Inteligență artificială

Versiunea 22 noiembrie 2022

Lucian M. Sasu, Ph.D.

Cuprins

Capitolul 1

Introducere

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

- modele cu învățare automată (machine learning) – modele liniare, mixturi Gaussiene, rețele neurale (sau "neuronale") artificiale, Support Vector Machines, arbori de decizie 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 computațională este rapidă, iar pentru o problemă se pot folosi mai multe abordări: de exemplu, optimizarea se poate face prin algoritmi genetici sau prin anumite familii de rețele neurale.

Cursul de față se axează pe primele două direcții.

Metodele din inteligența computațională 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 IC. 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 IC este total diferită: pentru rețele neurale sau algoritmi genetici, 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 IC 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 propuse în cadrul IC 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 IC 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 inteligenței computaționale: adaptarea.

Rețelele neurale artificiale reprezintă grupuri interconectate de neuroni artificiali care au abilitatea de a învăța din și a se adapta la mediul lor, construind un model al lumii. Ele au apărut ca răspuns la modelarea activității creierului biologic, precum și ca modalitate propusă pentru a obține sisteme artificiale capabile să recunoască șabloane. Exemple de rețele neurale și algoritmi de instruire se găsesc în [1], [2].

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 computațională pot reprezenta hibridizări ale celor de mai sus; de exemplu, există sisteme neuro-fuzzy, iar ajustarea parametrilor pentru un sistem adaptiv se poate face prin algoritmi genetici. Alegerea 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 ([3]) 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.

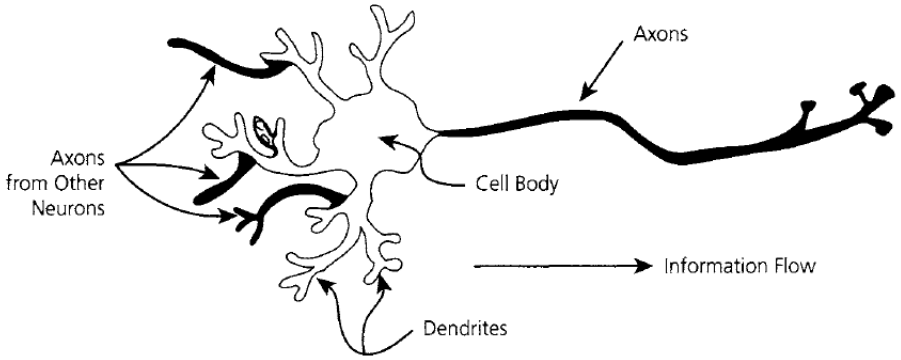


Figura 1.1: Neuron natural [3]

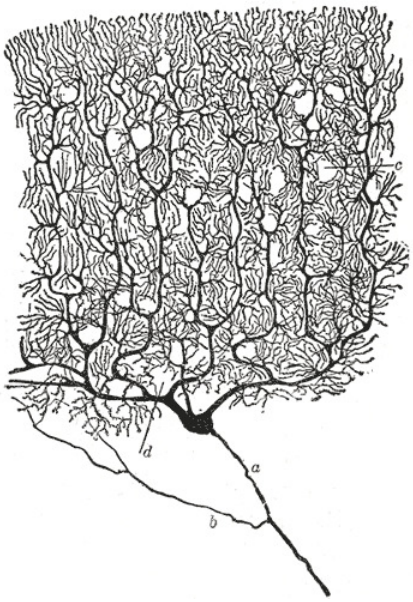


Figura 1.2: Neuron de tip Purkinje din cortexul cerebelar; sursa http://en.wikipedia.org/wiki/Neuron.

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. Modelele neurale .

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 inteligența computațională

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

Există trei tipuri principale de învățare:

1. supervizată
2. nesupervizată
3. prin întărire

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 obținerea unui sistem ce poate să furnizeze o valoare de ieșire adecvată pentru o anumită valoare de intrare ce nu este prezentă în setul de instruire.

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

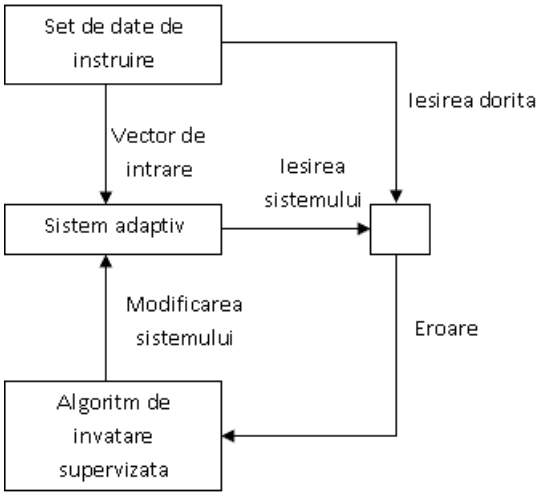


Figura 1.3: Schema de lucru pentru învățare supervizată

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.

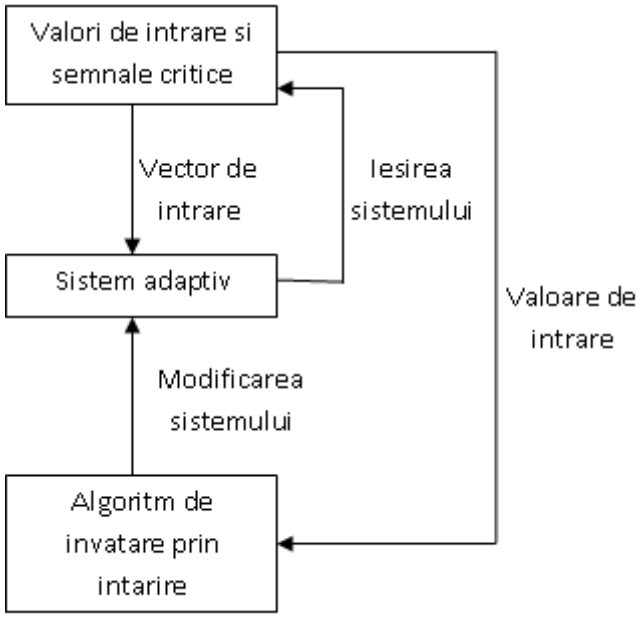


Figura 1.4: Schema de lucru pentru învățare prin întărire

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 computaționale inteligente. Este prezentă în sistemele naturale, de exemplu în creierul nou născuților, unde auto-organizarea se manifestă în principal prin distrugerea legăturilor nefuncționale. Auto-organizarea este definită astfel:

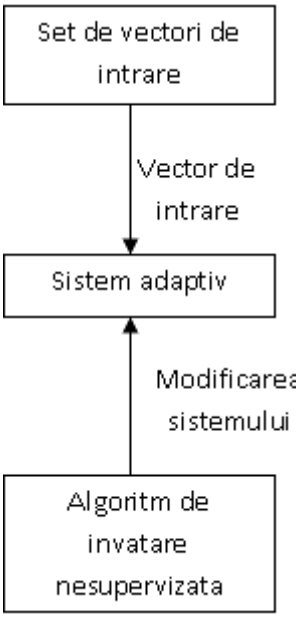


Figura 1.5: Schema de lucru pentru învățare nesupervizată

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 [4].*

sau:

Definiția 3. *Sistemele cu auto-organizare se auto-organizează pentru a clasifica percepțiile din mediu în percepții ce pot fi recunoscute, sau șabloane [4].*

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 construire de model de regresie.

2.1 Exemplu și notații

Regresia liniară este o metodă folosită pentru predicția unei valori numerice dintr-o mulțime infinită de valori. Ca exemplu, să presupunem că vrem să facem predicția costului unui apartament, dată fiind suprafața sa. Se cunosc date anterioare despre vânzarea unor astfel de apartamente, precum în tabelul 2.1.

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 care să ne permită aproximarea prețului (număr real) pentru alte apartamente. O exemplificare este dată în figura 2.1.

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"[1] norul de puncte reprezentat[2]. Valoarea estimată de model pentru apartamentul cu

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

[2] Pentru cazul cu mai multe date de intrare se obține o varietate liniară – adică plan

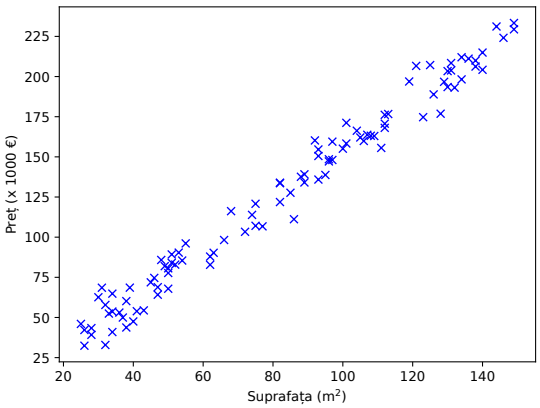


Figura 2.1: Reprezentarea grafică a datelor de vânzare a unor apartamente. Pe abscisă este măsurată suprafața (în metri pătrați), pe ordonată este prețul (în mii de euro).

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.

Eroarea estimării este influențată de diferența dintre valoarea actuală și cea prezisă de model, pentru cazurile cunoscute.

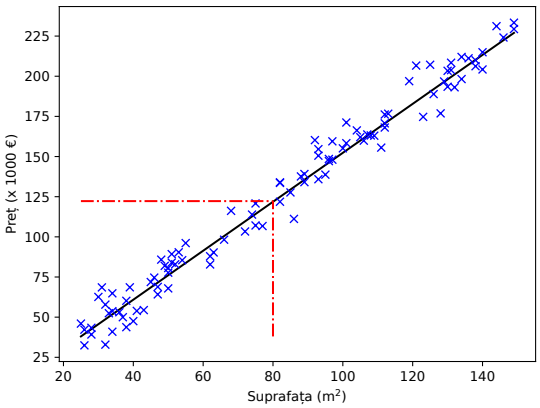


Figura 2.2: Model liniar de predicție și aproximarea costului unui apartament de 80 de metri pătrați.

pentru 2 dimensiuni de intrare etc. – dar modul de determinare a varietății este similar cu ceea ce se prezintă pentru o singură variabilă.

Modelul liniar are forma:

$$pret\_estimat = a \cdot suprafata + b$$

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 forme 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 se poate dovedi un punct de plecare bun, iar modelele mai simple se recomandă a fi încercate printre primele. În plus, un model liniar este ușor de interpretat: creșterea valorii variabilei $suprafata$ cu o unitate (1 metru pătrat) duce la creșterea prețului total cu a unități monetare; valoarea b este prețul de pornire.

Avem mai sus un exemplu de instruire supervizată: se pornește de la un set de date cu perechi formate din valoare de intrare (*e.g.* suprafața) și valoare de ieșire asociată (*e.g.* costul apartamentului de 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[3]. 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) 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:

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

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

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

Fluxul de lucru în învățarea automată[4] 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.

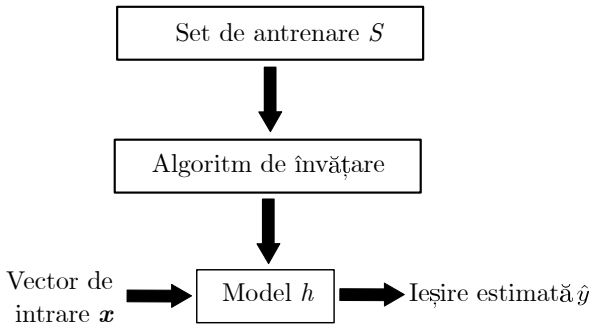


Figura 2.3: Fluxul de lucru într-un proces de instruire automată.

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:

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

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 de intrare considerată până acum – să se mai considere și alte variabile de intrare: etaj, număr de balcoane, număr de locuri de parcare, gradul de poluare a zonei etc.; în acest caz, modelul ar fi unul multivariat (mai multe valori de intrare considerate). Coeficienții θ_0 și θ_1 din ecuația (2.3) se mai numesc parametri ai modelului de predicție și se determină prin pasul de învățare.

[4]În limba engleză: machine learning.

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 $(0, 0)$.

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

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

Funcția de eroare J se mai numește și funcție de eroare – sau de cost – a modelului (în limba engleză: error, loss, cost function; în unele lucrări se face distincție între loss function – eroare pentru un singur caz – și cost function – eroarea agregată peste mai multe valori, de exemplu pe tot setul de antrenare). Se pot folosi și alte funcții de cost, de exemplu incluzând constrângeri impuse valorilor parametrilor $\boldsymbol{\theta}$ – a se vedea secțiunea 2.5. Funcția de mai sus este o alegere populară pentru problemele de regresie, dar nu singura posibilă. Factorul m de la numitor apare pentru a calcula media erorii (altfel, eroarea ar crește de fiecare dată când se adaugă în setul de instruire o pereche $(\mathbf{x}^{(i)}, y^{(i)})$ pentru care $h_\theta(\mathbf{x}^{(i)}) \neq y^{(i)}$, în timp ce media permite compararea erorilor modelului peste seturi de date de dimensiuni diferite); numitorul 2 poate fi omis, dar se utilizează din motive estetice pentru calculele de mai târziu. Eroarea J din 2.5 este jumătate din eroarea pătratică medie (eng: mean squared error):

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

E mai simplu 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

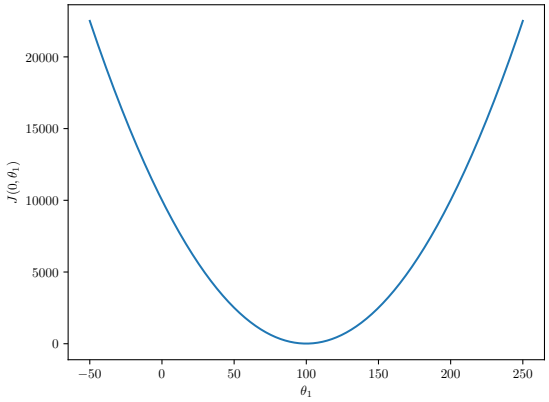


Figura 2.4: Funcția de eroare pătratică, cu $\theta_0 = 0$ și θ_1 variabil.

