

# Calcul numeric

## Rețele neuronale

Paul Irofti  
Andrei Pătrașcu  
Cristian Rusu

Departmentul de Informatică  
Facultatea de Matematică și Informatică  
Universitatea din București  
Email: `prenume.nume@fmi.unibuc.ro`



- ▶ Funcții de activare
- ▶ Funcții obiectiv
- ▶ Minimzarea erorii. Metoda gradient.
- ▶ Neuroni artificiali
- ▶ Rețele neuronale

Cursul folosește resursele și urmează capitolele 2–6 din O. Calin (2020). *Deep learning architectures*. Springer



# Funcții de activare

Pentru a obține un model neliniar, rețelele neuronale folosesc funcții de activare neliniare.

Tipuri de funcții:

- ▶ liniare
- ▶ treaptă
- ▶ sigmoid
- ▶ băț de hochei

Proprietăți:

- ▶ diferențiabile sau cu discontinuități
- ▶ mărginite sau nemărginite



# Activare: liniară

Funcția liniară:

$$f(x) = kx, \quad k > 0 \quad (1)$$

folosită de obicei în ultimul strat al rețelelor pentru ieșirea modelului.

Derivata funcției de activare (numită și rata de tragere):

$$f'(x) = k \quad (2)$$

este folositoare în analiză și, mai ales, la algoritmul de retropropagare.

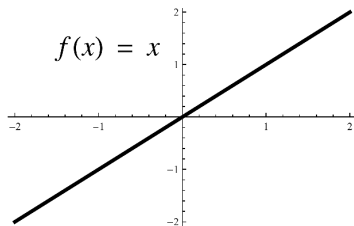
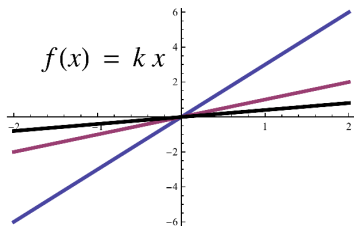
Funcția identitate:

$$f(x) = x \quad (3)$$

folosită pentru neuronii liniari.



# Curbă liniară



Cunoscută drept funcția treaptă, unitate, Heaviside:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (4)$$

Proprietăți:

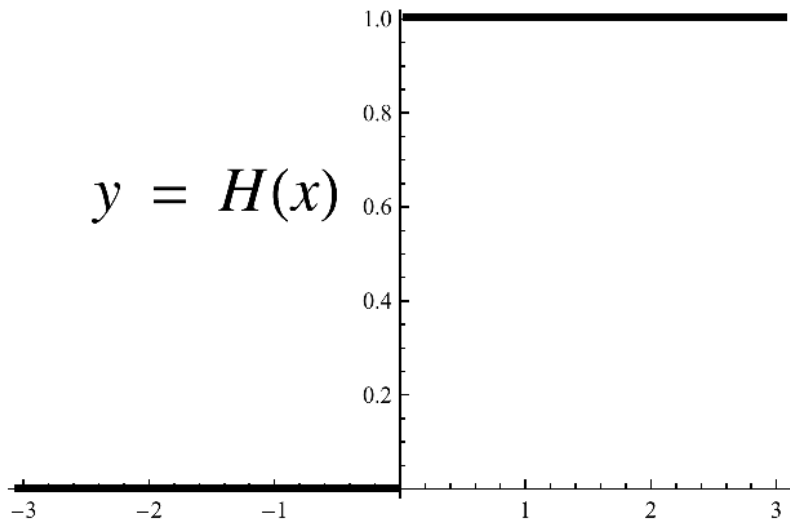
- ▶ este activat doar când  $x \geq 0$
- ▶ nu este diferențiabil în  $x = 0$
- ▶ privită ca o funcție generalizată (ca o distribuție) are derivată:

$$H'(x) = \delta(x) \quad (5)$$

unde  $\delta(x)$  este funcția Dirac, funcția impuls.



# Curbă treaptă



Cunoscută drept funcția signum, unitate bipolară:

$$S(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (6)$$

Proprietăți:

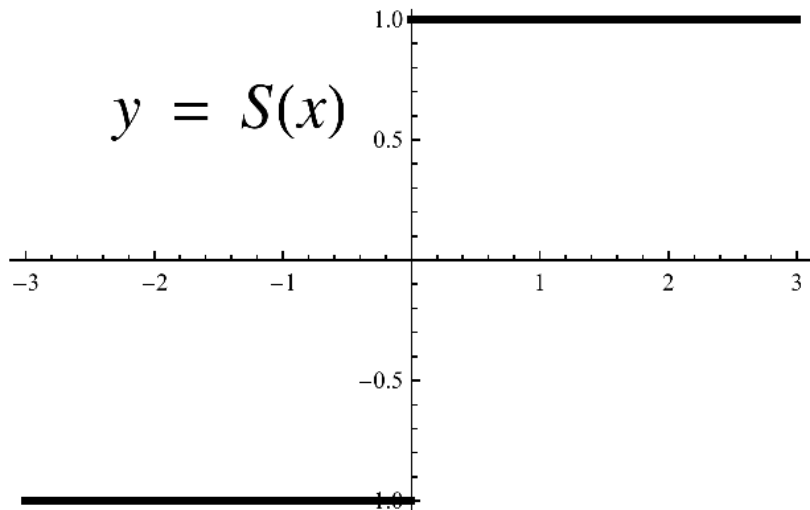
- ▶ este activată în ambele ramuri
- ▶ în continuare nu este diferențiabilă în  $x = 0$
- ▶ este legată de treaptă prin  $S(x) = 2H(x) - 1$
- ▶ privită ca o funcție generalizată (ca o distribuție) are derivată:

$$S'(x) = 2H'(x) = 2\delta(x) \quad (7)$$





# Curbă signum



# Activare: ReLU

Funcțiile de tip băț de hochei (hockey-stick) sunt în formă de L, și în general pleacă de la rectificarea funcției liniare (rectified linear unit) ce au la bază partea pozitivă a argumentului primit.

ReLU (rectified linear unit) este cea mai des folosită și este folosită în crearea unor noi funcții după nevoia aplicației de desubt:

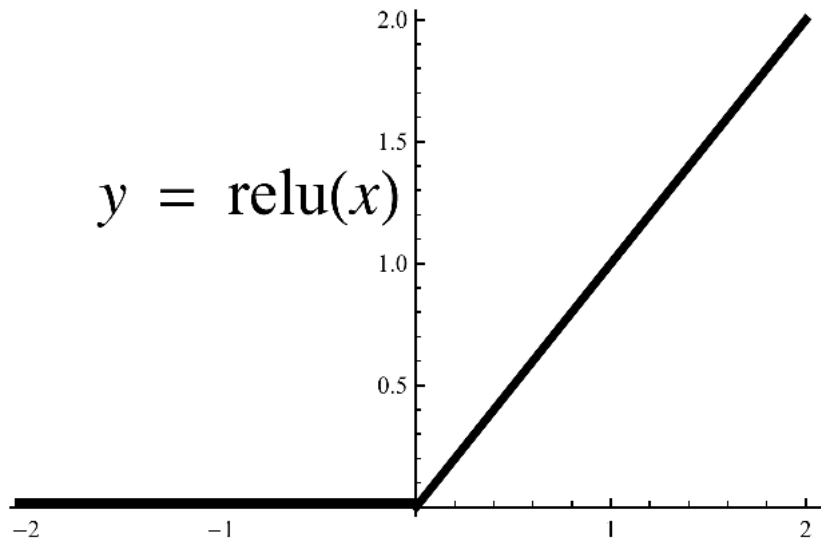
$$ReLU(x) = \max\{x, 0\} = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (8)$$

