Universitatea „Lucian Blaga“ din Sibiu
Facultatea de inginerie „Hermann Oberth“
Catedra de Calculatoare şi Automatizări

Evaluarea filtrării informațiilor prin utilizarea unei ontologii de domeniu

(Metaclasificator bazat pe rețea neuronală)

Referat de doctorat nr. 3

Autor:
mat. Radu CREȚULESCU

Coordonator:
Prof. univ. dr. ing. Lucian N. VINȚAN

SIBIU, 2009

Cuprins

1 Introducere şi obiective principale

Majoritatea informațiilor din lumea reală se găsesc în format text. Aceste date sunt considerate ca având un format semistructurat, deoarece nu conțin metainformații despre structura lor. Modelarea şi implementarea de tehnici pentru lucrul cu date semistructurate au crescut continuu în ultimii ani. Mai mult decât atât aplicațiile de regăsire a informațiilor ca şi metode de indexare a documentelor de tip text au fost adaptate astfel încât să funcționeze cu aceste documente nestructurate.

Tehnicile tradiționale de regăsire a informației devin astfel inadecvate pentru căutarea în colecții mari de date nestructurate. De obicei, doar o mică parte din documentele disponibile sunt relevante, la un moment dat pentru utilizator. Fără a şti ce conțin aceste colecții mari de date, este dificil de a formula interogări eficiente pentru analiza şi extragerea de informații „interesante“. Astfel că, în ultimii ani utilizatorii au nevoie de tot mai multe unelte pentru a compara diferite documente din punct de vedere al gradului de relevanță şi utilitate precum şi găsirea de reguli pentru organizarea lor.

Clasificarea de text este un proces general, care include numeroşi paşi care trebuie executați pentru a rezolva această problemă. Fiecare dintre aceşti paşi are o influență majoră asupra acurateței finale de clasificare. În acest referat, prezint contribuțiile mele în dezvoltarea şi îmbunătățirea unui metaclasificator bazat pe clasificatoare de tip SVM şi Naive Bayes. Acest metaclasificator va cuprinde în etapa de postclasificare o rețea neuronală feed-forward cu învățare de tip Backpropagation.

Consider procesul de clasificare automată de documente ca o înlănțuire de etape (Fig. 1.1). Fiecare etapă primeşte la intrare anumiți parametri, îi procesează şi îi transmite mai departe următoarei etape. În acest referat m-am axat mai mult pe ultima etapă din acest proces, cea a metaclasificatorului. Ca şi componente ale metaclasificatorului am inclus mai mulți clasificatori de tip SVM, prezentați în [Mora06], şi un clasificator de tip Naive Bayes dezvoltat şi prezentat în [Cret08].

În capitolul 2 am dezvoltat şi prezentat mai mulți metaclasificatori neadaptivi, care folosesc diferite metode de ponderare a rezultatelor întoarse de fiecare clasificator în parte pentru calcularea clasei corespunzătoare documentului primit de către metaclasificator la intrare. Etapele de preprocesare pentru crearea vectorilor de intrare pentru metaclasificatori sunt

prezentate în [Mora08]. În cadrul metaclasificatorului final aceşti metaclasificatori neadaptivi (selectori) vor avea un rol de preclasificare. În cele ce urmează propun un metaclasificator format dintr-un selector neadaptiv folosit în faza de preclasificare şi o rețea neuronală în faza de postclasificare, pe care îl voi evalua.

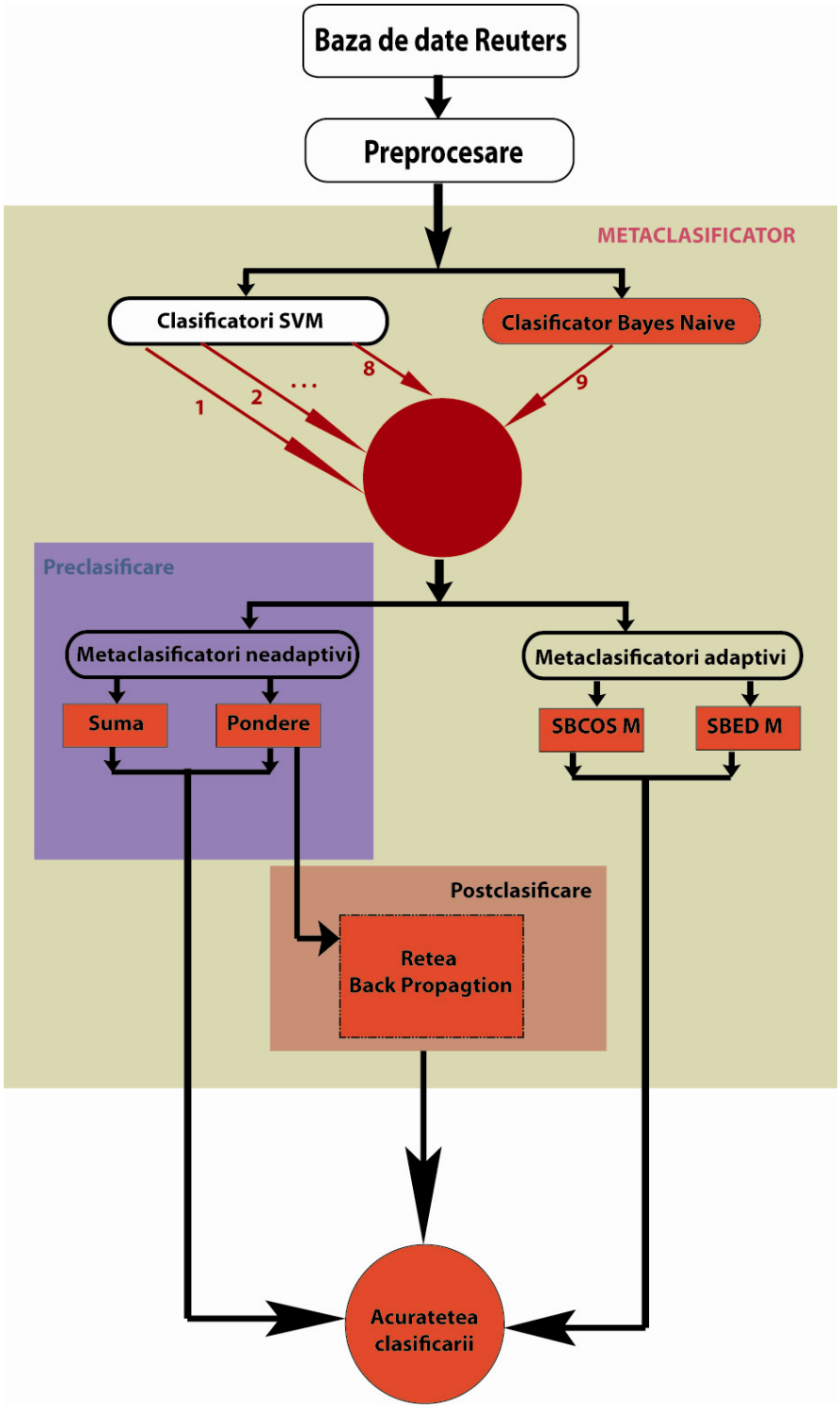


Fig. 1.1 Etape în procesul de clasificare automată de documente

În capitolul 3 am prezentat arhitectura rețelelor neuronale cu structura de tip feed-forward precum şi cunoştințele matematice de bază necesare pentru dezvoltarea unei rețele cu învățare

supervizata de tip backpropagation. Această rețea o voi utiliza în etapa de postclasificare din cadrul metaclasificatorului adaptiv. Rețelei i se vor prezenta la intrare un set de vectori de valori corespunzătoare pentru fiecare clasă generat de către selector iar la ieşire va prezice clasa corespunzătoare documentului curent. În finalul capitolului sunt prezentate rezultatele obținute în urma unor simulări efectuate utilizând diverse seturi de date

În ultimul capitol am prezentat o serie de concluzii extrase în urma analizei rezultatelor obținute pe baza de date Reuters [Reut00]. De asemenea sunt propuse câteva perspective de dezvoltare în acest domeniu.

2 Metaclasificatori în clasificarea de documente text

Metaclasificarea sau clasificarea hibridă se bazează pe predicția clasificatorului (algoritmului) corect pentru o problemă particulară, pe baza caracteristicilor vectorului de intrare şi a istoriei clasificărilor. Una dintre problemele principale care apare când sunt utilizați în practică algoritmi de clasificare este de a determina dacă clasificatorul obținut este fezabil şi pentru noi instanțe. Utilizarea metaclasificării este una dintre cele mai simple soluții de abordare ale acestei probleme. Având mai mulți clasificatori de bază, ideea este de a învăța un metaclasificator care prezice gradul de corectitudine pentru fiecare dintre clasificatorii de bază.

Selectarea unui clasificator pentru a eticheta o instanță se face în funcție de încrederea acordată acelui clasificator, încredere dobândită în urma clasificărilor corecte realizate de acesta pentru instanțe de tipul respectiv.

Regula de clasificare a metaclasificatorului este ca fiecare clasificator de bază să atribuie o clasă la instanța curentă cu o anumită probabilitate şi apoi metaclasificatorul să decidă dacă care clasificare este cea mai demnă de încredere. Pe lângă creşterea acurateței de clasificare prin exploatarea sinergismului mai multor clasificatoare, un alt avantaj al metaclasificării constă în posibilitatea de exploatare a paralelismelor funcționale (utilizând sisteme multiprocesor).

Clasificatorii de tip SVM şi de tip Bayes prezentați în [Cret08] au ca set de antrenare un număr de 4702 documente selectate din baza de date Reuters şi un set de testare de 2351 documente. Maximul acurateței de clasificare a fost obținut de clasificatorul de tip SVM, cu nucleu polinomial de grad 2 folosind reprezentarea Cornell Smart pentru date, atingând valoarea de 87,11%.

Mai mulți clasificatori de tip SVM şi un clasificator de tip Bayes au fost combinați pentru a crea un metaclasificator care să îmbunătățească acuratețea clasificării. Limita teoretică maximă a acurateței de clasificare la care se poate ajunge folosind această combinație de clasificatori este de 98,63%

Au fost implementate 3 tipuri de metaclasificatori: unul neadaptiv bazat pe votul majoritar şi două adaptive, bazate pe cozi de erori, unul folosind distanța euclidiană şi unul bazat pe cosinus. Dintre toți aceştia, metaclasificatorul bazat pe cosinus a îmbunătățit acuratețea de clasificare, ajungând la valoarea de 93,10%.[Cret08].

În cazul metaclasificatorilor adaptivi, există posibilitatea ca, după o perioadă de utilizare să apară suspiciunea ca, deşi un clasificator este ales la un moment dat ca fiind cel mai potrivit pentru clasificarea documentului curent, acesta să clasifice incorect acel document. În acest caz, se va alege clasa cu o valoare mai mică cu un prag ε=0.5 față de clasa cu valoarea cea mai mare dată de acel clasificator. Astfel, acuratețea clasificării finale a metaclasificatorului s-a îmbunătățit ajungând la 93,87% în cazul celui bazat pe cosinus. Având în vedere faptul că limita maximă prezentată mai sus este 98,63%, aceste rezultate obținute sunt încurajatoare.

În acest capitol voi prezenta realizarea unui nou metaclasificator. Acesta este realizat dintr-un metaclasificator neadaptiv care va folosi o sumă ponderată pentru stabilirea clasei finale urmat de un metaclasificator neuronal adaptiv. Acest metaclasificator neuronal utilizează un metaclasificator neadaptiv cu rol de preclasificare, şi o rețea neurală cu rol de postclasificare. Rețeaua neuronală va fi prezentată în capitolul următor.

2.1 Metaclasificator neadaptiv bazat pe sumă ponderată (Eurovision)

Metaclasificatorul, propus în continuare, conține cei 9 clasificatori utilizați în secțiunea anterioară şi pleacă de la premisa că ar conta şi numărul şi locul pe care apare fiecare clasă în parte. De exemplu în cazul a doi clasificatori şi 3 clase, dacă o clasă apare o dată pe locul 1 şi o dată pe locul 3 şi o altă clasă apare de 2 ori pe locul 2, este posibil ca cea de-a doua clasă să fie mai valoroasă, chiar dacă nu a obținut niciodată locul 1.

2.1.1 Metaclasificator neadaptiv bazat pe sumă (M-SUM)

În metaclasificator sunt incluşi 8 clasificatori de tip SVM şi unul de tip Bayes. Fiecare dintre aceştia produce un vector care conține 16 scalari, vezi Fig. 2.1. Fiecare scalar reprezintă valoarea funcției de decizie a clasificatorului pentru clasa respectivă. Până acum în [Mora08] se alegea întotdeauna valoarea cea mai mare şi clasa corespunzătoare a acestei valori era considerată clasa pe care o prezice clasificatorul respectiv. Pentru fiecare document în parte vom obține 9 astfel de vectori.

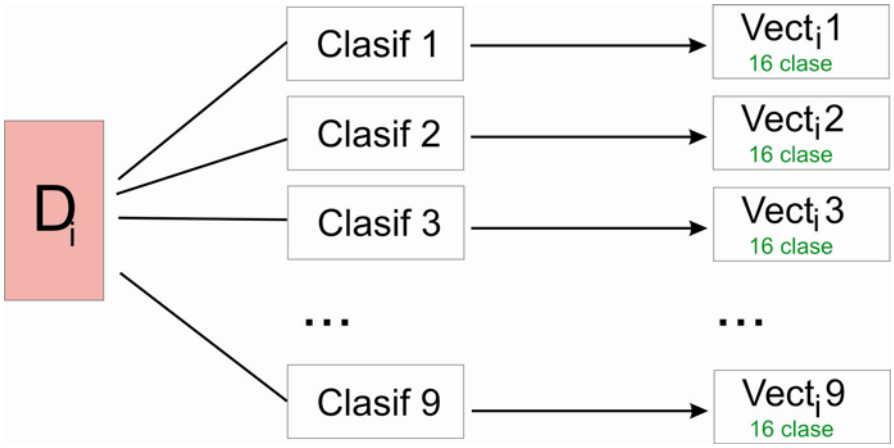


Fig. 2.1 Metaclasificator neadaptiv

Valorile funcțiilor de decizie pentru clasificatorii de tip SVM se află în intervalul (-∞,∞) dar în apropierea valorii 0, iar pentru clasificatorul de tip Bayes valorile se află în intervalul (-∞, 0). Având în vedere aceste diferențe şi pentru a putea realiza însumarea valorilor vectorilor, am transpus valorile vectorilor în intervalul [1, ∞).

$$V_i = V_i + \left|\min(\vec{V})\right| + 1 \qquad (2.1)$$

Astfel, pentru fiecare vector cea valorile lor de ieşire ai clasificatorilor de tip SVM se păstrează. La fel şi pentru clasificatorul de tip Bayes. Pentru a putea realiza însumarea acestor vectori în următorul pas am normalizat vectorii aducând valorile acestora în intervalul (0,1].

$$V_i = \frac{V_i}{\max(\vec{V})} \qquad (2.2)$$

În cazul metaclasificatorului care realizează doar sumele (numit in continuare M-SUM) am însumat cele 16 valori ale acestor 9 vectori, vezi Fig. 2.2, clasa câştigătoare fiind clasa cu valoarea cea mai mare obținută.

$$Class = \max_{c_i, i=1,16} \sum_{k=1}^{9} V_i[k] \qquad (2.3)$$

Acest metaclasificator, fiind unul neadaptiv, va obține întotdeauna acelaşi rezultat pentru o anumită instanță de intrare. În cazul rulării pe cele 2351 documente de test (setul T1 din [Cret08]) am obținut un număr de 313 documente clasificate eronat, care reprezintă o acurateţe a clasificării de 86,68%, cu 0,59% mai mare decât valoarea obținută folosind votul majoritar şi toți cei 9 clasificatori. Astfel putem concluziona că metoda bazată pe luarea în considerare doar a clasei învingătoare (*majority vote*) are dezavantaje față de metoda prezentată mai sus. În acest

caz, există şansa ca o clasă care poate niciodată nu a obținut locul 1 dar a obținut valori apropiate de maxim să fie în final clasa corectă.

