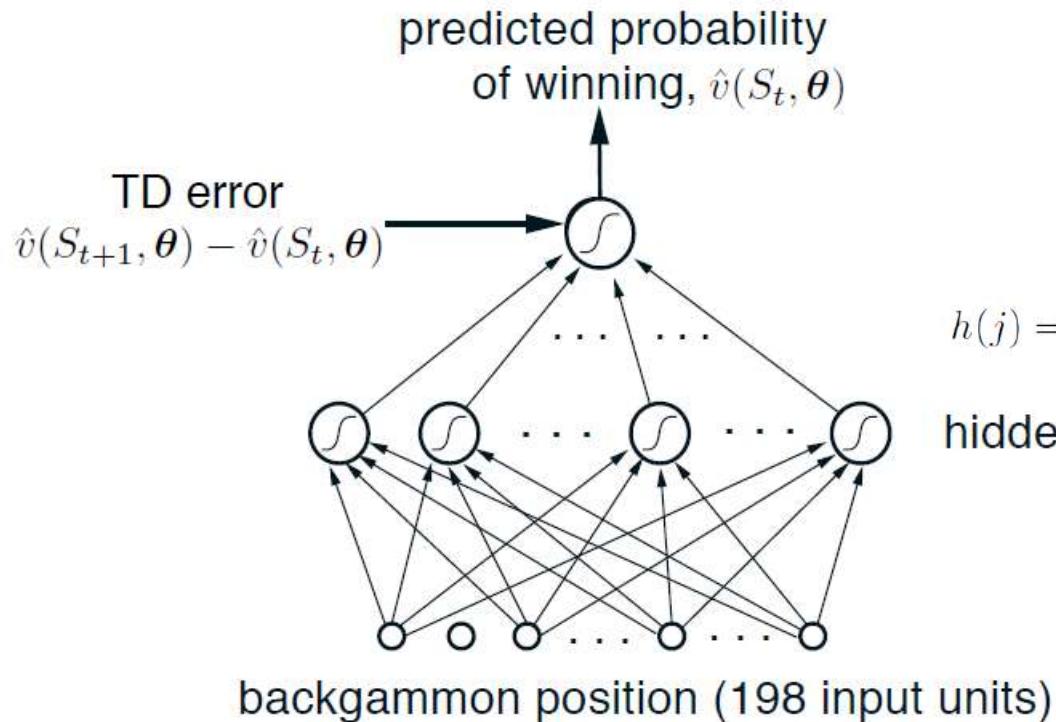
# Exemplu

- 7 pași de vizitare în care sunt vizitate 5 stări



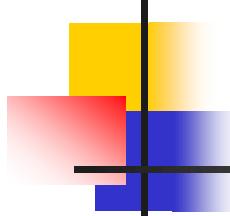
- Pentru probleme în care recompensele sunt rare (de exemplu, doar la sfârșitul unui joc), este nevoie de multă explorare pentru a propaga valorile diferite de zero în restul spațiului stărilor
- Urmele de eligibilitate permit actualizarea mai multor stări, iar stările mai recente sunt actualizate mai mult

# Rețeaua neuronală



$$h(j) = \sigma \left( \sum_i w_{ij} x_i \right) = \frac{1}{1 + e^{-\sum_i w_{ij} x_i}}$$

hidden units (40-80)



# Antrenarea

- Gradientii sunt calculati prin *backpropagation*:

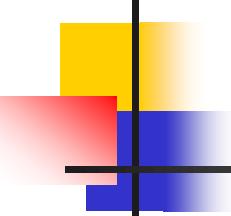
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left[ R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{\theta}_t) - \hat{v}(S_t, \boldsymbol{\theta}_t) \right] \mathbf{e}_t$$

- Parametrii  $\boldsymbol{\theta}$  sunt ponderile rețelei
- Pentru fiecare  $\theta_t$  există un  $e_t$  (*eligibility trace*):

$$\mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \nabla \hat{v}(S_t, \boldsymbol{\theta}_t)$$

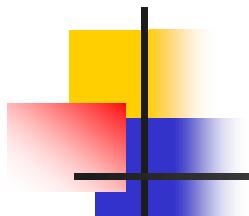
- În cazul nostru:  $\gamma = 1$ , recompensele sunt întotdeauna 0, cu excepția câștigului, iar eroarea TD este:

$$\hat{v}(S_{t+1}, \boldsymbol{\theta}) - \hat{v}(S_t, \boldsymbol{\theta})$$



# Metodologia de antrenare

- TD-Gammon juca cu el însuși
  - Versiunile 0.0, 1.0: 300.000 de jocuri
  - Versiunea 2.0: 800.000 de jocuri
  - Versiunile 2.1, 3.0: 1.500.000 de jocuri
- Fiecare joc este un episod de învățare
- În fiecare rundă, rețeaua calculează valorile tuturor mutărilor posibile și se alege mutarea cea mai promițătoare
- La început, ponderile fiind aleatorii, mutările nu erau bune, iar jocurile puteau dura sute sau mii de mutări
- După câteva zeci de jocuri, mutările s-au îmbunătățit
- TD-Gammon 0.0 era la fel de bun ca programele de table existente, deși nu folosea informații din jocuri umane

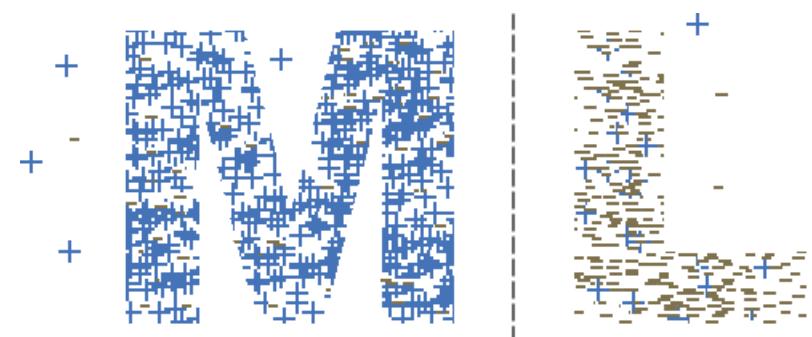


# Metodologia de antrenare

- Versiunile ulterioare au adăugat căutarea pe 2 și 3 niveluri, ca în algoritmul de căutare minimax
- TD-Gammon 3.0 era la fel de bun ca jucătorii profesioniști umani
- A descoperit niște strategii de joc noi, care sunt chiar folosite în prezent de jucătorii umani

# Aplicații complexe de învățare automată

1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe*
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network***
  - 2.3. *AlphaGo*

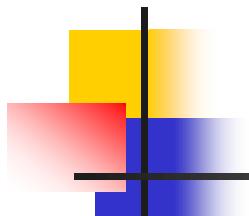


# Jocuri Atari 2600

- 49 de jocuri: *Pong*, *Breakout*, *Space Invaders*, *Seaquest*, *Beam Rider* etc.

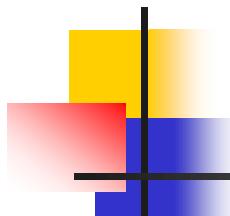


- Deep Q-Network (DQN) s-a dorit un pas către inteligență artificială generală, deoarece folosește același algoritm pentru jocuri destul de diferite



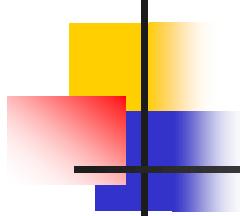
# Preprocesarea intrărilor

- Un jucător uman vede imagini de 210x160 cu 128 de culori RGB
- Pentru *DQN*, fiecare cadru este preprocesat pentru a produce o matrice 84x84 de valori de luminanță
  - $L = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$



# Modelul jocurilor

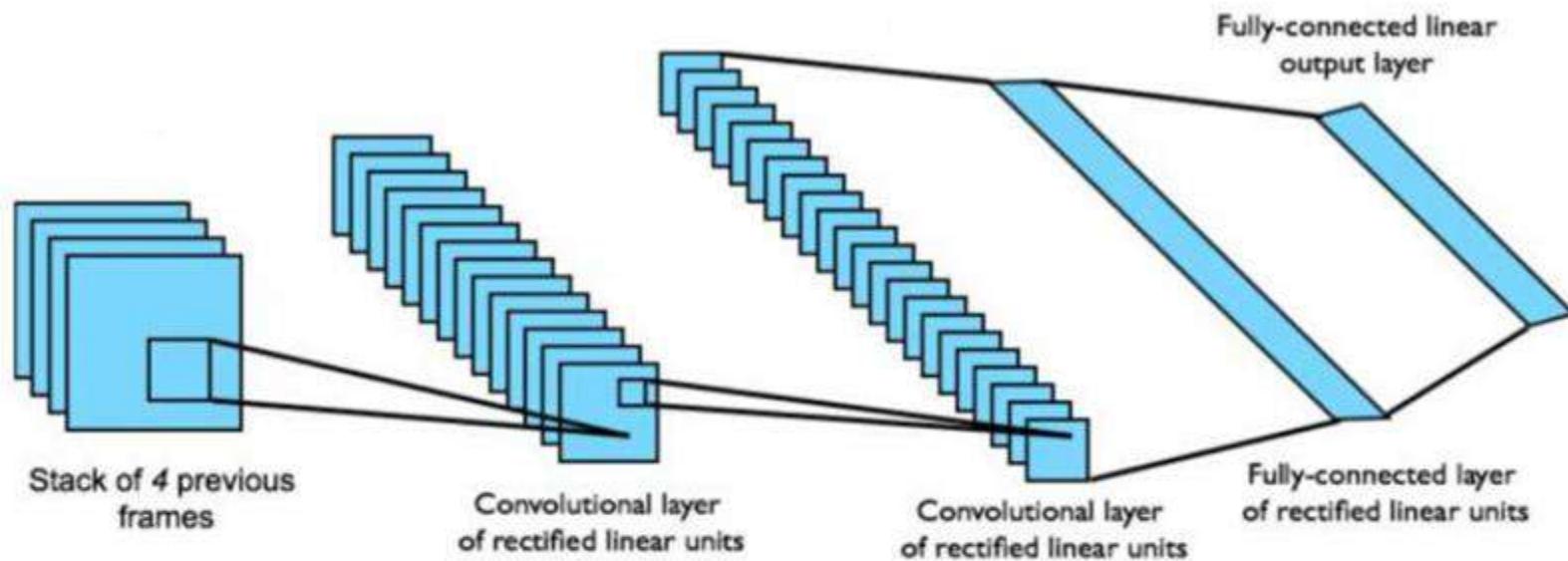
- Intrări: 4 imagini consecutive ale jocului, pentru a putea estima viteza și direcția personajelor sau elementelor din joc
- Ieșiri: valorile Q ale tuturor acțiunilor posibile
- Numărul de acțiuni depinde de joc și poate fi între 2 și 18 (corespunzător comenziilor unui *joystick*), de exemplu: sus, jos, dreapta, stânga, foc, accelerează, frânează, ia cheia, deschide ușa etc.
- Rețeaua interacționează cu un emulator pentru jocuri Atari
- Aproximarea valorilor se face cu rețele de convoluție (prezentate în cursul 8)
- Reacția mediului este dată doar de imagini și scor



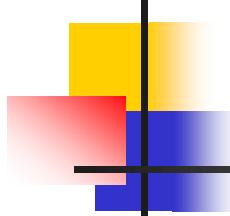
# Recompensele

- Scorul este standardizat, pentru a fi tratat în același fel pentru toate jocurile
- Recompensa este:
  - +1 dacă scorul crește
  - -1 dacă scorul scade
  - 0 dacă scorul rămâne neschimbat
- Astfel, se poate folosi și aceeași rată de învățare pentru toate jocurile

# Arhitectura DQN



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18



# Functia de cost

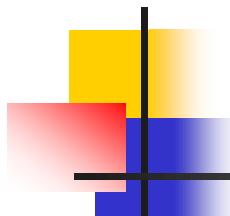
- Învățare supervizată

$$l(\theta) = (y - f(x; \theta))^2$$

- *Deep Q-Learning*

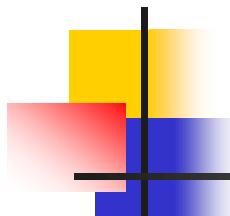
$$l(\theta) = \frac{\left( r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta) \right)^2}{\text{ținta} \qquad \qquad \qquad \text{estimarea}}$$

Și ținta și estimarea se bazează pe  
aceeași funcție  $Q$  aproximată de rețea!



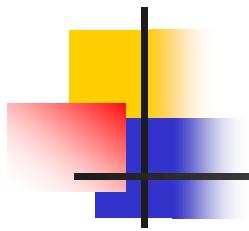
# Instabilitatea antrenării

- Probleme:
  - Valorile ţintă nu sunt fixe
  - Experienţele succesive sunt corelate şi depind de politică
  - Mici modificări ale parametrilor provoacă modificări mari ale politicii, care conduc la schimbări mari în distribuţia datelor
- Soluţii:
  - Folosirea metodei *experience replay*
  - Înghetarea reţelei Q ţintă
  - Limitarea termenului de eroare



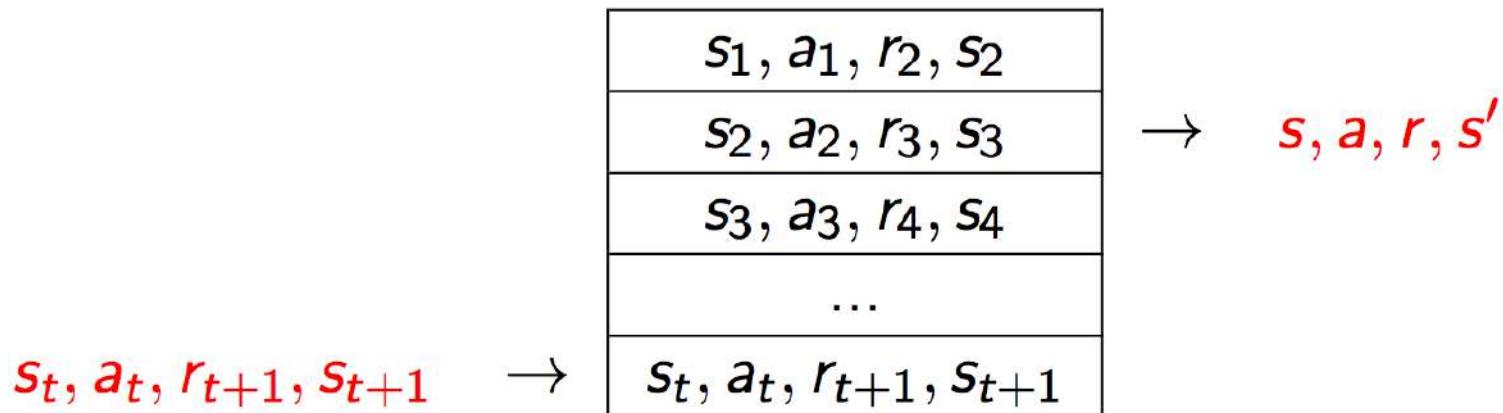
# Experience replay

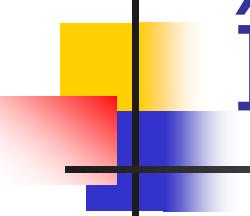
- În timpul jocului, toate tranzițiile  $\langle s, a, r, s' \rangle$  sunt memorate într-o structură numită *replay memory*
- Când se antrenează rețeaua, sunt folosite mini-loturi aleatorii din *replay memory* în loc de tranziția cea mai recentă
- Această metodă evită problema similarității eșantioanelor de antrenare succesive, care ar conduce rețeaua într-un optim local
- Este posibilă și colectarea tranzițiilor din jocul unui om și antrenarea rețelei cu acestea



# Experience replay

- Se alege o acțiune în mod  $\varepsilon$ -greedy
- Se adaugă tranzitia  $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$  în *replay memory*
- Se trece în starea  $s_{t+1}$  și se continuă jocul, dar actualizarea ponderilor rețelei se face cu un număr mic de tranzitii din *replay memory*



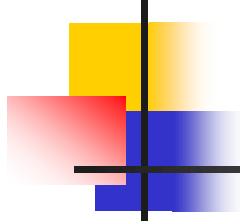


# Înghetarea rețelei țintă

- Parametrii curenti ai rețelei determină următoarele date de antrenare, ceea ce poate conduce la fenomenul de reacție pozitivă și deci instabilitate
- După un număr anumit de actualizări, rețeaua curentă este copiată în altă rețea, iar ponderile acestei rețele sunt menținute fixe și constituie țintele de antrenare

$$\theta_{t+1} = \theta_t + \alpha \cdot \left( r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a; \theta_t^-) - Q(s_t, a_t; \theta_t) \right) \frac{dQ(s_t, a_t; \theta_t)}{d\theta_t}$$

rețeaua țintă  
„înghetată”



# Limitarea erorii

- Termenul de eroare

$$e = r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a; \theta_t^-) - Q(s_t, a_t; \theta_t)$$

este limitat la intervalul [-1, 1]

---

**Algorithm** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

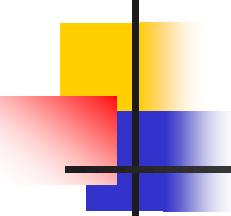
        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$

**end for**

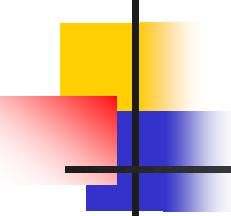
**end for**

---



# Explicații

- Fiecare episod este un joc complet
- $t$  reprezintă fiecare pas din joc
- $\phi$  reprezintă imaginile  $x$  procesate
- Se folosește strategia  $\varepsilon$ -greedy: cu probabilitatea  $\varepsilon$ , se selectează o acțiune aleatorie, altfel se selectează în mod greedy acțiunea cea mai bună conform politicii (cu valoarea Q maximă)
  - $\varepsilon$  scade de la 1 la 0,1 pe parcursul primului milion de cadre, apoi rămâne constant 0,1
- Optimizarea s-a făcut cu algoritmul RMSProp
  - Dar se poate aplica orice alt algoritm, de exemplu Adam etc.



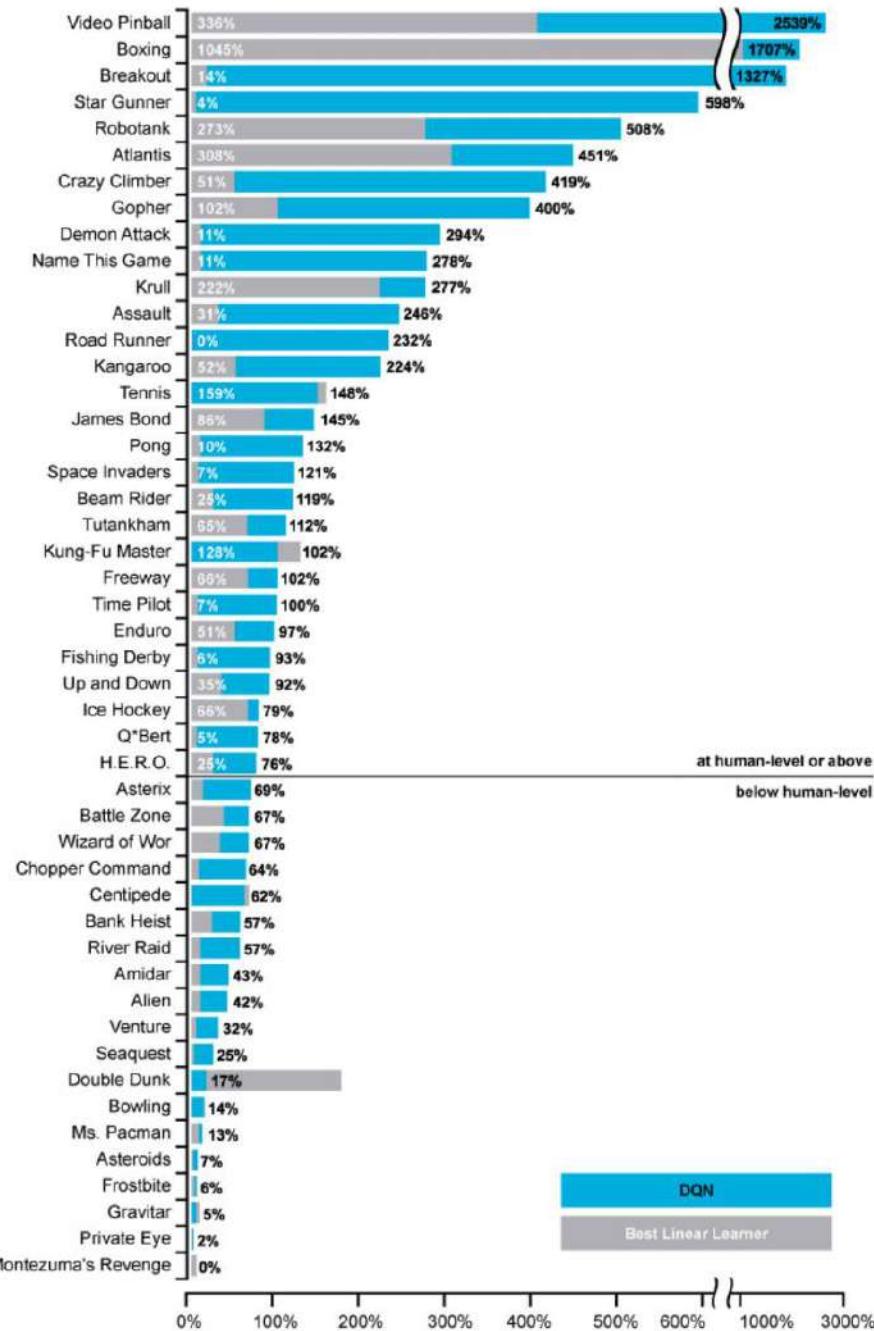
# Metodologia de antrenare

- DQN a învățat fiecare joc interacționând cu emulatorul de joc pe parcursul a 50 de milioane de cadre, echivalent cu 38 de zile de joc
- Pentru evaluarea performanțelor după învățare, pentru fiecare joc Atari s-a făcut media a 30 de jocuri de câte 5 minute, care începeau cu o stare aleatorie

# Rezultate

Axa x arată scorul obținut de modelul DQN ca procente în comparație cu jucătorii umani

La jocul *Montezuma's Revenge*, unde DQN obține 0%, eroul poate să moară foarte repede, iar rețeaua nu reușește să învețe



# Exemplu: *Breakout*



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

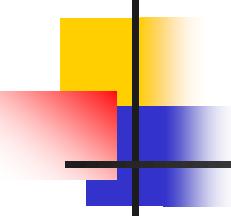
# Aplicații complexe de învățare automată

1. Vectori de cuvinte (*word embeddings*)
  - 1.1. Modelul *word2vec*
  - 1.2. Descompunerea valorilor singulare
  - 1.3. Modelul *GloVe*
2. Învățare cu întărire profundă
  - 2.1. *TD-Gammon*
  - 2.2. *Deep Q-Network*
  - 2.3. *AlphaGo*



# Go

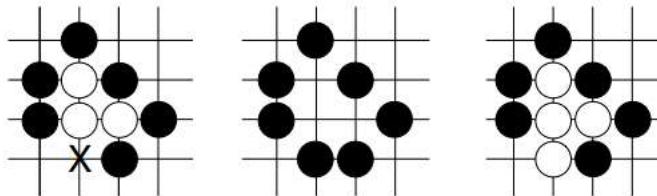




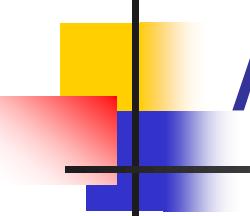
# Reguli

- Dimensiunea tablei este  $19 \times 19$
- Jucătorii pun piese pe tablă alternativ, iar piesele puse nu se pot mișca
- Jucătorul care pune al doilea primește niște puncte de compensație
- Piesele unui jucător complet înconjurate de piesele adversarului sunt eliminate din joc
- Jocul se termină când nu mai există mutări „profitabile”
- Nu se acceptă mutări care să repete poziții anterioare
- Câștigă jucătorul care are în final „teritoriul” cel mai mare

# Exemplu

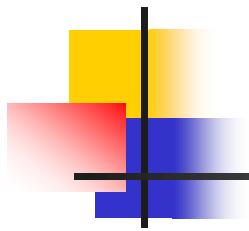


Go capturing rule. Left: the three white stones are not surrounded because point X is unoccupied. Middle: if black places a stone on X, the three white stones are captured and removed from the board. Right: if white places a stone on point X first, the capture is blocked.



# AlphaGo Zero

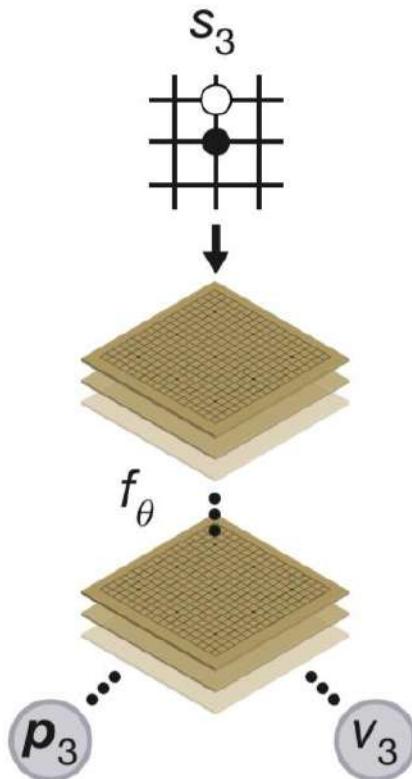
- Combină modelele neuronale cu *Monte Carlo Tree Search* (MCTS), o metodă stochastică de căutare a soluțiilor în jocuri cu factori mari de ramificare
- Folosește o rețea cu două capete, care aproximează politicile (probabilitățile de selecție ale acțiunilor), respectiv valorile
- Nu se bazează pe trăsături ale jocului identificate manual sau prin învățare din jocuri umane
- Se folosesc doar regulile jocului, de aici partea de Zero a numelui
- Antrenarea se face exclusiv prin *self-play*, nu învață din jocuri ale jucătorilor profesioniști umani
- A descoperit variante noi ale strategiilor clasice de joc



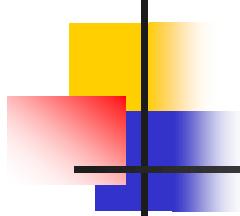
# Modelul jocurilor

- Intrările rețelei: o stivă de imagini  $19 \times 19 \times 17$
- Pentru fiecare poziție de pe tabla de joc există 17 trăsături binare:
  - Primele 8 indică dacă poziția este ocupată de AlphaGo în starea curentă, respectiv 7 stări anterioare: 1 dacă da, 0 dacă nu
  - Următoarele 8 indică același lucru pentru adversar
  - Ultima indică mutarea curentă: 1 pentru negru, 0 pentru alb

# Modul de antrenare



- Pentru o configurație de joc dată, rețeaua neuronală calculează atât probabilitățile mutărilor  $P$ , cât și probabilitatea de a câștiga  $V$
- Se rulează MCTS pentru a rafina probabilitățile de mutare  $P'$  și câștig  $V'$
- Se actualizează parametrii rețelei pentru a apropi  $P$  și  $V$  de  $P'$  și  $V'$
- Procesul seamănă cu algoritmul de **iterare a politicilor**: *self-play* cu MCTS reprezintă evaluarea politicilor, iar actualizarea rețelei reprezintă îmbunătățirea politicilor
- Pentru MCTS, **capătul de politică** ajută la scăderea lățimii de căutare dintr-un nod (preferând acțiunile mai promițătoare), iar **capătul de valoare** ajută la scăderea adâncimii de căutare în nodurile frunză

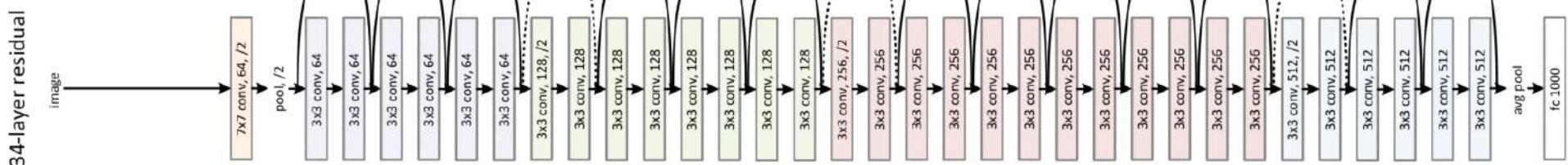


# Modul de antrenare

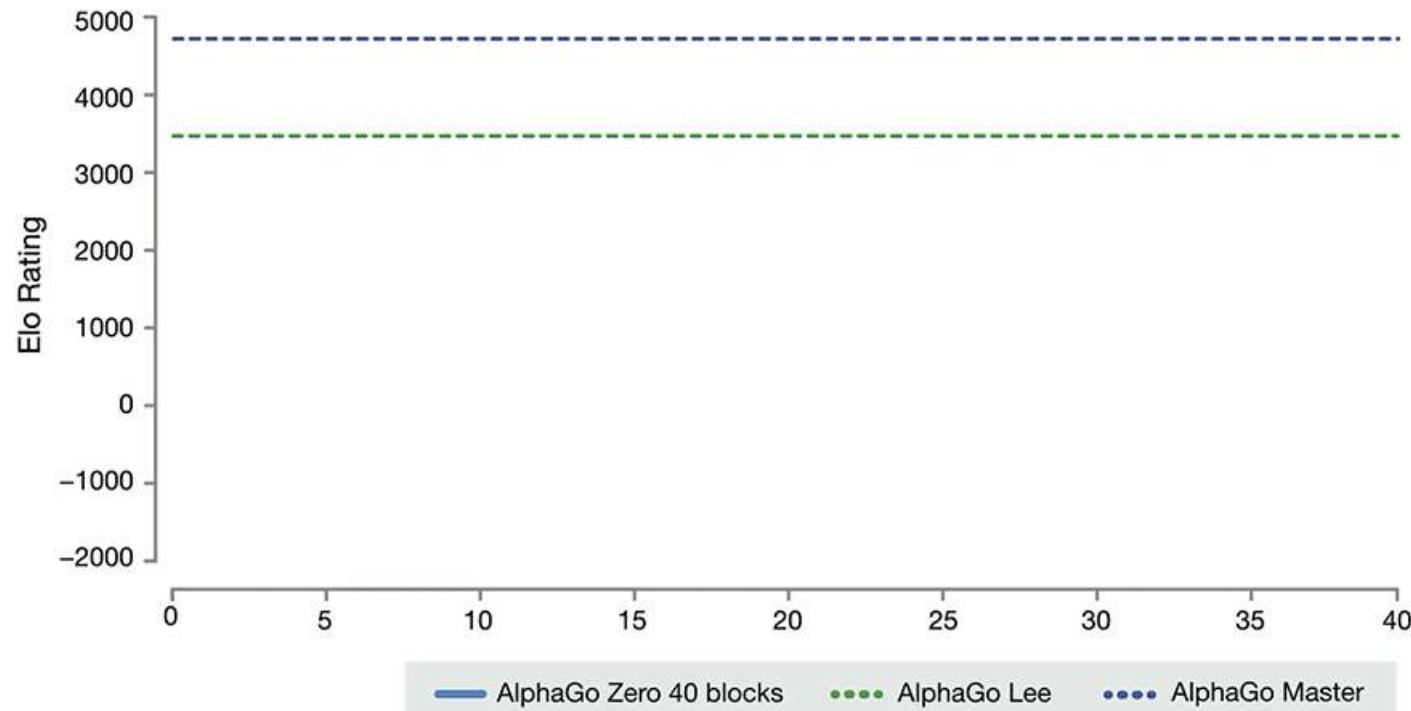
- Se generează multe rețele cu parametri aleatorii
- Învățarea este împărțită în perioade
- Pentru fiecare perioadă, se alege o rețea „evaluator”
- Evaluatatorul joacă 25.000 de jocuri *self-play*, cu 1.600 de simulări MCTS pentru fiecare mutare
- Celealte rețele învață din jocurile evaluatorului
- Fiecare rețea joacă 250 de jocuri împotriva evaluatorului
- Rețeaua cu cele mai multe victorii înlocuiește evaluatorul

# Arhitectura rețelelor

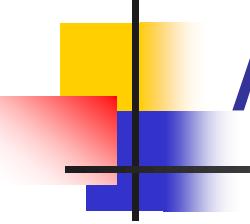
- AlphaGo Zero folosește **rețele reziduale** (Microsoft Research, 2015) în loc de rețele de convoluție



# Cum învață AlphaGo Zero

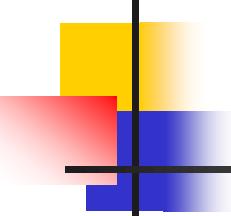


<https://deepmind.com/blog/alphago-zero-learning-scratch/>



# AlphaZero

- AlphaZero este un program asemănător cu AlphaGo Zero, dar care poate învăța și alte jocuri, de exemplu șah sau shogi (șah japonez)
- După numai 4 ore de antrenare pornind doar de la regulile jocului, AlphaZero a reușit să învingă cel mai bun program de șah existent în 2017, Stockfish
- *Google's self-learning AI AlphaZero masters chess in 4 hours*
  - <https://www.youtube.com/watch?v=0g9SlVdv1PY>



# Concluzii

- Tehnicile de *word embeddings* au condus la mari progrese în aplicațiile de prelucrare a limbajului natural, deoarece încorporează relații semantice în reprezentare
- Învățarea cu întărire este o metodă utilă în situațiile în care un sistem este dificil de proiectat explicit din cauza complexității sale, precum jocurile. Rețelele neuronale pot comprima spațiul stărilor și pot aproxima funcția de valoare

# Învățare automată

## 8. Rețele neuronale conveționale I

**Marius Gavrilăescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

- “Convolutional Neural Network” – abreviat CNN
- sunt rețele neuronale asociate conceptului “deep learning” (învățare profundă)
- concepute inițial pentru clasificarea imaginilor

- Deep learning
  - rețele neuronale cu straturi multiple (>5)
    - tratează probleme de complexitate mare
    - date de intrare de mari dimensiuni
  - se utilizează pentru selecția trăsăturilor și implementarea de diverse transformări
  - algoritmii aferenți pot fi
    - supervizați (clasificare)
    - nesupervizați (diverse metode de analiză a datelor – pattern analysis etc.)

## ■ CNN

- similare în multe privințe cu rețelele neuronale clasice
- straturi de neuroni cu intrări multiple
  - fiecărei intrări i se asociază o pondere
  - se realizează produsul scalar dintre vectorul intrărilor și al cel ponderilor
  - urmează aplicarea unei funcții neliniare (“funcție de activare”)
  - scopul poate fi
    - clasificare: încadrarea datelor de intrare într-o clasă (dintr-o mulțime finită de clase)
    - regresie: determinarea relației dintre datele de intrare și una sau mai multe variabile dependente
    - transformare: învățarea unor operații de modificare a datelor / generare a unor noi date de același tip

## ■ CNN

- cel mai frecvent, datele de intrare sunt imagini
- cel mai adesea se utilizează pentru probleme de clasificare
  - se dorește încadrarea conținutului imaginii într-o anumită categorie
- exemple:
  - identificarea unei anumite persoane într-o serie de poze
  - identificarea unui anumit obiect, dintr-o mulțime de obiecte cunoscute
  - identificarea expresiilor / trăsăturilor faciale etc.

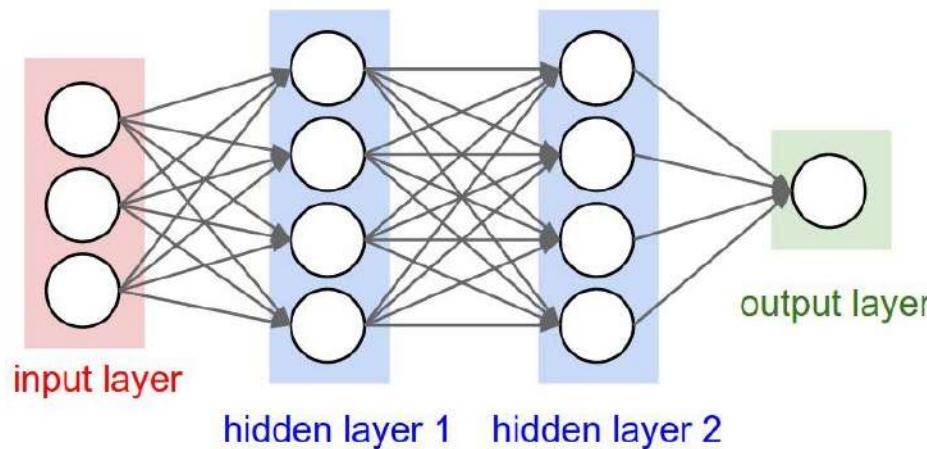
- spre deosebire de rețelele neuronale clasice
  - straturile sunt mai numeroase și de mai multe tipuri (straturi de conoluție, agregare, fully-connected etc.)
  - au o topologie clară și inflexibilă
    - sunt proiectate pentru anumite tipuri de imagini, pentru identificarea anumitor trăsături etc.
    - sunt proiectate pentru rezolvarea unei anumite probleme – rețele proiectate pentru detecția unor anumite categorii de obiecte și care nu sunt utilizabile în alte situații

- spre deosebire de rețelele neuronale clasice

- neuronii din unele straturi nu sunt complet conectați
- în unele straturi, ponderile sunt partajate
- metodele de antrenare diferă în anumite aspecte

## ■ Necesitatea CNN

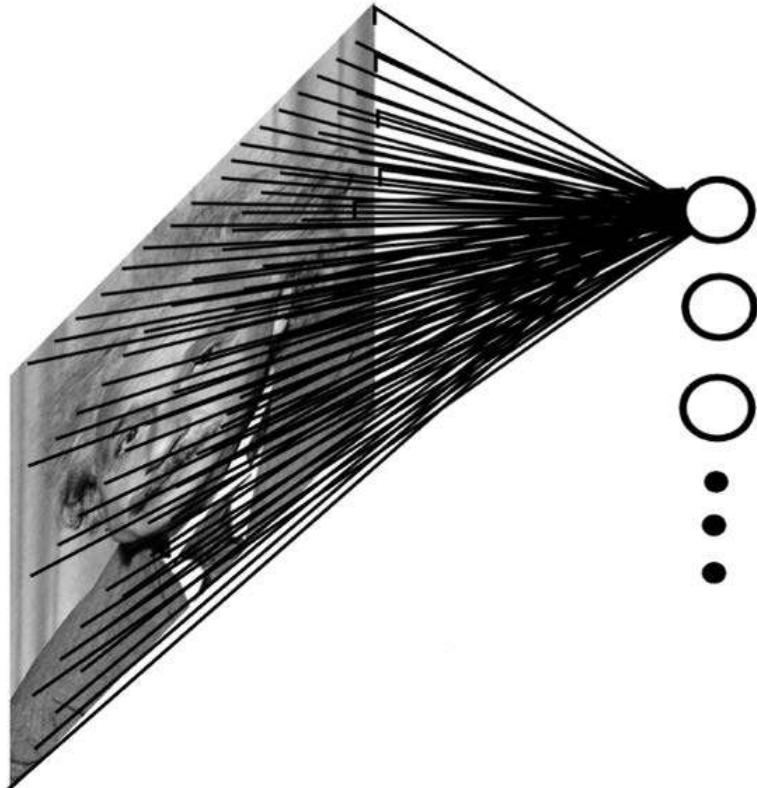
- În cazul unei rețele clasice, fiecare neuron din stratul de la intrare este conectat la fiecare neuron din primul strat ascuns



- Necesitatea CNN
  - prelucrarea imaginilor presupune unui număr mare de parametri
    - număr mare de intrări (inputs)
    - număr imens de conexiuni între neuroni
  - ex: stratul de intrare (input layer) al unei CNN are  $w \times h$  neuroni, unde  $w$  și  $h$  sunt dimensiunile imaginii
  - presupunem că primul strat ascuns ar avea  $w/2 \times h/2$  neuroni

- Necesitatea CNN

- pentru o imagine de 256 x 256 pixeli
    - stratul de intrare ar avea 65536 neuroni
    - primul strat ascuns ar avea 16384 neuroni
    - ... rezultă 1073741824 conexiuni doar între primele două straturi
    - cel puțin 1073741824 ponderi de actualizat în fiecare fază de antrenare

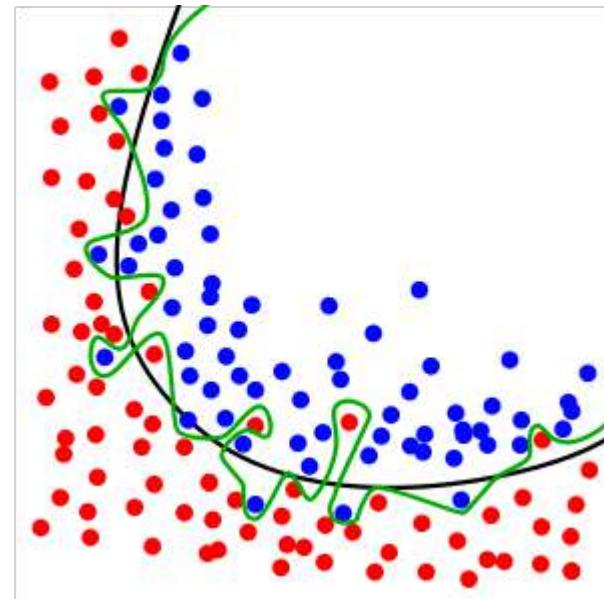


- Fiecare neuron are un număr imens de conexiuni
- E posibil să fie nevoie de câteva zeci de mii de neuroni în straturile ascunse
- Rezultă un număr de parametri dificil de gestionat

- rezultă necesitatea unor resurse de calcul semnificative
- e posibil să nu dispunem de suficiente date de antrenare cât să acoperim spațiul de soluții
- un număr mare de parametri poate conduce la *overfitting*
  - situație în care rezultatul clasificării este afectat de zgomot sau de mici variațiuni ale datelor

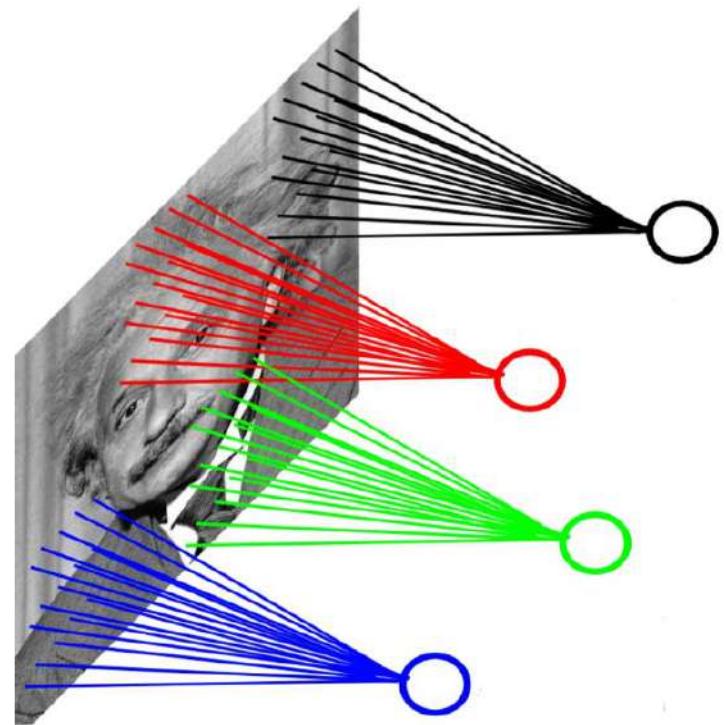
- metoda afectată de overfitting (linia verde) descrie cel mai bine datele, dar

- depinde în prea mare măsură de un anumit set de date
- este susceptibilă la erori semnificative în cazul unor date noi



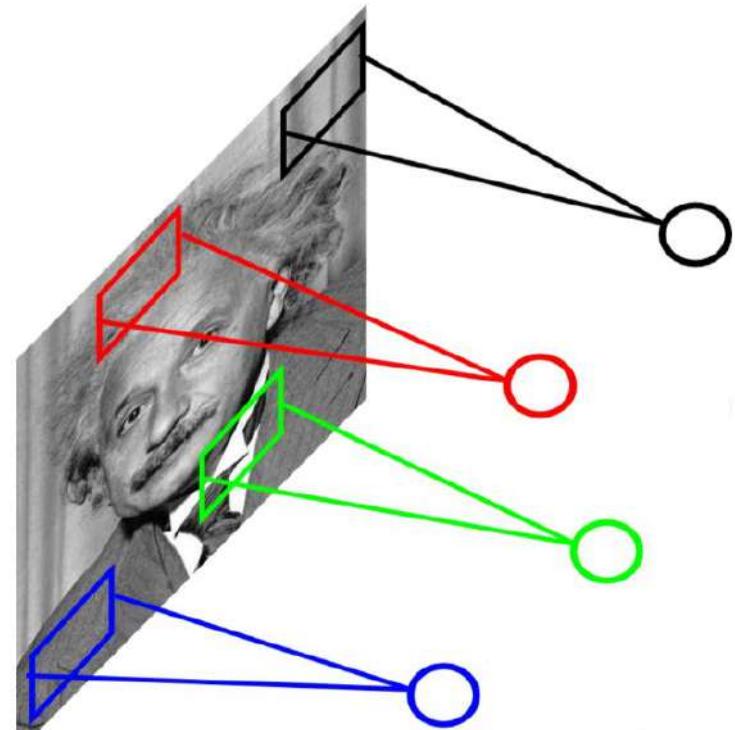
- Soluția CNN la problema numărului mare de parametri:

- straturi conectate local
- neuronii din straturile ascunse se vor axa pe clustere de pixeli



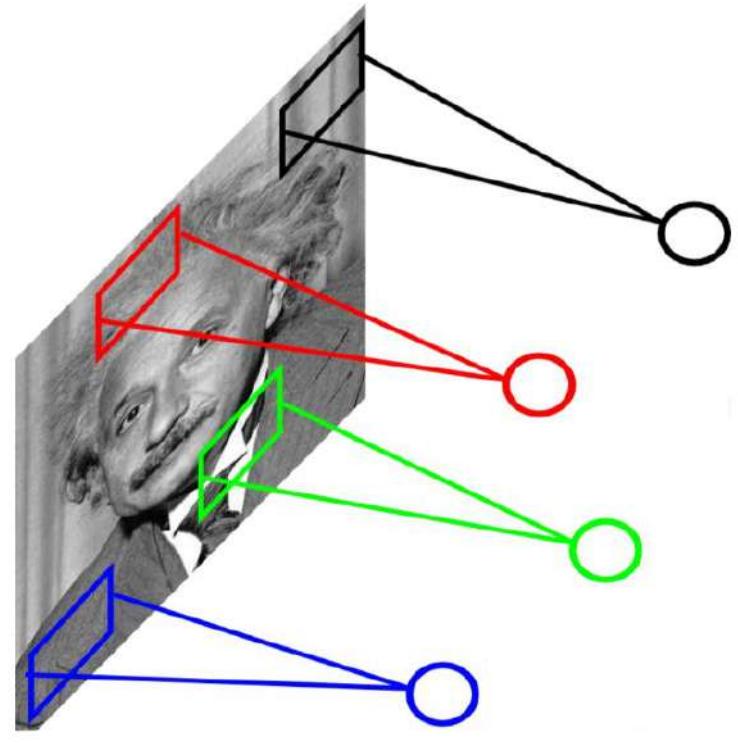
- Soluția CNN la problema numărului mare de parametri:

- fiecare neuron va analiza o subregiune a imaginii
- conexiunile din aceleasi poziții relative ale subregiunilor vor avea ponderi comune



- Soluția CNN la problema numărului mare de parametri:

- reducerea semnificativă a numărului de parametri implicați
- exemplu: pentru subregiuni de  $10 \times 10$  un strat ascuns de  $128 \times 128$  neuroni va avea **1638400** ponderi
  - reducere semnificativă față de cazul anterior al unei rețele neuronale clasice

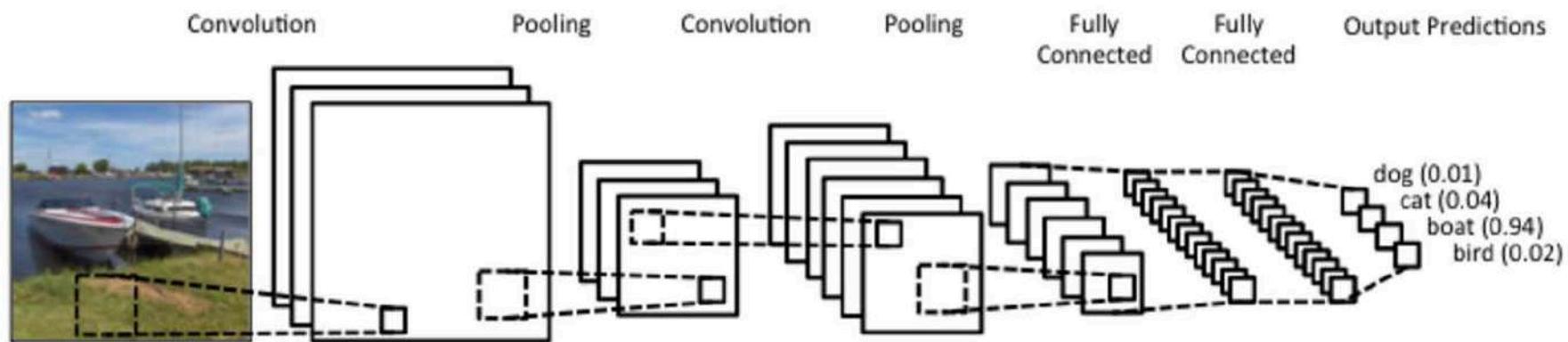


## ■ CNN – clasificare

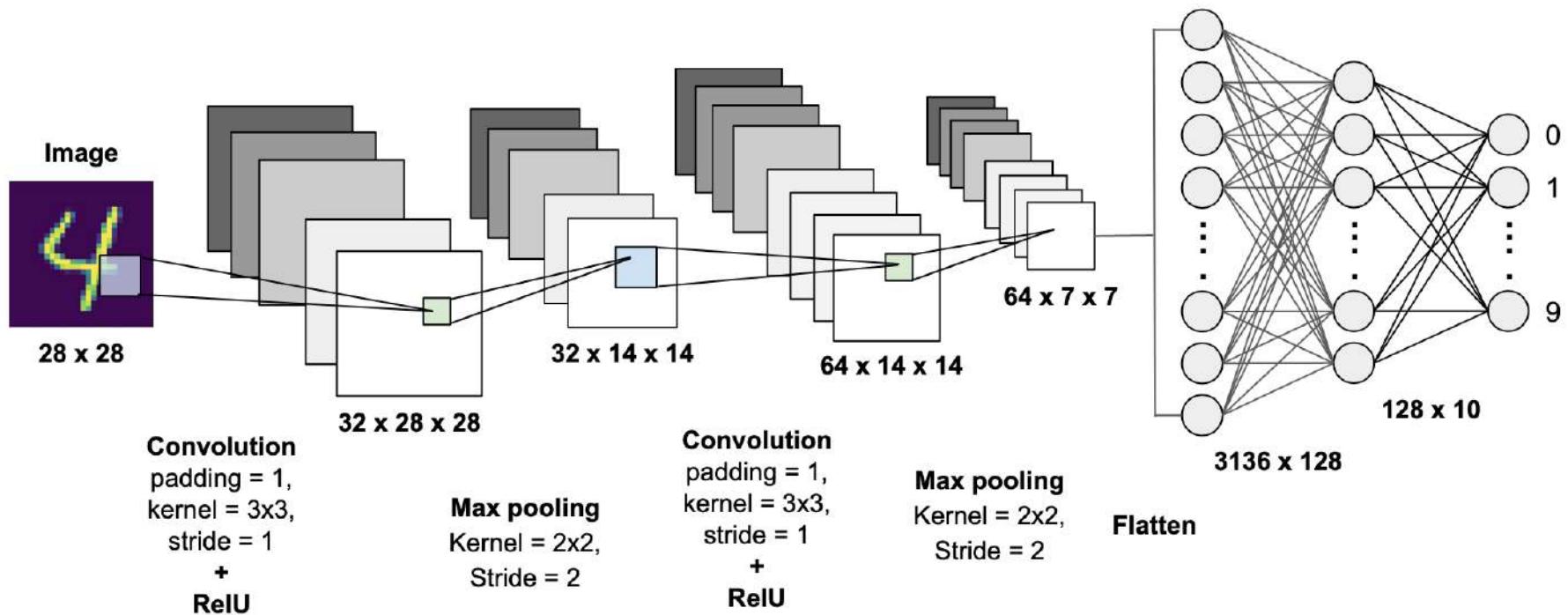
- se dorește detectia anumitor trăsături sau obiecte din imaginea de intrare
- încadrarea acestora într-o serie de clase, cu o anumită probabilitate
- aşadar
  - datele de intrare sunt valorile pixelilor imaginii
    - intensități, în cazul imaginilor grayscale
    - valori RGB, în cazul imaginilor color
  - ieșirile sunt în același număr cu clasele în care dorim să încadrăm imaginea
  - valorile ieșirilor sunt probabilitățile cu care imaginea se încadrează în fiecare clasă

