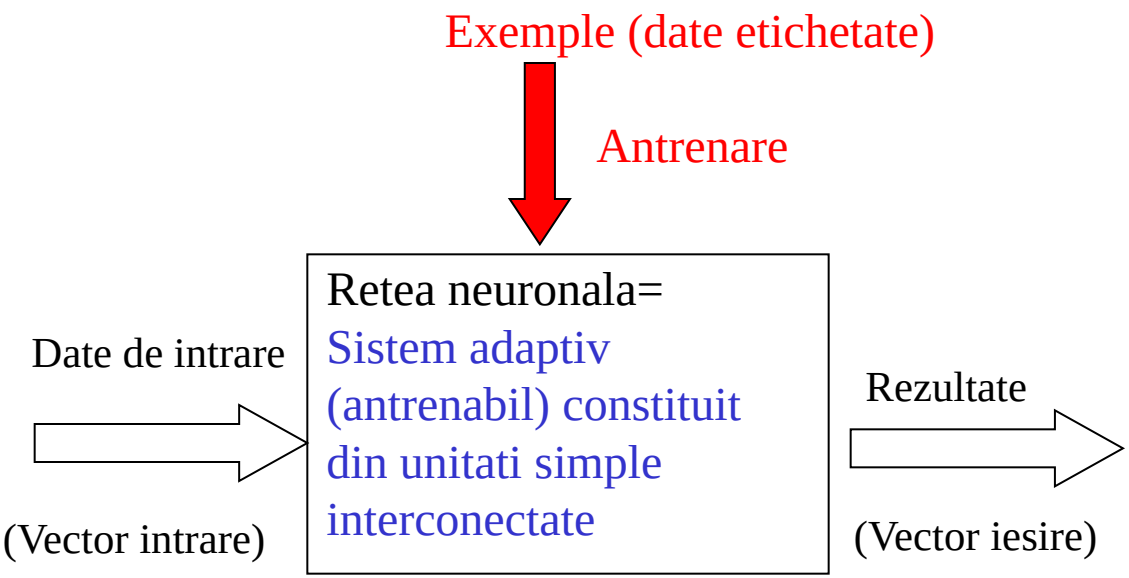
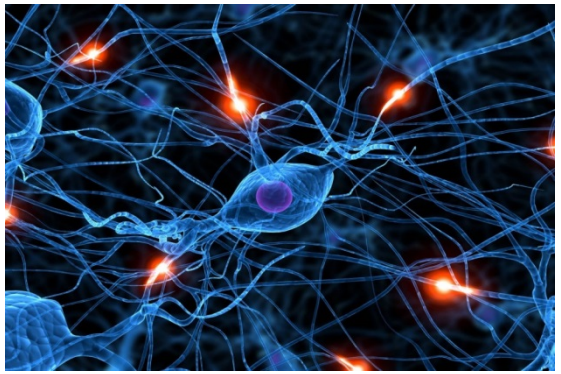
Rețele neuronale artificiale

- Probleme de asociere = clasificare, aproximare, predicție
- Rețele feed-forward uninivel si multinivel (multilayer perceptrons)
- Rețele cu funcții radiale (RBF networks)

Rețele neuronale artificiale

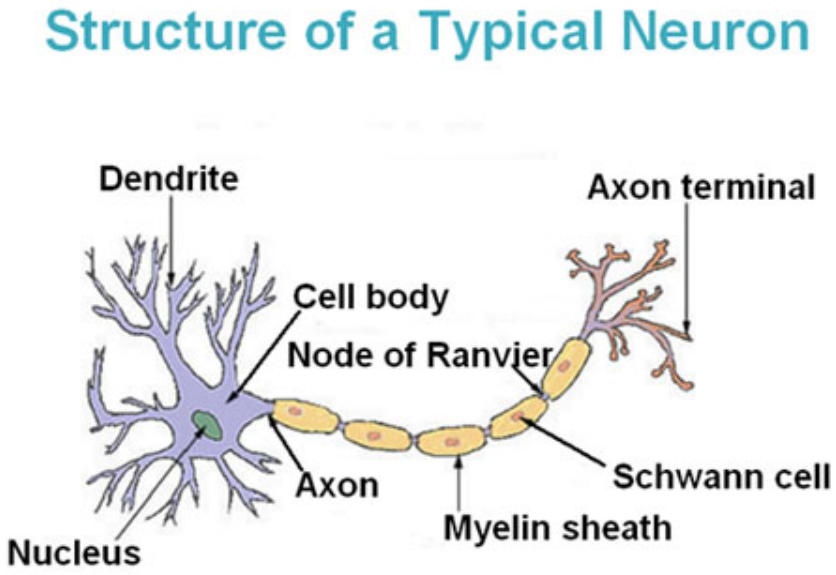
Retelele neuronale sunt sisteme adaptive de tip cutie neagra (black-box) care permit extragerea unui model pornind de la date printr-un proces de invatare

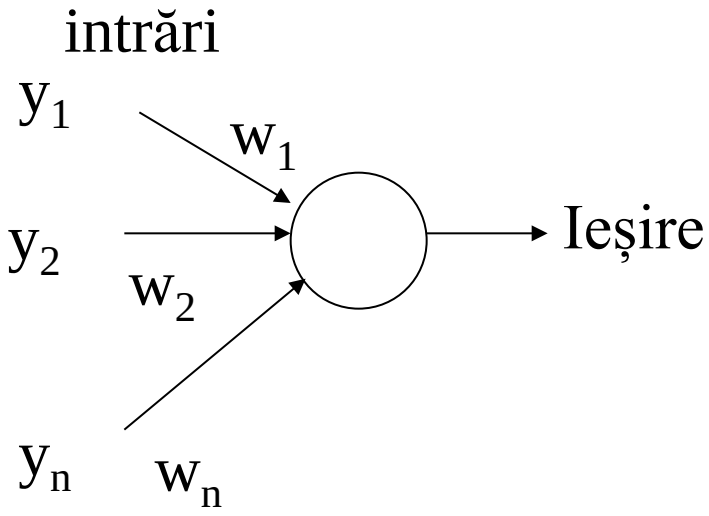




- Retelele neuronale sunt inspirate de modul de structurare si functionare a creierului

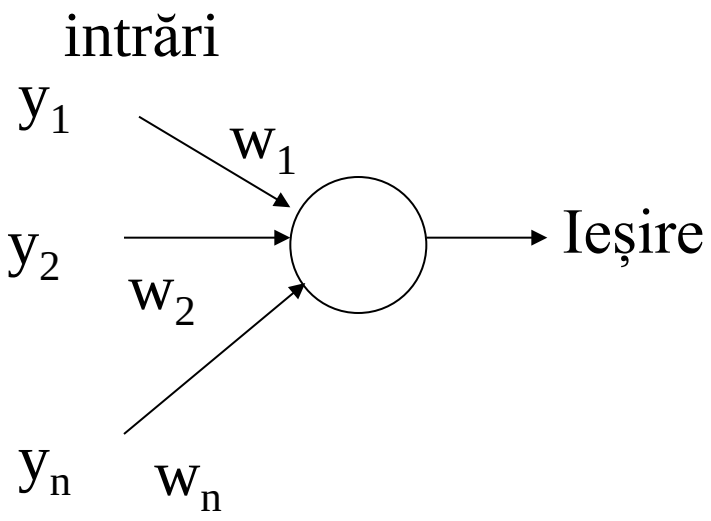
Rețele neuronale artificiale





w_1,w_2, ...:
Ponderi numerice atașate conexiunilor

Rețele neuronale artificiale



w_1,w_2, ...:
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)

Notații:

semnale de intrare: $y_1,y_2,\ldots,y_n$

ponderi sinaptice: $w_1,w_2,\ldots,w_n$

(modelează permeabilitatea sinaptică)

prag: b (sau w_0)

(modelează pragul de activare al neuronului)

ieșire: y

Obs: Toate valorile sunt numere reale

Retele neuronale artificiale

Retea neuronala artificiala = set de unitati functionale interconectate

Componentele unei retele neuronale:

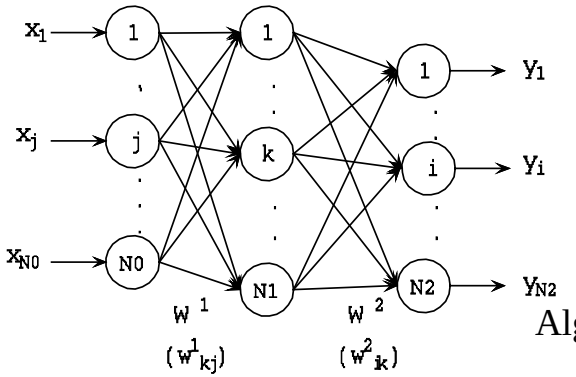
- Arhitectura:
  - Modul de plasare si interconectare a unitatilor functionale
  - Definita de un graf suport
- Functionare:
  - Modul de calcul a vectorului de iesire pornind de la vectorul de intrare
- Antrenare:
  - Determinarea parametrilor retelei pornind de la datele din setul de antrenare

Retele neuronale artificiale

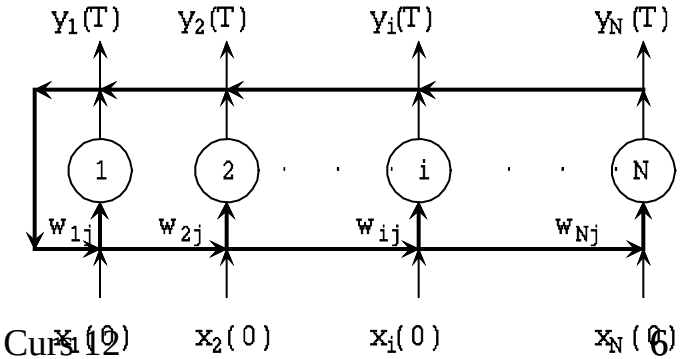
Tipuri principale de arhitecturi:

- Uni-directionale (feed-forward):
  - Graful suport nu contine cicluri (unitatile functionale sunt plasate pe nivele)
  - Vectorul de iesire se determina prin calcul direct din vectorul de intrare
- Recurente:
  - Graful suport contine cicluri
  - Vectorul de iesire se determina printr-un proces iterativ (simularea unui sistem dinamic)

Retea feed-forward



Retea recurenta (total interconectata)



Retele neuronale artificiale

Metode de invatare:

- Supervizata
  - Se cunosc exemple de antrenare de forma (date intrare, raspuns corect)
  - Scop invatare: estimarea parametrilor care minimizeaza eroarea (diferente intre raspunsurile corecte si cele produse de catre retea)

- Nesupervizata
  - Se cunosc doar date de intrare
  - Scop invatare: estimarea parametrilor modelului pe baza proprietatilor statistice ale datelor de intrare

Retele neuronale artificiale

Aplicatii:

- Clasificare / recunoastere
- Estimare
- Predictie
- Grupare
- Asociere

Aplicatii

Exemplu 1: identificarea speciei din care face parte o floare de iris

- Atribute: lungime sepale / petale, lățime petale/ sepale
- Clase: Iris setosa, Iris versicolor, Iris virginica

Aplicatii

Exemplu 1: identificarea speciei din care face parte o floare de iris

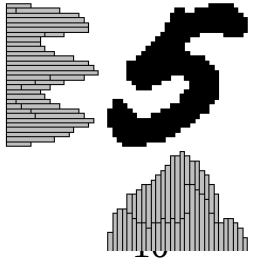
- Atribute: lungime sepale / petale, lățime petale/ sepale
- Clase: Iris setosa, Iris versicolor, Iris virginica

Exemplu 2: recunoașterea caracterelor

- Atribute: caracteristici statistice, caracteristici geometrice
- Clase: corespund setului de caractere care urmeaza a fi recunoscute

$\Rightarrow$ Probleme de clasificare = asociere

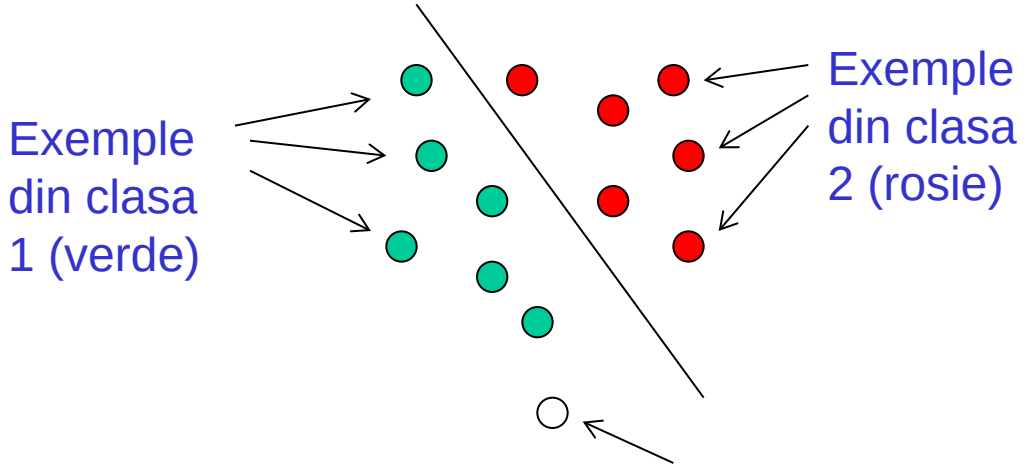
între vectori de atribute și indici de clase