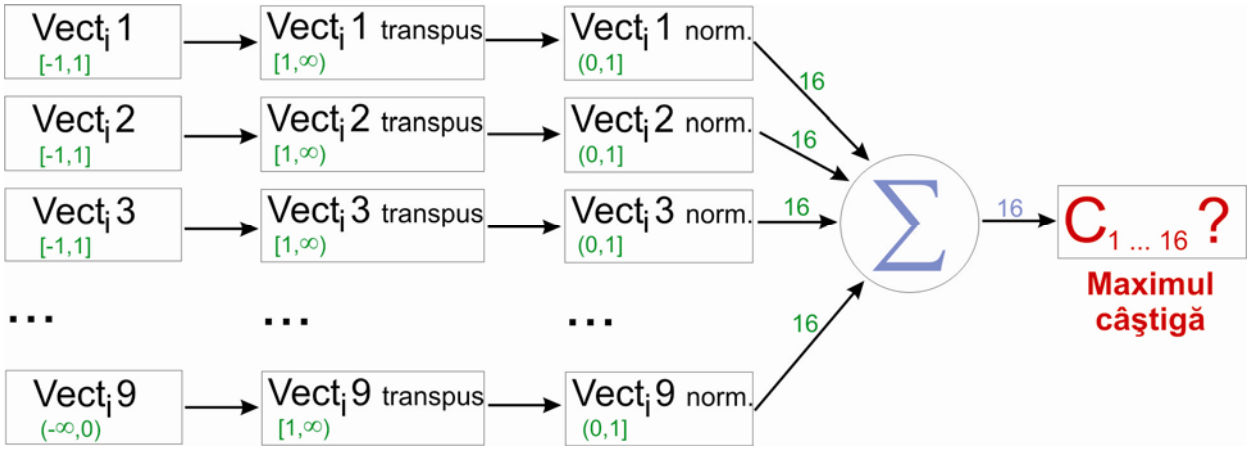


Fig. 2.2 Metaclasificator neadaptiv bazat pe sumă (M-SUM)

2.1.2 Metaclasificator neadaptiv bazat pe sumă normalizată (M-ESUM)

Această metodă are la bază metoda prezentată în secțiunea 2.1 doar că înainte de însumarea celor 16 valori din cei 9 vectori se realizează o ponderare a acestor valori. Astfel în cazul acestei ponderări vom atribui în fiecare vector pentru clasa de pe locul 1, valoarea 12, pentru clasa de pe locul 2 valoarea 10 iar în continuare pentru fiecare clasă de pe următoarele locuri valori descrescătoare până la valoarea 1. Astfel în fiecare vector clasele de pe primele 11 locuri vor avea valori diferite iar celelalte clase, până la 16, vor avea valoarea 1. Domeniul de reprezentare a valorilor vectorilor este {1, 2, 3, ...,10,12}, vezi Fig. 2.3. După această etapă se va realiza însumarea vectorilor şi selectarea clasei care obține valoarea cea mai mare, analog cu metoda anterior prezentată în par. 2.1.1.

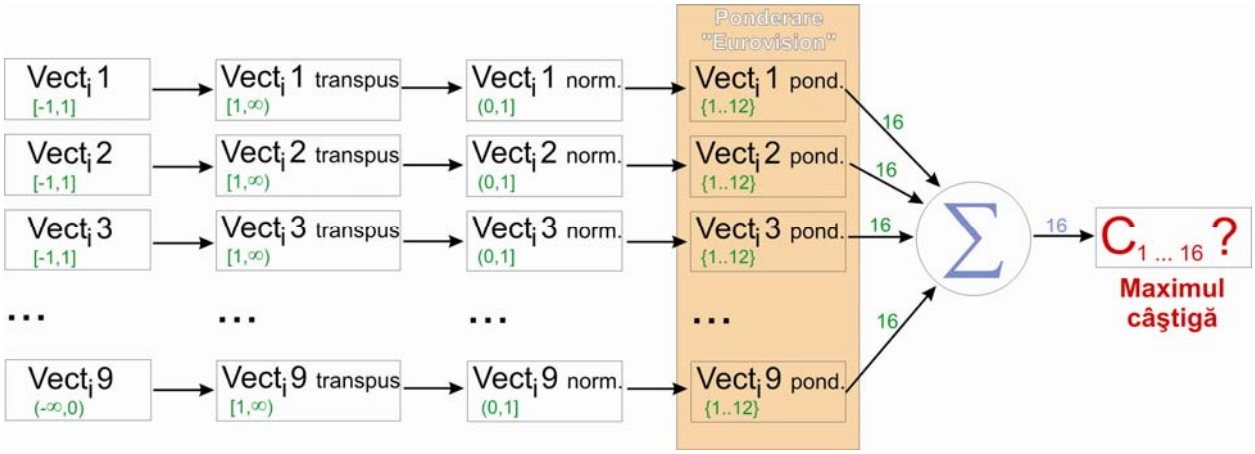


Fig. 2.3 Metaclasificator neadaptiv bazat pe sumă ponderată (M-ESUM)

În urma acestei ponderări, am obținut un număr de 316 erori de clasificare, ceea ce reprezintă o acuratețe a clasificării pe setul T1 de 86,55% pentru această metodă. Rezultatele obținute sunt cu 0,12% mai slabe decât cele obținute direct pe sumă.

2.1.3 Metaclasificator neadaptiv bazat pe sumă ponderată (M-WSUM)

În această secțiune introducem un nou metaclasificator neadaptiv bazat pe sumă ponderata (notat în continuare M-WSUM). Acest metaclasificator în pasul în care se realizează ponderarea, am decis ca în fiecare vector reprezentat ca în Par. 2.1.1, pentru clasa de pe primul loc, noua valoare să fie vechea valoare înmulțită cu 12. Pentru clasa de pe locul 2 va fi vechea valoare înmulțită cu 10, pentru locul 3 valoarea veche înmulțită cu 9 ş.a.m.d. pentru toate celelalte clase; valoarea minimă cu care înmulțim va fi 1. În cazul acesta domeniul de reprezentare al valorilor vectorilor este (0,12], vezi Fig. 2.4. Ca şi mai sus, în continuare se vor însuma valorile celor 9 vectori şi se va alege ca şi clasă învingătoare clasa care obține valoare maximă.

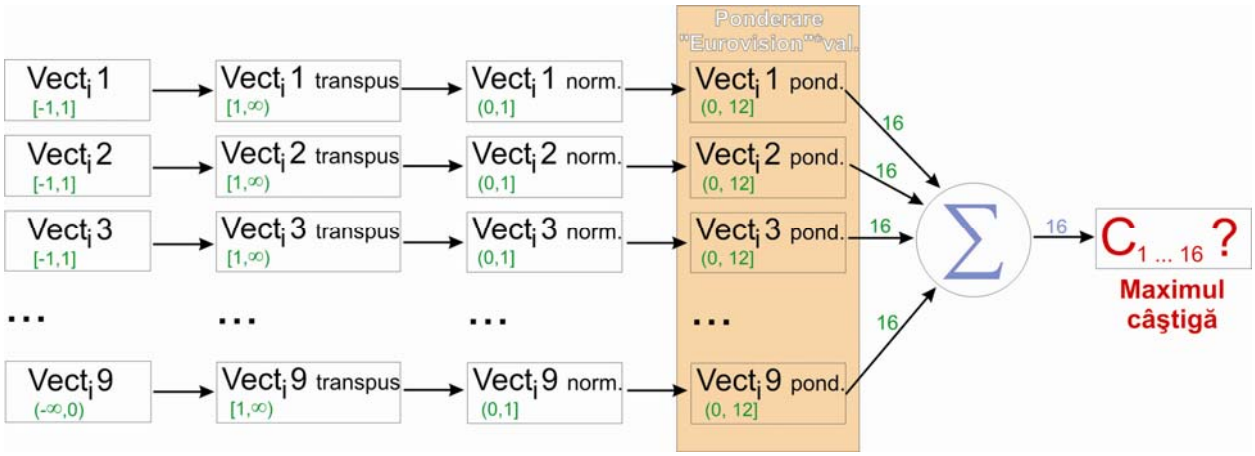


Fig. 2.4 - Metaclasificator neadaptiv bazat pe sumă ponderată (M-WSUM)

În acest caz am obținut un număr de 305 documente clasificate eronat pe setul T1, acuratețea clasificării pentru acest metaclasificator fiind de 87,02%. Această acuratețe de clasificare obținută este cea mai mare care a fost obținută prin utilizarea unui metaclasificator neadaptiv, dar evident mai mică decât limita maximă de 98,63%, la care poate ajunge teoretic metaclasificatorul.

2.1.4 Cercetări privind alte variante de ponderare a elementelor vectorilor

În această secțiune prezint câteva experimente efectuate asupra valorilor de ponderare pentru elementele celor 9 vectori rezultați în urma folosirii celor 9 clasificatori. Aceste valori reprezintă ponderea care este înmulțită cu valoarea obținută de o clasă în cadrul clasificatorului înainte de a realiza însumarea celor 9 rezultate.

2.1.4.1 Înjumătățirea ponderii (M-HW)

În primul experiment am ales ca valoarea de ponderare să se înjumătățească pentru fiecare clasă, clasele fiind în prealabil ordonate descrescător. Astfel, valoarea clasei de pe prima poziție se înmulțeşte cu constanta „16”, valoarea clasei de pe a doua poziție se înmulțeşte cu constanta „8”, cea de pe a treia poziție cu constanta „4” şi aşa mai departe, valorile claselor de pe ultimele 12 poziții se înmulțesc cu constanta „1”. În acest caz doar clasele de pe primele 4 poziții vor avea ponderi distincte, celelalte rămânând cu valoarea inițială. Ideea este de a favoriza foarte mult primele locuri. Rezultatul obținut de metaclasificator este de 324 documente incorect clasificate, ceea ce reprezintă o acuratețe a clasificării de 86,22%. Conform rezultatelor obținute, se observă că clasa corectă nu este întotdeauna prima clasă întoarsă de fiecare clasificator în parte. Această concluzie am formulat-o şi în cazul votului majoritar care a obținut o acuratețe de clasificare de 86,09% adică 327 documente clasificate incorect.

2.1.4.2 Ponderi mici descrescătoare linear

Pas 0,1 (M-0.1W)

În acest experiment pentru ponderi am ales valori mici, diferența dintre ele fiind de 0,1. Astfel, ponderea valorii clasei cele mai probabile va fi de 2,5 iar a celei mai puțin probabile va fi 1. Ideea este de a nu face o diferență foarte mare între clasele de pe diferite poziții, dar totuşi să favorizăm puțin clasa situată pe o poziție superioară. În acest caz, metaclasificatorul a avut un număr de 304 documente clasificate incorect, ajungând astfel la o acuratețe de clasificare de

87,07%. Alegerea ponderării distincte pentru fiecare loc cu valori apropiate este benefică în acest context.

Pas 1,0 (M-1.0W)

De aceea, în următorul experiment am ponderat clasele cu valori descrescătoare distincte cu pasul 1. Astfel, pentru prima poziție valoarea ponderii este de 16, pentru a doua poziție valoarea ponderii este de 15 ş.a.m.d. până la ultima poziție la care valoarea ponderii este de 1. În acest caz numărul de documente incorect clasificate de către metaclasificator a scăzut la 303, ceea ce reprezintă o acuratețe a clasificării de 87,11%.

Pas 0,5 (M-0.5W)

Totuşi, cele mai bune rezultate le-am obținut în cazul în care valorile ponderilor scad liniar cu un pas egal cu valoarea 0,5. Valoarea de prima poziție va fi ponderată cu valoarea 8,5 ş.a.m.d. descrescător până la ultima poziție unde valoare ponderii este 1,0. Astfel, numărul de documente incorect clasificate de către metaclasificator a scăzut la 301 rezultând o acuratețe a clasificării de 87,20%.

În Fig. 2.5 prezentăm comparativ rezultatele obținute în acest capitol.

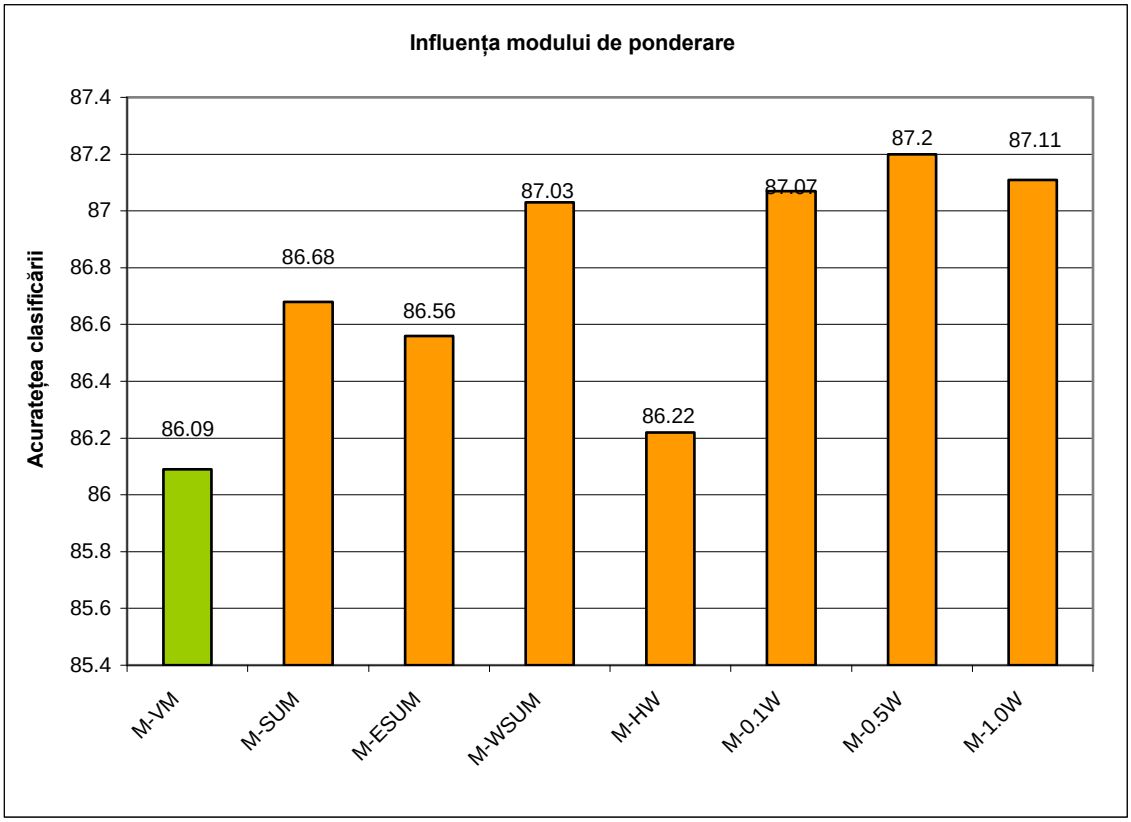


Fig. 2.5 - Comparație rezultate metaclasificatori neadaptivi

3 Metaclasificator bazat pe rețea neuronală

Deoarece metodele prezentate atât în capitolul anterior cât şi în [Cret08] nu obțin rezultate satisfăcătoare, am dezvoltat un metaclasificator care să îşi modifice comportamentul în funcție de datele de intrare mult mai dinamic decât metaclasificatoarele bazate pe distanță euclidiană şi cosinus. Pentru a face aceasta, am realizat un metaclasificator care în faza de postclasificare utilizează o rețea neuronală de tip backpropagation pentru a selecta clasa învingătoare.

