Inteligență artificială

Versiunea 22 ianuarie 2024

Lucian M. Sasu, Ph.D.

Cuprins

Listă de figuri **5**

1 Introducere **8**
1.1 Rețele neurale artificiale 10
1.1.1 Bazele biologice 10
1.1.2 Diferențe între rețele neurale artificiale și naturale . . 12
1.1.3 Aplicabilitate 12
1.2 Calcul evoluționist 13
1.2.1 Bazele biologice 13
1.2.2 Cromozomi 14
1.2.3 Diferențe între cromozomii biologici și cei artificiali . . 14
1.2.4 Aplicabilitate 14
1.3 Tipuri de învățare în machine learning 14
1.3.1 Învățarea supervizată 15
1.3.2 Învățarea prin întărire 15
1.3.3 Învățarea nesupervizată 16
1.4 Auto-organizarea 16

2 Regresia liniară **18**
2.1 Exemplu și notații 18
2.2 Funcția de eroare 22
2.3 Metoda de căutare după direcția gradientului 25
2.4 Metoda ecuațiilor normale 31

3 Regresia logistică **33**
3.1 Încadrare și exemple 33
3.2 Regresia logistică binară 34
3.2.1 Setul de instruire 34
3.2.2 Reprezentarea modelului 34
3.2.3 Suprafața de decizie a regresiei logistice binare 37
3.2.4 Funcția de cost 38
3.2.5 Algoritmul de instruire 41
3.3 Regresia logistică multinomială 43

3.3.1 Setul de instruire 43
3.3.2 Funcția softmax 43
3.3.3 Reprezentarea modelului 44
3.3.4 Funcția de cost 45
3.3.5 Gradientul funcției de cost 46
3.3.6 Algoritmul de instruire 47
3.4 Precizări suplimentare 48
3.4.1 Redundanța parametrilor pentru regresia logistică multinomială 48
3.4.2 Relația dintre cele două tipuri de regresii logistice . . 49
3.4.3 Calculul numeric al funcției softmax 50
3.4.4 Trucul "log sum exp" 50

4 Regularizarea modelelor 52
4.1 Overfitting, underfitting pentru regresia liniară 52
4.2 Regularizarea L_2 pentru regresia liniară 54
4.3 Regularizarea modelelor de regresie logistică 56
4.3.1 Regularizarea L_2 pentru regresia logistică binară . . . 56
4.3.2 Regularizarea L_2 pentru regresia logistică multinomială 58

5 Perceptronul liniar 60
5.1 Încadrare și exemplu 60
5.2 Definiții și notații 62
5.3 Perceptronul liniar 63
5.4 Algoritmul de instruire a perceptronului 64
5.5 Exemplu numeric 67
5.6 Modificarea ponderilor ca algoritm de tip gradient descent . . 69
5.7 Convergența perceptronului 70
5.8 Algoritmul lui Gallant 72
5.9 Comentarii 73

6 Perceptronii multistrat 75
6.1 Motivație pentru rețele neurale multistrat 75
6.2 Notații 76
6.3 Setul de instruire 76
6.4 Rețeaua neurală MLP 76
6.4.1 Arhitectură 76
6.4.2 Funcții de activare 81
6.5 Pasul de propagare înainte 85
6.6 Funcții de cost 87
6.6.1 Funcția de cost pentru problemă de regresie 88
6.6.2 Funcția de cost pentru discriminarea a două clase . . . 89
6.6.3 Funcția de cost pentru mai mult de două clase independente 91

6.6.4 Funcția de cost pentru clasificare cu mai mult de două clase . . . 91
6.7 Inițializarea ponderilor rețelei . . . 92
6.8 Grafuri computaționale și calcul de derivate parțiale . . . 92
6.8.1 Derivarea funcțiilor compuse . . . 92
6.8.2 Exemplul 1: căi unice . . . 94
6.8.3 Exemplul 2: căi multiple . . . 96
6.9 Algoritmul backpropagation . . . 98
6.9.1 Scopul algoritmului . . . 98
6.9.2 Variante de calcul . . . 98
6.9.3 Calculul gradienților – justificare matematică . . . 99
6.9.4 Algoritmul backpropagation . . . 104
6.10 Utilizarea rețelei pentru inferență . . . 106
6.11 Discuții . . . 106

7 Rețele neurale cu funcții de activare radială 107
7.1 Motivația rețelei: teorema lui Cover . . . 107
7.2 Funcții cu activare radială . . . 109
7.3 Rețele cu funcții cu activare radială . . . 111
7.4 Clustering folosind algoritmul K-means . . . 113
7.5 Algoritmul de inițializare K-means++ . . . 115
7.6 Determinarea ponderilor pentru RBF . . . 116
7.7 Algoritmul de instruire a rețelei RBF . . . 116

8 Rețele neurale convoluționale 118
8.1 Convoluție . . . 120
8.1.1 Exemplificare de convoluție . . . 121
8.1.2 Bordarea intrării . . . 124
8.1.3 Pasul kernelului . . . 126
8.1.4 Numărul de ponderi și de operații pentru convoluție cu intrare pe un singur canal . . . 129
8.1.5 Convoluții pentru intrări cu mai multe canale . . . 129
8.2 Funcții de activare . . . 133
8.3 Straturi de pooling . . . 133
8.3.1 Max pooling . . . 133
8.3.2 Average pooling . . . 133
8.3.3 Global average pooling . . . 134
8.4 Straturi complet conectate . . . 134
8.5 Funcții de cost, regularizare . . . 135
8.6 Rețelele neurale de convoluție: soluție end-to-end . . . 136
8.7 Exemple de rețele neurale de convoluție . . . 138
8.7.1 LeNet-5 . . . 140
8.7.2 VGG16 . . . 140

9 Calcul evoluționist **142**
9.1 Taxonomie 142
9.2 Algoritmi genetici 143
9.3 Fundamente teoretice 146
9.4 Problema reprezentării datelor în algoritmii genetici 149
9.4.1 Varianta cu penalizare 152
9.4.2 Varianta cu reparare 152
9.4.3 Codificarea adecvată a indivizilor 153
9.5 Exemplu: problema orarului 154

Bibliografie **156**

Listă de figuri

1.1 Neuron natural 11
1.2 Neuron de tip Purkinje din cortexul cerebelar 11
1.3 Schema de lucru pentru învățare supervizată 15
1.4 Schema de lucru pentru învățare prin întărire 16
1.5 Schema de lucru pentru învățare nesupervizată 17

2.1 Reprezentarea grafică a datelor de vânzare a unor apartamente 19
2.2 Model liniar de predicție și aproximarea costului unui apartament de 80 de metri pătrați. 20
2.3 Fluxul de lucru într-un proces de instruire automată. 21
2.4 Regresie liniară univariată 22
2.5 Funcția de eroare pătratică, cu $\theta_0 = 0$ și θ_1 variabil. 23
2.6 Funcția de eroare pentru model liniar univariat cu θ_0 și θ_1 variabili. 24
2.7 Curbe de contur pentru functia de eroare a unui model liniar univariat, cu parametrii θ_0, θ_1 variabili. 24
2.8 Graful de calcul pentru modelul de regresie liniară multivariată 28

3.1 Graficul sigmoidei logistice definită în ecuația 3.4 35
3.2 Graf de calcul pentru regresia logistică binară 36
3.3 Minim global și minim local pentru funcție neconvexă 39
3.4 Cele două ramuri ale funcției *Cost* din ecuația (3.13) 40

4.1 Trei polinoame pentru aproximarea prețului pornind de la suprafață 53

5.1 Mai multe drepte pentru mulțimi liniar separabile 61
5.2 Semiplanele pozitive și negative 61
5.3 Suprafața de decizie S, distanța de la punctul de coordonate $\mathbf{x}$ la S, poziția vectorului de ponderi $\mathbf{w}$ față de S 63
5.4 Graful de calcul al unui perceptron liniar 64
5.5 Exemplu numeric: dreapta inițială 68
5.6 Exemplu numeric: prima modificare a dreptei 68
5.7 Exemplu numeric: Ultima modificare a dreptei 69

5.8 Problema neseparabilă liniar XOR 73

6.1 Două clase de puncte ce nu sunt liniar separabile 75
6.2 Rețea MLP cu 3 straturi 79
6.3 Rețea MLP cu 4 straturi 79
6.4 Legăturile între noduri din straturile $l-1$ și l. 80
6.5 Graficul funcției de activare ReLU 83
6.6 Graficul funcției de activare PReLU pentru $\alpha = 0.1$ 84
6.7 Graficul funcției de activare ELU pentru $\alpha = 0.1$ 85
6.8 Graficul funcției de activare Swish pentru $\beta \in \{0.5, 2\}$ 86
6.9 Functiile mean squared error, mean absolute error, Huber ($\delta = 0.5$) pentru cazul unidimensional 90
6.10 Graf computațional pentru funcție multivariată compusă . . . 93
6.11 Graful computațional pentru calculul valorii funcției $f(x, y, z) = x \cdot (y + z)$ 95
6.12 Graful pentru calculul derivatelor parțiale 95
6.13 Graf computațional pentru $f(x, y, z) = (x + y) \cdot (y + z)$ . . . 97
6.14 Calculul gradienților pentru $f(x, y, z) = (x + y) \cdot (y + z)$ . . . 97
6.15 Curbe de eroare pentru antrenare și validare 99
6.16 Rețea neurală cu un strat ascuns, folosită ca schemă pentru determinarea fomulelor gradienților. 101
6.17 Graf computațional pentru ponderile și valorile de bias din ultimul strat. 101
6.18 Graf computațional pentru ponderile și valorile de bias din stratul ascuns. 103

7.1 Set de dete neseparabil liniar transformat în set liniar separabil 108
7.2 Transformarea problemei XOR, ce nu permite separare liniară (partea stângă) în problemă liniar separabilă 109
7.3 Structura unei rețele RBF, plecând de la funcția de interpolare din ecuația 7.4. 112
7.4 Structura unei rețele RBF cu centri selectați 113
7.5 Caz nefavorabil pentru K-means la alegerea centroizilor inițiali 115
7.6 Alegerea optimă a centroizilor inițiali 116

8.1 Clasificare de imagini cu rețea neurală de convoluție 119
8.2 Detectare de obiecte cu rețea neurală de convoluție 119
8.3 Segmentare semantică și segmentare de instanțe 119
8.4 Exemplu de cifră 5 din setul de date MNIST 120
8.5 Imagine color, cu reprezentare numerică pe trei canale de culoare 120
8.6 Exemplificare intrare, kernel și ieșire 122
8.7 Prima convoluție (cu kernel răsturnat), aplicată peste intrările marcate cu linie punctată 122
8.8 A doua convoluție 122

8.9 A treia convoluție 122
8.10 Ultima convoluție 122
8.11 Exemplu numeric pentru convoluție de 3×3 peste o intrare de 5×5, fără bordare și cu pas 1. Preluare din [1] 123
8.12 O valoare centrală din intrare este de mai multe ori folosită într-o convoluție 125
8.13 Convoluție validă cu pas 2 127
8.14 Exemplu numeric pentru convoluție de 3×3 peste o intrare de 5×5, cu pas 2 si padding 1 (convoluție asemenea). Preluare din [1] 128
8.15 Filtru de convoluție pentru un strat de intrare cu 4 canale . . 130
8.16 Convoluție cu două filtre pentru un strat de intrare cu 4 canale 132
8.17 Max pooling 134
8.18 Average pooling 134
8.19 Global average pooling pe un canal, respectiv pe 3 canale . . 135
8.20 Straturi complet conectate 136
8.21 Arhitectura generică a unei rețele neurale convoluționale . . . 137
8.22 Succesiune de trăsături învățate de o rețea neurală profundă . 137
8.23 Schiță a unei rețele de convoluție 139
8.24 Arhitectura rețelei convoluționale LeNet5 139
8.25 Arhitectura rețelei convoluționale VGG16 141

9.1 Reprezentarea unui cromozom în algoritmii genetici 144

Capitolul 1

Introducere

Inteligența artificială (IA) 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. Câteva subdomenii ale inteligenței artificiale sunt:

- modele cu învățare automată (machine learning) – modele liniare, mixturi Gaussiene, rețele neurale (sau "neuronale") artificiale, Support Vector Machines, arbori de decizie, rețele neurale convoluționale etc.;
- calcul evoluționist;
- mulțimi și logică fuzzy;
- imunitate artificială;
- 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 artificială este rapidă, iar pentru o problemă se pot folosi mai multe abordări: de exemplu, optimizarea se poate face prin algoritmi genetici sau prin algoritmi de căutare locală.

Cursul de față se axează pe primele două direcții.

Metodele din inteligența artificială sunt frecvent inspirate din biologie: rețelele neurale au pornit de la modelul imaginat pentru neuronul biologic, calculul evoluționist este bazat pe teoria evoluției și pe genetică. Sistemele fuzzy sunt introduse pentru a permite manipularea impreciziei, altfel decât prin teoria probabilităților.

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

imaginează algoritmul pentru a rezolva problema; este un demers creativ, depinzând de imaginația, puterea de abstractizare și experiența persoanei în cauză; este un proces creativ, la ora actuală efectuat de cele mai multe ori de către oameni. Tot aici, de cele mai multe ori rezultatele sunt exacte și se insistă permanent pe micșorarea complexității de calcul sau de memorie a algoritmilor; de multe ori însă o soluție exactă presupune un resurse de timp și memorie prohibitive.

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

Sistemele studiate aici sunt cu un mare grad de aplicabilitate. De exemplu, algoritmii genetici pot fi folosiți pentru o clasă largă de funcții, nedepinzând atât de mult – precum se întâmplă în cercetările operaționale – de ipoteze care în practică pot fi prea restrictive.

O definiție a "inteligenței" potrivită pentru contextul de IA este:

Definiția 1. *Inteligența este abilitatea unui sistem de a–și adapta comportamentul pentru a–și îndeplini scopurile în mediul său. Este o proprietatate 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 subdomeniului de machine learning: 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 [2], [3].

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 de optimizarea unor funcții de cost și de probleme de căutare; tehnicile sunt bazate pe concepte 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 algoritmii genetici, programarea evoluționistă, programarea genetică și strategii de evoluție.

Sistemele rezultate prin inteligență artificială 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 uneltei potrivite pentru problema în cauză poate fi o provocare, deoarece de regulă se pot folosi mai multe abordări; nu se știe, de regulă, care e varianta cea mai potrivită de abordare.

1.1 Rețele neurale artificiale

1.1.1 Bazele biologice

Rețeaua neurală biologică a evoluat de-a lungul timpului, 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 și mai eficient decât cu sistemele electronice actuale.

Studiile efectuate în ultimul secol au permis enunțarea unor prinicipii 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, preluare din [4]

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

1.1.2 Diferențe între rețele neurale 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 rețele neurale 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 rețelele neurale 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 rețele neurale artificiale funcționează pe procesoare de câtiva 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, la un consum de energie mult mai redus.

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.

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;

- *Regăsirea de date pe baza conținutului*, folosind memorie asociativă – 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 automată (clustering)* - 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 îndeplini o anumită sarcină, cu anumite constrângeri; utilitatea rețelelor neurale provine din faptul că nu se presupune că există dependențe liniare între acțiune și efect.

1.2 Calcul evoluționist

Principalele paradigme[1] ale calculului evoluționist sunt:

- algoritmii 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 - similari cu algoritmii 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 tratează ereditatea, adică transmiterea trăsăturilor 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.

[1] "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 organismele vii variază în lungime, chiar dacă de la un organism la altul din aceeași specie pentru un cromozom specific lungimea este constantă. În algoritmii 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 algoritmii genetici, procentul de combinație poate să difere.

1.2.4 Aplicabilitate

Principala arie de aplicare este optimizarea, pentru situațiile în care căutarea soluției cere un timp îndelungat. Algoritmii genetici sunt folositi 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 Tipuri de învățare în machine learning

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

Există și variante intermediare, de exemplu învățarea semi–supervizată și cea activă.

1.3.1 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 obtinerea 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, rețelele convoluționale.

Figura 1.3: Schema de lucru pentru învățare supervizată

1.3.2 Învățarea prin întărire

Învățarea prin întărire (eng: reinforcement learning) 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 o ființă sau un agent artificial inteligent 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.

Figura 1.4: Schema de lucru pentru învățare prin întărire

1.3.3 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 competititiv. Aplicatiile clasice sunt analiza asociererilor, gruparea pe baza de similaritate si estimarea de densitate de probabilitate.

Schema de lucru este dată în figura 1.5. Acest tip de adaptare este prezent în modele ce efectuează clustering, analiza de asocieri, analiza componentelor principale, detectarea de anomalii, etc.

1.4 Auto-organizarea

Auto-organizarea, alături de învățare, este un alt atribut important al sistemelor cu inteligență artificială. 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:

Figura 1.5: Schema de lucru pentru învățare nesupervizată

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 [5].*

sau:

Definiția 3. *Sistemele cu auto-organizare se auto-organizează pentru a clasifica percepțiile din mediu în percepții ce pot fi recunoscute, sau șabloane [5].*

Modelele neurale care posedă proprietatea de auto-organizare includ: Self Organizing Maps[2], variantele de Neural Gas[3,4,5] sau variante de rețele neurale recurente[6].

[2]T. Kohonen, *Self-Organizing Maps*, Springer, 2001.

[3]T. Martinetz, S. Berkovich, K. Schulten, "Neural-gas network for vector quantization and its application to time-series prediction", IEEE Transactions on Neural Networks, vol. 4, no. 4, pp. 558–569, 1993.

[4]B. Fritzke, "A Growing Neural Gas Network Learns Topologies", Proceedings of the 7th International Conference on Neural Information Processing Systems, NIPS'94, (Cambridge, MA, USA), p. 625–632, MIT Press, 1994.

[5]Y. Prudent, A. Ennaji, "An incremental growing neural gas learns topologies", Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 2, pp. 1211–1216 vol. 2, 2005.

[6]D. Han, K. Doya, J. Tani, "Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks", Neural networks, vol. 129, pp. 149–162, 2020.

Capitolul 2

Regresia liniară

Regresia liniară se încadrează în învățarea supervizată și e utilizată pentru construirea unui model de regresie. Modelul construit pleacă de la presupunerea că valoarea ce trebuie prezisă depinde în mod liniar de valorile trăsăturilor de intrare.

2.1 Exemplu și notații

Modelele de regresie pot învăța și prezice valori numerice dintr-o mulțime infinită. Ca exemplu, să presupunem că vrem să facem estimarea (predicția) costului unui apartament, dată fiind suprafața sa. Se cunosc date anterioare despre vânzarea unor astfel de apartamente, precum în tabelul 2.1; datele sunt culese într–o perioadă scurtă de timp, în care prețul nu evoluează odată cu timpul[1]. Pentru simplitatea expunerii, să presupunem că singura trăsătură considerată pentru un apartament este suprafața lui, exprimată în metri pătrați.

Suprafața (m^2)	Prețul (€)
62	87.900
30	62.600
54	85.400
...	...

Tabelul 2.1: Valorile de vânzare ale unor apartamente, pentru care se știe doar trăsătura "suprafața".

Pe baza acestor date vom construi o funcție – numit mai jos și model de predicție – care să ne permită aproximarea prețului (număr real) pentru alte apartamente similare - în același interval de timp, în aceeași arie din care provin datele inițiale. Reprezentarea grafică a datelor este în figura 2.1.

[1] Altfel zis, pe perioada scurtă considerată timpul nu influențează prețul.

Figura 2.1: Reprezentarea grafică a datelor de vânzare a unor apartamente. Pe abscisă este măsurată suprafața (în metri pătrați), pe ordonată este prețul (în mii de euro).

Să presupunem că se dorește estimarea valorii unui apartament de suprafață $s = 80$ de metri pătrați; neavând în exemplele setul nostru de date o atare suprafață, va trebui să "ghicim" un preț. Se poate proceda în felul următor: se trasează o dreaptă care să aproximeze "cât mai bine"[2] norul de puncte reprezentat. Valoarea estimată de model pentru apartamentul cu o suprafață s se află simplu: se duce verticala prin punctul de coordonate $(s, 0)$ și se găsește punctul de intersecție cu dreapta dată de model; valoarea ordonatei $y(s)$ corespunzătoare punctului de intersecție este prețul estimat de model - figura 2.2.

Eroarea estimării este dată de diferența dintre valoarea actuală (frecvent numită în limba engleză ground truth) și cea prezisă de model, pentru cazurile cunoscute.

Modelul liniar de estimare a prețului unui apartament luând în considerare suprafața este:

$$pret_estimat = a \cdot suprafata + b$$

unde a și b sunt coeficienți reali ce vor fi determinați; a se numește pantă (eng: slope) iar b termen liber (eng: intercept). Desigur, se pot folosi funcții polinomiale de grad mai mare decât 1, modele local liniare, rețele neurale artificiale etc. Alegerea celui mai bun model pentru un set de date cunoscut este o problemă în sine. Preferința pentru model liniar se motivează prin aceea că în practică e considerat un punct de plecare bun; în plus, modelele mai simple se recomandă a fi încercate primele[3]. Nu în ultimul rând, *un model liniar este ușor de interpretat*: pentru problema asta, coeficientul a

[2] O formulă pentru a măsura cât de bună e aproximarea rezultată se dă în secțiunea 2.2.

[3] Așa numitul brici al lui Occam: *Entia non sunt multiplicanda praeter necessitatem*, entitățile nu trebuie să fie multiplicate dincolo de necesar.

Figura 2.2: Model liniar de predicție și aproximarea costului unui apartament de 80 de metri pătrați.

este prețul pe metru pătrat (exprimat în €/m^2), b (măsurat în €) este prețul de pornire – niciun apartament nu costă mai puțin de b €– iar creșterea valorii $suprafata$ cu 10m^2 duce la creșterea prețului total cu $10 \cdot a$ €.

Avem mai sus un caz 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 apartamentului cu acea suprafață). 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:

- m, reprezentând numărul de perechi de valori (sau cazuri, sau înregistrări) din setul de instruire; pentru desenul din figura 2.1 este numărul de puncte reprezentate, adică numărul de apartamente pentru care se știe prețul de vânzare;

- $\mathbf{x}^{(i)}$ reprezentând m vectori de intrare, $1 \leq i \leq m$; un astfel de vector este compus de regulă din n valori numerice, numite atribute sau trăsături (eng: features): $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right)^t$: suprafața, numărul de camere, numărul de băi etc. Trăsăturile $x_j^{(i)}$ se mai numesc și variabile predictive sau independente[4]. Dacă nu se precizează altfel, astfel de vectori sunt considerați vectori coloană. Simbolul ^{t} pus în partea superioară reprezintă transpunerea de vector sau de matrice.

- $y^{(i)}$, $1 \leq i \leq m$ – variabila de ieșire (sau de predicție, sau dependentă;

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

eng. ground truth) aferentă valorii $\mathbf{x}^{(i)}$; în cazul exemplificat este un număr real (prețul), dar în general poate fi un vector de valori reale.

Perechea i din setul de antrenare este $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$, $1 \leq i \leq m$. Întregul set de antrenare se scrie ca:

$$\mathscr{S} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) | 1 \leq i \leq m\right\} \tag{2.1}$$

Setul de antrenare se specifică frecvent sub formă tabelară, precum în tabelul 2.1.

Fluxul de lucru în învățarea automată[5] 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 modelul rezultat se mai numește și "ipoteză" și se notează de regulă cu h. Algoritmul de instruire are ca scop determinarea unei forme adecvate a modelului, în cazul de față a unor valori potrivite a coeficienților funcției h.

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

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

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

unde notația cu căciulă se folosește pentru valori estimate de model.

Una din întrebările esențiale este: ce formă are modelul h? Există mai multe variante. Mai sus am pornit cu presupunerea că prețul crește liniar cu suprafața vândută, deci:

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

unde indicele lui h este vectorul coloană de coeficienți $\boldsymbol{\theta} = (\theta_0, \theta_1)^t$.

Acest model (ipoteză) se numește regresie liniară cu o variabilă, sau regresie liniară univariată. Se poate ca pe lângă suprafață – singura valoare

[5] În limba engleză: machine learning.

de intrare considerată până acum – să se mai considere și alte trăsături de intrare; 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:

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

unde: x_0 e mereu 1, $x_1 = x$, vectorul $\boldsymbol{\theta}$ este $(\theta_0, \theta_1)^t$ ca mai sus, vectorul $\mathbf{x}$ este $(x_0, x_1)^t$. Valoarea $x_0 = 1$ permite existența unui termen liber θ_0 în modelul liniar, adică pentru cazul considerat, dreapta de regresie nu trebuie să treacă neapărat prin punctul de coordonate $(0, 0)$.

Graful de calcul pentru modelul de regresie liniară 2.4 este reprezentat în figura 2.4.

Figura 2.4: Graful de calcul pentru regresie liniară univariată

2.2 Funcția de eroare

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, $\hat{y}^{(i)} = h_\theta\left(\mathbf{x}^{(i)}\right)$, să fie cât mai apropiate de valorile *cunoscute* $y^{(i)}$, pentru setul de antrenare $\mathscr{S}$ din (2.1). Pentru toate valorile din setul de instruire, eroarea se poate măsura cu funcția de cost

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

Funcția de eroare J se mai numește și funcție de cost a modelului (în limba engleză: error, loss, cost function; în unele prezentări se face distincție între loss function – eroare pentru un singur caz – și cost function – eroarea agregată peste mai multe valori). Se pot folosi și alte funcții de cost, de exemplu incluzând constrângeri impuse valorilor parametrilor $\boldsymbol{\theta}$ – a se vedea capitolul 4 privind regularizarea. Funcția de mai sus este o alegere populară pentru problemele de regresie, dar nu singura posibilă. Factorul m de la

numitor arată că vizăm calcularea unei valori medii a erorii – altfel, eroarea ar crește de fiecare dată când se adaugă în setul de instruire o pereche $(\mathbf{x}^{(i)}, y^{(i)})$ pentru care $h_\theta(\mathbf{x}^{(i)}) \neq y^{(i)}$, în timp ce media permite compararea erorilor modelului pentru mulțimi cu număr diferite de elemente; numitorul 2 poate fi omis, dar se utilizează din motive estetice pentru calculele de mai târziu. Eroarea J din formula (2.5) este jumătate din eroarea pătratică medie (eng: mean squared error):

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 = 2 \cdot J(\theta_0, \theta_1) \tag{2.6}$$

E util 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 doi depinzând de o singură variabilă (θ_1) și având minimul mai mare sau egal cu zero, a se vedea figura 2.5. Pentru θ_0, θ_1 oarecare forma funcției de eroare este dată în figura 2.6.

Figura 2.5: Funcția de eroare pătratică, cu $\theta_0 = 0$ și θ_1 variabil.

O altă variantă de reprezentare grafică a funcției de eroare este pe baza curbelor de contur: reprezentarea este plană, având pe cele două axe respectiv pe θ_0, θ_1. Pentru o valoare oarecare v se consideră mulțimea tuturor perechilor de parametri θ_0, θ_1 pentru care se obține aceeasi valoare a erorii, adică $J(\theta_0, \theta_1) = v$. Rezultatul este dat de o mulțime de curbe, precum cele reprezentate în figura 2.7. Se poate arăta că pentru model de predicție liniar aceste contururi sunt eliptice; pentru valori v tot mai mari avem elipse tot mai întinse.

Trebuie să găsim acele valori ale coeficienților $\theta_0^{(min)}, \theta_1^{(min)}$ pentru care

Figura 2.6: Funcția de eroare pentru model liniar univariat cu θ_0 și θ_1 variabili.

Figura 2.7: Curbe de contur pentru functia de eroare a unui model liniar univariat, cu parametrii θ_0, θ_1 variabili.

se atinge minimul funcției de eroare[6]:

$$\begin{aligned}\left(\theta_0^{(min)}, \theta_1^{(min)}\right)^t &= \arg \min_{(\theta_0,\theta_1)^t \in \mathbb{R}^2} J(\theta_0, \theta_1) = \\ &= \arg \min_{(\theta_0,\theta_1)^t \in \mathbb{R}^2} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \\ &= \arg \min_{(\theta_0,\theta_1)^t \in \mathbb{R}^2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \\ &= \arg \min_{(\theta_0,\theta_1)^t \in \mathbb{R}^2} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2\end{aligned} \tag{2.7}$$

Coeficienții $\theta_0^{(min)}, \theta_1^{(min)}$ pentru care valoarea erorii J e cea mai mică ne dau modelul de regresie cel mai bun pentru setul pe care s–a facut antrenarea. Pentru o valoare de intrare x vom face predicția cu modelul:

$$h_{\boldsymbol{\theta}^{(min)}}(x) = \mathbf{x}^t \cdot \boldsymbol{\theta} = (x_0, x_1) \cdot (\theta_0^{(min)}, \theta_1^{(min)})^t = \theta_0^{(min)} + \theta_1^{(min)} \cdot x = \hat{y} \tag{2.8}$$

Înbtrebarea cea mai importantă acum este: cum determinăm valorile θ pentru care se atinge minimul funcției de eroare J? Metodele sunt date în cele ce urmează.

2.3 Metoda de căutare după direcția gradientului

În această secțiune se va prezenta o metodă iterativă – coborârea după direcția gradientului (eng: gradient descent) – prin care se face minimizarea funcției de eroare J. Metoda este ierativă, adică se consideră succesiv mai multe valori candidat pentru coeficienții θ_0, θ_1.

Ideea e simplă:

- se pornește cu valori θ_0, θ_1 inițiale, setate aleator sau chiar 0;
- se modifică în mod iterativ valorile curente ale parametrilor θ_0, θ_1 de așa manieră încât J să scadă.

Pentru ultimul punct: valorile curente ale parametrilor θ_0, θ_1 se modifică conform

$$\theta_0 = \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) \tag{2.9}$$

$$\theta_1 = \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \tag{2.10}$$

[6]Notația "$\arg\min \mathbf{X}$" se referă la acele argumente pentru care se obține minimul pentru $\mathbf{X}$. De exemplu, pentru un vector de valori reale $\mathbf{v} = (v_0, \ldots, v_{n-1})^t$, $\min \mathbf{v}$ este valoarea minimă, în timp ce $\arg\min_{0 \le i \le n-1} \mathbf{v}$ este un indice pe care se află valoarea minimă a lui $\mathbf{v}$.

și (important!) atribuirile se operează în mod simultan pentru θ_0, θ_1.

Această simultaneitate e cerută din cauză că la calculele (2.9), (2.10) trebuie să ne asigurăm că aceiași θ_0, θ_1 sunt folosiți pentru evaluarea ambelor derivate parțiale. Simultaneitatea se poate obține astfel: se calculează expresiile din membrii drepți ai ecuațiilor (2.9) și (2.10) și se asignează unor variabile temporare $\theta_0^{(temp)}$ și respectiv $\theta_1^{(temp)}$; doar după ce ambele variabile temporare sunt calculate, valorile lor se atribuie corespunzător: $\theta_0 = \theta_0^{(temp)}$ și $\theta_1 = \theta_1^{(temp)}$. Alternativ, se poate folosi calcul vectorizat, în care se operează concomitent peste toate componentele vectorului $\boldsymbol{\theta}$.

Aceleași formule se scriu în mod vectorial ca:

$$\begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) \\ \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \end{pmatrix} \tag{2.11}$$

Dacă folosim notația nabla pentru vectorul de derivate parțiale (vectorul gradient)[7]:

$$\nabla_{\boldsymbol{\theta}} J = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix} \tag{2.12}$$

atunci mai putem scrie formula de modificare a ponderilor ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.13}$$

În bibliotecile care permit calcul matricial atribuirea simultană este implementată automat. O formă matricială pentru gradientul modelului liniar este dată în continuare.

Coeficientul $\alpha > 0$ 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 sau să aibă alterneze perioade de scădere cu cele de creștere). Dacă se constată acest al doilea fenomen, valoarea lui α trebuie scăzută. Odată ce o valoare potrivită pentru α este găsită, nu e neapărat nevoie ca aceasta să fie modificată de-a lungul iterațiilor.

Metoda se poate folosi pentru reducerea valorilor unei funcții ce depinde de oricâte variabile. În general, se poate ajunge într-un minim – local sau global – sau punct de inflexiune al funcției căreia i se aplică.

Valorile θ_0, θ_1 se inițializează aleator cu valori mici în jurul lui 0, sau chiar cu 0. Algoritmul de căutare după direcția gradientului are forma:

[7] Vectorul de derivate parțiale este un vector de funcții; acestea se vor evalua pentru perechea de valori θ_0, θ_1, obținând deci valori numerice concrete. Vectorul de derivate parțial este, totodată, transpusa matricei Jacobiene a funcției J.

```
repeta{
```

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{simultan pentru } j = 0, 1 \tag{2.14}$$

```
} pana la convergenta
```

Condiția de convergență poate fi: de la o iterație la alta valoarea lui J nu mai scade semnificativ, sau norma diferenței între două valori succesive ale vectorului $\boldsymbol{\theta}$ este sub un prag mic $\varepsilon > 0$ setat, sau se atinge un număr maxim de iterații permise etc.

Putem explicita derivatele parțiale pentru forma funcției de eroare considerate:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\hat{y}^{(i)} - y^{(i)} \right) \cdot x_j^{(i)} \right] \tag{2.15}$$

și formula de modificare a ponderilor din (2.14) devine:

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \quad \text{simultan pentru } j = 0, 1 \tag{2.16}$$

În ce situație iterațiile din algoritmul de mai sus se opresc? Dacă funcția de cost J este într-un punct de extrem (local sau global)[8], gradientul ei în acel punct este vectorul zero (vectorul nul) și deci valoarea parametrilor $\boldsymbol{\theta}$ nu se va mai modifica, odată ce s–a atins o valoare de minim a lui J. Această observație explică două din condițiile de convergență.

Se recomandă a se 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.

Valoarea optimă a lui α depinde de setul de date peste care se calculează funcției de eroare J. α este un hiperparametru care influențează succesul și viteza învățării.

Extinderea la date de intrare cu mai multe trăsături (date multivariate), *i.e.* $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)} \right)^t$ se poate trata tot prin metoda de căutare după direcția gradientului, prin modificări imediate:

1. modelul de predicție devine

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \ldots \theta_n \cdot x_n = \boldsymbol{\theta}^t \cdot \mathbf{x} = \mathbf{x}^t \cdot \boldsymbol{\theta} \tag{2.17}$$

   unde

$$\mathbf{x} = (x_0 = 1, x_1, \ldots, x_n)^t, \ \boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^t \tag{2.18}$$

[8] Sau punct de inflexiune; dar pentru funcția de eroare considerată în acest curs avem doar o valoare de minim și niciun punct de inflexiune.

