Despre autori

Începând cu anul universitar 1992/1993, Răzvan Andonie a predat un curs de Reţele neurale la Universitatea "Transilvania", specializările Electronică şi calculatoare, Electrotehnică, Informatică. Acelaşi curs l-a predat la Universitatea Texas din San Antonio, SUA, în anul universitar 1999/2000.

Angel Caţaron a făcut parte din prima generaţie de absolvenţi care au frecventat acest curs. În perioada 1997-1999, el a predat cursul de Reţele neurale la Universitatea "Transilvania", specializarea Electronică şi calculatoare. Pe lânga aceasta, el este coordonatorul orelor de laborator pentru acest curs şi este doctorand în domeniul reţelelor neurale.

Prefaţă

Chiar dacă acest material este suportul cursului de Reţele neurale, de-a lungul timpului acest curs a început să conţină capitole noi, la interferenţa reţelelor neurale cu sistemele fuzzy şi algoritmii genetici. Aceasta este în concordanţă cu tendinţele actuale în lume.

De aceea, am preferat denumirea de *inteligenţă computaţionala*. Inteligenţa computaţională, aşa cum a fost definită de Bezdek[1], are ca obiectiv modelarea inteligenţei biologice. Din acest punct de vedere, ea este similară domeniului numit *inteligenţă artificială*. Spre deosebire de inteligenţa artificială, care este bazată pe noţiunea de cunoştinţă, inteligenţă computaţională este o modelare numerică a inteligenţei biologice. Putem vorbi deci de trei tipuri de "inteligenţă": biologică, artificială şi computaţională.

Inteligenţa computaţională este formată din următoarele subdomenii: reţele neurale, algoritmi genetici, programare evoluţionară, sisteme fuzzy, viaţă artificială. În contextul inteligenţei computaţionale, toate aceste subdomenii sunt legate de modelarea numerică a inteligenţei biologice.

Acest curs acoperă majoritatea subdomeniilor inteligenţei computaţionale. Fiecare capitol oferă, pe lângă informaţii teoretice, câte o secţiune de aplicaţii, urmată de un set de exerciţii, cele marcate prin (C) fiind destinate implementării pe calculator.

[1]Bezdek, J. "On the Relationship Between Neural Networks", Pattern Recognition and Intelligence, Int. J. Approximate Reasoning, 6, 85-107, 1992.

Cuprins

Capitolul 1

Preliminarii în calculul neural

Calculul neural se efectuează pe o reţea densă de noduri şi conexiuni. Aceste noduri lucrează în mod colectiv şi simultan şi se numesc *neuroni artificiali*, sau *neuroni*. Neuronii pot opera, de exemplu, ca sumatoare sau comparatoare. De obicei, neuronii lucrează în paralel şi sunt configuraţi în arhitecturi regulate. Astfel, ei pot fi organizaţi pe nivele ierarhice şi se permit conexiuni feedback în cadrul unui nivel sau conexiuni feedback către nivelele adiacente. Puterea fiecărei conexiuni este exprimată printr-o valoare numerică numită *pondere*, care poate fi modificată.

Domeniul reţelelor neurale este cunoscut şi sub denumiri similare: neurocalcul, conexionism, procesare paralelă distribuită, sisteme adaptive, reţele cu autoorganizare etc. Această varietate indică de fapt tot atâtea perspective din care se studiază reţelele neurale.

Reţelele neurale funcţionează ca reţele paralele distribuite. Caracteristica lor de bază este arhitectura. Unele reţele sunt caracterizate de comportarea lor în timp, de *dinamica* lor. Reţelele neurale diferă între ele prin modul de învăţare: există o varietate de reguli de învăţare care stabilesc când şi cum se modifică ponderile conexiunilor. În fine, reţelele diferă prin viteza şi eficienţa de învăţare.

Spre deosebire de calculatoarele convenţionale, care sunt programate să efectueze anumite lucrări, majoritatea reţelelor neurale trebuie să fie învăţate (sau instruite). Ele învaţă noi asocieri, noi pattern-uri, noi dependenţe funcţionale. După cum vom vedea mai târziu, faptul că reţelele învaţă reguli şi algoritmi înlocuieşte programarea necesară în calculul convenţional. Utilizatorii reţelelor neurale nu specifică un algoritm care să fie executat de către un anumit neuron, cum s-ar întâmpla pe o maşină tradiţională. În loc de aceasta, ei aleg o configuraţie care li se pare cea mai bună arhitectură, specifică toate caracteristicile neuronilor şi ponderile iniţiale, apoi aleg modul de instruire pentru reţea. În următoarea fază, sunt aplicate diferite date de intrare din care reţeaua îşi extrage cunoştinţe, adică învaţă. Ca rezultat, reţeaua acumulează informaţie care poate fi apoi utilizată.

Calculul cu reţele neurale se situează între inginerie şi inteligenţa artificială. Se folosesc tehnicile matematice inginereşti clasice, dar şi metode euristice specifice inteligenţei artificiale.

În acest sens vom răspunde la următoarele întrebări:

- Cum poate fi instruită eficient o reţea şi cum se învaţă ea?
- Ce modele de neuroni trebuie folosite?
- Care sunt cele mai adecvate arhitecturi pentru anumite clase de probleme?
- Care sunt cele mai bune metode pentru a extrage cunoştinţele acumulate într-o reţea?
- Care sunt aplicaţiile tipice pentru calculul neural şi cât de eficiente sunt aceste aplicaţii?

Reţelele neurale au atras atenţia specialiştilor din numeroase discipline. Neurobiologii sunt interesaţi în modelarea reţelelor neurale biologice. Fizicienii sunt atraşi de analogiile dintre reţelele neurale şi sistemele dinamice neliniare pe care le studiază. Matematicienii sunt fascinaţi de potenţialul modelării matematice aplicat în sistemele foarte mari şi complexe. Inginerii în electronică şi calculatoare aplică reţelele neurale în procesarea semnalelor şi construiesc pe baza reţelelor neurale circuite integrate inteligente. Psihologii caută în reţelele neurale structurile prototip care modelează procesarea informaţiei de către om. În fine, informaticienii sunt interesaţi în posibilităţile de calcul ale reţelelor masiv paralele în domeniile inteligenţei artificiale, teoriei calculabilităţii, modelării şi simulării etc.

1.1 Calculul neural: exemple

Ne propunem să dăm câteva exemple de utilizare a reţelelor neurale în rezolvarea unor probleme reale.

Clasificatori şi sisteme automate de orientare în spaţiu

Vom defini reţelele neurale care răspund instantaneu datelor de intrare. Pentru început, vom analiza performanţele unui simplu clasificator, apoi vom extinde această problemă.

Fie $P_0, P_1, \ldots, P_7$ opt puncte în spaţiul tridimensional. Mulţimea constă din toate vârfurile unui cub tridimensional:

$$\{P_0(-1,-1,-1), P_1(-1,-1,1), P_2(-1,1,-1), P_3(-1,1,1),$$
$$P_4(1,-1,1), P_5(1,-1,1), P_6(1,1,-1), P_7(1,1,1)\}$$

Considerăm două clase de puncte:

1. puncte cu două sau mai multe coordonate pozitive: P_3, P_5, P_6, P_7;
2. restul de puncte.

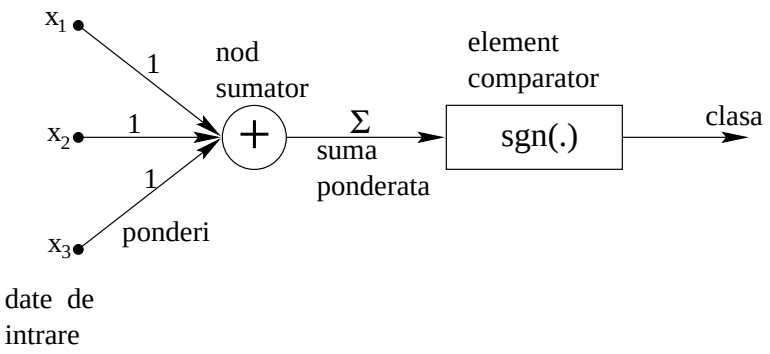


Figura 1.1: Clasificator realizat cu o singură unitate.

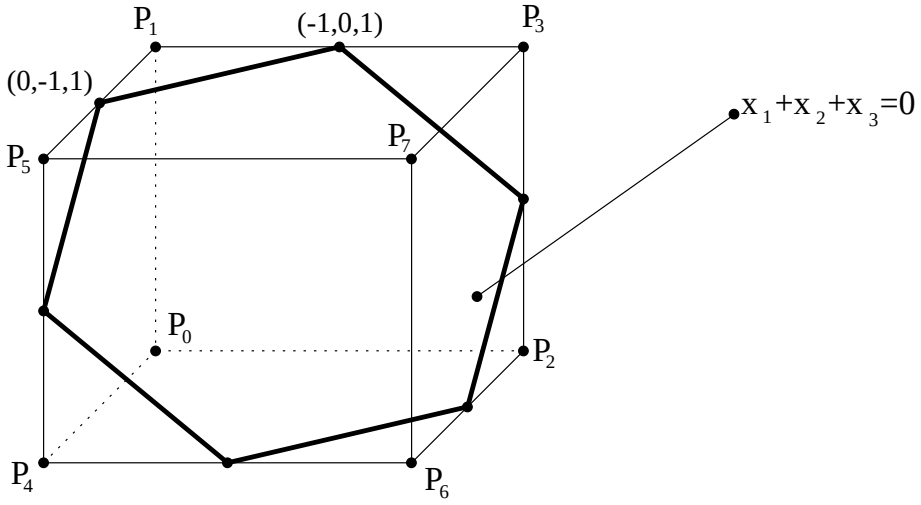


Figura 1.2: Partiţionarea spaţiului cartezian.

Pentru orice punct $P_i(x_1, x_2, x_3)$, $i = 1, \ldots, 7$, apartenenţa la una dintre clasele de mai sus poate fi stabilită prin următorul calcul:

$$\mathrm{sgn}(x_1, x_2, x_3) = \begin{cases} 1 & \text{pentru clasa } 1 \\ -1 & \text{pentru clasa } 2 \end{cases}$$

Această expresie descrie funcţia de decizie a unui clasificator. Nu este necesară o instruire a acestui clasificator. Reţeaua neurală rezultată este extrem de simplă (fig. 1.1).

Am realizat clasificarea printr-o singură *unitate*, sau *nod de calcul*. Aceasta implementează însumarea cu ponderile respective (1 în acest caz) şi este urmată de o comparare cu prag.

De fapt, se realizează o partiţionare a spaţiului cartezian tridimensional prin planul $x_1 + x_2 + x_3 = 0$ (fig. 1.2).

Punctele de deasupra planului fac parte din clasa 1, iar punctele de dedesupt din clasa 2.

Problema pe care ne-o punem este dacă o funcţie continuă nu poate fi mai avantajoasă (fig. 1.3).

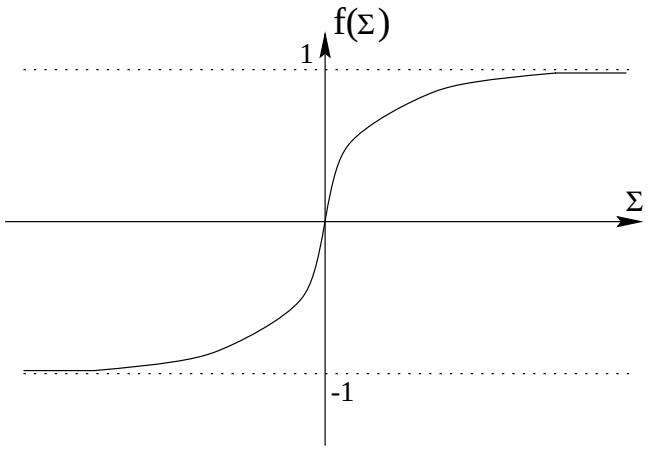


Figura 1.3: Funcţie continuă de decizie.

În acest caz, datele de ieşire pot fi în intervalul dintre -1 şi 1. Folosirea neuronilor cu caracteristici continue oferă posibilităţi mult mai mari. Se obţine o granularitate (o rafinare) mai mare a datelor de ieşire.

Neurologii folosesc electro-encefalograma (EEG) care preia pe mai multe canale impulsurile electrice ale creierului. Evaluarea EEG este dificilă şi de aceea se face de obicei de către neurologi calificaţi. Pentru o monitorizare pe calculator, să ne concentrăm pe un caz concret: detectarea unei iminente crize de epilepsie. Este nevoie de o prelucrare on-line a semnalelor EEG.

În 1990, Eberhart şi Dobbins au realizat detectarea semnalelor EEG pentru crizele de epilepsie folosind un clasificator neural. Datele sunt monitorizate prin patru canale de interes. Semnalele EEG sunt eşantionate de 200 sau 250 ori pe secundă într-o fereastră de 240 ms. Aceasta are ca rezultat obţinerea a 48 sau 60 de eşantioane de date pentru fiecare canal. Aceste eşantioane trebuie evaluate şi sunt introduse într-o reţea neurală de 40 de unităţi interconectate, cu caracteristici continue. Un total de 41 de unităţi aranjate pe trei nivele ierarhice procesează datele. Două unităţi de ieşire sunt utilizate pentru identificarea vârfurilor de semnal.

Reţeaua a fost elaborată de o echipă de ingineri şi neurologi. După ce a fost instruită, reţeaua a dat rezultate excelente dovedindu-şi utilitatea în spitale.

Să considerăm un alt exemplu: proiectul ALVINN (Autonomous Land Vehicle In a Neural Network), raportat de Pomerleau (1989). Reţeaua ALVINN preia imagini ale drumului printr-o cameră şi printr-un laser care detectează profunzimea obiectelor. La ieşire este generată direcţia pe care trebuie să circule autovehiculul pentru a urma drumul. Arhitectura reţelei este cea din figura 1.4.

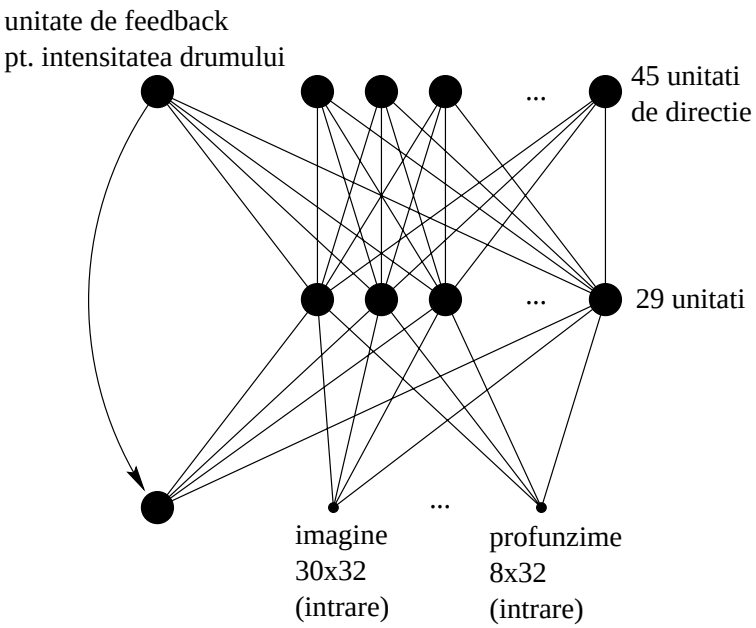


Figura 1.4: Arhitectura reţelei ALVINN.

Informaţia video este o retină de 30x32 care sesizează în albastru (oferă cel mai bun contrast). Unitatea de feedback reglează contrastul. Cele 1217 intrări conduc spre 29 unităţi cu funcţie continuă. Unitatea din mijlocul celor 45 de unităţi de ieşire arată cât de puternică este tendinţa de a merge înainte. Unităţile din stânga-dreapta reprezintă tendinţele de a o lua la stânga-dreapta. Unităţile din extremităţile stângă şi dreaptă corespund virajelor stânga-dreapta cât mai accentuate.

ALVINN a fost instruit prin imagini sintetizate pe calculator. Performanţele obţinute au fost comparabile cu caracteristicile celor mai bune sisteme tradiţionale de orientare prin vedere artificială.

Capacitatea sistemului ALVINN de a păstra direcţia drumului nu a fost implementată prin programare (algoritm), ci se bazează pe cunoştinţele asimilate prin instruire. După 1/2 oră de instruire, sistemul era capabil să se orienteze singur pe drum. Altfel ar fi fost necesare luni de zile pentru dezvoltarea algoritmilor care să recunoască pattern-urile din imaginile preluate.

S-au făcut observaţii interesante. Astfel, reţeaua s-a comportat mai bine după ce a fost instruită să refacă erori de conducere. Prin aceasta, reţeaua şi-a dezvoltat măsurile de corectare a conducerii.

Memorie simplă şi restaurarea pattern-urilor

Vom discuta acum despre reţele neurale care răspund în timp datelor de intrare (un pattern în acest caz). Deoarece ele fac acest lucru într-un mod foarte caracteristic, prin reconstrucţia treptată a unui pattern memorat, vom numi aceste

reţele *memorii*.

Fie reţeaua simplă din figura 1.5.

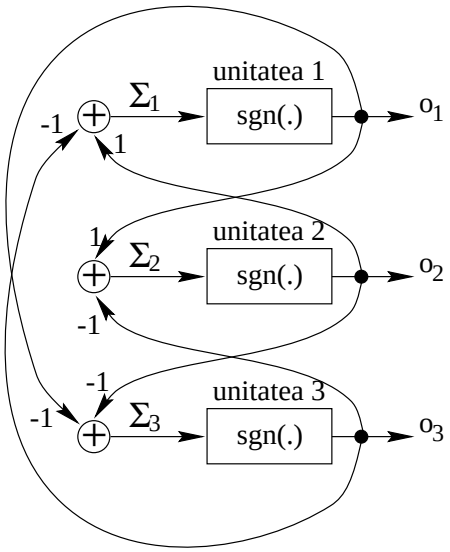


Figura 1.5: Reţea memorie.

Reţeaua constă din trei unităţi care calculează valorile funcţiei signum, trei noduri de însumare şi şase ponderi care pot fi ± 1. Semnalele care trec prin ponderi sunt înmulţite cu valoarea ponderilor. Presupunem că reţeaua este iniţializată la ieşire cu $o_1 = o_2 = o_3 = 1$ (cazul 1, tab. 1.1). Când se permite reţelei să calculeze, intrările la unităţile 1 şi 2 sunt 0, în timp ce la unitatea 3 este -2. Ca rezultat, o_1 şi o_2 nu se modifică, deoarece sgn(0) nu este definit, în timp ce o_3 devine -1. Urmează cazul 2, care este o ieşire finală deoarece nu se mai poate modifica. Cazurile 3 şi 4 sunt alte exemple de tranziţii posibile. Observăm că şi cazurile 3 şi 4 duc apoi la cazul 2, operând câte o singură modificare.

Iată cum se reprezintă geometric aceste actualizări efectuate de reţeaua de tip memorie descrisă. Se vede că $P_6(1, 1, -1)$ este ieşirea stabilă a memoriei. Când o singură componentă a vectorului de iniţializare binar (o_1, o_2, o_3) diferă de P_6, reţeaua corectează această componentă. Ieşirea este apoi menţinută constantă. Dacă P_6 reprezintă descrierea corectă a unui pattern, iar P_4, P_7, P_2 variante distorsionate ale lui P_6, memoria are capacitatea de a restaura variantele distorsionate asimilându-le cu P_6 (fig. 1.6).

Acest concept de memorie poate fi extins cu uşurinţă la lumea reală a aplicaţiilor.

Probleme de optimizare

Reţelele neurale pot fi aplicate cu succes pentru rezolvarea unor probleme de optimizare.

Tabelul 1.1: Exemple de tranziţii ale reţelei tip memorie. X înseamnă sgn(0), iar ieşirile încadrate sunt cele care se modifică.

Cazul	Nr. unităţii	Ieşirea actuală	Σ	sgn(Σ)	Ieşirea următoare
	1	1	0	X	1
1	2	1	0	X	1
	3	1	-2	-1	**-1**
	1	1	2	1	1
2	2	1	2	1	-1
	3	-1	-2	-1	-1
	1	-1	2	1	**1**
3	2	1	0	X	1
	3	-1	0	X	-1
	1	1	0	X	1
4	2	-1	2	1	**1**
	3	-1	0	X	-1

Să presupunem că valoarea analogică x, $0 \leq x \leq 3,5$ trebuie digitizată într-un număr binar $v_1 v_0$, $(v_1, v_0 \in \{0, 1\})$, astfel incât

$$x \approx 2v_1 + v_0.$$

Există, evident, patru posibilităţi: 00, 01, 10, 11. O reţea similară cu memoria din exemplul precedent poate rezolva această conversie (fig. 1.7).

Reţeaua constă din două unităţi cu caracteristici continue, fiecare cu răspuns în intervalul dintre 0 şi 1. La cele două unităţi, reţelei i se mai adaugă un număr de elemente de interconectare pe care nu le specificăm.

Conversia corespunde minimizării erorii de conversie A/D, unde eroarea este $(x - 2v_1 - v_0)^2$, în prezenţa restricţiei $v_0, v_1 \in \{0, 1\}$. Această problemă de minimizare este rezolvabilă de către o anumită clasă de reţele neurale. Caracteristic acestei clase este că se minimizează aşa numita *funcţie de energie* pe parcursul calculului.

Proprietatea de minimizare a energiei este de importanţă foarte mare şi se formulează astfel: reţeaua îşi caută singură energia ei minimă şi se stabilizează acolo. O serie de probleme de optimizare se transpun direct în minimizarea funcţiei de energie a unei reţele neurale. Minimizarea energiei de către reţea poate fi considerată analogă minimizării erorii de conversie A/D.

Clasa de reţele neurale exemplificată prin conversia A/D este utilizabilă şi pentru probleme de optimizare combinatorică în care complexitatea este exponenţială sau, mai rău, în ordinul lui $n!$. În aceste probleme, este important să se reducă ordinul timpului de căutare. Reţelele neurale oferă şi în acest sens o alternativă.

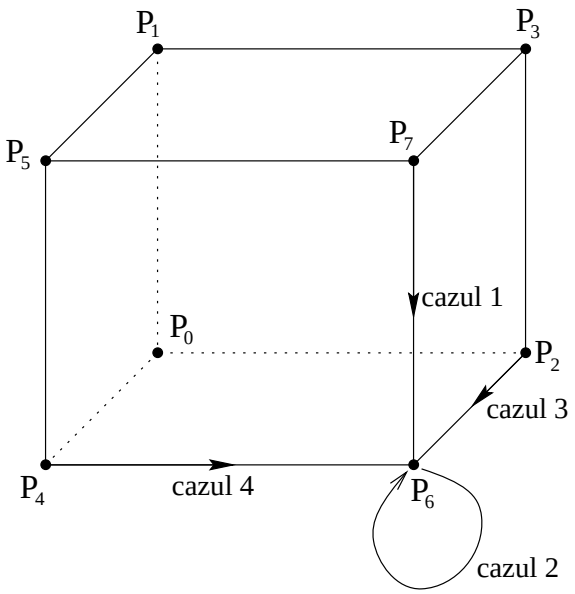


Figura 1.6: Actualizările efectuate de reţeaua de tip memorie.

Detectarea grupărilor şi a trăsăturilor

O clasă importantă de reţele neurale pot fi utilizate pentru detectarea grupărilor de date. Aceste reţele sunt calate pe anumite aspecte de similaritate în datele evaluate. De exemplu, se poate să fim interesaţi în a grupa anumite rezultate de măsurare în scopul eliminării erorilor sistematice care pot apărea în timpul măsurării. Deoarece zgomotul este aleator, el nu formează grupări, ci doar perturbă formarea grupărilor reale de date.

Detectarea grupărilor şi trăsăturilor prezintă importante proprietăţi de autoorganizare care sunt legate de inteligenţa artificială şi teoria informaţiei. Reţelele din această clasă au în general arhitecturi simple, dar subtilităţile apar în timpul procesului de autoorganizare.

Detectarea trăsăturilor se raporteaza la reducerea dimensionalităţii datelor. De exemplu, semnalul vorbirii constă din 15 canale de frecvenţă audio. Fonemele sunt deci descrise într-un spaţiu 15-dimensional. Problema este că nu putem reprezenta astfel fonemele, deoarece capacitatea noastră de vizualizare se reduce la trei dimensiuni. Utilizând o reţea neurală, este posibil să reprezentăm spectrul 15-dimensional al vorbirii într-un tablou în plan. Secvenţa fonemelor unui cuvânt formează o traiectorie specifică în plan. Aceste *hărţi fonotopice* pot fi foarte utile în construirea maşinilor de scris fonetice, în învăţarea vorbirii şi pentru terapie.

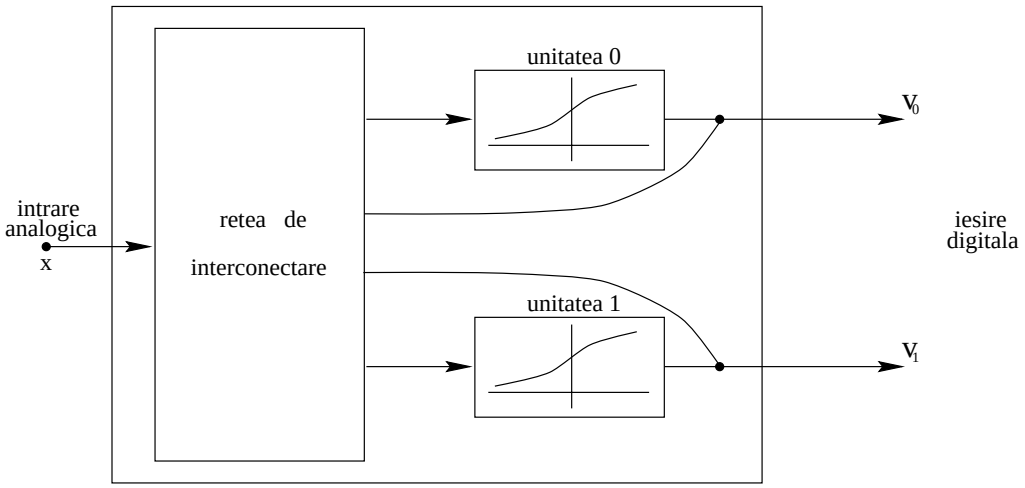


Figura 1.7: Diagramă bloc a unui convertor A/D pe doi biţi.

1.2 Istoricul dezvoltării reţelelor neurale

McCulloch şi Pitts (1943) au propus primul model pentru neuron. Acest model include toate elementele pentru a efectua operaţii logice, dar la acel nivel tehnologic era imposibil de implementat.

Donald Hebb (1949) a propus un model de învăţare pentru a actualiza conexiunile neuronului, cunoscut acum ca *regula hebbiană de învăţare*. El a formulat ideea că informaţia poate fi memorată în conexiuni.

Primele neurocalculatoare au fost construite în anii '50 (Minsky, 1954). Ele îşi adaptau automat conexiunile. În 1958, Frank Rosenblatt a inventat un element neural numit *perceptron*. Era conceput ca o maşină instruibilă capabilă să înveţe să clasifice anumite pattern-uri prin modificarea conexiunilor la elementele comparatoare.

La începutul anilor '60, a fost propus ADALINE (ADAptive LINEar combiner), un dispozitiv bazat pe regula de învăţare Windrow-Hoff (Bernard Windrow, Marcian Hoff). Regula minimiza eroarea pătratică însumată pe parcursul instruirii. Aplicaţiile erau de recunoaşterea formelor, control adaptiv şi previziunea vremii.

În ciuda entuziasmului anilor '60, maşinile existente nu permiteau abordarea unor probleme complexe. Pe de altă parte, nici schemele de învăţare nu erau suficient de dezvoltate. S-a intrat astfel într-o perioadă de stagnare a cărei cauze erau de fapt cunoscute.

Episodul final al acestei ere a fost lucrarea lui Minsky şi Papert[1], care a demonstrat limitele reţelelor bazate pe perceptroni. În acest timp, majoritatea cercetătorilor se îndreptau spre alte domenii. Domeniul reţelelor neurale (care făcea la acea vreme parte din cibernetică) părea închis. În schimb se dezvolta

[1]Minsky, M., S. Papert "Perceptrons". Cambridge, MA, MIT Press, 1969.

promiţător domeniul inteligenţei artificiale, preluând şi sarcini pe care reţelele neurale nu puteau să le rezolve la acel stadiu.

În perioada 1965-1984, câţiva cercetători au reuşit să dezvolte totuşi cercetările. Kunihiko Fukushima a definit o clasă de reţele neurale numite *neocognitroni* (1980). Neocognitronul modelează recunoaşterea vizuală a formelor prin emularea imaginilor de pe retină şi procesarea lor folosind neuroni ierarhizaţi pe două nivele.

Cercetările în domeniul memoriilor asociative au evoluat în Finlanda (Teuvo Kohonen) şi SUA (James Anderson). Stephen Grossberg şi Gail Carpenter au introdus câteva arhitecturi de reţele neurale şi au dezvoltat teoria reţelelor adaptive prin rezonanţă, ART.

Era renaşterii a început odată cu introducerea arhitecturii recurente pentru memoriile asociative (John Hopfield, 1982). O altă revitalizare a domeniului provine din lucrările lui James McClelland şi David Rumelhart (1986).

Începând cu anii 1986-1987, s-au iniţiat multe programe de cercetare şi interesul a devenit extrem de mare. Au apărut aplicaţii complexe. Au fost fabricate chip-uri VLSI de reţele neurale. Cu toate că domeniul calculului neural are o istorie interesantă, el este încă la începutul dezvoltării sale.

1.3 Viitorul

Datorită denumirii, domeniul reţelelor neurale este supus unei supraestimări populiste. Este tentant pentru om să-şi imagineze o maşină care să fie asemeni lui. Terminologia antropomorfă trebuie privită cu multă reţinere.

Putem fi aproape siguri că reţelele neurale nu vor înlocui calculatoarele clasice. Aceasta, deoarece calculatoarele clasice sunt foarte ieftine şi eficiente pentru efectuarea calculelor numerice (procesări de text, CAD, procesări de date).

Sunt însă domenii întregi în care reţelele neurale devin mai avantajoase. Cele mai interesante aplicaţii sunt cele care presupun inferenţa de tip uman şi perceperea vorbirii şi a imaginilor. Aceste aplicaţii nu pot fi decât parţial rezolvate pe calculatoare clasice.

Este de aşteptat ca reţelele neurale să fie aplicate în procesarea semnalelor şi sisteme expert. Reţelele neurale nu vor înlocui aplicaţiile de inteligenţă artificială de pe calculatoarele clasice, ci vor oferi o tehnologie complementară.

Neurocalculatoarele actuale sunt de multe ori calculatoare convenţionale care execută software de simulare a reţelelor neurale. Alte neurocalculatoare folosesc deja componente (plăci, chip-uri) dedicate. Cele mai interesante sunt, desigur, chip-urile VLSI care implementează reţele neurale. Fabricarea acestor chip-uri este deja actuală. În 1986, AT&T a fabricat primul circuit integrat de memorie neurală.

Capitolul 2

Concepte fundamentale

Există două posibilităţi de a defini reţelele neurale. La o extremă, reţelele neurale sunt o clasă de algoritmi matematici, deoarece o reţea poate fi privită în esenţă ca o notaţie grafică pentru o clasă largă de algoritmi. La cealaltă extremă, reţelele neurale emulează reţelele neurale biologice din organismele vii. În lumina cunoştinţelor limitate pe care le avem în prezent asupra reţelelor neurale biologice, cea mai plauzibilă definiţie se apropie mai mult de cea algoritmică.

Reţelele neurale sunt în mod cert inspirate din biologie, dar există mari diferenţe între reţelele neurale artificiale şi cele naturale. Nu există încă modele care să concureze cu succes performanţele creierului uman. De fapt şi cunoştinţele noastre despre funcţionarea creierului sunt extrem de limitate. Creierul rămâne mai curând o metaforă pentru reţelele neurale dezvoltate până acum.

Cu toate că analogia dintre reţelele neurale artificiale şi cele naturale este vagă, vom începe totuşi prin a menţiona modelul neuronului biologic. Vom defini apoi neuronul artificial şi reţelele neurale artificiale elementare. În fine, vom discuta formele de bază ale procesării în reţele neurale artificiale, cu un accent deosebit pe procesele de învăţare.

2.1 Neuronii biologici şi modelele lor artificiale

Creierul uman constă din aproximativ 10^{11} neuroni. Ei comunică printr-o reţea de conexiuni formate din axoni şi sinapse, având o densitate de aproximativ 10^4 sinapse/neuron. Ipoteza cea mai recentă privind modelarea sistemului nervos natural este că neuronii comunică între ei prin impulsuri electrice. Totodată, neuronii operează într-un mediu chimic. Creierul poate fi considerat o reţea densă de conexiuni electrice condiţionate în mare măsură de procese biochimice. Reţeaua neurală are o structură elaborată, cu interconexiuni foarte complexe. Intrarea în reţea este asigurată de către receptorii senzoriali. Receptorii furnizează stimuli atât din partea corpului cât şi din partea organelor senzoriale care preiau stimulii lumii exterioare. Stimulii au forma impulsurilor electrice care conduc informaţia în reţeaua de neuroni. Ca rezultat al procesării informaţiei în sistemul nervos central, efectorii sunt controlaţi şi dau răspunsuri sub forma diferitelor

acţiuni. Avem deci un sistem constând din receptori, reţeaua neurală şi efectori, care controlează organismul şi acţiunile sale. Fluxul informaţional este descris în figura 2.1.

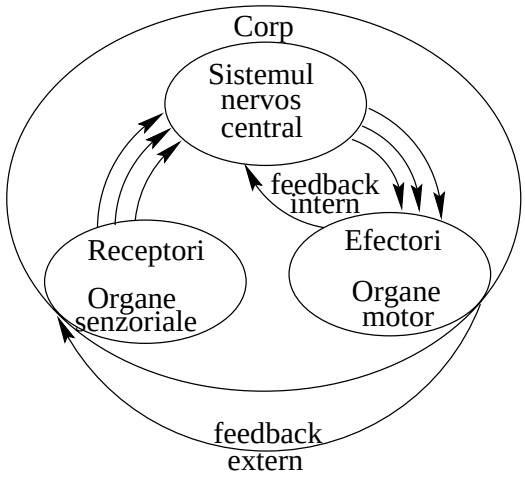


Figura 2.1: Fluxul informaţional în sistemul nervos.

Neuronul biologic

Neuronul este celula nervoasă şi are trei componente:

- *soma* - corpul celulei

- *axonul* - fibră lungă care serveşte ca linie de comunicaţie

- *dendritele*

Dendritele formează un arbore de fibre fine în jurul corpului neuronului. Dendritele recepţionează informaţia de la alţi neuroni prin axonii acestora. Axonul este o conexiune cilindrică lungă care în partea finală devine arborescentă. Fiecare ramură are o terminaţie care aproape atinge dendritele neuronilor vecini. *Sinapsa* este interfaţa prin care neuronul îşi introduce semnalul către dendrita altui neuron. Semnalele care ajung la o sinapsă plecând de la dendritele neuronului respectiv sunt impulsuri electrice (fig. 2.2).

Transmisia interneuronală este uneori electrică dar de obicei este efectuată prin eliberarea de transmiţători chimici la sinapse. Astfel, terminaţia axonului generează substanţa chimică, care afectează neuronul receptor. Neuronul receptor fie generează un impuls către axonul său, fie nu produce nici un răspuns.

Neuronul este capabil să răspundă totalului intrărilor sale agregate într-un scurt interval de timp numit *perioadă de latenţă*. Răspunsul neuronului este generat dacă totalul potenţialului membranei sale atinge un anumit nivel. Membrana poate fi considerată ca o învelitoare care agregă magnitudinea semnalelor

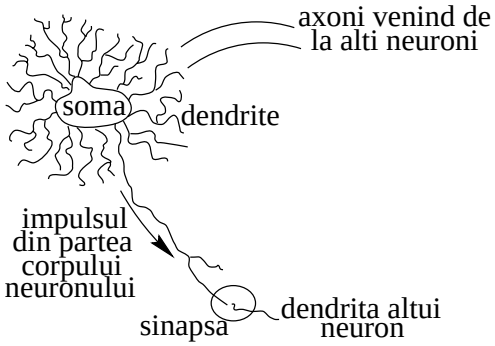


Figura 2.2: Modelul neuronului biologic.

care intră pe parcursul unei anumite durate de timp. Neuronul generează un impuls-răspuns şi îl transmite axonului său numai dacă condiţiile necesare sunt îndeplinite.

Impulsurile care intră în neuron pot fi *excitatoare* – dacă cauzează generarea de către neuron a unui impuls, sau *inhibitoare* – dacă împiedică generarea unui astfel de impuls. O condiţie mai precisă pentru ca neuronul să genereze un impuls este ca excitaţia să depăşească inhibiţia cu o valoare de aproximativ 40 mV, numită *pragul neuronului*. Deoarece o conexiune sinaptică produce reacţia de excitaţie sau de inhibiţie a neuronului receptor, este practic să ataşăm ponderile ± 1 acestor conexiuni. **Neuronul generează un impuls atunci când suma ponderilor impulsurilor receptate depăşeşte valoarea pragului pe parcursul perioadei de însumare latentă.**

Procesarea în reţelele neurale biologice este complexă şi mai puţin structurată decât calculul digital. Spre deosebire de cazul calculului digital, impulsurile neurale nu sunt sincronizate în timp. O caracteristică importantă a neuronului biologic este că semnalele generate nu diferă în magnitudine. Cu alte cuvinte, informaţia transmisă între celulele nervoase este sub forma semnalelor binare.

După transmiterea unui impuls, axonul rămâne pentru un timp într-o staree de neexcitabilitate completă, acest interval numindu-se *perioadă refractară*. Putem diviza timpul în intervale consecutive, fiecare de durata perioadei refractare. Aceasta ne permite o descriere discretă a comportării neuronului. De exemplu, putem preciza care neuroni vor genera impulsuri la momentul $k+1$ bazându-ne pe condiţiile de excitaţie de la momentul k.

Neuronul va fi excitat la un anumit moment dat, dacă numărul sinapselor excitate excitatoare depăşeşte numărul sinapselor excitate inhibitoare la momentul precedent cu cel puţin numărul T, unde T este valoarea pragului neuronului.

Intervalele de timp pot fi luate de ordinul milisecundelor. Perioada refractară nu este însă uniformă: depinde de tipul de neuroni şi de modul în care sunt ei conectaţi. Avem deci o reţea densă de neuroni interconectaţi care generează semnale asincrone. Semnalele sunt transmise apoi către neuronii vecini dar sunt

şi retransmise (feedback) neuronilor generatori.

Această discuţie este o simplificare foarte mare din punct de vedere neurobiologic. Reţelele neurale artificiale sunt mult mai simple decât corespondentul lor natural. Să examinăm un model de neuron artificial cu semnificaţie istorică.

Modelul neural McCulloch-Pitts

Este prima definiţie formală a unui neuron artificial (1943).

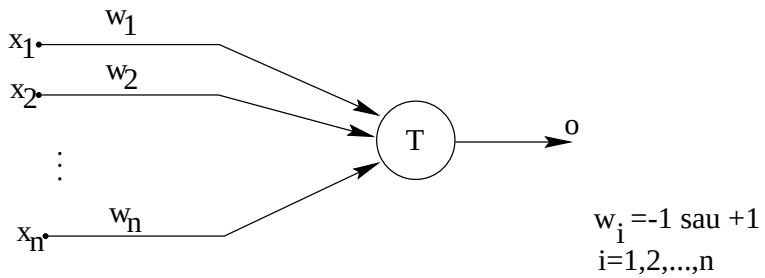


Figura 2.3: Neuronul McCulloch-Pitts.

Intrările $x_i^k, i = 1, 2 \ldots n$ sunt 0 sau 1, în funcţie de absenţa sau prezenţa impulsului la momentul k (fig 2.3). Semnalul de ieşire al neuronului este o. Regula după care neuronul generează un semnal este:

$$o^{k+1} = \begin{cases} 1 & \text{dacă} \quad \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{dacă} \quad \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

$w_i = 1$ pentru o sinapsă excitatoare şi $w_i = -1$ pentru o sinapsă inhibitoare.

Cu toate că este foarte simplu, acest model are un remarcabil potenţial computaţional. Poate realiza operaţiile logice NOT, OR şi AND. După cum ştim, orice funcţie logică de mai multe variabile poate fi implementată utilizând sau NOT şi OR, sau NOT şi AND. De exemplu, funcţiile NOR şi NAND pot fi implementate prin reţele de neuroni conform modelelor din figura 2.4.

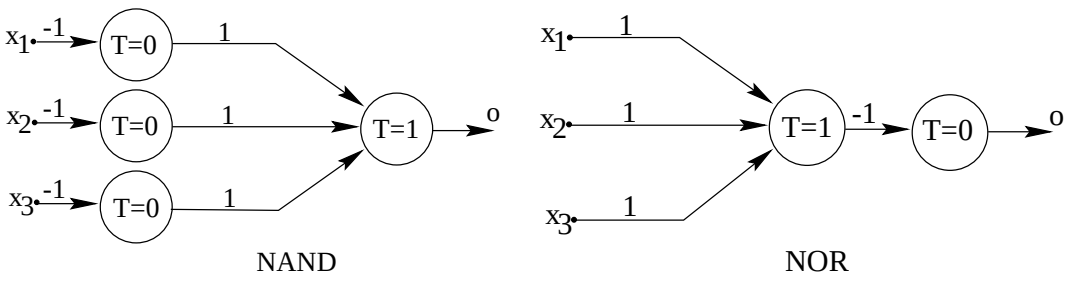


Figura 2.4: Funcţiile NOR şi NAND.

Neuronul McCulloch-Pitts are o întârziere cu durata de o unitate. Această proprietate permite construirea circuitelor digitale secvenţiale. Notăm pentru început că un singur neuron, cu o singură intrare x, cu ponderea şi valoarea de prag unitare, calculează $o^{k+1} = x^k$. Un astfel de neuron se comportă ca un registru simplu, capabil să reţină intrarea pentru un interval de timp de o unitate. O celulă de memorie se construieşte ca în figura 2.5.

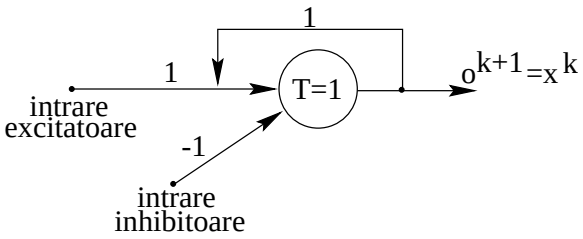


Figura 2.5: Celula de memorie.

După ce s-a iniţializat celula, astfel încât să genereze sau să nu genereze un semnal, această valoare de ieşire este susţinută indefinit, în absentă unor intrări.

Hardware-ul unui calculator digital de orice complexitate poate fi obţinut prin utilizarea unei reţele neurale constituite din blocuri elementare pentru operaţii logice şi pentru memorie. Ne interesează, însă altceva: care este puterea de calcul a reţelelor neurale ţinând cont de capacitatea lor de a învăţa. Vom reveni asupra acestor aspecte mai târziu.

Modelarea unui neuron general

Modelul neural McCulloch-Pitts este elegant şi are o expresie matematică precisă. El operează însă câteva simplificări drastice. Astfel, permite doar stări binare (0 şi 1), presupune că timpul este discret şi presupune sincronismul operaţiilor tuturor neuronilor. De asemenea, ponderile şi pragurile sunt presupuse fixe. În continuare, vom prezenta generalizări ale modelului McCulloch-Pitts, care vor fi de altfel modelele noastre operaţionale.

Fiecare model neural constă dintr-un element de procesare cu conexiuni sinaptice de intrare şi cu o singură ieşire. Intrările şi ieşirile sunt unidirecţionale. Definim modelul unui neuron general ca in figura 2.6

$$o = f(\mathbf{w}^t\mathbf{x}),$$

sau

$$o = f(\sum_{i=1}^{n} w_i x_i)$$

unde $\mathbf{w}$ este vectorul de ponderi definit astfel:

$$\mathbf{w} = [w_1 w_2 \dots w_n]^t$$

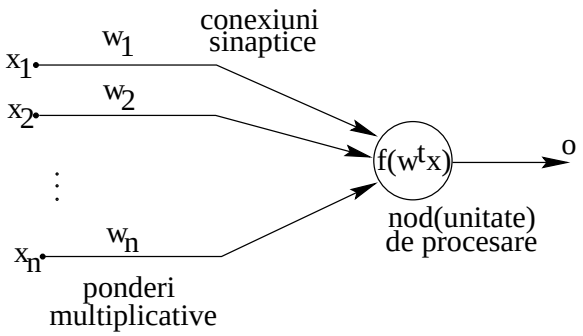


Figura 2.6: Modelul unui neuron general.

iar $\mathbf{x}$ este vectorul de intrări:

$$\mathbf{x} = [x_1 x_2 \dots x_n]^t.$$

Toţi vectorii din acest curs sunt vectori coloană, iar indicele t notează o transpunere. Funcţia $f(\mathbf{w}^t\mathbf{x})$ este *funcţia de activare*. Domeniul ei de definiţie este mulţimea valorilor de activare, net, a unui model neural:

$$net = \mathbf{w}^t\mathbf{x}.$$

De aceea, folosim şi notaţia $f(net)$. Variabila net este analogul potenţialului membranei neuronului biologic.

Convenim că există $n-1$ conexiuni sinaptice şi că $x_n = -1$, $w_n = T$. Uneori vom extrage explicit pragul T ca parametru separat. De acum încolo, vom înţelege prin neuroni – modele de neuroni, iar prin reţele neurale – reţele neurale artificiale compuse din modele de neuroni.

Funcţii de activare tipice sunt:

- *bipolară continuă*: $f(net) = \frac{2}{1+\exp^{-\lambda net}} - 1, \lambda > 0$

- *bipolară binară*: $f(net) = \text{sgn}(net) = \begin{cases} 1 & net > 0 \\ -1 & net < 0 \end{cases}$

Se observă că pentru $\lambda \to \infty$, funcţia bipolară continuă devine bipolar binară.

Funcţiile unipolare sunt:

- *unipolară continuă*: $f(net) = \frac{1}{1+\exp^{-\lambda net}}$

- *unipolară binară*: $f(net) = \begin{cases} 1 & net > 0 \\ 0 & net < 0 \end{cases}$

Pentru $\lambda \to \infty$, funcţiile continue devin funcţii discrete. Cele două funcţii continue se numesc şi *caracteristici sigmoidale*.

Majoritatea neuronilor utilizează funcţii de activare bipolare. Desigur, funcţiile de activare pot fi definite şi altfel.

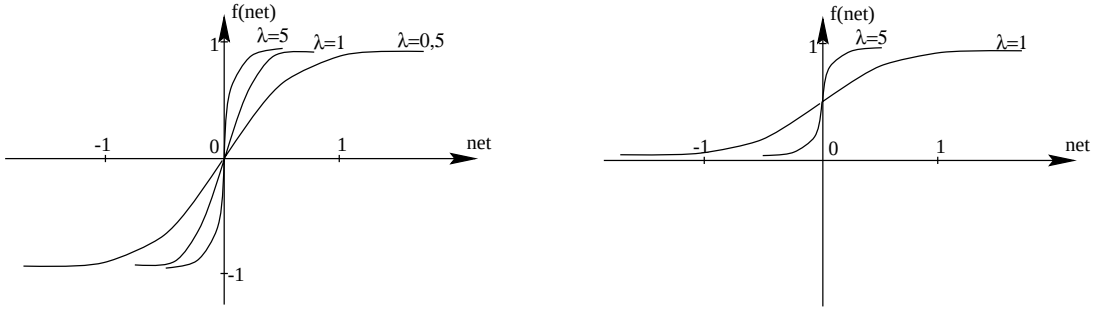


Figura 2.7: Funcţii de activare continue bipolară şi unipolară.

Dacă funcţia de activare este bipolară binară, putem folosi reprezentarea din figura 2.8 a unui *perceptron binar*, iar pentru funcţia bipolară continuă putem folosi reprezentarea din figura 2.9 a unui *perceptron continuu*.

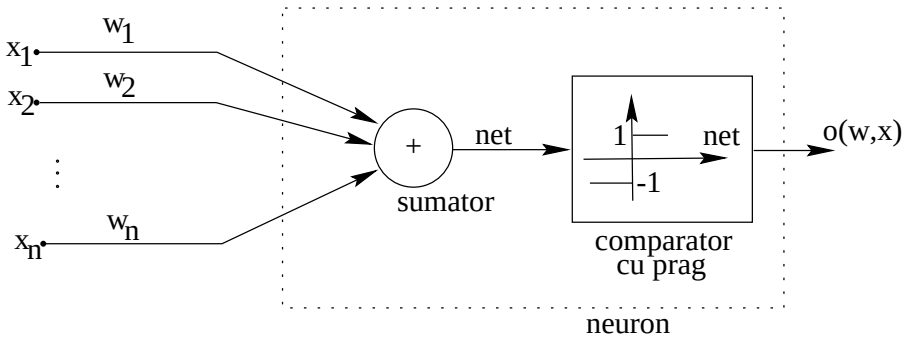


Figura 2.8: Perceptron binar.

Perceptronul discret a fost introdus de Rosenblatt (1958) şi a fost prima maşină instruibilă.

Ieşirile neuronilor pot fi discrete (binare) sau continue. Pentru un strat de m neuroni, valorile lor de ieşire o_1, o_2, ..., o_m pot fi date de:

$$\mathbf{o} = [o_1, o_2 \ldots o_m].$$

Domeniul de definiţie al vectorilor $\mathbf{o}$ este un spaţiu m-dimensional definit pentru cazul continuu astfel:

$$(-1,1)^m \equiv \{\mathbf{o} \in \Re^m, o_i \in (-1,1)\} \quad \text{pentru cazul bipolar}$$

sau

$$(0,1)^m \equiv \{\mathbf{o} \in \Re^m, o_i \in (0,1)\} \quad \text{pentru cazul unipolar.}$$

Domeniul lui $\mathbf{o}$ este interiorul unui cub m-dimensional.

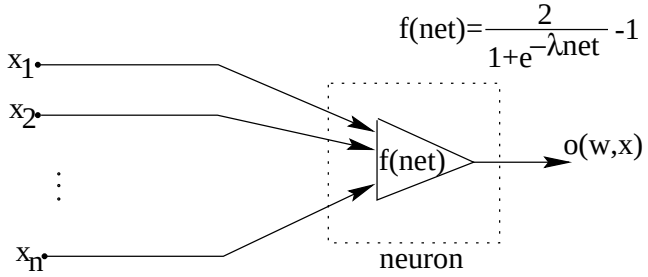


Figura 2.9: Perceptron continuu.

Pentru cazul discret, avem:

$$\{-1,1\}^m \equiv \{\mathbf{o} \in \Re^m, o_i \in \{-1,1\}\} \quad \text{pentru cazul bipolar}$$

sau

$$\{0,1\}^m \equiv \{\mathbf{o} \in \Re^m, o_i \in \{0,1\}\} \quad \text{pentru cazul unipolar.}$$

Domeniul lui $\mathbf{o}$ este format din vârfurile unui cub m-dimensional. Un vector poate avea deci 2^m valori.

2.2 Modelarea reţelelor neurale

Cunoscând acum definiţia modelului unui neuron general, putem defini o reţea neurală ca *o interconectare de neuroni astfel încât ieşirea fiecărui neuron este conectată, via ponderi, cu toţi neuronii, inclusiv cu neuronul respectiv.*

Reţea feedforward

Să considerăm o *arhitectură feedforward* de m neuroni care recepţionează n intrări (fig. 2.10). Pentru acest model definim

$$\mathbf{o} = [o_1 o_2 \dots o_m]^t$$

şi

$$\mathbf{x} = [x_1 x_2 \dots x_n]^t.$$

Ponderea w_{ij} caracterizează al i-lea neuron cu a j-a intrare. Valoarea de activare pentru al i-lea neuron este atunci:

$$\mathbf{net}_i = \sum_{j=1}^{n} w_{ij} x_j \qquad i = 1, 2, \dots, m.$$

Fie $\mathbf{w}_i = [w_{i1} w_{i2} \dots w_{in}]^t$, atunci:

$$\mathbf{net}_i = \mathbf{w}_i^t \mathbf{x}, \quad o_i = f(\mathbf{w}_i^t \mathbf{x}), \qquad i = 1, 2, \dots, m.$$

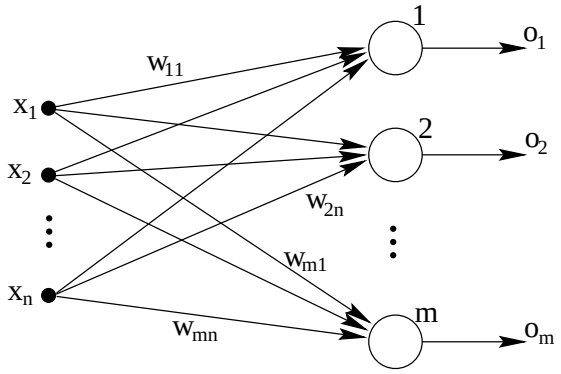


Figura 2.10: Reţea neurală feedforward.

Definim operatorul matricial neliniar Γ pentru care:

$$\mathbf{o} = \Gamma(\mathbf{W}\mathbf{x}),$$

unde $\mathbf{W}$ este matricea ponderilor (matricea conexiunilor):

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & & & \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}$$

iar

$$\Gamma[\cdot] = \begin{bmatrix} f(\cdot) & 0 & \dots & 0 \\ 0 & f(\cdot) & \dots & 0 \\ \vdots & & & \\ 0 & \dots & 0 & f(\cdot) \end{bmatrix}.$$

Funcţiile $f(\cdot)$ sunt funcţii neliniare de activare.

Vectorii $\mathbf{x}$, $\mathbf{o}$ sunt numiţi şi pattern-uri de intrare, respectiv de ieşire. Transformarea unui pattern de intrare într-un pattern de ieşire are loc fără întârziere, instantaneu. De aceea, spunem că o astfel de reţea este de tip feedforward. Avem (fig. 2.11):

$$\mathbf{o}(t) = \Gamma[\mathbf{W}\mathbf{x}(t)]$$

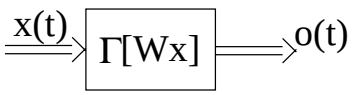


Figura 2.11: Diagrama bloc a unei reţele neurale feedforward.

Reţeaua feedforward este caracterizată prin lipsa de feedback. O reţea feedforward poate fi conectată în cascadă pentru a forma o reţea pe mai multe straturi. Într-o astfel de reţea, ieşirea unui strat este intrarea următorului strat.

Reţea feedback

O reţea feedback se poate obţine dintr-o reţea feedforward prin conectarea ieşirilor neuronilor cu propriile intrări (fig. 2.12).

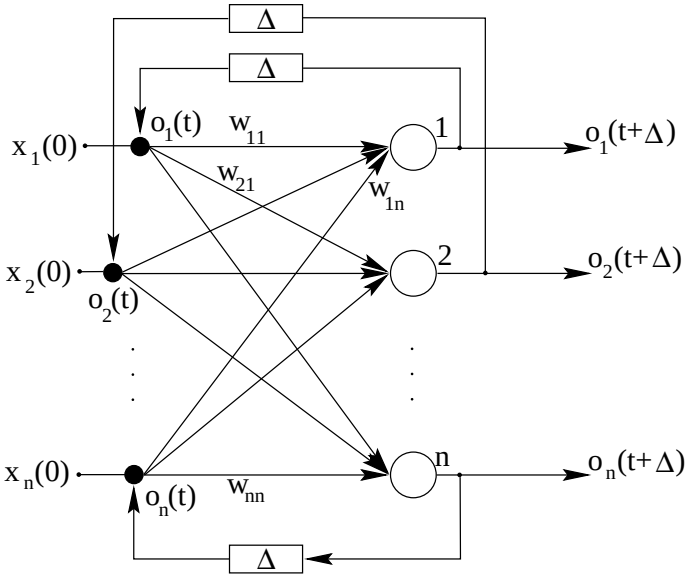


Figura 2.12: Reţea neurală feedback.

Perioada de timp Δ este analogă perioadei refractare a modelului neuronului biologic. Avem (fig. 2.13):

$$\mathbf{o}(t+\Delta) = \Gamma[\mathbf{W}\mathbf{o}(t)].$$

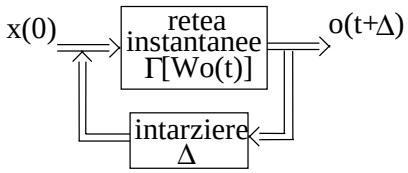


Figura 2.13: Diagrama bloc a reţelei neurale feedback.

Intrarea $\mathbf{x}(t)$ este folosită doar pentru a iniţializa această reţea, astfel încât $\mathbf{o}(0) = \mathbf{x}(0)$. Intrarea este apoi îndepărtată şi, pentru $t > 0$, sistemul devine autonom.

Considerând timpul ca o variabilă discretă şi decidem să observăm funcţionarea reţelei la momentele $\Delta, 2\Delta, 3\Delta, \ldots$, sistemul este cu *timp discret*. Convenţional, putem considera că pasul timpului este unitar. Atunci, notăm:

$$\mathbf{o}^{k+1} = \Gamma(\mathbf{W}\mathbf{o}^k), \qquad \text{pentru } k = 1, 2, \ldots$$

Această reţea este *recurentă* deoarece răspunsul ei la momentul $k+1$ depinde de întregul istoric al reţelei începând cu momentul $k = 0$:

$$\mathbf{o}^1 = \Gamma[\mathbf{W}\mathbf{x}^0]$$

$$\mathbf{o}^2 = \Gamma[\mathbf{W}\Gamma[\mathbf{W}\mathbf{x}^0]]$$

$$\ldots$$

$$\mathbf{o}^{k+1} = \Gamma[\mathbf{W}\Gamma[\ldots\Gamma[\mathbf{W}\mathbf{x}^0]\ldots]].$$

Reţelele recurente operează de obicei cu o reprezentare discretă a datelor şi folosesc neuroni cu o funcţie de activare discretă. Un sistem având intrări cu timp discret şi o reprezentare discretă a datelor se numeşte un *automat*. Deci, reţelele neurale recurente din această categorie pot fi considerate nişte automate.

Numim $\mathbf{o}^1, \mathbf{o}^2, \ldots$ *stări* ale reţelei la momentele 1, 2,... şi ecuaţiile de mai sus oglindesc secvenţa tranziţiilor stărilor. O stare de echilibru se numeşte şi *atractor*. Un atractor constă dintr-o singură stare, sau dintr-un număr limitat de stări. În capitolul 1 am văzut că $\mathbf{o} = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}^t$ este un atractor. Secvenţa de stări ale unei reţele recurente este în general nedeterministă.

O reţea cu *timp continuu* se obţine înlocuind elementele de întârziere discrete cu elemente continue. Fie, de exemplu, reţeaua feedback cu timp continuu din figura 2.14.

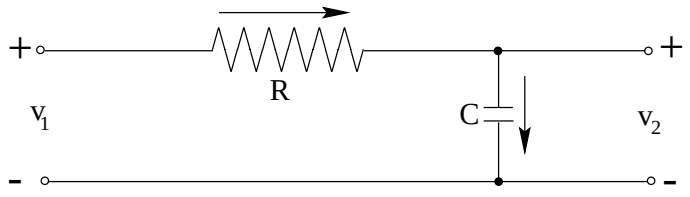


Figura 2.14: Reţea feedback cu timp continuu.

Ea este o reţea electrică constând dintr-o rezistenţă şi un condensator, unde v_1 este tensiunea la intrare, iar v_2 este tensiunea la ieşire. De fapt, reţelele electrice sunt utilizate frecvent pentru a modela calculele efectuate de o reţea neurală. Reţelele electrice posedă flexibilitatea de a modela toate fenomenele liniare şi neliniare întâlnite în acest curs. Din această cauză, reţelele electrice reprezintă modele fizice funcţionale ale reţelelor neurale. Din legea lui Kirchoff obţinem:

$$\frac{v_1 - v_2}{R} = C\frac{dv_2}{dt} \Rightarrow$$

$$\frac{dv_2}{dt} + \frac{v_2}{RC} = \frac{v_1}{RC}. \tag{2.1}$$

De aici, obţinem:

$$\Delta v_2 \cong \frac{\Delta t}{C} \cdot \frac{v_1 - v_2}{R}$$

care reprezintă modificarea tensiunii v_2 în intervalul Δt.

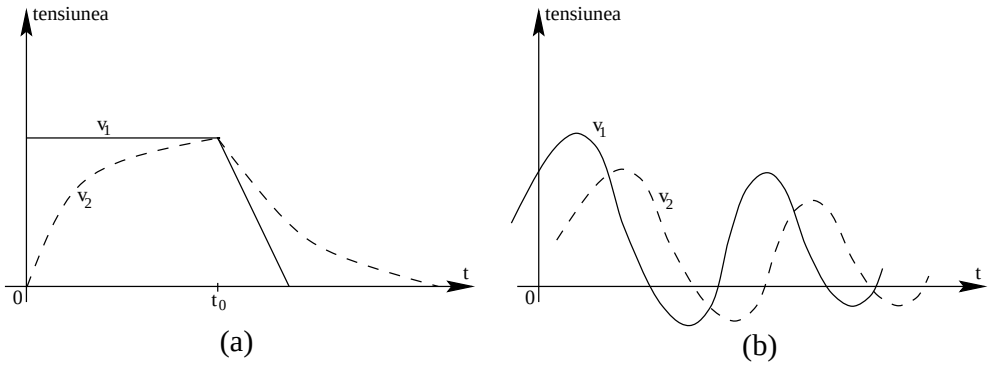


Figura 2.15: Răspunsul în timp (a) la un impuls şi (b) la o tensiune de intrare de tip undă armonică al reţelei cu timp continuu din figura anterioară.

De obicei, reţelele cu timp continuu folosesc neuroni cu funcţii de activare continue. În figura 2.16 avem o conexiune sinaptică bazată pe circuitul electric descris în figura 2.14.

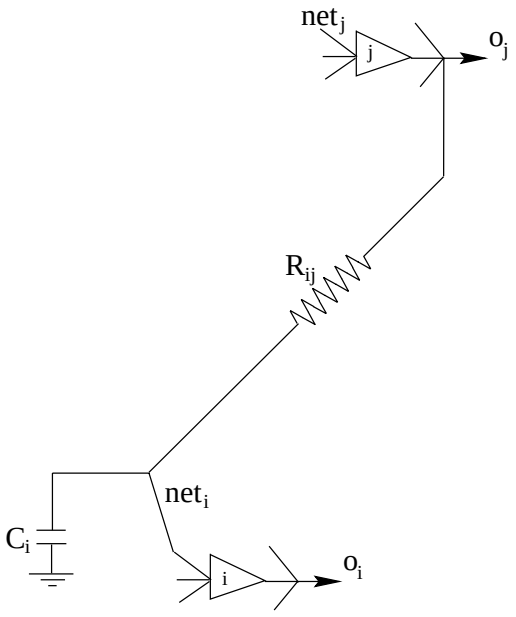


Figura 2.16: Conexiune sinaptică.

Rezistenţa R_{ij} serveşte ca pondere de la ieşirea neuronului j către intrarea neuronului i. Ecuaţia 2.1 poate fi discretizată astfel:

$$\frac{net_i^{k+1} - net_i^k}{\Delta t} \cong \frac{1}{R_{ij}C_i}(\mathbf{o}_j^k - net_i^k).$$

De aici:

$$net_i^{k+1} \cong net_i^k + \frac{\Delta t}{R_{ij}C_i}(\mathbf{o}_j^k - net_i^k).$$

O reţea de mai mulţi neuroni de acest tip are, din punct de vedere dinamic, o comportare complexă.

2.3 Învăţare şi adaptare

În general, învăţarea este schimbarea comportamentului datorită experienţei. La om şi animale procesul de învăţare nu poate fi observat direct, ci presupunem că a avut loc observând modificările de comportament. La reţelele neurale, procesul de învăţare este mai direct: putem observa fiecare pas al învăţării ca o relaţie distinctă de tip cauză-efect.

Un cadru general pentru procesul de învăţare la reţele neurale este dat de teoria aproximării.

Învăţarea ca aproximare

Teoria aproximării se referă la aproximarea unei funcţii continue de mai multe variabile $h(\mathbf{x})$ printr-o altă funcţie $H(\mathbf{w}, \mathbf{x})$, unde $\mathbf{x} = [x_1\ x_2 \ldots x_n]^t$ este vectorul de intrare şi $\mathbf{w} = [w_1\ w_2 \ldots w_m]^t$ este vectorul parametrilor (ponderilor). Scopul învăţării este să se găsească $\mathbf{w}$ pentru care se obţine cea mai bună aproximare a lui $h(\mathbf{x})$, având la dispoziţie o mulţime de exemple, $\{\mathbf{x}\}$, folosite pentru învăţare. O problemă importantă este alegerea funcţiei $H(\mathbf{w}, \mathbf{x})$, aceasta numindu-se *problema reprezentării*. După alegerea lui $H(\mathbf{w}, \mathbf{x})$, se aplică algoritmul de învăţare al reţelei pentru găsirea parametrilor optimi $\mathbf{w}^*$:

$$\rho[H(\mathbf{w}^*, \mathbf{x}), h(\mathbf{x})] \leq \rho[H(\mathbf{w}, \mathbf{x}), h(\mathbf{x})]$$

unde ρ este o metrică (o distanţă).

Reţelele feedback sunt sisteme dinamice. De aceea, învăţarea în acest caz se referă la învăţarea stărilor de echilibru.

Învăţarea supervizată şi învăţarea nesupervizată

Majoritatea reţelelor neurale pe care le discutăm învaţă incremental, pas cu pas. Dacă ponderile reţelei sunt ajustate printr-un singur pas, atunci informaţia feedback produsă de reţea nu mai este necesară şi vorbim de o învăţare de tip *batch*, adică "la grămadă".

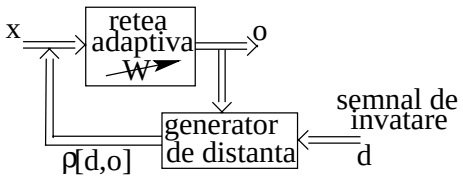


Figura 2.17: Învăţare supervizată.

Ne referim în continuare la învăţarea incrementală, în care conceptul de feedback are un rol central.

