

Sisteme computaționale inteligente

Versiunea 2017.886

Ph.D. Lucian Sasu

27 mai 2017

Cuprins

1	Introducere	5
1.1	Rețele neurale	7
1.1.1	Bazele biologice	7
1.1.2	Diferențe între RN artificiale și naturale	9
1.1.3	Aplicabilitate	9
1.2	Calcul evoluționist	10
1.2.1	Bazele biologice	10
1.2.2	Cromozomi	11
1.2.3	Diferențe între cromozomii biologici și cei artificiali	11
1.2.4	Aplicabilitate	11
1.3	Sistemele fuzzy	11
1.4	Tipuri de învățare în inteligența computațională	12
1.4.1	Învățarea supervizată	12
1.4.2	Învățarea prin întărire	13
1.4.3	Învățarea nesupervizată	13
1.5	Auto-organizarea	15
2	Regresia liniară	16
2.1	Exemplu, notații	16
2.2	Funcția de cost	19
2.3	Metoda de căutare după direcția gradientului	20
2.4	Metoda algebrică	23
2.5	Overfitting, underfitting, regularizare	26
2.5.1	Overfitting, underfitting	26
2.5.2	Regularizare	28
3	Regresia logistică	31
3.1	Încadrare, motivație	31
3.2	Regresia logistică	32
3.2.1	Setul de instruire	32
3.2.2	Reprezentarea ipotezei	32
3.2.3	Suprafața de decizie a regresiei logistice	33
3.2.4	Funcția de cost	34

3.2.5	Algorimul de instruire	36
3.2.6	Regularizare	37
3.3	Regresia logistică multinomială	38
3.3.1	Reprezentarea ipotezei	38
3.3.2	Funcția de cost	39
3.3.3	Algoritmul de instruire	39
3.3.4	Regularizare	40
4	Rețele neurale artificiale - fundamente	41
4.1	Încadrarea domeniului	41
4.2	Neuronul biologic	43
4.3	Modele de neuroni artificiali	44
4.3.1	Modelul McCulloch–Pitts	44
4.3.2	Modelarea neuronului pentru sisteme neurale artificiale	44
4.4	Modele de rețea neurală artificială	47
4.4.1	Rețea cu propagare înainte	47
4.4.2	Rețele cu conexiune inversă	48
4.5	Învățarea ca problemă de aproximare	50
4.6	Reguli de învățare	50
4.6.1	Regula de învățare Hebbiană	51
4.6.2	Regula de învățare a perceptronului	52
4.6.3	Regula de învățare delta	53
4.6.4	Regula de învățare Widrow-Hoff	53
4.6.5	Regula de învățare prin corelație	54
4.6.6	Regula “câștigătorul ia tot”	54
5	Perceptroni monostrat	55
6	Perceptroni multistrat	56
6.1	Motivație pentru rețele neurale multistrat	56
6.2	Sumar de notații	57
6.3	Setul de instruire	57
6.4	Rețeaua neurală multistrat	59
6.4.1	Arhitectură	59
6.4.2	Funcții de activare	61
6.5	Pasul de propagare înainte	63
6.6	Pasul de propagare înapoi a erorii	64
6.6.1	Funcții de cost	65
6.6.2	Algoritmul backpropagation	66
6.6.3	Justificarea matematică a algoritmului	69
6.7	Utilizarea rețelei	69
6.8	Discuții	69

7	Memorii asociative bidirecționale	70
7.1	Distanța Hamming	70
7.2	Asociatori	71
7.3	Memoria asociativă bidirecțională	72
7.4	Funcția de energie a MAB	75
7.5	Capacitatea de memorie	76
8	Rețele neurale cu funcții de activare radială	77
8.1	Teorema lui Cover	77
8.2	Funcții cu activare radială	79
8.3	Rețele cu funcții cu activare radială	81
8.4	Clustering folosind algoritmul K-means	82
8.5	Determinarea ponderilor pentru RBF	85
8.6	Algoritmul de instruire a rețelei RBF	86
9	Fuzzy ARTMAP	87
9.1	Învățarea incrementală	87
9.2	Proprietăți dezirabile ale sistemelor instruibile	87
9.3	Dilema stabilitate-plasticitate	88
9.4	Fuzzy ARTMAP	89
9.4.1	Arhitectura rețelei FAM	90
9.4.2	Algoritmul de învățare pentru FAM	92
10	Hărți cu auto-organizare	98
10.1	Încadrare	98
10.2	Intrări, arhitectură și algoritm de instruire	99
10.3	Lema Johnson–Lindenstrauss	101
11	Calcul evoluționist	103
11.1	Taxonomie	103
11.2	Algoritmi genetici	104
11.3	Fundamente teoretice	107
11.4	Problema reprezentării datelor în algoritmii genetici	110
11.4.1	Varianta cu penalizare	112
11.4.2	Varianta cu reparare	113
11.4.3	Codificarea adecvată a indivizilor	114
11.5	Exemplu: problema orarului	115
12	Mulțimi și logică fuzzy	117
12.1	Prezentare generală	117
12.2	Teoria mulțimilor fuzzy	118
12.3	Operații cu mulțimi fuzzy	120
12.3.1	Egalitatea mulțimilor fuzzy	121
12.3.2	Incluziunea mulțimilor fuzzy	121

12.3.3	Complementara unei mulțimi fuzzy	121
12.3.4	Intersecția a două mulțimi fuzzy	122
12.3.5	Reuniunea a două mulțimi fuzzy	122
12.3.6	Operatori de compensare	123
12.4	Reguli fuzzy	123
12.5	Măsuri ale gradului de nuanțare	126

Capitolul 1

Introducere

Inteligența computațională (IC) este un domeniu care combină elemente de învățare automată, adaptare, evoluție și logică fuzzy pentru a rezolva probleme care, abordate tradițional, sunt dificil sau imposibil de abordat. Este o ramură a inteligenței artificiale. Subdomeniile majore ale inteligenței computaționale sunt:

- modele de învățare parametrică
- modele de învățare neparametrice, precum rețele neurale¹ artificiale, Support vector machines, modele probabiliste;
- mulțimi și logică fuzzy;
- calcul evoluționist;
- sisteme de imunitate artificiale;
- inteligența mușuroiului.

Fiecare din aceste subdomenii a evoluat rapid și s-au impus ca potențiale metode de rezolvare efectivă a unor probleme complexe și presante, pentru care abordările uzuale sunt nefructuase. De regulă, prototipizarea unui sistem inspirat din inteligența computațională este rapidă, iar pentru o problemă se pot folosi mai multe abordări: de exemplu, optimizarea se poate face prin algoritmi genetici sau prin anumite familii de rețele neurale.

Metodele din inteligența computațională (IC) sunt frecvent inspirate din biologie: rețelele neurale au pornit de la modelul imaginat pentru neuronul biologic, calculul evoluționist este bazat pe teoria evoluției enunțată de Charles Darwin. Sistemele fuzzy sunt introduse pentru a permite manipularea incertitudinii, altfel decât prin teoria probabilităților.

Este o mare diferență între abordarea clasică, algoritmică a unei probleme și cea dată de IC. În primul caz este pusă la bătaie toată abilitatea

¹Sau “neuronale”

celui care imaginează algoritmul pentru a rezolva problema; este un demers anevoios, depinzând esențial de imaginația, puterea de abstractizare și experiența persoanei în cauză; evident, este un proces creativ, la ora actuală efectuat exclusiv de oameni. Totodată, de cele mai multe ori rezultatele sunt exacte; de asemenea, se are în vedere permanent mișorarea complexității de calcul a problemei respective, dar în destule situații o soluție exactă presupune un efort de calcul sau resurse de memorie prohibitive.

Abordarea IC este total diferită: pentru rețele neurale sau algoritmi genetici, definitorie este capacitatea de *adaptare* automată sau *auto-organizare* la condițiile problemei. Este modelul inspirat din natură, unde un sistem biologic preia semnale din mediu și printr-un proces de învățare se adaptează, astfel încât să își îndeplinească scopul, sau pentru a obține o mai bună integrare. Soluția la care se ajunge nu este neapărat optimă, dar este un răspuns suficient de bun pentru problema propusă. În implementarea unui sistem din cadrul IC accentul cade mai mult pe abilitatea sistemului rezultat de a se adapta, de a învăța, decât pe imaginația celui care îl concepe. Abilitățile de programare pentru implementarea sau personalizarea sistemului sunt însă esențiale.

Sistemele propuse în cadrul IC sunt cu un mare grad de aplicabilitate. De exemplu, algoritmi genetici pot fi folosiți pentru o clasă largă de funcții, nedepinzând atât de mult precum cercetările operaționale de ipoteze care în practică pot fi prea restrictive.

O definiție a “intelenței” potrivită pentru contextul de IC este:

Definiția 1. *Intelența este abilitatea unui sistem de a-și adapta comportamentul pentru a-și îndeplini scopurile în mediul său. Este o proprietate a tuturor entităților ce trebuie să ia decizii și al căror comportament este condus de scop.*

Definiția de mai sus a fost dată în 1995 de către David Fogel, scoțând în evidență elementul esențial al comportamentului inteligent și în particular al intelenței computaționale: adaptarea.

Rețelele neurale artificiale reprezintă grupuri interconectate de neuroni artificiali care au abilitatea de a învăța din și a se adapta la mediul lor, construind un model al lumii. Ele au apărut ca răspuns la modelarea activității creierului biologic, precum și ca modalitate propusă pentru a obține sisteme artificiale capabile să recunoască șabloane. Exemple de rețele neurale și algoritmi de instruire se găsesc în [5], [6], [16].

Sistemele fuzzy sunt introduse pentru a putea gestiona imprecizia, noțiunile vagi (“înalt”, “acum”) și aproximarea. Sunt elemente des întâlnite în modelarea de limbaj sau în situații de cunoaștere incompletă. Teoria mulțimilor fuzzy permite ca un element să aibă un anumit grad de apartenență (număr între 0 și 1) la o mulțime, spre deosebire de teoria clasică a mulțimilor. Logica fuzzy permite considerarea mai multor valori de adevăr decât

cele din logica clasică, sau altfel zis, a unor grade de adevăr diferite. Este variantă de realizare a raționamentului aproximativ.

Calculul evoluționist se ocupă în special cu optimizarea și de probleme de căutare, bazate pe mecanismele preluate din genetică și evoluționism. Se pleacă de la ideea evoluției unei populații de indivizi, fiecare din ei fiind o soluție potențială a problemei ce se vrea rezolvată. Domeniul include algoritmi genetici, programarea evoluționistă, programarea genetică și strategii de evoluție.

Sistemele rezultate prin inteligență computațională pot reprezenta hibridizări ale celor de mai sus; de exemplu, există sisteme neuro-fuzzy, iar ajustarea parametrilor pentru un sistem adaptiv se poate face prin algoritmi genetici. Alegerea uneia potrivite pentru problema în cauză este o problemă deloc simplă, deoarece de regulă se pot folosi mai multe abordări.

1.1 Rețele neurale

1.1.1 Bazele biologice

Rețeaua neurală biologică a evoluat de-a lungul câtorva milenii, ajungând la performanțe care astăzi nu sunt accesibile calculatoarelor electronice: de exemplu, recunoașterea de imagini, specifică animalelor; sau interpretarea ecoului reflectat de către obstacole sau insecte, în cazul liliecilor - chiar dacă au creierul foarte mic, procesarea în cazul lor se face mai rapid decât pentru cele mai performante sisteme electronice actuale.

Studiile efectuate în ultimul secol au permis enunțarea unor principii asupra modului de funcționare a sistemelor neurale biologice; suntem însă departe de a cunoaște toate detaliile funcționale și structurale. Chiar și așa, prin implementarea modelelor obținute, rezultatele sunt mai mult decât notabile.

Figura 1.1 ([4]) reprezintă cel mai comun tip de neuron natural. În scoarța neurală există circa 86 de miliarde de neuroni interconectați, fiecare putând avea până la 10^4 conexiuni cu alți neuroni; modul de grupare a acestora și interdependențele nu sunt pe deplin cunoscute.

Un neuron artificial are o structură asemănătoare, fiind un element de procesare conectat cu alte elemente ce preia intrare de la niște neuroni și produce o ieșire ce devine intrare pentru alți neuroni; legăturile neurale sunt niște coeficienți numerici, iar prin algoritmi de învățare se obține adaptarea convenabilă a rețelei neurale. Adaptarea (sau învățarea) este aspectul esențial al rețelelor neurale: plecând de la seturi de date, se detectează automat șabloanele existente și se construiesc niste modele care pot fi folosite mai departe.



Figura 1.1: Neuron natural [4]



Figura 1.2: Neuron de tip Purkinje din cortexul cerebelar; sursa <http://en.wikipedia.org/wiki/Neuron>.

1.1.2 Diferențe între RN artificiale și naturale

În mod cert însă, există diferențe: nu sunt modelate toate tipurile cunoscute de neuroni; apoi, o lege biologică spune că un neuron poate să excite sau să inhibe un neuron cu care este conectat; în modelarea de RN artificiale, o pondere de legătură este fie excitatoare, fie inhibitoare, dar forma ei este fixată după ce s-a făcut învățarea.

O altă diferență (și punct de critică pentru rețelele neurale artificiale) este faptul că modelarea semnalului făcută sub formă de valori continue este de negăsit în rețelele biologice; în RN biologice se folosesc de fapt trenuri de impulsuri care sunt transmise către neuroni, apărând variație în frecvența semnalului. Acest aspect a fost abordat relativ târziu, în cadrul rețelelor neurale cu pulsuri.

Viteza rețelelor neurale este iarăși un loc în care apar diferențe. Se estimează că neuronii naturali au cicli de timp între 10 și 100 milisecunde; implementările de RN artificiale funcționează pe procesoare de câțiva gigahertzi, deci cu un ciclu de mai puțin de o nanosecundă. Chiar și așa, rețelele neurale biologice sunt cu mult mai performante decât cele artificiale.

Altă diferență este că neuronii naturali sunt grupați în cantități mari, uneori de sute de milioane de unități. Se ajunge astfel la un grad de paralelism masiv, ce nu a fost încă atins în simulările artificiale.

1.1.3 Aplicabilitate

- *Clasificarea* - pe baza unui set de date de forma (intrare - ieșire asociată) se construiește un sistem care detectează asocierile dintre datele de intrare și etichetele ce le sunt asociate; etichetele - sau clasele - sunt dintr-o mulțime discretă, finită. Clasificarea se folosește pentru recunoașterea automată a formelor, recunoașterea vorbirii, diagnoză medicală și altele.
- *Estimarea de probabilitate condiționată* - similar cu clasificarea, dar se produce un sistem care estimează probabilitatea ca un obiect să aparțină unei clase, date fiind trăsăturile de intrare; de exemplu, dat fiind conținutul unui mesaj de email care este probabilitatea ca să fie mail legitim sau spam;
- *Regresie* - asemănător cu clasificarea, dar ieșirile nu sunt dintr-o mulțime discretă și finită, ci valori numerice continue;
- *Memorie asociativă*, sau memorie adresabilă prin conținut - se poate regăsi o dată pe baza unei părți a ei. Este un mecanism diferit de modul în care calculatoarele regăsesc informația - pe baza adreselor sau a unei căutări - dar apropiată de modul în care se face regăsirea elementelor reținute de către o persoană.

- *Grupare* - pe baza similarităților existente într-un set de date, se detectează grupările de date; elementele dintr-un grup sunt mai apropiate între ele decât de altele din alt grup;
- *Detectarea automată de trăsături* – a acelor elemente care fac ca procesul de recunoaștere a unui obiect să fie mai bun decât dacă se folosesc cunoștințe specifice domeniului;
- *Controlul sistemelor* - folosite pentru cazul în care un proces trebuie să fie ghidat pentru a se încadra în parametri; utilitatea rețelelor neurale provine din faptul că nu se presupune că există dependențe liniare între acțiuni și reacțiuni.

1.2 Calcul evoluționist

Principalele paradigme² ale calculului evoluționist sunt:

- algoritmi genetici - evoluția unei populații de indivizi (cromozomi), folosind selecția, încrucișarea și mutația;
- programarea evoluționistă - similar cu precedenta, dar fără a folosi încrucișarea; este văzută ca evoluția de specii diferite, între care nu există hibridizări;
- strategiile de evoluție - similare cu algoritmi genetici, dar se folosește recombinarea în loc de încrucișare și deseori alte metode de mutație
- programarea genetică - metode evolutive aplicate programelor de calculator.

1.2.1 Bazele biologice

Domeniile de inspirație sunt genetica și teoria evoluționistă. Genetica explică ereditatea, adică transmiterea caracterelor de la părinți la urmași. Astfel, adaptarea obținută în generațiile anterioare este preluată de către urmași și continuată. Codificarea caracteristicilor este dată de cromozomi. Noțiunile și mecanismele sunt preluate din teoria eredității întemeiată de Gregor Mendel și teoria evoluționistă a lui Charles Darwin.

²“Paradigma este o construcție mentală larg acceptată, care oferă unei comunități sau unei societăți pe perioada îndelungată o bază pentru crearea unei identități de sine (a activității de cercetare de exemplu) și astfel pentru rezolvarea unor probleme sau sarcini.”, conform [Wikipedia](#).

1.2.2 Cromozomi

Cromozomii sunt structuri din interiorul celulelor care mențin informația genetică. În cazul oamenilor, sunt 46 de cromozomi, jumătate moșteniți de la tată și jumătate de la mamă. Cromozomii sunt alcătuiți din gene, fiecare fiind identificată prin locația pe care o ocupă și prin funcția asociată.

1.2.3 Diferențe între cromozomii biologici și cei artificiali

Cromozomii artificiali sunt reprezentări simplificate a celor biologici. În timp ce neuronii biologici sunt secvențe de acizi nucleici, cromozomii artificiali sunt șiruri de cifre binare.

Cromozomii biologici care definesc organisme vii variază în lungime, chiar dacă de la un organism la altul din aceeași specie pentru un cromozom specific lungimea este constantă. În algoritmi genetici, lungimea este fixă.

La reproducerea indivizilor dintr-o populație naturală, jumătate din informația genetică este preluată de la tată și jumătate de la mamă. În algoritmi genetici, procentul de combinație poate să difere.

1.2.4 Aplicabilitate

Principală arie de aplicare este optimizarea, pentru situațiile în care căutarea soluției cere un timp îndelungat. Algoritmi genetici sunt folosiți ca o metodă euristică; problemele abordate sunt din cele mai diverse — optimizarea unui plan de lucru sau circuit, balansarea încărcării, optimizarea ingredientelor, design automat, încărcarea containerelor, optimizarea structurilor moleculare, testarea mutațiilor, optimizarea sistemelor de compresie, selectarea modelelor optime, găsirea defectelor hardware *etc.*

1.3 Sistemele fuzzy

Sistemele fuzzy și logica fuzzy nu sunt de inspirație biologică, ci preluate din partea comportamentală umană. Este o modalitate de manipulare a incertitudinii, modelând imprecizia, caracterul vag, ambiguitatea. Este vorba de un alt tip de incertitudine decât cel modelat prin intermediul variabilelor aleatoare din cadrul teoriei probabilităților. Se folosește pentru modelarea impreciziei lingvistice (“Maria e înaltă”, “Livrarea se face în aproximativ 3 ore”). Teoria mulțimilor fuzzy a fost dezvoltată de către Lotfi Zadeh începând cu anul 1965.

Un exemplu de raționament fuzzy este:

```
IF temperature IS very cold THEN stop fan
IF temperature IS cold THEN turn down fan
IF temperature IS normal THEN maintain level
IF temperature IS hot THEN speed up fan
```

Toate variantele sunt evaluate și în funcție de rezultat se ajustează viteza ventilatorului. Modelarea conceptelor de “very cold”, “normal” etc. se face prin mulțimi vagi.

Pornind de la acest curent, s-au dezvoltat următoarele: fuzzificare/de-fuzzificarea, sisteme de control fuzzy, jocuri fuzzy, matematică fuzzy, teoria măsurii fuzzy, căutare fuzzy.

Teoria fuzzy este folosită intens în sisteme ce presupun control: camere video, sisteme de frânare sau accelerare, sisteme de control al debitului și presiunii etc. De asemenea, sistemele expert din domeniu medical, financiar, navigațional, diagnoza mecanică etc. se folosesc masiv de suportul pentru imprecizie și ambiguitate.

1.4 Tipuri de învățare în inteligența computațională

Învățarea permite unui sistem să se adapteze la mediul în care operează; pe baza semnalelor provenite din exterior, sistemul inteligent își modifică parametrii pentru o îndeplinire cât mai bună a sarcinii propuse. Trebuie făcută distincția între “învățare” și “memorare cu regăsire exactă” – această din urmă problemă este rezolvată de structuri și baze de date.

Există trei tipuri principale de învățare:

1. supervizată
2. nesupervizată
3. prin întărire

La acestea se adaugă și învățarea semi-supervizată.

1.4.1 Învățarea supervizată

Se presupune că există un “profesor” care poate prezenta un set de date de instruire având forma (intrare — ieșire asociată), relevant, care este preluat de către sistem și învățat. Se folosește o funcție de eroare, care măsoară cât de departe este răspunsul cerut față de cel furnizat de sistem; pe baza erorii se desfășoară un proces de ajustare a valorilor din sistemul computațional inteligent până când eroarea scade sub un anumit prag. Rezultatul final este obținerea unui sistem ce poate să furnizeze o valoare de ieșire adecvată pentru o anumită valoare de intrare ce nu este prezentă în setul de instruire.

Exemple de sisteme ce folosesc instruirea supervizată: perceptronul, perceptronul multistrat, Fuzzy ARTMAP, rețelele cu activare radială.



Figura 1.3: Schema de lucru pentru învățare supervizată

1.4.2 Învățarea prin întărire

Este similară cu învățarea supervizată, numai că în loc de a se furniza ieșirea asociată unei intrări, se pune la dispoziție o indicație care arată cât de bine a acționat sistemul respectiv. Acesta este un sistem bazat pe critică sau aprobare, fiind instruit în raport cu măsura în care ieșirea obținută de un sistem corespunde valorii dorite (dar fără ca această valoare dorită să fie precizată sistemului!). Rolul profesorului este luat de un critic, care precizează în ce măsură ieșirea obținută se apropie de cea dorită. Pe termen lung, sistemul își va modifica propriul comportament astfel încât să se reducă criticile obținute.

Acest tip de învățare este plauzibil din punct de vedere biologic, deoarece un animal va încerca să își minimizeze starea de disconfort prilejuită de comportament neadecvat. Rolul criticului este dat aici de mediul înconjurător. Schema de lucru este dată în figura 1.4.

1.4.3 Învățarea nesupervizată

Spre deosebire de precedentele moduri de învățare, în acest caz nu se primește niciun semnal de tip ieșire sau critică asociată. Sistemului capabil de grupare i se dau doar valori de intrare. El face o grupare automată sau folosește o învățare de tip competitiv. Aplicațiile clasice sunt analiza asocierilor, gruparea pe baza de similaritate și estimarea de densitate de probabilitate.

Schema de lucru este dată în figura 1.5. Acest tip de adaptare este prezent în rețelele de tip *Self organizing feature maps* sau *Vector quantization*.



Figura 1.4: Schema de lucru pentru învățare prin întărire



Figura 1.5: Schema de lucru pentru învățare nesupervizată

1.5 Auto-organizarea

Auto-organizarea, alături de învățare, este un alt atribut important al sistemelor computaționale inteligente. Este prezentă în sistemele naturale, de exemplu în creierul nou născuților, unde auto-organizarea se manifestă în principal prin distrugerea legăturilor nefuncționale. Auto-organizarea este definită astfel:

Definiția 2. *Spunem că un sistem se auto-organizează dacă, după ce se primesc intrarea și ieșirea unui fenomen necunoscut, sistemul se organizează singur astfel încât să simuleze fenomenul necunoscut [3].*

sau:

Definiția 3. *Sistemele cu auto-organizare se auto-organizează pentru a clasifica percepțiile din mediu în percepții ce pot fi recunoscute, sau șabloane [3].*

Capitolul 2

Regresia liniară

2.1 Exemplu, notații

Notă: expunerea din acest curs este făcută după [1].

Regresia liniară este o metodă folosită pentru predicția unei valori numerice dintr-o mulțime infinită de valori. Ca exemplu, să presupunem că vrem să facem predicția costului unei proprietăți imobiliare, dată fiind suprafața sa. Se cunosc date anterioare despre vânzarea unor astfel de proprietăți. Pe baza acestor date vom construi o funcție care să ne permită aproximarea prețului (număr real) pentru alte proprietăți de interes. O exemplificare este dată în figura 2.1.

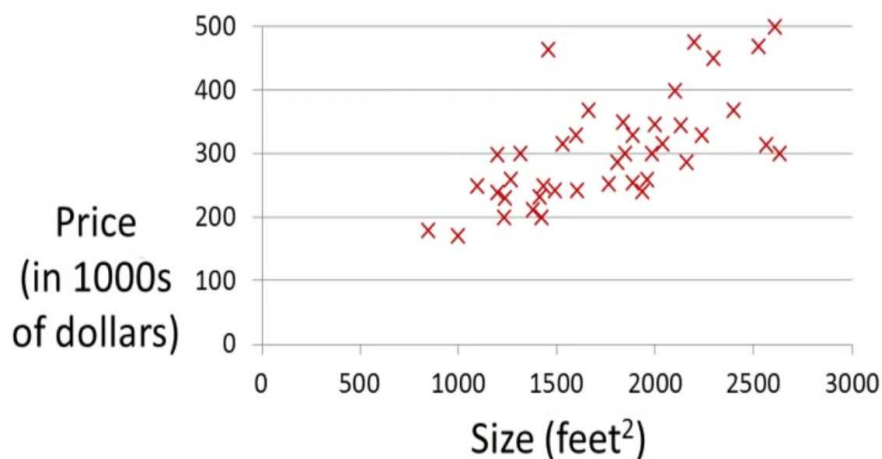


Figura 2.1: Reprezentarea grafică a datelor de vânzare a unor proprietăți imobiliare. Pe abscisă este măsurată suprafața, pe ordonată este prețul [1].

Să presupunem că se dorește estimarea valorii unei proprietăți de suprafață 1300. Se poate proceda în felul următor: se trasează o dreaptă care

să aproximeze “cât mai bine”¹ norul de puncte reprezentat². Se constată apoi care este valoarea de pe dreaptă, corespunzătoare lui 1300 (figura 2.2). Estimarea obținută este de circa 220000.

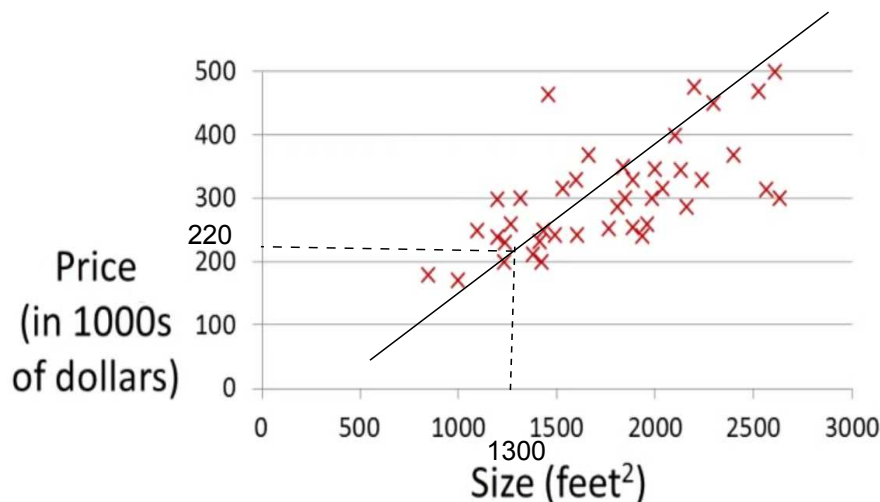


Figura 2.2: Aproximarea prețului pentru o suprafață de 1300 feet².

Modelul de predicție figurat mai sus este un model liniar³:

$$pret = a \cdot suprafața + b$$

unde a și b sunt coeficienți reali ce vor fi determinați. Desigur, se pot folosi forme polinomiale de grad mai mare decât 1, sau modele local liniare, sau rețele neurale etc. Alegerea celui mai bun model pentru un set de date cunoscut este o problemă în sine. Preferarea unui model liniar se motivează prin aceea că în practică se dovedește a fi suficient de bun pentru multe probleme. În plus, un model liniar este ușor de interpretat: creșterea valorii variabilei *suprafața* cu o unitate duce la creșterea prețului total cu a unități monetare.

Avem mai sus un exemplu de instruire supervizată: se pornește de la un set de date cu perechi formate din valoare de intrare (*e.g.* suprafața) și valoare de ieșire asociată (*e.g.* costul suprafeței). Se cere determinarea unui model care să fie folosit pentru prezicerea (aproximarea) unor valori de ieșire, date fiind valori de intrare furnizate; pentru exemplul considerat, vrem să vedem care e costul estimat al unor suprafețe.

Formal, într-o problemă de regresie se dau:

¹În sensul unei funcții de eroare, vezi secțiunea 2.2.

²Pentru cazul cu mai multe date de intrare se obține o varietate liniară, dar modul de determinare a ei este similar cu ceea ce se prezintă pentru o singură variabilă.

³Cu siguranță se pot propune și alte modele pentru predicție.

- m – numărul de perechi de valori (sau cazuri, sau înregistrări) din setul de instruire; pentru desenul din figura 2.1 este numărul de puncte desenate;
- $x^{(j)}$ – variabilele (trăsăturile) de intrare⁴, $1 \leq j \leq m$; variabilelor $x^{(j)}$ li se mai spune și variabile predictive sau independente⁵; în exemplul dat este vorba de suprafață (un singur număr), dar în general pot fi mai multe trăsături, adică un vector de valori pentru fiecare caz: $\mathbf{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)})^t$;
- $y^{(j)}$ – variabila de ieșire (sau de predicție, sau dependentă) aferentă valorii $x^{(j)}$ este în cazul exemplificat un număr real (prețul), dar în general poate fi un vector de valori reale.

O pereche din setul de antrenare se reprezintă prin perechi $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})$, $1 \leq j \leq m$. Întregul set de antrenare este deci

$$\mathcal{S} = \{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}) | 1 \leq j \leq m\} \quad (2.1)$$

Setul de antrenare se specifică frecvent sub formă tabelară, a se vedea tabelul 2.1.

Suprafața (picioare pătrate)	Prețul (mii de dolari)
2100	450
1410	243
800	188
...	...

Tabela 2.1: Set de date de instruire

Fluxul de lucru în învățarea automată⁶ este dat în figura 2.3: se pornește de la un set de instruire, se aplică un algoritm de învățare și se produce un model. Din motive istorice acest model se mai numește și ipoteză și se notează de regulă cu h . Algoritmul de instruire are ca scop determinarea unei forme adecvate a modelului, de exemplu a unor valori potrivite a coeficienților funcției h .

După ce instruirea se termină, modelului rezultat i se furnizează o intrare (în exemplul nostru: suprafața) și modelul va calcula o valoare de ieșire estimată (prețul). În notație formală avem ecuația 2.2.

$$\mathbf{y} = h(\mathbf{x}) \quad (2.2)$$

⁴Engl: input features.

⁵A nu se confunda cu noțiunea de independența liniară din algebră, sau independența evenimentelor și a variabilelor aleatoare din teoria probabilităților.

⁶În limba engleză: învățarea automată = machine learning.



Figura 2.3: Fluxul de lucru într-un proces de instruire automată.

Una din întrebările esențiale este: cum se reprezintă ipoteza h ? După cum am afirmat deja, există mai multe variante ce pot fi utilizate. Pentru exemplul de mai sus am pornit cu presupunerea că prețul crește liniar cu suprafața vândută, deci:

$$h(x) = h_{\theta}(x) = \theta_0 + \theta_1 \cdot x \quad (2.3)$$

unde indicele lui h este vectorul coloană $\theta = (\theta_0, \theta_1)^t$.

Acest model (ipoteză) se numește regresie liniară cu o variabilă, sau regresie liniară univariată. Se poate ca pe lângă suprafață – singura valoare de intrare considerată până acum – să se mai considere și alte variabile de intrare: distanța de la proprietate la utilități, gradul de poluare a zonei etc.; în acest caz, modelul ar fi unul multivariat (mai multe valori de intrare considerate). Coeficienții θ_0 și θ_1 din ecuația (2.3) se mai numesc parametri ai modelului de predicție și se determină prin pasul de învățare.

Modelul (2.3) se mai poate scrie astfel:

$$h_{\theta}(x) = \theta_0 \cdot 1 + \theta_1 \cdot x = \theta_0 \cdot x_0 + \theta_1 \cdot x_1 = \theta^t \cdot \mathbf{x} \quad (2.4)$$

unde: $x_0 = 1$, $x_1 = x$, vectorul θ a fost precizat mai sus, vectorul \mathbf{x} este $(x_0, x_1)^t$.

2.2 Funcția de cost

Există o infinitate de moduri în care se poate trasa dreapta din figura 2.2; altfel zis, există o infinitate de valori pentru coeficienții din modelul dat de ecuația (2.3).

Se pune problema: cum alegem cât mai bine acești coeficienți? O variantă naturală este determinarea acestora de așa manieră încât valorile prezise de model, $h_{\theta}(x^{(j)})$, să fie cât mai apropiate de valorile cunoscute $y^{(j)}$,

pentru tot setul de antrenare \mathcal{S} din (2.1). Pentru toate valorile din setul de instruire, eroarea cumulată se poate măsura cu funcția de cost

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{j=1}^m \left(h_{\theta}(x^{(j)}) - y^{(j)} \right)^2 \quad (2.5)$$

deci trebuie să găsim acele valori ale coeficienților $\theta_0^{(min)}, \theta_1^{(min)}$ pentru care se atinge minimul funcției de eroare:

$$\left(\theta_0^{(min)}, \theta_1^{(min)} \right) = \arg \min_{(\theta_0, \theta_1)} J(\theta_0, \theta_1) = \arg \min_{(\theta_0, \theta_1)} \frac{1}{2m} \sum_{j=1}^m \left(h_{\theta}(x^{(j)}) - y^{(j)} \right)^2 \quad (2.6)$$

Factorul m de la numitor apare pentru a calcula media erorii (altfel, eroarea ar crește de fiecare dată când se adaugă în setul de instruire o pereche $(x^{(j)}, y^{(j)})$ pentru care $h_{\theta}(x^{(j)}) \neq y^{(j)}$, în timp ce media permite compararea erorilor modelului peste seturi de dimensiuni diferite); numitorul 2 se utilizează din motive estetice pentru formulele de mai târziu. Oricum, factorul $1/2m$ nu influențează poziția de optim a lui $\theta_0^{(min)}$ și $\theta_1^{(min)}$.

Funcția de eroare J se mai numește și funcție de cost a modelului⁷. Se pot folosi și alte funcții de cost, de exemplu incluzând constrângeri impuse valorilor parametrilor θ . Funcția de eroare pătratică (norma L_2 la pătrat) este o alegere populară pentru problemele de regresie, dar nu singura posibilă.

Merită să discutăm comportamentul funcției J pentru cazuri particulare. De exemplu, dacă $\theta_0 = 0$, funcția de eroare $J(0, \theta_1)$ este o funcție de gradul 2 depinzând de o singură variabilă (θ_1) și având minimul mai mare sau egal cu zero. Pentru θ_0, θ_1 oarecare forma funcției de eroare este dată în figura 2.4 [1].

O altă variantă de reprezentare grafică a funcției de eroare este pe baza curbilor de contur: reprezentarea este plană, având pe cele două axe respectiv pe θ_0, θ_1 . Pentru o valoare oarecare a funcției de eroare se consideră mulțimea tuturor perechilor de parametri θ_0, θ_1 pentru care se obține aceeași valoare a erorii. Rezultatul este dat de o mulțime de curbe, precum cele reprezentate în figura 2.5 [1]. Se poate arăta că aceste contururi sunt eliptice.

2.3 Metoda de căutare după direcția gradientului

În această secțiune se va prezenta o metodă iterativă pentru minimizarea funcției de eroare J ⁸. Ideea este simplă:

- se pornește cu valori θ_0, θ_1 inițiale, setate aleator sau chiar 0;

⁷În limba engleză: loss function, error function, cost function.

⁸Eng: gradient descent.



Figura 2.4: Funcția de eroare pentru model liniar univariat, cu coeficienți oarecare [1].

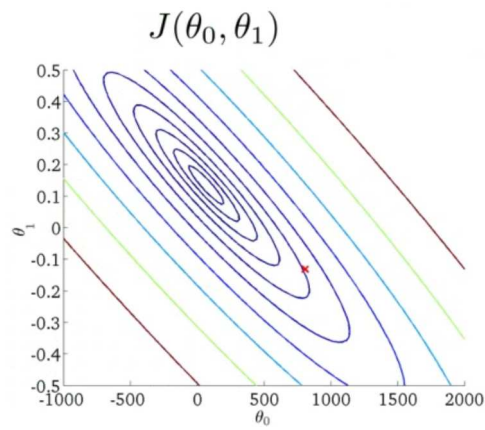


Figura 2.5: Curbe de contur pentru funcția de eroare a unui model liniar univariat [1].

- se modifică în mod repetat valorile curente ale parametrilor θ_0, θ_1 de așa manieră încât J să scadă.

Ultimul punct se concretizează astfel: valorile curente ale parametrilor θ_0, θ_1 se modifică conform

$$\theta_0 = \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) \quad (2.7)$$

$$\theta_1 = \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \quad (2.8)$$

și atribuirile se operează în mod simultan pentru θ_0, θ_1 .

Această simultaneitate e cerută din cauză că la calculele (2.7, 2.8) trebuie să ne asigurăm că aceiași θ_0, θ_1 sunt folosiți pentru ambele derivate parțiale. Simultaneitatea se obține astfel: se calculează expresiile din membrii dreپți ai ecuațiilor (2.7) și (2.8) și se asignează unor variabile temporare $\theta_0^{(nou)}$ și respectiv $\theta_1^{(nou)}$; doar după ce ambele variabile temporare sunt calculate, valorile lor se atribuie corespunzător lui θ_0 și θ_1 . Doar după ce cele două derivate parțiale se calculează folosind aceleași valori θ_0 și θ_1 , noile valori $\theta_0^{(nou)}$ și $\theta_1^{(nou)}$ își intră în drepturi. În limbajele și bibliotecile care permit calcul vectorizat acest lucru se obține foarte simplu, lucrând cu vectori și nu cu variabile considerate pe rând.

