Czym jest MLP?

MLP (Multi-Layer Perceptron) to rodzaj sztucznej sieci neuronowej, która składa się z wielu warstw neuronów. Każdy neuron w jednej warstwie jest połączony ze wszystkimi neuronami w kolejnej warstwie. MLP przekształca dane wejściowe i produkuje wynik poprzez szereg obliczeń matematycznych.

Jak działa neuron?
Każdy neuron w MLP działa następująco: pobiera kilka wejść, mnoży je przez odpowiednie **wagi** i dodaje pewną **wartość progową** (bias). Następnie wynik przechodzi przez **funkcję aktywacji**, która decyduje, jak silny powinien być sygnał przekazany dalej.
Matematycznie można to zapisać jako:

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$$

gdzie:

- x1,x2,...,xn – wejścia do neuronu (np. piksele obrazu, cechy tekstu),
- w1,w2,...,wn– wagi połączeń (mówią, jak ważne są dane wejściowe),
- b – bias (wartość przesunięcia),
- z – suma ważona wejść.

Funkcja aktywacji
Po obliczeniu z, neuron przepuszcza go przez **funkcję aktywacji**, która wprowadza nieliniowość do modelu (dzięki czemu MLP może uczyć się bardziej złożonych wzorców).

Warstwy w MLP
MLP składa się z kilku warstw neuronów:

1. **Warstwa wejściowa** – przyjmuje dane wejściowe
2. **Warstwy ukryte** – przekształcają dane za pomocą wag i funkcji aktywacji.
3. **Warstwa wyjściowa** – generuje końcowy wynik (np. klasyfikacja do kilku kategorii).

Działanie każdej warstwy można zapisać jako:

$$h = f(Wx + b)$$

gdzie:

- x – wejście do warstwy,
- W – macierz wag,
- b – bias,
- f – funkcja aktywacji,
- h – wyjście warstwy.

Przepływ informacji (Forward Propagation)

1. Dane wejściowe x przechodzą przez pierwszą warstwę neuronów.
2. Każdy neuron oblicza z i przepuszcza go przez funkcję aktywacji.
3. Wynik jest przekazywany do kolejnej warstwy.
4. Proces powtarza się, aż dotrzemy do warstwy wyjściowej.

Implementacja:

Proponuję zrobić implementację w postaci klasy i poniższych metod:

```python
import numpy as np

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

def linear(z):
    return z

class MLPNoBackprop:
    def __init__(self, layer_sizes,
                 hidden_activation='sigmoid',
                 output_activation='linear'):
        """
        layer_sizes: lista z rozmiarami warstw, np. [1, 5, 1]
                     1 -> wymiar wejścia,
                     5 -> liczba neuronów w warstwie ukrytej,
                     1 -> liczba neuronów w warstwie wyjściowej.
        hidden_activation: 'sigmoid' lub 'linear' - aktywacja w warstwach ukrytych
        output_activation: 'sigmoid' lub 'linear' - aktywacja w warstwie wyjściowej
        """


    def forward(self, X):
        """
        Przejście w przód przez sieć.
        X: macierz wejściowa (liczba_próbek x wymiar_wejścia) lub wektor (1D).

        Zwraca listę kolejnych aktywacji oraz ostateczne wyjście sieci.
        """

        return activations

    def predict(self, X):
        """
        Zwraca tylko końcową odpowiedź sieci (ostatnią aktywację).
        """
        return self.forward(X)[-1]

    def set_weights_and_biases(self, layer_idx, W, b):
        """
        Pozwala ręcznie ustawić wagi i biasy w warstwie o indeksie layer_idx.
        layer_idx: numer warstwy (0 = pierwsze połączenie, 1 = drugie, itp.)
        W: nowa macierz wag (numpy array o wymiarach [layer_sizes[layer_idx], layer_sizes[layer_idx+1]])
        b: nowy wektor biasów (numpy array o wymiarach [layer_sizes[layer_idx+1], ])
        """
        self.weights[layer_idx] = W
        self.biases[layer_idx] = b

    def mse(self, y_true, y_pred):
        """
        Błąd średniokwadratowy (MSE).
        """
        return np.mean((y_true - y_pred)**2)
```

Przykładowe wywołanie dla sztucznych danych – Państwo powinni dokonać testów sieci na danych opisanych w zadaniu:

```
# 1. Tworzymy sieć MLP z architekturą 1 -> 5 -> 1:
mlp = MLPNoBackprop(layer_sizes=[1, 5, 1],
                    hidden_activation='sigmoid',
                    output_activation='linear')

# 2. Przygotowujemy sztuczne dane (np. wektor wejść X od 0 do 1)
X = np.linspace(0, 1, 5).reshape(-1, 1)  # 5 punktów wejściowych
y_true = X * 2.0  # Przykładowa funkcja docelowa y=2x

# 3. (Opcjonalnie) Ustawiamy ręcznie wybrane wagi w warstwie 0 i warstwie 1
#    Oczywiście kształty macierzy W0 i W1 muszą się zgadzać:
#    - W warstwie 0: shape [1, 5], b0 shape [5]
#    - W warstwie 1: shape [5, 1], b1 shape [1]

# Na potrzeby przykładu inicjalizujemy je losowo, a potem np. lekko modyfikujemy
W0, b0 = mlp.weights[0], mlp.biases[0]
W1, b1 = mlp.weights[1], mlp.biases[1]

# Załóżmy drobne zmiany "ręczne":
b0 += 0.1
b1 += 0.2
mlp.set_weights_and_biases(0, W0, b0)
mlp.set_weights_and_biases(1, W1, b1)

# 4. Obliczamy predykcje i błąd
y_pred = mlp.predict(X)
mse_error = mlp.mse(y_true, y_pred)

print("Wejścia (X):\n", X.ravel())
print("Prawdziwe (y_true):\n", y_true.ravel())
print("Predykcje (y_pred):\n", y_pred.ravel())
print("MSE =", mse_error)
```