În această metodă ca şi intrare pentru rețeaua neuronală vom avea vectorul de ieşire a preclasificării obținute în secțiunea 2.1.3. Am folosit aceşti vectori deoarece au obținut cele mai bune rezultate în etapa neadaptivă de preclasificare. Pentru antrenarea rețelei neuronale am folosit setul A1 de 4702 documente iar pentru testarea rețelei am folosit setul T1 de 2351 documente. Ambele seturi au fost inițial procesate folosind preclasificarea prezentată în secțiunea 2.1.3 şi pentru fiecare document obținem un vector de 16 elemente deoarece dispunem de 16 clase distincte. În consecință rețeaua neuronală de tip backpropagation cu un nivel ascuns pe care am dezvoltat-o are 16 neuroni pe stratul de intrare şi 16 neuroni pe stratul de ieşire. Pentru simplificarea problemei, pe stratul de ieşire am decis utilizarea tot a unui număr de 16 neuroni, rețeaua activând pentru un document doar unul din cei 16 neuroni de ieşire.

3.1 Postclasificare utilizând metoda Backpropagation

Prin dezvoltarea sistemelor inteligente, unele inspirate din rețelele neuronale biologice, au fost obținute numeroase soluții avantajoase. Cercetătorii din multe domenii ştiințifice proiectează rețele neuronale artificiale pentru a rezolva o varietate de probleme cum ar fi: recunoaşterea de patternuri, predicție, optimizare şi control etc. Abordări convenționale au fost propuse pentru rezolvarea acestor probleme. De asemenea, pot fi aplicate cu succes în foarte multe domenii care nu sunt suficient de flexibile pentru utilizarea altor metode. În acestea rețelele artificiale neuronale furnizează o alternativă viabilă [Hayk94].

În lungul curs al evoluției, creierul uman a dobândit multe trăsături care nu se regăsesc în modelul von Neumann sau în calculatoarele paralele moderne. Unele dintre aceste trăsături ar fi (conform [Jain96]):

- paralelism masiv
- reprezentare şi procesare distribuită
- abilități de învățare
- abilități de generalizare
- adaptabilitate
- procesarea a informației pe bază de context
- toleranță la erori
- consum redus de energie

Calculatoarele numerice actuale domină net omul în ceea ce priveşte prelucrările numerice. Totuşi, omul poate fără efort să rezolve unele probleme complexe de percepție şi recunoaştere a formelor cu o viteză incomparabil superioară celor mai performante calculatoare. Aceasta diferență provine din arhitectura complet diferită față de cea a maşinii von Neuman.

Inspirate din rețelele neuronale biologice, **R**ețelele **N**euronale **A**rtificiale (**RNA**) sunt sisteme de calcul cu paralelism masiv constituite dintr-un număr mare de elemente de procesare simple - numite **neuroni** - cu multe interconexiuni între ele. Modelele propuse pentru RNA respectă anumite principii de organizare presupuse ca fiind folosite în creierul uman.

Considerăm următoarele proleme de interes pentru domeniul ştiinței calculatoarelor şi ingineriei:

- **Clasificarea de patternuri** - problema este de a atribui unui pattern de intrare, reprezentat printr-un vector de trăsături una sau mai multe clase prespecificate.Ca şi aplicații binecunoscute amintesc recunoaşterea de caractere, clasificarea de documente, clasificarea celulelor sangvine etc.
- **Clustering/grupare** - cunoscută şi sub denumirea de clasificarea nesupervizată de patternuri în care nu avem date de antrenament la care să cunoaştem clasele. Algoritmul de clustering va exploata similaritatea dintre patternuri şi va plasa patternuri similare în acelaşi cluster. Ca şi aplicații amintim cele de compresie de date, analiza datelor şi data mining.
- **Aproximarea funcției** - presupunem un set de *n* date de antrenament etichetate, care au fost generate de o funcție necunoscută (susceptibile la zgomot). Problema este de a găsi o estimare cât mai exactă a funcției necunoscute.
- **Predicție/pronostic -** dându-se un set de *n* eşantioane preluate într-o secvență de timp, problema este de a prezice valoarea următorului eşantion. Spre exemplu această problemă are un impact semnificativ pe piața de capital.

- **Optimizare** - o varietate mare de probleme din matematică, statistică, inginerie şi economie sunt probleme de optimizare. Ideea acestui algoritm este de a găsi o soluție care satisface un set de constrângeri astfel încât funcția scop este maximizată sau minimizată.
- **Memoria adresabilă prin conținut -** în modelul von Neumann, o intrare în memorie este accesată doar prin intermediul adresei, care este independentă de conținutul memoriei. Mai mult decât atât, dacă se produce o eroare în calcularea adresei, se poate obține o valoare complet diferită. Memoria asociativă sau memoria adresabilă prin conținut poate fi accesată prin conținutul ei. Conținutul memoriei poate fi obținut chiar dacă avem o intrare incompletă sau un conținut distorsionat.

În evoluția RNA există trei perioade distincte. Prima are loc în anii '40, prin munca de pionierat a lui McCulloch şi Pitts. A doua perioadă, în anii '60, are la bază teorema lui Rosenblatt de convergență a perceptronului şi demonstrarea de către Minsk şi Papert a limitărilor perceptronului simplu. Abia începând cu anii '80 domeniul RNA şi-a redobândit interesul. Aceasta are la bază introducerea noțiunii de energie în rețeaua Hopfield în 1982 şi găsirea algoritmului de învățare cu retropropagarea erorii ("Backpropagation") pentru rețele cu propagare înainte ("feedforward") multistrat, propus inițial de Paul Werbos, în 1974, şi redescoperit şi popularizat de Rumelhart et al în 1986.[Maca03]

3.1.1 Modelul neuronului artificial

Modelul neuronului artificial [Wass89], [Kung93] are la bază modelul propus de McCulloch şi Pitts şi generalizat apoi în multe feluri. Prezentăm în continuare cea mai des întâlnită variantă.

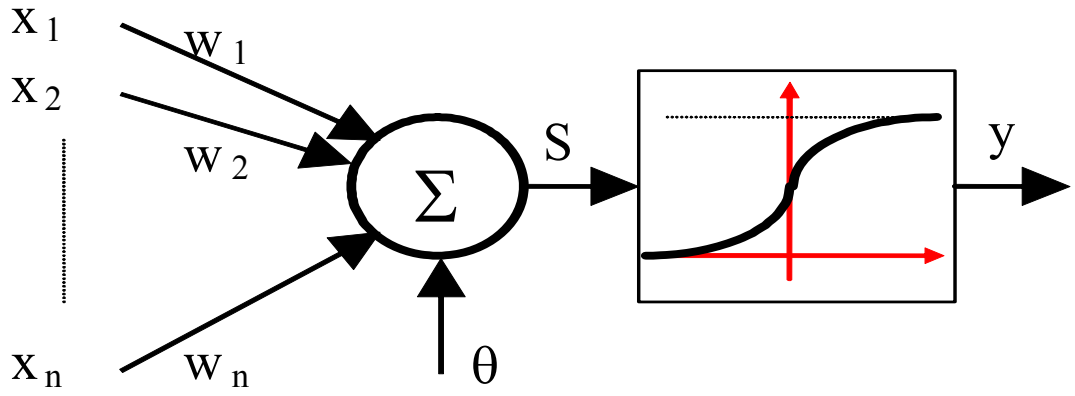


Fig. 3.1 Modelul neuronului artificial

Acest neuron artificial calculează suma ponderată a *n* semnale de intrare, adaugă o valoare numită prag şi apoi aplică acestei valori o funcție de activare generând ca ieşire o valoare cuprinsă în intervalul (0,1)

$$S = \sum_{i=1}^{n} x_i w_i + \theta \qquad y = f(S) \tag{3.1}$$

În aceste relații x_i reprezintă semnalul intrării i şi w_i sinapsa (ponderea, tăria sinaptică) asociată acestei intrări. Termenul θ reprezintă o valoare de prag (de offset, bias), care deplasează (transpune) ieşirea S a neuronului. Ieşirii S i se aplică o funcție de activare f care va transpune (normaliza) ieşirea neuronului în domeniul de valori dorit.

Există o analogie a acestui model cu neuronul biologic: interconectările modelează axonul şi dendritele, ponderile conexiunilor reprezintă sinapsele, iar funcția de activare aproximează activitatea din soma (corpul neuronului).

Modelul de neuron propus de McCulloch-Pitts a fost generalizat în mai multe feluri. Una dintre cele mai evidente modificări este utilizarea de funcții de activare în locul funcției de prag.

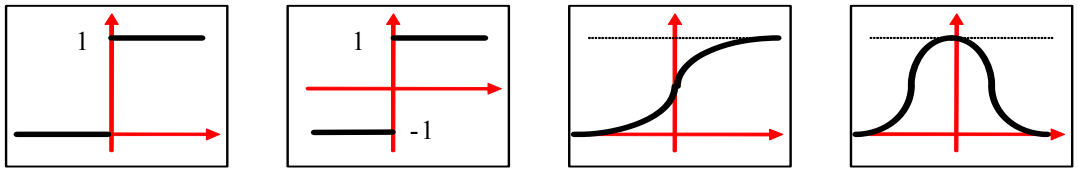
Pentru funcția de activare, cele mai des întâlnite funcții sunt cele prezentate în Fig. 3.2:

a - funcția de activare treaptă, $step\ (x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases}$

b - funcția de activare semn, $sign(x) = \begin{cases} +1, if\ x \geq 0 \\ -1, if\ x < 0 \end{cases}$

c - funcția de activare sigmoidală, $sigmoid(x) = \dfrac{1}{1+e^{-x}}$

d - funcția de activare gaussiană, $Gauss(x) = e^{-\frac{(m-x)^2}{2\sigma^2}}$



a b c d

Fig. 3.2 Funcțiile de activare cel mai frecvent întâlnite

Modelul neuronului prezentat anterior, având funcția de activare treaptă este modelul inițial propus de McCulloch şi Pitts în 1943. Cel mai popular model al neuronului a devenit însă cel cu funcția de activare sigmoidală, care este strict monoton crescătoare, mărginită şi derivabilă:

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \quad (3.2)$$

unde α este un factor de scară luând valori strict pozitive. Pentru α tinzând la infinit funcția sigmoidă devine funcția treaptă.

3.1.2 Arhitectura rețelelor neuronale

Există o varietate de tipuri de structuri de rețele, fiecare dintre acestea rezultă din diferite posibilități de calcul şi în funcție de problema care trebuie rezolvată. O rețea neuronală poate fi privită ca un graf orientat ponderat, în care neuronii sunt nodurile, iar arcele orientate (cu ponderile asociate) sunt legăturile între neuroni Din punct de vedere al construcției, rețelele neuronale se împart în două categorii principale: rețelele ***feed-forward*** şi rețelele ***recurente***.

- În rețelele **feed-forward** (cu propagare înainte), legăturile dintre neuroni sunt unidirecționale şi nu există bucle de reacție (legături de la un neuron de pe un strat superior la un neuron de pe un strat inferior). În rețelele feed-forward, legăturile pot pleca de la topologii arbitrare, neexistând nici o legătură spre stratul anterior sau legături care sar peste un anumit strat. În aceste tipuri de rețele, toți neuronii de pe un anumit strat sunt actualizați sincron, la o anumită perioadă de timp. Aceste rețele sunt rețele statice ele neavând un comportament dinamic propriu-zis, ieşirea rețelei depinzând doar de valoarea curentă a intrării, nu şi de valorile anterioare ale intrării.
- Rețelele **recurente** sunt rețele în care pot exista legături înapoi, un neuron de pe un strat superior având legături cu neuroni de pe nivelurile inferioare. De asemenea, în rețelele recurente pot exista legături care sar peste anumite straturi. În acest caz, rețeaua reprezintă un graf orientat complet şi are un comportament dinamic propriu-zis. În această categorie se încadrează rețelele competitive, hărțile topografice ale lui Kohonen, rețeaua Hopfield, rețeaua recurentă Elman, maşina Boltzmann şi modelele ART.

3.1.3 Învățarea rețelelor neuronale

Capacitatea de învățare este o trăsătură fundamentală a inteligenței. O definiție a învățării fiind dificil de dat, dar vom spune că, în contextul RNA, învățarea constă în modificarea rețelei pentru a-şi adapta comportamentul la necesitățile rezolvării unei probleme. În general sunt modificați coeficienții de conectivitate sinaptică, uneori modificându-se şi numărul de unități şi / sau configurația rețelei. Învățarea este importantă prin posibilitatea de adaptare la un mediu în schimbare (sistemele de IA clasice sunt rigide), fiind utilă şi în cazul în care mediul nu evoluează, dar pentru care nu dispunem de un model.

Principalul avantaj al RNA în raport cu sistemele expert clasice este acela că, în loc de a folosi un **set de reguli** date de un expert uman, are loc o **învățare prin exemple**.

Din punct de vedere al organizării datelor de intrare, există două categorii de învățare [Jain96]:

- învățarea ***nesupervizată,*** în care se prezintă rețelei doar datele de intrare fără a se specifica şi ieşirea dorită pentru acestea, astfel că rețeaua nu are nicio informație despre prezența sau valoarea erorii. În acest caz, rețeaua este lăsată să evolueze liber, urmând ca la sfârşit să constatăm rezultatul învățării. Rețeaua analizează corelațiile între datele de intrare şi organizează datele în categorii pe baza acestor corelații.
- învățarea ***supervizată,*** în care mulțimea de exemple de antrenament este organizată sub forma de perechi intrare-ieşire, specificând rețelei la fiecare pas care trebuie să fie ieşirea corectă, urmând ca rețeaua să generalizeze datele de intrare. Ponderile sunt modificate astfel încât rețeaua să producă ieşiri cât mai apropiate de răspunsul corect. Învățarea prin întărire ("reinforcement learning") este o variantă a învățării supervizate în care se furnizează rețelei doar o informație despre prezența erorii nu şi a valorii propriu zise a ei.