Coeeficientul $\alpha > 0$ se numește rată de învățare; poate fi o constantă sau o cantitate care variază de-a lungul iterațiilor. Alegerea lui α este crucială: dacă valoarea lui e prea mică, atunci algoritmul va face foarte multe iterații până se va opri, deci am avea un cost computațional mare. Dacă e prea mare, procesul poate să rateze minimul sau chiar să diveargă (valoarea lui J să crească mereu). Dacă se constată acest al doilea fenomen, valoarea lui α trebuie scăzută. Odată ce o valoare potrivită pentru α este găsită, nu e nevoie ca aceasta să fie modificată. Alte strategii de setare a ratei de învățare sunt: algoritmul gradientului adaptiv (AdaGrad [20]), sau AdaDelta [21], metoda momentum [22] etc⁹.

Metoda se poate folosi pentru reducerea valorilor unei funcții de oricâte variabile. Menționăm că în general se poate ajunge într-un minim local al funcției căreia i se aplică.

Algoritmul de căutare după direcția gradientului are forma (considerăm că valorile $\theta_0, \theta_1, \dots, \theta_n$ sunt inițializate aleator):

repetă{

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1) \quad \text{simultan pentru } i = 0, 1 \quad (2.9)$$

⁹O prezentare a acestora se găsește la <http://sebastianruder.com/optimizing-gradient-descent/>.

} pana la convergenta

Criteriul de convergență poate fi: de la o iterație la alta valoarea lui J nu mai scade semnificativ, sau se atinge un număr maxim de iterații permise.

Putem explicita derivatele parțiale pentru forma funcției de eroare considerate și atunci (2.9) devine:

$$\theta_i := \theta_i - \alpha \left[\frac{1}{m} \sum_{j=1}^m (h_{\theta}(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)} \right] \quad \text{simultan pentru } i = 0, 1 \quad (2.10)$$

Menționăm că pentru o funcție oarecare procedura expusă poate duce la oprirea în minim local. Deoarece gradientul unei funcții în extremele funcției este vectorul zero, rezultă că valoarea parametrilor θ nu se va mai modifica, odată ce s-a atins o valoare de minim al lui J .

Pentru trecerea la cazul multivariat, algoritmul dat mai sus se păstrează cu modificarea domeniului de valori al lui i de la finalul ecuației (2.9): $0 \leq i \leq n$.

Următoarele sugestii trebuie să fie luate în considerare:

- valorile trăsăturilor de intrare să fie în scale similare; dacă acest lucru nu se întâmplă, se recomandă a se face în prealabil o scalare a datelor la un interval convenabil ales, *e.g.* $[0, 1]$; ca efect se obține de regulă un număr mult mai mic de iterații până la convergența algoritmului;
- se vor urmări valorile lui J ; dacă ele au nu o tendință descrescătoare (funcția J crește sau are scăderi urmate de creșteri) atunci se va încerca o valoare mai mică pentru rata de învățare α ;
- dacă valoarea funcției J scade foarte lent se poate mări valoarea lui α .

Pe scurt, valoarea optimă a lui α și modalitatea optimă de modificare a sa depinde de setul de date peste care se calculează funcției de eroare J .

2.4 Metoda algebrică

Există o metodă care dă o soluție pe baza unui calcul algebric.

Vom discuta în cele ce urmează cazul unei funcții liniare multivariate, cu notații din algebra liniară. Pentru problema exemplificată anterior, dorim să facem estimarea valorii unei proprietăți imobiliare considerând pe lângă suprafață și distanța la utilități, gradul de poluare a zonei etc. Pentru început, se va nota cu \mathbf{X} matricea datelor de intrare:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \quad (2.11)$$

unde linia $\mathbf{x}^{(j)}$ conține valorile predictive asociate celui de al j -lea caz din setul de instruire, iar vectorul coloană de indice $1 \leq l \leq n$ corespunde unei trăsături predictive, de exemplu pentru problema noastră: distanța la utilități. Valorile de ieșire corespunzătoare sunt de asemenea stocate matricial, folosind un vector coloană:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \quad (2.12)$$

Modelul de predicție liniar multivariat are forma:

$$h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n \quad (2.13)$$

unde

$$\mathbf{x} = (x_1, \dots, x_n)^t \quad (2.14)$$

Dacă se definește termenul $x_0 = 1$, atunci modelul multivariat se rescrie ca:

$$h_{\theta}(\mathbf{x}) = \sum_{i=0}^n \theta_i \cdot x_i \quad (2.15)$$

Vectorul de intrare \mathbf{x} este¹⁰ $\mathbf{x} = (x_0, \dots, x_n)^t$, vectorul de parametri e $\boldsymbol{\theta} = (\theta_0, \dots, \theta_n)^t$, ambii sunt din \mathbb{R}^{n+1} iar suma din ecuația (2.15) se poate scrie sub forma unui produs scalar de vectori:

$$h_{\theta}(x) = \boldsymbol{\theta}^t \cdot \mathbf{x} \quad (2.16)$$

cu $\mathbf{x} = (1, x_1, \dots, x_n)^t$. Putem extinde matricea de date X din ecuația (2.11) ca având o coloană plină cu valoarea 1 adăugată drept primă coloană; tot pentru simplitatea notațiilor, vom folosi și în continuare litera \mathbf{X} pentru această matrice:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \quad (2.17)$$

Vom folosi următoarele:

- $(A + B)^t = A^t + B^t$;
- $(A_1 \cdot A_2 \cdots A_p)^t = A_p^t \cdot A_{p-1}^t \cdots A_1^t$;

¹⁰Pentru simplitate, vom folosi tot notația \mathbf{x} pentru acest vector.

- dacă a este un număr real, atunci el poate fi interpretat ca o matrice de o linie și o coloană și atunci $a^t = a$.

Funcția J se rescrie matricial astfel:

$$\begin{aligned}
J(\boldsymbol{\theta}) &= \frac{1}{2m} \sum_{j=1}^m \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}) - y^{(j)} \right)^2 \\
&= \frac{1}{2m} \sum_{j=1}^m \left(\boldsymbol{\theta}^T \mathbf{x}^{(j)} - y^{(j)} \right)^2 \\
&= \frac{1}{2m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^t (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\
&= \frac{1}{2m} \left\{ (\mathbf{X}\boldsymbol{\theta})^t \mathbf{X}\boldsymbol{\theta} - (\mathbf{X}\boldsymbol{\theta})^t \mathbf{y} - \mathbf{y}^t (\mathbf{X}\boldsymbol{\theta}) + \mathbf{y}^t \mathbf{y} \right\} \\
&= \frac{1}{2m} \left\{ \boldsymbol{\theta}^t \mathbf{X}^t \mathbf{X} \boldsymbol{\theta} - 2\boldsymbol{\theta}^t \mathbf{X} \mathbf{y} + \mathbf{y}^t \mathbf{y} \right\}
\end{aligned} \tag{2.18}$$

Valorile căutate pentru $\boldsymbol{\theta}$ sunt cele care produc minimul valorii lui J :

$$\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^{n+1}} J(\boldsymbol{\theta}) \tag{2.19}$$

Conform teoremei lui Fermat, o condiție necesară pentru ca $\boldsymbol{\theta}^{(min)}$ să minimizeze pe J este ca vectorul derivatelor parțiale să fie vectorul nul:

$$\frac{\partial J}{\partial \theta_i} \left(\boldsymbol{\theta}^{(min)} \right) = 0, \text{ pentru orice } i \text{ cu } 0 \leq i \leq n \tag{2.20}$$

sau într-o notație mai compactă [19]:

$$\frac{\partial J}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^{(min)} \right) = \mathbf{0} \tag{2.21}$$

unde $\mathbf{0}$ este vectorul coloană de $n + 1$ elemente zero.

Pentru funcția de eroare pătratică J , având în vedere că e și convexă, din ecuația (2.5), condiția necesară de extrem dată de (2.21) este și suficientă pentru a garanta că se ajunge în unicul minim al lui J . Avem (vezi [19], secțiunea 2.4):

$$\frac{\partial J}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^{(min)} \right) = \frac{1}{2m} \left(2\mathbf{X}^t \mathbf{X} \boldsymbol{\theta}^{(min)} - 2\mathbf{X}^t \mathbf{y} \right) \tag{2.22}$$

Impunând condiția (2.21) obținem:

$$\mathbf{X}^t \mathbf{X} \boldsymbol{\theta}^{(min)} = \mathbf{X}^t \mathbf{y} \tag{2.23}$$

numite și “ecuațiile normale”. Mai departe, dacă matricea $\mathbf{X}^t \mathbf{X}$ este nesingulară, vectorul de parametri $\boldsymbol{\theta}^{(min)}$ se determină ca

$$\boldsymbol{\theta}^{(min)} = \left(\mathbf{X}^t \mathbf{X} \right)^{-1} \mathbf{X}^t \cdot \mathbf{y} \tag{2.24}$$

Precizări:

1. Expresia $(\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t$ se mai numește și pseudo-inversa Moore–Penrose; pentru o matrice inversabilă inversa și pseudo-inversa ei coincid; pentru calculul pseudoinversei unei matrice A se poate folosi în Octave și Matlab funcția `pinv`, iar în Python funcția `pinv` din pachetul de algebră liniară `numpy.linalg`;
2. Când se folosește metoda ecuațiilor normale, nu este necesar să se facă scalarea trăsăturilor de intrare, precum se recomandă la metoda iterativă.

Una din problemele care trebuie discutate este: cum se procedează când matricea $\mathbf{X}^t \mathbf{X}$ este singulară? Acest lucru se datorează de regulă uneia din situațiile de mai jos:

- există trăsături de intrare redundante, adică două coloane ale lui \mathbf{X} sunt liniar dependente; în acest caz avem în mod clar o redundanță informațională și putem elimina oricare din aceste două coloane;
- se folosesc prea multe trăsături față de numărul de cazuri din setul de instruire ($m \leq n$); în acest caz se poate renunța la câteva trăsături, adică se elimină coloane din \mathbf{X} , sau se folosește regularizarea – a se vedea secțiunea 2.5.

Ordinea de mai sus este cea sugerată de acționare: se elimină din coloanele redundante, apoi dacă încă e nevoie, se folosește regularizarea.

Dat fiind faptul că avem două metode de determinare a lui $\theta^{(min)}$, se pune problema pe care din ele să o preferăm. Iată câteva comparații:

1. În timp ce pentru metoda gradient descent trebuie ca rata de învățare să fie aleasă cu grijă, pentru varianta algebrică așa ceva nu e necesar;
2. În timp ce metoda de calcul bazată pe gradient descent sunt necesare mai multe iterații, metoda algebrică necesită un singur pas;
3. Metoda bazată pe gradient descent funcționează bine chiar și pentru valori mari ale lui m și n ; pentru valori m sau n mari, calculul pseudoinversei poate fi prohibitiv din punct de vedere al memoriei și timpului de calcul necesar.

2.5 Overfitting, underfitting, regularizare

2.5.1 Overfitting, underfitting

Pentru problema estimării prețului unei proprietăți, să presupunem că există 5 perechi de valori în setul de instruire, o pereche fiind constituită din variabila predictivă *suprafata* și variabila de ieșire *pret*. Să considerăm 3 modele de predicție:

1. polinom de gradul întâi: prețul estimat este de forma:

$$\text{pret} = \theta_0 + \theta_1 x \quad (2.25)$$

2. polinom de gradul al doilea:

$$\text{pret} = \theta_0 + \theta_1 x + \theta_2 x^2 \quad (2.26)$$

3. polinom de gradul 4:

$$\text{pret} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \quad (2.27)$$

unde pentru fiecare caz x este suprafața; spunem că am introdus noi trăsături pe baza celor existente, corespunzătoare mai sus cantităților x^2 , x^3 , x^4 . Primul model este cel liniar discutat până acum, iar celelalte două sunt modele polinomiale (de grad 2, respectiv 4). Ca și mai înainte, se pune problema determinării coeficienților $\theta_0, \theta_1, \dots$.

Graficele celor trei forme polinomiale sunt date în figura 2.6 [1]. Putem considera cantitățile x, x^2, x^3, x^4 ca fiind variabile de intrare pe baza cărora se realizează predicția; faptul că ele provin de la același x (suprafața) este o chestiune secundară.

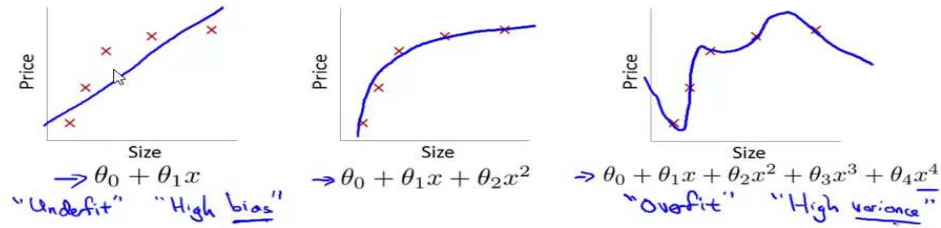


Figura 2.6: Trei polinoame pentru aproximarea prețului pornind de la suprafață, notată x [1].

Intuitiv, polinomul de gradul întâi nu reușește să facă o aproximare prea bună a evoluției prețului față de suprafață. Spunem că modelul dat de primul polinom suferă de “underfitting”¹¹, puterea lui de reprezentare fiind prea slabă pentru problema în cauză. Se mai spune despre un asemenea model că are “high bias”¹², deoarece face o presupunere mult prea simplistă pentru datele disponibile.

Pentru polinomul de grad 4, dacă nu se găsesc două prețuri pe aceeași verticală (adică nu avem două suprafețe egale vândute cu prețuri diferite), se pot determina coeficienții $\theta_0, \dots, \theta_4$ astfel încât curba să treacă prin toate cele 5 puncte (interpolare polinomială). Remarcăm însă forma nemonotonă, cu variații mari a predicției, fiind cazuri în care valoarea estimată scade în raport cu prețul. Intuitiv, modelul suferă de “overfitting”¹³, fiind prea fidel

¹¹Aproximativ, în limba română: incapacitate de reprezentare

¹²TODO: de tradus în romana.

¹³Supraspecializare

construit pe datele din setul de instruire; dacă aceste date conțin zgomot, atunci modelul va învăța perfect zgomotul din setul de date. Deși reprezentarea pe datele de instruire este perfectă, polinomul de gradul 4 dând chiar prețurile cunoscute, în rest nu face o predicție prea credibilă (remarcăm scăderea prețului între al treilea și al patrulea punct din grafic). Se mai spune că modelul are varianță (variabilitate) mare¹⁴, datorită faptului că e prea complex pentru problema tratată.

Polinomul de gradul 2 prezintă cea mai credibilă (în sens intuitiv) formă, chiar dacă nu reprezintă exact cele 5 perechi de valori din setul de instruire. Este un compromis între capacitatea de a reproduce setul de instruire și capacitatea de generalizare, aceasta din urmă fiind abilitatea de a prezice valori de ieșire pentru cazuri care nu fac parte din setul de date de instruire.

Pe scurt, un model care suferă de “underfitting” este incapabil de a reprezenta de antrenare, cât și de a face estimări pentru alte valori. Un model care suferă de “overfitting” poate reprezenta fidel datele din setul de instruire, dar nu reușește să facă estimări prea bune în afara lui; în acest ultim caz spunem că nu generalizează bine, generalizarea fiind capacitatea unui model de a estima cât mai aproape de adevăr în afara cazurilor cu care a fost instruit.

Exemplificarea s-a făcut plecând de la o variabilă predictivă x reprezentând suprafața, care produce alte valori de predicție: x^2, x^3, x^4 . Mai general, putem să presupunem că avem trăsături de intrare definite în domeniul problemei; în cazul nostru, poate fi distanța dintre suprafața respectivă și utilități, gradul de poluare al zonei etc. Trebuie însă să fim capabili să detectăm cazurile de overfitting și underfitting și să le tratăm.

O modalitate de evitare a overfitting-ului este reducerea numărului de trăsături: pentru problema noastră se evită folosirea variabilelor x^3, x^4 . O altă variantă este alegerea judicioasă a modelului de predicție. Cea de a treia opțiune este regularizarea: se păstrează variabilele predictive, dar se impun constrângeri asupra parametrilor modelului — în cazul nostru asupra coeficienților θ_i . Strategia funcționează bine pentru cazul în care toate variabilele predictive contribuie câte puțin la predicția finală.

2.5.2 Regularizare

Să considerăm că predicția se formează pe baza funcției polinomiale

$$h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \quad (2.28)$$

Intuitiv, vrem ca în funcția de eroare să includem o constrângere asupra coeficienților θ_3, θ_4 ; ei se înmulțesc cu cantitățile x^3 , respectiv x^4 care determină o variație rapidă a funcției h ; altfel zis, o modificare mică a cantității x

¹⁴Engl: high variance.

duce la o modificări majore ale lui $h_\theta(x)$. Constrângerea pe care o impunem este deci ca valorile absolute ale lui θ_3 și θ_4 să fie cât mai apropiate de zero.

Pentru aceasta, vom include în expresia funcției de eroare $J(\cdot)$ și pătratele lui θ_3 și θ_4 :

$$J(\theta) = \frac{1}{2m} \sum_{j=1}^m \left(h_\theta(x^{(j)}) - y^{(j)} \right)^2 + 100 \cdot \theta_3^2 + 100 \cdot \theta_4^2 \quad (2.29)$$

Minimizarea lui J din ec. (2.29) va duce la o valoare mică a erorii de predicție dar și la valori absolute mici ale lui θ_3, θ_4 . Exemplul de mai sus este gândit pentru a aduce funcția polinomială de grad patru la una mai apropiată de gradul al doilea, pentru care agreăm ideea că generalizează mai bine.

În general nu știm care dintre coeficienții care se înmulțesc cu puteri nenule ale lui x ar trebui să aibă valori absolute mici. Intuitiv, ne dăm seama că valoarea lui θ_0 nu ar fi necesar a fi supusă unei constrângeri (nu se înmulțește cu nicio variabilă predictivă); vom impune deci constrângeri doar asupra lui $\theta_1, \theta_2, \dots$ – să aibă valori absolute mici. Scopul final este de a evita overfitting-ul. Principiul se aplică și dacă se pleacă de la variabile de intrare independente, nu neapărat *suprafata, suprafata*², \dots . Ca atare, ec. (2.29) se rescrie mai general ca:

$$J(\theta) = \frac{1}{2m} \left[\sum_{j=1}^m \left(h_\theta(x^{(j)}) - y^{(j)} \right)^2 + \lambda \sum_{i=1}^n \theta_i^2 \right] \quad (2.30)$$

unde $\lambda > 0$. Cu cât λ e mai mare, cu atât constrângerea impusă coeficienților $\theta_1, \dots, \theta_n$ e mai pronunțată. La extrem, dacă $\lambda \rightarrow \infty$ atunci se ajunge la $h_\theta(\mathbf{x}) = \theta_0$, ceea ce aproape sigur înseamnă underfitting (model de aproximare prea simplist): funcția de estimare returnează mereu θ_0 , indiferent de intrare.

Algoritmul de căutare după direcția gradientului devine (considerăm că avem coeficienții $\theta_0, \dots, \theta_n$ inițializați aleator):

repetă{

$$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{j=1}^m (h_\theta(x^{(j)}) - y^{(j)}) \cdot x_0^{(j)} \right]$$

$$\theta_i := \theta_i - \alpha \left[\frac{1}{m} \sum_{j=1}^m (h_\theta(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)} + \frac{\lambda}{m} \cdot \theta_i \right], \quad i = 1, \dots, n$$

} pana la convergenta

Pentru metoda algebrică se poate arăta că regularizarea produce următoarea valoare pentru $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \left(\mathbf{X}^t \mathbf{X} + \lambda \cdot \underbrace{\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}}_{(n+1) \times (n+1)} \right)^{-1} \cdot \mathbf{X}y \quad (2.31)$$

unde matricea care se înmulțește cu λ se obține din matricea unitate de ordinul $n + 1$, modificând primul element în 0.

Capitolul 3

Regresia logistică

3.1 Încadrare, motivație

Regresia logistică este folosită pentru estimare de probabilitate condiționată și clasificare. Inițial dezvoltată pentru lucrul cu două clase, a fost extinsă pentru a discrimina între oricâte clase — regresia logistică multinomială.

Ca mod de instruire face parte din învățarea supervizată. Intrările sunt vectori numerici, iar clasele sunt fie două (pentru regresia logistică), fie mai multe (pentru regresia logistică multinomială).

Exemple de probleme de clasificare cu două clase, tratate de regresia logistică, sunt:

- clasificarea unui email ca fiind de tip spam sau nonspam, dându-se conținutul lui, subiectul emailului, faptul că expeditorul face sau nu parte din lista de contacte etc.
- clasificarea unei tumori ca fiind benignă sau malignă, date fiind rezultatele unor analize;
- clasificarea unui fruct dintr-o imagine ca fiind măr sau pară.

Exemple de probleme pentru care există mai mult de două clase sunt:

- clasificarea unui email ca fiind de tip: știri, muncă, prieteni, anunțuri, spam etc.;
- clasificarea unui fruct dintr-o imagine ca fiind măr, pară, banană, cireșă etc.

De fapt, modelul dat de regresia logistică (fie ea simplă sau multinomială) construiește o estimare a probabilității condiționate, dată fiind intrarea curentă (conținut email, imagine cu fructe etc.); mai precis, se determină

care este probabilitatea ca obiectul descris de vectorul de intrare specificat să fie dintr-o clasă sau alta; matematic, se scrie

$$P(\text{clasa}_i | \text{intrare}) \quad (3.1)$$

Faptul că se estimează probabilități, adică valori continue din $[0, 1]$ justifică cuvântul “regresie” din denumirile metodelor. Clasificarea se face prin determinarea acelei clase_i pentru care probabilitatea din ecuația (3.1) este maximă.

3.2 Regresia logistică

3.2.1 Setul de instruire

În cazul regresiei logistice se urmărește clasificarea între două clase. Clasele sunt convenabil date ca fiind “0” (clasa negativă) și respectiv “1” (clasa pozitivă). Setul de instruire este de forma:

$$\mathcal{S} = \{(\mathbf{x}^{(j)}, y^{(j)}) | 1 \leq j \leq m\} \quad (3.2)$$

unde $\mathbf{x}^{(j)} = (x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)})^t \in \mathbb{R}^{n+1}$, $y_j \in \{0, 1\}$ (simbolul t reprezintă transpunerea unui vector sau a unei matrice). Ca și în capitolul precedent, vom considera că $x_0^{(j)} = 1$ pentru orice j .

3.2.2 Reprezentarea ipotezei

Pentru regresia logistică modelul de predicție trebuie să producă o valoare reprezentând probabilitatea condiționată (3.1). Vom folosi în acest scop o funcție $h_{\boldsymbol{\theta}}(\cdot)$ cu proprietatea $0 \leq h_{\boldsymbol{\theta}}(\mathbf{x}) \leq 1$:

$$P(y = 1 | \mathbf{x}; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \quad (3.3)$$

unde $\boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_n) \in \mathbb{R}^{n+1}$ este un vector de $n+1$ coeficienți (ponderi) ce vor fi determinați prin procesul de învățare. Probabilitatea din (3.3) este o probabilitate condiționată de intrarea curentă (în cazul nostru: vectorul) \mathbf{x} și parametrizată de $\boldsymbol{\theta}$. Se exprimă aici că probabilitatea este estimarea încrederii că \mathbf{x} face parte din clasa 1, dar modelul de probabilitate este totodată influențat de parametrul $\boldsymbol{\theta}$.

Funcția care stă la baza definirii modelului $h_{\boldsymbol{\theta}}$ este:

$$\sigma : \mathbb{R} \rightarrow (0, 1), \sigma(z) = \frac{1}{1 + \exp(-z)} \quad (3.4)$$

și numită sigmoida logistică, sau funcția logistică, reprezentată grafic în figura 3.1. După cum se remarcă, codomeniul funcției logistice este $(0, 1)$,

deci compatibil cu valorile admisibile pentru funcție de probabilitate. Avem că funcția g este derivabilă, monoton (strict) crescătoare, $\lim_{z \rightarrow -\infty} g(z) = 0$, $\lim_{z \rightarrow \infty} g(z) = 1$. Denumirea de “sigmoidă” este dată de alura graficului, amintind de litera S.

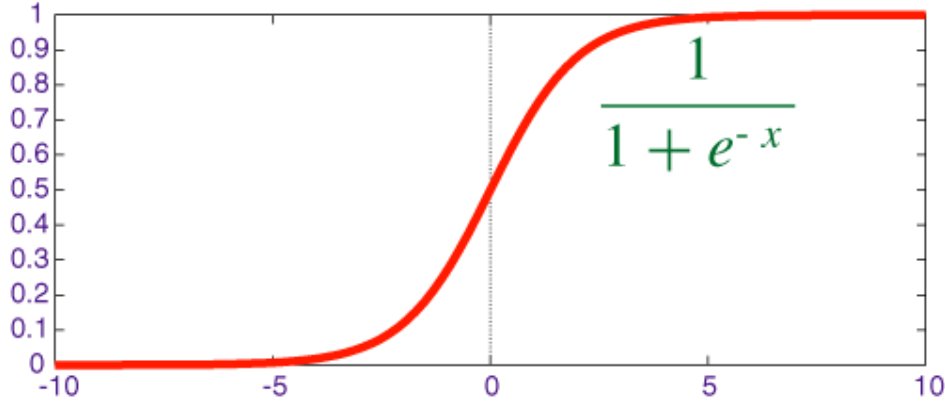


Figura 3.1: Sigmoida logistică definită în ecuația 3.4

Probabilitatea evenimentului contrar $P(y = 0|\mathbf{x}; \boldsymbol{\theta})$ este complementul față de 1 al evenimentului $y = 1$ dat fiind vectorul \mathbf{x} , adică:

$$P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \quad (3.5)$$

3.2.3 Suprafața de decizie a regresiei logistice

Modelul probabilist din ecuația (3.3) este mai departe folosit pentru a face clasificare, astfel: dacă $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta})$ atunci se estimează că obiectul descris de vectorul \mathbf{x} este din clasa 1 (pozitivă), altfel din clasa 0 (negativă).

În pofida caracterului nelinier al funcției ce definește modelul conform ecuației (3.3), se arată ușor că suprafața care desparte regiunea \mathbf{x} de clasă pozitivă și cea de clasă negativă este o varietate liniară¹.

Inegalitatea de mai sus se scrie:

$$\begin{aligned} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) & \iff \frac{1}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \geq \frac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ & \iff 1 \geq \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x}) \\ \text{(logaritmând)} & \iff 0 \geq -\boldsymbol{\theta}^t \cdot \mathbf{x} \\ & \iff \boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0 \end{aligned} \quad (3.6)$$

¹Prin abuz se folosește și denumirea “hiperplan”; în timp ce un hiperplan obligatoriu trebuie să treacă prin origine, varietatea liniară este doar o formă liniară fără această constrângere.

Am obținut deci că dacă \mathbf{x} are proprietatea că $\boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0$ atunci \mathbf{x} este clasificat ca fiind de clasă 1, altfel este de clasă 0. Separarea dintre cele două clase se face de către varietatea liniară $\boldsymbol{\theta}^t \cdot \mathbf{x} = 0$: dacă \mathbf{x} e în partea pozitivă a varietății liniare $\boldsymbol{\theta}^t \cdot \mathbf{x} = 0$ sau chiar pe această varietate, atunci e de clasă 1, altfel e de clasă 0.

Dacă se permite ca în componența vectorului \mathbf{x} să intre și forme pătratice, cubice etc. bazate pe trăsăturile originare, atunci suprafața de decizie poate fi mai complicată. De exemplu, să considerăm că $\mathbf{x} = (x_0 = 1, x_1, x_2, x_1^2, x_2^2, x_1x_2)^t$; rezultă că $\boldsymbol{\theta} \in \mathbb{R}^6$; avem că $\boldsymbol{\theta}^t \cdot \mathbf{x} = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_1^2 + \theta_4x_2^2 + \theta_5x_1x_2$. Pentru valorile² $\theta_0 = -4, \theta_1 = \theta_2 = \theta_5 = 0, \theta_3 = \theta_4 = 1$ se obține ecuația suprafeței de decizie $x_1^2 + x_2^2 = 4$, reprezentând un cerc; în funcție de poziția față de cerc (înăuntrul sau înafara lui), obiectul de coordonate $(x_1, x_2)^t$ este estimat ca fiind de clasă pozitivă sau negativă.

3.2.4 Funcția de cost

Funcția care ne permite să estimăm cât de bun este un vector de ponderi $\boldsymbol{\theta}$ și care e de asemenea utilizată pentru ajustarea acestor ponderi este notată tradițional cu $J(\cdot)$, argumentul ei fiind $\boldsymbol{\theta}$. Valoarea se va calcula peste setul de instruire \mathcal{S} din ecuația (3.2).

O variantă este dată de utilizarea aceleiași funcții de eroare din capitolul de regresie liniară, ecuația (2.18) pagina 25. Se arată însă că pentru problema estimării de probabilitate condiționată, dată fiind forma funcției (modelului) $h_{\boldsymbol{\theta}}$ din ecuația (3.3), funcția de eroare nu mai este convexă și în acest caz o căutare bazată pe gradient se poate opri într-un minim local.

Vom defini J de așa manieră încât să fie convexă:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^m \text{Cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}), y^{(j)}) \quad (3.7)$$

unde $\text{Cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}), y^{(j)})$ ar trebui să verifice următoarele condiții:

1. dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ și y sunt apropiate, atunci valoarea lui $\text{Cost}(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie apropiată de 0, și reciproc;
2. dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ și y sunt îndepărtate, atunci valoarea lui $\text{Cost}(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie mare, și reciproc;

Definim $\text{Cost}(\cdot, \cdot)$ astfel:

$$\text{Cost}(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\log h_{\boldsymbol{\theta}}(\mathbf{x}) & \text{dacă } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{dacă } y = 0 \end{cases} \quad (3.8)$$

baza logaritmului fiind orice număr supraunitar.

²Valorile lui $\boldsymbol{\theta}$ în mod normal se determină prin proces de instruire.

Cele două ramuri ale funcției *Cost* sunt reprezentate în figura 3.2. Dreptele $x = 0$ și respectiv $x = 1$ sunt asimptote verticale pentru cele două ramuri ale lui *Cost*.

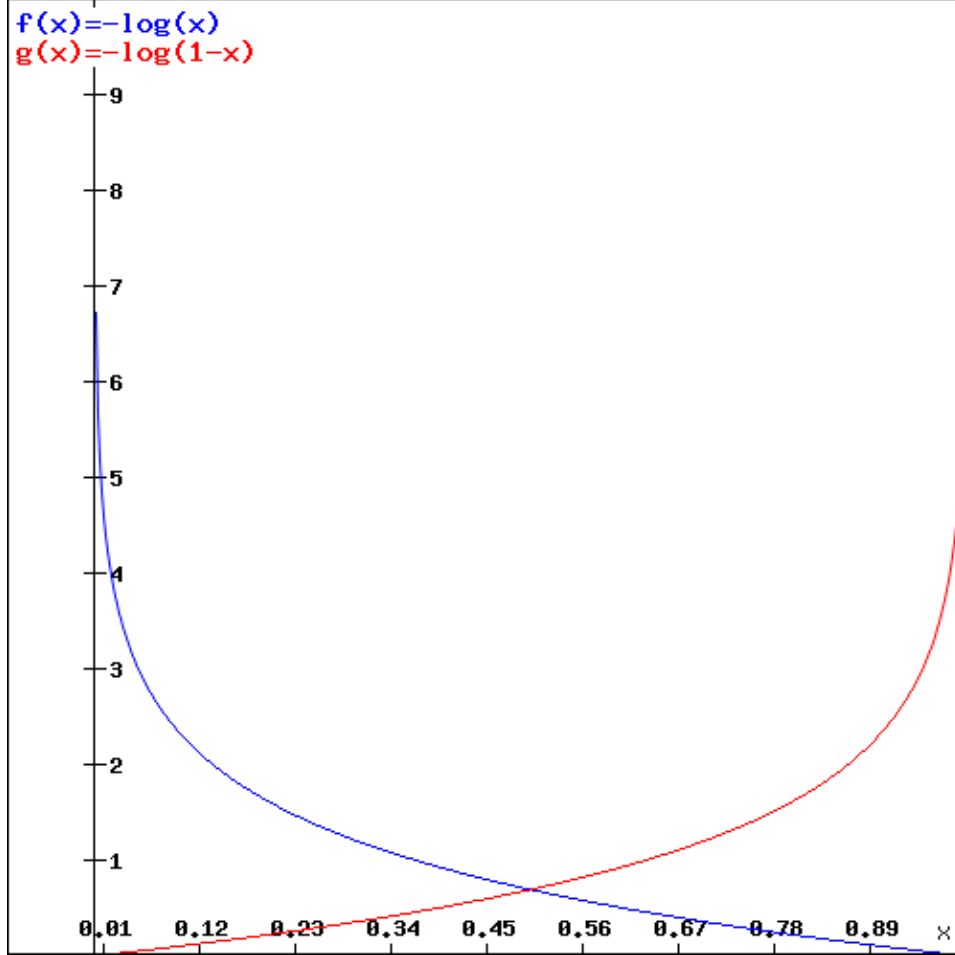


Figura 3.2: Cele două ramuri ale funcției *Cost* din eq. (3.8). Pe axa Ox este reprezentată variabila $h_{\theta}(\mathbf{x})$.

Rescriem funcția *Cost* sub forma:

$$Cost(h_{\theta}(\mathbf{x}), y) = -y \cdot \log h_{\theta}(\mathbf{x}) - (1 - y) \cdot \log(1 - h_{\theta}(\mathbf{x})) \quad (3.9)$$

Să verificăm că se îndeplinesc cele două condiții cerute mai sus. Pentru prima condiție, avem:

$$Cost(h_{\theta}(\mathbf{x}), y) \approx 0 \Leftrightarrow -y \cdot \log h_{\theta}(\mathbf{x}) - (1 - y) \cdot \log(1 - h_{\theta}(\mathbf{x})) \approx 0 \quad (3.10)$$

Pentru cazul $y = 1$, (3.10) devine:

$$Cost(h_{\theta}(\mathbf{x}), y) \approx 0 \Leftrightarrow -\log h_{\theta}(\mathbf{x}) \approx 0 \Leftrightarrow h_{\theta}(\mathbf{x}) \approx 1 \Leftrightarrow h_{\theta}(\mathbf{x}) \approx y$$

Pentru cazul $y = 0$, (3.10) devine:

$$Cost(h_{\theta}(\mathbf{x}), y) \approx 0 \Leftrightarrow -\log(1 - h_{\theta}(\mathbf{x})) \approx 0 \Leftrightarrow h_{\theta}(\mathbf{x}) \approx 0 \Leftrightarrow h_{\theta}(\mathbf{x}) \approx y$$

deci prima condiție, legată de valoarea apropiată de zero a cantității $Cost(h_{\theta}(\mathbf{x}), y)$ este îndeplinită de definiția funcției $Cost$ din ecuația (3.8).

Pentru cea de a doua condiție, dacă $Cost(h_{\theta}(\mathbf{x}), y) = M > 0$, atunci obținem echivalent $h_{\theta}(\mathbf{x}) = \exp(-M)$ (pentru $y = 1$) respectiv $h_{\theta}(\mathbf{x}) = 1 - \exp(-M)$ (pentru $y = 0$); y este unul din capetele intervalului $[0, 1]$, iar dacă M este o valoare din ce în ce mai mare, atunci $h_{\theta}(\mathbf{x})$ se îndreaptă spre celălalt capăt al intervalului unitate. Rezultă deci că și a doua condiție, vizând valoarea mare a lui $Cost(h_{\theta}(\mathbf{x}), y)$ este îndeplinită de definiția din formula (3.8).

În plus, se poate arăta ușor că funcția $Cost$ este convexă, deci funcția J , ca sumă a unui număr finit de funcții convexe, este ea însăși convexă. Ca atare, orice punct de minim al lui J este garantat și minim global; acest lucru este deosebit de util și justifică forma aparte a funcției de cost din ecuația (3.8).

Funcția de eroare J se rescrie astfel:

$$J(\theta) = -\frac{1}{m} \sum_{j=1}^m \left[y^{(j)} \cdot \log h_{\theta}(\mathbf{x}^{(j)}) + (1 - y^{(j)}) \cdot \log(1 - h_{\theta}(\mathbf{x}^{(j)})) \right] \quad (3.11)$$

și această formă se numește cross-entropy.

3.2.5 Algorimul de instruire

Setul de antrenare \mathcal{S} este utilizat pentru a deduce valori adecvate ale lui θ , astfel încât predicțiile date de model pentru valorile de intrare $\mathbf{x}^{(j)}$ să fie cât mai apropiate de valoarea actuală a etichetelor corespunzătoare $y^{(j)}$. Datorită proprietăților funcției $Cost$ din ecuația (3.8) și a faptului că J este valoarea medie a funcției $Cost$ peste setul de instruire, deducem că minimizând valoarea lui J , obținem (în medie) valori $h_{\theta}(\mathbf{x}^{(j)})$ apropiate de $y^{(j)}$.

Pentru determinarea lui $\theta^{(min)} = \arg \min_{\theta} J(\theta)$ se folosește algoritmul de căutare după direcția gradientului: se pornește cu valori aleatoare setate componentelor vectorului³ θ și se modifică în mod iterativ, scăzând la fiecare pas valoarea gradientului înmulțită cu un coeficient pozitiv mic (rata de învățare).

Formal, algoritmul are forma:

1. Setează componentele lui θ la valori inițiale aleatoare;
2. Iterații:

³Se poate seta chiar ca θ să fie vectorul nul.

repetă{

$$\theta_i := \theta_i - \alpha \cdot \frac{\partial J}{\partial \theta_i}(\boldsymbol{\theta}) \quad \text{simultan pentru } i = 0, 1, \dots, n$$

} pana la convergenta

unde $\alpha > 0$ este rata de învățare. Condiția de convergență este la fel ca la regresia liniară.

Explicitând derivatele parțiale ale lui J obținem:

repetă{

$$\theta_i := \theta_i - \alpha \frac{1}{m} \sum_{j=1}^m (h_{\boldsymbol{\theta}}(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)} \quad \text{simultan pentru } i = 0, \dots, n$$

} pana la convergenta

Se observă că modificarea ponderilor $\theta_0, \dots, \theta_n$ are aceeași formă ca la regresia liniară. Singura diferență este forma funcției $h_{\boldsymbol{\theta}}$.

3.2.6 Regularizare

Dacă se permite ca valorile parametrilor $\theta_1, \dots, \theta_n$ ⁴ să fie lăsate neconstrânse, atunci valorile lor absolute pot crește și influența negativ performanța de generalizare a modelului: pentru variații mici ale datelor de intrare vom avea variații mari ale valorilor funcției model; mai mult, dacă luăm în considerare trăsături de intrare de forma $x_i \cdot x_j$, $x_i \cdot x_j \cdot x_k$ etc. modelul poate ajunge să aproximeze foarte bine perechile din setul de instruire, dar fără a avea o performanță bună pe datele din set de testare. Ca atare, se preferă impunerea unor constrângeri parametrilor $\theta_1, \dots, \theta_n$ astfel încât aceștia să fie cât mai mici în valoare absolută.