Ex:

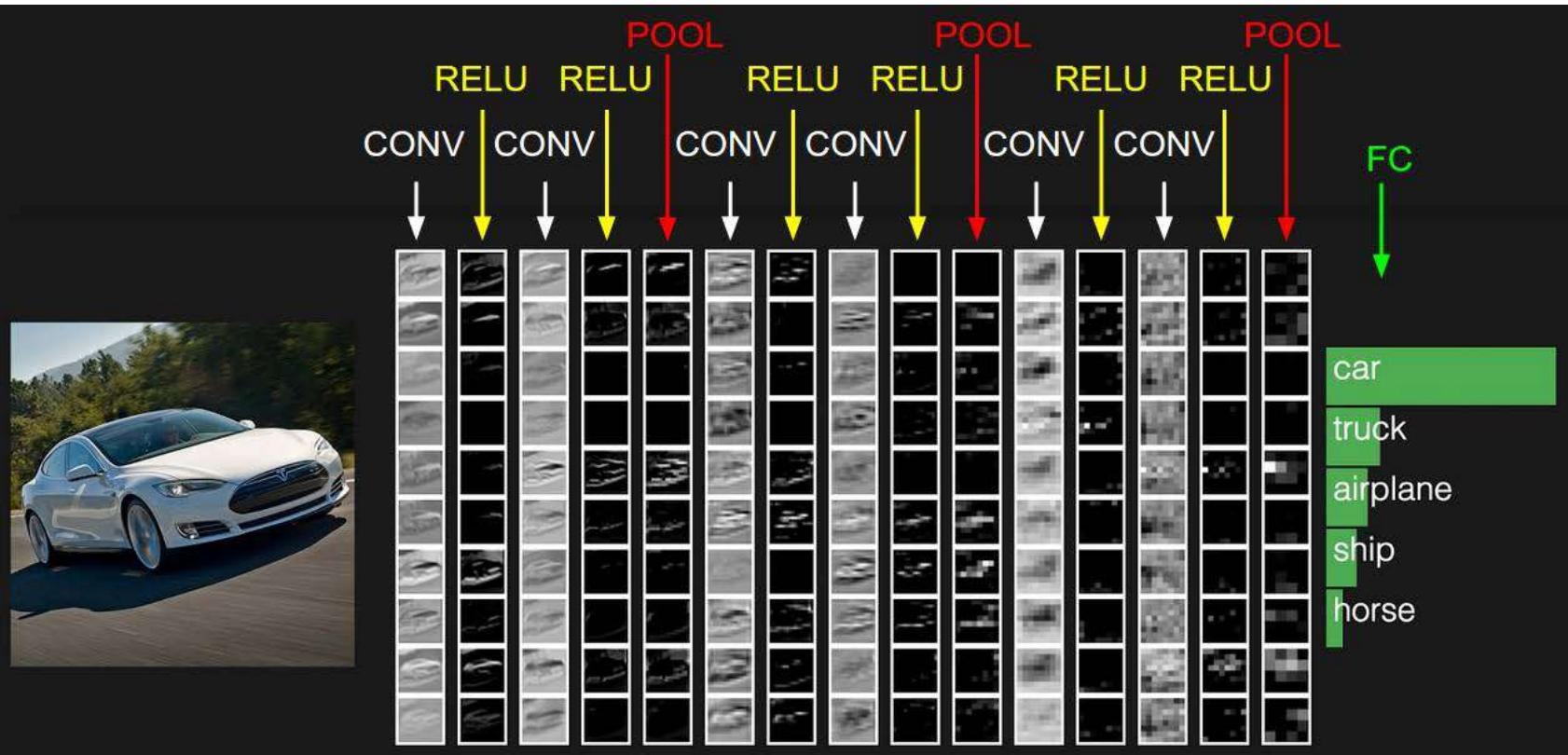
- se pornește cu o imagine unde trebuie identificat un obiect
- se cunosc clasele de apartenență ale obiectului (ce poate reprezenta acesta)
- fiecare clasă se regăsește la output-ul rețelei
- în urma unei succesiuni de operații, se obțin probabilitățile ca obiectul din imagine să aparțină fiecăreia dintre clasele de la ieșire



- Două aspecte importante ale unei CNN
  - Identificarea trăsăturilor
  - Clasificarea pe baza acestor trăsături
- Ca atare, o CNN are două componente majore:
  - o etapă de “feature extraction”, care presupune prelucrarea imaginii de intrare, inițial la nivel de pixel, și ulterior la nivel de trăsătură
  - o etapă de clasificare – asemănătoare cu o rețea neuronală clasică



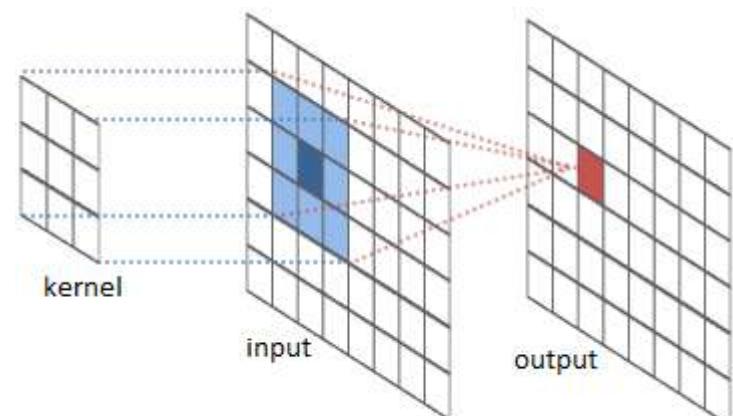
- Etapa de identificare a trăsăturilor presupune următoarele operații
  - Convoluția – se obține o trăsătură oarecare
  - Rectificarea – se aplică funcția de activare ReLU
  - Agregarea (*pooling*) – constă în subeșantionarea rezultatului celor două operații anterioare
- Etapa de clasificare
  - constă într-o rețea neuronală complet conectată (FC – fully connected) care preia trăsăturile și identifică probabilitățile cu care acestea se încadrează în clasele disponibile



- Convoluție
  - operație de filtrare a unei imagini
  - permite
    - identificarea anumitor trăsături
    - atenuarea anumitor componente de frecvență (filtru trecere-jos, trecere-sus etc.)
    - accentuarea anumitor detaliu (contururi, regiuni de interes etc.)
  - termenii sunt imaginea propriu zisă și o matrice numită *kernel*

## ■ Convoluție

- kernelul se centrează în fiecare pixel al imaginii inițiale (asemănător cu o mască)
- pentru toți pixelii peste care “se suprapune” kernel-ul, se realizează suma produselor dintre valorile kernelului și valorile corespunzătoare ale pixelilor din aceleasi poziții
- rezultatul obținut va constitui valoarea pixelului din imaginea care rezultă



- Fie o imagine descrisă prin funcția

$$f(x, y) = I \quad x = 1..w \quad y = 1..h$$

...care asociază fiecărei poziții din plan o valoare  $I$  a intensității (poate fi și o grupare RGB în cazul imaginilor color)

- Fie o matrice  $k$ , numită *kernel* sau *filtru*, de dimensiuni  $n \times m$  cu elementele

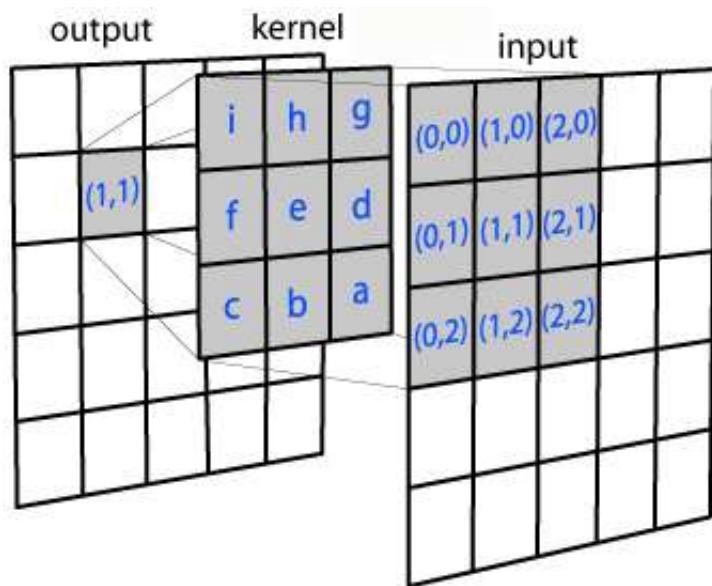
$$k(i, j) \quad i = 1..n, j = 1..m$$

... unde  $n$  și  $m$  sunt mult mai mici decât  $w$ , respectiv  $h$

- Operația de conoluție dintre imagine și matrice
  - rezultă o imagine  $g(x, y)$  de aceleasi dimensiuni ca și cea inițială, unde

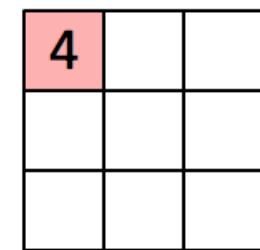
$$g(x, y) = \sum_{i=1}^n \sum_{j=1}^m f\left(x - \frac{n}{2} + i, y - \frac{m}{2} + j\right) k(i, j)$$

$$out(1, 1) = i * in(0,0) + h * in(1,0) + g * in(2,0) + f * in(0,1) + \dots + a * in(2,2)$$



1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	0	0
0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1	0
0 <small><math>\times 1</math></small>	0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1	1
0	0	1	1	0
0	1	1	0	0

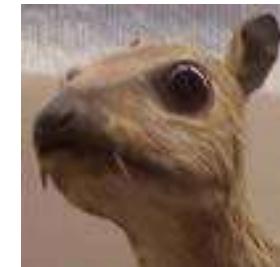
Image



Convolved Feature

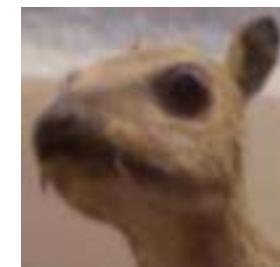
## ■ Convoluție – exemple

- imaginea inițială



- filtru trece-jos (blur)

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



- filtru trece-sus (sharpen)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



- accentuarea contururilor

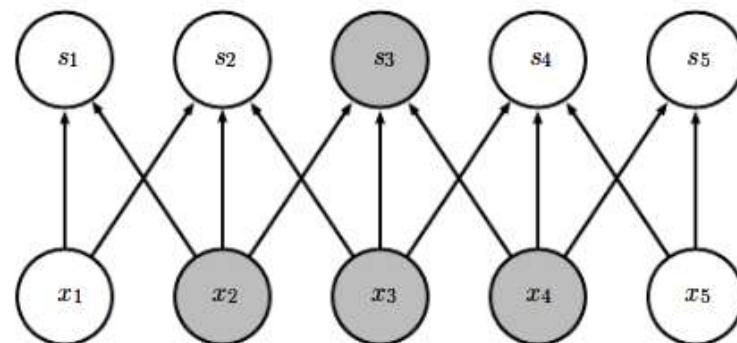
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$



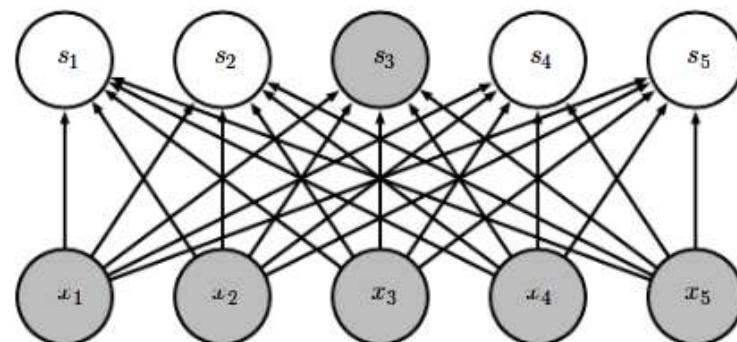
- Aspectele pe care le facilitează operația de convoluție
  - conectivitate redusă (engl. *sparse connectivity*)
    - filtrul de convoluție are dimensiuni mult reduse față de imaginea care se prelucrează
    - scopul este de a detecta anumite trăsături – o porțiune redusă din informația totală existentă în imagine
    - se stochează / actualizează mult mai puțini parametri
      - consum redus de resurse de calcul
      - eficiență dpdv al calculelor statistice
    - se permite interacțiunea indirectă a straturilor ascunse ulterioare cu o porțiune mai mare a datelor de intrare
      - rețeaua poate descrie interacțiuni complexe dintre variabile multiple - prin construirea acestor interacțiuni din componente simple descrise prin intermediul conectivității reduse

- Aspectele pe care le facilitează operația de convoluție
  - conectivitate redusă (engl. *sparse connectivity*)

- conectivitate redusă



- conectivitate completă
  - straturi FC (fully connected)



- Aspectele pe care le facilitează operația de convoluție
  - partajarea parametrilor (engl. *parameter sharing*)
    - se utilizează aceleași filtre pentru întreaga imagine
    - pentru fiecare filtru, există un singur set de ponderi
    - valoarea unei ponderi aplicate unei anumite valori de intrare este aceeași cu valoarea ponderii aplicate altor valori de intrare
      - spre deosebire de cazul rețelelor neuronale tradiționale, unde fiecare intrare are propria pondere
    - aplicarea filtrului pe mai multe porțiuni din imagine nu presupune crearea unui nou set de ponderi

- Aspectele pe care le facilitează operația de convoluție:
  - invarianță la translație
    - aplicarea convoluției pe o regiune dintr-o imagine este independentă de poziția regiunii
    - dacă regiunea este afectată de o transformare de translație, rezultatul convoluției este același
    - de exemplu, detecția contururilor dintr-o imagine nu depinde de poziția acestora
      - un contur nu își modifică proprietățile geometrice în urma unei translații
      - o altă interpretare: conturul unui obiect este același indiferent de poziția obiectului în imagine

- Aspectele pe care le facilitează operația de convoluție:
  - operația de convoluție nu este invariantă la transformări de rotație sau scalare
  - sunt necesare mecanisme suplimentare care să asigure invarianța la aceste transformări
    - SIFT (Scale-Invariant Feature Transform)
    - SURF (Speeded-Up Robust Features)

- Efectul important al operației de convoluție dpdv al CNN

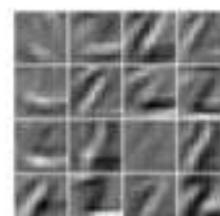
Obținerea unui *feature map* (“hartă de trăsături”)

- = o imagine (sau o colecție de imagini) care vor scoate în evidență acele trăsături din imaginea inițială care fac obiectul clasificării, de exemplu
  - contururi semnificative
  - linii de demarcație între obiecte și background
  - pozițiile diverselor componente ale obiectelor (ochii, nasul, bărbia etc. în cazul recunoașterii faciale)

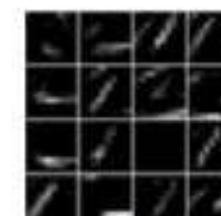
- *Feature maps - exemplu*



conv1



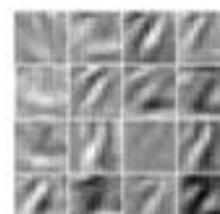
conv2



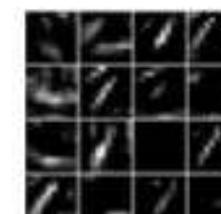
relu2



conv1



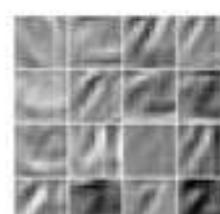
conv2



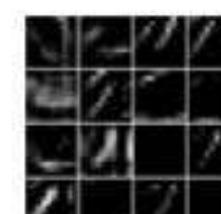
relu2



conv1



conv2



relu2



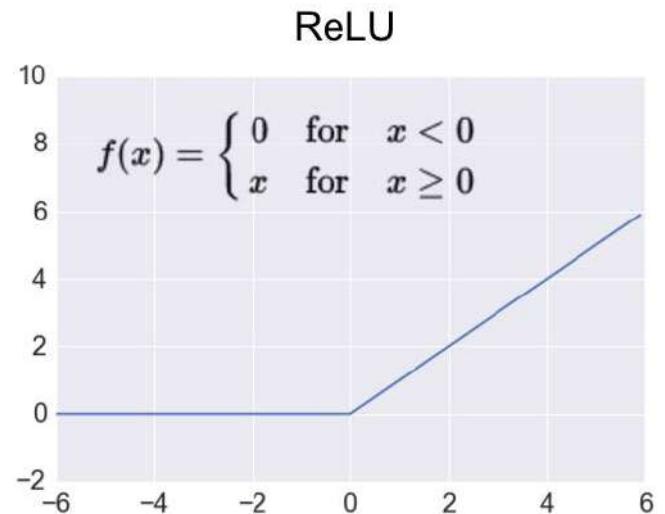
Input

- filtrele pot avea dimensiuni arbitrară (în general impare)
- filtrele din cadrul CNN se învață în cadrul procesului de antrenare
- numărul filtrelor și dimensiunea lor se stabilesc de la bun-început
  - sunt parametri care definesc arhitectura rețelei (“hiperparametri”)
- CNN vor modifica, în cadrul etapei de antrenare, valorile filtrelor cu care se generează feature map-urile

- Dimensiunea unui feature map depinde de:
- numărul de filtre cu care se face convoluția
  - 1 singur filtru => feature map 2D
  - mai multe filtre => feature map 3D, sub forma unui volum
- deplasarea (*stride*) – numărul de pixeli peste care “sare” filtrul la generarea feature map-ului
  - stride = 1 => feature map de aceeași dimensiune ca și imaginea inițială
  - stride > 1 => feature map de dimensiuni mai mici
- măsura în care imaginea inițială se “bordează” cu zero-uri, pentru a putea aplica filtrul pe pixelii din margini

## ■ Rectificarea

- etapă ulterioară operației de conoluție
- se dorește eliminarea valorilor negative din feature map
- toate valorile negative din feature map sunt setate pe 0
- se aplică funcția de activare ReLU



## ■ Rectificarea

- scopul rectificării este de a conferi CNN un comportament neliniar în etapa de identificare a trăsăturilor
- operația de convoluție este liniară

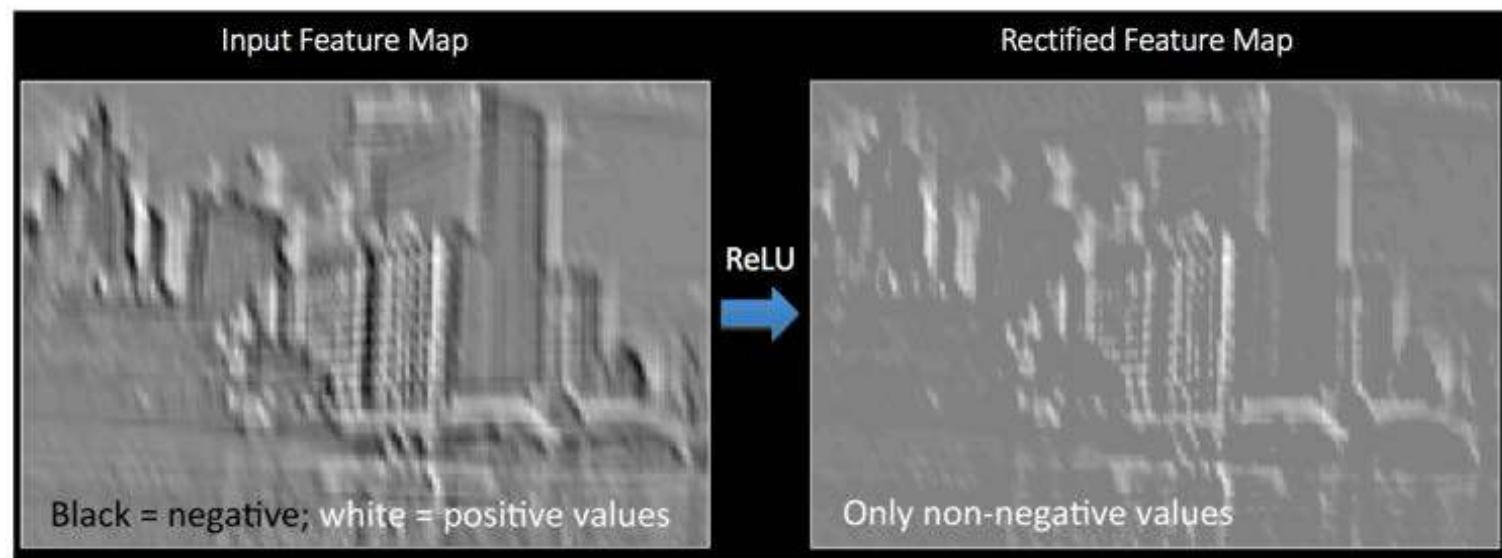
$$g(x, y) = \sum_{i=1}^n \sum_{j=1}^m f\left(x - \frac{n}{2} + i, y - \frac{m}{2} + j\right) k(i, j)$$

- $g(x, y)$  este un polinom de gradul I în x, y

## ■ Rectificarea

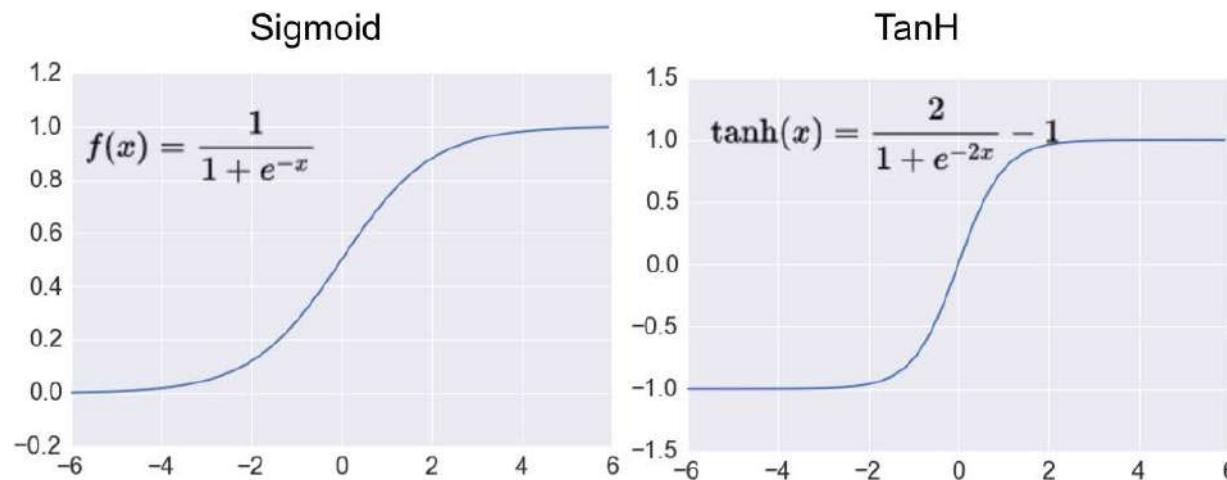
- în absența rectificării:
  - componenta de identificare a trăsăturilor se reduce la o multitudine de operații liniare
  - rețeaua se poate reduce la un perceptron cu un singur strat
    - cu funcție de activare liniară
    - funcția de activare rezultă din compunerea operațiilor liniare anterioare
  - poate realiza clasificarea corectă doar dacă spațiul de soluții este liniar separabil
    - există un hiperplan care separă hiperspațiul soluțiilor în două semi-hiperspații ce conțin elemente din clase disjuncte

## ■ Rectificarea



## ■ Rectificarea

- Alternative: funcția sigmoid, funcția tanh



- Totuși s-a demonstrat faptul că ReLU oferă cele mai bune rezultate în majoritatea situațiilor

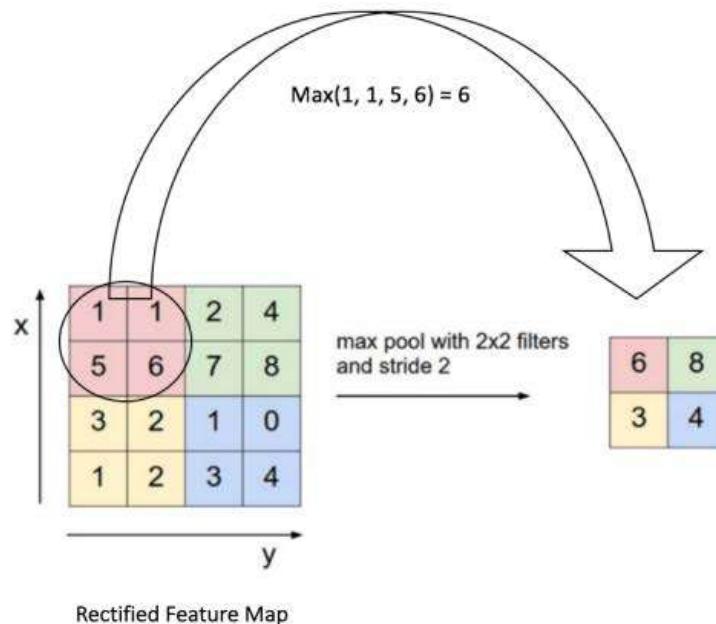
## ■ Agregarea (pooling)

- implică o subeșantionare a feature map-ului
- se reduce dimensiunea feature map-ului, dar se pastrează detaliile importante
- se alege o subimagine de dimensiuni mici din feature map
- se reduce subimaginea la o singură valoare
- de ex:
  - feature map-ul are dimensiunea 256x256
  - subimaginea este 2x2
  - în urma agregării rezultă un feature map de 128x128 cu aceleași trăsături

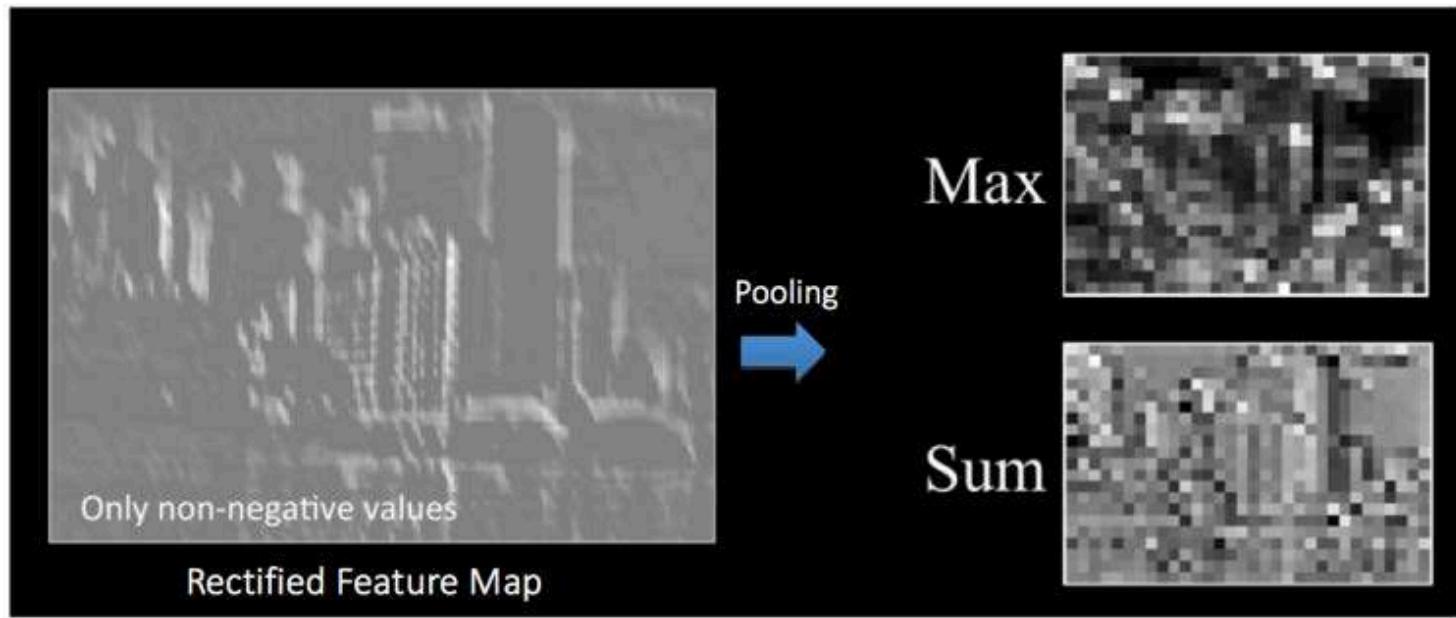
- Agregarea (pooling)
- De mai multe tipuri:
  - Max Pooling = din fiecare subimagine se alege valoarea maximă
  - Sum Pooling = valoarea rezultată este suma valorilor din subimaginea corespunzătoare
  - Average Pooling = valoarea rezultată este media valorilor din subimaginea corespunzătoare

## ■ Agregarea (pooling)

- Exemplu: Max Pooling = agregare care implică reducerea unei subimagini pixelul cu valoarea maximă



- Agregarea (pooling)

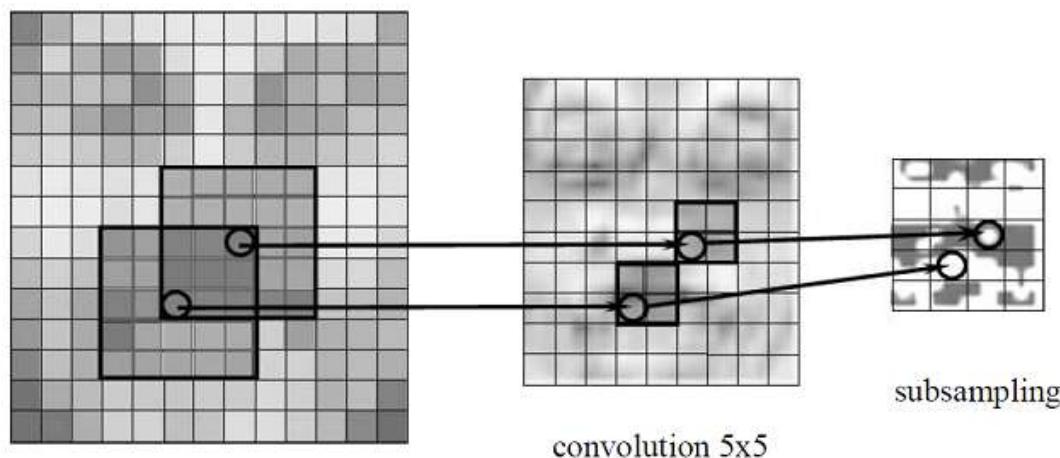


- Agregarea (pooling)

- Scop:
  - reducerea dimensiunii reprezentării imaginilor de la intrare
  - reducerea numărului de parametri din CNN – reducerea efectului de overfitting
  - trăsăturile selectate vor fi invariante la transformări geometrice de translație, distorsionare etc.
    - aceste transformări nu vor schimba în mod semnificativ (sau deloc) rezultatul calculului maximului, al valorii medii etc.

## ■ Agregarea (pooling)

- rezultat: obiectele vor fi detectate indiferent de poziția lor în imagine, indiferent de gradul de distorsiune la care sunt supuse
  - dacă CNN este configurață corect



# Învățare automată

## 9. Rețele neuronale conveționale II

**Marius Gavrilăescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

## ■ Straturi multiple

### ■ Convoluție

- operații de conoluție cu unul sau mai multe filtre
- rezultă una sau mai multe feature maps
- valorile filtrelor se învață prin antrenare

### ■ Agregare (*pooling*)

- se reduce dimensiunea imaginilor (a feature map-urilor)
- diminuarea numărului de parametri, simplificarea arhitecturii rețelei, reducerea calculelor necesare pentru clasificare / antrenare

### ■ Fully connected (se mai numesc și straturi *dense*)

- straturi clasice, complet conectate (pentru fiecare pereche de neuroni din straturi vecine există câte o pondere)
- realizează clasificarea propriu-zisă pe baza feature map-urilor din straturile anterioare

## Exemplu

- clasificarea imaginilor care conțin cifre
- identificarea cifrelor din imagini



...



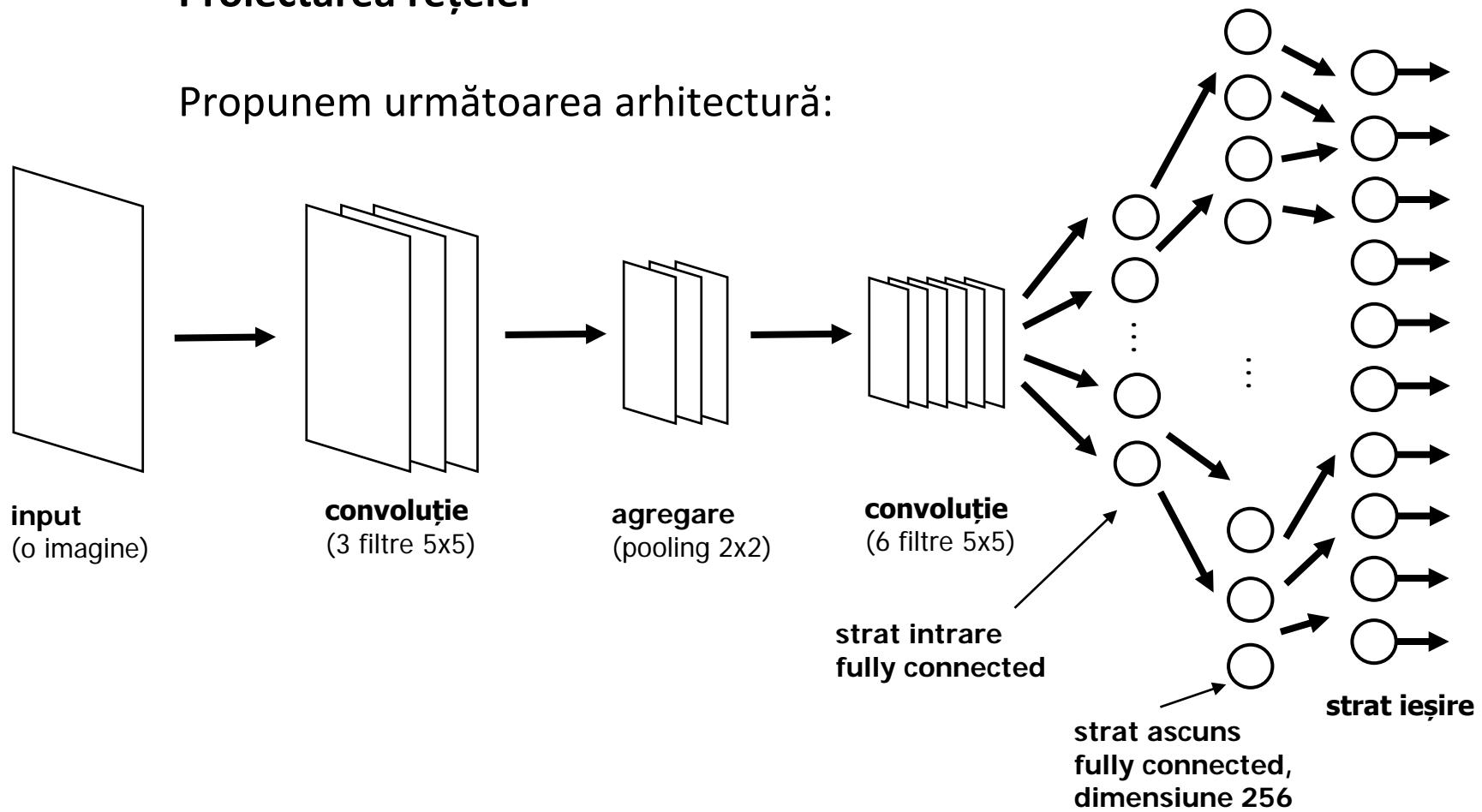
imagini 28x28, grayscale

## **Etape**

- proiectarea rețelei de conoluție:
  - stabilirea straturilor, a parametrilor lor
  - stabilirea arhitecturii rețelei, a configurației straturilor
- antrenarea rețelei
  - se obțin date pentru care se cunoaște soluția (imagini în care a fost deja identificată cifra corespunzătoare)
  - se folosesc aceste date pentru a ajusta parametrii rețelei
  - rețeaua va face asocieri între imaginile de la intrare și cifrele pe care le conțin
- evaluarea rețelei
  - determinarea acurateței rețelei folosind date de test (alte imagini decât cele utilizate pentru antrenare)

## Proiectarea rețelei

Propunem următoarea arhitectură:



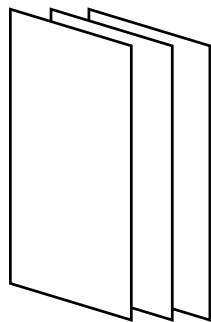


**input**  
(o imagine)

### **Stratul de intrare**

- dimensiunea 28x28
- valorile pixelilor imaginii

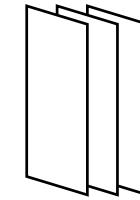
## **Primul strat de conoluție**



- 3 filtre  $5 \times 5$
- din imaginea din stratul de intrare se obțin 3 feature maps de dimensiune  $24 \times 24$
- dimensiunea redusa rezultă din modul de aplicare al filtrului (rămâne o margine de 2 pixeli nefiltrată)
- pe valorile feature map-urilor se aplică funcția de activare ReLU

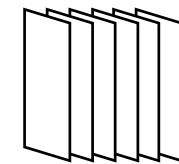
## **Stratul de agregare**

- reducerea dimensiunii feature map-urilor din stratul anterior cu  $2 \times 2$
- rezultă 3 feature map-uri de dimensiune  $12 \times 12$



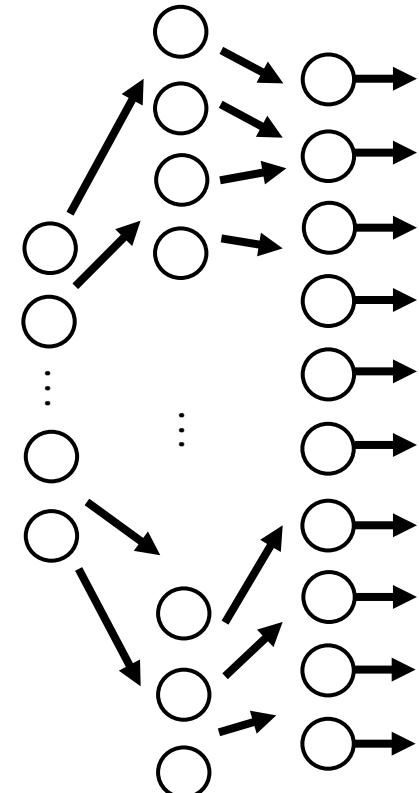
## Al doilea strat de convoluție

- 6 filtre 5x5
- din fiecare feature map generat de stratul anterior rezultă 6 feature map-uri noi
- în total se generează  $3 \times 6 = 18$  feature map-uri de dimensiune 8x8
- pe valorile feature map-urilor se aplică funcția de activare ReLU



## Straturile fully-connected

- straturi “clasice”, neuronii din straturi succesive sunt interconectați prin ponderi
- primul strat - dimensiunea totală a ieșirilor din stratul de conoluție anterior
- al doilea strat
  - dimensiunea 256
  - activare sigmoid
- stratul de ieșire
  - dimensiunea 10 (câte o ieșire pentru fiecare cifră)
  - activare softmax
  - fiecare ieșire = probabilitatea ca imaginea inițială să conțină cifra corespunzătoare



## ■ Antrenarea CNN

Datele de antrenare arată astfel:

<b>0</b>	0	<b>1</b>	1	<b>2</b>	2		<b>5</b>	5		<b>9</b>	9
<b>0</b>	0	<b>1</b>	1	<b>2</b>	2	...	<b>5</b>	5	...	<b>9</b>	9
<b>0</b>	0	<b>1</b>	1	<b>2</b>	2		<b>5</b>	5		<b>9</b>	9
:		:		:			:			:	

approx. 1000 imagini din fiecare categorie

## ■ Antrenarea CNN

Se realizează codificarea *one hot* a claselor:

0 – [0 0 0 0 0 0 0 0 1]

1 – [0 0 0 0 0 0 0 1 0]

2 – [0 0 0 0 0 0 1 0 0]

3 – [0 0 0 0 0 1 0 0 0]

4 – [0 0 0 0 1 0 0 0 0]

5 – [0 0 0 1 0 0 0 0 0]

6 – [0 0 1 0 0 0 0 0 0]

7 – [0 0 1 0 0 0 0 0 0]

8 – [0 1 0 0 0 0 0 0 0]

9 – [1 0 0 0 0 0 0 0 0]

## ■ Antrenarea CNN

- se initializează filtrele și ponderile cu valori aleatoare
- CNN primește la intrare, rând pe rând, imaginile din setul de date de antrenare
- CNN primesc la ieșire vectorii *one hot* corespunzători imaginilor

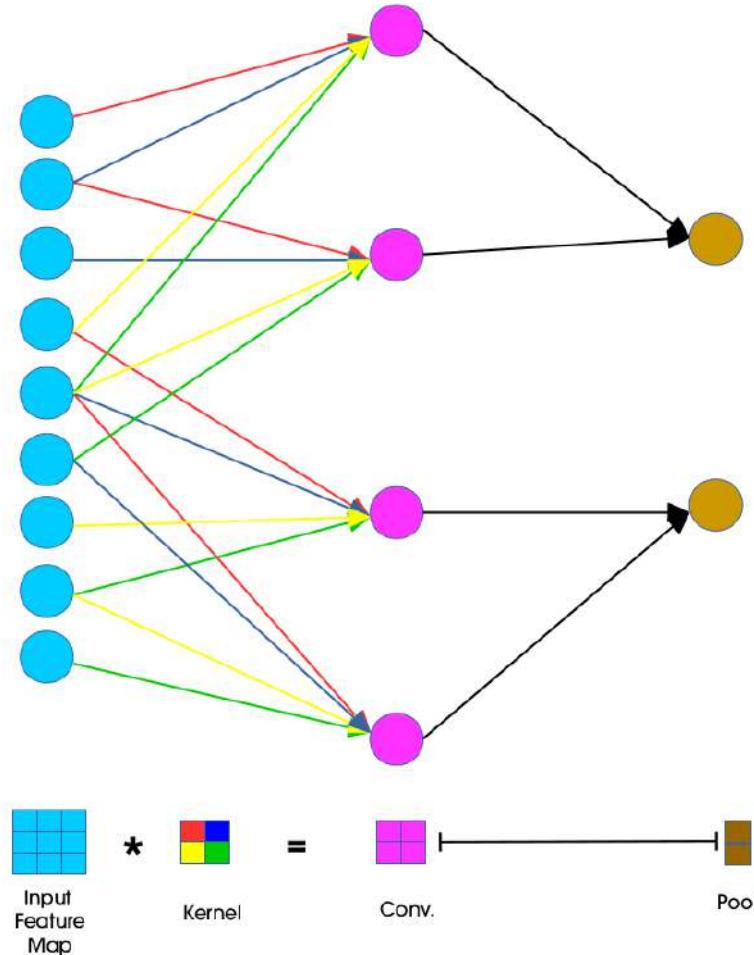
## ■ Antrenarea CNN

- imaginile se propagă înainte prin CNN (*forward propagation*)
  - imaginile sunt supuse operațiilor de convoluție, rectificare, agregare
    - rezultă setul de feature maps
  - aceste feature maps se propagă prin componenta FC
  - rezultă la ieșire un vector cu probabilitățile de încadrare în fiecare clasă

## ■ Antrenarea CNN

### ■ Principalele operații

- conoluția dintre un filtru (*kernel*) și feature-map-urile din acel strat
- agregarea – reducerea dimensiunii feature-map-ului (optional)



## ■ Antrenarea CNN

- în urma efectuării tuturor operațiilor din faza de propagare înapoi rezultă o eroare
  - eroare = diferența dintre vectorul de valori furnizat inițial și cel obținut în urma propagării înapoi
- pentru calculul erorii se utilizează frecvent:
  - eroarea medie pătratică (Mean Squared Error):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- entropia încrucișată (Cross Entropy)

$$CS = - \sum_{i=1}^n y_i \log \hat{y}_i$$

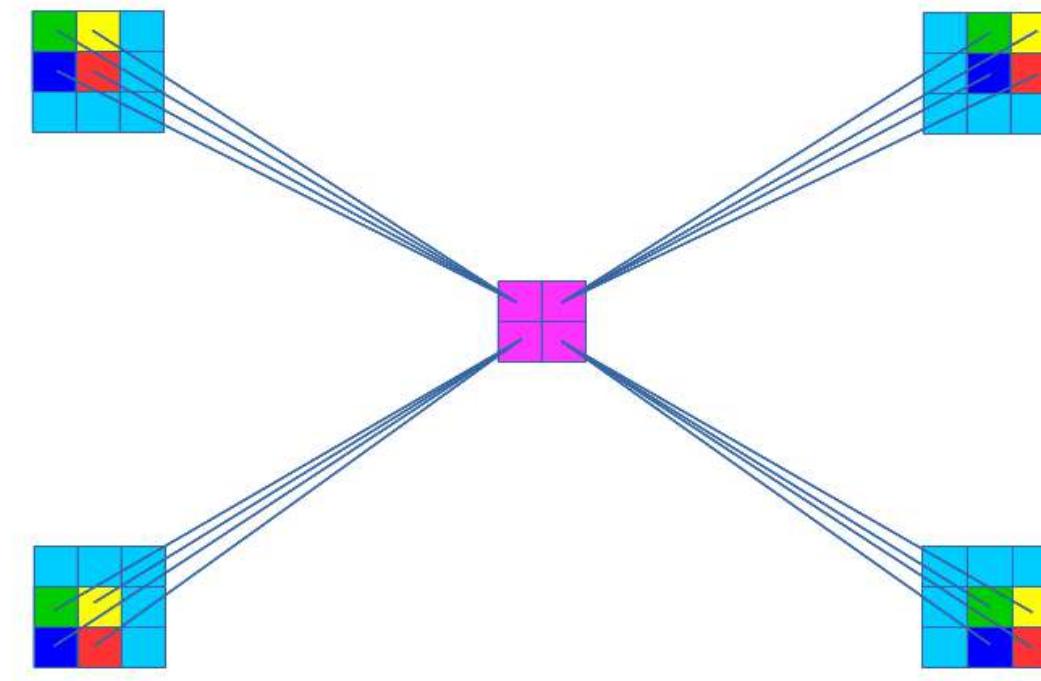
- unde:
  - n = nr de elemente ale vectorului de la ieșirea rețelei (de exemplu numărul de clase)
  - y = elementele vectorului din setul de date de antrenare (de exemplu elementele vectorilor one-hot ai claselor)
  - $\hat{y}$  = elementele vectorului generat de rețea

## ■ Antrenarea CNN

- **scopul antrenării:** ajustarea ponderilor  $w$  ale rețelei astfel încât să se minimizeze eroarea  $E$  dintre output-ul acesteia și valorile target din setul de date de antrenare
- apare o eroare nenulă (sau prea mare) deoarece ponderile nu au valori adecvate
- trebuie determinată variația erorii cauzată de ponderi:
  - gradienții erorii în raport cu ponderile:  $\frac{\partial E}{\partial w}$
- se ajustează ponderile funcție de gradienții erorii utilizând o metodă de optimizare (ex. metoda gradientului descendente – Gradient Descent)

- Antrenarea CNN
- un pas complet al antrenării implică:
  - determinarea valorilor de la ieșirea rețelei
    - aplicarea operațiilor aferente straturilor rețelei:
      - pe valorile de intrare ale rețelei și pe valorile din fiecare strat
      - utilizarea valorilor actuale ale ponderilor  $w$
    - operațiile se realizează pornind de la stratul de intrare și încheind cu valorile stratului de ieșire ("propagare înainte")
  - determinarea gradenților erorii rețelei în raport cu ponderile
    - gradienții se calculează în fiecare strat, pornind de la stratul de ieșire (unde apare eroarea) și încheind cu stratul de intrare ("propagarea înapoi a gradenților prin straturile rețelei")
    - ajustarea ponderilor funcție de gradienți
      - de exemplu, în cazul metodei gradientului descendente:  $w = w - \alpha \frac{\partial E}{\partial w}$
      - $\alpha$  = rata de învățare (valoare reală în general subunitară)

- Antrenarea CNN
  - Propagarea înainte printr-un strat convolutional - exemplu



## ■ Antrenarea CNN

### ■ Propagarea înainte printr-un strat conoluțional - exemplu

- conoluția dintre un feature map X și un filtru W generează un feature map O

$$\begin{bmatrix} o_{11} & o_{12} \\ o_{21} & o_{22} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \otimes \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

- valorile noului feature map sunt:

$$o_{11} = x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22}$$

$$o_{12} = x_{12}w_{11} + x_{13}w_{12} + x_{22}w_{21} + x_{23}w_{22}$$

$$o_{21} = x_{21}w_{11} + x_{22}w_{12} + x_{31}w_{21} + x_{32}w_{22}$$

$$o_{22} = x_{22}w_{11} + x_{23}w_{12} + x_{32}w_{21} + x_{33}w_{22}$$

## ■ Antrenarea CNN

- **Propagarea înapoi printr-un strat convolutional - exemplu**
- trebuie determinați
  - gradienții erorii E în raport cu ponderile W
    - pentru a putea ajusta valorile ponderilor
  - gradienții erorii E în raport cu valorile de intrare X
    - pentru a putea face mai departe calculele atunci când se trece la stratul anterior – valorile de intrare X sunt valorile de ieșire ale stratului anterior
- regula de descompunere în lanț a derivatelor (“chain rule”):

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

## ■ Antrenarea CNN

### ■ Propagarea înapoi printr-un strat convolutional – exemplu

- valorile de la ieșirea rețelei se obțin astfel:

$$O = X \otimes W$$

- în ultimul strat al rețelei rezultă o eroare E

$$\frac{\partial E}{\partial O} = \text{gradientul erorii in raport cu valorile de iesire}$$

- gradientul erorii E în raport cu intrarea X se poate descompune astfel:

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial X}$$

- gradientul erorii E în raport cu ponderile W se poate descompune astfel:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial W}$$

## ■ Antrenarea CNN

- Propagarea înapoi printr-un strat convolutional – exemplu

- pentru una dintre valorile de ieșire:

$$o_{11} = x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22}$$

- gradienții în raport cu ponderile  $w_{ij}$  sunt:

$$\frac{\partial o_{11}}{\partial w_{11}} = x_{11}$$

$$\frac{\partial o_{12}}{\partial w_{12}} = x_{12}$$

$$\frac{\partial o_{21}}{\partial w_{21}} = x_{21}$$

$$\frac{\partial o_{22}}{\partial w_{22}} = x_{22}$$

- similar, se determină gradienții celorlalte valori de ieșire

## ■ Antrenarea CNN

- Propagarea înapoi printr-un strat convolutional – exemplu
- gradienții erorii în raport cu ponderile

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial W}$$

- $O$  și  $W$  sunt matrice. Regula descompunerii derivatelor se aplică astfel:

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^M \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_i}$$

- pentru fiecare pondere  $w_i$  din matricea  $W$ , se sumează descompunerile care rezultă pentru fiecare  $o_k$  din matricea  $O$

## ■ Antrenarea CNN

- Propagarea înapoi printr-un strat convolutional – exemplu
- gradienții erorilor în raport cu fiecare pondere:

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{11}} + \frac{\partial E}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{11}} + \frac{\partial E}{\partial o_{21}} \frac{\partial o_{21}}{\partial w_{11}} + \frac{\partial E}{\partial o_{22}} \frac{\partial o_{22}}{\partial w_{11}}$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{12}} + \frac{\partial E}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{12}} + \frac{\partial E}{\partial o_{21}} \frac{\partial o_{21}}{\partial w_{12}} + \frac{\partial E}{\partial o_{22}} \frac{\partial o_{22}}{\partial w_{12}}$$

$$\frac{\partial E}{\partial w_{21}} = \frac{\partial E}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{21}} + \frac{\partial E}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{21}} + \frac{\partial E}{\partial o_{21}} \frac{\partial o_{21}}{\partial w_{21}} + \frac{\partial E}{\partial o_{22}} \frac{\partial o_{22}}{\partial w_{21}}$$

$$\frac{\partial E}{\partial w_{22}} = \frac{\partial E}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{22}} + \frac{\partial E}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{22}} + \frac{\partial E}{\partial o_{21}} \frac{\partial o_{21}}{\partial w_{22}} + \frac{\partial E}{\partial o_{22}} \frac{\partial o_{22}}{\partial w_{22}}$$

## ■ Antrenarea CNN

- **Propagarea înapoi printr-un strat convolutional – exemplu**
- anterior, s-au determinat gradienții ieșirilor în raport cu ponderile. Înlocuind, rezultă:

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial o_{11}}x_{11} + \frac{\partial E}{\partial o_{12}}x_{12} + \frac{\partial E}{\partial o_{21}}x_{21} + \frac{\partial E}{\partial o_{22}}x_{22}$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial o_{11}}x_{12} + \frac{\partial E}{\partial o_{12}}x_{13} + \frac{\partial E}{\partial o_{21}}x_{22} + \frac{\partial E}{\partial o_{22}}x_{23}$$

$$\frac{\partial E}{\partial w_{21}} = \frac{\partial E}{\partial o_{11}}x_{21} + \frac{\partial E}{\partial o_{12}}x_{22} + \frac{\partial E}{\partial o_{21}}x_{31} + \frac{\partial E}{\partial o_{22}}x_{32}$$

$$\frac{\partial E}{\partial w_{22}} = \frac{\partial E}{\partial o_{11}}x_{22} + \frac{\partial E}{\partial o_{12}}x_{23} + \frac{\partial E}{\partial o_{21}}x_{32} + \frac{\partial E}{\partial o_{22}}x_{33}$$

- ceea ce reprezintă o operație de conoluție:

$$\begin{bmatrix} \frac{\partial E}{\partial w_{11}} & \frac{\partial E}{\partial w_{12}} \\ \frac{\partial E}{\partial w_{21}} & \frac{\partial E}{\partial w_{22}} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \otimes \begin{bmatrix} \frac{\partial E}{\partial o_{11}} & \frac{\partial E}{\partial o_{12}} \\ \frac{\partial E}{\partial o_{21}} & \frac{\partial E}{\partial o_{22}} \end{bmatrix}$$

- Antrenarea CNN
  - Propagarea înapoi printr-un strat convolutional – exemplu
  - prin urmare: gradienții erorii în raport cu ponderile se determină realizând o conoluție între:
    - valorile de intrare  $X$  în stratul curent
    - gradienții erorii din stratul următor (ale cărui intrări sunt valorile de ieșire din stratul curent)

$$\begin{bmatrix} \frac{\partial E}{\partial w_{11}} & \frac{\partial E}{\partial w_{12}} \\ \frac{\partial E}{\partial w_{21}} & \frac{\partial E}{\partial w_{22}} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \otimes \begin{bmatrix} \frac{\partial E}{\partial o_{11}} & \frac{\partial E}{\partial o_{12}} \\ \frac{\partial E}{\partial o_{21}} & \frac{\partial E}{\partial o_{22}} \end{bmatrix}$$

## ■ Antrenarea CNN

- Propagarea înapoi printr-un strat convolutional – exemplu
- gradienții erorii în raport cu intrările

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial X}$$

- gradienții ieșirii în raport cu valorile de intrare:
  - valorile de ieșire se obțin astfel

$$o_{11} = x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22}$$

- de unde rezultă:  $\frac{\partial o_{11}}{\partial x_{11}} = w_{11}$

$$\frac{\partial o_{12}}{\partial x_{12}} = w_{12}$$

$$\frac{\partial o_{21}}{\partial x_{21}} = w_{21}$$

$$\frac{\partial o_{22}}{\partial x_{22}} = w_{22}$$

- În mod similar se determină gradienții pentru  $o_{12}, o_{21}, o_{22}$

## ■ Antrenarea CNN

- Propagarea înapoi printr-un strat convolutional – exemplu
- gradienții erorii în raport cu intrările

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial X}$$

- O și X sunt matrice, așadar se aplică regula descompunerii în lanț a derivatelor pentru matrice:

$$\frac{\partial E}{\partial x_i} = \sum_{k=1}^M \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial x_i}$$

- de unde rezultă...