Proprietăți:

- ▶ este legată de treaptă prin  $ReLU(x) = xH(x)$
- ▶ derivabilă cu  $ReLU'(x) = H(x)$
- ▶ nu ajunge la saturație când folosim gradientul pentru retropropagare



# Curbă ReLU



PReLU (parametric rectified linear unit) este varianta parametrizată a ReLU:

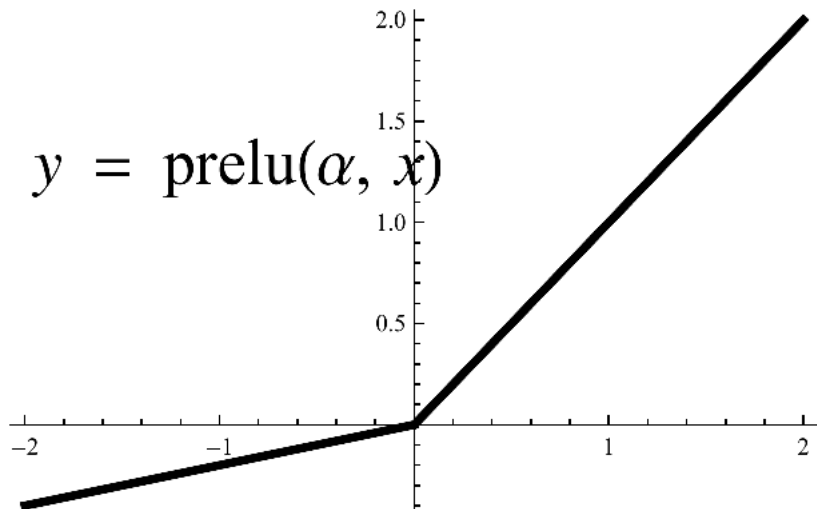
$$PReLU(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}, \alpha > 0 \quad (9)$$

Proprietăți:

- ▶ este liniară pe părți
- ▶ are rate de activare diferite pentru  $x < 0$  și  $x > 0$
- ▶ cum este legată de celelalte funcții?
- ▶ cum derivăm?



## Curbă ReLU parametrizată



# Activare: sigmoid

Funcțiile de tip sigmoid au avantajul de a fi netede (smooth) și de a putea să aproximeze funcția treaptă la orice nivel de acuratețe impus.

Pentru funcțiile ce au valori în intervalul  $[0, 1]$  valoarea lor pot fi folosite drept probabilități cu aplicații directe în problemele de clasificare, identificare, detecție de anomalii etc.



# Activare: sigmoid logistic

Cunoscută drept funcția logistică sau soft-step cu parametru  $c > 0$ :

$$\sigma_c(x) = \sigma(c, x) = \frac{1}{1 + e^{-cx}} \quad (10)$$

unde  $c$  influențează rata de activare: valorile mari duc la o schimbare bruscă de la 0 la 1.

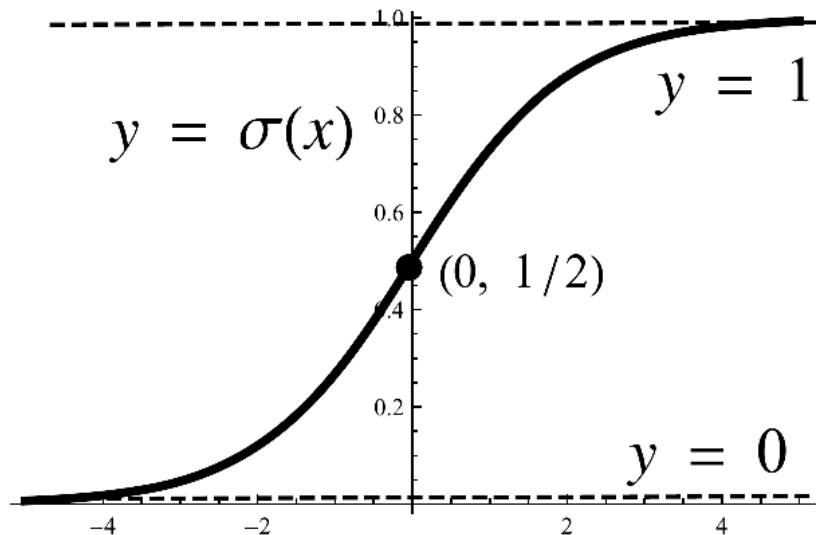
Proprietăți:

- ▶ când  $c \rightarrow \infty$  funcția devine  $H(x)$  (**demonstrați!**)
- ▶ graficul funcției este independent de  $c$  la  $x = 0$ :  $\sigma_c(0) = \frac{1}{2}$
- ▶  $\sigma_c$  reprezintă o funcție monotonă ce transformă linia reală în intervalul  $(0, 1)$
- ▶ derivabilă cu  $\sigma'_c = c\sigma_c(1 - \sigma_c)$  (**demonstrați!**)
- ▶ pentru  $c = 1$  avem funcția standard logistică  $\sigma(x)$
- ▶ inversa funcției standard, **logit**, este

$$\sigma^{-1}(x) = \log\left(\frac{x}{1-x}\right) \quad (11)$$



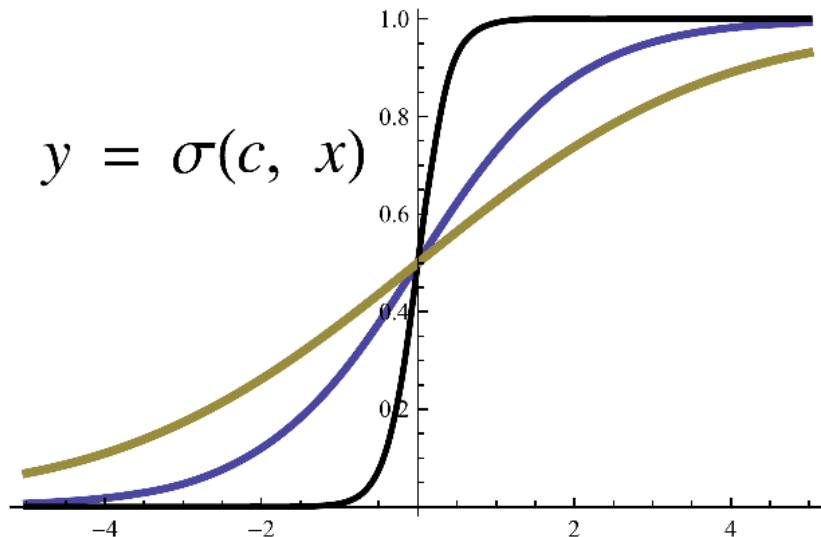
# Curbă sigmoid logistic





## Curbă sigmoid logistic parametrizat

$$y = \sigma(c, x)$$



# Activare: sigmoid hiperbolic

Cunoscută drept funcția hiperbolic tangentă sau sigmoidă bipolară:

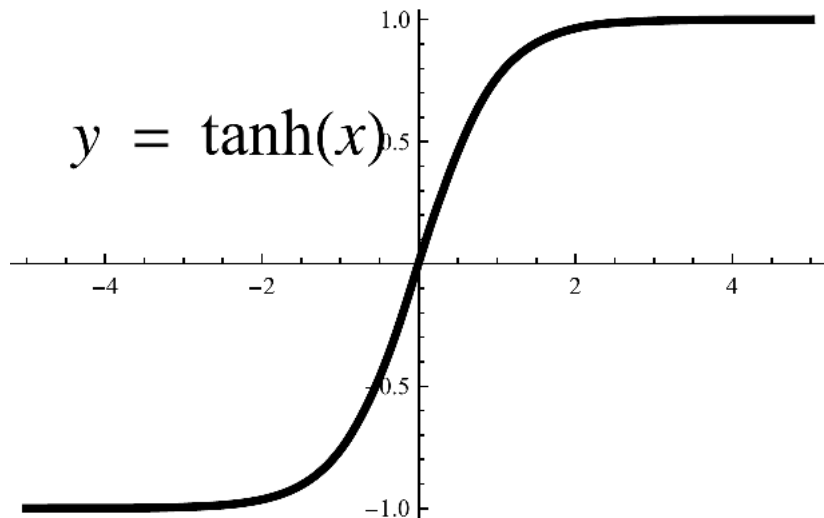
$$t(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (12)$$

Proprietăți:

- ▶  $t$  reprezintă o funcție monotonă ce transformă linia reală în intervalul  $(0, 1)$
- ▶ tinde orizontal asimptotic la  $\pm 1$
- ▶ este legată de sigmoidul logistic prin  $t(x) = 2\sigma_2(x) - 1$  (**demonstrați!**)
- ▶ derivabilă cu  $t'(x) = 1 - t^2(x)$  (**demonstrați!**)
- ▶ graficul funcției trece prin origine și este simetric



## Curbă sigmoid hiperbolic



Funcția softmax este o variantă netedă (smooth) a funcției max:

$$\text{softmax}_c(x)_i = \frac{e^{cx_i}}{\|e^{cx}\|_\ell}, \quad c > 0 \quad (13)$$

unde  $x_i$  reprezintă elementul  $i$  al vectorului  $x$ , iar norma  $\ell$  este de regulă  $\ell_1$ .

Proprietăți:

- ▶ vectorii unitate  $e_i$  mai sunt denumiți *one-hot* vectori; atunci pentru  $c \rightarrow \infty$  avem (**demonstrați**):

$$\lim_{c \rightarrow \infty} \text{softmax}_c(x) = e_k, \quad \text{unde } k = \arg \max \{x_1, \dots, x_n\} \quad (14)$$

- ▶ folosit adesea ca funcția de activare a ultimului strat din rețeaua neuronală



# Funcții obiectiv

Parametrii rețelei neuronale caută să minimizeze o funcție obiectiv cunoscută și drept cost, eroare sau de pierdere (loss).

Funcția obiectiv măsoară similaritatea sau distanța dintre intrarea și ieșirea modelului. Acestea pot fi scalari, vectori, matrice, tensori sau alte obiecte matematice.

Notății:

- ▶ intrarea  $x$ , ieșirea  $y$
- ▶ ponderile  $w$ , bias  $b$
- ▶ rețeaua poate fi privită drept o problemă de aproximare de funcții ce produce  $y = f_{w,b}(x)$
- ▶ funcția ce trebuie să fie aproximată este  $z = \phi(x)$
- ▶ funcția de cost este astfel  $C(w, b) = \text{dist}(y, z)$
- ▶ parametrii optimi ai rețelei sunt

$$(w^*, b^*) = \arg \min_{w, b} C(w, b) \quad (15)$$



# Obiectiv: supremum

Rețeaua primește la intrare  $x \in [0, 1]$  și învață funcția continuă  $\phi : [0, 1] \rightarrow \mathbb{R}$  astfel încât:

$$C(w, b) = \sup_{x \in [0, 1]} |f_{w, b}(x) - \phi(x)| \quad (16)$$

sau în cazul discret în care  $x \in \mathbb{R}^n$  și  $z_i = \phi(x_i)$ :

$$C(w, b) = \max_{1 \leq i \leq n} \{f_{w, b}(x_i) - z_i\} = \max_{1 \leq i \leq n} \{y_i - z_i\} \quad (17)$$



# Obiectiv: distanța euclidiană

Rețeaua primește la intrare  $x \in [0, 1]$  și învață funcția continuă  $\phi : [0, 1] \rightarrow \mathbb{R}$  este pătrat integrabilă:

$$C(w, b) = \int_0^1 (f_{w,b}(x) - \phi(x))^2 dx \quad (18)$$

sau în cazul discret în care  $x \in \mathbb{R}^n$  și  $z_i = \phi(x_i)$ :

$$C(w, b) = \sum_{i=1}^n (f_{w,b}(x_i) - z_i)^2 = \|f_{w,b}(x) - z\|^2 = \|y - z\|^2 \quad (19)$$

ce conduce la rezolvarea ecuațiilor normale aplicând metoda gradient.



## Obiectiv: eroare pătratică medie

Rețeaua primește la intrare variabila aleatoare  $X$  și produce la ieșire variabila aleatoare  $Y = f_{w,b}(X)$  ce trebuie să aproximeze variabila aleatoare  $Z$ :

$$C(w, b) = \mathbb{E}[(Y - Z)^2] = \mathbb{E}[(f_{w,b}(x) - Z)^2] \quad (20)$$

sau în cazul discret în care avem măsurătorile  $(x_1, z_1), \dots, (x_n, z_n)$ :

$$C(w, b) = \frac{1}{n} \sum_{i=1}^n (f_{w,b}(x_i) - z_i)^2 \quad (21)$$

ce conduce la rezolvarea ecuațiilor normale aplicând metoda gradient.





## Obiectiv: antrenare vs. testare

Datele de intrare se mai numesc și date de antrenare.

