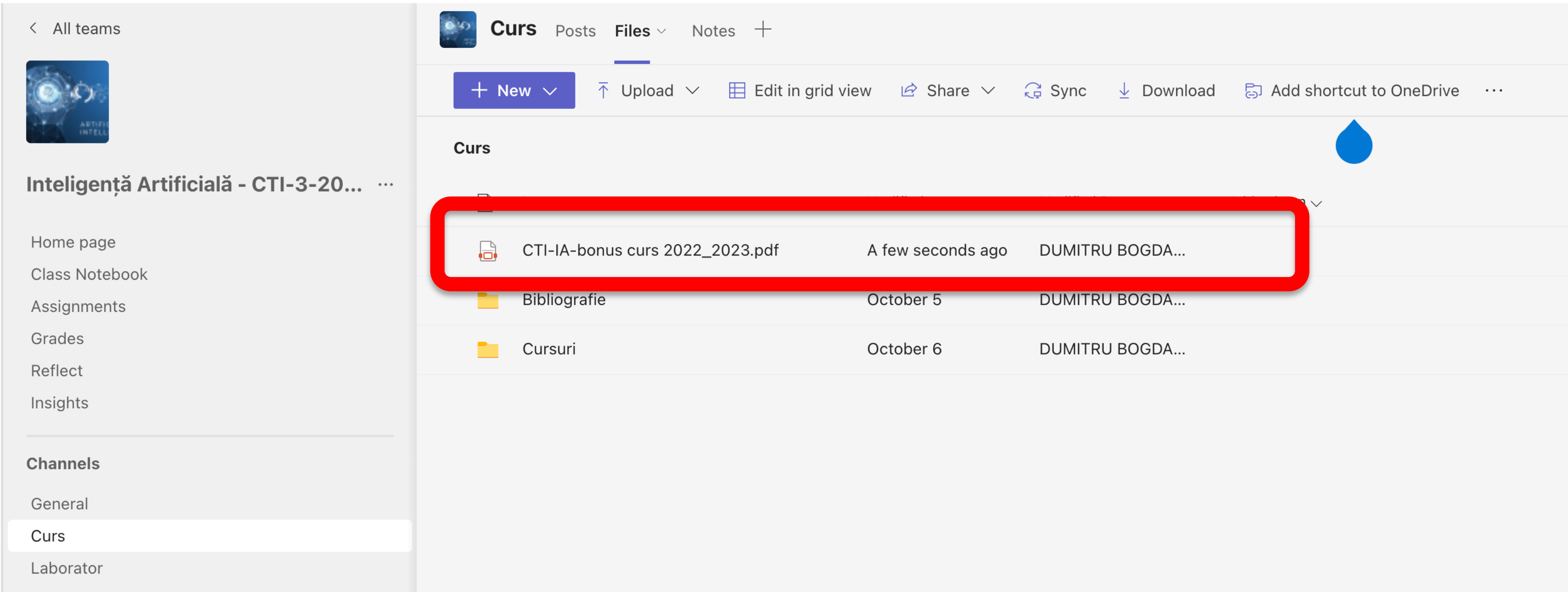
Inteligență Artificială

Bogdan Alexe
bogdan.alexe@fmi.unibuc.ro

Secția Tehnologia Informației, anul III, 2022-2023
Cursul 6


All teams
Inteligență Artificială - CTI-3-20...
Home page
Class Notebook
Assignments
Grades
Reflect
Insights
Channels
General
Curs
Laborator
Curs
Posts
Files
Notes
New
Upload
Edit in grid view
Share
Sync
Download
Add shortcut to OneDrive
Curs
CTI-IA-bonus curs 2022_2023.pdf
A few seconds ago
DUMITRU BOGDA...
Bibliografie
October 5
DUMITRU BOGDA...
Cursuri
October 6
DUMITRU BOGDA...

Proiect – deadline mărit cu o săptămână

- sincronizare bună curs/laborator

- paletă largă de metode clasificare: kNN, Bayes, SVM, perceptron, rețele de perceptroni

- număr mic de participanți până acum (aprox. 50)

Recapitulare – cursul trecut

1. Mașini cu vectori support (SVMs– support vector machines)

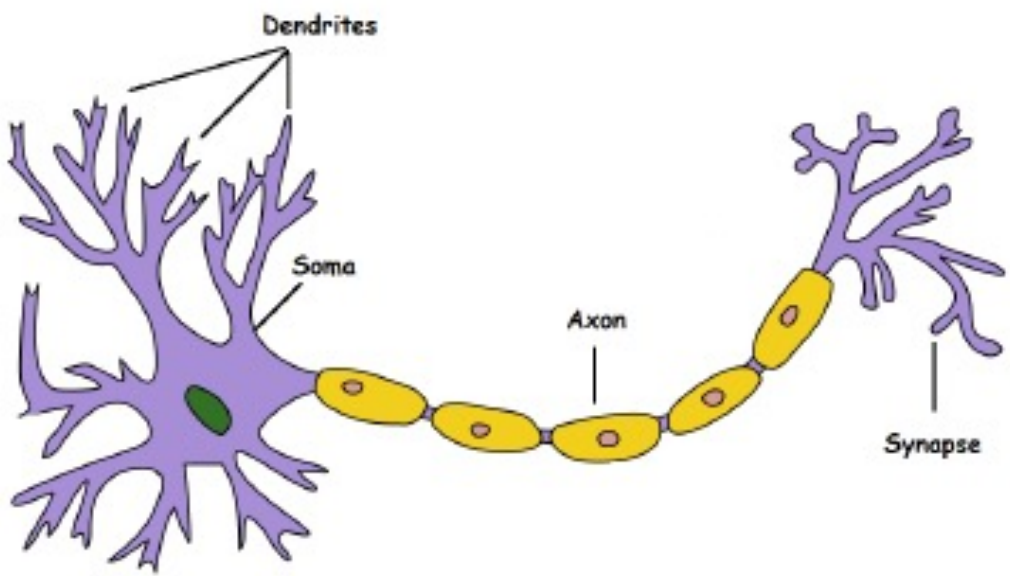
2. Perceptronul

Perceptronul

Neuronul biologic

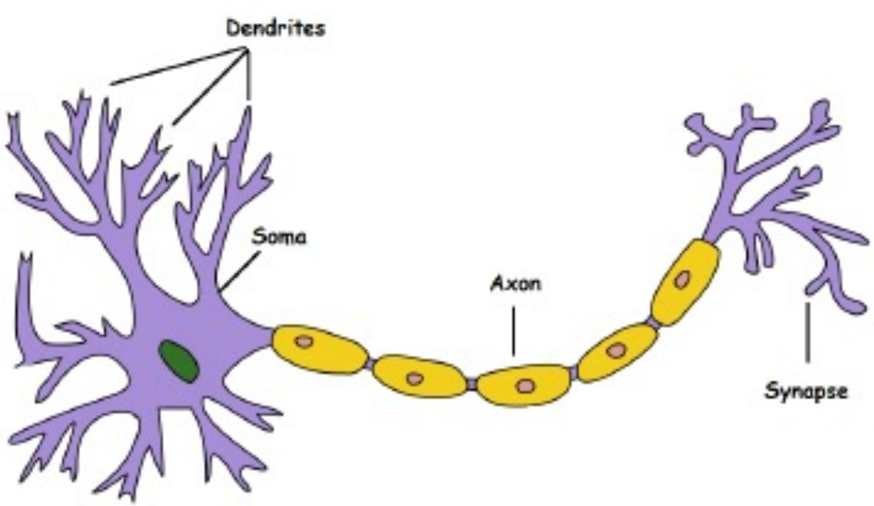
Cum funcționează un neuron:

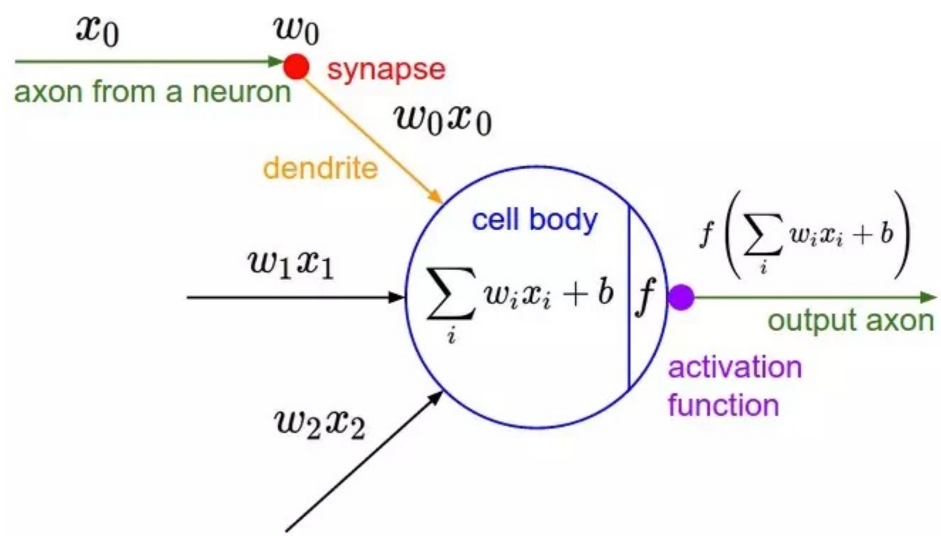
- semnale electrice (intrări) sunt transmise prin ***dendrite***
- aceste semnale duc la acumularea unui potențial electric în ***corpul neuronului (soma)***
- când potențialul electric acumulat depășește un anumit prag (threshold) un semnal electric (ieșirea neuronului) este transmis prin intermediul unui ***axon***
- conectarea axonului cu dendritele altor neuroni se realizează prin **sinapse**

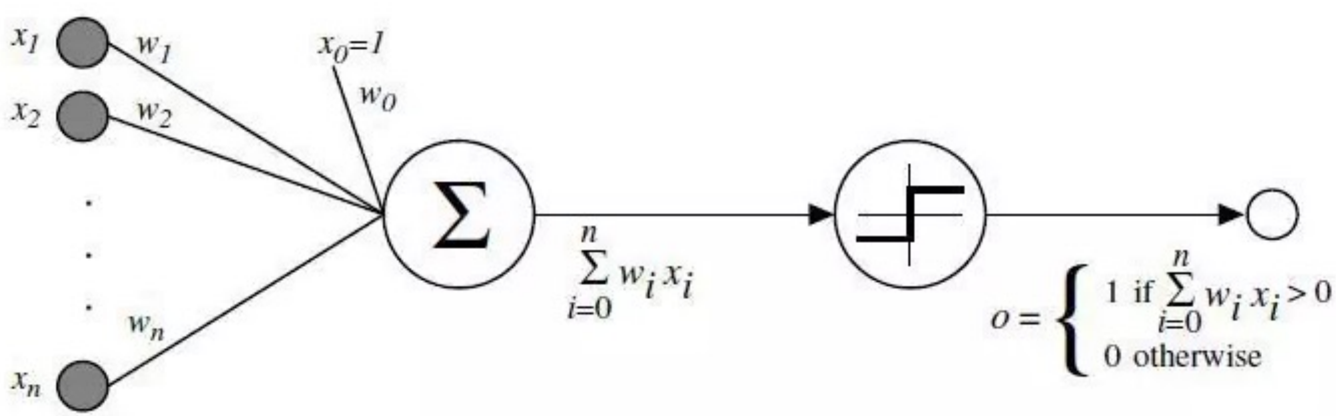


Perceptronul

Perceptronul este un clasificator liniar inspirat de neuronul uman.







Algoritmul de învățare al perceptronului

- Mulțimea de exemple de antrenare:

$$E = \{(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(m)}, y^{(m)})\},\ \vec{x}^{(i)} \in \{0,1\}^{n+1} \left(x_0^{(i)} = 1, \forall i\right),\ y^{(i)} \in \{0,1\}$$

- Ponderile w_j sunt inițializate cu 0 (pe acestea le învățăm)
- Pentru fiecare exemplu de antrenare $(\vec{x}^{(i)}, y^{(i)})$ din mulțimea E:
  - dacă eticheta prezisă = eticheta reală $\hat{y}^{(i)} == y^{(i)}$ nu facem actualizare
  - dacă $\hat{y}^{(i)} == 0$ și $y^{(i)} == 1$ creștem ponderile tuturor intrărilor active
  - dacă $\hat{y}^{(i)} == 1$ și $y^{(i)} == 0$ scădem ponderile tuturor intrărilor active
- Regula de actualizare a perceptronului

$w_j = w_j + \Delta w_j$, unde

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$$

$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } \sum_{j=0}^{n} w_j x_j^{(i)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Cantitatea prin care schimbăm ponderea (rata de învățare)

Setează semnul actualizării

Selectează intrările active

Algoritmul de învățare al perceptronului

```
def perceptron(X, y, n_epochs, η):
    m, n = X.shape # number of samples, number of inputs
    for j in range(n):
        w_j = 0
    for epoch in range(n_epochs): # an “epoch” is a run through all training data.
        for i in range(m):        # a “training step” is one    update of the weights.
            ŷ^(i) = unit_step_function(Σ_{j=0}^{n} w_j x_j^(i))
            for j in range(n):
                w_j += η(y^(i) − ŷ^(i)) · x_j^(i)
```

$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } \sum_{j=0}^{n} w_j x_j^{(i)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Exemplu – învățarea funcției OR

	x_0	x_1	x_2
$\vec{x}^{(1)}$	1	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	1	1	1

y
0
1
1
1

w_0	w_1	w_2
0	0	0

Accuracy: 25%

$\hat{y}$
0
0
0
0

$\eta = 1$

$\Delta w_j = \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$

Exemplu – învățarea funcției OR

	x_0	x_1	x_2
$\vec{x}^{(1)}$	1	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	1	1	1

y
0
1
1
1

w_0	w_1	w_2
1	0	1

Accuracy: 75%

$\hat{y}$
1
1
1
1

	Δw_0	Δw_1	Δw_2
$\vec{x}^{(1)}$	0	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	0	0	0
$\vec{x}^{(4)}$	0	0	0

Epoch 1

$\eta = 1$

$\Delta w_j = \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$

Exemplu – învățarea funcției OR

	x_0	x_1	x_2
$\vec{x}^{(1)}$	1	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	1	1	1

y
0
1
1
1

w_0	w_1	w_2
1	1	1

Accuracy: 75%

$\hat{y}$
1
1
1
1

	Δw_0	Δw_1	Δw_2
$\vec{x}^{(1)}$	-1	0	0
$\vec{x}^{(2)}$	0	0	0
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	0	0	0

Epoch 2

$\eta = 1$

$\Delta w_j = \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$

Exemplu – învățarea funcției OR

	x_0	x_1	x_2
$\vec{x}^{(1)}$	1	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	1	1	1

y
0
1
1
1

w_0	w_1	w_2
0	1	1

Accuracy: 100%

$\hat{y}$
0
1
1
1

	Δw_0	Δw_1	Δw_2
$\vec{x}^{(1)}$	-1	0	0
$\vec{x}^{(2)}$	0	0	0
$\vec{x}^{(3)}$	0	0	0
$\vec{x}^{(4)}$	0	0	0

Epoch 3

$\eta = 1$

$\Delta w_j = \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$

Exemplu – învățarea funcției OR

	x_0	x_1	x_2
$\vec{x}^{(1)}$	1	0	0
$\vec{x}^{(2)}$	1	0	1
$\vec{x}^{(3)}$	1	1	0
$\vec{x}^{(4)}$	1	1	1

y
0
1
1
1

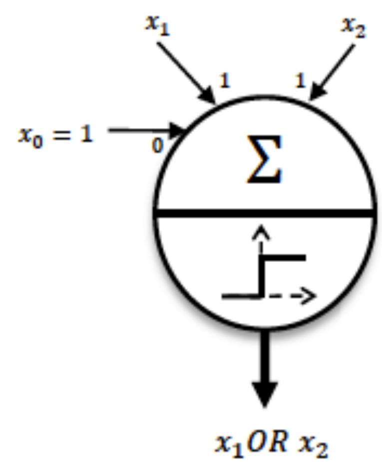
w_0	w_1	w_2
0	1	1

Accuracy: 100%

$\hat{y}$
0
1
1
1

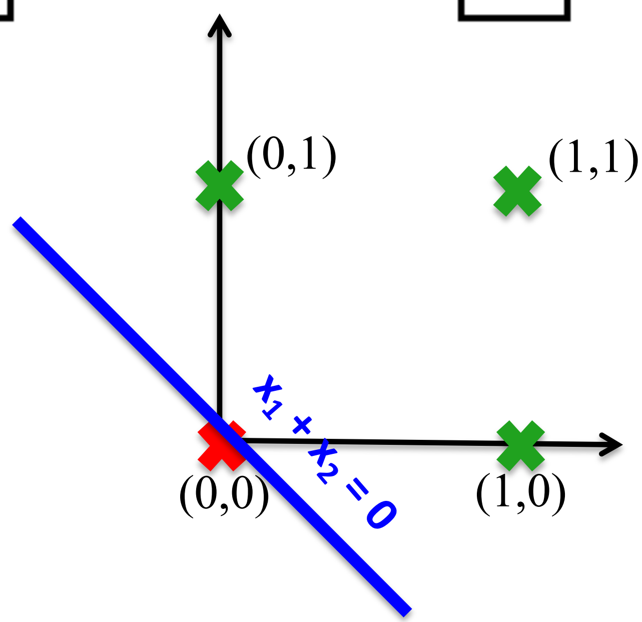
OR

Perceptronul a învațat funcția OR în 3 epoci



Eticheta 1

Eticheta 0



$w_0x_0 + w_1x_1 + w_2x_2 > 0$

$x_1 + x_2 > 0$

$w_0x_0 + w_1x_1 + w_2x_2 \leq 0$

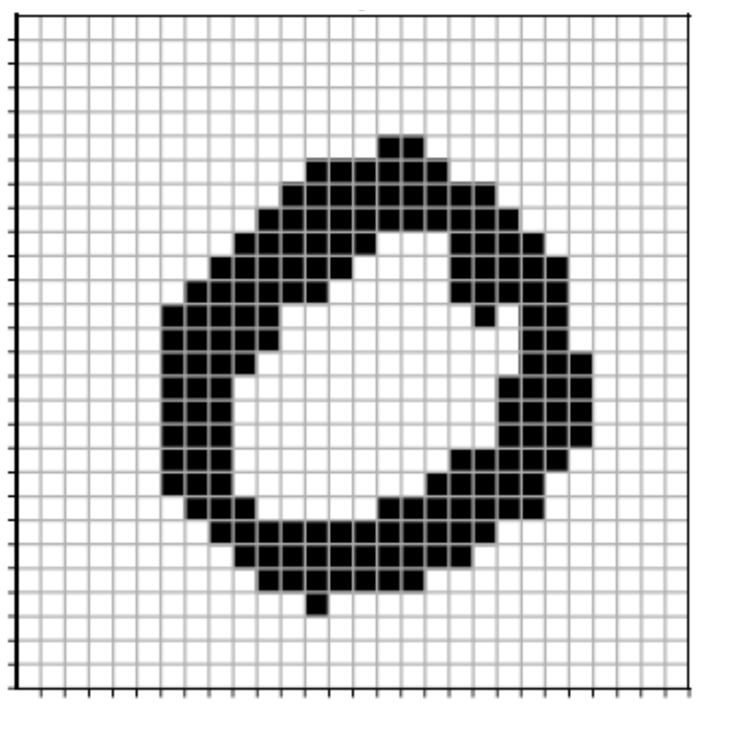
$x_1 + x_2 \leq 0$

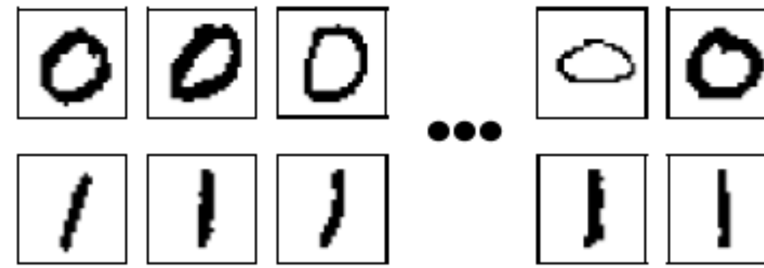
$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } \sum_{j=0}^{n} w_j x_j^{(i)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Exemplu – Recunoașterea cifrelor

Putem antrena un perceptron care să discrimineze între imagini cu cifre scrise de mână cu cifrele 0 și 1?

28 × 28 valori binare

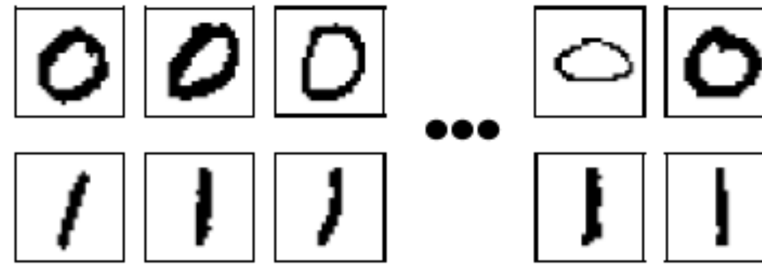




100 de exemple pentru fiecare clasă

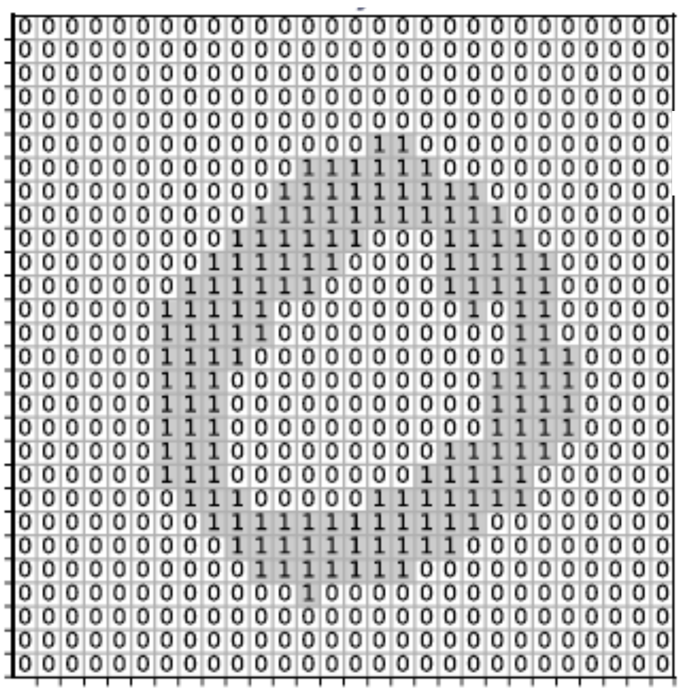
Exemplu – Recunoașterea cifrelor

Putem antrena un perceptron care să discrimineze între imagini cu cifre scrise de mână cu cifrele 0 și 1?