## ■ Antrenarea CNN

### ■ Propagarea înapoi printr-un strat convolutional – exemplu

- gradientii erorii în raport cu fiecare valoare de intrare:

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial o_{11}} w_{11}$$

$$\frac{\partial E}{\partial x_{12}} = \frac{\partial E}{\partial o_{11}} w_{12} + \frac{\partial E}{\partial o_{12}} w_{11}$$

$$\frac{\partial E}{\partial x_{13}} = \frac{\partial E}{\partial o_{12}} w_{12}$$

$$\frac{\partial E}{\partial x_{21}} = \frac{\partial E}{\partial o_{11}} w_{21} + \frac{\partial E}{\partial o_{21}} w_{11}$$

$$\frac{\partial E}{\partial x_{22}} = \frac{\partial E}{\partial o_{11}} w_{22} + \frac{\partial E}{\partial o_{12}} w_{21} + \frac{\partial E}{\partial o_{21}} w_{12} + \frac{\partial E}{\partial o_{22}} w_{11}$$

$$\frac{\partial E}{\partial x_{23}} = \frac{\partial E}{\partial o_{12}} w_{22} + \frac{\partial E}{\partial o_{22}} w_{12}$$

$$\frac{\partial E}{\partial x_{31}} = \frac{\partial E}{\partial o_{21}} w_{21}$$

$$\frac{\partial E}{\partial x_{32}} = \frac{\partial E}{\partial o_{21}} w_{22} + \frac{\partial E}{\partial o_{22}} w_{21}$$

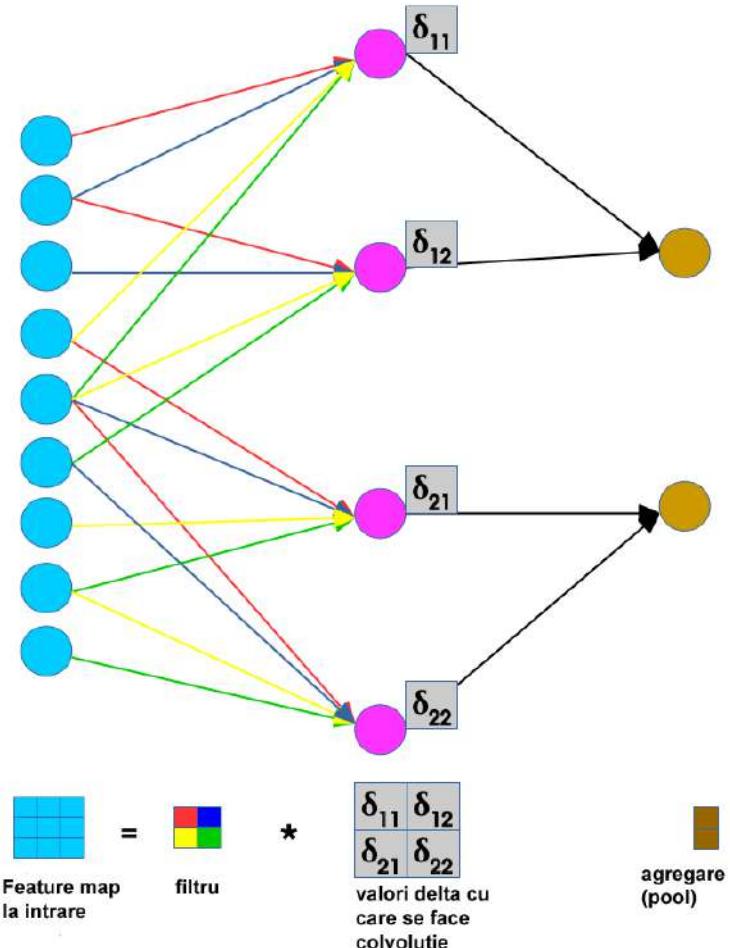
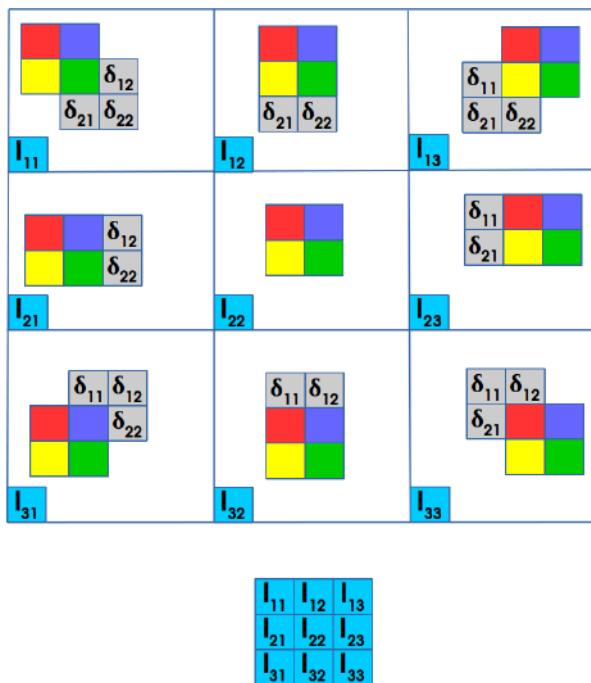
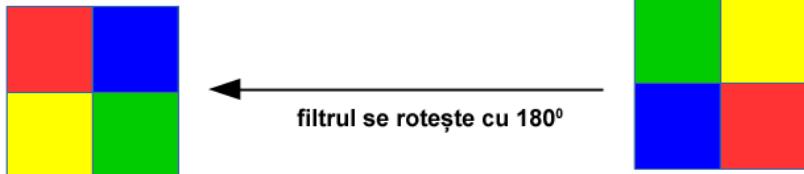
$$\frac{\partial E}{\partial x_{33}} = \frac{\partial E}{\partial o_{22}} w_{22}$$

- Antrenarea CNN
  - Propagarea înapoi printr-un strat convolutional – exemplu

- rezultatul anterior reprezintă o operație de conoluție

$$\begin{bmatrix} \frac{\partial E}{\partial x_{11}} & \frac{\partial E}{\partial x_{12}} & \frac{\partial E}{\partial x_{13}} \\ \frac{\partial E}{\partial x_{21}} & \frac{\partial E}{\partial x_{22}} & \frac{\partial E}{\partial x_{23}} \\ \frac{\partial E}{\partial x_{31}} & \frac{\partial E}{\partial x_{32}} & \frac{\partial E}{\partial x_{33}} \end{bmatrix} = \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix} \otimes \begin{bmatrix} \frac{\partial E}{\partial o_{11}} & \frac{\partial E}{\partial o_{12}} \\ \frac{\partial E}{\partial o_{21}} & \frac{\partial E}{\partial o_{22}} \end{bmatrix}$$

- așadar, gradienții erorii în raport valorile de intrare se obțin prin conoluția dintre:
    - filtrul convolutional rotit cu  $180^0$
    - gradienții erorii din stratul următor (se mai numesc *valori delta*)

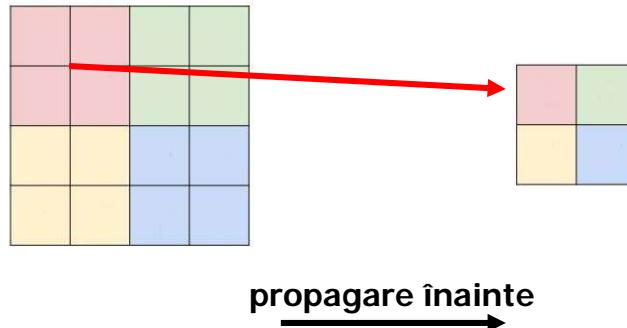


## ■ Antrenarea CNN

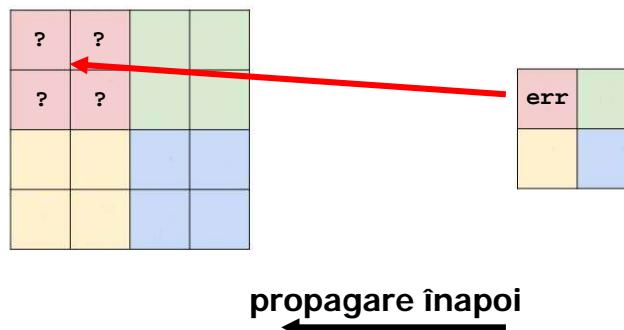
### ■ straturile de agregare

- eroarea se determină pentru un feature map de dimensiuni mai mici decât cel anterior
- trebuie deduse valorile pentru feature map-ul anterior, care este de dimensiuni mai mari
- valorile depind de tipul de agregare (max pooling, sum pooling, average pooling)

## ■ Antrenarea CNN



- se generează un feature map de dimensiune redusă
- un bloc  $2 \times 2$  se transformă într-o valoare scalară

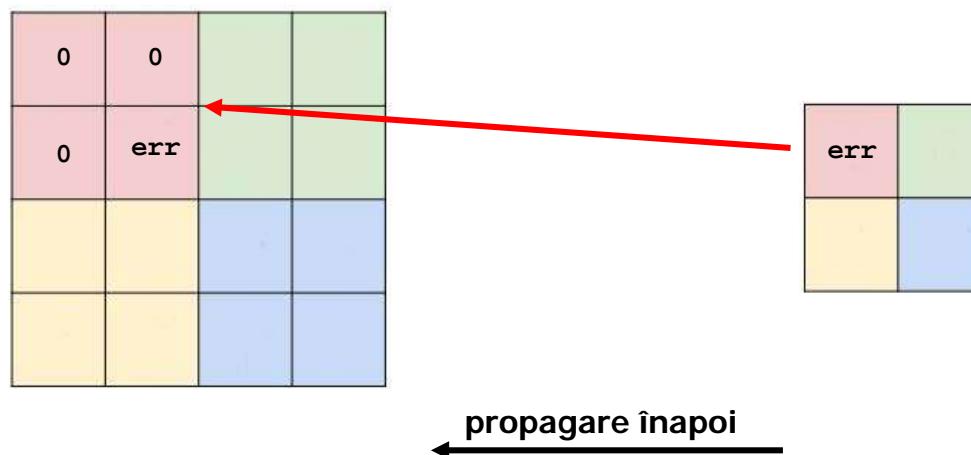


- feature map-ul de dimensiune redusă conține valori care exprimă eroarea din acel pas
- care sunt valorile blocurilor  $2 \times 2$  echivalente ?

## ■ Antrenarea CNN

### ■ max pooling

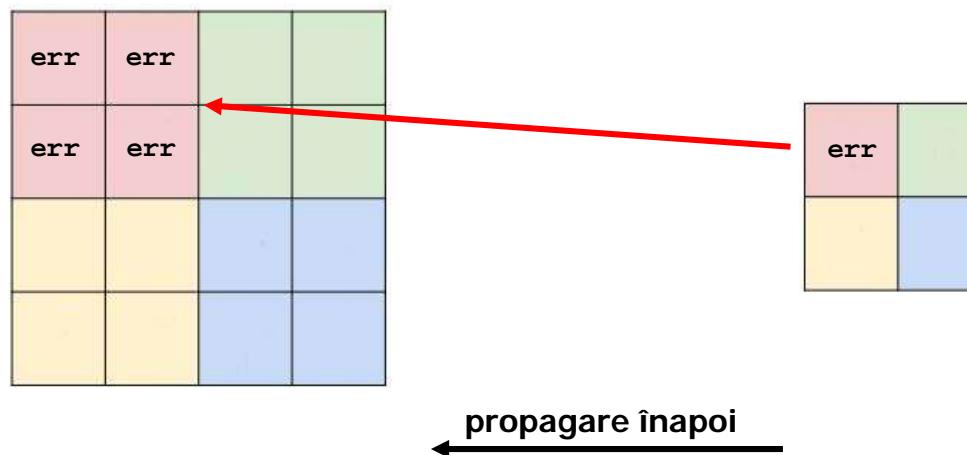
- elementul corespunzător valorii maxime din etapa de propagare înainte este singurul care contribuie la eroare
- trebuie reținut indexul său
- acest element primește valoarea erorii, restul vor fi nuli



## ■ Antrenarea CNN

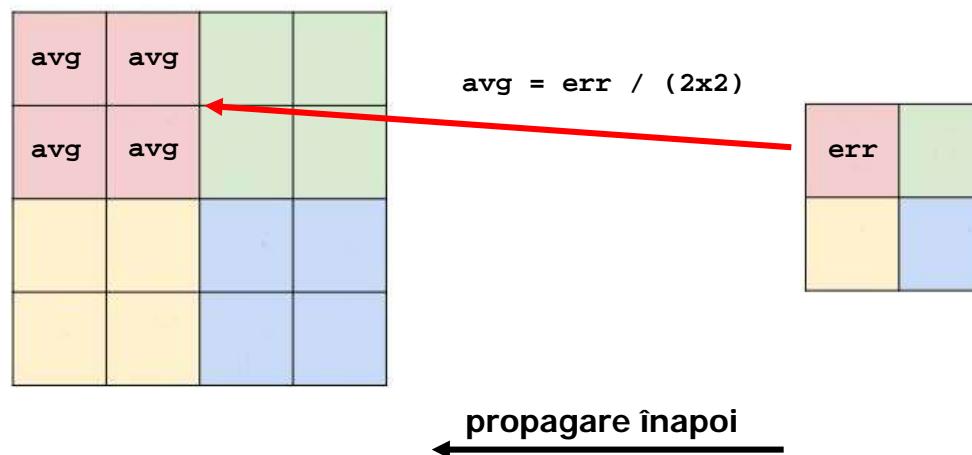
### ■ sum pooling

- fiecare element din blocul corespunzător contribuie în mod egal la eroare
- toate elementele vor primi valoarea erorii



## ■ Antrenarea CNN

- average pooling
  - similar cu sum pooling
  - fiecare element contribuie în mod egal la eroare
  - contribuția elementelor este ponderată, cu ponderile  $1/(m*n)$ , unde  $m, n$  sunt dimensiunile blocului



## ■ Antrenarea CNN

- dacă se furnizează la intrarea rețelei aceeași imagine din nou:
  - vectorul de probabilități obținut la ieșirea rețelei ar trebui să fie mai apropiat de cel furnizat inițial (altfel spus, eroarea ar trebui să se diminueze)
- există o serie de parametri care nu se modifică pe parcursul antrenării
  - numărul de filtre, dimensiunile lor, arhitectura rețelei etc.

## ■ Antrenarea CNN

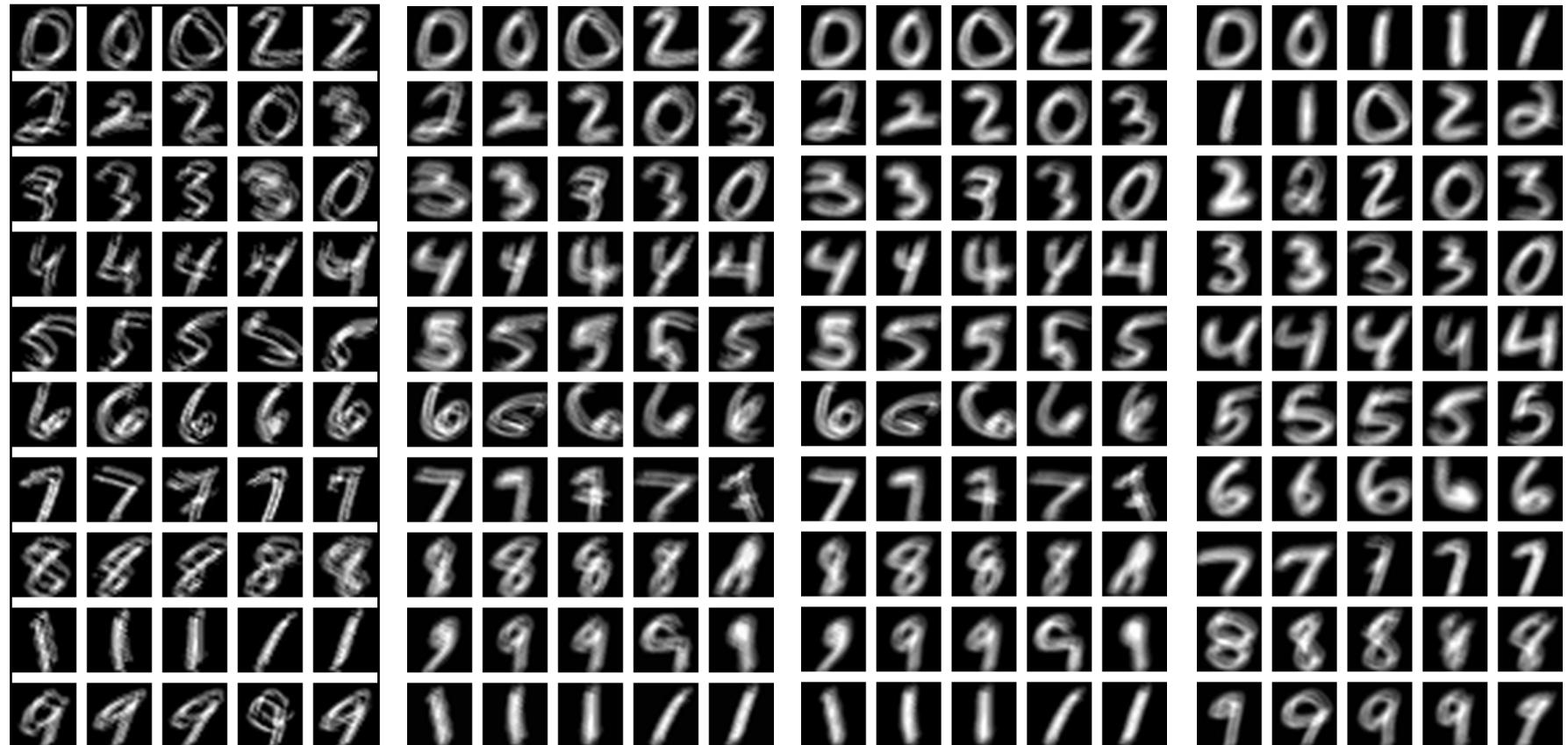
- se repetă pașii anteriori pentru fiecare imagine din setul de date de antrenare
  - în urma acestui proces, ponderile și filtrele CNN se vor fi ajustat astfel încât să clasifice corect imaginile de antrenare
  - scopul este ca, fiind dată o nouă imagine
    - aceasta să se propage înainte prin rețea
    - probabilitățile obținute la ieșire să fie în concordanță cu clasa în care se încadrează imaginea
- Întregul set de date se propagă prin rețea de mai multe ori, fiecare iterație se numește “epocă”

- Rezultate:
  - diverse feature maps obținute pe parcursul antrenării, din **primul** strat de conoluție



## ■ Rezultate:

- câteva feature maps obținute cu primul filtru din **primul strat de conoluție** (slide-ul următor...)
  - în total sunt câte 10000 de feature maps pentru fiecare filtru și fiecare epocă



Epoca 1

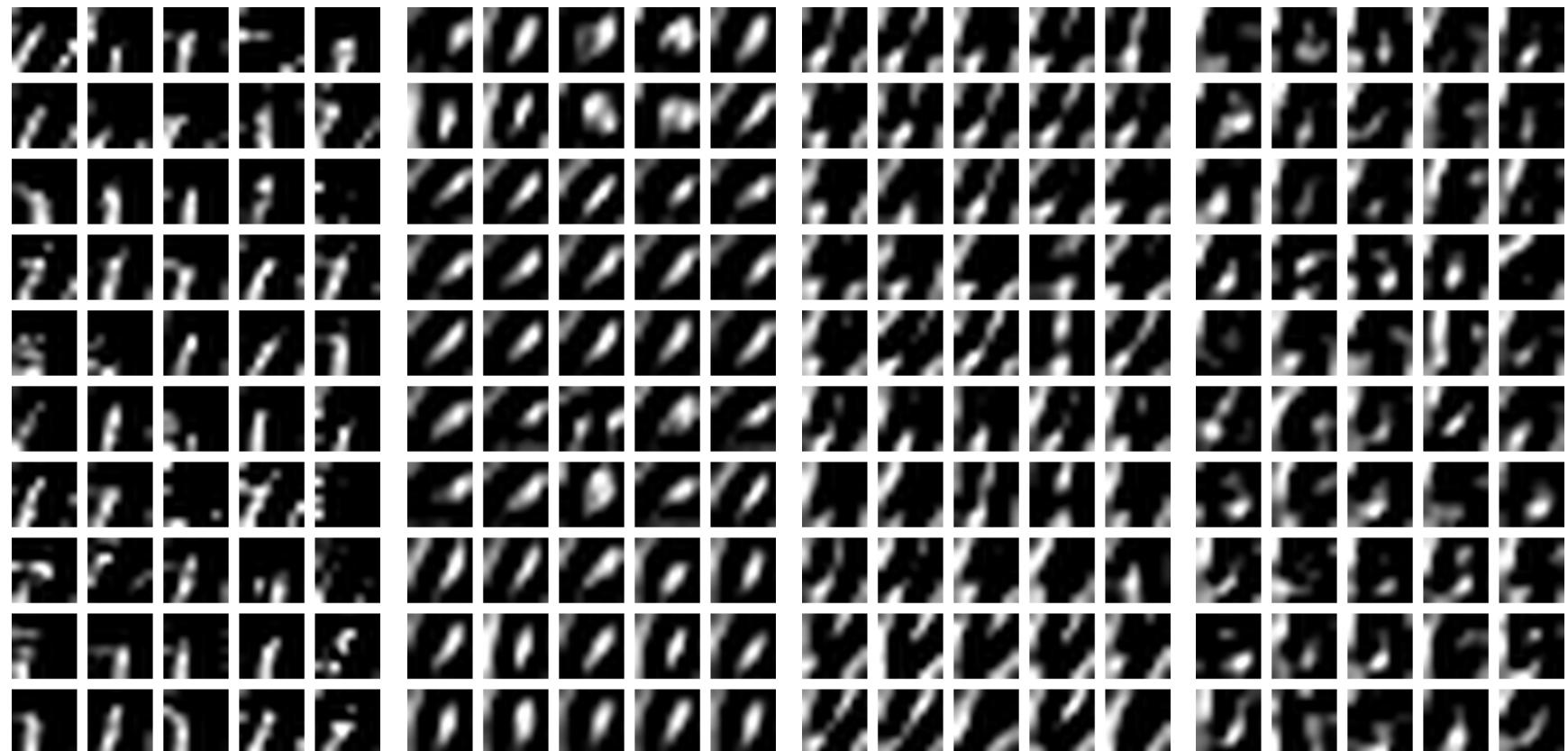
Epoca 10

Epoca 50

Epoca 100

## ■ Rezultate:

- câteva feature maps obținute cu primul filtru din **al doilea** strat de convoluție (slide-ul următor...)
  - în total sunt câte 10000 de feature maps pentru fiecare filtru și fiecare epocă



Epoca 1

Epoca 10

Epoca 50

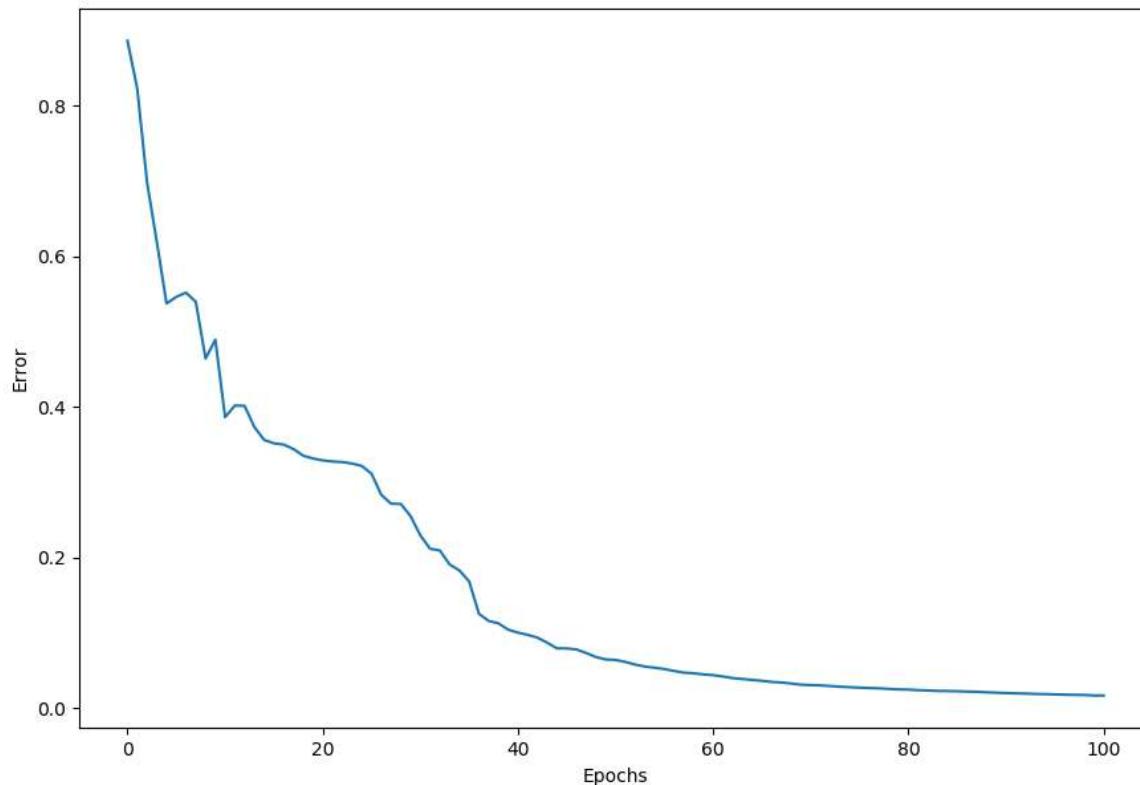
Epoca 100

## ■ Evaluarea

- Verificarea acurateței rețelei
- De obicei se folosesc date de test
  - similară celor de antrenare, clasele lor sunt deja cunoscute
- Se pot utiliza înseși datele de antrenare

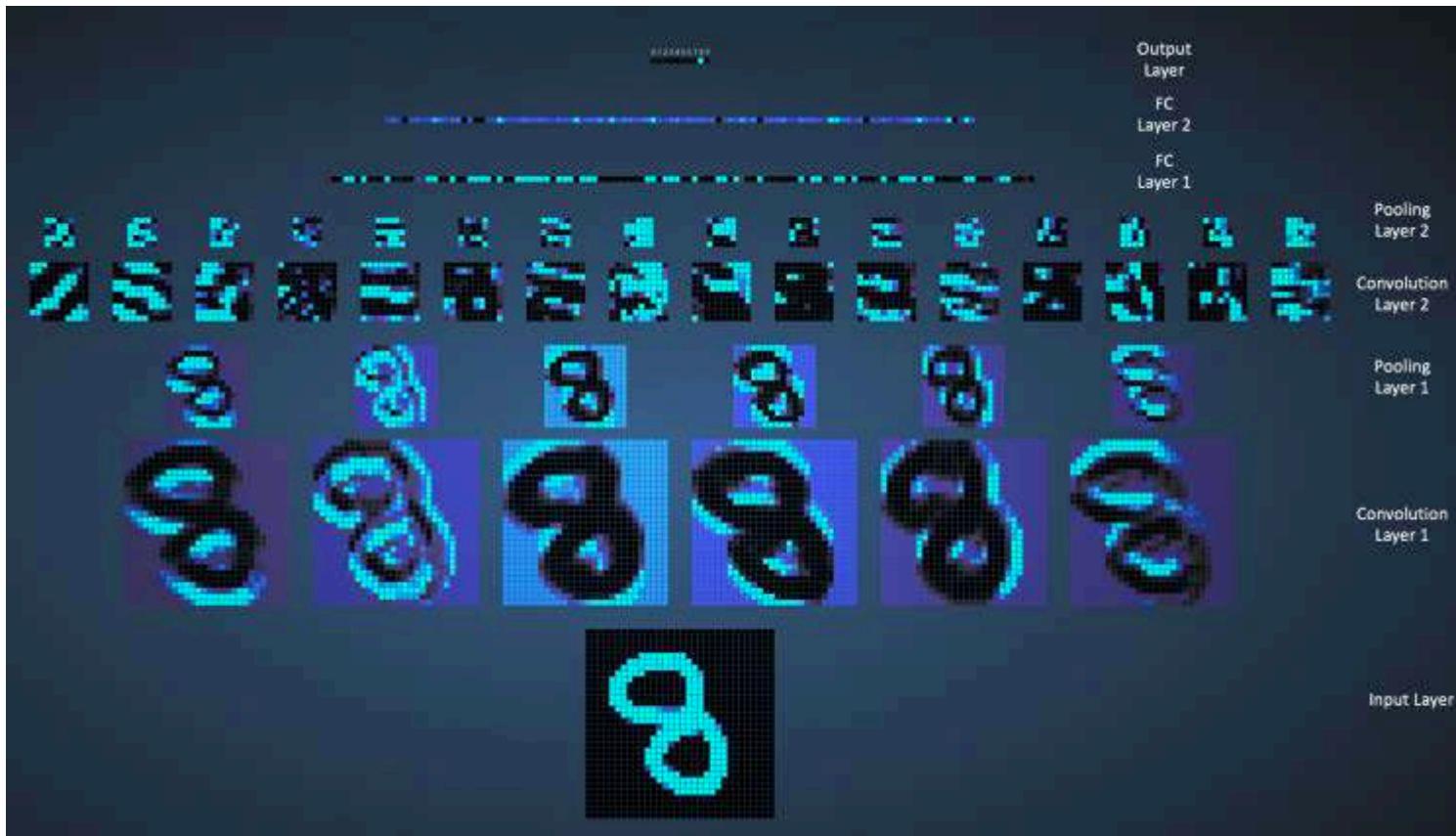
eroarea de clasificare = nr imagini clasificate gresit / nr total de imagini

## ■ Evaluarea:

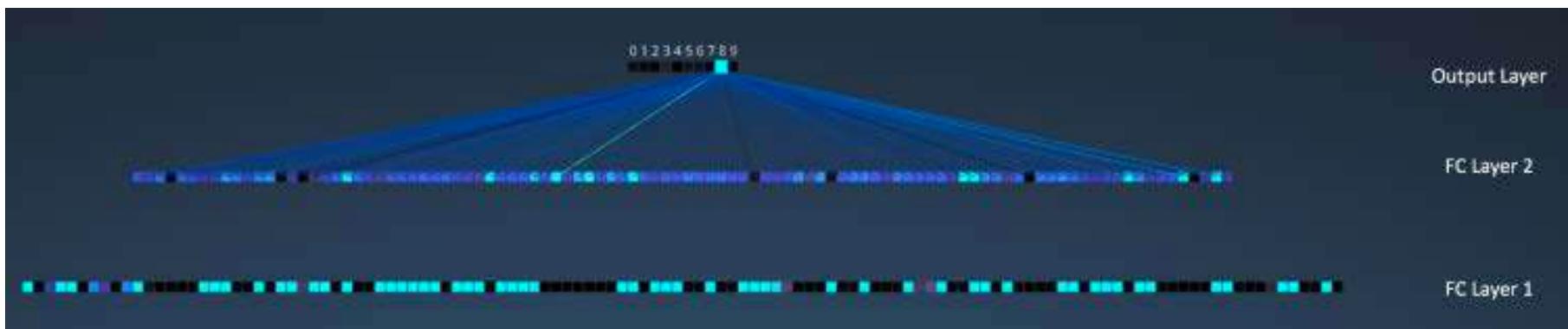


Eroarea finală după 100 epoci: 0.032

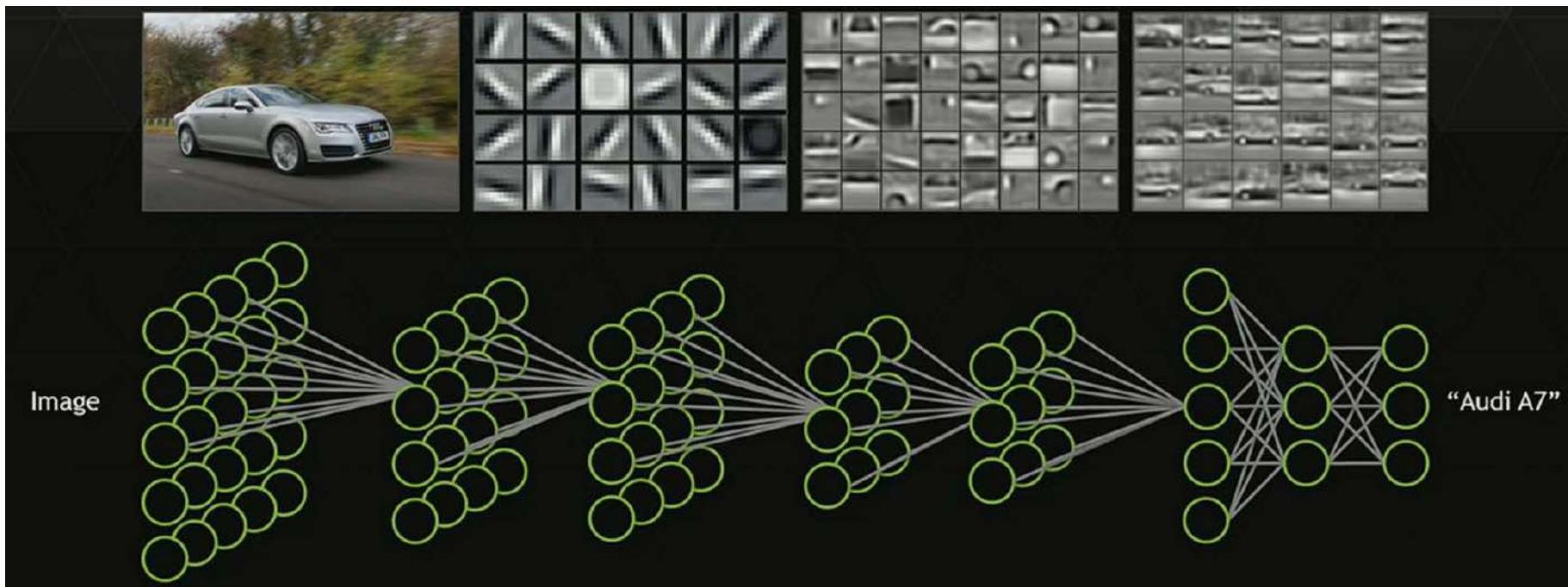
## ■ Vizualizarea CNN – alte exemple



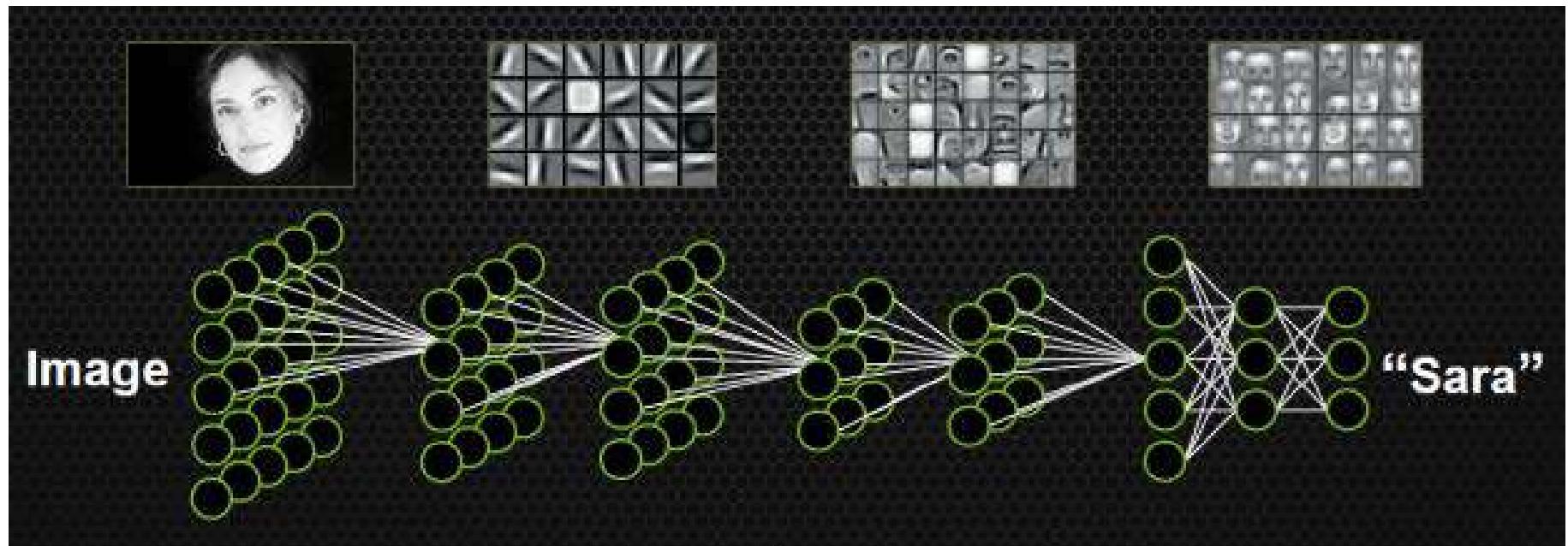
- Vizualizarea CNN - exemple



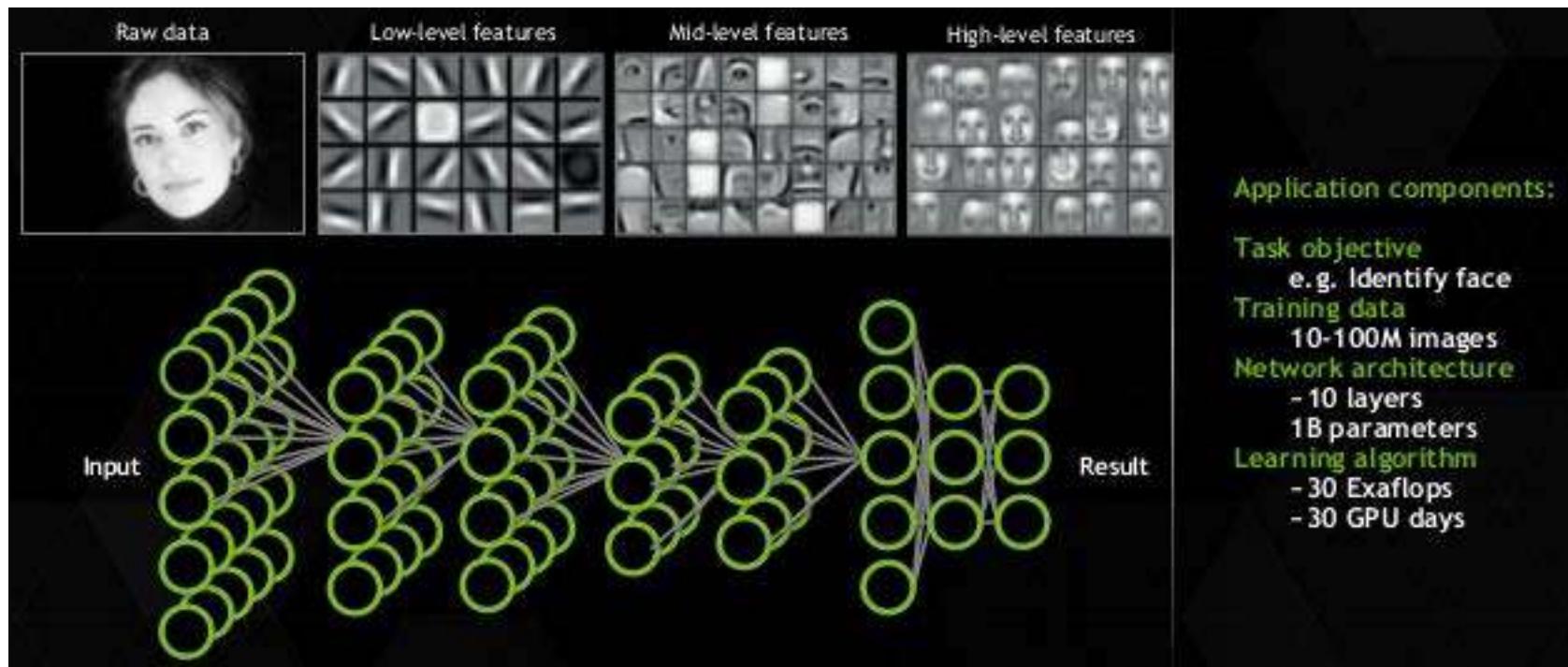
## ■ Vizualizarea CNN – alte exemple



## ■ Vizualizarea CNN – alte exemple



## ■ Vizualizarea CNN – alte exemple



# Învățare automată

## 10. Sisteme de recomandare

**Marius Gavrilescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

# Sisteme de recomandare

- **Abreviat RS (*Recommender System*)**
- **RS sunt algoritmi care analizează interesele și preferințele utilizatorilor și fac recomandări de produse, servicii etc.**
- **Susținerea și îmbunătățirea calității deciziilor pe care le fac utilizatorii care caută și selectează produse online.**
- **Scop – de a face asocieri între multimi de utilizatori și multimi de obiecte**
  - **filtrarea informației – recomandarea anumitor obiecte**
  - **oferingea de suport pentru luarea de decizii**

# Sisteme de recomandare

- **Realizarea de predicții**
  - în ce măsură un produs solicită interesul unui utilizator
  - în ce măsură un caz de utilizare/scenariu de test este util într-o anumită situație sau pentru un anumit studiu
- **Îmbunătățirea interacțiunii**
  - sporirea disponibilității și gradului de încredere ale unui utilizator
  - informarea utilizatorilor cu privire la domeniul/gama unor produse
  - convingerea și îndrumarea utilizatorilor către anumite produse

# Sisteme de recomandare

**RS = funcție ce presupune:**

- **date de intrare**
  - specificul utilizatorului (ratinguri, preferințe, locație, etc.)
  - caracteristicile obiectelor (tip, calitate, popularitate etc.)
- **rezultate la ieșire**
  - scoruri ce indică relevanța obiectelor pentru utilizatori
  - date utile pentru ranking

# Sisteme de recomandare

## Tipuri de RS

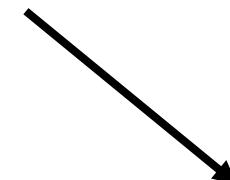
- **personalizate**
- **colaborative**
- **bazate pe conținut**
- **bazate pe cunoștere**
- **hibride**

# Sisteme de recomandare

- RS personalizat – face recomandări pe baza caracteristicilor utilizatorului



Profilul utilizatorului,  
contextul în care se află



RS

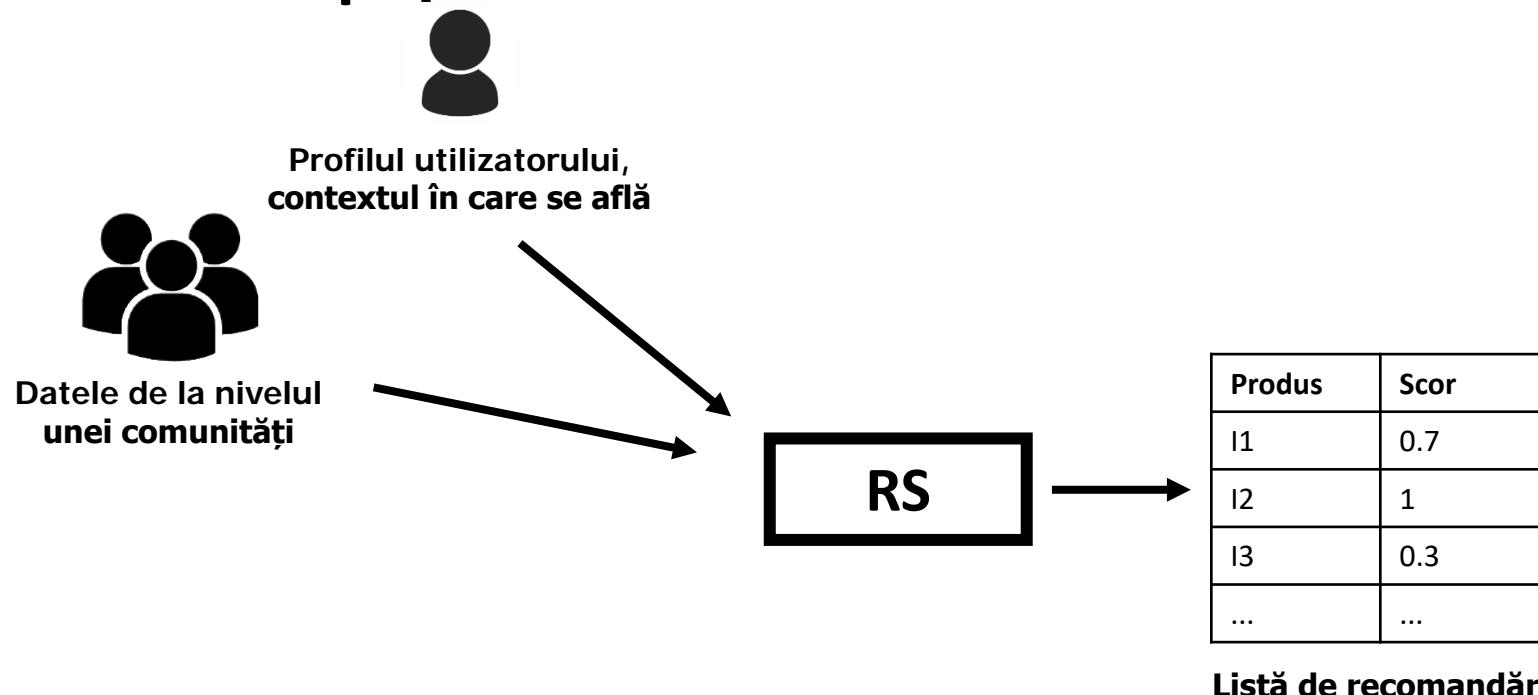


Produs	Scor
I1	0.7
I2	1
I3	0.3
...	...

