

Curs 6:

Clasificarea datelor (III)

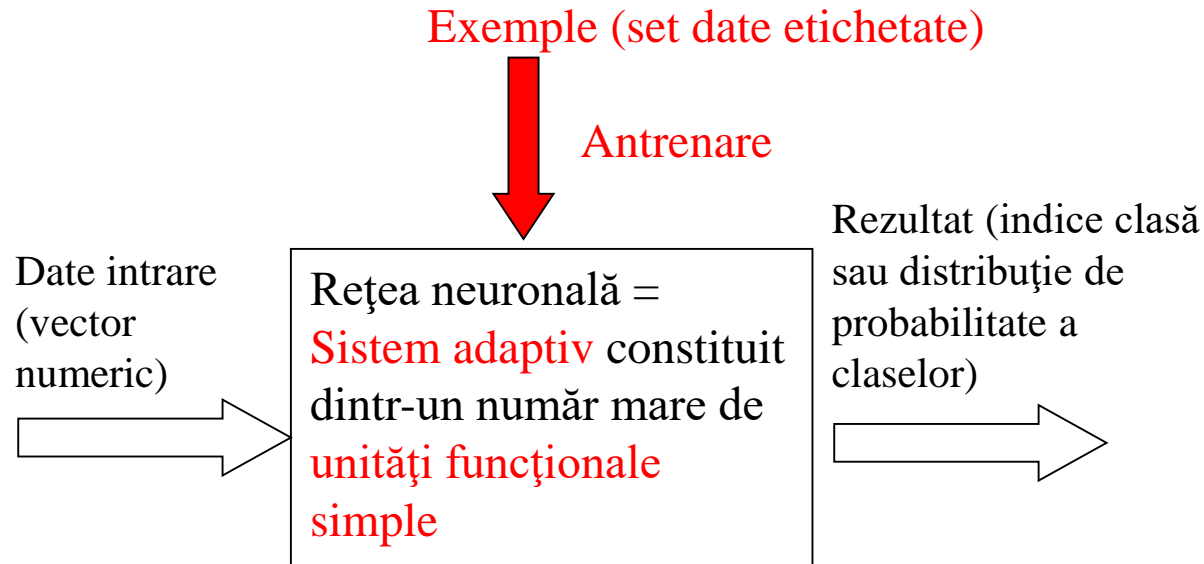
Structura

- Clasificatori bazați pe rețele neuronale
 - Rețele neuronale feedforward (Multilayer Perceptrons)
- Clasificatori bazați pe vectori suport (Support Vector Machines)

Rețele neuronale artificiale

Particularități:

- Sunt clasificatori de tip black-box = permit predicția clasei dar nu furnizează direct reguli explicite de clasificare (nu posedă modul explicativ)

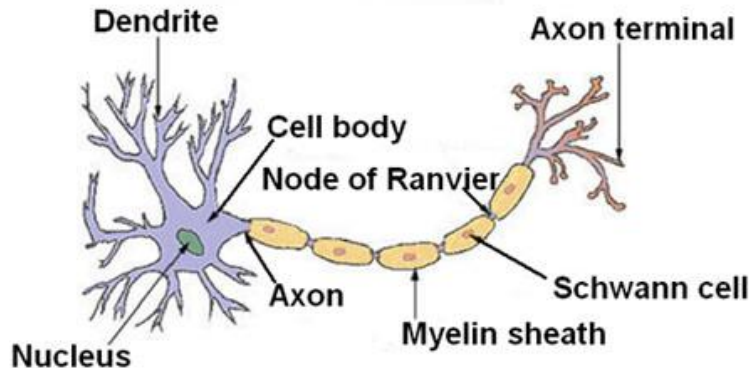


Rețele neuronale – modelul biologic

Particularități:

- Inspirate inițial de structura și funcționarea creierului = sistem de neuroni interconectați
- Creier = cca 10^{10} neuroni și 10^{14} sinapse

Structure of a Typical Neuron



Rețele neuronale artificiale

- RNA = set de neuroni artificiali (**unități functionale**) interconectați
 - Fiecare neuron primește mai multe semnale de intrare și produce un semnal de ieșire
 - RNA primește un vector de intrare (prin neuronii de intrare) și produce un vector de ieșire (prin neuronii de ieșire)
- Aspecte principale ale unei RNA:
 - **Arhitectura** = graf orientat etichetat; fiecare arc are asociată o **pondere numerică** care modelează permeabilitatea sinaptică
 - **Funcționare** = procesul prin care RNA transformă un vector de intrare într-un vector de ieșire
 - **Antrenare** = procesul prin care sunt stabilite valorile ponderilor sinaptice și ale altor parametri ai rețelei (de exemplu, praguri de activare – vezi slider-uri următoare)

Rețele neuronale artificiale

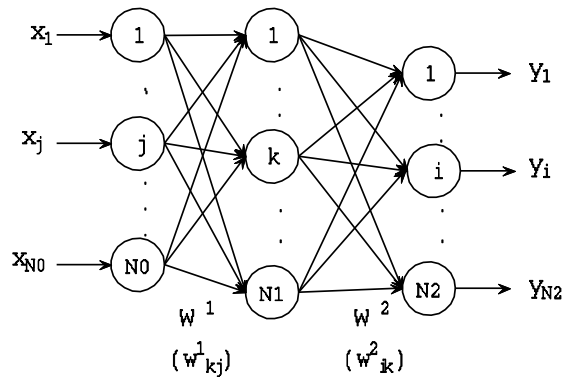
Principalele tipuri de arhitecturi:

- **Unidirecțională (Feed-forward):**

- Graful suport **nu conține cicluri** (neuronii sunt de obicei plasați pe mai multe nivele)
- Semnalul de ieșite poate fi calculat prin compunerea unor **funcții de agregare** și de **activare (transfer)**

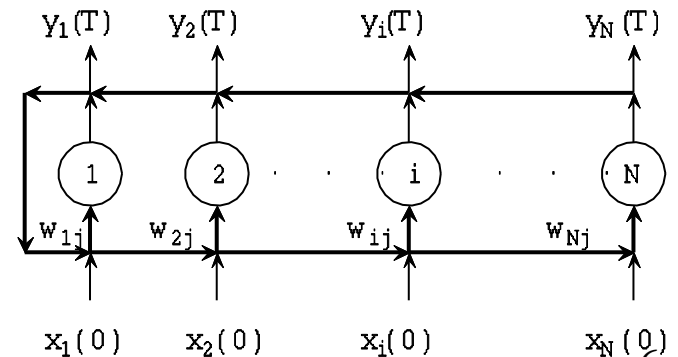
- **Recurentă (Recurrent):**

- Graful suport **conține cicluri**
- Semnalul de ieșire este calculat prin simularea unui sistem dinamic (proces iterativ)



Feed-forward (multilayer perceptron) Data Mining - Curs 6 (2020)

RNA recurentă (rețea complet interconectată)



Rețele neuronale artificiale

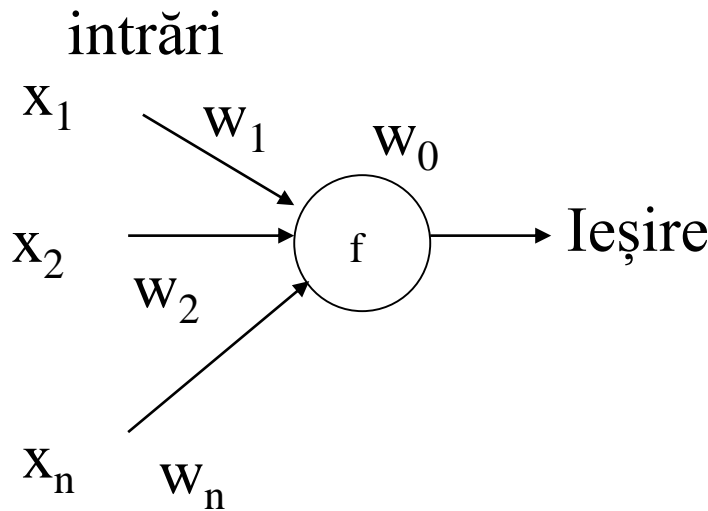
Proiectarea unei RNA:

- **Alegerea arhitecturii:** număr de nivele, număr de unități pe fiecare nivel, funcții de activare, tip interconectare
- **Antrenare:** determinarea valorilor ponderilor folosind un set de antrenare și un algoritm de învățare
- **Validare/testare:** analiza comportamentului rețelei pentru exemple care nu fac parte din setul de antrenare

Obs:

- Pt o problemă de clasificare a unor **date N-dimensionale** în **M clase** rețeaua ar trebui să aibă:
 - **N unități de intrare**
 - **M unități de ieșire**
- Modelul de clasificare este încorporat în ponderile sinaptice (ponderile asociate conexiunilor dintre neuroni)

Rețele neuronale artificiale



w_1, w_2, \dots : Ponderi numerice atașate conexiunilor

Rețea neuronală artificială = ansamblu de unități simple de prelucrare (neuroni) interconectate

Unitate funcțională: mai multe intrări, o ieșire (model computațional simplificat al neuronului)

