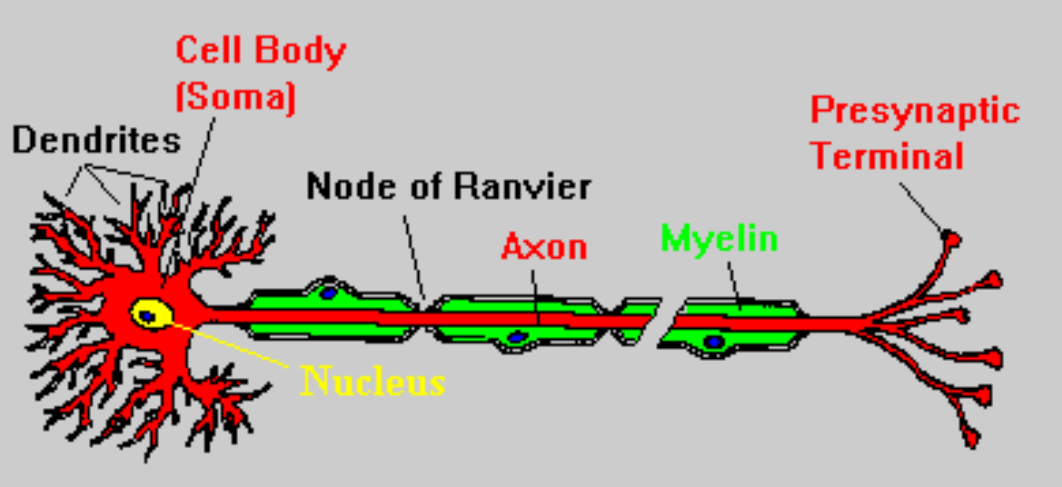
Perceptronul

Perceptronul

- A fost primul sistem de machine learning
- Inspirat din neuronul biologic
- Capabil de doar separații liniare
  - Această limitare a produs un blocaj de 15 ani
- Algoritmul de învățare este un precursor al lui backpropagation gradient descent

Neuron Biologic



Neuronul Artificial - Perceptron

Functia de bază a neuronului este suma impulsurilor de intrare, trecută printr-o funcție de activare.

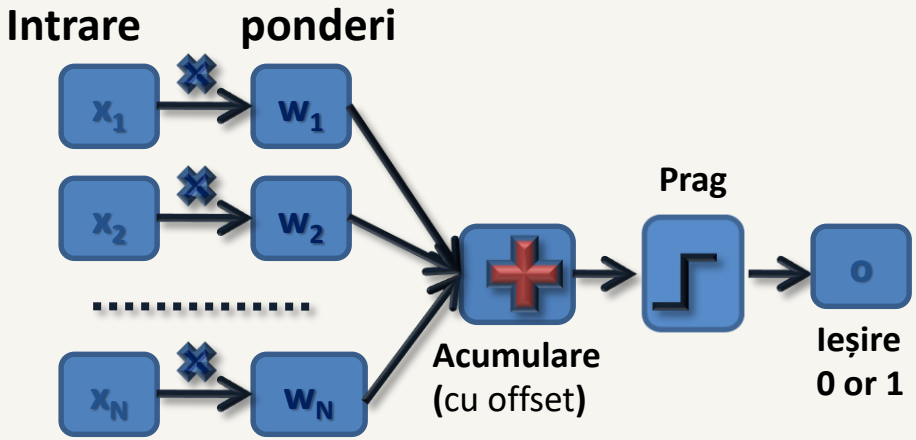
Neuronul – procesor de informații biologice

- ***dentrite*** - receptorii
- ***pericarion (soma)*** – celula- însumează impulsurile de intrare
- ***axonul*** – transmițătorul
- ***sinapsele*** - punctul de transmisie
- Neuronul se activează după ce un anumit prag de intensitate este atins

Perceptronul Artificial

- Model similar cu regresia logistică
- Neuronul Artificial – Perceptron
- Modelul a fost introdus de McCullogh and Pitts în 1943 cu o funcție de prăguire dură (treaptă)
- Istoric a fost antrenat cu algoritm de tip regulă Delta

Modelul Neuronului



$$o = f\left(w_0 + \sum_{i=1}^{N} x_i w_i\right)$$

Antrenare:
-În procesul de antrenare ponderile w_0, $w_1,\ldots,w_N$ sunt determinate

-Perceptronul funcționeaza doar pe probleme **linear separabile**!!!

Funcții de activare

1) **Funcție Prag**

$$f(v) = \begin{cases} 1 & \text{pt } v \geq 0 \\ 0 & \text{altfel} \end{cases}$$

2) **Funcție liniară pe porțiuni**

$$f(v) = \begin{cases} 1 & \text{pt } v \geq ½ \\ v & \text{pt } ½ > v > -½ \\ 0 & \text{altfel} \end{cases}$$

