7. Antrenarea reţelelor neurale

Funcția de cost
Metoda propagării înapoi a erorii
Aplicație

7.1 Funcția de cost

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 neurală 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. Partea a doua a ecuaţiei este termenul de regularizare, fiind destinat menţinerii unor valori mici ai parametriilor Θ. Numărul de coloane din matricea Θ este egal cu numărul de noduri din stratul curent (inclusiv unitatea bias). Numărul de rânduri este egal cu numărul de noduri din următorul strat (fără unitatea bias).

Observații:

- suma dublă însumează costuri de tip regresie logică pentru fiecare nod din stratul de ieșire;
- suma triplă 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 $J(\Theta)$:

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

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 7.3:

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

unde, $.*$ este un operator care denotă înmulțirea element-cu-element a doi vectori.

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, derivatele parțiale 7.1:

$$\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}} = \frac{1}{m} \sum_{t=1}^{m} a_j^{(t)(l)} \delta_i^{(t)(l+1)}. \tag{7.4}$$

Forma aceasta nu conţine regularizare. Astfel, se poate rezuma algoritmul propagării înapoi a erorii:

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 $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$;
- se calculează $D_{i,j}^{(l)}$: $D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)}\right)$, pentru $j \neq 0$;
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$, pentru $j = 0$.

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

$$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}. \tag{7.5}$$

Observații:

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

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

7.2.1 Exemplu

Vom aplica algoritmul de propagare înapoi a erorii pe rețeaua neurală ilustrată în figura 7.1. Dându-se un exemplu de antrenare (x, y), se execută mai întâi propagarea în față (*feedforward* 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 backpropagation, ce are ca rezultat $\delta_j^{(l)}$, adică eroarea nodului j din stratul l:

1. $\delta_j^{(4)} = a_j(4) - y_j$;

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 $D_{i,j}^{(l)}$.

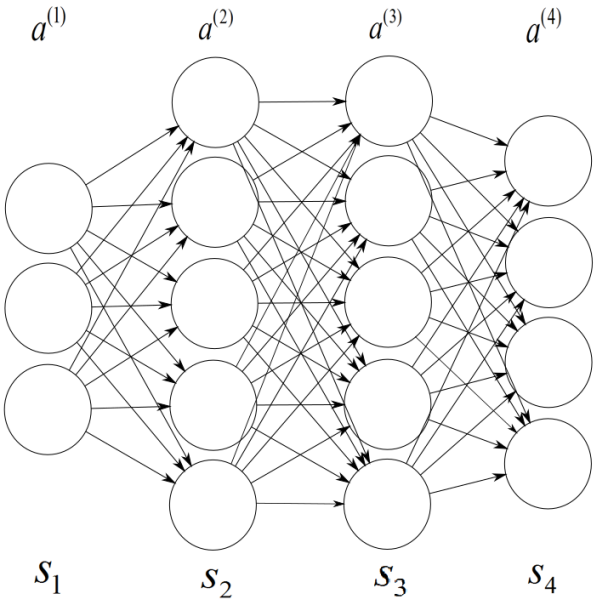

Fig. 7.1 Exemplu de strat de neuroni.

7.2.2 Inițializarea parametrilor Θ

Inițializarea ponderilor Θ cu zero nu funcționează pentru rețelele neurale. 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. Putem evita acest lucru inițializând ponderile cu valori aleatoare cuprinse între $[-\epsilon, \epsilon]$. Similar se procedează și pentru bias-urile nodurilor.

7.3 Cerinţe

De această dată, vom reproiecta rețeaua neurală 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 neurală care simulează un circuit de însumare a doi biți format din porți logice ȘI-NU.