

Despre autori

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

Angel Cațaron a făcut parte din prima generație de absolvenți care au frecventat acest curs. În perioada 1997-1999, el a predat cursul de Rețele neurale la Universitatea "Transilvania", specializarea Electronică și calculatoare. Pe lângă aceasta, el este coordonatorul orelor de laborator pentru acest curs și este doctorand în domeniul rețelelor neurale.

Prefață

Chiar dacă acest material este suportul cursului de Rețele neurale, de-a lungul timpului acest curs a început să conțină capitole noi, la interferența rețelelor neurale cu sistemele fuzzy și algoritmi genetici. Aceasta este în concordanță cu tendințele actuale în lume.

De aceea, am preferat denumirea de *inteligență computațională*. Inteligența computațională, așa cum a fost definită de Bezdek¹, are ca obiectiv modelarea inteligenței biologice. Din acest punct de vedere, ea este similară domeniului numit *inteligență artificială*. Spre deosebire de inteligența artificială, care este bazată pe noțiunea de cunoștință, inteligența computațională este o modelare numerică a inteligenței biologice. Putem vorbi deci de trei tipuri de "inteligență": biologică, artificială și computațională.

Inteligența computațională este formată din următoarele subdomenii: rețele neurale, algoritmi genetici, programare evoluționară, sisteme fuzzy, viață artificială. În contextul inteligenței computaționale, toate aceste subdomenii sunt legate de modelarea numerică a inteligenței biologice.

Acest curs acoperă majoritatea subdomeniilor inteligenței computaționale. Fiecare capitol oferă, pe lângă informații teoretice, câte o secțiune de aplicații, urmată de un set de exerciții, cele marcate prin (C) fiind destinate implementării pe calculator.

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

Cuprins

1	Preliminarii în calculul neural	9
1.1	Calculul neural: exemple	10
1.2	Istoricul dezvoltării rețelelor neurale	17
1.3	Viitorul	18
2	Concepte fundamentale	19
2.1	Neuronii biologici și modelele lor artificiale	19
2.2	Modelarea rețelelor neurale	26
2.3	Învățare și adaptare	31
2.4	Reguli de învățare	32
2.5	Exemple	40
2.6	Exerciții	48
3	Perceptroni monostrat	53
3.1	Clasificare	53
3.2	Funcții discriminant	55
3.3	Clasificatori liniari	56
3.4	Perceptronul discret ca dihotomizator liniar	59
3.5	Perceptronul continuu ca dihotomizator liniar	63
3.6	Teorema de convergență a perceptronului	66
3.7	Rețele monostrat de perceptroni	68
3.8	Exemple	69
3.9	Exerciții	76
4	Rețele neurale feedforward multistrat	79
4.1	Clasificarea pattern-urilor liniar neselectabile	79
4.2	Regula de învățare delta	80
4.3	Regula delta generalizată	84
4.4	Algoritmul de instruire backpropagation	87
4.5	Factori ai învățării	89
4.6	Aproximatori universali	92
4.7	Teorema lui Kolmogorov și rețelele neurale	94
4.8	Aplicații	95
4.9	Exemple	96

4.10	Exerciții	100
5	Rețele neurale feedback monostrat	107
5.1	Rețele Hopfield cu timp discret	107
5.2	Rețele Hopfield cu timp continuu	110
5.3	Aplicație: problema comis-voiajorului	114
5.4	Exemple	117
5.5	Exerciții	122
6	Memorii asociative	125
6.1	Concepte de bază	125
6.2	Asociatori liniari	127
6.3	Memorii autoasociative recurente	130
6.4	Analiza performanței	135
6.5	Memoria asociativă bidirecțională (MAB)	137
6.6	Exerciții	141
7	Rețele neurale cu auto-organizare	145
7.1	Rețelele Hamming și MAXNET	145
7.2	Instruirea nesupervizată a clusterelor	148
7.3	Hărți Kohonen	150
7.4	Exerciții	156
8	Rețele neurale RBF	159
8.1	Funcții radiale	159
8.2	O tehnică de grupare (clustering)	162
8.3	Discuție	163
9	Rețele neurale fuzzy	165
9.1	Logica fuzzy	165
9.1.1	De ce logică fuzzy?	165
9.1.2	Logica fuzzy și cea convențională	167
9.2	Rețele neurale fuzzy	169
9.2.1	Neuroni fuzzy	169
9.2.2	Structura unei RNF	171
9.2.3	Algoritmul de instruire a unei RNF	172
9.2.4	Analiza RNF	173
9.2.5	Rezultatele simulării	174
9.2.6	Concluzii	175
10	Algoritmi genetici	177
10.1	Introducere	177
10.2	Exemplu	178
10.3	Fundamente matematice	180
10.4	Exerciții	183

11 Puterea și complexitatea de calcul	185
11.1 Mașina Turing	185
11.2 Puterea de calcul a rețelelor neurale	187
11.3 Reprezentarea funcțiilor booleene	188
11.4 Complexitatea instruirii	190
12 Considerații epistemologice	193
12.1 Scopul unei rețele neurale	193
12.2 Funcțiile neurale biologice sunt localizate sau distribuite?	194
12.3 Este neliniaritatea esențială în calculul neural?	194
12.4 Deosebiri esențiale	195
12.5 Cum pot fi programate calculatoarele neurale	196
12.6 Poate creierul să se autoperceapă?	196
A Complemente matematice	199
A.1 Vectori și matrici	199
A.2 Forme pătratice	200
A.3 Elemente de geometrie analitică	201
A.4 Operația XOR	202
A.5 Iacobianul și hessianul	202
A.6 Probleme de optimizare	205
A.7 Metoda lui Euler (metoda tangentei)	206
A.8 Stabilitatea sistemelor dinamice neliniare	208
A.9 Variabile aleatoare	209
B Subiecte-tip pentru examen	211
C Link-uri	215
Bibliografie	217

Capitolul 1

Preliminarii în calculul neural

Calculul neural se efectuează pe o rețea densă de noduri și conexiuni. Aceste noduri lucrează în mod colectiv și simultan și se numesc *neuroni artificiali*, sau *neuroni*. Neuronii pot opera, de exemplu, ca sumatoare sau comparatoare. De obicei, neuronii lucrează în paralel și sunt configurați în arhitecturi regulate. Astfel, ei pot fi organizați pe nivele ierarhice și 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 rețelelor neurale este cunoscut și sub denumiri similare: neurocalcul, conexionism, procesare paralelă distribuită, sisteme adaptive, rețele cu autoorganizare etc. Această varietate indică de fapt tot atâtea perspective din care se studiază rețelele neurale.

Rețelele neurale funcționează ca rețele paralele distribuite. Caracteristica lor de bază este arhitectura. Unele rețele sunt caracterizate de comportarea lor în timp, de *dinamica* lor. Rețelele neurale diferă între ele prin modul de învățare: există o varietate de reguli de învățare care stabilesc când și cum se modifică ponderile conexiunilor. În fine, rețelele diferă prin viteza și eficiența de învățare.

Spre deosebire de calculatoarele convenționale, care sunt programate să efectueze anumite lucrări, majoritatea rețelelor neurale trebuie să fie învățate (sau instruite). Ele învață noi asocieri, noi pattern-uri, noi dependențe funcționale. După cum vom vedea mai târziu, faptul că rețelele învață reguli și algoritmi înlocuiește programarea necesară în calculul convențional. Utilizatorii rețelelor neurale nu specifică un algoritm care să fie executat de către un anumit neuron, cum s-ar întâmpla pe o mașină tradițională. În loc de aceasta, ei aleg o configurație care li se pare cea mai bună arhitectură, specifică toate caracteristicile neuronilor și ponderile inițiale, apoi aleg modul de instruire pentru rețea. În următoarea fază, sunt aplicate diferite date de intrare din care rețeaua își extrage cunoștințe, adică învață. Ca rezultat, rețeaua acumulează informație care poate fi apoi utilizată.

Calculul cu rețele neurale se situează între inginerie și inteligența artificială. Se folosesc tehnicile matematice ingineresti clasice, dar și metode euristice specifice inteligenței artificiale.

În acest sens vom răspunde la următoarele întrebări:

- Cum poate fi instruită eficient o rețea și cum se învață 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 cunoștințele acumulate într-o rețea?
- Care sunt aplicațiile tipice pentru calculul neural și cât de eficiente sunt aceste aplicații?

Rețelele neurale au atras atenția specialiștilor din numeroase discipline. Neurobiologii sunt interesați în modelarea rețelelor neurale biologice. Fizicienii sunt atrași de analogiile dintre rețelele neurale și sistemele dinamice neliniare pe care le studiază. Matematicienii sunt fascinați de potențialul modelării matematice aplicat în sistemele foarte mari și complexe. Inginerii în electronică și calculatoare aplică rețelele neurale în procesarea semnalelor și construiesc pe baza rețelelor neurale circuite integrate inteligente. Psihologii caută în rețelele neurale structurile prototip care modelează procesarea informației de către om. În fine, informaticienii sunt interesați în posibilitățile de calcul ale rețelelor masiv paralele în domeniile inteligenței artificiale, teoriei calculabilității, modelării și simulării etc.

1.1 Calculul neural: exemple

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

Clasificatori și sisteme automate de orientare în spațiu

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

Fie P_0, P_1, \dots, P_7 opt puncte în spațiul tridimensional. Mulțimea 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.

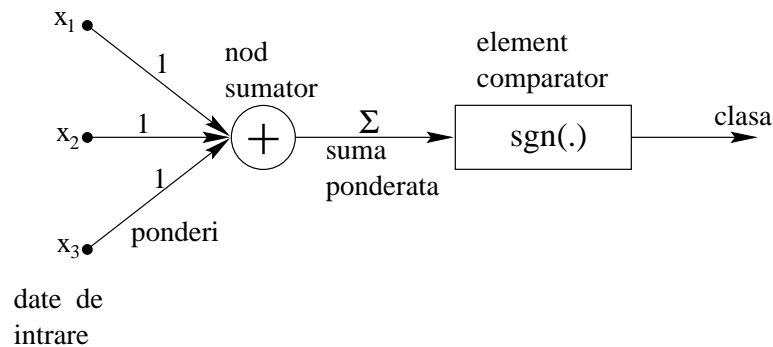


Figura 1.1: Clasificator realizat cu o singură unitate.

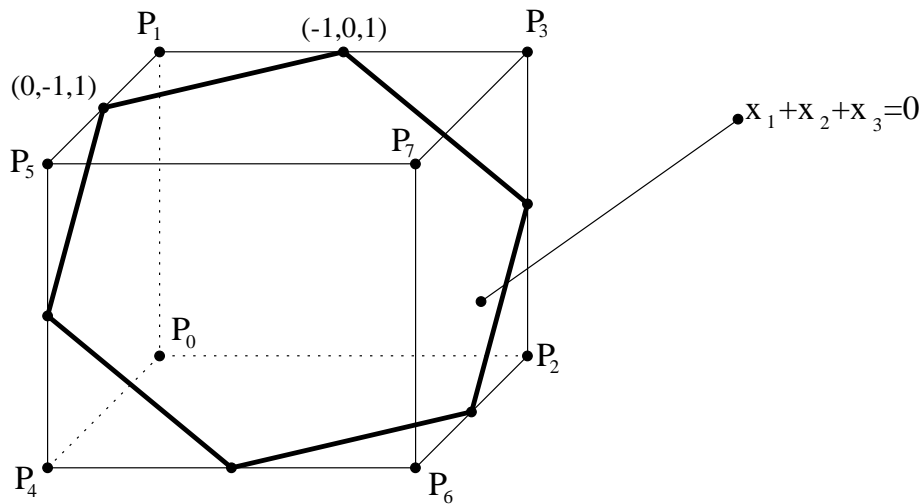


Figura 1.2: Partiționarea spațiului cartezian.

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

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

Această expresie descrie funcția de decizie a unui clasificator. Nu este necesară o instruire a acestui clasificator. Rețeaua 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) și este urmată de o comparare cu prag.

De fapt, se realizează o partiționare a spațiului 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 funcție continuă nu poate fi mai avantajoasă (fig. 1.3).

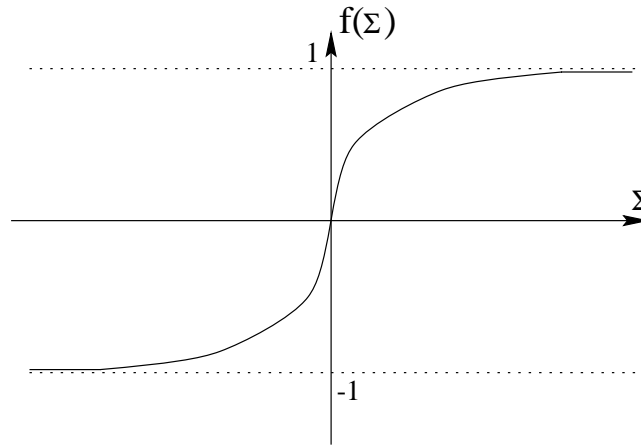


Figura 1.3: Funcție continuă de decizie.

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

Neurologii folosesc electro-encefalograma (EEG) care preia pe mai multe canale impulsurile electrice ale creierului. Evaluarea EEG este dificilă și de aceea se face de obicei de către neurologi calificați. 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 și 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 eșantionate de 200 sau 250 ori pe secundă într-o fereastră de 240 ms. Aceasta are ca rezultat obținerea a 48 sau 60 de eșantioane de date pentru fiecare canal. Aceste eșantioane trebuie evaluate și sunt introduse într-o rețea neurală de 40 de unități interconectate, cu caracteristici continue. Un total de 41 de unități aranjate pe trei nivele ierarhice procesează datele. Două unități de ieșire sunt utilizate pentru identificarea vârfurilor de semnal.

Rețeaua a fost elaborată de o echipă de ingineri și neurologi. După ce a fost instruită, rețeaua a dat rezultate excelente dovedindu-și utilitatea în spitale.

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

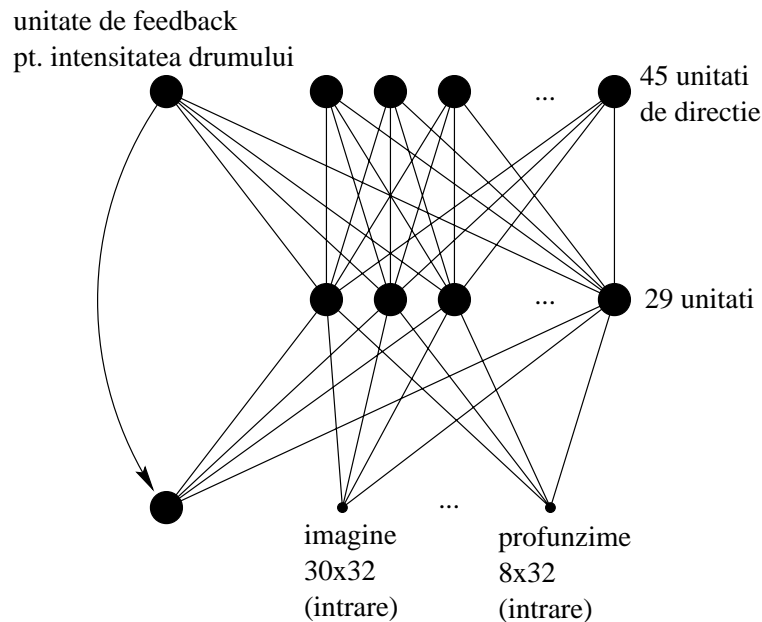


Figura 1.4: Arhitectura rețelei ALVINN.

Informația video este o rețină de 30x32 care sesizează în albastru (oferă cel mai bun contrast). Unitatea de feedback reglează contrastul. Cele 1217 intrări conduc spre 29 unități cu funcție continuă. Unitatea din mijlocul celor 45 de unități de ieșire arată cât de puternică este tendința de a merge înainte. Unitățile din stânga-dreapta reprezintă tendințele de a o lua la stânga-dreapta. Unitățile din extremitățile stângă și dreaptă corespund virajelor stânga-dreapta cât mai accentuate.

ALVINN a fost instruit prin imagini sintetizate pe calculator. Performanțele obținute au fost comparabile cu caracteristicile celor mai bune sisteme tradiționale de orientare prin vedere artificială.

Capacitatea sistemului ALVINN de a păstra direcția drumului nu a fost implementată prin programare (algoritm), ci se bazează pe cunoștințele 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 observații interesante. Astfel, rețeaua s-a comportat mai bine după ce a fost instruită să refacă erori de conducere. Prin aceasta, rețeaua și-a dezvoltat măsurile de corectare a conducerii.

Memorie simplă și restaurarea pattern-urilor

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

rețele *memorii*.

Fie rețeaua simplă din figura 1.5.

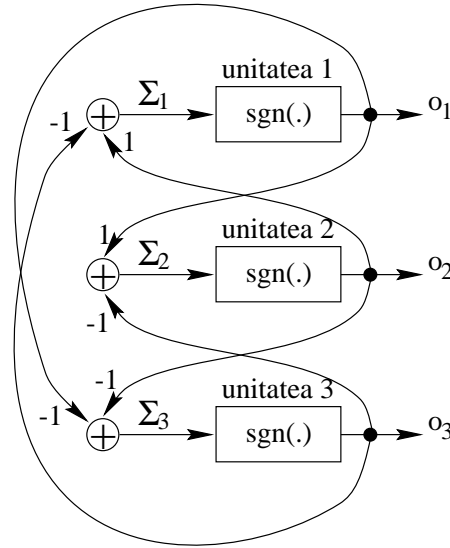


Figura 1.5: Rețea memorie.

Rețeaua constă din trei unități care calculează valorile funcției signum, trei noduri de însumare și șase ponderi care pot fi ± 1 . Semnalele care trec prin ponderi sunt înmulțite cu valoarea ponderilor. Presupunem că rețeaua este inițializată la ieșire cu $o_1 = o_2 = o_3 = 1$ (cazul 1, tab. 1.1). Când se permite rețelei să calculeze, intrările la unitățile 1 și 2 sunt 0, în timp ce la unitatea 3 este -2. Ca rezultat, o_1 și o_2 nu se modifică, deoarece $\text{sgn}(0)$ nu este definit, în timp ce o_3 devine -1. Urmează cazul 2, care este o ieșire finală deoarece nu se mai poate modifica. Cazurile 3 și 4 sunt alte exemple de tranziții posibile. Observăm că și cazurile 3 și 4 duc apoi la cazul 2, operând câte o singură modificare.

Iată cum se reprezintă geometric aceste actualizări efectuate de rețeaua de tip memorie descrisă. Se vede că $P_6(1, 1, -1)$ este ieșirea stabilă a memoriei. Când o singură componentă a vectorului de inițializare binar (o_1, o_2, o_3) diferă de P_6 , rețeaua corectează această componentă. Ieșirea este apoi menținută 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 ușurință la lumea reală a aplicațiilor.

Probleme de optimizare

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

Tabelul 1.1: Exemple de tranziții ale rețelei tip memorie. X înseamnă $\text{sgn}(0)$, iar ieșirile încadrate sunt cele care se modifică.

Cazul	Nr. unității	Ieșirea actuală	Σ	$\text{sgn}(\Sigma)$	Ieșirea următoare
1	1	1	0	X	1
	2	1	0	X	1
	3	1	-2	-1	-1
2	1	1	2	1	1
	2	1	2	1	-1
	3	-1	-2	-1	-1
3	1	-1	2	1	1
	2	1	0	X	1
	3	-1	0	X	-1
4	1	1	0	X	1
	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_1v_0 , ($v_1, v_0 \in \{0, 1\}$), astfel încât

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

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

Rețeaua constă din două unități cu caracteristici continue, fiecare cu răspuns în intervalul dintre 0 și 1. La cele două unități, rețelei 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 prezența restricției $v_0, v_1 \in \{0, 1\}$. Această problemă de minimizare este rezolvabilă de către o anumită clasă de rețele neurale. Caracteristic acestei clase este că se minimizează așa numita *funcție de energie* pe parcursul calculului.

Proprietatea de minimizare a energiei este de importanță foarte mare și se formulează astfel: rețeaua își caută singură energia ei minimă și se stabilizează acolo. O serie de probleme de optimizare se transpun direct în minimizarea funcției de energie a unei rețele neurale. Minimizarea energiei de către rețea poate fi considerată analogă minimizării erorii de conversie A/D.

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

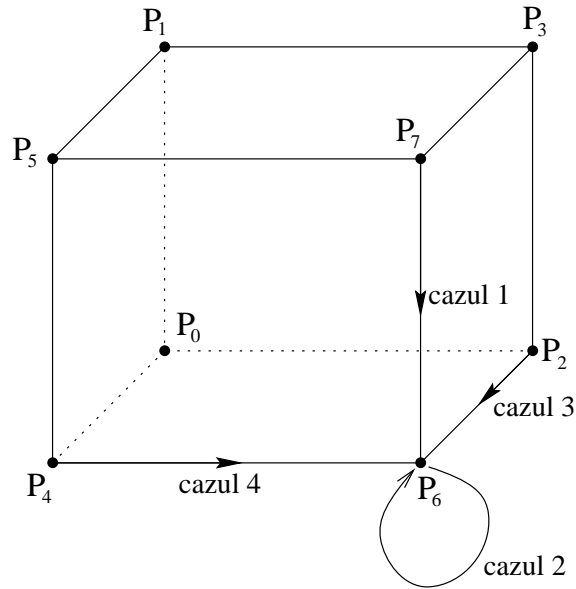


Figura 1.6: Actualizările efectuate de rețeaua de tip memorie.

Detectarea grupărilor și a trăsăturilor

O clasă importantă de rețele neurale pot fi utilizate pentru detectarea grupărilor de date. Aceste rețele sunt calate pe anumite aspecte de similaritate în datele evaluate. De exemplu, se poate să fim interesați 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 și trăsăturilor prezintă importante proprietăți de autoorganizare care sunt legate de inteligența artificială și teoria informației. Rețelele din această clasă au în general arhitecturi simple, dar subtilitățile apar în timpul procesului de autoorganizare.

Detectarea trăsăturilor se raportează la reducerea dimensionalității datelor. De exemplu, semnalul vorbirii constă din 15 canale de frecvență audio. Fonemele sunt deci descrise într-un spațiu 15-dimensional. Problema este că nu putem reprezenta astfel fonemele, deoarece capacitatea noastră de vizualizare se reduce la trei dimensiuni. Utilizând o rețea neurală, este posibil să reprezentăm spectrul 15-dimensional al vorbirii într-un tablou în plan. Secvența fonemelor unui cuvânt formează o traiectorie specifică în plan. Aceste *hărți fonotopice* pot fi foarte utile în construirea mașinilor de scris fonetice, în învățarea vorbirii și pentru terapie.

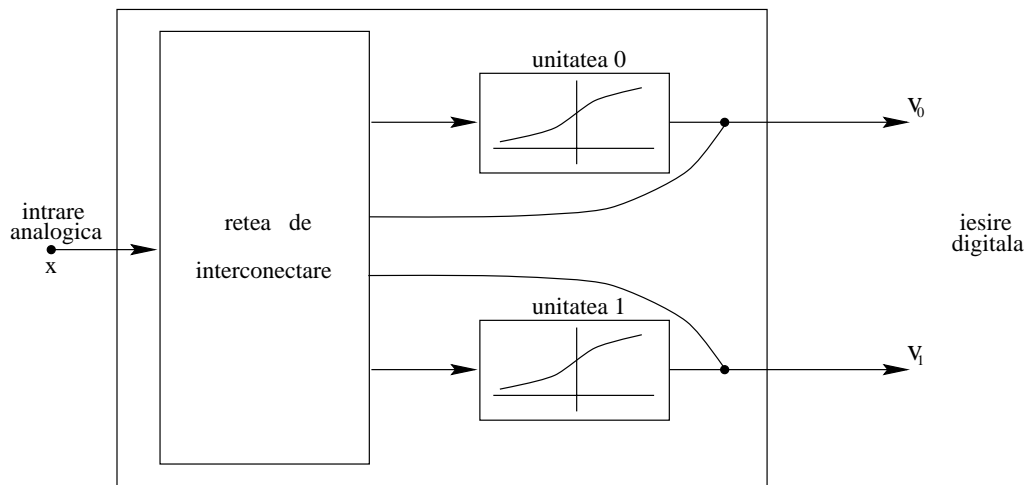


Figura 1.7: Diagramă bloc a unui convertor A/D pe doi biți.

1.2 Istoricul dezvoltării rețelelor neurale

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

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

Primele neurocalculatoare au fost construite în anii '50 (Minsky, 1954). Ele își adaptau automat conexiunile. În 1958, Frank Rosenblatt a inventat un element neural numit *perceptron*. Era conceput ca o mașină instruibilă capabilă să învețe 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 învățare Windrow-Hoff (Bernard Windrow, Marcian Hoff). Regula minimizează eroarea pătratică însumată pe parcursul instruirii. Aplicațiile erau de recunoașterea formelor, control adaptiv și previziunea vremii.

În ciuda entuziasmului anilor '60, mașinile existente nu permiteau abordarea unor probleme complexe. Pe de altă parte, nici schemele de învățare 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 și Papert¹, care a demonstrat limitele rețelelor bazate pe perceptroni. În acest timp, majoritatea cercetătorilor se îndreptau spre alte domenii. Domeniul rețelelor neurale (care făcea la acea vreme parte din cibernetică) părea închis. În schimb se dezvolta

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

promițător domeniul inteligenței artificiale, preluând și sarcini pe care rețelele neurale nu puteau să le rezolve la acel stadiu.

În perioada 1965-1984, câțiva cercetători au reușit să dezvolte totuși cercetările. Kunihiro Fukushima a definit o clasă de rețele neurale numite *neocognitroni* (1980). Neocognitronul modelează recunoașterea vizuală a formelor prin emularea imaginilor de pe retină și procesarea lor folosind neuroni ierarhizați pe două nivele.

Cercetările în domeniul memoriilor asociative au evoluat în Finlanda (Teuvo Kohonen) și SUA (James Anderson). Stephen Grossberg și Gail Carpenter au introdus câteva arhitecturi de rețele neurale și au dezvoltat teoria rețelelor adaptive prin rezonanță, ART.

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

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

1.3 Viitorul

Datorită denumirii, domeniul rețelelor neurale este supus unei supraestimări populiste. Este tentant pentru om să-și imagineze o mașină care să fie asemeni lui. Terminologia antropomorfă trebuie privită cu multă rețineră.

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

Sunt însă domenii întregi în care rețelele neurale devin mai avantajoase. Cele mai interesante aplicații sunt cele care presupun inferența de tip uman și perceperea vorbirii și a imaginilor. Aceste aplicații nu pot fi decât parțial rezolvate pe calculatoare clasice.

Este de așteptat ca rețelele neurale să fie aplicate în procesarea semnalelor și sisteme expert. Rețelele neurale nu vor înlocui aplicațiile de inteligență artificială de pe calculatoarele clasice, ci vor oferi o tehnologie complementară.

Neurocalculatoarele actuale sunt de multe ori calculatoare convenționale care execută software de simulare a rețelelor neurale. Alte neurocalculatoare folosesc deja componente (plăci, chip-uri) dedicate. Cele mai interesante sunt, desigur, chip-urile VLSI care implementează rețele 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 posibilități de a defini rețelele neurale. La o extremă, rețelele neurale sunt o clasă de algoritmi matematici, deoarece o rețea poate fi privită în esență ca o notație grafică pentru o clasă largă de algoritmi. La cealaltă extremă, rețelele neurale emulează rețelele neurale biologice din organismele vii. În lumina cunoștințelor limitate pe care le avem în prezent asupra rețelelor neurale biologice, cea mai plauzibilă definiție se apropie mai mult de cea algoritmică.

Rețelele neurale sunt în mod cert inspirate din biologie, dar există mari diferențe între rețelele neurale artificiale și cele naturale. Nu există încă modele care să concureze cu succes performanțele creierului uman. De fapt și cunoștințele noastre despre funcționarea creierului sunt extrem de limitate. Creierul rămâne mai curând o metaforă pentru rețelele neurale dezvoltate până acum.

Cu toate că analogia dintre rețelele neurale artificiale și cele naturale este vagă, vom începe totuși prin a menționa modelul neuronului biologic. Vom defini apoi neuronul artificial și rețelele neurale artificiale elementare. În fine, vom discuta formele de bază ale procesării în rețele neurale artificiale, cu un accent deosebit pe procesele de învățare.

2.1 Neuronii biologici și modelele lor artificiale

Creierul uman constă din aproximativ 10^{11} neuroni. Ei comunică printr-o rețea de conexiuni formate din axoni și 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 rețea densă de conexiuni electrice condiționate în mare măsură de procese biochimice. Rețeaua neurală are o structură elaborată, cu interconexiuni foarte complexe. Intrarea în rețea este asigurată de către receptorii senzoriali. Receptorii furnizează stimuli atât din partea corpului cât și din partea organelor senzoriale care preiau stimuli lumii exterioare. Stimulii au forma impulsurilor electrice care conduc informația în rețeaua de neuroni. Ca rezultat al procesării informației în sistemul nervos central, efectorii sunt controlați și dau răspunsuri sub forma diferitelor

acțiuni. Avem deci un sistem constând din receptori, rețeaua neurală și efectori, care controlează organismul și acțiunile sale. Fluxul informațional este descris în figura 2.1.

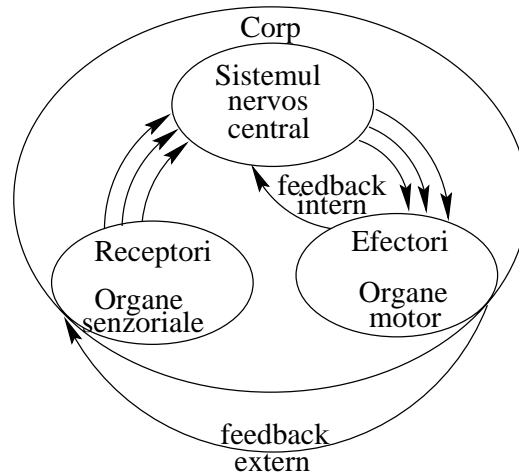


Figura 2.1: Fluxul informațional în sistemul nervos.

Neuronul biologic

Neuronul este celula nervoasă și are trei componente:

- *soma* - corpul celulei
- *axonul* - fibră lungă care servește ca linie de comunicație
- *dendritele*

Dendritele formează un arbore de fibre fine în jurul corpului neuronului. Dendritele recepționează informația de la alți neuroni prin axonii acestora. Axonul este o conexiune cilindrică lungă care în partea finală devine arborescentă. Fiecare ramură are o terminație care aproape atinge dendritele neuronilor vecini. *Sinapsa* este interfața prin care neuronul își 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 transmitători chimici la sinapse. Astfel, terminația axonului generează substanța 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 latență*. Răspunsul neuronului este generat dacă totalul potențialului membranei sale atinge un anumit nivel. Membrana poate fi considerată ca o învelitoare care agregă magnitudinea semnalelor

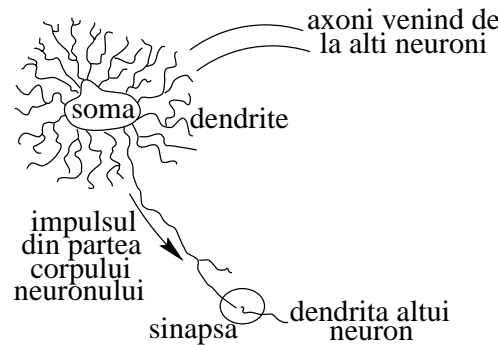


Figura 2.2: Modelul neuronului biologic.

care intră pe parcursul unei anumite durate de timp. Neuronul generează un impuls-răspuns și îl transmite axonului său numai dacă condițiile 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 condiție mai precisă pentru ca neuronul să genereze un impuls este ca excitația să depășească inhibiția cu o valoare de aproximativ 40 mV, numită *pragul neuronului*. Deoarece o conexiune sinaptică produce reacția de excitație sau de inhibiție a neuronului receptor, este practic să atașăm ponderile ± 1 acestor conexiuni. **Neuronul generează un impuls atunci când suma ponderilor impulsurilor receptate depășește valoarea pragului pe parcursul perioadei de însumare latentă.**

Procesarea în rețelele neurale biologice este complexă și mai puțin 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, informația transmisă între celulele nervoase este sub forma semnalelor binare.

După transmiterea unui impuls, axonul rămâne pentru un timp într-o stare 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 condițiile de excitație de la momentul k .

Neuronul va fi excitat la un anumit moment dat, dacă numărul sinapselor excitate excitatoare depășește numărul sinapselor excitate inhibitoare la momentul precedent cu cel puțin 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 și de modul în care sunt ei conectați. Avem deci o rețea densă de neuroni interconectați care generează semnale asincrone. Semnalele sunt transmise apoi către neuronii vecini dar sunt

și retransmise (feedback) neuronilor generatori.

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

Modelul neural McCulloch-Pitts

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

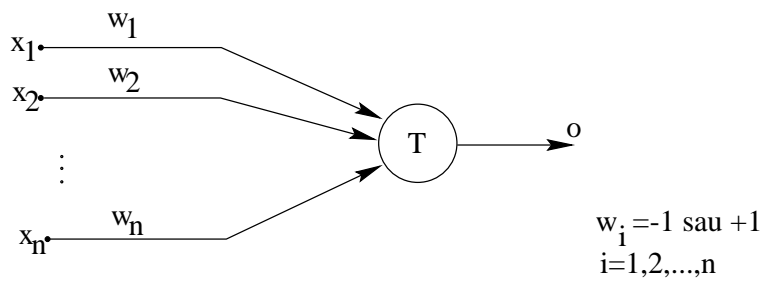


Figura 2.3: Neuronul McCulloch-Pitts.

Intrările $x_i^k, i = 1, 2 \dots n$ sunt 0 sau 1, în funcție de absența sau prezența impulsului la momentul k (fig 2.3). Semnalul de ieșire al neuronului este o . Regula după care neuronul generează un semnal este:

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

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

Cu toate că este foarte simplu, acest model are un remarcabil potențial computațional. Poate realiza operațiile logice NOT, OR și AND. După cum știm, orice funcție logică de mai multe variabile poate fi implementată utilizând sau NOT și OR, sau NOT și AND. De exemplu, funcțiile NOR și NAND pot fi implementate prin rețele de neuroni conform modelelor din figura 2.4.

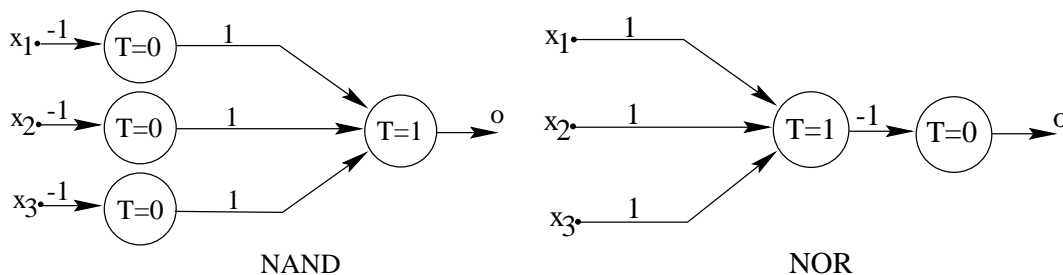


Figura 2.4: Funcțiile NOR și NAND.

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

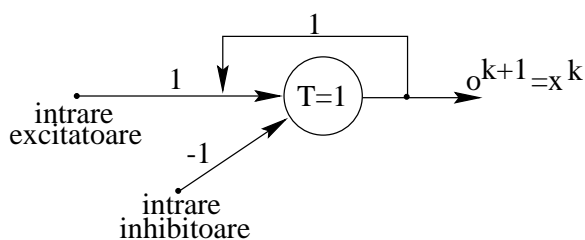


Figura 2.5: Celula de memorie.

După ce s-a inițializat celula, astfel încât să genereze sau să nu genereze un semnal, această valoare de ieșire este susținută indefinit, în absență unor intrări.

Hardware-ul unui calculator digital de orice complexitate poate fi obținut prin utilizarea unei rețele neurale constituite din blocuri elementare pentru operații logice și pentru memorie. Ne interesează, însă altceva: care este puterea de calcul a rețelelor neurale ținând cont de capacitatea lor de a învăța. Vom reveni asupra acestor aspecte mai târziu.

Modelarea unui neuron general

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

Fiecare model neural constă dintr-un element de procesare cu conexiuni sinaptice de intrare și cu o singură ieșire. Intrările și ieșirile sunt unidirecționale. Definim modelul unui neuron general ca în figura 2.6

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

sau

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

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

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

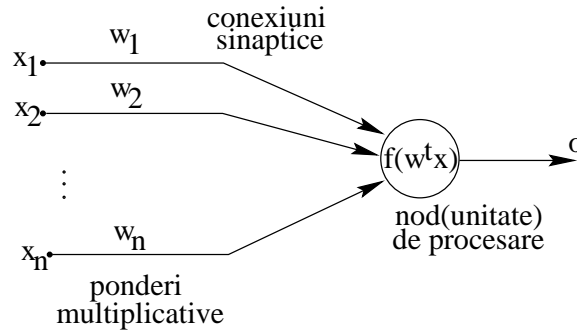


Figura 2.6: Modelul unui neuron general.

iar \mathbf{x} este vectorul de intrări:

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

Toți vectorii din acest curs sunt vectori coloană, iar indicele t notează o transpunere. Funcția $f(\mathbf{w}^t \mathbf{x})$ este *funcția de activare*. Domeniul ei de definiție este mulțimea valorilor de activare, *net*, a unui model neural:

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

De aceea, folosim și notația $f(net)$. Variabila *net* este analogul potențialului membranei neuronului biologic.

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

Funcții 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 $\lambda \rightarrow \infty$, funcția bipolară continuă devine bipolar binară. Funcțiile 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 $\lambda \rightarrow \infty$, funcțiile continue devin funcții discrete. Cele două funcții continue se numesc și *caracteristici sigmoidale*.

Majoritatea neuronilor utilizează funcții de activare bipolare. Desigur, funcțiile de activare pot fi definite și altfel.

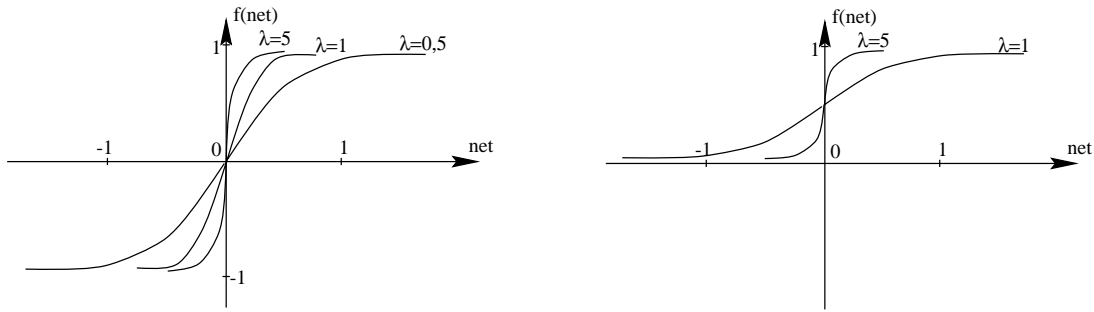


Figura 2.7: Funcții de activare continue bipolară și unipolară.

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

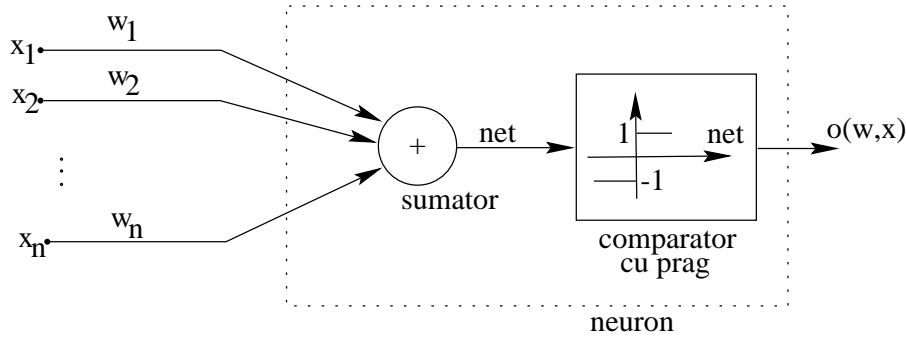


Figura 2.8: Perceptron binar.

Perceptronul discret a fost introdus de Rosenblatt (1958) și a fost prima mașină instruibilă.

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

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

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

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

sau

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

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

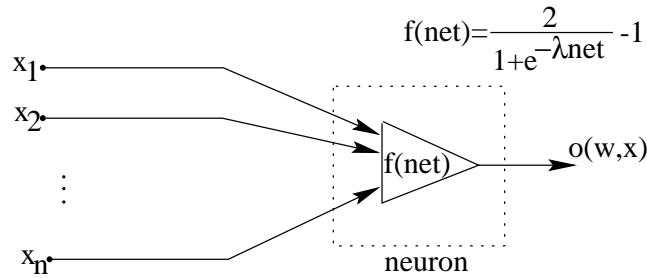


Figura 2.9: Perceptron continuu.

Pentru cazul discret, avem:

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

sau

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

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

2.2 Modelarea rețelelor neurale

Cunoscând acum definiția modelului unui neuron general, putem defini o rețea neurală ca o *interconectare de neuroni astfel încât ieșirea fiecărui neuron este conectată, via ponderi, cu toți neuronii, inclusiv cu neuronul respectiv.*

Rețea feedforward

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

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

și

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

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

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

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

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

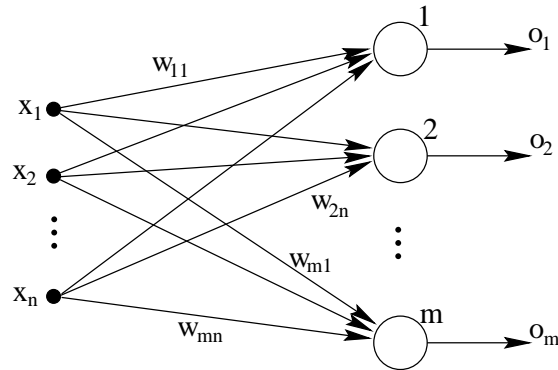


Figura 2.10: Rețea neurală feedforward.

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

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

Funcțiile $f(\cdot)$ sunt funcții neliniare de activare.

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

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

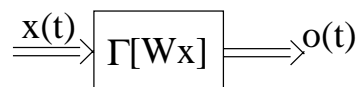


Figura 2.11: Diagrama bloc a unei rețele neurale feedforward.

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

Rețea feedback

O rețea feedback se poate obține dintr-o rețea feedforward prin conectarea ieșirilor neuronilor cu propriile intrări (fig. 2.12).

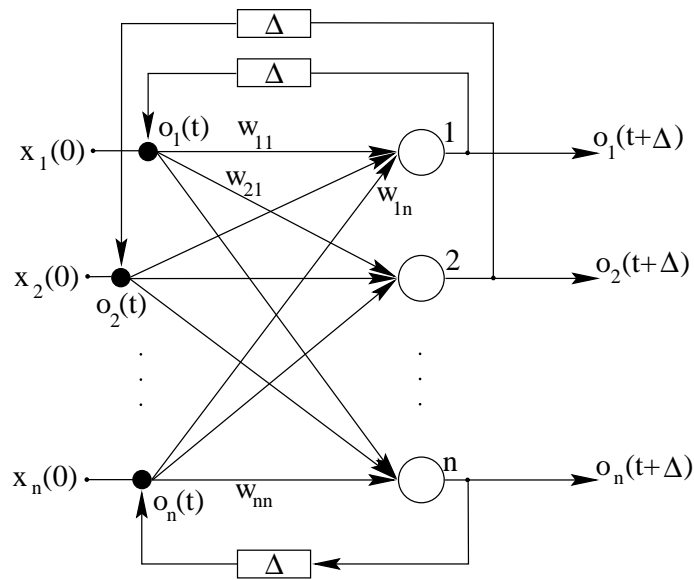


Figura 2.12: Rețea neurală feedback.

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

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

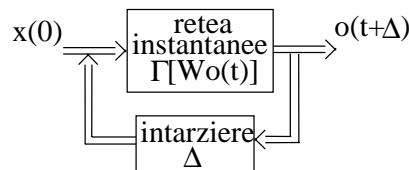


Figura 2.13: Diagrama bloc a rețelei neurale feedback.

