

Inteligență Artificială

cursul 11

Bogdan Alexe

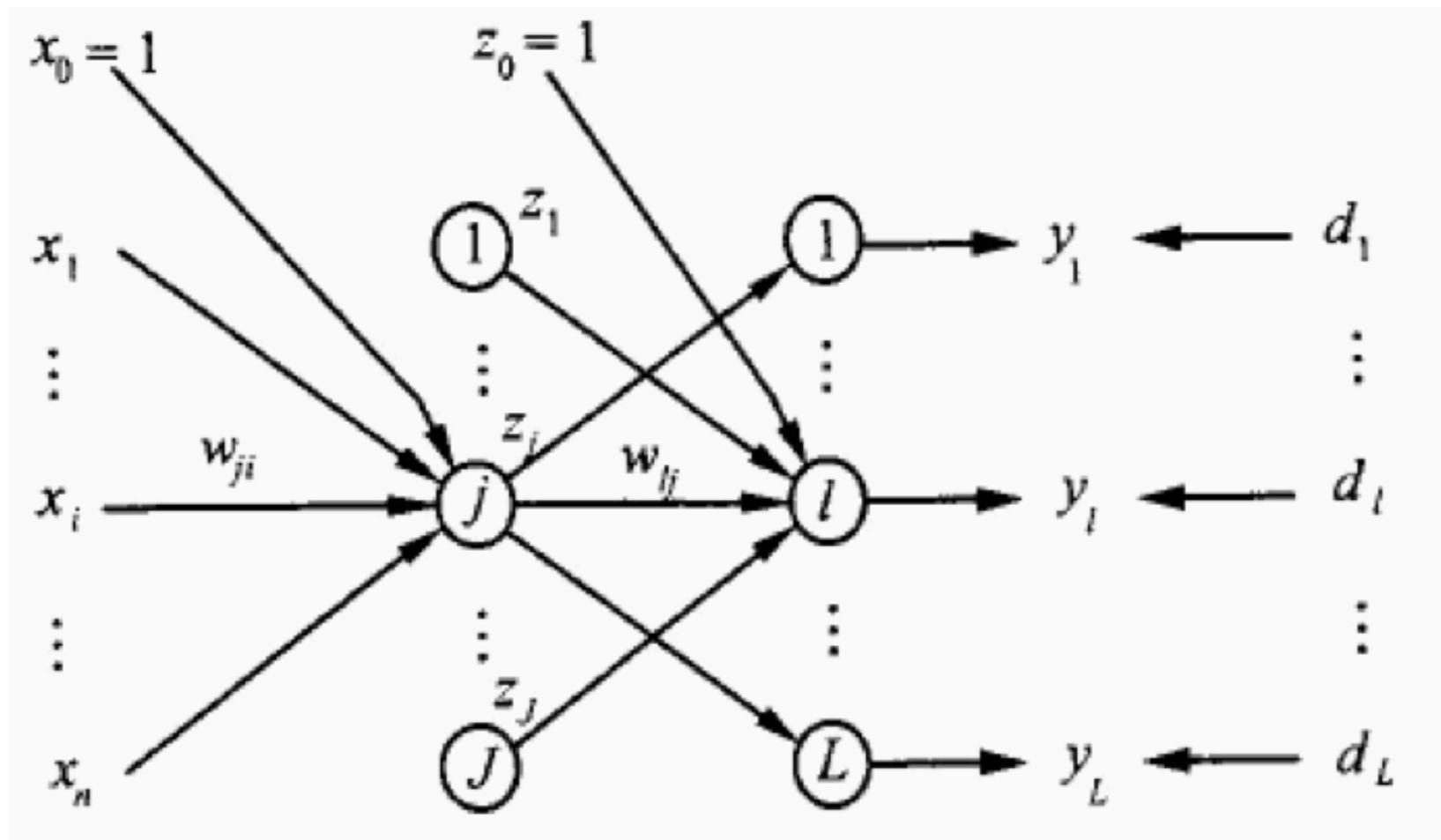
Material de curs realizat cu ajutorul domnului profesor
Denis Enăchescu

Laborator IA pentru grupa 235

- miercuri, 17 mai, 12-14, faceți Rețele de Calculatoare în loc de laboratorul de Inteligență Artificială (toată grupa)
- joi, 18 mai, 14-16 în loc de Rețele de Calculatoare faceți Inteligență Artificială cu mine la 221A (împreună cu cei de la 2322)

