2NClub #1

2N Club

My First Neural Network

CLUB

Cadette
for the love of code

Part I

The Math behind a Neural Network

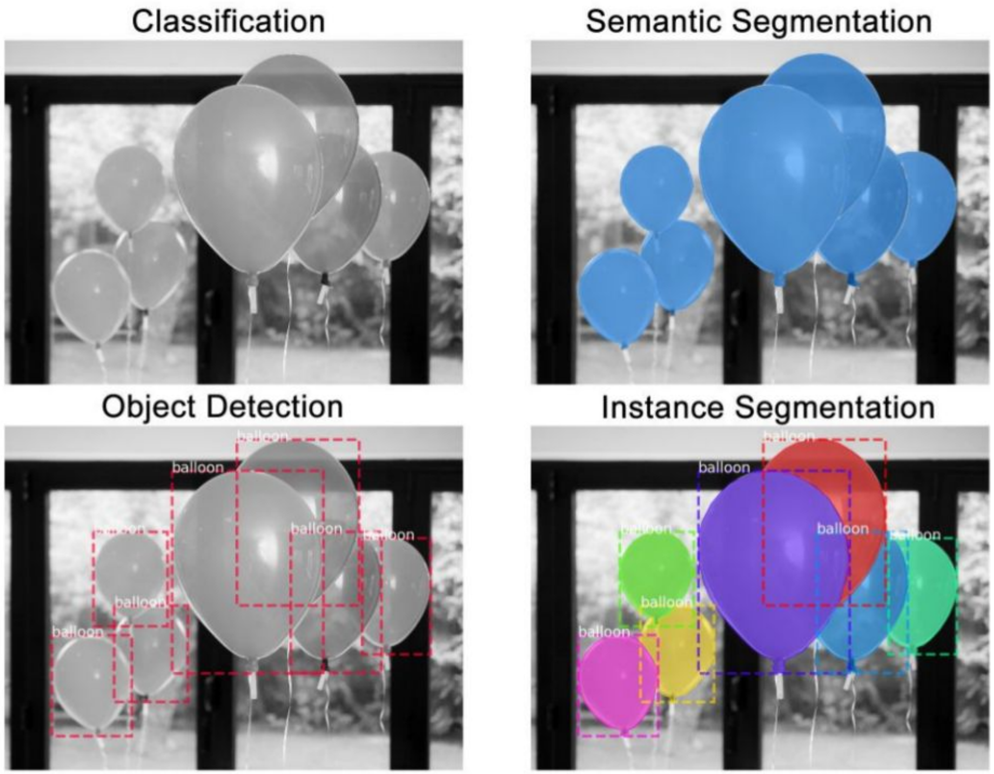
1.1. Modele liniare

Andrei Nicolicioiu

Machine Learning Researcher

@ Bitdefender

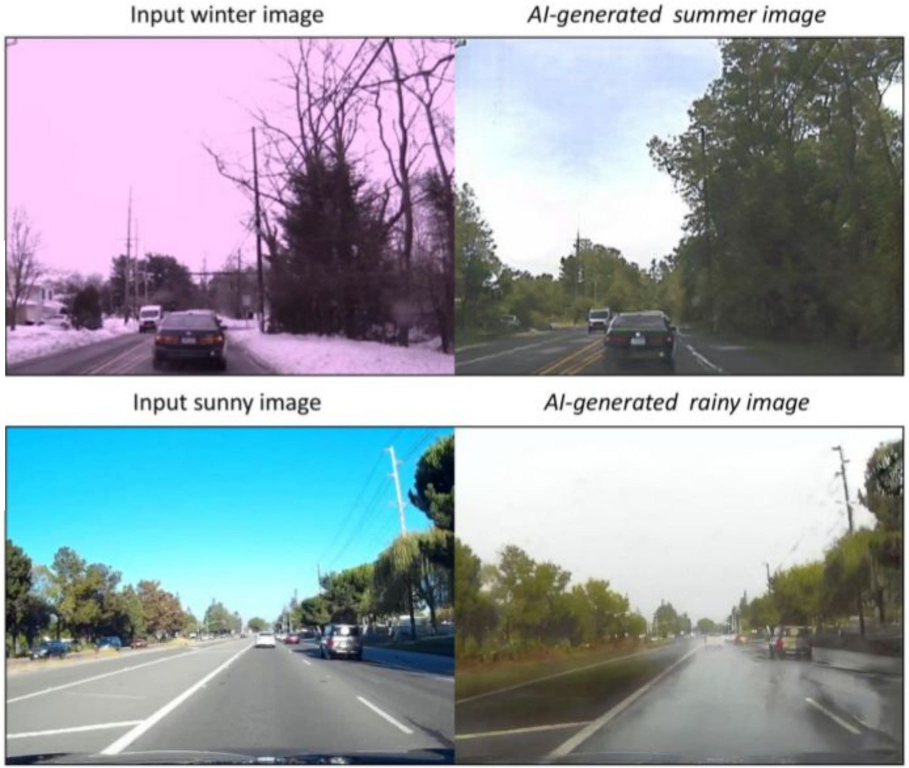
Clasificare - Detecție - Segmentare



Generare Imagini

(Brundage et al, 2018)

Adaptare



- CycleGAN

Captioning

Generated:

0.650. a group of people are sitting in a line with a tiger

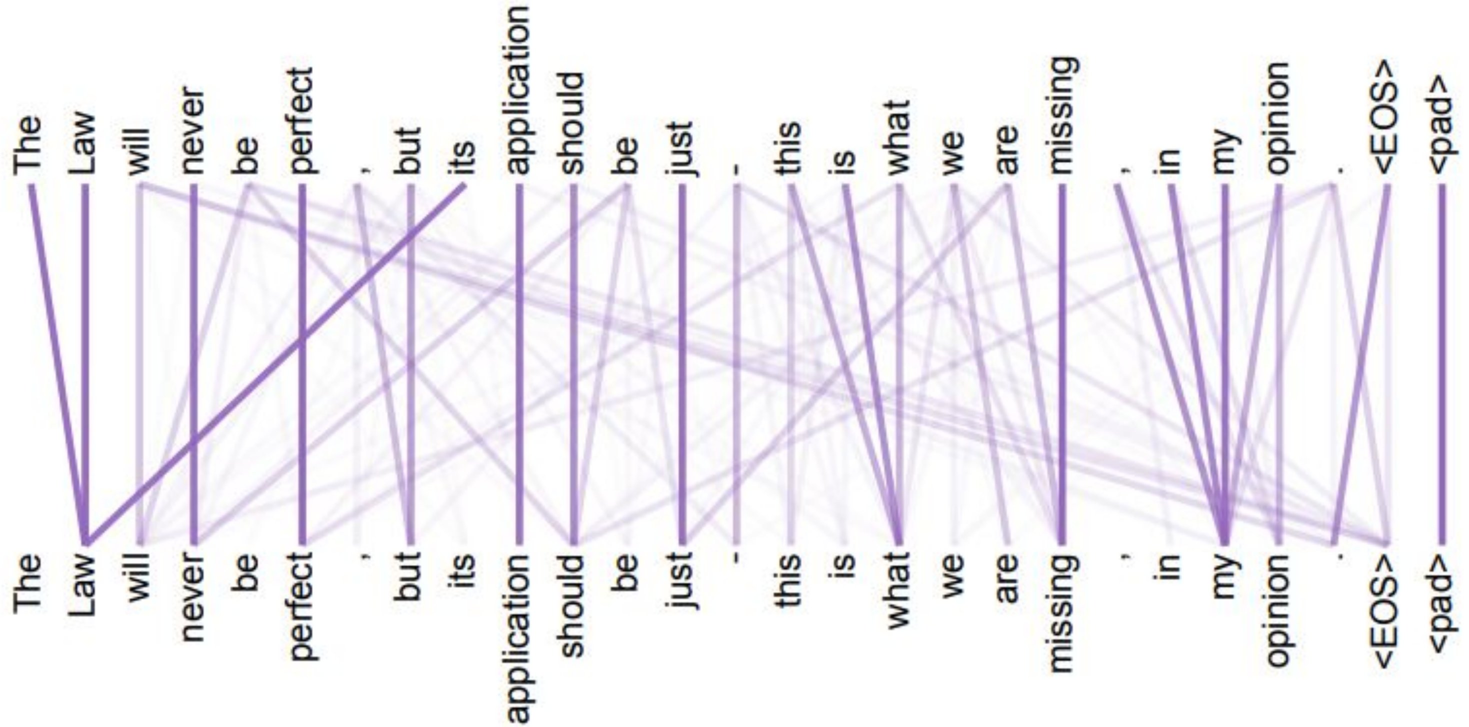
0.599. a man is sitting in a chair with a tiger

0.588. a man and a woman are eating a tiger in a bowl

0.547. a man is talking about a tiger

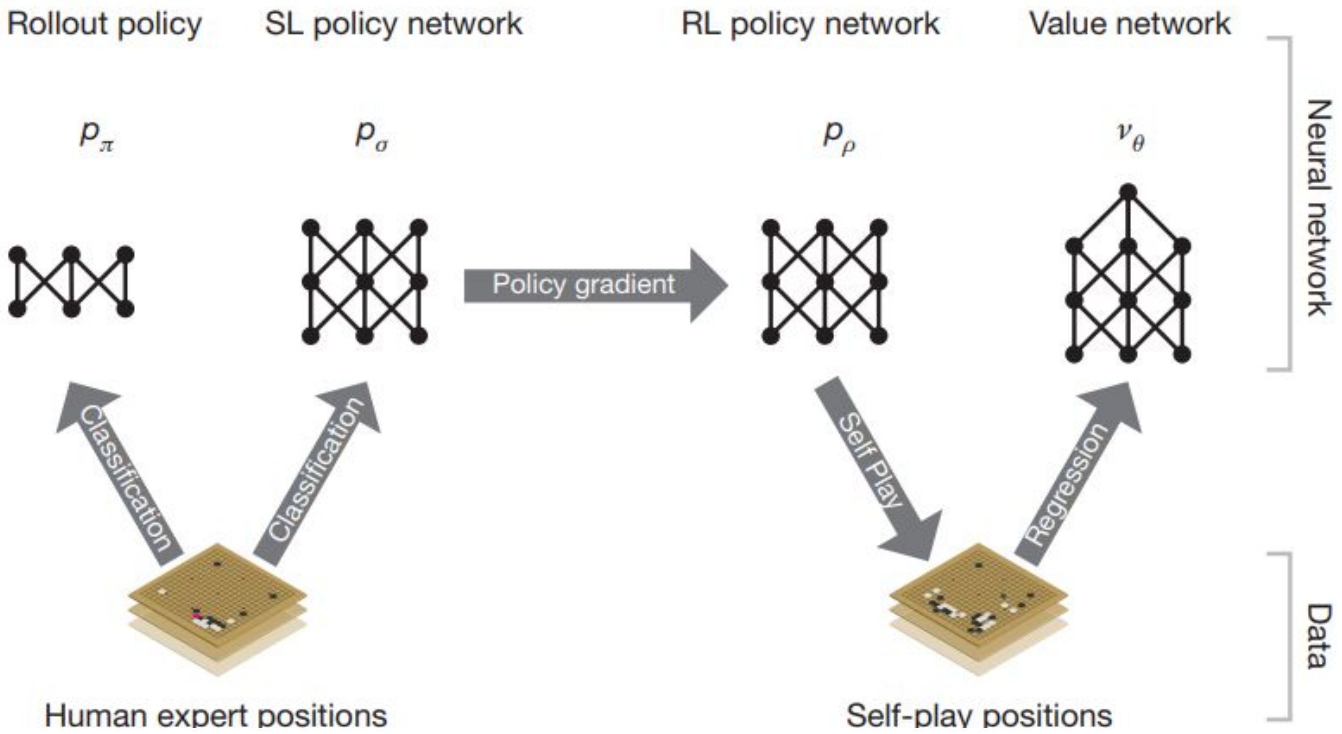
0.180. a man and a woman are sitting in a table

Analiză text - Traduceri



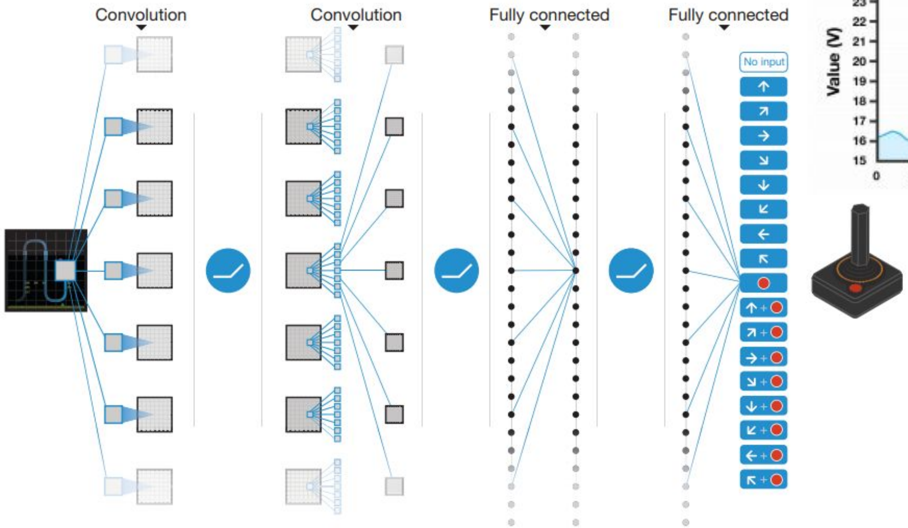
- Vaswani et al - Attention os all you need [2017]

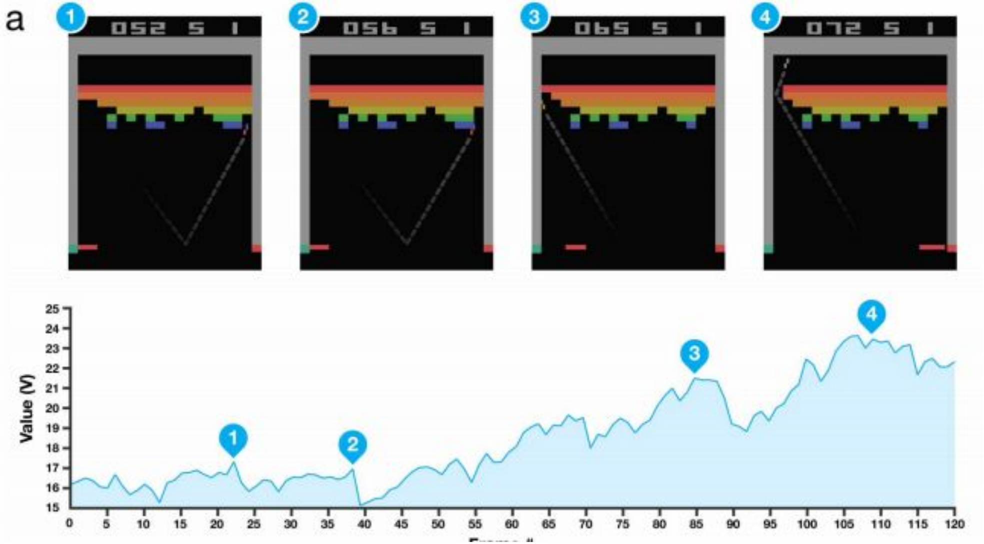
Alpha Go



- Alpha Go - 2016

Reinforcement Learning





V. Mnih et. all - DQN - 2015

De ce Machine Learning?

- Problemă clasică: Recunoașterea cifrelor scrise de mână

De ce Machine Learning?

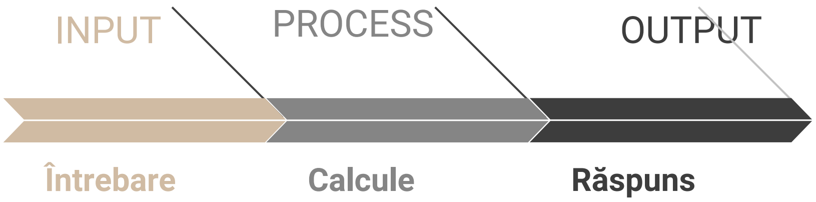
- Vrem să recunoaștem cifre scrise de mână. Cum procedăm?

- Găsim reguli pentru fiecare cifră. Exemplu:
  - pentru cifra 8 găsim 2 cercuri de aceeași dimensiune așezate unul peste altul
  - cunoaștem ecuația cercului: x^2 + y^2 = R
  - găsim toate cercurile, le păstram doar pe cele așezate unul peste altul

- Probleme:
  - foarte multe reguli greu de identificat
  - varietate foarte mare in date
  - nu pot găsi reguli care să acopere toate posibilitățile

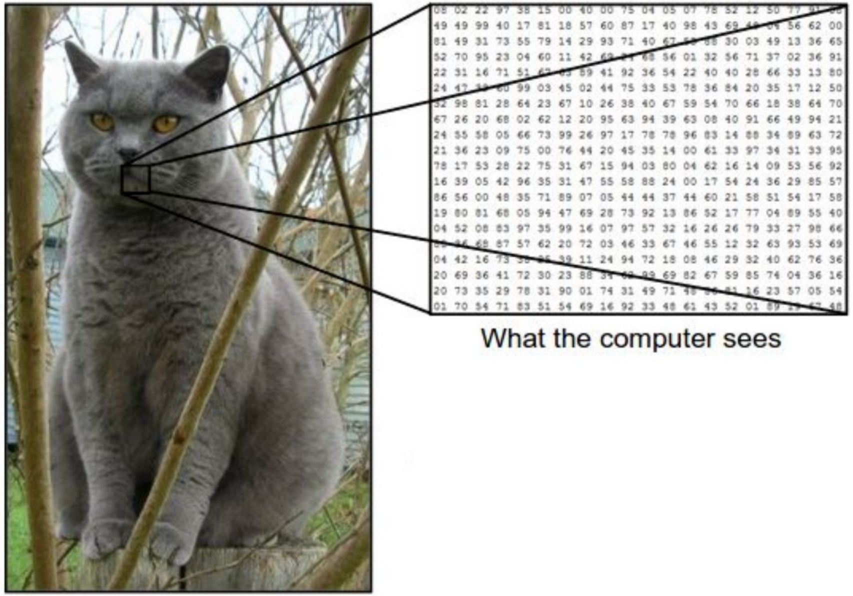
Machine Learning

- Scop:
  - Crearea unui program capabil să învețe automat
- Mijloace:
  - **Dataset:** perechi **input - target**
  - **Procesare** input
  - La ieșire vom avea un **rezultat**
- Urmărim 3 lucruri:
  - Ce primeste programul la input: **reprezentare**
  - Care sunt operațiile de procesare: **model**
  - Cum învață modelul : **optimizare**

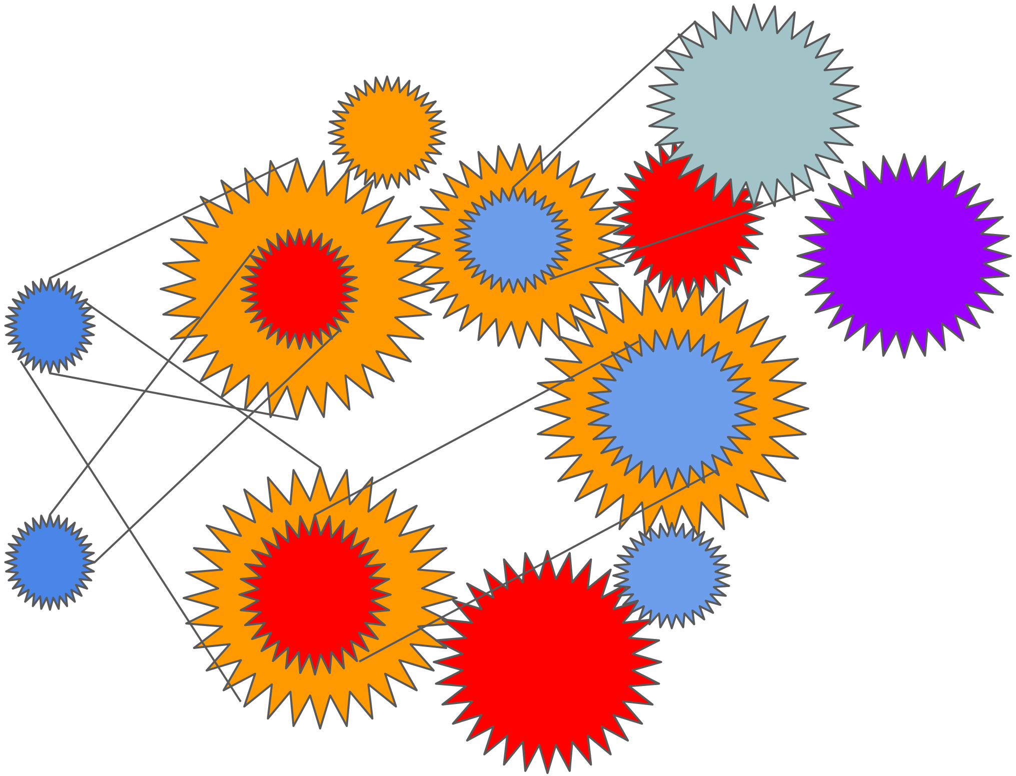


Human vs Computer

- Procesare de date
  - => **reprezentare**
- Din date neprelucrate (pixeli) extragem niște reprezentări, caracteristici - **features**
- Putem folosi direct datele neprelucrate și vom **învăța** automat reprezentări