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

Considerând timpul ca o variabilă discretă și decidem să observăm funcționarea rețelei la momentele $\Delta, 2\Delta, 3\Delta, \dots$, sistemul este cu *timp discret*. Convențional, putem considera că pasul timpului este unitar. Atunci, notăm:

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

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

$$\begin{aligned} \mathbf{o}^1 &= \Gamma[\mathbf{W}\mathbf{x}^0] \\ \mathbf{o}^2 &= \Gamma[\mathbf{W}\Gamma[\mathbf{W}\mathbf{x}^0]] \\ &\dots \\ \mathbf{o}^{k+1} &= \Gamma[\mathbf{W}\Gamma[\dots \Gamma[\mathbf{W}\mathbf{x}^0] \dots]]. \end{aligned}$$

Rețelele recurente operează de obicei cu o reprezentare discretă a datelor și folosesc neuroni cu o funcție de activare discretă. Un sistem având intrări cu timp discret și o reprezentare discretă a datelor se numește un *automat*. Deci, rețelele neurale recurente din această categorie pot fi considerate niște automate.

Numim $\mathbf{o}^1, \mathbf{o}^2, \dots$ *stări* ale rețelei la momentele 1, 2, ... și ecuațiile de mai sus oglindesc secvența tranzițiilor stărilor. O stare de echilibru se numește și *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. Secvența de stări ale unei rețele recurente este în general nedeterministă.

O rețea cu *timp continuu* se obține înlocuind elementele de întârziere discrete cu elemente continue. Fie, de exemplu, rețeaua feedback cu timp continuu din figura 2.14.

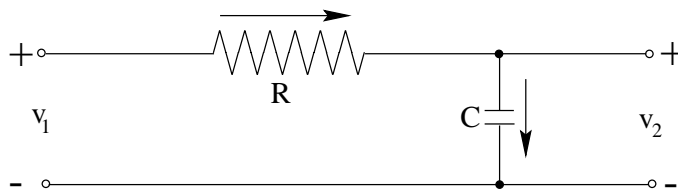


Figura 2.14: Rețea feedback cu timp continuu.

Ea este o rețea electrică constând dintr-o rezistență și un condensator, unde v_1 este tensiunea la intrare, iar v_2 este tensiunea la ieșire. De fapt, rețelele electrice sunt utilizate frecvent pentru a modela calculele efectuate de o rețea neurală. Rețelele electrice posedă flexibilitatea de a modela toate fenomenele liniare și neliniare întâlnite în acest curs. Din această cauză, rețelele electrice reprezintă modele fizice funcționale ale rețelelor neurale. Din legea lui Kirchoff obținem:

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

$$\frac{dv_2}{dt} + \frac{v_2}{RC} = \frac{v_1}{RC}. \quad (2.1)$$

De aici, obținem:

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

care reprezintă modificarea tensiunii v_2 în intervalul Δt .

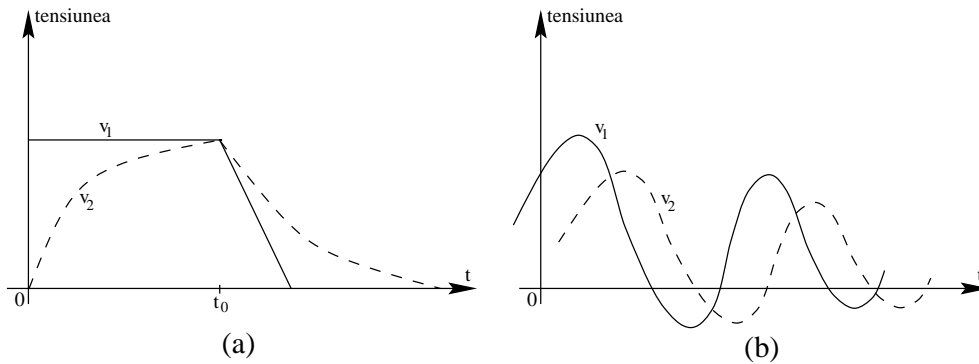


Figura 2.15: Răspunsul în timp (a) la un impuls și (b) la o tensiune de intrare de tip undă armonică al rețelei cu timp continuu din figura anterioară.

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

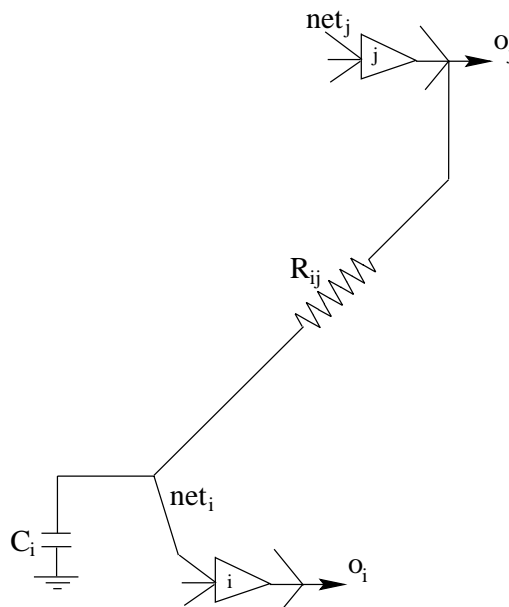


Figura 2.16: Conexiune sinaptică.

Rezistența R_{ij} servește ca pondere de la ieșirea neuronului j către intrarea neuronului i . Ecuația 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 rețea de mai mulți neuroni de acest tip are, din punct de vedere dinamic, o comportare complexă.

2.3 Învățare și adaptare

În general, învățarea este schimbarea comportamentului datorită experienței. La om și animale procesul de învățare nu poate fi observat direct, ci presupunem că a avut loc observând modificările de comportament. La rețelele neurale, procesul de învățare este mai direct: putem observa fiecare pas al învățării ca o relație distinctă de tip cauză-efect.

Un cadru general pentru procesul de învățare la rețele neurale este dat de teoria aproximării.

Învățarea ca aproximare

Teoria aproximării se referă la aproximarea unei funcții continue de mai multe variabile $h(\mathbf{x})$ printr-o altă funcție $H(\mathbf{w}, \mathbf{x})$, unde $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^t$ este vectorul de intrare și $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_m]^t$ este vectorul parametrilor (ponderilor). Scopul învățării este să se găsească \mathbf{w} pentru care se obține cea mai bună aproximare a lui $h(\mathbf{x})$, având la dispoziție o mulțime de exemple, $\{\mathbf{x}\}$, folosite pentru învățare. O problemă importantă este alegerea funcției $H(\mathbf{w}, \mathbf{x})$, aceasta numindu-se *problema reprezentării*. După alegerea lui $H(\mathbf{w}, \mathbf{x})$, se aplică algoritmul de învățare al rețelei 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 distanță).

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

Învățarea supervizată și învățarea nesupervizată

Majoritatea rețelelor neurale pe care le discutăm învață incremental, pas cu pas. Dacă ponderile rețelei sunt ajustate printr-un singur pas, atunci informația feedback produsă de rețea nu mai este necesară și vorbim de o învățare de tip *batch*, adică "la grămadă".

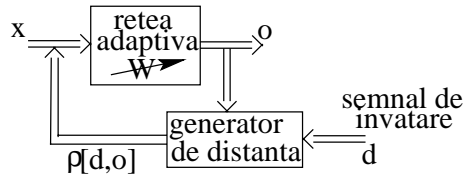


Figura 2.17: Învățare supervizată.

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

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

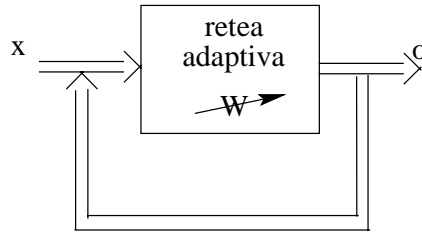


Figura 2.18: Învățare nesupervizată.

Învățarea nesupervizată (fig. 2.18) se folosește de multe ori, de exemplu, la *clustering* (fig. 2.19), atunci când informația *a priori* este minimă.

2.4 Reguli de învățare

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 învățare: Vectorul $\mathbf{w}_i = [w_{i1} \ w_{i2} \ \dots \ w_{in}]^t$ crește proporțional cu produsul intrării \mathbf{x} și a semnalului de învățare r . Semnalul r este în general o funcție de \mathbf{w}_i , \mathbf{x} și, dacă 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 *constantă de învățare*, pozitivă, care determină rata învățării. Atunci:

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

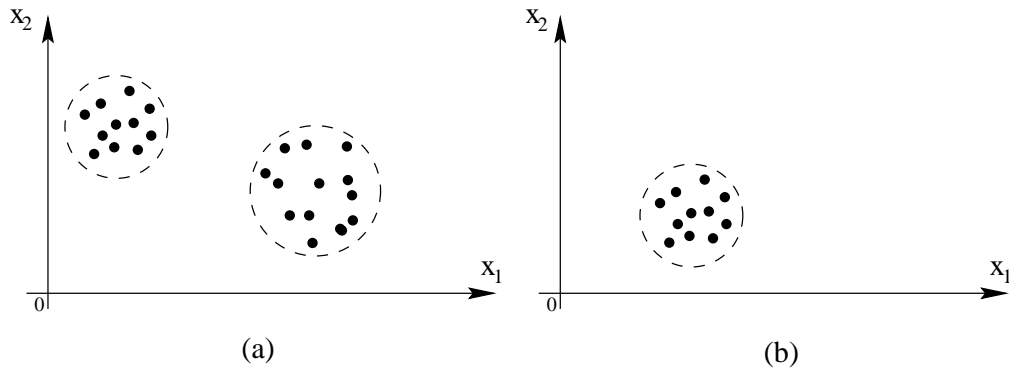


Figura 2.19: (a) Pattern-uri bidimensionale care formează două clustere. (b) Aparent, nu se disting mai multe clustere și nu știm câte sunt. În acest caz învățarea nesupervizată nu este indicată.

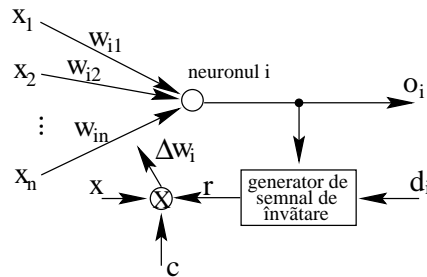


Figura 2.20: Modelul instruirii în cazul învățării supervizate.

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 învățarea cu timp discret (pas cu pas). Se presupune că ponderile sunt inițializate convenabil, înainte ca procesul de învățare să înceapă.

Regula de învățare a lui Hebb (1949)

Această regulă particularizează semnalul de învățare astfel:

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

adică semnalul de învățare este chiar ieșirea neuronului. Atunci:

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

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

Este necesar ca, inițial, ponderile să aibă valori aleatoare mici. Învățarea după regula lui Hebb este de tip feedforward și 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 creștere (schimbare metabolică) într-una sau în ambele celule, astfel încât eficiența excitării lui B de către A crește.”

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

Regula de învățare a perceptronului (Rosenblatt, 1958)

Semnalul de învățare pentru această regulă este:

$$r = d_i - o_i$$

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

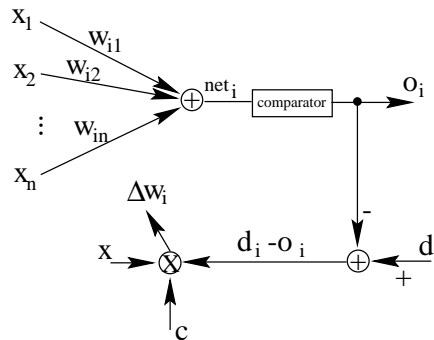


Figura 2.21: Regula de învățare a perceptronului.

Avem:

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

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

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

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

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

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

Regula de învățare a perceptronului este importantă pentru învățarea supervizată a rețelelor neurale. Ponderile sunt inițializate cu orice valoare.

Regula de învățare delta (McClelland, Rumelhart, 1986)

Este valabilă pentru funcții de activare continue și pentru învățarea supervizată. Semnalul de învățare pentru această regulă (fig. 2.22) este numit *delta* și este dat de

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

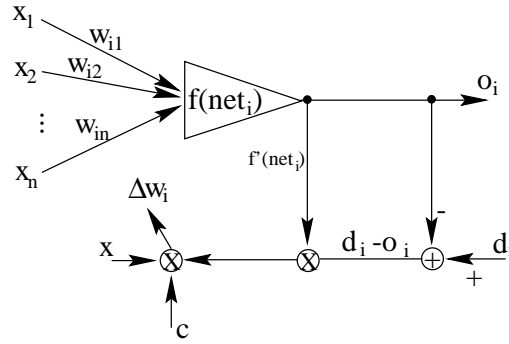


Figura 2.22: Regula de învățare delta.

Această regulă se deduce din condiția de cea mai mică eroare pătratică dintre o_i și 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:

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

și reprezintă panta pe direcția w_{ij} . Alegem termenul Δw_{ij} astfel încât să fie proporțional cu panta negativă (deci să minimizăm pe E) și obținem:

$$\begin{aligned} \Delta w_{ij} &= \eta (d_i - o_i) f'(\text{net}_i) x_j, \quad j = 1, 2, \dots, n \\ \Delta \mathbf{w}_i &= \eta (d_i - o_i) f'(\text{net}_i) \mathbf{x} \end{aligned} \quad (2.2)$$

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

unde η este o constantă pozitivă.

Presupunând că semnalul de învățare este

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

obținem:

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

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

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

Regula de învățare Windrow-Hoff (1962)

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

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

Atunci,

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

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

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 funcția de activare este funcția identică $f(net) = net$ și obținem $f'(net) = 1$.

Ponderile pot fi inițializate oricum.

Regula corelației

Substituind $r = d_i$ în regula generală de învățare obținem regula corelației:

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

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

De obicei, regula corelației se aplică în rețele de memorie cu neuroni cu funcție de activare binară. Se poate interpreta ca un caz particular al regulii lui Hebb în care funcția de activare este binară și $o_i = d_i$. Totuși, învățarea hebbiană este nesupervizată, în timp ce regula corelației se aplică la învățarea supervizată. Ca și în cazul învățării hebbiene, este necesar ca ponderile să fie inițializate cu valori apropiate de 0.

Regula ”câștigătorul ia tot”

Această regulă diferă mult de regulile prezentate până acum. Poate fi explicată pe o mulțime de neuroni și este un exemplu de învățare competitivă folosită în instruirea nesupervizată. În general, această regulă se folosește pentru învățarea proprietăților statistice ale intrarilor.

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

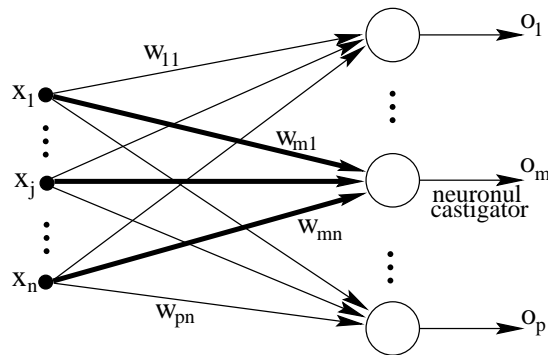


Figura 2.23: Regula de învățare câștigătorul ia tot.

Ca rezultat,

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

este singurul vector ajustat:

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

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

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

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

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

Ponderile se inițializează cu valori aleatoare și sunt normalizate. În general, α scade pe parcursul procesului de învățare.

Paradigme ale învățării în rețele neurale

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

Tabelul 2.1: Comparație între calculul convențional și calculul neural din punct de vedere al etapelor care trebuie parcurse.

Sarcina	Calcul convențional	Calcul neural
Rezolvarea unei probleme	Formularea algoritmului	Selectarea unei arhitecturi și definirea mulțimii de exemple reprezentative
Achiziționarea cunoștințelor	Programare	Instruire
Calcul	Aritmetic, precizie mare	Precizie mică, aplicații neliniare
Memorarea datelor	ROM, RAM - memorii binare	Ponderi ale conexiunilor având în general valori continue

2. *Heteroasociere*. Este o variantă a autoasocierii. O mulțime de perechi de pattern-uri este prezentată în mod repetat. Paradigmele autoasocierii și heteroasocierii sunt tipice memoriilor asociative.
3. *Clasificare*. Paradigma clasificării poate fi considerată o variantă a autoasocierii. În acest caz avem o mulțime fixată de categorii în care trebuie clasificate pattern-urile de intrare. În perioada instruirii, sistemului i se prezintă pattern-uri din secvența de instruire precizând categoriile din care fac parte. Scopul este de a învăța corect clasificarea, astfel încât, ulterior, orice pattern de intrare (eventual perturbat față de pattern-urile din secvența de instruire) să fie clasificat corect. Aceasta este paradigma tipică pentru algoritmul perceptronului și instruirea prin propagarea în urmă a erorii în rețelele feedforward multistrat. Instruirea este supervizată. Instruirea este mai bună dacă în secvența de instruire apar și pattern-uri perturbate (spre deosebire de paradigma autoasocierii).
4. *Detectare de regularități (clustering)*. Sistemul trebuie să descopere caracteristici (trăsături) comune într-o mulțime de pattern-uri. Nu există categorii *a priori* de pattern-uri. Sistemul trebuie să-și 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 și α sunt pozitive.

Regula de învățare	Ajustarea Δw_{ij}	Ponderile inițiale	Tipul învățării	Caracteristicile neuronilor	Neuron sau strat
Hebb	$co_i x_j \quad j = 1, 2, \dots, n$	0	nesuperv.	oricare	neuron
perceptron	$c[d_i - \text{sgn}(\mathbf{w}_i^t \mathbf{x})]x_j \quad j = 1, 2, \dots, n$	oricare	superv.	binar	neuron
delta	$c(d_i - o_i)f'(net_i)x_j \quad j = 1, 2, \dots, n$	oricare	superv.	continuu	neuron
Windrow-Hoff	$c(d_i - \mathbf{w}_i^t \mathbf{x})x_j \quad j = 1, 2, \dots, n$	oricare	superv.	oricare	neuron
corelație	$cd_i x_j \quad j = 1, 2, \dots, n$	0	superv.	oricare	neuron
căștiătorul ia tot	$\Delta w_{mj} = \alpha(x_j - w_{mj}) \quad j = 1, 2, \dots, n$	aleatoare normalizate	nesuperv.	continuu	strat de neuroni

2.5 Exemple

1. Exemplu de rețea feedforward pe două straturi cu funcție de activare bipolară

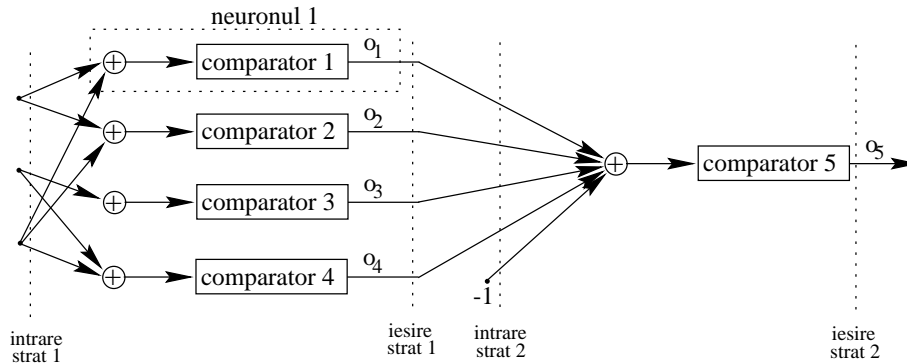


Figura 2.24: Rețea feedforward pe două straturi.

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

Pentru stratul 1:

$$\mathbf{o} = \Gamma [o_1 \ o_2 \ o_3 \ o_4]^t$$

$$\mathbf{x} = [x_1 \ x_2 \ -1]^t$$

$$W_1 = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & -1 & -3 \end{bmatrix}$$

Pentru stratul 2:

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

$$\mathbf{x} = [o_1 \ o_2 \ o_3 \ o_4 \ -1]^t$$

$$W_2 = [1 \ 1 \ 1 \ 1 \ 3,5]^t$$

Răspunsurile primului strat se calculează astfel:

$$\mathbf{o} = [\text{sgn}(x_1 - 1) \ \text{sgn}(-x_1 + 2) \ \text{sgn}(x_2) \ \text{sgn}(-x_2 + 3)]^t$$

Răspunsul pentru al doilea strat este:

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

Observăm că $o_5 = +1$ dacă și numai dacă $o_1 = o_2 = o_3 = o_4 = 1$. Rezultă că această rețea este o aplicație a spațiului bidimensional x_1x_2 în punctele ± 1 .

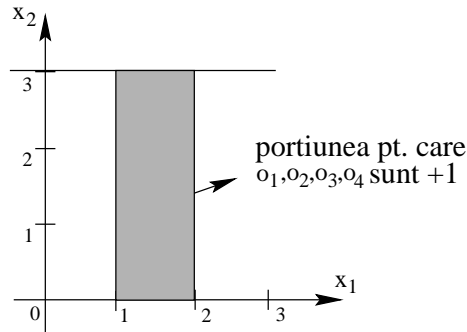


Figura 2.25: Reprezentare în spațiul bidimensional.

Să presupunem acum că avem aceeași arhitectură, dar că neuronii au caracteristici sigmoïdale. Obținem:

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

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

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

2. Exemplu de rețea recurentă

Stările rețelei sunt vârfurile unui cub 4-dimensional: $\{-1, 1\}^4$. Trecerea dintr-o stare în 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 $\mathbf{x}^0 = [1 \ 1 \ 1 \ -1]^t$, obținem

$$\mathbf{o}^1 = [\text{sgn}(3) \ \text{sgn}(3) \ \text{sgn}(3) \ \text{sgn}(-3)]^t = [1 \ 1 \ 1 \ -1]^t$$

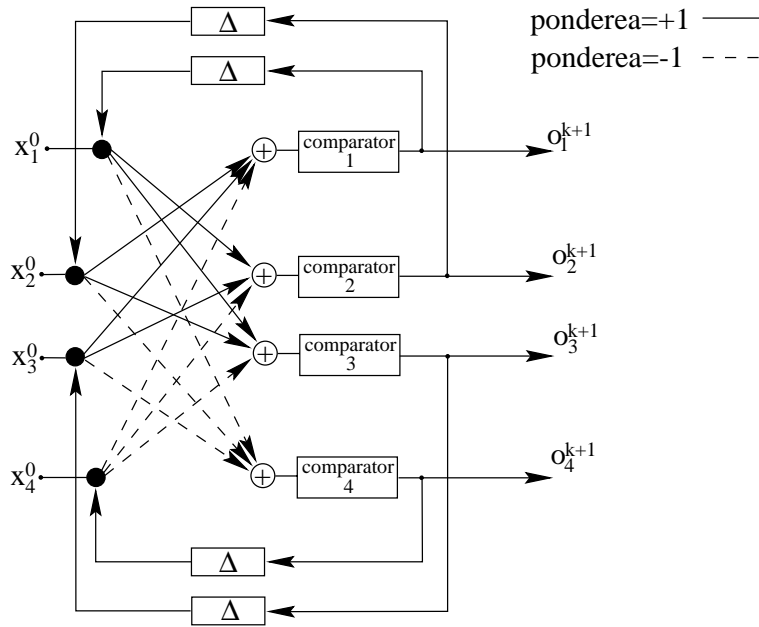


Figura 2.26: Rețea neurală recurentă.

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

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

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

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

Verificați singuri ce se întâmplă 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 rețea feedback cu timp continuu

Fie rețeaua din figura 2.27 unde $o_i = f(\text{net}_i)$, $i = 1, 2$, iar f este funcția de activare continuă bipolară. În această figură, R_{12} și R_{21} sunt ponderi negative, întârzierile sunt realizate de dispozitivul RC , iar rezistențele R modelează curentul absorbit de neuroni.

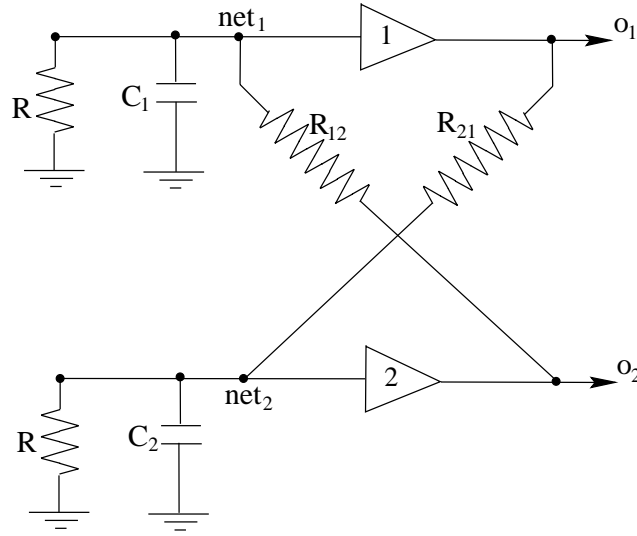


Figura 2.27: Rețea electrică a doi neuroni.

Obținem:

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

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 rețea. Presupunem că circuit a fost inițializat prin încărcarea condensatoarelor (cu \mathbf{o}_1^0 , \mathbf{o}_2^0 , nu neapărat egale). Rețeaua își caută unul din cele două puncte de echilibru (fig. 2.28).

Simularea rețelei se face pe calculator prin ecuațiile date.

4. Exemplu de învățare hebbiană

În figura 2.29, avem intrarea:

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

și ponderile inițiale:

$$\mathbf{w}^1 = [1 \quad -1 \quad 0 \quad 0,5]^t.$$

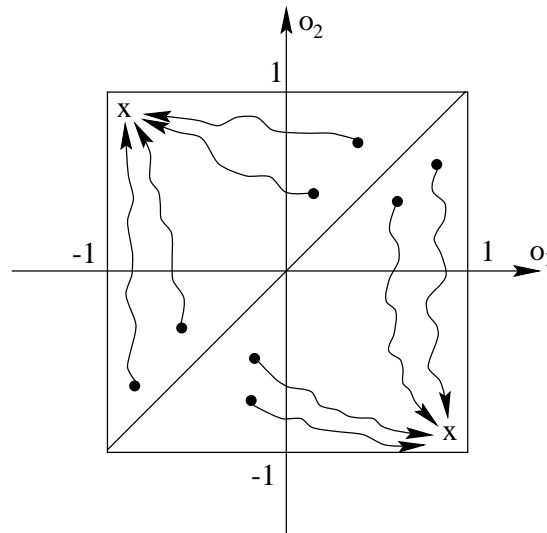


Figura 2.28: Modul în care rețeaua își găsește cele două puncte de echilibru.

Presupunem că această rețea este instruită folosind mulțimea 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: $f(\text{net}) = \text{sgn}(\text{net})$.

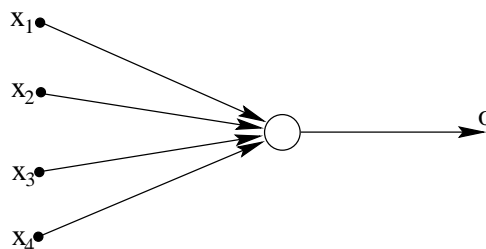


Figura 2.29: Rețea neurală pentru învățarea hebbiană.

Pasul 1. Se aplică \mathbf{x}_1 .

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

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

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

Pasul 2. Se aplică \mathbf{x}_2 .

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

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

Pasul 3. Se aplică \mathbf{x}_3 .

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

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

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

Pasul 1. Se aplică \mathbf{x}^1 .

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

Pasul 2. Se aplică \mathbf{x}^2 .

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

Pasul 3. Se aplică \mathbf{x}^3 .

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

Se observă că pentru funcția de activare continuă se obțin în general rezultate în aceeași direcție, valorile fiind însă mai diferențiate.

5. Exemplu de învățare prin regula perceptronului

Considerăm următorii vectori de intrare:

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

și $c = 0,1$. Răspunsurile dorite de către supervisor pentru \mathbf{x}_1 , \mathbf{x}_2 , \mathbf{x}_3 sunt -1 , -1 și, 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 corecție 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 corecție.

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 corecție:

$$\mathbf{w}^4 = \mathbf{w}^3 + 0,1(1 + 1)\mathbf{x}_3 = \begin{bmatrix} 0,6 \\ -0,4 \\ 0,1 \\ 0,5 \end{bmatrix}.$$

Nu este o simplă coincidență faptul că ultima componentă a vectorilor \mathbf{x}_1 , \mathbf{x}_2 , \mathbf{x}_3 este invariabilă. Învățarea 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 rețeaua a "învățat". 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 , obținem $net^4 = 0,9$, 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 $w_1 + w_2 > 0$ și $w_1 + w_2 < 0$, 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 $w_1 + w_2 > w_3$ și $w_1 + w_2 < -w_3$. Acest sistem are soluții, evident, de exemplu $w_1 = 1$, $w_2 = 1$, $w_3 = -3$. Ce am făcut de fapt? Deoarece este evident că \mathbf{x}_1 și \mathbf{x}_2 nu pot fi învățați în mod supervizat, am considerat vectorii extinși \mathbf{y}_1 și \mathbf{y}_2 care se obțin din \mathbf{x}_1 și \mathbf{x}_2 prin adăugarea unei componente constante, -1. Am văzut că pentru \mathbf{y}_1 și \mathbf{y}_2 există \mathbf{w}_1 și \mathbf{w}_2 astfel încât perceptronul să învețe corect. Puteți acum folosi regula perceptronului pentru a învăța \mathbf{y}_1 și \mathbf{y}_2 . Încercați să explicați din punct de vedere geometric care este diferența dintre a învăța \mathbf{x}_1 și \mathbf{x}_2 față de a învăța \mathbf{y}_1 și \mathbf{y}_2 .

6. Exemplu de învățare prin regula delta

Folosim aceiași 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 funcția 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 Exerciții

- Folosiți neuronul McCulloch-Pitts pentru a elabora rețele care implementează următoarele funcții logice:
 - $o^{k+1} = x_1^k o_2^k o_3^k$, unde x_3^k este complementul lui x_3^k . Se va folosi un neuron.
 - $o^{k+2} = x_1^k x_2^k x_3^k$. Se vor folosi doi neuroni în cascadă.
 - $o^{k+2} = x_1^k x_2^k$. Se vor folosi doi neuroni în cascadă.
- Arătați că rețelele neurale din figura 2.30 sunt echivalente (implementează aceeași funcție). Se presupune că:

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

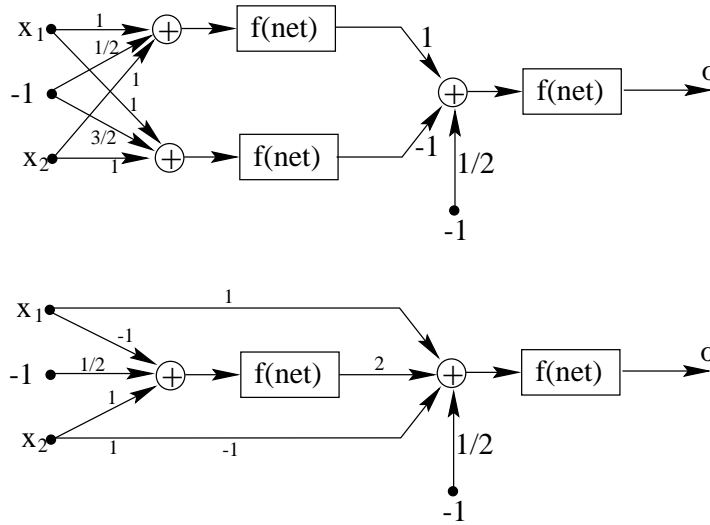


Figura 2.30: Exemplu de rețele neurale echivalente.

3. Rețeaua din figura 2.31 este un convertor A/D și poate fi utilizată pentru codificarea unei valori continue x într-un cod binar unipolar pe 4 biți ($o_3 o_2 o_1 o_0$). Analizați rețeaua și găsiți pentru ce valori ale lui x se obțin codificările $(0\ 0\ 0\ 0), \dots, (1\ 1\ 1\ 1)$. Presupuneți $-1 \leq x \leq 16$ și că

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

4. Rețeaua din figura 2.32 folosește neuroni cu funcție de activare bipolară continuă ($\lambda = 1$). S-a măsurat că $o_1 = 0,28$; $o_2 = -0,73$. Găsiți vectorul de intrare $\mathbf{x} = [x_1\ x_2]^t$ care a fost aplicat.
5. Rețeaua din figura 2.33 cu funcție de activare bipolară continuă este proiectată pentru a atribui vectorilor \mathbf{x}_1 , \mathbf{x}_2 , \mathbf{x}_3 clusterule 1 sau 2. Numărul cluster-ului este identic cu numărul neuronului care produce cel mai mare răspuns. Determinați din ce clusterule 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) Rețeaua din figura 2.34 folosește neuroni cu funcție de activare bipolară continuă ($\lambda = 5$) și implementează o aplicație de la planul $(x_1 x_2)$ la segmentul $|o_1| < 1$. Simulați funcționarea rețelei și tabulați funcția $o_1(x_1, x_2)$ pentru $|x_1| < 2,5$ și $|x_2| < 2,5$.
7. Învățare hebbiană pentru un neuron care este instruit cu:

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

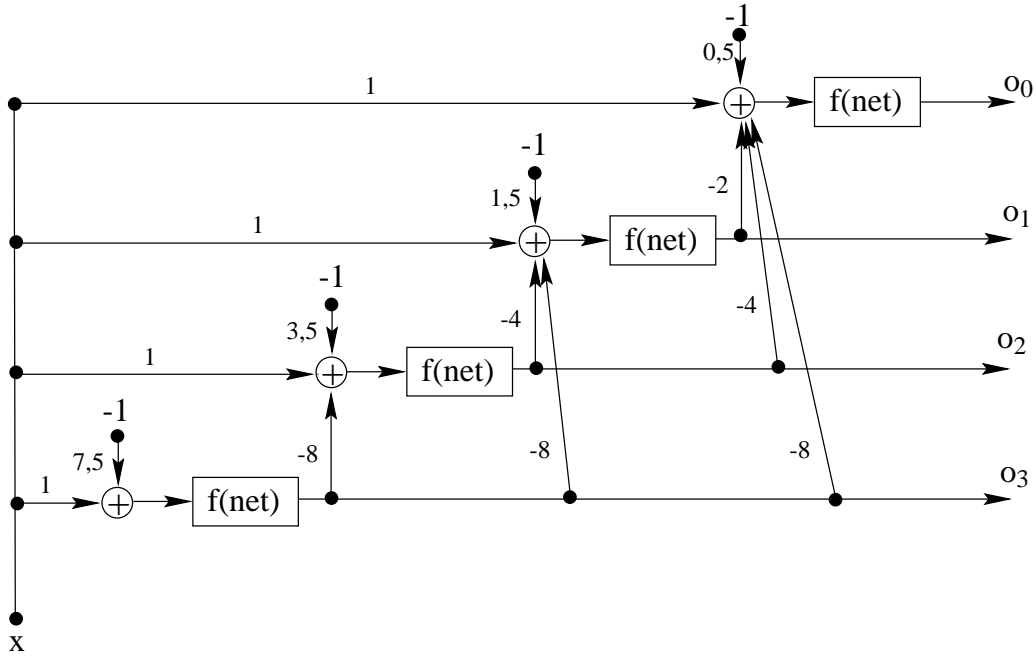


Figura 2.31: Convertor A/D.

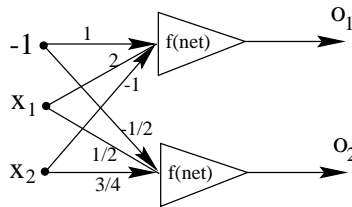


Figura 2.32: Rețea neurală care folosește neuroni cu funcție de activare bipolară continuă.

și $c = 1$. Găsiți ponderile finale pentru cazul când funcția de activare este

- bipolară binară
- bipolară continuă, $\lambda = 1$.

8. (C) Implementați învățarea unui neuron prin regula perceptronului folosind:

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

Repetati secvența de instruire $(\mathbf{x}_1, \mathbf{d}_1)$, $(\mathbf{x}_2, \mathbf{d}_2)$ până când se obțin răspunsurile corecte. Listați valorile net^k obținute.

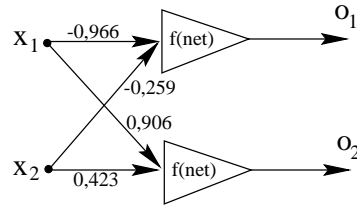


Figura 2.33: Rețea neurală care implementează un clasificator.

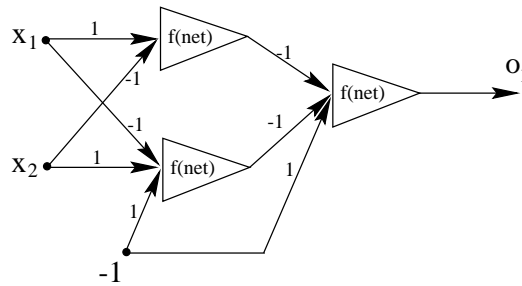


Figura 2.34: Rețea neurală care implementează o aplicație de la un plan la o dreaptă.

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

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

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

Repetăți secvența de instruire $(\mathbf{x}_1, \mathbf{d}_1)$, $(\mathbf{x}_2, \mathbf{d}_2)$ și studiați comportarea.

10. (C) Folosiți aceleași date ca și în problema precedentă, aplicând regula Windrow-Hoff.

Repetăți secvența de instruire.

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

- tipul funcției de activare
- λ , dacă este cazul
- mărimea rețelei
- vectorii de intrare pentru testarea rețelei

12. (C) Implementați algoritmi de învățare pentru un neuron cu cel mult șase intrări. Parametrii de intrare sunt:

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

Capitolul 3

Clasificatori pe bază de perceptroni monostrat

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

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

3.1 Clasificare

Una dintre cele mai importante categorii de probleme care poate fi efectuată de o rețea neurală este clasificarea pattern-urilor. Un *pattern* este descrierea cantitativă a unui obiect, eveniment sau fenomen. Clasificarea se poate referi atât la pattern-uri spațiale cât și temporale (ex.: imagini video de vapoare, hărți 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.

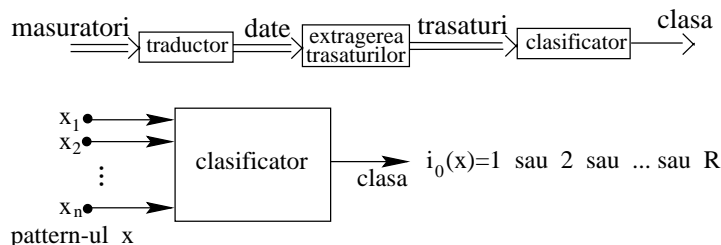
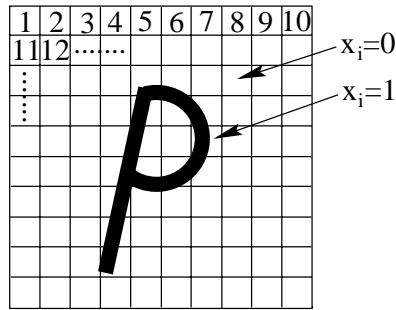


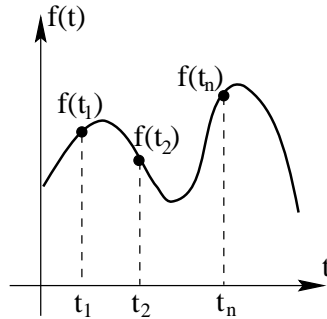
Figura 3.1: Sistem de clasificare.

Trăsăturile sunt date comprimate obținute prin reducerea dimensionalității.



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 recepționate de pe satelitul LANDSAT, sunt necesare doar câteva din componentele spectrale.

Rețelele neurale pot fi utilizate atât pentru clasificare cât și pentru extragerea trăsăturilor.

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

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

Clasificarea poate fi descrisă de multe ori în termeni geometrici. Orice pattern poate fi reprezentat printr-un punct în spațiul euclidian n -dimensional E^n numit spațiul pattern-urilor. Un clasificator aplică mulțimi de puncte din E^n într-unul din numerele $1, 2, \dots, R$. Mulțimile care conțin pattern-urile din clasele $1, 2, \dots, R$ le vom nota cu $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_R$, respectiv. În figura 3.3, $n = 2$, $R = 4$ și $i_0 = j$ 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 *suprafețe de decizie*. Presupunem că pattern-urile de pe suprafețele de decizie nu aparțin nici unei clase. În spațiul n -dimensional, suprafețele de decizie sunt hipersuprafețe $(n - 1)$ dimensionale.

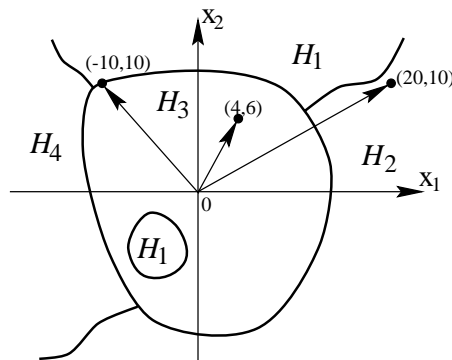


Figura 3.3: Regiuni de decizie.

3.2 Funcții discriminant

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

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

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

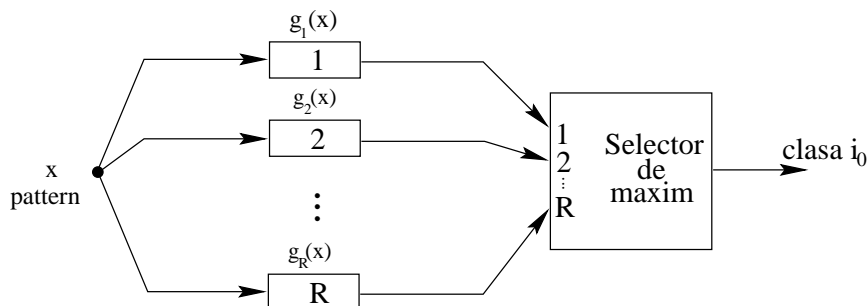


Figura 3.4: Clasificator.

Dacă \mathbf{x} se află pe suprafața de decizie dintre \mathcal{H}_i și \mathcal{H}_j , atunci

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

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

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

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

Funcția

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

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

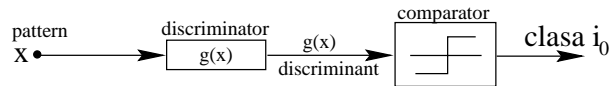


Figura 3.5: Dihotomizator.

Ne vom concentra asupra clasificatorilor ale căror funcții discriminant se obțin prin învățare iterativă, cu ajutorul pattern-urilor de instruire. Presupunem că:

1. Mulțimea pattern-urilor de instruire și clasificarea lor sunt cunoscute, deci învățarea este supervizată.
2. Funcțiile discriminant sunt liniare și doar coeficienții lor sunt ajustați pe parcursul procesului de învățare.

3.3 Clasificatori liniari

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

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

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

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

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

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

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

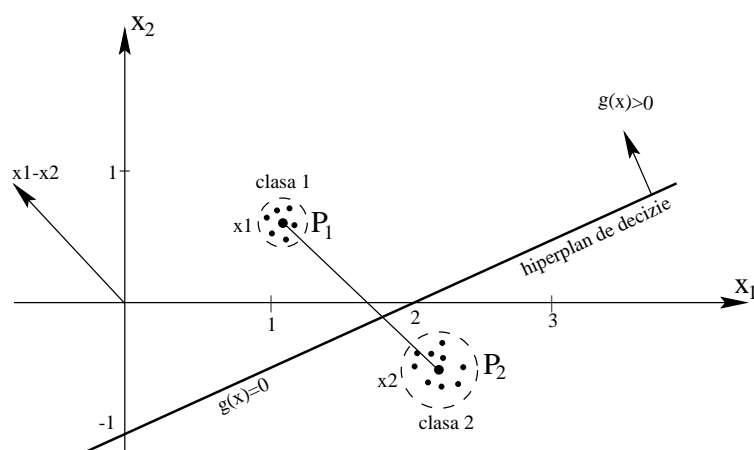


Figura 3.6: Separarea a două clase printr-un hiperplan de decizie.