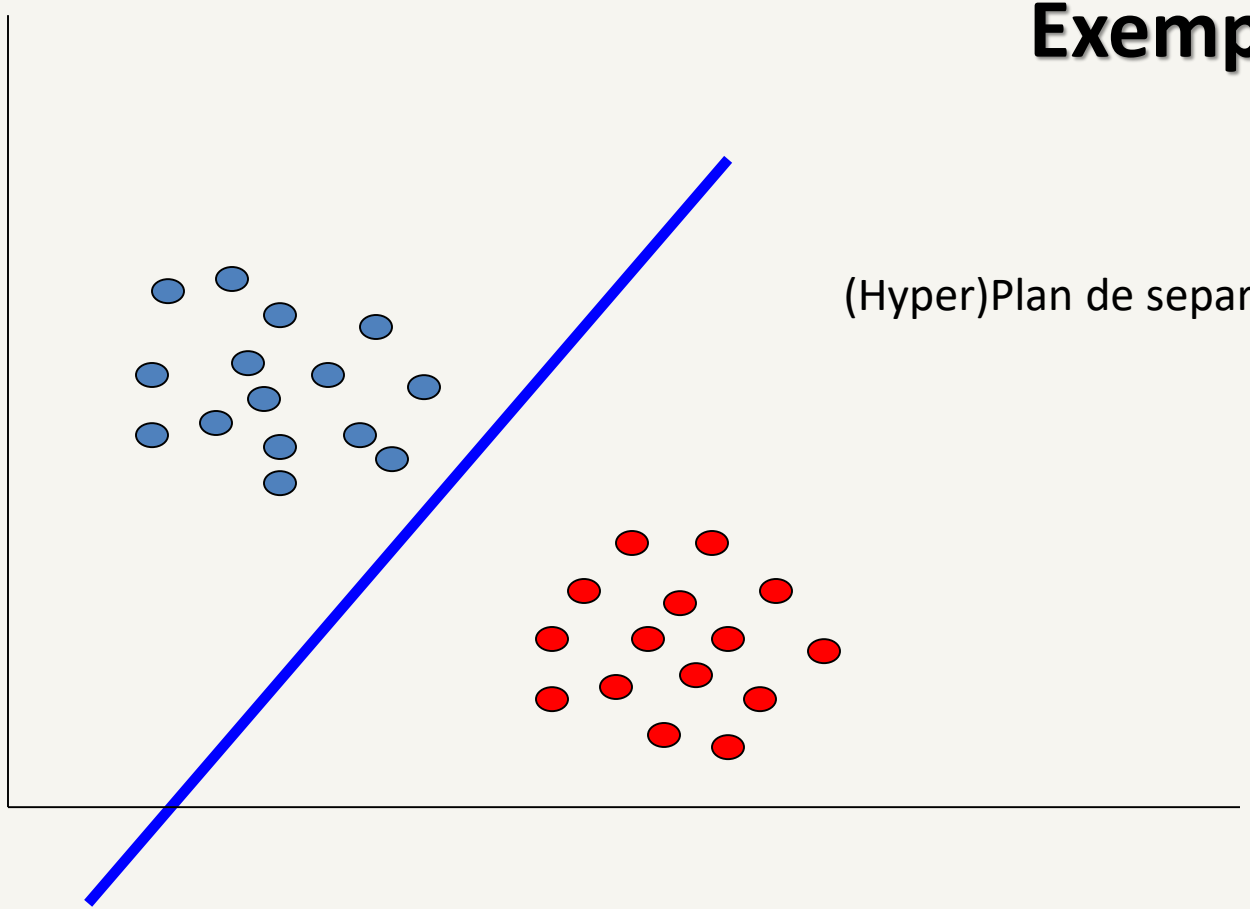
3) **Funcție Sigmoid**

$f(v) = 1/\{1 + exp(- av)\}$

etc..

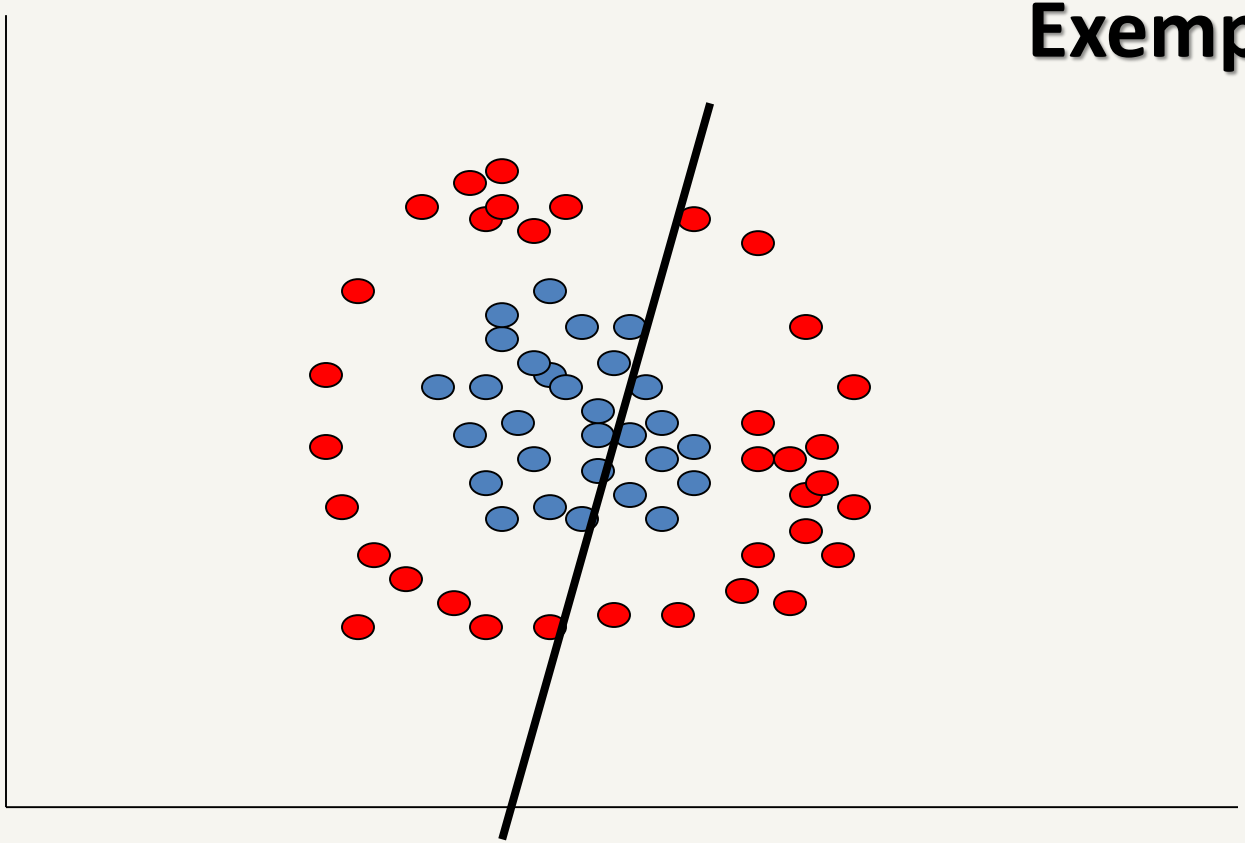
Neuronii pot fi cuplați cu orice funcție **diferențiabilă de transfer f**