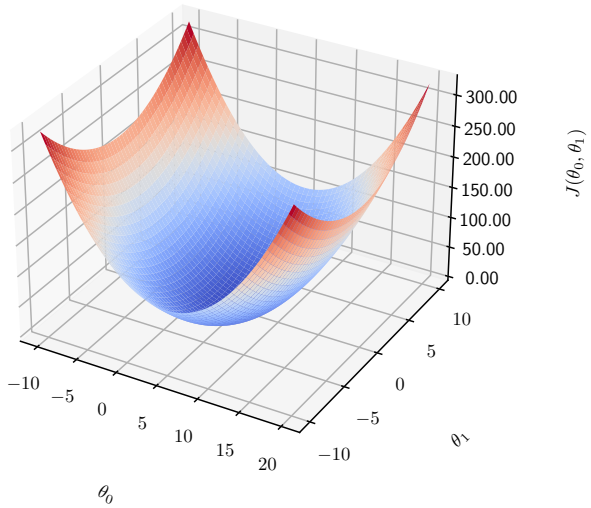


Figura 2.5: Funcția de eroare pentru model liniar univariat cu θ_0 și θ_1 variabili.

mai mare sau egal cu zero, a se vedea figura 2.4. Pentru θ_0, θ_1 oarecare forma funcției de eroare este dată în figura 2.5.

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 a funcției de eroare 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 2.6. Se poate arăta că aceste contururi sunt eliptice, pentru model de predicție liniar; pentru valori v tot mai mari avem elipse tot mai întinse.

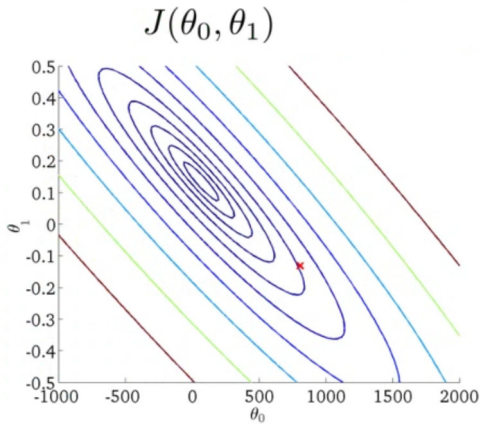


Figura 2.6: Curbe de contur pentru functia de eroare a unui model liniar univariat, cu parametrii θ_0, θ_1 variabili.

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

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

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.

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.

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

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

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

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

Această simultaneitate e cerută din cauză că la calculele (2.8), (2.9) 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 (2.8) și (2.9) 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 simultan peste componentele vectorului $\boldsymbol{\theta}$.

Aceleași formule se scriu în mod vectorial ca:

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

Dacă folosim notația nabla pentru vectorul de derivate parțiale (vectorul gradient)[5]:

$$\nabla_{\boldsymbol{\theta}} J = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix} \tag{2.11}$$

atunci mai putem scrie formula de modificare a ponderilor ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}} J(\theta_0, \theta_1) \tag{2.12}$$

În bibliotecile care permit calcul matriceal atribuirea simultană este implementată automat. O formă matriceală generală pentru gradient 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

[5]Vectorul de derivate parțiale este un vector de funcții; acestea se vor evalua pentru perechea de valori θ_0, θ_1.

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 de oricâte variabile. Menționăm că în general se poate ajunge într-un minim local 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:

`repeta{`

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{simultan pentru } j = 0, 1 \tag{2.13}$$

`} 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:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \tag{2.14}$$

și formula de modificare a ponderilor din (2.13) devine:

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \quad \text{simultan pentru } j = 0, 1 \tag{2.15}$$

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)[6], 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 – global sau local – 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:

[6] Sau punct de inflexiune; dar pentru funcția de eroare considerată avem doar o valoare de minim.

1. modelul de predicție devine

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \ldots \theta_n \cdot x_n = \boldsymbol{\theta}^t \cdot \mathbf{x} \tag{2.16}$$

unde

$$\mathbf{x} = (x_0 = 1, x_1, \ldots, x_n)^t, \ \boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^t \tag{2.17}$$

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.

2. funcția de eroare J se păstrează, dar modelul h este cel din ecuația (2.16);

3. ecuația (2.13) din algoritmul de căutare devine:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \tag{2.18}$$

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

Pentru modelul multivariat sde 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 (2.18) folosind calcul matriceal. 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 $\mathscr{S}$:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \ldots & x_n^{(m)} \end{pmatrix} \tag{2.19}$$

unde linia $\mathbf{x}^{(i)}$ conține valorile predictive asociate celui de al i–lea caz din setul de instruire, iar vectorul coloană de indice $1 \leq j \leq n$ corespunde unei trăsături predictive. Valorile de ieșire corespunzătoare sunt de asemenea stocate matriceal, folosind un vector coloană:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m \tag{2.20}$$

Așa cum în (2.17) 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 (2.19) 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:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{2.21}$$

Funcția de cost J se rescrie matriceal astfel:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^t \mathbf{x}^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{2m} \left(X\boldsymbol{\theta} - \boldsymbol{y} \right)^t \left(X\boldsymbol{\theta} - \boldsymbol{y} \right) \end{aligned} \tag{2.22}$$

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:

$$(A + B)^t = A^t + B^t \tag{2.23}$$

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

$$(A_1 \cdot A_2 \dots A_p)^t = A_p^t \cdot A_{p-1}^t \dots A_1^t \tag{2.24}$$

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

$$a^t = a \tag{2.25}$$

- conform lucrării [5] secțiunea 2.4.1, ecuația (69):

$$\nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^t \mathbf{A} \right) = \mathbf{A} \tag{2.26}$$

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

$$\nabla_{\boldsymbol{\theta}} \left(\theta^t \mathbf{A} \theta \right) = (\mathbf{A} + \mathbf{A}^t) \cdot \theta \tag{2.27}$$

  pentru $\mathbf{A}$ matrice pătratică.

Pentru calculul matriceal al vectorului de derivate parțiale (gradienți) $\left(\frac{\partial}{\partial\theta_j}J(\theta)\right)_{j=0,n}$ dezvoltăm (2.22):

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m}\left(X\boldsymbol{\theta}-\boldsymbol{y}\right)^t\left(X\boldsymbol{\theta}-\boldsymbol{y}\right) \\ &= \frac{1}{2m}\left\{\left(\mathbf{X}\boldsymbol{\theta}\right)^t\mathbf{X}\boldsymbol{\theta}-\left(\mathbf{X}\boldsymbol{\theta}\right)^t\mathbf{y}-\mathbf{y}^t\left(\mathbf{X}\boldsymbol{\theta}\right)+\mathbf{y}^t\mathbf{y}\right\} \\ &= \frac{1}{2m}\left\{\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{X}\boldsymbol{\theta}-2\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{y}+\mathbf{y}^t\mathbf{y}\right\} \end{aligned} \tag{2.28}$$

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

$$\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \frac{1}{2m}\left\{\nabla_{\boldsymbol{\theta}}\left(\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{X}\boldsymbol{\theta}\right)-2\nabla_{\boldsymbol{\theta}}\left(\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{y}\right)+\nabla_{\boldsymbol{\theta}}\left(\mathbf{y}^t\mathbf{y}\right)\right\} \tag{2.29}$$

Considerând relațiile (2.23–2.27) și observând că scalarul $\mathbf{y}^t\mathbf{y}$ nu depinde de $\boldsymbol{\theta}$, obținem vectorul gradient:

$$\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \frac{1}{m}\left\{\mathbf{X}^t\mathbf{X}\boldsymbol{\theta}-\mathbf{X}^t\mathbf{y}\right\} = \frac{1}{m}\mathbf{X}^t\left(\mathbf{X}\boldsymbol{\theta}-\mathbf{y}\right) \tag{2.30}$$

Modificările de ponderi θ_j din ecuația (2.18) se scriu matriceal pentru vectorul $\boldsymbol{\theta}$ ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta}-\alpha\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \boldsymbol{\theta}-\frac{\alpha}{m}\mathbf{X}^t(\mathbf{X}\boldsymbol{\theta}-\mathbf{y}) \tag{2.31}$$

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

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

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:

$$\boldsymbol{\theta}^{(min)} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^{n+1}} J(\boldsymbol{\theta}) \tag{2.32}$$

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:

$$\nabla_{\boldsymbol{\theta}}J\left(\boldsymbol{\theta}^{(min)}\right) = \mathbf{0} \tag{2.33}$$

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

Deoarece funcția de eroare J e și convexă, condiția de minimizare necesară dată de (2.33) este și suficientă și deci se ajunge în unicul minim al lui J. Înlocuind formula gradientului $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ din ecuația (2.30) în (2.33) obținem:

$$\mathbf{X}^t \mathbf{X} \boldsymbol{\theta}^{(min)} = \mathbf{X}^t \mathbf{y} \tag{2.34}$$

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

$$\boldsymbol{\theta}^{(min)} = \left(\mathbf{X}^t \mathbf{X}\right)^{-1} \mathbf{X}^t \cdot \mathbf{y} \tag{2.35}$$

Precizări:

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

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 secțiunea 2.5.

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.

2.5 Overfitting, underfitting, regularizare

2.5.1 Overfitting, underfitting

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

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

$$pret = \theta_0 + \theta_1 x \tag{2.36}$$

2. polinom de gradul al doilea:

$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{2.37}$$

3. polinom de gradul 4:

$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{2.38}$$

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 $\theta_0, \theta_1, \ldots$.

Graficele celor trei forme polinomiale sunt date în figura 2.7 [6]. Putem considera cantitățile x, x^2, x^3, x^4 ca fiind variabile de intrare pe baza cărora se realizează predicția; faptul că ele provin de la același x (suprafața) este o chestiune secundară.

Intuitiv, polinomul de gradul întâi nu reușește să facă o aproximare prea bună a evoluției prețului față de suprafață. Spunem că modelul dat de primul polinom suferă de "underfitting"[7], fiind prea simplu pentru problema în cauză. Se mai spune despre un asemenea model că are "high bias"[8], deoarece face o presupunere mult prea simplistă pentru problema tratată.

[7] Aproximativ, în limba română: incapacitate de reprezentare; simplism.
[8] Înclinare prea pronunțată spre un model cu performanță predictivă slabă.

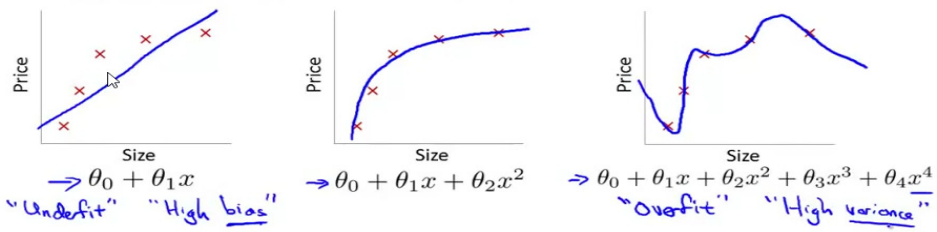


Figura 2.7: Trei polinoame pentru aproximarea prețului pornind de la suprafață, notată x [6].

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). Remarcăm însă forma nemonotonă, cu variații mari a predicției, fiind cazuri în care valoarea estimată scade în raport cu suprafața. Intuitiv, modelul suferă de "overfitting"[9], 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 (variabililtate) mare[10], datorită faptului că e prea complex pentru problema tratată.

Polinomul de gradul 2 prezintă cea mai credibilă formă (în sens intuitiv), 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 reprezenta foarte precis 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.

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

[9] Supraspecializare
[10] Engl: high variance.

O modalitate de evitare a overfitting–ului este reducerea numărului de trăsături: pentru problema noastră se evită folosirea variabilelor x^3, x^4. O altă variantă este alegerea judicioasă a modelului de predicție. Cea de a treia opțiune este regularizarea: se păstrează variabilele predictive, dar se impun constrângeri asupra parametrilor modelului — în cazul nostru asupra coeficienților θ_i.

2.5.2 Regularizare

Să considerăm că predicția se formează pe baza funcției polinomiale

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{2.39}$$

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:

$$J(\boldsymbol{\theta}) = \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right\} + 100 \cdot \theta_3^2 + 100 \cdot \theta_4^2 \tag{2.40}$$

Minimizarea lui J din ec. (2.40) 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 înmulțește cu nicio variabilă predictivă); vom impune deci constrângeri doar asupra lui $\theta_1, \theta_2, \ldots$ – să aibă valori absolute mici. Scopul final este de a evita overfitting–ul. Principiul se aplică și dacă se pleacă de la variabile de intrare independente, nu neapărat $suprafata, suprafata^2, \ldots$. Ca atare, ec. (2.40) se rescrie mai general ca:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right] + \lambda \sum_{j=1}^{n} \theta_j^2 \tag{2.41}$$

unde $\lambda > 0$. 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 prea simplist): funcția de estimare returnează mereu θ_0, indiferent de intrare.

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

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \left(X\boldsymbol{\theta} - \boldsymbol{y}\right)^t \left(X\boldsymbol{\theta} - \boldsymbol{y}\right) + \lambda \left\|\theta[1:]\right\|^2 \tag{2.42}$$

unde $\|\cdot\|$ este norma Euclidiană a vectorului argument. Se observă omiterea termenului liber din calculul de normă de vector.

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

`repeta{`

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

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)} + 2 \cdot \lambda \cdot \theta_j\right],\ j = 1, \ldots, n$$

`} pana la convergenta`

unde atribuirile se efectuează în mod simultan.

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

$$\boldsymbol{\theta} = \left(\mathbf{X}^t\mathbf{X} + 2 \cdot \lambda \cdot \underbrace{\begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}}_{(n+1)\times(n+1)}\right)^{-1} \cdot \mathbf{X}y \tag{2.43}$$

unde matricea care se înmulțește cu λ se obține din matricea unitate de ordinul $n+1$, modificând primul element în 0 (termenul θ_0 nu se regularizează). Se poate arăta că pentru $\lambda > 0$ termenul din ecuația (2.43) pentru care se cere inversarea este o matrice nesingulară, indiferent de valorile din matricea $\mathbf{X}$.

Capitolul 3

Regresia logistică

3.1 Încadrare, motivație

Regresia logistică este folosită pentru estimare de probabilitate condiționată și clasificare. Inițial dezvoltată pentru recunoașterea a două clase, a fost ulterior 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 conținutul lui, subiectul emailului, faptul că expeditorul face sau nu parte din lista de contacte etc.
- clasificarea unei tumori ca fiind benignă sau malignă, date fiind rezultatele unor analize;
- clasificarea unei imagini: conține un câine sau o pisică (fiecare imagine are exact unul din aceste două animale); sau dacă 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 unui pixel dintr–o imagine ca aparținând unui măr, pară, banană, cireașă sau fundal.

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. Clasificarea se face prin determinarea acelei clase pentru care probabilitatea condiționată $P(clasa_k|vector\_de\_intrare)$ 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. Mai departe, probabilitatea din (3.3) este o probabilitate condiționată de intrarea curentă – în cazul nostru vectorul $\mathbf{x}$

[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".