Pentru regularizare se modifică forma funcției de eroare astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{j=1}^m \left[y^{(j)} \cdot \log h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}) + (1 - y^{(j)}) \cdot \log(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)})) \right] + \frac{\lambda}{2m} \sum_{i=1}^n \theta_i^2 \quad (3.12)$$

Modificarea adusă algoritmului de instruire este simplă: mai trebuie inclusă și derivata parțială a lui θ_i^2 :

repetă{

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^m (h_{\boldsymbol{\theta}}(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)} \\ \theta_i &:= \theta_i - \alpha \frac{1}{m} \sum_{j=1}^m (h_{\boldsymbol{\theta}}(x^{(j)}) - y^{(j)}) \cdot x_i^{(j)} + \frac{\lambda}{m} \theta_i \end{aligned}$$

⁴Se remarcă lipsa termenului θ_0 ; coeficientul θ_0 nu este regularizat.

} pana la convergenta

unde atribuirile se fac simultan pentru inidicii $0, 1, \dots, n$ ai lui θ .

3.3 Regresia logistică multinomială

Pentru cazul în care se cere discriminarea pentru mai mult de două clase, regresia logistică a fost extinsă să lucreze cu mai multe funcții (modele h) simultan.

Pentru acest caz considerăm că avem $k > 2$ clase, iar eticheta unei clase este un număr $j, 1 \leq j \leq k$.

3.3.1 Reprezentarea ipotezei

Punctul de plecare pentru reprezentarea modelului de estimare de probabilitate condiționată este funcția *softmax*, care transformă un vector de numere într-un vector de valori din intervalul $(0, 1)$ și care însumate dau 1: dacă pornim de la vectorul $\mathbf{z} = (z_1, \dots, z_k)^t \in \mathbb{R}^k$, el e transformat de funcția softmax astfel:

$$\text{softmax}(\mathbf{z}) = (\text{softmax}(\mathbf{z}; 1), \dots, \text{softmax}(\mathbf{z}; k))^t \quad (3.13)$$

unde valoarea reală $\text{softmax}(\mathbf{z}; l)$ e:

$$\text{softmax}(\mathbf{z}; l) = \frac{\exp(z_l)}{\sum_{i=1}^k \exp(z_i)} \quad (3.14)$$

pentru $1 \leq l \leq k$. Dat fiind că $\text{softmax}(\mathbf{z}; l) \in (0, 1)$ și $\sum_{l=1}^k \text{softmax}(\mathbf{z}; l) = 1$, vom folosi funcția *softmax* pentru producerea de estimări de probabilități condiționate de intrarea \mathbf{x} .

Dându-se o intrare \mathbf{x} , ipotezele construite trebuie să estimeze probabilitatea ca intrarea să fie de clasă l , $1 \leq l \leq k$, adică $P(y = l | \mathbf{x}; \theta)$. Ipoteza ia forma:

$$h_{\theta}(\mathbf{x}) = \begin{pmatrix} P(y = 1 | \mathbf{x}; \theta) \\ P(y = 2 | \mathbf{x}; \theta) \\ \vdots \\ P(y = k | \mathbf{x}; \theta) \end{pmatrix} = \frac{1}{\sum_{l=1}^k \exp(\theta_l^t \cdot \mathbf{x})} \begin{pmatrix} \exp(\theta_1^t \cdot \mathbf{x}) \\ \exp(\theta_2^t \cdot \mathbf{x}) \\ \vdots \\ \exp(\theta_k^t \cdot \mathbf{x}) \end{pmatrix} \quad (3.15)$$

Fiecare clasă l are asociat propriul vector de ponderi θ_l ; aceștia vor fi determinați prin învățare. Impreună, vectorii θ_l definesc matricea θ (ce apare

ca indice al funcției model din ecuația (3.15)), prin stivuirea pe orizontală a transpuselor lor:

$$\boldsymbol{\theta} = \begin{pmatrix} \boldsymbol{\theta}_1^t \\ \boldsymbol{\theta}_2^t \\ \dots \\ \boldsymbol{\theta}_k^t \end{pmatrix} \quad (3.16)$$

deci matricea $\boldsymbol{\theta}$ e de forma $k \times (n + 1)$.

Mai clar, probabilitatea ca un vector \mathbf{x} să fie de clasă l ($1 \leq l \leq k$) este:

$$P(y = l | \mathbf{x}; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta}_l^t \cdot \mathbf{x})}{\sum_{h=1}^k \exp(\boldsymbol{\theta}_h^t \cdot \mathbf{x})} \quad (3.17)$$

Numitorul ultimului termen din ecuația (3.15) are rolul de a face ca toate probabilitățile condiționate să se însumeze la 1:

$$\sum_{l=1}^k P(y^{(j)} = l | x^{(j)}; \boldsymbol{\theta}) = 1 \quad (3.18)$$

3.3.2 Funcția de cost

Ca și în cazul funcției de eroare pentru regresia logistică, se va cuantifica diferența dintre clasa actuală a unei intrări oarecare și predicția dată de model. Introducem funcția indicator $I(\cdot)$ care pentru o valoare logică returnează 1 dacă valoarea este adevărată și 0 altfel:

$$I(\text{valoare_logica}) = \begin{cases} 1 & \text{dacă } \text{valoare_logica} = \text{adevarat} \\ 0 & \text{dacă } \text{valoare_logica} = \text{fals} \end{cases} \quad (3.19)$$

Funcția de eroare pentru regresia logistică multinomială se definește ca:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \left[\sum_{j=1}^m \sum_{l=1}^k I(y^{(j)} = l) \cdot \log \frac{\exp(\boldsymbol{\theta}_l^t \mathbf{x}^{(j)})}{\sum_{h=1}^k \exp(\boldsymbol{\theta}_h^t \mathbf{x}^{(j)})} \right] \quad (3.20)$$

3.3.3 Algoritmul de instruire

Determinarea lui $\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ se face prin căutarea după direcția gradientului⁵. Formula gradientului este:

$$\frac{\partial J}{\partial \boldsymbol{\theta}_l}(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{j=1}^m \left[x^{(j)} \left(I(y^{(j)} = l) - P(y^{(j)} = l | \mathbf{x}^{(j)}; \boldsymbol{\theta}) \right) \right] \quad (3.21)$$

⁵Spre deosebire de regresia liniară, nu există o variantă algebrică de determinare a valorilor din matricea $\boldsymbol{\theta}$.

unde $\frac{\partial J}{\partial \theta_j}(\theta)$ este un vector. Modificarea vectorului θ_l este:

$$\theta_l := \theta_l - \alpha \cdot \frac{\partial J}{\partial \theta_l}(\theta) \quad (3.22)$$

unde $1 \leq l \leq k$.

3.3.4 Regularizare

Din rațiuni similare cu cele prezentate la regresia liniară, se preferă penalizarea valorilor absolute mari ale parametrilor θ_{li} . Funcția de eroare se modifică astfel:

$$J(\theta) = -\frac{1}{m} \left[\sum_{j=1}^m \sum_{l=1}^k I(y^{(j)} = l) \cdot \log \frac{\exp(\theta_l^t \mathbf{x}^{(j)})}{\sum_{h=1}^k \exp(\theta_h^t \mathbf{x}^{(j)})} \right] + \frac{\lambda}{2} \sum_{l=1}^k \sum_{i=1}^n \theta_{li}^2 \quad (3.23)$$

Coeficientul λ este un hiperparametru care arată cât de mult contează regularizarea în cadrul funcției de eroare; evident, pentru $\lambda = 0$ nu avem regularizare, iar dacă λ se alege foarte mare, atunci funcția de eroare cross-entropy este neglijată în favoarea micșorării valorii coeficienților θ_{li} .

Formula gradientului folosit în căutarea de tip gradient descent este:

$$\frac{\partial J}{\partial \theta_l}(\theta) = -\frac{1}{m} \sum_{j=1}^m \left[x^{(j)} \left(I(y^{(j)} = l) - P(y^{(j)} = l | \mathbf{x}^{(j)}; \theta) \right) \right] + \lambda \theta_l \quad (3.24)$$

Valoarea hiperparametrului λ se determină prin încercări repetate, cross-validation, căutare aleatoare, optimizare bayesiană, algoritmi genetici sau alte euristici de căutare.

Capitolul 4

Rețele neurale artificiale - fundamente

4.1 Încadrarea domeniului

Studiul rețelilor neurale artificiale este motivat de observația că un sistem biologic este mai performant decât calculatoarele și programele existente la ora actuală într-o serie de sarcini precum recunoașterea de imagini, regăsirea informației, înțelegerea vorbirii. Acest lucru se întâmplă cu toate că neuronii biologici operează mult mai lent decât procesoarele actuale. Studiul rețelilor neurale artificiale are ca scop producerea unor sisteme care să obțină rezolvări pentru probleme de tipul celor de mai sus, dar și altele de natură sintetică, pe baza experienței acumulate din mediu.

Definiția următoare ([16]) ia în considerare abilitatea de adaptare pe baza experienței:

Definiția 4. *Sistemele neurale artificiale, sau rețelele neurale sunt sisteme celulare fizice care pot achiziționa, stoca și utiliza cunoașterea experimentală.*

Natura neliniară, complexă și cu grad mare de paralelism reprezintă diferențe majore față de modelele de calcul actuale. O rețea neurală artificială modelează felul în care creierul biologic procesează semnalele. Modelele de rețele neurale artificiale sunt structurate ca niște grafuri ale căror noduri sunt neuroni artificiali, iar legăturile dintre noduri au ponderi¹ - valori numerice - ajustabile printr-un proces de învățare. Definiția din [5] sumarizează acest fapt:

Definiția 5. *O rețea neurală este un sistem de procesare masiv paralel-distribuit constând din unități de procesare simple, care au predispoziție naturală pentru stocarea cunoștințelor experimentale și utilizarea lor. Este asemănătoare creierului în două aspecte:*

¹În limba engleză: weights.

1. *Cunoașterea este achiziționată de către rețea din mediu printr-un proces de învățare;*
2. *Ponderile legăturilor dintre neuroni, cunoscute ca ponderi sinaptice sunt folosite pentru a reține cunoașterea dobândită.*

Procedura prin care se obține adaptarea ponderilor din cadrul rețelei se numește *algorithm de învățare*. Mai menționăm că învățarea poate duce la modificarea numărului de noduri din rețea (a se vedea de exemplu Fuzzy ARTMAP, capitolul 9), ceea ce în cadrul rețelelor neurale biologice are ca și corespondent faptul că unii neuroni mor (efectul de rețezare din rețele neurale) și alți neuroni se pot alătura celor existenți pentru a sprijini învățarea.

Numele sub care mai sunt cunoscute aceste rețele sunt: neurocalculatoare, rețele conexiuniste, sisteme adaptive stratificate, rețele cu auto-organizare, sisteme neuromorfe etc.

Există multe moduri în care pot fi privite aceste rețele neurale de către diferite categorii profesionale:

- oamenii de știință din domeniul neurobiologiei sunt interesați de modul în care rețelele neurale artificiale confirmă rezultatele sau modelele cunoscute pentru sisteme biologice; facem precizarea că transferul de cunoștințe nu este doar dinspre biologie spre domeniul artificial, ci și invers: modele teoretice sunt confirmate de descoperirile biologice²;
- fizicienii văd analogii între rețelele neurale și sistemele dinamice neliniare pe care le studiază;
- matematicienii sunt interesați de potențialul de modelare matematică pentru sisteme complexe;
- inginerii din domeniul electric le folosesc pentru procesarea de semnale;
- informaticienii sunt interesați de oportunitățile care apar în zonele de inteligență artificială, teorie computațională, modelare și simulare etc.

Beneficiile aduse de rețele neurale artificiale sunt:

1. neliniaritatea: un neuron artificial poate avea comportament liniar sau nu; caracteristica neliniară este importantă pentru cazul în care mecanismul care generează semnalul este neliniar - de exemplu semnalul de vorbire;
2. detectarea de asocieri între intrări și ieșiri: este cazul învățării supervizate, în care antrenarea se face pe baza unor perechi de semnale,

²Vezi de exemplu "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity", *A mechanism predicted theoretically by a Coneural scientist has been found experimentally in the brain.*

corespunzătoare intrărilor și respectiv ieșirilor asociate. Se poate pleca de la un model care nu are cunoștințe apriori despre domeniu și pe baza datelor se învață asocierea.

3. adaptabilitate: rețelele neurale au capacitatea naturală de a-și adapta ponderile în funcție de semnalele provenite din mediu; mediul poate să fie nestăionar, adică să sufere modificări în ceea ce privește distribuția semnalelor sau a asocierilor;
4. rezistența la zgomot: o rețea neurală poate să accepte date care au imprecizie sau sunt afectate de zgomot; sunt raportate multe situații în care adăugarea de zgomot la datele de antrenare îmbunătățește calitatea învățării.

4.2 Neuronul biologic

Este utilă o scurtă incursiune în biologie, pentru a înțelege modelarea ce se face pentru un neuron artificial.

Neuronul biologic este o celulă nervoasă elementară, elementul constructiv de bază pentru rețeaua neurală biologică. Neuronul are trei părți principale: corpul celulei, numit și *soma*, *axonul* și *dendritele*. O schemă a unui neuron a fost dată în figura 1.1. Dendritele formează o arborescență prin care sunt primite impulsuri de la alți neuroni. Axonul este un fir conductor lung, cu un capăt în corpul celulei iar cu celălalt ramificat, prin care se trimite semnal către dendritele altor axoni. Contactul dintre un axon și o dendrită se cheamă sinapsă. Ca mediator ai semnalului în sinapse se folosesc adrenalina, noradrenalina, acetilcolina, serotonina.

Neuronul este capabil să dea un răspuns pe baza intrărilor furnizate de către dendrite. Răspunsul acesta este generat dacă potențialul membranei depășește un anumit prag de activare. Impulsurile care vin prin membrană sunt *excitatoare* dacă ele favorizează formarea semnalului de ieșire din neuron și *inhibitoare* dacă inhibă răspunsul. Se face o agregare a semnalelor primite de celulă de-a lungul unei perioade de sumare latentă, luându-se în considerare și apropierea în timp a semnalelor excitatoare primite; nu se cere o sincronizare a semnalelor, iar valoarea de ieșire este de regulă văzut ca una binară: dacă suma semnalelor primite este mai mare decât pragul de activare (nu contează cu cât mai mare) se trimite semnal mai departe prin axon, către alți neuroni; dacă este mai mic, atunci nu se face trimitere.

După emiterea semnalului prin axon există o perioadă refractară, în care neuronul nu mai ia în considerare niciun semnal sosit, indiferent de gradul de intensitate. După scurgerea acestei perioade, neuronul este gata să proceseze noi semnale. Perioada refractară nu are neapărat aceeași durată pentru toate celulele. Timpul necesar acestui ciclu este de ordinul milisecundelor.

Facem precizarea că prezentarea făcută este o variantă simplificată; considerarea și modelarea unor detalii duce la rețele neurale artificiale de diferite tipuri (generații).

4.3 Modele de neuroni artificiali

4.3.1 Modelul McCulloch–Pitts

Reprezintă prima definiție formală a unui neuron artificial, ce pleacă de la descrierea a neuronului biologic. Modelul a fost formulat în 1943 de neurobiologul și ciberneticianul Warren McCulloch și respectiv logicianul Walter Pitts. Modelul este arătat în figura 4.1. Intrările x_i sunt 0 sau 1, $i = \overline{1, n}$, ponderile $w_i \in \{-1, 1\}$, T este pragul neuronului iar ieșirea o se calculează ca:

$$o^{k+1} = \begin{cases} 1 & \text{dacă } \sum_{i=1}^n w_i x_i^k \geq T \\ 0 & \text{altfel} \end{cases} \quad (4.1)$$

unde indicele superior k este momentul de timp, $k = 0, 1, \dots$. Ponderile $w_i = 1$ reprezintă sinapsele excitatoare, cele cu valoare -1 sunt inhibitoare. Cu toate că modelul este simplu, el poate fi folosit pentru implementarea operațiilor logice **not**, **and** și **or**, dacă ponderile și pragul sunt setate corespunzător.

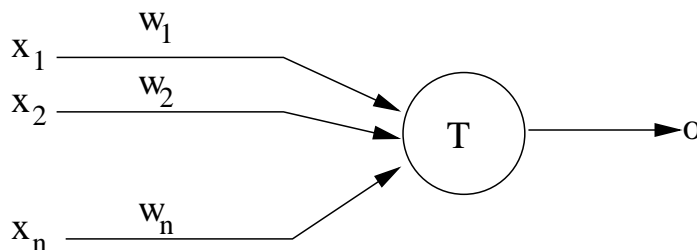


Figura 4.1: Modelul McCulloch-Pitts pentru neuron artificial.

4.3.2 Modelarea neuronului pentru sisteme neurale artificiale

Modelul McCulloch-Pitts este o viziune simplificată: permite doar intrări, ieșiri și ponderi binare, operează în timp discret, presupune sincronizarea intrărilor (toate intrările trebuie să sosească în același timp), ponderile și pragul sunt fixe. Ca atare, se propune modelul dat în figura 4.2 cu următoarele precizări:

1. fluxul semnalului de intrare x_i este unidirecțional

2. valoarea de ieșire a neuronului este:

$$o = f(\mathbf{w}^t \cdot \mathbf{x}) \quad (4.2)$$

adică

$$o = f\left(\sum_{i=1}^n w_i x_i\right) \quad (4.3)$$

unde vectorul ponderilor \mathbf{w} este

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = (w_1 \ w_2 \ \dots \ w_n)^t$$

iar vectorul \mathbf{x} al intrărilor este

$$\mathbf{x} = (x_1, x_2, \dots, x_n)^t$$

3. valoarea produsului scalar $\mathbf{w}^t \cdot \mathbf{x}$ se notează cu *net* și se numește valoare de activare a neuronului;

4. funcția f se numește *funcție de activare*; poate avea diferite forme.

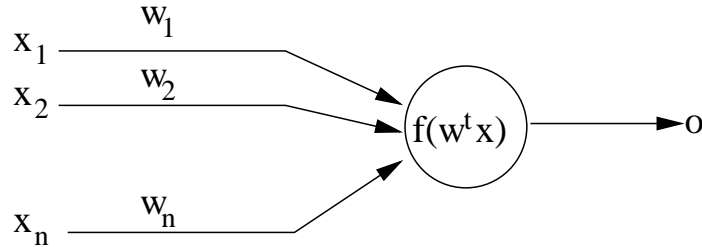


Figura 4.2: Model general de neuron

Se remarcă lipsa pragului T din ecuațiile de mai sus; de fapt, se poate considera că neuronul are doar $n - 1$ intrări sinaptice iar valoarea a n -a este $x_n = -1$ permanent, având ponderea asociată $w_n = T$. Funcțiile de activare $f(\cdot)$ larg folosite sunt

$$f_1(net) = \frac{2}{1 + e^{-\lambda \cdot net}} - 1, \lambda > 0 \quad (4.4)$$

cu graficul dat în figura 4.3 și

$$f_2(net) = \text{sgn}(net) = \begin{cases} +1, & \text{dacă } net > 0 \\ -1, & \text{dacă } net < 0 \end{cases} \quad (4.5)$$

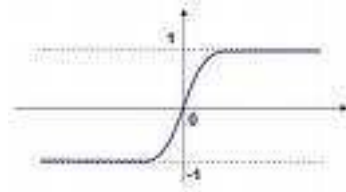


Figura 4.3: Funcția sigmoidă dată de ecuația 4.4 pentru λ fixat.

Se observă că $\lambda/4 = f'_1(0)$, deci λ este proporțională cu panta tangentei la graficul lui f_1 în origine. Pentru $\lambda \rightarrow \infty$ f_1 tinde către f_2 . Datorită alurii funcției f_1 , amintind de forma literei *S*, ea se mai numește și sigmoidă. Se mai poate folosi drept funcție de activare tangenta hiperbolică:

$$f_3(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\frac{\exp(x) - \exp(-x)}{2}}{\frac{\exp(x) + \exp(-x)}{2}} = \frac{\exp(2x) - 1}{\exp(2x) + 1} \quad (4.6)$$

Deoarece funcțiile date iau valori între -1 și +1 se mai numesc bipolare, iar aspectul (dis)continuu le dă denumirea de *bipolară continuă* și respectiv *bipolară discretă*. Funcțiile bipolare pot fi redusă la formă unipolară, mărginită de 0 și 1:

$$f_4(\text{net}) = \frac{1}{1 + e^{-\lambda \cdot \text{net}}}, \lambda > 0 \quad (4.7)$$

(sigmoida logistică) și

$$f_5(\text{net}) = \begin{cases} +1, & \text{dacă } \text{net} > 0 \\ 0, & \text{dacă } \text{net} < 0 \end{cases} \quad (4.8)$$

Drept funcție de activare se poate folosi de fapt orice funcție care este monoton nedescrescătoare, mărginită și preferabil derivabilă³. Aspectul continuu asigură mai multă flexibilitate procesului de instruire, motiv pentru care funcțiile de activare continue sunt cele folosite.

Ieșirile pot fi binare sau continue, bipolare sau unipolare. Pentru m neuroni, mulțimea valorilor de ieșire este:

- $(-1, 1)^m$ pentru funcție de activare bipolară continuă
- $(0, 1)^m$ pentru funcție de activare unipolară continuă
- $\{-1, 1\}^m$ pentru funcție de activare bipolară discretă
- $\{0, 1\}^m$ pentru funcție de activare unipolară discretă

³Pentru cazul funcțiilor nederivabile se poate folosi metoda subgradientului: http://www.stanford.edu/class/ee392o/subgrad_method.pdf.

4.4 Modele de rețea neurală artificială

4.4.1 Rețea cu propagare înainte

Vom considera o arhitectură de rețea cu propagare înainte⁴ cu n intrări și m neuroni de ieșire, precum în figura 4.4. Intrările și ieșirile sunt respectiv:

$$\begin{aligned}\mathbf{x} &= (x_1, x_2, \dots, x_n)^t \\ \mathbf{o} &= (o_1, o_2, \dots, o_m)^t\end{aligned}\tag{4.9}$$

Dacă considerăm vectorul de ponderi \mathbf{w}_i care leagă neuronul de ieșire i cu

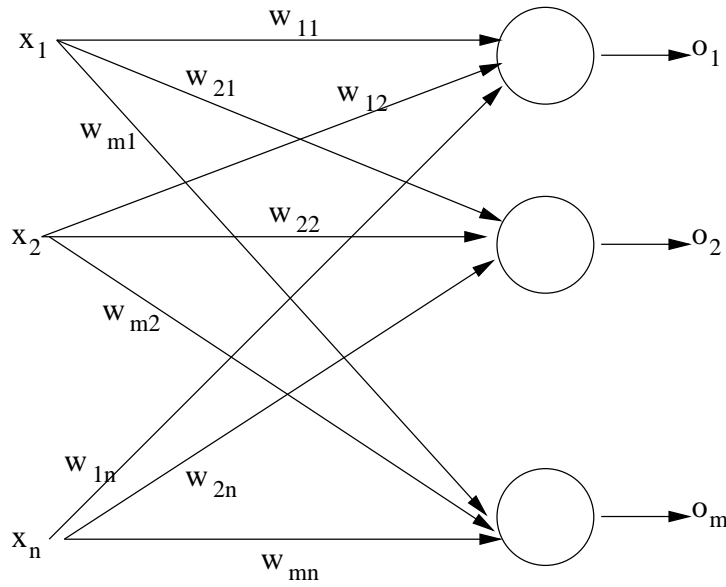


Figura 4.4: Model de rețea neurală artificială

toate intrările, $\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{in})^t$, atunci valoarea de activare pentru neuronul i este

$$net_i = \mathbf{w}_i^t \mathbf{x} = \sum_{j=1}^n w_{ij} x_j, \forall i = \overline{1, m}\tag{4.10}$$

Valoarea de ieșire o_i pentru fiecare neuron este $o_i = f(net_i) = f(\mathbf{w}_i^t \mathbf{x})$.

Putem nota cu \mathbf{W} matricea ponderilor dintre neuroni:

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix}\tag{4.11}$$

⁴În original: feedforward network.

și introducem operatorul matricial $\Gamma(\cdot)$ definit ca

$$\Gamma \left(\begin{pmatrix} net_1 \\ net_2 \\ \vdots \\ net_m \end{pmatrix} \right) = \begin{pmatrix} f(net_1) \\ f(net_2) \\ \vdots \\ f(net_m) \end{pmatrix} \quad (4.12)$$

ceea ce ne permite să scriem ieșirea rețelei ca fiind:

$$\mathbf{o} = \Gamma(\mathbf{W} \cdot \mathbf{x}) \quad (4.13)$$

Valorile de intrare \mathbf{x} și cele de ieșire \mathbf{o} se numesc pattern-uri de intrare și respectiv de ieșire. Rețeaua acționează instantaneu, adică de îndată ce intrarea este furnizată, rețeaua dă și valoarea de ieșire asociată. Dacă se consideră momentul de timp t , atunci putem rescrie 4.13 ca:

$$\mathbf{o}(t) = \Gamma(\mathbf{W} \cdot \mathbf{x}(t)) \quad (4.14)$$

4.4.2 Rețele cu conexiune inversă

La rețeaua prezentată anterior putem adăuga niște conexiuni care să facă legătura de la ieșiri la intrări. Rețeaua nou obținută este denumită “cu conexiune inversă”⁵; o reprezentare este dată în figura 4.5. În felul acesta, ieșirile controlează intrările. Mai mult, valorile $\mathbf{o}(t)$ controlează valorile $\mathbf{o}(t + \Delta)$. Δ reprezintă aici perioada refractară a neuronului. Ieșirea este dată de ecuația:

$$\mathbf{o}(t + \Delta) = \Gamma(\mathbf{W}\mathbf{o}(t)) \quad (4.15)$$

Intrarea \mathbf{x} este necesară doar la început ($t = 0$), după care sistemul se auto-întreține. Dacă considerăm timpul ca valoare discretă și urmărim sistemul la momentele $0, \Delta, 2\Delta, \dots, k\Delta, \dots$ atunci sistemul se numește discret. Putem lua convenabil $\Delta = 1$ și atunci avem:

$$\mathbf{o}^{k+1} = \Gamma(\mathbf{W}\mathbf{o}^k), k = 1, 2, \dots \quad (4.16)$$

sau

$$\mathbf{o}^{k+1} = \Gamma \left(\mathbf{W}\Gamma \left(\dots \Gamma(\mathbf{W}\mathbf{x}^0) \dots \right) \right) \quad (4.17)$$

Șirul de valori $\mathbf{o}^1, \mathbf{o}^2, \dots$ reprezintă stările succesive ale rețelei, care în acest caz este văzut ca un sistem dinamic. Este posibil ca de la un moment dat $\mathbf{o}^k = \mathbf{o}^{k+1} = \dots$, adică \mathbf{o}^k să fie un atractor, iar rețeaua se stabilizează. Mai general, un atractor poate să fie o mulțime finită de valori. Un exemplu de astfel de rețea neurală este memoria asociativă bidirecțională.

⁵În limba engleză: feedback network.

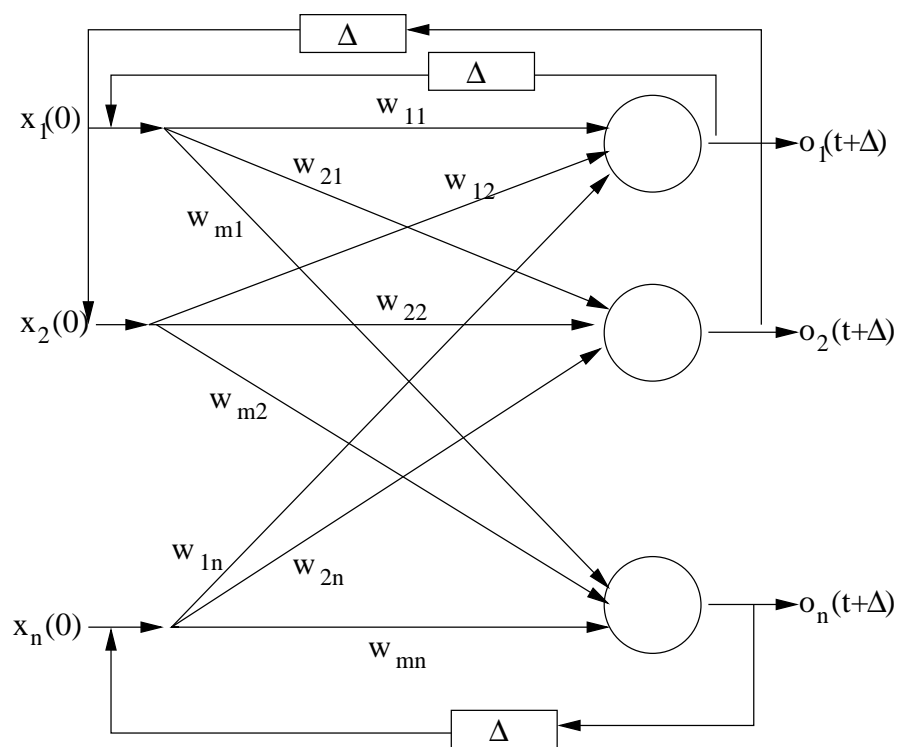


Figura 4.5: Rețea cu conexiune inversă.

4.5 Învățarea ca problemă de aproximare

În urma procesului de învățare nu putem obține în toate cazurile reproducerea perfectă a ceea ce s-a învățat. Se poate obține o aproximare a unei funcții $h(\cdot)$ printr-o funcție $H(\mathbf{w}, \cdot)$ unde $\mathbf{w} = (w_1, w_2, \dots, w_m)^t$ iar argumentul notat “.” este un vector $\mathbf{x} = (x_1, x_2, \dots, x_n)^t$. Problema este de a găsi vectorul \mathbf{w} care dă cea mai bună aproximare pentru un set de antrenare $\{(\mathbf{x}^1, h(\mathbf{x}^1)), \dots, (\mathbf{x}^p, h(\mathbf{x}^p))\}$. Primul pas este alegerea funcției aproximante $H(\cdot, \cdot)$, apoi un proces de învățare este folosit pentru a determina o valoare bună pentru vectorul \mathbf{w} . Altfel zis, se caută un vector \mathbf{w}^* pentru care

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{k=1}^p \rho \left(H(\mathbf{w}, \mathbf{x}^k), h(\mathbf{x}^k) \right) \quad (4.18)$$

unde ρ este o funcție distanță care măsoară calitatea aproximării. Învățarea este procesul de găsire a unui bun aproximant.

Deși formularea este simplă, există două dificultăți în rezolvarea generală a acestei probleme de aproximare:

1. o valoare potrivită pentru m poate fi greu de determinat; atunci când aproximarea se face prin rețele neurale de tip feedforward cu un strat ascuns (vezi curs 6), valoarea lui m este numărul de neuroni din stratul ascuns;
2. determinarea efectivă a lui \mathbf{w}^* cunoaște rezolvări pentru câteva cazuri – a se vedea teoria cercetărilor operaționale – dar o metodă eficientă pentru toate cazurile **nu este cunoscută**⁶; putem vorbi de probleme de programare liniară sau de programare pătratică, dar sunt multe alte situații care nu au o rezolvare teoretică cunoscută. O abordare este folosirea unor metode de căutare euristica – metodele “hill-climbing”, “simulated annealing”, algoritmi genetici, dar aceștia nu garantează obținerea minimului absolut.

4.6 Reguli de învățare

În această secțiune vom privi neuronul artificial ca pe o entitate adaptivă, pentru care ponderile se pot modifica pe baza unui proces de învățare. Ponderile se modifică în funcție de:

- valoarea actuală a ponderilor, \mathbf{w} sau \mathbf{W} ;
- semnalul de intrare \mathbf{x} ;
- ieșirea rezultată \mathbf{o}

⁶A se vedea http://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization.

- (opțional) ieșirea furnizată de un “profesor”, în cazul învățării supervizate, numită și ieșirea dorită, \mathbf{d}

Putem presupune că intrarea x_n are valoarea fixă -1 , pentru a permite includerea parametrului prag. Putem considera că sunt n neuroni de intrare și m de ieșire, iar vectorul ponderilor care leagă al i -lea neuron de ieșire de toți neuronii de intrare este $\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{in})^t$.

Regula de învățare generală este că ponderile \mathbf{w}_i variază proporțional cu produsul dintre intrarea \mathbf{x} și semnalul de învățare r . r este o funcție care ia în considerare trei din valorile date mai sus, deci

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i) \quad (4.19)$$

unde d_i este eventuala valoare de ieșire ce corespunde neuronului de ieșire i . Mai precis, avem

$$\Delta \mathbf{w}_i(t) = c \cdot r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)) \cdot \mathbf{x}(t) \quad (4.20)$$

unde $c > 0$ este rata de învățare. Expresia (4.20) dă modul în care se modifică ponderile de la un moment de timp la următorul:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + c \cdot r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)) \cdot \mathbf{x}(t) \quad (4.21)$$

Pentru cazul discret se folosește scrierea:

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + c \cdot r(\mathbf{w}_i^k, \mathbf{x}^k, d_i^k) \cdot \mathbf{x}^k, k = 0, 1, \dots \quad (4.22)$$

iar pentru cazul continuu se scrie ecuația diferențială

$$\frac{d\mathbf{w}_i(t)}{dt} = cr(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t))\mathbf{x}(t) \quad (4.23)$$

Pe baza formei funcției r avem variantele de învățare menționate mai jos.

4.6.1 Regula de învățare Hebbiană

Este o regulă de învățare nesupervizată formulată de Donald Hebb în 1949 astfel:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Matematic, se scrie:

$$r = f(\mathbf{w}_i^t \mathbf{x}) \quad (4.24)$$

deci modificarea ponderilor devine

$$\Delta \mathbf{w}_i = cf(\mathbf{w}_i^t \mathbf{x}) \mathbf{x} \quad (4.25)$$

ceea ce pe componente se scrie

$$\Delta w_{ij} = cf(\mathbf{w}_i^t \mathbf{x}) x_j \quad (4.26)$$

sau în funcție de ieșirea neuronului i

$$\Delta w_{ij} = co_i x_j \quad (4.27)$$

Ponderile sunt inițializate cu valori aleatoare mici, în jurul lui 0. Conform formulelor date, putem avea o creștere a ponderii w_{ij} dacă produsul $o_i x_j$ este pozitiv și o scădere în caz contrar. Se arată ușor că intrările prezentate frecvent au o influență mai mare asupra ponderilor și vor produce o valoare de ieșire mare.

Regula trebuie înțeleasă ca un principiu, existând la ora actuală variațiuni pe această temă.

Exemplu numeric: [16] pag 61.

4.6.2 Regula de învățare a perceptronului

Este folosită pentru învățare supervizată; regula a fost formulată de către Rosenblatt în 1958. Semnalul de învățare este diferența între valoarea dorită și cea obținută:

$$r = d_i - o_i \quad (4.28)$$

unde $o_i = \text{sgn}(\mathbf{w}_i^t \mathbf{x})$, iar d_i este răspunsul dorit, furnizat de “profesor”. Modificarea ponderilor este deci

$$\Delta \mathbf{w}_i = c \left(d_i - \text{sgn}(\mathbf{w}_i^t \mathbf{x}) \right) \mathbf{x} \quad (4.29)$$

Formula se aplică pentru cazul în care ieșirile sunt bipolare binare. Modificarea în ponderi apare doar dacă ieșirea furnizată de neuronul de ieșire i diferă de valoarea dorită — cunoscută aprioric. Explicitând, se poate vedea că modificarea de pondere în cazul neconcordanței ieșirii cu valoarea dorită este

$$\Delta \mathbf{w}_i = \pm 2c \mathbf{x} \quad (4.30)$$

Ponderile pot fi inițializate cu orice valoare.

Exemplu: [16] pag 65.

4.6.3 Regula de învățare delta

Prezenta regulă se folosește pentru cazul învățării supervizate cu funcție de activare continuă; a fost introdusă de McClelland și Rumelhart în 1986. Semnalul de învățare se cheamă în acest context “delta” și are forma:

$$r = (d_i - f(\mathbf{w}_i^t \mathbf{x}))f'(\mathbf{w}_i^t \mathbf{x}) \quad (4.31)$$

Motivația prezenței derivatei în formulă este dată de minimizarea erorii pătratice:

$$E = \frac{1}{2}(d_i - o_i)^2 = \frac{1}{2} \left(d_i - f(\mathbf{w}_i^t \mathbf{x}) \right)^2 \quad (4.32)$$

Tehnica de reducere a valorii funcției constă în mișcarea în sens opus gradientului ∇E

$$\nabla E = -(d_i - o_i)f'(\mathbf{w}_i^t \mathbf{x})\mathbf{x} \quad (4.33)$$

Componentele gradientului sunt

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i)f'(\mathbf{w}_i^t \mathbf{x})x_j, \forall j = \overline{1, n} \quad (4.34)$$

Modificarea ponderilor se face astfel:

$$\Delta \mathbf{w}_i = -\eta \nabla E \quad (4.35)$$

unde η este o constantă pozitivă. O altă scriere este:

$$\Delta \mathbf{w}_i = \eta(d_i - o_i)f'(net_i)\mathbf{x} \quad (4.36)$$

Regula poate fi generalizată pentru rețele cu mai multe straturi.

Exemplu: [16] pag 68.

4.6.4 Regula de învățare Widrow-Hoff

A fost enunțată în 1962 și se aplică pentru învățarea supervizată. Regula folosește ca funcție de activare funcția identică $f(x) = x$ și minimizează pătratul diferenței dintre ieșirea dorită d_i și activarea net_i :

$$r = d_i - \mathbf{w}_i^t \mathbf{x} \quad (4.37)$$

deci ajustarea de ponderi se face cu

$$\Delta \mathbf{w}_i = c(d_i - \mathbf{w}_i^t \mathbf{x})\mathbf{x} \quad (4.38)$$

Regula Widrow-Hoff este o formă particulară a regulii delta și mai este cunoscută sub numele de regula celor mai mici pătrate⁷. Ponderile sunt inițializate cu orice valori.

⁷Least Mean Square (LMS).

4.6.5 Regula de învățare prin corelație

Se obține prin considerarea lui $r = d_i$. Ponderile se modifică cu valoarea

$$\Delta \mathbf{w}_i = c d_i \mathbf{x} \quad (4.39)$$

Ponderile sunt inițializate cu zero.