În *învăţarea supervizată* (fig. 2.17), presupunem că la fiecare moment când se aplică intrarea, răspunsul **d** al sistemului este dat de către instructor (un factor uman).

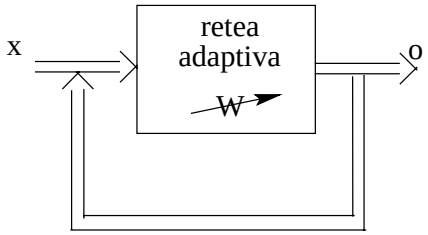


Figura 2.18: Învăţare nesupervizată.

Învăţarea nesupervizată (fig. 2.18) se foloseşte de multe ori, de exemplu, la *clustering* (fig. 2.19), atunci când informaţia *a priori* este minimă.

2.4 Reguli de învăţare

Să studiem procesul de instruire a vectorului w_i de ponderi, având componentele w_{ij} (unde w_{ij} este ponderea conexiunii celei de-a j-a intrări cu cel de-al i-lea neuron) (fig. 2.20).

Fie următoarea regulă generală de învăţare: Vectorul $\mathbf{w}_i = [w_{i1}\ w_{i2} \ldots w_{in}]^t$ creşte proporţional cu produsul intrării $\mathbf{x}$ şi a semnalului de învăţare r. Semnalul r este in general o funcţie de $\mathbf{w}_i$, $\mathbf{x}$ şi, daca este cazul, d_i:

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i).$$

Incrementul lui $\mathbf{w}_i$ produs la momentul t este:

$$\Delta\mathbf{w}_i(t) = cr[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)]\mathbf{x}(t)$$

unde c este *constanta de învăţare*, pozitivă, care determină rata învăţării. Atunci:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + cr[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)]\mathbf{x}(t).$$

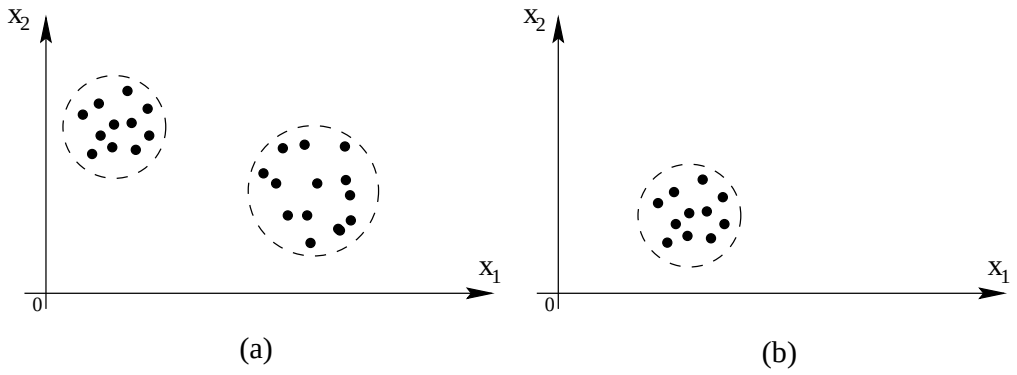


Figura 2.19: (a) Pattern-uri bidimensionale care formează două clustere. (b) Aparent, nu se disting mai multe clustere şi nu ştim câte sunt. În acest caz învăţarea nesupervizată nu este indicată.

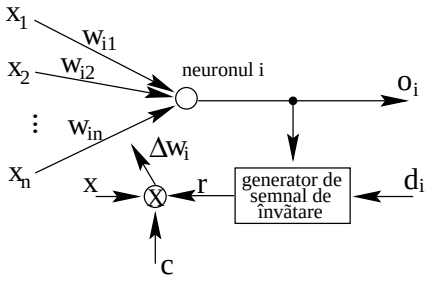


Figura 2.20: Modelul instruirii în cazul învăţării supervizate.

Pentru cazul discret:

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + cr(\mathbf{w}_i^k, \mathbf{x}_i^k, d_i^k)\mathbf{x}^k.$$

Pentru cazul continuu:

$$\frac{d\mathbf{w}_i(t)}{dt} = cr\mathbf{x}(t).$$

Vom studia în continuare învăţarea cu timp discret (pas cu pas). Se presupune că ponderile sunt iniţializate convenabil, înainte ca procesul de învăţare să înceapă.

Regula de învăţare a lui Hebb (1949)

Această regulă particularizează semnalul de învăţare astfel:

$$r = f(\mathbf{w}_i^t\mathbf{x}),$$

adică semnalul de învăţare este chiar ieşirea neuronului. Atunci:

$$\Delta\mathbf{w}_i = cf(\mathbf{w}_i^t\mathbf{x})\mathbf{x}$$

$$\Delta w_{ij} = cf(\mathbf{w}_i^t\mathbf{x})x_j = co_i x_j, \qquad \text{pentru } j = 1, 2, \ldots, n.$$

Este necesar ca, iniţial, ponderile să aibă valori aleatoare mici. Învăţarea după regula lui Hebb este de tip feedforward şi nesupervizată. Este implementată următoarea idee:

> "Dacă celula A excită în mod repetat celula B, făcând-o să genereze un impuls, atunci are loc un proces de creştere (schimbare metabolică) într-una sau în ambele celule, astfel încât eficienţa excitării lui B de către A creşte."

Cu alte cuvinte, pattern-urile de intrare mai frecvente vor avea şi influenţa cea mai mare asupra ponderilor conexiunilor.

Regula de învăţare a perceptronului (Rosenblatt, 1958)

Semnalul de învăţare pentru această regulă este:

$$r = d_i - o_i$$

unde $o_i = sgn(\mathbf{w}_i^t\mathbf{x})$ şi d_i este răspunsul dorit (pentru cazul bipolar, ± 1) (fig. 2.21).

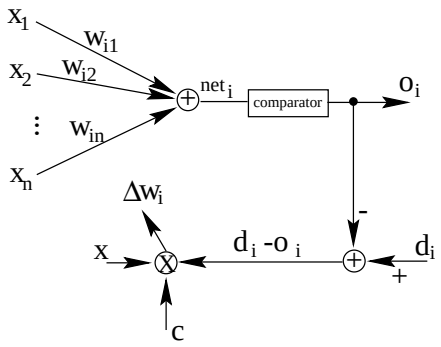


Figura 2.21: Regula de învăţare a perceptronului.

Avem:

$$\Delta\mathbf{w}_i = c[d_i - sgn(\mathbf{w}_i^t\mathbf{x})]\mathbf{x}$$

$$\Delta w_{ij} = c[d_i - sgn(\mathbf{w}_i^t\mathbf{x})]x_j, \qquad \text{pentru } j = 1, 2, \ldots, n.$$

Să observăm că această regulă este aplicabilă doar pentru situaţia când funcţia de activare este binară. Are loc o ajustare a ponderilor dacă şi numai dacă o_i este incorect. Deoarece răspunsul dorit poate fi 1 sau -1, avem:

$$\Delta\mathbf{w}_i = 2c\mathbf{x}, \qquad \text{dacă } d_i = 1 \text{ şi } sgn(\mathbf{w}_i^t\mathbf{x}) = -1$$

$$\Delta\mathbf{w}_i = -2c\mathbf{x}, \qquad \text{dacă } d_i = -1 \text{ şi } sgn(\mathbf{w}_i^t\mathbf{x}) = 1$$

Dacă $d_i = sgn(\mathbf{w}_i^t\mathbf{x})$, se observă că $\Delta w_i = 0$.

Regula de învăţare a perceptronului este importantă pentru învăţarea supervizată a reţelelor neurale. Ponderile sunt iniţializate cu orice valoare.

Regula de învăţare delta (McClelland, Rummelhart, 1986)

Este valabilă pentru funcţii de activare continue şi pentru învăţarea supervizată. Semnalul de învăţare pentru această regulă (fig. 2.22) este numit *delta* şi este dat de

$$r = [d_i - f(\mathbf{w}_i^t \mathbf{x})] f'(\mathbf{w}_i^t \mathbf{x}).$$

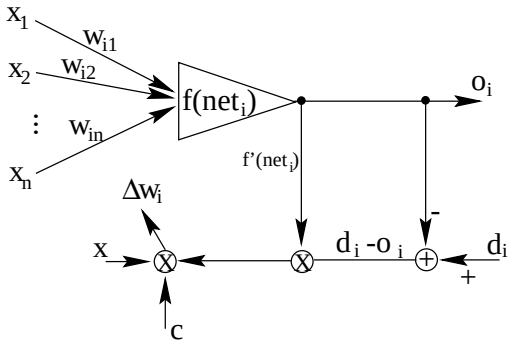


Figura 2.22: Regula de învăţare delta.

Această regulă se deduce din condiţia de cea mai mică eroare pătratică dintre o_i şi d_i. Fie E eroarea pătratică:

$$E = \frac{1}{2}(d_i - o_i)^2,$$

ceea ce este echivalent cu

$$E = \frac{1}{2}[d_i - f(\mathbf{w}_i^t \mathbf{x})]^2.$$

De aici,

$$\nabla E = -(d_i - o_i) f'(\mathbf{w}_i^t \mathbf{x}) \mathbf{x}.$$

Componentele gradientului erorii sunt:

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i) f'(\mathbf{w}_i^t \mathbf{x}) x_j, \qquad \text{pentru } j = 1, 2, \ldots, n$$

şi reprezintă panta pe direcţia w_{ij}. Alegem termenul Δw_{ij} astfel încât să fie proporţional cu panta negativă (deci să minimizăm pe E) şi obţinem:

$$\Delta w_{ij} = \eta (d_i - o_i) f'(net_i) x_j, \qquad j = 1, 2, \ldots, n$$

$$\Delta \mathbf{w}_i = \eta (d_i - o_i) f'(net_i) \mathbf{x} \tag{2.2}$$

$$\Delta \mathbf{w}_i = -\eta \nabla E$$

unde η este o constantă pozitivă.

Presupunând că semnalul de învăţare este

$$r = [d_i - f(\mathbf{w}_i^t \mathbf{x})] f'(\mathbf{w}_i^t \mathbf{x})$$

obţinem:

$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(net_i)\mathbf{x}$$

ceea ce este identic cu 2.4, c şi η fiind constante arbitrare.

Regula delta este de fapt o transpunere a regulii de învăţare a perceptronului discret la cazul perceptronului continuu. Ponderile pot fi iniţializate cu orice valoare.

Regula de învăţare Windrow-Hoff (1962)

Această regulă este aplicabilă pentru învăţarea supervizată. Este independentă de funcţia de activare folosită deoarece minimizează eroarea pătratică dintre răspunsul dorit d_i şi valoarea de activare a neuronului $net_i = \mathbf{w}_i^t \mathbf{x}$. Semnalul de învăţare este

$$r = d_i - \mathbf{w}_i^t \mathbf{x}.$$

Atunci,

$$\Delta \mathbf{w}_i = c(d_i - \mathbf{w}_i^t \mathbf{x})\mathbf{x}$$

$$\Delta w_{ij} = c(d_i - \mathbf{w}_i^t \mathbf{x}) x_j, \qquad j = 1, 2, \ldots, n.$$

Această regulă poate fi considerată un caz special al regulii delta: presupunem $f(\mathbf{w}_i^t \mathbf{x}) = \mathbf{w}_i^t \mathbf{x}$, adică funcţia de activare este funcţia identică $f(net) = net$ şi obţinem $f'(net) = 1$.

Ponderile pot fi iniţializate oricum.

Regula corelaţiei

Substituind $r = d_i$ în regula generală de învăţare obţinem regula corelaţiei:

$$\Delta \mathbf{w}_i = c d_i \mathbf{x}$$

$$\Delta w_{ij} = c d_i x_j, \qquad j = 1, 2, \ldots, n.$$

De obicei, regula corelaţiei se aplică în reţele de memorie cu neuroni cu funcţie de activare binară. Se poate interpreta ca un caz particular al regulii lui Hebb în care funcţia de activare este binară şi $o_i = d_i$. Totuşi, învăţarea hebbiană este nesupervizată, în timp ce regula corelaţiei se aplică la învăţarea supervizată. Ca şi în cazul învăţării hebbiene, este necesar ca ponderile să fie iniţializate cu valori apropiate de 0.

Regula "*câştigătorul ia tot*"

Această regulă diferă mult de regulile prezentate până acum. Poate fi explicată pe o mulţime de neuroni şi este un exemplu de învăţare competitivă folosită în instruirea nesupervizată. În general, această regulă se foloseşte pentru învăţarea proprietăţilor statistice ale intrarilor.

Învăţarea se bazează pe premiza că unul din neuroni, fie el neuronul m, are răspunsul maxim pentru intrarea $\mathbf{x}$. Acest neuron este *câştigătorul* (fig. 2.23).

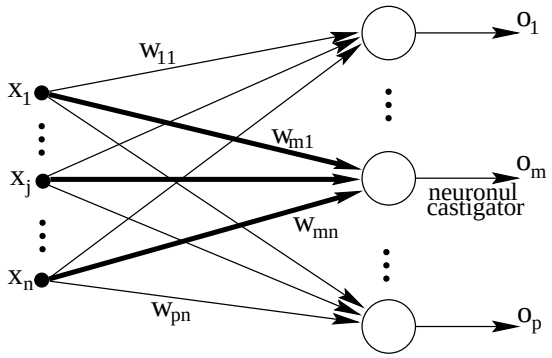


Figura 2.23: Regula de învăţare câştigătorul ia tot.

Ca rezultat,

$$\mathbf{w}_m = [w_{m1}\ w_{m2} \ldots w_{mn}]^t$$

este singurul vector ajustat:

$$\Delta \mathbf{w}_m = \alpha(\mathbf{x} - \mathbf{w}_m)$$

$$\Delta w_{mj} = \alpha(x_j - w_{mj}), \qquad \text{pentru } j = 1, 2, \ldots, n$$

unde $\alpha > 0$ este o constantă de învăţare mică. Criteriul de alegere a câştigătorului este:

$$\mathbf{w}_m^t \mathbf{x} = \max_{i=1,2,\ldots,p} (\mathbf{w}_i^t \mathbf{x}).$$

Numai ponderile neuronului câştigător se modifică. După ajustare, w_m va estima şi mai bine pattern-ul $\mathbf{x}$.

Ponderile se iniţializează cu valori aleatoare şi sunt normalizate. În general, α scade pe parcursul procesului de învăţare.

Paradigme ale învăţarii în reţele neurale

1. *Autoasociere.* O mulţime de pattern-uri este prezentată în mod repetat şi sistemul *memorează* aceste pattern-uri. Apoi, un pattern similar cu unul din pattern-urile memorate este prezentat. Sarcina este de a *regăsi* pattern-ul originar.

Tabelul 2.1: Comparaţie între calculul convenţional şi calculul neural din punct de vedere al etapelor care trebuie parcurse.

Sarcina	**Calcul convenţional**	**Calcul neural**
Rezolvarea unei probleme	Formularea algoritmului	Selectarea unei arhitecturi şi definirea mulţimii de exemple reprezentative
Achiziţionarea cunoştinţelor	Programare	Instruire
Calcul	Aritmetic, precizie mare	Precizie mică, aplicaţii neliniare
Memorarea datelor	ROM, RAM - memorii binare	Ponderi ale conexiunilor având în general valori continue

2. *Heteroasociere.* Este o variantă a autoasocierii. O mulţime de perechi de pattern-uri este prezentată în mod repetat. Paradigmele autoasocierii şi heteroasocierii sunt tipice memoriilor asociative.

3. *Clasificare.* Paradigma clasificării poate fi considerată o variantă a autoasocierii. În acest caz avem o mulţime fixată de categorii în care trebuie clasificate pattern-urile de intrare. În perioada instruirii, sistemului i se prezintă pattern-uri din secvenţa de instruire precizând categoriile din care fac parte. Scopul este de a învăţa corect clasificarea, astfel încât, ulterior, orice pattern de intrare (eventual perturbat faţă de pattern-urile din secvenţa de instruire) să fie clasificat corect. Aceasta este paradigma tipică pentru algoritmul perceptronului şi instruirea prin propagarea în urmă a erorii în reţelele feedforward multistrat. Instruirea este supervizată. Instruirea este mai bună dacă în secvenţa de instruire apar şi pattern-uri perturbate (spre deosebire de paradigma autoasocierii).

4. *Detectare de regularităţi (clustering).* Sistemul trebuie să descopere caracteristici (trăsături) comune într-o mulţime de pattern-uri. Nu există categorii *a priori* de pattern-uri. Sistemul trebuie să-şi construiască singur categoriile. Instruirea este nesupervizată. Numărul clusterelor poate fi, sau poate nu fi, cunoscut *a priori*.

Tabelul 2.2: Principalele reguli de instruire. Constantele c şi α sunt pozitive.

Regula de învăţare	**Ajustarea Δw_{ij}**	**Ponderile iniţiale**	**Tipul învăţării**	**Caracteristicile neuronilor**	**Neuron sau strat**
Hebb	$co_i\mathbf{x}_j \; j = 1, 2, \ldots, n$	0	nesuperv.	oricare	neuron
perceptron	$c[d_i - sgn(\mathbf{w}_i^t\mathbf{x})]x_j \; j = 1, 2, \ldots, n$	oricare	superv.	binar	neuron
delta	$c(d_i - o_i)f'(net_i)x_j \; j = 1, 2, \ldots, n$	oricare	superv.	continuu	neuron
Windrow-Hoff	$c(d_i - \mathbf{w}_i^t\mathbf{x})x_j \; j = 1, 2, \ldots, n$	oricare	superv.	oricare	neuron
corelaţie	$cd_ix_j \; j = 1, 2, \ldots, n$	0	superv.	oricare	neuron
câştigătorul ia tot	$\Delta w_{mj} = \alpha(x_j - w_{mj}) \; j = 1, 2, \ldots, n$	aleatoare normalizate	nesuperv.	continuu	strat de neuroni

2.5 Exemple

1. Exemplu de reţea feedforward pe două straturi cu funcţie de activare bipolară

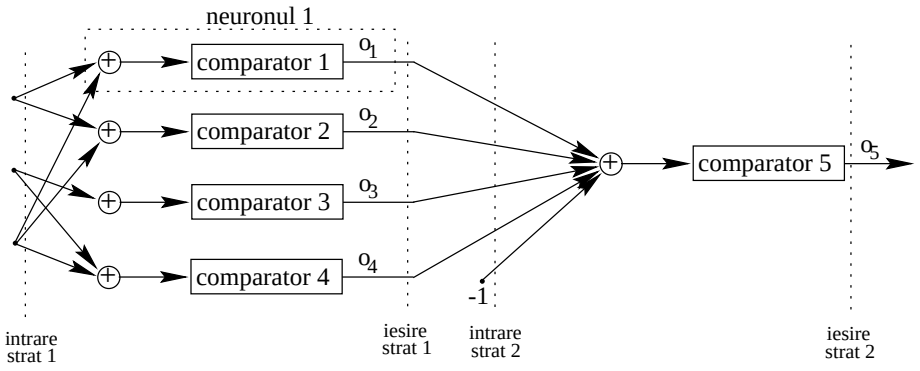


Figura 2.24: Reţea feedforward pe două straturi.

$$\mathbf{o} = \Gamma\left[\mathbf{W}\mathbf{x}\right]$$

Pentru stratul 1:

$$\mathbf{o} = \Gamma\left[\begin{array}{cccc} o_1 & o_2 & o_3 & o_4 \end{array}\right]^t$$

$$\mathbf{x} = \left[\begin{array}{ccc} x_1 & x_2 & -1 \end{array}\right]^t$$

$$W_1 = \left[\begin{array}{ccc} 1 & 0 & 1 \\ -1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & -1 & -3 \end{array}\right]$$

Pentru stratul 2:

$$\mathbf{o} = [o_5]$$

$$\mathbf{x} = \left[\begin{array}{ccccc} o_1 & o_2 & o_3 & o_4 & -1 \end{array}\right]^t$$

$$W_2 = \left[\begin{array}{ccccc} 1 & 1 & 1 & 1 & 3,5 \end{array}\right]^t$$

Răspunsurile primului strat se calculează astfel:

$$\mathbf{o} = \left[\begin{array}{cccc} \mathrm{sgn}(x_1 - 1) & \mathrm{sgn}(-x_1 + 2) & \mathrm{sgn}(x_2) & \mathrm{sgn}(-x_2 + 3) \end{array}\right]^t$$

Răspunsul pentru al doilea strat este:

$$o_5 = \mathrm{sgn}(o_1 + o_2 + o_3 + o_4 - 3,5).$$

Observăm că $o_5 = +1$ dacă şi numai dacă $o_1 = o_2 = o_3 = o_4 = 1$. Rezultă că această reţea este o aplicaţie a spaţiului bidimensional x_1x_2 în punctele ± 1.

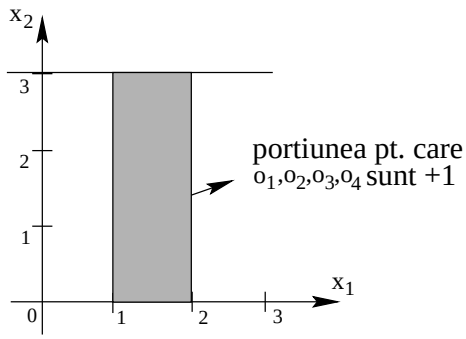


Figura 2.25: Reprezentare în spaţiul bidimensional.

Să presupunem acum că avem aceeaşi arhitectură, dar că neuronii au caracteristici sigmoidale. Obţinem:

$$\mathbf{o} = \begin{bmatrix} \frac{2}{1+e^{(1-x_1)\lambda}} - 1 \\ \frac{2}{1+e^{(x_1-2)\lambda}} - 1 \\ \frac{2}{1+e^{(-x_2)\lambda}} - 1 \\ \frac{2}{1+e^{(x_2-3)\lambda}} - 1 \end{bmatrix}$$

$$o_5 = \frac{2}{1+e^{(3,5-o_1-o_2-o_3-o_4)\lambda}} - 1.$$

Aplicaţia spaţiului bidimensional x_1x_2 într-un segment al axei reale este în acest caz mai complicată.

2. Exemplu de reţea recurentă

Stările reţelei sunt vârfurile unui cub 4-dimensional: $\{-1,1\}^4$. Trecerea dintr-o stare in alta se face de-a lungul unei muchii a hipercubului, deci două stări succesive diferă doar printr-o componentă. Avem:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Presupunând $\mathbf{x}^0 = \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^t$, obţinem

$$\mathbf{o}^1 = \begin{bmatrix} \text{sgn}(3) & \text{sgn}(3) & \text{sgn}(3) & \text{sgn}(-3) \end{bmatrix}^t = \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^t$$

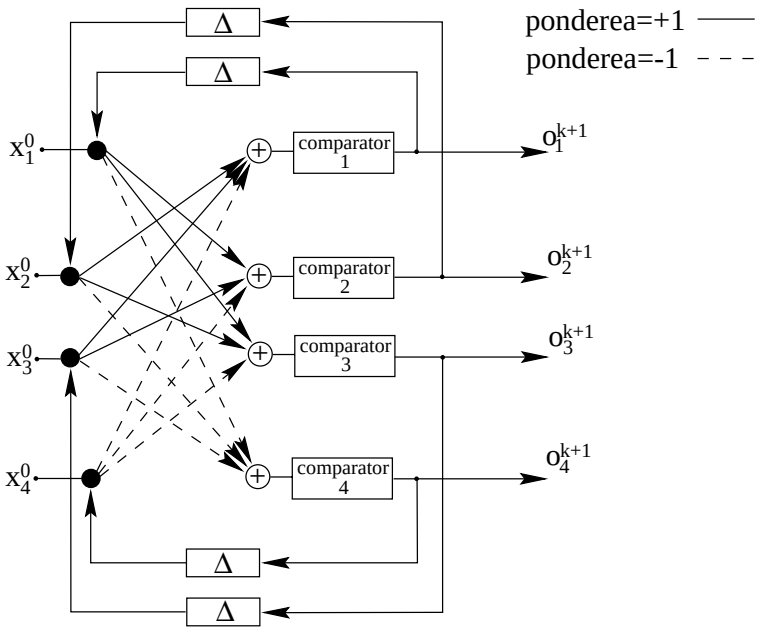


Figura 2.26: Reţea neurală recurentă.

şi deci $\begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^t$ este o stare de echilibru. Tot o stare de echilibru este şi $\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^t$.

Presupunând $\mathbf{x}^0 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^t$, obţinem:

$$\mathbf{o}^1 = \begin{bmatrix} \text{sgn}(1) & \text{sgn}(1) & \text{sgn}(1) & \text{sgn}(-3) \end{bmatrix}^t.$$

Are deci loc tranziţia $\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$ şi se ajunge la cea mai apropiată stare de echilibru.

Verificaţi singuri ce se intâmplă când $\mathbf{x}^0$ ia alte valori:

$$\mathbf{x}^0 = \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}^t$$

$$\mathbf{x}^0 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}^t$$

$$\mathbf{x}^0 = \begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix}^t.$$

3. Exemplu de reţea feedback cu timp continuu

Fie reţeaua din figura 2.27 unde $o_i = f(net_i)$, $i = 1, 2$, iar f este funcţia de activare continuă bipolară. În această figură, R_{12} şi R_{21} sunt ponderi negative, întârzierile sunt realizate de dispozitivul RC, iar rezistenţele R modelează curentul absorbit de neuroni.

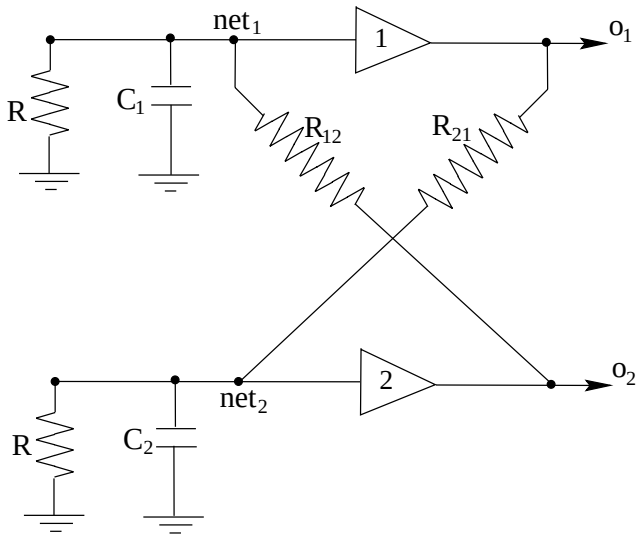


Figura 2.27: Reţea electrică a doi neuroni.

Obţinem:

$$\begin{cases} C_1 \frac{dnet_1}{dt} = \frac{o_2 - net_1}{R_{12}} - \frac{net_1}{R} \\ C_2 \frac{dnet_2}{dt} = \frac{o_1 - net_2}{R_{21}} - \frac{net_2}{R} \end{cases}.$$

Discretizăm:

$$\begin{cases} net_1^{k+1} \cong net_1^k + \frac{\Delta t}{R_{12} C_1}(o_2^k - net_1^k) - \frac{net_1^k}{RC_1}\Delta t \\ net_2^{k+1} \cong net_2^k + \frac{\Delta t}{R_{21} C_2}(o_1^k - net_2^k) - \frac{net_2^k}{RC_2}\Delta t \end{cases}.$$

Presupunem că $C_1 = C_2$, $R_{12} = R_{21} < 0$ pentru această reţea. Presupunem că circuit a fost iniţializat prin încărcarea condensatoarelor (cu $\mathbf{o}_1^0$, $\mathbf{o}_2^0$, nu neapărat egale). Reţeaua îşi caută unul din cele două puncte de echilibru (fig. 2.28).

Simularea reţelei se face pe calculator prin ecuaţiile date.

4. Exemplu de învăţare hebbiană

În figura 2.29, avem intrarea:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

şi ponderile iniţiale:

$$\mathbf{w}^1 = \begin{bmatrix} 1 & -1 & 0 & 0,5 \end{bmatrix}^t.$$

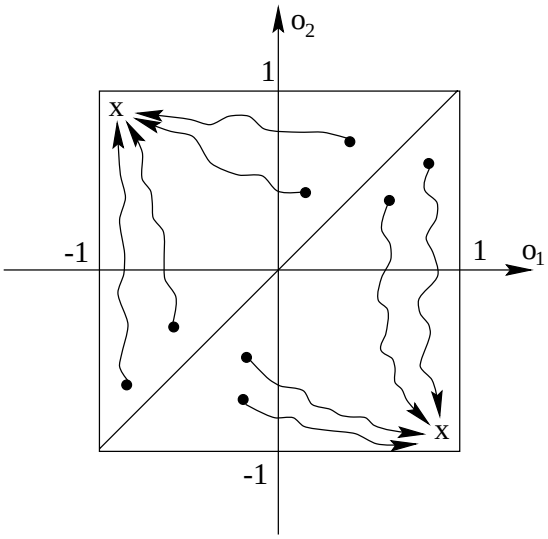


Figura 2.28: Modul în care reţeaua îşi găseşte cele două puncte de echilibru.

Presupunem că această reţea este instruită folosind mulţimea de vectori de intrare:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 1,5 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ -0,5 \\ -2 \\ -1,5 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1,5 \end{bmatrix} .$$

Considerăm $c = 1$. Presupunem pentru început că avem neuroni bipolari binari: $f(net) = \text{sgn}(net)$.

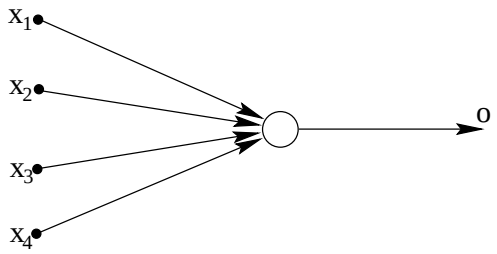


Figura 2.29: Reţea neurală pentru învăţarea hebbiană.

Pasul 1. Se aplică $\mathbf{x}_1$.

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0,5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1,5 \\ 0 \end{bmatrix} = 3$$

$$\mathbf{w}^2 = \mathbf{w}^1 + \text{sgn}(net^1)\mathbf{x}_1 = \mathbf{w}^1 + \mathbf{x}_1$$

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0,5 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1,5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1,5 \\ 0,5 \end{bmatrix}.$$

Pasul 2. Se aplică $\mathbf{x}_2$.

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 2 & -3 & 1,5 & 0,5 \end{bmatrix} \begin{bmatrix} 1 \\ -0,5 \\ -2 \\ -1,5 \end{bmatrix} = -0,25$$

$$\mathbf{w}^3 = \mathbf{w}^2 + \text{sgn}(net^2)\mathbf{x}_2 = \mathbf{w}^2 - \mathbf{x}_2 = \begin{bmatrix} 1 \\ -2,5 \\ 3,5 \\ 2 \end{bmatrix}.$$

Pasul 3. Se aplică $\mathbf{x}_3$.

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = -3$$

$$\mathbf{w}^4 = \mathbf{w}^3 + \text{sgn}(net^3)\mathbf{x}_2 = \mathbf{w}^3 - \mathbf{x}_3 = \begin{bmatrix} 1 \\ -3,5 \\ 4,5 \\ 0,5 \end{bmatrix}.$$

Să presupunem acum că funcţia de activare este continuă şi bipolară. Luăm $\lambda = 1$ şi pornim din nou de la $\mathbf{w}^1$.

Pasul 1. Se aplică $\mathbf{x}^1$.

$$f(net^1) = 0,905 \quad \mathbf{w}^1 = \begin{bmatrix} 1,905 \\ -2,81 \\ 1,357 \\ 0,5 \end{bmatrix}$$

Pasul 2. Se aplică $\mathbf{x}^2$.

$$f(net^2) = -0,077 \quad \mathbf{w}^2 = \begin{bmatrix} 1,828 \\ -2,772 \\ 1,512 \\ 0,616 \end{bmatrix}$$

Pasul 3. Se aplică $\mathbf{x}^3$.

$$f(net^3) = -0,932 \quad \mathbf{w}^3 = \begin{bmatrix} 1,828 \\ -3,7 \\ 2,44 \\ -0,783 \end{bmatrix}.$$

Se observă că pentru funcţia de activare continuă se obţin în general rezultate în aceeaşi direcţie, valorile fiind însă mai diferenţiate.

5. Exemplu de învăţare prin regula perceptronului

Considerăm următorii vectori de intrare:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1,5 \\ -0,5 \\ -1 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0,5 \\ -1 \end{bmatrix} \quad \mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0,5 \end{bmatrix}$$

şi $c = 0,1$. Răspunsurile dorite de către supervizor pentru $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ sunt -1, -1 şi, respectiv, 1.

Pasul 1. Se aplică $\mathbf{x}_1$.

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0,5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2,5.$$

Deoarece $-1 \neq \text{sgn}(2,5)$, are loc o corecţie a ponderilor:

$$\mathbf{w}^2 = \mathbf{w}^1 + 0,1(-1-1)\mathbf{x}_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0,5 \end{bmatrix} - 0,2 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0,8 \\ -0,6 \\ 0 \\ 0,7 \end{bmatrix}.$$

Pasul 2. Se aplică $\mathbf{x}_2$.

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 0 & 1,5 & -0,5 & -1 \end{bmatrix} \begin{bmatrix} 0,8 \\ -0,6 \\ 0 \\ 0,7 \end{bmatrix} = -1,6.$$

Deoarece $-1 = \text{sgn}(-1,6)$, nu se aplică nici o corecţie.

Pasul 3. Se aplică $\mathbf{x}_3$.

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = \begin{bmatrix} -1 & 1 & 0,5 & -1 \end{bmatrix} \begin{bmatrix} 0,8 \\ -0,6 \\ 0 \\ 0,7 \end{bmatrix} = -2,1.$$

Deoarece $1 \neq \text{sgn}(-2,1)$, se aplică o corecţie:

$$\mathbf{w}^4 = \mathbf{w}^3 + 0,1(1+1)3 = \begin{bmatrix} 0,6 \\ -0,4 \\ 0,1 \\ 0,5 \end{bmatrix}.$$

Nu este o simplă coincidenţă faptul că ultima componentă a vectorilor $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ este invariabilă. Învăţarea prin regula perceptronului necesită ca o componentă a vectorului de intrare să fie fixată (nu neapărat la valoarea -1).

Dacă reciclăm din nou vectorii $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ la intrare, erorile vor fi mai mici, deci reţeaua a "învăţat". De exemplu, dacă după ce am aplicat o dată $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ mai aplicăm o dată $\mathbf{x}_1$, obţinem $net^4 = 0,9$, deci răspunsul este mai bun decât $net^1 = 2,5$.

Un contraexemplu important:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad d_1 = 1 \quad d_2 = 1 \ .$$

Trebuie ca $w_1 + w_2 > 0$ şi $w_1 + w_2 < 0$, ceea ce este imposibil. Fie acum:

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad d_1 = 1 \quad d_2 = 1 \ .$$

În acest caz trebuie ca $w_1 + w_2 > w_3$ şi $w_1 + w_2 < -w_3$. Acest sistem are soluţii, evident, de exemplu $w_1 = 1$, $w_2 = 1$, $w_3 = -3$. Ce am făcut de fapt? Deoarece este evident că $\mathbf{x}_1$ şi $\mathbf{x}_2$ nu pot fi învăţaţi în mod supervizat, am considerat vectorii extinşi $\mathbf{y}_1$ şi $\mathbf{y}_2$ care se obţin din $\mathbf{x}_1$ şi $\mathbf{x}_2$ prin adăugarea unei componente constante, -1. Am văzut că pentru $\mathbf{y}_1$ şi $\mathbf{y}_2$ există $\mathbf{w}_1$ şi $\mathbf{w}_2$ astfel încât perceptronul să înveţe corect. Puteţi acum folosi regula perceptronului pentru a învăţa $\mathbf{y}_1$ şi $\mathbf{y}_2$. Încercaţi să explicaţi din punct de vedere geometric care este diferenţa dintre a învăţa $\mathbf{x}_1$ şi $\mathbf{x}_2$ faţă de a învăţa $\mathbf{y}_1$ şi $\mathbf{y}_2$.

6. Exemplu de învăţare prin regula delta

Folosim aceiaşi vectori ca în exemplul precedent:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1,5 \\ -0,5 \\ -1 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0,5 \\ -1 \end{bmatrix} \quad \mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0,5 \end{bmatrix}$$

La fel, răspunsurile dorite sunt -1, -1, 1, iar $c = 0,2$.

Se demonstrează mai târziu că:

$$f'(net) = \frac{1}{2}(1 - o^2)$$

dacă f este funcţia de activare bipolară continuă. Considerăm $\lambda = 1$.

Pasul 1. Se aplică $\mathbf{x}_1$.

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = 2,5$$

$$\mathbf{o}^1 = f(net^1) = 0,848$$

$$f'(net^1) = \frac{1}{2}[1 - (\mathbf{o}^1)^2] = 0,14$$

$$\mathbf{w}^2 = c(-1 - \mathbf{o}^1)f'(net^1)\mathbf{x}_1 + \mathbf{w}^1 = \begin{bmatrix} 0,974 \\ -0,948 \\ 0 \\ 0,526 \end{bmatrix}.$$

Pasul 2. Se aplică $\mathbf{x}_2$.

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = -1,948$$
$$\mathbf{o}^2 = f(net^2) = -0,75$$
$$f'(net^2) = \frac{1}{2}[1 - (\mathbf{o}^2)^2] = 0,218$$

$$\mathbf{w}^3 = c(-1 - \mathbf{o}^2)f'(net^2)\mathbf{x}_2 + \mathbf{w}^2 = \begin{bmatrix} 0,974 \\ -0,956 \\ 0,002 \\ 0,531 \end{bmatrix}.$$

Pasul 3. Se aplică $\mathbf{x}_3$.

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = -2,46$$
$$\mathbf{o}^3 = f(net^3) = -0,842$$
$$f'(net^3) = \frac{1}{2}[1 - (\mathbf{o}^3)^2] = 0,145$$

$$\mathbf{w}^4 = c(-1 - \mathbf{o}^3)f'(net^3)\mathbf{x}_3 + \mathbf{w}^3 = \begin{bmatrix} 0,974 \\ -0,929 \\ 0,016 \\ 0,505 \end{bmatrix}.$$

Metoda necesită de obicei valori mici pentru c.

2.6 Exerciţii

1. Folosiţi neuronul McCulloch-Pitts pentru a elabora reţele care implementează următoarele funcţii logice:
   (a) $o^{k+1} = x_1^k o_2^k o_3^{'k}$, unde $x_3^{'k}$ este complementul lui x_3^k. Se va folosi un neuron.
   (b) $o^{k+2} = x_1^{'k} x_2^k x_3^{'k}$. Se vor folosi doi neuroni în cascadă.
   (c) $o^{k+2} = x_1^k x_2^k$. Se vor folosi doi neuroni în cascadă.

2. Arătaţi că reţelele neurale din figura 2.30 sunt echivalente (implementează aceeaşi funcţie). Se presupune că:

$$f(net) = \begin{cases} 0 & net \leq 0 \\ 1 & net > 0 \end{cases}.$$

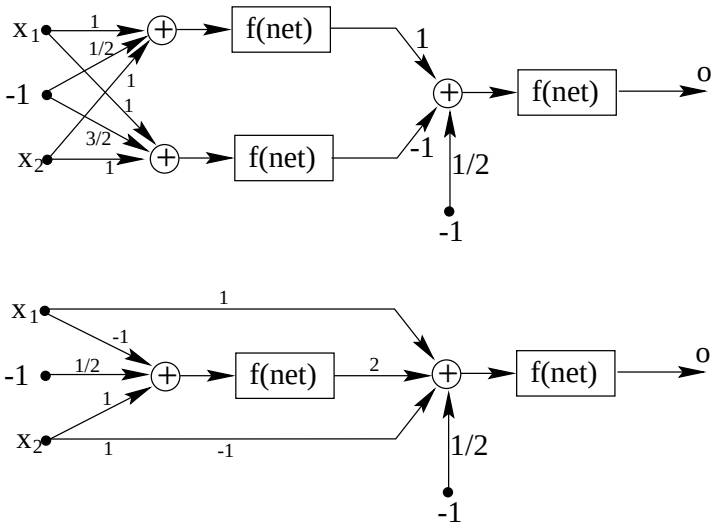


Figura 2.30: Exemplu de reţele neurale echivalente.

3. Reţeaua din figura 2.31 este un convertor A/D şi poate fi utilizată pentru codificarea unei valori continue x într-un cod binar unipolar pe 4 biţi $o_3\ o_2\ o_1\ o_0$). Analizaţi reţeaua şi găsiţi pentru ce valori ale lui x se obţin codificările $(0\ 0\ 0\ 0), \ldots, (1\ 1\ 1\ 1)$. Presupuneţi $-1 \leq x \leq 16$ şi că

$$f(net) = \begin{cases} 0 & net \leq 0 \\ 1 & net > 0 \end{cases} .$$

4. Reţeaua din figura 2.32 foloseşte neuroni cu funcţie de activare bipolară continuă ($\lambda = 1$). S-a măsurat că $o_1 = 0,28$; $o_2 = -0,73$. Găsiţi vectorul de intrare $\mathbf{x} = [x_1\ x_2]^t$ care a fost aplicat.

5. Reţeaua din figura 2.33 cu funcţie de activare bipolară continuă este proiectată pentru a atribui vectorilor $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ clusterele 1 sau 2. Numărul cluster-ului este identic cu numărul neuronului care produce cel mai mare răspuns. Determinaţi din ce clustere fac parte vectorii:

$$\mathbf{x}_1 = \begin{bmatrix} 0,866 \\ 0,5 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} -0,985 \\ -0,174 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 0,342 \\ -0,94 \end{bmatrix} .$$

6. (C) Reţeaua din figura 2.34 foloseşte neuroni cu funcţie de activare bipolară continuă ($\lambda = 5$) şi implementează o aplicaţie de la planul (x_1x_2) la segmentul $|o_1| < 1$. Simulaţi funcţionarea reţelei şi tabulaţi funcţia $o_1(x_1, x_2)$ pentru $|x_1| < 2,5$ şi $|x_2| < 2,5$.

7. Învăţare hebbiană pentru un neuron care este instruit cu:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

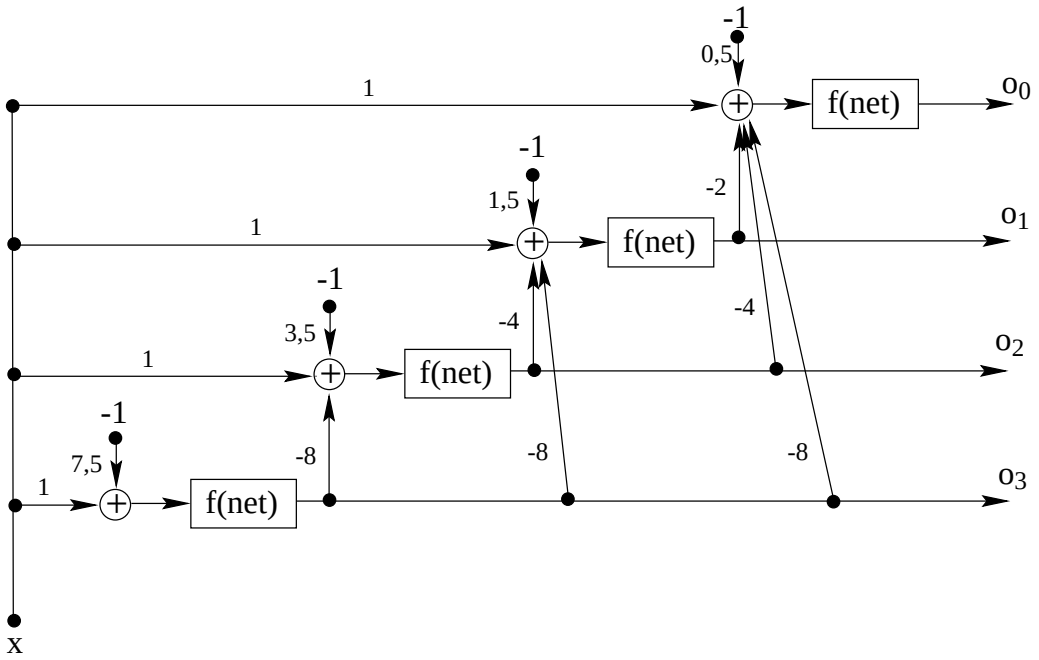


Figura 2.31: Convertor A/D.

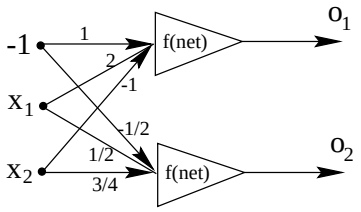


Figura 2.32: Reţea neurală care foloseşte neuroni cu funcţie de activare bipolară continuă.

şi $c = 1$. Găsiţi ponderile finale pentru cazul când funcţia de activare este

(a) bipolară binară

(b) bipolară continuă, $\lambda = 1$.

8. (C) Implementaţi învăţarea unui neuron prin regula perceptronului folosind:

$$\mathbf{w}^1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \left(\mathbf{x}_1 = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \quad \mathbf{d}_1 = -1 \right) \quad \left(\mathbf{x}_2 = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}, \quad \mathbf{d}_2 = 1 \right)$$

Repetaţi secvenţa de instruire $(\mathbf{x}_1, \mathbf{d}_1)$, $(\mathbf{x}_2, \mathbf{d}_2)$ până când se obţin răspunsurile corecte. Listaţi valorile net^k obţinute.

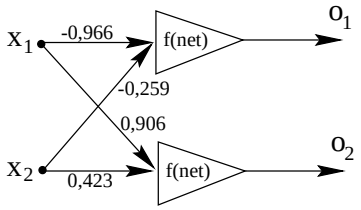


Figura 2.33: Reţea neurală care implementează un clasificator.

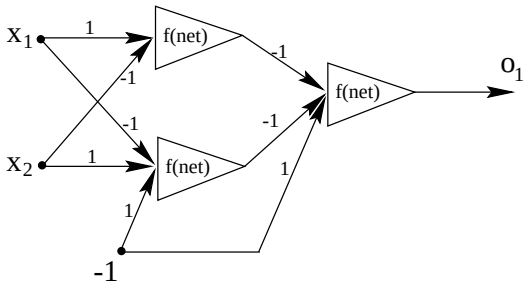


Figura 2.34: Reţea neurală care implementează o aplicaţie de la un plan la o dreaptă.

9. (C) Instruiţi prin regula delta un neuron folosind $\lambda = 1$, $c = 0,25$ şi

$$\mathbf{w}^1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \left(\mathbf{x}_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{d}_1 = -1 \right) \quad \left(\mathbf{x}_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, \quad \mathbf{d}_2 = 1 \right)$$

   Se va folosi $f'(net) = \frac{1}{2}(1 - o^2)$.

   Repetaţi secvenţa de instruire $(\mathbf{x}_1, \mathbf{d}_1)$, $(\mathbf{x}_2, \mathbf{d}_2)$ şi studiaţi comportarea.

10. (C) Folosiţi aceleaşi date ca şi în problema precedentă, aplicând regula Windrow-Hoff.

    Repetaţi secvenţa de instruire.

11. (C) Elaboraţi un program pentru analiza reţelelor feedforward cu două straturi. Parametrii de intrare sunt:

    - tipul funcţiei de activare
    - λ, dacă este cazul
    - mărimea reţelei
    - vectorii de intrare pentru testarea reţelei

12. (C) Implementaţi algoritmii de învăţare pentru un neuron cu cel mult şase intrări. Parametrii de intrare sunt:

- regula (Hebb, perceptron, delta, Windrow-Hoff)
- funcţia de activare
- λ (daca este cazul)
- datele pentru instruire
- numărul de paşi pentru instruire (se pot repeta datele).

Capitolul 3

Clasificatori pe bază de perceptroni monostrat

În acest capitol vom formula fundamentele reţelelor neurale instruibile, folosite în procese de decizie. Funcţia principală a unui sistem de decizie este luarea deciziilor în ceea ce priveşte clasa din care face parte pattern-ul de intrare considerat.

Arhitecturile discutate în acest capitol sunt de tip feedforward, neuronii fiind dispuşi pe un singur strat.

3.1 Clasificare

Una dintre cele mai importante categorii de probleme care poate fi efectuată de o reţea neurală este clasificarea pattern-urilor. Un *pattern* este descrierea cantitativă a unui obiect, eveniment sau fenomen. Clasificarea se poate referi atât la pattern-uri spaţiale cât şi temporale (ex.: imagini video de vapoare, hărţi meteo, amprente, caractere, semnale acustice, electrocardiograme etc.).

Scopul *clasificării pattern-urilor* este de a atribui un obiect fizic, eveniment sau fenomen unei clase specificate în prealabil. Un sistem de clasificare arată ca în figura 3.1.

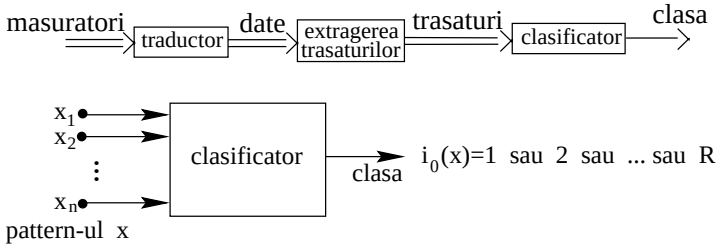
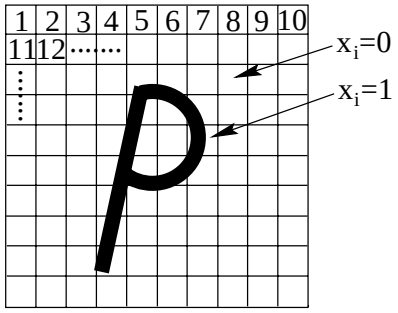


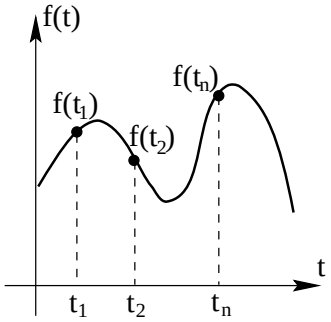
Figura 3.1: Sistem de clasificare.

Trăsăturile sunt date comprimate obţinute prin reducerea dimensionalităţii.



de unde putem extrage vectorul:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



de unde putem extrage vectorul:

$$\mathbf{x} = \begin{bmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ f(t_n) \end{bmatrix}$$

Figura 3.2: Exemple de codificare.

De exemplu, din imaginile recepţionate de pe satelitul LANDSAT, sunt necesare doar câteva din componentele spectrale.

Reţelele neurale pot fi utilizate atât pentru clasificare cât şi pentru extragerea trăsăturilor.

În continuare, vom reprezenta componentele de intrare ale unui clasificator ca un vector $\mathbf{x}$ (fig. 3.2). Clasificarea este obţinută prin implementarea unei funcţii de decizie i_0, unde $i_0(\mathbf{x}) \in \{1, 2 \ldots R\}$.

Atunci când pattern-urile de intrare nu sunt identice cu pattern-urile folosite pentru instruirea clasificatorului, procesul de clasificare se numeşte *recunoaştere*.

Clasificarea poate fi descrisă de multe ori în termeni geometrici. Orice pattern poate fi reprezentat printr-un punct în spaţiul euclidian n-dimensional E^n numit spaţiul pattern-urilor. Un clasificator aplică mulţimi de puncte din E^n într-unul din numerele $1, 2 \ldots R$. Mulţimile care conţin pattern-urile din clasele $1, 2, \ldots R$ le vom nota cu $\mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_R$, respectiv. În figura 3.3, $n = 2$, $R = 4$ şi $i_0 = j$ pentru orice $\mathbf{x} \in \mathcal{H}_j$, $j = 1, 2, 3, 4$.

Regiunile $\mathcal{H}_i$ sunt *regiuni de decizie* separate între ele prin *suprafeţe de decizie*. Presupunem că pattern-urile de pe suprafeţele de decizie nu aparţin nici unei clase. În spaţiul n-dimensional, suprafeţele de decizie sunt hipersuprafeţe $(n-1)$ dimensionale.

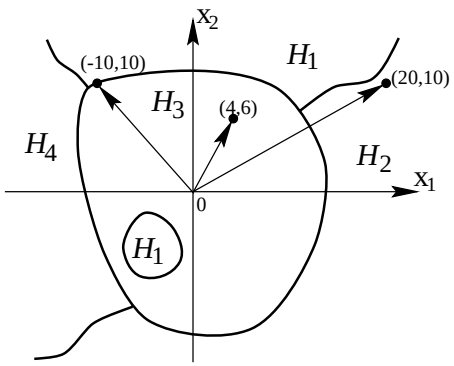


Figura 3.3: Regiuni de decizie.

3.2 Funcţii discriminant

Presupunem că avem o mulţime finită de pattern-uri n-dimensionale $\mathbf{x}_1$, $\mathbf{x}_2, \ldots,$ $\mathbf{x}_P$ şi cunoaştem clasificarea dorită pentru fiecare dintre ele. Presupunem că avem funcţiile discriminant g_1, g_2, ..., g_R astfel încât $\mathbf{x}$ face parte din clasa i dacă şi numai dacă

$$g_i(\mathbf{x}) > g_j(\mathbf{x}), \text{pentru } j = 1, 2, \ldots, R \text{ şi } i \neq j.$$

Valorile $g_i(\mathbf{x})$ şi $g_j(\mathbf{x})$ sunt scalare. Conform relaţiei de mai sus, în regiunea $\mathcal{H}_i$, $g_i(\mathbf{x})$ este maxim (fig. 3.4).

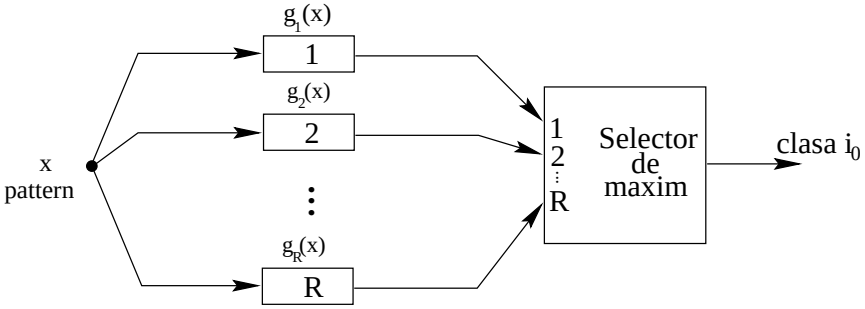


Figura 3.4: Clasificator.

Dacă $\mathbf{x}$ se află pe suprafaţa de decizie dintre $\mathcal{H}_i$ şi $\mathcal{H}_j$, atunci

$$g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0.$$

Pentru cazul când $R = 2$, clasificatorul se numeşte *dihotomizator*. Acest cuvânt provine din alăturarea cuvintelor greceşti *dicha* (în două) şi *tomia* (tăietură). În cazul dihotomizatorului, regula de clasificare devine:

$$g(\mathbf{x}) > 0\text{: clasa 1}$$
$$g(\mathbf{x}) < 0\text{: clasa 2}$$

unde $g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$.

Funcţia

$$i_0 = \text{sgn}[g(\mathbf{x})] = \begin{cases} -1 & g(\mathbf{x}) < 0 \\ \text{nedefinit} & g(\mathbf{x}) = 0 \\ +1 & g(\mathbf{x}) > 0 \end{cases}$$

implementează un dihotomizator (fig. 3.5). Problema găsirii funcţiilor discriminant este esenţială. Ele pot fi liniare sau neliniare.

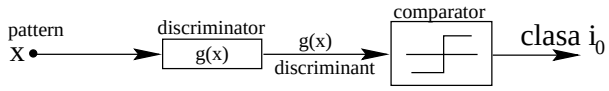


Figura 3.5: Dihotomizator.

Ne vom concentra asupra clasificatorilor ale căror funcţii discriminant se obţin prin învăţare iterativă, cu ajutorul pattern-urilor de instruire. Presupunem că:

1. Mulţimea pattern-urilor de instruire şi clasificarea lor sunt cunoscute, deci învăţarea este supervizată.

2. Funcţiile discriminant sunt liniare şi doar coeficienţii lor sunt ajustaţi pe parcursul procesului de învăţare.

3.3 Clasificatori liniari

Vom discuta în detaliu funcţiile discriminant liniare. În acest caz, suprafeţele de decizie sunt hiperplane. Considerăm iniţial că $R = 2$. Centrii P_1, P_2 ale clusterelor care formează cele două clase sunt vectorii $\mathbf{x}_1$ şi $\mathbf{x}_2$, respectiv (fig. 3.6).

Punctele prototip P_1 şi P_2 pot fi interpretate ca centrii de greutate ale clusterelor respective. Vom prefera ca hiperplanul de decizie să conţină mijlocul segmentului care conectează P_1 şi P_2 şi să fie perpendicular pe vectorul $\mathbf{x}_1 - \mathbf{x}_2$. Ecuaţia hiperplanului de decizie este atunci:

$$g(\mathbf{x}) = (\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}\left(\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2\right).$$

În general, ecuaţia lui $g(\mathbf{x})$ este de forma:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} = 0,$$

$$\text{sau: } \mathbf{w}^t \mathbf{x} + w_{n+1} = 0$$

$$\text{sau: } \begin{bmatrix} \mathbf{w} \\ w_{n+1} \end{bmatrix}^t \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = 0,$$

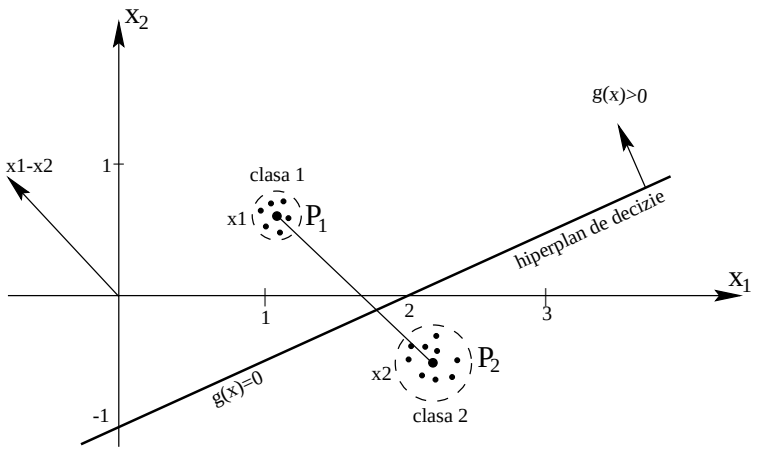


Figura 3.6: Separarea a două clase printr-un hiperplan de decizie.

$$\text{unde: } \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

este vectorul ponderilor. În cazul precedent,

$$\mathbf{w} = \mathbf{x}_1 - \mathbf{x}_2, \quad w_{n+1} = \frac{1}{2}\left(\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2\right).$$

Deci, dacă punctele prototip P_1 şi P_2 sunt cunoscute, se deduce şi $g(\mathbf{x})$.

Funcţiile discriminant liniare pot fi folosite şi dacă avem mai mult decât două clase. Dacă avem R clase, două câte două separabile, atunci vor fi până la $\frac{R(R-1)}{2}$ hiperplane de decizie.

Să presupunem că folosim criteriul distanţei minime pentru a clasifica pattern-uri şi că avem R clase. Fiecare clasă este reprezentată de un punct prototip $P_1, P_2, \ldots, P_R$ reprezentat de un vector $\mathbf{x}_1, \ldots, \mathbf{x}_R$. Distanţa euclidiană dintre un pattern de intrare $\mathbf{x}$ şi pattern-ul prototip $\mathbf{x}_i$ este

$$\|\mathbf{x} - \mathbf{x}_i\| = \sqrt{(\mathbf{x} - \mathbf{x}_i)^t(\mathbf{x} - \mathbf{x}_i)}.$$

Clasificatorul calculează distanţele dintre $\mathbf{x}$ şi $\mathbf{x}_i$, $i = 1, 2, \ldots, R$, alegând apoi clasa pentru care distanţa este minimă. Avem:

$$\|\mathbf{x} - \mathbf{x}_i\|^2 = \mathbf{x}^t\mathbf{x} - 2\mathbf{x}_i^t\mathbf{x} + \mathbf{x}_i^t\mathbf{x}_i, \quad i = 1, 2, \ldots, R.$$

De fapt, căutăm $\mathbf{x}_i$ pentru care se obţine $\max(\mathbf{x}_i^t\mathbf{x} - \frac{1}{2}\mathbf{x}_i^t\mathbf{x}_i)$. Putem lua atunci:

$$g_i(\mathbf{x}) = \mathbf{x}_i^t\mathbf{x} - \frac{1}{2}\mathbf{x}_i^t\mathbf{x}_i, \quad i = 1, 2, \ldots, R.$$

Funcţia discriminant are forma

$$g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + w_{i,n+1}, \quad i = 1, 2, \ldots, R,$$

$$\text{unde: } \mathbf{w}_i = \mathbf{x}_i, \quad w_{i,n+1} = -\frac{1}{2}\mathbf{x}_i^t \mathbf{x}_i, \quad i = 1, 2, \ldots, R.$$

Clasificatorii care folosesc criteriul distanţei minime pot fi consideraţi *clasificatori liniari* şi ei se numesc uneori *maşini liniare*. Un clasificator liniar arată ca în figura 3.7.

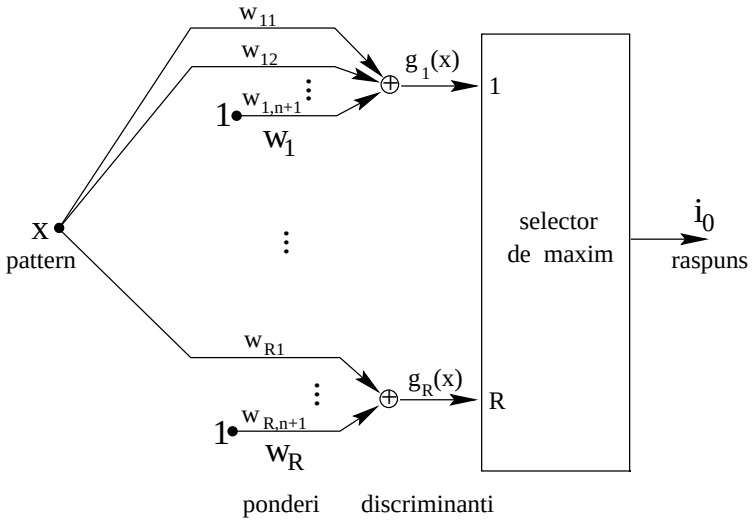


Figura 3.7: Clasificator liniar.

Să observăm că suprafaţa de decizie S_{ij} dintre regiunile contigue (alăturate, vecine) H_i şi H_j este un hiperplan:

$$g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0$$

$$\text{sau: } \mathbf{w}_i^t \mathbf{x} + w_{i,n+1} - \mathbf{w}_j^t \mathbf{x} - w_{j,n+1} = 0.$$

Dacă definim:

$$y = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

obţinem:

$$g_i(\mathbf{y}) = \mathbf{w}_i^t \mathbf{y}.$$

Să presupunem că avem o mulţime $\mathcal{H}$ de pattern-uri pe care o împărţim în submulţimile $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_R$. Dacă o maşină liniară poate clasifica pattern-urile din $\mathcal{H}_i$ ca făcând parte din clasa i, pentru $i = 1, 2, \ldots, R$, atunci mulţimile $\mathcal{H}_i$ sunt *liniar separabile*. Mai formal, dacă există R funcţii liniare pentru care:

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \text{pt. } \mathbf{x} \in \mathcal{H}_i, \quad i = 1, 2, \ldots, R; \; j = 1, 2, \ldots, R; \; i \neq j,$$

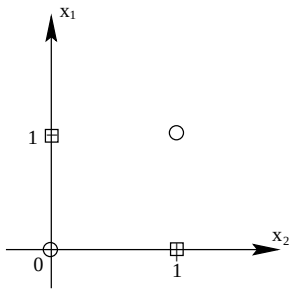


Figura 3.8: Pattern-uri neseparabile liniar.

atunci mulţimile $\mathcal{H}_i$ sunt liniar separabile.

În figura 3.8 avem un exemplu de pattern-uri liniar neseparabile care pot fi modelate prin funcţia de paritate unipolară

$$\mathrm{XOR}(x_1, x_2) = x_1 \oplus x_2,$$

unde $\oplus$ este operatorul OR exclusiv.

3.4 Utilizarea perceptronului discret în instruirea unui dihotomizator liniar

Am văzut în secţiunea precedentă cum se pot determina coeficienţii discriminanţilor liniari (ponderile) din informaţia *a priori* asupra pattern-urilor şi a apartenenţei lor la diferite clase.

În această secţiune, ponderile se vor obţine printr-un proces de învăţare. Pattern-urile de instruire $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P$ sunt prezentate maşinii liniare (fig. 3.9) împreună cu răspunsurile corecte. Această secvenţă de pattern-uri se numeşte *secvenţă de instruire*. Răspunsul este dat de către supervizor, iar clasificatorul îşi modifică parametrii pe baza învăţării iterative supervizate.

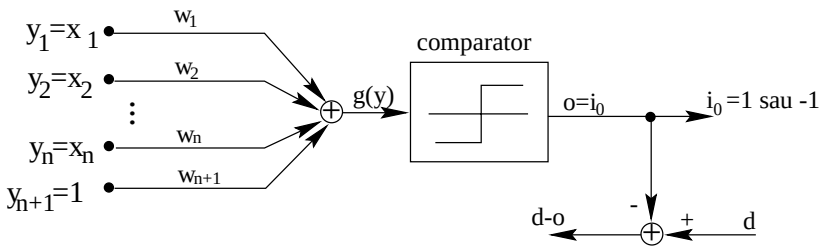


Figura 3.9: Maşina liniară.

Fie $R = 2$, deci analizăm cazul unui dihotomizator (foloseşte două clase).

Suprafaţa de decizie în spaţiul E^n are ecuaţia:

$$\mathbf{w}^t\mathbf{x} + w_{n+1} = 0$$

$$\text{sau: } \mathbf{w}^t\mathbf{y} = 0, \quad \mathbf{y} \in \mathrm{E}^{n+1}.$$

Această ecuaţie descrie un hiperplan care trece prin origine şi este perpendicular pe vectorul $\mathbf{y}$, iar $\mathbf{y}$ este îndreptat către semispaţiul pentru care $\mathbf{w}^t\mathbf{y} > 0$, deci către semispaţiul unde se află clasa 1.

