Retea neuronala de tip multistrat pentru setul de date Iris

Varvara Ioan
Davidoaia Alexandru

Introducere

Proiectul implementează o rețea neuronală perceptron multistrat (MLP) utilizând setul de date Iris pentru clasificarea speciilor de flori. Rețeaua este antrenată folosind o combinație de propagare înainte și înapoi pentru optimizarea ponderilor. Aplicația este construită în Python folosind Flask pentru a oferi o interfață API REST.

Descrierea problemei

- Setul de date Iris conține 150 de instante distribuite pe trei clase (Setosa, Versicolor, Virginica), fiecare descrisă prin patru caracteristici:
    - Lungimea sepalei
    - Latimea sepalei
    - Lungimea petalei
    - Latimea petalei
- Scopul este de a dezvolta o rețea neuronală care să clasifice corect specia pe baza acestor caracteristici.

Aspecte teoretice privind algoritmul

- Rețeaua perceptron multistrat (MLP) este un tip de rețea neuronală feedforward, în care informația curge de la stratul de intrare, prin straturile ascunse, până la stratul de ieșire. Avantajul major al MLP constă în capacitatea sa de a aproxima funcții complexe, fiind astfel potrivită pentru probleme de clasificare și regresie.
- **Structura rețelei**
    - Stratul de intrare:
        - Primește datele brute (caracteristicile florilor) și le transmite stratului ascuns.
    - Straturile ascunse:
        - Sunt responsabile cu corespondenta intrare-iesire a retelei.
        - Utilizeaza functii de activare precum sigmoida si ReLu.
    - Stratul de iesire:
        - Produce predictii bazate pe informatia prelucrata in straturile ascunse.
        - Functia softmax este utilizata pentru clasificarea multiclasa, oferind probabilitati pentru fiecare clasa.
- **Functia Sigmoid**

- Formula: $S(x) = \frac{1}{1 + e^{-x}}$
- Este folosită în stratul ascuns pentru activarea neuronilor.

- **Functia ReLu**
  - Formula: $f(x) = \max(0, x)$
  - O functie de activare alternativa care poate fi folosita in locul functiei Sigmoid.
- **SoftMax**
  - Formula: $$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
  - Normalizează ieșirile stratului final astfel încât să reprezinte probabilități pentru fiecare clasă.
- **Propagarea inainte**
  - La intrările rețelei se aplică un vector din mulțimea de antrenare.
  - Se calculează ieșirile neuronilor din stratul ascuns cu urmatoarea formula, f reprezentand functia sigmoida/ReLu:

$$y_j = f\left(\sum_{i=1}^{n+1} x_i \cdot w_{ij}\right).$$

  - Se calculeaza iesirile reale ale retelei, folosind formula:

$$y_k = f\left(\sum_{j=1}^{m+1} y_j \cdot w_{jk}\right),$$

  - Se actualizează eroarea medie pătratică pe epocă, folosind formula:

$$MSE = \frac{1}{K \cdot N} \cdot \sum_k \sum_n e_{kn}^2 = \frac{1}{K \cdot N} \cdot \sum_k \sum_n \left(y_{kn}^d - y_{kn}\right)^2 .$$

- **Propagarea inapoi**
  - Se calculează gradienții erorilor pentru neuronii din stratul de ieșire, avand formula:

$$\delta_k = f' \cdot e_k ,$$

unde f' este derivata funcției de activare iar eroarea $e_k = y_k^d - y_k$.

  - Se actualizează corecțiile ponderilor dintre stratul ascuns și stratul de ieșire, avand formula:

$$\Delta w_{jk} = \Delta w_{jk} + \alpha \cdot y_j \cdot \delta_k ,$$

- Se calculează gradienții erorilor pentru neuronii din stratul ascuns:

$$\delta_j = y_j \cdot (1 - y_j) \cdot \sum_{k=1}^{l} \delta_k \cdot w_{jk} ,$$

unde *l* este numărul de ieşiri ale rețelei.