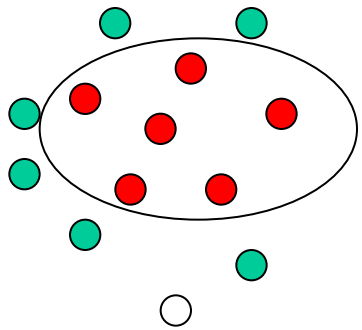
(Du Trier et al; Feature extraction methods for character Recognition. A Survey. Pattern Recognition, 1996)

Aplicatii

- Clasificare:
  - Problema: identificarea clasei căreia îi aparține un obiect descris printr-un set de atribute;
  - Se cunosc: exemple de clasificare corectă, numărul claselor și etichetele acestora (clasificare supervizată)
  - Grad dificultate: de la scăzut (clase clar delimitate) la ridicat (clase delimitate de suprafețe cu structură complexă)



Problemă liniar separabilă



Problemă neliniar separabilă

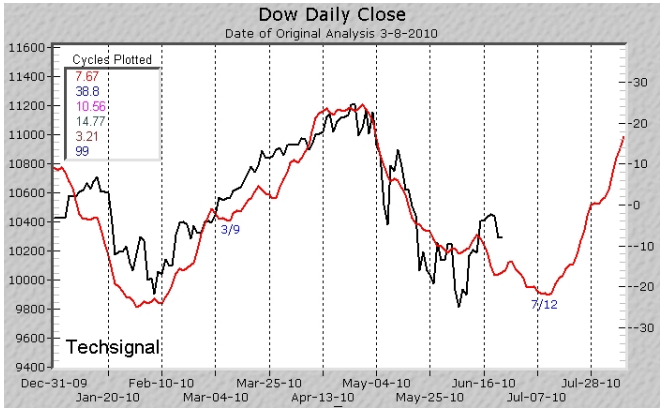
Aplicatii

- Estimarea prețului unei case cunoscând:
  - Suprafața totală,
  - Numărul de camere,
  - Dimensiunea grădinii,
  - Zona, etc

  => problemă de aproximare

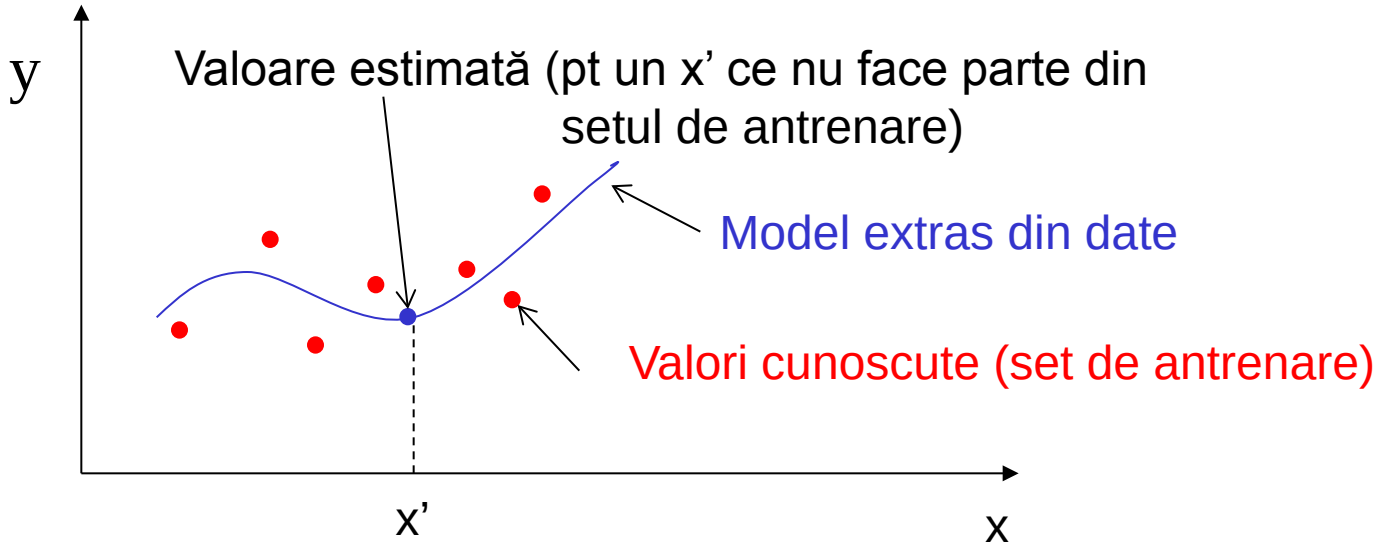
- Estimarea numărului de clienți ai unui serviciu web cunoscând evoluția acestora de-a lungul unei perioade de timp sau estimarea prețului unor active financiare ...

  => problemă de predicție=
  asociere între valori anterioare si valoarea sau tendința următoare



Aplicatii

- Regresie (aproximare, predicție):
  - Problema: determinarea relației dintre două sau mai multe mărimi (aproximarea unei funcții pornind de la cunoașterea valorilor pentru un set de argumente)
  - Se cunosc: perechi de valori corespondente sau succesiune de valori anterioare (set de antrenare)



Retele neuronale si optimizare

Reformulare ca problemă de optimizare:

Pornind de la un set de date (X^i,Y^i), X^i din R^N iar Y^i din R^M se caută o funcție F:R^N -> R^M , F=(F_1,...,F_M), care să minimizeze un criteriu de eroare (de exemplu, suma pătratelor erorilor corespunzătoare exemplelor din set):

$$\sum_i \sum_{j=1}^{M} (y_j^i - F_j(X^i))^2$$

Scopul urmărit este să se determine funcția F (având componentele (F_1,...,F_M)) care aproximează cât mai bine datele din set (explică cât mai bine dependența dintre Y și X)

Intrebări:

- Ce forma are F ?
- Cum se determină parametrii corespunzători lui F ?

Proiectarea retelelor neuronale

Poate fi rezolvată o astfel de problemă utilizând rețele neuronale ?

Da, dpdv teoretic rețelele neuronale sunt "aproximatori universali" [Hornik, 1985]:

" Orice funcție continuă poate fi aproximată de o rețea feed-forward având cel puțin un nivel ascuns. Acuratețea aproximării depinde de numărul de unități ascunse"

- Forma funcției este determinată de arhitectura rețelei și de proprietățile funcției de activare (transfer)
- Parametrii sunt ponderile asociate conexiunilor dintre unități
- Pentru unele probleme numărul de unități ascunse necesare pentru a obține o reprezentare acceptabilă poate fi inacceptabil de mare; acest fapt este unul dintre motivele dezvoltării arhitecturilor adânci (cu multe nivele)

Proiectarea retelelor neuronale

Etape:

- Alegerea arhitecturii:
  - număr de nivele
  - număr de unități pe fiecare nivel
  - mod de interconectare (topologie)
  - funcții de activare

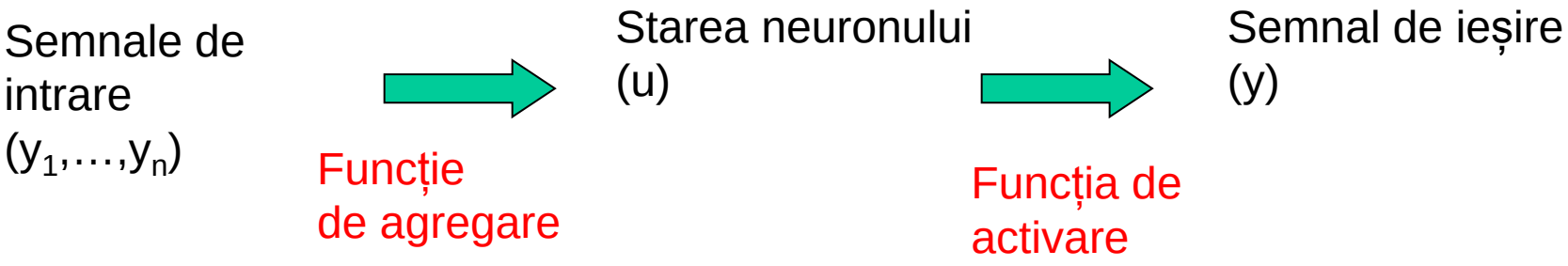
- Antrenarea: determinarea valorilor ponderilor pornind de la setul de antrenare folosind un algoritm de învățare

- Validarea rețelei: analiza comportării rețelei pentru date ce nu fac parte din setul de antrenare

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 aplicand o funcție de activare (transfer)
  - corespunde generării impulsurilor de-a lungul axonului



Unități funcționale

Exemple de funcții clasice de agregare

Suma ponderată

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

Distanța euclidiană

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

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

Neuron multiplicativ

$$u = \sum_{j=1}^{n} w_j y_j + \sum_{i,j=1}^{n} w_{ij} y_i y_j + \ldots$$

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:

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

Unități funcționale

Exemple de funcții de activare (transfer)

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

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

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

$$f(u) = u$$ liniară

$$f(u) = \max\{0, u\}$$ Semi-liniară (rectified linear unit - ReLU)

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

Unități funcționale

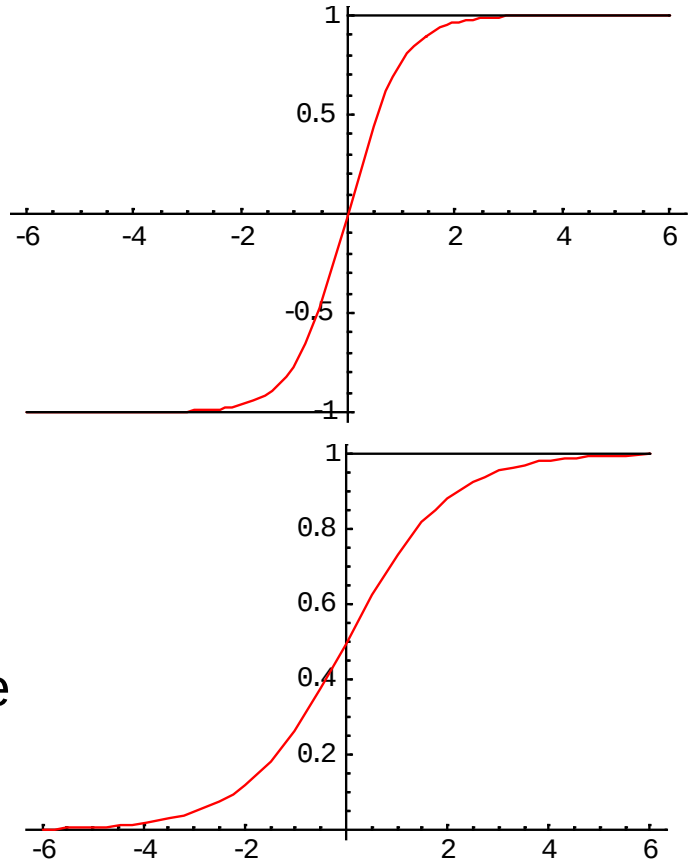
Exemple de funcții de activare (funcții sigmoidale)

(tangenta hiperbolică)

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

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

(logistică)



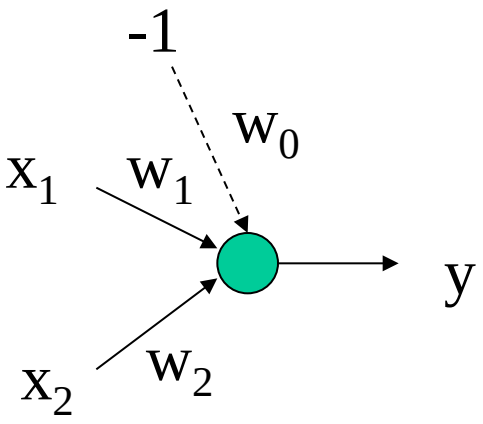
Observație: uneori se folosește un parametru numit pantă (slope) care multiplică argumentul funcției de activare: y=f(p*u)

Unități funcționale

- Ce se poate face cu un singur neuron ?