Listă de recomandări

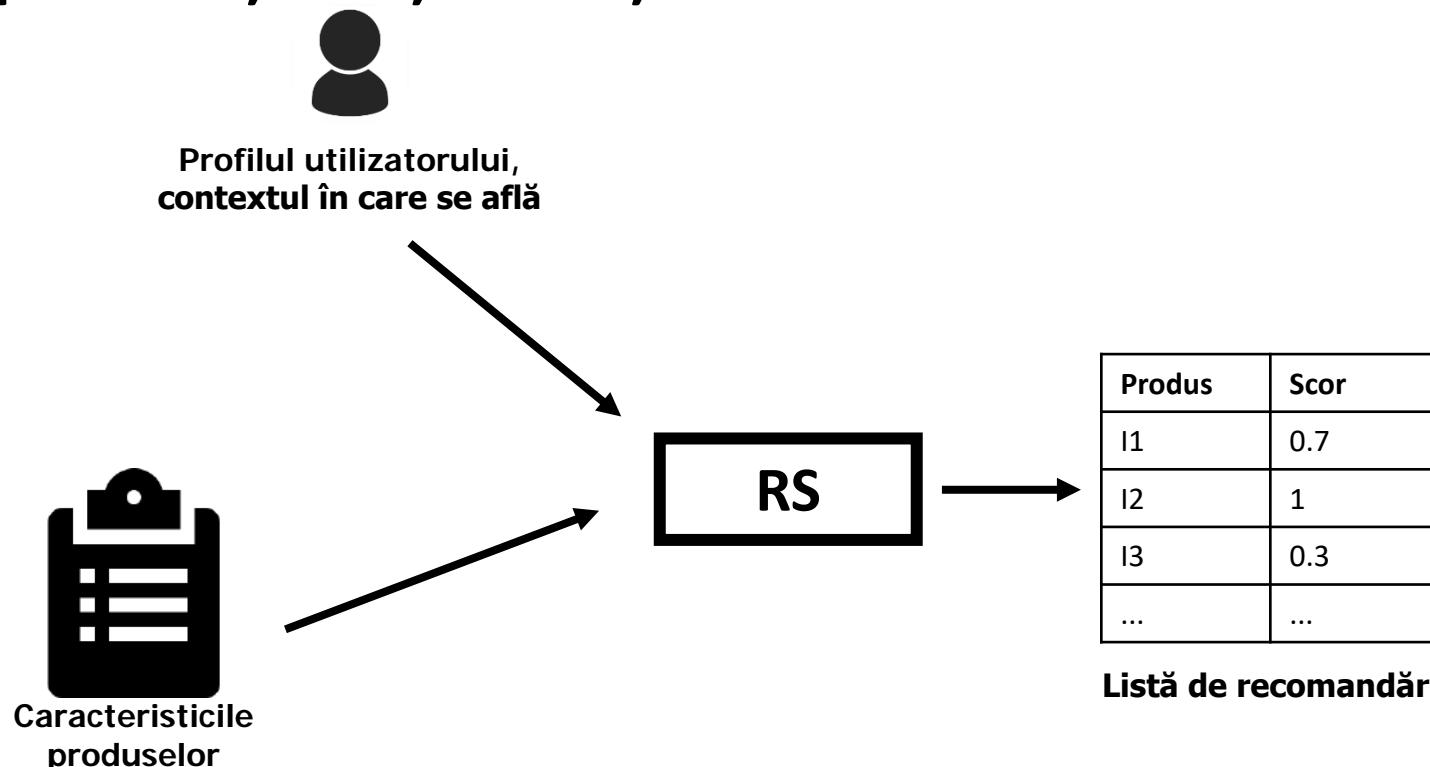
# Sisteme de recomandare

- RS colaborativ – face recomandări pe baza a ceea ce este popular într-o comunitate



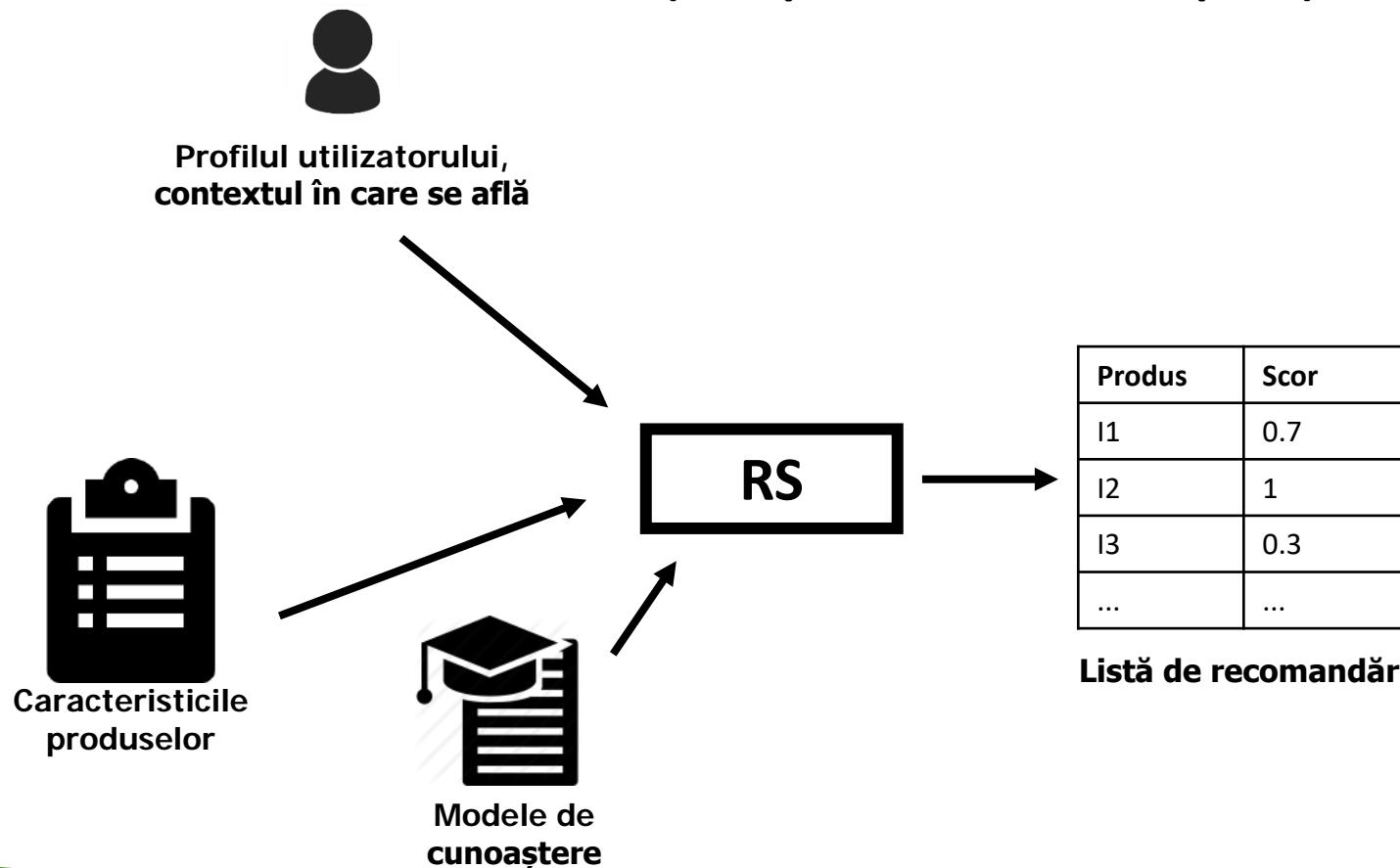
# Sisteme de recomandare

- RS bazat pe conținut – face recomandări pe baza preferințelor și selecțiilor anterioare ale utilizatorului



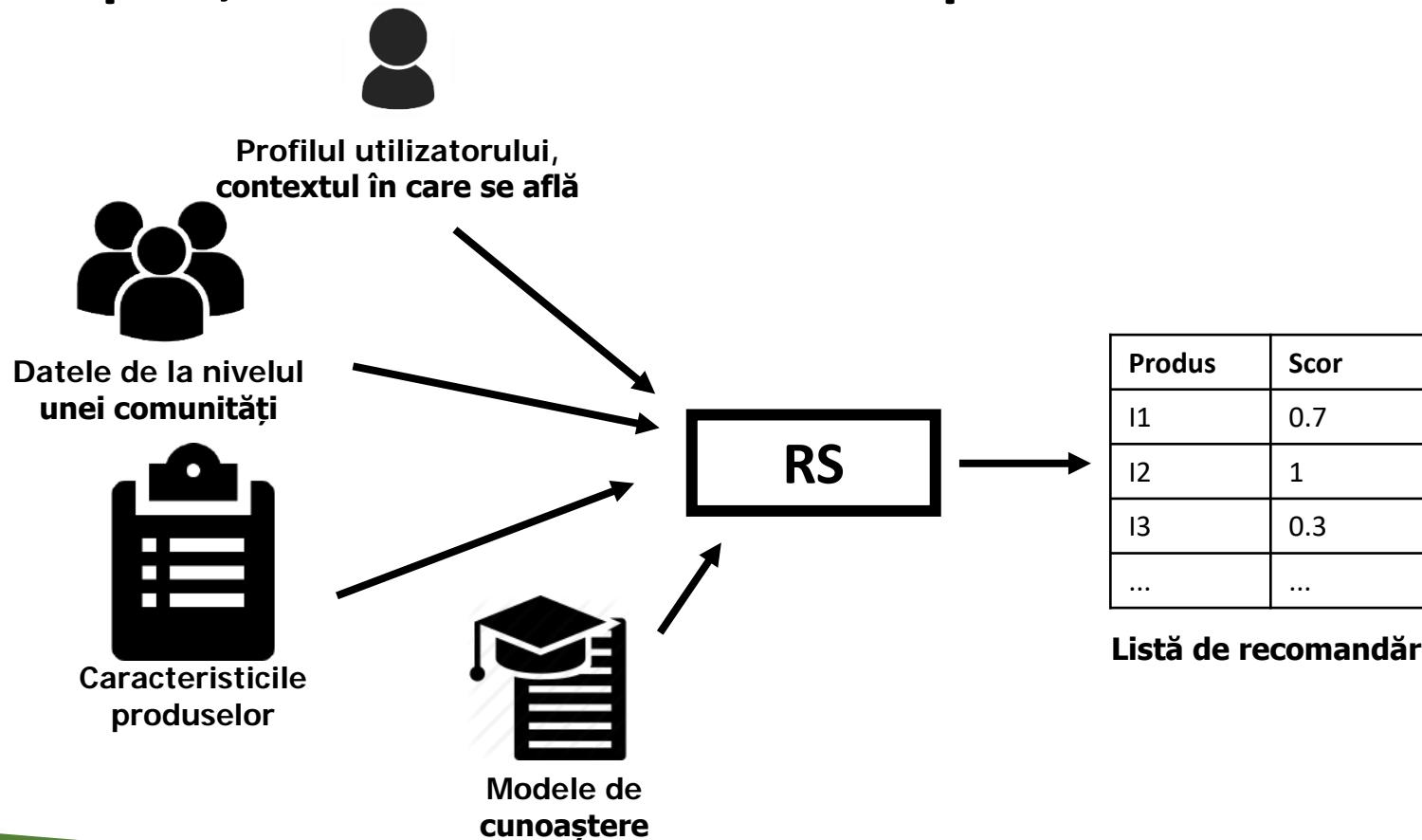
# Sisteme de recomandare

- RS bazat pe cunoștere – face recomandări pe baza nevoilor utilizatorului (care poate fi întrebat explicit)



# Sisteme de recomandare

- RS hibrid – combinații de date de intrare de diverse tipuri, diverse metode de compunere a lor



# RS Colaborativ

- RS Colaborativ = modelul de RS utilizat cel mai frecvent
- Produsele se recomandă funcție de popularitatea lor
- Principii:
  - utilizatorii oferă rating-uri pentru diverse produse
  - utilizatorii care au avut preferințe similare în trecut vor avea preferințe similare și în viitor

# RS Colaborativ

- **Date de intrare**
  - matrice de rating-uri (conține rating-ul dat de anumiți utilizatori pentru anumite produse)
- **Date de ieșire**
  - predicții în format numeric ce indică satisfacția sau lipsa de satisfacție a unui utilizator relativ la un produs
  - un top al produselor celor mai recomandate

# RS Colaborativ

- **fiind dat un utilizator A și un produs i necunoscut lui A**
  - **se identifică o multime de utilizatori care**
    - au avut în trecut preferințe similare cu cele ale lui A
    - care au dat rating-uri produsului i
  - **pe baza preferințelor utilizatorilor, se estimează rating-ul pe care l-ar da A obiectului i**
    - se estimează câte un rating pentru toate produsele
    - se recomandă lui A primele n produse cu cele mai mari rating-uri
- **se face următoarea presupunere: preferințele utilizatorilor sunt stabile și consecvente în timp**
  - un utilizator va avea în viitor preferințe similare cu cele din trecut

# RS Colaborativ

- matricea rating-urilor = matrice rară (*engl. sparse matrix*)
  - linii – o serie de utilizatori
  - coloane – o serie de produse (*items*)
  - valori – rating-urile unor utilizatori pentru unele produse
  - majoritatea valorilor lipsesc
    - doar unii utilizatori au dat rating unor produse

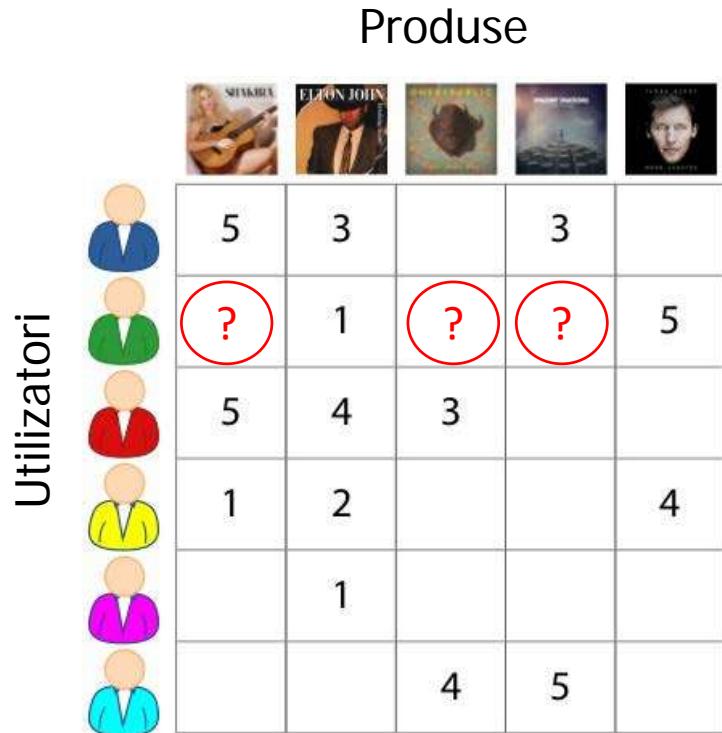


5	3		3	
	1			5
5	4	3		
1	2			4
	1			
		4	5	

# RS Colaborativ

**Scopul RS – realizarea de recomandări:**

- estimarea valorilor lipsă din matricea rating-urilor
- identificarea produselor preferențiale pe baza estimărilor
- ex: care produs i s-ar potrivi celui de-al doilea utilizator?
  - altfel spus: care ar fi produsele, dintre cele pe care al doilea utilizator nu le-a întâlnit încă, pentru care acesta ar da rating-uri mari?
  - pe baza acestui rezultat i se pot face recomandări utilizatorului



# RS Colaborativ

- Exemplu: o matrice de rating-uri ce conține rating-urile date de diversi utilizatori unor obiecte (scoruri din [1, 5])
- scopul este de a prezice preferința pentru Item5 a utilizatorului Alice, care încă nu a întâlnit acest obiect
  - pentru a răspunde la întrebarea: i se poate recomanda produsul Item5 lui Alice?

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

# RS Colaborativ

- **Colaborare bazată pe similaritatea utilizatorilor:**
  - rating-ul unui utilizator se prezice pe baza asemănării cu ratingurile celorlalți
  - se determină similaritatea dintre Alice și ceilalți utilizatori, pe baza rating-urilor disponibile
  - se pot lua în considerare o parte din utilizatori, sau toți, ponderat

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

# RS Colaborativ

- **Calculul similarității:**
  - **Corelația Pearson**
    - **$a, b$  – doi utilizatori**
    - **$r_{a,p}$  – rating-ul dat de utilizatorul  $a$  obiectului  $p$**
    - **$\bar{r}_a$  = rating-ul mediu dat de utilizatorul  $a$**
    - **$P$  – mulțimea de obiecte care au primit rating-uri de la  $a$  și  $b$**

$$sim(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

# RS Colaborativ

- Calculul similarității:
  - Corelația Pearson
  - Reprezintă raportul dintre:
    - covarianța rating-urilor celor doi utilizatori
      - covarianță pozitivă – când rating-urile lui *a* cresc/scad, rating-urile lui *b* tind să crească/scadă
      - covarianță negativă – când rating-urile lui *a* cresc/scad, rating-urile lui *b* tind să scadă/crească
      - covarianță nulă – nu există o relație predictibilă între rating-urile date de cei doi utilizatori
    - produsul varianțelor ratingurilor fiecărui utilizator
      - varianță - abaterea rating-urilor unui utilizator față de rating-ul său mediu

$$sim(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

# RS Colaborativ

- similaritate calculată folosind corelația Pearson
- pentru a prezice rating-ul lui Alice pentru Item5 se poate folosi:

$$pred(a, p) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} sim(a, b)}$$

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1



sim = 0.85  
sim = 0.00  
sim = 0.70  
sim = -0.79

# RS Colaborativ

estimarea rating-ului pe care l-ar da utilizatorul a obiectului p

rating-ul mediu al utilizatorului a

suma abaterilor, ponderate de similaritatea dintre utilizatori

deviația rating-urilor celorlalți utilizatori de la rating-ul lor mediu

$$pred(a, p) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} sim(a, b)}$$

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0.85  
sim = 0.00  
sim = 0.70  
sim = -0.79

# RS Colaborativ

- **Colaborare bazată pe similaritatea obiectelor:**
  - se determină similaritatea dintre obiecte, pe baza ratingurilor utilizatorilor
  - se determină elementele similare cu Item5
  - se folosesc rating-urile elementelor celor mai similare pentru a prezice rating-ul lui Alice pentru Item5

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

# RS Colaborativ

- Colaborare bazată pe similaritatea obiectelor:

- similaritatea cosinus:

- obiectele sunt tratate ca vectori

- valorile fiecărui vector = ratingurile utilizatorilor

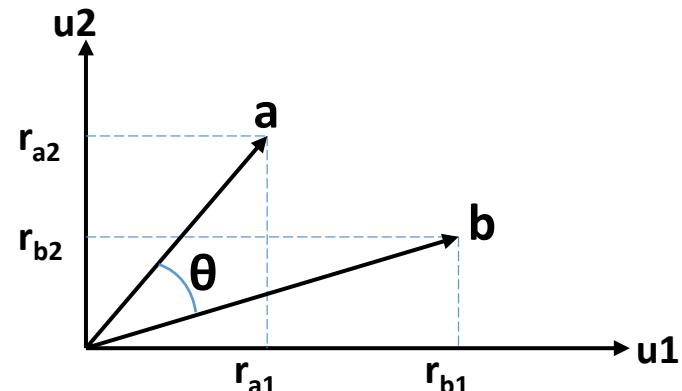
- exemplu: obiectele **a**, **b** au primit rating-urile  $r_{a1}$ ,  $r_{a2}$ ,  $r_{b1}$ ,  $r_{b2}$  de la userii

**u1, u2**

$$\vec{a} = [r_{a1} \ r_{a2}] \quad \vec{b} = [r_{b1} \ r_{b2}]$$

- similaritatea = cosinusul unghiului format de cei doi vectori

$$sim(\vec{a}, \vec{b}) = \cos(\theta)$$



# RS Colaborativ

- Colaborare bazată pe similaritatea obiectelor:

- similaritatea cosinus:

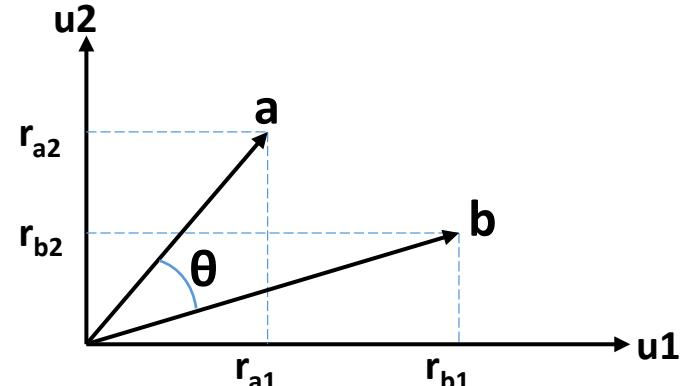
$$sim(\vec{a}, \vec{b}) = \cos(\theta)$$

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}$$

$\vec{a} \cdot \vec{b}$  = produs scalar

$$\vec{a} \cdot \vec{b} = r_{a1} * r_{b1} + r_{a2} * r_{b2}$$

$$|\vec{a}| = \sqrt{r_{a1}^2 + r_{a2}^2}$$



# RS Colaborativ

- **Colaborare bazată pe similaritatea obiectelor:**
  - **similaritatea cosinus:**
    - **în caz general: ratingurile unui obiect formează un vector n-dimensional (n = numărul utilizatorilor care au dat rating obiectului)**
$$\vec{a} = [r_{a1} \ r_{a2} \ r_{a3} \dots \ r_{an}]$$
    - **pentru calculul similarității a două obiecte: n = numărul utilizatorilor care au dat rating ambelor obiecte**

# RS Colaborativ

- **Colaborare bazată pe similaritatea obiectelor:**
  - pentru predicția ratingului obiectului vizat
    - se determină media ponderată a ratingurilor date de utilizator pentru celealte obiecte
    - ponderile = similaritățile dintre obiectul vizat și celealte obiecte

$$pred(u, p) = \frac{\sum_{i \in ratedItem(u)} sim(i, p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i, p)}$$

# RS Colaborativ

- Colaborare bazată pe similaritatea obiectelor:

estimarea rating-ului pe care l-ar da utilizatorul u obiectului p

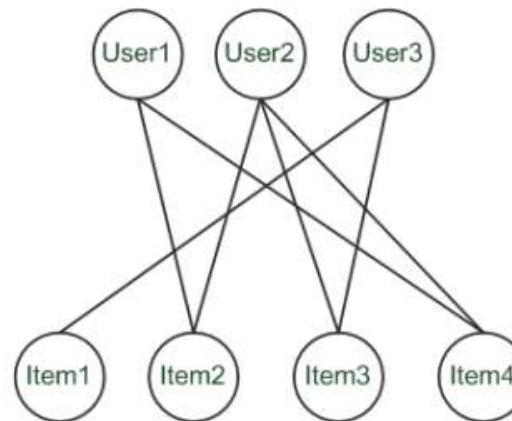
suma rating-urilor date de utilizatorul u celoralte obiecte i, ponderată de similaritățile dintre obiecte

$$pred(u, p) = \frac{\sum_{i \in ratedItem(u)} sim(i, p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i, p)}$$

# RS Colaborativ

## Metode bazate pe grafuri

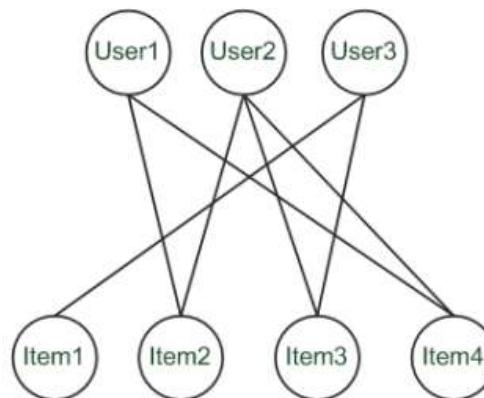
- exploatează caracterul tranzitiv al preferințelor utilizatorilor
- **graful conține**
  - noduri utilizator
  - noduri obiect
  - muchii între nodurile utilizator și cele obiect
    - muchie între  $User_i$  și  $Item_j$  =  $User_i$  a ales  $Item_j$  (sau  $Item_j$  se află printre preferințele lui  $User_i$ )



# RS Colaborativ

## Metode bazate pe grafuri

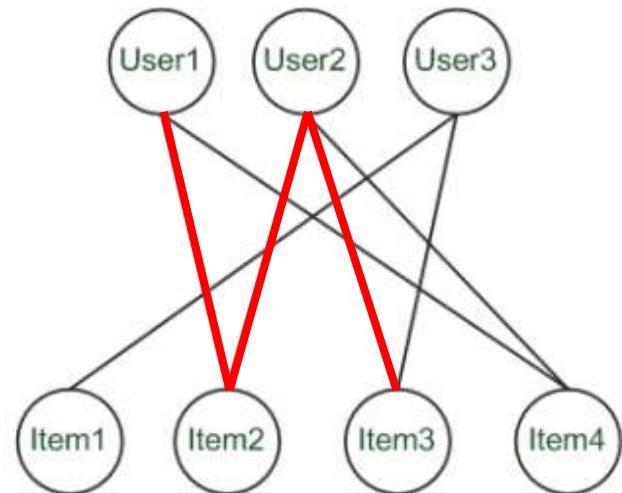
- exemplu de raționament:
  - presupunem că dorim să facem o recomandare lui User1
    - unul dintre Item-urile pe care nu le cunoaște (pentru care nu există muchie către User1)
  - se observă că User1 și User2 preferă ambii Item2 și Item4
  - User2 preferă și Item3
  - prin urmare, i se recomandă Item3 și lui User1



# RS Colaborativ

## Metode bazate pe grafuri

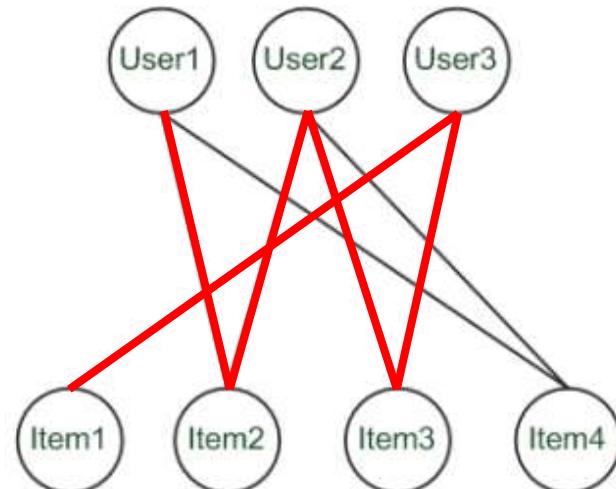
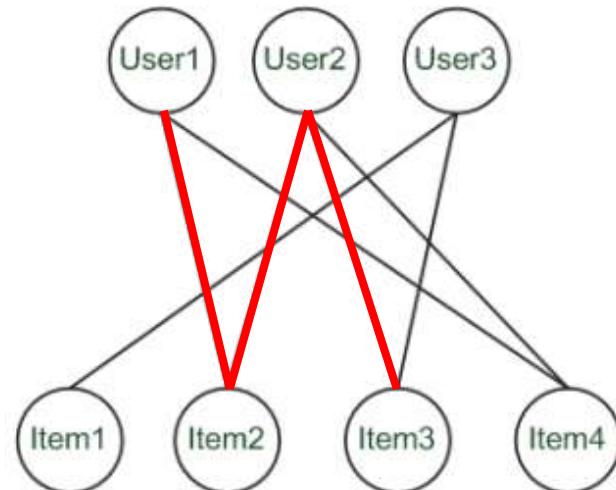
- lui User1 i se recomandă Item3 deoarece există un drum minimal prin graf între User1 și Item3
- lungimea maximă a drumului se stabilește explicit



# RS Colaborativ

## Metode bazate pe grafuri

- dacă  $drum_{maxim} = 3$ , lui User1 i se poate recomanda Item3
- dacă  $drum_{maxim} = 5$ , lui User1 i se pot recomanda Item1 și Item3



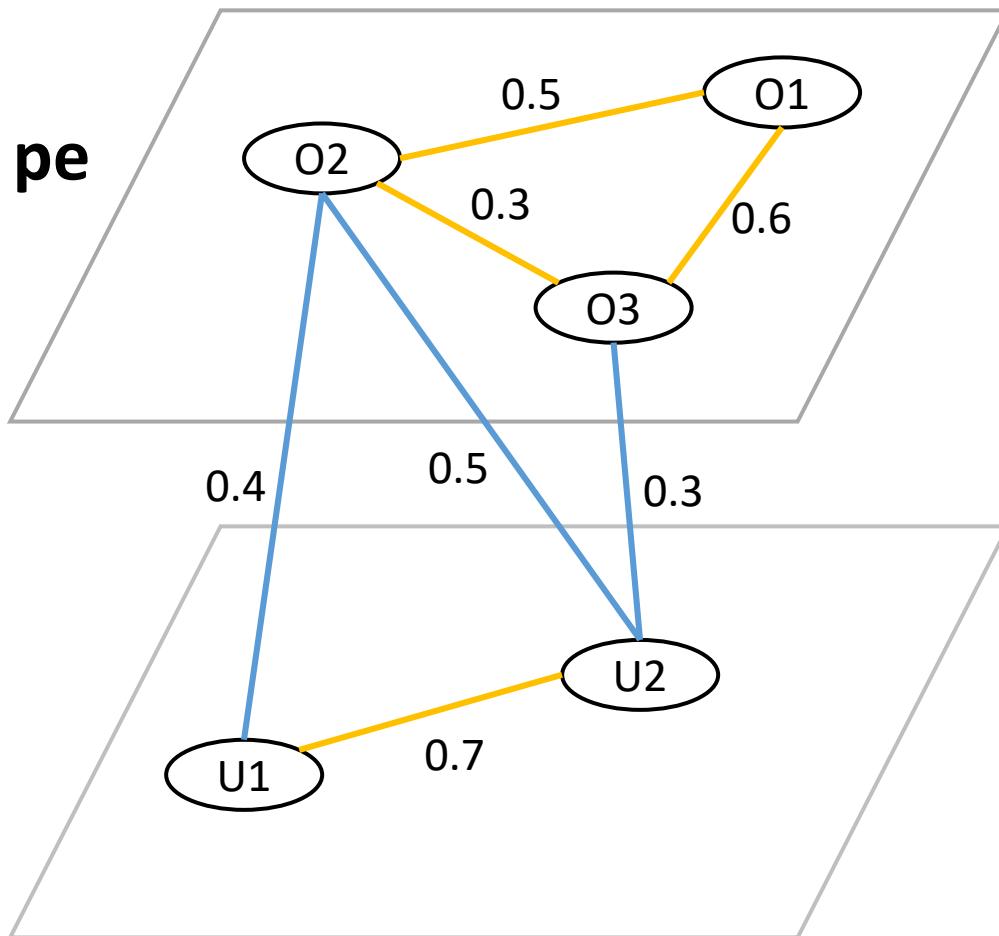
# RS Colaborativ

## Metode bazate pe grafuri

- **în caz general:**
  - muchiile conțin valori care indică nivelul de interacțiune dintre utilizatori și obiecte
  - pot exista muchii între obiecte
    - valorile acestor muchii = similaritatea dintre obiecte
  - pot exista muchii între utilizatori
    - valorile acestor muchii = similaritatea dintre utilizatori (care se poate determina pe baza profilurilor și comportamentelor lor)
- se determină asocierea dintre un utilizator și un obiect
  - asociere de grad n = se iau în calcul distanțele de maxim n muchii între două noduri

# RS Colaborativ

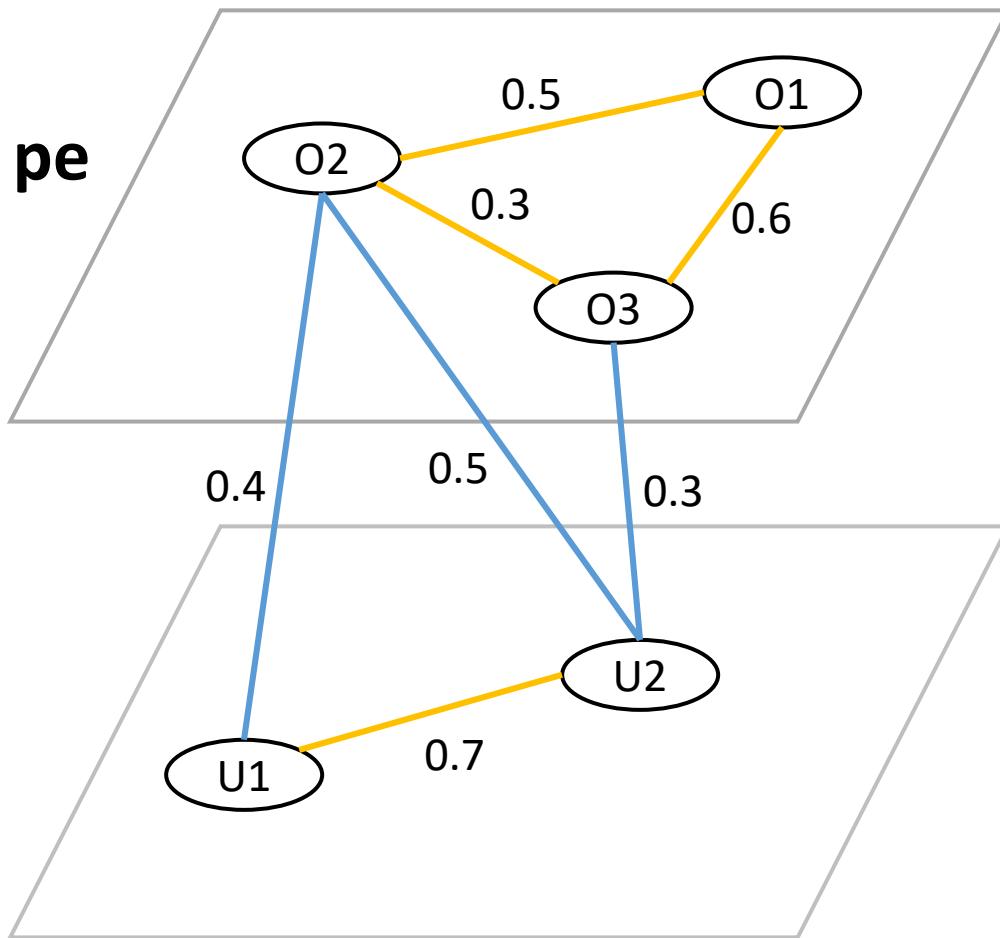
## Metode bazate pe grafuri



Asociere grad 1: U1-O1 = 0

# RS Colaborativ

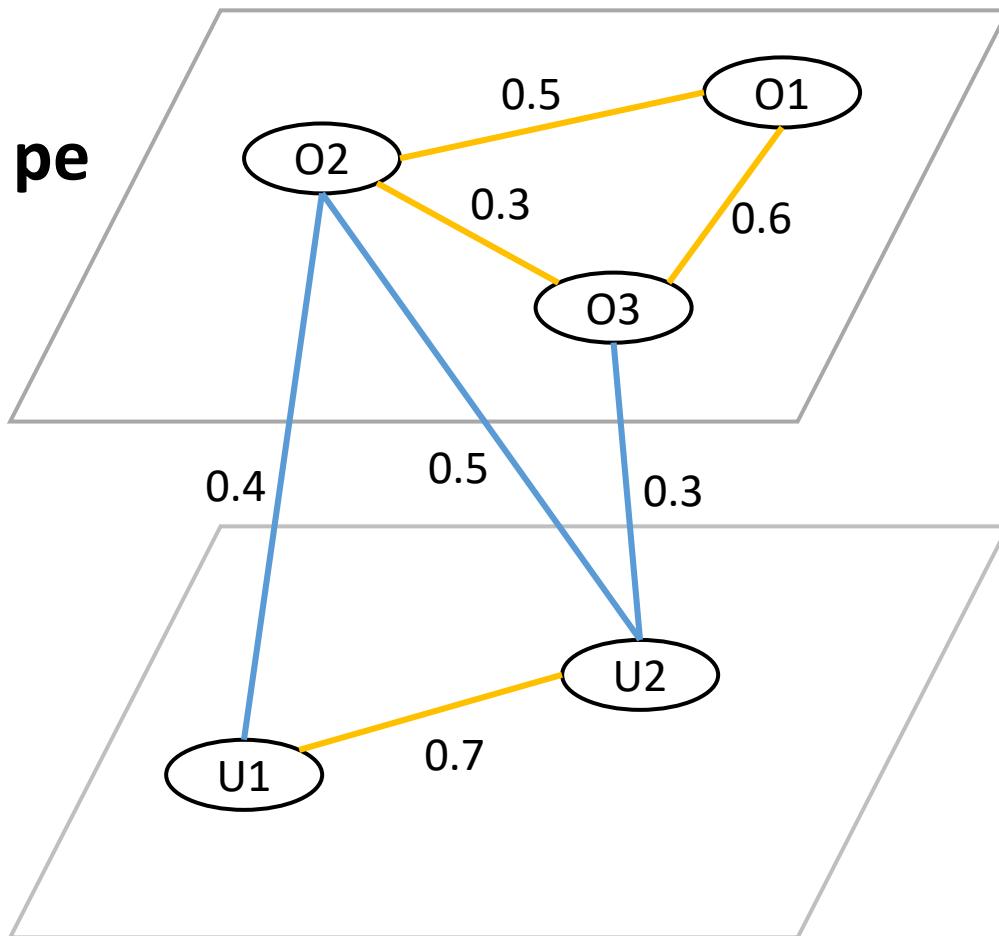
## Metode bazate pe grafuri



**Asociere grad 2:  $U1-O1 = 0.4 * 0.5 = 0.2$**   
**(U1-O2-O1)**

# RS Colaborativ

## Metode bazate pe grafuri



Asociere grad 3:  $U_1-O_1 = 0.4 \cdot 0.5 + 0.7 \cdot 0.5 \cdot 0.5 + 0.4 \cdot 0.3 \cdot 0.6 + 0.7 \cdot 0.3 \cdot 0.6 = 0.573$   
( $U_1-O_2-O_1$ ,  $U_1-U_2-O_2-O_1$ ,  $U_1-O_2-O_3-O_1$ ,  $U_1-U_2-O_3-O_1$ )

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- **factorizare = descompunerea matricei într-un produs de matrice de dimensiuni reduse**
- **Se folosește matricea ratingurilor R**
  - N useri
  - M obiecte
  - $R = \text{matrice } N \times M$
  - $R[n, m] = \text{rating-ul dat de userul } n \text{ obiectului } m$ 
    - $n = [1..N], m = [1..M]$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- descompunerea matricei rating-urilor:

$$\boxed{R_{N \times M}} = \boxed{U_{N \times K}} \times \boxed{V_{K \times M}}$$

- $U_{N \times K}$  = matricea utilizatorilor (*user matrix*)
- $V_{K \times M}$  = matricea obiectelor (*item matrix*)
- $K$  = numărul trăsăturilor (features) care rezultă
  - pentru fiecare utilizator în matricea  $U$
  - pentru fiecare obiect în matricea  $V$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- matricea rating-urilor este rară, i.e. majoritatea elementelor sale sunt nedefinite (nu toți utilizatorii reușesc să dea rating-uri tuturor obiectelor)
- prin factorizarea matricii se pot face predicții
  - se pot estima valorile elementelor nedefinite
  - adică se pot estima rating-urile utilizatorilor pentru obiecte pe care aceștia încă nu le-au întâlnit
  - pe baza rating-urilor estimate se pot face recomandări

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- Exemplu
  - matricea ratingurilor:

	O1	O2
U1	2.7	3.3
U2	6.3	???

- utilizatorul U2 nu a întâlnit încă obiectul O2
- trebuie stabilit dacă lui U2 să i se recomande O2
  - se estimează rating-ul pe care l-ar da U2 obiectului O2

# RS Colaborativ

# Metode bazate pe factorizarea matricelor

- Exemplu
    - dacă matricea ratingurilor se descompune astfel:

$$\begin{array}{c}
 \begin{array}{cc} O_1 & O_2 \end{array} \\
 \begin{array}{c} U_1 \\ U_2 \end{array} \quad \left[ \begin{array}{cc} 2.7 & 3.3 \\ 6.3 & \textcolor{red}{???} \end{array} \right] = \left[ \begin{array}{c} 1.5 \\ 3.5 \end{array} \right] \times \left[ \begin{array}{cc} 1.8 & 2.2 \end{array} \right]
 \end{array}$$

Matricea  
 utilizatorilor, 2x1      Matricea  
 obiectelor, 1x2

- atunci estimarea ratingului dat de U2 obiectului O2 este:  
$$R(U2, O2) = 3.5 * 2.2 = 7.7$$
  - pe baza valorii se decide dacă O2 i se va recomanda lui U2

# RS Colaborativ

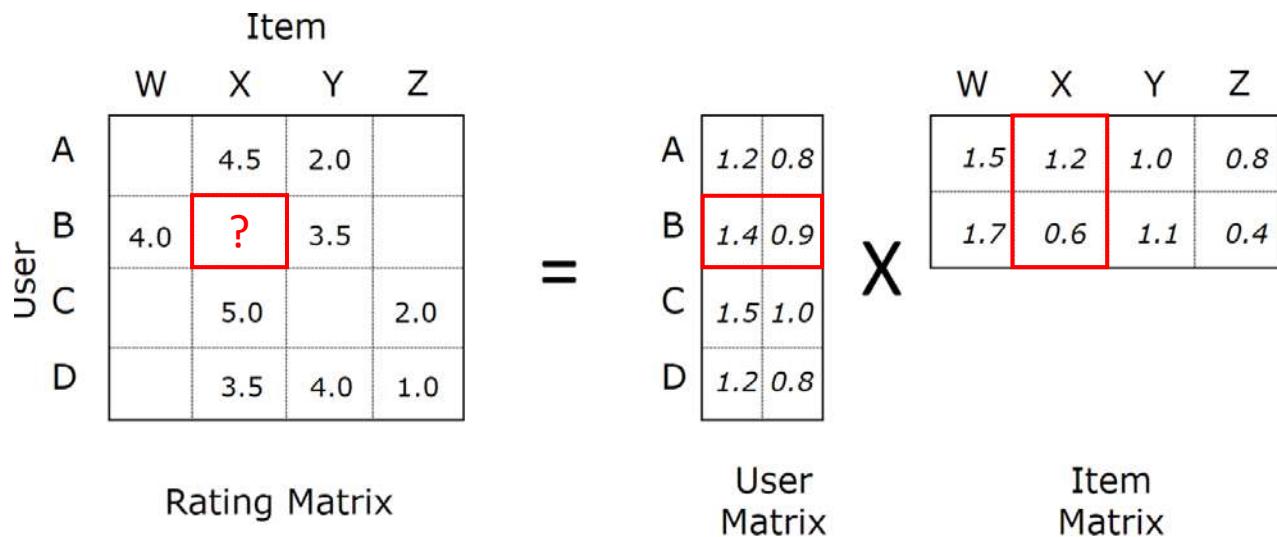
## Metode bazate pe factorizarea matricelor

$$\text{Rating Matrix} = \begin{matrix} & \text{User} \\ & \begin{matrix} A & B & C & D \end{matrix} \\ \text{Item} & \begin{matrix} W & X & Y & Z \end{matrix} \\ \begin{matrix} A & B & C & D \end{matrix} & \begin{matrix} 4.5 & 2.0 & & \\ 4.0 & & 3.5 & \\ 5.0 & & & 2.0 \\ 3.5 & 4.0 & 1.0 & \end{matrix} \end{matrix} = \begin{matrix} & \text{User} \\ & \begin{matrix} A & B & C & D \end{matrix} \\ \text{User} & \begin{matrix} 1.2 & 0.8 \\ 1.4 & 0.9 \\ 1.5 & 1.0 \\ 1.2 & 0.8 \end{matrix} \\ \text{Matrix} & \times \\ & \begin{matrix} W & X & Y & Z \end{matrix} \\ \text{Item} & \begin{matrix} 1.5 & 1.2 & 1.0 & 0.8 \\ 1.7 & 0.6 & 1.1 & 0.4 \end{matrix} \\ \text{Matrix} & \end{matrix}$$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- odată descompusă matricea rating-urilor se pot face predicții folosind cei doi termeni ai produsului rezultat în urma factorizării



Predicția rating-ului user-ului B pentru obiectul X este  $r_{B,X} = 1.4 * 1.2 + 0.9 * 0.6$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- Factorizare
  - În general se folosesc metode numerice
    - $R_{NM} = U_{NK} * V_{KM}$
    - se inițializează aleator elementele din U și V
    - se stabilește un obiectiv (de ex minimizarea unei funcții eroare)
    - se ajustează din aproape în aproape (pe parcursul mai multor iterații) valorile din U și V a.î. să se îndeplinească obiectivul

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- eroarea: diferența dintre
  - elementele din matricea ratingurilor
  - elementele care rezultă din produsul matricelor user și item

$$R_{AX} \quad \text{vs} \quad U_{A1} * V_{X1} + U_{A2} * V_{X2}$$

=

		Item			
		W	X	Y	Z
User	A		4.5	2.0	
	B	4.0		3.5	
	C		5.0		2.0
	D		3.5	4.0	1.0

Rating Matrix

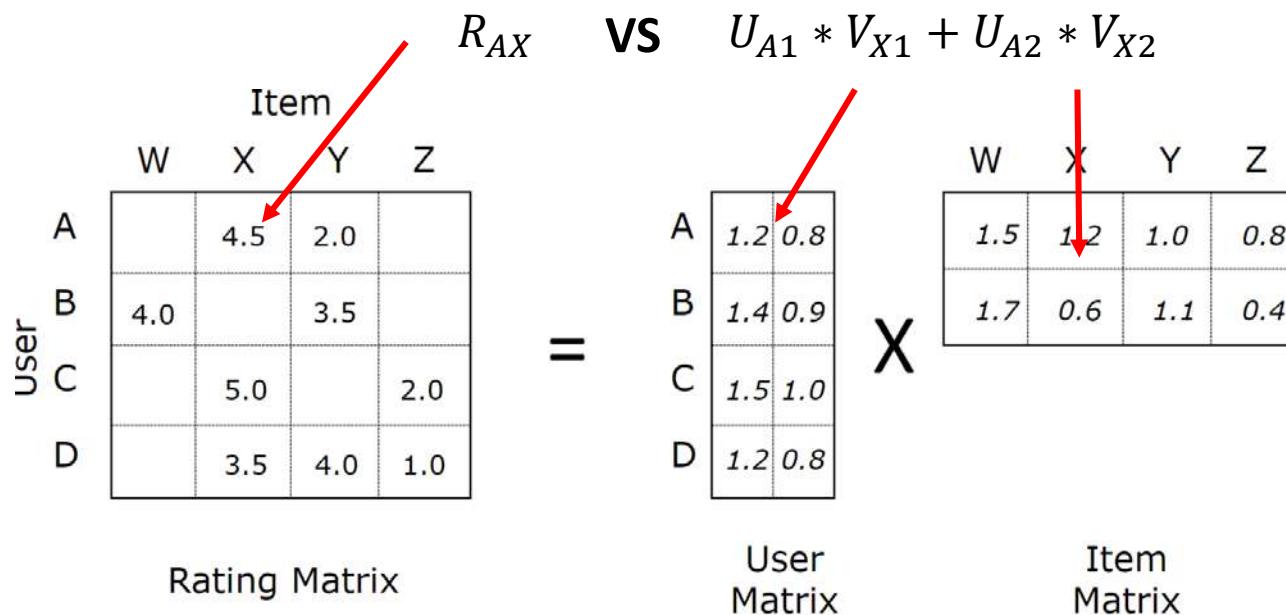
User		
A	1.2	0.8
B	1.4	0.9
C	1.5	1.0
D	1.2	0.8

User Matrix

Item				
W	1.5	1.2	1.0	0.8
X	1.7	0.6	1.1	0.4
Y				
Z				

Item Matrix



# RS Colaborativ

## Metode bazate pe factorizarea matricelor

$$\boxed{R_{NxM}} = \boxed{U_{NxK}} \times \boxed{V_{KxM}}$$

Scopul metodei: Determinarea elementelor  $u_{nk}$  și  $v_{km}$  astfel încât să se minimizeze diferența dintre

- valorile cunoscute din  $R_{nm}$
- ceea ce rezultă în aceleași poziții n,m din produsul  $\sum_k (u_{nk} * v_{km})$ 
  - $n = [1..N]$  ,  $m = [1..M]$  ,  $k = [1..K]$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

$$\boxed{R_{N \times M}} = \boxed{U_{N \times K}} \times \boxed{V_{K \times M}}$$

Pentru toate elementele cunoscute, eroarea se poate formula astfel:

$$MSE = \frac{1}{2} (r_{nm} - \sum_k u_{nk} v_{km})^2$$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- **Exemplu: gradient descendente**
  - inițializare  $U_{nk}$ ,  $V_{km}$  cu valori aleatoare
  - eroarea (funcția care trebuie minimizată) este:

$$MSE = \frac{1}{2} (r_{nm} - \sum_k u_{nk} v_{km})^2$$

- gradienții erorii în raport cu valorile căutate

$$\frac{\partial MSE}{\partial u_{nk}} = (\sum_k u_{nk} v_{km} - r_{nm}) v_{km}$$

$$\frac{\partial MSE}{\partial v_{km}} = (\sum_k u_{nk} v_{km} - r_{nm}) u_{nk}$$

- ajustarea valorilor

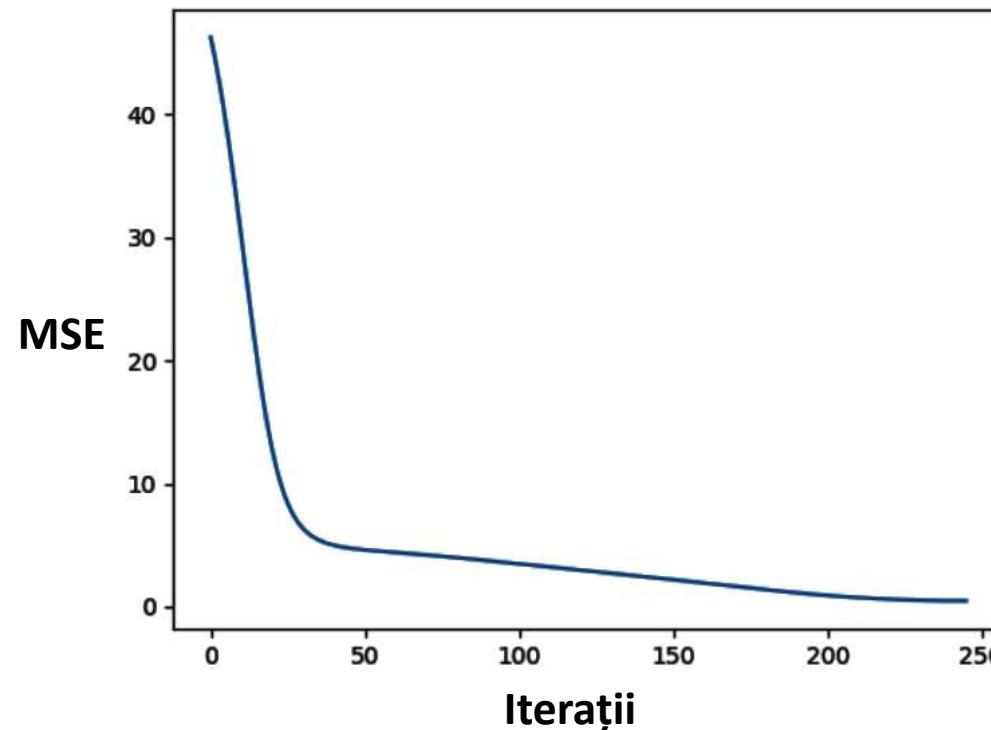
$$u_{nk} = u_{nk} - \alpha \frac{\partial MSE}{\partial u_{nk}}$$

$$v_{km} = v_{km} - \alpha \frac{\partial MSE}{\partial v_{km}}$$

# RS Colaborativ

## Metode bazate pe factorizarea matricelor

- Exemplu: gradient descendente
  - Evoluția erorii pe parcursul mai multor iterații



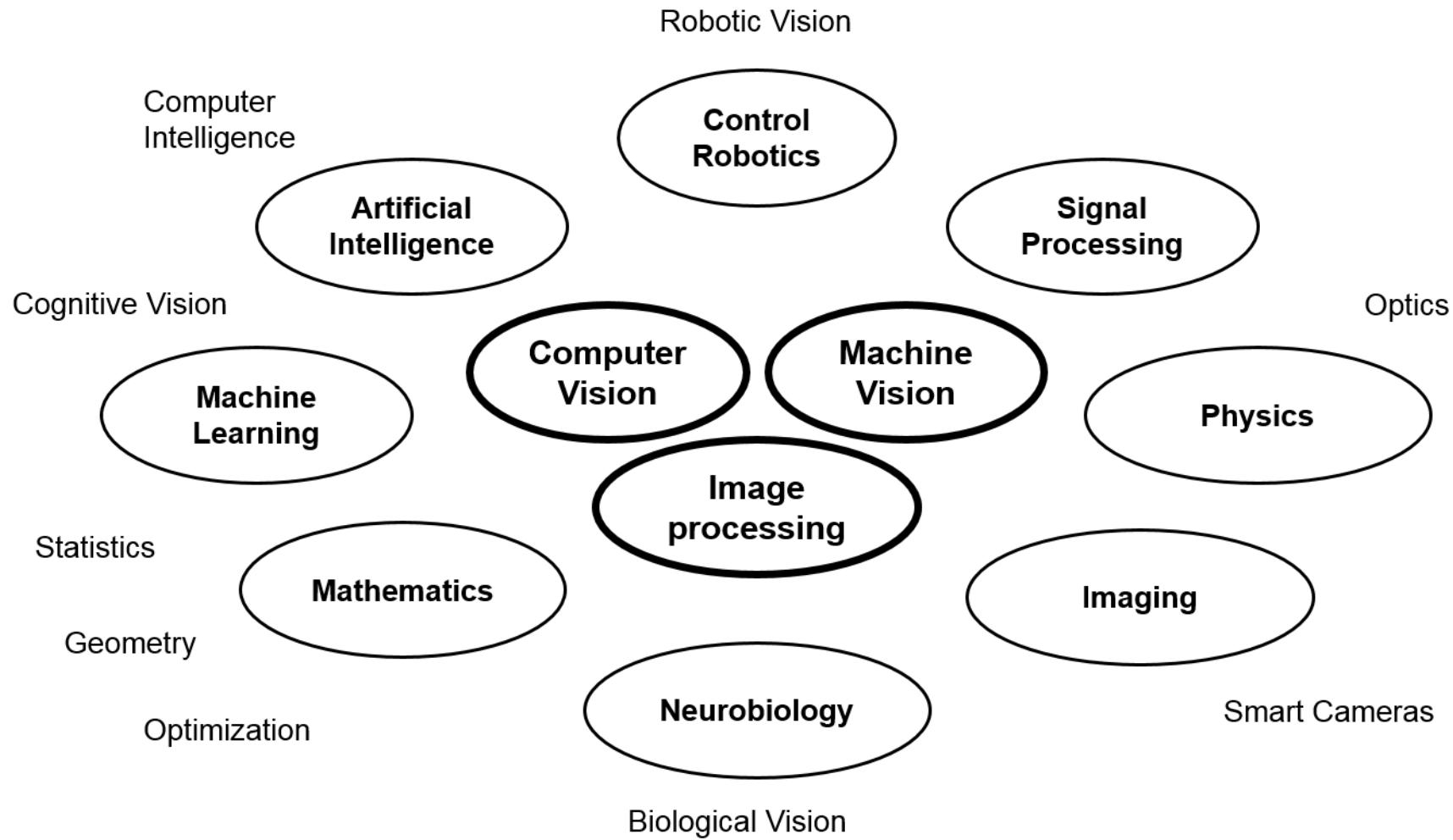
# Învățare automată

## 11. Recunoașterea obiectelor în imagini

**Marius Gavrilescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

- Domeniu complex, multidisciplinar
  - Implică mult mai mult decât o simplă procesare a imaginilor
- Aplicabilitate extinsă în numeroase alte domenii
- Tehnicile diferă substanțial funcție de natura și caracteristicile obiectelor ce se doresc detectate



- Interpretarea imaginilor se realizează la mai multe niveluri:
  - nivelul de jos – procesarea imaginii
    - operații de filtrare
    - eliminarea zgomotului
    - reglarea luminozității, contrastului
    - binarizare etc.