- Se actualizează corecțiile ponderilor dintre stratul de intrare și stratul ascuns:

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

Modalitatea de rezolvare

Reteaua neuronala are urmatoarea structura:

- **Stratul de intrare:** Conține 4 neuroni, corespunzători celor 4 caracteristici ale florilor (sepal_length, sepal_width, petal_length, petal_width).
- **Stratul ascuns:** Contine 6 neuroni. Acesta aplică o funcție de activare (în acest caz, funcția sigmoid), pentru a introduce non-liniaritate în model.
- **Stratul de ieșire:** Conține 3 neuroni, corespunzători celor 3 clase posibile de varietăți ale florii Iris (Setosa, Versicolor, Virginica). În acest strat, utilizăm funcția de activare softmax pentru a calcula probabilitățile de apartenență ale fiecărei exemple la una dintre cele 3 clase.

Rezolvarea problemei începe cu încărcarea și pregătirea setului de date, urmată de împărțirea acestuia în seturi de antrenament și testare. Datele de intrare sunt apoi procesate prin rețea în timpul propagării înainte, unde sunt multiplicate cu ponderile și trecute prin funcțiile de activare pentru a obține o predicție. Eroarea dintre predicție și valoarea reală este calculată și propagată înapoi prin rețea pentru a ajusta ponderile și bias-urile folosind algoritmul de propagare inapoi. Acest proces se repetă pe mai multe epoci, îmbunătățind treptat capacitatea retelei de a face predicții corecte. După antrenare, rețeaua este capabilă să facă predicții asupra datelor noi, prin propagarea înainte a caracteristicilor și alegerea clasei cu probabilitatea cea mai mare. Modelul învață astfel să diferențieze între clasele din setul de date Iris și poate face predicții precise pe date noi.

Setul de date de intrare este impartit in 70% date de antrenare si 30% date de testare.

Parti semnificative din cod

- Propagarea inainte -implementează propagarea înainte în rețea. Primește intrările și le multiplică cu ponderile, adaugă bias-urile și aplică funcțiile de activare. Rezultatul este ieșirile stratului ascuns.

```python
# Propagarea semnalului inainte
def propagare_inainte(x, weights_input_hidden, bias_hidden, weights_hidden_output, bias_output):
    hidden_layer_input = np.dot(x, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    output_layer_output = softmax(output_layer_input)

    return hidden_layer_output, output_layer_output
```

- Propagarea inapoi - implementează propagarea înapoi (backpropagation), procesul prin care erorile sunt propagate înapoi prin rețea pentru a calcula gradienții și a actualiza ponderile. Calculează gradienții erorilor pentru fiecare strat și returnează corecțiile necesare ponderilor.Este esentiala pentru antrenarea retelei.

```python
# Propagarea erorilor inapoi
def propagare_inapoi(x, y, hidden_layer_output, output_layer_output, weights_hidden_output, learning_rate):
    # 4.1 Se calculează gradienții erorilor pentru neuronii din stratul de ieșire
    output_error = y - output_layer_output
    output_gradient = output_layer_output * (1 - output_layer_output) * output_error

    # 4.2 Se actualizează corecțiile ponderilor dintre stratul ascuns și stratul de ieșire
    delta_weights_hidden_output = learning_rate * np.dot(hidden_layer_output.T, output_gradient)
    bias_output_correction = learning_rate * np.sum(output_gradient, axis=0, keepdims=True)

    # 4.3 Se calculează gradienții erorilor pentru neuronii din stratul ascuns
    hidden_error = np.dot(output_gradient, weights_hidden_output.T)  # ∑(δk * wjk)
    hidden_gradient = hidden_layer_output * (1 - hidden_layer_output) * hidden_error  # δj = yj * (1 - yj) * ∑(...)

    # 4.4 Se actualizează corecțiile ponderilor dintre stratul de intrare și stratul ascuns
    delta_weights_input_hidden = learning_rate * np.dot(x.T, hidden_gradient)
    bias_hidden_correction = learning_rate * np.sum(hidden_gradient, axis=0, keepdims=True)

    return delta_weights_input_hidden, bias_hidden_correction, delta_weights_hidden_output, bias_output_correction
```

- Ajustarea ponderilor - este bucla principală de antrenare a modelului. Aceasta rulează pentru un număr definit de epoci și actualizează iterativ ponderile și bias-urile folosind propagarea înainte și înapoi. La fiecare epocă, calculează eroarea (MSE) și ajustează rețeaua în funcție de aceasta. La finalul antrenamentului, funcția returnează rezultatele pentru seturile de date de antrenament și testare.

```python
def ajustare_ponderi(delta_weights_input_hidden, bias_hidden_correction, delta_weights_hidden_output, bias_output_correction,
    weights_input_hidden, bias_hidden, weights_hidden_output, bias_output):
    # Actualizăm ponderile și bias-urile pentru stratul ascuns și stratul de ieșire
    weights_input_hidden += delta_weights_input_hidden
    bias_hidden += bias_hidden_correction

    weights_hidden_output += delta_weights_hidden_output
    bias_output += bias_output_correction

    return weights_input_hidden, bias_hidden, weights_hidden_output, bias_output
```