Notății:

semnale de intrare: x_1, x_2, \dots, x_n

ponderi sinaptice: w_1, w_2, \dots, w_n
(modelează permeabilitatea sinaptică)

prag (bias): b (sau w_0)
(modelează pragul de activare al neuronului)

ieșire: y

$$y = f\left(\sum_{j=1}^n w_j x_j - w_0\right)$$

Funcție activare

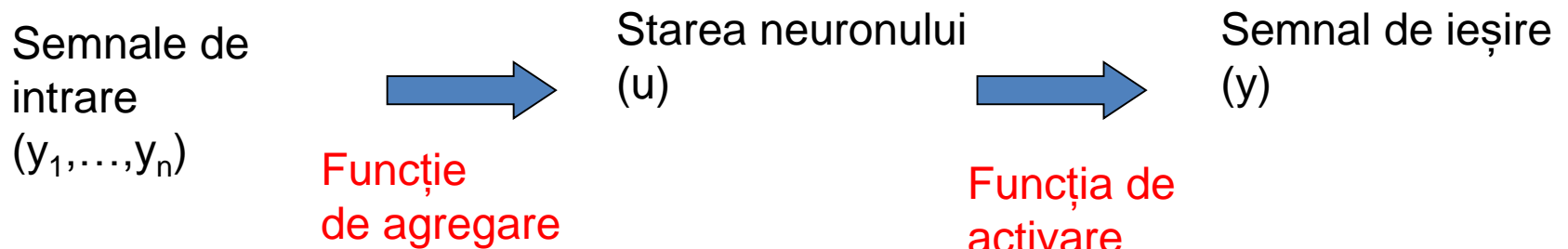
prag

Obs: Toate valorile sunt numere reale

Unități funcționale

Generarea semnalului de ieșire:

- Se “combină” semnalele de intrare utilizând ponderile sinaptice și pragul de activare
 - Valoarea obținută modelează potențialul local al neuronului
 - Combinarea semnalelor de intrare în unitate se realizează printr-o **funcție de agregare** (integrare)
- Se generează semnalul de ieșire aplicând o **funcție de activare** (transfer)
 - corespunde generării impulsurilor de-a lungul axonului



Unități funcționale

Exemple de funcții clasice de agregare

Suma ponderată

$$u = \sum_{j=1}^n w_j x_j - w_0$$

$$u = \prod_{j=1}^n x_j^{w_j}$$

Distanța euclidiană

$$u = \sqrt{\sum_{j=1}^n (w_j - x_j)^2}$$

$$u = \sum_{j=1}^n w_j x_j + \sum_{i,j=1}^n w_{ij} x_i x_j + \dots$$

Neuron multiplicativ

Conexiuni de ordin superior

Observatie: pentru varianta cu suma ponderată se poate asimila pragul cu o pondere sinaptică corespunzătoare unei intrări fictive (cu valoare -1) astfel că starea neuronului poate fi exprimată prin suma ponderată:

$$u = \sum_{j=0}^n w_j x_j$$

Unități funcționale

Exemple de funcții de activare (transfer)

$$f(u) = \text{sgn}(u) = \begin{cases} -1 & u \leq 0 \\ 1 & u > 0 \end{cases} \quad \text{signum}$$

$$f(u) = H(u) = \begin{cases} 0 & u \leq 0 \\ 1 & u > 0 \end{cases} \quad \text{Heaviside}$$

$$f(u) = \begin{cases} -1 & u < -1 \\ u & -1 \leq u \leq 1 \\ 1 & u > 1 \end{cases} \quad \text{rampă}$$

$$f(u) = u \quad \text{liniară}$$

Unități funcționale

Exemple de funcții de activare (transfer)

$$f(u) = \max\{0, u\} \quad \text{ReLU}$$

Semi-liniară (rectified linear unit - **ReLU**)

$$f(u) = \begin{cases} a(\exp(u) - 1) & u \leq 0 \\ 1 & u > 0 \end{cases} \quad \text{ELU}$$

Obs: utilizate în rețelele cu structură adâncă (Deep NN)

$$\textit{softmax}(y) = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2) + \dots + \exp(y_M)}$$

Softmax – se utilizează doar pentru nivelul de ieșire în cazul rețelelor neuronale utilizate pentru clasificare – vectorul de ieșire poate fi interpretat ca o distribuție de probabilitate

Unități funcționale

Exemple de funcții de activare (funcții sigmoide)

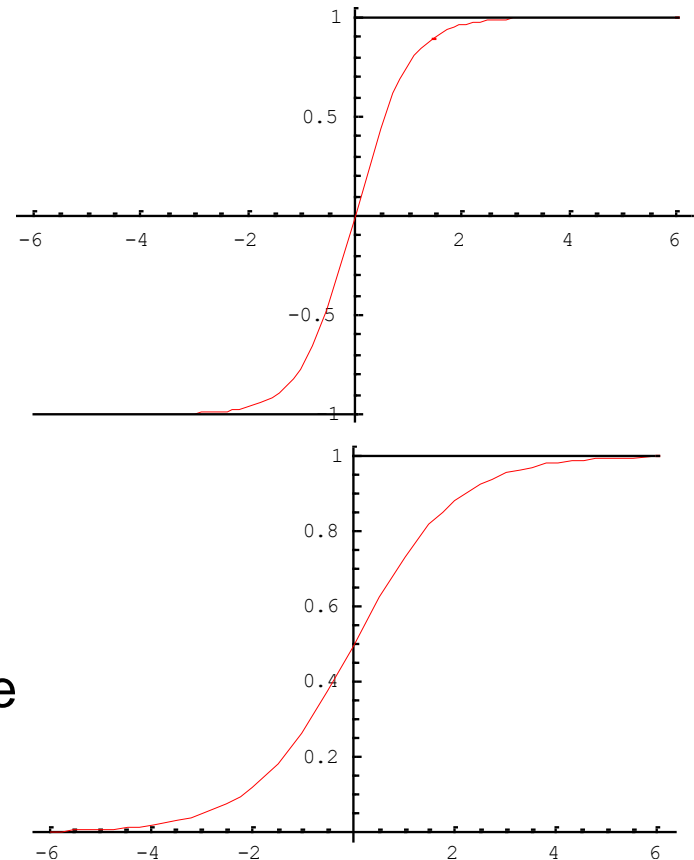
(tangenta hiperbolică)

$$f(u) = \tanh(u) = \frac{\exp(2u) - 1}{\exp(2u) + 1}$$

$$f(u) = \frac{1}{1 + \exp(-u)}$$

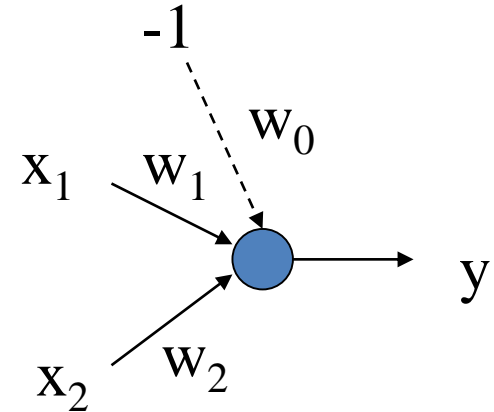
(logistică)

Observație: uneori se folosește un parametru numit pantă (slope) care multiplică argumentul funcției de activare: $y=f(p*u)$



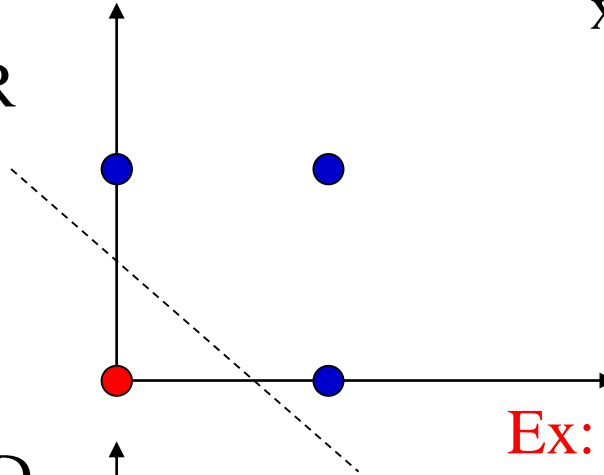
Unități funcționale

- Ce se poate face cu un singur neuron ?
Se pot rezolva probleme simple de clasificare
(ex: se pot reprezenta funcții booleene simple)



	0	1
0	0	1
1	1	1

OR

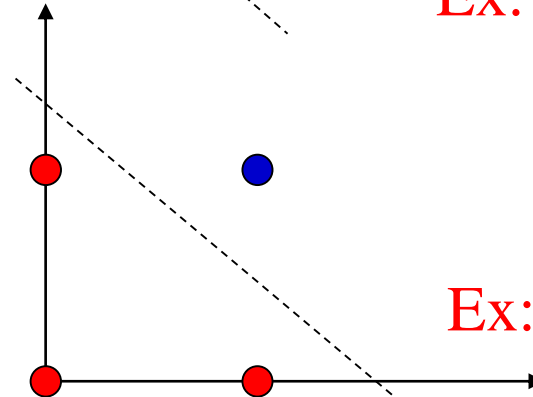


$$y = H(w_1x_1 + w_2x_2 - w_0)$$

Ex: $w_1 = w_2 = 1, w_0 = 0.5$

	0	1
0	0	0
1	0	1

AND

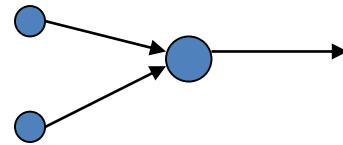
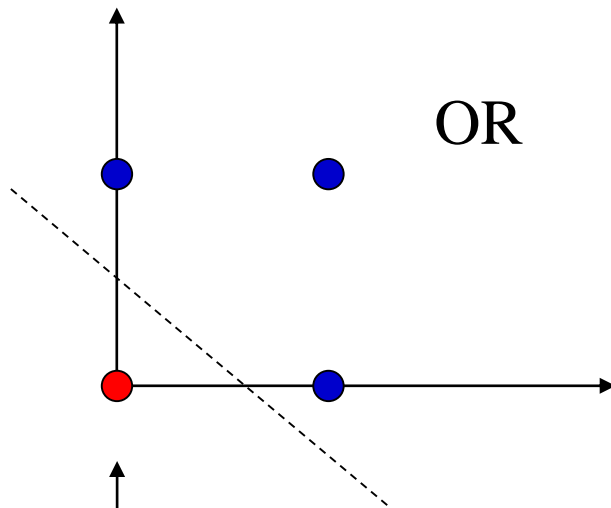


