7. Antrenarea reţelelor neuronale

Funcția de cost
Metoda propagării înapoi a erorii
Aplicație

7.1 Funcția de cost pentru o rețea neuronală

În acest curs se vor folosi următoarele notaţii:

- L - numărul total de straturi din rețea;
- s_l - numărul de unități din stratul l;
- K - numărul de unități de ieșire (clase).

Notăm cu $h_\theta(x)_k$ o ipoteză care rezultă din ieșirea k. Funcția de cost pentru rețeaua neuronală este o generalizare a funcției folosite pentru regresia logistică:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2.$$

În prima parte a ecuației, s-a adăugat o sumă adițională peste numărul de noduri de ieșire. Aceasta însumează valorile nodurilor de pe stratul de ieșire.

Partea a doua a ecuaţiei este termenul de regularizare, fiind destinat menţinerii unor valori mici ai parametriilor Θ. Valoarea termenului de regularizare este dată de hiperparametrul λ înmulțit cu pătratele ponderilor Θ. Regularizarea funcției de cost ne ajută în determinarea unor ponderi optime care nu au valori mari. Cu cât ponderile sunt mai mari, cu atât valoarea termenului de regularizare este mai mare, implicit crescând și valoarea funcției de cost.

Dimensiunea matricei ponderilor (parametrilor) este $\Theta \in \Re^{M\times N}$, unde:

- M: numărul de rânduri este egal cu numărul de noduri din următorul strat (fără unitatea bias).
- N: numărul de coloane este egal cu numărul de noduri din stratul curent (inclusiv unitatea bias).

Observații:

- suma dublă (primul termen al ecuației) însumează costuri de tip regresie logică pentru fiecare nod din stratul de ieșire;
- suma triplă (al doilea termen al ecuației) însumează pătratele Θ din întreaga rețea.

7.2 Metoda propagării înapoi a erorii

Metoda propagării înapoi a erorii este similară cu minimizarea gradientului de la regresia logistică sau regresia liniară. Scopul este de a calcula $min_{\Theta}J(\Theta)$. Dorim să minimizăm funcția de cost J folosind un set optim de parametrii Θ.

Primul pas este de a calcula derivatele parțiale ale lui $J(\Theta)$:

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} \tag{7.1}$$

Mai exact, vom calcula derivatele parţiale pentru fiecare nod $\delta_j^{(l)}$. **Derivatele reprezintă eroarea nodului j din stratul l.** Pentru ultimul strat al rețelei, putem calcula valorile δ în felul următor:

$$\delta^{(L)} = y - a^{(L)}, \tag{7.2}$$

unde, L este numărul total de straturi, iar $a^{(L)}$ este vectorul activărilor nodurilor de pe ultimul strat. Așadar, eroarea pentru ultimul strat reprezintă diferențele dintre activările nodurilor din ultimul strat și valorile dorite ale acestor noduri. Pentru straturile anterioare ale rețelei, putem calcula erorile folosind ecuația:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; a'(z^{(l)}). \tag{7.3}$$

unde:

- $.*$ este un operator care denotă produsul scalar dintre doi vectori (înmulțirea element-cu-element a doi vectori),
- $a'(z^{(l)})$ reprezintă derivata funției de activare.

În acest curs am folosit funcția sigmoid pentru a modela activarea fiecărui neuron. Astfel, funcția de activare $a(z^{(l)})$ și derivata sa devine:

$$\begin{aligned} a(z^{(l)}) &= g(z^{(l)}) \\ g'(z^{(l)}) &= g(z^{(l)}) \; .* \; (1 - g(z^{(l)})) \end{aligned} \tag{7.4}$$

a cărei derivată este dată în Ecuația 6.13. Ecuația completă pentru calculul gradienților din

stratul l devine astfel:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; g(z^{(l)}) \; .* \; (1 - g(z^{(l)})). \tag{7.5}$$

Valorile δ pentru stratul l se calculează înmulțind δ din stratul următor $(l+1)$ cu matricea Θ a stratului curent. Având erorile și activările, putem determina gradientul fiecărui parametru θ față de valoarea funcției de cost, pentru fiecare exemplu de antrenare t:

$$\frac{\partial J(\theta)}{\partial \theta_{i,j}^{(l)}} = \frac{1}{m} \sum_{t=1}^{m} g_j^{(t)(l)} \delta_i^{(t)(l+1)}, \tag{7.6}$$

unde:

- $\delta^{(l+1)}$ și $g^{(l+1)}$ sunt vectori de dimensiune s_{l+1};
- $g^{(l)}$ este funcția sigmoid, fiind un vector de s_l elemente;
- multiplicând $g^{(l)}$ cu $\delta^{(l+1)}$ vom obține o matrice de dimensiune $s_{l+1} \times s_l$, având aceeași dimensiune cu $\Theta^{(l)} = s_{l+1} \times s_l$;
- procesul produce un termen gradient pentru fiecare element din $\Theta^{(l)}$.