Data trecută

Algoritmul backpropagation de antrenare a unei rețele neuronale feedforward



Data trecută

Algoritmul backpropagation de antrenare a unei rețele neuronale feedforward

$$\mathcal{S} = \{\mathbf{x}^k, \mathbf{d}^k\}_{k=1}^m$$

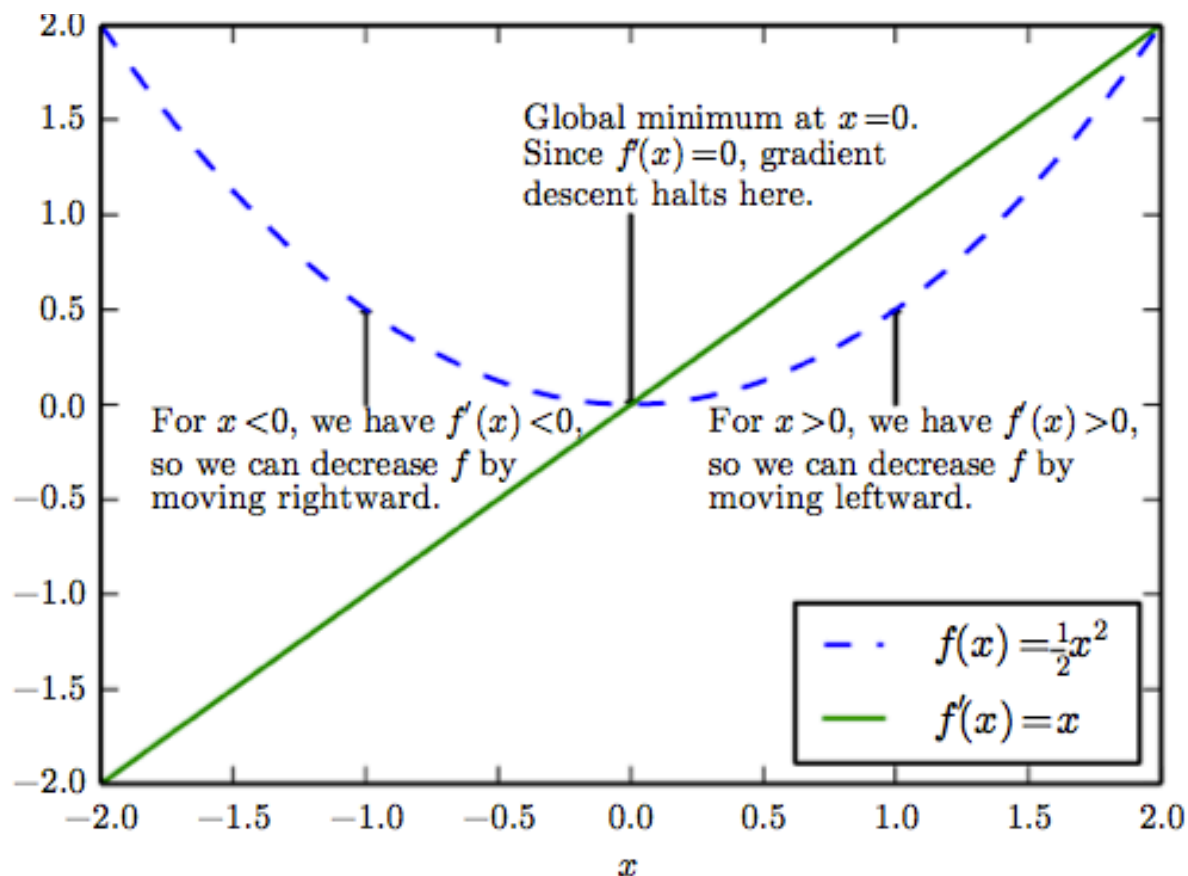
Multime de antrenare:
exemple \mathbf{x} au dimensiunea n ,
etichetele \mathbf{d} au dimensiunea L

$$E(\mathbf{w}) = \frac{1}{2} \sum_{l=1}^L (d_l - y_l)^2 \quad (3.1)$$

\mathbf{w} are $J(n+1) + L(J+1)$ ponderi

Minimizare folosind algoritmul de coborâre pe gradient (gradient descent)

Coborâre pe gradient



$$f(x + \epsilon) \approx f(x) + \epsilon f'(x)$$

Coborâre pe gradient

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x)$$

$$f'(x) > 0 \rightarrow f \text{ crește}$$

$$f'(x) < 0 \rightarrow f \text{ scade}$$

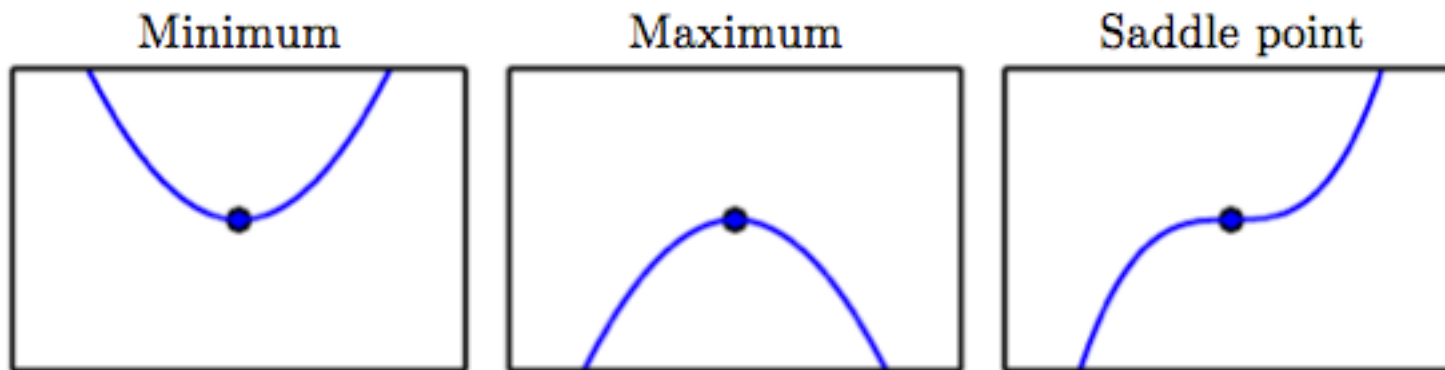
$$\text{Dacă } \epsilon > 0 \rightarrow f(x - \epsilon \cdot \text{sign}(f'(x))) \leq f(x)$$