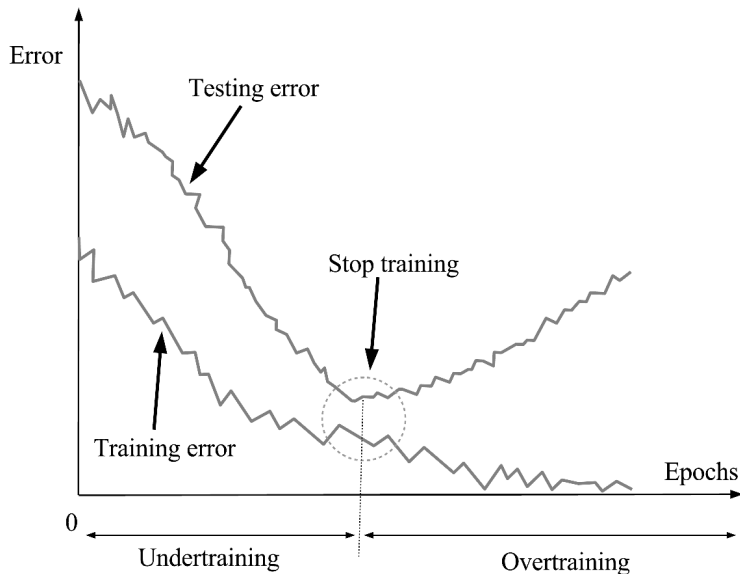
Parcurgerea întregului set de date în timpul antrenării reprezintă o epocă. Repetarea antrenării implică mai multe epoci ce duc la scăderea suplimentară a erorii.

Un număr mic de epoci conduce la o oprire prematură ce oferă o aproximare suboptimală (undertraining). Numărul excesiv de epoci duce la fenomenul de supraspecializare (overtraining).

La finalul antrenării este păstrat un set separat de date de intrare numit set de testare. În cazul suboptimal, testarea va indica erori ridicate de aproximare. La fel și în cazul supraspecializării datorită lipsei capacității de generalizare a modelului.



# Scenariu antrenare vs. testare



# Minimizarea erorii



# Metoda gradient



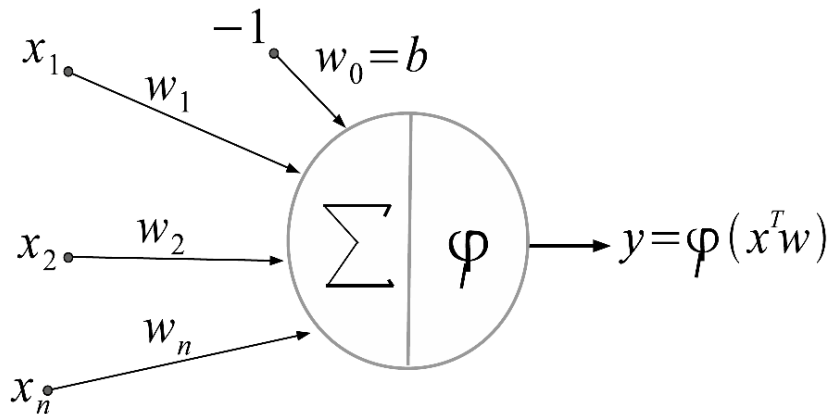
## Definiție

*Un neuron artificial (sau abstract) este cvadrupul  $(x, w, \varphi, y)$  unde  $x$  reprezintă intrările  $x_0, \dots, x_n$  și  $w$  ponderile asociate (cu  $x_0 = -1$  intrarea asociată bias-ului  $w_0 = b$ ), iar  $\varphi$  este funcția de activare care produce ieșirea  $y = \varphi(w^T x) = \varphi(x^T w)$ .*

- ▶ intrările reprezintă nodurile (verticele)
- ▶ ponderile reprezintă muchiile
- ▶ neuronul este împărțit în unitatea de calcul  $\Sigma$  și funcția de activare (sau primitivă)  $\varphi$ .



## Schemă neuroni



## Definiție

*Perceptronul este un neuron artificial cu intrare binară  $x_i \in \{0, 1\}$  și cu funcție de activare Heaviside (treaptă):*

$$\varphi(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (22)$$

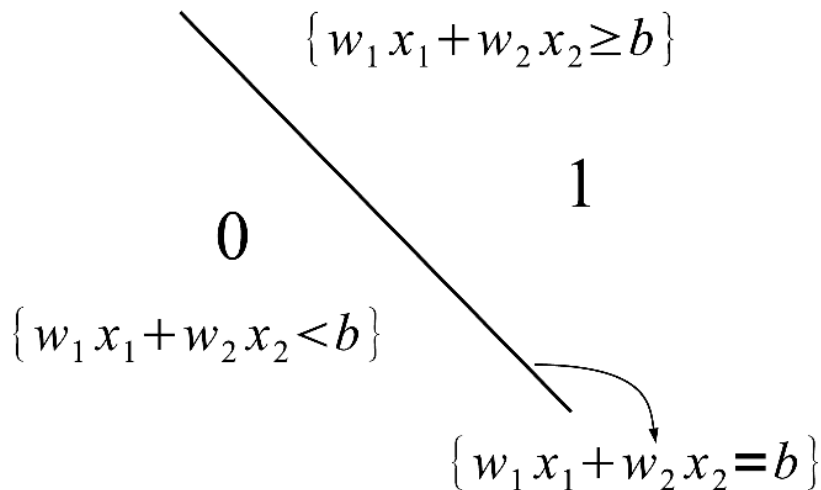
leșirea perceptronului poate fi interpretată drept un operator de thresholding:

$$y = \varphi(w^T x - b) = \begin{cases} 0, & w^T x < b \\ 1, & w^T x \geq b \end{cases} \quad (23)$$

ce ne duce cu gândul la funcția de hinge-loss folosită la SVM.



## Perceptron: hiperplan





# Perceptron: ȘI logic

Plecând de la tabelul de adevăr al funcției boolene ȘI:

$x_1$	$x_2$	$y = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

cum putem modela operația cu ajutorul unui perceptron?



# Perceptron: ȘI logic

Plecând de la tabelul de adevăr al funcției boolene ȘI:

$x_1$	$x_2$	$y = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

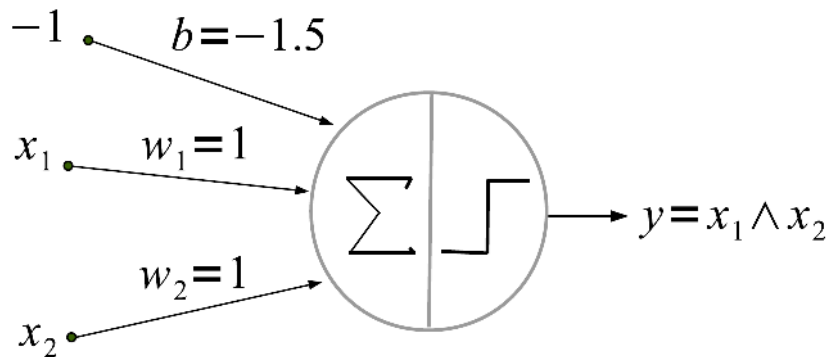
cum putem modela operația cu ajutorul unui perceptron?