$$y = H(w_1x_1 + w_2x_2 - w_0)$$

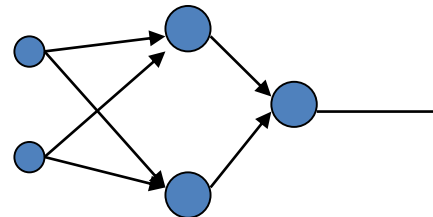
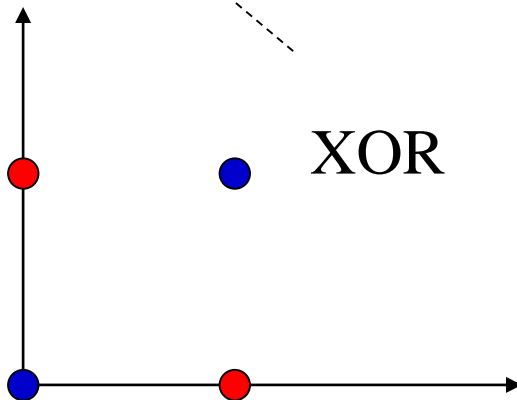
Ex: $w_1 = w_2 = 1, w_0 = 1.5$

Liniar/nelinier separabilitate

Reprezentarea unor funcții booleene: $f: \{0,1\}^N \rightarrow \{0,1\}$



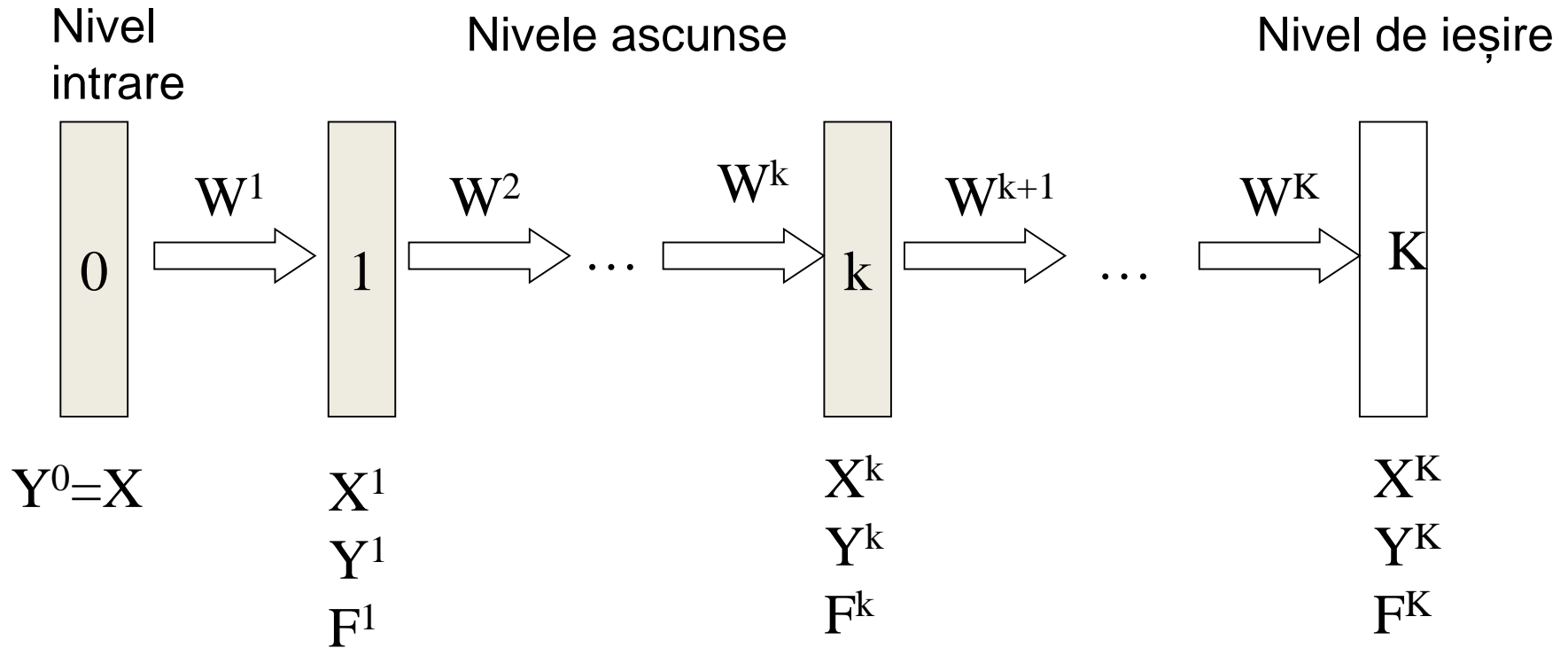
Problema liniar
separabilă – e suficientă
o **rețea univel**



Problema neliniar
separabilă – e necesară
o **rețea multinivel**
(cel puțin un **nivel ascuns** –
hidden layer)

Rețele feedforward - arhitectura

Arhitectura și funcționare (K nivele funcționale)



X = vector intrare, Y = vector ieșire, F = funcție vectorială de activare

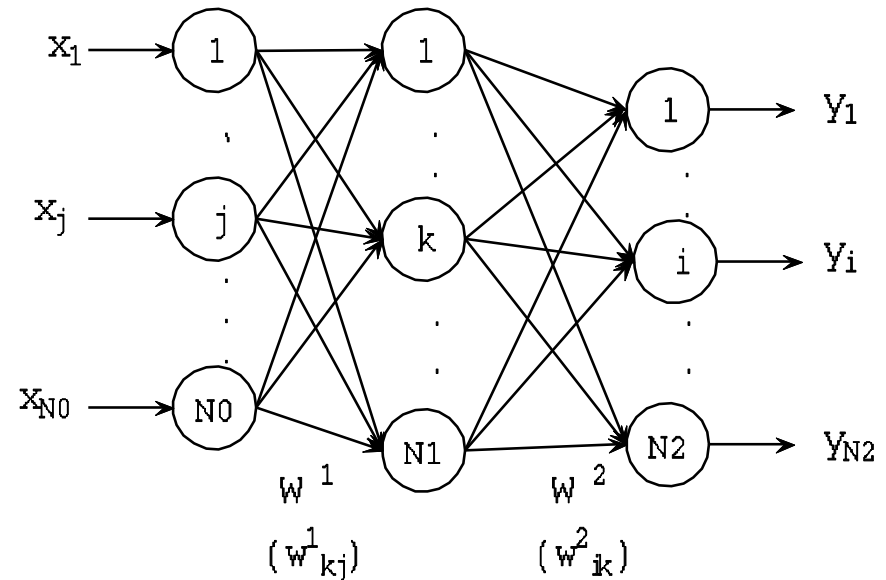
Calcul vector de ieșire: $Y = F^K(W^K * F^{K-1}(W^{K-1} * F^{K-2}(\dots F^1(W^1 * X))))$

Rețele feedforward – funcționare

Arhitectura și funcționare
(caz particular: un nivel ascuns)

Parametrii modelului: matricile cu ponderi W^1 și W^2 (setul tuturor ponderilor e notat cu W)

$$y_i = f_2 \left(\sum_{k=0}^{N_1} w^{(2)}_{ik} f_1 \left(\sum_{j=0}^{N_0} w^{(1)}_{kj} x_j \right) \right), \quad i = 1..N_2$$



Obs:

- În mod tradițional se lucrează cu unul sau două nivele ascunse
- rețelele cu număr mare de nivele sau cu structură adâncă (**Deep Neural Networks**) sunt folosite frecvent în particular pentru recunoașterea imaginilor și a vorbirii (<https://www.deeplearningbook.org/>)

Rețele feedforward - antrenare

Antrenare (supervizată):

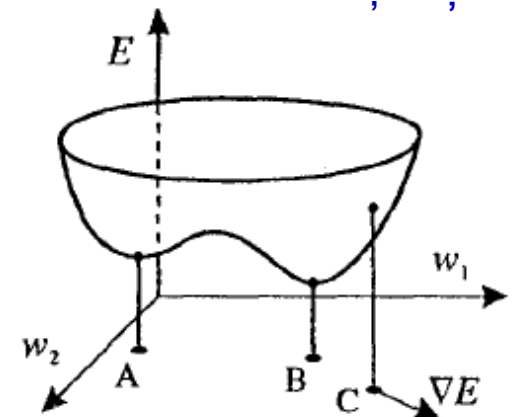
- Set de antrenare: $\{(x^1, d^1), \dots, (x^L, d^L)\}$
(x^l = vector intrare, d^l = vector de ieșire corect)
- Funcție de eroare (suma pătratelor erorilor):

$$E(W) = \frac{1}{2} \sum_{l=1}^L \sum_{i=1}^M \left(d_i^l - f_2 \left(\sum_{k=0}^K w_{ik} f_1 \left(\sum_{j=0}^N w_{kj} x_j^l \right) \right) \right)^2$$

- Scopul antrenării: **minimizarea funcției de eroare**
- Metoda de minimizare: **metoda gradientului (gradient descent)**

Notății:

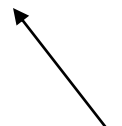
- N = nr unități intrare
- K = nr unități ascunse
- M = nr unități ieșire