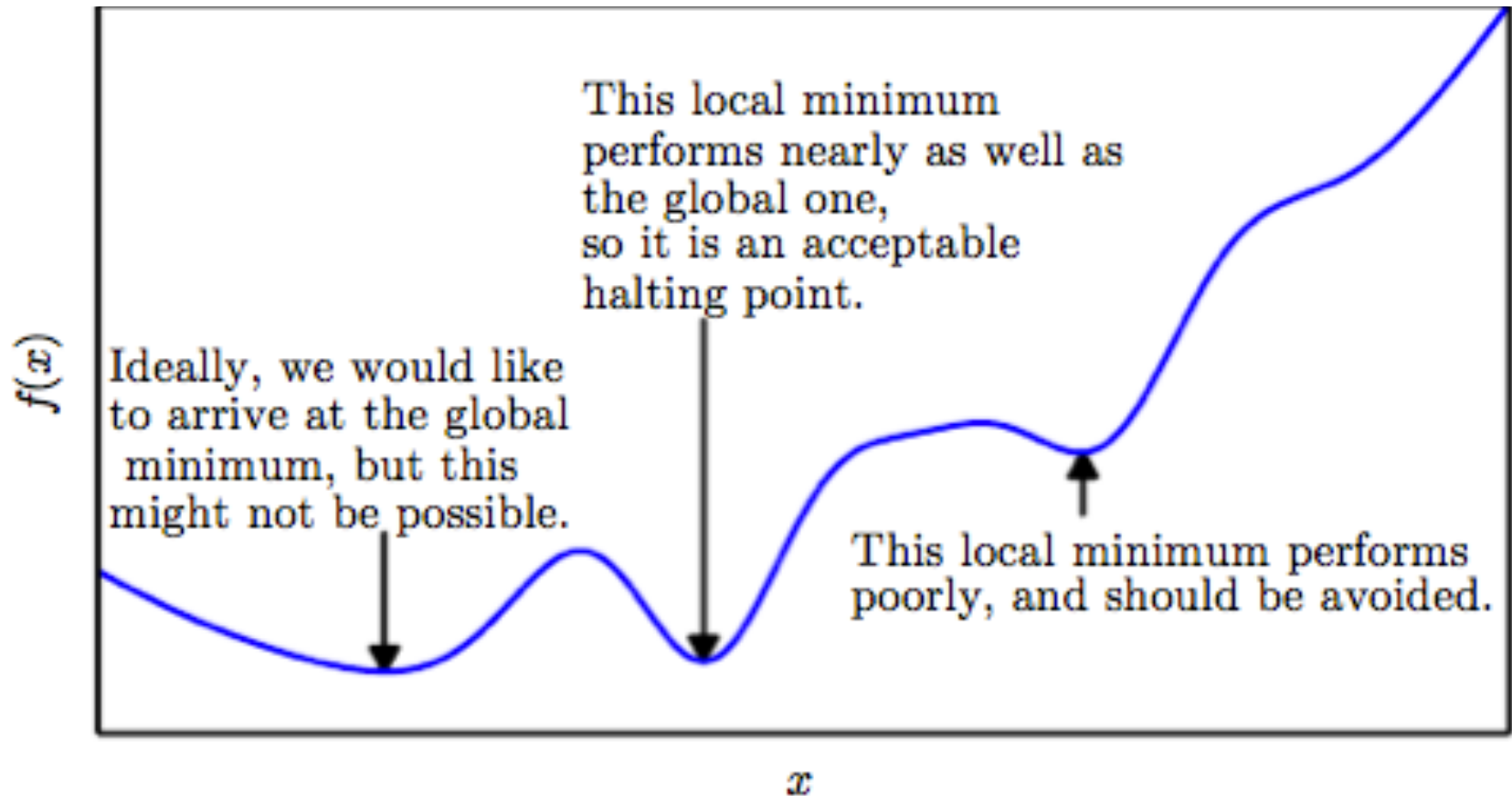
coborâre pe gradient în 1D

Tipuri de puncte critice

x e punct critic $\rightarrow f'(x) = 0$



Minim local vs minim global



Coborâre pe gradient în R^s

\mathbf{w} are $s = J(n+1) + L(J+1)$ ponderi

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \rho \cdot \nabla_{\mathbf{w}^k} E(\mathbf{w}^k)$$

coborâre pe gradient în s dimensiuni

$$\Delta w_{lj} = w_{lj}^{\text{new}} - w_{lj}^c = -\rho_o \frac{\partial E}{\partial w_{lj}} = \rho_o (d_l - y_l) f'_o(\text{net}_l) z_j \quad (3.2)$$

$$\Delta w_{ji} = -\rho_h \frac{\partial E}{\partial w_{ji}} \quad j=1,2,\dots,J; \quad i=0,1,2,\dots,n \quad (3.4)$$

$$\Delta w_{ji} = \rho_h \left[\sum_{l=1}^L (d_l - y_l) f'_o(\text{net}_l) w_{lj} \right] f'_h(\text{net}_j) x_i \quad (3.9)$$

Incremental vs batch

$$E(\mathbf{w}) = \frac{1}{2} \sum_{l=1}^L (d_l - y_l)^2 \quad (3.1)$$

Eroarea calculată pentru un singur exemplu (cel curent)

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^m \sum_{l=1}^L (d_l - y_l)^2 \quad (3.13)$$

Eroarea calculată pentru toate exemplele

Coborâre pe gradient folosind minibatch-uri

Aproximează gradientul pe baza numai a m' exemple (mini-batch) $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m')}\}$

$$E(\mathbf{w}) = \frac{1}{m'} \sum_{j=1}^{m'} \sum_{l=1}^L (d_l^{(j)} - y_l^{(j)})^2$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \rho_{k+1} \cdot \nabla_{\mathbf{w}^k} E(\mathbf{w}^k)$$

↑
rata la iteratia k+1, trebuie sa scada la fiecare iteratie, altfel introduce zgomot
Pentru convergenta este suficient ca:

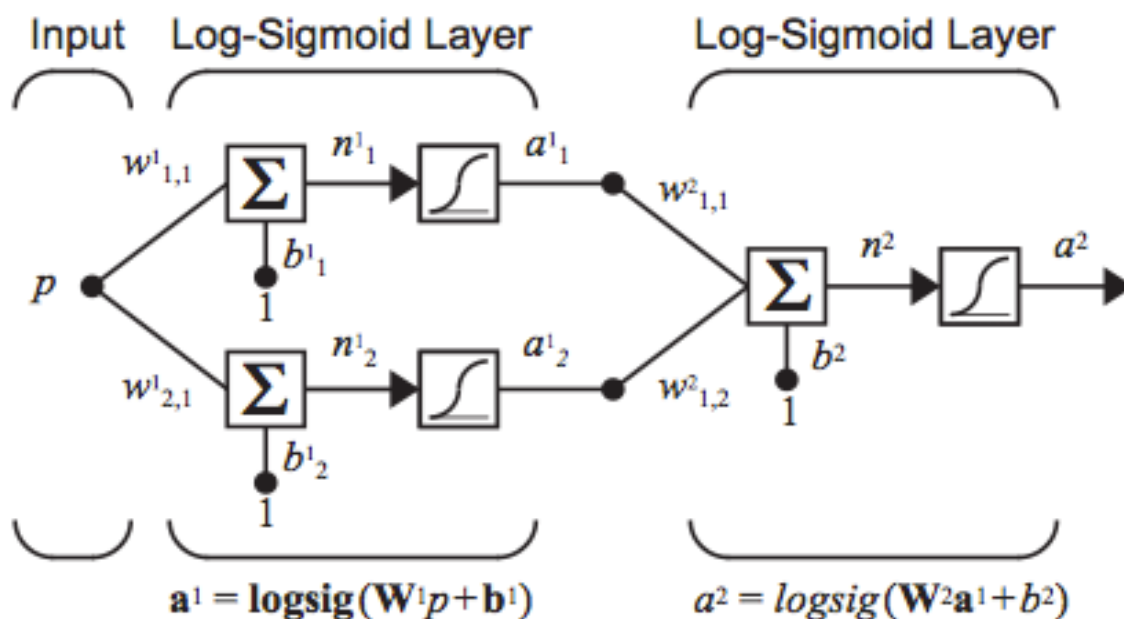
$$\sum_{k=1}^{\infty} \rho_k = \infty, \sum_{k=1}^{\infty} \rho_k^2 < \infty,$$