Exemplu



(Hyper)Plan de separație

Exemplu



Antrenare unui simplu neuron

Algoritm de antrenare a Perceptronului :

1. Initializăm ponderile cu valori mici aleatoare
2. Se prezintă o dată, $\boldsymbol{x_i}$ din setul de antrenament și ieșirea dorită, $\boldsymbol{y_i}$
3. Se calculează ieșirea : $$o = f\left(w_0 + \sum_{i=1}^{N} x_i w_i \right)$$
4. Se actualizează ponderi:

$$w(t+1) = w(t) + \Delta w(t)$$

Repetă începând cu punctul 2, până când eroarea devine acceptabilă

Antrenarea unui simplu neuron

Algoritmul Widrow-Hoff sau **Regula Delta** pentru modificarea ponderilor

$$w(t+1) = w(t) + \Delta w(t)$$

Pentru ponderile de intrare

$$\Delta w_i(t) = \eta\, \varepsilon(t) x_i(t) = \eta \left[y(t) - o(t) \right] x_i(t)$$

Where:

η = rata de învățare ($0 < \eta <= 1$), De obicei pusă la 0.1 sau 0.2

$\varepsilon(t)$ = *eroarea* = ieșirea dorită (***y()***) – ieșirea rețelei (***o()***)

$$w_0(t+1) = w_0(t) + \eta \varepsilon(t)$$

Pentru bias (offset)

Actualizarea ponderilor

- Clasificare binară– actualizarea ponderilor

$$\Delta w(t) = \eta\, \varepsilon(t) x(t) = \eta \left[y(t) - o(t) \right] x(t)$$

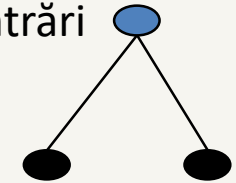
o(t)	y(t)	ε(t)	Δw(t)
0	0	0	0
0	1	+1	ηx(t)
1	0	-1	-ηx(t)
1	1	0	0

Antrenarea perceptronului

- Dacă perceptronul are suficiente exemple de antrenare, antunci vectorul de ponderi **w**(*n*) va converge către valoare corectă **w**.
- Rosenblatt a demonstrat că dacă exemplele de antrenament sunt liniar separabile atunci perceptronul converge garantat

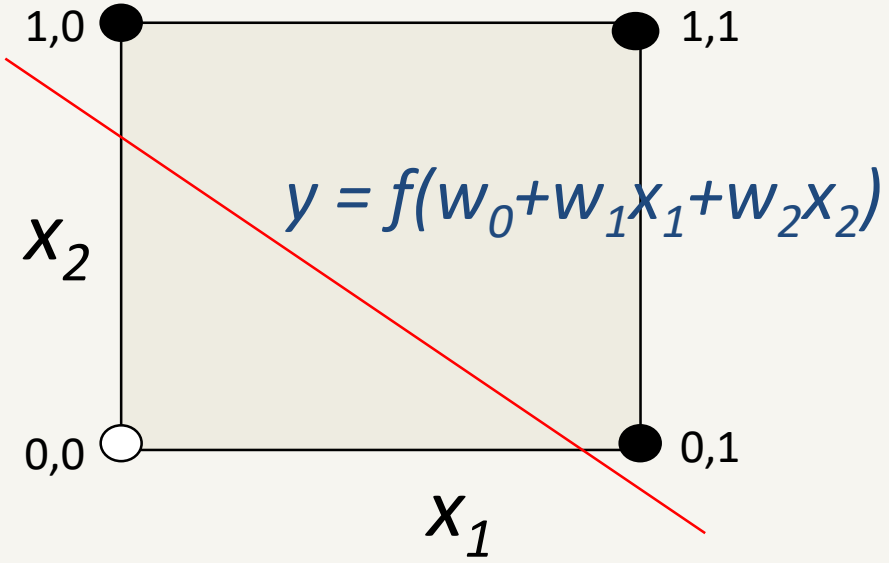
Exemplu

Percepton cu două intrări



Funcția
Logică OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



Ecuația dreptei de separație: $?x_1+?x_2 = ?$ or $-?x_1-?x_2 + ? = 0$

Antrenare

o(t)	y(t)	ε(t)	Δw(t)
0	0	0	0
0	1	+1	ηx(t)
1	0	-1	-ηx(t)
1	1	0	0

$$f(x) = \begin{cases} 1 & x > 0.5 \\ 0 & x \le 0.5 \end{cases}$$

η=0.2 $o = f(w_0+w_1x_1+w_2x_2) = f(o')$

Iterat	w_1	w_2	w_0	x_1	x_2	y= (x_1 or x_2)	o'	o	ε	Δw_1	Δw_2	Δw_0
1	0.02	-0.15	0.09	1	0	1	0.11	0	1	0.2	0	0.2
2	0.22	-0.15	0.29	0	1	1	0.14	0	1	0	0.2	0.2
3	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
4	0.22	0.05	0.49	1	0	1	0.71	1	0	0	0	0
5	0.22	0.05	0.49	0	0	0	0.49	0	0	0	0	0
6	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
1	0.02	-0.15	0.09	1	0	1	0.11	0	1	0.2	0	0.2