**Soluție:** două intrări  $x_{1,2} \in \{0, 1\}$  cu ponderile  $w_{1,2} = 1$  și  $b = -1,5$  (și  $x_0 = -1$ ).

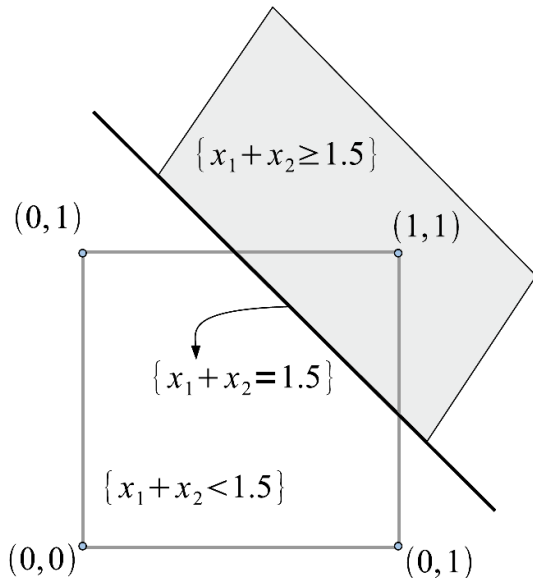
$$y = \varphi(x_1 + x_2 - 1,5) = \begin{cases} 0, & x_1 + x_2 < 1,5 \\ 1, & x_1 + x_2 \geq 1,5 \end{cases} \quad (24)$$



## Perceptron: implementare ȘI logic



# Perceptron: Şİ logic



# Perceptron: SAU logic

Plecând de la tabelul de adevăr al funcției boolene SAU:

$x_1$	$x_2$	$y = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

cum putem modela operația cu ajutorul unui perceptron?



# Perceptron: SAU logic

Plecând de la tabelul de adevăr al funcției boolene SAU:

$x_1$	$x_2$	$y = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

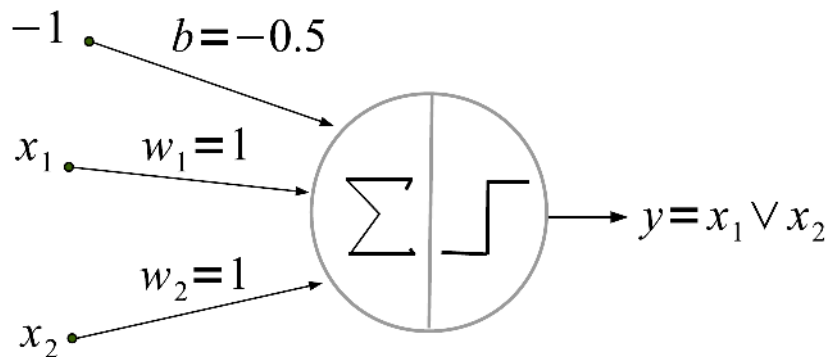
cum putem modela operația cu ajutorul unui perceptron?

**Soluție:** două intrări  $x_{1,2} \in \{0, 1\}$  cu ponderile  $w_{1,2} = 1$  și  $b = -0,5$  (și  $x_0 = -1$ ).

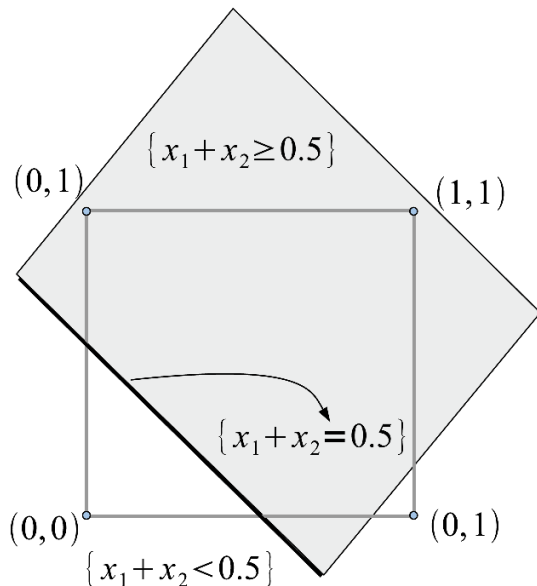
$$y = \varphi(x_1 + x_2 - 0,5) = \begin{cases} 0, & x_1 + x_2 < 0,5 \\ 1, & x_1 + x_2 \geq 0,5 \end{cases} \quad (25)$$



## Perceptron: implementare SAU logic



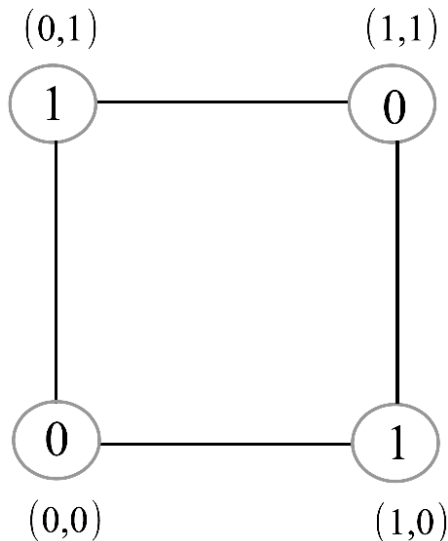
# Perceptron: SAU logic





# Perceptron: problema XOR

Cum modelăm operația booleană XOR cu un perceptron?



# Perceptron: problema grupării (clustering)

Extinderea problemei de separare către problema generală de clasificare duce la acceptarea intrărilor de tip real  $x \in [0, 1]$ .

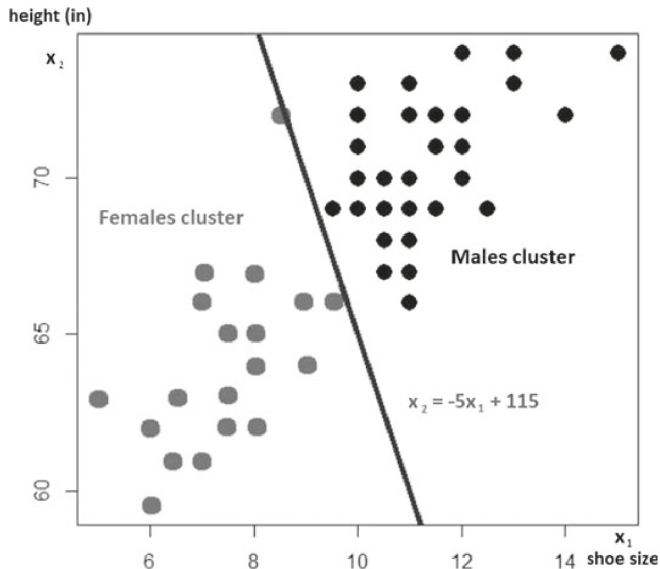
Pentru cazul  $n = 2$  cu două clase  $c_1$  și  $c_2$  avem:

$$z(x_1, x_2) = \begin{cases} 0, & (x_1, x_2) \in c_1 \\ 1, & (x_1, x_2) \in c_2 \end{cases} \quad (26)$$

iar dacă cele două clase pot fi despărțite de o line (în pătratul unitate) atunci un perceptron poate modela corect funcția  $z$ .



