Simularea reţelelor neuronale artificiale

21 martie 2000

Cuprins

Capitolul 1

Introducere în reţele neuronale

1.1 Inteligenţă artificială. Simbolism şi conexionism

1.2 Definiţii

Terminologia folosită în diversele articole sau programe de simulare a reţelelor neuronale artificiale este departe de a fi unitară. Ca un exemplu, "întrebările des-puse" (FAQ) ale discuţiilor USENET `comp.ai.neural-nets` îngrijite de Warren S. Sarle [6] menţionează trei modalităţi diferite de numărare a straturilor: incluzându-le pe cele de intrare sau nu, incluzându-le pe cele de ieşire sau nu, numărând seturile de ponderi în loc de straturile de neuroni ş.a.m.d.

Definiţiile reţelelor neuronale sunt şi ele destul de diverse. Astfel, ele sunt considerate a fi*"sisteme compuse din multe elemente de procesare simple, operând în paralel, ale căror funcţii sunt determinate de structura reţelei, ponderile conexiunilor şi de procesările efectuate de noduri sau elementele de calcul[1]"*, sau *"procesoare distribuite cu un grad foarte mare de paralelism, care au o înclinaţie naturală de a memora cunoştinţe experimentale şi a le face disponibile pentru a fi folosite. Seamănă cu creierul în două aspecte: 1. Cunoştinţele sunt acumulate de reţea printr-un proces de învăţare; 2. Ponderile conexiunilor inter-neuronale, cunoscute sub numele de ponderi sinaptice, sunt folosite pentru a stoca aceste cunoştinţe[2]"*, sau *"circuite compuse dintr-un număr mare de elemente simple de procesare care sunt bazate pe neuroni. Fiecare element operează numai asupra unei informaţii locale. Mai mult, fiecare element operează asincron; astfel, nu există un ceas al întregului sistem[3]"*.

Literatura română de specialitate propune şi ea definiţii. Dumitrescu şi Costin [1], de exemplu, arată în prefaţa lucrării *Reţele neuronale. Teorie şi aplicaţii*: *"Caracteristic reţelelor neuronale este faptul că, pornind de la o mulţime de exemple, ele sunt capabile să sintetizeze în mod implicit un anumit model al problemei. Am putea spune că o reţea neuronală construieşte singură algoritmul pentru rezolvarea unei probleme"* (p. 14). Pe de altă parte, Neagoe şi

[1] *DARPA Neural Network Study* (1988, AFCEA International Press, p. 60), apud. [6].
[2] *Neural Networks: A Comprehensive Foundation*, Haykin, S., NY, MacMillan, p. 2, apud. [6].
[3] *Introduction to Artificial Neural Systems,* Zurada, J.M., 1992, Boston, p. xv, apud [6].

Stănăşilă [2], definesc reţelele neuronale ca fiind *“modele computaţionale paralele cu grade de complexitate variate, constituite din unităţi de procesare elementare, de cele mai multe ori adaptative (denumite neuroni), interconectate masiv (dens), conform unei organizări ierarhice”.*

După cum vedem, definiţiile abundă. Putem însă esenţializa principalele trăsături ale unei reţele neuronale:

1. Este un sistem *distribuit*, format dintr-un număr mare de *elemente simple* de procesare;

2. Este un sistem profund şi masiv paralel – datele prelucrate sunt de cele mai multe ori locale, deci este *foarte adecvat simulării paralele.* În plus, elementele de procesare sunt *masiv interconectate.*

3. Este un sistem care formează intern o *reprezentare* a realităţii din afară.

4. *Reprezentarea* cunoştinţelor stocate este de asemenea *distribuită.* Acesta este un avantaj pentru că deteriorări minore ale reţelei duc la deteriorări minore ale funcţionalităţii ei.