Am făcut trecerea de la vectorul $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right)^t$ de n componente la unul de $n+1$ componente, prin adăugarea unei valori $x_0 = 1$, care se va înmulți cu termenul liber θ_0. Se observă că valorile trăsăturilor de intrare sunt la puterea întâi, *modelul rămânând deci liniar*. Graful de calcul pentru regresia liniară multivariată 2.17 este reprezentat în figura 2.8

Figura 2.8: Graful de calcul pentru modelul de regresie liniară multivariată. Este o extindere a grafului reprezentat în figura 2.4.

2. funcția de eroare J se păstrează, dar pentru predicție se folosește modelul $h_{\boldsymbol{\theta}}$ din (2.17);

3. ecuația (2.14) din algoritmul de căutare devine:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \quad (2.19)$$

simultan pentru $j = 0, 1, \ldots, n$.

Pentru modelul multivariat se recomandă ca valorile trăsăturilor de intrare să fie în scale similare. Acest lucru se obține făcând în prealabil o scalare a datelor la un interval convenabil ales, *e.g.* $[0, 1]$. Alternativ, se poate face standardizarea datelor: datele sunt transformate astfel încât fiecare atribut devine cu media zero si dispersia 1. În ambele cazuri, beneficiul este un număr mult mai mic de iterații până la convergența algoritmului.

Rescriem în cele ce urmează funcția de cost (2.5) și formula de modificare a ponderilor din ecuația (2.19) folosind calcul matricial. Acest lucru favorizează implementare eficientă în medii precum NumPy sau Matlab.

Pentru început, notăm cu $\mathbf{X}$ matricea datelor de intrare din setul de instruire $\mathscr{S}$:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \ldots & x_n^{(m)} \end{pmatrix} \quad (2.20)$$

unde linia i conține valorile de intrare asociate vectorului $\mathbf{x}^{(i)}$, iar vectorul coloană de indice $1 \leq j \leq n$ corespunde unei trăsături predictive. Valorile

de ieșire asociate vectorilor de intrare sunt de asemenea stocate matricial, folosind un vector coloană:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m \tag{2.21}$$

Așa cum în (2.18) am adăugat o componentă $x_0 = 1$ pentru a permite un termen liber în modelul liniar, vom extinde matricea de date $\mathbf{X}$ din ecuația (2.20) cu o primă coloană plină cu 1; pentru simplitatea notațiilor, vom folosi și în continuare litera $\mathbf{X}$ pentru această matrice, numită matrice de design:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{2.22}$$

Notăm cu $\hat{\mathbf{y}}$ vectorul de valori prezise de modelul $h_{\boldsymbol{\theta}}$ pentru matricea $\mathbf{X}$:

$$\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{pmatrix} \in \mathbb{R}^m \tag{2.23}$$

Se verifică ușor că

$$\hat{\mathbf{y}} = \mathbf{X} \cdot \boldsymbol{\theta} \tag{2.24}$$

Funcția de cost J se rescrie matricial astfel:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{2m} (\hat{\mathbf{y}} - \mathbf{y})^t \cdot (\hat{\mathbf{y}} - \mathbf{y}) \\ &= \frac{1}{2m} \left(X\boldsymbol{\theta} - \boldsymbol{y} \right)^t \cdot \left(X\boldsymbol{\theta} - \boldsymbol{y} \right) \end{aligned} \tag{2.25}$$

Pentru calculul vectorului gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ vom utiliza următoarele relații cunoscute din algebra liniară și analiza matematică:

- transpusa unei sume de matrice este suma transpuselor:

$$(A + B)^t = A^t + B^t \tag{2.26}$$

- transpusa unui produs de matrice este produsul transpuselor matricelor în ordine inversă:

$$(A_1 \cdot A_2 \dots A_p)^t = A_p^t \cdot A_{p-1}^t \dots A_1^t \tag{2.27}$$

- dacă a este un număr real, atunci el poate fi interpretat ca o matrice cu o linie și o coloană și din acest motiv

$$a^t = a \tag{2.28}$$

- conform excelentei lucrări pentru calcul matricial a lui Petersen si Pedersen[9], avem

$$\nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^t \mathbf{A} \right) = \mathbf{A} \tag{2.29}$$

- conform aceleiași lucrări secțiunea 2.4.2, ecuația (81):

$$\nabla_{\boldsymbol{\theta}} \left(\theta^t \mathbf{A} \theta \right) = (\mathbf{A} + \mathbf{A}^t) \cdot \theta \tag{2.30}$$

  pentru $\mathbf{A}$ matrice pătratică.

Pentru calculul matricial al vectorului de derivate parțiale (gradienți) $\left(\frac{\partial}{\partial \theta_j} J(\theta) \right)_{j=0,n}$ dezvoltăm (2.25):

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m} \left(X\boldsymbol{\theta} - \boldsymbol{y} \right)^t \left(X\boldsymbol{\theta} - \boldsymbol{y} \right) \\ &= \frac{1}{2m} \left\{ \left(\mathbf{X}\boldsymbol{\theta} \right)^t \mathbf{X}\boldsymbol{\theta} - \left(\mathbf{X}\boldsymbol{\theta} \right)^t \mathbf{y} - \mathbf{y}^t \left(\mathbf{X}\boldsymbol{\theta} \right) + \mathbf{y}^t \mathbf{y} \right\} \\ &= \frac{1}{2m} \left\{ \boldsymbol{\theta}^t \mathbf{X}^t \mathbf{X} \boldsymbol{\theta} - 2\boldsymbol{\theta}^t \mathbf{X}^t \mathbf{y} + \mathbf{y}^t \mathbf{y} \right\} \end{aligned} \tag{2.31}$$

Ţinem cont de faptul că derivata parțială e operator liniar, deci derivata parțială a unei sume de funcții este suma derivatelor parțiale ale lor:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2m} \left\{ \nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^t \mathbf{X}^t \mathbf{X} \boldsymbol{\theta} \right) - 2\nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^t \mathbf{X}^t \mathbf{y} \right) + \nabla_{\boldsymbol{\theta}} \left(\mathbf{y}^t \mathbf{y} \right) \right\} \tag{2.32}$$

Considerând relațiile (2.26–2.30) și observând că scalarul $\mathbf{y}^t \mathbf{y}$ nu depinde de $\boldsymbol{\theta}$ (și deci derivarea în raport cu $\boldsymbol{\theta}$ produce valoarea 0), obținem vectorul gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \left\{ \mathbf{X}^t \mathbf{X} \boldsymbol{\theta} - \mathbf{X}^t \mathbf{y} \right\} = \frac{1}{m} \mathbf{X}^t \left(\mathbf{X}\boldsymbol{\theta} - \mathbf{y} \right) \tag{2.33}$$

Se observă dispariția lui 2 de la numitor; se înțelege acum de ce am preferat ca funcția de eroare să fie jumătate din eroarea pătratică medie.

Modificările de ponderi θ_j din ecuația (2.19) se scriu matricial pentru vectorul $\boldsymbol{\theta}$ ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \boldsymbol{\theta} - \frac{\alpha}{m} \mathbf{X}^t (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \tag{2.34}$$

iar rescrierea algoritmului de instruire prin gradient descent este imediată. Avantajele acestei scrieri matriciale sunt:

- Se poate face o implementare vectorizată, mai eficientă la execuție;
- Atribuirile simultane pentru θ_j se realizează automat pentru forma matricială.

[9] K. B. Petersen și M. S. Pedersen, "The matrix cookbook", 2012, https://www2.imm.dtu.dk/pubdb/edoc/imm3274.pdf.

2.4 Metoda ecuațiilor normale

Există o metodă care dă valorile optime $\boldsymbol{\theta}^{(min)}$ pe baza unui calcul algebric.

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

$$\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^{n+1}} J(\boldsymbol{\theta}) \tag{2.35}$$

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

$$\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(min)}\right) = \mathbf{0} \tag{2.36}$$

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

Deoarece funcția de eroare J e și convexă, condiția de minimizare necesară dată de (2.36) este și suficientă și deci se ajunge în unicul minim al lui J. Înlocuind formula gradientului $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ din ecuația (2.33) în (2.36) obținem:

$$\mathbf{X}^t \mathbf{X} \boldsymbol{\theta}^{(min)} - \mathbf{X}^t \mathbf{y} = \mathbf{0} \tag{2.37}$$

ce definește un sistem de ecuații numite "ecuații normale". Mai departe, dacă matricea $\mathbf{X}^t\mathbf{X}$ este inversabilă (nesingulară), vectorul de parametri $\boldsymbol{\theta}^{(min)}$ se determină ca

$$\boldsymbol{\theta}^{(min)} = \left(\mathbf{X}^t \mathbf{X}\right)^{-1} \mathbf{X}^t \cdot \mathbf{y} \tag{2.38}$$

Precizări:

1. Expresia $\left(\mathbf{X}^t\mathbf{X}\right)^{-1}\mathbf{X}^t$ se mai numește și pseudo–inversa Moore–Penrose și uzual se notează cu $\mathbf{X}^+$; pentru o matrice inversabilă inversa și pseudo–inversa ei coincid; pentru calculul pseudoinversei unei matrice se poate folosi în Octave și Matlab funcția `pinv`, iar în Python funcția `numpy.linalg.pinv` apelată pentru matricea $\mathbf{X}$;

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 discutată 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, de exemplu 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; mai general, una din coloane poate fi combinație liniară a altor coloane și dacă se știe care e, se poate elimina;

- se folosesc prea multe trăsături față de numărul de cazuri din setul de instruire ($m < n$); în acest caz se poate renunța la câteva trăsături, adică se elimină coloane din $\mathbf{X}$, sau se folosește regularizarea – a se vedea capitolul 4.

Ordinea de mai sus este cea sugerată pentru 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 $\boldsymbol{\theta}^{(min)}$, 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, neavând de fapt rată de învățare;

2. În timp ce pentru 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 pseudo-inversei poate fi prohibitiv din punct de vedere al memoriei și timpului de calcul necesar.

Capitolul 3

Regresia logistică

3.1 Încadrare și exemple

Regresia logistică este folosită pentru estimare de probabilitate condiționată și clasificare. Inițial dezvoltată pentru recunoașterea a două clase, a fost extinsă pentru a discrimina între oricâte categorii.

Ca mod de instruire se folosește învățarea supervizată. Intrările sunt vectori numerici, iar clasele sunt fie două – pentru regresia logistică binară – 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 legitim, dându–se atribute ale emailului: 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 de laborator;
- clasificarea unei imagini: conține un câine sau o pisică (aici presupunem că fiecare imagine are exact unul din aceste două animale); sau vrem ca modelul să învețe să recunoască dacă imaginea conține sau nu un anumit obiect.

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 unei imagini ca reprezentând măr, pară, banană, cireașă.

Modelul dat de regresia logistică (fie ea binară sau multinomială) construiește o estimare a probabilității condiționate, dată fiind intrarea curentă (conținut

email, imagine etc.); mai precis, se determină care este probabilitatea ca obiectul descris de vectorul de intrare să fie dintr-o clasă anume:

$$P(clasa_1|vector_de_intrare), \ldots, P(clasa_K|vector_de_intrare) \quad (3.1)$$

de exemplu probabilitățile ca emailul să fie de tip știre, respectiv muncă, prieteni etc. Faptul că se estimează probabilități, adică valori continue din $(0, 1)$ justifică cuvântul "regresie" din denumirile modelului.

Odată ce avem un model care produce probabillități condiționate, clasificarea se face găsind $clasa_k$ pentru care probabilitatea ei din (3.1) este maximă.

3.2 Regresia logistică binară

3.2.1 Setul de instruire

În cazul regresiei logistice binare se urmărește discriminarea între două clase. Clasele sunt convenabil date ca fiind "1" – clasa pozitivă – și respectiv "0"– clasa negativă[1]. Setul de instruire este de forma:

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | 1 \leq i \leq m \right\} \quad (3.2)$$

unde vectorul $\mathbf{x}^{(i)} = \left(x_0^{(i)}, x_1^{(i)}, \ldots, x_n^{(i)} \right)^t \in \mathbf{R}^{n+1}$ conține valorile trăsăturilor obiectului i, iar $y^{(i)} \in \{0, 1\}$ este clasa de care aparține obiectul i. Ca și până acum, notația $\mathbf{v}^t$ reprezintă transpunerea vectorului sau a matricei $\mathbf{v}$, iar m e numărul de date din setul de instruire. Vom considera că $x_0^{(i)} = 1$ pentru orice i, pentru a permite un termen liber în discriminatorul implementat de regresia logistică.

3.2.2 Reprezentarea modelului

Pentru regresia logistică modelul de predicție trebuie să producă o valoare reprezentând probabilitatea condiționată $P(clasa|vector_de_intrare)$. Vom folosi în acest scop o funcție $h_{\boldsymbol{\theta}}(\cdot)$:

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

unde $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^t \in \mathbf{R}^{n+1}$ este un vector coloană cu $n + 1$ coeficienți – sau ponderi (eng: weights) – ce vor fi determinați prin procesul de învățare. Se arată ușor că funcția $h_{\boldsymbol{\theta}}$ este cu valori în intervalul $(0, 1)$, deci putem să o folosim pe post de probabilitate. Probabilitatea din (3.3) este o probabilitate

[1] De exemplu, clasa pozitivă poate fi "mail de tip spam" sau "poză cu pisică". Clasa negativă este "mail legitim" și respectiv "poză cu câine".

condiționată de intrarea curentă – adică vectorul $\mathbf{x}$ – și parametrizată de $\boldsymbol{\theta}$. $P(y = 1|\mathbf{x}; \boldsymbol{\theta})$ este gradul de încredere că obiectul descris de vectorul $\mathbf{x}$ face parte din clasa 1, iar încrederea este influențată și de ponderile din vectorul $\boldsymbol{\theta}$.

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

$$\sigma : \mathbf{R} \to (0, 1),\ \sigma(z) = \frac{1}{1 + \exp(-z)} \tag{3.4}$$

și numită sigmoida logistică[2]; este reprezentată grafic în figura 3.1. Codomeniul funcției logistice este $(0, 1)$, compatibil cu valorile admisibile pentru funcție de probabilitate; extremele 0 și 1 care sunt permise pentru probabilități nu se ating însă de către sigmoida logistică. Avem că funcția σ este derivabilă, strict crescătoare și $\lim_{z\to-\infty} \sigma(z) = 0$, $\lim_{z\to\infty} \sigma(z) = 1$. Denumirea de "sigmoidă" este dată de alura graficului, amintind de litera S.

Figura 3.1: Graficul sigmoidei logistice definită în ecuația 3.4

Graful de calcul implementat de regresia logistică binară este dat în figura 3.2.

Pentru funcția σ următoarele proprietăți sunt notabile:

$$1 - \sigma(z) = \sigma(-z) \tag{3.5}$$

iar derivata lui σ se calculează ușor pe baza lui $\sigma(z)$:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{3.6}$$

[2]Eng: logistic sigmoid, expit. Sigmoida logistică este inversa unei alte funcții cunoscute, logit: $logit : (0, 1) \to \mathbf{R}$, $logit(p) = \ln \frac{p}{1-p}$.

Figura 3.2: Graf de calcul pentru regresia logistică binară

Probabilitatea ca obiectul $\mathbf{x}$ să fie de clasă 0, sau clasă negativă, este $P(y = 0|\mathbf{x}; \boldsymbol{\theta})$ și e determinată ca find complementul față de 1 al evenimentului de a fi de clasă pozitivă[3]:

$$P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = 1 - \sigma(\boldsymbol{\theta}^t\mathbf{x}) = \sigma(-\boldsymbol{\theta}^t\mathbf{x}) \tag{3.7}$$

Dorim să determinăm ponderile din vectorul $\boldsymbol{\theta}$ astfel încât:

1. pentru acei vectori $\mathbf{x}^{(i)}$ pentru care eticheta asociată $y^{(i)}$ este 1 să avem $P(y^{(i)} = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ cât mai aproape de 1, adică de $y^{(i)}$;

2. pentru datele de intrare $\mathbf{x}^{(i)}$ cu $y^{(i)} = 0$ să avem $P(y^{(i)} = 0|\mathbf{x}^{(i)}; \boldsymbol{\theta})$ cât mai apropiată de 1; echivalent, vrem aici ca $P(y^{(i)} = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ să fie cât mai aproape de 0, adică de $y^{(i)}$.

Pe scurt, dorim ca vectorul de ponderi $\boldsymbol{\theta}$ să fie astfel încât predicțiile $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ să fie cât mai apropiat de $y^{(i)}$.

Ponderile din vectorul $\boldsymbol{\theta}$ vor fi determinate prin învățare automată (eng. machine learning).

După învățare, modelul probabilist dat de (3.3) este mai departe folosit pentru a face clasificare astfel: dacă pentru un vector de intrare $\mathbf{x}$ avem

$$P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) \tag{3.8}$$

atunci se decide că obiectul descris de vectorul $\mathbf{x}$ este din clasa 1, pozitivă; altfel este din clasa 0 – negativă.

Având în vedere că $P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta})$, decizia bazată pe inecuația (3.8) se reformulează echivalent ca: vom clasifica obiectul descris de vectorul $\mathbf{x}$ ca fiind de clasă pozitivă dacă $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq 0.5$ și negativă altfel.

Unii autori consideră că dacă avem o intrare $\mathbf{x}$ pentru care $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 0.5$, atunci modelul nu ar trebui să facă o clasificare a intrării: este tot atât de probabil să fie de clasă pozitivă pe cât e de probabil să fie de clasă negativă. În inecuația (3.11) am considerat – în mod mai degrabă arbitrar – că situația cu egalitate să fie tratată ca un caz pozitiv.

[3]Avem două evenimente complementare: un obiect e fie de clasă pozitivă, fie negativă. Sumele probabilităților acestor două evenimente este 1, conform axiomelor care fundamentează teoria probabilităților.

Se poate să se folosească alt prag de discriminare pentru clase decât 0.5, de exemplu: obiectul descris de $\mathbf{x}$ este de clasă pozitivă dacă $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq 0.8$ și de clasă negativă altfel. O astfel de setare de prag se efectuează pentru date de instruire în care clasele pozitive și cele negative sunt puternic debalansate (mult mai multe exemple de tip pozitiv decât negativ, sau invers), sau pentru cazul în care penalizarea care se plătește pentru o clasificare eronată de tip pozitiv (un fals pozitiv) este foarte mare față de penalizarea pentru clasificarea eronată de tip negativ (fals negativ) – sau invers. Pentru ultimul caz, un exemplu este: e mai grav dacă un mail legitim este clasificat ca fiind de tip spam și scos din inbox, față de cazul în care un mail spam este clasificat eronat ca fiind legitim: vom impune ca $P(spam|email)$ sa fie comparat cu un prag t mare (de exemplu 0.8, sau 0.99) pentru a decide că e vorba într–adevăr de spam[4].

E util să menționăm vectorizarea calculului: notăm cu $\mathbf{X}$ matricea care are drept linii vectorii $\mathbf{x}^{(i)t}$ – matricea de design întâlnită și la regresia liniară:

$$\mathbf{X} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{3.9}$$

atunci cele m probabilități $\hat{y}^{(i)} = P(y^{(i)} = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x}^{(\mathbf{i})})$ sunt componentele vectorului coloană

$$\hat{\mathbf{y}} = \sigma(\mathbf{X} \cdot \boldsymbol{\theta}) \tag{3.10}$$

cu funcția σ aplicată pe fiecare componentă a vectorului coloană $\mathbf{X} \cdot \boldsymbol{\theta}$.

3.2.3 Suprafața de decizie a regresiei logistice binare

În pofida caracterului neliniar al funcției ce definește modelul – dat în ecuația (3.3) – se arată ușor că suprafața care desparte regiunea $\mathbf{x}$ de clasă pozitivă și cea de clasă negativă este o varietate liniară[5], dacă pragul este 0.5.

Inegalitatea (3.8) se scrie echivalent:

$$\begin{aligned} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) \quad & \Longleftrightarrow \quad \tfrac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \geq \tfrac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ & \Longleftrightarrow \quad 1 \geq \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x}) \\ & \overset{\text{(logaritmând)}}{\Longleftrightarrow} \quad 0 \geq -\boldsymbol{\theta}^t \cdot \mathbf{x} \\ & \Longleftrightarrow \quad \boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0 \end{aligned} \tag{3.11}$$

[4]Invităm cititorul să verifice că pentru un prag diferit de 0.5 suprafața de separare este tot formă liniară.

[5]Prin abuz se folosește și denumirea "hiperplan"; în timp ce un hiperplan obligatoriu trebuie să treacă prin origine, varietatea liniară este 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. ale trăsăturilor originare, atunci suprafața de decizie poate fi mai complicată. De exemplu, plecăm de la vectorul (x_1, x_2) și să considerăm extinderea lui cu trăsături polinomiale, $\mathbf{x} = (x_0 = 1, x_1, x_2, x_1^2, x_2^2, x_1x_2)^t$; rezultă că $\boldsymbol{\theta} \in \mathbf{R}^6$; avem $\boldsymbol{\theta}^t \cdot \mathbf{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$. Pentru valorile[6] $\theta_0 = -4, \theta_1 = \theta_2 = \theta_5 = 0$, $\theta_3 = \theta_4 = 1$ 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 în afara lui), obiectul de coordonate $(x_1, x_2)^t$ este estimat ca fiind de clasă negativă, respectiv pozitivă. Am arătat deci că suprafața de separare poate fi neliniară, dacă se introduc trăsături suplimentare.

3.2.4 Funcția de cost

Funcția care ne permite să decidem cât de bun este un vector de ponderi $\boldsymbol{\theta}$ și care e totodată utilizată pentru ajustarea ponderilor în procesul de instruire este notată tradițional cu $J(\cdot)$, $J : \mathbf{R}^{n+1} \to \mathbf{R}_+$. Argumentul ei este vectorul de ponderi $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^t \in \mathbf{R}^{n+1}$. Valoarea se va calcula peste setul de instruire $\mathscr{S}$ din (3.2), sau peste orice mulțime cu perechi formate din vectori de intrare si clasă de ieșire asociată.

Considerarea mediei numărului de clasificări corecte nu e o soluție: funcția care calculează acuratețea este discretă, producând valori în mulțimea $\{0, \frac{1}{m}, \frac{2}{m} \ldots, 1\}$, și deci nu permite calcul de derivate parțiale.

O altă variantă este dată de utilizarea aceleiași funcții de eroare din capitolul de regresie liniară, ecuația (2.25) pagina 29. Se arată însă că pentru problema estimării de probabilitate condiționată, dată fiind forma funcției $h_{\boldsymbol{\theta}}$ din ecuația (3.3), funcția de cost nu mai este convexă și în acest caz o căutare bazată pe gradient se poate opri într–un minim local — situație exemplificată în figura 3.3.

Vom defini J de așa manieră încât să fie convexă, deci fără minime locale. Pentru fiecare pereche de intrare-ieșire asociată $(\mathbf{x}^{(i)}), y^{(i)})$ vom considera termen de eroare $Cost\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}\right)$, iar funcția de cost va fi media acestor erori:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} Cost\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}\right) \tag{3.12}$$

și în plus funcția $Cost(\cdot, \cdot)$ să îndeplinească următoarele condiții de (di)similaritate[7]:

[6] În mod normal, valorile lui $\boldsymbol{\theta}$ se determină prin proces de instruire.

[7] În cele ce urmează, pentru simplificarea notațiilor vom renunța la indicele i.

Figura 3.3: Minim global și minim local pentru funcția neconvexă $f : [0, 15] \to \mathbf{R}$, $f(x) = 15 + \sin(x) \cdot x + \frac{1}{x^2+3}$

1. condiția de apropiere: dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ și y sunt valori apropiate, atunci valoarea lui $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie apropiată de 0, și reciproc;

2. condiția de depărtare: dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ și y sunt valori îndepărtate, atunci valoarea lui $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie mare, și reciproc;

Definim convenabil funcția $Cost(\cdot, \cdot)$ astfel:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\ln h_{\boldsymbol{\theta}}(\mathbf{x}) & \text{dacă } y = 1 \\ -\ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{dacă } y = 0 \end{cases} \tag{3.13}$$

logaritmii fiind în baza e.

Cele două ramuri ale funcției $Cost$ sunt reprezentate în figura 3.4. Dreptele $x = 0$ și respectiv $x = 1$ sunt asimptote verticale pentru cele două ramuri ale lui $Cost$.

Rescriem funcția $Cost$ sub forma echivalentă și mai compactă:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = -y \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}) - (1 - y) \cdot \ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) \tag{3.14}$$

$$= -y \cdot \ln \hat{y} - (1 - y) \cdot \ln(1 - \hat{y}) \tag{3.15}$$

Figura 3.4: Cele două ramuri ale funcției *Cost* din ecuația (3.13)

Să verificăm că se îndeplinesc cele două condiții cerute mai sus. Pentru condiția de apropiere, vrem să verificăm că:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) \approx 0 \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx y \tag{3.16}$$

Pentru cazul $y = 1$ avem:

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

Pentru cazul $y = 0$, (3.16) devine:

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

deci prima condiție, legată apropiere de 0 a lui $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ este îndeplinită de definiția din ecuația (3.15).

Pentru condiția de depărtare, dacă[8] $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = M \gg 0$, atunci obținem echivalent $h_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(-M)$ (pentru $y = 1$) respectiv $h_{\boldsymbol{\theta}}(\mathbf{x}) = 1 - \exp(-M)$ (pentru $y = 0$); y este unul din capetele intervalului $[0, 1]$, iar dacă M este o valoare din ce în ce mai mare, atunci $h_{\boldsymbol{\theta}}(\mathbf{x})$ se îndreaptă spre

[8] Relația $\gg$ este "mult mai mare decât".

celălalt capăt al intervalului. Rezultă deci că și această a doua condiție este îndeplinită de definiția din formula (3.15).

În plus, deoarece fiecare din cele două ramuri ale funcției de cost din (3.13) e convexă, rezultă că de fapt funcția *Cost* este convexă (ea e una din cele două ramuri, în funcție de valoarea lui $y^{(i)}$). Mai departe, 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.

Funcția de eroare J se rescrie astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \cdot \ln\left(1 - h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right)\right] \tag{3.17}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \cdot \ln\left(1 - \hat{y}^{(i)}\right)\right] \tag{3.18}$$

și acestă formă se numește binary cross–entropy.

Dacă considerăm vectorii coloană $\mathbf{y} = \left(y^{(1)}, \ldots, y^{(m)}\right)^t$ și vectorul $\hat{\mathbf{y}}$ din ecuația (3.10) pagina 37, funcția de eroare binary cross–entropy se rescrie vectorizat ca:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left(\mathbf{y}^t \cdot \ln \hat{\mathbf{y}} + (\mathbb{1}_m - \mathbf{y})^t \cdot \ln(\mathbb{1}_m - \hat{\mathbf{y}})\right) \tag{3.19}$$

unde $\mathbb{1}_m$ este vectorul coloană cu m componente, toate egale cu 1, iar funcția logaritm se aplică punctual, pe fiecare componentă a vectorului argument. Observăm că se folosesc produse scalare de vectori, de exemplu $\mathbf{y}^t \cdot \ln \hat{\mathbf{y}}$, deci $J(\boldsymbol{\theta})$ este într-adevăr un număr.

3.2.5 Algoritmul de instruire

Setul de antrenare $\mathscr{S}$ este utilizat pentru a deduce valori adecvate ale lui $\boldsymbol{\theta}$, astfel încât predicțiile $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ date de model pentru valorile de intrare $\mathbf{x}^{(i)}$ să fie cât mai apropiate de valorile actuale ale etichetelor corespunzătoare $y^{(i)}$. Datorită proprietăților funcției *Cost* din ecuația (3.13) 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 $\hat{y}^{(i)} = h_{\boldsymbol{\theta}}(x^{(i)})$ apropiate de $y^{(i)}$.

Pentru determinarea lui $\boldsymbol{\theta}^{(min)}$ care minimizează funcția J:

$$\boldsymbol{\theta}^{(min)} = \arg\min_{\boldsymbol{\theta} \in \mathbf{R}^{n+1}} J(\boldsymbol{\theta}) \tag{3.20}$$

se folosește algoritmul de căutare după direcția gradientului (eng: gradient descent): se pornește cu valori aleator setate componentelor vectorului $\boldsymbol{\theta}$ – sau chiar cu vectorul nul – și se modifică în mod iterativ, scăzând la fiecare pas valoarea gradientului înmulțită cu un coeficient pozitiv mic α, numit rată de învățare.

Algoritmul de instruire are forma:

1. Setează componentele lui $\boldsymbol{\theta}$ la valori inițiale aleatoare în jurul lui zero sau chiar zero;

2. `repeta{`

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

`} pana la convergenta`

Condiția de convergență poate avea forma:

- valorile succesive ale funcției de eroare scad și diferența dintre două valori succesive este sub un prag $\varepsilon > 0$ specificat a priori;
- norma diferenței L_2 dintre două valori succesive ale vectorului $\boldsymbol{\theta}$ este sub un prag $\varepsilon > 0$; cu alte cuvinte, distanța dintre doi vectori succesivi de ponderi devine foarte mică;
- (mai rar) se atinge un număr maxim de iterații, specificat a priori.

Derivatele parțiale $\partial J / \partial \theta_j$ sunt:

$$\frac{\partial J}{\partial \theta_j}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) \cdot x_j^{(i)} \right] = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\hat{y}^{(i)} - y^{(i)} \right) \cdot x_j^{(i)} \right] \tag{3.21}$$

și modificarea din interiorul iterației devine:

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

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

Atribuirile simultane cerute pentru $j = 0, 1, \ldots n$ se fac fie prin utilizarea unor variabile temporare θ_j^{temp}, fie prin folosirea de calcul vectorizat. Se arată destul de ușor că gradientul lui J este:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \mathbf{X}^t (\hat{\mathbf{y}} - \mathbf{y}) \tag{3.22}$$

și deci modificarea din interiorul iterațiilor de instruire devine:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{1}{m} \mathbf{X}^t (\hat{\mathbf{y}} - \mathbf{y}) \tag{3.23}$$

unde vectorul $\hat{\mathbf{y}}$ e calculat ca în (3.10).

Se păstrează discuția legată de valorile ratei de învățare α (dinamica funcției de eroare, valori α prea mari sau prea mici) din secțiunea 2.3, pagina 26.

Nu există o formă analitică de determinare a coeficienților, cum aveam la regresia liniară (metoda ecuațiilor normale).

3.3 Regresia logistică multinomială

Pentru cazul în care se cere discriminarea pentru K clase, $K > 2$, regresia logistică se extinde să construiască K funcții (modele) simultan:

$$h_{\mathbf{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y=1|\mathbf{x};\mathbf{\Theta}) \\ P(y=2|\mathbf{x};\mathbf{\Theta}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{\Theta}) \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{pmatrix} = \hat{\mathbf{y}} \tag{3.24}$$

Facem o comparație între valoarea de ieșire a acestui model multinomial și ieșirea produsă de regresia logistică binară: pentru două clase e suficientă construirea unei singure funcții, sigmoida logistică, producând o singură valoare. Pentru $K > 2$ clase ar fi suficientă construirea a $K-1$ funcții, dar pentru comoditate se preferă construirea a K funcții, oricare din ele fiind acceptată ca redundantă: suma celor K funcții fiind 1, fiecare din ele poate fi determinată ca unu minus suma celorlalte $K-1$. Detaliem acest aspect în secțiunile 3.4.1 și 3.4.2.

3.3.1 Setul de instruire

Setul de instruire suferă modificări doar pentru valorile lui $y^{(i)}$, care acum pot fi din mulțimea $\{1, \ldots, K\}$. Formal, setul de instruire se scrie ca:

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | 1 \leq i \leq m \right\} \tag{3.25}$$

unde vectorul de intrare $\mathbf{x}^{(i)} \in \mathbf{R}^{n+1}$ – ca și la regresia logistică binară – iar $y^{(i)} \in \{1, \ldots, K\}$.

3.3.2 Funcția softmax

În cazul regresiei logistice cu două clase s–a folosit sigmoidă logistică, producându–se valori de ieșire ce pot fi folosite direct ca probabilități condiționate. Pentru mai mult de două clase, estimarea de probabilitate condiționată se face cu funcția *softmax*, care transformă un vector oarecare de K numere într-un vector de K valori din intervalul $(0, 1)$ și care însumate dau 1 – un așa-numit vector stohastic.

Pornind de la vectorul $\mathbf{z} = (z_1, \ldots, z_K)^t \in \mathbf{R}^K$, el e transformat de funcția *softmax* astfel:

$$softmax(\mathbf{z}) = (softmax(\mathbf{z}; 1), \ldots, softmax(\mathbf{z}; K))^t \tag{3.26}$$

unde $softmax(\mathbf{z}; k)$ este:

$$softmax(\mathbf{z}; k) = \frac{\exp(z_k)}{\sum\limits_{l=1}^{K} \exp(z_l)} \tag{3.27}$$

pentru $1 \leq k \leq K$. Se verifică ușor că $softmax(\mathbf{z}; k) \in (0, 1)$ pentru orice k, $1 \leq k \leq K$ și $\sum_{k=1}^{K} softmax(\mathbf{z}; k) = 1$, adică vectorul $softmax(\mathbf{z})$ este stohastic. Vom folosim funcția *softmax* pentru producerea de estimări de probabilități condiționate de intrarea curentă $\mathbf{x}$.

3.3.3 Reprezentarea modelului

Ponderile instruibile care definesc modelul de clasificare se grupează într–o matrice $\mathbf{\Theta} \in \mathbf{R}^{(n+1)\times K}$, construită ca:

$$\mathbf{\Theta} = \begin{bmatrix} | & | & | & | \\ \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \cdots & \boldsymbol{\theta}_K \\ | & | & | & | \end{bmatrix} \tag{3.28}$$

unde vectorul coloană $\boldsymbol{\theta}_k \in \mathbf{R}^{n+1}$ conține ponderile (parametrii instruibili) pentru clasa k, $1 \leq k \leq K$.

Modelul care conține toate cele K modele, câte unul pentru fiecare clasă, are forma:

$$h_{\mathbf{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y = 1|\mathbf{x}; \mathbf{\Theta}) \\ P(y = 2|\mathbf{x}; \mathbf{\Theta}) \\ \vdots \\ P(y = K|\mathbf{x}; \mathbf{\Theta}) \end{pmatrix} = softmax \begin{pmatrix} \boldsymbol{\theta}_1^t \cdot \mathbf{x} \\ \boldsymbol{\theta}_2^t \cdot \mathbf{x} \\ \vdots \\ \boldsymbol{\theta}_K^t \cdot \mathbf{x} \end{pmatrix} = \tag{3.29}$$

$$= softmax(\mathbf{\Theta}^t \cdot \mathbf{x}) = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{pmatrix} = \hat{\mathbf{y}} \tag{3.30}$$

pentru un vector de intrare $\mathbf{x} \in \mathbf{R}^{n+1}$. Valorile $\boldsymbol{\theta}_k^t \cdot \mathbf{x}$ se mai numesc și logiți.