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

este vectorul ponderilor. În cazul precedent,

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

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

Funcțiile discriminant liniare pot fi folosite și 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 distanței minime pentru a clasifica pattern-uri și că avem R clase. Fiecare clasă este reprezentată de un punct prototip P_1, P_2, \dots, P_R reprezentat de un vector $\mathbf{x}_1, \dots, \mathbf{x}_R$. Distanța euclidiană dintre un pattern de intrare \mathbf{x} și 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 distanțele dintre \mathbf{x} și \mathbf{x}_i , $i = 1, 2, \dots, R$, alegând apoi clasa pentru care distanța este minimă. Avem:

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

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

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

Funcția discriminant are forma

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

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

Clasificatorii care folosesc criteriul distanței minime pot fi considerați *clasificatori liniari* și ei se numesc uneori *mașini liniare*. Un clasificator liniar arată ca în figura 3.7.

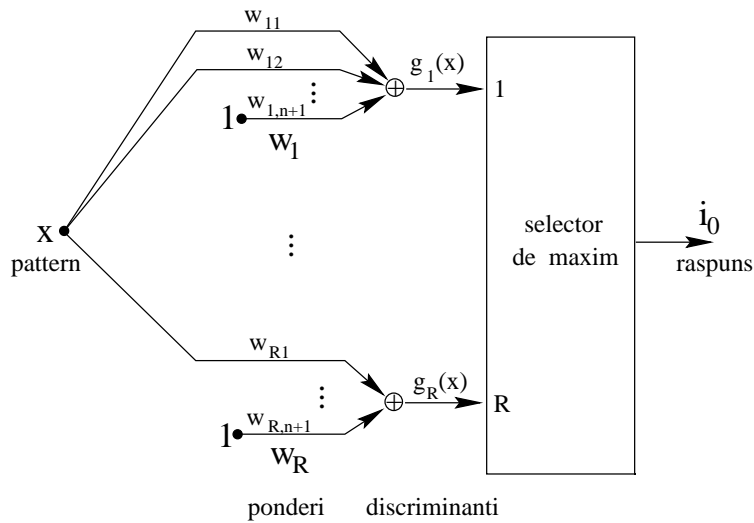


Figura 3.7: Clasificator liniar.

Să observăm că suprafața de decizie S_{ij} dintre regiunile contigue (alăturate, vecine) H_i și 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:

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

obținem:

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

Să presupunem că avem o mulțime \mathcal{H} de pattern-uri pe care o împărțim în submulțimile $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_R$. Dacă o mașină liniară poate clasifica pattern-urile din \mathcal{H}_i ca făcând parte din clasa i , pentru $i = 1, 2, \dots, R$, atunci mulțimile \mathcal{H}_i sunt *liniar separabile*. Mai formal, dacă există R funcții liniare pentru care:

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

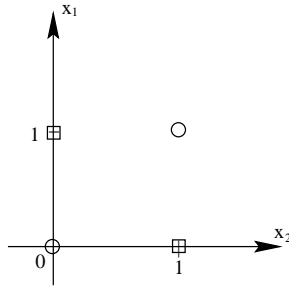


Figura 3.8: Pattern-uri neseperabile liniar.

atunci mulțimile \mathcal{H}_i sunt liniar separabile.

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

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

unde \oplus este operatorul OR exclusiv.

3.4 Utilizarea perceptronului discret în instruirea unui dihotomizator liniar

Am văzut în secțiunea precedentă cum se pot determina coeficienții discriminanților liniari (ponderile) din informația *a priori* asupra pattern-urilor și a apartenenței lor la diferite clase.

În această secțiune, ponderile se vor obține printr-un proces de învățare. Pattern-urile de instruire $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_P$ sunt prezentate mașinii liniare (fig. 3.9) împreună cu răspunsurile corecte. Această secvență de pattern-uri se numește *secvență de instruire*. Răspunsul este dat de către supervisor, iar clasificatorul își modifică parametrii pe baza învățării iterative supervizate.

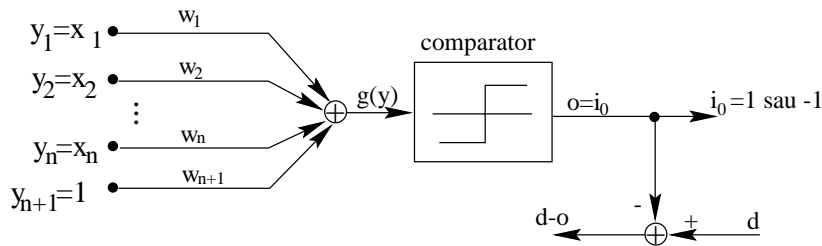


Figura 3.9: Mașina liniară.

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

Suprafața de decizie în spațiul E^n are ecuația:

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

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

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

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

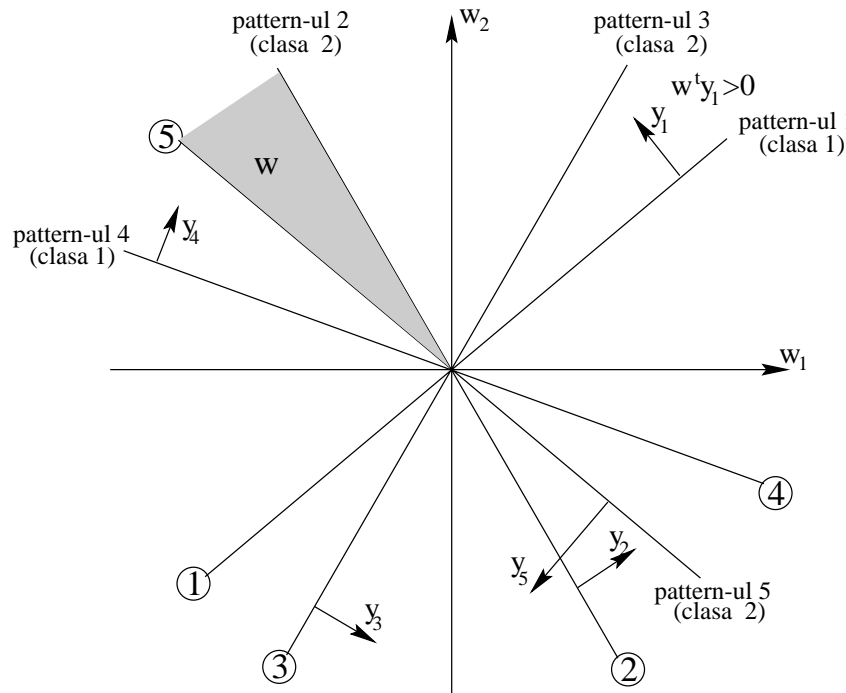


Figura 3.10: Reprezentarea unor suprafețe de decizie în spațiul ponderilor.

Să presupunem acum că ponderile inițiale 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 direcția 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 greșit, vom ajusta ponderile astfel:

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

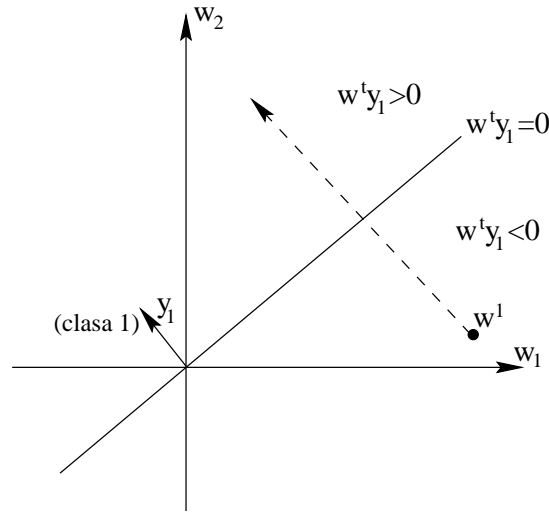


Figura 3.11: Ajustarea ponderilor pentru un pattern din clasa 1.

unde $c > 0$ este *incrementul de corecție*. 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 inițiale 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 $\mathbf{w}^{1t} \mathbf{y}_1 > 0$. Descreștem cât mai rapid pe $\mathbf{w}^t \mathbf{y}_1$, deci în direcția gradientului negativ:

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

Procedura de instruire supervizată este, așadar:

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

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

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

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

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

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

Se poate demonstra următoarea teoremă:

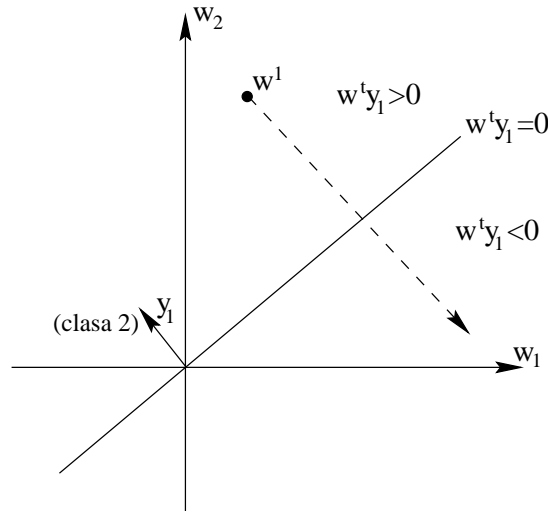


Figura 3.12: Ajustarea ponderilor pentru un pattern din clasa 2.

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

Vom discuta mai detaliat această teoremă în secțiunea următoare.

Valoarea lui k_0 depinde de incrementul de corecție și de secvența de pattern-uri utilizate pentru instruire.

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

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

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

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

Dacă $d^k = -1$ și $o^k = 1$, 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 $(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_P, d_P)$, unde \mathbf{x}_i este un vector de n elemente, $i = 1, 2, \dots, P$. Se definește vectorul $\mathbf{y}_1 = [\mathbf{x}_i \ 1]^t$, $i = 1, 2, \dots, P$.

1. Se alege $c > 0$.

2. Se inițializează vectorul \mathbf{w} de $n + 1$ elemente cu valori aleatoare și mici. Se fac următoarele inițializări: $k \leftarrow 1$, $i \leftarrow 1$ și $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_i$, $d \leftarrow d_i$, $o \leftarrow \text{sgn}(\mathbf{w}^t \mathbf{y})$.
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 diferențiabile ale comparatorului, ceea ce permite calcularea gradientului erorii.

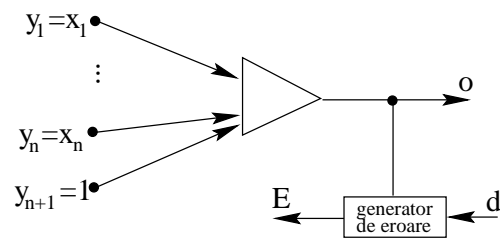


Figura 3.13: Modelul folosit la instruirea unui perceptron continuu folosit ca dihotomizator liniar.

Problema modificării ponderilor se rezolvă foarte elegant prin minimizarea funcției care măsoară eroarea de clasificare, iar această minimizare se realizează prin folosirea gradientului (fig. 3.13). Se pornește de la un vector \mathbf{w} arbitrar și se calculează $\nabla E(\mathbf{w})$, gradientul funcției eroare curente. Următoarea valoare a lui \mathbf{w} este obținută îndreptându-ne în direcția gradientului negativ de-a lungul suprafeței multidimensionale a erorii. Direcția gradientului negativ este cea a descreșterii cea mai rapidă. Luăm deci:

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

unde η este *constantă de învățare*, o valoare pozitivă.

Eroarea de clasificare care trebuie minimizată este:

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

Obținem (pentru simplificarea scrierii, oținem temporar indicii k):

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

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

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

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

Dacă funcția 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 $o = f(net)$ și obținem:

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

Rezultă:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \frac{1}{2}\eta(d^k - o^k)(1 - o^{k2})\mathbf{y}^k.$$

Această regulă corectează ponderile în aceeași direcție ca și 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 corecție a ponderilor. Ambele reguli implică adunarea sau scăderea unei fracțiuni din \mathbf{y} . Diferența esențială este prezența factorului moderator $1 - o^{k2}$. Acest factor este $0 < 1 - o^{k2} < 1$. Atunci când răspunsurile sunt eronate, avem următoarea proprietate: pentru net apropiat de 0, factorul de corecție este mai mare decât dacă net este departe de 0. Aceasta contribuie la stabilitatea procesului de învățare.

O altă diferență semnificativă este că algoritmul pe baza perceptronului discret duce întotdeauna la o soluție (dacă soluția 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 direcția lui $-\nabla E(\mathbf{w})$. Pentru η mic, putem considera că mergem în această direcție. 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 convergența către o soluție.

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

Se dau P perechi $(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_P, d_P)$, unde \mathbf{x}_i este un vector de n elemente, $i = 1, 2, \dots, P$. Se definește vectorul $\mathbf{y}_1 = [\mathbf{x}_i \ 1]^t$, $i = 1, 2, \dots, P$.

1. Se alege $\eta > 0$, $\lambda = 1$ și $E_{max} > 0$.
2. Se inițializează vectorul \mathbf{w} de $n + 1$ elemente cu valori aleatoare și mici. Se fac următoarele inițializări: $k \leftarrow 1$, $i \leftarrow 1$ și $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_i$, $d \leftarrow d_i$, $o \leftarrow f(\mathbf{w}^t \mathbf{y})$, care este funcția 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).

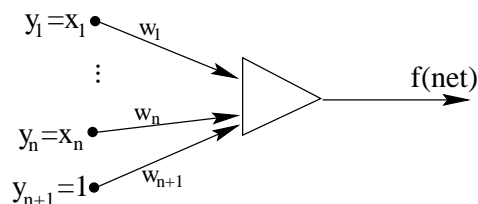


Figura 3.14: Perceptron instruit ca dihotomizator.

Ca urmare a procesului de instruire, obținem același 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:

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

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

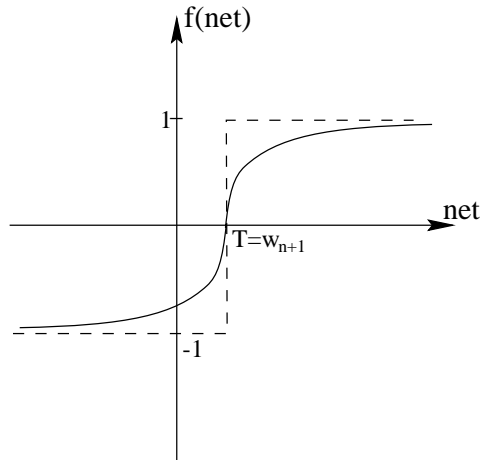


Figura 3.15: Funcția de activare folosită de un perceptron instruit ca dihotomizator.

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 $y_{n+1} = -1$ și $w_{n+1} = T$, în cazul în care nu specificăm altfel.

3.6 Mai multe despre teorema de convergență a perceptronului

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

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

Iterațiile se referă la relația:

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

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) și de Minsky și Papert (1969):

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

Trebuie să notăm că:

1. Ponderile inițiale pot fi numere reale oarecare.
2. Nu îl putem determina pe k_0 deoarece depinde de \mathbf{w}^* , unde \mathbf{w}^* este un vector soluție pentru ponderi. Deci ar trebui să îl cunoaștem *a priori* pe \mathbf{w}^* pentru a determina numărul de iterații. 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 secvență 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 pași, exact una dintre următoarele două variante:

1. Produce un vector al ponderilor pentru care perceptronul clasifică corect toate pattern-urile de instruire, dacă și numai dacă aceste pattern-uri sunt liniar separabile.
2. Părăsește și revizitează un vector al ponderilor, dacă și numai dacă pattern-urile de instruire nu sunt liniar separabile.

Se pot face acum următoarele observații:

1. Rezultă astfel o procedură de testare a liniar separabilității.
2. Nu se cunoaște 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 convergă către o soluție, chiar dacă pattern-urile sunt liniar separabile (constantă de învățare fiind oarecare).

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

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

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

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

Definim vectorul ponderilor

$$\mathbf{w}_q = \left[w_{q1} \quad w_{q2} \quad \dots \quad w_{q,n+1} \right]^t.$$

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

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

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:

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

sau:

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

Acest clasificator are la ieșire un selector de maxim.

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

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

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

Aceasta este regula de instruire a clasificatorului bazat pe o rețea de perceptroni discreți. În relația de mai sus, d_i și o_i se referă la al i -lea perceptron.

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

Algoritmul de instruire a unei rețele monostrat de perceptroni folosită ca și clasificator liniar este următorul:

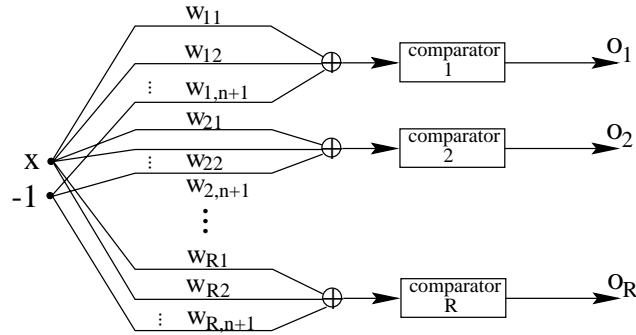


Figura 3.16: Clasificator liniar cu R clase.

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

1. Se alege $c > 0$, ca în cazul perceptronului discret.
2. Se inițializează matricea \mathbf{W} de $R \times (n + 1)$ cu valori aleatoare și mici. Se inițializează $k \leftarrow 1$, $p \leftarrow 1$ și $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_p$, $\mathbf{d} \leftarrow \mathbf{d}_p$, $o_i \leftarrow \text{sgn}(\mathbf{w}_i^t \mathbf{y})$, pentru $i = 1, 2, \dots, R$, 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 $i = 1, 2, \dots, R$, unde i reprezintă elementul al i -lea din vectorul \mathbf{d} .
5. $E \leftarrow \frac{1}{2}(d_i - o_i)^2 + E$, $i = 1, 2, \dots, R$.
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 și o rețea de perceptroni de tip continuu (fig. 3.17).

În acest caz, ajustarea ponderilor se face astfel:

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

Asupra acestei formule și a rețelei 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):

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

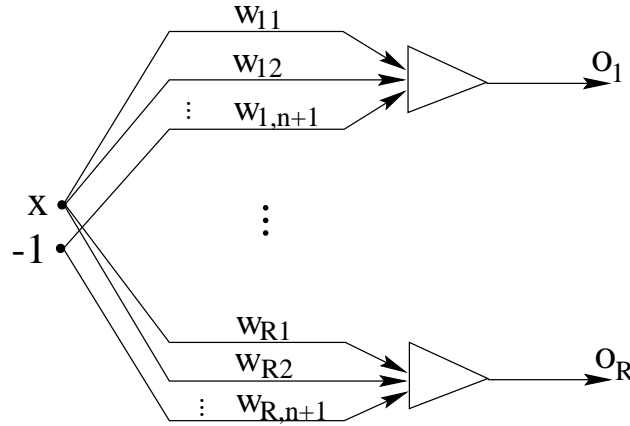


Figura 3.17: Rețea de perceptroni de tip continuu.

cu $n = 2$ și $R = 3$.

Folosim formula

$$\mathbf{w}_i = \mathbf{x}_i$$

$$w_{i,n+1} = -\frac{1}{2} \mathbf{x}_i^t \mathbf{x}_i, \quad i = 1, 2, \dots, R.$$

și obținem 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:

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

obținem funcțiile 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 $g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0$, obținem:

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

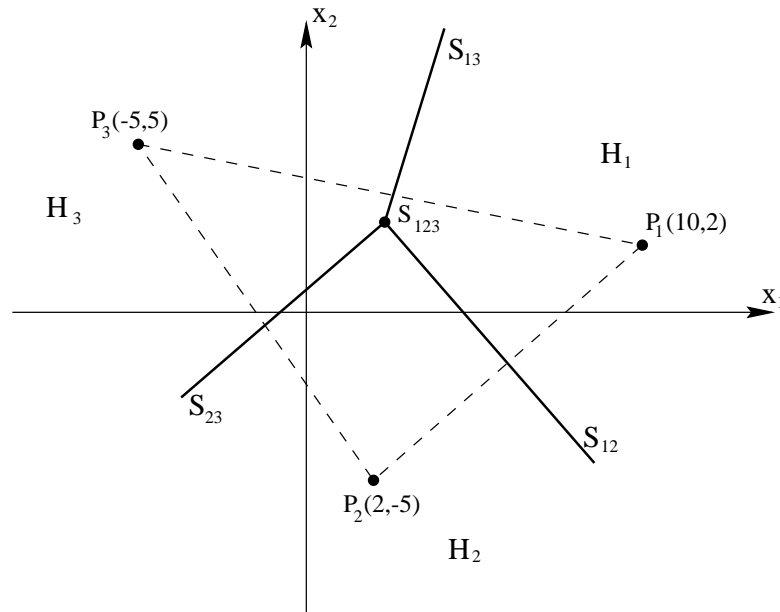


Figura 3.18: Suprafețele de decizie pentru un clasificator liniar. $S_{123} = (2, 337; 2, 686)$.

Exemplu de instruire a unui dihotomizator cu ajutorul unui perceptron discret

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

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

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 inițiale $\mathbf{w}^1 = [-2,5 \quad 1,75]^t$ alese arbitrar. Regula $\mathbf{w}' = \mathbf{w} \pm c\mathbf{y}$ devine în cazul perceptronului:

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

Alegem $c = 1$.

Pasul 1. Aplicăm \mathbf{y}_1

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

$$d_1 - o_1 = 2$$

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

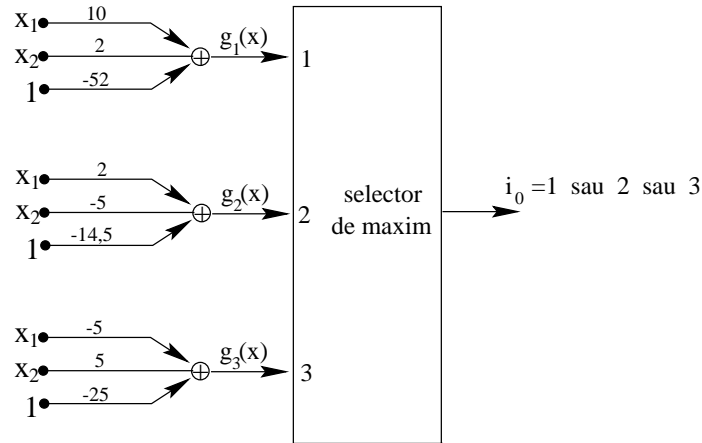


Figura 3.19: Clasificatorul linear care implementeaza regiunile de decizie din figura 3.18.

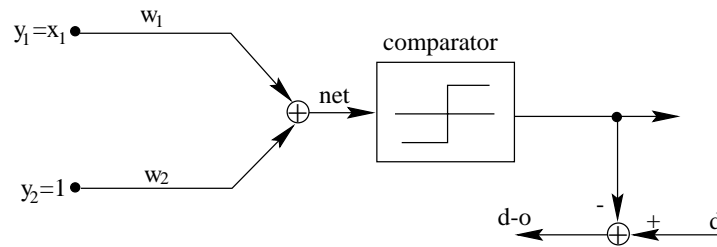


Figura 3.20: Perceptron discret folosit ca un clasificator instruibil.

Pasul 2. Aplicăm 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 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 y_4 . După citirea lui y_4 , nu se modifică ponderile. Reciclăm secvența y_1, \dots, y_4 deoarece nu știm dacă perceptronul a reușit să învețe 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

$$\mathbf{w}^7 = [2,5 \quad 1,75]^t.$$

Pasul 8. Aplicăm \mathbf{y}_4

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

Reciclăm din nou și obținem:

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

$$\mathbf{w}^{11} = [3 \quad 0,75] \text{ care sunt ponderile finale.}$$

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 $k = 1, 2, 3, 4$, obținem:

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

Considerăm $\lambda = 1$ și obținem:

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

La următorii pași obținem:

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

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

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

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

Exemplu de instruire a unui clasificator reprezentat ca rețea de perceptroni discreți

Reluăm clasificatorul liniar ($R = 3$) construit deja pe baza distanței minime. Sunt ușor de calculat valorile funcțiilor discriminant în \mathbf{x}_1 , \mathbf{x}_2 și \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 obțin, desigur, pe diagonală. Construim (nu prin instruire) rețeaua de perceptroni corespunzătoare acestui clasificator (fig. 3.21).

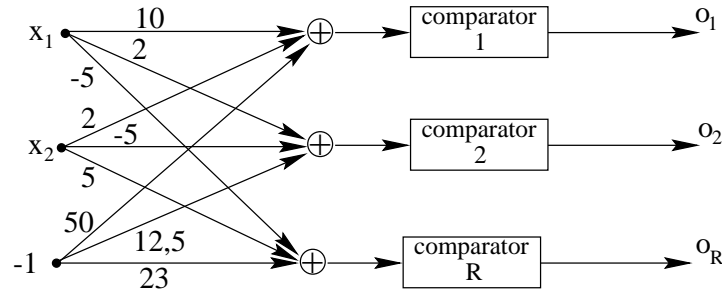


Figura 3.21: Rețea de perceptroni.

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

Vom construi acum direct clasificatorul prin instruirea rețelei de perceptroni. Alegem aleator vectorii inițiali:

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

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

Pasul 1. (\mathbf{y}_1)

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

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

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 secvența etc. Se modifică doar ponderile primului perceptron.

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

Obținem o rețea 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 \mathbf{Q} avem $\mathbf{o} = [1 \ 1 \ -1]^t$. Pe de altă parte, pattern-uri ca acest \mathbf{Q} nu au fost utilizate la instruire.

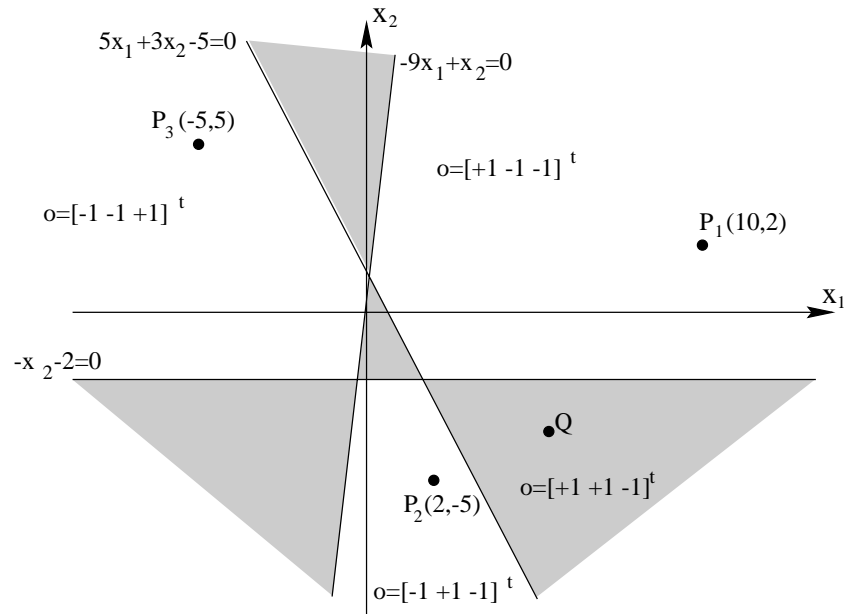


Figura 3.22: Regiuni de indecizie.

3.9 Exerciții

1. (C) Implementați algoritmul de instruire a unui dihotomizator cu ajutorul unui perceptron discret luând $c = 1$, $\mathbf{w} = [0 \ 0 \ 0 \ 0]^t$ și

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

Încercați și alte valori pentru c .

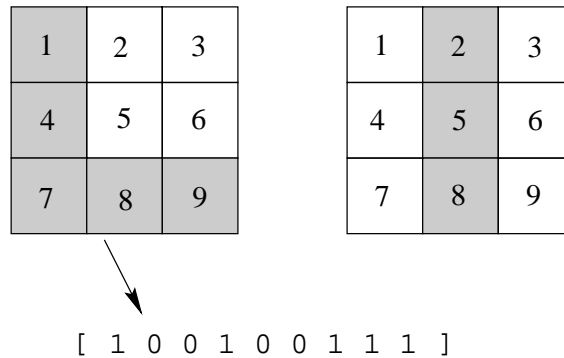


Figura 3.23: Caracterele tipărite L și I folosite al instruirea unui dihotomizator.

2. (C) Implementați algoritmul de instruire a unui dihotomizator cu ajutorul unui perceptron continuu în condițiile problemei precedente. Încercați diferite valori pentru constanta de învățare.
3. Prin algoritmul de instruire a unui clasificator liniar cu trei clase reprezentat printr-o rețea de trei perceptroni discreți, s-a ajuns ca în trei pași să se termine instruirea. Se consideră $c = 1$.

Pasul 1. (\mathbf{x}_1). Se ajustează $\mathbf{w}_1^1 = \mathbf{w}_2^1 = \mathbf{w}_3^1 = [0 \ 0 \ 0]^t$.

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

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

În final,

$$\mathbf{w}_1^4 = [1 \ 3 \ -1]^t, \quad \mathbf{w}_2^4 = [5 \ -1 \ -2]^t, \quad \mathbf{w}_3^4 = [1 \ -1 \ 2]^t.$$

Găsiți pattern-urile \mathbf{x}_1 , \mathbf{x}_2 și \mathbf{x}_3 .

4. (C) Implementați algoritmul de instruire a unui clasificator liniar reprezentat ca rețea de perceptroni discreți folosind $c = 1$ și următoarele pattern-uri de instruire:

$$\begin{aligned} i_0 = 1 \text{ pentru } \mathbf{x}: & \left[\begin{array}{ccc} 1 & 0 & 0 \end{array} \right]^t, & \left[\begin{array}{ccc} 1 & 1 & 0 \end{array} \right]^t \\ i_0 = 2 \text{ pentru } \mathbf{x}: & \left[\begin{array}{ccc} 1 & 0 & 1 \end{array} \right]^t, & \left[\begin{array}{ccc} 1 & 1 & 1 \end{array} \right]^t \\ i_0 = 3 \text{ pentru } \mathbf{x}: & \left[\begin{array}{ccc} 1 & 1 & 0 \end{array} \right]^t, & \left[\begin{array}{ccc} 0 & 1 & 1 \end{array} \right]^t \\ i_0 = 4 \text{ pentru } \mathbf{x}: & \left[\begin{array}{ccc} 0 & 0 & 0 \end{array} \right]^t, & \left[\begin{array}{ccc} 0 & 0 & 1 \end{array} \right]^t. \end{aligned}$$

Reprezentați suprafețele de decizie rezultate.

5. Proiectați și instruiți un clasificator pentru caracterele tipărite L și I (fig. 3.23).

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

Rezultat: Apare indecizia. Ar trebui să instruiem perceptronul și cu aceste litere distorsionate.

Capitolul 4

Rețele neurale feedforward multistrat

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

- fie prin utilizarea unor funcții discriminant neliniare (de ex. liniare pe porțiuni);
- fie printr-o rețea multistrat.

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

Rețelele multistrat pot implementa suprafețe de decizie arbitrare. În completare la capitolul 3, se pot rezolva astfel multe alte aplicații: aproximarea funcțiilor, recunoașterea caracterelor scrise, recunoașterea vorbirii, sisteme expert, generarea traiectoriilor etc.

Rețelele neurale multistrat sunt în prezent cele mai răspândite arhitecturi. În acest capitol, vom studia rețelele neurale multistrat instruibile.

4.1 Clasificarea pattern-urilor liniar neseparabile

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

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

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

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

Această clasificare se poate implementa ca în figura 4.2.

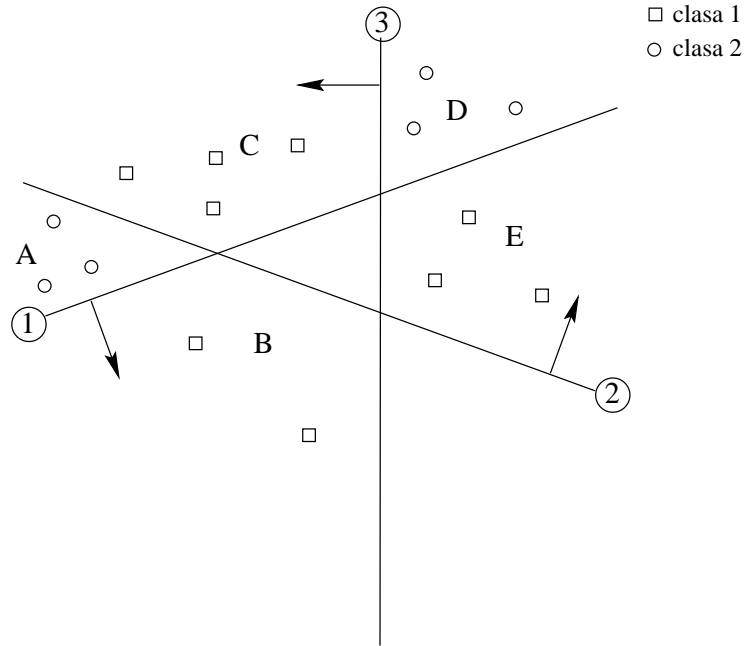


Figura 4.1: Două mulțimi de pattern-uri care trebuie clasificate în două categorii.

Fiecare dintre cele șapte compartimente (inclusiv A,...,E) sunt aplicate într-unul din vârfurile cubului, în spațiul $o_1o_2o_3$ (spațiul o). Se observă că în spațiul imagine o , pattern-urile din cele două clase sunt ușor 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 învățare delta pentru o rețea de neuroni monostrat

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

În această rețea, $\mathbf{o} = \Gamma[\mathbf{W}\mathbf{y}]$, unde:

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

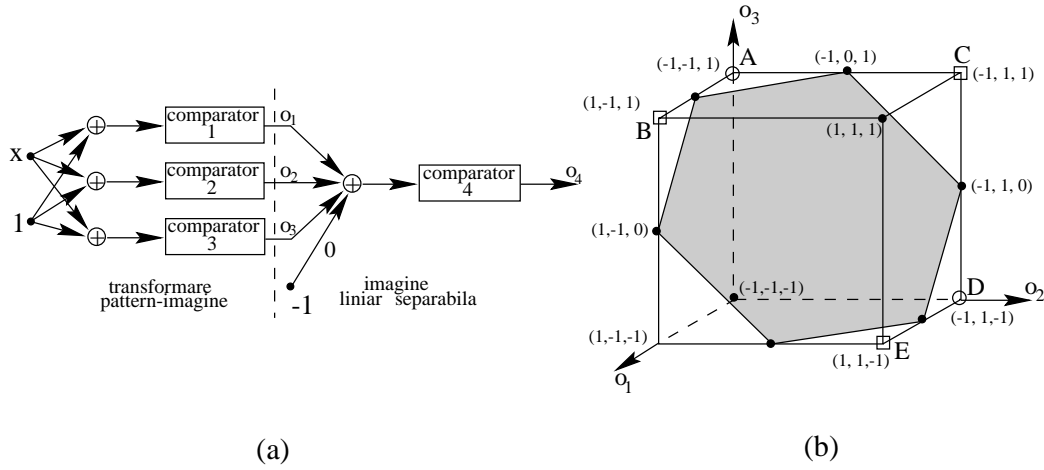


Figura 4.2: (a) Clasificator monostrat; (b) Reprezentarea pattern-urilor din figura 4.1 în spațiul imagine o .

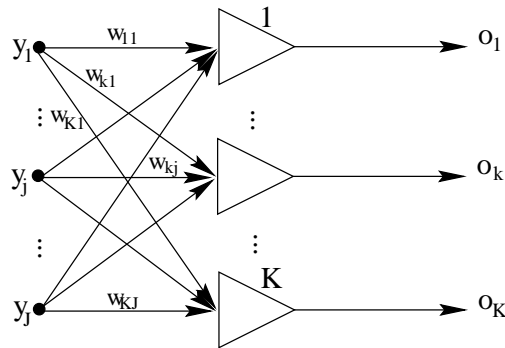


Figura 4.3: Rețea monostrat de perceptoni de tip continuu.

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

Prin definiție,

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

este răspunsul dorit. Generalizăm expresia erorii:

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

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

$$\mathbf{d}_p = \begin{bmatrix} d_{p1} \\ \vdots \\ d_{pK} \end{bmatrix} \quad \mathbf{o}_p = \begin{bmatrix} o_{p1} \\ \vdots \\ o_{pK} \end{bmatrix}.$$

Pentru simplitate, să presupunem că

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

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

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

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

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

Semnalul de eroare pentru al k -lea neuron este:

$$\delta_{o_k} = -\frac{\partial E}{\partial \text{net}_k}.$$

Acest semnal l-am numit și *semnal de învățare*. Avem:

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

deoarece

$$\frac{\partial \text{net}_k}{\partial w_{kj}} = y_j.$$

Putem scrie:

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

Calculăm:

$$\delta_{o_k} = -\frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial \text{net}_k}$$

Dar

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

și

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

Obținem:

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

Formula de ajustare a ponderilor este atunci:

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

și este identică cu regula de învățare delta pentru un singur perceptron.

Pentru funcția de activare continuă unipolară, obținem

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

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:

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

și deci:

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

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

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

pentru

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

și

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

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 rețele monostrat de perceptroni de tip continuu prin regula delta este următorul:

Se dau perechile $(\mathbf{y}_1, \mathbf{d}_1), (\mathbf{y}_2, \mathbf{d}_2), \dots, (\mathbf{y}_P, \mathbf{d}_P)$, 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 , $i = 1, 2, \dots, P$, este -1 .

1. Se alege $\eta > 0$ și $E_{max} > 0$.
2. Se inițializează matricea \mathbf{W} de $K \times J$ elemente cu valori aleatoare și mici. Se inițializează: $p \leftarrow 1$, $q \leftarrow 1$ și $E \leftarrow 0$.
3. $\mathbf{y} \leftarrow \mathbf{y}_p$, $\mathbf{d} \leftarrow \mathbf{d}_p$, $o_k \leftarrow f(\mathbf{w}_k^t \mathbf{y})$ pentru $k = 1, 2, \dots, K$, 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 $k = 1, 2, \dots, K$ în cazul bipolar
și conform regulii

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

pentru $k = 1, 2, \dots, K$ în cazul unipolar.

5. Se calculează eroarea pătratică cumulată:

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

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 învățare pentru rețele feedforward multi-strat. Considerăm pentru aceasta o rețea de neuroni cu două straturi (fig. 4.4).

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

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

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

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

unde δ_{yj} este semnalul de eroare pentru stratul ascuns având ieșirea y . Vom scrie

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

și atunci

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

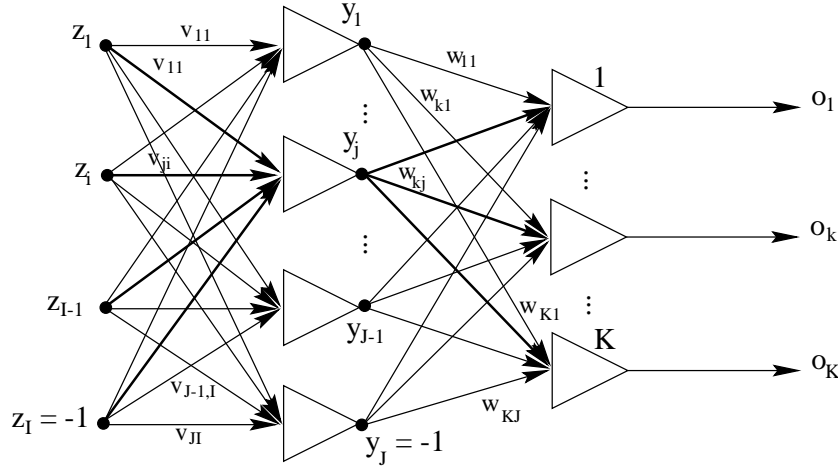


Figura 4.4: Rețea neurală feedforward cu un strat ascuns.

unde

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

Dar

$$f'_i(\text{net}_j) = \frac{\partial y_j}{\partial \text{net}_j}$$

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

deoarece

$$\delta_{o_k} = (d_k - o_k) f'(\text{net}_k)$$

$$\text{net}_k = \sum_{j=1}^J w_{kj} y_j.$$

Atunci:

$$\delta_{y_j} = f'_j(\text{net}_j) \sum_{k=1}^K \delta_{o_k} w_{kj}, \quad j = 1, 2, \dots, J$$

$$\Delta v_{ji} = \eta f'_j(\text{net}_j) z_i \sum_{k=1}^K \delta_{o_k} w_{kj}, \quad j = 1, 2, \dots, J, \quad i = 1, 2, \dots, I.$$

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

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

sau, matriceal:

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

unde

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

În această relație,

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

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:

$$\begin{array}{ll} \text{pentru stratul de ieșire:} & \mathbf{W}' = \mathbf{W} + \eta \delta_o \mathbf{y}^t \\ \text{pentru stratul ascuns:} & \mathbf{V}' = \mathbf{V} + \eta \delta_y \mathbf{z}^t \end{array}$$

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

$$\begin{array}{l} \delta_o = \left[\delta_{o_1} \quad \dots \quad \delta_{o_K} \right]^t, \quad \delta_{o_k} = (d_k - o_k) f'_{o_k}(\text{net}_k) \\ \delta_y = \left[\delta_{y_1} \quad \dots \quad \delta_{y_J} \right]^t, \quad \delta_{y_j} = \mathbf{w}_j^t \delta_o f'_{y_j}(\text{net}_j). \end{array}$$

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 rețea neurală multistrat.

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

În general, o rețea neurală multistrat aplică vectorul de intrare \mathbf{z} în vectorul de ieșire \mathbf{o} astfel:

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

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

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

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

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

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

Se dau perechile $(\mathbf{z}_1, \mathbf{d}_1), (\mathbf{z}_2, \mathbf{d}_2), \dots, (\mathbf{z}_P, \mathbf{d}_P)$, unde \mathbf{z}_i este un vector de I elemente, $z_{i_i} = -1$ și $i = 1, 2, \dots, P$. Se determină mărimea stratului ascuns. Acesta va avea ieșirile \mathbf{y} , unde \mathbf{y} este un vector de J elemente, iar $y_J = -1$. Ieșirile \mathbf{o} din rețea sunt vectori de K elemente.

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

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

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

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

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

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

3. Se calculează eroarea pătratică cumulată:

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

4. Se calculează semnalele de eroare pentru cazul bipolar:

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

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

respectiv, pentru cazul unipolar:

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

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

5. Se ajustează ponderile stratului de ieșire:

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

6. Se ajustează ponderile stratului ascuns:

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

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 direcția gradientului negativ al erorii, deoarece ponderile se modifică la fiecare pas. Pot apărea oscilații dacă η este mare. Dacă η este mic, oscilațiile sunt neglijabile, dar rata învățării este prea mică. Termenul în care apare α filtrează oscilațiile. De obicei, se ia $\alpha = 0,9$ și se poate mări η . Dacă $\alpha = 0$, ajungem la același 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 gradientilor negativi ai valorilor individuale pe care le ia E_p , adică pentru un pattern, și 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, funcția de activare bipolară duce la rezultate mai bune decât funcția unipolară.

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

generalizată, intrarea fixă este recomandată nu de o teoremă, ci de ideea obținerii 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 și 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 rețeaua este instruită ca și 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, ieșirile dorite sunt 0, cu excepția uneia care este 1 și corespunde clasei din care face parte pattern-ul respectiv. O rețea multistrat poate fi transformată într-un clasificator, după instruire, prin înlocuirea perceptronilor de tip continuu cu perceptroni discreți.

Clasificarea este o formă importantă a calculului neural, însă ea limitează potențialul de calcul al unei rețele, deoarece folosește răspunsul binar. O rețea multistrat poate fi utilizată și pentru aproximarea funcțiilor.

4.5 Factori ai învățării

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 direcția gradientului negativ.

O dificultate tipică ar fi că prin acest algoritm se poate să obținem doar un minim local pentru funcția eroare. Prin faptul că impunem o valoare minimă acceptabilă pentru E_m , putem controla acest lucru. Se poate însă să nu avem o convergență 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*.

Totuși, 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 rețeaua este mai mare, cu atât mai bine va funcționa instruirea. Natura stohastică este dată atât de ponderile inițiale, cât și de pattern-urile de instruire. Chiar când pattern-urile de instruire au o natură deterministă, adăugarea unui zgomot cu medie zero poate duce al îmbunătățirea eficienței instruirii.

Instruirea funcționează cel mai bine în condiții de randomizare. De aceea, ponderile inițiale se aleg cu valori aleatoare, iar pattern-urile de instruire este bine să fie într-o secvență 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 funcția de activare continuă bipolară.

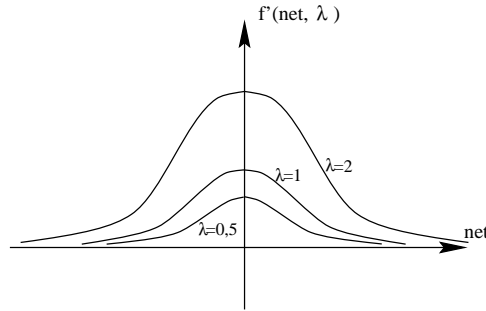


Figura 4.5: Efectul lui λ asupra lui $f'(net)$.