Model - Analogie



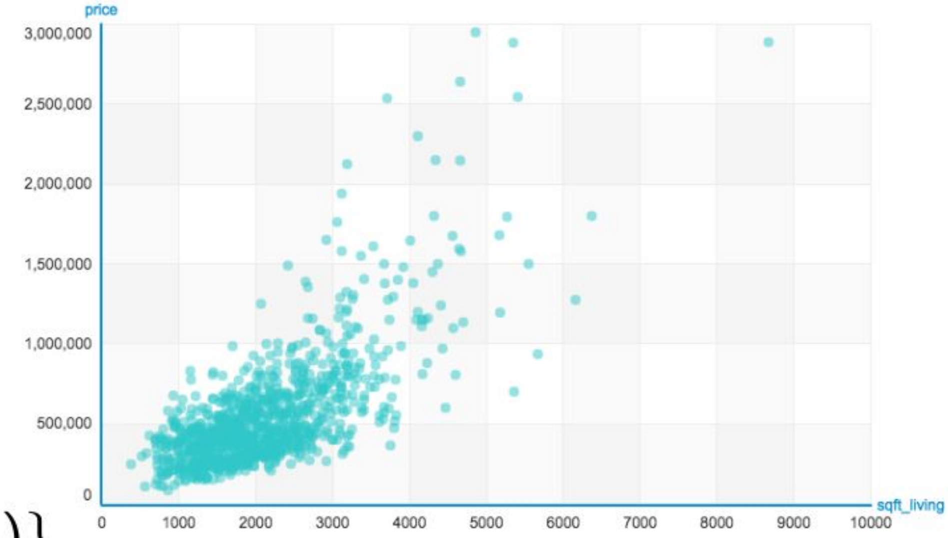
- Un model are nevoie de:
- **Arhitectura** (cum sunt conectate roțile)
- **Parametrii**: (dimensiunea roților, cu cât amplifică)

- Pentru o arhitectura aleasa vrem un mode capabil să **învețe** singur parametrii

Regresie

- Problemă:
    - **Prezicerea prețului de vânzare a unei locuințe în funcție de suprafață**
- Set de date despre locuințe:

$$D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_M, t_M)\}$$



    - **Reprezentare**: suprafața x
    - **Target**: prețul adevărat al apartamentului
- Scop:
    - Folosirea unui **model** adecvat pentru a putea prezice prețul
    - **Optimizarea** (învățarea) modelului cu ajutorul setului de date

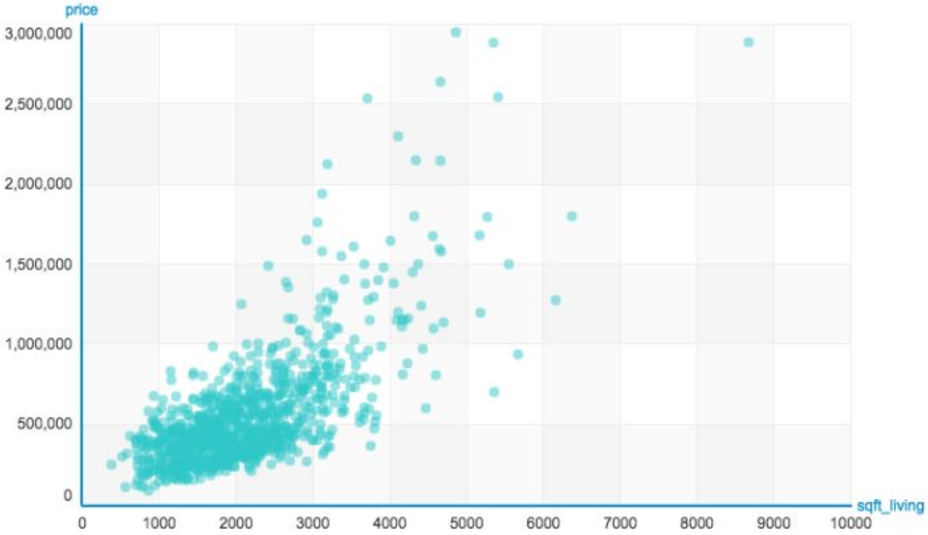
Regresie

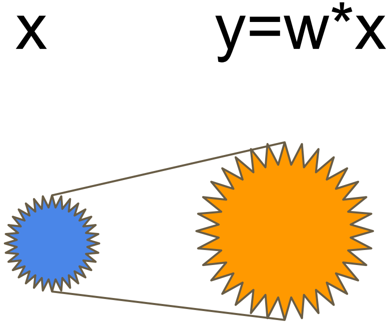
Model Liniar

- Reprezentare: x \in R
- Cel mai simplu model:

$$y = w \cdot x + b$$

- Analogie cu o roată zimțată
- Găsește parametrul (raza roții) a.î. prețul prezis pentru orice casă sa fie cat mai apropiat de cel real





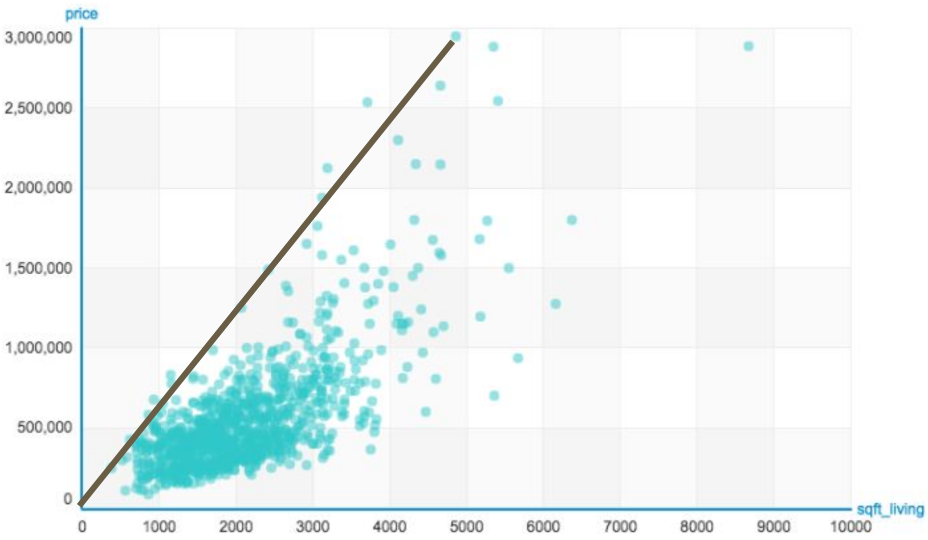
Regresie

Model Liniar

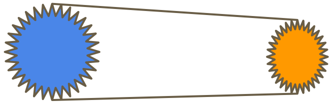
- Reprezentare: x \in R
- Cel mai simplu model:

$$y = w \cdot x + b$$

- Analogie cu o roată zimțată
- Găsește parametrul (raza roții) a.î. prețul prezis pentru orice casă sa fie cat mai apropiat de cel real



x y=w*x



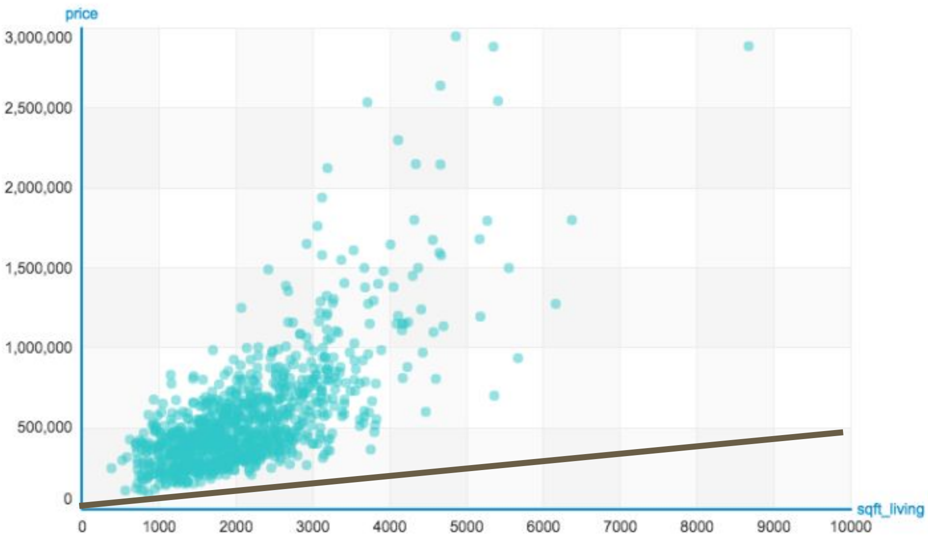
Regresie

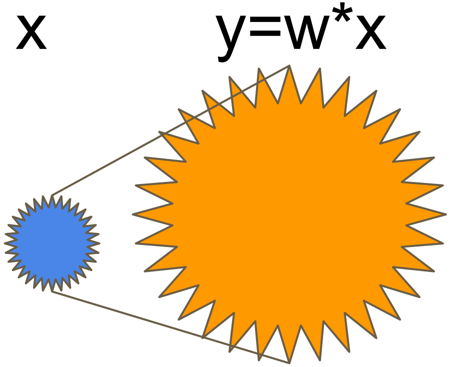
Model Liniar

- Reprezentare: x \in R
- Cel mai simplu model:

$$y = w \cdot x + b$$

- Analogie cu o roată zimțată
- Găsește parametrul (raza roții) a.î. prețul prezis pentru orice casă sa fie cat mai apropiat de cel real





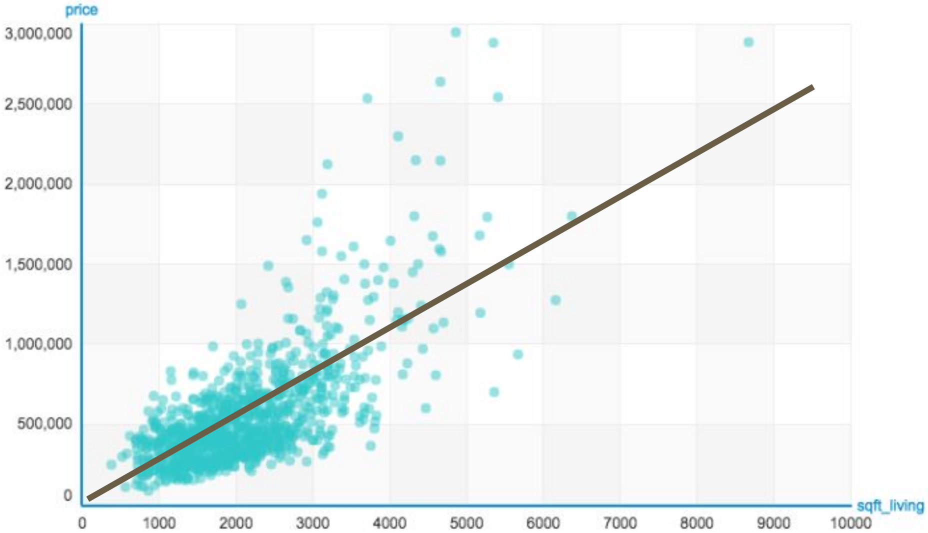
Regresie

Model Liniar

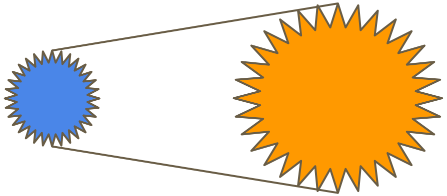
- Reprezentare: x \in R
- Cel mai simplu model:

$$y = w \cdot x + b$$

- Analogie cu o roată zimțată
- Găsește parametrul (raza roții) a.î. prețul prezis pentru orice casă sa fie cat mai apropiat de cel real



x y=w*x



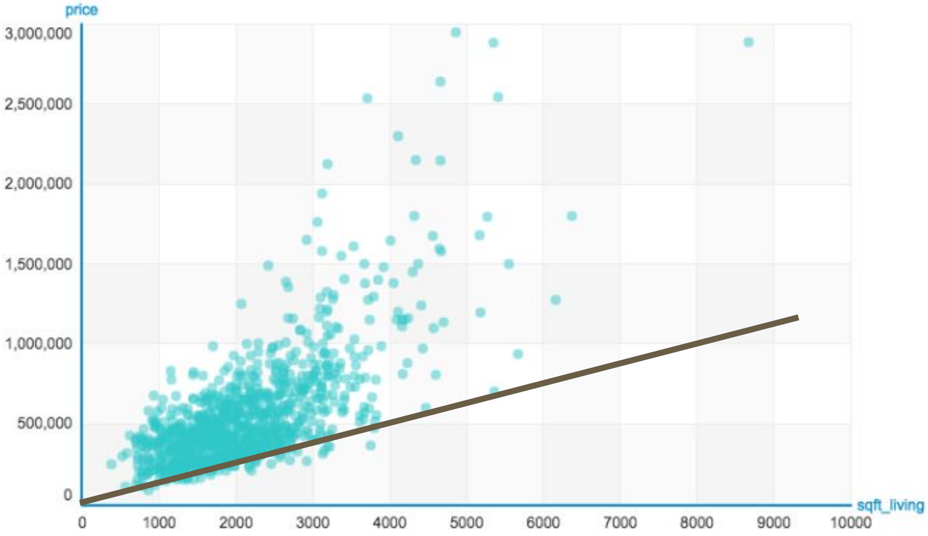
Regresie

Optimizare

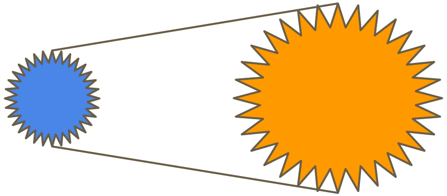
- Ponderi aleatoare
- Funcție de eroare(cost), cât de prost prezice modelul nostru baza de date:

$$E = \sum_i |t_i - y_i|$$

- Ajustăm parametrii
- Păstrăm parametrii care minimizează eroarea



x y=w*x

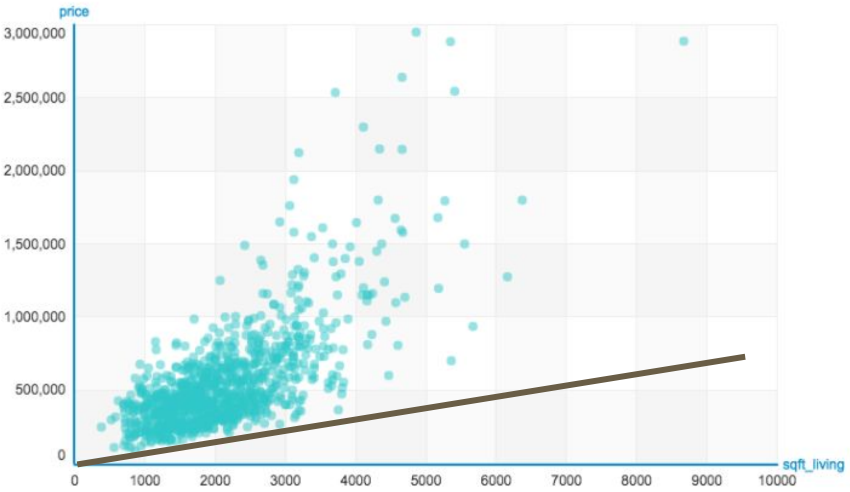


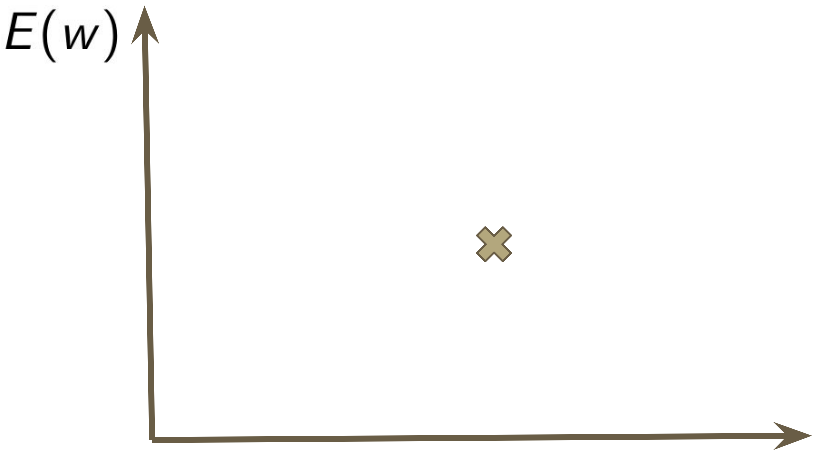
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente



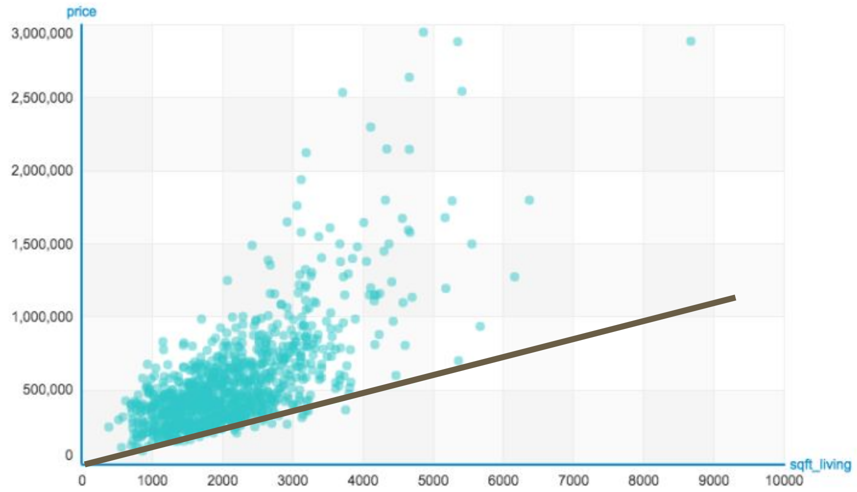


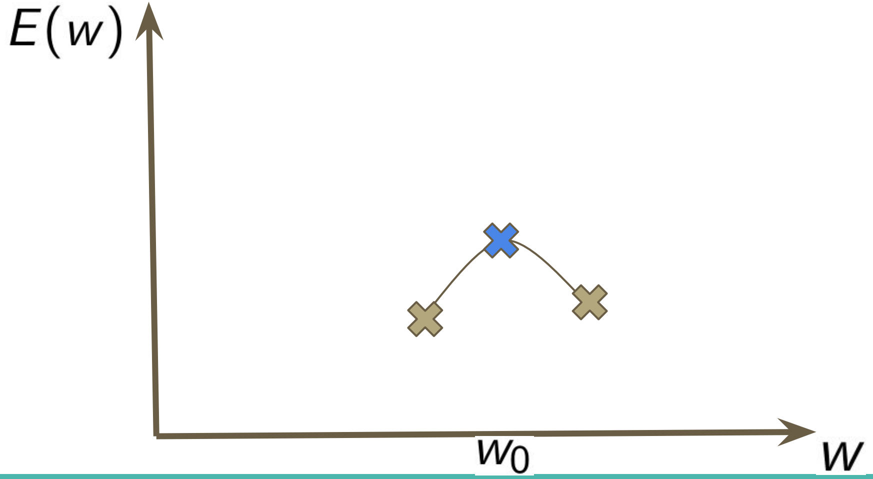
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente



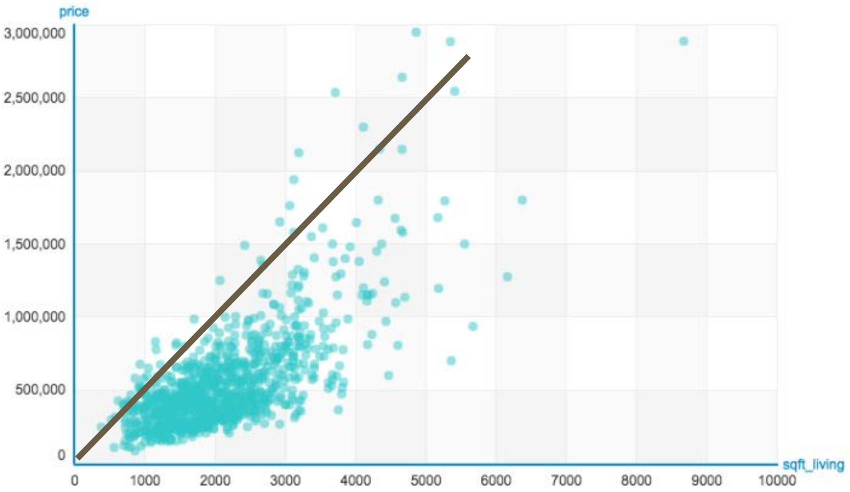


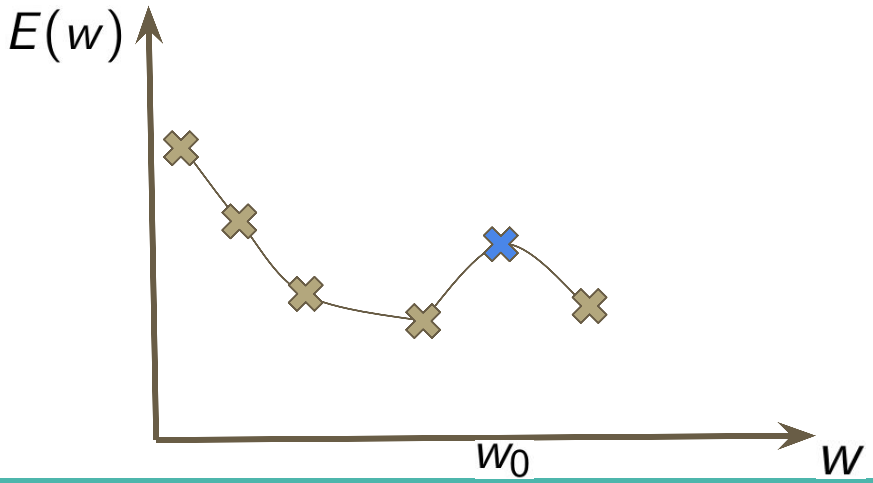
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente



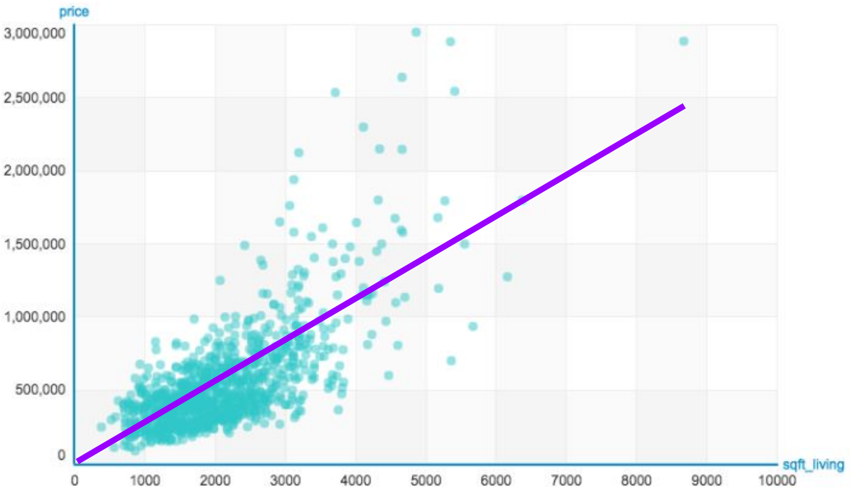


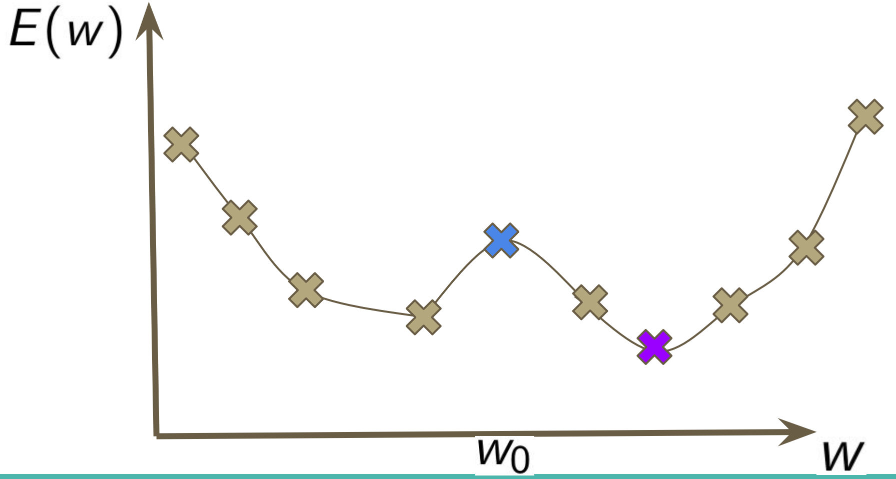
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente



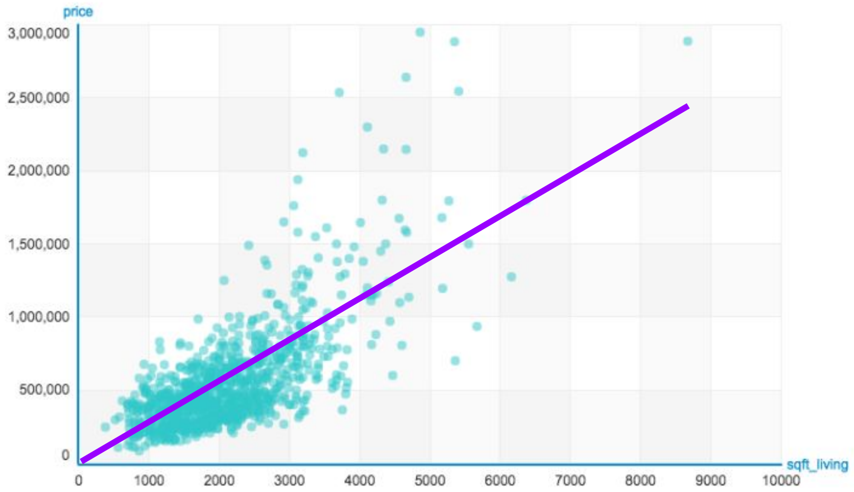


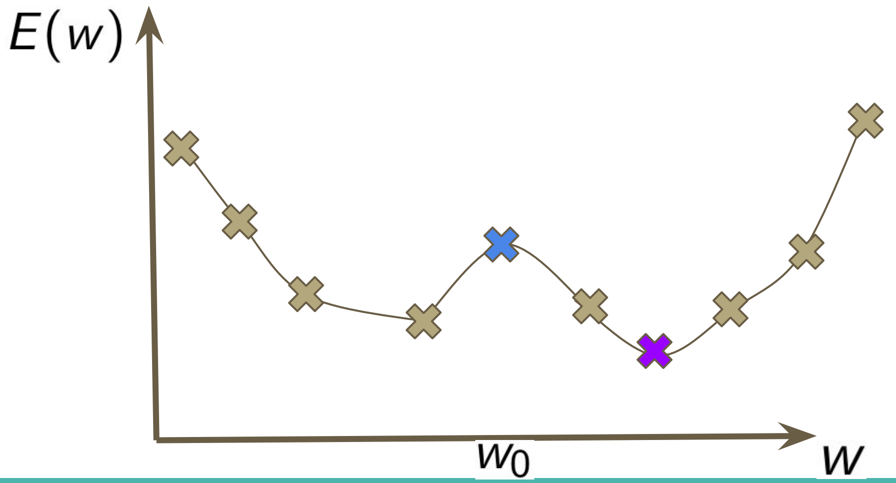
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente
- Ineficient!
  - Calcule pentru fiecare variantă



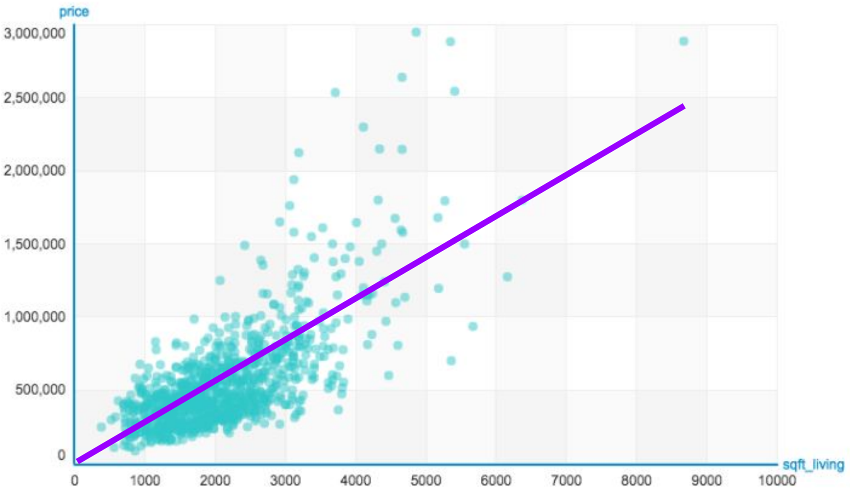


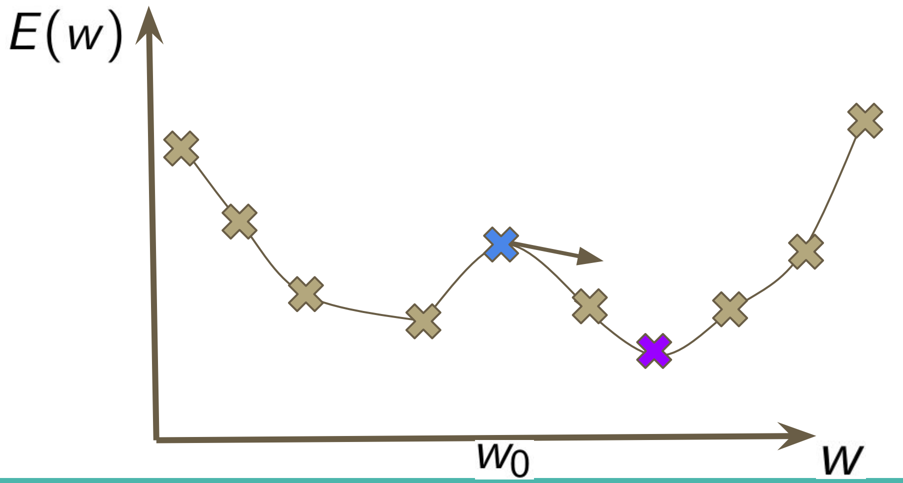
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente
- Ineficient!
  - Calcule pentru fiecare variantă
- Soluție: urmăm panta funcției



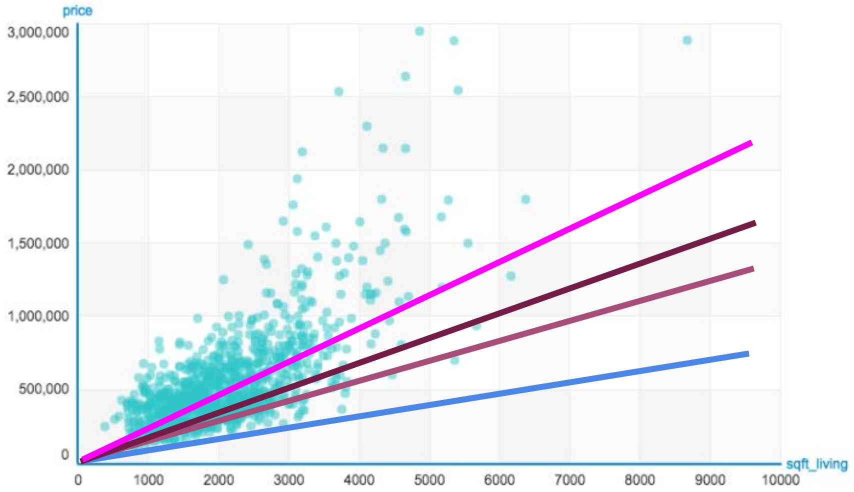


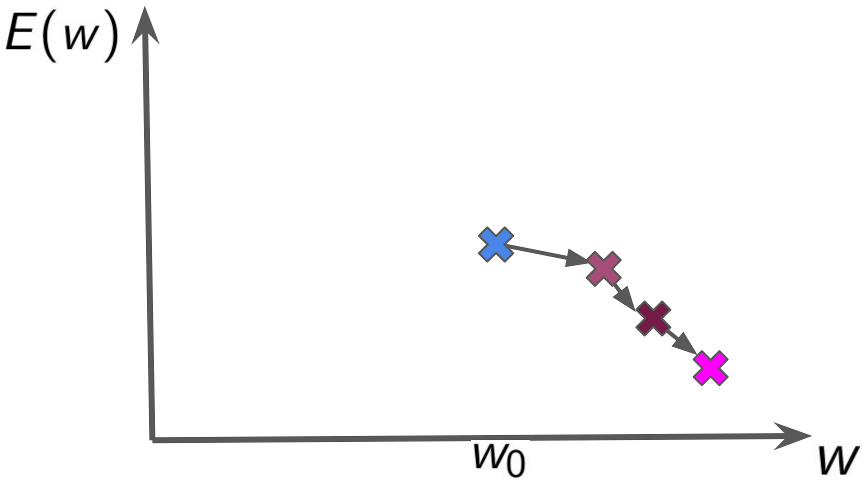
Regresie

Optimizare

Varianta 1:

- Încercăm multe valori pentru parametrii w_0 în jurul valorii curente
- Ineficient!
  - Calcule pentru fiecare variantă
- Soluție: urmăm panta funcției
  - În punctul curent w_0 calculăm panta funcției de eroare
  - Găsim parametri noi urmând această direcție
  - Necesită câțiva pași





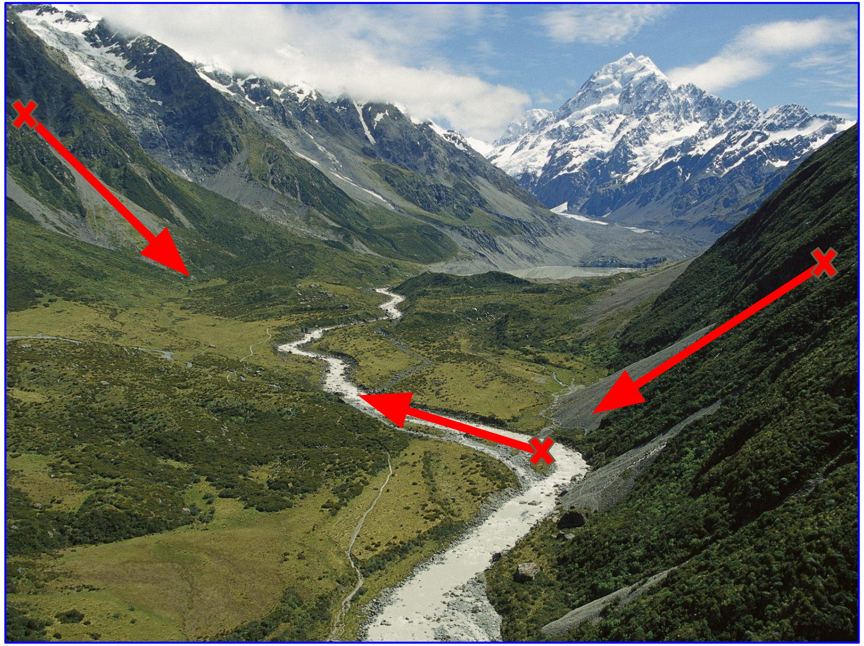
Gradient descent

Scop:

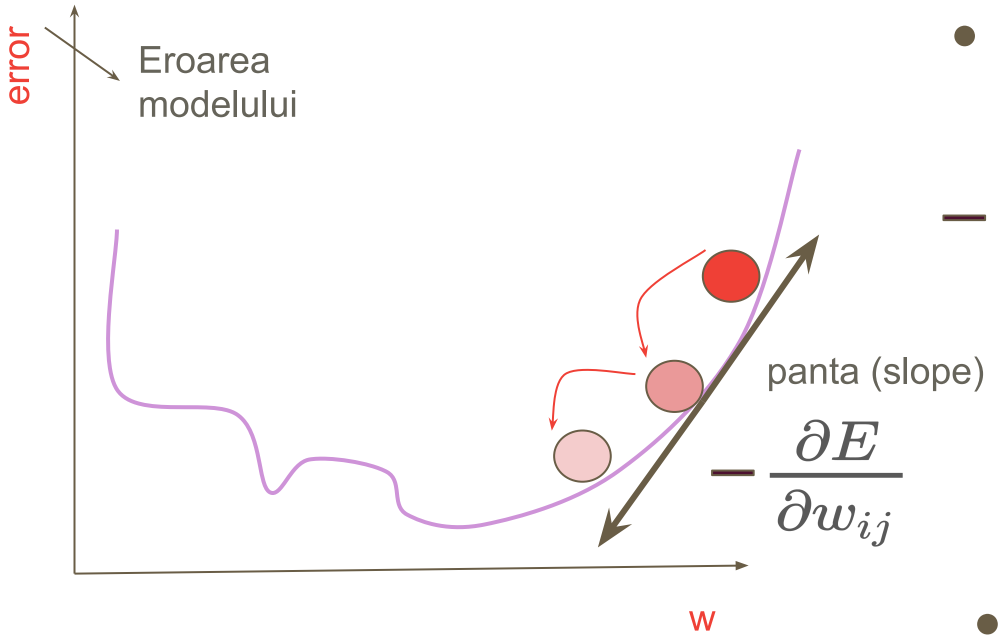
- găsirea minimului unei funcții.

Soluție:

- urmarea “pantei”
- Panta = direcția în care funcția scade cel mai brusc



Gradient descent



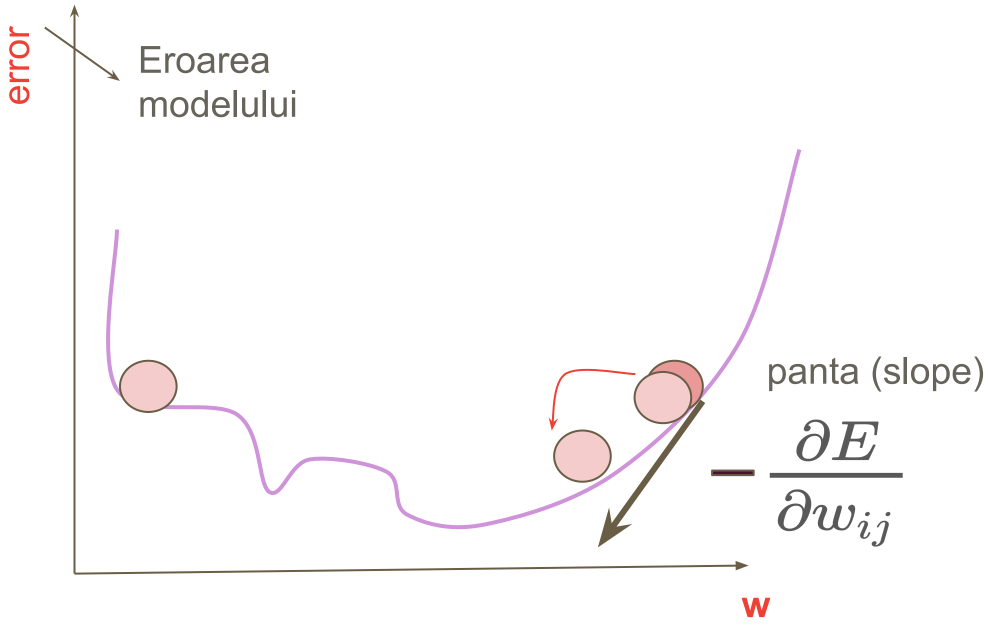
- Căutăm direcția în care E scade

$$-\quad E(w+\Delta w)-E(w)$$

- Panta = Derivată = Gradient

Gradient descent

$$-\lim_{\Delta x \to 0} \frac{E(w+\Delta w) - E(w)}{\Delta w} = \frac{\mathrm{d}E}{\mathrm{d}w}$$



$$-\frac{\partial E}{\partial w_{ij}}$$

$$w = w - \alpha \frac{\mathrm{d}E}{\mathrm{d}w}$$

Learning rate

- Alpha mic -> învățare prea lentă
- Alpha mare -> risc să sar peste puncte de minim

Regresie liniară

- Avem dat un dataset

$$D = \{(x_1, t_1), (x_2, t_2), ..., (x_M, t_M)\}$$

- Stabilim o funcție de cost / eroare E
- Căutăm parametrii unei funcții f care să aibă eroare cât mai mică

$$y = f(x) = w \cdot x + b$$

Regresie liniară

1. Inițializăm w random
2. Calculăm ieșirile:

$$y_i = w \cdot x_1 + b$$

3. Calculăm eroarea pe întreg datasetul:

$$E = \frac{1}{2}\sum_{i}^{M}(t_i - y_i)^2$$

4. Actualizăm parametrii:

$$w = w - \alpha \frac{\mathrm{d}E}{\mathrm{d}w}$$

5. Reluam pașii 2-3 cât timp eroarea scade

Clasificare

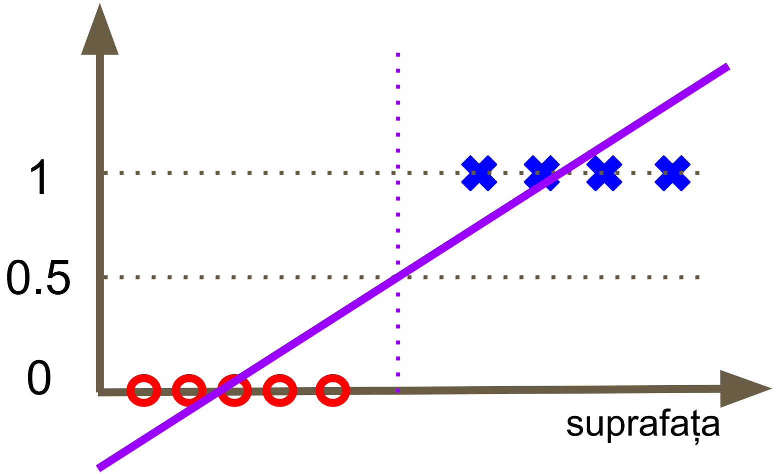
- Avem de ales între două variante.
- De exemplu: știm doar suprafața unui apartament. Cum decidem dacă are 2 sau 3 camere?