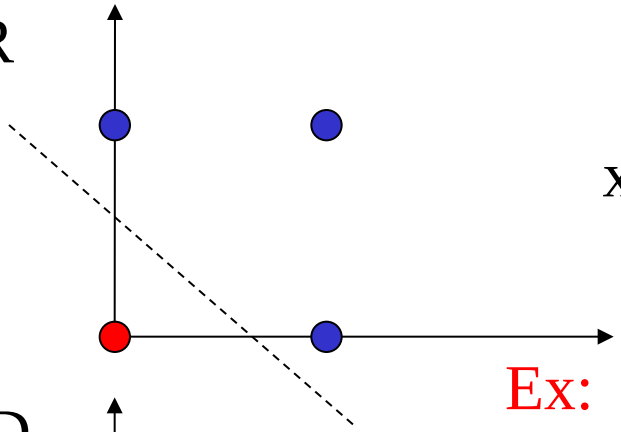
  Se pot rezolva probleme simple de clasificare

(ex: se pot reprezenta funcții booleene simple)



	0	1
0	0	1
1	1	1

OR

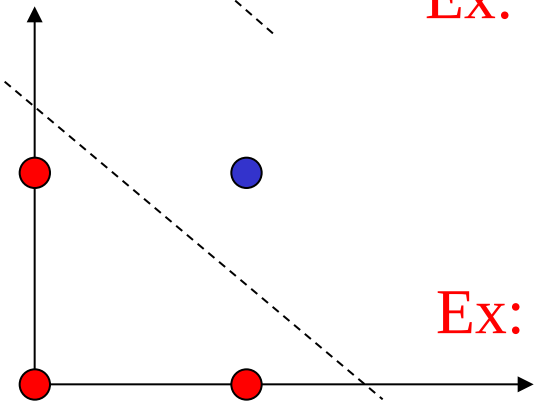


$y=H(w_1x_1+w_2x_2-w_0)$

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

	0	1
0	0	0
1	0	1

AND

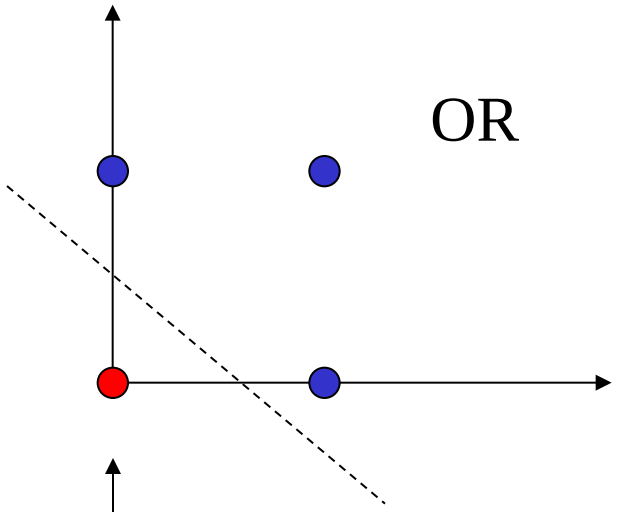


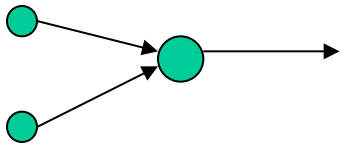
$y=H(w_1x_1+w_2x_2-w_0)$

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

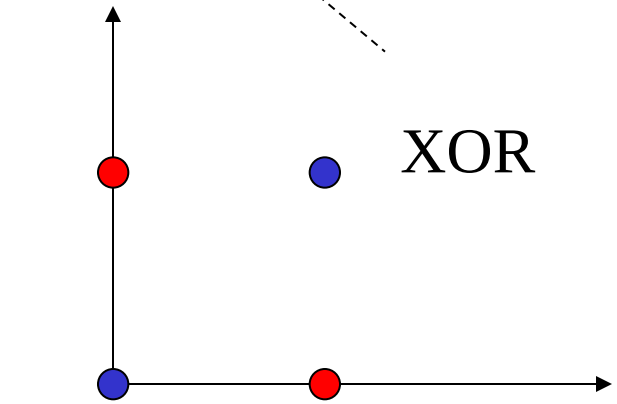
Liniar/neliniar separabilitate

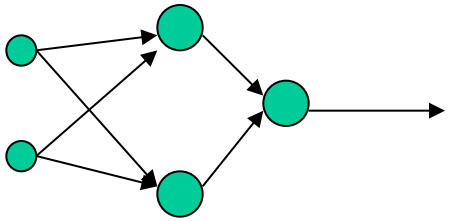
Reprezentarea unor funcții booleene: $f:\{0,1\}^N \rightarrow \{0,1\}$





Problema liniar separabilă – e suficientă o rețea uninivel

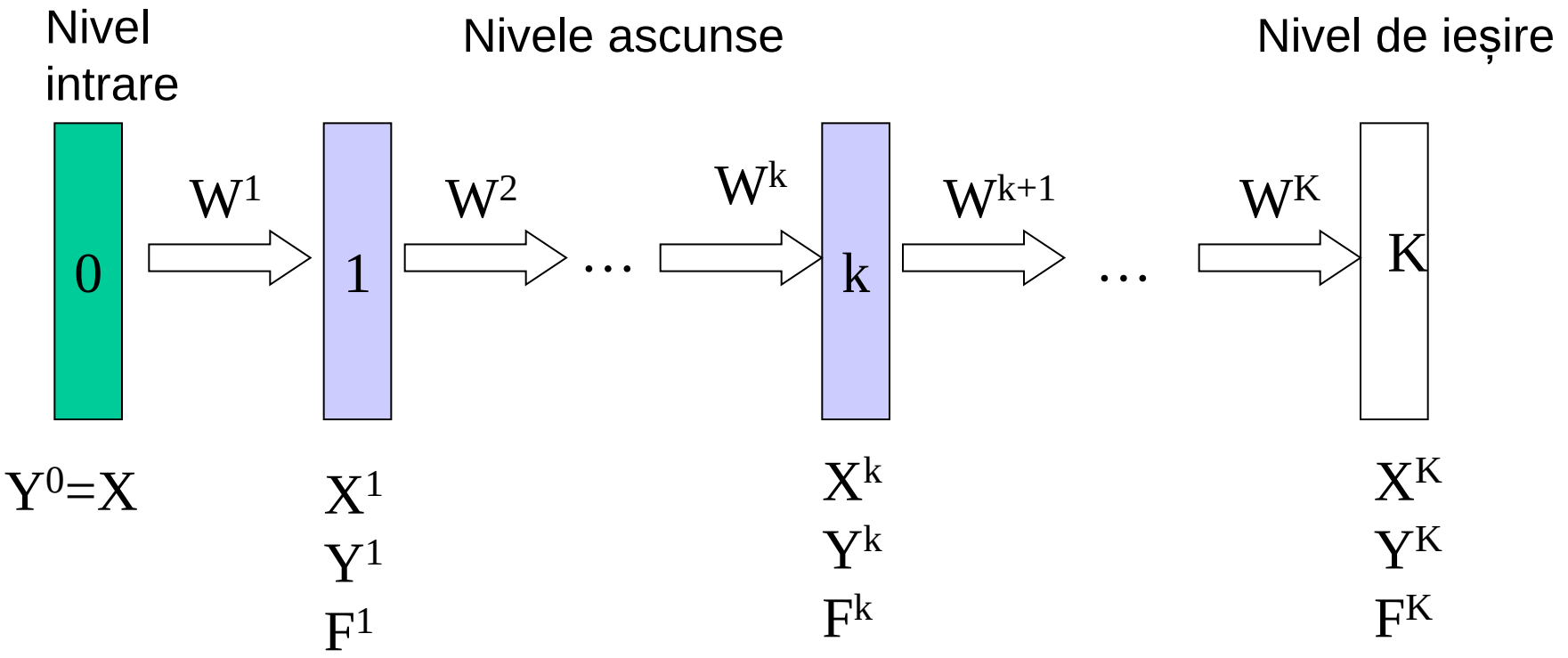




Problema neliniar separabilă – e necesară o rețea multinivel (care contine un nivel ascuns)

Rețele feedforward - arhitectura

Arhitectura și funcționare (K nivele funcționale)



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

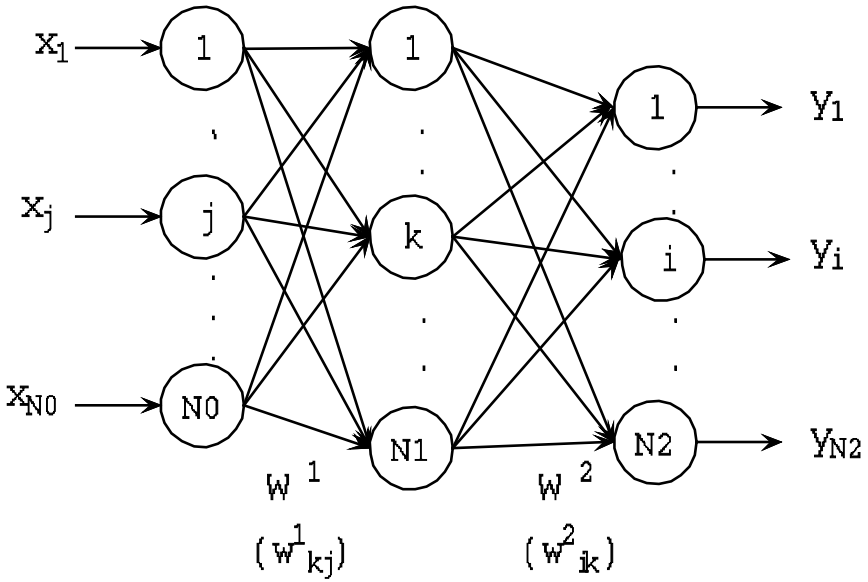
Calcul vector de ieșire: $Y=F^K(W^K*F^{K-1}(W^{K-1}*F^{K-2}(.....F^1(W^1*X))))$

Rețele feedforward – funcționare

Arhitectura și funcționare

(caz particular: un nivel ascuns)

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



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

Obs:

- în mod tradițional se lucrează cu unul sau două nivele ascunse
- În ultimii ani au devenit din ce în ce mai folosite rețelele cu număr mare de nivele (Deep Neural Networks) folosite în particular pentru recunoașterea imaginilor și a vorbirii (http://deeplearning.net)

Rețele feedforward - antrenare

Antrenare (supervizată):

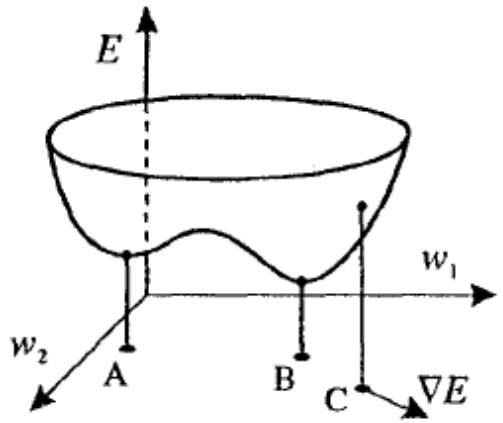
- Set de antrenare: $\{(x^1,d^1), \ldots, (x^L,d^L)\}$

  (x^l = vector intrare, d^l = vector de ieșire corect)

- Funcție de eroare (suma pătratelor erorilor):

$$E(W) = \frac{1}{2}\sum_{l=1}^{L}\sum_{i=1}^{N2}\left(d_i^l - f_2\left(\sum_{k=0}^{N1} w_{ik} f_1\left(\sum_{j=0}^{N0} w_{kj} x_j^l\right)\right)\right)^2$$

- Scopul antrenării: minimizarea funcției de eroare
- Metoda de minimizare: metoda gradientului



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

Functia de eroare:

$$E(W) = \frac{1}{2} \sum_{l=1}^{L} \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\sum_{k=0}^{N1} w_{ik} f_1 \underbrace{\left(\sum_{j=0}^{N0} w_{kj} x_j^l \right)}_{x_k} \right) \right)^2$$

y_k

x_i

y_i

Notații:

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

Rețele feedforward - antrenare

- Calculul derivatelor partiale

$$E(W) = \frac{1}{2}\sum_{l=1}^{L}\sum_{i=1}^{N2}\left(d_i^l - f_2\left(\sum_{k=0}^{N1} w_{ik}\, f_1\left(\underbrace{\sum_{j=0}^{N0} w_{kj} x_j^l}_{x_k} \right) \right) \right)^2$$

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

$$\frac{\partial E_l(W)}{\partial w_{kj}} = -\sum_{i=1}^{N2} w_{ik}(d_i^l - y_i)\, f_2'(x_i)\, f_1'(x_k) x_j^l = -\left(f_1'(x_k) \sum_{i=1}^{N2} w_{ik}\delta_i^l \right) x_j = -\delta_k^l x_j^l$$

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 in rețea a erorii de la nivelul de ieșire)

Rețele feedforward - antrenare

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

$$\frac{\partial E_l(W)}{\partial w_{kj}} = -\sum_{i=1}^{N2} w_{ik}(d_i^l - y_i) f_2^{'}(x_i) f_1^{'}(x_k) x_j^l = -\left(f_1^{'}(x_k) \sum_{i=1}^{N2} w_{ik} \delta_i^l \right) x_j = -\delta_k^l x_j^l$$

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

Logistica: f'(x)=f(x)(1-f(x)) => f'(x)=y(1-y)

Tanh: f'(x)=1-f(x)^2 => f'(x)=1-y^2

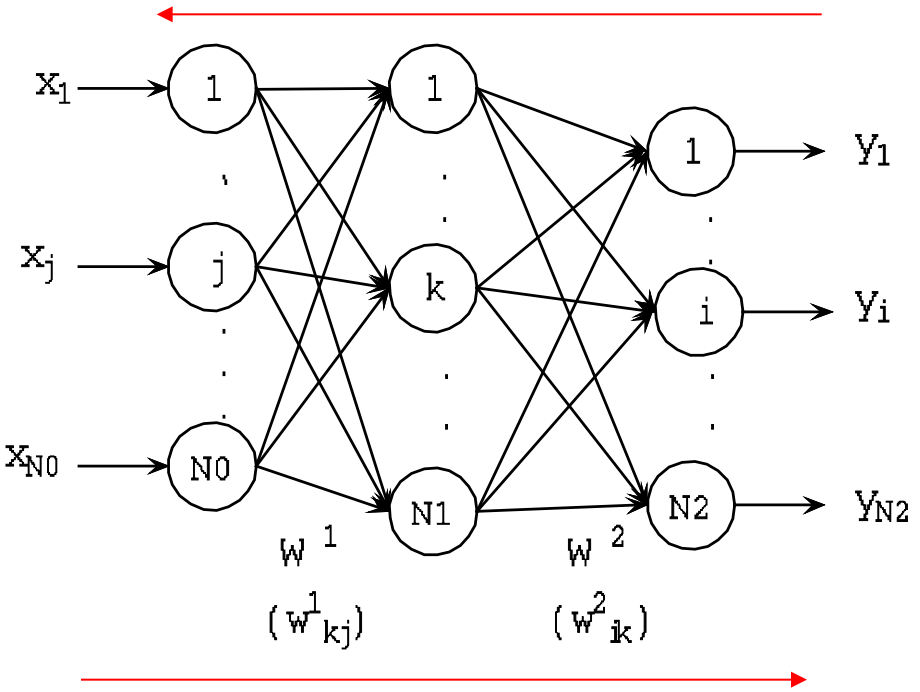
Algoritmul BackPropagation

Idee:

Pentru fiecare exemplu din setul de antrenare:

- 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

REPEAT

epoca

- FOR l=1,L DO
  - etapa FORWARD
  - etapa BACKWARD
  - ajustare ponderi
- Recalcularea erorii

UNTIL <condiție oprire>

Obs.

