Wrocław 10.12.2023

Jakub Piotrowski 266502

Sprawozdanie z Implementacji Sieci Neuronowej Realizującej Funkcję XOR

Wprowadzenie

Projekt obejmuje implementację sieci neuronowej do realizacji dwuargumentowej funkcji logicznej XOR. Zastosowano specyficzną architekturę sieci z dwoma warstwami: pierwszą ukrytą zawierającą dwa neurony i drugą wyjściową z jednym neuronem. Kluczowym elementem było wykorzystanie metody propagacji wstecznej z techniką momentum. Celem było skuteczne nauczenie sieci oraz analiza wpływu różnych parametrów na proces uczenia.

Architektura Sieci

- **Warstwy i Neurony:** Dwuwarstwowa sieć z łącznie trzema neuronami.
- **Połączenia:** Każdy neuron otrzymuje sygnały od obu argumentów XOR i bias.
- **Wagi:** Łącznie 9 wag (3 wagi na neuron).
- **Konstrukcja Neuronu:** Suma ważona wejść przetwarzana przez funkcję aktywacji.

Algorytm Uczenia

- **Metoda Uczenia:** Propagacja wsteczna.
- **Proces:** Przetwarzanie i aktualizacja wag na podstawie gradientu błędu, współczynnika uczenia i momentum.
- **Warunek Stopu:** Zastosowany do uniknięcia nadmiernego uczenia.

Symulacja i Dostrajanie Sieci

- **Eksperymenty:** Testy z różnymi warunkami początkowymi.
- **Optymalizacja Parametrów:** Badanie wpływu wartości początkowych wag, współczynnika uczenia, momentu i liczby epok.

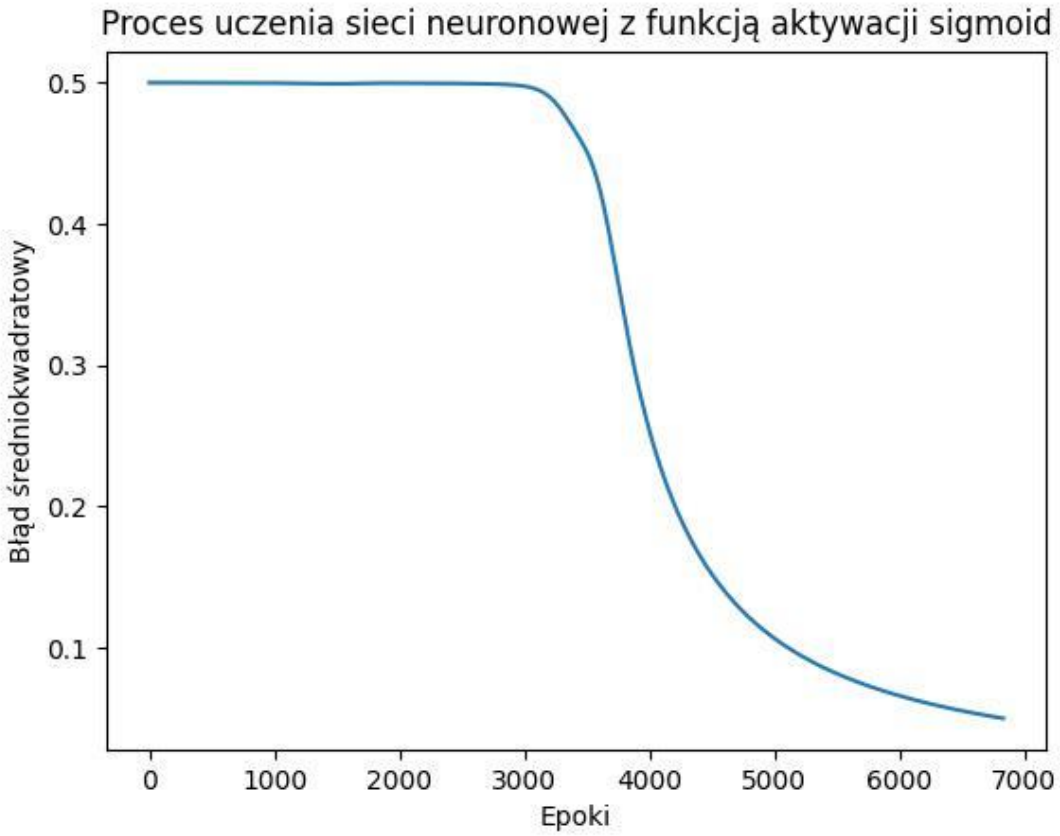
- **Porównanie Funkcji Aktywacji:** Efektywność funkcji sigmoid, tanh i relu.