– 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 totodată influențată 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.

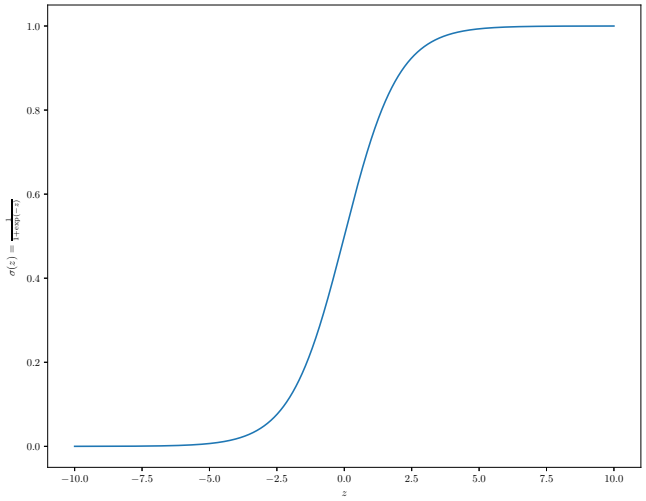


Figura 3.1: Graficul sigmoidei logistice definită în ecuația 3.4

Pentru funcția σ următoarele proprietăți sunt notabile:

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

iar valoarea derivatei se calculează ușor pe baza valorii lui σ:

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

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

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]:

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

$$= \frac{1}{1 + \exp(\boldsymbol{\theta}^t \cdot \mathbf{x})} = \sigma(-\boldsymbol{\theta}^t \cdot \mathbf{x}) \tag{3.8}$$

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 $P(y^{(i)} = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta})$ cât mai aproape de 1;

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 $P(y^{(i)} = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta})$ să fie cât mai aproape de 0.

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

$$P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) \tag{3.9}$$

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 (3.9) 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.

Se poate însă să se folosească și alt prag de discriminare pentru clase, 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 seturi de date 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

[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.

comparat cu un prag t mare (de exemplu 0.8, sau 0.99) pentru a decide că e vorba într–adevăr de spam.

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:

$$\mathbf{X} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{3.10}$$

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

$$\hat{\mathbf{y}} = \sigma(\mathbf{X} \cdot \boldsymbol{\theta}) \tag{3.11}$$

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

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[4], dacă pragul este 0.5.

Inegalitatea (3.9) se scrie echivalent:

$$\begin{aligned} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) \quad &\iff \quad \tfrac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \geq \tfrac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ &\iff \quad 1 \geq \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x}) \\ &\overset{\text{(logaritmând)}}{\iff} \quad 0 \geq -\boldsymbol{\theta}^t \cdot \mathbf{x} \\ &\iff \quad \boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0 \end{aligned} \tag{3.12}$$

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.

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 (3.12) am considerat – în mod mai degrabă arbitrar – că situația cu egalitate să fie tratată ca un caz pozitiv.

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

[4]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.

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 $\boldsymbol{\theta}^t \cdot \mathbf{x} = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_1^2 + \theta_4x_2^2 + \theta_5x_1x_2$. Pentru valorile[5] $\theta_0 = -4, \theta_1 = \theta_2 = \theta_5 = 0$, $\theta_3 = \theta_4 = 1$ 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 o clasă sau de cealaltă. 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.

O variantă este dată de utilizarea aceleiași funcții de eroare din capitolul de regresie liniară, ecuația (2.22) pagina 24. Se arată însă că pentru problema estimării de probabilitate condiționată, dată fiind forma funcției $h_{\boldsymbol{\theta}}$ din ecuația (3.3), funcția de eroare nu mai este convexă și în acest caz o căutare bazată pe gradient se poate opri într–un minim local — situație exemplificată în figura 3.2.

Vom defini J de așa manieră încât să fie convexă:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}) \tag{3.13}$$

și în plus funcția $Cost(\cdot, \cdot)$ să îndeplinească următoarele condiții:

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:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\ln h_{\boldsymbol{\theta}}(\mathbf{x}) & \text{dacă } y = 1 \\ -\ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{dacă } y = 0 \end{cases} \tag{3.14}$$

logaritmii fiind în baza e.

Cele două ramuri ale funcției $Cost$ sunt reprezentate în figura 3.3. Dreptele $x = 0$ și respectiv $x = 1$ sunt asimptote verticale pentru cele două ramuri ale lui $Cost$.

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

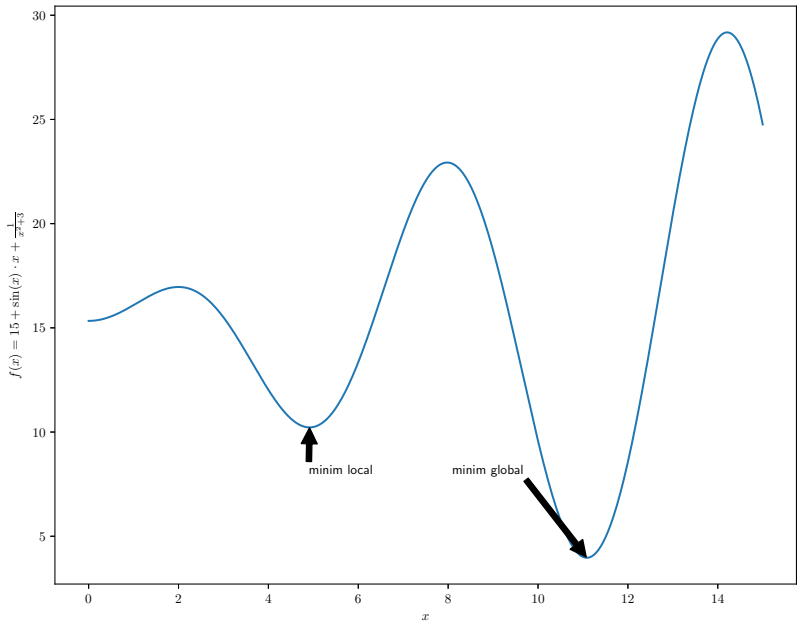


Figura 3.2: Minim global și minim local pentru funcția neconvexă $f : [0, 15] \to \mathbf{R}$, $f(x) = 15 + \sin(x) \cdot x + \frac{1}{x^2+3}$

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

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = -y \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}) - (1 - y) \cdot \ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) \tag{3.15}$$

$$= -y \cdot \ln \hat{y} - (1 - y) \cdot \ln(1 - \hat{y}) \tag{3.16}$$

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

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) \approx 0 \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx y \tag{3.17}$$

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 $y = 0$, (3.17) 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 (3.16).

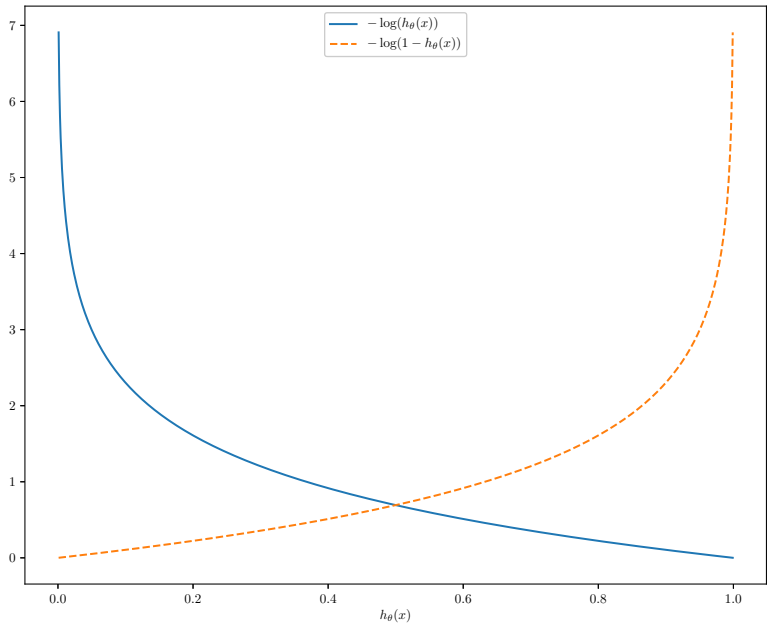


Figura 3.3: Cele două ramuri ale funcției *Cost* din ecuația (3.14)

Pentru condiția de depărtare, dacă[6] $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = M \gg 0$, 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 celălalt capăt al intervalului. Rezultă deci că și această a doua condiție este îndeplinită de definiția din formula (3.16).

În plus, deoarece fiecare din cele două ramuri ale funcției de cost din (3.14) 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:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \cdot \ln\left(1 - h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right)\right)\right] \tag{3.18}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \cdot \ln\left(1 - \hat{y}^{(i)}\right)\right] \tag{3.19}$$

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

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 (3.11) pagina 35, funcția de eroare binary cross–entropy se rescrie vectorizat ca:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left(\mathbf{y}^t \cdot \ln \hat{\mathbf{y}} + (\mathbb{1}_m - \mathbf{y})^t \cdot \ln(\mathbb{1}_m - \hat{\mathbf{y}})\right) \tag{3.20}$$

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 (3.14) 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:

$$\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbf{R}^{n+1}} J(\boldsymbol{\theta}) \tag{3.21}$$

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:

$$\frac{\partial J}{\partial \theta_j}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\cdot x_j^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)\cdot x_j^{(i)} \quad (3.22)$$

și modificarea din interiorul iterației devine:

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

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:

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

și deci modificarea din interiorul iterațiilor de instruire devine:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\frac{1}{m}\mathbf{X}^t(\hat{\mathbf{y}} - \mathbf{y}) \quad (3.24)$$

unde vectorul $\hat{\mathbf{y}}$ e calculat ca în (3.11).

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 21.

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

3.2.6 Regularizare

Dacă se permite ca valorile parametrilor[7] $\theta_1, \ldots, \theta_n$ 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

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

avea o performanță bună pe datele din set de testare[8]. Ca atare, se preferă 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:

$$J(\boldsymbol{\theta}) = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + (1-y^{(i)}) \cdot \ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right]}_{\text{Funcția de eroare din ecuația (3.18)}} \tag{3.25}$$

$$+ \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{\text{termenul de regularizare}} \tag{3.26}$$

$$= \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot \ln \hat{y}^{(i)} + (1-y^{(i)}) \cdot \ln(1 - \hat{y}^{(i)})\right]}_{\text{Funcția de eroare din ecuația (3.19)}} \tag{3.27}$$

$$+ \underbrace{\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2}_{\text{termenul de regularizare}} \tag{3.28}$$

Vectorizat se scrie astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left(\mathbf{y}^t \cdot \ln \hat{\mathbf{y}} + (\mathbb{1} - \mathbf{y})^t \cdot \ln(\mathbb{1} - \hat{\mathbf{y}})\right) + \|\boldsymbol{\theta}[1:]\|_2^2 \tag{3.29}$$

unde $\boldsymbol{\theta}[1:]$ este vectorul format din toate componentele lui $\boldsymbol{\theta}$ mai puțin prima, iar $\|\mathbf{v}\|_2$ este norma 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_i^2 în raport cu θ_i:

`repeta{`

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

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

`} pana la convergenta`

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

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

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \left[\frac{1}{m} \mathbf{X}^t (\hat{\mathbf{y}} - \mathbf{y}) + \lambda (0, \theta_1, \ldots, \theta_n)^t \right] \tag{3.30}$$

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

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:

$$h_{\boldsymbol{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y = 1 | \mathbf{x}; \boldsymbol{\Theta}) \\ P(y = 2 | \mathbf{x}; \boldsymbol{\Theta}) \\ \vdots \\ P(y = K | \mathbf{x}; \boldsymbol{\Theta}) \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{pmatrix} = \hat{\mathbf{y}} \tag{3.31}$$

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, oricare din ele poate fi determinată ca unu minus suma celorlalte $K - 1$. Discutăm suplimentar 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:

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | 1 \leq i \leq m \right\} \tag{3.32}$$

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:

$$softmax(\mathbf{z}) = (softmax(\mathbf{z}; 1), \ldots, softmax(\mathbf{z}; K))^t \tag{3.33}$$

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

$$softmax(\mathbf{z}; k) = \frac{\exp(z_k)}{\sum_{l=1}^{K} \exp(z_l)} \tag{3.34}$$

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:

$$\mathbf{\Theta} = \begin{bmatrix} | & | & | & | \\ \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \cdots & \boldsymbol{\theta}_K \\ | & | & | & | \end{bmatrix} \tag{3.35}$$

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:

$$h_{\mathbf{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y = 1|\mathbf{x}; \mathbf{\Theta}) \\ P(y = 2|\mathbf{x}; \mathbf{\Theta}) \\ \vdots \\ P(y = K|\mathbf{x}; \mathbf{\Theta}) \end{pmatrix} = softmax \begin{pmatrix} \boldsymbol{\theta}_1^t \cdot \mathbf{x} \\ \boldsymbol{\theta}_2^t \cdot \mathbf{x} \\ \vdots \\ \boldsymbol{\theta}_K^t \cdot \mathbf{x} \end{pmatrix} = \tag{3.36}$$

$$= softmax(\mathbf{\Theta}^t \cdot \mathbf{x}) = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_K \end{pmatrix} = \hat{\mathbf{y}} \tag{3.37}$$

pentru un vector de intrare $\mathbf{x} \in \mathbf{R}^{n+1}$.