În figura 3.10 este dat un exemplu în care pattern-urile $\mathbf{y}_1$, $\mathbf{y}_2$,...,$\mathbf{y}_5$ sunt deplasate de la origine, în paralel, de-a lungul hiperplanelor lor de decizie. Este umbrită regiunea din spaţiul ponderilor care satisface condiţia de separabilitate liniară a claselor 1 şi 2.

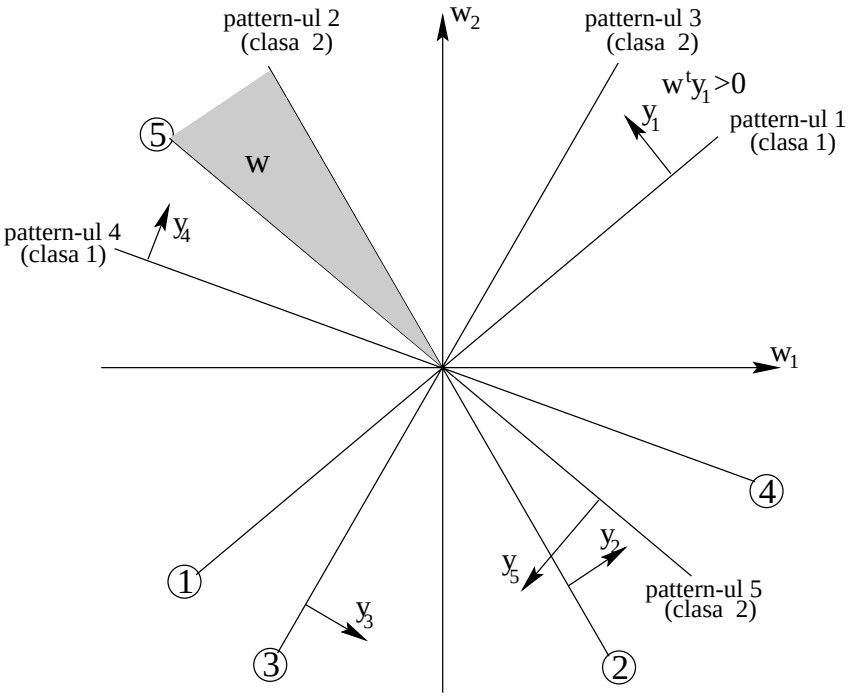


Figura 3.10: Reprezentarea unor suprafeţe de decizie în spaţiul ponderilor.

Să presupunem acum că ponderile iniţiale sunt $\mathbf{w}^1$. Discutăm despre pattern-ul $\mathbf{y}_1$ din clasa 1. Conform figurii 3.11, $\mathbf{y}_1$ nu este clasificat corect, deoarece $\mathbf{w}^{1t}\mathbf{y}_1 < 0$.

Ajustăm vectorul $\mathbf{w}$ astfel încât $\mathbf{w}^t\mathbf{y}_1$ să crească cât mai rapid, deci în direcţia gradientului. Gradientul este:

$$\nabla_{\mathbf{w}}(\mathbf{w}^t\mathbf{y}_1) = \mathbf{y}_1.$$

Deci, atunci când pattern-ul $\mathbf{y}_1$ este clasificat greşit, vom ajusta ponderile astfel:

$$\mathbf{w}' = \mathbf{w}^1 + c\mathbf{y}_1$$

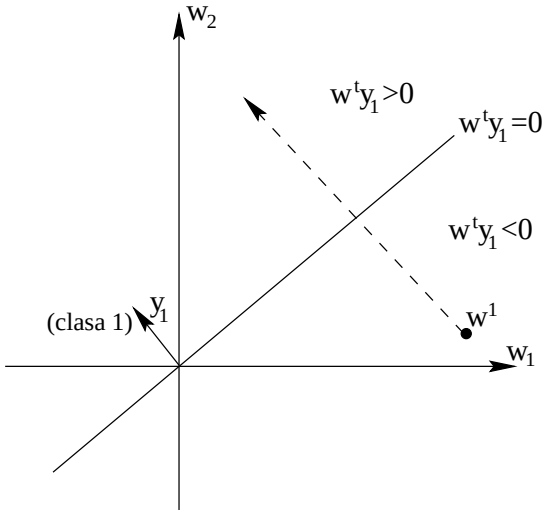


Figura 3.11: Ajustarea ponderilor pentru un pattern din clasa 1.

unde $c > 0$ este *incrementul de corecţie*. Repetând ajustarea de câteva ori, indiferent de valoarea lui c, ajungem cu ponderile în regiunea corectă, unde $\mathbf{w}^t\mathbf{y}_1 > 0$.

Să presupunem că ponderile iniţiale sunt $\mathbf{w}^1$. De data aceasta, discutăm despre pattern-ul $\mathbf{y}_1$ din clasa 2 (fig. 3.12).

În acest caz, $\mathbf{y}_1$ nu este clasificat corect, deoarece $\mathbf{w}^{1t}\mathbf{y}_1 > 0$. Descreştem cât mai rapid pe $\mathbf{w}^t\mathbf{y}_1$, deci în direcţia gradientului negativ:

$$\mathbf{w}' = \mathbf{w}^1 - c\mathbf{y}_1.$$

Procedura de instruire supervizată este, aşadar:

$$\mathbf{w}' = \mathbf{w}^1 \pm c\mathbf{y}_1$$

unde "+" este valabil când se clasifică greşit un pattern din clasa 1, iar "-" este valabil când se clasifică greşit un pattern din clasa 2. Atunci când pattern-ul este clasificat corect, nu se face nici o corecţie. Procedura se repetă iterativ pentru toate pattern-urile de instruire. Vom vedea că această formulă este aceeaşi cu regula de învăţare a perceptronului discret (secţiunea 3.8).

Să rezumăm clasificarea pattern-urilor liniar separabile aparţinând la doar două clase. Se caută un vector $\mathbf{w}$ astfel încât:

$$\begin{array}{ll} \mathbf{w}^t\mathbf{y} > 0 & \text{pentru } \mathbf{x} \in \mathcal{H}_1 \\ \mathbf{w}^t\mathbf{y} < 0 & \text{pentru } \mathbf{x} \in \mathcal{H}_2 \end{array}.$$

Perceptronul se instruieşte pornind cu un vector al ponderilor care este ales arbitrar şi cu un increment de corecţie ales, de asemenea, arbitrar. Se ajunge la un vector $\mathbf{w}^* = \mathbf{w}^{k_0}$, k_0 fiind numărul iteraţiilor necesare pentru instruire. În continuare,

$$\mathbf{w}^* = \mathbf{w}^{k_0} = \mathbf{w}^{k_0+1} = \ldots.$$

Se poate demonstra următoarea teoremă:

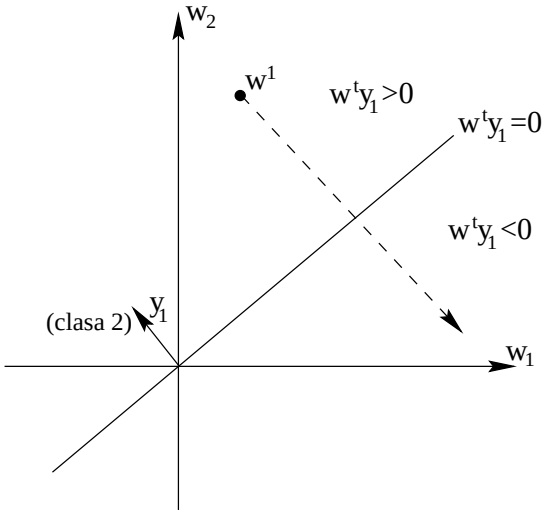


Figura 3.12: Ajustarea ponderilor pentru un pattern din clasa 2.

Teorema de convergenţă a perceptronului. Un clasificator pentru două clase liniar separabile de pattern-uri este întotdeauna instruibil într-un număr finit de iteraţii.

Vom discuta mai detaliat această teoremă în secţiunea următoare.

Valoarea lui k_0 depinde de incrementul de corecţie şi de secvenţa de pattern-uri utilizate pentru instruire.

Regula $\mathbf{w}' = \mathbf{w}^1 \pm c\mathbf{y}_1$ devine în cazul instruirii percepronului discret:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \frac{c}{2}(d^k - o^k)\mathbf{y}^k$$

unde k este numărul iteraţiei, o^k este ieşirea actuală iar d^k reprezintă ieşirea dorită pentru vectorul $\mathbf{y}^k$ aplicat la intrare. Dacă $d^k = o^k$, nu se ajustează $\mathbf{w}^k$. Dacă $d^k = 1$ şi $o^k = -1$, atunci:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + c\mathbf{y}^k.$$

Dacă $d^k = -1$ şi $o^k = 1$, atunci:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - c\mathbf{y}^k.$$

Algoritmul de instruire a unui perceptron discret care poate fi folosit apoi ca dihotomizator liniar este următorul:

Se dau P perechi $(\mathbf{x}_1, d_1)$, $(\mathbf{x}_2, d_2)$,...,$(\mathbf{x}_P, d_P)$, unde $\mathbf{x}_i$ este un vector de n elemente, $i = 1, 2, \ldots, P$. Se defineşte vectorul $\mathbf{y}_1 = \left[\begin{array}{cc} \mathbf{x}_i & 1 \end{array} \right]^t$, $i = 1, 2, \ldots, P$.

1. Se alege $c > 0$.

2. *Se iniţializează vectorul* $\mathbf{w}$ *de* $n+1$ *elemente cu valori aleatoare şi mici. Se fac următoarele iniţializări:* $k \leftarrow 1$, $i \leftarrow 1$ *şi* $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_i$, $d \leftarrow d_i$, $o \leftarrow sgn(\mathbf{w}^t\mathbf{y})$.
4. $\mathbf{w} \leftarrow \mathbf{w} + \frac{1}{2}c(d-o)\mathbf{y}$.
5. *Se calculează eroarea pătratică cumulată:*

$$E \leftarrow E + \frac{1}{2}(d-o)^2.$$

6. `if` $i < P$ `then` $i \leftarrow i+1$, $k \leftarrow k+1$, `go to` *3.*
7. `if` $E > 0$ `then` $E \leftarrow 0$, $i \leftarrow 1$, `go to` *3.*
   `else` *"s-a terminat instruirea",* `write` $\mathbf{w}$, k.

3.5 Utilizarea perceptronului continuu în instruirea unui dihotomizator liniar

Utilizarea perceptronului continuu are două avantaje:

1. Un control mai fin asupra procedurii de instruire.
2. Se lucrează cu caracteristici diferenţiabile ale comparatorului, ceea ce permite calcularea gradientului erorii.

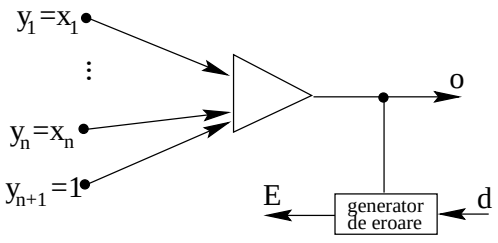


Figura 3.13: Modelul folosit la instruirea unui perceptron continuu folosit ca dihotomizator liniar.

Problema modificării ponderilor se rezolvă foarte elegant prin minimizarea funcţiei care măsoară eroarea de clasificare, iar această minimizare se realizează prin folosirea gradientului (fig. 3.13). Se porneşte de la un vector $\mathbf{w}$ arbitrar şi se calculează $\nabla E(\mathbf{w})$, gradientul funcţiei eroare curente. Următoarea valoare a lui $\mathbf{w}$ este obţinută îndreptându-ne în direcţia gradientului negativ de-a lungul suprafeţei multidimensionale a erorii. Direcţia gradientului negativ este cea a descreşterii cea mai rapide. Luăm deci:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta\nabla E(\mathbf{w}^k)$$

unde η este *constanta de învăţare*, o valoare pozitivă.

Eroarea de clasificare care trebuie minimizată este:

$$\begin{aligned} E_k &= \frac{1}{2}(d^k - o^k)^2 \\ &= \frac{1}{2}\left[d^k - f(\mathbf{w}^{kt}\mathbf{y}^k)\right]^2. \end{aligned}$$

Obţinem (pentru simplificarea scrierii, omitem temporar indicii k):

$$\nabla E(\mathbf{w}) = -(d-o)f'(net)\mathbf{y},$$

unde $net = \mathbf{w}^t\mathbf{y}$, sau:

$$\frac{\partial E}{\partial w_i} = -(d-o)f'(net)y_i, \quad i = 1, 2, \ldots, n+1$$

care este, de fapt, regula delta de învăţare pentru perceptronul continuu.

Dacă funcţia de activare este

$$f(net) = \frac{2}{1+e^{-net}} - 1,$$

în care luăm $\lambda = 1$, atunci

$$f'(net) = \frac{2e^{-net}}{(1+e^{-net})^2}.$$

Aplicăm identitatea

$$\frac{2e^{-net}}{(1+e^{-net})^2} = \frac{1}{2}(1-o^2)$$

care se verifică luând $o = f(net)$ şi obţinem:

$$\nabla E(\mathbf{w}) = -\frac{1}{2}(d-o)(1-o^2)\mathbf{y}.$$

Rezultă:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \frac{1}{2}\eta(d^k - o^k)(1-o^{k^2})\mathbf{y}^k.$$

Această regulă corectează ponderile în aceeaşi direcţie ca şi regula

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \frac{c}{2}(d^k - o^k)\mathbf{y}^k$$

din cazul perceptronului discret. Diferă mărimea vectorului de corecţie a ponderilor. Ambele reguli implică adunarea sau scăderea unei fracţiuni din $\mathbf{y}$. Diferenţa esenţială este prezenţa factorului moderator $1 - o^{k^2}$. Acest factor este $0 < 1 - o^{k^2} < 1$. Atunci când răspunsurile sunt eronate, avem următoarea proprietate: pentru *net* apropiat de 0, factorul de corecţie este mai mare decât dacă *net* este departe de 0. Aceasta contribuie la stabilitatea procesului de învăţare.

O altă diferenţă semnificativă este că algoritmul pe baza perceptronului discret duce întotdeauna la o soluţie (dacă soluţia există). Acest lucru nu este garantat în cazul perceptronului continuu pentru orice valoare a lui η. Deoarece valorile ponderilor se modifică de la un pas la altul, nu mergem de fapt exact în direcţia lui $-\nabla E(\mathbf{w})$. Pentru η mic, putem considera că mergem în această direcţie. Pentru o valoare a lui η suficient de mică, se ajunge la un $\mathbf{w}$ care îl minimizează pe $E(\mathbf{w})$. Nu avem o teoremă a perceptronului de tip continuu care să garanteze convergenţa către o soluţie.

Algoritmul de instruire a unui perceptron continuu pentru utilizarea sa ca dihotomizator liniar este următorul:

Se dau P perechi $(\mathbf{x}_1, d_1)$, $(\mathbf{x}_2, d_2)$,...,$(\mathbf{x}_P, d_P)$, unde $\mathbf{x}_i$ este un vector de n elemente, $i = 1, 2, \ldots, P$. Se defineşte vectorul $\mathbf{y}_1 = \begin{bmatrix} \mathbf{x}_i & 1 \end{bmatrix}^t$, $i = 1, 2, \ldots, P$.

1. *Se aleg $\eta > 0$, $\lambda = 1$ şi $E_{max} > 0$.*
2. *Se iniţializează vectorul $\mathbf{w}$ de $n+1$ elemente cu valori aleatoare şi mici. Se fac următoarele iniţializări: $k \leftarrow 1$, $i \leftarrow 1$ şi $E \leftarrow 0$.*
3. *$\mathbf{y} \leftarrow \mathbf{y}_i$, $d \leftarrow d_i$, $o \leftarrow f(\mathbf{w}^t\mathbf{y})$, care este funcţia de activare continuă.*
4. *$\mathbf{w} \leftarrow \mathbf{w} + \frac{1}{2}\eta(d-o)(1-o^2)\mathbf{y}$.*
5. *Se calculează eroarea pătratică cumulată:*

$$E \leftarrow E + \frac{1}{2}(d-o)^2.$$

6. `if` $i < P$ `then` $i \leftarrow i+1$, $k \leftarrow k+1$, `go to` *3.*
7. `if` $E \geq E_{max}$ `then` $E \leftarrow 0$, $i \leftarrow 1$, `go to` *3.*
   `else` *"s-a terminat instruirea",* `write` $\mathbf{w}$, k, E.

În acest capitol am presupus până acum că $y_{n+1} = +1$. Fie acum un dihotomizator bazat pe perceptron, în care $y_{n+1} = -1$ (fig. 3.14).

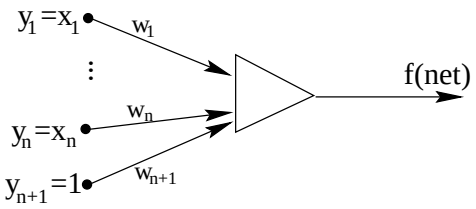


Figura 3.14: Perceptron instruit ca dihotomizator.

Ca urmare a procesului de instruire, obţinem acelaşi rezultat, doar semnul lui w_{n+1} se inversează. Putem lua deci $y_{n+1} = +1$ sau $y_{n+1} = -1$.

Să presupunem, în cazul dihotomizatorului bazat pe perceptronul continuu, că luăm $y_{n+1} = -1$. Atunci:

$$net = \mathbf{w}^t\mathbf{x} - w_{n+1}.$$

Luăm $T = w_{n+1}$. Avem:

$$\begin{cases} f(net) > 0 & \text{pentru } \mathbf{w}^t\mathbf{x} > T \\ f(net) < 0 & \text{pentru } \mathbf{w}^t\mathbf{x} < T \end{cases}.$$

Funcţia de activare f cu argumentul $\mathbf{w}^t\mathbf{x}$ este cea din figura 3.15.

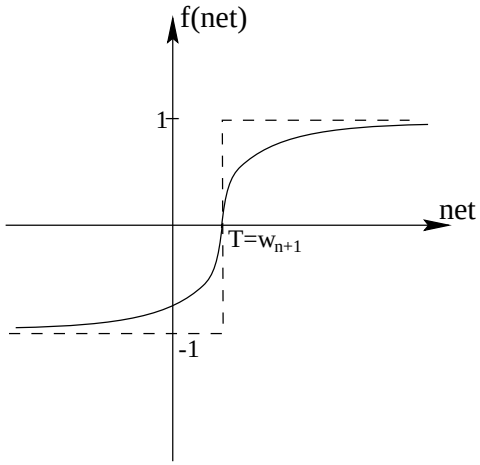


Figura 3.15: Funcţia de activare folosită de un perceptron instruit ca dihotomizator.

Procesul de instruire acceptă pentru y_{n+1} orice valoare constantă. Luând însă $y_{n+1} = -1$, w_{n+1} devine egal cu pragul T al neuronului. De acum încolo, vom considera $y_{n+1} = -1$ şi $w_{n+1} = T$, în cazul în care nu specificăm altfel.

3.6 Mai multe despre teorema de convergenţă a perceptronului

Vom discuta în această secţiune mai în detaliu acest important rezultat teoretic. Să ne reamintim enunţul teoremei:

> **T1**. Un clasificator pentru două clase liniar separabile de pattern-uri este întotdeauna instruibil într-un număr finit de iteraţii.

Iteraţiile se referă la relaţia:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \frac{c}{2}(d^k - o^k)\mathbf{y}^k,$$

adică, dacă:

$$d^k \neq o^k \Rightarrow \mathbf{w}^{k+1} = \mathbf{w}^k \pm c\mathbf{y}^k.$$

O altă variantă a acestei teoreme este cea formulată de Rosenblatt (1957) şi de Minsky şi Papert (1969):

> **T2**. Dacă pattern-urile de intrare sunt din $\{-1, 0, +1\}^n$, atunci un perceptron învaţă într-un număr finit de paşi să clasifice corect pattern-urile, presupunând că acestea sunt liniar separabile.

Trebuie să notăm că:

1. Ponderile iniţiale pot fi numere reale oarecare.
2. Nu îl putem determina pe k_0 deoarece depinde de $\mathbf{w}^*$, unde $\mathbf{w}^*$ este un vector soluţie pentru ponderi. Deci ar trebui să îl cunoaştem *a priori* pe $\mathbf{w}^*$ pentru a determina numărul de iteraţii. De asemenea, k_0 depinde de $\mathbf{w}^1$.

Ne punem acum problema ce se întâmplă când aplicăm algoritmul perceptronului asupra a două clase de pattern-uri care nu sunt liniar separabile. Se poate demonstra următoarea teoremă:

> **T3**. Pentru o secvenţă finită de pattern-uri de instruire cu n componente din $\{-1, 0, +1\}^n$, algoritmul de instruire a perceptronului va executa, într-un număr finit de paşi, exact una dintre următoarele două variante:
>
> 1. Produce un vector al ponderilor pentru care perceptronul clasifică corect toate pattern-urile de instruire, dacă şi numai dacă aceste pattern-uri sunt liniar separabile.
> 2. Părăseşte şi revizitează un vector al ponderilor, dacă şi numai dacă pattern-urile de instruire nu sunt liniar separabile.

Se pot face acum următoarele observaţii:

1. Rezultă astfel o procedură de testare a liniar separabilităţii.
2. Nu se cunoaşte o limită superioară pentru cât ar dura acest test.

În final, se poate demonstra următoarea teoremă:

> **T4**. Dacă pattern-urile de instruire au componente reale, algoritmul de instruire a perceptronului poate să nu conveargă către o soluţie, chiar dacă pattern-urile sunt liniar separabile (constanta de învăţare fiind oarecare).

Cu alte cuvinte, atunci când pattern-urile de instruire au componente reale oarecare, alegerea unei constante de învăţare adecvate devine critică şi această constantă nu mai poate fi un număr real pozitiv oarecare.

3.7 Instruirea unui clasificator liniar prin utilizarea unei reţele monostrat de perceptroni

Fie $R > 2$ clase două câte două liniar separabile. Există deci R funcţii discriminant liniare astfel încât:

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \text{pentru } i, j = 1, 2, \ldots, R, \quad i \neq j \quad \forall \mathbf{x} \in H_i.$$

Definim vectorul ponderilor

$$\mathbf{w}_q = \left[\begin{array}{cccc} w_{q1} & w_{q2} & \ldots & w_{q,n+1} \end{array} \right]^t.$$

Să presupunem că un pattern $\mathbf{y}$ este prezentat unui clasificator liniar. Dacă $\mathbf{w}_i^t \mathbf{y} > \mathbf{w}_j^t \mathbf{y}$, $j = 1, 2, \ldots, R$, $i \neq j$, clasificarea este corectă şi

$$\begin{aligned} \mathbf{w}_1{}' &= \mathbf{w}_1, \\ \mathbf{w}_2{}' &= \mathbf{w}_2, \\ &\vdots \\ \mathbf{w}_R{}' &= \mathbf{w}_R \end{aligned}$$

sunt valorile ajustate ale vectorilor ponderilor. Dacă avem $\mathbf{w}_i^t \mathbf{y} \leq \mathbf{w}_m^t \mathbf{y}$ pentru un $m \neq i$, atunci:

$$\begin{aligned} \mathbf{w}_i{}' &= \mathbf{w}_i + c\mathbf{y} \\ \mathbf{w}_m{}' &= \mathbf{w}_m - c\mathbf{y} \\ \mathbf{w}_k{}' &= \mathbf{w}_k \qquad \text{pentru } k = 1, 2, \ldots, R, \quad k \neq i, m \end{aligned}$$

sau:

$$\begin{aligned} w_{ij}{}' &= w_{ij} + cy_j \quad \text{pentru } j = 1, 2, \ldots, n+1 \\ w_{mj}{}' &= w_{mj} + cy_j \quad \text{pentru } j = 1, 2, \ldots, n+1 \\ w_{kj}{}' &= w_{kj} \qquad \text{pentru } k = 1, 2, \ldots, R, \quad k \neq i, m, \quad j = 1, 2, \ldots, n+1 \end{aligned}$$

Acest clasificator are la ieşire un selector de maxim.

Să încercăm acum să înlocuim selectorul de maxim dintr-un clasificator liniar cu R perceptroni discreţi (fig. 3.16).

Dacă $\mathbf{x} \in H_i$, trebuie ca $o_i = 1$ şi $o_j = -1$ pentru $j = 1, 2, \ldots, R$ şi $j \neq i$. Eliminăm astfel selectorul de maxim. Ajustarea ponderilor se face astfel:

$$w_i^{k+1} = w_i^k + \frac{c^k}{2} \left(d_i^k - o_i^k \right) \mathbf{y}^k, \quad \text{pentru } i = 1, 2, \ldots, R.$$

Aceasta este regula de instruire a clasificatorului bazat pe o reţea de perceptroni discreţi. În relaţia de mai sus, d_i şi o_i se referă la al i-lea perceptron.

Teorema perceptronului se poate generalizaa şi demonstra şi pentru cazul cu $R > 2$ clase două câte două liniar separabile.

Algoritmul de instruire a unei reţele monostrat de perceptroni folosită ca şi clasificator liniar este următorul:

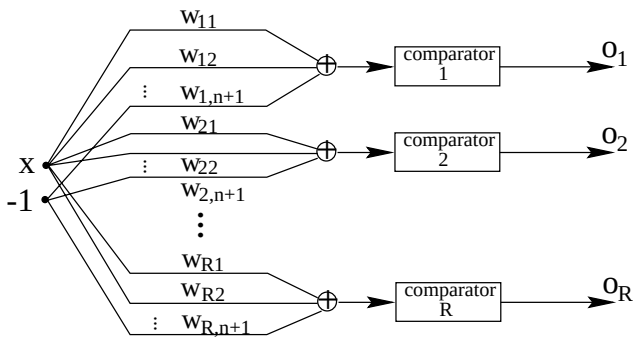


Figura 3.16: Clasificator liniar cu R clase.

Se dau perechile $(\mathbf{x}_1, \mathbf{d}_1)$,... ,$(\mathbf{x}_P, \mathbf{d}_P)$, *unde* $\mathbf{x}_i$ *este un vector de* R *elemente şi* $i = 1, 2, \ldots, P$. *Se definesc vectorii* $\mathbf{y}_i = \begin{bmatrix} \mathbf{x}_i & -1 \end{bmatrix}^t$, $i = 1, 2, \ldots, P$.

1. *Se alege* $c > 0$, *ca în cazul perceptronului discret.*
2. *Se iniţializează matricea* $\mathbf{W}$ *de* $R \times (n+1)$ *cu valori aleatoare şi mici. Se iniţializează* $k \leftarrow 1$, $p \leftarrow 1$ *şi* $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_p$, $\mathbf{d} \leftarrow \mathbf{d}_P$, $o_i \leftarrow sgn(\mathbf{w}_i^t \mathbf{y})$, *pentru* $i = 1, 2, \ldots, R$, *unde* $\mathbf{w}_i$ *este linia* i *din matricea* $\mathbf{W}$.
4. $\mathbf{w}_i \leftarrow \mathbf{w}_i + \frac{1}{2} c(d_i - o_i)\mathbf{y}$, *pentru* $i = 1, 2, \ldots, R$, *unde* i *reprezintă elementul al* i*-lea din vectorul* $\mathbf{d}$.
5. $E \leftarrow \frac{1}{2}(d_i - o_i)^2 + E$, $i = 1, 2, \ldots, R$.
6. `if` $i < P$ `then` $i \leftarrow i + 1$, $p \leftarrow p + 1$, `go to` *3.*
7. `if` $E \geq E_{max}$ `then` $E \leftarrow 0$, $i \leftarrow 1$, `go to` *3.*
   `else` *"s-a terminat instruirea",* `write` $\mathbf{w}$, k, E.

Putem folosi şi o reţea de perceptroni de tip continuu (fig. 3.17).
În acest caz, ajustarea ponderilor se face astfel:

$$w_i^{k+1} = w_i^k + \frac{1}{2}\eta(d_i^k - o_i^k)\left(1 - o_i^{k2}\right)\mathbf{y}^k, \quad \text{pentru } i = 1, 2, \ldots, R.$$

Asupra acestei formule şi a reţelei de perceptroni de tip continuu vom discuta pe larg în capitolul următor.

3.8 Exemple

Exemplu de construire a unui clasificator liniar

Avem următoarele pattern-uri (3.18):

$$\mathbf{x}_1 = \begin{bmatrix} 10 \\ 2 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 2 \\ -5 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -5 \\ 5 \end{bmatrix}$$

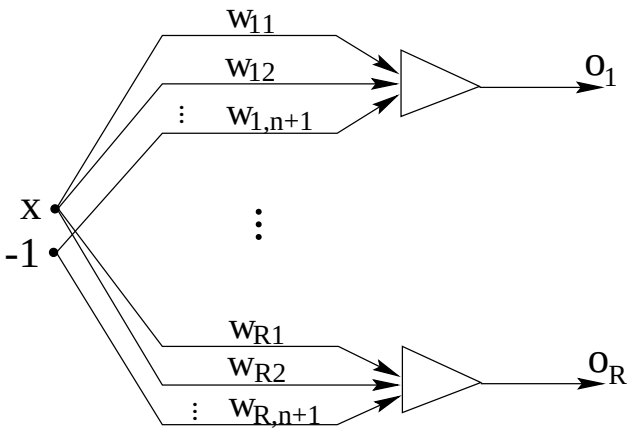


Figura 3.17: Reţea de perceptroni de tip continuu.

cu $n = 2$ şi $R = 3$.

Folosim formula

$$\begin{aligned} &\mathbf{w}_i = \mathbf{x}_i \\ &w_{i,n+1} = -\tfrac{1}{2}\mathbf{x}_i^t\mathbf{x}_i, \quad i = 1, 2, \ldots, R. \end{aligned}$$

şi obţinem ponderile:

$$\mathbf{w}_1 = \begin{bmatrix} 10 \\ 2 \\ -52 \end{bmatrix}, \quad \mathbf{w}_2 = \begin{bmatrix} 2 \\ -5 \\ -14,5 \end{bmatrix}, \quad \mathbf{w}_3 = \begin{bmatrix} -5 \\ 5 \\ -25 \end{bmatrix}.$$

Din formula:

$$g_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} + w_{i,i+1}, \quad i = 1, 2, \ldots, R$$

obţinem funcţiile discriminant

$$\begin{aligned} g_1(\mathbf{x}) &= 10x_1 + 2x_2 - 52 \\ g_2(\mathbf{x}) &= 2x_1 - x_2 - 14,5 \\ g_3(\mathbf{x}) &= -5x_1 + 5x_2 - 25. \end{aligned}$$

Din

$$\mathbf{w}_i^t\mathbf{x} + w_{i,n+1} - \mathbf{w}_j^t\mathbf{x} - w_{j,n+1} = 0$$

sau din $g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0$, obţinem:

$$\begin{aligned} S_{12}: \quad 8x_1 + 7x_2 - 37,5 &= 0 \\ S_{13}: \quad -15x_1 + 3x_2 + 27 &= 0 \\ S_{23}: \quad -7x_1 + 10x_2 - 10,5 &= 0. \end{aligned}$$

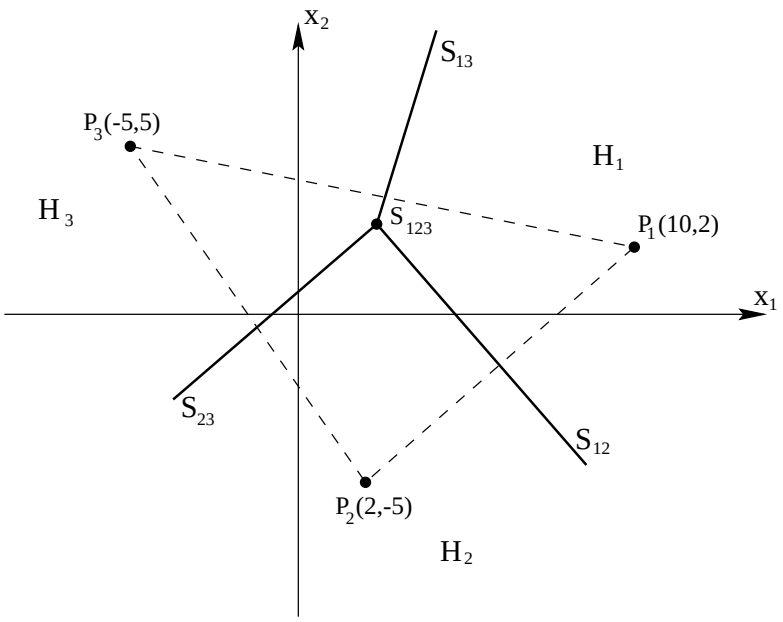


Figura 3.18: Suprafeţele de decizie pentru un clasificator liniar. $S_{123} = (2,337; 2,686)$.

Exemplu de instruire a unui dihotomizator cu ajutorul unui perceptron discret

Avem următoarele patru pattern-uri şi răspunsuri dorite pentru fiecare dintre ele:

$$\begin{array}{lllll} \text{Clasa 1:} & \mathbf{x}_1 = 1, & \mathbf{x}_3 = 3, & d_1 = d_3 = 1 \\ \text{Clasa 2:} & \mathbf{x}_2 = -0,5, & \mathbf{x}_4 = -2, & d_2 = d_4 = -1. \end{array}$$

Pattern-urile extinse sunt:

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{y}_2 = \begin{bmatrix} -0,5 \\ 1 \end{bmatrix}, \quad \mathbf{y}_3 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \quad \mathbf{y}_4 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}.$$

Fie ponderile iniţiale $\mathbf{w}^1 = \begin{bmatrix} -2,5 & 1,75 \end{bmatrix}^t$ alese arbitrar. Regula $\mathbf{w}' = \mathbf{w} \pm c\mathbf{y}$ devine în cazul perceptronului:

$$\Delta \mathbf{w}^k = \frac{c}{2}\left[d^k - \operatorname{sgn}(\mathbf{w}^{kt}\mathbf{y}^k)\right]\mathbf{y}^k.$$

Alegem $c = 1$.

Pasul 1. Aplicăm $\mathbf{y}_1$

$$o_1 = \operatorname{sgn}\left(\begin{bmatrix} -2,5 & 1,75 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = -1$$

$$d_1 - o_1 = 2$$

$$\mathbf{w}^2 = \mathbf{w}^1 + \mathbf{y}_1 = \begin{bmatrix} -1,5 \\ 2,75 \end{bmatrix}.$$

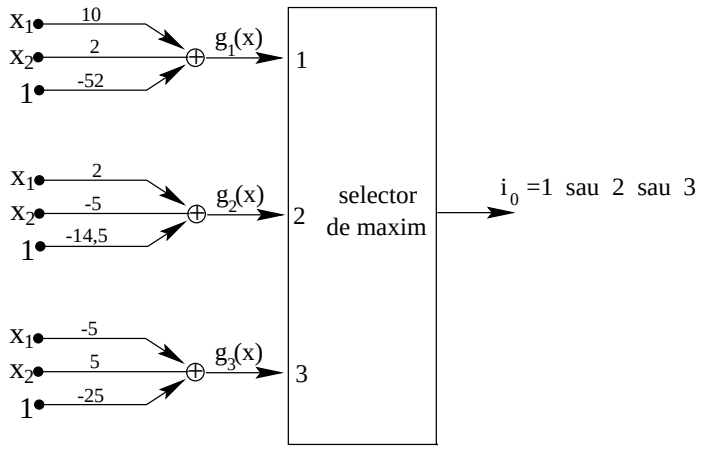


Figura 3.19: Clasificatorul liniar care implementeaza regiunile de decizie din figura 3.18.

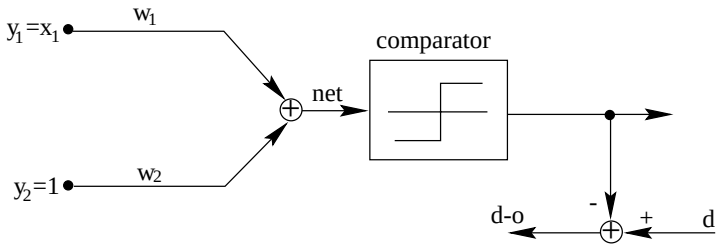


Figura 3.20: Perceptron discret folosit ca un clasificator instruibil.

Pasul 2. Aplicăm $\mathbf{y}_2$

$$o_2 = \text{sgn}\left(\begin{bmatrix} -1,5 & 2,75 \end{bmatrix} \begin{bmatrix} -0,5 \\ 1 \end{bmatrix} \right) = 1$$

$$d_2 - o_2 = -2$$

$$\mathbf{w}^3 = \mathbf{w}^2 + \mathbf{y}_2 = \begin{bmatrix} -1 \\ 1,75 \end{bmatrix}.$$

Pasul 3. Aplicăm $\mathbf{y}_3$

$$o_3 = -1$$

$$d_3 - o_3 = 2$$

$$\mathbf{w}^4 = \mathbf{w}^3 + \mathbf{y}_3 = \begin{bmatrix} 2 \\ 2,75 \end{bmatrix}.$$

Pasul 4. Aplicăm $\mathbf{y}_4$. După citirea lui $\mathbf{y}_4$, nu se modifică ponderile. Reciclăm secvenţa $\mathbf{y}_1$, ..., $\mathbf{y}_4$ deoarece nu ştim dacă perceptronul a reuşit să înveţe să clasifice corect.

Pasul 5. Aplicăm $\mathbf{y}_1$

$$\mathbf{w}^5 = \mathbf{w}^4.$$

Pasul 6. Aplicăm $\mathbf{y}_2$

$$\mathbf{w}^6 = \mathbf{w}^5.$$

Pasul 7. Aplicăm $\mathbf{y}_3$

$$\mathbf{w}^7 = \left[\begin{array}{cc} 2,5 & 1,75 \end{array} \right]^t.$$

Pasul 8. Aplicăm $\mathbf{y}_4$

$$\mathbf{w}^8 = \mathbf{w}^7.$$

Reciclăm din nou şi obţinem:

$$\mathbf{w}^{10} = \mathbf{w}^9 = \mathbf{w}^8 = \mathbf{w}^7$$

$$\mathbf{w}^{11} = \left[\begin{array}{cc} 3 & 0,75 \end{array} \right] \text{ care sunt ponderile finale.}$$

Exemplu de instruire a unui dihotomizator liniar cu ajutorul unui perceptron continuu

Reluăm exemplul din cazul discret. Valoarea erorii la pasul k este:

$$E_k = \frac{1}{2} \left\{ d^k - \left[\frac{2}{1 + e^{-\lambda net^k}} - 1 \right] \right\}^2,$$

unde $net^k = \mathbf{w}^{kt}\mathbf{y}$. Luând $\mathbf{y}^k = \mathbf{y}_k$, pentru $k = 1, 2, 3, 4$, obţinem:

$$E_1(\mathbf{w}) = \frac{1}{2} \left\{ 1 - \left[\frac{2}{1 + e^{-\lambda(w_1 + w_2)}} - 1 \right] \right\}^2.$$

Considerăm $\lambda = 1$ şi obţinem:

$$E_1(\mathbf{w}) = \frac{2}{\left[1 + e^{w_1 + w_2}\right]^2}.$$

La următorii paşi obţinem:

$$E_2(\mathbf{w}) = \frac{2}{\left[1 + e^{0,5w_1 - w_2}\right]^2}$$

$$E_3(\mathbf{w}) = \frac{2}{\left[1 + e^{3w_1 + w_2}\right]^2}$$

$$E_4(\mathbf{w}) = \frac{2}{\left[1 + \exp(2w_1 - w_2)\right]^2}.$$

Luăm $\eta = 0,5$ şi facem calculele conform algoritmului. Deoarece o nu poate fi niciodată ± 1, vom lua de obicei pentru d valorile -0,9 şi 0,9.

Exemplu de instruire a unui clasificator reprezentat ca reţea de perceptroni discreţi

Reluăm clasificatorul liniar ($R = 3$) construit deja pe baza distanţei minime. Sunt uşor de calculat valorile funcţiilor discriminant în $\mathbf{x}_1$, $\mathbf{x}_2$ şi $\mathbf{x}_3$:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
$g_1(\mathbf{x})$	52	-42	-92
$g_2(\mathbf{x})$	-4,5	14,5	-49,5
$g_3(\mathbf{x})$	-65	-60	25

Răspunsurile maxime se obţin, desigur, pe diagonală. Construim (nu prin instruire) reţeaua de perceptroni corespunzătoare acestui clasificator (fig. 3.21).

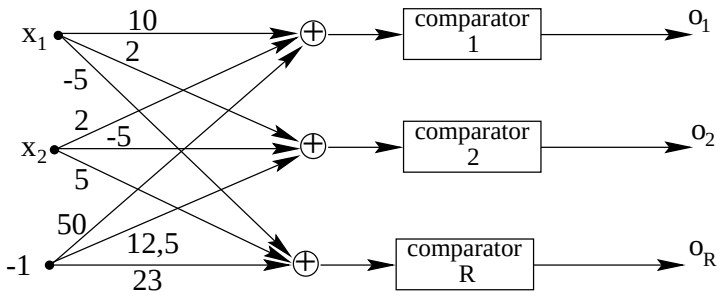


Figura 3.21: Reţea de perceptroni.

Ar trebui să avem $w_{1,3} = 52$, $w_{2,3} = 14,5$ şi $w_{3,3} = 25$ ca valori de prag. Putem scădea aceste praguri cu 2 fără ca să modificăm răspunsurile clasificatorului.

Vom construi acum direct clasificatorul prin instruirea reţelei de perceptroni. Alegem aleator vectorii iniţiali:

$$\mathbf{w}_1^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} \quad \mathbf{w}_2^1 = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix} \quad \mathbf{w}_3^1 = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}.$$

Fie $c = 1$. Cu "*" marcăm răspunsurile incorecte.

Pasul 1. $(\mathbf{y}_1)$

$$\begin{aligned} \operatorname{sgn}\begin{pmatrix} 1 & -2 & 0 \end{pmatrix}\begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} &= 1 \\ \operatorname{sgn}\begin{pmatrix} 0 & -1 & 2 \end{pmatrix}\begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} &= -1 \\ \operatorname{sgn}\begin{pmatrix} 1 & 3 & -1 \end{pmatrix}\begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} &= 1 \quad * \end{aligned}$$

$$\mathbf{w}_1^2 = \mathbf{w}_1^1, \quad \mathbf{w}_2^2 = \mathbf{w}_2^1, \quad \mathbf{w}_3^2 = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix} - \begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} -9 \\ 1 \\ 0 \end{bmatrix}.$$

Pasul 2. $(\mathbf{y}_2)$

$$\begin{aligned} \operatorname{sgn}\begin{pmatrix} 1 & -2 & 0 \end{pmatrix} \begin{bmatrix} 2 \\ -5 \\ -1 \end{bmatrix} &= 1 \quad * \\ \operatorname{sgn}\begin{pmatrix} 0 & -1 & 2 \end{pmatrix} \begin{bmatrix} 2 \\ -5 \\ -1 \end{bmatrix} &= 1 \\ \operatorname{sgn}\begin{pmatrix} -9 & 1 & 0 \end{pmatrix} \begin{bmatrix} 2 \\ -5 \\ -1 \end{bmatrix} &= -1 \quad * \end{aligned}$$

$$\mathbf{w}_1^3 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 \\ -5 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 1 \end{bmatrix}, \quad \mathbf{w}_2^3 = \mathbf{w}_2^2, \quad \mathbf{w}_3^3 = \mathbf{w}_3^2.$$

Pasul 3. $(\mathbf{y}_3)$

$$\begin{aligned} \operatorname{sgn}(\mathbf{w}_1^{3t}\mathbf{y}_3) &= 1 \quad * \\ \operatorname{sgn}(\mathbf{w}_2^{3t}\mathbf{y}_3) &= -1 \\ \operatorname{sgn}(\mathbf{w}_3^{3t}\mathbf{y}_3) &= 1 \end{aligned}$$

$$\mathbf{w}_1^4 = \begin{bmatrix} 4 \\ -2 \\ 2 \end{bmatrix}, \quad \mathbf{w}_2^4 = \mathbf{w}_2^3, \quad \mathbf{w}_3^4 = \mathbf{w}_3^3.$$

Se încheie primul ciclu, apoi reluăm secvenţa etc. Se modifică doar ponderile primului perceptron.

$$\mathbf{w}_1^5 = \mathbf{w}_1^4, \quad \mathbf{w}_1^6 = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}, \quad \mathbf{w}_1^7 = \begin{bmatrix} 7 \\ -2 \\ 4 \end{bmatrix}, \quad \mathbf{w}_1^8 = \mathbf{w}_1^7, \quad \mathbf{w}_1^9 = \begin{bmatrix} 5 \\ 3 \\ 5 \end{bmatrix}.$$

Obţinem o reţea de perceptroni pentru care:

$$\begin{aligned} o_1 &= \operatorname{sgn}(5x_1 + 3x_2 - 5) \\ o_2 &= \operatorname{sgn}(-x_2 - 2) \\ o_3 &= \operatorname{sgn}(-9x_1 + x_2). \end{aligned}$$

S-au produs regiuni de indecizie (în fig 3.22, sunt zonele umbrite). De exemplu, pentru Q avem $\mathbf{o} = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}^t$. Pe de altă parte, pattern-uri ca acest Q nu au fost utilizate la instruire.

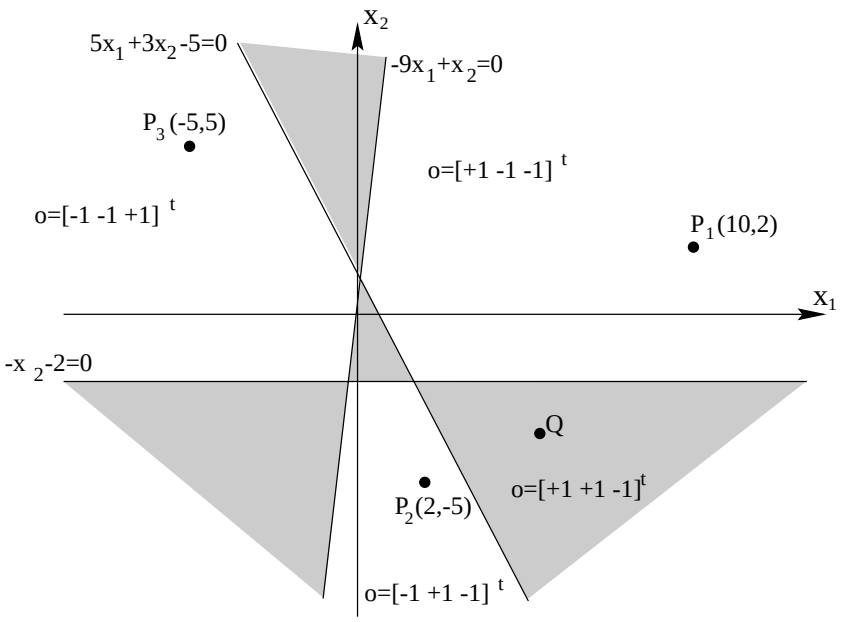


Figura 3.22: Regiuni de indecizie.

3.9 Exerciţii

1. (C) Implementaţi algoritmul de instruire a unui dihotomizator cu ajutorul unui perceptron discret luând $c = 1$, $\mathbf{w} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^t$ şi

   clasa 1, $\mathbf{x}$: $\begin{bmatrix} 0,8 & 0,5 & 0 \end{bmatrix}^t$ $\begin{bmatrix} 0,9 & 0,7 & 0,3 \end{bmatrix}^t$ $\begin{bmatrix} 1 & 0,8 & 0,5 \end{bmatrix}^t$
   clasa 2, $\mathbf{x}$: $\begin{bmatrix} 0 & 0,2 & 0,3 \end{bmatrix}^t$ $\begin{bmatrix} 0,2 & 0,1 & 1,3 \end{bmatrix}^t$ $\begin{bmatrix} 0,2 & 0,7 & 0,8 \end{bmatrix}^t$.

   Încercaţi şi alte valori pentru c.

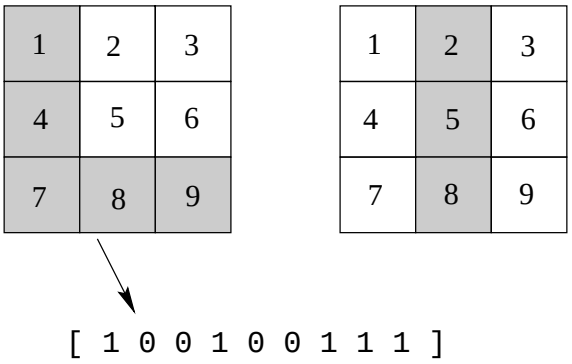


Figura 3.23: Caracterele tipărite L şi I folosite al instruirea unui dihotomizator.

2. (C) Implementaţi algoritmul de instruire a unui dihotomizator cu ajutorul unui perceptron continuu în condiţiile problemei precedente. Încercaţi diferite valori pentru constanta de învăţare.

3. Prin algoritmul de instruire a unui clasificator liniar cu trei clase reprezentat printr-o reţea de trei perceptroni discreţi, s-a ajuns ca în trei paşi să se termine instruirea. Se consideră $c = 1$.

   Pasul 1. ($\mathbf{x}_1$). Se ajustează $\mathbf{w}_1^1 = \mathbf{w}_2^1 = \mathbf{w}_3^1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^t$.

   Pasul 2. ($\mathbf{x}_2$). Se ajustează $\mathbf{w}_3^2$.

   Pasul 3. ($\mathbf{x}_3$). Se ajustează $\mathbf{w}_2^3$.

   În final,

   $$\mathbf{w}_1^4 = \begin{bmatrix} 1 & 3 & -1 \end{bmatrix}^t, \quad \mathbf{w}_2^4 = \begin{bmatrix} 5 & -1 & -2 \end{bmatrix}^t, \quad \mathbf{w}_3^4 = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}^t.$$

   Găsiţi pattern-urile $\mathbf{x}_1$, $\mathbf{x}_2$ şi $\mathbf{x}_3$.

4. (C) Implementaţi algoritmul de instruire a unui clasificator liniar reprezentat
   ca reţea de perceptroni discreţi folosind $c = 1$ şi următoarele pattern-uri de instruire:

   $$\begin{aligned} i_0 = 1 \text{ pentru } \mathbf{x}\text{:} &\quad \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^t, \quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^t \\ i_0 = 2 \text{ pentru } \mathbf{x}\text{:} &\quad \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^t, \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^t \\ i_0 = 3 \text{ pentru } \mathbf{x}\text{:} &\quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^t, \quad \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^t \\ i_0 = 4 \text{ pentru } \mathbf{x}\text{:} &\quad \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^t, \quad \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^t. \end{aligned}$$

   Reprezentaţi suprafeţele de decizie rezultate.

5. Proiectaţi şi instruiţi un clasificator pentru caracterele tipărite L şi I (fig. 3.23).

   Folosiţi un perceptron discret. Ce se întâmplă dacă literele sunt distorsionate?

   Rezultat: Apare indecizia. Ar trebui să instruim perceptronul şi cu aceste litere distorsionate.

Capitolul 4

Reţele neurale feedforward multistrat

Pentru pattern-uri de instruire care nu sunt liniar separabile, reţeaua neurală introdusă în capitolul 3 trebuie modificată. Modificarea se poate face astfel:

- fie prin utilizarea unor funcţii discriminant neliniare (de ex. liniare pe porţiuni);

- fie printr-o reţea multistrat.

Alegem ultima variantă. În acest caz, fiecare strat este compus dintr-o reţea care se bazează pe conceptul de funcţie discriminant liniară.

Reţelele multistrat pot implementa suprafeţe de decizie arbitrare. În completare la capitolul 3, se pot rezolva astfel multe alte aplicaţii: aproximarea funcţiilor, recunoaşterea caracterelor scrise, recunoaşterea vorbirii, sisteme expert, generarea traiectoriilor etc.

Reţelele neurale multistrat sunt în prezent cele mai răspândite arhitecturi.

În acest capitol, vom studia reţelele neurale multistrat instruibile.

4.1 Clasificarea pattern-urilor liniar neseparabile

Fie două mulţimi de pattern-uri $\mathcal{Y}_1$ şi $\mathcal{Y}_2$. Dacă nu există un vector $\mathbf{w}$ al ponderilor astfel încât:

$$\begin{array}{lll} \mathbf{y}^t\mathbf{w} > 0 & \text{pentru orice} & \mathbf{y} \in \mathcal{Y}_1 \\ \mathbf{y}^t\mathbf{w} < 0 & \text{pentru orice} & \mathbf{y} \in \mathcal{Y}_2 \end{array}$$

atunci $\mathcal{Y}_1$ şi $\mathcal{Y}_2$ sunt *liniar neseparabile*. Vom presupune că pattern-urile din $\mathcal{Y}_1$ şi $\mathcal{Y}_2$ sunt vectori măriţi cu o componentă.

Să presupunem pentru început că două mulţimi de pattern-uri $\mathcal{H}_1$ şi $\mathcal{H}_2$ trebuie clasificate în două categorii (fig. 4.1).

Această clasificare se poate implementa ca în figura 4.2.

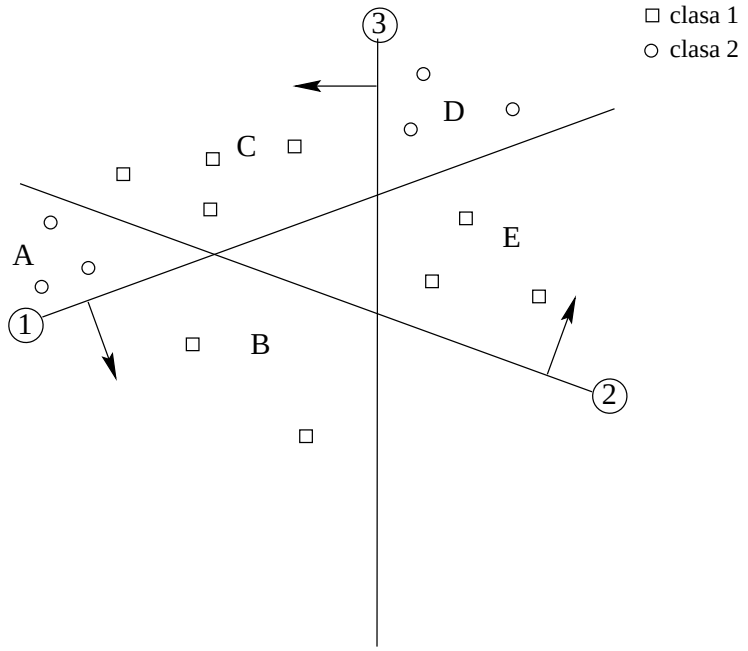


Figura 4.1: Două mulţimi de pattern-uri care trebuie clasificate în două categorii.

Fiecare dintre cele şapte compartimente (inclusiv A,...,E) sunt aplicate întrunul din vârfurile cubului, în spaţiul $o_1o_2o_3$ (spaţiul o). Se observă că în spaţiul imagine o, pattern-urile din cele două clase sunt uşor separabile printr-un plan, de exemplu prin planul $o_1 + o_2 + o_3 = 0$. Perceptronul cu intrările o_1, o_2, o_3 este un dihotomizator liniar:

$$o_4 = \begin{cases} \text{sgn}(o_1 + o_2 + o_3) > 0 & : \text{clasa 1} \\ \text{sgn}(o_1 + o_2 + o_3) < 0 & : \text{clasa 2} \end{cases} .$$

4.2 Regula de învăţare delta pentru o reţea de neuroni monostrat

Fie o reţea monostrat de perceptroni de tip continuu (fig. 4.3) în care dacă $y_J = -1$, atunci w_{kJ}, $k = 1, 2, \ldots, K$ sunt chiar valorile pragurilor celor K neuroni.

În această reţea, $\mathbf{o} = \Gamma[\mathbf{Wy}]$, unde:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_J \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_K \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1J} \\ w_{21} & \ldots & & \\ \vdots & & & \\ w_{K1} & \ldots & & w_{KJ} \end{bmatrix}$$

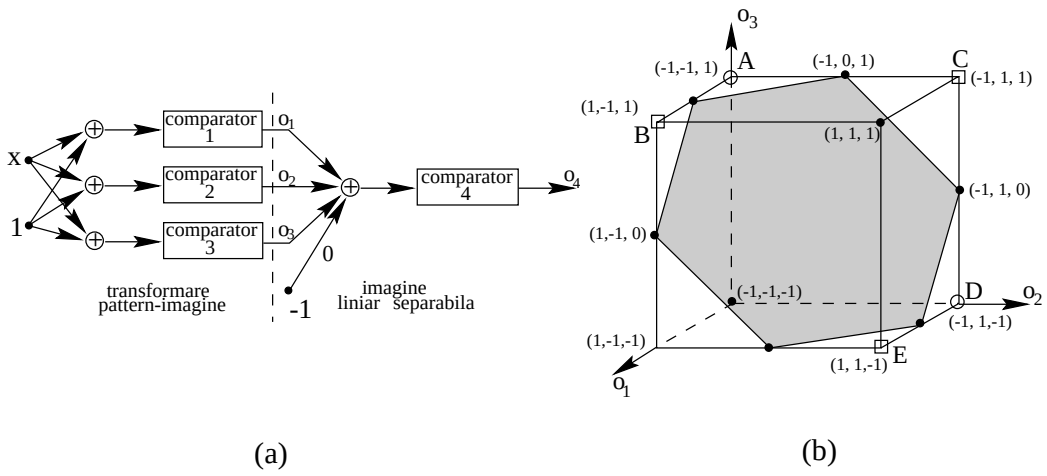


Figura 4.2: (a) Clasificator monostrat; (b) Reprezentarea pattern-urilor din figura 4.1 în spaţiul imagine o.

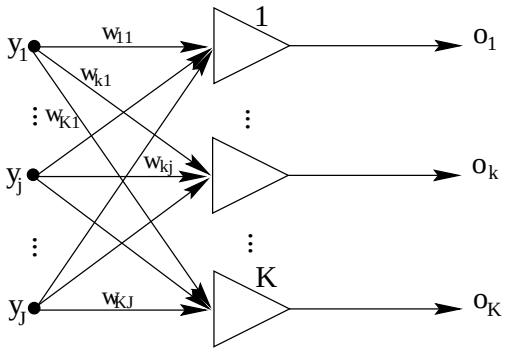


Figura 4.3: Reţea monostrat de perceptorni de tip continuu.

$$\Gamma[\cdot] = \begin{bmatrix} f(\cdot) & 0 & \dots & 0 \\ 0 & f(\cdot) & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & f(\cdot) \end{bmatrix}.$$

Prin definiţie,

$$\mathbf{d} = \begin{bmatrix} d_1 \\ \vdots \\ d_K \end{bmatrix}.$$

este răspunsul dorit. Generalizăm expresia erorii:

$$E_p = \frac{1}{2}\sum_{k=1}^{K}\left(d_{p_k} - o_{p_k}\right)^2 = \frac{1}{2}\|\mathbf{d}_p - \mathbf{o}_p\|^2$$

pentru un pattern p, $p = 1, 2, \ldots, P$, unde $\mathbf{d}_p$ este răspunsul dorit, iar $\mathbf{o}_p$ răspunsul reţelei pentru pattern-ul p:

$$\mathbf{d}_p = \begin{bmatrix} d_{p_1} \\ \vdots \\ d_{p_K} \end{bmatrix} \quad \mathbf{o}_p = \begin{bmatrix} o_{p_1} \\ \vdots \\ o_{p_K} \end{bmatrix}.$$

Pentru simplitate, să presupunem că

$$y_J = -1 \text{ şi } \mathbf{w}_{kJ} = T_k \text{ pentru } k = 1, 2, \ldots, K$$

unde T_k sunt praguri. Ca şi în cazul unui singur perceptron, calculăm

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}},$$

deci pe direcţia gradientului negativ al erorii, unde pentru simplitate am omis indicele lui E. Avem:

$$o_k = f(net_k); \quad net_k = \textstyle\sum_{j=1}^{J} w_{kj} y_j \quad \text{pentru } k = 1, 2, \ldots, K.$$

Semnalul de eroare pentru al k-lea neuron este:

$$\delta_{o_k} = -\frac{\partial E}{\partial net_k}.$$

Acest semnal l-am numit şi *semnal de învăţare*. Avem:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_{o_k} y_j,$$

deoarece

$$\frac{\partial net_k}{\partial w_{kj}} = y_j.$$

Putem scrie:

$$\Delta w_{kj} = \eta \delta_{o_k} y_j, \quad \text{pentru } k = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, J.$$

Calculăm:

$$\delta_{o_k} = -\frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial net_k}$$

Dar

$$f'_k(net_k) = \frac{\partial o_k}{\partial net_k}$$

şi

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k).$$

Obţinem:

$$\delta_{o_k} = (d_k - o_k) f'_k(net_k), \quad \text{pentru } k = 1, 2, \ldots, K.$$

Formula de ajustare a ponderilor este atunci:

$$\Delta w_{kj} = \eta(d_k - o_k) f_k'(net_k) y_j$$

şi este identică cu regula de învăţare delta pentru un singur perceptron.

Pentru funcţia de activare continuă unipolară, obţinem

$$f'(net) = \frac{e^{-net}}{(1+e^{-net})^2} = \frac{1}{1+e^{-net}} \cdot \frac{1+e^{-net}-1}{1+e^{-net}} = o(1-o).$$

În acest caz, avem:

$$\delta_{o_k} = (d_k - o_k) o_k (1 - o_k).$$

În cazul bipolar, am arătat în capitolul 3 că:

$$f'(net) = \frac{1}{2}(1 - o^2)$$

şi deci:

$$\delta_{o_k} = \frac{1}{2}(d_k - o_k)(1 - o_k^2).$$

În concluzie, actualizarea ponderilor se face pentru $k = 1, 2, \ldots, K$, $j = 1, 2, \ldots, J$ astfel:

$$w_{kj}' = w_{kj} + \eta(d_k - o_k) o_k (1 - o_k) y_j$$

pentru

$$o_k = \frac{1}{1 + e^{-net_k}}$$

şi

$$w_{kj}' = w_{kj} + \frac{1}{2}\eta(d_k - o_k)(1 - o_k^2) y_j$$

pentru

$$o_k = 2\left(\frac{1}{1 + e^{-net_k} - \frac{1}{2}}\right).$$

În general, putem scrie

$$\mathbf{W}' = \mathbf{W} + \eta \delta_o \mathbf{y}^t,$$

unde

$$\delta_o = \begin{bmatrix} \delta_{o1} \\ \delta_{o2} \\ \vdots \\ \delta_{oK} \end{bmatrix}.$$

Algoritmul de instruire a unei reţele monostrat de perceptroni de tip continuu prin regula delta este următorul:

Se dau perechile $(\mathbf{y}_1, \mathbf{d}_1)$, $(\mathbf{y}_2, \mathbf{d}_2)$,...,$(\mathbf{y}_P, \mathbf{d}_P)$, *unde* $\mathbf{y}_i$ *este un vector de* J *elemente,* $\mathbf{d}_i$ *este un vector de* K *elemente, iar componenta* J *a vectorului* $\mathbf{y}_i$, $i = 1, 2, \ldots, P$, *este* -1.

1. *Se aleg* $\eta > 0$ *şi* $E_{max} > 0$.
2. *Se iniţializează matricea* $\mathbf{W}$ *de* $K \times J$ *elemente cu valori aleatoare şi mici. Se iniţializează:* $p \leftarrow 1$, $q \leftarrow 1$ *şi* $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_p$, $\mathbf{d} \leftarrow \mathbf{d}_p$, $o_k \leftarrow f(\mathbf{w}_k^t \mathbf{y})$ *pentru* $k = 1, 2, \ldots, K$, *unde* $\mathbf{w}_k$ *este linia* k *din* $\mathbf{W}$.
4. *Se actualizează ponderile conform regulii*

$$\mathbf{w}_k \leftarrow \mathbf{w}_k + \frac{1}{2}\eta(d_k - o_k)(1 - o_k^2)\mathbf{y}$$

   pentru $k = 1, 2, \ldots, K$ *în cazul bipolar*
   şi conform regulii

$$\mathbf{w}_k \leftarrow \mathbf{w}_k + \eta(d_k - o_k)o_k(1 - o_k)\mathbf{y}$$

   pentru $k = 1, 2, \ldots, K$ *în cazul unipolar.*
5. *Se calculează eroarea pătratică cumulată:*

$$E \leftarrow E + \frac{1}{2}(d_k - o_k)^2, \quad k = 1, 2, \ldots, K.$$

6. `if` $p < P$ `then` $p \leftarrow p + 1$, $q \leftarrow q + 1$, `go to` *3.*
7. `if` $E \geq E_{max}$ `then` $E \leftarrow 0$, $p \leftarrow 1$, `go to` *3.*
   `else` *"s-a terminat instruirea",* `write` $\mathbf{W}$, q, E.

4.3 Regula delta generalizată

Vom generaliza acum regula delta de învăţare pentru reţele feedforward multistrat. Considerăm pentru aceasta o reţea de neuroni cu două straturi (fig. 4.4).