4.6.6 Regula “câștigătorul ia tot”

Regula “winner-takes-all” este un exemplu de învățare competitivă și e folosită pentru învățarea în mod nesupervizat a proprietăților statistice ale datelor. Învățarea se bazează pe premisa că din toți neuronii de ieșire unul (fie el de indice k) dă răspunsul maxim pentru o intrare \mathbf{x} . Ponderea aferentă acestui vector va fi modificată astfel:

$$\Delta \mathbf{w}_i = \alpha (\mathbf{x} - \mathbf{w}_i) \quad (4.40)$$

unde $\alpha > 0$ este o valoare mică, de regulă descrescătoare pe măsură ce procesul de învățare continuă. Indicele i este ales deci

$$i = \arg \max_k (\mathbf{w}_k^t \mathbf{x}) \quad (4.41)$$

După ajustare, ponderile tind să estimeze mai bine patternul de intrare. Ponderile sunt inițializate cu valori aleatoare și lungimile lor sunt apoi normalizate: $\|\mathbf{w}_i\| = \text{const}, \forall i$.

Capitolul 5

Perceptroni monostrat

[6], cap. 3

Capitolul 6

Perceptroni multistrat

Rețelele neurale multistrat — sau perceptronii multistrat, multilayer perceptrons (MPLs) — sunt folosite pentru probleme de regresie, de clasificare și de estimare de probabilități condiționate. Instruirea este supervizată. Sunt cea mai populară variantă de rețele neurale artificiale.

6.1 Motivație pentru rețele neurale multistrat

Conform celor din cursul precedent, un perceptron liniar este capabil să găsească un hiperplan de separare pentru două mulțimi, dacă ele sunt liniar separabile. Există însă exemple de probleme simple care nu sunt liniar separabile — și deci nerezolvabile de către perceptron liniar — dar care pot fi totuși separate. În plus, dorim să rezolvăm și altfel de probleme decât de clasificare binară: regresie (estimare de valoare de ieșire de tip continuu), estimare de probabilitate condiționată, clasificare între mai mult de două clase. Cursul de față conține modele bazate pe neuroni neliniari în care se pot rezolva toate aceste tipuri de probleme.

Figura 6.1 conține cea mai celebră problemă care nu se poate rezolva de către un clasificator liniar: problema XOR. Se consideră setul

$$\mathcal{S} = \{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 0)\}$$

unde fiecare din cele patru elemente este compus din pereche de intrare $(x, y) \in \{0, 1\}^2$ și dintr-o etichetă de clasă binară. Se poate demonstra algebric că într-adevăr nu se pot separa cele două clase printr-o singură dreaptă. Un alt exemplu este dat în figura 6.2 [5].

Intuim că un singur neuron e prea puțin pentru probleme complexe de separare. Pe de altă parte, concatenarea mai multor neuroni cu funcție prag sau chiar cu funcție de activare liniară nu ne duce prea departe, deoarece compunerea repetată a unor asemenea operații de același tip nu schimbă natura rezultatului. De exemplu, pentru funcție de activare liniară: aplicarea unei succesiuni de neuroni liniari pentru o intrare vector este echivalentă cu

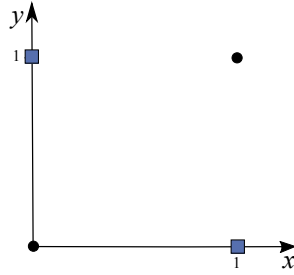


Figura 6.1: Problema XOR. Clasele sunt marcate cu forme și culori diferite. Se poate demonstra că nu se poate trasa o dreaptă în plan care să aibă de o parte a ei doar puncte din clasa “0” și de cealaltă parte doar puncte de clasă “1”.

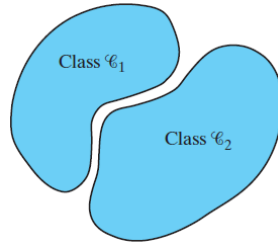


Figura 6.2: Două clase de puncte ce nu sunt separabile liniar [5].

o înmulțire a unei secvențe de matrice cu vectorul de intrare. Datorită distributivității înmulțirii matriceale, operația este echivalentă cu înmulțirea vectorului de intrare cu o matrice rezultată din produsul celorlalte matrice. Obținem de aici că succesiunea de neuroni cu funcție de activare liniară este echivalentă cu un singur neuron.

Vom folosi deci mai mulți neuroni, iar funcțiile lor de activare vor fi neliniare.

6.2 Sumar de notații

Tabelul 6.1 conține notațiile principale care se folosesc în acest curs.

6.3 Setul de instruire

Rețelele din acest capitol sunt pentru instruire de tip supervizat. Setul de instruire este de forma:

$$\mathcal{S} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{d}^{(1)} \right), \left(\mathbf{x}^{(2)}, \mathbf{d}^{(2)} \right), \dots, \left(\mathbf{x}^{(p)}, \mathbf{d}^{(p)} \right) \right\} \quad (6.1)$$

unde $\mathbf{x}^{(i)} \in \mathbb{R}^n$ iar $\mathbf{d}^{(i)}$ este, după caz:

Notăție	Explicație
p	număr de perechi în setul de instruire
$\mathbf{x}^{(i)}$	vector de intrare din setul de instruire, $\mathbf{x}^{(i)} = (\mathbf{x}_1^{(i)}, \dots, \mathbf{x}_n^{(i)})^t, 1 \leq i \leq p$
$\mathbf{d}^{(i)}$	ieșirea asociată intrării $\mathbf{x}^{(i)}$, din setul de instruire, $\mathbf{d}^{(i)} = (\mathbf{d}_1^{(i)}, \dots, \mathbf{d}_m^{(i)})^t, 1 \leq i \leq p$
L	numărul de straturi din rețeaua neurală, inclusiv cele de intrare și de ieșire
nod	neuron – dacă apare pe stratul $2, 3, \dots, L$ – sau nod de intrare – dacă apare pe primul strat
n_l	numărul de noduri din stratul $l, 1 \leq l \leq L$;
$z_i^{(l)}$	valoarea de activare a neuronului i din stratul l , pentru $2 \leq l \leq L, 1 \leq i \leq n_l$
$\mathbf{z}^{(l)}$	vectorul conținând valorile de activare ale neuronilor din stratul $l, \mathbf{z}^{(l)} = (\mathbf{z}_1^{(l)}, \dots, \mathbf{z}_{n_l}^{(l)})^t, 2 \leq l \leq L$
$a_i^{(l)}$	valoarea de ieșire a celui de al i -lea nod din stratul l , pentru $1 \leq l \leq L, 1 \leq i \leq n_l$
$\mathbf{a}^{(l)}$	vectorul cu valorile de ieșire ale nodurilor din stratul l , pentru $1 \leq l \leq L, \mathbf{a}^{(l)} = (\mathbf{a}_1^{(l)}, \dots, \mathbf{a}_{n_l}^{(l)})^t$
$w_{ij}^{(l)}$	ponderea legăturii între neuronul i de pe stratul $l + 1$ și nodul j de pe stratul anterior $l, 1 \leq l \leq L - 1$, $1 \leq i \leq n_{l+1}, 1 \leq j \leq n_l$
$\mathbf{W}^{(l)}$	matricea de ponderi dintre stratul l și stratul $l + 1$, $1 \leq l \leq L - 1, \mathbf{W}_{ij}^{(l)} = w_{ij}^{(l)}, 1 \leq i \leq n_{l+1}, 1 \leq j \leq n_l$
$\mathbf{W}_i^{(l)}$	linia i a matricei $\mathbf{W}^{(l)}, 1 \leq l \leq L - 1, 1 \leq i \leq n_{l+1}$
$b_i^{(l)}$	ponderea de bias pentru neuronul i din stratul $l + 1$, $1 \leq l \leq L - 1, 1 \leq i \leq n_{l+1}$
$\mathbf{b}^{(l)}$	vectorul ponderilor de bias de la stratul l la $l + 1$, $\mathbf{b}^{(l)} = (b_1^{(l)}, \dots, b_{n_{l+1}}^{(l)})^t, 1 \leq l \leq L - 1$
f	funcție de activare a neuronului
\mathbf{W}	secvența de matrice de ponderi $(\mathbf{W}^1, \mathbf{W}^2, \dots, \mathbf{W}^{L-1})$
\mathbf{b}	secvența de vectori de ponderi de bias $(\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^{L-1})$
$J(\mathbf{W}, \mathbf{b})$	eroare empirică pentru setul de instruire sau minibatch
$J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$	eroarea pentru perechea de vectori de instruire $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$, $1 \leq i \leq p$
$\mathbf{o}^{(i)}$	vector coloană de ieșire corespunzător intrării $\mathbf{x}^{(i)}$, calculat de rețea
δ^l	semnalul de eroare pentru stratul $l, 2 \leq l \leq L$
\odot	produs Hadamard

Tabela 6.1: Notății folosite

- pentru o problemă de regresie: vector oarecare din \mathbb{R}^m ;
- pentru o problemă de clasificare sau estimare de probabilități pentru m clase: vectori de forma $(1, 0, \dots, 0)$, $(0, 1, 0, \dots, 0)$, \dots , $(0, 0, \dots, 0, 1)$ cu m valori binare, din care doar cea de pe poziția aferentă clasei curente este unu iar restul sunt zero¹.

6.4 Rețeaua neurală multistrat

6.4.1 Arhitectură

Există mai multe modalități de dispunere a neuronilor; noi vom folosi o arhitectură de tip multistrat, feedforward, numită *perceptron* multistrat – chiar dacă neuronii folosiți nu sunt perceptroni, ci neuroni cu funcție de activare continuă. O rețea multistrat se compune din minim trei straturi:

- strat de intrare ce preia valorile de intrare; nu are rol computațional, nu este format din neuroni²;
- măcar un strat ascuns, compus din neuroni;
- strat de ieșire, de asemenea compus din neuroni, produce estimări de valori care sunt apoi comparate cu ieșirile dorite.

Un strat ascuns este unul care nu primește direct intrări și nu produce valori de ieșire. Neuronii ascunși produc trăsături noi pe baza vectorilor de intrare, trăsături care sunt mai apoi necesare rețelei neurale pentru producerea unei estimări. Este posibil ca o rețea să aibă mai mult de un neuron în stratul de ieșire, așa cum se vede în figura 6.4.

Se consideră că instruirea e mai eficientă dacă pe lângă valorile de intrare (sau pe lângă valorile calculate de un strat de neuroni) se mai furnizează o valoare constantă, de regulă $+1$, înmulțită cu o pondere de *bias*³. Ponderile dintre straturi precum și aceste ponderi de *bias* sunt instruibile, adică se vor modifica prin procesul de învățare⁴.

O reprezentare de rețea neurală cu trei straturi și o ieșire este dată în figura 6.3; o rețea cu 4 straturi și două ieșiri este reprezentată în figura 6.4. Nu există o relație anume între numărul de straturi ascunse și numărul de neuroni de ieșire. Pentru oricare dintre figuri:

¹Așa-numita codificare *one-hot* sau *1-din-m*.

²Motiv pentru care unii autori nu îl consideră un strat propriu-zis; se preferă o referire la o rețea anume ca având “ k ” straturi ascunse, cele de intrare și ieșire fiind oricum existente.

³Unii autori consideră valoarea constantă -1 ; nu este esențial, deoarece ponderile sunt instruibile și își pot modifica valoarea de la pozitiv la negativ și invers.

⁴O discuție asupra necesității considerării *bias*-ului este la ftp://ftp.sas.com/pub/neural/FAQ2.html#A_bias

- valorile x_1, x_2, x_3 sunt componentele vectorilor de intrare; vectorul provine din datele de intrare originare x_1 și x_2 la care se mai adaugă componenta x_3 cu valoarea constantă $+1$, aferentă *bias*-ului;
- valoarea $a_i^{(l)}$ este ieșirea nodului i din stratul l , $1 \leq l \leq L$, $1 \leq i \leq n_l$; se remarcă în figuri prezența valorilor constante $+1$ în toate straturile, exceptând cel de ieșire. Pentru stratul de intrare, $a_i^{(1)} = x_i$, unde $\mathbf{x} = (x_1, \dots, x_n)$ e vectorul de intrare;
- valoarea $h_{W,b}(x)$ este ieșirea calculată de către rețea; $h_{W,b}(x) \in \mathbb{R}$ pentru figura 6.3 și $h_{W,b}(x) \in \mathbb{R}^2$ pentru figura 6.4.

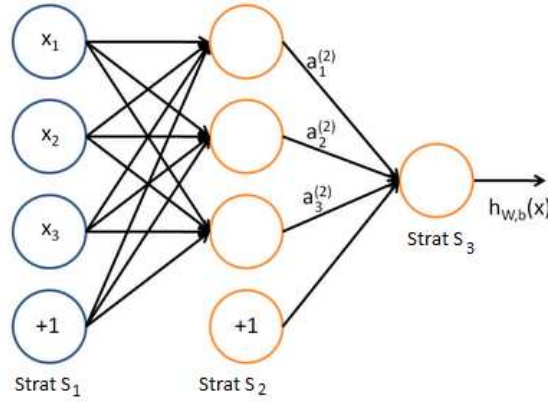


Figura 6.3: Rețea neurală cu propagare înainte cu 3 straturi

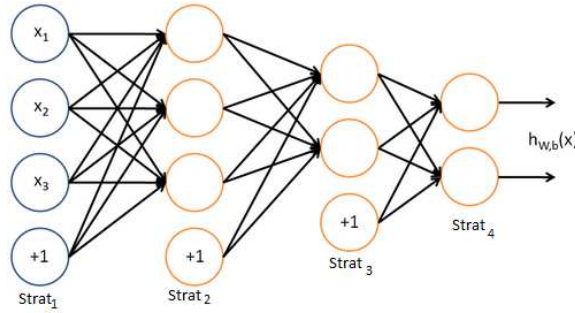


Figura 6.4: Rețea neurală cu propagare înainte cu 4 straturi

Vom considera că avem $L \geq 3$ straturi și în fiecare strat l ($2 \leq l \leq L$) un număr de n_l neuroni. Stratul de intrare are $n_1 = n$ noduri, numărul de neuroni din stratul de ieșire este $n_L = m$ dat de: numărul de clase pentru care se face recunoașterea (la problemă de clasificare sau estimare de probabilitate condiționată) sau numărul de ieșiri care se doresc a fi approximate (la regresie).

Perechea (\mathbf{W}, \mathbf{b}) formează mulțimea ponderilor interneurale și a valorilor de bias din în rețea. Avem următoarele notații:

- ponderile dintre stratul de intrare și stratul ascuns sunt conținute în matricea $\mathbf{W}^{(1)}$: $w_{ij}^{(1)}$ este ponderea legăturii dintre neuronul i al stratului al doilea și neuronul j din stratul de intrare; se remarcă ordinea indicilor, utilă mai departe pentru operațiile de algebră liniară ce vor fi folosite;
- mai general, notăm cu $W_{ij}^{(l)}$ ponderea legăturii dintre al i -lea neuron din stratul $l+1$ și al j -lea neuron din stratul precedent l ($1 \leq l \leq L-1$);
- valoarea ponderii dintre intrarea constantă $+1$ din stratul de intrare și neuronul i din primul strat ascuns este stocată de $b_i^{(1)}$, $1 \leq i \leq n_2$;
- în general, coeficientul de bias provenind din stratul l ($1 \leq l \leq L-1$) și care afectează neuronul i din stratul $l+1$ este notat cu $b_i^{(l)}$, $1 \leq i \leq n_{l+1}$.

În ce privește numărul de ponderi (atât cele din matricele $\mathbf{W}^{(l)}$ cât și ponderile de bias) avem, pentru $1 \leq l \leq L-1$:

- matricea $\mathbf{W}^{(l)}$ de ponderi dintre stratul l și stratul $l+1$ are n_{l+1} linii și n_l coloane;
- vectorul coloană de coeficienți bias $\mathbf{b}^{(l)}$ conține n_{l+1} valori, având forma $\mathbf{b}^{(l)} = (b_1^{(l)}, b_2^{(l)}, \dots, b_{n_{l+1}}^{(l)})^t$.

6.4.2 Funcții de activare

Fiecare neuron agregă valorile din nodurile de pe stratul anterior – incluzând și termenul constant $+1$ multiplicat cu coeficientul de bias. Neuronul de indice i de pe stratul $l > 1$ are valoarea de activare calculată ca:

$$z_i^{(l)} = W_{i1}^{(l-1)} \cdot a_1^{(l-1)} + W_{i2}^{(l-1)} \cdot a_2^{(l-1)} + \dots + W_{i, n_{l-1}}^{(l-1)} \cdot a_{n_{l-1}}^{(l-1)} + b_i^{(l-1)} \quad (6.2)$$

$$= \mathbf{W}_i^{(l-1)} \cdot \mathbf{a}^{(l-1)} + b_i^{(l-1)}, \quad 1 \leq i \leq n_l \quad (6.3)$$

unde: $\mathbf{W}_i^{(l-1)}$ este linia i a matricei $\mathbf{W}^{(l-1)}$, $a_i^{(l-1)}$ este valoarea de ieșire a neuronului i de pe stratul $l-1$ (dacă $l-1 > 1$) sau chiar intrarea x_i din vectorul de intrare curent (dacă $l-1 = 1$). Notând cu $\mathbf{z}^{(l)}$ vectorul $(z_1^{(l)}, z_2^{(l)}, \dots, z_{n_l}^{(l)})^t$ putem scrie matricial:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l-1)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l-1)}, \quad 2 \leq l \leq L \quad (6.4)$$

Pe baza valorii de activare $z_i^{(l)}$ a neuronului i de pe stratul l se calculează ieșirea neuronului:

$$a_i^{(l)} = f(z_i^{(l)}) \quad (6.5)$$

pentru $2 \leq l \leq L, 1 \leq i \leq n_l$. Dacă folosim notația $f((h_1, h_2, \dots, h_k)^t) = (f(h_1), f(h_2), \dots, f(h_k))^t$ (adică se aplică funcția f pe fiecare valoare din vectorul argument, sau mai pe scurt se vectorizează f) atunci putem scrie mai compact ecuația (6.5) sub forma:

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \quad (6.6)$$

Funcția $f(\cdot)$ este funcție de activare; ea este necesară în pasul de propagare înainte; pentru pasul de propagare înapoi a erorii este folosită derivata ei. Funcția de activare poate avea formele⁵:

1. funcția logistică sigmoidă:

$$f = \sigma : \mathbb{R} \rightarrow (0, 1), f(z) = \sigma(z) = \frac{1}{1 + \exp(-z)} \quad (6.7)$$

Valoarea derivatei acestei funcții este:

$$f'(z) = \sigma'(z) = \sigma(z)(1 - \sigma(z)) = f(z) \cdot (1 - f(z)) \quad (6.8)$$

2. funcția tangentă hiperbolică:

$$f = \tanh : \mathbb{R} \rightarrow (-1, 1), f(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \quad (6.9)$$

a cărei derivată este:

$$f'(z) = \tanh'(z) = 1 - \tanh^2(z) = 1 - f^2(z) \quad (6.10)$$

Se poate arăta că între cele două funcții de activare \tanh și σ există relația:

$$\tanh(z) = 2 \cdot \sigma(2z) - 1 \quad (6.11)$$

În practică funcția \tanh dă rezultate mai bune decât sigmoida logistică. O explicație teoretică se găsește în [23]; demonstrații empirice sunt în [24].

3. funcția liniară:

$$f(z) = a \cdot z + b \quad (6.12)$$

cu derivata $f'(z) = a$; frecvent se iau $a = 1, b = 0$. Este utilizată dacă se dorește ca la ieșire valorile să fie în afara intervalelor $(0, 1)$ și $(-1, 1)$, cum se întâmplă la funcțiile de activare de mai sus.

⁵Dar alte funcții de activare mai pot fi considerate, de exemplu combinații liniare de polinoame Hermite.

4. funcția softmax:

$$\text{softmax}(\mathbf{z}; c) = \frac{\exp(z_c)}{\sum_{i=1}^m \exp(z_i)} \quad (6.13)$$

unde c este indicele neuronului din stratul lui, iar m este numărul total de neuroni din același strat. A mai fost folosită la regresia logistică pentru cazul a mai mult de două clase.

Funcția softmax este utilă pentru a transforma din valori oarecare în distribuție de probabilitate: se arată ușor că $1 > \text{softmax}(\mathbf{z}; c) > 0 \forall c$ și valorile $\text{softmax}(\mathbf{z}; c)$ însumate dau 1. Valoarea $\text{softmax}(\mathbf{z}; c)$ se interpretează convenabil drept probabilitatea ca intrarea curentă să fie de clasă c ; clasificarea se face găsind acel indice $1 \leq c \leq m$ pentru care $\text{softmax}(\mathbf{z}; c)$ este maxim. Se utilizează în în stratul de ieșire al rețelei neurale de clasificare sau estimare de probabilitate.

Derivatele parțiale ale funcției softmax sunt:

$$\frac{\partial \text{softmax}(\mathbf{z}; i)}{\partial z_j} = \begin{cases} \text{softmax}(\mathbf{z}; i) \cdot (1 - \text{softmax}(\mathbf{z}; i)) & \text{dacă } i = j \\ -\text{softmax}(\mathbf{z}; i) \cdot \text{softmax}(\mathbf{z}; j) & \text{dacă } i \neq j \end{cases} \quad (6.14)$$

5. funcția Rectified Linear Unit (ReLU):

$$f(z) = \max(0, z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \quad (6.15)$$

Chiar dacă funcția este liniară pe porțiuni, ea este neliniară în ansamblu. Faptul că doar într-un punct nu e derivabilă nu deranjează în practică.

6. funcția Parametrized ReLU (PReLU):

$$f(z) = \begin{cases} \alpha z & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \quad (6.16)$$

unde $\alpha > 0$ [26], reprezentând o ușoară generalizare a funcției ReLU.

Este admis ca funcția de activare să difere de la strat la strat sau de la neuron la neuron.

6.5 Pasul de propagare înainte

Odată ce arhitectura rețelei e definită – numărul de straturi ascunse și numări de neuroni în fiecare strat plus funcțiile de activare – se poate trece

la instruirea și apoi utilizarea ei. Pasul de propagare înainte preia un vector de intrare $\mathbf{x} = (x_1, \dots, x_n)^t$ și produce modificări în neuronii straturilor succesive ale rețelei, ceea ce dă și numele tipului de rețea: “cu propagare înainte”, sau “feedforward”. Stratul de ieșire va furniza o valoare care va fi folosită ca predicție.

După cum s-a mai afirmat, stratul de intrare nu are rol computațional; valoare sa de ieșire este chiar vectorul de intrare furnizat rețelei:

$$\mathbf{a}^{(1)} = \mathbf{x} \quad (6.17)$$

Dacă se cunosc valorile de ieșire ale nodurilor din stratul l se pot calcula valorile de activare ale neuronilor din stratul $l + 1$ și apoi valorile lor de ieșire, astfel:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l-1)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l-1)} \quad (6.18)$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \quad (6.19)$$

pentru $l = 1, 2, \dots, L - 1$, cu $f(\cdot)$ funcție de activare ce se aplică pe fiecare componentă a vectorului (vectorizare). Vom nota cu \mathbf{o} vectorul de m valori de ieșire produs de către rețea:

$$\mathbf{o} = \mathbf{a}^{(L)} \quad (6.20)$$

Se recomandă ca operațiile date mai sus să fie implementate folosind biblioteci optimizate de algebră liniară, ce permit înmulțirea eficientă de matrice și calcul vectorizat — Octave, Matlab, Numpy, ND4J + Canova, Theano, Tensorflow etc.

6.6 Pasul de propagare înapoi a erorii

La etapa de instruire se urmărește ca ponderile — atât cele din matricele $\mathbf{W}^{(l)}$ cât și valorile bias $\mathbf{b}^{(l)}$ — să fie modificate de așa manieră încât valoarea de ieșire furnizată de rețea pentru vectorul de intrare $\mathbf{x}^{(i)}$, și anume $\mathbf{o}^{(i)}$, să fie cât mai apropiată de ieșirea corespunzătoare $\mathbf{d}^{(i)}$; perechea $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ face parte din setul de instruire \mathcal{S} . Pentru a măsura cât de apropiate sunt valorile de ieșire față de cele dorite se folosesc funcții de cost (sau de eroare; engl: *loss functions*, *error functions*).

Valorile inițiale ale ponderilor \mathbf{W} și \mathbf{b} sunt setate aleator, în jurul lui zero. Este necesar ca valorile ponderilor să nu fie toate egale, deoarece în acest caz fiecare neuron ar avea exact aceași valoare de intrare: fiecare neuron dintr-un strat e legat la exact aceleași intrări ca și ceilalți din stratul său; mai departe, dacă ponderile cu care se înmulțesc intrările sunt egale, atunci valoarea de activare a fiecărui neuron de pe acel strat e aceeași (ponderea constantă folosită se dă factor comun); argumentul se verifică începând cu

propagarea de la stratul de intrare. Am avea deci neuroni care calculează exact aceleași valori, ceea ce e inutil.

Strategii mai rafinate de inițializare sunt cele propuse de Glorot *et al.* [25] și He *et al.* [26]. Pentru arhitecturile de tip deep learning se preferă o preantrenare nesupervizată a ponderilor [23].

6.6.1 Funcții de cost

Fiecare pereche din \mathcal{S} , $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ ($1 \leq i \leq p$) va produce valoare de eroare astfel: se furnizează vectorul $\mathbf{x}^{(i)}$ ca intrare în rețea și se produce un vector de ieșire $\mathbf{o}^{(i)}$, reprezentând estimarea produsă de rețea pentru intrarea furnizată; se folosește o funcție de cost – sau de eroare – $J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ care se dorește a fi cu atât mai mică cu cât vectorul $\mathbf{o}^{(i)}$ e mai apropiat de $\mathbf{d}^{(i)}$. În plus, se mai consideră un factor de regularizare care împiedică ponderile să devină prea mari în valoare absolută, caz asociat de regulă cu un comportament instabil al rețelei: variații mici ale intrării duc la salturi mari îstraturile ascunse sau la ieșire.

Forma generală a funcției de eroare este:

$$J(\mathbf{W}, \mathbf{b}) = \underbrace{\left[\frac{1}{p} \sum_{i=1}^p \overbrace{J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})}^{\text{eroare empirică pentru } (\mathbf{x}^{(i)}, \mathbf{d}^{(i)})} \right]}_{\text{Eroarea empirică pe tot setul de antrenare}} + \underbrace{\frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{n_l} \sum_{j=1}^{n_{l+1}} (W_{ji}^{(l)})^2}_{\text{Factor de regularizare}} \quad (6.21)$$

unde $\lambda > 0$ este coeficientul de regularizare. Ultimul termen este regularizare L_2 , o sumă de pătrate de norme Frobenius peste matricele $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L-1)}$:

$$\|\mathbf{W}^{(l)}\|_F^2 = \sum_{i=1}^{n_l} \sum_{j=1}^{n_{l+1}} |w_{ji}^{(l)}|^2 \quad (6.22)$$

De regulă, bibliotecile pentru algebra liniară includ implementări eficiente pentru calculul normei Frobenius. O subliniere importantă este că ponderile de bias nu sunt supuse regularizării.

În cazul unei probleme de regresie, cea mai utilizată funcție de eroare $J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ ce măsoară calitatea unei predicții pentru perechea $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ este eroarea L_2 pătratică:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = \frac{1}{2} \cdot \|\mathbf{d}^{(i)} - \mathbf{o}^{(i)}\|^2 \quad (6.23)$$

unde $\|\mathbf{v}\|$ este norma L_2 a vectorului $\mathbf{v} = (v_1, \dots, v_m)^t$:

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^m v_i^2}$$

În acest caz funcția de eroare pentru tot setul de instruire devine:

$$J(\mathbf{W}, \mathbf{b}) = \left[\frac{1}{2p} \sum_{i=1}^p \|\mathbf{d}^{(i)} - \mathbf{o}^{(i)}\|^2 \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \|\mathbf{W}^{(l)}\|_F^2 \quad (6.24)$$

Pentru probleme de clasificare se preferă utilizare funcției de eroare cross-entropy iar în stratul de ieșire funcția de activare să fie softmax:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = - \sum_{j=1}^m \left[d_j^{(i)} \ln o_j^{(i)} + (1 - d_j^{(i)}) \ln (1 - o_j^{(i)}) \right] \quad (6.25)$$

unde pentru vectorul $\mathbf{d}^{(i)} = (d_j^{(1)}, d_j^{(2)}, \dots, d_j^{(m)})$ se folosește codificarea one-hot. În acest fel, funcția totală de eroare $J(\mathbf{W}, \mathbf{b})$ calculată pentru setul de instruire devine:

$$J(\mathbf{W}, \mathbf{b}) = \left\{ -\frac{1}{p} \sum_{i=1}^p \sum_{j=1}^m \left[d_j^{(i)} \ln o_j^{(i)} + (1 - d_j^{(i)}) \ln (1 - o_j^{(i)}) \right] \right\} + \frac{\lambda}{2} \sum_{l=1}^{L-1} \|\mathbf{W}^{(l)}\|_F^2 \quad (6.26)$$

6.6.2 Algoritmul backpropagation

Se dorește modificarea atât a ponderilor din matricele $\mathbf{W}^{(l)}$ cât și a coeficienților de bias $\mathbf{b}^{(l)}$ astfel încât valoarea funcției de eroare $J(\mathbf{W}, \mathbf{b})$ să scadă. Se va folosi metoda gradient descent, în care modificarea unei ponderi $w_{ij}^{(l)}$ se efectuează astfel:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial J}{\partial w_{ij}^{(l)}}(\mathbf{W}, \mathbf{b}) \quad (6.27)$$

Ponderile de bias $b_i^{(l)}$ se modifică similar:

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b_i^{(l)}}(\mathbf{W}, \mathbf{b}) \quad (6.28)$$

unde pentru ambele ecuații de mai sus $\alpha > 0$ este rata de învățare.

Există trei variante de lucru pentru modificarea ponderilor:

1. **varianta stochastic gradient descent:** pentru fiecare pereche din setul de instruire $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ se calculează valoarea de eroare $J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ și se fac modificările pentru ponderile $w_{ij}^{(l)}$ și $b_i^{(l)}$; următoarea pereche de instruire folosește valorile de ponderi modificate la acest pas;
2. **varianta off-line:** se agregă modificările care trebuie efectuate pentru fiecare pereche în parte și de abia la final se face modificarea ponderilor cu valorile agregate. Mai este numită și *batch learning*;

3. **varianta minibatch:** se procedează similar cu varianta batch, dar se face agregarea peste un subset al setului de instruire; este o versiune intermediară între instruirea incrementală și cea off-line; în practică este cea mai folosită strategie.

Vom prezenta varianta de instruire *batch*, întrucât poate fi ușor adaptată la minibatch sau stochastic gradient descent. Trecerea de la ecuațiile (6.24) și (6.26) la cazul în care se face antrenarea pe minibatch-uri este imediată: însumarea de p termeni din setul de instruire se substituie cu însumare peste termenii care compun acel minibatch.

Profitând de faptul că funcția de eroare este o sumă de termeni și derivata unei sume de funcții este suma derivatelor, avem:

$$\frac{\partial J}{\partial w_{ij}^{(l)}}(\mathbf{W}, \mathbf{b}) = \left(\frac{1}{p} \sum_{k=1}^p \frac{\partial J}{\partial w_{ij}^{(l)}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{d}^{(k)}) \right) + \lambda w_{ij}^{(l)} \quad (6.29)$$

respectiv pentru ponderile de bias

$$\frac{\partial J}{\partial b_i^{(l)}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p} \sum_{k=1}^p \frac{\partial}{\partial b_i^{(l)}} J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{d}^{(k)}) \quad (6.30)$$

Formele matriceale ale (6.29) și (6.30) rezultă imediat și sunt explicitate în algoritmul de instruire ce urmează.

Algoritmul backpropagation este cel care specifică ordinea de calculare a derivatelor parțiale. Vom folosi în cele ce urmează funcția de eroare L_2 pătratică conform ecuației (6.23); pentru funcția de eroare cross-entropy e nevoie să se calculeze corespunzător formele derivatelor parțiale ale funcției de eroare – a se vedea finalul acestei secțiuni.

Algoritmul funcționează astfel: pentru o pereche de instruire $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ se face pasul de propagare înainte, până se obține valoarea de ieșire $\mathbf{o}^{(i)}$; pentru fiecare strat de neuroni l , începând de la ultimul, se calculează un termen de eroare $\delta^{(l)}$ care măsoară cât de mult e “responsabil” stratul l (mai precis: fiecare neuron din strat) pentru discrepanța dintre ieșirea $\mathbf{o}^{(i)}$ și valoarea dorită $\mathbf{d}^{(i)}$.

Înainte de detalia algoritmul de instruire a rețelei MLP, este nevoie să introducem produsul Hadamard al două matrice; produsul se notează cu \odot și se aplică pentru două matrice care au același număr de linii și respectiv același număr de coloane: dacă $A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ și $B = (b_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ sunt cele două matrice, atunci matricea produs Hadamard $C = A \odot B = (c_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ are elementele:

$$c_{ij} = a_{ij} \cdot b_{ij} \quad (6.31)$$

Algoritmul backpropagation detaliat – varianta batch – este:

1. Inițializează valorile $\Delta \mathbf{W}^{(l)}$, $\Delta \mathbf{b}^{(l)}$ cu matrice nule, pentru $l = 1, \dots, L-1$:

$$\Delta \mathbf{W}^{(l)} = \mathbf{0}_{n_{l+1} \times n_l} \quad (6.32)$$

$$\Delta \mathbf{b}^{(l)} = \mathbf{0}_{n_{l+1}}, \text{ vector coloană} \quad (6.33)$$

2. Pentru fiecare pereche $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ calculează corecția pentru ponderi și ponderile de bias:

- 2.1. Efectuează pasul de propagare înainte, conform secțiunii 6.5, și obține ieșirea estimată $\mathbf{o}^{(i)}$

- 2.2. Pentru stratul de ieșire calculează semnalul de eroare:

$$\delta^{(L)} = - \left(\mathbf{d}^{(i)} - \mathbf{o}^{(i)} \right) \odot f' \left(\mathbf{z}^{(L)} \right) \quad (6.34)$$

unde am presupus că funcția f' se vectorizează peste vectorul $\mathbf{z}^{(L)}$.

- 2.3. Pentru straturile $l = L-1, \dots, 2$ se calculează semnalul de eroare:

$$\delta^{(l)} = \left[\left((\mathbf{W}^{(l)})^t \cdot \delta^{(l+1)} \right) \odot f' \left(\mathbf{z}^{(l)} \right) \right] \quad (6.35)$$

- 2.4. Calculează derivatele parțiale dorite, pentru $l = 1, \dots, L-1$:

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = \delta^{(l+1)} \cdot \left(\mathbf{a}^{(l)} \right)^t \quad (6.36)$$

respectiv pentru ponderile de bias:

$$\frac{\partial J}{\partial \mathbf{b}^{(l)}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = \delta^{(l+1)} \quad (6.37)$$

- 2.5. Acumulează semnalul de corecție, pentru $l = 1, \dots, L-1$:

$$\Delta \mathbf{W}^{(l)} = \Delta \mathbf{W}^{(l)} + \frac{\partial J}{\partial \mathbf{W}^{(l)}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) \quad (6.38)$$

$$\Delta \mathbf{b}^{(l)} = \Delta \mathbf{b}^{(l)} + \frac{\partial J}{\partial \mathbf{b}^{(l)}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) \quad (6.39)$$

3. După ce toate perechile din setul de instruire au fost considerate, modifică valorile ponderilor și coeficienții de bias, pentru $l = 1, \dots, L-1$:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha \left[\left(\frac{1}{p} \Delta \mathbf{W}^{(l)} \right) + \lambda \mathbf{W}^{(l)} \right] \quad (6.40)$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \left[\left(\frac{1}{p} \Delta \mathbf{b}^{(l)} \right) \right] \quad (6.41)$$

4. Repetare: se repetă de la pasul 1 până când eroarea $J(\mathbf{W}, \mathbf{b})$ scade sub un prag E_{max} .

Pentru cazul în care funcția de eroare este cross-entropy (6.26) în loc de eroarea L_2 pătratică, se arată că pentru perechea $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$:

$$\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{d}^{(i)} \quad (6.42)$$

substituind formula de calcul (6.34). Utilizarea funcției de eroare cross entropy duce la o viteză mai mare de învățare pentru probleme de clasificare decât dacă se folosește L_2 pătratică.

6.6.3 Justificarea matematică a algoritmului

TODO

6.7 Utilizarea rețelei

După ce se face antrenarea rețelei, ea se poate folosi pentru a face predicții pentru date din setul de testare $\mathcal{T} = \{\mathbf{x}^{(j)} | 1 \leq j \leq q\}$. Fiecare vector din \mathcal{T} este trecut prin rețea, conform pasului de propagare înainte și se obțin valori de ieșire estimate (predicții) $\mathbf{o}^{(1)}, \dots, \mathbf{o}^{(k)}$, toate din \mathbb{R}^m .

Dacă valorile de ieșire sunt interpretate ca probabilități condiționate, adică:

$$o_i = P(\text{clasa } i | \mathbf{x}), 1 \leq i \leq m \quad (6.43)$$

atunci clasificarea se face găsind acel indice i pentru care o_i e maxim.

6.8 Discuții

TODO

Capitolul 7

Memorii asociative bidirecționale

Memoriile asociative permit stocarea și regăsirea datelor. Căutarea se face pe baza similarității care există între pattern-ul furnizat ca intrare și ceea ce este stocat în rețea. Regăsirea se face pe baza similarității între pattern-ul furnizat și a unui pattern stocat de rețea. Se lucrează cu perechi de pattern-uri asociate memorate de rețea; plecând de la oricare dintre ele sau de la unul similar cu ele se dorește regăsirea celuilalt. Datele memorate sunt reprezentate în întreaga rețea.

