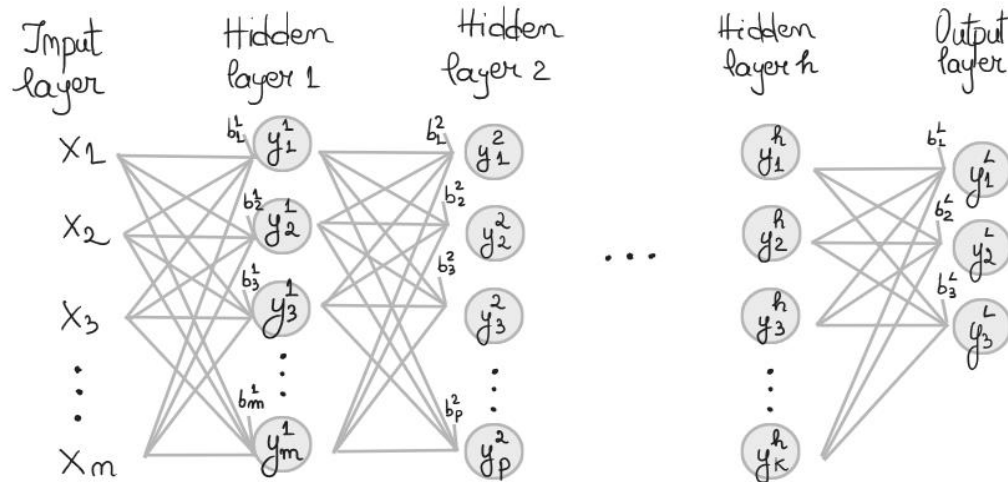


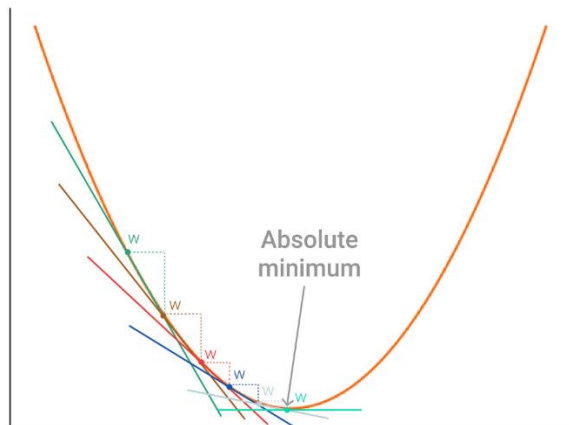
Rețele neuronale



Algoritmul **Backpropagation** ne ajută să antrenăm rețele neuronale.

A antrena o rețea = a updata ponderile și biasurile rețelei a.î. să se minimizeze o anumită funcție de cost (/funcție de eroare). **Funcția de cost** ne spune cât de apropiate sunt outputurile date de rețea de outputurile reale (clasele) fiecărei instanțe.

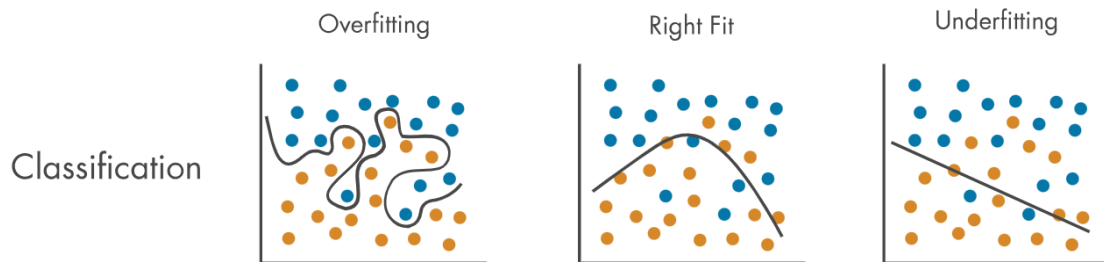
Algoritmul **Gradient Descent** ne ajută să minimizăm funcția de cost. La fiecare pas se fac mișcări în direcția opusă a gradientului (Gradientul ne dă direcția celei mai mari creșteri ale funcției).



Obs:

1. Dimensiunea stratului de intrare = dimensiunea unei instanțe (numărul de atribute de input)
2. Numărul de straturi ascunse și dimensiunea fiecărui strat ascuns se pot calcula conform [unor euristici](#).

3. Dimensiunea stratului de ieșire în general depinde de taskul realizat (regresie, clasificare, ...).
Pentru clasificare în mai multe clase de obicei:
 - se folosește funcția de activare *softmax* (outputează probabilități)
 - numărul de neuroni de pe stratul de output = nr. de clase
4. O rată de învățare prea mică → convergență lentă
O rată de învățare prea mare → se poate trece peste soluție în algoritmul de Gradient Descent
5. Un număr prea mic de epoci → underfitting
Un număr prea mare de epoci → overfitting



6. Ponderile pot fi reținute în matrici (cu dimensiunea nr neuroni strat anterior * nr neuroni strat următor). Ponderile în general se inițializează cu valori random mici, diferite de 0.
Dacă ponderile sunt 0 → gradientii vor fi 0 → rețeaua nu va învăța
Dacă ponderile au valori prea mari → în funcție de funcția de activare utilizată, convergența poate fi foarte lentă / pot apărea alte probleme (e.g., *exploding gradients*)
7. Funcția de cost
 - Clasificare: *cross entropy*
 - Regresie: *mean squared error, mean absolute error, ...*

Cross entropy ne spune cât de asemănătoare sunt 2 distribuții de probabilitate (adică cea outputată de rețea – după ce aplicăm softmax și cea reală – construită pe baza clasei reale a instanței)