Rețele feedforward - antrenare

Relația de ajustare (metoda gradientului):

$$w_{ij}(t + 1) = w_{ij}(t) - \eta \frac{\partial E(w(t))}{\partial w_{ij}}$$


 Pas descreștere
 =
Rata de învățare

Funcția de eroare:

$$E(W) = \frac{1}{2} \sum_{l=1}^L \sum_{i=1}^M \left(d_i^l - f_2 \left(\sum_{k=0}^K w_{ik} f_1 \left(\sum_{j=0}^N w_{kj} x_j^l \right) \right) \right)^2$$

$$\underbrace{\quad \quad \quad}_{x_k}$$

$$\underbrace{\quad \quad \quad}_{y_k}$$

$$\underbrace{\quad \quad \quad}_{x_i}$$

$$\underbrace{\quad \quad \quad}_{y_i}$$

$E_l(W)$ (eroarea corespunzătoare exemplului l)

Notații:

Rețele feedforward - antrenare

- Calculul derivatelor parțiale

$$E(W) = \frac{1}{2} \sum_{l=1}^L \sum_{i=1}^M \left(d_i^l - f_2 \left(\underbrace{\sum_{k=0}^K w_{ik} f_1 \left(\underbrace{\sum_{j=0}^N w_{kj} x_j^l \right)}_{x_k} \right)}_{y_k} \right) \right)^2$$

$\underbrace{\hspace{10em}}_{x_i}$
 $\underbrace{\hspace{10em}}_{y_i}$

$$\frac{\partial E_l(W)}{\partial w_{ik}} = -(d_i^l - y_i) f_2'(x_i) y_k = -\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}} = - \sum_{i=1}^M w_{ik} (d_i^l - y_i) f_2'(x_i) f_1'(x_k) x_j^l = - \left(f_1'(x_k) \sum_{i=1}^M w_{ik} \delta_i^l \right) x_j^l = -\delta_k^l x_j^l$$

Obs: δ_i reprezintă o măsură a erorii corespunzătoare unității de ieșire i iar δ_k reprezintă eroarea de la nivelul unității ascuns k (obținut prin propagarea înapoi în rețea a erorii de la nivelul de ieșire)

Rețele feedforward - antrenare

$$\frac{\partial E_l(W)}{\partial w_{ik}} = -(d_i^l - y_i) f_2'(x_i) y_k = -\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}} = - \sum_{i=1}^M w_{ik} (d_i^l - y_i) f_2'(x_i) f_1'(x_k) x_j^l = - \left(f_1'(x_k) \sum_{i=1}^M w_{ik} \delta_i^l \right) x_j^l = -\delta_k^l x_j^l$$

Obs: derivatele funcțiilor tradiționale de activare (logistica și tanh) pot fi calculate simplu folosind următoarele proprietăți:

Logistica: $f'(x) = f(x)(1-f(x)) \Rightarrow f'(x) = y(1-y)$

Tanh: $f'(x) = 1-f(x)^2 \Rightarrow f'(x) = 1-y^2$

ReLU: $f'(x) = 0$ pt $x < 0$, $f'(x) = 1$ pt $x > 0$

ELU: $f'(x) = f(x) + a$ pt $x < 0$, $f'(x) = 1$ pt $x > 0$

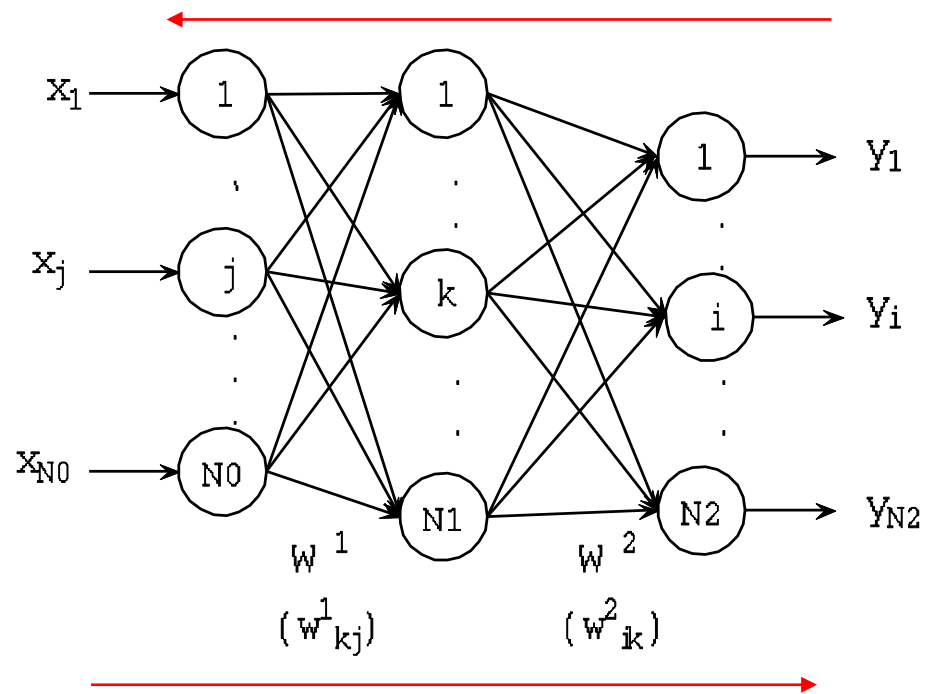
Algoritmul BackPropagation

Idee:

Pentru fiecare exemplu din setul de antrenare (sau din subset – minibatch):

- se determină semnalul de ieșire
- se calculează eroarea la nivelul de ieșire
- se propagă eroarea înapoi în rețea și se reține factorul delta corespunzător fiecărei ponderi
- se aplică ajustarea corespunzătoare fiecărei ponderi

Calcul semnal eroare (BACKWARD)



Calcul semnal ieșire (FORWARD)

Algoritmul BackPropagation

Inițializarea aleatoare a ponderilor

Obs.

REPEAT

FOR I=1,L DO

etapa FORWARD

etapa BACKWARD

ajustare ponderi

Recalcularea erorii

UNTIL <condiție oprire>

epoca

- Valorile inițiale se aleg aleator in $[0,1]$ sau $[-1,1]$ (preferabil)
- La ajustare se ține cont de rata de învățare (parametrul eta)
- Recalcularea erorii presupune determinarea semnalului de ieșire pentru fiecare dată de intrare
- Condiția de oprire depinde de valoarea erorii și/sau numărul de epoci de antrenare

Algoritmul BackPropagation

Varianta serială

$$w_{kj}^1 = \text{rand}(-1,1), w_{ik}^2 = \text{rand}(-1,1)$$

$$p = 0$$

REPEAT

FOR $l = 1, L$ DO

/* Etapa FORWARD */

$$x_k^l = \sum_{j=0}^N w_{kj}^1 x_j^l, y_k^l = f_1(x_k^l), x_i^l = \sum_{k=0}^K w_{ik}^2 y_k^l, y_i^l = f_2(x_i^l)$$

/* Etapa BACKWARD */

$$\delta_i^l = f_2'(x_i^l)(d_i^l - y_i^l), \delta_k^l = f_1'(x_k^l) \sum_{i=1}^M w_{ik}^2 \delta_i^l$$

/* Etapa de ajustare */

$$w_{kj}^1 = w_{kj}^1 + \eta \delta_k^l x_j^l, \quad w_{ik}^2 = w_{ik}^2 + \eta \delta_i^l y_k^l$$

ENDFOR

Obs. varianta “stochastic gradient descent” se caracterizează prin selectia, aleatoare, la fiecare epocă, a unui subset din setul de antrenare și parcurgerea acestuia

Algoritmul BackPropagation

/* Calculul erorii */

$$E = 0$$

FOR $l = 1, L$ DO

/* Etapa FORWARD (cu noile valori ale ponderilor)*/

$$x_k^l = \sum_{j=0}^N w_{kj}^1 x_j^l, y_k^l = f_1(x_k^l), x_i^l = \sum_{k=0}^K w_{ik}^2 y_k^l, y_i^l = f_2(x_i^l)$$

/* Sumarea erorii */

$$E = E + \sum_{l=1}^L (d_i^l - y_i^l)^2$$

ENDFOR

$$E = E / (2L)$$

$$p = p + 1$$

UNTIL $p > p_{\max}$ OR $E < E^*$

E^* reprezintă toleranța la erori a rețelei
 p_{\max} reprezintă numărul maxim de epoci
de antrenare

Algoritmul BackPropagation

Varianta pe blocuri (se bazează pe cumularea ajustarilor)

– batch variant

$$w_{kj}^1 = rand(-1,1), w_{ik}^2 = rand(-1,1), \quad i = 1..M, k = 0..K, j = 0..N$$

$$p = 0$$

REPEAT

$$\Delta_{kj}^1 = 0, \Delta_{ik}^2 = 0$$

FOR $l = 1, L$ DO

/* Etapa FORWARD */

$$x_k^l = \sum_{j=0}^{N0} w_{kj}^1 x_j^l, y_k^l = f_1(x_k^l), x_i^l = \sum_{k=0}^{N1} w_{ik}^2 y_k^l, y_i^l = f_2(x_i^l)$$

/* Etapa BACKWARD */

$$\delta_i^l = f_2'(x_i^l)(d_i^l - y_i^l), \delta_k^l = f_1'(x_k^l) \sum_{i=1}^{N2} w_{ik}^2 \delta_i^l$$

/* Etapa de ajustare */

$$\Delta_{kj}^1 = \Delta_{kj}^1 + \eta \delta_k^l x_j^l, \quad \Delta_{ik}^2 = \Delta_{ik}^2 + \eta \delta_i^l y_k^l$$

ENDFOR

$$w_{kj}^1 = w_{kj}^1 + \Delta_{kj}^1, \quad w_{ik}^2 = w_{ik}^2 + \Delta_{ik}^2$$