- Antrenarea modelului

```
def antreneaza_model():
    global weights_input_hidden, weights_hidden_output, bias_hidden, bias_output, model_trained

    losses = []

    for epoch in range(epochs):
        hidden_layer_output, output_layer_output = propagare_inainte(
            x_train, weights_input_hidden, bias_hidden, weights_hidden_output, bias_output
        )

        delta_weights_input_hidden, bias_hidden_correction, delta_weights_hidden_output, bias_output_correction = propagare_inapoi(
            x_train, y_train, hidden_layer_output, output_layer_output, weights_hidden_output, learning_rate
        )

        weights_input_hidden, bias_hidden, weights_hidden_output, bias_output = ajustare_ponderi(
            delta_weights_input_hidden, bias_hidden_correction,
            delta_weights_hidden_output, bias_output_correction,
            weights_input_hidden, bias_hidden, weights_hidden_output, bias_output
        )

        loss = calculeaza_mse(y_train, output_layer_output)
        losses.append(loss)

        if (epoch + 1) % 1000 == 0:
            print(f"Epoca {epoch + 1}/{epochs}, Pierdere: {loss:.4f}")
```

- Functia sigmoid- funcție de activare produce o ieșire între 0 și 1, utilă pentru a transforma valorile interne ale neuronilor în intervalul [0, 1], adică probabilități.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Este folosită pentru a activa neuronii din stratul ascuns al rețelei.

- Functia sigmoid_derivative - Derivata funcției sigmoid este folosită în propagarea înapoi pentru a calcula gradienții erorilor. Acesta este folosită pentru a ajusta ponderile și bias-urile în timpul învățării.

```
def sigmoid_derivative(x):
    return x * (1 - x)
```

- Functia reLu - o funcție de activare care returnează valoarea de intrare dacă este pozitivă și 0 dacă este negativă.

```
def relu(x):
    return np.maximum(0, x)
```

- Functia softmax - transformă un vector de valori într-un vector de probabilități, în care fiecare valoare este între 0 și 1, iar suma totală a acestora este 1. Este folosită pentru a obține probabilitățile claselor în problemele de clasificare. Este aplicată pe ieșirea stratului final al rețelei.