5. Se aseamănă, principial, cu structura cunoscută a cortexului cerebral — cu toate acestea unele tipuri de reţele (printre care şi *backprop*, cea mai cunoscută), deşi pornesc de la acelaşi principii constructive enumerate mai sus, nu au întotdeauna acoperire biologică deplină şi plauzibilă. Asemănarea cu modelele biologice este, prin urmare, limitată şi nuanţată. Dr. Teuvo Kohonen, inventatorul tipului de reţele neuronale care îi poartă numele, arată, în legătură cu acest subiect, într-un interviu:

   > “nu este neapărat necesar ca reţelele artificale să imite biologia. Ca analogie, roata care se învârte liberă nu apare în natura biologică pentru că în acest caz ar fi dificilă vindecarea. Vasele de sânge nu pot trece prin articulaţie”.

1.3 Tipuri de reţele neuronale

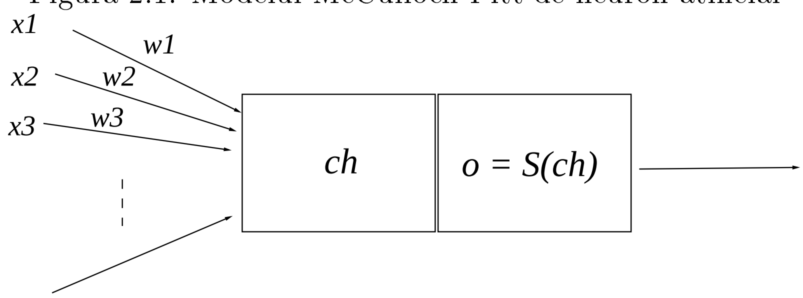
1.4 Terminologie

Capitolul 2

Reţele neuronale cu propagare înapoi

Fie următoarea reţea neuronală cu un număr de n straturi din Figura 1. Notăm pachetul de ponderi între straturile m şi $m+1$ cu w_m. Valoarea ponderii conexiunii între neuronul i de pe stratul m şi neuronul j de pe stratul $m+1$ va fi notată w_{mij}. Înlăuntrul unui neuron, vom păstra, în implementare, suma ponderată a intrărilor adunată cu valoarea pragului într-o valoare *cache* numită $ch_{m,k}$, unde m este numărul stratului iar k numărul neuronului în cadrul stratului. Valoarea *cache*, trecută prin funcţia-sigmoid va fi de asemenea păstrată în neuron în locaţia numită $o_{m,k}$, cu indicii m şi k păstrând semnificaţiile pomenite. $o_{m,k}$ este deci ieşirea neuronului şi poate fi scrisă şi ca $S(ch_{m,k})$, unde S este funcţia sigmoid (de obicei o funcţie de tipul $S = \frac{1}{1-e^{-x}}$ cu valori în intervalul $[0, 1]$ sau $S = \frac{2}{1-e^{-x}} - 1$, cu valori în intervalul $[-1, 1]$.

Figura 2.1: Modelul McCulloch-Pitt de neuron atificial



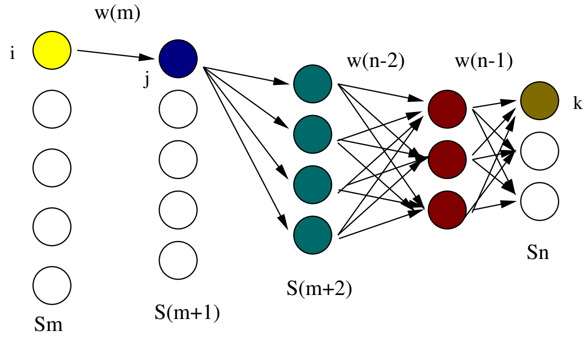
Activarea neuronului, memorata inlauntrul sau intr-o locatie *cache* numita *ch* este:

$$ch = \sum_i x_i \cdot w_i$$

Ieşirea neuronului este funcţie de această activare, trecută printr-o *funcţie de activare* de tip sigmoid notată cu S, după cum am aratat mai sus.

În mod evident, vor exista $n-1$ matrice de ponderi, numerotate de la 0 la $n-2$ în felul următor: w_{mij}, având semnificaţia ponderii conexiunii existente între neuronul nr. i de pe stratul m de neuroni şi neuronul nr. j de pe stratul de neuroni $m+1$.