- Valorile inițiale se aleg aleator in [0,1] sau [-1,1]
- La ajustare se ține cont de rata de învățare
- 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

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

$p = 0$

REPEAT

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 * /

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

ENDFOR

Algoritmul BackPropagation

/ * Calculul erorii * /

$E = 0$

FOR $l = 1, L$ DO

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

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

/ * 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)

$w_{kj}^1 = rand(-1,1), w_{ik}^2 = rand(-1,1),\ i = 1..N2, k = 0..N1,\ j = 0..N0$

$p = 0$

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 * /

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

ENDFOR

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

Algoritmul BackPropagation

/ * Calculul erorii * /

$E = 0$

FOR $l = 1, L$ DO

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

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

/ * 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 $E < E^*$

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: Eroarea descreşte 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)
  - Daca eroarea descrește semnificativ atunci rata de învățare poate fi mărită (ajustările sunt acceptate)
  - In toate celelalte cazuri rata de învățare rămâne neschimbată

$$E(p) > (1+\gamma)E(p-1) \Rightarrow \eta(p) = a\,\eta(p-1), 0 < a < 1$$
$$E(p) < (1-\gamma)E(p-1) \Rightarrow \eta(p) = b\,\eta(p-1), 1 < b < 2$$
$$(1-\gamma)E(p-1) \le E(p) \le (1+\gamma)E(p-1) \Rightarrow \eta(p) = \eta(p-1)$$

Exemplu: γ=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

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

$$\alpha = 0.9$$

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

$E : R^n \to R,\ w \in R^n$ (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 ec :

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

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 patratică)

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

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

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

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

Termen de perturbare care elimina cazurile singulare (cand matricea este neinversabila)

Avantaje:

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

Probleme ale algoritmului BP

P2: 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):

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

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

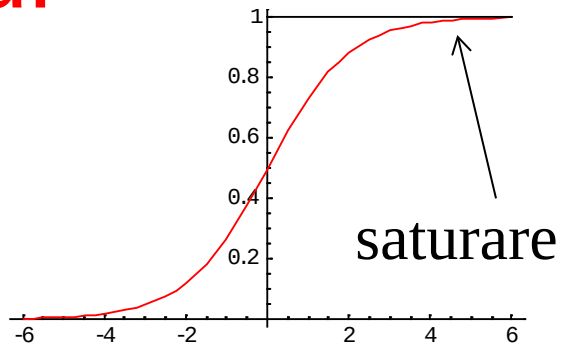
$$\Delta_{ij} := \text{valori aleatoare}$$

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

Obs:

- Ajustările sunt de regulă generate în conformitate cu repartiția normală de medie 0 și dispersie adaptivă
- Daca 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 garanteaza obținerea minimului

Probleme ale algoritmului BP



- Pb 3: 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 sigmoidale 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
  - Se utilizeaza doar semnele derivatelor nu și valorile lor
  - Se normalizează datele de intrare (valori în apropierea intervalului (-1,1)

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:

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

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 4: Supraantrenare și capacitate limitată de generalizare

Cauze:

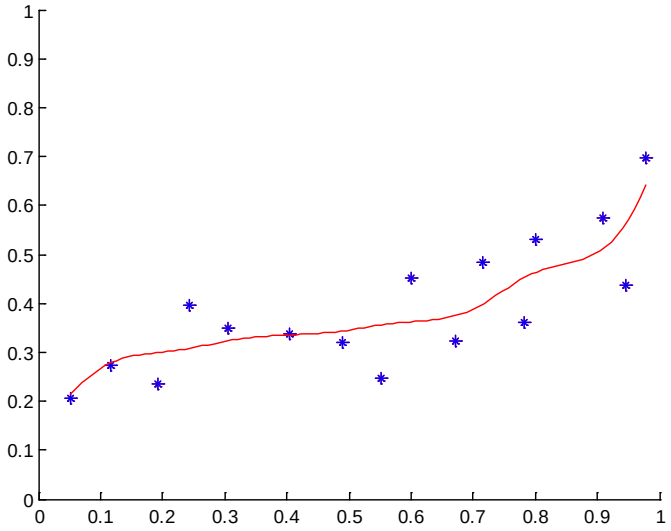
- Arhitectura rețelei (numărul de unitați ascunse)
  - Un număr prea mare de unități ascunse poate provoca supraantrenare (rețeaua extrage nu doar informațiile utile din setul de antrenare ci și zgomotul)
- Dimensiunea setului de antrenare
  - Prea puține exemple nu permit antrenarea și asigurarea capacității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

Soluții:

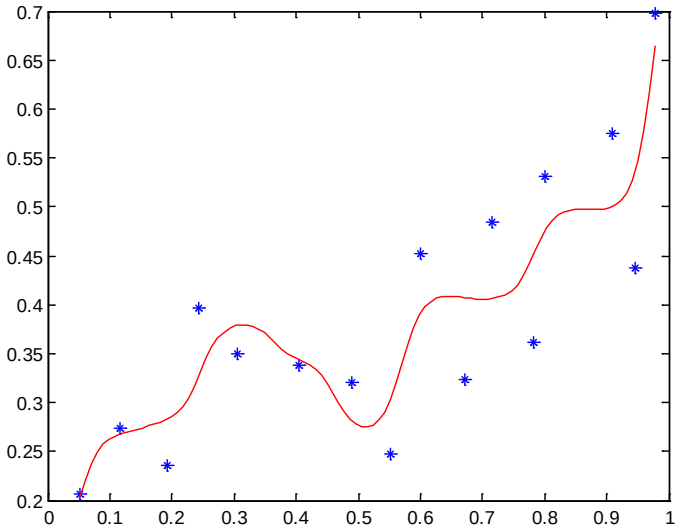
- Modificarea dinamică a arhitecturii
- Criteriul de oprire se bazează nu pe eroarea calculată pentru setul de antrenare ci pentru un set de validare

Probleme ale algoritmului BP

Supraantrenare – influența numărului de unități ascunse



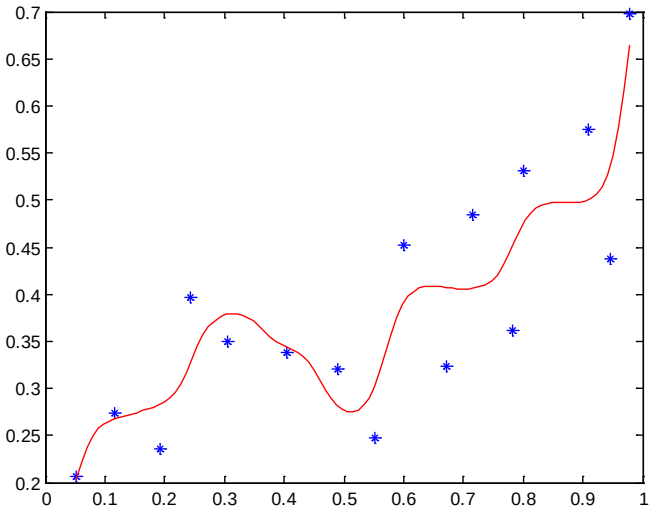
5 unități ascunse



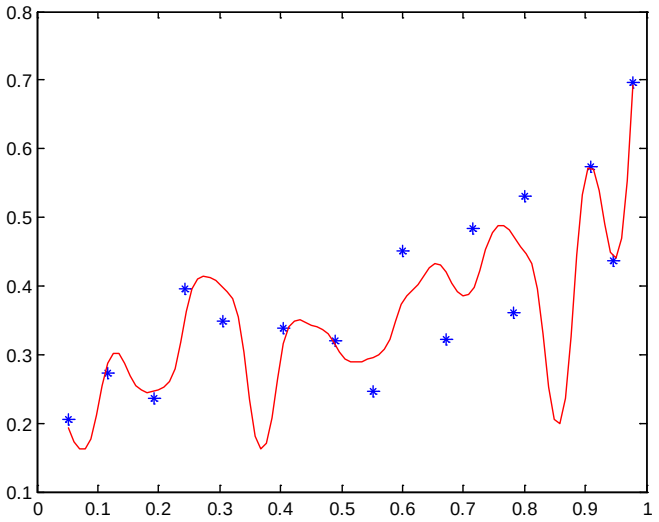
10 unități ascunse

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

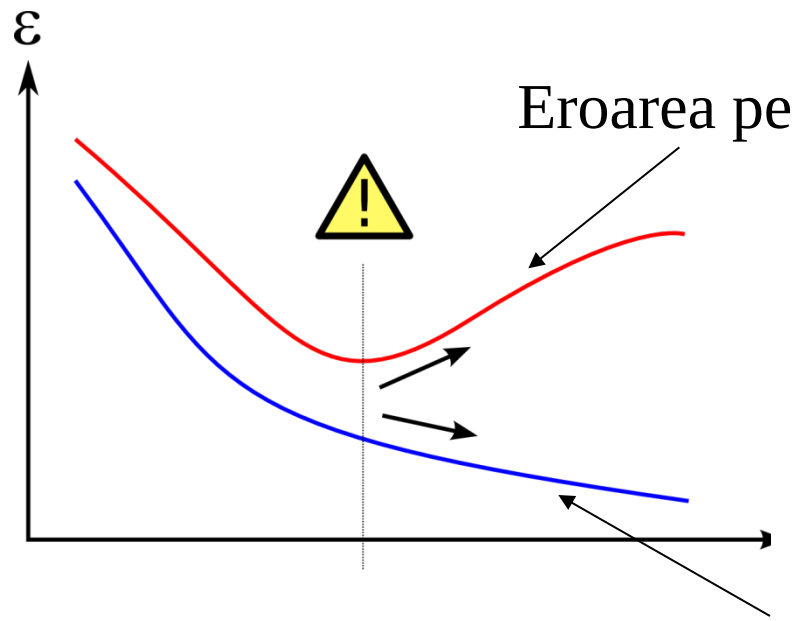
1: S=(S1,S2, ...,Sm)

2: S=(S1,S2, ...,Sm)

....

m: S=(S1,S2, ...,Sm)

Probleme ale algoritmului BP



Rețele neuronale recurente

- Arhitectura
  - Se caracterizează prin prezența conexiunilor inverse (graful suport conține cicluri)
  - In funcție de densitatea conexiunilor inverse există:
    - Rețele total recurente (modelul Hopfield)
    - Rețele parțial recurente:
      - Cu unități contextuale (modelul Elman)
      - Celulare (modelul Chua-Yang)
- Aplicabilitate
  - Memorare asociativă
  - Rezolvarea problemelor de optimizare combinatorială
  - Predicție în serii temporale
  - Prelucrarea imaginilor
  - Modelarea sistemelor dinamice și a fenomenelor haotice

Rețele de tip Hopfield

Arhitectura: N unități total interconectate

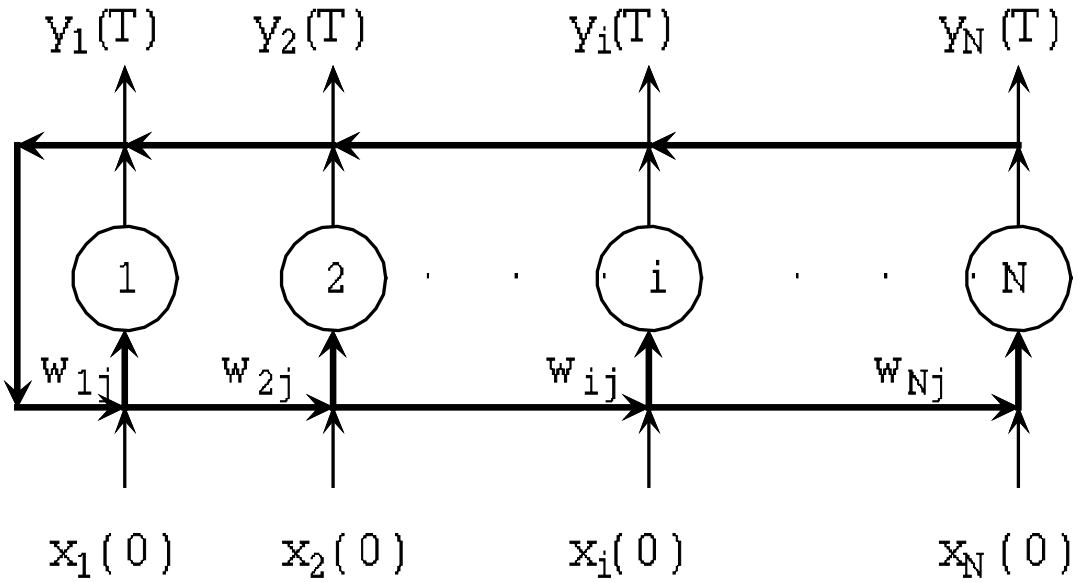
Funcții de transfer:

Signum/Heaviside

Logistica/Tanh

Parametrii:

matricea cu ponderi a conexiunilor



Notatii: $x_i(t)$ – potențialul (starea) neuronului i la momentul t

$y_i(t)=f(x_i(t))$ – semnalul de ieșire produs de neuronul i la momentul t

$I_i(t)$ – semnalul de intrare primit din partea mediului

w_{ij} – ponderea conexiunii dintre neuronul j și neuronul i

Rețele de tip Hopfield

Funcționare: - semnalul de ieșire este rezultatul unui proces dinamic

- rețelele de tip Hopfield pot fi asimilate unor sisteme dinamice a căror stare se modifică în timp

Starea rețelei:

- vectorul stărilor neuronilor $X(t)=(x_1(t), \ldots, x_N(t))$

sau

- vectorul semnalelor de ieșire $Y(t)=(y_1(t),\ldots,y_N(t)$

Descrierea dinamicii

- Timp discret – relații de recurență (ecuații cu diferențe)
- Timp continuu – ecuații diferențiale

Rețele de tip Hopfield

Funcționare în timp discret:

starea rețelei la momentul t+1 este descrisă în funcție de starea rețelei la momentul t

Starea rețelei: Y(t)

Variante:

- Asincronă: un singur neuron își poate modifica starea la un moment dat
- Sincronă: toți neuronii își schimbă starea simultan

Răspunsul rețelei: starea în care se stabilizează rețeaua

Retele de tip Hopfield

Varianta asincronă:

$$y_{i*}(t+1) = f\left(\sum_{j=1}^{N} w_{i*j} y_j(t) + I_{i*}(t)\right)$$

$$y_i(t+1) = y_i(t), \quad i \neq i*$$

Alegerea lui i*:

- prin parcurgerea sistematică a mulțimii {1,2,…,N}
- aleator (dar astfel încât la fiecare etapă să se modifice starea fiecărui neuron)

Simularea rețelei:

- alegerea unei stări inițiale (în funcție de problema de rezolvat)
- calculul stării următoare până când rețeaua se stabilizează

(distanța dintre două stări succesive este mai mică decât o valoare prestabilită, ε)

Retele de tip Hopfield

Varianta asincronă:

$$y_{i*}(t+1) = f\left(\sum_{j=1}^{N} w_{i*j} y_j(t) + I_{i*}(t)\right)$$

$$y_i(t+1) = y_i(t), \quad i \neq i*$$

Alegerea lui i*:

- prin parcurgerea sistematică a mulțimii {1,2,…,N}
- aleator (dar astfel încât la fiecare etapă să se modifice starea fiecărui neuron)

Simularea rețelei:

- alegerea unei stări inițiale (în funcție de problema de rezolvat)
- calculul stării următoare până când rețeaua se stabilizează

(distanța dintre două stări succesive este mai mică decât o valoare prestabilită, ε)

Retele de tip Hopfield

Varianta sincronă:

$$y_i(t+1) = f\left(\sum_{j=1}^{N} w_{ij} y_j(t) + I_i(t)\right),\ i = \overline{1, N}$$

In simulări se utilizează fie funcții de transfer discontinue fie continue

Algoritm de funcționare:

Stabilire stare inițială

REPEAT

se calculează noua stare pornind de la starea curentă

UNTIL < diferența dintre starea curenta și cea anterioară este suficient de mică >

Rețele de tip Hopfield

Funcționare in timp continuu:

$$\frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^{N} w_{ij} f(x_j(t)) + I_i(t),\ i = \overline{1, N}$$

Simularea rețelei: rezolvarea numerică a sistemului de ecuații pornind de la o valoare inițială $x_i(0)$

Exemplu: metoda lui Euler explicită

$$\frac{x_i(t+h) - x_i(t)}{h} \cong -x_i(t) + \sum_{j=1}^{N} w_{ij} f(x_j(t)) + I_i(t),\ i = \overline{1, N}$$

$$x_i(t+h) \cong (1-h)x_i(t) + h\sum_{j=1}^{N} w_{ij} f(x_j(t)) + I_i(t),\ i = \overline{1, N}$$

Semnal de intrare constant :

$$x_i^{nou} \cong (1-h)x_i^{vechi} + h(\sum_{j=1}^{N} w_{ij} f(x_j^{vechi}) + I_i),\ i = \overline{1, N}$$

Proprietăți de stabilitate

Comportări posibile ale unei rețele:

- X(t) tinde către o stare staționară X* (punct fix al dinamicii rețelei)
- X(t) oscilează între două sau mai multe stări posibile
- X(t) are o comportare haotică sau ||X(t)|| crește pe măsură ce t crește

Comportări utile din punct de vedere practic:

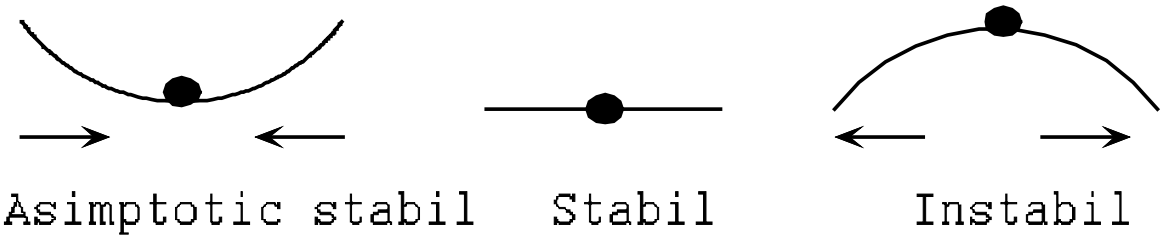
- Rețeaua tinde către o stare staționară
  - Există o mulțime de stări staționare: memorare asociativă
  - Stare staționară unică: rezolvarea pb. de optimizare

- Rețeaua are comportare periodică
  - Modelarea seriilor temporale

Obs. Cea mai convenabilă variantă este cea în care rețeaua tinde către o stare staționară stabilă (perturbații mici în condițiile inițiale nu afectează semnificativ soluția)

Proprietăți de stabilitate

Tipuri de echilibru (intuitiv)



Formalizare:

$$\frac{dX(t)}{dt} = F(X(t)), \quad X(0) = X_0$$

$$F(X^*) = 0$$

X* este soluție staționară asimptotic stabilă (în raport cu condițiile inițiale) dacă:

- stabilă
- atractivă

Proprietăți de stabilitate

Stabilitate:

X* este stabilă dacă pentru orice ε>0 există δ(ε) > 0 cu proprietățile: $||X_0-X^*||< \delta(\varepsilon)$ implică $||X(t;X_0)-X^*||< \varepsilon$

Atractivitate:

X* este atractivă dacă există δ > 0 cu proprietatea: $||X_0-X^*||< \delta$ implică $X(t;X_0)->X^*$

Pentru a studia proprietatea de (asimptotic) stabilitate a soluțiilor în teoria sistemelor dinamice se poate folosi metoda funcției Liapunov

Proprietati de stabilitate

Funcție Liapunov:

$$V : R^N \to R, \text{ marginita inferior}$$

$$\frac{dV(X(t))}{dt} < 0, \text{ pentru orice t} > 0$$

- Funcție mărginită inferior care descrește de-a lungul soluțiilor => soluția trebuie să se stabilizeze
- Dacă unui sistem i se poate asocia o funcție Liapunov atunci soluțiile staționare ale acestuia sunt asimptotic stabile
- Funcțiile Liapunov sunt similare funcțiilor de energie din fizică (sistemele fizice tind în mod natural catre stări de energie minimă)
- Stările pentru care funcția Liapunov este minimă corespund stărilor de energie minimă, care sunt stări stabile
- Daca rețeaua neuronală satisface anumite proprietăți atunci există functie Liapunov asociată

Proprietăți de stabilitate

Rezultate de stabilitate pentru rețelele neuronale (cazul continuu):

Dacă:

- matricea ponderilor este simetrică ($w_{ij}=w_{ji}$)
- funcția de transfer este strict crescătoare ($f'(u)>0$)
- semnalul de intrare este constant ($I(t)=I$)

Atunci toate soluțiile staționare asociate rețelei sunt asimptotic stabile

Funcția Liapunov asociată:

$$V(x_1,\ldots,x_N)=-\frac{1}{2}\sum_{i,j=1}^{N} w_{ij} f(x_i) f(x_j)-\sum_{i=1}^{N} f(x_i) I_i+\sum_{i=1}^{N}\int_0^{f(x_i)} f^{-1}(z)dz$$

Proprietăți de stabilitate

Rezultate de stabilitate pentru rețelele neuronale (cazul discret - asincron):

Dacă:

- matricea ponderilor este simetrică ($w_{ij}=w_{ji}$)
- funcția de transfer este signum sau Heaviside
- semnalul de intrare este constant (I(t)=I)

Atunci toate soluțiile staționare asociate rețelei sunt asimptotic stabile

Functia Liapunov asociată

$$V(y_1,\ldots,y_N)=-\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}y_i y_j-\sum_{i=1}^{N} y_i I_i$$

Proprietăți de stabilitate

Rezultatul anterior sugerează că stările staționare (punctele fixe) ale dinamicii sistemului sunt stabile

Fiecarei stări staționare îi corespunde o regiune de atracție (dacă starea inițială a rețelei se află în regiunea de atracție a unei stări staționare atunci sistemul va tinde către starea staționară corespunzătoare

Această proprietate este utilă în cazul memoriilor asociative

In cazul dinamicii sincrone rezultatul nu mai este valabil, putându-se arăta în schimb că soluțiile periodice de periodă 2 (rețeaua oscilează între două stări) sunt stabile

Memorii asociative

Memorie = sistem de stocare și regăsire a informației

Memorie bazată pe adresă:

- Stocare localizată: toate “elementele” (biții) unei unități de informație (ex: o valoare numerică) sunt stocate în aceeași zonă de memorie identificabilă prin adresă
- Regăsire pe baza adresei

Memorie asociativă:

- Informația este stocată distribuit (nu mai exista conceptul de adresă)
- Regăsirea se realizează pe baza conținutului (se pornește de la un indiciu privind informația de regăsit – ex: imagine parțială sau alterată de zgomot)

Memorii asociative

Proprietăți:

- Robustețe (distrugerea unei componente nu impiedică regăsirea informației)

Modalități de implementare:

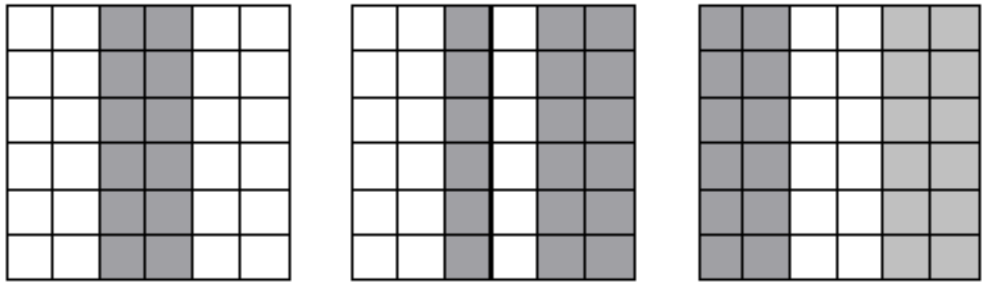
- Hardware:
  - Circuite electrice
  - Sisteme optice

- Software:
  - Rețea de tip Hopfield

Memorii asociative

Simularea software a memoriilor asociative:

- Informațiile sunt codificate binar: vectori cu elemente din {-1,1}
- Fiecarei componente a vectorului ii va corespunde o unitate funcțională în cadrul rețelei



Exemplu (a)

(-1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1, -1,-1,1,1,-1,-1)

Memorii asociative

Proiectarea rețelelor pentru simularea software a memoriilor asociative:

- Rețea total interconectata cu N unități de tip signum (N este dimensiunea șabloanelor ce urmează a fi memorate)

Stocarea informației:

- Stabilirea ponderilor (elementele matricii W) astfel încât șabloanele de memorat să devină puncte fixe (stări staționare) ale dinamicii rețelei

Regăsirea informației:

- Inițializarea stării rețelei cu indiciul de start și simularea dinamicii rețelei până la identificarea stării staționare corespunzătoare; starea staționară va reprezenta informația regăsită

Memorii asociative

Set de vectori de memorat: $\{X^1,\ldots,X^L\}$, X^l din $\{-1,1\}^N$

Metode de stocare:

- Regula Hebb
- Metoda pseudo-inversei (algoritmul Diedierich – Opper)

Regula Hebb:

- Se bazează pe principiul enuntat de D. Hebb: “permeabilitatea sinaptică a doi neuroni activați simultan crește”

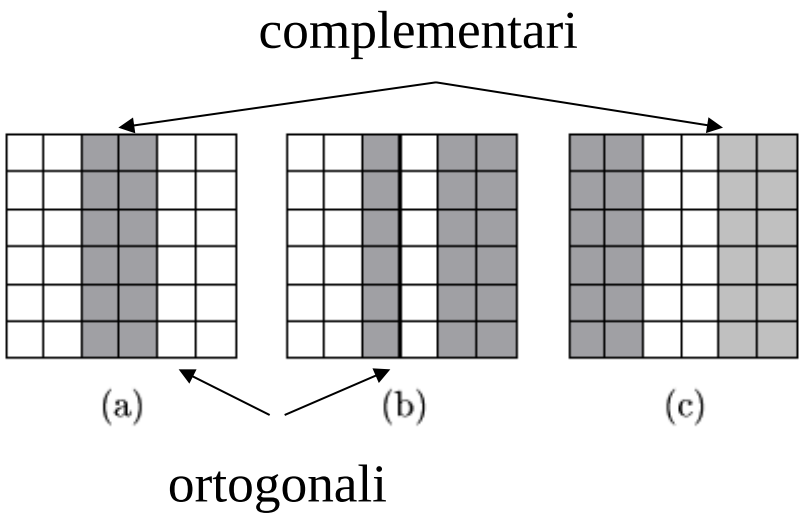
$$w_{ij} = \frac{1}{N}\sum_{l=1}^{L} x_i^l x_j^l$$

