Algorytmy genetyczne i Sztuczne sieci neuronowe

Lista 6. - Implementacja jednowarstwowej sieci neuronowej

W ramach niniejszej listy zadań należy zaimplementować jednowarstwową sieć neuronową w Pythonie, zdolną do nauki operatora XOR. Realizacja zadania wymaga stworzenia kompletnego procesu uczenia, obejmującego przygotowanie danych, inicjalizację modelu, obliczanie propagacji w przód i wstecz oraz iteracyjne dostosowywanie wag.

Opis operatora XOR: Operator logiczny XOR (ang. *exclusive OR*) zwraca wartość prawda (1), jeśli dokładnie jedno z dwóch wejść jest prawdziwe (1), a drugie fałszywe (0). W przeciwnym razie zwraca wartość fałsz (0). Tabela prawdy dla operatora XOR przedstawia się następująco:

Wejście A	Wejście B	Wyjście XOR
0	0	0
0	1	1
1	0	1
1	1	0

Operator XOR jest nieliniowy, co sprawia, że jego nauka wymaga zastosowania sieci neuronowej z nieliniową funkcją aktywacji.

Zadania do wykonania:

1. **Opracować zbór danych dla operatora XOR:**
    - Przygotować zbór danych wejściowych i wyjściowych reprezentujący operator logiczny XOR:
        - Dane wejściowe (cechy): [(0, 0), (0, 1), (1, 0), (1, 1)]
        - Oczekiwane wyniki (etykiety): [0, 1, 1, 0]
    - Podzielić zbiór danych na cechy (X) i etykiety (y).

2. **Zainicjalizować parametry sieci:**
    - Zdefiniować strukturę sieci neuronowej:
        - Liczba neuronów w warstwie wejściowej: 2.
        - Liczba neuronów w warstwie wyjściowej: 1.
    - Zainicjalizować wagi i biasy losowymi wartościami (np. z rozkładu normalnego lub jednostajnego):
        - Wagi: macierz o wymiarach (2, 1).
        - Bias: wektor o wymiarze (1).

3. **Zaimplementować funkcje aktywacji:**
    - Zaimplementować funkcję sigmoidalną jako funkcję aktywacji:

$$\sigma(x) = 1/(1 + e^{-x})$$

    - Zaimplementować pochodną funkcji sigmoidalnej, potrzebną do obliczeń propagacji wstecznej:

$$d\sigma(x)/d(x) = \sigma(x) * (1 - \sigma(x))$$

4. **Zaimplementować propagację w przód:**
    - Obliczyć sumę ważoną wejścia:
    - Zastosować funkcję aktywacji, aby uzyskać wynik wyjściowy sieci:

5. **Zaimplementować propagację wstecz:**
    - Obliczyć błąd dla każdego przykładu:
    - Wyznaczyć gradienty dla wag i biasów:

6. **Zaimplementować aktualizację wag i biasów:**
    - Zaktualizować wagi i biasy z użyciem algorytmu spadku gradientu: Gdzie to współczynnik uczenia (np. 0.1).

7. **Zaimplementować pętlę uczenia:**
    - Iteracyjnie aktualizować wagi i biasy przez ustaloną liczbę epok (np. 10 000 iteracji).
    - Monitorować błąd średniokwadratowy (ang. *mean squared error*, *MSE*) w każdej epoce.

8. **Warunki opracowywanych rozwiązań:**
    - Opracowane rozwiązania powinny być implementowane w języku Python. Dopuszczalne jest wykorzystanie innego języka programowania pod warunkiem uzyskania zgody prowadzącego.
    - Podczas implementacji wolno posługiwać się bibliotekami do obliczeń numerycznych ogólnego przeznaczenia (np. NumPy, SciPy) oraz przetwarzania danych (np. Pandas).
    - Podczas implementacji nie wolno posługiwać się dedykowanymi bibliotekami do tworzenia algorytmów genetycznych (np. PyGAD, DEAP itp.) oraz architektur sieci neuronowych (np. scikit-learn, PyTorch, Tensorflow, JAX itp.).