Fiecare tip de rețea îşi modifică ponderile în funcție de anumite reguli de învățare care depind atât de tipul datelor de intrare cât şi de modul de construcție al rețelei. Din punctul acesta de vedere, există patru tipuri consacrate de reguli de învățare principale:

- învățare prin corecția erorii;
- regula lui Boltzmann;
- regula lui Hebb;
- învățarea competitivă.

3.1.3.1 Reguli de învățare prin corecție a erorii ("error-correction rules")

În învățarea supervizată, rețeaua dispune de ieşirea dorită pentru fiecare vector de intrare. În timpul învățării, ieşirea rețelei nu este de obicei egală cu această valoare dorită. Principiul corecției erorii foloseşte semnalul de eroare pentru modificarea ponderilor în scopul minimizării erorii. Relația cea mai generală de modificare a unui coeficient sinaptic *w* conform acestei reguli este (evoluție în direcția gradientului)

$$\Delta \mathbf{w} = -\eta \frac{\partial E}{\partial \mathbf{w}} \tag{3.3}$$

unde *E* este eroarea globală (dependentă de *w*) şi η este viteza de învățare (mărimea pasului făcut pe direcția gradientului). Această relație stă la baza învățării în rețelele feed-forward multistrat.

Ideea de bază este de a utiliza *panta gradientului* pentru a căuta în spațiul ipotezelor de posibili vectori de ponderi pentru a găsi acele ponderi care aproximează cel mai bine exemplele de antrenament. Această regulă este importantă, deoarece furnizează bazele algoritmului Backpropagation, care este utilizat în cazul rețelelor cu multe unități interconectate.

Panta gradientului caută să determine vectorul pondere care minimizează eroarea pornind de la un vector pondere inițial arbitrar, care este apoi modificat repetat în paşi mici. La fiecare pas, vectorul pondere este modificat în direcția în care produce o pantă descendentă de-a lungul suprafeței erorii. Acest proces continuă până când eroarea minimă globală este atinsă.

Regula de învățare a perceptronului simplu propusă de Rosenblatt în 1962 foloseşte o variantă simplificată a regulii de minimizare a erorii. În acest caz avem

$$\Delta \mathbf{w} = \eta (d - y)\mathbf{x} \tag{3.4}$$

în care **w** şi **x** sunt vectorul ponderilor şi vectorul de intrare, *d* este ieşirea dorită şi *y* ieşirea reală.

Regula de învățare pentru rețeaua Adaline (strat de perceptroni cu ieşirea liniară), cunoscută şi ca Regula Widrow-Hoff, are şi ea la bază minimizarea erorii

$$\Delta w_{ij} = \eta\, x_j (T_i - y_i) \tag{3.5}$$

unde w_{ij} este ponderea legăturii ieşirii i cu intrarea *j*, *x* vectorul de intrare, *T* vectorul dorit la ieşire şi *y* vectorul ieşirii reale. Se poate demonstra că regula anterioară este o particularizare a regulii gradientului în cazul definirii erorii conform

$$E = \frac{1}{2}\sum_{i=1}^{N}(T_i - y_i)^2 \tag{3.6}$$

3.1.3.2 Regula de învățare Boltzmann

Maşinile Boltzmann sunt rețele recurente simetrice ($w_{ij} = w_{ji}$), constând din unități binare. Un subset al neuronilor rețelei sunt **vizibili** şi interacționează cu mediul (cei de intrare şi de ieşire) iar ceilalți sunt **ascunşi**. Starea unei ieşiri este 0 sau 1 cu o probabilitate

$$p_i = \frac{1}{1 + e^{-\frac{\tilde{x}_i}{T}}} \quad \text{unde} \quad \tilde{x}_i = \sum_{j \neq i} w_{ij} x_j - \theta_i \tag{3.7}$$

x fiind stările celorlalte unități, w_{ij} coeficienții sinaptici, θ valori de prag iar *T* "temperatura". Alegerea noii stări se face în concordanță cu probabilitatea p_i. Pentru a învăța asocieri între vectori de intrare şi vectori de ieşire se procedează astfel:

- rulare în **mod forțat** ("clamped") - pentru fiecare pereche de vectori intrare-ieşire se forțează unitățile de intrare şi ieşire la aceste valori rețeaua evoluând până la atingerea echilibrului termic. După atingerea acestui echilibru se determină probabilitatea ca două unități

să fie simultan active. Se repetă experiența pentru fiecare pereche de vectori intrare-ieşire. Se estimează $<ij>^+$ probabilitatea ca unitățile *i* şi *j* să fie active simultan când unitățile vizibile sunt forțate la valorile dorite.

- rulare în **mod liber** ("free") - se repetă paşii anteriori forțând însă doar unitățile de intrare, cele de ieşire fiind lăsate să evolueze liber. Se estimează $<ij>^-$ probabilitatea ca unitățile *i* şi *j* să fie active simultan când unitățile de ieşire sunt libere.

Ponderile coeficienților sinaptici se modifică apoi conform regulii de învățare Boltzmann

$$\Delta w_{ij} = \eta \, (\, <ij>^+ - <ij>^- \,) \qquad (3.8)$$

unde η este rata de învățare.

Regula de învățare Boltzmann poate fi privită ca şi un caz special de învățare prin reducere a erorii, în care eroarea nu este măsurată direct ci ca diferență a corelației între ieşiri în cele două moduri. Se încearcă astfel ca rețeaua să evolueze la fel atât în mod forțat cât şi liber.

3.1.3.3 Regula de învățare Hebb

Cea mai veche regula de învățare este postulatul lui Hebb, apărut în 1949 în „Organization of behavior"[Hebb49]. Aceasta are la bază observația neurobiologică:

Dacă ambii neuroni legați printr-o sinapsă sunt activi simultan coeficientul sinaptic al acestei legături creşte.

Matematic, regula lui Hebb poate fi descrisă astfel:

$$\Delta w_{ij} = \eta \, y_i x_i \qquad (3.9)$$

unde x_i şi y_j sunt activitățile celor doi neuroni *i* şi *j* conectați prin sinapsa w_{ij} şi η este rata de învățare.

Regula lui Hebb este plauzibilă biologic şi prezintă avantajul că învățarea se face în mod local, modificarea ponderii unei sinapse depinzând doar de neuronii alăturați ceea ce facilitează implementarea în circuite VLSI.

3.1.3.4 Regula de învățare competitivă

Spre deosebire de învățarea bazată pe regula lui Hebb (în care mai mulți neuroni pot fi activi simultan), în cazul învățării competitive între unitățile de ieşire are loc o competiție pentru activare. În final, o singură unitate va fi activă la un moment dat. Acest fenomen este cunoscut ca „învingătorul ia totul" („winner takes all").

Cea mai simplă rețea competitivă constă dintr-un singur strat de neuroni, fiecare conectat la vectorul de intrare. Fiecare neuron îşi stabileşte activarea după care, în urma unui proces de competiție, se determină un singur neuron i^* câştigător.

Regula de modificare a ponderilor sinaptice este:

$$\Delta w_{ij} = \begin{cases} \eta(x_j - w_{i^* j}) & i = i^* \\ 0 & i \neq i^* \end{cases} \quad (3.10)$$

Se observă că se modifică numai vectorul ponderilor legăturilor sinaptice al neuronului câştigător. Efectul aplicării acestei reguli de învățare este acela că vectorul *w* (memorat) se apropie de vectorul de intrare. Conform regulii de învățare competitivă rețeaua va termina învățarea (actualizarea ponderilor) doar în momentul în care rata de învățare η este 0. Un pattern de intrare particular poate activa diferite unități de ieşire la iterații diferite pe durata învățării. Aceasta duce la un comportament stabil al sistemului de învățare. Un sistem este stabil dacă nici un pattern din datele de antrenament nu-şi schimbă categoria după un număr finit de iterații de învățare. O metodă de a obține un sistem stabil este de a forța rata de învățare să descrească gradual pe parcursul procesului de învățare până când devine 0. Această înghețare artificială a învățării cauzează o altă problemă numită *adaptibilitate*, care reprezintă abilitatea unei rețele de a se adapta la noi date. Aceasta este cunoscută ca dilema „stabilitate-adaptabilitate" a lui Grossberg.

3.1.4 Perceptronul

Una din cele mai simple rețele neurale este perceptronul (o singură celulă). este prezentată în figura

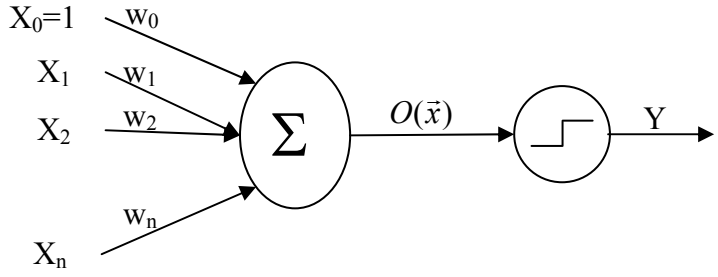


Fig. 3.3 Calcularea ieşirii perceptronului

$\overline{X} = \{x_0, x_1, ..., x_n\}$, reprezintă vectorul cu valorile de intrare,

$\overline{W} = \{w_0, w_1, ..., w_n\}$, reprezintă vectorul cu valorile ponderilor

$O(\overline{X}) = \overline{W} \cdot \overline{X} = w_0 + \sum_{k=1}^{n} w_k \cdot x_k$, reprezintă ieşirea

$Y = \begin{cases} +1 & if\ O(\overline{X}) > 0 \\ -1 & if\ O(\overline{X}) \leq 0 \end{cases}$, reprezintă semnul la ieşire.

Perceptronul poate fi considerat a fi reprezentarea unei suprafețe de decizie într-un hiperplan în spațiul n-dimenisonal al intrărilor. Ecuația acestui hiperplan de decizie este

$\overline{W} \cdot \overline{X} = \mathbf{0}$

Astfel perceptronul poate fi utilizat ca fi un clasificator binar sau un predictor (Taken = +1 or Not_Taken = -1). Bineînțeles acest perceptron poate clasifica corect doar un set de exemple ($\overline{X}$).care sunt linear separabile. De exemplu funcția logică XOR nu poate fi reprezentată de un singur perceptron.

Problema principală este cum să formulăm o regulă de învățare pentru un perceptron simplu pentru a învăța corect un set de vectori de antrenament pe care îl vom nota cu *D*. Dacă considerăm pentru fiecare exemplu (vector de antrenament) o regulă de învățare supervizată $d \in D$ este necesar să cunoaştem ieşirea corespunzătoare denumită t_d.

Dacă $O_d = w_0 + \sum_{k=1}^{n} w_k x_{dk}$ este ieşirea reală o măsură comună a erorii E este:

$$E(\overline{w}) = \frac{1}{2}\sum_{d \in D}(t_d - O_d)^2$$

Dată fiind formula pentru $E(\overline{w})$ suprafața trebuie să fie întotdeauna un paraboloid cu un singur minim global. Bineînțeles în particular $\vec{w}$ care dă minimul clasifică în cea mai bună măsură exemplul X_{dk}, *k=0,1,..,n*.. Gradientul $E(\overline{w})$ se notează

$$\nabla E(\overline{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right] = \sum_{k=0}^{n} \frac{\partial E}{\partial w_k} \bar{i}_k$$

unde $\bar{i}_k$ sunt vectorii unitate ortogonali in spațiul n dimensional. Se ştie că gradientul specifică direcția în care se produce cea mai rapidă micşorare a lui E. În acest caz regula de învățare ar fi

$\overline{W} \leftarrow \overline{W} + \Delta\overline{W}$, unde $\Delta\overline{W} = -\alpha \nabla E(\overline{W})$, α = rata de învățare (a un număr real mic pozitiv). Aceasta este echivalent cu :

$$w_k \leftarrow w_k - \alpha \frac{\partial E}{\partial w_k}, (\forall) k = 0,1,...,n$$

Dar: $\frac{\partial E}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{2}\sum_{d \in D}(t_d - O_d)^2\right) = \sum_{d \in D}(t_d - O_d)\frac{\partial(t_d - \overline{W} \cdot \overline{X})}{\partial w_k} = -\sum_{d \in D} x_{dk} \cdot (t_d - O_d)$

În final regula de învățare supervizată este:

$$w_k \leftarrow w_k + \alpha \sum_{d \in D}(t_d - O_d) x_{dk}, (\forall) k = 0,1,...,n$$

Această regulă se numeşte regula de gradient descendent sau regula delta. Implementarea algoritmului este descrisă mai jos [Vintan07]:

Initialize each W_k to random values $\in \left[-\frac{2}{n}, \frac{2}{n}\right]$

Until $E(\vec{w}) < T(threshold)$, DO

Initialize each $\Delta W_k = 0$

For each pair (x_d, t_d), from training examples, DO:

Compute O_d

For each W_k, DO:

$$\Delta w_k \leftarrow \Delta w_k + \alpha(t_d - O_d)x_{dk}$$

For each w_k, DO:

$$w_k = w_k + \Delta w_k$$

O idee alternativă este găsirea aproximării gradientului descendent prin actualizarea ponderilor incremental, urmat de calcularea erorii pentru fiecare exemplu de antrenament. O modalitate de a implementa stohastic acest gradient descendent este să considerăm eroarea distinctă $E_d(\vec{w}) = \frac{1}{2}(t_d - O_d)^2$

Utilizând aleator exemplele X_d obținem o aproximare rezonabilă a micşorării gradientului în comparație cu eroarea globală $E(\vec{w})$

Regula stohastică pentru gardientul descendent este:

Initialize each w_k randomly to $\left[-\frac{2}{n}, +\frac{2}{n}\right]$

Until the termination condition is met $(E_d(\vec{w}) < T \; or \; |O_d| > T, etc.)$, DO:

For each (x_d, t_d), DO:

Compute O_d

For each w_k, do:

$$w_k \leftarrow w_k + \alpha(t_d - O_d)x_{dk}$$

Regula standard a gradientului descendent este consumatoare de timp datorită însumării a multiplelor exemple dar se utilizează adesea cu o rată de învățare per exemplu mai mare decât rata de învățare per exemplu la regula stohastică cu gradientul incremental descendent. Dacă $\mathbf{E}(\overline{W})$ are multiple minime locale gradientul stohastic poate evita în unele cazuri oprirea în aceste minime locale deoarece utilizează diverse $\nabla \mathbf{E}_d(\overline{W})$ în găsirea minimului

Dacă considerăm ieşirea perceptronului $O(\overline{X}) = \text{sgn}(\overline{W} \cdot \overline{X})$ în locul $O(\overline{X}) = \overline{W} \cdot \overline{X}$ atunci această regulă se denumeşte regula de antrenare a perceptronului

$w_{\mathbf{k}} \leftarrow w_{\mathbf{k}} + \alpha(t - o)x_{\mathbf{k}}, \quad (\forall)k = 0,1,...,n$

Dacă exemplul de antrenament este corect clasificat (t=o) nu se actualizează nicio pondere. Presupunem acum o=-1 şi t = +1. În acest caz toate ponderile w_k cu valorile pozitive x_k vor fi incrementate iar celelalte ponderi w_k vor fi decrementate. Similar pentru o = +1 and t = -1 toate ponderile w_k cu valori x_k negative vor fi incrementate iar restul ponderilor w_k vor fi decrementate. Ca şi o regulă intuitivă dacă $\operatorname{sgn} t = \operatorname{sgn} x_k$ atunci w_k va fi incrementat iar altfel w_k va fi decrementat.