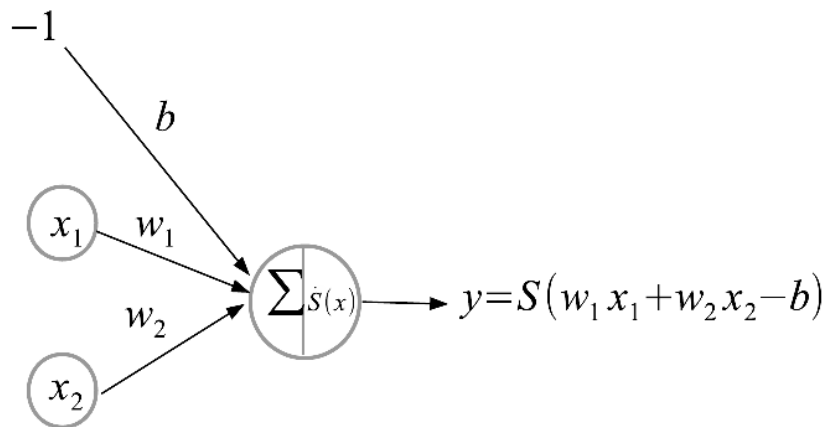
# Perceptron: exemplu clustering



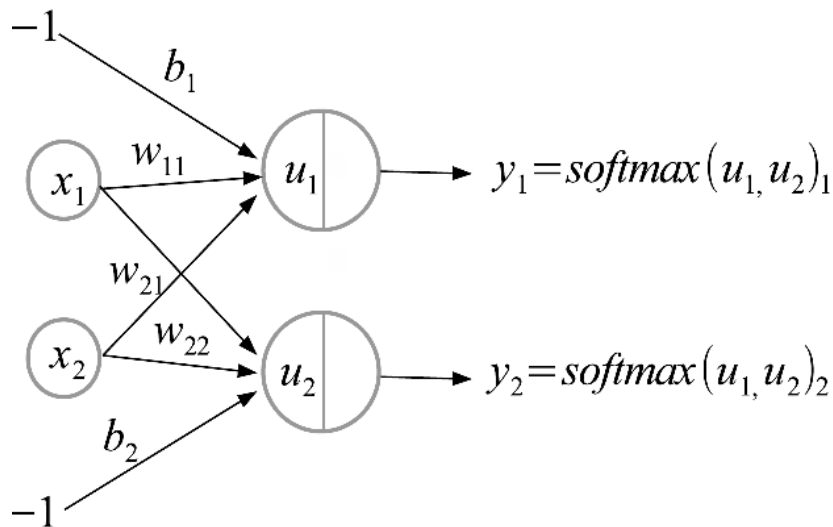
Modelați o astfel de soluție folosind un perceptron!



## Clasificare signum



## Clasificare softmax



# Neuroni: sigmoid

O altă relaxare a perceptronului privește ieșirea.

Înloc să emită stările 0 sau 1, dorim ca neuronul să ofere o plajă continuă  $y \in (0, 1)$  astfel încât ieșirea să capete rol de probabilitate sau de verosimilitatea asumării unei decizii ponderate.

## Definiție

*Un neuron sigmoid este alcătuit dintr-o unitate de calcul ce are asociată funcția de activare logistică  $\sigma$ .*

Astfel ieșirea neuronului este un număr în intervalul  $(0, 1)$ :

$$y = \sigma(w^T x - b) = \frac{1}{1 + e^{-w^T x + b}} \quad (27)$$

Dacă înlocuim funcția cu varianta parametrizată  $\sigma_c$  și acordăm lui  $c$  valori suficient de mari, atunci ieșirea neuronului sigmoid este echivalentă cu cea a perceptronului (**demonstrați!**).



# Rețele neuronale

## Definiție

*Rețelele neuronale sunt formate din mai multe straturi de neuroni a căror ieșire este conectată la alte straturi de neuroni.*

## Definiție

*Legăturile dintre neuroni, denumite și muchii sau sinapse, au asociate câte o pondere  $w_{ij}^{(\ell)}$  unde:*

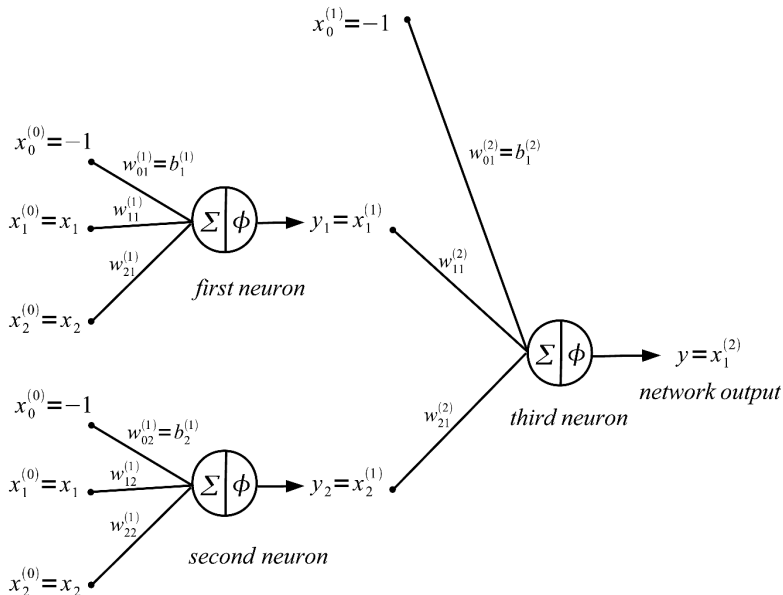
- ▶  $(\ell)$  – indicele stratului destinație
- ▶  $i$  – reprezintă neuronul sursă din stratul  $\ell - 1$
- ▶  $j$  – reprezintă neuronul destinație din stratul  $\ell$

Stratul 0 reprezintă intrarea modelului, notată cu  $x$  anterior, și cu  $x^{(0)}$  în cazul rețelelor neuronale.

Stratul  $L$  reprezintă ieșirea modelului, notată cu  $y$  anterior, și cu  $x^{(L)}$  în cazul rețelelor neuronale.



# Exemplu didactic: perceptroni multipli





## Exemplu didactic: relații între straturi

Notăm cu  $s_i^{(\ell)}$  semnalul din perceptronul  $i$  din stratul  $\ell$ .

Pentru primul perceptron din primul strat:

$$\begin{aligned} s_1^{(1)} &= w_1^{(1)T} x^{(0)} = w_{01}^{(1)} x_0^{(0)} + w_{11}^{(1)} x_1^{(0)} + w_{21}^{(1)} x_2^{(0)} \\ y_1 &= x_1^{(1)} = \varphi(s_1^{(1)}) = \varphi(w_1^{(1)T} x^{(0)}) \end{aligned} \quad (28)$$

Pentru cel de-al doilea perceptron din primul strat:

$$\begin{aligned} s_2^{(1)} &= w_2^{(1)T} x^{(0)} = w_{02}^{(1)} x_0^{(0)} + w_{12}^{(1)} x_1^{(0)} + w_{22}^{(1)} x_2^{(0)} \\ y_2 &= x_2^{(1)} = \varphi(s_2^{(1)}) = \varphi(w_2^{(1)T} x^{(0)}) \end{aligned} \quad (29)$$

În formă matricială:

$$s^{(1)} = w^{(1)T} x^{(0)} ; \quad x^{(1)} = \varphi(s^{(1)}) = \varphi(w^{(1)T} x^{(0)}) \quad (30)$$