Metodologia

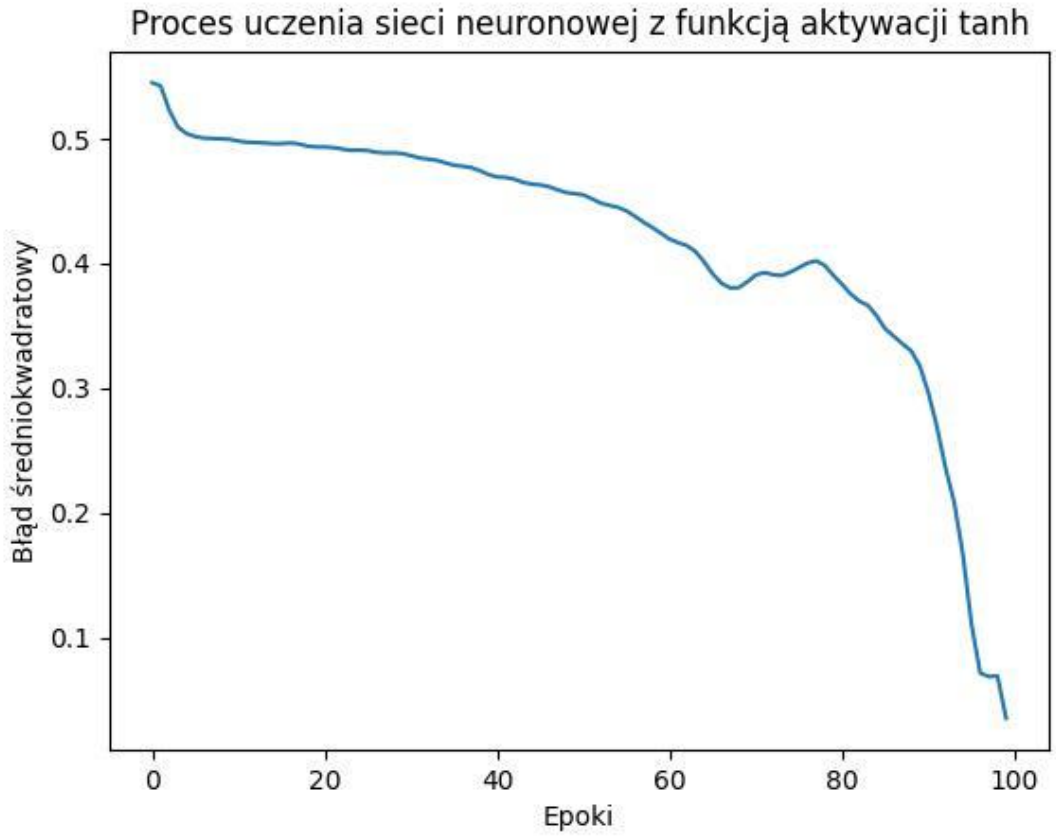
1. **Funkcje Aktywacji:** Sigmoid, tanh, relu.
2. **Inicjalizacja Wag:** Losowanie z zakresu (-1, 1), próba zastosowania inicjalizacji Xavier/Glorot.
3. **Parametry Eksperymentalne:**
    - Współczynnik uczenia: 0.1
    - Liczba epok: 10000
    - Momentum: 0.95
    - Minimalny błąd: 0.05
4. **Testowanie Sieci:** Możliwość przetestowania sieci z różnymi zestawami wejściowymi.