Exemplu

Convergență

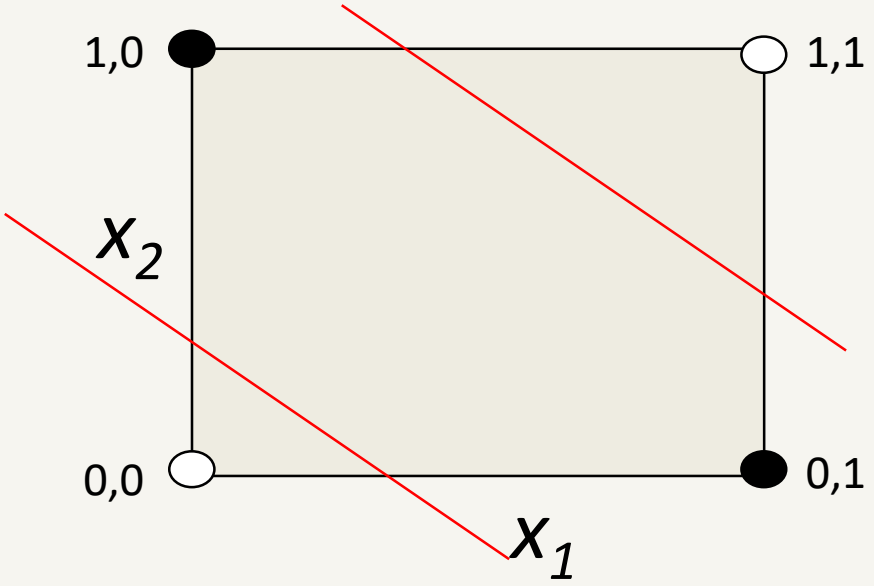
Iterat	w_1	w_2	w_0	x_1	x_2	y	o'	o	ε	Δw_1	Δw_2	Δw_0
1	0.02	-0.15	0.09	1	0	1	0.11	0	1	0.2	0	0.2
2	0.22	-0.15	0.29	0	1	1	0.14	0	1	0	0.2	0.2
3	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
4	0.22	0.05	0.49	1	0	1	0.71	1	0	0	0	0
5	0.22	0.05	0.49	0	0	0	0.49	0	0	0	0	0
6	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
7	0.22	0.05	0.49	0	0	0	0.49	0	0	0	0	0
8	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
9	0.22	0.05	0.49	1	0	1	0.71	1	0	0	0	0
10	0.22	0.05	0.49	0	1	1	0.54	1	0	0	0	0
11	0.22	0.05	0.49	0	0	0	0.49	0	0	0	0	0
12	0.22	0.05	0.49	1	1	1	0.76	1	0	0	0	0
13	0.22	0.05	0.49	1	0	1	0.71	1	0	0	0	0
14	0.22	0.05	0.49	0	1	1	0.54	1	0	0	0	0
15	0.22	0.05	0.49	1	0	1	0.71	1	0	0	0	0
16	0.22	0.05	0.49	0	0	0	0.49	0	0	0	0	0

Dreapta: $0.22x_1+0.25x_2 = -0.11$

Eșec

Funcție logică
XOR

x1	*x2*	*y*
0	0	0
0	1	1
1	0	1
1	1	0



Doi neuroni liniari sunt necesari pentru o clasificare corectă

XOR- Eșec

Iterat	w_1	w_2	w_0	x_1	x_2	y	o'	o	ε	Δw_1	Δw_2	Δw_0
1	0.02	-0.15	0.09	1	0	0	0.11	0	0	0	0	0
2	0.02	-0.15	0.09	0	1	0	-0.06	0	0	0	0	0
3	0.02	-0.15	0.09	1	1	1	-0.04	0	1	0.2	0.2	0.2
4	0.22	0.05	0.29	1	0	0	0.51	1	-1	-0.2	0	-0.2
5	0.02	0.05	0.09	0	0	1	0.09	0	1	0	0	0.2
6	0.02	0.05	0.29	1	1	1	0.36	0	1	0.2	0.2	0.2
7	0.22	0.25	0.49	0	0	1	0.49	0	1	0	0	0.2
8	0.22	0.25	0.69	1	1	1	1.16	1	0	0	0	0
9	0.22	0.25	0.69	1	0	0	0.91	1	-1	-0.2	0	-0.2
10	0.02	0.25	0.49	0	1	0	0.74	1	-1	0	-0.2	-0.2
11	0.02	0.05	0.29	0	0	1	0.29	0	1	0	0	0.2
12	0.02	0.05	0.49	1	1	1	0.56	1	0	0	0	0
13	0.02	0.05	0.49	1	0	0	0.51	1	-1	-0.2	0	-0.2
14	-0.18	0.05	0.29	0	1	0	0.34	0	0	0	0	0
15	-0.18	0.05	0.29	1	0	0	0.11	0	0	0	0	0
16	-0.18	0.05	0.29	0	0	1	0.29	0	1	0	0	0.2

Antrenare cu gradient descent

- Perceptron:
  - Date de intrare vector 2D
  - Funcție de activare sigmoid $f(x) = \frac{e^x}{e^x + 1} = \frac{1}{e^{-x} + 1}$
    - Derivata este $f'(x) = \frac{e^x(1 + e^x) - e^x \cdot e^x}{(e^x + 1)^2} = \frac{e^x}{(e^x + 1)^2} = f(x)\big(1 - f(x)\big)$
  - Ieșirea este: $o = f\left(w_1 x_1 + w_2 x_2 + w_0\right)$

Antrenare cu gradient descent

- Funcția **cost** este eroare pătratică medie

$$E = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - o_i\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f\left(w_1 x_1^i + w_2 x_2^i + w_0\right)\right)^2$$

- Inițializăm cu ponderi aleatoare, mici
- Alegem learning rate λ=0.1
- **Iterăm:**

$$w_1^{(k)} = w_1^{(k-1)} + \lambda \cdot \Delta w_1^{(k-1)} = w_1^{(k-1)} + \lambda \cdot \frac{\partial E}{\partial w_1}^{(k-1)}$$

$$w_2^{(k)} = w_2^{(k-1)} + \lambda \cdot \Delta w_2^{(k-1)} = w_2^{(k-1)} + \lambda \cdot \frac{\partial E}{\partial w_2}^{(k-1)}$$

$$w_0^{(k)} = w_0^{(k-1)} + \lambda \cdot \Delta w_0^{(k-1)} = w_0^{(k-1)} + \lambda \cdot \frac{\partial E}{\partial w_0}^{(k-1)}$$