## Exemplu didactic: relații între straturi

Neuronul din stratul al doilea:

$$\begin{aligned}s^{(2)} &= s_1^{(1)} = w_1^{(2)T} x^{(1)} = w_{01}^{(2)} x_0^{(1)} + w_{11}^{(2)} x_1^{(1)} + w_{21}^{(2)} x_2^{(1)} \\ y = x_1^{(2)} &= \varphi(s^{(2)}) = \varphi(w^{(2)T} x^{(1)})\end{aligned}\tag{31}$$

Astfel ieșirea este o înlănțuire:

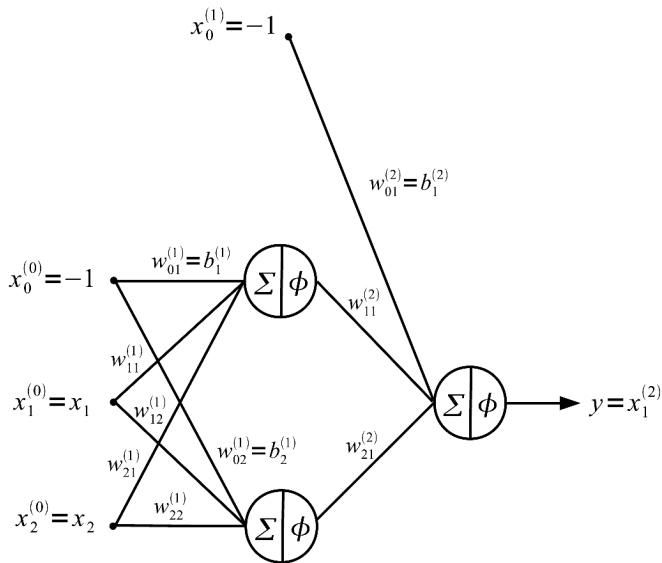
$$f_w(x^{(0)}) = y = \varphi(w^{(2)T} \varphi(w^{(1)T} x^{(0)}))\tag{32}$$

Ce se întâmplă dacă  $\varphi(x) = x$ ?

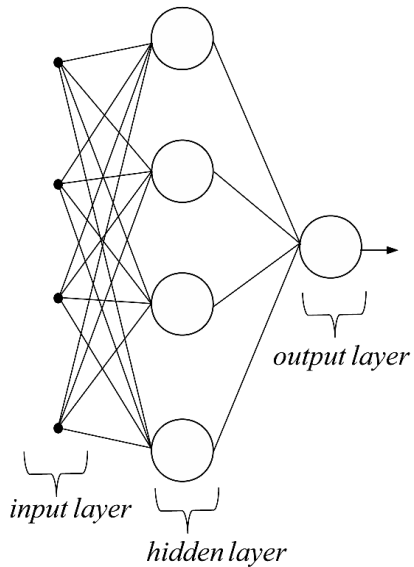
Dar dacă  $\varphi(x) = H(x)$ ?



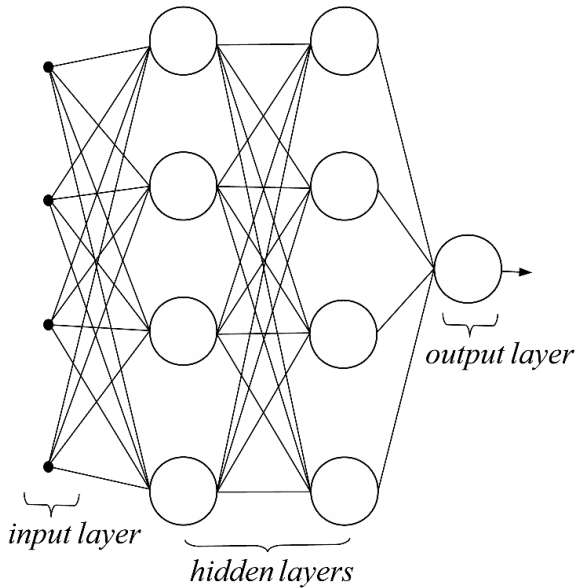
# Exemplu didactic: rețeaua neuronală rezultată



# Rețea cu 3 nivele



# Rețea cu 4 nivele



## Rețele: variația totală

Capacitatea de generalizare a rețelelor neuronale depinde de sensibilitatea ieșirii la perturbațiile de la intrare.

Fixăm ponderile  $w$  și calculăm iterativ variațiile ieșirii datorate unor diferențe mici aplicate intrării  $x^{(0)}$ :

$$\begin{aligned} dy &= \frac{dy}{ds^{(2)}} ds^{(2)} = \frac{dy}{ds^{(2)}} \frac{ds^{(2)}}{dx^{(1)}} dx^{(1)} \\ &= \frac{dy}{ds^{(2)}} \frac{ds^{(2)}}{dx^{(1)}} \frac{dx^{(1)}}{ds^{(1)}} ds^{(1)} \\ &= \frac{dy}{ds^{(2)}} \frac{ds^{(2)}}{dx^{(1)}} \frac{dx^{(1)}}{ds^{(1)}} \frac{ds^{(1)}}{dx^{(0)}} dx^{(0)} \end{aligned} \quad (33)$$

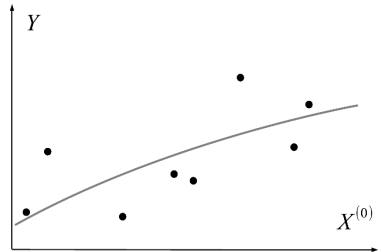
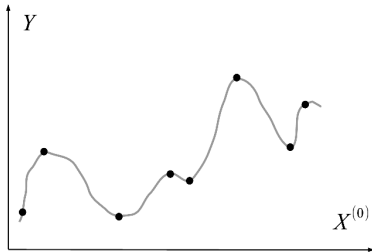
De unde, derivând fiecare termen în parte ajungem la:

$$dy = \varphi'(s^{(2)}) w^{(2)T} \varphi'(s^{(1)}) w^{(1)T} dx^{(0)} \quad (34)$$

Concluzie:  $dy$  depinde strict de ponderile  $w^{(\ell)}$ . Le menținem mici prin regularizare:  $\|w^{(\ell)}\|_{1,2} \leq 1$ .



# Proprietatea de generalizare



# Rețele: antrenarea

Antrenarea rețelelor neuronale se face în doi pași:

- ▶ pasul înainte sau pasul de propagare
- ▶ pasul înapoi sau pasul de retropropagare

Propagare:

- ▶ ponderile  $w$  sunt fixate
- ▶ calculăm ieșirea fiecărui neuron în funcție de intrare  
 $x_i^{(\ell)} = \varphi(s_i^{(\ell)}) = \varphi(w_i^{(\ell)} x^{(\ell-1)})$
- ▶ mergem de la primul strat către ultimul pentru a determina ieșirea rețelei  $y = x^{(L)}$  unde  $L$  este numărul total de straturi

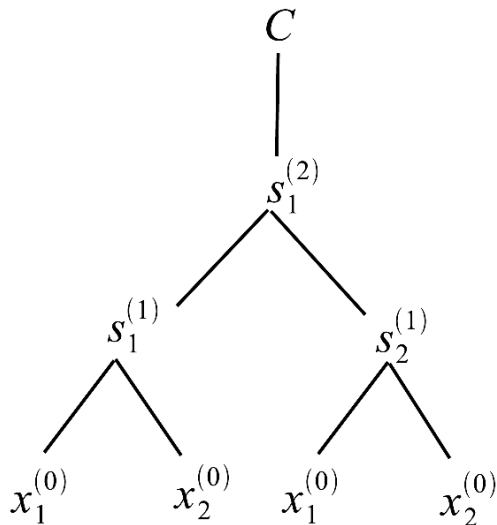
Retropropagare:

- ▶ ieșirile fiecărui neuron sunt fixate
- ▶ fiecare pondere este calculată dinspre ultimul strat către primul
- ▶ gradientul calculat în stratul  $\ell$  este necesar calculării gradientului stratului  $\ell - 1$





# Retropropagare: arborele de dependențe



## Rețele: retropropagare

Retropropagarea pleacă de la ultimul strat și calculează gradientul costului  $C$  în funcție de fiecare pondere  $w_{ij}^{(\ell)}$  din rețea.