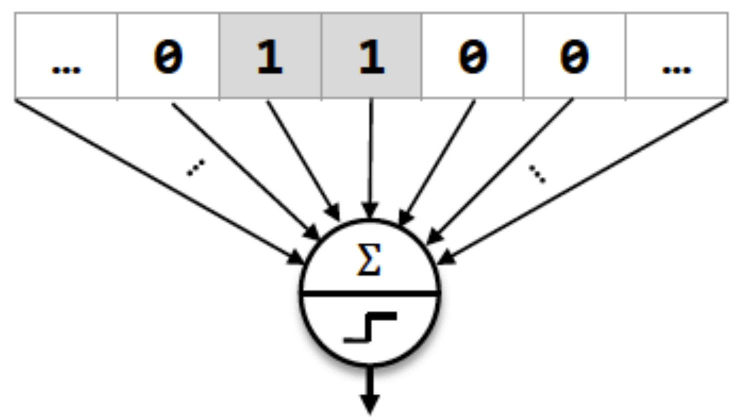
100 de exemple pentru fiecare clasă

28 × 28 valori binare

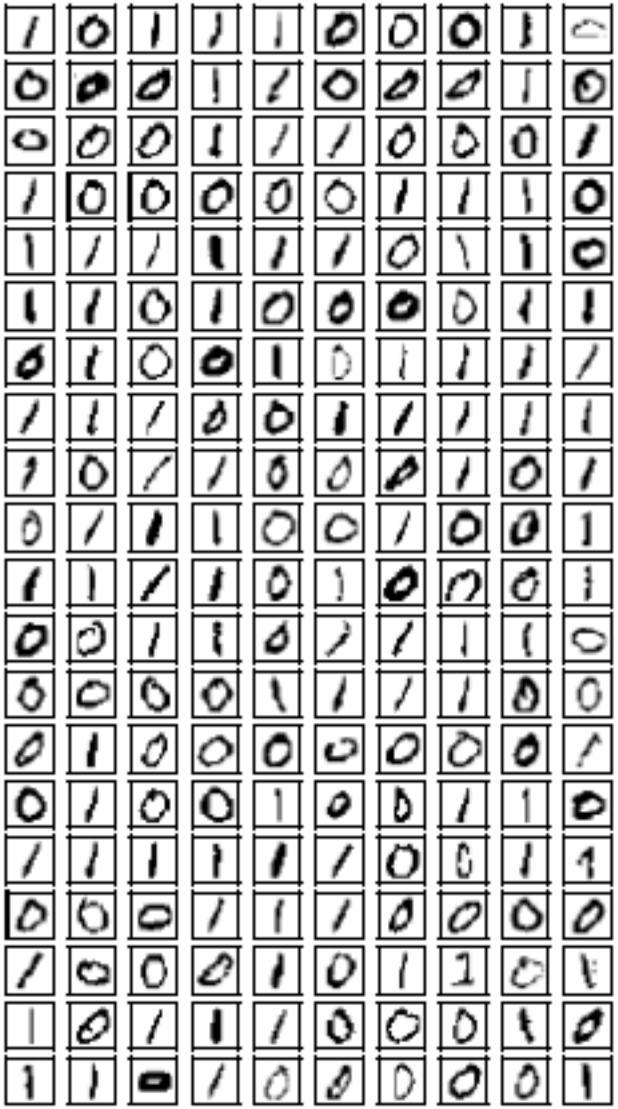


liniarizare

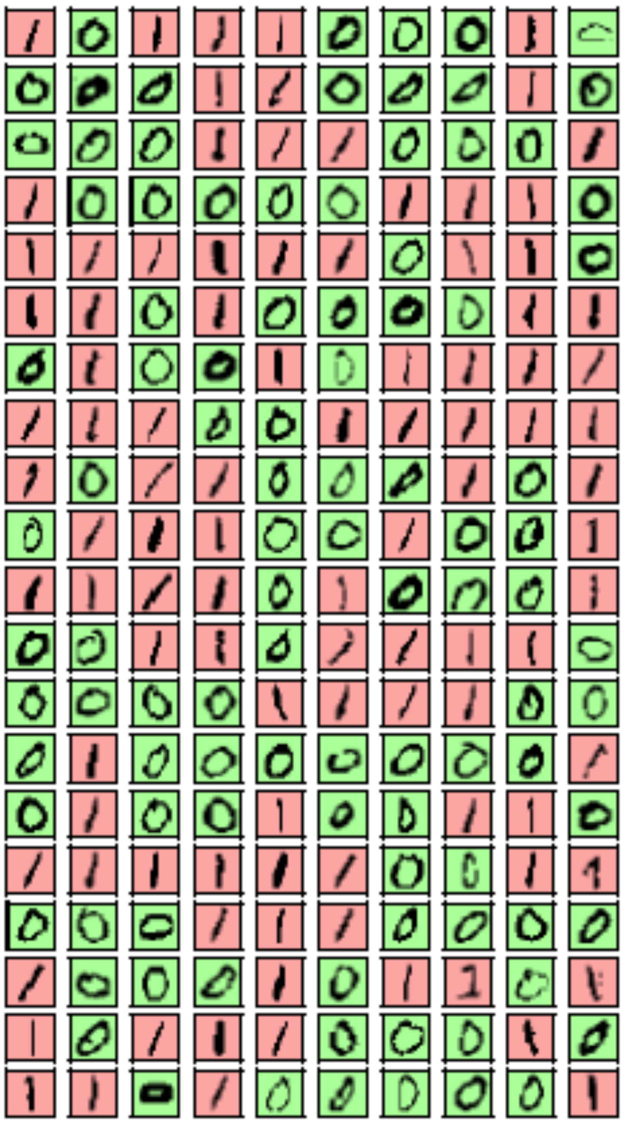
784 de intrări



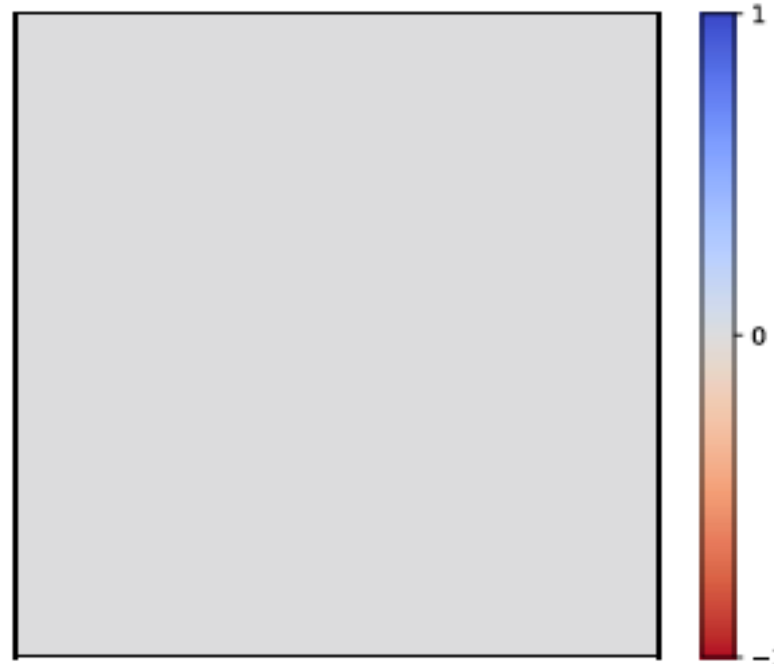
Exemplu – Recunoașterea cifrelor



Exemplu – Recunoașterea cifrelor

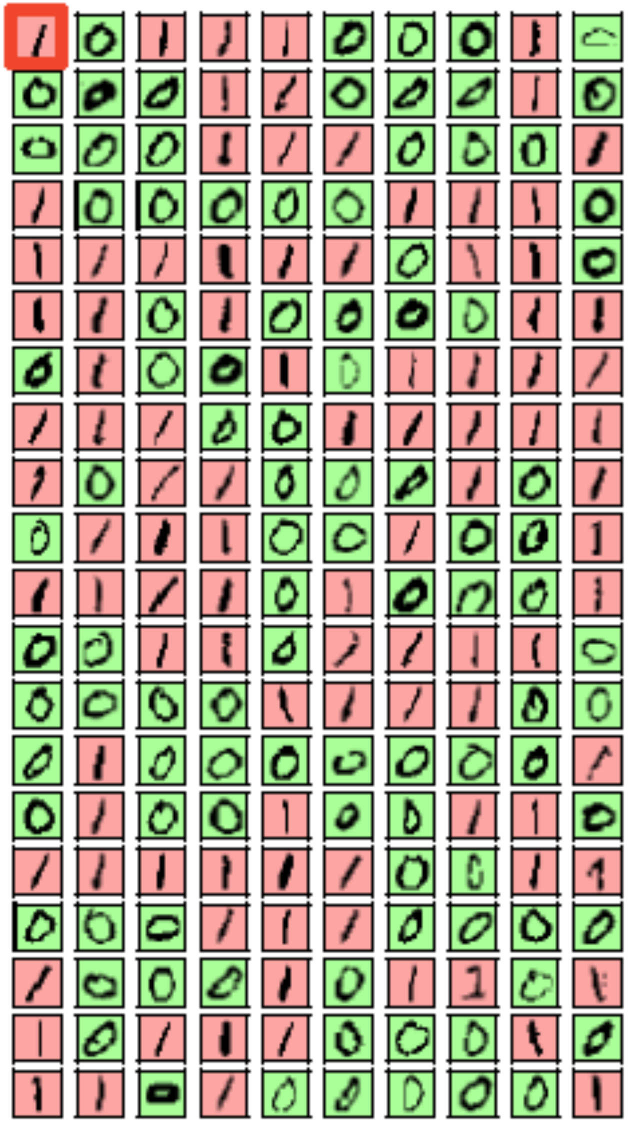


Valorile ponderilor (inițial 0) afișate ca o imagine 28 ×28

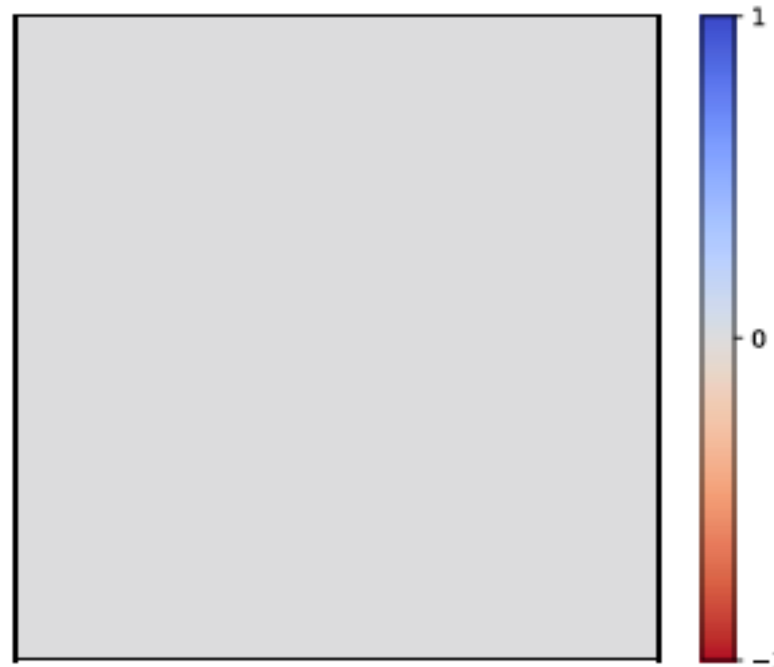


Pasul 0 de antrenare
Acuratețe 50%

Exemplu – Recunoașterea cifrelor

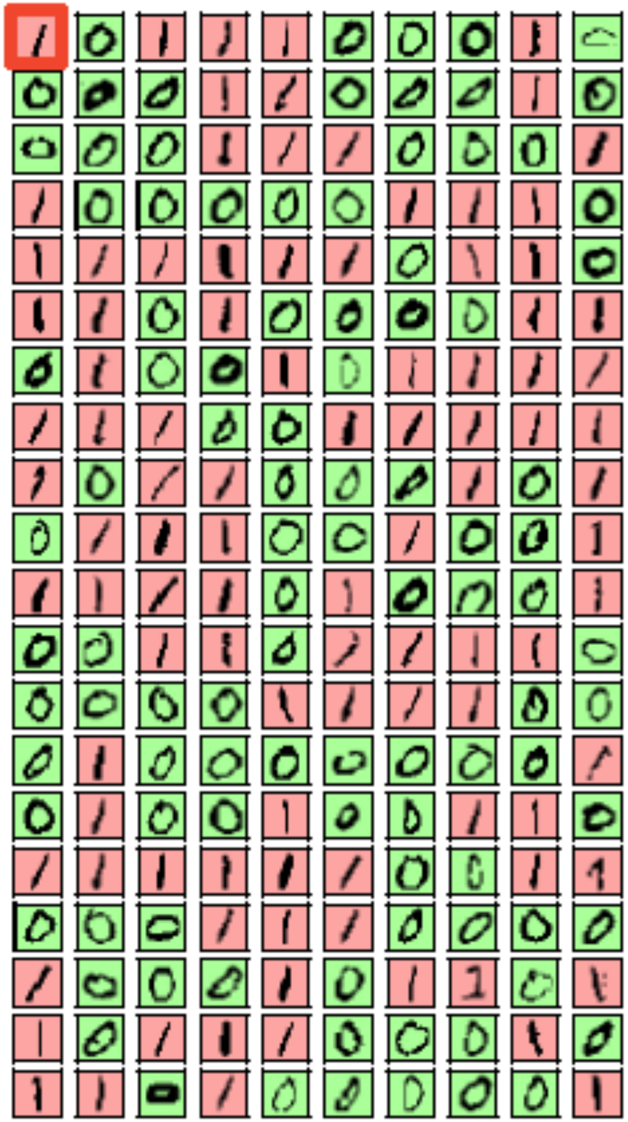


Valorile ponderilor afișate ca o imagine 28 ×28



Pasul 1 de antrenare
Acuratețe 50%

Exemplu – Recunoașterea cifrelor



Regula de actualizare a perceptronului

$w_j = w_j + \Delta w_j$, unde

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$$

Cantitatea prin care schimbăm ponderea (rata de învățare)

Setează semnul actualizării

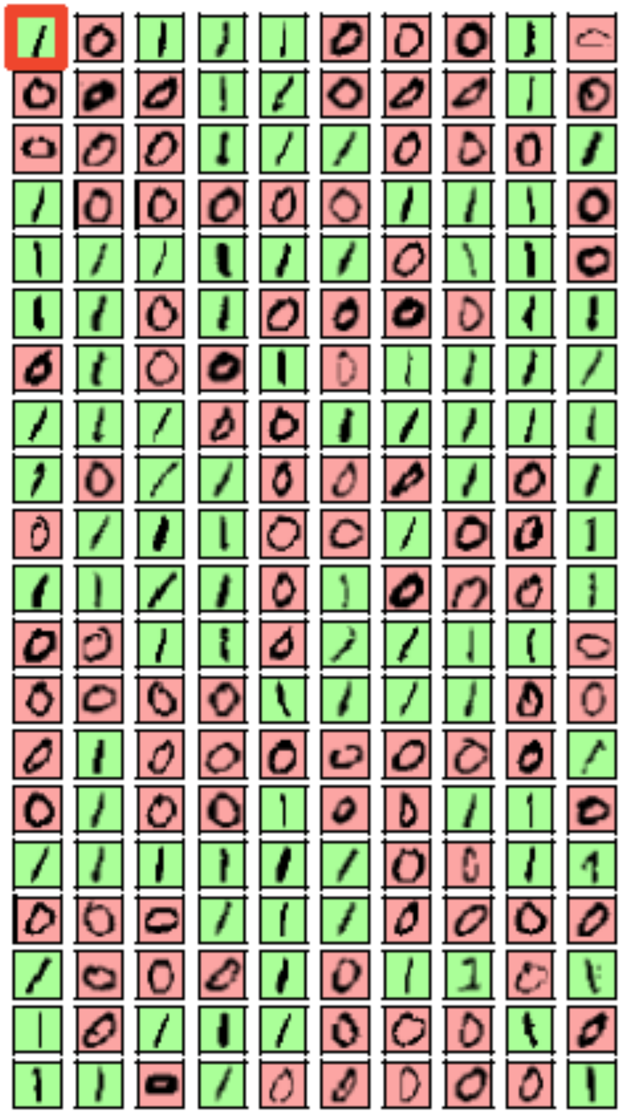
Selectează intrările active

Pentru exemple de antrenare misclasificate:

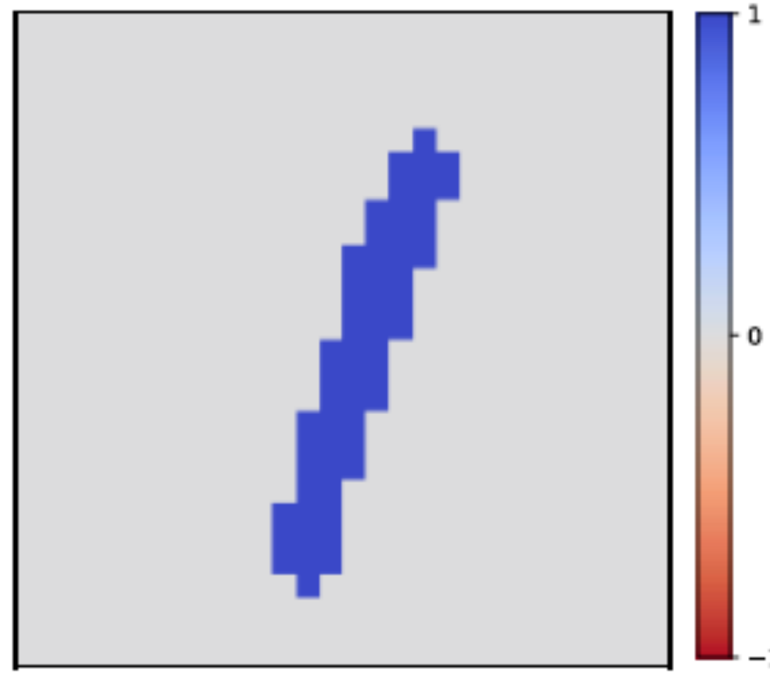
-daca $y^{(i)}$= 1 atunci adaug η*(1-0)*1 pentru ponderile corespunzatoare pixelilor negri (cei din exemplul de antrenare curent cu cifra 1)

-daca $y^{(i)}$= 0 atunci adaug η*(0-1)*1 pentru ponderile corespunzatoare pixelilor negri (cei din exemplul de antrenare curent cu cifra 0)