Forma finală a metodei de propagare înapoi a erorii se obține introducând:

- $\Delta^{(l)}$: matrice acumulator pentru stratul l, pe care o folosim la acumularea termenilor derivatelor parţiale, fiind idexată prin ij;
- $D_{i,j}^{(l)}$: derivatele parțiale pentru toate exemplele de antrenare;
- i: index în baza de date de antrenare ($i \in \{1, ..., m\}$);
- l: index în straturile rețelei;
- j: index în neuronii dintr-un strat (neuronul j din stratul l);
- i, j: index în matricea de ponderi Θ.

Astfel, se poate rezuma algoritmul propagării înapoi a erorii (backpropagation):

Algoritmul 7.1 Algoritmul propagării înapoi a erorii (backpropagation)

- Dându-se setul de antrenare $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$.
- Se inițializează $\Delta_{i,j}^{(l)} = 0$ pentru toate (l, i, j)
- Pentru fiecare exemplu de antrenare t între 1 și m:
  - se setează $a^{(1)} := x^{(t)}$;
  - se execută feedforward pentru a obține $a^l = g^l$ pentru $l = 2, 3, \ldots, L$;
  - folosind $y^{(t)}$, se calculează eroarea pe ultimul strat: $\delta^{(L)} = g^{(L)} - y^{(t)}$;
  - se calculează eroarea pe straturile anterioare: $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; g^{(l)} \; .* \; (1 - g^{(l)})$;
  - se ajustează $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (g^{(l)})^T$;
- Pentru $j \neq 0$: $D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \theta_{i,j}^{(l)} \right)$
- Pentru $j = 0$: $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$

Termenii $D_{i,j}^{(l)}$ sunt derivatele parțiale căutate:

$$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \theta_{i,j}^{(l)}}. \tag{7.7}$$

Observații:

1. termenul λ se numește termen de *regularizare*; aceasta ajută la anumite probleme ale rețelelor neuronale (ex. supraantrenarea);

2. cazul $j = 0$ pentru calculul matricei D corespunde bias-ului; în acest caz nu se ia în considerare termenul de regularizare.

7.3 Exemplu de calcul

Vom aplica algoritmul de propagare înapoi a erorii pe rețeaua neuronală ilustrată în Figura 7.1.

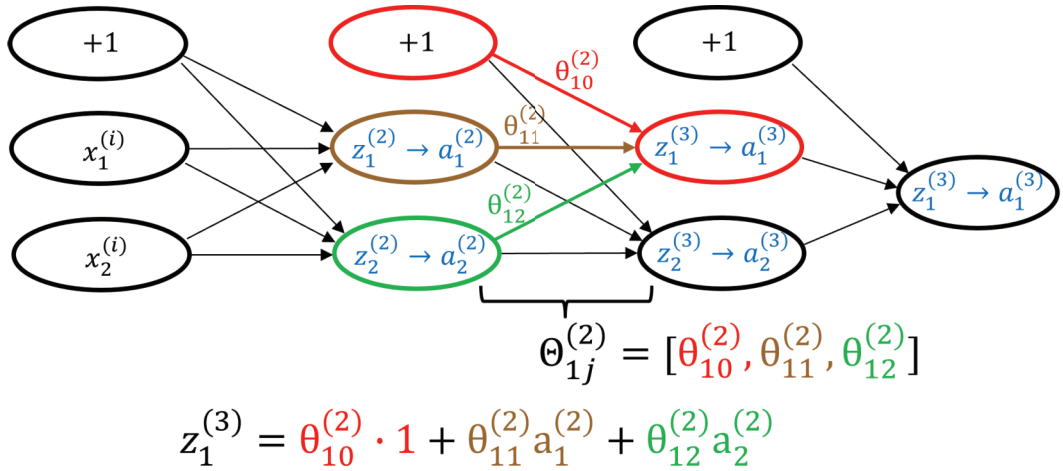

Fig. 7.1 Rețea neuronală cu două straturi ascunse.