Figura 2.2: Schema unei reţele neuronale total conectate cu propagare înapoi. Notaţii.



2.1 Relaţii în funcţionarea unei RN cu propagare înapoi

În atribuirea următoare, exponentul t are semnificaţia momentului de timp la care este surpinsă valoarea variabilei respective (deci nu are semnificaţie de putere). Actualizarea valorii ponderii se face după atribuirea:

$$w_{mij}^{t+1} = w_{mij}^{t} + c \cdot \Delta w_{mij} \,,$$

unde $\Delta w[m]_{ij}$ se calculează după formula următoare:

$$\Delta w_{mij} = -\frac{\partial E^t}{\partial w_{mij}}$$

Dacă notăm cu $\#n$ numărul de neuroni din stratul n mai puţin unul (pentru a respecta convenţia limbajului C prin care elementele unui vector cu n elemente sunt numerotate de la 0 la $n-1$), şi convenim să numerotăm ultimul strat cu n, E este eroarea pătratică, calculabilă după formula:

$$E = \frac{1}{2}\sum_{i=0}^{\#n}(d_j - o_j)^2 \qquad (1)$$

Ne interesează aşadar derivata parţială (dependenţa) erorii E de ponderea conexiunii $w[m]_{ij}$. Ea poate fi scrisă în felul următor:

$$\frac{\partial E}{\partial w_{mij}} = \frac{\partial E}{\partial o_{n-1}} \cdot \frac{\partial o_{n-1}}{\partial w_{mij}}.$$

Această scriere ne permite să ne ocupăm de cazul general al dependenţei ieşirii unui neuron dintr-un strat oarecare (în cazul cel mai interesant, cel final), faţă de o pondere oarecare din matricele de conexiuni dintre straturi.

Ne interesează, aşadar, derivata parţială (dependenţa) funcţiei-ieşire a neuronului k din stratul n faţă de ponderea w_{mij} :

$$\frac{\partial o_{nk}}{\partial w_{mij}} = \frac{\partial o_{nk}}{\partial o_{m+1,j}} \cdot S'_{m+1}(ch_{m+1,j}) \cdot o_{mi}$$

Formula anterioară transformă o dependenţă de pondere într-o dependenţă de ieşire a neuronului corespunzător ponderii.

Problema revine la a calcula, la modul general, dependenţa după formula recursivă:

$$\frac{\partial o_{nk}}{\partial o_{mj}} = \sum_{i=0}^{\#(m+1)} \frac{\partial o_{nk}}{\partial o_{m+1,i}} \cdot S'_{m+1}(ch_{m+1,i)} \cdot w_{mji} \qquad (2)$$

În cazul special (de ieşire din recursie) în care $n = m$, derivata parţială nu se mai calculează după formula de mai sus ci întoarce direct valoarea 1 dacă $k = j$ sau 0 altfel.

În continuare ne interesează dependenţa erorii totale E de o pondere oarecare. Notaţiile sunt: n pentru stratul final, w_{mij} cu semnificaţia amintită mai sus .

$$\frac{\partial E}{\partial w_{mij}} = \sum_{k=0}^{\#n} \frac{\partial E}{\partial o_{nk}} \cdot \frac{\partial o_{nk}}{\partial w_{mij}}$$

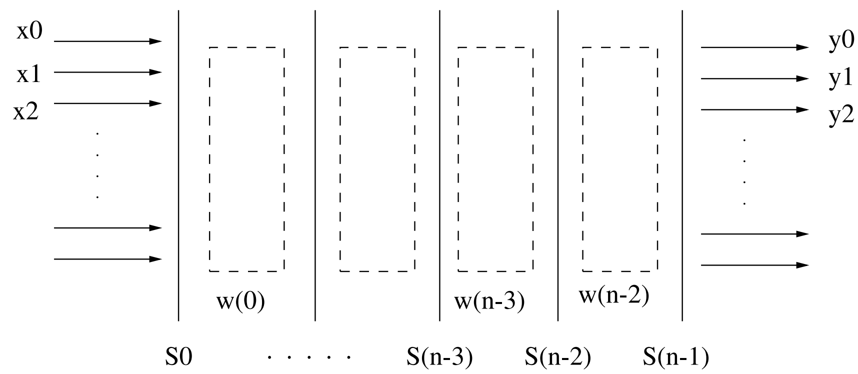
Şi cum expresia funcţiei-eroare E ne este cunoscută din ecuaţia 2.1, iar ecuaţia 2.1 ne ajută de asemenea, rezultă:

$$\frac{\partial E}{\partial w_{mij}} = \sum_{k=0}^{\#n} (d_k - o_k) \cdot \frac{\partial o_{nk}}{\partial o_{m+1,j}} \cdot S'_{m+1}(ch_{m+1,j}) \cdot o_{mi}$$

Problema, cu această abordare este că recurenţa — recalcularea ponderilor dintr-un strat pe baza celor din stratul imediat următor este mai greu de implementat.

2.2 Recurenţă cu ajutorul semnalului de eroare

Figura 2.3: Schema alternanţei între straturi de neuroni şi matrice de ponderi:



Câteva explicaţii în ce priveşte notaţiile pentru Figura 2.3: cele n straturi S sunt numerotate de la 0 la $n-1$. Matricele de ponderi w sunt figurate între straturile aferente şi sunt în număr de $n-1$, fiind aşadar numerotate în stil C de la 0 la $n-2$. Vom folosi pentru consistenţă indicii j pentru ultimul strat, q pentru penultimul şi i pentru antepenultimul strat de neuroni.

Există însă un model interesant în [?]: calculul recurent pe baza unei mărimi numite *semnal de eroare* care poate fi folosită în recurenţă foarte elegant.

Mai întâi să observăm că în relaţia:

$$w_{mij}^{t+1} = w_{mij}^{t} + c \cdot \Delta w_{mij}\,,$$

termenul-diferenţă Δw_{mij} este egal cu $-\frac{\partial E}{\partial w_{mij}}$.

Aşa cum am văzut mai sus, dacă ţinem minte că $o_{n-1,j} = S(ch_{n-1})$ — adică ieşirea este valoarea obţinută prin activarea ch trecută prin funcţia-sigmoid — dependenţa erorii totale de ponderea unei conexiuni $w_{n-2,ij}$ ce leagă ultimul strat de penultimul poate fi scrisă ca:

$$\begin{aligned}\Delta w_{n-2,qj} &= -\frac{\partial E}{\partial w_{n-2,qj}} = \frac{\partial E}{\partial o_{n-1,j}} \cdot \frac{\partial S}{\partial ch_{n-1,j}} \cdot \frac{\partial ch_{n-1,j}}{\partial w_{n-2,qj}} \\ &= -\left[-(d_j - o_{n-1,j})\right] \cdot S_{n-1}^{'}(ch_{n-1,j}) \cdot o_{n-2}\end{aligned}$$

Notăm cu e_k diferenţa între rezultatul dorit şi ieşirea reală, $d_k - o_k$.

$$\Delta w_{n-2,qj} = e_j \cdot S^{'}(ch_{n-1,j}) \cdot o_{n-2,q}\,.$$

Dacă definim semnalul de eroare $\delta_{n-1,j} = e_j \cdot S^{'}(ch_{n-1,j})$, pentru ultimul strat, ajustarea ponderilor dinaintea ultimului strat capătă expresia familiară de la perceptron — vedem $o_{n-2,q}$ ca intrările stratului $n-1$:

$$w_{n-2,qj}^{t+1} = w_{n-2,qj}^{t} + c \cdot \delta_{n-1,j} \cdot on - 2, q$$

Acum nu ne rămâne decât să calculăm ajustarea penultimului strat de ponderi. Ne apropiem de soluţie:

$$\begin{aligned}\Delta w_{n-3,iq} &= -\frac{\partial E}{\partial w_{n-3,iq}} = \\ &= -\frac{\partial o_{n-2}}{\partial ch_{n-2}} \cdot \frac{\partial ch_{n-2}}{\partial w_{n-3,i}} \cdot \sum_{j=0}^{\#(n-1)} \frac{\partial E}{\partial o_{n-1,j}} = \\ &= S^{'}(ch_{n-2,q}) \cdot o_{n-3,i} \cdot \sum_{j=0}^{\#(n-1)} e_j \cdot S^{'}(ch_{n-1}) \cdot w_{n-2,qj}\end{aligned}$$