Exemplu – Recunoașterea cifrelor

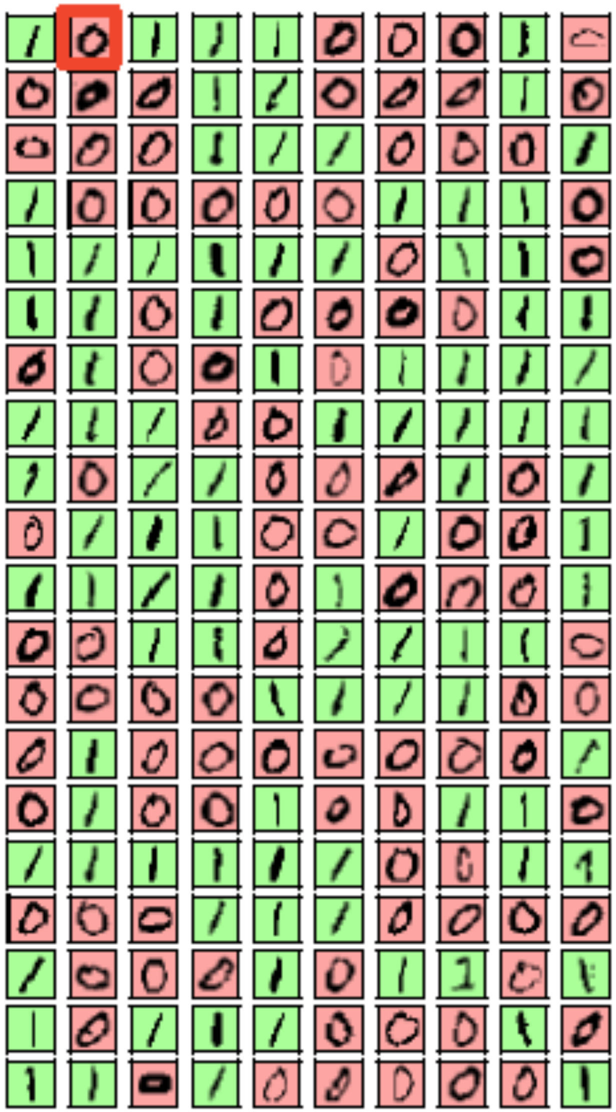


Valorile ponderilor afișate ca o imagine 28 ×28

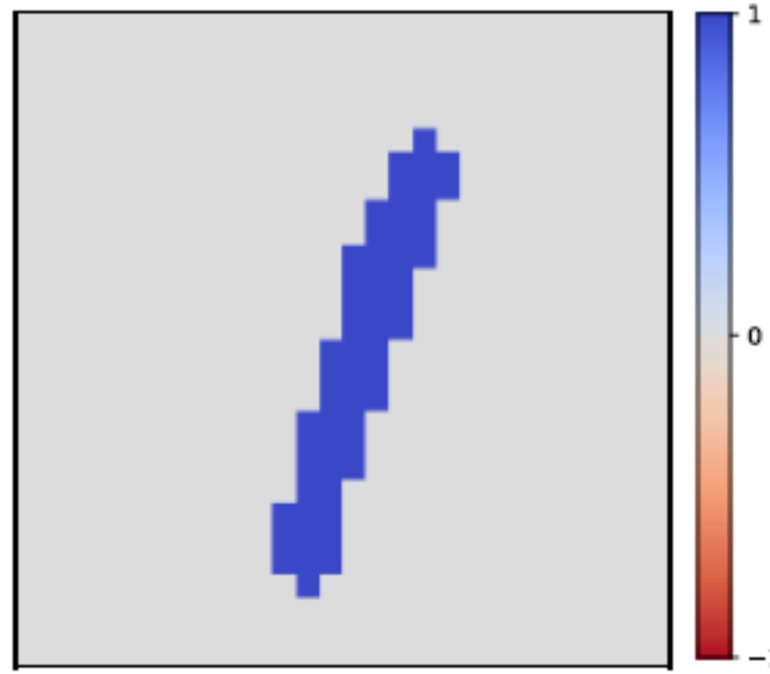


Pasul 1 de antrenare
Acuratețe 50%

Exemplu – Recunoașterea cifrelor

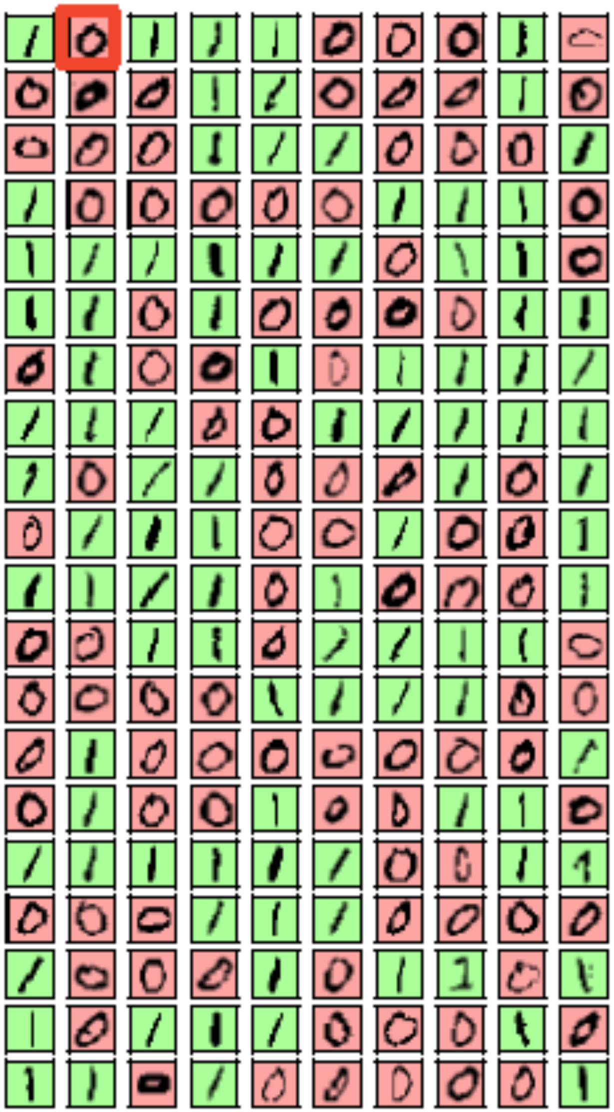


Valorile ponderilor afișate ca o imagine 28 ×28



Pasul 2 de antrenare
Acuratețe 50%

Exemplu – Recunoașterea cifrelor



Regula de actualizare a perceptronului

$w_j = w_j + \Delta w_j$, unde

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}$$

Cantitatea prin care schimbăm ponderea (rata de învățare)

Setează semnul actualizării

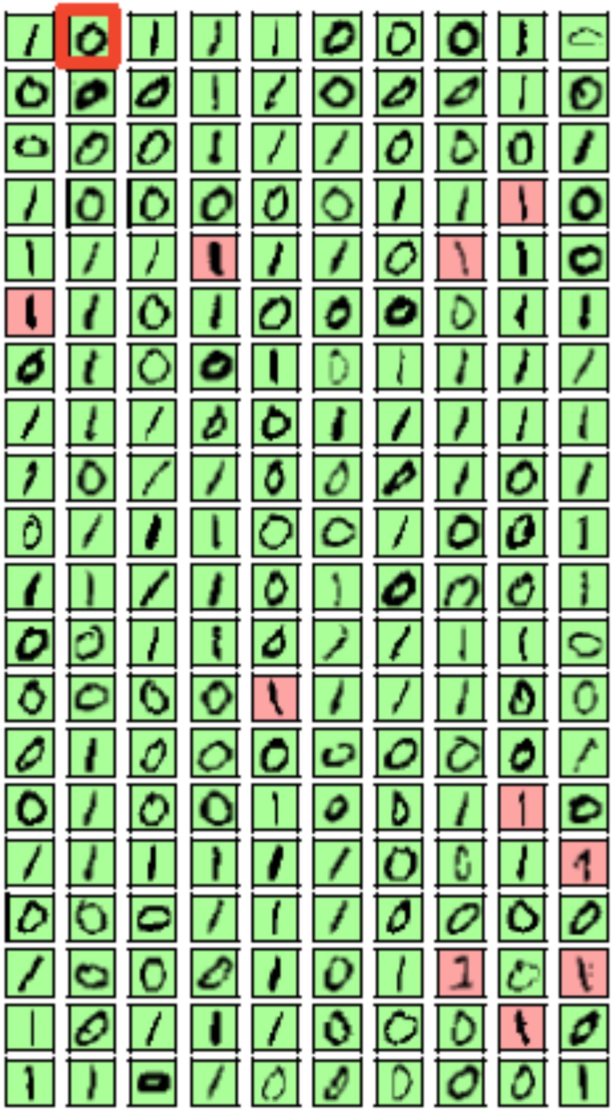
Selectează intrările active

Pentru exemple de antrenare misclasificate:

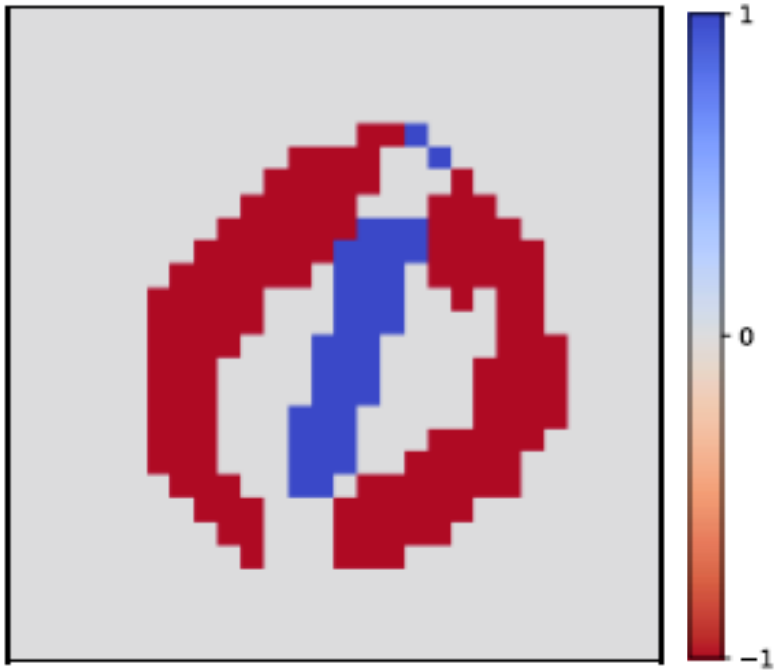
-daca $y^{(i)}$= 1 atunci adaug η*(1-0)*1 pentru ponderile corespunzatoare pixelilor negri (cei din exemplul de antrenare curent cu cifra 1)

-daca $y^{(i)}$= 0 atunci adaug η*(0-1)*1 pentru ponderile corespunzatoare pixelilor negri (cei din exemplul de antrenare curent cu cifra 0)

