7. Antrenarea reţelelor neuronale

Funcția de cost
Metoda propagării înapoi a erorii
Aplicație

7.1 Gradientul unei funcții

Derivata, sau gradientul unei funcții, ne indică rata de modificare a funcției față de o variabilă, în jurul unei regiuni infinitezimal de mică, aproape de un anumit punct pentru care se evaluează derivata:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{7.1}$$

Fracția din partea stângă a Ecuației 7.1 nu reprezintă operația de împărțire, ci indică notația operatorului $\frac{d}{dx}$ aplicat funcției f, ce returnează derivata lui f. Atunci când h este foarte mic, funcție este aproximată printr-o linie dreaptă, unde derivata este panta acestei linii. Cu alte cuvinte, derivata unei funcții f față de o variabilă x indică senzitivitatea lui f față de valoarea lui x. Acest lucru este vizibil atunci cand rearanjăm expresia 7.1 astfel:

$$f(x+h) = f(x) + h\frac{df(x)}{dx} \tag{7.2}$$

Expresia 7.2 ne spune că valoarea funcției f crește sau descrește atunci cand valoarea lui x se modifică la $x+h$, valoarea funcției modificându-se cu cantitatea h înmulțită cu derivata funcției f.

Gradientul unei funcții poate fi obținut analitic, prin formulele de diferențiere, sau numeric. Gradientul numeric este încet, imprecis, însă ușor de programat, în timp ce gradientul analitic: este rapid, precis, însă susceptibil la erori. În practică, se folosește gradientul analitic, însă implementarea gradientului analitic este verificată folosind gradientul numeric.

Gradientul unei funcții f se notează cu ∇f. Spre exemplu, gradientul unei funcții de 2 variabile $f(x, y)$ este:

$$\nabla f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right], \tag{7.3}$$

unde operatorul $\frac{\partial}{\partial x}$ indică derivata parțială a funcției $f(x, y)$ față de variabila x.

Pentru simplitate, vom folosi termenul "gradient față de x", în locul termenului complet de "derivată parțială față de x"

7.2 Interpretarea grafică a gradientului

În rețelele neuronale adânci, precum rețelele neuronale convoluționale, gradientul funcției de cost poate atinge dimensiune foarte mari, aproape imposibil de scris analitic. O simplificare a formei gradientului este sa îl reprezentăm ca și un graf computațional.

Fie funcția de trei variabile:

$$f(x, y, z) = (x + y)z \tag{7.4}$$

reprezentată grafic prin diagrama din Figura 7.1. Pentru valorile argumentelor $x = -2$, $y = 5$, $z = -4$, funcția va returna valoarea $f(x, y, z) = -12$. Calculul valori de ieșire a funcției, dându-se valorile de intrare, îl vom denumi ca și "propagare înainte" (feedforward propagation): dându-se valorile de argumentelor, calculăm valorile de ieșire a fiecărui nod din graf, de la stânga la dreapta.