Şi cum $e_j \cdot S'(ch_{n-1,j}) = \delta_{n-1,j}$, rezultă că, dacă acceptăm notaţia $\delta_{n-2,q} = S'(ch_{n-2,q}) \cdot \sum_j^{\#(n-1)} \delta_{n-1,j} \cdot w_{n-2,qj}$ (cu alte cuvinte, semnalul de eroare curent este derivata funcţiei sigmoid aplicată activării neuronului înmulţită cu suma ponderată a semnalelor de eroare care provin din stratul imediat următor), avem:

$$\Delta w_{n-3,iq} = \delta_{n-2,q} \cdot o_{n-3,i} \,,$$

adică diferenţa cu care trebuie să ajustăm ponderea arată şi de data asta ca un produs între semnalul de eroare aferent stratului penultim $\delta_{n-2,q}$ şi intrarea în acest strat penultim, care este totodată ieşirea stratului care îl precedă, de ordin $n-3$: $o_{n-3,i}$.

Formula suportă recurenţă şi în cazul mai multor straturi ascunse.

Observaţie foarte importantă: Avem relaţia deosebit de interesantă: $\delta_{mj} = \frac{\partial E}{\partial ch_{mj}}$, baza calculului recurent al semnalelor de eroare.

Apendice: Structura fişierului de salvare a ponderilor unei reţele CNet

Implementarea unei reţele neuronale cu backpropagation se face folosind clasa CNet, descrisă în fişierele net.h şi net.cpp.

După o perioadă de antrenare, utilizatorul poate dori ca ponderile reţelei să fie păstrate într-un fişier pentru a evita reantrenarea reţelei într-o folosire ulterioară. Acestui scop este dedicată metoda Save a clasei CNet. Ponderile sunt salvate într-un fişier binar (am preferat acest format pentru că un fişier text, chiar dacă are avantajul indiscutabil al lizibilităţii, pierde din exactitatea reprezentării numerelor reale). Formatul fişierului folosit este următorul:

IdentifierString (7 bytes) este şirul "Weights";

VersionNumber (2 bytes) care identifică numărul de versiune a programului şi deci a tipului de document;

FloatSize (1 byte) indică dimensiunea tipului de date în virgulă mobilă folosit (poate fi 4 pentru float, de exemplu, sau 8 pentru double);

Strata (2 bytes) memorează numărul de straturi ale reţelei;

NeuronsPerStratum(i) (Strata * 2 bytes) reţine numărul de neuroni pe fiecare strat. Este, în mod evident, un câmp cu lungime variabilă;

Urmează apoi, de $NeuronsPerStratum - 1$ ori, matricea propriu-zisă de ponderi, stocată linie după linie, cu dimensiunea

Weights(i) (NeuronsPerStratum(i) * NeuronsPerStratum(i+1) * FloatSize bytes) sunt ponderile propriu zise

2.3 Îmbunătăţiri ale algoritmului standard backprop

2.3.1 Metoda momentului

2.4 Algoritmul Quick-prop

Publicat în 1988 de Scott Fahlman, articolul *An Empirical Study of Learning Speed in Back-Propagation Networks*, face un studiu asupra procesului de învăţare în reţelele backprop, observă câteva neajunsuri şi propune unele soluţii care, chiar dacă nu sunt foarte riguroase, sunt considerate a fi foarte eficiente.