Deoarece ponderile sunt ajustate proporțional 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} și δ_{yj} sunt de asemenea proporționale 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 învățare fixată, ajustarea ponderilor este proporțională cu λ . Folosirea funcțiilor de activare cu valoare mare pentru λ are același efect cu utilizarea unei constante de învățare mari. De aceea, este recomandabil să păstrăm $\lambda = 1$ și să controlăm viteza de învățare doar prin η .

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

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

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

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

Alegerea lui J este încă o problemă în studiu. Rețelele cu un singur strat ascuns pot forma regiuni arbitrare de decizie pentru pattern-uri n -dimensionale. Să presupunem că avem o colecție de pattern-uri n -dimensionale liniar separabile în M regiuni disjuncte aparținând 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 și Cao¹ au arătat care este numărul maxim de regiuni liniar separabile folosind J neuroni:

$$\sum_{k=0}^n \binom{J}{k},$$

unde $\binom{J}{k} = 0$ pentru $k > J$.

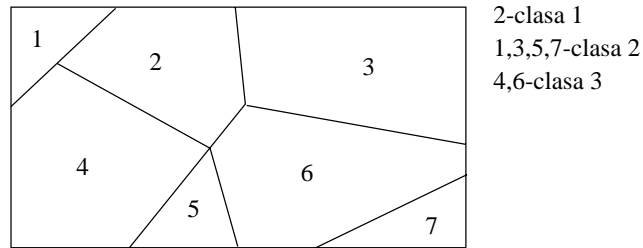


Figura 4.6: Regiuni disjuncte care aparțin la trei clase.

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

$$\binom{J}{0} + \binom{J}{1} + \binom{J}{2} = 1 + \frac{J}{2} + \frac{J^2}{2} \geq 7.$$

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

$$\binom{J}{0} + \binom{J}{1} + \dots + \binom{J}{J} = 2^J$$

și de aici îl obținem pe J .

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

4.6 Rețele feedforward multistrat folosite ca aproximatori universali

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

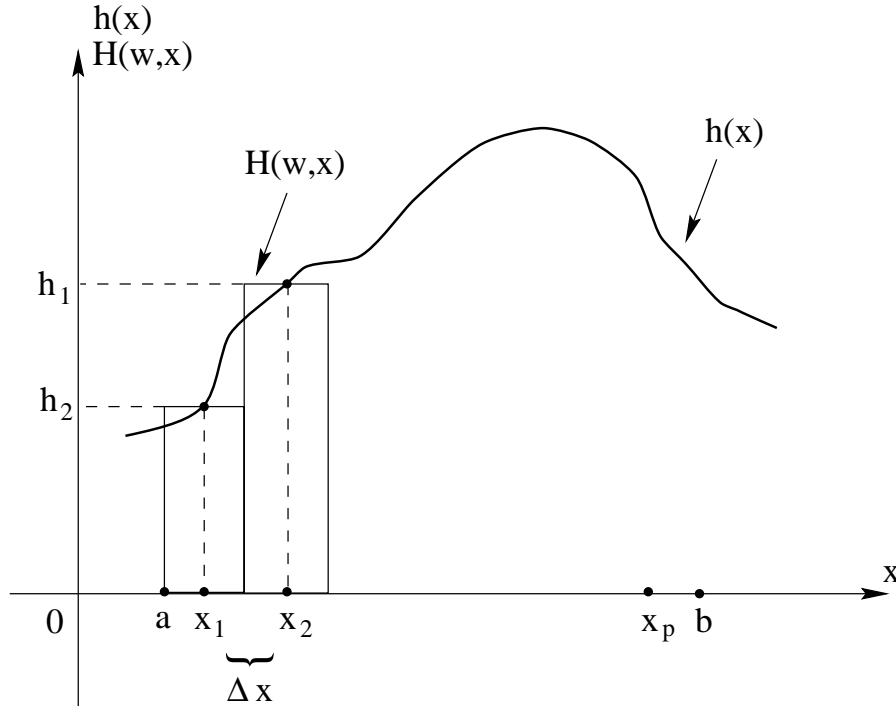


Figura 4.7: Aproximarea funcției $h(x)$ cu funcția treaptă $H(\mathbf{w}, x)$.

Presupunem că se cunosc P eșantioane, adică valorile funcției $h(x)$ în punctele x_1, x_2, \dots, x_P . Presupunem că

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

Definim funcția

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

Scriem atunci:

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

unde $h_i = h(x_i)$.

Putem scrie:

$$\begin{aligned} & \xi\left(x - x_i + \frac{\Delta x}{2}\right) - \xi\left(x - x_i - \frac{\Delta x}{2}\right) = \\ & \frac{1}{2} \operatorname{sgn}\left(x - x_i + \frac{\Delta x}{2}\right) - \frac{1}{2} \operatorname{sgn}\left(x - x_i - \frac{\Delta x}{2}\right). \end{aligned}$$

Acest termen este o fereastră de înălțime unitară și lățime Δx (fig. 4.8).

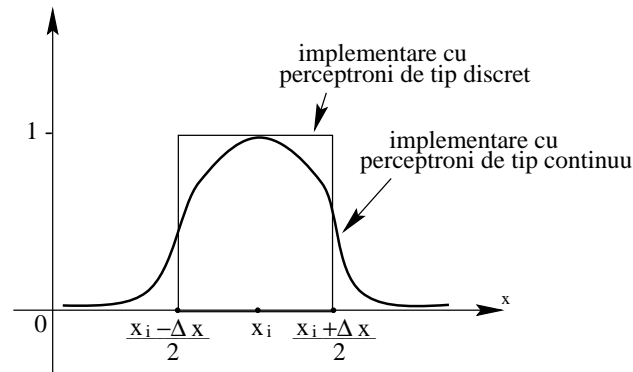


Figura 4.8: Fereastra de înălțime unitară și lățime Δx .

Fereastra se poate implementa ca în figura 4.9.

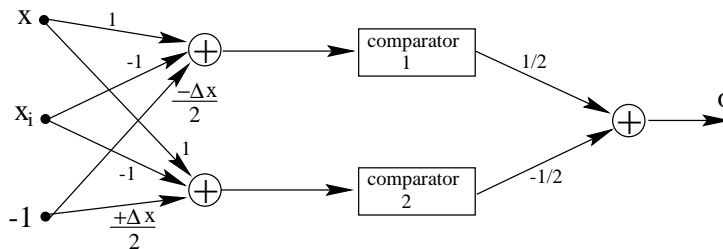


Figura 4.9: Rețea neurală care implementează fereastra de înălțime unitară și lățime Δx .

O rețea cu $2P$ perceptroni discreți poate implementa funcția $H(\mathbf{w}, x)$. Pentru un termen, rețeaua se poate descrie ca în figura 4.10.

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

Fără a demonstra, vom menționa două importante proprietăți:

1. O rețea feedforward cu un singur strat ascuns poate aproxima oricât de bine orice funcție neliniară continuă.
2. O rețea feedforward cu un singur strat ascuns poate implementa o funcție booleană arbitrară.

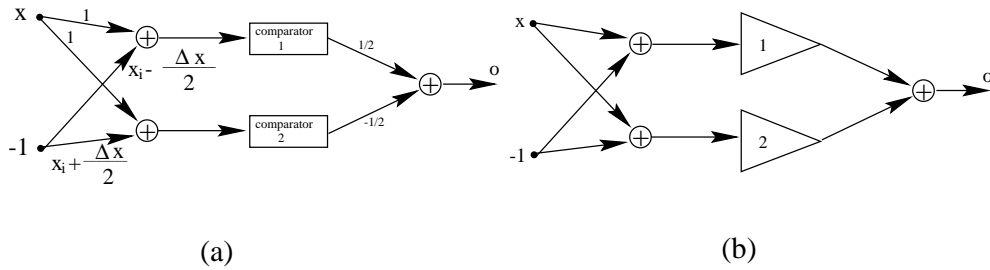


Figura 4.10: Implementare folosind (a) perceptroni de tip discret și (b) perceptroni de tip continuu; cu cât λ este mai mare, cu atât mai mult forma ferestrei se apropie de cea rectangulară.

4.7 Teorema lui Kolmogorov și rețelele neurale

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

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

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

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

unde φ_q , $q = 1, \dots, 2n + 1$ și ψ_{pq} , $p = 1, \dots, n$ sunt funcții continue de o singură variabilă.

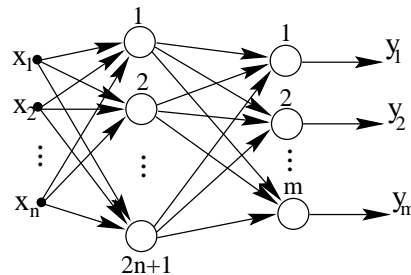


Figura 4.11: Rețea neurală care implementează o funcție oarecare f .

Interpretarea neurală este următoarea:

T1 (Hecht-Nielsen, 1986). Fie $f : [0, 1]^n \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$, $f(x) = y$, o funcție continuă oarecare. Atunci, f poate fi implementată exact

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

Definim acum funcțiile de activare pentru această arhitectură.

Fie z_k ieșirea 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 funcție monoton crescătoare, independentă de f , ε este o constantă rațională $0 < \varepsilon \leq \delta$, unde δ este o constantă pozitivă arbitrar aleasă.

Ieșirile y_i se calculează astfel:

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

unde funcțiile g_i , $i = 1, \dots, m$ sunt reale și continue, depinzând de f și ε .

Această teoremă nu ne spune cum să obținem funcția ψ sau constanta ε . Este o teoremă *existențială*. Ea se poate extinde pentru orice funcții continue pe un compact (mulțime închisă și mărginită). Acest rezultat a fost primul pas. Cybenko (1989) a arătat că orice funcție continuă definită pe un compact din \mathbb{R}^n poate fi aproximată oricât de bine de o rețea feedforward cu un singur strat ascuns folosind funcții de activare sigmoideale. Uneori se pot obține aproximări mai compacte folosind mai multe straturi ascunse.

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

4.8 Aplicații

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

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

Să notăm că ieșirile cu mai mult de două valori pentru un neuron pot modela incertitudinea. Modul convențional 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

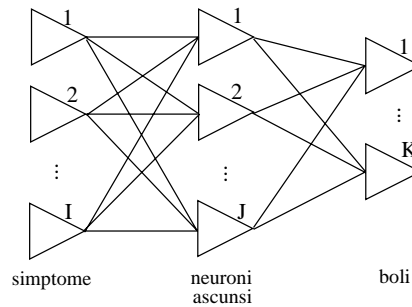


Figura 4.12: Sistem expert conexionist pentru diagnoza medicală.

multe ori este dificil de formulat aceste reguli. Deoarece rețelele neurale pot fi instruite fără ca ele să încapsuleze cunoștințele în reguli, să vedem în ce măsură ele pot fi aplicate ca o alternativă la sistemele expert convenționale.

Rețelele neurale pentru diagnosticare, defectoscopie, predicție, recunoașterea formelor rezolvă în esență probleme de clasificare, asociere și generalizare. Aceste rețele pot achiziționa cunoștințe fără să extragă reguli de tip IF-THEN de la un expert uman. După instruire, ele pot funcționa ca sisteme expert. Ceea ce va lipsi este explicarea: un sistem expert neural nu poate explica utilizatorului raționamentul deciziilor luate. Numim aceste sisteme expert *sisteme expert conexioniste*.

Atunci când o rețea instruită este testată cu un pattern substanțial diferit de cele folosite pentru instruire, răspunsul obținut se presupune că rezolvă o problemă de generalizare. Exemple tipice de generalizare sunt diagnosticarea și predicția. Figura 4.12 descrie funcționarea unui sistem expert conexionist pentru diagnostic medical.

Într-un sistem expert convențional, o dificultate este formularea și introducerea regulilor de către expertul uman. Sistemul expert conexionist poate învăța din diagnosticul curent care a fost acceptat. O atenție specială trebuie dată alegerii lui J . Dacă J este prea mic, apar dificultăți de aplicare a celor I intrări în cele K ieșiri. Dacă J este prea mare, crește inutil timpul de instruire și de diagnosticare și ponderile sunt dificil de estimat corect.

O serie de aplicații folosesc pattern-uri temporale care se obțin prin eșantionarea unor semnale de tip continuu (fig. 4.13).

4.9 Exemple

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

Fie funcția de decizie XOR:

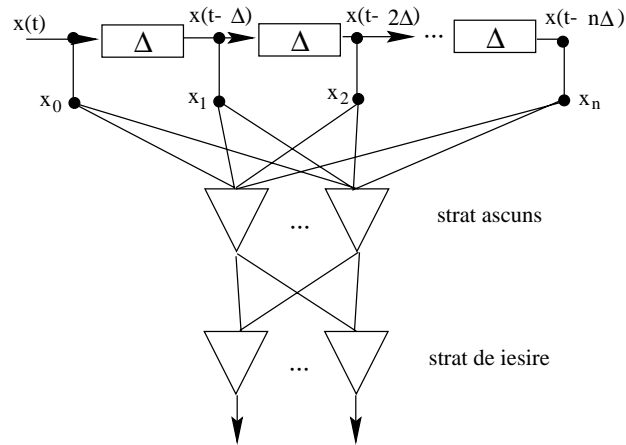


Figura 4.13: Rețea neurală cu întârziere cu un singur canal de achiziție.

x_1	x_2	ieșirea
0	0	1
0	1	-1
1	0	-1
1	1	1

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

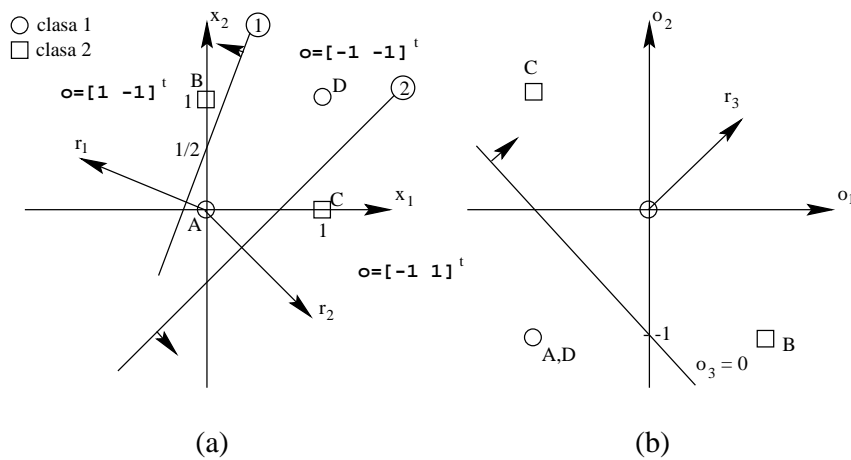


Figura 4.14: Suprafețe de decizie în problema XOR.

Alegem:

$$\begin{aligned}
 1 : & \quad -2x_1 + x_2 - \frac{1}{2} = 0 \\
 2 : & \quad x_1 - x_2 - \frac{1}{2} = 0
 \end{aligned}$$

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

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

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

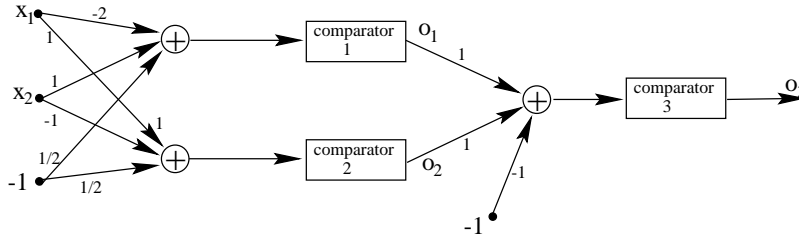


Figura 4.15: Rețeaua neurală care implementează problema XOR.

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

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

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

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

și atunci

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

Rețeaua 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 rețea cu trei neuroni cu un strat ascuns (fig. 4.16).

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

$$\delta_5 = (d_5 - o_5)o_5(1 - o_5)$$

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

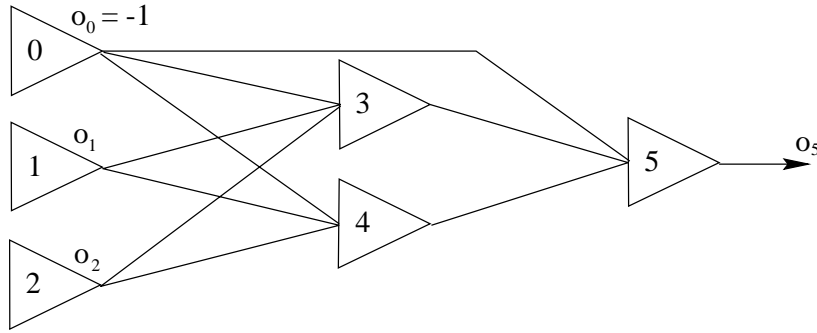


Figura 4.16: Rețea neurală multistrat. Funcțiile de activare sunt unipolare.

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

Apoi:

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

Putem astfel să construim prin instruire un clasificator multistrat pentru patternurile linear neseperabile 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 rețeaua să funcționeze ca un clasificator cu ieșiri binare. Înllocuim neuronii de tip continuu cu neuroni binari și 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 $J \leq n$, obținem pentru $J = 2$ numărul maxim de regiuni $2^2 = 4$.

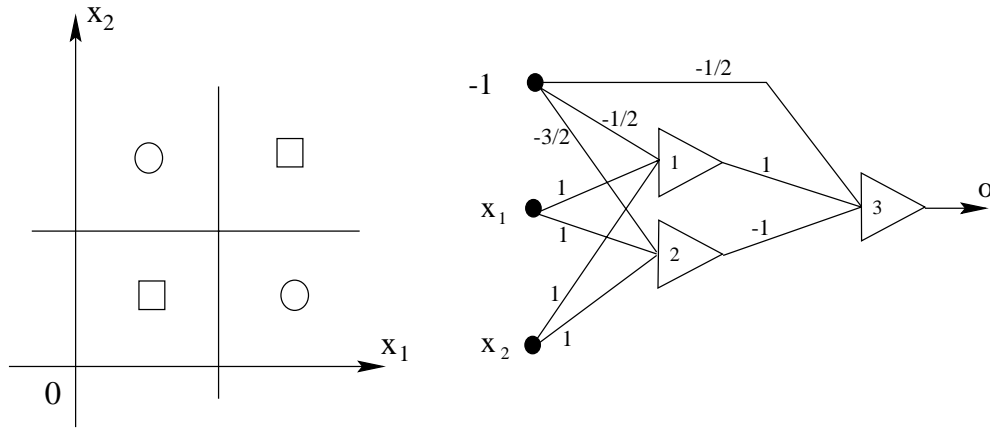


Figura 4.17: Problema XOR pentru $n = 2$.

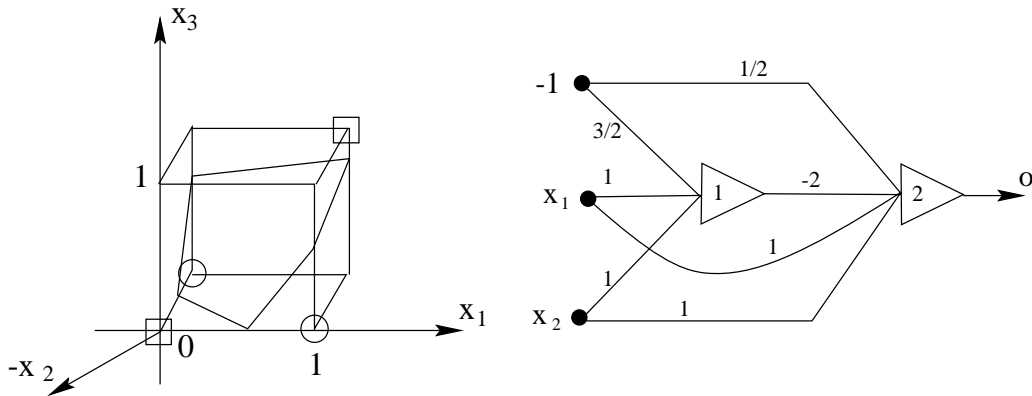


Figura 4.18: Problema XOR pentru $n = 3$.

Să considerăm acum aceeași problemă pentru $n = 3$ (fig. 4.18), unde $x_3 = x_1x_2$.

Luând $n = 3$ și $M = 2$ obținem $J = 1$. Am rezolvat deci aceeași problemă, dar cu mai puțini neuroni.

4.10 Exerciții

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:

clasa 1: $\mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9$
 clasa 2: restul.

Elaborați o rețea neurală de percepționi discreți cu un strat ascuns care să funcționeze ca un clasificator. Nu instruiți rețeaua de neuroni, ci folosiți 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 mașină multistrat cu doi percepționi discreți bipolari în stratul ascuns și un singur perceptron discret bipolar de ieșire trebuie să clasifice prototipurile astfel încât:

clasa 1: $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$
 clasa 2: restul .

(a) Verificați 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) Completați elaborarea clasificatorului folosind rezultatul anterior și calculați ponderile perceptronului de ieșire.

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

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

4. Fie problema clasificării în două clase a unor pattern-uri planare ($n = 2$) rezolvată printr-o arhitectură de rețea neurală cu $J = 8$ și $K = 2$. Determinați o margine inferioară pentru P , care este numărul vectorilor de instruire. Indicație: Acest număr este egal cu numărul regiunilor separabile, M .

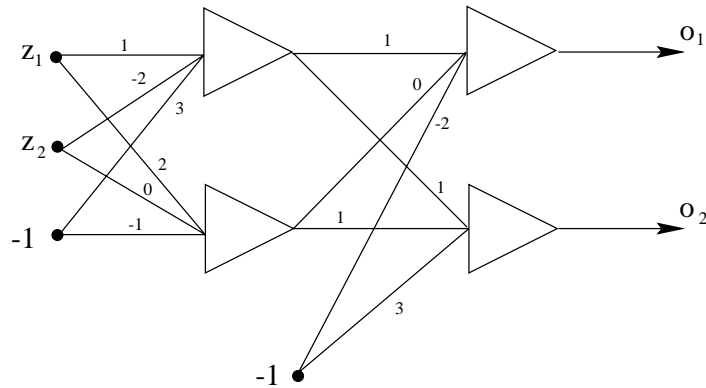


Figura 4.19: Rețea neurală multistrat pentru exercițiul 5.

5. Rețeaua din figura 4.19, dacă a fost instruită corect, trebuie să răspundă cu $o_1 = 0,95$ și $o_2 = 0,05$ la pattern-ul de intrare

$$\begin{bmatrix} z_1 \\ z_2 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}.$$

Ponderile au fost inițializate ca în figură. Se alege $\eta = 1$, iar neuronii folosesc funcția de activare

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

Analizați un pas al algoritmului de instruire prin propagarea în urmă a erorii realizând următoarele operații:

- Găsiți matricile ponderilor, \mathbf{V} , \mathbf{W} .
 - Calculați net_j , \mathbf{y} , net_k , \mathbf{o} .
 - Calculați $f'(net_j)$ și $f'(net_k)$.
 - Calculați δ_o și δ_y .
 - Calculați $\Delta\mathbf{V}$ și $\Delta\mathbf{W}$.
 - Găsiți noile valori pentru \mathbf{V} și \mathbf{W} .
6. (C) Implementați algoritmul de instruire prin propagarea în urmă a erorii pentru I , J , K și η selectabile și un singur strat ascuns. Folosiți perceptroni bipolari de tip continuu.
7. (C) Elaborați clasificatorul pentru literele A, I și O definite ca în figura 4.20, cu ajutorul programului de la exercițiul 6. Presupuneți două tipuri de reprezentări interne:

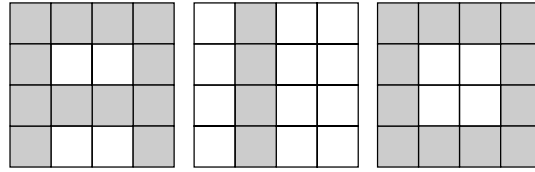


Figura 4.20: Literele A, I și O pentru exercițiul 7.

(a) $n = 16$, $P = 3$, $I = 17$, $J = 9$, $K = 3$ și

$$\begin{aligned} \text{clasa A: } \mathbf{x}_1 &= [1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1]^t \\ \text{clasa I: } \mathbf{x}_2 &= [0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0]^t \\ \text{clasa O: } \mathbf{x}_3 &= [1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1]^t \end{aligned}$$

(b) $n = 2$, $P = 16$, $I = 3$, $J = 9$, $K = 3$ și

$$\begin{aligned} \text{clasa A, O: } \mathbf{x}_1 &= [1 \ 1]^t \\ \text{clasa A, I, O: } \mathbf{x}_1 &= [1 \ 2]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [1 \ 3]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [1 \ 4]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [2 \ 1]^t \\ \text{clasa I: } \mathbf{x}_1 &= [2 \ 2]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [2 \ 4]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [3 \ 1]^t \\ \text{clasa A, I: } \mathbf{x}_1 &= [3 \ 2]^t \\ \text{clasa A: } \mathbf{x}_1 &= [3 \ 3]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [3 \ 4]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [4 \ 1]^t \\ \text{clasa I, O: } \mathbf{x}_1 &= [4 \ 2]^t \\ \text{clasa O: } \mathbf{x}_1 &= [4 \ 3]^t \\ \text{clasa A, O: } \mathbf{x}_1 &= [4 \ 4]^t \end{aligned}$$

Încercați diferite valori pentru η . Evaluați numărul de cicluri de instruire pentru ambele arhitecturi. Ieșirile trebuie să fie:

$$\begin{aligned} \text{pentru A: } & [1 \ -1 \ -1] \\ \text{pentru I: } & [-1 \ 1 \ -1] \\ \text{pentru O: } & [-1 \ -1 \ 1]. \end{aligned}$$

8. (C) Elaborați o rețea pentru pattern-urile liniar neseparabile din figura 4.21 folosind programul scris la exercițiul 6.

(a) Selectați un număr potrivit de neuroni în stratul ascuns. Folosiți $K =$

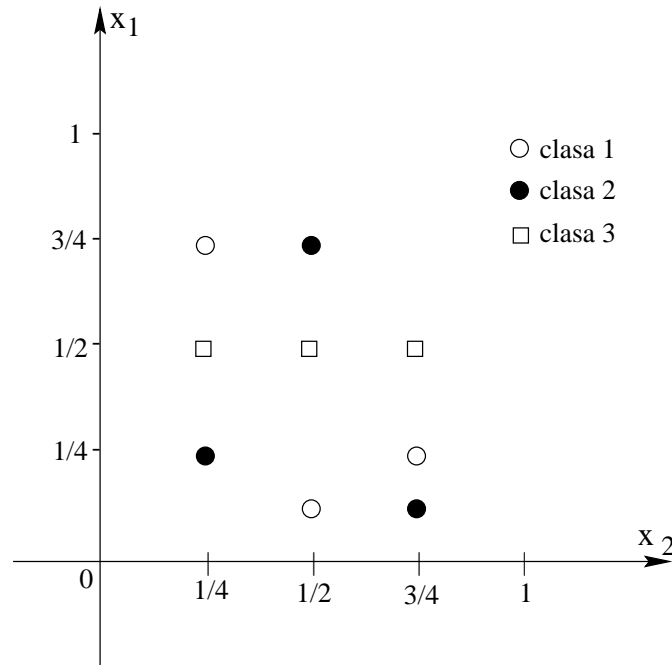


Figura 4.21: Pattern-uri neseperabile pentru exercițiul 8.

3, astfel încât ieșirile să fie:

$$\begin{aligned} \text{pentru clasa 1: } & [1 \quad -1 \quad -1]^t \\ \text{pentru clasa 2: } & [-1 \quad 1 \quad -1]^t \\ \text{pentru clasa 3: } & [-1 \quad -1 \quad 1]^t \end{aligned}$$

(b) Instruiți rețeaua pentru vectorii din figură și diferite valori pentru η .

9. (C) Fie $h(x) = 0,8 \sin \pi x$, $-1 \leq x \leq 1$ și aproximarea ei printr-o rețea neurală $o(x)$. Construiți o rețea de perceptroni bipolari de tip continuu cu un strat ascuns care aproximează $h(x)$ prin $o(x)$.
10. (C) Realizați un sistem expert conexiunist care să vă ajute la planificarea timpului vostru liber. În funcție de factorii de circumstanță, sistemul expert vă va spune ce activitate să alegeți. Activitățile posibile sunt:
- (a) vizită
 - (b) sport
 - (c) film
 - (d) somn
 - (e) cumpărături

Factorii care determină alegerea activității sunt:

- (a) resursele financiare disponibile
- (b) starea vremii
- (c) durata de timp care poate fi afectată
- (d) existența unui prieten care poate fi vizitat
- (e) starea de oboseală
- (f) necesitatea de a face cumpărături
- (g) ultima activitate făcută în timpul liber
- (h) existența unui film bun

Sistemul va fi realizat ca o rețea de perceptroni. Propuneți o secvență de 20 de perechi de forma (activități, factori de decizie) ca secvență de instruire, cu cel puțin 2 perechi pentru fiecare activitate din listă. Codificați intrările ca variabile mixte (continue și binare) din intervalul $[0, 1]$ și atribuiți 0 intrărilor care nu au importanță. Instruiți sistemul și, apoi, testați-l cu date de intrare diferite de cele folosite pentru instruire.

11. (C) Scrieți un program care implementează o rețea de perceptroni capabilă să clasifice cifrele 0–9 în categoriile par – impar. Pentru instruirea rețelei, folosiți o secvență de instruire în care caracterele sunt codificate ca matrici binare de 5×3 pixeli. După instruire, utilizați rețeaua pentru clasificarea pattern-urilor perturbate.

Capitolul 5

Rețele neurale feedback monostrat

Rețelele neurale pe care le vom studia în acest capitol sunt sisteme dinamice care evoluează în timp, într-un spațiu discret sau continuu. Evoluția unei astfel de rețele este, așa cum vom vedea, disipativ, în sensul că are tendința de scădere a energiei. Tranziția într-o rețea neurală dinamică este către o soluție asimptotic stabilă care este un minim local al unei funcții a energiei disipate.

Rețelele discutate în acest capitol se bazează în special pe lucrările lui Hopfield (1984, 1985, 1986).

Să menționăm câteva concepte fundamentale ale sistemelor dinamice.

Am văzut că un atractor este o stare către care sistemul evoluează în timp pornind de la anumite condiții inițiale. Mulțimea acestor condiții formează *bazinul de atracție* al atractorului. Dacă atractorul este un punct unic în spațiul stărilor, atunci el este un *punct fix*. Un atractor poate însă consta și dintr-o secvență de stări, caz în care se numește *ciclu limită*.

O rețea de acest tip este capabilă să tolereze anumite distorsiuni ale vectorilor de inițializare și să le elimine.

5.1 Fundamentele matematice ale rețelelor Hopfield cu timp discret

Acest tip de rețele au proprietăți foarte interesante. Utilizând tehnologii microelectronice și optoelectronice, s-au fabricat microsisteme care se bazează pe aceste rețele.

Conform postulatelor lui Hopfield, o rețea feedback monostrat este de forma celei din figura 5.1.

Această rețea constă din n neuroni cu pragurile T_1, \dots, T_n . Pentru fiecare neuron se calculează:

$$\text{net}_i = \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j + i_i - T_i, \text{ pentru } i = 1, 2, \dots, n,$$

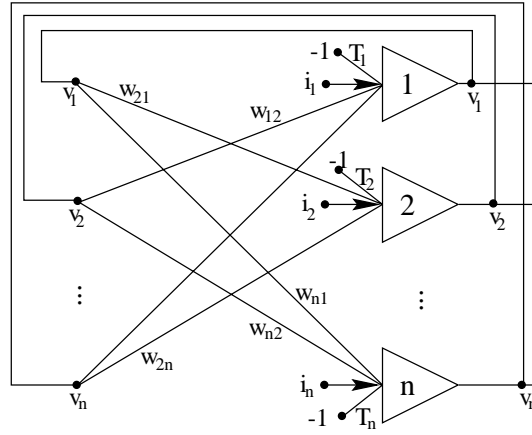


Figura 5.1: Rețea feedback monostrat.

unde i_i este intrarea externă a neuronului i . Vectorial, putem scrie:

$$net_i = \mathbf{w}_i^t + i_i - T_i, \text{ pentru } i = 1, 2, \dots, n,$$

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 relația de mai sus astfel:

$$\mathbf{net} = \mathbf{W}\mathbf{v} + \mathbf{i} - \mathbf{T},$$

unde

$$\mathbf{net} = \begin{bmatrix} net_1 \\ \vdots \\ net_n \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} i_1 \\ \vdots \\ i_n \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} T_1 \\ \vdots \\ T_n \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^t \\ \vdots \\ \mathbf{w}_n^t \end{bmatrix} = \begin{bmatrix} 0 & w_{12} & \dots & w_{1n} \\ w_{21} & 0 & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & 0 \end{bmatrix}.$$

În acest model, matricea \mathbf{W} este simetrică.

Să presupunem pentru moment că funcția de activare este $\text{sgn}(\cdot)$. Atunci:

$$\begin{cases} v_i \text{ rămâne neschimbat dacă } net_i = 0 \\ v_i \leftarrow -1 \text{ dacă } net_i < 0 \\ v_i \leftarrow +1 \text{ dacă } net_i > 0 \end{cases} \quad (5.1)$$

Adică:

$$v_i^{k+1} = \text{sgn}(\mathbf{w}_i^t \mathbf{v}^k + i_i - T_i), \quad i = 1, 2, \dots, n, \quad k = 0, 1, \dots$$

Regula se aplică *asincron*, adică pentru câte o singură valoare a lui i la un moment dat. Se pornește 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 ceilalți neuroni, ieșirile rămân aceleași. 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 rețelelor Hopfield. Trebuie să facem distincție între algoritmul asincron și cel sincron (paralel) descris de formula:

$$\mathbf{v}^{k+1} = \Gamma[\mathbf{W}\mathbf{v}^k + \mathbf{i} - \mathbf{T}], \quad k = 0, 1, \dots$$

Conform acestei formule, toți neuronii își pot modifica simultan ieșirile. 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 $k = 1, 2, \dots$, avem n actualizări aleatoare individuale, la nivel de neuron. Algoritmul se oprește când $\mathbf{v}^{k+1} = \mathbf{v}^k$.

Reprezentând vectorul de ieșire în $\{-1, 1\}^n$, acesta se mișcă dintr-un vârf al cubului n -dimensional *într-un vârf adiacent*, până când se stabilizează într-unul din vârfuri. Poziția finală a lui \mathbf{v}^k , pentru $k \rightarrow \infty$, este determinată de ponderile inițiale, pragurile inițiale, intrări, \mathbf{v}^0 , dar și de ordinea tranzițiilor.

Următoarele probleme sunt deschise: dacă sistemul are atractori și dacă sistemul se stabilizează.

Pentru a evalua proprietatea de stabilitate, vom defini *funcția energiei computaționale*:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \mathbf{T}^t\mathbf{v}$$

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 ținut 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 funcția de activare $\text{sgn}(\cdot)$, din relația 5.1 avem:

$$\begin{aligned}\text{dacă } net_i < 0 &\Rightarrow \Delta v_i \leq 0 \\ \text{dacă } net_i > 0 &\Rightarrow \Delta v_i \geq 0 \\ \text{dacă } net_i = 0 &\Rightarrow \Delta v_i = 0.\end{aligned}$$

Adică, avem mereu $net_i \Delta v_i \geq 0$ și $\Delta E \leq 0$. Aceasta înseamnă că energia rețelei poate doar să scadă sau să rămână constantă. Presupunând că $net_i \neq 0$, atunci $\Delta E = 0$ dacă și numai dacă $\Delta v_i = 0$. Cu alte cuvinte, E scade strict dacă și numai dacă $\Delta v_i \neq 0$ și E rămâne constantă dacă și 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:

$$\mathbf{v}^{k+1} = \lim_{k \rightarrow \infty} \text{sgn}(\mathbf{W}\mathbf{v}^k + \mathbf{i} - \mathbf{T}).$$

Am arătat că energia este necrescătoare. Ea este, în mod evident, și mărginită, ceea ce înseamnă că E își atinge minimumul.

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.

$$\begin{aligned}\mathbf{W} &= \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} & \mathbf{v}^0 &= \begin{bmatrix} -1 & -1 \end{bmatrix}^t \\ \mathbf{v}^1 &= \Gamma \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} \text{sgn}(1) \\ \text{sgn}(1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \mathbf{v}^2 &= \begin{bmatrix} -1 \\ -1 \end{bmatrix} \text{ etc.}\end{aligned}$$

Vom vedea că pattern-urile complementare corespund unor nivele identice ale energiei.

5.2 Fundamentele matematice ale rețelelor Hopfield cu timp continuu

Rețelele feedback monostrat cu timp continuu se numesc și *rețele de tip gradient*. O rețea de tip gradient converge către unul dintre minimele stabile din spațiul

stărilor. Evoluția sistemului este în direcția generală a gradientului negativ al funcției de energie. Căutarea unui minim energetic corespunde căutării unei soluții pentru o problemă de optimizare. Rețelele de tip gradient sunt exemple de sisteme neliniare, dinamice și asimptotic stabile.

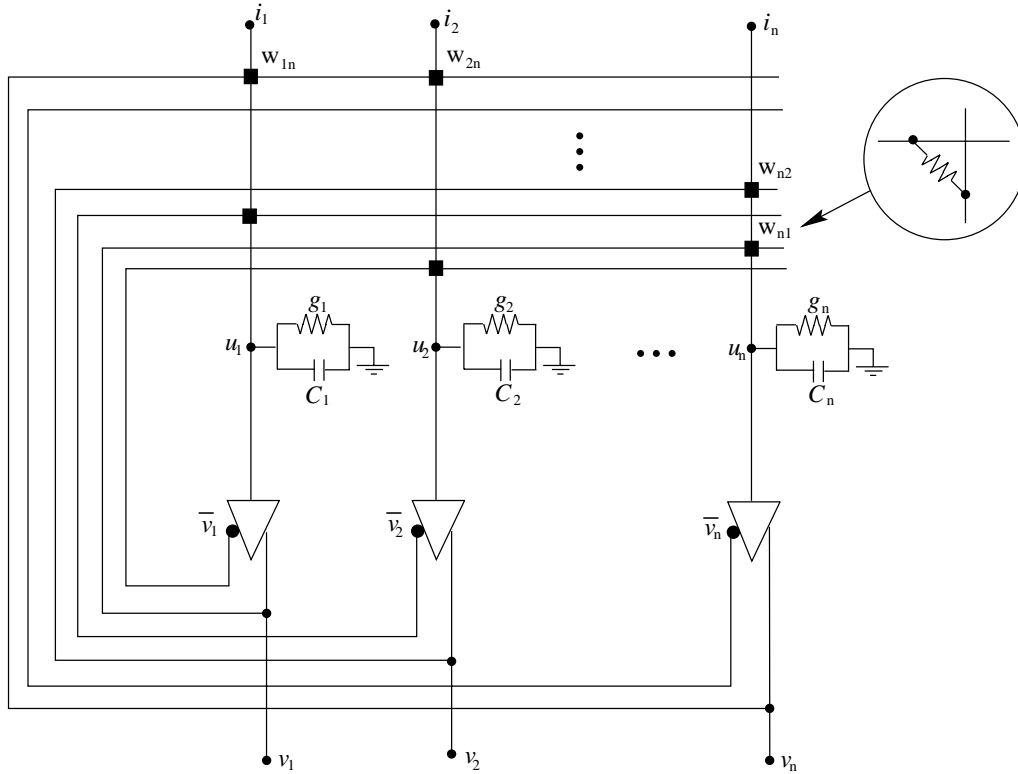


Figura 5.2: Rețea neurală de tip gradient care folosește componente electrice.

Vom exemplifica un model de rețea neurală de tip gradient prin folosirea unor componente electrice (fig. 5.2). Pentru a păstra notația originală a lui Hopfield, tensiunile u_i sunt de fapt net_i . Ponderile w_{ij} sunt conductanțe. Ieșirile inversate \bar{v}_i sunt folosite pentru a modela conductanțe negative, adică inhibitorii. Avem:

$$w_{ij} = w_{ji} \text{ și } w_{ii} = 0, \text{ pentru } i, j = 1, \dots, n.$$

Neuronii pot fi interpretați ca amplificatori: curentul prin conductanța g_i este amplificat de neuronul i (fig. 5.3). $C_i, i = 1, \dots, n$ sunt capacități.

Folosind acest model de neuron, putem scrie legea lui Kirchhoff:

$$i_i + \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j - u_i \left(\sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} + g_i \right) = C_i \frac{du_i}{dt}. \quad (5.2)$$

Partea stângă a acestei ecuații reprezintă curentul total care intră în condensatorul de capacitate C_i . Notăm conductanța totală care este conectată la in-

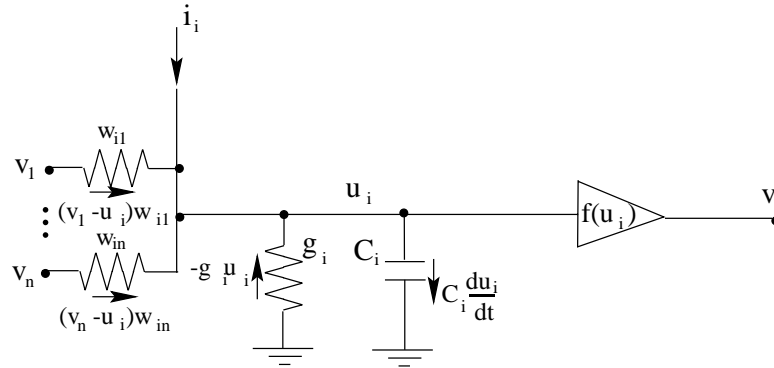


Figura 5.3: Modelul unui neuron.

trarea neuronului i cu

$$G_i = \sum_{j=1}^n w_{ij} + g_i.$$

Atunci, ecuația 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:

$$\mathbf{C} = \text{diag} [C_1 \ C_2 \ \dots \ C_n], \quad \mathbf{G} = \text{diag} [G_1 \ G_2 \ \dots \ G_n],$$

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_n(t) \end{bmatrix}, \quad \mathbf{v}(t) = \begin{bmatrix} v_1(t) \\ \vdots \\ v_n(t) \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} i_1 \\ \vdots \\ i_n \end{bmatrix},$$

unde $\mathbf{u}(t)$ este vectorul stărilor, iar $\mathbf{v}(t)$ reprezintă vectorul ieșirilor. Avem:

$$\begin{cases} C \frac{d\mathbf{u}(t)}{dt} = \mathbf{W}\mathbf{v}(t) - \mathbf{G}\mathbf{u}(t) + \mathbf{i} & \text{ecuația stărilor} \\ \mathbf{v}(t) = f(\mathbf{u}(t)) & \text{ecuația ieșirilor} \end{cases}$$

Să studiem stabilitatea sistemului descris de aceste ecuații. Funcția 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 funcției de activare f_i . Această formulă conține termenul $\mathbf{T}^t \mathbf{v}$ din funcția 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 și dispare pentru $\lambda \rightarrow \infty$. 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:

$$\begin{aligned} \frac{dE[\mathbf{v}(t)]}{dt} &= \nabla E^t(\mathbf{v}) \dot{\mathbf{v}} = (-\mathbf{W}\mathbf{v} - \mathbf{i} + \mathbf{G}\mathbf{u})^t \dot{\mathbf{v}} = - \left(C \frac{d\mathbf{u}}{dt} \right)^t \frac{d\mathbf{v}}{dt} \\ &= - \sum_i C_i \frac{df_i^{-1}(v_i)}{dv_i} \left(\frac{dv_i}{dt} \right)^2 \end{aligned}$$

deoarece

$$\frac{du_i}{dt} = \frac{df_i^{-1}(v_i)}{dv_i} \frac{dv_i}{dt}.$$

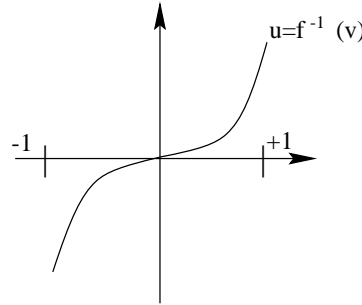


Figura 5.4: Inversa funcției de activare.