Dându-se un exemplu de antrenare (x, y), se execută mai întâi propagarea înainte (*feed-forward* sau *forward propagation*):

1. $a^{(1)} = x$;

2. $z^{(2)} = \Theta^{(1)} a^{(1)}$;

3. $a^{(2)} = g(z^{(2)})$;

4. $z^{(3)} = \Theta^{(2)} a^{(2)}$;

5. $a^{(3)} = g(z^{(3)})$;

6. $z^{(4)} = \Theta^{(3)} a^{(3)}$;

7. $a^{(4)} = h_\Theta(x) = g(z^{(4)})$.

Observație: în calculul lui $z^{(l)}$, trebuie să se țină cont și de bias.

După ce s-a executat pasul de feedforward, se execută propagarea înapoi (backpropagation), de la dreapta la stânga:

$$\delta^{(2)} \leftarrow \delta^{(3)} \leftarrow \delta^{(4)} \tag{7.8}$$

Termenul $\delta^{(1)}$ este inexistent datorită faptului că el corespunde stratului de intrare (nu dorim sa modificăm caracteristicile de intrare, deci nu avem nevoie de gradienți pentru ele).

Graful propagării înapoi a erorii este redat în Figura 7.2.

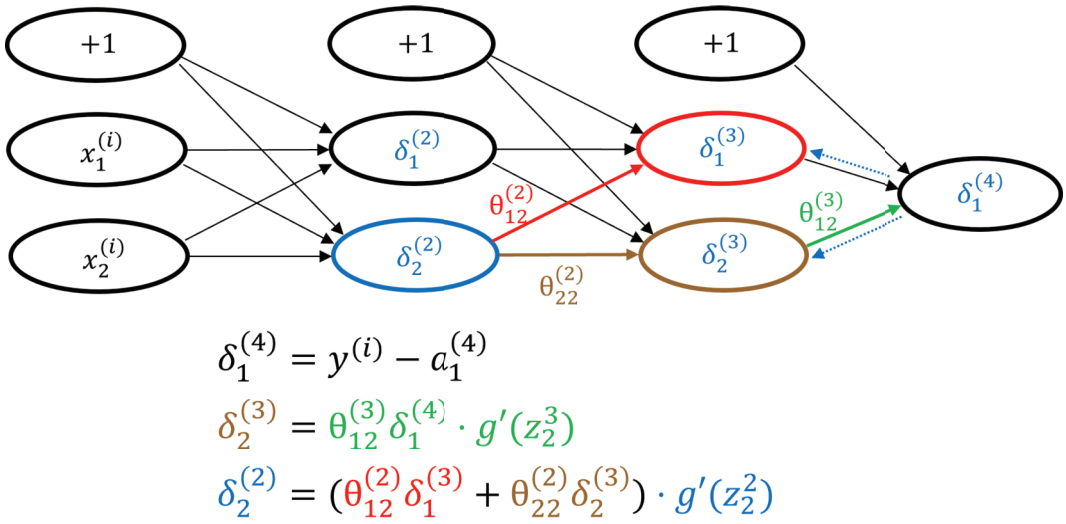

Fig. 7.2 Graful propagării înapoi a erorii.

Propagarea înapoi a erorii ce are ca rezultat $\delta_j^{(l)}$, adică eroarea nodului j din stratul l pentru variabilele de intrare date:

1. $\delta_j^{(4)} = y_j - a_j^{(4)}$;
2. $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^3)$;
3. $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^2)$.

După ce s-au obținut erorile $\delta^{(l)}$ ale nodurilor, se obțin derivatele parțiale $D_{i,j}^{(l)}$.

Observații:

1. δ ne spune cu cât dorim să modificăm ponderile reţelei pentru a schimba valorile intermediare z;
2. în funcţie de modul de implementare al algoritmului de propagare înapoi a erorii, se pot calcula valorile δ şi pentru neuronii bias;
3. Parametrii Θ nu mai sunt vectori, ca în cazul regresiei logistice, ci matrici.

7.4 Calculul numeric al gradientului

Există două modalități de calcul al gradienților:

- *analitic*, așa cum am făcut până acum (este mai rapid, mai exact);
- *numeric*, care este o metodă înceată și imprecisă, însă ușor de implementat.

Calculul numeric al gradientului se obține prin metoda diferențelor finite, aplicând una dintre formulele de mai jos:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{7.9}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h} \tag{7.10}$$

Metoda aplică o schimbare mică în valoarea de intrare a funcției (de ex. $h = 0.00001$), calculând astfel derivata funcției prin compararea noii valori de ieșire $f(x+h)$ cu valoarea inițială dată de $f(x)$.