Clasificare

- Avem de ales între două variante.
- De exemplu: știm doar suprafața unui apartament. Cum decidem dacă are 2 sau 3 camere?

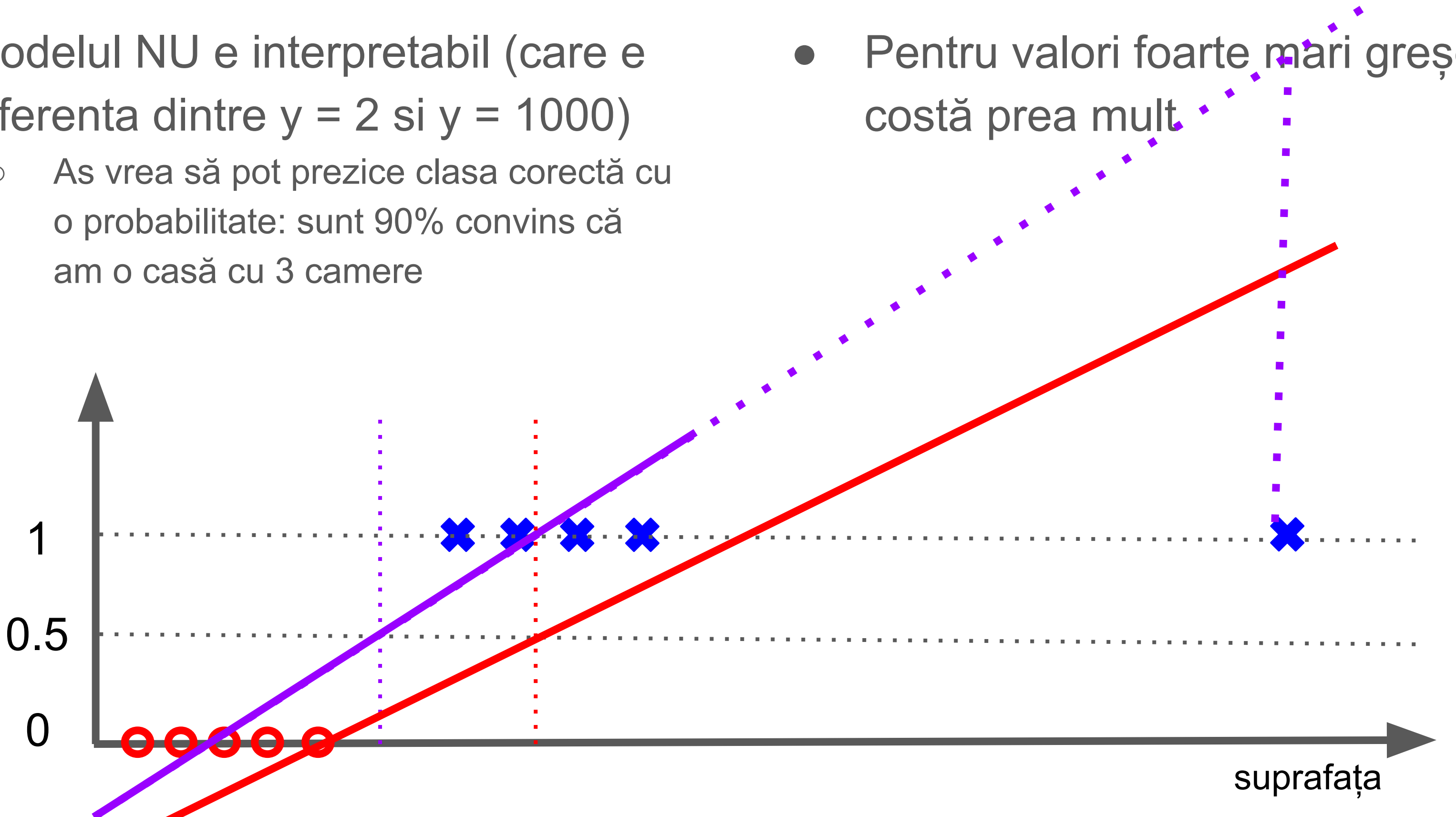
Varianta 1:

1. Colectăm date:
   a. y = 0 => două camere
   b. y = 1 => trei camere
2. Aplicăm regresie liniară
   a. Toate valorile y > 0.5 vor fi clasificate ca exemplu pozitiv (3 camere).



Clasificare - probleme

- Modelul NU e interpretabil (care e diferenta dintre y = 2 si y = 1000)
  - As vrea să pot prezice clasa corectă cu o probabilitate: sunt 90% convins că am o casă cu 3 camere
- Pentru valori foarte mari greșelile costă prea mult

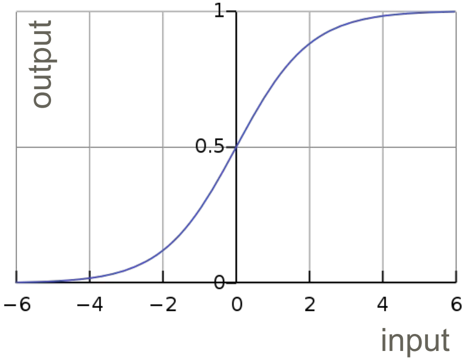


Clasificare - Soluție

- Vom folosi funcția sigmoid
- Modelul nostru devine

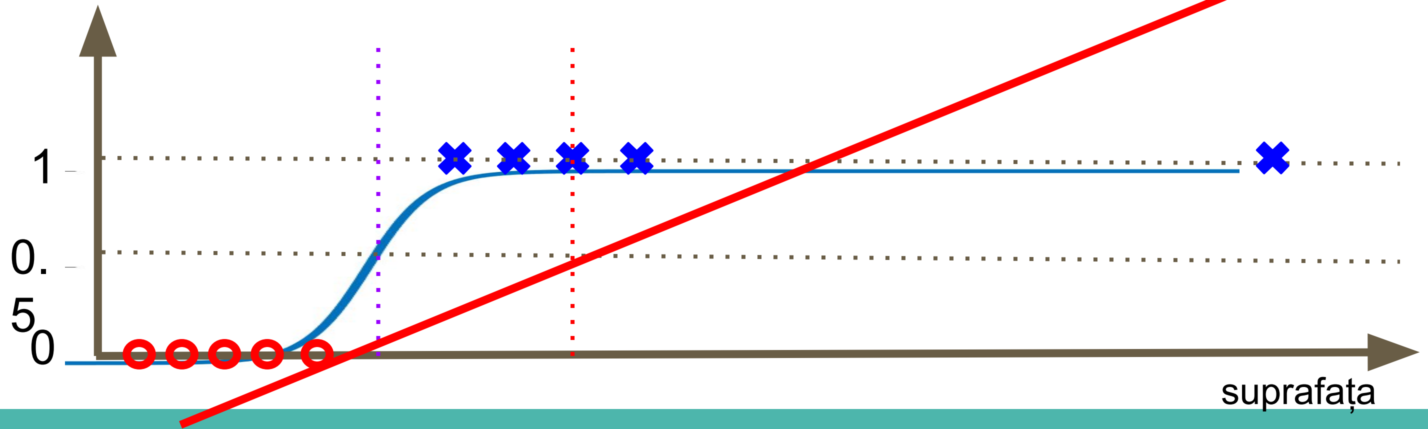
  a = sigmoid(w * x + b)

Funcția sigmoidă



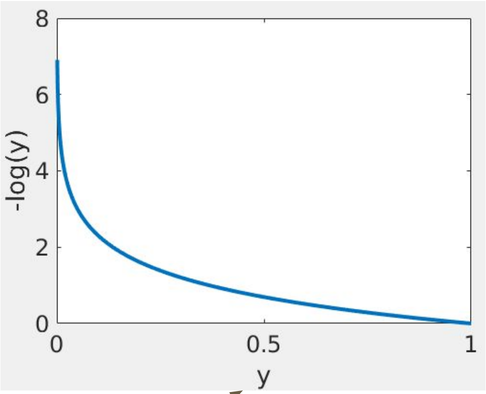
$$y = \frac{1}{1+e^{-x}}$$

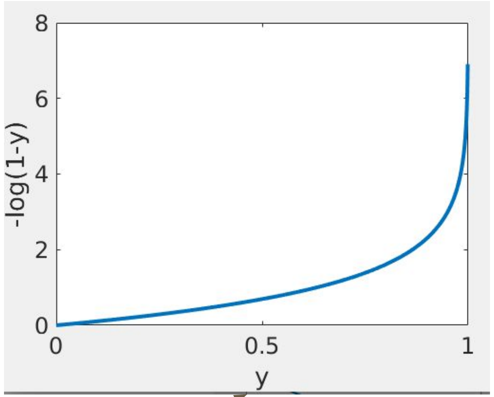
- Toate valorile mari se comportă la fel: sunt apropiate de 1
- Modelul nostru ia valori intre 0 (2 camere) si 1 (3 camere)



Clasificare - Logistic Regression

- Aceeasi pași ca la regresie liniară:
  - Inițializez parametrii
  - Calculăm ieșirile
  - Calculăm eroarea
  - Updatăm parametrii
  - Repetăm
- Ce funcție de eroare folosim?
  - În cazul regresiei liniare am folosit loss-ul L2:
- Pentru clasificare folosim cross-entropy:



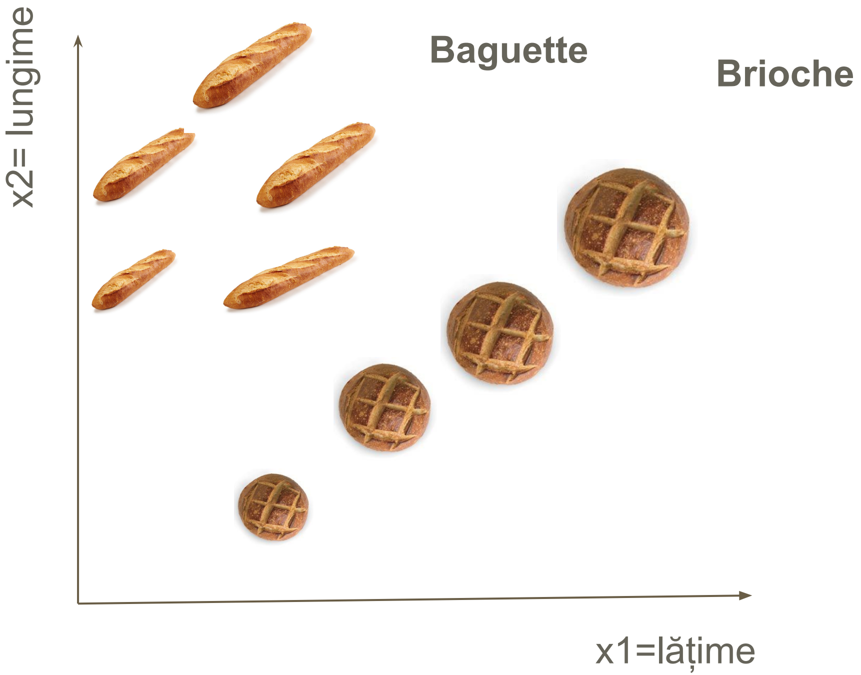


ti = 1

ti = 0

$$E = -\frac{1}{M}\sum_{i}^{M}(t_i \log a_i + (1 - t_i)\log(1 - a_i))$$

Clasificare - Exemplu

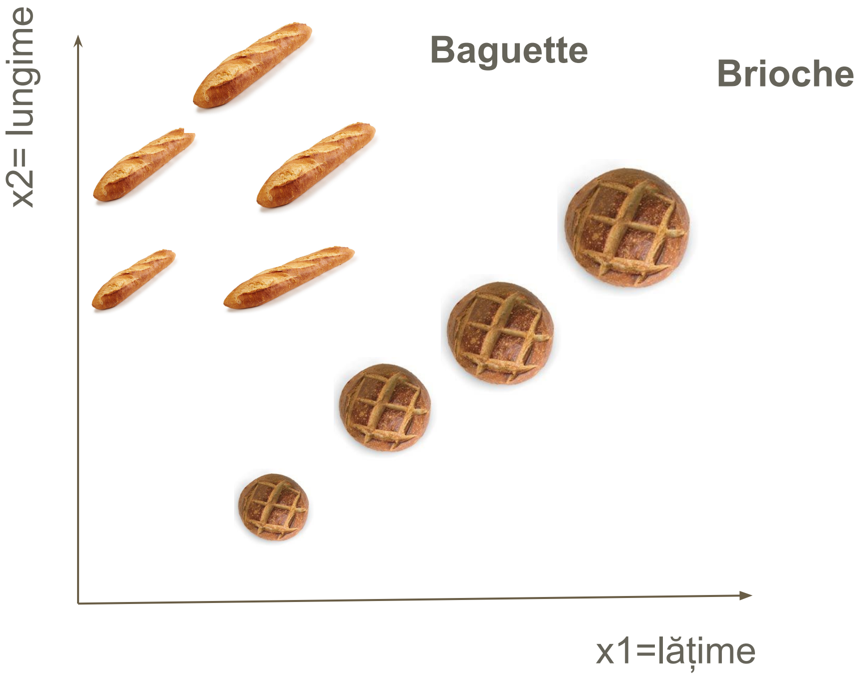


- Ce se întâmplă dacă inputul e caracterizat de mai multe dimensiuni?
- Inputul are dimensiune mai mare: $\mathbb{R}^2$
- Trebuie să învățăm o funcție cu mai mulți parametrii

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2 \qquad W = [w_1 w_2]$$

$$\begin{aligned} y(x) &= w_1 \cdot x_1 + w_2 \cdot x_2 + b \\ &= [w_1 w_2] \begin{bmatrix} x1 \\ x_2 \end{bmatrix} + b \\ &= W\, x + b \end{aligned}$$

Clasificare - Exemplu



- Ce se întâmplă dacă inputul e caracterizat de mai multe dimensiuni?
- Inputul are dimensiune mai mare: $\mathbb{R}^N$
- Trebuie să învățăm o funcție cu mai mulți parametri

$$x = \begin{bmatrix} x1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N \qquad W = [w_1 w_2 \ldots w_N]$$

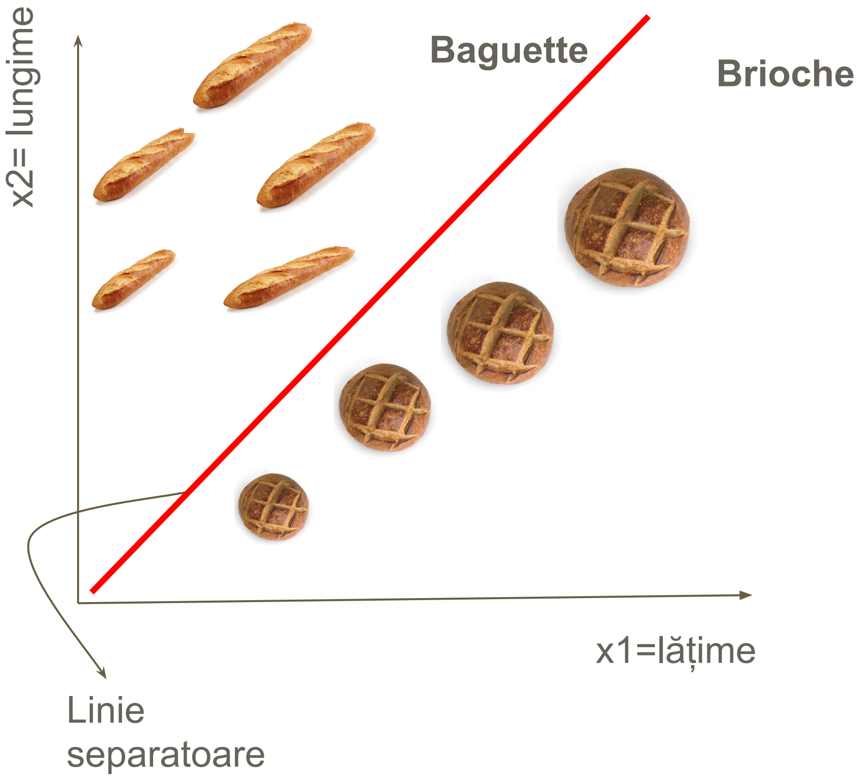
$$\begin{aligned} y(x) &= w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_N \cdot x_N + b \\ &= [w_1 w_2 \ldots w_N] \begin{bmatrix} x1 \\ \vdots \\ x_N \end{bmatrix} + b \\ &= W\,x + b \end{aligned}$$

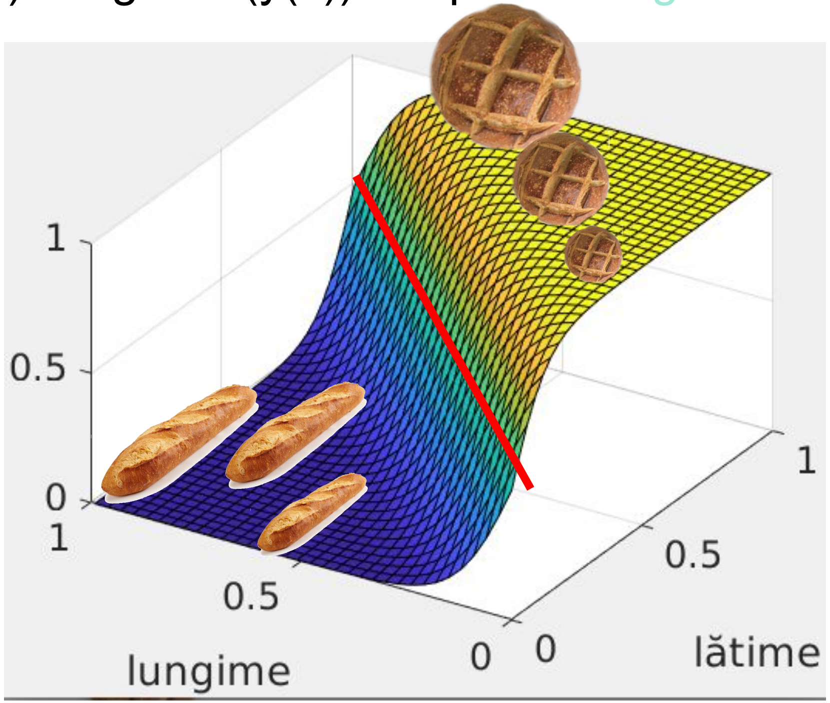
Clasificare - Exemplu

- Vrem să învățăm W astfel încât

a(x) = sigmoid(y(x)) = 1 pentru brioche

a(x) = sigmoid(y(x)) = 0 pentru baguette



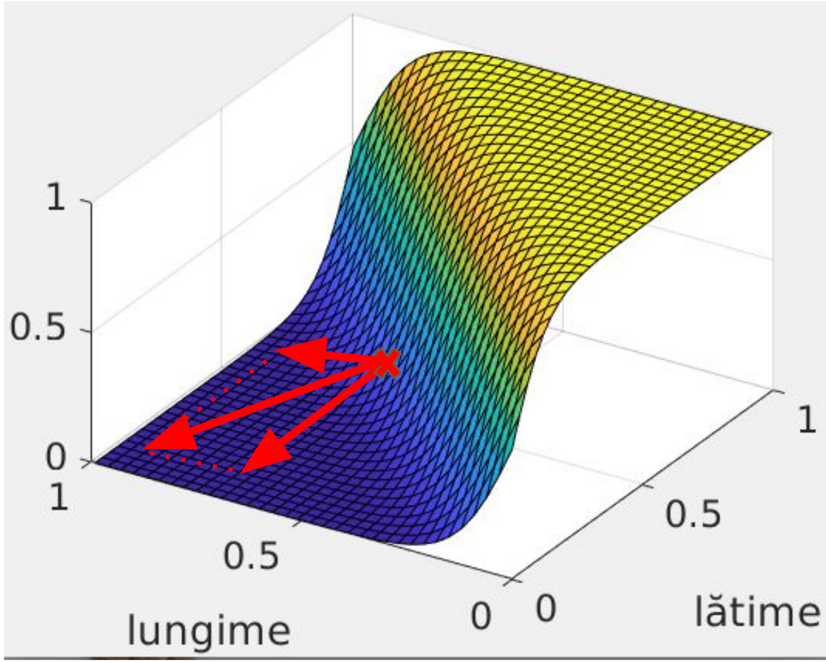


Gradientul în două dimensiuni

- Avem doi parametrii de modificat

- Putem calcula pentru fiecare parametru în parte direcția pentru care Eroarea scade cel mai mult