Memorii asociative

$$w_{ij} = \frac{1}{N} \sum_{l=1}^{L} x_i^l x_j^l$$

Proprietăți ale regulii Hebb:

- Dacă vectorii din setul de memorat sunt ortogonali doi câte doi (din punct de vedere statistic ar însemna că sunt necorelați) atunci toți sunt puncte fixe ale dinamicii rețelei
- O dată cu stocarea vectorului X se stochează și opusul său: -X
- O variantă care elimină condiția de ortogonalitate este cea bazată pe metoda pseudo-inversei



Memorii asociative

Metoda pseudo-inversei:

$$w_{ij} = \frac{1}{N} \sum_{l,k} x_i^l (Q^{-1})_{lk} x_j^l$$

$$Q_{lk} = \frac{1}{N} \sum_{i=1}^{N} x_i^l x_i^k$$

- Se poate arăta că dacă matricea Q este inversabilă atunci toate elementele lui $\{X^1,\ldots,X^L\}$ sunt puncte fixe ale dinamicii retelei
- Pentru a evita calculul inversei matricii Q se poate folosi un algoritm iterativ de ajustare a ponderilor

Memorii asociative

Algoritmul Diederich-Opper:

$t := 0$

Se iniţializează matricea $W(0)$ folosind regula Hebb

REPEAT

  FOR $l := 1, L$ DO

    $y_i^l := \sum_{j=1}^{N} w_{ij}(t) x_j^l,\ i = \overline{1, N}$

    $w_{ij}(t+1) := w_{ij}(t) + \frac{1}{N}(x_i^l - y_i^l) x_j^l,\ i = \overline{1, N},\ j = \overline{1, N}$

  $t := t + 1$

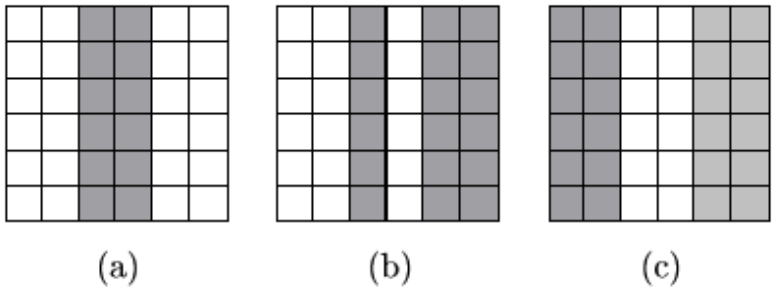
UNTIL $\|W(t) - W(t-1)\| < \epsilon$

Memorii asociative

Regăsirea informației:

- Inițializarea stării rețelei cu indiciul de la care se pornește
- Simularea funcționarii rețelei până la atingerea stării staționare

Vectori stocați



Vectori alterați de zgomot (25%, 50%, 75%)

Memorii asociative

Capacitatea de stocare:

- numărul de vectori care pot fi stocați și regăsiți (în limita unei anumite erori)
- Regăsire exactă: capacitate = N/(4lnN)
- Regăsire aproximativă (prob(eroare)=0.005): capacitate = 0.15*N

Atractori paraziți:

- Stări staționare ale rețelei care nu au fost explicit stocate ci au rezultat ca urmare a stocării altor vectori

Evitare atractori paraziți

- Modificarea metodei de stocare
- Introducerea de perturbații aleatoare in dinamica retelei

Formularea neuronală a unei probleme de optimizare combinatorială

- Prima abordare: Hopfield & Tank (1985)
  - Propun utilizarea unei rețele total interconectate pentru rezolvarea problemei comis-voiajorului
  - Ideea constă în a proiecta o rețea a cărei funcție de energie să fie similară funcției obiectiv a problemei de rezolvat (lungimea unui circuit hamiltonian) și in a o lăsa să "evolueze" în mod natural către starea de energie minima; aceasta ar reprezenta soluția problemei

Formularea neuronală a unei probleme de optimizare combinatorială

Problema de optimizare:

determină $(y_1,\ldots,y_N)$ cu proprietățile:

minimizează o funcție cost $C:R^N->R$

satisface restricții de forma $R_k\ (y_1,\ldots,y_N) =0$ cu R_k funcții nenegative

Etape:

- Transformarea problemei într-una de optimizare fără restricții (metoda penalizării)
- Reorganizarea funcției cost ca o funcție Liapunov
- Identificarea valorilor parametrilor și a semnalelor de intrare pornind de la forma funcției Liapunov
- Simularea rețelei

Formularea neuronală a unei probleme de optimizare combinatorială

Transformarea problemei într-una de optimizare fără restricții (metoda penalizării)

$$C^*(y_1,\ldots,y_N) = aC(y_1,\ldots,y_N) + \sum_{k=1}^{r} b_k R_k(y_1,\ldots,y_N)$$

$$a, b_k > 0$$

Valorile coeficienților a și b_k se aleg astfel încât să reflecte importanța relativă a funcției obiectiv și a restricțiilor

Formularea neuronală a unei probleme de optimizare combinatorială

Reorganizarea funcției cost ca o funcție Liapunov

$$C(y_1,....,y_N) = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}^{obj} y_i y_j - \sum_{i=1}^{N} I_i^{obj} y_i$$

$$R_k(y_1,....,y_N) = -\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}^{k} y_i y_j - \sum_{i=1}^{N} I_i^{k} y_i, \quad k = \overline{1,r}$$

Observatie: Ideea funcționeaza doar dacă funcția obiectiv și restricțiile sunt de grad cel mult 2

Formularea neuronală a unei probleme de optimizare combinatorială

Identificarea valorilor parametrilor si a semnalelor de intrare pornind de la forma funcției Liapunov

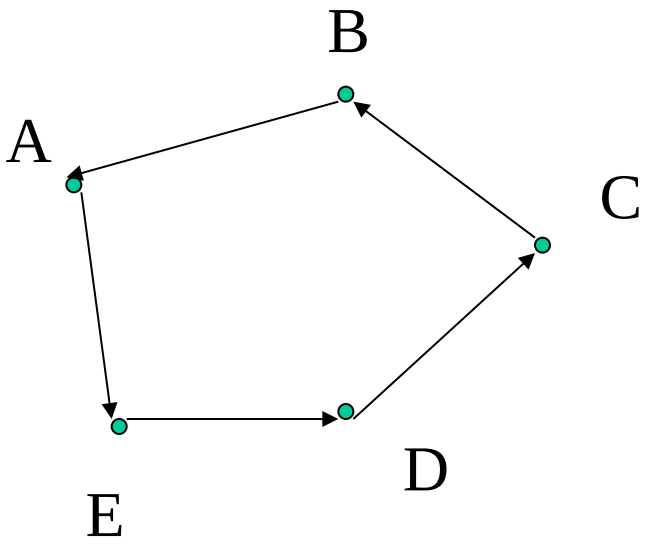
$$w_{ij} = a w_{ij}^{obj} + \sum_{k=1}^{r} b_k w_{ij}^k, \quad i, j = \overline{1, N}$$

$$I_i = a I_i^{obj} + \sum_{k=1}^{r} b_k I_i^k, \quad i = \overline{1, N}$$

Formularea neuronală a unei probleme de optimizare combinatorială

Exemplu: problema comis voiajorului

Se consideră un set de n orașe și se caută un traseu care trece o singură dată prin fiecare oraș și are costul minim



Formularea neuronală a unei probleme de optimizare combinatorială

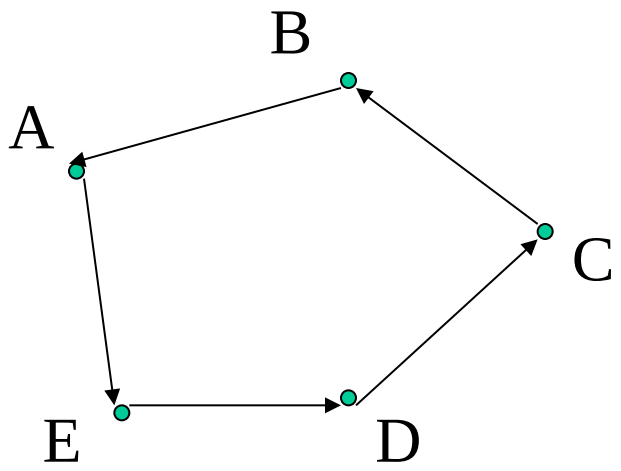
Proiectarea rețelei neuronale:

N=n*n neuroni

Starea neuronului (i,j):

1 - la etapa j este vizitat orașul i

0 - altfel



AEDCB

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

Formularea neuronala a unei probleme de optimizare combinatoriala

Descrierea problemei:

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

Restricții:

- la un moment dat este vizitat un singur oraș (fiecare coloană conține un singur 1)
- fiecare oraș este vizitat o singură dată (fiecare linie conține un singur 1)

Criteriu de optim:

costul traseului este minim

Formularea neuronală a unei probleme de optimizare combinatorială

Descrierea problemei:

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{n} y_{ij} - 1\right)^2 = 0$$

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{n} y_{ij} - 1\right)^2 = 0$$

$$C(Y) = \sum_{i=1}^{n}\sum_{k=1,k\neq i}^{n}\sum_{j=1}^{n} c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1})$$

Rescriere ca problemă de optimizare fără restricții

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{n} y_{ij} - 1\right)^2 = 0$$

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{n} y_{ij} - 1\right)^2 = 0$$

$$C(Y) = \sum_{i=1}^{n}\sum_{k=1,k\neq i}^{n}\sum_{j=1}^{n} c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1})$$

$$C^*(Y) = \frac{a}{2}\sum_{i=1}^{n}\sum_{k=1,k\neq i}^{n}\sum_{j=1}^{n} c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1}) +$$

$$\frac{b}{2}\left(\sum_{j=1}^{n}\left(\sum_{i=1}^{n} y_{ij} - 1\right)^2 + \sum_{i=1}^{n}\left(\sum_{j=1}^{n} y_{ij} - 1\right)^2\right)$$

Identificarea cu fctia Liapunov

$$V(Y) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} w_{ij,kl}\, y_{ij}\, y_{kl} - \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij} I_{ij}$$

$$C*(Y) = \frac{a}{2}\sum_{i=1}^{n}\sum_{k=1,k\neq i}^{n}\sum_{j=1}^{n} c_{ik}\, y_{ij}\,(y_{k,j-1} + y_{k,j+1}) +$$

$$\frac{b}{2}\left(\sum_{j=1}^{n}\left(\sum_{i=1}^{n} y_{ij} - 1\right)^2 + \sum_{i=1}^{n}\left(\sum_{j=1}^{n} y_{ij} - 1\right)^2\right)$$

Valorile parametrilor:

$$w_{ij,kl} = -ac_{ik}(\delta_{l,j-1} + \delta_{l,j+1}) - b(\delta_{ik} + \delta_{jl} + \delta_{ik}\delta_{jl})$$

$$w_{ij,ij} = 0$$

$$I_{ij} = 2b$$

Serii temporale

- Serie temporală = serie de date înregistrate la momente succesive de timp (corespunzătoare unor mărimi ce evoluează în timp)

- Exemple:
  - Evolutia cursului valutar
  - Evolutia pretului unei actiuni
  - Evolutia temperaturii
  - Semnale biologice (electrocardiograma –EKG)

- Scopul analizei unei serii temporale: efectuarea de predicții

Serii temporale

Predicția se bazează pe un model asociat seriei care exprimă dependența dintre valoarea viitoare și valoarea curentă, respectiv valori anterioare

$$x(t+1) = F(x(t), x(t-1), \ldots, x(t-p+1), \varphi_1, \varphi_2, \ldots, \varphi_s)$$

ordinul modelului

Parametrii ce modelează factorii externi

Serii temporale

Modelul asociat unei serii temporale poate fi:

- Liniar
- Neliniar

- Determinist
- Aleator

Exemplu: modelul auto-regresiv (AR(p))

$$x(t+1) = w_0 x(t) + w_1 x(t-1) + ... + w_{p-1} x(t-p+1) + zgomot$$

Model zgomot (variabila aleatoare din N(0,1))

Serii temporale

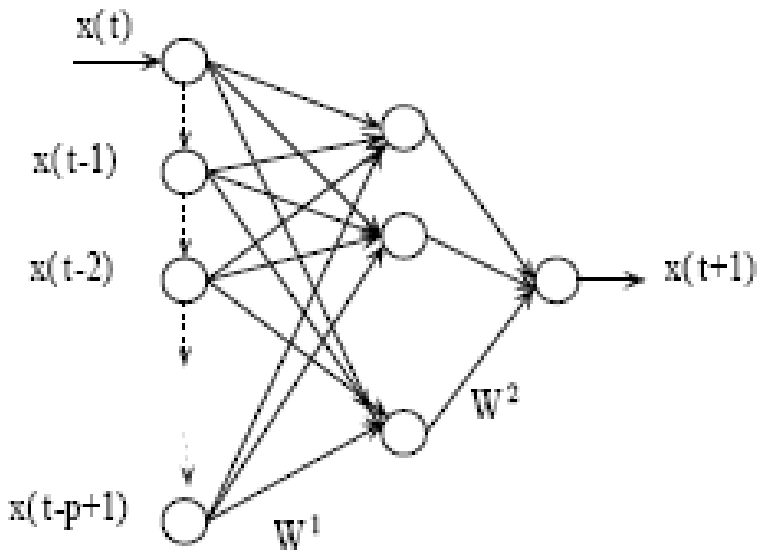
Modelare cu rețele neuronale

- Ordinul modelului este cunoscut
  - Rețea feedforward cu ferestre temporale (p unitati de intrare)