Dezavantaje backpropagation

- viteză lentă de convergență
- oscilații în găsirea soluției optime (zig –zag)
- soluție minim local
- soluție cu valoare mică a funcției eroare dar cu capacitate mică de generalizare (overfitting)

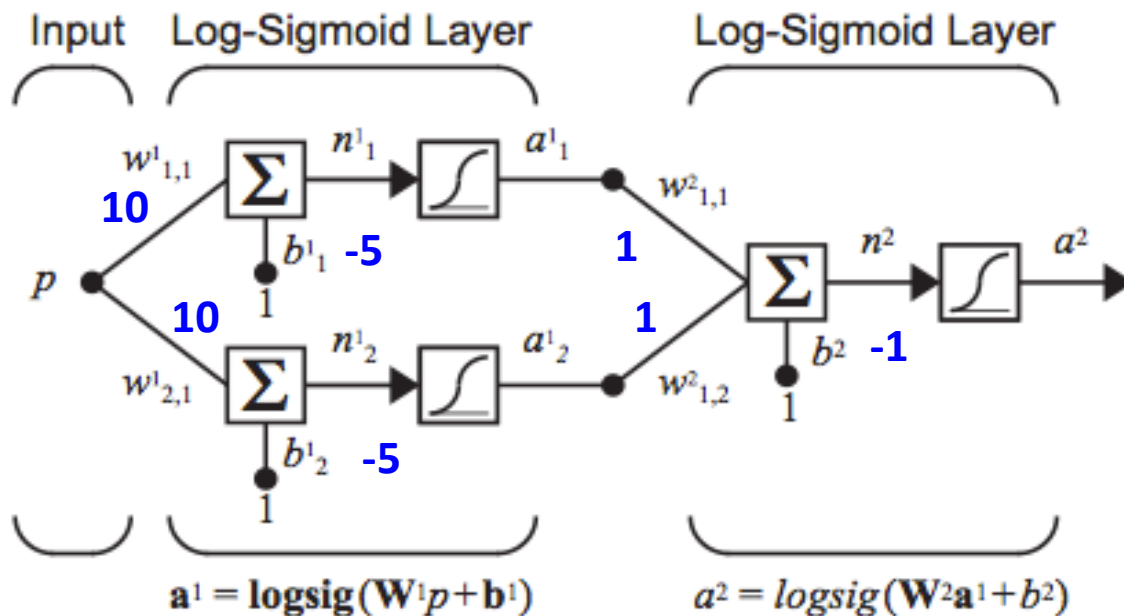
Dezavantaje backpropagation

Exemplu: rețeaua netf care implementează funcția f



Dezavantaje backpropagation

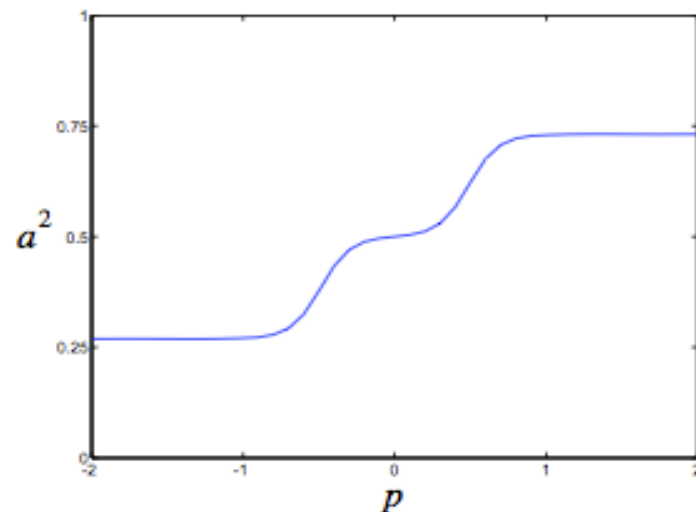
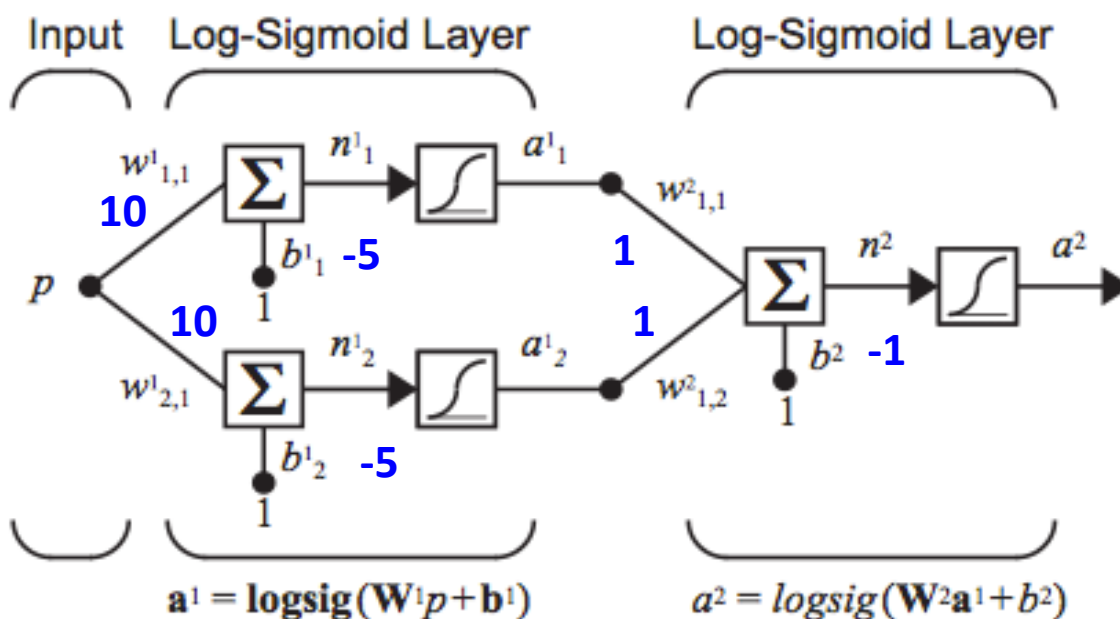
Exemplu: rețeaua netf care implementează funcția f