Straturile de neuroni ale căror ieşiri nu sunt direct accesibile se numesc *straturi interne* sau *straturi ascunse*. Pentru stratul ascuns avem:

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}}, \quad j = 1, 2, \ldots, J, \quad i = 1, 2, \ldots, I,$$

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial v_{ji}},$$

$$\Delta v_{ji} = \eta \delta_{yj} z_i,$$

unde δ_{y_j} este semnalul de eroare pentru stratul ascuns având ieşirea $\mathbf{y}$. Vom scrie

$$\delta_{yj} = -\frac{\partial E}{\partial net_j}, \quad j = 1, 2, \ldots, J$$

şi atunci

$$\delta_{y_j} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j}$$

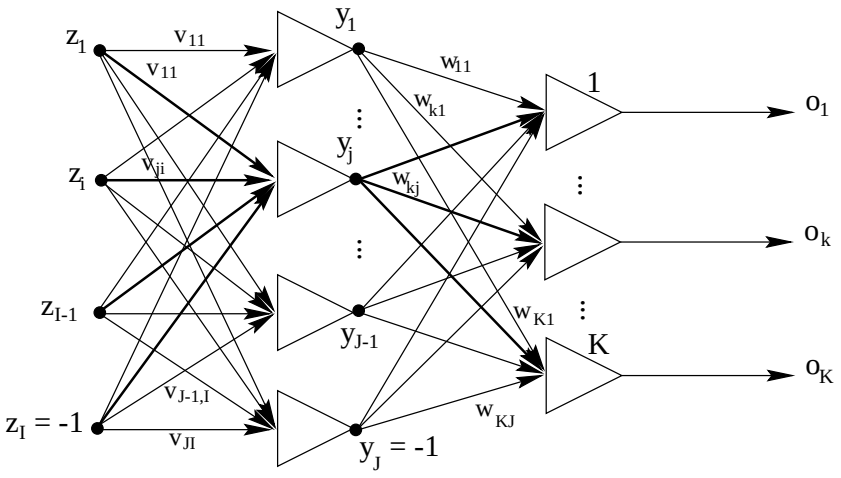


Figura 4.4: Reţea neurală feedforward cu un strat ascuns.

unde

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{K}\left\{d_k - f\left[net_k(\mathbf{y})\right]\right\}^2\right).$$

Dar

$$f_i'(net_j) = \frac{\partial y_j}{\partial net_j}$$

$$\begin{aligned}\frac{\partial E}{\partial y_j} &= -\sum_{k=1}^{K}(d_k - o_k)\cdot\frac{\partial}{\partial y_j}(f[net_k(\mathbf{y})]) \\ &= -\sum_{k=1}^{K}(d_k - o_k)f'(net_k)\frac{\partial net_k}{\partial y_j} \\ &= -\sum_{k=1}^{K}\delta_{o_k}w_{kj}\end{aligned}$$

deoarece

$$\delta_{o_k} = (d_k - o_k)f'(net_k)$$

$$net_k = \sum_{j=1}^{J} w_{kj}y_j.$$

Atunci:

$$\delta_{y_j} = f_j'(net_j)\sum_{k=1}^{K}\delta_{o_k}w_{kj}, \quad j = 1, 2, \ldots, J$$

$$\Delta v_{ji} = \eta f_j'(net_j)z_i\sum_{k=1}^{K}\delta_{o_k}w_{kj}, \quad j = 1, 2, \ldots, J, \quad i = 1, 2, \ldots, I.$$

Ultima relaţie este regula delta generalizată. Această regulă se poate rescrie în felul următor:

$$v'_{ji} = v_{ji} + \eta f'_j(net_j) z_i \sum_{k=1}^{K} \delta_{o_k} w_{kj}, \quad j = 1, 2, \ldots, J, \quad i = 1, 2, \ldots, I$$

sau, matriceal:

$$\mathbf{V}' = \mathbf{V} + \eta \delta_y \mathbf{z}^t,$$

unde

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_I \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} v_{11} & \ldots & v_{1I} \\ \vdots & \ddots & \vdots \\ v_{J1} & \ldots & v_{JI} \end{bmatrix} \quad \delta_y = \begin{bmatrix} \delta_{y_1} \\ \vdots \\ \delta_{y_J} \end{bmatrix}.$$

În această relaţie,

$$\delta_{y_j} = f'_j(net_j) \sum_{k=1}^{K} \delta_{o_k} w_{kj}$$

sau

$$\delta_y = \mathbf{w}_j^t \delta_o \mathbf{f}'_y,$$

unde $\mathbf{w}_j$ este coloana j din $\mathbf{W}$, iar

$$\mathbf{f}'_y = \begin{bmatrix} f'_{y_1} \\ \vdots \\ f'_{y_J} \end{bmatrix}.$$

Pentru cazul unipolar, folosim $f'_{y_j} = y_j(1 - y_j)$, iar pentru cazul bipolar avem $f'_{y_j} = \frac{1}{2}(1 - y_j^2)$.

Comparând regulile de actualizare a ponderilor:

pentru stratul de ieşire:	$\mathbf{W}' = \mathbf{W} + \eta \delta_o \mathbf{y}^t$
pentru stratul ascuns:	$\mathbf{V}' = \mathbf{V} + \eta \delta_y \mathbf{z}^t$

se observă că diferenţa semnificativă este indicele care localizează stratul şi modul în care se calculează vectorul semnalului de eroare δ:

$$\delta_o = \begin{bmatrix} \delta_{o_1} & \ldots & \delta_{o_K} \end{bmatrix}^t, \quad \delta_{o_k} = (d_k - o_k) f'_{o_k}(net_k)$$
$$\delta_y = \begin{bmatrix} \delta_{y_1} & \ldots & \delta_{y_J} \end{bmatrix}^t, \quad \delta_{y_j} = \mathbf{w}_j^t \delta_o f'_{y_j}(net_j).$$

Se observă că $\mathbf{w}_j^t \delta_o$ este produsul scalar al semnalelor de eroare provenind de la stratul următor.

Regula delta generalizată propagă eroarea cu câte un strat în urmă. Vom formaliza în continuare această metodă de instruire pentru o reţea neurală multistrat.

4.4 Instruirea unei reţele neurale multistrat prin propagarea în urmă a erorii

În general, o reţea neurală multistrat aplică vectorul de intrare $\mathbf{z}$ în vectorul de ieşire $\mathbf{o}$ astfel:

$$\mathbf{o} = N[\mathbf{z}],$$

unde N este un operator compus neliniar matricial. Pentru reţeaua cu un strat ascuns avem:

$$\mathbf{o} = \Gamma[\mathbf{W}\Gamma[\mathbf{V}\mathbf{z}]].$$

În acest caz, trebuie să ajustăm matricile $\mathbf{V}$ şi $\mathbf{W}$ astfel încât eroarea

$$\|\mathbf{d} - \mathbf{o}\|^2$$

să fie minimă. Algoritmul de instruire a unei reţele neurale multistrat prin propagarea în urmă a erorii (*error backpropagation*) este:

Se dau perechile $(\mathbf{z}_1, \mathbf{d}_1)$, $(\mathbf{z}_2, \mathbf{d}_2)$,...,$(\mathbf{z}_P, \mathbf{d}_P)$, *unde* $\mathbf{z}_i$ *este un vector de* I *elemente,* $\mathbf{z}_{i_I} = -1$ *şi* $i = 1, 2, \ldots, P$. *Se determină mărimea stratului ascuns. Acesta va avea ieşirile* $\mathbf{y}$, *unde* $\mathbf{y}$ *este un vector de* J *elemente, iar* $y_J = -1$. *Ieşirile* $\mathbf{o}$ *din reţea sunt vectori de* K *elemente.*

1. *Se aleg* $\eta > 0$ *şi* $E_{max} > 0$. *Se iniţializează matricea* $\mathbf{W}$ *de* $K \times J$ *şi matricea* $\mathbf{V}$ *de* $J \times I$ *elemente cu valori aleatoare, mici. Se iniţializează* $p \leftarrow 1$, $q \leftarrow 1$ *şi* $E \leftarrow 0$.

2. *Se iniţializează*

$$\mathbf{z} \leftarrow \mathbf{z}_p, \quad \mathbf{d} \leftarrow \mathbf{d}_p,$$

$$y_j \leftarrow f(\mathbf{v}_j^t \mathbf{z}) \text{ pentru } j = 1, 2, \ldots, J,$$

unde $\mathbf{v}_j$ *este un vector coloana reprezentând linia* j *din* $\mathbf{V}$,

$$o_k \leftarrow f(\mathbf{w}_k^t \mathbf{y}) \text{ pentru } k = 1, 2, \ldots, K,$$

unde $\mathbf{w}_k$ *este un vector coloana reprezentând linia* k *din* $\mathbf{W}$.

3. *Se calculează eroarea pătratică cumulată:*

$$E \leftarrow E + \frac{1}{2}(d_k - o_k)^2, \quad k = 1, 2, \ldots, K.$$

4. *Se calculează semnalele de eroare pentru cazul bipolar:*

$$\delta_{o_k} = \frac{1}{2}(d_k - o_k)(1 - o_k^2), \quad k = 1, 2, \ldots K,$$

$$\delta_{y_j} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^{K} \delta_{o_k} w_{kj}, \quad j = 1, 2, \ldots, J$$

respectiv, pentru cazul unipolar:

$$\delta_{o_k} = (d_k - o_k)(1 - o_k)o_k, \quad k = 1, 2, \ldots, K,$$

$$\delta_{y_j} = y_j(1 - y_j)\sum_{k=1}^{K} \delta_{o_k} w_{kj}, \quad j = 1, 2, \ldots, J.$$

5. *Se ajustează ponderile stratului de ieşire:*

$$w_{kj} \leftarrow w_{kj} + \eta\delta_{o_k} y_j, \quad k = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, J.$$

6. *Se ajustează ponderile stratului ascuns:*

$$v_{ji} \leftarrow v_{ji} + \eta\delta_{y_j} z_i, \quad j = 1, 2, \ldots J, \quad i = 1, 2, \ldots, I.$$

7. `if` $p < P$ `then` $p \leftarrow p + 1$, $q \leftarrow q + 1$, `go to` *2.*

8. `if` $E \geq E_{max}$ `then` $E \leftarrow 0$, $p \leftarrow 1$, `go to` *2.*
   `else` *"s-a terminat instruirea"*, `write` $\mathbf{W}$, $\mathbf{V}$, q, E.

În algoritmul de instruire prin propagarea în urmă a erorii, în loc de

$$\Delta w_{kj}^{n+1} = \eta\delta_{o_k} y_j$$

$$\Delta v_{ji}^{n+1} = \eta\delta_{y_j} z_i$$

putem lua

$$\Delta w_{kj}^{n+1} = \eta\delta_{o_k} y_j + \alpha\Delta w_{kj}^{n}$$
$$\Delta v_{ji}^{n+1} = \eta\delta_{y_j} z_i + \alpha\Delta v_{ji}^{n},$$

unde α este o constantă numită *momentum*. Prin regula delta, nu ne deplasăm de fapt în direcţia gradientului negativ al erorii, deoarece ponderile se modifică la fiecare pas. Pot apărea oscilaţii dacă η este mare. Dacă η este mic, oscilaţiile sunt neglijabile, dar rata învăţării este prea mică. Termenul în care apare α filtrează oscilaţiile. De obicei, se ia $\alpha = 0,9$ şi se poate mări η. Dacă $\alpha = 0$, ajungem la acelaşi rezultat, dar trebuie să folosim o valoare mai mică pentru η. Instruirea poate dura însă mai mult.

Algoritmul de instruire prin propagarea în urmă a erorii face o serie de deplasări în sensul gradienţilor negativi ai valorilor individuale pe care le ia E_p, adică pentru un pattern, şi nu în sensul gradientului negativ al lui E. Acest lucru se întâmplă deoarece η nu este infinit de mic. Din acest motiv, în anumite cazuri, după citirea lui $\mathbf{z}_p$, se poate ca E să crească! Eroarea referitoare doar la $\mathbf{z}_p$ scade însă, sau rămâne zero.

În general, funcţia de activare bipolară duce la rezultate mai bune decât funcţia unipolară.

În cazul discret, intrarea fixă este cerută de teorema perceptronului, permiţând crearea de hiperplane de separaţie care nu trec prin origine. La regula delta

generalizată, intrarea fixă este recomandată nu de o teoremă, ci de ideea obţinerii unei aproximări mai flexibile.

Eroarea cumulată este:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk} - o_{pk})^2.$$

Această eroare poate fi foarte mare dacă P şi K sunt mari. De aceea, este mai adecvat să folosim:

$$E_m = \frac{1}{PK}\sqrt{\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk} - o_{pk})^2},$$

adică *eroarea medie normalizată*.

Atunci când reţeaua este instruită ca şi clasificator, ne interesează doar *eroarea de decizie*:

$$E_d = \frac{N_{er}}{PK},$$

unde N_{er} este numărul total de erori de clasificare. De obicei, ieşirile dorite sunt 0, cu excepţia uneia care este 1 şi corespunde clasei din care face parte pattern-ul respectiv. O reţea multistrat poate fi transformată într-un clasificator, după instruire, prin înlocuirea perceptronilor de tip continuu cu perceptroni discreţi.

Clasificarea este o formă importantă a calculului neural, însă ea limitează potenţialul de calcul al unei reţele, deoarece foloseşte răspunsul binar. O reţea multistrat poate fi utilizată şi pentru aproximarea funcţiilor.

4.5 Factori ai învăţării

Algoritmul de instruire prin propagarea în urmă a erorii poate fi interpretat ca o problemă de optimizare: se minimizează eroarea medie cumulată E_m, mergând pe direcţia gradientului negativ.

O dificultate tipică ar fi că prin acest algoritm se poate să obţinem doar un minim local pentru funcţia eroare. Prin faptul că impunem o valoare minimă acceptabilă pentru E_m, putem controla acest lucru. Se poate însă să nu avem o convergenţă a algoritmului, dacă ne îndreptăm spre un minim local în loc de a ne îndrepta spre un minim global. Este de fapt un algoritm de tip *greedy*.

Totuşi, problema minimelor locale nu este o problemă majoră a acestui algoritm. Aceasta deoarece algoritmul are o natură stohastică: există o anumită randomizare în instruire. Cu cât reţeaua este mai mare, cu atât mai bine va funcţiona instruirea. Natura stohastică este dată atât de ponderile iniţiale, cât şi de pattern-urile de instruire. Chiar când pattern-urile de instruire au o natură deterministă, adăugarea unui zgomot cu medie zero poate duce al îmbunătăţirea eficienţei instruirii.

Instruirea funcţionează cel mai bine în condiţii de randomizare. De aceea, ponderile iniţiale se aleg cu valori aleatoare, iar pattern-urile de instruire este bine să fie într-o secvenţă aleatoare.

Să studiem acum efectul lui λ asupra lui $f'(net)$ (fig. 4.5). Avem

$$f'(net) = \frac{2\lambda e^{-\lambda net}}{(1+e^{-\lambda net})^2}$$

pentru funcţia de activare continuă bipolară.

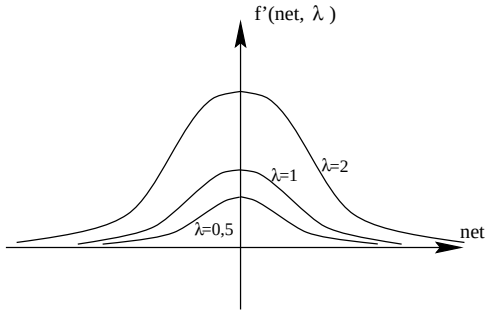


Figura 4.5: Efectul lui λ asupra lui $f'(net)$.

Deoarece ponderile sunt ajustate proporţional cu $f'(net)$, înseamnă că ponderile corespunzătoare neuronilor pentru care $net \cong 0$ se modifică cel mai mult. Ponderile neuronilor cu răspunsuri incerte sunt cele mai afectate. Deoarece semnalele de eroare δ_{ok} şi δ_{yj} sunt de asemenea proporţionale cu $f'(net)$, rezultă că, componentele erorii care se propagă în urmă sunt mari doar pentru neuronii cu răspunsuri incerte.

Se observă că, pentru o constantă de învăţare fixată, ajustarea ponderilor este proporţională cu λ. Folosirea funcţiilor de activare cu valoare mare pentru λ are acelaşi efect cu utilizarea unei constante de învăţare mari. De aceea, este recomandabil să păstrăm $\lambda = 1$ şi să controlăm viteza de învăţare doar prin η.

Valoarea optimă pentru constanta de învăţare η depinde de problema de rezolvat. O valoare mare pentru η duce al viteză mai mare de învăţare, dar se poate ca astfel să trecem de soluţie. O valoare mai mică pentru η duce la o rafinare (paşi mai mici) care însă înseamnă şi paşi mai mulţi, deci timp de procesare mai îndelungat. În concluzie, η trebuie ales în mod experimental pentru fiecare problemă. Cel mai bine, se porneşte cu η având valoare mică şi se încearcă mărirea acestui parametru pentru a mări viteza algoritmului. S-a observat că în general valorile pentru η sunt între 10^{-3} şi 10.

Una dintre cele mai importante probleme este alegerea arhitecturii reţelei. Să presupunem că avem un singur strat ascuns de J neuroni, că stratul de ieşire are K neuroni şi că avem I noduri de intrare în reţea. Evident, I este dependent de problema de rezolvat, deci îl considerăm fixat. Cum îi alegem pe J şi pe K?

Dacă reţeaua funcţionează ca un clasificator, K poate fi egal cu numărul de clase. Dar K poate fi considerat şi $\log_2 C$, unde C este numărul claselor, dacă folosim o reprezentare binară a claselor. De exemplu, pentru 2 neuroni de ieşire putem avea clasele 0, 1, 2, 3, corespunzătoare ieşirilor 00, 01, 10, 11. Numărul

de neuroni de ieşire pentru un clasificator cu C clase poate fi deci un întreg între $\log_2 C$ şi C. Pe de altă parte, comprimarea stratului de ieşire sub C neuroni poate afecta perioada de instruire şi robusteţea reţelei finale. De aceea, în cazul unui clasificator, se ia de obicei $K = C$.

Alegerea lui J este încă o problemă în studiu. Reţelele cu un singur strat ascuns pot forma regiuni arbitrare de decizie pentru pattern-uri n-dimensionale. Să presupunem că avem o colecţie de pattern-uri n-dimensionale liniar separabile în M regiuni disjuncte aparţinând fiecare la una dintre cele R clase, $R \leq M$. Evident, trebuie ca numărul de pattern-uri de instruire, P, să fie mai mare sau cel mult egal cu M.

Cu cât P/M este mai mare, cu atât instruirea este mai fină. Mirchandini şi Cao [1] au arătat care este numărul maxim de regiuni liniar separabile folosind J neuroni:

$$\sum_{k=0}^{n} \left(\begin{array}{c} J \\ k \end{array} \right),$$

unde $\left(\begin{array}{c} J \\ k \end{array} \right) = 0$ pentru $k > J$.

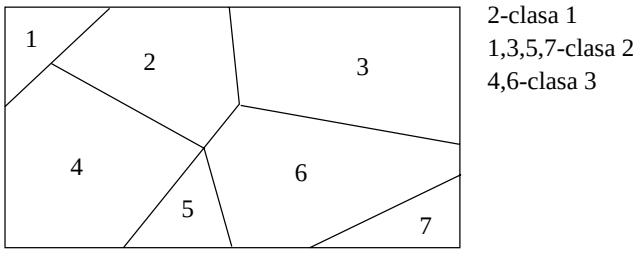


Figura 4.6: Regiuni disjuncte care aparţin la trei clase.

În exemplul nostru (fig. 4.6), avem $n = 2$ şi $M = 7$. Deci,

$$\left(\begin{array}{c} J \\ 0 \end{array} \right) + \left(\begin{array}{c} J \\ 1 \end{array} \right) + \left(\begin{array}{c} J \\ 2 \end{array} \right) = 1 + \frac{J}{2} + \frac{J^2}{2} \geq 7.$$

De aici, rezultă o estimare pentru mărimea stratului ascuns: $J = 3$. Dacă avem $n \geq J$, atunci numărul maxim de regiuni este

$$\left(\begin{array}{c} J \\ 0 \end{array} \right) + \left(\begin{array}{c} J \\ 1 \end{array} \right) + \ldots \left(\begin{array}{c} J \\ J \end{array} \right) = 2^J$$

şi de aici îl obţimem pe J.

[1] Mirchandini, G., W. Cao "On Hidden Nodes in Neural Nets". IEEE Trans. Circuits and Systems, 36, 1989, 661-664.

4.6 Reţele feedforward multistrat folosite ca aproximatori universali

Ne propunem să aproximăm funcţia continuă $h(x)$ cu funcţia treaptă $H(\mathbf{w}, x)$ (fig. 4.7).

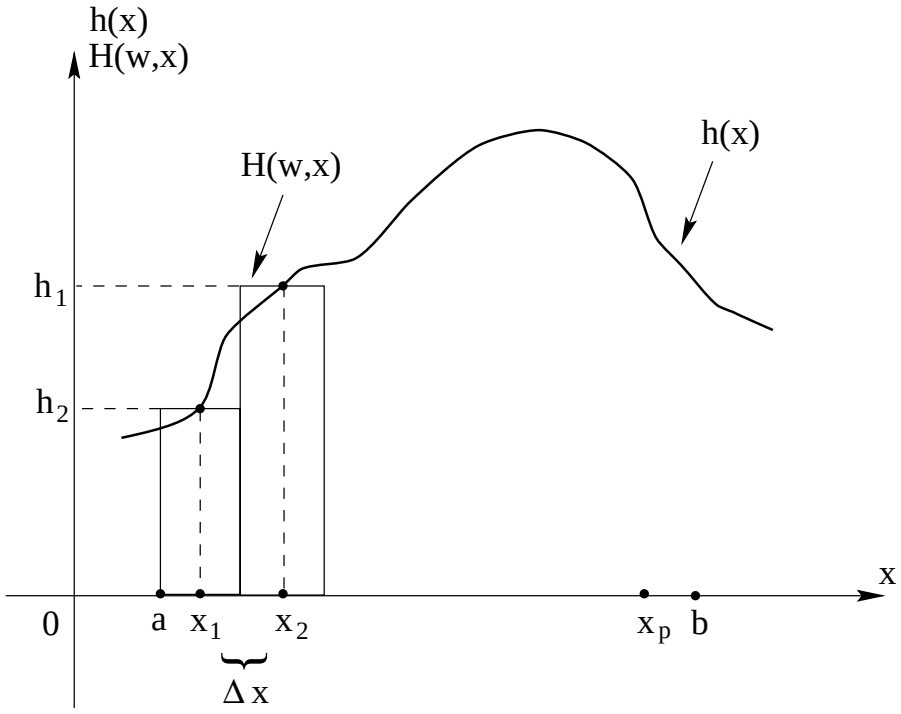


Figura 4.7: Aproximarea funcţiei $h(x)$ cu funcţia treaptă $H(\mathbf{w}, x)$.

Presupunem că se cunosc P eşantioane, adică valorile funcţiei $h(x)$ în punctele x_1, x_2,..., x_P. Presupunem că

$$x_{i+1} - x_i = \Delta\mathbf{x} = \frac{b-a}{P}, \quad i = 1, 2, \ldots, P.$$

Definim funcţia

$$\xi(x) = \frac{1}{2}\text{sgn}(x) + \frac{1}{2} = \begin{cases} 0 & \text{pentru } x < 0 \\ \text{nedefinit} & \text{pentru } x = 0 \\ 1 & \text{pentru } x > 0 \end{cases}.$$

Scriem atunci:

$$H(\mathbf{w}, x) = \sum_{i=1}^{P} h_i[\xi\left(x - x_i + \frac{\Delta x}{2}\right) - \xi\left(x - x_i - \frac{\Delta x}{2}\right)],$$

unde $h_i = h(x_i)$.

Putem scrie:

$$\xi\left(x - x_i + \frac{\Delta x}{2}\right) - \xi\left(x - x_i - \frac{\Delta x}{2}\right) =$$
$$\frac{1}{2}\text{sgn}\left(x - x_i + \frac{\Delta x}{2}\right) - \frac{1}{2}\text{sgn}\left(x - x_i - \frac{\Delta x}{2}\right).$$

Acest termen este o fereastră de înalţime unitară şi lăţime Δx (fig. 4.8).

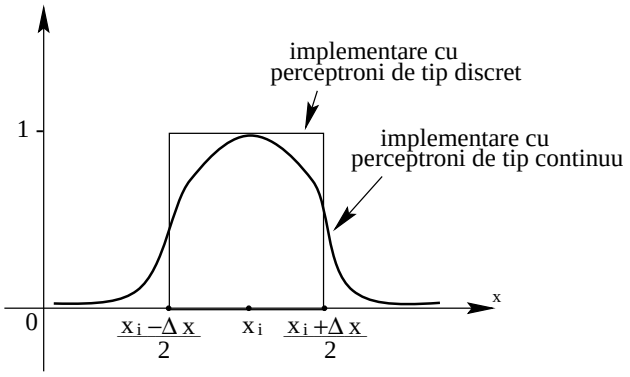


Figura 4.8: Fereastra de înălţime unitară şi lăţime Δx.

Fereastra se poate implementa ca în figura 4.9.

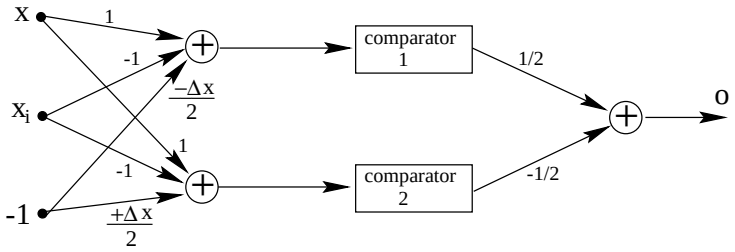


Figura 4.9: Reţea neurală care implementează fereastra de înălţime unitară şi lăţime Δx.

O reţea cu $2P$ perceptroni discreţi poate implementa funcţia $H(\mathbf{w}, x)$. Pentru un termen, reţeaua se poate descrie ca în figura 4.10.

În general, o funcţie $h(\mathbf{x})$, unde $\mathbf{x}$ este un vector, poate fi aproximată printr-o reţea neurală cu un singur strat ascuns. În ce măsură reţeaua este instruibilă (de pildă, prin regula delta generalizată), rămâne o problemă deschisă.

Fără a demonstra, vom menţiona două importante proprietăţi:

1. O reţea feedforward cu un singur strat ascuns poate aproxima oricât de bine orice funcţie neliniară continuă.

2. O reţea feedforward cu un singur strat ascuns poate implementa o funcţie booleană arbitrară.

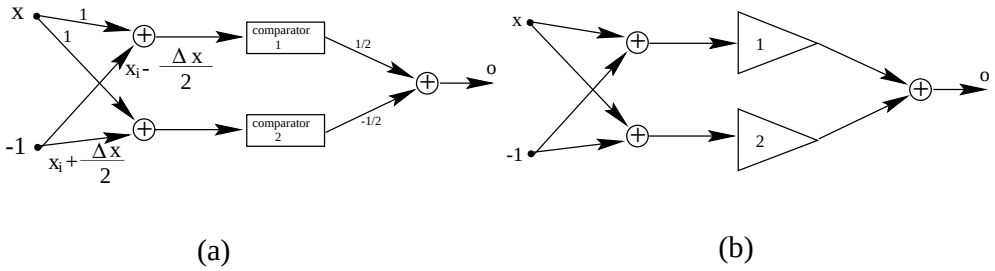


Figura 4.10: Implementare folosind (a) perceptroni de tip discret şi (b) perceptroni de tip continuu; cu cât λ este mai mare, cu atât mai mult forma ferestrei se apropie de cea rectangulară.

4.7 Teorema lui Kolmogorov şi reţelele neurale

În loc de a aproxima o funcţie continuă, ne punem acum problema reprezentării ei exacte printr-o reţea neurală.

În 1900, David Hilbert a formulat următoarea ipoteză: *o funcţie continuă de mai multe variabile nu este în mod necesar decompozabilă într-o suprapunere de funcţii continue de un număr mai mic de variabile.* Această presupunere a fost contrazisă în 1957 de Kolmogorov printr-o teoremă a cărei aplicaţii au apărut abia acum, în contextul reţelelor neurale.

Teorema lui Kolmogorov afirmă că orice funcţie f continuă definită pe un cub n-dimensional este reprezentabilă astfel:

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \varphi_q \left(\sum_{p=1}^{n} \psi_{pq}(x_p) \right),$$

unde φ_q, $q = 1, \ldots, 2n+1$ şi ψ_{pq}, $p = 1, \ldots, n$ sunt funcţii continue de o singură variabilă.

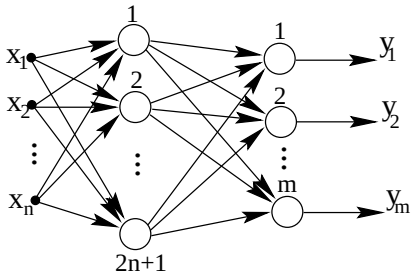


Figura 4.11: Reţea neurală care implementează o funcţie oarecare f.

Interpretarea neurală este următoarea:

T1 (Hecht-Nielsen, 1986). Fie $f : [0,1]^n \subset \Re^n \to \Re^m$, $f(x) = y$, o funcţie continuă oarecare. Atunci, f poate fi implementată exact

de către o reţea neurală feedforward cu un singur strat ascuns având arhitectura din figura 4.11.

Definim acum funcţiile de activare pentru această arhitectură.

Fie z_k ieşirea din neuronul ascuns k. Avem:

$$z_k = \sum_{j=1}^{n} \lambda^k \psi(x_j + k\varepsilon) + k,$$

unde λ este o constantă reală, ψ este o funcţie monoton crescătoare, independentă de f, ε este o constantă raţională $0 < \varepsilon \leq \delta$, unde δ este o constantă pozitivă arbitrar aleasă.

Ieşirile y_i se calculează astfel:

$$y_i = \sum_{k=1}^{2n+1} g_i(z_k),$$

unde funcţiile g_i, $i = 1, \ldots, m$ sunt reale şi continue, depinzând de f şi ε.

Această teoremă nu ne spune cum să obţinem funcţia ψ sau constanta ε. Este o teoremă *existenţială*. Ea se poate extinde pentru orice funcţii continue pe un compact (mulţime închisă şi mărginită). Acest rezultat a fost primul pas. Cybenco (1989) a arătat că orice funcţie continuă definită pe un compact din $\Re^n$ poate fi aproximată oricât de bine de o reţea feedforward cun un singur strat ascuns folosind funcţii de activare sigmoidale. Uneori se pot obţine aproximări mai compacte folosind mai multe straturi ascunse.

Sprecher (1996) a găsit algoritmul de construcţie a funcţiei ψ (din teorema lui Hecht-Nielsen), astfel încât ψ este monoton crescătoare, independent de f şi n.

4.8 Aplicaţii

Reţelele feedforward multistrat pot fi aplicate cu succes la multe probleme de clasificare şi de recunoaştere. Pentru aceasta, nu este necesar să fie cunoscut un model formal pentru clasificare sau recunoaştere. Este suficient să utilizăm o arhitectură potrivită şi o mulţime suficientă de pattern-uri de instruire, apoi aplicăm algoritmul de instruire prin propagarea în urmă a erorii. Soluţia se obţine deci prin experimentare şi simulare, nu printr-o abordare formală riguroasă.

Cu toate că nu este foarte clar ce constituie o mulţime adecvată de pattern-uri de instruire, sunt raportate o serie de aplicaţii în recunoaşterea vorbirii, a semnalelor sonore etc.

Să notăm că ieşirile cu mai mult de două valori pentru un neuron pot modela incertitudinea. Modul convenţional de a construi un *sistem expert* necesită un expert uman care să formuleze regulile cu ajutorul cărora să fie analizate datele de intrare. Numărul acestor reguli poate fi foarte mare. Pe de altă parte, de

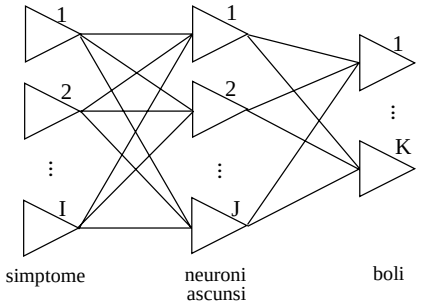


Figura 4.12: Sistem expert conexionist pentru diagnoza medicală.

multe ori este dificil de formulat aceste reguli. Deoarece reţelele neurale pot fi instruite fără ca ele să încapsuleze cunoştinţele în reguli, să vedem în ce măsură ele pot fi aplicate ca o alternativă la sistmele expert convenţionale.

Reţelele neurale pentru diagnosticare, defectoscopie, predicţie, recunoaşterea formelor rezolvă în esenţă probleme de clasificare, asociere şi generalizare. Aceste reţele pot achiziţiona cunoştinţe fără să extragă reguli de tip IF-THEN de la un expert uman. După instruire, ele pot funcţiona ca sisteme expert. Ceea ce va lipsi este explicarea: un sistem expert neural nu poate explica utilizatorului raţionamentul deciziilor luate. Numim aceste sistem expert *sisteme expert conexioniste*.

Atunci când o reţea instruită este testată cu un pattern substanţial diferit de cele folosite pentru instruire, răspunsul obţinut se presupune că rezolvă o problemă de generalizare. Exemple tipice de generalizare sunt diagnosticarea şi predicţia. Figura 4.12 descrie funcţionarea unui sistem expert conexionist pentru diagnostic medical.

Într-un sistem expert convenţional, o dificultate este formularea şi introducerea regulilor de către expertul uman. Sistemul expert conexionist poate învăţa din diagnosticul curent care a fost acceptat. O atenţie specială trebuie dată alegerii lui J. Dacă J este prea mic, apar dificultăţi de aplicare a celor I intrări în cele K ieşiri. Dacă J este prea mare, creşte inutil timpul de instruire şi de diagnosticare şi ponderile sunt dificil de estimat corect.

O serie de aplicaţii folosesc pattern-uri temporale care se obţin prin eşantionarea unor semnale de tip continuu (fig. 4.13).

4.9 Exemple

Exemplu de construire a unui clasificator multistrat pentru pattern-uri liniar neseparabile

Fie funcţia de decizie XOR:

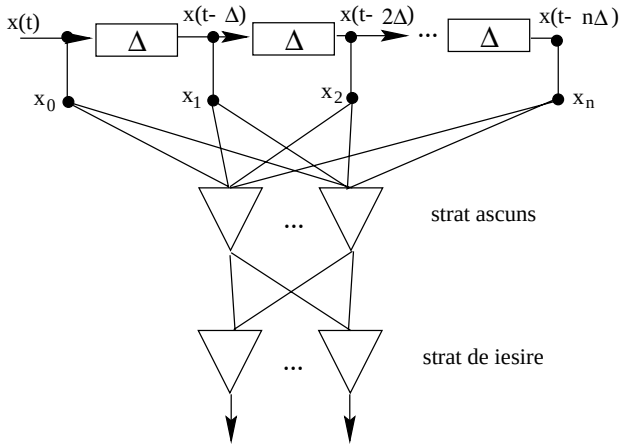


Figura 4.13: Reţea neurală cu întârziere cu un singur canal de achiziţie.

x_1	x_2	ieşirea
0	0	1
0	1	-1
1	0	-1
1	1	1

Punctele având coordonatele x_1 şi x_2 de mai sus sunt notate cu A, B, C, D.

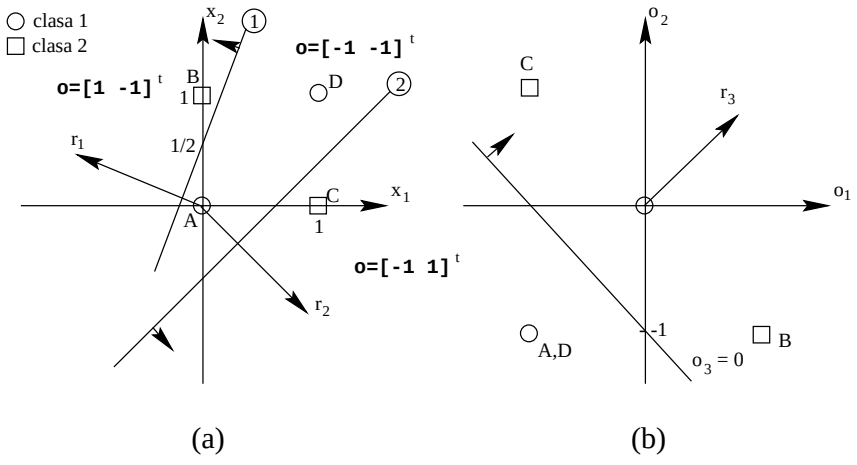


Figura 4.14: Suprafeţe de decizie în problema XOR.

Alegem:

$$1: \quad -2x_1 + x_2 - \tfrac{1}{2} = 0$$
$$2: \quad x_1 - x_2 - \tfrac{1}{2} = 0$$

în mod arbitrar (fig. 4.14a). Pentru aceste linii, vectorii normali unitari sunt:

$$r_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \end{bmatrix}^t$$
$$r_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \end{bmatrix}^t.$$

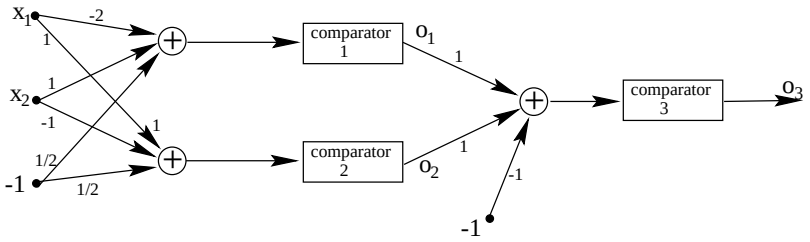


Figura 4.15: Reţeaua neurală care implementează problema XOR.

Reţeaua neurală din figura 4.15 implementează aceste suprafeţe de decizie prin intermediul stratului ascuns. Pentru acestea, avem:

$$o_1 = \text{sgn}\left(-2x_1 + x_2 - \tfrac{1}{2}\right) \quad o_2 = \text{sgn}\left(x_1 - x_2 - \tfrac{1}{2}\right).$$

Arbitrar, se alege linia de decizie (fig. 4.14b)

$$o_1 + o_2 + 1 = 0,$$

şi atunci

$$o_3 = \text{sgn}(o_1 + o_2 + 1).$$

Reţeaua neurală implementată aici clasifică pattern-urile A, B, C, D astfel:

Simbol	x_1	x_2	o_1	o_2	$o_1 + o_2 = 1$	o_3	clasa
A	0	0	-1	-1	-	-1	2
B	0	1	+1	-1	+	+1	1
C	1	0	-1	+1	+	+1	1
D	1	1	-1	-1	-	-1	2

Exemplu de instruire prin propagarea în urmă a erorii

Presupunem că avem o reţea cu trei neuroni cu un strat ascuns (fig. 4.16).

Neuronii 1,2 şi 3 sunt fictivi şi ne simplifică notaţia. Presupunem că ponderile au fost iniţializate şi că o_1 şi o_2 sunt calculate conform unui pattern de intrare. Calculăm o_3, o_4, o_5 şi apoi:

$$\delta_5 = (d_5 - o_5)o_5(1 - o_5)$$
$$\delta_3 = f'(net_3)\sum_{k=5}^{5} \delta_k w_{k3} = o_3(1 - o_3)\delta_5 w_{53}$$

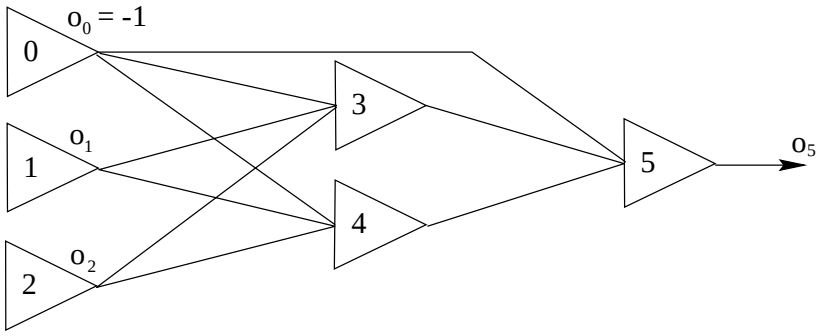


Figura 4.16: Reţea neurală multistrat. Funcţiile de activare sunt unipolare.

$$\begin{aligned}
\delta_4 &= o_4(1-o_4)\delta_5 w_{54} \\
\Delta w_{50} &= -\eta\delta_5 \\
\Delta w_{53} &= \eta\delta_5 o_3 \\
\Delta w_{54} &= \eta\delta_5 o_4.
\end{aligned}$$

Apoi:

$$\begin{array}{ll}
\Delta w_{30} = -\eta\delta_3 & \Delta w_{40} = -\eta\delta_4 \\
\Delta w_{31} = \eta\delta_3 o_1 & \Delta w_{41} = \eta\delta_4 o_1 \\
\Delta w_{32} = \eta\delta_3 o_2 & \Delta w_{42} = \eta\delta_4 o_2.
\end{array}$$

Putem astfel să construim prin instruire un clasificator multistrat pentru pattern-urile liniar neseparabile din exemplul precedent (problema XOR cu două variabile). Definim matricile:

$$\mathbf{W} = \begin{bmatrix} w_{50} & w_{53} & w_{54} \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} w_{30} & w_{31} & w_{32} \\ w_{40} & w_{41} & w_{42} \end{bmatrix}.$$

Pentru $\eta = 0,1$ se ajunge după instruire la:

$$\mathbf{W} = \begin{bmatrix} -3,967 & -8,160 & -5,376 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 6,169 & 3,854 & 4,281 \\ -1,269 & -4,674 & -4,578 \end{bmatrix}.$$

Dorim ca reţeaua să funcţioneze ca un clasificator cu ieşiri binare. Înlocuim neuronii de tip continuu cu neuroni binari şi păstrăm ponderile pe care le-am calculat pentru cazul continuu.

Exemplu de alegere a mărimii stratului ascuns

În figura 4.17 avem un clasificator pentru problema XOR cu $n = 2$. Presupunând $J \leq n$, obţinem pentru $J = 2$ numărul maxim de regiuni $2^2 = 4$.

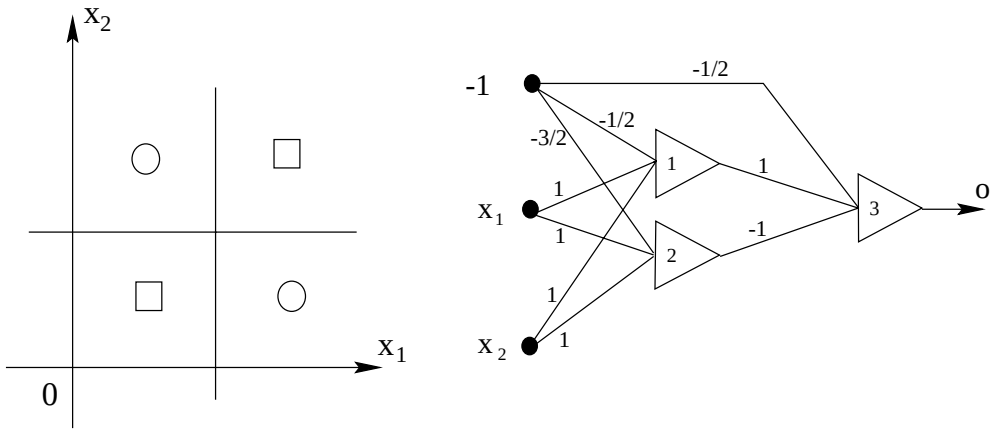


Figura 4.17: Problema XOR pentru $n = 2$.

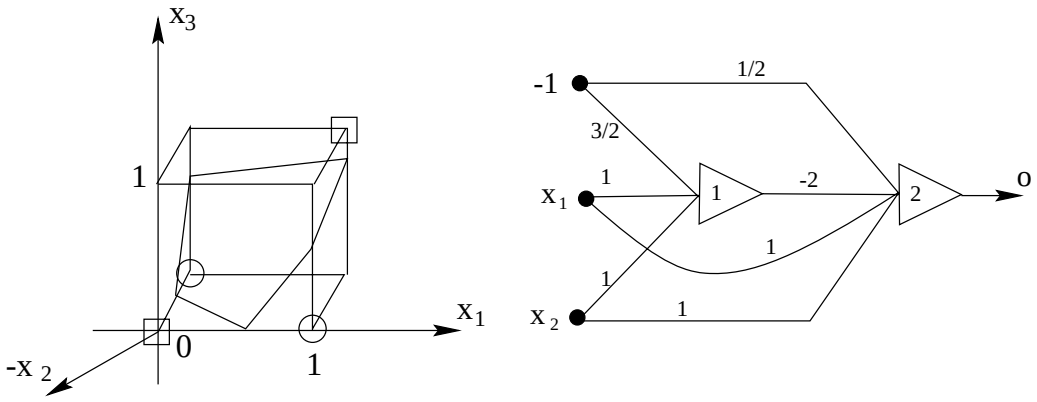


Figura 4.18: Problema XOR pentru $n = 3$.

Să considerăm acum aceeaşi problemă pentru $n = 3$ (fig. 4.18), unde $x_3 = x_1 x_2$.

Luând $n = 3$ şi $M = 2$ obţinem $J = 1$. Am rezolvat deci aceeaşi problemă, dar cu mai puţini neuroni.

4.10 Exerciţii

1. Pattern-urile liniar neseparabile

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad \mathbf{x}_5 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\mathbf{x}_6 = \begin{bmatrix} 2 \\ 2,5 \end{bmatrix} \quad \mathbf{x}_7 = \begin{bmatrix} -2 \\ 0 \end{bmatrix} \quad \mathbf{x}_8 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_9 = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \quad \mathbf{x}_{10} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

trebuie clasificate astfel:

$$\begin{array}{ll} \text{clasa 1:} & \mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9 \\ \text{clasa 2:} & \text{restul.} \end{array}$$

Elaboraţi o reţea neurală de perceptroni discreţi cu un strat ascuns care să funcţioneze ca un clasificator. Nu instruiţi reţeaua de neuroni, ci folosiţi metoda din primul exemplu.

2. Fie prototipurile în formă extinsă

$$\mathbf{x}_1 = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$$

$$\mathbf{x}_5 = \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{x}_6 = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{x}_7 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_8 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

O maşină multistrat cu doi perceptroni discreţi bipolari în stratul ascuns şi un singur perceptron discret bipolar de ieşire trebuie să clasifice prototipurile astfel încât:

$$\begin{array}{ll} \text{clasa 1:} & \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \\ \text{clasa 2:} & \text{restul .} \end{array}$$

(a) Verificaţi dacă vectorii pondere

$$\mathbf{w}_1 = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$

asigură separarea liniară a pattern-urilor (primul strat).

(b) Completaţi elaborarea clasificatorului folosind rezultatul anterior şi calculaţi ponderile perceptronului de ieşire.

3. Demonstraţi că, pentru $n = 2$, numărul J al neuronilor stratului ascuns necesari pentru partiţionarea prin hiperplane în M regiuni este

$$J = \frac{1}{2}\left(\sqrt{8M-7} - 1\right).$$

4. Fie problema clasificării în două clase a unor pattern-uri planare ($n = 2$) rezolvată printr-o arhitectură de reţea neurală cu $J = 8$ şi $K = 2$. Determinaţi o margine inferioară pentru P, care este numărul vectorilor de instruire. Indicaţie: Acest număr este egal cu numărul regiunilor separabile, M.

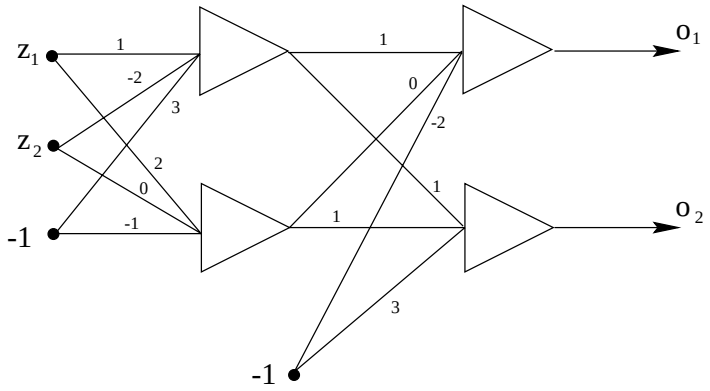


Figura 4.19: Reţea neurală multistrat pentru exerciţiul 5.

5. Reţeaua din figura 4.19, dacă a fost instruită corect, trebuie să răspundă cu $o_1 = 0,95$ şi $o_2 = 0,05$ la pattern-ul de intrare

$$\begin{bmatrix} z_1 \\ z_2 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}.$$

Ponderile au fost iniţializate ca în figură. Se alege $\eta = 1$, iar neuronii folosesc funcţia de activare

$$f(net) = \frac{1}{1 + e^{-net}}.$$

Analizaţi un pas al algoritmului de instruire prin propagarea în urmă a erorii realizând următoarele operaţii:

(a) Găsiţi matricile ponderilor, $\mathbf{V}$, $\mathbf{W}$.

(b) Calculaţi net_j, $\mathbf{y}$, net_k, $\mathbf{o}$.

(c) Calculaţi $f'(net_j)$ şi $f'(net_k)$.

(d) Calculaţi δ_o şi δ_y.

(e) Calculaţi $\Delta\mathbf{V}$ şi $\Delta\mathbf{W}$.

(f) Găsiţi noile valori pentru $\mathbf{V}$ şi $\mathbf{W}$.

6. (C) Implementaţi algoritmul de instruire prin propagarea în urmă a erorii pentru I, J, K şi η selectabile şi un singur strat ascuns. Folosiţi perceptroni bipolari de tip continuu.

7. (C) Elaboraţi clasificatorul pentru literele A, I şi O definite ca în figura 4.20, cu ajutorul programului de la exerciţiul 6. Presupuneţi două tipuri de reprezentări interne:

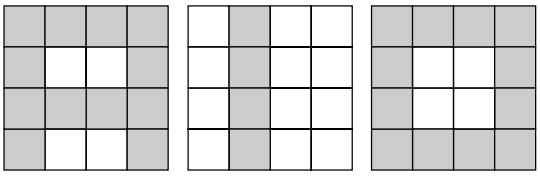


Figura 4.20: Literele A, I şi O pentru exerciţiul 7.

(a) $n = 16$, $P = 3$, $I = 17$, $J = 9$, $K = 3$ şi

$$\begin{array}{rl} \text{clasa A:} & \mathbf{x}_1 = [\ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ ]^t \\ \text{clasa I:} & \mathbf{x}_2 = [\ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ ]^t \\ \text{clasa O:} & \mathbf{x}_3 = [\ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ ]^t \end{array}$$

(b) $n = 2$, $P = 16$, $I = 3$, $J = 9$, $K = 3$ şi

$$\begin{array}{rl} \text{clasa A, O:} & \mathbf{x}_1 = [\ 1\ \ 1\ ]^t \\ \text{clasa A, I, O:} & \mathbf{x}_1 = [\ 1\ \ 2\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 1\ \ 3\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 1\ \ 4\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 2\ \ 1\ ]^t \\ \text{clasa I:} & \mathbf{x}_1 = [\ 2\ \ 2\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 2\ \ 4\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 3\ \ 1\ ]^t \\ \text{clasa A, I:} & \mathbf{x}_1 = [\ 3\ \ 2\ ]^t \\ \text{clasa A:} & \mathbf{x}_1 = [\ 3\ \ 3\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 3\ \ 4\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 4\ \ 1\ ]^t \\ \text{clasa I, O:} & \mathbf{x}_1 = [\ 4\ \ 2\ ]^t \\ \text{clasa O:} & \mathbf{x}_1 = [\ 4\ \ 3\ ]^t \\ \text{clasa A, O:} & \mathbf{x}_1 = [\ 4\ \ 4\ ]^t \end{array}$$

Încercaţi diferite valori pentru η. Evaluaţi numărul de cicluri de instruire pentru ambele arhitecturi. Ieşirile trebuie să fie:

$$\begin{array}{ll} \text{pentru A:} & [\ 1\ \ -1\ \ -1\ ] \\ \text{pentru I:} & [\ -1\ \ 1\ \ -1\ ] \\ \text{pentru O:} & [\ -1\ \ -1\ \ 1\ ]. \end{array}$$

8. (C) Elaboraţi o reţea pentru pattern-urile liniar neseparabile din figura 4.21 folosind programul scris la exerciţiul 6.

   (a) Selectaţi un număr potrivit de neuroni în stratul ascuns. Folosiţi $K =$

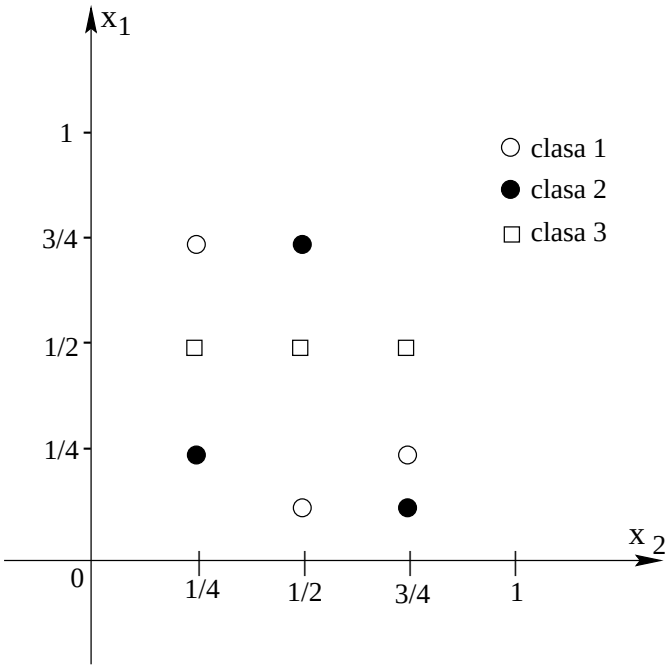


Figura 4.21: Pattern-uri neseparabile pentru exerciţiul 8.

3, astfel încât ieşirile să fie:

$$\begin{array}{ll} \text{pentru clasa 1:} & [\ 1 \quad -1 \quad -1\ ]^t \\ \text{pentru clasa 2:} & [\ -1 \quad 1 \quad -1\ ]^t \\ \text{pentru clasa 3:} & [\ -1 \quad -1 \quad 1\ ]^t \end{array}$$

(b) Instruiţi reţeaua pentru vectorii din figură şi diferite valori pentru η.

9. (C) Fie $h(x) = 0,8 \sin \pi x, \quad -1 \leq x \leq 1$ şi aproximarea ei printr-o reţea neurală $o(x)$. Construiţi o reţea de perceptroni bipolari de tip continuu cu un strat ascuns care aproximează $h(x)$ prin $o(x)$.

10. (C) Realizaţi un sistem expert conexionist care să vă ajute la planificarea timpului vostru liber. În funcţie de factorii de circumstanţă, sistemul expert vă va spune ce activitate să alegelti. Activităţile posibile sunt:

    (a) vizită

    (b) sport

    (c) film

    (d) somn

    (e) cumpărături

    Factorii care determină alegerea activităţii sunt:

(a) resursele financiare disponibile

(b) starea vremii

(c) durata de timp care poate fi afectată

(d) existenţa unui prieten care poate fi vizitat

(e) starea de oboseală

(f) necesitatea de a face cumpărături

(g) ultima activitate făcută în timpul liber

(h) existenţa unui film bun

Sistemul va fi realizat ca o reţea de perceptroni. Propuneţi o secvenţă de 20 de perechi de forma (activităţi, factori de decizie) ca secvenţă de instruire, cu cel puţin 2 perechi pentru fiecare activitate din listă. Codificaţi intrările ca variabile mixte (continue şi binare) din intervalul $[0, 1]$ şi atribuiţi 0 intrărilor care nu au importantă. Instruiţi sistemul şi, apoi, testaţi-l cu date de intrare diferite de cele folosite pentru instruire.

11. (C) Scrieţi un program care implementează o reţea de perceptroni capabilă să clasifice cifrele 0–9 în categoriile par – impar. Pentru instruirea reţelei, folosiţi o secvenţă de instruire în care caracterele sunt codificate ca matrici binare de 5×3 pixeli. După instruire, utilizaţi reţeaua pentru clasificarea pattern-urilor perturbate.

Capitolul 5

Reţele neurale feedback monostrat

Reţelele neurale pe care le vom studia în acest capitol sunt sisteme dinamice care evoluează în timp, într-un spaţiu discret sau continuu. Evoluţia unei astfel de reţele este, aşa cum vom vedea, disipativ, în sensul că are tendinţa de scădere a energiei. Tranziţia într-o reţea neurală dinamică este către o soluţie asimptotic stabilă care este un minim local al unei funcţii a energiei disipate.

Reţelele discutate în acest capitol se bazează în special pe lucrările lui Hopfield (1984, 1985, 1986).

Să menţionăm câteva concepte fundamentale ale sistemelor dinamice.

Am văzut că un atractor este o stare către care sistemul evoluează în timp pornind de la anumite condiţii iniţiale. Mulţimea acestor condiţii formează *bazinul de atracţie* al atractorului. Dacă atractorul este un punct unic în spaţiul stărilor, atunci el este un *punct fix*. Un atractor poate însă consta şi dintr-o secvenţă de stări, caz în care se numeşte *ciclu limită*.

O reţea de acest tip este capabilă să tolereze anumite distorsiuni ale vectorilor de iniţializare şi să le elimine.

5.1 Fundamentele matematice ale reţelelor Hopfield cu timp discret

Acest tip de reţele au proprietăţi foarte interesante. Utilizând tehnologii microelectronice şi optoelectronice, s-au fabricat microsisteme care se bazează pe aceste reţele.

Conform postulatelor lui Hopfield, o reţea feedback monostrat este de forma celei din figura 5.1.

Această reţea constă din n neuroni cu pragurile T_1,...,T_n. Pentru fiecare neuron se calculează:

$$\text{net}_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j + i_i - T_i, \text{ pentru } i = 1, 2, \ldots, n,$$

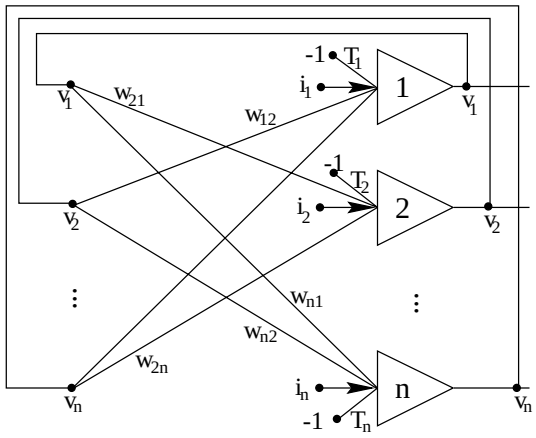


Figura 5.1: Reţea feedback monostrat.

unde i_i este intrarea externă a neuronului i. Vectorial, putem scrie:

$$net_i = \mathbf{w}_i^t + i_i - T_i, \text{ pentru } i = 1, 2, \ldots, n,$$

unde

$$\mathbf{w}_i = \begin{bmatrix} w_{i1} \\ \vdots \\ w_{in} \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

Putem, de asemenea, să rescriem relaţia de mai sus astfel:

$$\mathbf{net} = \mathbf{W}\mathbf{v} + \mathbf{i} - \mathbf{T},$$

unde

$$\mathbf{net} = \begin{bmatrix} net_1 \\ \vdots \\ net_n \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} i_1 \\ \vdots \\ i_n \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} T_1 \\ \vdots \\ T_n \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^t \\ \vdots \\ \mathbf{w}_n^t \end{bmatrix} = \begin{bmatrix} 0 & w_{12} & \ldots & w_{1n} \\ w_{21} & 0 & \ldots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \ldots & 0 \end{bmatrix}.$$

În acest model, matricea $\mathbf{W}$ este simetrică.

Să presupunem pentru moment că funcţia de activare este $\text{sgn}(\cdot)$. Atunci:

$$\begin{cases} v_i \text{ rămâne neschimbat dacă } net_i = 0 \\ v_i \leftarrow -1 \text{ dacă } net_i < 0 \\ v_1 \leftarrow +1 \text{ dacă } net_i > 0 \end{cases} \tag{5.1}$$

Adică:

$$v_i^{k+1} = \text{sgn}(\mathbf{w}_i^t \mathbf{v}^k + i_i - T_i), \quad i = 1, 2, \ldots, n, \quad k = 0, 1, \ldots$$

Regula se aplică *asincron*, adică pentru câte o singură valoare a lui i la un moment dat. Se porneşte de la $\mathbf{v}^0$. Pentru $k = 1$, avem n actualizări. Pentru început se actualizează v_i^1, unde numărul neuronului i este aleator. Pentru ceilalţi neuroni, ieşirile rămân aceleaşi. Se calculează apoi v_j^1, $j \neq i$, j aleator etc., până se completează $\mathbf{v}^1$. Apoi se calculează $\mathbf{v}^2$ etc. Acest algoritm particular de actualizare a ponderilor este cunoscut ca o *recursivitate stohastică asincronă* a reţelelor Hopfield. Trebuie să facem distincţie între algoritmul asincron şi cel sincron (paralel) descris de formula:

$$\mathbf{v}^{k+1} = \Gamma[\mathbf{W}\mathbf{v}^k + \mathbf{i} - \mathbf{T}], \quad k = 0, 1, \ldots$$

Conform acestei formule, toţi neuronii îşi pot modifica simultan ieşirile. În algoritmul asincron, după ce se calculează v_i^{k+1}, această valoare ia locul lui v_i^k participând la calculul celorlalte componente din $\mathbf{v}^{k+1}$. Procesul continuă până se calculează toate valorile din $\mathbf{v}^{k+1}$. Pentru fiecare $k = 1, 2, \ldots$, avem n actualizări aleatoare individuale, la nivel de neuron. Algoritmul se opreşte când $\mathbf{v}^{k+1} = \mathbf{v}^k$.

Reprezentând vectorul de ieşire în $\{-1, 1\}^n$, acesta se mişcă dintr-un vârf al cubului n-dimensional *într-un vârf adiacent*, până când se stabilizează într-unul din vârfuri. Poziţia finală a lui $\mathbf{v}^k$, pentru $k \to \infty$, este determinată de ponderile iniţiale, pragurile iniţiale, intrări, $\mathbf{v}^0$, dar şi de ordinea tranziţiilor.

Următoarele probleme sunt deschise: dacă sistemul are atractori şi dacă sistemul se stabilizează.

Pentru a evalua proprietatea de stabilitate, vom defini *funcţia energiei computaţionale*:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \mathbf{T}^t\mathbf{v}$$

care este o formă pătratică. Adică:

$$E = -\frac{1}{2}\sum_{\substack{i=1\\ i\neq j}}^{n}\sum_{j=1}^{n} w_{ij}v_iv_j - \sum_{i=1}^{n} i_iv_i + \sum_{i=1}^{n} T_iv_i.$$

Fie $v_i^{k+1} - v_i^k = \Delta v_i$, adică presupunem că nodul i s-a modificat la momentul k. Modificarea este asincronă, deci se referă doar la neuronul i. Calculăm

$$\nabla E = -\mathbf{W}\mathbf{v} - \mathbf{i}^t + \mathbf{T}^t$$

în care am ţinut cont că $\mathbf{W}$ este simetrică. Incrementul energiei este

$$\Delta E = \nabla E^t \Delta\mathbf{v}$$

unde

$$\Delta\mathbf{v} = \begin{bmatrix} 0 \\ \vdots \\ \Delta v_i \\ \vdots \\ 0 \end{bmatrix}.$$

Atunci:

$$\begin{aligned}\Delta E &= (-\mathbf{w}_i^t\mathbf{v} - i_i + T_i)\Delta v_i \\ \Delta E &= -\left(\sum_{j=1}^{n} w_{ij}v_j + i_i - T_i\right)\Delta v_i, \text{ pentru } i \neq j \\ \Delta E &= -net_i\Delta v_i\end{aligned}$$

Pentru funcţia de activare $\text{sgn}(\cdot)$, din relaţia 5.1 avem:

$$\begin{aligned}\text{dacă } net_i < 0 &\Rightarrow \Delta v_i \leq 0 \\ \text{dacă } net_i > 0 &\Rightarrow \Delta v_i \geq 0 \\ \text{dacă } net_1 = 0 &\Rightarrow \Delta v_i = 0.\end{aligned}$$

Adică, avem mereu $net_i\Delta v_i \geq 0$ şi $\Delta E \leq 0$. Aceasta înseamnă că energia reţelei poate doar să scadă sau să rămână constantă. Presupunând că $net_i \neq 0$, atunci $\Delta E = 0$ dacă şi numai dacă $\Delta v_i = 0$. Cu alte cuvinte, E scade strict dacă şi numai dacă $\Delta v_i \neq 0$ şi E rămâne constantă dacă şi numai dacă $\Delta v_i = 0$. Dacă $net_i = 0$, atunci $\Delta v_i = 0$, deci $\Delta E = 0$.

Pentru o stare stabilă, adică pentru un atractor, în cazul unei scheme asincrone avem:

$$\mathbf{v}^{k+1} = \lim_{k\to\infty} \text{sgn}(\mathbf{W}\mathbf{v}^k + \mathbf{i} - \mathbf{T}).$$

Am arătat că energia este necrescătoare. Ea este, în mod evident, şi mărginită, ceea ce înseamnă că E îşi atinge minimul.

Dacă algoritmul de actualizare a ponderilor este sincron, se poate întâmpla ca algoritmul să cicleze între două pattern-uri complementare. Acest lucru este ilustrat de exemplul de mai jos.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \quad \mathbf{v}^0 = \begin{bmatrix} -1 & -1 \end{bmatrix}^t \\ \mathbf{v}^1 &= \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} \text{sgn}(1) \\ \text{sgn}(1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \mathbf{v}^2 &= \begin{bmatrix} -1 \\ -1 \end{bmatrix} \text{ etc.}\end{aligned}$$

Vom vedea că pattern-urile complementare corespund unor nivele identice ale energiei.

5.2 Fundamentele matematice ale reţelelor Hopfield cu timp continuu

Reţelele feedback monostrat cu timp continuu se numesc şi *reţele de tip gradient*. O reţea de tip gradient converge către unul dintre minimele stabile din spaţiul

stărilor. Evoluţia sistemului este în direcţia generală a gradientului negativ al funcţiei de energie. Căutarea unui minim energetic corespunde căutării unei soluţii pentru o problemă de optimizare. Reţelele de tip gradient sunt exemple de sisteme neliniare, dinamice şi asimptotic stabile.

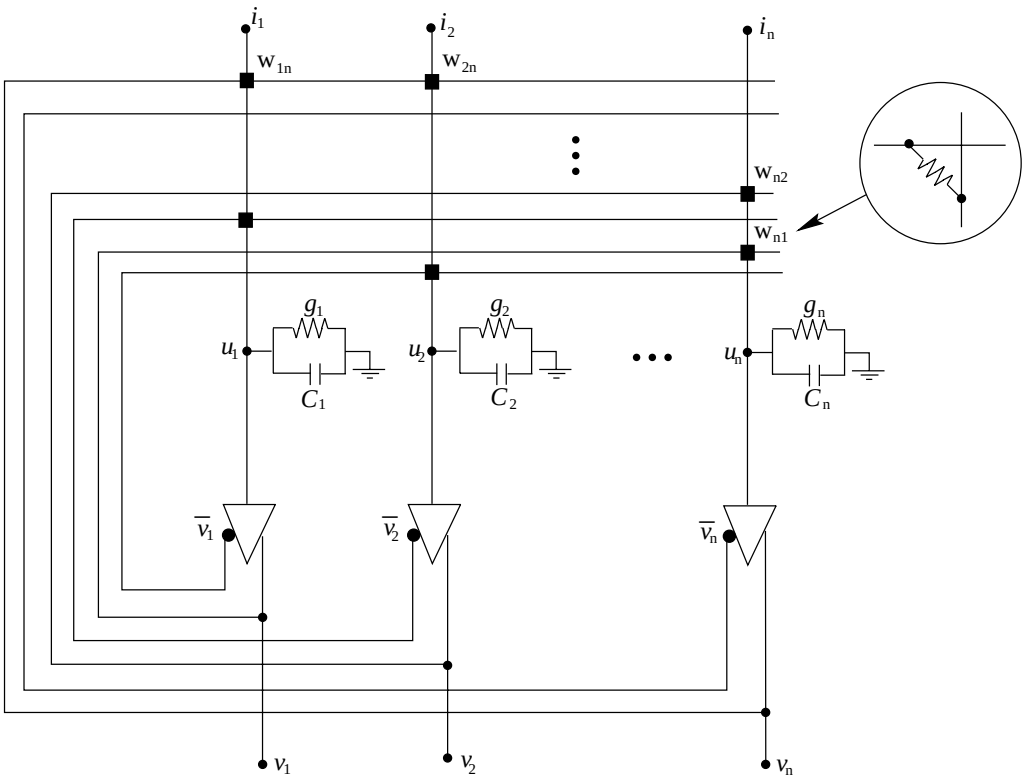


Figura 5.2: Reţea neurală de tip gradient care foloseşte componente electrice.