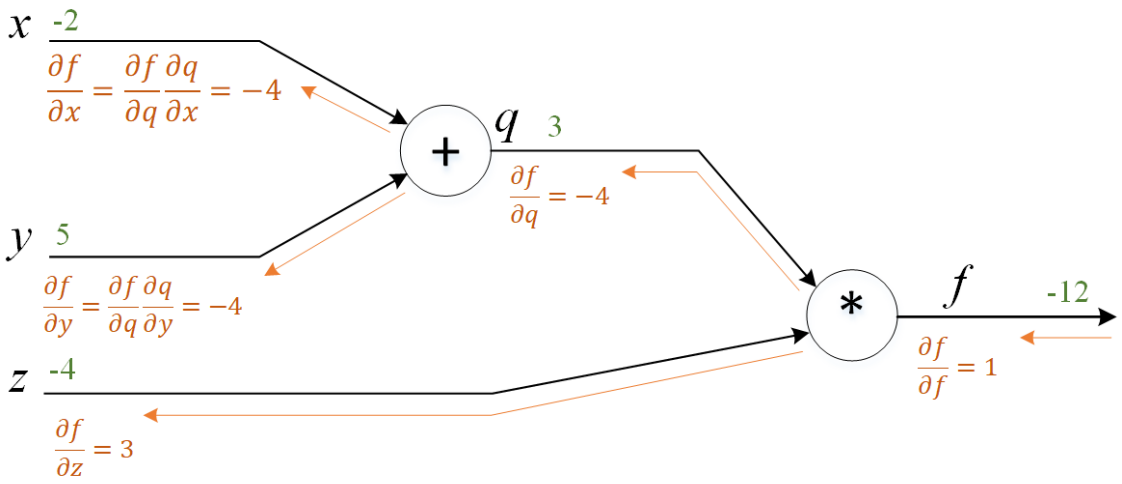

Fig. 7.1 Graful computațional pentru funcția $f(x, y, z) = (x + y)z$. Valorile de intare și ieșire din noduri sunt scrise în verde, deasupra liniei de propagare înainte (feedforward), iar gradienții în roșu, sub linie.

În următoarea fază, dorim să calculăm gradientul funcție $f(x, y, z)$ față de argumentele x, y, z. Pentru aceasta, introducem o funcție intermediară q ce calculează valoarea nodului "+", și anume $q = x + y$. q este o valoare intermediară folosită în calculul valorii funcției f. Gradientul lui f reprezintă derivatele parțiale ale funcției față de variabilele de intrare:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \tag{7.5}$$

Derivatele parțiale, sau gradienții, fiecărui nod din graful computațional sunt:

$$q = x + y : \ \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \tag{7.6}$$

$$f = qz : \ \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \tag{7.7}$$

Vom denumi "gradienți locali" derivatele parțiale $\frac{\partial q}{\partial x}$ și $\frac{\partial q}{\partial y}$.

Gradientul funcției din Figura 7.1 se calculează prin "propagarea înapoi" (backpropagation), de la dreapta la stânga, pornind de la valoarea de ieșire a funcției și calculând derivatele fiecărui nod pana la variabilele de intrare.

Pornind de la ieșire, gradientul funcției f față de ea însăși este identitatea $\frac{\partial f}{\partial f} = 1$. Gradientul lui f față de z este $\frac{\partial f}{\partial z} = q = 3$. Intuitiv, valoarea gradientului în acest caz, ne spune că influența variabilei z asupra valorii finale a lui f este pozitivă, având o pantă, sau "forță", egală cu 3. Daca incrementăm z cu o valoare h, atunci ieșirea grafului va fi crescută cu o valoare $3h$.

Gradientul lui f față de q ne spune că dacă q va crește, atunci ieșirea grafului va scădea. De exemplu, dacă q va crește cu valoarea h, atunci ieșirea grafului va scădea cu $4h$.

Gradientul funcției f față de variabila de intrare y se calculeaza folosind regula de înlănțuire, și anume:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = -4 \tag{7.8}$$

Influența lui y asupra lui q este 1 ($\frac{\partial q}{\partial y} = 1$). Influența variabilei de intrare y asupra ieșirii grafului este dată prin multiplicarea gradienților $\frac{\partial f}{\partial q} = -4$ și $\frac{\partial q}{\partial y} = 1$.

x și y au o influență pozitivă asupra lui q cu o pantă de valoarea 1. Crescând x cu o valoarea h, va crește q cu valoarea h

Creșterea lui x va crește valoarea lui q, care la rândul lui va descrește valoarea lui f.

Figura 7.1 ilustrează un nod dintr-un graf computațional ce calculează valoarea de ieșire z folosind funcția de activare f. De îndată ce nodul primește valorile de intrare x și y, în timpul operației de propagare înainte, acesta poate calcula și valorile gradienților locali, și anume $\frac{\partial z}{\partial x}$ și $\frac{\partial z}{\partial y}$. Gradienții locali vor fi folosiți în calculul gradientului unei funcții finale L, funcție ce se află la ieșirea grafului. În cursul de față, L reprezintă funcția de cost.