Algoritmul BackPropagation

```
/* Calculul erorii */  
E = 0  
FOR l = 1, L DO  
  /* Etapa FORWARD (cu noile valori ale ponderilor)*/  
  
$$x_k^l = \sum_{j=0}^N w_{kj}^1 x_j^l, y_k^l = f_1(x_k^l), \quad x_i^l = \sum_{k=0}^K w_{ik}^2 y_k^l, \quad y_i^l = f_2(x_i^l)$$
  
  /* Sumarea erorii */  
  
$$E = E + \sum_{l=1}^L (d_i^l - y_i^l)^2$$
  
ENDFOR  
E = E/(2L)  
p = p + 1  
UNTIL p > p_max OR E>E*
```

Variante

Altă funcție de eroare:

- MSE (eroarea medie pătratică) este mai potrivită pentru problemele de regresie
- In cazul problemelor de clasificare o variantă mai adecvată este entropia încrucișată (**cross-entropy error**)
- Caz particular: clasificare binară (un neuron de ieșire):
 - d_l aparține lui $\{0,1\}$ (0 corespunde clasei 0 și 1 corespunde clasei 1)
 - y_l aparține lui $(0,1)$ și poate fi interpretat ca probabilitatea clasei 1

$$CE(W) = - \sum_{l=1}^L (d_l \log y_l + (1 - d_l) \log(1 - y_l))$$

Obs: forma derivatelor parțiale se schimbă, deci și termenii utilizați în ajustarea ponderilor – principiul general al propagării înapoi a erorii rămâne însă valabil;

Variante

Entropia încrucișată + funcție de activare logistică:

- În cazul problemelor de clasificare o variantă mai adecvată este entropia încrucișată (**cross-entropy error**)
- Caz particular: clasificare binară (un neuron de ieșire):
 - d_l aparține lui $\{0,1\}$ (0 corespunde clasei 0 și 1 corespunde clasei 1)
 - y_l aparține lui $(0,1)$ și poate fi interpretat ca probabilitatea clasei 1

$$CE(W) = - \sum_{l=1}^L (d_l \log y_l + (1 - d_l) \log(1 - y_l))$$

$$\delta_l = \left(\frac{d_l}{y_l} - \frac{1 - d_l}{1 - y_l} \right) f_2'(x^{(2)}) = \frac{d_l(1 - y_l) - y_l(1 - d_l)}{y_l(1 - y_l)} \cdot y_l(1 - y_l)$$

$$= d_l(1 - y_l) - y_l(1 - d_l) = d_l - y_l$$

Probleme ale algoritmului Backpropagation

- P1. Viteza mică de convergență (eroarea descrește prea încet)
- P2. Oscilații (valoarea erorii oscilează în loc să descrească în mod continuu)
- P3. Problema minimelor locale (procesul de învățare se blochează într-un minim local al funcției de eroare)
- P4. Stagnare (procesul de învățare stagnează chiar dacă nu s-a ajuns într-un minim local)
- P5. Supraantrenarea și capacitatea limitată de generalizare

Probleme ale algoritmului BP

P1-P2: Eroarea descrește prea încet sau oscilează în loc să descrească

Cauze:

- Valoare inadecvată a ratei de învățare (valori prea mici conduc la convergența lentă iar valori prea mari conduc la oscilații)

Soluție: adaptarea ratei de învățare

- Metoda de minimizare are convergență lentă

Soluții:

- modificarea euristică a variantei standard (**varianta cu moment**)
- utilizarea unei alte metode de minimizare (**Newton, gradient conjugat**)

Probleme ale algoritmului BP

- Rata adaptivă de învățare:
 - Dacă eroarea crește semnificativ atunci rata de învățare trebuie redusă (ajustările obținute pentru valoarea curentă a ratei sunt ignorate)
 - Dacă eroarea descrește semnificativ atunci rata de învățare poate fi mărită (ajustările sunt acceptate)
 - In toate celelalte cazuri rata de învățare rămâne neschimbată

$$E(p) > (1 + \gamma)E(p - 1) \Rightarrow \eta(p) = a\eta(p - 1), 0 < a < 1$$

$$E(p) < (1 - \gamma)E(p - 1) \Rightarrow \eta(p) = b\eta(p - 1), 1 < b < 2$$

$$(1 - \gamma)E(p - 1) \leq E(p) \leq (1 + \gamma)E(p - 1) \Rightarrow \eta(p) = \eta(p - 1)$$

Exemplu: $\gamma=0.05$

Probleme ale algoritmului BP

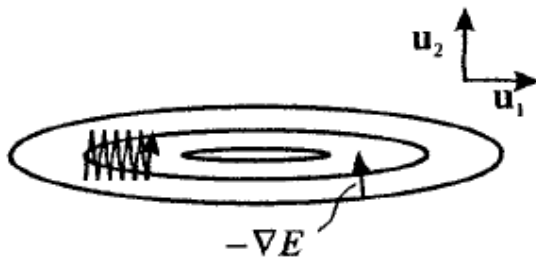
- Varianta cu “moment” (termen de inerție):
 - Se introduce o “inerție” în calculul ponderilor:
 - termenul de ajustare a ponderilor de la epoca curentă se calculează pe baza semnalului de eroare precum și a ajustărilor de la epoca anterioară
 - Acționează ca o adaptare a ratei de învățare: ajustările sunt mai mari în porțiunile plate ale funcției de eroare și mai mici în cele abrupte
 - Se combină cu varianta pe blocuri (batch)

$$\Delta w_{ij}(p + 1) = \eta \delta_i y_j + \alpha \Delta w_{ij}(p)$$

$$\alpha = 0.9$$

Probleme ale algoritmului BP

- Varianta cu “moment” (termen de inerție):
 - Se introduce o “inerție” în calculul ponderilor:
 - termenul de ajustare a ponderilor de la epoca curentă se calculează pe baza semnalului de eroare precum și a ajustărilor de la epoca anterioară



Metoda clasică



Utilizarea unui termen de inerție

Probleme ale algoritmului BP

Alte metode de minimizare (mai rapide însă mai complexe):

- Metoda gradientului conjugat (și variante ale ei)
- Metoda lui Newton (caz particular: Levenberg Marquardt)

Particularități ale acestor metode:

- Convergența rapidă (ex: metoda gradientului conjugat converge în n iterații pentru funcții pătratice cu n variabile)
- Necesită calculul matricii hessiene (matrice conținând derivatele de ordin doi ale funcției de eroare) și uneori a inversei acesteia

Probleme ale algoritmului BP

- Exemplu: metoda lui Newton

$$E: R^n \rightarrow R, \quad w \in R^n \text{ (vectorul ce contine toate ponderile)}$$

Prin dezvoltare in serie Taylor in $w(p)$ (estimarea corespunzatoare epocii p)

$$E(w) \cong E(w(p)) + (\nabla E(w(p)))^T (w - w(p)) + \frac{1}{2} (w - w(p))^T H(w(p)) (w - w(p))$$

$$H(w(p))_{ij} = \frac{\partial^2 E(w(p))}{\partial w_i \partial w_j}$$

Derivand dezvoltarea in serie Taylor in raport cu w si punand conditia de punct critic noua aproximare pentru w se va obtine ca solutie a ecuatiei:

$$H(w(p))w - H(w(p))w(p) + \nabla E(w(p)) = 0$$

Noua estimare a lui w va fi:

$$w(p + 1) = w(p) - H^{-1}(w(p)) \cdot \nabla E(w(p))$$

Probleme ale algoritmului BP

Caz particular: metoda Levenberg-Marquardt

- Metoda lui Newton adaptată pentru cazul în care eroarea este o sumă de pătrate de diferențe (cum este eroarea medie patrativă)

$$E(w) = \sum_{l=1}^L E_l(w), \quad e: R^n \rightarrow R^L, e(w) = (E_1(w), \dots, E_L(w))^T$$

$$w(p+1) = w(p) - (J^T(w(p)) \cdot J(w(p)) + \mu_p I)^{-1} J^T(w(p)) e(w(p))$$

$J(w)$ = jacobianul lui $e(w)$ = matricea derivatelor lui e în raport cu toate argumentele

$$J_{ij}(w) = \frac{\partial E_i(w)}{\partial w_j}$$

Termen de perturbare care elimina cazurile singulare (când matricea este neinvertibilă)

Avantaje:

- Nu necesită calculul hessianei
- Pentru valori mari ale factorului de atenuare ajustarea devine similară celei de la metoda gradientului

Probleme ale algoritmului BP

P3: Problema minimelor locale (procesul de învățare se blochează într-un minim local al funcției de eroare)

Cauza: metoda gradientului este o metodă de minimizare locală

Soluții:

- Se restartează antrenarea de la alte valori inițiale ale ponderilor
- Se introduc perturbații aleatoare (se adaugă la ponderi după aplicarea ajustărilor):

$w_{ij} := w_{ij} + \xi_{ij}$, ξ_{ij} = valori aleatoare uniform sau normal distribuite

Probleme ale algoritmului BP

Soluție:

- Inlocuirea metodei gradientului cu o metodă aleatoare de optimizare
- Inseamnă utilizarea unei perturbații aleatoare în locul celei calculate pe baza gradientului
- Ajustările pot conduce la creșterea valorii erorii

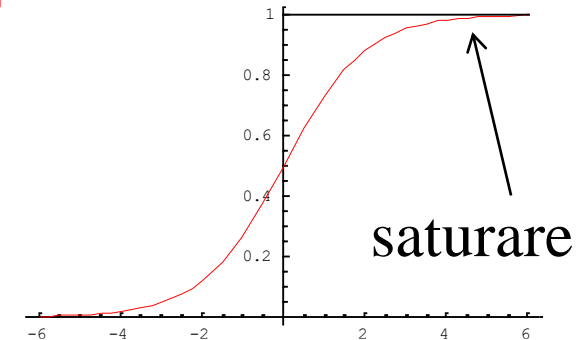
Δ_{ij} : = valori aleatoare

IF $E(W + \Delta) < E(W)$ THEN se accepta ajustare ($W := W + \Delta$)

Obs:

- Ajustările sunt de regulă generate în conformitate cu repartiția normală de medie 0 și dispersie adaptivă
- Dacă ajustarea nu conduce la o descreștere a valorii erorii atunci nu se acceptă deloc sau se acceptă cu o probabilitate mică
- Algoritmii aleatori de minimizare nu garantează obținerea minimumului dar unii dintre ei satisfac proprietăți de convergență în sens probabilist.