Vom exemplifica un model de reţea neurală de tip gradient prin folosirea unor componente electrice (fig. 5.2). Pentru a păstra notaţia originală a lui Hopfield, tensiunile u_i sunt de fapt net_i. Ponderile w_{ij} sunt conductanţe. Ieşirile inversate $\bar{v}_i$ sunt folosite pentru a modela conductanţe negative, adică inhibitorii. Avem:

$$w_{ij} = w_{ji} \text{ şi } w_{ii} = 0, \text{ pentru } i, j = 1, \ldots, n.$$

Neuronii pot fi interpretaţi ca amplificatori: curentul prin conductanţa g_i este amplificat de neuronul i (fig. 5.3). $C_i, i = 1, \ldots, n$ sunt capacităţi.

Folosind acest model de neuron, putem scrie legea lui Kirchhoff:

$$i_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - u_i \left(\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} + g_i \right) = C_i \frac{du_i}{dt}. \tag{5.2}$$

Partea stângă a acestei ecuaţii reprezintă curentul total care intră în condensatorul de capacitate C_i. Notăm conductanţa totală care este conectată la in-

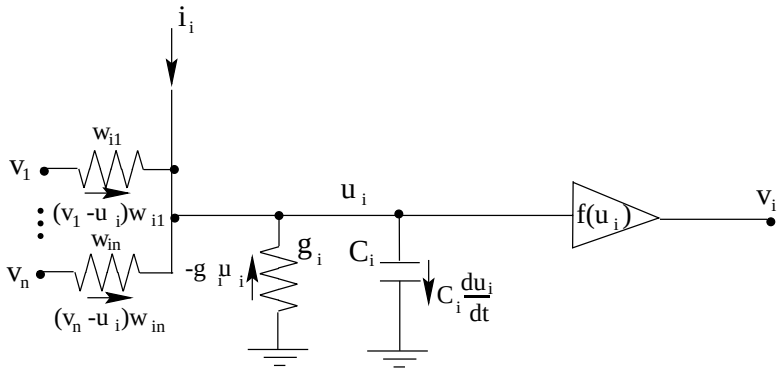


Figura 5.3: Modelul unui neuron.

trarea neuronului i cu

$$G_i = \sum_{j=1}^{n} w_{ij} + g_i.$$

Atunci, ecuaţia 5.2 se poate scrie:

$$i_i + \sum_{j=1}^{n} w_{ij} v_j - u_i G_i = C_i \frac{du_i}{dt}.$$

Fie matricile:

$$\mathbf{C} = \text{diag}\left[\begin{array}{cccc} C_1 & C_2 & \dots & C_n \end{array}\right], \quad \mathbf{G} = \text{diag}\left[\begin{array}{cccc} G_1 & G_2 & \dots & G_n \end{array}\right],$$

$$\mathbf{u}(t) = \left[\begin{array}{c} u_1(t) \\ \vdots \\ u_n(t) \end{array}\right], \quad \mathbf{v}(t) = \left[\begin{array}{c} v_1(t) \\ \vdots \\ v_n(t) \end{array}\right], \quad \mathbf{i} = \left[\begin{array}{c} i_1 \\ \vdots \\ i_n \end{array}\right],$$

unde $\mathbf{u}(t)$ este vectorul stărilor, iar $\mathbf{v}(t)$ reprezintă vectorul ieşirilor. Avem:

$$\begin{cases} C\frac{d\mathbf{u}(t)}{dt} = \mathbf{W}\mathbf{v}(t) - \mathbf{G}\mathbf{u}(t) + \mathbf{i} & \textit{ecuaţia stărilor} \\ \mathbf{v}(t) = f(\mathbf{u}(t)) & \textit{ecuaţia ieşirilor} \end{cases}$$

Să studiem stabilitatea sistemului descris de aceste ecuaţii. Funcţia energiei care ne permite să analizăm modelul continuu este:

$$E(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \sum_{i=1}^{n} G_i \int_0^{v_i} f_i^{-1}(z)dz,$$

unde $f_i^{-1}(z)$ este inversa funcţiei de activare f_i. Această formulă conţine termenul $\mathbf{T}^t\mathbf{v}$ din funcţia energiei discrete

$$-\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \mathbf{T}^t\mathbf{v}$$

în termenul al doilea, iar ultimul termen este specific cazului continuu şi dispare pentru $\lambda \to \infty$. Avem:

$$\frac{d}{dv_i}\left(G_i \int_0^{v_i} f_i^{-1}(z)dz\right) = G_i u_i$$

$$\nabla E(\mathbf{v}) = -\mathbf{W}\mathbf{v} - \mathbf{i} + \mathbf{G}\mathbf{u}.$$

Atunci:

$$\begin{aligned}\frac{dE[\mathbf{v}(t)]}{dt} &= \nabla E^t(\mathbf{v})\,\dot{v} = (-\mathbf{W}\mathbf{v} - \mathbf{i} + \mathbf{G}\mathbf{u})^t\dot{v} = -\left(C\frac{d\mathbf{u}}{dt}\right)^t \frac{d\mathbf{v}}{dt} \\ &= -\sum_i C_i \frac{df_i^{-1}(v_i)}{dv_i}\left(\frac{dv_i}{dt}\right)^2\end{aligned}$$

deoarece

$$\frac{du_i}{dt} = \frac{df_i^{-1}(v_i)}{dv_i}\,\frac{dv_i}{dt}.$$

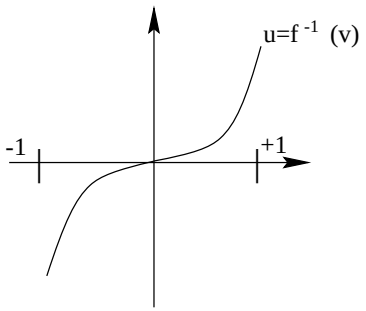


Figura 5.4: Inversa funcţiei de activare.

În figura 5.4, funcţia f_i^{-1} este monoton crescătoare, deci $\frac{df_i^{-1}(v_i)}{dv_i} > 0$. Atunci:

$$\frac{dE}{dt} < 0 \text{ pentru } \frac{d\mathbf{v}}{dt} \neq 0 \text{ şi}$$

$$\frac{dE}{dt} > 0 \text{ pentru } \frac{d\mathbf{v}}{dt} = 0.$$

Am găsit o funcţie Liapunov a sistemului, deci sistemul este asimptotic stabil. Funcţia E este mărginită în spaţiul ieşirilor, deci tinde către un minim.

Modificările în timp nu sunt exact pe direcţia gradientului funcţiei de energie (adică pe direcţia scăderii cât mai rapide a energiei).

Reţeaua de tip gradient, în cazul în care spaţiul ieşirilor este mărginit, converge către unul dintre minimele lui $E(\mathbf{v})$, deci către o stare stabilă. Acest minim este în interiorul hipercubului ieşirilor. Dacă $\lambda \to \infty$, atunci minimul este într-un vârf al hipercubului.

Faptul că sistemul este asimptotic stabil în spaţiul ieşirilor este echivalent cu faptul că sistemul este asimptotic stabil în spaţiul stărilor $\mathbf{u}$. Aceasta, deoarece ecuaţia

$$\mathbf{v}(t) = f(\mathbf{u}(t))$$

este monotonă.

5.3 Aplicaţie: problema comis-voiajorului

Vom da o soluţie a problemei comis-voiajorului pentru n oraşe folosind reţele de tip gradient (Hopfield, Tank, 1985). Problema este NP-completă.

Reprezentând problema sub forma unui graf cu n vârfuri, trebuie să găsim ciclul de lungime minimă care trece prin toate vârfurile. Numărul de astfel de cicluri care trebuie evaluate este $\frac{(n-1)!}{2}$.

Se poate elabora o reţea neurală de n^2 neuroni unipolari de tip continuu care să rezolve această problemă. Aranjăm neuronii într-un tablou de $n \times n$ elemente. Tabloul este reprezentat în figura 5.5, în care *Poz.* reprezintă poziţia în ciclu.

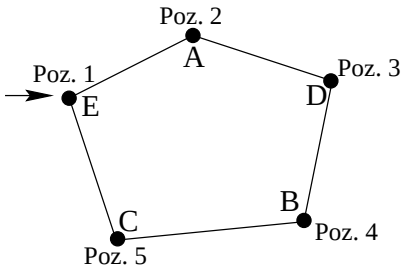


Figura 5.5: Reprezentarea cu ajutorul unui graf a problemei comis-voiajorului.

Fie oraşele A, B,..., E şi fie acest ciclu. Îl reprezentăm prin tabelul 5.1, unde v_{xi} este ieşirea neuronului corespunzător oraşului x şi poziţiei i în ciclu.

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

Tabelul 5.1: Reprezentarea ciclului din figura 5.5.

Pentru un ciclu care vizitează toate vârfurile, tabloul trebuie să aibă un singur 1 pe fiecare coloană şi un singur 1 pe fiecare linie. Poziţionăm neuronii în acest tablou de $n \times n$ elemente, în fiecare element punând un neuron. Vom avea exact n

neuroni activi pentru soluţie. Presupunem pentru λ o valoare mare, deci energia este trunchiată:

$$E = -\frac{1}{2}\sum_{x,i=1}^{n}\sum_{y,j=1}^{n} w_{xi,yj}v_{xi}v_{yj} - \sum_{x,i=1}^{n} i_{xi}v_{xi},$$

unde $w_{xi,yj}$ este ponderea de la neuronul yj la neuronul xi. Observăm că domeniul vectorilor $\mathbf{v}$ unde se minimizează funcţia E este format din cele 2^{n^2} vârfuri ale hipercubului n^2-dimensional $[0,1]$.

Scopul optimizării este selectarea circuitelor minime. Trebuie să avem obligatoriu câte un neuron activ pe linie şi pe coloană. În plus, trebuie să eliminăm soluţiile triviale, pentru care avem, în tablou, linii sau coloane nule. Să exprimăm aceste condiţii de optimizare folosind patru funcţii de energie, fiecare îndeplinind un rol separat.

$$E_1 = A\sum_{x=1}^{n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}} v_{xi}v_{xj}.$$

Produsul $v_{xi}v_{xj}$ se referă la acelaşi oraş, x. Relaţia

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} v_{xi}v_{xj} = 0$$

se îndeplineşte dacă cel mult un $v_{xi} = 1$ iar restul sunt 0, adică dacă oraşul x apare cel mult o singură dată pe circuit. $E_1 = 0$ înseamnă că fiecare oraş este vizitat cel mult o dată. Dacă $E_1 > 0$, atunci există oraşe vizitate de mai multe ori.

$$E_2 = B\sum_{i=1}^{n}\sum_{x=1}^{n}\sum_{\substack{y=1\\ y\neq x}}^{n} v_{xi}v_{yi}.$$

$E_2 = 0$ înseamnă că fiecare poziţie din ciclu are cel mult un singur oraş asociat ei. Dacă $E_2 > 0$, atunci există mai multe oraşe vizitate simultan. Observăm că E_1 şi E_2 sunt, de asemenea, nule pentru soluţii triviale în care avem linii sau coloane în tablou care nu conţin nici un 1. De aceea, mai introducem:

$$E_3 = C\left(\sum_{x=1}^{n}\sum_{i=1}^{n} v_{xi} - n\right)^2$$

pentru a ne asigura că nu avem o soluţie trivială, deci o soluţie cu mai puţin de n de 1 în tablou. $E_3 = 0$ înseamnă că avem exact n de 1 în tablou. Deci $E_1 + E_2 + E_3$ are minimul 0 pentru toate matricile care au exact un 1 pe fiecare linie şi pe fiecare coloană. Dorim să minimizăm lungimea ciclului. Notăm cu d_{xy} distanţa dintre x şi y şi $d_{xy} = d_{yx}$. Lungimea ciclului este:

$$\frac{1}{2}\sum_{x=1}^{n}\sum_{y=1}^{n}\sum_{i=1}^{n} d_{xy}v_{xi}(v_{y,i+1} + v_{y,i-1}),$$

dar vom folosi următoarea funcţie de energie:

$$E_4 = D \sum_{x=1}^{n} \sum_{\substack{y=1 \\ y \neq x}}^{n} \sum_{i=1}^{n} d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1}),$$

unde $v_{x,n+1} = v_{x1}$ şi $v_{x0} = v_{xn}$.

Dorim să minimizăm $E_1 + E_2 + E_3 + E_4$. Prin identificarea termenilor acestei sume cu cei din E, obţinem:

$$w_{xi,yj} = -2A\delta_{xy}(1 - \delta_{ij}) - 2B\delta_{ij}(1 - \delta_{xy}) - 2C - 2Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1}),$$

unde δ_{ij} este simbolul lui Kronecker ($\delta_{ij} = 0$ pentru $i \neq j$, $\delta_{ii} = 1$). Constantele pozitive A, B, C, D sunt selectate euristic pentru a construi o sumă convenabilă a termenilor E_1,..., E_4. Ele dau ponderea fiecăruia dintre aceşti termeni. Obţinem, de asemenea:

$$i_{xi} = 2Cn.$$

Am generat reţeaua. Mai trebuie acum să găsim soluţia prin simularea funcţionării ei.

Din formula generală a ecuaţiei stărilor:

$$C_i \left(\frac{du_i}{dt} \right) = i_i + \sum_{j=1}^{n} w_{ij} v_j - u_i G_i$$

obţinem:

$$\begin{aligned} \frac{du_{xi}}{dt} &= -\frac{u_{xi}}{\tau} - 2A \sum_{\substack{j=1 \\ j \neq i}}^{n} v_{xj} - 2B \sum_{\substack{y=1 \\ y \neq x}}^{n} v_{yi} \\ &\quad -2C \left(\sum_{x=1}^{n} \sum_{j=1}^{n} v_{xj} - n \right) - 2D \sum_{\substack{y=1 \\ y \neq x}}^{n} d_{xy} (v_{y,i+1} + v_{y,i-1}), \end{aligned}$$

unde τ este o constantă a timpului.

S-au luat $\tau = 1$, $\lambda = 50$, $A = B = D = 250$, $C = 100$, $10 \leq n \leq 30$ şi s-a rezolvat numeric acest sistem de ecuaţii. Nu se ajunge neapărat la soluţia optimă. Pentru $n = 10$, 50% din teste au produs cele mai scurte cicluri din 181440 de soluţii posibile. Pentru valori mari ale lui n apar, însă, dificultăţi.

Aceasta duce la întrebarea fundamentală dacă reţelele neurale sunt mai adecvate decât calculatoarele clasice pentru rezolvarea problemelor nedeterminist polinomiale.

S-a demonstrat (Bruck, Goodman, 1988) că nu există nici o reţea de mărime polinomială în n care să rezolve problema comis-voiajorului cu o precizie dorită. Aceasta, atâta timp cât NP $\neq$ P!

Problema comis-voiajorului ilustrează utilizarea reţelelor Hopfield în probleme de optimizare. S-au obţinut rezultate în probleme de alocare a resurselor, planificarea lucrărilor, optimizarea traficului, optimizarea conexiunilor în circuitele integrate (1990), programare liniară şi neliniară.

În concluzie, putem spune că una dintre limitările reţelelor Hopfield este faptul că ajungem de multe ori la un minim local şi nu global. Aceasta se datorează formei complexe a funcţiei $E(\mathbf{v})$.

Una din dificultăţi este şi traducerea problemei de optimizare într-o problemă de minimizare a energiei. Singurele forme ale funcţiei de energie care se cunosc sunt cele date de Hopfield. Dacă λ este mare, se poate lua energia trunchiată. Dacă intrările externe sunt nule, dispare şi al doilea termen. Schema de rezolvare a unei probleme de optimizare folosind reţele feed-back monostrat este cea din figura 5.6.

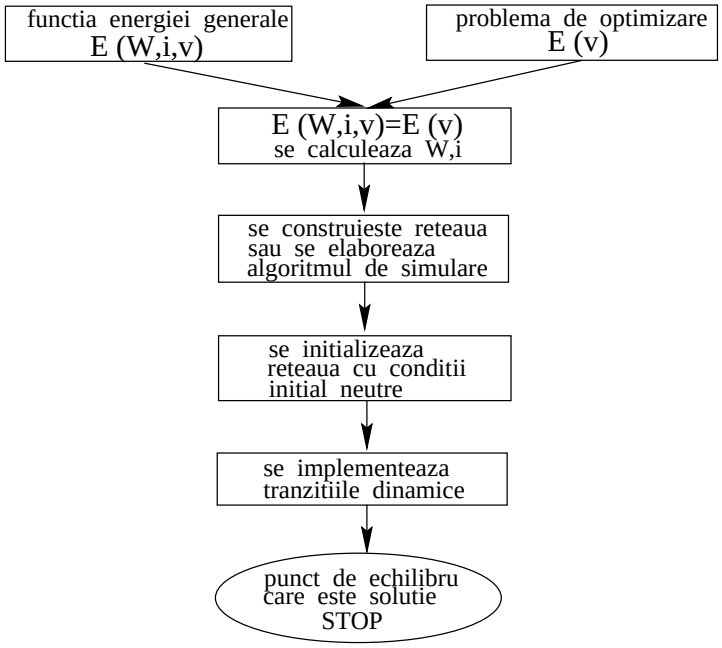


Figura 5.6: Modul de rezolvare a unei probleme de optimizare folosind reţele feedback monostrat.

5.4 Exemple

Reţea Hopfield cu timp discret

Fie:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Presupunem pragurile şi intrările externe zero. Calculăm:

$$\begin{aligned} E(\mathbf{v}) &= -\frac{1}{2}\begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}\mathbf{W}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \\ &= -v_1(v_2+v_3-v_4)-v_2(v_3-v_4)+v_3v_4. \end{aligned}$$

Calculând expresia pentru toate vârfurile hipercubului: $\begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^t$,..., $\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^t$ se ajunge la nivelele de energie -6, 0, 2. Tranziţiile au loc pe muchiile hipercubului, de la un vârf la un alt vârf cu energie mai scăzută. Minimul este atins în vârfurile $\begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^t$ şi $\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^t$, unde avem stări stabile. Tranziţiile au loc asincron. De aceea, o tranziţie este doar de-a lungul unei singure muchii (se poate modifica cel mult o componentă).

Dacă tranziţiile au loc sincron, atunci ele nu minimizează energia şi nu tind către o stare stabilă. De exemplu, pornind din $\mathbf{v}^0 = \begin{bmatrix} 1 & -1 & 1 & 1 \end{bmatrix}^t$, ajungem la:

$$\begin{aligned} \mathbf{v}^1 &= \Gamma\left[\mathbf{W}\mathbf{v}^0\right] = \Gamma\begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{sgn}(-1) \\ \mathrm{sgn}(1) \\ \mathrm{sgn}(-1) \\ \mathrm{sgn}(-1) \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix}. \end{aligned}$$

Tranziţia nu mai este de-a lungul unei muchii. Se calculează că

$$\mathbf{v}^2 = \mathbf{v}^0,$$

deci oscilează între $\mathbf{v}^0$ şi $\mathbf{v}^1$. În $\mathbf{v}^0$ şi $\mathbf{v}^1$ energia este 2. Spre deosebire de tranziţiile sincrone, tranziţiile asincrone duc întotdeauna la unul dintre minimele energetice. Dacă la tranziţiile asincrone pornim de la $\begin{bmatrix} -1 & 1 & 1 & 1 \end{bmatrix}$ şi scădem energia, atunci evoluţia se poate desfăşura ca în figura 5.7.

Minimul la care ajungem depinde de ordinea de actualizare a componentelor.

Reţea de tip gradient

Vom elabora un convertor A/D (Tank, Hopfield, 1986) pe 2 biţi. Presupunem că $f(u_i)$ este continuă unipolară.

Se poate arăta că o funcţie energiei totale aleasă convenabil poate fi:

$$E = \frac{1}{2}\left(x - \sum_{i=0}^{1} v_i 2^i\right)^2 - \frac{1}{2}\sum_{i=0}^{1} 2^{2i} v_i(v_i-1).$$

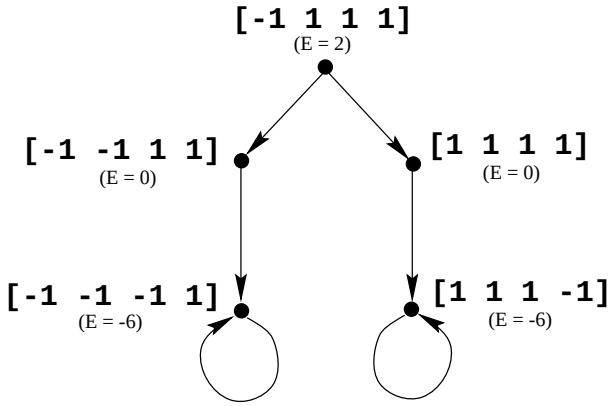


Figura 5.7: Variante de evoluţie a stărilor pentru o serie de tranziţii asincrone.

Dorim să minimizăm pe E, ceea ce implică minimizarea lui $\frac{1}{2}\left(x - \sum_{i=0}^{1} v_i 2^i\right)^2$ care este eroarea de conversie A/D, unde $v_0 + 2v_1$ este valoarea zecimală a vectorului binar $\left[\begin{array}{cc} v_1 & v_0 \end{array}\right]^t$ şi corespunde valorii analogice x. Calculând, ajungem la:

$$E = \frac{1}{2}x^2 + \frac{1}{2}\sum_{i=0}^{1}\sum_{\substack{j=0 \\ j\neq i}}^{1} 2^{i+j} v_i v_j + \sum_{i=0}^{1}\left(2^{2i-1} - 2^i x\right) v_i.$$

Pentru x fixat, $\frac{1}{2}x^2$ este o constantă, deci irelevantă atunci când minimizăm pe E. Expresia energiei în reţeaua Hopfield este:

$$-\frac{1}{2}\sum_{i=0}^{1}\sum_{j=0}^{1} w_{ij} v_i v_j \sum_{i=0}^{1} i_i v_i.$$

Ultimul termen dispare deoarece minimele sunt în vârfurile hipercubului. Prin identificarea termenilor din cele două expresii, ajungem la:

$$\begin{aligned} w_{01} &= w_{10} = -2 \\ i_0 &= x - \frac{1}{2} \\ i_1 &= 2x - 2 \end{aligned}$$

Convertorul rezultat este un caz special al unei reţele de tip gradient (fig. 5.8).

Avem:

$$\begin{aligned} C_0 \frac{du_0}{dt} &= x - \frac{1}{2} + (-2)v_1 - u_0(-2 + g_0) \\ C_1 \frac{du_1}{dt} &= 2x - 2 + (-2)v_0 - u_1(-2 + g_1) \end{aligned}$$

care sunt ecuaţiile stărilor. De exemplu, putem presupune că folosim o tensiune de referintă având valoarea de -1V (fig. 5.9).

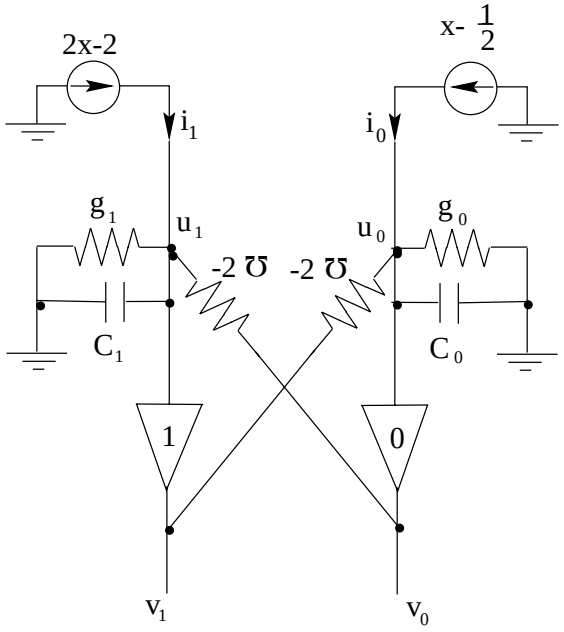


Figura 5.8: Convertor A/D pe doi biţi implementat printr-o reţea de tip gradient.

Ecuaţiile stărilor devin:

$$\begin{aligned} C_0\frac{du_0}{dt} &= x - \frac{1}{2} + 2\bar{v}_1 - u_0(2+g_0) \\ C_1\frac{du_1}{dt} &= 2x - 2 + 2\bar{v}_0 - u_1(2+g_1). \end{aligned}$$

Condiţia $w_{ij} \geq 0$ asigură realizabilitatea reţelei folosind rezistenţe de valoare $R_{ij} = \frac{1}{w_{ij}}$. Înlocuind ponderile şi valorile curenţilor externi în expresia energiei, obţinem pentru valori mari ale lui λ:

$$E = 2v_0v_1 + \frac{v_0}{2} + 2v_1 - x(v_0 + 2v_1). \tag{5.3}$$

Reprezentând grafic funcţia E, obţinem:

- Pentru $x = 0$, punctul staţionar este punct de minim: $v_0 = v_1 = 0$

- Pentru $x = 1$, punctul staţionar este punct de minim: $v_0 = 1$, $v_1 = 0$

- Pentru $x = 1,25$ punctele staţionare sunt: punctul de minim dorit $v_0 = 1$, $v_1 = 0$, dar şi punctul de minim nedorit $v_0 = 0$, $v_1 = 1$, precum şi un punct şa, undeva în interiorul pătratului: $0 < v_0 < 1$, $0 < v_1 < 1$.

În loc să rezolvăm sistemul de ecuaţii diferenţiale, oare nu putem căuta minimele lui E?

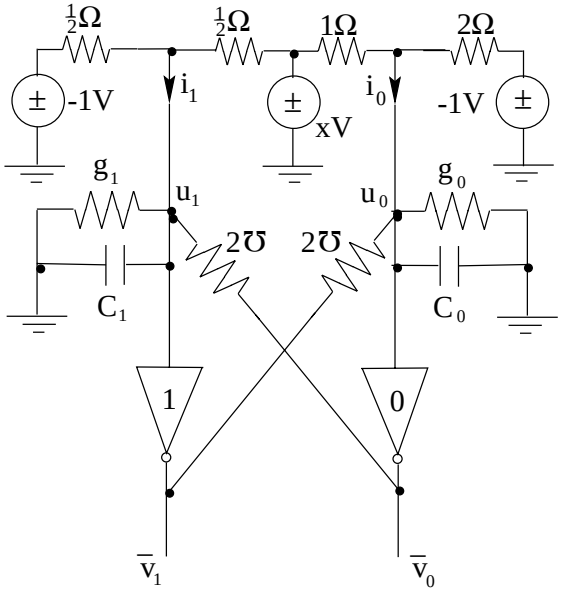


Figura 5.9: Convertor A/D pe doi biţi implementat printr-o reţea de tip gradient şi care foloseşte o tensiune de referinţă de 1V.

Din relaţia 5.3 obţinem:

$$\begin{aligned} \nabla E(\mathbf{v}) &= \begin{bmatrix} 2v_1 + \frac{1}{2} - x \\ 2v_0 + 2 - 2x \end{bmatrix} \\ \mathbf{H} = \nabla^2 E(\mathbf{v}) &= \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}. \end{aligned}$$

Observăm :

$$\begin{aligned} \det H_{11} &= 0 \\ \det H_{22} &= \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} = -4. \end{aligned}$$

Matricea H nu este pozitiv sau negativ definită. E nu are deci minim sau maxim, dacă nu considerăm alte restricţii. Impunem restricţia ca spaţiul de ieşire să fie pătratul $\begin{pmatrix} 0 & 1 \end{pmatrix}^2$. Atunci, minimele lui E sunt situate în vârfurile pătratului, dacă aceste minime există. Se poate demonstra acest lucru pentru $\lambda \to \infty$. Dacă v^* este o soluţie pentru $\nabla E(\mathbf{v}) = 0$ şi se află în interiorul pătratului, atunci este un punct şa. Rezultă că pentru a găsi soluţia avem două posibilităţi:

1. Rezolvarea numerică a sistemului de ecuaţii diferenţiale
2. Simularea reţelei printr-un program de simulare a circuitelor electronice, de exemplu SPICE.

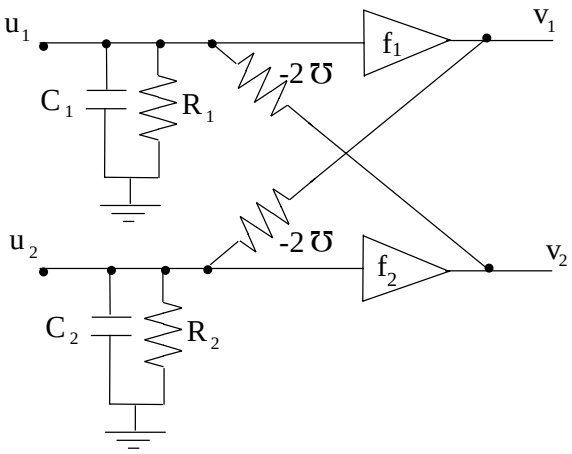


Figura 5.10: Reţeaua neurală propusă pentru exerciţiul 2.

5.5 Exerciţii

1. Simulaţi în SPICE convertorul A/D. Generalizaţi-l pentru patru biţi.
2. Pentru reţeaua neurală din figura 5.10, cu λ mare, găsiţi:
   (a) Ecuaţiile stărilor;
   (b) Ponderile (conductanţele) $\mathbf{W}$;
   (c) Funcţia energiei $E(\mathbf{v})$ trunchiată;
   (d) $\nabla E(\mathbf{v})$.

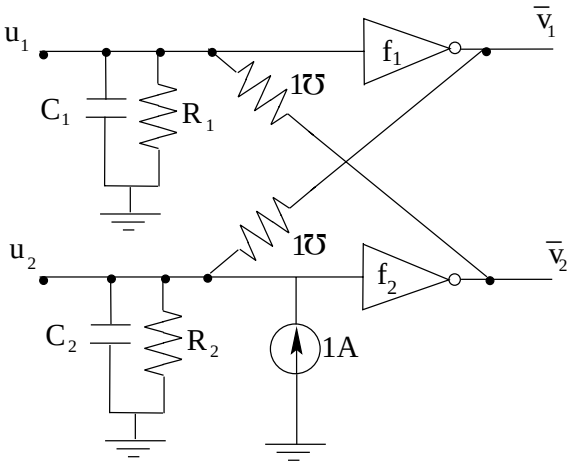


Figura 5.11: Reţeaua neurală propusă pentru exerciţiul 3.

3. Pentru reţeaua neurală din figura 5.11, cu λ mare, găsiţi:

   (a) Ecuaţiile stărilor;

   (b) Ponderile (conductanţele) $\mathbf{W}$;

   (c) Vectorul $\mathbf{i}$ al intrărilor externe;

   (d) Funcţia energiei $E(\mathbf{v})$ trunchiată;

   (e) $\nabla E(\mathbf{v})$.

4. Fie o reţea neurală cu trei neuroni, a cărei funcţie de energie trunchiată este:

$$E(\mathbf{v}) = v_1^2 - v_2^2 + v_3^2 - 2v_1v_3 - v_1v_3 + 4v_1 + 12.$$

   Găsiţi:

   (a) $\nabla E(\mathbf{v})$;

   (b) $\nabla^2 E(\mathbf{v})$;

   (c) Minimele, maximele şi punctele şa ale lui $E(\mathbf{v})$, atunci când nu avem alte restricţii.

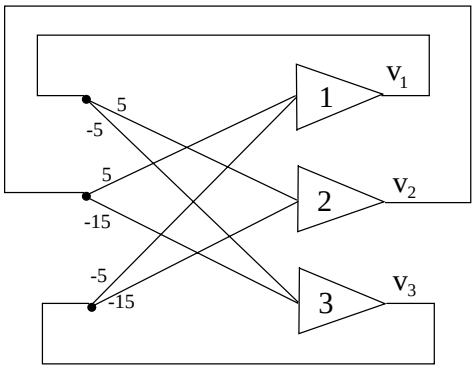


Figura 5.12: Reţeaua neurală propusă pentru exerciţiul 5.

5. Fie reţeaua neurală din figura 5.12, cu λ mare. Evaluând funcţia de energie trunchiată, găsiţi minimele, maximele şi punctele şa în $[\ -1 \quad 1\ ]^3$. Calculaţi apoi valorile energiei în toate vârfurile cubului.

6. O reţea neurală este descrisă de ecuaţiile:

$$\begin{cases} \frac{du_i}{dt} = \frac{1}{C_i}\left(\sum_{j=1}^{n} w_{ij}v_j - u_iG_i + i_i\right) & \text{pentru } i = 1, 2, \ldots, n \\ v_i = f(u_i) \end{cases}.$$

   Rezolvaţi numeric acest sistem.

Indicaţie: Alegem, de exemplu, o metodă foarte simplă, cea a lui Euler:

$$\begin{cases} u_i^{k+1} &= u_i^k + h\frac{1}{C_i}\left(\sum_{j=1}^n w_{ij}v_j^k - u_i^k G_i + i_i\right) \text{ pentru } i = 1, \ldots, n, \\ & \text{unde } h = t^{k+1} - t^k \\ v_i^k &= f(u_i^k) \end{cases}$$

7. Implementaţi metoda din exerciţiul 6 pentru problema convertorului A/D pe doi biţi şi găsiţi $v(t)$ (ieşirea stabilă) pentru:

$$C_0 = C_1 = 1F, \quad g_0 = g_1 = 1\Omega^{-1}, \quad x = 0,$$

$$v_i = f(u_i) = \frac{1}{1 + e^{-10u_i}}, \quad i = 0, 1$$

Testaţi programul pentru:

$$\begin{aligned} \mathbf{v}^0 &= [\ 0,25 \quad 0,25\ ]^t \\ \mathbf{v}^0 &= [\ 0,5 \quad 0,5\ ]^t \\ \mathbf{v}^0 &= [\ 0,75 \quad 0,75\ ]^t. \end{aligned}$$

8. (C) Proiectaţi o memorie Hopfield recurentă autoasociativă care să memoreze ca prototipuri caracterele A, I şi O, codificate ca matrici binare de 7×5 pixeli. Implementaţi algoritmul de regăsire şi simulaţi convergenţa asincronă a memoriei către fiecare din prototipurile memorate, folosind ca pattern-uri cheie caracterele A, I şi O, perturbate.

Capitolul 6

Memorii asociative

În capitolul precedent, ne-am concentrat în special pe rezolvarea unor probleme de optimizare. În acest capitol, vom interpreta evoluţia unui sistem dinamic ca o mişcare a unui pattern de intrare către un alt pattern care este memorat şi se numeşte *prototip* sau *memorie stocată*. Reţelele neurale din această clasă se numesc *memorii asociative*.

O memorie asociativă eficientă poate stoca un mare număr de pattern-uri. Pe parcursul apelului, memoria este excitată cu un *pattern cheie* (numit şi *argument al căutării*) care conţine o porţiune de informaţie despre un anumit pattern dintr-o mulţime de pattern-uri memorate. Acest prototip anume poate fi regăsit prin asocierea pattern-ului cheie şi a informaţiei memorate.

S-au elaborat mai multe modele de memorii asociative. Ne vom concentra asupra reţelelor feedback din capitolul precedent, dar vom discuta şi arhitecturi feedforward de memorie.

În general, memoriile asociative achiziţionează informaţie *a priori*. Scrierea în memorie provoacă modificări ale conexiunilor dintre neuroni. Nu există adrese, toată informaţia din memorie este distribuită spaţial în reţea.

Care sunt cerinţele pentru o astfel de memorie? Ea trebuie să aibă o capacitate mare de stocare a prototipurilor. Ea trebuie să fie robustă, astfel încât o distrugere locală a structurii să nu provoace căderea întregului sistem. În plus, o astfel de memorie trebuie să fie capabilă de a adăuga/elimina asociaţii, pe măsură ce se modifică cerinţele de stocare.

Memoriile asociative permit de obicei căutarea paralelă. Scopul căutării este de a extrage unul sau toate pattern-urile care se potrivesc cu pattern-ul cheie.

Se presupune că memoria biologică operează conform principiilor unei memorii asociative: nu există locaţii de memorie cu adrese; stocarea este distribuită într-o reţea densă de neuroni interconectaţi.

6.1 Concepte de bază

Diagrama bloc a unei memorii asociative este reprezentată în figura 6.1.

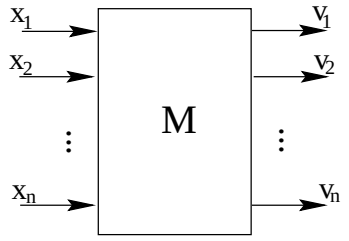


Figura 6.1: Diagrama bloc a unei memorii asociative.

În această figură, $\mathbf{x}$ este vectorul intrărilor, $\mathbf{v}$ este vectorul ieşirilor, $\mathbf{x} \in \Re^n$, $\mathbf{v} \in \Re^m$ şi $\mathbf{v} = M[\mathbf{x}]$, unde M este un operator neliniar matricial specific fiecărui tip de memorie. Structura lui M reflectă o paradigmă specifică. Pentru memorii dinamice, M implică şi variabila timp.

Pentru un model de memorie dat, forma operatorului M este exprimată de obicei în funcţie de vectorii prototip care trebuie stocaţi. Algoritmul care permite calcularea lui M se numeşte *algoritm de înregistrare* sau *stocare*. De obicei, neuronii sunt aşezaţi pe unul sau două straturi.

Maparea $\mathbf{v} = M[\mathbf{x}]$ este numită *regăsire*. Notăm prototipurile cu un indice superior în paranteză.

Să presupunem că în memorie sunt stocaţi anumiţi vectori prototip, astfel încât, dacă se aplică un pattern cheie, ieşirea produsă de memorie şi asociată cheii este răspunsul memoriei.

Presupunând că există p perechi de asocieri stocate:

$$\mathbf{x}^{(i)} \rightarrow \mathbf{v}^{(i)} \quad \text{pentru } i = 1, \ldots, p$$

şi

$$\mathbf{v}^{(i)} \neq \mathbf{x}^{(i)} \quad \text{pentru } i = 1, \ldots, p,$$

atunci reţeaua este o *memorie heteroasociativă*.

Dacă aplicaţia este de forma:

$$\mathbf{x}^{(i)} \rightarrow \mathbf{v}^{(i)}\Big|_{\mathbf{v}^{(i)}=\mathbf{x}^{(i)}} \text{ pentru } i = 1, \ldots, p$$

atunci memoria este *autoasociativă*.

În memoria heteroasociativă, asocierile se fac între mulţimile ordonate de vectori $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\}$ şi $\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(p)}\}$. Memoriile autoasociative asociază vectori din cadrul unei singure mulţimi care este $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\}$. Evident, proiectarea unui vector $\mathbf{x}^{(i)}$ în el însuşi nu are nici o semnificaţie. O aplicaţie mai realistă a unei mapări autoasociative este asocierea unui pattern cheie perturbat cu prototipul său stocat în memorie.

Spre deosebire de memoriile calculatoarelor, care sunt adresabile *prin adresă*, memoriile asociative sunt adresabile *prin conţinut*.

Prin reţele neurale se pot realiza memorii statice (instantanee, nerecurente) şi memorii dinamice (recurente). Memoria din figura 6.2 este una statică realizată

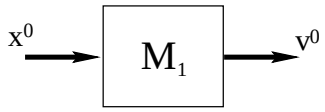


Figura 6.2: Memorie statică realizată printr-o reţea feedforward.

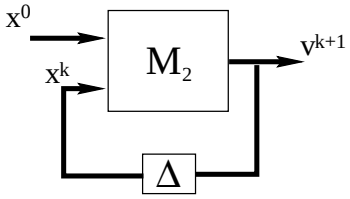


Figura 6.3: Memorie dinamică autoasociativă realizată printr-o reţea recurentă.

printr-o reţea feedforward şi $\mathbf{v}^0 = M_1[\mathbf{x}^0]$ care reprezintă un sistem de ecuaţii algebrice neliniare. În figura 6.3 este o memorie dinamică autoasociativă realizată printr-o reţea recurentă pentru care $\mathbf{v}^{k+1} = M_2[\mathbf{x}^0, \mathbf{v}^k]$, reprezentând un sistem de ecuaţii diferenţiale neliniare.

Un **exemplu de memorie dinamică** este reţeaua Hopfield în care

$$\mathbf{v}^{k+1} = M_2[\mathbf{v}^k],$$

adică în care avem un $\mathbf{v}^0$, dar nu avem intrări externe (fig. 6.4).

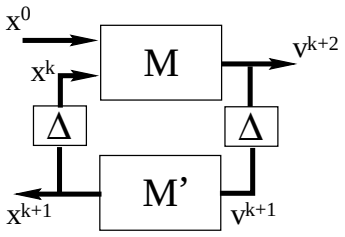


Figura 6.4: Memorie dinamică heteroasociativă.

6.2 Asociatori liniari

Memoriile asociative tradiţionale sunt de tip instantaneu, feedforward. Pentru *memoria asociativă liniară* avem:

$$\mathbf{v} = \mathbf{W}\mathbf{x}$$

unde $\mathbf{x}$ este pattern-ul de intrare, iar $\mathbf{v}$ pattern-ul de ieşire. Nu apar neuroni. Dacă totuşi introducem formal neuroni, atunci avem:

$$\mathbf{v} = M_1[\mathbf{W}\mathbf{x}]$$

unde $M_1[\cdot]$ este un operator matricial liniar unitate. Avem un strat de neuroni de ieşire pentru care

$$v_1 = f(\text{net}_i) = \text{net}_i.$$

Să presupunem că dorim să stocăm în acest asociator liniar p. Se dau perechile de vectori $\{\mathbf{s}^{(i)}, \mathbf{f}^{(i)}\}$, $i = 1, \ldots, p$. Vectorii $\mathbf{s}$ reprezintă stimulii, iar vectorii $\mathbf{f}$ sunt răspunsurile forţate. Avem:

$$\mathbf{s}^{(i)} = \left[s_1^{(i)} \ldots s_n^{(i)}\right]^t$$

$$\mathbf{f}^{(i)} = \left[f_1^{(i)} \ldots f_m^{(i)}\right]^t.$$

Practic, $\mathbf{s}^{(i)}$ pot fi pattern-uri iar $\mathbf{f}^{(i)}$ informaţii asupra apartenenţei lor la o clasă, sau imagini ale lor.

Obiectivul asociatorului liniar este să implementeze maparea

$$\mathbf{v} = \mathbf{W}\mathbf{x}$$

sub forma

$$\mathbf{f}^{(i)} + \eta^{(i)} = \mathbf{W}\mathbf{s}^{(i)}$$

sau, folosind simbolul de mapare

$$\mathbf{s}^{(i)} \to \mathbf{f}^{(i)} + \eta^{(i)} \quad \text{pentru } i = 1, 2, \ldots, p$$

astfel încât vectorul zgomot $\eta^{(i)}$ să fie minimizat.

Problema se poate pune şi astfel: pentru un număr mare de observaţii, să se găsească $\mathbf{W}$ care minimizează $\sum_i \|\eta^{(i)}\|$. Este o problemă tipică de statistică matematică şi nu o vom aborda aici.

Dorim să găsim $\mathbf{W}$ care permite stocarea eficientă a datelor în memorie. Aplicăm regula de învăţare hebbiană pentru a instrui asociatorul:

$$w_{ij}{}' = w_{ij} + f_i s_j \quad \text{pentru } i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n$$

unde vectorii $\mathbf{f}$ şi $\mathbf{s}$ trebuie să fie membrii ai perechii de asociere.

$$\mathbf{W}' = \mathbf{W} + \mathbf{f}\mathbf{s}^t$$

Iniţializând ponderile din $\mathbf{W}$ cu zero, avem:

$$\mathbf{W}' = \mathbf{f}^{(i)}\mathbf{s}^{(i)t}.$$

Acesta este primul pas al instruirii. Avem p perechi de învăţat, deci:

$$\mathbf{W}' = \sum_{i=1}^{p} \mathbf{f}^{(i)}\mathbf{s}^{(i)t}$$

$$\mathbf{W}' = \mathbf{F}\mathbf{S}^t$$

unde $\mathbf{W}'$ este o matrice de corelaţie $m \times n$ şi

$$\mathbf{F} = \left[\begin{array}{cccc} \mathbf{f}^{(1)} & \mathbf{f}^{(2)} & \ldots & \mathbf{f}^{(p)} \end{array} \right]$$

$$\mathbf{S} = \left[\begin{array}{cccc} \mathbf{s}^{(1)} & \mathbf{s}^{(2)} & \ldots & \mathbf{s}^{(p)} \end{array} \right].$$

Să presupunem că, după instruire, aplicăm vectorul cheie $\mathbf{s}^{(j)}$. Atunci:

$$\begin{aligned} \mathbf{v} &= \left(\sum_{i=1}^{p} \mathbf{f}^{(i)} \mathbf{s}^{(i)t} \right) \mathbf{s}^{(j)} \\ &= \mathbf{f}^{(1)} \mathbf{s}^{(1)t} \mathbf{s}^{(j)} + \ldots + \mathbf{f}^{(p)} \mathbf{s}^{(p)t} \mathbf{s}^{(j)}. \end{aligned}$$

Ideal ar fi să avem $\mathbf{v} = \mathbf{f}^{(j)}$. Aceasta se întâmplă dacă

$$\begin{aligned} \mathbf{s}^{(i)t} \mathbf{s}^{(j)} &= 0 \quad \text{pentru } i \neq j \\ \mathbf{s}^{(j)t} \mathbf{s}^{(j)} &= 1. \end{aligned}$$

Deci, mulţimea de vectori ortonormali $\{\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(p)}\}$ asigură maparea perfectă. Ortonormalitatea este o condiţie supusă intrărilor şi ea nu este întotdeauna respectată.

Să presupunem că $\mathbf{s}^{(j)'}$ este $\mathbf{s}^{(j)}$ perturbat:

$$\mathbf{s}^{(j)'} = \mathbf{s}^{(j)} + \Delta^{(j)}$$

unde termenul perturbaţie $\Delta^{(j)}$ se poate presupune statistic independent de $\mathbf{s}^{(j)}$. Pentru cazul când vectorii stocaţi în memorie sunt ortonormali, avem:

$$\begin{aligned} \mathbf{v} &= \mathbf{f}^{(j)} \mathbf{s}^{(j)t} \mathbf{s}^{(j)} + \mathbf{f}^{(j)} \mathbf{s}^{(j)t} \Delta^{(j)} + \sum_{\substack{i=1 \\ i \neq j}}^{p} \left(\mathbf{f}^{(i)} \mathbf{s}^{(i)t} \right) \Delta^{(j)} \\ &= \mathbf{f}^{(j)} + \sum_{\substack{i=1 \\ i \neq j}}^{p} \left(\mathbf{f}^{(i)} \mathbf{s}^{(i)t} \right) \Delta^{(j)}. \end{aligned}$$

Am ţinut cont şi de faptul că $\Delta^{(j)}$ este statistic independent de $\mathbf{s}^{(j)}$, deci pot fi consideraţi ortogonali, adică:

$$\mathbf{s}^{(j)t} \Delta^{(j)} = \|\mathbf{s}^{(j)t}\| \|\Delta^{(j)}\| \cos\psi = 0.$$

Observăm că răspunsul memoriei este asocierea dorită $\mathbf{f}^{(j)}$, plus o componentă aditivă datorată perturbaţiei $\Delta^{(j)}$. Această componentă conţine în paranteză aproape toţi termenii lui $\mathbf{W}$ ponderaţi de $\Delta^{(j)}$. Chiar şi în cazul în care vectorii memoraţi sunt ortonormali, la $\mathbf{f}^{(j)}$ se adaugă un zgomot foarte mare. Metoda este ineficientă pentru a regăsi un pattern perturbat.

În final, să notăm o proprietate interesantă a asociatorului liniar în cazul autoasociativ. În acest caz, reţeaua este un *autocorelator*. Luăm $\mathbf{f}^{(i)} = \mathbf{s}^{(i)}$ şi obţinem *matricea de autocorelaţie* $\mathbf{W}'$:

$$\mathbf{W}' = \sum_{i=1}^{p} \mathbf{s}^{(i)}\mathbf{s}^{(i)t} = \mathbf{S}\mathbf{S}^t,$$

ţinând cont de faptul că $\mathbf{W}'$ este simetrică. Să notăm însă că $\mathbf{W}'$ nu are diagonala egală cu zero. Presupunând $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(p)}$ ortonormali, pentru vectorul de intrare perturbat $\mathbf{s}^{(j)'}$ obţinem:

$$\begin{aligned} \mathbf{v} &= \mathbf{s}^{(j)} + \sum_{\substack{i=1 \\ i \neq j}} \mathbf{s}^{(i)}\mathbf{s}^{(i)t}\Delta^{(j)} \\ &= \mathbf{s}^{(j)} + (p-1)\Delta^{(j)} \end{aligned}$$

unde termenul $\Delta^{(j)}$ apare amplificat de $p-1$ ori!

Asociatorii liniari şi autoasociatorii pot fi folosiţi şi atunci când vectorii $\mathbf{s}^{(1)}$, ..., $\mathbf{s}^{(p)}$ sunt liniar independenţi (o condiţie mai slabă decât ortogonalitatea). Kohonen a arătat că, în acest caz, avem:

$$\mathbf{W} = \mathbf{F}(\mathbf{S}^t\mathbf{S})^{-1}\mathbf{S}^t.$$

Acest $\mathbf{W}$ minimizează eroarea pătratică dintre $\mathbf{f}^{(j)}$ şi $\mathbf{v}^{(j)}$.

Deoarece vectorii prototip nu sunt în general liniar independenţi, nici această metodă nu poate fi aplicată oricând. În plus, apare problema zgomotului.

În concluzie, asociatorii liniari nu sunt în general utilizabili în practică.

6.3 Concepte de bază ale memoriilor autoasociative recurente

O astfel de memorie este în esenţă o reţea feedback monostrat cu timp discret de tip Hopfield (fig. 6.5).

Recursivitatea este stohastic asincronă. Fiecare feedback se produce cu o întârziere Δ. Neuronii sunt alcătuiţi dintr-un sumator şi un comparator şi sunt bipolari discreţi. Ieşirile sunt în mulţimea $\{-1, 1\}^n$. Presupunem că $i_i = T_i = 0$, pentru $i = 1, 2, \ldots, n$.

Algoritmul de regăsire

$$v_i^{k+1} = \operatorname{sgn}\left(\sum_{j=1}^{n} w_{ij} v_j^k\right)$$

Presupunând că pornim din $\mathbf{v}^0$ şi că se alege în mod aleator secvenţa m, p, q de neuroni, avem:

$$\begin{array}{lll} \text{Prima actualizare:} & \mathbf{v}^1 = [& v_1^0 \quad v_2^0 \quad \ldots \quad v_m^1 \quad \ldots \quad v_p^0 \quad \ldots \quad v_q^0 \quad \ldots v_n^0 \ ]^t \\ \text{A doua actualizare:} & \mathbf{v}^2 = [& v_1^0 \quad v_2^0 \quad \ldots \quad v_m^1 \quad \ldots \quad v_p^2 \quad \ldots \quad v_q^0 \quad \ldots v_n^0 \ ]^t \\ \text{A treia actualizare:} & \mathbf{v}^3 = [& v_1^0 \quad v_2^0 \quad \ldots \quad v_m^1 \quad \ldots \quad v_p^2 \quad \ldots \quad v_q^3 \quad \ldots v_n^0 \ ]^t \end{array}$$

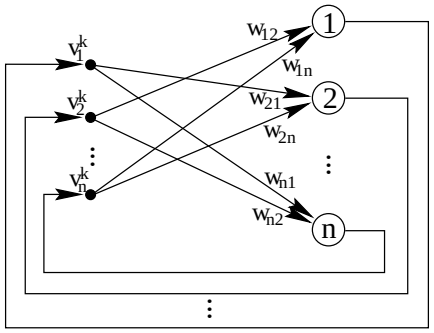


Figura 6.5: Memorie Hopfield autoasociativă.

Avem:

$$E(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v}.$$

Am văzut că energia este necrescătoare şi reţeaua se stabilizează într-unul din minimele locale energetice.

Fie $-\mathbf{v}$ complementul vectorului $\mathbf{v}$.

Se observă că $E(\mathbf{v}) = E(-\mathbf{v})$. Deci, un minim pentru $E(\mathbf{v})$ are aceeaşi valoare cu un minim pentru $E(-\mathbf{v})$. Cu alte cuvinte, tranziţiile memoriei se pot termina în aceeaşi măsură atât în $\mathbf{v}$, cât şi în $-\mathbf{v}$. Factorul decisiv care determină convergenţa este *similaritatea* dintre $\mathbf{v}^0$ şi $\mathbf{v}$, respectiv $-\mathbf{v}$.

Algoritmul de stocare

Fie $\mathbf{s}^{(1)}$, ..., $\mathbf{s}^{(p)}$ prototipurile bipolar binare care trebuie stocate. Algoritmul de stocare pentru calcularea matricii ponderilor este:

$$\mathbf{W} = \sum_{m=1}^{p} \mathbf{s}^{(m)}\mathbf{s}^{(m)t} - p\mathbf{I}$$

sau

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} s_i^{(m)} s_j^{(m)}$$

unde $\delta_{ij} = 0$ pentru $i \neq j$, $\delta_{ii} = 1$ şi $\mathbf{I}$ este matricea unitate.

Matricea $\mathbf{W}$ este foarte similară cu matricea de autocorelaţie obţinută prin învăţare hebbiană în cazul asociatorului liniar autoasociativ. Diferenţa este că $w_{ii} = 0$. Sistemul nu memorează vectorul $\mathbf{s}^{(m)}$, ci doar ponderile w_{ij} care reprezintă corelaţii între componentele acestui vector.

Oricând se pot adăuga noi autoasocieri adiţionale la memoria existentă, prin incrementarea ponderilor existente. Oricând se pot elimina autoasociatori prin decrementarea ponderilor existente. Regula de stocare este invariantă la ordinea în care sunt stocate prototipurile.

Dacă avem vectorii unipolari $\mathbf{s}^{(1)}$, ..., $\mathbf{s}^{(m)}$, algoritmul de stocare trebuie modificat astfel încât o componentă -1 să înlocuiască elementul 0 în vectorul unipolar originar:

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} (2s_i^{(m)} - 1)(2s_j^{(m)} - 1).$$

Să observăm că regula de stocare este invariantă la operaţia de complementare binară. Stocând vectorul $\mathbf{s}'^{(m)}$, complementar lui $\mathbf{s}^{(m)}$, obţinem:

$$w_{ij}{}' = (1 - \delta_{ij}) \sum_{m=1}^{p} (2s'^{(m)}_i - 1)(2s'^{(m)}_j - 1).$$

Substituind $s'^{(m)}_i = 1 - s_i^{(m)}$, obţinem $w_{ij}{}' = w_{ij}$. Cu alte cuvinte, este indiferent dacă stocăm un pattern sau complementul său binar.

Algoritmul de stocare şi regăsire pentru o memorie autoasociativă recurentă este următorul:

Se dau vectorii binar bipolari $\mathbf{s}^{(1)}$, ..., $\mathbf{s}^{(p)}$ *de câte* n *componente. Vectorul iniţial (care se cere regăsit) este* $\mathbf{v}^0$, *tot de* n *componente.*

Stocarea

1. *Se iniţializează* $\mathbf{W} \leftarrow 0$, $m \leftarrow 1$, *unde* $\mathbf{W}$ *este matricea* $n \times n$ *a ponderilor.*
2. *Se stochează* $\mathbf{s}^{(m)}$*:*

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{s}^{(m)}\mathbf{s}^{(m)t} - \mathbf{I}.$$

3. `if` $m < p$ `then` $m \leftarrow m + 1$, `go to` *2.*

Regăsirea

1. *Se iniţializează* $k \leftarrow 1$, $\mathbf{v} \leftarrow \mathbf{v}^0$, *unde* k *este un contor de ciclu.*
2. *Se iniţializează* $i \leftarrow 1$, *contorul de actualizare în ciclu. Se obţine o permutare aleatoare a întregilor* $1, \ldots, n$*:* $\alpha_1, \ldots, \alpha_n$. *actualizare* $\leftarrow FALSE$.
3. *Este actualizat neuronul* α_i, *dacă este cazul:*
   $net_{\alpha_i} = \sum_{j=1}^{n} w_{\alpha_i j} v_j$, $vnew \leftarrow sgn(net_{\alpha_i})$
   `if` $v_{\alpha_i} \neq vnew$ `then` *actualizare* $\leftarrow TRUE$, $v_{\alpha_i} \leftarrow vnew$.
4. `if` $i < n$ `then` $i \leftarrow i + 1$, `goto` *3.*
5. `if` *actualizare* $= FALSE$ `then` {*nu a avut loc nici o actualizare în acest ciclu, regăsirea se termină*} `display` k, v
   `else` $k \leftarrow k + 1$, `goto` *2.*

Consideraţii asupra modului de funcţionare

Memoria autoasociativă Hopfield este referită de multe ori ca un decodor corector de erori. Memoria lucrează cel mai bine cu valori mari ale lui n.

Să presupunem că n are o valoare mare. Presupunem că pattern-ul $\mathbf{s}^{(m')}$ este unul din cele p pattern-uri stocate. Acest pattern este acum la intrarea memoriei. Pentru neuronul i avem:

$$net_i = \sum_{j=1}^{n} w_{ij} s_j^{(m')}.$$

Aplicând formula:

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} s_i^{(m)} s_j^{(m)}$$

şi, neglijând pentru moment termenul $1 - \delta_{ij}$, obţinem:

$$\begin{aligned} net_i &\cong \sum_{j=1}^{n} \sum_{m=1}^{p} s_i^{(m)} s_j^{(m)} s_j^{(m')} \\ &= \sum_{m=1}^{p} s_i^{(m)} \sum_{j=1}^{n} s_j^{(m)} s_j^{(m')}. \end{aligned}$$

Dacă vectorii $\mathbf{s}^{(m)}$ şi $\mathbf{s}^{(m')}$ sunt statistic independenţi, atunci termenul

$$\sum_{j=1}^{n} s_j^{(m)} s_j^{(m')} \tag{6.1}$$

este în medie zero. De altfel, această sumă este produsul scalar $\mathbf{s}^{(m)}\mathbf{s}^{(m')}$. Dacă $\mathbf{s}^{(m)}$ şi $\mathbf{s}^{(m')}$ sunt ortogonali, deci statistic independenţi, acest produs devine zero. Dacă $\mathbf{s}^{(m)}$ şi $\mathbf{s}^{(m')}$ sunt într-o anumită măsură identici, atunci termenul 6.1 devine pozitiv. Dacă $\mathbf{s}^{(m)} = \mathbf{s}^{(m')}$, atunci termenul 6.1 este n.

Deci, dacă $\mathbf{s}^{(m)} = \mathbf{s}^{(m')}$ pentru un anumit $1 \leq m \leq p$, atunci

$$net_i \cong s_i^{(m')} n, \quad \text{pentru } i = 1, 2, \ldots, n. \tag{6.2}$$

net_i are acelaşi semn ca şi $s_i^{(m')}$, pentru $i = 1, \ldots, n$. În concluzie, vectorul $\mathbf{s}^{(m')}$ este stabil şi nu se mai poate modifica.

Vom da acum o explicaţie intuitivă a faptului că pattern-urile perturbate pot fi refăcute.

Să presupunem că vectorul de la intrare este o versiune perturbată a prototipului $\mathbf{s}^{(m')}$ care este stocat în memorie, iar perturbaţia afectează doar o mică parte a componentelor (majoritatea componentelor sunt neperturbate). În acest caz, în 6.2 multiplicatorul n va avea o valoare ceva mai mică, egală cu numărul de componente identice dintre $\mathbf{s}^{(m')}$ şi vectorul de intrare. Semnul lui $s_i^{(m')}$ se propagă asupra lui net_i. Componentele perturbate sunt actualizate, una câte una, cu valorile lor neperturbate din $\mathbf{s}^{(m')}$. Evident, este necesar ca majoritatea componentelor vectorului iniţial să fie neperturbate şi ca n să fie mare. Componentele perturbate se conformează deci dorinţei majorităţii.

Aceasta explică intuitiv cum un pattern perturbat poate fi asociat celui mai apropiat prototip stocat. Discuţia este însă valabilă doar pentru valori mari ale lui n.

Să presupunem acum că prototipurile stocate sunt ortogonale. Avem:

$$\mathbf{net} = \left(\sum_{m=1}^{p} \mathbf{s}^{(m)} \mathbf{s}^{(m)t} - p\mathbf{I} \right) \mathbf{s}^{(m')}.$$

Deoarece $\mathbf{s}^{(i)t}\mathbf{s}^{(j)} = 0$ pentru $i \neq j$ şi $\mathbf{s}^{(i)t}\mathbf{s}^{(j)} = n$, obţinem

$$\mathbf{net} = (n-p)\mathbf{s}^{(m')}.$$

Dacă $n > p$, rezultă că $\mathbf{s}^{(m')}$ este stabil. Funcţia de energie este:

$$\begin{aligned} E(\mathbf{v}) &= -\frac{1}{2}\mathbf{v}^t \mathbf{W} \mathbf{v} \\ &= -\frac{1}{2}\mathbf{v}^t \left(\sum_{m=1}^{p} \mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{v} + \frac{1}{2}\mathbf{v}^t p\mathbf{I}\mathbf{v}. \end{aligned}$$

Pentru fiecare prototip stocat $\mathbf{s}^{(m')}$ avem:

$$\begin{aligned} E(\mathbf{s}^{(m')}) &= -\frac{1}{2} \sum_{m=1}^{p} \left(\mathbf{s}^{(m')t} \mathbf{s}^{(m)} \right) \left(\mathbf{s}^{(m)t} \mathbf{s}^{(m')} \right) + \frac{1}{2} \mathbf{s}^{(m')t} p\mathbf{I} \mathbf{s}^{(m')} \\ &= -\frac{1}{2}(n^2 - pn). \end{aligned}$$

Memoria are o stare de echilibru la fiecare prototip $\mathbf{s}^{(m')}$, iar energia are valoarea ei minimă $-\frac{1}{2}(n^2 - pn)$ în aceste stări.

Pentru cazul mai general, când prototipurile nu sunt mutual ortogonale, energia nu mai este în mod necesar minimizată pentru fiecare prototip, iar prototipurile nu mai sunt în mod necesar stări de echilibru. În acest caz, avem:

$$\begin{aligned} \mathbf{net} &= n\mathbf{s}^{(m')} - p\mathbf{s}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^{p} \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')} \\ &= (n-p)\mathbf{s}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^{p} \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')}. \end{aligned}$$

Faţă de situaţia anterioară, apare suplimentar ultimul temen:

$$\sum_{\substack{m=1 \\ m \neq m'}}^{p} \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')}.$$

Acesta poate fi interpretat ca un vector *zgomot*. Cu cât $(n-p)$ este mai mare, cu atât scade importanţa zgomotului şi prototipurile devin stări de echilibru.

6.4 Analiza performanţei memoriilor autoasociative recurente

În această secţiune, vom stabili relaţii între mărimea n a memoriei şi numărul de pattern-uri distincte care pot fi refăcute în mod eficient. Aceasta depinde şi de gradul de similaritate dintre vectorul cheie şi cel mai apropiat prototip, precum şi de gradul de similaritate dintre prototipuri.

Pentru a măsura "similaritatea", vom folosi *distanţa Hamming*. De fapt, ea este proporţională cu "disimilaritatea". Pentru vectorii bipolari de n componente $\mathbf{x}$ şi $\mathbf{y}$, avem:

$$\mathrm{DH}(\mathbf{x},\mathbf{y}) = \frac{1}{2}\sum_{i=1}^{n}|x_i - y_i|.$$

Această înseamnă că $\mathrm{DH}(\mathbf{x},\mathbf{y})$ reprezintă numărul de poziţii în care diferă biţii. Prin actualizarea asincronă, la fiecare pas se modifică vectorul de ieşire cu $\mathrm{DH} = 1$.

Fie, de exemplu:

$$\begin{aligned}\mathbf{s}^{(1)} &= [\ -1 \quad -1 \quad 1 \quad 1\ ]^t\\ \mathbf{s}^{(2)} &= [\ -1 \quad 1 \quad 1 \quad -1\ ]^t\end{aligned}$$

Construim:

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & -2 & 0\\ 0 & 0 & 0 & -2\\ -2 & 0 & 0 & 0\\ 0 & -2 & 0 & 0 \end{bmatrix}.$$

Pornind din

$$\mathbf{v}^0 = [\ -1 \quad 1 \quad -1 \quad 1\ ]^t,$$

obţinem, prin actualizări asincrone ascendente:

$$\begin{aligned}\mathbf{v}^1 &= [\ 1 \quad -1 \quad -1 \quad 1\ ]^t\\ \mathbf{v}^2 &= [\ 1 \quad -1 \quad -1 \quad 1\ ]^t = \mathbf{v}^3 = \ldots\end{aligned}$$

Actualizările au fost astfel descrise pas cu pas. Convergenţa este către $-\mathbf{s}^{(2)}$ şi nu către $\mathbf{s}^{(1)}$ sau $\mathbf{s}^{(2)}$. Avem:

$$\mathrm{DH}\left(\mathbf{v}^0,\mathbf{s}^{(1)}\right) = \mathrm{DH}\left(\mathbf{v}^0,\mathbf{s}^{(2)}\right) = 2,$$

deci $\mathbf{v}^0$ nu este atras către $\mathbf{s}^{(1)}$ sau $\mathbf{s}^{(2)}$.