În figura 5.4, funcția f_i^{-1} este monoton crescătoare, deci $\frac{df_i^{-1}(v_i)}{dv_i} > 0$. Atunci:

$$\frac{dE}{dt} < 0 \text{ pentru } \frac{d\mathbf{v}}{dt} \neq 0 \text{ și}$$

$$\frac{dE}{dt} > 0 \text{ pentru } \frac{d\mathbf{v}}{dt} = 0.$$

Am găsit o funcție Liapunov a sistemului, deci sistemul este asimptotic stabil. Funcția E este mărginită în spațiul ieșirilor, deci tinde către un minim.

Modificările în timp nu sunt exact pe direcția gradientului funcției de energie (adică pe direcția scăderii cât mai rapide a energiei).

Rețeaua de tip gradient, în cazul în care spațiul ieșirilor este mărginit, converge către unul dintre minimele lui $E(\mathbf{v})$, deci către o stare stabilă. Acest minim este în interiorul hipercubului ieșirilor. Dacă $\lambda \rightarrow \infty$, atunci minimul este într-un vârf al hipercubului.

Faptul că sistemul este asimptotic stabil în spațiul ieșirilor este echivalent cu faptul că sistemul este asimptotic stabil în spațiul stărilor \mathbf{u} . Aceasta, deoarece ecuația

$$\mathbf{v}(t) = f(\mathbf{u}(t))$$

este monotonă.

5.3 Aplicație: problema comis-voiajorului

Vom da o soluție a problemei comis-voiajorului pentru n orașe folosind rețele 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 rețea 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 poziția în ciclu.

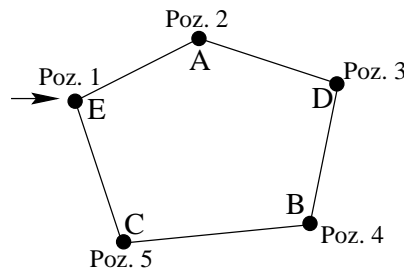


Figura 5.5: Reprezentarea cu ajutorul unui graf a problemei comis-voiajorului.

Fie orașele A, B, ..., E și fie acest ciclu. Îl reprezentăm prin tabelul 5.1, unde v_{xi} este ieșirea neuronului corespunzător orașului x și poziției i în ciclu.

	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
		1	2	3	4	5
orașul		poziția				

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 și un singur 1 pe fiecare linie. Poziționăm neuronii în acest tablou de $n \times n$ elemente, în fiecare element punând un neuron. Vom avea exact n

neuroni activi pentru soluție. 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 funcția 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 și pe coloană. În plus, trebuie să eliminăm soluțiile triviale, pentru care avem, în tablou, linii sau coloane nule. Să exprimăm aceste condiții de optimizare folosind patru funcții de energie, fiecare îndeplinind un rol separat.

$$E_1 = A \sum_{x=1}^n \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n v_{xi} v_{xj}.$$

Produsul $v_{xi} v_{xj}$ se referă la același oraș, x . Relația

$$\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n v_{xi} v_{xj} = 0$$

se îndeplinește dacă cel mult un $v_{xi} = 1$ iar restul sunt 0, adică dacă orașul x apare cel mult o singură dată pe circuit. $E_1 = 0$ înseamnă că fiecare oraș este vizitat cel mult o dată. Dacă $E_1 > 0$, atunci există orașe 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 poziție din ciclu are cel mult un singur oraș asociat ei. Dacă $E_2 > 0$, atunci există mai multe orașe vizitate simultan. Observăm că E_1 și E_2 sunt, de asemenea, nule pentru soluții triviale în care avem linii sau coloane în tablou care nu conțin 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 soluție trivială, deci o soluție cu mai puțin 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 și pe fiecare coloană. Dorim să minimizăm lungimea ciclului. Notăm cu d_{xy} distanța dintre x și y și $d_{xy} = d_{yx}$. 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 funcție 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 $v_{x,n+1} = v_{x1}$ și $v_{x0} = v_{xn}$.

Dorim să minimizăm $E_1 + E_2 + E_3 + E_4$. Prin identificarea termenilor acestei sume cu cei din E , obținem:

$$w_{xi,yj} = -2A\delta_{xy}(1 - \delta_{ij}) - 2B\delta_{ij}(1 - \delta_{xy}) - 2C - 2Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1}),$$

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, \dots, E_4 . Ele dau ponderea fiecăruia dintre acești termeni. Obținem, de asemenea:

$$i_{xi} = 2Cn.$$

Am generat rețeaua. Mai trebuie acum să găsim soluția prin simularea funcționării ei.

Din formula generală a ecuației stărilor:

$$C_i \left(\frac{du_i}{dt} \right) = i_i + \sum_{j=1}^n w_{ij} v_j - u_i G_i$$

obținem:

$$\begin{aligned} \frac{du_{xi}}{dt} = & -\frac{u_{xi}}{\tau} - 2A \sum_{\substack{j=1 \\ j \neq i}}^n v_{xj} - 2B \sum_{\substack{y=1 \\ y \neq x}}^n v_{yi} \\ & - 2C \left(\sum_{x=1}^n \sum_{j=1}^n v_{xj} - n \right) - 2D \sum_{\substack{y=1 \\ y \neq x}}^n d_{xy} (v_{y,i+1} + v_{y,i-1}), \end{aligned}$$

unde τ este o constantă a timpului.

S-au luat $\tau = 1$, $\lambda = 50$, $A = B = D = 250$, $C = 100$, $10 \leq n \leq 30$ și s-a rezolvat numeric acest sistem de ecuații. Nu se ajunge neapărat la soluția optimă. Pentru $n = 10$, 50% din teste au produs cele mai scurte cicluri din 181440 de soluții posibile. Pentru valori mari ale lui n apar, însă, dificultăți.

Aceasta duce la întrebarea fundamentală dacă rețelele neurale sunt mai adecvate decât calculatoarele clasice pentru rezolvarea problemelor nedeterminist polinomiale.

S-a demonstrat (Bruck, Goodman, 1988) că nu există nici o rețea 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 rețelelor Hopfield în probleme de optimizare. S-au obținut rezultate în probleme de alocare a resurselor, planificarea lucrărilor, optimizarea traficului, optimizarea conexiunilor în circuitele integrate (1990), programare liniară și neliniară.

În concluzie, putem spune că una dintre limitările rețelelor Hopfield este faptul că ajungem de multe ori la un minim local și nu global. Aceasta se datorează formei complexe a funcției $E(\mathbf{v})$.

Una din dificultăți este și traducerea problemei de optimizare într-o problemă de minimizare a energiei. Singurele forme ale funcției de energie care se cunosc sunt cele date de Hopfield. Dacă λ este mare, se poate lua energia trunchiată. Dacă intrările externe sunt nule, dispăre și al doilea termen. Schema de rezolvare a unei probleme de optimizare folosind rețele feed-back monostrat este cea din figura 5.6.

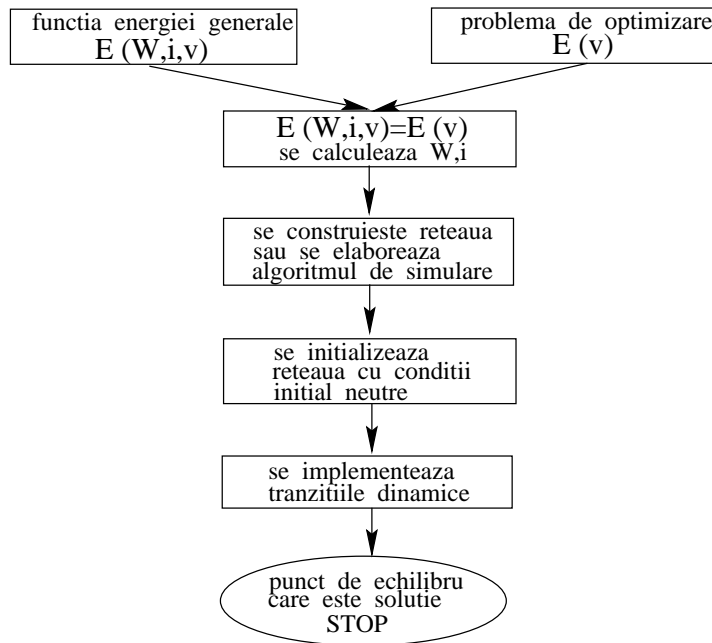


Figura 5.6: Modul de rezolvare a unei probleme de optimizare folosind rețele feed-back monostrat.

5.4 Exemple

Rețea 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 și 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: $\begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}^t, \dots, \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^t$ se ajunge la nivelele de energie -6, 0, 2. Tranzițiile au loc pe muchiile hipercubului, de la un vârf la un alt vârf cu energie mai scăzută. Minimum este atins în vârfurile $\begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^t$ și $\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^t$, unde avem stări stabile. Tranzițiile au loc asincron. De aceea, o tranziție este doar de-a lungul unei singure muchii (se poate modifica cel mult o componentă).

Dacă tranzițiile au loc sincron, atunci ele nu minimizează energia și nu tind către o stare stabilă. De exemplu, pornind din $\mathbf{v}^0 = \begin{bmatrix} 1 & -1 & 1 & 1 \end{bmatrix}^t$, ajungem la:

$$\begin{aligned} \mathbf{v}^1 &= \Gamma[\mathbf{W}\mathbf{v}^0] = \Gamma \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \text{sgn}(-1) \\ \text{sgn}(1) \\ \text{sgn}(-1) \\ \text{sgn}(-1) \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix}. \end{aligned}$$

Tranziția nu mai este de-a lungul unei muchii. Se calculează că

$$\mathbf{v}^2 = \mathbf{v}^0,$$

deci oscilează între \mathbf{v}^0 și \mathbf{v}^1 . În \mathbf{v}^0 și \mathbf{v}^1 energia este 2. Spre deosebire de tranzițiile sincrone, tranzițiile asincrone duc întotdeauna la unul dintre minimele energetice. Dacă la tranzițiile asincrone pornim de la $\begin{bmatrix} -1 & 1 & 1 & 1 \end{bmatrix}$ și scădem energia, atunci evoluția se poate desfășura ca în figura 5.7.

Minimum la care ajungem depinde de ordinea de actualizare a componentelor.

Rețea de tip gradient

Vom elabora un convertor A/D (Tank, Hopfield, 1986) pe 2 biți. Presupunem că $f(u_i)$ este continuă unipolară.

Se poate arăta că o funcție energiei totale aleasă convenabil poate fi:

$$E = \frac{1}{2} \left(x - \sum_{i=0}^1 v_i 2^i \right)^2 - \frac{1}{2} \sum_{i=0}^1 2^{2i} v_i (v_i - 1).$$

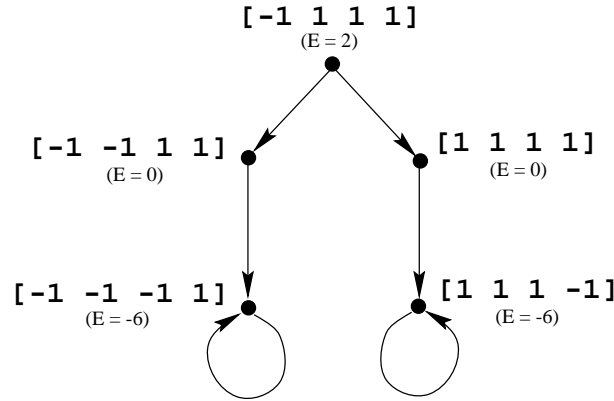


Figura 5.7: Variante de evoluție a stărilor pentru o serie de tranziții asincrone.

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 $\begin{bmatrix} v_1 & v_0 \end{bmatrix}^t$ și 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 rețeaua 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 rețele 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 ecuațiile stărilor. De exemplu, putem presupune că folosim o tensiune de referință având valoarea de -1V (fig. 5.9).

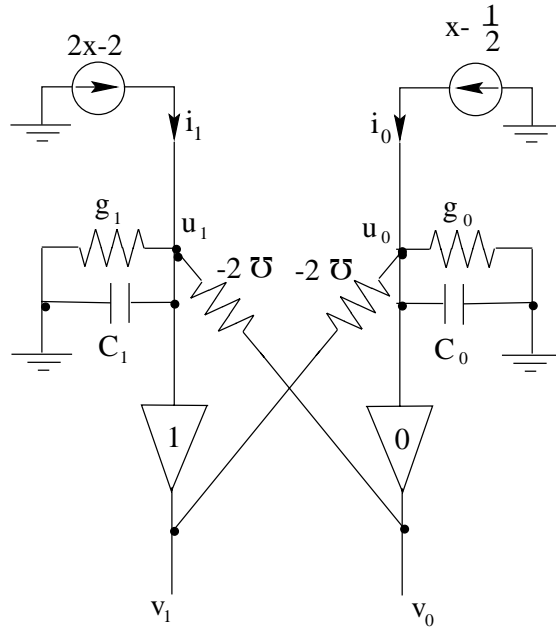


Figura 5.8: Convertor A/D pe doi biți implementat printr-o rețea de tip gradient.

Ecuatiile stărilor devin:

$$\begin{aligned} C_0 \frac{du_0}{dt} &= x - \frac{1}{2} + 2\bar{v}_1 - u_0(2 + g_0) \\ C_1 \frac{du_1}{dt} &= 2x - 2 + 2\bar{v}_0 - u_1(2 + g_1). \end{aligned}$$

Condiția $w_{ij} \geq 0$ asigură realizabilitatea rețelei folosind rezistențe de valoare $R_{ij} = \frac{1}{w_{ij}}$. Înlocuind ponderile și valorile curenților externi în expresia energiei, obținem pentru valori mari ale lui λ :

$$E = 2v_0v_1 + \frac{v_0}{2} + 2v_1 - x(v_0 + 2v_1). \quad (5.3)$$

Reprezentând grafic funcția E , obținem:

- Pentru $x = 0$, punctul staționar este punct de minim: $v_0 = v_1 = 0$
- Pentru $x = 1$, punctul staționar este punct de minim: $v_0 = 1, v_1 = 0$
- Pentru $x = 1,25$ punctele staționare sunt: punctul de minim dorit $v_0 = 1, v_1 = 0$, dar și punctul de minim nedorit $v_0 = 0, v_1 = 1$, precum și un punct șa, undeva în interiorul pătratului: $0 < v_0 < 1, 0 < v_1 < 1$.

În loc să rezolvăm sistemul de ecuații diferențiale, oare nu putem căuta minimele lui E ?

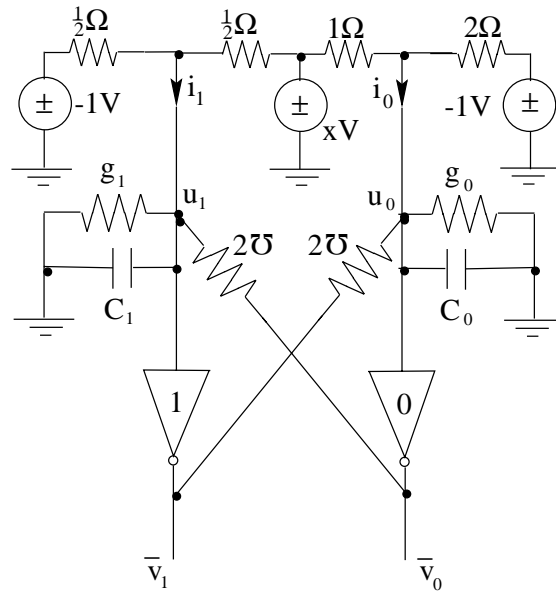


Figura 5.9: Convertor A/D pe doi biți implementat printr-o rețea de tip gradient și care folosește o tensiune de referință de 1V.

Din relația 5.3 obținem:

$$\nabla E(\mathbf{v}) = \begin{bmatrix} 2v_1 + \frac{1}{2} - x \\ 2v_0 + 2 - 2x \end{bmatrix}$$

$$\mathbf{H} = \nabla^2 E(\mathbf{v}) = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}.$$

Observăm :

$$\det H_{11} = 0$$

$$\det H_{22} = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} = -4.$$

Matricea H nu este pozitiv sau negativ definită. E nu are deci minim sau maxim, dacă nu considerăm alte restricții. Impunem restricția ca spațiul de ieșire să fie pătratul $(0 \ 1)^2$. Atunci, minimele lui E sunt situate în vârfurile pătratului, dacă aceste minime există. Se poate demonstra acest lucru pentru $\lambda \rightarrow \infty$. Dacă v^* este o soluție pentru $\nabla E(\mathbf{v}) = 0$ și se află în interiorul pătratului, atunci este un punct șa. Rezultă că pentru a găsi soluția avem două posibilități:

1. Rezolvarea numerică a sistemului de ecuații diferențiale
2. Simularea rețelei printr-un program de simulare a circuitelor electronice, de exemplu SPICE.

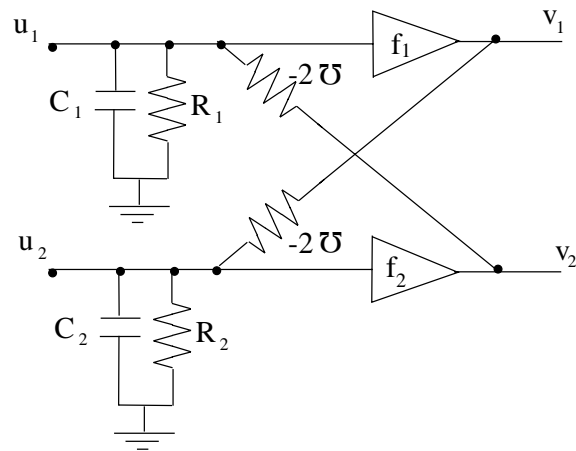


Figura 5.10: Rețeaua neurală propusă pentru exercițiul 2.

5.5 Exerciții

1. Simulați în SPICE convertorul A/D. Generalizați-l pentru patru biți.
2. Pentru rețeaua neurală din figura 5.10, cu λ mare, găsiți:
 - (a) Ecuațiile stărilor;
 - (b) Ponderile (conductanțele) \mathbf{W} ;
 - (c) Funcția energiei $E(\mathbf{v})$ trunchiată;
 - (d) $\nabla E(\mathbf{v})$.

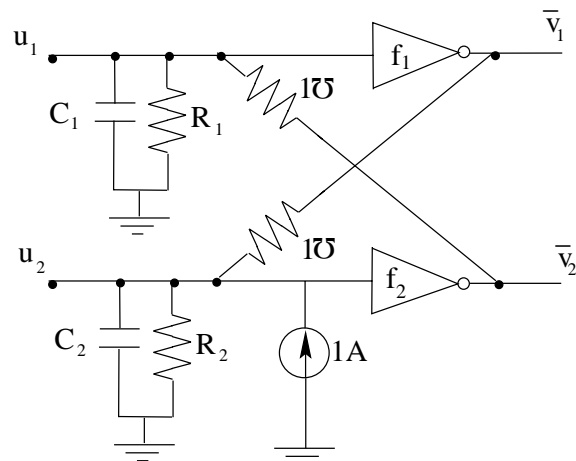


Figura 5.11: Rețeaua neurală propusă pentru exercițiul 3.

3. Pentru rețeaua neurală din figura 5.11, cu λ mare, găsiți:

- Ecuatiile stărilor;
- Ponderile (conductanțele) \mathbf{W} ;
- Vectorul \mathbf{i} al intrărilor externe;
- Funcția energiei $E(\mathbf{v})$ trunchiată;
- $\nabla E(\mathbf{v})$.

4. Fie o rețea neurală cu trei neuroni, a cărei funcție de energie trunchiată este:

$$E(\mathbf{v}) = v_1^2 - v_2^2 + v_3^2 - 2v_1v_3 - v_1v_3 + 4v_1 + 12.$$

Găsiți:

- $\nabla E(\mathbf{v})$;
- $\nabla^2 E(\mathbf{v})$;
- Minimele, maximele și punctele șa ale lui $E(\mathbf{v})$, atunci când nu avem alte restricții.

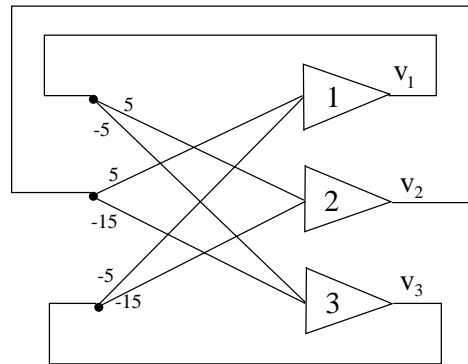


Figura 5.12: Rețeaua neurală propusă pentru exercițiul 5.

5. Fie rețeaua neurală din figura 5.12, cu λ mare. Evaluând funcția de energie trunchiată, găsiți minimele, maximele și punctele șa în $[-1 \ 1]^3$. Calculați apoi valorile energiei în toate vârfurile cubului.

6. O rețea neurală este descrisă de ecuațiile:

$$\begin{cases} \frac{du_i}{dt} = \frac{1}{C_i} \left(\sum_{j=1}^n w_{ij}v_j - u_iG_i + i_i \right) & \text{pentru } i = 1, 2, \dots, n \\ v_i = f(u_i) \end{cases}$$

Rezolvați numeric acest sistem.

Indicație: Alegem, de exemplu, o metodă foarte simplă, cea a lui Euler:

$$\begin{cases} u_i^{k+1} = u_i^k + h \frac{1}{C_i} \left(\sum_{j=1}^n w_{ij} v_j^k - u_i^k G_i + i_i \right) & \text{pentru } i = 1, \dots, n, \\ \quad \text{unde } h = t^{k+1} - t^k \\ v_i^k = f(u_i^k) \end{cases}$$

7. Implementați metoda din exercițiul 6 pentru problema convertorului A/D pe doi biți și găsiți $v(t)$ (ieșirea stabilă) pentru:

$$C_0 = C_1 = 1F, \quad g_0 = g_1 = 1\Omega^{-1}, \quad x = 0,$$

$$v_i = f(u_i) = \frac{1}{1 + e^{-10u_i}}, \quad i = 0, 1$$

Testați 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) Proiectați o memorie Hopfield recurentă autoasociativă care să memoreze ca prototipuri caracterele A, I și O, codificate ca matrici binare de 7×5 pixeli. Implementați algoritmul de regăsire și simulați convergența asincronă a memoriei către fiecare din prototipurile memorate, folosind ca pattern-uri cheie caracterele A, I și 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 evoluția unui sistem dinamic ca o mișcare a unui pattern de intrare către un alt pattern care este memorat și se numește *prototip* sau *memorie stocată*. Rețelele 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 și *argument al căutării*) care conține o porțiune de informație despre un anumit pattern dintr-o mulțime de pattern-uri memorate. Acest prototip anume poate fi regăsit prin asocierea pattern-ului cheie și a informației memorate.

S-au elaborat mai multe modele de memorii asociative. Ne vom concentra asupra rețelelor feedback din capitolul precedent, dar vom discuta și arhitecturi feedforward de memorie.

În general, memoriile asociative achiziționează informație *a priori*. Scrierea în memorie provoacă modificări ale conexiunilor dintre neuroni. Nu există adrese, toată informația din memorie este distribuită spațial în rețea.

Care sunt cerințele 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 asociații, pe măsură ce se modifică cerințele 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 locații de memorie cu adrese; stocarea este distribuită într-o rețea densă de neuroni interconectați.

6.1 Concepte de bază

Diagrama bloc a unei memorii asociative este reprezentată în figura 6.1.

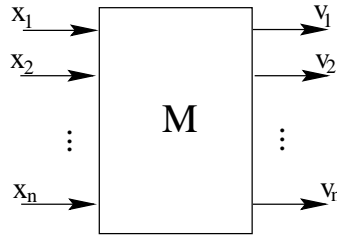


Figura 6.1: Diagrama bloc a unei memorii asociative.

În această figură, \mathbf{x} este vectorul intrărilor, \mathbf{v} este vectorul ieșirilor, $\mathbf{x} \in \mathfrak{R}^n$, $\mathbf{v} \in \mathfrak{R}^m$ și $\mathbf{v} = M[\mathbf{x}]$, unde M este un operator neliniar matricial specific fiecărui tip de memorie. Structura lui M reflectă o paradigmă specifică. Pentru memorii dinamice, M implică și variabila timp.

Pentru un model de memorie dat, forma operatorului M este exprimată de obicei în funcție de vectorii prototip care trebuie stocați. Algoritm care permite calcularea lui M se numește *algoritm de înregistrare* sau *stocare*. De obicei, neuronii sunt așezați 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 stocați anumiți vectori prototip, astfel încât, dacă se aplică un pattern cheie, ieșirea produsă de memorie și asociată cheii este răspunsul memoriei.

Presupunând că există p perechi de asocieri stocate:

$$\mathbf{x}^{(i)} \rightarrow \mathbf{v}^{(i)} \quad \text{pentru } i = 1, \dots, p$$

și

$$\mathbf{v}^{(i)} \neq \mathbf{x}^{(i)} \quad \text{pentru } i = 1, \dots, p,$$

atunci rețeaua este o *memorie heteroasociativă*.

Dacă aplicația este de forma:

$$\mathbf{x}^{(i)} \rightarrow \mathbf{v}^{(i)} \Big|_{\mathbf{v}^{(i)} = \mathbf{x}^{(i)}} \quad \text{pentru } i = 1, \dots, p$$

atunci memoria este *autoasociativă*.

În memoria heteroasociativă, asocierile se fac între mulțimile ordonate de vectori $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(p)}\}$ și $\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \dots, \mathbf{v}^{(p)}\}$. Memoriile autoasociative asociază vectori din cadrul unei singure mulțimi care este $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(p)}\}$. Evident, proiectarea unui vector $\mathbf{x}^{(i)}$ în el însuși nu are nici o semnificație. O aplicație 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 conținut*.

Prin rețele neurale se pot realiza memorii statice (instantanee, nerecurente) și memorii dinamice (recurente). Memoria din figura 6.2 este una statică realizată



Figura 6.2: Memorie statică realizată printr-o rețea feedforward.

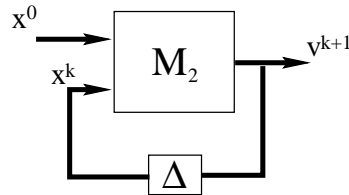


Figura 6.3: Memorie dinamică autoasociativă realizată printr-o rețea recurentă.

printr-o rețea feedforward și $\mathbf{v}^0 = M_1[\mathbf{x}^0]$ care reprezintă un sistem de ecuații algebrice neliniare. În figura 6.3 este o memorie dinamică autoasociativă realizată printr-o rețea recurentă pentru care $\mathbf{v}^{k+1} = M_2[\mathbf{x}^0, \mathbf{v}^k]$, reprezentând un sistem de ecuații diferențiale neliniare.

Un **exemplu de memorie dinamică** este rețeaua 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).

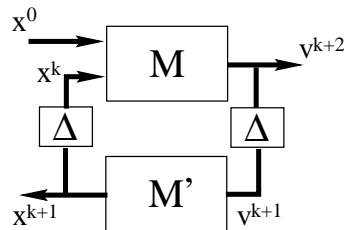


Figura 6.4: Memorie dinamică heteroasociativă.

6.2 Asociatori liniari

Memoriile asociative tradiționale 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 ieșire. Nu apar neuroni. Dacă totuși 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 ieșire 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 $\{\mathbf{s}^{(i)}, \mathbf{f}^{(i)}\}$, $i = 1, \dots, p$. Vectorii \mathbf{s} reprezintă stimulii, iar vectorii \mathbf{f} sunt răspunsurile forțate. Avem:

$$\mathbf{s}^{(i)} = [s_1^{(i)} \dots s_n^{(i)}]^t$$

$$\mathbf{f}^{(i)} = [f_1^{(i)} \dots f_m^{(i)}]^t.$$

Practic, $\mathbf{s}^{(i)}$ pot fi pattern-uri iar $\mathbf{f}^{(i)}$ informații asupra apartenenței 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

$$\mathbf{s}^{(i)} \rightarrow \mathbf{f}^{(i)} + \eta^{(i)} \quad \text{pentru } i = 1, 2, \dots, p$$

astfel încât vectorul zgomot $\eta^{(i)}$ să fie minimizat.

Problema se poate pune și astfel: pentru un număr mare de observații, să se găsească \mathbf{W} care minimizează $\sum_i \|\eta^{(i)}\|$. Este o problemă tipică de statistică matematică și nu o vom aborda aici.

Dorim să găsim \mathbf{W} care permite stocarea eficientă a datelor în memorie. Aplicăm regula de învățare hebbiană pentru a instrui asociatorul:

$$w_{ij}' = w_{ij} + f_i s_j \quad \text{pentru } i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n$$

unde vectorii \mathbf{f} și \mathbf{s} trebuie să fie membrii ai perechii de asociere.

$$\mathbf{W}' = \mathbf{W} + \mathbf{f}\mathbf{s}^t$$

Inițializând 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 învățat, 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 corelație $m \times n$ și

$$\mathbf{F} = \begin{bmatrix} \mathbf{f}^{(1)} & \mathbf{f}^{(2)} & \dots & \mathbf{f}^{(p)} \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}^{(1)} & \mathbf{s}^{(2)} & \dots & \mathbf{s}^{(p)} \end{bmatrix}.$$

Să presupunem că, după instruire, aplicăm vectorul cheie $\mathbf{s}^{(j)}$. Atunci:

$$\begin{aligned} \mathbf{v} &= \left(\sum_{i=1}^p \mathbf{f}^{(i)} \mathbf{s}^{(i)t} \right) \mathbf{s}^{(j)} \\ &= \mathbf{f}^{(1)} \mathbf{s}^{(1)t} \mathbf{s}^{(j)} + \dots + \mathbf{f}^{(p)} \mathbf{s}^{(p)t} \mathbf{s}^{(j)}. \end{aligned}$$

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, mulțimea de vectori ortonormali $\{\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}\}$ asigură maparea perfectă. Ortonormalitatea este o condiție supusă intrărilor și 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 perturbație $\Delta^{(j)}$ se poate presupune statistic independent de $\mathbf{s}^{(j)}$. Pentru cazul când vectorii stocați 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 ținut cont și de faptul că $\Delta^{(j)}$ este statistic independent de $\mathbf{s}^{(j)}$, deci pot fi considerați 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 perturbației $\Delta^{(j)}$. Această componentă conține în paranteză aproape toți termenii lui \mathbf{W} ponderați de $\Delta^{(j)}$. Chiar și în cazul în care vectorii memorati 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, rețeaua este un *autocorelator*. Luăm $\mathbf{f}^{(i)} = \mathbf{s}^{(i)}$ și obținem *matricea de autocorelație* \mathbf{W}' :

$$\mathbf{W}' = \sum_{i=1}^p \mathbf{s}^{(i)} \mathbf{s}^{(i)t} = \mathbf{S}\mathbf{S}^t,$$

ținând cont de faptul că \mathbf{W}' este simetrică. Să notăm însă că \mathbf{W}' nu are diagonala egală cu zero. Presupunând $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}$ ortonormali, pentru vectorul de intrare perturbat $\mathbf{s}^{(j)}$ obținem:

$$\begin{aligned} \mathbf{v} &= \mathbf{s}^{(j)} + \sum_{\substack{i=1 \\ i \neq j}}^p \mathbf{s}^{(i)} \mathbf{s}^{(i)t} \Delta^{(j)} \\ &= \mathbf{s}^{(j)} + (p-1)\Delta^{(j)} \end{aligned}$$

unde termenul $\Delta^{(j)}$ apare amplificat de $p-1$ ori!

Asociatorii liniari și autoasociatorii pot fi folosiți și atunci când vectorii $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}$ sunt liniar independenți (o condiție 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 $\mathbf{f}^{(j)}$ și $\mathbf{v}^{(j)}$.

Deoarece vectorii prototip nu sunt în general liniar independenți, 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 esență o rețea 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 alcătuiți dintr-un sumator și un comparator și sunt bipolari discreți. Ieșirile sunt în mulțimea $\{-1, 1\}^n$. Presupunem că $i_i = T_i = 0$, pentru $i = 1, 2, \dots, n$.

Algoritmul de regăsire

$$v_i^{k+1} = \text{sgn} \left(\sum_{j=1}^n w_{ij} v_j^k \right)$$

Presupunând că pornim din \mathbf{v}^0 și că se alege în mod aleator secvența m, p, q de neuroni, avem:

$$\begin{aligned} \text{Prima actualizare:} \quad \mathbf{v}^1 &= [v_1^0 \quad v_2^0 \quad \dots \quad v_m^1 \quad \dots \quad v_p^0 \quad \dots \quad v_q^0 \quad \dots \quad v_n^0]^t \\ \text{A doua actualizare:} \quad \mathbf{v}^2 &= [v_1^0 \quad v_2^0 \quad \dots \quad v_m^1 \quad \dots \quad v_p^2 \quad \dots \quad v_q^0 \quad \dots \quad v_n^0]^t \\ \text{A treia actualizare:} \quad \mathbf{v}^3 &= [v_1^0 \quad v_2^0 \quad \dots \quad v_m^1 \quad \dots \quad v_p^2 \quad \dots \quad v_q^3 \quad \dots \quad v_n^0]^t \end{aligned}$$

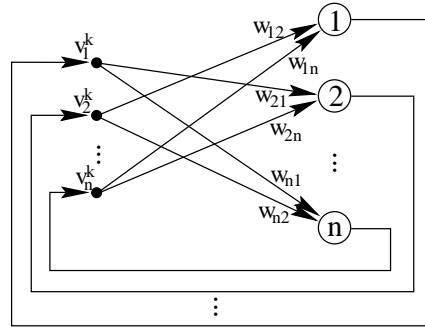


Figura 6.5: Memorie Hopfield autoasociativă.

Avem:

$$E(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v}.$$

Am văzut că energia este necrescătoare și rețeaua 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 aceeași valoare cu un minim pentru $E(-\mathbf{v})$. Cu alte cuvinte, tranzițiile memoriei se pot termina în aceeași măsură atât în \mathbf{v} , cât și în $-\mathbf{v}$. Factorul decisiv care determină convergența este *similaritatea* dintre \mathbf{v}^0 și \mathbf{v} , respectiv $-\mathbf{v}$.

Algoritmul de stocare

Fie $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}$ 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 $i \neq j$, $\delta_{ii} = 1$ și \mathbf{I} este matricea unitate.

Matricea \mathbf{W} este foarte similară cu matricea de autocorelație obținută prin învățare hebbiană în cazul asociatorului linear autoasociativ. Diferența este că $w_{ii} = 0$. Sistemul nu memorează vectorul $\mathbf{s}^{(m)}$, ci doar ponderile w_{ij} care reprezintă corelații între componentele acestui vector.

Oricând se pot adăuga noi autoasocieri adiționale 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 $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(m)}$, 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 operația de complementare binară. Stocând vectorul $\mathbf{s}'^{(m)}$, complementar lui $\mathbf{s}^{(m)}$, obținem:

$$w_{ij}' = (1 - \delta_{ij}) \sum_{m=1}^p (2s_i'^{(m)} - 1)(2s_j'^{(m)} - 1).$$

Substituind $s_i'^{(m)} = 1 - s_i^{(m)}$, obținem $w_{ij}' = w_{ij}$. Cu alte cuvinte, este indiferent dacă stocăm un pattern sau complementul său binar.

Algoritmul de stocare și regăsire pentru o memorie autoasociativă recurentă este următorul:

Se dau vectorii binar bipolarari $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}$ de câte n componente. Vectorul inițial (care se cere regăsit) este \mathbf{v}^0 , tot de n componente.

Stocarea

1. Se inițializează $\mathbf{W} \leftarrow 0$, $m \leftarrow 1$, 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 inițializează $k \leftarrow 1$, $\mathbf{v} \leftarrow \mathbf{v}^0$, unde k este un contor de ciclu.
2. Se inițializează $i \leftarrow 1$, contorul de actualizare în ciclu. Se obține o permutare aleatoare a întregilor $1, \dots, n$: $\alpha_1, \dots, \alpha_n$. actualizare $\leftarrow FALSE$.
3. Este actualizat neuronul α_i , dacă este cazul:
 $net_{\alpha_i} = \sum_{j=1}^n w_{\alpha_i j} v_j$, $vnew \leftarrow \text{sgn}(net_{\alpha_i})$
 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.

Considerații asupra modului de funcționare

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:

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^p s_i^{(m)} s_j^{(m)}$$

și, neglijând pentru moment termenul $1 - \delta_{ij}$, obținem:

$$\begin{aligned} net_i &\cong \sum_{j=1}^n \sum_{m=1}^p s_i^{(m)} s_j^{(m)} s_j^{(m')} \\ &= \sum_{m=1}^p s_i^{(m)} \sum_{j=1}^n s_j^{(m)} s_j^{(m')}. \end{aligned}$$

Dacă vectorii $\mathbf{s}^{(m)}$ și $\mathbf{s}^{(m')}$ sunt statistic independenți, 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 $\mathbf{s}^{(m)}$ și $\mathbf{s}^{(m')}$ sunt ortogonali, deci statistic independenți, acest produs devine zero. Dacă $\mathbf{s}^{(m)}$ și $\mathbf{s}^{(m')}$ 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

$$net_i \cong s_i^{(m')} n, \quad \text{pentru } i = 1, 2, \dots, n. \tag{6.2}$$

net_i are același semn ca și $s_i^{(m')}$, pentru $i = 1, \dots, n$. În concluzie, vectorul $\mathbf{s}^{(m')}$ este stabil și nu se mai poate modifica.

Vom da acum o explicație 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 perturbația 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 $\mathbf{s}^{(m')}$ și 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 inițial să fie neperturbate și ca n să fie mare. Componentele perturbate se conformează deci dorinței majorității.

Aceasta explică intuitiv cum un pattern perturbat poate fi asociat celui mai apropiat prototip stocat. Discuția 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 $i \neq j$ și $\mathbf{s}^{(i)t} \mathbf{s}^{(i)} = n$, obținem

$$\mathbf{net} = (n - p) \mathbf{s}^{(m')}.$$

Dacă $n > p$, rezultă că $\mathbf{s}^{(m')}$ este stabil. Funcția 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:

$$\begin{aligned} \mathbf{net} &= n\mathbf{s}^{(m')} - p\mathbf{s}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^p \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')} \\ &= (n - p) \mathbf{s}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^p \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')}. \end{aligned}$$

Față de situația anterioară, apare suplimentar ultimul termen:

$$\sum_{\substack{m=1 \\ m \neq m'}}^p \left(\mathbf{s}^{(m)} \mathbf{s}^{(m)t} \right) \mathbf{s}^{(m')}.$$

Acesta poate fi interpretat ca un vector *zgomot*. Cu cât $(n - p)$ este mai mare, cu atât scade importanța zgomotului și prototipurile devin stări de echilibru.

6.4 Analiza performanței memoriilor autoasociative recurente

În această secțiune, vom stabili relații între mărimea n a memoriei și numărul de pattern-uri distincte care pot fi refăcute în mod eficient. Aceasta depinde și de gradul de similaritate dintre vectorul cheie și cel mai apropiat prototip, precum și de gradul de similaritate dintre prototipuri.

Pentru a măsura "similaritatea", vom folosi *distanța Hamming*. De fapt, ea este proporțională cu "disimilaritatea". Pentru vectorii bipolari de n componente \mathbf{x} și \mathbf{y} , avem:

$$\text{DH}(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^n |x_i - y_i|.$$

Această înseamnă că $\text{DH}(\mathbf{x}, \mathbf{y})$ reprezintă numărul de poziții în care diferă biții. Prin actualizarea asincronă, la fiecare pas se modifică vectorul de ieșire cu $\text{DH} = 1$.

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

$$\mathbf{v}^0 = [-1 \quad 1 \quad -1 \quad 1]^t,$$

obținem, prin actualizări asincrone ascendente:

$$\begin{aligned} \mathbf{v}^1 &= [1 \quad -1 \quad -1 \quad 1]^t \\ \mathbf{v}^2 &= [1 \quad -1 \quad -1 \quad 1]^t = \mathbf{v}^3 = \dots \end{aligned}$$

Actualizările au fost astfel descrise pas cu pas. Convergența este către $-\mathbf{s}^{(2)}$ și nu către $\mathbf{s}^{(1)}$ sau $\mathbf{s}^{(2)}$. Avem:

$$\text{DH}(\mathbf{v}^0, \mathbf{s}^{(1)}) = \text{DH}(\mathbf{v}^0, \mathbf{s}^{(2)}) = 2,$$

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:

$$\text{DH}(\mathbf{v}^0, \mathbf{s}^{(1)}) = \text{DH}(\mathbf{v}^0, \mathbf{s}^{(2)}) = 1.$$

Prin actualizări asincrone ascendente, obținem:

$$\begin{aligned}\mathbf{v}^1 &= \mathbf{v}^0 \\ \mathbf{v}^2 &= [-1 \ -1 \ 1 \ 1]^t = \mathbf{v}^3 = \dots = \mathbf{s}^{(1)}\end{aligned}$$

Pe de altă parte însă, dacă actualizările sunt în ordine descendentă, obținem $\mathbf{s}^{(2)}$.

Pentru acest exemplu, avem $p/n = 2/4$, deci memoria este supraîncărcată și sensibilă la perturbații. De aceea, refacerea pattern-urilor perturbate nu este întotdeauna eficientă. Se observă și 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, convergența *nu este neapărat* către cel mai apropiat pattern memorat (apropiere măsurată prin DH). Aceste două deficiențe devin deranjante atunci când memoria este supraîncărcată, deci când p/n este mare.

Capacitatea memoriei autoasociative recurente

Vom menționa aici rezultatele obținute de McEliece *et al.*¹.

Un vector stare \mathbf{v}^k al unei memorii este *stabil* dacă

$$\mathbf{v}^{k+1} = \Gamma [\mathbf{W}\mathbf{v}^k] = \mathbf{v}^k.$$

Această definiție nu este legată de tipul de actualizare: sincron sau asincron.

Fie ρ *raza de atracție* definită astfel: orice vector la o DH mai mică decât ρn (*distanța de atracție*) de starea stabilă \mathbf{v} este eventual atras de \mathbf{v} . Am văzut că distanța de atracție este între 1 și $n/2$, deci ρ este între $1/n$ și $1/2$.

Pentru ca sistemul să funcționeze ca o memorie, impunem ca fiecare vector memorat $\mathbf{s}^{(m)}$ să fie stabil. Mai puțin restrictiv este să cerem să existe un vector stabil la o distanță 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 toleranța la eroare, adică de ρ . În loc de 2^n pattern-uri, putem stoca doar c pattern-uri!

Studiul lui McEliece *et al.* relevă prezența unor atractori în care nu au fost memorate pattern-uri; aceștia au în general bazinele de atracție 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 aplicații în procesarea vorbirii, baze de date, prelucrarea imaginilor.

¹McEliece, 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 bidirecțională (MAB)

MAB (fig. 6.6) este o memorie heteroasociativă adresabilă prin conținut constând din două straturi (Kosko, 1987, 1988).

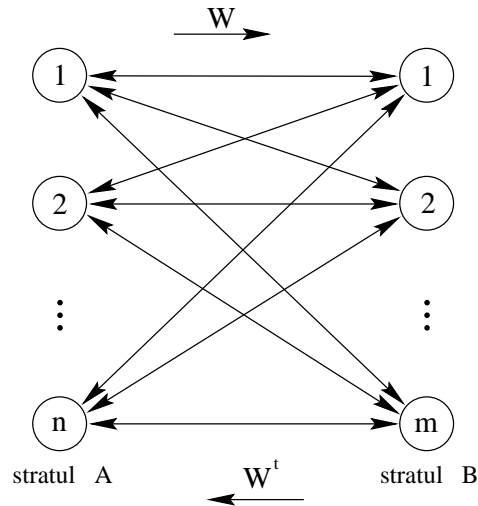


Figura 6.6: Memorie asociativă bidirecțională.

Fie p perechi de pattern-uri memorate:

$$\{(\mathbf{a}^{(1)}, \mathbf{b}^{(1)}), (\mathbf{a}^{(2)}, \mathbf{b}^{(2)}), \dots, (\mathbf{a}^{(p)}, \mathbf{b}^{(p)})\}$$

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:

$$a_i' = \text{sgn} \left(\sum_{j=1}^m w_{ij} b_j \right), \quad i = 1, \dots, n.$$

Procesarea are loc sincron. Urmează:

$$\begin{aligned} \mathbf{b}' &= \Gamma[\mathbf{W}^t \mathbf{a}'] \\ b_j' &= \text{sgn} \left(\sum_{i=1}^n w_{ij} a_i' \right), \quad j = 1, \dots, m. \end{aligned}$$

Apoi:

$$\begin{aligned} \mathbf{a}'' &= \Gamma[\mathbf{W}\mathbf{b}'] \\ \mathbf{b}'' &= \Gamma[\mathbf{W}^t \mathbf{a}''] \text{ etc.} \end{aligned}$$

Procesul continuă până când \mathbf{a} și \mathbf{b} nu se mai modifică.