Algoritmul este, într-adevăr, mai degrabă empiric decât riguros dovedit. El se bazează pe presupunerea că toate porţiunile învecinate unui minim sunt asemănătoare unei curbe de gradul II. Odată acceptată această presupunere, coborârea pe pantă nu se mai face iterativ — proces îndelungat uneori — ci se sare direct la minimul presupusei curbe de gradul II. Chiar dacă este doar o aproximaţie, pentru antrenamente repetate, algoritmul se dovedeşte a fi foarte eficient. Autorul este constrâns să facă:

> "...două presupuneri riscante: în primul rând, aceea că se poate apoxima curba erorii ca funcţie de fiecare pondere printr-o parabolă cu braţele deschise în sus; în al doilea rând, că schimbarea pantei curbei erorii, aşa cum e văzută de fiecare pondere, nu este afectată de celelalte ponderi care se schimbă şi ele în acelaşi timp. Pentru fiecare pondere, independent, folosim pantele de eroare precedentă şi curentă şi modificarea ponderii între punctele în care aceste pante au fost măsurate, pentru a determina o parabolă; apoi, sărim direct la punctul de minim al acestei parabole. Calculul este foarte simplu şi foloseşte numai informaţie locală ponderii ce va fi modificată.[1]"

Formula propusă de autor este, într-adevăr, foarte simplă:

$$\triangle w(t) = \frac{S(t)}{S(t-1) - S(t)} \cdot \triangle w(t-1)\,,$$

unde $S(t-1)$ şi $S(t)$ sunt valorile curentă şi precendentă ale derivatei $\frac{\partial E}{\partial w}$.

Demonstraţia acestei figuri nu este prea complicată: se cunosc două tangente la o parabolă, distanţa dintre abscisele punctelor de tangenţă şi se cere abscisa distanţei dintre cel de-al doilea punct şi vârful parabolei, aşa cum este ilustrat în figură. Avem trei ecuaţii, pe care le alăturăm într-un sistem (presupunem că ecuaţie curbei de gradul II se poate scrie $ax^2 + bx + c = 0$) :

[1]Fahlman, Scott E., *An Empirical Study of Learning Speed in Back-Propagation Networks,* p. 10

$$\begin{cases} 2ax_0 + b = S(t-1) \\ 2a\left[x_0 + \triangle w(t-1)\right] + b = S(t) \\ 2a\left[x_0 + \triangle w(t-1) + \triangle w(t)\right] = 0 \end{cases}$$

Din ultima ecuaţie se deduce că $\triangle w(t) = -\frac{2a[x_0+\triangle w(t-1)]+b}{2a}$. Dar, din cea de-a doua ecuaţie ştim că numărătorul egalităţii precedente este chiar $S(t)$, deci rămâne $\triangle w(t) = -\frac{S(t)}{2a}$. Mai departe, dacă scădem prima ecuaţie din a doua şi o prelucrăm, obţinem egalitatea $2a = \frac{S(t)-S(t-1)}{\triangle w(t-1)}$. Unind cele două rezultate, obţinem expresia dorită: $\triangle w(t) = \frac{S(t)}{S(t-1)-S(t)} \cdot \triangle w(t-1)$.

2.5 Algoritmul Cascade-correlation (CASCOR)

Articolul *The Cascade-Correlation Learning Architecture*, publicat in 1991 de Fahlman[2]şi Lebiere, propune o idee originală şi eficientă pentru instruirea reţelelor neuronale. El urmează unui studiu aprofundat asupra funcţionării structurilor neuronale organizate pe straturi, în special cu propagare înapoi[3]. Metoda presupune ca unităţile să fie adăugate într-o reţea pe parcursul antrenării astfel încât fiecare unitate nou adăugată să încerce să preia maximum din eroarea deja existentă. Lucrul acesta se face prin urmărirea covarianţei între valoarea neuronului candidat la introducere şi eroarea sesizată la ieşire. Printr-o metodă asemănătoare cu cea din algoritmul backprop clasic (ascensiune pe curba covarianţei), se găseşte maximul (local, eventual al) acestei covarianţe, unitatea este introdusă în reţea cu valoarea găsită şi — în mod original — ponderile ce intră în această unitate nu se mai modifică ulterior. Prin această tactică, se urmăreşte reducerea efectului denumit de autori *moving target.*