3.2 Metoda Backpropagation

3.2.1 Perceptroni multistrat cu funcție de activare neliniară

Perceptronii cu un singur strat de parametri modificabili nu pot clasifica decât mulțimi liniar separabile de vectori de intrare. Acest lucru a fost demonstrat încă din 1969 de către Minsky şi Papert şi a îndepărtat interesul cercetătorilor de rețelele neuronale. Se ştia de atunci că pentru perceptronii multistrat aceste probleme nu apar dar nu era clar cum să se modifice ponderile straturilor ascunse. Problema contribuției unităților interne a fost rezolvată şi diseminată pe scară largă abia în 1986 ducând la renaşterea interesului pentru rețelele neuronale.

3.2.2 Perceptronul multistrat

Cea mai populară categorie de rețele feed-forward multistrat este perceptronul multistrat în care fiecare unitate de calcul utilizează funcția de prag sau funcția sigmoidă. Perceptronii multistrat pot forma funcții de decizie complexe şi pot reprezenta orice funcție booleană. Dezvoltarea algoritmului de învățare backpropagation pentru determinarea ponderilor în perceptronii multistrat a făcut ca aceste rețele să devină foarte populare în cercetare.

O rețea feed-forward este un caz particular de rețea nerecurentă, în care neuronii sunt aranjați în straturi. Fiecare neuron primeşte intrarea doar de la neuronii de pe stratul anterior şi transmite ieşirea sa numai neuronilor de pe stratul următor, neexistând legături în cadrul unui strat. Problema pe care trebuie să o rezolve rețeaua este aceea de a învăța asocierea între vectori de intrare şi de ieşire. Rezolvarea „contribuției unităților interne” are la bază derivarea funcțiilor compuse.

Fie

$$x_i(k+1) = f\left(\sum_{j=1}^{N_k} w_{ij}(k) \cdot x_j(k)\right) \qquad (3.11)$$

activarea unității *i* din stratul *k+1*, N_k numărul de unități din stratul *k* şi *f* este funcția de activare.

Notăm $u_i(k+1)$ argumentul funcției *f,* deci

$$u_i(k+1) = \sum_{j=1}^{N_k} w_{ij}(k) \cdot x_j(k) \tag{3.12}$$

Pentru fiecare vector de ieşire eroarea globală este dată de

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(T_i - x_i\right)^2 \tag{3.13}$$

x_i fiind activitățile stratului de ieşire şi T_i valorile dorite la ieşire.

Numim eroare a unei unități:

- pentru ultimul strat

$$err_i = -(T_i - x_i) \cdot f'(u_i) \tag{3.14}$$

- pentru un strat ascuns *k*

$$err_i(k) = f'\left[u_i(k)\right] \cdot \sum_{j=1}^{N_{k+1}} \left[err_j(k+1) \cdot w_{ij}(k) \right] \tag{3.15}$$

Cu aceste notații se poate demonstra că modificarea parametrilor în direcția gradientului

$$\Delta w_{ij}(k) = -\eta \cdot \frac{\partial E}{\partial w_{ij}(k)} \tag{3.16}$$

devine, pentru toate straturile,

$$\Delta w_{ij}(k) = -\eta \cdot x_j(k) \cdot err_i(k+1) \tag{3.17}$$

Relațiile anterioare pun în evidență "retropropagarea erorii" ("backpropagation"). Ele sugerează ideea că informația de eroare de la ieşire se propaga înapoi prin rețea contrar sensului legăturilor sinaptice (lucru însă foarte puțin plauzibil a avea loc în rețelele neuronale biologice.)

Cu toată această, probabilă, îndepărtare de funcționarea rețelelor neuronale biologice regula backpropagation a făcut aceste rețele foarte populare ducând la renaşterea interesului şi utilizării rețelelor neuronale.

Întotdeauna trebuie învățate asocieri între mai mulți vectori de intrare şi de ieşire. În acest caz, funcția de eroare totală este suma funcțiilor de eroare corespunzătoare perechilor individuale intrare/ieşire. Această eroare poate fi minimizată în două moduri:

1 *off-line* - se determină, pentru fiecare pereche intrare/ieşire modificarea ce trebuie adusă coeficienților sinaptici. Aceste modificări se sumează şi se aplică numai după ce au fost prezentate toate perechile intrare/ieşire. Algoritmul realizează o optimizare deterministă după gradient a erorii totale.

2 *on-line* - modificarea coeficienților calculată pentru o pereche intrare/ieşire este aplicată imediat după prezentarea acestei perechi. Algoritmul realizează o optimizare după gradient pentru eroarea totală. Prezintă, în raport cu precedentul, avantajul că este în general mai rapid şi poate părăsi unele minime locale ale funcției de eroare totală.

În ceea ce priveşte parametrul η - mărimea pasului în direcția gradientului - acesta determină viteza de convergență spre un minim al erorii E. Când η este redus, convergența este lentă dar traiectoria urmează în mod fidel „relieful" funcției de eroare. Dacă E are minime locale, procedura deterministă poate eşua în acestea. Când η este mare, traiectoria nu mai urmăreşte fidel relieful funcției de eroare, ceea ce poate duce la imposibilitatea convergenței (salturi de o parte şi de alta a minimului căutat), dar permite uneori evadarea din minime locale. În practică se începe cu un η relativ mare, iar apoi, pe măsură ce rețeaua învață, această valoare se reduce treptat.

O interpretare geometrică poate ajuta la explicarea rolului neuronilor (cu funcție de activare) de pe stratul ascuns. Fiecare unitate din stratul de intrare formează un hiperplan în spațiul eşantioanelor de antrenament. Granițele dintre clasele eşantioanelor de antrenament pot fi aproximate de către hiperplane. O unitate de pe nivelul ascuns formează o hiperregiune pentru ieşirile unităților de pe primul nivel. O suprafață de decizie este obținută prin efectuarea unei operații AND între hiperplane. Unitățile de pe nivelul de ieşire combină suprafețele de decizie create de unitățile de pe nivelul ascuns prin efectuarea operațiilor OR logice. Acest scenariu este doar pentru a explica rolul unităților ascunse, iar comportamentul normal al rețelei, după ce rețeaua este antrenată, poate diferi.

De cele mai multe ori se utilizează un singur strat de neuroni ascunşi (rețele cu trei straturi), deoarece s-a demonstrat că o asemenea rețea (având un număr suficient de neuroni în stratul intermediar) poate aproxima oricât de bine orice funcție având un număr finit de discontinuități dacă funcțiile de activare ale neuronilor stratului ascuns sunt de tip sigmoidal.

3.2.3 Algoritmul Backpropagation

Algoritmul Backpropagation învață ponderile pentru o rețea pe mai multe nivele, dându-se o rețea cu o mulțime fixă de unități şi de interconexiuni. El utilizează panta gradientului pentru a încerca să minimizeze pătratul erorii dintre valoarea ieşirii rețelei şi valoarea țintă pentru acele ieşiri.

Problema învățării în Backpropagation este de a căuta în spațiul mare al ipotezelor definit de toate valorile posibile ale ponderilor pentru toate unitățile din rețea. Algoritmul Backpropagation este prezentat în continuare. Algoritmul descris aici se aplică rețelelor feed-

forward care conțin două nivele de unități, cu funcția de activare sigmoidă, fiecare unitate de pe un nivel fiind conectată cu toate unitățile de pe nivelul anterior. Aceasta este o versiune a algoritmului backpropagation care calculează, incremental sau stohastic, panta gradientului.

Backpropagation (*exemple_antrenament*, η, n_{in}, n_{out}, n_{hidden})

Fiecare exemplu de antrenament este o pereche de forma $\langle \vec{x}, \vec{s} \rangle$, *unde* $\vec{x}$ *reprezintă valorile vectorului de intrare şi* $\vec{s}$ *reprezintă valorile s ale vectorului de ieşire.*

η reprezintă rata de învățare care este o valoare (0,1]

n_{in} *numărul de neuroni de pe stratul de intrare*

n_{hidden} *numărul de neuroni de pe stratul ascuns*

n_{out} *numărul de neuroni de pe stratul de ieşire*

intrarea pentru unitatea i de la unitatea j este notată cu x_{ji} *şi ponderea de la unitatea i la unitatea j este notată* w_{ji}

- Se creează o rețea feed-forward cu n_{in} intrări, n_{hidden} unități ascunse şi n_{out}unități de ieşire
- Se inițializează toate ponderile din rețea cu valori aleatoare mici (de exemplu $\in \left[-\frac{2}{n_{in}}, \frac{2}{n_{in}} \right]$) [Vintan07]
- Până când condiția de terminare nu este îndeplinită execută (exemplu, eroarea....)
  - Pentru fiecare $\langle \vec{x}, \vec{s} \rangle$ *din exemple_antrenament* execută
    - Propagă semnalul forward prin rețea:
      1. se introduce instanța $\vec{x}$ în rețea şi se calculează ieşirea *y* pentru fiecare neuron din rețea
    - Propagă eroarea înapoi prin rețea - backward
      2. pentru fiecare neuron de ieşire din rețea se calculează eroarea conform formulei (3.14)
      3. pentru fiecare neuron de pe stratul ascuns se calculează eroarea conform formulei (3.15)
      4. calculează $\Delta w_{ji} = \eta \cdot err_i \cdot x_{ji}$ pentru nivelele anterioare prin propagarea backward a erorii
      5. se actualizează ponderile rețelei $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

Algoritmul este descris aici pentru o rețea feed-forward conținând două straturi de unități cu funcția sigmoidă de activare în cazul general (Fig. 3.4). Fiecare unitate de pe fiecare strat este conectată cu toate unitățile de pe stratul precedent. Unitățile de pe stratul de intrare sunt

considerate unități repetoare care prezintă la ieşire valoarea primită la intrare. De asemenea sunt prezentate formulele de calcul pentru această rețea atât pentru pasul forward cât şi pentru pasul backward. Pentru pasul backward s-a luat în considerare formula de calculul a erorii prezentată în ecuația

$$E = \sum_{o=1}^{n_{out}} \left(Out_3[o] - Scop[o]\right)^2 \quad (3.20)$$

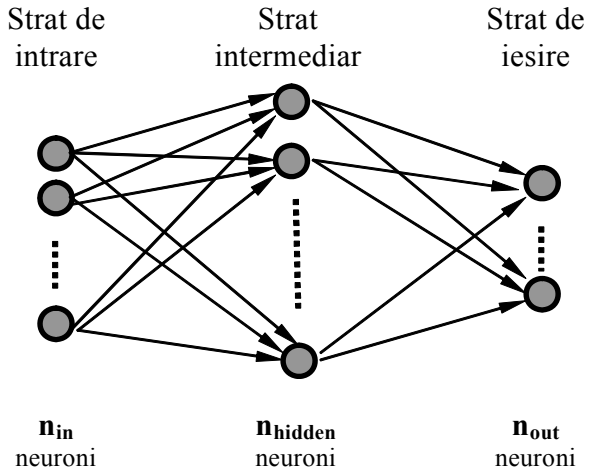


Fig. 3.4 - Arhitectura rețelei

Pentru acest caz, conform [Brea06], formulele generale date de regula backpropagation devin, cu notațiile următoare:

$w_{12}[h][i]$	ponderea legăturii neuronului **h** ("hidden") din stratul intermediar (2) cu neuronul **i** din stratul de intrare ("input") (1)
$\theta_2[h]$	valoarea de prag a neuronului **h** din stratul intermediar (2)
$w_{23}[o][h]$	ponderea legăturii neuronului **o** ("output") din stratul de ieşire (3) cu neuronul **h** din stratul intermediar (2)
$\theta_3[o]$	valoarea de prag a neuronului **o** din stratul de ieşire (3)
$Out_1[i]$	valoarea ieşirii neuronului **i** din stratul de intrare
$Out_2[h]$	valoarea ieşirii neuronului **h** din stratul de intermediar
$Out_3[o]$	valoarea ieşirii neuronului **o** din stratul de ieşire
Scop[i]	valoarea dorită la ieşire
F(.)	funcția de activare a tuturor neuronilor
n_{in}, n_{hidden}, n_{out}	numărul de neuroni din stratul 1, 2 respectiv 3

3.2.3.1 Pasul forward

$$\mathrm{Out}_2[\mathrm{h}] = \mathrm{F}(\sum_{\mathrm{i}=1}^{\mathrm{n_{in}}} \mathrm{w}_{12}[\mathrm{h}][\mathrm{i}] \cdot \mathrm{Out}_1[\mathrm{i}] + \theta_2[\mathrm{h}]) \quad (3.18)$$

$$\mathrm{Out}_3[\mathrm{o}] = \mathrm{F}(\sum_{\mathrm{h}=1}^{\mathrm{n}_{hidden}} \mathrm{w}_{23}[\mathrm{o}][\mathrm{h}] \cdot \mathrm{Out}_2[\mathrm{h}] + \theta_3[\mathrm{o}]) \quad (3.19)$$

$$\mathrm{E} = \sum_{\mathrm{o}=1}^{\mathrm{n_{out}}} \left(\mathrm{Out}_3[\mathrm{o}] - \mathrm{Scop}[\mathrm{o}]\right)^2 \quad (3.20)$$

3.2.3.2 Pasul backward

$\Delta w = -\eta \cdot \frac{\partial E}{\partial w}$, unde w poate fi $\theta_3[o]$, $w_{23}[o][h]$, $\theta_2[h]$, $w_{12}[h][i]$ (3.21)

$$\frac{\partial E}{\partial \theta_3[o]} = 2 \cdot (Out_3[o] - Scop[o]) \cdot F'(Out_3[o]) \quad (3.22)$$

$$\frac{\partial E}{\partial w_{23}[o][h]} = 2 \cdot (Out_3[o] - Scop[o]) \cdot F'(Out_3[o]) \cdot Out_2[h] \quad (3.23)$$

$$\frac{\partial E}{\partial \theta_2[h]} = 2 \cdot \sum_{o=1}^{n_{out}} (Out_3[o] - Scop[o]) \cdot F'(Out_3[o]) \cdot w_{23}[o][h] \cdot F'(Out_2[h]) \quad (3.24)$$

$$\frac{\partial E}{\partial w_{12}[h][i]} = 2 \cdot \sum_{o=1}^{n_{out}} (Out_3[o] - Scop[o]) \cdot F'(Out_3[o]) \cdot w_{23}[o][h] \cdot F'(Out_2[h]) \cdot Out_1[i] \quad (3.25)$$

Considerând pentru funcția F funcția sigmoidă clasică, derivata se determină uşor din valoarea funcției conform relației:

$$F'(x) = F(x) \bullet (1 - F(x)) \quad (3.26)$$

3.2.4 Rezultate privind evitarea saturării ieşirii neuronilor

Această problemă se pune în cazul în care neuronii din stratul de ieşire au funcția de activare sigmoidă. Pentru a adapta spațiul problemei la spațiul rețelei neuronale cea mai simplă soluție este translatarea domeniului valorilor componentelor vectorilor aplicați ieşirii rețelei [V_{MIN} ÷ V_{MAX}] printr-o funcție liniară în domeniul [L ÷ H] cu L = 0.0 şi H = 1.0 (având în vedere domeniul de ieşire al neuronilor cu funcții de activare sigmoide clasice).