Probabilitatea ca un vector $\mathbf{x}$ să fie de clasă k ($1 \leq k \leq K$) este:

$$P(y = k|\mathbf{x}; \mathbf{\Theta}) = \frac{\exp(\boldsymbol{\theta}_k^t \cdot \mathbf{x})}{\sum\limits_{l=1}^{K} \exp(\boldsymbol{\theta}_l^t \cdot \mathbf{x})} = \hat{y}_k \in (0, 1) \tag{3.31}$$

Pentru clasificarea unui vector $\mathbf{x}$ dat la intrare, se calculează probabilitatea de apartenență la clasa k, $P(y = k|\mathbf{x}; \mathbf{\Theta})$ pentru toți k între 1 și K și se alege acel $k^{(max)}$ care realizează probabilitatea maximă:

$$k^{(max)} = \arg\max_{1 \leq k \leq K} P(y = k|\mathbf{x}; \mathbf{\Theta}) = \arg\max_{1 \leq k \leq K} \hat{y}_k \tag{3.32}$$

Predicția făcută de model este că $\mathbf{x}$ aparține clasei $k^{(max)}$. O decizie similară era folosită și pentru regresia logistică binară.

Instruirea vizează determinarea acelei matrice $\mathbf{\Theta}$ pentru care modelul învață să recunoască cât mai bine datele din setul de instruire (eventual adăugându–se și factor de regularizare, a se vedea capitolul 4). Mai clar, dacă un obiect $\mathbf{x}$ din setul de instruire $\mathscr{S}$ este de clasă k, atunci vrem ca valoarea $P\left(y = k|\mathbf{x}; \mathbf{\Theta}\right)$ să fie cât mai aproape de 1.

3.3.4 Funcția de cost

Ca și în cazul funcției de cost (de eroare) pentru regresia logistică, se va cuantifica în ce măsură diferă clasa actuală a unei intrări de predicția dată de modelul $h_{\boldsymbol{\theta}}$. Introducem funcția indicator $I(\cdot)$ care pentru o valoare logică returnează 1 dacă argmentul este adevărat și 0 altfel:

$$I(valoare_logica) = \begin{cases} 1 & \text{dacă } valoare_logica = adevarat \\ 0 & \text{dacă } valoare_logica = fals \end{cases} \tag{3.33}$$

Funcția de eroare pentru regresia logistică multinomială se definește ca:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)} = k\right) \cdot \ln \hat{y}_k^{(i)}\right] \tag{3.34}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)} = k\right) \cdot \ln P\left(y^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{\Theta}\right)\right] \tag{3.35}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)} = k\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{l=1}^{K}\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)}\right] \tag{3.36}$$

și se numește cross–entropy.

Pentru reprezentarea clasei curente $y^{(i)}$ se folosește frecvent așa–numita codificare one-hot (unul din K), astfel: dacă $y^{(i)} = k$, atunci codificarea sa $\mathbf{y}_{ohe}^{(i)}$ este un vector coloană cu K valori, avand pe poziția k valoarea 1 şi 0 în rest. Cu această codificare se rescrie funcția de eroare mai compact ca:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{y}_{ohe}^{(i)}\right)^t \cdot \ln\left(\hat{\mathbf{y}}^{(i)}\right) \tag{3.37}$$

unde logaritmul se aplică pe fiecare componentă[9] a vectorului $\hat{\mathbf{y}}^{(i)}$.

În cazul în care se folosesc biblioteci de calcul vectorizat, se poate folosi produsul Hadamard în locul celui algebric: dacă $\mathbf{A}$ și $\mathbf{B}$ sunt 2 matrice de tip $m \times n$, atunci produsul Hadamard (sau punctual) al lor este $\mathbf{C}$ de același tip, $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$, $c_{ij} = a_{ij} \cdot b_{ij}$, $1 \leq i \leq m, 1 \leq j \leq n$. Dacă considerăm

[9] Datorită valorilor apropiate de 0 pe care le pot lua componentele lui $\hat{y}^{(i)}$, la implementare trebuie făcută o aducere a valorilor mici la niște praguri convenabile – *clipping*.

matricele $\mathbf{Y}_{ohe}$, respectiv $\hat{\mathbf{Y}}$ de tip $K \times m$, având fiecare coloana i vectorul $\mathbf{y}_{ohe}^{(i)}$, respectiv $\hat{\mathbf{y}}^{(i)}$, atunci putem considera matricea $-\frac{1}{m} \cdot \mathbf{Y}_{ohe} \odot \ln \hat{\mathbf{Y}}$ pentru care se calculează apoi suma tuturor elementelor folosind funcții eficiente din bibliotecă.

Alternativ, suma elementelor unei matrice $\mathbf{S}$ de dimensiune $m \times n$ se poate obține cu:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} S_{ij} = \mathbb{1}_m^t \cdot \mathbf{S} \cdot \mathbb{1}_n \tag{3.38}$$

unde $\mathbb{1}_p$ este vector coloană de p elemente egale cu 1. Funcția de eroare din ecuatia (3.34) se rescrie ca:

$$J(\mathbf{\Theta}) = -\frac{1}{m} \mathbb{1}_m^t \cdot \left(\mathbf{Y}_{ohe} \odot \ln \hat{\mathbf{Y}} \right) \cdot \mathbb{1}_K \tag{3.39}$$

Secțiunea 3.4 conține alte discuții legate de calculul funcției de eroare.

3.3.5 Gradientul funcției de cost

Determinarea lui $\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbf{R}^{m \times (n+1)}} J(\boldsymbol{\theta})$ se face prin căutarea după direcția gradientului. Formula gradientului este, pentru $1 \leq k \leq K$:

$$\nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(\hat{y}^{(i)} - I(y^{(i)} = k) \right) \right] = \tag{3.40}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(P(y^{(i)} = k | \mathbf{x}^{(i)}; \mathbf{\Theta}) - I(y^{(i)} = k) \right) \right] \tag{3.41}$$

pe care o demonstrăm în cele ce urmează: Având în vedere că gradientul e operator liniar, calculăm gradientul pentru un termen de cost corespunzător unei perechi de antrenare $\left(\mathbf{x}^{(i)}, y^{(i)} \right)$, $J_i(\mathbf{\Theta})$:

$$J_i(\mathbf{\Theta}) = -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)}{\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} = \tag{3.42}$$

$$= -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln \left(\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right) \right) + \tag{3.43}$$

$$+ \sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln \left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right) \right) = \tag{3.44}$$

$$= -\sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)} \right] + \tag{3.45}$$

$$+ \sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \ln \left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right) \right) \right] \tag{3.46}$$

În suma din linia (3.46), termenul cu logaritm nu depinde de indicele de însumare l, deci poate fi scos în fața sumei; mai departe, pentru suma rămasă, pentru exact un indice l avem $I\left(y^{(i)} = l\right) = 1$ și în rest indicatorii sunt 0, deci suma e 1. $J_i(\mathbf{\Theta})$ devine:

$$J_i(\mathbf{\Theta}) = -\sum_{l=1}^{K}\left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right] + \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.47}$$

Aplicând gradientul, obținem:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}_k} J_i(\mathbf{\Theta}) &= -\nabla_{\boldsymbol{\theta}_k} \sum_{l=1}^{K}\left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right] + \nabla_{\boldsymbol{\theta}_k} \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \\
&= -I\left(y^{(i)} = k\right)\mathbf{x}^{(i)} + \frac{\nabla_{\boldsymbol{\theta}_k} \sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} && (3.48) \\
&= -I\left(y^{(i)} = k\right)\mathbf{x}^{(i)} + \frac{\mathbf{x}^{(i)} \exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} && (3.49) \\
&= -I\left(y^{(i)} = k\right)\mathbf{x}^{(i)} + \mathbf{x}^{(i)} P(y^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{\Theta}) && (3.50) \\
&= \mathbf{x}^{(i)}\left(P(y^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{\Theta}) - I\left(y^{(i)} = k\right)\right) && (3.51)
\end{aligned}$$

sau forma mai compactă

$$\nabla_{\boldsymbol{\theta}_k} J_i(\mathbf{\Theta}) = \mathbf{x}^{(i)}\left(\hat{y}^{(i)} - I\left(y^{(i)} = k\right)\right). \tag{3.52}$$

Calculând media celor m termeni $\nabla_{\boldsymbol{\theta}_k} J_i(\mathbf{\Theta})$ se obține formula gradientului total $\nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta})$ din ecuația (3.41).

Calculul de gradienți pentru toată matricea $\boldsymbol{\theta}$ este:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\mathbf{X}^t \cdot (\hat{\mathbf{Y}} - \mathbf{Y}) \tag{3.53}$$

3.3.6 Algoritmul de instruire

Modificarea vectorului $\boldsymbol{\theta}_k$ este:

$$\begin{aligned}
\boldsymbol{\theta}_k &= \boldsymbol{\theta}_k - \alpha \cdot \nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta}) && (3.54) \\
&= \boldsymbol{\theta}_k - \frac{\alpha}{m}\sum_{i=1}^{m}\left[\mathbf{x}^{(i)}\left(\hat{y}^{(i)} - I(y^{(i)} = k)\right)\right] && (3.55)
\end{aligned}$$

simultan pentru toți $1 \leq k \leq K$. Modificarea se face iterativ, până când matricea $\mathbf{\Theta}$ se stabilizează[10], funcția de eroare evoluează prea puțin[11], sau se atinge un număr maxim de iterații.

Algoritmul de instruire are forma:

1. Setează componentele lui $\mathbf{\Theta}$ la valori inițiale aleatoare mici (în jurul lui 0) sau chiar 0;

2. `repeta{`

$$\boldsymbol{\theta}_k := \boldsymbol{\theta}_k - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(\hat{y}^{(i)} - I(y^{(i)} = k) \right) \right] \tag{3.56}$$

`} pana la convergenta`

Matricial, algoritmul se rescrie ca:

1. Setează componentele lui $\mathbf{\Theta}$ la valori inițiale aleatoare mici (în jurul lui 0) sau chiar 0;

2. `repeta{`

$$\mathbf{\Theta} := \mathbf{\Theta} - \alpha \frac{1}{m} \mathbf{X}^t \cdot (\hat{\mathbf{Y}} - \mathbf{Y}_{ohe}) \tag{3.57}$$

`} pana la convergenta`

În formula (3.57) $\hat{\mathbf{Y}}$ și $\mathbf{Y}_{ohe}$ sunt cele definite la pagina 46. Spre deosebire de regresia liniară, nu există o variantă algebrică de determinare a valorilor din matricea $\mathbf{\Theta}$.

3.4 Precizări suplimentare

Elementele din această secțiune aduc precizări asupra modelelor de regresie logistică, precum și discuții legate de eficientizarea calculelor.

3.4.1 Redundanța parametrilor pentru regresia logistică multinomială

Pentru regresia logistică cu două clase a fost suficient un singur parametru $\boldsymbol{\theta}$. Pentru modelul de regresie logistică multinomială cu K clase se folosesc K vectori $\boldsymbol{\theta}$ — o discrepanță vizibilă, ne–am fi așteptat să fie suficienți

[10] Respectiv: norma diferenței dintre două versiuni succesive ale matricei $\mathbf{\Theta}$ devine mai mică decât un prag ε dat, mic și pozitiv. Drept normă se alege de regulă norma Frobenius.

[11] Mai clar: $|J(\mathbf{\Theta}_t) - J(\mathbf{\Theta}_{t-1})| < \varepsilon$, pentru un ε dat, pozitiv și mic.

$K-1$ vectori. Discuția care urmează arată că, într-adevăr, regresia logistică multinomială are redundanță de parametri.

Pentru o dată de intrare $\mathbf{x}$, modelul estimează probabilitatea condiționată ca:

$$P(y=k|\mathbf{x};\boldsymbol{\Theta}) = \frac{\exp(\boldsymbol{\theta}_k^t\mathbf{x})}{\sum\limits_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}\right)} \tag{3.58}$$

Dacă scădem din fiecare vector $\boldsymbol{\theta}_j$ un același vector $\boldsymbol{\psi}$, obținem:

$$\frac{\exp\left((\boldsymbol{\theta}_k-\boldsymbol{\psi})^t\mathbf{x}\right)}{\sum\limits_{j=1}^{K}\exp\left((\boldsymbol{\theta}_j-\boldsymbol{\psi})^t\mathbf{x}\right)} = \frac{\exp(-\boldsymbol{\psi}^t\mathbf{x})\exp\left(\boldsymbol{\theta}_k^t\mathbf{x}\right)}{\exp(-\boldsymbol{\psi}^t\mathbf{x})\sum\limits_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}\right)} = \tag{3.59}$$

$$= \frac{\exp(\boldsymbol{\theta}_k^t\mathbf{x})}{\sum\limits_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}\right)} = P(y=k|\mathbf{x};\boldsymbol{\Theta}) \tag{3.60}$$

deci predicțiile vor fi aceleași ca la setul de parametri originar; mai mult, dacă avem vectorii $\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K$ care minimizează funcția de cost J, atunci aceeași valoare minimă poate fi atinsă cu $\boldsymbol{\theta}_1-\boldsymbol{\psi},\ldots,\boldsymbol{\theta}_K-\boldsymbol{\psi}$ pentru orice vector $\boldsymbol{\psi}$. Dacă pe post de $\boldsymbol{\psi}$ se ia oricare din vectorii $\boldsymbol{\theta}^k$, de exemplu $\boldsymbol{\theta}^K$, atunci vectorul de parametri devine neredundant, $\boldsymbol{\theta}_1-\boldsymbol{\theta}_K,\boldsymbol{\theta}_2-\boldsymbol{\theta}_K,\ldots,\mathbf{0}$ cu același comportament ca mai înainte. Sunt deci suficienți $K-1$ vectori de ponderi. Acceptăm însă redundanța, pentru simplitatea și simetria formulelor rezultate. Redundanța este demonstrată și mai jos pentru cazul particular $K=2$, când se regăsește regresia logistică binară.

O observație interesantă e: chiar și în cazul cu redundanță funcția de eroare J rămâne încă funcție convexă, cu o infinitate de valori minime și egale între ele. Chiar și așa, funcția de eroare nu are minime locale mai mari decât minimul global, deci situația din figura 3.3 nu apare.

3.4.2 Relația dintre cele două tipuri de regresii logistice

Pentru regresia logistică multinomială, e de așteptat ca să regăsim regresia logistică binară pentru cazul $K=2$. Într-adevăr, ecuația (3.30) devine:

$$h_\Theta(\mathbf{x}) = \frac{1}{\exp(\boldsymbol{\theta}_1^t\mathbf{x})+\exp(\boldsymbol{\theta}_2^t\mathbf{x})}\begin{pmatrix}\exp(\boldsymbol{\theta}_1^t\mathbf{x})\\ \exp(\boldsymbol{\theta}_2^t\mathbf{x})\end{pmatrix} \tag{3.61}$$

$$= \frac{1}{\exp(\boldsymbol{\theta}_1^t\mathbf{x})+\exp(\boldsymbol{\theta}_2^t\mathbf{x})}\frac{\exp(\boldsymbol{\theta}_1^t\mathbf{x})}{\exp(\boldsymbol{\theta}_1^t\mathbf{x})}\begin{pmatrix}\exp(\boldsymbol{\theta}_1^t\mathbf{x})\\ \exp(\boldsymbol{\theta}_2^t\mathbf{x})\end{pmatrix} \tag{3.62}$$

$$= \frac{1}{\exp\left((\boldsymbol{\theta}_1-\boldsymbol{\theta}_1)^t\mathbf{x}\right)+\exp\left((\boldsymbol{\theta}_2-\boldsymbol{\theta}_1)^t\mathbf{x}\right)}\begin{pmatrix}\exp((\boldsymbol{\theta}_1-\boldsymbol{\theta}_1)^t\mathbf{x})\\ \exp((\boldsymbol{\theta}_2-\boldsymbol{\theta}_1)^t\mathbf{x})\end{pmatrix} \tag{3.63}$$

Notând $\boldsymbol{\theta} = \boldsymbol{\theta_1} - \boldsymbol{\theta_2}$, avem:

$$h_\Theta(\mathbf{x}) = \frac{1}{1 + \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right)} \begin{pmatrix} 1 \\ \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right) \end{pmatrix} \tag{3.64}$$

$$= \begin{pmatrix} \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ 1 - \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \end{pmatrix} \tag{3.65}$$

$$= \begin{pmatrix} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \\ 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \end{pmatrix} \tag{3.66}$$

Tot pentru $K = 2$, se verifică ușor că funcția de eroare (3.36) devine cea din (3.18).

3.4.3 Calculul numeric al funcției softmax

La implementarea directă a funcție softmax se poate întâmpla ca valoarea funcției exponențiale să depășească posibilitățile de reprezentare numerică, pentru valori z_l pozitive mari. Un truc de calcul frecvent aplicat în practică este următorul: în loc de a se calcula funcția softmax precum în ecuația (3.27), se calculează prima dată valoarea maximă din $(z_1, \ldots, z_K)^t$, fie ea M, apoi se calculează softmax pentru vectorul de valori $\mathbf{z}' = (z_1 - M, \ldots, z_K - M)^t$. Avem că:

$$softmax(\mathbf{z}'; l) = \frac{\exp(z_l - M)}{\sum\limits_{k=1}^{K} \exp(z_k - M)} = \frac{\exp(z_l)/\exp(M)}{\sum\limits_{k=1}^{K} [\exp(z_k)/\exp(M)]} =$$

$$= \frac{\exp(z_l)}{\sum\limits_{k=1}^{K} \exp(z_k)} = softmax(\mathbf{z}; l) \tag{3.67}$$

Rezultatul e același, dar calculul se face cu exponenți mai mici, dacă $M > 0$ (de unde rezultă $z_l - M < z_l$). Se evită astfel depășirea de reprezentare (overflow) dacă vectorul $\mathbf{z}$ are componente pozitive foarte mari.

3.4.4 Trucul "log sum exp"

Funcția de eroare pentru regresia logistică multinomială are forma:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \left[\sum_{i=1}^{m} \sum_{k=1}^{K} I\left(y^{(i)} = k\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} \right] \tag{3.68}$$

Termenul cu logaritm permite simplificări de calcule:

$$\ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} = \ln\left(\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)\right) - \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.69}$$

$$= \boldsymbol{\theta}_k^t \mathbf{x}^{(i)} - \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.70}$$

Pentru al doilea termen din (3.70) se folosește un truc asemănător celui din subsecțiunea precedentă, pentru a evita calcul de funcție exponențială cu exponent mare. Trucul "log sum exp" se bazează pe identitatea:

$$\ln\left(\sum_{j=1}^{K} \exp(z_i)\right) = a + \ln\left(\sum_{j=1}^{K} \exp(z_i - a)\right) \tag{3.71}$$

pentru un a oarecare, demonstrată în cele ce urmează:

$$\ln\left(\sum_{j=1}^{K} \exp(z_i)\right) = \ln\left(\sum_{j=1}^{K} (\exp(z_i)\exp(a)\exp(-a))\right) = \tag{3.72}$$

$$= \ln\left(\exp(a) \cdot \sum_{j=1}^{K} (\exp(z_i)\exp(-a))\right) = \tag{3.73}$$

$$= \ln\exp(a) + \ln\left(\sum_{j=1}^{K} \exp(z_i - a)\right) = \tag{3.74}$$

$$= a + \ln\left(\sum_{j=1}^{K} \exp(z_i - a)\right) \tag{3.75}$$

Pentru a reduce din magnitudinea exponenților considerăm, în mod convenabil, $a = \max_i z_i$ și reducem riscul de depășire de reprezentare numerică (overflow).

Trucurile din secțiunea curentă și cea precedentă explică și de ce în unele implementări de biblioteci pentru machine learning, pentru funcția de eroare se cer valorile vectorului $\mathbf{z}$ (eng: logits) și nu valorile calculate de $softmax$.

Capitolul 4

Regularizarea modelelor

4.1 Overfitting, underfitting pentru regresia liniară

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 $suprafata$ – notată mai jos cu x – și variabila de ieșire $pret$; punctele sunt reprezentate în figura 4.1. Să considerăm 3 modele de predicție:

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

$$pret = \theta_0 + \theta_1 x \tag{4.1}$$

2. polinom de gradul al doilea:

$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{4.2}$$

3. polinom de gradul 4:

$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{4.3}$$

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 polonomiale (de grad 2, respectiv 4). Ca și mai înainte, se pune problema determinării coeficienților $\theta_0, \theta_1, \ldots$. Modelul este determinat tot prin regresie liniară, pe baza datelor multivariate.

Graficele celor trei forme polinomiale sunt date în figura 4.1 [6].

Intuitiv, polinomul de gradul întâi nu reușeste să facă o aproximare prea bună a evoluției prețului în funcție de suprafață. Spunem că modelul dat de primul polinom suferă de "underfitting"[1], fiind prea simplu pentru problema

[1]Aproximativ, în limba română: incapacitate de reprezentare; simplism.

Figura 4.1: Trei polinoame pentru aproximarea prețului pornind de la suprafață, notată x [6].

în cauză. Se mai spune despre un asemenea model că are "high bias"[2], deoarece face o presupunere simplistă pentru problema tratată.

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 $\theta_0, \ldots, \theta_4$ astfel încât curba să treacă prin toate cele 5 puncte (interpolare polinomială, vezi de exemplu polinoamele Lagrange). Remarcăm însă forma nemonotonă, cu variații mari a predicției, fiind cazuri în care valoarea estimată scade în raport cu suprafața. Aici modelul suferă de "overfitting"[3], fiind prea fidel 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 intre al treilea și al patrulea punct din grafic). Se mai spune că modelul are varianță (variabilitate) mare[4], datorită faptului că e prea complex pentru problema tratată. Complexitatea crescută e dată de numărul de mare de coeficienți θ, rezultați din gradul mare al polinomului de aproximare.

Intuitiv, pentru problema dată polinomul de gradul 2 prezintă cea mai credibilă 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 setul de antrenare, cât și de a face estimări pentru alte valori. Un model care suferă de "overfitting" poate învăța foarte bine datele din setul de instruire, dar nu reușește să facă estimări prea bune în afara lui; în ambele cazuri 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. Pentru modele de predicție, în final, ne interesează cât de bine pot fi

[2]Înclinare prea pronunțată spre un model cu performanță predictivă slabă.
[3]Supraspecializare
[4]Eng: high variance.

estimate situațiile din afara setului de instruire, *i.e.* generalizarea modelului.

Exemplificarea s–a făcut plecând de la o variabilă predictivă x reprezentând suprafața, care produce alte atribute: 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. O altă opțiune populară – și discutată aici – este regularizarea: se păstrează variabilele predictive, dar se impun constrângeri asupra parametrilor modelului — în cazul nostru asupra coeficienților θ_i.

4.2 Regularizarea L_2 pentru regresia liniară

Să considerăm că predicția se formează pe baza funcției polinomiale

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{4.4}$$

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 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:

$$J(\boldsymbol{\theta}) = \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right\} + 100 \cdot \theta_3^2 + 100 \cdot \theta_4^2 \tag{4.5}$$

Minimizarea lui J din ec. (4.5) va urmări simultan micșorarea diferenței dintre valorile estimate de model și cele reale, dar și reducerea valorilor absolute 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 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 înmulțeste cu nicio variabilă predictivă; dacă avem overfit, nu e din cauza lui); vom impune deci constrângeri doar asupra lui $\theta_1, \theta_2, \ldots$ – să aibă valori absolute mici; echivalent, pătratele lor să fie mici, deci suma pătratelor lor să nu fie prea mare. Scopul final este de a evita overfitting–ul. Principiul se

aplică și dacă se pleacă de la variabile de intrare independente, nu neapărat $suprafata, suprafata^2, \ldots$. Ca atare, ec. (4.5) se rescrie mai general ca:

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\right]}_{\text{eroarea de calitate}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{\text{termen de regularizare}} \tag{4.6}$$

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

În formă matricială, funcția de eroare incluzând termenul de regularizare se scrie ca:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left(X\boldsymbol{\theta} - \boldsymbol{y}\right)^t\left(X\boldsymbol{\theta} - \boldsymbol{y}\right) + \frac{\lambda}{2}\left\|\theta[1:]\right\|^2 \tag{4.7}$$

unde $\|\cdot\|$ este norma L_2 (Euclidiană) a vectorului argument. Se observă omiterea termenului liber din calculul de normă de vector. Folosirea normei Euclidiene dă denumirea regularizării: regularizare L_2.

Algoritmul de căutare după direcția gradientului devine (considerăm că avem coeficienții $\theta_0, \ldots, \theta_n$ inițializați aleator sau chiar cu vectorul nul):

```
repeta{
```

$$\theta_0 := \theta_0 - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x_0^{(i)}\right] \tag{4.8}$$

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)} + \lambda\cdot\theta_j\right],\ j = 1, \ldots, n \tag{4.9}$$

```
} pana la convergenta
```

unde atribuirile se efectuează în mod simultan. În formă vectorială scriem:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \boldsymbol{\theta} - \alpha\left[\frac{1}{m}\mathbf{X}^t(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \lambda[0, \theta_1, \ldots, \theta_n]^t\right] \tag{4.10}$$

unde vectorul $[0, \theta_1, \ldots, \theta_n]^t$ se formează din vectorul de ponderi $\boldsymbol{\theta}$ punând pe prima poziție valoarea 0.

Rescriem modificarea lui θ_j din (4.9):

$$\theta_j := \theta_j(1 - \alpha\lambda) - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)}\right] \tag{4.11}$$

Observăm mai sus o dublă modificare a ponderilor θ_j: pe de o parte se scade gradientul termenului de calitate a predicției înmulțit cu rata de învățare (gradient descent), dar se și scalează ponderea θ_j cu un termen multiplicativ $1 - \alpha\lambda$ care de regulă e subunitar; spunem că în acest al doilea caz avem un efect de decădere a ponderilor – eng: weight decay. Pentru metoda stochastic gradient descent (fără termen momentum) regularizarea L_2 este deci echivalentă cu weight decay. Pentru alte metode de optimizare afirmația nu mai este valabilă [7].

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

$$\boldsymbol{\theta} = \left(\mathbf{X}^t\mathbf{X} + \lambda \cdot \underbrace{\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}}_{(n+1)\times(n+1)} \right)^{-1} \cdot \mathbf{X}y \tag{4.12}$$

unde matricea care se înmulțește cu λ se obține din matricea unitate de ordinul $n+1$, modificând primul element în 0 (din același motiv ca până acum: termenul θ_0 nu se regularizează). Se poate arăta că pentru $\lambda > 0$ termenul din ecuația (4.12) pentru care se cere inversa este o matrice nesingulară, indiferent de valorile din matricea $\mathbf{X}$.

4.3 Regularizarea modelelor de regresie logistică

4.3.1 Regularizarea L_2 pentru regresia logistică binară

Pentru regresia logistică binară formula modelului este:

$$P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + exp(-\boldsymbol{\theta}^t\mathbf{x})} \tag{4.13}$$

cu vectorul de ponderi $\boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_n)$.

Dacă se permite ca valorile parametrilor[5] $\theta_1, \dots, \theta_n$ 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[6]. Ca atare, se preferă

[5] Se remarcă lipsa termenului liber θ_0; el nu este regularizat.

[6] Alfel zis: creștem numărul de trăsături din setul de antrenare, dar numărul de exemplare din acest set rămâne același: m.

impunerea unor constrângeri parametrilor $\theta_1, \ldots, \theta_n$ astfel încât aceștia să fie cât mai mici în valoare absolută.

Pentru regularizare se modifică forma funcției de eroare astfel:

$$J(\boldsymbol{\theta}) = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \cdot \ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right]}_{\text{Funcția de eroare din ecuația (3.17) pagina 41}} \tag{4.14}$$

$$+ \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{\text{termenul de regularizare}} \tag{4.15}$$

$$= \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln \hat{y}^{(i)} + (1 - y^{(i)}) \cdot \ln(1 - \hat{y}^{(i)})\right]}_{\text{Funcția de eroare (3.18) pagina 41}} \tag{4.16}$$

$$+ \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{\text{termenul de regularizare}} \tag{4.17}$$

Vectorizat se scrie astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left(\mathbf{y}^t \cdot \ln \hat{\mathbf{y}} + (\mathbb{1}_m - \mathbf{y})^t \cdot \ln(\mathbb{1}_m - \hat{\mathbf{y}})\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}[1:]\|_2^2 \tag{4.18}$$

unde $\boldsymbol{\theta}[1:]$ este vectorul format din toate componentele lui $\boldsymbol{\theta}$ mai puțin prima, iar $\|\mathbf{v}\|_2$ este norma L_2 (sau Euclidiană) a vectorului $\mathbf{v}$:

$$\|\mathbf{v}\|_2 = \sqrt{\sum_i v_i^2}$$

Modificarea adusă algoritmului de instruire este simplă: mai trebuie inclusă și derivata parțială a lui θ_j^2 în raport cu θ_j:

`repeta{`

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \tag{4.19}$$

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)} + \lambda\theta_j\right] \tag{4.20}$$

`} pana la convergenta`

unde atribuirile se fac simultan pentru indicii 0, 1, ..., n ai componentelor vectorului $\boldsymbol{\theta}$ – se folosesc variabile tempoare sau calcul vectorizat.

Forma vectorizată pentru algoritmul de instruire cu funcția de eroare incluzând regularizare L_2 se scrie ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \left[\frac{1}{m} \mathbf{X}^t (\hat{\mathbf{y}} - \mathbf{y}) + \lambda (0, \theta_1, \ldots, \theta_n)^t \right] \tag{4.21}$$

unde $(0, \theta_1, \ldots, \theta_n)^t$ este obținut vectorul $\boldsymbol{\theta}$ punând prima componentă 0.

Discuția din secțiunea de regualizare L_2 a regresiei liniare legată de echivalența regularizării L_2 și weight decay în algoritmul de gradient descent este valabilă și aici.

4.3.2 Regularizarea L_2 pentru regresia logistică multinomială

În practică se preferă penalizarea valorilor absolute mari ale parametrilor θ_{ik} $(1 \leq i \leq n, 1 \leq k \leq K)$; nu se penalizează ponderile de termeni liberi θ_{0k} $(1 \leq k \leq K)$. Se adaugă penalizări pentru pătratele valorilor ponderilor rămase și funcția de eroare devine:

$$J(\boldsymbol{\Theta}) = \underbrace{-\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[I\left(y^{(i)} = k\right) \cdot \ln \hat{y}_k^{(i)} \right]}_{\text{eroare de calitate}} + \underbrace{\frac{\lambda}{2} \sum_{i=1}^{n} \sum_{k=1}^{K} \theta_{ki}^2}_{\text{termen de regularizare}} \tag{4.22}$$

$$= -\frac{1}{m} \mathbb{1}_m^t \cdot \left(\mathbf{Y}_{ohe} \odot \ln \hat{\mathbf{Y}} \right) \cdot \mathbb{1}_K + \frac{\lambda}{2} \left\| \boldsymbol{\Theta}[\mathbf{1}:,:] \right\|_F^2 \tag{4.23}$$

unde $\|\mathbf{A}\|_F$ e norma Frobenius a matricei $\mathbf{A}_{m \times n}$:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2} \tag{4.24}$$

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}, fără a da vreo importanță prea mare nepotrivirilor între clasele estimate și cele reale. Evident, trebuie realizat un echilibru între aceste două extreme.

Formula gradientului folosit în căutarea de tip gradient descent este:

$$\nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{j=1}^{m} \left[\mathbf{x}^{(j)} \left(I(y^{(j)} = k) - \hat{y}^{(i)} \right) \right] + \lambda \cdot [0, \boldsymbol{\Theta}_{k,1:}]^t \tag{4.25}$$

unde $[0, \boldsymbol{\Theta}_{k,1:}]^t$ este vectorul coloană care are pe prima poziție 0 (termenii liberi nu se regularizează) și pe următoarele n poziții valorile $\Theta_{k1}, \ldots, \Theta_{kn}$.

Formula matricială pentru toți gradienții, pentru tot setul de date este:

$$\nabla_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}) = -\frac{1}{m} \mathbf{X}^t \cdot (\mathbf{Y}_{ohe} - \hat{\mathbf{Y}}) + \lambda \boldsymbol{\Theta}_0 \tag{4.26}$$

unde $\mathbf{\Theta}_0$ este matricea $\mathbf{\Theta}$ cu prima linie umplută cu 0.

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

Capitolul 5

Perceptronul liniar

5.1 Încadrare și exemplu

Un perceptron liniar este cel mai simplu tip de neuron, capabil să învețe să separe două clase de puncte care sunt liniar separabile. Instruirea este supervizată.

Pentru exemplificare, să presupunem că avem următoarele șase puncte în plan Cartezian: $(-4, 2), (-4, 6), (-2, 4), (3, -2), (5, -2), (4, 4)$. Primele trei puncte sunt de clasă negativă (*i.e.* -1) și restul de clasă pozitivă (*i.e.* +1). Reprezentarea lor este dată în figura 5.1. Observăm că pentru exemplul considerat putem găsi măcar o dreaptă – de exemplu dreapta d_1 de ecuație $4x - 3y + 12 = 0$ care are de o parte a ei doar cele 3 puncte de clasă pozitivă și de cealaltă parte a ei celelalte 3 de clasă negativă. Spunem că cele clase – pozitivă și negativă – sunt liniar separabile. De fapt, se poate să existe mai mult de o dreaptă de care să le separe: de exemplu, avem dreapta d_2 de ecuație $3x + y - 12 = 0$. Se poate arăta că dacă există măcar o dreaptă de separare, atunci există o infinitate de drepte care separă aceleași mulțimi de puncte.

Din geometria analitică, pentru un spațiu bidimensional, știm că o dreaptă de forma $ax + by + c = 0$ împarte un plan în două seminplane. Pentru toate punctele de coordonate (x_p, y_p) aflate de o aceeași parte a dreptei, expresia $ax_p + by_p + c$ este mereu strict mai mare (sau pentru partea cealaltă a dreptei: strict mai mică) decât zero. De exemplu, pentru punctele figurate cu "+" în figura 5.2 toate valorile sunt strict pozitive, iar cele figurate cu "-" produc valori strict negative. Spunem că semiplanele rezultate sunt pozitve, respectiv negative.

Exemplul a fost dat pentru puncte din spațiul bidimensional pentru a putea face reprezentarea grafică. Considerente similare sunt valabile în spații Euclidiene multidimensionale.

Figura 5.1: Pentru două mulțimi de puncte liniar separabile pot exista mai multe suprafețe de separare. În cazul dat, ambele drepte d_1 și d_2 separă punctele de clasă pozitive de cele de clasă negativă.

Figura 5.2: Orice dreaptă împarte un plan în două semiplane. Pentru o dreaptă de ecuație $d(x, y) = 0$ si pentru un semiplan fixat, orice punct de coordonate (x_p, y_p) din acel semiplan produce același semn pentru $d(x_p, y_p)$. Semnul rezultat dă natura semiplanului: pozitiv sau negativ.