> “Pe scurt, problema este că fiecare unitate din interiorul reţelei încearcă să devină un detector al unei caracteristici care va avea un rol important in calculul total al reţelei, însă acest lucru este complicat în mare măsură de faptul că toate unităţile se schimbă în acelaşi timp. Unităţile ascunse dintr-un strat oarecare al reţelei nu pot comunica între ele în mod direct; fiecare unitate vede numai intrările sale şi semnalul de eroare propagat înapoi spre el de la ieşirea reţelei. Semnalul de eroare defineşte problema pe care unitatea încearcă să o rezolve, dar această problemă se schimb în mod constant. În locul unei situaţii în care fiecare unitate merge rapid şi direct înspre a-şi asuma un rol folositor, vedem un dans complex între toate unităţile, care are nevoie de mult timp pentru a se linişti[4].”

[2]Articolul, împreună cu câteva alte lucrări ale lui Scott E. Fahlmann (CMU-CS-90-100), poate fi găsit pe site-ul ftp al Universităţii Carnegie-Mellon.

[3]Cf. *An Empirical Study of Learning Speed in Back-Propagation Networks*, Fahlmann, 1988. CMU-CS-88-162

[4]*The Cascade-Correlation Learning Architecture*, p. 2

Algoritmul combină deci "structura cascadată, în care unităţi ascunse sunt adăugate una câte una şi nu se schimbă după ce au fost adăugate" cu "maximizarea corelării între ieşirea unităţii şi semnalul de eroare care trebuie eliminat[5]".

Mecanismul de funcţionare al reţelei este deci următorul: se începe doar cu stratul de intrare şi cu cel de ieşire, fără nici un strat ascuns. Se antrenează reţeaua până când atenuarea erorii devine neglijabilă, moment în care se iau un candidat sau mai mulţi şi se conecetează cu intrările la unităţile deja existente în reţea (cu excepţia celor de ieşire). Ieşirea componentelor-candidat nu este conectată la ieşire, rămâne în gol, deocamdată. Se face apoi antrenament pentru toate exemplele de antrenament *(samples)* cu scopul de maximiza S, covarianţa lui V şi E (adică valoarea unităţii-candidat şi eroarea observată la ieşire).

$$S = \sum_o \left| \sum_p \left(V_p - \overline{V}\right) \left(E_{p,o} - \overline{E_o}\right) \right|$$

Scopul este de a maximiza aceată covarianţă. Maximizarea se face asemănător cu algoritmul backpropagation, calculând derivata de ordinul întâi şi urcând pe pantă:

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o \left(E_{p,o} - \overline{E_o}\right) f_p^{'} I_{i,p} \,,$$

unde σ_o este semnul corelaţiei între valoarea candidatului şi ieşirea o, $f_p^{'}$ este derivata pentru exemplul p aplicată activării candidatului, iar $I_{i,p}$ este intrarea pe care o primeşte candidatul de la unitatea i pentru exemplul p.

[5]Idem, p. 3

Bibliografie

[1] Dumitrescu, D.; Costin Hariton — *Reţele neuronale. Teorie şi aplicaţii*, Teora, 1996, Bucureşti.

[2] Neagoe, Victor; Stănăşilă, Octavian — *Recunoaşterea formelor şi reţele neurale*, Matrix Rom, 1998, Bucureşti.

[3] Toderean, Gavril; Coşteiu, Mircea; Giurgiu, Mircea — *Reţele neuronale*, Microinformatica, Cluj, 1994.

[4] Fahlman, Scott E.; Christian Lebiere — *The Cascade-Correlation Learning Architecture*, 1991, CMU-CS-90-100.

[5] Fahlman, Scott E. — *An Empirical Study of Learning Speed in Back-Propagation Networks*, 1988, CMU-CS-88-162.

[6] Neural Network Frequently Asked Questions, maintained by Warren S. Sarle. Poate fi găsit la adresa `ftp://ftp.sas.com/pub/neural/FAQ.html`.

[7] Davalo, Eric — *Introduction to Neural Networks.*