7.1 Distanța Hamming

Fie $\mathbf{x} = (x_1, \dots, x_n)^t$ și $\mathbf{y} = (y_1, \dots, y_n)^t$ doi vectori n -dimensionali din spațiul Euclidian având restricțiile $x_i, y_i \in \{-1, +1\}$, $i = 1, \dots, n$. Distanța Euclidiană dintre cei doi vectori este:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (7.1)$$

Având în vedere valorile pe care le pot lua componentele vectorilor \mathbf{x} și \mathbf{y} , avem că:

$$(x_i - y_i)^2 = \begin{cases} 0 & \text{dacă } x_i = y_i \\ (\pm 2)^2 = 4 & \text{dacă } x_i \neq y_i \end{cases} \quad (7.2)$$

deci distanța Euclidiană este $d_E(\mathbf{x}, \mathbf{y}) = \sqrt{4 \cdot \text{diferente}(\mathbf{x}, \mathbf{y})}$ unde prin $\text{diferente}(\mathbf{x}, \mathbf{y})$ am notat numărul de componente din \mathbf{x} și \mathbf{y} care diferă pentru poziții de același indice. Pentru doi vectori \mathbf{x} și \mathbf{y} ca mai sus se definește funcția de *distanță Hamming* d_H ca fiind tocmai numărul de diferențe de pe pozițiile corespunzătoare:

$$d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n (1 - \delta_{x_i, y_i}) \quad (7.3)$$

unde δ este indicatorul (sau simbolul) lui Kronecker:

$$\delta_{a,b} = \begin{cases} 1, & \text{dacă } a = b \\ 0, & \text{altfel} \end{cases} \quad (7.4)$$

Există relația:

$$d_E(\mathbf{x}, \mathbf{y}) = 2\sqrt{d_H(\mathbf{x}, \mathbf{y})} \quad (7.5)$$

Considerăm hipercubul n -dimensional centrat în origine cu latura de lungime 2^1 :

$$\mathbf{H}^n = \left\{ \mathbf{x} = (x_1, x_2, \dots, x_n)^t \in \mathbf{R}^n, x_i \in \{-1, +1\} \right\} = \{-1, +1\}^n \quad (7.6)$$

\mathbf{H}^n se mai numește și cub Hamming.

7.2 Asociatori

Considerăm mulțimea de perechi de pattern-uri $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_p, \mathbf{y}_p)\}$, unde $\mathbf{x}_i \in \mathbf{R}^n$, $\mathbf{y}_i \in \mathbf{R}^m$, $i = \overline{1, p}$. Există trei tipuri de memorii asociative:

1. *Memorii heteroasociative*: implementează o funcție $\Phi : \mathbf{R}^n \rightarrow \mathbf{R}^m$ cu proprietatea că $\Phi(\mathbf{x}_i) = \mathbf{y}_i$, $i = \overline{1, p}$. În plus, cerem ca dacă un vector $\mathbf{x} \in \mathbf{R}^n$ este mai apropiat de un exemplar \mathbf{x}_i ($1 \leq i \leq p$) decât de ceilalți, atunci $\Phi(\mathbf{x}) = \Phi(\mathbf{x}_i) = \mathbf{y}_i$. Apropierea se consideră în sensul unei distanțe convenabil alese.
2. *Memorii interpolative*: implementează o funcție $\Phi : \mathbf{R}^n \rightarrow \mathbf{R}^m$ astfel încât $\Phi(\mathbf{x}_i) = \mathbf{y}_i$, $i = \overline{1, p}$. În plus, dacă $\mathbf{x} \in \mathbf{R}^n$ este $\mathbf{x} = \mathbf{x}_i + \mathbf{d}$, atunci ieșirea memoriei este:

$$\Phi(\mathbf{x}) = \Phi(\mathbf{x}_i + \mathbf{d}) = \mathbf{y}_i + \mathbf{e}, \quad \mathbf{e} \in \mathbf{R}^m \quad (7.7)$$

3. *Memorie autoasociativă*: dacă $\mathbf{y}_i = \mathbf{x}_i$, $i = \overline{1, p}$, atunci o memorie autoasociativă trebuie să respecte proprietățile date de memoria heteroasociativă: $\Phi(\mathbf{x}_i) = \mathbf{x}_i$, $i = \overline{1, p}$ și dacă \mathbf{x} este mai apropiat de \mathbf{x}_i decât de oricare alt exemplar, atunci $\Phi(\mathbf{x}) = \Phi(\mathbf{x}_i) = \mathbf{x}_i$.

Pentru cazul în care setul de vectori $\{\mathbf{x}_i\}_{1 \leq i \leq p}$ este ortonormat² putem construi simplu un asociator interpolativ. Vom defini funcția Φ ca fiind:

$$\Phi(\mathbf{x}) = \left(\mathbf{y}_1 \mathbf{x}_1^t + \dots + \mathbf{y}_p \mathbf{x}_p^t \right) \mathbf{x} \quad (7.8)$$

¹O variantă este considerarea vârfurilor ca având coordonatele 0 sau 1.

²Adică $\mathbf{x}_i^t \cdot \mathbf{x}_j = \delta_{ij}$, $i, j = \overline{1, p}$, cu δ indicatorul lui Kronecker. Dintr-un set de vectori liniar independenți putem obține întotdeauna un sistem de vectori ortonormați prin procedeul Gram-Schmidt de ortonormare. Pentru a reduce din efectul erorilor de rotunjire, se poate folosi procedeul Gram-Schmidt modificat.

Avem că:

$$\begin{aligned}\Phi(\mathbf{x}_j) &= (\mathbf{y}_1 \mathbf{x}_1^t + \cdots + \mathbf{y}_p \mathbf{x}_p^t) \mathbf{x}_j = \sum_{i=1}^p ((\mathbf{y}_i \mathbf{x}_i^t) \mathbf{x}_j) = \\ &= \sum_{i=1}^p (\mathbf{y}_i (\mathbf{x}_i^t \mathbf{x}_j)) = \sum_{i=1}^p (\mathbf{y}_i \delta_{ij}) = \mathbf{y}_j \quad (7.9)\end{aligned}$$

Dacă un argument \mathbf{x} are forma $\mathbf{x} = \mathbf{x}_i + \mathbf{d}$, atunci:

$$\Phi(\mathbf{x}) = \Phi(\mathbf{x}_i + \mathbf{d}) = \mathbf{y}_i + \mathbf{e} \quad (7.10)$$

unde

$$\mathbf{e} = (\mathbf{y}_1 \mathbf{x}_1^t + \cdots + \mathbf{y}_p \mathbf{x}_p^t) \mathbf{d} \quad (7.11)$$

7.3 Memoria asociativă bidirecțională

O memorie asociativă bidirecțională (MAB) este o memorie heteroasociativă constând în două straturi de elemente de procesare (noduri) care sunt interconectate. Elementele pot sau nu să aibă legături cu ele însele (bucle). O reprezentare este dată în figura 7.1. Valorile \mathbf{x} sunt din \mathbf{H}^n iar \mathbf{y} din \mathbf{H}^m . Între noduri există legături cu diferite ponderi.

Spre deosebire de alte tipuri de rețele neurale, ponderile pot fi determinate dacă se cunoaște de dinainte setul de exemplare ce trebuie memorat. Conexiunile sunt bidirecționale: putem furniza ca intrare o valoare în stratul \mathbf{x} iar ieșirea să fie dată de stratul \mathbf{y} sau invers.

Pentru construirea matricii ponderilor se poate folosi o idee similară cu cea de la memoria interpolativă:

$$\mathbf{w} = \mathbf{y}_1 \mathbf{x}_1^t + \cdots + \mathbf{y}_p \mathbf{x}_p^t, \quad (7.12)$$

matrice care dă ponderile legăturilor de la stratul \mathbf{x} la stratul \mathbf{y} . Matricea ponderilor în sens invers este \mathbf{w}^t . Memoria poate deveni autoasociativă prin stabilirea lui \mathbf{w} ca fiind:

$$\mathbf{w} = \mathbf{x}_1 \mathbf{x}_1^t + \cdots + \mathbf{x}_p \mathbf{x}_p^t \quad (7.13)$$

Odată matricea de ponderi contruită, se poate utiliza MAB pentru regăsirea datelor stocate prin furnizarea unor date cheie, suficient de apropiate de cele din setul de instruire. Vom vedea că această intrare poate fi obținută prin perturbarea unei valori din setul de instruire, iar MAB poate încă să determine cheia originală și valoarea asociată ei.

Pașii de lucru sunt următorii:

1. se aplică perechea inițială de vectori $(\mathbf{x}_0, \mathbf{y}_0)$ celor două straturi de neuroni;

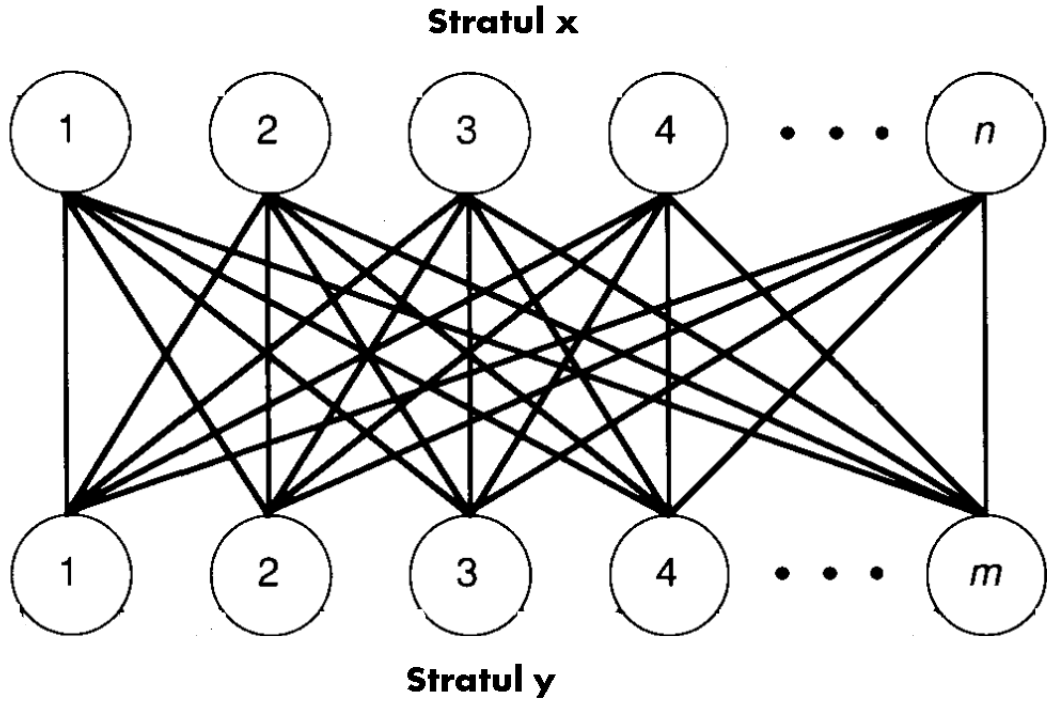


Figura 7.1: Arhitectura unei memorii asociative bidirecționale

2. se propagă informația de la stratul x la stratul y și se modifică valorile din stratul y ;
3. se propagă informația de la y la x și se modifică valorile din stratul x ;
4. se repetă pașii 2 și 3 până când nu mai apare nicio modificare în noduri.

Se poate ca datele să înceapă să se propage de la stratul y la stratul x . Plimbarea datelor în ambele sensuri dă natura bidirecțională a rețelei. Când rețeaua se stabilizează, de regulă se regăsește în stratul x valoarea x_i care este cea mai apropiată de x relativ la distanța Hamming și valoarea y_i asociată cu x_i (sau complementele acestora, a se vedea exemplul următor).

Procesarea care se face în momentul transmiterii informației de la stratul x la stratul y este dată de ecuația:

$$\mathbf{net}^y = \mathbf{w} \cdot \mathbf{x} \quad (7.14)$$

sau pe componente:

$$net_i^y = \sum_{j=1}^n w_{ij}x_j, \quad i = \overline{1, m} \quad (7.15)$$

unde \mathbf{net}^y este vectorul de stare pentru stratul y . Pentru transmiterea în

sens invers are loc un proces asemănător:

$$\mathbf{net}^x = \mathbf{w}^t \mathbf{y} \quad (7.16)$$

sau pe componente:

$$net_i^x = \sum_{j=1}^m w_{ji} \cdot y_j, \quad i = \overline{1, n} \quad (7.17)$$

Valoarea de ieșire a unui neuron depinde de intrări și de valoarea lui curentă. Mai clar, valoarea de la momentul $t + 1$ pentru nodul y_i este dată de:

$$y_i(t+1) = \begin{cases} +1, & \text{dacă } net_i^y > 0 \\ y_i(t), & \text{dacă } net_i^y = 0 \\ -1, & \text{dacă } net_i^y < 0 \end{cases} \quad (7.18)$$

Valorile de ieșire pentru stratul \mathbf{x} se calculează similar.

Exemplu: plecăm de la perechea de pattern-uri:

$$\mathbf{x}_1 = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \quad (7.19)$$

$$\mathbf{x}_2 = (1, 1, 1, -1, -1, -1, 1, 1, -1, -1)^t \quad (7.20)$$

cu ieșirile asociate corespunzătoare

$$\mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t \quad (7.21)$$

$$\mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \quad (7.22)$$

Matricea ponderilor este:

$$\mathbf{w} = \begin{pmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{pmatrix} \quad (7.23)$$

Vom considera ca vector de intrare $\mathbf{x}_0 = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$ cu vectorul \mathbf{y}_0 asociat $(1, 1, 1, 1, -1, -1)^t$. Remarcăm că valorile \mathbf{x}_0 și \mathbf{y}_0 nu sunt printre valorile învățate. Valoarea de ieșire \mathbf{y}_0 poate fi dată ca un vector binar bipolar cu componente aleatoare. Propagarea valorilor dinspre stratul \mathbf{x} către \mathbf{y} duce la determinarea valorii $\mathbf{net}^y = (4, -12, -12, -12, -4, 12)^t$. Noul vector din stratul \mathbf{y} este $\mathbf{y} = (1, -1, -1, -1, -1, 1)^t$. Propagând înapoi către stratul \mathbf{x} obținem $\mathbf{x} = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$. Propagări succesive într-un sens sau în celălalt nu duc la modificări ale valorilor din straturile \mathbf{x} sau \mathbf{y} . Perechea (\mathbf{x}, \mathbf{y}) la care se stabilizează rețeaua este chiar perechea $(\mathbf{x}_1, \mathbf{y}_1)$. S-a regăsit astfel o pereche de exemplare din cele cu care

s-a instruit rețeaua, chiar dacă s-a plecat de la valori care nu se regăsesc printre cele învățate.

Să considerăm situația în care se pleacă de la valorile inițiale:

$$\mathbf{x}_0 = (-1, 1, 1, -1, 1, 1, 1, -1, 1, -1)^t, \mathbf{y}_0 = (-1, 1, -1, 1, -1, -1)^t$$

Propagând de la stratul \mathbf{x} la \mathbf{y} , obținem $\mathbf{y} = (-1, 1, 1, 1, 1, -1)^t$. Propagând în direcția inversă, obținem $\mathbf{x} = (-1, 1, 1, -1, 1, -1, -1, 1, 1, -1)^t$ și rețeaua se stabilizează. Se observă că valorile stabile (\mathbf{x}, \mathbf{y}) sunt chiar $(\mathbf{x}_1^c, \mathbf{y}_1^c)$ unde \mathbf{a}^c este vectorul format cu valorile complementate ce compun pe \mathbf{a}^3 . Aceasta este o proprietate a MAB: dacă memoria stochează perechea (\mathbf{x}, \mathbf{y}) , atunci stochează și perechea $(\mathbf{x}^c, \mathbf{y}^c)$ și stabilizarea rețelei se poate face pe o astfel de pereche de complemente.

7.4 Funcția de energie a MAB

În timpul propagării valorilor dinspre stratul \mathbf{x} spre \mathbf{y} sau invers, valorile din nodurile rețelei se modifică, ceea ce ne permite să vedem evoluția stării acestora ca o funcție de timp. Vom asocia memoriei o funcție de energie a cărei valoare este dependentă de valorile \mathbf{x} și \mathbf{y} din noduri; vrem să arătăm că funcția de energie converge la un punct limită pe durata propagării datelor între cele două straturi. Convergența se traduce prin stabilizarea rețelei.

Funcția de energie considerată este:

$$E(\mathbf{x}, \mathbf{y}) = -\mathbf{y}^t \mathbf{w} \mathbf{x} = -\sum_{i=1}^m \sum_{j=1}^n y_i w_{ij} x_j$$

Avem următoarea teoremă privitoare la comportamentul MAB pentru funcția de energie:

Teorema 1. 1. Orice modificare a stării stratului \mathbf{x} sau \mathbf{y} în timpul procesării din MAB duce la scăderea lui E ;

2. E este mărginită inferior de $E_{min} = -\sum_{i,j} |w_{ij}|$;

3. Dacă valoarea lui E se schimbă atunci modificarea nu este arbitrar de mică.

Demonstrație. Pentru cazul în care se face propagare dinspre stratul x către stratul y^4 , presupunem că se face o modificare pentru vectorul \mathbf{y} pe o singură poziție, fie ea l , $1 \leq l \leq m$. Energia asociată intrării curente este:

$$E = -\sum_{j=1}^n y_l w_{lj} x_j - \sum_{i=1, i \neq l}^m \sum_{j=1}^n y_i w_{ij} x_j \quad (7.24)$$

³Aici “complementar” se definește ca “de semn opus”, deci $\mathbf{a}^c = -\mathbf{a}$.

⁴Dacă propagarea începe de la stratul y la x , demonstrația se face în mod analog.

Dacă se face modificarea valorii y_l în y_l^{nou} , noua valoare a energiei va fi:

$$E^{nou} = - \sum_{j=1}^n y_l^{nou} w_{lj} x_j - \sum_{i=1, i \neq l}^m \sum_{j=1}^n y_i w_{ij} x_j \quad (7.25)$$

și deci variația energiei este:

$$\Delta E = E^{nou} - E = (y_l - y_l^{nou}) \sum_{j=1}^n w_{lj} x_j = (y_l - y_l^{nou}) net_l^y \quad (7.26)$$

Avem posibilitățile:

1. dacă $y_l = +1$, atunci $y_l^{nou} = -1$. Avem $y_l - y_l^{nou} = 2 > 0$ dar dacă $y_l^{nou} = -1$, asta se datorează lui $net_l^y < 0$ (a se vedea ecuația 7.18). Valoarea lui ΔE este produsul a doi termeni întregi nenuli și de semn contrar și deci este o valoare întreagă strict mai mică decât zero.
2. dacă $y_l = -1$, atunci $y_l^{nou} = +1$ și de aici $y_l - y_l^{nou} = -2 < 0$. Dar trecerea de la y_l la y_l^{nou} presupune că $net_l^y > 0$ (ecuația 7.18) și din nou ΔE este produsul a două valori întregi nenule și de semn contrar, ca atare de valoare întreagă strict mai mică decât zero.

Situația în care mai mult de un termen din \mathbf{y}^{nou} este modificat față de \mathbf{y} se tratează similar, cu observația că scăderea lui ΔE este și mai accentuată.

Similar se arată că modificarea stării unui neuron din stratul de intrare de asemenea scade valoarea funcției de energie.

Pentru cea de a doua parte a teoremei avem:

$$\begin{aligned} -E(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^m \sum_{j=1}^n y_i w_{ij} x_j \stackrel{a \leq |a|}{\leq} \left| \sum_{i=1}^m \sum_{j=1}^n y_i w_{ij} x_j \right| \stackrel{|a+b| \leq |a| + |b|}{\leq} \\ &\stackrel{|a+b| \leq |a| + |b|}{\leq} \sum_{i=1}^m \sum_{j=1}^n |y_i| \cdot |w_{ij}| \cdot |x_j| \stackrel{|x_i| = |y_i| = 1}{=} \sum_{i=1}^m \sum_{j=1}^n |w_{ij}|. \end{aligned} \quad (7.27)$$

de unde $E(\mathbf{x}, \mathbf{y}) \geq - \sum_{i=1}^m \sum_{j=1}^n |w_{ij}|$.

Partea a treia a teoremei rezultă din cele obținute în prima a demonstrației: dacă a apare modificare în stările neuronilor, ΔE este un număr întreg strict mai mic decât zero. \square

Demonstrație practică: <http://facstaff.cbu.edu/~pong/ai/bam/bamapplet.html>.

7.5 Capacitatea de memorie

Capacitatea de memorare a unei memorii asociative bidirecționale este $\min(m, n)$. După alți autori, un prag superior pentru numărul de perechi de pattern-uri care pot fi memorate ar fi $\sqrt{\min(m, n)}$.

Capitolul 8

Rețele neurale cu funcții de activare radială

8.1 Teorema lui Cover

Pentru cazul vectorilor ce nu pot fi separați liniar, perceptronul multistrat poate determina o funcție de separare, datorită caracterului de aproximator universal. Există și o altă variantă de rezolva problema discernerii între clase ce nu se pot separa liniar, *folosind însă un separator liniar*, bazată pe doi pași:

1. mulțimea de instruire dată în spațiul original este transformată într-un alt spațiu, în care, în anumite condiții, liniar separabilitatea poate apărea cu probabilitate mare; fundamentul matematic este dat de teorema lui Cover (vedeți mai jos);
2. prin utilizarea unui model de separare liniară (perceptron, SVM liniar, regresie logistică) se separă clasele în cel de-al doilea spațiu.

Rezultatul se poate materializa printr-o rețea cu funcții de activare radială, formată din 3 straturi:

- stratul de intrare alcătuit din unități senzoriale care conectează rețeaua la mediu;
- stratul de neuroni ascunși ce aplică transformări neliniare pe datele din spațiul de intrare. Neuronii din acest strat sunt antrenați prin instruire nesupervizată;
- stratul de ieșire produce o transformare liniară, iar ponderile dintre stratul ascuns și stratul de ieșire sunt obținute prin instruire supervizată. Acesta furnizează valoarea de ieșire pentru vectorul de intrare curent.

Vector de intrare	$\varphi_1(\mathbf{x}_i)$	$\varphi_2(\mathbf{x}_i)$
$\mathbf{x}_1 = (1, 1)^t$	1	0.1353
$\mathbf{x}_2 = (0, 1)^t$	0.3678	0.3678
$\mathbf{x}_3 = (0, 0)^t$	0.1353	1
$\mathbf{x}_4 = (1, 0)^t$	0.3678	0.3678

Tabela 8.1: Valorile funcțiilor φ pentru punctele problemei XOR

Următoarea teoremă arată motivul pentru care se face o transformare a datelor originare în alte date dintr-un spațiu cu mai multe dimensiuni decât cel original:

Teorema 2 (Cover, 1965). *O problemă de clasificare de vectori transformați în mod neliniar dintr-un spațiu A de o anumită dimensiune într-un alt spațiu cu dimensiune mai mare este mai probabil să devină liniar separabilă decât în A , cu condiția ca A să nu fie dens populat.*

Rezultatul este util, deoarece pentru cazuri liniar separabile, un perceptron discret poate să obțină un hiperplan de separare în timp finit. Pentru a se obține o asemenea transformare, se pleacă de la spațiul A n dimensional în care se găsesc vectorii $\mathbf{x}_1, \dots, \mathbf{x}_P$ și se ajunge la un spațiu m dimensional ($m \geq n$) prin funcția:

$$\mathbf{x} \xrightarrow{\phi} \phi(\mathbf{x}) = (\varphi_1(\mathbf{x}), \dots, \varphi_m(\mathbf{x}))^t \quad (8.1)$$

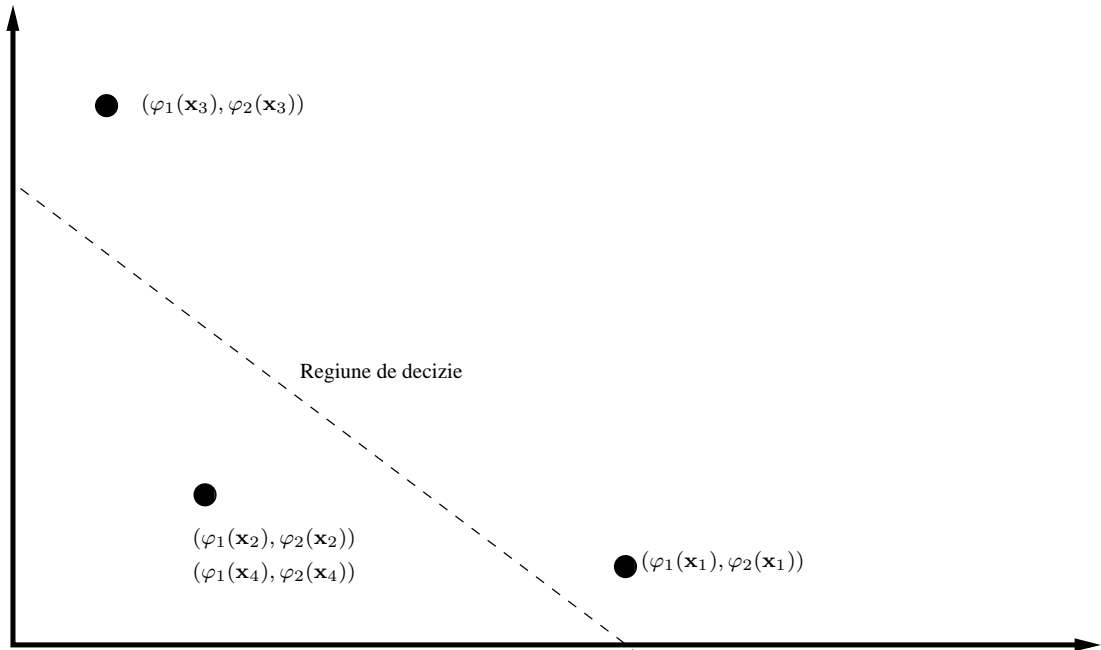
unde $\varphi_i(\cdot), i = \overline{1, m}$ sunt funcții reale neliniare; în rețeaua neurală funcția φ_i e calculată de neuronul i din stratul ascuns.

Exemplu: considerăm problema XOR, în care 4 puncte sunt asiguate la două clase, astfel: punctele de coordonate $(0, 0)^t$ și $(1, 1)^t$ aparțin unei clase, iar $(0, 1)^t$ și $(1, 0)^t$ aparțin celeilalte clase. Nu se poate determina o dreaptă în plan care să separe cele două clase de puncte. Considerăm funcțiile $\varphi_1, \varphi_2 : \mathbb{R}^2 \rightarrow \mathbb{R}$

$$\varphi_1(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{t}_1\|), \mathbf{t}_1 = (1, 1)^t$$

$$\varphi_2(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{t}_2\|), \mathbf{t}_2 = (0, 0)^t$$

unde $\mathbf{x} \in \mathbb{R}^2$ iar $\|\cdot\|$ este norma Euclidiană L_2 . Pornind de la un vector de intrare $\mathbf{x} \in \mathbb{R}^2$ se ajunge la un vector tot din \mathbb{R}^2 dat de $(\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}))$. Valorile rezultate pentru funcțiile $\varphi_{1,2}$ calculate în cele 4 puncte ale problemei XOR sunt date în tabelul 8.1. Figura 8.1 dă reprezentarea punctelor transformate prin aplicarea celor două funcții. Se observă că problema devine una liniar separabilă, folosind modificări neliniare ale datelor inițiale; mai mult, nu a fost nevoie în acest caz să se mărească dimensiunea spațiului de ieșire.



8.2 Funcții cu activare radială

Teorema lui Cover afirmă că pentru o problemă ce nu e liniar separabilă, prin transformări adecvate cresc șansele de a se transforma într-una care e liniar separabilă. Să considerăm o rețea neurală de tip feedforward cu un strat de intrare cu n noduri, un singur strat ascuns și un strat de ieșire cu un singur nod¹. Această rețea produce o funcție de la un spațiu n -dimensional la unul unidimensional:

$$s : \mathbb{R}^n \rightarrow \mathbb{R} \quad (8.2)$$

Funcția s poate fi văzută ca o hipersuprafață $\Gamma \subset \mathbb{R}^{n+1}$; hipersuprafața Γ este necunoscută și se determină pe baza setului de instruire.

Se lucrează în două etape: una de instruire și alta de generalizare. În etapa de instruire se folosește o procedură oarecare prin care se determină hipersuprafața Γ , plecând de la setul de date de antrenare, adică se obține funcția s . În etapa de generalizare se folosește un procedeu de interpolare pentru a determina valori de ieșire corespunzătoare unor vectori din spațiul de intrare \mathbb{R}^n .

Problema de interpolare este:

Dându-se un set de P puncte diferite $\{\mathbf{x}_i \in \mathbb{R}^n | i = \overline{1, P}\}$ și un set corespunzător de P numere reale $\{d_i \in \mathbb{R} | i = \overline{1, P}\}$, să se determine o funcție

¹În cele ce urmează în acest capitol, ieșirea unică este pentru simplificarea prezentării. Pentru cazul în care ieșirea este din spațiul \mathbb{R}^m sau dacă avem o problemă de clasificare cu m clase, stratul de ieșire va avea m neuroni.

$F : \mathbb{R}^n \rightarrow \mathbb{R}$ care satisface proprietatea de interpolare:

$$F(\mathbf{x}_i) = d_i, \quad i = \overline{1, P} \quad (8.3)$$

Tehnica funcțiilor cu activare radială (Radial Basis Functions, RBF) consideră că funcția F are forma:

$$F(\mathbf{x}) = \sum_{i=1}^P w_i \varphi_i(\|\mathbf{x} - \mathbf{x}_i\|) \quad (8.4)$$

unde φ_i sunt funcții neliniare reale, cunoscute ca funcții cu activare radială. Punctele \mathbf{x}_i sunt “centrele” (parametri ai) funcțiilor RBF.

Impunând condiția (8.3) asupra formei (8.4), avem următorul sistem liniar în care necunoscutele sunt $w_i, i = \overline{1, P}$:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1P} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2P} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{P1} & \varphi_{P2} & \cdots & \varphi_{PP} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_P \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{bmatrix} \quad (8.5)$$

unde

$$\varphi_{ij} = \varphi_i(\|\mathbf{x}_j - \mathbf{x}_i\|), \quad i, j = \overline{1, P} \quad (8.6)$$

Notăm cu $\mathbf{d} = (d_1, d_2, \dots, d_P)^t$, $\mathbf{w} = (w_1, w_2, \dots, w_P)^t$, $\Phi = \{\varphi_{ij}\}_{i,j=\overline{1,P}}$. Numim matricea Φ matricea de interpolare. Se poate rescrie (8.5) sub forma:

$$\Phi \cdot \mathbf{w} = \mathbf{d} \quad (8.7)$$

Dacă matricea Φ este nesingulară, atunci ponderile sunt $\mathbf{w} = \Phi^{-1}\mathbf{d}$. Pentru discuția asupra inversabilității lui Φ , considerăm teorema lui Michelli:

Teorema 3 (Michelli, 1986). *Fie $\{\mathbf{x}_i\}_{i=\overline{1,P}}$ un set de puncte distincte din \mathbb{R}^n . Atunci matricea de interpolare Φ este inversabilă dacă funcțiile φ_i au una din formele:*

1. funcție multipătratică:

$$\varphi_i(r_i) = \sqrt{r_i^2 + c^2}, \quad c > 0 \quad (8.8)$$

2. funcție inversă de multipătratică:

$$\varphi_i(r_i) = \frac{1}{\sqrt{r_i^2 + c^2}}, \quad c > 0 \quad (8.9)$$

3. funcție Gaussiană:

$$\varphi_i(r_i) = \exp\left(-\frac{r_i^2}{2\sigma^2}\right), \quad \sigma > 0 \quad (8.10)$$

unde în toate cele trei cazuri r_i este distanța Euclidiană dintre vectorii \mathbf{x} și \mathbf{x}_i (echivalent: norma diferenței dintre \mathbf{x} și \mathbf{x}_i , $\|\mathbf{x} - \mathbf{x}_i\|$).

8.3 Rețele cu funcții cu activare radială

O rețea cu funcții cu activare radială este ilustrată în figura 8.1; ea constă din trei straturi:

1. *stratul de intrare*, care constă din n noduri, unde n este dimensiunea spațiului de intrare.
2. *stratul ascuns*, care e format din același număr de neuroni ca numărul de date din setul de antrenare, P ; fiecare neuron j , $j = \overline{1, P}$ are funcție cu activare radială $\varphi_j(\|\mathbf{x} - \mathbf{x}_j\|)$;
3. *stratul de ieșire*, care în cazul exemplificat este format dintr-un singur neuron. Acest strat poate însă avea oricâți neuroni.

Pentru funcțiile φ_j vom considera funcțiile Gaussiene:

$$\varphi_j(\|\mathbf{x} - \mathbf{x}_j\|) = \exp\left(-\frac{1}{2\sigma_j^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right), \quad j = \overline{1, P} \quad (8.11)$$

unde σ_j este lățimea unei funcții Gaussiene centrate în \mathbf{x}_j . De regulă tuturor Gaussienelor li se asignează aceeași lățime σ ; diferența dintre funcții este dată în acest caz de centrele \mathbf{x}_j .

Din punct de vedere practic se evită folosirea tuturor datelor din setul de instruire pentru crearea de funcții de activare de tip radial. Un motiv ar fi că setul $\{(\mathbf{x}_i, d_i) | i = \overline{1, P}\}$ poate prezenta zgomot, cel mai frecvent datorat erorilor de măsurare. Folosirea unui procedeu de interpolare plecând de la un set de date cu zgomot duce la rezultate proaste. În plus, numărul de noduri rezultat în rețeaua din figura 8.1 s-ar putea să fie prohibitiv. Ca atare, în practică numărul de noduri din stratul ascuns este mult redus. Funcția F se transformă într-o funcție de aproximare de forma:

$$F(\mathbf{x}) = \sum_{j=1}^K w_j \varphi_j(\|\mathbf{x} - \hat{\boldsymbol{\mu}}_j\|) \quad (8.12)$$

unde dimensiunea vectorului de intrare \mathbf{x} este aceeași ca și cea de până acum, $K < P$ iar punctele \mathbf{x}_j nu sunt neapărat din setul de instruire (ele pot proveni dintr-un proces de grupare automată). Interpretarea ca rețea neurală este dată în figura 8.2.

Pentru determinarea celor K centri $\hat{\boldsymbol{\mu}}_k$, $1 \leq k \leq K$ ale centrilor din stratul ascuns se poate utiliza o metodă oarecare de grupare automată pe baza similarităților (clustering), dar care să producă noțiunea de centroid². Vom prezenta în cele ce urmează metoda K -means clustering.

²Excludem deci algoritmi de clustering precum **DBSCAN** sau **OPTICS**.

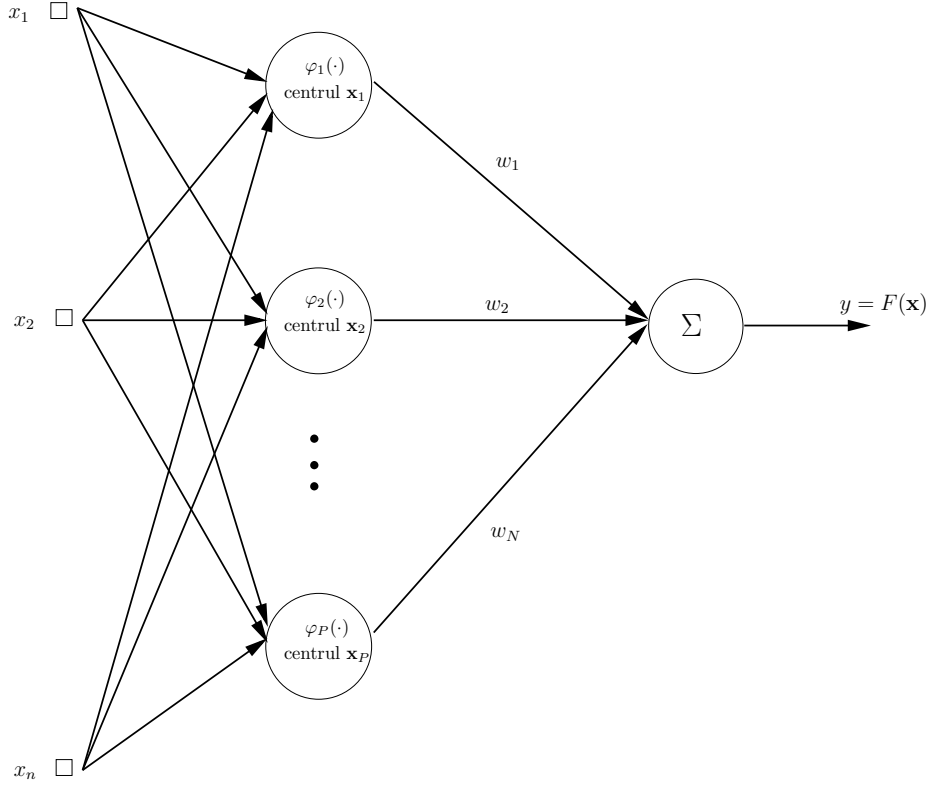


Figura 8.1: Structura unei rețele RBF, plecând de la funcția de interpolare din ecuația 8.4.

8.4 Clustering folosind algoritmul K-means

Clustering-ul este o formă de învățare nesupervizată în care un set de vectori este partiționat în grupuri. Se urmărește minimizarea unei funcții de cost definită convenabil, care cuantifică disimilaritatea totală a vectorilor. Clusterele ar trebui obținute de așa manieră încât vectorii similari să fie grupați în același cluster, iar cei nesimilari să aparțină de alte clustere.

Considerăm un set de P puncte, $\{\mathbf{x}_i\}_{i=\overline{1,P}}$ ce urmează să fie partiționat în K clustere³, $K < P$. Fie $C(i)$ indicele de cluster de care aparține vectorul \mathbf{x}_i , $i = \overline{1,P}$. Evident, $1 \leq C(i) \leq K$. Considerăm $d(\mathbf{x}_i, \mathbf{x}_j)$ o măsură a deosebirii dintre perechile de vectori \mathbf{x}_i și \mathbf{x}_j . Pentru clustering se cere minimizarea funcției:

$$J(C) = \frac{1}{2} \sum_{k=1}^K \sum_{i:C(i)=k} \sum_{j:C(j)=k} d(\mathbf{x}_i, \mathbf{x}_j) \quad (8.13)$$

³Cel mai frecvent, valoarea lui K este furnizată de utilizator.