5.2 Definiții și notații

Definiția 4. *(Clase liniar separabile) Mulțimile $\mathscr{C}_1$ și $\mathscr{C}_2$ din spațiul $\mathbb{R}^n$ se numesc liniar separabile dacă există o funcție $y : \mathbb{R}^n \to \mathbb{R}$ de forma:*

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i \cdot x_i + b \tag{5.1}$$

astfel încât:

$$\begin{cases} y(\mathbf{x}) > 0 \ \textit{pentru} \ \mathbf{x} \in \mathscr{C}_1 \\ y(\mathbf{x}) < 0 \ \textit{pentru} \ \mathbf{x} \in \mathscr{C}_2 \end{cases} \tag{5.2}$$

În condițiile de mai sus, numim clasa $\mathscr{C}_1$ clasă pozitivă, iar $\mathscr{C}_2$ clasă negativă.

Plecând de la un set de instruire format din mulțimile $\mathscr{C}_1$, $\mathscr{C}_2$ despre care se știe că sunt liniar separabile – dar pentru care funcția $y(\cdot)$ nu se cunoaște – perceptronul liniar determină ponderile (coeficienții) $b, w_1, \ldots, w_n$ pentru care condițiile din (5.2) sunt îndeplinite. Determinarea ponderilor se face prin învățare.

Ecuația $y(\mathbf{x}) = 0$, precum și mulțimile $\mathscr{C}_1$, $\mathscr{C}_2$ pe care y le separă au interpretări geometrice simple. Punctele $\mathbf{x} \in \mathbb{R}^n$ pentru care $y(\mathbf{x}) = 0$ formează o varietate liniară – notată cu S – de dimensiune $n-1$; dacă $b = 0$ atunci $y(\mathbf{x}) = 0$ este un hiperplan (subspațiu afin de dimensiune $n-1$ și deci trecând prin origine); prin abuz de limbaj, în literatură se folosește tot denumirea de hiperplan pentru suprafața $y(\mathbf{x}) = 0$, chiar dacă $b \neq 0$.

Suprafața liniară $y(\mathbf{x}) = 0$ separă planul în două submulțimi – în cazul $n = 2$, în două semiplane. Oricum am alege un punct $\mathbf{x}$ dintr-o submulțime (semiplan) oarecare, semnul lui $y(\mathbf{x})$ este mereu același, iar la trecerea în cealaltă submulțime semnul se schimbă. Pentru $n = 2$ vorbim de semiplan pozitiv și semiplan negativ.

Dacă $\mathbf{x_1}$ și $\mathbf{x_2}$ sunt două puncte de pe varietatea liniară S dată de $y(\mathbf{x}) = 0$, atunci:

$$y(\mathbf{x_1}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_1} + b = 0 \tag{5.3}$$

$$y(\mathbf{x_2}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_2} + b = 0 \tag{5.4}$$

Scăzând (5.4) din (5.3) obținem $\mathbf{w}^t \cdot (\mathbf{x_1} - \mathbf{x_2}) = 0$, sau, echivalent, vectorul $\mathbf{w}$ este perpendicular pe varietatea liniară $y(\mathbf{x}) = 0$, a se vedea figura 5.3.

Distanța dintre un punct de coordonate $\mathbf{x}$ și suprafața S este:

$$z = \frac{|y(\mathbf{x})|}{\|w\|} \tag{5.5}$$

unde $\mathbf{w} = (w_1, \ldots, w_n)^t$ iar $\|\mathbf{w}\|$ este norma Euclidiană[1] a vectorului $\mathbf{w}$, $\|\mathbf{w}\| = \sqrt{w_1^2 + \ldots w_n^2}$.

[1] Peste tot în cursul de azi considerăm norma Euclidiană.

Figura 5.3: Suprafața de decizie S, distanța de la punctul de coordonate $\mathbf{x}$ la S, poziția vectorului de ponderi $\mathbf{w}$ față de S

Putem renota termenul liber b din (5.1) cu w_0, putem considera că vectorul $\mathbf{x}$ mai are o componentă $x_0 = 1$; dacă notăm tot cu $\mathbf{w}$ vectorul de ponderi extins $\mathbf{w} = (w_0, w_1, \ldots, w_n)^t$ și $\mathbf{x} = (x_0, x_1, \ldots, x_n)^t$, cu $x_0 = 1$ și $w_0 = b$, atunci funcția de separare y din (5.1) se rescrie ca:

$$y(\mathbf{x}) = \mathbf{w}^t \cdot \mathbf{x} \tag{5.6}$$

5.3 Perceptronul liniar

Setul de instruire este

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | \mathbf{x}^{(i)} \in \mathbb{R}^{n+1}, y^{(i)} \in \{-1, 1\}, 1 \leq i \leq m \right\} \tag{5.7}$$

unde $y^{(i)}$ este eticheta vectorului de intrare $\mathbf{x}^{(i)}$. Putem partiționa vectorii de intrare $\mathbf{x}^{(i)}$ în submulțimile:

- $\mathscr{C}_1$ - clasa pozitivă, acei $\mathbf{x}^{(i)} \in \mathbb{R}^{n+1}$ pentru care $y^{(i)} = +1$,
- $\mathscr{C}_2$ - clasa negativă, acei $\mathbf{x}^{(i)} \in \mathbb{R}^{n+1}$ pentru care $y^{(i)} = -1$.

Setul $\mathscr{S}$ este astfel dat încât mulțimile $\mathscr{C}_1$ și $\mathscr{C}_2$ sunt liniar separabile – aceasta fiind condiția în care perceptronul poate învăța să construiască o suprafață de separare.

Graful de calcul pentru un perceptron este dat în figura 5.4.

Valoarea de stare a perceptronului este $z = \mathbf{w}^t \cdot \mathbf{x} = \sum_{i=0}^{n} w_i \cdot x_i$. Valoarea de ieșire (numită și activare) a perceptronului se calculează cu funcția de

Figura 5.4: Graful de calcul al unui perceptron liniar

activare "treaptă":

$$f(z) = \begin{cases} +1, & \text{dacă } z > 0 \\ -1, & \text{dacă } z < 0 \\ \text{nedefinit}, & \text{dacă } z = 0 \end{cases} \tag{5.8}$$

Valoarea produsă pentru ponderile $\mathbf{w}$ și intrarea $\mathbf{x}$ este:

$$\hat{y} = f(\mathbf{w}^t \cdot \mathbf{x}) \tag{5.9}$$

La inferență făcută cu ponderile $\mathbf{w}$ pentru un vector $\mathbf{x}$:

- dacă $\mathbf{w}^t \cdot \mathbf{x} > 0$, atunci $f(\mathbf{w}^t \cdot \mathbf{x}) = 1$ și spunem că perceptronul indică $\mathbf{x}$ ca fiind de clasă pozitivă
- dacă $\mathbf{w}^t \cdot \mathbf{x} < 0$, atunci $f(\mathbf{w}^t \cdot \mathbf{x}) = -1$ și spunem că perceptronul indică $\mathbf{x}$ ca fiind de clasă negativă

Valoarea funcției f este deliberat nedefinită în 0: dacă $\mathbf{w}^t \cdot \mathbf{x} = 0$ atunci punctul $\mathbf{x}$ se află pe suprafața de separare și pentru un atare caz nu se poate spune despre el că ar fi de clasă pozitivă sau negativă.

5.4 Algoritmul de instruire a perceptronului

Scopul algoritmului este ca, plecând de la setul de instruire $\mathscr{S}$ definit în (5.7), să obținem ponderile $\mathbf{w} = (w_0, w_1, \ldots, w_n)^t$ astfel încât:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_1 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_2 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{5.10}$$

adică să obținem o suprafață liniară ce separă clasele $\mathscr{C}_1$ și $\mathscr{C}_2$.

Com demonstra – vezi secțiunea 5.7 – că dacă mulțimile $\mathscr{C}_1$ și $\mathscr{C}_2$ sunt liniar separabile, atunci algoritmul de instruire poate să determine o funcție de separare liniară.

Instruirea pornește de la ponderi aleatoare pentru valorile $w_0, w_1, \ldots, w_n$; acestea pot fi luate chiar și 0. Notăm acest vector inițial cu $\mathbf{w}(1)$. Printr-un proces iterativ vom obține vectorii de ponderi $\mathbf{w}(2), \ldots, \mathbf{w}(k), \ldots$. Vectorii de ponderi se vor stabiliza la un moment dat: $\mathbf{w}(l) = \mathbf{w}(l+1) = \ldots$.

Învățarea (adaptarea) ponderilor $\mathbf{w}$ se face prin adăugarea succesivă a unor vectori de diferențe:

$$\mathbf{w}(1) \xrightarrow{+\Delta\mathbf{w}(1)} \mathbf{w}(2) \xrightarrow{+\Delta\mathbf{w}(2)} \mathbf{w}(3) \ldots \mathbf{w}(k) \xrightarrow{+\Delta\mathbf{w}(k)} \mathbf{w}(k+1) \ldots \tag{5.11}$$

Pentru un vector de ponderi curente $\mathbf{w}(k)$, intrarea curentă $\mathbf{x}^{(i)}$ și eticheta asociată $y^{(i)}$, estimarea produsă de perceptron este $\hat{y}^{(i)} = f\left(\mathbf{w}(k)^t \mathbf{x}^{(i)}\right)$. Dacă pentru un vector de ponderi $\mathbf{w}(k)$ avem $\hat{y}^{(i)} = y^{(i)}$ pentru tot setul de instruire, atunci perceptronul recunoaște corect clasele datelor din $\mathscr{S}$. Altfel, este necesar să se opereze modificări pentru vectorul de ponderi curent $\mathbf{w}(k)$, detaliile fiind date în cele ce urmează.

Să considerăm ponderile de la momentul k, $\mathbf{w}(k)$. Dacă pentru o pereche $\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathscr{S}$ avem că $\hat{y}^{(i)} = y^{(i)}$, atunci pentru acest caz vectorul de ponderi $\mathbf{w}(k)$ nu trebuie modificat: perceptronul clasifică corect intrarea $\mathbf{x}^{(i)}$. Dacă $\hat{y}^{(i)} \neq y^{(i)}$, atunci avem unul din următoarele două cazuri:

1. $\mathbf{x}^{(i)}$ e de clasă pozitivă, dar e clasificat momentan ca negativ: $y^{(i)} = +1$ și $\hat{y}^{(i)} = -1$

2. $\mathbf{x}^{(i)}$ e de clasă negativă, dar e clasificat momentan ca pozitiv: $y^{(i)} = -1$ și $\hat{y}^{(i)} = +1$

Pentru cazul 1 avem că $\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} < 0$ dar ar trebui ca produsul să fie pozitiv. Trebuie deci modificat vectorul $\mathbf{w}(k)$ astfel încât, pentru noul vector $\mathbf{w}(k+1)$ valoarea produsului scalar cu $\mathbf{x}^{(i)}$ să crească: $\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)} > \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}$, în speranța că asta va duce la $\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)} > 0$ și deci la $\hat{y}^{(i)} = f(\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)}) = +1 = y^{(i)}$. Modificarea propusă $\Delta\mathbf{w}(k)$ este:

$$\Delta\mathbf{w}(k) = \alpha \cdot \mathbf{x}^{(i)} \tag{5.12}$$

unde α este un coeficient strict pozitiv. Noul vector de ponderi va fi:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)} \tag{5.13}$$

Verificăm că vectorul $\mathbf{w}(k+1)$ duce la creșterea valorii produsului scalar dintre ponderi și vectorul de intrare.

Avem:

$$\begin{aligned}\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)} = \left(\mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)}\right)^t \cdot \mathbf{x}^{(i)} = \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} + \alpha \cdot \left(\mathbf{x}^{(i)}\right)^t \cdot \mathbf{x}^{(i)} = \\ = \mathbf{w}(k)^t \cdot \mathbf{x} + \left\|\mathbf{x}^{(i)}\right\|^2 \geq \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}\end{aligned} \tag{5.14}$$

cu inegalitate strictă pentru $\left\|\mathbf{x}^{(i)}\right\| \neq 0$ (și intrucât $x_0^{(i)} = 1$, avem că norma lui $\mathbf{x}^{(i)}$ e strict pozitivă, deci inegalitatea (5.14) e de fapt strictă). Prin urmare, obținem creșterea produsului scalar dintre ponderi și vectorul de intrare, așa cum ne-am propus.

Pentru cazul 2, facem modificarea ponderilor astfel:

$$\Delta\mathbf{w}(k) = -\alpha \cdot \mathbf{x}^{(i)} \tag{5.15}$$

deci

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) - \alpha \cdot \mathbf{x}^{(i)} \tag{5.16}$$

cu aceeași condiție pentru α. Se verifică, similar ca în (5.14) că $\mathbf{w}(k+1)^t \cdot \mathbf{x} < \mathbf{w}(k)^t \cdot \mathbf{x}$.

Sintetizând:

$$\Delta\mathbf{w}(k) = \begin{cases} \mathbf{0}_{n+1}, & \text{dacă } y^{(i)} = \hat{y}^{(i)} \\ \alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = +1 \text{ și } \hat{y}^{(i)} = -1 \\ -\alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = -1 \text{ și } \hat{y}^{(i)} = +1 \end{cases} \tag{5.17}$$

$$= \begin{cases} \mathbf{0}_{n+1}, & \text{dacă } y^{(i)} = \hat{y}^{(i)} \\ \alpha y^{(i)} \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} \neq \hat{y}^{(i)} \end{cases} \tag{5.18}$$

Putem scrie modificările din ecuația (5.18), astfel:

$$\Delta\mathbf{w}(k) = \frac{\alpha}{2}(y^{(i)} - \hat{y}^{(i)}) \cdot \mathbf{x}^{(i)} \tag{5.19}$$

E posibil ca o singură modificare a ponderilor să nu fie suficientă pentru ca tot setul de instruire $\mathscr{S}$ să ajungă să fie clasificat corect. Dacă în decursul unei epoci de instruire – epocă înseamnă parcurge în întregime a setului de instruire $\mathscr{S}$ – apare măcar o modificare, se va efectua o nouă epocă. Procesul se încheie când ponderile se stabilizează, deci setul de instruire $\mathscr{S}$ e învățat. Teorema 1 ne asigură că instruirea se termină după un număr finit de modificări de ponderi.

Algoritmul de instruire a perceptronului este:

1. Inițializează $\mathbf{w}(1)$ cu componente aleatoare sau cu vectorul nul; setează $k = 1$
2. Repetă{
   (a) $corectClasificat = adevarat$
   (b) pentru $i = 1, m$
      i. calculează $\hat{y}^{(i)} = f\left(\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}\right)$
      ii. dacă $\hat{y}^{(i)} \neq y^{(i)}$ atunci:
         A. $corectClasificat = fals$

B. $\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha y^{(i)}\mathbf{x}^{(i)}$
C. $k = k + 1$

}până când $corectClasificat = adevarat$

3. Vectorul $\mathbf{w}(k)$ este vectorul final de ponderi pentru perceptron, iar k este numărul total de setări ale vectorului de ponderi $\mathbf{w}$.

Deși algoritmul modifică instantaneu ponderile pentru fiecare caz în care ieșirea furnizată de perceptron nu coincide cu eticheta cunoscută, algoritmul nu este incremental în sensul dat în secțiunea **??**, având de regulă nevoie de mai multe epoci de instruire.

Faptul că algoritmul se oprește după un număr finit de pași este demonstrat în secțiunea 5.7.

După ce algoritmul de instruire se oprește, clasificarea unui vector ca fiind de clasă pozitivă sau negativă se face ca în secțiunea 5.3.

5.5 Exemplu numeric

Considerăm setul de instruire:

$$\mathscr{S} = \{((-4,2),-1),((-4,6),-1),((-2,4),-1),((3,-2),1),((5,-2),1),((4,4),1)\} \tag{5.20}$$

unde un element de forma $((x,y),c)$ se interpretează ca: punctul de coordonate (x,y) este de clasă c. Plecăm cu dreapta inițială $d: -5x - 10y + 1 = 0$, deci $\mathbf{w} = (1,-5,-10)^t$. Punctele și dreapta sunt reprezentate în figura 5.5. Observăm că dreapta d nu separă clasele. În algoritmul de instruire considerăm $\alpha = 2$.

Se itereează pe rând punctele, în ordinea dată în $\mathscr{S}$. La punctul de coordonate $(-4,2)$ valoarea $\hat{y}$ este 1. Se face modificarea dictată de algoritm și vectorul de ponderi $\mathbf{w}$ devine $\mathbf{w} = (-1,3,-14)^t$, cu reprezentarea din figura 5.6. Observăm că nici această dreaptă nu separă corect clasele, deci e de așteptat să se modifice vectorul de ponderi.

Punctele de coordonate $(-4,6)$, $(-2,4)$, $(3,-2)$, $(5,-2)$ sunt corect clasificate de dreapta curentă. Punctul de coordonate $(4,4)$ este incorect clasificat ca fiind negativ, deci se face modificare asupra ponderilor și se obține $\mathbf{w} = (1,11,-6)$, deci dreapta va fi $d: 11x - 6y + 1 = 0$, cu reprezentarea în figura 5.7. Algoritmul mai face o epocă de instruire; deoarece nu mai e nevoie să se facă modificări asupra ponderilor $\mathbf{w}$, algoritmul se oprește la finalul celei de a doua epoci.

Figura 5.5: Se pleacă cu dreapta $d : -5x - 10y + 1 = 0$.

Figura 5.6: La prima eroare de clasificare dreapta se schimbă în $d : 3x - 14y - 1 = 0$.

Figura 5.7: Dreapta este schimbată în algoritmul de instruire la $d : 11x - 6y + 1 = 0$.

5.6 Modificarea ponderilor ca algoritm de tip gradient descent

Algoritmii de instruire pentru regresia liniară și cea logistică au folosit gradienții unor funcții de eroare. Putem da și pentru perceptron o regulă de instruire cu gradienți, considerând funcția de eroare pentru tot setul de instruire:

$$J(\mathbf{w}) = \sum_{i:\hat{y}^{(i)} \neq y^{(i)}} \hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(\mathbf{i})} \tag{5.21}$$

unde mulțimea peste care se face însumarea este mulțimea acelor cazuri din $\mathscr{S}$ pentru care perceptronul are erori de clasificare.

În mod evident:

- atunci când perceptronul face clasificare corectă pentru tot setul de instruire, funcția J are valoarea 0, deoarece suma se calculează peste o mulțime vidă;
- pentru un caz $\hat{y}^{(i)} \neq y^{(i)}$ avem că $\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(\mathbf{i})} > 0$ conform justificărilor din cele două cazuri tratate la pagina 65.

Pentru un caz $\hat{y}^{(i)} \neq y^{(i)}$ oarecare, gradientul lui $\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(\mathbf{i})}$ în raport cu $\mathbf{w}$ este:

$$\nabla_{\mathbf{w}} \left[\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(i)} \right] = \hat{y}^{(i)} \nabla_{\mathbf{w}} \left[\mathbf{w}^t \cdot \mathbf{x}^{(i)} \right] = \hat{y}^{(i)} \mathbf{x}^{(i)} \tag{5.22}$$

pentru ultimul pas a se vedea lucrarea [6] secțiunea 2.4.1, ecuația (69); s-a mai folosit această proprietate și în capitolul de regresie liniară, ecuația (2.29) pagina 30.

Dacă aplicăm algoritmul gradient descent pentru intrarea curentă, înseamnă că modificăm[2] ponderile actuale cu $-\alpha \cdot \nabla_{\mathbf{w}} \left[\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(i)}\right] = \alpha(-\hat{y}^{(i)})\mathbf{x}^{(i)} = \alpha y^{(i)} \mathbf{x}^{(i)}$ și recunoaștem termenul de modificare a ponderilor din algoritmul de instruire a perceptronului.

Se poate remarca o deosebire între algoritmul perceptronului și cei de la regresia liniară sau logistică: la aceștia din urmă funcția de eroare este calculată pe tot setul de instruire, pe când în algoritmul perceptronului se calculează eroarea și se aplică gradientul ei pentru fiecare caz de clasificare greșită. Modificarea algoritmului perceptronului pentru a folosi funcția agregată (5.21) și gradienții aferenți este imediată. Am preferat algoritmul de la pagina 66, fiind clasic.

5.7 Convergența perceptronului

Vom demonstra că algoritmul de instruire a perceptronului se termină în număr finit de pași, dacă setul de instruire $\mathscr{S}$ este compus din două mulțimi $\mathscr{C}_1$, $\mathscr{C}_2$ liniar separabile.

Condiția de liniar separabilitate ne permite să spunem că există un vector $\mathbf{w}_* \in \mathbb{R}^{n+1}$ cu:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_1 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_2 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{5.23}$$

(a se revedea condițiile 5.10), sau, mai pe scurt,

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \ \text{ pentru orice } i, 1 \leq i \leq m \tag{5.24}$$

Putem presupune că vectorul $\mathbf{w}_*$ este de normă 1; în caz contrar se poate face normarea lui[3], iar inegalitatea din (5.24) rămâne adevărată.

Întrucât numărul de elemente din setul de instruire este finit, putem găsi $\gamma > 0$ astfel încât:

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > \gamma \ \text{ pentru orice } i, 1 \leq i \leq m \tag{5.25}$$

de exemplu

$$\gamma = \frac{1}{2} \cdot \min_{1 \leq i \leq m} \left\{ y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} \right\} > 0 \tag{5.26}$$

[2] Ca și până acum, $\alpha > 0$ este rata de învățare.

[3] Un vector nenul împărțit la norma lui produce un vector de normă 1. Se observă că norma lui w_* nu poate fi 0: dacă ar fi, atunci tot vectorul ar fi 0, iar inecuația 5.24 nu ar mai putea fi îndeplinită pentru niciun i.

Deoarece vectorul $\mathbf{w}_*$ nu este cunoscut (știm însă că există, pentru că $\mathscr{C}_1$ și $\mathscr{C}_2$ sunt liniar separabile), cantitatea γ e și ea necunoscută. Considerarea ei e totuși utilă pentru demonstrarea convergenței algoritmului de instruire a perceptronului.

Tot datorită faptului că mulțimea de instruire e finită, avem că există un $R > 0$ astfel încât $\|x^{(i)}\| \leq R$: toți vectorii de intrare din setul de instruire sunt cuprinși în interiorul unei hiper–sfere centrate în origine și de rază R suficient de mare. Putem, de exemplu, să luăm

$$R = \max_{1 \leq i \leq m} \left\{ \|\mathbf{x}^{(i)}\| \right\} \tag{5.27}$$

Două presupuneri care simplifică partea de demonstrație pentru teorema ce urmează sunt:

1. $\alpha = 1$; valoarea lui α nu are importanță în algoritm, atâta timp cât e mai mare ca zero;

2. inițial vectorul $\mathbf{w}(1)$ are toate elementele 0.

Teorema 1. *(Teorema de convergență a perceptronului) Algoritmul de instruire a perceptronului efectuează mai puțin de R^2/γ^2 modificări ale ponderilor, după care returnează un hiperplan de separare.*

Demonstraţie. Pentru $k \geq 1$, fie un $\mathbf{x}^{(i)}$ clasificat greșit de perceptron, adică $f(\mathbf{w}(k)^t \mathbf{x}^{(i)}) \neq y^{(i)}$, sau, echivalent:

$$f(\mathbf{w}(k)^t \mathbf{x}^{(i)}) \neq y^{(i)} \Leftrightarrow y^{(i)} \cdot \mathbf{w}(k)^t \mathbf{x}^{(i)} < 0 \tag{5.28}$$

Pentru această situație de clasificare greșită a unui vector din setul de instruire, algoritmul efectuează modificare a lui $\mathbf{w}(k)$ conform ecuației (5.18).

Avem:

$$\begin{aligned}
\mathbf{w}(k+1)^t \cdot \mathbf{w}_* &= (\mathbf{w}(k) + y^{(i)} \mathbf{x}^{(i)})^t \cdot \mathbf{w}_* && (5.29)\\
&= \mathbf{w}(k)^t \cdot \mathbf{w}_* + \underbrace{y^{(i)} (\mathbf{x}^{(i)})^t \cdot \mathbf{w}_*}_{>\gamma \text{ cf. } (5.26)} && (5.30)\\
&> \mathbf{w}(k)^t \cdot \mathbf{w}_* + \gamma && (5.31)
\end{aligned}$$

Prin inducție matematică și ținând cont că $\mathbf{w}(1) = \mathbf{0}$, se arată că:

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* > k\gamma \tag{5.32}$$

Folosim inegalitatea Cauchy—Schwartz ($|\mathbf{a}^t \cdot \mathbf{b}| \leq \|\mathbf{a}\| \cdot \|\mathbf{b}\|$), faptul că pentru orice număr real z avem că $z \leq |z|$ și obținem:

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* \leq |\mathbf{w}(k+1)^t \cdot \mathbf{w}_*| \leq \|\mathbf{w}(k+1)\| \cdot \underbrace{\|\mathbf{w}_*\|}_{=1} = \|\mathbf{w}(k+1)\| \tag{5.33}$$

și din (5.32) și (5.33) avem:

$$\|\mathbf{w}(k+1)\| > k\gamma \tag{5.34}$$

Pe de altă parte, avem că:

$$\begin{aligned}
\|\mathbf{w}(k+1)\|^2 &= \|\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)}\|^2 && (5.35)\\
&= (\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)})^t \cdot (\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)}) && (5.36)\\
&= \|\mathbf{w}(k)\|^2 + \|y^{(i)}\mathbf{x}^{(i)}\|^2 + 2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)} && (5.37)\\
&= \|\mathbf{w}(k)\|^2 + \underbrace{|y^{(i)}|^2}_{=1} \cdot \underbrace{\|\mathbf{x}^{(i)}\|^2}_{\leq R^2} + \underbrace{2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)}}_{<0 \text{ cf. } (5.28)} && (5.38)\\
&< \|\mathbf{w}(k)\|^2 + R^2 && (5.39)
\end{aligned}$$

Ţinând cont că $\mathbf{w}(1) = \mathbf{0}_{n+1}$, prin inducție matematică se arată că:

$$\|\mathbf{w}(k+1)\|^2 < kR^2 \tag{5.40}$$

Din inecuațiile (5.34) și (5.40) rezultă că:

$$k^2\gamma^2 < \|\mathbf{w}(k+1)\|^2 \leq kR^2 \tag{5.41}$$

și deci $k < R^2/\gamma^2$. □

Am obținut deci că indicele k pentru care se fac modificare de ponderi nu poate fi oricât de mare, deci algoritmul se termină în timp finit. Finalizarea lui înseamnă totodată obținerea unui set de ponderi pentru forma de separare liniară care să clasifice corect cazurile din setul de instruire.

Comentariu: Deoarece valoarea lui γ nu e cunoscută, rezultatul de mai sus nu ne spune care e numărul maxim de pași necesari. Totuși, s-a găsit o dovadă că algoritmul nu rulează la infinit. Din criteriul de terminare a algoritmului deducem că la final avem un perceptron care separă $\mathscr{C}_1$ de $\mathscr{C}_2$.

5.8 Algoritmul lui Gallant

Algoritmul lui Gallant – sau algoritmul "buzunarului" – tratează cazul în care setul de instruire nu este liniar separabil. Ideea algoritmului este de a menține vectorul $\mathbf{w}$ de ponderi care face cele mai puține erori de clasificare pentru date succesive. La finalul unei epoci se contorizează câte cazuri din setul de instruire sunt corect clasificate de vectorul curent de ponderi. Dacă numărul de clasificări corecte este mai mare decât pentru vectorul menținut până până acum într-un "buzunar" (la început: vectorul $\mathbf{w}(1)$ cu număr de clasificări corecte produse de el), atunci se actualizează conținutul "buzunarului": vectorul curent de ponderi și numărul de clasificări corecte. Procesul se repetă de un număr de ori specificat. La final se returnează vectorul de ponderi din "buzunar".

5.9 Comentarii

Spre deosebire de regresia logistică, perceptronul liniar nu produce o valoare care să exprime în ce măsură modelul consideră că un vector de intrare aparține clasei pozitive, $P(\mathscr{C}_1|\mathbf{x})$. Totuși, vine cu demonstrație matematică pentru convergență, dacă o varietate liniară desparte clasa pozitivă de cea negativă.

La momentul apariției, perceptronul liniar a fost privit ca un motiv clar pentru care problemele cele mai complexe sunt rezolvabile prin perceptroni. Cartea *Perceptrons*[4] a lui Minsky si Papert, din 1969, arată însă că perceptronul nu poate rezolva probleme care sunt neseparabile liniar, de exemplu problema XOR. Conjectura lor că utilizarea de mai mulți perceptroni nu poate să ducă la rezolvarea de probleme neseparabile liniar a devenit extrem de populară, motiv pentru care cercetările în domeniul rețelelor neurale artificiale au fost descurajate. Revenirea s–a produs în 1986, când Rumelhart, Hinton și Williams[5] au propus o procedură de învățare pentru rețele cu mai multe straturi de neuroni neliniari care permitea abordarea claselor neseparabile liniar.

Setul de instruire pentru problema XOR este următorul:

$$\Big\{((0,0)^t,0),((1,1)^t,0),((1,0)^t,1),((0,1)^t,1)\Big\}$$

unde fiecare din cele 4 tuple contine o pereche de valori de intrare din $\{0,1\}^2$ împreună cu eticheta de clasă asociată, 0 sau 1. Reprezentarea este dată în figura 5.8.

Figura 5.8: Problema XOR. Clasele sunt marcate cu forme diferite. Se poate demonstra că nu există 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”.

Demonstrăm algebric că nu există un vector de 3 ponderi $(w_0, w_1, w_2)^t \in \mathbb{R}^3$ pentru care:

$$sgn(w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 0) = sgn(w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1) = 1 \qquad (5.42)$$

[4]Marvin Minsky și Seymour Papert, *Perceptrons: an introduction to computational geometry*, MIT Press, 1969.

[5]David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, *Learning representations by back-propagating errors*, Nature, volume 323, issue 6088, pp. 533-536, 1986.

iar

$$sgn(w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 1) = sgn(w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 0) = -1 \quad (5.43)$$

Incompatibilitatea sistemului de ecuații (5.42, 5.43) se arată ușor: să presupunem că ar exista w_0, w_1, w_2 care să satisfacă (5.42, 5.43), pe care le rescriem echivalent:

$$w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 0 > 0 \quad (5.44a)$$
$$w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1 > 0 \quad (5.44b)$$
$$w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 1 < 0 \quad (5.44c)$$
$$w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 0 < 0 \quad (5.44d)$$

Grupând primele și ultimele două inecuații și adunând, obținem:

$$2 \cdot w_0 + w_1 + w_2 > 0 \quad (5.45a)$$
$$2 \cdot w_0 + w_1 + w_2 < 0 \quad (5.45b)$$

Pentru problema n–dimensională, clasa de ieșire pentru vectorul binar $\mathbf{x} \in \{0, 1\}^n$ este 1 dacă numărul de componente 1 este impar, altfel 0. Se poate arăta că și pentru cazul n dimensional problema nu e rezolvabilă printr–un separator liniar.

Capitolul 6

Perceptronii multistrat

Rețelele neurale multistrat — sau perceptronii multistrat, multilayer perceptrons (MLP) — 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 și fac parte din familia rețelelor cu propagare înainte.

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 – și numai dacă – ele sunt liniar separabile. Există însă exemple de clase de puncte care nu sunt liniar separabile — și deci nerezolvabile de către perceptronul liniar — dar pentru care separare se poate, totuși, face. 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, clasificarea oricâtor clase. Cursul de față descrie un model bazat pe straturi de neuroni cu funcții de activare neliniare, împreună cu algoritmul de învățare asociat, în care se pot rezolva toate aceste tipuri de probleme.

În capitolul anterior s–a dat un exemplu clasic de două mulțimi care nu sunt liniar separabile și deci, un perceptron liniar nu le poate discrimina. Un alt exemplu este dat în figura 6.1.

Figura 6.1: Două clase de puncte ce nu sunt liniar separabile

Intuim că un singur neuron e prea puțin pentru probleme complexe de separare. Totodată, concatenarea mai multor neuroni cu funcție de activare liniară este echivalentă cu produsul dintre vectorul de intrare si o matrice rezultată din înmulțirea unei succesiuni de matrice de ponderi. Datorită faptului că înmulțirea de matrice produce tot o matrice, operația este echivalentă cu înmulțirea unei matrice cu vectorul de intrare. Obținem de aici că succesiunea de neuroni cu funcție de activare liniară este echivalentă cu un singur strat de neuroni cu funcție de activare liniară.