Probleme ale algoritmului BP



- **Pb 4: Stagnare**
(procesul de învățare stagnează chiar dacă nu s-a ajuns într-un minim local)
- **Cauza:** ajustările sunt foarte mici întrucât se ajunge la argumente mari ale funcțiilor sigmoide ceea ce conduce la valori foarte mici ale derivatelor; argumentele sunt mari fie datorită faptului ca **datele de intrare nu sunt normalizate** fie întrucât **valorile ponderilor sunt prea mari**
- **Soluții:**
 - Se “penalizează” valorile mari ale ponderilor prin **regularizare**
 - Se utilizează doar semnele derivatelor nu și valorile lor
 - Se normalizează datele de intrare (valori în apropierea intervalului (-1,1))
 - Se utilizează funcții de activare de tip ReLU

Probleme ale algoritmului BP

Penalizarea valorilor mari ale ponderilor: se adaugă un termen de **penalizare** la funcția de eroare (similar cu tehnicile de **regularizare** folosite în metodele de optimizare)

$$E_{(r)}(W) = E(W) + \lambda \sum_{i,j} w_{ij}^2$$

Ajustarea va fi:

$$\Delta_{ij}^{(r)} = \Delta_{ij} - 2\lambda w_{ij}$$

Obs: o altă variantă de regularizare este cea în care în loc de pătrate ale valorii ponderilor se consideră valoarea absolută ($|w_{ij}|$) – regularizare de tip Lasso (favorizează creșterea numărului de ponderi nule)

Probleme ale algoritmului BP

Utilizarea semnului derivatei nu și a valorii

(Resilient BackPropagation – RPROP)

$$\Delta w_{ij}(p) = \begin{cases} -\Delta_{ij}(p) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} > 0 \\ \Delta_{ij}(p) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} < 0 \end{cases}$$

$$\Delta_{ij}(p) = \begin{cases} a\Delta_{ij}(p-1) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \frac{\partial E(W(p-2))}{\partial w_{ij}} > 0 \\ b\Delta_{ij}(p-1) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \frac{\partial E(W(p-2))}{\partial w_{ij}} < 0 \end{cases}$$

$$0 < b < 1 < a$$

Probleme ale algoritmului BP

Pb 5: Supraantrenare și capacitate limitată de generalizare

Cauze:

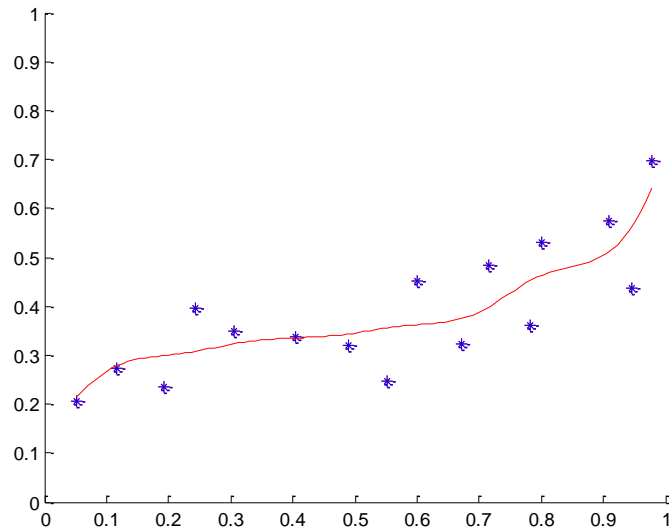
- **Arhitectura rețelei (numărul de unități ascunse)**
 - Un număr prea mare de unități ascunse poate provoca supraantrenare (rețeaua extrage nu doar informațiile utile din setul de antrenare ci și zgomotul)
- **Dimensiunea setului de antrenare**
 - Prea puține exemple nu permit antrenarea și asigurarea capacității de generalizare
- **Numărul de epoci (toleranța la antrenare)**
 - Prea multe epoci pot conduce la supraantrenare

Soluții:

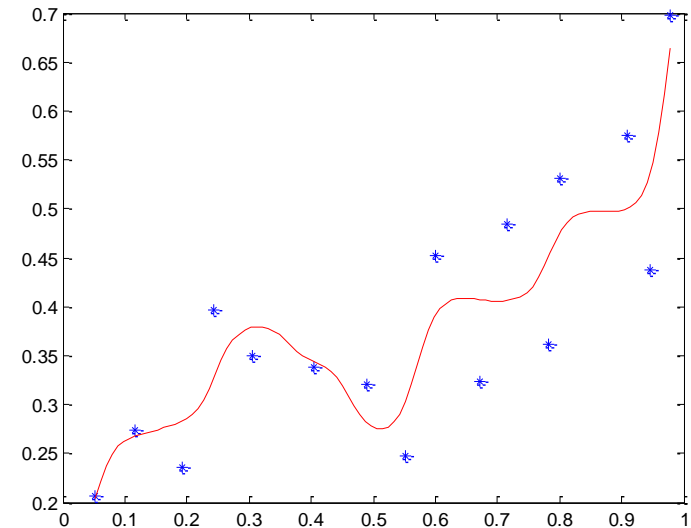
- Modificarea dinamică a arhitecturii
- Criteriul de oprire se bazează nu pe eroarea calculată pentru setul de antrenare ci pentru un **set de validare**

Probleme ale algoritmului BP

Supraantrenare – influența numărului de unități ascunse



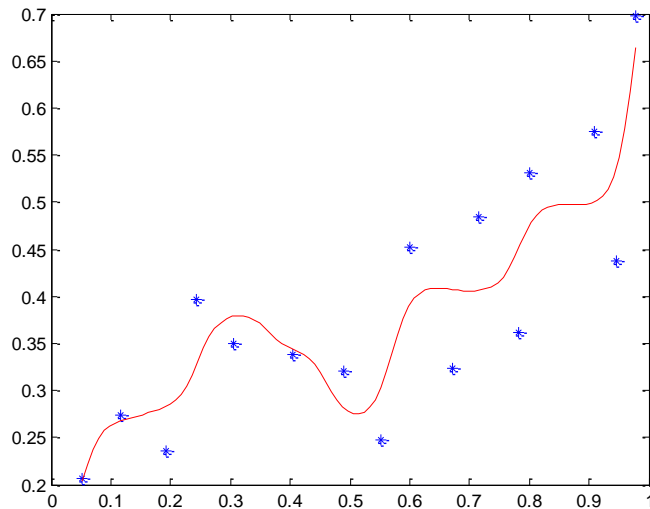
5 unități ascunse



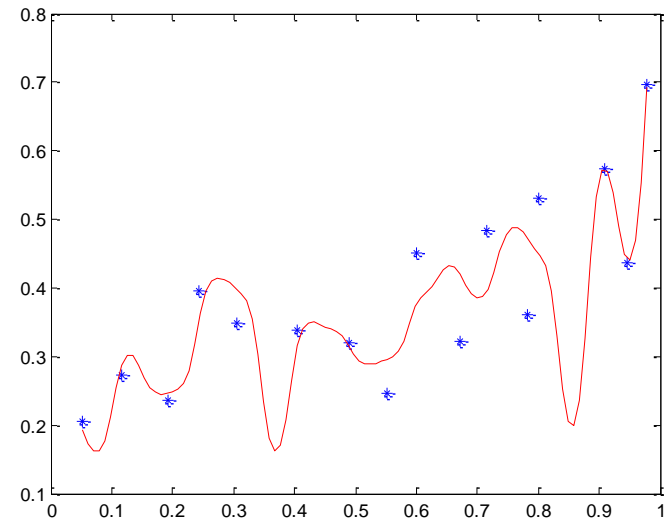
10 unități ascunse

Probleme ale algoritmului BP

Supraantrenare – influența numărului de unități ascunse



10 unități ascunse



20 unități ascunse

Probleme ale algoritmului BP

Modificarea dinamică a arhitecturii:

- **Strategie incrementală:**
 - Se pornește cu un număr mic de unități ascunse
 - Dacă antrenarea nu progresează se adaugă succesiv unități; pentru asimilarea lor se ajustează în câteva epoci doar ponderile corespunzătoare
- **Strategie decrementală:**
 - Se pornește cu un număr mare de unități
 - Dacă există unități care au impact mic asupra semnalului de ieșire atunci acestea se elimină

Probleme ale algoritmului BP

Criteriu de oprire bazat pe eroarea pe setul de validare :

- Se imparte setul de antrenare în m părți: $(m-1)$ sunt folosite pentru antrenare și una pentru validare
- Ajustarea se aplică până când eroarea pe setul de validare începe să crească (sugerează că rețeaua începe să piardă din abilitatea de generalizare)

Validare încrucișată:

- Algoritmul de învățare se aplică de m ori pentru cele m variante posibile de selecție a subsetului de validare