Iterații- antrenare

- Derivata funcției cost în raport cu o pondere

$$\lambda \cdot \frac{\partial E}{\partial w_1}^{(k-1)} = \lambda \cdot \frac{\partial}{\partial w_1}\left(\frac{1}{N}\sum_{i=1}^{N}\left(y_i - f\left(w_1 x_1^i + w_2 x_2^i + w_0\right)\right)\right)^2 =$$

$$= \frac{\lambda}{N}\cdot\left(\sum_{i=1}^{N}\frac{\partial}{\partial w_1}\left(y_i - f\left(w_1 x_1^i + w_2 x_2^i + w_0\right)\right)^2\right)$$

$$= \frac{\lambda}{N}\cdot\left(\sum_{i=1}^{N}(-2)\underbrace{\left(y_i - f\left(w_1 x_1^i + w_2 x_2^i + w_0\right)\right)}_{(y_i - o_i)}\cdot \underbrace{f'\left(w_1 x_1^i + w_2 x_2^i + w_0\right)}_{f'\left(\vec{x^i}\right) = f\left(\vec{x^i}\right)\left(1 - f\left(\vec{x^i}\right)\right)}\cdot x_1^i\right)$$

$$= \frac{-2\lambda}{N}\cdot\left(\sum_{i=1}^{N}(y_i - o_i)\cdot f\left(\vec{x^i}\right)\cdot\left(1 - f\left(\vec{x^i}\right)\right)\cdot x_1^i\right) =$$

$$= \frac{-2\lambda}{N}\cdot\left(\sum_{i=1}^{N}(y_i - o_i)\cdot o_i\cdot(1 - o_i)\cdot x_1^i\right) =$$

Derivata unei funcții compuse

$$t(x) = f\left(g\left(h(x)\right)\right)$$

$$t'(x) = f'\left(g\left(h(x)\right)\right)\cdot g'\left(h(x)\right)\cdot h'(x)$$

$$o_i = f\left(w_1 x_1^i + w_2 x_2^i + w_0\right) = f\left(\vec{x^i}\right)$$

https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

Iterații- antrenare

- Corecțiile sunt:

$$\lambda \cdot \frac{\partial E}{\partial w_1}^{(k-1)} = \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \cdot x_1^i \right)$$

$$\lambda \cdot \frac{\partial E}{\partial w_2}^{(k-1)} = \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \cdot x_2^i \right)$$

$$\lambda \cdot \frac{\partial E}{\partial w_0}^{(k-1)} = \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \right)$$

- Actualizările sunt:

$$w_1^{(k)} = w_1^{(k-1)} + \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \cdot x_1^i \right)$$

$$w_2^{(k)} = w_2^{(k-1)} + \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \cdot x_2^i \right)$$

$$w_0^{(k)} = w_0^{(k-1)} + \frac{-2\lambda}{N} \cdot \left(\sum_{i=1}^{N} (y_i - o_i) \cdot o_i \cdot (1 - o_i) \right)$$

Perceptron Multi-Strat

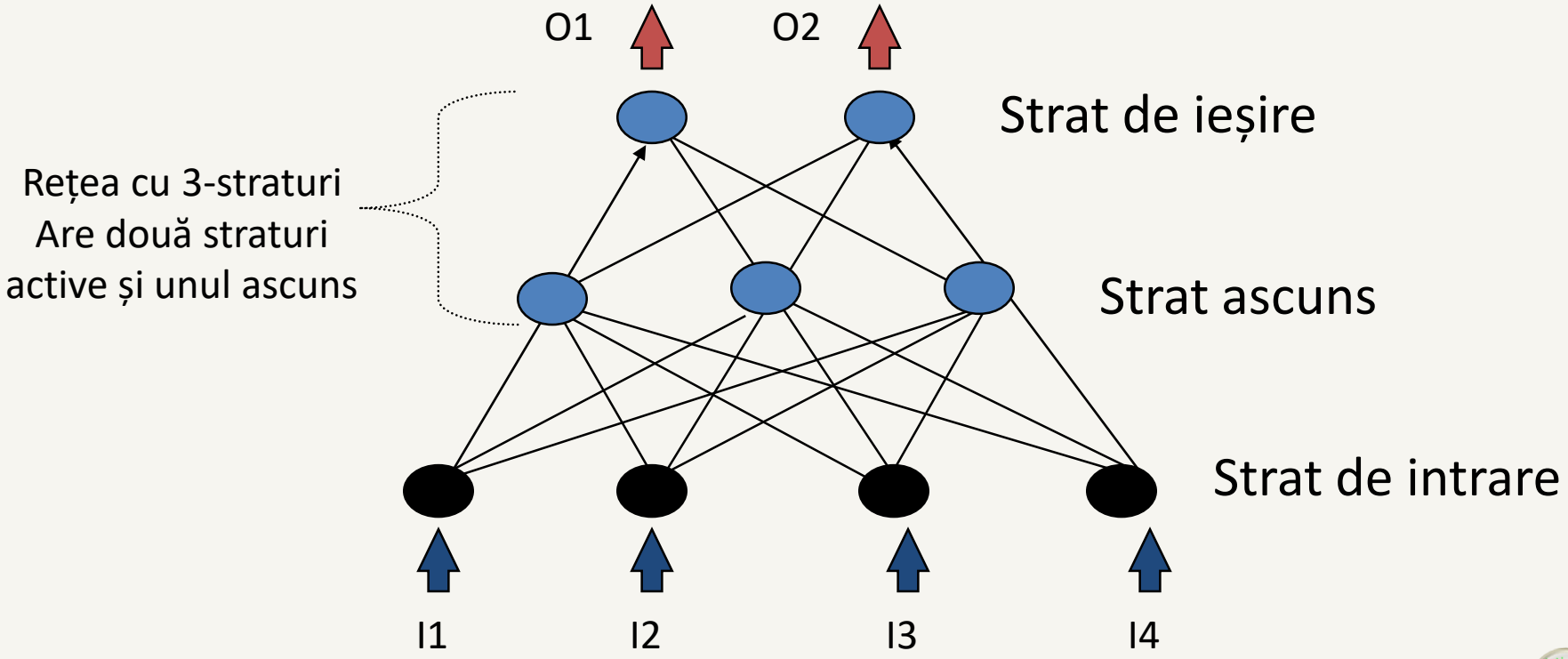
Perceptron Multi-Strat

- Multi-Layer Perceptron = Artificial Neural Network=
  = Feed Forward Network= Fully Connected Network

Dezvoltat în 15 ani (1969-1984) ...