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

6.2 Notații

Tabelul 6.1 conține notațiile 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:

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \ldots, \left(\mathbf{x}^{(p)}, \mathbf{y}^{(p)}\right) \right\} \tag{6.1}$$

unde $\mathbf{x}^{(i)} \in \mathbb{R}^n$ iar $\mathbf{y}^{(i)}$ este eticheta asociată intrării i (eng: ground truth), după caz:

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

6.4 Rețeaua neurală MLP

6.4.1 Arhitectură

Există mai multe modalități de dispunere a neuronilor artificiali într–o rețea neurală; 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 neliniară. O rețea multistrat se compune din minim trei straturi:

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

Noțiune sau notație	Explicație
p	numărul de perechi din setul de instruire
$\mathbf{x}^{(i)}$	vector de intrare din setul de instruire, $\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_n^{(i)}\right)^t$, $1 \leq i \leq p$
$\mathbf{y}^{(i)}$	ieșirea asociată intrării $\mathbf{x}^{(i)}$, din setul de instruire, $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \ldots, y_m^{(i)}\right)^t$, $1 \leq i \leq p$
L	numărul de straturi din rețeaua neurală, inclusiv straturile de intrare și de ieșire
nod	neuron – dacă apare în stratul $1, 2, \ldots, L-1$ – sau nod de intrare – dacă apare în primul strat (de indice 0)
n_l	numărul de noduri din stratul l, $0 \leq l \leq L-1$; $n_0 = n$, $n_{L-1} = m$
$z_i^{[l]}$	starea neuronului i din stratul l, $1 \leq l \leq L-1$, $0 \leq i \leq n_l$
$\mathbf{z}^{[l]}$	vectorul conținând stările neuronilor din stratul l, $\mathbf{z}^{[l]} = \left(z_1^{[l]}, \ldots, z_{n_l}^{[l]}\right)^t$, $1 \leq l \leq L-1$
$a_i^{[l]}$	activarea, valoarea de ieșire a celui de al i–lea nod din stratul l, $0 \leq l \leq L-1$, $1 \leq i \leq n_l$
$\mathbf{a}^{[l]}$	vectorul cu activările nodurilor din stratul l, $0 \leq l \leq L-1$, $\mathbf{a}^{[l]} = \left(a_1^{[l]}, \ldots, a_{n_l}^{[l]}\right)^t$
$w_{ij}^{[l]}$	ponderea legăturii între neuronul i de pe stratul l și nodul j de pe stratul $l-1$, $1 \leq l \leq L-1$, $1 \leq i \leq n_l$, $1 \leq j \leq n_{l-1}$
$\mathbf{W}^{[l]}$	matricea de ponderi dintre straturile $l-1$ și l, $1 \leq l \leq L-1$, $\mathbf{W}_{ij}^{[l]} = w_{ij}^{[l]}$, $1 \leq i \leq n_l$, $1 \leq j \leq n_{l-1}$
$\mathbf{W}_i^{[l]}$	linia i a matricei $\mathbf{W}^{[l]}$, $1 \leq l \leq L-1$, $1 \leq i \leq n_l$
$b_i^{[l]}$	ponderea de bias pentru neuronul i din stratul l, $1 \leq l \leq L-1$, $1 \leq i \leq n_l$
$\mathbf{b}^{[l]}$	vectorul ponderilor de bias către stratul l, $\mathbf{b}^{[l]} = \left(b_1^{[l]}, \ldots, b_{n_l}^{[l]}\right)^t$, $1 \leq l \leq L-1$
$f^{[l]}$	funcție de activare a neuronilor din stratul l, $1 \leq l \leq L-1$
$\mathbf{W}$	secvența de matrice de ponderi $(\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^{L-1})$
$\mathbf{b}$	secvența de vectori de ponderi de bias $(\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^{L-1})$
$J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$	eroarea pentru perechea de vectori $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$
$J(\mathbf{W}, \mathbf{b})$	eroare empirică medie pentru set de vectori cu etichete cunoscute
$\hat{\mathbf{y}}^{(i)}$	vector coloană de ieșire corespunzător intrării $\mathbf{x}^{(i)}$, calculat de rețea
$\boldsymbol{\delta}^l$	vectorul cu semnalul de eroare pentru stratul l, $1 \leq l \leq L-1$
$\odot$	produs Hadamard

Tabelul 6.1: Notații folosite pentru MLP

- strat de intrare ce preia valorile de intrare; nu are rol computațional, nu este format din neuroni[2];

- măcar un strat ascuns, compus din neuroni;

- strat de ieșire, de asemenea compus din neuroni, produce valori estimate care sunt apoi comparate cu ieșirile dorite.

Începem numerotarea straturilor de la 0. Neuronii din straturile ascunse produc trăsături noi pe baza vectorilor de intrare și a funcțiilor de activare neliniare, 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 6.3.

Instruirea e mai eficientă dacă pe lângă valorile de intrare și 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 învățare[4].

O reprezentare de rețea neurală cu trei straturi și o ieșire este dată în figura 6.2; o rețea cu 4 straturi și două ieșiri este reprezentată în figura 6.3. Nu există o relație anume între numărul de straturi ascunse, numărul de neuroni de pe aceste straturi și numărul de noduri de intrare și ieșire.

Vom considera că avem $L \geq 3$ straturi și în fiecare strat ascuns l ($1 \leq l \leq L-1$) un număr de n_l noduri. Stratul de intrare are $n_0 = n$ noduri, numărul de neuroni din stratul de ieșire este $n_{L-1} = m$ dat de: numărul de clase pentru care se face recunoașterea (la problemă de clasificare sau estimare de probabilitate condiționată) respectiv numărul de ieșiri care se doresc a fi aproximate (la regresie).

Pentru oricare dintre figurile 6.2 și 6.3:

- valorile x_1, x_2, x_3 sunt componentele vectorului de intrare $\mathbf{x} = (x_1, x_2, x_3)^t$; se mai consideră încă o intrare cu valoarea constantă +1, aferentă coeficientului de *bias*;

- valoarea $a_i^{[l]}$ este ieșirea (activare) nodului i din stratul l, $0 \leq l \leq L-1$, $1 \leq i \leq n_l$. Pentru stratul de intrare activările sunt chiar intrările, *i.e.* $a_i^{[0]} = x_i$;

[2]Motiv pentru care unii autori nu îl consideră un strat propriu–zis; frecvent se folosește exprimarea că o rețea are "k" straturi ascunse, cele de intrare și ieșire existând oricum. În cele ce urmează considerăm intrarea ca formând un strat.

[3]Unii autori consideră valoarea constantă -1 în loc de +1; nu e esențial, deoarece ponderile sunt sunt determinate prin instruire și pot avea orice semn.

[4]O discuție asupra necesității considerării bias–ului este la https://web.archive.org/web/20210207195343/ftp://ftp.sas.com/pub/neural/FAQ2.html

- valoarea $h_{\mathbf{W},\mathbf{b}}(\mathbf{x})$ este ieșirea calculată de către rețea pentru vectorul curent de intrare $\mathbf{x}$; $h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \in \mathbb{R}$ pentru figura 6.2; $h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \in \mathbb{R}^2$ pentru figura 6.3. Rețeaua din figura 6.2 poate fi folosită pentru estimarea unei valori continue (problemă de regresie) sau pentru estimare de probabilitate condiționată pentru două clase. Rețeaua din figura 6.3 se poate folosi pentru estimarea a două valori de ieșire, numere reale – problemă de regresie.

Figura 6.2: Rețea MLP cu 3 straturi. Poate fi folosită pentru estimarea unei valori continue (problemă de regresie) sau pentru discriminare/estimare de probabilitate condiționată pentru două clase.

Figura 6.3: Rețea MLP cu 4 straturi ce se poate folosi pentru estimarea a două valori de ieșire.

Perechea $(\mathbf{W}, \mathbf{b})$ este formată din matricele ponderilor și a coeficienților de bias din în rețea. Folosim 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

(a) Ponderile dintre nodul j din stratul $l-1$ și nodurile din stratul l.

(b) Valoarea de bias și ponderile dintre stratul $l-1$ și nodul i din stratul l.

Figura 6.4: Legăturile între noduri din straturile $l-1$ și l.

al doilea (de indice 1) și nodul j din stratul de intrare; se remarcă ordinea indicilor inferiori, utilă mai departe pentru operațiile folosite;

- în general, notăm cu $w_{ij}^{[l]}$ ponderea legăturii dintre al i–lea neuron din stratul de indice l și al j–lea nod (neuron sau nod de intrare) din stratul $l-1$ ($1 \leq l \leq L-1$); a se vedea figurile 6.4a, 6.4b. Ele formează matricea de ponderi:

$$\mathbf{W}^{[l]} = \begin{pmatrix} w_{11}^{[l]} & \dots & w_{1n_{l-1}}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n_l 1}^{[l]} & \dots & w_{n_l n_{l-1}}^{[l]} \end{pmatrix} \tag{6.2}$$

  cu $n_l \times n_{l-1}$ ponderi;

- valoarea ponderii de bias dintre intrarea constantă $+1$ din stratul de intrare și neuronul i din primul strat ascuns este $b_i^{[1]}$, $1 \leq i \leq n_2$;

- în general, ponderea de bias provenind din stratul $l-1$ ($1 \leq l \leq L-1$) și pentru neuronul i din stratul l este notată cu $b_i^{[l]}$, $1 \leq i \leq n_l$, — figura 6.4b. Coeficienții $b_i^{[l]}$ formează vectorul coloană $\mathbf{b}^{[l]}$ cu n_l coeficienți.

Datorită faptului că avem legături între oricare doi neuroni aflati pe straturi succesive, rețelele MLP se mai numesc și *dens conectate* (eng: fully connected networks). Există alte tipuri de rețele (de exemplu cele de tip convoluțional, capitolul 8) care folosesc legături mai rare.

6.4.2 Funcții de activare

Fiecare neuron agregă valorile din nodurile din stratul anterior – incluzând și termenul constant +1 înmulțit cu coeficientul de bias. Neuronul de indice i din stratul $l \geq 1$ are starea calculată ca:

$$z_i^{[l]} = w_{i1}^{[l]} \cdot a_1^{[l-1]} + w_{i2}^{[l]} \cdot a_2^{[l-1]} + \cdots + w_{i,n_{l-1}}^{[l]} \cdot a_{n_{l-1}}^{[l-1]} + b_i^{[l]} \tag{6.3}$$

$$= \mathbf{W}_i^{[l]} \cdot \mathbf{a}^{[l-1]} + b_i^{[l]},\ 1 \leq i \leq n_l \tag{6.4}$$

unde: $\mathbf{W}_i^{[l]}$ este linia i a matricei $\mathbf{W}^{[l]}$, $a_i^{[l-1]}$ este, după caz: valoarea de ieșire a neuronului i din stratul $l-1$ (dacă $l \geq 1$) sau valoarea x_i din vectorul de intrare curent (dacă $l = 0$); evident, vectorul $\mathbf{a}^{[l]}$ este $(a_1^{[l]}, \ldots, a_{n_l}^{[l]})^t$. Notând cu $\mathbf{z}^{[l]}$ vectorul coloană $\left(z_1^{[l]}, z_2^{[l]}, \ldots, z_{n_l}^{[l]}\right)^t$, putem scrie matricial:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]},\ 1 \leq l \leq L-1 \tag{6.5}$$

Pe baza stării $z_i^{[l]}$ a neuronului i din stratul l se calculează ieșirea (activarea) – sa folosind funcția de activare $f^{[l]}$:

$$a_i^{[l]} = f^{[l]}\left(z_i^{[l]}\right) \tag{6.6}$$

pentru $1 \leq l \leq L-1, 1 \leq i \leq n_l$. Dacă folosim notația $f^{[l]}\left((z_1, z_2, \ldots, z_k)^t\right) \stackrel{\text{def}}{=} \left(f^{[l]}(z_1), f^{[l]}(z_2), \ldots, f^{[l]}(z_k)\right)^t$ – adică se aplică funcția $f^{[l]}$ pe fiecare valoare din vectorul argument – atunci putem scrie mai compact ecuația (6.6) sub forma:

$$\mathbf{a}^{[l]} = f^{[l]}\left(\mathbf{z}^{[l]}\right) \tag{6.7}$$

Funcția de activare $f^{[l]}(\cdot)$ este necesară în pasul de propagare înainte; pentru pasul de propagare înapoi a erorii este folosită derivata ei. Alegeri populare pentru funcția de activare sunt:

1. **funcția logistică sigmoidă:**

$$f = \sigma : \mathbb{R} \to (0,1), f(z) = \sigma(z) = \frac{1}{1+\exp(-z)} \tag{6.8}$$

   Derivata acestei funcții este:

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z)) \tag{6.9}$$

2. **funcția tangentă hiperbolică**:

$$\begin{aligned} f &= \tanh : \mathbb{R} \to (-1,1), \\ f(z) &= \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} && (6.10) \\ &= \frac{2}{1+\exp(-2z)} - 1 && (6.11) \end{aligned}$$

a cărei derivată este:

$$\tanh'(z) = 1 - \tanh^2(z) \tag{6.12}$$

Se arată ușor că între cele două funcții de activare tanh și σ există relația:

$$\tanh(z) = 2 \cdot \sigma(2z) - 1 \tag{6.13}$$

În practică funcția tanh dă rezultate mai bune decât sigmoida logistică. O explicație teoretică se găsește în [7]; rezultate empirice sunt în [8].

Atât sigmoida logistică, cât și tangenta hiperbolică au problema gradienților mici (eng. vanishing gradients): pentru acei z pentru care $\sigma(z) \approx 0$, $\sigma(z) \approx 1$, $\tanh(z) \approx -1$, $\tanh(z) \approx 1$ avem că derivatele lor sunt apropiate de zero. Întrucât derivatele sunt mai departe folosite prin înmulțire, modificările ponderilor vor fi nesemnificative și învățarea va stagna.

3. **funcția liniară:**

$$f(z) = a \cdot z + b \tag{6.14}$$

cu derivata $f'(z) = a$; frecvent se iau $a = 1$, $b = 0$ (funcție identitate, $f(z) = z$). 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.

4. **funcția softmax:**

$$softmax(\mathbf{z}; i) = \frac{\exp(z_i)}{\sum\limits_{j=1}^{m} \exp(z_j)} \tag{6.15}$$

unde i este indicele neuronului, iar m este numărul total de valori din vectorul $\mathbf{z}$. Funcția softmax a mai fost folosită la regresia logistică pentru cazul a mai mult de două clase; $\mathbf{z}$ este vector cu valori de stare a neuronilor, $\mathbf{z} = (z_1, \ldots, z_m)^t$.

Funcția softmax este utilă pentru a transforma un vector de valori oarecare în distribuție de probabilitate: se arată ușor că $softmax(\mathbf{z}; c) \in (0, 1)$ $\forall c$ și $\sum_{c=1}^{m} softmax(\mathbf{z}; c) = 1$. De regulă, $softmax$ se folosește pentru stratul de ieșire și valorile furnizate se interpretează convenabil drept probabilitatea ca intrarea curentă să fie de clasă c, $1 \leq c \leq m$; clasificarea se face găsind acel indice $1 \leq c \leq m$ pentru care $softmax(\mathbf{z}; c)$ este maxim. Se utilizează în stratul de ieșire a rețelei neurale de clasificare sau estimare de probabilitate.

Derivatele parțiale ale funcției softmax sunt:

$$\frac{\partial softmax(\mathbf{z}; i)}{\partial z_j} = \begin{cases} softmax(\mathbf{z}; i) \cdot (1 - softmax(\mathbf{z}; i)) & \text{dacă } i = j \\ -softmax(\mathbf{z}; i) \cdot softmax(\mathbf{z}; j) & \text{dacă } i \neq j \end{cases} \tag{6.16}$$

Putem folosi funcția delta al lui Kronecker: $\delta_{ij} = 1$ dacă $i = j$ și 0 altfel și rescriem (6.16) ca:

$$\frac{\partial softmax(\mathbf{z}; i)}{\partial z_j} = softmax(\mathbf{z}; i) \cdot (\delta_{ij} - softmax(\mathbf{z}; j)) \tag{6.17}$$

5. **funcția Rectified Linear Unit (ReLU):**

$$ReLU(z) = \max(0, z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.18}$$

Derivatele pe subintervale sunt ușor de calculat și cu evaluare rapidă. În plus, spre deosebire de sigmoida logistică și de tangenta hiperbolică, ele nu saturează.

Reprezentarea grafică e dată în figura 6.5.

Figura 6.5: Graficul funcției de activare ReLU

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. Derivata funcției ReLU în origine se ia în mod convenabil[5] 1. Avem deci:

$$ReLU'(z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ 1 & \text{dacă } z > 0 \end{cases} \tag{6.19}$$

Teoretic, există posibilitatea ca o mare parte din neuronii care folosesc ReLU să producă activare 0; asta se poate întâmpla, de exemplu, dacă valorile de bias sunt negative și foarte mici. E foarte probabil ca un

[5]Convine mai mult să considerăm derivata în 0 ca fiind 1: gradient egal cu 0 ar însemna că nu se face modificarea ponderilor.

astfel de neuron să devină insensibil la orice intrare și să producă mereu 0; spunem despre el că e un ReLU mort (eng. dying ReLU). Dacă toate activările sunt 0, rețeaua va produce o valoare constantă indiferent de intrare. Chiar și la învățare avem problemă, căci gradientul pentru un ReLU mort este 0, deci ponderile neuronului lui nu se modifică.

6. **funcția Parametric ReLU (PReLU)**, reprezentând o ușoară generalizare a funcției ReLU:

$$PReLU(z) = \begin{cases} \alpha z & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.20}$$

cu $\alpha > 0$. Graficul funcției pentru $\alpha = 0.1$ este dat în figura 6.6. Derivatele pe subintervale sunt:

$$PReLU'(z) = \begin{cases} \alpha & \text{dacă } z \leq 0 \\ 1 & \text{dacă } z > 0 \end{cases} \tag{6.21}$$

Pentru $\alpha = 0.01$ se obține un caz particular vehiculat în literatură, Leaky ReLU.

PRelu rezolvă problema "dying ReLU", deoarece permite modificarea ponderilor, chiar dacă starea neuronului este negativă.

Figura 6.6: Graficul funcției de activare PReLU pentru $\alpha = 0.1$

7. **funcția Exponential Linear Units (ELU)** este definită ca:

$$f(z) = \max(\alpha(e^z - 1), z) = \begin{cases} \alpha(e^z - 1) & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.22}$$

cu $0 < \alpha \ll 1$. Graficul este dat în figura 6.7. Derivata lui ELU este:

$$f'(z) = \begin{cases} \alpha \cdot e^z & \text{dacă } z \leq 0 \\ 1 & \text{dacă } z > 0 \end{cases} \tag{6.23}$$

Figura 6.7: Graficul funcției de activare ELU pentru $\alpha = 0.1$

8. **funcția Swish**:

$$Swish(z) = z \cdot \sigma(\beta z) = \frac{z}{1 + \exp(-\beta z)} \tag{6.24}$$

unde σ este logistica sigmoidă, β este parametru constant sau antrenabil. Se arată[6] că pentru rețele cu multe straturi Swish funcționează mai bine decât ReLU. Reprezentarea ei este dată în figura 6.8. Derivata ei este:

$$Swish'(z) = \sigma(\beta z)(1 + \beta z - \beta z \sigma(\beta z))) \tag{6.25}$$

Lista funcțiilor de activare de mai sus nu e deschisă. Este admis ca funcția de activare să difere de la strat la strat sau de la neuron la neuron. În practică se preferă folosirea aceleiași funcții de activare în toată rețeaua, exceptând eventual ultimul strat.

6.5 Pasul de propagare înainte

Odată ce arhitectura rețelei e fixată – numărul de straturi ascunse și numărul de neuroni în fiecare strat, precum și funcțiile de activare – se poate trece la instruirea și apoi utilizarea ei. Pasul de propagare înainte preia un

[6] Prajit Ramachandran, Barret Zoph, Quoc V. Le, "Searching for Activation Functions", https://arxiv.org/abs/1710.05941

Figura 6.8: Graficul funcției de activare Swish pentru $\beta \in \{0.5, 2\}$

vector de intrare $\mathbf{x} = (x_1, \ldots, x_n)^t$ și produce modificări în starea neuronilor rețelei pornind de la intrare, acționând succesiv asupra straturilor $1, \ldots, L-1$. Aceasta dă și numele familiei din care face parte rețeaua: "cu propagare înainte" – eng. "feedforward". Ieșirile din ultimul strat sunt folosite pentru predicție – regresie, estimare de probabilitate condiționată sau clasificare.

După cum s–a mai afirmat, stratul de intrare nu are rol computațional; valoare sa de ieșire este chiar vectorul de intrare – considerat ca vector coloană – $\mathbf{x}$ furnizat rețelei:

$$\mathbf{a}^{[0]} = \mathbf{x} \tag{6.26}$$

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

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \tag{6.27}$$

$$\mathbf{a}^{[l]} = f^{[l]}(\mathbf{z}^{[l]}) \tag{6.28}$$

pentru $l = 1, \ldots L-1$, cu $f^{[l]}(\cdot)$ funcție de activare. Vom nota cu $\hat{\mathbf{y}}$ vectorul de m valori de ieșire produs de către rețea:

$$\hat{\mathbf{y}} = \mathbf{a}^{[L-1]} \tag{6.29}$$

Pentru cazul în care se lucrează pe un set de date format din perechi vectori intrare–ieșire, vectorii coloană de date de intrare se pot concatena pe orizontală într–o matrice $\mathbf{X}$. De exemplu, pentru mulțimea $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \ldots, \left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)\right\}$, matricea $\mathbf{X}$ se formează ca:

$$\mathbf{X} = \begin{pmatrix} | & \ldots & | \\ \mathbf{x}^{(1)} & \ldots & \mathbf{x}^{(k)} \\ | & \ldots & | \end{pmatrix} \tag{6.30}$$

În cele ce urmează, se poate să avem $\mathbf{X}$ ca fiind matricea formată cu toate datele $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}$ din setul de instruire, sau doar cu un eșantion al acestora de k vectori (eng. mini-batch). În particular, acest minibatch poate fi format dintr-un singur vector de date, $\mathbf{x}^{(i)}$.

Propagarea înainte presupune că la intrare este furnizat un set de date $\mathbf{X}$ de k coloane. Intrarea dată este propagată înainte prin rețea și transformată succesiv:

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]} \cdot \mathbf{X} + \mathbf{b}^{[1]} = \mathbf{W}^{[1]} \cdot \mathbf{A}^{[0]} + \mathbf{b}^{[1]} \tag{6.31}$$

$$\mathbf{A}^{[1]} = f^{[1]}\left(\mathbf{Z}^{[1]}\right) \tag{6.32}$$

$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]} \cdot \mathbf{A}^{[1]} + \mathbf{b}^{[2]} \tag{6.33}$$

$$\mathbf{A}^{[2]} = f^{[2]}\left(\mathbf{Z}^{[2]}\right) \tag{6.34}$$

$$\ldots$$

$$\mathbf{Z}^{[L-1]} = \mathbf{W}^{[L-1]} \cdot \mathbf{A}^{[L-2]} + \mathbf{b}^{[L-1]} \tag{6.35}$$

$$\mathbf{A}^{[L-1]} = f^{[L-1]}\left(\mathbf{Z}^{[L-1]}\right) \tag{6.36}$$

$\mathbf{Z}^{[l]}$ și $\mathbf{A}^{[l]}$ sunt matrice cu n_l linii și k coloane. Pentru adunările din (6.31, 6.33, 6.35) se consideră că se aplică mecanismul de "broadcasting": vectorii coloană $\mathbf{b}^{[l]}$ produc prin copiere matrice cu k coloane.

Matricea $\mathbf{A}^{[L-1]}$ conține k coloane cu predicții, corespunzătoare celor k vectori de intrare din $\mathbf{X}$.

În practică, se recomandă ca operațiile date mai sus să fie implementate folosind biblioteci optimizate de algebră liniară, ce permit înmulțirea eficientă de matrice și calcul pe CPU sau GPU — Octave, Matlab, NumPy, PyTorch, Tensorflow, CuPy etc.

6.6 Funcții de cost

O pereche $(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m)$ va produce o valoare de eroare astfel: se furnizează vectorul $\mathbf{x}$ ca intrare în rețea și se calculează un vector de ieșire $\hat{\mathbf{y}}$, reprezentând estimarea produsă de rețea pentru intrarea furnizată; se folosește o funcție de cost, sau de eroare, $J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$ care se dorește a fi cu atât mai mică cu cât vectorul $\hat{\mathbf{y}}$ e mai apropiat de $\mathbf{y}$, și cu atât mai mare cu cât cei doi vectori sunt mai depărtați. În plus, se mai consideră un termen 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 în straturile ascunse și la ieșire.

Să considerăm că avem un set de date pentru care dorim să calculăm funcția de eroare (de cost). Vectorii de intrare sunt organizați în batch ca în ecuația (6.30), unde pentru fiecare vector coloană $\mathbf{x}^{(i)}$ avem o etichetă $\mathbf{y}^{(i)}$ asociată.

Forma generală a funcției de eroare calculată pentru setul de date etichetate $\mathbf{X}$ cu k date etichetate este:

$$
\begin{aligned}
J(\mathbf{W},\mathbf{b}) &= \underbrace{\left[\frac{1}{k}\sum_{i=1}^{k} \overbrace{J\left(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{y}^{(i)}\right)}^{\text{eroare empirică pentru}\left(\mathbf{x}^{(i)},\mathbf{y}^{(i)}\right)}\right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} \\
&+ \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_l}\sum_{j=1}^{n_{l-1}}\left(w_{ij}^{[l]}\right)^2}_{\text{Termen de regularizare } J_{reg}}
\end{aligned}
\tag{6.37}
$$

unde $\lambda > 0$ este coeficientul de regularizare. Factorul de regularizare este aici regularizarea L_2, o sumă de pătrate de norme Frobenius ale matricelor $\mathbf{W}^{[1]},\ldots,\mathbf{W}^{[L-1]}$:

$$
\left\|\mathbf{W}^{[l]}\right\|_F^2 \stackrel{\text{def}}{=} \sum_{i=1}^{n_l}\sum_{j=1}^{n_{l-1}}\left(w_{ij}^{[l]}\right)^2 \tag{6.38}
$$

Regularizarea impune o presiune asupra ponderilor – ele sunt direcționate spre 0. O subliniere importantă este că ponderile de bias nu sunt regularizate; de aceea vectorii $b^{[1]},\ldots,b^{[L-1]}$ nu apar în termenul de regularizare; la fel s-a întâmplat și la modelele studiate anterior.

Cu norma Frobenius, termenul de regularizare J_{reg} din (6.37) se scrie succint:

$$
J_{reg}(\mathbf{W}) = \frac{\lambda}{2}\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2 \tag{6.39}
$$

6.6.1 Funcția de cost pentru problemă de regresie

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

$$
J(\mathbf{W},\mathbf{b};\mathbf{x},\mathbf{y}) = \frac{1}{2}\cdot\|\mathbf{y}-\hat{\mathbf{y}}\|_2^2 \tag{6.40}
$$

unde $\hat{\mathbf{y}} \in \mathbb{R}^m$ este vectorul de ieșire din rețeaua neurală corespunzând intrării $\mathbf{x}$, iar $\|\mathbf{v}\|_2$ este norma L_2 (Euclidiană) a vectorului $\mathbf{v} = (v_1,\ldots,v_q)^t$:

$$
\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{q} v_i^2}
$$

În acest caz funcția de eroare pentru un set de date $\mathbf{X}$ cu k perechi

$\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$ și incluzând termen de regularizare devine:

$$J(\mathbf{W}, \mathbf{b}) = \underbrace{\left[\frac{1}{2k}\sum_{i=1}^{k}\left\|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right\|_2^2\right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} + \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2}_{\text{Termen de regularizare}} \tag{6.41}$$

Termenul $\frac{1}{k}\sum_{i=1}^{k}\left\|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right\|_2^2$ se numește eroare pătratică medie (mean squared error, MSE). Deși larg folosită pentru probleme de regresie, la o analiză mai atentă se observă ca dă mai multă importanță erorilor mai mari. De exemplu, dă aceeași importanță unei norme de diferență $\|\mathbf{y} - \hat{\mathbf{y}}\|_2 = 10$ ca la 100 de cazuri pentru care $\|\mathbf{y} - \hat{\mathbf{y}}\|_2 = 1$. Se tinde deci a se preocupa de cazuri ce produc erori mari, dar puține, în detrimentul unor cazuri frecvente cu erori mai mici. Funcția de eroare este dominată de valorile de eroare extreme.

O altă funcție de eroare empirică frecvent folosită este eroarea absolută medie (mean absolute error, MAE; eroarea L_1), definită ca (excludem pentru moment eroarea de regularizare):

$$J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}\right) = \frac{1}{k}\|\mathbf{y} - \hat{\mathbf{y}}\|_1 = \frac{1}{k}\sum_{i=1}^{k}|y_i - \hat{y}_i| \tag{6.42}$$

MAE are avantajul că nu e atât de afectată de erorile mari comparativ cu MSE; pentru același exemplu: o eroare de 10 este echivalentă cu doar 10 erori de 1. Dezavantajul ei este că nu e derivabilă peste tot.

O variantă care combină diferențiabilitatea lui MSE și mai mica senzitivitate la erori extreme a lui MAE este funcția de eroare a lui Huber, exprimată în continuare pentru perechea $(\mathbf{y}, \hat{\mathbf{y}})$:

$$HL(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} \frac{1}{2}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 & \text{dacă } \|\mathbf{y} - \hat{\mathbf{y}}\|_2 \leq \delta \\ \delta\left(\|\mathbf{y} - \hat{\mathbf{y}}\|_1 - \frac{\delta}{2}\right) & \text{dacă } \|\mathbf{y} - \hat{\mathbf{y}}\|_2 > \delta \end{cases} \tag{6.43}$$

unde δ este un hiper–parametru care stabilește zona de trecere de la o funcție la alta; valoarea adecvată a lui δ poate fi determinată prin încercări repetate. Indiferent de δ, funcția de eroare a lui Huber este diferențiabilă.

Reprezentările MAE, MSE si Huber pentru o diferență $\|\mathbf{y} - \hat{\mathbf{y}}\|_2$ fixată sunt date în figura 6.9.

6.6.2 Funcția de cost pentru discriminarea a două clase

Dacă problema este de discriminare a două clase, atunci stratul de ieșire conține un singur neuron, cu funcția de activare sigmoidă logistică – idee preluată de la regresia logistică binară. Funcția de eroare pentru o singură

Figura 6.9: Functiile mean squared error, mean absolute error, Huber ($\delta = 0.5$) pentru cazul unidimensional

pereche $(\mathbf{x}, y)$ este[7]:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, y) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y}) \tag{6.44}$$

deci funcția de eroare medie pentru setul de date $\mathbf{X}$, incluzând termenul de regularizare va fi:

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) &= \underbrace{-\frac{1}{k} \sum_{i=1}^{k} \left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln \left(1 - \hat{y}^{(i)}\right) \right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} + \\ &+ \underbrace{\frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2}_{\text{Termen de regularizare}} \end{aligned} \tag{6.45}$$

Pentru forma vectorizată a erorii empirice medii (prima sumă din ecuația de mai sus) a se vedea ecuația (4.18), pagina 57.

[7]Pentru probleme de clasificare binară eticheta de clasă este $y \in \{0, 1\}$. Ieșirea $\hat{y}$ este un număr in intervalul $(0, 1)$.

6.6.3 Funcția de cost pentru mai mult de două clase independente

Pentru anumite probleme se poate cere să se determine dacă intrarea curentă conține sau face parte din niște clase independente. De exemplu, pentru o imagine se cere să se determine dacă conține obiecte de clasă 1, clasă 2, ..., clasă m. Se va crea o rețea neurală cu m neuroni de ieșire, neuronul k dând estimarea ca intrarea să conțină element/să fie de clasă k.

Funcția de eroare pentru o singură pereche $(\mathbf{x}, \mathbf{y})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{m} \left[y_j \ln \hat{y}_j + (1 - y_j) \ln (1 - \hat{y}_j) \right] \tag{6.46}$$

unde vectorul $\mathbf{y} = (y_1, y_2, \ldots, y_m)^t$ conține valori 0 și 1. Funcția de eroare totală $J(\mathbf{W}, \mathbf{b})$ calculată pentru setul de date $\mathbf{X}$ și incluzând termenul de regularizare devine:

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) &= \underbrace{-\frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} \left[y_j^{(i)} \ln \hat{y}_j^{(i)} + \left(1 - y_j^{(i)}\right) \ln \left(1 - \hat{y}_j^{(i)}\right) \right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} + \\ &+ \underbrace{\frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2}_{\text{Termen de regularizare}} \end{aligned} \tag{6.47}$$

6.6.4 Funcția de cost pentru clasificare cu mai mult de două clase

Pentru probleme de clasificare se preferă utilizarea funcției de eroare cross–entropy iar în stratul de ieșire funcția de activare să fie softmax. Funcția de eroare pentru o singură pereche $(\mathbf{x}, \mathbf{y})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{m} y_j \log \hat{y}_j \tag{6.48}$$

unde folosim, ca mai sus, codificarea one–hot. Eroarea pentru setul de date $\mathbf{X}$ și cu termen de regularizare este:

$$J(\mathbf{W}, \mathbf{b}) = \underbrace{-\frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} y_j^{(i)} \log \hat{y}_j^{(i)}}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} + \underbrace{\frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2}_{\text{Termen de regularizare}} \tag{6.49}$$

Pentru forma vectorizată a erorii empirice se poate vedea ecuația (3.39), pagina 46.

6.7 Inițializarea ponderilor rețelei

Valorile inițiale ale ponderilor **W** și **b** sunt setate aleator, în jurul lui zero. Este necesar ca valorile ponderilor să nu fie toate egale; dacă ar fi toate egale, fiecare neuron dintr-un strat ar avea exact aceeași stare ca și ceilalți din stratul lui, deoarece fiecare neuron e legat la exact aceleași intrări ca și ceilați din stratul lui; mai departe, ponderile cu care se înmulțesc intrările fiind egale, valoarea de activare a fiecărui neuron de pe acel strat e aceeași (ponderea constantă folosită se dă factor comun); argumentul e valabil începând cu primul strat ascuns. Am ajunge deci ca neuronii de pe același strat să calculeze exact aceleași valori, ceea ce e redundant și inutil, iar pentru stratul de ieșire s–ar prezice valori egale pe toți neuronii de ieșire. Efectul de simetrie obținut cu ponderi egale în **W** se elimină prin inițializare cu numere aleatoare. În ce privește ponderile de bias – din **b** – ele se pot inițializa cu 0; inițializarea aleatoare a ponderilor **W** este suficientă pentru "spargerea simetriei".

Strategii rafinate de ințializare pentru ponderile sunt cele propuse de Xavier Glorot *et al.*[8] și He *et al.*[9].

Pentru arhitecturile de tip deep learning se preferă o preantrenare nesupervizată a ponderilor [7] sau preluarea unor ponderi care au fost antrenate pentru un set de date similar cu cel curent – transfer de învățare (transfer learning).

6.8 Grafuri computaționale și calcul de derivate pațiale

6.8.1 Derivarea funcțiilor compuse

Pentru algoritmul backpropagation avem nevoie de calcul de derivate parțiale de funcții compuse.

Reamintim definiția derivatei unei funcții $f : \mathbb{R} \to \mathbb{R}$:

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

ceea ce ne permite ca pentru δ valori mici (în jurul lui 0) să folosim aproximarea:

$$f(x+\delta) \approx f(x) + \delta f'(x)$$

[8] Glorot, Xavier and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International Conference on Artificial Intelligence and Statistics (2010).

[9] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026-1034, doi: 10.1109/ICCV.2015.123.

Aproximarea permite sa ne dăm seama că o modificare cu δ a lui x face ca valoarea lui $f(x)$ să se modifice cu $\delta f'(x)$.

Reamintim formula de derivare a funcțiilor compuse: dacă avem $f : B \to C$, $g : A \to B$ și notăm $h(x) = f(g(x))$, $h : A \to C$, atunci derivata lui h în raport cu x se calculează ca[10]:

$$\frac{\partial h}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} \tag{6.50}$$

Pentru funcții multivariate $f : \mathbb{R}^m \to \mathbb{R}$, $g_i, h : \mathbb{R}^n \to \mathbb{R}$ de forma[11]:

$$h(x_1, \ldots, x_n) = f\left(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)\right) \tag{6.51}$$

derivatele parțiale se calculează astfel:

$$\frac{\partial h}{\partial x_i} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x_i} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x_i} + \cdots + \frac{\partial f}{\partial g_m} \cdot \frac{\partial g_m}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial f}{\partial g_j} \cdot \frac{\partial g_j}{\partial x_i} \tag{6.52}$$

În mod clar, formula (6.52) generalizează (6.50).