- Interpretarea imaginilor se realizează la mai multe niveluri:
  - nivel mediu – identificarea de caracteristici
    - identificarea de trăsături (*features*) utile
    - segmentare

- Interpretarea imaginilor se realizează la mai multe niveluri:
  - nivel înalt – înțelegerea imaginii
  - clasificarea obiectelor funcție de caracteristicile lor
  - interpretarea scenei din imagine, per ansamblu
  - parcurgerea automată și autonomă a obiectelor din imagine

- Pași implicați în procesare:

- achiziția imaginii – dispozitive dintr-o gamă diversă:
  - camere de diferite tipuri
  - dispozitiv de scanare 3D (CT, RMN)
  - microscop, ecograf
  - radar din diverse domenii (seismic, meteorologic etc)

- Recunoașterea automată a obiectelor
  - se respectă aceleași principii ca în cazul recunoașterii obiectelor de către oameni
  - **Detectia** obiectelor separate
  - **Descrierea** proprietăților geometrice și a poziției obiectelor
  - **Clasificarea** obiectelor – identificarea categoriei căreia îi aparțin
  - **Înțelegerea** relațiilor dintre obiecte

- Pași implicați în procesare:
  - filtrarea imaginii
    - îmbunătățirea contrastului, a detaliilor, a vizibilității obiectelor



- Pași implicați în procesare:

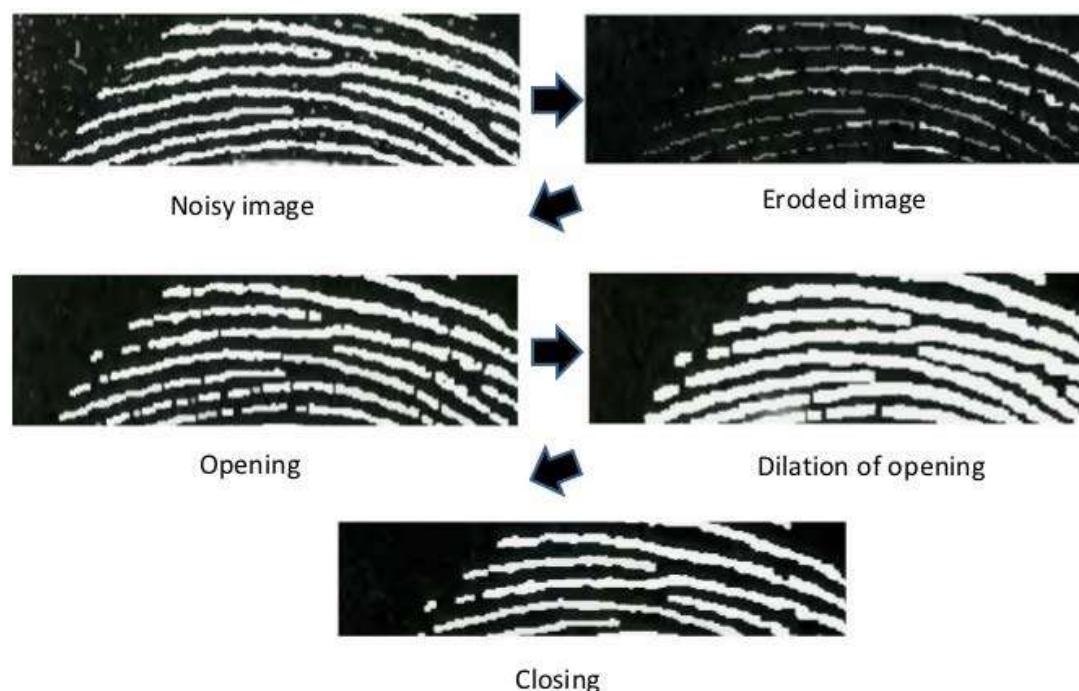
- reconstrucția / îmbunătățirea imaginii / reducerea zgomotului
  - în cazul în care apar erori la achiziție



## ■ Pași implicați în procesare:

### ■ operații morfologice

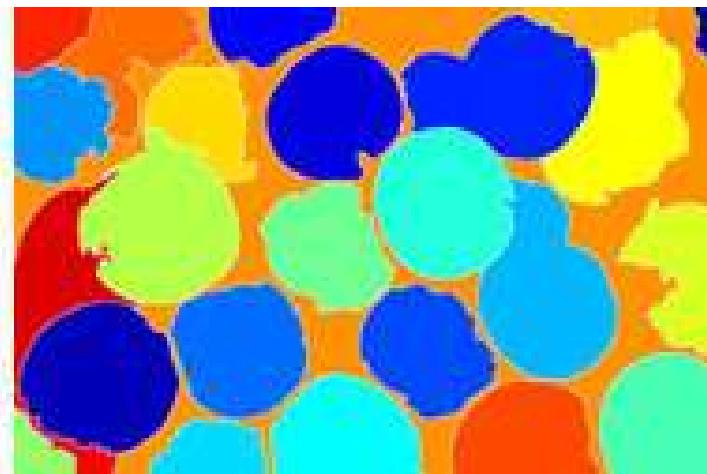
- asigurarea integrității și continuității formelor prin operații de eroziune, dilatare etc.



- Pași implicați în procesare:

- segmentare

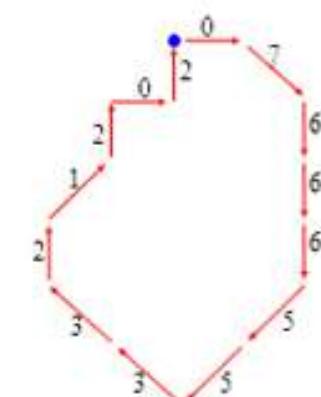
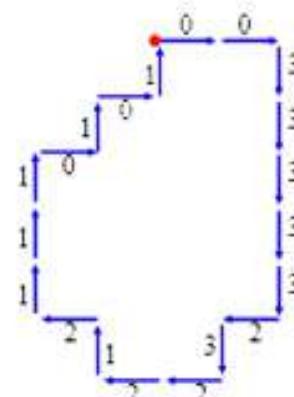
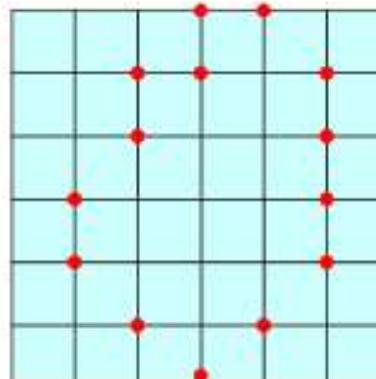
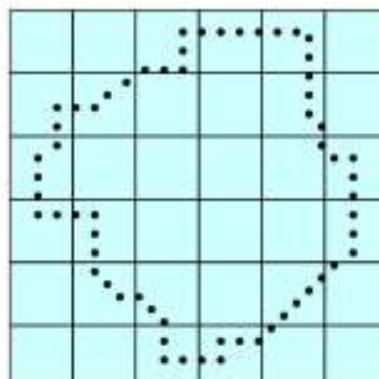
- separarea obiectelor de background



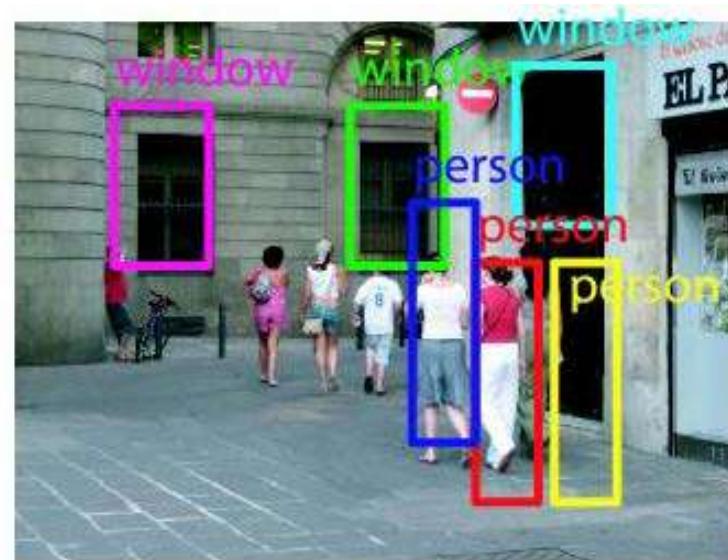
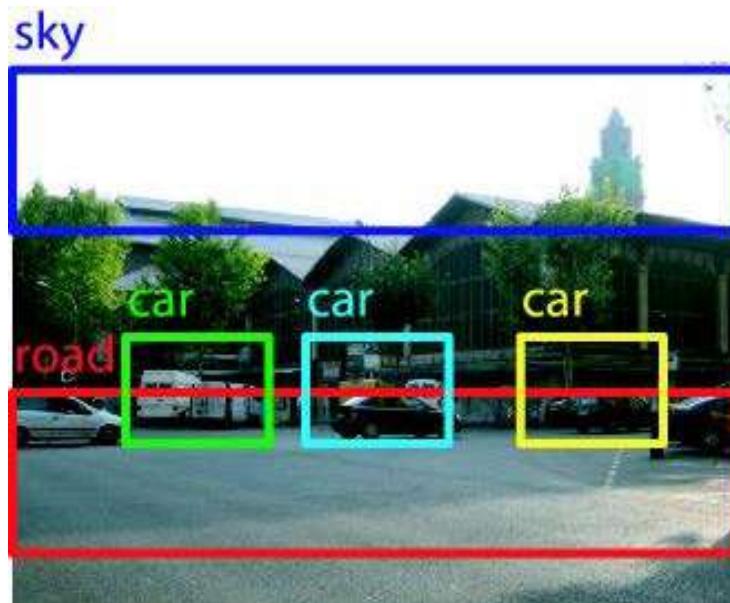
## ■ Pași implicați în procesare:

### ■ reprezentarea, descrierea obiectelor

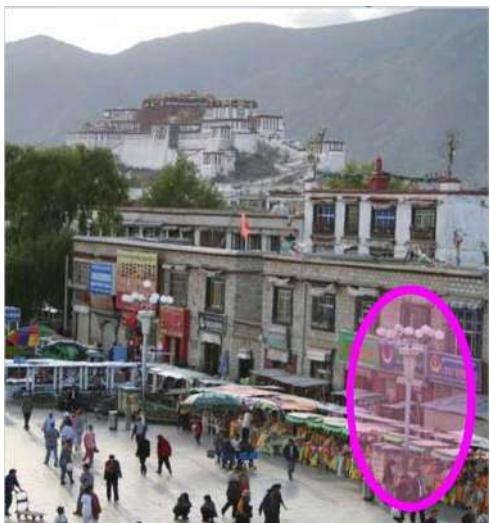
- generarea de descriptori și reprezentări ale contururilor și trăsăturilor obiectelor



- Pași implicați în procesare:
  - recunoașterea obiectelor
    - clasificare, împărțirea obiectelor pe categorii



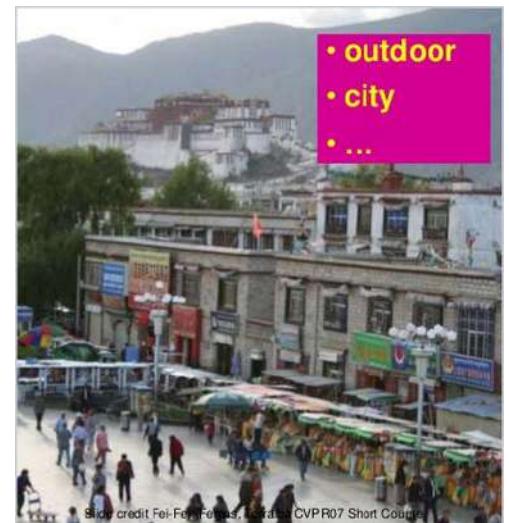
## Detectie, Descriere geometrică



## Clasificare

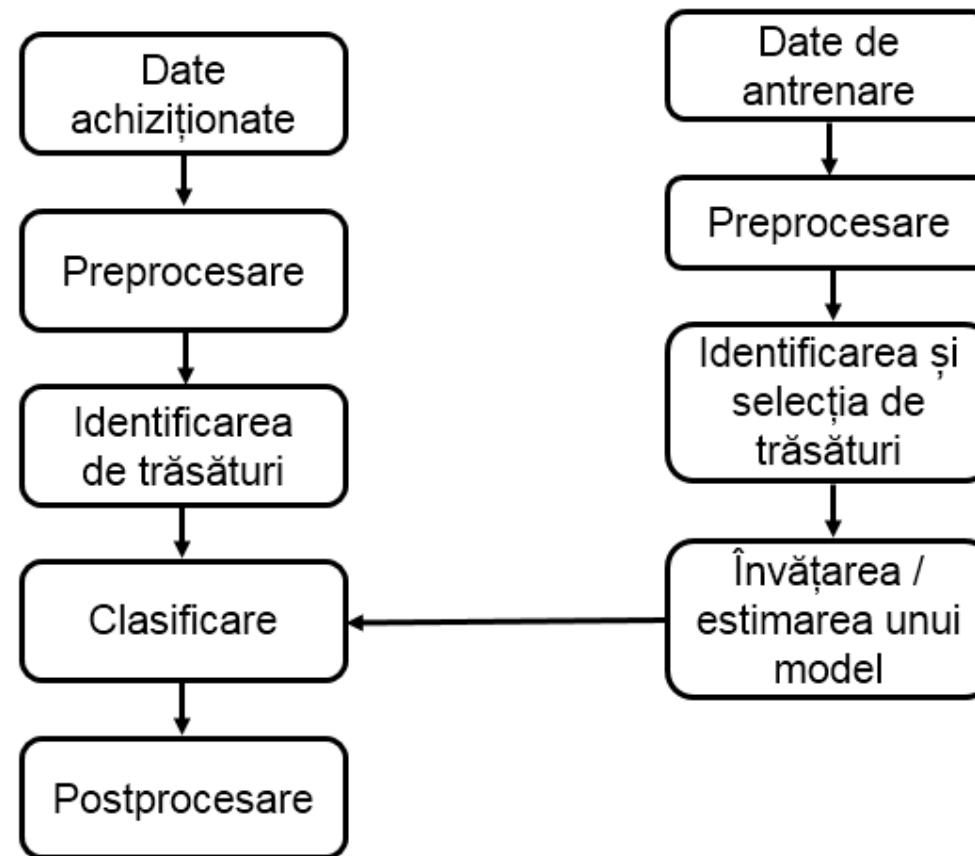


## Înțelegerea imaginii



- La baza procesului de identificare a obiectelor stă recunoașterea de tipare (*pattern*)
  - tipar - obiect, proces sau eveniment identificabil în mod sistematic
  - clasă (categorie) de tipare – mulțimea tiparelor care au o serie de attribute comune
  - clasificarea implică identificarea de tipare și încadrarea lor într-o clasă dintr-o mulțime prestabilită de clase

- Componentele de bază ale unui sistem de recunoaștere a tipelor



- Componentele de bază ale unui sistem de recunoaștere a tiparelor
  - Preprocesare
    - eliminarea zgomotului
    - accentuarea contururilor
    - scoaterea în evidență a unor componente reprezentative ale imaginii
  - Identificarea trăsăturilor
    - generarea de imagini care să conțină submultimi reprezentative ale informațiilor din imaginea initială

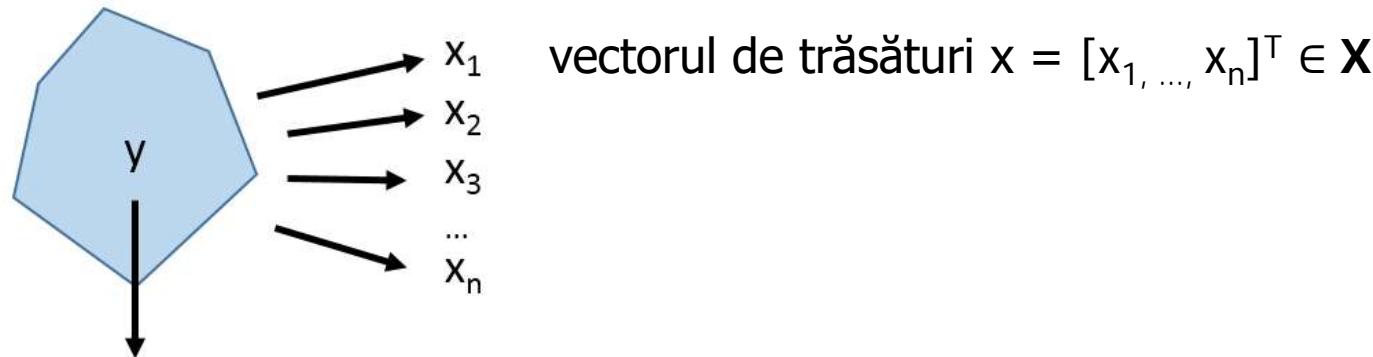
- Componentele de bază ale unui sistem de recunoaștere a tiparelor
  - Învățarea și estimarea unui model
    - găsirea legăturilor dintre trăsături și tipare
    - realizarea unei “mapări” de la mulțimea de trăsături la mulțimea de tipare
  - Clasificarea
    - utilizarea modelului anterior pentru încadrarea tiparelor în categorii predefinite

## ■ Reprezentarea tiparelor

- tiparul constă într-o mulțime de trăsături, organizate sub forma unui vector de trăsături (*feature vector*)

$$x = (x_1, x_2, \dots, x_n)$$

## ■ Reprezentarea tiparelor



Starea ascunsă  $y \in Y$  a obiectului

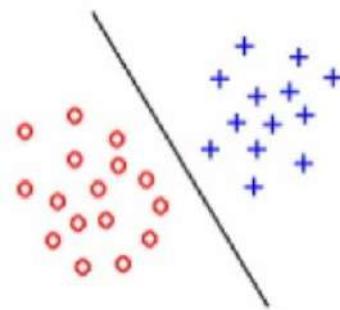
- mulțime de caracteristici abstrakte reprezentative pentru obiect
- nu poate fi direct măsurată
- tiparele cu stări ascunse egale aparțin aceleiași clase

Sarcina unui clasificator:

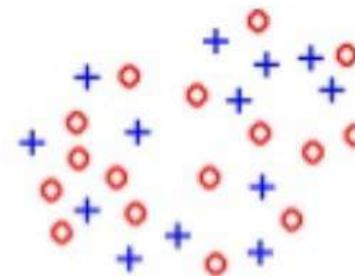
- identificarea unei reguli de decizie  $q : X \rightarrow Y$
- identificarea stării ascunse pe baza unei observații

## ■ Identificarea trăsăturilor

- scopul constă în identificarea acelor trăsături care să asigure o clasificare relevantă
- trăsăturile utile asigură faptul că
  - obiectele din aceeași clasă au caracteristici similare
  - obiectele din clase diferite au caracteristici diferite



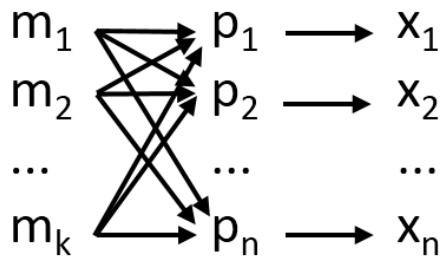
Trăsături utile



Trăsături irelevante

## ■ Identificarea trăsăturilor

- Problema se poate exprima ca o optimizare a parametrilor  $p_i$  ai unui identificator de trăsături (i.e. ponderile unei rețele neuronale)



Identificarea trăsăturilor

$m_1 \rightarrow x_1$   
 $m_2 \rightarrow x_2$   
 $\dots \quad \dots$   
 $m_k \rightarrow x_n$

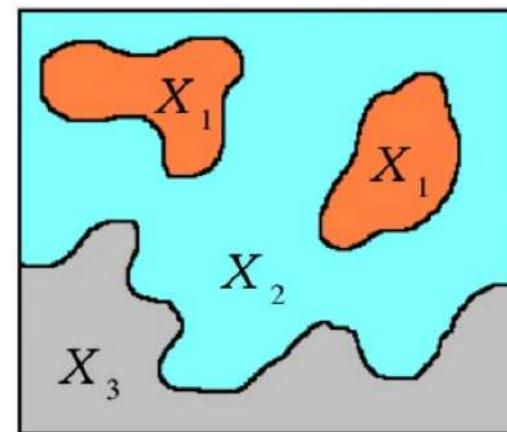
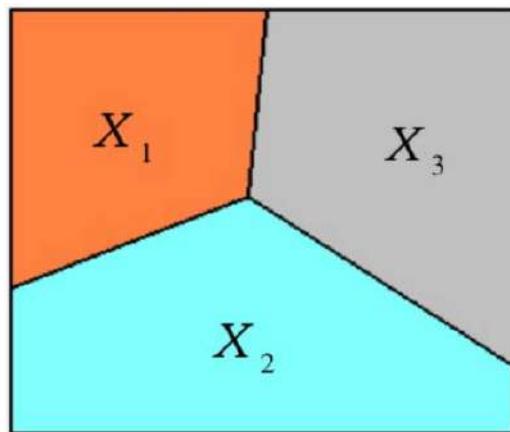
Selectia trăsăturilor

## ■ Clasificare

- Metode supervizate – se caută o funcție-obiectiv care să confere un criteriu de separare a obiectelor deja împărțite pe categorii
  - i.e. prin ce trăsături se deosebesc obiectele din categorii diferite
- Metode nesupervizate – se caută o reprezentare simplificată (adesea prin reducerea dimensionalității) care să conțină caracteristicile reprezentative ale obiectelor

## ■ Clasificare

- un clasificator partiiionează mulțimea  $\mathbf{X}$  a trăsăturilor în submulțimi corespunzătoare fiecărei clase
  - în cazul metodelor supervizate, acest lucru se realizează în faza de antrenare
- procesul de clasificare – stabilirea submulțimii de apartenență a unui vector oarecare de trăsături  $x$

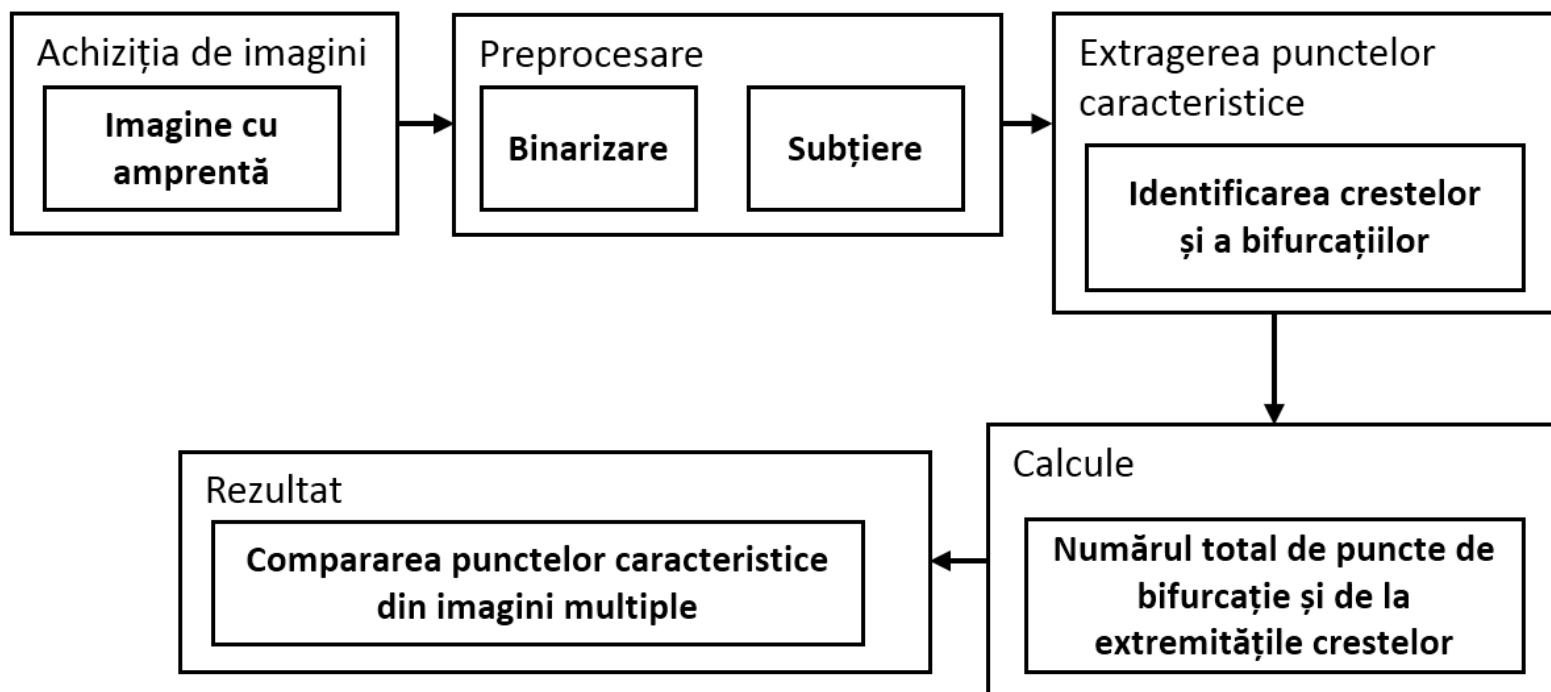


## ■ Studiu de caz 1: Recunoașterea amprentelor

- amprentele sunt considerate ca fiind formate din tipare de “crestă” și “văi”
- crestele sunt identificate folosind puncte caracteristice:
  - punctele cu care se încheie o creastă
  - punctele în care o creastă se bifurcă



## ■ Recunoașterea amprentelor



- Achiziția imaginilor
  - metode cu / fără cerneală



## ■ Binarizare

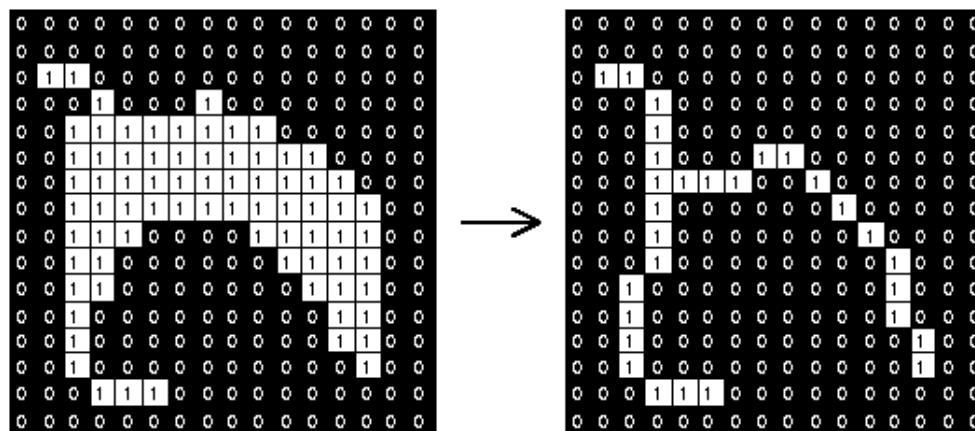
- convertirea unei imagini grayscale într- imagine binară prin stabilirea unui prag
  - valorile de intensitate ale pixelilor se setează pe 1 dacă depășesc valoarea prag, respectiv pe 0 altfel



## ■ Subtiere (*thinning*)

- reducerea grosimii contururilor

- se dorește ca liniile caracteristice ale amprentelor să aibă grosimea de 1 pixel
  - există mai multe metode (block filtering, diverse operații morfologice)



- Subțiere (*thinning*)

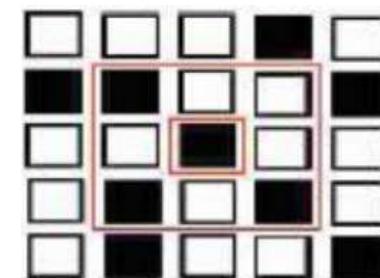
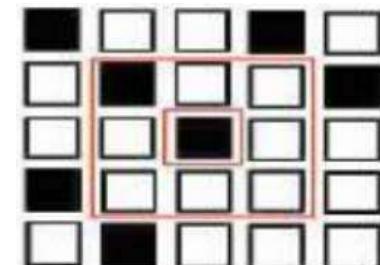
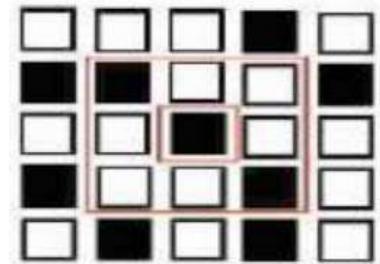
- nu se modifică punctele caracteristice ale amprentei



- Identificarea punctelor caracteristice
  - pentru fiecare pixel, se consideră o vecinătate  $3 \times 3$
  - se determină numărul de tranziții (*crossing number*)
  - numărul de tranziții =  $0.5 * \text{suma diferențelor dintre valorile intensităților a doi pixeli învecinați}$

## ■ Identificarea punctelor caracteristice

- dacă numărul de tranziții are valoarea
  - 1 - pixelul corespunzător se află la extremitatea unei creste
  - 2 - pixelul se află pe o creastă (dar nu la capătul ei)
  - > 3 – pixelul se află în poziția unei bifurcații



## ■ Clasificarea amprentelor

- punctele caracteristice constituie trăsăturile reprezentative ale unei imagini cu amprente
- pozițiile lor formează un vector de trăsături
- acesta poate fi utilizat ca valori de intrare într-un clasificator supervizat / nesupervizat

## ■ Studiu de caz 2: *Eigenfaces*

- Extragerea trăsăturilor din imagini pentru recunoaștere facială
- Trăsăturile = vectori care indică cele mai importante variații din imagine
- Variații importante: contururile feței, regiunile ochilor, nasului, gurii etc.
- Variații nedorite: zgomot, patternuri din fundal

## ■ Studiu de caz 2: *Eigenfaces*

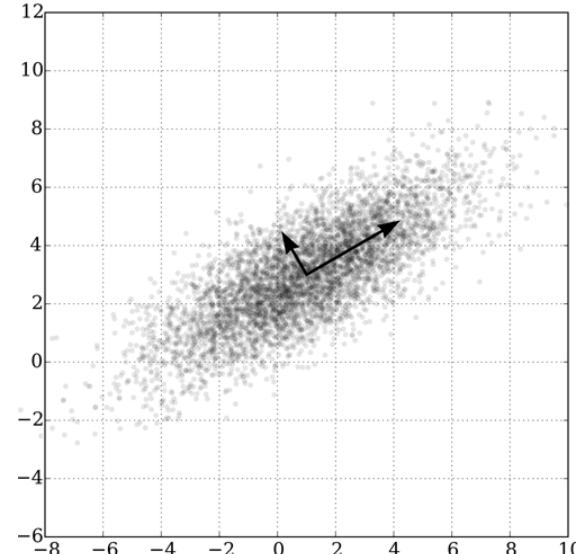
- Eigenfaces = “fete proprii”
- Originea denumirii
  - pentru detectia trăsăturilor se identifică componentele principale ce rezultă din valorile pixelilor imaginii
  - componentele principale = vectorii proprii (eigenvector) ai unei matrice construită pe baza variațiilor pixelilor

## ■ Componente principale

- Vectori de indică variațiile cele mai ample dintr-un set de date
- Exemplu 2D (datele și vectorii au două coordonate):

Cei doi vectori indică direcțiile celor mai pronunțate variații ale punctelor din setul de date

Vectorii constituie componentele principale ale setului de puncte



## ■ Componente principale

- Datele: puncte 2D  $P_i(x_i, y_i)$
- Varianță:
  - $\text{Var}(x) = \sum (x_i - \bar{x})^2$
  - $\text{Var}(y) = \sum (y_i - \bar{y})^2$
- Covarianță
  - $\text{Covar}(x, y) = \sum (x_i - \bar{x})(y_i - \bar{y})$
- Matricea de covarianță

$$\text{cov} = \begin{bmatrix} \text{Var}(x) & \text{Covar}(x, y) \\ \text{Covar}(y, x) & \text{Var}(y) \end{bmatrix}$$

## ■ Componente principale

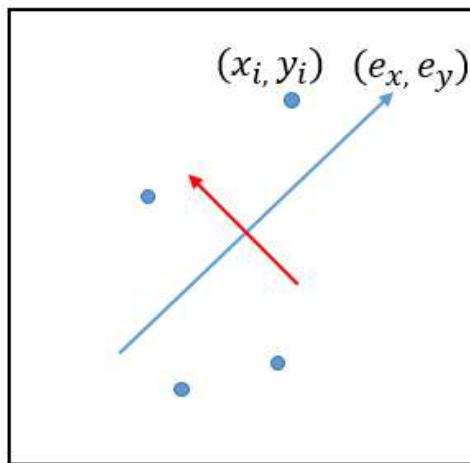
- Componentele principale sunt vectorii proprii (engl. *eigenvectors*) ai matricei de covariantă
- $V$  – vectori proprii,  $\lambda$  = valori proprii
- $\text{cov} * V = \lambda * V$
- $\det(\text{cov} - \lambda I) = 0 \Rightarrow$  calculul valorilor și vectorilor proprii

## ■ Pentru puncte din plan:

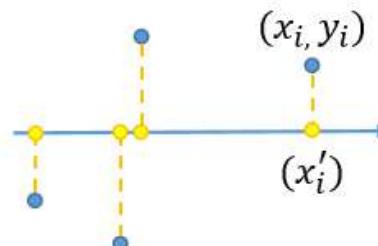
- Se obțin doi vectori proprii și două valori proprii
- Se proiectează punctele pe direcția uneia dintre vectorii proprii (de ex. vectorul cu valoarea proprie corespunzătoare cea mai mare)
- Rezultă punctele într-un spațiu 1D (o singură coordonată în loc de două)
- Se aplică proiecția inversă și rezultă alt set de puncte 2D pe baza coordonatei punctelor 1D
- Spunem că noul set de puncte s-a reconstruit pe baza reprezentării lor într-un spațiu cu mai puține dimensiuni

## ■ Reducerea dimensiunii datelor

- Prin proiecția punctelor  $x_i, y_i$  pe direcția uneia dintre vectorii proprii (ex  $e_y$ ), din puncte 2D rezulta puncte 1D

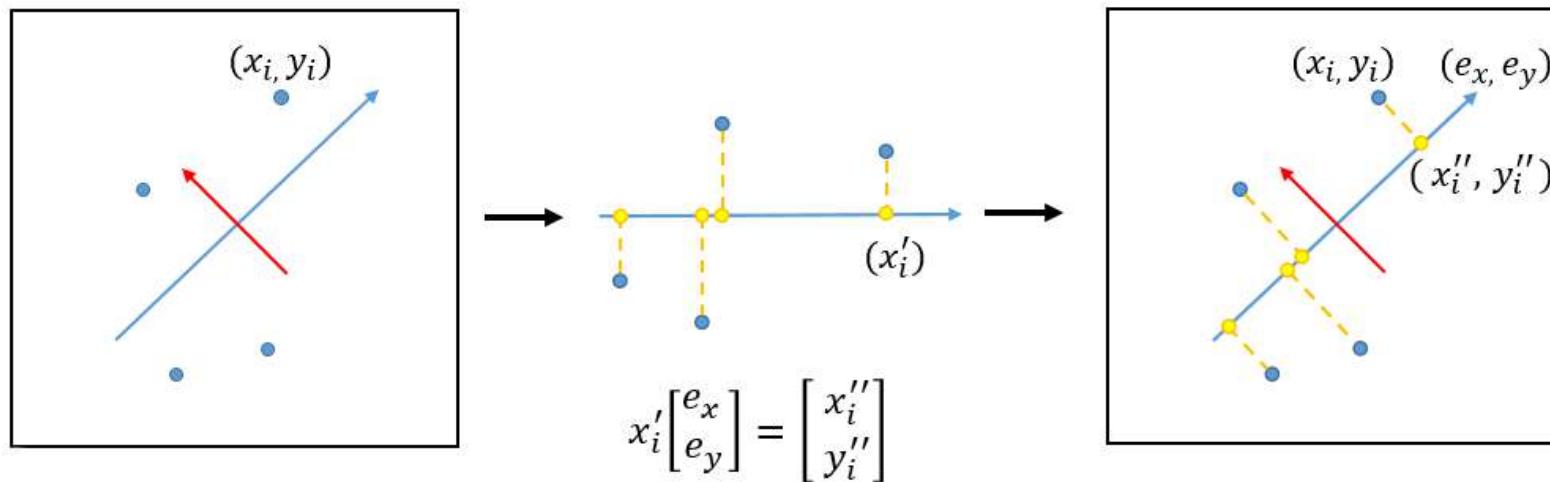


$$[e_x \quad e_y] \begin{bmatrix} x_i \\ y_i \end{bmatrix} = x'_i$$



- Reconstrucția datelor pe baza celor de dimensiune redusă

- Prin proiecție inversă rezultă coordonatele punctelor 2D calculate pe baza coordonatei determinate anterior în spațiul 1D



## ■ În caz general

- Datele au n dimensiuni
- Punctele dintr-un spațiu n-dimenional au n coordonate  $P = [x_1, x_2, \dots, x_n]$
- Sunt (maxim) n componente principale (n vectori proprii și n valori proprii)
- Se alege un numar limitat de componente principale

- Proiecția datelor din n dimensiuni în k < n dimensiuni

$$\begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \dots & \dots & \dots & \dots \\ e_{k1} & e_{k2} & \dots & e_{kn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_{p1} \\ x_{p2} \\ \vdots \\ x_{pk} \end{bmatrix}$$

Matrice de proiecție formată  
din k vectori proprii

Punctul inițial, n-dimensional

Punctul proiectat, k-dimensional

- Reconstucția datelor în n dimensiuni pe baza datelor din k dimensiuni

$$\begin{bmatrix} x_{p1} \\ x_{p2} \\ \dots \\ x_{pk} \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \dots & \dots & \dots & \dots \\ e_{k1} & e_{k2} & \dots & e_{kn} \end{bmatrix}^T = \begin{bmatrix} x_1' \\ x_2' \\ \dots \\ x_n' \end{bmatrix}$$

Punctul proiectat, k-dimensional

Transpusa matricii de proiecție

Punctul reconstruit, n-dimensional,  
este o aproximatie a celui initial

## ■ Cazul imaginilor

- Dataset-ul format din imagini 100x100
- Fiecare imagine este un punct într-un spațiu cu  $100 \times 100 = 10000$  dimensiuni
- Nu toate dimensiunile conțin trăsături reprezentative pentru conținutul imaginii



## ■ Cazul imaginilor

- Se pot determina maxim 10000 componente principale (vectori proprii)
- Doar câțiva dintre acești vectori indică variații induse de trăsături semnificative ale imaginilor
- Cei mai importanți vectori proprii:



## ■ Eigenfaces:

Exemple de imagini originale



Imaginiile reconstruite în urma proiecției folosind un număr limitat de vectori proprii



# Învățare automată

## 12. Notiuni de OCR

**Marius Gavrilescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

- Recunoașterea caracterelor (OCR – Optical Character Recognition)

- 4 etape:

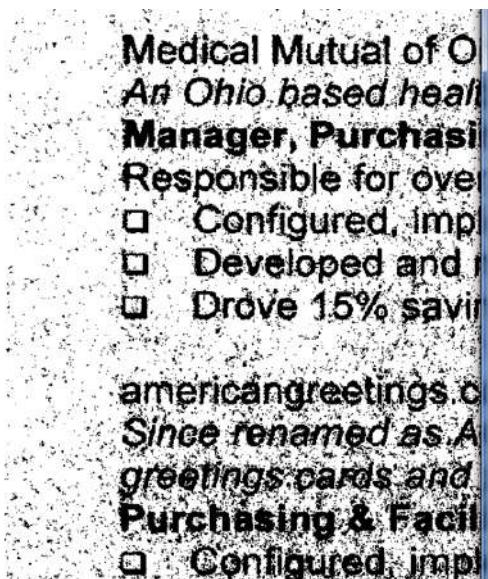
- Preprocesare
  - Segmentare
  - Identificarea trăsăturilor
  - Clasificare

## ■ Preprocesare

- Reducerea zgomotului
- Binarizare
- Alinierea textului
- Înlăturarea înclinării caracterelor

## ■ Reducerea zgomotului

- Filtrare de mai multe tipuri (filtru median, gaussian etc.)

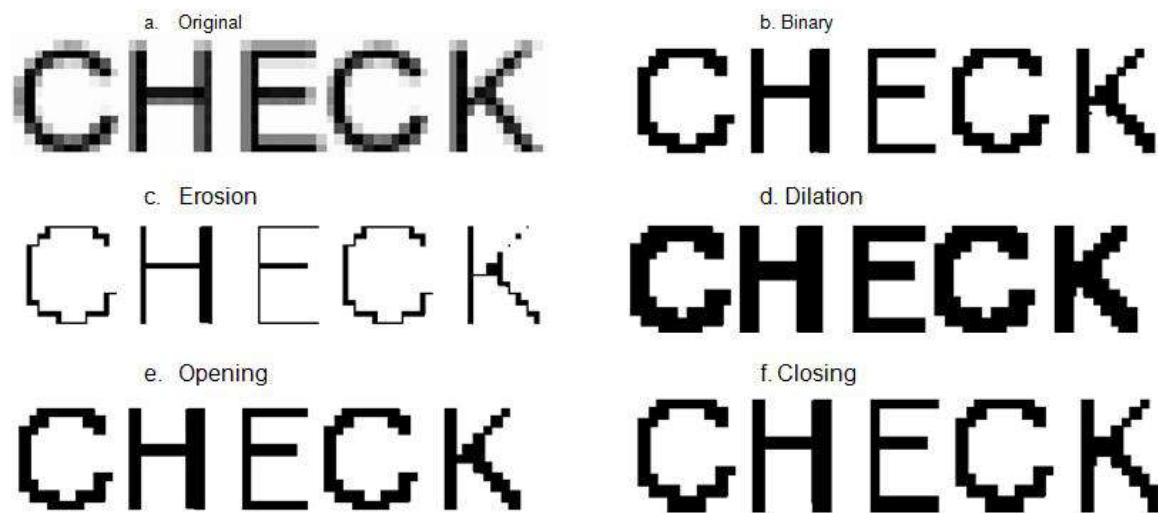


Medical Mutual of Ohio  
An Ohio based health care company.  
**Manager, Purchasing**  
Responsible for overall purchasing strategy.  
□ Configured, implemented and managed the system.  
□ Developed and maintained relationships with suppliers.  
□ Drove 15% savings in procurement costs.

american greetings.com  
Since renamed as AG International.  
A greeting cards and products company.  
**Purchasing & Facilities**  
□ Configured, implemented and managed the system.

## ■ Binarizare

- Convertirea valorilor dintr-un interval oarecare în mulțimea {0, 1}, folosind o valoare prag
- Se poate îmbunătăți rezultatul prin aplicarea de operații morfologice de dilatare, închidere etc.



## ■ Alinierea textului

- Se dorește ca rândurile textului să fie
  - Paralele între ele
  - Aliniate la sistemul de coordonate al imaginii  
(i.e. paralele cu axa orizontală)

This is a document image  
that will present you the common problem of OCL

The non-parallel text line is  
a very usually not problem  
making difficult the slant angle estimation.

The hill anddale writing is also not  
as well as the slanted and connected  
characters.



This is a document image  
that will present you the common problems of OCL

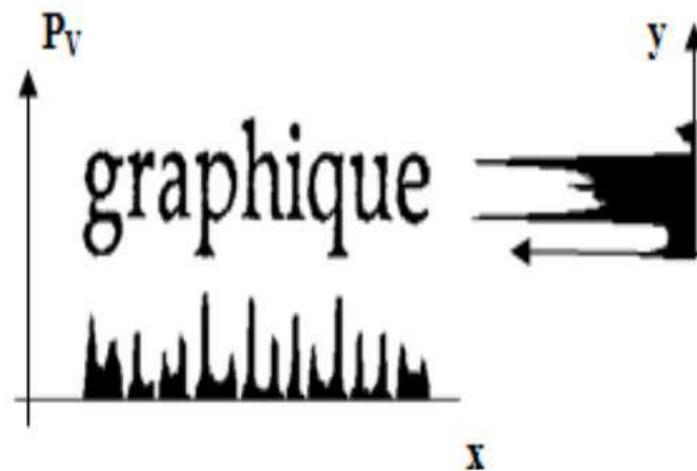
The non-parallel text line is  
a very usually not problem  
making difficult the slant angle estimation.

The hill and dale writing is also not  
as well as the slanted and connected  
characters.

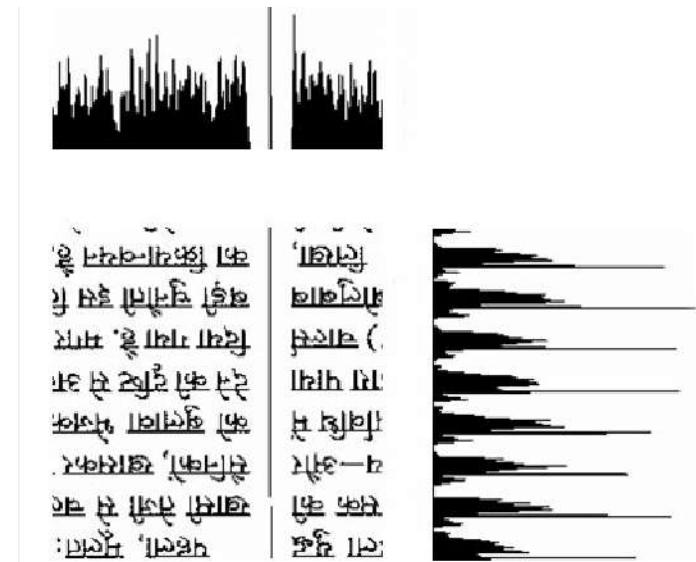
## ■ Metode de aliniere

- Metoda profilurilor de proiecție

- Profilul de proiecție este o histogramă a pixelilor situați pe drepte orizontale / verticale - Rezultă două tipuri de profiluri: orizontale și verticale



- Metode de aliniere
  - Metoda profilurilor de proiecție
  - Profilul vertical = vector ce conține numerele de pixeli negri situați pe drepte perpendiculare pe axa y
  - Profilul orizontal = vector ce conține numerele de pixeli negri situați pe drepte perpendiculare pe axa x

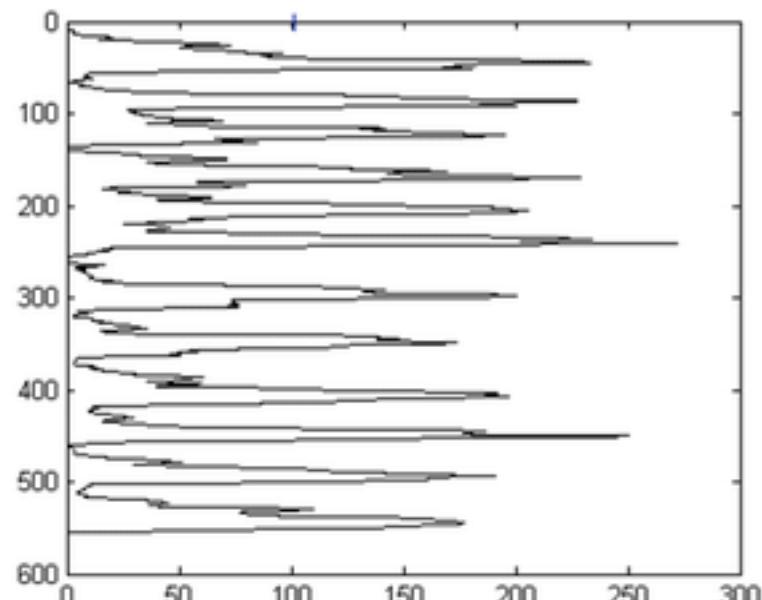


## ■ Metode de aliniere

### ■ Metoda profilurilor de proiecție

Inizialmente abbiamo affermato che soltanto riuscirebbe a riparare ogni e qualunque macchia del tutto ciò che è riportato a c basterebbe che qualche volta Francesco non il suo comportamento incolpevole. Qualcun penserebbe che non abbiano fatto abbastanza ➡

Francesco che riparare ogni errore baster bisognato e a non attuare un comportam Più di tutto pensiamo che avere affermato a creare le condizioni idonee per fran comportamento alle regole sociali. Ciò che non abbiamo del tutto sostenuto è d



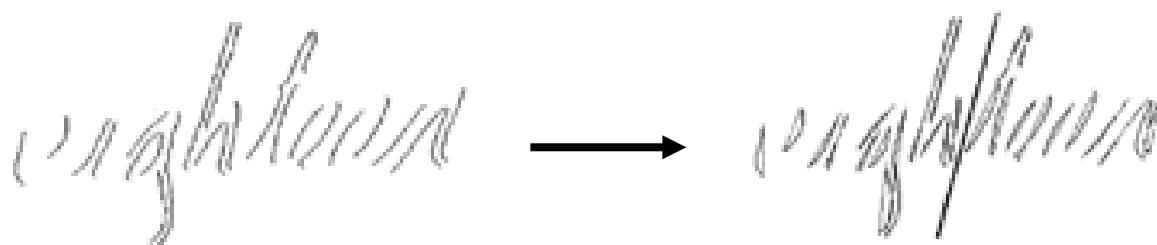
## ■ Metoda profilurilor de proiecție

- Profilul de proiecție orizontală va atinge
  - maxime locale pentru regiunile din imagine care conțin rânduri de text
  - minime locale pentru regiunile ce conțin spațiile dintre rânduri
- Se determină o serie de profiluri de proiecție din mai multe unghiuri
- Pentru fiecare unghi, se evaluatează diferențele dintre minimele și maximele locale ale profilurilor
  - Unghiurile cu diferențe mari corespund unor linii de text rotite sub acele unghiuri
- Unghiul la care trebuie aliniat textul este o medie ponderată a unghiurilor identificate anterior

- Înlăturarea înclinării caracterelor
  - Uniformizarea gradului de înclinare a caracterelor
  - Înclinarea diferă funcție de stitul de redactare, font, format etc.
  - Se urmărește ca toate caracterele să fie înclinate sub un unghi unic, standard.