În mod ideal, procesul se stabilizează într-o pereche $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ din mulțimea perechilor memorate. Putem opera asemănător pentru cazul unipolar. Presupunem că straturile sunt activate alternativ.

Stabilitatea unei astfel de rețele nu este afectată de procesarea sincronă ca în cazul memoriilor Hopfield, deoarece oricum cele două straturi lucrează asincron. De aceea, se preferă procesarea sincronă, convergența fiind mai rapidă decât în cazul în care am folosi procesarea asincronă.

Dacă \mathbf{W} este pătrată și simetrică, atunci $\mathbf{W} = \mathbf{W}^t$ și 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, obținem:

$$\begin{aligned}\mathbf{b} &= \Gamma \left[\sum_{m=1}^p (\mathbf{b}^{(m)} \mathbf{a}^{(m)t}) \mathbf{a}^{(m')} \right] \\ &= \Gamma \left[n \mathbf{b}^{(m')} + \sum_{\substack{m=1 \\ m \neq m'}}^p \mathbf{b}^{(m)} \mathbf{a}^{(m)t} \mathbf{a}^{(m')} \right].\end{aligned}$$

Ultimul termen din relația 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 $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(p)}$ sunt ortogonale. Atunci, $\eta = 0$ și se obține $\mathbf{b} = \mathbf{b}^{(m')}$ 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 și depinde de mai mulți factori: DH dintre vectorul cheie și prototipuri, ortogonalitatea sau DH dintre vectorii $\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(p)}$ etc.

Considerații asupra stabilității

Când memoria MAB ajunge la o stare de echilibru, astfel încât $\mathbf{a}^{(k)} \rightarrow \mathbf{b}^{(k+1)} \rightarrow \mathbf{a}^{(k+2)} = \mathbf{a}^{(k)}$, spunem că ea este bidirecțional stabilă.

Fie funcția 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:

$$\begin{aligned} \Delta a_{i=1,\dots,n} &= \begin{cases} \geq 0 & \text{pentru } \sum_{j=1}^m w_{ij} b_j > 0 \\ 0 & \text{pentru } \sum_{j=1}^m w_{ij} b_j = 0 \\ \leq 0 & \text{pentru } \sum_{j=1}^m w_{ij} b_j < 0 \end{cases} \\ \Delta b_{j=1,\dots,n} &= \begin{cases} \geq 0 & \text{pentru } \sum_{i=1}^n w_{ij} a_i > 0 \\ 0 & \text{pentru } \sum_{i=1}^n w_{ij} a_i = 0 \\ \leq 0 & \text{pentru } \sum_{i=1}^n w_{ij} a_i < 0 \end{cases} \end{aligned} \quad (6.3)$$

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:

$$\begin{aligned} \Delta E_{a_i}(\mathbf{a}, \mathbf{b}) &= - \left(\sum_{j=1}^m w_{ij} b_j \right) \Delta a_i, \quad \text{pentru } i = 1, \dots, n \\ \Delta E_{b_j}(\mathbf{a}, \mathbf{b}) &= - \left(\sum_{i=1}^n w_{ij} a_i \right) \Delta b_j, \quad \text{pentru } j = 1, \dots, m. \end{aligned}$$

Rezultă că $\Delta E \leq 0$, dacă ținem cont și 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 funcția de energie. Deci, memoria este bidirecțional 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 și regăsite cu succes, este $\min(n, m)$. O măsură mai strictă a capacității este:

$$p = \sqrt{\min(n, m)}.$$

Observații

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 și toate pattern-urile stocate pot fi, în mod garantat, recuperate printr-un pas (dacă nu sunt perturbate). Wang (1990) a propus *creșterea fictivă* a pattern-urilor de instruire pentru a obține o ortogonalizare a lor.

Perechile $(\mathbf{a}^{(i)}, \mathbf{a}^{(j)})$ și $(\mathbf{b}^{(i)}, \mathbf{b}^{(j)})$ sunt *fără zgomot*, unde $i, j = 1, \dots, p$, dacă:

$$\begin{aligned} \text{HD}(\mathbf{a}^{(i)}, \mathbf{a}^{(j)}) &= \frac{n}{2} \\ \text{HD}(\mathbf{b}^{(i)}, \mathbf{b}^{(j)}) &= \frac{m}{2}, \end{aligned}$$

acestea fiind condițiile de ortogonalitate între $\mathbf{a}^{(i)}$ și $\mathbf{a}^{(j)}$, respectiv $\mathbf{b}^{(i)}$ și $\mathbf{b}^{(j)}$. Dacă avem această situație, atunci recuperarea datelor este imediată și fără erori (dacă nu sunt perturbate). Prin creșterea vectorilor $\mathbf{a}^{(i)}$ și $\mathbf{b}^{(i)}$, $i = 1, \dots, p$, cu componente adiționale obținem această ortogonalitate la nivel de perechi.

Prin această tehnică se îmbunătățește și capacitatea MAB de a corecta erori.

Utilizarea MAB pentru pattern-uri temporale

Fie secvența $S = \{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(p)}\}$ de vectori n -dimensionali bipolari binari reprezentând tranzițiile stărilor (pattern-uri temporale). Rețeaua MAB este capabilă să memoreze secvența S , astfel încât:

$$\mathbf{s}^{(i+1)} = \Gamma[\mathbf{W}\mathbf{s}^{(i)}]$$

unde i este modulo $p + 1$.

Pornind de la starea inițială $\mathbf{x}(0)$ în vecinătatea lui $\mathbf{s}^{(i)}$, secvența S este apelată ca un ciclu de tranziții ale stărilor. Acest model este numit *memorie asociativă temporală*.

Pentru stocarea secvenței, 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 și B, avem:

$$\begin{aligned} \mathbf{a} &= \Gamma[\mathbf{W}\mathbf{b}] \\ \mathbf{b} &= \Gamma[\mathbf{W}\mathbf{a}]. \end{aligned}$$

Dacă secvența $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(p)}$ este aplicată la intrarea în stratul A, avem:

$$\mathbf{a} = \Gamma \left[\underbrace{(\mathbf{s}^{(2)}\mathbf{s}^{(1)t} + \dots + \mathbf{s}^{(k+1)}\mathbf{s}^{(k)t} + \dots + \mathbf{s}^{(1)}\mathbf{s}^{(p)t})}_{\mathbf{w}} \mathbf{s}^{(k)} \right].$$

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 $\eta = 0$ și $\mathbf{a} = \mathbf{s}^{(k+1)}$, dintr-un singur pas.

Așadar, dacă la intrare avem $\mathbf{s}^{(k)}$, obținem în mod circular:

$$\mathbf{s}^{(k+1)} \rightarrow \mathbf{s}^{(k+2)} \rightarrow \dots \rightarrow \mathbf{s}^{(p)} \rightarrow \dots$$

Dacă vectorii din S nu sunt ortogonali, presupunând însă că ei sunt stocați la o DH $\ll n$, în general ne putem aștepta să obținem totuși secvența corectă S dacă aplicăm $\mathbf{s}^{(k)}$.

Această memorie poate stoca cel mult k secvențe de lungimi p_1, p_2, \dots, p_k , unde:

$$p = \sum_{i=1}^k p_i, \quad p < n.$$

6.6 Exerciții

1. Asociatorul liniar trebuie să asocieze următoarele perechi de vectori:

$$\begin{aligned} \mathbf{s}^{(1)} &= \left[\frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \right]^t \rightarrow \mathbf{f}^{(1)} = [0 \quad 1 \quad 0]^t \\ \mathbf{s}^{(2)} &= \left[\frac{1}{2} \quad -\frac{5}{6} \quad \frac{1}{6} \quad \frac{1}{6} \right]^t \rightarrow \mathbf{f}^{(2)} = [1 \quad 0 \quad 1]^t \\ \mathbf{s}^{(3)} &= \left[\frac{1}{2} \quad \frac{1}{6} \quad \frac{1}{6} \quad -\frac{5}{6} \right]^t \rightarrow \mathbf{f}^{(3)} = [0 \quad 0 \quad 0]^t \end{aligned}$$

- Verificați dacă vectorii $\mathbf{s}^{(1)}$, $\mathbf{s}^{(2)}$, $\mathbf{s}^{(3)}$ sunt ortonormali.
- Calculați matricea ponderilor.
- Verificați asocierea produsă de rețea pentru unul din vectorii $\mathbf{s}^{(i)}$.
- Perturbați $\mathbf{s}^{(i)}$ dându-i unei componente valoarea 0 și calculați vectorul rezultat perturbat.

2. Asociatorul liniar din figura 6.7 a fost instruit să asocieze vectorilor de intrare:

$$\begin{aligned} \mathbf{s}^{(1)} &= \left[\frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{3}} \right]^t \\ \mathbf{s}^{(2)} &= \left[\frac{1}{\sqrt{2}} \quad 0 \quad -\frac{1}{\sqrt{2}} \right]^t \\ \mathbf{s}^{(3)} &= \left[\frac{1}{\sqrt{6}} \quad -\frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}} \right]^t \end{aligned}$$

vectorii $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, $\mathbf{f}^{(3)}$. Găsiți matricea \mathbf{W} și vectorii $\mathbf{f}^{(1)}$, $\mathbf{f}^{(2)}$, $\mathbf{f}^{(3)}$ care sunt codificați în rețea.

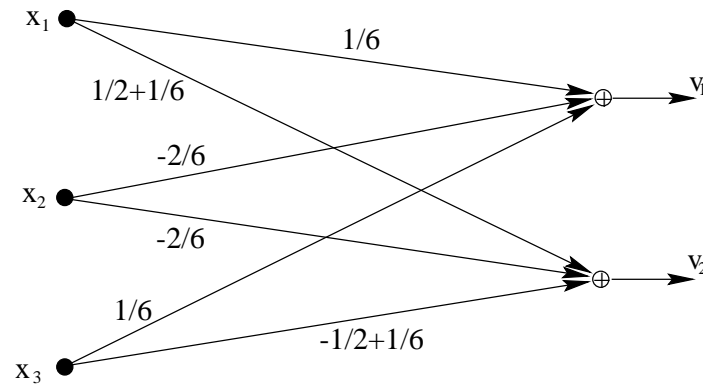


Figura 6.7: Asociatorul linear propus pentru exercițiul 2.

3. Autoasociatorul linear trebuie să asocieze versiuni perturbate ale vectorilor $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3$ cu prototipurile lor neperturbate din problema 1. Calculați matricea \mathbf{W} .
4. Următorii vectori trebuie stocați într-o memorie autoasociativă recurentă:

$$\begin{aligned}\mathbf{s}^{(1)} &= [1 \ 1 \ 1 \ 1 \ 1]^t \\ \mathbf{s}^{(2)} &= [1 \ -1 \ -1 \ 1 \ -1]^t \\ \mathbf{s}^{(3)} &= [-1 \ 1 \ 1 \ 1 \ 1]^t\end{aligned}$$

- (a) Calculați matricea \mathbf{W} .
 - (b) Aplicați la intrare vectorul $\mathbf{v}^0 = [1 \ -1 \ -1 \ 1 \ 1]^t$ și folosiți actualizarea asincronă ascendentă.
 - (c) Aplicați \mathbf{v}^0 prin actualizare asincronă descendentă.
 - (d) Comentați dacă memoria Hopfield oferă o soluție optimă din punct de vedere al DH. (Atenție, memoria este supraîncărcată. Luați $\text{sgn}(0) = 1$ în această problemă).
5. O memorie asociativă temporală trebuie să stocheze următoarea secvență:

$$\begin{aligned}\mathbf{s}^{(1)} &= [1 \ -1 \ -1 \ 1 \ -1]^t \\ \mathbf{s}^{(2)} &= [-1 \ 1 \ 1 \ -1 \ -1]^t \\ \mathbf{s}^{(3)} &= [-1 \ 1 \ 1 \ 1 \ -1]^t\end{aligned}$$

Calculați \mathbf{W} și verificați regăsirea pattern-urilor. Încercați să introduceți la intrare pattern-uri perturbate.

6. (C) Proiectați o memorie heteroasociativă bidirecțională care asociază caracterele (A, C), (I, I) și (O, T). Reprezentați caracterele A, I, O ca matrici

binare de 5×3 pixeli, iar caracterele C, I, T ca matrici binare de 3×3 pixeli. Verificați memoria folosind la intrare pattern-uri neperturbate și pattern-uri perturbate.

Capitolul 7

Rețele neurale cu auto-organizare

În acest capitol ne vom concentra pe rețele neurale care trebuie să descopere singure relațiile interesante din cadrul pattern-urilor de intrare, deci nu vom folosi instruirea supervizată. Obiectivul nostru este să explorăm algoritmi de instruire nesupervizată ai rețelelor neurale, rețele care au astfel capacitatea de *auto-organizare*, adică de adaptare.

7.1 Rețelele Hamming și MAXNET

Vom considera un clasificator cu două straturi pentru vectori binari bipolari (fig. 7.1).

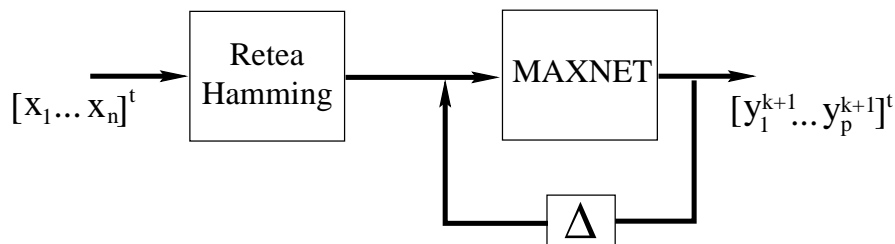


Figura 7.1: Clasificator cu două straturi pentru vectori binari.

Prima parte este o rețea Hamming, detaliată în figura 7.2.

Fie $\mathbf{s}^{(m)}$, $m = 1, \dots, p$, vectorul prototip pentru clasa m . Presupunem că avem pentru fiecare clasă un vector prototip și, corespunzător, un neuron.

Fie \mathbf{x} vectorul de intrare. Folosim produsul scalar al vectorilor ca o măsură de matching (potrivire) între acești vectori. Ideea este să luăm:

$$\mathbf{w}_m = [w_{m1} \ \dots \ w_{mn}]^t \quad \text{pentru } m = 1, \dots, p$$

și 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 poziții în care \mathbf{x} și $\mathbf{s}^{(m)}$ diferă. Numărul

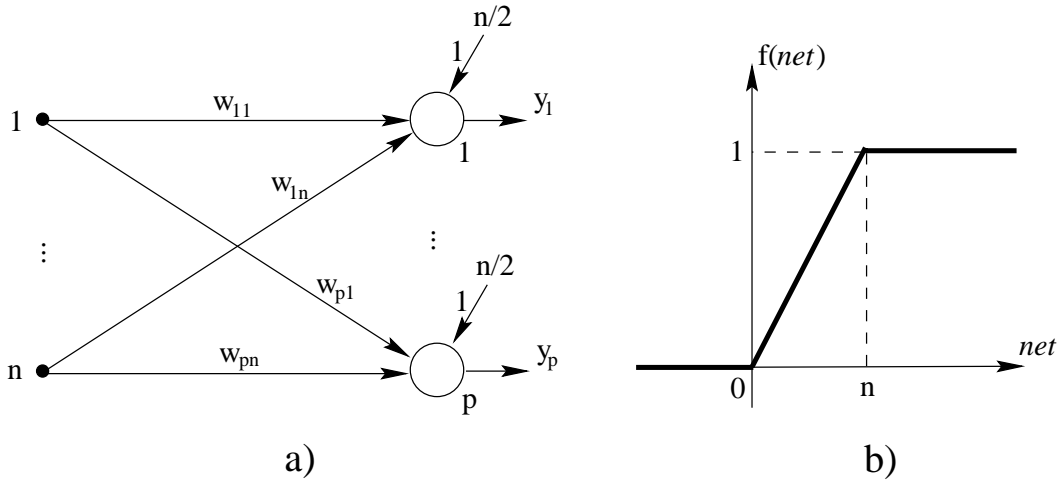


Figura 7.2: a) Rețeaua Hamming; b) Funcția de activare a neuronilor rețelei Hamming. Zonele în care $f(net) < 0$ și $f(net) > 1$ sunt inactive pentru că $0 \leq net \leq n$.

de poziții în care acești doi vectori diferă este însă chiar distanța Hamming, $DH(\mathbf{x}, \mathbf{s}^{(m)})$, prin definiție. Atunci:

$$\mathbf{x}^t \mathbf{s}^{(m)} = \underbrace{(n - DH(\mathbf{x}, \mathbf{s}^{(m)}))}_{\substack{\text{nr. de poziții} \\ \text{în care sunt egali}}} - DH(\mathbf{x}, \mathbf{s}^{(m)}).$$

Obținem:

$$\frac{1}{2} \mathbf{x}^t \mathbf{s}^{(m)} = \frac{n}{2} - DH(\mathbf{x}, \mathbf{s}^{(m)}).$$

Luăm:

$$\mathbf{W}_H = \frac{1}{2} \begin{bmatrix} s_1^{(1)} & \dots & s_n^{(1)} \\ s_1^{(2)} & \dots & s_n^{(2)} \\ \vdots & & \vdots \\ s_1^{(p)} & \dots & s_n^{(p)} \end{bmatrix}$$

unde factorul $\frac{1}{2}$ folosește la scalare. La intrarea în nodul m , $m = 1, \dots, p$ avem:

$$\begin{aligned} net_m &= \frac{1}{2} \mathbf{x}^t \mathbf{s}^{(m)} + \frac{n}{2} \\ &= n - DH(\mathbf{x}, \mathbf{s}^{(m)}). \end{aligned}$$

Se observă că am adăugat termenul $\frac{n}{2}$. Luăm $f(net_m) = \frac{1}{n} net_m$, $m = 1, \dots, p$. 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 $DH(\mathbf{x}, \mathbf{s}^{(m)}) = 0$ și $net_m = n$, deci $f(net_m) = 1$.

2. Dacă \mathbf{x} este complementul lui $\mathbf{s}^{(m)}$, atunci $DH(\mathbf{x}, \mathbf{s}^{(m)}) = n$, $net_m = 0$ și $f(net_m) = 0$.

În general, numărul neuronului cu cea mai mare ieșire indică numărul clasei căreia îi este atribuit \mathbf{x} , din punct de vedere al DH. Rețeaua Hamming este, evident, de tip feedforward.

Rețeaua MAXNET (fig. 7.3) este recurentă și are rolul de a transforma ieșirile din rețeaua Hamming, astfel încât toate aceste ieșiri să devină 0, cu excepția celei maxime.

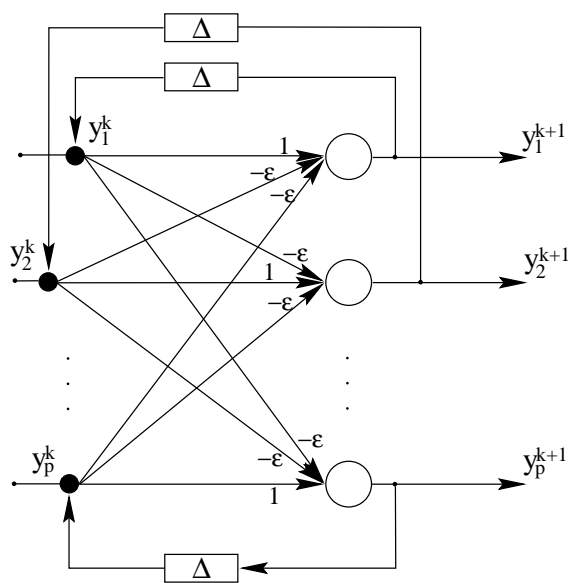


Figura 7.3: Rețeaua MAXNET.

Matricea ponderilor rețelei 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 interacție laterală*.

Luând (fig. 7.4):

$$f(net) = \begin{cases} 0, & net < 0 \\ net, & net \geq 0 \end{cases}$$

$$\mathbf{y}^{(k+1)} = \Gamma[\mathbf{W}_M \mathbf{y}^k]$$

și presupunând $0 \leq y_i^0 \leq 1$, $i = 1, \dots, p$, valorile de inițializare, obținem următorul proces:

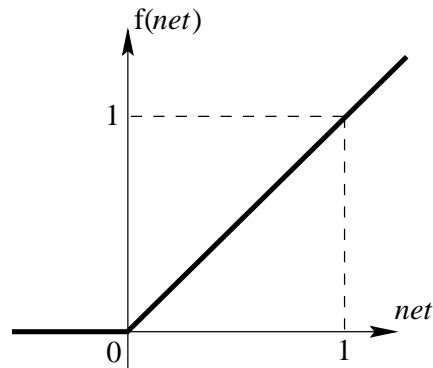


Figura 7.4: Funcția de activare a neuronilor rețelei MAXNET. Zona în care avem $net > 1$ este inactivă.

La fiecare recurență, 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, descreșterea este mai lentă.

Să comparăm această rețea cu asociatorul Hopfield. Presupunem că trebuie să clasificăm în 5 clase posibile un vector cu 50 de componente.

1. Rețeaua Hamming necesită 255 de conexiuni pentru cei 5 neuroni.
2. Rețeaua Hopfield necesită 2450 de conexiuni pentru cei 50 de neuroni. În plus, capacitatea de stocare este limitată.

Rețeaua Hamming pare mai eficientă. Totuși, ea nu regăsește pattern-ul prototip, ci doar indicele clasei din care face parte pattern-ul de intrare. Deci, rețeaua Hamming nu poate restaura un pattern perturbat, ci găsește doar distanța minimă a acestui pattern față de prototipuri.

Rețeaua Hamming este un clasificator propriu-zis, nu o memorie autoasocia-tivă.

7.2 Instruirea nesupervizată a clusterelor

Clustering înseamnă gruparea pattern-urilor similare și separarea celor nesimilare. Există tehnici clasice de clustering: k-means, ISODATA etc, dar noi ne vom concentra pe metodele conexioniste.

Instruirea câștigătorul ia tot

Presupunem că avem secvența de instruire $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ care reprezintă p clustere. Pentru rețeaua Kohonen (fig. 7.5), despre care vom discuta în continuare, avem:

$$\mathbf{y} = \Gamma[\mathbf{W}\mathbf{x}]$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^t \\ \mathbf{w}_2^t \\ \vdots \\ \mathbf{w}_p^t \end{bmatrix}, \quad \text{unde } \mathbf{w}_i = \begin{bmatrix} w_{i1} \\ \vdots \\ w_{in} \end{bmatrix}, \quad i = 1, \dots, p$$

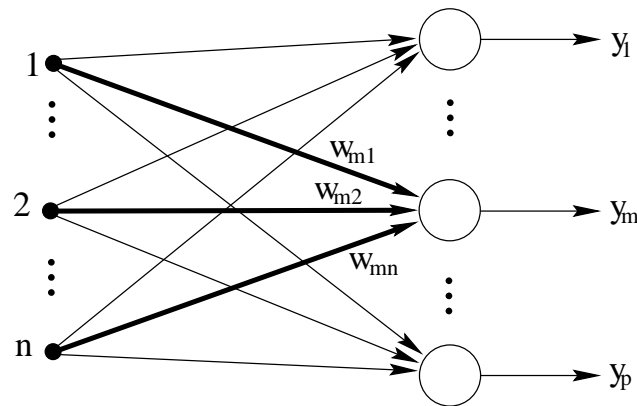


Figura 7.5: Rețeaua Kohonen.

Algoritmul de instruire este de tip *câștigătorul ia tot*. Vectorii \mathbf{w}_i , $i = 1, \dots, p$ sunt variabili și trebuie învățați. Înainte de instruire, se normalizează vectorii ponderilor:

$$\hat{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}, \quad i = 1, \dots, p.$$

Se calculează indicele m pentru care:

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = \min_{i=1, \dots, p} \{\|\mathbf{x} - \hat{\mathbf{w}}_i\|\}$$

unde \mathbf{x} este vectorul de intrare. Deoarece:

$$\|\mathbf{x} - \hat{\mathbf{w}}_m\| = (\mathbf{x}^t \mathbf{x} - 2\hat{\mathbf{w}}_m^t \mathbf{x} - 1)^{1/2},$$

avem:

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

Neuronul al m -lea, cu cea mai mare valoare pentru intrare este declarat *câștigător*.

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 și criteriu de similaritate între \mathbf{x} și \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} și \mathbf{x}_i , pentru că altfel criteriul nu mai funcționează: se poate ca $\|\mathbf{x} - \mathbf{x}_i\|$ să fie mare, dar ψ să fie mic.

Reducem acum $\|\mathbf{x} - \mathbf{w}_m\|$ în direcția gradientului negativ al distanței:

$$\begin{aligned}\nabla_{\mathbf{w}_m} \|\mathbf{x} - \mathbf{w}_m\|^2 &= -2(\mathbf{x} - \mathbf{w}_m) \\ \Delta \hat{\mathbf{w}}_m' &= \alpha (\mathbf{x} - \hat{\mathbf{w}}_m)\end{aligned}$$

unde α este de obicei între 0,1 și 0,7. Celelalte ponderi nu se schimbă. Avem atunci:

$$\begin{cases} \hat{\mathbf{w}}_m^{k+1} = \hat{\mathbf{w}}_m^k + \alpha^k (\mathbf{x} - \hat{\mathbf{w}}_m^k) \\ \hat{\mathbf{w}}_i^{k+1} = \hat{\mathbf{w}}_i^k, \quad i \neq m \end{cases}.$$

Procesul continuă pentru un nou pattern \mathbf{x} . Parametrul α se reduce monoton și instruirea încetinește pe parcurs. Vectorul modificat $\hat{\mathbf{w}}_m^{k+1}$ este normalizat și 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.

Funcțiile de activare ale neuronilor nu au relevanță, dar în general sunt de tip continuu.

Obținem în final un clasificator în care pattern-ul de intrare \mathbf{x} este asociat grupării m , unde $\mathbf{y}_m = \max(\mathbf{y}_1, \dots, \mathbf{y}_p)$.

După instruire, rețeaua Kohonen poate fi folosită ca și clasificator. Pentru aceasta, trebuie să o *calibrăm* supervizat: se aleg p reprezentanți ai celor p clustere și se aplică, etichetând corespunzător cele p noduri cu ieșirile respective. Astfel, etichetăm de fapt clusterelor.

Ponderile inițiale sunt luate, în general, în mod aleator, astfel încât să acopere uniform spațiul pattern-urilor.

Rețeaua Kohonen are limitări legate de arhitectura monostrat: nu poate manipula eficient pattern-uri liniar neseperabile. Chiar și pentru pattern-uri liniar separabile, ponderile pot să se blocheze în regiuni izolate, fără să formeze clusterelor adecvate. În astfel de situații, se reinițializează 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. Acești neuroni nu reprezintă clustere, deci pot fi omiși în faza de utilizare a rețelei.

7.3 Hărți Kohonen

Centrii anumitor activități (vorbire, vedere, auz, funcții motoare) sunt localizați în anumite arii specifice ale cortexului. Mai mult, aceste arii prezintă o ordonare logică a funcționalității lor. Putem da ca exemplu, *harta tonotopică* a regiunilor auditive, unde neuronii vecini răspund unor frecvențe similare ale sunetului, de la frecvențe înalte la frecvențe joase. Un alt exemplu este *harta somatotopică*.

Astfel de regiuni, cum este harta tonotopică, se numesc *hărți de trăsături ordonate* (*ordered feature maps*). Vom studia în această secțiune mecanismul prin care aceste hărți de trăsături ordonate se pot dezvolta natural.

Cortexul este o suprafață de 2-4 mm grosime, având aproximativ 1m^2 și constând din 6 straturi de neuroni de tip și densitate variabile. El are forma pe care o cunoaștem pentru a maximiza densitatea în craniu.

Vom studia un model al cortexului care este o suprafață bidimensională de elemente de procesare. Vectorii de intrare vor fi proiectați, prin reducerea dimensionalității, pe această suprafață, menținându-se însă ordinea între ei. Această suprafață este o hartă care conservă topologia datelor de intrare (*topology preserving map*). O astfel de hartă este o hartă cu auto-organizare și este instruită nesupervizat. Neuronii biologici de pe suprafața menționată au conexiuni laterale a căror tărie depinde de distanța dintre neuronii respectivi (fig. 7.6).

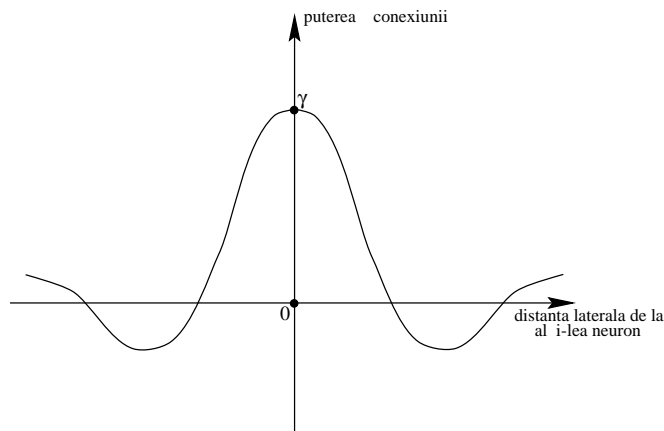


Figura 7.6: "Pălăria mexicană" descrie puterea conexiunii laterale a neuronilor biologici.

În vecinătatea cu raza de $50\text{-}100\ \mu\text{m}$, 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 $200\text{-}500\ \mu\text{m}$.

Autoorganizarea unei astfel de hărți poate fi exemplificată pe un model liniar de 10 neuroni, reprezentând o secțiune arbitrară a unei hărți 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.

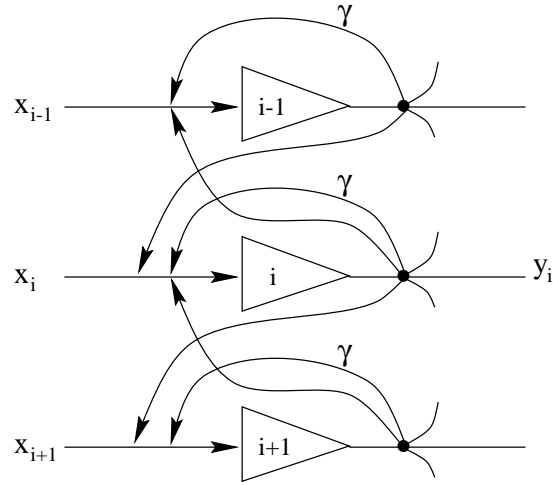


Figura 7.7: Modelarea interacțiunilor dintre neuronii biologici.

Coeficienții de feedback, γ_k , depind de distanța dintre neuroni. Luând:

$$x_i(t) = 0,5 \sin^3 \left(\frac{\pi(i+3)}{17} \right), \quad i = 1, \dots, 10$$

$$f(net) = \begin{cases} 0 & net \leq 0 \\ net & 0 < net < 2 \\ 2 & net \geq 2 \end{cases}$$

$$b = 0,4$$

$$c = 0,2$$

se obțin ieșirile din figura 7.9.

Un *algoritm de proiecție a trăsăturilor* convertește un pattern de dimensiune arbitrară în răspunsurile unui tablou unidimensional sau bidimensional de neuroni.

Rezultatele următoare aparțin lui Kohonen (1990).

Presupunem că vectorii de intrare, cu un număr arbitrar, dar fixat, de componente, sunt proiectați pe un tablou de neuroni având o topologie hexagonală (fig. 7.10). În această figură, m este neuronul câștigător, iar $N_m(t_1)$ este vecinătatea lui m la momentul t_1 .

După instruire, fiecare intrare \mathbf{x} produce un răspuns în rețea, poziția locală a răspunsului fiind semnificativă: ea reflectă caracteristica trăsăturii dominante a lui \mathbf{x} . Această proiecție a trăsăturilor este o proiecție neliniară a spațiului patternurilor pe tabloul neuronilor. Proiecția face ca relațiile de vecinătate topologică să explicitizeze relațiile de ordine dintre datele de intrare, aceste relații din urmă fiind, însă, într-un spațiu de dimensiune mult mai mare.

Să vedem cum are loc instruirea.

Intrarea \mathbf{x} este aplicată simultan tuturor neuronilor. Fie

$$\|\mathbf{x} - \mathbf{w}_m\| = \min_i \{\|\mathbf{x} - \mathbf{w}_i\|\}$$

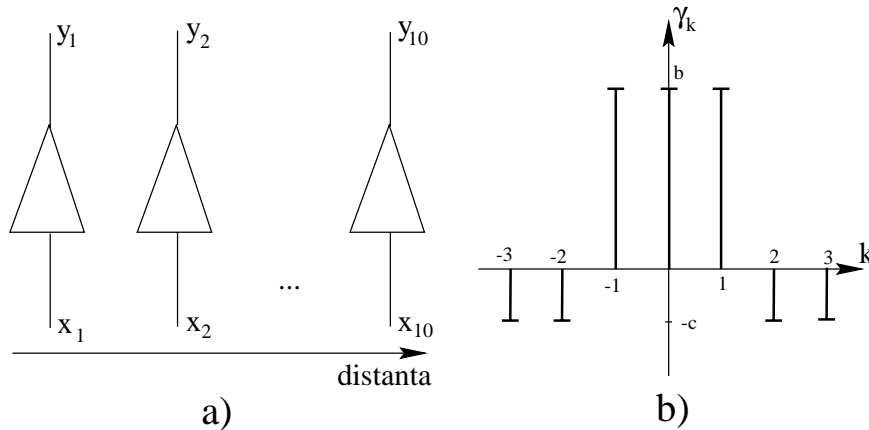


Figura 7.8: a) Secțiune arbitrară a unei hărți bidimensionale; b) Feedback-ul lateral.

unde \mathbf{w}_i este vectorul ponderilor către neuronul i .

Dacă neuronul m este câștigător, deci răspunde maxim pentru \mathbf{x} , aceasta antrenează, ca în exemplul precedent, neuronii vecini. Nu avem un neuron câștigător, ci o vecinătate N_m câștigătoare. Se ajustează doar ponderile către neuronii din N_m .

Raza lui N_m trebuie să scadă pe măsură ce instruirea progresează:

$$N_m(t_1) > N_m(t_2) > N_m(t_3) \dots \text{ pentru } t_1 < t_2 < t_3 \dots$$

Se începe cu o rază foarte mare și se termină cu vecinătatea care include doar neuronul câștigător. Mecanismul interacțiilor laterale, tip ”pălărie mexicană”, este încadrat în definiția acestor vecinătăți locale și a modului de adaptare a ponderilor.

Adaptarea ponderilor pentru hărțile 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 învățare. 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 poziției neuronului m , r_i este vectorul poziției neuronului i , iar $\alpha(t)$ și $\sigma(t)$ sunt funcții descrescătoare în timp.

Ca rezultat al instruirii ponderilor, se obține o hartă de neuroni în care ponderile codifică funcțiile de densitate $p(\mathbf{x})$ ale probabilităților pattern-urilor de instruire. Răspunsul neuronilor unei arii se întărește dacă apar mai multe pattern-uri similare corespunzătoare acelei arii.

Fie de exemplu 32 de vectori 5-dimensionali diferiți, etichetați cu A, ..., 6 (tab. 7.1).

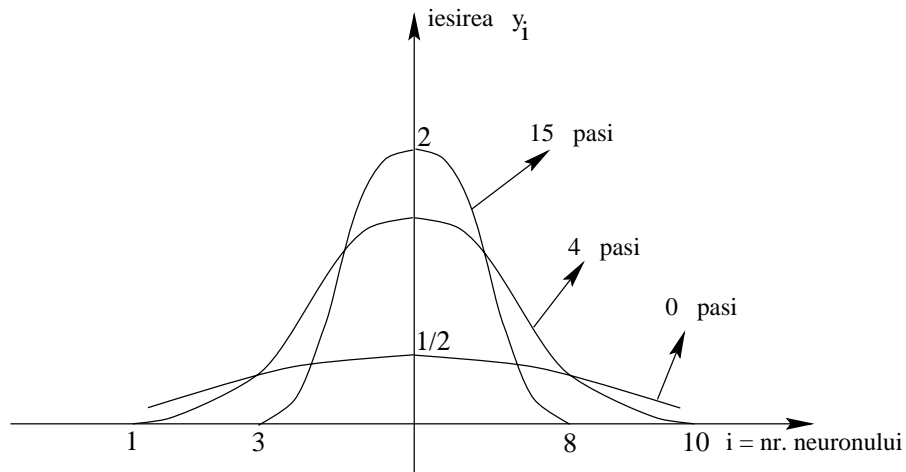


Figura 7.9: Evoluția ieșirilor celor 10 neuroni.

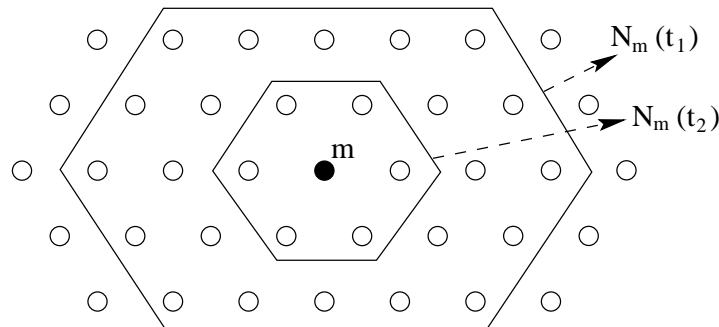


Figura 7.10: Tablou de neuroni având o topologie hexagonală.

Proiecția se face pe un tablou de 70 de neuroni, conectați fiecare prin 5 ponderi cu componentele x_1, \dots, x_5 . Tabloul se instruieste folosind cei 36 de vectori selectați în ordine aleatoare. După 10000 de pași de instruire, ponderile se stabilizează și rețeaua obținută este calibrată. Calibrarea se face prin etichetare supervizată. De exemplu, având vectorul B la intrare, observăm unde apare răspunsul maxim în tabloul de neuroni și etichetăm acel neuron cu B. Obținem harta din fig. 7.11.

32 de neuroni vor fi etichetați și 38 nu. Inspectăm similaritățile dintre vectorii de instruire. Reprezentăm vectorii ca vârfuri într-un graf. Fiecărei muchii îi atașăm distanța dintre cele două pattern-uri corespunzătoare vârfurilor. De exemplu, distanța dintre A și C este egală cu:

$$\|A - C\| = 2.$$

Construim în continuare arborele parțial 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 obținută în urma instruirii.

poate observa că harta trăsăturilor produsă prin auto-organizare are aceeași structură ca și acest arbore. Înseamnă că am reușit să păstrăm topologia dintr-un spațiu 5-dimensional într-un plan. Distanța dintre vectorii \mathbf{x}_i și \mathbf{x}_j din spațiul pattern-urilor:

$$\|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{k=1}^5 (x_{ik} - x_{jk})^2}$$

se menține și în spațiul trăsăturilor, unde devine o simplă distanță în plan.

Algoritm de instruire ține cont atât de relațiile topologice dintre datele de intrare cât și de istoricul datelor de intrare, adică de frecvența de apariție a pattern-urilor similare sau identice.

Acest algoritm are numeroase aplicații în controlul roboților, recunoașterea vorbirii etc.

Un exemplu celebru este *mașina de scris fonetică* (Kohonen, 1988), folosită în recunoașterea vorbirii.

La intrare avem un vector 15-dimensional. Semnalul vocal este convertit prin FFT în semnal în frecvență acoperind pe 15 canale frecvențele dintre 200Hz și 5kHz. Fiecare fonem tinde să ocupe un segment al spectrului 15-dimensional Fourier. Cu toate că spectrele unor foneme diferite se suprapun parțial, un tablou de neuroni este capabil, după instruire, să extragă trăsăturile fonemelor și să le proiecteze pe o hartă plană. Rețeaua neurală a fost instruită cu semnale spectrale generate la fiecare 9,83 ms de vorbirea continuă. Eșantioanele spectrului FFT au fost folosite în ordinea lor naturală

După instruire, anumiți neuroni devin sensibili la spectrul diferitelor foneme (fig. 7.13).

Harta a fost calibrată folosind foneme standard. Majoritatea celulelor au

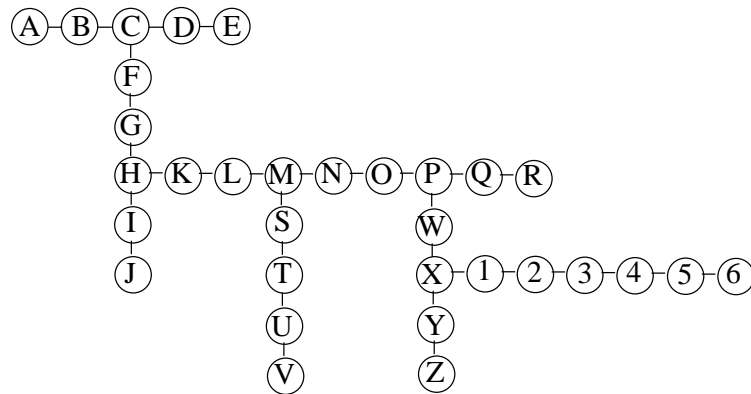


Figura 7.12: Arborele parțial de lungime minimă construit pe baza distanțelor dintre vectori.

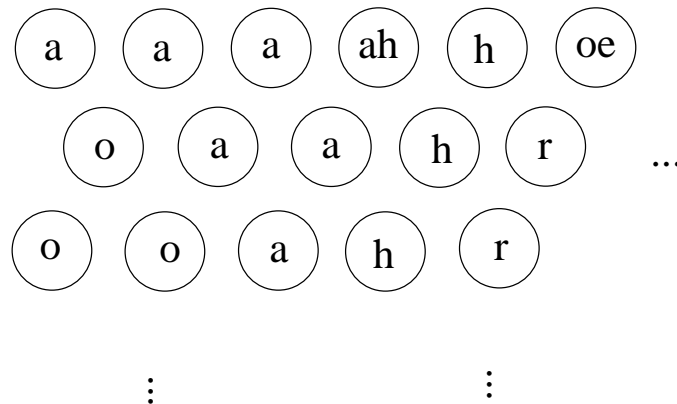


Figura 7.13: Harta fonemelor după instruire (în limba finlandeză).

învățat 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 Exerciții

1. (C) Proiectați rețeaua Hamming și rețeaua MAXNET asociată care realizează clasificarea optimă a vectorilor binari bipolari care reprezintă caracterele A, I și O din problema 7, capitolul 4 în termenii distanței Hamming minime.
2. (C) Implementați algoritmul de instruire *câștigătorul ia tot* pentru gruparea caracterelor C, I și T afectate de zgomot, în mod similar reprezentării folosite în problema anterioară. Presupuneți că aceste caractere afectate de

zgomot au $DH=1$ față de prototipurile neafectate de zgomot. Calculați vectorii finali de ponderi \mathbf{w}_C , \mathbf{w}_I și \mathbf{w}_T . În timpul operației de testare, calculați răspunsurile $f(net_C)$, $f(net_I)$ și $f(net_T)$ fiecărui neuron cu funcție de activare unipolar continuă. Intrările sunt vectori bipolari binari obținuți din literele reprezentate matriceal. Folosiți $\lambda = 1$.

3. (C) Fie mulțimea de pattern-uri

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_5 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Implementați harta Kohonen pentru a proiecta acești vectori pe o rețea de 5×5 neuroni. Folosiți vecinătăți hexagonale. Descreșteți pe α de la 0,5 la 0,004 pentru primii 1000 de pași, apoi descreșteți-l de la 0,04 la 0 pentru următorii 9000 de pași. Descreșteți raza vecinătății de la 2 la 0 pe parcursul primilor 1000 de pași. Calibrați apoi harta și desenați arborele parțial 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

Rețele neurale cu funcție de activare radială

Aceste rețele neurale au un timp de instruire mult mai rapid decât rețelele neurale feedforward cu propagare în urmă a erorii, evitând și problema minimului local. Dezavantajul este că, după instruire, necesită un timp de calcul mai mare în exploatare. În 1990 s-a demonstrat că aceste rețele sunt aproximatori universali, ca și rețelele feedforward cu funcție de activare continuă. Aceasta înseamnă că pot aproxima orice funcție continuă, oricât de bine.