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

$$P(y = k|\mathbf{x}; \mathbf{\Theta}) = \frac{\exp(\boldsymbol{\theta}_k^t \cdot \mathbf{x})}{\sum_{l=1}^{K} \exp(\boldsymbol{\theta}_l^t \cdot \mathbf{x})} = \hat{y}_k \in (0, 1) \tag{3.38}$$

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

$$k^{(max)} = \arg\max_{1\leq k\leq K} P(y=k|\mathbf{x};\boldsymbol{\Theta}) = \arg\max_{1\leq k\leq K} \hat{y}_k \tag{3.39}$$

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

3.3.4 Funcția de cost

Ca și în cazul funcției 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:

$$I(valoare\_logica) = \begin{cases} 1 & \text{dacă } valoare\_logica = adevarat \\ 0 & \text{dacă } valoare\_logica = fals \end{cases} \tag{3.40}$$

Funcția de eroare pentru regresia logistică multinomială se definește ca:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)}=k\right)\cdot\ln\hat{y}_k^{(i)}\right] \tag{3.41}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)}=k\right)\cdot\ln P\left(y^{(i)}=k|\mathbf{x}^{(i)};\boldsymbol{\Theta}\right)\right] \tag{3.42}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)}=k\right)\cdot\ln\frac{\exp\left(\boldsymbol{\theta}_k^t\mathbf{x}^{(i)}\right)}{\sum\limits_{l=1}^{K}\exp\left(\boldsymbol{\theta}_l^t\mathbf{x}^{(i)}\right)}\right] \tag{3.43}$$

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 poziţia k valoarea 1 şi 0 în rest. Cu această codificare se rescrie funcția de eroare mai compact ca:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{y}_{ohe}^{(i)}\right)^t\cdot\ln\left(\hat{\mathbf{y}}^{(i)}\right) \tag{3.44}$$

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

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 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 tipul $m \times n$ se poate obține cu:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} S_{ij} = \mathbb{1}_m^t \cdot \mathbf{S} \cdot \mathbb{1}_n \tag{3.45}$$

unde $\mathbb{1}_p$ este vector coloană de p elemente egale cu 1. Funcția de eroare din ecuatia (3.44) se rescrie ca:

$$J(\mathbf{\Theta}) = -\frac{1}{m} \mathbb{1}_m^t \cdot \left(\mathbf{Y} \odot \ln \hat{\mathbf{Y}} \right) \cdot \mathbb{1}_K \tag{3.46}$$

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

3.3.5 Calcularea gradientului

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 $1 \leq k \leq K$:

$$\nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(I(y^{(i)} = k) - \hat{y}^{(i)} \right) \right] = \tag{3.47}$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(I(y^{(i)} = k) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \mathbf{\Theta}) \right) \right] \tag{3.48}$$

pe care o demonstrăm în cele ce urmează: Având în vedere că gradientul e operator liniar, calculăm gradientul pentru un termen de cost corespunzător

[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*.

unei perechi de antrenare $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$, $J_i(\boldsymbol{\Theta})$:

$$J_i(\boldsymbol{\Theta}) = -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} = \tag{3.49}$$

$$= -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln\left(\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)\right) + \tag{3.50}$$

$$+ \sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) = \tag{3.51}$$

$$= -\sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right] + \tag{3.52}$$

$$+ \sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right)\right] \tag{3.53}$$

În suma din linia (3.53), 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. $J_i(\boldsymbol{\Theta})$ devine:

$$J_i(\boldsymbol{\Theta}) = -\sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right] + \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.54}$$

Aplicând gradientul, obținem:

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

$$= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \frac{\nabla_{\boldsymbol{\theta}_k} \sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} \tag{3.55}$$

$$= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \frac{\mathbf{x}^{(i)} \exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} \tag{3.56}$$

$$= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \mathbf{x}^{(i)} P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) \tag{3.57}$$

$$= -\mathbf{x}^{(i)} \left(I\left(y^{(i)} = k\right) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta})\right) \tag{3.58}$$

sau forma mai compactă

$$\nabla_{\boldsymbol{\theta}_k} J_i(\boldsymbol{\Theta}) = -\mathbf{x}^{(i)} \left(I\left(y^{(i)} = k\right) - \hat{y}^{(i)}\right). \tag{3.59}$$

Calculând media celor m termeni $\nabla_{\boldsymbol{\theta}_k} J_i(\boldsymbol{\Theta})$ se obține formula gradientului total $\nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta})$ din ecuația (3.48).

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

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{m}\mathbf{X}^t \cdot (\mathbf{Y} - \ln \hat{\mathbf{Y}}) \tag{3.60}$$

3.3.6 Algoritmul de instruire

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

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_k - \alpha \cdot \nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta}) \tag{3.61}$$

$$= \boldsymbol{\theta}_k + \frac{\alpha}{m}\sum_{i=1}^{m}\left[\mathbf{x}^{(i)}\left(I(y^{(i)} = k) - \hat{y}^{(i)}\right)\right] \tag{3.62}$$

simultan pentru toți $1 \leq k \leq K$. Modificarea se face iterativ, până când matricea $\boldsymbol{\Theta}$ 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 $\boldsymbol{\Theta}$ la valori inițiale aleatoare mici (în jurul lui 0) sau chiar 0;
2. `repeta{`

$$\boldsymbol{\theta}_k := \boldsymbol{\theta}_k + \frac{\alpha}{m}\sum_{i=1}^{m}\left[\mathbf{x}^{(i)}\left(I(y^{(i)} = k) - \hat{y}^{(i)}\right)\right] \tag{3.63}$$

`} pana la convergenta`

Matricial, algoritmul se rescrie ca:

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

$$\boldsymbol{\Theta} := \boldsymbol{\Theta} + \alpha\frac{1}{m}\mathbf{X}^t \cdot (\mathbf{Y} - \ln \hat{\mathbf{Y}}) \tag{3.64}$$

`} pana la convergenta`

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

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

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

3.3.7 Regularizare

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:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[I\left(y^{(i)}=k\right)\cdot \ln \hat{y}_k^{(i)}\right] + \frac{\lambda}{2}\sum_{i=1}^{n}\sum_{k=1}^{K}\theta_{ki}^2 \tag{3.65}$$

$$= -\frac{1}{m}\mathbb{1}_m^t \cdot \left(\mathbf{Y}\odot \ln \hat{\mathbf{Y}}\right)\cdot \mathbb{1}_K + \frac{\lambda}{2}\left\|\mathbf{\Theta[1:,:]}\right\|_F^2 \tag{3.66}$$

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

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{3.67}$$

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:

$$\nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[\mathbf{x}^{(j)}\left(I(y^{(j)}=k)-\hat{y}^{(i)}\right)\right] + \lambda\cdot[0,\boldsymbol{\theta}_{k,1:}]^t \tag{3.68}$$

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:

$$\nabla_{\mathbf{\Theta}} J(\mathbf{\Theta}) = -\frac{1}{m}\mathbf{X}^t\cdot(\mathbf{Y}-\ln\hat{\mathbf{Y}}) + \lambda\mathbf{\Theta}_0 \tag{3.69}$$

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.

3.4 Comentarii

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. 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:

$$P(y = k|\mathbf{x}; \boldsymbol{\Theta}) = \frac{\exp(\boldsymbol{\theta}_k^t \mathbf{x})}{\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} \tag{3.70}$$

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

$$\frac{\exp\left((\boldsymbol{\theta}_k - \boldsymbol{\psi})^t \mathbf{x}\right)}{\sum_{j=1}^{K} \exp\left((\boldsymbol{\theta}_j - \boldsymbol{\psi})^t \mathbf{x}\right)} = \frac{\exp(-\boldsymbol{\psi}^t \mathbf{x}) \exp\left(\boldsymbol{\theta}_k^t \mathbf{x}\right)}{\exp(-\boldsymbol{\psi}^t \mathbf{x}) \sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} = \tag{3.71}$$

$$= \frac{\exp(\boldsymbol{\theta}_k^t \mathbf{x})}{\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} = P(y = k|\mathbf{x}; \boldsymbol{\Theta}) \tag{3.72}$$

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. 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 3.2 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 (3.37) devine:

$$h_\Theta(\mathbf{x}) = \frac{1}{\exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x}) + \exp(\boldsymbol{\theta}_\mathbf{2}^t\mathbf{x})} \begin{pmatrix} \exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x}) \\ \exp(\boldsymbol{\theta}_\mathbf{2}^t\mathbf{x}) \end{pmatrix} \tag{3.73}$$

$$= \frac{1}{\exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x}) + \exp(\boldsymbol{\theta}_\mathbf{2}^t\mathbf{x})} \frac{\exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x})}{\exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x})} \begin{pmatrix} \exp(\boldsymbol{\theta}_\mathbf{1}^t\mathbf{x}) \\ \exp(\boldsymbol{\theta}_\mathbf{2}^t\mathbf{x}) \end{pmatrix} \tag{3.74}$$

$$= \frac{1}{\exp\left((\boldsymbol{\theta_1} - \boldsymbol{\theta_1})^t\mathbf{x}\right) + \exp\left((\boldsymbol{\theta_2} - \boldsymbol{\theta_1})^t\mathbf{x}\right)} \begin{pmatrix} \exp((\boldsymbol{\theta_1} - \boldsymbol{\theta_1})^t\mathbf{x}) \\ \exp((\boldsymbol{\theta_2} - \boldsymbol{\theta_1})^t\mathbf{x}) \end{pmatrix} \tag{3.75}$$

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

$$h_\Theta(\mathbf{x}) = \frac{1}{1 + \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right)} \begin{pmatrix} 1 \\ \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right) \end{pmatrix} \tag{3.76}$$

$$= \begin{pmatrix} \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ 1 - \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \end{pmatrix} \tag{3.77}$$

$$= \begin{pmatrix} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \\ 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \end{pmatrix} \tag{3.78}$$

Tot pentru $K = 2$, se verifică ușor că funcția de eroare (3.43) devine cea din (3.19).

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 mari. Un truc de calcul frecvent aplicat în practică este următorul: în loc de a se calcula funcția softmax precum în ecuația (3.34), 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:

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

$$= \frac{\exp(z_l)}{\sum_{k=1}^{K} \exp(z_k)} = softmax(\mathbf{z}; l) \tag{3.79}$$

Rezultatul e același, dar calculul se face cu exponenți mai mici, $z_l - M < z_l$. Se evită astfel depășirea de reprezentare (overflow).

3.4.4 Trucul "log sum exp"

Funcția de eroare pentru regresia logistică multinomială are forma:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} I\left(y^{(i)}=k\right)\cdot \ln\frac{\exp\left(\boldsymbol{\theta}_k^t\mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}^{(i)}\right)}\right] \tag{3.80}$$

Termenul cu logaritm permite simplificări de calcule:

$$\ln\frac{\exp\left(\boldsymbol{\theta}_k^t\mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}^{(i)}\right)} = \ln\left(\exp\left(\boldsymbol{\theta}_k^t\mathbf{x}^{(i)}\right)\right) - \ln\left(\sum_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}^{(i)}\right)\right) \tag{3.81}$$

$$= \boldsymbol{\theta}_k^t\mathbf{x}^{(i)} - \ln\left(\sum_{j=1}^{K}\exp\left(\boldsymbol{\theta}_j^t\mathbf{x}^{(i)}\right)\right) \tag{3.82}$$

Pentru al doilea termen din (3.82) 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:

$$\ln\left(\sum_{j=1}^{K}\exp(z_i)\right) = a + \ln\left(\sum_{j=1}^{K}\exp(z_i-a)\right) \tag{3.83}$$

pentru un a oarecare, demonstrată în cele ce urmează:

$$\ln\left(\sum_{j=1}^{K}\exp(z_i)\right) = \ln\left(\sum_{j=1}^{K}\left(\exp(z_i)\exp(a)\exp(-a)\right)\right) = \tag{3.84}$$

$$= \ln\left(\exp(a)\cdot\sum_{j=1}^{K}\left(\exp(z_i)\exp(-a)\right)\right) = \tag{3.85}$$

$$= \ln\exp(a) + \ln\left(\sum_{j=1}^{K}\exp(z_i-a)\right) = \tag{3.86}$$

$$= a + \ln\left(\sum_{j=1}^{K}\exp(z_i-a)\right) \tag{3.87}$$

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

Perceptronul liniar

4.1 Motivație, definiții, notații

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.

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:*

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i \cdot x_i + b \tag{4.1}$$

astfel încât:

$$\begin{cases} y(\mathbf{x}) > 0 \text{ pentru } \mathbf{x} \in \mathscr{C}_1 \\ y(\mathbf{x}) < 0 \text{ pentru } \mathbf{x} \in \mathscr{C}_2 \end{cases} \tag{4.2}$$

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 coeficienții (ponderile) $b, w_1, \ldots, w_n$ pentru care condițiile din (4.2) sunt îndeplinite.

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 $b \neq 0$.

Pentru cazul $n = 2$, $w_1 > 0$, $w_2 > 0$, $b < 0$ a se vedea reprezentarea din figura 4.1.

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

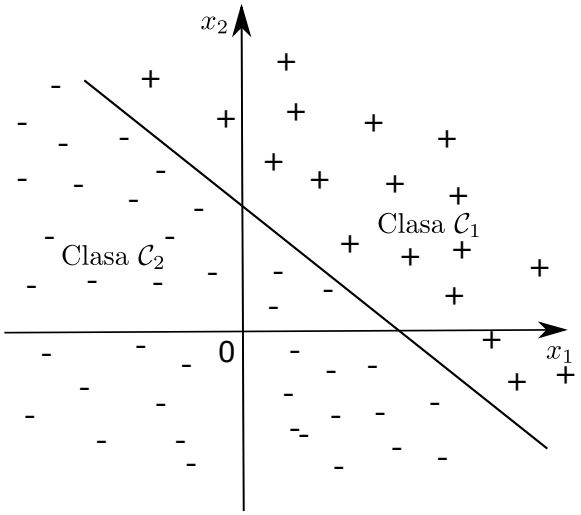


Figura 4.1: Mulțimi separabile liniar și suprafață de separare liniară $y(\mathbf{x}) = w_1x_1 + w_2x_2 + b$, $w_1 > 0$, $w_2 > 0$, $b < 0$. Clasa $\mathscr{C}_1$ e clasă pozitivă, clasa $\mathscr{C}_2$ e clasă negativă

(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, atunci:

$$y(\mathbf{x_1}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_1} + b = 0 \tag{4.3}$$

$$y(\mathbf{x_2}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_2} + b = 0 \tag{4.4}$$

Scăzând (4.4) din (4.3) 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 4.2.

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

$$z = \frac{|y(\mathbf{x})|}{\|w\|} \tag{4.5}$$

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

Putem renota termenul liber b din (4.1) 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 (4.1) se rescrie ca:

$$y(\mathbf{x}) = \mathbf{w}^t \cdot \mathbf{x} \tag{4.6}$$

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

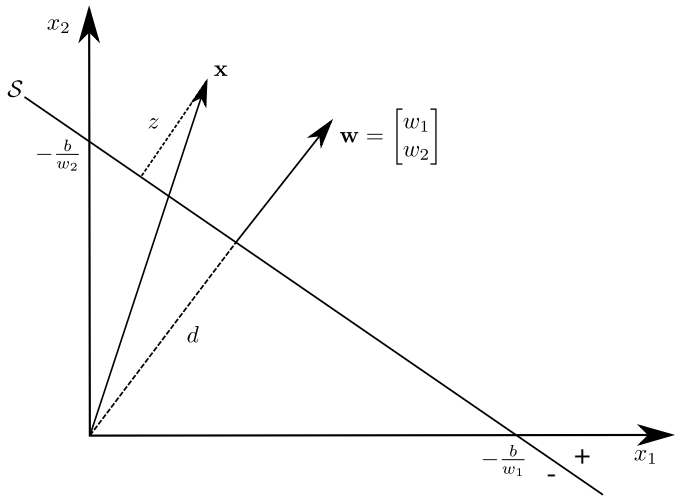


Figura 4.2: Suprafața de decizie S, distanța de la punctul de coordonate $\mathbf{x}$ la S, poziția vectorului de ponderi $\mathbf{w}$ față de S

4.2 Perceptronul liniar

Setul de instruire este

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | \mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{-1, 1\}, 1 \leq i \leq m \right\} \tag{4.7}$$

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

Reprezentarea unui perceptron este dată în figura 4.3.