Exemplu – Recunoașterea cifrelor

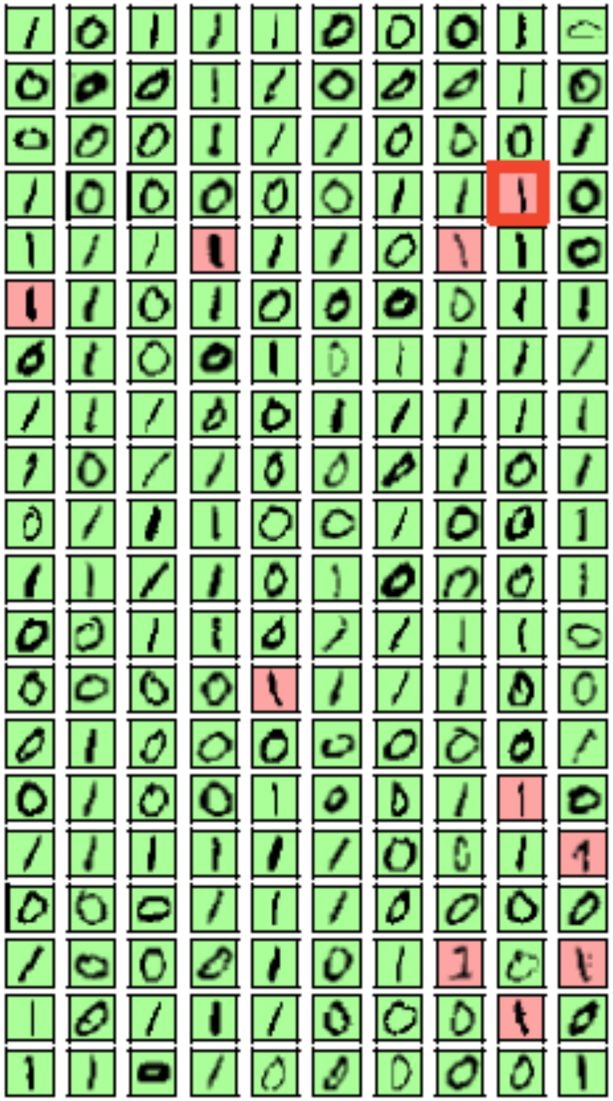


Valorile ponderilor afișate ca o imagine 28 ×28

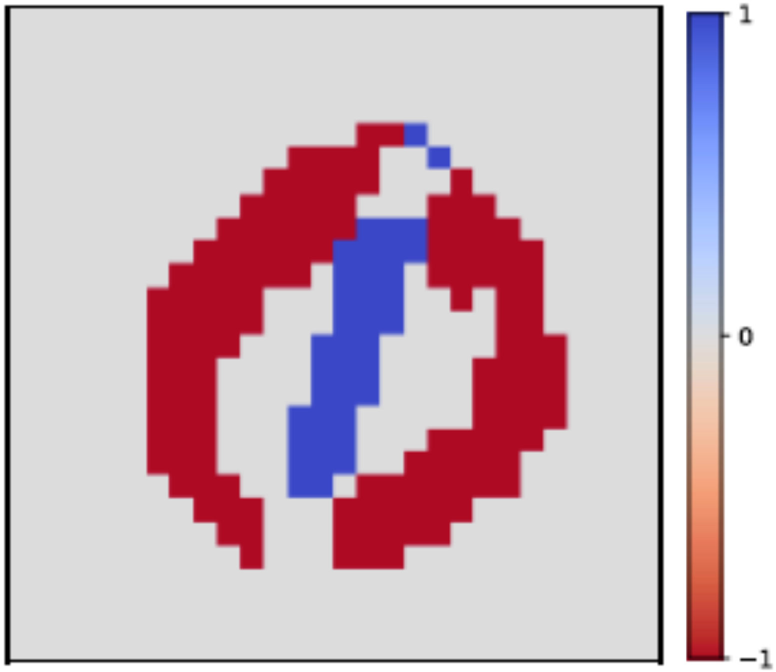


Pasul 2 de antrenare
Acuratețe 95%

Exemplu – Recunoașterea cifrelor

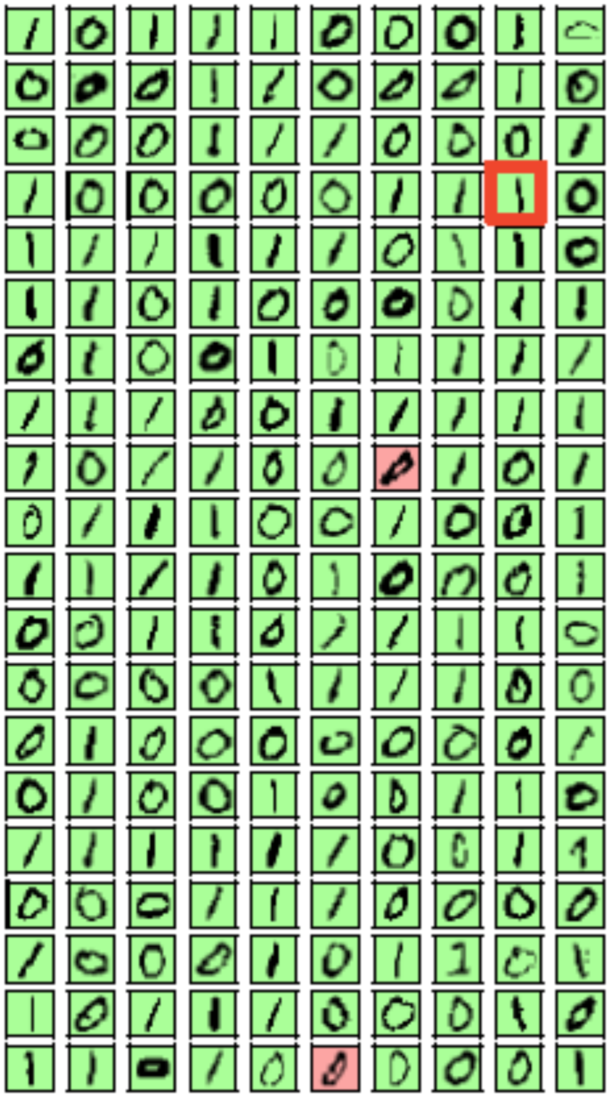


Valorile ponderilor afișate ca o imagine 28 ×28

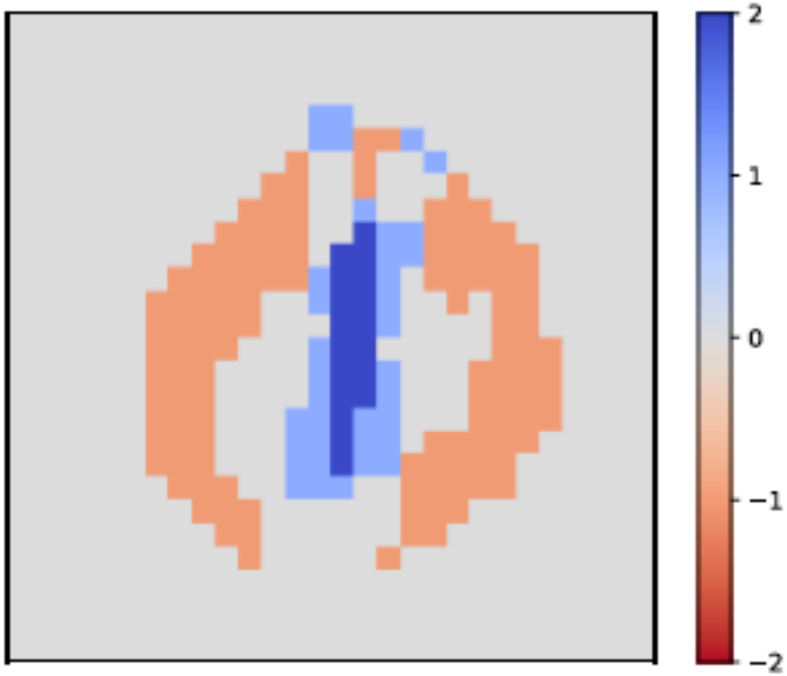


Pasul 39 de antrenare
Acuratețe 95%

Exemplu – Recunoașterea cifrelor

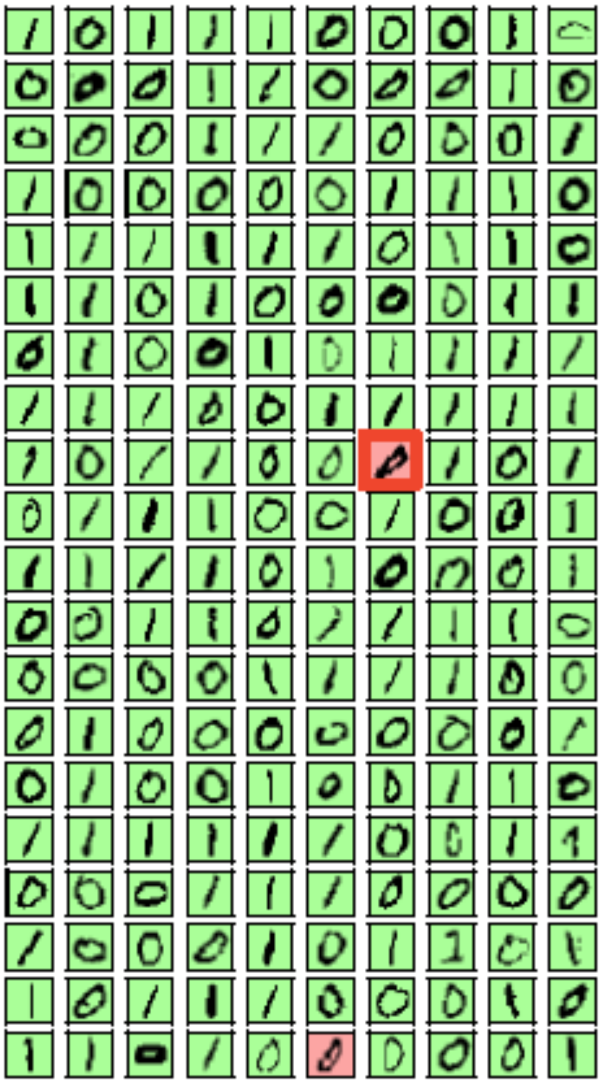


Valorile ponderilor afișate ca o imagine 28 ×28

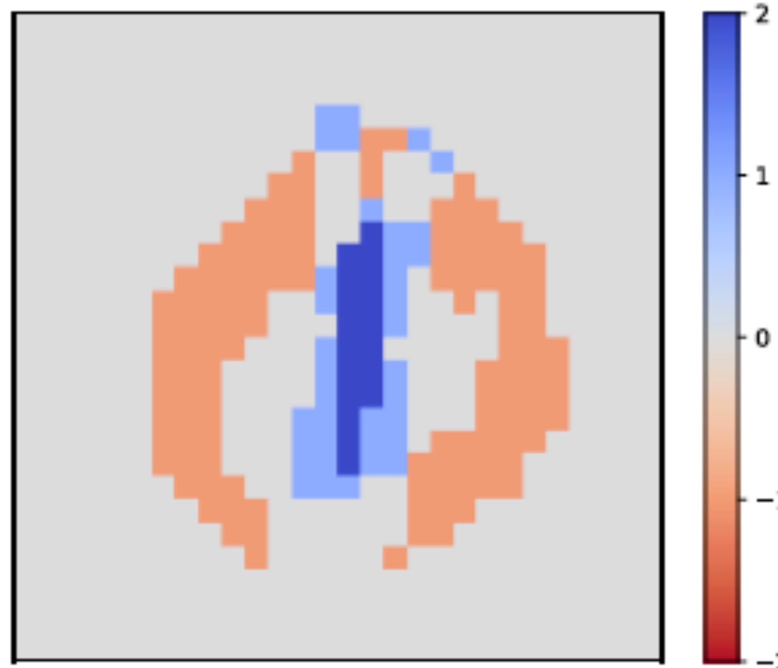


Pasul 39 de antrenare
Acuratețe 99%

Exemplu – Recunoașterea cifrelor

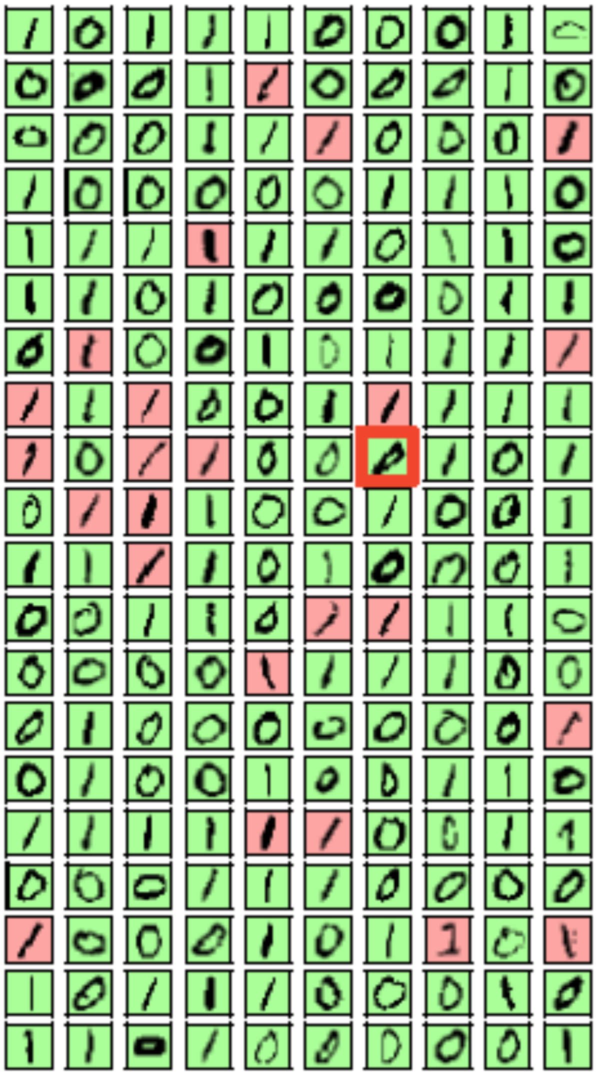


Valorile ponderilor afișate ca o imagine 28 ×28

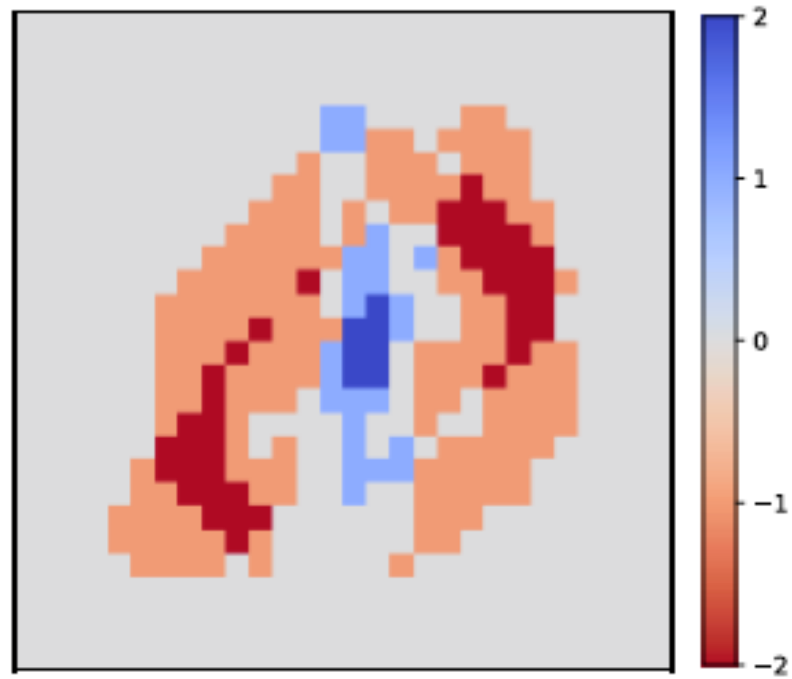


Pasul 87 de antrenare
Acuratețe 99%

Exemplu – Recunoașterea cifrelor

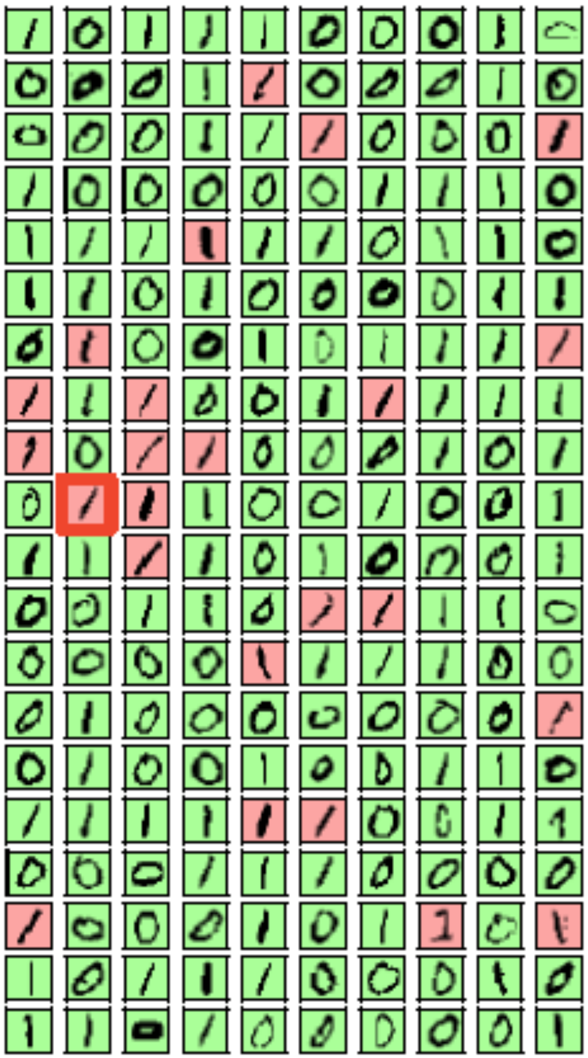


Valorile ponderilor afișate ca o imagine 28 ×28

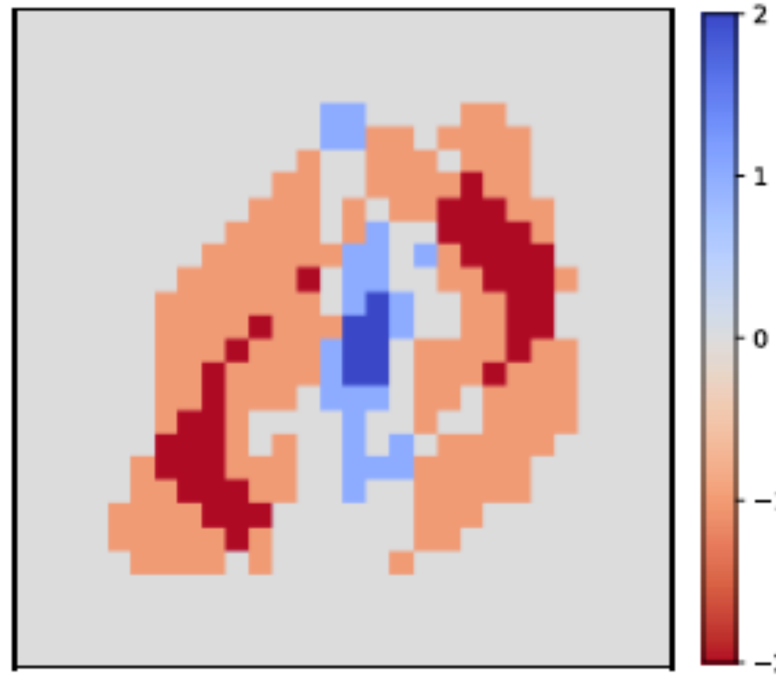


Pasul 87 de antrenare
Acuratețe 88%

Exemplu – Recunoașterea cifrelor

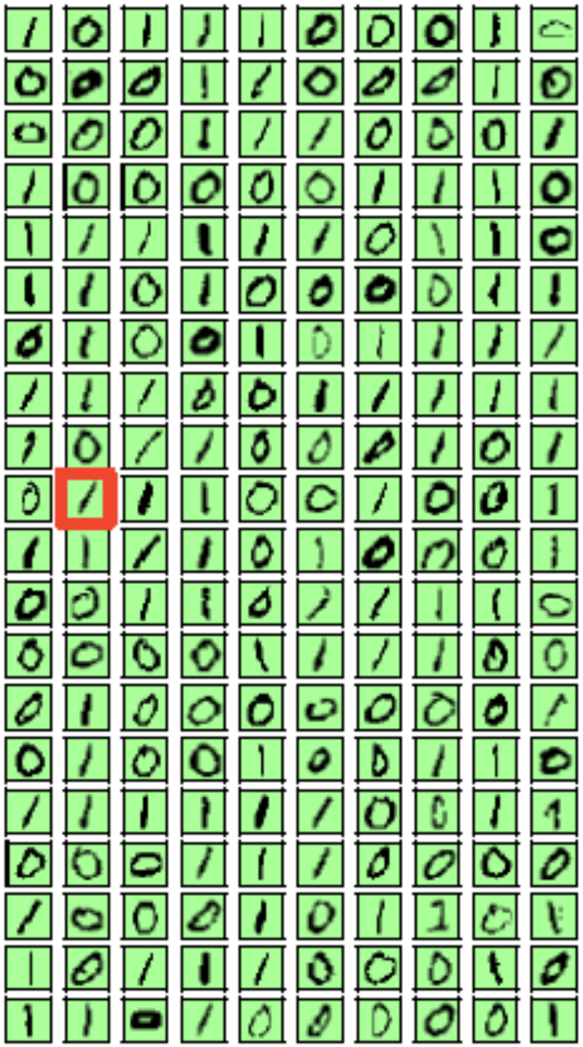


Valorile ponderilor afișate ca o imagine 28 ×28

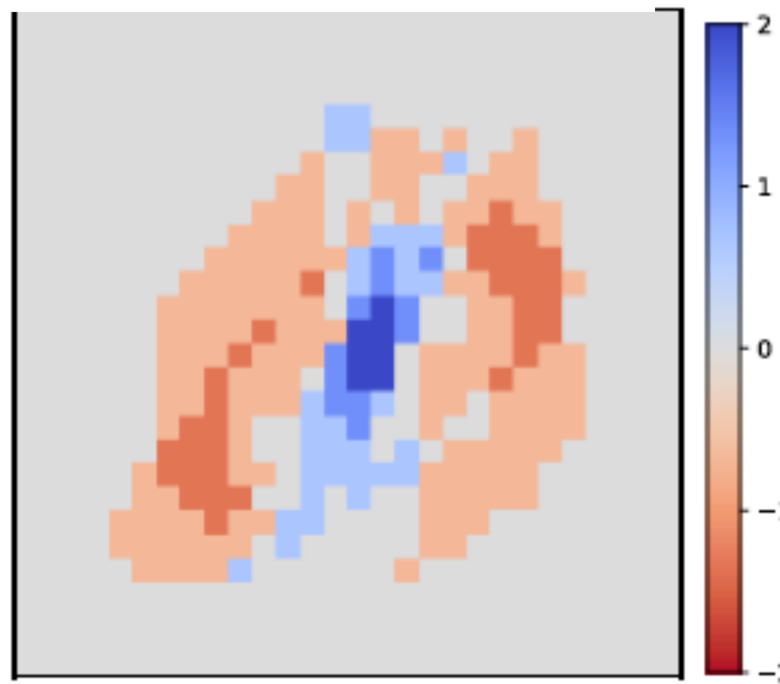


Pasul 92 de antrenare
Acuratețe 88%

Exemplu – Recunoașterea cifrelor

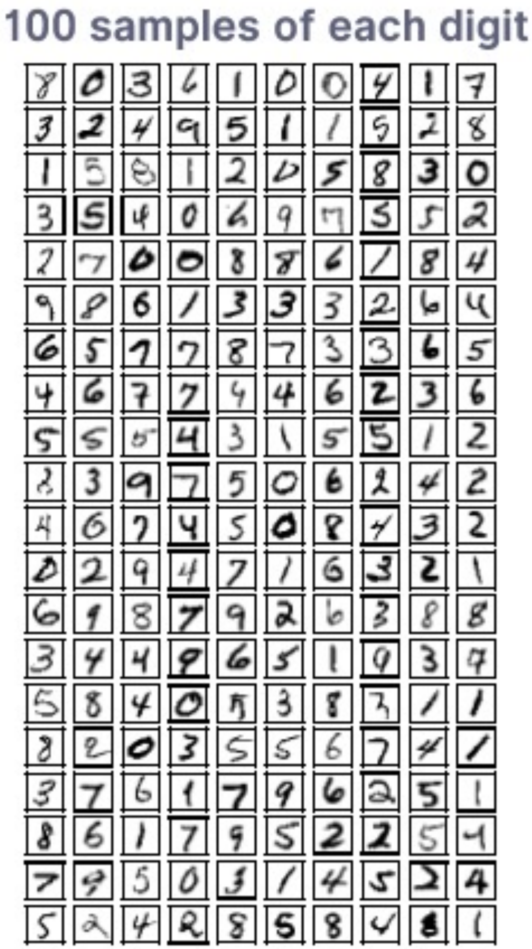


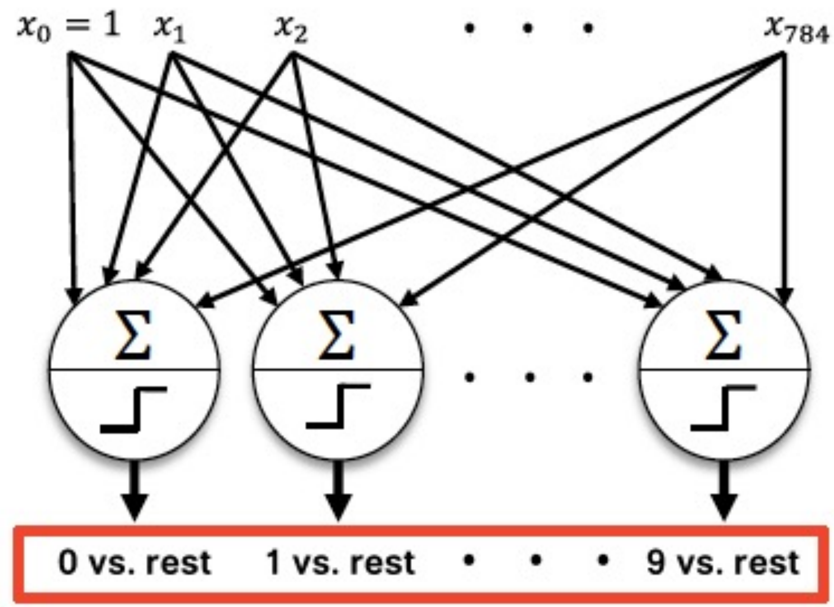
Valorile ponderilor afișate ca o imagine 28 ×28



Pasul 92 de antrenare
Acuratețe 100%

Exemplu – Recunoașterea cifrelor





reprezentare one-hot
(o componentă 1, restul 0)

O retea poate prezice clase multiple folosind un neuron separat pentru fiecare clasă
Pe problema dată atinge ușor acuratețe de ~ 99%

Percerptronul în IA

Extrase din ziarul The New York Times, 8 iulie, 1958

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI)—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

- http://jcblackmon.com/wp-content/uploads/2018/01/MBC-Rosenblatt-Perceptron-NYT-article.jpg.pdf

Limitările perceptronului

Limitările perceptronului

- Perceptronul învață funcțiile booleene AND/OR și reușește să rezolve probleme de recunoaștere a cifrelor. Probleme simple sau grele?

- Considerăm funcția XOR (exclusiv OR):

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

Pentru ca un Perceptron să învețe funcția XOR e nevoie ca:

$$w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0$$
$$w_0 + 0 \cdot w_1 + 1 \cdot w_2 \geq 0$$
$$w_0 + 1 \cdot w_1 + 0 \cdot w_2 \geq 0$$
$$w_0 + 1 \cdot w_1 + 1 \cdot w_2 < 0$$

Limitările perceptronului

- Perceptronul învață funcțiile booleene AND/OR și reușește să rezolve probleme de recunoaștere a cifrelor. Probleme simple sau grele?

- Considerăm funcția XOR (exclusiv OR):

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

Pentru ca un Perceptron să învețe funcția XOR e nevoie ca:

$$w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0 \Rightarrow \boldsymbol{w_0 < 0}$$
$$w_0 + 0 \cdot w_1 + 1 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_2$$
$$w_0 + 1 \cdot w_1 + 0 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_1$$
$$w_0 + 1 \cdot w_1 + 1 \cdot w_2 < 0 \Rightarrow w_0 < -w_1 - w_2$$

Limitările perceptronului

- Perceptronul învață funcțiile booleene AND/OR și reușește să rezolve probleme de recunoaștere a cifrelor. Probleme simple sau grele?

- Considerăm funcția XOR (exclusiv OR):

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

Pentru ca un Perceptron să învețe funcția XOR e nevoie ca:

$$w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0 \Rightarrow \boldsymbol{w_0 < 0}$$

$$\left.\begin{array}{l} \left.\begin{array}{l} w_0 + 0 \cdot w_1 + 1 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_2 \\ w_0 + 1 \cdot w_1 + 0 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_1 \end{array}\right\} 2w_0 \geq -w_1 - w_2 \\ w_0 + 1 \cdot w_1 + 1 \cdot w_2 < 0 \Rightarrow w_0 < -w_1 - w_2 \end{array}\right\} 2w_0 > w_0 \Rightarrow \mathbf{w_0 > 0}$$

Limitările perceptronului

- Perceptronul învață funcțiile booleene AND/OR și reușește să rezolve probleme de recunoaștere a cifrelor. Probleme simple sau grele?
- Considerăm funcția XOR (exclusiv OR):

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

Pentru ca un Perceptron să învețe funcția XOR e nevoie ca:

$$w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0 \Rightarrow \boldsymbol{w_0 < 0}$$

$$\left.\begin{array}{l} \left.\begin{array}{l} w_0 + 0 \cdot w_1 + 1 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_2 \\ w_0 + 1 \cdot w_1 + 0 \cdot w_2 \geq 0 \Rightarrow w_0 \geq -w_1 \end{array}\right\} 2w_0 \geq -w_1 - w_2 \\ w_0 + 1 \cdot w_1 + 1 \cdot w_2 < 0 \Rightarrow w_0 < -w_1 - w_2 \end{array}\right\} 2w_0 > w_0 \Rightarrow \boldsymbol{w_0 > 0}$$

Contradicție

- Perceptronul nu poate învăța XOR oricât de mulți pași ar face
- Mai mult, Perceptronul poate învață numai clase care sunt liniar separabile

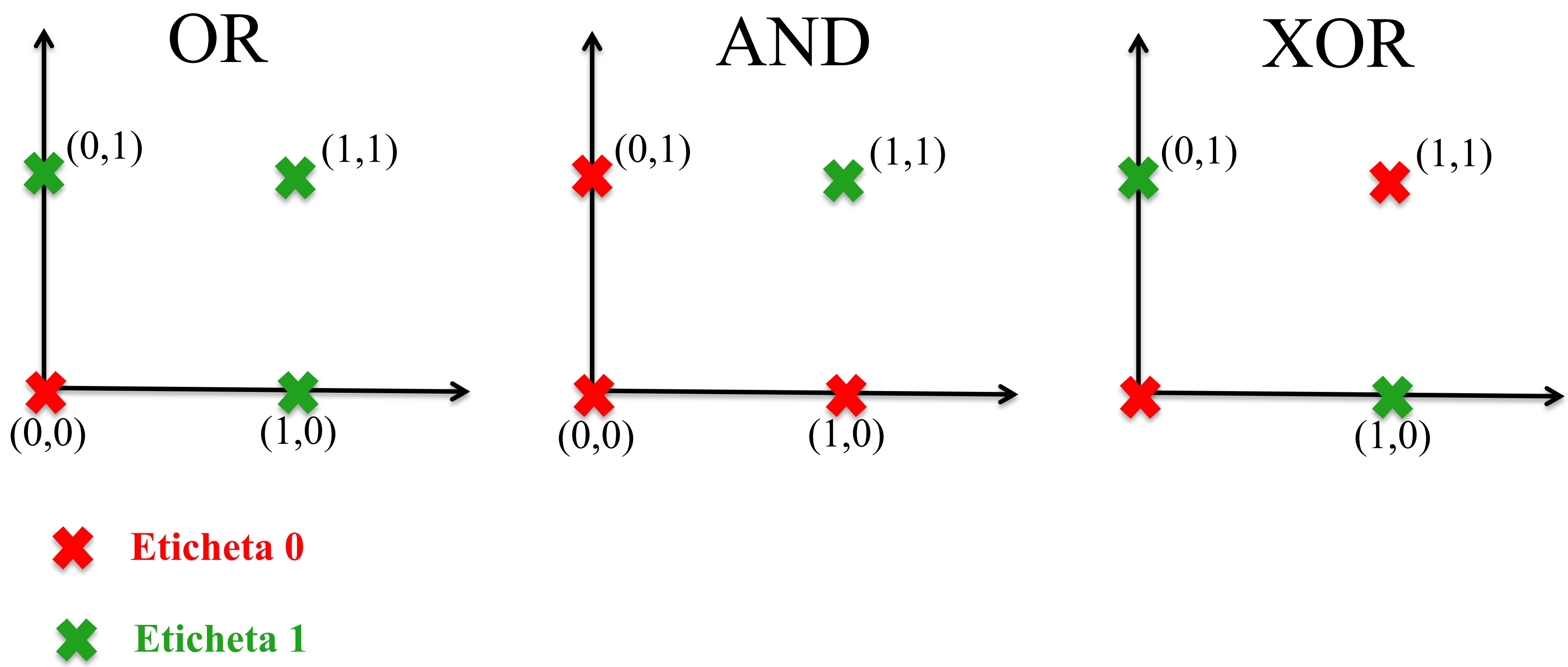
Teorema de convergență a perceptronului

- Dacă mulțimea de antrenare E este liniar separabilă cu margine γ, algoritmul de învățare a perceptronului este garantat că va converge într-un număr finit de pași către o soluție în care nu se fac greșeli pe mulțimea de antrenare

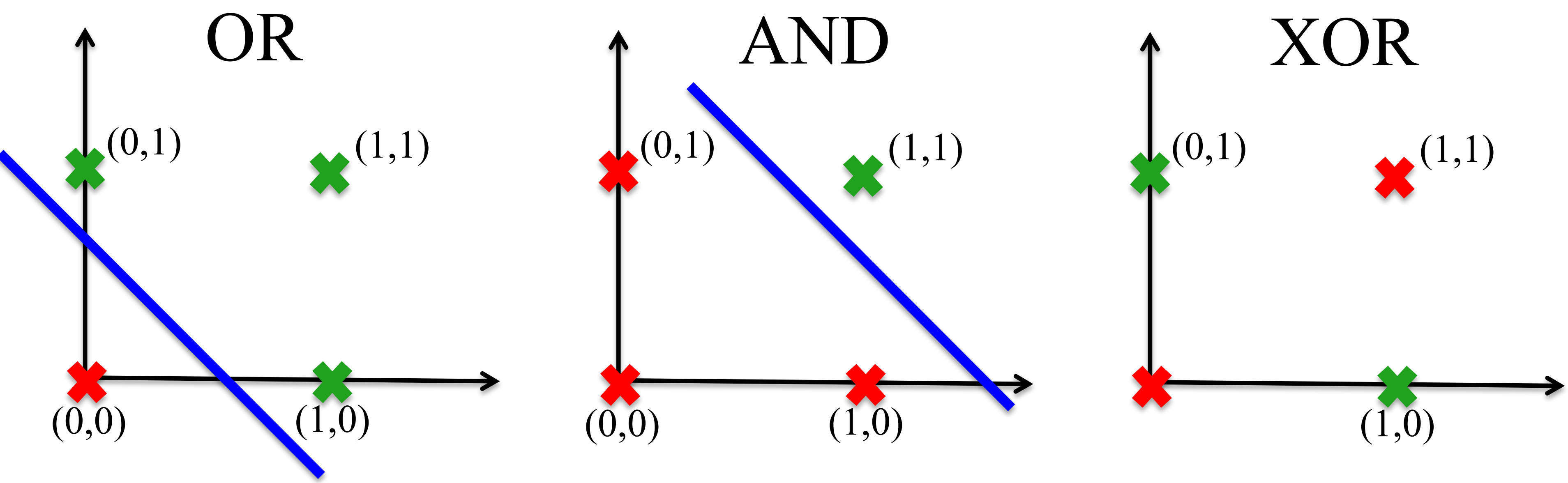
- Numărul de pași k satisface relația $k \leq \frac{R^2}{\gamma^2}$, unde R este raza sferei din spațiul caracteristicilor care cuprinde toate exemplele de antrenare

- Teorema spune că va converge către o soluție, nu este necesar să fie o soluție bună (SVM oferă soluția de margine maximă). Dacă mulțimea de antrenare E nu este liniar separabilă algoritmul nu va converge către o soluție.

