***SPRAWOZDANIE PODSTAWY UCZENIA MASZYNOWEGO***

***RAFAŁ POPŁAWSKI***

**KOD:**

import numpy as np

import matplotlib.pyplot as plt

# Inicjalizacja wag losowo

def inicjalizuj\_wagi(num\_inputs, num\_hidden1, num\_hidden2):

    weights1 = np.random.uniform(-0.1, 0.1, size=(num\_inputs + 1, num\_hidden1))

    weights2 = np.random.uniform(-0.1, 0.1, size=(num\_hidden1 + 1, num\_hidden2))

    return weights1, weights2

# Funkcja sigmoidalna (funkcja aktywacji)

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

# Przeprowadzenie propagacji w przód przez sieć neuronową

def propagacja\_w\_przod(weights1, weights2, input\_data):

    beta = 5

    input\_with\_bias = np.concatenate(([-1], input\_data))

    hidden1\_output = sigmoid(beta \* np.dot(weights1.T, input\_with\_bias))

    hidden1\_with\_bias = np.concatenate(([-1], hidden1\_output))

    hidden2\_output = sigmoid(beta \* np.dot(weights2.T, hidden1\_with\_bias))

    return hidden1\_output, hidden2\_output

# Trenowanie sieci neuronowej

def trenuj(weights1\_initial, weights2\_initial, input\_data, target\_data, num\_iterations):

    num\_examples = input\_data.shape[1]

    weights1 = weights1\_initial

    weights2 = weights2\_initial

    learning\_rate = 0.1

    beta = 5

    errors1 = []

    errors2 = []

    for \_ in range(num\_iterations):

        example\_index = np.random.randint(num\_examples)

        input\_example = input\_data[:, example\_index]

        hidden1\_output, hidden2\_output = propagacja\_w\_przod(weights1, weights2, input\_example)

        hidden1\_with\_bias = np.concatenate(([-1], hidden1\_output))

        delta2 = target\_data[example\_index] - hidden2\_output

        error2 = beta \* delta2 \* hidden2\_output \* (1 - hidden2\_output)

        delta1 = np.dot(weights2[1:, :], error2)

        error1 = beta \* delta1 \* hidden1\_output \* (1 - hidden1\_output)

        dW1 = learning\_rate \* np.outer(np.concatenate(([-1], input\_example)), error1)

        dW2 = learning\_rate \* np.outer(hidden1\_with\_bias, error2)

        weights1 += dW1

        weights2 += dW2

        errors1.append(np.mean(np.abs(error1)))

        errors2.append(np.mean(np.abs(error2)))

    return weights1, weights2, errors1, errors2

# Dane wejściowe i docelowe

input\_data = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])

target\_data = np.array([0, 1, 1, 0])

# Architektura sieci i parametry treningowe

num\_inputs = 2

num\_hidden1 = 2

num\_hidden2 = 1

num\_iterations = 5000

# Inicjalizacja wag

weights1\_initial, weights2\_initial = inicjalizuj\_wagi(num\_inputs, num\_hidden1, num\_hidden2)

# Trenowanie sieci

weights1\_final, weights2\_final, errors1, errors2 = trenuj(weights1\_initial, weights2\_initial, input\_data, target\_data, num\_iterations)

# Wykres błędów

plt.plot(errors1, label='Warstwa 1')

plt.plot(errors2, label='Warstwa 2')

plt.title('Błędy w kolejnych iteracjach')

plt.xlabel('Iteracja')

plt.ylabel('Błąd')

plt.legend()

plt.show()

**2. WYKRES**

Obraz zawierający tekst, zrzut ekranu, Wykres, diagram

Opis wygenerowany automatycznie

**3. PODSUMOWANIE:**

Najpierw importujemy niezbędne biblioteki: numpy do obliczeń numerycznych i matplotlib.pyplot do generowania wykresów. Następnie definiujemy funkcję initialize\_weights, która inicjalizuje wagi sieci neuronowej losowymi wartościami z zakresu -0,1 do 0,1. Funkcja sigmoidalna jest funkcją aktywacji, która konwertuje wartości wejściowe na wartości z zakresu (0, 1) za pomocą funkcji logistycznej. forward\_propagation przekształca dane wejściowe przez ukryte warstwy przy użyciu wag i sigmoidalnej funkcji aktywacji. Funkcja trenuj trenuje sieć neuronową. Wykorzystuje losowe próbki z danych treningowych i wykonuje propagację do przodu i do tyłu, aktualizując wagi na podstawie błędów przewidywania. Powtarza ten proces dla określonej liczby iteracji. Następnie definiujemy input\_data i target\_data potrzebne do uczenia sieci. Definiujemy architekturę sieci oraz parametry treningowe. Inicjalizujemy wagi początkowe za pomocą funkcji initialize\_weights. Uczenie sieci neuronowej przeprowadzamy za pomocą funkcji trenuj, która zwraca ostateczną listę wag i błędów dla każdej warstwy. Na koniec iteracyjnie generujemy wykresy błędów dla każdej warstwy przy użyciu biblioteki matplotlib.pyplot.