Reţele neuronale - aspecte introductive

Asist.univ.dr Mihai Tudor

Obiective

- Înțelegerea conceptului de reţea neuronală.
- Înțelegerea conceptului de neuron, strat de neuroni, pondere, eroare.
- Implementarea scheletului unei reţele neuronale.

1 Conceptul de reţea neuronală

Rețelele neuronale sunt bazate pe modele matematice inspirate de funcționarea creierului. Ele permit procesarea informației printr-o rețea de neuroni artificiali, folosind ponderi și funcții de activare. Cele trei elemente de bază sunt:

- neuronul (unitate de procesare / unit / node) — unitatea de bază,
- ponderile (weights) — echivalentul forței conexiunilor sinaptice,
- funcția de activare (activation function) — determină dacă şi în ce măsură un neuron „se activează” şi transmite semnalul mai departe.

Modul în care neuronii sunt legați (arhitectura rețelei), metoda de determinare a ponderilor (training) şi funcţia de activare joacă un rol esenţial în capacitatea rețelei de a procesa informația. Acest curs are scopul de a construi o înţelegere intuitivă şi matematică a principiilor de bază.

2 O abordare vizuală a reţelelor neuronale

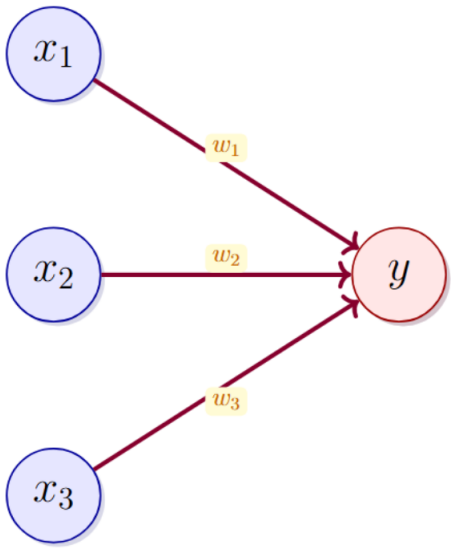
La fel ca și neuronii biologici, cei artificiali funcționează pe baza unei stări interne (state) numită (activation/activity level) care, odată calculată, generează un singur semnal de ieșire difuzat simultan către multiple ținte post-sinaptice, diferențierea informațională realizându-se prin ponderile specifice ale fiecărei conexiuni individuale.

Pentru a putea avea o imagine mai intuitivă asupra aspectelor teoretice ne raportăm la următorul exemplu. Vom considera un neuron Y care primește semnale de intrare (inputs) de la trei neuroni X_1, X_2, X_3. Nivelul de activitate al acestor neuroni (activity level) este dat de x_1, x_2, respectiv x_3 (în acest exemplu, cu rol de fixare la nivelul percepției asupra conceptelor, x_1, x_2 şi x_3 pot fi privite ca fiind valori numerice concrete). „Forța

conexiunilor sinaptice" dintre neuronii X_1, X_2, X_3 și neuronul Y se realizează prin ponderile w_1, w_2 şi w_3 În aceste condiții semnalul primit de neuronul Y (net input), este o sumă ponderată de forma:

$$y_{\text{input}} = w_1 x_1 + w_2 x_2 + w_3 x_3.$$

Activarea neuronului Y (activity level) se realizează prin aplicarea funcției de activare (activation function) asupra semnalului primit y_{input} astfel încât $Y = f(y_{\text{input}})$, unde y devine nivelul de activitate corespunzător neuronului Y, iar f funcția de activare.



3 De la simplu la complex

Componentele de bază ale oricărei rețele neuronale determină următoarea triadă:

$$\text{input} \longrightarrow \text{weight} \longrightarrow \text{prediction}$$

Simplist, le putem caracteriza pe acestea astfel:

- input: valoarea pe care o primește rețeaua
- weight (ponderea): parametrul care controlează transformarea
- prediction (predicția): rezultatul furnizat de rețea

Cu alte cuvinte, cea mai simplă transformare liniară $\text{prediction} = \text{input} \times \text{weight}$ devine fundamentul tuturor operațiilor mai complexe din rețelele neuronale.