- Ordinul modelului este necunoscut
  - Retea cu unități contextuale (retea Elman)

Rețele cu ferestre temporale

Arhitectura:



Functionare:

$$y = \sum_{k=1}^{K} w_k^2 f(\sum_{j=0}^{p-1} w_{kj}^1 x(t-j))$$

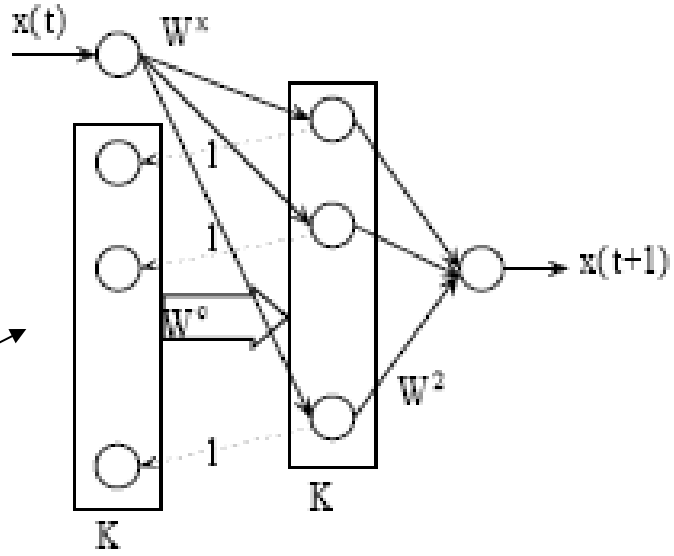
Rețele cu ferestre temporale

Antrenare:

- Set antrenare: $\{((x_l, x_{l-1}, \ldots, x_{l-p+1}), x_{l+1})\}_{l=1..L}$
- Algoritm de antrenare: BackPropagation
- Dezavantaj: presupune cunoașterea lui p

Rețele cu unități contextuale

Arhitectura (retea Elman):



Unități contextuale

Funcționare:

$$y = \sum_{k=1}^{K} w_k^2 h_k(t)$$

$$h_k(t) = f\left(w_k^x x(t) + \sum_{j=1}^{K} w_{kj}^c h_j(t-1)\right) \qquad h_j(0) = 0.$$

Obs: unitățile contextuale conțin copii ale stărilor unităților ascunse

Rețele cu unități contextuale

Antrenare

Set antrenare: {(x(1),x(2)),(x(2),x(3)),…(x(t-1),x(t))}

Ponderi:

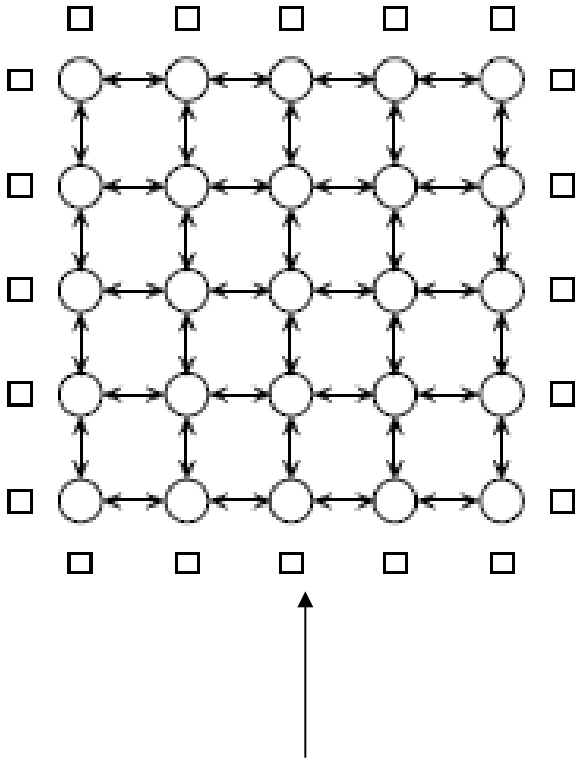
- Ajustabile: W^x, W^c si W^2
- Neajustabile: ponderile conexiunilor dintre nivelul ascuns si cel contextual

Algoritm antrenare: BackPropagation

Rețele celulare

Arhitectura:

- Toate unitățile au dublu rol: unități de intrare și unități de ieșire
- Unitățile sunt plasate în nodurile unei grile bidimensionale (notata cu L)
- Fiecare unitate este identificată prin poziția ei p=(i,j) în cadrul grilei
- Fiecare unitate este conectată doar cu unitățile aflate în vecinătatea sa (vecinătățile se definesc pe baza unei distanțe; cel mai frecvent se folosește d((i,j),(k.l))=max{|i-k|,|j-l|}

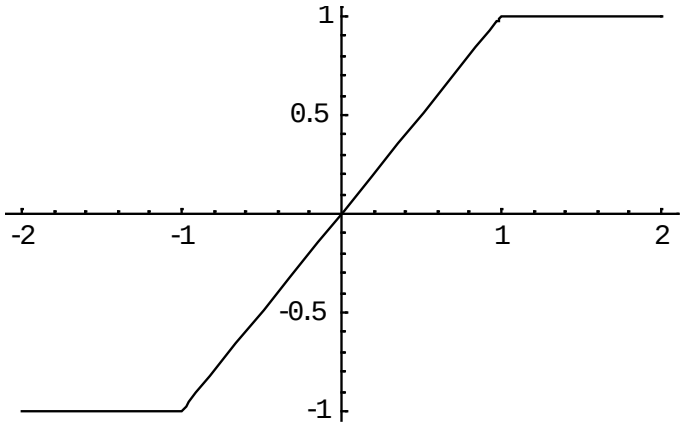


Unități virtuale

Rețele celulare

Funcție de activare: rampă

$$f(u) = (|u+1| - |u-1|)/2$$



Notatii

- $X(t) = \{x_p(t) | p \in \mathcal{L}\}$ reprezintă ansamblul semnalelor de intrare în unităţile funcţionale;
- $Y(t) = \{y_p(t) | p \in \mathcal{L}\}$ reprezintă ansamblul semnalelor de ieşire, $y_p(t) = f(x_p(t))$;
- $U(t) = \{u_p(t) | p \in \mathcal{L}\}$ reprezintă ansamblul semnalelor de control transmise unităţilor;
- $I(t) = \{i_p(t) | p \in \mathcal{L}\}$ reprezintă ansamblul pragurilor corespunzătoare unităţilor;
- a_{pq} este ponderea conexiunii dintre unitatea q şi unitatea p;
- b_{pq} este ponderea conexiunii de la componenta q a semnalului de control la unitatea funcţională p;

Rețele celulare

Funcționare:

Semnal produs de alte unități

Semnal de control (sau de intrare)

$$\frac{dx_p(t)}{dt} = -x_p(t) + \sum_{q \in V(p)} a_{pq} y_q(t) + \sum_{q \in V(p)} b_{pq} u_q(t) + i_p(t), \quad p \in \mathcal{L}$$

Semnal de intrare (similar unui prag)

Obs:

- Grila asociată rețelei este bordată cu celule virtuale care produc o valoare specifică (in funcție de condițiile la frontieră)
- Caz particular: ponderile conexiunilor dintre unitățile vecine nu depind de poziția acestor unități

Exemplu: dacă p=(i,j), q=(i-1,j), p'=(i',j'), q'=(i'-1,j') atunci

$$a_{pq} = a_{p'q'} = a_{-1,0}$$

Rețele celulare

Condițiile la frontieră definesc valorile produse de către celulele virtuale

Variante de condiții la frontieră:

- Celulele virtuale produc o valoare constantă (aceeași pentru toate sau diferite în funcție de poziția lor) – condiții Dirichlet
- Celulele virtuale preiau valoarea unității funcționale vecine (de exemplu: $z_{i,0}=z_{i,1}$, $z_{i,n+1}=z_{i,n}$, $z_{0,j}=z_{1,j}$, $z_{n+1,j}=z_{n,j}$) – condiții Neumann
- Celulele virtuale preiau valoarea unității funcționale de pe frontiera opusă (de exemplu: $z_{i,0}=z_{i,n}$, $z_{i,n+1}=z_{i,1}$, $z_{0j}=z_{nj}$, $z_{n+1,j}=z_{1,j}$) – condiții periodice la frontieră

Retele celulare

Rețelele ce folosesc ponderi care depind doar de poziția relativă a unităților se numesc rețele cu șabloane independente de poziție

Exemplu:

$$V_1(i,j) = \{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\}$$

$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \qquad B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

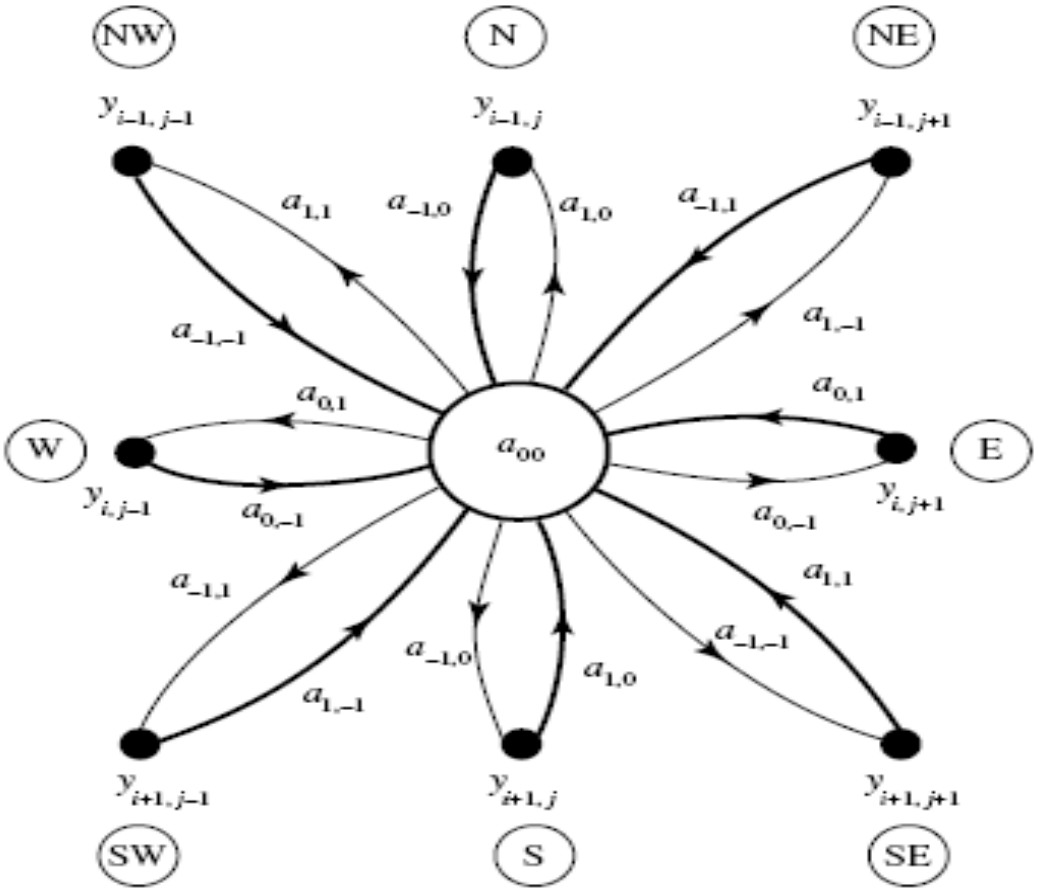
$$\frac{dx_{i,j}(t)}{dt} = -x_{i,j}(t) + \sum_{(k,l)\in V^*} a_{k,l} y_{i+k,j+l}(t) + \sum_{(k,l)\in V^*} b_{k,l} u_{i+k,j+l}(t) + I, \qquad i,j \in \{1,\ldots,n\}$$

cu

$$V^* = \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)\}$$

Retele celulare

Utilizarea elementelor din șabloanele independente de poziție:



Rețele celulare

Simulare software = rezolvare numerică a sistemului de ecuații diferențiale completat cu o condiție inițială

Se poate folosi metoda explicită a lui Euler

$$x_p(t+1) = (1-h)x_p(t) + h\Big(\sum_{(k,l)\in V^*} a_{k,l}y_{i+k,j+l}(t) + \sum_{(k,l)\in V^*} b_{k,l}u_{i+k,j+l}(t) + I\Big), \qquad i,j \in \{1,\ldots,n\}$$

p=(i,j)

Aplicabilitate:

- Prelucrarea imaginilor alb-negru sau pe nivele de gri
- Fiecare pixel este asociat cu o unitate a rețelei
- Nivelul de gri al unui pixel este codificat în [-1,1]

Rețele celulare

Prelucrarea imaginilor:

- In functie de alegerea matricilor șablon, a semnalului de control (u), a condiției inițiale (x(0)) și a condițiilor pe frontieră (z) se obțin diferite tipuri de prelucrări:
  - Binarizare (imagine pe nivele de gri -> imagine binară)
  - Umplerea golurilor în imagini binare
  - Eliminarea zgomotului (pixeli izolați) din imagini binare
  - Detectarea contururilor în imagini binare

Observație: imaginile alb negru sunt codificate în {-1,1}:
Alb: -1
Negru: +1

Rețele celulare

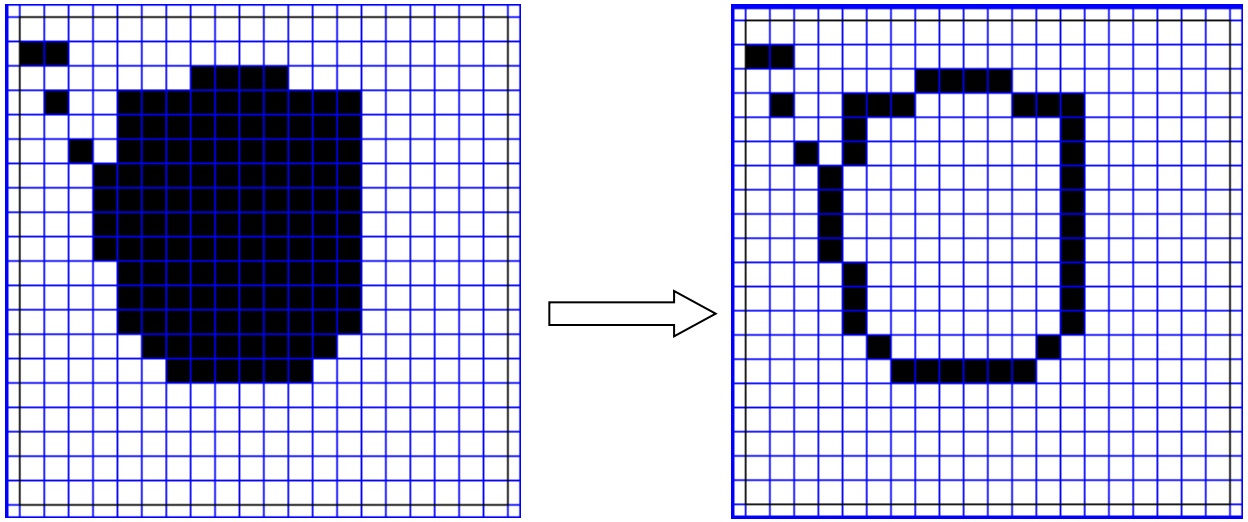
Exemplu 1: identificarea contururilor

z=-1, U=imaginea de prelucrat,

h=0.1

$$A=\begin{bmatrix}0 & 0 & 0\\ 0 & 3 & 0\\ 0 & 0 & 0\end{bmatrix},\quad B=\begin{bmatrix}0 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 0\end{bmatrix}$$

$$I=-1,\ X(0)=U$$



http://www.isiweb.ee.ethz.ch/haenggi/CNN_web/CNNsim_adv.html

Rețele celulare

Exemplu 2: umplerea golurilor

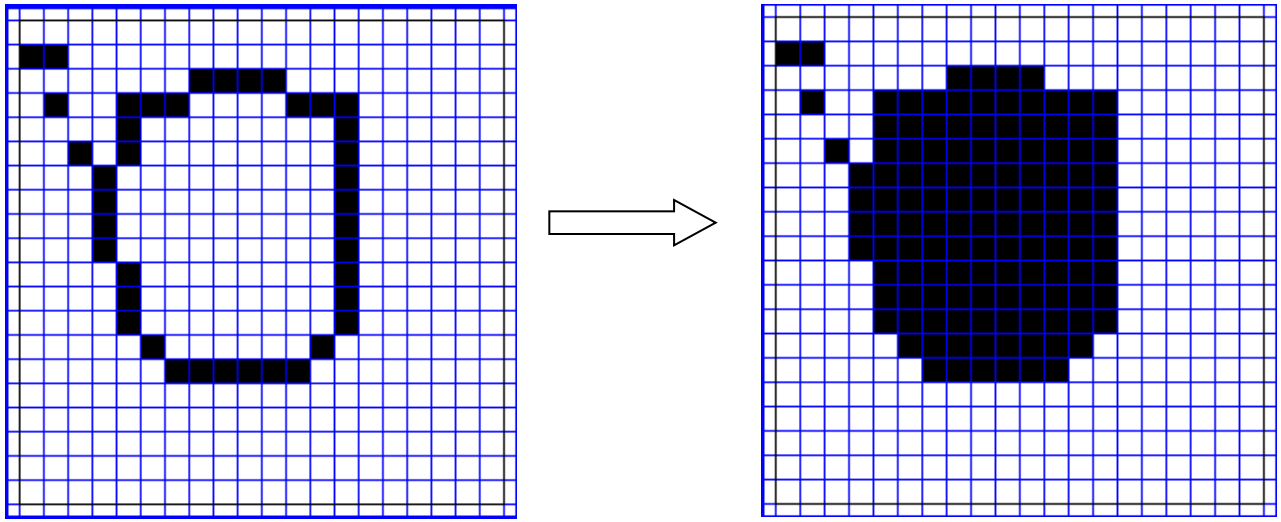
z=-1,

U=imaginea de prelucrat,

h=0.1

$$A=\begin{bmatrix}0 & 1 & 0\\ 1 & 1.5 & 1\\ 0 & 1 & 0\end{bmatrix},\quad B=\begin{bmatrix}0 & 0 & 0\\ 0 & 4 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$I=0.5,\ x_{ij}(0)=1\ (\text{toti pixelii sunt egali cu } 1)$$



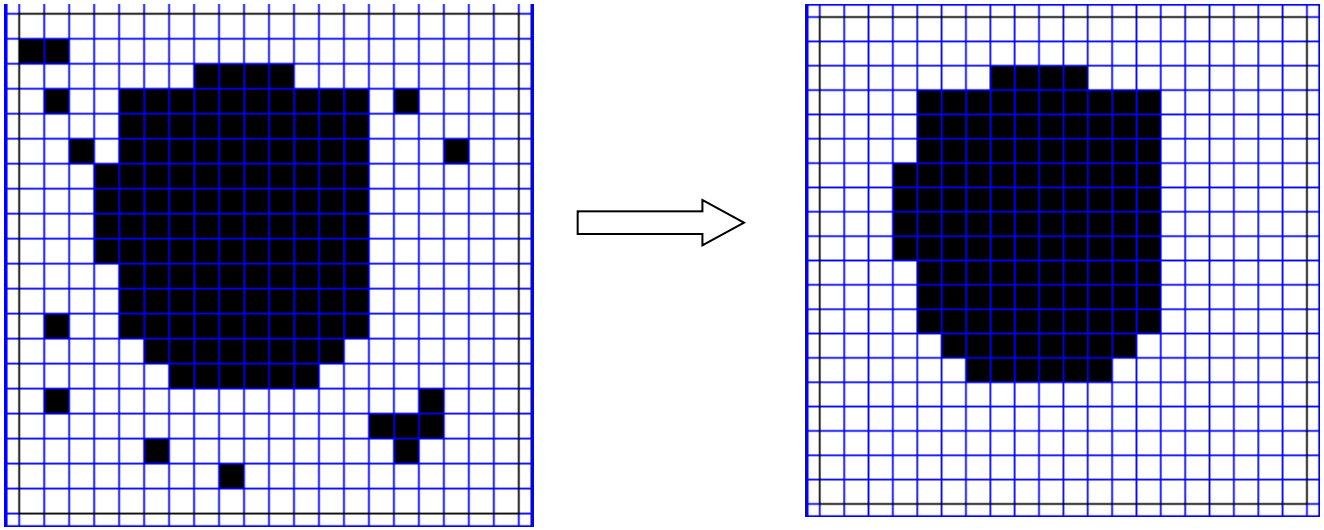
Rețele celulare

Exemplu 3: eliminarea zgomotului

z=-1, U=imaginea de prelucrat, h=0.1

$$A=\begin{bmatrix}0 & 1 & 0\\ 1 & 2 & 1\\ 0 & 1 & 0\end{bmatrix},\quad B=\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$I=0,\ X(0)=U$$



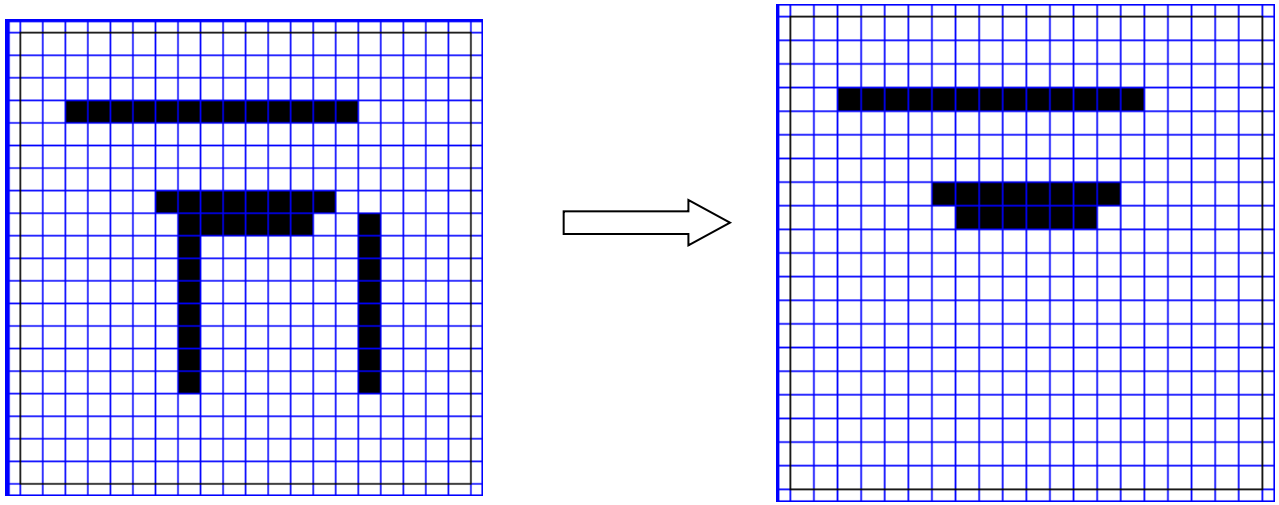
Rețele celulare

Exemplu 4: detectare linii orizontale

z=-1, U=imaginea de prelucrat, h=0.1

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = -1, X(0) = U$$



Alte clase de rețele recurente

Reservoir computing (www.reservoir-computing.org)

Specific:

- Utilizeaza un set de unități ascunse (numit rezervor) conectate arbitrar (ponderile conexiunilor sunt stabilite aleator); fiecare dintre aceste unități realizează o transformare neliniară a semnalelor preluate de către unitățile de intrare

- Semnalele de ieșire sunt determinate prin combinarea liniară a semnalelor produse de unitățile din rezervor (și a celor de intrare)

- Doar ponderile conexiunilor către unitățile de ieșire sunt antrenate

Alte clase de rețele recurente

Reservoir computing (www.reservoir-computing.org)

Variante:

- *Temporal Recurrent Neural Network* (Dominey 1995)
- *Liquid State Machines* (Natschläger, Maass and Markram 2002)
- *Echo State Networks* (Jaeger 2001)
- *Decorrelation-Backpropagation Learning* (Steil 2004)

Alte clase de rețele recurente

Reservoir computing (www.reservoir-computing.org)

Echo State Networks:

U(t) = vector cu semnale de intrare

X(t) = vector de stare corespunzător unităților din rezervor

Z(t)=[U(t);X(t)] = vector intrare concatenat cu vector de stare

Y(t) = vector cu semnalul de ieșire

$X(t)=(1-a)x(t-1)+a\ \tanh(W^{in}\ U(t)+W\ x(t-1))$

$Y(t)=W^{out}\ Z(t)$

W^{in} ,W – stabilite aleator (W scalata a.i. raza spectrală = valoarea proprie maximă să aibă o valoare prestabilită);

W^{out} determinat prin antrenare

Alte clase de rețele recurente

Reservoir computing (www.reservoir-computing.org)

Aplicații:

- recunoașterea vorbirii
- Recunoașterea scrisului de mână
- Controlul roboților
- Predicția datelor financiare
- Predicția în timp real a atacurilor de epilepsie

Alte clase de rețele recurente

Deep learning (http://deeplearning.net/)

Specific:

- Arhitectura cu multe nivele (scop: extragerea ierarhica a caracteristicilor datelor);

- Pre-antrenare nesupervizata bazata pe RBM (Restricted Boltzmann Machines) urmata de antrenare supervizata (ex: Backpropagation)

Obs:

- Boltzmann Machines = retele recurente cu unitati cu functionare aleatoare
- Restricted BM = retele recurente organizate pe nivele cu conexiuni complete doar intre unitatile aflate pe nivele diferite
- Unele retele cu structura “adâncă” sunt rețele feed-forward (ex: Convolutional Neural Networks)

Alte clase de rețele recurente

Deep learning (http://deeplearning.net/)

Aplicatii:

- Clasificare imagini si identificare obiecte (ex: recunoaștere fețe – Deep Face)
- Recunoasterea vorbirii (Google Brain, Siri)
- Indexare semantica (ex: word2vec) si traducere automata
- Simulare vise (http://npcontemplation.blogspot.ca/2012/02/machine-that-can-dream.html)