Particularizez cei 7 parametri

Dezavantaje backpropagation

Exemplu: rețeaua netf care implementează funcția f

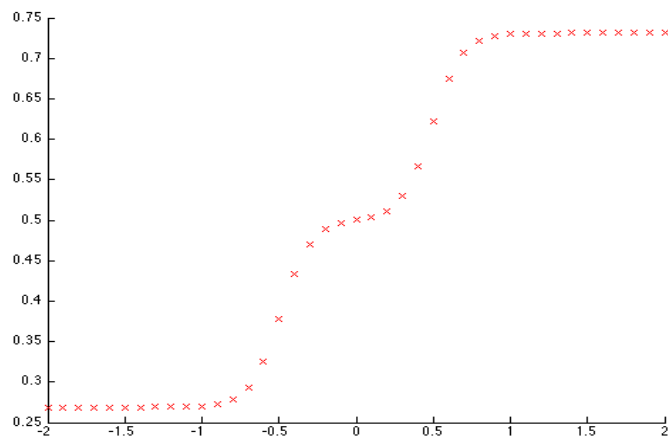


Particularizez cei 7 parametri

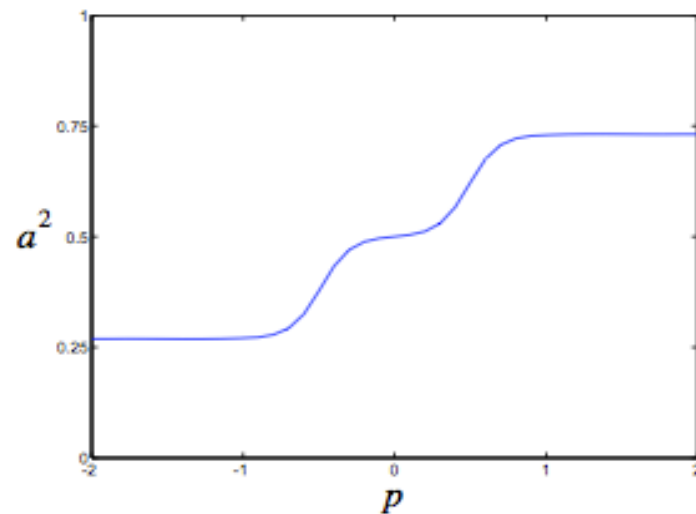
Funcția f

Dezavantaje backpropagation

Exemplu: consideram o rețeaua **net** cu aceeași arhitectură ca **netf** pentru care vrem să estimăm parametri optimi pe baza unei mulțimi de antrenare



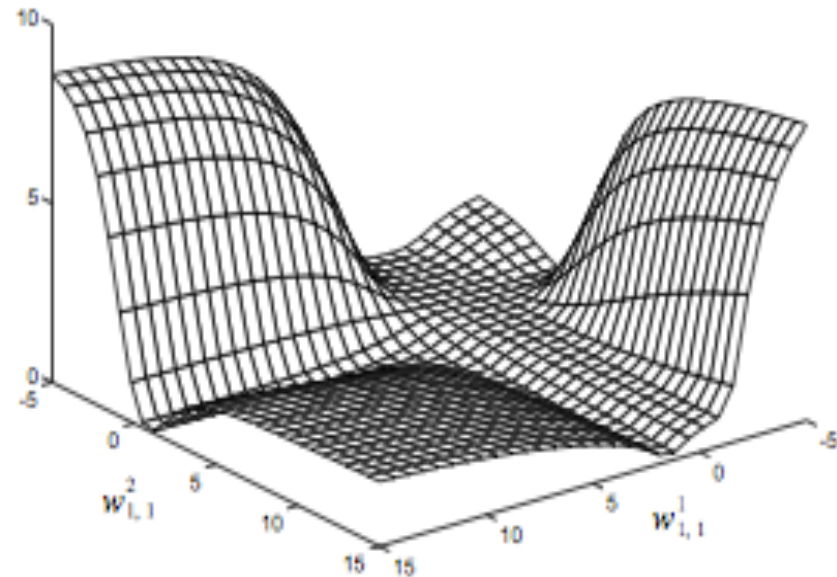
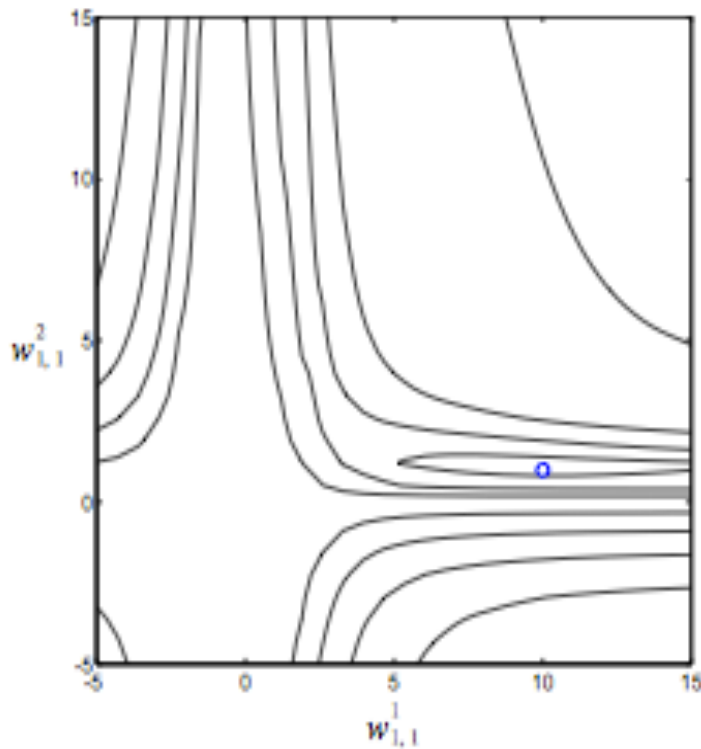
Mulțimea de antrenare



Funcția f

Dezavantaje backpropagation

Vizualizare suprafața de eroare E
(fixăm 5 parametri, vizualizăm 3D)



