Zadanie nr 3 - uczenie i testowanie sieci MLP

Inteligentna Analiza Danych

Karol Kazusek - 254189, Sebastian Zych - 254264

21.10.2024

1 Cel zadania

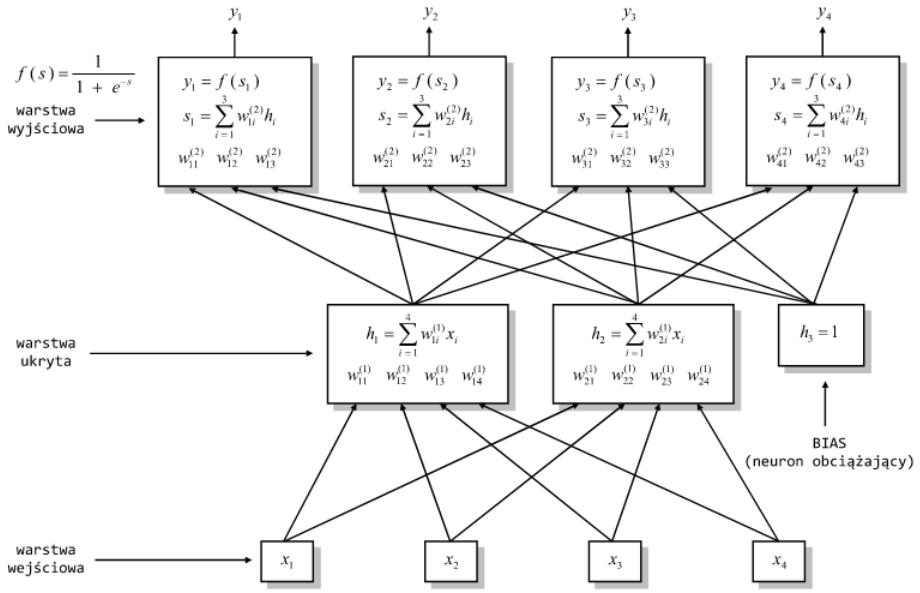
Celem zadania było zaimplementowanie i przetestowanie sieci wielowarstwowego perceptronu (MLP) oraz zbadanie wpływu wartości biasu oraz liczby epok na dokładność predykcji i proces uczenia się sieci.

2 Wstęp teoretyczny

2.1 Wielowarstwowy perceptron (MLP)

Wielowarstwowy perceptron (ang. *Multilayer Perceptron, MLP*) to typ sieci neuronowej typu feedforward, składający się z co najmniej jednej warstwy ukrytej między warstwą wejściową a warstwą wyjściową. Sieci MLP są zdolne do rozwiązywania problemów nieliniowo separowalnych, co czyni je odpowiednimi do zadań bardziej złożonych niż perceptron jednopoziomowy.

2.1.1 Architektura MLP



Rysunek 1: Przykład architektury wielowarstwowego perceptronu (MLP)

Podstawowa architektura MLP składa się z następujących elementów:

- **Warstwa wejściowa** - zawiera neurony odpowiadające cechom wejściowym. Liczba neuronów w tej warstwie jest równa liczbie cech opisujących dane wejściowe.

- **Warstwa ukryta (jedna lub więcej)** - neurony w tej warstwie przekształcają dane wejściowe przy pomocy wag i biasów. Zazwyczaj w warstwie ukrytej stosuje się funkcję aktywacji nieliniową, np. sigmoidalną.
- **Warstwa wyjściowa** - dostarcza wyniki sieci. W przypadku klasyfikacji, liczba neuronów odpowiada liczbie klas, natomiast w przypadku regresji może być jeden neuron.

2.2 Algorytm backpropagation (wsteczna propagacja błędów)

Algorytm *backpropagation* (wsteczna propagacja błędów) jest podstawowym mechanizmem uczenia sieci MLP. Jest to technika nadzorowanego uczenia, która modyfikuje wagi i biasy w sieci w celu minimalizacji różnicy między przewidywanymi a oczekiwanymi wynikami. Proces ten składa się z dwóch głównych etapów:

1. **Forward pass** - Przepływ sygnału przez sieć, podczas którego sieć generuje przewidywane wyniki na podstawie wag i biasów.
2. **Backward pass** - Propagacja błędu wstecznie w celu aktualizacji wag i biasów zgodnie z zasadami gradientu prostego.

2.2.1 Obliczanie błędu i gradientu

Algorytm backpropagation wykorzystuje metodę gradientu prostego do aktualizacji wag. Celem jest minimalizacja funkcji straty, najczęściej używaną funkcją dla regresji jest średni błąd kwadratowy (MSE), a dla klasyfikacji funkcja krzyżowej entropii. Funkcja kosztu dla MSE jest definiowana jako:

$$L = \frac{1}{2} \sum_i (y^{(i)}_{predicted} - y^{(i)}_{expected})^2$$

Następnie obliczane są pochodne cząstkowe funkcji straty względem wag, a wagi są aktualizowane zgodnie z kierunkiem przeciwnym do gradientu:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W}$$

gdzie η to współczynnik uczenia (learning rate).