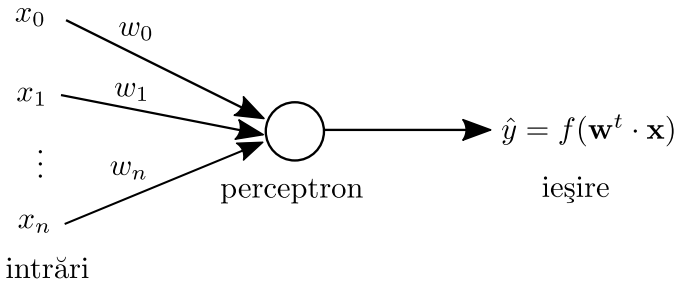


Figura 4.3: Reprezentarea unui perceptron liniar

Valoarea de activare a perceptronului este $z = \mathbf{w}^t \cdot \mathbf{x} = \sum_{i=0}^{n} w_i \cdot x_i$. Valoarea de ieșire a perceptronului se calculează cu funcția de activare "treaptă":

$$f(z) = \begin{cases} 1, & \text{dacă } z > 0 \\ -1, & \text{dacă } z < 0 \\ \text{nedefinit}, & \text{dacă } z = 0 \end{cases} \tag{4.8}$$

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

$$\hat{y} = f(\mathbf{w}^t \cdot \mathbf{x}) \tag{4.9}$$

La inferență, 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 deci pentru un atare caz nu se poate spune despre el că ar fi de clasă pozitivă sau negativă.

4.3 Algoritmul de instruire a perceptronului

Scopul algoritmului este ca, plecând de la setul de instruire $\mathscr{S}$ din ecuația (4.7), să obținem ponderile $\mathbf{w} = (w_0, w_1, \ldots, w_n)^t$ astfel încât:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_1 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_2 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{4.10}$$

Se demonstrează (vezi secțiunea 4.5) că dacă $\mathscr{C}_1$ și $\mathscr{C}_2$ sunt liniar separabile, atunci algoritmul de instruire poate să determine o funcție de separare y precum în ecuația (4.6).

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 diferență:

$$\mathbf{w}(1) \xrightarrow{+\Delta\mathbf{w}(1)} \mathbf{w}(2) \xrightarrow{+\Delta\mathbf{w}(2)} \mathbf{w}(3) \ldots \mathbf{w}(k) \xrightarrow{+\Delta\mathbf{w}(k)} \mathbf{w}(k+1) \ldots \tag{4.11}$$

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:

$$\Delta\mathbf{w}(k) = \alpha \cdot \mathbf{x}^{(i)} \tag{4.12}$$

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

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)} \tag{4.13}$$

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

Avem:

$$\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)} = \left(\mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)}\right)^t \cdot \mathbf{x}^{(i)} = \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} + \alpha \cdot \left(\mathbf{x}^{(i)}\right)^t \cdot \mathbf{x}^{(i)} =$$
$$= \mathbf{w}(k)^t \cdot \mathbf{x} + \left\|\mathbf{x}^{(i)}\right\|^2 \geq \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \tag{4.14}$$

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 (4.14) e de fapt strictă). Avem deci creșterea produsului scalar dintre ponderi și vectorul de intrare, așa cum ne-am propus.

Pentru cazul 2, facem modificarea ponderilor astfel:

$$\Delta\mathbf{w}(k) = -\alpha \cdot \mathbf{x}^{(i)} \tag{4.15}$$

deci

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) - \alpha \cdot \mathbf{x}^{(i)} \tag{4.16}$$

cu aceeași condiție pentru α. Se verifică, similar ca în (4.14) că $\mathbf{w}(k+1)^t \cdot \mathbf{x} < \mathbf{w}(k)^t \cdot \mathbf{x}$.

Avem deci că:

$$\Delta\mathbf{w}(k) = \begin{cases} \mathbf{0}_{n+1}, & \text{dacă } y^{(i)} = \hat{y}^{(i)} \\ \alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = +1 \text{ și } \hat{y}^{(i)} = -1 \\ -\alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = -1 \text{ și } \hat{y}^{(i)} = +1 \end{cases} \tag{4.17}$$

$$= \begin{cases} \mathbf{0}_{n+1}, & \text{dacă } y^{(i)} = \hat{y}^{(i)} \\ \alpha y^{(i)} \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} \neq \hat{y}^{(i)} \end{cases} \tag{4.18}$$

Putem scrie modificările din ecuația (4.18), astfel:

$$\Delta\mathbf{w}(k) = \frac{\alpha}{2}(y^{(i)} - \hat{y}^{(i)}) \cdot \mathbf{x}^{(i)} \tag{4.19}$$

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.

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. $k = k + 1$
3. Până când $corectClasificat = adevarat$
4. 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 4.5. Notăm cu $\mathbf{w}$ vectorul de ponderi produs la terminarea algoritmului, $\mathbf{w} = \mathbf{w}(k)$.

După ce algoritmul de instruire se oprește, clasificarea unui vector ca fiind de clasă pozitivă sau negativă se face ca în secțiunea 4.2.

4.4 Modificarea ponderilor ca gradient

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

$$J(\mathbf{w}) = \sum_{i:\hat{y}^{(i)} \neq y^{(i)}} \hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x^{(i)}} \tag{4.20}$$

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 $\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x^{(i)}} > 0$ conform justificărilor din cele două cazuri tratate la pagina 56.

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

$$\nabla_{\mathbf{w}} \left[\hat{y}^{(i)} \mathbf{w}^t \cdot \mathbf{x}^{(i)} \right] = \hat{y}^{(i)} \nabla_{\mathbf{w}} \left[\mathbf{w}^t \cdot \mathbf{x}^{(i)} \right] = \hat{y}^{(i)} \mathbf{x}^{(i)} \tag{4.21}$$

Dacă aplicăm algoritmul gradient descent pentru intrarea curentă, însamnă că modificăm 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 (4.20) și gradienții aferenți este imediată. Am preferat forma dată mai sus, fiind clasică.

4.5 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:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_1 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in \mathscr{C}_2 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{4.22}$$

(a se revedea condițiile 4.10), sau, mai pe scurt,

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \text{ pentru orice } i, 1 \leq i \leq m \tag{4.23}$$

Putem presupune că vectorul $\mathbf{w}_*$ este de normă 1; în caz contrar se poate face normarea lui[2], iar inegalitatea din (4.23) rămâne adevărată.

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

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > \gamma \text{ pentru orice } i, 1 \leq i \leq m \tag{4.24}$$

de exemplu

$$\gamma = \frac{1}{2} \cdot \min_{1 \leq i \leq m} \left\{ y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} \right\} > 0 \tag{4.25}$$

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 sfere centrate în origine și de rază R suficient de mare. Putem, de exemplu, să luăm

$$R = \max_{1 \leq i \leq m} \left\{ \|\mathbf{x}^{(i)}\| \right\} \tag{4.26}$$

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

1. $\alpha = 1$; valoarea lui α nu are de fapt importanță, 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 cel mult R^2/γ^2 modificări ale ponderilor, după care returnează un hiperplan de separare.*

[2]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 4.23 nu ar mai putea fi îndeplinită pentru niciun i.

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:

$$f(\mathbf{w}(k)^t\mathbf{x}^{(i)}) \neq y^{(i)} \Leftrightarrow y^{(i)} \cdot \mathbf{w}(k)^t\mathbf{x}^{(i)} < 0 \tag{4.27}$$

Pentru această situație de clasificare greșită a unui vector din setul de instruire, algoritmul efectuează modificare a lui $\mathbf{w}(k)$ conform ecuației (4.18).

Avem:

$$\begin{aligned}
\mathbf{w}(k+1)^t \cdot \mathbf{w}_* &= (\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)})^t \cdot \mathbf{w}_* && (4.28)\\
&= \mathbf{w}(k)^t \cdot \mathbf{w}_* + \underbrace{y^{(i)}(\mathbf{x}^{(i)})^t \cdot \mathbf{w}_*}_{>\gamma \text{ cf. } (4.25)} && (4.29)\\
&> \mathbf{w}(k)^t \cdot \mathbf{w}_* + \gamma && (4.30)
\end{aligned}$$

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

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* > k\gamma \tag{4.31}$$

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

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* \leq |\mathbf{w}(k+1)^t \cdot \mathbf{w}_*| \leq \|\mathbf{w}(k+1)\| \cdot \|\mathbf{w}_*\| = \|\mathbf{w}(k+1)\| \tag{4.32}$$

și din ultimele două inegalități avem:

$$\|\mathbf{w}(k+1)\| > k\gamma \tag{4.33}$$

Pe de altă parte, avem că:

$$\begin{aligned}
\|\mathbf{w}(k+1)\|^2 &= \|\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)}\|^2 && (4.34)\\
&= \|\mathbf{w}(k)\|^2 + \|y^{(i)}\mathbf{x}^{(i)}\|^2 + 2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)} && (4.35)\\
&= \|\mathbf{w}(k)\|^2 + \underbrace{|y^{(i)}|^2}_{=1} \cdot \|\mathbf{x}^{(i)}\|^2 + \underbrace{2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)}}_{<0 \text{ cf. } (4.27)} && (4.36)\\
&< \|\mathbf{w}(k)\|^2 + \|\mathbf{x}^{(i)}\|^2 && (4.37)\\
&\leq \|\mathbf{w}(k)\|^2 + R^2 && (4.38)
\end{aligned}$$

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

$$\|\mathbf{w}(k+1)\|^2 < kR^2 \tag{4.39}$$

Din inecuațiile (4.33) și (4.39) rezultă că:

$$k^2\gamma^2 < \|\mathbf{w}(k+1)\|^2 \leq kR^2 \tag{4.40}$$

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, adică 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 $\mathscr{C}_2$.

4.6 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".

4.7 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*[3] 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[4] au propus o procedură de învățare pentru rețele

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

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

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 4.4.

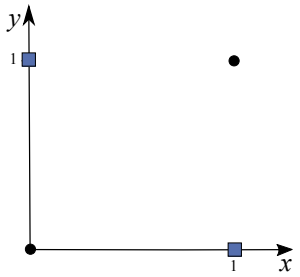


Figura 4.4: Problema XOR. Clasele sunt marcate cu forme și culori diferite. Se poate demonstra că nu se poate trasa o dreaptă în plan care să aibă de o parte a ei doar puncte din clasa "0" și de cealaltă parte doar puncte de clasă "1".

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

$$sgn(w_0\cdot 1+w_1\cdot 0+w_2\cdot 0)=sgn(w_0\cdot 1+w_1\cdot 1+w_2\cdot 1)=1 \tag{4.41}$$

iar

$$sgn(w_0\cdot 1+w_1\cdot 0+w_2\cdot 1)=sgn(w_0\cdot 1+w_1\cdot 1+w_2\cdot 0)=-1 \tag{4.42}$$

Incompatibilitatea sistemului de ecuații (4.41, 4.42) se arată ușor: să presupunem că ar exista w_0, w_1, w_2 care să satisfacă (4.41, 4.42), pe care le rescriem echivalent:

$$w_0\cdot 1+w_1\cdot 0+w_2\cdot 0>0 \tag{4.43a}$$
$$w_0\cdot 1+w_1\cdot 1+w_2\cdot 1>0 \tag{4.43b}$$
$$w_0\cdot 1+w_1\cdot 0+w_2\cdot 1<0 \tag{4.43c}$$
$$w_0\cdot 1+w_1\cdot 1+w_2\cdot 0<0 \tag{4.43d}$$

Grupând primele și ultimele două inecuații și adunând, obținem:

$$2\cdot w_0+w_1+w_2>0 \tag{4.44a}$$
$$2\cdot w_0+w_1+w_2<0 \tag{4.44b}$$

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 5

Perceptronii multistrat

Rețelele neurale multistrat — sau perceptronii multistrat, multilayer perceptrons (MLPs) — 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.

5.1 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 probleme de interes care nu sunt liniar separabile — și deci nerezolvabile de către perceptronul liniar — dar care pot fi totuși separate. În plus, dorim să rezolvăm și altfel de probleme decât de clasificare binară: regresie (estimare de valoare de ieșire de tip continuu), estimare de probabilitate condiționată, clasificare pentru mai mult de două clase. Cursul de față conține modele bazate pe neuroni cu funcții de activare neliniare 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 5.1.

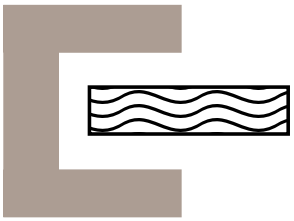


Figura 5.1: Două clase de puncte ce nu sunt liniar separabile

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.

5.2 Notații folosite

Tabelul 5.1 conține notațiile care se folosesc în acest curs.

5.3 Setul de instruire

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

$$\mathscr{S} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)} \right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)} \right), \ldots, \left(\mathbf{x}^{(p)}, \mathbf{y}^{(p)} \right) \right\} \tag{5.1}$$

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

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

5.4 Rețeaua neurală multistrat

5.4.1 Arhitectură

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

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

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

[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.

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, $0 \leq l \leq L-1$;
$z_i^{[l]}$	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 $(\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^{L-1})$
$J(\mathbf{W}, \mathbf{b})$	eroare empirică medie pentru set de vectori cu etichete cunoscute
$J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$	eroarea pentru perechea de vectori $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$
$\hat{\mathbf{y}}^{(i)}$	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 5.1: Notații folosite în acest curs

- 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.

Numerotarea straturilor începe 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 5.3.