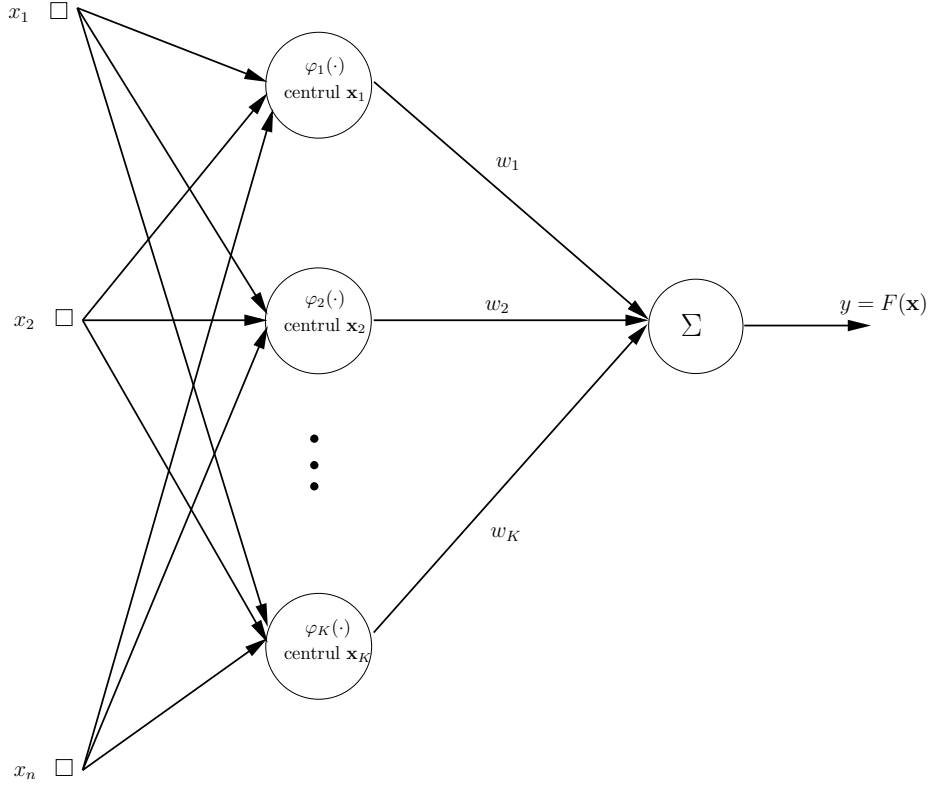


Figura 8.2: Structura unei rețele RBF, folosind mai puține noduri decât în figura 8.1. Centrii \mathbf{x}_i , $i = \overline{1, K}$ se obțin printr-un procedeu de clustering.

unde C este partiția dată de cele K clustere:

$$C = \left\{ \{i | 1 \leq i \leq P, C(i) = k\} | k = \overline{1, K} \right\}$$

În algoritmul K -means drept măsură de ne-similaritate se folosește pătratul distanței Euclidiene:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2 \quad (8.14)$$

În urma procesului de clustering vor rezulta K centroizi – centri de clustere – notați $\hat{\boldsymbol{\mu}}_k \in \mathbb{R}^n$, $k = \overline{1, K}$.

Modificăm forma funcției de eroare J astfel încât să se ia în considerare distanțele dintre vectorii \mathbf{x}_i și centroizii $\hat{\boldsymbol{\mu}}_k$ ai clusterelor de care aparțin:

$$J(C) = \frac{1}{2} \sum_{k=1}^K \sum_{i: C(i)=k} \|\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k\|^2 \quad (8.15)$$

Presupunând funcția C cunoscută, cum anume se poziționează centroizii astfel încât să se minimizeze $J(C)$? Algoritmul K -means determină niște

valori pentru $\hat{\mu}_k$ printr-un proces iterativ, astfel încât $J(C)$ să scadă. Este un algoritm euristic, nu garantează faptul că se ajunge la minimul global al lui $J(C)$.

Algoritmul alege aleator K centroizi inițiali $\hat{\mu}_i^{(1)}$, $i = \overline{1, K}$, inițializându-i cu valori fie din setul de instruire, fie setate la întâmplare cu valori din spațiul de intrare – sau conform algoritmului K-means++, a se vedea mai jos. Avem apoi o succesiune de iterații cu pașii:

- *Pasul de asignare:* Calculează pentru orice k , $k = 1 \dots K$:

$$S_k^{(t)} = \left\{ \mathbf{x}_i : \left\| \mathbf{x}_i - \hat{\mu}_k^{(t)} \right\| \leq \left\| \mathbf{x}_i - \hat{\mu}_j^{(t)} \right\|, \forall j = \overline{1, K}, j \neq k, i = \overline{1, P} \right\}$$

adică pentru fiecare punct \mathbf{x}_i se determină care este cel mai apropiat centroid de care aparține; $S_k^{(t)}$ este mulțimea vectorilor din setul de instruire ce sunt cel mai apropiate de centroidul $\hat{\mu}_k^{(t)}$, la iterația t .

- *Modificarea centroizilor:*

$$\hat{\mu}_k^{(t+1)} = \frac{1}{|S_k^{(t)}|} \cdot \sum_{\mathbf{x}_i \in S_k^{(t)}} \mathbf{x}_i, \quad k = \overline{1, K}$$

unde $|S_k^{(t)}|$ este numărul de elemente ale mulțimii $S_k^{(t)}$.

Algoritmul K-means se oprește atunci când pasul de asignare nu mai modifică mulțimile $S_k^{(t)}$.

Nu există nicio demonstrație că algoritmul converge către minimul global al funcției J ; fiind însă rapid în practică – adică necesitând puțini pași până la oprire – se poate reporni cu alte valori ale centroizilor inițiali $\hat{\mu}_k^{(1)}$, $k = \overline{1, K}$. Situația (centroizii) pentru care $J(C)$ are valoarea cea mai mică în aceste încercări este reținută.

Se poate considera că inițializarea centroizilor inițiali $\hat{\mu}_k^{(1)}$, $i = \overline{1, K}$ nu ar trebui lăsată la voia întâmplării și că se poate îmbunătăți considerabil performanța algoritmului printr-o alegere îngrijită a lor. Un caz nefavorabil este dat în figura 8.4. Să considerăm un dreptunghi cu laturile de lungime $L \gg l$, având în cele patru vârfuri câte un punct $\mathbf{x}_i \in \mathbb{R}^2$, $i = \overline{1, 4}$. Dacă considerăm $K = 2$ și centroizii sunt aleși inițial la jumătatea laturilor de lungime mai mare, atunci algoritmul se oprește după o iterație cu $J(C) = \frac{1}{2} \cdot 4 \left(\frac{L}{2} \right)^2 = \frac{1}{2} L^2$ (punctele \mathbf{x}_1 și \mathbf{x}_3 aparțin clusterului de centroid $\hat{\mu}_1^{(1)}$, iar celelalte două celui de al doilea cluster). Dacă alegerea punctelor se face ca în figura 8.4, atunci se obține valoarea $J(C) = \frac{1}{2} l^2$ – și se poate arăta că aceasta este și valoarea minimă a lui J . Având în vedere că L poate fi luat oricât de mare față de l , rezultă că o alegere neinspirată a centroizilor poate să ducă la o valoare oricât de depărtată față de optim pentru funcția J .

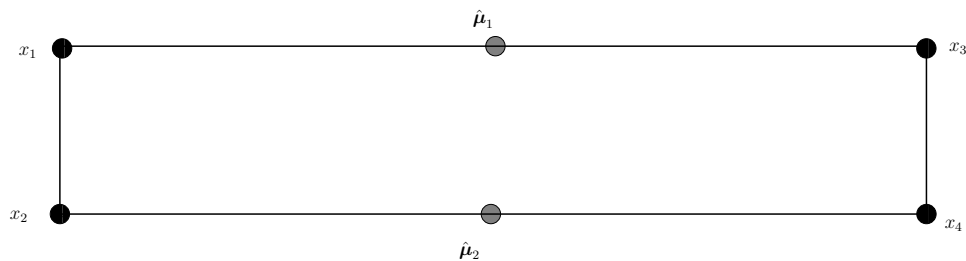


Figura 8.3: Caz nefavorabil pentru K -means la alegerea centrozilor inițiali.

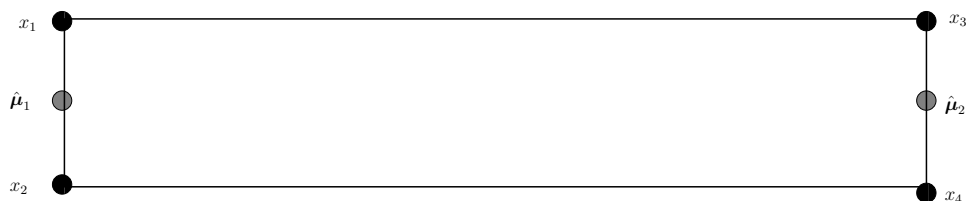


Figura 8.4: Alegerea optimă a centrozilor inițiali.

Ca atare, s-a dezvoltat algoritmul K -means++ care are ca scop determinarea unor centroizi inițiali aleși mai potrivit. Alegerea celor K centroizi se face după următorii pași [17]:

1. Alege primul centroid aleator din setul de antrenare;
2. Pentru fiecare punct \mathbf{x}_i calculează $D(\mathbf{x}_i)$ ($1 \leq i \leq N$), distanța de la el până la cel mai apropiat din centroizii determinați până la pasul curent;
3. Alege aleator un nou punct din setul de antrenare, folosind o probabilitate de alegere o funcție crescătoare cu distanța dată de D ;
4. Repetă pașii 2 și 3 până când s-au ales K centroizi;

Se aplică apoi algoritmul K -means pentru centroizii astfel determinați.

Costul suplimentar indus de determinarea celor K centroizi ca mai sus este neglijabil față de efectele benefice asupra rezultatului final. Rezultate teoretice pentru K -means++ se găsesc în [17].

8.5 Determinarea ponderilor pentru RBF

Determinarea ponderilor legăturilor dintre stratul ascuns și cel de ieșire este următorul pas. Problema este una de determinare a ponderilor pentru

o problemă de regresie liniară și se tratează cu tehnicile din secțiunile 2.3 și 2.4.

Pentru cazul în care problema este una de regresie în care vectorii de ieșire sunt din \mathbb{R}^m , fiecare neuron de ieșire își poate ajusta setul de ponderi independent de ponderile celorlalte ieșiri; se aplică una din cele două metode de mai sus.

8.6 Algoritmul de instruire a rețelei RBF

Sintetizăm pe baza expunerii de până acum procedura de instruire a unei rețele RBF. Stratul de intrare este fix, având numărul de noduri dat de dimensiunea intrării. Stratul ascuns se obține rulând algoritm de clustering (*e.g.* K -means precedat de K -means++) peste setul de antrenare și rezultând K centroizi $\hat{\mu}_j, j = \overline{1, K}$. Acești centri de clustere devin centrii unor funcții Gaussiene asignate nodurilor ascunse. Pentru fiecare astfel de Gaussiană este asignată o aceeași lățime σ , calculată ca:

$$\sigma = \frac{d_{max}}{\sqrt{2K}} \quad (8.16)$$

unde d_{max} este distanța maximă dintre perechile de centroizi. În stratul de ieșire sunt tot atâtea noduri cât este dimensiunea spațiului de ieșire. Ponderile legăturilor dintre stratul ascuns și stratul de ieșire se calculează ca pentru o problemă de regresie liniară (metoda de căutare după gradient din secțiunea 2.3 sau prin metoda pseudoinversei din secțiunea 2.4).

Capitolul 9

Fuzzy ARTMAP

9.1 Învățarea incrementală

Învățarea incrementală este o caracteristică asociată unor sisteme adaptive care:

1. agregă cunoștințe noi din date noi;
2. nu cer acces la datele utilizate pentru a antrena sistemul până la momentul curent;
3. păstrează cunoștințele deprinse anterior;
4. se pot acomoda cu noi categorii care pot fi introduse de noi date de instruire.

9.2 Proprietăți dezirabile ale sistemelor instruibile

Pentru un sistem instruibil următoarele proprietăți sunt văzute ca fiind esențiale:

1. învățare rapidă;
2. învățare din noi date fără a fi nevoie să se reantreneze cu datele parcurse anterior – regăsită în învățarea incrementală;
3. rezolvarea de probleme neseparabile liniar – o varietate liniară nu este întotdeauna o suprafață de separare bună;
4. în cazul unui clasificator: abilitate de a da nu doar clasa de apartenență a unui vector de intrare, ci și plauzibilitatea acestei apartenențe; sunt favorizați aici estimatorii de probabilitate condiționată de forma $P(\text{clasa}|\text{intrare})$; de exemplu, $P(\text{email} = \text{spam}|\text{continut email})$;

5. pentru clasificatori: oferire de explicații asupra modului în care datele sunt clasificate, de ce sunt clasificate într-un anumit mod; prin această trăsătură se evită tratarea clasificatorului ca o cutie neagră ce nu poate să explice modul de producere a deciziilor;
6. posibilitate de reglare automată a hiperparametrilor modelului; de exemplu, pentru perceptronul multistrat hiperparametrii de interes sunt numărul de straturi ascunse, numărul neuronilor din fiecare strat ascuns, rata de învățare etc.;
7. aproximarea de funcții fără a cunoaște distribuția inițială a datelor; rareori datele se supun unei distribuții clasice;
8. pentru clase care prezintă suprapuneri, să se creeze regiuni în spațiul de intrare care să realizeze cea mai mică suprapunere; problema asocierilor de tip un vector de intrare-la-mai multe clase trebuie tratată explicit.

9.3 Dilema stabilitate-plasticitate

Un sistem instruibil ar trebui să aibă două proprietăți:

1. *plasticitate* - înseamnă adaptarea la mediul din care provin vectorii de instruire. Altfel zis, plasticitatea este capacitatea de învățare.
2. *stabilitate* - se referă la păstrarea cunoștințelor învățate anterior.

Atunci când se prezintă noi intrări unei rețele neurale, cele vechi pot fi uitate. Ponderile rețelei trebuie să fie suficient de flexibile pentru a învăța noi cunoștințe (trăsătura de plasticitate), dar nu atât de mult încât să uite ceea ce s-a învățat anterior (trăsătura de stabilitate). Acest conflict dintre stabilitate și plasticitate se numește *dilema stabilitate-plasticitate*. Cele mai multe dintre rețelele neurale existente sunt fie stabile dar incapabile de a învăța rapid noi vectori (de exemplu memoriile asociative bidirecționale), fie plastice dar instabile; de aceea, dilema menționată este una din problemele de interes în domeniul modelelor instruibile. S-a formulat întrebarea: cum poate un sistem de învățare să fie atât stabil cât și plastic?

Dilema a fost abordată de Carpenter și Grossberg în [13]. Teoria rezonanței adaptive (Adaptive Resonance Theory, ART) dezvoltată de cei doi autori este unul din răspunsurile concrete date dilemei. De asemenea, sistemul prezintă abilitate de învățare incrementală și mare parte din proprietățile dezirabile ale sistemelor instruibile.

9.4 Fuzzy ARTMAP

Familia Fuzzy ARTMAP de rețele neurale (FAM) este cunoscută ca una din puținele care posedă capacitate de învățare incrementală, rezolvă dilema stabilitate-plasticitate și are multe din proprietățile dorite pentru un sistem instruibil.

Carpenter și Grossberg au fost interesați de obținerea de sisteme care se pot organiza singure. Paradigma ART poate fi descrisă ca un tip de grupare incrementală a datelor, având posibilitatea de a învăța fără antrenare supervizată și este de asemenea în acord cu modelele cognitive și de comportament. Folosește învățare nesupervizată; rețeaua este capabilă să găsească automat categoria asociată intrării curente sau să creeze una nouă atunci când este nevoie: numărul de neuroni din rețea nu este fixat aprioric.

Rețelele neurale Fuzzy ART sunt capabile să producă rapid o învățare stabilă a unor categorii de semnale ca răspuns la secvențe arbitrare de intrări binare sau continue. Fuzzy ART încorporează operatori din teoria mulțimilor fuzzy.

Sistemele de tip Fuzzy ARTMAP învață în mod autonom să clasifice vectori arbitrar de mulți, prezentați într-o ordine oarecare în categorii de recunoaștere create în funcție de succesul de predicție. Acest sistem de învățare supervizată este construit dintr-o pereche de module ART capabile de auto-organizare și obținere de categorii de recunoaștere stabile.

Succesul rețelelor bazate pe teoria rezonanței adaptive este dat de avantajele pe care le au față de alte rețele multistrat dezvoltate anterior:

- permite crearea dinamică a nodurilor (neuronilor) fără distrugerea celor existente;
- necesită mai puține cicluri de antrenare cerute, se poate folosi chiar cu învățare incrementală;
- are convergență garantată datorită utilizării unor ponderi mărginite și monotone.

Fuzzy ARTMAP este utilizabil pentru probleme de clasificare, estimare de probabilitate și regresie (aproximări de funcții). S-a demonstrat că FAM este aproximativ universal. Deoarece atât clasificarea cât și estimarea de probabilitate sunt cazuri particulare ale aproximărilor de funcții, este evident acum că FAM poate fi utilizat în orice problemă ce presupune stabilirea de legături dintre două submulțimi din \mathbb{R}^n și respectiv din \mathbb{R}^m .

În final, mai precizăm că FAM mai are o virtute: reprezentarea vectorilor învățați prin categorii facilitează extragerea de reguli sub forma de relații, aspect esențial în domeniul extragerii de cunoștințe din date.

9.4.1 Arhitectura rețelei FAM

O rețea FAM constă într-o pereche de module ART notate ART_a și ART_b , conectate printr-un modul numit Mapfield, notat F^{ab} . ART_a și ART_b sunt folosite pentru codificarea vectorilor de intrare și respectiv de ieșire, iar Mapfield permite asocierea între intrări și ieșiri. Figura 9.1 conține componentele unei arhitecturi FAM.

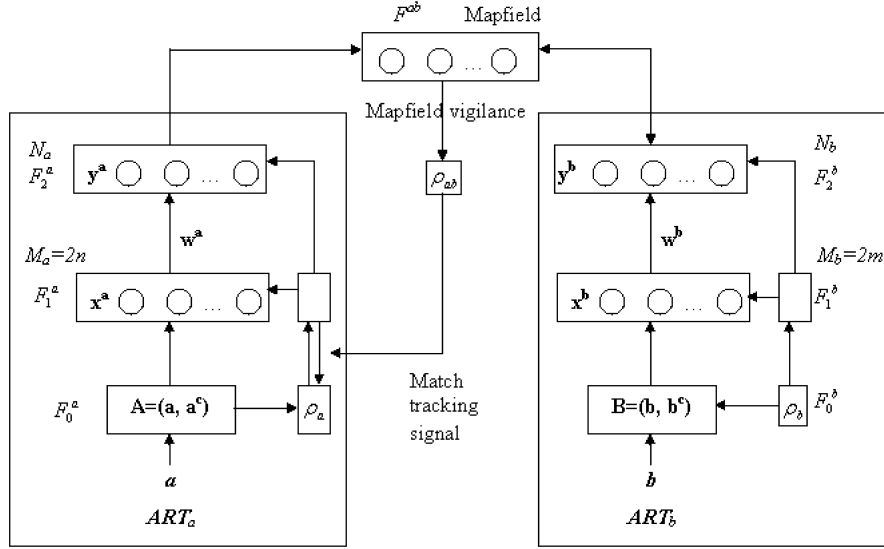


Figura 9.1: Arhitectura Fuzzy ARTMAP

Modulul Fuzzy ART_a conține stratul de intrare F_1^a și stratul competitiv F_2^a . Se adaugă de asemenea un strat de preprocesare F_0^a înaintea lui F_1^a . Straturi echivalente apar în ART_b .

Vectorii de intrare inițiali sunt dați sub forma:

$$\mathbf{a} = (a_1, \dots, a_n), \quad a_i \in [0, 1] \quad i = 1 \dots n \quad (9.1)$$

În cazul în care datele inițiale nu sunt din intervalul $[0, 1]$, se poate aplica o scalare:

$$a_i^p \rightarrow \frac{a_i^p - MIN}{MAX - MIN}, \quad i = 1, \dots, n$$

pentru fiecare vector de intrare $\mathbf{a}_p = (a_1^p, \dots, a_n^p)$, $1 \leq p \leq P$, P fiind numărul de vectori din setul de instruire, iar MIN și MAX sunt valoarea minimă și respectiv maximă din vectorii de intrare:

$$MIN(MAX) = \min(\max) \{a_i^p\}, \quad i = 1, \dots, n, p = 1, \dots, P$$

sau se poate lua un minorant (respectiv majorant) al valorilor de intrare.

O tehnică de preprocesare numită *codificare complementară* este efectuată în cele două module fuzzy ART de către stratul F_0^a , respectiv F_0^b pentru a evita proliferarea nodurilor. S-a dovedit geometric că fără codificarea complementară se vor produce numeroase categorii grupate lângă origine, fără a crea altele care să le înlocuiască. Codificarea complementară este utilizată pentru a obține vectori normalizați, adică vectori cu normă constantă:

$$|\mathbf{A}| = \text{const} \quad (9.2)$$

unde $|\cdot|$ este o funcție normă. În cazul nostru $|\cdot|$ reprezintă norma L_1 : pentru un vector $\mathbf{x} = (x_1, \dots, x_k)$

$$L_1(\mathbf{x}) = |\mathbf{x}| = \sum_{i=1}^k |x_i| \quad (9.3)$$

Fiecare vector de intrare $\mathbf{a} = (a_1, \dots, a_n)$ produce vectorul:

$$\mathbf{A} = (\mathbf{a}, \mathbf{a}^c) = (\mathbf{a}, \mathbf{1} - \mathbf{a}) = (a_1, \dots, a_n, 1 - a_1, \dots, 1 - a_n) \quad (9.4)$$

a cărui normă este:

$$|\mathbf{A}| = \sum_{i=1}^n a_i + \sum_{i=1}^n (1 - a_i) = n = \text{constant} \quad (9.5)$$

motiv pentru care spunem că vectorul \mathbf{A} este normalizat.

Pentru ART_a folosim următoarele notații: M_a este numărul de noduri în F_1^a și N_a este numărul de noduri din F_2^a . Datorită pasului de preprocesare, $M_a = 2n$. Fiecare nod F_2^a reprezintă un grup de intrări similare (numit în alte contexte *cluster*); vom folosi termenul “categorie” pentru a ne referi la un nod F_2^a . Fiecare categorie F_2^a are propriul set de ponderi adaptive stocate sub forma unui vector:

$$\mathbf{w}_j^a = (w_{j,1}^a, \dots, w_{j,M_a}^a), \quad j = 1, \dots, N_a. \quad (9.6)$$

Aceste ponderi formează memoria pe termen lung a sistemului. Inițial, toți vectorii au valorile:

$$w_{ji}^a = 1, \quad j = 1, \dots, N_a, \quad i = 1, \dots, M_a \quad (9.7)$$

Spunem că un nod din F_2^a este *necomis* dacă nu a învățat încă nici un vector de intrare, *comis* în caz contrar. Modulul ART_a este responsabil cu crearea grupărilor de vectori de intrare. În timpul etapei de învățare, N_a este numărul de noduri (categorii) comise. Notății și afirmații similare se folosesc pentru ART_b , care primește vectori m -dimensionali. Pentru o problemă de clasificare, adică o problemă pentru care numărul total de clase de ieșire este

aprioric cunoscut, indexul de clasă este același cu indexul de categorie din F_2^b și astfel ART_b poate fi substituit de un vector N_b -dimensional.

Modulul Mapfield permite FAM să creeze legături între cele două module ART , stabilind legături de tip mulți-la-unu între categorii din ART_a și ART_b . Numărul de noduri din F^{ab} este egal cu numărul de noduri din F_2^b . Fiecare nod j din F_2^a este legat cu fiecare nod din F_2^b via un vector de ponderi \mathbf{w}_j^{ab} , unde \mathbf{w}_j^{ab} este a j -a linie dintr-o matrice \mathbf{w}^{ab} ($j = 1, \dots, N_a$). Toate ponderile din \mathbf{w}^{ab} sunt inițializate cu 1:

$$w_{jk}^{ab} = 1, \quad j = 1, \dots, N_a, \quad k = 1, \dots, N_b \quad (9.8)$$

9.4.2 Algoritmul de învățare pentru FAM

În următorul algoritm, operatorul \wedge este așa numitul operator fuzzy AND definit ca:

$$(\mathbf{p} \wedge \mathbf{q})_i = \min(p_i, q_i), \quad i = 1, \dots, k \quad (9.9)$$

unde

$$\mathbf{p} = (p_1, \dots, p_k), \quad \mathbf{q} = (q_1, \dots, q_k) \quad (9.10)$$

1. Se setează parametrul de factor de vigilență ρ_a la o valoare egală cu o valoare de bază prestabilită: $\rho_a = \bar{\rho}_a \in [0, 1]$ și se consideră că toate categoriile din F_2^a sunt neinhibate – adică fiecare nod participă la căutarea unei categorii adecvate pentru vectorul de intrare curent;
2. Pentru fiecare vector de intrare preprocesat \mathbf{A} , o funcție fuzzy este folosită pentru a obține un răspuns de la fiecare categorie F_2^a :

$$T_j(\mathbf{A}) = \frac{|\mathbf{A} \wedge \mathbf{w}_j^a|}{\alpha_a + |\mathbf{w}_j^a|}, \quad j = 1, \dots, N_a \quad (9.11)$$

3. Fie J indicele de nod neinhibat care dă cea mai mare valoare calculată precum în (9.11):

$$J = \arg \max \{T_j | j = 1, \dots, N_a \text{ și nodul } j \text{ nu este inhibat}\} \quad (9.12)$$

4. Verifică condiția de rezonanță, i.e. dacă intrarea este suficient de similară cu prototipul câștigătorului:

$$\frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} \geq \rho_a \quad (9.13)$$

Dacă condiția este îndeplinită, atunci mergi la pasul 5, altfel inhibă nodul J astfel încât el nu va mai participa la competiția pentru vectorul curent. Dacă există noduri neinhibate, atunci mergi la pasul 3, altfel recrutează o nouă categorie (creează un nou nod în F_2^a) pentru a reprezenta vectorul de intrare și fie J indicele acestui nou nod.

5. Un proces similar se desfășoară și în ART_b . Fie K indicele nodului câștigător din ART_b . Vectorul de ieșire F_2^b este setat la:

$$y_k^b = \begin{cases} 1, & \text{pentru } k = K \\ 0, & \text{altfel} \end{cases} \quad k = 1, \dots, N_b \quad (9.14)$$

În Mapfield se formează vector de ieșire \mathbf{x}^{ab} :

$$\mathbf{x}^{ab} = \mathbf{y}^b \wedge \mathbf{w}_J^{ab} \quad (9.15)$$

6. Un test de verificare în Mapfield controlează potrivirea dintre valoarea prezisă \mathbf{x}^{ab} și vectorul de ieșire atașat vectorului de instruire curent \mathbf{y}^b :

$$\frac{|\mathbf{x}^{ab}|}{|\mathbf{y}^b|} \geq \rho_{ab} \quad (9.16)$$

unde $\rho_{ab} \in [0, 1]$ este un parametru de vigilență Mapfield. Dacă testul din ecuația (9.16) este trecut, atunci se face învățare ART_a , ART_b și Mapfield (pasul 7). Altfel, se inițiază o secvență de pași numită *match tracking* (pasul 8).

7. În modulele fuzzy ART și în Mapfield se efectuează învățare:

$$\mathbf{w}_J^{a(new)} = \beta_a (\mathbf{A} \wedge \mathbf{w}_J^{a(old)}) + (1 - \beta_a) \mathbf{w}_J^{a(old)} \quad (9.17)$$

(și analog în ART_b) și

$$w_{Jk}^{ab} = \begin{cases} 1, & \text{pentru } k = K \\ 0, & \text{pentru } k \neq K \end{cases} \quad (9.18)$$

Se merge la pasul 9.

8. Faza de match tracking, în care se intră doar dacă inecuația (9.16) nu e în deplină: mărește ρ_a :

$$\rho_a = \frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} + \delta \quad (9.19)$$

unde $0 < \delta < 1$. Dacă $\rho_a > 1$ atunci vectorul curent este rejectat, altfel mergi la pasul 3.

9. Dacă mai sunt vectori în setul de învățare, mergi la pasul 1, altfel STOP.

Urmează câteva comentarii privind pașii de mai sus:

1. La pasul 2, fiecare vector este preprocesat datorită straturilor F_0^a și respectiv F_0^b . $\alpha_a > 0$ este un parametru de alegere. Pentru doi vectori \mathbf{p} și \mathbf{q} , raportul:

$$r = \frac{|\mathbf{p} \wedge \mathbf{q}|}{|\mathbf{q}|} \quad (9.20)$$

cu $0 \leq r \leq 1$ dă gradul în care \mathbf{p} este subset fuzzy al lui \mathbf{q} și deci pentru $0 < \alpha_a \ll 1$, $T_j(\mathbf{A})$ din ecuația 9.11 măsoară gradul în care \mathbf{A} este o submulțime fuzzy a categoriei w_j^a . Dacă se crește valoarea lui α_a atunci se va mări numărul de categorii, lucru nu întotdeauna benefic. Este deci sugerat ca să se mențină α_a la o valoare mică, de exemplu $\alpha_a = 0.001$; valori mai mici ale lui α_a nu duc la o diferență semnificativă.

2. La pasul 3, dacă există mai multe categorii ale căror funcție de alegere atinge maximul, vom considera pe cea categorie care are indicele minim.
3. Parametrul ρ_a calibrează încrederea minimă pe care ART_a trebuie să aibă vizavi de o categorie activată de o intrare pentru ca ART_a să accepte această categorie, în loc de a căuta o categorie mai bună. Valori mici ρ_a duc la un grad mare de generalizare și un număr mai mic de categorii ART_a .
4. Dacă inecuația (9.13) este îndeplinită, spunem că avem rezonanță în ART_a pentru vectorul de intrare curent. Datorită pasului de preprocesare, conform (9.5), numitorul din (9.13) este exact dimensiunea originară a vectorilor de intrare, n .
5. Aceiași pași ca în 1 – 4 sunt efectuați în paralel pentru modulul ART_b , dacă nu cumva acesta este substituit cu un vector N_b —dimensional; în acest din urmă caz indicele nodului câștigător K este indicele de clasă corespunzător intrării curente;
6. Când se intră în pasul 5, avem rezonanță atât în ART_a cât și în ART_b . Vectorul \mathbf{x}^{ab} dă activarea Mapfield și se folosește atât când ambele module F_2^a și F_2^b sunt active, *i.e.* la faza de învățare, cât și când F_2^a este activ și F_2^b e inactiv (faza de predicție). În faza de învățare \mathbf{x}^{ab} are forma din ecuația (9.15); în faza de predicție acest vector este calculat ca:

$$\mathbf{x}^{ab} = \mathbf{w}_J^{ab} \quad (9.21)$$

7. Atunci când ambele module F_2^a și F_2^b sunt active, a J -a categorie câștigătoare din ART_a va corespunde unui vector de ponderi \mathbf{w}_J^{ab} din Mapfield care leagă nodul F_2^a cu categoria F_2^b prezisă. În paralel, pentru modulul ART_b am obținut vectorul de ieșire \mathbf{y}^b ca în ecuația (9.14). Operația fuzzy AND ne asigură că \mathbf{x}^{ab} nu e plin cu valoarea

zero dacă și numai dacă valoarea de ieșire prezisă și cea actuală coincid. Atunci când categoria J este necomisă avem:

$$\mathbf{w}_J^{ab} = (1, 1, \dots, 1) \quad (9.22)$$

și deci $\mathbf{x}^{ab} = \mathbf{y}^b$. Când doar modulul F_2^a este activ, matricea \mathbf{w}^{ab} dă valoarea prezisă: indicele categoriei din ART_b asociată cu intrarea curentă este unica poziție k din linia j a matricei \mathbf{w}^{ab} pentru care $w_{jk}^{ab} = 1$.

Ecuția (9.18) indică faptul că al J -lea nod din ART_a este legat cu categoria de ieșire K , iar legătura, odată făcută, nu se mai schimbă.

8. Pentru β_a (Pasul 7), există două moduri de învățare:
 - (a) *fast learning* corespunde la a seta $\beta_a = 1$ atât pentru nodurile comise cât și pentru cele necomise;
 - (b) *fast-commit and slow-recode learning* corespunde la a seta $\beta_a = 1$ pentru nod necomis și $\beta_a < 1$ pentru cele comise.
9. La faza de match tracking, datorită creșterii valorii lui ρ_a conform ecuației (9.19), nodul J nu va mai fi în stare să câștige în competițiile următoare pentru vectorul de intrare curent. Match tracking-ul va declanșa o nouă căutare în ART_a pentru a găsi un alt nod câștigător. Se poate ca asta să ducă la crearea unui nou nod în ART_a . Căutarea se repetă până când $\rho_a > 1$, sau până când se găsește o categorie adecvată în ART_a .
10. O valoare mare a lui δ în pasul 8 va crește numărul de categorii. Se folosește de regulă o valoare mică $\delta = 0.001$
11. Ordinea de prezentare a vectorilor din setul de instruire influențează comportamentul rețelei, adică numărul și pozițiile categoriilor va diferi. Putem face antrenarea în paralel cu diferite permutări ale setului de intrare, apoi se contorizează voturile pentru fiecare vector care trebuie clasificat.

Pasul de învățare este repetat până când nu mai este nicio eroare pentru setul de învățare, sau până se atinge o eroare acceptabilă; dacă se vrea învățare incrementală atunci se face o singură trecere pe setul de antrenare. După învățare, FAM poate fi utilizat pentru predicție. În această fază, stratul F_2^b este inactiv și F_2^a este activ. Conform ecuației (9.21), predicția este făcută doar pe baza lui \mathbf{w}_j^{ab} .

Există o interpretare geometrică interesantă a categoriilor ART_a : fiecare categorie j se reprezintă ca un hiper-dreptunghi R_j , deoarece vectorul de ponderi poate fi scris:

$$\mathbf{w}_j^a = (\mathbf{u}_j, \mathbf{v}_j^c) \quad (9.23)$$

unde

$$\mathbf{v}_j^c = (v_{j1}^c, \dots, v_{jn}^c) \quad (9.24)$$

(atât \mathbf{u} cât și \mathbf{v} au n elemente). Vectorul \mathbf{u}_j definește un colț al hiperdreptunghiului R_j iar \mathbf{v}_j este colț diagonal opus lui. Pentru $n = 2$, reprezentarea grafică este dată în figura 9.2. Dimensiunea lui R_j este definită ca:

$$|R_j| = |\mathbf{v}_j - \mathbf{u}_j| \quad (9.25)$$

În modulul ART_a , atunci când se creează un nou nod în pasul 4, acesta va fi de fapt vectorul de intrare preprocesat curent: $\mathbf{w}_j^{(new)} = \mathbf{A} = (\mathbf{a}, \mathbf{a}^c)$; cu alte cuvinte, $R_j^{(new)}$ este de fapt un punct reprezentând vectorul de intrare preprocesat A . În timpul fiecărui pas de învățare fast-learning ($\beta_a = 1$), R_j se expandează la $R_j \oplus \mathbf{a}$, hiperdreptunghiul minim care conține R_j și \mathbf{a} – vezi figura 9.3; dacă punctul \mathbf{A} este chiar în interiorul lui R_j , atunci R_j nu se modifică. Colțurile lui $R_j \oplus \mathbf{a}$ sunt definite de $\mathbf{a} \wedge \mathbf{u}_j$ și $\mathbf{a} \vee \mathbf{v}_j$, unde operatorul \vee este definit ca operatorul fuzzy OR:

$$(\mathbf{p} \vee \mathbf{q})_i = \max(p_i, q_i), \quad i = 1 \dots, n \quad (9.26)$$

pentru \mathbf{p} și \mathbf{q} ca în ecuația (9.10).

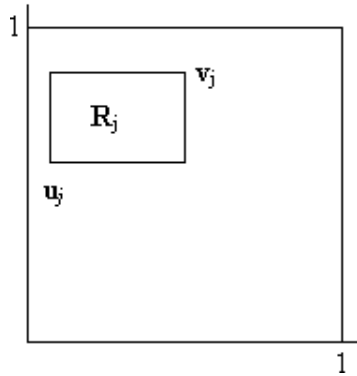


Figura 9.2: Fiecare vector pondere \mathbf{w}_j are o interpretare geometrică precum un hiperdreptunghi R_j cu colțurile definite de \mathbf{u}_j și \mathbf{v}_j

Se poate arăta că:

$$|R_j| = n - |\mathbf{w}_j| \quad (9.27)$$

iar hiperdreptunghiul corespunzător unei categorii nu poate crește oricât de mult:

$$|R_j| \leq (1 - \rho)n \quad (9.28)$$

unde prin ρ ne referim la ρ_a sau ρ_b , în funcție de modulul vizat.

Aceste proprietăți sunt sumarizate de teorema:

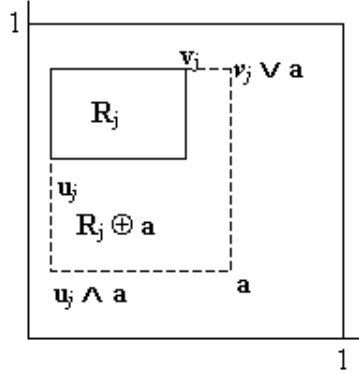


Figura 9.3: Expandarea de categorie în timpul fast learning: de la R_j la hiper-dreptunghiul mai mare conținând R_j și \mathbf{a} .

Teorema 4. [18] *Un sistem Fuzzy ART cu codificarea complementară, fast learning și termen de vigilență constant formează categorii hiper-dreptunghiuri care converg în limită la o secvență arbitrară de vectori analogici sau binari. Hiper-dreptunghiurile cresc monoton în toate dimensiunile. Dimensiunea $|R_j|$ a unui hiper-dreptunghi este $n - |\mathbf{w}_j|$, unde \mathbf{w}_j este vectorul pondere corespunzător. Dimensiunea $|R_j|$ este mărginită superior de $n(1 - \rho)$. Dacă $0 \leq \rho < 1$, numărul de categorii este mărginit, chiar dacă numărul de exemple din setul de antrenare este infinit. Proprietăți similare au loc pentru fast-learn, slow-recode, exceptând cazul în care este nevoie de prezentări repetate ale fiecărei intrări înainte de stabilizarea sistemului.*

Sumarizând: FAM aplică o învățare bazată pe potrivire, conform căreia vectorii sunt grupați în categorii pe baza unor măsuri de similaritate. Dacă un vector nu se potrivește suficient de bine cu o categorie existentă, atunci se va crea una nouă pentru a o reprezenta. Datorită acestui comportament, FAM nu încearcă să minimizeze o funcție de cost, evitând problemele întâlnite în optimizarea funcțiilor. Această strategie de învățare este opusă celei bazate pe minimizarea erorii, care cere de regulă reantrenarea dacă valoarea erorii este inacceptabilă sau dacă niște vectori mult diferiți de cei învățați anterior sunt prezentați rețelei.

Capitolul 10

Hărți cu auto-organizare

10.1 Încadrare

Harta cu autoorganizare este o rețea neurală utilizată pentru clustering, proiecție din spații multidimensionale în spațiu bidimensional și vizualizare. Pentru instruire se folosesc date ne-etichetate, deci vorbim de instruire ne-supervizată¹

Harta cu auto-organizare (sau harta Kohonen) este creația lui Teuvo Kohonen [9] și se consideră că e un model plauzibil din punct de vedere biologic. Pentru o excelentă prezentare a acestui model și a legăturii cu modelul biologic al neocortexului, a se vedea [10].

Din punct de vedere matematic, lema Johnson–Lindenstrauss (vezi secțiunea 10.3) arată că o reprezentare prin mai puține dimensiuni cu păstrarea aproximativă a relațiilor de vecinătate este posibilă.