- Direcția finală e formată din compunerea acestor direcții

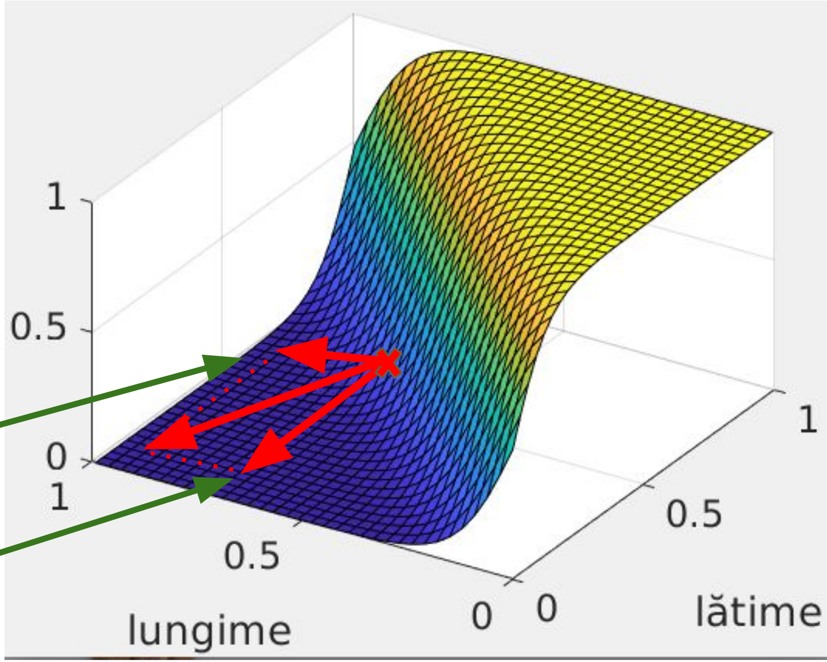


Gradientul în două dimensiuni

$$y = \sigma(w_1 * latime + w_2 * lungime)$$

- Calculăm **derivatele parțiale**
- Formăm **gradientul**

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$



Cum calculăm $\frac{dE}{dw}$ pentru un singur exemplu

Regresie liniară

Clasificare: Regresie logistică

Cum calculăm $\frac{\mathrm{d}E}{\mathrm{d}w}$ pentru un singur exemplu

Regresie liniară

Clasificare: Regresie logistică

$$y = w \cdot x + b$$

$$E = \frac{1}{2}(t - y)^2$$

$$\frac{\mathrm{d}E}{\mathrm{d}y} = \frac{1}{2} \cdot 2(t - y) \cdot (-1) = y - t$$

$$\boxed{\frac{\mathrm{d}E}{\mathrm{d}w} = \frac{\mathrm{d}E}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}w} = (y - t) \cdot x}$$

Cum calculăm $\frac{dE}{dw}$ pentru un singur exemplu

Regresie liniară

$$y = w \cdot x + b$$

$$E = \frac{1}{2}(t - y)^2$$

$$\frac{dE}{dy} = \frac{1}{2} \cdot 2(t - y) \cdot (-1) = y - t$$

$$\boxed{\frac{dE}{dw} = \frac{dE}{dy}\frac{dy}{dw} = (y - t) \cdot x}$$

Clasificare: Regresie logistică

$$y = w \cdot x + b$$

$$a = sigmoid(y) = \frac{1}{1 + e^{-y}}$$

$$E = -t \log a - (1 - t) \log (1 - a)$$

$$\frac{dE}{da} = -\frac{t}{a} + \frac{1 - t}{1 - a}$$

$$\frac{da}{dy} = \ldots = a(1 - a)$$

$$\frac{dE}{dy} = \frac{dE}{da}\frac{da}{dy} = (-\frac{t}{a} + \frac{1 - t}{1 - a})a(1 - a) = a - t$$

$$\boxed{\frac{dE}{dw} = \frac{dE}{dy}\frac{dy}{dw} = (a - t)x}$$

1.2. Rețele complet conectate - Intro

Andrei Roman

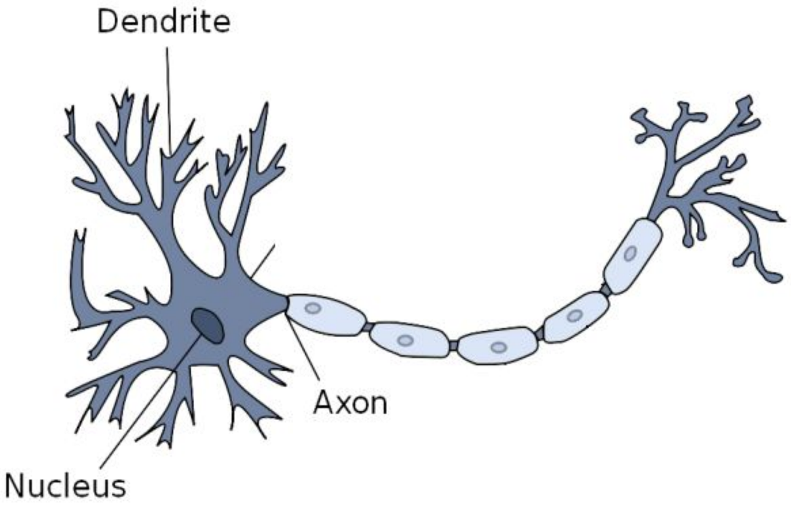
Software Developer

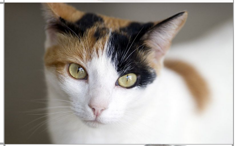
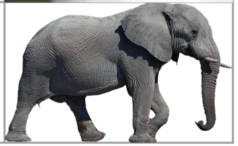
@Teamnet

Neuroni

- Mașini de calcul din natură
- Cum funcționează neuronii?
  - Transmit **semnale electrice** de la un capăt la celălalt
    - (sinapse -> dendrite -> nucleu -> axon -> sinapse)
  - Apoi la următorul neuron, etc.
- Creierul unui animal vs computer:
  - **Fuziness** vs **calcul secvențial**
- Creierul uman:
  - 100 de miliarde (10^11) de neuroni
  - **Conștiința**:
    - un mister so far
  - DAR: știm suficient cât să-i punem la treabă **programatic** în rezolvarea unor probleme.

Neuron

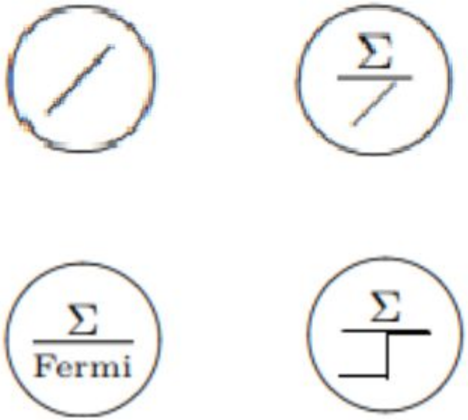


Neuroni	Nume	Imagine
10^4	Furnică	
5*10^7	Liliac	
1.6*10^8	Câine	
3*10^8	Pisică	
2*10^11	Elefant	

Neuroni (2)

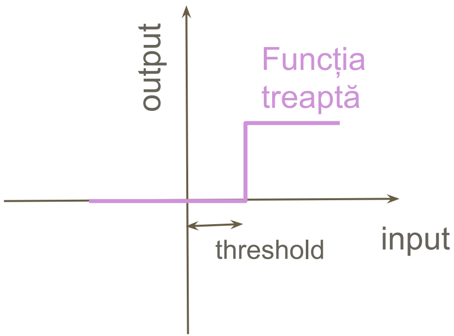
- **Reprezentare**
    - Nu putem folosi funcții liniare (out = c1 * in + c2)
- **Considerente**
    - Nu acționează instantaneu
    - Nu dau mai departe **input-ul** până nu ating un **threshold...**
    - ... apoi produc **outpt-ul**
    - **Analogie**:
        - Precum o cană care nu se varsă până nu este plină.
    - **Altfel spus...**
        - ...fără zgomot.

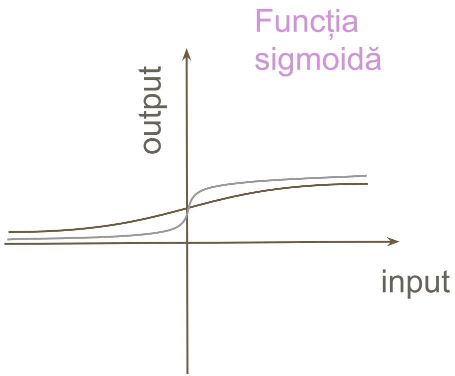
Neuron (2)



Neuroni	Nume	Task
6*10^5	LeNet	MNIST
6*10^7	AlexNet	ImageNet 15.3%
1.3*10^8	VGG-16	ImageNet 7.3%
4*10^6	GoogleNet	ImageNet 6.67%
2.5*10^7	ResNet-50	ImageNet 3.6%

Neuroni (3)





Funcția sigmoidă de activare

$$y = \frac{1}{1+e^{-x}}$$

Taie axa la y = 1/2:
x = 0 => y = 1/2

Funcția treaptă

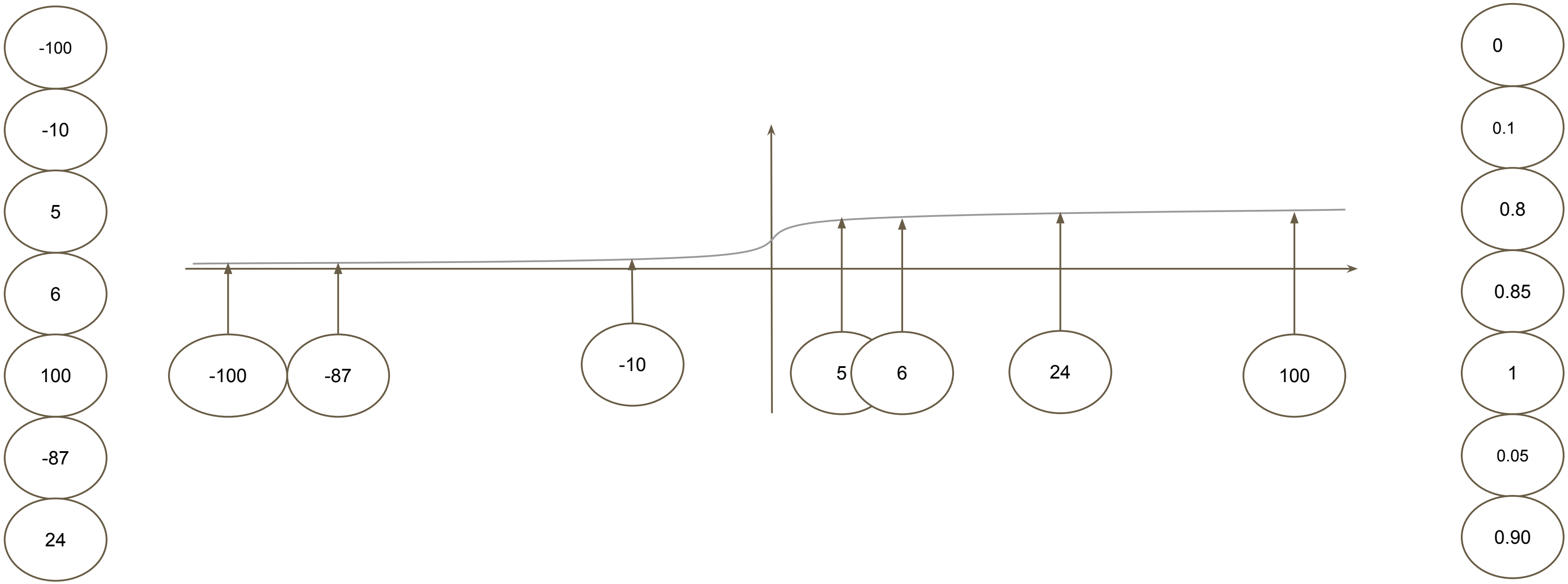
Valori de intrare mici
=> output = 0

Trecem de prag (threshold)
=> neuronii se activează (fire)
=> avem output

Funcția logistică (sigmoidă)

Funcția de activare utilizată mai departe.
De ce?
- Mai smooth decât funcția treaptă
- Simplitate
- Scurtături pentru calcule

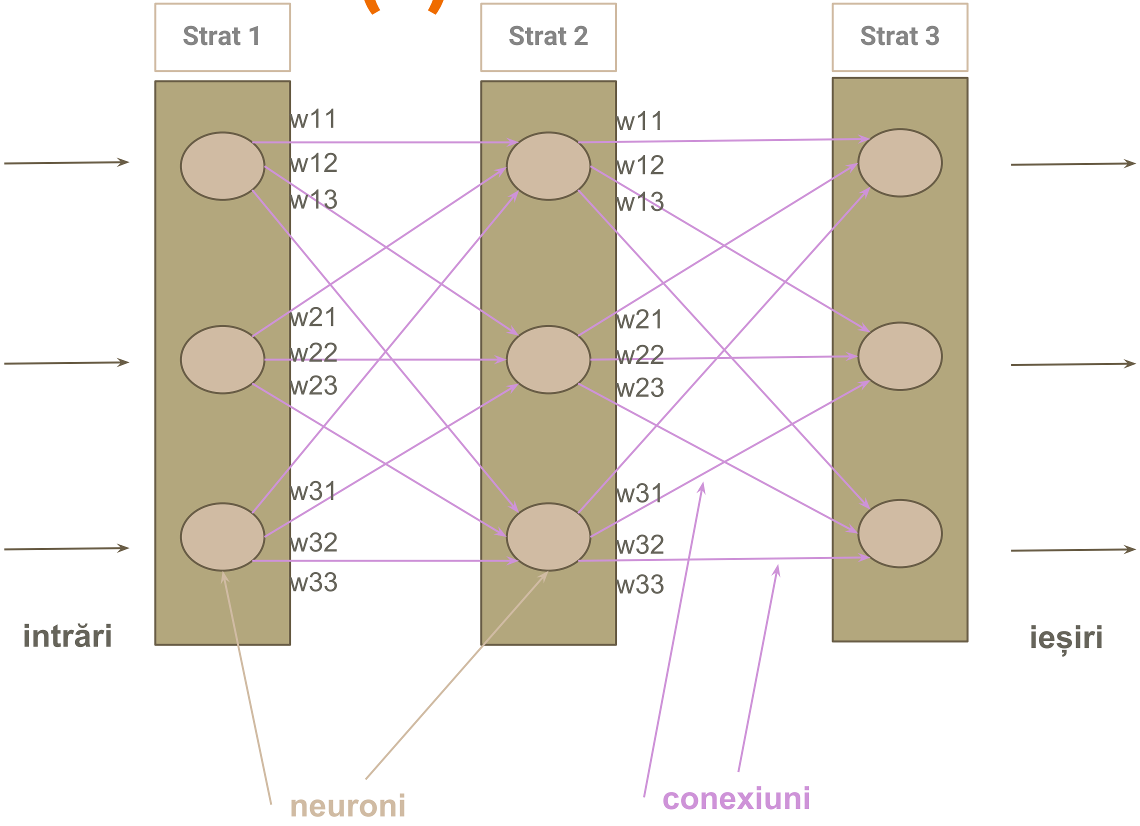
Neuron (3)



Neuroni (4)


Inputs
a
b
c
Sumă inputs
x = a + b + c
Sigmoid threshold function
Output
y

Neuroni (5)

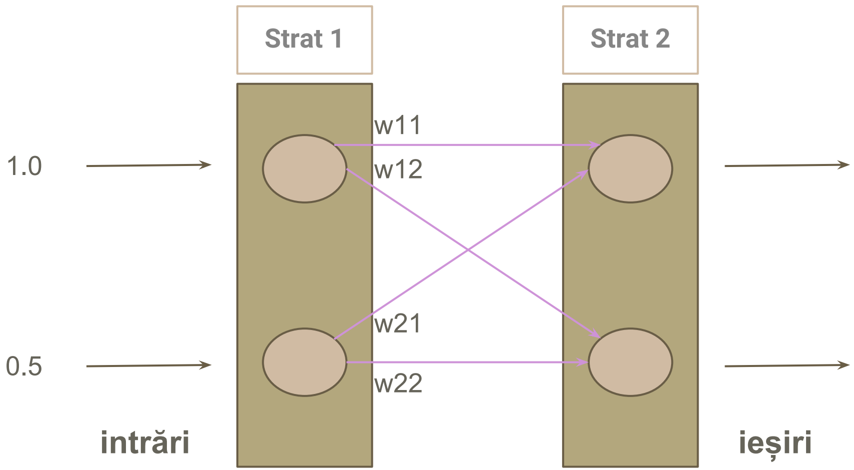


- Exemplu de rețea neuronală **fully connected** cu 3 straturi a câte 3 neuroni fiecare strat.

Învățarea:

- Ajustăm weight-urile.
- **Weight**:
  - Slabe: diminuează semnalul
  - Puternice: amplifică semnalul (au mai mult aport la decizie).

Propagarea semnalelor prin rețea

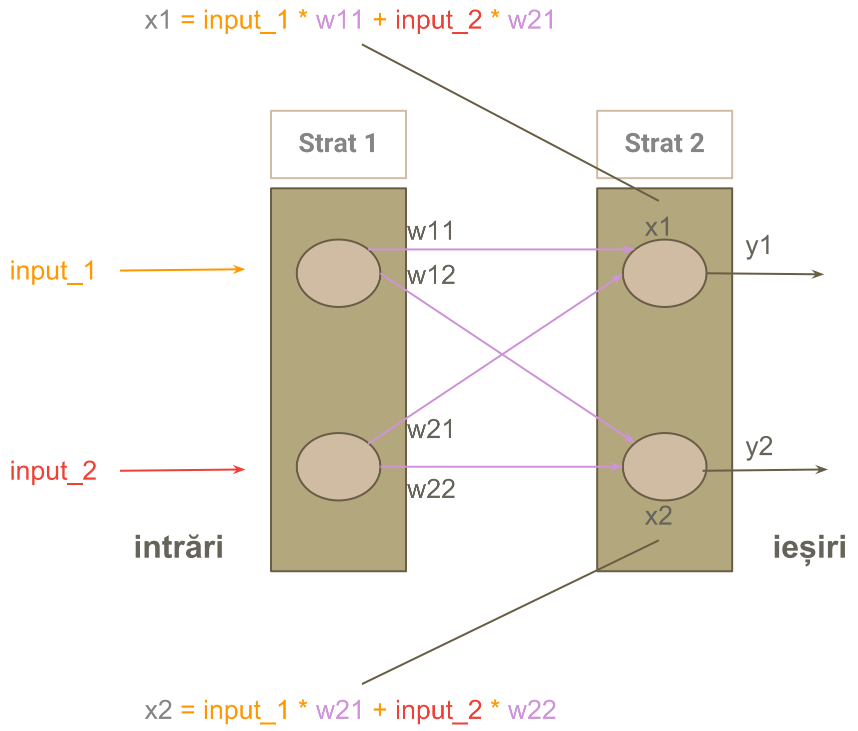


- Weight-uri aleatorii:
  - w11 = 0.9, w12 = 0.2
  - w21 = 0.3, w22 = 0.8

- Funcția de activare:

$$y = \frac{1}{1+e^{-x}}$$

Propagarea semnalelor prin rețea (2)



Layer 1: Reprezentare intrări

Layer 2:

- Input-ul moderat al neuronului 1 de pe layer 2:

$$x_1 = 1.0 * 0.9 + 0.5 * 0.3 = 1.05$$

- Output-ul neuronului 1 de pe layer 2:

$$y_1 = \frac{1}{1+0.3499} = 0.7408$$

- Input-ul moderat al neuronului 2 de pe layer 2:

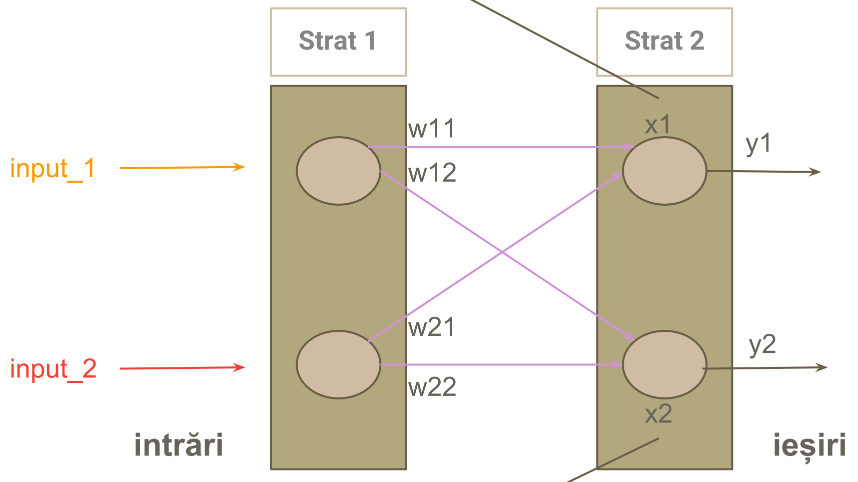
$$x_2 = 1.0 * 0.2 + 0.5 * 0.8 = 0.6$$

- Output-ul neuronului 2 de pe layer 2:

$$y_2 = \frac{1}{1+0.5488} = 0.6457$$

Propagarea semnalelor prin rețea (2)

x1 = input_1 * w11 + input_2 * w21



x2 = input_1 * w21 + input_2 * w22

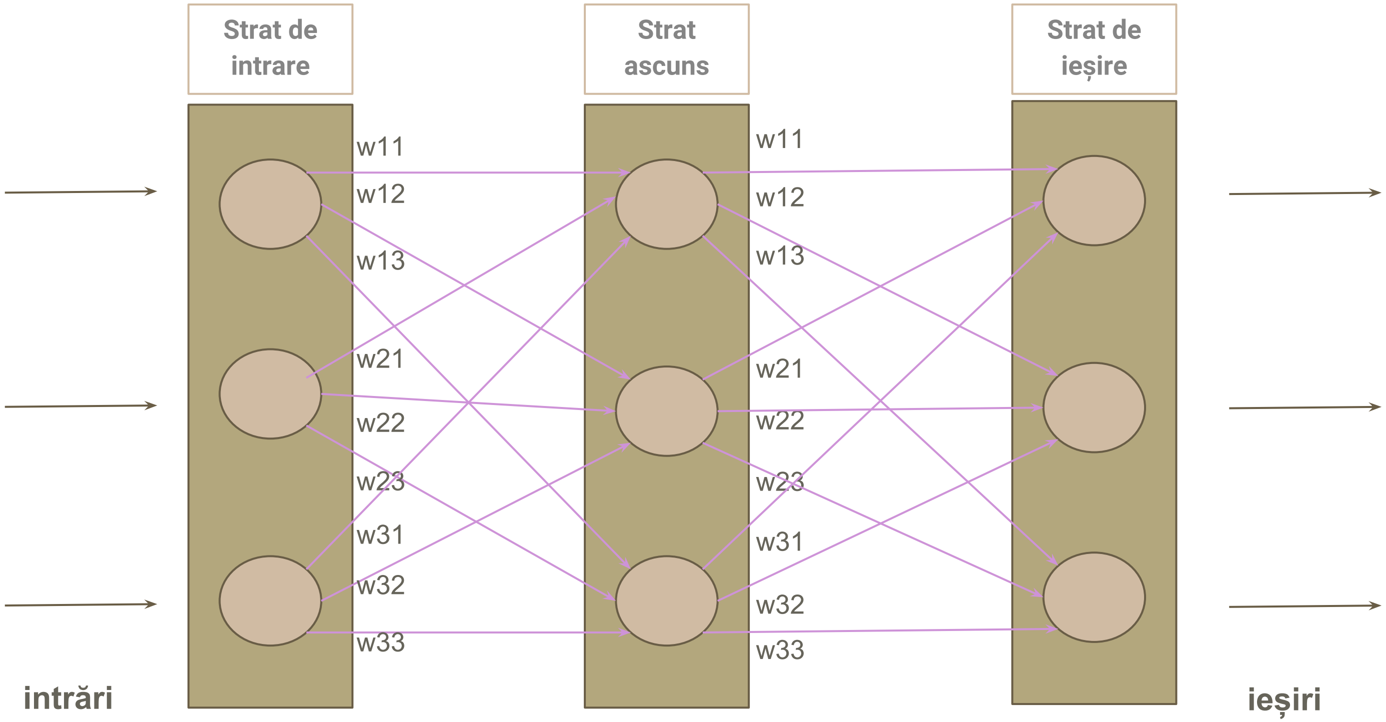
- Folosind calcul matriceal:
  W * I = X

$$\begin{pmatrix} w11 & w21 \\ w12 & w22 \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} =$$

$$= \begin{pmatrix} input\_1 * w11 + input\_2 * w21 \\ input\_1 * w12 + input\_2 * w22 \end{pmatrix}$$

Semnalul moderat combinat care intră în layer 2

Propagarea semnalelor prin rețea (3)



Propagarea semnalelor prin rețea (4)

$$I = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix}$$

$$W_{input\_hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix}$$

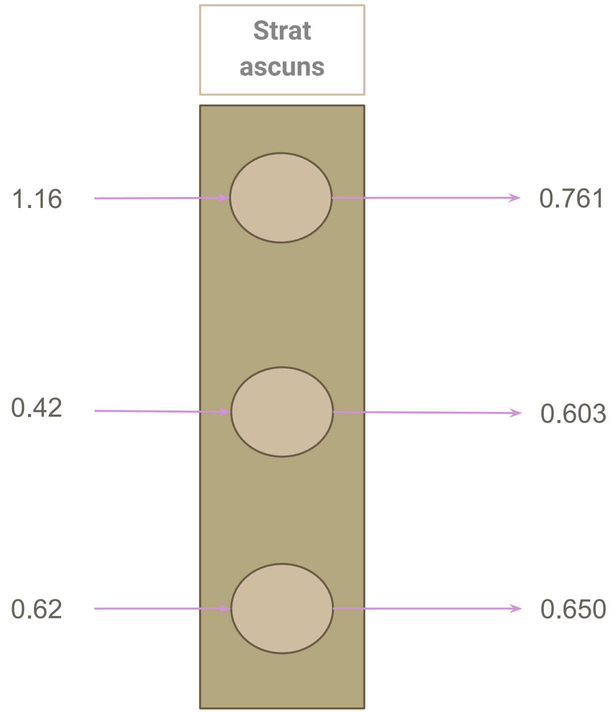
$$W_{hidden\_output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix}$$

Input-uri din setul de training

Alegem aleatoriu weight-urile conexiunilor dintre neuronii stratului de **intrare** și cei de pe stratul **ascuns**.

Alegem aleatoriu weight-urile conexiunilor dintre neuronii stratului **ascuns** și cel de **ieșire**.

Propagarea semnalelor prin rețea (5)



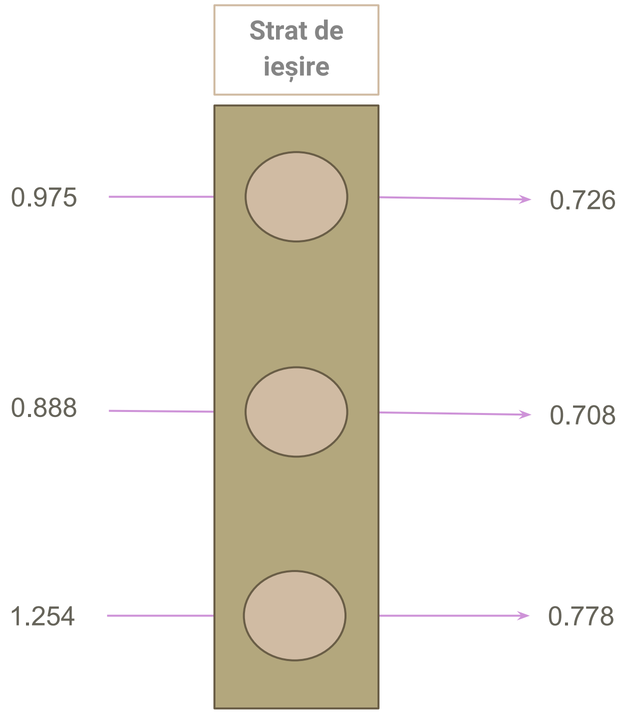
$$X_{hidden} = W_{input\_hidden} * I$$

$$O_{hidden} = sigmoid(X_{hidden})$$

$$X_{hidden} = \begin{pmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{pmatrix} \begin{pmatrix} 0.9 \\ 0.1 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 0.16 \\ 0.42 \\ 0.62 \end{pmatrix}$$

$$O_{hidden} = sigmoid \begin{pmatrix} 0.16 \\ 0.42 \\ 0.62 \end{pmatrix} = \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix}$$

Propagarea semnalelor prin rețea (6)



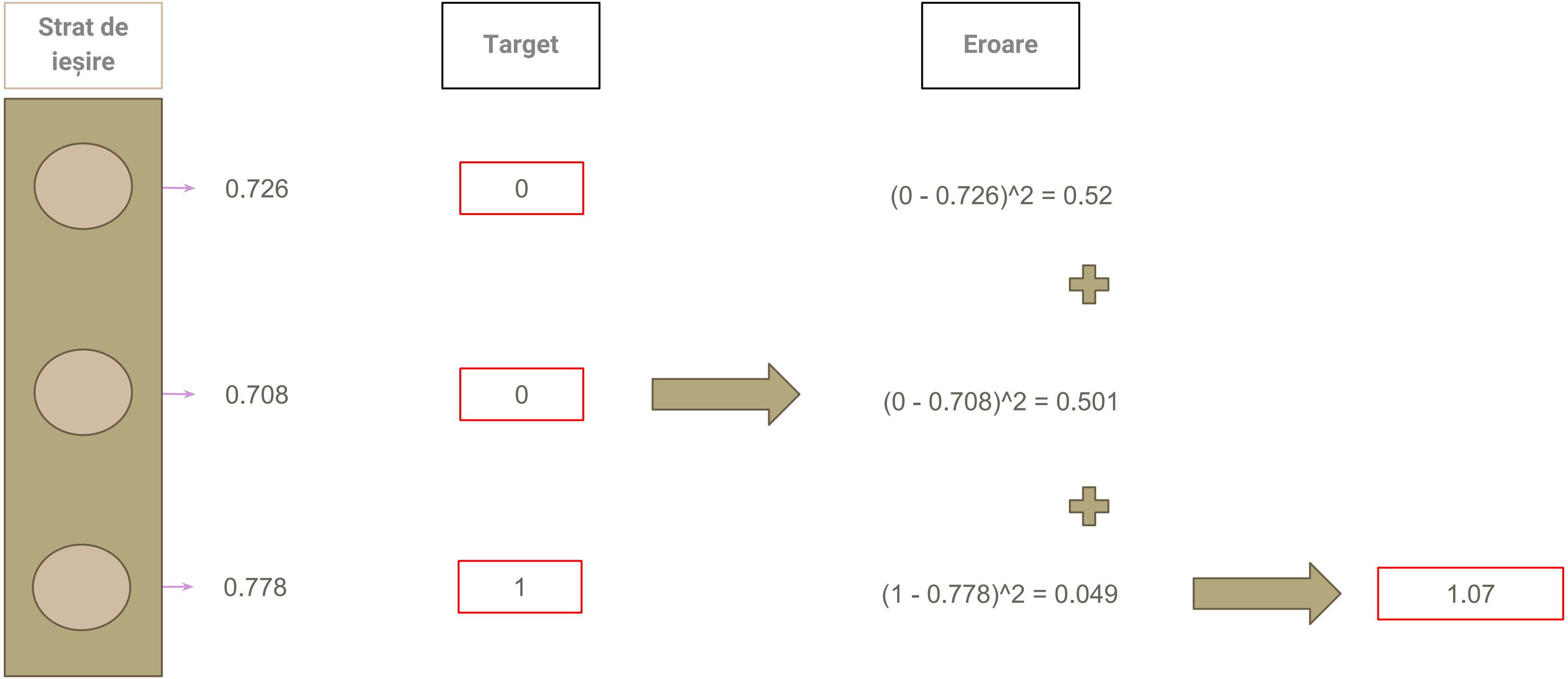
$X_{output} = W_{hidden\_output} * O_{hidden}$

$O_{output} = \text{sigmoid}(X_{output})$

$$X_{output} = \begin{pmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{pmatrix} \begin{pmatrix} 0.761 \\ 0.603 \\ 0.650 \end{pmatrix} = \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix}$$

$$O_{output} = \text{sigmoid} \begin{pmatrix} 0.975 \\ 0.888 \\ 1.254 \end{pmatrix} = \begin{pmatrix} 0.726 \\ 0.708 \\ 0.778 \end{pmatrix}$$

Eroare



Strat de ieșire
Target
Eroare
0.726
0
(0 - 0.726)^2 = 0.52
0.708
0
(0 - 0.708)^2 = 0.501
0.778
1
(1 - 0.778)^2 = 0.049
1.07

Problema

Sa se scrie ecuatile

Input - vector cu 784 de neuroni

Strat ascuns 1 - cu 100 de neuroni

Strat ascuns 2 - cu 10 neuroni

Eroare

Solutie

```
y1 = I * W1
a1 = sigmoid(y1)
y2 = a1 * W2
a2 = sigmoid(y2)
E = SSE(a2, T)
```

1.3. Rețele Neurale (Backpropagation)

Ruxandra Burtică

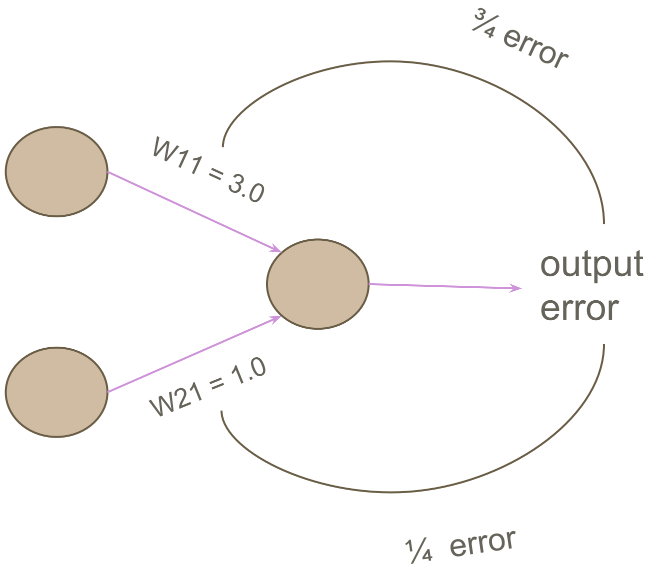
Computer Scientist

@Adobe

Backpropagation

- În general, pentru a calcula gradientul fiecărui strat vom avea doua operații
- **Forward pass:**
  - Calculăm toate rezultatele intermediare, ieșirile și eroarea
- **Backward pass:**
  - Calculăm pe rând gradienții fiecărui strat, de la ultimul la primul

Backpropagation (2)



Propagarea înapoi a erorilor de la un nod:

- Update proporțional
- Propagare cantitate eroare finală în funcție de cât de puternic este link-ul.

Notă:

- Abia pe stratul de ieșire se poate vorbi despre o eroare!

Calcularea erorii

Ce funcție alegem pentru eroare?

1. (target - actual)
2. |target - actual|
3. $(\text{target} - \text{actual})^2$

Calcularea erorii

Ce funcție alegem pentru eroare?

1. **(target - actual)** - NU! Anulare erori cu + vs erori cu -.
2. **|target - actual|** - NU! Nu este continuă în punctul de minim.
3. **(target - actual)^2 -** DA. De ce?
   a. Ecuații simple.
   b. Funcție netedă și continuă
      i. Fără sărituri abrupte
   c. Gradientul se micșorează în apropierea minimului
      i. Fără overshoot dacă micșorăm rata de învățare aici (pasul)

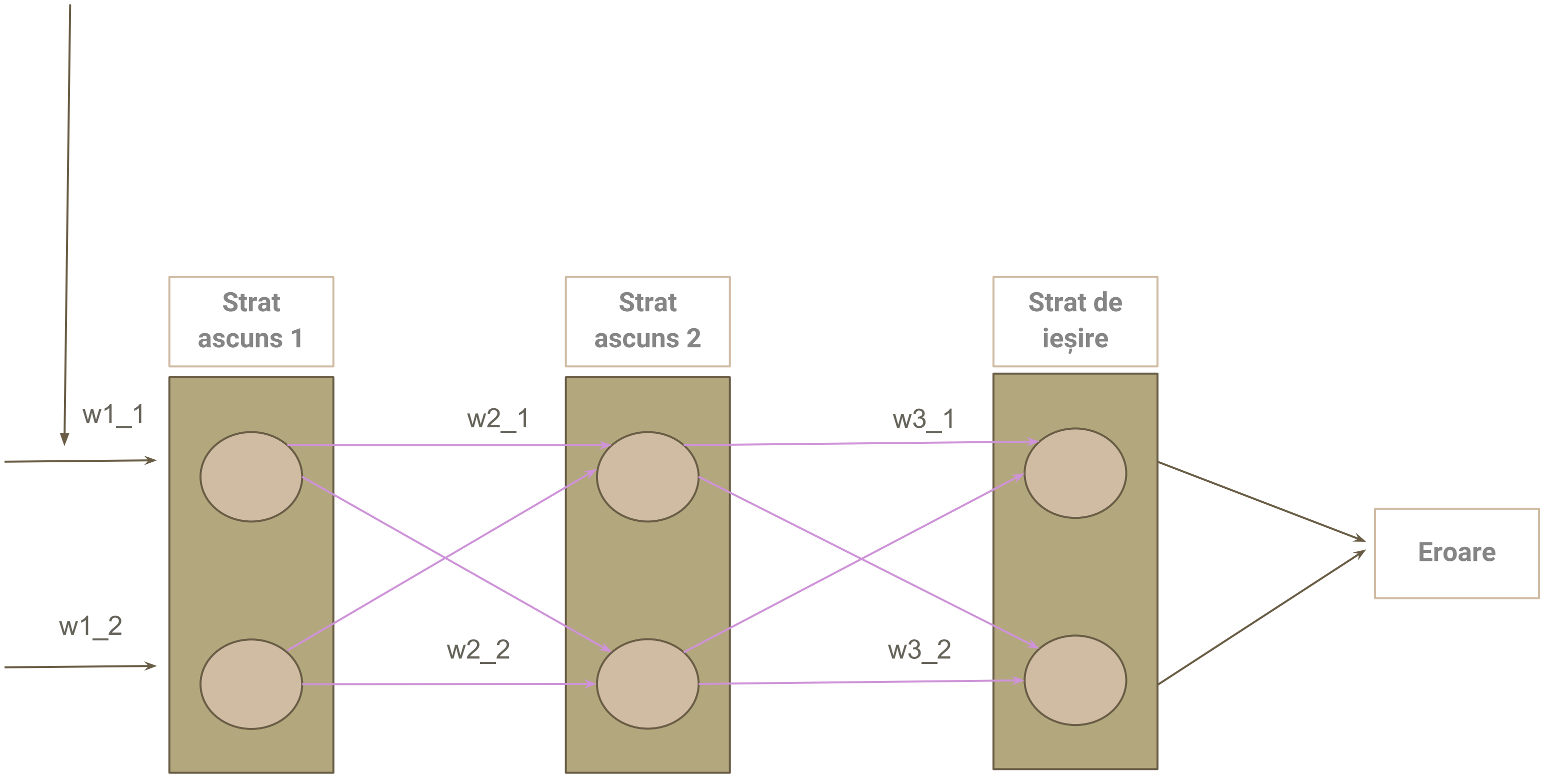
Cum updatăm parametrii rețelei?