Observăm că ponderea  $w_{ij}^{(\ell)}$  unește neuronul  $i$  din stratul  $\ell - 1$  de neuronul  $j$  din stratul  $\ell$  participând la semnalul  $s_j^\ell$  și la nici un alt semnal din stratul  $\ell$ .

$$\frac{\partial C}{\partial w_{ij}^{(\ell)}} = \frac{\partial C}{\partial s_j^{(\ell)}} \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \delta_j^{(\ell)} x_i^{(\ell-1)} \quad (35)$$

pentru că

$$\begin{aligned} \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} &= \frac{\partial}{\partial w_{ij}^{(\ell)}} \sum_{k=1}^{d^{(\ell-1)}} w_{kj}^{(\ell)} x_k^{(\ell-1)} = x_i^{(\ell-1)} \\ \delta_j^{(\ell)} &= \frac{\partial C}{\partial s_j^{(\ell)}} \end{aligned} \quad (36)$$

unde am notat cu  $d^{(\ell)}$  numărul de neuroni de pe stratul  $\ell$ .



# Retropropagare: calculul $\delta$

Pornind cu costul  $C$  din ultimul strat  $L$

$$C = \frac{1}{2} \sum_{i=1}^{d^{(L)}} \left( x_j^{(L)} - z_j \right)^2 = \frac{1}{2} \left\| x^{(L)} - z \right\|^2 = \frac{1}{2} \left\| \varphi(s^{(L)}) - z \right\|^2 \quad (37)$$

unde

$$\frac{1}{2} \left\| \varphi(s^{(L)}) - z \right\|^2 = \frac{1}{2} \sum_{j=1}^L \left( \varphi(s_j^{(L)}) - z_j \right)^2$$

avem

$$\delta_j^{(L)} = \frac{\partial C}{\partial s_j^{(L)}} = (x^{(L)} - z) \varphi'(s_j^{(L)}) \quad (38)$$

Observăm că semnalul  $s_j^{(\ell-1)}$  din stratul  $\ell - 1$  afectează toate semnalele  $s_j^{(\ell)}$  cu  $1 \leq j \leq d^{(\ell)}$ :

$$\delta_i^{(\ell-1)} = \frac{\partial C}{\partial s_i^{(\ell-1)}} = \sum_{j=1}^{d^{(\ell)}} \frac{\partial C}{\partial s_j^{(\ell)}} \frac{\partial s_j^{(\ell)}}{\partial s_i^{(\ell-1)}} = \sum_{j=1}^{d^{(\ell)}} \delta_j^{(\ell)} \frac{\partial s_j^{(\ell)}}{\partial s_i^{(\ell-1)}} \quad (39)$$



# Retropropagare: calculul $\delta$

Calculăm sensibilitatea semnalului din stratul  $\ell$  față de stratul precedent:

$$\begin{aligned}\frac{\partial s_j^{(\ell)}}{\partial s_i^{(\ell-1)}} &= \frac{\partial}{\partial s_i^{(\ell-1)}} \sum_{j=1}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_j^{(\ell-1)} \\ &= \frac{\partial}{\partial s_i^{(\ell-1)}} \sum_{j=1}^{d^{(\ell-1)}} w_{ij}^{(\ell)} \varphi(s_j^{(\ell-1)}) \\ &= w_{ij}^{(\ell)} \varphi'(s_i^{(\ell-1)})\end{aligned}\tag{40}$$

și reintroducem în formula lui  $\delta_i^{(\ell-1)}$ :

$$\delta_i^{(\ell-1)} = \varphi'(s_i^{(\ell-1)}) \sum_{j=1}^{d^{(\ell)}} \delta_j^{(\ell)} w_{ij}^{(\ell)}\tag{41}$$



Rețelele neuronale sunt antrenate pe setul de date alcătuit din  $n$  puncte astfel încât la final ponderile optime  $w$  sunt:

$$w_{ij}^{(\ell)}(n+1) = w_{ij}^{(\ell)}(n) - \alpha \delta_j^{(\ell)}(n) x_i^{(\ell-1)}(n) \quad (42)$$

unde  $\alpha \in \mathbb{R}$  este rata de învățare și  $\delta_j^{(\ell)} = \frac{\partial C}{\partial s_j^{(\ell)}}$ .

Observăm că  $x_i^{(\ell)}$  și  $\delta_j^{(\ell)}$  depind de ponderile  $w_{ij}^{(\ell)}$  obținute la pasul anterior  $n$ .



Scrierea compactă în formă matricială devine:

$$x^\ell = \varphi(W^{(\ell)T} x^{(\ell-1)}) \quad (43)$$

$$\delta^L = (x^{(L)} - z) \odot \varphi'(s^{(L)}) \quad (44)$$

$$\delta^{\ell-1} = W^{(\ell)} \delta^{(\ell)} \odot \varphi'(s^{(\ell-1)}) \quad (45)$$

$$\frac{\partial C}{\partial W^{(\ell)}} = x^{(\ell-1)} \delta^{(\ell)T} \quad (46)$$

unde  $\odot$  este produsul Hadamard a doi vectori i.e.  $(a \odot b)_i = a_i b_i$ .



# Rețele: problema plafonării gradientului

- ▶ fie un semnal  $x$  cu magnitudine mare
- ▶ în acest caz anumite funcții de activare saturează rapid gradientul  $\delta_j$
- ▶ gradientul este micșorat ceea ce conduce la pași de gradient mici
- ▶ pașii mici de gradient conduc la o convergență lentă
- ▶ dacă pașii devin aproape nuli, apare fenomenul de plafonare
- ▶ funcția logistică standard este o astfel de funcție



- ▶ rare ori putem antrena pe toate datele din setul de date deodată (i.e. să folosim matrice de dimensiuni mari la intrare)
- ▶ dacă folosim câte un singur punct la intrare metoda GD converge lent
- ▶ compromis: folosim loturi de puncte (i.e. blocuri mici din matricea mare)
- ▶ variantă: alegem aleator puncte din setul de date și calculăm gradientii pe fiecare direcție după care facem o medie – denumit în practică Stochastic Gradient Descent (SGD)