8. Antrenarea rețelei

Pt. un număr de epoci și/sau cât timp funcția de cost nu scade sub o valoare:
Pt. fiecare instanță X în parte (/ pt. un batch de instanțe):

Feed/forward

Se dă instanța ca input la rețea și se calculează succesiv:

- Ieșirea neuronilor din primul strat ascuns
- Ieșirea neuronilor din al doilea strat ascuns
- ...
- Ieșirea neuronilor din a h-lea strat ascuns
- Ieșirea neuronilor din stratul de ieșire

Se calculează valoarea funcției de cost

Backpropagation

Pornind de la valoarea funcției de cost se calculează succesiv:

Gradienții pentru neuronii din stratul de ieșire

Corecțiile de ponderi dintre stratul ascuns h și cel de ieșire

Gradienții pentru neuronii din stratul ascuns h

Corecțiile de ponderi dintre stratul ascuns $h-1$ și h

...

Gradienții pentru neuronii din primul stratul ascuns

Corecțiile de ponderi dintre stratul de intrare și primul

strat ascuns

După ce am prezentat instanța / batch-ul la rețea se folosesc corecțiile de ponderi pentru a updata ponderile (și bias-urile).

Notății

Fie $X = (x_1, x_2, x_3, \dots, x_n)$ instanța de input.

Fie $Y^s = (y_1^s, y_2^s, \dots)$ ieșirile pentru stratul ascuns cu numărul s

Fie w_{ij}^s ponderea dintre stratul ascuns numărul $s - 1$ și stratul ascuns s , care conectează neuronul i din stratul $s - 1$ cu neuronul j din stratul s . Dacă s este 1, atunci sunt ponderile dintre primul strat de intrare și primul strat ascuns).

Fie b_j^s biasul neuronului j din stratul s .

f reprezintă funcția/funțiile de activare (atenție, se pot folosi funcții de activare diferite de la strat la strat). f' reprezintă derivata funcției de activare.

Fie δ_j^s gradientul pentru neuronul j din stratul s .

α este rata de învățare.

Formule

Ieșirile neuronilor din primul strat ascuns:

$$y_j^1 = f(b_j^1 + \sum x_i \cdot w_{ij}^1)$$

Ieșirile neuronilor din a-s-lea strat ascuns:

$$y_j^s = f(b_j^s + \sum y_j^{s-1} \cdot w_{ij}^s)$$

Ieșirea neuronilor din stratul de ieșire:

$$y_j^L = f(b_j^L + \sum y_j^h \cdot w_{ij}^L)$$

Funcția de cost/ de eroare, definită pentru o instanță: $c(y^D, y^L)$ primește ca argumente outputurile neuronilor din stratul de ieșire y^L și outputurile dorite/reale y^D . O valoare mică pentru funcția de cost ne indică apropierea dintre outputurile emise de rețea și cele reale. Valoarea funcției de cost pentru o epocă se definește însumând valorile funcției de cost pentru toate instanțele.

Exemple de funcții de cost:

- $C_{MSE}(y^D, y^L) = \frac{1}{n \cdot k} \sum_{i=1}^n \sum_{j=1}^k (y_{ij}^D - y_{ij}^L)^2$
- $C_{CrossEntropy}(y^D, y^L) = -\frac{1}{n} * \sum_{i=1}^n \sum_{j=1}^k (y_{ij}^D * \log(y_{ij}^L))$

, unde n este numărul de instanțe din setul de antrenare, k este numărul de neuroni de pe stratul de output, y_{ij}^L este outputul neuronului j al instanței numărul i din setul de antrenare.

Gradienții pentru neuronii din stratul de ieșire:

1. $\delta_j^L = f'(b_j^L + \sum y_j^h \cdot w_{ij}^L) \cdot (y_j^D - y_j^L)$ (dacă folosim funcția de cost MSE)
2. $\delta_j^L = y_j^D - y_j^L$ (dacă folosim funcția de cost Cross Entropy)

Demonstrația pentru prima formulă a fost discutată la curs.

Pentru a ajunge la a doua formulă (gradienții pentru stratul de ieșire când se folosește cross entropia) găsiți o demonstrație [aici](#). În unele demonstrații veți vedea inversate y_j^D și y_j^L în formulă, deoarece la update-ul corecțiilor gradienții se scad (aici se adună).

În ambele cazuri se folosește regula de înlănțuire (chaining) a derivatelor deoarece funcția de cost C nu depinde direct de ponderi și de biasuri.

Gradienții pentru neuronii din stratul s :

$$\delta_j^s = f'(b_j^s + \sum y_j^{s-1} \cdot w_{ij}^s) \cdot \sum_k \delta_k^{s+1} \cdot w_{jk}^{s+1}$$

Corecțiile de ponderi dintre stratul $s - 1$ și s :

$$\Delta w_{ij}^s = \Delta w_{ij}^s + \alpha \cdot y_i^{s-1} \cdot \delta_j^s$$

Corecțiile de ponderi dintre stratul de intrare și primul strat ascuns:

$$\Delta w_{ij}^1 = \Delta w_{ij}^1 + \alpha \cdot x_i \cdot \delta_j^1$$

Corecțiile de biasuri pentru stratul s :

$$\Delta b_j^s = \Delta b_j^s + \alpha \cdot \delta_j^s$$

În final (după ce am prezentat la rețea toate instanțele/un batch de instanțe/o instanță), ponderile și biasurile se vor update folosind corecțiile:

$$w_{ij}^s = w_{ij}^s + \Delta w_{ij}^s$$

$$b_j^s = b_j^s + \Delta b_j^s$$

Toate aceste operații se pot implementa folosind operații cu matrici.

Pentru evaluarea rezultatelor se pot folosi: acuratețea, precizia, recall-ul, o matrice de confuzie, etc.