- Pentru orice rețea neurală, trebuie să știm ce efect au perturbații mici ale parametrilor în funcția de cost.
- Variem fiecare parametru în parte

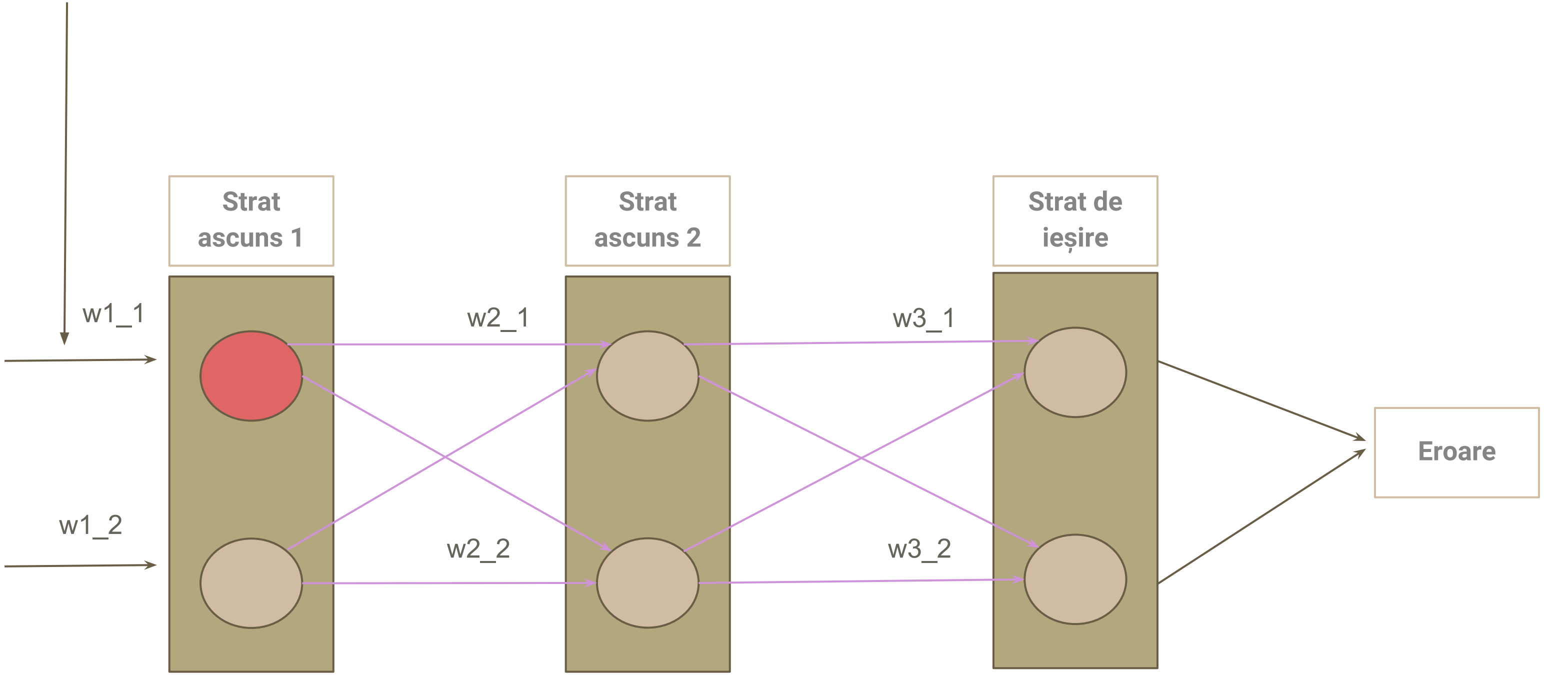
$$\begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ w_N \end{bmatrix} \rightarrow E(W)$$

$$\begin{bmatrix} w_1 + \Delta w_1 \\ w_2 \\ \cdot \\ \cdot \\ w_N \end{bmatrix} \rightarrow E(W + \Delta w1)$$

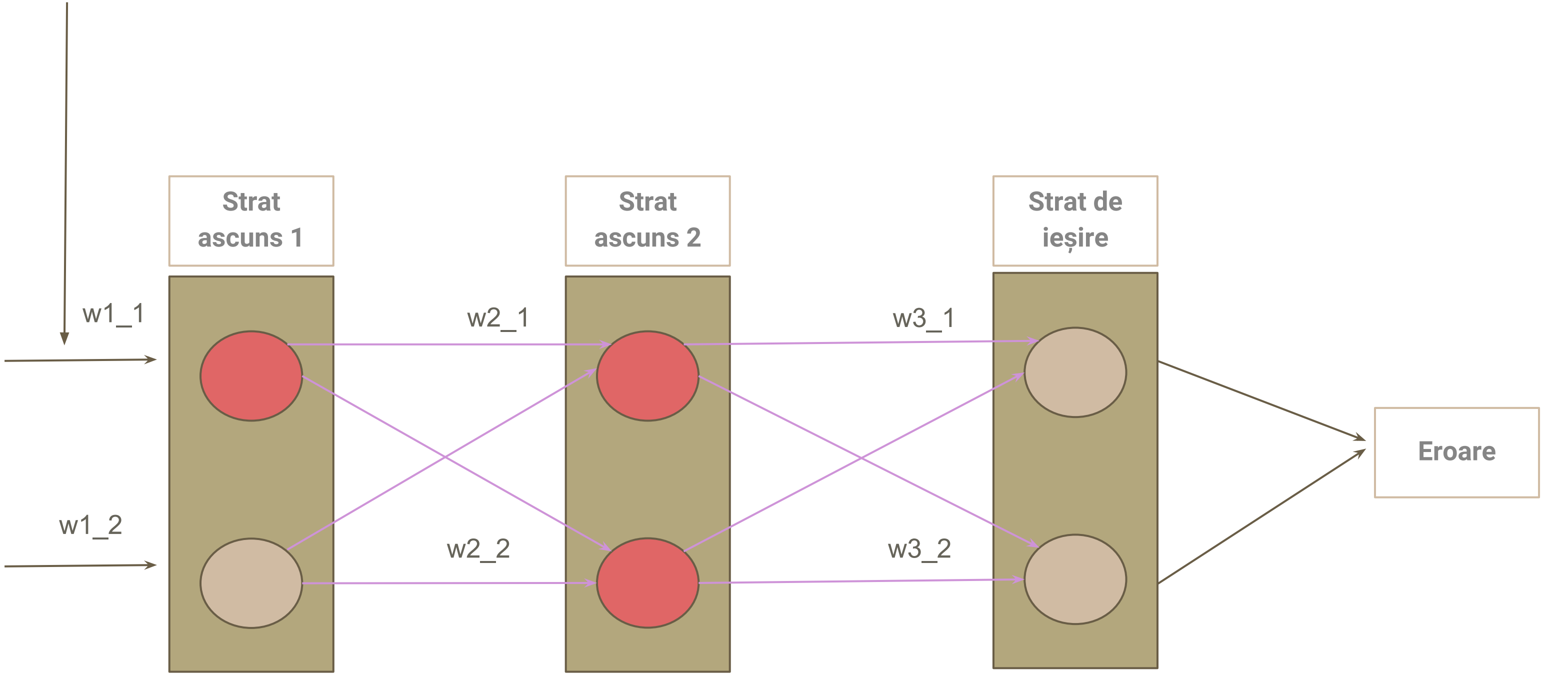
- Perturbăm un singur parametru



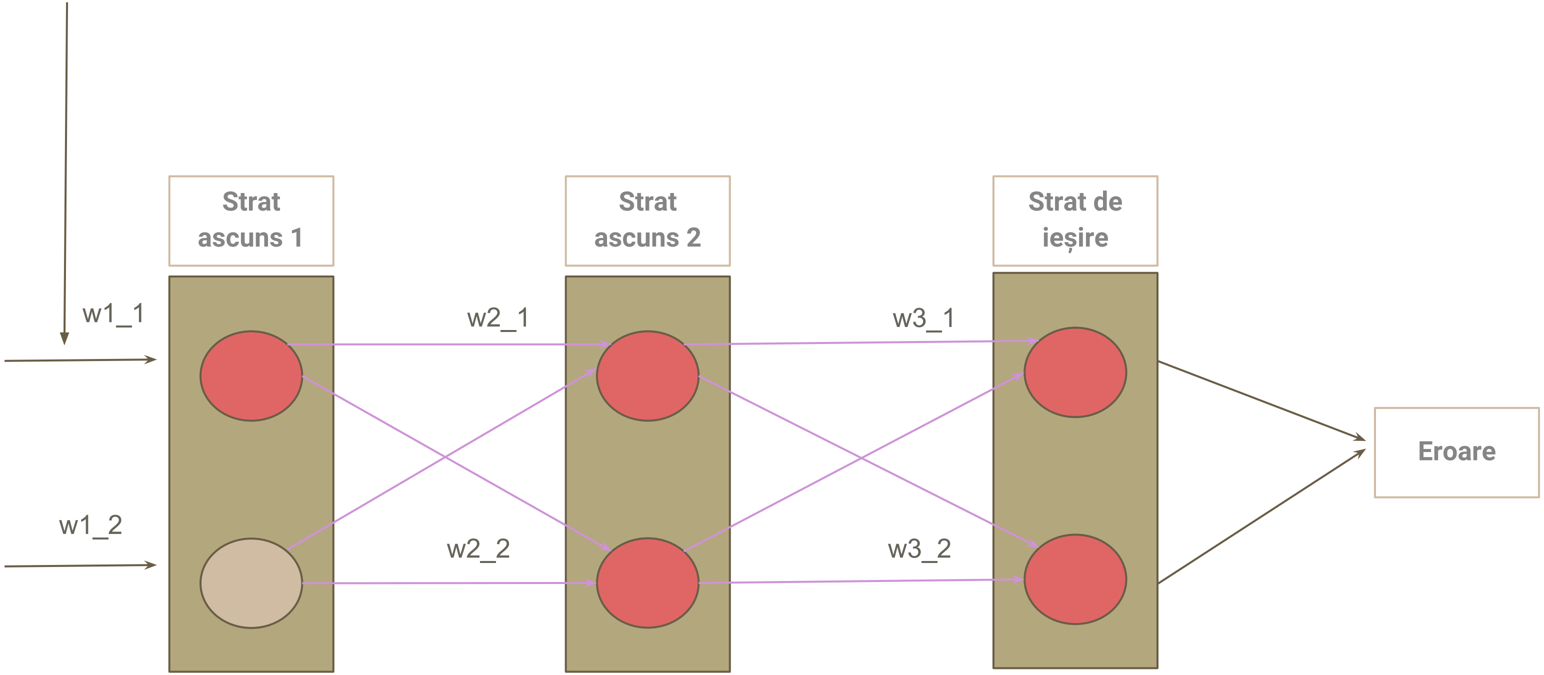
- Perturbăm un singur parametru
- Perturbațiile se propagă prin toții neuronii influențați de primul parametru



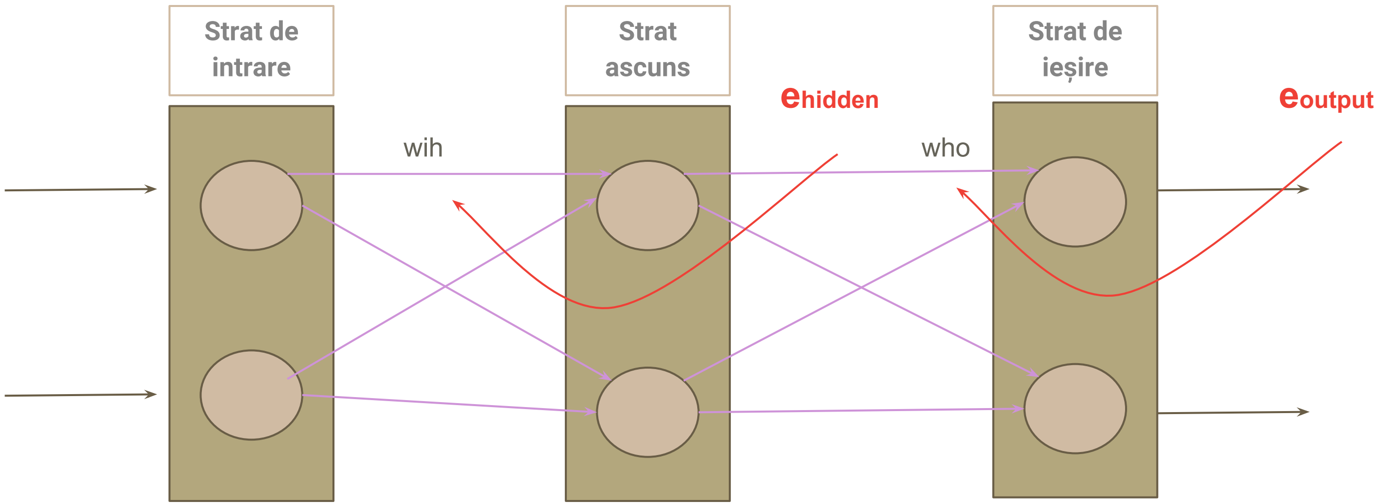
- Perturbăm un singur parametru
- Perturbațiile se propagă prin toții neuronii influentați de primul parametru



- Perturbăm un singur parametru
- Perturbațiile se propagă prin toții neuronii influentați de primul parametru

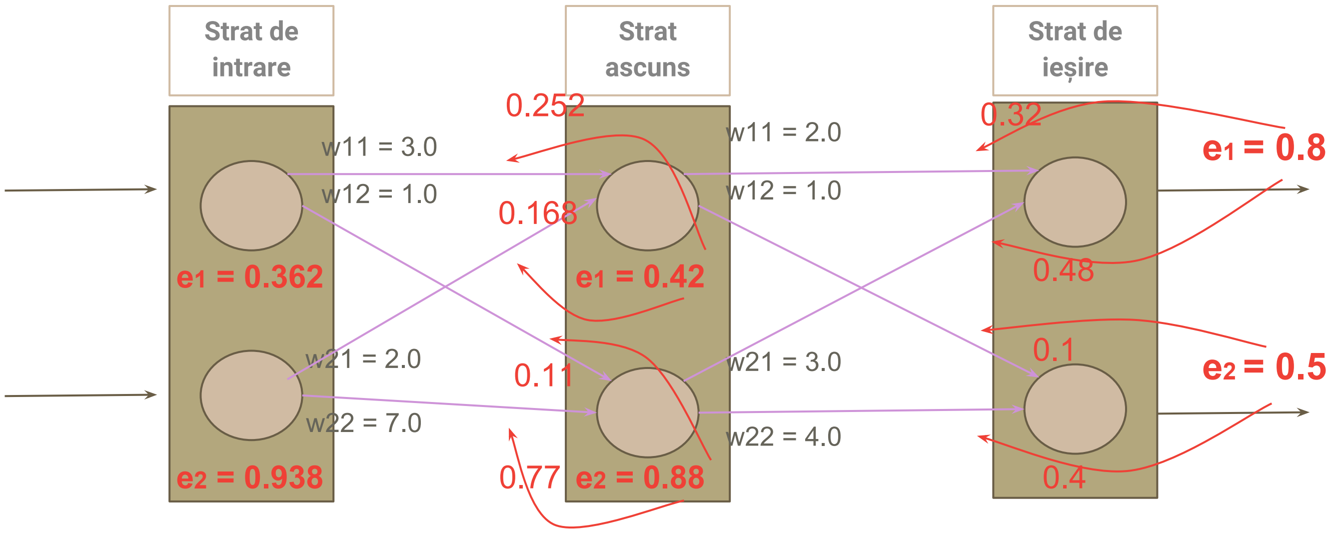


Propagarea erorii



- wih - weight-urile link-urilor dintre neuroni input și hidden
- who - weight-urile link-urilor dintre neuroni hidden și output

Propagarea erorii (2)



$e_{\text{hidden, 1}}$ = sum(split errors on links w_{11} and w_{12})

$= e_{\text{output, 1}} * w_{11} / (w_{11} + w_{21}) + e_{\text{output, 2}} * w_{12} / (w_{12} + w_{22})$

Gradient

- Ar fi ineficient să calculăm numeric fiecare din aceste perturbații
- Precum și la modelele liniare, putem observa cum se modifică eroarea în funcție de perturbații prin derivatele (parțiale)
- Avem câte o derivată parțială pentru fiecare parametru $\frac{\partial E}{\partial w_i} \in \mathbb{R}$
- Vectorul tuturor derivatelor parțiale: gradientul $\nabla E = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_N} \end{bmatrix}$

Gradient

- Calculăm gradienti pentru fiecare strat din rețea

$$\begin{bmatrix} \frac{\partial E}{\partial w_1^1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial E}{\partial w_N^1} \end{bmatrix}$$

- Vom vedea că gradienții din stratul 3 depind de gradienții din stratul 2, cei din stratul 2 depind de cei din stratul 1
- Spunem că **gradienții se propagă** din ultimele straturi spre primele
- Dupa calcularea gradientilor, putem face update la weight-uri:

$$\boxed{w_{jk} = w_{jk} - \alpha \frac{\partial E}{\partial w_{jk}}}$$

Chain rule

- Dacă o funcție **f(x,y)** depinde de de mai multe variabile, putem calcula derivate parțiale pentru fiecare
- Dacă o variabilă a funcției **y** depinde de altă variabila **x**, atunci putem calcula derivatele partiale ale lui **f** fata de **x** folosind regula lanțului (chain rule)

$$f = f(y), y = y(x)$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}$$

Backpropagation

- Dacă f depinde de x1 care depinde de x2 care depinde de x3 care depinde …

$$\frac{\partial f}{\partial x_N} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial x_2}\frac{\partial x_2}{\partial x_3}\cdots\frac{\partial x_{N-1}}{\partial x_N}$$

- Cazul rețelelor neurale, unde avem intrările unor staturi depind de ieșirile straturilor precedente, iar ieșirile straturilor depind de parametrii și intrări
- Vrem sa calculam gradientul fiecărui parametru, trebuie să avem un lanț de la funcția de eroare, prin toate operațiile intermediare

Calcularea gradientului: exemplu simplu

a = 2

b = 3

c = 5

t = 10

q = 6	p = 11	E = 0.5
q = a * b	p = q + c	$E = \frac{1}{2}(t-p)^2$
2	1	1

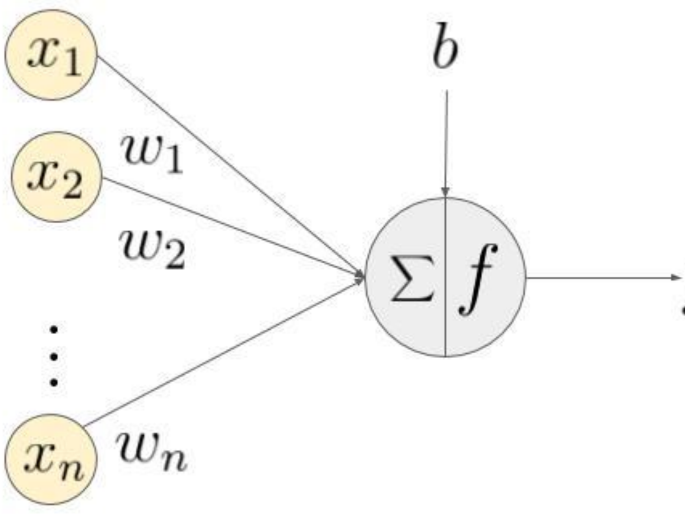
6

$$\frac{\mathrm{d}E}{\mathrm{d}p} = \frac{\mathrm{d}(\frac{1}{2}(t-p)^2)}{\mathrm{d}p} = p - t$$

$$\frac{\mathrm{d}E}{\mathrm{d}q} = \frac{\mathrm{d}E}{\mathrm{d}p}\frac{\mathrm{d}p}{\mathrm{d}q} = (p-t)2$$

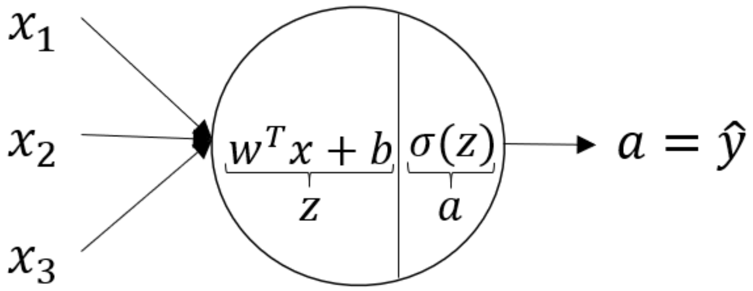
$$\frac{\mathrm{d}E}{\mathrm{d}a} = \frac{\mathrm{d}E}{\mathrm{d}q}\frac{\mathrm{d}q}{\mathrm{d}a} = 2(p-t)3$$

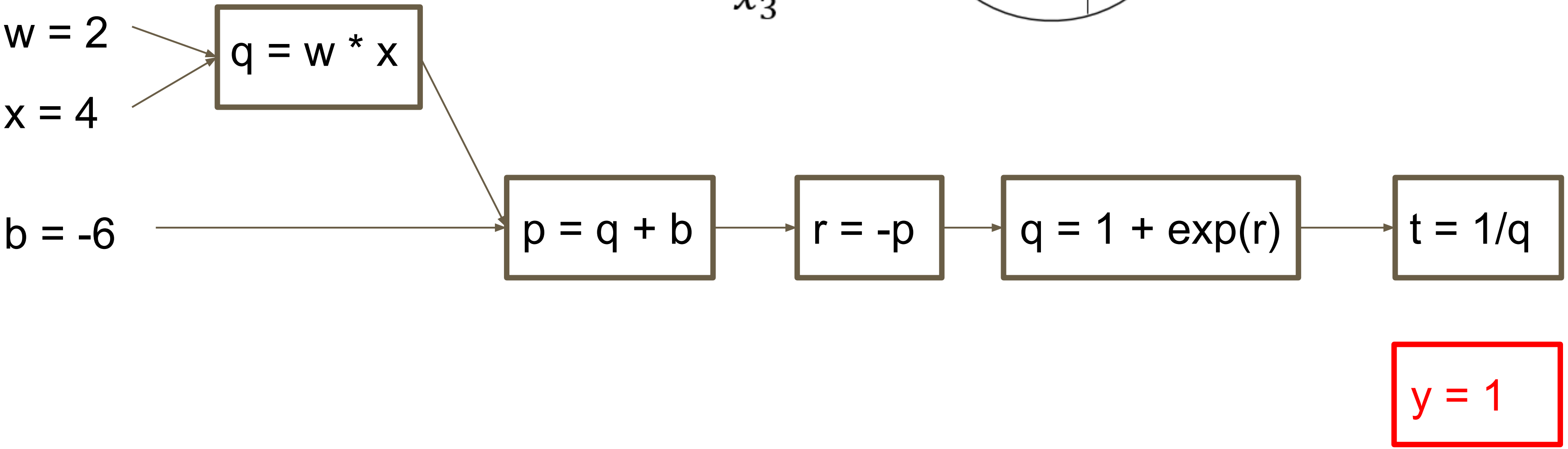
Calcularea gradientului: exemplu simplu (2)



