

# WSI

## Zadanie 5

### Perceptron wielowarstwowy

---

Łukasz Borowski 331159  
Bartosz Czerwiński 331165

20 maja 2025

## 1. Wstęp

Ćwiczenie polegało na implementacji perceptronu wielowarstwowego oraz dowolnego algorytmu optymalizacji gradientowej, przy użyciu metody propagacji wstecznej.

## 2. Opis implementacji

W pliku MLP.py zaimplementowano perceptron używający metody stochastycznego spadku gradientowego z mini-batchami, klasycznym sigmoidem i dowolną liczbą warstw ukrytych i neuronów w tych warstwach. Do testów użyto dwóch warstw, z 32 i 16 neuronami w pierwszej i drugiej warstwie. Funkcja kosztu była obliczana przy użyciu metody cross-entropy.

## 3. Ulepszona wersja

### 3.1. Regularyzacja L2

Regularyzacja L2 polega na zwiększeniu funkcji kosztu na podstawie wag, pozwala to wyeliminować duże wagi, co zapobiega przetrenowaniu. Zmiana została zaimplementowana w funkcji `compute_cost`.

```
def compute_cost(self, AL, Y):  
  
    m = Y.shape[1]  
    # dodaj epsilon dla stabilnosci  
    eps = 1e-8  
    cost = -np.sum(Y * np.log(AL + eps)) / m  
    return cost
```

Kod 1. Funkcja `compute_cost` w zwykłym modelu

```
def compute_cost(self, AL, Y, lambd=0.01):

    m = Y.shape[1]
    eps = 1e-8
    cross_entropy = -np.sum(Y * np.log(AL + eps)) / m
    # Regularyzacja L2 dla wszystkich wag
    L2_cost = 0
    for l in range(1, self.L + 1):
        L2_cost += np.sum(self.W[l] ** 2)
    L2_cost = (lambd / (2 * m)) * L2_cost
    return cross_entropy + L2_cost
```

Kod 2. Rozszerzona funkcja compute\_cost

Jak widać, do kosztu związanego z entropią została dodana suma kwadratów wag, pomnożona przez czynnik  $\frac{\lambda}{2m}$ , gdzie  $\lambda$  to hiperparametr modelu.

### 3.2. Dropout

Dropout, czyli losowe odrzucanie neuronów w celu zapobiegnięcia przeuczeniu, zaimplementowano w funkcji forward.

```
def forward(self, X):

    cache = {'AO': X}
    A = X
    # warstwy ukryte: sigmoid
    for l in range(1, self.L):
        Z = self.W[l] @ A + self.b[l]
        A = self.sigmoid(Z)
        cache[f'Z{l}'], cache[f'A{l}'] = Z, A
    # warstwa wyjściowa: softmax
    ZL = self.W[self.L] @ A + self.b[self.L]
    AL = self.softmax(ZL)
    cache[f'Z{self.L}'], cache[f'A{self.L}'] = ZL, AL
    return AL, cache
```

Kod 3. Funkcja forward w zwykłym modelu

```
def forward(self, X, is_training=True, keep_prob=0.8):

    cache = {'AO': X}
    A = X
    # Warstwy ukryte: sigmoid + dropout
    for l in range(1, self.L):
        Z = self.W[l] @ A + self.b[l]
        A = self.sigmoid(Z)
        if is_training:
            D = np.random.rand(*A.shape) < keep_prob
            A *= D / keep_prob # Skalowanie dla zachowania oczekiwanej wartosci
            cache[f'D{l}'] = D
        cache[f'Z{l}'], cache[f'A{l}'] = Z, A
    # Warstwa wyjściowa: softmax (bez dropoutu)
    ZL = self.W[self.L] @ A + self.b[self.L]
    AL = self.softmax(ZL)
    cache[f'Z{self.L}'], cache[f'A{self.L}'] = ZL, AL
    return AL, cache
```