Am comparat această primă variantă cu o a doua variantă, în care am realizat translatarea în domeniul [L ÷ H] cu L = 0.1 şi H = 0.9. Pentru comparație am ales o rețea feed-forward cu 2

neuroni de intrare, 2 neuroni în stratul ascuns şi un singur neuron de ieşire, cu funcții de activare sigmoide pentru toți neuronii, care să rezolve binecunoscuta problema XOR. Am folosit la antrenare metoda backpropagation clasică, învățare off-line şi rata de învățare constantă $\eta = 1$.

Evoluția comparativă a erorii rețelei în primii 4001 paşi este prezentată în Fig. 3.5.

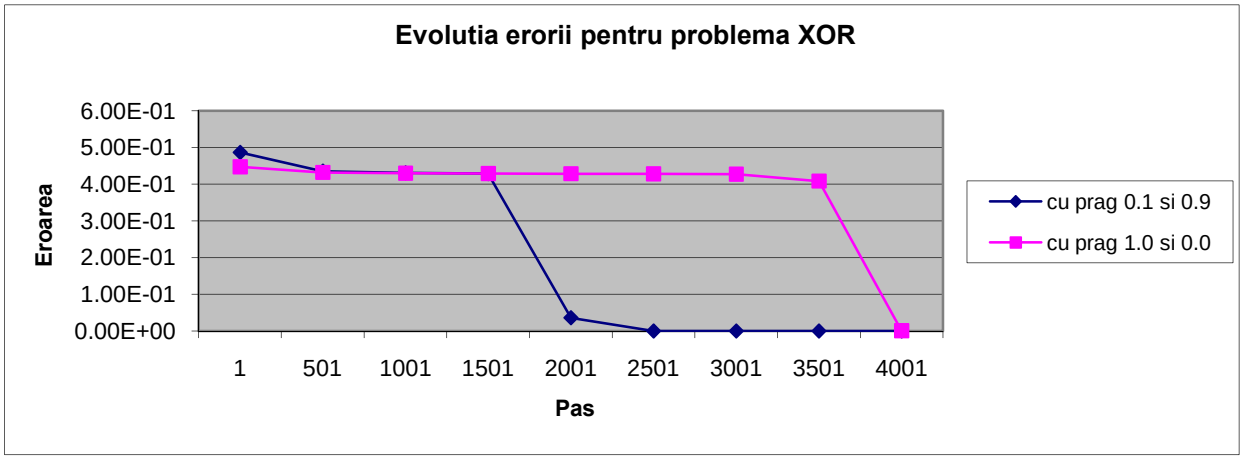


Fig. 3.5 Evoluția erorii în cazul problemei XOR

În primele etape de antrenament prima variantă duce la o învățare mai rapidă deoarece eroarea determinată de rețea este mai mare. În următoarele însă, pe măsura apropierii valorii ieşirii de valorile de saturație ale funcției sigmoide învățarea devine foarte lentă. În varianta a doua valorile dorite la ieşire se ating repede datorită evitării zonei de saturație a funcției sigmoide.

Rezultă deci că scalarea domeniului semnalului de intrare la domeniul $0.1 \div 0.9$ este benefică şi a fost aplicată în toate experimentele descrise în lucrare în care neuronii au funcția de activare sigmoidă.

3.3 Rezultate privind utilizarea rețelei BP în cadrul metaclasificatorului (M-BP)

Metaclasificatorul realizat se bazează pe o preclasificare de documente prezentată în secțiunea 2.1.3 şi o rețea neuronală de tip feed-forward cu învățare online. Am dorit să includem în metaclasificatorul **M-BP (Metaclasificator cu rețea BackPropagation)** o rețea neuronală deoarece am considerat că un metaclasificator adaptiv poate va reuşi să se „adapteze" şi la datele cu probleme, care există în setul de antrenare/testare. Rețelele neuronale sunt sisteme care se adaptează la schimbările survenite în seturile de date astfel că metaclasificatorul M-BP devine unul mult mai adaptiv decât metodele SBDE şi SBCOS dezvoltate şi prezentate în [Mor06].

Cel mai bun rezultat de până acum prezentat în lucrare s-a obținut cu ajutorul metaclasificatorului bazat pe cosinus, unde acuratețea de clasificare a atins valoarea de 93,87% pe setul de test T1 cu 2351 documente.

Pentru antrenarea şi testarea rețelei Backpropagation am plecat de la setul de vectori obținut de metaclasificatorul neadaptiv, prezentat în secțiunea 2.1.3. Am antrenat acel metaclasificator atât pentru setul de antrenament A1 (4702 documente) cât şi pe setul de test T1 (2351 documente) [Cret08] Ca şi intrare în acest metaclasificator avem setul de date, iar la ieşire, obținem un set de vectori, câte un vector pentru fiecare document de intrare, de 16 elemente fiecare. Setul de vectori obținut pornind de la setul de documente de antrenare A1, pe care îl vom numi în continuare setul AV1, va fi folosit în etapa de antrenare a rețelei. Setul de vectori obținut pornind de la setul de documente de testare T1, numit în continuare TV1, va fi folosit atât în etapa de testare cât şi în etapa de determinare a configurației rețelei.

În ceea ce priveşte arhitectura rețelei backpropagation, am ales una care conține două straturi de unități cu funcția sigmoidă de activare, iar fiecare unitate de pe fiecare strat este conectată cu toate unitățile de pe stratul precedent. Deoarece la intrare avem la dispoziție vectori de 16 elemente rețeaua va avea pe stratul de intrare 16 neuroni. La ieşire metaclasificatorul trebuie să „prezică” clasa în care se găseşte documentul curent. Atunci rețeaua Backpropagation va avea la ieşire tot un număr de 16 neuroni deoarece avem 16 clase distincte. În stratul ascuns avem un număr variabil de neuroni, alegerea acestui număr va fi făcut în funcție de rezultatele simulărilor care vor fi prezentate în secțiunea următoare (3.3.1).

În faza de antrenare, deoarece rețeaua este una cu învățare supervizată, pentru setul de antrenare am creat un set cu răspunsurile corecte pentru fiecare document în parte. Un astfel de răspuns conține valoarea „1” pe poziția clasei corecte şi valoarea „0” în rest. Structura metaclasificatorului adaptiv M-BP este prezentată în Fig. 3.6.

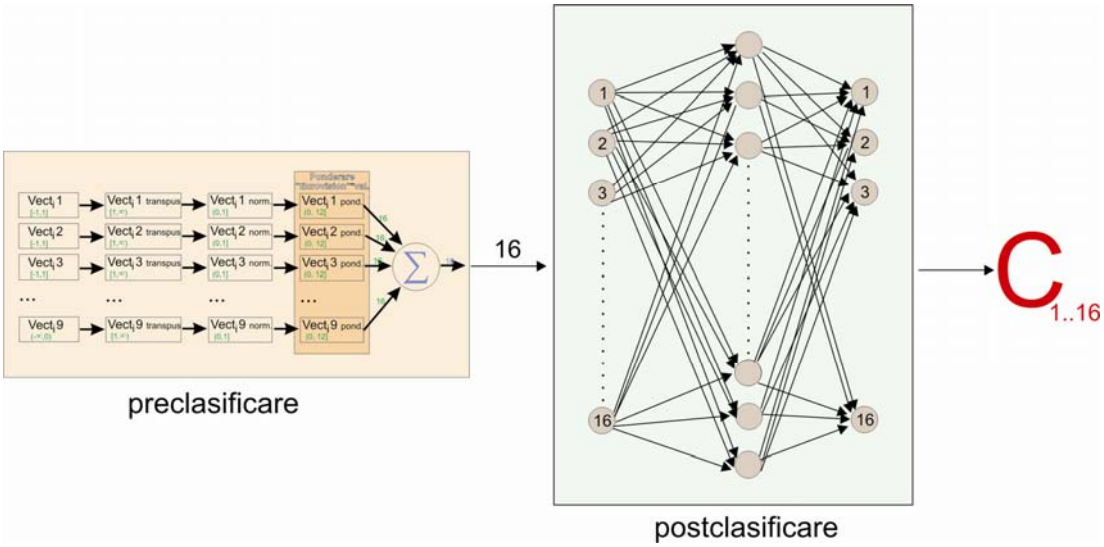


Fig. 3.6 Metaclasificator adaptiv M-BP

3.3.1 Influența numărului de neuroni de pe stratul ascuns

Deoarece nu există o formulă matematică pentru calcularea numărului optim de neuroni necesari pe stratul ascuns, în această secțiune prezentăm experimentele realizate în vederea determinării numărului optim de neuroni pentru rețeaua prezentată în Fig. 3.6. Experimentele prezentate în această secțiune sunt efectuate pe setul TV1, atât antrenarea cât şi testarea.

Ca şi metodă de evaluare, am oprit antrenarea rețelei după ce aceasta a ajuns din punct de vedere al erorii de antrenare la o anumită valoare, am evaluat rețeaua pe întreg setul din punct de vedere al numărului de documente incorect clasificate, după care am continuat antrenarea. Valorile erorii la care am oprit antrenarea rețelei sunt calculate ca fiind sumă a tuturor erorilor obținute pentru fiecare exemplu în parte din setul TV1. Pentru calculul erorii am folosit formula **Error! Reference source not found.**). Evaluarea rețelei în acel punct se face ca fiind numărul de documente incorect clasificate de către rețea. Având în vedere că setul TV1 conține 2351 vectori şi că eroarea pe fiecare vector reprezintă o sumă de 16 elemente, eroarea totală va avea valori supraunitare. Vom începe testarea pornind de la o valoare a erorii totale egală cu 500, ceea ce înseamnă o eroare medie pe fiecare vector de 0,21. Ideea este de ajunge cu eroarea de antrenare la o valoare cât mai mică într-un timp cât mai scurt.

În graficul următor (Fig. 3.7) prezentăm evoluția acurateței de clasificare a MC-BP în funcție de numărul de neuroni de pe stratul ascuns. Inițial am pornit de la un număr de 17 neuroni pe stratul ascuns. Toate experimentele prezentate în această secțiune au coeficientul de învățare η=1. În momentul în care timpul necesar rețelei pentru a reduce eroarea de antrenare devine mare am oprit antrenarea rețelei pentru acea configurație. Din acest motiv, în graficele

prezentate în continuare unele grafice nu coboară cu eroarea de antrenare până la valoarea minimă obținută de cea mai bună configurație testată.

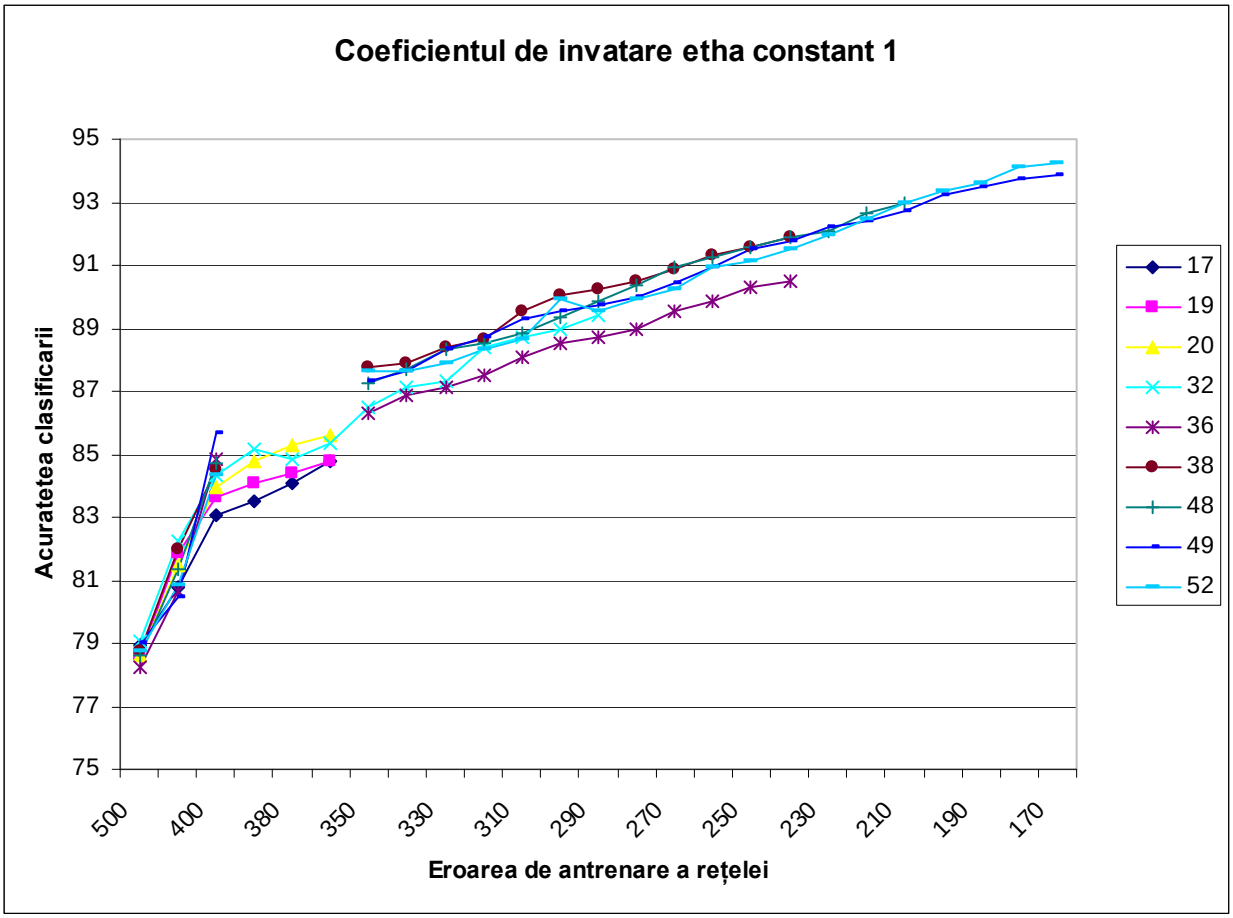


Fig. 3.7 Influența numărului de neuroni de pe stratul ascuns asupra acurateței de clasificare. Coeficient de învățare η=1

În acest grafic am început cu un număr mic de neuroni pe stratul ascuns pentru a avea un timp de învățare relativ mic (din punct de vedere al calculelor efectuate), dar, în momentul în care am ajuns la valori totale ale erorii de antrenare în jur de 200, timpul de antrenare creşte, iar, datorită coeficientului de învățare mare, rețeaua începe să fluctueze în jurul unei valori a erorii totale. Din acest motiv am oprit evaluarea rețelei la un număr de 52 de neuroni pe stratul ascuns chiar dacă acuratețea de clasificare creştea.