Perspectiva geometrică



Perspectiva geometrică



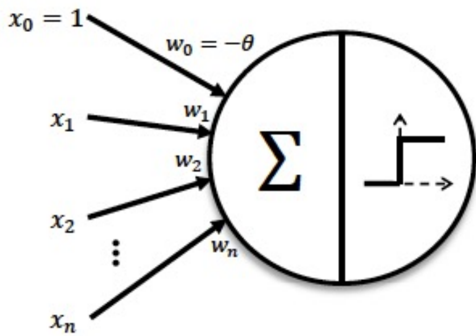
Liniar separabile **Liniar separabile** **Liniar neseparabile**

Eticheta 0

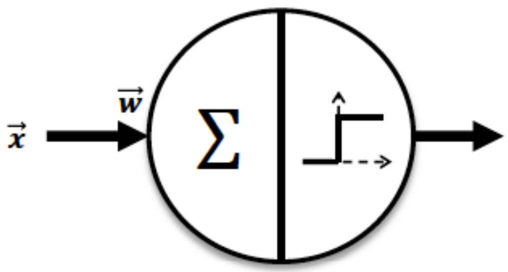
Eticheta 1

Alte reguli de învățare pentru perceptron

Notații

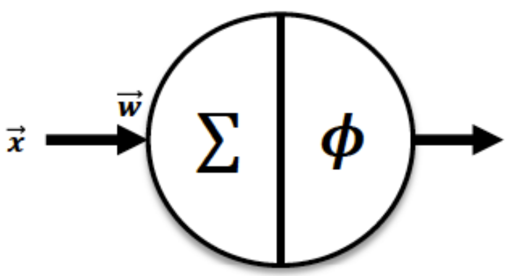


$$\hat{y} = \begin{cases} 1 & \text{if} \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



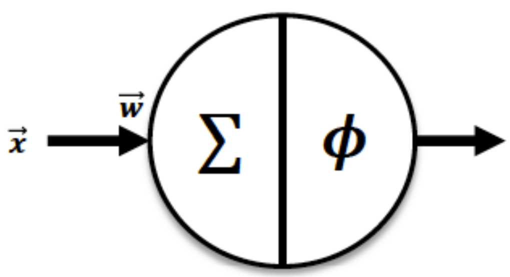
$$\hat{y} = \begin{cases} 1 & \text{if} \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Intrările x_i și ponderile w_i reprezentate sub formă de vectori



$$\hat{y} = \phi\left(\sum_{i=0}^{n} w_i x_i\right)$$

Funcția hardlim notată cu Φ și numită funcție de activare



$$\hat{y} = \phi(\langle \vec{x}, \vec{w} \rangle)$$

Suma intrărilor ponderate reprezentată ca produs scalar

Regula de învățare delta

- Ieșirea perceptronului este: $\hat{y} = \phi(\langle \vec{x}, \vec{w} \rangle)$

- Măsor diferența dintre ce voiam să obțin și ceea ce furnizează perceptronul la ieșire folosind o funcție de eroare E

- Există multe posibilități de alegere a funcției de eroare, spre exemplu pot alege E ca fiind eroarea pătratică:

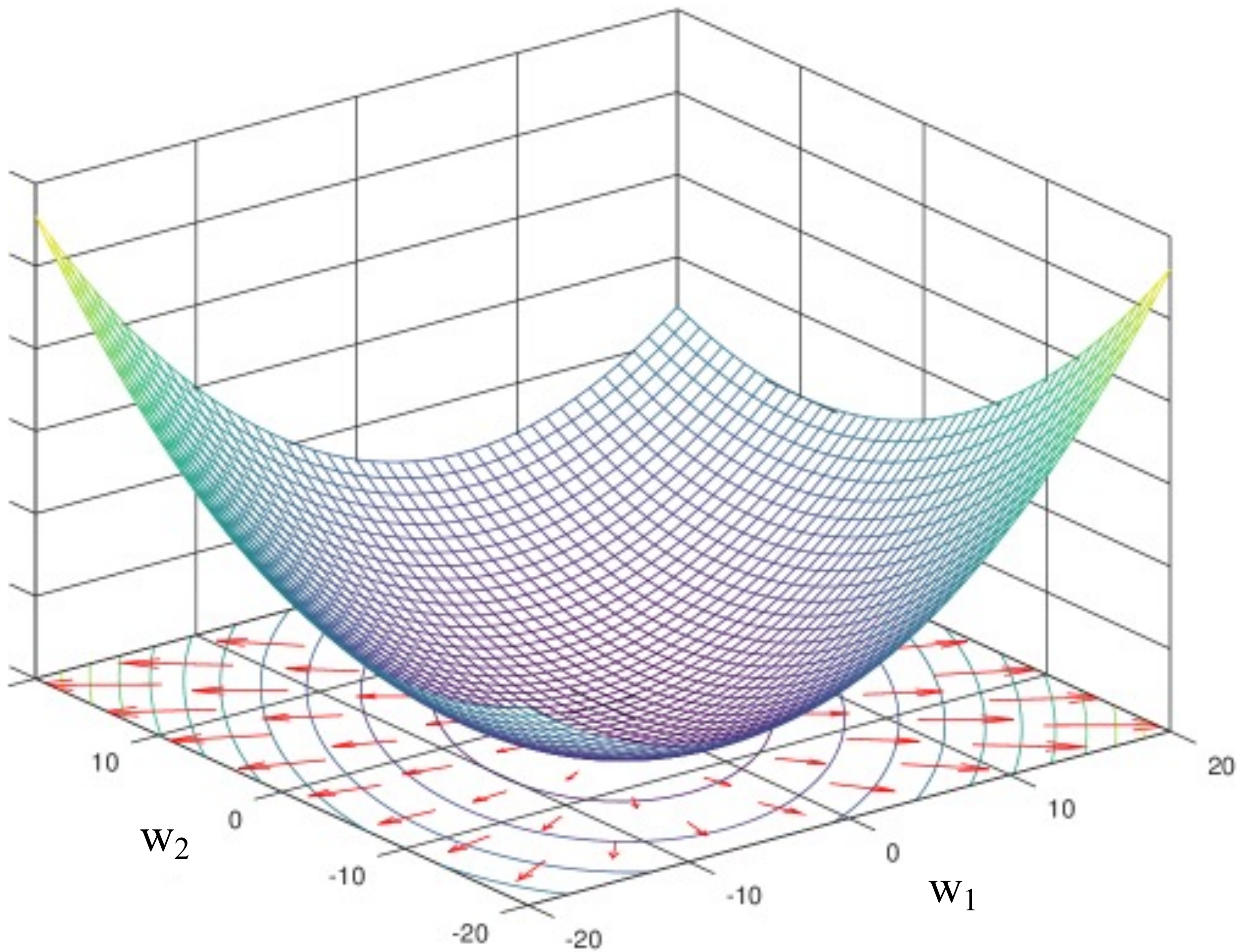
$$E(\vec{x}) = \frac{1}{2}(y - \hat{y})^2$$

Funcția de eroare E ia valori mici când cele două etichete iau valori apropiate și ia valori mari altfel

- Pot folosi metoda coborârii pe gradient (gradient descent – Cauchy 1847) pentru găsirea minimului unei funcții prin actualizarea setului de ponderi actual în direcția inversă a gradientului

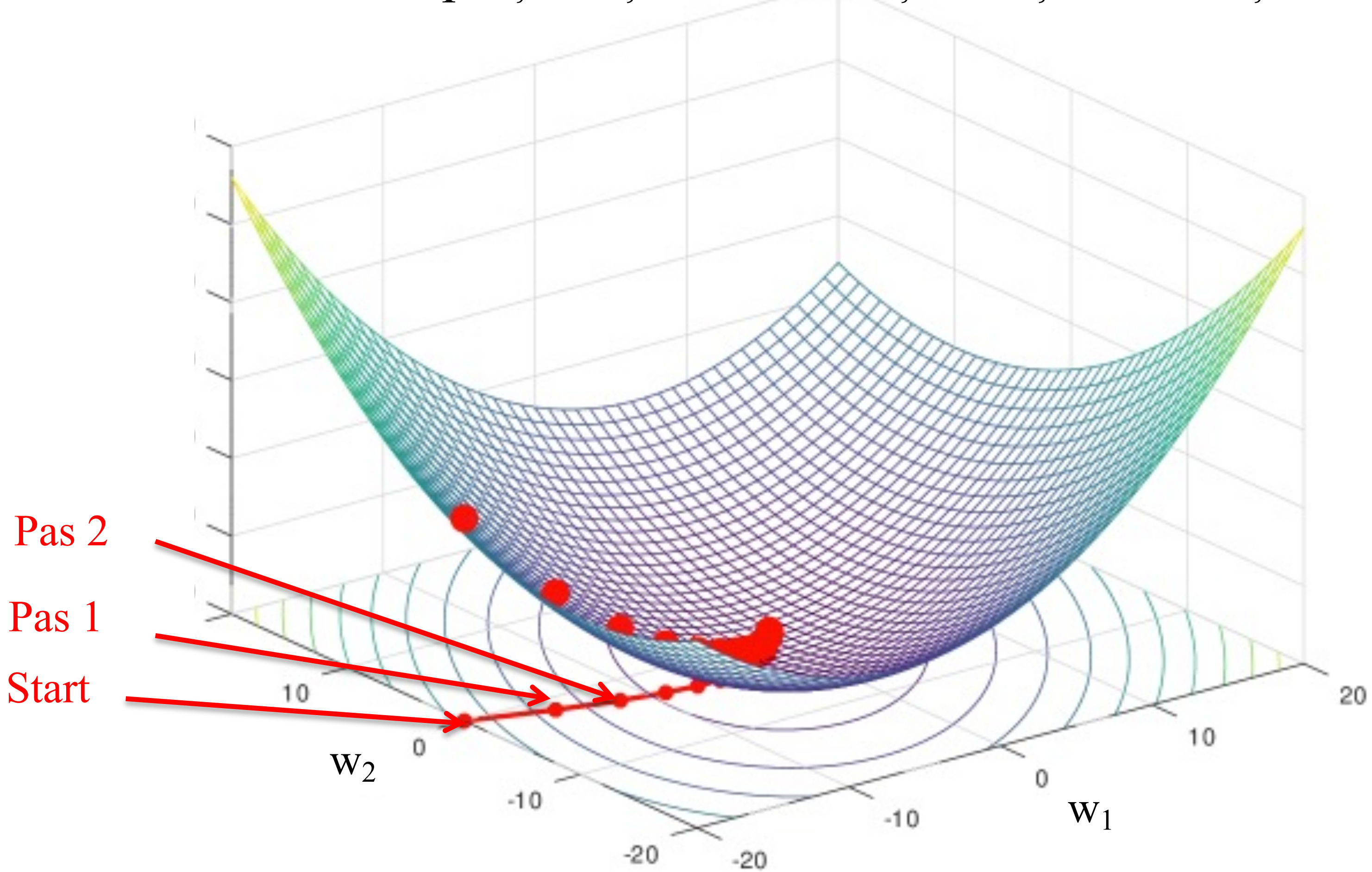
Suprafața descrisă de funcția de eroare E

Funcția de eroare pătratică E descrie un paraboloid în R^n



Algoritmul de coborâre pe gradient

Algoritm iterativ, la fiecare pas o ia în direcția inversă a gradientului pentru minimizarea valorii funcției E. Gradientul unei funcții într-un punct este vectorul cu derivate parțiale și arată direcția creșterii funcției.

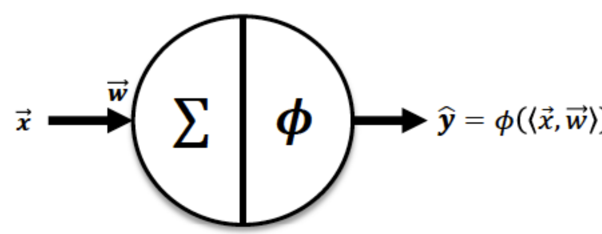


Regula de învățare delta

- Ieșirea perceptronului este: $\hat{y} = \phi(\langle \vec{x}, \vec{w} \rangle)$

- Aleg E ca fiind eroarea pătratică: $E(\vec{x}) = \frac{1}{2}(y - \hat{y})^2$

- Pot folosi metoda coborârii pe gradient (gradient descent – Cauchy 1847) pentru găsirea minimului funcției E prin actualizarea setului de ponderi actual $\vec{w}$ în direcția inversă a gradientului



$$\frac{\partial E(\vec{x})}{\partial w_j} = \frac{\partial E(\vec{x})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_j} = \frac{\partial E(\vec{x})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \langle \vec{x}, \vec{w} \rangle} \cdot \frac{\partial \langle \vec{x}, \vec{w} \rangle}{\partial w_j}$$

$$\frac{\partial E(\vec{x})}{\partial \hat{y}} = -(y - \hat{y}) \qquad \frac{\partial \hat{y}}{\partial \langle \vec{x}, \vec{w} \rangle} = \phi'(\langle \vec{x}, \vec{w} \rangle) \qquad \frac{\partial \langle \vec{x}, \vec{w} \rangle}{\partial w_j} = \frac{\partial \left(\sum_{j=0}^{n} w_j x_j \right)}{\partial w_j} = x_j$$

Regula de învățare delta

- Ieșirea perceptronului este: $\hat{y} = \phi(\langle \vec{x}, \vec{w} \rangle)$

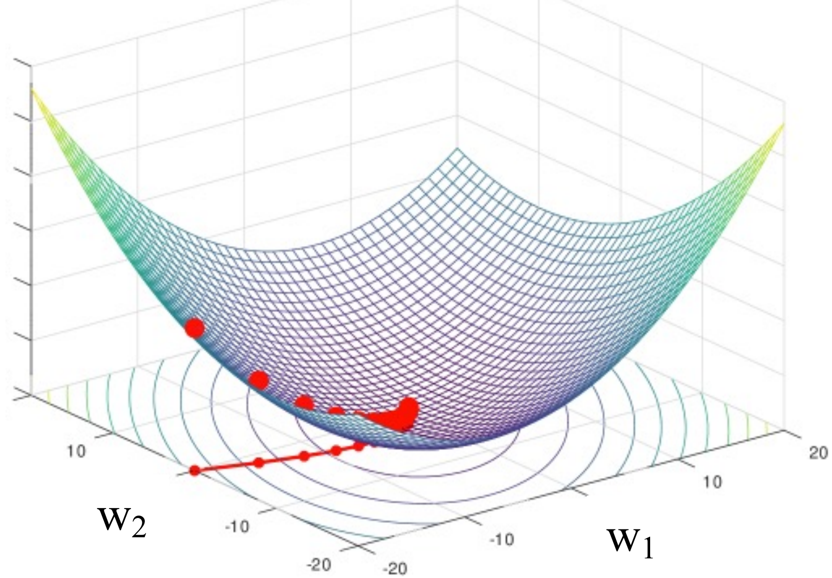
- Aleg E ca fiind eroarea pătratică: $E(\vec{x}) = \frac{1}{2}(y - \hat{y})^2$

- Pot folosi metoda coborârii pe gradient (gradient descent – Cauchy 1847) pentru găsirea minimului funcției E prin actualizarea setului de ponderi actual $\vec{w}$ în direcția inversă a gradientului

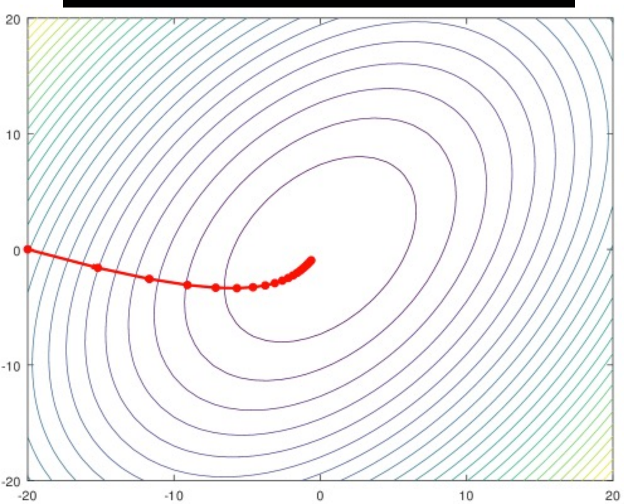
$$\frac{\partial E(\vec{x})}{\partial w_j} = \frac{\partial E(\vec{x})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_j} = \frac{\partial E(\vec{x})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \langle \vec{x}, \vec{w} \rangle} \cdot \frac{\partial \langle \vec{x}, \vec{w} \rangle}{\partial w_j} = -(y - \hat{y}) \cdot \phi'(\langle \vec{x}, \vec{w} \rangle) \cdot x_j$$

Regula de învățare delta $\Delta w_j = \underset{\substack{\uparrow \\ \text{rata de învățare}}}{\eta} \cdot (y - \hat{y}) \cdot \phi'(\langle \vec{x}, \vec{w} \rangle) \cdot x_j$

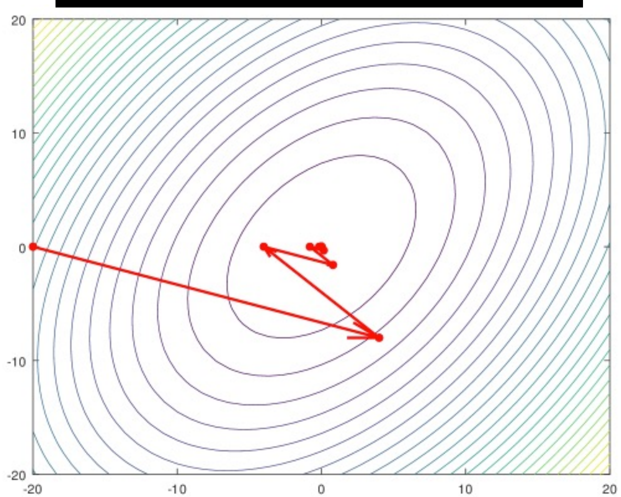
Influența ratei de învățare η



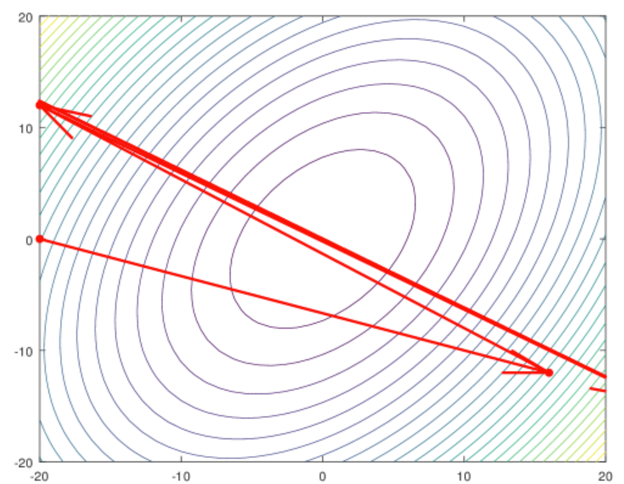
Rată de învățare prea mică
(algoritmul converge încet)



Rată de învățare bună
(viteză și convergență)



Rată de învățare mare
(algoritmul diverge)

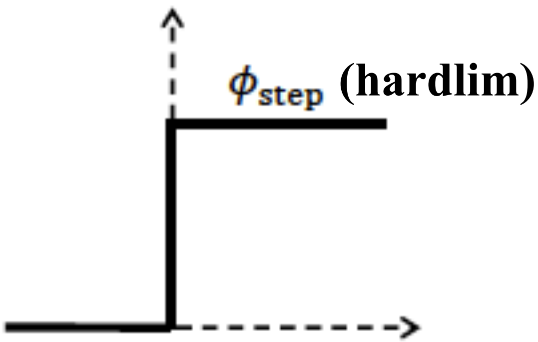


Regula de învățare delta

- Regula de învățare delta:

$w_j = w_j + \Delta w_j$, unde

$$\Delta w_j = \eta \cdot (y - \hat{y}) \cdot \phi'(\langle \vec{x}, \vec{w} \rangle) \cdot x_j$$



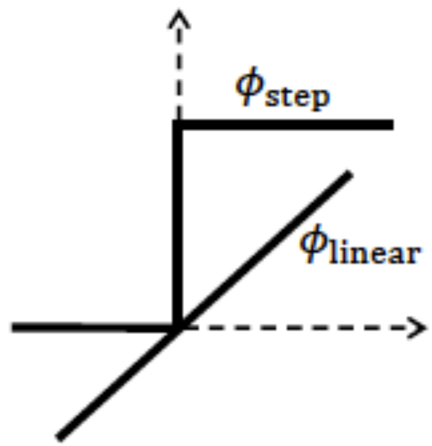
- Funcția Φ_{step} = hardlim (step function) nu este derivabilă în punctul 0 și în rest are derivata 0 (nu se va face nicio actualizare)

Regula de învățare adaline delta

- Regula de învățare delta:

$w_j = w_j + \Delta w_j$, unde

$$\Delta w_j = \eta \cdot (y - \hat{y}) \cdot \phi'(\langle \vec{x}, \vec{w} \rangle) \cdot x_j$$



- ADALINE (ADAptive Linear NEuron) este o variantă a Perceptronului propusă de Bernard Widrow în 1960, care folosește o funcție liniară $\Phi_{linear}(x) = x$ de activare la antrenare și una hardlim (step) la testare. Derivata $\Phi_{linear}'(x) = 1$.
- Obținem regula de învățare adaline delta:

$w_j = w_j + \Delta w_j$, unde $\Delta w_j = \eta \cdot (y - \hat{y}) \cdot x_j$

Algoritmul de învățare al perceptronului

```
def perceptron(X, y, n_epochs, η):
    m, n = X.shape # number of samples, number of inputs
    for j in range(n):
        w_j = 0
    for epoch in range(n_epochs): # an "epoch" is a run through all training data.
        for i in range(m):        # a "training step" is one   update of the weights.
            ŷ^(i) = unit_step_function(Σ_{j=0}^{n} w_j x_j^(i))
            for j in range(n):
                w_j += η(y^(i) − ŷ^(i)) · x_j^(i)
```

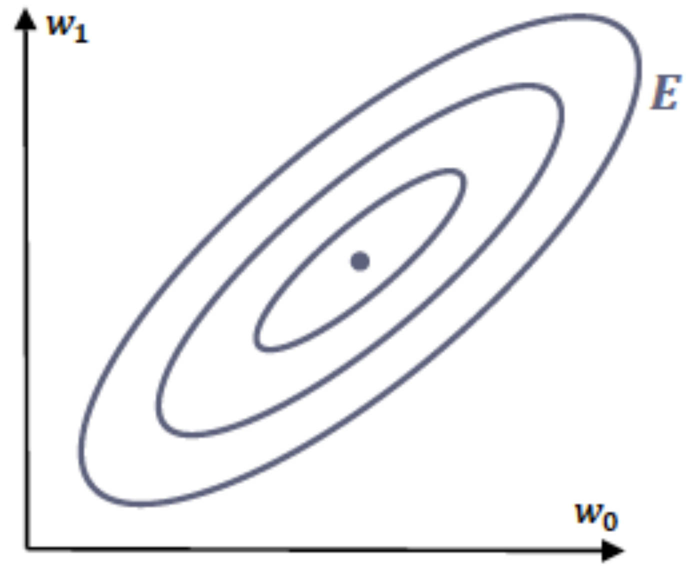
$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } \sum_{j=0}^{n} w_j x_j^{(i)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\boxed{w_j \mathrel{+}= \eta\left(y^{(i)} - \hat{y}^{(i)}\right) \cdot x_j^{(i)}}$$

Similar cu regula de învățare adaline delta $w_j = w_j + \Delta w_j$, unde $\Delta w_j = \eta \cdot (y - \hat{y}) \cdot x_j$

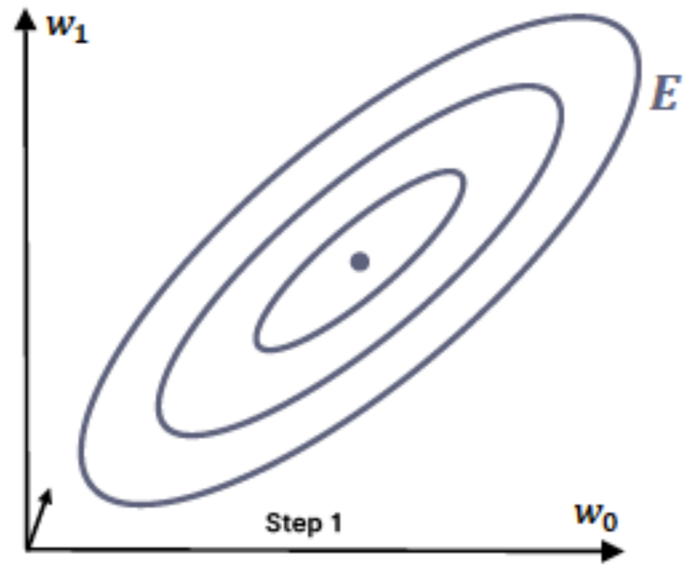
Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



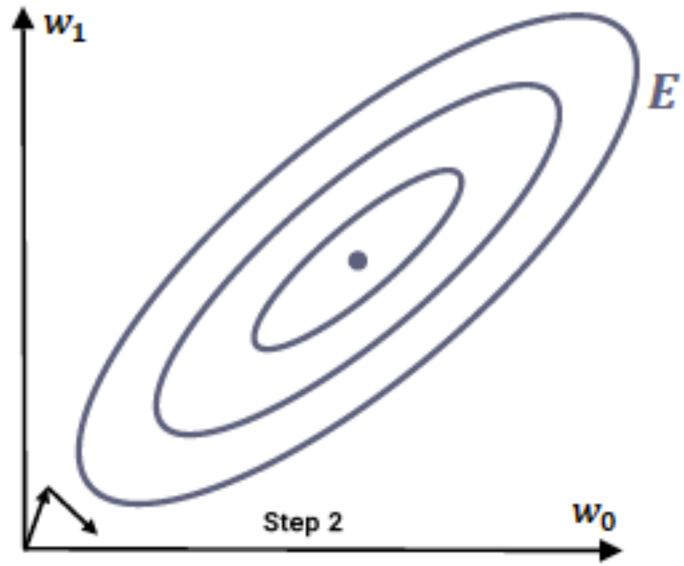
Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



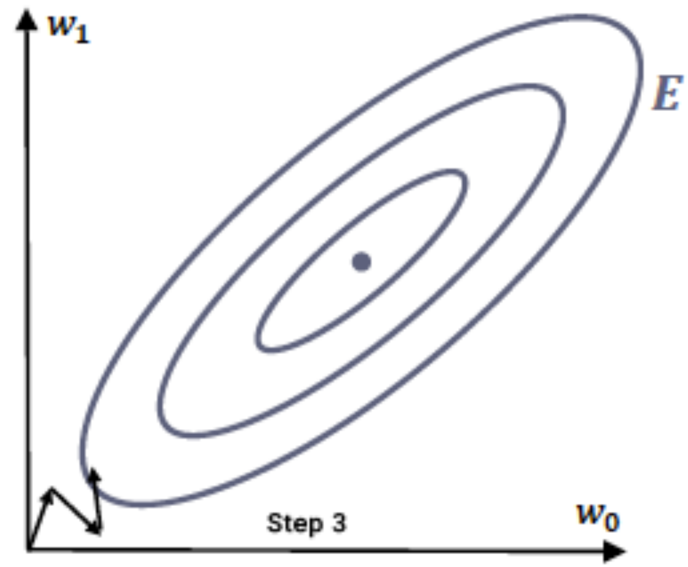
Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



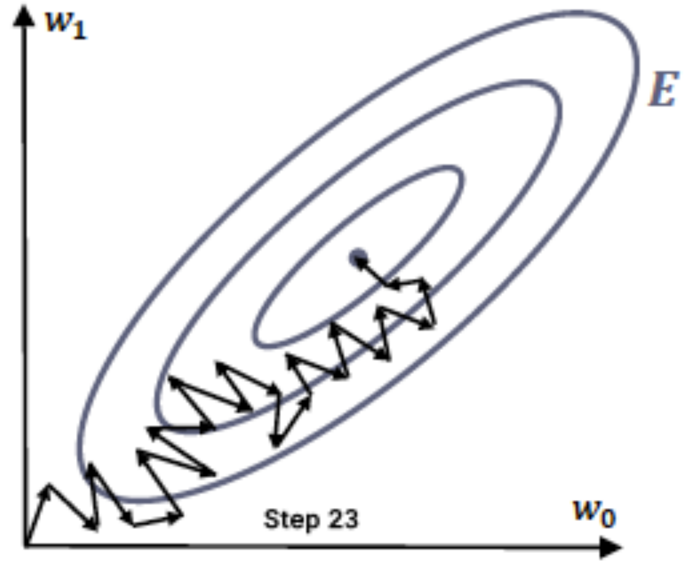
Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



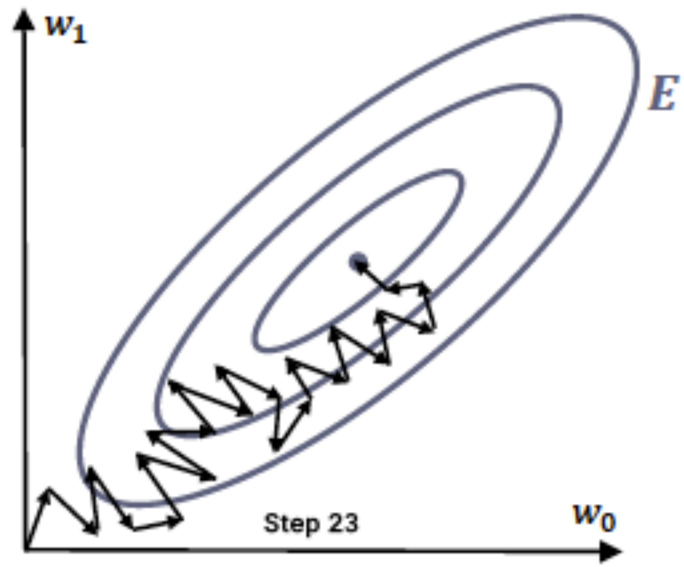
Învățarea de tip batch

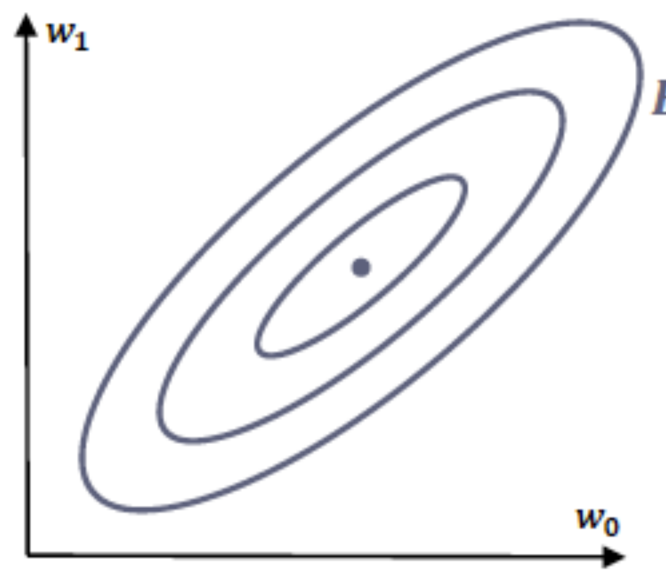
- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare

- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

$$\Delta w_j = \frac{\eta}{b} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)}\right) \cdot x_j^{(i+i_b)}$$

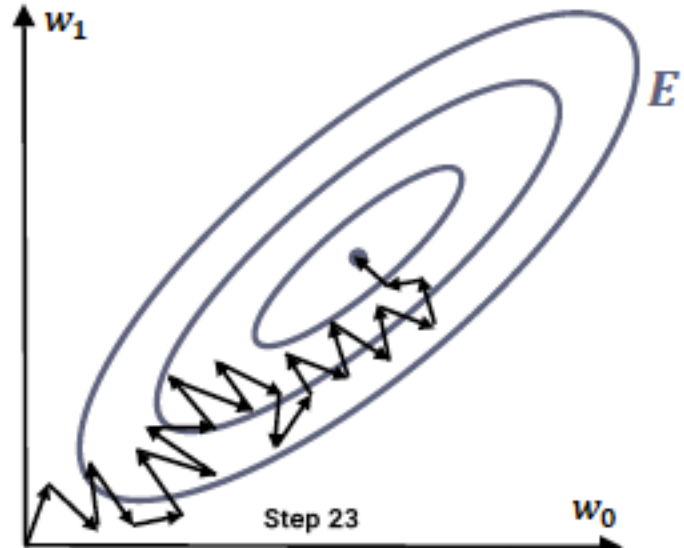
b – numărul de exemple în batch





Învățarea de tip batch

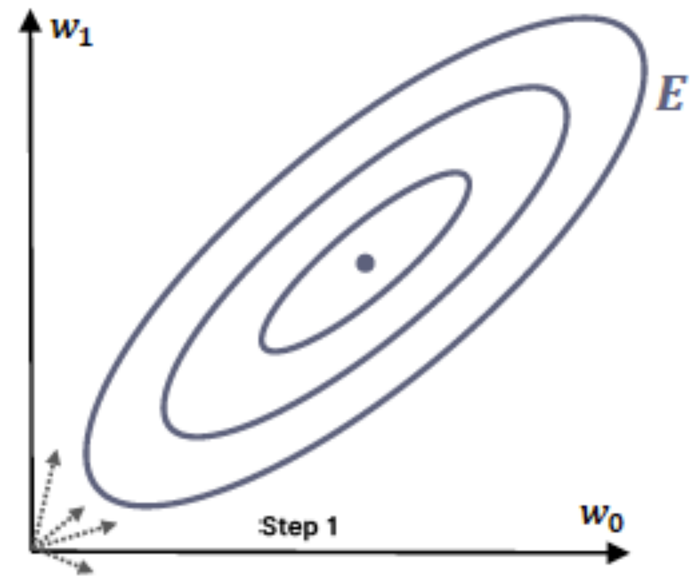
- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare



- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

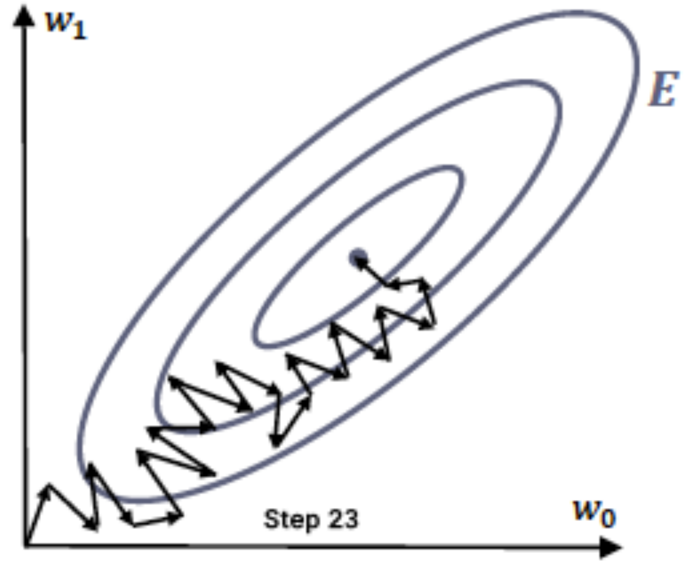
$$\Delta w_j = \frac{\eta}{\mathrm{b}} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)} \right) \cdot x_j^{(i+i_b)}$$

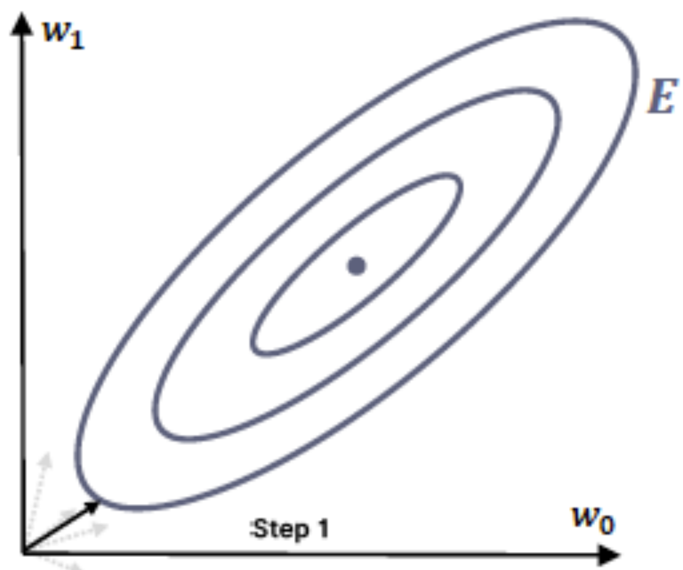
b – numărul de exemple în batch



Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare





- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

$$\Delta w_j = \frac{\eta}{b} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)} \right) \cdot x_j^{(i+i_b)}$$

b – numărul de exemple în batch

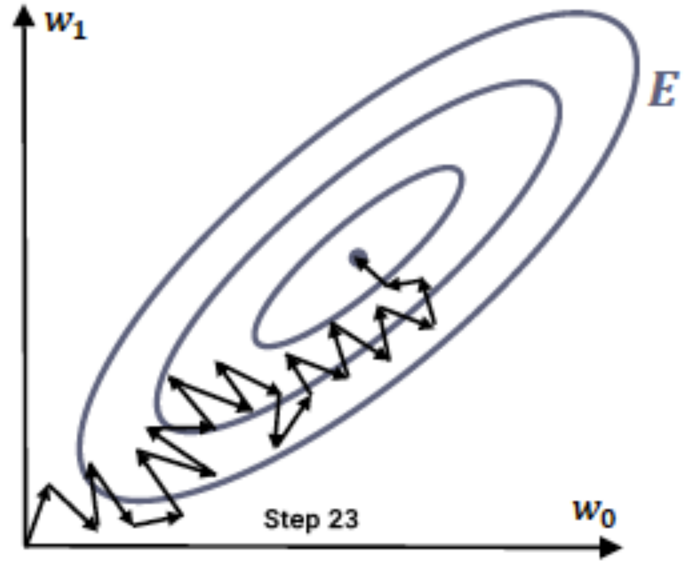
Învățarea de tip batch

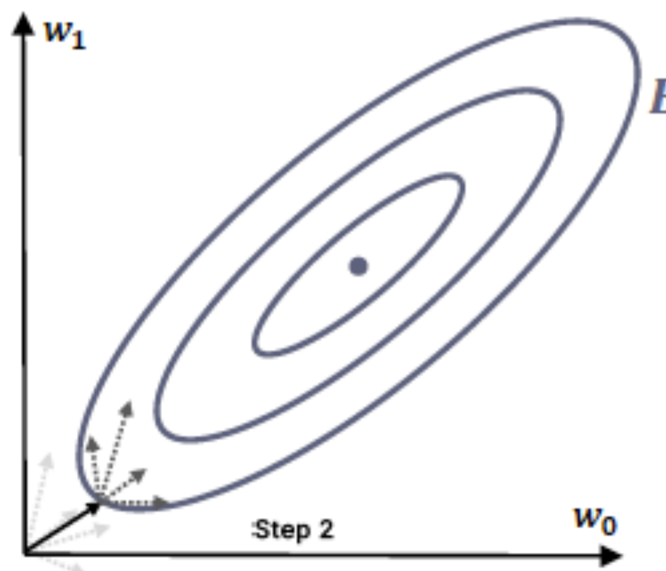
- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare

- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

$$\Delta w_j = \frac{\eta}{b} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)} \right) \cdot x_j^{(i+i_b)}$$

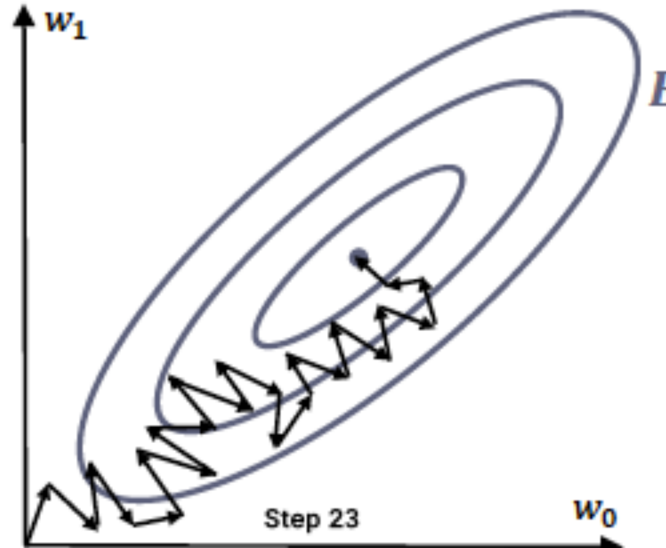
b – numărul de exemple în batch

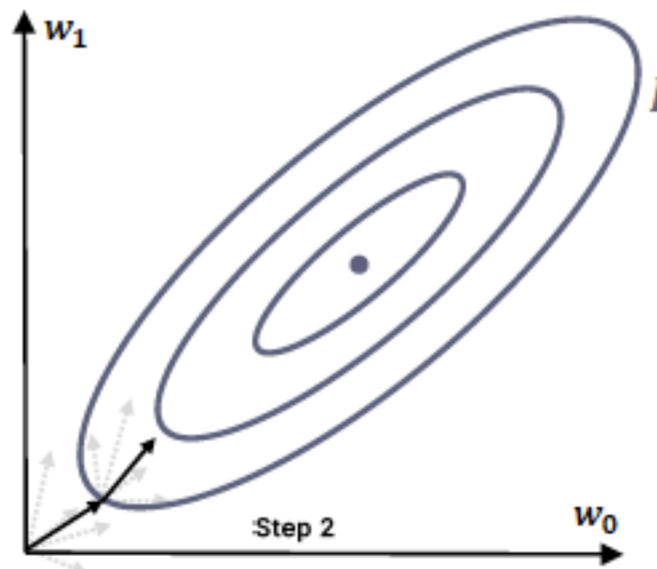




Învățarea de tip batch

- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare





- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

$$\Delta w_j = \frac{\eta}{b} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)} \right) \cdot x_j^{(i+i_b)}$$

b – numărul de exemple în batch

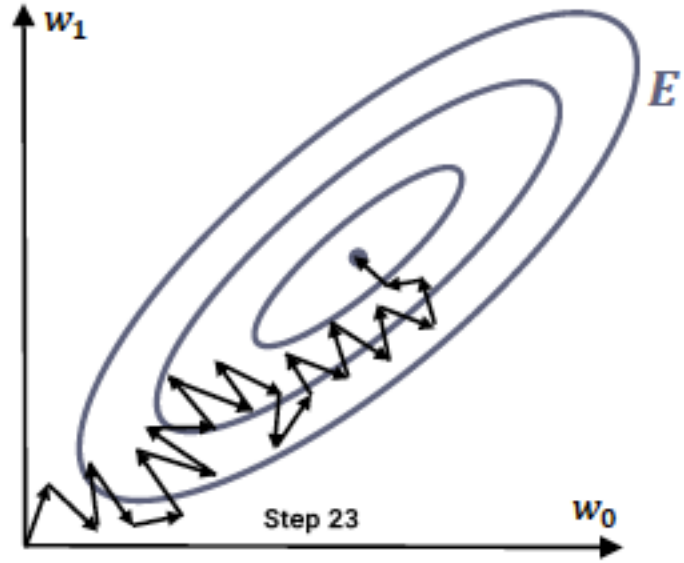
Învățarea de tip batch

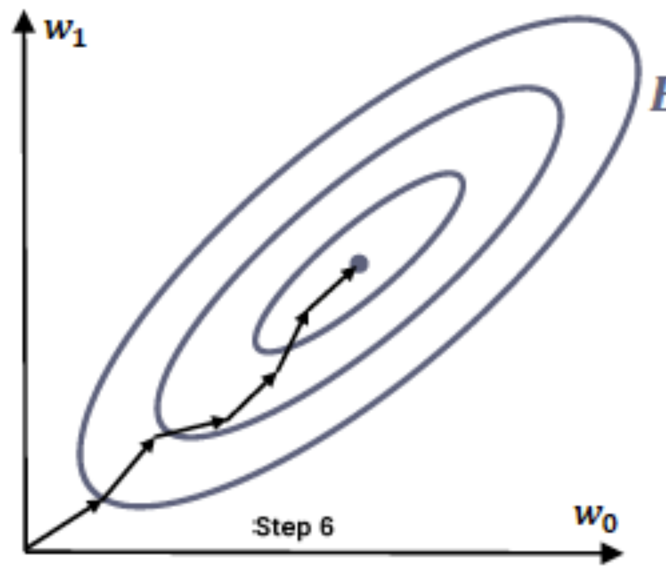
- Vrem să găsim un set de ponderi $\vec{w}$ care să minimizeze eroarea pe toate exemplele de antrenare
- În algoritmii precedenți de învățare modific ponderile după fiecare exemplu misclasificat
- Actualizarea după fiecare exemplu misclasificat în direcția dată de exemplul curent s-ar putea să nu fie cea mai bună idee raportat la întreaga mulțime de antrenare