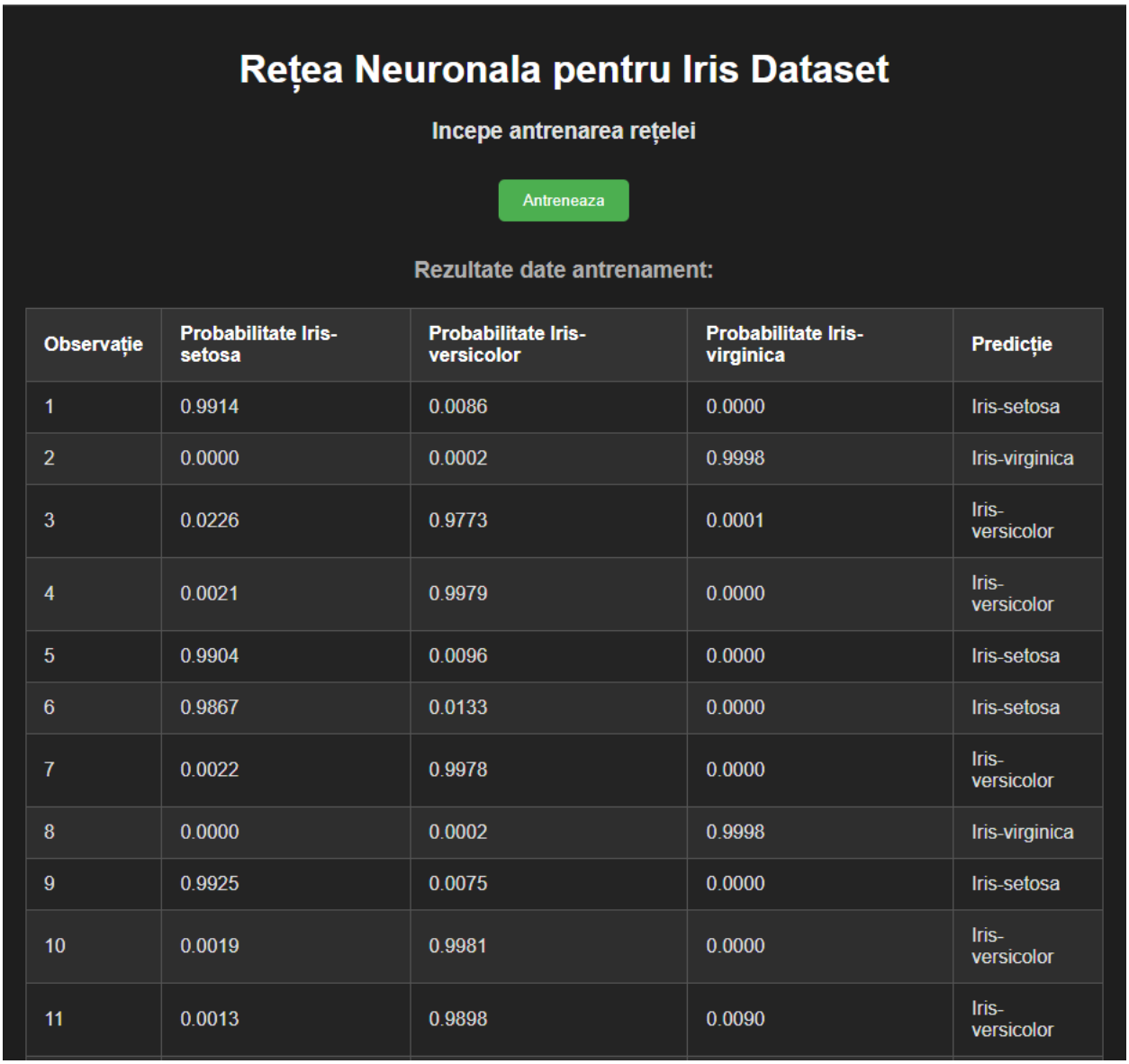
```
def softmax(x):   # transformă un vector de valori intr-un vector de probabilități, unde suma tuturor valorilor este 1
    e_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return e_x / np.sum(e_x, axis=1, keepdims=True)
```

- Functia calculeaza_mse - este folosită pentru a evalua performanța rețelei și pentru a ghida procesul de optimizare.

```
def calculeaza_mse(y_real, y_pred):
    mse = np.mean(np.sum((y_real - y_pred) ** 2, axis=1))
    return mse
```

Rezultatele obținute

Rezultate date antrenament



Rețea Neuronala pentru Iris Dataset

Incepe antrenarea rețelei

Antreneaza

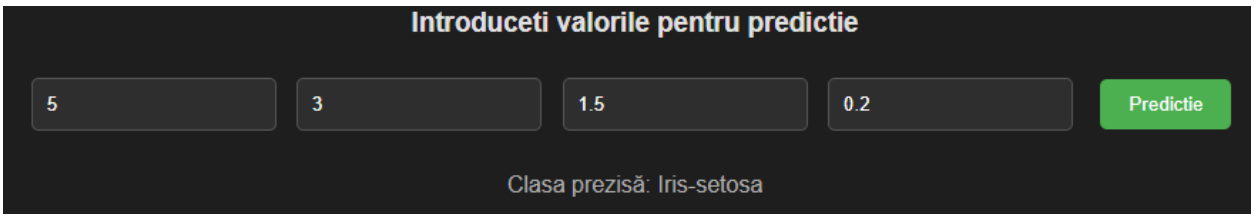
Rezultate date antrenament:

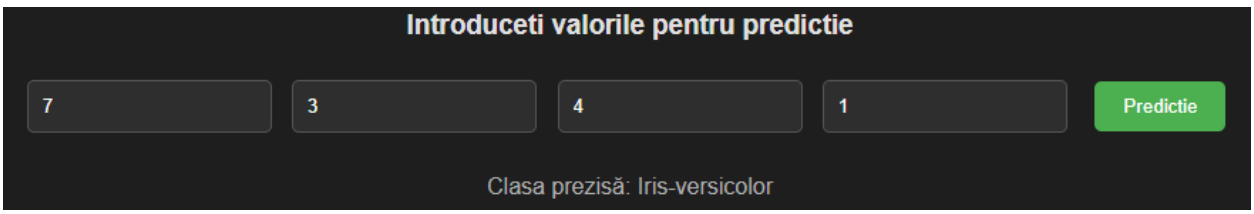
Observație	Probabilitate Iris-setosa	Probabilitate Iris-versicolor	Probabilitate Iris-virginica	Predicție
1	0.9914	0.0086	0.0000	Iris-setosa
2	0.0000	0.0002	0.9998	Iris-virginica
3	0.0226	0.9773	0.0001	Iris-versicolor
4	0.0021	0.9979	0.0000	Iris-versicolor
5	0.9904	0.0096	0.0000	Iris-setosa
6	0.9867	0.0133	0.0000	Iris-setosa
7	0.0022	0.9978	0.0000	Iris-versicolor
8	0.0000	0.0002	0.9998	Iris-virginica
9	0.9925	0.0075	0.0000	Iris-setosa
10	0.0019	0.9981	0.0000	Iris-versicolor
11	0.0013	0.9898	0.0090	Iris-versicolor

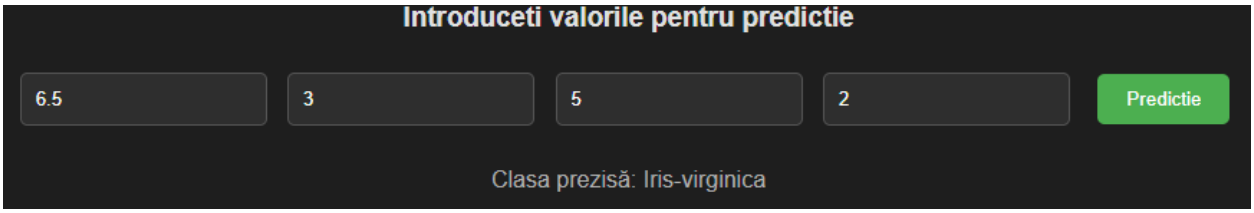
Rezultate date de testare

Observație	Probabilitate Iris-setosa	Probabilitate Iris-versicolor	Probabilitate Iris-virginica	Predicție
1	0.9922	0.0078	0.0000	Iris-setosa
2	0.9919	0.0081	0.0000	Iris-setosa
3	0.0000	0.0002	0.9998	Iris-virginica
4	0.9923	0.0077	0.0000	Iris-setosa
5	0.0000	0.0007	0.9993	Iris-virginica
6	0.0000	0.0044	0.9956	Iris-virginica
7	0.9923	0.0077	0.0000	Iris-setosa
8	0.9910	0.0090	0.0000	Iris-setosa
9	0.9913	0.0087	0.0000	Iris-setosa
10	0.0030	0.9970	0.0000	Iris-versicolor
11	0.0000	0.0002	0.9998	Iris-virginica
12	0.0007	0.9993	0.0000	Iris-versicolor
13	0.0170	0.9830	0.0000	Iris-versicolor
14	0.0028	0.9960	0.0012	Iris-versicolor

Rezultate exemplu predictie:







Concluzii

Acest proiect implementează o rețea neuronală artificială de tip perceptron multistrat pentru clasificarea datelor din setul Iris, utilizând un model cu un strat ascuns. Datele sunt preluate din setul Iris, care conține dimensiuni ale sepalei și petalei pentru trei specii de flori. Modelul folosește funcții de activare precum sigmoid și softmax pentru stratul ascuns și stratul de ieșire, respectiv. În timpul antrenării, se aplică propagarea înainte pentru calcularea ieșirilor și propagarea înapoi pentru ajustarea ponderilor și bias-urilor. Propagarea înapoi calculează erorile și gradienții utilizând derivatele funcțiilor de activare și actualizează parametrii rețelei pentru a minimiza eroarea totală. Rețeaua este evaluată pe un set de testare, iar rezultatele sunt exprimate sub formă de probabilități și predicții ale speciilor, permițând clasificarea noilor date Iris cu o acuratețe îmbunătățită.

Bibliografie

https://edu.tuiasi.ro/pluginfile.php/49456/mod_resource/content/9/IA11_RN1.pdf

https://archive.ics.uci.edu/dataset/53/iris

https://en.wikipedia.org/wiki/Multilayer_perceptron

Laborator 13 – IA

https://docs.python.org/