## ■ Înlăturarea înclinării caracterelor - Metode frecvent utilizate:

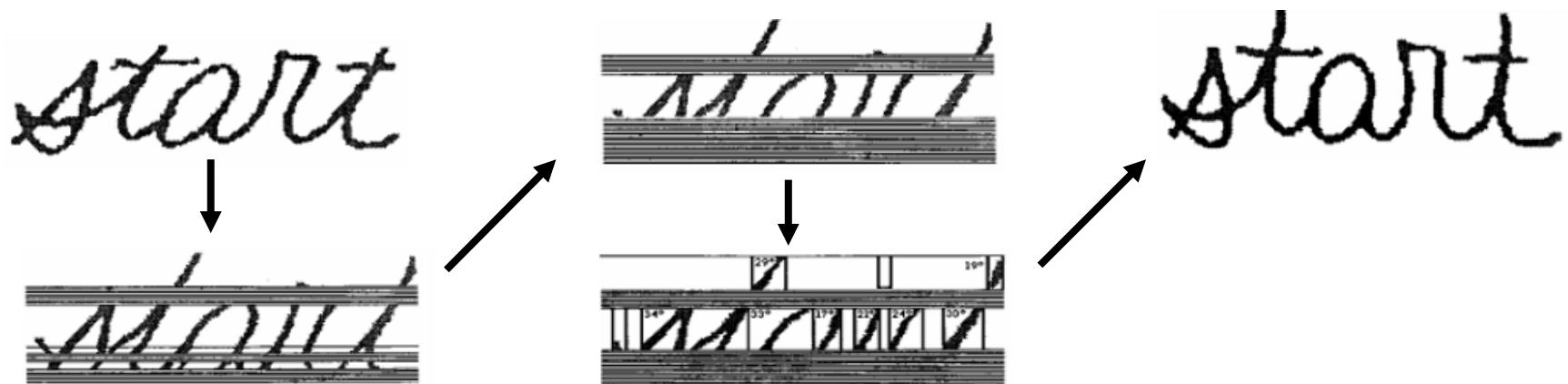
- Calculul unghiului mediu al componentelor aproape verticale (en. *near-vertical elements*)
  - Se estimează liniile al căror unghi se apropie de  $90^0$  (de ex. se pot folosi combinații de filtre 1 dimensionale)
  - Se determină unghiul de înclinare optimală ca fiind valoarea medie a unghiurilor acestor linii
  - Se aplică o serie de transformări geometrice care se deduc din unghiurile fiecărui caracter individual și din unghiul optimal



- Înlăturarea înclinării caracterelor - Metode frecvent utilizate:

- Metoda Bozinovic – Srihari

- Se izolează regiunile din imagine care conțin componentele cele mai reprezentative pentru gradul de înclinare al textului
    - Se transformă pixelii caracterelor în ideea de a minimiza diferențele dintre unghurile componente față de axa verticală



## ■ Segmentare

- Separarea cuvintelor și caracterelor individuale de restul imaginii

This is a document image  
that will present you the common problems of OCR

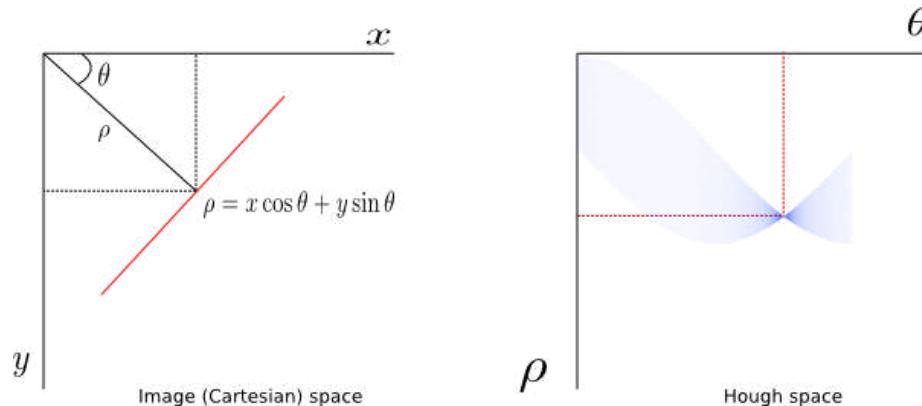
The non-parallel text line is  
a very usually met problem  
making difficult the skew angle estimation

The bold and slanted writing is also met  
as well as the slanted and connected  
characters.



## ■ Segmentare

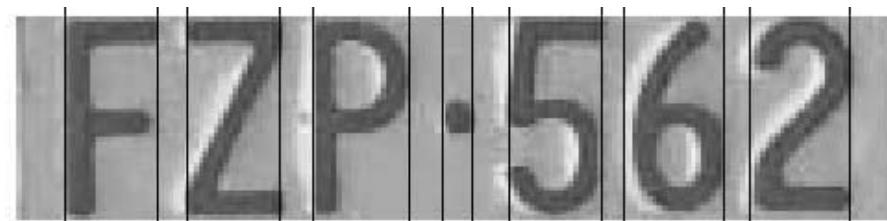
- Multiple metode care se utilizează funcție de specificul imaginilor procesate:
  - Detectia liniilor din imagine
    - Utilă în special pentru recunoașterea caracterelor de tipar
    - Adesea se folosește transformata Hough



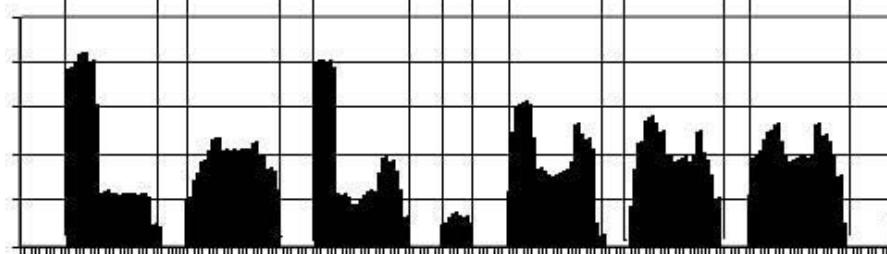
## ■ Segmentare

- Se pot utiliza profilurile de proiecție

Imaginea inițială



Histograma pixelilor  
pe coloane, folosind  
imaginea binarizată



## FAC-SIMILE OF GETTYSBURG ADDRESS.

*Address delivered at the dedication of the  
Cemetery at Gettysburg.*

*Four score and seven years ago our fathers  
brought forth on this continent, a new na-  
tion, conceived in liberty, and dedicated  
to the proposition that all men are cre-  
ated equal.*

*Now we are engaged in a great civil war,  
testing whether that nation, or any nation so  
conceived and so dedicated, can long  
endure. We are met on a great battlefield  
of that war. We have come to dedicate a  
portion of that field, as a final resting  
place for those who here gave their lives  
that this nation might live. It is alto-  
gether fitting and proper that we should  
do this.*

*But, in a larger sense, we can not dedi-*

## ■ Segmentare

### ■ Analiza componentelor conexe

- Se identifică regiunile care conțin pixeli cu cel puțin un vecin de aceeași valoare (sau de o valoare apropiată)
- Se asignează câte o etichetă (*label*) fiecărei regiuni
- Se reunesc etichetele caracterelor similare (în etapa de clasificare)

## ■ Segmentare

- Run-Length Smoothing Algorithm (RLSA)
  - Se subdivizează imaginea în regiuni dreptunghiulare
  - Fiecare regiune conține un element distinctiv din imagine (de exemplu un paragraf, un grup de cuvinte sau caractere etc.)
  - Presupunând că imaginea este binarizată, se transformă valorile pixelilor astfel încât:
    - Pixelii 0 devin 1 dacă numărul de pixeli 0 adiacenți este mai mic sau egal cu o limită impusă C
    - Pixelii cu valoarea 1 rămân neschimbați
  - Rezultatul este conectarea regiunilor aflate la distanță de cel mult C pixeli una de alta

- Segmentare
  - Run-Length Smoothing Algorithm (RLSA)
  - $C = 3$
  - Input: [1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1]
  - Output: [1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]

# ■ Segmentare

## ■ Run-Length Smoothing Algorithm (RLSA)

...and even they had many, twenty items and even so small or big from them express it. But what was added into my mind more than anything was the complete lack of professionalism and absence of content in those early shows.

If they could get away with that and still have listeners, I could do so much better and wipe the board, or so I thought.

Now in July 2002, we have hundreds of thousands of podcasts showing up the internet super-speed Internet, and one must ask, how things improved over that reality. Initially, there are some great shows out there, but there is still a lot of noise, nothing out (let's be honest) except.

That said, it doesn't matter really, does it? It's a hobby, after all, isn't it? And it's fun to do and there is plenty of room for success. I honestly do not have a problem with that. But it isn't for me, because I have already realized that this new venture in column, this RSS feed, medium, easy podcasting is actually a backwash step from my original goal of externalizing the masses.

Podcasting is slowly becoming better known among the free public. True, but better known in what? As a slightly geeky and complicated thing to do, for one thing, or completely understandable for another.

Some groups are calling what the BBC do with their "Listen Again" facility, or even the "They Are" sections on websites, podcasts. Certainly they are not something many are using immediately. They have always been under download and always will be under download.

*I am no longer referring to myself as a podcaster*

*To simply podcast would be limiting my potential*

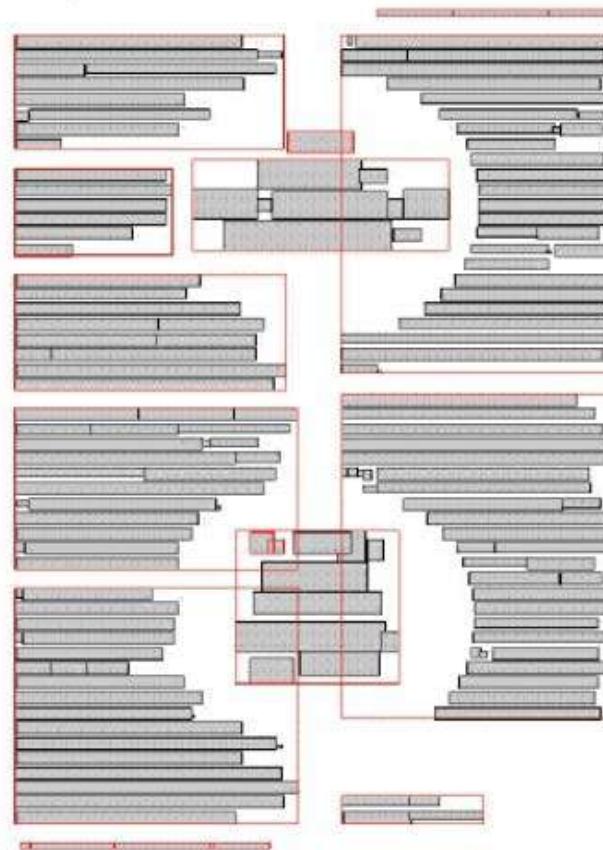
Richard Weber  
<http://www.rweber.com>

I am no longer referring to myself as a podcaster. I find the term limiting. I do more than that. I have a daily radio show, talk radio to people, and it can be listened to or the topic of a bulletin is discussed from the website. Yes, the show can be downloaded by the RSS feed as a podcast and listened to on an MP3 player at a later date if so required. But it isn't just an audio show. No, there is video and

photographs, news and views and fun and games. I am now thinking that my show is much better suited as a web-based entertainment medium where podcasting is just one small part of the show.

It is also where I believe the larger audience lies. There are thousands of people buying hi-fi gear now for the first time, according to the inference - so the new will audience they have learned, and they can book their holidays online,

watch TV online and talk to their friends all as off-the-cuff. The usual home computer and laptop are becoming the all-in-one entertainment centers. People are looking for something things to do with their computers, and this is where I come in. I want to enhance this. To simply podcast would be limiting my potential and, in my humble opinion, very much a backwash step.



## ■ Segmentare

### ■ Run-Length Smoothing Algorithm (RLSA)

Cyclone 'Titli'  
leaves 8 dead  
in Andhra

**I**PTI, AMARAVATI (AP)/  
**N**BHBANESWAR: A very se-  
vere cyclonic storm packing  
**P**winds of up to 150 kmph and  
**U**widespread rains hit eastern  
**T**India Thursday killing eight  
people in Andhra Pradesh  
and damaging homes, up-  
rooting trees and power lines  
**M**in the state and in Odisha.  
**A**

Cyclone 'Titli' made land-  
**G**fall on the eastern coast early  
**E**on Thursday wreaking havoc  
mainly in Srikakulam and  
Vizianagaram districts of  
Andhra Pradesh and Odisha's  
Gajapati and Ganjam dis-  
tricts.

Traffic on the Chennai-  
Kolkata National Highway  
was hit after the uprooted  
trees blocked some sections  
of the road, according to offi-  
cials. Pg 6 >>



Cyclone Titli  
leaves 8 dead  
in Andhra

**R**  
**L**  
**S**  
**A**  
**H**  
**O**  
**R**  
**I**  
**N**  
**Z**  
**O**  
**N**  
**T**  
**a**  
**t**  
**r**  
**i**  
**c**  
**a**



Cyclone 'Titli'  
leaves 8 dead  
in Andhra

**R**  
**L**  
**S**  
**A**  
**V**  
**e**  
**r**  
**t**  
**i**  
**c**  
**a**

Traffic on the Chennai-  
Kolkata National Highway  
was hit after the uprooted  
trees blocked some sections  
of the road, according to offi-  
cials.

## ■ Identificarea trăsăturilor

- Fiecare caracter se reprezintă folosind un vector de trăsături (*feature vector*)
- Caracterele se vor recunoaște pe baza vectorilor corespunzători
- Așadar, trăsăturile trebuie alese astfel încât să fie:
  - suficient de robuste (să maximizeze probabilitatea de recunoaștere a caracterelor corespunzătoare)
  - suficient de reprezentative (să minimizeze operațiile, resursele necesare, precum și dimensiunea datelor de antrenare)

- Identificarea trăsăturilor

- În cazul caracterelor, trăsăturile sunt de cel puțin trei tipuri

- Statistice

- Structurale

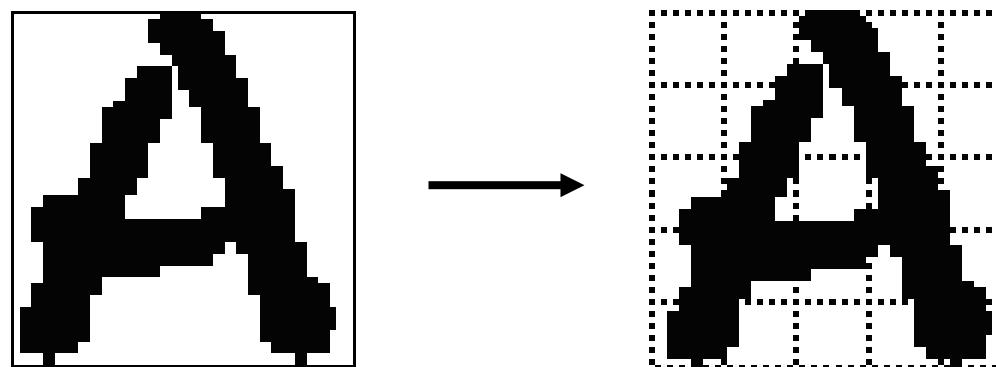
- Transformări globale

## ■ Trăsături statistice

- Reprezentarea caracterelor prin intermediul unei distribuții de probabilitate a anumitor grupuri de pixeli
- Principalele categorii de trăsături statistice:
  - Trăsături bazate pe regiuni (*zoning*)
  - Trăsături bazate pe proiecții și profiluri
  - Trăsături bazate pe tranziții

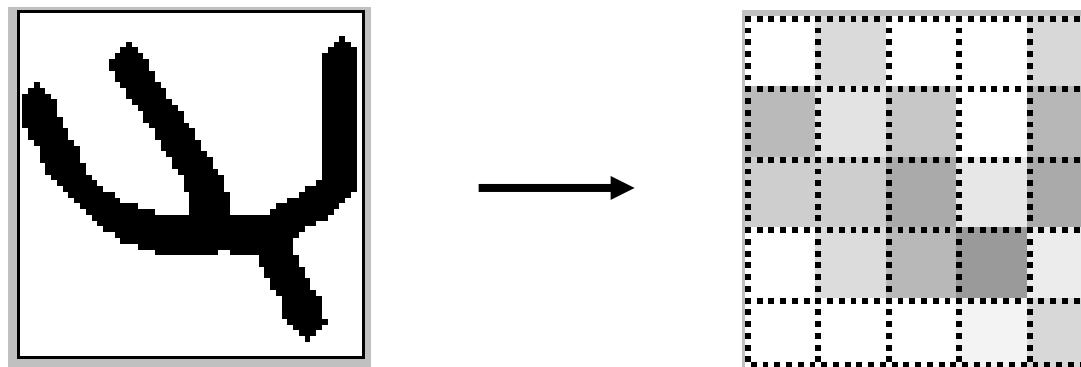
## ■ Trăsături bazate pe regiuni

- Imaginea se subdivizează în NxM regiuni
- Din fiecare regiune se calculează o componentă a vectorului de trăsături
- Aceste trăsături conțin caracteristicile locale ale imaginii (și nu pe cele globale)



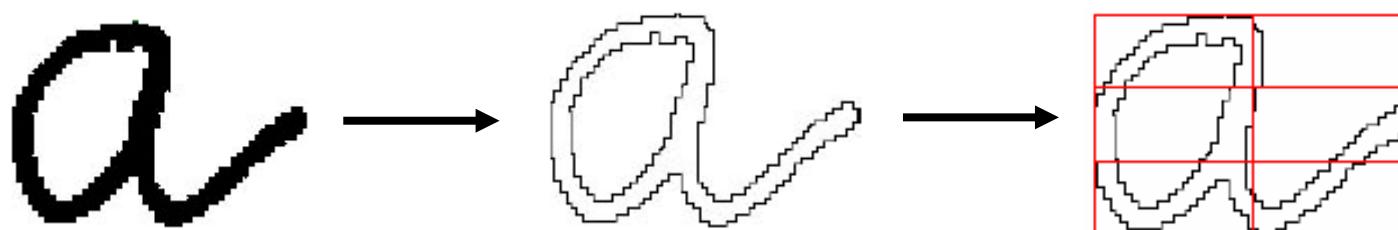
## ■ Trăsături bazate pe regiuni

- De exemplu, pentru fiecare regiune se poate determina numărul de pixeli negri
- Se normalizează aceste valori pentru toate regiunile
- Rezultă un vector de trăsături ce indică distribuția de probabilitate a pixelilor ce compun caracterul vizat



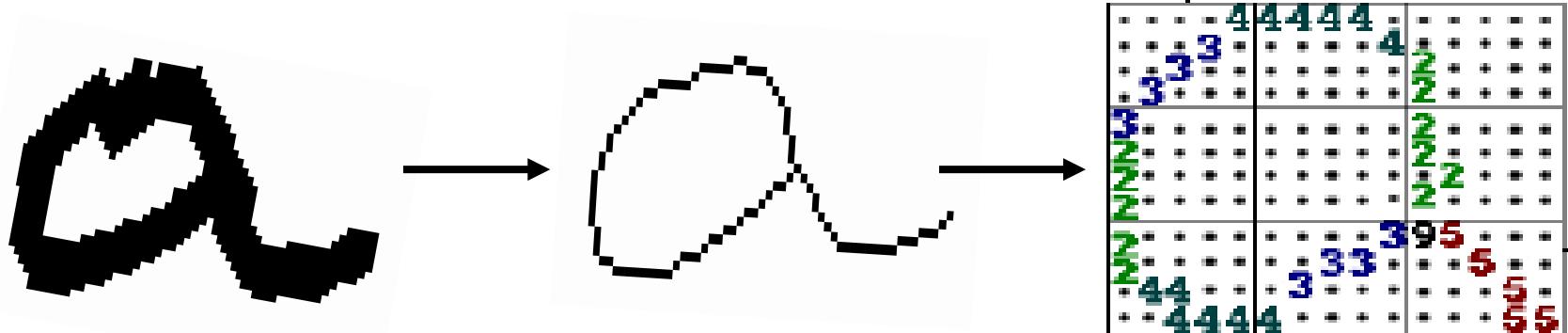
## ■ Trăsături bazate pe regiuni

- Se poate lua în calcul conturul caracterului din imagine
- Pentru fiecare zonă, se poate realiza o histogramă a direcțiilor contului respectiv



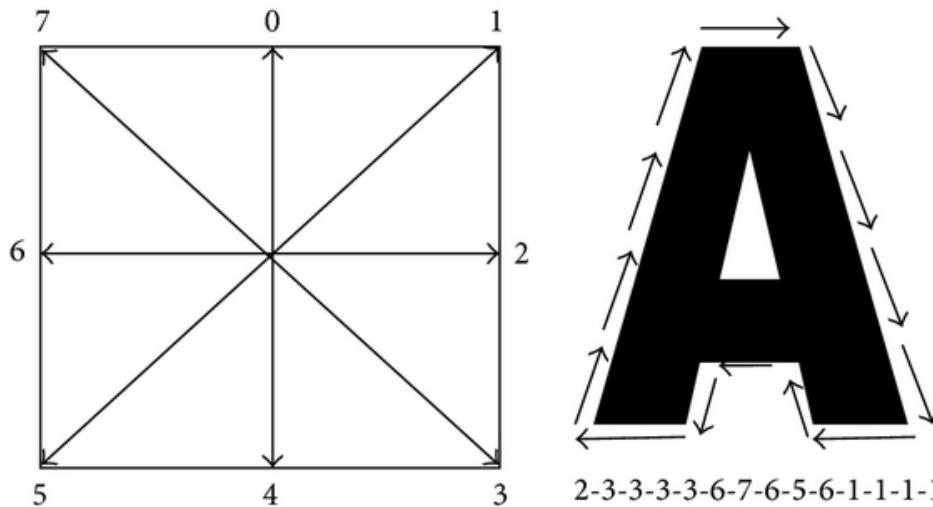
## ■ Trăsături bazate pe regiuni

- Se poate utiliza o variantă subțiată (*thinned*) a caracterului
- Direcțiile pe care le urmărește această variantă se codifică prin valori numerice
  - (de ex direcția orizontală = 4, direcția verticală = 2, bifurcație = 9 etc.)



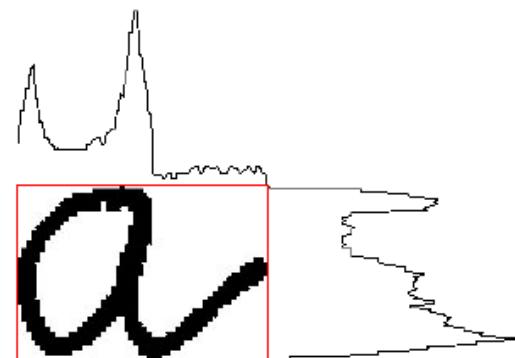
## ■ Trăsături bazate pe regiuni

- Se urmăresc direcțiile aproximative pe care le urmează conturul caracterului
  - *Chain coding*



## ■ Trăsături bazate pe proiecții

- Imaginele 2D se pot reprezenta prin intermediul unor semnale 1D, care sunt
  - invariante la zgomot
  - invariante la transformări de translație și scalare
  - afectate de rotații
- Semnalele se pot obține calculând histogramele de proiecție
  - numerele de pixeli de pe fiecare rând și coloană)



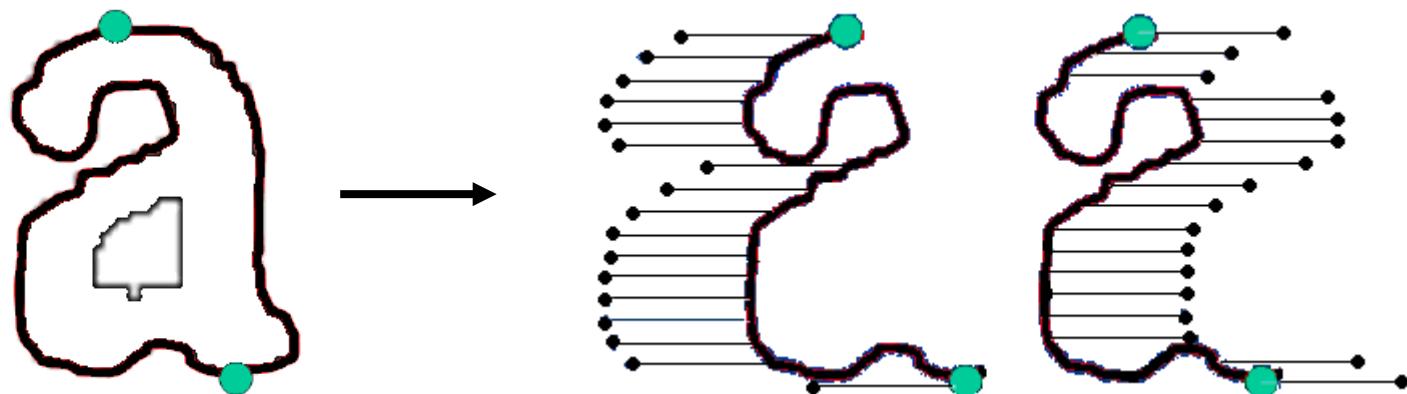
## ■ Trăsături bazate pe profiluri

- Se calculează distanțele dintre muchiile dreptunghiului care încadrează caracterul și cel mai apropiat pixel care aparține caracterului
- Aceste trăsături descriu forma și geometria exterioară a caracterului

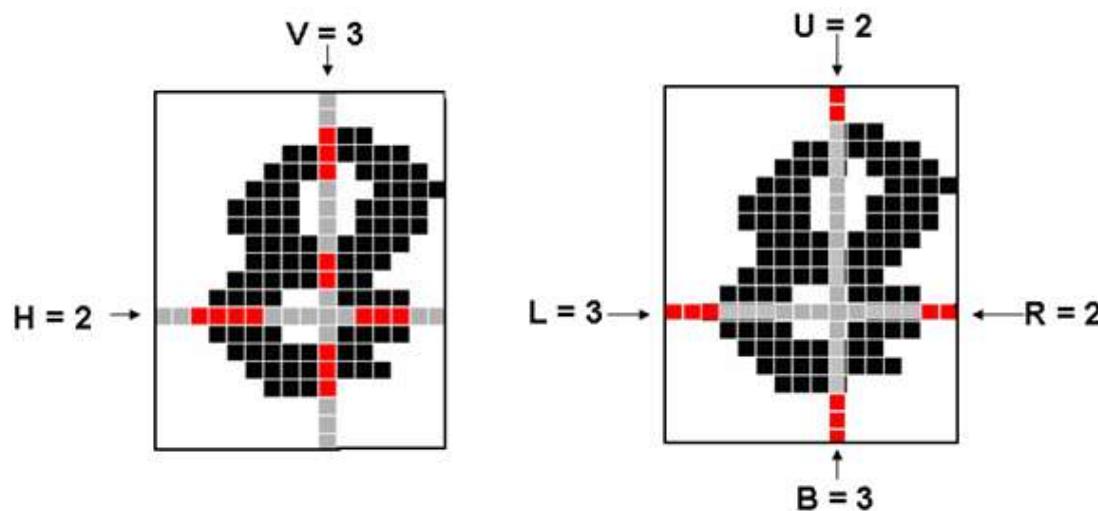


## ■ Trăsături bazate pe profiluri

- Se poate determina și un profil al conturului
  - Se estimează conturul caracterului
  - Se determină punctele din extremități
  - Se determină profilul conturului între cele două extremități

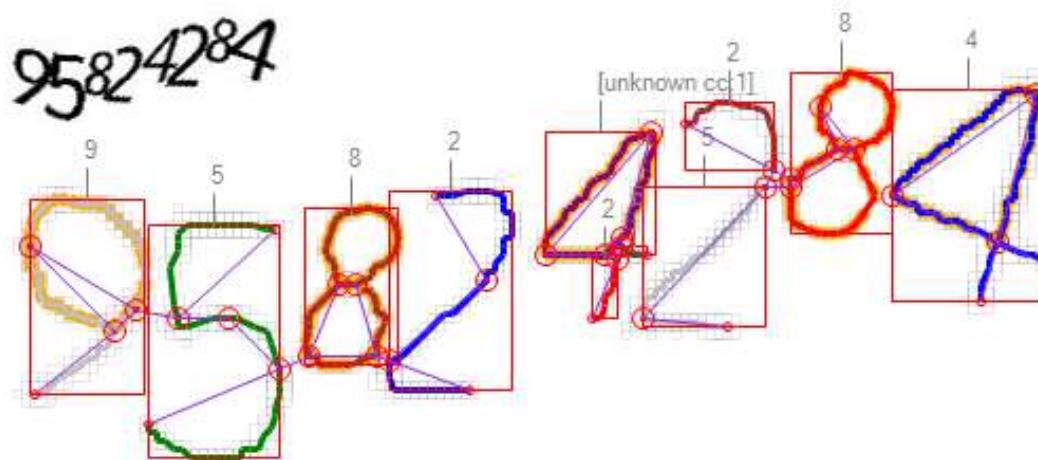


- Trăsături bazate pe tranziții
  - Se determină numărul de tranziții de la pixelii ce aparțin caracterului la pixelii din background
  - Vectorul astfel obținut se poate completa cu distanțele de la extremitățile imaginii până la extremitățile caracterului



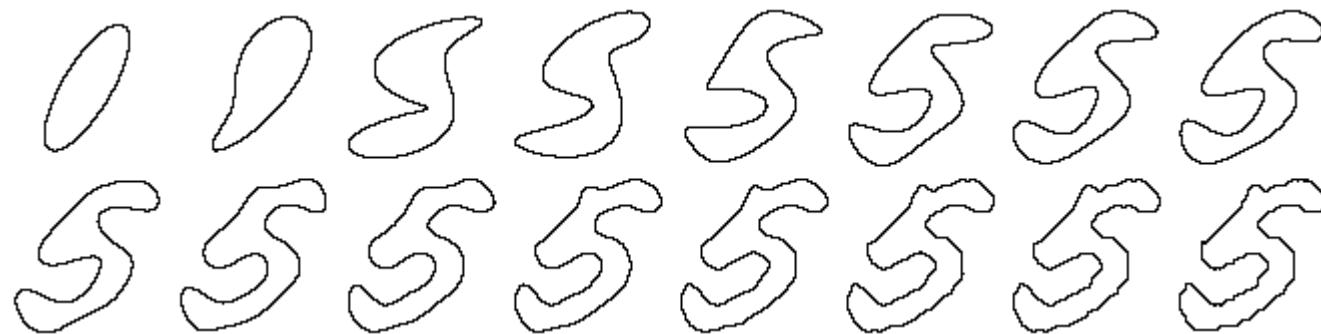
## ■ Trăsături structurale

- Se bazează pe proprietățile geometrice și topologice ale caracterului
  - gradul de curbură al conturului
  - buclele pe care le formează conturul
  - zonele de bifurcație
  - orientarea componentelor de tip linie
  - raportul lățime / lungime (*aspect ratio*) etc.



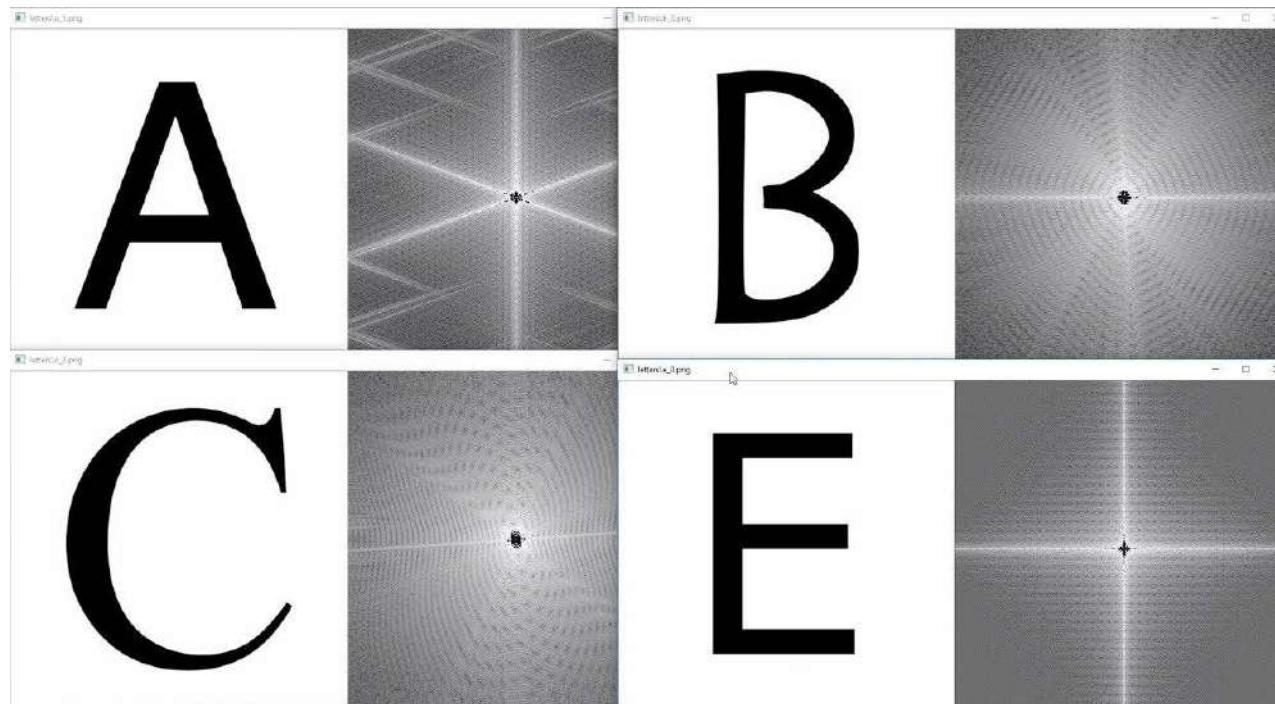
## ■ Transformări globale

- Valorile vectorilor de trăsături se deduc din întreaga imagine
- Exemplu:
  - Se aplică transformata Fourier pe conturul caracterului
  - Conturul se poate reconstitui pornind de la o parte din coeficienții transformatei
  - Așadar, vectorul de trăsături poate fi compus din acești coeficienți



## ■ Transformări globale

- Trăsături extrase prin prelucrarea în domeniul frecvență a imaginilor
- Se exploatează diferențele dintre spectrele imaginilor care conțin caractere diferite



## ■ Clasificare

- Presupune stabilirea categoriei din care fac parte elementele din imagine (caracterele sau cuvintele corespunzătoare) pe baza vectorului de trăsături
- Numeroase abordări, funcție de necesități:
  - K-Nearest Neighbor (KNN)
  - Clasificatori Bayes
  - Support Vector Machines (SVM)
  - Rețele neuronale etc.

## ■ Exemplu – clasificare folosind KNN

- Un set de date de antrenare

A 10x10 grid of handwritten digits from 0 to 9, used as training data for KNN classification. The digits are arranged in a 10x10 pattern:

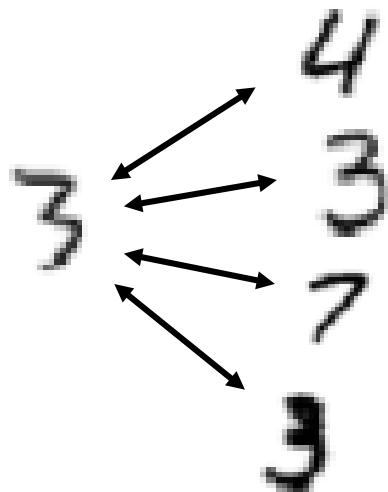
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9

- Clasele în care se încadrează (se cunoaște cifra corespunzătoare fiecărei imagini din setul de antrenare)

$$\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

## ■ Exemplu – clasificare folosind KNN

- Este nevoie de o funcție distanță care să evaluateze similaritatea dintre două imagini

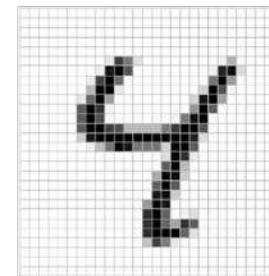


Care dintre imaginile din dreapta este cea mai apropiată de (seamănă cel mai mult cu) cea din stânga?

## ■ Exemplu – clasificare folosind KNN

- Pentru fiecare imagine se deduce un vector de trăsături
- $v : R^{wxh} \rightarrow R^d, v = (v_1, v_2, \dots, v_d)$
- În cel mai simplu caz, se poate considera ca vector de trăsături totalitatea valorilor pixelilor din imagine:

- $v : R^{28x28} \rightarrow R^{784}$

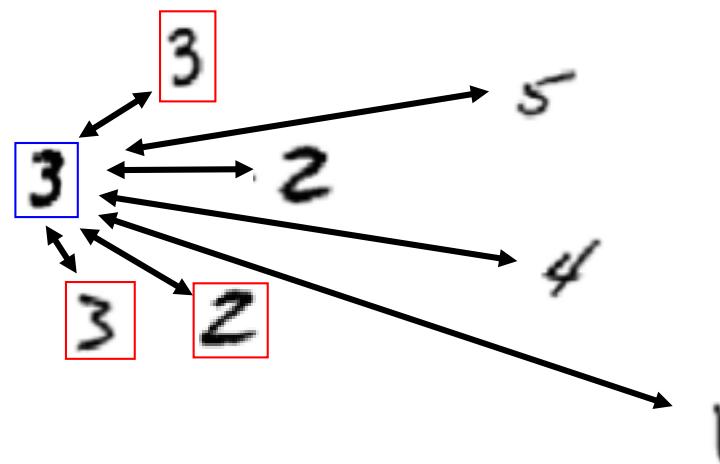


## ■ Exemplu – clasificare folosind KNN

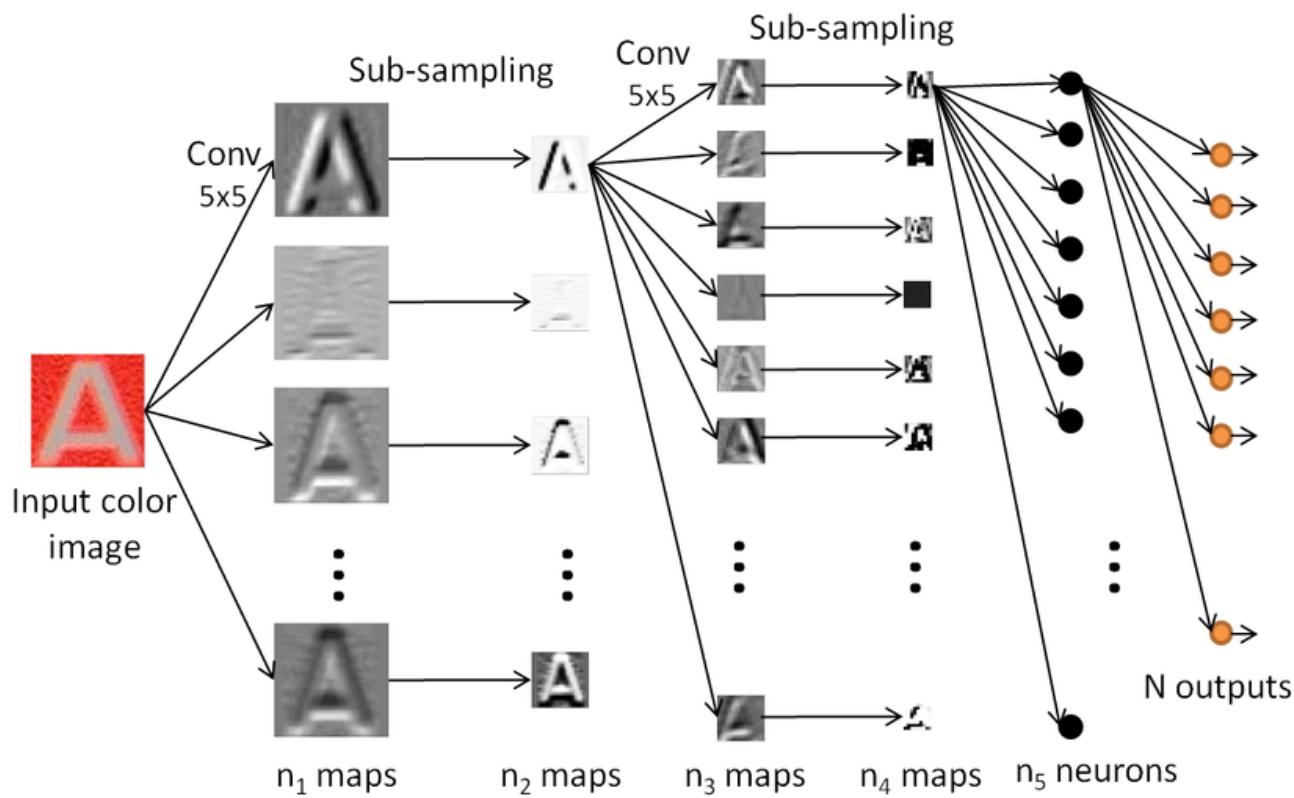
- Ca funcție distanță se poate folosi distanța euclidiană dintre componentele vectorilor de trăsături ale imaginilor
- $f_D(u, v) = \sqrt{\sum_{i=1}^d u_i - v_i}$
- Fiind dată o nouă imagine: 
- Se calculează distanța de la aceasta până la fiecare dintre imaginile din setul de antrenare
- Imaginea se va încadra în clasa celor mai apropiate  $k$  imagini din setul de antrenare

## ■ Exemplu – clasificare folosind KNN

- Presupunem  $k = 3$
- Cele 3 imagini care se apropie cel mai mult (ca similaritate) de cea necunoscută au clasele “3”, “3”, “2”
- Prin urmare, imaginea necunoscută va fi încadrată în clasa “3”



- Exemplu – clasificare folosind rețelele neuronale convecționale (CNN)



# Învățare automată

## 13. Rețele neuronale recurente

**Marius Gavrilescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

- “Recurrent Neural Network” – abreviat RNN
- sunt rețele neuronale în cadrul cărora unele legături între neuroni sunt ciclice
- se utilizează în special la procesarea datelor care formează secvențe

- Aplicate cu precădere în domenii ce țin de procesarea și sinteza textului
  - NLP (Natural Language Processing)
    - Capacitatea unui sistem intelligent de a înțelege limbajul uman și de a comunica folosind același limbaj
  - Language modeling
    - Generarea unui model de limbaj – o distribuție aplicată unei secvențe de cuvinte
  - Machine translation
    - Realizarea de traduceri automate

## ■ Procesarea și generarea textului

- RNN generează automat fraze pe baza unor secvențe de cuvinte din lumea reală
- RNN învață sintaxa și gramatica corectă în faza de antrenare
  - În etapa de propagare înainte – se propagă o secvență de text prin RNN și rezultă o altă secvență, inițial incorectă
  - În etapa de proparage înapoi – se determină eroarea cauzată de diferența dintre secvența generată de RNN și cea dorită și se actualizează ponderile în mod corespunzător

## ■ Exemplu:

- Text generat automat de o RNN (următorul slide)
- Pentru antrenare s-a folosit un text provenit din operele lui Shakespeare
- RNN învață din secvențele primite în faza de antrenare și e capabilă să genereze un nou text ce conține o secvență similară de cuvinte
- Din succesiunile de cuvinte din faza de antrenare RNN învață elemente de gramatică, sintaxă etc.

PANDARUS:

Alas, I think he shall be come approached and the day  
When little rain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and  
my fair nues begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

## ■ Exemplu:

- În mod similar, se poate antrena o RNN astfel încât să învețe sintaxa unui limbaj de programare
- Antrenarea se realizează folosind secvențe de cod reale
- RNN generează cod sursă aproximativ corect sintactic, chiar dacă este lipsit de semnificație
- Se pot realiza programe care scriu singure cod

## ■ Cod XML

```
<page>
  <title>Paris</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2015-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Tourist</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT [[Eiffel]]</text>
  </revision>
</page>
```

## ■ Cod C

```
/*
 * Increment the size file of the new incorrect UI_FILTER group
information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    return segtable;
}
```

- Exemplu de text din faze intermediare de antrenare:

- Epoca 100:

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

- Epoca 500:

we counter. He stutn co des. His stanted out one ofler that concossions and was to gearang reay Jotrets and with fre colt oft paitt thin wall. Which das stimm

- Epoca 700:

Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.

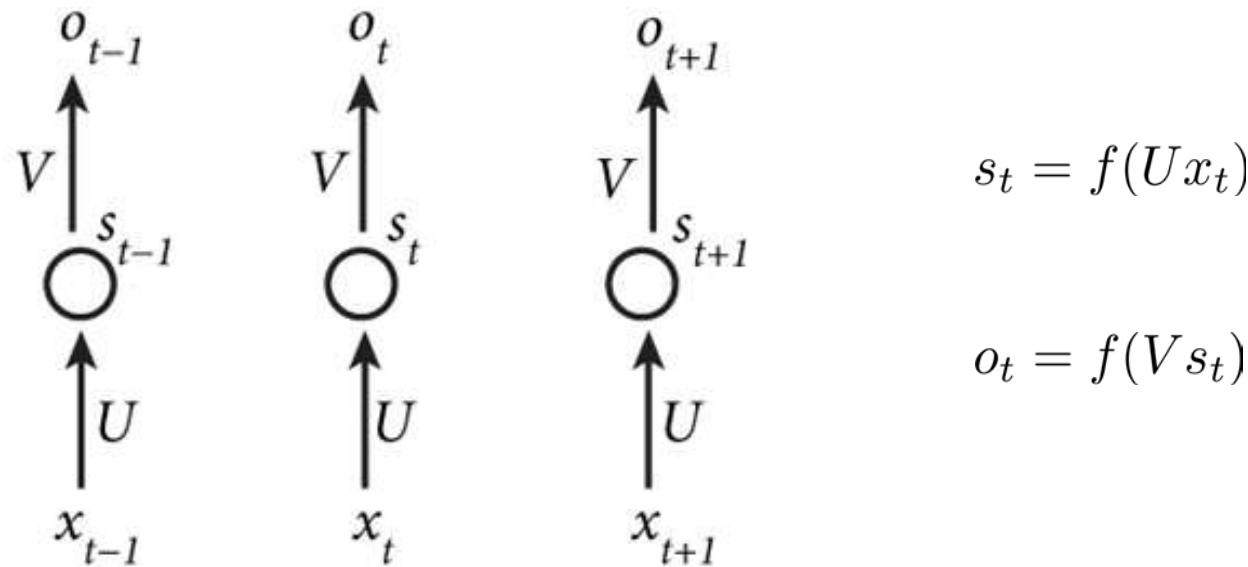
- Epoca 2000:

"Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him. Pierre aking his soul came to the packs and drove up his father-in-law women.

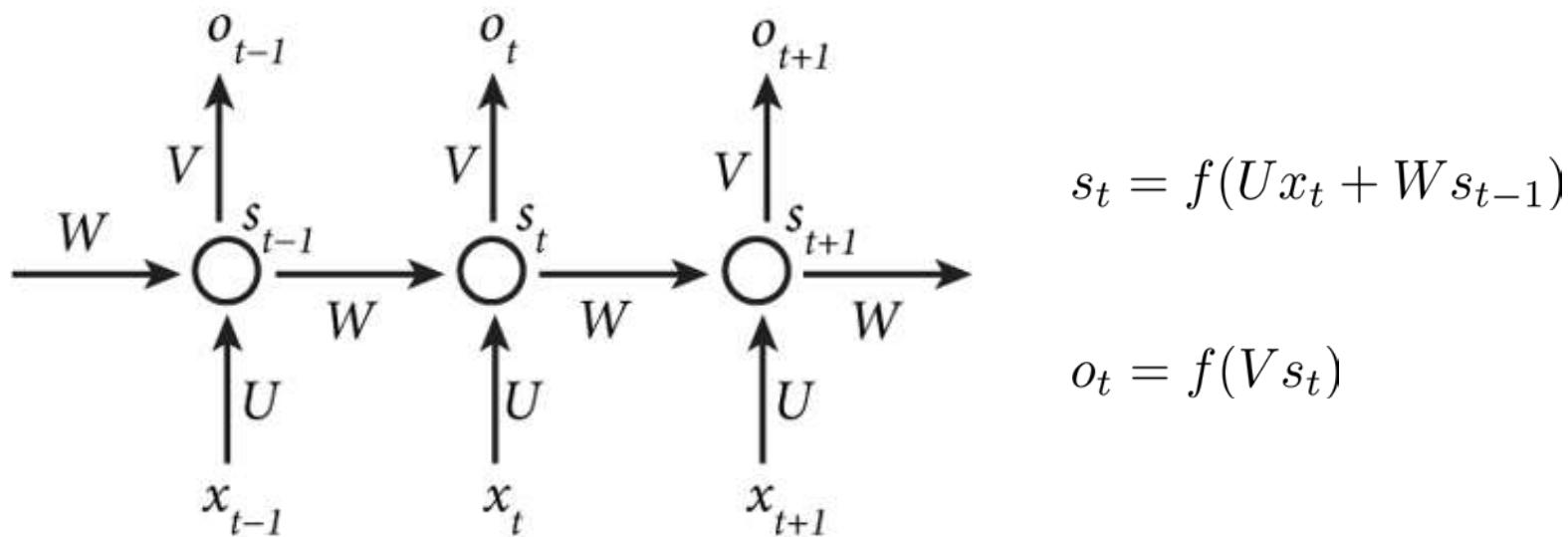
## ■ Prelucrarea secvențelor:

- Fie o secvență  $x_0, x_1, \dots, x_t, \dots$  de simboluri care pot fi
  - caractere
  - cuvinte
  - imagini
  - eșantioanele unui semnal
  - elemente de diverse tipuri

- Prelucrarea secvenței folosind rețele neuronale **non-recurente**:



- Prelucrarea secvenței folosind rețele neuronale recurente:



## ■ Principiile RNN:

- Pentru a realiza predicția unui element dintr-o sevență, trebuie cunoscute elementele anterioare
  - i.e. pentru a genera următorul cuvânt dintr-o frază, trebuie cunoscute cuvintele anterioare
- Prelucrarea elementului curent depinde de rezultatul prelucrării elementelor anterioare
- Într-o rețea neuronală clasică se presupune că intrările și ieșirile sunt independente
  - Datele de intrare de la epoca  $i$  nu conțin informații legate de datele de la epoca  $i-1$

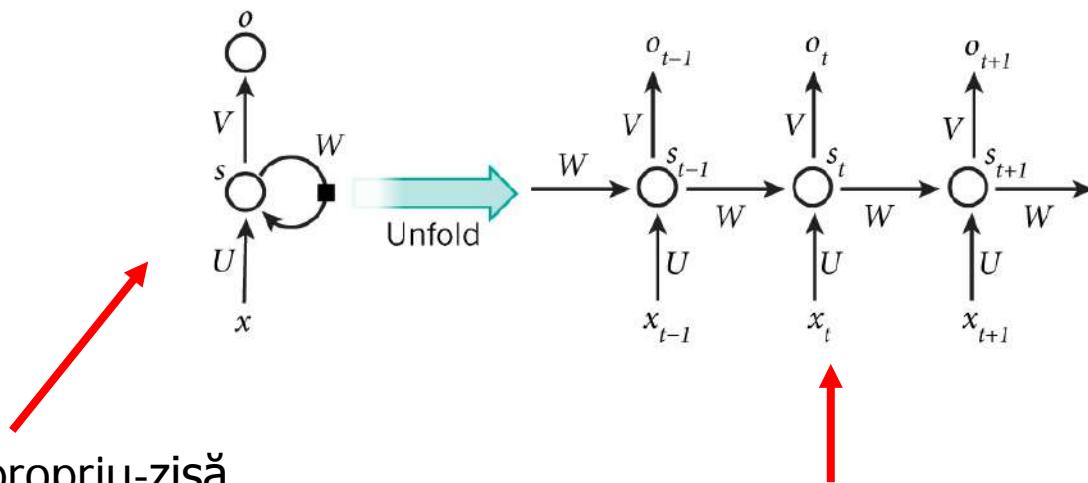
## ■ Principiile RNN:

- Rețelele neuronale clasice sunt limitate din punctul de vedere al prelucrării secvențelor:
  - Pentru a prelucra o secvență, ea ar trebui furnizată la intrarea rețelei în întregime, la fiecare iterare
    - imposibil / inefficient pentru secvențe de text de mari dimensiuni
  - Într-o etapă de antrenare nu se știe ce s-a realizat anterior
    - rețelele neuronale clasice nu au “memorie”, rezultatele nu persistă de la o iterare la alta

## ■ Principiile RNN:

- Într-o RNN, se realizează aceleasi operații pentru fiecare element din secvență
- Valorile de la ieșire depind de valorile obținute la iterațiile anterioare
- RNN au “memorie” – la iterația  $i$  se rețin informații legate de calculele de la iterațiile  $1..i-1$

## ■ Structura unei RNN simple:

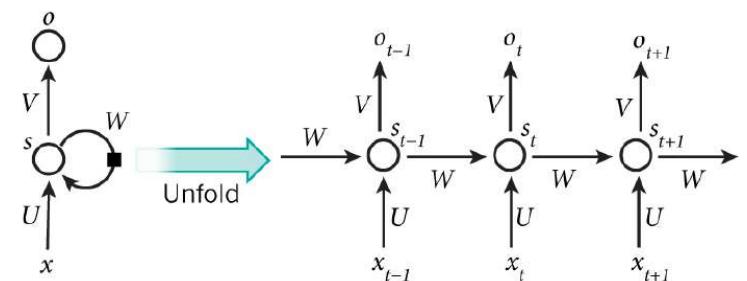


RNN propriu-zisă,  
Conține o legătură  
ciclică în stratul  
ascuns

Reprezentarea “desfășurată” a RNN  
Fiecare strat reprezintă câte o  
iterație, cu aceleasi ponderi

## ■ Structura unei RNN simple:

- $x_t$  = valorile de intrare de la momentul t
- $s_t$  = valorile din stratul ascuns la momentul t
- $o_t$  = ieșirea rețelei la momentul t
- Valorile din stratul ascuns la momentul t depind de
  - valorile de intrare de la momentul t
  - valorile de la ieșirea stratului ascuns de la momentul t-1



## ■ Structura unei RNN simple:

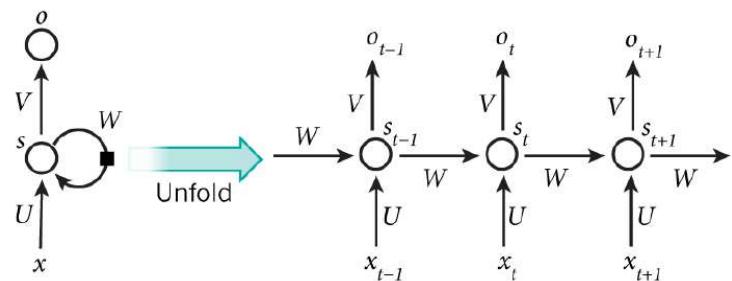
$$s_t = f(Ux_t + Ws_{t-1})$$

ieșirea stratului ascuns la momentul t

funcția de activare  
(tanh, ReLU etc.)

ponderile intrărilor în stratul ascuns  
(aceleși pentru toate momentele de timp)

intrările în stratul ascuns la momentul t



ieșirile din stratul ascuns la momentul anterior t-1  
(intrări recurente în stratul ascuns la momentul t)

ponderile intrărilor  
recurente în stratul ascuns (aceleși pentru toate momentele de timp)

- Structura unei RNN simple:
  - la ieșirea rețelei se utilizează funcția de activare *softmax*
  - ieșirile  $o_t$  sunt un vector de probabilități
    - elemente subunitare a căror sumă este 1
  - de exemplu, dacă se dorește clasificarea cuvintelor dintr-un text:
    - intrările în rețea la momentele 1, 2, ..., t-1 sunt o secvență de cuvinte
    - ieșirea de la momentul t – ce cuvânt urmează?
    - vectorul de la ieșire – probabilități de apartenență a cuvântului următor la elementele dintr-un dicționar / vocabular etc.