Wyniki Uczenia i Testowania

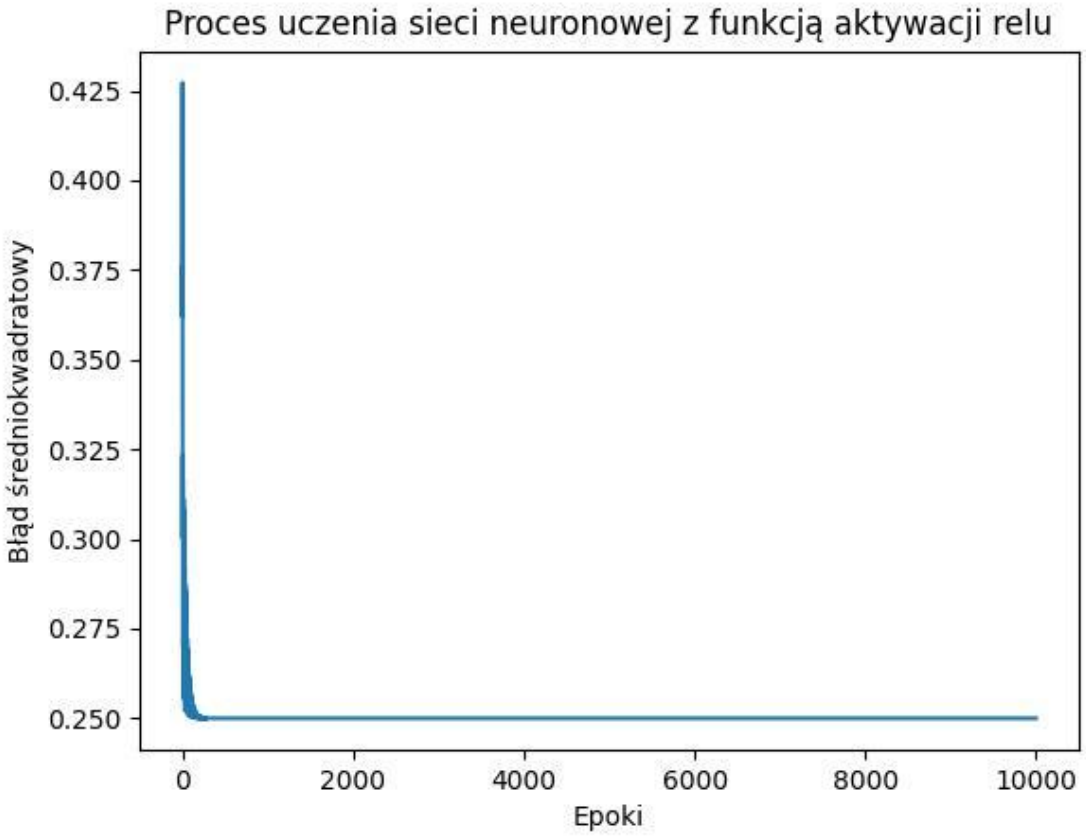
- **Proces Uczenia:** Wykresy zmian błędu dla każdej funkcji aktywacji.
- **Czas Uczenia:** Rejestrowany dla każdej funkcji aktywacji.
- **Błąd Końcowy:** Określony dla każdej funkcji aktywacji.
- **Testy Sieci:**
    1. **Sigmoid:**
        - Test:
            - [0, 0] -> 0.0451,
            - [0, 1] -> 0.9448,
            - [1, 0] -> 0.9448,
            - [1, 1] -> 0.0445
        - Czas Uczenia: 0.54 sekundy
        - Błąd Końcowy: 0.0500



2. **Tanh:**
    - Test:
        - [0, 0] -> 0.0061,
        - [0, 1] -> 0.9543,
        - [1, 0] -> 0.9656,
        - [1, 1] -> -0.0779
    - Czas Uczenia: 0.01 sekundy
    - Błąd Końcowy: 0.0351



3. **ReLU:**
    - Test:
        - [0, 0] -> 0.0,
        - [0, 1] -> 0.0,
        - [1, 0] -> 1.0,
        - [1, 1] -> 0.0
    - Czas Uczenia: 0.75 sekundy
    - Błąd Końcowy: 0.2500



Analiza Wyników

- **Skuteczność Funkcji Aktywacji:** Sigmoid i tanh okazały się bardziej efektywne w modelowaniu funkcji XOR niż ReLU.
- **Czas Uczenia:** Tanh wykazała się najkrótszym czasem uczenia.
- **Błąd Końcowy:** Niski błąd końcowy w przypadku sigmoid i tanh wskazuje na ich skuteczność.

Wnioski

- **Wybór Funkcji Aktywacji:** Jest kluczowy dla skuteczności sieci neuronowej. Sigmoid i tanh lepiej radzą sobie z funkcją XOR niż ReLU.
- **Optymalizacja Parametrów:** Istotna dla efektywności i skuteczności sieci.
- **Eksperymenty:** Dostarczają cennych informacji i wskazówek dla przyszłych badań.

Dodatkowe Uwagi

- **Implementacja w Pythonie:** Użycie biblioteki NumPy zapewnia łatwość modyfikacji i testowania różnych konfiguracji.
- **Przyszłe Kierunki Badań:** Zalecane są dalsze badania z użyciem różnych konfiguracji sieci.

Zastrzeżenia

- Wyniki mogą się różnić w zależności od warunków eksperymentu i implementacji. Niniejsze sprawozdanie dostarcza podstawowych wskazówek i wniosków.

Bibliografia:

- *https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/*
- *https://towardsdatascience.com/deep-learning-with-python-neural-networks-complete-tutorial-6b53c0b06af0*
- *https://www.ibm.com/docs/pl/spss-modeler/saas?topic=SS3RA7_sub/modeler_mainhelp_client_ddita/clementine/trainnetnode_general.htm*