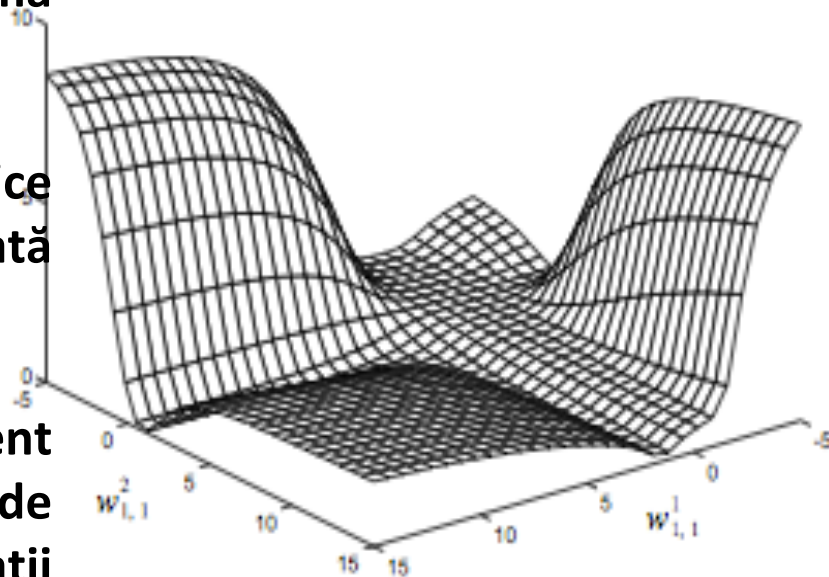
Dezavantaje backpropagation

Vizualizare suprafața de eroare E
(fixăm 5 parametri, vizualizăm 3D)

Suprafața de eroare E nu este convexă, variază puternic având uneori o formă plată (platou), alteori abruptă.

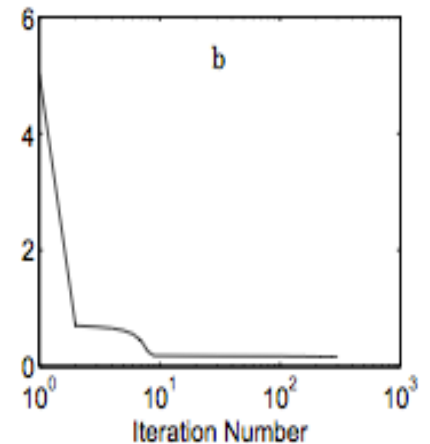
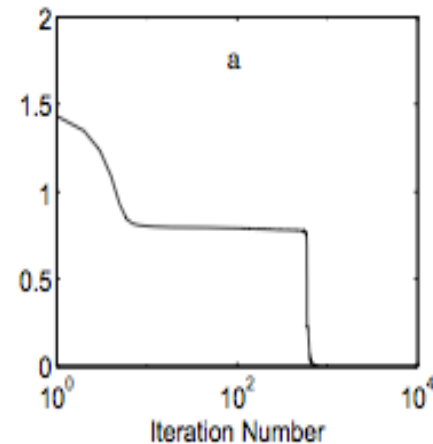
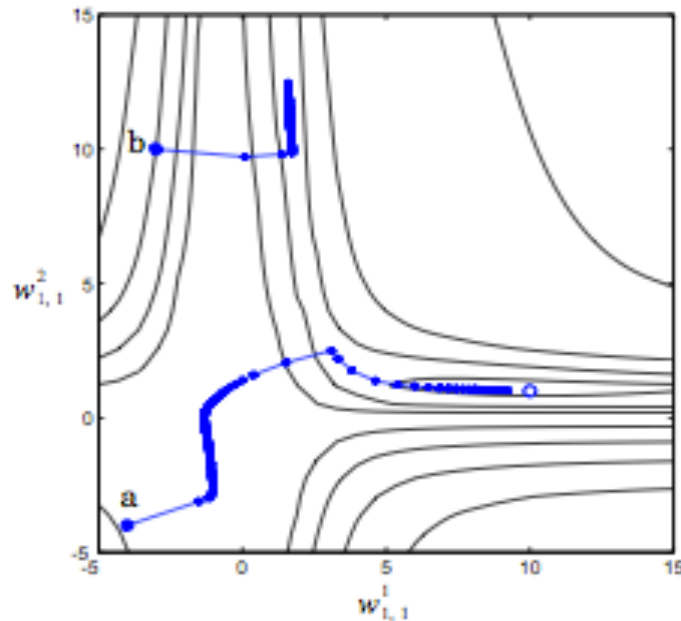
- în regiunile de platou (caracteristice pentru sigmoide) , ar fi de dorit o rată de învățare mare;

- în regiunile foarte abrupte, cu gradient foarte mare, ar fi de dorit o rată de învățare mica pentru a nu avea variații foarte mari ale funcției eroare E



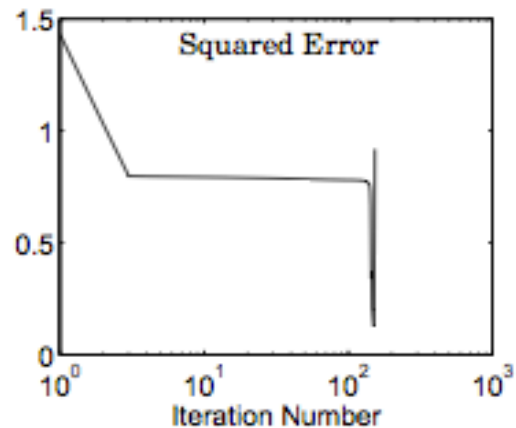
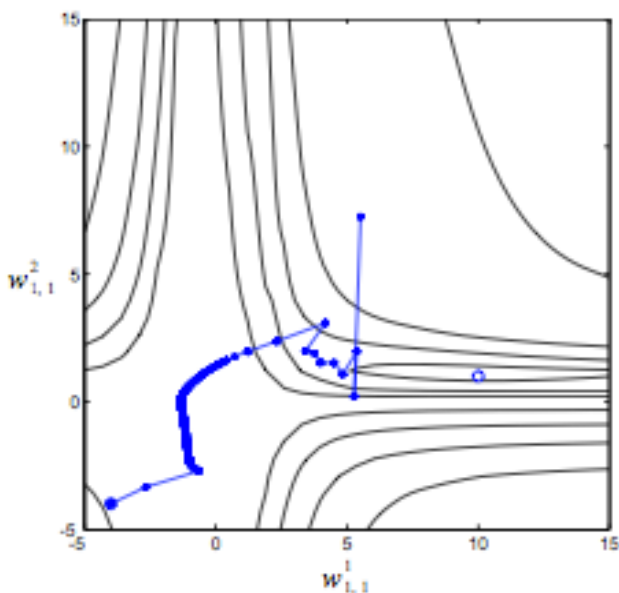
Dezavantaje backpropagation