Menționăm că în același scop – reprezentarea în spații cu mai puține dimensiuni decât cel din care provin datele inițiale, dar cu păstrarea relațiilor de vecinătate – se mai folosesc tehnici precum Sammon mapping (sau proiecția Sammon)², t-distributed stochastic neighbor embedding³, autoencoders⁴.

Self Organizing Map (SOM) se folosește de o funcție de vecinătate pentru a păstra proprietățile topologice ale spațiului de intrare. Ideea de bază este abdicarea de la principiul *câștigătorul ia tot* (secțiunea 4.6.6, pagina 54) și instruirea atât a neuronului câștigător, cât și a neuronilor din vecinătatea sa. Efectul este obținerea de zone de neuroni care răspund unor semnale similare.

¹O etapă opțională de etichetare poate face uz de clasele asignate datelor de intrare.

²J.W. Sammon, “A nonlinear mapping for data structure analysis”, IEEE Transactions on Computers nr. 18: pp. 401–409, 1969.

³L.J.P. van der Maaten și G. Hinton, “Visualizing High-Dimensional Data Using t-SNE”, Journal of Machine Learning Research no. 9, 2008

⁴G. Hinton și R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, Science 313 (5786):504–507, 2006.

10.2 Intrări, arhitectură și algoritm de instruire

Expunerea din această secțiune este preluată după [5], capitolul 9.

Datele de intrare sunt de tip numeric, fără valori lipsă, din spațiul n dimensional (n mare). Prin învățare, datele sunt proiectate pe o hartă – de regulă bidimensională – de neuroni.

Fiecare neuron are două atribute:

- un vector numeric de ponderi, de aceeași dimensiune ca și spațiul de intrare; neuronul j are vectorul $\mathbf{w}_j = (w_{j1}, \dots, w_{jn})^t$;
- poziția lui în hartă, de regulă dată ca o pereche de indici cu coordonate naturale.

Ponderile se ajustează prin instruire. Neuronii au relații de vecinătate cu alți neuroni. Vecinătatea se poate considera rectangulară sau hexagonală (precum în figura 10.2) sau Gaussiană (eq. (10.3)).

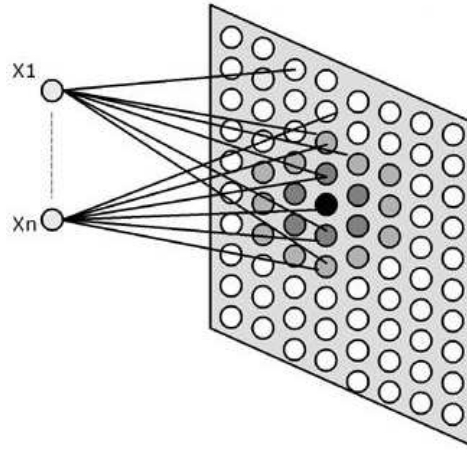


Figura 10.1: Proiectarea unui vector n -dimensional pe harta de neuroni

Inițial, ponderile din harta de neuroni sunt setate la valori aleatoare (cu toate că există și strategii de pre-antrenare care setează valori mai adecvate pentru ponderi). Pentru un vector de intrare \mathbf{x} folosim notația $i(\mathbf{x})$ pentru indicele neuronului care este cel mai apropiat de \mathbf{x} :

$$i(\mathbf{x}) = \arg \min_j \|\mathbf{x} - \mathbf{w}_j\| \quad (10.1)$$

unde indicele j este considerat peste toată mulțimea de indici de neuroni din SOM. Acesta este neuronul câștigător.

După ce acest neuron este descoperit, se face o modificare a ponderilor lui și a vecinilor din hartă. Vecinii j ai unui neuron i sunt dați de distanța $d_{i,j}$

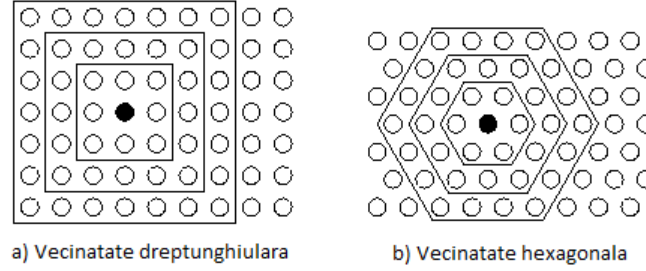


Figura 10.2: Modele de vecinătăți între neuronii din harta SOM. Ariile de valoare descrescătoare arată cum evoluează vecinătatea neuronului câștigător în decursul instruirii.

dintre pozițiile celor doi neuroni în hartă, dar vecinătatea este dependentă și de timp (iterația t curentă din algoritmul de instruire, vezi mai jos).

Un exemplu pentru funcție $d_{j,i}$ este:

$$d_{j,i} = \|\mathbf{r}_j - \mathbf{r}_i\| \quad (10.2)$$

unde \mathbf{r}_k este poziția în hartă a neuronului k , iar $\|\cdot\|$ este distanța Euclidiană. Pentru o hartă de neuroni bidimensională \mathbf{r}_k este o pereche de numere naturale; pentru o hartă unidimensională (neuroni ordonați pe o curbă) putem lua $d_{j,i} = |j - i|$, cu i și j numerele de ordine ale neuronilor.

Se introduce o funcție de vecinătate $h_{i,j}(t, d_{i,j})$, cu proprietățile:

1. $h_{i,j}$ este maximizată în cazul $d_{i,j} = 0$;
2. $h_{i,j}$ scade monoton cu scăderea $d_{i,j}$; $\lim_{d_{i,j} \rightarrow \infty} h_{i,j}(t, d_{i,j}) = 0$;

Un exemplu de funcție h este

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right) \quad (10.3)$$

De notat că această funcție de vecinătate este considerată mai plauzibilă din punct de vedere biologic decât cele din figura 10.2. Cantitatea σ este frecvent luată ca descrescătoare în timp, un exemplu fiind:

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right) \quad (10.4)$$

unde t este numărul iterației, iar τ_1 este un hiperparametru ales de către experimentator.

Agregând cele de mai sus, obținem o alegere populară pentru funcția de vecinătate h :

$$h_{j,i(\mathbf{x})}(t) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(t)}\right) \quad (10.5)$$

Conceptul de vecinătate este utilizat în cadrul algoritmului de instruire. În decursul instruirii, pentru vectorul curent $\mathbf{x} \in \mathbb{R}^n$ se consideră neuronul câștigător $i(\mathbf{x})$, precum și vecinii săi. Toți acești neuroni își vor modifica ponderile astfel încât acestea să devină mai apropiate (în sensul distanței Euclidiene) de vectorul \mathbf{x} . Formula următoare realizează acest lucru:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) \cdot h_{j,i(\mathbf{x})}(t) \cdot (\mathbf{x} - \mathbf{w}_j(t)) \quad (10.6)$$

unde $t = 0, 1, \dots$ este moment de timp, iar $\eta(t)$ este rata de învățare, pozitivă și descrescătoare în timp. Pentru $\eta(t)$ se consideră de exemplu:

$$\eta(t) = \eta_0 \exp\left(-\frac{t}{\tau_2}\right) \quad (10.7)$$

unde τ_2 este un alt hiperparametru al algoritmului.

Algoritmul de instruire este următorul:

1. **Pas 1: inițializare.** Fiecare neuron din hartă este are vectorul de ponderi inițializat cu valori aleatoare și mici. Se inițializează $t = 1$
2. **Pas 2: alegere pattern de instruire.** $t = t + 1$; se alege un vector \mathbf{x} din setul de instruire.
3. **Pas 3: găsirea celui mai apropiat neuron.** Se determină neuronul $i(\mathbf{x})$:

$$i(\mathbf{x}) = \arg \min_j \|\mathbf{x} - \mathbf{w}_j\|$$

4. **Pas 4: ajustarea ponderilor.** Neuronul cel mai apropiat și vecinii lui își modifică ponderile:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) \cdot h_{j,i(\mathbf{x})}(t) \cdot (\mathbf{x} - \mathbf{w}_j(t))$$

5. **Pas 5: repetare.** Se reia de la pasul 2 până când nu mai apar modificări notabile în ponderile neuronilor.

10.3 Lema Johnson–Lindenstrauss

Lema Johnson–Lindenstrauss afirmă că un set de puncte dintr-un spațiu cu număr mare de dimensiuni poate fi proiectat printr-o funcție liniară într-un spațiu cu mai puține dimensiuni, astfel încât distanțele dintre puncte să fie păstrate, cu un anumit grad de aproximare. Mai precis:

Teorema 5. (*Lema Johnson–Lindenstrauss*) Date fiind $0 < \varepsilon < 1$, o mulțime X de m puncte din \mathbb{R}^N și un număr $n > 8 \cdot \frac{\ln(m)}{\varepsilon^2}$, există o funcție liniară $f : \mathbb{R}^N \rightarrow \mathbb{R}^n$ astfel încât:

$$(1 - \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\|^2 \leq (1 + \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2 \quad (10.8)$$

pentru orice $\mathbf{u}, \mathbf{v} \in X$.

Deși minorantul pentru n este de regulă mult mai mare decât 2 (dimensiunea tradițională a unei hărți Kohonen), se arată principial că o reducere a dimensionalității datelor poate fi făcută. Lema este interesantă și pentru domenii conexe precum *compressive sensing* și *manifold learning*.

Capitolul 11

Calcul evoluționist

Calculul evoluționist este inspirat din teoria evoluției dezvoltate de către Charles Darwin și de genetică – știința eredității. Au în comun faptul că folosesc populații de elemente care sunt folosite pentru căutarea soluției unei probleme, spre deosebire de alte abordări care încercă îmbunătățirea printr-un proces iterativ a unei singure valori.

11.1 Taxonomie

Calculul evoluționist se împarte în:

1. algoritmi genetici
2. programare evoluționistă
3. strategii de evoluție
4. programare genetică

Domeniile enumerate au concepte comune; dintre toate, cele mai multe rezultate sunt în domeniul algoritmiilor genetici, dar la ora actuală există hibridizări ale acestor 4 arii.

Cel care este creditat ca fiind pionierul domeniului algoritmiilor genetici este John T. Holland de la Universitatea din Michigan. El a introdus conceptul de populație de indivizi care participă la căutarea unei soluții; de asemenea, a dat teorema schemelor. El a fost cel care a stabilit operațiile care trebuie să se aplice unei populații genetice - selecția, încrucișarea și mutația.

Programarea evoluționistă (avându-l ca pionier pe Larry J. Fogel) folosește ca operatori selecția celui mai potrivit individ și mutația, dar nu și încrucișarea. În timp ce algoritmiile genetice văd procesul evolutiv ca fiind aplicat pe o *populație de indivizi din aceeași specie*, programarea evoluționistă vede evoluția ca aplicându-se unei *populații de specii*. Fiecare element din populație este interpretat ca o specie întreagă.

Strategiile de evoluție au fost dezvoltate de Ingo Rechenberg și Hans-Paul Schwefel, care au experimentat diferite variante de mutație pentru rezolvarea unor probleme legate de optimizarea unor suprafețe aflate în contact cu un fluid. Mutațiile reprezentau perturbări ale unor stări, efectuând o căutare în vecinătate. Multiplele variante de perturbare au construit un întreg domeniu.

Programarea genetică (Richard Friedberg) a pornit cu coduri program de lungime fixă. Prin modificări efectuate în mod automat asupra acestor programe se dorea obținerea unor variante de cod optimizate. Esențiale sunt aici modul de reprezentare a acestor programe și funcțiile de măsurare a calității codului.

De cele mai multe ori, pentru o abordare dintr-unul din cele patru domenii se urmează pașii:

1. Inițializează populația
2. Calculează performanța fiecărui element din populație
3. Aplică un pas de selecție
4. Aplică operații precum încrucișarea sau mutația
5. Reia de la pasul 2 până când se îndeplinește o anumită condiție.

Diferența între domenii constă în detaliile fiecărui pas. Pașii sunt bazați pe alegeri de valori aleatoare, ceea ce dă de înțeles că rulări diferite pot duce la valori diferite. Totodată algoritmi nu garantează descoperirea unei valori optime. De cele mai multe ori, însă nu este nevoie să se cunoască exact optimul, ci o valoare suficient de bună. În practică, calculul evoluționist dă rezultate bune într-un timp rezonabil.

În cele ce urmează vom prezenta partea de algoritmi genetici.

11.2 Algoritmi genetici

Rolul mediului care apare ca factor modelator în teoria evoluționistă este preluat de către o funcție scop. Vom detalia algoritmul pentru maximizarea unei funcții $f : [a, b] \rightarrow R_+^*$. Indivizii care alcătuiesc populația se numesc *cromozomi* și sunt alcătuiți din *gene*.

Se pornește cu o populație inițială, care este supusă apoi unei secvențe de procese de tipul:

1. selecție: indivizii care sunt cei mai buni (considerând de valoarea funcției ce se vrea optimizată) sunt favorizați să apară de mai multe ori într-o populație nouă față de indivizii mai puțin performanți;

2. încrucișare: are loc un schimb de gene între perechi de părinți, formându-se copii; aceștia se presupune că moștenesc și combină performanțele părinților.
3. mutație: se efectuează niște modificări minore asupra materialului genetic existent.

Pas 1. Crearea unei populații inițiale de cromozomi. Se consideră mai multe valori pentru variabila $x \in [a, b]$. Numărul acestor valori – numit dimensiunea populației – este dat ca parametrul al algoritmului, n , dependent de problemă. Toate valorile sunt cuantificate prin cromozomi care sunt șiruri de k biți – un bit reprezintă în acest caz o genă a cromozomului, k fiind alt parametru de intrare.

Generarea celor n cromozomi se face aleator, prin setarea fiecărei gene la valoarea 0 sau 1, la întâmplare. Se obține astfel o populație inițială formată din cromozomii c_1, \dots, c_n .

Fiecare cromozom c (adică șir de k biți) va produce un număr $x(c)$ din intervalul $[a, b]$, astfel: dacă valoarea în baza 10 a cromozomului este $v(c)$ ($0 \leq v(c) \leq 2^k - 1$) atunci valoarea asociată din intervalul $[a, b]$ este:

$$x(c) = a + v(c) \cdot \frac{b - a}{2^k - 1} \in [a, b].$$

Pas 2. Evoluția populației. În acest pas se obțin generații succesive plecând de la populația inițială; populația de la generația $g + 1$ se obține pe baza populației de la generația g . Operatorii sunt selecția, împerecherea (crossover, încrucișarea) și mutația.

Pas 2.1. Selecția. Pentru fiecare cromozom c_i din populație se calculează funcția obiectiv $y_i = f(x(c_i))$, $1 \leq i \leq n$. Apoi se însușește valorile funcțiilor obiectiv obținute pentru fiecare cromozom în parte:

$$S = \sum_{i=1}^n y_i$$

Pentru fiecare din cei n cromozomi se calculează probabilitatea de selecție:

$$p_i = \frac{y_i}{S}, 1 \leq i \leq n$$

Pentru fiecare cromozom se calculează probabilitatea cumulativă de selecție:

$$q_j = \sum_{i=1}^j p_i, 1 \leq j \leq n$$

Remarcăm că se obține $0 < p_1 = q_1 < q_2 < \dots < q_n = 1$. Cu cât cromozomul c_i determină o valoare mai mare pentru funcția

f (i.e. cu cât valoarea $f(x(c_i))$ este mai mare), cu atât diferența dintre q_i și q_{i-1} este mai mare.

Se selectează n numere aleatoare uniform distribuite în $(0, 1]$. Pentru fiecare număr, dacă el se găsește în intervalul $(0, q_1]$ atunci cromozomul c_1 este ales și depus într-o populație nouă; dacă acest număr se află în intervalul $(q_i, q_{i+1}]$ atunci se alege cromozomul c_{i+1} . Remarcăm ca numărul de cromozomi prezenți în noua populație este tot n . Cu cât valoarea $y = f(x(c))$ asociată unui cromozom c este mai mare, cu atât cresc șansele lui spre a fi selectat și depus în noua populație. Este foarte probabil ca un astfel de cromozom valoros să apară de mai multe ori în populația nouă; de asemenea, este foarte probabil ca un cromozom cu o valoare mică pentru funcția f să nu apară deloc.

Pas 2.2. Încrucișarea (împerecherea, crossover) Pentru fiecare cromozom care a rezultat la pasul anterior se alege o valoare aleatoare, uniform distribuită în intervalul $(0, 1]$. Dacă această valoare este mai mică decât un parametru p_c (parametru al aplicației, e.g. 0.1), atunci cromozomul este ales pentru încrucișare. Se procedează astfel încât să se obțină un număr par de cromozomi (de exemplu se renunță la ultimul dacă numărul lor este impar). Cromozomii aleși se încrucisează astfel: primul selectat cu al doilea selectat, al 3-lea cu al 4-lea etc. Încrucișarea decurge astfel:

- se alege un număr aleator t între 1 și $k - 1$;
- se obțin 2 cromozomi copii astfel: primul va conține primele t gene ale primului părinte și ultimele $k - t$ gene ale celui de-al doilea părinte; al doilea copil conține primele t gene ale celui de-al doilea părinte și ultimele $k - t$ gene ale primului părinte;
- cei doi cromozomi copii îi vor înlocui în populație pe părinți

Pas 2.3. Mutația. Populației obținute i se aplică operator de mutație, astfel: pentru fiecare genă a fiecărui cromozom se alege o valoare aleatoare, uniform distribuită în $(0, 1]$; dacă acest număr este mai mic decât o probabilitate de mutație p_m (parametru al aplicației, e.g. $p_m = 0.01$), atunci se modifică valoarea curentă a genei cu complementul său față de 1.

Populația obținută în pasul 2 reia ciclul de evoluție. După ce se execută câteva astfel de evoluții (sau număr de generații, sau un timp alocat procesului este epuizat), se raportează valoarea celui mai bun cromozom din ultima generație¹.

¹Cei mai mulți practicieni folosesc strategia elitistă: se returnează cel mai bun individ al tuturor generațiilor.

Avantajul primar al algoritmilor genetici constă în schimbul de informație dintre indivizi realizat la etapa de încrucișare, adică schimbarea de blocuri de date care au evoluat. O utilizare eficientă a algoritmilor genetici presupune crearea unor structuri de date pentru gene și a unor operatori adecvați problemei ce trebuie rezolvată² – a se vedea secțiunea 11.4.

11.3 Fundamente teoretice

Studiul comportamentului algoritmilor genetici se face pe baza unor scheme (sau șabloane) care descriu colecții de cromozomi. O schemă se reprezintă ca un șir de caractere construit cu simbolurile “0”, “1” și “*”, unde “*” poate fi substituit cu orice bit; simbolul “*” poate apărea de ori-câte ori, inclusiv niciodată. De exemplu, schema (*0101) se potrivește cu doi cromozomi: (00101) și (10101)³. Dacă o schemă are l simboluri “*”, atunci ea poate să fie reprezentată de 2^l cromozomi, iar un cromozom de lungime k poate fi descris de $C_k^0 + C_k^1 + \dots + C_k^k = 2^k$ scheme.

Pentru o schemă S definim ordinul ei (și îl notăm cu $o(S)$) numărul de poziții pe care se află valorile 0 sau 1, adică numărul de poziții fixate. De exemplu, pentru schema $S = (* 0 * 1 1 0)$, $o(S) = 4$. Ordinul unei scheme dă gradul de specializare a ei și este utilă mai departe în calcularea probabilității de supraviețuire a sa în cadrul mutațiilor.

Lungimea unei scheme S , notată cu $\delta(S)$, este distanța dintre prima și ultima poziție fixată. Pentru schema dată mai sus, $\delta(S) = 6 - 2 = 4$ (sau $\delta(S) = 5 - 1 = 4$, după cum indicieră începe de la 1 sau de la 0). Noțiunea de lungime a unei scheme este utilă pentru calculul probabilității de supraviețuire a unei scheme în cadrul operațiilor de încrucișare.

Pentru o populație de indivizi aflată la momentul t al evoluției, vom nota cu $n(S, t)$ numărul de cromozomi din populație care reprezintă schema S . De asemenea, vom considera valoarea medie a schemei din populația de la un timp t , notată cu $f(S, t)$ și definită ca suma valorilor cromozomilor din populație care reprezintă schema S împărțită la numărul acestor cromozomi, $n(S, t)$.

La pasul de selecție, un cromozom A este copiat în populația următoare cu probabilitatea:

$$P(A) = \frac{f(A)}{\sum_{\text{cromozom } c} f(c)}$$

unde însumarea se face după toți cromozomii c ai populației curente. Rea-

²S-a stabilit “ecuația” Genetic Algorithms + Data Structures = Evolution Programs, [15].

³În acest caz spunem că schema este reprezentată de cei doi cromozomi.

mintind că n este numărul de cromozomi din populație, avem:

$$n(S, t + 1) = n(S, t) \cdot n \cdot \frac{f(S, t)}{\sum_{\text{cromozom } c} f(c)}$$

Cantitatea $\overline{f(t)} = \sum_{\text{cromozom } c} f(c)/n$ este chiar valoarea medie a populației de la momentul t , deci avem:

$$n(S, t + 1) = n(S, t) \cdot \frac{f(S, t)}{\overline{f(t)}}$$

Numărul de reprezentanți ai schemei S care vor exista la momentul $t + 1$ este dependent de valoarea schemei dată de cromozomii care există în populația de la momentul t . De exemplu, o schemă S care produce o valoare relativ mare a lui $f(S, t)$ față de $\overline{f(t)}$ va impune creșterea numărului de reprezentanți ai săi. Dacă presupunem de exemplu că $f(S, t) = \overline{f(t)} + \varepsilon \cdot \overline{f(t)} = \overline{f(t)}(1 + \varepsilon)$, $\forall t > 0$ (unde $\varepsilon > 0$) atunci se poate arăta prin inducție că:

$$n(S, t) = n(S, 0)(1 + \varepsilon)^t, \forall t \in \{1, 2, \dots\}$$

adică pentru scheme care au valoare medie desupra valorii medii a populației numărul de reprezentanți va crește exponențial în timp – respectiv dacă valoarea schemei este sub medie, numărul de reprezentanți obținuți prin selecție scade exponențial.

În ceea ce privește încrucișarea, să presupunem că cromozomul cu 7 gene $c = (1010100)$ este selectat pentru reproducere; există 2^7 scheme care îl au pe c drept reprezentant, de exemplu:

$$S_1 = (*01****)$$

și

$$S_2 = (1****0*)$$

Să presupunem că în procesul de încrucișare tăietura se face după a patra genă:

$$\begin{array}{rcl} c & = & (1 \ 0 \ 1 \ 0 \mid 1 \ 0 \ 0) \\ S_1 & = & (* \ 0 \ 1 \ * \mid * \ * \ *) \\ S_2 & = & (0 \ * \ * \ * \mid * \ 0 \ *) \end{array}$$

Se observă că pentru exemplul considerat schema S_1 sigur se va regăsi într-un descendent (deci schema supraviețuiește), deoarece valorile 0 și 1 se regăsesc pe pozițiile inițiale, în timp ce S_2 are șanse de a fi distrusă. Intuitiv, este clar faptul că lungimea mică a schemei S_1 mărește șansa de supraviețuire, față de S_2 care poate fi ușor “spart” în cromozomii copii. Desigur, poziția tăieturii este importantă.

Tăietura poate să apară uniform aleator (echipobabil) în $k - 1$ poziții. Probabilitatea de distrugere a unei scheme este:

$$P_d(S) = \frac{\delta(S)}{k - 1}$$

și evident probabilitatea evenimentului contrar, reprezentând supraviețuirea schemei este

$$P_s(S) = 1 - P_d(S) = 1 - \frac{\delta(S)}{k - 1}$$

Conform strategiei de alegere a cromozomilor ce se supun împerecherii, probabilitatea ca un cromozom să participe la încrucișare este p_c , deci probabilitatea de supraviețuire a unei scheme S este:

$$P_s(S) = 1 - p_c \cdot \frac{\delta(S)}{k - 1}$$

Se mai poate lua în considerare faptul că o schemă S poate totuși să supraviețuiască, dacă cromozomii care se încrucișează au pe pozițiile fixe ale schemei chiar valorile din S . Așa ceva este posibil și trebuie considerat ca măbind șansele de supraviețuire a unei scheme. Ca atare, șansa de supraviețuire este de fapt dată printr-o inegalitate:

$$P_s(S) \geq 1 - p_c \cdot \frac{\delta(S)}{k - 1}$$

deci schemele de lungime mică au șanse crescute de supraviețuire.

Combinând rezultatele obținute pentru partea de selecție și încrucișare, obținem:

$$n(S, t + 1) \geq n(S, t) \cdot \frac{f(S, t)}{f(t)} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k - 1} \right]$$

Mutația schimbă aleator biți din cromozom cu complementul lor. Este clar că pentru ca o schemă să supraviețuiască, pozițiile sale fixe nu trebuie să fie alese pentru mutație. Probabilitatea ca un singur bit să nu fie modificat este $(1 - p_m)$. Alegerile biților care să sufere mutație sunt evenimente independente, deci probabilitatea ca cei $o(S)$ biți fiși ai unei scheme să se mențină (și deci ca întreaga schemă să se mențină) este:

$$P_s(S) = (1 - p_m)^{o(S)}$$

Pentru că $p_m \ll 1$, putem aproxima $(1 - p_m)^{o(S)}$ cu $1 - p_m o(S)$. Am obținut că schemele cu ordinul mic au șanse crescute de supraviețuire.

Efectul combinat al operațiilor de selecție, încrucișare, mutație este deci:

$$n(S, t + 1) \geq n(S, t) \cdot \frac{f(S, t)}{f(t)} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k - 1} - p_m \cdot o(S) \right]$$

Se poate da acum enunțul teoremei schemelor, teorema fundamentală a algoritmilor genetici datorată lui Holland (1975):

Teorema 6 (Teorema schemelor). *Schemele scurte, de ordin mic, cu valoare peste medie cresc ca număr de reprezentanți în decursul generațiilor.*

S-a formulat următoarea ipoteză:

Ipoteza blocurilor de construcție, [15]. Un algoritm genetic execută un proces de căutare prin suprapunerea unor scheme scurte, de ordin mic și de valoare mare, numită blocuri de construcție. Se poate arăta că pentru o populație de n cromozomi, numărul de scheme efectiv procesate este în ordinul lui n^3 , ceea ce dă caracter de paralelism implicit al algoritmilor genetici: se procesează nu doar o singură schemă, ci mai multe.

11.4 Problema reprezentării datelor în algoritmii genetici

Reprezentarea indivizilor ca șiruri de biți este nenaturală pentru multe probleme practice. Să presupunem, de exemplu, problema comis-voiajorului: fiind date n orașe și distanțele dintre ele, să se determine un tur al lor, astfel încât fiecare oraș să fie vizitat exact o singură dată, să se revină la orașul de plecare iar costul total al drumului să fie minim⁴. O soluție este dată ca o permutare a mulțimii $\{1, \dots, n\}$.

Pentru cazul $n = 20$, dacă folosim reprezentarea binară, putem vedea că cinci biți sunt suficienți pentru a reprezenta orice număr de la 1 la 20, deci ar trebui să folosim $20 \cdot 5 = 100$ de biți pentru reprezentarea unei soluții potențiale. Să presupunem că la un moment dat avem grupul de 5 biți 01101 reprezentând orașul cu numărul 13; prin aplicarea mutației este posibil să se ajungă la valoarea binară 11101, adică în zecimal 29, un oraș care nu există. S-ar obține deci o valoare invalidă datorată unei reprezentări neadecvate a elementelor din problemă sau a unor operatori care nu sunt adaptați corespunzător. La fel de bine, se poate ca prin mutație sau încrucișare să se obțină valori de orașe repetate, deci un ciclu prematur.

Pentru cei 100 de biți asociați problemei, spațiul de căutare realizat este $2^{100} \simeq 10^{30}$, în timp ce mulțimea tuturor ciclurilor hamiltoniene este – considerând primul oraș ca fiind fixat și neconsiderând soluțiile simetrice de forma $A \rightarrow B \rightarrow C \rightarrow A \equiv A \rightarrow C \rightarrow B \rightarrow A$ – mulțimea permutărilor cu $19!/2 \approx 10^{17}$ elemente. În situația dată deducem că utilizarea unei codificări binare conduce la un spațiu de căutare mărit artificial, existând zone mari din spațiul binar care nu corespund unor soluții viabile.

Alte exemple de probleme aflate în aceeași situație pot fi încă date; se ajunge la concluzia că varianta naivă de reprezentare a valorilor și a operatorilor genetici nu se potrivește neapărat la orice problemă de căutare. Modelarea unui individ și a operatorilor asociați trebuie să se facă ținând

⁴În termeni de grafuri: se cere determinarea unui ciclu Hamiltonian de lungime minimă.

cont de domeniu și de particularitățile problemei. Vor fi exemplificate codificări adecvate pentru câteva probleme clasice.

O altă problemă care trebuie tratată este: cum procedăm când există constrângeri? De exemplu, dacă vrem să maximizăm funcția:

$$f(x, y) = x^2 - y^3 + 2 \cdot x \cdot \sin(y) \quad (11.1)$$

cu condiția ca variabilele x și y să satisfacă constrângerea:

$$1 \leq x^3 - \cos(y) + y^2 \leq 5 \quad (11.2)$$

cum încorporăm restricția în algoritmul genetic? Dacă folosim varianta clasică de codificare a unui individ, împreună cu operatorii de încrucișare și de mutație prezenți, cum asigurăm faptul că operatorii dați nu duc indivizii în zone în care constrângerea (11.2) nu este îndeplinită?

Pentru această din urmă problemă s-au dat următoarele variante:

1. impunerea de penalizări pentru indivizii care încalcă constrângerile;
2. implementarea unei metode de “reparare” a indivizilor care nu satisfac constrângerile;
3. implementarea unor operatori de încrucișare și de mutație care păstrează indivizii în condițiile impuse.

Pe marginea fiecăreia din cele trei variante există multiple versiuni:

- pentru penalizări, valoarea acestora poate fi constantă, sau să varieze cu gradul în care se încalcă constrângerile date; această ultimă variantă poate fi codificată sub forma unei funcții logaritmice, liniare, pătratice etc. O formă extremă de penalizare este eliminarea indivizilor care încalcă restricțiile, dar trebuie dat răspuns la întrebarea: cu ce se umple locul lăsat gol prin eliminare? sau cumva se permite populație de dimensiune variabilă? cei mai mulți autori afirmă că această eliminare este prea dură, în timp ce menținerea unor indivizi penalizați oferă variabilitate populației – se pot produce descendenți valizi, chiar și din cei care nu respectă constrângerile.
- pentru algoritmi de reparare – este posibil să se integreze cunoștințe din domeniu în metodele de corecție; trebuie zis însă că imaginarea unui algoritm de corecție poate uneori să fie o problemă la fel de grea ca și rezolvarea problemei de la care s-a plecat.
- pentru ultima variantă – este cunoscut deja că orice tip de date trebuie să vină cu un set de operatori dedicați care să permită prelucrarea tipurilor; o codificare potrivită problemei împreună cu operatorii asociați trebuie să favorizeze (ideal: să garanteze) generarea de indivizi valizi.

Aici se intervine cu cunoștințe despre problema care trebuie rezolvată, cunoștințe care, prin implementare, favorizează obținerea de indivizi care nu încalcă (prea mult, sau deloc) restricțiile;

Pentru fiecare din abordări s-au studiat variante și comportamente; studiul s-a făcut în mare măsură empiric, pe probleme concrete; la ora actuală, un rezultat precum teorema schemei este inexistent pentru alte codificări decât cea binară. Desigur, se poate folosi și o combinație a celor trei variante de mai sus.

Pentru numeroase probleme practice s-a constatat experimental că reprezentarea adecvată a indivizilor, împreună cu definirea unor operatori particularizați și cele 3 metode de mai sus dau rezultate mai bune decât aplicarea *ad literam* a algoritmului genetic peste o formă binarizată a problemei.

Vom exemplifica pentru problema discretă a rucsacului: se dă un rucsac de capacitate C , un set de n obiecte având greutatea $G_i > 0$ și valorile asociate $V_i > 0$, $1 \leq i \leq n$. Un obiect poate fi luat doar în întregime în rucsac; problema este: care sunt obiectele care trebuie încărcate, astfel încât greutatea totală să nu depășească C iar valoarea cumulată să fie maximă?

Problema este NP-completă, deci la ora actuală nu cunoaștem un algoritm de complexitate polinomială care să o rezolve. Totodată, menționăm că multe probleme pot fi reduse la aceasta, de aici interesul acordat.

O reprezentare naturală a unui individ – respectiv încărcare de rucsac – este un vector \mathbf{x} cu elementele x_i , $1 \leq i \leq n$, $x_i \in \{0, 1\}$, valoarea 0 însemnând că obiectul nu este luat, iar 1 – că e adăugat în rucsac. Se impune, evident, condiția de viabilitate a unui vector \mathbf{x} :

$$\sum_{i=1}^n x_i \cdot G_i \leq C$$

iar funcția de maximizat – numită și profit în acest caz – este:

$$P(\mathbf{x}) = \sum_{i=1}^n x_i \cdot V_i$$

11.4.1 Varianta cu penalizare

Pentru fiecare individ \mathbf{x} se va considera valoarea sa $val(\mathbf{x})$:

$$val(\mathbf{x}) = \sum_{i=1}^n x_i \cdot V_i - Pen(\mathbf{x})$$

unde $Pen(\cdot)$ este funcția de penalizare:

$$Pen(\mathbf{x}) \begin{cases} = 0, & \text{dacă } \mathbf{x} \text{ este viabil} \\ > 0, & \text{dacă } \mathbf{x} \text{ nu este viabil} \end{cases}$$

Dacă valoarea funcției de penalizare depinde de gradul în care se face încălcarea restricțiilor – gradul de încălcare poate fi de exemplu de diferența dintre $\sum_{i=1}^n x_i \cdot G_i$ și C – atunci se poate folosi o funcție de tip logaritm, liniar, pătratic, exponențial etc. Efectele alegerii unei asemenea funcții au fost analizate pe diferite situații; a se vedea [15] pentru rezultate experimentale și interpretarea lor.

11.4.2 Varianta cu reparare

Putem folosi aici tot codificarea binară. Algoritmul de corectare este simplu: dacă setul de obiecte ales depășește ca greutate totală capacitatea C , atunci se scot obiecte până când greutatea celor rămase devine acceptabilă (cel mult G). Vom transforma deci vectorul \mathbf{x} în $\mathbf{x}' = (x'_1, \dots, x'_n)$ astfel încât $\sum_{i=1}^n x'_i G_i \leq C$. Valoarea profitului, $P(\mathbf{x}')$, este:

$$P(\mathbf{x}') = \sum_{i=1}^n x'_i \cdot V_i$$

Algoritmul de reparare a unui individ este:

Listing 11.1: Repararea unui vector invalid

```
function reparare( $\mathbf{x}$ ,  $\mathbf{G}$ ,  $C$ ) returns a vector
begin
  rucsac-supraincarcat := false
   $\mathbf{x}' := \mathbf{x}$ 
  if  $\sum_{i=1}^n x'_i \cdot G_i > C$ 
    then rucsac-supraincarcat := true
  end if
  while rucsac-supraincarcat = true
     $i :=$  selecteaza un obiect din rucsac (#)
    scoate obiectul  $i$  din rucsac:  $x'_i := 0$ 
    if  $\sum_{i=1}^n x'_i \cdot G_i \leq C$ 
      then rucsac-supraincarcat := false
    end if
  end while
  return  $\mathbf{x}'$ 
end
```

În ce privește metoda de selectare a lui i din linia marcată cu (#), avem variantele:

- (reparare aleatoare) valoarea i se alege aleator din setul indicilor obiectelor care se găsesc în rucsac;
- (reparare greedy) se alege obiectul cel mai ușor, sau cel care are raportul P_i/G_i cel mai mic.

11.4.3 Codificarea adecvată a indivizilor

Vom prezenta o strategie de codificare a indivizilor, diferită de cea binară utilizată până acum, numită reprezentarea ordinală. Codificarea este larg utilizată și în alte probleme care presupun manipularea unor secvențe de valori, de exemplu în problema comis-voiajorului. Vectorul \mathbf{x} este cu cele n componente în baza 10, fiecare element x_i având proprietatea că $1 \leq x_i \leq n - i + 1, \forall i \in \{1, \dots, n\}$. De exemplu, pentru vectorul $\mathbf{x} = (4, 3, 4, 1, 1, 1)$ asociat liste de obiecte $L = (1, 2, 3, 4, 5, 6)$ decodificarea se obține astfel: se scoate elementul de indice $x_1 = 4$ din lista L , adică obiectul 4 și îl adăugăm în rucsac; L devine $(1, 2, 3, 5, 6)$; apoi se scoate elementul de indice $x_2 = 3$ (adică obiectul 3) din L și îl adăugăm în rucsac, L devine $(1, 2, 5, 6)$; se scoate elementul de indice $x_3 = 4$ din L , adică obiectul 6, L devine $(1, 2, 5)$ etc. Obținem astfel ordinea de depunere în rucsac: 4, 3, 6, 1, 2, 5. Fiecare cromozom codifică astfel o ordine de adăugare a obiectelor în rucsac. Adăugarea se face numai dacă nu duce la depășirea capacității rucsacului.

Se poate vedea că operație de încrucișare va duce întotdeauna la copii valizi, adică pentru un copil $\mathbf{z} = (z_1, \dots, z_n)$ avem că $1 \leq z_i \leq n - i + 1$ dacă și părinții au aceeași proprietate. Mutația este similară cu cea de la cazul binar: o componentă aleasă pentru mutație, fie ea x_i este modificată cu o valoare aleatoare uniform distribuită în mulțimea $\{1, \dots, n - i + 1\}$ diferită de x_i .

Listing 11.2: Utilizarea codificării ordinale

```
procedure decodificare( $\mathbf{x}$ )  
begin  
  construiește o lista  $L$  de obiecte  
  greutateTotala := 0  
  profitTotal := 0  
  for  $i = 1, n$   
     $j := x_i$   
     $o := L_j$   
    sterge elementul al  $j$ -lea din lista  $L$   
    if greutateTotala +  $G_o \leq C$   
      then begin  
        greutateTotala := greutateTotala +  $G_o$   
        profitTotal := profitTotal +  $P_o$   
      end  
    end if  
  end for  
end
```

Lista L se poate crea într-o ordine aleatoare, ca o permutare a mulțimii $\{1, \dots, n\}$. Indivizii rezultați — adică cei care realizează populația inițială — vor fi deci generați aleatori.

Rezultate experimentale pentru cele trei variante de rezolvare sunt date în [15]. Concluziile experimentelor însă nu pot fi generalizate la orice problemă care vine cu impunere de restricții. Se exemplifică însă că diferențele de performanță pot fi mari pentru aceste abordări.