7.3 Reprezentarea grafică a funcției de regresie logistică

în graful computațional putem reprezenta orice fel de nod, atâta timp cât funcție ce o implementează este derivabilă.

Un alt exemplu de graf computațional este redat în Figura 7.3, pentru funcția de regresie

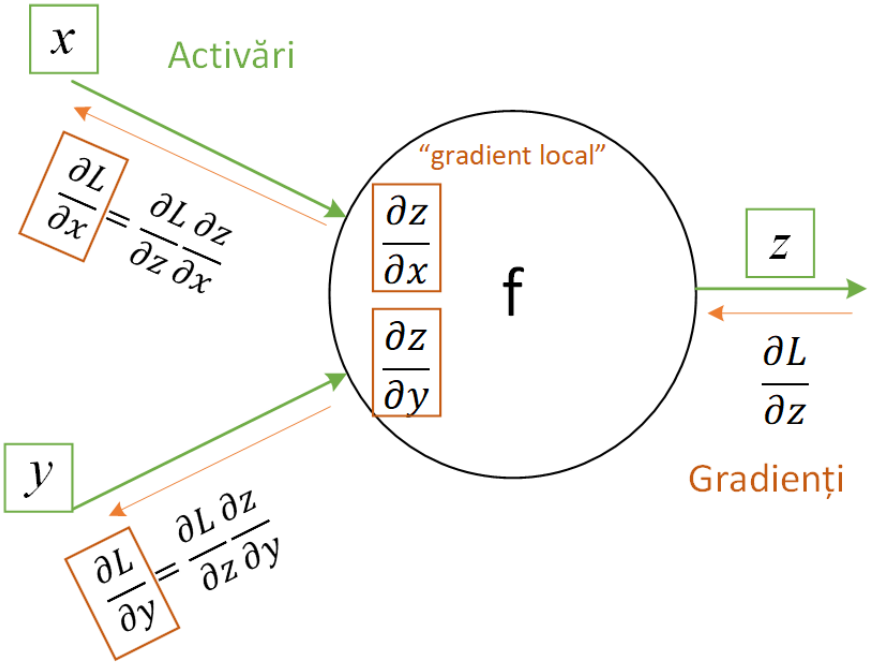

Fig. 7.2 Nod ce calculează funcția de activare f.

logistică:

$$f(\theta, x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}} \tag{7.9}$$

Se poate observa că funcția de regresie logistică reprezintă modelul unui neuron cu n intrări x și n parametrii θ.

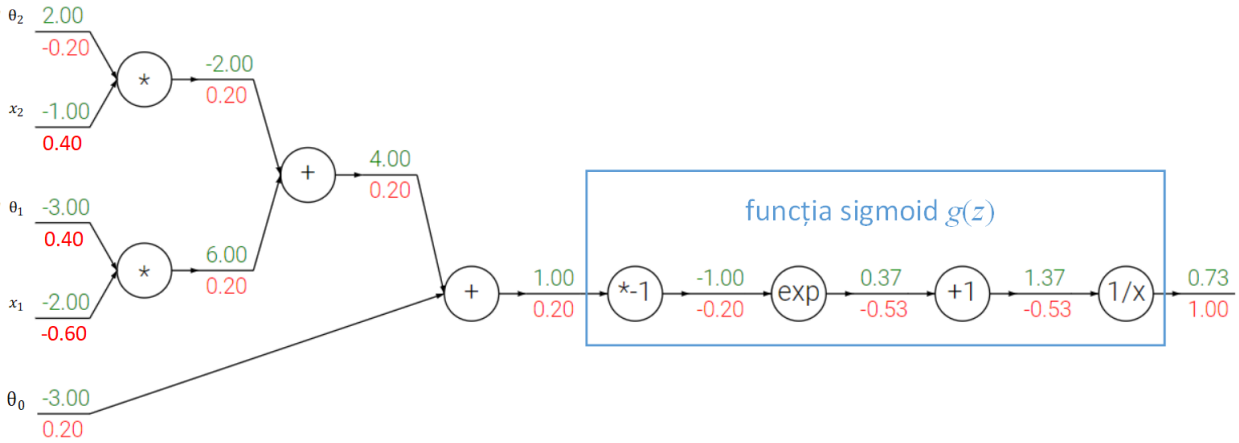

Fig. 7.3 Graf computațional pentru funcția de regresie logistică.