Gradientul numeric pentru o funcție de o variabilă este exemplificat în Figura 7.3. Derivata este aproximativ panta tangentei în punctul în care se calculează derivata. Dacă h este foarte mic, aproximarea numerică se apropie de valoarea derivatei analitice.

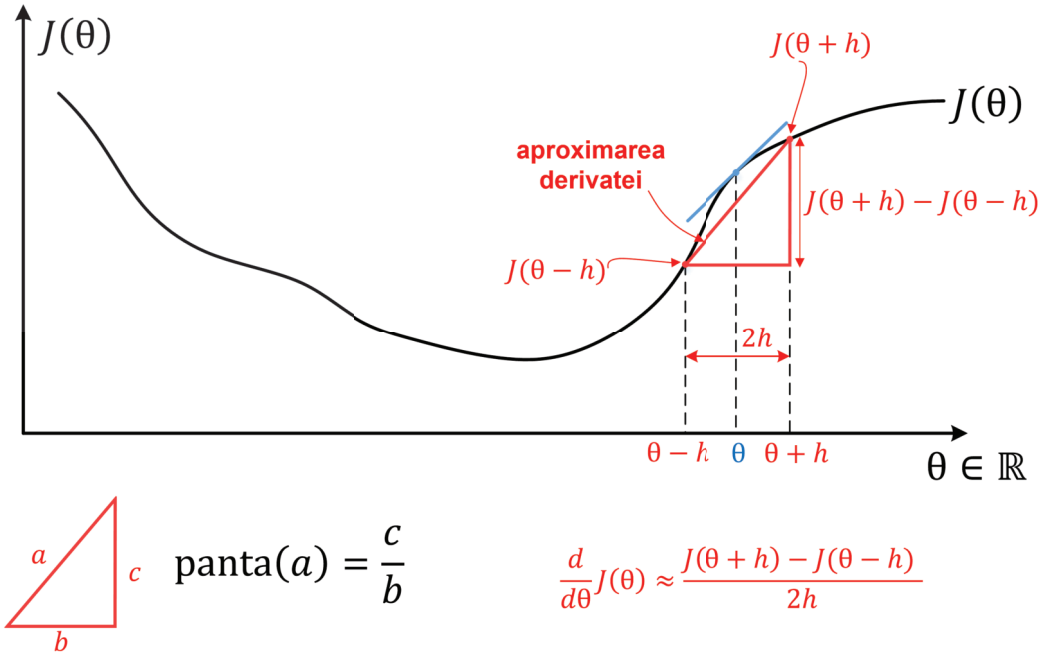

Fig. 7.3 Gradientul numeric pentru o funcție de o variabilă.

Matricea de ponderi Θ dintr-o rețea neuronală poate fi descompusă sub forma unui vector $\Theta \in \Re^n$ ce conţine ponderile $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$:

$$\Theta = [\theta_1, \theta_2, ..., \theta_n] \tag{7.11}$$

Derivatele parțiale obținute prin calculul gradientului numeric sunt ilustrate în Figura 7.4.

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + h, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1 - h, \theta_2, \theta_3, \ldots, \theta_n)}{2h}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + h, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2 - h, \theta_3, \ldots, \theta_n)}{2h}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n + h) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n - h)}{2h}$$

Fig. 7.4 Derivatele parțiale obținute prin calculul gradientului numeric.

Gradientul numeric aproximează derivatele parțiale $D_{i,j}$ obținute prin calcul analitic. În practică folosim următoarea abordare în antrenarea rețelelor neuronale:

- se implementează metoda de backpropagation folosind gradientul analitic;
- se implementează gradientul numeric;
- se verifică faptul că valorile obținute prin propagarea înpoi a erorii şi prin gradientul numeric sunt aproximativ aceleaşi;
- gradientului numeric este oprit (pentru că este încet la calcul), antrenarea reţelei neuronale utilizând doar propagarea înapoi a erorii.

7.5 Inițializarea ponderilor Θ

Inițializarea ponderilor Θ cu zero nu funcționează pentru rețelele neuronale. În acest caz, în timpul propagării înapoi, toate nodurile își vor modifica valorile cu aceeași valoare în mod repetat și nu vom putea minimiza eroarea. Fiecare neuron ascuns va calcula aceeaşi funcţie pe baza valorilor de intrare (la fiecare iteraţie de antrenare, ponderile vor fi egale între ele).

Putem evita acest lucru inițializând ponderile cu valori aleatoare cuprinse între $[-\epsilon, \epsilon]$. Similar se procedează și pentru bias-urile nodurilor.