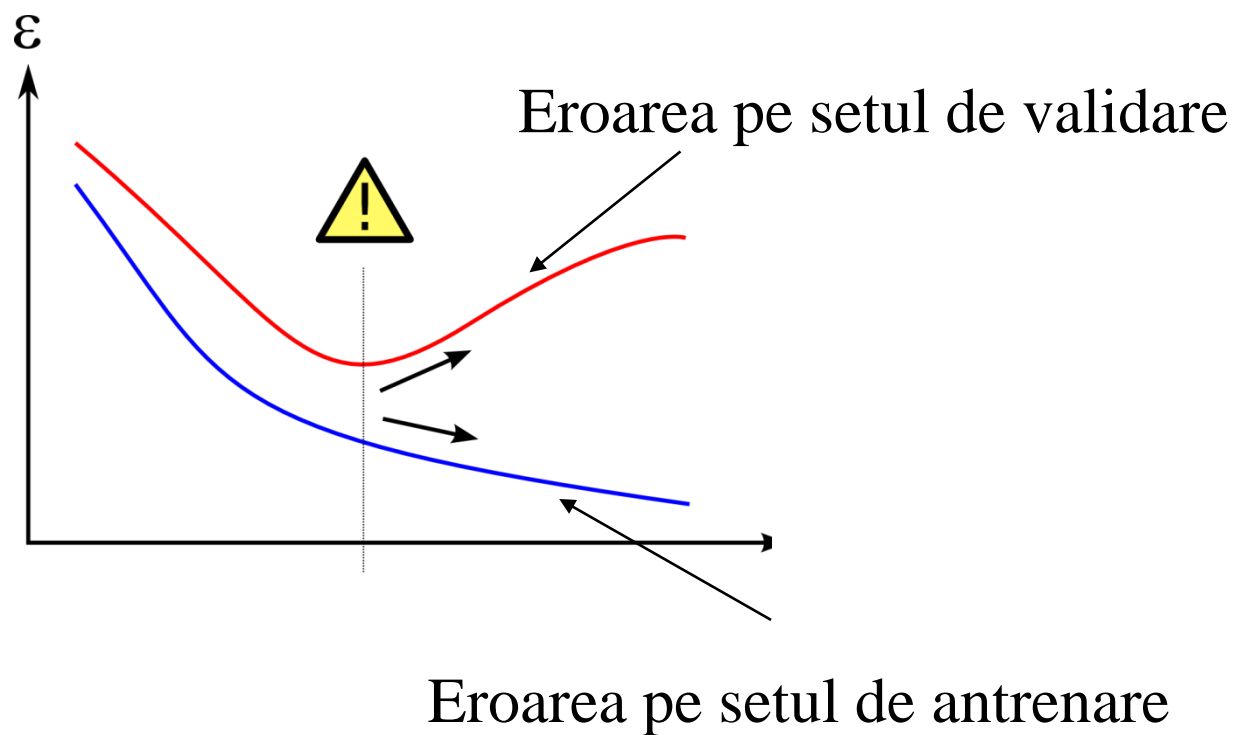
1: $S=(S_1, S_2, \dots, S_m)$

2: $S=(S_1, S_2, \dots, S_m)$

....

m : $S=(S_1, S_2, \dots, S_m)$

Probleme ale algoritmului BP



Support Vector Machines

Support Vector Machine (SVM) = tehnică de clasificare caracterizată prin:

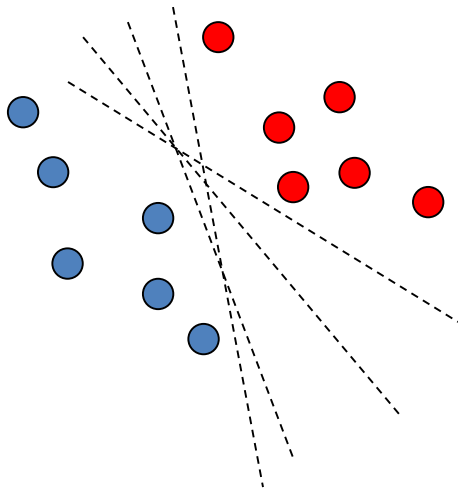
- Antrenare bazată pe o metodă de optimizare cu restricții și funcție obiectiv pătratică.

Obs: se evită problemele ce apar la antrenarea de tip Backpropagation (blocarea în minime locale și supraantrenarea)

- Asigură o bună capacitate de generalizare
- Se bazează pe rezultate teoretice din domeniul analizei statistice a metodelor de învățare (principalii contributori: Vapnik și Chervonenkis)
- Aplicații: recunoaștere scris, identificarea vorbitorului, recunoaștere obiecte etc
- Bibliografie: C.Burges – A Tutorial on SVM for Pattern Recognition, Data Mining and Knowledge Discovery, 2, 121–167 (1998)

Support Vector Machines

Considerăm o problemă simplă de clasificare binară



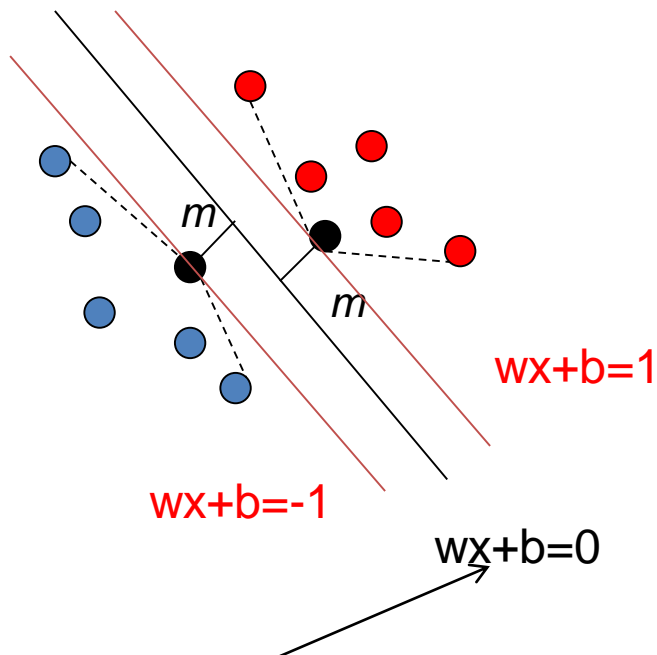
Problema e liniar separabilă și se observă că există o infinitate de drepte (hiperplane, în cazul general) care permit separarea celor două clase

Care dintre hiperplanele separatoare este mai bun ?

Cel care ar conduce la o bună capacitate de generalizare = clasificare corectă nu doar pentru datele din setul de antrenare ci și pentru potențialele date de test

Support Vector Machines

Care e cea mai bună dreaptă (hiperplan) separatoare ?



Ecuția dreptei
(hiperplanului) separatoare

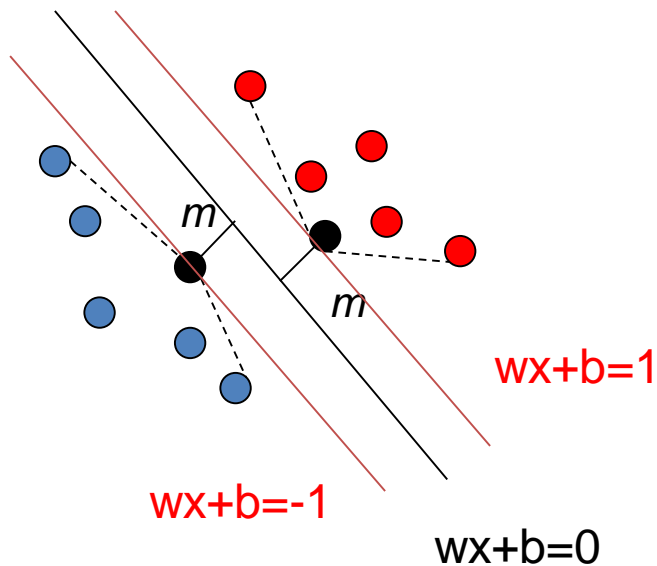
Cea pentru care distanța minimă față de punctele aflate pe înfășurătoarea convexă a setului de puncte corespunzător fiecărei clase este maximă

Dreptele care trec prin punctele marginale sunt considerate drepte canonice

Distanța dintre dreptele canonice este $2/\|w\|$, deci a maximiza lărgimea zonei separatoare este echivalent cu a minimiza norma lui w

Support Vector Machines

Cum se poate determina hiperplanul separator ?