Derivatele nodurilor grafului din Figura 7.3 sunt:

$$\begin{aligned} f(x) = \frac{1}{x} \quad &\rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \\ f_c(x) = c + x \quad &\rightarrow \quad \frac{df}{dx} = 1 \\ f(x) = e^x \quad &\rightarrow \quad \frac{df}{dx} = e^x \\ f_a(x) = ax \quad &\rightarrow \quad \frac{df}{dx} = a \end{aligned} \tag{7.10}$$

unde funcțiile f_c și f_a translatează (adunare) intrarea cu o constantă c, respectiv scalează (înmulțire) intrarea cu o constantă a.

În cazul nodului $\frac{1}{x}$, gradientul este negativ deoarece atunci când valoarea de intrare în nod crește, valoarea de ieșire a nodului scade cu $\frac{1}{x}$. Următorul nod adaugă constanta +1, derivata acestuia fiind 0.

Gradientul local al nodului "+" este 1, ceea ce înseamnă că acest nod va copia către intrare sa gradientul primit la ieșire. Un nod plus distribuie pur și simplu gradientul primit către nodurile anterioare.

Pentru verificare, vom deriva funcția sigmoid, marcată cu albastru în graful din Figura 7.3:

$$a(z) = \frac{1}{1 + e^{-z}} \tag{7.11}$$

$$\frac{dg(a)}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = \left(\frac{1 + e^{-z} - 1}{1 + e^{-z}}\right)\left(\frac{1}{1 + e^{-z}}\right) = (1 - a(z))a(z) \tag{7.12}$$

Prin derivare, calculul gradientului devine mult mai simplu. Spre exemplu, în timpul propagării înainte, dacă funcția sigmoid primește la intrare valoarea 1.0, atunci ieșirea ei va fi 0.73. Utilizând Ecuația 7.11, gradientul local va fi $(1 - 0.73) \cdot 0.73 = 0.2$. În practică, vom grupa astel de operații pentru simplificarea calculului gradientului.

7.4 Funcția de cost pentru o rețea neuronală

În acestă lucrare de laborator 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.

7.5 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 $J(\Theta)$:

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} \tag{7.13}$$

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:

$$\delta^{(L)} = y - a^{(L)}, \tag{7.14}$$

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 7.14:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; a'(z^{(l)}). \tag{7.15}$$

unde, $.*$ este un operator care denotă înmulțirea element-cu-element a doi vectori, iar $a'(z^{(l)})$ reprezintă derivata funției de activare, care în cazul nostru este funcția sigmoid:

$$a'(z^{(l)}) = a(z^{(l)}). * \; (1 - a(z^{(l)})). \tag{7.16}$$

Ecuația completă pentru calculul gradienților din stratul l devine:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; a(z^{(l)}). * \; (1 - a(z^{(l)})). \tag{7.17}$$

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

$$\frac{\partial J(\theta)}{\partial \Theta_{i,j}^{(l)}} = \frac{1}{m} \sum_{t=1}^{m} a_j^{(t)(l)} \delta_i^{(t)(l+1)}, \tag{7.18}$$

unde:

- $\delta^{(l+1)}$ și $a^{(l+1)}$ sunt vectori de dimensiune s_{l+1};
- $a^{(l)}$ este un vector de s_l elemente;
- multiplicând $a^{(l)}$ 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 aceasta nu conţine regularizare. Introducem:

- $\Delta^{(l)}$: 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 pentru $l = 2, 3, \ldots, L$;
  - folosind $y^{(t)}$, se calculează eroarea pe ultimul strat: $\delta^{(L)} = a^{(L)} - y^{(t)}$;
  - se calculează eroarea pe straturile anterioare: $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \, .* \, a^{(l)} \, .* \, (1 - a^{(l)})$;
  - se ajustează $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$;
- 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:

$$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}. \tag{7.19}$$

Observații:

1. termenul λ se numește *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 și regularizarea.