## ■ Structura unei RNN simple:

- $s_t$  de la momentul t se calculează pe baza
  - informațiilor de intrare generate pornind de la intrările curente în rețea
  - informațiilor generate de stratul ascuns pornind de la intrările în rețea din momente anterioare
- așadar: stratul ascuns cu valorile  $s_t$  constituie “memoria” RNN
  - valorile curente s-au determinat folosind valori anterioare

- Structura unei RNN simple:
  - nu este obligatoriu ca RNN să furnizeze date la ieșire în fiecare etapă (la fiecare moment de timp)
  - uneori, contează doar ieșirea  $o_t$  de la momentul t
  - în acest caz,  $o_{t-1}$ ,  $o_{t-2}$ , ... sunt irelevante și nu se generează
  - cel mai important aspect al RNN este informația din stratul ascuns

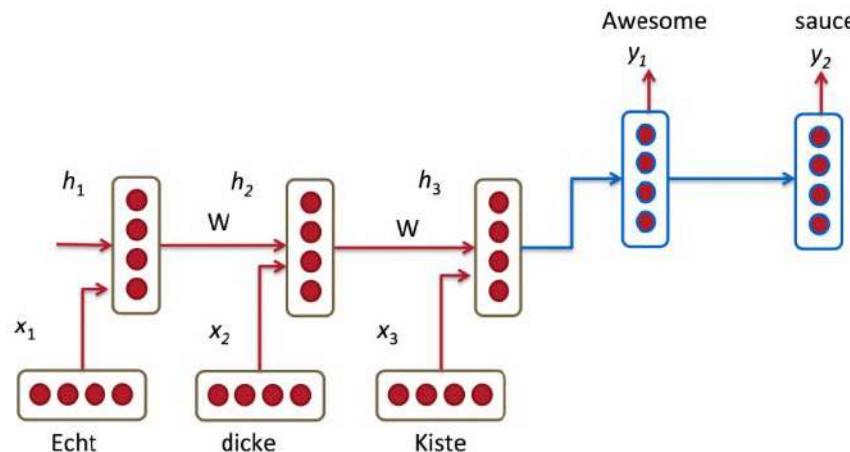
## ■ Aplicații ale RNN

- Realizarea de modele de limbaj și generarea de text
  - fiind dată o secvență de cuvinte, se poate prezice următorul cuvânt din secvență pe baza cuvintelor deja existente
  - un model de limbaj permite identificarea probabilității, sau a gradului de importanță, a unei anumite secvențe
    - util în aplicațiile care realizează traduceri automate
  - se pot construi modele generative
    - generare de text prin selecția probabilistică a textului de la ieșirea unei RNN

## ■ Aplicații ale RNN

### ■ Traducere automată

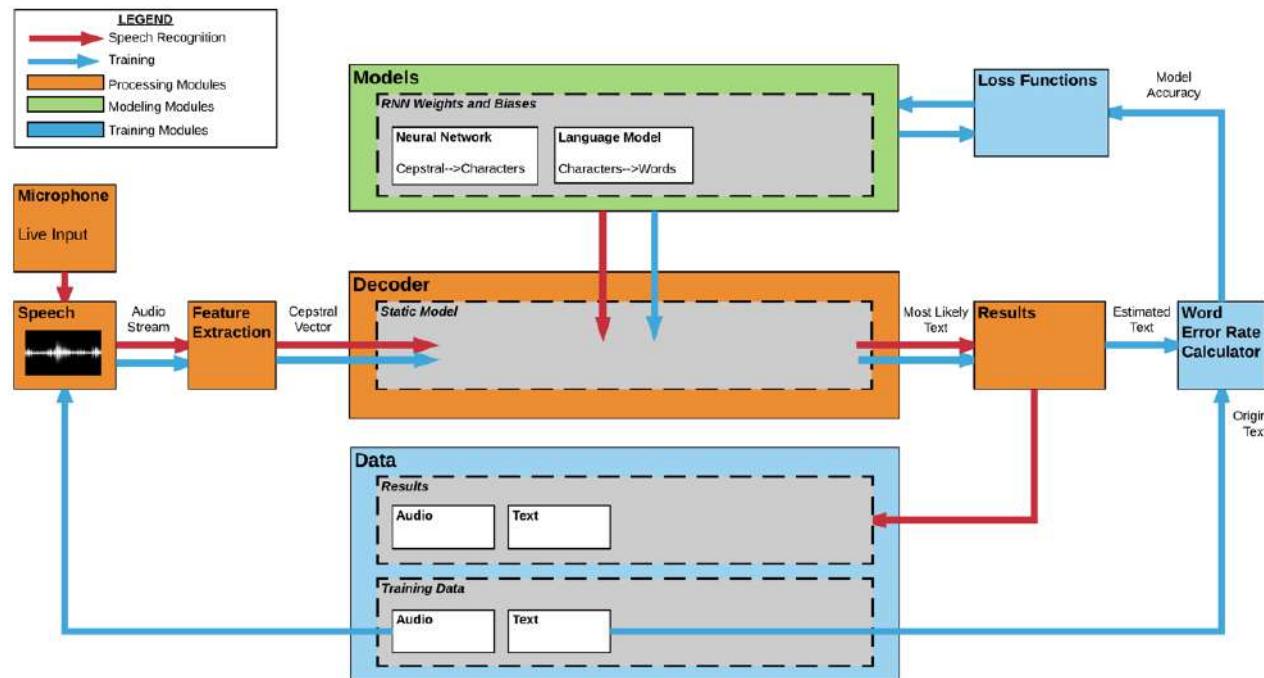
- intrarea este o secvență de cuvinte în limba sursă
- ieșirea este o secvență de cuvinte în limba destinație
- datele de ieșire se furnizează după prelucrarea întregii secvențe de la intrare
  - primul cuvânt din secvența tradusă poate depinde de informații preluate din întreaga secvență-sursă



# ■ Aplicații ale RNN

## ■ Recunoaștere vocală

- intrarea este o secvență de semnale acustice
- ieșirea este un vector de probabilități ale unor secvențe fonetice dintr-un vocabular audio cunoscut, sau o secvență de text ce corespunde sunetelor



## ■ Aplicații ale RNN

### ■ generarea automată de descrieri ale imaginilor

- se antrenează RNN cu un set de imagini la intrare și descrierile lor la ieșire
- rezultă o RNN capabilă să genereze descrieri ale altor imagini cu un conținut relativ similar



a group of people standing around a room with remotes



a young boy is holding a baseball bat



a cow is standing in the middle of a street

## ■ Antrenarea RNN

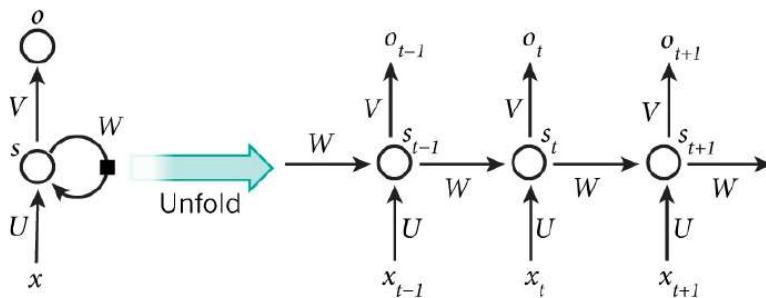
- metoda este similară cu cea de antrenare a rețelelor neuronale clasice
- se furnizează la intrarea rețelei un set de date, și la ieșirea rețelei rezultatul dorit pentru acele date
- de exemplu, în cazul prelucrării secvențelor de cuvinte
  - se furnizează la intrare un cuvânt dintr-o secvență
  - la ieșire se alege dintr-un vocabular cuvântul care ar trebui să urmeze în acea secvență (ieșirea dorită)
- se propagă înainte cuvântul de intrare și se obține ieșirea efectivă
- se calculează eroarea pe baza ieșirii dorite și a ieșirii efective
- se propagă eroarea înapoi prin rețea și se ajustează ponderile în sensul minimizării erorii

## ■ Antrenarea RNN

- diferența față de alte tipuri de rețele neuronale:
  - eroarea depinde de intrarea curentă, dar și de datele de intrare anterioare
  - algoritmul de propagare înapoi utilizat în alte cazuri nu este adecvat
  - se utilizează o variațiune a acestuia numită *propagarea înapoi în timp* (**BPTT – Back Propagation Through Time**)

## ■ Back Propagation Through Time

- considerăm rețeaua reprezentată anterior:



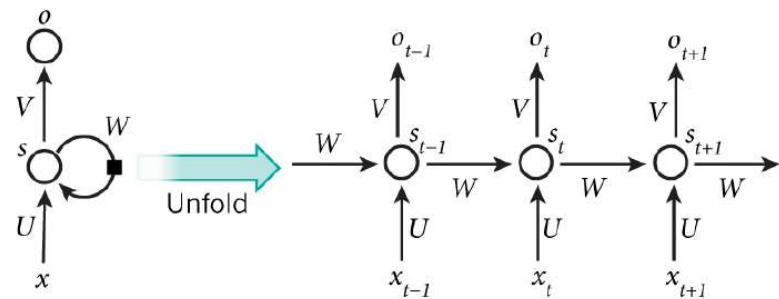
- ieșirile stratului ascuns se calculează astfel:

$$s_t = \tanh(Ux_t + Vs_{t-1})$$

- $\tanh$  este funcția de activare

## ■ Back Propagation Through Time

- considerăm rețeaua reprezentată anterior:



- ieșirile rețelei se calculează astfel:

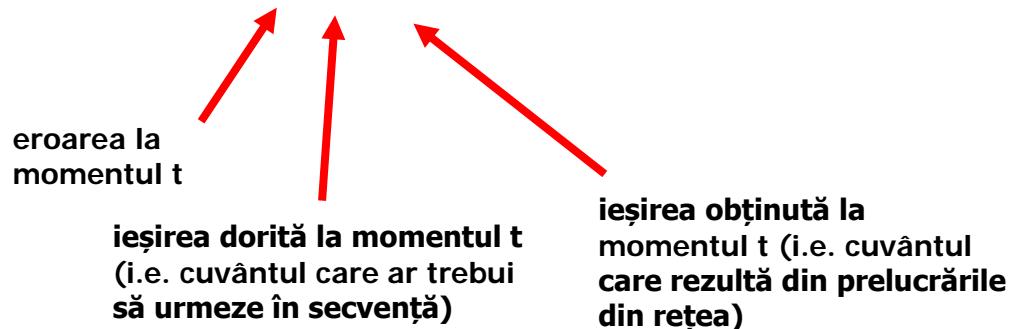
$$\hat{y}_t = \text{softmax}(Vs_t)$$

$\hat{y}_t$  are aceeași semnificație ca și  $o_t$

## ■ Back Propagation Through Time

- calculăm eroarea folosind funcția cross-entropy (alternativa este eroarea medie pătratică, MSE):

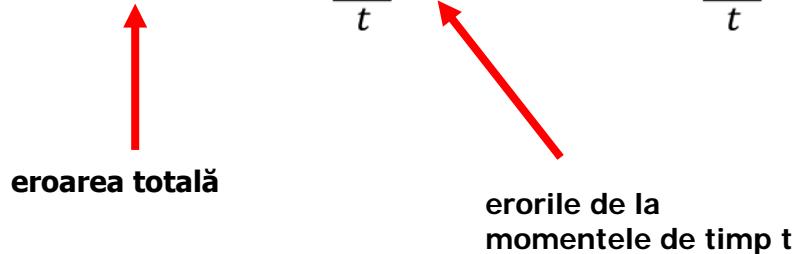
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$



## ■ Back Propagation Through Time

- rezultă eroarea totală, generată de întreaga secvență de antrenare:

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = - \sum_t y_t \log \hat{y}_t$$



## ■ Back Propagation Through Time

- antrenarea constă în
  - calculul gradienților erorii în raport cu ponderile U, V, W
  - actualizarea ponderilor folosind metoda gradienților descendenți
- cazul ponderilor V:

$$\frac{\partial E}{\partial V} = \sum_t \frac{\partial E_t}{\partial V}$$

eroarea totală de la ieșirea RNN (pentru întreaga secvență)

eroarea de la momentul t

## ■ Back Propagation Through Time

- cazul ponderilor V:

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial V}$$

se descompun  
derivatele parțiale

$$= \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial V} = \dots = (\hat{y}_t - y_t) \times s_t$$

calculele derivatelor  
parțiale ce rezultă

$$z_t = Vs_t$$

erori  
(ieșiri obținute – ieșiri dorite)

produs vectorial

ieșirile stratului ascuns

- Pentru ponderile U calculele sunt similare

## ■ Back Propagation Through Time

- cazul ponderilor W:

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \frac{\partial s_t}{\partial W}$$

- termenul  $s_t$  depinde de  $s_{t-1}$  care la rândul său depinde de  $s_{t-2}$ , etc. prin relația:

$$s_t = \tanh(Ux_t + Vs_{t-1})$$

- așadar gradientul se dezvoltă astfel:

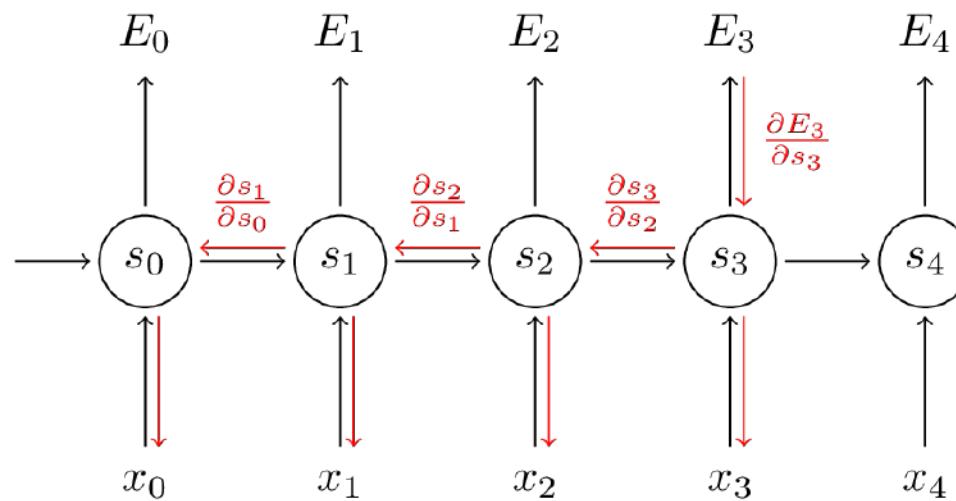
$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^t \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}$$

- prin urmare, gradientul de la momentul t depinde și de variațiile induse de stările de la momentele 0 .. t-1

## ■ Back Propagation Through Time

- cazul ponderilor W:

- variația stării de la momentul t a stratului ascuns rapport cu stările anterioare ale aceluiași strat se propagă înapoi în timp:



## ■ Limitări ale RNN simple

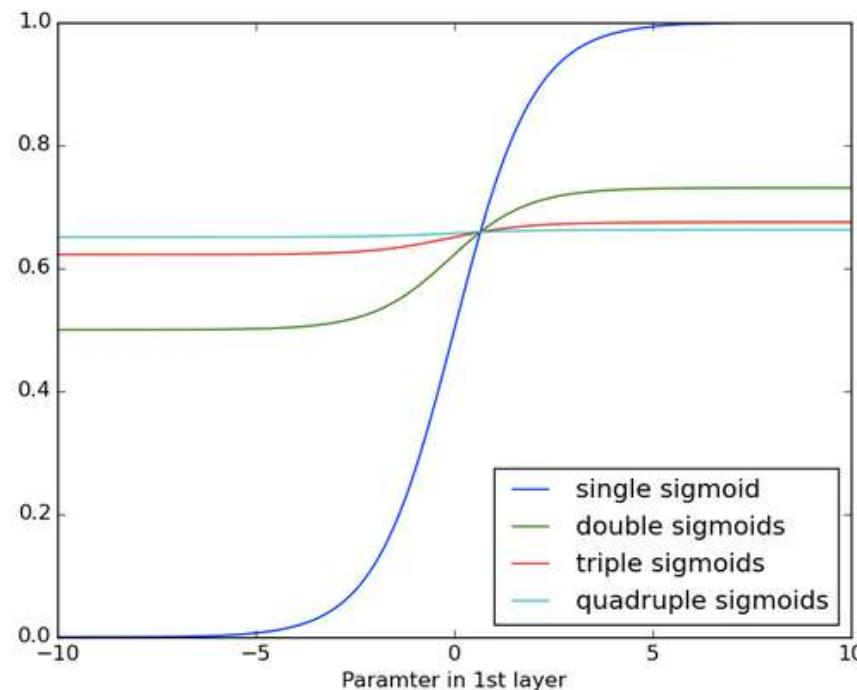
- RNN clasice funcționează corect în situațiile în care se analizează un număr limitat de elemente din secvență
- RNN pot analiza informația din trecut doar până într-un anumit punct
  - utilizează eficient doar informația relativ recentă
- RNN clasice devin incapabile de a învăța pornind de la informațiile din “trecutul îndepărtat”
- Problema este cauzată de modalitatea de calcul a gradienților

## ■ Limitări ale RNN simple

- Gradienții se calculează prin înmulțiri repetitive
  - straturile dintr-o rețea “comunică” prin
    - numeroase operații de înmulțire
    - funcții de activare
- În urma unui număr suficient de mare de operații de înmulțire este posibil ca
  - gradienții să capete valori foarte mari
    - se exagerează în mod eronat importanța anumitor ponderi
    - fenomenul se numește “**exploding gradient**”
  - gradienții să capete valori foarte mici
    - scade în mod eronat importanța anumitor ponderi
    - fenomenul se numește “**vanishing gradient**”

## ■ Limitări ale RNN simple

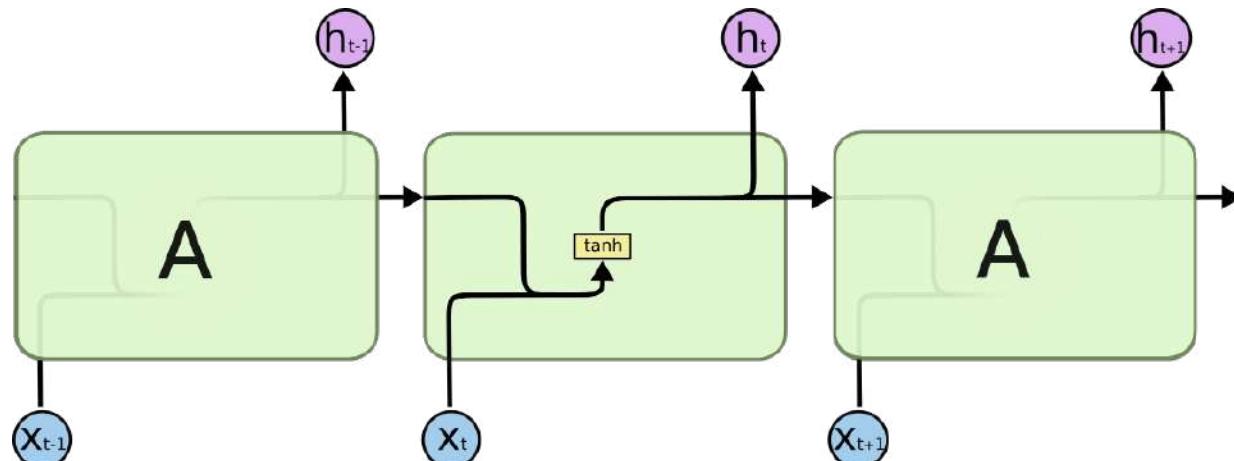
- Vanishing gradient: prin aplicarea repetată a funcției de activare sigmoid:
  - panta funcției devine din ce în ce mai mică
  - derivata rezultatului se diminuează (deoarece rezultatul variază într-un domeniu foarte limitat)



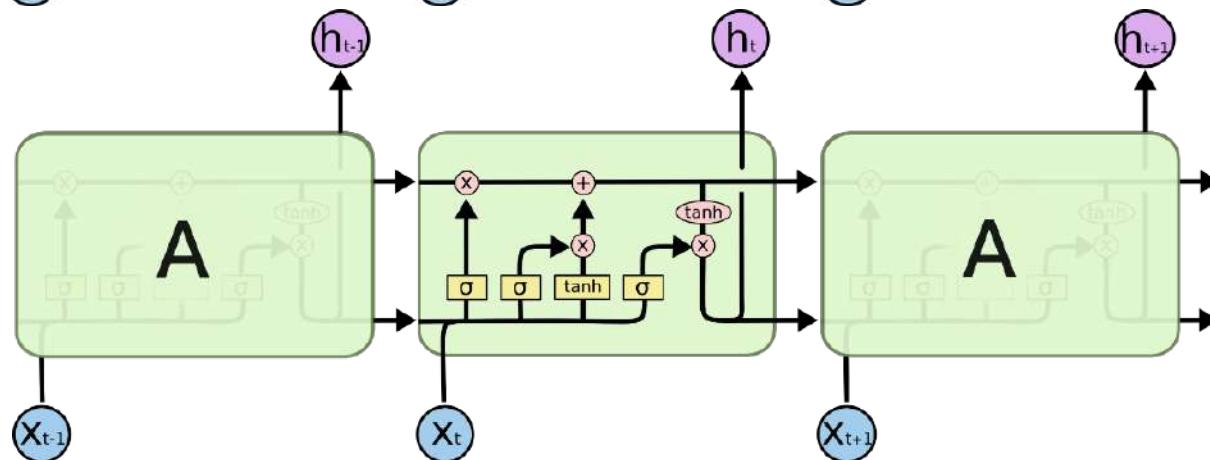
## ■ Soluția: Rețele de tip LSTM (Long Short-Term Memory)

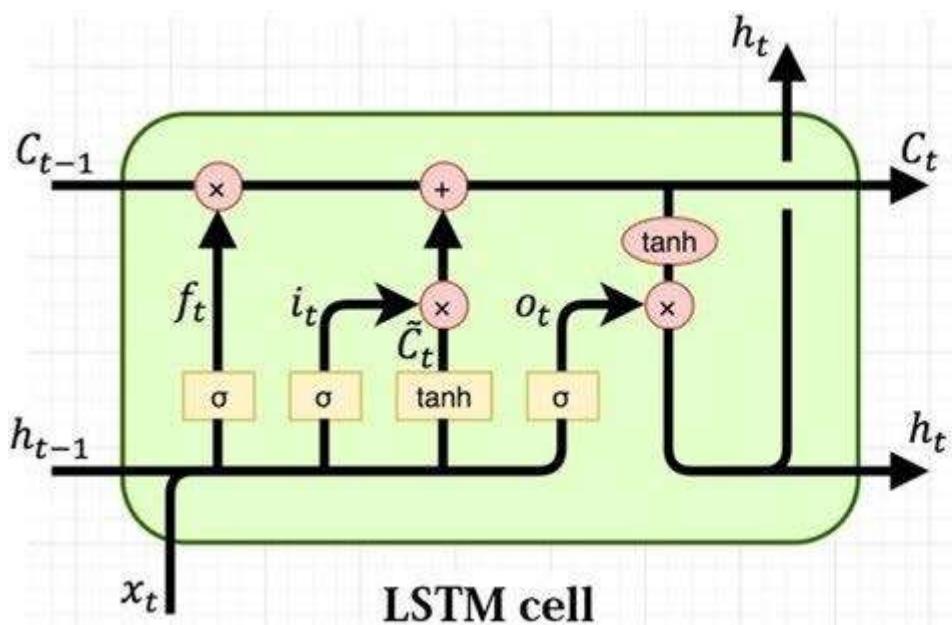
- Sunt rețele recurente în cadrul cărora
  - se încearcă menținerea valorilor erorilor care se propagă prin rețea la un nivel utilizabil
  - se permite învățarea folosind informații calculate cu un număr foarte mare de pași în urmă (care provin de la o iterare din trecutul îndepărtat)
- Principul de funcționare:
  - se controlează informația care circulă prin startul ascuns folosind straturi speciale numite porți (*gates*)

RNN clasică



LSTM





$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

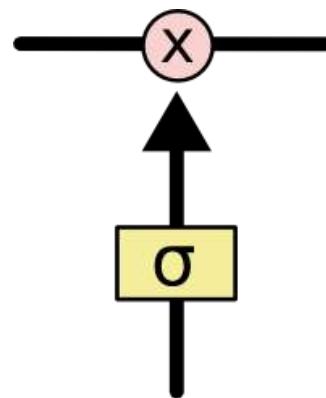
$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

$$h_t C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

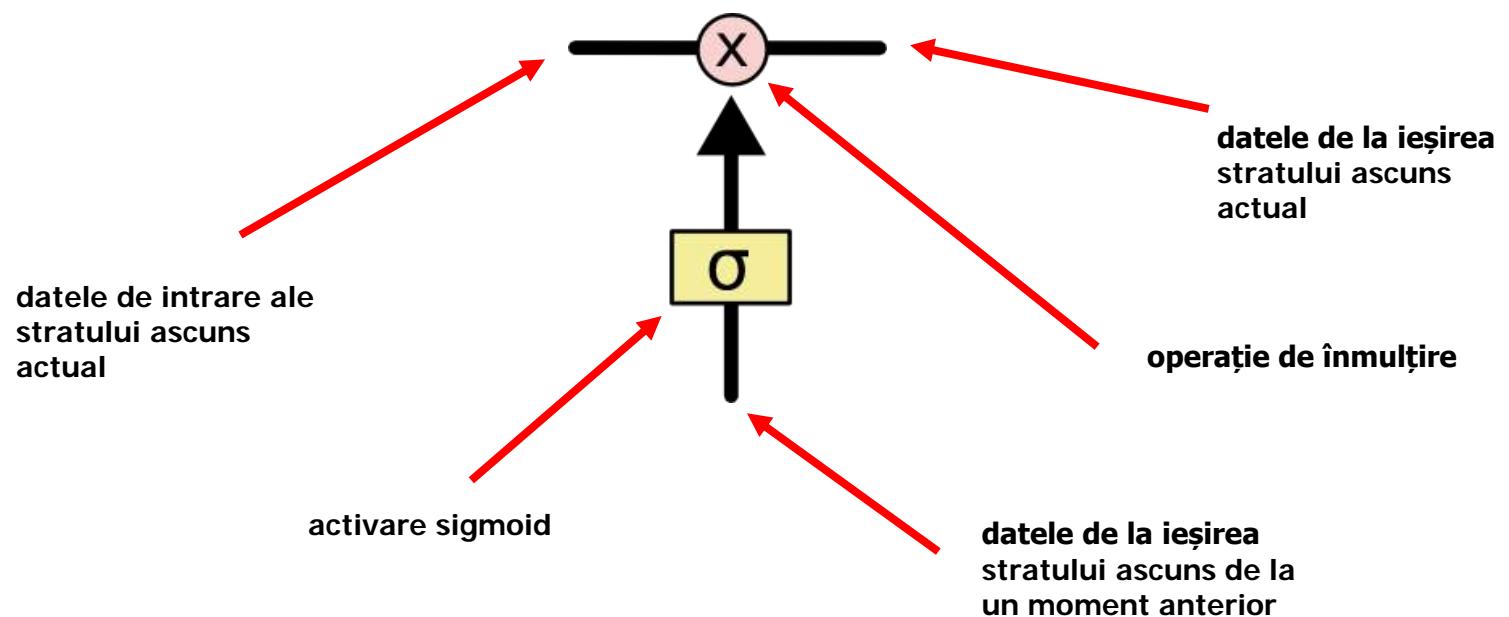
$$h_t = \tanh(C_t) * o_t$$

- LSTM:

- Poate să permită informației să circule prin rețea, sau să le inhibe:



- LSTM:
  - Poartile au capacitatea de a permite informației să circule prin rețea, sau de a o inhiba:

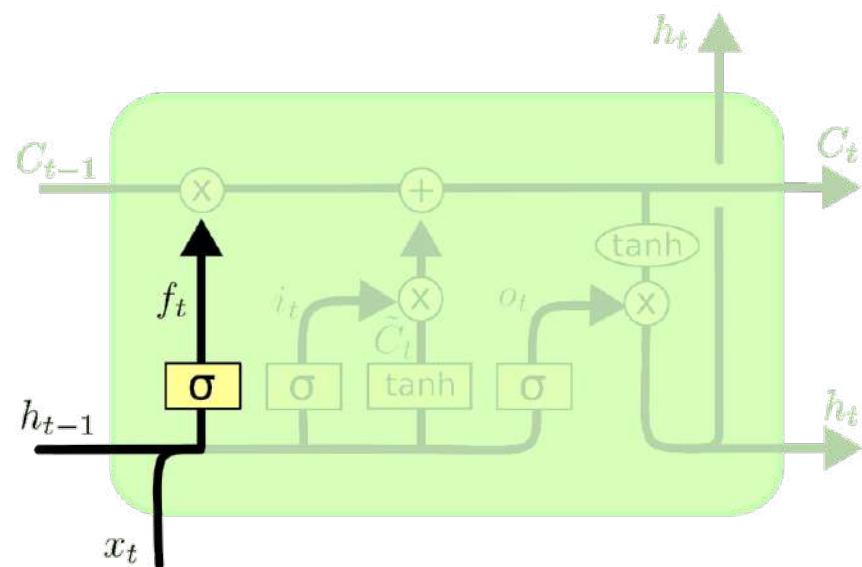


- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns

- **Poarta de uitare (*forget gate*)**

- pe parcursul operării rețelei, anumte informații pot deveni irelevante, și importanța lor trebuie diminuată
    - poarta de uitare primește la intrare aceeași informații ca și stratul ascuns
    - se calculează datele de ieșire ale porții, care are propriile ponderi
    - la ieșirea porții se aplică funcția sigmoid, rezultă valori din (0, 1)
    - acestea se înmulțesc cu datele de intrare în stratul ascuns
    - astfel, se controlează măsura în care aceste date se “memorează” sau se “uită”

- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns
  - Poarta de uitare (*forget gate*)

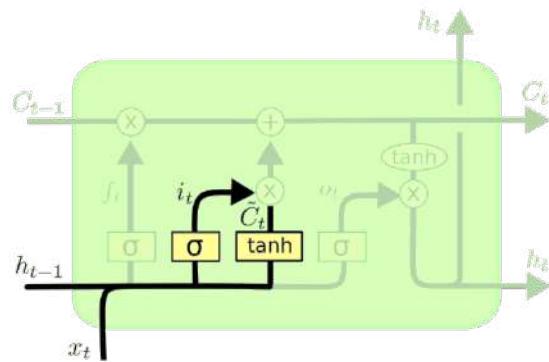


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns
  - Poarta de intrare (*input gate*)
    - controlează măsura în care informațiile nou-apărute se vor adăuga la starea stratului ascuns
    - poarta de intrare primește la intrare aceleași informații ca și stratul ascuns
    - se aplică o activare *sigmoid* pentru a decide care dintre valorile din stratul ascuns vor fi actualizate
    - se aplică o activare *tanh* pentru a crea un vector de “valori-candidat” - cele care se doresc încorporate în stratul ascuns
    - cele două rezultate anterioare (*sigmoid* și *tanh*) se înmulțesc
    - se obține un vector de valori care se adaugă la starea stratului ascuns

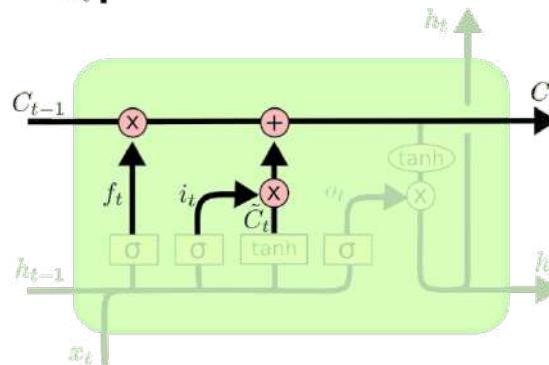
- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns

- Poarta de intrare (*input gate*)



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

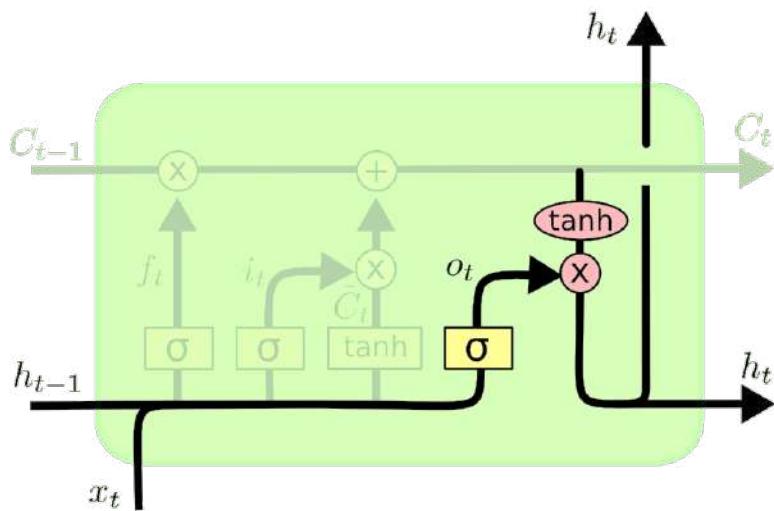
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns
  - Poarta de ieșire (*output gate*)
    - controlează măsura în care informațiile din stratul ascuns sunt furnizate la ieșirea acestuia
    - poarta de ieșire primește la intrare aceleași informații ca și stratul ascuns
    - se aplică o activare *sigmoid* pentru a decide care dintre valorile din stratul ascuns vor fi furnizate la ieșire
    - se înmulțesc valorile obținute (care sunt din  $(0, 1)$ ) cu rezultatul activării informațiilor din stratul ascuns
    - rezultatul obținut se furnizează la ieșirea stratului ascuns

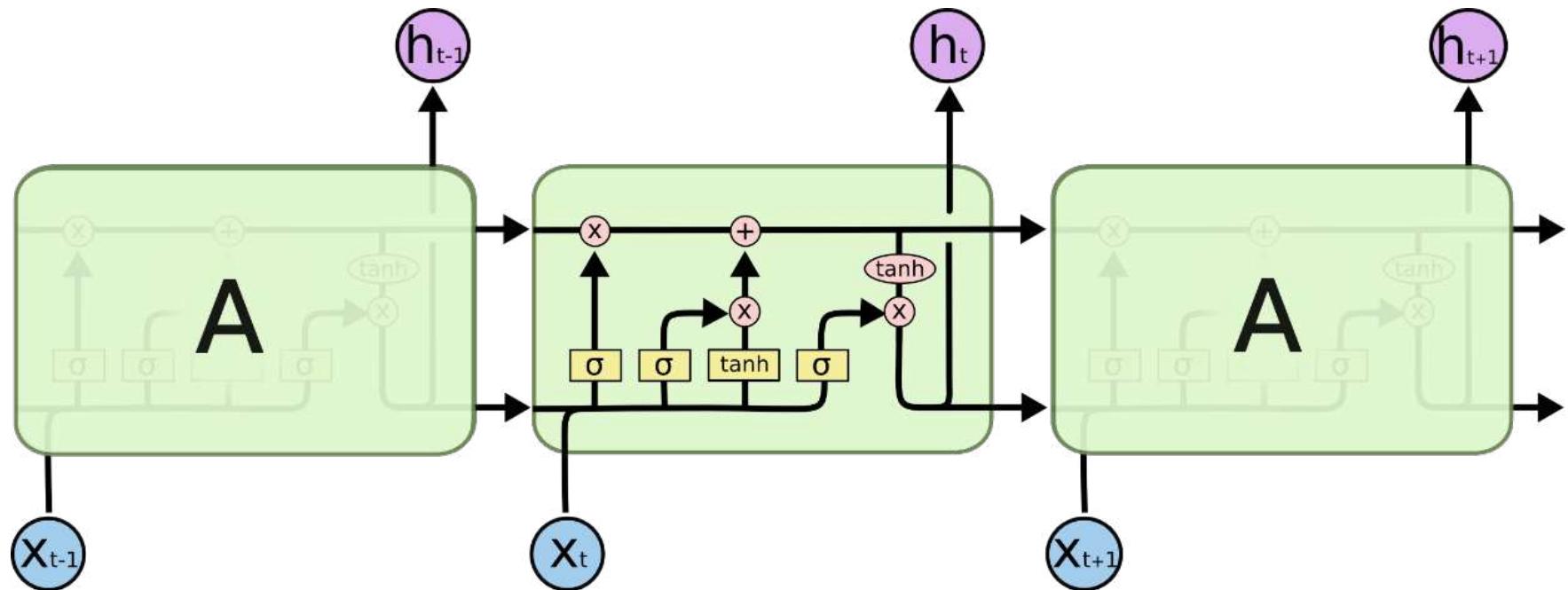
- LSTM – față de RNN clasică se adaugă trei porți pe lângă stratul ascuns
  - Poarta de ieșire (*output gate*)



$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$

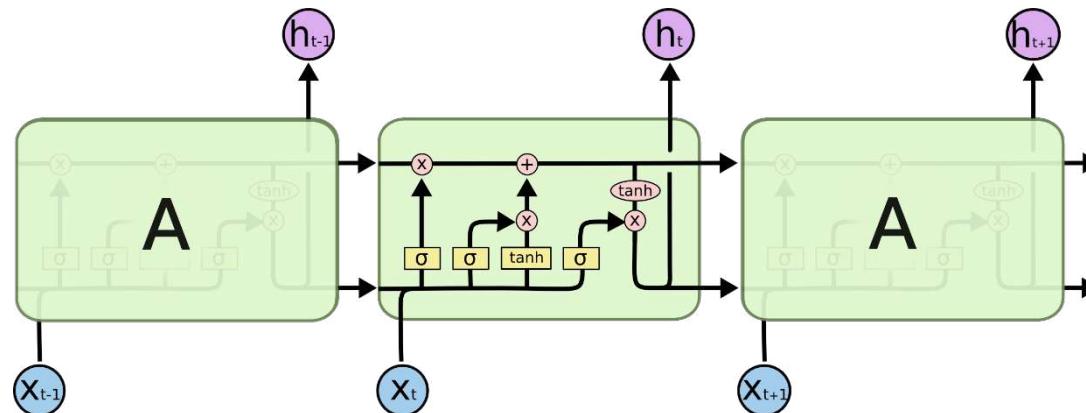
$$h_t = o_t * \tanh (C_t)$$

## ■ LSTM



## ■ LSTM

- informațiile din stratul ascuns sunt filtrate de cele trei porți la fiecare moment de timp
- operațiile aferente mențin variațiile din rețea la nivele utilizabile (gradenții se conservă de la o iterare la alta)
- se evită situația în care anumite informații sunt considerate fie mai importante, fie mai puțin importante decât ar trebui



- Un exemplu complet: **rețea neuronală LSTM pentru clasificarea secvențelor de cuvinte**
- Etape:
  - Stabilirea datelor de antrenare (o succesiune de cuvinte)
  - Stabilirea caracteristicilor rețelei
    - numărul de straturi
    - dimensiunea straturilor
    - lungimea secvenței din faza de antrenare
    - rata de învățare
    - numărul de epoci etc.
  - Antrenarea
    - propagarea unei secvențe de cuvinte
    - evaluarea diferenței dintre următorul cuvânt generat de rețea și următorul cuvânt din text
    - calculul gradienților, ajustarea ponderilor etc.
  - Utilizare:
    - furnizarea unei secvențe de cuvinte de start
    - rețeaua va genera propriul text pornind de la secvență

- Textul pentru antrenare:

on a mango tree in a jungle , there lived many birds . they were happy in their small nests . before the onset of the rainy season , all the animals of the jungle repaired their homes . the birds also made their homes more secure . many birds brought twigs and leaves and others wove their nests . we should also store some food for our children , chirped one of the birds . and they collected food , until they had enough to see them through the rainy season . they kept themselves busy preparing for the tough times . soon the rains came . it was followed by thunder and lighting . all the animals and birds stayed in their homes . it continued raining for many days . one day , a monkey went in the rain came into the forest . he sat on a branch , shivering with cold , water dripping from its body . the poor monkey tried his best to get shelter , but in vain . the leaves were not enough to save him from the rains . it is so cold , said the monkey . the birds were watching all this . they felt sorry for the monkey but there was little they could do for him . one of them said , brother , our small nests are not enough to give you shelter . another bird said , all of us prepared for the rainy season . if you had , you would not be in this situation . how dare you tell me what to do , said the monkey , growling at the bird . the monkey angrily pounced on the birds nest , tore it and threw it on the ground . the bird and her chicks were helpless . the poor bird thought , fools never value good advice . it is better not to advise them .

## ■ Generăm un **vocabulary**

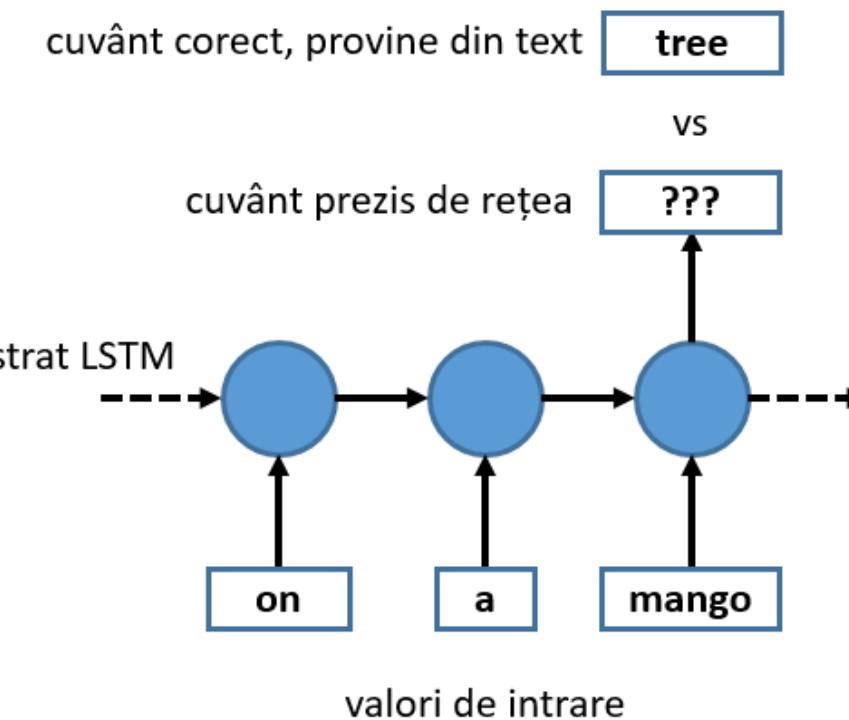
- totalitatea cuvintelor și simbolurilor care apar în text (în total, 149)
- rețeaua nu procesează direct siruri de caractere, ci prelucrează cuvintele pe baza **indecsilor** lor din vocabular
- string-urile propriu-zise se utilizează doar la afișarea rezultatelor

0: '.'  
1: 'the'  
2: ','  
3: 'birds'  
4: 'and'  
5: 'in'  
6: 'they'  
7: 'for'  
8: 'to'  
9: 'it'  
10: 'monkey'  
11: 'their'  
...  
143: 'never'  
144: 'value'  
145: 'good'  
146: 'advice'  
147: 'better'  
148: 'advise'

- Principiul de funcționare al rețelei
  - la intrare se furnizează o secvență de cuvinte
    - de fapt, o secvență de indecși din vocabular
  - rețeaua generează la ieșire un vector de probabilități de dimensiunea vocabularului
    - câte o probabilitate pentru fiecare cuvânt
    - fiecare element din vector = probabilitatea ca acel cuvânt să urmeze după secvența de la intrare
    - cuvântul următor = cel cu probabilitatea maximă
  - la intrare se furnizează secvența anterioară fără primul cuvânt, la care se adaugă noul cuvânt identificat de rețea
  - se determină un alt cuvânt pe baza noii secvențe
  - procesul se reia de câte ori se dorește, și rezultă un nou text (o “poveste” generată de rețea)

- Caracteristicile rețelei
  - strat de intrare – dimensiunea 3
    - dorim ca la intrare să furnizăm indecșii a trei cuvinte din vocabular
  - straturi ascunse – două straturi LSTM de dimensiune 512
  - strat de ieșire – dimensiunea 149 (numărul de cuvinte din vocabular)
    - cuvântul identificat de rețea = cel cu indexul valorii maxime de la ieșirea rețelei

- Antrenare



- Antrenare
  - Date de antrenare
    - teoretic: secvențe de trei cuvinte la intrare, cuvântul următor la ieșire
    - practic: indecșii cuvintelor dintr-o secvență de trei cuvinte la intrare, cuvântul următor la ieșire, în codificare *one-hot*
  - Se evaluează diferența (*loss*) dintre cuvântul prezis de rețea și cel corect
    - cross entropy între vectorii de probabilități de la ieșirea rețelei și codurile one-hot ale cuvintelor corecte
  - Se ajustează ponderile rețelei astfel încât să se minimizeze diferențele dintre valorile prezise de rețea și cele corecte
    - se folosește metoda de minimizare *RMSProp*

- Antrenare – exemplu de pe parcursul antrenării

```
Epoch: 1000, Average Loss: 4.655801, Average Accuracy: 5.80%
['all', 'the', 'animals'] - [and] vs [.]
```

```
Epoch: 5000, Average Loss: 2.479433, Average Accuracy: 38.30%
['angrily', 'pounced', 'on'] - [the] vs [do]
```

```
Epoch: 10000, Average Loss: 1.776092, Average Accuracy: 55.50%
['bird', 'said', ','] - [all] vs [all]
```

```
Epoch: 20000, Average Loss: 0.869596, Average Accuracy: 78.60%
[',', 'but', 'in'] - [vain] vs [vain]
```

```
Epoch: 30000, Average Loss: 0.748426, Average Accuracy: 82.40%
['enough', 'to', 'see'] - [them] vs [them]
```

```
Epoch: 40000, Average Loss: 0.628239, Average Accuracy: 85.90%
['mango', 'tree', 'in'] - [a] vs [the]
```

```
Epoch: 50000, Average Loss: 0.547765, Average Accuracy: 87.70%
['how', 'dare', 'you'] - [tell] vs [tell]
```

- Exploatarea rețelei antrenate

- Utilizatorul furnizează o secvență de pornire C0, C1, C2 (3 cuvinte)
- Secvența se furnizează la intrarea rețelei, care generează următorul cuvânt, C3
- Secvența C1, C2, C3 se furnizează la intrarea rețelei, care generează următorul cuvânt, C4
- Secvența C2, C3, C4, se furnizează la intrarea rețelei, rezultă următorul cuvânt, C5
- Procesul se reia, până când se generează un număr de cuvinte stabilit de utilizator

- Exemple de texte generate de retea:

rainy days in the jungle with the birds , and were happy in their small nests , before the onset of the rainy season . if bird was watching you the monkey , growling

in the jungle , before the rainy season . if you had thought , you would not be in this situation . how dare you tell me this , the poor bird thought for days

the monkey tried his best for the tough times . many birds brought twigs and in the jungle , there lived many birds . they were happy in their small nests , before the season

birds were watching all this . another bird had said , you would not be in this situation . a monkey said, how dare you tell me this . the poor bird thought of the rainy season

# Învățare automată

## 14. Modele Sequence-to-Sequence

**Marius Gavrilescu**

Universitatea Tehnică „Gheorghe Asachi” din Iași  
Facultatea de Automatică și Calculatoare

# Modele seq2seq

- **modele de clasificare, generative și de analiză a secvențelor**
- **principalul scop: conversia de text**
  - **traduceri automate**
- **se pot utiliza și pentru alte tipuri de date**
  - **transformarea unei secvențe într-o altă secvență echivalentă**
  - **aplicarea unor reguli dobândite prin învățare/antrenare**

# Modele seq2seq

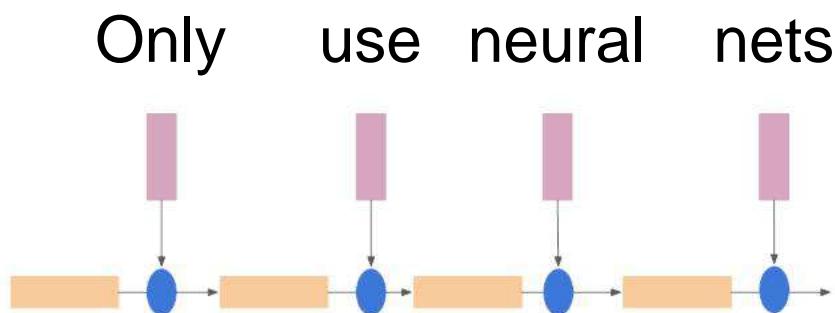
- **structura generală: 2 componente**
- **Encoder:**
  - rețea neuronală ce are ca principală componentă un strat recurrent
  - primește secvența de text care trebuie tradusă
  - determină o reprezentare a acesteia numită *context*
    - vector de valori numerice de dimensiune constantă

# Modele seq2seq

- **structura generală: 2 componente**
- **Decoder:**
  - **rețea neuronală de asemenea bazată pe un strat recurrent**
  - **preia contextul furnizat de encoder**
  - **generează secvența tradusă cuvânt-cu-cuvânt**