Se determină w și b care

Minimizează $\|w\|^2$

(maximizează marginea separatoare)

și **satisface**

$$(wx_i+b)d_i-1 \geq 0$$

pentru toate elementele setului de antrenare $\{(x_1, d_1), (x_2, d_2), \dots, (x_L, d_L)\}$

$d_i = -1$ pentru clasa albastră

$d_i = 1$ pentru clasa roșie

(clasifică corect exemplele din setul de antrenare)

Support Vector Machines

Problema de minimizare cu restricții se poate rezolva folosind metoda multiplicatorilor lui Lagrange:

Problema inițială:

Minimizează $\|w\|^2$ astfel încât $(w \cdot x_i + b) d_i - 1 \geq 0$ pentru $i=1..L$

Introducerea multiplicatorilor lui Lagrange transformă problema în determinarea punctului șa (saddle point) pentru V :

$$V(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^L \alpha_i (d_i (w \cdot x_i + b) - 1), \quad \alpha_i \geq 0$$

(w^*, b^*, α^*) este punct șa dacă: $V(w^*, b^*, \alpha^*) = \max_{\alpha} \min_{w, b} V(w, b, \alpha)$

Construirea funcției duale:

$$W(\alpha) = \min_{w, b} V(w, b, \alpha)$$

$$\frac{\partial V(w, b, \alpha)}{\partial w} = 0 \Rightarrow w = \sum_{j=1}^L \alpha_j d_j x_j \quad \frac{\partial V(w, b, \alpha)}{\partial b} = 0 \Rightarrow 0 = \sum_{j=1}^L \alpha_j d_j$$

Support Vector Machines

Se ajunge astfel la problema **maximizării funcției duale** (în raport cu α):

$$W(\alpha) = \sum_{i=1}^L \alpha_i - \frac{1}{2} \sum_{i,j=1}^L \alpha_i \alpha_j d_i d_j (x_i \cdot x_j)$$

Cu restricțiile:

(cunoscute din setul de antrenare)

$$\alpha_i \geq 0, \quad \sum_{i=1}^L \alpha_i d_i = 0$$

După rezolvarea problemei de mai sus (în raport cu multiplicatorii α) se calculează elementele hiperplanului separator astfel:

$$w^* = \sum_{i=1}^L \alpha_i d_i x_i, \quad b^* = 1 - w^* \cdot x_k$$

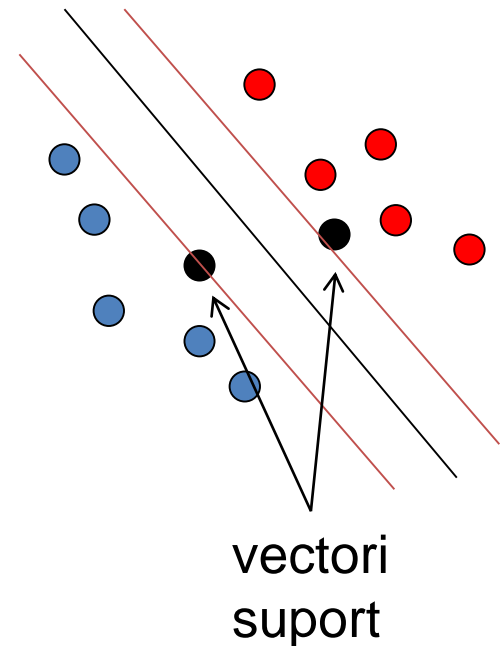
unde k este indicele unui multiplicator nenul iar x_k este exemplul corespunzător ce aparține clasei de etichetă +1

Support Vector Machines

Observații:

- Multiplicatorii nenuli corespund exemplelor pentru care restricțiile sunt **active** ($w \cdot x + b = 1$ sau $w \cdot x + b = -1$). Aceste exemple sunt denumite **vectori suport** și sunt singurele care influențează ecuația hiperplanului separator (celelalte exemple din setul de antrenare pot fi modificate fără a influența hiperplanul separator)
- Multiplicatorii nuli corespund elementelor din setul de antrenare care nu influențează hiperplanul separator
- Funcția de decizie obținută după rezolvarea problemei de optimizare pătratică este:

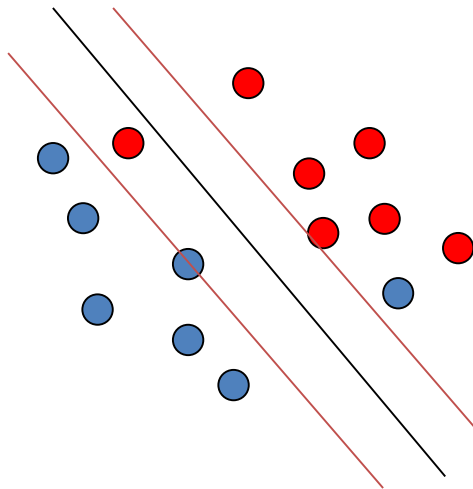
$$D(z) = \text{sgn}\left(\sum_{i=1}^L \alpha_i d_i (x_i \cdot z) + b^*\right)$$



Support Vector Machines

Ce se întâmplă în cazul în care datele nu sunt foarte bine separate ?

Se relaxează condiția de apartenență la o clasă:



$$\begin{aligned} w \cdot x_i + b &\geq 1 - \xi_i, & \text{daca } d_i = 1 \\ w \cdot x_i + b &\leq -1 + \xi_i, & \text{daca } d_i = -1 \end{aligned}$$

Funcția de minimizat devine:

$$V(w, b, \alpha, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^L \xi_i - \sum_{i=1}^L \alpha_i (d_i (w \cdot x_i + b) - 1)$$

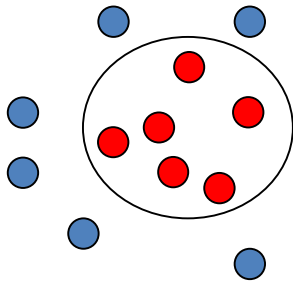
Ceea ce schimbă restricțiile din problema duală astfel:

$$\text{in loc de } \alpha_i \geq 0 \text{ se introduce } 0 \leq \alpha_i \leq C$$

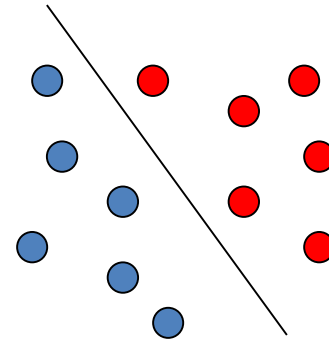
Obs: Parametrul C controlează compromisul între a accepta erori pe setul de antrenare și a avea margine largă (abilitate de generalizare). Cu cât C e mai mare cu atât restricția

Support Vector Machines

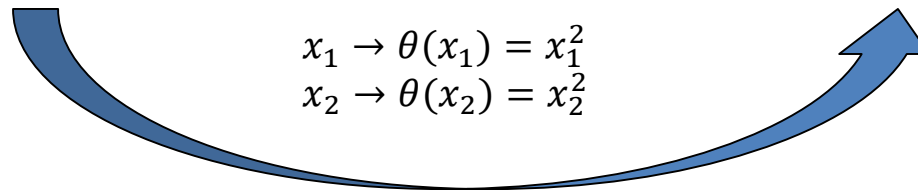
Ce se întâmplă în cazul în care problema NU este liniar separabilă?



$$x_1^2 + x_2^2 - R^2 = 0$$



$$w \cdot z + b = 0, \quad z_1 = x_1^2, \quad z_2 = x_2^2$$
$$w_1 = w_2 = 1, \quad b = -R^2$$



$$x_1 \rightarrow \theta(x_1) = x_1^2$$
$$x_2 \rightarrow \theta(x_2) = x_2^2$$

Support Vector Machines

In cazul general se aplică transformarea:

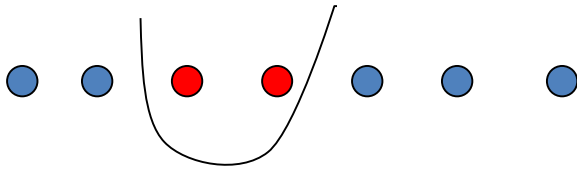
$$x \rightarrow \theta(x) \text{ iar produsul scalar al vectorilor transformați este}$$
$$\theta(x) \cdot \theta(x') = K(x, x')$$

Intrucât în rezolvarea problemei de optimizare intervin doar produsele scalare nu este necesară cunoașterea expresiei explicite a funcției de transformare θ ci este suficient să se cunoască doar **funcția nucleu K**

Support Vector Machines

Exemplu 1: Transformarea unei probleme neliniar separabile într-una liniar separabilă prin trecerea la o dimensiune mai mare

$$(x - \alpha)(x - \beta) = x^2 - (\alpha + \beta)x + \alpha\beta$$



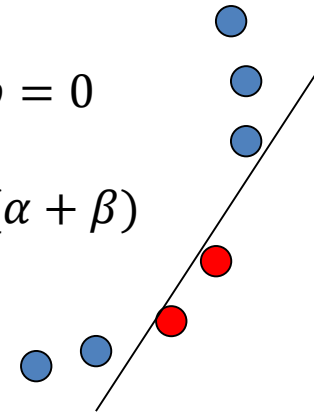
Pb. 1-dimensională neliniar separabilă

$$w_1 z_1 + w_2 z_2 + b = 0$$

$$z_1 = x^2, z_2 = x$$

$$w_1 = 1, w_2 = -(\alpha + \beta)$$

$$b = \alpha\beta$$



Pb. 2-dimensională liniar separabilă

Exemplu 2: Deducerea unei funcții nucleu în cazul în care suprafața de decizie este dată de o funcție pătratică oarecare (se trece de la dimensiunea 2 la dimensiunea 5)

$$\theta(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

$$K(x, x') = \theta(x_1, x_2) \cdot \theta(x'_1, x'_2) = (x^T \cdot x' + 1)^2$$

Support Vector Machines

Exemple de functii nucleu:

$$K(x, x') = (x^T \cdot x' + 1)^d$$

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

$$K(x, x') = \tanh(kx^T \cdot x' + b)$$

Functia de decizie devine:

$$D(z) = \text{sgn}\left(\sum_{i=1}^L \alpha_i y_i K(x_i, z) + b^*\right)$$

Support Vector Machines

Implementări

LibSVM [<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>]: (+ link-uri catre implementari in Java, Matlab, R, C#, Python, Ruby)

SVM-Light [http://www.cs.cornell.edu/People/tj/svm_light/]: implementare in C

Spider [<http://www.kyb.tue.mpg.de/bs/people/spider/tutorial.html>]: implementare Matlab

Interfață SciLab pt LibSVM (<http://atoms.scilab.org/toolboxes/libsvm>)

SciKit-learn – implementări în Python

R – pachet caret

Curs următor

Gruparea datelor

- Concepte de bază
- Evaluarea calității grupării
- Algoritmi partiționali
- Algoritmi ierarhici