În secțiunea următoare prezentăm experimente în care modificăm şi pasul de învățare. În cazurile în care rețeaua este mai simplă (are un număr mai mic de neuroni pe stratul ascuns), de la un moment dat, eroarea totală a început să scadă foarte încet, moment în care am oprit antrenarea rețelei. De aceea, din acel punct nu vor mai fi valori în graficele pe care le prezint. Am modificat numărul de neuroni utilizând multiplii lui 16 (v. Fig. 3.8 şi 3.10). Pe măsură ce am crescut numărul neuronilor de pe stratul ascuns păstrând pas de învățare η=1, eroarea a scăzut de la 0,2 la 0,04 per exemplu. Cea mai bună valoare a acurateței de clasificare obținută până în acest

moment este de 94,26%, fiind deja superioară celei mai bune valori obținute cu metaclasificatorul de tip SBCOS cu 9 clasificatoare (93,32%).

Totuşi, o dată cu creşterea numărului de neuroni de pe stratul ascuns am observat că timpul de antrenare pentru rețea scade, chiar dacă numărul de calcule care trebuie efectuate creşte. În cazul în care avem mulți neuroni pe stratul ascuns rețeaua ajunge mai repede la o eroare mai mică iar fluctuațiile apar la valori mici ale erorii. Numărul mai mare de neuroni pe stratul ascuns duce la o micşorare mai rapidă a erorii datorită unei distribuții mai adecvate. Această convergență superioară compensează timpul necesar efectuării unui număr mult mai mare de calcule. De asemenea şi acuratețea clasificării creşte semnificativ.

3.3.2 Influența coeficientului de învățare

În continuare vom prezenta experimente efectuate pe acelaşi număr de neuroni pe stratul ascuns, dar cu un coeficient de învățare care scade în timp. Practic, oprim antrenarea rețelei la anumite valori ale erorii totale de antrenare, efectuăm testarea, şi continuăm antrenarea rețelei cu un coeficient de învățare micşorat. De exemplu, până la o valoare a erorii totale egală cu 350, coeficientul de învățare este „1”, între 340 şi 320 este „0,9”, între 320 şi 300 este „0,8” şi scade până la valoarea de „0,1” la o valoare a erorii de 150.

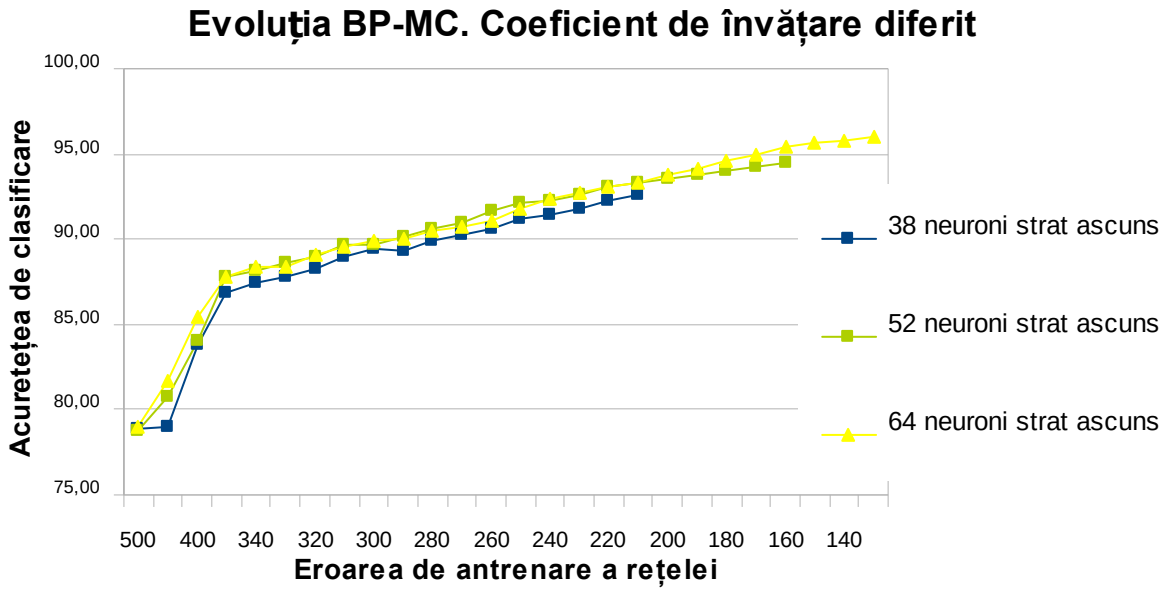


Fig. 3.8 Influența numărului de neuroni de pe stratul ascuns asupra acurateței de clasifcare. Coeficient de învățare η diferit

În Fig. 3.8 am prezentat doar evoluția rețelei pentru un număr de neuroni pe stratul ascuns egal cu 38, 52 şi 64 deoarece acestea au obținut rezultatele cele mai bune în cazul coeficientului de învățare constant. În acest grafic am coborât cu eroarea de învățare până la valoarea de 130

(coeficientul de învățare a ajuns la 0,01), deoarece rețeaua nu a mai fluctuat mult în jurul erorii şi astfel timpul de antrenare a fost redus. În acest caz, cu un număr de 52 de neuroni pe stratul ascuns numărul de vectori incorect clasificați pentru o eroare totală de antrenare egală cu 170 este de 136. În acest grafic am prezentat şi rezultate obținute pe o rețea cu 64 de neuroni pe stratul ascuns, caz în care eroarea totală de antrenare a scăzut la valoarea de „130”, iar numărul de documente incorect clasificate s-a redus la 95, ceea ce reprezintă o acuratețe de clasificare de 95,96%.

Am observat că, odată cu creşterea numărului de neuroni de pe stratul ascuns, se îmbunătățeşte acuratețea clasificării, deoarece putem ajunge la o eroare de antrenare mult mai mică. Am efectuat şi unele experimente în care numărul de neuroni de pe stratul ascuns este mai mare, regula de alegere a numărului de neuroni de pe stratul ascuns fiind multipli ai lui 16.

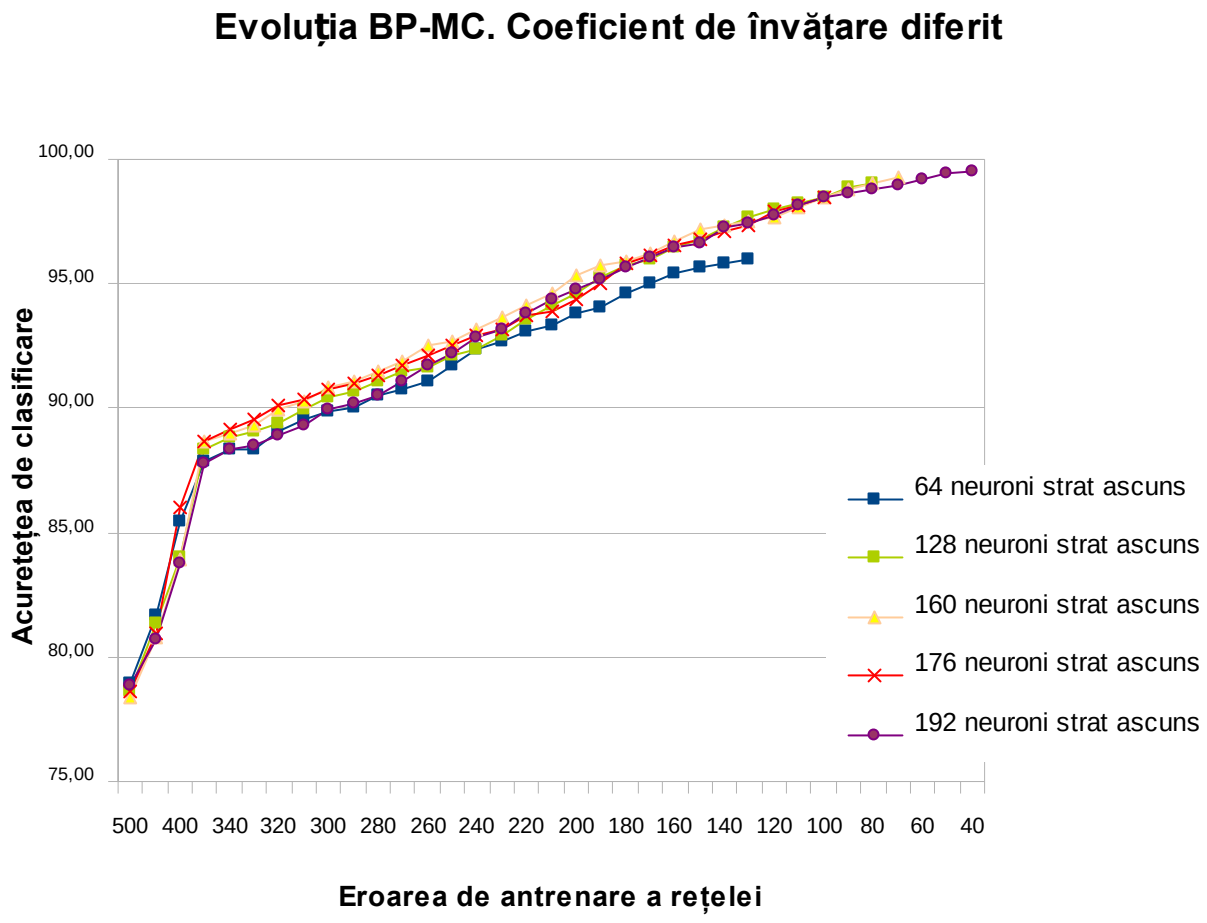


Fig. 3.9 Influența numărului de neuroni de pe stratul ascuns asupra acurateței de clasificare. Coeficient de învățare η diferit

În acest caz, arhitectura cu 160 de neuroni pe stratul ascuns a obținut cele mai multe rezultate bune, dar în momentul în care eroarea totală de antrenare a scăzut până la valoarea 70,

timpul de antrenare pentru ca eroarea să scadă la valoarea 60 a depăşit 24 ore. De aceea am realizat o arhitectură a rețelei cu 192 de neuroni pe stratul ascuns care a reuşit să coboare la o eroare de antrenare egală cu 40, caz în care numărul de documente incorect clasificate este de doar 11. Acest număr reprezintă o acuratețe a clasificării pentru metaclasificatorul M-BP de 99,53%. O eroare totală de antrenare egală cu 40 înseamnă o eroare medie per exemplu egală cu 0,017.

Experimentele prezentate au fost rulate pe un calculator P-IV dual core la 1.9GHz cu 2Gb DRAM şi sistem de operare Windows Vista. Prezentăm în Fig. 3.10 rezultatele comparative între arhitectura rețelei cu 52 neuroni pe stratul ascuns şi coeficient de învățare 1 şi respectiv aceeaşi arhitectură, dar coeficient de învățare descrescător în timp Pentru a ajunge la prima oprire (eroare 500) rețeaua are nevoie de mai mult timp deoarece porneşte de la o eroare mare dar care scade foarte repede. Timpii pentru următoarele opriri ale rețelei sunt timpii necesari rețelei pentru a ajunge de la valoarea erorii de la pasul curent la valoarea erorii de la următoarea oprire.

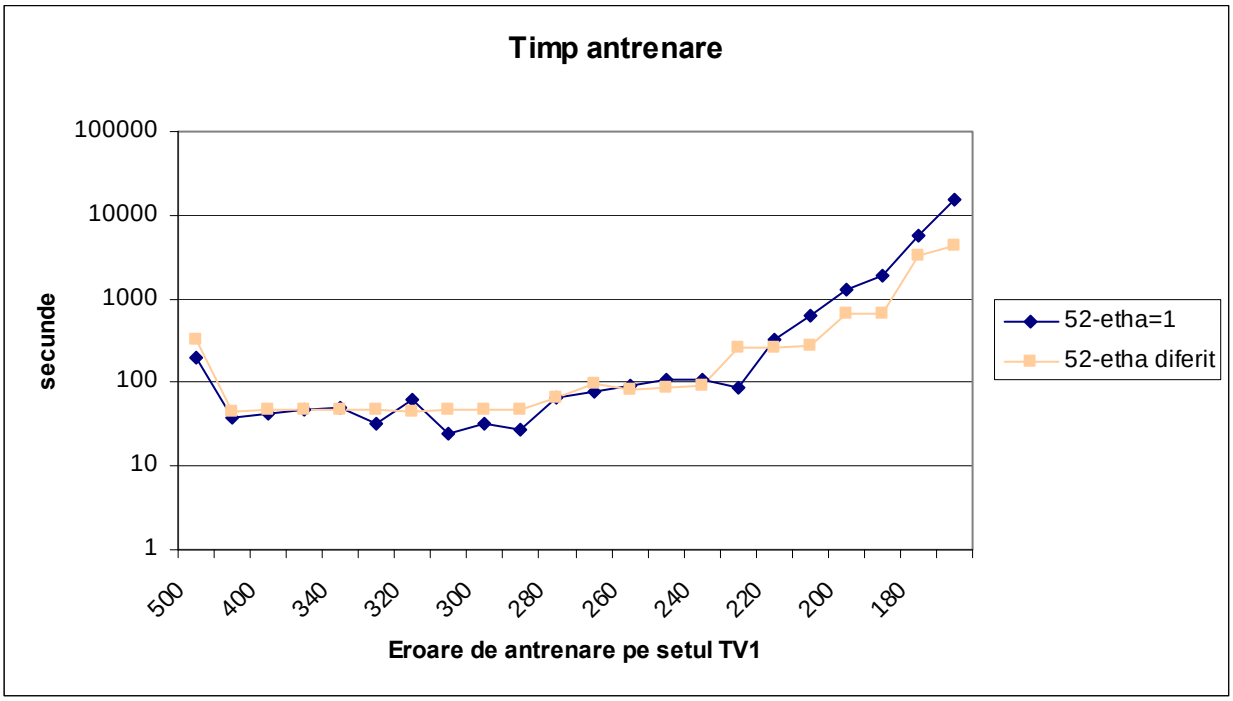


Fig. 3.10 Timpul de antrenare - comparație între două arhitecturi cu 52 neuroni pe stratul ascuns

Rezultatele prezentate în această secțiune au fost obținute antrenând şi testând rețeaua Backpropagation pe setul TV1 care conține 2351 vectori. În secțiunea următoare prezentăm rezultatele obținute în cazul antrenării pe setul AV1 (4702 vectori) şi ale testării pe setul TV1.