Exemplificăm (6.51, 6.52): plecăm de la $g_1, g_2 : \mathbb{R} \to \mathbb{R}$, $f : \mathbb{R}^2 \to \mathbb{R}$ diferențiabile. Pentru simplitate, notăm cu h funcția definită prin $h(x) = f(g_1(x), g_2(x))$. Graful computațional este dat în figura 6.10.

Figura 6.10: Graf computațional pentru funcție multivariată și compusă. Variabila x influențează funcția h prin două căi, exemplificând necesitatea folosirii formulei (6.52).

Vrem să calculăm $\partial h / \partial x$. Vom da o motivare intuitivă a egalității

$$\frac{\partial h}{\partial x} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x} \tag{6.53}$$

care e un caz particular al (6.52).

Folosind definția derivatei, avem următoarele aproximări pentru δ apropiat de 0:

$$g_1(x + \delta) \approx g_1(x) + \delta \frac{\partial g_1}{\partial x}(x) \tag{6.54}$$

[10] Chiar și pentru funcții de o singură variabilă vom folosi notația de derivare ∂, pentru omogenitatea notațiilor.

[11] m și n din definițiile f, g_i, h nu au nicio legătură cu dimensiunile vectorilor de intrare și ieșire din setul de instruire.

$$g_2(x+\delta) \approx g_2(x) + \delta \frac{\partial g_2}{\partial x}(x) \tag{6.55}$$

iar pentru δ_1, δ_2 apropiați de 0:

$$\begin{aligned} f(g_1(x)+\delta_1, g_2(x)+\delta_2) &\approx f(g_1(x), g_2(x)) + & (6.56)\\ &+ \delta_1 \frac{\partial f}{\partial g_1}(g_1(x), g_2(x)) + & (6.57)\\ &+ \delta_2 \frac{\partial f}{\partial g_2}(g_1(x), g_2(x)) & (6.58) \end{aligned}$$

Avem deci că

$$\begin{aligned} h(x+\delta) &= f(g_1(x+\delta), g_2(x+\delta)) & (6.59)\\ &\approx f\left(g_1(x) + \delta \frac{\partial g_1}{\partial x}(x), g_2(x) + \delta \frac{\partial g_2}{\partial x}(x)\right) & (6.60) \end{aligned}$$

și folosind (6.56) obținem:

$$\begin{aligned} h(x+\delta) \approx h(x) &+ \delta \frac{\partial g_1}{\partial x}(x) \frac{\partial f}{\partial g_1}(g_1(x), g_2(x))\\ &+ \delta \frac{\partial g_2}{\partial x}(x) \frac{\partial f}{\partial g_2}(g_1(x), g_2(x)) & (6.61) \end{aligned}$$

ce se transcrie ca

$$\begin{aligned} \frac{\partial h}{\partial x}(x) &\approx \frac{h(x+\delta) - h(x)}{\delta}\\ &\approx \frac{\partial g_1}{\partial x}(x) \cdot \frac{\partial f}{\partial g_1}(g_1(x), g_2(x)) +\\ &+ \frac{\partial g_2}{\partial x}(x) \cdot \frac{\partial f}{\partial g_2}(g_1(x), g_2(x)) & (6.62) \end{aligned}$$

ceea ce e apropiat de (6.53). Există, desigur, o demonstrație riguroasă pentru (6.52).

Formula (6.52) este utilă pentru cazul în care de la o variabilă – x în cazul nostru – la rezultat – $h(x)$ – există mai mult de o cale în graful computațional – via g_1 și g_2 în cazul dat. Pentru o rețea neurală cu măcar un strat ascuns, orice pondere $w_{ij}^{[k]}$ de dinainte de ultimul strat ascuns intervine în calculul ieșirii rețelei – și deci al funcției de eroare, pentru care se calculează gradienții – prin intermediul tuturor căilor de la neuronul i din stratul k la stratul de ieșire. Pentru un exemplu, a se vedea secțiunea 6.8.3.

6.8.2 Exemplul 1: căi unice

Exemplificăm și figurăm în cele ce urmează calculul de derivate parțiale, așa cum o efecturază metoda backpropagation pentru funcția $f : \mathbb{R}^3 \to \mathbb{R}$,

Figura 6.11: Graful computațional pentru calculul valorii funcției $f(x, y, z) = x \cdot (y + z) = x \cdot q(y, z)$ pentru $x = 3, y = 4, z = 5$. Valorile $q(y, z)$ și $f(x, y, z)$ se calculează de la stânga spre dreapta, prin propagare înainte.

$f(x, y, z) = x \cdot (y + z)$. Notăm cu q funcția $q(y, z) = y + z$, deci $f(x, y, z) = x \cdot q(y, z)$. Graful computațional pentru calculul valorii lui f este în figura 6.11. Dorim să calculăm derivatele parțiale ale lui f în raport cu x, y, z.

Fixăm $x = 3, y = 4, z = 5$. Calculul valorilor q, f este dat în graful computațional din figura 6.11. Valorile lui q și f se calculează de la stânga spre dreapta, prin propagare înainte, în sensul dat de săgețile negre.

Derivatele parțiale se calculează de la dreapta spre stânga (propagare inversă, eng. backropagation) și este calculul este arătat în figura 6.12. .

Figura 6.12: Graful computațional pentru calculul derivatelor parțiale $\partial f/\partial x$, $\partial f/\partial y$, $\partial f/\partial h$, pentru $x = 3, y = 4, z = 5$. Derivatele parțiale se calculează de la dreapta spre stânga. Utilizarea repetată a derivatelor parțiale "din amonte" este arătată prin linie punctată.

Astfel, calculăm începând de la ieșirea grafului computațional către variabilele de intrare, în ordine:

1. Un calcul imediat, dar util pentru a asigura omogenitatea implementării calculelor ce urmează:
$$\frac{\partial f}{\partial f} = 1 \tag{6.63}$$

2. Calculăm derivata parțială $\partial f/\partial q$, făcând înmulțind derivata "din amonte" $\partial f/\partial f$ cu derivata locală $\partial f/\partial q$. Derivatele locale sunt derivatele funcției reprezentate de nodul curent în raport cu fiecare din intrările sale. Una din derivatele locale este:

$$\frac{\partial f}{\partial q} = \frac{\partial (x \cdot q)}{\partial q} = x = 3 \tag{6.64}$$

Avem deci

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial q} = 1 \cdot 3 = 3 \tag{6.65}$$

3. Pentru calculul derivatei parțiale $\partial f/\partial y$, folosind regula derivării funcțiilor compuse, avem: $\partial f/\partial y = \partial f/\partial q \cdot \partial q/\partial y$, deci păstrăm principiul: se înmulțește derivata parțială din amonte – și deja calculată la pasul anterior, $\partial f/\partial q$ – cu derivata locală $\partial q/\partial y$:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = 3 \cdot \frac{\partial (y+z)}{\partial y} = 3 \cdot 1 = 3 \tag{6.66}$$

4. Similar, pentru calculul derivatei parțiale $\partial f/\partial z$ avem:

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial z} = 3 \cdot 1 = 3 \tag{6.67}$$

5. Pentru calculul lui $\partial f/\partial x$ avem:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial x} = 1 \cdot \frac{\partial (x \cdot q)}{\partial x} = q = 9 \tag{6.68}$$

Sumarizând, calculăm derivatele parțiale de la ieșire către intrare și folosim derivatele calculate până la acel pas pentru a determina derivatele aflate din ce în ce mai aproape de intrarea în graful computațional. Principiul din figura 6.12 stă la baza unor algoritmi de calcul automat al gradienților – bibliotecile de tip *autograd*.

6.8.3 Exemplul 2: căi multiple

Considerăm funcția $f : \mathbb{R}^3 \to \mathbb{R}$, $f(x, y, z) = (x + y) \cdot (y + z)$. Notăm expresiile intermediare: $p(x, y) = x + y$, $q(y, z) = y + z$ și deci $f(x, y, z) = p(x, y) \cdot q(y, z)$. Fixăm $x = -1$, $y = 3$ și $z = 5$. Graful computațional e dat în figura 6.13.

Se observă că y contribuie la rezultatul final prin două căi - via p și q, deci vom folosi rezultatul din ecuația (6.52).

Vrem să calculăm derivatele parțiale $\partial f/\partial x$, $\partial f/\partial y$, $\partial f/\partial z$. Calculăm derivatele locale:

Figura 6.13: Graf computațional pentru $f(x, y, z) = (x + y) \cdot (y + z)$, $x = -1$, $y = 3$, $z = 5$. Se fac calcule intermediare dec către nodurile p și q.

$$\frac{\partial f}{\partial p} = \frac{\partial (p \cdot q)}{\partial p} = q = 8 \qquad \frac{\partial f}{\partial q} = \frac{\partial (p \cdot q)}{\partial q} = p = 2 \qquad \frac{\partial p}{\partial x} = \frac{\partial (x + y)}{\partial x} = 1$$

$$\frac{\partial p}{\partial y} = \frac{\partial (x + y)}{\partial y} = 1 \qquad \frac{\partial q}{\partial y} = \frac{\partial (y + z)}{\partial y} = 1 \qquad \frac{\partial q}{\partial z} = \frac{\partial (y + z)}{\partial z} = 1$$

Gradienții lui f se calculează de la dreapta spre stânga. Se începe cu evidentul $\partial f / \partial f = 1$, iar restul de calcule e dat în figura 6.14. Derivata parțială $\partial f / \partial y$ este deosebită de celelalte: pentru că y dă valoarea ei atât lui p cât și lui q, ea primește "din amonte" gradienți, via nodurile p și q, pe care îi însumează. Însumarea e necesară conform formulei (6.52):

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial y} + \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = 8 + 2 = 10 \tag{6.69}$$

Figura 6.14: Calculul gradienților pentru $f(x, y, z) = (x + y) \cdot (y + z)$ folosind gradienții "din amonte" și gradienții locali. Variabila y influențează valoarea funcției f prin două căi.

6.9 Algoritmul backpropagation

6.9.1 Scopul algoritmului

Se dorește modificarea ponderilor din matricele $\mathbf{W}^{[l]}$ și a coeficienților de bias $\mathbf{b}^{[l]}$ astfel încât valoarea funcției de eroare $J(\mathbf{W}, \mathbf{b})$ să scadă; numim modificările ponderilor învățare. Se va folosi algoritmul de căutare după direcția gradientului (gradient descent), în care modificarea unei ponderi $w_{ij}^{[l]}$ se efectuează astfel:

$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{6.70}$$

Ponderile de bias $b_{ij}^{[l]}$ se modifică similar:

$$b_{i}^{[l]} = b_{i}^{[l]} - \alpha \frac{\partial J}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{6.71}$$

deci este esențială calcularea gradienților $\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b})$, $\frac{\partial J}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b})$. Ca și până acum, $\alpha > 0$ este rata de învățare.

6.9.2 Variante de calcul

Avem trei variante de calcul pentru modificarea ponderilor:

1. **stochastic gradient descent:** pentru fiecare pereche din setul de instruire $\left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)$ se calculează valoarea erorii $J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)$, se calculează gradienții și se aplică modificările pentru toate ponderile $w_{ij}^{[l]}$ și $b_{i}^{[l]}$; următoarea pereche de instruire folosește valorile de ponderi modificate la acest pas;

2. **off–line** sau **batch gradient descent**: se calculează gradienții pentru ponderile $w_{ij}^{[l]}$ și bias–urile $b_{i}^{[l]}$ pentru fiecare pereche de vectori din setul de instruire; la final se calculează media lor și se actualizează fiecare pondere $w_{ij}^{[l]}$ și $b_{i}^{[l]}$, scăzându–se din ea media înmulțită cu rata de învățare α.

3. **minibatch gradient descent:** se împarte setul de instruire $\mathscr{S}$ în subseturi disjuncte (mini–batches), de exemplu de câte 100 de perechi $\left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)$. Pentru fiecare minibatch se calculează media gradienților; se modifică toate ponderile $w_{ij}^{[l]}$ și $b_{i}^{[l]}$ folosind media înmulțită cu rata de învățare, apoi se trece la următorul minibatch. Este o variantă intermediară între stochastic gradient descent – unde modificarea se face imediat după fiecare pereche din $\mathscr{S}$ – și cea batch – în care modificarea ponderilor se face doar după ce se procesează tot setul $\mathscr{S}$; în practică este cea mai folosită strategie.

În toate cazurile de mai sus: o trecere completă peste setul de instruire se numește epocă, iar trecerea peste un subset – fie el și dintr-un singur exemplar de instruire – se numește iterație. Se execută mai multe epoci de instruire. Condiția de oprire a învățării poate fi:

- numărul de epoci parcurse este egal cu un număr dat a priori, de exemplu 100;
- se urmărește valoarea funcției de eroare peste un set de validare, un set disjunct față de setul de instruire $\mathscr{S}$. Dacă se constată că eroarea pe setul de validare începe să crească în timp ce eroarea pe setul de instruire continuă să scadă, atunci se oprește instruirea – figura 6.15;
- se urmărește evoluția normelor gradienților: dacă suma acestor norme e aproape de zero, înseamnă că s–a ajuns într-un punct de minim (local sau global) sau într–un punct de inflexiune și ponderile nu vor mai fi modificate semnificativ;
- valoarea funcției de eroare scade sub un anumit prag; etc.

Figura 6.15: Exemplu de evoluție a valorilor funcției de eroare pentru setul de antrenare, respectiv cel de validare. Dacă se continuă antrenarea, eroare pe setul de antrenare scade, dar pentru setul de testare începe să crească de la o anumită epocă. Se recomandă oprirea instruirii dacă eroarea pe setul de validare începe să crească.

6.9.3 Calculul gradienților – justificare matematică

Vom prezenta varianta de instruire *batch*, întrucât poate fi ușor adaptată la minibatch sau stochastic gradient descent. Trecerea la cazul în care se

face antrenarea pe minibatch-uri este imediată: media gradienților pentru p termeni din setul de instruire se substituie cu media gradienților calculați pe datele din acel minibatch. Evident, pentru stochastic gradient descent, media este chiar gradientul calculat pentru exemplarul de instruire curent.

Reamintim forma funcției de eroare pentru un set de date $\mathbf{X}$ de p perechi $\left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)$:

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) &= \underbrace{\left[\frac{1}{p}\sum_{k=1}^{p} \overbrace{J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)}^{\text{eroare empirică pentru}\left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)}\right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} \\ &+ \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_l}\sum_{j=1}^{n_{l-1}}\left(w_{ij}^{[l]}\right)^2}_{\text{Termen de regularizare}} \end{aligned} \tag{6.72}$$

Ne folosim de faptul că funcția de eroare este o sumă de termeni și derivata unei sume de funcții este suma derivatelor funcțiilor[12] și rezultă:

$$\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p}\sum_{k=1}^{p}\frac{\partial J}{\partial w_{ij}^{[l]}}\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right) + \lambda w_{ij}^{[l]} \tag{6.73}$$

respectiv pentru ponderile de bias:

$$\frac{\partial J}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p}\sum_{k=1}^{p}\frac{\partial J}{\partial b_{i}^{[l]}}\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right) \tag{6.74}$$

În (6.73) am inclus derivatele parțiale pentru termenul de regularizare (6.39) în raport cu $w_{ij}^{[l]}$, al căror calcul e imediat:

$$\frac{\partial}{\partial w_{ij}^{[l]}}\left(\frac{\lambda}{2}\cdot\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2\right) = \frac{\lambda}{2}\frac{\partial}{\partial w_{ij}^{[l]}}\sum_{l=1}^{L-1}\sum_{a=1}^{n_l}\sum_{b=1}^{n_{l-1}}\left(w_{ab}^{[l]}\right)^2 = \lambda\cdot w_{ij}^{[l]} \tag{6.75}$$

iar pentru derivata parțială a termenului de regularizare în raport cu $b_i^{[l]}$ reamintim că termenul de regularizare nu conține ponderile de bias, deci gradientul său în raport cu bias–urile este 0.

Rămâne deci să calculăm derivatele parțiale pentru o singură pereche de instruire $(\mathbf{x}, \mathbf{y})$, adică $\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$ și $\frac{\partial J}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$. Algoritmul backpropagation arată care e ordinea de calcul a derivatelor parțiale, bazat pe teoria din secțiunea 6.8. Odată derivatele parțiale calculate, ele pot fi folosite pentru modificarea ponderilor, proces pe care îl numin învățare.

[12] Sau pe scurt: derivarea este operator liniar.

Algoritmul a fost popularizat de articolul “Learning representations by back-propagating errors”[13].

Arătăm modul de calcul pentru derivatele parțiale $\partial J/\partial w_{ij}^{[k]}$ și $\partial J/\partial b_i^{[k]}$, cu exemplificare pe o rețea neurală pentru regresie și cu un singur strat ascuns, a se vedea figura 6.16. Fiind problemă de regresie, funcția de eroare empirică (de calitate) e jumătate din mean squared error, ecuația (6.40) pagina 88. Pentru problemă de clasificare se adaptează funcția de activare $f^{[2]}$ a ultimului strat. Vom explicita doar gradienții termenului de eroare de calitate, deoarece am văzut că gradienții termenului de regularizare L_2 se calculează simplu. Prezentăm calculul pentru o singură pereche de instruire $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^m$.

Figura 6.16: Rețea neurală cu un strat ascuns, folosită ca schemă pentru determinarea fomulelor gradienților.

Gradienții pentru stratul de ieșire

Pentru ponderile instruibile ale ultimului strat, $w_{ij}^{[2]}$ și $b_i^{[2]}$ unde $1 \leq i \leq m$, avem graful computațional din figura 6.17.

Figura 6.17: Graf computațional pentru ponderile și valorile de bias din ultimul strat.

[13] Rumelhart, D., Hinton, G. & Williams, R., *Learning representations by back-propagating errors*, Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

Deoarece J nu depinde direct de $w_{ij}^{[2]}$, ci via $z_i^{[2]}$, folosim formula de derivare de funcții compuse (6.50) și obținem:

$$\frac{\partial J}{\partial w_{ij}^{[2]}} = \frac{\partial J}{\partial z_i^{[2]}} \cdot \frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}} \tag{6.76}$$

Pentru a doua derivată parțială ținem cont că:

$$z_i^{[2]} = b_i^{[2]} + \sum_{k=1}^{n_1} w_{ik}^{[2]} a_k^{[1]} \tag{6.77}$$

de unde

$$\frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}} = a_j^{[1]} \tag{6.78}$$

și deci

$$\frac{\partial J}{\partial w_{ij}^{[2]}} = \frac{\partial J}{\partial z_i^{[2]}} \cdot a_j^{[1]} \tag{6.79}$$

Similar, pentru derivata parțială a lui J în raport cu $b_i^{[2]}$:

$$\frac{\partial J}{\partial b_i^{[2]}} = \frac{\partial J}{\partial z_i^{[2]}} \cdot \frac{\partial z_i^{[2]}}{\partial b_i^{[2]}} = \frac{\partial J}{\partial z_i^{[2]}} \cdot \frac{\partial \left(b_i^{[2]} + \sum\limits_{k=1}^{n} w_{ik}^{[2]} a_{ik}^{[1]} \right)}{\partial b_i^{[2]}} = \frac{\partial J}{\partial z_i^{[2]}} \tag{6.80}$$

Observăm că avem termenul $\partial J / \partial z_i^{[2]}$ în ambele derivate parțiale; îl notăm cu $\delta_i^{[2]}$ și îl calculăm. Ținem cont că J depinde de $z_i^{[2]}$ doar prin intermediul lui $a_i^{[2]} = f^{[2]}\left(z_i^{[2]}\right)$ și folosim din nou formula de derivare de funcții compuse:

$$\begin{aligned}
\delta_i^{[2]} &= \frac{\partial J}{\partial z_i^{[2]}} = \frac{\partial J}{\partial a_i^{[2]}} \cdot \frac{\partial a_i^{[2]}}{\partial z_i^{[2]}} && (6.81)\\
&= \left(\frac{\partial}{\partial a_i^{[2]}} \frac{1}{2} \sum_{k=1}^{m} (a_k^{[2]} - y_k)^2 \right) \cdot \frac{\partial f^{[2]}}{\partial z_i^{[2]}} \left(z_i^{[2]} \right) && (6.82)\\
&= \left(a_i^{[2]} - y_i \right) \cdot f^{[2]\prime} \left(z_i^{[2]} \right) && (6.83)
\end{aligned}$$

de unde, vectorial:

$$\boldsymbol{\delta}^{[2]} = \left(\mathbf{a}^{[2]} - \mathbf{y} \right) \odot f^{[2]\prime} \left(\mathbf{z}^{[2]} \right) \tag{6.84}$$

cu $\odot$ reprezentând produsul Hadamard:

$$(a_1, \ldots a_p)^t \odot (b_1, \ldots b_p)^t = (a_1 \cdot b_1, \ldots a_p \cdot b_p)^t$$

Componentele vectorului $\boldsymbol{\delta}^{[2]}$ sunt folosite pentru calculul derivatelor parțiale de ponderi instruibile, conform (6.79) și (6.80).

Gradienții pentru stratul ascuns

Pentru calcul de derivate parțiale de ponderi și valori de bias ale stratului ascuns, $\partial J/\partial w_{ij}^{[1]}$ și respectiv $\partial J/\partial b_i^{[1]}$ ținem cont că $w_{ij}^{[1]}$ și $b_i^{[1]}$ contribuie la funcția de eroare prin mai multe căi, conform grafului computațional din figura 6.18; indicii sunt $1 \leq i \leq n_1$, $1 \leq j \leq n$.

Figura 6.18: Graf computațional pentru ponderile și valorile de bias din stratul ascuns. Ponderile instruibile $w_{ij}^{[1]}$ și $b_i^{[1]}$ contribuie la valoarea funcției de eroare prin mai multe căi, via $z_1^{[2]}, \ldots, z_m^{[2]}$. Situația e similară cu cea din figura 6.10.

Ca atare, pentru $\partial J/\partial w_{ij}^{[1]}$ vom utiliza formula generală de derivare a funcțiilor compuse (6.52).

Avem:

$$\frac{\partial J}{\partial w_{ij}^{[1]}} = \frac{\partial J}{\partial z_i^{[1]}} \cdot \frac{\partial z_i^{[1]}}{\partial w_{ij}^{[1]}} \tag{6.85}$$

Explicităm cele două derivate parțiale de sub suma de mai sus. Deoarece

$$z_i^{[1]} = b_i^{[1]} + \sum_{k=1}^{n_1} w_{ik}^{[1]} a_k^{[0]} \tag{6.86}$$

obținem imediat

$$\frac{\partial z_i^{[1]}}{\partial w_{ij}^{[1]}} = a_j^{[0]} \tag{6.87}$$

și înlocuind înapoi în (6.85):

$$\frac{\partial J}{\partial w_{ij}^{[1]}} = \frac{\partial J}{\partial z_i^{[1]}} \cdot a_j^{[0]} \tag{6.88}$$

Similar, pentru derivata parțială în raport cu termenul de bias:

$$\frac{\partial J}{\partial b_i^{[1]}} = \frac{\partial J}{\partial z_i^{[1]}} \cdot \frac{\partial \left(b_i^{[1]} + \sum\limits_{k=1}^{n_1} \sum\limits_{k=1}^{n_1} w_{ik}^{[1]} a_k^{[0]} \right)}{\partial b_i^{[1]}} = \frac{\partial J}{\partial z_i^{[1]}} \tag{6.89}$$

Cantitatea $\frac{\partial J}{\partial z_i^{[1]}}$ apare atât în (6.89) cât și în (6.88); o vom nota cu $\delta_i^{[1]}$ și cele două formule devin:

$$\frac{\partial J}{\partial w_{ij}^{[1]}} = \delta_i^{[1]} \cdot a_j^{[0]} \tag{6.90}$$

$$\frac{\partial J}{\partial b_i^{[1]}} = \delta_i^{[1]} \tag{6.91}$$

Rămâne deci să calculăm $\delta_i^{[1]} = \frac{\partial J}{\partial z_i^{[1]}}$. Deoarece $z_i^{[1]}$ influențează funcția J prin intermediul variabilelor $z_1^{[2]}, \ldots, z_m^{[2]}$ – vezi figura 6.18 – folosim formula (6.52) și obținem:

$$\delta_i^{[1]} = \frac{\partial J}{\partial z_i^{[1]}} = \sum_{j=1}^{m} \frac{\partial J}{\partial z_j^{[2]}} \cdot \frac{\partial z_j^{[2]}}{\partial z_i^{[1]}} \tag{6.92}$$

Dar $\frac{\partial J}{\partial z_j^{[2]}}$ este chiar $\delta_j^{[2]}$, calculat anterior în (6.84). Explicităm $\frac{\partial z_j^{[2]}}{\partial z_i^{[1]}}$, $1 \leq i \leq n_1$, și pentru înțelegerea necesității însumării a se revedea figura 6.4b:

$$\begin{aligned} \frac{\partial z_j^{[2]}}{\partial z_i^{[1]}} &= \frac{\partial}{\partial z_i^{[1]}} \left(\sum_{l=1}^{n_1} w_{jl}^{[2]} \cdot f^{[1]} \left(z_l^{[1]} \right) \right) \\ &= w_{ji}^{[2]} \cdot f^{[1]\prime} \left(z_i^{[1]} \right) \end{aligned} \tag{6.93}$$

Înlocuind în (6.92) obținem:

$$\delta_i^{[1]} = \sum_{j=1}^{m} \left(\delta_j^{[2]} \cdot w_{ji}^{[2]} \cdot f^{[1]\prime} \left(z_i^{[1]} \right) \right) \tag{6.94}$$

sau pentru tot vectorul $\boldsymbol{\delta}^{[1]}$:

$$\boldsymbol{\delta}^{[1]} = \left(\mathbf{W}^{[2]t} \cdot \boldsymbol{\delta}^{[2]} \right) \odot f^{[1]\prime} \left(z^{[1]} \right) \tag{6.95}$$

Valorile din vectorul $\boldsymbol{\delta}^{[1]}$ sunt folosite pentru calculul de derivatelor parțiale din ecuațiile (6.90) și respectiv (6.91).

Extinderea la o rețea MLP cu mai multe straturi ascunse este ușor de făcut, fiind foarte similară cu ceea ce s–a arătat mai sus.

6.9.4 Algoritmul backpropagation

Algoritmul backpropagation detaliat – varianta batch – este:

1. Inițializează $\Delta\mathbf{W}^{[l]}$, $\Delta\mathbf{b}^{[l]}$ cu 0, pentru $l = 1, \ldots, L-1$:

$$\Delta\mathbf{W}^{[l]} = \mathbf{0}_{n_l \times n_{l-1}} \tag{6.96}$$

$$\Delta\mathbf{b}^{[l]} = \mathbf{0}_{n_l}, \text{vector coloană} \tag{6.97}$$

2. Pentru fiecare din cele p perechi $(\mathbf{x}, \mathbf{y})$ din batch calculează corecția pentru ponderi și ponderile de bias[14]:

   2.1. Efectuează pasul de propagare înainte, conform secțiunii 6.5, și obține ieșirea estimată $\hat{\mathbf{y}}$;

   2.2. Pentru fiecare strat $l = L-1, \ldots, 1$ se calculează semnalul de eroare: la ultimul strat semnalul de eroare este

$$\boldsymbol{\delta}^{[L-1]} = (\mathbf{a}^{[L-1]} - \mathbf{y}) \odot f^{[L-1]'}\left(\mathbf{z}^{[L-1]}\right) \tag{6.98}$$

   iar la restul de straturi

$$\boldsymbol{\delta}^{[l]} = \left[\left(\mathbf{W}^{[l+1]}\right)^t \cdot \boldsymbol{\delta}^{[l+1]}\right] \odot f^{[l]'}\left(\mathbf{z}^{[l]}\right) \tag{6.99}$$

   Am presupus mai sus că derivatele funcțiilor de activare se calculează punctual peste vectorii de stări.

   Putem acum calcula derivatele parțiale pentru ponderi și derivatele parțiale:

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = \boldsymbol{\delta}^{[l]} \cdot \left(\mathbf{a}^{[l-1]}\right)^t \tag{6.100}$$

$$\frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = \boldsymbol{\delta}^{[l]} \tag{6.101}$$

   2.3. Acumulează semnalul de corecție[15], pentru $l = L-1, \ldots, 1$:

$$\Delta\mathbf{W}^{[l]} = \Delta\mathbf{W}^{[l]} + \frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) \tag{6.102}$$

$$= \Delta\mathbf{W}^{[l]} + \boldsymbol{\delta}^{[l]} \cdot \left(\mathbf{a}^{[l-1]}\right)^t \tag{6.103}$$

$$\Delta\mathbf{b}^{[l]} = \Delta\mathbf{b}^{[l]} + \frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) \tag{6.104}$$

$$= \Delta\mathbf{b}^{[l]} + \boldsymbol{\delta}^{[l]} \tag{6.105}$$

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

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha\left[\left(\frac{1}{p}\Delta\mathbf{W}^{[l]}\right) + \lambda\mathbf{W}^{[l]}\right] \tag{6.106}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha\left(\frac{1}{p}\Delta\mathbf{b}^{[l]}\right) \tag{6.107}$$

[14] Respectiv: se iterează peste datele din minibatch. Alternativ, se poate face calcul în paralel peste datele din minibatchul curent.

[15] Evident, acumularea se poate face imediat după calculul derivatelor parțiale de pe stratul curent.

4. Ciclare: se repetă de la pasul 1 până când se îndeplinește una din condițiile de oprire de la pagina 99.

Pentru cazul în care se folosește clasificare și funcția de activare din ultimul strat este softmax, iar funcția de eroare este cross-entropy, se arată că pentru perechea $(\mathbf{x}, \mathbf{y})$:

$$\boldsymbol{\delta}^{[L-1]} = \mathbf{a}^{[L-1]} - \mathbf{y} \tag{6.108}$$

substituind formula de calcul (6.98). Utilizarea funcției de eroare cross entropy duce la o viteză mai mare de învățare pentru probleme de clasificare decât dacă se folosește eroarea pătratică [8].

6.10 Utilizarea rețelei pentru inferență

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

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

$$\hat{y}_i = P\left(\text{clasa } i | \mathbf{x}\right), 1 \leq i \leq m \tag{6.109}$$

atunci clasificarea se face găsind acel indice i pentru care $\hat{y}_i$ e maxim și acesta este indicele clasei prezise de rețeaua MLP.

6.11 Discuții

- problema minimelor locale și a punctelor de inflexiune
- arhitectura rețelei: număr de straturi ascunse, număr de neuroni pe strat
- dependența de ordinea de prezentare a datelor în etapa de învățare
- alți algoritmi de optimizare decât vanilla gradient descent
- learning rate scheduling
- capacitatea de aproximare universală

Capitolul 7

Rețele neurale cu funcții de activare radială

Rețelele neurale cu funcții de activare radială (eng: radial basis function networks, RBF networks) se folosesc pentru probleme de clasificare, estimare de probabilitate condiționată și regresie. Instruirea este nesupervizată (grupare, eng. clustering) pentru determinarea centrilor neuronilor din stratul ascuns și supervizată pentru învățarea ponderilor dintre stratul ascuns și cel de ieșire.

O rețea RBF stochează în stratul ascuns vectori prototip, determinați prin învățare. Inferența pentru un vector de intrare se face pe baza distanțelor dintre el și prototipuri.

7.1 Motivația rețelei: teorema lui Cover

Pentru cazul vectorilor ce nu pot fi separați liniar, perceptronul multistrat poate determina o suprafață de separare. Există și o altă variantă de rezolva problema discernerii între clase ce nu se pot separa liniar, *folosind însă un separator liniar*, ce lucrează în doi pași:

1. mulțimea de instruire dată în spațiul originar 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 liniar, SVM liniar, regresie logistică etc.) se separă clasele în cel de-al doilea spațiu.

Intuitiv, situația este prezentată în figura 7.1. În partea stângă avem două clase care nu sunt separabile liniar; punctele sunt cu coordonate în două dimensiuni. Printr–o funcție convenabil aleasă se trece într–un spațiu cu trei dimensiuni în care un plan poate separa clasa punctelor roșii de cele albastre.

Figura 7.1: Transformarea unui set de date cu clase neseparabile liniar, într-un spațiu cu două dimensiuni, într-o mulțime cu două clase liniar separabile în spațiu cu 3 dimensiuni.

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

- strat de intrare în care se furnizează vector de valori pentru care se face inferență;
- strat de neuroni ascunși ce aplică transformări neliniare pe datele din spațiul de intrare. Neuronii din acest strat sunt antrenați prin instruire nesupervizată;
- strat de ieșire ce produce o transformare liniară, iar ponderile dintre stratul ascuns și stratul de ieșire sunt obținute prin instruire supervizată. Acest ultim strat furnizează valoarea inferată pentru vectorul de intrare curent.

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 originar:

Teorema 2 (Cover, 1965)**.** *Printr–o transformare neliniară a unui set de date de intrare dintr-un spațiu A într-un spațiu B cu dimensiune mai mare, crește probabilitatea de a obține clase liniar separabile, dacă A nu este dens populat.*

Rezultatul este util, deoarece pentru cazuri liniar separabile, un perceptron discret poate să obțină un hiperplan de separare în timp finit; se pot folosi si alte modele de clasificatori liniari - regresie logistică, Support Vector Machines etc. Pentru a se obține o asemenea transformare, se pleacă de la spațiul A cu n dimensiuni în care se găsesc niște vectori $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}$ și se ajunge la un spațiu m dimensional ($m \geq n$) prin funcția:

$$\mathbf{x} \xrightarrow{\phi} \boldsymbol{\phi}(\mathbf{x}) = (\varphi_1(\mathbf{x}), \ldots, \varphi_m(\mathbf{x}))^t \in \mathbb{R}^m \tag{7.1}$$

unde $\varphi_i : \mathbb{R}^n \to \mathbb{R}, i = \overline{1, m}$ sunt funcții neliniare; în rețeaua neurală RBF funcția φ_i e calculată de neuronul i din stratul ascuns.

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

Tabelul 7.1: Valorile aproximative ale funcțiilor φ_i pentru punctele problemei XOR.

Exemplu: considerăm problema XOR, în care 4 puncte sunt asignate la două clase, astfel: vectorii $\mathbf{x}_1 = (1,1)^t$ și $\mathbf{x}_2 = (0,0)^t$ aparțin unei clase, iar $\mathbf{x}_3 = (0,1)^t$ și $\mathbf{x}_4 = (1,0)^t$ aparțin celeilalte clase. S–a arătat în secțiunea 5.9 că nu există o dreaptă în plan care să separe cele două clase de puncte. Considerăm funcțiile $\varphi_1, \varphi_2 : \mathbb{R}^2 \to \mathbb{R}$:

$$\varphi_1(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{x}_1\|\right)$$

$$\varphi_2(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{x}_2\|\right)$$

unde $\mathbf{x} \in \mathbb{R}^2$ iar $\|\cdot\|$ este norma Euclidiană în $\mathbb{R}^2$. 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 7.1. Figura 7.2 dă reprezentarea punctelor tranformate 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.