7.5.1 Exemplu

Vom aplica algoritmul de propagare înapoi a erorii pe rețeaua neuronală ilustrată în Figura 7.4.

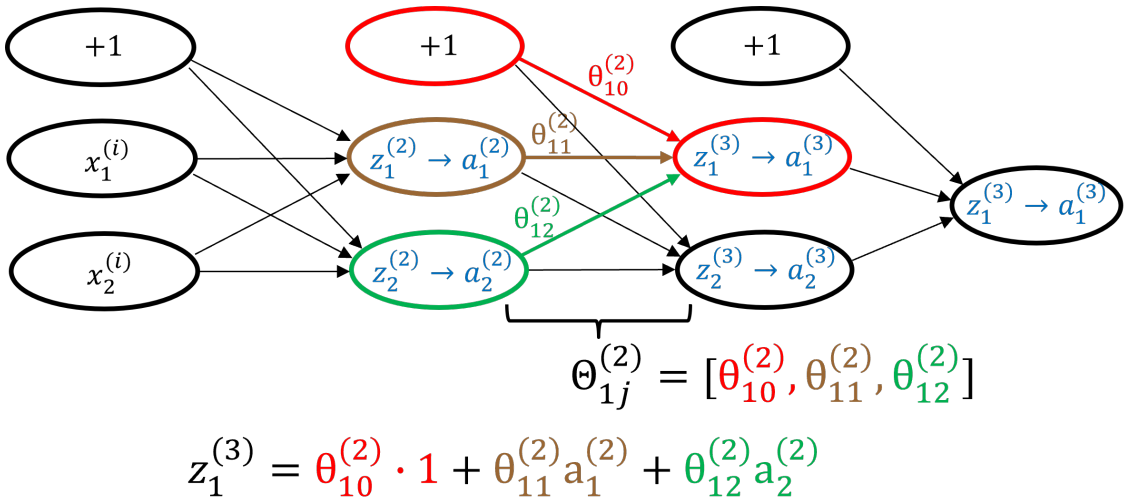

Fig. 7.4 Exemplu de strat de neuroni.

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:

$$\delta^{(2)} \leftarrow \delta^{(3)} \leftarrow \delta^{(4)} \tag{7.20}$$

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 7.5.

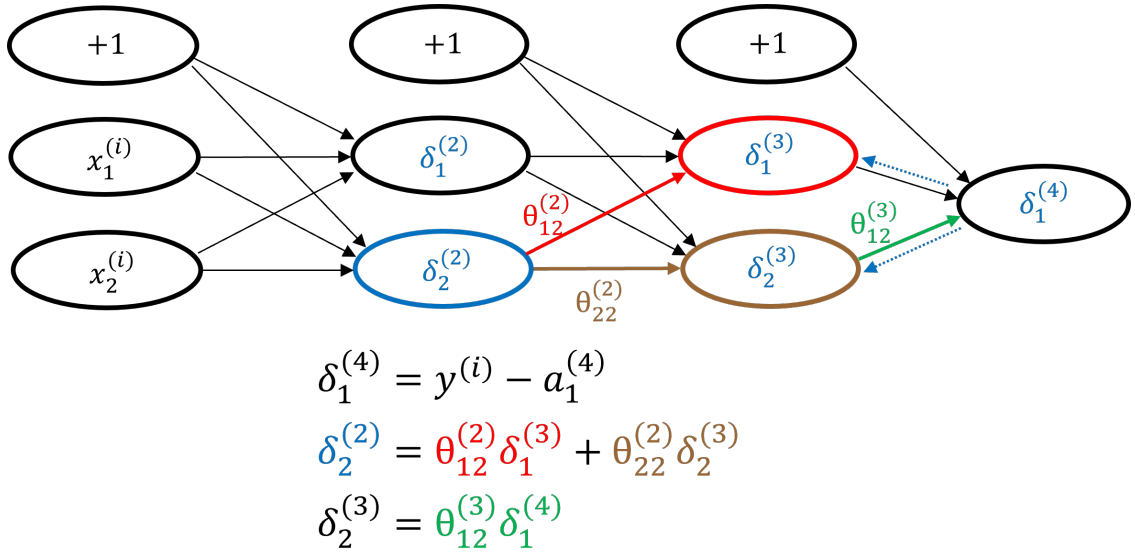

Fig. 7.5 Graful propagării înapoi a erorii.