Se consideră că 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 5.2; o rețea cu 4 straturi și două ieșiri este reprezentată în figura 5.3. 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 5.2 și 5.3:

- 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 *bias*–ului;
- valoarea $a_i^{[l]}$ este ieșirea nodului i din stratul l, $0 \leq l \leq L-1$, $1 \leq i \leq n_l$; se remarcă în figuri prezența valorilor constante +1 în toate straturile, exceptând cel de ieșire. Pentru stratul de intrare, $a_i^{[0]} = x_i$;
- 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 5.2; $h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \in \mathbb{R}^2$ pentru figura 5.3. Rețeaua din figura 5.2 poate fi folosită pentru estimarea unei valori continue (problemă de regresie) sau pentru estimare de probabilitate condiționată pentru două clase. Rețeaua din figura 5.3 se poate folosi pentru estimarea a două valori de ieșire, numere reale – problemă de regresie.

[3] Unii autori consideră valoarea constantă -1; nu este esențial, deoarece ponderile sunt coeficienți ce 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

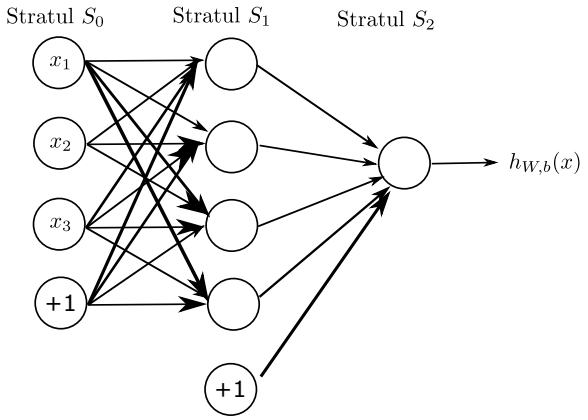


Figura 5.2: Rețea MLP cu 3 straturi. Poate fi folosită pentru estimarea unei valori continue (problemă de regresie) sau pentru discriminare/estimare de probabilitate condiționată pentru două clase.

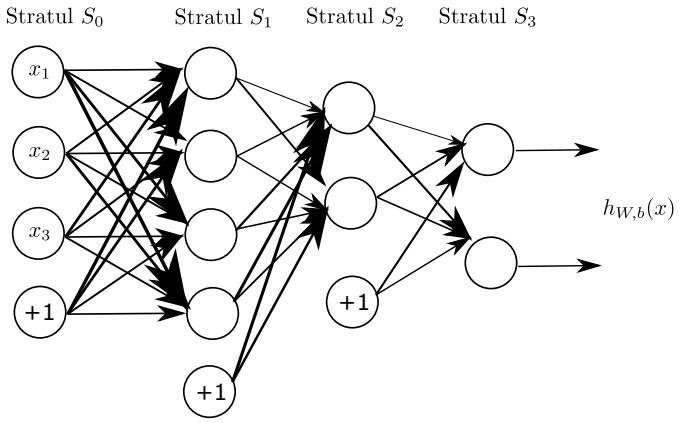
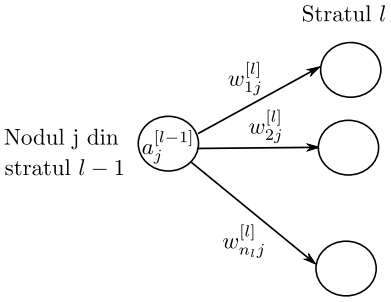
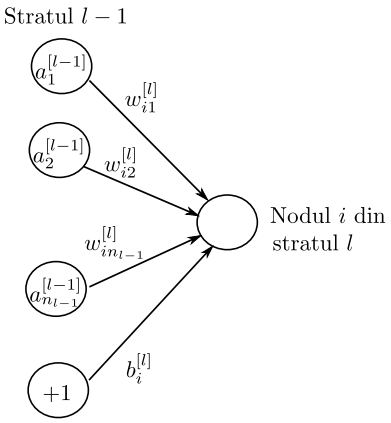


Figura 5.3: Rețea MLP cu 4 straturi ce se poate folosi pentru estimarea a două valori de ieșire.



(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.

Perechea $(\mathbf{W}, \mathbf{b})$ formează mulțimea ponderilor și a valorilor 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 al doilea (de indice 1) și nodul j din stratul de intrare; se remarcă ordinea indicilor inferiori, utilă mai departe pentru operațiile de algebră liniară ce vor fi 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 5.4a, 5.4b;
- valoarea ponderii dintre intrarea constantă $+1$ din stratul de intrare și neuronul i din primul strat ascuns este $b_i^{[1]}$, $1 \leq i \leq n_2$;
- coeficientul de bias provenind din stratul $l-1$ $(1 \leq l \leq L-1)$ și pentru neuronul i din stratul l este notat cu $b_i^{[l]}$, $1 \leq i \leq n_l$, a se vedea figura 5.4b.

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

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

5.4.2 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:

$$z_i^{[l]} = w_{i1}^{[l]} \cdot a_1^{[l-1]} + w_{i2}^{[l]} \cdot a_2^{[l-1]} + \cdots + w_{i,n_{l-1}}^{[l]} \cdot a_{n_{l-1}}^{[l-1]} + b_i^{[l]} \tag{5.2}$$

$$= \mathbf{W}_i^{[l]} \cdot \mathbf{a}^{[l-1]} + b_i^{[l]},\ 1 \leq i \leq n_l \tag{5.3}$$

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:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]},\ 1 \leq l \leq L-1 \tag{5.4}$$

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

$$a_i^{[l]} = f^{[l]}\left(z_i^{[l]}\right) \tag{5.5}$$

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, de exemplu prin vectorizare – atunci putem scrie mai compact ecuația (5.5) sub forma:

$$\mathbf{a}^{[l]} = f^{[l]}\left(\mathbf{z}^{[l]}\right) \tag{5.6}$$

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:**

$$f = \sigma : \mathbb{R} \to (0,1), f(z) = \sigma(z) = \frac{1}{1+\exp(-z)} \tag{5.7}$$

Derivata acestei funcții este:

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z)) \tag{5.8}$$

2. **funcția tangentă hiperbolică**:

$$f = \tanh : \mathbb{R} \to (-1,1), f(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \tag{5.9}$$

a cărei derivată este:

$$tanh'(z) = 1 - tanh^2(z) \tag{5.10}$$

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

$$\tanh(z) = 2 \cdot \sigma(2z) - 1 \tag{5.11}$$

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

3. **funcția liniară:**

$$f(z) = a \cdot z + b \tag{5.12}$$

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

4. **funcția softmax:**

$$softmax(\mathbf{z}; i) = \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)} \tag{5.13}$$

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:

$$\frac{\partial softmax(\mathbf{z}; i)}{\partial z_j} = \begin{cases} softmax(\mathbf{z}; i) \cdot (1 - softmax(\mathbf{z}; i)) & \text{dacă } i = j \\ -softmax(\mathbf{z}; i) \cdot softmax(\mathbf{z}; j) & \text{dacă } i \neq j \end{cases} \tag{5.14}$$

Putem folosi funcția delta al lui Kronecker: $\delta_{ij} = 1$ dacă $i = j$ și 0 altfel și rescriem (5.14) ca:

$$\frac{\partial softmax(\mathbf{z}; i)}{\partial z_j} = softmax(\mathbf{z}; i) \cdot (\delta_{ij} - softmax(\mathbf{z}; j)) \tag{5.15}$$

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

$$ReLU(z) = \max(0, z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{5.16}$$

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 5.5.

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 0. Avem deci:

$$ReLU'(z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ 1 & \text{dacă } z > 0 \end{cases} \tag{5.17}$$

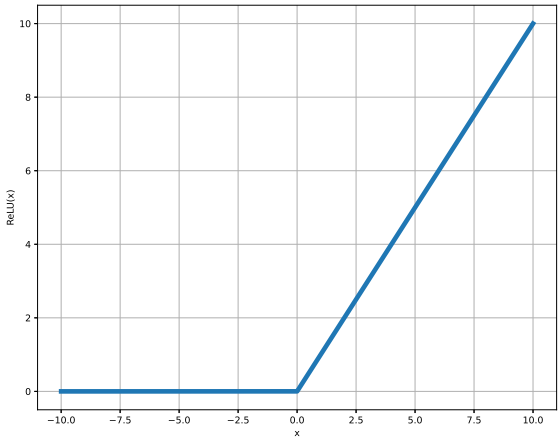


Figura 5.5: Graficul funcției de activare ReLU

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

$$PReLU(z) = \begin{cases} \alpha z & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{5.18}$$

unde $\alpha > 0$. Graficul funcției pentru $\alpha = 0.1$ este dat în figura 5.6. Derivatele pe subintervale sunt ușor de calculat.

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

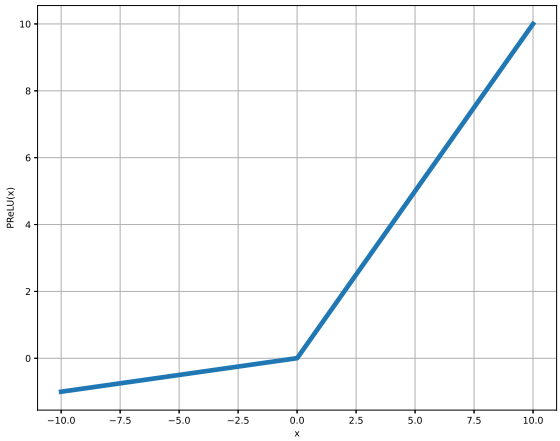


Figura 5.6: Graficul funcției de activare PReLU pentru $\alpha = 0.1$

7. **funcția Swish**:

$$Swish(z) = z \cdot \sigma(\beta z) = \frac{z}{1 + \exp(-\beta z)} \tag{5.19}$$

unde σ este logistica sigmoidă, β este parametru constant sau antrenabil. Se arată[5] că pentru rețele cu foarte multe straturi, ea tinde să funcționeze mai bine decât ReLU. Reprezentarea ei este dată în figura 5.7. Derivata ei este:

$$Swish'(z) = \sigma(\beta z)(1 + \beta z - \beta z \sigma(\beta z))) \tag{5.20}$$

Lista funcțiilor de activare de mai sus nu e exhaustivă. 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.

5.5 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 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 $1, \ldots, L-1$.

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

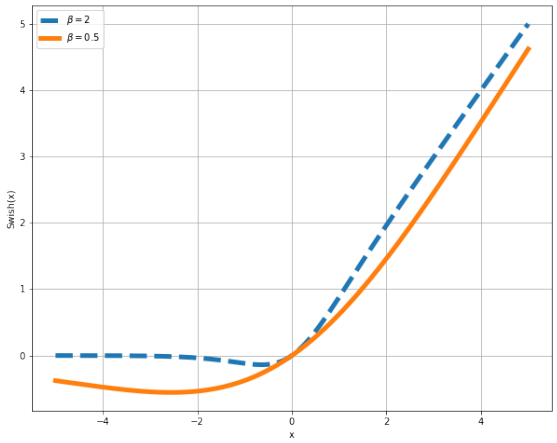


Figura 5.7: Graficul funcției de activare Swish pentru $\beta \in \{0.5, 2\}$

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:

$$\mathbf{a}^{[0]} = \mathbf{x} \tag{5.21}$$

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:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \tag{5.22}$$

$$\mathbf{a}^{[l]} = f^{[l]}(\mathbf{z}^{[l]}) \tag{5.23}$$

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:

$$\hat{\mathbf{y}} = \mathbf{a}^{[L-1]} \tag{5.24}$$

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 stivui pe orizontală într–o matrice $\mathbf{X}$. De exemplu, datele din setul de instruire $\mathscr{S} = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), \ldots, \left(\mathbf{x}^{(p)}, \mathbf{y}^{(p)}\right)\right\}$, matricea $\mathbf{X}$ se formează ca:

$$\mathbf{X} = \begin{pmatrix} — & \mathbf{x}^{(1)t} & — \\ \cdot & \cdot & \cdot \\ — & \mathbf{x}^{(p)t} & — \end{pmatrix} \tag{5.25}$$

În cele ce urmează, se poate să considerăm $\mathbf{X}$ 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 (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 linii. Rețeaua transmite această rețea mai departe prin straturile sale, succesiv:

$$\mathbf{Z}^{[1]} = \mathbf{X} \cdot \mathbf{W}^{[1]t} + \mathbf{b}^{[1]t} = \mathbf{A}^{[0]} \cdot \mathbf{W}^{[1]t} + \mathbf{b}^{[1]t} \tag{5.26}$$

$$\mathbf{A}^{[1]} = f^{[1]}\left(\mathbf{Z}^{[1]}\right) \tag{5.27}$$

$$\mathbf{Z}^{[2]} = \mathbf{A}^{[1]} \cdot \mathbf{W}^{[2]t} + \mathbf{b}^{[2]t} \tag{5.28}$$

$$\mathbf{A}^{[2]} = f^{[2]}\left(\mathbf{Z}^{[2]}\right) \tag{5.29}$$

$$\ldots \tag{5.30}$$

$$\mathbf{Z}^{[L-1]} = \mathbf{A}^{[L-2]} \cdot \mathbf{W}^{[L-1]t} + \mathbf{b}^{[L-1]t} \tag{5.31}$$

$$\mathbf{A}^{[L-1]} = f^{[L-1]}\left(\mathbf{Z}^{[L-1]}\right) \tag{5.32}$$

$\mathbf{Z}^{[l]}$ și $\mathbf{A}^{[l]}$ sunt matrice cu k linii și n_l coloane. Pentru adunările din (5.26, 5.28, 5.31) se consideră că se aplică mecanismul de "broadcasting": vectorii linie $\mathbf{b}^{[l]t}$ produc prin copiere matrice cu același număr de linii ca $\mathbf{X}$.

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.

5.6 Funcții de cost

O pereche $(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m)$ va produce valoare pentru funcția 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 factor de regularizare care împiedică ponderile să devină prea mari în valoare absolută, caz asociat de regulă cu un comportament instabil al rețelei: variații mici ale intrării duc la salturi mari 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 stivuiți orizontal în matricea $\mathbf{X}$ de forma:

$$\mathbf{X} = \begin{pmatrix} — & \mathbf{x}^{(1)t} & — \\ \cdot & \cdot & \cdot \\ — & \mathbf{x}^{(k)t} & — \end{pmatrix} \tag{5.33}$$

unde pentru fiecare vector coloană $\mathbf{x}^{(i)}$ avem o etichetă $\mathbf{y}^{(i)}$ asociată (eng. ground truth). Setul de date peste care se calculează funcția de eroare este eșantion (parte) din setul de antrenare $\mathscr{S}$ sau tot $\mathscr{S}$, eșantion din setul de testare etc. Esențial este ca datele să fie etichetate, adică valorile de ieșire $\mathbf{y}$ asociate vectorilor de intrare să fie cunoscute.