Fie acum:

$$\mathbf{v}^0 = [\ -1 \quad 1 \quad 1 \quad 1\ ]^t.$$

Avem:

$$\mathrm{DH}\left(\mathbf{v}^0,\mathbf{s}^{(1)}\right) = \mathrm{DH}\left(\mathbf{v}^0,\mathbf{s}^{(2)}\right) = 1.$$

Prin actualizări asincrone ascendente, obţinem:

$$\begin{aligned} \mathbf{v}^1 &= \mathbf{v}^0 \\ \mathbf{v}^2 &= [\ -1 \quad -1 \quad 1 \quad 1\ ]^t = \mathbf{v}^3 = \ldots = \mathbf{s}^{(1)} \end{aligned}$$

Pe de altă parte însă, dacă actualizările sunt în ordine descendentă, obţinem $\mathbf{s}^{(2)}$.

Pentru acest exemplu, avem $p/n = 2/4$, deci memoria este supraîncărcată şi sensibilă la perturbaţii. De aceea, refacerea pattern-urilor perturbate nu este întotdeauna eficientă. Se observă şi că nu putem evita stocarea complementelor pattern-urilor stocate.

În astfel de memorii, pot exista stări stabile care *nu reprezintă* pattern-uri memorate. De asemenea, convergenţa *nu este neapărat* către cel mai apropiat pattern memorat (apropiere măsurată prin DH). Aceste două deficienţe devin deranjante atunci când memoria este supraîncărcată, deci când p/n este mare.

Capacitatea memoriei autoasociative recurente

Vom menţiona aici rezultatele obţinute de McEliece *et al.*[1].

Un vector stare $\mathbf{v}^k$ al unei memorii este *stabil* dacă

$$\mathbf{v}^{k+1} = \Gamma\left[\mathbf{W}\mathbf{v}^k\right] = \mathbf{v}^k.$$

Această definiţie nu este legată de tipul de actualizare: sincron sau asincron.

Fie ρ *raza de atracţie* definită astfel: orice vector la o DH mai mică decât ρn (*distanţa de atracţie*) de starea stabilă $\mathbf{v}$ este eventual atras de $\mathbf{v}$. Am văzut că distanţa de atracţie este între 1 şi $n/2$, deci ρ este între $1/n$ şi $1/2$.

Pentru ca sistemul să funcţioneze ca o memorie, impunem ca fiecare vector memorat $\mathbf{s}^{(m)}$ să fie stabil. Mai puţin restrictiv este să cerem să existe un vector stabil la o distanţă mică de $\mathbf{s}^{(m)}$, εn, pentru toate cele p pattern-uri memorate.

Capacitatea asimptotică a unei memorii autoasociative cu n neuroni este:

$$c = \frac{(1-\rho)^2 n}{4 \ln n}.$$

Dacă $p < c$, atunci toate cele p pattern-uri stocate, sunt stabile cu probabilitatea aproape 1. Oricum ar fi valoarea $0 < \rho < 1/2$, capacitatea unei memorii Hopfield este deci între:

$$n/(4 \ln n) < c < n/(2 \ln n),$$

unde c depinde de fapt de toleranţa la eroare, adică de ρ. În loc de 2^n pattern-uri, putem stoca doar c pattern-uri!

Studiul lui McEliece *et al.* relevă prezenţa unor atractori în care nu au fost memorate pattern-uri; aceştia au în general bazinele de atracţie mai mici decât cele ale pattern-urilor stocate.

Chiar dacă numărul pattern-urilor ce pot fi stocate într-o memorie Hopfield este destul de mic, aceste memorii au o serie de aplicaţii în procesarea vorbirii, baze de date, prelucrarea imaginilor.

[1] McElice, R.J, Posner, E.C., Rodemich, E.R., Vankatesh, S.V. "The Capacity of the Hopfield Associative Memory". IEEE Trans. on Information Theory, 33, 1987, 461-482.

6.5 Memoria asociativă bidirecţională (MAB)

MAB (fig. 6.6) este o memorie heteroasociativă adresabilă prin conţinut constând din două straturi (Kosko, 1987, 1988).

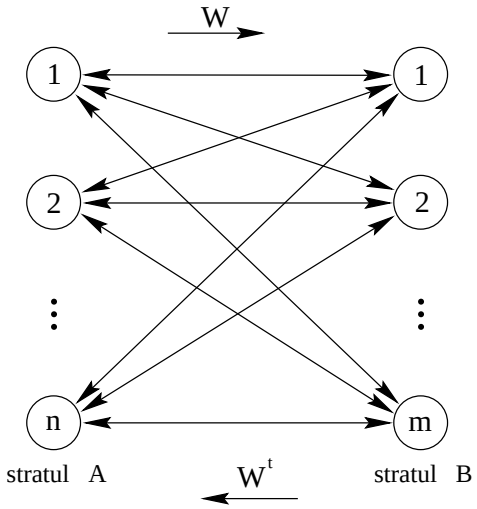


Figura 6.6: Memorie asociativă bidirecţională.

Fie p perechi de pattern-uri memorate:

$$\{\left(\mathbf{a}^{(1)},\mathbf{b}^{(1)}\right),\left(\mathbf{a}^{(2)},\mathbf{b}^{(2)}\right),\ldots\left(\mathbf{a}^{(p)},\mathbf{b}^{(p)}\right)\}$$

Presupunem că avem vectorul $\mathbf{b}$ la intrarea în stratul A al memoriei. Presupunem că neuronii sunt binari bipolari:

$$\mathbf{a}' = \Gamma[\mathbf{W}\mathbf{b}],$$

adică:

$$a_i{}' = \operatorname{sgn}\left(\sum_{j=1}^{m} w_{ij}b_j\right), \quad i = 1,\ldots,n.$$

Procesarea are loc sincron. Urmează:

$$\begin{aligned}\mathbf{b}' &= \Gamma\left[\mathbf{W}^t\mathbf{a}'\right]\\ b_j{}' &= \operatorname{sgn}\left(\sum_{i=1}^{n} w_{ij}a_i{}'\right), \quad j = 1,\ldots,m.\end{aligned}$$

Apoi:

$$\begin{aligned}\mathbf{a}'' &= \Gamma\left[\mathbf{W}\mathbf{b}'\right]\\ \mathbf{b}'' &= \Gamma\left[\mathbf{W}^t\mathbf{a}''\right] \text{ etc.}\end{aligned}$$

Procesul continuă până când $\mathbf{a}$ şi $\mathbf{b}$ nu se mai modifică.

În mod ideal, procesul se stabilizează într-o pereche $\left(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}\right)$ din mulţimea perechilor memorate. Putem opera asemănător pentru cazul unipolar. Presupunem că straturile sunt activate alternativ.

Stabilitatea unei astfel de reţele nu este afectată de procesarea sincronă ca în cazul memoriilor Hopfield, deoarece oricum cele două straturi lucrează asincron. De aceea, se preferă procesarea sincronă, convergenţa fiind mai rapidă decât în cazul în care am folosi procesarea asincronă.

Dacă $\mathbf{W}$ este pătrată şi simetrică, atunci $\mathbf{W} = \mathbf{W}^t$ şi avem de fapt o memorie autoasociativă cu un singur strat. Stocarea are loc conform regulii hebbiene:

$$\begin{aligned} \mathbf{W} &= \sum_{m=1}^{p} \mathbf{a}^{(m)} \mathbf{b}^{(m)t} \\ w_{ij} &= \sum_{m=1}^{p} a_i^{(m)} b_j^{(m)}. \end{aligned}$$

Presupunând că unul din pattern-urile stocate, $\mathbf{a}^{(m')}$, este prezentat la intrare, obţinem:

$$\begin{aligned} \mathbf{b} &= \Gamma\left[\sum_{m=1}^{p} \left(\mathbf{b}^{(m)} \mathbf{a}^{(m)t}\right) \mathbf{a}^{(m')}\right] \\ &= \Gamma\left[n\mathbf{b}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^{p} \mathbf{b}^{(m)} \mathbf{a}^{(m)t} \mathbf{a}^{(m')}\right]. \end{aligned}$$

Ultimul termen din relaţia anterioară, pe care îl notăm cu:

$$\eta = \sum_{\substack{m=1 \\ m \neq m'}}^{p} \mathbf{b}^{(m)} \mathbf{a}^{(m)t} \mathbf{a}^{(m')}$$

reprezintă *zgomotul*.

Să presupunem pentru moment că pattern-urile $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(p)}$ sunt ortogonale. Atunci, $\eta = 0$ şi se obţine $\mathbf{b} = \mathbf{b}^{(m')}$ dintr-un singur pas.

Dacă pattern-ul de intrare este o versiune perturbată a lui $\mathbf{a}^{(m')}$, stabilizarea la $\mathbf{b}^{(m')}$ nu mai este iminentă şi depinde de mai mulţi factori: DH dintre vectorul cheie şi prototipuri, ortogonalitatea sau DH dintre vectorii $\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(p)}$ etc.

Consideraţii asupra stabilităţii

Când memoria MAB ajunge la o stare de echilibru, astfel încât $\mathbf{a}^{(k)} \to \mathbf{b}^{(k+1)} \to \mathbf{a}^{(k+2)} = \mathbf{a}^{(k)}$, spunem că ea este bidirecţional stabilă.

Fie funcţia de energie:

$$E(\mathbf{a}, \mathbf{b}) = -\frac{1}{2}\mathbf{a}^t \mathbf{W} \mathbf{b} - \frac{1}{2}\mathbf{b}^t \mathbf{W}^t \mathbf{a} = -\mathbf{a}^t \mathbf{W} \mathbf{b}.$$

Vom evalua modificările energiei pentru un pas:

$$\begin{aligned}
\underset{i=1,\ldots,n}{\Delta a_i} &= \begin{cases} \geq 0 & \text{pentru } \sum_{j=1}^{m} w_{ij} b_j > 0 \\ 0 & \text{pentru } \sum_{j=1}^{m} w_{ij} b_j = 0 \\ \leq 0 & \text{pentru } \sum_{j=1}^{m} w_{ij} b_j < 0 \end{cases} \\
\underset{j=1,\ldots,n}{\Delta b_j} &= \begin{cases} \geq 0 & \text{pentru } \sum_{i=1}^{n} w_{ij} a_j > 0 \\ 0 & \text{pentru } \sum_{i=1}^{n} w_{ij} a_j = 0 \\ \leq 0 & \text{pentru } \sum_{i=1}^{n} w_{ij} a_j < 0 \end{cases}
\end{aligned} \tag{6.3}$$

Calculăm:

$$\begin{aligned}
\nabla_{\mathbf{a}} E(\mathbf{a}, \mathbf{b}) &= -\mathbf{W}\mathbf{b} \\
\nabla_{\mathbf{b}} E(\mathbf{a}, \mathbf{b}) &= -\mathbf{W}^t \mathbf{a}.
\end{aligned}$$

Modificările de energie cauzate de către modificarea unei singure componente sunt:

$$\begin{aligned}
\Delta E_{a_i}(\mathbf{a}, \mathbf{b}) &= -\left(\sum_{j=1}^{m} w_{ij} b_j \right) \Delta a_i, \quad \text{pentru } i = 1, \ldots, n \\
\Delta E_{b_j}(\mathbf{a}, \mathbf{b}) &= -\left(\sum_{i=1}^{n} w_{ij} a_i \right) \Delta b_j, \quad \text{pentru } j = 1, \ldots, m.
\end{aligned}$$

Rezultă că $\Delta E \leq 0$, dacă ţinem cont şi de 6.3. Deoarece E este inferior mărginită, adică:

$$E(\mathbf{a}, \mathbf{b}) \geq -\sum_{i=1}^{n} \sum_{j=1}^{m} |w_{ij}|$$

rezultă că memoria converge către un punct stabil care este un minim local pentru funcţia de energie. Deci, memoria este bidirecţional stabilă.

Să observăm că nu este important dacă procesarea într-un strat se face sincron sau asincron.

Kosko (1988) a arătat (euristic) că numărul maxim de perechi, p, care pot fi stocate şi regăsite cu succes, este $\min(n, m)$. O măsură mai strictă a capacităţii este:

$$p = \sqrt{\min(n, m)}.$$

Observaţii

1. Formula de stocare a perechilor de pattern-uri nu garantează că acestea corespund unor minime locale.

2. Am văzut că, dacă pattern-urile stocate sunt ortogonale, atunci zgomotul este zero şi toate pattern-urile stocate pot fi, în mod garantat, recuperate printr-un pas (dacă nu sunt perturbate). Wang (1990) a propus *creşterea fictivă* a pattern-urilor de instruire pentru a obţine o ortogonalizare a lor.

Perechile $\left(\mathbf{a}^{(i)}, \mathbf{a}^{(j)}\right)$ şi $\left(\mathbf{b}^{(i)}, \mathbf{b}^{(j)}\right)$ sunt *fără zgomot*, unde $i, j = 1, \ldots, p$, dacă:

$$\begin{aligned} \text{HD}\left(\mathbf{a}^{(i)}, \mathbf{a}^{(j)}\right) &= \frac{n}{2} \\ \text{HD}\left(\mathbf{b}^{(i)}, \mathbf{b}^{(j)}\right) &= \frac{m}{2}, \end{aligned}$$

acestea fiind condiţiile de ortogonalitate între $\mathbf{a}^{(i)}$ şi $\mathbf{a}^{(j)}$, respectiv $\mathbf{b}^{(i)}$ şi $\mathbf{b}^{(j)}$. Dacă avem această situaţie, atunci recuperarea datelor este imediată şi fără erori (dacă nu sunt perturbate). Prin creşterea vectorilor $\mathbf{a}^{(i)}$ şi $\mathbf{b}^{(i)}$, $i = 1, \ldots, p$, cu componente adiţionale obţinem această ortogonalitate la nivel de perechi.

Prin această tehnică se îmbunătăţeşte şi capacitatea MAB de a corecta erori.

Utilizarea MAB pentru pattern-uri temporale

Fie secvenţa $S = \left\{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(p)}\right\}$ de vectori n-dimensionali bipolari binari reprezentând tranziţiile stărilor (pattern-uri temporale). Reţeaua MAB este capabilă să memoreze secvenţa S, astfel încât:

$$\mathbf{s}^{(i+1)} = \Gamma\left[\mathbf{W}\mathbf{s}^{(i)}\right]$$

unde i este modulo $p+1$.

Pornind de la starea iniţială $\mathbf{x}(0)$ în vecinătatea lui $\mathbf{s}^{(i)}$, secvenţa S este apelată ca un ciclu de tranziţii ale stărilor. Acest model este numit *memorie asociativă temporală*.

Pentru stocarea secvenţei, astfel încât $\mathbf{s}^{(1)}$ să fie asociat cu $\mathbf{s}^{(2)}$, $\mathbf{s}^{(2)}$ cu $\mathbf{s}^{(3)}$, ..., $\mathbf{s}^{(p)}$ cu $\mathbf{s}^{(1)}$, procedăm astfel:

$$\mathbf{W} = \sum_{i=1}^{p-1} \mathbf{s}^{(i+1)}\mathbf{s}^{(i)t} + \mathbf{s}^{(1)}\mathbf{s}^{(p)t}$$

sau, dacă indicele este modulo $p+1$:

$$\mathbf{W} = \sum_{i=1}^{p} \mathbf{s}^{(i+1)}\mathbf{s}^{(i)t}.$$

Având straturile A şi B, avem:

$$\begin{aligned} \mathbf{a} &= \Gamma[\mathbf{W}\mathbf{b}] \\ \mathbf{b} &= \Gamma[\mathbf{W}\mathbf{a}]. \end{aligned}$$

Dacă secvenţa $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(p)}$ este aplicată la intrarea în stratul A, avem:

$$\mathbf{a} = \Gamma\Big[\overbrace{\underbrace{\left(\mathbf{s}^{(2)}\mathbf{s}^{(1)t} + \ldots + \mathbf{s}^{(k+1)}\mathbf{s}^{(k)t} + \ldots + \mathbf{s}^{(1)}\mathbf{s}^{(p)t}\right)}_{\mathbf{W}}\mathbf{s}^{(k)}}^{n\mathbf{s}^{(k+1)}+\eta}\Big].$$

Termenul

$$\eta = \sum_{\substack{i=1 \\ i \neq k}}^{p} \mathbf{s}^{(i+1)} \left(\mathbf{s}^{(i)t} \mathbf{s}^{(k)} \right)$$

este zgomotul, unde indicele de însumare este modulo $p+1$. Dacă vectorii din S sunt ortogonali, atunci $\eta = 0$ şi $\mathbf{a} = \mathbf{s}^{(k+1)}$, dintr-un singur pas.

Aşadar, dacă la intrare avem $\mathbf{s}^{(k)}$, obţinem în mod circular:

$$\mathbf{s}^{(k+1)} \rightarrow \mathbf{s}^{(k+2)} \rightarrow \ldots \rightarrow \mathbf{s}^{(p)} \rightarrow \ldots$$

Dacă vectorii din S nu sunt ortogonali, presupunând însă că ei sunt stocaţi la o DH $<< n$, în general ne putem aştepta să obţinem totuşi secvenţa corectă S dacă aplicăm $\mathbf{s}^{(k)}$.

Această memorie poate stoca cel mult k secvenţe de lungimi $p_1, p_2, \ldots p_k$, unde:

$$p = \sum_{i=1}^{k} p_i, \quad p < n.$$

6.6 Exerciţii

1. Asociatorul liniar trebuie să asocieze următoarele perechi de vectori:

$$\begin{aligned}
\mathbf{s}^{(1)} &= \left[\begin{array}{cccc} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{array} \right]^t \rightarrow \mathbf{f}^{(1)} = \left[\begin{array}{ccc} 0 & 1 & 0 \end{array} \right]^t \\
\mathbf{s}^{(2)} &= \left[\begin{array}{cccc} \frac{1}{2} & -\frac{5}{6} & \frac{1}{6} & \frac{1}{6} \end{array} \right]^t \rightarrow \mathbf{f}^{(2)} = \left[\begin{array}{ccc} 1 & 0 & 1 \end{array} \right]^t \\
\mathbf{s}^{(3)} &= \left[\begin{array}{cccc} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & -\frac{5}{6} \end{array} \right]^t \rightarrow \mathbf{f}^{(3)} = \left[\begin{array}{ccc} 0 & 0 & 0 \end{array} \right]^t
\end{aligned}$$

   (a) Verificaţi dacă vectorii $\mathbf{s}^{(1)}$, $\mathbf{s}^{(2)}$, $\mathbf{s}^{(3)}$ sunt ortonormali.

   (b) Calculaţi matricea ponderilor.

   (c) Verificaţi asocierea produsă de reţea pentru unul din vectorii $\mathbf{s}^{(i)}$.

   (d) Perturbaţi $\mathbf{s}^{(i)}$ dându-i unei componente valoarea 0 şi calculaţi vectorul rezultat perturbat.

2. Asociatorul liniar din figura 6.7 a fost instruit să asocieze vectorilor de intrare:

$$\begin{aligned}
\mathbf{s}^{(1)} &= \left[\begin{array}{ccc} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{array} \right]^t \\
\mathbf{s}^{(2)} &= \left[\begin{array}{ccc} \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{array} \right]^t \\
\mathbf{s}^{(3)} &= \left[\begin{array}{ccc} \frac{1}{\sqrt{6}} & -\frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{array} \right]^t
\end{aligned}$$

   vectorii $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, $\mathbf{f}^{(3)}$. Găsiţi matricea $\mathbf{W}$ şi vectorii $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, $\mathbf{f}^{(3)}$ care sunt codificaţi în reţea.

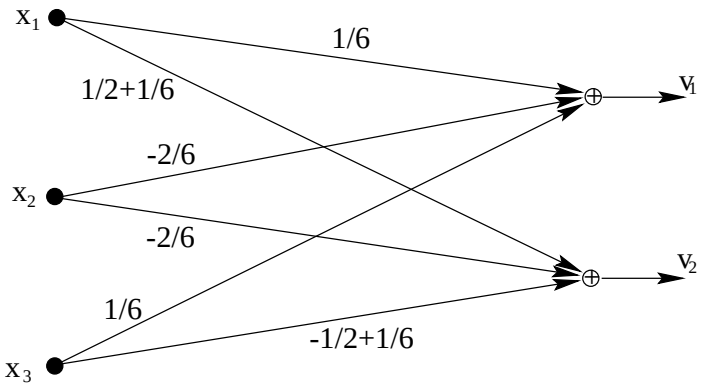


Figura 6.7: Asociatorul liniar propus pentru exerciţiul 2.

3. Autoasociatorul liniar trebuie să asocieze versiuni perturbate ale vectorilor $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ cu prototipurile lor neperturbate din problema 1. Calculaţi matricea $\mathbf{W}$.

4. Următorii vectori trebuie stocaţi într-o memorie autoasociativă recurentă:

$$\begin{aligned} \mathbf{s}^{(1)} &= [\ 1 \quad 1 \quad 1 \quad 1 \quad 1\ ]^t \\ \mathbf{s}^{(2)} &= [\ 1 \quad -1 \quad -1 \quad 1 \quad -1\ ]^t \\ \mathbf{s}^{(3)} &= [\ -1 \quad 1 \quad 1 \quad 1 \quad 1\ ]^t \end{aligned}$$

   (a) Calculaţi matricea $\mathbf{W}$.

   (b) Aplicaţi la intrare vectorul $\mathbf{v}^0 = [\ 1 \quad -1 \quad -1 \quad 1 \quad 1\ ]^t$ şi folosiţi actualizarea asincronă ascendentă.

   (c) Aplicaţi $\mathbf{v}^0$ prin actualizare asincronă descendentă.

   (d) Comentaţi dacă memoria Hopfield oferă o soluţie optimă din punct de vedere al DH. (Atenţie, memoria este supraîncărcată. Luaţi $\mathrm{sgn}(0) = 1$ în această problemă).

5. O memorie asociativă temporală trebuie să stocheze următoarea secvenţă:

$$\begin{aligned} \mathbf{s}^{(1)} &= [\ 1 \quad -1 \quad -1 \quad 1 \quad -1\ ]^t \\ \mathbf{s}^{(2)} &= [\ -1 \quad 1 \quad 1 \quad -1 \quad -1\ ]^t \\ \mathbf{s}^{(3)} &= [\ -1 \quad 1 \quad 1 \quad 1 \quad -1\ ]^t \end{aligned}$$

   Calculaţi $\mathbf{W}$ şi verificaţi regăsirea pattern-urilor. Încercaţi să introduceţi la intrare pattern-uri perturbate.

6. (C) Proiectaţi o memorie heteroasociativă bidirecţioanlă care asociază caracterele (A, C), (I, I) şi (O, T). Reprezentaţi caracterele A, I, O ca matrici

binare de 5×3 pixeli, iar caracterele C, I, T ca matrici binare de 3×3 pixeli. Verificaţi memoria folosind la intrare pattern-uri neperturbate şi pattern-uri perturbate.

Capitolul 7

Reţele neurale cu auto-organizare

În acest capitol ne vom concentra pe reţele neurale care trebuie să descopere singure relaţiile interesante din cadrul pattern-urilor de intrare, deci nu vom folosi instruirea supervizată. Obiectivul nostru este să explorăm algoritmii de instruire nesupervizată ai reţelelor neurale, reţele care au astfel capacitatea de *auto-organizare*, adică de adaptare.

7.1 Reţelele Hamming şi MAXNET

Vom considera un clasificator cu două straturi pentru vectori binari bipolari (fig. 7.1).

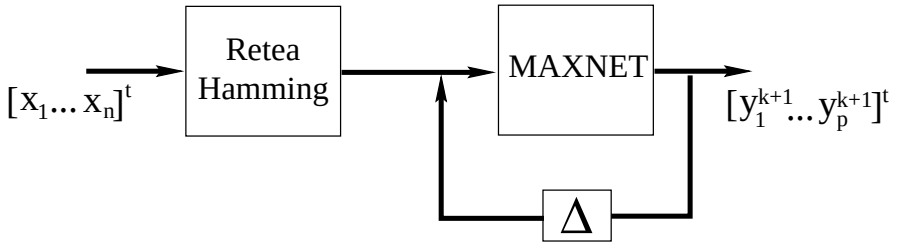


Figura 7.1: Clasificator cu două straturi pentru vectori binari.

Prima parte este o reţea Hamming, detaliată în figura 7.2.

Fie $\mathbf{s}^{(m)}$, $m = 1, \ldots, p$, vectorul prototip pentru clasa m. Presupunem că avem pentru fiecare clasă un vector prototip şi, corespunzător, un neuron.

Fie $\mathbf{x}$ vectorul de intrare. Folosim produsul scalar al vectorilor ca o măsură de matching (potrivire) între aceşti vectori. Ideea este să luăm:

$$\mathbf{w}_m = \left[\begin{array}{ccc} w_{m1} & \ldots & w_{mn} \end{array} \right]^t \quad \text{pentru } m = 1, \ldots, p$$

şi să avem $\mathbf{x}^t \mathbf{w}_m$ maxim când $\mathbf{x} = \mathbf{s}^{(m)}$. Deoarece vectorii sunt binari bipolari, produsul $\mathbf{x}^t \mathbf{s}^{(m)}$ reprezintă numărul de poziţii în care $\mathbf{x}$ şi $\mathbf{s}^{(m)}$ diferă. Numărul

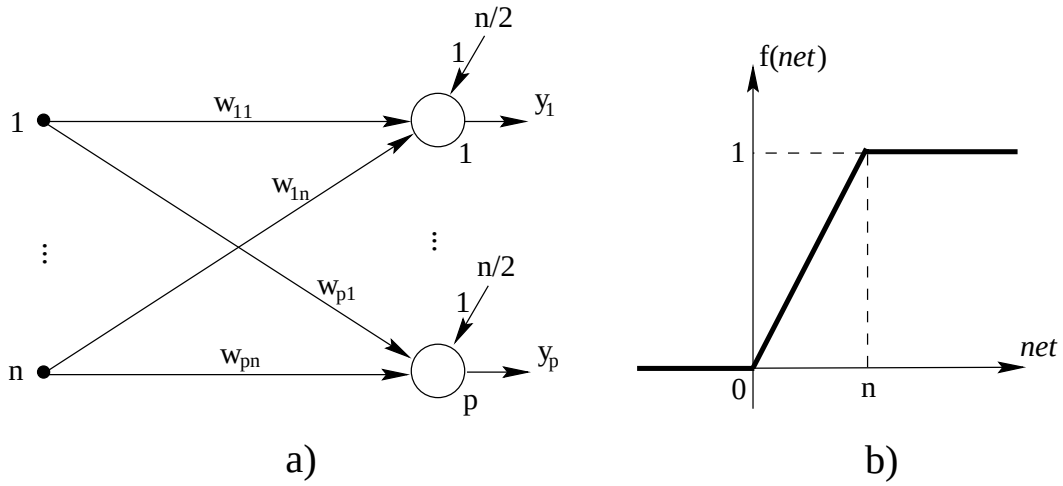


Figura 7.2: a) Reţeaua Hamming; b) Funcţia de activare a neuronilor reţelei Hamming. Zonele în care $f(net) < 0$ şi $f(net) > 1$ sunt inactive pentru că $0 \leq net \leq n$.

de poziţii în care aceşti doi vectori diferă este însă chiar distanţa Hamming, $\mathrm{DH}\left(\mathbf{x}, \mathbf{s}^{(m)}\right)$, prin definiţie. Atunci:

$$\mathbf{x}^t\mathbf{s}^{(m)} = \underbrace{\left(n - \mathrm{DH}\left(\mathbf{x}, \mathbf{s}^{(m)}\right)\right)}_{\substack{\text{nr. de poziţii} \\ \text{în care sunt egali}}} - \mathrm{DH}\left(\mathbf{x}, \mathbf{s}^{(m)}\right).$$

Obţinem:

$$\frac{1}{2}\mathbf{x}^t\mathbf{s}^{(m)} = \frac{n}{2} - \mathrm{DH}\left(\mathbf{x}, \mathbf{s}^{(m)}\right).$$

Luăm:

$$\mathbf{W}_H = \frac{1}{2}\begin{bmatrix} s_1^{(1)} & \dots & s_n^{(1)} \\ s_1^{(2)} & \dots & s_n^{(2)} \\ & \ddots & \\ s_1^{(p)} & \dots & s_n^{(p)} \end{bmatrix}$$

unde factorul $\frac{1}{2}$ foloseşe la scalare. La intrarea în nodul m, $m = 1, \dots, p$ avem:

$$\begin{aligned} net_m &= \frac{1}{2}\mathbf{x}^t\mathbf{s}^{(m)} + \frac{n}{2} \\ &= n - \mathrm{DH}\left(\mathbf{x}, \mathbf{s}^{(m)}\right). \end{aligned}$$

Se observă că am adăugat termenul $\frac{n}{2}$. Luăm $f\left(net_m\right) = \frac{1}{n} net_m$, $m = 1, \dots, p$. Deoarece $0 \leq net_m \leq n$, avem $0 \leq f(net_m) \leq 1$, $m = 1, \dots, p$.

Avem:

1. Dacă $\mathbf{x} = \mathbf{s}^{(m)}$, atunci $\mathrm{DH} = \left(\mathbf{x}, \mathbf{s}^{(m)}\right) = 0$ şi $net_m = n$, deci $f\left(net_m\right) = 1$.

2. Dacă $\mathbf{x}$ este complementul lui $\mathbf{s}^{(m)}$, atunci DH $\left(\mathbf{x}, \mathbf{s}^{(m)}\right) = n$, $net_m = 0$ şi $f\left(net_m\right) = 0$.

În general, numărul neuronului cu cea mai mare ieşire indică numărul clasei căreia îi este atribuit $\mathbf{x}$, din punct de vedere al DH. Reţeaua Hamming este, evident, de tip feedforward.

Reţeaua MAXNET (fig. 7.3) este recurentă şi are rolul de a transforma ieşirile din reţeaua Hamming, astfel încât toate aceste ieşiri să devină 0, cu excepţia celei maxime.

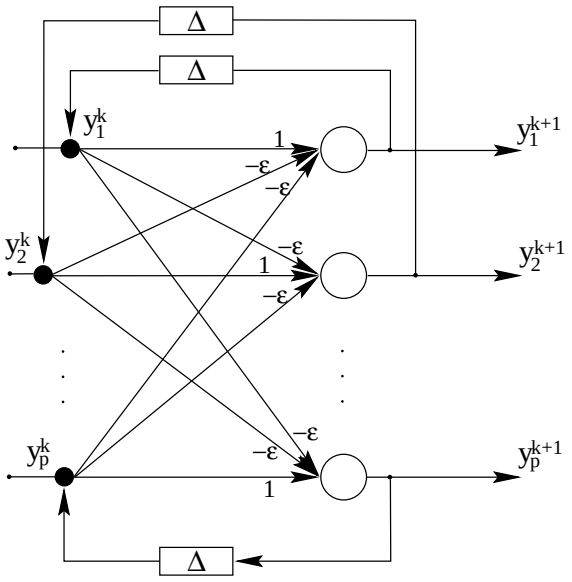


Figura 7.3: Reţeaua MAXNET.

Matricea ponderilor reţelei MAXNET este matricea $p \times p$:

$$\mathbf{W}_M = \begin{bmatrix} 1 & -\varepsilon & -\varepsilon & \dots & -\varepsilon \\ -\varepsilon & 1 & -\varepsilon & \dots & -\varepsilon \\ \vdots & & \ddots & & \vdots \\ -\varepsilon & & \dots & & 1 \end{bmatrix}$$

unde $0 < \varepsilon < 1/p$, ε fiind *coeficientul de interacţie laterală*.

Luând (fig. 7.4):

$$\begin{aligned} f(net) &= \begin{cases} 0, & net < 0 \\ net, & net \geq 0 \end{cases} \\ \mathbf{y}^{(k+1)} &= \Gamma\left[\mathbf{W}_M \mathbf{y}^k\right] \end{aligned}$$

şi presupunând $0 \leq y_i^0 \leq 1$, $i = 1, \dots, p$, valorile de iniţ ializare, obţinem următorul proces:

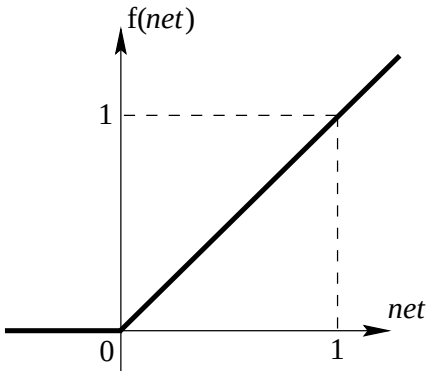


Figura 7.4: Funcţia de activare a neuronilor reţelei MAXNET. Zona în care avem $net > 1$ este inactivă.

> La fiecare recurenţă, fiecare componentă a vectorului $\mathbf{y}^{(k+1)}$ scade până devine 0. Componentele lui $\mathbf{y}^{(k+1)}$ devin 0 în ordinea inversă a mărimii lor. Procesul se stabilizează când o singură componentă (cea maximă) rămâne nenulă. Această componentă scade cel mai lent. Pe măsură ce sunt anulate componente, descreşterea este mai lentă.

Să comparăm această reţea cu asociatorul Hopfield. Presupunem că trebuie să clasificăm în 5 clase posibile un vector cu 50 de componente.

1. Reţeaua Hamming necesită 255 de conexiuni pentru cei 5 neuroni.

2. Reţeaua Hopfield necesită 2450 de conexiuni pentru cei 50 de neuroni. În plus, capacitatea de stocare este limitată.

Reţeaua Hamming pare mai eficientă. Totuşi, ea nu regăseşte pattern-ul prototip, ci doar indicele clasei din care face parte pattern-ul de intrare. Deci, reţeaua Hamming nu poate restaura un pattern perturbat, ci găseşte doar distanţa minimă a acestui pattern faţă de prototipuri.

Reţeaua Hamming este un clasificator propriu-zis, nu o memorie autoasociativă.

7.2 Instruirea nesupervizată a clusterelor

Clustering înseamnă gruparea pattern-urilor similare şi separarea celor nesimilare. Există tehnici clasice de clustering: k-means, ISODATA etc, dar noi ne vom concentra pe metodele conexioniste.

Instruirea *câştigătorul ia tot*

Presupunem că avem secvenţa de instruire $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ care reprezintă p clustere. Pentru reţeaua Kohonen (fig. 7.5), despre care vom discuta în continuare, avem:

$$\mathbf{y} = \Gamma[\mathbf{W}\mathbf{x}]$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^t \\ \mathbf{w}_2^t \\ \vdots \\ \mathbf{w}_p^t \end{bmatrix}, \quad \text{unde } \mathbf{w}_i = \begin{bmatrix} w_{i1} \\ \vdots \\ w_{in} \end{bmatrix}, \quad i = 1, \ldots, p$$

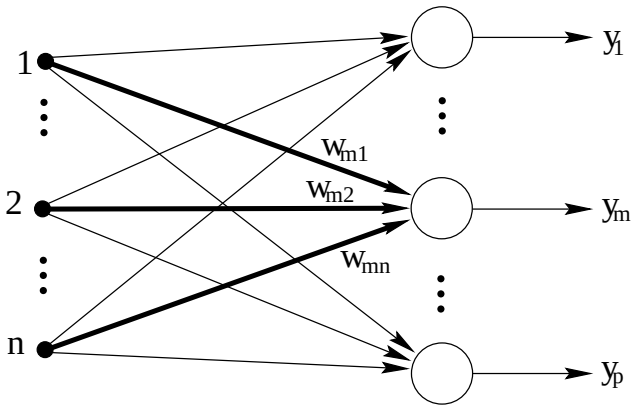


Figura 7.5: Reţeaua Kohonen.

Algoritmul de instruire este de tip *câştigătorul ia tot*. Vectorii $\mathbf{w}_i$, $i = 1, \ldots, p$ sunt variabili şi trebuie învăţaţi. Înainte de instruire, se normalizează vectorii ponderilor:

$$\hat{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}. \quad i = 1, \ldots, p.$$

Se calculează indicele m pentru care:

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = \min_{i=1,\ldots,p} \{\|\mathbf{x} - \hat{\mathbf{w}}_i\|\}$$

unde $\mathbf{x}$ este vectorul de intrare. Deoarece:

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = \left(\mathbf{x}^t\mathbf{x} - 2\hat{\mathbf{w}}_m^t\mathbf{x} - 1\right)^{1/2},$$

avem:

$$\hat{\mathbf{w}}_m^t\mathbf{x} = \max_{i=1,\ldots,p} (\hat{\mathbf{w}}_i^t\mathbf{x}).$$

Neuronul al m-lea, cu cea mai mare valoare pentru intrare este declarat *câştigător*.

În general, avem:

$$\cos\psi = \frac{\mathbf{x}^t\mathbf{x}_i}{\|\mathbf{x}\|\|\mathbf{x}_i\|}.$$

În loc de $\|\mathbf{x} - \mathbf{x}_i\|$, putem folosi $\cos\psi$ ca şi criteriu de similaritate între $\mathbf{x}$ şi $\mathbf{x}_i$, astfel încât $\mathbf{x}$ este "mai aproape" de $\mathbf{x}_i$ dacă ψ este mai mic. Este necesar însă să normalizăm vectorii $\mathbf{x}$ şi $\mathbf{x}_i$, pentru că altfel criteriul nu mai funcţionează: se poate ca $\|\mathbf{x} - \mathbf{x}_i\|$ să fie mare, dar ψ să fie mic.

Reducem acum $\|\mathbf{x} - \mathbf{w}_m\|$ în direcţia gradientului negativ al distanţei:

$$\begin{aligned} \nabla_{\mathbf{w}_m} \|\mathbf{x} - \mathbf{w}_m\|^2 &= -2(\mathbf{x} - \mathbf{w}_m) \\ \Delta \hat{\mathbf{w}}_m{}' &= \alpha\,(\mathbf{x} - \hat{\mathbf{w}}_m) \end{aligned}$$

unde α este de obicei între 0,1 şi 0,7. Celelalte ponderi nu se schimbă. Avem atunci:

$$\begin{cases} \hat{\mathbf{w}}_m^{k+1} = \hat{\mathbf{w}}_m^k + \alpha^k \left(\mathbf{x} - \hat{\mathbf{w}}_m^k\right) \\ \hat{\mathbf{w}}_i^{k+1} = \hat{\mathbf{w}}_i^k, \quad i \neq m \end{cases} .$$

Procesul continuă pentru un nou pattern $\mathbf{x}$. Parametrul α se reduce monoton şi instruirea încetineşte pe parcurs. Vectorul modificat $\hat{\mathbf{w}}_m^{k+1}$ este normalizat şi intră în pasul următor. În final, fiecare $\hat{\mathbf{w}}_i$ va reprezenta centrul de greutate al unei regiuni de decizie, deci a unui cluster.

Funcţiile de activare ale neuronilor nu au relevanţă, dar în general sunt de tip continuu.

Obţinem în final un clasificator în care pattern-ul de intrare $\mathbf{x}$ este asociat grupării m, unde $\mathbf{y}_m = \max(\mathbf{y}_1, \ldots, \mathbf{y}_p)$.

După instruire, reţeaua Kohonen poate fi folosită ca şi clasificator. Pentru aceasta, trebuie să o *calibrăm* supervizat: se aleg p reprezentanţi ai celor p clustere şi se aplică, etichetând corespunzător cele p noduri cu ieşirile respective. Astfel, etichetăm de fapt clusterele.

Ponderile iniţiale sunt luate, în general, în mod aleator, astfel încât să acopere uniform spaţiul pattern-urilor.

Reţeaua Kohonen are limitări legate de arhitectura monostrat: nu poate manipula eficient pattern-uri liniar neseparabile. Chiar şi pentru pattern-uri liniar separabile, ponderile pot să se blocheze în regiuni izolate, fără să formeze clusterele adecvate. În astfel de situaţii, se reiniţializează ponderile cu valori noi, sau se adaugă zgomot la vectorii pondere în timpul instruirii.

Dacă numărul grupărilor este *a priori* necunoscut, se ia un p suficient de mare. Pe parcursul instruirii, unii neuroni se vor comporta haotic, ponderile către ei nestabilizându-se. Aceşti neuroni nu reprezintă clustere, deci pot fi omişi în faza de utilizare a reţelei.

7.3 Hărţi Kohonen

Centrii anumitor activităţi (vorbire, vedere, auz, funcţii motoare) sunt localizaţi în anumite arii specifice ale cortexului. Mai mult, aceste arii prezintă o ordonare logică a funcţionalităţii lor. Putem da ca exemplu, *harta tonotopică* a regiunilor auditive, unde neuronii vecini răspund unor frecvenţe similare ale sunetului, de la frecvenţe înalte la frecvenţe joase. Un alt exemplu este *harta somatotopică*.

Astfel de regiuni, cum este harta tonotopică, se numesc *hărţi de trăsături ordonate* (*ordered feature maps*). Vom studia în această secţiune mecanismul prin care aceste hărţi de trăsături ordonate se pot dezvolta natural.

Cortexul este o suprafaţă de 2-4 mm grosime, având aproximativ $1m^2$ şi constând din 6 straturi de neuroni de tip şi densitate variabile. El are forma pe care o cunoaştem pentru a maximiza densitatea în craniu.

Vom studia un model al cortexului care este o suprafaţă bidimensională de elemente de procesare. Vectorii de intrare vor fi proiectaţi, prin reducerea dimensionalităţii, pe această suprafaţă, menţinându-se însă ordinea între ei. Această suprafaţă este o hartă care conservă topologia datelor de intrare (*topology preserving map*). O astfel de hartă este o hartă cu auto-organizare şi este instruită nesupervizat. Neuronii biologici de pe suprafaţa menţionată au conexiuni laterale a căror tărie depinde de distanţa dintre neuronii respectivi (fig. 7.6).

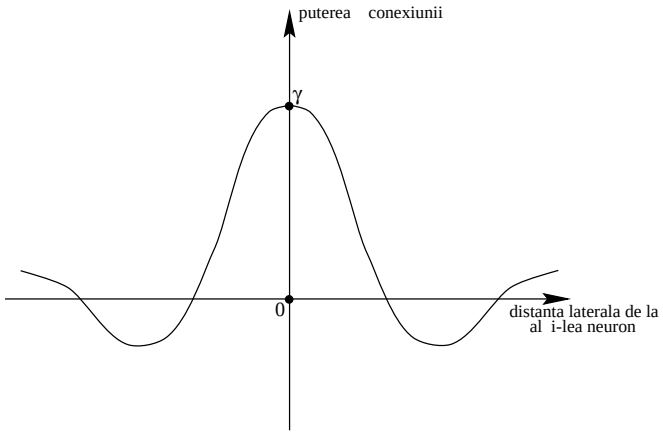


Figura 7.6: "Pălăria mexicană" descrie puterea conexiunii laterale a neuronilor biologici.

În vecinătatea cu raza de 50-100 μm, conexiunile sunt excitatoare. Deci, auto-feedback-ul fiecărui neuron este pozitiv, exprimat de coeficientul $\gamma > 0$ (fig. 7.7).

Aria excitatoare este înconjurată de un inel de conexiuni mai slabe, inhibitoare, cu raza de 200-500μm.

Autoorganizarea unei astfel de hărţi poate fi exemplificată pe un model liniar de 10 neuroni, reprezentând o secţiune arbitrară a unei hărţi bidimensionale (fig. 7.8).

Calculăm:

$$y_i(t+1) = f\left(x_i(t+1) + \sum_{k=-k_0}^{k_0} y_{i+k}(t)\gamma_k\right).$$

Semnalele $y(t)$ sunt întârziate cu o unitate înainte de a deveni feedback-uri.

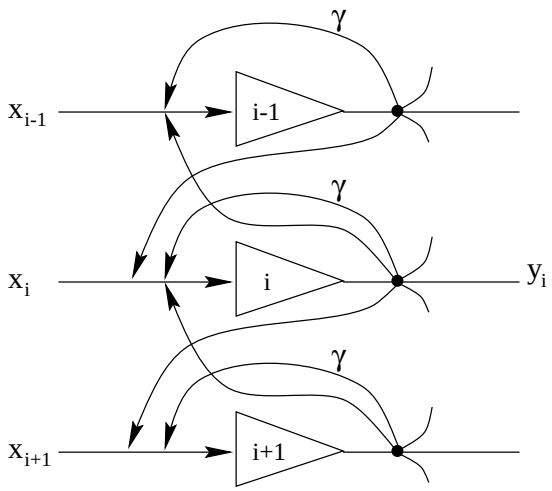


Figura 7.7: Modelarea interacţiunilor dintre neuronii biologici.

Coeficienţii de feedback, γ_k, depind de distanţa dintre neuroni. Luând:

$$x_i(t) = 0,5 \ \sin^3\left(\frac{\pi(i+3)}{17}\right), \quad i = 1, \ldots, 10$$

$$f(net) = \begin{cases} 0 & net \leq 0 \\ net & 0 < net < 2 \\ 2 & net \geq 2 \end{cases}$$

$$b = 0,4$$

$$c = 0,2$$

se obţin ieşirile din figura 7.9.

Un *algoritm de proiecţie a trăsăturilor* converteşte un pattern de dimensiune arbitrară în răspunsurile unui tablou unidimensional sau bidimensional de neuroni.

Rezultatele următoare aparţin lui Kohonen (1990).

Presupunem că vectorii de intrare, cu un număr arbitrar, dar fixat, de componente, sunt proiectaţi pe un tablou de neuroni având o topologie hexagonală (fig. 7.10). În această figură, m este neuronul câştigător, iar $N_m(t_1)$ este vecinătatea lui m la momentul t_1.

După instruire, fiecare intrare $\mathbf{x}$ produce un răspuns în reţea, poziţia locală a răspunsului fiind semnificativă: ea reflectă caracteristica trăsăturii dominante a lui $\mathbf{x}$. Această proiecţie a trăsăturilor este o proiecţie neliniară a spaţiului patternurilor pe tabloul neuronilor. Proiecţia face ca relaţiile de vecinătate topologică să expliciteze relaţiile de ordine dintre datele de intrare, aceste relaţii din urmă fiind, însă, într-un spaţiu de dimensiune mult mai mare.

Să vedem cum are loc instruirea.

Intrarea $\mathbf{x}$ este aplicată simultan tuturor neuronilor. Fie

$$\|\mathbf{x} - \mathbf{w}_m\| = \min_i \{\|\mathbf{x} - \mathbf{w}_i\|\}$$

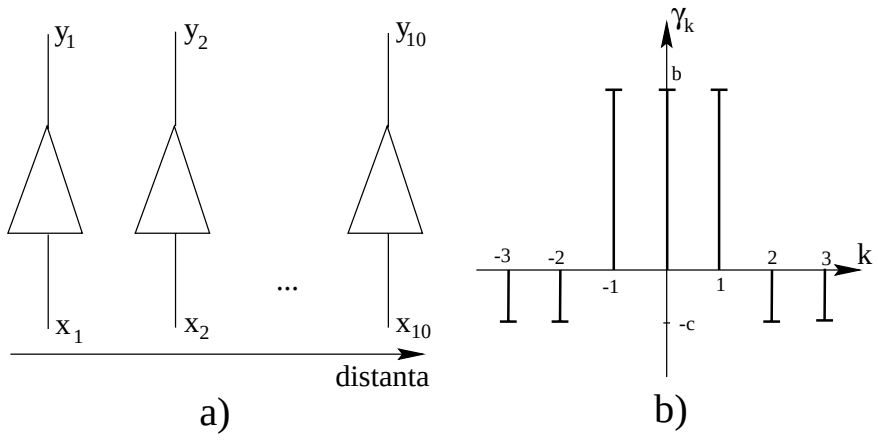


Figura 7.8: a) Secţiune arbitrară a unei hărţi bidimensionale; b) Feedback-ul lateral.

unde $\mathbf{w}_i$ este vectorul ponderilor către neuronul i.

Dacă neuronul m este câştigător, deci răspunde maxim pentru $\mathbf{x}$, aceasta antrenează, ca în exemplul precedent, neuronii vecini. Nu avem un neuron câştigător, ci o vecinătate N_m câştigătoare. Se ajustează doar ponderile către neuronii din N_m.

Raza lui N_m trebuie să scadă pe măsură ce instruirea progresează:

$$N_m(t_1) > N_m(t_2) > N_m(t_3) \ldots \text{ pentru } t_1 < t_2 < t_3 \ldots$$

Se începe cu o rază foarte mare şi se termină cu vecinătatea care include doar neuronul câştigător. Mecanismul interacţiilor laterale, tip "pălărie mexicană", este încadrat în definiţia acestor vecinătăţi locale şi a modului de adaptare a ponderilor.

Adaptarea ponderilor pentru hărţile cu auto-organizare se face astfel:

$$\Delta \mathbf{w}_i = \alpha(N_i, t) \cdot [\mathbf{x}(t) - \mathbf{w}_i(t)] \quad \text{pentru } i \in N_m(t)$$

unde $0 < \alpha(N_i, t) < 1$ este constanta de învăţare. De exemplu, Kohonen a folosit:

$$\alpha(N_i, t) = \alpha(t) \cdot e^{\frac{-\|r_i - r_m\|}{\sigma^2(t)}}$$

unde r_m este vectorul poziţiei neuronului m, r_i este vectorul poziţiei neuronului i, iar $\alpha(t)$ şi $\sigma(t)$ sunt funcţii descrescătoare în timp.

Ca rezultat al instruirii ponderilor, se obţine o hartă de neuroni în care ponderile codifică funcţiile de densitate $p(\mathbf{x})$ ale probabilităţilor pattern-urilor de instruire. Răspunsul neuronilor unei arii se întăreşte dacă apar mai multe pattern-uri similare corespunzătoare acelei arii.

Fie de exemplu 32 de vectori 5-dimensionali diferiţi, etichetaţi cu A, ..., 6 (tab. 7.1).

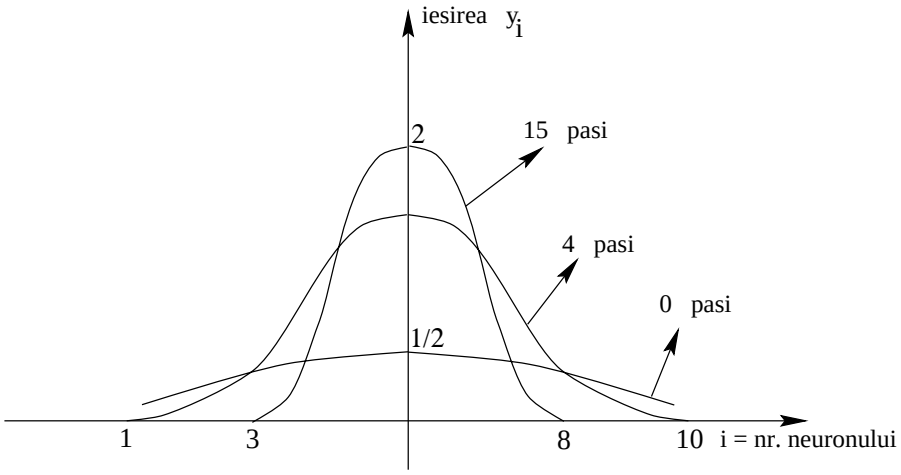


Figura 7.9: Evoluţia ieşirilor celor 10 neuroni.

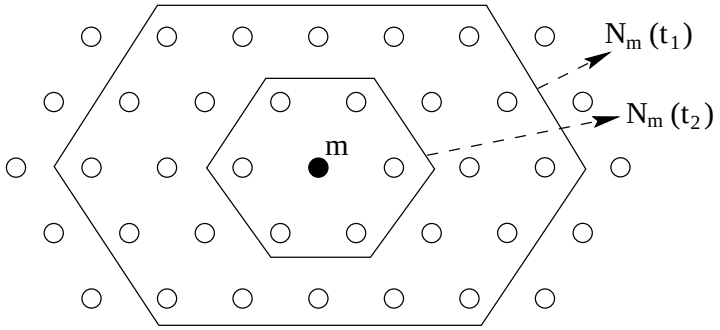


Figura 7.10: Tablou de neuroni având o topologie hexagonală.

Proiecţia se face pe un tablou de 70 de neuroni, conectaţi fiecare prin 5 ponderi cu componentele $x_1, \ldots, x_5$. Tabloul se instruieşte folosind cei 36 de vectori selectaţi în ordine aleatoare. După 10000 de paşi de instruire, ponderile se stabilizează şi reţeaua obţinută este calibrată. Calibrarea se face prin etichetare supervizată. De exemplu, având vectorul B la intrare, observăm unde apare răspunsul maxim în tabloul de neuroni şi etichetăm acel neuron cu B. Obţinem harta din fig. 7.11.

32 de neuroni vor fi etichetaţi şi 38 nu. Inspectăm similarităţile dintre vectorii de instruire. Reprezentăm vectorii ca vârfuri într-un graf. Fiecărei muchii îi ataşăm distanţa dintre cele două pattern-uri corespunzătoare vârfurilor. De exemplu, distanţa dintre A şi C este egală cu:

$$\|A - C\| = 2.$$

Construim în continuare arborele parţial de lungime minimă (fig. 7.12). Se

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
x_1	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
x_2	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
x_3	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
x_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
x_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

Tabelul 7.1: Vectori 5-dimensionali.

```
  B C D E * Q R * Y Z
A * * * * P * * X *
  * F * N O * W * * 1
* G * M * * * * 2 *
  H K L * T U * 3 * *
* I * * * * * * 4 *
  * J * S * * V * 5 6
```

Figura 7.11: Harta obţinută în urma instruirii.

poate observa că harta trăsăturilor produsă prin auto-organizare are aceeaşi structură ca şi acest arbore. Înseamnă că am reuşit să păstrăm topologia dintr-un spaţiu 5-dimensional într-un plan. Distanţa dintre vectorii $\mathbf{x}_i$ şi $\mathbf{x}_j$ din spaţiul pattern-urilor:

$$\|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{k=1}^{5}(x_{ik} - x_{jk})^2}$$

se menţine şi în spaţiul trăsăturilor, unde devine o simplă distanţă în plan.

Algoritmul de instruire ţine cont atât de relaţiile topologice dintre datele de intrare cât şi de istoricul datelor de intrare, adică de frecvenţa de apariţie a pattern-urilor similare sau identice.

Acest algoritm are numeroase aplicaţii în controlul roboţilor, recunoaşterea vorbirii etc.

Un exemplu celebru este *maşina de scris fonetică* (Kohonen, 1988), folosită în recunoaşterea vorbirii.

La intrare avem un vector 15-dimensional. Semnalul vocal este convertit prin FFT în semnal în frecvenţă acoperind pe 15 canale frecvenţele dintre 200Hz şi 5kHz. Fiecare fonem tinde să ocupe un segment al spectrului 15-dimensional Fourier. Cu toate că spectrele unor foneme diferite se suprapun parţial, un tablou de neuroni este capabil, după instruire, să extragă trăsăturile fonemelor şi să le proiecteze pe o hartă plană. Reţeaua neurală a fost instruită cu semnale spectrale generate la fiecare 9,83 ms de vorbirea continuă. Eşantioanele spectrului FFT au fost folosite în ordinea lor naturala

După instruire, anumiţi neuroni devin sensibili la spectrul diferitelor foneme (fig. 7.13).

Harta a fost calibrată folosind foneme standard. Majoritatea celulelor au

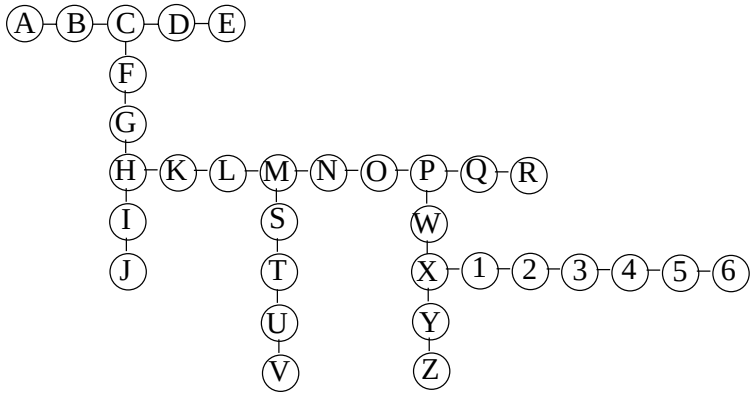


Figura 7.12: Arborele parţial de lungime minimă construit pe baza distanţelor dintre vectori.

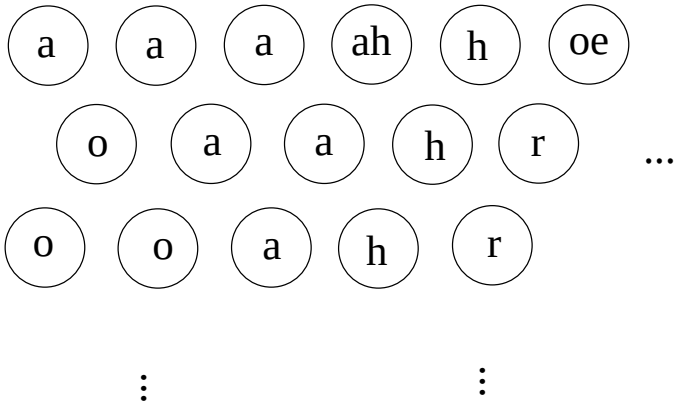


Figura 7.13: Harta fonemelor după instruire (în limba finlandeză).

învăţat un singur fonem. Există câteva celule care indică răspunsul a două foneme diferite, necesitând o discriminare suplimentară. Pe această hartă, un cuvânt generează o traiectorie.

7.4 Exerciţii

1. (C) Proiectaţi reţeaua Hammimg şi reţeaua MAXNET asociată care realizează clasificarea optimă a vectorilor binari bipolari care reprezintă caracterele A, I şi O din problema 7, capitolul 4 în termenii distanţei Hamming minime.

2. (C) Implementaţi algoritmul de instruire *câştigătorul ia tot* pentru gruparea caracterelor C, I şi T afectate de zgomot, în mod similar reprezentării folosite în problema anterioară. Presupuneţi că aceste caractere afectate de

zgomot au DH=1 faţă de prototipurile neafectate de zgomot. Calculaţi vectorii finali de ponderi $\mathbf{w}_C$, $\mathbf{w}_I$ şi $\mathbf{w}_T$. În timpul operaţiei de testare, calculaţi răspunsurile $f(net_C)$, $f(net_I)$ şi $f(net_T)$ fiecărui neuron cu funcţie de activare unipolar continuă. Intrările sunt vectori bipolari binari obţinuţi din literele reprezentate matriceal. Folosiţi $\lambda = 1$.

3. (C) Fie mulţimea de pattern-uri

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_5 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Implementaţi harta Kohonen pentru a proiecta aceşti vectori pe o reţea de 5×5 neuroni. Folosiţi vecinătăţi hexagonale. Descreşteţi pe α de la 0,5 la 0,004 pentru primii 1000 de paşi, apoi descreşteţi-l de la 0,04 la 0 pentru următorii 9000 de paşi. Descreşteţi raza vecinătăţii de la 2 la 0 pe parcursul primilor 1000 de paşi. Calibraţi apoi harta şi desenaţi arborele parţial de lungime minimă produs pe hartă pentru intrările $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$, $\mathbf{x}_5$.

Capitolul 8

Reţele neurale cu funcţie de activare radială

Aceste reţele neurale au un timp de instruire mult mai rapid decât reţelele neurale feedforward cu propagare în urmă a erorii, evitând şi problema minimului local. Dezavantajul este că, după instruire, necesită un timp de calcul mai mare în exploatare. În 1990 s-a demostrat că aceste reţele sunt aproximatori universali, ca şi reţelele feedforward cu funcţie de activare continuă. Aceasta înseamnă că pot aproxima orice funcţie continuă, oricât de bine.

O astfel de reţea este o reţea feedforward în care funcţia de activare este o funcţie de tip gaussiană. Deoarece gaussiana este simetric radială, vom numi aceste reţele "cu funcţie de activare radială". Gaussiana nu este însă decât o funcţie particulară de acest tip.

Spre deosebire de reţelele feedforward, ponderile conexiunilor către straturile ascunse nu sunt iniţializate aleator. Ele sunt iniţializate cu valorile care produc răspunsul dorit.

8.1 Funcţii radiale

Fie cazul unei singure intrări într-un neuron ascuns (fig. 8.1).

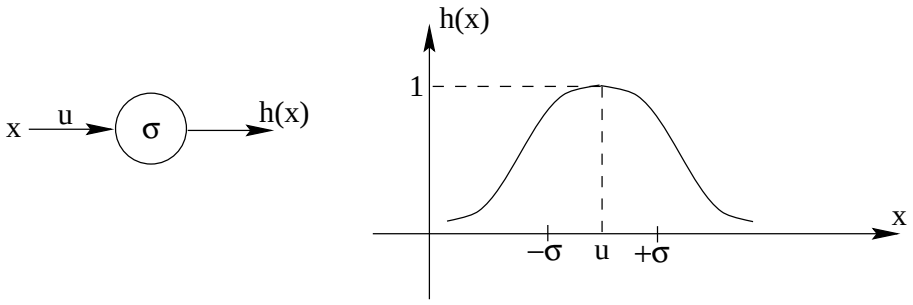


Figura 8.1: Neuron cu funcţie de activare radială.

Pentru neuronii de acest tip, avem:

$$\begin{aligned} h(x) &= e^{\frac{-(x-u)^2}{2\sigma^2}} \\ h(u) &= 1, \end{aligned}$$

unde σ este un parametru, iar u este o pondere. $h(x)$ scade foarte rapid dacă x este diferit de u.

De această dată nu mai există analogia cu potenţialul membranei, cum este cazul funcţiei de activare de forma $f(\mathbf{x}^t\mathbf{w})$. De aceea, reţelele de acest tip se numesc "radial basis function neural networks". Ponderile nu se mai înmulţesc cu intrările. Le vom numi totuşi astfel, fără a crea confuzie.

Ieşirea $h(x)$ are un răspuns semnificativ doar pentru un interval de valori ale lui x, acest interval fiind *domeniul receptiv al neuronului*. Mărimea acestui domeniu este determinată de σ. Prin analogie cu distribuţia normală, u poate fi valoarea medie, iar σ^2 dispersia funcţiei de activare a neuronului.

Forma generală a unei reţele neurale bazată pe acest tip de neuron este cea din figura 8.2.

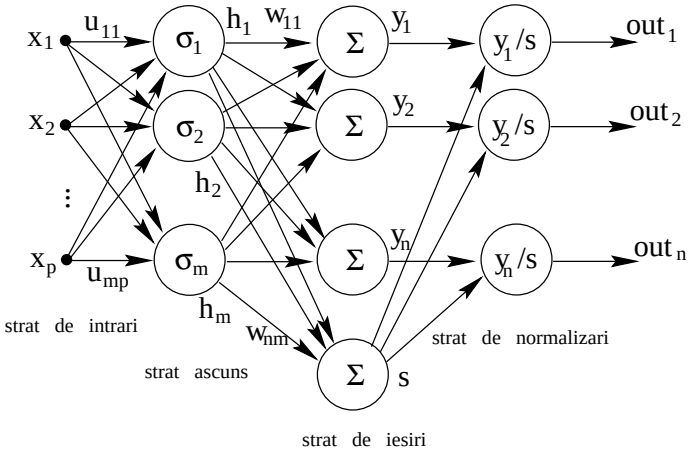


Figura 8.2: Reţea neurală cu funcţie de activare radială.

Avem:

$$h_i = e^{-\frac{(\mathbf{x}-\mathbf{u}_i)^t(\mathbf{x}-\mathbf{u}_i)}{2\sigma_i^2}}$$

unde:

$$\mathbf{x} = [\ x_1 \quad \ldots \quad x_p\ ]^t,$$

σ_i este parametrul pentru al i-lea neuron ascuns şi

$$\mathbf{u}_i = [\ u_{i1} \quad \ldots \quad u_{im}\ ]^t$$

este vectorul ponderilor dintre intrări şi al i-lea neuron ascuns.

Fiecărui pattern învăţat îi corespunde un neuron ascuns.

În stratul de ieşire se calculează:

$$y_j = \sum_{i=1}^{m} h_i w_{ji},$$

iar pe ultimul strat, cel al normalizărilor, se calculează ieşirile finale:

$$\begin{aligned} s &= \sum_{j=1}^{n} y_j \\ out_i &= \frac{y_i}{s}. \end{aligned}$$

Aceste din urmă normalizări sunt opţionale.

Domeniul receptiv al celui de-al i-lea neuron ascuns este determinat de centrul său, $\mathbf{u}_i$, şi de mărimea sa, dată de σ_i. La neuronii studiaţi până acum, acest domeniu era determinat de conexiunile neurale, nu de forma funcţiei de activare. Se pot folosi şi alte funcţii de activare. Esenţial este ca ele să tindă rapid către zero atunci când distanţa dintre vectorii $\mathbf{x}$ şi $\mathbf{u}$ creşte, iar răspunsul maxim să îl aibă când $\mathbf{x} = \mathbf{u}$.

Reţeaua trebuie instruită prin fixarea parametrilor σ_i şi a ponderilor de tip $\mathbf{u}$ şi $\mathbf{w}$. Instruirea se face în două faze. La început se atribuie valori pentru $\mathbf{u}_i$ şi σ_i, $i = 1, \ldots, m$. Apoi, se instruieşte matricea $\mathbf{W}$ prin învăţare supervizată.

Fixarea centrilor domeniilor receptive

Trebuie să fixăm centrii domeniilor receptive. Fixarea ponderilor $\mathbf{u}_i$ se poate face în mai multe feluri. Putem atribui fiecărui vector din secvenţa de instruire câte un neuron ascuns, cu centrul corespunzător. Deoarece vectorii sunt, în general, dispuşi în grupări (clustere), această metodă duce la un număr prea mare de neuroni ascunşi. Mai bine este să găsim centrul fiecărui cluster de vectori din secvenţa de instruire şi să atribuim fiecărui astfel de centru câte un neuron ascuns.

Fixarea diametrelor domeniilor receptive

Diametrul domeniului receptiv, determinat de valoarea lui σ, poate avea un efect foarte profund asupra reţelei. Obiectivul este să acoperim spaţiul pattern-urilor de intrare, cât mai uniform, cu domenii receptive. Dacă distanţa dintre centrii domeniilor receptive nu este uniformă, este necesar ca fiecare neuron ascuns să aibă o valoare specifică pentru σ. Dacă centrii domeniilor receptive sunt la distanţă mare, atunci σ trebuie să fie suficient de mare pentru a acoperi golurile; dacă centrii sunt apropiaţi, atunci neuronii respectivi trebuie să aibă σ mic. Aceasta este ca o egalizare de histogramă, procedeu cunoscut în procesarea imaginilor.