Figura 7.2: Transformarea problemei XOR, ce nu permite separare liniară (partea stângă) în problemă liniar separabilă

7.2 Funcții cu activare radială

Teorema lui Cover afirmă că pentru o problemă ce nu e liniar separabilă, prin transformare adecvată 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 nod[1]. Această rețea produce o funcție de la un spațiu n–dimensional la unul unidimensional:

$$F : \mathbb{R}^n \to \mathbb{R} \tag{7.2}$$

Funcția F 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 inferență. 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 F. În etapa de inferență se folosește un procedeu de interpolare pentru a determina valori de ieșire corespunzătoare unor vectori din spațiul de intrare $\mathbb{R}^n$.

Plecăm de la o problemă de interpolare enunțată astfel:
Dându–se un set de instruire de p perechi

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | \mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}, i = \overline{1,p} \right\}$$

se cere să se determine o funcție $F : \mathbb{R}^n \to \mathbb{R}$ care satisface proprietatea de interpolare:

$$F\left(\mathbf{x}^{(i)}\right) = y^{(i)}, \quad i = \overline{1,p} \tag{7.3}$$

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

$$F(\mathbf{x}) = \sum_{i=1}^{p} w_i \varphi_i \left(\left\| \mathbf{x} - \mathbf{x}^{(i)} \right\| \right) \tag{7.4}$$

unde φ_i sunt funcții neliniare, cunoscute ca funcții cu activare radială. Punctele $\mathbf{x}^{(i)}$ sunt "centrele" (parametri ai) funcțiilor RBF. Pentru un vector $\mathbf{x} \in \mathbb{R}^n$, valoarea returnată de funcția φ_i se bazează pe distanța dintre $\mathbf{x}$ și $\mathbf{x}^{(i)}$, adică pe $\|\mathbf{x} - \mathbf{x}^{(i)}\|$.

Impunând condiția (7.3) asupra formei (7.4), avem următorul sistem liniar în care necunoscutele sunt $w_i, i = \overline{1,p}$:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1p} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{p1} & \varphi_{p2} & \cdots & \varphi_{pp} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(p)} \end{bmatrix} \tag{7.5}$$

unde

$$\varphi_{ij} = \varphi_i \left(\left\| \mathbf{x}^{(j)} - \mathbf{x}^{(i)} \right\| \right), \; i, j = \overline{1,p} \tag{7.6}$$

[1] În 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.

Notăm $\mathbf{y} = \left(y^{(1)}, y^{(2)}, \ldots, y^{(p)}\right)^t$, $\mathbf{w} = (w_1, w_2, \ldots, w_p)^t$, $\mathbf{\Phi} = (\varphi_{ij})_{i,j=\overline{1,p}}$. Numim $\mathbf{\Phi}$ matricea de interpolare. Se poate rescrie (7.5) sub forma:

$$\mathbf{\Phi} \cdot \mathbf{w} = \mathbf{y} \tag{7.7}$$

Dacă matricea $\mathbf{\Phi}$ este nesingulară, atunci ponderile sunt $\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{y}$. Pentru discuția asupra caracterului nesingular al matricei $\mathbf{\Phi}$, 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 $\mathbf{\Phi}$ este inversabilă dacă funcțiile φ_i au una din formele:*

1. *funcție multipătratică:*

$$\varphi_i(r_i) = \sqrt{r_i^2 + c^2},\ c > 0 \tag{7.8}$$

2. *funcție inversă de multipătratică:*

$$\varphi_i(r_i) = \frac{1}{\sqrt{r_i^2 + c^2}},\ c > 0 \tag{7.9}$$

3. *funcție Gaussiană:*

$$\varphi_i(r_i) = \exp\left(-\frac{r_i^2}{2\sigma^2}\right),\ \sigma > 0 \tag{7.10}$$

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 $\mathbf{x}^{(i)}$, $\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|$.

7.3 Rețele cu funcții cu activare radială

O rețea cu funcții cu activare radială este ilustrată în figura 7.3; 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 i, $i = \overline{1,p}$ are funcție cu activare radială $\varphi_i\left(\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|\right)$, unde φ_i e funcție cu una din formele enunțate în Teorema 3;

3. *stratul de ieșire*, care în cazul exemplificat este format dintr-un singur neuron. Stratul de ieșire poate avea m neuroni, pentru a trata problemele de clasificare sau de estimare de probabilitate condiționată pentru m clase, sau pentru probleme de regresie cu m ieșiri.

Pentru funcțiile φ_i vom considera în continuare funcțiile Gaussiene:

$$\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|\right)=\exp\left(-\frac{1}{2\sigma_i^2}\left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|^2\right),\quad i=\overline{1,p} \tag{7.11}$$

cu $\sigma_i > 0$ este parametru al funcție Gaussiene φ_i centrate în $\mathbf{x}^{(i)}$. De regulă tuturor Gaussienelor li se asigneaza aceeași lățime σ; diferența dintre funcții este dată în acest caz de centrele $\mathbf{x}^{(i)}$.

Figura 7.3: Structura unei rețele RBF, plecând de la funcția de interpolare din ecuația 7.4.

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 $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) | i=\overline{1,p}\right\}$ poate prezenta zgomot, de exemplu datorită erorilor de măsurare. Folosirea unui procedeu de aproximare plecând de la un set de date cu zgomot duce la model de predicție slab. Alt motiv e că numărul de neuroni din stratul ascuns ar putea să fie prohibitiv. Ca atare, în practică numărul de noduri din stratul ascuns este mult mai mic decât numărul de date din setul de instruire. Funcția F se transformă într-o funcție de aproximare de forma:

$$F(\mathbf{x})=\sum_{i=1}^{k} w_i\varphi_i\left(\|\mathbf{x}-\hat{\boldsymbol{\mu}}_i\|\right) \tag{7.12}$$

unde dimensiunea vectorului de intrare $\mathbf{x}$ este aceeași ca și cea de până acum, $k < p$ iar punctele $\hat{\boldsymbol{\mu}}_i$ pot proveni dintr–un proces de grupare automată (clustering). Interpretarea ca rețea neurală este dată în figura 7.4. Diferențele față de figura 7.3 sunt că stratul ascuns are k neuroni în loc de p, iar funcțiile

Figura 7.4: Structura unei rețele RBF, folosind mai puține noduri decât în figura 7.3. Centrii $\hat{\boldsymbol{\mu}}_i$, $i = \overline{1,k}$ se obțin printr-un procedeu de clustering.

de activare radială sunt centrate în vectori $\hat{\boldsymbol{\mu}}_k$ în locul datelor din setul de instruire, $\mathbf{x}^{(i)}$.

Pentru determinarea celor k centri $\hat{\boldsymbol{\mu}}_i$, $i = \overline{1,k}$ ale centrilor din stratul ascuns se poate utiliza o metodă oarecare de grupare automată pe baza similarităților (clustering), dar care să producă centroizi[2]. Vom prezenta în cele ce urmează metoda K-means clustering.

7.4 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 doi vectori nesimilari să fie dispuși în clustere diferite.

Considerăm un set de p puncte, $\left\{\mathbf{x}^{(i)}\right\}_{i=\overline{1,p}}$ ce urmează să fie partiționat în k clustere[3]; de regulă, $k \ll p$. 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 $d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ o măsură a similarității[4] – dintre perechile de vectori $\mathbf{x}^{(i)}$ și $\mathbf{x}^{(j)}$. Pentru

[2]Excludem deci algoritmi de clustering precum DBSCAN sau OPTICS.

[3]Cel mai frecvent, valoarea lui k este furnizată de utilizator.

[4]Ne putem folosi aici de orice funcție care măsoară distanța dintre doi vectori; dacă distanța este mică, atunci similaritatea dintre vectori este mare – și invers.

clustering se cere minimizarea funcției:

$$J(C) = \frac{1}{2}\sum_{l=1}^{k}\sum_{i:C(i)=l}\sum_{j:C(j)=l} d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \tag{7.13}$$

unde C este partiționarea dată de cele k clustere:

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

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

$$d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|^2 \tag{7.14}$$

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

Modificăm forma funcției de eroare J astfel încât să se ia în considerare pătratul distanțelor dintre vectorii $\mathbf{x}^{(i)}$ și centroizii $\hat{\boldsymbol{\mu}}_l$ ai clusterelor de care aparțin:

$$J(C) = \frac{1}{2}\sum_{l=1}^{k}\sum_{i:C(i)=l}\left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_l\right\|^2 \tag{7.15}$$

Cum se poziționează centroizii? Algoritmul K-means determină niște valori pentru $\hat{\boldsymbol{\mu}}_l$ 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 $\hat{\boldsymbol{\mu}}_l^{(1)}$, $l = \overline{1,k}$, inițializându–i cu valori fie din setul de instruire, sau setate la întâmplare cu valori din spațiul de intrare, sau conform algoritmului K–means++, a se vedea secțiunea 7.5. Avem apoi o succesiune de iterații cu pașii:

- *Pasul de asignare:* Calculează pentru orice l, $l = 1 \ldots k$:

  $$S_l^{(t)} = \left\{\mathbf{x}^{(i)} : \left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_l^{(t)}\right\|^2 \leq \left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_j^{(t)}\right\|^2, j = \overline{1,k}, j \neq l, i = \overline{1,p}\right\}$$

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

- *Modificarea centroizilor:*

  $$\hat{\boldsymbol{\mu}}_l^{(t+1)} = \frac{1}{\left|S_l^{(t)}\right|} \cdot \sum_{\mathbf{x}^{(i)} \in S_l^{(t)}} \mathbf{x}^{(i)}, \text{ pentru } l = \overline{1,k}$$

  unde $\left|S_l^{(t)}\right|$ este numărul de elemente ale mulțimii $S_l^{(t)}$.

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

În general, algoritmul nu 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 $\hat{\boldsymbol{\mu}}_l^{(1)}$, $l = \overline{1,k}$. Situația (centroizii) pentru care $J(C)$ are valoarea cea mai mică în aceste încercări este reținută.

7.5 Algoritmul de inițializare K-means++

Se poate consideră că inițializarea centroizilor inițiali $\hat{\boldsymbol{\mu}}_l^{(1)}$, $l = \overline{1,k}$ 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 7.5. 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 $J(C) = \frac{1}{2} \cdot 4 \left(\frac{L}{2}\right)^2 = \frac{L^2}{2}$ (punctele $\mathbf{x}^{(1)}$ și $\mathbf{x}^{(3)}$ aparțin clusterului de centroid $\hat{\boldsymbol{\mu}}_1$, iar celelalte două celui de al doilea cluster). Dacă alegerea punctelor se face ca în figura 7.6, atunci se obține valoarea $J(C) = \frac{l^2}{2}$ – 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 7.5: Caz nefavorabil pentru K-means la alegerea centroizilor 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 [9]:

1. Alege primul centroid aleator din setul de antrenare;

2. Pentru fiecare punct ce nu a fost încă ales drept centroid $\mathbf{x}^{(i)}$ calculează $D\left(\mathbf{x}^{(i)}\right)$ $(1 \leq i \leq p)$, distanța de la el până la cel mai apropiat din centroizii determinați până la pasul curent;

Figura 7.6: Alegerea optimă a centroizilor inițiali

3. Alege aleator un nou punct din setul de antrenare, folosind o probabilitate de alegere o funcție crescătoare cu distanța dată de vectorul de distanțe $\mathbf{D}$;

4. Repetă pașii 2 și 3 până când s–au ales toți cei 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. Motivațiile teoretice pentru K-means++ se găsesc în [9].

7.6 Determinarea ponderilor pentru RBF

Determinarea ponderilor legăturilor dintre stratul ascuns și cel de ieșire este ultimul 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.

7.7 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 $\hat{\boldsymbol{\mu}}_j, j = \overline{1,k}$. 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:

$$\sigma = \frac{d_{max}}{\sqrt{2 \cdot k}} \tag{7.16}$$

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 sau clasificare, definind o funcție de eroare și minimizând–o printr-un procedeu adecvat (gradient descent, metoda pseudoinversei etc.)

Capitolul 8

Rețele neurale convoluționale

Rețelele neurale convoluționale (eng. convolutional neural networks, CNNs) sunt modele instruibile care s–au dovedit eficiente în învățarea de date perceptuale — imagini și secvențe de imagini, sunete, serii de timp etc. Domeniile de utilizare ale rețelelor neurale de convoluție includ clasificare, detectare de obiecte, segmentare de imagini – figurile 8.1–8.3.

În problemele tratate de modelel de până acum, intrarea era dată de trăsături selectate sau create de un expert în domeniu. De exemplu, pentru predicția prețului de vânzare al unui apartament se consideră relevante anumite trăsături (suprafața, numărul de camere, numărul de băi, etajul, locația etc.); pentru alte probleme se pot folosi metode de feature engineering care să producă acele trăsături care sunt folosite mai departe într-un proces de învățare (de exemplu, pentru un medic e mai important indicele de masă corporală decât greutatea și înălțimea separate; la procesarea de sunet se face trecerea în spectrul de frecvețe prin transformata Fourier; etc.). *În deep learning formarea de trăsături este efectuată direct de rețeaua neurală* care face și restul muncii (clasificare, segmentare etc.). Efectivitatea acestei abordări a fost clar demonstrată în 2012, când o rețea convoluțională a câștigat competiția ImageNet Large Scale Visual Recognition Challenge cu o diferență mai mare de 10% față de locul următor. Succesul a fost dat și de utilizarea de hardware eficient pentru instruirea rețelei.

Datele abordate de rețelele neurale convoluționale sunt organizate ca tablouri uni- sau multi-dimensionale; spre exemplu, o imagine în tonuri de gri este o matrice de pixeli (0 și 255 codifică negru, respectiv alb) – figura 8.4 – iar o imagine color este un tablou tridimensional de date – figura 8.5. De asemenea, ponderile instruibile sunt structurate ca tablouri. Tablourile se mai numesc și tensori; tensorii de date sunt populați cu valori provenite din măsurătorile obiectelor sau fenomenelor (de exemplu, intensitatea luminoasă a unui pixel). Un scalar este un tensor de dimensiune 0, un vector este un tensor de dimensiune 1, o matrice este un tensor de dimensiune 2, o imagine RGB ca în figura 8.5 este un tensor de dimensiune 3, o secvență video color

Figura 8.1: Clasificare de imagini cu rețea neurală de convoluție

Figura 8.2: Detectare de obiecte cu rețea neurală de convoluție

Imagine de intrare | Segmentare semantică | Segmentare de instanțe

Figura 8.3: Segmentare semantică și segmentare de instanțe

e un tensor de dimensiune 4 etc.

Figura 8.4: Exemplu de cifră 5 din setul de date MNIST. Porțiunea de imagine încadrată este văzută ca matrice de de valori întregi

Figura 8.5: Imagine color separată pe canale roșu, verde, albastru; fiecare pixel este o colecție de 3 valori, pentru canalele de culoare roșu, verde, respectiv albastru

Pe lângă înmulțirile matriciale prezente în rețelele de tip perceptron multistrat, aceste rețele folosesc și operații de convoluție. Mai sunt utilizate straturi de pooling și cu conexiuni complete (eng. fully connected), acestea din urmă fiind deja cunoscute din modelele de regresie liniară și logistică și din rețelele de tip perceptron multistrat.

8.1 Convoluție

Operația de convoluție este des întâlnită în procesarea de semnale audio sau de imagini. În cazul procesării de imagini, unde intrarea este văzută ca o (stivă de) matrice de valori, prin convoluție discretă se generează pixeli noi: pentru fiecare pixel de intrare se consideră o vecinătate convenabil aleasă, peste care se calculează o medie ponderată; ponderile sunt preluate dintr–o matrice de valori, ce formează nucleul (eng. kernel) de convoluție. Mai clar,

un pixel $y[m,n]$ rezultat prin aplicarea convoluției cu un kernel k peste o matrice de intrare x are valoarea:

$$y[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x[i,j] \cdot k[m-i, n-j] \tag{8.1}$$

În formula de mai sus, centrul matricei kernelului k are coordonatele $(0,0)$, deci indicii pentru matricea k pot fi pozitivi și negativi. În practică, indicii de sumare din (8.1) au valori finite: kernelul este o matrice cu dimensiuni finite, deci i și j sunt mărginite.

8.1.1 Exemplificare de convoluție

Un kernel bidimensional este o matrice de valori (fixe sau determinate prin instruire); este deci este caracterizat de numărul de linii și coloane (forma sa), se deplasează cu un anumit pas (eng. stride); intrarea peste care se aplică poate să fie sau nu bordată (eng. padding). Pasul dă numărul de pixeli cu care kernelul se mută față de aplicarea precedentă. Bordarea arată câte rânduri de pixeli sunt adăugați la marginea matricei de intrare, pentru a produce o dimensiune convenabilă la ieșire. În figurile 8.7—8.10 sunt date formulele de calcul pentru convoluția unei matrice de intrare de 4×4 cu un kernel de dimensiune 3×3, kernelul este deplasat la fiecare pas cu o poziție (stride de 1), intrarea este fără bordare (padding 0). Dimensiunea ieșirii este 2×2, fiind dată în funcție de dimensiunea intrării, mărimea kernelului, padding și stride.

Tehnic, formulele din figurile 8.7—8.10 calculează corelarea încrucișată (eng. cross–correlation), sau convoluție cu kernel răsturnat (eng. flipped kernel). Umărind formula 8.1, valoarea de convoluție din figura 8.7 ar fi de fapt $(a_{11}w_{33} + a_{12}w_{32} + a_{13}w_{31}) + (a_{21}w_{23} + a_{22}w_{22} + a_{23}w_{21}) + (a_{31}w_{13} + a_{32}w_{12} + a_{33}w_{11})$ în loc de $(a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13}) + (a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23}) + (a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33})$. Se ajunge de la o formulă la alta făcând inversarea ordinii liniilor și apoi a ordinii coloanelor din matricea kernel. În cele ce urmează vom folosi termenul de convoluție, chiar dacă formula de calcul este cu adică corelație încrucișată.

Un exemplu numeric, preluat din excelenta sinteză a lui Dumoulin și Visin [1] este dat în figura 8.11.

În funcție de configurația stratului de convoluție, la sumă se mai poate adăuga un termen liber – coeficient de bias, b – deci de exemplu prima convoluție (figura 8.7) va produce valoarea: $(a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13}) + (a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23}) + (a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}) + b$. În figurile 8.7—8.11 nu s–a folosit coeficient de bias.

Se observă că același kernel se "plimbă" peste matricea de intrare. Kernelul se comportă ca o lupă care scanează toată imaginea, încercând să găsească o trăsătură anume undeva în intrare, oriunde ar fi ea situată. Trăsătura

Intrare

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

?	?
?	?

Figura 8.6: O intrare de 4×4 cu o convoluție de 3×3, stride 1 si padding 0 (fără bordare) produce o ieșire de 2×2. Cele 4 valori de ieșire se calculează prin aplicarea repetată a kernelului de convoluție pe fiecare sub–matrice de 3×3 din matricea de intrare.

Intrare

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	?
?	?

Figura 8.7: Prima convoluție (cu kernel răsturnat), aplicată peste intrările marcate cu linie punctată

Intrare

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{12}w_{11} + a_{13}w_{12} + a_{14}w_{13} +$ $a_{22}w_{21} + a_{23}w_{22} + a_{24}w_{23} +$ $a_{32}w_{31} + a_{33}w_{32} + a_{34}w_{33}$
?	?

Figura 8.8: A doua convoluție

Intrare

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{12}w_{11} + a_{13}w_{12} + a_{14}w_{13} +$ $a_{22}w_{21} + a_{23}w_{22} + a_{24}w_{23} +$ $a_{32}w_{31} + a_{33}w_{32} + a_{34}w_{33}$
$a_{21}w_{11} + a_{22}w_{12} + a_{23}w_{13} +$ $a_{31}w_{21} + a_{32}w_{22} + a_{33}w_{23} +$ $a_{41}w_{31} + a_{42}w_{32} + a_{43}w_{33}$	?

Figura 8.9: A treia convoluție

Intrare

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{12}w_{11} + a_{13}w_{12} + a_{14}w_{13} +$ $a_{22}w_{21} + a_{23}w_{22} + a_{24}w_{23} +$ $a_{32}w_{31} + a_{33}w_{32} + a_{34}w_{33}$
$a_{21}w_{11} + a_{22}w_{12} + a_{23}w_{13} +$ $a_{31}w_{21} + a_{32}w_{22} + a_{33}w_{23} +$ $a_{41}w_{31} + a_{42}w_{32} + a_{43}w_{33}$	$a_{22}w_{11} + a_{23}w_{12} + a_{24}w_{13} +$ $a_{32}w_{21} + a_{33}w_{22} + a_{34}w_{23} +$ $a_{42}w_{31} + a_{43}w_{32} + a_{44}w_{33}$

Figura 8.10: Ultima convoluție

Figura 8.11: Exemplu numeric pentru convoluție de 3×3 peste o intrare de 5×5, fără bordare și cu pas 1. Preluare din [1]

căutată este dependentă de coeficienții numerici din kernel. Plimbarea kernelului se face peste toate datele din tensorul (mai sus: matricea de intrare), eventual considerând și valorile date de bordare (padding).

Un exemplu numeric concret este arătat mai jos pentru detectarea de muchii verticale. Matricea de kernel este anume aleasă pentru a depista tranziția de la o zonă mai luminată la una mai întunecată. Dacă valorile din matricea de ieșire se compară cu o valoare de prag, atunci va rezulta o muchie verticală.

$$\underbrace{\begin{pmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{pmatrix}}_{\text{Intrare}} * \underbrace{\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}}_{\text{Kernel}} = \underbrace{\begin{pmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{pmatrix}}_{\text{Ieșire}} \tag{8.2}$$

O diferență între kernelele populare în procesarea de imagini și cele din rețelele de convoluție este că acestea din urmă își ajustează ponderile în etapa de învățare; în particular, prin învățare se poate ajunge la coeficienții din kernelele din procesarea de imagini. Scopul este ca învățarea automată să determine kernelele potrivite pentru sarcina vizată, de exemplu pentru clasificarea de imagini.

Comentăm efectul folosirii aceluiași kernel peste toată intrarea considerând numărul de parametri: pentru intrarea de dimensiune 4×4 și kernelul de 3×3 exemplificate în figurile 8.6–8.10, avem $3 \times 3 = 9$ coeficienți instruibili – cei din kernel – la care se mai adaugă eventual ponderea de bias, deci 10 ponderi instruibile. Dacă am folosi precum în rețelele de tip MLP, câte o legătură (și deci câte o pondere) de la fiecare din cele 16 valori din matricea de intrare la fiecare valoare din matricea de ieșire (4 valori), la care adăugăm și ponderile de bias, obținem $16 \times 4 + 4 = 68$ de ponderi instruibile, mai multe decât cele 10 de la convoluție. Pe lângă reducerea numărului de ponderi – și deci: a memoriei necesare modelului, a numărului de calcule necesare – avem și o scădere a probabilității de overfitting, deoarece sunt mai puțini parametri. Un kernel produce deci *interacțiuni rare* (eng. sparse interactions): el angrenează un număr limitat de intrări în calcul, față de straturile complet conectate din MLP, unde valoarea de intrare într–un neuron e calculată cu activările tuturor nodurilor din stratul anterior.

De asemenea, un kernel promovează *partajarea ponderilor* (eng. parameter sharing): exact aceleași ponderi sunt folosite când kernelul este deplasat peste intrare. Prin contrast, ponderea unei legături dintre doi neuroni dintr-o rețea MLP nu este reutilizată, fiind specifică acelei perechi de neuroni.

Mai sus s–a considerat mereu că nu se face bordarea matricei de intrare, adică valoarea de padding este $p = 0$; aceasta se mai numește si o *convoluție validă*: kernelul se plimbă peste matricea de intrare, fără a depăși marginile ei. Pentru o intrare de dimensiuni $w \times h$ și un kernel de dimensiune $k \times l$ și pas (stride) 1, ieșirea rezultată pentru convoluție validă va avea dimensiunea:

$$w' \times h' = (w - k + 1) \times (h - l + 1) \tag{8.3}$$

Se observă că aplicarea unui kernel, în aceste condiții, duce la reducerea dimensiunilor matricei de ieșire. Aplicarea în mod repetat duce la micșorarea succesivă a intrării. Uneori, acest lucru nu e de dorit.

8.1.2 Bordarea intrării

Dacă se ia coeficientul de padding $p > 0$, atunci matricea de intrare se va borda cu p linii și coloane setate cu o valoare constantă, de regulă 0, de fiecare parte a intrării[1]; peste această matrice de dimensiune $(w + 2 \cdot p) \times (h + 2 \cdot p)$ se va aplica kernelul de convoluție de $k \times l$ cu stride 1, rezultând o ieșire de dimensiune

$$w' \times h' = (w + 2 \cdot p - k + 1) \times (h + 2 \cdot p - l + 1) \tag{8.4}$$

Putem justifica bordarea astfel:

[1] Se poate face o bordare cu număr diferit de coloane la stânga și la dreapta, respectiv număr aparte de linii deasupra și dedesubtul intrării.

1. În lipsa bordării, valorile din colțurile matricelor sunt folosite fiecare o singură dată în producerea unei valori rezultat; valorile ce nu sunt la marginea matricei sunt folosite de mai multe ori, deoarece kernelul se plimbă peste matricea de intrare; pentru pasul de deplasare 1, o valoare de intrare care nu e pe laturile matricei de intrare este acoperită de mai multe ori de kernel. Acest efect se arată în figura 8.12, pentru un kernel de dimensiune 3×3: în timp ce valoarea a_{22} este folosită de patru ori în calcularea diverselor valori din matricea de ieșire, valoarea a_{11} este folosită o singură dată, doar în prima convoluție figurată. Similar, valorile de pe marginile matricei sunt de mai puține ori folosite în aplicarea de kernel decât valorile care nu sunt pe margini.

Figura 8.12: O valoare centrală a intrării va fi folosită de mai multe ori în aplicarea kernelului. În exemplul arătat, valoarea a_{22} este folosită de 4 ori la aplicarea kernelului. În schimb, o valoare dintr-un colț – de exemplu a_{11} – este folosită o singură dată.

2. Vrem să evităm micșorarea dimensiunilor matricei de ieșire după convoluție; putem cere ca după aplicarea convoluției, dimensiunile matricei de ieșire să fie aceleași ca la matricea de intrare. Spre exemplu, pentru un kernel de $k \times l = 3 \times 3$, aplicarea lui peste o matrice de dimensiune $w \times h$ ar duce la obținerea unei ieșiri de forma $(w - k + 1) \times (h - l + 1) = (w - 2) \times (h - 2)$. Dacă bordăm pe fiecare latură cu un rând de valori 0 ($p = 1$), atunci obținem dimensiunile $(w + 2 \cdot p - k + 1) \times (h + 2 \cdot p - l + 1) = w \times h$.

Bordarea de așa manieră ca dimensiunea intrării să coincidă cu cea a intrării se numește *convoluție asemenea* (eng. same convolution). Exemplu: plecăm de la matricea de intrare

$$I = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \tag{8.5}$$

și kernelul de 3×3

$$K = \begin{bmatrix} -4 & -3 & -2 \\ -1 & 0 & 1 \\ 2 & 3 & 4 \end{bmatrix} \tag{8.6}$$

Convoluția validă dintre I și K produce ieșirea

$$I * K = \begin{bmatrix} 78 & 78 \\ 78 & 78 \end{bmatrix} \tag{8.7}$$

Pentru convoluție asemenea facem bordarea cu câte o linie și coloană pline cu 0, adică $p = 1$ și obținem intrarea

$$I' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 7 & 0 \\ 0 & 8 & 9 & 10 & 11 & 0 \\ 0 & 12 & 13 & 14 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.8}$$

Făcând convoluția lui I' cu K obținem:

$$I' * K = \begin{bmatrix} 33 & 49 & 58 & 31 \\ 63 & 78 & 78 & 30 \\ 75 & 78 & 78 & 18 \\ -29 & -77 & -86 & -87 \end{bmatrix} \tag{8.9}$$

și observăm că ieșirea are aceeași dimensiune ca și intrarea inițială I.

8.1.3 Pasul kernelului

În cele de mai sus s–a considerat că kernelul de convoluție se mută cu câte o poziție mai la dreapta sau mai jos. Se poate seta ca pasul de deplasare[2] să fie $s > 1$. Figura 8.13 arată o convoluție validă cu pas 2. Se observă că dimensiunea ieșirii este influențată de pasul de deplasare: deoarece se face salt peste valori și kernelul se aplică de mai puține ori decât anterior, matricea de ieșire are dimensiuni mai mici față de cazul în care pasul este 1.

[2]Deplasarea se poate face cu pas diferit pe orizontală față de verticală. În cele ce urmează vom folosi pas identic pe cele două direcții.

Intrare

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{54}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	?
?	?

(a) Prima convoluție

Intrare

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{54}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{13}w_{11} + a_{14}w_{12} + a_{15}w_{13} +$ $a_{23}w_{21} + a_{24}w_{22} + a_{25}w_{23} +$ $a_{33}w_{31} + a_{34}w_{32} + a_{35}w_{33}$
?	?

(b) A doua convoluție. Se observă că s–a mutat kernelul cu două poziții mai la dreapta

Intrare

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{54}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{13}w_{11} + a_{14}w_{12} + a_{15}w_{13} +$ $a_{23}w_{21} + a_{24}w_{22} + a_{25}w_{23} +$ $a_{33}w_{31} + a_{34}w_{32} + a_{35}w_{33}$
$a_{31}w_{11} + a_{32}w_{12} + a_{33}w_{13} +$ $a_{41}w_{21} + a_{42}w_{22} + a_{43}w_{23} +$ $a_{51}w_{31} + a_{52}w_{32} + a_{53}w_{33}$	?

(c) A treia convoluție – s–a sarit de la prima la a treia linie

Intrare

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{54}

$*$

Kernel

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

Ieșire

$a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} +$ $a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} +$ $a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33}$	$a_{13}w_{11} + a_{14}w_{12} + a_{15}w_{13} +$ $a_{23}w_{21} + a_{24}w_{22} + a_{25}w_{23} +$ $a_{33}w_{31} + a_{34}w_{32} + a_{35}w_{33}$
$a_{31}w_{11} + a_{32}w_{12} + a_{33}w_{13} +$ $a_{41}w_{21} + a_{42}w_{22} + a_{43}w_{23} +$ $a_{51}w_{31} + a_{52}w_{32} + a_{53}w_{33}$	$a_{33}w_{11} + a_{34}w_{12} + a_{35}w_{13} +$ $a_{43}w_{21} + a_{44}w_{22} + a_{45}w_{23} +$ $a_{53}w_{31} + a_{54}w_{32} + a_{55}w_{33}$

(d) Ultima convoluție

Figura 8.13: Convoluție validă cu pas 2

Se obține astfel un efect de subeșantionare; un proces similar are loc la folosirea straturilor de pooling (secțiunea 8.3), doar că la convoluții cu stride > 1 subeșantionarea este învățată, și nu o strategie fixă. Subeșantionarea este utilă deoarece se reduce din redundanța semnalului – de exemplu, într–o imagine, pixelii alăturați sunt foarte similari și se referă frecvent la același obiect.

Dăm mai jos formula de calcul pentru dimensiunea ieșirii, în funcție de:

- dimensiunea intrării $w \times h$
- dimensiunea kernelului, $k \times l$
- padding, $p \geq 0$
- stride, $s \geq 1$

Ieșirea va fi de dimensiune:

$$w' \times h' = \left(\left[\frac{w + 2p - k}{s} \right] + 1 \right) \times \left(\left[\frac{h + 2p - l}{s} \right] + 1 \right) \tag{8.10}$$

unde $[x]$ este partea întreagă a lui x. Evident, această formulă generalizează variantele date mai sus.

Un exemplu numeric de convoluție cu pas 2 și padding 1, similar cu 8.11 este dat in figura 8.14 [1].

Figura 8.14: Exemplu numeric pentru convoluție de 3×3 peste o intrare de 5×5, cu pas 2 si padding 1 (convoluție asemenea). Preluare din [1]

8.1.4 Numărul de ponderi și de operații pentru convoluție cu intrare pe un singur canal

Este important să precizăm numărul de ponderi instruibile și operații efectuate pentru convoluțiile discutate până acum. Să presupunem că lucrăm cu un kernel de convoluție de dimensiune $k \times l$. Pentru calculul unei singure valori din matricea de ieșire trebuie să facem $k \times l$ inmulțiri de doi termeni (unul e pondere din kernel, altul e valoare din matricea de intrare); fiecare din cele $k \times l$ valori este acumulată la o variabilă setată inițial la 0 pentru a obține termenul final, obținând deci $k \times l$ înmulțiri și acumulări (eng. multiply–accumulate operations, MAC). Acest lucru se repetă pentru calculul fiecărui termen din matricea de ieșire.

Sintetizând: dacă la o matrice de intrare de forma $w \times h$ aplicăm un kernel de convoluție de dimensiune $k \times l$, cu stride s și padding p, atunci:

- numărul de ponderi instruibile din kernel este:

$$n_{ponderi} = k \times l \tag{8.11}$$

- dimensiunea ieșirii $w' \times h'$ este cea din ecuația 8.10;
- numărul de înmulțiri și acumulări este

$$MAC = (w' \times h') \times (k \times l) \tag{8.12}$$

Dacă considerăm și bias, atunci va fi un unic coeficient de bias pentru tot kernelul, și formulele de mai sus trebuie ajustate corespunzător: $n_{ponderi} = k \times l + 1$, $MAC = (w' \times h') \times (k \times l + 1)$.

8.1.5 Convoluții pentru intrări cu mai multe canale

În exemplificările anterioare s-a considerat mereu intrarea ca matrice, adică un tensor bidimensional. În general, putem avea ca intrare pentru operația de convoluție un tensor cu mai mult de două dimensiuni, deoarece:

1. semnalul de intrare inițial poate fi cu mai multe canale, de exemplu imagine codificată RGB – a se vedea figura 8.5;
2. putem decide ca pe o intrare oarecare să se detecteze mai multe trăsături, rezultând mai multe hărți de trăsături corespunzătoare. Aceste multiple hărți vor produce următoarea intrare, care va fi deci cu mai multe canale.

Un *filtru de convoluție* este o colecție de kernele de convoluție, câte un kernel pentru fiecare canal de intrare. În figura 8.15 este reprezentat un filtru de convoluție aplicat peste o intrare cu $c = 4$ canale, fiecare canal de intrare fiind câte o matrice de 6×6; forma intrării este deci $w \times h \times c = 6 \times 6 \times 4$.