Sensibil la inițializare: inițializări diferite duc la soluții diferite



Dezavantaje backpropagation

Sensibil la rata de învățare: rată de învățare mare conduce la oscilații mari în traiectorie, diverge de la punctul de la minim local



Matlab: programele demonstrative **nnd12sd1, nnd12sd2**

Îmbunătățiri la backpropagation

1. strategii de inițializare a parametrilor
2. învățare cu moment
3. rată de învățare variabilă
4. bazate pe metode de optimizare: Newton, gradient conjugat, Levenberg Marquardt

Strategii de inițializare a parametrilor

- bazate pe euristici astfel încât rețeaua astfel inițializată să prezinte anumite proprietăți “dorite”
- nu e clar dacă asemenea proprietăți se păstrează când se începe învățarea
- trade-off: inițializări bune pentru optimizare (conduc la puncte cu valoare mică a funcției E) dar care oferă o capacitate de generalizare slabă

Strategii de inițializare a parametrilor

- un singur lucru cert: inițializează parametri astfel încât fiecare perceptron de pe același strat (are aceeași funcție de activare) să învețe o altă funcție (“fără simetrii”). Dacă doi perceptroni sunt inițializați cu aceleași ponderi, vor învăța aceeași funcție (vor primi întotdeauna același update)
- uzual se inițializează ponderile cu valori aleatoare selectate din distribuția uniformă $[-r, r]$ sau Gaussiană de medie 0

Strategii de inițializare a parametrilor

- scala r este aleasă astfel încât:

1. să nu fie prea mică – să nu apară perceptroni care învață aceeași funcție

2. să nu fie prea mare - produce “saturație” pentru perceptronii cu funcții de activare sigmoide

-exemplu: se inițializează ponderile unui perceptron cu n_i intrări ca valori aleatoare în $[-3*n_i^{-0.5}, 3*n_i^{-0.5}]$; dacă intrările perceptronului urmează distribuția $N(0,1)$ atunci net_i (semnalul primit de perceptron după integrare) e de medie 0 și deviație standard 1.

Învățare cu moment

$$\Delta w_{lj} = w_{lj}^{\text{new}} - w_{lj}^c = -\rho_o \frac{\partial E}{\partial w_{lj}} = \rho_o (d_l - y_l) f'_o(\text{net}_l) z_j \quad (3.2)$$

$$\Delta w_{ji} = -\rho_h \frac{\partial E}{\partial w_{ji}} \quad j=1,2,\dots,J; \quad i=0,1,2,\dots,n \quad (3.4)$$


$$\Delta w_{ji} = \rho_h \left[\sum_{l=1}^L (d_l - y_l) f'_o(\text{net}_l) w_{lj} \right] f'_h(\text{net}_j) x_i \quad (3.9)$$

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$

Învățare cu moment

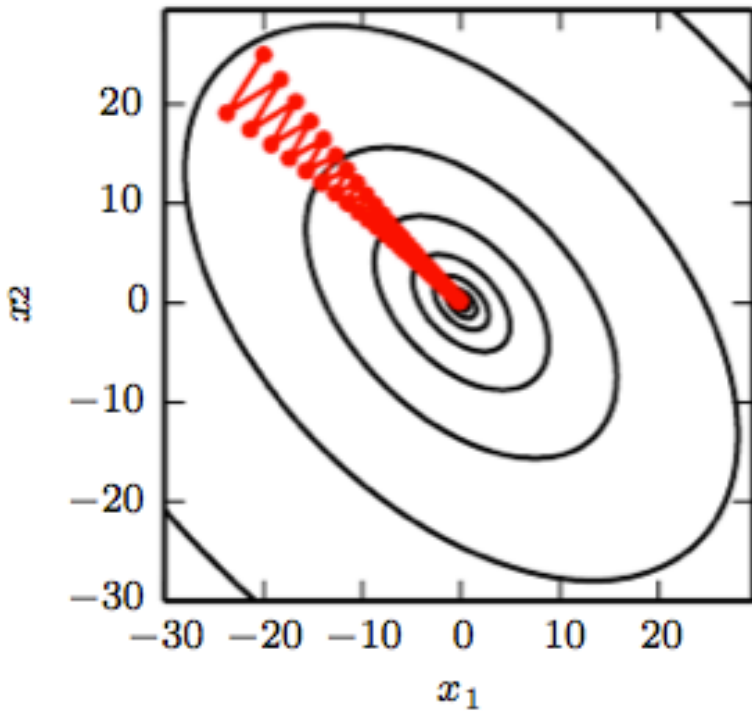
Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$


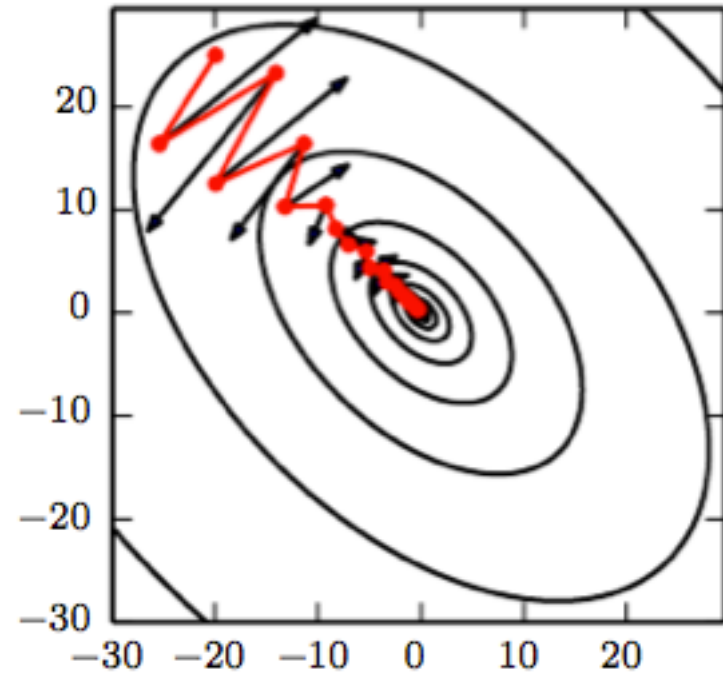
Momentul: acumulează contribuția gradientilor din trecut