- **Conține straturi ascunse** – combinații neliniare ale stratului anterior
- **Funcții de activare neliniare**
  - ieșirea variză continuu dar neliniar
  - Diferențiabile.... *Sigmoid, ReLu, etc*

Colecție de neuroni artificiali



Formulare matematică

- Fiecare al (i-lea) neuron din stratul de ieșire este calculat ca:

$$o_{ij} = f\left(w_{0ij} + \sum_{i=1}^{N} x_{ij} w_{ij}\right)$$

Antrenare:
- Determinarea valorilor ponderilor

Unde:
- f() este o funcție neliniară
- x_i sunt:
  - Intrările pentru stratul de intrare
  - Ieșirile stratului anterior pentru straturile ascunse sau cele de ieșire
- w_i – ponderi
- w_0 – bias/offset

Antrenare MLPs

Algoritmul Back-propagation (retro-propagare)

- 1986:
- Conceptual simplu –
  - eroarea globală este propagată de la ieșire spre intrare (back) prin nodurile rețelei,
  - ponderile sunt modificate proporționale cu contribuția lor
- Cel mai important algoritm de antrenare

Algoritmul Back-Propagation

- Ca și în cazul Perceptronului – calculul erorii este bazat pe diferența dintre ieșirea rețelei și ieșirea dorită:

$$L = \frac{1}{2}\sum_j (y_j - o_j)^2$$

- În algoritmul BP este importantă rata de variație a erorii și cum se modifică ea prin rețea

  generalized delta rule

$$\Delta w_{ij} = -\lambda \frac{\delta L}{\delta w_{ij}}$$

Algoritmul Back-Propagation

Obiectiv: calculează $\frac{\delta L}{\delta w_{ij}}$ pentru toate ponderile w_{ij}

Formal:

- w_{ij} = Ponderea de la nodul i la nodul j
- x_j= suma ponderată a intrărilor in nodul j $x_j = \sum_{i=0}^{n} w_{ij} o_i$
- o_j= ieșirea nodului $o = f(x_j) = 1/(1+e^{-x_j})$
- L= funcția cost globală a rețelei

Algoritmul Back-Propagation

Obiectiv: Calculează derivatele $\frac{\delta L}{\delta w_{ij}}$ pentru toate ponderile w_{ij}

Proces în patru pași:

1. Calculează cât de repede se modifică eroarea când ieșirea nodului *j* este modificată
2. Calculează cât de repede se modifică eroarea când valoarea nodului *j* se modifică
3. Calculează cât de repede se modifică eroarea când ponderea w_{ij} ce intră în nodul *j* se modifică
4. Calculează cât de repede se modifică eroarea când ieșirea nodului *i* din stratul anterior se modifică

Algoritmul Back-Propagation

Varianta On-Line a algoritmului:

1. Inițializăm ponderile
2. Se consideră un exemplu de antrenare $\mathbf{x_i}$ și ieșirea dorită $\mathbf{y_i}$
3. Calculează ieșirea : $o_{ij} = f\left(w_{0ij} + \sum_{i=1}^{N} x_{ij} w_{ij} \right)$
4. Actualizează ponderile: $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$

unde $\Delta w_{ij} = -\lambda \frac{\delta E}{\delta w_{ij}}$

Se repetă până se atinge un neivel acceptabil al erorii

Back-Propagation

Unde: $$\Delta w_{ij} = \lambda \varepsilon_j o_i = -\lambda \frac{\partial L}{\partial w_{ij}} = -\lambda L W_{ij}$$

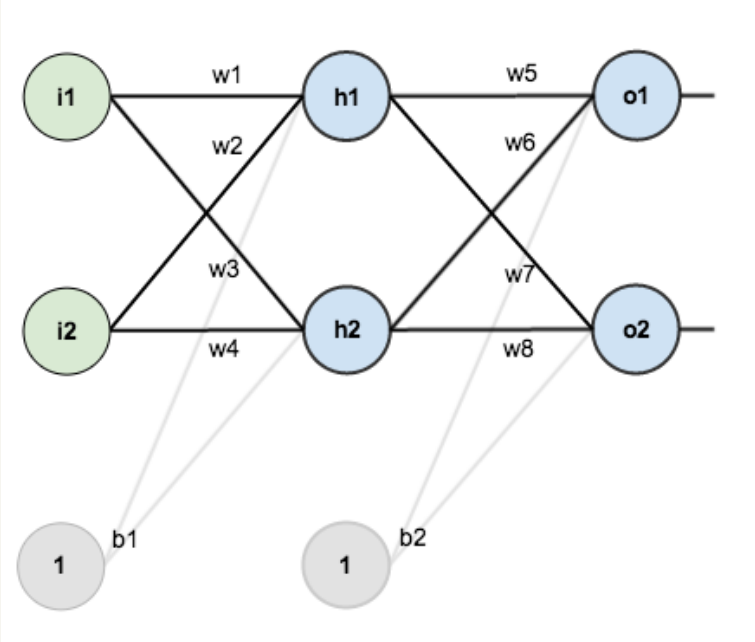
Pentru nodurile de ieșire: $$\varepsilon_j = LI_j = LA_j o_j (1 - o_j) = o_j (1 - o_j)(y_j - o_j)$$