Pentru exemplul dat filtrul are forma $k \times l \times c = 3 \times 3 \times 4$; se folosește aici convoluție validă cu pas 1. Rezultatul aplicării filtrului este o matrice de forma 4×4. Lungimea și lățimea ieșirii sunt calculate conform ecuației 8.10. Pentru întreg filtrul de convoluție se poate adăuga un coeficient de bias. Pentru exemplul din figura 8.15 numărul de parametri instruibil este dat de câte valori sunt în filtru – $3 \cdot 3 \cdot 4 = 36$ – la care se mai adaugă eventual un coeficient de bias.

Figura 8.15: Convoluție validă pentru o intrare cu 4 canale, cu stride 1: intrarea are forma $w \times h \times c$ dată de lățimea și înălțimea intrării (6×6) și numărul de canale de intrare, adică 4; filtrul de convoluție este un tensor tridimensional, cu mărimea fiecărui kernel 3×3, iar a treia dimensiune e numărul de canale de intrare. Rezultatul este un tensor de forma 4×4.

Pentru calculul unei singure valori din matricea de ieșire, acești 36 de coeficienți se înmulțesc cu tot atâtea valori din tensorul de intrare (3×3 valori din matricea din plan apropiat; alte 3×3 valori din matricea din planul următor; la fel până la matricea de intrare din plan îndepărtat), deci 36 de înmulțiri; fiecare din acești termeni este acumulat la o variabilă setată inițial la 0 pentru a obține termenul final, obținând deci 36 de înmulțiri și acumulări (operații MAC). Formal, prima valoare din matricea de iesire o se calculează astfel (tensorul tridimensional a este matrice de intrare, cu $a_{1,1..6,1..6}$ matricea din plan apropiat și $a_{4,1..6,1..6}$ matricea de valori din plan indepărtat; similar pentru ponderile w din tensorul tridimensional al filtrului):

$$
\begin{aligned}
o[1,1] = & \overbrace{a_{111}w_{111} + a_{112}w_{112} + a_{113}w_{113} + \cdots + a_{131}w_{131} + a_{132}w_{132} + a_{133}w_{133}}^{\text{convoluția cu primul kernel din filtru: 9 MACs}} + \\
& + \overbrace{a_{211}w_{211} + a_{212}w_{212} + a_{213}w_{213} + \cdots + a_{231}w_{231} + a_{232}w_{232} + a_{233}w_{233}}^{\text{convoluția cu al doilea kernel din filtru: 9 MACs}} + \\
& + \overbrace{a_{311}w_{311} + a_{312}w_{312} + a_{313}w_{313} + \cdots + a_{331}w_{331} + a_{332}w_{332} + a_{333}w_{333}}^{\text{convoluția cu al treilea kernel din filtru: 9 MACs}} + \\
& + \overbrace{a_{411}w_{411} + a_{412}w_{412} + a_{413}w_{413} + \cdots + a_{431}w_{431} + a_{432}w_{432} + a_{433}w_{433}}^{\text{convoluția cu ultimul kernel din filtru: 9 MACs}}
\end{aligned} \tag{8.13}
$$

Similar se procedează pentru a calcula restul de valori din matricea de ieșire, obținând în total $4 \times 4 \times 36 = 576$ MACs.

Mai general, dacă pornim de la o intrare de forma $w \times h \times n_{in}$ și facem o convoluție cu un singur filtru de forma $k \times l \times n_{in}$, cu stride s și padding p, atunci:

1. numărul de ponderi din filtru este

$$n_{ponderi} = k \times l \times n_{in} \tag{8.14}$$

2. ieșirea este o matrice de dimensiune $w' \times h'$ cu dimensiunile date ca în formula 8.10;

3. numărul total de adunări și acumulări pentru matricea de ieșire este

$$MAC = (w' \times h') \times n_{ponderi} = (w' \times h') \times (k \times l \times n_{in}) \tag{8.15}$$

Dacă se dorește și includere de bias, atunci va fi un unic coeficient de bias pentru tot filtrul și cele două formule de mai sus trebuie actualizate corespunzător: $n_{ponderi} = k \times l \times n_{in} + 1$, $MAC = (w' \times h') \times (k \times l \times n_{in} + 1)$.

Faptul că în exemplul 8.15 ieșirea este un tensor cu două dimensiuni (o matrice) se explică prin aceea că s–a aplicat un singur filtru. Folosirea de n_{out} filtre pentru intrarea din figura 8.15 produce un tensor de forma $4 \times 4 \times n_{out}$. Fiecare filtru trebuie să fie de forma $3 \times 3 \times 4$, ultima valoare (4) fiind numărul de canale de intrare. Cazul convoluției cu două filtre este reprezentat în figura 8.16.

Folosirea mai multor filtre este justificată de faptul că fiecare produce o vedere proprie pornind de la aceeași intrare. De exemplu, un filtru poate fi specializat pentru detectarea de muchii orizontale, altul pentru detectarea de muchii verticale. Combinarea detecțiilor în straturile următoare poate ajuta în sarcina propusă – clasificare, segmentare, detectare de obiecte etc. În final, trăsăturile detectate prin filtrare succesivă sunt date unor straturi complet conectate, care fac clasificarea finală.

Figura 8.16: Convoluție validă pentru un strat de intrare cu 4 canale, 2 filtre, padding 0 și stride 1: intrarea are forma $w \times h \times c = 6 \times 6 \times 4$; fiecare filtru este un tensor $3 \times 3 \times 4$. Rezultatul este un tensor de forma $4 \times 4 \times 2$.

În general, dacă pe o intrare de forma $w \times h \times n_{in}$ – adică: n_{in} canale de intrare, fiecare canal fiind o matrice de dimensiune $w \times h$ – se aplica n_{out} filtre, toate cu padding p și stride s, atunci:

1. fiecare filtru trebuie să traverseze toate cele n_{in} canale de intrare, fiind deci compus din n_{in} kernele;

2. un filtru are forma $k \times l \times n_{in}$ (8.14); numărul total de ponderi instruibile din toate filtrele este

$$n_{ponderi} = k \times l \times n_{in} \times n_{out} \tag{8.16}$$

3. ieșirea va fi un tensor de forma

$$w' \times h' \times n_{out} = \left(\left[\frac{w + 2p - k}{s} \right] + 1 \right) \times \left(\left[\frac{h + 2p - l}{s} \right] + 1 \right) \times n_{out} \tag{8.17}$$

   unde w' și h' sunt calculate ca în ecuația 8.10;

4. numărul de operații de adunare și acumulare este:

$$MAC = (w' \times h' \times n_{out}) \times (k \times l \times n_{in}) \tag{8.18}$$

   formula fiind obținută pornind de la 8.15 și considerând și numărul de filtre – același cu numărul de canale de ieșire – n_{out}.

Dacă se folosește bias, atunci fiecare filtru de bias va avea propriul său coeficient de bias, ca mai sus; se actualizează ușor numărul de ponderi și de operații. De exemplu, numărul de ponderi calculat în (8.16) devine $n_{ponderi} = (k \times l \times n_{in} + 1) \times n_{out}$.

8.2 Funcții de activare

Dacă urmărim calculele din imaginile 8.7–8.10, observăm că operația de convoluție e o operație liniară; a se vedea și [1] secțiunea 4.1 pentru argumentare riguroasă a liniarității convoluției. Ca atare, se impune utilizarea de funcții de activare neliniare, de forma celor folosite la rețelele de tip perceptron multistrat, secțiunea 6.4.2 pagina 81. În arhitecturile deep learning actuale sunt populare funcțiile de activare ReLU, PReLU, ELU. Funcțiile de activare se aplică punctual, adică peste fiecare valoare din rezultatul dat de convoluție.

Tensorii rezultați după aplicarea funcției de activare se numesc *hărți de trăsături* (eng. feature maps).

8.3 Straturi de pooling

Un strat de pooling este un filtru bidimensional care se aplică peste fiecare canal din harta de trăsături. Prin pooling se produc sumarizări locale ale valorilor din feature maps și totodată se reduce dimensiunea acestora (subeșantionare). Suplimentar, deoarece valoarea agregată prin pooling nu depinde de pozițiile pe care se află valorile folosite (în cadrul vecinătății), se obține o invarianță la mici translații.

Un strat de pooling este caracterizat de: dimensiunea lui (lățime, înălțime), pas (stride) și forma de agregare a valorilor; nu se practică bordarea intrării. Filtrele se pot aplica pe regiuni adiacente sau cu suprapuneri. Tensorii rezultați au mărimea conform ecuației 8.10. Deoarece folosesc agregări de valori (maxim, medie), straturile de pooling nu au ponderi instruibile.

8.3.1 Max pooling

Filtrul max pooling calculează valoarea maximă peste fiecare regiune aleasă. Un exemplu este dat în figura 8.17, pentru un filtru de tip max pooling de dimensiune 2×2, cu pasul 2, aplicat pe o intrare cu un singur canal. Dacă sunt mai multe canale, se va aplica operația de pooling pe fiecare canal în parte, în mod independent.

8.3.2 Average pooling

Un filtru de tip average pooling folosește calculul de medie ca formă de agregare. Restul detaliilor de la filtrul max pooling sunt valabile și aici. Un exemplu numeric este dat în figura 8.18, pentru un filtru de tip max pooling de dimensiune 2×2, cu pasul 2, un singur canal de intrare. Dacă intrarea este pe mai multe canale, atunci se face aplică filtrul de average pooling în mod independent, pe fiecare canal.

Figura 8.17: Max pooling aplicat peste un canal al unei hărți de trăsături de dimensiune 4×4

Figura 8.18: Average pooling aplicat peste un canal al unei hărți de trăsături de dimensiune 4×4

Sunt mai puțin folosite decât max pooling, pentru că se consideră că e mai util a se procesa prezența maximală a unei trăsături dintr–o hartă decât valoarea medie a trăsăturilor.

8.3.3 Global average pooling

Un filtru de global average pooling (GAP) reduce fiecare canal din harta de trăsături peste care se aplică la o singură valoare, media de pe acel canal. Este echivalent cu a aplica un filtru de average pooling de dimensiunea intrării. Astfel, se reduce o hartă de trăsături de forma $w \times h \times n_{in}$ la un tensor de forma $1 \times 1 \times n_{in}$, indiferent de valorile lui w și h.

Două exemple sunt date în figura 8.19.

8.4 Straturi complet conectate

Pentru rețelele neurale de convoluție folosite pentru estimare de probabilitate condiționată, clasificare sau regresie se folosesc în finalul rețelei straturi complet conectate (eng. fully connected layers), exact ca într-o rețea neurală perceptron multistrat. Se acționează pe principiul: filtrele de convoluție și straturile de pooling fac filtrări succesive, plecând de la intrarea originară (e.g. imagine RGB) și obținând trăsături din ce în ce mai rafinate; acestea

(a) Filtru de global average pooling pe o hartă de trăsături cu un canal de forma 4×4

(b) Filtru de global average pooling aplicat pe o hartă de trăsături de forma $4 \times 4 \times 3$

Figura 8.19: Global average pooling pe un canal, respectiv pe 3 canale

devin intrare pentru o arhitectură de tip MLP – parte a rețelei convoluționale – și aici se face efectiv clasificarea, regresia sau estimarea de probabilitate condiționată.

Înainte de primul strat complet conectat se face o aplatizare, adică se dispun toți neuronii într–un șir – a se vedea figura 8.20. De la stratul aplatizat urmează unul sau mai multe straturi complet conectate, care efectuează operație de înmulțire matricială, urmate imediat de funcții de activare neliniare. De exemplu, pentru problemă de clasificare sau de estimare de probabilitate condiționată se folosește după ultimul strat complet conectat funcția softmax, definită în secțiunea 3.3.2 pagina 43.

Numărul de neuroni din stratul de ieșire este dat de numărul de ieșiri cerute de problemă – de exemplu, pentru o problemă de clasificare este numărul de clase, pentru fiecare producându–se o valoare de probabilitate condiționată. Numărul de neuroni din celelalte straturi complet conectate ține de decizie arhitecturală a rețelei neurale convoluționale din care fac parte straturile complet conectate.

În figura 8.20 este arătată o succesiune de straturi complet conectate, plecând de la o hartă de trăsături (sau rezultatul aplicării unui filtru de pooling peste o hartă de trăsături) ce se aplatizează.

8.5 Funcții de cost, regularizare

Pentru modificare ponderilor se folosește o funcție de cost, care măsoară discrepanța între ieșire produsă de rețea și valoarea cunoscută pentru intrarea curentă. Funcția de cost trebuie să fie diferențiabilă, pentru a permite calcul de gradienți ce vor fi folosiți pentru modificarea ponderilor rețelei. Se pot folosi funcțiile de cost din secțiunea 6.6, pagina 87. În literatura de deep learning există și alte propuneri, influențate de problema ce trebuie rezolvată.

Se recomandă utilizarea unei forme de regularizare precum penalizare L_1, L_2 sau utilizarea unor forme aparte de regularizare, dezvoltate specific

Figura 8.20: Straturi complet conectate la final de rețea neurală de convoluție

pentru rețele de tip deep learning, precum Dropout[3], Stochastic Depth[4].

Se utilizează algoritmi de modificare a parametrilor rețelei folosind gradienții, de exemplu gradient descent, ADAM[5], Adagrad[6], RMSProp[7].

8.6 Rețelele neurale de convoluție: soluție end-to-end

O rețea neurală de convoluție are o arhitectură schițată în figura 8.21.

În succesiunea convoluție - neliniaritate - pooling pleacă de la o intrare (imagine RGB, de exemplu) care este bogată în detalii, dar cu semantică puțină. Pe măsură ce se extrag succesiv trăsături, se pierd detalii (considerate redundante) și se câștigă în plan semantic. Figura 8.22[8] arată acest lucru.

[3]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1 (January 2014), 1929–1958.

[4]Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q. (2016). Deep Networks with Stochastic Depth. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Scienc, vol 9908. Springer, Cham. https://doi.org/10.1007/978-3-319-46493-0_39

[5]Kingma, D. P., Ba, J. (2014). Adam: A Method for Stochastic Optimization (cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015)

[6]Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[7]Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report.

[8]Preluare din Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. 2011. Unsupervised learning of hierarchical representations with convolutional deep belief networks. Commun. ACM 54, 10 (October 2011), 95–103. https://doi.org/10.1145/2001269.2001295

Figura 8.21: Arhitectura generică a unei rețele neurale convoluționale folosită pentru clasificare de imagini. Secvența convoluție + neliniaritate + pooling se poate repeta. La final se află straturile complet conectate. Aici, ultimul strat produce probabilități condiționate

Figura 8.22: Trăsături învățate de o rețea neurală profundă. Primele trăsături de intrare sunt chiar pixelii, detaliați, dar nu de mare ajutor în clasificarea dorită. Pe măsură ce se avansează cu filtrarea, se obțin trăsături din ce în ce mai rafinate – segmente cu diferite orientări, elemente de trăsături faciale, părți mai mari din față etc. Fiecare trăsătură poate fi văzută ca un filtru, care caută în imagine o anumită trăsătura – de exemplu, un nas. Dacă trăsătura este găsită, valoarea de activare corespunzătoare este mare. În straturile următoare se combină trăsăturile detectate. Clasificarea din final este sarcină rezolvată de straturile complet conectate.

8.7 Exemple de rețele neurale de convoluție

Punând cap la cap elementele arhitecturale (convoluție, funcție de activare, strat de pooling, straturi complet conectate), rezultă că, schematic, o rețea neurală de convoluție ar arăta precum în figura 8.23. Blocul format din succesiunea convoluție $\rightarrow$ neliniaritate $\rightarrow$ pooling se poate repeta de mai multe ori, înainte de a trece la straturile complet conectate.

Figura 8.23: Schiță a unei rețele de convoluție folosită pentru clasificare

Figura 8.24: Arhitectura rețelei convoluționale LeNet5

În cele ce urmează prezentăm două arhitecturi clasice de rețele de convoluție.

8.7.1 LeNet-5

Rețeaua LeNet-5 a fost introdusă de Yann LeCun[9] pentru recunoașterea de cifre scrise de mână. Rețeaua constă din 3 straturi convoluționale, 2 straturi de pooling și 2 straturi complet conectate. Arhitectura rețelei este desenată în figura 8.24 și detaliată în tabelul 8.1.

Strat	Nr. filtre /neuroni	Dim. kernel	Pas	Dim. hartă de trăsături	Activare
Intrare	-	-	-	$32 \times 32 \times 1$	-
Conv 1	6	5×5	1	$28 \times 28 \times 6$	tanh
Pool 2	-	2×2	2	$14 \times 14 \times 6$	-
Conv 3	16	5×5	1	$10 \times 10 \times 16$	tanh
Pool 4	-	2×2	2	$5 \times 5 \times 16$	-
Conv 5	120	5×5	1	$1 \times 1 \times 120$	tanh
Fully connected 6	-	-	-	84	tanh
Fully connected 7	-	-	-	10	softmax

Tabelul 8.1: Detalii arhitecturale pentru rețeaua LeNet5.

8.7.2 VGG16

Rețeaua VGG16 este o rețea de tip convoluțional introdusă in articolul "Very Deep Convolutional Networks for Large-Scale Image Recognition"[10]. Modelele VGG au plecat de la idea de a folosi succesiuni de convoluții cu kernel de 3×3 în locul kernelelor de convoluție de dimensiuni mai mari – 5×5, 7×7 – intens folosite în modelele anterioare; astfel, numărul de ponderi instruibile scade. Rețelele VGG au fost folosite cu succes la clasificarea de imagini (fiind antrenate pe setul de date ImageNet, care are 1000 de clase), detectare de obiecte și calcul de reprezentări numerice (eng. embeddings). VGG16 este una din variantele propuse de autori și cu cele mai bune rezultatele empirice.

O reprezentare grafică a arhitecturii este în figura 8.25. Intrarea este dată de imagini de dimensiune 224×224 pe 3 canale (RGB); se face o preprocesare a imaginilor astfel încât fiecare canal să fie cu valoarea medie

[9] Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition", Proceedings of the IEEE. 86 (11): 2278–2324. doi:10.1109/5.726791, link

[10] K. Simonyan i A. Zisserman, "Very deep convolutional networks for large-scale image recognition", in ICLR, 2015, link

0 și deviația standard 1 (standardizare). Imaginea este trecută printr–o secvență de straturi de convoluție cu kernel de dimensiune 3×3. Fiecare strat are 64 de filtre de convoluție. Convoluția e de tip asemenea (bordare cu un rând de valori 0) și pas 1. Ultima hartă de trăsături este trecută printr–un filtru de tip max pooling de dimensiune 2×2 cu pas 2. La finalul acestui bloc se ajunge la tensori de forma $112 \times 112 \times 64$.

Urmează un bloc similar, cu 128 de filtre de convoluție, urmate de max pooling cu aceiași parametri ca mai sus. Rezultă un tensor de forma $56 \times 56 \times 128$. Al treilea bloc are trei straturi convoluționale de câte 256 de filtre (convoluție asemenea, bordare și pas de 1) urmate de un max pooling ca mai sus. Rezultă un tensor de forma $28 \times 28 \times 256$. Urmează două blocuri de câte 3 straturi de convoluție, fiecare cu 512 filtre + max pooling. Se ajunge aici la tensor de forma $7 \times 7 \times 512$. În final se pun 3 straturi complet conectate, două cu 4096 de neuroni și ultimul cu 1000 de neuroni (numărul de clase).

Ultimul strat folosește funcție de activare softmax. Celelalte straturi complet conectate sau de convoluție sunt urmate de funcție de activare ReLU.

Figura 8.25: Arhitectura rețelei convoluționale VGG16

Capitolul 9

Calcul evoluționist

Calculul evoluționist este inspirat din teoria evoluției dezvoltate de către Charles Darwin și de genetică – știința eredității, având ca părinte pe Gregor Mendel. Au în comun faptul că folosesc populații de elemente care sunt folosite pentru căutarea soluției unei probleme, spre deosebire de alte abordări care încearcă îmbunătățirea printr-un proces iterativ a unei singure valori.

9.1 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 domeniului algoritmilor genetici, dar la ora actuală există hibridizări ale acestor 4 arii.

Cel care este creditat ca fiind pionierul domeniului algoritmilor 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 algoritmii genetici 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 optimizarea de suprafețe aflate în contact cu un fluid. Mutațiile reprezentau perturbări ale unor suprafețe, efectuând o căutare locală. 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 individ 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 înseamnă că rulări diferite pot duce la rezultate diferite. Totodată, algoritmii 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 algoritmii genetici.

9.2 Algoritmi genetici

Rolul mediului ca factor modelator în teoria evoluționistă este preluat în algoritmii genetici de către o funcție scop (sau funcție obiectiv). Vom detalia algoritmul pentru maximizarea unei funcții $f : [a, b] \to \mathbb{R}_+^*$.

Constrângerea ca funcția f să fie strict pozitivă poate fi asigurată prin adunarea unei cantități convenabile la funcția inițială, dacă are și porțiuni negative, sau considerarea funcției $g(x) = \max(\varepsilon, f(x))$, unde ε este o constantă strict pozitivă și mică; sau aplicăm o transformare inversabilă cu codomeniu strict pozitiv de forma $\exp f(x)$ etc. Alegerea de a maximiza funcția obiectiv este convenabilă pentru descrierea care urmează. Dacă se dorește minimizare de funcție, se poate ține cont de relația:

$$\min_{x \in \mathscr{D}} f(x) = -\max_{x \in \mathscr{D}} [-f(x)] \tag{9.1}$$

Indivizii care alcătuiesc populația se numesc *cromozomi* și sunt alcătuiți din *gene*, reprezentați ca biți.

Figura 9.1: Reprezentarea unui cromozom în algoritmii genetici. Cromzomomul se transformă în final într-o valoare din intervalul $[a, b]$

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 valoarea funcției f ce se vrea maximizată) 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, iar k e un 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 $c_1, \ldots, c_n$.

Fiecare cromozom c – șir de k biți – va produce un număr $x(c)$ din intervalul $[a, b]$ (figura 9.1), astfel: dacă valoarea în baza 10 a cromozomului este $v(c)$ (unde, evident, avem $v(c) \in \{0, \ldots, 2^k - 1\}$) atunci valoarea asociată din intervalul $[a, b]$ este:

$$x(c) = a + v(c) \cdot \frac{b-a}{2^k - 1} \in \left\{ a, a + \frac{b-a}{2^k - 1}, a + 2 \cdot \frac{b-a}{2^k - 1}, \ldots, b \right\} \quad (9.2)$$

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 populatie se calculează funcția obiectiv $y_i = f(x(c_i))$, $1 \leq i \leq n$. Apoi se însumează valorile funcțiilor obiectiv obținute pentru fiecare cromozom în parte:

$$S = \sum_{i=1}^{n} y_i \tag{9.3}$$

Pentru fiecare din cei n cromozomi se calculează probabilitatea de selecție:

$$p_i = \frac{y_i}{S}, 1 \leq i \leq n \tag{9.4}$$

Se observă că $0 < p_i < 1,\ 1 \leq i \leq n$ și $\sum_{i=1}^{n} p_i = 1$.

Pentru fiecare cromozom se calculează probabilitatea cumulativă de selecție:

$$q_j = \sum_{i=1}^{j} p_i, 1 \leq j \leq n \tag{9.5}$$

Remarcăm că se obține $0 < p_1 = q_1 < q_2 < \cdots < q_n = 1$. 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} – adică lungimea intervalului $(q_{i-1}, q_i]$ – este mai mare[1].

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 in 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 incrucișare. 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 încrucișează astfel: primul selectat cu al doilea selectat, al 3-lea cu al 4-lea etc. Pentru fiecare pereche, încrucișarea decurge astfel:

[1] Putem considera conveabil că $q_0 = 0$.

- se alege un număr aleator t intre 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 părinții din care provin.

Subliniem că punctul de tăiere poate diferi de la o pereche la alta.

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, valoare mică, *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 obțin câteva astfel de generații (sau se epuizează un timp alocat procesului, sau se observă că media populației nu se îmbunătățește în ultimele iterații efectuate), se raportează valoarea celui mai bun cromozom din ultima generație[2].

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 rezolvată[3] – a se vedea secțiunea 9.4.

9.3 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 oricâte ori, inclusiv niciodată. De exemplu, schema $(*0101)$ se potrivește cu doi cromozomi[4]: (00101) și (10101). 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 $C_k^0 + C_k^1 + \cdots + C_k^k = 2^k$ 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

[2] În practică se preferă strategia elitistă: se returnează cel mai bun individ al tuturor generațiilor.

[3] S-a stabilit "ecuația" Genetic Algorithms + Data Structures = Evolution Programs, [12].

[4] În acest caz spunem că schema este reprezentată de cei doi cromozomi.

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 indicierea 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 (se potrivesc cu) 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:

$$P(A) = \frac{f(A)}{\sum\limits_c f(c)} \tag{9.6}$$

unde însumarea se face după toți cromozomii c ai populației curente. Reamintind că n este numărul de cromozomi din populație, avem:

$$n(S,t+1) = n(S,t) \cdot \frac{f(S,t)}{\frac{1}{n}\sum\limits_c f(c)} \tag{9.7}$$

Cantitatea $\overline{f(t)} = \sum_c f(c)/n$ este chiar valoarea medie a populației de la momentul t, deci avem:

$$n(S,t+1) = n(S,t) \cdot \frac{f(S,t)}{\overline{f(t)}} \tag{9.8}$$

Interpretarea ecuației de mai sus este următoarea: numărul de reprezentanți ai schemei S care vor exista la momentul $t+1$ este dependent de valoarea schemei 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 arată prin inducție că:

$$n(S,t) = n(S,0)(1+\varepsilon)^t, \ \forall t \in \{1, 2, \dots\} \tag{9.9}$$

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 obținuti 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:

$$S_1 = (*01****) \tag{9.10}$$

și

$$S_2 = (1****0*) \tag{9.11}$$

Să presupunem că în procesul de încrucișare tăietura se face după a patra genă:

$$\begin{array}{rcl} c & = & (1 \quad 0 \quad 1 \quad 0 \quad | \quad 1 \quad 0 \quad 0) \\ S_1 & = & (* \quad 0 \quad 1 \quad * \quad | \quad * \quad * \quad *) \\ S_2 & = & (0 \quad * \quad * \quad * \quad | \quad * \quad 0 \quad *) \end{array} \tag{9.12}$$

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[5]. Intuitiv, este clar faptul că lungimea mică a schemei S_1 mărește șansa de supraviețuire, față de S_2 care poate fi ușor "spartă" în cromozomii copii. Desigur, contează poziția tăieturii.

Tăietura poate să apară uniform aleator (echiprobabil) în $k-1$ poziții. Probabilitatea de distrugere a unei scheme este:

$$P_d(S) = \frac{\delta(S)}{k-1} \tag{9.13}$$

și evident probabilitatea evenimentului contrar, reprezentând supraviețuirea schemei este

$$P_s(S) = 1 - P_d(S) = 1 - \frac{\delta(S)}{k-1} \tag{9.14}$$

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:

$$P_s(S) = 1 - p_c \cdot \frac{\delta(S)}{k-1} \tag{9.15}$$

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 mărind șansele de supraviețuire a unei scheme. Ca atare, șansa de supraviețuire este de fapt dată printr–o inegalitate:

$$P_s(S) \geq 1 - p_c \cdot \frac{\delta(S)}{k-1} \tag{9.16}$$

[5] În exemplul dat, schema S_2 nu e distrusă dacă al doilea cromozom care participă la încrucișare vine cu aceleași gene pe pozițiile fixe din S_2.

deci schemele de lungime mică au șanse crescute de supraviețuire.

Combinând rezultatele obținute pentru partea de selecție și încrucișare, obținem:

$$n(S,t+1) \geq n(S,t) \cdot \frac{f(S,t)}{\overline{f(t)}} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k-1}\right] \tag{9.17}$$

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 bit oarecare să nu fie modificat este $(1 - p_m)$. Alegerile biților care să sufere mutație sunt evenimente independente, deci probabilitatea ca cei $o(S)$ biți ficși ai unei scheme să se mențină (și deci ca întreaga schemă să se mențină) este:

$$P_s(S) = (1 - p_m)^{o(S)} \tag{9.18}$$

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:

$$n(S,t+1) \geq n(S,t) \cdot \frac{f(S,t)}{\overline{f(t)}} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k-1} - p_m \cdot o(S)\right] \tag{9.19}$$

Se poate da acum enunțul teoremei schemelor, teorema fundamentală a algoritmilor genetici datorată lui Holland (1975):

Teorema 4 (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, [12]. Un algoritm genetic execută un proces de căutare prin suprapunerea unor scheme scurte, de ordin mic și de valoare mare, numite 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.

9.4 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 minim[6]. O soluție este dată ca o permutare a mulțimii $\{1, \ldots, n\}$.

[6] În termeni de grafuri: se cere determinarea unui ciclu Hamiltonian de cost minim.

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 si neconsiderând soluțiile simetrice de forma $A \to B \to C \to A \equiv A \to C \to B \to A$ – mulțimea permutărilor cu $19!/2 \ < 10^{17}$ 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 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:

$$f(x, y) = x^2 - y^3 + 2 \cdot x \cdot \sin(y) \tag{9.20}$$

cu condiția ca variabilele x și y să satisfacă constrângerea:

$$1 \leq x^3 - \cos(y) + y^2 \leq 5 \tag{9.21}$$

cum încorporăm restricția în algoritmul genetic? Dacă folosim varianta clasică de codificare a unui individ, împreună cu operatorii de încrucișare și de mutație prezentați, cum asigurăm faptul că operatorii dați nu duc indivizii în zone în care constrângerea (9.21) 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 tipuri de date particularizate pentru reprezentarea de indivizi și a unor operatori de încrucișare și de mutație care produce indivizi valizi.

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 permit populații 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 algoritmii de reparare – este posibil să se integreze cunoștințe din domeniu în metodele de corecție; trebuie zis însă că elaborarea 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 a problemei împreună 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 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 m obiecte având greutățile $G_i > 0$ și valorile asociate $V_i > 0$, $1 \leq i \leq m$. 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. 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, $1 \leq i \leq m$, $x_i \in \{0, 1\}$, valoarea 0

însemnând că obiectul nu este luat, iar 1 - că e adăugat în rucsac. Se impune, evident, condiția de viabilitate a unui vector $\mathbf{x}$:

$$\sum_{i=1}^{m} x_i \cdot G_i \leq C$$

iar funcția de maximizat – numită și profit în acest caz – este:

$$P(\mathbf{x}) = \sum_{i=1}^{m} x_i \cdot V_i$$

9.4.1 Varianta cu penalizare

Pentru fiecare individ $\mathbf{x}$ se va considera valoarea sa $f(\mathbf{x})$:

$$f(\mathbf{x}) = \sum_{i=1}^{m} x_i \cdot V_i - Pen(\mathbf{x})$$

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 $\sum_{i=1}^{m} x_i \cdot G_i$ și C – atunci se poate folosi o funcție de tip logaritmic, liniar, pătratic, exponențial etc. Efectele alegerii unei asemenea funcții au fost analizate pe diferite situații; a se vedea [12] pentru rezultate experimentale și interpretarea lor.

9.4.2 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 $\mathbf{x}' = (x'_1, \ldots, x'_m)$ astfel încât $\sum_{i=1}^{m} x'_i G_i \leq C$.

Algoritmul de reparare a unui individ este:

Listing 9.1: Repararea unui vector invalid

```
function reparare(x, G, C) returns a vector
begin
 rucsac−supraincarcat := false
 x' := x
 if ∑_{i=1}^{m} x'_i · G_i > C
  then rucsac−supraincarcat := true
```

```
 end if
 while rucsac−supraincarct = true
  i:= selecteaza un obiect din rucsac (#)
  scoate obiectul i din rucsac: x'_i := 0
  if Σ_{i=1}^{m} x'_i · G_i ≤ C
   then rucsac−supraincarcat := false
  end if
 end while
 return 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.

9.4.3 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 m componente în baza 10, fiecare element x_i având proprietatea că $1 \leq x_i \leq m - i + 1$, $\forall i \in \{1, \ldots, m\}$. De exemplu, pentru vectorul $\mathbf{x} = (3, 3, 4, 1, 1, 1)$ asociat listei de obiecte $L = (1, 2, 3, 4, 5, 6)$, decodificarea se obține astfel: se scoate elementul de indice $x_1 = 3$ din lista L, adică obiectul 3 și îl adăugăm în rucsac; L devine $(1, 2, 4, 5, 6)$; apoi se scoate elementul de indice $x_2 = 3$ (adică obiectul 4) 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: 3, 4, 6, 1, 2, 5. Fiecare cromozom codifică astfel o ordine de adăugare a obiectelor în rucsac. Adăugarea obiectului în rucsac se face numai dacă nu duce la depășirea capacității C.

Se poate vedea că operație de încrucișare va duce întotdeauna la copii valizi, adică pentru un copil $\mathbf{z} = (z_1, \ldots, z_m)$ avem că $1 \leq z_i \leq m - i + 1$ dacă și părinții au aceeasi 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 $\{1, \ldots, m - i + 1\}$ diferită de x_i.

Listing 9.2: Utilizarea codificării ordinale

```
procedure decodificare(x)
```

```
begin
 construieste o lista L de obiecte
 greutateTotala := 0
 profitTotal := 0
 for i = 1,m
  j := x_i
  o := L_j
  sterge elementul al j -lea din lista L
  if greutateTotala + G_o ≤ 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 $\{1, \ldots, n\}$. Indivizii rezultați — adică cei care realizează populația inițială — vor fi deci generați aleatori.

Rezultate experimentale pentru cele trei variante de rezolvare sunt date în [12]. 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.

9.5 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 [12].

Se dau următoarele:

- o mulțime de profesori $\{T_1, \ldots, T_m\}$
- o listă de intervale de timp (ore) $\{H_1, \ldots, H_n\}$
- o listă de săli de clasă $\{C_1, \ldots, C_k\}$

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 preda 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 fi[7]:

- 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.

[7] Dar nimic nu ne împiedică să concepem alți operatori.

Bibliografie

[1] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016.

[2] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[3] R. Andonie and A. Cațaron, *Inteligenţă computaţională*. Universitatea Transilvania din Braşov, 2002. https://www.cwu.edu/faculty/sites/cts.cwu.edu.faculty/files//users/142/documents/Cursul%20de%20IA%20Brasov.pdf.

[4] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. Wiley, 2002.

[5] R. C. Eberhart and Y. Shi, *Computational intelligence - concepts to implementations*. Elsevier, 2007.

[6] "Stanford Machine Learning, curs online Coursera." https://www.coursera.org/specializations/machine-learning-introduction, 2023. Accesat: 2023-10-18.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2018. https://neuralnetworksanddeeplearning.com.

[9] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, (Philadelphia, PA, USA), pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[10] G. A. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, no. 3, pp. 77–88, 1988.

[11] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE transactions on neural networks*, vol. 3 5, pp. 698–713, 1992.

[12] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Heidelberg: Springer-Verlag, third ed., 1996.