3.3.3 Rezultate obținute în cazul antrenării pe setul AV1 şi ale testării pe TV1

Prezint rezultate doar pentru arhitecturi ale rețelei cu un număr de neuroni mai mare de 96 pe stratul ascuns şi un coeficient de învățare descrescător în timp. Şi în acest caz, pentru testare, oprim rețeaua în momentul în care atinge un anumit prag al erorii de antrenare, o testăm pentru a obține numărul de documente incorect clasificate, după care continuăm cu antrenarea. În acest caz eroarea totală de antrenare este obținută ca o sumă a tuturor celor 4702 erori, ceea ce reprezintă o medie a erorii per exemplu de 0,11 în cazul erorii totale egale cu 500. În acest experiment am ajuns la o eroare totală egală cu 80, ceea ce înseamnă o eroare medie de 0,017 per exemplu.

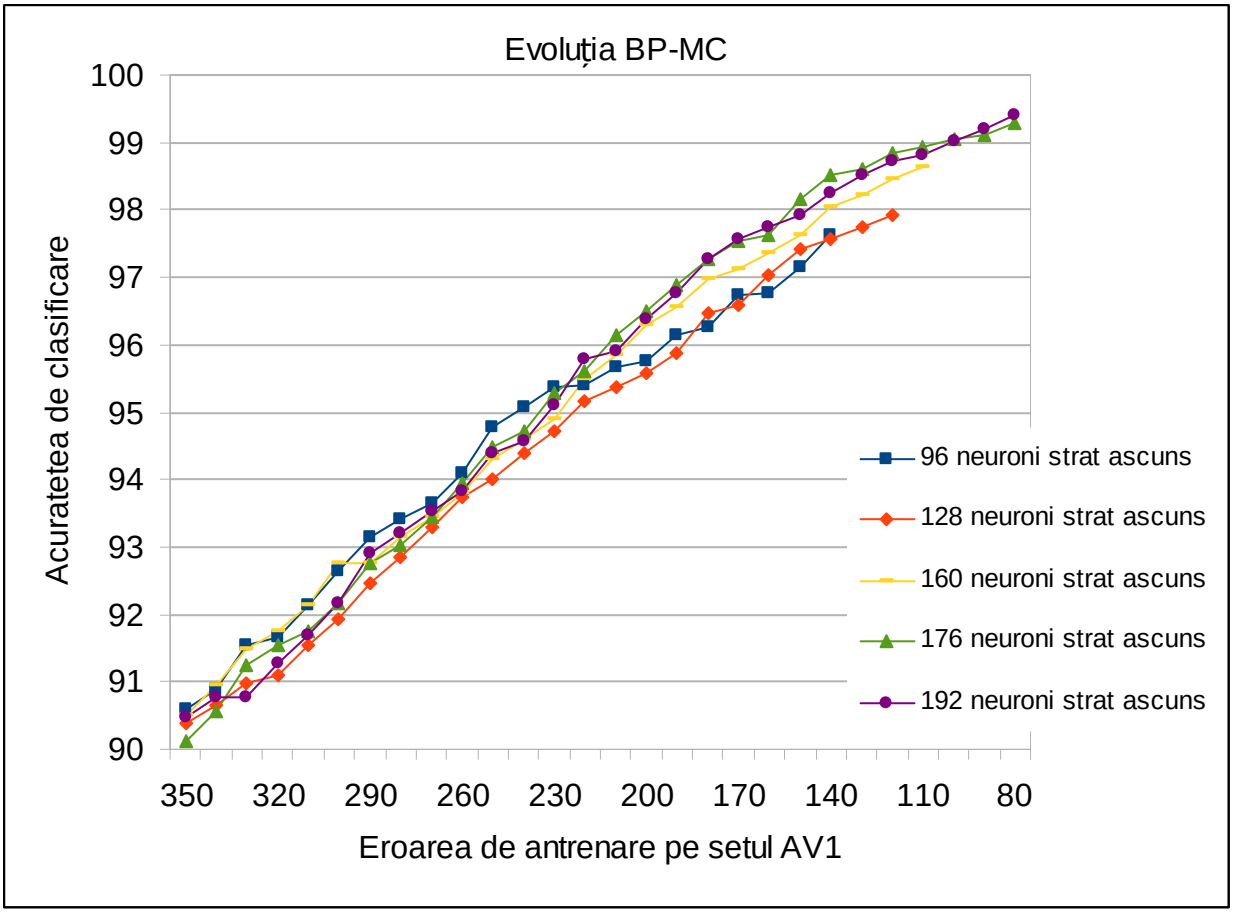


Fig. 3.11 Acuratețea de clasificare în cazul antrenării pe setul AV1 şi a testării pe setul TV1

În acest caz, arhitectura cu 176 de neuroni pe stratul ascuns a obținut cele mai multe valori minime pentru numărul de documente incorect clasificate, dar, în momentul în care eroarea totală de antrenare a scăzut sub valoarea 100, rezultatele cele mai bune au fost obținute de arhitectura cu 192 de neuroni pe stratul ascuns. În acest caz am obținut un număr de 14 documente incorect clasificate, ceea ce reprezintă o acuratețe de clasificare a metaclasificatorului

de 99,40%. Diferența față de cea mai bună valoare față de cea cu 176 de neuroni pe stratul ascuns este de doar 3 documente incorect clasificate.

Coeficientul de învățare	Eroarea totală de antrenare	Număr de neuroni pe stratul ascuns				
		96	128	160	176	192
1	500	381	369	**368**	376	374
1	450	345	325	334	324	**321**
1	400	278	282	**268**	282	288
1	350	**221**	226	223	232	224
0,9	340	214	220	**213**	222	217
0,9	330	**199**	212	200	206	217
0,8	320	196	209	**194**	199	205
0,8	310	186	199	**185**	194	195
0,7	300	173	190	**170**	184	184
0,7	290	**161**	177	170	170	167
0,6	280	**155**	168	161	164	160
0,6	270	**149**	158	154	154	152
0,6	260	**139**	147	146	142	145
0,6	250	**123**	141	134	130	132
0,5	240	**116**	132	127	124	128
0,5	230	112	124	120	**111**	115
0,5	220	108	114	106	103	**99**
0,4	210	102	109	98	**91**	96
0,4	200	100	104	87	**82**	85
0,3	190	91	97	81	**73**	76
0,3	180	88	83	71	**64**	**64**
0,2	170	77	80	68	58	**57**
0,2	160	76	70	62	56	**53**
0,1	150	67	61	56	**43**	49
0,1	140	56	57	46	**35**	41
0,1	130		53	42	**33**	35
0,1	120		49	36	**27**	30
0,1	110			32	**25**	28
0,1	100				**22**	23
0,01	90				21	**19**
0,01	80				17	**14**

Tabel 3.1 Număr de documente incorect clasificate

În tabelul de mai sus am prezentat numărul de documente incorect clasificate obținut de arhitecturile testate. Pentru fiecare arhitectură am prezentat valoarea obținută pentru toate testele efectuate în timpul antrenării rețelei. Astfel, în coloana a doua se află valorile erorii totale de antrenament la care rețeaua a fost oprită şi testată. În prima coloană sunt date valorile coeficientului de învățare care a fost folosit pentru rețea, astfel încât eroarea de antrenare a rețelei să coboare la valoarea precizată.

Acest nou metaclasificator cu o rețea neuronală cu număr suficient de mare de neuroni pe stratul ascuns a reuşit să depăşească şi limita maximă de 98,63% la care ar fi putut ajunge „teoretic" clasificatorii incluşi în cadrul metaclasificatorului.

Foarte interesant, acest metacalsificator neuronal cu învățare supervizată a demonstrat faptul că acurtețea de 98,63% nu este de fapt limita maximă a metaclasificării aşa cum eu considerasem. Datorită procesului de învățare supervizată această limită poate fi depăşită. Spre exemplu în cazul unui vectoor de intrare în rețea al cărui element maxim nu se află situat pe poziția clasei corecte acesta poate activa la ieşire celula corectă tocmai datorită unui proces de învățare adecvat (în care rețelei i s-au mai livrat exemple asemănătoare)

4 Concluzii

În acest referat de doctorat prezint contribuțiile mele în domeniul clasificării de documente text. Din tot fluxul de etape necesare în procesul de regăsire al informațiilor, m-am axat în acest referat pe etapa de metaclasificare. În această etapă combin eficiența mai multor clasificatori individuali diferiți în scopul obținerii unor rezultate superioare de clasificare a documentelor. Am gândit acest metaclasificator ca fiind format din două componente. O componentă, considerată ca fiind etapa de preclasificare, realizată dintr-un metaclasificator (selector) neadaptiv şi o altă componentă, considerată ca fiind etapa de postclasificare, realizată dintr-o rețea neuronală de tip backpropagation

În capitolul 1 am prezentat o vedere de ansamblu asupra procesului de regăsire al informațiilor detaliind etapa de metaclasificare.

În capitolul următor am prezentat o serie de metaclasificatori neadaptivi care folosesc diferite procedee pentru ponderarea valorilor generate de către fiecare clasificator în parte cu scopul de a îmbunătăți acuratețea finală a clasificării. În prima secțiune am prezentat un metaclasificator care însumează simplu toate valorile generate de către clasificatoare. Rezultatele obținute de acest metaclasificator sunt mai bune decât votul majoritar, dar nu semnificativ. În următoarele secțiuni am prezentat o serie de experimente care încearcă diferite valori pentru a pondera vectorii generați de către clasificatori. Aceste valori ponderează vectorii, în funcție de ordinea obținută de fiecare clasă în cadrul vectorului. Cele mai bune rezultate obținute au fost de 301 documente incorect clasificate, ceea ce reprezintă o acuratețe a clasificării de 87,20%. Aceste rezultate s-au obținut când am utilizat ponderarea liniară cu pasul de „0,5”. Vectorii obținuți în urma acestei etape vor fi utilizați şi în următoarea etapă din metaclasificator, cea de postclasificare. Chiar dacă cele mai bune rezultate au fost obținute cu ponderarea prezentată mai sus, în următoarea etapă din metaclasificator am folosit rezultatele obținute utilizând ponderarea de tip „Eurovision” care a obținut un scor 87,03%.

În capitolul 3 am prezentat elementele necesare pentru dezvoltarea unei rețele neuronale de tip backpropagation, adaptată pentru funcționarea în acest context. Parametrii rețelei care au fost experimentați în această lucrare sunt numărul de neuroni de pe stratul ascuns şi coeficientul de învățare al rețelei. Algoritmul prezentat se aplică rețelelor feed-forward care conțin 2 nivele de unități cu funcția de activare sigmoidă, fiecare unitate de pe un nivel fiind conectată la toate

unitățile de pe nivelul anterior. Deoarece rețeaua neuronală prezentată este o rețea cu învățare supervizată, a avut nevoie de o etapă de antrenare. Pentru antrenare şi ulterior testare am folosit inițial acelaşi set de vectori numit TV1. Folosind acest set am testat influența numărului de neuroni de pe stratul ascuns şi a coeficientului de învățare asupra acurateței de clasificare. Astfel am variat numărul de neuroni de pe stratul ascuns între valoarea 17 şi valoarea 52 cu un coeficient de învățare constant (η=1). Cele mai bune rezultate au fost obținute de arhitectura cu 52 de neuroni pe stratul ascuns, ajungând la o acuratețe a clasificării de 94,26%. Totuşi odată cu creşterea numărului de neuroni de pe stratul ascuns am observat că timpul de antrenare pentru rețea nu creşte, chiar dacă numărul de calcule care trebuie efectuate cresc. De aceea am încercat şi utilizarea unui număr mai mare de neuroni pe stratul ascuns.

Tot în acest capitol am prezentat experimente realizate utilizând seturi diferite pentru antrenare şi testare. În acest caz am folosit şi valori descrescătoare ale coeficientului de învățare. În momentul în care am redus şi coeficientul de învățare am reuşit să antrenăm rețeaua până la o valoare mică a erorii de antrenare (medie 0,017 per exemplu de antrenament). Cele mai bune rezultate (99,40% acuratețe de clasificare!) le-am obținut folosind o rețea neuronală cu 192 de neuroni pe stratul ascuns. Totuşi, comparativ ca număr de rezultate bune pe parcursul antrenării chiar înainte de a atinge eroarea de antrenare minimă, le-am obținut utilizând o rețea cu 176 de neuroni pe stratul ascuns.

În urma experimentelor efectuate am observat că introducerea unei rețele neuronale în cadrul metaclasificatorului face ca acesta să se adapteze mult mai bine la documentele care trebuie clasificate, reuşind astfel să clasifice şi documentele cu problemă pe care metaclasificatorii prezentați anterior nu au reuşit să le „învețe”. Acest nou metaclasificator a reuşit să depăşească şi limita maximă de 98,63% la care ar fi putut ajunge „teoretic” clasificatorii incluşi în cadrul metaclasificatorului.

Ca şi dezvoltări ulterioare se încearcă îmbunătățirea rețelei neuronale, astfel încât aceasta să conveargă mult mai rapid. De asemenea s-ar putea testa rețeaua înlocuind funcția de activare sigmoidă cu alte tipuri de funcții de activare.

5 Bibliografie

[Brea06] Breazu, M., *Tehnici fractale şi neuronale în compresia de imagini*, Editura universităţii „Lucian Blaga" din Sibiu, ISBN 978-973-739-251-0, 2006

[Cret08] Cretulescu R., *Support Vector Machine versus Bayes Naïve*, 2nd PhD report, "Lucian Blaga" University of Sibiu, 2008, http://webspace.ulbsibiu.ro/radu.kretzulescu/html/phdreport2.pdf

[Hayk94] Haykin, S., Neural Networks: A comprehensive Foundation, MacMillan College, New York, 1994

[Hebb49] Hebb, D.O., The Organization of Behavior, John Wiley & Sons, New York, 1949

[Jaeg08] Jaeger, S., Huanfeng, M., Dörmann, D., Combinig Calssifiers with Informational Confidence, Studies in Computational Intelligence (SCI) 90, pag. 163-191, 2008

[Jain96] Jain, A., Mao, J., Mohiuddin, K.M., Artificial Neural Networks: A Tutorial, Journal of IEEE Computational Science and Engineering, pp. 31-44, 1996

[Kung93] Kung S.Y., Digital Neural Networks, Prentice Hall, New Jersey, 1993

[Mora06] Morariu, D., Vintan, L., Tresp, V., Meta-classification using SVM classifier for Text Document, Proceedings of the 3rd International Conference on Machine Learning and Pattern Recognition (MLPR'06), ISSN 1503-5313, vol. 15, pp. 222-227, Barcelona, Spain, October, 2006.

[Mora08] Morariu, D., Text Mining Methods based on Support Vector Machine, Ed. MatrixRom, Bucureşti, 2008.

[Reut00] Misha Wolf and Charles Wicksteed - Reuters Corpus: http://www.reuters.com/researchandstandards/corpus/ lansat în noiembrie 2000, accesat în septembrie 2009

[Vintan07] Vinţan N. L., – Prediction Techniques in Advanced Computing Architectures (in limba engleza), Editura Matrix Rom, Bucureşti, ISBN 978-973-755-137-5, 2007

[Wass89] Wassermann, P.D., Neural Computing. Theory and Practice, Van Nostrand Reinhold, 1989