De exemplu, putem fixa σ astfel:

1. Pentru fiecare neuron ascuns, găseşte distanţa medie dintre centrul său, $\mathbf{u}_i$ şi centrii celor N vecini ai săi cei mai apropiaţi.

2. Atribuie această valoare lui σ_i

Instruirea matricii W

1. *Se aplică vectorul* $\mathbf{x}$ *la intrare, unde* $\mathbf{x}$ *este din secvenţa de instruire.*

2. *Se calculează ieşirile* $\mathbf{h}$ *din stratul ascuns.*

3. *Se calculează ieşirile* $\mathbf{y}$ *şi se compară cu vectorul* $\mathbf{t}$ *al ieşirilor dorite. Se ajustează* W *astfel:*

$$w_{ij}(k+1) = w_{ij}(k) + \eta(t_i - y_i)x_j,$$

   unde η *este coeficientul de învăţare. De obicei* $\eta << 1$*, iar aceasta este de fapt regule Windrow-Hoff. Se pot folosi şi alte reguli, de exemplu regula perceptronului.*

4. *Repetă 1-3 pentru fiecare pattern de instruire.*

5. *Repetă 1-4 până când eroarea este suficient de mică.*

Deoarece domeniile receptive sunt limitate, ajustările apar destul de rar, ceea ce conduce la o viteză mare.

Algoritmul de instruire a lui $\mathbf{W}$ se referă la un singur strat, iar pattern-urile corespunzătoare neuronilor din stratul ascuns sunt liniar separabile, deoarece fiecare clasă este formată dintr-un pattern. Rezultă că algoritmul de instruire a lui $\mathbf{W}$ converge către minimul global, conform teoremei perceptronului. Practic, convergenţa este de aproximativ 1000 de ori mai rapidă decât în cazul instruirii prin propagarea în urmă a erorii.

8.2 O tehnică de grupare (clustering)

Am văzut că este avantajos să obţinem câte un reprezentant pentru fiecare cluster şi să instruim apoi reţeaua doar cu aceşti reprezentanţi. Putem pondera reprezentanţii cu numărul de pattern-uri pe care îi reprezintă:

$$y_j = \frac{\sum_{i=1}^{n} m_i h_i w_{ji}}{\sum_{i=1}^{n} m_i h_i}$$

unde m_i este numărul de pattern-uri din clusterul i.

Iată o tehnică de clustering (Specht, 1991) simplă şi eficientă. Se defineşte o rază r. Primul pattern de instruire devine centrul primului cluster. Fiecare pattern din secvenţa de instruire este considerat pe rând. Dacă distanţa dintre pattern-ul considerat şi cel mai apropiat cluster este mai mică sau egală decât r, atunci pattern-ul este atribuit acelui cluster; dacă distanţa este mai mare decât r, pattern-ul devine primul membru al unui nou cluster.

Procedeul de clustering este folositor atunci când există un exces de instruire cu date care formează clustere dense.

8.3 Discuţie

Reţelele cu funcţie de activare radială sunt foarte rapide la instruire, dar lente la utilizare. Procedeul de clustering poate fi util. Foarte importantă este alegerea corectă a lui σ.

Capitolul 9

Reţele neurale fuzzy

9.1 Logica fuzzy

9.1.1 De ce logică fuzzy?

În 1985, Hitachi a instalat în Japonia un metrou controlat prin logică fuzzy. Trenurile accelerează mai rapid şi uniform, economisind 10% energie comparativ cu soluţia convenţională.

În Vest, interesul este ceva mai mic. Explicaţia este probabil şi tradiţia logicii aristotelice binare care spune că ceva există sau nu există. Filosofia şi religia orientale se bazează pe principiul că ceva există şi, în acelaşi timp, nu există, de exemplu YIN şi YANG.

Atunci când nu au putut converti binar o informaţie conform principiului true/false, occidentalii au apelat la statistică. Astfel, probabilitatea ca o maşină să fie lentă sau rapidă se exprimă printr-un număr între 0 şi 1. Logica fuzzy produce acelaşi rezultat, etichetând maşinile în funcţie de viteză cu numere între 0 şi 1. Aceasta a produs o mare confuzie, neînţelegându-se diferenţa dintre cele două demersuri. Vom vedea însă că diferenţa este fundamentală.

Iniţiatorul logicii fuzzy este, paradoxal, un occidental: Zadeh în 1965.

Controlere fuzzy

Sistemele fuzzy au fost folosite foarte des în controlere. Aceste controlere sunt uşor de proiectat, uneori în câteva ore. Performanţele lor sunt mai bune decât în cazul controlerelor clasice. Proiectarea unui sistem fuzzy nu necesită cunoştinţe de logică fuzzy. Vom demonstra aceasta proiectând un controler fuzzy pentru un lift.

Primul pas în proiectarea unui sistem fuzzy este să decidem asupra unei mulţimi de reguli fuzzy. În cazul nostru, ele sunt:

1. Dacă liftul se ridică lent şi poziţia sa este departe de poziţia dorită, atunci măreşte viteza.

2. Dacă liftul se ridică cu o viteză medie şi este aproape de poziţia dorită, atunci micşorează viteza.

Exprimăm acelaşi lucru prin reguli IF-THEN:

1. IF viteza este SP (small positive) AND poziţia este LN (large negative) THEN tensiunea motorului este LP (large positive).

2. IF viteza este MP (medium positive) AND poziţia este SN (small negative) THEN tensiunea motorului este SP (small positive).

Sarcina controlerului fuzzy este să proceseze aceste două reguli, eventual şi altele, după ce a determinat poziţia şi acceleraţia.

O regulă are două părţi: partea *antecedentă* şi cea *consecventă*. Partea antecedentă poate conţine mai multe *clauze* legate prin AND, OR şi NOT. O clauză este, de exemplu "viteza este SP". Clauzele pot fi descompuse în *variabile* şi *adjective*. De exemplu, variabile sunt "viteza" şi "poziţia", iar adjective sunt SP, LN şi LP.

Gradul în care o regulă este relevantă într-o situaţie măsurată este acelaşi cu gradul în care antecedentul regulii este adevărat în sens fuzzy. Valoarea de adevăr fuzzy a unui antecedent depinde de valoarea de adevăr fuzzy a fiecărei clauze ale sale şi de operaţiile logice care leagă aceste clauze.

În logica fuzzy, fiecărui adjectiv (cum este şi SP) i se atribuie o funcţie de apartenenţă. Valoarea funcţiei de apartenenţă este între 0 şi 1. Dată fiind valoarea unei variabile, funcţia de apartenenţă este folosită pentru a calcula valoarea de adevăr a acelei variabile.

Exemplu: Un bărbat de 1,60 m în mod cert nu este "înalt", deci funcţia de apartenenţă la mulţimea bărbaţilor înalţi a acestuia are valoarea 0. Pentru 1,75m avem o valoare între 0 şi 1.

Folosind funcţiile de apartenenţă fuzzy, fiecărei clauze dintr-un antecedent i se poate atribui o valoare de adevăr fuzzy. Mai trebuie acum să atribuim o valoare de adevăr întregului antecedent.

Exemplu: Funcţia de apartenenţă a vitezei la SP are valoarea 0,6 şi funcţia de apartenenţă a poziţiei la LN are valoarea 0,8, deci valoarea de adevăr a întregului antecedent este 0,6 pentru că se consideră valoarea minimă.

Pasul final constă în combinarea valorilor de ieşire pentru cele două reguli. Acest proces se numeşte *defuzzificare*. La acest proces, nu există un consens asupra modului de operare.

Tipul de defuzzificare ilustrat în figura 9.1 se numeşte defuzzificare min-max.

Într-un sistem fuzzy, toate regulile acţionează simultan asupra ieşirii. La ieşire avem tensiunea corespunzătoare centrului de greutate al sumei regiunilor

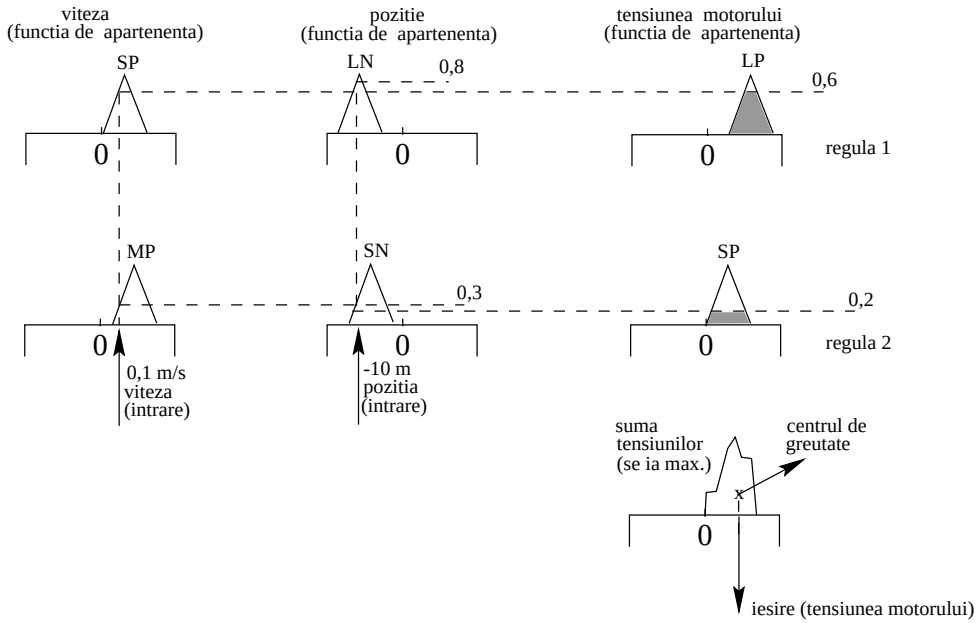


Figura 9.1: Defuzzificare min-max.

haşurate. Trebue să facem observaţia că forma funcţiei de apartenenţă este dată de proiectant. De obicei se alege o funcţie triunghiulară sau trapezoidală.

Logica fuzzy generalizează logica binară, permiţând valori reale între 0 şi 1.

9.1.2 Logica fuzzy, logica convenţională şi teoria mulţimilor

Ca şi teoria mulţimilor, logica fuzzy se ocupă de conceptul apartenenţei la o mulţime. În teoria clasică a mulţimilor, un element poate sau nu poate să aparţină unei mulţimi, ceea ce este nenatural.

Exemplu: Un bărbat de 1,80 m poate avea un grad de apartenenţă de 0,6 la mulţimea bărbaţilor înalţi. Aceasta înseamnă şi că acest bărbat are un grad de apartenenţă de 0,4 la mulţimea bărbaţilor care "nu sunt înalţi". Acest bărbat aparţine deci simultan celor două mulţimi complementare, ceea ce ar fi imposibil în teoria convenţională.

Logica convenţională defineşte o funcţie de apartenenţă la o mulţime $\mathcal{A}$ astfel: $f_{\mathcal{A}}(x) \in \{0, 1\}$, unde 0 înseamnă $x \notin \mathcal{A}$, iar 1 înseamnă $x \in \mathcal{A}$. În logica fuzzy, funcţia de apartenenţă devine: $f_{\mathcal{A}}(x) \in [0, 1]$.

Ideea de a vedea o mulţime ca o funcţie capabilă să definească grade de apartenenţă stă la baza teoriei mulţimilor fuzzy. Evident, este nevoie de o nouă matematică, cea introdusă de Zadeh.

Logica convenţională şi logica fuzzy

Logica convenţională foloseşte AND, OR şi NOT. Orice relaţie poate fi descrisă prin combinarea acestor operatori. Funcţiile logice fuzzy operează asupra unor valori de adevăr în $[0, 1]$. Pentru valorile 0 şi 1, o astfel de funcţie trebuie să dea acelaşi rezultat ca şi în logica convenţională. Evident, există o infinitate de funcţii care să satisfacă aceste cerinţe. Trebuie alese cele mai avantajoase.

Funcţia AND poate fi implementată astfel:

$$and(X, Y) = min(X, Y)$$

unde $X, Y \in [0, 1]$.

Funcţia OR: $or(X, Y) = max(X, Y)$.

Funcţia NOT: $not(X) = 1 - X$.

Teoria clasică şi fuzzy a mulţimilor

Fie $m_{\mathcal{A}}(x)$ valoarea funcţiei de apartenenţă a punctului x la mulţimea $\mathcal{A}$ şi $m_{\mathcal{B}}(x)$ valoarea funcţiei de apartenenţă a punctului x la mulţimea $\mathcal{B}$. Definim:

$$m_{\mathcal{A}\cap\mathcal{B}}(x) = min[m_{\mathcal{A}}(x), m_{\mathcal{B}}(x)].$$

Este o definiţie compatibilă cu definiţia funcţiei AND. Este relaţia pe care am folosit-o în exemplul cu controlerul. Similar:

$$\begin{aligned} m_{\mathcal{A}\cup\mathcal{B}}(x) &= max[m_{\mathcal{A}}, m_{\mathcal{B}}] \\ m_{\mathcal{A}^c}(x) &= 1 - m_{\mathcal{A}}(x) \end{aligned}$$

unde $\mathcal{A}^c$ este complementul lui $\mathcal{A}$.

Ca exerciţiu, calculaţi intersecţia şi reuniunea mulţimilor $\mathcal{A}$ şi $\mathcal{A}^c$.

Mulţimi fuzzy şi paradoxuri logice

Bertrand Russell a dat următorul paradox al bărbierului: Într-un oraş există un bărbier care bărbiereşte pe oricine nu se bărbiereşte singur. Problema este dacă bărbierul se bărbiereşte singur. Apare o contradicţie logică, un paradox. Dacă propoziţia P:"se bărbiereşte singur" are valoarea de adevărat, $truth(P)$, atunci $truth(P) = truth(notP)$, deoarece bărbierul se bărbiereşte şi, în acelaşi timp, nu se bărbiereşte singur. Folosind complementul fuzzy,

$$truth(notP) = 1 - truth(P).$$

Atunci,

$$truth(P) = 1 - truth(P),$$

deci

$$truth(P) = 0,5.$$

În logica fuzzy, propoziţiile pot avea valori de adevărat oriunde între 0 şi 1, deci paradoxul lui Russell nu mai există.

Desigur, eliminând toate restricţiile asupra imaginaţiei, putem practic explica orice. De exemplu, să considerăm superstiţiile popoarelor primitive care explică fenomenele naturale cu ajutorul entităţilor supranaturale. Logicienii convenţionali acuză teoreticienii fuzzy de "construcţie artificală". Teoriile matematice trebuie însă judecate prin consistenţa, frumuseţea şi utilitatea lor. Teoria fuzzy satisface toate aceste cerinţe.

Observaţii matematice

Fie $\mathcal{X}$ o mulţime oarecare. Fie $f_{\mathcal{A}} : \mathcal{X} \to [0,1]$ o funcţie. Funcţia $f_{\mathcal{A}}$ defineşte o *mulţime fuzzy* (în $\mathcal{X}$), adică mulţimea $\mathcal{A} = \{(x, f_{\mathcal{A}}(x) | x \in \mathcal{X}\}$. Funcţia $f_{\mathcal{A}}$ este funcţia de apartenenţă la mulţimea $\mathcal{A}$.

Apare un abuz de limbaj între "mulţimea" fuzzy $\mathcal{A}$ şi "funcţia" de apartenenţă la $\mathcal{A}$. Acest abuz se datorează faptului că definim o mulţime prin gradul de apartenenţă la această mulţime, nu prin enumerarea mulţimii, cum suntem obişnuiţi.

9.2 Reţele neurale fuzzy

9.2.1 Neuroni fuzzy

Un neuron non-fuzzy are N intrări ponderate şi o ieşire. Ieşirea se calculează astfel:

$$y = f\left(\sum_{i=1}^{N} w_i x_i - T\right)$$

unde f este funcţia de activare, în general neliniară. Un neuron fuzzy (NF) (fig. 9.2) are N intrări ponderate şi M ieşiri.

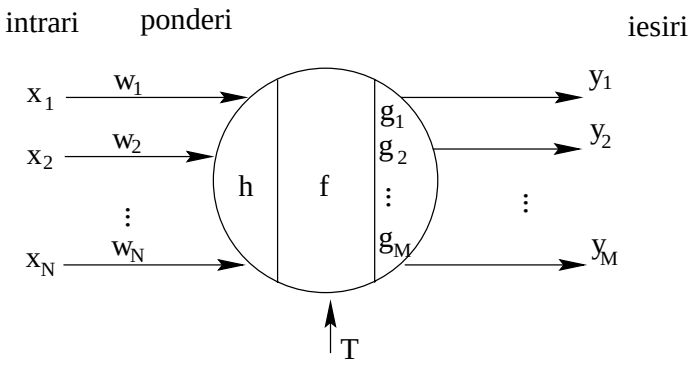


Figura 9.2: Neuronul fuzzy.

Toate intrările şi ponderile sunt valori reale, iar ieşirile sunt valori reale în intervalul [0,1]. Fiecare ieşire poate fi asociată gradului de apartenenţă (în sensul fuzzy), adică exprimă în ce măsură pattern-ul cu intrările $\{x_1, x_2, \ldots, x_N\}$ aparţine unei mulţimi fuzzy. Avem:

$$z = h(w_1x_1, w_2x_2, \ldots, w_Nx_N)$$

unde z este intrarea în NF, iar h este funcţia de agregare,

$$s = f(z - T),$$

unde s este starea NF, f este funcţia de activare, iar T este pragul,

$$y_j = g_j(s), \quad j = 1, \ldots, M,$$

unde g_j, $j = 1, \ldots, M$ sunt funcţiile de ieşire ale NF reprezentând gradele de apartenenţă ale lui $\{x_1, \ldots, x_N\}$ la cele M mulţimi fuzzy. În general, ponderile, pragul şi funcţiile de ieşire sunt modificabile în timpul procesului de instruire. Funcţia de agregare şi funcţia de activare sunt fixe. Se pot defini multe tipuri de NF prin schimbarea funcţiilor f şi h.

Definim acum patru tipuri de NF.

1. un *NF de intrare* este un NF folosit în stratul de intrare a unei RNF (reţele de neuroni fuzzy), având doar o singură intrare x, astfel încât $z = x$.

2. un *MAX-NF* are funcţia de agregare:

$$z = max_{i=1,\ldots,N}(w_ix_i).$$

3. un *MIN-NF* are funcţia de agregare:

$$z = min_{i=1,\ldots,N}(w_ix_i).$$

4. un *COMP-NF* (NF competitiv) are pragul variabil şi o singură ieşire, astfel încât:

$$\begin{aligned} y &= g(s - T) = \begin{cases} 0 & \text{dacă } s < T \\ 1 & \text{dacă } s \geq T \end{cases} \\ T &= t(c_1, c_2, \ldots, c_K), \end{aligned}$$

unde s este starea NF, t este funcţia prag, iar c_k, $k = 1, \ldots, K$ sunt variabile competitive ale NF.

9.2.2 Structura unei RNF

RNF propusă[1] este pe patru straturi. Primul strat, de intrare, foloseşte neuroni fuzzy de intrare. Presupunând că dorim să recunoaştem caractere, fiecare NF din acest strat corespunde unui pixel din pattern-ul de intrare. Dacă fiecare pattern de intrare are $N_1 \times N_2$ pixeli, atunci primul strat are $N_1 \times N_2$ NF de intrare. Pentru al (i,j)-lea NF de intrare din primul strat avem:

$$s_{ij}^{[1]} = z_{ij}^{[1]} = x_{ij} \quad \text{pentru} \quad i = 1,\ldots,N_1, \quad j = 1,\ldots,N_2$$
$$y_{ij}^{[1]} = \frac{s_{ij}^{[1]}}{P_{vmax}} \quad \text{pentru} \quad i = 1,\ldots,N_1, \quad j = 1,\ldots,N_2$$

unde x_{ij} este valoarea pixelului (i,j), $x_{ij} \geq 0$, iar P_{vmax} este valoarea maximă a pixelilor pentru toate pattern-urile de intrare.

Al doilea strat constă din $N_1 \times N_2$ MAX-NF. Acest al doilea strat realizează fuzzificarea pattern-urilor de intrare prin funcţia pondere $w(m,n)$. Starea celui de-al (p,q)-lea MAX-NF din al doilea strat este:

$$s_{pq}^{[2]} = max_{i=1,\ldots,N_1}(max_{j=1,\ldots,N_2}(w(p-i,q-j)y_{ij}^{[1]})),$$

pentru $p = 1,\ldots,N_1$ şi $q = 1,\ldots,N_2$, unde $w(p-i,q-j)$ este ponderea conexiunii dintre al (i,j)-lea NF de intrare din primul strat şi al (p,q)-lea MAX-NF din al doilea strat:

$$w(m,n) = e^{-\beta^2(m^2+n^2)}$$

pentru $m = -(N_1-1),\ldots,(N_1-1)$ şi $n = -(N_2-1),\ldots,(N_2-1)$. Prin această funcţie pondere care are o formă de gaussiană, fiecare NF din al doilea strat este ca o lentilă, astfel încât fiecare NF focalizează pe un pixel din pattern-ul de intrare, dar "vede" şi pixelii vecini. Parametrul β determină câţi pixeli "vede" un NF, aceasta fiind decis prin algoritmul de instruire. Funcţia w se numeşte şi *funcţie de fuzzificare*. Fiecare MAX-NF din acest strat are M ieşiri diferite, câte una pentru fiecare NF din al treilea strat. Ieşirile celui de-al (p,q)-lea MAX-NF din al doilea strat sunt:

$$y_{pqm}^{[2]} = g_{pqm}(s_{pq}^{[2]}),$$

pentru $p = 1,\ldots,N_1$, $q = 1,\ldots,N_2$, $m = 1,\ldots,M$, unde $y_{pqm}^{[2]}$ este a m-a ieşire a celui de-al (p,q)-lea MAX-NF către al m-lea MIN-NF din stratul trei. Funcţia de ieşire g_{pqm} se determină prin algoritmul de instruire. Pentru simplitate, putem alege triunghiuri isoscele de înălţime 1 şi bază α (fig. 9.3).

Avem:

$$y_{pqm}^{[2]} = g_{pqm}(s_{pq}^{[2]}) = \begin{cases} 1 - \frac{2|s_{pq}^{[2]} - \theta_{pqm}|}{\alpha} & \text{pentru } |s_{pq}^{[2]} - \theta_{pqm}| \leq \frac{\alpha}{2} \\ 0 & \text{în rest} \end{cases},$$

[1] Kwan, H.K., Cai.Y "A Fuzzy Neural Network and its Application to Pattern Recognition", IEEE Trans. on Fuzzy Systems, 2, 1994, 185-193.

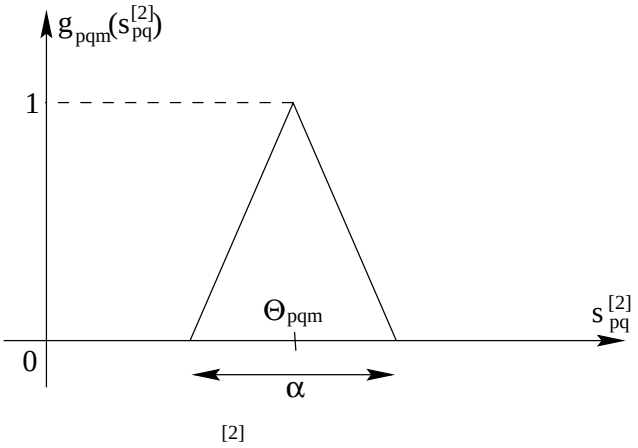


Figura 9.3: Funcţia de ieşire g_{pqm}.

pentru $\alpha \geq 0$, $p = 1, \ldots, N_1$, $q = 1, \ldots, N_2$, $m = 1, \ldots, M$, unde θ_{pqm} este mijlocul bazei triunghiului. Prin algoritmul de instruire se determină pentru fiecare p, q, m valorile corespunzătoare pentru α şi θ_{pqm}.

Al treilea strat constă din M neuroni de tip MIN-NF. Fiecare din aceşti neuroni reprezintă un pattern învăţat. Deci, M poate fi determinat doar la încheierea procesului de instruire. Ieşirea celui de-al m-lea MIN-NF este:

$$y_m^{[3]} = s_m^{[3]} = min_{p=1,\ldots,N_1}(min_{q=1,\ldots,N_2}(y_{pqm}^{[2]}))$$

pentru $m = 1, \ldots, M$.

Al patrulea strat este stratul de ieşire şi constă din M neuroni de tip COMP-NF, câte unul pentru fiecare pattern învăţat. Dacă un pattern de intrare este cel mai asemănător cu al m-lea pattern învăţat, atunci ieşirea celui de-al m-lea COMP-NF este 1, iar celelalte ieşiri sunt 0. Avem:

$$\begin{aligned}
&s_m^{[4]} = z_m^{[4]} = y_m^{[3]} \text{ pentru } m = 1, \ldots, M \\
&y_m^{[4]} = g[s_m^{[4]} - T] = \begin{cases} 0 & \text{dacă } s_m^{[4]} < T \\ 1 & \text{dacă } s_m^{[4]} = T \end{cases} \text{ pentru } m = 1, \ldots, M \\
&T = max_{m=1,\ldots,M}(y_m^{[3]}) \text{ pentru } m = 1, \ldots, M
\end{aligned}$$

unde T este pragul de activare pentru toţi neuronii din cel de-al patrulea strat.

9.2.3 Algoritmul de instruire a unei RNF

Prin instruire, trebuie să determinăm următorii parametri: α, θ_{pqm}, β şi M. Definim T_f, $0 \leq T_f \leq 1$, ca toleranţa la eroare a RNF. Fie K numărul total de pattern-uri de instruire.

Pasul 1. Creează $N_1 \times N_2$ neuroni de tip NF de intrare în primul strat şi $N_1 \times N_2$ neuroni de tip MAX-NF în al doilea strat. Alege o valoare pentru α, $\alpha \geq 0$ şi o valoare pentru β.

Pasul 2. $M = 0$, $k = 1$.

Pasul 3. $M = M + 1$. Creează al M-lea MIN-NF în al treilea strat şi al M-lea COMP-NF în al patrulea strat.

$$\theta_{pqM} = s_{pq}^{[2]} = max_{i=1,\ldots,N_1}(max_{j=1,\ldots,N_2}(w(p-i,q-j)x_{ijk}))$$

pentru $p = 1,\ldots,N_1$ şi $q = 1,\ldots,N_2$, unde θ_{pqM} este mijlocul bazei triunghiului pentru cea de-a M-a funcţie de ieşire a celui de-al (p,q)-lea MAX-NF în stratul 2. $X_k = \{x_{ijk}\}$, $i = 1,\ldots,N_1$, $j = 1,\ldots,N_2$ este al k-lea pattern de instruire. Presupunem că valorile x_{ijk} sunt deja normalizate.

Pasul 4. $k = k + 1$. Dacă $k > K$, algoritmul de instruire se termină.
Altfel, acceptă la intrarea în reţea al k-lea pattern şi calculează ieşirea RNF curente, cu M neuroni în straturile 3 şi 4.

$$\sigma = 1 - max_{j=1,\ldots,M}(y_{jk}^{[3]}),$$

unde $y_{jk}^{[3]}$ este ieşirea celui de-al j-lea MIN-NF din stratul 3 pentru al k-lea pattern de instruire X_k.
Dacă:
$\sigma \leq T_f$, `goto` Pasul 4.
$\sigma > T_f$, `goto` Pasul 3.

9.2.4 Analiza RNF

Primul strat de neuroni acceptă datele de intrare şi transformă valoarea pixelilor în valori normalizate, în intervalul [0,1].

Al doilea strat fuzzifică pattern-ul de intrare. Valoarea unui pixel în pattern-ul de intrare va afecta stările mai multor NF din stratul 2. Cu cât β este mai mic, cu atât mai mulţi NF din stratul 2 vor fi afectaţi de un pixel din pattern-ul de intrare. Deci, β controlează gradul de fuzzificare. Dacă β este prea mic, RNF nu va putea separa pattern-urile distincte. Dacă β este prea mare, RNF îşi pierde abilitatea de a recunoaşte pattern-uri perturbate. Valoarea lui β trebuie aleasă astfel încât toate pattern-urile distincte să poată fi separate, iar RNF să aibă o rată de recunoaştere acceptabilă.

Valoarea $y_{pqm}^{[2]}$ exprimă conceptul fuzzy cu privire la gradul în care valoarea pixelilor în jurul pixelului (p,q) din pattern-ul de intrare sunt similare cu valoarea pixelilor în jurul pixelului (p,q) din cel de-al m-lea pattern învăţat. Funcţia de ieşire g_{pqm} pentru neuronii din cel de-al doilea strat este, de fapt, o funcţie de apartenenţă fuzzy: $g_{pqm}(s_{pq}^{[2]})$ arată gradul de apartenenţă al pattern-ului de intrare la clasa reprezentată de al m-lea pattern învăţat, apartenenţa referindu-se la vecinătăţile corespunzătoare ale pixelului (p,q).

Neuronii din al treilea strat determină similarităţile dintre pattern-ul de intrare şi pattern-urile învăţate. Avem de fapt:

$$y_m^{[3]} = \begin{cases} min_{p,q}(1 - \frac{2}{\alpha}|s_{pq}^{[2]} - \theta_{pqm}|) & \text{pentru } max_{p,q}(|s_{pq}^{[2]} - \theta_{pqm}|) \leq \frac{\alpha}{2} \\ 0 & \text{în rest} \end{cases}$$

pentru $m = 1, \ldots, M$. Valoarea $y_m^{[3]}$ măsoară similaritatea pattern-ului de intrare cu al m-lea pattern învăţat. Dacă pattern-ul de intrare a fost deja învăţat, va exista un m pentru care $y_m^{[3]} = 1$. Dacă pattern-ul de intrare nu a fost învăţat, toate valorile $y_m^{[3]}$, $m = 1, \ldots, M$ vor fi mai mici decât 1. Valoarea lui α trebuie să acopere toate valorile posibile la nivel de pixel.

Stratul de ieşire este folosit pentru defuzzificare. Deci, procedura de recunoaştere prin RNF constă din:

1. datele de intrare (stratul 1)

2. fuzzificare (stratul 2)

3. deducţie fuzzy (stratul 3)

4. defuzzificare (stratul 4)

Straturile 3 şi 4 sunt construire prin instruirea reţelei. Dacă un pattern de intrare este similar cu un pattern deja învăţat, el este tratat ca acel pattern, fără a mai fi învăţat şi el. În caz contrar, pattern-ul de intrare este învăţat. Procesul de instruire poate fi continuu: RNF nu trebuie în mod necesar instruită complet înainte de a fi folosită; ea poate si instruită şi pe parcursul folosirii.

Parametrii α, β şi T_f trebuie fixaţi convenabil.

9.2.5 Rezultatele simulării

În implementarea autorilor, literele şi cifrele au fost construite ca pattern-uri de 16×16 pixeli, pixelii având doar valori binare. Pattern-urile au fost apoi mişcate prin deplasări în cele opt direcţii cu unul şi doi pixeli. Cele 36 de pattern-uri corecte şi câteva din pattern-urile obţinute prin deplasări au fost folosite pentru a instrui reţeaua.

Pentru a testa performanţele reţelei, s-au folosit pattern-uri perturbate prin: mărire/micşorare, subţiere/îngroşare, adăugarea/extragerea unor mici porţiuni etc.

S-au folosit diferite valori pentru α, β, T_f la instruire:

$$1,5 \leq \alpha \leq 3,5$$

$$0,1 \leq T_f \leq 0,35$$

$$0,1 \leq \beta \leq 1,6.$$

După instruire, s-a testat RNF la recunoaşterea unor pattern-uri perturbate, rezultatele fiind foarte bune.

9.2.6 Concluzii

Timpul de instruire a RNF este mai mic decât în cazul instruirii prin propagarea în urmă a erorii. Recunoaşterea este şi ea mai rapidă. Să observăm că această RNF învaţă nesupervizat, fiind în esenţă un algoritm de clustering. Ulterior instruirii, RNF este folosită ca un clasificator. Există variante ale acestei RNF care învaţă în mod supervizat[2].

[2]Cai, Y., H.K. Kwan "Fuzzy Classification Using Fuzzy Inference Networks". IEEE Trans. Syst., Man, and Cyb., Part B: Cybernetics, 28, 1998, 334-374.

Capitolul 10

Algoritmi genetici

Natura a demonstrat puterea geneticii. Cei ce vor să multiplice performanţele sistemelor biologice, chiar parţial, nu pot ignora modul în care acestea s-au dezvoltat, cum au evoluat.

Algoritmii genetici sunt mijloace prin care maşinile pot emula mecanismele selecţiei naturale. Aceşti algoritmi sunt simpli, robuşti şi generali. Cu reţelele neurale au în comun următoarele:

- nu sunt necesare cunoştinţe asupra spaţiului de căutare
- consumă uneori mult timp
- sunt paraleli prin natura lor
- ambele metode se aplică unor probleme similare.

De aceea, de multe ori algoritmii genetici şi reţelele neurale se aplică împreună. De exemplu, algoritmii genetici se pot aplica pentru accelerarea procesului de instruire a reţelelor neurale. În mod ciudat, printr-un algoritm genetic se poate obţine algoritmul genetic care rezolvă o problemă dată.

10.1 Introducere

Metodele convenţionale de optimizare se bazează pe ajustarea parametrilor unui model pentru a produce un rezultat dorit. La reţelele neurale, prin instruire se modifică iterativ ponderile conexiunilor, astfel încât să se producă relaţia dorită între intrări şi ieşiri. De remarcat că este menţinută o singură soluţie care este optimizată în mod iterativ.

Algoritmii genetici operează optimizări conform tehnicilor cunoscute în genetica biologică: se modifică şi se menţin caracteristicile unei *populaţii* de soluţii (indivizi) pe parcursul unui mare număr de generaţii. Acest proces produce populaţii succesive având un număr din ce în ce mai mare de indivizi cu caracteristicile dorite. Ca şi în natură, procesul este probabilistic, dar nu complet aleator. După cum vom vedea, regulile geneticii reţin caracteristicile dorite prin

maximizarea probabilităţii de proliferare a soluţiilor (indivizilor) care au aceste caracterisitci.

Genetica nu optimizează, ci îmbunătăţeşte. Un individ nu trebuie să fie optim pentru a supravieţui; el trebuie să fie superior altor indivizi din populaţia respectivă.

Algoritmii genetici operează asupra unor codificări de parametri, nu direct asupra parametrilor. Codificările sunt şiruri de lungime finită, de obicei binare. Fiind dată o populaţie iniţială de şiruri, un algoritm genetic produce o nouă populaţie de şiruri folosind o mulţime de reguli genetice. Regulile sunt astfel construite încât această nouă generaţie tinde să aibă şiruri superioare celor din generaţia precedentă, din punct de vedere al unei funcţii obiectiv.

10.2 Exemplu

Dorim să minimizăm valoarea unui şir binar, considerat aici ca un număr binar. Vom efectua următorii paşii:

1. iniţializare
2. reproducere
3. încrucişare
4. mutaţie
5. repetă paşii 2-4.

Iniţializare

Fiecare şir este generat ca un număr binar aleator. Pornim cu următoarele şase şiruri:

i	şirul A_i	val. zecimală	f_i	p_i
1	011010	26	0,0385	0,142
2	001011	11	0,0909	0,334
3	110110	54	0,0185	0,068
4	010011	19	0,0526	0,194
5	100111	39	0,0256	0,094
6	010110	22	0,0455	0,168
Total			0,2716	1

Ca funcţie obiectiv se alege (val. zecimală)$^{-1}$.

Reproducere prin selecţie

Se obţine o a doua generaţie de şiruri care au, în medie, funcţia obiectiv mai mare. Fiecare şir A_i este copiat în generaţia următoare de un număr de ori care

este proporţional cu p_i, unde:

$$p_i = \frac{f_i}{\sum_j f_j},$$

f_i fiind funcţia obiectiv, iar p_i este funcţia obiectiv relativă. "Discul norocului" (fig. 10.1) este reprezentarea grafică a acestor funcţii.

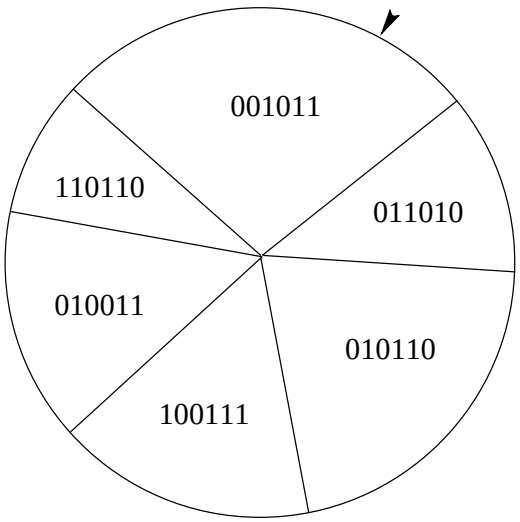


Figura 10.1: "Discul norocului". Secţiunile sunt proporţionale cu p_i.

Pentru fiecare copiere în generaţia următoare, se copiază şirul la care s-a oprit discul. Se fac şase copieri, pentru ca a doua generaţie să fie la fel de mare:

011010
001011
001011
010011
100111
010110

Primul şir se reproduce, în medie, de $6 \cdot 0,412 = 0,71$ ori deci, prin rotunjire, o dată.

Încrucişare

Se simulează schimbul de material genetic care are loc în timpul reproducerii biologice. În mod aleator, se obţin perechi de şiruri. Pentru simplitate, vom încrucişa perechile adiacente. Pentru fiecare pereche se obţine un număr întreg aleator între 1 şi 6. Acesta arată câţi biţi din dreapta se schimbă între cele două şiruri. De exemplu, încrucişăm şirurile A_1 şi A_2, având întregul aleator 2:

$$\left\{ \begin{array}{c} 011010 \\ 001011 \end{array} \right\} \rightarrow \left\{ \begin{array}{c} 011011 \\ 001010 \end{array} \right\}.$$

Procesul continuă pentru şirurile A_3 şi A_4, având întregul aleator 4 şi pentru şirurile A_5 şi A_6 având întregul aleator 3. Obţinem şirurile:

011011
001010
000011
011011
100110
010111

Mutaţie

Biologic, ea perturbă caracteristicile populaţiei, în mod aleator, prevenind degenerările. Cele mai multe mutaţii mai mult distrug decât să fie benefice. De aceea, mutaţia trebuie să fie rară, pentru a nu distruge specia. Pe de altă parte, ea e necesară. În simularea noastră, vom presupune o rată a mutaţiei de 1/1000 biţi, mutaţia însemnând aici o inversare de bit. Deoarece avem 36 de biţi în populaţie, probabilitatea unei mutaţii este 0,036. Presupunem că nu are loc nici o mutaţie la acest pas. Obţinem astfel a doua generaţie:

i	şirul A_i	val. zecimală	f_i	p_i
1	011011	27	0,037	0,064
2	001010	10	0,10	0,174
3	000011	3	0,333	0,578
4	011011	27	0,037	0,064
5	100110	38	0,0263	0,046
6	010111	23	0,043	0,075
Total			0,576	1

Observăm că totalul funcţiei obiectiv a crescut vizibil.

Întregul proces se reia, obţinând următoarele generaţii.

În general, populaţia poate să crească sau să scadă de la o generaţie la alta.

Acest proces continuă până când funcţia obiectiv atinge o valoare acceptabilă pentru o anumită generaţie. Convergenţa este, în general, rapidă.

Se poate întâmpla să ajungem la o degenerare, de exemplu dacă cele şase şiruri sunt 000011. De aici putem ieşi doar aşteptând o mutaţie favorabilă. Aceasta poate avea loc doar după multe generaţii. Astfel se poate întâmpla şi în natură, unde o specie poate suferi o adaptare neoptimă pentru milenii, până când apare un organism mai bine adaptat care ajută la perfecţionarea speciei.

10.3 Fundamente matematice

O *schemă* este o submulţime de şiruri cu similarităţi în anumite poziţii. Vom considera şiruri peste alfabetul $V = \{0, 1\}$. Un şir va fi notat astfel: $A = a_1 a_2 a_3 \ldots a_l$, unde a_i poate fi 0 sau 1, iar l este lungimea şirului. O schemă H peste alfabetul

$V+ = \{0, 1, *\}$ este, de exemplu, $H = *11*0**$. Schema H este, de pildă, reprezentată de şirul 0110000.

Dacă V are k şiruri, atunci se pot forma $(k+1)^l$ scheme. Fiecare şir este conţinut în 2^l scheme deoarece, fiecare caracter din şir poate avea fie valoarea pe care o are, fie $*$.

Deci, într-o populaţie de n şiruri, există cel mult $n \cdot 2^l$ scheme şi cel puţin 2^l scheme reprezentate efectiv în populaţie, acest număr depinzând de diversitatea populaţiei. *Ordinul* unei scheme H, $o(H)$, este numărul poziţiilor fixate din H. De exemplu, $o(011*1**) = 4$.

Lungimea de definiţie a unei scheme H, $\delta(h)$, este distanţa dintre prima şi ultima poziţie fixată. De exemplu, $\delta(011*1**) = 4$.

Vom considera efectele operaţiilor de reproducere, încrucişare şi mutaţie asupra schemelor reprezentate într-o populaţie de şiruri.

Reproducţie

Presupunem că la momentul t există m reprezentanţi (m şiruri) ai schemei H în generaţia $A(t)$. Vom scrie: $m = m(H, t)$. Conform algoritmului de reproducţie, un şir A_i este copiat în generaţia următoare cu probabilitatea $p_i = \frac{f_i}{\sum f_j}$. Presupunem că selectăm n şiruri distincte din $A(t)$ pentru următoarea generaţie, $A(t+1)$, cu "reintroducere". Avem:

$$m(H, t+1) = m(H, t) \cdot n \cdot \frac{f(H)}{\sum f_j}$$

unde $f(H)$ este valoarea medie a funcţiei obiectiv pentru şirurile care reprezintă schema H la momentul t.

Fie $\bar{f} = \frac{\sum f_j}{n}$ valoarea medie a funcţiei obiectiv pentru întreaga populaţie ($A(t)$ are n şiruri). Atunci:

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}}.$$

O schemă H creşte deci direct proporţional cu raportul:

$$\frac{\text{valoarea medie a funcţiei obiectiv pentru schema } H}{\text{valoarea medie a funcţiei obiectiv pentru întreaga populaţie}}.$$

Efectul reproducerii este clar: din punct de vedere al numărului de reprezentanţi, o schemă H cu $f(H) > \bar{f}$ creşte, iar o schemă H cu $f(H) < \bar{f}$ scade.

Să presupunem acum că pentru o schemă H avem $f(H) = \bar{f} + c\bar{f}$, unde c este o constantă. Atunci:

$$m(H, t+1) = m(H, t)\frac{\bar{f} + c\bar{f}}{\bar{f}} = (1+c)\, m(H, t).$$

Pornind din $t = 0$, obţinem:

$$m(H, t) = m(H, 0)(1+c)^t$$

adică, obţinem o progresie geometrică. *Creşterea / scăderea unei scheme prin reproducţie este exponenţială.*

Încrucişare

Fie $l = 7$ şi fie:

$$\begin{aligned} A &= \texttt{0111000} \\ H_1 &= \texttt{*1****0} \\ H_2 &= \texttt{***10**} \end{aligned}$$

Şirul A reprezintă schemele H_1 şi H_2. "Tăietura" după care se execută încrucişarea se alege aleator, echiprobabil între cele 6 posibilităţi. De exemplu:

$$\begin{aligned} A &= \texttt{011|1000} \\ H_1 &= \texttt{*1*|***0} \\ H_2 &= \texttt{***|10**} \end{aligned}$$

Încrucişarea are loc prin schimbarea primelor 3 caractere din A. Observăm ca H_1 este distrusă, deoarece $*1*$ şi $***0$ vor fi plasate în şiruri diferite. H_2 supravieţuieşte însă. Probabilitatea ca H_1 să fie distrusă este mai mare, deoarece $\delta(H_1) > \delta(H_2)$. Probabilitatea ca H_1 să fie distrusă este

$$p_d = \frac{\delta(H_1)}{l-1} = \frac{5}{6}.$$

Probabilitatea ca H_1 să supravieţuiască este

$$p_s = 1 - p_d = \frac{1}{6}$$

. Similar, pentru H_2, $p_d = \frac{1}{6}$.

Nu am considerat deocamdată situaţia în care, prin operaţia de încrucişare, poziţiile fixate din schemă nu se modifică.

În general,

$$p_s = \frac{1 - \delta(H)}{l-1}.$$

Fie p_c probabilitatea ca să efectuăm operaţia de încrucişare. Avem:

$$p_s \geq 1 - p_c \, \frac{\delta(H)}{l-1}.$$

Se foloseşte semnul $\geq$ pentru că acum ţinem cont şi de situaţia în care, prin operaţia de încrucişare, poziţiile fixate nu se modifică.

Considerăm acum efectul combinat al reproducţiei şi încrucişării. Presupunând că aceste două operaţii sunt independente obţinem:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left(1 - p_c \, \frac{\delta(H)}{l-1}\right).$$

Observăm că *schemele cu lungime mică sunt favorizate.*

Mutaţie

Alterăm o poziţie fixată cu probabilitatea p_m, deci supravieţuieşte cu probabilitatea $1-p_m$. O schemă H supravieţuieşte dacă fiecare dintre cele $o(H)$ poziţii fixate supravieţuieşte. Rezultă că probabilitatea ca o schemă H să supravieţuiască este $(1-p_m)^{o(H)}$. Pentru valori mici ale lui p_m, această probabilitate se poate aproxima cu $1-o(H)p_m$, conform inegalităţii Bernoulli: Oricare ar fi $a \geq -1$ şi n natural, avem:

$$(1+a)^n \geq 1+na.$$

Ţinând cont de toate operaţiile genetice, obţinem (rotunjit):

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left(1-p_c\,\frac{\delta(H)}{l-1}-o(H)p_m\right).$$

Observăm că *schemele cu ordin mic sunt favorizate.*

În final, putem formula următoarea teoremă:

> **Teorema Fundamentală a Algoritmilor Genetici (Holland, 1975)**. Schemele cu valoare a funcţiei obiectiv peste medie, cu lungimea de definiţie şi ordinul mici, cresc exponenţial în generaţiile următoare.

Am văzut că într-o populaţie de n şiruri de lungime l sunt procesate între 2^l şi $n2^l$ scheme. Nu toate aceste scheme sunt procesate cu o probabililitate mare, datorită operaţiilor de încrucişare şi mutaţie (schemele lungi sunt distruse).

Se poate demonstra că numărul de scheme efectiv procesate este în ordinul lui n^3, ţinând cont şi de operaţiile de încrucişare şi mutaţie.

Cu alte cuvinte, cu toate că se procesează n şiruri la fiecare generaţie, un algoritm genetic procesează de fapt în ordinul lui n^3 scheme. Această proprietate defineşte *paralelismul implicit* al unui algoritm genetic. Manipulând n elemente, procesăm de fapt aproximativ n^3 similarităţi.

Acestui paralelism implicit i se datorează faptul că spaţiul căutărilor este suficient de mare pentru ca algoritmul să nu ducă, în general, la un optim local ci la unul global.

Fiind dată o populaţie de şiruri, cu valorile respective ale funcţiei obiectiv, ne punem problema de ce informaţie dispunem pentru a ghida căutarea unor şiruri mai bune. Răspunsul este: similarităţile dintre şiruri. Aceste similarităţi sunt descrise de schemele populaţiei.

10.4 Exerciţii

1. (C) Minimizaţi funcţia $f(x,y,z) = x^2+y^2+z^2$, $x,y,z \in [-512,512]$. Folosiţi şiruri binare de lungime 10.

Capitolul 11

Puterea şi complexitatea de calcul a reţelelor neurale

11.1 Maşina Turing

Maşina Turing este un model abstract de calculabilitate. O maşină Turing este formată dintr-o bandă divizată în celule, infinită la dreapta şi dintr-un cap de citire/scriere care la un moment dat este poziţionat exact asupra unei celule. Fiecare celulă poate conţine cel mult un simbol. În plus, există o mulţime finită de stări care controlează mutările maşinii. La o mutare, maşina Turing, în funcţie de simbolul din celula de bandă pe care este poziţionat capul de citire/scriere şi de starea în care se află maşina, efectuează următoarele:

1. Trece într-o nouă stare.
2. Scrie un simbol în celula pe care maşina se află poziţionată, înlocuind ceea ce era scris înainte (eventual poate tipări acelaşi simbol).
3. Deplasează capul de citire o celulă la dreapta sau stânga, sau lasă capul de citire/scriere poziţionat asupra aceleiaşi celule.

Iniţial, capul de citire/scriere este poziţionat asupra celei mai din stânga celule. Primele n celule din stânga, pentru un $n \geq 0$, conţin simboluri care reprezintă datele de intrare, iar restul de celule, în număr infinit, conţin un simbol special ce nu poate fi folosit pentru reprezentarea datelor de intrare (de obicei, blank).

Maşina porneşte dintr-o stare precis determinată, numită *stare iniţială*. Unele stări din mulţimea finită a stărilor se numesc *stări finale*. Stările finale sunt de două tipuri: de acceptare şi de respingere.

Mulţimea simbolurilor formează un *alfabet* (finit). Secvenţele finite de simboluri se numesc *cuvinte*. O maşină M *acceptă* un cuvânt w dacă, pornind din starea iniţială, cu w ca dată de intrare, ajunge, după un număr finit de paşi, într-o stare finală de acceptare.

Limbajul acceptat de o maşină M este mulţimea (finită sau infinită):

$$L(M) = \{w | M \text{ acceptă } w\}.$$

Mulţimea limbajelor acceptate de maşini Turing formează *clasa limbajelor recursiv enumerabile,* $\mathcal{L}_0$.

În cazul în care numărul de paşi efectuaţi este finit, modul de operare al unei maşini Turing este descris de către un algoritm.

În afară de imaginea maşinii Turing ca şi o procedură (algoritm - dacă este finit) de recunoaştere a unei mulţimi, putem considera că un astfel de dispozitiv calculează o funcţie adăugând o bandă de scriere. O astfel de maşină calculează o funcţie naturală f dacă, pornind cu data de intrare x, se opreşte cu $f(x)$ pe banda de scriere. Funcţia f se numeşte, în acest caz, *Turing-calculabilă.*

Calculabilitatea intuitivă este un concept prematematic care nu permite nici o restricţie inevitabil impusă de o definiţie riguroasă.

Orice funcţie Turing-calculabilă este şi intuitiv calculabilă. Reciproca acestei afirmaţii este mai dificil de formulat, deoarece identifică două obiecte din care unul este matematic iar unul prematematic.

Teza lui Church-Turing (1936). Orice funcţie intuitiv calculabilă este Turing-calculabilă.

Teza lui Church-Turing nu poate fi demonstrată, ci doar justificată prematematic. Cu alte cuvinte, este greu de acceptat existenţa unor funcţii naturale intuitiv calculabile care să nu fie Turing-calculabile.

În conformitate cu teza lui Church-Turing, clasa mulţimilor $\mathcal{L}_0$ coincide cu clasa mulţimilor algoritmic calculabile.

Teoremă (Turing, 1936). Problema opririi maşinii Turing este nerezolvabilă.

Cu alte cuvinte, nu există un algoritm care să determine dacă o maşină Turing arbitrară, acţionată dintr-o configuraţie arbitrară (q, w, i) se va opri sau nu (q este starea, w este data de intrare, iar i este poziţia capului).

Puterea de calcula a maşinilor Turing este foarte mare: orice algoritm , chiar şi NP, poate fi implementat pe o maşină Turing corespunzătoare. O maşină Turing poate rezolva orice problemă rezolvabilă pe un calculator de uz general, având avantajul că poate fi descrisă matematic foarte concis.

Maşinile Turing pot fi *deterministe* sau *nedeterministe.* Mulţimea limbajelor acceptate de maşinile nedeterministe este, însă, egală cu mulţimea limbajelor acceptate de maşinile deterministe.

Un algoritm NP poate fi rezolvat într-un timp polinomial pe o maşină Turing nedeterministă. Dacă se doreşte rezolvarea pe o maşină deterministă, timpul nu mai rămâne polinomial.

Un algoritm P poate fi rezolvat într-un timp polinomial pe o maşină Turing deterministă.

11.2 Puterea de calcul a reţelelor neurale

Un *model masiv paralel* (MMP) este o mulţime de celule interconectate printr-o reţea de comunicaţie. Fiecărei celule îi corespunde o vecinătate din care provin conexiunile care intră în celulă. Numărul de celule poate fi infinit, numărul de conexiuni este, la nivel local, finit, iar conexiunile sunt orientate.

În particular, modele masiv paralele sunt: reţelele neurale, automatele celulare şi reţelele de automate.

În cadrul maşinilor Turing, o problemă este *rezolvabilă algoritmic* dacă există un algoritm având la intrare un cuvând finit w şi la ieşire un cuvânt care reprezintă informaţia determinată de w. Spre deosebire de aceasta, un MMP rezolvă *probleme de configuraţie*, având la intrare o configuraţie x a modelului şi la ieşire o configuraţie a modelului care reprezintă informaţia determinată de x.

Generalizăm acum noţiunea de rezolvabilitate pentru MMP.

Definiţie. O problemă de configuraţie este *p-rezolvabilă* printr-un model masiv paralel M dacă, indiferent de configuraţia de intrare x, există un moment t astfel încât avem:

$$\begin{array}{ll} \textit{stabilitate:} & T(t',x) = T(t,x), \text{ pentru } t' > t \\ \textit{corectitudine:} & T(t',x) \text{ este exact ieşirea dorită.} \end{array}$$

Prin T am notat funcţia care defineşte dinamica globală a modelului.

Câteva rezultate fundamentale

McCulloch şi Pitts afirmau în 1943 că reţelele neurale sunt calculatoare universale.

Franklin şi Garzon au arătat că orice problemă rezolvabilă algoritmic (printr-o maşină Turing) poate fi efectiv formulată sub forma unei probleme de configuraţie p-rezolvabilă pe o reţea neurala Aşadar, *reţelele neurale sunt ce puţin la fel de puternice ca şi maşinile Turing.*

Configuraţiile unui MMP pot fi însă infinite, deci p-rezolvabilitatea este o noţiune mai complexă decât calculabilitatea. Franklin şi Garzon au demonstrat că *problema opririi este p-rezolvabilă pe o reţea neurală infinită.* Această problemă nu este însă rezolvabilă pe o maşină Turing, după cum am văzut. Conform acestui rezultat, dar şi al altor rezultate de acest fel, s-au demonstrat de către Garzon şi Franklin[1] următoarele relaţii:

$$\text{maşini Turing} \subset \text{automate celulare} \subseteq \text{reţele neurale} \subseteq \text{reţele de automate}$$

[1]Garzon, M., S. Franklin "NeuralComputability II", Report of Memphis State University, MSU-TR-89-12, 1989.

11.3 Reprezentarea funcţiilor booleene prin reţele feedforward

Orice funcţie finită poate fi implementată exact printr-o reţea neurală feedforward, cu un strat ascuns, de perceptroni de tip continuu (Ito [2])

Prin definiţie, o funcţie finită este de forma $f : \mathcal{Q} \to \mathcal{R}$, unde $\mathcal{Q} \subset \mathcal{R}^d$, $\mathcal{Q}$ finită.

Orice funcţie continuă poate fi aproximată oricât de bine de o reţea neurală feedforward cu un singur strat ascuns, de perceptroni de tip continuu (Funahashi[3], Hecht-Nielsen [4]).

Să considerăm cazul funcţiilor booleene. Domeniul de definiţie al unei funcţii booleene cu N argumente conţine 2^N elemente. Pentru fiecare din aceste 2^N elemente există 2 valori posibile ale funcţiei. Deci, există 2^{2^N} funcţii booleene diferite cu N argumente. Vom reprezenta o funcţie booleeană arbitrară printr-o reţea feedforward cu un strat ascuns. Aceasta nu surprinde, o funcţie booleeană fiind un caz particular de funcţie finită. Primul strat va fi format din N perceptroni discreţi $\sigma_1, \ldots, \sigma_N$, reprezentând argumentele funcţiei. Ultimul strat constă dintr-un neuron discret S, reprezentând valoarea funcţiei $F(\sigma_1, \ldots, \sigma_N)$. Valorile logice sunt reprezentate de ± 1. În stratul ascuns sunt 2^N neuroni discreţi $q_0, \ldots, q_{2^N-1}$. Ponderea w_{jk} dintre σ_k şi q_j este $\pm w$, calculată astfel: fie $(\alpha_1, \ldots, \alpha_N)$ reprezentarea binară a lui j. Luăm $w_{jk} = +w$ dacă $\alpha_k = 1$ şi $w_{jk} = -w$ dacă $\alpha_k = 0$. Adică, $w_{jk} = (2\alpha_k - 1)w$, unde w este o constantă pozitivă. Alegem pragul pentru neuronii ascunşi: $(N-1)w$. Potenţialul pentru un neuron ascuns este:

$$\begin{aligned} h_j &= \sum_k w_{jk}\sigma_k - (N-1)w \\ &= w[(\sum_k (2\alpha_k - 1)\sigma_k) - (N-1)]. \end{aligned}$$

Avem $h_j > 0$ dacă şi numai dacă

$$(2\alpha_k - 1)\sigma_k = 1 \text{ pentru } k = 1, \ldots, N,$$

unde $2\alpha_k - 1 = \pm 1$ şi $\sigma_k = \pm 1$. Deci, condiţia este:

$$2\alpha_k - 1 = \sigma_k \text{ pentru } k = 1, \ldots, N.$$

Cu alte cuvinte, pentru o intrare, exact un neuron din stratul ascuns va deveni activ. Acest neuron este q_{j_0}, unde j_0 este reprezentat binar ca $\left(\frac{\sigma_1+1}{2}, \ldots, \frac{\sigma_N+1}{2}\right)$.

[2] Ito, Y. "Finite Mapping by Neural Networks and Truth Functions", Math. Scientist, 17, 1992, 69-77.

[3] Funahashi, K. "On the Approximate Realization of Continuous Mapping by Neural Networks", Neural Networks, 2, 1989, 183-192.

[4] Hecht-Nielsen, R. "Theory of the Backpropagation Neural Network", In: "Neural Networks for Perception", H. Wechsler (Ed.), Academic Press, 1992, 65-93.

Avem, deci, următoarele ieşiri din neuronii ascunşi:

$$s_j = \begin{cases} +1 & \text{pentru } j = j_0 \\ -1 & \text{pentru } j \neq j_0 \end{cases} .$$

Ponderea v_j dintre s_j şi neuronul S va fi:

$$v_j = \begin{cases} +1 & \text{dacă } F(\sigma_1, \ldots, \sigma_N) = true \\ -1 & \text{dacă } F(\sigma_1, \ldots, \sigma_N) = false \end{cases} ,$$

unde $(\sigma_1, \ldots, \sigma_N)$ corespunde lui s_j. Luăm pragul:

$$-\sum_j v_j$$

. Potenţialul pentru neuronul S este:

$$k = \sum_j v_j s_j + \sum_j v_j = \sum_j v_j(s_j + 1) = 2v_{j_0}.$$

Ieşirea din reţea va fi $\text{sgn}(h) = F(\sigma_1, \ldots, \sigma_N)$ (fig. 11.1).

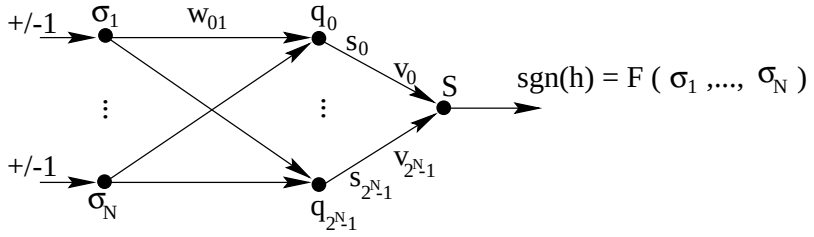


Figura 11.1: Reţea neurală care implementează funcţia $F(\sigma_1, \ldots, \sigma_N)$.

Se observă că numărul de neuroni ascunşi este 2^N, deci foarte mare. Pe de altă parte, problema reprezentării unei funcţii booleene (SATI) este NP-completă (teorema lui Cook). Deci, este explicabil. De fapt, am folosit forma normală disjunctivă, cu o poartă OR şi 2^N porţi AND.

În cazuri particulare, reprezentarea se poare face însă mult mai eficient (cu mai puţini neuroni). De exemplu, XOR su 2 argumente se poate reprezenta cu 2, nu 4 neuroni ascunşi.

Apare problema *minimizării* reţelei neurale care reprezintă o funcţie booleană dată. Pentru reţelele cu un strat ascuns, corespunzând funcţiilor booleene în forma normală conjunctivă, problema minimizării este cunoscută: se minimizează funcţia booleană printr-unul din algoritmii uzuali (Karnaugh sau Quine - Mc-Cluskey). Minimizarea se referă aici la numărul de neuroni, adică numărul de porţi. O problemă cu totul diferită este dacă o funcţie booleană cu N argumente poate fi "învăţată" de către o reţea neurală, într-un timp polinomial. Valiant[5] a

[5] Valiant, L.G. "A Theory of the Learnable", Commun. ACM, 27, 1134, 1984.

arătat că funcţiile logice sub formă normală conjunctivă pot fi instruite folosind doar un număr polinomial de exemple pozitive. În cazul funcţiilor în formă normală disjunctivă mai este nevoie, în plus, de încă un nivel de învăţare.

O problemă interesantă este dacă funcţia poate fi învăţată folosind doar puţine exemple şi lăsând-o apoi să "generalizeze".

11.4 Complexitatea instruirii reţelelor neurale feedforward

Fie cazul reţelelor feedforward multistrat cu perceptroni de tip continuu sau discret (nu are importanţă) şi fie cazul învăţării supervizate.

> **Teoremă** (Judd[6]). Problema generală a instruirii unei astfel de reţele este NP-completă.

Cu alte cuvinte, presupunând P$\neq$NP, nu există un algoritm cu timp polinomial care să instruiască o reţea în condiţiile în care:

1. Arhitectura reţelei (neuronii şi conexiunile, fără ponderi) este arbitrară.
2. Numărul de pattern-uri care trebuie memorate în reţea este arbitrar.

Mărimea unui caz este aici proporţională cu mărimea reţelei (numărul de perceptroni, conexiuni) plus numărul pattern-urilor de memorat.

Dacă, de exemplu, pentru a accelera instruirea, mărim numărul de neuroni în reţea, timpul de execuţie creşte foarte mult, deoarece timpul poate creşte exponenţial faţă de numărul de neuroni.

Teorema rămâne valabilă indiferent de funcţia de activare folosită. De asemenea, teorema rămâne valabilă chiar dacă arhitectura reţelelor considerate verifică restricţia:

$$\text{nr. de straturi ascunse } \leq 1.$$

Observaţii:

1. Această teoremă se referă la instruire în general. Există clase mari de reţele şi secvenţe de instruire care duc la o instruire în timp polinomial. Problema generală a instruirii unei reţele cu o funcţie booleană oarecare este însă NP-completă.
2. Se observă empiric faptul că algoritmul de instruire prin propagarea în urmă a erorii este foarte lent dacă numărul de straturi este mare. S-ar putea deduce că aceasta ar fi o proprietate generală: reţelele cu straturi puţine se instruiesc mai eficient. Teorema lui Judd ne arată că nu este neapărat aşa. Importantă este mărimea reţelei, complexitatea ei.

[6]Judd, J.S. "Neural Network Design and the Complexity of Learning", The MIT Press, Cambridge, 1990.

3. Instruirea este de fapt memorare. Memorarea unor perechi asociate de şiruri se face în timp liniar, pe o maşină von Neumann, adică mult mai rapid. Este, însă, adevărat că astfel se pierd avantajele procesării neurale, de exemplu invarianţa la perturbaţii.

4. Rămâne ca subiect de cercetare, din punct de vedere al NP-completitudinii, problema instruirii reţelelor recurente.

Capitolul 12

Consideraţii epistemologice asupra calculului neural

12.1 Scopul unei reţele neurale

Formele superioare de viaţă, inclusiv cele bazate pe sisteme nervoase centrale, tind să faciliteze supravieţuirea speciei respective în condiţii de mediu care devin din ce în ce mai dificile[1].

Natura nu îşi foloseşte niciodată în mod inutil resursele, ci acţionează conform unui principiu de optimalitate. De aceea, putem afirma că mecanismele creierului tind să asigure în mod optim supravieţuirea. De exemplu, somnul, în interpretarea lui Freud, tinde să integreze zgomotele, păstrând homeostazia sistemului.

Din această perspectivă, analogiile dintre calculatoarele digitale şi reţelele neurale biologice sunt nerezonabile. Problemele aritmetice sunt rareori rezolvate cu scopuri biologice; poate că păsările învaţă să-şi numere ouăle, însă, chiar şi această abilitate poate fi explicată, până la un anumit nivel, prin funcţii elementare de recunoaştere a formelor. Necesitatea de a număra sclavi şi soldaţi folosind semne este de origine mai târzie în dezvoltarea omului şi este realizată pe baza unor mijloace nebiologice. Orice încercare de a construi modele neurale pentru sumatoare, multiplicatoare, convertoare A/D şi alte circuite de calcul este de aceea, din acest punct de vedere, iraţională.

Cea mai importantă funcţie a sistemului nervos este monitorizarea şi controlul condiţiilor în care organismul trăieşte în mediul respectiv. Această funcţie se realizează prin modificarea stărilor organismului şi/sau a mediului. Cele mai elementare stări controlabile ale organismului sunt globale, nestructurate (de exemplu, temperatura). Cele mai evoluate astfel de stări sunt legate de comportament social, care implică planificarea anticipativă şi interacţiunile sociale. Chiar şi acestea sunt, însă, motivate biologic.

Diferenţa dintre calculatoare şi creier rezultă din scopurile lor diferite. Calcu-

[1]Kohonen, T. "Self-Organization and Associative Memory". Springer, Berlin, 3rd ed., 1989, chap. 9.

latoarele au fost dezvoltate iniţial pentru a rezolva probleme matematice, mai ales aritmetice. Ulterior, s-a descoperit că instrucţiunile folosite pentru calculul simbolic pot fi utilizate şi pentru procesarea limbajului natural. Această descoperire a fost un efect secundar în istoria dezvoltării calculatoarelor. Încă nu s-a observat că o anumită generaţie de calculatoare să-şi dorească să supravieţuiască!