2.2.2 Rola biasów w MLP

Biasy są kluczowym elementem perceptronów i MLP. Umożliwiają one przesunięcie funkcji aktywacji, co pozwala sieci lepiej dopasować się do danych. Bez biasów sieć MLP mogłaby mieć ograniczoną zdolność do efektywnego uczenia się, zwłaszcza gdy wymagane jest dopasowanie funkcji, które nie przechodzą przez początek układu współrzędnych.

3 Opis implementacji

Zaimplementowana klasa MLP obejmuje:

- Inicjalizację wag sieci losowymi wartościami.
- Propagację sygnału przez sieć przy użyciu funkcji forward.
- Wsteczną propagację błędów (backpropagation) przy użyciu funkcji backward, co umożliwia aktualizację wag.
- Ustawienie wartości bias w warstwie wyjściowej, które może być konfigurowane.

Trening sieci przeprowadzono na zadanym zbiorze danych, a błędy monitorowano w trakcie procesu uczenia.

4 Eksperymenty i wyniki

4.1 Zbiór danych

Dane wejściowe i wyjściowe użyte do treningu i testowania sieci MLP to proste wzorce binarne. Każdy z czterech wzorców wejściowych odpowiada jednemu neuronowi wejściowemu, a każdy wzorzec wyjściowy reprezentuje jedno oczekiwane wyjście.

- **Wejścia (inputs)**:

$$inputs = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Oczekiwane wyjścia (expected outputs)**:

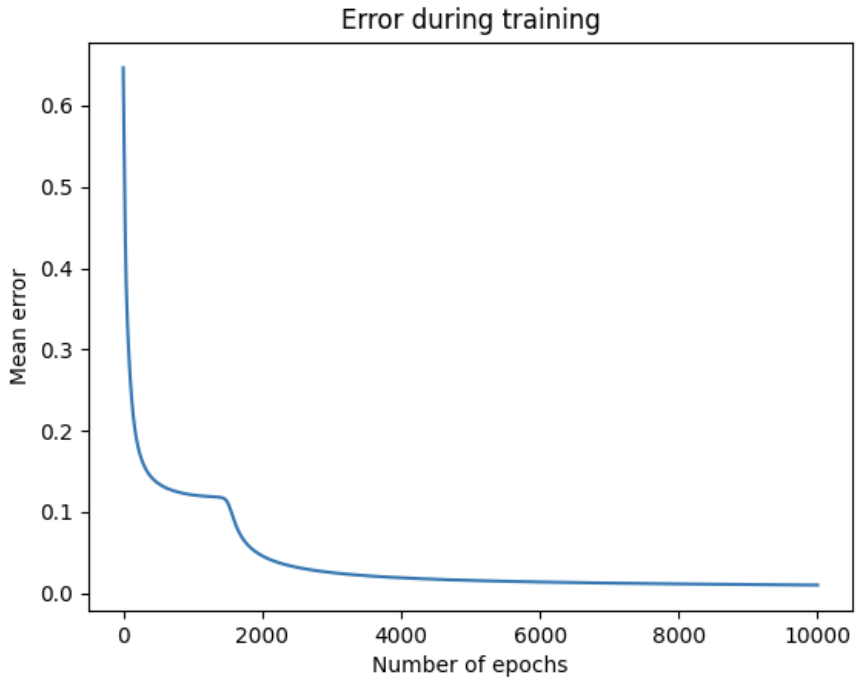
$$expected\_output = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2 Eksperyment 1: Trening MLP z bias ustawionym na 1

W pierwszym eksperymencie przeprowadzono trening sieci MLP z biasem ustawionym na 1. Sieć miała 4 neurony wejściowe, 2 neurony w warstwie ukrytej oraz 4 neurony w warstwie wyjściowej. Trening odbywał się przez 10000 epok z współczynnikiem uczenia (learning rate) równym 0.1.