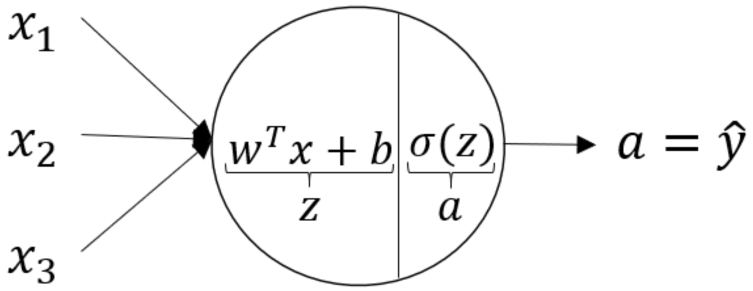
$$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

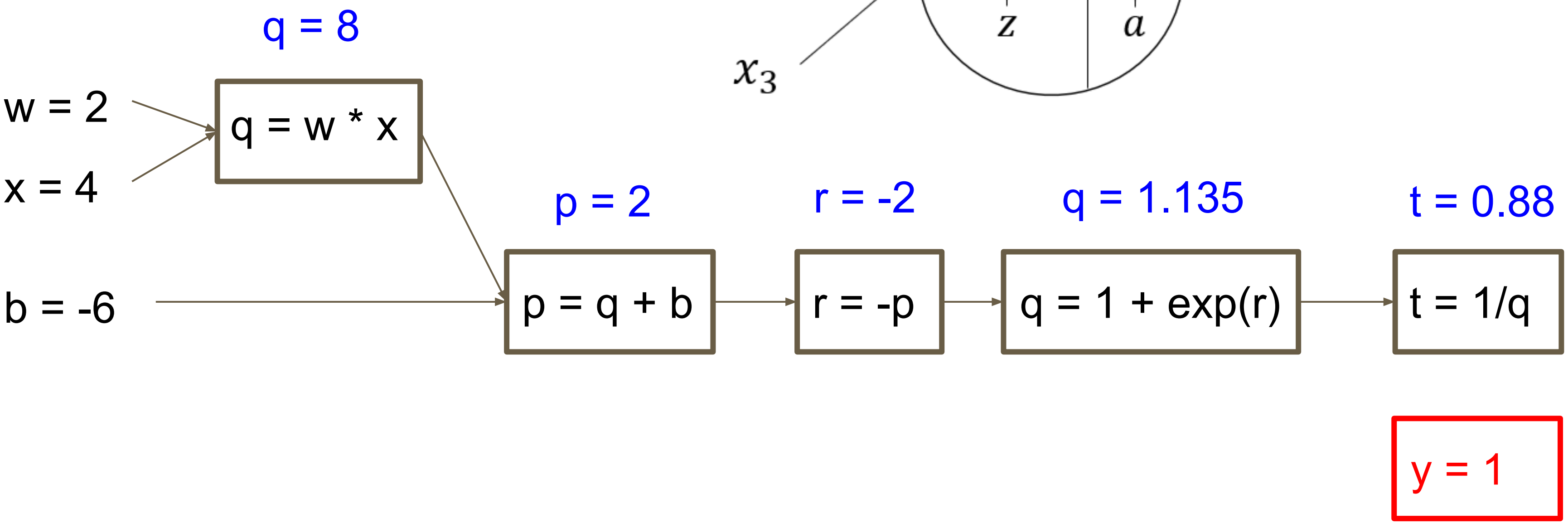
Calcularea gradientului (exercitiu)





Calcularea gradientului (exercitiu)

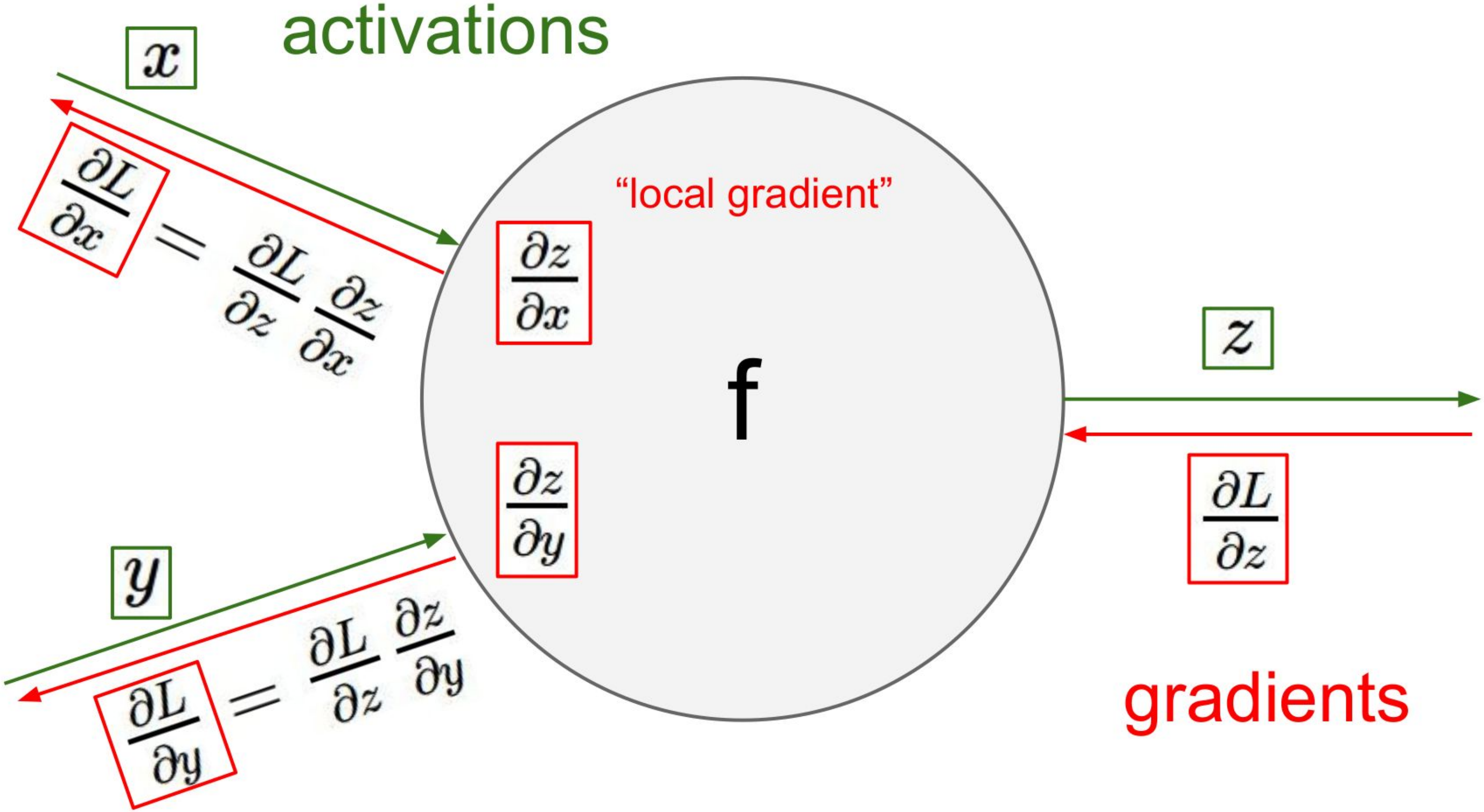




Calcularea gradientului (hints)

$$\left(\frac{1}{x}\right)' = -\frac{1}{x^2}$$

$$\frac{\partial t}{\partial w} = \frac{\partial t}{\partial q} \cdot \frac{\partial q}{\partial r} \cdot \frac{\partial r}{\partial p} \cdot \frac{\partial p}{\partial w}$$



Slide din cursul cs231n Stanford

Actualizarea weight-urilor

(forward propagation)

$$h_1 = xW_1 + b_1$$

$$z_1 = \sigma(h_1)$$

$$z_2 = z_1 W_2 + b_2$$

$$Loss = (z_2 - y)^2$$

(backward propagation)

$$\frac{\partial h_1}{\partial x} = W_1^T$$

$$\frac{\partial z_1}{\partial h_1} = \sigma'(h_1) = z_1 \circ (1 - z_1)$$

$$\frac{\partial z_2}{\partial z_1} = W_2^\top$$

$$\frac{\partial Loss}{\partial z_2} = 2(z_2 - y)$$

Slide din cursul cs231n Stanford

Actualizarea weight-urilor (2)

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} * \sum_n (t_n - o_n)^2 = \frac{\partial (t_k - o_k)^2}{\partial w_{jk}}$$

Din suma anterioară doar (t_k - o_k) are treabă w_{jk}. Restul se anulează la derivare.

E - eroarea tuturor nodurilor din output

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_{jk}} = -2(t_k - o_k) \frac{\partial o_k}{\partial w_{jk}}$$

Chain rule

$$o_k = sigmoid(\sum_j w_{jk} o_j)$$

Output-ul nodului k

Actualizarea weight-urilor (3)

$$\frac{\partial sigmoid(x)}{\partial x} = sigmoid(x)(1 - sigmoid(x))$$

$$\Downarrow$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k)sigmoid(\sum_j w_{jk}o_j)(1 - sigmoid(\sum_j w_{jk}o_j))\frac{\partial}{\partial w_{jk}}(\sum_j w_{jk}o_j)$$

$$\Downarrow$$

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k)sigmoid(\sum_j w_{jk}o_j)(1 - sigmoid(\sum_j w_{jk}o_j))o_j$$

Putem scăpa de doi-ul din față. Pe noi oricum ne interesează doar direcția pantei.

Actualizarea weight-urilor (4)

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) sigmoid(\sum_j w_{jk} o_j)(1 - sigmoid(\sum_j w_{jk} o_j)) o_j$$

e_j = target - actual = eroarea

Semnalul ce intră în ultimul nod **ij**

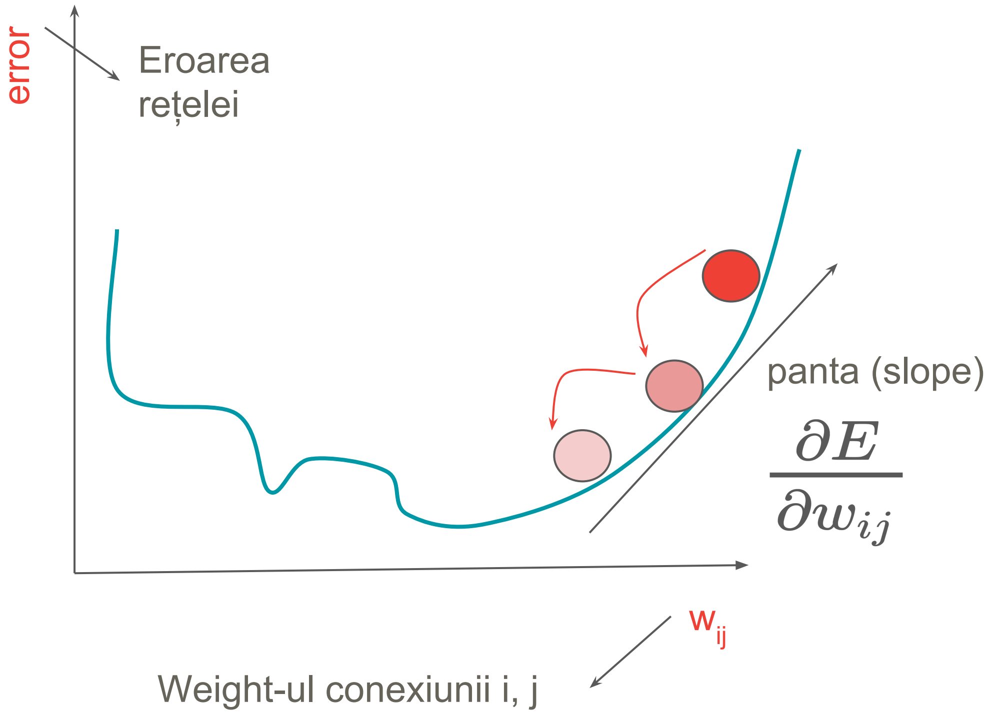
o_j - output-ul layer-ului ascuns anterior j

Panta dintre output și hidden layer:
Vom folosi formula asta la actualizarea weight-urilor.

$$\frac{\partial E}{\partial w_{ij}} = -e_j sigmoid(\sum_i w_{ij} o_i)(1 - sigmoid(\sum_i w_{ij} o_i)) o_i$$

Panta dintre hidden și input layer

Actualizarea weight-urilor (5)



Gradient Descent

- Metodă bună de găsit minimul unei funcții.
- Minimizarea lui E, prin rafinarea w_{ij}.
- Derivata lui E în raport cu w_{ij} ne spune cum se schimbă E odată cu modificarea weight-urilor.

Actualizarea weight-urilor (6)

$$w_{jk} = w_{jk} - \alpha \frac{\partial E}{\partial w_{jk}}$$

Learning rate - moderează dimensiunea salturilor către minim.

Ajustăm vechiul weight cu minus panta erorii.

Matricea schimbărilor de weights

$$\begin{pmatrix} delta\_w_{11} & delta\_w_{21} \\ \ldots & \\ delta\_w_{12} & delta\_w_{22} \\ \ldots & \\ delta\_w_{13} & delta\_w_{jk} \ldots \\ \ldots\ldots\ldots & \ldots\ldots\ldots\ldots \end{pmatrix} = \begin{pmatrix} E_1S_1(1 - S_1) \\ E_2S_2(1 - S_2) \\ E_kS_k(1 - S_k) \\ \ldots\ldots\ldots\ldots \end{pmatrix} \begin{pmatrix} o_1 & o_2 & o_j & \ldots \end{pmatrix}$$

Leagă nodul j de pe un layer cu nodul k de pe următorul.

Valori de pe stratul următor

Valori de pe stratul anterior

$$\Delta w_{jk} = \alpha E_k o_k (1 - o_k) o_j^T$$

Sigmoidele înlocuite cu output-urile nodurilor o_k

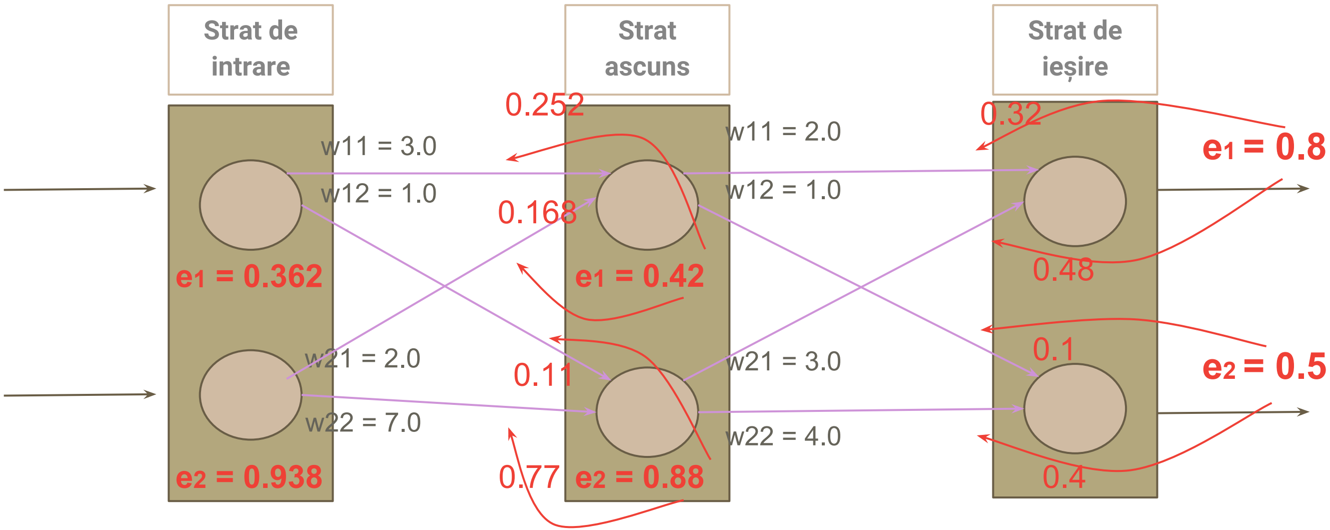
Actualizarea weight-urilor (7)

NOTE

- Rețea neuronală ⇔ funcție liniară de weight-uri
- Îmbunătățirea rețelei ⇔ minimizarea erorii ⇔ rafinare weights
- Îmbunătățim weight-urile iterativ prin descreșterea treptată a erorii folosind Gradient Descent.
- Fiecare pas spre minimul erorii este făcut în direcția în care panta este cea mai abruptă în jos!

Actualizarea weight-urilor (8)

Exemplu



Actualizarea weight-urilor (9)

Exemplu

Vrem să actualizăm w_{11} = 2 (dintre hidden și output layer).

$$\boxed{\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k)\,sigmoid\left(\sum_j w_{jk} o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk} o_j\right)\right) o_j}$$

$(t_k - o_k)$ = e_1 = 0.8

$1 - sigmoid(\sum_j w_{jk} o_j)$ = 1 - 0.88

o_j = o_1 = 0.4

$\sum_j w_{jk} o_j$ = 2.0 * 0.4 + 3.0 * 0.4 = 2

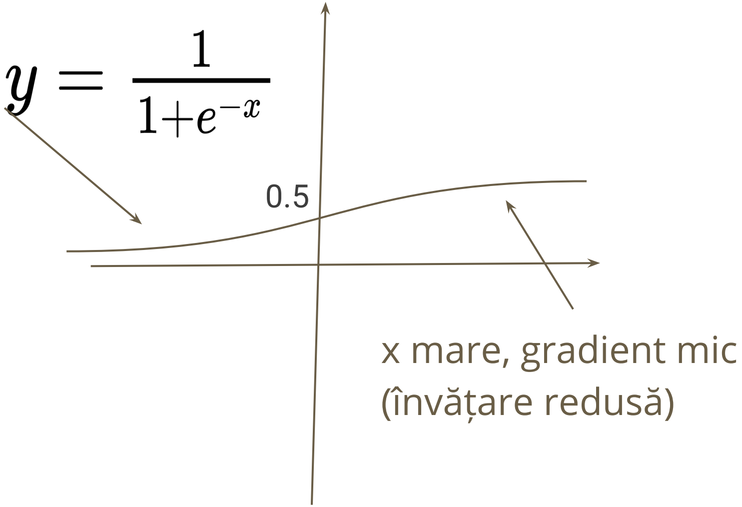
$sigmoid(\sum_j w_{jk} o_j)$ = sigmoid(2) = 0.88

$\frac{\partial E}{\partial w_{jk}}$ = **- 0.0675**

w_{11}= 2.0 + 0.1 * 0.0675 = 2.00675 $\longrightarrow$ Observăm o schimbare destul de mică, dar pe măsură ce iterăm, ea va crește.

Pregătirea datelor

$$y = \frac{1}{1+e^{-x}}$$



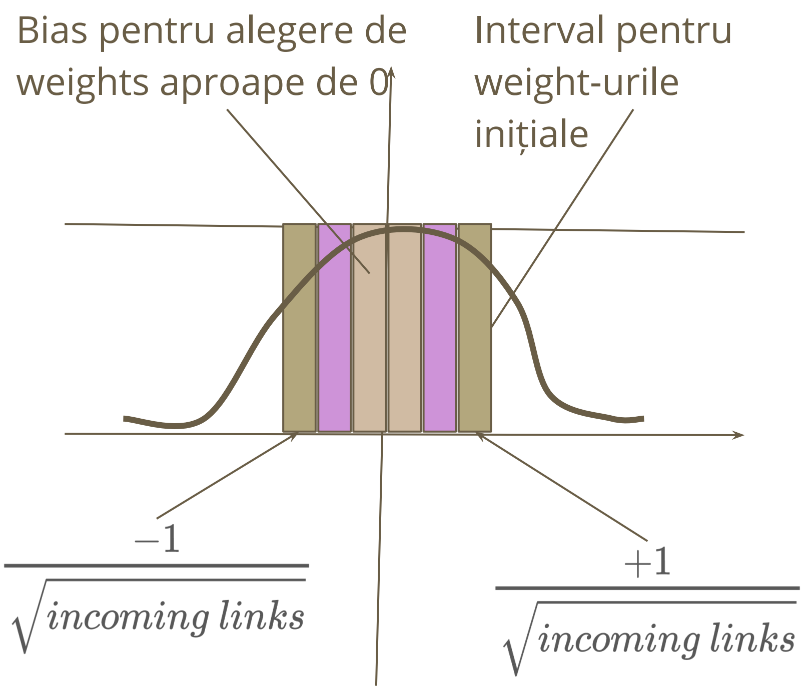
INPUT

- Valori foarte mari
  => aplatizare grafic
  => greu de anticipat încotro ne îndreptăm cu update-urile
  => **saturare**
- Țineți input-urile mici!
- Recomandare: input în **(0.0, 1.0)**

OUTPUT

- Grijă la valorile target!
- Funcția de activare trebuie să poată genera outputs în același interval cu target-urile!

Pregătirea datelor (2)



Weight-uri inițiale aleatorii

- Nu setăm toate weight-urile la aceeași valoare constantă. Cu atât mai puțin zero!
  - O să anuleze input-urile.
- Ideal e să facem sample dintr-o distribuție normală cu media aproape de zero și deviația standard 1/sqrt(incoming links).

Part II

2NClub #1

Hands on:

Recunoașterea Cifrelor Scrise de Mână

CLUB

Cadette

for the love of code