O astfel de rețea este o rețea feedforward în care funcția de activare este o funcție de tip gaussiană. Deoarece gaussiană este simetric radială, vom numi aceste rețele "cu funcție de activare radială". Gaussiană nu este însă decât o funcție particulară de acest tip.

Spre deosebire de rețelele feedforward, ponderile conexiunilor către straturile ascunse nu sunt inițializate aleator. Ele sunt inițializate cu valorile care produc răspunsul dorit.

8.1 Funcții radiale

Fie cazul unei singure intrări într-un neuron ascuns (fig. 8.1).

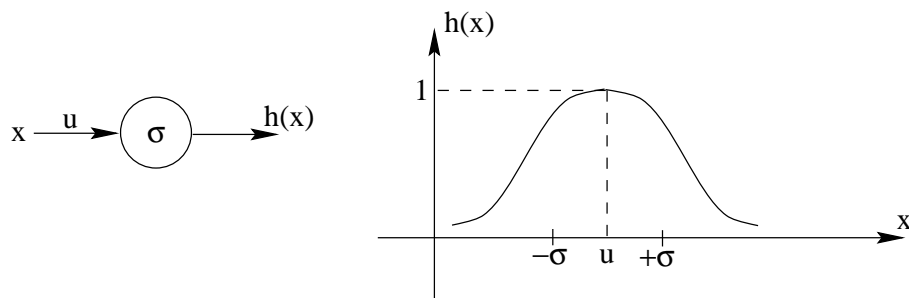


Figura 8.1: Neuron cu funcție de activare radială.

Pentru neuronii de acest tip, avem:

$$h(x) = e^{-\frac{(x-u)^2}{2\sigma^2}}$$

$$h(u) = 1,$$

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 potențialul membranei, cum este cazul funcției de activare de forma $f(\mathbf{x}^t \mathbf{w})$. De aceea, rețelele de acest tip se numesc "radial basis function neural networks". Ponderile nu se mai înmulțesc cu intrările. Le vom numi totuși astfel, fără a crea confuzie.

Ieșirea $h(x)$ 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 distribuția normală, u poate fi valoarea medie, iar σ^2 dispersia funcției de activare a neuronului.

Forma generală a unei rețele neurale bazată pe acest tip de neuron este cea din figura 8.2.

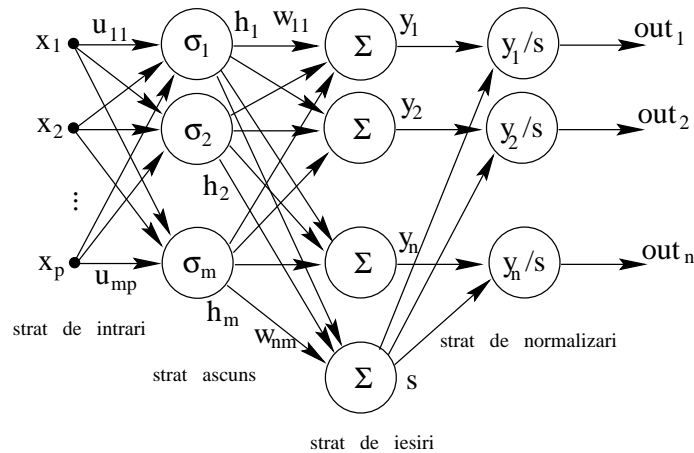


Figura 8.2: Rețea neurală cu funcție de activare radială.

Avem:

$$h_i = e^{-\frac{(\mathbf{x}-\mathbf{u}_i)^t(\mathbf{x}-\mathbf{u}_i)}{2\sigma_i^2}}$$

unde:

$$\mathbf{x} = [x_1 \quad \dots \quad x_p]^t,$$

σ_i este parametrul pentru al i -lea neuron ascuns și

$$\mathbf{u}_i = [u_{i1} \quad \dots \quad u_{ip}]^t$$

este vectorul ponderilor dintre intrări și al i -lea neuron ascuns.

Fiecărui pattern învățat îi corespunde un neuron ascuns.

În stratul de ieșire se calculează:

$$y_j = \sum_{i=1}^m h_i w_{ji},$$

iar pe ultimul strat, cel al normalizărilor, se calculează ieșirile finale:

$$s = \sum_{j=1}^n y_j$$

$$out_i = \frac{y_i}{s}.$$

Aceste din urmă normalizări sunt opționale.

Domeniul receptiv al celui de-al i -lea neuron ascuns este determinat de centrul său, \mathbf{u}_i , și de mărimea sa, dată de σ_i . La neuronii studiați până acum, acest domeniu era determinat de conexiunile neurale, nu de forma funcției de activare. Se pot folosi și alte funcții de activare. Esențial este ca ele să tindă rapid către zero atunci când distanța dintre vectorii \mathbf{x} și \mathbf{u} crește, iar răspunsul maxim să îl aibă când $\mathbf{x} = \mathbf{u}$.

Rețeaua trebuie instruită prin fixarea parametrilor σ_i și a ponderilor de tip \mathbf{u} și \mathbf{w} . Instruirea se face în două faze. La început se atribuie valori pentru \mathbf{u}_i și σ_i , $i = 1, \dots, m$. Apoi, se instruieste matricea \mathbf{W} prin învățare 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 secvența de instruire câte un neuron ascuns, cu centrul corespunzător. Deoarece vectorii sunt, în general, dispuși în grupări (clustere), această metodă duce la un număr prea mare de neuroni ascunși. Mai bine este să găsim centrul fiecărui cluster de vectori din secvența de instruire și 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 rețelei. Obiectivul este să acoperim spațiul pattern-urilor de intrare, cât mai uniform, cu domenii receptive. Dacă distanța 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 distanță mare, atunci σ trebuie să fie suficient de mare pentru a acoperi golurile; dacă centrii sunt apropiați, 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, găsește distanța medie dintre centrul său, \mathbf{u}_i și centrii celor N vecini ai săi cei mai apropiați.
2. Atribuie această valoare lui σ_i

Instruirea matricii \mathbf{W}

1. Se aplică vectorul \mathbf{x} la intrare, unde \mathbf{x} este din secvența de instruire.
2. Se calculează ieșirile \mathbf{h} din stratul ascuns.
3. Se calculează ieșirile \mathbf{y} și se compară cu vectorul \mathbf{t} al ieșirilor dorite. Se ajustează \mathbf{W} astfel:

$$w_{ij}(k+1) = w_{ij}(k) + \eta(t_i - y_i)x_j,$$

unde η este coeficientul de învățare. De obicei $\eta \ll 1$, iar aceasta este de fapt regula Windrow-Hoff. Se pot folosi și 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, convergența 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 obținem câte un reprezentant pentru fiecare cluster și să instruiem apoi rețeaua doar cu acești reprezentanți. Putem pondera reprezentanții 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 și eficientă. Se definește o rază r . Primul pattern de instruire devine centrul primului cluster. Fiecare pattern din secvența de instruire este considerat pe rând. Dacă distanța dintre pattern-ul considerat și cel mai apropiat cluster este mai mică sau egală decât r , atunci pattern-ul este atribuit aceluia cluster; dacă distanța este mai mare decât r , pattern-ul devine primul membru al unui nou cluster.

Procedeeul de clustering este folositor atunci când există un exces de instruire cu date care formează clustere dense.

8.3 Discuție

Rețelele cu funcție de activare radială sunt foarte rapide la instruire, dar lente la utilizare. Procedul de clustering poate fi util. Foarte importantă este alegerea corectă a lui σ .

Capitolul 9

Rețele 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 și uniform, economisind 10% energie comparativ cu soluția convențională.

În Vest, interesul este ceva mai mic. Explicația este probabil și tradiția logicii aristotelice binare care spune că ceva există sau nu există. Filosofia și religia orientale se bazează pe principiul că ceva există și, în același timp, nu există, de exemplu YIN și YANG.

Atunci când nu au putut converti binar o informație conform principiului true/false, occidentalii au apelat la statistică. Astfel, probabilitatea ca o mașină să fie lentă sau rapidă se exprimă printr-un număr între 0 și 1. Logica fuzzy produce același rezultat, etichetând mașinile în funcție de viteză cu numere între 0 și 1. Aceasta a produs o mare confuzie, neînțelegându-se diferența dintre cele două demersuri. Vom vedea însă că diferența este fundamentală.

Inițiatorul logicii fuzzy este, paradoxal, un occidental: Zadeh în 1965.

Controlere fuzzy

Sistemele fuzzy au fost folosite foarte des în controlere. Aceste controlere sunt ușor de proiectat, uneori în câteva ore. Performanțele lor sunt mai bune decât în cazul controlerelor clasice. Proiectarea unui sistem fuzzy nu necesită cunoștințe 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 mulțimi de reguli fuzzy. În cazul nostru, ele sunt:

1. Dacă liftul se ridică lent și poziția sa este departe de poziția dorită, atunci mărește viteza.

2. Dacă liftul se ridică cu o viteză medie și este aproape de poziția dorită, atunci micșorează viteza.

Exprimăm același lucru prin reguli IF-THEN:

1. IF viteza este SP (small positive) AND poziția este LN (large negative) THEN tensiunea motorului este LP (large positive).
2. IF viteza este MP (medium positive) AND poziția este SN (small negative) THEN tensiunea motorului este SP (small positive).

Sarcina controlerului fuzzy este să proceseze aceste două reguli, eventual și altele, după ce a determinat poziția și accelerația.

O regulă are două părți: partea *antecedentă* și cea *consecventă*. Partea antecedentă poate conține mai multe *clauze* legate prin AND, OR și NOT. O clauză este, de exemplu "viteza este SP". Clauzele pot fi descompuse în *variabile* și *adjective*. De exemplu, variabile sunt "viteza" și "poziția", iar adjective sunt SP, LN și LP.

Gradul în care o regulă este relevantă într-o situație măsurată este același 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 și de operațiile logice care leagă aceste clauze.

În logica fuzzy, fiecărui adjectiv (cum este și SP) i se atribuie o funcție de apartenență. Valoarea funcției de apartenență este între 0 și 1. Dacă fiind valoarea unei variabile, funcția de apartenență 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 funcția de apartenență la mulțimea bărbaților înalți a acestuia are valoarea 0. Pentru 1,75m avem o valoare între 0 și 1.

Folosind funcțiile de apartenență 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: Funcția de apartenență a vitezei la SP are valoarea 0,6 și funcția de apartenență a poziției 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 ieșire pentru cele două reguli. Acest proces se numește *defuzzificare*. La acest proces, nu există un consens asupra modului de operare.

Tipul de defuzzificare ilustrat în figura 9.1 se numește defuzzificare min-max.

Într-un sistem fuzzy, toate regulile acționează simultan asupra ieșirii. La ieșire avem tensiunea corespunzătoare centrului de greutate al sumei regiunilor

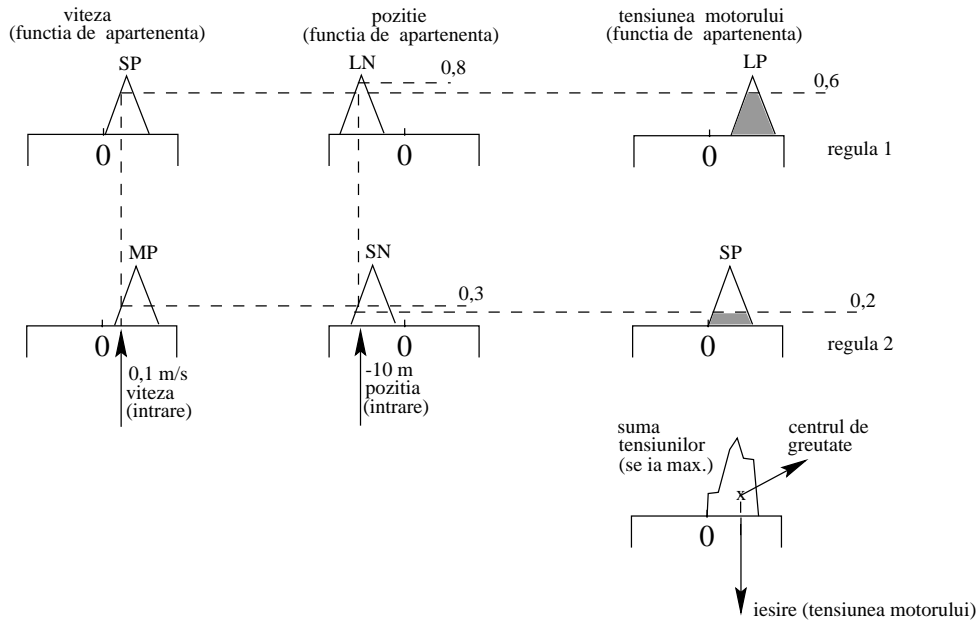


Figura 9.1: Defuzzificare min-max.

hașurate. Trebuie să facem observația că forma funcției de apartenență este dată de proiectant. De obicei se alege o funcție triunghiulară sau trapezoidală.

Logica fuzzy generalizează logica binară, permițând valori reale între 0 și 1.

9.1.2 Logica fuzzy, logica convențională și teoria mulțimilor

Ca și teoria mulțimilor, logica fuzzy se ocupă de conceptul apartenenței la o mulțime. În teoria clasică a mulțimilor, un element poate sau nu poate să aparțină unei mulțimi, ceea ce este nenatural.

Exemplu: Un bărbat de 1,80 m poate avea un grad de apartenență de 0,6 la mulțimea bărbaților înalți. Aceasta înseamnă și că acest bărbat are un grad de apartenență de 0,4 la mulțimea bărbaților care "nu sunt înalți". Acest bărbat aparține deci simultan celor două mulțimi complementare, ceea ce ar fi imposibil în teoria convențională.

Logica convențională definește o funcție de apartenență la o mulțime \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, funcția de apartenență devine: $f_{\mathcal{A}}(x) \in [0, 1]$.

Ideea de a vedea o mulțime ca o funcție capabilă să definească grade de apartenență stă la baza teoriei mulțimilor fuzzy. Evident, este nevoie de o nouă matematică, cea introdusă de Zadeh.

Logica convențională și logica fuzzy

Logica convențională folosește AND, OR și NOT. Orice relație poate fi descrisă prin combinarea acestor operatori. Funcțiile logice fuzzy operează asupra unor valori de adevăr în $[0, 1]$. Pentru valorile 0 și 1, o astfel de funcție trebuie să dea același rezultat ca și în logica convențională. Evident, există o infinitate de funcții care să satisfacă aceste cerințe. Trebuie alese cele mai avantajoase.

Funcția AND poate fi implementată astfel:

$$\text{and}(X, Y) = \min(X, Y)$$

unde $X, Y \in [0, 1]$.

Funcția OR: $\text{or}(X, Y) = \max(X, Y)$.

Funcția NOT: $\text{not}(X) = 1 - X$.

Teoria clasică și fuzzy a mulțimilor

Fie $m_{\mathcal{A}}(x)$ valoarea funcției de apartenență a punctului x la mulțimea \mathcal{A} și $m_{\mathcal{B}}(x)$ valoarea funcției de apartenență a punctului x la mulțimea \mathcal{B} . Definim:

$$m_{\mathcal{A} \cap \mathcal{B}}(x) = \min[m_{\mathcal{A}}(x), m_{\mathcal{B}}(x)].$$

Este o definiție compatibilă cu definiția funcției AND. Este relația 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 exercițiu, calculați intersecția și reuniunea mulțimilor \mathcal{A} și \mathcal{A}^c .

Mulțimi fuzzy și paradoxuri logice

Bertrand Russell a dat următorul paradox al bărbierului: Într-un oraș există un bărbier care bărbierește pe oricine nu se bărbierește singur. Problema este dacă bărbierul se bărbierește singur. Apare o contradicție logică, un paradox. Dacă propoziția P : "se bărbierește singur" are valoarea de adevărat, $\text{truth}(P)$, atunci $\text{truth}(P) = \text{truth}(\text{not}P)$, deoarece bărbierul se bărbierește și, în același timp, nu se bărbierește singur. Folosind complementul fuzzy,

$$\text{truth}(\text{not}P) = 1 - \text{truth}(P).$$

Atunci,

$$\text{truth}(P) = 1 - \text{truth}(P),$$

deci

$$\text{truth}(P) = 0,5.$$

În logica fuzzy, propozițiile pot avea valori de adevărat oriunde între 0 și 1, deci paradoxul lui Russell nu mai există.

Desigur, eliminând toate restricțiile asupra imaginației, putem practic explica orice. De exemplu, să considerăm superstițiile popoarelor primitive care explică fenomenele naturale cu ajutorul entităților supranaturale. Logicienii convenționali acuză teoreticienii fuzzy de "construcție artificială". Teoriile matematice trebuie însă judecate prin consistența, frumusețea și utilitatea lor. Teoria fuzzy satisface toate aceste cerințe.

Observații matematice

Fie \mathcal{X} o mulțime oarecare. Fie $f_{\mathcal{A}} : \mathcal{X} \rightarrow [0, 1]$ o funcție. Funcția $f_{\mathcal{A}}$ definește o mulțime fuzzy (în \mathcal{X}), adică mulțimea $\mathcal{A} = \{(x, f_{\mathcal{A}}(x)) | x \in \mathcal{X}\}$. Funcția $f_{\mathcal{A}}$ este funcția de apartenență la mulțimea \mathcal{A} .

Apare un abuz de limbaj între "mulțimea" fuzzy \mathcal{A} și "funcția" de apartenență la \mathcal{A} . Acest abuz se datorează faptului că definim o mulțime prin gradul de apartenență la această mulțime, nu prin enumerarea mulțimii, cum suntem obișnuiți.

9.2 Rețele neurale fuzzy

9.2.1 Neuroni fuzzy

Un neuron non-fuzzy are N intrări ponderate și o ieșire. Ieșirea se calculează astfel:

$$y = f \left(\sum_{i=1}^N w_i x_i - T \right)$$

unde f este funcția de activare, în general neliniară. Un neuron fuzzy (NF) (fig. 9.2) are N intrări ponderate și M ieșiri.

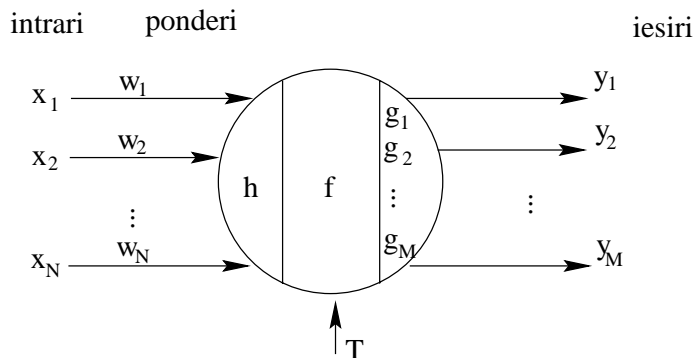


Figura 9.2: Neuronul fuzzy.

Toate intrările și ponderile sunt valori reale, iar ieșirile sunt valori reale în intervalul $[0,1]$. Fiecare ieșire poate fi asociată gradului de apartenență (în sensul fuzzy), adică exprimă în ce măsură pattern-ul cu intrările $\{x_1, x_2, \dots, x_N\}$ aparține unei mulțimi fuzzy. Avem:

$$z = h(w_1x_1, w_2x_2, \dots, w_Nx_N)$$

unde z este intrarea în NF, iar h este funcția de agregare,

$$s = f(z - T),$$

unde s este starea NF, f este funcția de activare, iar T este pragul,

$$y_j = g_j(s), \quad j = 1, \dots, M,$$

unde g_j , $j = 1, \dots, M$ sunt funcțiile de ieșire ale NF reprezentând gradele de apartenență ale lui $\{x_1, \dots, x_N\}$ la cele M mulțimi fuzzy. În general, ponderile, pragul și funcțiile de ieșire sunt modificabile în timpul procesului de instruire. Funcția de agregare și funcția de activare sunt fixe. Se pot defini multe tipuri de NF prin schimbarea funcțiilor f și h .

Definim acum patru tipuri de NF.

1. un *NF de intrare* este un NF folosit în stratul de intrare a unei RNF (rețele de neuroni fuzzy), având doar o singură intrare x , astfel încât $z = x$.
2. un *MAX-NF* are funcția de agregare:

$$z = \max_{i=1, \dots, N}(w_i x_i).$$

3. un *MIN-NF* are funcția de agregare:

$$z = \min_{i=1, \dots, N}(w_i x_i).$$

4. un *COMP-NF* (NF competitiv) are pragul variabil și o singură ieșire, astfel încât:

$$y = g(s - T) = \begin{cases} 0 & \text{dacă } s < T \\ 1 & \text{dacă } s \geq T \end{cases}$$

$$T = t(c_1, c_2, \dots, c_K),$$

unde s este starea NF, t este funcția prag, iar c_k , $k = 1, \dots, K$ sunt variabile competitive ale NF.

9.2.2 Structura unei RNF

RNF propusă¹ este pe patru straturi. Primul strat, de intrare, folosește neuroni fuzzy de intrare. Presupunând că dorim să recunoaștem 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:

$$\begin{aligned} s_{ij}^{[1]} &= z_{ij}^{[1]} = x_{ij} & \text{pentru } i = 1, \dots, N_1, \quad j = 1, \dots, N_2 \\ y_{ij}^{[1]} &= \frac{s_{ij}^{[1]}}{P_{vmax}} & \text{pentru } i = 1, \dots, N_1, \quad j = 1, \dots, N_2 \end{aligned}$$

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 funcția pondere $w(m, n)$. Starea celui de-al (p, q) -lea MAX-NF din al doilea strat este:

$$s_{pq}^{[2]} = \max_{i=1, \dots, N_1} (\max_{j=1, \dots, N_2} (w(p-i, q-j)y_{ij}^{[1]})),$$

pentru $p = 1, \dots, N_1$ și $q = 1, \dots, N_2$, unde $w(p-i, q-j)$ este ponderea conexiunii dintre al (i, j) -lea NF de intrare din primul strat și al (p, q) -lea MAX-NF din al doilea strat:

$$w(m, n) = e^{-\beta^2(m^2+n^2)}$$

pentru $m = -(N_1 - 1), \dots, (N_1 - 1)$ și $n = -(N_2 - 1), \dots, (N_2 - 1)$. Prin această funcție 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" și pixelii vecini. Parametrul β determină câți pixeli "vede" un NF, aceasta fiind decis prin algoritmul de instruire. Funcția w se numește și *funcție de fuzzificare*. Fiecare MAX-NF din acest strat are M ieșiri diferite, câte una pentru fiecare NF din al treilea strat. Ieșirile celui de-al (p, q) -lea MAX-NF din al doilea strat sunt:

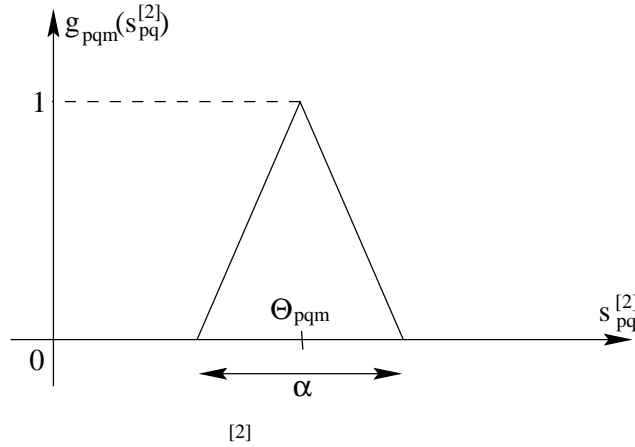
$$y_{pqm}^{[2]} = g_{pqm}(s_{pq}^{[2]}),$$

pentru $p = 1, \dots, N_1$, $q = 1, \dots, N_2$, $m = 1, \dots, M$, unde $y_{pqm}^{[2]}$ este a m -a ieșire a celui de-al (p, q) -lea MAX-NF către al m -lea MIN-NF din stratul trei. Funcția de ieșire g_{pqm} se determină prin algoritmul de instruire. Pentru simplitate, putem alege triunghiuri isoscele de înălțime 1 și bază α (fig. 9.3).

Avem:

$$y_{pqm}^{[2]} = g_{pqm}(s_{pq}^{[2]}) = \begin{cases} 1 - \frac{2|s_{pq}^{[2]} - \theta_{pqm}|}{\alpha} & \text{pentru } |s_{pq}^{[2]} - \theta_{pqm}| \leq \frac{\alpha}{2} \\ 0 & \text{în rest} \end{cases},$$

¹Kwan, 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: Funcția de ieșire g_{pqm} .

pentru $\alpha \geq 0$, $p = 1, \dots, N_1$, $q = 1, \dots, N_2$, $m = 1, \dots, M$, unde θ_{pqm} este mijlocul bazei triunghiului. Prin algoritmul de instruire se determină pentru fiecare p , q , m valorile corespunzătoare pentru α și θ_{pqm} .

Al treilea strat constă din M neuroni de tip MIN-NF. Fiecare din acești neuroni reprezintă un pattern învățat. Deci, M poate fi determinat doar la încheierea procesului de instruire. Ieșirea celui de-al m -lea MIN-NF este:

$$y_m^{[3]} = s_m^{[3]} = \min_{p=1, \dots, N_1} (\min_{q=1, \dots, N_2} (y_{pqm}^{[2]}))$$

pentru $m = 1, \dots, M$.

Al patrulea strat este stratul de ieșire și constă din M neuroni de tip COMP-NF, câte unul pentru fiecare pattern învățat. Dacă un pattern de intrare este cel mai asemănător cu al m -lea pattern învățat, atunci ieșirea celui de-al m -lea COMP-NF este 1, iar celelalte ieșiri sunt 0. Avem:

$$\begin{aligned} s_m^{[4]} &= z_m^{[4]} = y_m^{[3]} \text{ pentru } m = 1, \dots, M \\ y_m^{[4]} &= g[s_m^{[4]} - T] = \begin{cases} 0 & \text{dacă } s_m^{[4]} < T \\ 1 & \text{dacă } s_m^{[4]} = T \end{cases} \text{ pentru } m = 1, \dots, M \\ T &= \max_{m=1, \dots, M} (y_m^{[3]}) \text{ pentru } m = 1, \dots, M \end{aligned}$$

unde T este pragul de activare pentru toți 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 și M . Definim T_f , $0 \leq T_f \leq 1$, ca toleranța 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 și $N_1 \times N_2$ neuroni de tip MAX-NF în al doilea strat. Alege o valoare pentru α , $\alpha \geq 0$ și 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 și al M -lea COMP-NF în al patrulea strat.

$$\theta_{pqM} = s_{pq}^{[2]} = \max_{i=1, \dots, N_1} (\max_{j=1, \dots, N_2} (w(p-i, q-j)x_{ijk}))$$

pentru $p = 1, \dots, N_1$ și $q = 1, \dots, N_2$, unde θ_{pqM} este mijlocul bazei triunghiului pentru cea de-a M -a funcție de ieșire a celui de-al (p, q) -lea MAX-NF în stratul 2. $X_k = \{x_{ijk}\}$, $i = 1, \dots, N_1$, $j = 1, \dots, N_2$ 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 rețea al k -lea pattern și calculează ieșirea RNF curente, cu M neuroni în straturile 3 și 4.

$$\sigma = 1 - \max_{j=1, \dots, M} (y_{jk}^{[3]}),$$

unde $y_{jk}^{[3]}$ este ieșirea 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 și 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 mulți NF din stratul 2 vor fi afectați 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 își pierde abilitatea de a recunoaște 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 recunoaștere 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 învățat. Funcția de ieșire g_{pqm} pentru neuronii din cel de-al doilea strat este, de fapt, o funcție de apartenență fuzzy: $g_{pqm}(s_{pq}^{[2]})$ arată gradul de apartenență al pattern-ului de intrare la clasa reprezentată de al m -lea pattern învățat, apartenența referindu-se la vecinătățile corespunzătoare ale pixelului (p, q) .

Neuronii din al treilea strat determină similaritățile dintre pattern-ul de intrare și pattern-urile învățate. 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 $m = 1, \dots, M$. Valoarea $y_m^{[3]}$ măsoară similaritatea pattern-ului de intrare cu al m -lea pattern învățat. Dacă pattern-ul de intrare a fost deja învățat, va exista un m pentru care $y_m^{[3]} = 1$. Dacă pattern-ul de intrare nu a fost învățat, toate valorile $y_m^{[3]}$, $m = 1, \dots, M$ vor fi mai mici decât 1. Valoarea lui α trebuie să acopere toate valorile posibile la nivel de pixel.

Stratul de ieșire este folosit pentru defuzzificare. Deci, procedura de recunoaștere prin RNF constă din:

1. datele de intrare (stratul 1)
2. fuzzificare (stratul 2)
3. deducție fuzzy (stratul 3)
4. defuzzificare (stratul 4)

Straturile 3 și 4 sunt construite prin instruirea rețelei. Dacă un pattern de intrare este similar cu un pattern deja învățat, el este tratat ca acel pattern, fără a mai fi învățat și el. În caz contrar, pattern-ul de intrare este învățat. Procesul de instruire poate fi continuu: RNF nu trebuie în mod necesar instruită complet înainte de a fi folosită; ea poate să fie instruită și pe parcursul folosirii.

Parametrii α , β și T_f trebuie fixați convenabil.

9.2.5 Rezultatele simulării

În implementarea autorilor, literele și cifrele au fost construite ca pattern-uri de 16×16 pixeli, pixelii având doar valori binare. Pattern-urile au fost apoi mișcate prin deplasări în cele opt direcții cu unul și doi pixeli. Cele 36 de pattern-uri corecte și câteva din pattern-urile obținute prin deplasări au fost folosite pentru a instrui rețeaua.

Pentru a testa performanțele rețelei, s-au folosit pattern-uri perturbate prin: mărire/micșorare, subțiere/îngroșare, adăugarea/extragerea unor mici porțiuni 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 recunoașterea 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. Recunoașterea este și ea mai rapidă. Să observăm că această RNF învață nesupervizat, fiind în esență un algoritm de clustering. Ulterior instruirii, RNF este folosită ca un clasificator. Există variante ale acestei RNF care învață în mod supervizat².

²Cai, 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 performanțele sistemelor biologice, chiar parțial, nu pot ignora modul în care acestea s-au dezvoltat, cum au evoluat.

Algoritmii genetici sunt mijloace prin care mașinile pot emula mecanismele selecției naturale. Acești algoritmi sunt simpli, robuști și generali. Cu rețelele neurale au în comun următoarele:

- nu sunt necesare cunoștințe asupra spațiului 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 și rețelele neurale se aplică împreună. De exemplu, algoritmii genetici se pot aplica pentru accelerarea procesului de instruire a rețelelor neurale. În mod ciudat, printr-un algoritm genetic se poate obține algoritmul genetic care rezolvă o problemă dată.

10.1 Introducere

Metodele convenționale de optimizare se bazează pe ajustarea parametrilor unui model pentru a produce un rezultat dorit. La rețelele neurale, prin instruire se modifică iterativ ponderile conexiunilor, astfel încât să se producă relația dorită între intrări și ieșiri. De remarcat că este menținută o singură soluție care este optimizată în mod iterativ.

Algoritmii genetici operează optimizări conform tehnicilor cunoscute în genetica biologică: se modifică și se mențin caracteristicile unei *populații* de soluții (indivizi) pe parcursul unui mare număr de generații. Acest proces produce populații succesive având un număr din ce în ce mai mare de indivizi cu caracteristicile dorite. Ca și în natură, procesul este probabilistic, dar nu complet aleator. După cum vom vedea, regulile geneticii rețin caracteristicile dorite prin

maximizarea probabilității de proliferare a soluțiilor (indivizilor) care au aceste caracteristici.

Genetica nu optimizează, ci îmbunătățește. Un individ nu trebuie să fie optim pentru a supraviețui; el trebuie să fie superior altor indivizi din populația respectivă.

Algoritmii genetici operează asupra unor codificări de parametri, nu direct asupra parametrilor. Codificările sunt șiruri de lungime finită, de obicei binare. Fiind dată o populație inițială de șiruri, un algoritm genetic produce o nouă populație de șiruri folosind o mulțime de reguli genetice. Regulile sunt astfel construite încât această nouă generație tinde să aibă șiruri superioare celor din generația precedentă, din punct de vedere al unei funcții obiectiv.

10.2 Exemplu

Dorim să minimizăm valoarea unui șir binar, considerat aici ca un număr binar. Vom efectua următorii pași:

1. inițializare
2. reproducere
3. încrucișare
4. mutație
5. repetă pașii 2-4.

Inițializare

Fiecare șir este generat ca un număr binar aleator. Pornim cu următoarele șase șiruri:

i	șirul 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
<u>Total</u>			0,2716	1

Ca funcție obiectiv se alege $(\text{val. zecimală})^{-1}$.

Reproducere prin selecție

Se obține o a doua generație de șiruri care au, în medie, funcția obiectiv mai mare. Fiecare șir A_i este copiat în generația următoare de un număr de ori care

este proporțional cu p_i , unde:

$$p_i = \frac{f_i}{\sum_j f_j},$$

f_i fiind funcția obiectiv, iar p_i este funcția obiectiv relativă. "Discul norocului" (fig. 10.1) este reprezentarea grafică a acestor funcții.

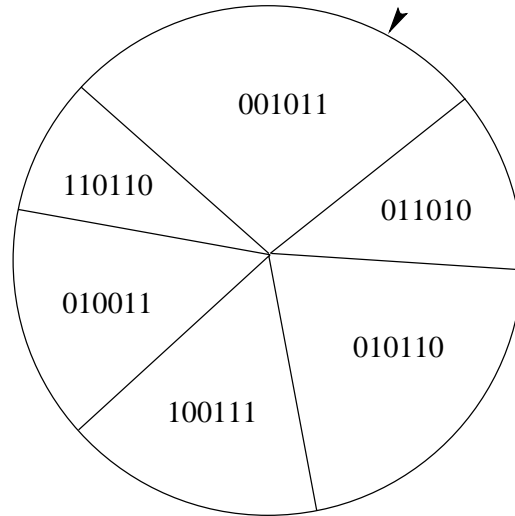


Figura 10.1: "Discul norocului". Secțiunile sunt proporționale cu p_i .

Pentru fiecare copiere în generația următoare, se copiază șirul la care s-a oprit discul. Se fac șase copieri, pentru ca a doua generație să fie la fel de mare:

011010
001011
001011
010011
100111
010110

Primul șir se reproduce, în medie, de $6 \cdot 0,412 = 0,71$ ori deci, prin rotunjire, o dată.

Încrucișare

Se simulează schimbul de material genetic care are loc în timpul reproducerii biologice. În mod aleator, se obțin perechi de șiruri. Pentru simplitate, vom încrucișa perechile adiacente. Pentru fiecare pereche se obține un număr întreg aleator între 1 și 6. Acesta arată câți biți din dreapta se schimbă între cele două șiruri. De exemplu, încrucișăm șirurile A_1 și A_2 , având întregul aleator 2:

$$\left\{ \begin{array}{l} 011010 \\ 001011 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} 011011 \\ 001010 \end{array} \right\}.$$

Procesul continuă pentru șirurile A_3 și A_4 , având întregul aleator 4 și pentru șirurile A_5 și A_6 având întregul aleator 3. Obținem șirurile:

011011
001010
000011
011011
100110
010111

Mutație

Biologic, ea perturbă caracteristicile populației, în mod aleator, prevenind degenerările. Cele mai multe mutații mai mult distrug decât să fie benefice. De aceea, mutația trebuie să fie rară, pentru a nu distruge specia. Pe de altă parte, ea e necesară. În simularea noastră, vom presupune o rată a mutației de 1/1000 biți, mutația însemnând aici o inversare de bit. Deoarece avem 36 de biți în populație, probabilitatea unei mutații este 0,036. Presupunem că nu are loc nici o mutație la acest pas. Obținem astfel a doua generație:

i	șirul 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
<u>Total</u>			0,576	1

Observăm că totalul funcției obiectiv a crescut vizibil.

Întregul proces se reia, obținând următoarele generații.

În general, populația poate să crească sau să scadă de la o generație la alta.

Acest proces continuă până când funcția obiectiv atinge o valoare acceptabilă pentru o anumită generație. Convergența este, în general, rapidă.

Se poate întâmpla să ajungem la o degenerare, de exemplu dacă cele șase șiruri sunt 000011. De aici putem ieși doar așteptând o mutație favorabilă. Aceasta poate avea loc doar după multe generații. Astfel se poate întâmpla și în natură, unde o specie poate suferi o adaptare neoptimă pentru milenii, până când apare un organism mai bine adaptat care ajută la perfecționarea speciei.

10.3 Fundamente matematice

O *schemă* este o submulțime de șiruri cu similarități în anumite poziții. Vom considera șiruri peste alfabetul $V = \{0, 1\}$. Un șir va fi notat astfel: $A = a_1 a_2 a_3 \dots a_l$, unde a_i poate fi 0 sau 1, iar l este lungimea șirului. O schemă H peste alfabetul

$V+ = \{0, 1, *\}$ este, de exemplu, $H = *11 * 0 * *$. Schema H este, de pildă, reprezentată de șirul 0110000.

Dacă V are k șiruri, atunci se pot forma $(k + 1)^l$ scheme. Fiecare șir este conținut în 2^l scheme deoarece, fiecare caracter din șir poate avea fie valoarea pe care o are, fie $*$.

Deci, într-o populație de n șiruri, există cel mult $n \cdot 2^l$ scheme și cel puțin 2^l scheme reprezentate efectiv în populație, acest număr depinzând de diversitatea populației. *Ordinul* unei scheme H , $o(H)$, este numărul pozițiilor fixate din H . De exemplu, $o(011 * 1 * *) = 4$.

Lungimea de definiție a unei scheme H , $\delta(h)$, este distanța dintre prima și ultima poziție fixată. De exemplu, $\delta(011 * 1 * *) = 4$.

Vom considera efectele operațiilor de reproducere, încrucișare și mutație asupra schemelor reprezentate într-o populație de șiruri.

Reproducție

Presupunem că la momentul t există m reprezentanți (m șiruri) ai schemei H în generația $A(t)$. Vom scrie: $m = m(H, t)$. Conform algoritmului de reproducție, un șir A_i este copiat în generația următoare cu probabilitatea $p_i = \frac{f_i}{\sum f_j}$. Presupunem că selectăm n șiruri distincte din $A(t)$ pentru următoarea generație, $A(t + 1)$, 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 funcției obiectiv pentru șirurile care reprezintă schema H la momentul t .

Fie $\bar{f} = \frac{\sum f_j}{n}$ valoarea medie a funcției obiectiv pentru întreaga populație ($A(t)$ are n șiruri). Atunci:

$$m(H, t + 1) = m(H, t) \frac{f(H)}{\bar{f}}$$

O schemă H crește deci direct proporțional cu raportul:

$$\frac{\text{valoarea medie a funcției obiectiv pentru schema } H}{\text{valoarea medie a funcției obiectiv pentru întreaga populație}}$$

Efectul reproducerii este clar: din punct de vedere al numărului de reprezentanți, o schemă H cu $f(H) > \bar{f}$ crește, 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 $t = 0$, obținem:

$$m(H, t) = m(H, 0)(1 + c)^t$$

adică, obținem o progresie geometrică. *Creșterea / scăderea unei scheme prin reproducție este exponențială.*

Încrucișare

Fie $l = 7$ și fie:

$$\begin{aligned} A &= 0111000 \\ H_1 &= *1****0 \\ H_2 &= ***10** \end{aligned}$$

Șirul A reprezintă schemele H_1 și H_2 . "Tăietura" după care se execută încrucișarea se alege aleator, echiprobabil între cele 6 posibilități. De exemplu:

$$\begin{aligned} A &= 011|1000 \\ H_1 &= *1*|***0 \\ H_2 &= ***|10** \end{aligned}$$

Încrucișarea are loc prin schimbarea primelor 3 caractere din A . Observăm ca H_1 este distrusă, deoarece $*1*$ și $** *0$ vor fi plasate în șiruri diferite. H_2 supraviețuiește 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 supraviețuiască este

$$p_s = 1 - p_d = \frac{1}{6}$$

. Similar, pentru H_2 , $p_d = \frac{1}{6}$.

Nu am considerat deocamdată situația în care, prin operația de încrucișare, pozițiile fixate din schemă nu se modifică.

În general,

$$p_s = \frac{1 - \delta(H)}{l-1}.$$

Fie p_c probabilitatea ca să efectuăm operația de încrucișare. Avem:

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1}.$$

Se folosește semnul \geq pentru că acum ținem cont și de situația în care, prin operația de încrucișare, pozițiile fixate nu se modifică.

Considerăm acum efectul combinat al reproducției și încrucișării. Presupunând că aceste două operații sunt independente obținem:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{f} \left(1 - p_c \frac{\delta(H)}{l-1} \right).$$

Observăm că *schemele cu lungime mică sunt favorizate.*

Mutație

Alterăm o poziție fixată cu probabilitatea p_m , deci supraviețuiește cu probabilitatea $1 - p_m$. O schemă H supraviețuiește dacă fiecare dintre cele $o(H)$ poziții fixate supraviețuiește. Rezultă că probabilitatea ca o schemă H să supraviețuiască 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 inegalității Bernoulli: Oricare ar fi $a \geq -1$ și n natural, avem:

$$(1 + a)^n \geq 1 + na.$$

Ținând cont de toate operațiile genetice, obținem (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 funcției obiectiv peste medie, cu lungimea de definiție și ordinul mici, cresc exponențial în generațiile următoare.

Am văzut că într-o populație de n șiruri de lungime l sunt procesate între 2^l și $n2^l$ scheme. Nu toate aceste scheme sunt procesate cu o probabilitate mare, datorită operațiilor de încrucișare și mutație (schemele lungi sunt distruse).

Se poate demonstra că numărul de scheme efectiv procesate este în ordinul lui n^3 , ținând cont și de operațiile de încrucișare și mutație.

Cu alte cuvinte, cu toate că se procesează n șiruri la fiecare generație, un algoritm genetic procesează de fapt în ordinul lui n^3 scheme. Această proprietate definește *paralelismul implicit* al unui algoritm genetic. Manipulând n elemente, procesăm de fapt aproximativ n^3 similarități.

Acestui paralelism implicit i se datorează faptul că spațiul căutărilor este suficient de mare pentru ca algoritmul să nu ducă, în general, la un optim local ci la unul global.

Fiind dată o populație de șiruri, cu valorile respective ale funcției obiectiv, ne punem problema de ce informație dispunem pentru a ghida căutarea unor șiruri mai bune. Răspunsul este: similaritățile dintre șiruri. Aceste similarități sunt descrise de schemele populației.

10.4 Exerciții

1. (C) Minimizați funcția $f(x, y, z) = x^2 + y^2 + z^2$, $x, y, z \in [-512, 512]$. Folosiți șiruri binare de lungime 10.

Capitolul 11

Puterea și complexitatea de calcul a rețelelor neurale

11.1 Mașina Turing

Mașina Turing este un model abstract de calculabilitate. O mașină Turing este formată dintr-o bandă divizată în celule, infinită la dreapta și dintr-un cap de citire/scriere care la un moment dat este poziționat exact asupra unei celule. Fiecare celulă poate conține cel mult un simbol. În plus, există o mulțime finită de stări care controlează mutările mașinii. La o mutare, mașina Turing, în funcție de simbolul din celula de bandă pe care este poziționat capul de citire/scriere și de starea în care se află mașina, efectuează următoarele:

1. Trece într-o nouă stare.
2. Scrie un simbol în celula pe care mașina se află poziționată, înlocuind ceea ce era scris înainte (eventual poate tipări același simbol).
3. Deplasează capul de citire o celulă la dreapta sau stânga, sau lasă capul de citire/scriere poziționat asupra aceleiași celule.

Inițial, capul de citire/scriere este poziționat asupra celei mai din stânga celule. Primele n celule din stânga, pentru un $n \geq 0$, conțin simboluri care reprezintă datele de intrare, iar restul de celule, în număr infinit, conțin un simbol special ce nu poate fi folosit pentru reprezentarea datelor de intrare (de obicei, blank).

Mașina pornește dintr-o stare precis determinată, numită *stare inițială*. Unele stări din mulțimea finită a stărilor se numesc *stări finale*. Stările finale sunt de două tipuri: de acceptare și de respingere.

Mulțimea simbolurilor formează un *alfabet* (finit). Secvențele finite de simboluri se numesc *cuvinte*. O mașină M *acceptă* un cuvânt w dacă, pornind din starea inițială, cu w ca dată de intrare, ajunge, după un număr finit de pași, într-o stare finală de acceptare.