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

$$J(\mathbf{W},\mathbf{b}) = \underbrace{\left[\frac{1}{k}\sum_{i=1}^{k} \overbrace{J\left(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{y}^{(i)}\right)}^{\text{eroare empirică pentru}\left(\mathbf{x}^{(i)},\mathbf{y}^{(i)}\right)}\right]}_{\text{Eroarea empirică medie pe setul } \mathbf{X}} + \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_l}\sum_{j=1}^{n_{l-1}}\left(w_{ij}^{[l]}\right)^2}_{\text{Factor de regularizare}} \tag{5.34}$$

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

$$\left\|\mathbf{W}^{[l]}\right\|_F^2 \overset{\text{def}}{=} \sum_{i=1}^{n_l}\sum_{j=1}^{n_{l-1}}\left(w_{ij}^{[l]}\right)^2 \tag{5.35}$$

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 Fromenius, factorul de regularizare din (5.34) se scrie mai succint ca:

$$\frac{\lambda}{2}\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2$$

5.6.1 Funcția de cost pentru problemă de regresie

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

$$J\left(\mathbf{W},\mathbf{b};\mathbf{x},\mathbf{y}\right) = \frac{1}{2}\cdot\left\|\mathbf{y}-\hat{\mathbf{y}}\right\|^2 \tag{5.36}$$

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

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

În acest caz funcția de eroare pentru un set de date cu k perechi $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$ și incluzând termen de regularizare devine:

$$J(\mathbf{W}, \mathbf{b}) = \left[\frac{1}{2k}\sum_{i=1}^{k}\left\|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right\|^2\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2 \tag{5.37}$$

Termenul $\frac{1}{k}\sum_{i=1}^{k}\left\|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right\|^2$ 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 $\|\mathbf{y} - \hat{\mathbf{y}}\| = 10$ ca la 100 de cazuri pentru care $\|\mathbf{y} - \hat{\mathbf{y}}\| = 1$. Se tinde deci a se preocupa de cazuri ce produc erori mari, dar puține, în detrimentul unor cazuri frecvente cu erori mai mici. Altfel zis, funcția de eroare este dominată de valorile de eroare extreme.

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

$$J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}\right) = \frac{1}{k}\|\mathbf{y} - \hat{\mathbf{y}}\|_1 = \frac{1}{k}\sum_{i=1}^{k}|y_i - \hat{y}_i| \tag{5.38}$$

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 $(\mathbf{y}, \hat{\mathbf{y}})$:

$$HL(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} \frac{1}{2}\|\mathbf{y} - \hat{\mathbf{y}}\|^2 & \text{dacă } \|\mathbf{y} - \hat{\mathbf{y}}\| \leq \delta \\ \delta(\|\mathbf{y} - \hat{\mathbf{y}}\|_1 - \frac{\delta}{2}) & \text{dacă } \|\mathbf{y} - \hat{\mathbf{y}}\| > \delta \end{cases} \tag{5.39}$$

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, spre deosebire de L_1.

Reprezentările MAE, MSE si Huber pentru o diferență $\|\mathbf{y} - \hat{\mathbf{y}}\|$ fixată sunt date în figura 5.8.

5.6.2 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

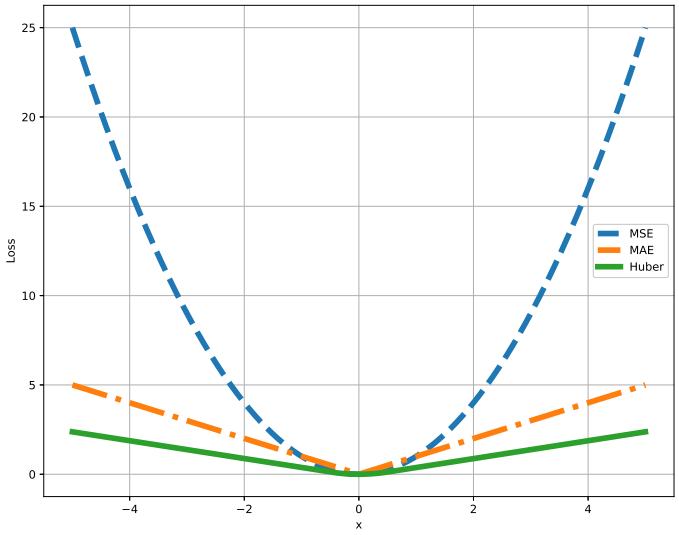


Figura 5.8: Functiile mean squared error, mean absolute error, Huber ($\delta = 0.5$) pentru cazul unidimensional

pereche $(\mathbf{x}, y)$, $J(\mathbf{W}, \mathbf{b}; \mathbf{x}, y)$ este[6]:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, y) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y}) \tag{5.40}$$

deci funcția de eroare totală va fi:

$$J(\mathbf{W}, \mathbf{b}) = -\sum_{i=1}^{k} \left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln \left(1 - \hat{y}^{(i)}\right) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{5.41}$$

Pentru forma vectorizată a erorii empirice medii (prima sumă din ecuația de mai sus) a se vedea ecuația (3.29), pagina 41.

5.6.3 Funcția de cost pentru $m > 2$ clase independente

Dacă în stratul de ieșire se folosesc funcții de activare sigmoidă logistică, iar ieșirile sunt independente unele de altele, atunci funcția de eroare pentru o singură pereche $(\mathbf{x}, \mathbf{y})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{m} \left[y_j \ln \hat{y}_j + (1 - y_j) \ln (1 - \hat{y}_j) \right] \tag{5.42}$$

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

unde pentru vectorul $\mathbf{y} = (y_1, y_2, \ldots, y_m)^t$ se folosește codificarea one–hot, a se vedea pagina 44. În acest fel, funcția totală de eroare $J(\mathbf{W}, \mathbf{b})$ calculată pentru setul de instruire devine:

$$J(\mathbf{W}, \mathbf{b}) = -\frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} \left[y_j^{(i)} \ln \hat{y}_j^{(i)} + \left(1 - y_j^{(i)}\right) \ln \left(1 - \hat{y}_j^{(i)}\right) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{5.43}$$

5.6.4 Funcția de cost pentru clasificare cu $m > 2$ 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 $(\mathbf{x}, \mathbf{y})$, $J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{m} y_j \log \hat{y}_j \tag{5.44}$$

unde folosim, ca mai sus, codificarea one–hot. Eroarea totală este:

$$J(\mathbf{W}, \mathbf{b}) = -\frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{m} y_j^{(i)} \log \hat{y}_j^{(i)} + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{5.45}$$

Pentru forma vectorizată a erorii empirice (suma compusă din 5.45) se poate vedea ecuația (3.46), pagina 45.

5.7 Inițializarea ponderilor rețelei

Valorile inițiale ale ponderilor $\mathbf{W}$ și $\mathbf{b}$ sunt setate aleator, în jurul lui zero. Este necesar ca valorile ponderilor să nu fie toate egale; dacă ar fi toate egale, fiecare neuron ar avea exact aceeași stare, deoarece: fiecare neuron e legat la exact aceleași intrări ca și ceilați din stratul său; mai departe, dacă ponderile cu care se înmulțesc intrările sunt egale, atunci valoarea de activare a fiecărui neuron de pe acel strat e aceeași (ponderea constantă folosită se dă factor comun); argumentul 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 $\mathbf{W}$ se elimină prin inițializare cu numere aleatoare. În ce privește ponderile de bias – din $\mathbf{b}$ – ele se pot inițializa cu 0; inițializarea aleatoare a ponderilor $\mathbf{W}$ este suficientă pentru "spargerea simetriei".

Strategii rafinate de ințializare pentru ponderile sunt cele propuse de Xavier Glorot *et al.*[7] și He *et al.*[8].

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

[8] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-

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 (o problemă) similară cu cea curentă – transfer de învățare (transfer learning).

5.8 Derivate parțiale de funcții compuse

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

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:

$$\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} \tag{5.46}$$

Pentru funcții multivariate $f : \mathbb{R}^n \to \mathbb{R}$, $g, h : \mathbb{R}^m \to \mathbb{R}$ de forma[9]:

$$h(x_1, \ldots, x_m) = f\left(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m)\right) \tag{5.47}$$

derivatele parțiale se calculează astfel:

$$\frac{\partial h}{\partial x_i} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x_i} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x_i} + \cdots + \frac{\partial f}{\partial g_n} \cdot \frac{\partial g_n}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial g_j} \cdot \frac{\partial g_j}{\partial x_i} \tag{5.48}$$

Exemplificăm și figurăm în cele ce urmează pentru funcția $f : \mathbb{R}^3 \to \mathbb{R}$, $f(x, y, z) = x \cdot (y + z)$. Notăm cu q funcția $q(y, z) = y + z$, deci $f(x, y, z) = x \cdot q(y, z)$. Dorim să calculăm derivatele parțiale ale lui f în raport cu x, y, z.

Întrucât q nu depinde de x, avem:

$$\frac{\partial f}{\partial x} = \frac{\partial (x \cdot q(y, z))}{\partial x} = q(y, z) = y + z \tag{5.49}$$

Pentru calculul lui $\partial f / \partial y$, avem că f depinde indirect de y prin q, deci vom aplica regula de derivare a funcțiilor compuse:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} \tag{5.50}$$

Calculăm pe rând cele două derivate parțiale și obținem:

$$\frac{\partial f}{\partial q} = \frac{\partial (x \cdot q)}{\partial q} = x \tag{5.51}$$

Level Performance on ImageNet Classification," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026-1034, doi: 10.1109/ICCV.2015.123.

[9] Numerele m și n din definiția lui h nu au aici vreo legătură cu dimensiunea vectorilor de intrare și ieșire din setul de instruire.

și respectiv

$$\frac{\partial q}{\partial y} = \frac{\partial (y+z)}{\partial y} = 1 \tag{5.52}$$

deci

$$\frac{\partial f}{\partial y} = x \cdot 1 = x \tag{5.53}$$

Similar, pentru derivata parțială $\partial f/\partial z$ avem:

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial z} = x \cdot 1 = x \tag{5.54}$$

Următoarele două observații sunt esențiale pentru algoritmul backpropagation: deoarece f depinde de y și de z prin intermediul lui q, avem că:

- derivata parțială $\partial f/\partial q$ trebuie calculată ca prim pas;
- ea este folosită în mod repetat, pentru calculul fiecăreia din derivatele $\partial f/\partial y$, $\partial f/\partial z$: se înmulțește cu gradienții locali $\partial q/\partial y$ și $\partial q/\partial z$.

Fixăm $x = 3, y = 4, z = 5$. Calculul valorilor q, f și al derivatelor parțiale este reprezentat în graful computațional figura 5.9. 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, sensul este figurat prin săgețile de sub formule.

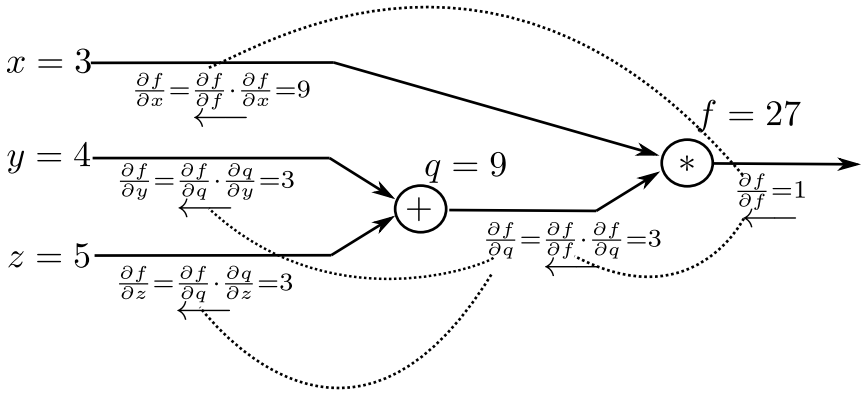


Figura 5.9: Graful computațional pentru calculul valorii funcției $f(x, y, z) = x \cdot (y + z)$ și al derivatelor parțiale $\partial f/\partial x$, $\partial f/\partial y$, $\partial f/\partial z$, pentru valorile fixe $x = 3, y = 4, z = 5$. Valorile lui q și f se calculează de la stânga spre dreapta, prin propagare înainte, sens dat de săgețile negre. Derivatele parțiale se calculează de la dreapta spre stânga, sensul este figurat prin săgețile de sub formule. Utilizarea repetată a unor derivate parțiale "din amonte" este arătată prin linie punctată.

Cel mai important lucru reprezentat în figura 5.9, și care nu apare explicit în calculele 5.49—5.54, este faptul că pentru calculul derivatei parțiale a lui f la cantitatea dintr–un nod – x, y, z, q – se folosește gradientul calculat la dreapta acelui nod ("din amonte"[10]) și se înmulțește cu gradientul local. De exemplu, pentru calculul lui $\partial f/\partial z$ se înmulțește gradientul "din amonte" $\partial f/\partial q$ cu gradientul local $\partial q/\partial z$. Acest principiu explică de ce am scris explicit ultimul gradient din graful computațional $\partial f/\partial f$, care e tot timpul 1, indiferent de forma lui f: el se folosește pentru înmulțirea cu gradienții locali $\partial f/\partial x$ și $\partial f/\partial q$. Principiul din figura 5.9 stă la baza unor algoritmi de calcul automat al gradienților – bibliotecile de tip *autograd*.

5.9 Algoritmul backpropagation pentru perceptronul multistrat

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; dacă eroarea scade, 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:

$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{5.55}$$

Ponderile de bias $b_{ij}^{[l]}$ se modifică similar:

$$b_{i}^{[l]} = b_{i}^{[l]} - \alpha \frac{\partial J}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{5.56}$$

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.

Avem trei variante de lucru 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**: se calculează gradienții pentru ponderile $w_{ij}^{[l]}$ și bias–urile $b_{i}^{[l]}$ pentru fiecare pereche de vectori din setul de instruire; la

[10] Eng: upstream gradient. Sensul de "curgere" este dat de ordinea de calcul a gradienților: calculul începe de la ieșirea din graful computațional.

final se calculează media tuturor acestor gradienți 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:** 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 200;
- 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 5.10;
- 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) și ponderile nu vor mai fi modificate semnificativ;
- valoarea funcției de eroare scade sub un anumit prag; etc.

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.