Pentru a modela cât mai bine sistemele neurale biologice, reţelele neurale artificiale trebuie să interacţioneze cu lumea reală, cu obiectele naturale. Statistica semnalelor naturale, de exemplu vorbirea, sunt complet diferite de statistica presupusă în teoria matematică a probabilităţilor. Natura gaussiană a zgomotului şi distribuţia normală a datelor sunt ipoteze foarte proaste în acest context.

12.2 Funcţiile neurale biologice sunt localizate sau distribuite?

Este unul dintre subiectele de dezbatere continuă. Circuitele neurale trebuie să proceseze funcţiile local, în timp ce distribuirea globală a activităţilor este o funcţie colectivă a activităţilor tuturor părţilor.

De fapt, dihotomia distribuire - localizare poate fi urmărită şi la nivelul neuronului:

- putem localiza cu acurateţe de 10 microni corpul şi axonul neuronului
- ramurile dendritelor aceleiaşi celule sunt însă distribuite pe o arie cu un diametru de câţiva milimetri.

Cu alte cuvinte, răspunsul poate fi localizat, cu toate că funcţiile (adaptive) pot fi distribuite spaţial pe o arie mare.

12.3 Este neliniaritatea esenţială în calculul neural?

Semnalele procesate în reţelele neurale sunt, de obicei, continue. În electronică, cu toate că, de fapt, caracteristicile componentelor sunt neliniare, prin folosirea feedback-urilor negative şi a altor metode de stabilizare, se organizează componentele şi circuitele în module integrate care se comportă liniar.

Neuronii biologici au caracteristici şi mai neregulate decât componentele electronice. Faptul că neuronii au răspunsuri neliniare nu înseamnă însă că sistemul integrat nu poate să se comporte liniar. Iată un exemplu de astfel de integrare: când creierul recunoaşte o situaţie importantă, sistemul reticular de activare adaugă în circuitul cortical o excitaţie care face ca neuronii să opereze în mijlocul plajei lor dinamice. Se adaugă deci un semnal pentru a modifica plaja în care operează neuronii. Acest lucru se face doar pentru un timp scurt. În plaja lor dinamică, neuronii pot apoi să funcţioneze liniar. În ansambul, sistemul funcţionează, în acest caz, *temporar liniar*. Dacă sistemul reticular de activare

nu recunoaşte o situaţie ca fiind importantă, neuronii rămân inactivi, sub pragul de saturaţie.

Există şi cazuri de neliniaritate. De exemplu, în cazul percepţiei vizuale la insecte. Ochiul insectei este format din multe lentile, fiecare captând un sector al câmpului vizual. Semnalele rezultate din fotoreceptoarele acestor lentile sunt prelucrate în câţiva centri nervoşi, numiţi *ganglia*, situaţi lângă organele senzoriale. Aceşti centri combină semnalele receptate într-un mod *neliniar*, formând produse de perechi de semnale - unul obţinut din figură şi unul din fundal. Corelarea perechilor de semnale este diferită în cazurile când figura este, respectiv nu este, în mişcare.

12.4 Deosebirile esenţiale dintre calculatoarele neurale şi cele digitale

1. *Gradul de paralelism al reţelelor neurale este mai mare decât cel al oricărui calculator digital masiv paralel.*

   Calculatoarele masiv paralele sunt reţele de procesoare care lucrează asincron. Aceste calculatoare sunt, totuşi, digitale şi au circuite digitale în nodurile lor. Deosebirea lor esenţială faţă de calculatoarele von Neumann constă în faptul că este imposibil de prezis ordinea operaţiilor realizate în timpul calculului.

   Reţelele neurale lucrează tot asincron. Semnalele neurale nu au însă o formă exactă, deci nu este posibil să combinăm nici o informaţie de control (coduri de control, biţi de paritate, adrese) cu datele. De exemplu, multiplexarea nu mai este, din această cauză, posibilă.

   Calculatoarele neurale sunt calculatoare analogice. Ele nu pot executa programe memorate. Operaţiile implementate pe ele nu pot fi definite algoritmic.

2. *De ce semnalele neurale nu pot fi aproximate de variabile booleene.*

   Impulsurile neurale au un aspect binar: amplitudinea nu contează. Dar durata unui impuls nu este variabilă: nu este posibil să definim semnale logice statice similare cu cele folosite în calculatoarele digitale. Mai mult, impulsurile neurale nu sunt nici sincronizate, deci semnalele neurale diferă de cele digitale.

3. *Circuitele neurale nu implementează automate finite.*

   Filosofia calculului digital, dar şi a inteligenţei artificiale, se bazează pe teoria automatelor finite. Prin feedback (fig. 12.1), un automat finit poate implementa orice secvenţă de stări, fiind capabil de a defini recursiv funcţiile calculabile.

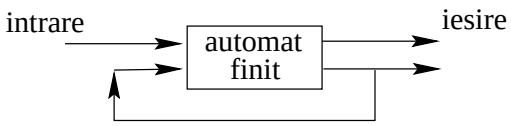


Figura 12.1: Automat finit.

Toate calculatoarele digitale aplică principiul recursivităţii la diferite nivele, de la hardware la procedurile definite în limbaje de programare. Aplicarea recursivităţii este posibilă din două motive tehnologice:

- valorile semnalelor sunt stabile
- toate combinaţiile de semnale şi stări sunt accesibile

O reţea neurală biologică nu poate implementa un automat finit. Funcţiile feedback din reţelele neurale servesc altor scopuri. Procesul de calcul neural artificial este mai apropiat unei aproximări stohastice.

12.5 Cum pot fi programate calculatoarele neurale

Este greu de imaginat cum ar putea fi programaţi paşii individuali de procesare în creier şi cum să fie programat un calculator neural ca un calculator digital.

Structurile anatomice în sine constituie "un program", incluzând şi codul genetic. În acest sens, creierul este "programat". Similar, arhitectura reţelei neurale artificiale reprezintă "programul" ei.

În natura biologică, structurile par atât de semnificative, ca şi cum ar fi create conform unui plan *a priori*. Explicaţia ştiinţifică este că mutaţiile, pe parcursul unei lungi evoluţii, duc prin selecţie la soluţia optimă. Soluţiile optime la care se ajunge sunt însă foarte greu de înţeles, datorită faptului că sunt definite implicit prin codurile genetice. Este imposibil să adoptăm această tehnică în calculul reţelelor neurale artificiale.

Problema principală este cum să obţinem răspunsuri corecte din partea mediului fără intervenţia şi interpretarea programatorului şi să le prezentăm sistemului.

Din toate aceste motive, putem afirma că o reţea neurală veritabilă nu poate fi programată. Cele mai utile aplicaţii ale reţelelor neurale sunt cele pentru care structurile sau mecanismele sunt deja familiare.

12.6 Poate creierul să se autoperceapă?

Numim *complexitate* a unui sistem numărul de bucle (feedback-uri) care pot apărea în sistemul respectiv. Astfel, complexitatea unei singure bucle este 1.

Creierul care poate să perceapă comportamentul unui alt creier mai mic trebuie să aibă o complexitate de cel puţin 10^9 ori mai mare decât complexitatea creierului investigat (Stakermann[2]).

În mod ironic, cunoştinţele deja achiziţionate sunt pierdute în mod continuu prin moartea indivizilor şi trebuie restabilite de către cei tineri.

În concluzie, o identitate nu se poate autopercepe. Creierul biologic nu se poate autoînţelege.

[2]Starkermann, R. "The Functional Intricacy of Neural Networks - A Mathematical Study", Proceedings, Insbruck, 1993.

Anexa A

Complemente matematice

A.1 Vectori şi matrici

Vector coloană: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$; Vector linie: $\mathbf{x} = [x_1 \dots x_n]^t$

Fie vectorii n-dimensionali $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$. Vectorul $\mathbf{x}$ este o *combinaţie liniară* a vectorilor $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ dacă există scalarii c_1, ..., c_m astfel încât:

$$\mathbf{x} = \sum_{i=1}^{m} c_i \mathbf{x}_i.$$

Mulţimea de vectori $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ este *liniar dependentă* dacă există c_1, ..., c_m, nu toţi nuli, astfel încât:

$$\sum_{i=1}^{m} c_i \mathbf{x}_i = 0.$$

În caz contrar, vectorii $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ sunt liniar independenţi.

Norma euclidiană a unui vector: $\|\mathbf{x}\| = (\mathbf{x}^t\mathbf{x})^{1/2} = \sqrt{\sum_{i=1}^{n} x_i^2}$.

Produsul scalar a doi vectori $\mathbf{x}$ şi $\mathbf{y}$: $\mathbf{x}^t\mathbf{y} = \sum_{i=1}^{n} x_i y_i = \|\mathbf{x}\| \cdot \|\mathbf{y}\| \cdot \cos\psi$ unde ψ este unghiul dintre cei doi vectori.

Proprietate: $\mathbf{x}^t\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\| cos\psi$, unde ψ este unghiul dintre $\mathbf{x}$ şi $\mathbf{y}$.

Se observă că $\mathbf{x}^t\mathbf{y} = \mathbf{y}^t\mathbf{x}$, deci produsul scalar este comutativ.

Vectorii $\mathbf{x}$ şi $\mathbf{y}$ sunt *ortogonali* dacă şi numai dacă produsul lor scalar este 0.

Dacă vectorii nenuli $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$ sunt doi câte doi ortogonali, atunci sunt liniar independenţi.

Exemplu: Vectorii $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $\mathbf{x}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ sunt doi câte doi ortogonali şi sunt independenţi.

Exemplu: Găsiţi produsul scalar dintre vectorii $\mathbf{x}$ şi $\mathbf{y}$, lungimea lor şi unghiul dintre ei, unde:

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ -1 \\ 4 \end{bmatrix}$$

Soluţie: $\mathbf{x}^t\mathbf{y} = 9$, $\|\mathbf{x}\| = 3$, $\|\mathbf{y}\| = \sqrt{18}$, $\cos\psi = \frac{\mathbf{x}^t\mathbf{y}}{\|\mathbf{x}\|\cdot\|\mathbf{y}\|} = \frac{1}{\sqrt{2}}$, deci $\psi = 45^o$.

A.2 Forme pătratice

O *formă pătratică (FP)* este o funcţie $E(\mathbf{x}) = \mathbf{x}^t A\mathbf{x}$, unde $\mathbf{x}$ este un vector n-dimensional iar A o matrice simetrică de $n \times n$ elemente.

Dacă $\mathbf{A} = (a_{ij})_{i,j=1,\ldots,n}$, atunci:

$$\begin{aligned} E(x_1, x_2, \ldots, x_n) &= a_{11}x_1^2 + 2a_{12}x_1x_2 + 2a_{13}x_1x_3 + \ldots + 2a_{1n}x_1x_n \\ &+ a_{22}x_2^2 + 2a_{23}x_2x_3 + \ldots + 2a_{2n}x_2x_n + \ldots + a_{nn}x_n^2 = \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j \end{aligned}$$

Proprietăţi ale formelor pătratice

1. $E(\mathbf{x})$ este o *FP pozitiv semidefinită*, iar A este o matrice *pozitiv semidefinită* dacă $[(\forall)\mathbf{x} \neq 0 \Rightarrow E(\mathbf{x}) \geq 0]$.

2. $E(\mathbf{x})$ este o *FP pozitiv definită*, iar A este o matrice *pozitiv definită* dacă $[(\forall)\mathbf{x} \neq 0 \Rightarrow E(\mathbf{x}) > 0]$.

3. $E(\mathbf{x})$ este o *FP negativ semidefinită*, iar A este o matrice *negativ semidefinită* dacă $[(\forall)\mathbf{x} \neq 0 \Rightarrow E(\mathbf{x}) \leq 0]$.

4. $E(\mathbf{x})$ este o *FP negativ definită*, iar A este o matrice *negativ definită* dacă $[(\forall)\mathbf{x} \neq 0 \Rightarrow E(\mathbf{x}) < 0]$.

$E(\mathbf{x})$ este pozitiv definită dacă şi numai dacă pentru $\mathbf{A} = (a_{ij})_{i,j=1,\ldots,n}$ avem:

$$\begin{aligned} \det \mathbf{A}_{11} &= \|a_{11}\| > 0 \\ \det \mathbf{A}_{22} &= \begin{Vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{Vmatrix} > 0 \\ &\vdots \\ \det \mathbf{A}_{nn} &= \det \mathbf{A} > 0. \end{aligned}$$

Condiţii suficiente pentru ca $E(\mathbf{x})$ să fie:

a) FP pozitiv semidefinită:

$$\det A_{11} \geq 0,\ \det A_{22} \geq 0\ \ldots\ \det A_{nn} \geq 0,$$

b) FP pozitiv definită:

$$\det A_{11} > 0,\ \det A_{22} > 0\ \ldots\ \det A_{nn} > 0,$$

c) FP negativ semidefinită:

$$\det A_{11} \leq 0,\ \det A_{22} \geq 0,\ \det A_{33} \leq 0\ \ldots,$$

d) FP negativ definită:

$$\det A_{11} < 0,\ \det A_{22} > 0, \det A_{33}, \ldots$$

e) În toate celelalte cazuri, FP este **nedefinită**.

Să notăm că o formă pătratică $E(\mathbf{x})$ este nedefinită dacă este pozitivă pentru anumite valori ale lui $\mathbf{x}$ şi negativă pentru alte valori, deci depinde de $\mathbf{x}$.

A.3 Elemente de geometrie analitică

Ecuaţia unei drepte în plan este:

$$a\,x + b\,y + c = 0,\ \text{cu}\ a^2 + b^2 > 0.$$

Vectorul

$$\mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix}$$

este un *vector normal* la dreapta dată. Acesta este îndreptat spre semiplanul pozitiv, adică pentru care $a\,x + b\,y + c > 0$.

Ecuaţia unei drepte a cărei vector normal este $\mathbf{n} = [n_1, n_2]^t$ şi trece prin punctul reprezentat de vectorul $\mathbf{x}_1$ este:

$$\mathbf{n}^t \cdot (\mathbf{x} - \mathbf{x_1}) = 0.$$

Având două puncte x_1 şi x_2 în plan şi vectorii asociaţi lor, $\mathbf{x}_1$ şi $\mathbf{x}_2$, ecuaţia mediatoarei segmentului dintre x_1 şi x_2 este:

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \cdot (\mathbf{x} - \mathbf{x}_0) = 0,$$

unde x_0 este mijlocul segmentului respectiv.

Din relaţia anterioară rezultă

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} - \frac{1}{2}(\mathbf{x}_1 - \mathbf{x}_2)^t (\mathbf{x}_1 + \mathbf{x}_2) = 0$$

şi obţinem

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}(\mathbf{x}_2^t \mathbf{x}_2 - \mathbf{x}_1^t \mathbf{x}_1) = 0.$$

Din

$$\mathbf{x}^t \mathbf{x} = \|\mathbf{x}\|^2$$

obţinem următoarea ecuaţie a mediatoarei:

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}(\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2) = 0.$$

Ecuaţia planului care trece prin trei puncte necoliniare este:

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0$$

Ecuaţia planului care trece printr-un punct $\mathbf{x}(x_1, y_1, z_1)$ şi este perpendicular pe vectorul $\mathbf{n} = [a, b, c]^t$ este:

$$a\,(x - x_1) + b\,(y - y_1) + c\,(z - z_1) = 0.$$

Ecuaţia planului care trece prin mijlocul segmentului $(x_1 y_1 z_1)$, $(x_2 y_2 z_2)$ şi este perpendicular pe vectorul $\mathbf{x}_1 - \mathbf{x}_2$, unde $\mathbf{x}_1 = [x_1 y_1 z_1]^t$, $\mathbf{x}_2 = [x_2 y_2 z_2]^t$ este:

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}(||\mathbf{x}_2||^2 - ||\mathbf{x}_1||^2) = 0$$

A.4 Operaţia XOR

XOR este o operaţie binară asupra argumentelor logice x_1, x_2. Avem:

$$(x_1 \oplus x_2) \oplus x_3 = x_1 \oplus (x_2 \oplus x_3) = x_1 \oplus x_2 \oplus x_3.$$

Putem extinde operaţia XOR pentru n argumente logice:

$$\mathrm{XOR}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

Această funcţie se numeşte şi *funcţia de paritate* deoarece are valoarea 1 dacă şi numai dacă numărul argumentelor care au valoarea 1 este impar.

A.5 Iacobianul şi hessianul

Un vector $\mathbf{x}(t)$ dependent de timp se defineşte astfel:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} \qquad \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}x_1}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}x_n}{\mathrm{d}t} \end{bmatrix} = \dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix}$$

Fie funcţia $E(\mathbf{x})$, unde $\mathbf{x}$ este un vector cu n componente. *Gradientul* se defineşte astfel:

$$\nabla_{\mathbf{x}} E(\mathbf{x}) = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix}$$

Proprietăţi:

a) $\nabla_{\mathbf{x}}(\mathbf{x}^t\,\mathbf{y}) = \mathbf{y}$

b) $\nabla_{\mathbf{x}}(\mathbf{x}^t\mathbf{A}\,\mathbf{y}) = \mathbf{A}\,\mathbf{y}$

c) $\nabla_{\mathbf{x}}(\mathbf{x}^t\mathbf{A}\,\mathbf{y}) = \mathbf{A}\,\mathbf{x} + \mathbf{A}^t\,\mathbf{x}$

d) Când lucrăm cu forme pătratice, $\mathbf{A}$ este simetrică şi avem $\mathbf{A} = \mathbf{A}^t$. Rezultă $\nabla_{\mathbf{x}}(\mathbf{x}^t\mathbf{A}\,\mathbf{x}) = 2\,\mathbf{A}\,\mathbf{x}$.

Gradientul într-un punct $\mathbf{x}_0$ al unei funcţii reprezintă direcţia creşterii maxime a lui $E(\mathbf{x})$. Gradientul reprezintă tangenta la curbă pe direcţia de creştere şi este nul în punctele de maxim şi minim.

Dacă avem o funcţie $E[\mathbf{x}(t)]$ (nu neapărat o formă pătratică), unde $\mathbf{x}$ este un vector de n componente dependente de timp, atunci

$$\frac{\mathrm{d}E[\mathbf{x}(t)]}{\mathrm{d}t} = \frac{\partial E}{\partial x_1}\cdot\frac{\partial x_1}{\partial t} + \cdots + \frac{\partial E}{\partial x_n}\cdot\frac{\partial x_n}{\partial t} = \sum_{i=1}^{n}\frac{\partial E}{\partial x_i}\dot{x}_i(t).$$

Observăm că

$$\frac{\mathrm{d}E[\mathbf{x}(t)]}{\mathrm{d}t} = [\nabla_x E(\mathbf{x})]^t\dot{\mathbf{x}}(t).$$

Exemplu: Fie:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} e^{2t} \\ t \end{bmatrix}.$$

Forma pătratică a lui $\mathbf{x}(t)$ cu matricea

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

este:

$$\begin{aligned} E[\mathbf{x}(t)] &= \begin{bmatrix} e^{2t} & t \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}\begin{bmatrix} e^{2t} \\ t \end{bmatrix} \\ \nabla_{\mathbf{x}} E(\mathbf{x}) &= \begin{bmatrix} \frac{d}{dx_1}(x_1^2 + 4x_1x_2 + 3x_2^2) \\ \frac{d}{dx_2}(x_1^2 + 4x_1x_2 + 3x_2^2) \end{bmatrix} = \begin{bmatrix} 2x_1 + 4x_2 \\ 4x_1 + 6x_2 \end{bmatrix} \\ \dot{\mathbf{x}}(t) &= \begin{bmatrix} 2e^{2t} \\ 1 \end{bmatrix}. \end{aligned}$$

Atunci

$$\frac{dE[\mathbf{x}(t)]}{dt} = \left[\begin{array}{cc} 2e^{2t}+4t & 4e^{2t}+6t \end{array}\right] \left[\begin{array}{c} 2e^{t} \\ 1 \end{array}\right] = 2(2e^{4t}+(4t+2)e^{2t}+3t).$$

Fie m funcţii:

$$\begin{aligned} E_1(\mathbf{x}) &= E_1(x_1, x_2, \cdots, x_n) \\ &\vdots \\ E_m(\mathbf{x}) &= E_m(x_1, x_2, \cdots, x_n) \end{aligned}$$

Matricea iacobiană se defineşte în felul următor:

$$\mathbf{I}(\mathbf{x}) = \frac{d\mathbf{E}}{d\mathbf{x}} = \left[\begin{array}{ccc} \frac{\partial E_1}{\partial x_1} & \cdots & \frac{\partial E_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial E_m}{\partial x_1} & \cdots & \frac{\partial E_m}{\partial x_n} \end{array}\right]$$

Exemplu: Pentru:

$$\begin{aligned} \mathbf{x} &= \left[\begin{array}{cc} 2 & 1 \end{array}\right]^t \\ E_1(\mathbf{x}) &= x_1^2 - x_1 x_2 \\ E_2(\mathbf{x}) &= x_1 x_2 + x_2^2 \end{aligned}$$

avem:

$$\mathbf{I}(\mathbf{x}) = \left[\begin{array}{cc} 2x_1 - x_2 & -x_1 \\ x_2 & x_1 + 2x_2 \end{array}\right] = \left[\begin{array}{cc} 3 & -2 \\ 1 & 4 \end{array}\right].$$

Matricea hessiană a funcţiei $E(\mathbf{x}) = E(x_1, \cdots, x_n)$ se defineşte astfel:

$$\mathbf{H}(\mathbf{x}) = \nabla^2_{\mathbf{x}} E(\mathbf{x}) = \nabla_{\mathbf{x}}[\nabla_{\mathbf{x}} E(\mathbf{x})] = \left[\begin{array}{cccc} \frac{\partial^2 E}{\partial x_1^2} & \frac{\partial^2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial x_1 \partial x_n} & \frac{\partial^2 E}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 E}{\partial^2 x_n} \end{array}\right].$$

Matricea hessiană este simetrică.

Exemplu: Pentru:

$$E(\mathbf{x}) = (x_2 - x_1)^2 + (1 - x_1)^2$$

avem:

$$\nabla_{\mathbf{x}} E(\mathbf{x}) = 2\left[\begin{array}{c} 2x_1 - x_2 - 1 \\ x_2 - x_1 \end{array}\right] \qquad \nabla^2_{\mathbf{x}} E(\mathbf{x}) = 2\left[\begin{array}{cc} 2 & -1 \\ -1 & 1 \end{array}\right].$$

Deoarece determinanţii principali sunt 4 şi 2, matricea hessiană este pozitiv definită.

A.6 Probleme de optimizare

Presupunem că avem o funcţie obiectiv $E(x)$, unde x este o variabilă scalară. Această funcţie îşi atinge minimul în punctul $x = x^*$, pentru care se îndeplinesc următoarele condiţii:

$$\begin{cases} \frac{\mathrm{d}E(x^*)}{\mathrm{d}x^*} &= 0 \\ \frac{\mathrm{d}^2E(x^*)}{\mathrm{d}^2x^*} &> 0 \end{cases}.$$

Dacă $\mathbf{x}$ este un vector, generalizarea se face astfel:

$$\begin{cases} \nabla_{\mathbf{x}} E(\mathbf{x}^*) = 0 \\ \nabla^2_{\mathbf{x}} E(\mathbf{x}^*) \text{ pozitiv definită} \end{cases}.$$

Pentru a înţelege aceste condiţii, să considerăm, de exemplu:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Dezvoltăm în serie Taylor în punctul $\mathbf{x} = \mathbf{x}^*$ şi reţinem primii termeni ai dezvoltării:

$$\begin{aligned} E(\mathbf{x}) \cong\ & E(\mathbf{x}^*) + (x_1 - x_1^*)\frac{\partial E(\mathbf{x}^*)}{\partial x_1} + (x_2 - x_2^*)\frac{\partial E(\mathbf{x}^*)}{\partial x_2} \\ & + \frac{1}{2}(x_1 - x_1^*)^2\frac{\partial^2 E(\mathbf{x}^*)}{\partial^2 x_1} + \frac{1}{2}(x_2 - x_2^*)^2\frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2^2} \\ & + (x_1 - x_1^*)(x_2 - x_2^*)\frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1 \partial x_2}. \end{aligned}$$

Pentru $\Delta x_1 = x_1 - x_1^*$, $\Delta x_2 = x_2 - x_2^*$, obţinem:

$$\begin{aligned} E(\mathbf{x}) \cong\ & E(\mathbf{x}^*) + \begin{bmatrix} \frac{\partial E(\mathbf{x}^*)}{\partial x_1} & \frac{E(\mathbf{x}^*)}{\partial x_2} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} \\ & + \frac{1}{2} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}^t \begin{bmatrix} \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2 \partial x_2} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2^2} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}. \end{aligned}$$

Luând

$$\Delta\mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix},$$

obţinem:

$$E(\mathbf{x}) \cong E(\mathbf{x}^*) + [\nabla_{\mathbf{x}} E(\mathbf{x}^*)]^t \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^t [\nabla^2_{\mathbf{x}} E(\mathbf{x}^*)]\Delta\mathbf{x}. \tag{A.1}$$

Dacă $E(\mathbf{x})$ are un minim în $\mathbf{x}^*$. orice schimbare infinitezimală $\Delta\mathbf{x}$ în $\mathbf{x}^*$ trebuie să rezulte în $E(\mathbf{x}^* + \Delta\mathbf{x}) > E(\mathbf{x}^*)$. Pentru aceasta trebuie ca:

1. Gradientul în $\mathbf{x}^*$ să scadă şi să facă termenul liniar din A.1 să fie zero.
2. Hessiana în $\mathbf{x}^*$ să fie pozitiv definită, ceea ce va face forma pătratică din A.1 pozitivă, independent de $\Delta\mathbf{x}$.

Soluţia analitică a unei probleme de optimizare speciale

Fie funcţia:

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^t\mathbf{A}\mathbf{x} + \mathbf{b}^t\mathbf{x},$$

unde $\mathbf{x}$ este un vector n-dimensional, $\mathbf{A}$ este o matrice $n \times n$ simetrică, pozitiv definită, $\mathbf{b}$ este un vector n-dimensional constant. $E(\mathbf{x})$ este una dintre puţinele funcţii pentru care există o soluţie analitică pentru minimul fără restrincţii.

Calculăm:

$$\begin{aligned} \nabla_{\mathbf{x}} E(\mathbf{x}) &= \mathbf{A}\mathbf{x} + \mathbf{b} \\ \nabla^2_{\mathbf{x}} E(\mathbf{x}) &= \mathbf{A}. \end{aligned}$$

Prin ipoteză, $\mathbf{A}$ este pozitiv definită. Impunem condiţia $\mathbf{A}\mathbf{x} + \mathbf{b} = 0$. Soluţia este $\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{b}$. Rezultă că $\mathbf{x}^*$ este un minim.

A.7 Metoda lui Euler (metoda tangentei)

Fie ecuaţia diferenţială de ordinul întâi:

$$y' = f(x, y), \quad \text{unde } y = y(x)$$

cu condiţia iniţială

$$y_0 = y(x_0).$$

Dacă f este complicată, nu putem integra direct.

Dorim o aproximare a lui $y^* = \Phi(x)$, care se demonstrează că este soluţia unică (fig. A.1). Scriem:

$$\begin{aligned} y_1 &= y_0 + \Delta y \\ \Phi'(x_0) &= \frac{\Delta y}{\Delta x} \Rightarrow \Delta y = \Phi'(x_0)\Delta x. \end{aligned}$$

$$\begin{aligned} y_1 &= y_0 + \Phi'(x_0)(x_1 - x_0) = y_0 + f(x_0, y_0)(x_1 - x_0) \\ \text{Apoi: } y_2 &= y_1 + {y_1}'(x_2 - x_1) = y_1 + f(x_1, y_1)(x_2 - x_1). \end{aligned}$$

În general,

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + h{y_n}', \quad \text{dacă } x_{n+1} = x_n + h, \quad n = 0, 1, 2, \ldots.$$

Această relaţie este *formula lui Euler*.

Exemplu:

$$\begin{cases} y' = 1 - x + 4y \\ y(0) = 1 \end{cases}.$$

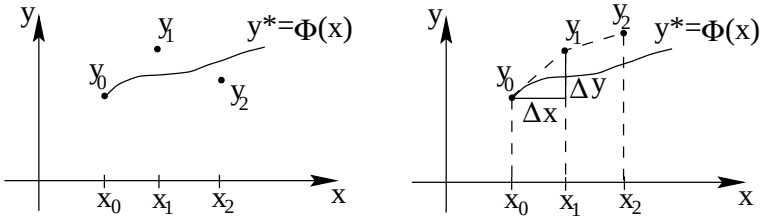


Figura A.1: Ne deplasăm pe tangenta la y_0, până la x_1.

Se ştie că soluţia exactă a acestui sistem are următoarea expresie analitică:

$$y^* = \Phi(x) = \frac{1}{4}x - \frac{3}{16} + \frac{19}{16}e^{4x}.$$

Aproximaţi pe $\Phi(0,2)$, presupunând că nu aveţi expresia analitică a lui Φ.

Avem:

$$\begin{aligned} h &= 0,1 \\ y_0{}' &= f(0,1) = 5 \\ y_1 &= y_0 + hf(0,1) = 1 + 0,1 \cdot 5 = 1,5 \\ y_2 &= y_1 + hf(x_1, y_1) = 1,5 + 0,1 \cdot (1 - 0,1 + 6) = 2,19. \end{aligned}$$

Eroarea este:

$$|2,19 - \Phi(0,2)| = 0,32.$$

Luând o valoare mai mică pentru h, obţinem o eroare mai mică.

Fie acum un sistem de două ecuaţii diferenţiale de ordinul I:

$$\begin{cases} x' = f(t,x,y) \\ y' = f(t,x,y) \\ x(t_0) = x_0, y(t_0) = y_0 \end{cases}.$$

Căutăm valorile aproximative x_1, x_2, ..., x_n, .., y_1, y_2, ..., y_n, ... ale soluţiei exacte $x^* = \Phi(t)$, $y^* = \Psi(t)$ în punctele $t_n = t_0 + nh$. Avem:

$$\begin{aligned} x_1 &= hf(t_0, x_0, y_0) \\ y_1 &= hg(t_0, x_0, y_0). \end{aligned}$$

În general:

$$\begin{aligned} x_{n+1} &= x_n + hf(t_n, x_n, y_n) = x_n + hx_n{}' \\ y_{n+1} &= y_n + hg(t_n, x_n, y_n) = y_n + hy_n{}'. \end{aligned}$$

Exemplu: Determinaţi valorile aproximative ale lui $x^* = \Phi(t)$, $y^* = \Psi(t)$ în punctele $t = 0{,}1$ şi $t = 0{,}2$ pentru sistemul:

$$\begin{cases} x' = x - 4y \\ y' = -x + y \\ x(0) = 1, y(0) = 0 \end{cases}.$$

Pentru $h = 0{,}1$ avem:

$$\begin{array}{ll} {x_0}' = 1 - 4 \cdot 0 = 1 & {y_0}' = -1 + 0 = -1 \\ x_1 = 1 + 0,1 \cdot 1 = 1,1 & y_1 = 0 + 0,1 \cdot (-1) = -0,1 \\ {x_1}' = 1,1 - 4 \cdot (-0,1) = 1,5 & {y_1}' = -1,1 + (-0,1) = -1,2 \\ x_2 = 1,1 + 0,1 \cdot 1,5 = 1,25 & y_2 = -0,1 + 0,1 \cdot (-1,2) = -0,22 \end{array}.$$

Fie sistemul:

$$\begin{cases} {x_1}' = F_1(t, x_1, \ldots, x_n) \\ {x_2}' = F_2(t, x_1, \ldots, x_n) \\ \vdots \\ {x_n}' = F_n(t, x_1, \ldots, x_n) \end{cases}$$

cu condiţiile iniţiale:

$$\begin{array}{l} x_1(t_0) = x_1^0 \\ \vdots \\ x_n(t_0) = x_n^0. \end{array}$$

Acest sistem are *o soluţie* în intervalul $\alpha < t < \beta$ dacă există funcţiile $x_1^* = \Phi_1(t)$, ..., $x_n^* = \Phi_n(y)$, diferenţiabile în $\alpha < t < \beta$ şi care satisfac condiţiile iniţiale.

> **Teoremă de existenţă şi unicitate**: Dacă F_1, ..., F_n şi $\frac{\partial F_1}{\partial x_1}$, ..., $\frac{\partial F_1}{\partial x_n}$, ..., $\frac{\partial F_n}{\partial x_1}$, ..., $\frac{\partial F_n}{\partial x_n}$ sunt continue în regiunea R care conţine punctul $(t_0, x_1^0, \ldots, x_n^0)$, atunci există un interval $|t - t_0| < R$ în care există o soluţie unică $x_1^* = \Phi_1(t)$, ..., $x_n^* = \Phi_n(t)$ a sistemului, care satisface condiţiile iniţiale.

Observăm că nu se spune nimic despre derivatele $\frac{\partial F_i}{\partial t}$.

A.8 Stabilitatea sistemelor dinamice neliniare

La sfârşitul secolului XIX, Liapunov a dezvoltat o metodă pentru analiza stabilităţii sistemelor dinamice. Această metodă se bazează pe conceptul de energie generalizată şi presupune evaluarea funcţiei lui Liapunov.

Fie un sistem autonom descris de sistemul de ecuaţii diferenţiale liniare sau neliniare:

$$\dot{x}_1 = f_1(\mathbf{x})$$

$$\dot{x}_2 = f_2(\mathbf{x})$$

$$\vdots$$

$$\dot{x}_n = f_n(\mathbf{x})$$

care este echivalent cu $\dot{\mathbf{x}} = f(\mathbf{x})$, unde $\mathbf{x}$ este vectorul stărilor sistemului. Presupunem că $f(0) = 0$, adică $\mathbf{x} = 0$ este un *punct de echilibru*. Vom formula o condiţie pentru ca $\mathbf{x} = 0$ să fie *asimptotic stabil*, adică vectorul stărilor să tindă către zero când timpul tinde către infinit. Fie $E(\mathbf{x})$ energia acumulată în sistem.

> **Teorema lui Liapunov**. Dacă E este o funcţie pozitiv definită pentru care:
>
> 1. E este continuă faţă de toate componentele $x_i, i = 1, 2, \ldots, n$
> 2. $\frac{\mathrm{d}E[\mathbf{x}(\mathbf{t})]}{\mathrm{d}t} < 0$, adică funcţia energiei descreşte în timp,
>
> atunci originea este asimptotic stabilă.

Funcţia E care satisface aceste condiţii se numeşte *funcţia lui Liapunov*. În general, ea nu este unică.

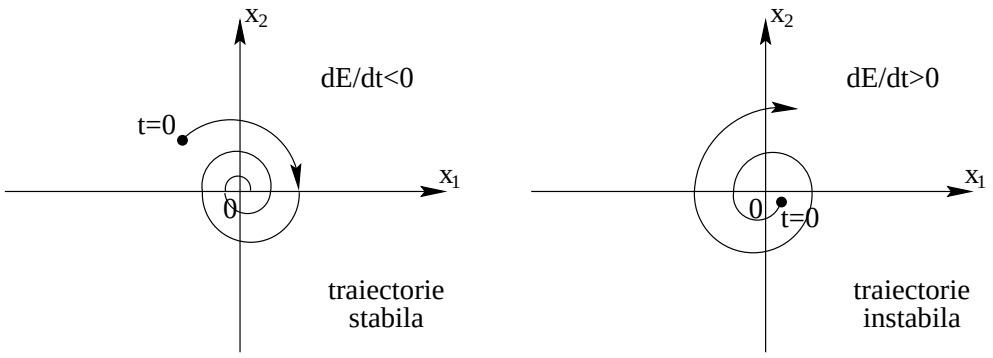


Figura A.2: Funcţia lui Liapunov

Dacă cel puţin o astfel de funcţie satisface condiţiile teoremei, atunci sistemul este asimptotic stabil. Uneori nu este foarte clar despre ce *energie* este vorba. Consideraţiile sunt mai curând matematice decât fizice. În cazul reţelelor neurale vorbim de *funcţia energiei computaţionale*, ea neavând legătură cu energia din fizică.

A.9 Variabile aleatoare

Fie o variabilă aleatoare discretă

$$\xi = \begin{pmatrix} a_1 & \ldots & a_n \\ p_1 & \ldots & p_n \end{pmatrix}, \sum p_i = 1.$$

Definim:

Valoarea medie: $m = \sum a_i p_i = E(\xi)$

Varianţa (dispersia): $\sigma^2 = E(\xi - E(\xi))^2 = E(\xi^2) - (E(\xi))^2 = \sum a_i^2 p_i^2 - (\sum a_i p_i)^2$

Deviaţia standard: σ.

Funcţia de repartiţie a unei variabile aleatoare normale ξ cu media m şi dispersia σ^2 este:

$$\begin{aligned} F(x) &= P(\xi < x) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(u-m)^2}{2\sigma^2}} du, \quad \text{pentru } -\infty < x < \infty \end{aligned}$$

şi este notată cu $N(m, \sigma^2)$. Distribuţia $N(0,1)$ este *standard*. Funcţia de densitate a lui $N(m, \sigma^2)$ este:

$$f(x) = F'(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}}, \quad \text{pentru } -\infty < x < \infty.$$

Anexa B

Subiecte-tip pentru examen

1. Clasificatorul liniar, funcţia discriminant liniară.
2. Regula hebbiană.
3. Regula Windrow-Hoff.
4. Deduceţi regula delta pentru un neuron din condiţia de minimizare a erorii.
5. Demonstraţi că funcţia energiei computaţionale:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \sum_{i=1}^{n} G_i \int_0^{v_i} f_i^{-1}(z)dz$$

   într-o reţea Hopfield cu timp continuu este necrescătoare.
6. Principiul de funcţionare a unui convertor A/D pe 2 biţi implementat printr-o reţea de tip gradient.
7. Teorema perceptronului.
8. Algoritmul de stocare a pattern-urilor prototip într-o memorie autoasociativă recurentă unipolară.
9. Aproximarea unei funcţii continue prin reţele neurale feedforward.
10. Alegerea arhitecturii unei reţele neurale feedforward cu un strat ascuns.
11. Teorema lui Liapunov.
12. Problema zgomotului într-un asociator liniar.
13. Reprezentarea unei funcţii booleene printr-o reţea neurală feedforward.
14. Adăugarea / eliminarea unui pattern la o / dintr-o memorie autoasociativă recurentă.
15. Influenţa factorilor χ şi η în algoritmul backpropagation.

16. Demonstraţi că funcţia energiei computaţionale:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \mathbf{T}^t\mathbf{v}$$

    într-o reţea Hopfield cu timp discret este necrescătoare.

17. Perioada refractară a unui neuron biologic.

18. Perioada de însumare latentă a unui neuron biologic.

19. Algoritmul backpropagation este convergent?

20. Care este numărul maxim de regiuni liniar separabile folosind J neuroni ascunşi?

21. Într-o reţea Hopfield cu timp continuu, energia scade cât mai rapid?

22. De ce algoritmul de actualizare la reţelele Hopfield trebuie să fie stochastic asincron?

23. Cum putem găsi punctele de stabilitate ale unei reţele Hopfield de tip gradient?

24. Ce este un asociator liniar?

25. Teorema fundamentală a geneticii.

26. Paralelismul implicit al algoritmilor genetici.

27. Mulţime fuzzy.

28. Logica fuzzy.

29. Rolul intrării fixe în algoritmul de instruire a unei reţele feedforward de perceptroni discreţi.

30. Teza Church-Turing.

31. Maşina Turing.

32. Adăugaţi relaţiile de incluziune corespunzătoare între următoarele mulţimi: maşini Turing, automate celulare, reţele neurale, reţele de automate.

33. Teorema lui Judd referitoare la complexitatea instruirii reţelelor neurale feedforward multistrat.

34. Capacitatea memoriei autoasociative recurente.

35. De ce în cazul memoriei asociative bidirecţionale se preferă procesarea sincronă?

36. Principiul de funcţionare al reţelei Hamming.
37. Principiul de funcţionare al reţelei MAXNET.
38. Care este scopul calibrării supervizate a unui clasificator neural?
39. Principiul de funcţionare a reţelei Kohonen.
40. Ce este o hartă Kohonen?
41. Principiul de funcţionare a reţelei neurale fuzzy.
42. Principiul de funcţionare a reţelei neurale cu funcţie de activare radială.
43. Teorema lui Judd.
44. Teorema lui Hecht-Nielsen.
45. Teorema lui Kolmogorov.

Anexa C

Link-uri

- Neural Networks Commercial Software Tools
  http://www.emsl.pnl.gov:2080/proj/neuron/neural/systems/software.html
- SNNS - Stuttgart Neural Network Simulator
  http://www-ra.informatik.uni-tuebingen.de/SNNS/
- OFAI Library Information System Biblio - Neural Networks
  http://www.ai.univie.ac.at/oefai/nn/conn_biblio.html
- An Introduction to Genetic Algorithms
  http://www.cs.qub.ac.uk/~/M.Sullivan/ga/ga_index.html
- The Genetic Algorithms Archive
  http://www.aic.nrl.navy.mil/galist/
- David W. Aha Machine Learning Page
  http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html

Bibliografie

[1] Davis, L. "Handbook of Genetic Algorithms", Van Nostrand Reinhold, 1991.

[2] Dumitrescu, D., H. Costin "Reţele neuronale; teorie şi aplicaţii". Teora, Bucureşti, 1999.

[3] Freeman, J.A., D.M. Skapura "Neural Networks: Algorithms, Applications, and Programming Techniques". Addison-Wesley, Reading, Mass., 1991.

[4] Goldberg, D.E. "Genetic Algorithms - in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[5] Hagan, M. T., H.B. Demuth, M. Beale "Neural Network Design". PWS Publishing Co., Boston, 1996.

[6] Hassoun, M.H. "Fundamentals of Artificial Neural Networks". The MIT Press, Cambridge, Mass., 1995.

[7] Haykin, S. "Neural Networks - A Comprehensive Foundation". Macmillan College Publishing Company, New York, 1999 (second edition).

[8] Hecht-Nielsen, R. "Neurocomputing", Addison-Wesley, Reading, MA, 1990.

[9] Kosko, B. "Neural Networks and Fuzzy Systems", Prentice Hall, Englewood Cliffs, 1992.

[10] Masters, T. "Practical Neural Network Recipes in C++". Academic Press, Boston, 1993.

[11] Michalewicz, Z. "Genetic Algorithms + Data Structures = Evolution Programs". Springer-Verlag, Berlin, 1994 (second edition).

[12] Muller, B., J. Reinhardt "Neural Networks - An Introduction", Springer-Verlag, Berlin, 1990.

[13] Negoiţă, C.V., D.A. Ralescu "Mulţimi vagi şi aplicaţii ale lor", Ed. Tehnică, Bucureşti, 1974.

[14] Ştefan, Gh. "Funcţie şi structură în sistemele digitale", Ed. Academiei, Bucureşti, 1991.

[15] Tou, J.T., R.C. Gonzales "Pattern Recognition Principles", Reading, Addison-Wesley, 1974.

[16] Tsoukalas, L.H., R.E. Uhrig "Fuzzy and Neural Approaches in Engineering". John Wiley & Sons, New York, 1997.

[17] Wasserman, P.D. "Advanced Methods in Neural Computing", Van Nostrand Reinhold Inc., Computer Science Press, NY, 1993.

[18] Zurada, J. "Introduction to Artificial Neural Systems". West Publishing Company, St. Paul, 1992.

Prelucrarea datelor

1. Încărcați fișierul de date ***Iris*** de la https://archive.ics.uci.edu/ml/datasets/iris.

2. Pentru fiecare coloană cu valori numerice calculați valoarea minima, maximă, medie, mediană.
3. Normalizați valorile de pe fiecare coloană folosind formula $x_{norm} = \frac{x-min}{max-min}$.
4. Calculați suma ponderată a valorilor numerice de pe fiecare linie înmulțind fiecare valoare cu un coeficient pe care îl numim pondere și salvați rezultatele într-o nouă coloană a tabelului vostru. Folosiți următoarele valori pentru ponderi: [0.2, 1.1, -0.9, 1].

Exemplu

Pentru citirea valorilor numerice din fișierul iris.data și încărcarea într-o listă, puteți folosi codul de mai jos:

```
import csv
myList = []

with open('c:\Users\User\Desktop\iris.data') as csvfile:
 readCSV = csv.reader(csvfile, delimiter=',')
 for row in readCSV:
  if len(row) == 5:
   myList.append([float(row[0]), float(row[1]), float(row[2]), float(row[3])])
```

Puteți folosi numpy pentru operații cu vectori. Convertiți mai întâi myList în array de tip numpy și folosiți apoi metodele din acest pachet asupra coloanelor din array.

```
import numpy as np
npArray = np.array(myList)

col0_min = np.min(npArray[:,0])
col0_norm = (npArray[:,0]-min(npArray[:,0])) / (max(npArray[:,0])-
min(npArray[:,0]))
```

Inteligenţă computaţională - laborator

Preliminarii

1. Operații cu vectori. Găsiți produsul scalar dintre vectorii **x** și **y**, lungimea lor și cosinusul unghiului dintre ei, unde **x**=$[2\ 1\ 2]^t$ și **y**=$[1\ -1\ 4]^t$ (anexa A.1).
2. Implementaţi clasificatorul realizat cu o singură unitate, după modelul din figura 1.1, pag. 11. Folosiţi-l pentru a clasifica cele 8 puncte care reprezintă vârfurile unui cub, ca în exemplul din secţiunea 1.1.
3. Scrieţi un program care implementează reţeaua de memorie din figura 1.5, pagina 14. Folosiţi pentru testarea reţelei aceleaşi pattern-uri ca la exemplul anterior. Care sunt ieşirile stabile ale reţelei?

Inteligenţă computaţională - laborator

Metoda gradientului

Gradientul într-un punct al unei funcţii reprezintă direcţia creşterii maxime a acelei funcţii. Gradientul reprezintă tangenta la curbă pe direcţia de creştere şi este nul în punctele de maxim şi minim. Având funcţia $E(\mathbf{x})$, unde $\mathbf{x}$ este un vector cu n componente, gradientul se defineşte prin:

$$\nabla E_{\mathrm{x}} = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix}$$

Metoda gradientului este un algoritm iterativ de optimizare care constă în ***găsirea maximului sau a minimului local al unei funcţii*** folosind gradientul, respectiv gradientul său negat.

Minimul unei funcţii se poate găsi folosind metoda gradientului astfel: $\mathbf{x}_{n+1} = \mathbf{x}_n - c\nabla f(\mathbf{x}_n)$, unde c este o constantă pozitivă, suficient de mică astfel încât $f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq \ldots \geq f(\mathbf{x}_n) \geq f(\mathbf{x}_{n+1})$. Algoritmul foloseşte o iniţializare aleatoare a lui $\mathbf{x}_0$. Constanta de învăţare c determină o convergenţă a valorilor $\mathbf{x}_n$ către optim mai rapidă şi mai brută daca este mare sau mai lentă şi mai fină dacă este mică. Se poate alege, de exemplu, c=0,01, dar se poate opta şi pentru modificarea acestei valori chiar în timpul rulării programului. Algoritmul continuă până când modificările componentelor lui $\mathbf{x}_n$ scad sub un prag acceptabil, de exemplu 0,00001.

Pentru funcţii $f(x)$ care depind de o singură variabilă, relaţia de mai sus se rescrie astfel: $x_{n+1} = x_n - cf'(x_n)$.
Pentru funcţii $f(x,y)$ care depind de două variabile, relaţia se rescrie prin:

$$\begin{cases} x_{n+1} = x_n - c\frac{\partial f(x_n)}{\partial x} \\ y_{n+1} = y_n - c\frac{\partial f(y_n)}{\partial y} \end{cases}$$

1. (70%) Scrieţi un program care găseşte minimul funcţiei $f(x) = 6x^2-12x+1$ prin metoda gradientului. Folosiţi diverse valori pentru c. Afişaţi valoarea lui $f(x)$ după fiecare iteraţie.
2. (30%) Scrieţi un program care găseşte prin metoda gradientului minimul funcţiilor:
   - $g(x,y) = x^2+2y^2$
   - $h(x,y) = (1-x)^2+100(x-y^2)^2$.

Inteligenţă computaţională - laborator

Regula Hebb

1. Scrieţi un program care implementează învăţarea hebbiană din exemplul 4, secţiunea 2.5, pagina 43.
2. Folosind implementarea de mai sus, repetaţi invăţarea hebbiană pentru pattern-urile de la exerciţiul 7, pagina 49.

Inteligenţă computaţională - laborator

Regula “câștigătorul ia tot”

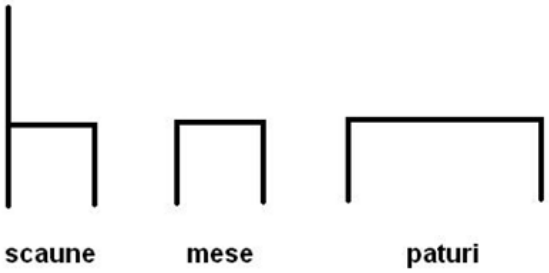
Instruirea prin regula *câştigătorul ia tot* se bazează pe învăţarea competitivă prin care, în urma unui proces de selecţie, un pattern de intrare activează un singur prototip care este declarat câştigător. Acest prototip este, apoi, actualizat.
Algoritmul poate fi sintetizat astfel:

1. Se iniţializează aleator prototipurile $\mathbf{w}_1$, $\mathbf{w}_2$, ..., $\mathbf{w}_p$. Se iniţializează constanta de instruire c. Se stabileşte numărul de epoci de instruire. Se citesc pattern-urile de instruire $\mathbf{x}_1$, $\mathbf{x}_2$,...,$\mathbf{x}_n$.
2. Având un pattern $\mathbf{x}_i$ dintre cele n, câştigător este prototipul $\mathbf{w}_m$ pentru care
$$||\mathbf{x}_i-\mathbf{w}_m|| = \min_j||\mathbf{x}_i-\mathbf{w}_j||,\ j=1...p.$$
3. Se actualizează prototipul câştigător prin relaţia
$$\mathbf{w}_m^{(k+1)} = \mathbf{w}_m^{(k)}+c(\mathbf{x}_i-\mathbf{w}_m).$$
4. Se revine la pasul 2 cu un nou vector de instruire, până la epuizarea setului de n pattern-uri.
5. Se revine la pasul 2 pentru o nouă epocă de instruire în care se reiau toate pattern-urile de instruire.

Implementaţi algoritmul de instruire *câştigătorul ia tot* pentru p=3 prototipuri $\mathbf{w}_1$, $\mathbf{w}_2$ şi $\mathbf{w}_3$ reprezentate ca vectori bidimensionali, având la dispoziţie următoarele 9 pattern-uri de instruire $\mathbf{x}_1$, $\mathbf{x}_2$ ... $\mathbf{x}_9$:

```
45 85
50 43
40 80
55 42
200 43
48 40
195 41
43 87
190 40
```

Aceste pattern-uri reprezintă lăţimea şi înălţimea unor obiecte de mobilier asemnănătoare celor din imaginea de mai jos.



Iniţializaţi aleator cele trei prototipuri. Folosiţi mai întâi constanta de instruire c = 1, apoi testaţi şi alte valori. O *epocă* a instruirii se încheie după ce au fost folosite toate pattern-urile. Stabiliţi-vă de la început numărul de epoci, acesta reprezentând criteriul de oprire a învăţării.
La final afişaţi prototipurile şi pattern-urile de instruire asociate fiecărui prototip.

Inteligenţă computaţională - laborator

Tema

Învăţarea unui neuron prin regula perceptronului: Exerciţiul 8, pagina 50.

Inteligenţă computaţională - laborator

Algoritmi genetici

Rezolvaţi, la alegere, unul dintre cele două exerciţii de mai jos. Se acordă 70% din punctaj pentru implementarea paşilor de iniţializare a populaţiei şi selecţie, iar pentru implementarea operatorilor genetici se acordă şi diferenţa de 30%. Detaliile legate de modul în care se scrie un algoritm genetic sunt in capitolul 10 din materialul de curs.

1. Scrieţi un program care implementează un algoritm genetic pentru exemplul din secţiunea 10.2.

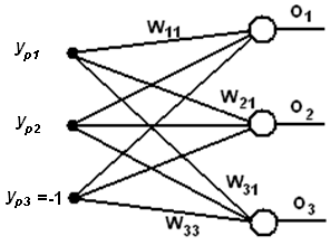
2. Presupunem că avem la dispoziţie 10 cărţi de joc numerotate de la 1 la 10. Cum trebuie să împărţim cele 10 cărţi în 2 pachete astfel încât suma cărţilor din primul pachet să fie cât mai apropiată de 36, iar produsul cărţilor din al doilea pachet să fie cât mai apropiat de 360? Rezolvaţi această problemă cu ajutorul unui algoritm genetic. O idee de implementare este cea de mai jos, dar puteţi opta si pentru propria voastră solutie.
   a. *Codificarea*. Fiecare cromozom constă din 10 cifre binare, de ex. 1100001010. Interpretăm aceasta codificare în felul următor: c[*i*] = 0 – cartea cu cifra *i* face parte din primul pachet şi c[*i*] = 1 – cartea cu cifra *i* face parte din al doilea pachet;
   b. *Initializarea populaţiei*. Se generează aleator 50 de cromozomi;
   c. *Evaluarea cromozomilor*. Se calculează valoarea functiei obiectiv pentru fiecare cromozom si se calculează apoi probabilitatea de selectie;
   d. *Selecţia cromozomilor pentru generaţia următoare*. Se aplică ruleta selecţiei;
   e. *Aplicarea operaţiilor genetice*. Se realizează împerecherea cromozomilor vecini şi mutaţiile.
   f. *Se reiau paşii c, d şi e până când se îndeplineşte criteriul global de eroare.*

Inteligenţă computaţională - laborator

Regula delta

Regula delta este o metodă de instruire supervizată care se aplică unei reţele monostrat de perceptroni de tip continuu. În figura de mai jos avem o reţea care foloseşte la intrare pattern-uri cu două componente. Întotdeauna se adaugă o intrare suplimentară cu valoarea constantă -1. Între intrări şi ieşiri se realizează toate legăturile posibile. În exemplul nostru se obţin 3x3 conexiuni notate prin w_{ji}, unde j este indicele perceptronului, iar i este indicele intrării. De exemplu, conexiunea care leagă intrarea 1 de perceptronul 2 este w_{21}, conexiunea care leagă intrarea suplimentară cu perceptronul 3 este w_{33}.

Funcţia de activare a perceptronilor este continuă. Vom folosi varianta continuă bipolară care are formula $f(net) = \frac{2}{1+e^{-net}} - 1$.



Algoritmul de mai jos implementează *regula delta*:

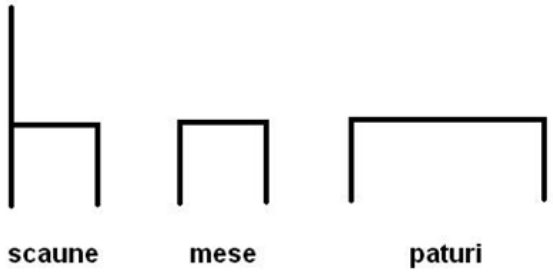
1. Se citesc pattern-urile de instruire $\mathbf{y}_1$, $\mathbf{y}_2$,...,$\mathbf{y}_N$ şi ieşirile dorite $\mathbf{d}_1$, $\mathbf{d}_2$,...,$\mathbf{d}_N$. Se scalează toate valorile din setul de date. Se stabileşte numărul M de intrări şi numărul K de perceptroni. Se iniţializează ponderile $\mathbf{w}_{ji}$ cu valori aleatoare din intervalul [-1;1]. Se iniţializează constanta de instruire c. Se stabileşte eroarea maximă E_{max}. Se iniţializează E=0.
2. Având un pattern $\mathbf{y}_p=[y_{p1},\ldots,y_{pi},\ldots,y_{pM}]$ dintre cele n, se calculează ieşirile fiecărui perceptron:
$$o_j=f(w_{j1}y_{p1}+w_{j2}y_{p2}+\ldots+w_{jM}y_{pM}),\ j=1\ldots K,$$
unde K este numărul de perceptroni, M este numărul de intrări, iar f este funcţia de activare bipolară.
3. Se actualizează toate ponderile reţelei conform regulii:
$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + c(d_{pj}-o_j)(1-o_j^2)y_{pi},$$
unde $\mathbf{d}_p=[d_{p1},\ldots,d_{pj},\ldots,d_{pK}]$ reprezintă ieșirile dorite pentru pattern-ul $\mathbf{y}_p$.
4. Se calculează eroarea cumulată:
$$E^{(t+1)}=E^{(t)} + (d_{p1}-o_1)^2 + (d_{p2}-o_2)^2 + \ldots + (d_{pK}-o_K)^2.$$
5. Se revine la pasul 2 cu un nou vector de instruire, până la epuizarea setului de N pattern-uri.
6. Dacă $E > E_{max}$, se revine la pasul 2 pentru o nouă epocă de instruire în care se reiau toate pattern-urile de instruire şi se reiniţializează E=0.

Implementaţi *regula delta* pentru o reţea de K=3 perceptroni care foloseşte pattern-uri bidimensionale cărora li se adaugă a treia intrare -1, $\mathbf{y}_1$, $\mathbf{y}_2$ ... $\mathbf{y}_9$ cu ieşirile dorite $\mathbf{d}_1$, $\mathbf{d}_2$ ... $\mathbf{d}_9$:

```
45 85        1  -1 -1
50 43        -1  1 -1
40 80        1  -1 -1
55 42        -1  1 -1
200 43       -1 -1  1
48 40        -1  1 -1
195 41       -1 -1  1
43 87        1  -1 -1
190 40       -1 -1  1
```

Aceste pattern-uri reprezintă lăţimea şi înălţimea unor obiecte de mobilier asemnănătoare celor din imaginea de mai jos, iar cele trei clase sunt codificate astfel:

- scaunele prin (1, -1, -1)
- mesele prin (-1, 1, -1)
- paturile prin (-1, -1, 1).



La final afişaţi pattern-urile de instruire şi răspunsurile celor trei perceptroni pentru fiecare pattern.

Inteligenţă computaţională

Tema

Algoritmul *K-means clustering* este o metodă de determinare a clusterelor pe care le formează mai multe pattern-uri. Procedura este una de instruire nesupervizată. Se presupune cunoscut numărul *K* al clusterelor, acesta fiind un parametru stabilit *a priori*.

Fiecare cluster are un centroid. Algoritmul lucrează cu *K* clustere, deci *K* dintre punctele folosite la instruire vor fi centriozii celor *K* clustere. Întrucât iniţializarea centroizilor se face aleator, există posibilitatea ca mai multe rulări ale algoritmului să conducă la rezultate diferite.

Fiecare punct este asociat clusterului determinat de cel mai apropiat centriod. Distanţa dintre punct şi centriod poate fi calculată, de exemplu, ca distanţă euclidiană, dar se poate opta şi pentru alte variante.

Algoritmul este următorul:

1. *Se aleg aleator K puncte ca centroizi iniţiali.*
2. *Se formează K clustere prin asignarea tuturor punctelor celor mai apropiaţi centroizi.*
3. *Se recalculează centroizii astfel: noul centriod va fi centrul de greutate determinat de punctele clusterului.*
4. *Se reiau paşii 2 şi 3 până când centroizii nu se mai modifică.*

Aplicaţi algoritmul *K-means clustering* pentru următoarele puncte, cu *K*=3:

```
P1: 45 85
P2: 50 43
P3: 40 80
P4: 55 42
P5: 200 43
P6: 48 40
P7: 195 41
P8: 43 87
P9: 190 40
```

Listaţi cei 3 centroizi şi punctele asociate fiecărui cluster.

Pentru o descriere detaliată a algoritmului puteţi consulta şi:
https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxzdWpveXNhaGFzaXRlfGd4OjJhMjI5ZTU1ZTgxMDRhYjk

Inteligenţă artificială - laborator

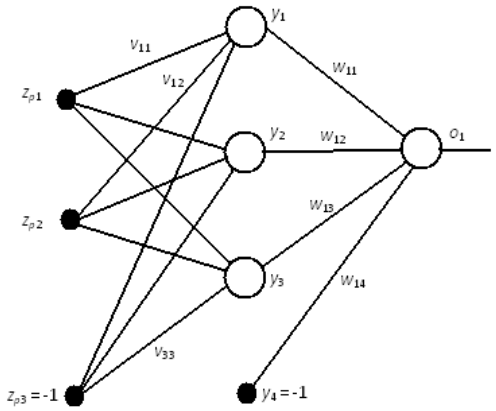
Algoritmul “Backpropagation of error”

Algoritmul de instruire *backpropagation of error* implementează *regula delta generalizată* care este o metodă de instruire supervizată a unei reţele multistrat de perceptroni de tip continuu. Spre deosebire de regula delta care funcţionează pentru pattern-uri liniar separabile, regula delta generalizată se poate aplica şi pattern-urilor neseparabile liniar. Algoritmul *backpropagation of error* a fost propus de D.E. Rumelhart, G.E. Hinton şi R.J. Williams într-un articol apărut în 8 octombrie 1986 în revista *Nature*. O copie a articolului se găseşte la http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf.

În figura de mai jos avem o reţea care foloseşte:

- La intrare pattern-uri cu două componente la care se adaugă o intrare suplimentară cu valoarea constantă -1. Notăm cu z_{p1}, z_{p2} şi z_{p3} intrările reţelei neurale, unde p este indicele pattern-ului curent de instruire extras din setul de n pattern-uri de instruire;
- Un strat ascuns cu 3 perceptroni de tip continuu ale căror ieşiri devin intrările neuronului de pe stratul de ieşire. Ieşirile neuronilor ascunşi sunt notate cu y_1, y_2 şi y_3. Legăturile dintre intrări şi neuronii ascunşi sunt notate cu v_{11}, v_{12} ... v_{33}. În exemplul nostru sunt 3x3 astfel de conexiuni.
- Un strat de ieşire format dintr-un singur perceptron continuu. Intrările sale sunt y_1, y_2, y_3 şi intrarea suplimentară y_4=-1. Legăturile dintre neuronii ascunşi şi neuronul de ieşire sunt notate cu w_{11}, w_{12} şi w_{13}, la care se adaugă w_{14}. Ieşirea reţelei este notată cu o_1.

Funcţia de activare a perceptronilor este continuă. Vom folosi varianta continuă bipolară care are formula $f(net) = \frac{2}{1+e^{-net}} - 1$.



Algoritmul *backpropagation of error* se poate implementa astfel:

1. Se citesc pattern-urile de instruire $\mathbf{z}_1$, $\mathbf{z}_2$,...,$\mathbf{z}_n$ şi ieşirile dorite $\mathbf{d}_1$, $\mathbf{d}_2$,...,$\mathbf{d}_n$.Pattern-urile sunt extinse cu o componentă suplimentară cu valoarea -1. Se stabileşte numărul de intrări şi numărul de perceptroni. Se iniţializează aleator ponderile $\mathbf{v}_{ji}$ şi $\mathbf{w}_{kj}$. Se iniţializează constanta de instruire c. Pentru a accelera instruirea puteţi sa iniţializaţi constanta de instruire c cu o valoare mare, de exemplu c=50, şi să o scădeţi treptat. Se stabileşte eroarea maximă E_{max}. Se iniţializează E=0.
2. Având un pattern $\mathbf{z}_p$ dintre cele n, se calculează ieşirile fiecărui perceptron:

$$y_j=f(v_{j1}z_{p1}+v_{j2}z_{p2}+...+v_{jI}z_{pI}),\ j=1...J,$$

$$o_k=f(w_{k1}y_1+w_{k2}y_2+...+w_{kJ}y_J),\ k=1...K,$$

unde *I* este numărul de intrări, J este numărul de perceptroni ascunşi, *K* este numărul de perceptroni de ieşire, iar *f* este funcţia de activare bipolară continuă.

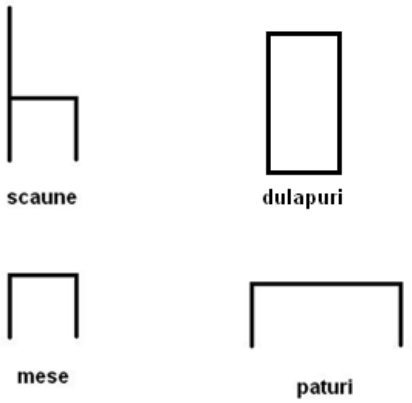
3. Se calculează semnalele de eroare:
$$\delta_{yj}=0.5(1-y_j^2)(\delta_{o1}w_{1j}+...+\delta_{oK}w_{Kj})$$
$$\delta_{ok}=0.5(d_j-o_j)(1-o_j^2)$$
4. Se actualizează toate ponderile reţelei conform regulii:
$$v_{ji}^{(t+1)} = v_{ji}^{(t)}+c\delta_{yj}z_{pi}$$
$$w_{kj}^{(t+1)} = w_{kj}^{(t)}+c\delta_{ok}y_j.$$
5. Se calculează eroarea cumulată:
$$E^{(t+1)}=E^{(t)}+0.5(d_1-o_1)^2 + 0.5(d_2-o_2)^2 + ... + 0.5(d_K-o_K)^2.$$
6. Se revine la pasul 2 cu un nou vector de instruire, până la epuizarea setului de *n* pattern-uri.
7. Dacă $E > E_{max}$, se revine la pasul 2 pentru o nouă epocă de instruire în care se reiau toate pattern-urile de instruire şi se reiniţializează E=0.

Implementaţi *regula delta generalizată* pentru o reţea cu două straturi de perceptroni, ca în figura de mai sus. Reţeaua foloseşte pattern-uri bidimensionale cărora li se adaugă a treia intrare -1, $\mathbf{z}_1$, $\mathbf{z}_2$ ... $\mathbf{z}_{12}$ cu ieşirile dorite $\mathbf{d}_1$, $\mathbf{d}_2$ ... $\mathbf{d}_{12}$:

45 85	1	48 40	-1
50 43	-1	195 41	1
40 80	1	43 87	1
187 107	-1	192 105	-1
55 42	-1	190 40	1
200 43	1	188 100	-1

Aceste pattern-uri reprezintă lăţimea şi înălţimea unor obiecte de mobilier asemnănătoare celor din imaginea de mai jos. Obiecte sunt depozitate în două incinte separate, scaunele şi paturile într-una, iar mesele şi dulapurile în cealaltă. Obiectele trebuie clasificate automat astfel:

- Scaunele şi dulapurile în clasa 1;
- Mesele şi paturile în clasa -1.



La final afişaţi pattern-urile de instruire şi răspunsul reţelei pentru fiecare pattern.