- **Parametry sieci**:
  - Liczba neuronów wejściowych: 4
  - Liczba neuronów w warstwie ukrytej: 2
  - Liczba neuronów w warstwie wyjściowej: 4
  - Współczynnik uczenia: 0.1
  - Bias: 1
  - Liczba epok: 10000

Po zakończeniu treningu, sieć była w stanie poprawnie odwzorować dane wejściowe. Ostateczne przewidywane wyjścia były następujące:



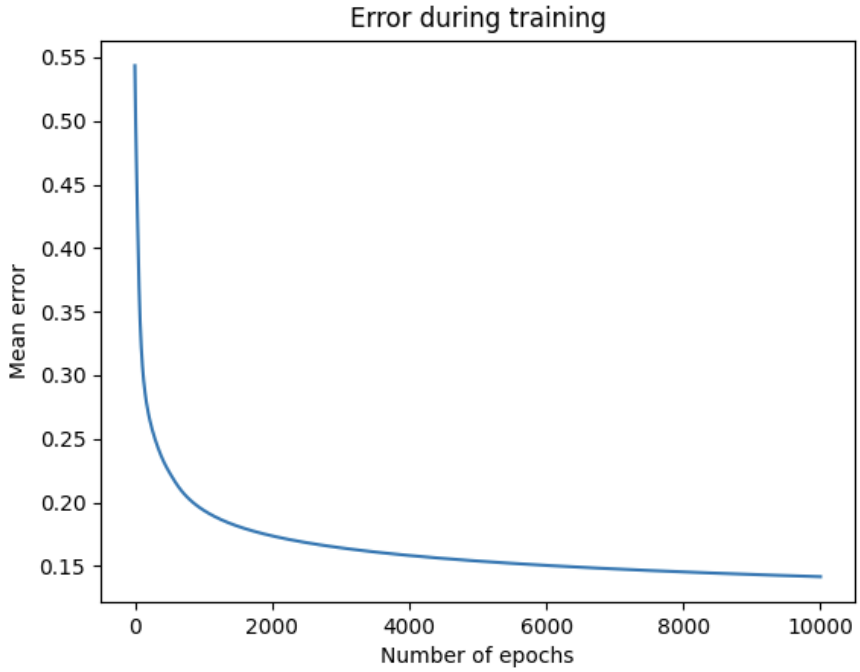
Rysunek 2: Wykres błędu w trakcie treningu z bias = 1

$$predicted\_output = \begin{bmatrix} 0.986 & 0.001 & 0.009 & 0.000 \\ 0.024 & 0.995 & 0.000 & 0.015 \\ 0.018 & 0.000 & 0.991 & 0.012 \\ 0.000 & 0.018 & 0.028 & 0.991 \end{bmatrix}$$

4.3 Eksperyment 2: Trening MLP bez biasu (bias = 0)

W drugim eksperymencie przeprowadzono trening sieci bez biasu (bias = 0). Sieć używała tych samych danych co w eksperymencie 1, jednak brak biasu wpłynął na wyniki.

- **Parametry sieci**:
  - Liczba neuronów wejściowych: 4
  - Liczba neuronów w warstwie ukrytej: 2
  - Liczba neuronów w warstwie wyjściowej: 4
  - Współczynnik uczenia: 0.1
  - Bias: 0
  - Liczba epok: 10000



Rysunek 3: Wykres błędu w trakcie treningu bez biasu

Brak biasu spowodował, że sieć miała trudności z dokładnym odwzorowaniem wzorców. Ostateczne przewidywane wyjścia były mniej precyzyjne:

$$predicted\_output = \begin{bmatrix} 0.852 & 0.049 & 0.000 & 0.000 \\ 0.618 & 0.852 & 0.000 & 0.169 \\ 0.000 & 0.000 & 0.950 & 0.148 \\ 0.169 & 0.000 & 0.148 & 0.381 \end{bmatrix}$$

5 Wnioski

1. **Skuteczność sieci z biasem**:
   Sieć z biasem ustawionym na 1 była w stanie skutecznie odwzorować zadane wzorce wejściowe. Wyniki były bliskie oczekiwanym wartościom, co świadczy o efektywnym procesie uczenia.

2. **Wpływ braku biasu**:
   Trening bez biasu prowadził do większych odchyleń w wynikach. Sieć miała trudności z dokładnym odwzorowaniem wzorców, co sugeruje, że bias odgrywa kluczową rolę w zwiększeniu elastyczności modelu.

3. **Czy perceptron byłby w stanie być skutecznie wytrenowany, gdyby neurony w warstwie wyjściowej nie posiadały wejść obciążających (BIAS)?**
   Bez wejść obciążających (BIAS) w warstwie wyjściowej perceptron miałby ograniczoną zdolność dopasowania się do danych. Bias pozwala na przesunięcie funkcji aktywacji, co umożliwia lepsze odwzorowanie zależności w danych wejściowych. Brak biasu może prowadzić do niedostatecznej elastyczności sieci, co może utrudniać trening.

4. **Jak można zinterpretować rolę wejść obciążających (BIAS) w neuronach warstwy wyjściowej perceptronu?**
   Bias działa podobnie do wyrazu wolnego w równaniach liniowych, pozwalając na przesunięcie funkcji aktywacji w osi poziomej. Dzięki temu neuron może aktywować się, nawet gdy wszystkie wejścia mają wartość 0. Umożliwia to lepsze dopasowanie modelu do danych i modelowanie bardziej skomplikowanych funkcji.