Limbaajul acceptat de o mașină M este mulțimea (finită sau infinită):

$$L(M) = \{w | M \text{ acceptă } w\}.$$

Mulțimea limbajelor acceptate de mașini Turing formează *clasa limbajelor recursiv enumerabile*, \mathcal{L}_0 .

În cazul în care numărul de pași efectuați este finit, modul de operare al unei mașini Turing este descris de către un algoritm.

În afară de imaginea mașinii Turing ca și o procedură (algoritm - dacă este finit) de recunoaștere a unei mulțimi, putem considera că un astfel de dispozitiv calculează o funcție adăugând o bandă de scriere. O astfel de mașină calculează o funcție naturală f dacă, pornind cu data de intrare x , se oprește cu $f(x)$ pe banda de scriere. Funcția f se numește, în acest caz, *Turing-calculabilă*.

Calculabilitatea intuitivă este un concept prematematic care nu permite nici o restricție inevitabil impusă de o definiție riguroasă.

Orice funcție Turing-calculabilă este și intuitiv calculabilă. Reciproca acestei afirmații este mai dificil de formulat, deoarece identifică două obiecte din care unul este matematic iar unul prematematic.

Teza lui Church-Turing (1936). Orice funcție intuitiv calculabilă este Turing-calculabilă.

Teza lui Church-Turing nu poate fi demonstrată, ci doar justificată prematematic. Cu alte cuvinte, este greu de acceptat existența unor funcții naturale intuitiv calculabile care să nu fie Turing-calculabile.

În conformitate cu teza lui Church-Turing, clasa mulțimilor \mathcal{L}_0 coincide cu clasa mulțimilor algoritmic calculabile.

Teoremă (Turing, 1936). Problema opririi mașinii Turing este nerezolvabilă.

Cu alte cuvinte, nu există un algoritm care să determine dacă o mașină Turing arbitrară, acționată dintr-o configurație arbitrară (q, w, i) se va opri sau nu (q este starea, w este data de intrare, iar i este poziția capului).

Puterea de calcula a mașinilor Turing este foarte mare: orice algoritm, chiar și NP, poate fi implementat pe o mașină Turing corespunzătoare. O mașină Turing poate rezolva orice problemă rezolvabilă pe un calculator de uz general, având avantajul că poate fi descrisă matematic foarte concis.

Mașinile Turing pot fi *deterministe* sau *nedeterministe*. Mulțimea limbajelor acceptate de mașinile nedeterministe este, însă, egală cu mulțimea limbajelor acceptate de mașinile deterministe.

Un algoritm NP poate fi rezolvat într-un timp polinomial pe o mașină Turing nedeterministă. Dacă se dorește rezolvarea pe o mașină deterministă, timpul nu mai rămâne polinomial.

Un algoritm P poate fi rezolvat într-un timp polinomial pe o mașină Turing deterministă.

11.2 Puterea de calcul a rețelelor neurale

Un *model masiv paralel* (MMP) este o mulțime de celule interconectate printr-o rețea de comunicație. 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, modelele masiv paralele sunt: rețelele neurale, automatele celulare și rețelele de automate.

În cadrul mașinilor Turing, o problemă este *rezolvabilă algoritmic* dacă există un algoritm având la intrare un cuvânt finit w și la ieșire un cuvânt care reprezintă informația determinată de w . Spre deosebire de aceasta, un MMP rezolvă *probleme de configurație*, având la intrare o configurație x a modelului și la ieșire o configurație a modelului care reprezintă informația determinată de x .

Generalizăm acum noțiunea de rezolvabilitate pentru MMP.

Definiție. O problemă de configurație este *p-rezolvabilă* printr-un model masiv paralel M dacă, indiferent de configurația de intrare x , există un moment t astfel încât avem:

$$\begin{array}{ll} \text{stabilitate:} & T(t', x) = T(t, x), \text{ pentru } t' > t \\ \text{corectitudine:} & T(t', x) \text{ este exact ieșirea dorită.} \end{array}$$

Prin T am notat funcția care definește dinamica globală a modelului.

Câteva rezultate fundamentale

McCulloch și Pitts afirmă în 1943 că rețelele neurale sunt calculatoare universale.

Franklin și Garzon au arătat că orice problemă rezolvabilă algoritmic (printr-o mașină Turing) poate fi efectiv formulată sub forma unei probleme de configurație p-rezolvabilă pe o rețea neurală Așadar, *rețelele neurale sunt ce puțin la fel de puternice ca și mașinile Turing.*

Configurațiile unui MMP pot fi însă infinite, deci p-rezolvabilitatea este o noțiune mai complexă decât calculabilitatea. Franklin și Garzon au demonstrat că *problema opririi este p-rezolvabilă pe o rețea neurală infinită.* Această problemă nu este însă rezolvabilă pe o mașină Turing, după cum am văzut. Conform acestui rezultat, dar și al altor rezultate de acest fel, s-au demonstrat de către Garzon și Franklin¹ următoarele relații:

$$\text{mașini Turing} \subset \text{automate celulare} \subseteq \text{rețele neurale} \subseteq \text{rețele de automate}$$

¹Garzon, M., S. Franklin "NeuralComputability II", Report of Memphis State University, MSU-TR-89-12, 1989.

11.3 Reprezentarea funcțiilor booleene prin rețele feedforward

Orice funcție finită poate fi implementată exact printr-o rețea neurală feedforward, cu un strat ascuns, de perceptroni de tip continuu (Ito ²)

Prin definiție, o funcție finită este de forma $f : \mathcal{Q} \rightarrow \mathcal{R}$, unde $\mathcal{Q} \subset \mathcal{R}^d$, \mathcal{Q} finită.

Orice funcție continuă poate fi aproximată oricât de bine de o rețea neurală feedforward cu un singur strat ascuns, de perceptroni de tip continuu (Funahashi³, Hecht-Nielsen ⁴).

Să considerăm cazul funcțiilor booleene. Domeniul de definiție al unei funcții booleene cu N argumente conține 2^N elemente. Pentru fiecare din aceste 2^N elemente există 2 valori posibile ale funcției. Deci, există 2^{2^N} funcții booleene diferite cu N argumente. Vom reprezenta o funcție booleeană arbitrară printr-o rețea feedforward cu un strat ascuns. Aceasta nu surprinde, o funcție booleeană fiind un caz particular de funcție finită. Primul strat va fi format din N perceptroni discreți $\sigma_1, \dots, \sigma_N$, reprezentând argumentele funcției. Ultimul strat constă dintr-un neuron discret S , reprezentând valoarea funcției $F(\sigma_1, \dots, \sigma_N)$. Valorile logice sunt reprezentate de ± 1 . În stratul ascuns sunt 2^N neuroni discreți q_0, \dots, q_{2^N-1} . Ponderea w_{jk} dintre σ_k și q_j este $\pm w$, calculată astfel: fie $(\alpha_1, \dots, \alpha_N)$ reprezentarea binară a lui j . Luăm $w_{jk} = +w$ dacă $\alpha_k = 1$ și $w_{jk} = -w$ dacă $\alpha_k = 0$. Adică, $w_{jk} = (2\alpha_k - 1)w$, unde w este o constantă pozitivă. Alegem pragul pentru neuronii ascunși: $(N - 1)w$. Potențialul 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 și numai dacă

$$(2\alpha_k - 1)\sigma_k = 1 \text{ pentru } k = 1, \dots, N,$$

unde $2\alpha_k - 1 = \pm 1$ și $\sigma_k = \pm 1$. Deci, condiția este:

$$2\alpha_k - 1 = \sigma_k \text{ pentru } k = 1, \dots, N.$$

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 $(\frac{\sigma_1+1}{2}, \dots, \frac{\sigma_N+1}{2})$.

²Ito, Y. "Finite Mapping by Neural Networks and Truth Functions", Math. Scientist, 17, 1992, 69-77.

³Funahashi, K. "On the Approximate Realization of Continuous Mapping by Neural Networks", Neural Networks, 2, 1989, 183-192.

⁴Hecht-Nielsen, R. "Theory of the Backpropagation Neural Network", In: "Neural Networks for Perception", H. Wechsler (Ed.), Academic Press, 1992, 65-93.

Avem, deci, următoarele ieșiri din neuronii ascunși:

$$s_j = \begin{cases} +1 & \text{pentru } j = j_0 \\ -1 & \text{pentru } j \neq j_0 \end{cases} .$$

Ponderea v_j dintre s_j și neuronul S va fi:

$$v_j = \begin{cases} +1 & \text{dacă } F(\sigma_1, \dots, \sigma_N) = true \\ -1 & \text{dacă } F(\sigma_1, \dots, \sigma_N) = false \end{cases} ,$$

unde $(\sigma_1, \dots, \sigma_N)$ corespunde lui s_j . Luăm pragul:

$$-\sum_j v_j$$

. Potențialul pentru neuronul S este:

$$k = \sum_j v_j s_j + \sum_j v_j = \sum_j v_j (s_j + 1) = 2v_{j_0} .$$

Ieșirea din rețea va fi $\text{sgn}(h) = F(\sigma_1, \dots, \sigma_N)$ (fig. 11.1).

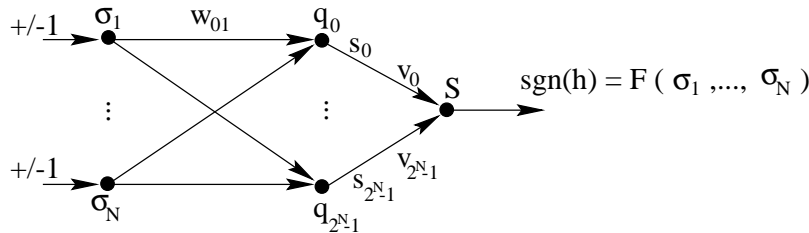


Figura 11.1: Rețea neurală care implementează funcția $F(\sigma_1, \dots, \sigma_N)$.

Se observă că numărul de neuroni ascunși este 2^N , deci foarte mare. Pe de altă parte, problema reprezentării unei funcții booleene (SATI) este NP-completă (teorema lui Cook). Deci, este explicabil. De fapt, am folosit forma normală disjunctivă, cu o poartă OR și 2^N porți AND.

În cazuri particulare, reprezentarea se poate face însă mult mai eficient (cu mai puțini neuroni). De exemplu, XOR su 2 argumente se poate reprezenta cu 2, nu 4 neuroni ascunși.

Apare problema *minimizării* rețelei neurale care reprezintă o funcție booleană dată. Pentru rețelele cu un strat ascuns, corespunzând funcțiilor booleene în forma normală conjunctivă, problema minimizării este cunoscută: se minimizează funcția booleană printr-unul din algoritmi uzuali (Karnaugh sau Quine - McCluskey). Minimizarea se referă aici la numărul de neuroni, adică numărul de porți. O problemă cu totul diferită este dacă o funcție booleană cu N argumente poate fi "învățată" de către o rețea neurală, într-un timp polinomial. Valiant⁵ a

⁵Valiant, L.G. "A Theory of the Learnable", Commun. ACM, 27, 1134, 1984.

arătat că funcțiile logice sub formă normală conjunctivă pot fi instruite folosind doar un număr polinomial de exemple pozitive. În cazul funcțiilor în formă normală disjunctivă mai este nevoie, în plus, de încă un nivel de învățare.

O problemă interesantă este dacă funcția poate fi învățată folosind doar puține exemple și lăsând-o apoi să "generalizeze".

11.4 Complexitatea instruirii rețelelor neurale feedforward

Fie cazul rețelelor feedforward multistrat cu perceptroni de tip continuu sau discret (nu are importanță) și fie cazul învățării supervizate.

Teoremă (Judd⁶). Problema generală a instruirii unei astfel de rețele este NP-completă.

Cu alte cuvinte, presupunând $P \neq NP$, nu există un algoritm cu timp polinomial care să instruiască o rețea în condițiile în care:

1. Arhitectura rețelei (neuronii și conexiunile, fără ponderi) este arbitrară.
2. Numărul de pattern-uri care trebuie memorate în rețea este arbitrar.

Mărimea unui caz este aici proporțională cu mărimea rețelei (numărul de perceptroni, conexiuni) plus numărul pattern-urilor de memorat.

Dacă, de exemplu, pentru a accelera instruirea, mărim numărul de neuroni în rețea, timpul de execuție crește foarte mult, deoarece timpul poate crește exponențial față de numărul de neuroni.

Teorema rămâne valabilă indiferent de funcția de activare folosită. De asemenea, teorema rămâne valabilă chiar dacă arhitectura rețelelor considerate verifică restricția:

$$\text{nr. de straturi ascunse} \leq 1.$$

Observații:

1. Această teoremă se referă la instruire în general. Există clase mari de rețele și secvențe de instruire care duc la o instruire în timp polinomial. Problema generală a instruirii unei rețele cu o funcție 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: rețelele cu straturi puține se instruiesc mai eficient. Teorema lui Judd ne arată că nu este neapărat așa. Importantă este mărimea rețelei, complexitatea ei.

⁶Judd, 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 șiruri se face în timp liniar, pe o mașină von Neumann, adică mult mai rapid. Este, însă, adevărat că astfel se pierd avantajele procesării neurale, de exemplu invarianța la perturbații.
4. Rămâne ca subiect de cercetare, din punct de vedere al NP-completitudinii, problema instruirii rețelelor recurente.

Capitolul 12

Considerații epistemologice asupra calculului neural

12.1 Scopul unei rețele neurale

Formele superioare de viață, inclusiv cele bazate pe sisteme nervoase centrale, tind să faciliteze supraviețuirea speciei respective în condiții de mediu care devin din ce în ce mai dificile¹.

Natura nu își folosește niciodată în mod inutil resursele, ci acționează conform unui principiu de optimalitate. De aceea, putem afirma că mecanismele creierului tind să asigure în mod optim supraviețuirea. De exemplu, somnul, în interpretarea lui Freud, tinde să integreze zgomotele, păstrând homeostazia sistemului.

Din această perspectivă, analogiile dintre calculatoarele digitale și rețelele neurale biologice sunt nerezonabile. Problemele aritmetice sunt rareori rezolvate cu scopuri biologice; poate că păsările învață să-și numere ouăle, însă, chiar și această abilitate poate fi explicată, până la un anumit nivel, prin funcții elementare de recunoaștere a formelor. Necesitatea de a număra sclavi și soldați folosind semne este de origine mai târzie în dezvoltarea omului și este realizată pe baza unor mijloace nebiologice. Orice încercare de a construi modele neurale pentru sumatoare, multiplicatoare, convertoare A/D și alte circuite de calcul este de aceea, din acest punct de vedere, irațională.

Cea mai importantă funcție a sistemului nervos este monitorizarea și controlul condițiilor în care organismul trăiește în mediul respectiv. Această funcție se realizează prin modificarea stărilor organismului și/sau a mediului. Cele mai elementare stări controlabile ale organismului sunt globale, nestructurate (de exemplu, temperatura). Cele mai evolute astfel de stări sunt legate de comportament social, care implică planificarea anticipativă și interacțiunile sociale. Chiar și acestea sunt, însă, motivate biologic.

Diferența dintre calculatoare și creier rezultă din scopurile lor diferite. Calcu-

¹Kohonen, T. "Self-Organization and Associative Memory". Springer, Berlin, 3rd ed., 1989, chap. 9.

latoarele au fost dezvoltate inițial pentru a rezolva probleme matematice, mai ales aritmetice. Ulterior, s-a descoperit că instrucțiunile folosite pentru calculul simbolic pot fi utilizate și pentru procesarea limbajului natural. Această descoperire a fost un efect secundar în istoria dezvoltării calculatoarelor. Încă nu s-a observat că o anumită generație de calculatoare să-și dorească să supraviețuiască!

Pentru a modela cât mai bine sistemele neurale biologice, rețelele neurale artificiale trebuie să interacționeze cu lumea reală, cu obiectele naturale. Statistica semnalelor naturale, de exemplu vorbirea, sunt complet diferite de statistica presupusă în teoria matematică a probabilităților. Natura gaussiană a zgomotului și distribuția normală a datelor sunt ipoteze foarte proaste în acest context.

12.2 Funcțiile neurale biologice sunt localizate sau distribuite?

Este unul dintre subiectele de dezbateră continuă. Circuitele neurale trebuie să proceseze funcțiile local, în timp ce distribuirea globală a activităților este o funcție colectivă a activităților tuturor părților.

De fapt, dihotomia distribuire - localizare poate fi urmărită și la nivelul neuronului:

- putem localiza cu acuratețe de 10 micrometri corpul și axonul neuronului
- ramurile dendritelor aceleiași celule sunt însă distribuite pe o arie cu un diametru de câțiva milimetri.

Cu alte cuvinte, răspunsul poate fi localizat, cu toate că funcțiile (adaptive) pot fi distribuite spațial pe o arie mare.

12.3 Este neliniaritatea esențială în calculul neural?

Semnalele procesate în rețelele neurale sunt, de obicei, continue. În electronică, cu toate că, de fapt, caracteristicile componentelor sunt neliniare, prin folosirea feedback-urilor negative și a altor metode de stabilizare, se organizează componentele și circuitele în module integrate care se comportă liniar.

Neuronii biologici au caracteristici și 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 recunoaște o situație importantă, sistemul reticular de activare adaugă în circuitul cortical o excitație 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 funcționeze liniar. În ansamblu, sistemul funcționează, în acest caz, *temporar liniar*. Dacă sistemul reticular de activare

nu recunoaște o situație ca fiind importantă, neuronii rămân inactivi, sub pragul de saturație.

Există și cazuri de neliniaritate. De exemplu, în cazul percepției 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 câțiva centri nervoși, numiți *ganglia*, situați lângă organele senzoriale. Acești centri combină semnalele receptate într-un mod *neliniar*, formând produse de perechi de semnale - unul obținut din figură și unul din fundal. Corelarea perechilor de semnale este diferită în cazurile când figura este, respectiv nu este, în mișcare.

12.4 Deosebirile esențiale dintre calculatoarele neurale și cele digitale

1. *Gradul de paralelism al rețelelor neurale este mai mare decât cel al oricărui calculator digital masiv paralel.*

Calculatoarele masiv paralele sunt rețele de procesoare care lucrează asincron. Aceste calculatoare sunt, totuși, digitale și au circuite digitale în nodurile lor. Deosebirea lor esențială față de calculatoarele von Neumann constă în faptul că este imposibil de prezis ordinea operațiilor realizate în timpul calculului.

Rețelele neurale lucrează tot asincron. Semnalele neurale nu au însă o formă exactă, deci nu este posibil să combinăm nici o informație de control (coduri de control, biți 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. Operațiile implementate pe ele nu pot fi definite algoritmic.

2. *De ce semnalele neurale nu pot fi approximate 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 și a inteligenței artificiale, se bazează pe teoria automatelor finite. Prin feedback (fig. 12.1), un automat finit poate implementa orice secvență de stări, fiind capabil de a defini recursiv funcțiile calculabile.

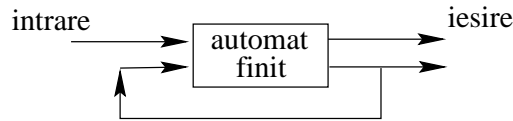


Figura 12.1: Automat finit.

Toate calculatoarele digitale aplică principiul recursivității la diferite nivele, de la hardware la procedurile definite în limbaje de programare. Aplicarea recursivității este posibilă din două motive tehnologice:

- valorile semnalelor sunt stabile
- toate combinațiile de semnale și stări sunt accesibile

O rețea neurală biologică nu poate implementa un automat finit. Funcțiile feedback din rețelele 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 programați pașii individuali de procesare în creier și cum să fie programat un calculator neural ca un calculator digital.

Structurile anatomiche în sine constituie "un program", incluzând și codul genetic. În acest sens, creierul este "programat". Similar, arhitectura rețelei neurale artificiale reprezintă "programul" ei.

În natura biologică, structurile par atât de semnificative, ca și cum ar fi create conform unui plan *a priori*. Explicația științifică este că mutațiile, pe parcursul unei lungi evoluții, duc prin selecție la soluția optimă. Soluțiile optime la care se ajunge sunt însă foarte greu de înțeles, datorită faptului că sunt definite implicit prin codurile genetice. Este imposibil să adoptăm această tehnică în calculul rețelelor neurale artificiale.

Problema principală este cum să obținem răspunsuri corecte din partea mediului fără intervenția și interpretarea programatorului și să le prezentăm sistemului.

Din toate aceste motive, putem afirma că o rețea neurală veritabilă nu poate fi programată. Cele mai utile aplicații ale rețelelor 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 puțin 10^9 ori mai mare decât complexitatea creierului investigat (Starkermann²).

În mod ironic, cunoștințele deja achiziționate sunt pierdute în mod continuu prin moartea indivizilor și trebuie restabilite de către cei tineri.

În concluzie, o identitate nu se poate autopercepe. Creierul biologic nu se poate autoînțelege.

²Starkermann, R. "The Functional Intricacy of Neural Networks - A Mathematical Study", Proceedings, Innsbruck, 1993.

Anexa A

Complemente matematice

A.1 Vectori și 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 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$. Vectorul \mathbf{x} este o *combinație liniară* a vectorilor $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ dacă există scalarii c_1, \dots, c_m astfel încât:

$$\mathbf{x} = \sum_{i=1}^m c_i \mathbf{x}_i.$$

Mulțimea de vectori $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ este *liniar dependentă* dacă există c_1, \dots, c_m , nu toți nuli, astfel încât:

$$\sum_{i=1}^m c_i \mathbf{x}_i = \mathbf{0}.$$

În caz contrar, vectorii $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ sunt *liniar independenți*.

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} și \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} și \mathbf{y} .

Se observă că $\mathbf{x}^t \mathbf{y} = \mathbf{y}^t \mathbf{x}$, deci produsul scalar este comutativ.

Vectorii \mathbf{x} și \mathbf{y} sunt *ortogonali* dacă și numai dacă produsul lor scalar este 0.

Dacă vectorii nenuli $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ sunt doi câte doi ortogonali, atunci sunt *liniar independenți*.

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 și sunt independenți.

Exemplu: Găsiți produsul scalar dintre vectorii \mathbf{x} și \mathbf{y} , lungimea lor și unghiul dintre ei, unde:

$$\mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ -1 \\ 4 \end{bmatrix}$$

Soluție: $\mathbf{x}^t \mathbf{y} = 9$, $\|\mathbf{x}\| = 3$, $\|\mathbf{y}\| = \sqrt{18}$, $\cos \psi = \frac{\mathbf{x}^t \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{1}{\sqrt{2}}$, deci $\psi = 45^\circ$.

A.2 Forme pătratice

O *formă pătratică (FP)* este o funcție $E(\mathbf{x}) = \mathbf{x}^t \mathbf{A} \mathbf{x}$, unde \mathbf{x} este un vector n -dimensional iar \mathbf{A} o matrice simetrică de $n \times n$ elemente.

Dacă $\mathbf{A} = (a_{ij})_{i,j=1,\dots,n}$, atunci:

$$\begin{aligned} E(x_1, x_2, \dots, x_n) &= a_{11}x_1^2 + 2a_{12}x_1x_2 + 2a_{13}x_1x_3 + \dots + 2a_{1n}x_1x_n \\ &+ a_{22}x_2^2 + 2a_{23}x_2x_3 + \dots + 2a_{2n}x_2x_n + \dots + a_{nn}x_n^2 = \\ &= \sum_{i=1}^n \sum_{j=1}^n a_{ij}x_i x_j \end{aligned}$$

Proprietăți ale formelor pătratice

1. $E(\mathbf{x})$ este o *FP pozitiv semidefinită*, iar \mathbf{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 \mathbf{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 \mathbf{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 \mathbf{A} este o matrice *negativ definită* dacă $[(\forall)\mathbf{x} \neq 0 \Rightarrow E(\mathbf{x}) < 0]$.

$E(\mathbf{x})$ este pozitiv definită dacă și numai dacă pentru $\mathbf{A} = (a_{ij})_{i,j=1,\dots,n}$ avem:

$$\begin{aligned} \det \mathbf{A}_{11} &= \|a_{11}\| > 0 \\ \det \mathbf{A}_{22} &= \left\| \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right\| > 0 \\ &\vdots \\ \det \mathbf{A}_{nn} &= \det \mathbf{A} > 0. \end{aligned}$$

Condiții suficiente pentru ca $E(\mathbf{x})$ să fie:

a) FP pozitiv semidefinită:

$$\det A_{11} \geq 0, \det A_{22} \geq 0 \dots \det A_{nn} \geq 0,$$

b) FP pozitiv definită:

$$\det A_{11} > 0, \det A_{22} > 0 \dots \det A_{nn} > 0,$$

c) FP negativ semidefinită:

$$\det A_{11} \leq 0, \det A_{22} \geq 0, \det A_{33} \leq 0 \dots,$$

d) FP negativ definită:

$$\det A_{11} < 0, \det A_{22} > 0, \det A_{33}, \dots$$

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} și negativă pentru alte valori, deci depinde de \mathbf{x} .

A.3 Elemente de geometrie analitică

Ecuția unei drepte în plan este:

$$ax + by + c = 0, \text{ cu } a^2 + b^2 > 0.$$

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 $ax + by + c > 0$.

Ecuția unei drepte a cărei vector normal este $\mathbf{n} = [n_1, n_2]^t$ și 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 și x_2 în plan și vectorii asociați lor, \mathbf{x}_1 și \mathbf{x}_2 , ecuația mediatoarei segmentului dintre x_1 și 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 relația anterioară rezultă

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} - \frac{1}{2}(\mathbf{x}_1 - \mathbf{x}_2)^t (\mathbf{x}_1 + \mathbf{x}_2) = 0$$

și obținem

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}(\mathbf{x}_2^t \mathbf{x}_2 - \mathbf{x}_1^t \mathbf{x}_1) = 0.$$

Din

$$\mathbf{x}^t \mathbf{x} = \|\mathbf{x}\|^2$$

obținem următoarea ecuație a mediatoarei:

$$(\mathbf{x}_1 - \mathbf{x}_2)^t \mathbf{x} + \frac{1}{2}(\|\mathbf{x}_2\|^2 - \|\mathbf{x}_1\|^2) = 0.$$

Ecuația planului care trece prin trei puncte necoliniare este:

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0$$

Ecuația planului care trece printr-un punct $\mathbf{x}(x_1, y_1, z_1)$ și este perpendicular pe vectorul $\mathbf{n} = [a, b, c]^t$ este:

$$a(x - x_1) + b(y - y_1) + c(z - z_1) = 0.$$

Ecuația planului care trece prin mijlocul segmentului $(x_1 y_1 z_1), (x_2 y_2 z_2)$ și 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 Operația XOR

XOR este o operație 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 operația XOR pentru n argumente logice:

$$\text{XOR}(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n.$$

Această funcție se numește și *funcția de paritate* deoarece are valoarea 1 dacă și numai dacă numărul argumentelor care au valoarea 1 este impar.

A.5 Iacobianul și hessianul

Un vector $\mathbf{x}(t)$ dependent de timp se definește astfel:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix} \quad \frac{d\mathbf{x}(t)}{dt} = \begin{bmatrix} \frac{dx_1}{dt} \\ \vdots \\ \frac{dx_n}{dt} \end{bmatrix} = \dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix}$$

Fie funcția $E(\mathbf{x})$, unde \mathbf{x} este un vector cu n componente. *Gradientul* se definește astfel:

$$\nabla_{\mathbf{x}}E(\mathbf{x}) = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix}$$

Proprietăți:

- 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) $\nabla_{\mathbf{x}}(\mathbf{x}^t \mathbf{A} \mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{A}^t \mathbf{x}$
- d) Când lucrăm cu forme pătratice, \mathbf{A} este simetrică și 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 funcții reprezintă direcția creșterii maxime a lui $E(\mathbf{x})$. Gradientul reprezintă tangenta la curbă pe direcția de creștere și este nul în punctele de maxim și minim.

Dacă avem o funcție $E[\mathbf{x}(t)]$ (nu neapărat o formă pătratică), unde \mathbf{x} este un vector de n componente dependente de timp, atunci

$$\frac{dE[\mathbf{x}(t)]}{dt} = \frac{\partial E}{\partial x_1} \cdot \frac{\partial x_1}{\partial t} + \dots + \frac{\partial E}{\partial x_n} \cdot \frac{\partial x_n}{\partial t} = \sum_{i=1}^n \frac{\partial E}{\partial x_i} \dot{x}_i(t).$$

Observăm că

$$\frac{dE[\mathbf{x}(t)]}{dt} = [\nabla_x E(\mathbf{x})]^t \dot{\mathbf{x}}(t).$$

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

$$\frac{dE[\mathbf{x}(t)]}{dt} = \begin{bmatrix} 2e^{2t} + 4t & 4e^{2t} + 6t \end{bmatrix} \begin{bmatrix} 2e^t \\ 1 \end{bmatrix} = 2(2e^{4t} + (4t + 2)e^{2t} + 3t).$$

Fie m funcții:

$$\begin{aligned} E_1(\mathbf{x}) &= E_1(x_1, x_2, \dots, x_n) \\ &\vdots \\ E_m(\mathbf{x}) &= E_m(x_1, x_2, \dots, x_n) \end{aligned}$$

Matricea iacobiană se definește în felul următor:

$$\mathbf{I}(\mathbf{x}) = \frac{d\mathbf{E}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial E_1}{\partial x_1} & \dots & \frac{\partial E_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial E_m}{\partial x_1} & \dots & \frac{\partial E_m}{\partial x_n} \end{bmatrix}$$

Exemplu: Pentru:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} 2 & 1 \end{bmatrix}^t \\ E_1(\mathbf{x}) &= x_1^2 - x_1 x_2 \\ E_2(\mathbf{x}) &= x_1 x_2 + x_2^2 \end{aligned}$$

avem:

$$\mathbf{I}(\mathbf{x}) = \begin{bmatrix} 2x_1 - x_2 & -x_1 \\ x_2 & x_1 + 2x_2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix}.$$

Matricea hessiană a funcției $E(\mathbf{x}) = E(x_1, \dots, x_n)$ se definește astfel:

$$\mathbf{H}(\mathbf{x}) = \nabla_{\mathbf{x}}^2 E(\mathbf{x}) = \nabla_{\mathbf{x}}[\nabla_{\mathbf{x}} E(\mathbf{x})] = \begin{bmatrix} \frac{\partial^2 E}{\partial x_1^2} & \frac{\partial^2 E}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 E}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial x_1 \partial x_n} & \frac{\partial^2 E}{\partial x_2 \partial x_n} & \dots & \frac{\partial^2 E}{\partial x_n^2} \end{bmatrix}.$$

Matricea hessiană este simetrică.

Exemplu: Pentru:

$$E(\mathbf{x}) = (x_2 - x_1)^2 + (1 - x_1)^2$$

avem:

$$\nabla_{\mathbf{x}} E(\mathbf{x}) = 2 \begin{bmatrix} 2x_1 - x_2 - 1 \\ x_2 - x_1 \end{bmatrix} \quad \nabla_{\mathbf{x}}^2 E(\mathbf{x}) = 2 \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}.$$

Deoarece determinanții principali sunt 4 și 2, matricea hessiană este pozitiv definită.

A.6 Probleme de optimizare

Presupunem că avem o funcție obiectiv $E(x)$, unde x este o variabilă scalară. Această funcție își atinge minimumul în punctul $x = x^*$, pentru care se îndeplinesc următoarele condiții:

$$\begin{cases} \frac{dE(x^*)}{dx^*} = 0 \\ \frac{d^2E(x^*)}{d^2x^*} > 0 \end{cases} .$$

Dacă \mathbf{x} este un vector, generalizarea se face astfel:

$$\begin{cases} \nabla_{\mathbf{x}}E(\mathbf{x}^*) = 0 \\ \nabla_{\mathbf{x}}^2E(\mathbf{x}^*) \text{ pozitiv definită} \end{cases} .$$

Pentru a înțelege aceste condiții, să considerăm, de exemplu:

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

Dezvoltăm în serie Taylor în punctul $\mathbf{x} = \mathbf{x}^*$ și reținem 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 $\Delta x_1 = x_1 - x_1^*$, $\Delta x_2 = x_2 - x_2^*$, obținem:

$$\begin{aligned} E(\mathbf{x}) &\cong E(\mathbf{x}^*) + \begin{bmatrix} \frac{\partial E(\mathbf{x}^*)}{\partial x_1} & \frac{\partial E(\mathbf{x}^*)}{\partial x_2} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} \\ &+ \frac{1}{2} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}^t \begin{bmatrix} \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 E(\mathbf{x}^*)}{\partial x_2^2} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} . \end{aligned}$$

Luând

$$\Delta \mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} ,$$

obținem:

$$E(\mathbf{x}) \cong E(\mathbf{x}^*) + [\nabla_{\mathbf{x}}E(\mathbf{x}^*)]^t \Delta \mathbf{x} + \frac{1}{2} \Delta \mathbf{x}^t [\nabla_{\mathbf{x}}^2 E(\mathbf{x}^*)] \Delta \mathbf{x} . \quad (\text{A.1})$$

Dacă $E(\mathbf{x})$ are un minim în \mathbf{x}^* , orice schimbare infinezimală $\Delta \mathbf{x}$ 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 și 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 $\Delta \mathbf{x}$.

Soluția analitică a unei probleme de optimizare speciale

Fie funcția:

$$E(\mathbf{x}) = \frac{1}{2} \mathbf{x}^t \mathbf{A} \mathbf{x} + \mathbf{b}^t \mathbf{x},$$

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 puținele funcții pentru care există o soluție analitică pentru minimul fără restricții.

Calculăm:

$$\begin{aligned} \nabla_{\mathbf{x}} E(\mathbf{x}) &= \mathbf{A} \mathbf{x} + \mathbf{b} \\ \nabla_{\mathbf{x}}^2 E(\mathbf{x}) &= \mathbf{A}. \end{aligned}$$

Prin ipoteză, \mathbf{A} este pozitiv definită. Impunem condiția $\mathbf{A} \mathbf{x} + \mathbf{b} = 0$. Soluția este $\mathbf{x}^* = -\mathbf{A}^{-1} \mathbf{b}$. Rezultă că \mathbf{x}^* este un minim.

A.7 Metoda lui Euler (metoda tangentei)

Fie ecuația diferențială de ordinul întâi:

$$y' = f(x, y), \quad \text{unde } y = y(x)$$

cu condiția inițială

$$y_0 = y(x_0).$$

Dacă f este complicată, nu putem integra direct.

Dorim o aproximare a lui $y^* = \Phi(x)$, care se demonstrează că este soluția 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,

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + hy_n', \quad \text{dacă } x_{n+1} = x_n + h, \quad n = 0, 1, 2, \dots$$

Această relație este *formula lui Euler*.

Exemplu:

$$\begin{cases} y' = 1 - x + 4y \\ y(0) = 1 \end{cases}.$$

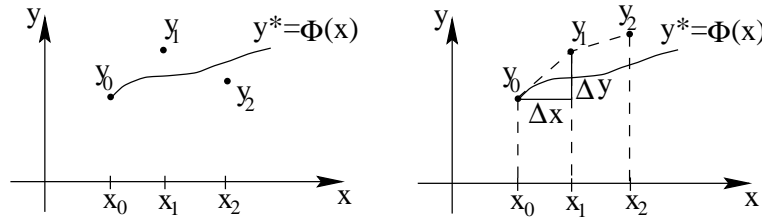


Figura A.1: Ne deplasăm pe tangenta la y_0 , până la x_1 .

Se știe că soluția exactă a acestui sistem are următoarea expresie analitică:

$$y^* = \Phi(x) = \frac{1}{4}x - \frac{3}{16} + \frac{19}{16}e^{4x}.$$

Aproximați pe $\Phi(0, 2)$, presupunând că nu aveți 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 , obținem o eroare mai mică.

Fie acum un sistem de două ecuații diferențiale de ordinul I:

$$\begin{cases} x' = f(t, x, y) \\ y' = g(t, x, y) \\ x(t_0) = x_0, y(t_0) = y_0 \end{cases}.$$

Căutăm valorile aproximative $x_1, x_2, \dots, x_n, \dots, y_1, y_2, \dots, y_n, \dots$ ale soluției 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: Determinați valorile aproximative ale lui $x^* = \Phi(t)$, $y^* = \Psi(t)$ în punctele $t = 0,1$ și $t = 0,2$ 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:

$$\begin{cases} x_1' = F_1(t, x_1, \dots, x_n) \\ x_2' = F_2(t, x_1, \dots, x_n) \\ \vdots \\ x_n' = F_n(t, x_1, \dots, x_n) \end{cases}$$

cu condițiile inițiale:

$$\begin{array}{l} x_1(t_0) = x_1^0 \\ \vdots \\ x_n(t_0) = x_n^0 . \end{array}$$

Acest sistem are o soluție în intervalul $\alpha < t < \beta$ dacă există funcțiile $x_1^* = \Phi_1(t)$, ..., $x_n^* = \Phi_n(t)$, diferențiabile în $\alpha < t < \beta$ și care satisfac condițiile inițiale.

Teoremă de existență și unicitate: Dacă F_1, \dots, F_n și $\frac{\partial F_1}{\partial x_1}, \dots, \frac{\partial F_1}{\partial x_n}, \dots, \frac{\partial F_n}{\partial x_1}, \dots, \frac{\partial F_n}{\partial x_n}$ sunt continue în regiunea R care conține punctul $(t_0, x_1^0, \dots, x_n^0)$, atunci există un interval $|t - t_0| < R$ în care există o soluție unică $x_1^* = \Phi_1(t)$, ..., $x_n^* = \Phi_n(t)$ a sistemului, care satisface condițiile inițiale.

Observăm că nu se spune nimic despre derivatele $\frac{\partial F_i}{\partial t}$.

A.8 Stabilitatea sistemelor dinamice neliniare

La sfârșitul secolului XIX, Liapunov a dezvoltat o metodă pentru analiza stabilității sistemelor dinamice. Această metodă se bazează pe conceptul de energie generalizată și presupune evaluarea funcției lui Liapunov.

Fie un sistem autonom descris de sistemul de ecuații diferențiale liniare sau neliniare:

$$\begin{array}{l} \dot{x}_1 = f_1(\mathbf{x}) \\ \dot{x}_2 = f_2(\mathbf{x}) \end{array}$$

$$\begin{aligned} & \vdots \\ \dot{x}_n &= f_n(\mathbf{x}) \end{aligned}$$

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 condiție 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 funcție pozitiv definită pentru care:

1. E este continuă față de toate componentele $x_i, i = 1, 2, \dots, n$
2. $\frac{dE[\mathbf{x}(t)]}{dt} < 0$, adică funcția energiei descrește în timp,

atunci originea este asimptotic stabilă.

Funcția E care satisface aceste condiții se numește *funcția lui Liapunov*. În general, ea nu este unică.

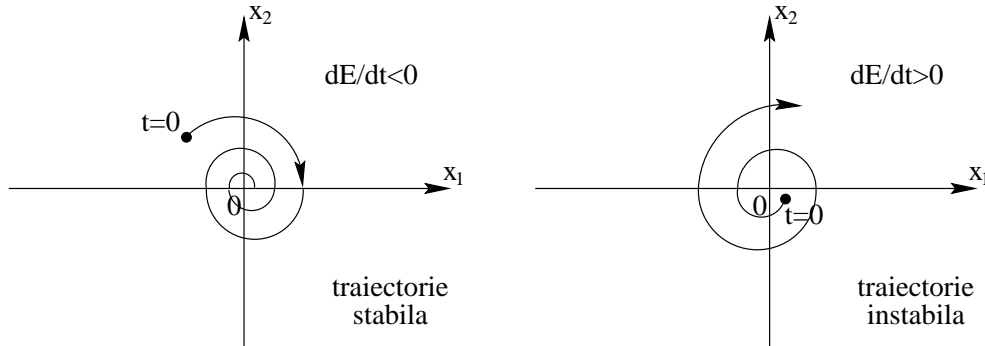


Figura A.2: Funcția lui Liapunov

Dacă cel puțin o astfel de funcție satisface condițiile teoremei, atunci sistemul este asimptotic stabil. Uneori nu este foarte clar despre ce *energie* este vorba. Considerațiile sunt mai curând matematice decât fizice. În cazul rețelelor neurale vorbim de *funcția energiei computaționale*, ea neavând legătură cu energia din fizică.

A.9 Variabile aleatoare

Fie o variabilă aleatoare discretă

$$\xi = \begin{pmatrix} a_1 & \dots & a_n \\ p_1 & \dots & p_n \end{pmatrix}, \sum p_i = 1.$$

Definim:

Valoarea medie: $m = \sum a_i p_i = E(\xi)$

Varianța (dispersia): $\sigma^2 = E(\xi - E(\xi))^2 = E(\xi^2) - (E(\xi))^2 = \sum a_i^2 p_i^2 - (\sum a_i p_i)^2$

Deviația standard: σ .

Funcția de repartiție a unei variabile aleatoare normale ξ cu media m și dispersia σ^2 este:

$$\begin{aligned} F(x) &= P(\xi < x) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{(u-m)^2}{2\sigma^2}} du, \quad \text{pentru } -\infty < x < \infty \end{aligned}$$

și este notată cu $N(m, \sigma^2)$. Distribuția $N(0, 1)$ este *standard*. Funcția 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, funcția discriminant liniară.
2. Regula hebbiană.
3. Regula Windrow-Hoff.
4. Deduceți regula delta pentru un neuron din condiția de minimizare a erorii.
5. Demonstrați că funcția energiei computaționale:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \sum_{i=1}^n G_i \int_0^{v_i} f_i^{-1}(z)dz$$

într-o rețea Hopfield cu timp continuu este necrescătoare.

6. Principiul de funcționare a unui convertor A/D pe 2 biți implementat printr-o rețea de tip gradient.
7. Teorema perceptronului.
8. Algoritmul de stocare a pattern-urilor prototip într-o memorie autoasociativă recurentă unipolară.
9. Aproximarea unei funcții continue prin rețele neurale feedforward.
10. Alegerea arhitecturii unei rețele neurale feedforward cu un strat ascuns.
11. Teorema lui Liapunov.
12. Problema zgomotului într-un asociator liniar.
13. Reprezentarea unei funcții booleene printr-o rețea neurală feedforward.
14. Adăugarea / eliminarea unui pattern la o / dintr-o memorie autoasociativă recurentă.
15. Influența factorilor χ și η în algoritmul backpropagation.

16. Demonstrați că funcția energiei computaționale:

$$E = -\frac{1}{2}\mathbf{v}^t\mathbf{W}\mathbf{v} - \mathbf{i}^t\mathbf{v} + \mathbf{T}^t\mathbf{v}$$

într-o rețea 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 ascunși?
21. Într-o rețea Hopfield cu timp continuu, energia scade cât mai rapid?
22. De ce algoritmul de actualizare la rețelele Hopfield trebuie să fie stochastic asincron?
23. Cum putem găsi punctele de stabilitate ale unei rețele Hopfield de tip gradient?
24. Ce este un asociator liniar?
25. Teorema fundamentală a geneticii.
26. Paralelismul implicit al algoritmilor genetici.
27. Mulțime fuzzy.
28. Logica fuzzy.
29. Rolul intrării fixe în algoritmul de instruire a unei rețele feedforward de perceptroni discreți.
30. Teza Church-Turing.
31. Mașina Turing.
32. Adăugați relațiile de incluziune corespunzătoare între următoarele mulțimi: mașini Turing, automate celulare, rețele neurale, rețele de automate.
33. Teorema lui Judd referitoare la complexitatea instruirii rețelelor neurale feedforward multistrat.
34. Capacitatea memoriei autoasociative recurente.
35. De ce în cazul memoriei asociative bidirecționale se preferă procesarea sincronă?

36. Principiul de funcționare al rețelei Hamming.
37. Principiul de funcționare al rețelei MAXNET.
38. Care este scopul calibrării supervizate a unui clasificator neural?
39. Principiul de funcționare a rețelei Kohonen.
40. Ce este o hartă Kohonen?
41. Principiul de funcționare a rețelei neurale fuzzy.
42. Principiul de funcționare a rețelei neurale cu funcție 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 "Rețele neuronale; teorie și aplicații". Teora, București, 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] Negoită, C.V., D.A. Ralescu "Mulțimi vagi și aplicații ale lor", Ed. Tehnică, București, 1974.
- [14] Ștefan, Gh. "Funcție și structură în sistemele digitale", Ed. Academiei, București, 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.