- Învățarea de tip batch înseamnă actualizarea ponderilor în funcție de mai multe exemple de antrenare pe baza mediei gradienților:

$$\Delta w_j = \frac{\eta}{b} \sum_{i_b=0}^{b-1} \left(y^{(i+i_b)} - \hat{y}^{(i+i_b)} \right) \cdot x_j^{(i+i_b)}$$

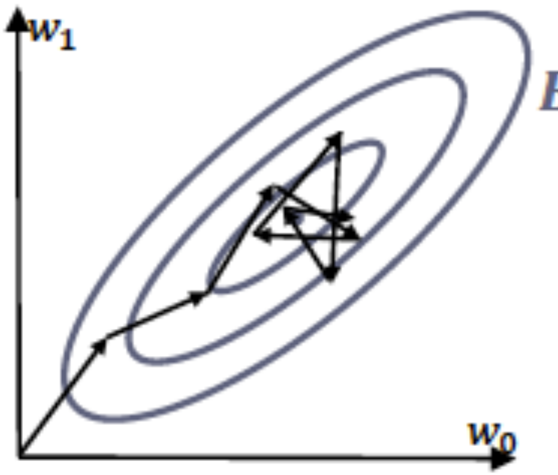
b – numărul de exemple în batch



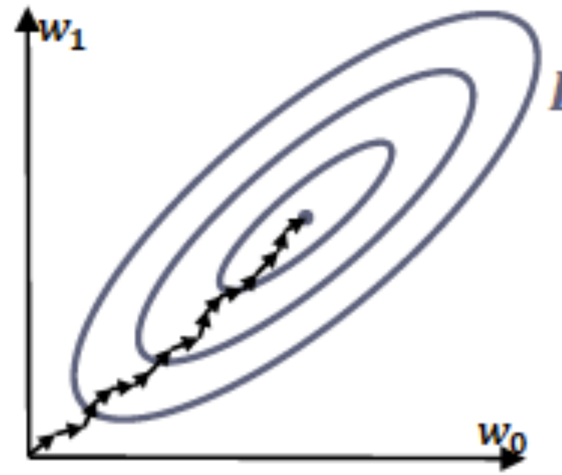


Influența ratei de învățare

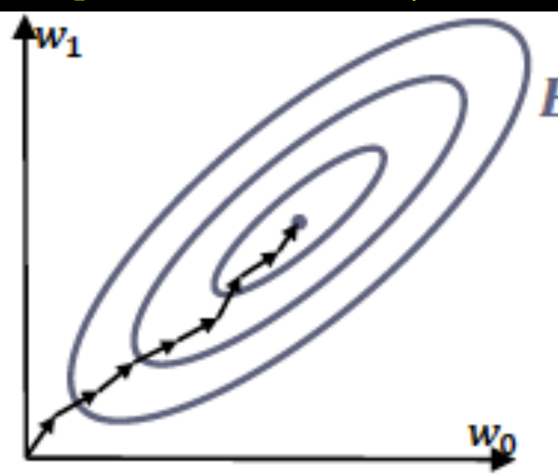
Rată de învățare prea mare
(este posibil ca algoritmul să nu conveargă)



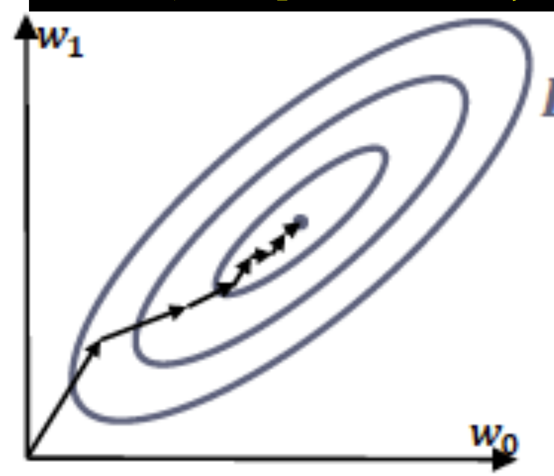
Rată de învățare prea mică
(algoritmul converge foarte încet)



Rată de învățare bună
(compromis între viteză și convergență)



Rată de învățare descrescătoare
(în timp rata de învățare scade)



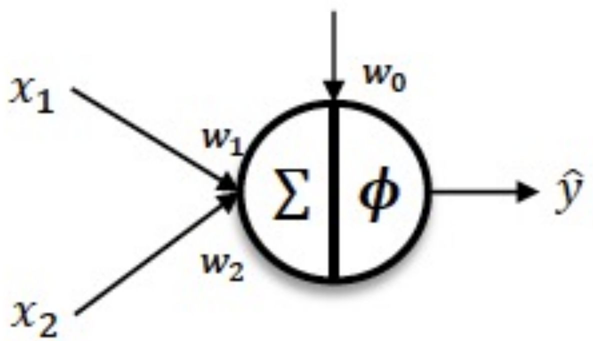
Rețele feedforward multistrat de perceptroni
(Multilayer perceptrons)

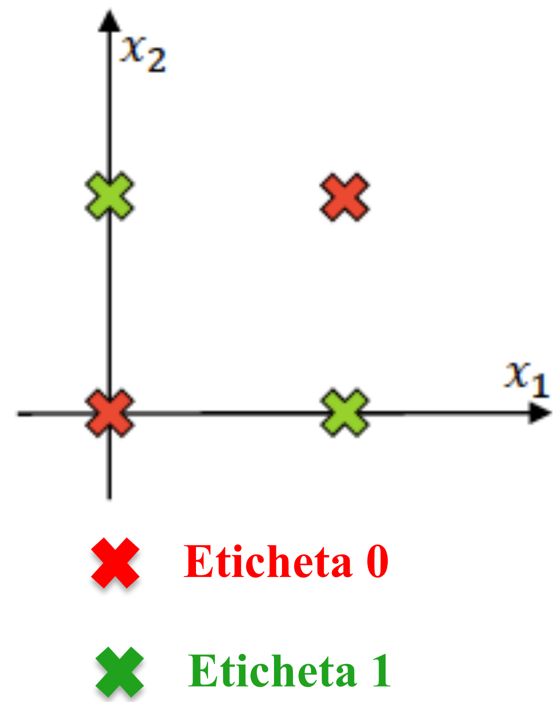
Funcția XOR

- Un singur perceptron nu poate învăța funcția XOR întrucât nu este liniar separabilă

XOR

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



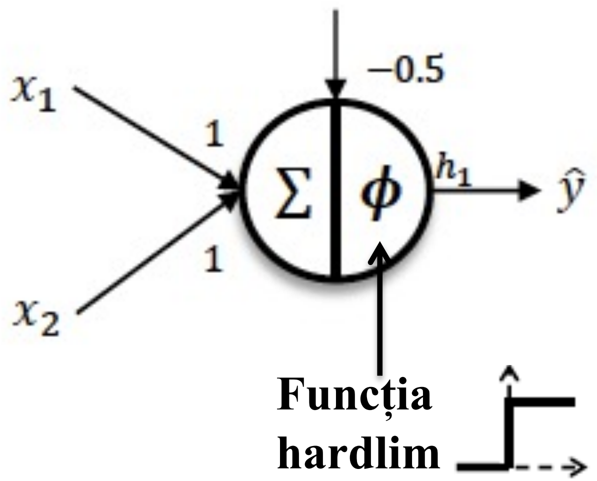


Funcția XOR

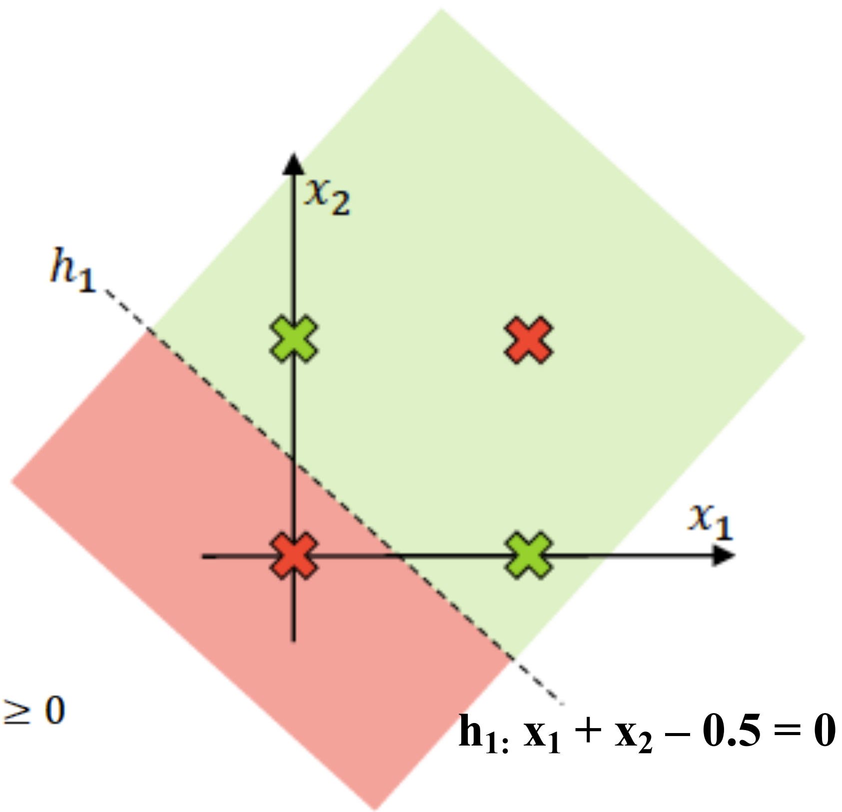
- Un singur perceptron nu poate învăța funcția XOR întrucât nu este liniar separabilă
- Putem depăși această limitare combinând ieșirile mai multor perceptroni

XOR

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



$$\hat{y} = \begin{cases} 1 & \text{if} \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

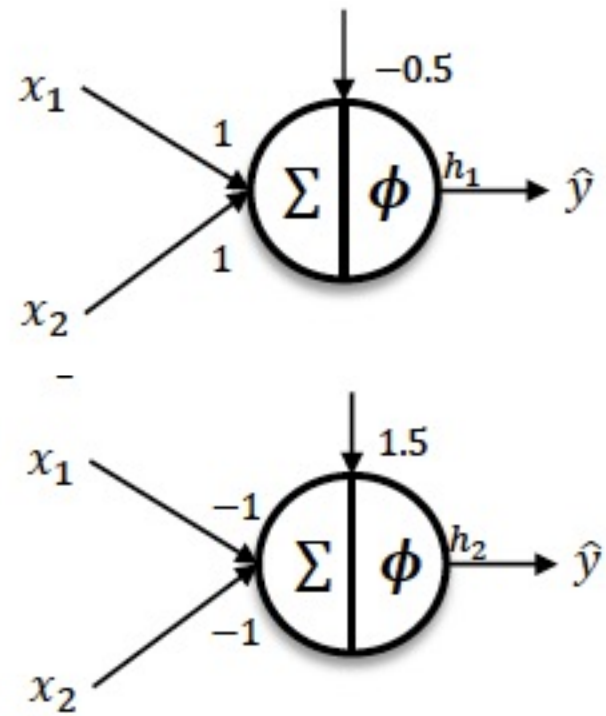


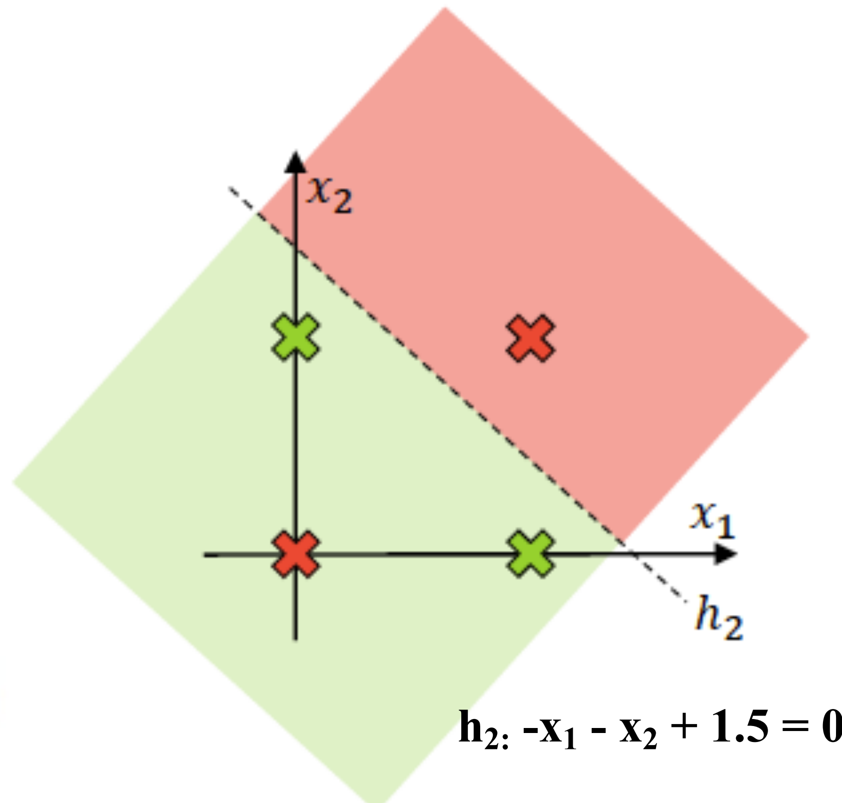
Funcția XOR

- Un singur perceptron nu poate învăța funcția XOR întrucât nu este liniar separabilă
- Putem depăși această limitare combinând ieșirile mai multor perceptroni

XOR

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



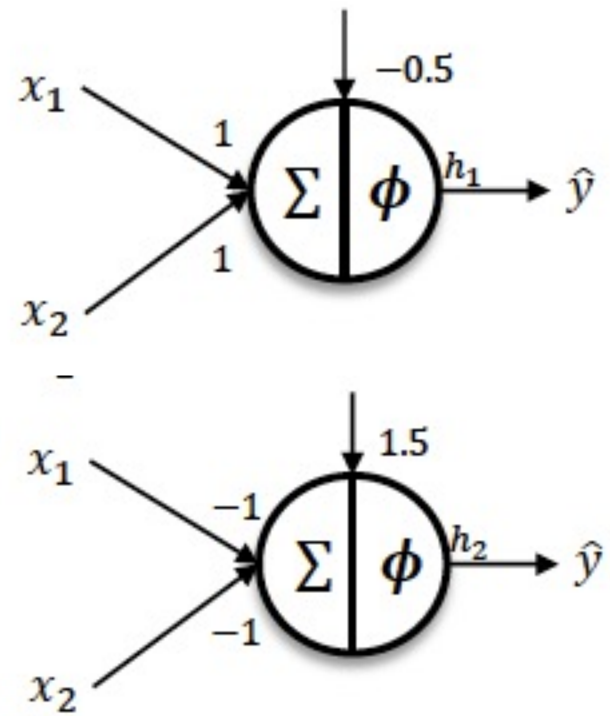


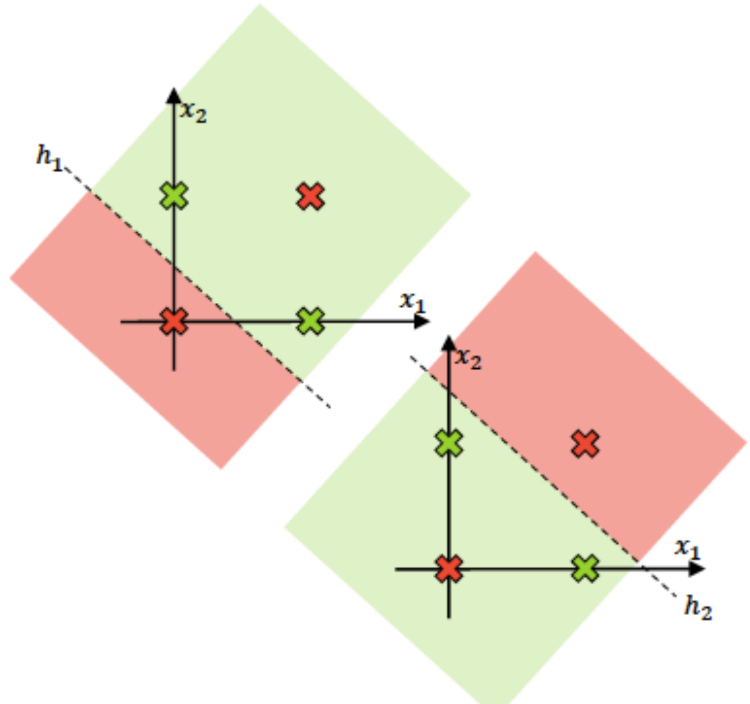
Funcția XOR

- Un singur perceptron nu poate învăța funcția XOR întrucât nu este liniar separabilă
- Putem depăși această limitare combinând ieșirile mai multor perceptroni

XOR

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



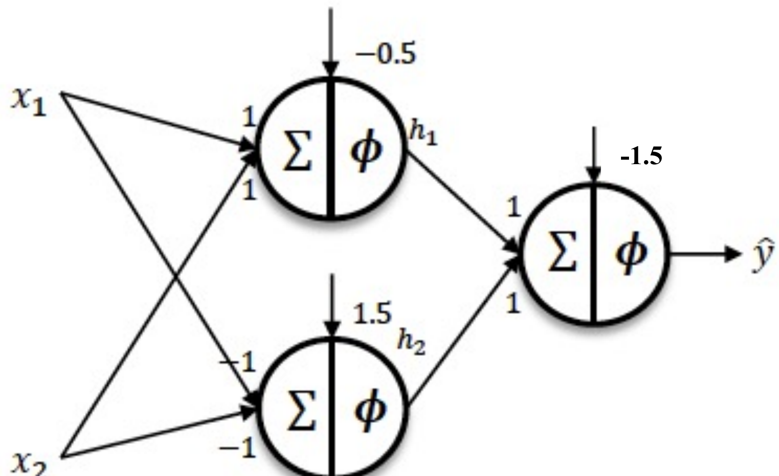


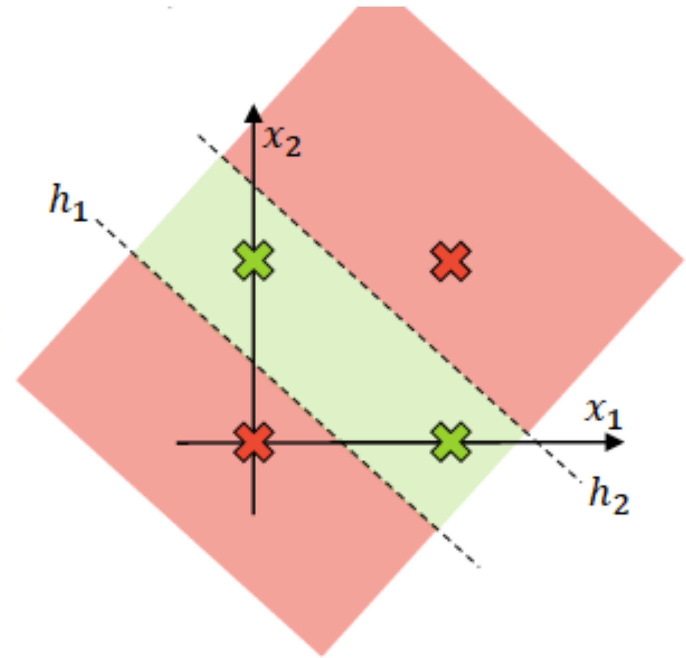
Funcția XOR

- Un singur perceptron nu poate învăța funcția XOR întrucât nu este liniar separabilă
- Putem depăși această limitare combinând ieșirile mai multor perceptroni formând o rețea de perceptroni