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

$$\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p} \sum_{k=1}^{p} \frac{\partial J}{\partial w_{ij}^{[l]}} \left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right) + \lambda w_{ij}^{[l]} \tag{5.57}$$

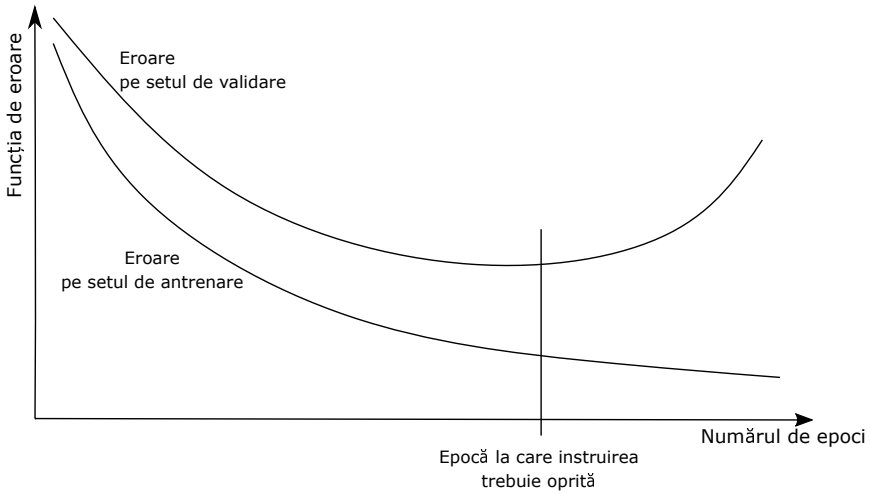


Figura 5.10: Evoluția valorilor funcției de eroare pentru setul de antrenare, respectiv cel de validare. Dacă se continuă antrenarea, eroare pe setul de antrenare scade, dar pentru setul de testare începe să crească de la o anumită epocă. Se recomandă oprirea instruirii dacă eroarea pe setul de validare începe să crească.

respectiv pentru ponderile de bias

$$\frac{\partial J}{\partial b_i^{[l]}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p} \sum_{k=1}^{p} \frac{\partial J}{\partial b_i^{[l]}} \left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{y}^{(k)} \right) \tag{5.58}$$

deci este de interes calculul derivatelor parțiale pentru o singură pereche de instruire $(\mathbf{x}, \mathbf{y})$, respectiv $\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$ 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, asemănător cu ce s–a prezentat în secțiunea 5.8. Odată calculate, ele pot fi folosite pentru modificarea ponderilor (învățare). Algoritmul a fost introdus în articolul "Learning representations by back-propagating errors"[11].

Algoritmul funcționează astfel: pentru o pereche de instruire $(\mathbf{x}, \mathbf{y})$ se face pasul de propagare înainte și se obține vectorul de ieșire $\hat{\mathbf{y}}$; pentru fiecare strat de neuroni l, începând de la ultimul, se calculează un termen de eroare $\boldsymbol{\delta}^{[l]} = \frac{\partial J}{\partial \mathbf{z}^{[l]}}$ care măsoară cât de mult e "responsabil" stratul l (și deci fiecare neuron din stratul l) pentru discrepanța dintre ieșirea $\hat{\mathbf{y}}$ și valoarea dorită $\mathbf{y}$. În decursul pasului de propagare înainte ponderile nu se modifică.

Înainte de detalia algoritmul de instruire a rețelei MLP, definim produsul Hadamard a două matrice; produsul se notează de regula cu $\odot$ și se aplică pentru două matrice care au același număr de linii și respectiv același număr de coloane: dacă $A = (a_{ij})_{1 \leq i \leq q, 1 \leq j \leq r}$ și $B = (b_{ij})_{1 \leq i \leq q, 1 \leq j \leq r}$ sunt cele două

[11] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

matrice, atunci matricea produs Hadamard $C = A \odot B = (c_{ij})_{1 \leq i \leq q, 1 \leq j \leq r}$ are elementele:

$$c_{ij} = a_{ij} \cdot b_{ij} \tag{5.59}$$

Algoritmul backpropagation detaliat – varianta batch – este:

1. Inițializează $\Delta \mathbf{W}^{[l]}$, $\Delta \mathbf{b}^{[l]}$ cu 0, pentru $l = 1, \ldots, L-1$:

$$\Delta \mathbf{W}^{[l]} = \mathbf{0}_{n_l \times n_{l-1}} \tag{5.60}$$

$$\Delta \mathbf{b}^{[l]} = \mathbf{0}_{n_l}, \text{vector coloană} \tag{5.61}$$

2. Pentru fiecare din cele p perechi $(\mathbf{x}, \mathbf{y})$ din batch calculează corecția pentru ponderi și ponderile de bias[12]:

   2.1. Efectuează pasul de propagare înainte, conform secțiunii 5.5, 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

$$\boldsymbol{\delta}^{[L-1]} = \nabla_a J(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) \odot f^{[L-1]'}\left(\mathbf{z}^{[L-1]}\right) \tag{5.62}$$

   iar la restul de straturi

$$\boldsymbol{\delta}^{[l]} = \left[\left(\mathbf{W}^{[l+1]}\right)^t \cdot \boldsymbol{\delta}^{[l+1]}\right] \odot f^{[l]'}\left(\mathbf{z}^{[l]}\right) \tag{5.63}$$

   Am presupus mai sus că derivatele funcțiilor de activare se vectorizează peste vectorii de stări.

   Putem acum calcula derivatele parțiale pentru ponderi și derivatele parțiale:

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = \boldsymbol{\delta}^{[l]} \cdot \left(\mathbf{a}^{[l-1]}\right)^t \tag{5.64}$$

$$\frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) = \boldsymbol{\delta}^{[l]} \tag{5.65}$$

   2.3. Acumulează semnalul de corecție[13], pentru $l = L-1, \ldots, 1$:

$$\Delta \mathbf{W}^{[l]} = \Delta \mathbf{W}^{[l]} + \frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) \tag{5.66}$$

$$= \Delta \mathbf{W}^{[l]} + \boldsymbol{\delta}^{[l]} \cdot \left(\mathbf{a}^{[l-1]}\right)^t \tag{5.67}$$

$$\Delta \mathbf{b}^{[l]} = \Delta \mathbf{b}^{[l]} + \frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}) \tag{5.68}$$

$$= \Delta \mathbf{b}^{[l]} + \boldsymbol{\delta}^{[l]} \tag{5.69}$$

[12] Respectiv: se iterează peste datele din minibatch.

[13] Evident, acumularea se poate face imediat după calculul derivatelor parțiale de pe stratul curent.

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

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \left[\left(\frac{1}{p} \Delta \mathbf{W}^{[l]} \right) + \lambda \mathbf{W}^{[l]} \right] \tag{5.70}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \left(\frac{1}{p} \Delta \mathbf{b}^{[l]} \right) \tag{5.71}$$

4. Ciclare: se repetă de la pasul 1 până când se îndeplinește una din condițiile de oprire de la pagina 83.

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 $(\mathbf{x}, \mathbf{y})$:

$$\boldsymbol{\delta}^{[L-1]} = \mathbf{a}^{[L-1]} - \mathbf{y} \tag{5.72}$$

substituind formula de calcul (5.62). 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].

Dacă problema rezolvată este una de regresie, atunci funcția de eroare este bazată pe mean squared error. În acest caz, pentru perechea $(\mathbf{x}, \mathbf{y})$:

$$\boldsymbol{\delta}^{[L-1]} = (\mathbf{a}^{[L-1]} - \mathbf{y}) \odot f^{[L-1]'}(\mathbf{z}^{L-1}) \tag{5.73}$$

5.10 Justificarea matematică a algoritmului de back-propagation

Figura 5.11 prezintă, pentru o rețea neurală cu un strat ascuns, propagarea înainte și înapoi prin rețea. Propagarea înainte preia o pereche de vectori $(\mathbf{x}, \hat{\mathbf{y}})$. Se calculează succesiv:

- vectorul de stări pentru neuronii din stratul ascuns:

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \tag{5.74}$$

- vectorul de valori de ieșire din stratul ascuns:

$$\mathbf{a}^{[1]} = f^{[1]}(\mathbf{z}^{[1]}) \tag{5.75}$$

- vectorul de stări pentru neuronii din stratul de ieșire:

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \tag{5.76}$$

- vectorul de valori de ieșire din rețea:

$$\hat{\mathbf{y}} = \mathbf{a}^{[2]} = f^{[2]}(\mathbf{z}^{[2]}) \tag{5.77}$$

- Valoarea de eroare $J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y}\right)$, notată în figura 5.11 cu $J(\mathbf{a}^{[2]}, \mathbf{y})$.

Avem nevoie de derivatele parțiale:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}}, \frac{\partial J}{\partial \mathbf{b}^{[2]}}, \frac{\partial J}{\partial \mathbf{W}^{[1]}}, \frac{\partial J}{\partial \mathbf{b}^{[1]}} \tag{5.78}$$

pentru a face modificarea ponderilor.

Conform algoritmului backpropagation, vom calcula gradienții pornind de la ultimul strat. Folosim formulele de derivare a funcțiilor compuse din secțiunea 5.8 pagina 80.

Pentru calculul derivatelor parțiale $\frac{\partial J}{\partial \mathbf{W}^{[2]}}$ avem:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} \tag{5.79}$$

Derivata parțială $\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$ se calculează simplu, ținând cont de forma liniară din ec. (5.76):

$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} = \mathbf{a}^{[1]} \tag{5.80}$$

deci (5.79) devine:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}} \mathbf{a}^{[1]} \tag{5.81}$$

Pentru derivata parțială $\frac{\partial J}{\partial \mathbf{b}^{[2]}}$ avem:

$$\frac{\partial J}{\partial \mathbf{b}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}} \tag{5.82}$$

unde al doilea termen din partea dreaptă se calculează simplu, folosind definiția din ec. (5.76):

$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}} = \mathbf{1} \tag{5.83}$$

unde $\mathbf{1}$ este vector coloană cu același număr de elemente ca și vectorul $\mathbf{b}^{[2]}$; (5.82) devine:

$$\frac{\partial J}{\partial \mathbf{b}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}} \tag{5.84}$$

Observăm că în ambele ecuații (5.81) și (5.84) apare cantitatea $\frac{\partial J}{\partial \mathbf{z}^{[2]}}$ care se calculează simplu, astfel:

$$\frac{\partial J}{\partial \mathbf{z}^{[2]}} = \frac{\partial J}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = \frac{\partial J}{\partial \mathbf{a}^{[2]}} \cdot f^{(2)'}(\mathbf{z}^{[2]}) \tag{5.85}$$

Termenul $\frac{\partial J}{\partial \mathbf{z}^{[2]}}$ este semnalul de eroare – numit uneori si gradient local – pentru stratul 2. Termenul $\frac{\partial J}{\partial \mathbf{a}^{[2]}}$, notat cu $da^{[2]}$ în figura 5.11 depinde de forma concretă a funcției de eroare J.

Pentru derivatele parțiale după $\mathbf{W}^{[1]}$ și $\mathbf{b}^{[1]}$ procedăm similar:

$$\frac{\partial J}{\partial \mathbf{W}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \mathbf{x} \tag{5.86}$$

respectiv

$$\frac{\partial J}{\partial \mathbf{b}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \tag{5.87}$$

unde semnalul de eroare $\frac{\partial J}{\partial \mathbf{z}^{[1]}}$ pentru stratul 1 este

$$\frac{\partial J}{\partial \mathbf{z}^{[1]}} = \frac{\partial J}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} = \frac{\partial J}{\partial \mathbf{a}^{[1]}} \cdot f^{[1]'} \left(z^{[1]} \right) \tag{5.88}$$

5.11 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:

$$\hat{y}_i = P\left(\text{clasa } i | \mathbf{x}\right), 1 \leq i \leq m \tag{5.89}$$

atunci clasificarea se face găsind acel indice i pentru care $\hat{y}_i$ e maxim și acesta este indicele clasei prezise de rețeaua MLP.

5.12 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

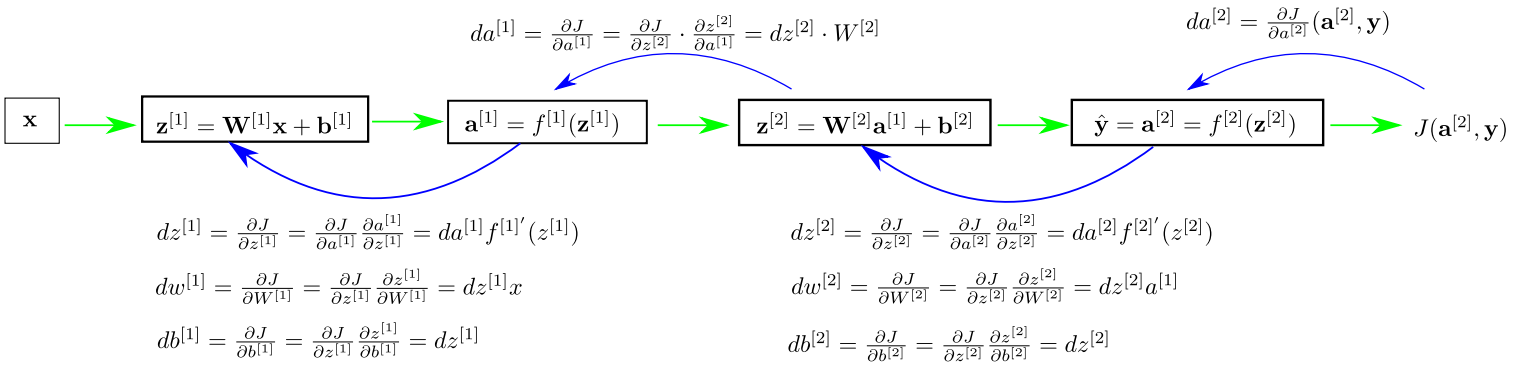


Figura 5.11: Reprezentarea procesului de propagare înainte (arcele verzi) și înapoi (arcele albastre). Valoarea de eroare $J\left(\mathbf{a}^{[2]}, \mathbf{y}\right)$ este asociată unei perechi $(\mathbf{x}, \mathbf{y})$.

Bibliografie

[1] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[2] R. Andonie and A. Cațaron, *Inteligenţă computaţională*. Universitatea Transilvania din Braşov, 2002. http://vega.unitbv.ro/~cataron/Publications/curs_rn.pdf.

[3] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. Wiley, 2002.

[4] R. C. Eberhart and Y. Shi, *Computational intelligence - concepts to implementations*. Elsevier, 2007.

[5] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," 2008. Version 20081110.

[6] "Stanford Machine Learning, curs online Coursera." https://www.coursera.org/learn/machine-learning, 2019. Accesat: 2021-02-24.

[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.