Pentru nodurile ascunse:

$$\varepsilon_i = LI_i = LA_i o_i (1 - o_i) = o_i (1 - o_i) \sum_j LI_j w_{ij}$$

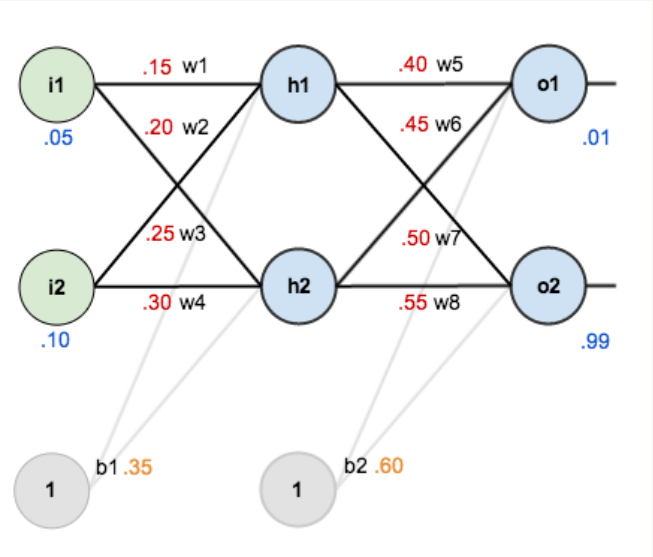
Exemplu

- Preluat de la: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-Exemplu/
- Rețeaua:



Exemplu

- Ponderi inițiale



Pasul de propagare

Valorile din noduri sunt :

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Ieșirea lui h_1 se obține aplicând sigmoida

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Și la fel pt h_2 :

$$out_{h2} = 0.596884378$$

Exemplu

Continuăm propgarea forward

- Procesul se repetă pâna la neuronii de la ieșire.
-

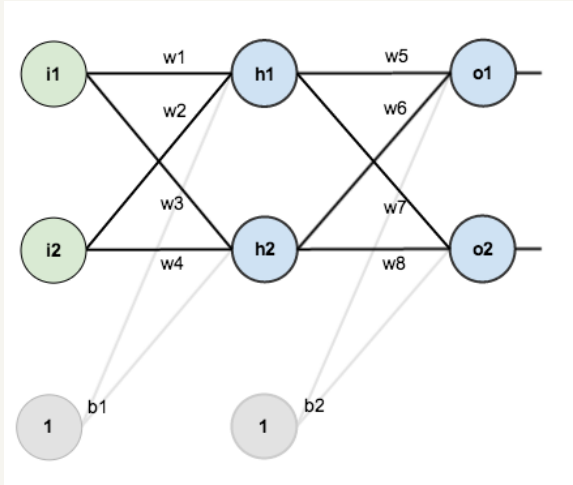
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

Si pentru $\mathbf{o_2}$:

$$out_{o2} = 0.772928465$$



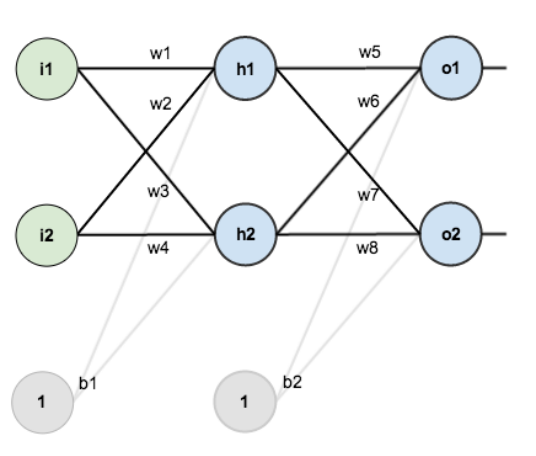
$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Exemplu

Calculăm Eroarea totală

Însumăm erorile pe cei doi neuroni:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$



Ieșirea dorită pt x_1 este 0.01 dar rețeaua produce 0.75136507,
Atunci eroare este:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repetând procesul pentru celălalt neuron unde dorim 0.99, atunci:

$$E_{o2} = 0.023560026$$

Eroarea totală e

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

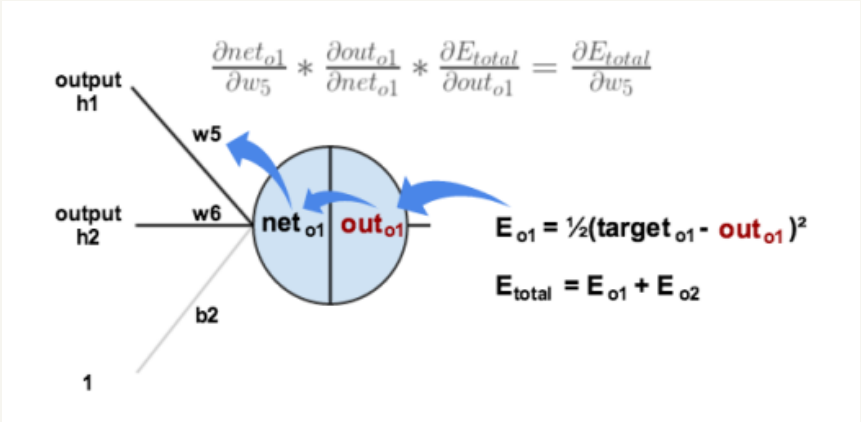
Exemplu

Pasul de retropropagare
Actualizăm ponderile astfel încât să minimizăm eroarea pe fiecare neuron și deci per total

Stratul de ieșire
Consider ăm $\mathbf{w_5}$ Vrem. $\frac{\partial E_{total}}{\partial w_5}$

Aplicând derivata produsului: $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$

Vizual:



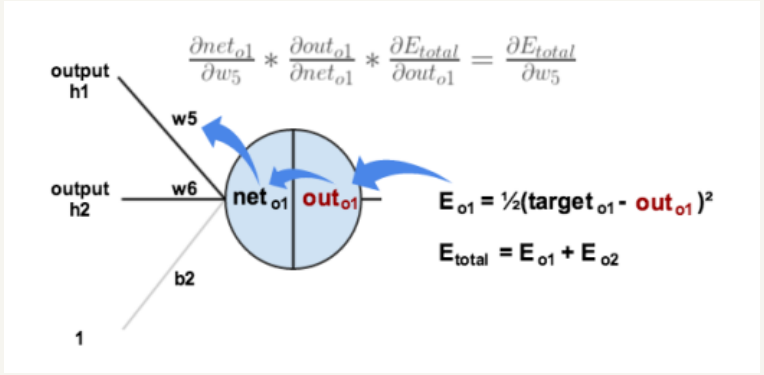
Exemplu

We need to figure out each piece in this equation.
First, how much does the total error change with respect to the output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$



When we take the partial derivative of the total error with respect to ***out$_{o1}$*** , the quantity ***[1/2(target$_{o2}$ – out$_{o2}$)^2]*** becomes zero because ***out$_{o1}$*** does not affect it which means we're taking the derivative of a constant which is zero.

Exemplu

Next, how much does the ieșire of o_1 change with respect to its total net input?
The partial derivative of the logistic Funcție (a.k.a. sigmoid) is the ieșire multiplied by 1 minus the output:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

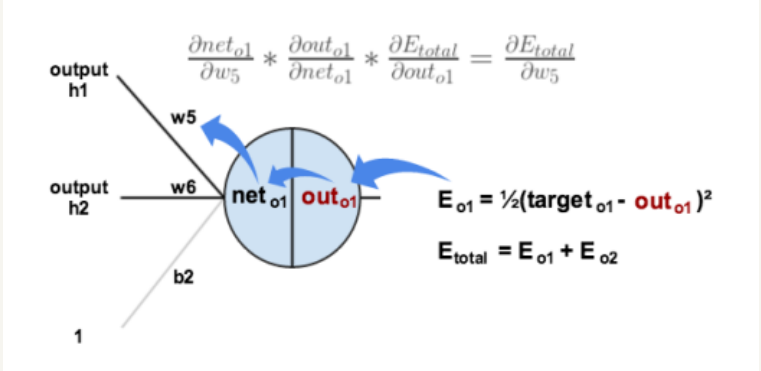
Finally, how much does the total net input of o_1 change with respect to w_5 ?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$



Exemplu

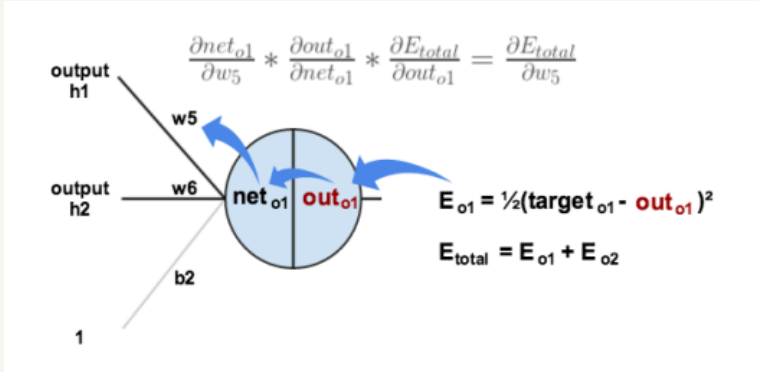
You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$ aka δ_{o1} aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$



Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from $\boldsymbol{\delta}$ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

To decrease the error, we then subtract this value from the current Pondere(multiplied by some **learning rate**, eta, which is set here to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

We can repeat this process to get the new ponderi :

$$w_6^+ = 0.408666186 \qquad w_7^+ = 0.511301270 \qquad w_8^+ = 0.561370121$$

Exemplu

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, w_4.

We need to figure out:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

The process is similar with the one for the ieșire layer, but slightly different to account for the fact that the ieșire of each hidden layer neuron contributes to the ieșire (and therefore error) of multiple ieșire neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both ieșire neurons:

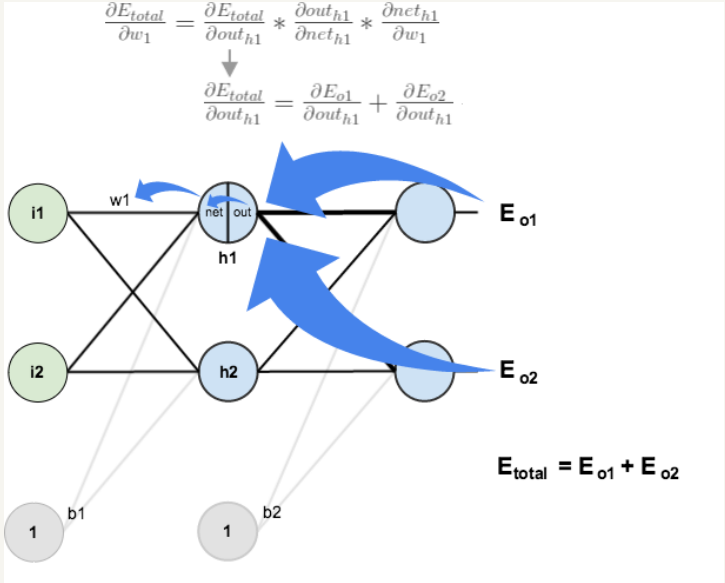
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to : $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$



Exemplu

Plugging them in: $\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$ we get: $\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$

Therefore: $\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$ we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight: $out_{h1} = \frac{1}{1+e^{-net_{h1}}}$

$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the ieșire neuron:

$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$

$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

Putting it all together: $\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$

$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$

Exemplu

We can now Actualizeaz$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$

Repeating this for w_2, w_3 and w_4:

$w_2^+ = 0.19956143$ $w_3^+ = 0.24975114$ $w_4^+ = 0.29950229$

Finally, we've updated all of our weights!
When we fed forward the 0.05 and 0.1 inputs originally, the error on the rețea was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for Exemplu, the error plummets to 0.0000351085. At this point, when we feed forward 0.05 and 0.1, the two outputs Neuroniigenerate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).