Kod 4. Rozszerzona funkcja forward

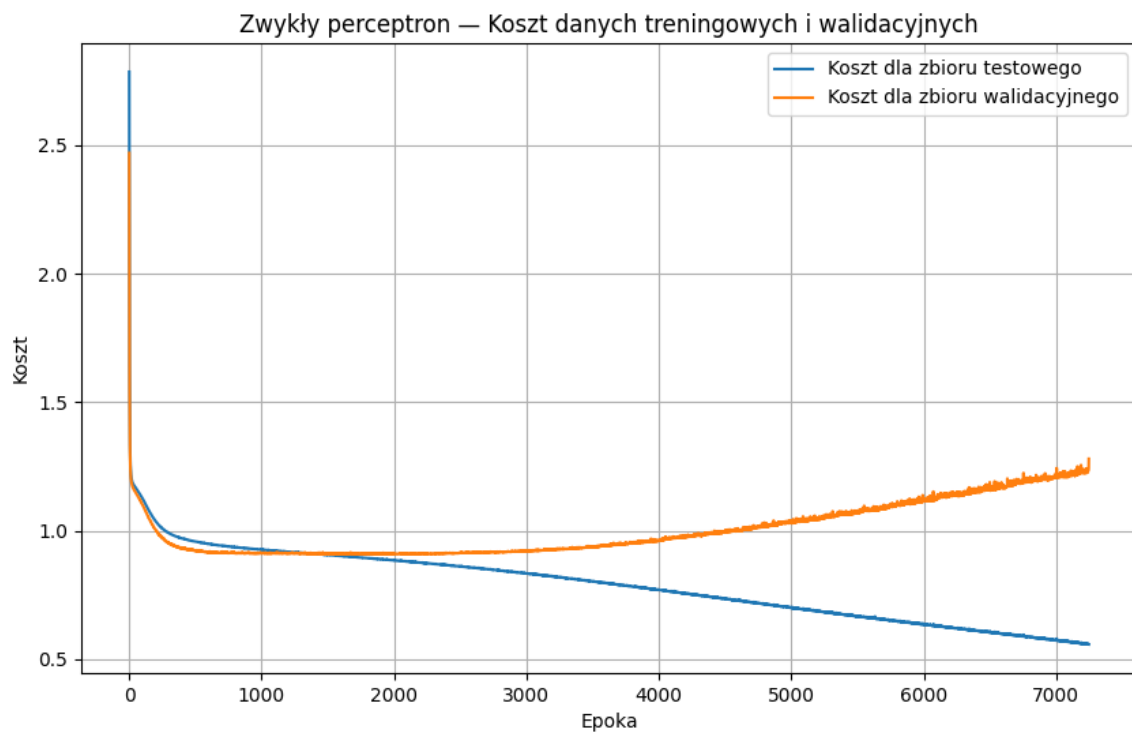
Do funkcji **forward** przy trenowaniu sieci dodano losowe wybranie zbioru neuronów D będącego podzbiorem pierwotnego zbioru neuronów A, z prawdopodobieństwem wyboru określonym jako hiperparametr modelu. Funkcja kosztu zostaje wtedy przeskalowana dla zachowania oczekiwanej wartości.

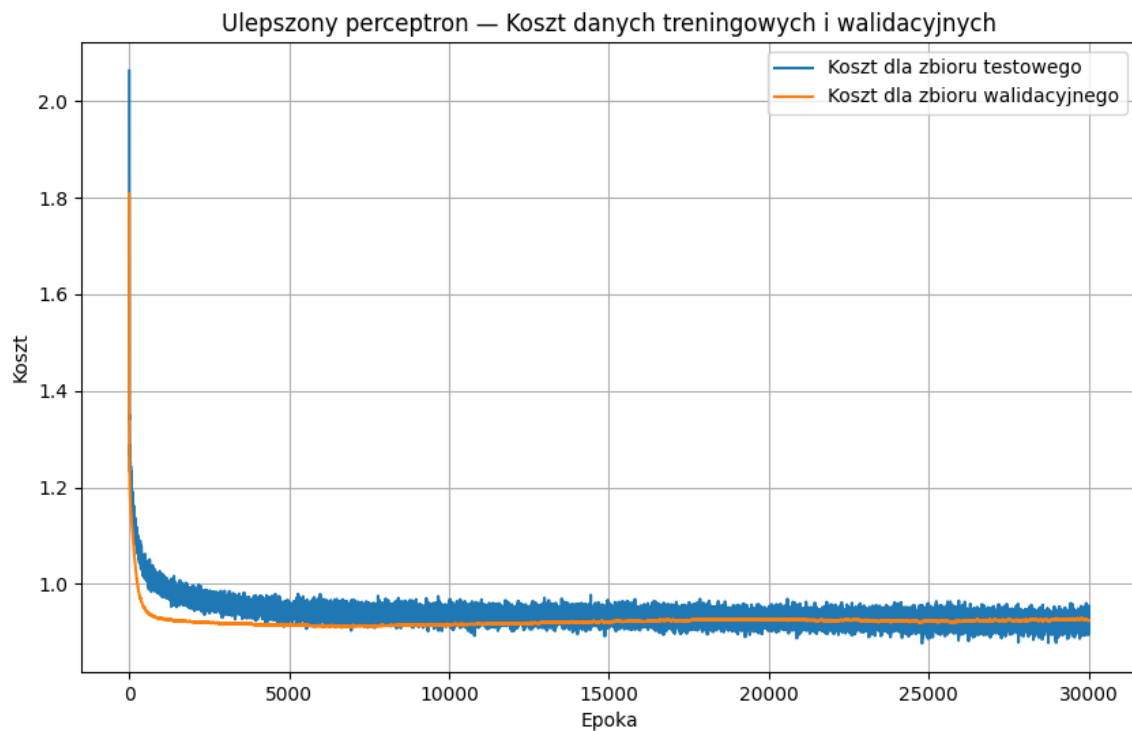
## 4. Testy

Dane zostały podzielone na 3 grupy: treningowe, walidacyjne, testowe w stosunku 60:20:20. 5 razy losowo podzielono dane na 3 zbiory i wykonano na nich uczenie perceptronu, a następnie sprawdzono dokładność na danych testowych. Osiągnięto następujące wyniki:

- Dokładność zwykłego perceptronu: 64.22%
- Dokładność perceptrony z mechanizmami dropoutu oraz regularyzacji: 66.18%

Poniżej znajdują się także wykresy kosztu dla danych walidacyjnych i treningowych w kolejnych epokach dla obu perceptronów:





## 5. Opis wyników

Wykorzystanie bardziej zaawansowanego perceptronu daje lepszą dokładność jednak model ten uczy się wolniej i potrzebuje więcej iteracji w celu uzyskania zadowalających wyników.

Na wykresach można zauważyć, że dla zwykłego perceptronu dochodzi do przeuczenia już po niecałych 1000 epokach, natomiast perceptron wykorzystujący regularyzację oraz dropout jest znacznie bardziej odporny na przeuczanie.