# Rețelele neuronale recurente (RNN)

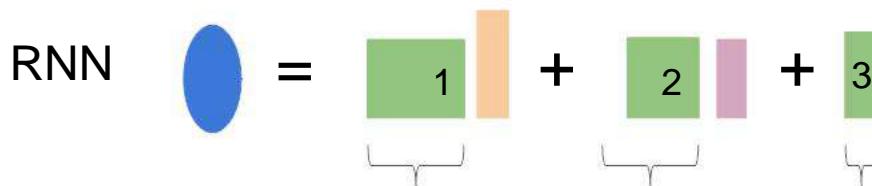
- aplicate frecvent pentru realizarea modelelor lingvistice



- Starea rețelei, vector ce conține *memoria* acesteia.
- Vector ce reprezintă codificarea cuvântului (embedding)
- Funcționalitatea RNN, se combină starea curentă cu vectorul corespunzător cuvântului curent pentru a genera o nouă stare

# Funcționalitatea RNN

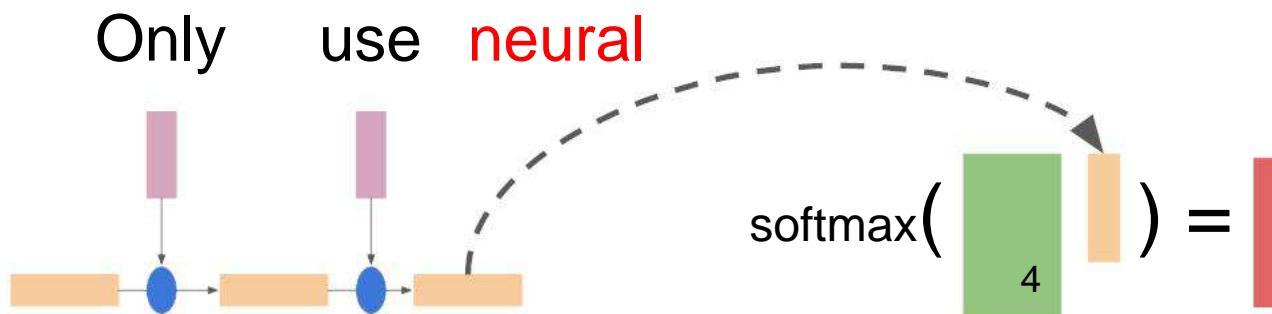
RNN preia starea curentă și reprezentarea unui cuvânt și generează o nouă stare prin care se codifică textul până în acel punct



Noua stare se generează prin combinarea ponderată a stării anterioare și a reprezentării cuvântului curent  
Valorile ponderilor se învață în urma antrenării

# Predictia: generarea cuvintelor din secvența de la ieșire

- se codifică în starea curentă a RNN textul până la cuvântul curent
- RNN generează valori care permit selecția următorului cuvânt dintr-un vocabular prestabilit

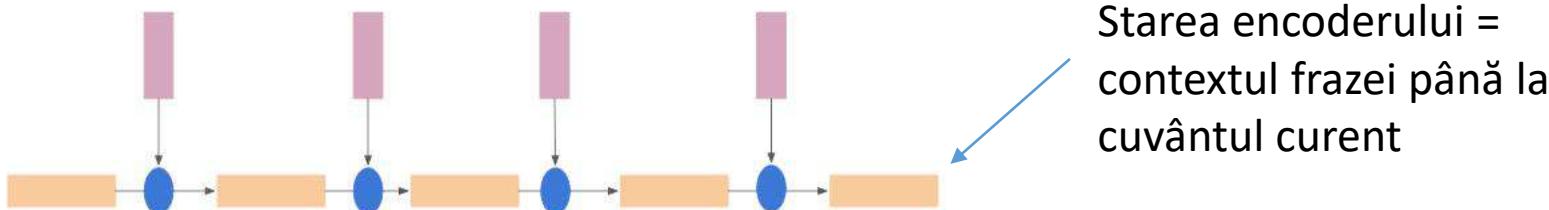


În stratul de ieșire, RNN generează o mulțime de valori, câte una pentru fiecare element al vocabularului  
Prin aplicarea funcției softmax în stratul de ieșire rezultă distribuție de probabilități (i.e. câte o probabilitate pentru fiecare cuvânt din vocabular)

# Determinarea contextului

- encoderul realizează reprezentarea frazei din limba inițială
  - generează **contextul** corespunzător frazei

Only use neural nets

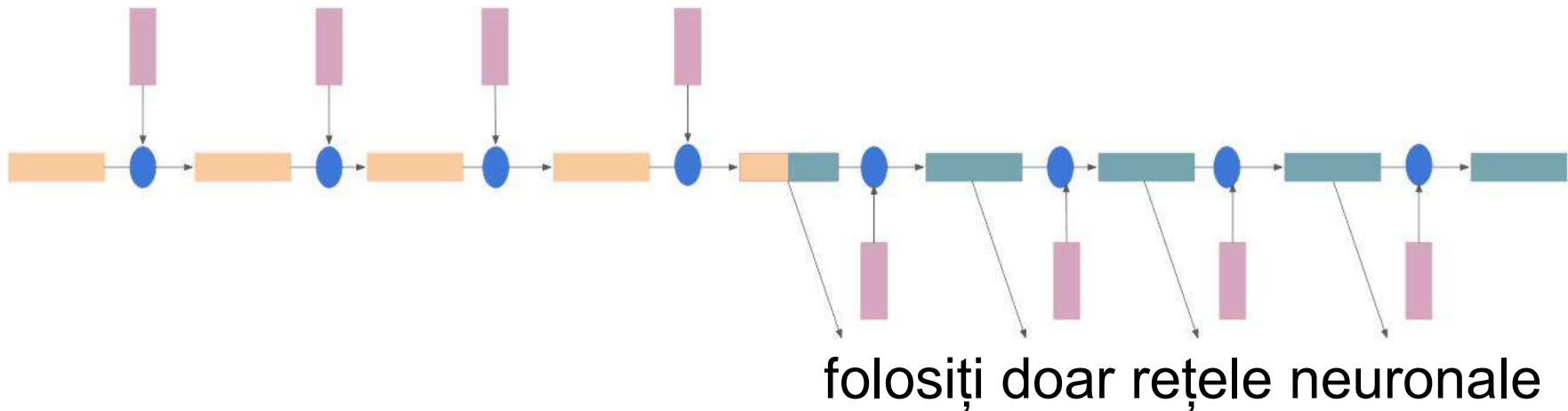


- la fiecare pas se determină contextul frazei până la cuvântul corespunzător
- de cele mai multe ori contextul este reprezentativ mai ales pentru ultimul cuvânt

# Generarea frazei de ieșire (a traducerii)

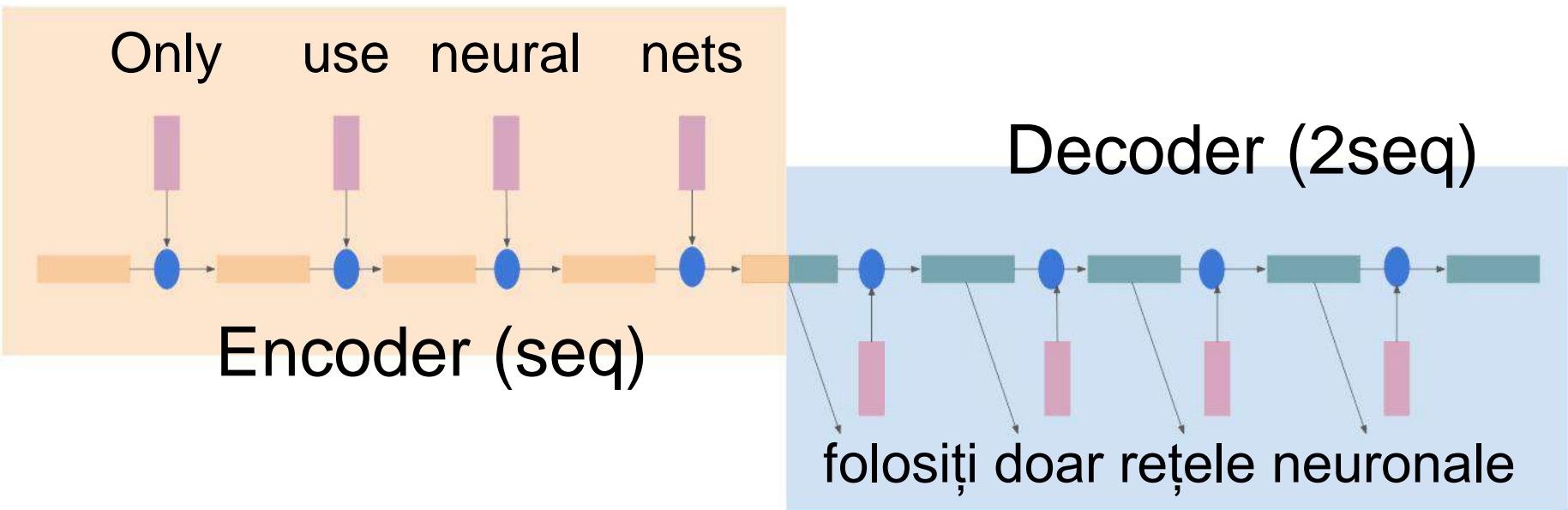
- se realizează cuvânt cu cuvânt de către decoder
- decoderul preia contextul generat de encoder

Only use neural nets



# Generarea frazei de ieșire (a traducerii)

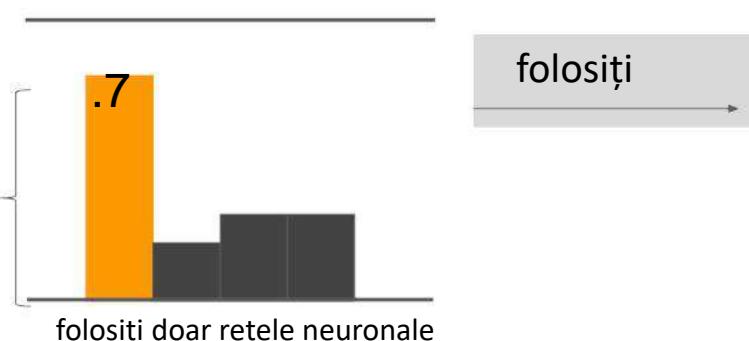
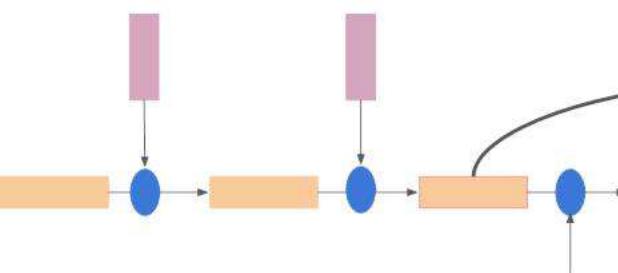
- se realizează cuvânt cu cuvânt de către decoder
- decoderul preia contextul generat de encoder



# Antrenarea

- minimizarea unei funcții eroare
- în general se utilizează funcția eroare cross entropy
- eroarea la generarea unui cuvânt:  
 $-\log(\text{probabilitatea determinată de rețea pentru cuvântul corect din vocabular})$

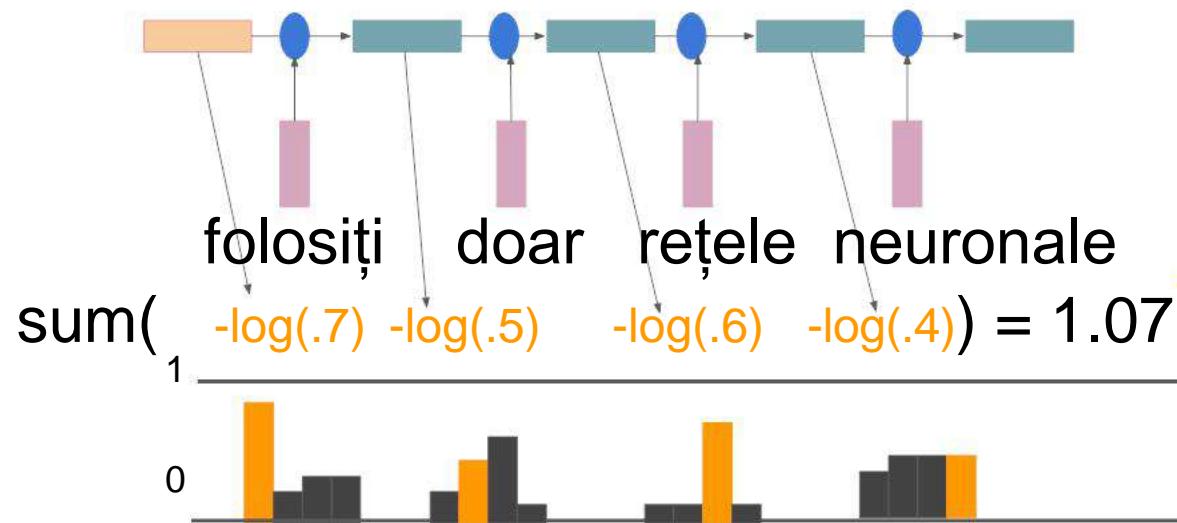
neural nets



Loss  
 $-\log(.7)$

# Antrenarea la nivelul frazei

- suma erorilor generate de fiecare cuvânt prezentat de rețea
- la eroare se adaugă un termen pozitiv ori de câte ori rețeaua prezice o probabilitate <1 pentru cuvântul corect din vocabular (i.e. pentru cuvântul care urmează în fraza tradusă)
- scopul antrenării – determinarea valorilor ponderilor care minimizează eroarea



# Starea RNN

$h_t$  = starea RNN la momentul t (pentru cuvântul din poziția t)

$x_t$  = reprezentarea cuvântului de intrare din momentul t

- RNN nu procesează cuvinte, ci codificări (embeddings) ale acestora

$W_{hx}$  = ponderile stratului de intrare

$W_{hh}$  = ponderile recurente ale startului ascuns

$b_h$  = bias-ul stratului ascuns

f = funcția de activare (sigmoid / tanh / relu)

Starea RNN se determină astfel:

$$h_t = f(W_{hx} x_t + W_{hh} h_{t-1} + b_h)$$

# Predictie

- se initializează starea decoder-ului cu vectorul context generat de encoder, pe baza frazei inițiale
- scopul decoderului este să genereze o traducere a acestei fraze, cuvânt cu cuvânt
  - $c_t$  - cuvântul de la momentul t
  - V - vocabularul (mulțimea tuturor cuvintelor)
  - $s_{t-1}$  – starea cea mai recentă a decoder-ului

$$c_t = \operatorname{argmax}_{c \in V} p(c | x_{1:n}, c_{1:t-1})$$

$$P(*) | x_{1:n}, c_{1:t-1}) = \operatorname{softmax}_V (W_{ch} s_{t-1} + b_c)$$

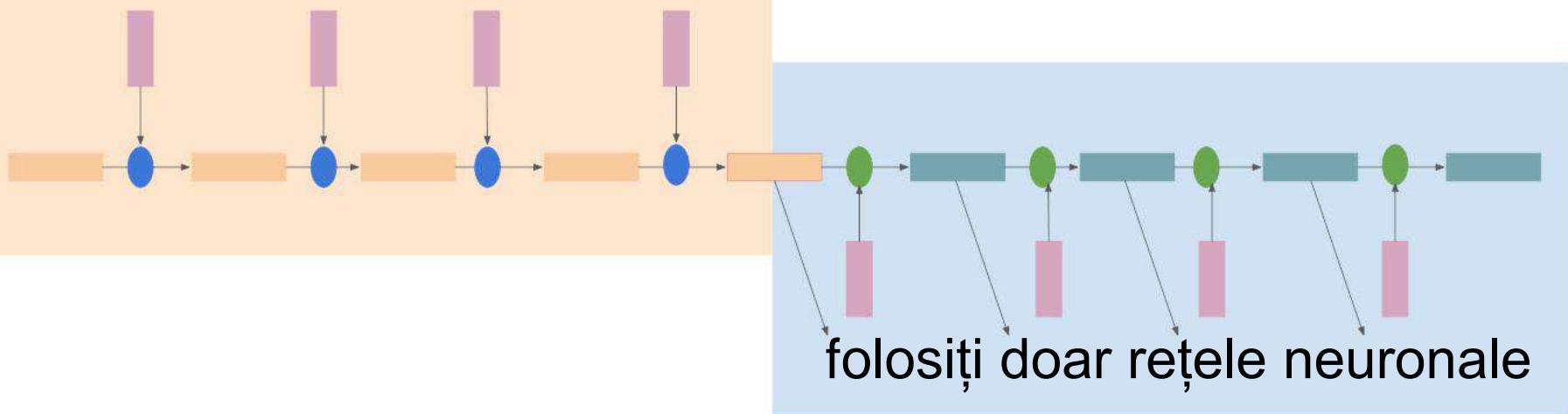
Explicații:

- cuvântul generat de decoder se alege din vocabular – cel care are cea mai mare probabilitate, având în vedere cuvintele x din fraza de intrare și cuvintele c generate anterior
- probabilitățile cuvintelor se determină aplicând funcția softmax pe valorile generate la ieșirea decoderului

# Limitări ale modelului seq2seq standard

- contextul întregii fraze de intrare = starea ecoderului generată pentru ultimul cuvânt
- ultimul cuvânt are importanța cea mai mare = problematic pentru fraze lungi

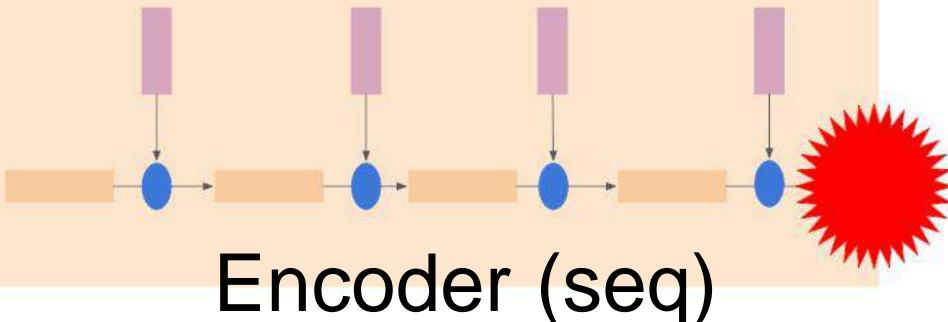
Only use neural nets



# Limitări ale modelului seq2seq standard

- encoderul realizează codificarea unor fraze de lungime variabilă folosind un vector de dimensiune constantă
  - în general dimensiunea este de ordinul sutelor)

Only use neural nets

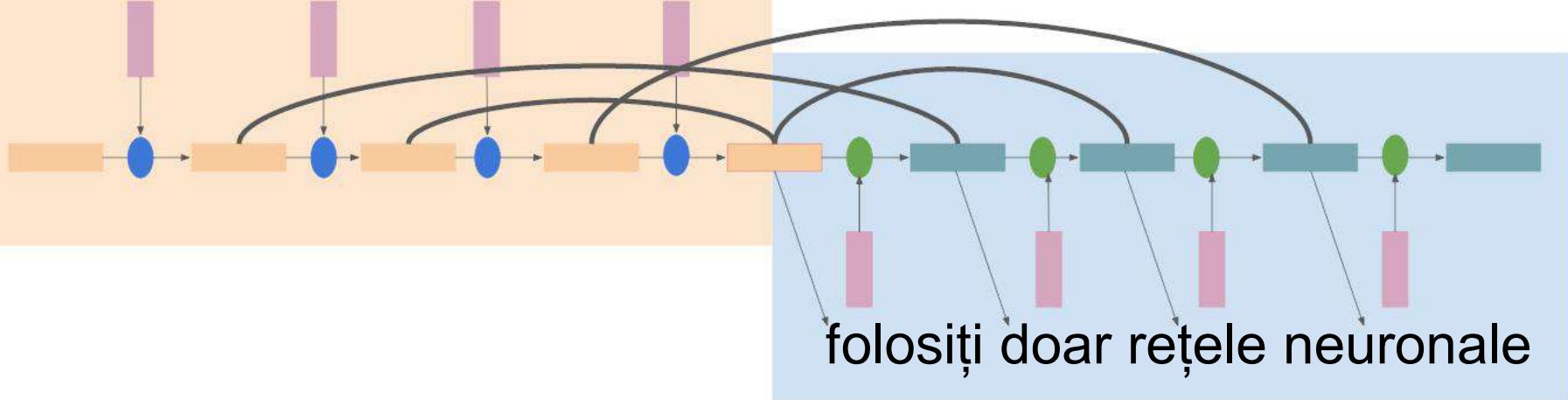


- **bottleneck** = ultima stare ascunsă a encoderului
- prelucrarea întregii fraze se reduce doar la această ultimă stare

# Soluția – Încorporarea unei componente de atenție

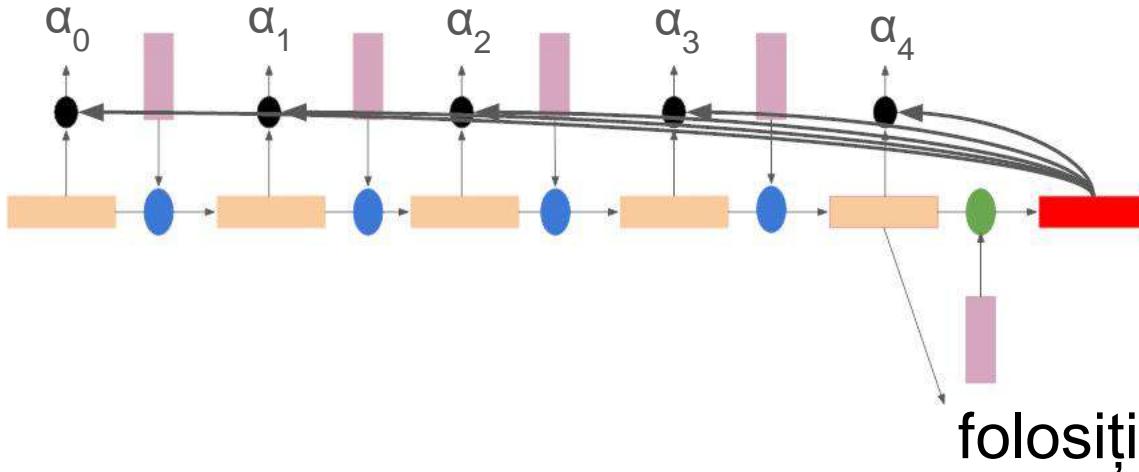
- se permite decoder-ului să aibă în vedere toate stările encoderului, nu doar pe ultima, pentru generarea fiecărui cuvânt din fraza tradusă
  - principiul se numește **atenție**
- în urma antrenării, decoder-ul învață care sunt cuvintele mai importante, i.e. care sunt stările mai importante ale encoderului

Only use neural nets



# Implementarea atenției

- atenția permite decoderului să ia în calcul anumite stări ale encoderului
- pentru fiecare stare a encoderului, din momentul fiecărui cuvânt, se determină afinitatea  $\alpha_i$  a acelei stări



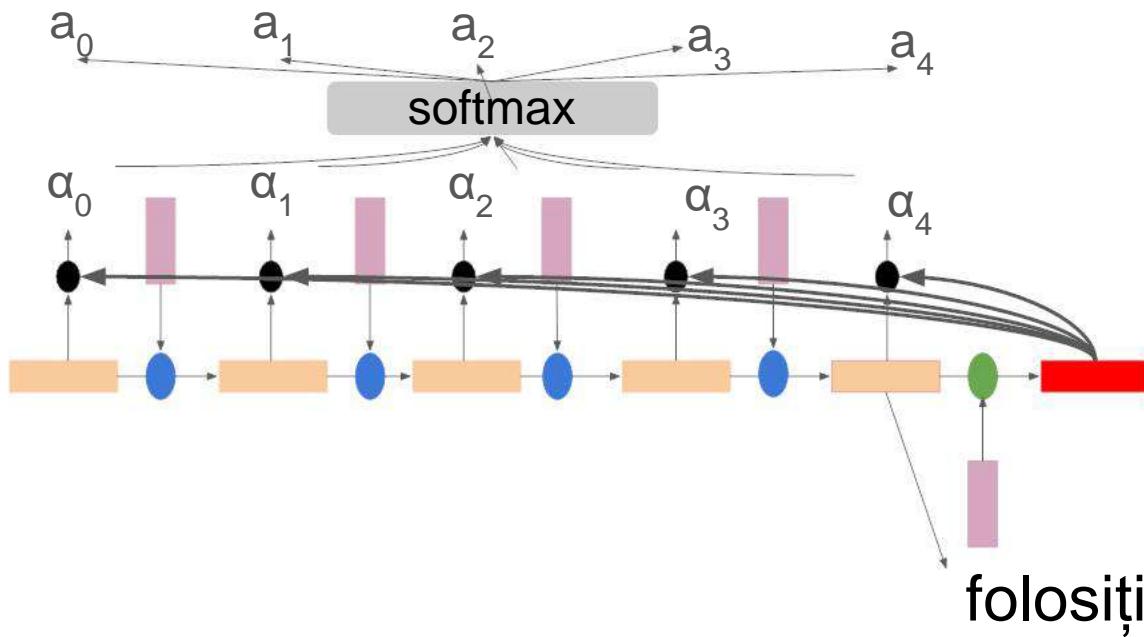
Funcția de afinitate

Spre exemplu se poate realiza produsul scalar:

$$\bullet : \text{red bar} \cdot \text{orange bar} = \alpha_i$$

# Implementarea atenției

- prin normalizarea valorilor afinităților rezultă un set de ponderi pentru fiecare stare ascunsă
- pentru normalizare se aplică funcția softmax



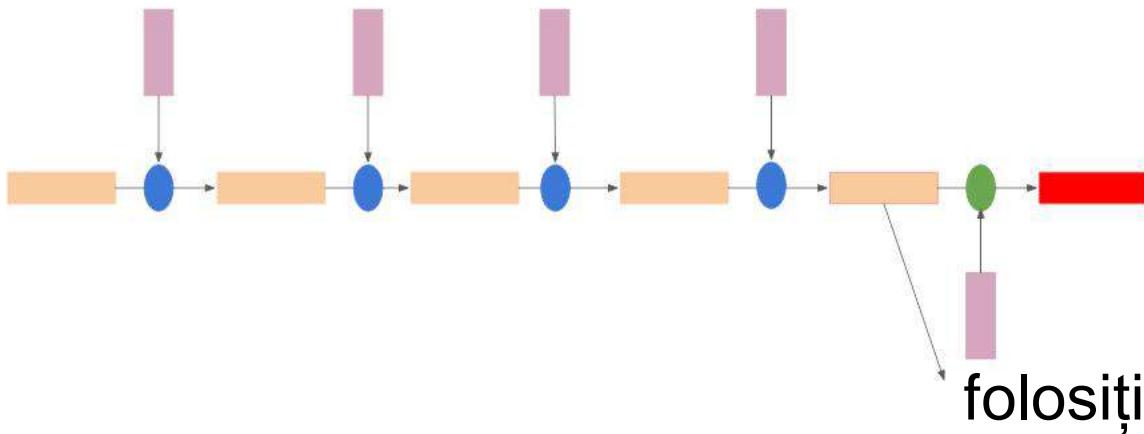
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

folosiți,

# Implementarea atenției

- contextul se obține prin suma ponderată a stărilor encoderului
- în absența atenției, contextul ar fi format doar din ultima stare a encoderului

$$a_0 \boxed{1} + a_1 \boxed{1} + a_2 \boxed{1} + a_3 \boxed{1} + a_4 \boxed{1} = \boxed{\textcolor{orange}{\text{vector context}}}$$



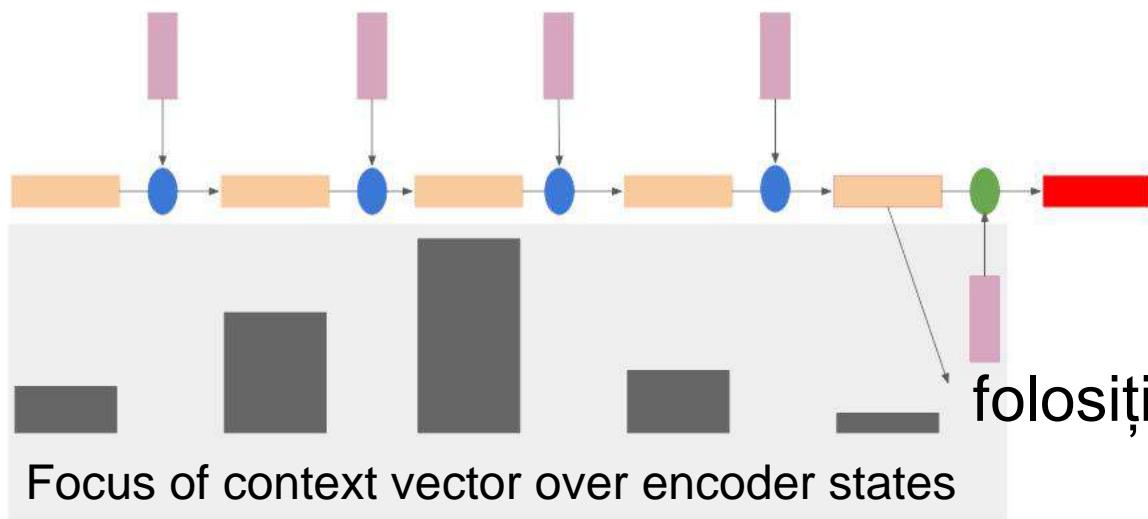
Suma ponderată:

este vectorul context obținut pe baza stărilor encoderului

# Implementarea atenției

- la generarea primului cuvânt “folosiți”, se acordă atenție sporită cuvântului “use”, care îi corespunde în fraza tradusă.
- ponderea stării encoderului corespunzătoare acestui cuvânt este mai mare decât a cuvintelor adiacente

Only use neural nets



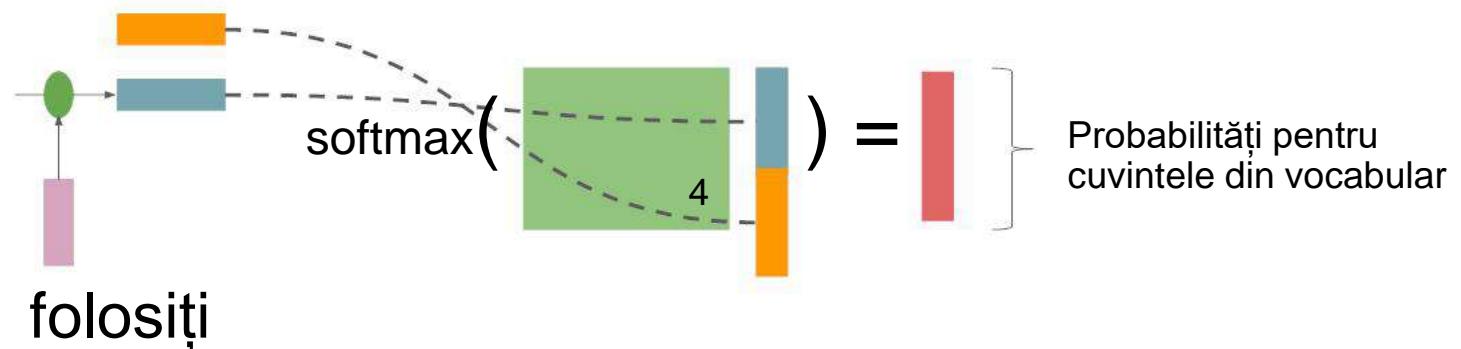
nu este necesar ca elementele corespunzătoare ale secvențelor de intrare și ieșire să fie pe aceleași poziții

unui cuvânt din fraza de intrare îi pot corespunde mai multe cuvinte din fraza tradusă, și reciproc

ponderile sunt valori reale din (0, 1)

# Implementarea atenției

- vectorul context obținut se concatenează cu starea decoderului pentru a realiza predicția.



vectorul compus deține informațiile din decoder  
codificare a frazei de la encoder

# Calculul atenției

- atenție = afinități = relații dintre starea decoderului și toate stările encoder-ului
- pentru calculul afinităților de obicei se realizează produsul scalar dintre starea decoderului și stările corespunzătoare ale encoder-ului
- presupunem că sunt n stări ale encoderului (n cuvinte în fraza de input).
  - $\alpha_i$  = afinitatea dintre starea  $i$  a encoderului și starea decoderului
  - $h_{1:n}$  = cele n stări ale encoderului
  - $s_{t-1}$  = starea decoderului (care urmează să genereze cuvântul de la momentul t)

Atunci:

$$\alpha_i = f(h_i, s_{t-1}) = h_i^T s_{t-1}$$

ponderile stărilor decoderului  $a = \text{softmax}(\alpha)$

Vectorul context  $k = \sum_{i=1:n} h_i a_i$

# Predictie

- notam  $[s;k]$  rezultatul concatenarii stării decoderului și contextului atunci

$$c_t = \operatorname{argmax}_{c \in V} p(c | x_{1:n}, c_{1:t-1})$$

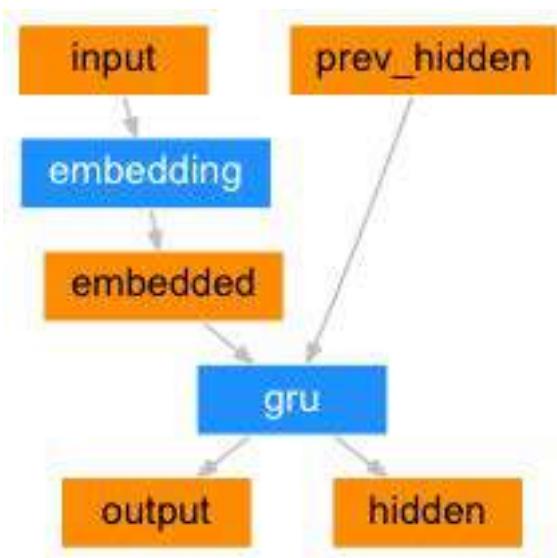
$$P(*) | x_{1:n}, c_{1:t-1}) = \operatorname{softmax}_V (W_{c2h} [s_{t-1}; k] + b_c)$$

- diferențe față de modelul fără atenție:
  - contextul se determină ca medie ponderată a stărilor encoderului (anterior se utiliza doar ultima stare)
  - pentru fiecare cuvânt, predicția se realizează atât pe baza stării decoderului, cât și a contextului
    - nu este singura posibilitate, se pot lua în calcul și alte valori, de ex. input-ul decoderului

# Proiectarea unui model encoder-decoder

## Encoder:

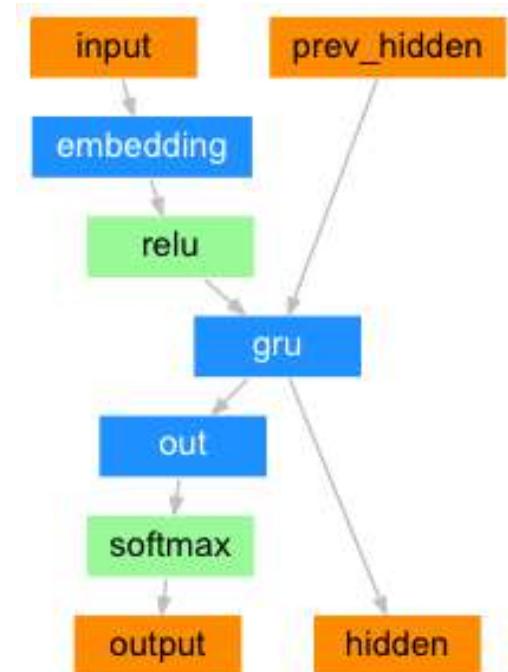
- **input**: cuvânt din fraza care trebuie tradusă
- **embedding**: strat unde se determină codificarea cuvântului
- **gru**: strat recurrent de tip Gated Recurrent Unit (rolul este similar straturilor LSTM studiate anterior)
- **output**: valoarea de la ieșirea encoderului
- **hidden**: starea encoderului generată de cuvântul curent
- **prev\_hidden**: starea anterioară a encoderului, se utilizează pentru calcului stării curente



# Proiectarea unui model encoder-decoder

## Decoder fără atenție:

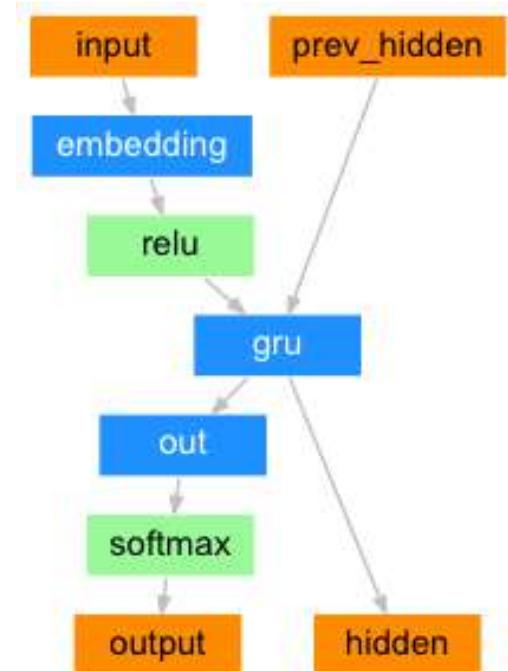
- **input**: cuvânt din fraza ce trebuie tradusă (decoderul va genera cuvântul care urmează); primul cuvânt poate fi un simbol de tip <start>
- **embedding**: strat ce realizează codificarea acestui cuvânt
- **relu**: funcție de activare
- **gru**: componenta recursivă a decoderului, de asemenea implementată printr-un strat Gated Recurrent Unit



# Proiectarea unui model encoder-decoder

## Decoder fără atenție:

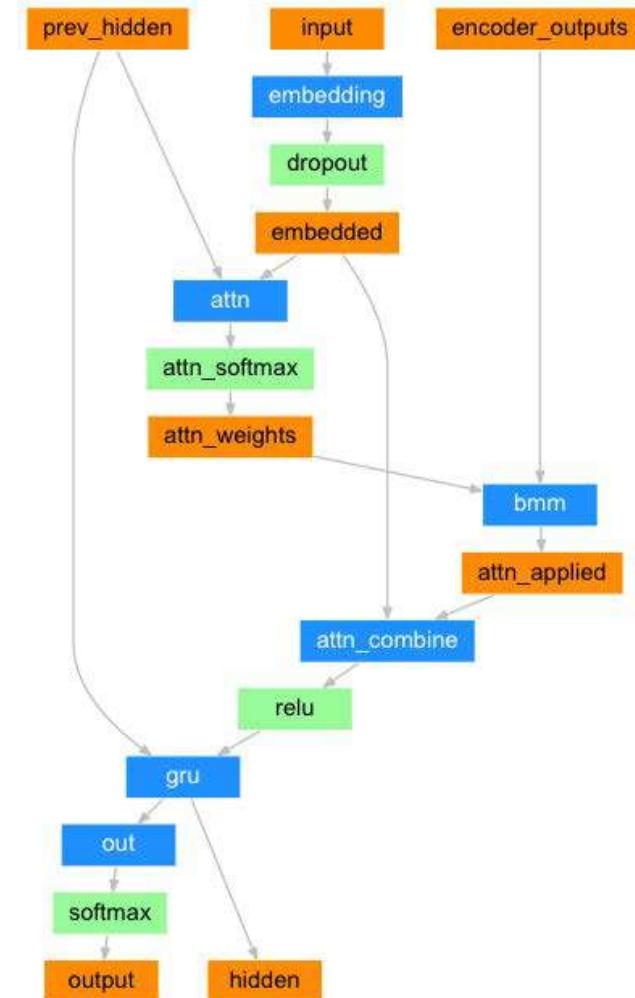
- **out + softmax**: stratul de ieșire, căruia i se aplică activare softmax
- **output**: următorul cuvânt din fraza generată de decoder, sub forma unui vector de probabilități egal cu dimensiunea vocabularului
- **hidden**: starea a decoderului, pentru primul cuvânt se initializează cu vectorul context
- **prev\_hidden**: starea anterioară a decoderului, se utilizează la calculul stării ascunse în stratul recurrent



# Proiectarea unui model encoder-decoder

## Decoder cu atenție:

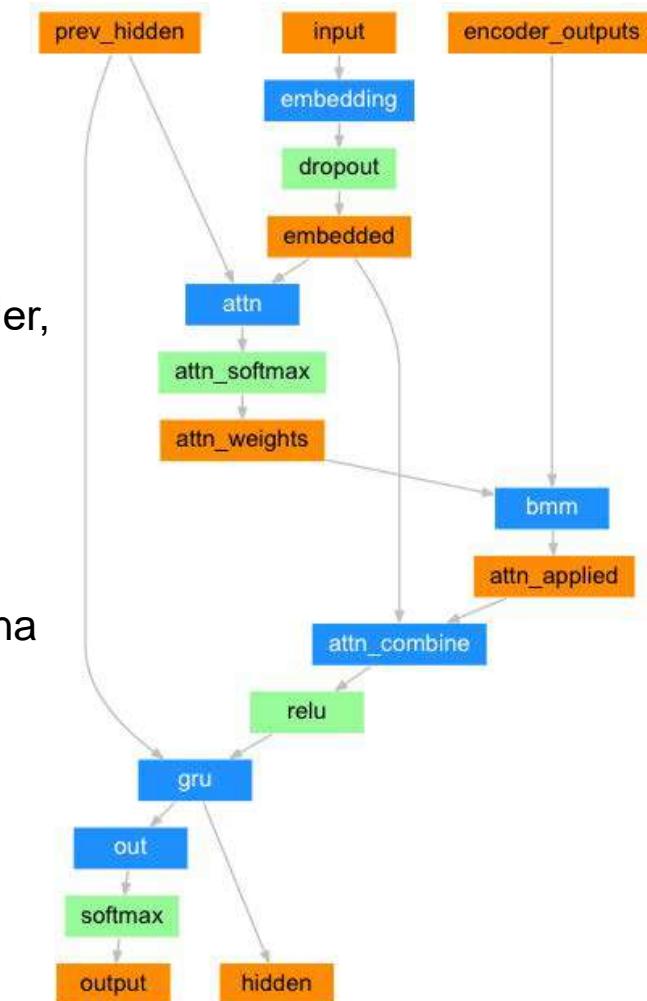
- **attn**: strat unde se determină valorile afinităților  $\alpha$ 
  - strat fully connected unde valorile  $\alpha$  se învăță în urma antrenării
- se aplică apoi activarea softmax și rezultă valorile ponderilor **attn\_weights**



# Proiectarea unui model encoder-decoder

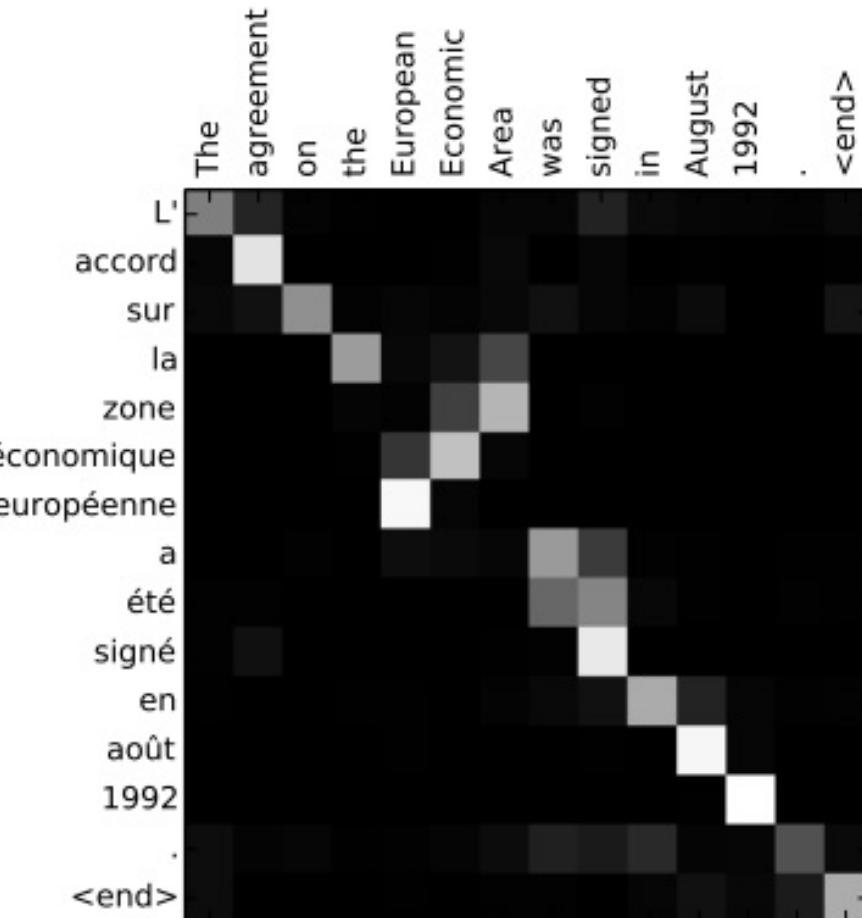
## Decoder cu atenție:

- ponderile se aplică reprezentărilor generate de encoder, de unde rezultă vectorul context **attn\_applied**
  - **bmm** = batch matrix multiplication
- **atten\_combine**: strat fully-connected unde se combina datele de intrare cu vectorul context
  - stratul recurrent al decoderului (**gru**) primește rezultatul combinării celor două valori



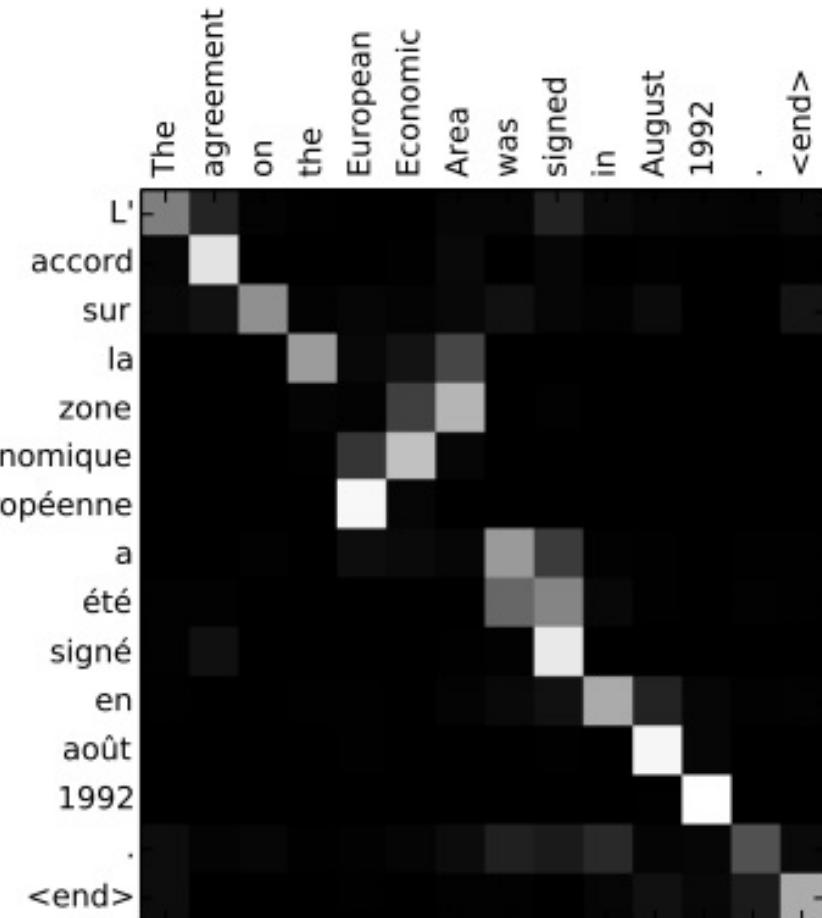
# Vizualizarea atenției

- **ponderile care compun vectorul context se reprezintă prin valori grayscale**
- **fiecare valoare = atenția care se acordă cuvintelor din fraza inițială relativ la cea tradusă**



# Vizualizarea atenției

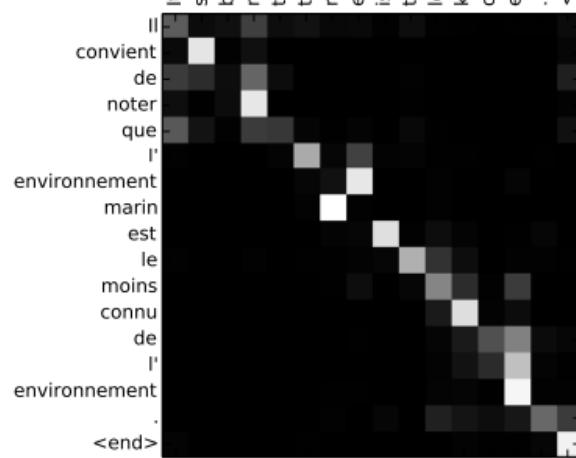
- corespondența cuvintelor din cele două fraze NU este 1:1
- există situații în care
  - unei expresii din prima frază îi corespunde altă expresie din cealaltă frază
  - cuvintele nu corespund direct și/sau nu sunt în aceeași ordine în cele două limbi
    - ex: *European Economic Area vs zone économique européennes*
  - atenția are valori mai mari pentru cuvinte sau grupuri de cuvinte corelate, nu se limitează doar la perechi de cuvinte



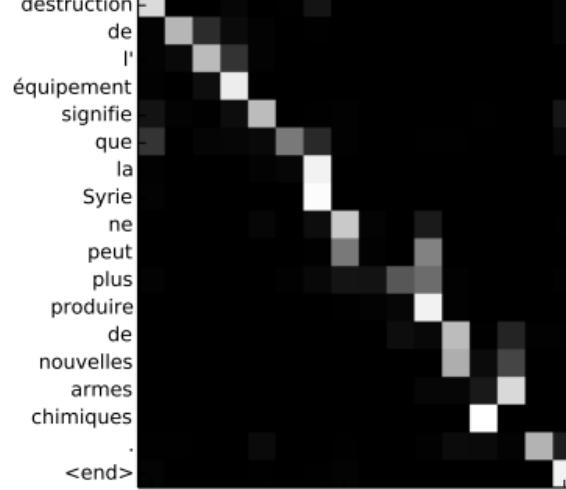
# Vizualizarea atenției

- alte exemple

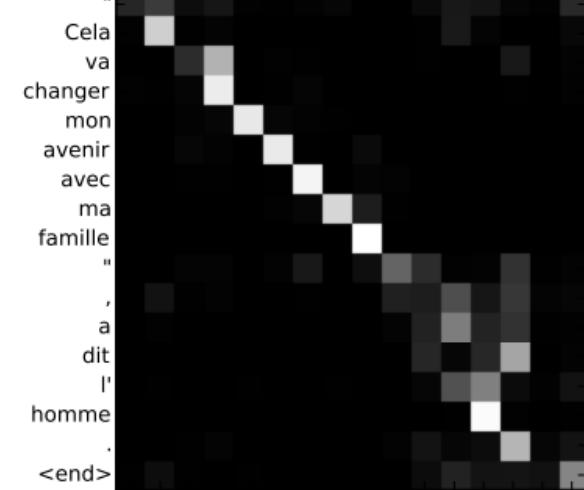
Il convient de noter que l'environnement marin est le moins connu de l'environnement .  
It should be noted that the marine environment is the least known of environments .  
<end>



La destruction de l'équipement signifie que la Syrie ne peut plus produire de nouvelles armes chimiques .  
Destruction of the equipment means that Syria can no longer produce new chemical weapons .  
<end>



Cela va changer mon avenir avec ma famille .  
" , a dit l'homme .  
" This will change my future with my family .  
the man said .  
<end>



# Vizualizarea atenției

## - alte exemple

Il convient de noter que l'environnement marin est le moins connu de l'environnement .  
It should be noted that the marine environment is the least known of environments .

La destruction de l'équipement signifie que la Syrie ne peut plus produire de nouvelles armes chimiques .  
Destruction of the equipment means that Syria can no longer produce new chemical weapons .

" Cela va changer mon avenir avec ma famille " , a dit l'homme .  
" This will change my future with my family , " the man said .

valori mari pe diagonala principală = atenția pentru un cuvânt se acordă unui alt cuvânt corespunzător, aflat pe aceeași poziție în fraza tradusă

grupare de valori ale atenției: corespondența este la nivel de expresie, și nu de cuvânt individual  
unei expresii din fraza inițială îi corespunde o altă expresie din fraza tradusă,  
cuvintele nu sunt în ordine și nu corespund 1:1