XOR

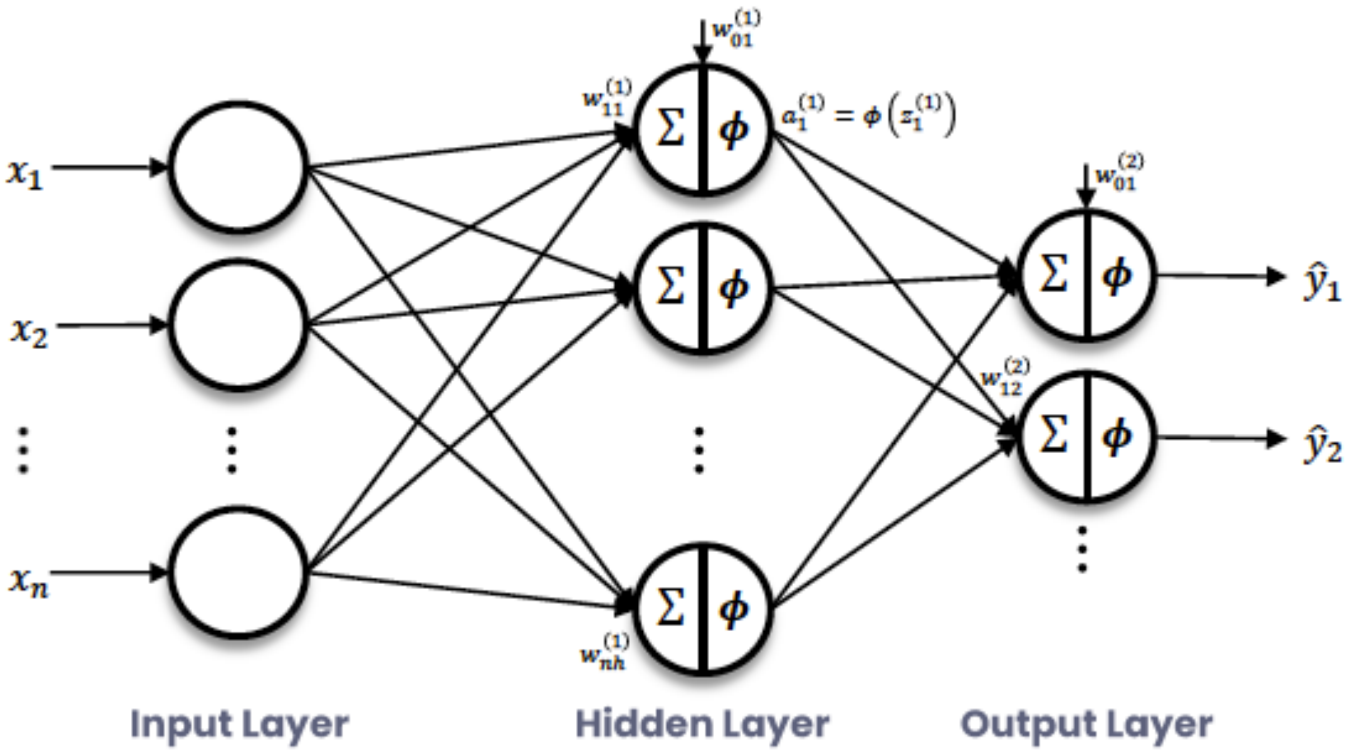
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





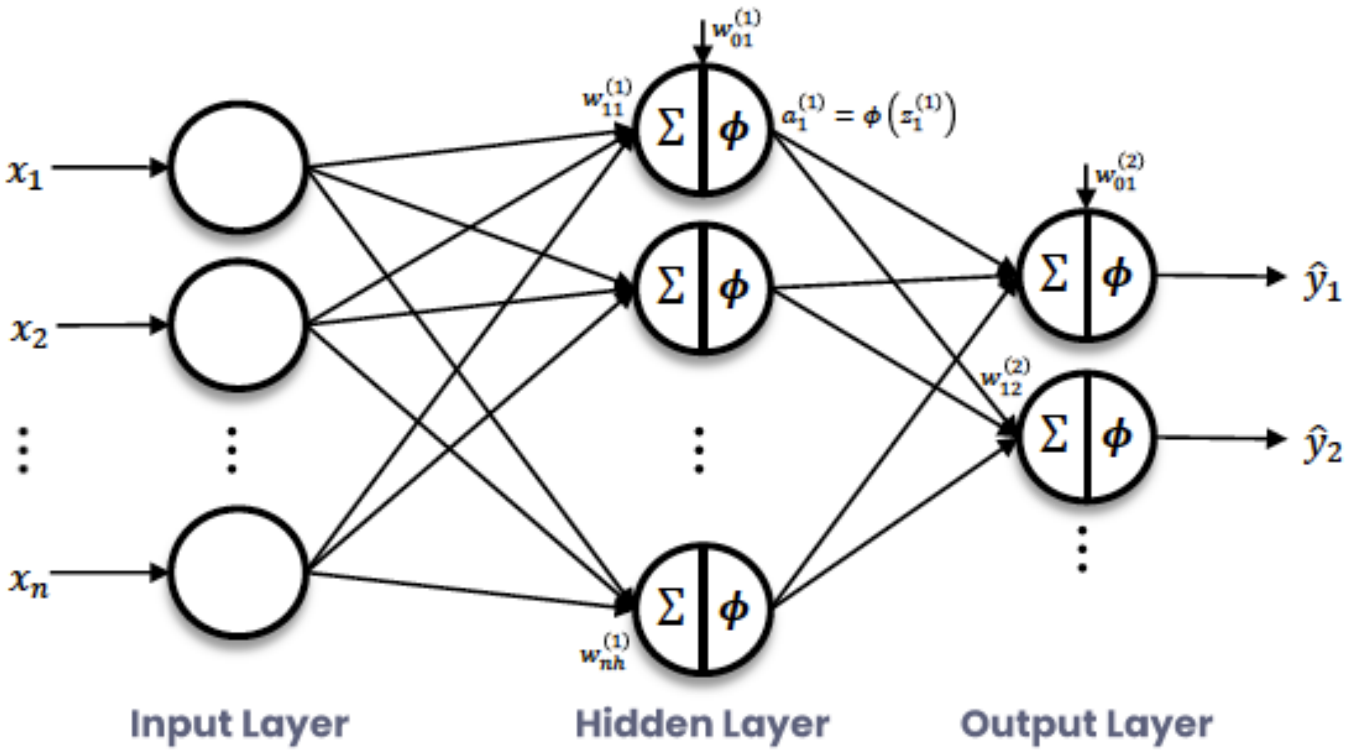
Rețele feedforward multistrat de neuroni

- O rețea feedforward multistrat de neuroni (perceptroni) este o rețea de neuroni grupați pe straturi (layere), in care propagarea informației se realizează numai dinspre intrare spre ieșire (de la stânga la dreapta). Rețeaua are un strat de intrare (input layer), unul sau mai multe straturi ascunse (hidden layers) și un strat de ieșire (output layer).



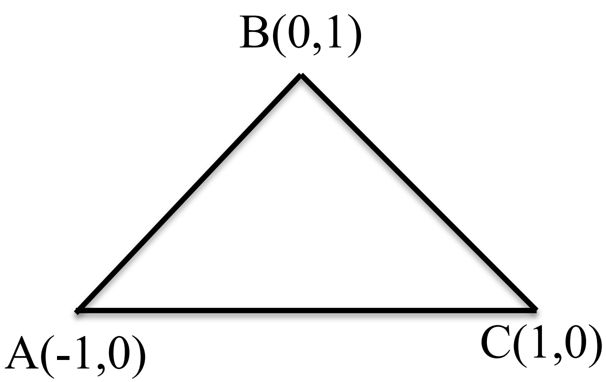
Rețele feedforward multistrat de neuroni

- Toți neuronii din rețea, cu excepția celor din stratul de intrare aplică o funcție de activare sumei ponderate ale intrărilor.
- Fiecare pereche de neuroni din două straturi consecutive are o pondere asociată



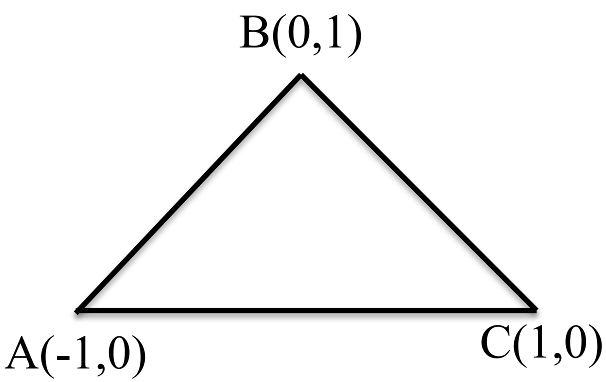
Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



AB: $x_1 - x_2 + 1 = 0$

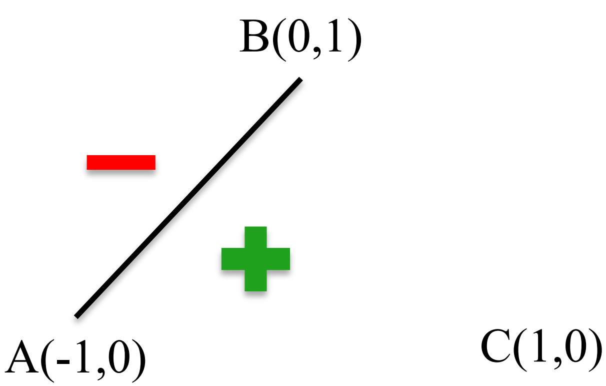
BC: $x_1 + x_2 - 1 = 0$

AC: $x_2 = 0$

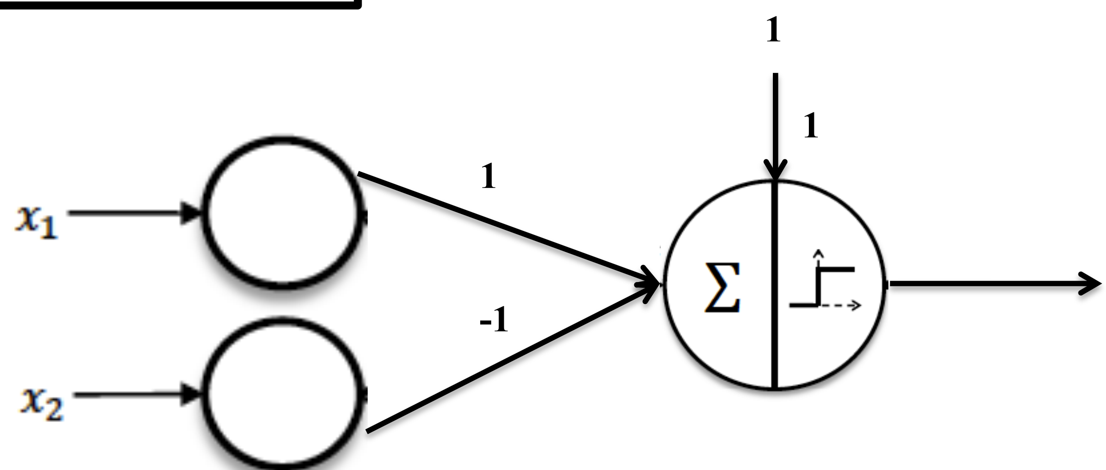
Construiesc o rețea cu 3 perceptroni pe primul strat care implementează fiecare ecuația unei drepte și cu coeficienții setați astfel încât punctele din interiorul și pe frontiera triunghiului se află în semiplanul care primește valoarea 1 (în cazul funcției de activare hardlim)

Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).

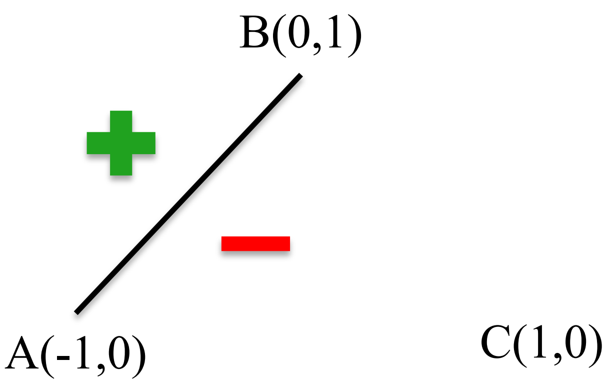


$$\boxed{AB: x_1 - x_2 + 1 = 0}$$

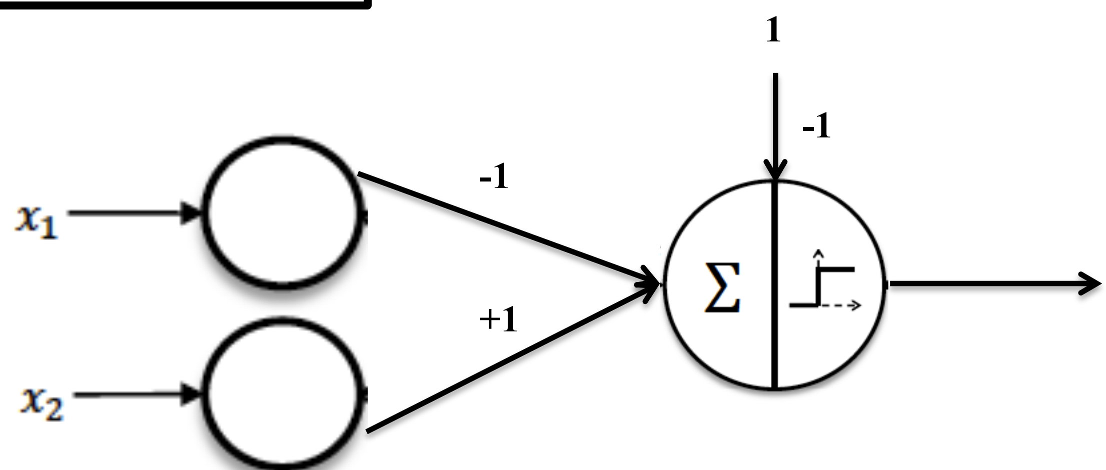


Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).

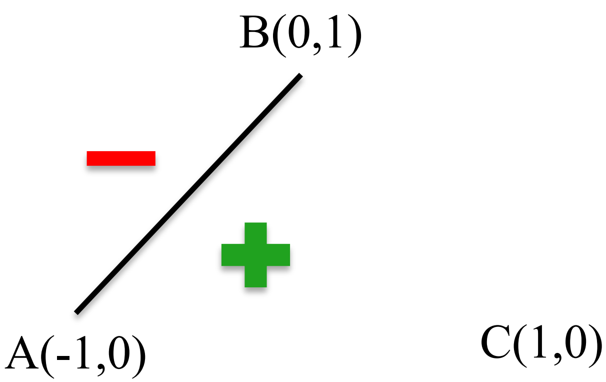


$\boxed{AB: -x_1 + x_2 - 1 = 0}$



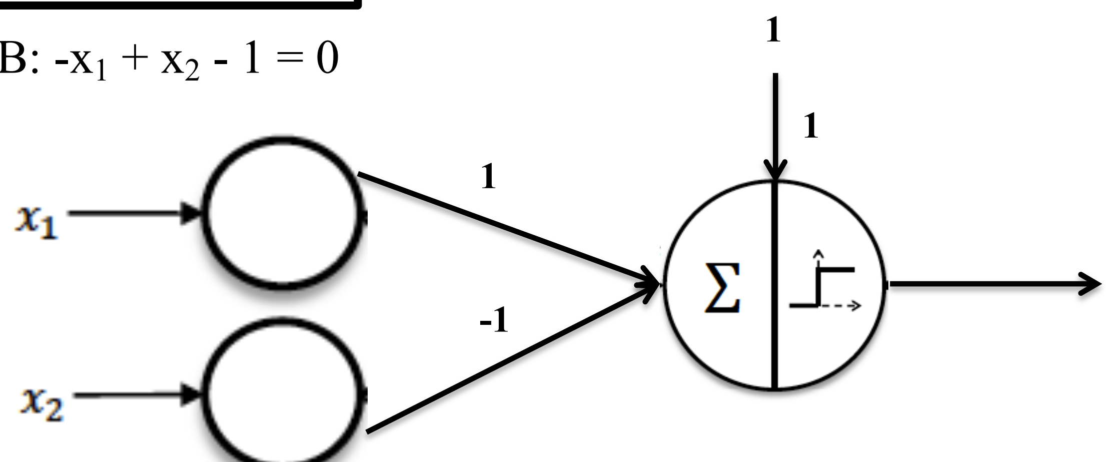
Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



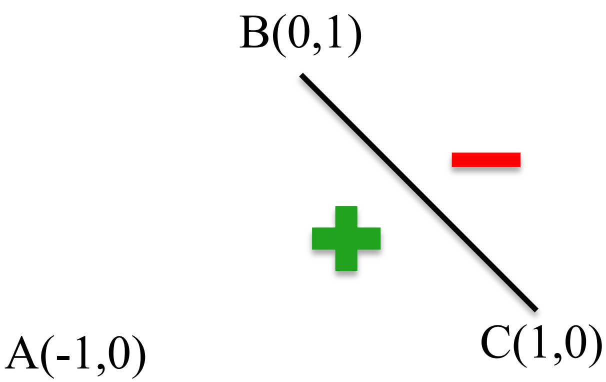
$\boxed{\text{AB: } x_1 - x_2 + 1 = 0}$

AB: $-x_1 + x_2 - 1 = 0$



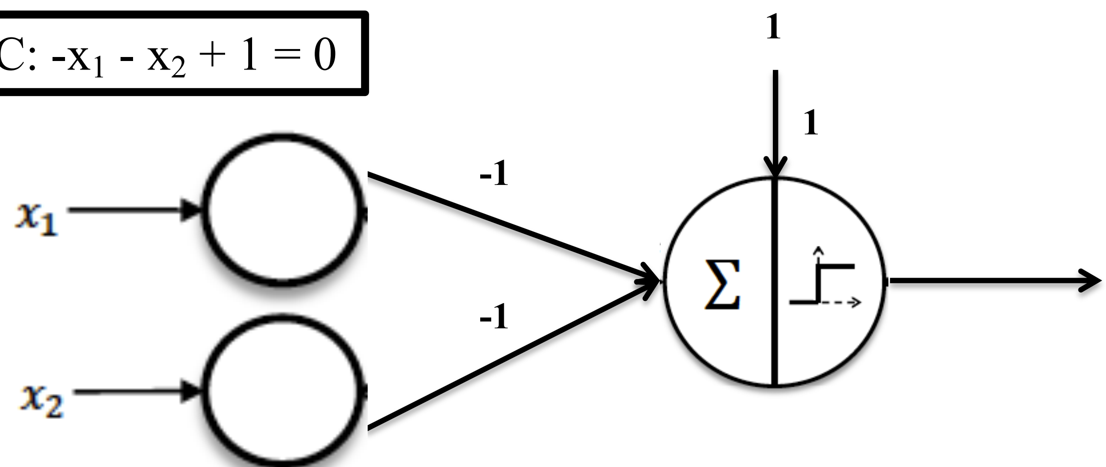
Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



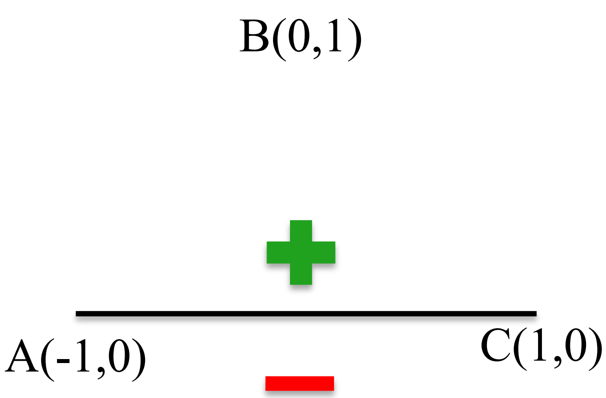
BC: $x_1 + x_2 - 1 = 0$

$\boxed{\text{BC: } -x_1 - x_2 + 1 = 0}$



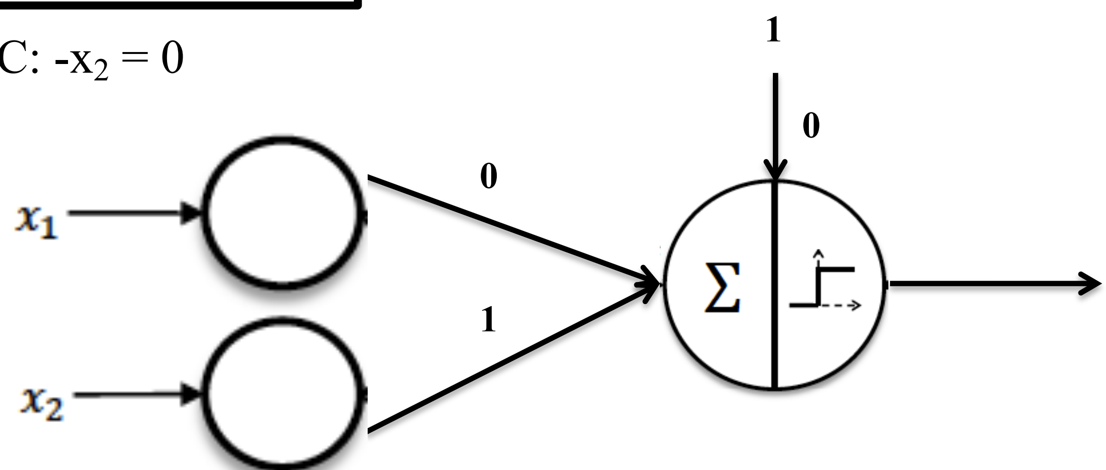
Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



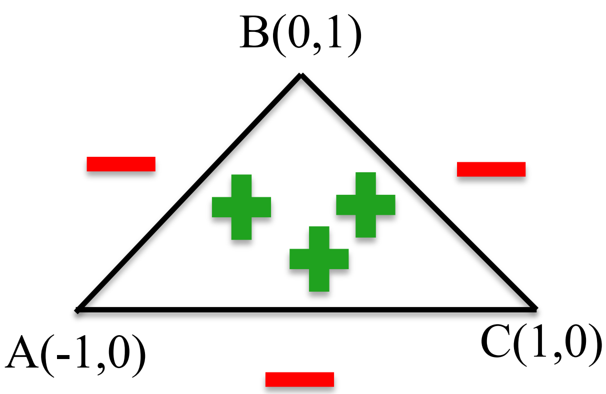
$\boxed{AC: x_2 = 0}$

$AC: -x_2 = 0$



Rețele feedforward multistrat de neuroni setate manual

Construiți o rețea care să implementeze funcția indicator a triunghiului ABC, cu vârfurile având coordonatele A(-1,0), B(0,1), C(1,0). Funcția indicator a unui triunghi ia valoarea 1 pentru punctele din triunghi (interior + frontieră) și 0 altfel (exterior).



Construiesc o rețea cu 3 perceptroni pe primul strat care implementează fiecare ecuația unei drepte și cu coeficienții setați astfel încât punctele din interiorul și pe frontiera triunghiului se află în semiplanul care primește valoarea 1 (în cazul funcției de activare hardlim)

Pe ultimul strat pun un perceptron care face AND, un punct primește eticheta 1 dacă primește 1 de la toți cei trei perceptroni.

Rețele feedforward multistrat de neuroni setate manual