11.5 Exemplu: problema orarului

Problema orarului este o altă situația practică care se abordează prin intermediul algoritmilor genetici. Problema este NP-completă. Are diferite enunțuri, vom prezenta varianta dată în [15].

Se dau următoarele:

- o mulțime de profesori $\{T_1, \dots, T_m\}$
- o listă de intervale de timp (ore) $\{H_1, \dots, H_n\}$
- o listă de săli de clasă $\{C_1, \dots, C_k\}$

Orarul *trebuie* să respecte următoarele cerințe:

- există un număr predefinit de ore pentru fiecare profesor și clasă;
- la un moment dat, la o clasă predă un singur profesor;
- un profesor nu poate predă la mai multe clase simultan;
- la fiecare clasă programată la o anumită oră trebuie să existe exact un profesor

Mai avem și constrângeri care *ar trebui* respectate; acestea sunt legate de:

- nicio “fereastră” în orarul elevilor, cât mai puține în cel al profesorilor;
- preferințe exprimate de profesori sau elevi: ore doar într-o anumită parte a zilei sau săptămânii;
- împărțire cât mai echilibrată a orelor;
- număr maxim de ore pe zi pentru elevi/profesori

Codificarea binară pentru această problemă, cu toate că este posibilă, poate apărea drept nenaturală; mai mult decât atât, există riscul ca spațiul de căutare să se mărească artificial, precum la problema comis voiajorului. Putem să codificăm un orar (un individ) ca fiind o matrice \mathbf{O} cu m linii și n coloane, unde liniile corespund profesorilor iar coloanele – orelor disponibile. Fiecare celulă este fie liberă, fie conține o clasă C_i , $1 \leq i \leq k$.

Pentru reprezentarea dată, operatorii genetici ar putea fi⁵:

⁵Dar nimic nu ne împiedică să concepem alți operatori.

- mutația de ordin k : se iau 2 secvențe adiacente formate din p elemente și se interschimbă
- mutația de zile: se iau două zile și se interschimbă între ele
- încrucișare: se pornește de la două orare părinte O_1 și O_2 , se efectuează tăietură pe orizontală sau pe verticală și se face interschimbarea de porțiuni, întocmai ca la cromozomii binari

Este posibil ca să fie nevoie să se intervină cu algoritmi de corecție după aplicarea unor astfel de operatori. Din punct de vedere practic, abordarea prin algoritmi genetici este confirmată ca o metodă funcțională de către mai mulți autori.

Capitolul 12

Mulțimi și logică fuzzy

12.1 Prezentare generală

Capitolul conține o introducere a logicii fuzzy și mulțimilor fuzzy¹. Domeniile vizează modelarea incertitudinii din lumea reală, raționamentul aproximativ, imprecizia în exprimare. Majoritatea conceptelor folosite în lumea reală sunt neclare, vagi, ambigue, dar cu toate acestea oamenii operează cu ele foarte bine.

Termenul de “fuzzy” a fost introdus de către Lotfi A. Zadeh, profesor la University of California at Berkley, în lucrarea sa “Fuzzy Sets” [14]. Mulțimile fuzzy – sau mulțimile nuanțate – se bazează pe conceptul de grad de apartenență a unui element la o mulțime; acest grad este un număr din intervalul $[0, 1]$, spre deosebire de mulțimile clasice care sunt văzute ca asignând grade de apartenență fie 0, fie 1². Într-o mulțime fuzzy, gradul de apartenență se exprimă printr-o funcție cu valori în intervalul $[0, 1]$.

Alături de teoria probabilităților, sistemele fuzzy sunt folosite pentru modelarea incertitudinii. Incertitudinea existentă în ceea ce privește rezultatul aruncării unui zar este modelată prin variabile aleatoare - urmărindu-se determinarea probabilităților asociate diferitelor valori, sau comportamentul obținut prin repetarea experimentelor etc. Tipul de incertitudine pe care îl abordează sistemele fuzzy este însă diferit. De exemplu, propoziția “Maria este destul de înaltă” nu are o incertitudine de tip statistic în ea: nu este vorba de evenimente aleatoare repetate sau condiționări probabiliste. Caracterul vag al unui sistem este o caracteristică intrinsecă a sa; ea nu este dată în vreun fel de observații repetate sau încrederea în legătura dintre o stare cunoscută și una posibil influențată de ea.

¹Fuzzy: vag, neclar; în acest context este tradus în limba română și ca “nuanțat”.

²Se poate face o paralelă cu funcția caracteristică definită pentru o submulțime A a lui X :

$$f_A(x) = \begin{cases} 1 & \text{dacă } x \in A \\ 0 & \text{dacă } x \in X \setminus A \end{cases}$$

Insistând pe direcția aceasta, putem afirma că modul în care se enunță regulile de producție bazate pe logica tradițională:

Daca A atunci B

este aplicabil doar pentru cazul în care caracterul vag lipsește cu desăvârșire, de exemplu în matematică. Totuși, considerând regula: “Dacă e înnorat, atunci va ploua” realizăm că enunțul este vag, cel puțin din cauza următoare: noțiunea de înnorat este vagă – rareori cerul este în totalitate acoperit de nori; vorbim de “parțial înnorat” sau “un pic înnorat” sau “foarte înnorat” și niciunul din acești termeni nu are o caracterizare clară; dacă nu luăm în considerare aceste nuanțe, atunci regula anterioară ar fi utilizabilă doar pentru cazul în care cerul e complet acoperit de nori. Chiar și “ploaia” poate fi nuanțată – picură, plouă torențial etc.

Logica fuzzy este asociată deci cu incertitudinea nestatistică. Trăsătura esențială a teoriei mulțimilor și a logicii fuzzy este manipularea riguroasă a incertitudinii. Se pun la dispoziție modalități de definire, descriere și analiză a caracteristicilor vagi.

12.2 Teoria mulțimilor fuzzy

În teoria clasică a mulțimilor, un element fie face parte dintr-o mulțime, fie nu. În mulțimile fuzzy însă, apartenența la o mulțime se exprimă printr-un grad de apartenență, pentru care valoarea este un număr cuprins între 0 și 1. Putem vedea aceasta ca o generalizare a mulțimilor clasice: dacă un element aparține unei mulțimi clasice, atunci valoarea funcției de apartenență este 1, altfel 0 - de fapt, funcția caracteristică a unei mulțimi.

Să considerăm de exemplu mulțimea oamenilor înalți. Evident, putem spune că o persoană care are înălțimea de 2.10 metri face parte din această mulțime. La fel se poate spune și despre un om cu înălțimea de 2 m sau de 1.90 m; putem nuanța aici faptele, spunând că ultimele două persoane aparțin într-o măsură mai mică acestei mulțimi. O persoană de 1.60 m sau mai puțin nu mai poate fi considerată ca făcând parte din mulțimea oamenilor înalți. Soluția schițată aici este asignarea unor grade de apartenență la o mulțime pentru elementele în discuție. Să considerăm tabelul 12.1 în care pentru diferite exemple de înălțimi vom specifica gradul de apartenență la mulțimea considerată. Mulțimea oamenilor înalți este considerată din acest moment o mulțime fuzzy (nuanțată).

Observăm că o mulțime fuzzy se poate specifica prin perechi de elemente de grad de apartenență/element. O notație mai compactă pentru tabelul 12.1 este:

$$\hat{\text{Înalt}} = \{1.0/2.10, 0.8/2, 0.6/1.90, 0.4/1.80, 0.2/1.70, 0/1.60\}$$

Persoană	Înălțime	Grad de apartenență
A	2.10 m	1
B	2 m	0.8
C	1.90 m	0.6
D	1.80 m	0.4
E	1.70 m	0.2
F	1.60 m	0.0

Tabela 12.1: Înălțimi și gradul de apartenență la mulțimea fuzzy a oamenilor înalți.

Valorile extreme 0 și 1 se pot interpreta astfel: dacă $\mu_A(x) = 1$ atunci spunem că x cu certitudine aparține lui A , dacă $\mu_A(x) = 0$ atunci x cu certitudine nu aparține lui A .

O altă variantă este specificarea unei funcții de apartenență $\mu_A(x)$, unde μ_A este o funcție cu valori în intervalul $[0, 1]$, A este o mulțime fuzzy, x este un element din universul discursului pentru care se stabilește gradul de apartenență la mulțimea A . Astfel, $\mu_{\text{Înalt}}(2.10 \text{ m}) = 1$, $\mu_{\text{Înalt}}(1.90 \text{ m}) = 0.4$ etc.

Funcțiile de apartenență se pot reprezenta grafic, pe axa orizontală fiind valori ale elementelor, iar pe verticală valoarea funcției de apartenență, precum în figurile 12.1 sau 12.2.

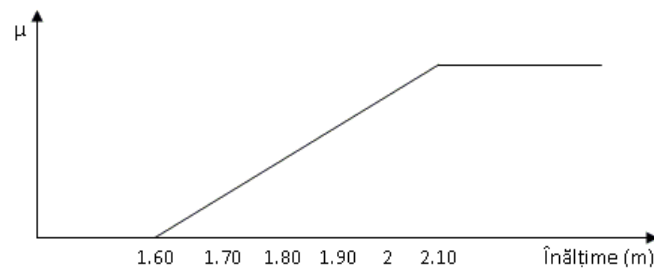


Figura 12.1: Reprezentarea grafică a funcției de apartenență pentru mulțimea “Înalt”

Formele poligonale date în graficele din figura 12.1 și 12.2 nu sunt singurele care se pot folosi. Se poate de exemplu utiliza o funcție de tip Gaussian pentru modelarea gradului de apartenență:

$$\mu_{\text{Cald}}(T) = e^{-\frac{(T-25^\circ)^2}{50}}$$

unde T este temperatura exprimată în grade Celsius. Totuși, funcțiile formate cu porțiuni liniare sunt mai ușor de calculat și în practică au un comportament bun; eventuala lor nederivabilitate nu este o problemă. Din rațiuni evidente, spunem că funcția din figura 12.2 este triunghiulară.

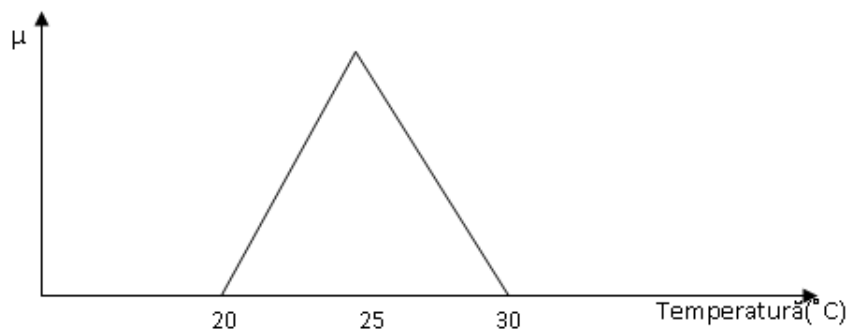


Figura 12.2: Reprezentarea grafică a funcției de apartenență pentru mulțimea “cald”

În sistemele fuzzy un element poate să aparțină la două mulțimi fuzzy, simultan. De exemplu, o persoană cu înălțimea de 1.75 metri face parte din mulțimea oamenilor înalți în măsura 0.3³ și totodată aparține mulțimii oamenilor de înălțime medie în măsura 0.45 - a se vedea graficele din figura 12.3. Remarcăm că noțiunile nu se exclud reciproc.

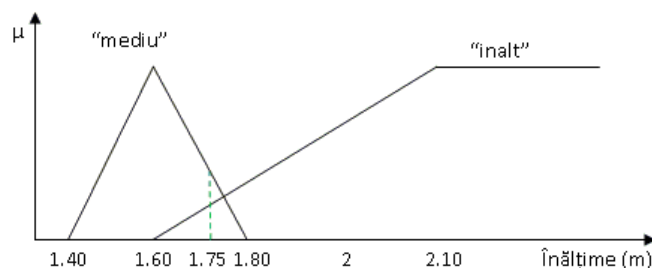


Figura 12.3: Valorile fuzzy “Mediu” și “Înalt” reprezentate pe același grafic

12.3 Operații cu mulțimi fuzzy

Raționamentul presupune operații logice; o mare parte din noțiunile și operațiile din algebra booleană au fost preluate și adaptate la logica fuzzy.

Înainte de a defini aceste operații, este cazul să enumerăm două paradoxuri din logica binară:

1. (Paradoxul mincinosului, paradoxul cretanului) O persoană spune: “eu mint”. Dacă presupunem că această propoziție este adevărată, atunci înseamnă că spusele persoanei sunt false, deci de fapt ea nu minte,

³Valoarea exactă se află intersectând o dreaptă verticală care trece prin valoarea 1.75 de pe abscisă cu graficul funcției de apartenență.

ceea ce e o contradicție cu presupunerea inițială. Dacă presupunem că afirmația persoanei este falsă, atunci înseamnă că dimpotrivă, persoana nu minte, deci din nou contradicție cu presupunerea noastră. Oricare din cele două valori de adevăr am vrea să asociem afirmației “eu mint”, ajungem la o contradicție. Ori, cum doar una din valorile Adevărat și Fals pot fi asociate unei propoziții⁴, avem un paradox.

2. (Paradoxul bărbierului) Într-un sat există un bărbier care barbiereste pe toți bărbații care nu se barbieresc singuri. Care este valoarea de adevăr a propoziției “Bărbierul se barbiereste singur”? Printr-un procedeu asemănător cu cel anterior, se ajunge la concluzia că niciuna din cele două valori de adevăr nu pot fi asociate propoziției, pentru că s-ar ajunge la contradicție.

Aceste probleme sunt rezolvate de către logica fuzzy, putându-se da valori de adevăr pentru propozițiile discutate. Intuitiv, pentru ambele afirmații am putea spune că ele sunt tot atât de adevărate pe cât sunt de false.

Vom trece acum la definirea operațiilor pentru mulțimi fuzzy. Definițiile îi aparțin lui Zadeh.

12.3.1 Egalitatea mulțimilor fuzzy

În teoria clasică a mulțimilor, două mulțimi sunt egale dacă au exact aceleași elemente. Pentru că o mulțime fuzzy înseamnă elemente cu grad de apartenență la ea, spunem că două mulțimi fuzzy sunt egale dacă pentru domenii de valori identice au exact aceleași valori ale funcțiilor de apartenență.

12.3.2 Incluziunea mulțimilor fuzzy

În teoria clasică a mulțimilor, o mulțime A este o submulțime a mulțimii B dacă orice element din A se găsește și în B . Pentru cazul mulțimilor fuzzy, folosim următorul exemplu ca suport intuitiv pentru definirea incluziunii: mulțimea oamenilor foarte înalți este inclusă în mulțimea oamenilor înalți. Evident, pentru un element x pentru care $\mu_{\text{foarte înalt}}(x) = m$, valoarea asociată lui față de mulțimea oamenilor înalți este cel puțin la fel de mare: $\mu_{\text{înalt}}(x) = m + \varepsilon, \varepsilon \geq 0$. Ca atare, spunem că mulțimea fuzzy A este inclusă în mulțimea fuzzy B dacă cele două mulțimi conțin aceleași elemente și $\mu_A(x) \leq \mu_B(x), \forall x$. Desigur, și alte definiții sunt posibile.

12.3.3 Complementara unei mulțimi fuzzy

În teoria clasică a mulțimilor, complementul unei mulțimi A este mulțimea formată din toate elementele care nu aparțin lui A .

⁴Conform principiului terțului exclus, a treia variantă nu este posibilă.

Pentru definiția relativ la mulțimi fuzzy, pornim de la un exemplu: considerăm mulțimea oamenilor de înălțime medie, pentru care funcția de apartenență este dată în figura 12.3. Se pune problema: când spunem că o persoană nu este de înălțime medie? Dacă persoana are înălțimea 1.80 m, evident că face parte din mulțimea oamenilor de înălțime medie; dacă are 1.90 m sau 1.70 m, atunci e evident că nu face parte din ea. Pentru o persoană pentru care gradul de apartenență la mulțimea oamenilor de înălțime medie este, să spunem, 0.7, pare rezonabil să spunem că ea nu aparține la această mulțime cu măsura $1 - 0.7 = 0.3$. Putem deci defini valoarea de apartenență a complementului mulțimii ca fiind unu minus valoarea de apartenență la mulțime. Desigur, și alte definiții sunt posibile.

Definiția dată contrazice principiul terțului exclus: în logica clasică se spune că ceva fie este A, fie este non-A. Gradul de adevăr pentru afirmațiile discutate în cele două paradoxuri poate fi luat 0.5, deci fiecare propoziție are aceeași valoare de adevăr ca și contrara ei.

12.3.4 Intersecția a două mulțimi fuzzy

În varianta clasică, intersecția a două mulțimi este o mulțime formată din elementele comune celor care se intersectează.

Definirea pentru mulțimi fuzzy nu este unică; oricare ar fi varianta folosită, trebuie să se respecte următoarele:

1. operația să fie comutativă: $\mu_{A \cap B}(x) = \mu_{B \cap A}(x)$
2. de asemenea, să avem asociativitate: $\mu_{(A \cap B) \cap C}(x) = \mu_{A \cap (B \cap C)}(x)$
3. monotonie: dacă valoarea funcției de apartenență a unui element la o mulțime scade, atunci valoarea funcției de apartenență pentru mulțimea respectivă intersectată cu o alta nu trebuie să crească.

În practică, cel mai mic grad de apartenență relativ la cele două mulțimi determină gradul de apartenență la intersecție. Varianta de operator de intersecție dată de către Zadeh este:

$$\mu_{A \cap B}(x) = \min \{ \mu_A(x), \mu_B(x) \}$$

Se poate arăta ușor că dacă $A \subset B$, atunci $A \cap B = A$, în sens fuzzy, ceea ce este în concordanță cu ceea ce avem și în teoria clasică a mulțimilor. Desigur, și alte definiții sunt posibile pentru acest operator.

12.3.5 Reuniunea a două mulțimi fuzzy

În teoria clasică a mulțimilor, reuniunea a două mulțimi este o mulțime formată din toate elementele care se regăsesc în ele. Pentru definirea relativ la mulțimi fuzzy, se iau în considerare proprietăți similare cu cele de la

intersecție (doar la monotonie apare diferența), iar varianta dată de către Zadeh este:

$$\mu_{A \cup B}(x) = \max \{ \mu_A(x), \mu_B(x) \}$$

Se pot da și alte definiții pentru acest operator.

12.3.6 Operatori de compensare

În timp ce operațiile din cadrul teoriei clasice a mulțimilor sunt unic definite, pentru mulțimile fuzzy există și alte posibilități de definire a lor decât cele date mai sus. Operatorii de compensare tratează în special cazul reuniunii și al intersecției de mulțimi fuzzy. Intersecția este des întâlnită în cadrul regulilor fuzzy.

Folosind definiția intersecției dată de Zadeh, valoare funcției de apartenență pentru intersecția a 2 mulțimi fuzzy este controlată de cea mai mică din valorile existente. De exemplu, pentru regula “dacă A și B și C atunci D”, dacă valorile de apartenență ale lui A, B și C sunt respectiv 0.2, 0.8, 0.9, efectul pe care îl are A asupra rezultatului final este prea pronunțat. În practică, definiția intersecției dată de Zadeh nu e întotdeauna adecvată.

S-au definit mai mulți operatori de compensare. Ei formulează răspunsuri la întrebarea: cât de mult poate să compenseze creșterea unor variabile valorile mici ale altora? Vom prezenta două variante ale acestor operatori: operatorul de medie și operatorul gama.

Prin operatorul de medie se stabilește că valoarea funcției de apartenență este media valorilor individuale:

$$\mu_{X_1 \cap X_2 \cap \dots \cap X_n}(x) = \frac{\sum_{i=1}^n \mu_{X_i}(x)}{n}$$

Operatorul gama este mai complex și pare să reprezinte mai bine procesul de decizie umană decât definițiile lui Zadeh. El este definit ca:

$$\mu_\gamma = \left(\prod_{i=1}^n \mu_i \right)^{1-\gamma} \cdot \left(1 - \prod_{i=1}^n (1 - \mu_i) \right)^\gamma$$

unde $0 \leq \gamma \leq 1$, iar n este numărul de valori fuzzy implicate în intersecție. În practică, cel mai frecvent valorile lui γ sunt între 0.2 și 0.4.

12.4 Reguli fuzzy

Regulile clasice au forma următoare:

$$\text{Dacă } A_1 \text{ și } A_2 \text{ și } \dots \text{ și } A_n \text{ atunci } C$$

unde “ A_1 și A_2 și \dots și A_n ” se numește antecedent sau premisă iar C este consecvent sau consecință sau concluzie. De exemplu:

Dacă înălțimea bărbatului este mai mare de 1.80 m, atunci masa lui este mai mare de 50 kg.

Regulile fuzzy păstrează această formă generală, dar pot să apară diferențe pe partea de consecvent. Cele două variante des folosite sunt datorate lui Mamdani:

Dacă X_1 este A_1 și ... și X_n este A_n atunci Y este B

și respectiv lui Takagi, Sugeno și Kang:

Dacă X_1 este A_1 și ... și X_n este A_n atunci $Y = p_0 + p_1X_1 + \dots + p_nX_n$

unde X_i sunt variabile fuzzy de intrare, A_i sunt mulțimi fuzzy peste variabilele X_i ($1 \leq i \leq n$), Y este o variabilă fuzzy de ieșire, B este o mulțime fuzzy definită peste valorile lui Y iar p_j sunt coeficienți reali ($0 \leq j \leq n$).

Pentru probleme de clasificare există următoarea formă de regulă:

Dacă X_1 este A_1 și ... și X_n este A_n atunci Y face parte din clasa i în măsura GC_i .

Nuanțarea⁵ este pasul prin care se combină valorile din antecedentul unei reguli, folosind operațiile cu mulțimi fuzzy; pasul se aplică pentru fiecare regulă în parte. Prin combinarea regulilor date la pas de denuanțare se obține o ieșire care poate fi folosită ca rezultat inferențial sau ca indicație de control al unui sistem.

Exemplificarea acestor reguli se face pentru cazul unei centrale de încălzire, pentru care sunt date niște reguli privind reglarea debitului de gaz astfel încât să se obțină o temperatură potrivită. Se pleacă de la reguli în care se folosesc noțiuni vagi (temperatură potrivită, variație mare, mărește debitul etc.) și se obține o indicație pentru regulatorul de gaz.

Vom considera că avem valori de intrare precum temperatura interioară (notată *TempIn*), cea exterioară (*TempExt*), modificarea de temperatură interioară în ultimele 5 minute (*DeltaTempIn*); ca valoare de ieșire avem *ModificareDebit*. Fiecare valoare de intrare concretă va avea un grad de apartenență fuzzy la diferite mulțimi (de exemplu, pentru *TempIn* avem apartenență la mulțimi precum *rece*, *confortabil* etc.

Pentru valoarea *TempIn* avem trei mulțimi fuzzy: *rece*, *confortabil*, *prea cald*. Pentru *TempExt* avem mulțimile fuzzy *foarte rece*, *rece*, *cald*, *foarte cald* și *fierbinte*. Pentru *DeltaTempIn* definim mulțimile fuzzy: *larg negativ*, *mic negativ*, *aproximativ zero*, *pozitiv mic*, *larg pozitiv*⁶, iar pentru *ModificareDebit* avem seturile fuzzy *scade mult*, *scade puțin*, *nu schimbă*, *crește puțin*, *crește mult*.

Vom considera doar câteva reguli, suficiente pentru exemplificarea nuanțării și denuanțării:

Regula 1: *Dacă $TempIn$ este confortabilă și $DeltaTempIn$ este aproximativ zero, atunci $ModificareDebit$ este nu schimbă;*

⁵În original: fuzzyfication.

⁶“Mic” și “larg” se referă la valorile absolute (modulul) cantităților măsurate.

Regula 2: Dacă $TempExt$ este rece și $DeltaTempIn$ este mic negativ, atunci $ModificareDebit$ este crește puțin;

Regula 3: Dacă $TempIn$ este prea cald și $DeltaTempIn$ este larg pozitivă, atunci $ModificareDebit$ este scade mult;

Regula 4: Dacă $TempIn$ este rece și $DeltaTempIn$ este aproximativ zero, atunci $ModificareDebit$ este crește puțin.

Pentru $TempIn$, mulțimea *confortabil* se definește ca $\{0/15^\circ, 1/21^\circ, 0/27^\circ\}$, interpretată ca o funcție de apartenență de tip triunghiular. De exemplu, pentru 18° și 24° gradele de apartenență sunt ambele 0.5. Pentru *rece* avem mulțimea fuzzy $\{1/10^\circ, 1/16^\circ, 0/21^\circ\}$, iar *prea cald* este mulțimea $\{0/21^\circ, 1/27^\circ, 1/33^\circ\}$.

Pentru $DeltaTempIn$:

$$\begin{aligned} negativ\ mic &= \{0/-4^\circ, 1/-2^\circ, 0/0^\circ\} \\ aproape\ zero &= \{0/-2^\circ, 1/0^\circ, 0/+2^\circ\} \\ larg\ pozitiv &= \{0/2^\circ, 1/4^\circ, 1/6^\circ\} \end{aligned}$$

Pentru $TempExt$, $rece = \{0/-1^\circ, 1/10^\circ, 0/21^\circ\}$.

Să presupunem că temperatura interioară este de 20° , diferența de temperatură din ultimele 5 minute este -1.5° iar temperatura exterioară este de 11° .

Conform mulțimilor fuzzy date anterior, avem:

- pentru $TempIn$, $\mu_{rece}(20^\circ) = 0.25$, $\mu_{confortabil}(20^\circ) = 0.75$, $\mu_{prea\ cald}(20^\circ) = 0$;
- pentru $DeltaTempIn$, $\mu_{mic\ negativ}(-1.5^\circ) = 0.80$, $\mu_{aproximativ\ zero}(-1.5^\circ) = 0.20$, $\mu_{larg\ pozitiv}(-1.5^\circ) = 0$
- pentru $TempExt$, $\mu_{rece}(11^\circ) = 0.90$

Aplicăm aceste valori celor 4 reguli fuzzy de mai sus. Ținem cont de faptul că antecedentele din reguli sunt exprimate cu conjuncție, corespunzătoare intersecției de mulțimi, ceea ce în logica fuzzy se implementează prin funcția min. Obținem:

Regula 1: $0.75 \cap 0.20 = 0.20 = \mu_{nu\ schimba}(ModificareDebit)$

Regula 2: $0.90 \cap 0.80 = 0.80 = \mu_{creste\ putin}(ModificareDebit)$

Regula 3: $0 \cap 0 = 0 = \mu_{scade\ mult}(ModificareDebit)$

Regula 4: $0.25 \cap 0.20 = 0.20 = \mu_{creste\ putin}(ModificareDebit)$

Activarea acestor reguli se face în paralel. Observăm că pentru regula a treia ieșirea este zero, deci ea nu se va aplica. Din regulile 2 și 4 avem două valori pentru apartenența $\mu_{crește puțin}(ModificareDebit)$; se va lua maximum celor două valori, deci $\mu_{crește puțin}(ModificareDebit) = 0.8$.

Variația de debit este modelată la rândul ei fuzzy, așa cum se arată în figura 12.4.

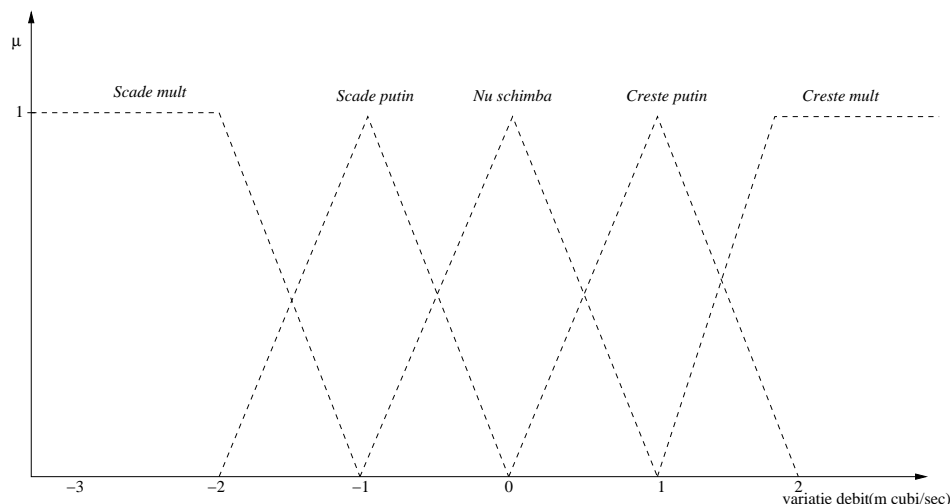


Figura 12.4: Mulțimi fuzzy pentru debitul de gaz

Denuanțarea este operația prin care se obține un răspuns concret la problemă, adică se furnizează o valoare exprimată în metri cubi pe secundă pentru debitul de gaz. Plecând de la graficul anterior, se trasează conturul mărginit de orizontalele $y = 0.2$ - pentru *nu schimbă* - și $y = 0.8$ - pentru *crește puțin*. Se obține figura geometrică desenată cu linie continuă în figura 12.5, pentru care se determină centrul de greutate; verticala dusă prin acest centru de greutate intersecționează axa debitului la valoarea $+0.76$, aceasta fiind indicația dată regulatorului de gaz: crește cu $0.76 \text{ m}^3/\text{s}$ debitul de gaz.

Există mai multe metode care se pot folosi pentru denuanțare; a se vedea [3] pentru detalii.

12.5 Măsuri ale gradului de nuanțare

În cazul unei mulțimi fuzzy se poate pune întrebarea: cât este ea de nuanțată? Pentru o mulțime fuzzy discretă, se pot introduce câteva măsuri care cuantifică gradul de nuanțare. Acestea au ca scop măsurarea gradului de incertitudine, care apare, de exemplu, în cazul exprimării vagi. În continuare, prezentarea merge pe exemplele din [3].

Dacă considerăm mulțimea peștilor, atunci pentru diferite elemente de mai jos avem gradele exprimate: $\mu_{pești}(biban) = 1.0$, $\mu_{pești}(peștișor auriu) =$

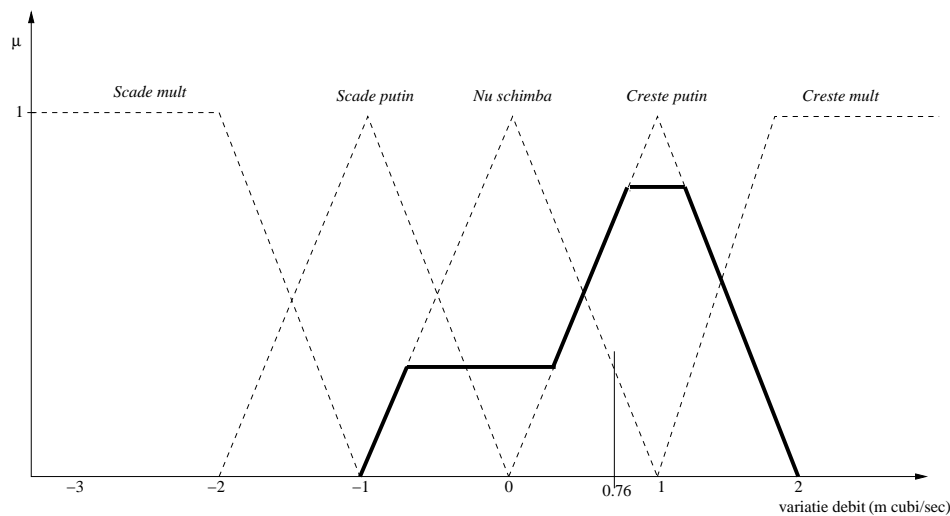


Figura 12.5: Procesul de denuanțare.

1.0, $\mu_{pești}(\text{căluț de mare}) = 0.8$, $\mu_{pești}(\text{balenă}) = 0.0$. Pentru mulțimea fuzzy *flori*, gradul de apartenență ar putea fi: $\mu_{flori}(\text{trandafir}) = 1.0$, $\mu_{flori}(\text{pâine}) = 0.0$, $\mu_{flori}(\text{lemn câinesc}) = 0.5$. Intuitiv, putem spune că mulțimea de flori exemplificată este mai vagă decât mulțimea peștilor: în ultimul caz, valorile de apartenență sunt mai apropiate de cele ale unei funcții de apartenență din cazul unei mulțimi clasice, 0 și 1.

Se preia din fizică și din teoria informației noțiunea de *entropie*, care măsoară gradul de dezorganizare a unui sistem. Spre deosebire de teoria informației unde măsura entropiei este unic determinată⁷, în teoria mulțimilor fuzzy sunt mai multe variante acceptate. Se pleacă de la o sumă de proprietăți pe care ar trebui să le respecte o astfel de măsură entropică și se introduc mai multe funcții care respectă o parte sau toate aceste proprietăți. De exemplu, pentru o mulțime clasică, din care un element fie face parte, fie nu, măsura gradului de nuanțare ar trebui să dea rezultatul 0 - *i.e.* o mulțime clasică nu este vagă. De asemenea, cu cât sunt mai multe valori pentru care valoarea funcției de apartenență este 0.5 sau apropiată de aceasta, cu atât mulțimea este mai vagă: un element care aparține cu măsura 0.5 la o mulțime fuzzy face și nu face parte din mulțimea dată în aceeași măsură.

Înainte de a da diferite variante de măsurare a gradului de nuanțare, se introduce noțiunea de “mulțime mai ascuțită”, ce exprimă relația între două mulțimi fuzzy: spunem că o mulțime S^* este *mai ascuțită* decât o altă mulțime fuzzy S - ambele definite peste același univers al discursului - dacă $\mu_{S^*}(x) \leq \mu_S(x)$ pentru cazul în care $\mu_S(x) < 0.5$ și $\mu_{S^*}(x) \geq \mu_S(x)$ dacă $\mu_S(x) > 0.5$; pentru $\mu_S(x) = 0.5$ valoarea $\mu_{S^*}(x)$ poate fi oricât.

Proprietățile de mai jos sunt punct de plecare pentru determinarea dife-

⁷ Abstracție făcând de o constantă multiplicativă

ritelor funcții de măsurare a gradului de nuanțare.

caracterul exact: $H(A) = 0$ dacă și numai dacă A este o mulțime clasică;

maximalitatea: $H(A)$ este maximă dacă $\mu_A(x) = 0.5, \forall x$ din universul discursului;

ascuțirea: $H(A) \geq H(A^*)$ dacă A^* este mai ascuțită decât A ;

simetria: $H(A) = H(\overline{A})$;

principiul includerii și excluderii: $H(A \cup B) = H(A) + H(B) - H(A \cap B)$

O variantă de funcție de măsurare a gradului de nuanțare, introdusă de DeLuca și Termini și care respectă toate cele 5 proprietăți de mai sus este:

$$H_{DT}(A) = -K \sum_{i=1}^n (\mu_i \log \mu_i + (1 - \mu_i) \log(1 - \mu_i)) \quad (12.1)$$

unde K este un număr pozitiv oarecare. Varianta introdusă de Pal și Pal este:

$$H_{PP}(A) = K \sum_{i=1}^n (\mu_i e^{1-\mu_i} + (1 - \mu_i) e^{\mu_i}) \quad (12.2)$$

Se pot defini și alte variante de măsurare a gradului de nuanțare a unei mulțimi vagi.

Bibliografie

- [1] **Stanford Machine Learning**, curs online, [Coursera](#), Andrew Ng
- [2] **Neural network design**, 2nd edition, Martin T. Hagan, Howard B. Demuth, Mark Hudson Beale, Orlado de Jesús, 2014, <http://hagan.okstate.edu/NNDesign.pdf>
- [3] **Computational Intelligence. Concepts to Implementations**, Russell C. Eberhart and Yuhui Shi, Morgan Kaufmann, 2007
- [4] **Computational Intelligence. An Introduction**, Andries P. Engelbrecht, John Willeys and Sons, 2007
- [5] **Neural Networks and Learning Machines**, ediția a treia, Simon Haykin, Prentice Hall, 2008.
- [6] **Inteligență computațională**, Răzvan Andonie, Angel Cațaron, Universitatea Transilvania din Brașov, <http://www.cwu.edu/~andonie/Cursul%20de%20IA%20Brasov.pdf>
- [7] **An Elementary Introduction to Statistical Learning Theory**, Sanjeev Kulkarni, Gilbert Harman, Willey, 2011
- [8] **A Brief Introduction to Neural Networks**, David Kriesel, 2007, http://www.dkriesel.com/en/science/neural_networks
- [9] **Self-Organized Formation of Topologically Correct Feature Maps**, Teuvo Kohonen, 1982, în Biological Cybernetics 43 (1), pp. 59–69
- [10] **Self-Organizing Maps**, Kohonen, T., Springer Berlin Heidelberg, 2001
- [11] **Principiile inteligenței artificiale**, Dan Dumitrescu, Editura Albastrea
- [12] **Algoritmi genetici și strategii evolutive - aplicații în inteligența artificială**, Dan Dumitrescu, Editura Albastrea

- [13] **The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network**, G. A. Carpenter and S. Grossberg, IEEE Computer, Vol. 21, No. 3, 1988
- [14] **Fuzzy Sets**, Lotfi Zadeh, Information and Control, Vol. 8, 1965
- [15] **Genetic Algorithms + Data Structures = Evolution Programs**, Zbigniew Michalewicz, 1996, Ed. Springer-Verlag
- [16] **Introduction to artificial neural networks**, Jacek M. Zurada, 1992, West Publishing Company
- [17] ***K*-means++: the advantages of careful seeding**, Arthur, D. and Vassilvitskii, S., Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 1027–1035.
- [18] **Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps**, G. A. Carpenter and S. Grossberg and N. Markuzon and J. H. Reynolds and D. B. Rosen, IEEE Transactions on Neural Networks, 1992, vol. 3, no. 5, pp. 698–713.
- [19] **The Matrix Cookbook**, Kaare Brandt Petersen, Michael Syskind Pedersen, 2012, Technical University of Denmark
- [20] **Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**, John Duchi, Elad Hazan, Yoram Singer, Journal of Machine Learning Research (12), pp. 2121–2159, 2011
- [21] **ADADELTA: An adaptive learning rate method**, Matthew D. Zeiler, [arXiv:1212.5701](https://arxiv.org/abs/1212.5701), 2012
- [22] **Learning representations by back-propagating errors**, David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, Nature 323 (6088), pp. 533–536, 1986
- [23] **Deep Learning**, Ian Goodfellow, Yoshua Bengio, Aaron Courville, MIT Press, 2016
- [24] Michael A. Nielsen, **Neural Networks and Deep Learning**, Determination Press, 2015
- [25] Xavier Glorot, Yoshua Bengio, **Understanding the difficulty of training deep feedforward neural networks**, Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics, 2010

- [26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**, Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), 2015