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.

7.6 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 din formulele de mai jos:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{7.21}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h} \tag{7.22}$$

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

Gradientul numeric pentru o funcție de o variabilă este exemplificat în Figura 7.6. 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.

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

$$\Theta = [\theta_1, \theta_2, ..., \theta_n] \tag{7.23}$$

Derivatele parțiale obținute prin calculul gradientului numeric sunt ilustrate în Figura 7.7.

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

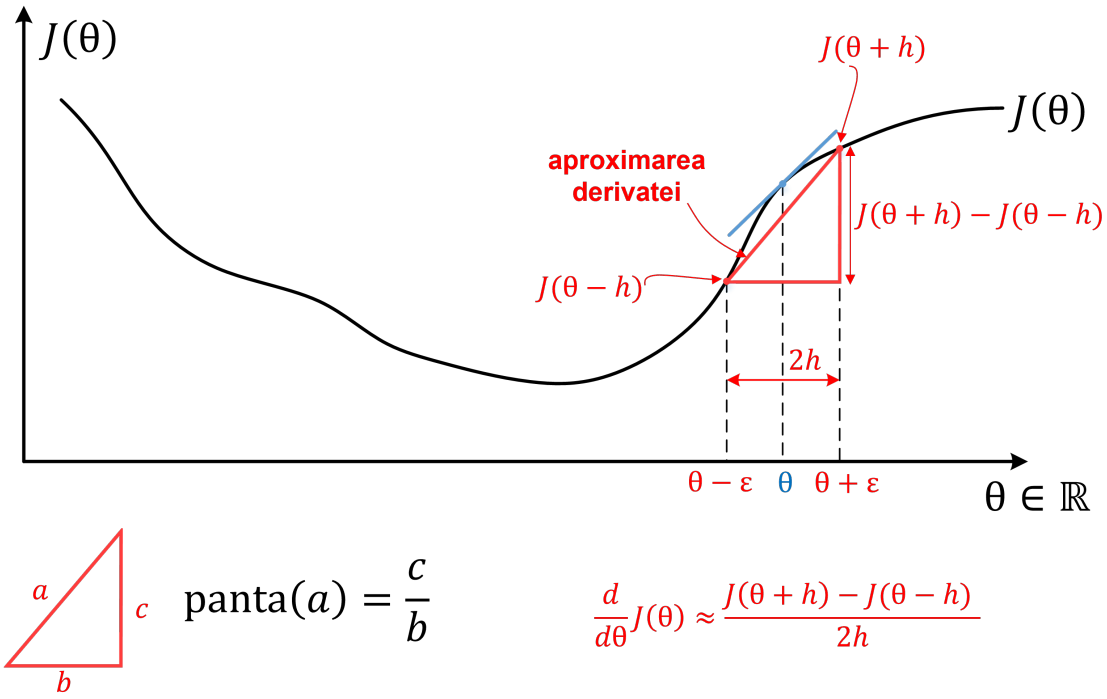

Fig. 7.6 Gradientul numeric pentru o funcție de o variabilă.

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

Fig. 7.7 Derivatele parțiale obținute prin calculul gradientului numeric.

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

7.7 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 aceaşi 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.

7.8 Cerinţe

De această dată, vom reproiecta rețeaua neuronală din lucrarea de laborator anterioră, astfel încât să nu fie nevoie să introducem manual ponderile reţelei. Ponderile vor fi învăţate automat prin algoritmul de backpropagation. Așadar, vom proiecta o rețea neuronală care simulează un circuit de însumare a doi biți format din porți logice ȘI-NU.