4 Prima reţea neuronală

Ca primă reţea neuronală minimală, putem considera modelul:

$$\text{prediction} = \text{input} \times \text{weight}$$

Pornim de la următorul context concret: Cerem o predicție a notei finale a studenților pe baza orelor de studiu săptămânale. Datele folosite vor fi reținute în următoarele două liste:

```
study_hours_weekly = [15, 25, 35, 20]  # ore de studiu pe saptamana (input)
final_grade_percentage = [0.65, 0.78, 0.89, 0.72]  # nota finala (0-1) (predictia dorita)
```

Înainte să oferim o implementare a rețelei în Python și să discutăm aspectele tehnice, facem precizarea că lista "final grade percentage" reține nota finală reală obținută de fiecare student în urma numărului de ore de studiu efectuate. Astfel, un student care studiază 15 ore obține 0,65, pe când un student care studiază 35 de ore obține 0,89. Obiectivul rețelei este să prezică aceste note cu o acuratețe cât mai bună, tehnic vorbind aceasta reprezentând faza de antrenare a rețelei neuronale (training phase).

Implementare

```
weight = 0.02  # pondere initiala (ghicitura)

def neural_network(study_hours, weight):
    grade_prediction = study_hours * weight
    return grade_prediction

# Testarea modelului
for i in range(len(study_hours_weekly)):
    predicted_grade = neural_network(study_hours_weekly[i], weight)
    goal_pred = final_grade_percentage[i]
    error = (predicted_grade - goal_pred) ** 2

    print(f"Student {i+1}:")
    print(f"  Ore studiu/saptamana: {study_hours_weekly[i]}")
    print(f"  Nota reala: {goal_pred:.2f}")
    print(f"  Nota prezisa: {predicted_grade:.3f}")
    print(f"  Eroare patratica: {error:.6f}")
    print()
```

În urma rulării programului obținem următoarele rezultate:

```
Student 1:
  Ore studiu/saptamana: 15
  Nota reala: 0.65
  Nota prezisa: 0.375
  Eroare patratica: 0.075625

Student 2:
  Ore studiu/saptamana: 25
  Nota reala: 0.78
  Nota prezisa: 0.625
  Eroare patratica: 0.024025

Student 3:
```

```
  Ore studiu/saptamana: 35
  Nota reala: 0.89
  Nota prezisa: 0.875
  Eroare patratica: 0.000225

Student 4:
  Ore studiu/saptamana: 20
  Nota reala: 0.72
  Nota prezisa: 0.500
  Eroare patratica: 0.048400
```

Analizând rezultatele furnizate de rețea, observăm că singurul caz în care rețeaua reușește să se apropie destul de mult de rezultatul real este în cazul "Student 3", unde eroarea este mică (0.000225). În celelalte cazuri aceasta are tendința de a subevalua.

În acest moment avem cadrul necesar pentru a discuta câteva aspecte de ordin tehnic care se desprind:

1. Surprinzător sau nu, codul propus definește scheletul celei mai simple rețele neuronale având cele trei concepte care stau la baza arhitecturii acesteia:

$$\text{prediction} = \text{input} \times \text{weight}$$

   Mai exact, până în acest punct, codul mimează o regresie liniară simplificată. Vorbim în acest caz de Supervised Learning - rețeaua învață să mapeze input-urile la output-urile dorite.

2. Pentru a putea interpreta erorile, am folosit Eroarea pătratică (Mean Squared Error - MSE), aceasta având câteva avantaje clare comparativ cu eroarea absolută:

   - rețeaua "se concentrează" mai mult pe corectarea erorilor mari decât pe cele mici
   - Eroarea pătratică oferă proprietăți matematice valoroase în contextul optimizării rețelei propuse:
     - este o funcție diferențiabilă pe tot domeniul de definiție (eroarea absolută nu este diferențiabilă în punctul 0), ceea ce ne permite calculul gradientului pentru optimizare (aspect pe care îl vom analiza în cadrul postării)
     - în cazul transformării liniare (cazul particular abordat de rețeaua propusă), MSE este convexă:

$$\text{prediction} = \text{weight} \times \text{input}$$

$$\text{MSE}(\text{weight}) = \frac{1}{n}\sum(\text{weight} \times x_i - y_i)^2$$

$$\frac{\partial^2 \text{MSE}}{\partial w^2} = \frac{2}{n}\sum x_i^2 \geq 0 \text{ (întotdeauna pozitivă!)}$$

Acest aspect conduce la determinarea unui minim global, viteza de convergență fiind bună. În cazul rețelelor neuronale complexe în care intervin funcții de activare neliniare, convexitatea se pierde (altfel spus pot apărea mai multe minime locale posibile).

3. În cazul rețelei noastre întâlnim o limitare fundamentală - ponderea fixă:
   - nu avem nicio justificare pentru această valoare
   - pentru diferite probleme, am avea nevoie de ponderi diferite
   - performanța depinde complet de "ghicirea ponderii"

Aspectele dezbătute la punctul 3 deschid drumul spre următoarea întrebare:

„Dacă performanța depinde de ponderi, cum aleg ponderea optimă?"