Intuiție: accelerează coborârea în direcția medie, previne oscilațiile

Învățare cu moment




Învățare fără moment



Învățare cu moment

Învățare cu moment

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$


Parametrul $0 \leq \alpha \leq 1$ controlează rata cu care contribuția din trecut a gradientilor influențează direcția actuală de coborâre

Se realizează un update foarte mare în momentul în care se acumulează foarte mulți gradienti cu aceeași direcție

Învățare cu moment

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$

Update-ul cu momentul după N iterații:

$$\Delta w_i(t) = -\rho \sum_{n=0}^{N-1} \alpha^n \frac{\partial E}{\partial w_i(t-n)} + \alpha^N \Delta w_i(t-N) \quad (3.19)$$

În regiunile platou, am un gradient mic, posibil constant:

$$\Delta w_i(t) \approx -\rho \frac{\partial E}{\partial w_i(t)} \sum_{n=0}^{N-1} \alpha^n = -\frac{\rho}{1-\alpha} \frac{\partial E}{\partial w_i(t)} \quad (3.20)$$

Învățare cu moment

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

În regiunile platou, am un gradient mic, posibil constant:

$$\Delta w_i(t) \approx -\rho \frac{\partial E}{\partial w_i(t)} \sum_{n=0}^{N-1} \alpha^n = -\frac{\rho}{1-\alpha} \frac{\partial E}{\partial w_i(t)} \quad (3.20)$$

Algoritmul accelerează ieșirea din platou cu o rata crescută de factor $1/(1-\alpha)$

Învățare cu moment

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$


Pot avea parametrul alpha care se modifică în timp:

$$\alpha(t) = \frac{\frac{\partial E}{\partial w_i(t)}}{\frac{\partial E}{\partial w_i(t-1)} - \frac{\partial E}{\partial w_i(t)}}$$

Pentru aceeași direcție a derivatei parțiale dar de magnitudine mai mică algoritmul accelerează. Pentru direcție opusă, algoritmul decelerează.

Învățare cu moment

Metoda momentelor adaugă un termen reprezentând “momentul” la ecuațiile de update (3.2 și 3.9)

$$\Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i(t)} + \alpha \Delta w_i(t-1) \quad (3.18)$$


Momentul: acumulează contribuția gradientilor din trecut

Pentru regiuni abrupte termenul de moment are magnitudine mică

Rată de învățare variabilă

- trade-off pentru rate de învățare:
 - valoare mare: convergență inițială rapidă (bună pentru regiuni platou), dar apoi oscilații (divergență de la soluția optimă)
 - valoare mică: convergență lentă (se fac multe update-uri), previne oscilațiile
- diverse euristici legate de gradient, de valoarea funcției obiectiv

Rată de învățare variabilă

- **exemplul 1:** rate de învățare constante invers proporționale cu numărul de intrări ale unui perceptron (pentru a preveni “saturația” perceptronilor cu funcții de activare sigmoide)
- **exemplul 2:** rate de învățare variabile pentru fiecare pondere în funcție de derivatele parțiale ale funcției E : dacă ele își mențin același semn (direcția e bună) atunci crește rata de învățare, altfel scade

Rată de învățare variabilă

- **exemplul 3:** rate de învățare variabilă în funcție de valoarea lui E:
 - dacă valoarea funcției eroare E crește cu un procent x (în MATLAB $max_perf_inc = 1.04$ implicit) după realizarea unui update, atunci update-ul nu se mai realizează, rata de învățare curentă se înmulțește cu o valoare subunitară $0 < lr_dec < 1$ (implicit parametrul $lr_dec = 0.7$), iar coeficientul mc (în cazul în care este folosit) care specifică momentul se setează la 0.
 - dacă valoarea funcției eroare E scade după realizarea unui update, se acceptă acest update iar rata de învățare curentă se înmulțește cu o valoare supraunitară (parametrul $lr_inc = 1.05$ implicit). Dacă coeficientul mc care specifică momentul este setat la 0, se resetează la valoarea inițială.

Metode de optimizare

Până acum am folosit update-uri pe bază de gradienti (derivate de ordinul întâi)

Alte metode (Newton, gradient conjugat, Levenberg-Marquardt) folosesc **aproximarea unui funcții folosind dezvoltarea în serie Taylor până la ordinul 2**

$$\tilde{E}(\mathbf{w}^c + \Delta \mathbf{w}) = E(\mathbf{w}^c) + \nabla E(\mathbf{w}^c)^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \nabla^2 E(\mathbf{w}^c) \Delta \mathbf{w}$$

Metoda lui Newton

$$\tilde{E}(\mathbf{w}^c + \Delta \mathbf{w}) = E(\mathbf{w}^c) + \nabla E(\mathbf{w}^c)^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \nabla^2 E(\mathbf{w}^c) \Delta \mathbf{w}$$

$$\tilde{E}(\mathbf{w}^c + \Delta \mathbf{w}) = 0, \Rightarrow \Delta \mathbf{w} = -[\nabla^2 E(\mathbf{w}^c)]^{-1} \nabla E(\mathbf{w}^c) = -[H(\mathbf{w}^c)]^{-1} \nabla E(\mathbf{w}^c).$$

$\mathbf{H}_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$. matricea Hessiana, trebuie sa fie pozitiv definită pentru ca update-ul să meargă în direcția bună

Algoritm foarte încet, trebuie să inverseze matricea Hessiana în fiecare punct.

Gradienți conjugăți

$$\mathbf{d}(t) = -\nabla E(t) + \beta \mathbf{d}(t-1), \text{ with } \mathbf{d}(0) = -\nabla E(0).$$

Dirrecția de coborâre $\mathbf{d}(t)$ este un compromis între direcția gradientului și direcția anterioară. Direcția este conjugată în raport cu matricea Hessiană \mathbf{H} :

$$\mathbf{d}(t-1)^T \mathbf{H}(t-1) \mathbf{d}(t) = 0,$$

Regula Polak-Ribiere:
$$\beta = \beta(t) = \frac{[\nabla E(t) - \nabla E(t-1)]^T \nabla E(t)}{\|\nabla E(t-1)\|^2}$$

Accelerează convergența când ne aflăm în vecinătatea unui minim local

Levenberg-Marquardt

Combină backpropagation în variantă incrementală la început cu gradient conjugat la final

Este opțiunea implicită de antrenare în Matlab (destul de înceată – calculează Hessiana)