Zadanie nr 1 - The Delta Rule

Inteligentne przetwarzanie danych

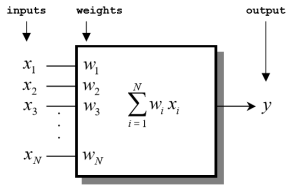
Mateusz Szczęsny, 233266 Dawid Wójcik, 233271

27.10.2019 r.

1 Cel zadania

Celem zadania jest napisanie programu, który implementuje pojedynczy neuron liniowy trenowany regułą delta z wykorzystaniem (a) pojedynczego i (b) wielu wzorców treningowych.

2 Wstęp teoretyczny



Rysunek 1: Pojedynczy neuron liniowy

Rysunek 1 przedstawia schemat graficzny pojedynczego neuronu liniowego. Składa się z następujących części:

- $x = [x_1, x_2, ..., x_n] \in \mathbf{R}^N$ - wektor wejściowy
- $w = [w_1, w_2, ..., w_n] \in \mathbf{R}^N$ - wektor wag
- $y = \sum_{i=1}^{N} w_i x_i = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_N \cdot x_N \in \mathbf{R}$ - wartość wyjściowa

Pojedynczy neuron liniowy wykonuje ważoną sumę iloczynu swoich wejść wraz z ich odpowiednimi wagami.
Reguła delta pozwala no korekcje wag pod koniec uczenia każdego zestawu treningowego. Skorygowaną wagę obliczamy na podstawie następującego równania:

$$w_i = w_i + \eta \cdot (x - y) \cdot x_i$$

3 Przebieg eksperymentu

Eksperyment polega na wykonaniu treningu pojedynczego neuronu liniowego z jednym i wieloma wzorami treningowymi zgodnie z regułą delta.

3.1 Eksperyment nr 1

3.1.1 Założenia

Uczenie neuronu liniowego pojedynczym zestawem treningowym. Porównanie wpływu kroku treningowego na prędkość oraz dokładność obliczeniową.

3.1.2 Przebieg

Wartości wejściowe neuronu dla przypadku 1) to:

- $X = [([1, 15, 3], 40)]$
- $trainingstep = 0.001$
- $K = 100$

Wartości wejściowe neuronu dla przypadku 2) to:

- $X = [([1, 15, 3], 40)]$
- $trainingstep = 0.0001$
- $K = 200$

X jest wektorem treningowym składającym się z 3 wartości oraz wartości oczekiwanej.

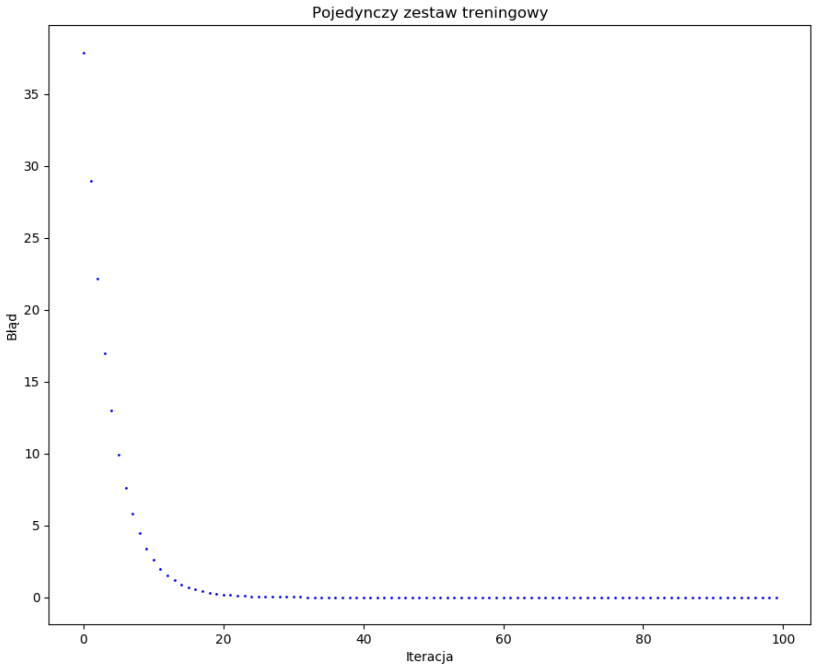
Training step określa skok, który będzie wykonywany podczas korekcji poprzednich wag.

K jest ilością iteracji, podczas których neuron, będzie dostosowywał swoje wagi w celu uzyskania wartości oczekiwanej.

3.1.3 Rezultat

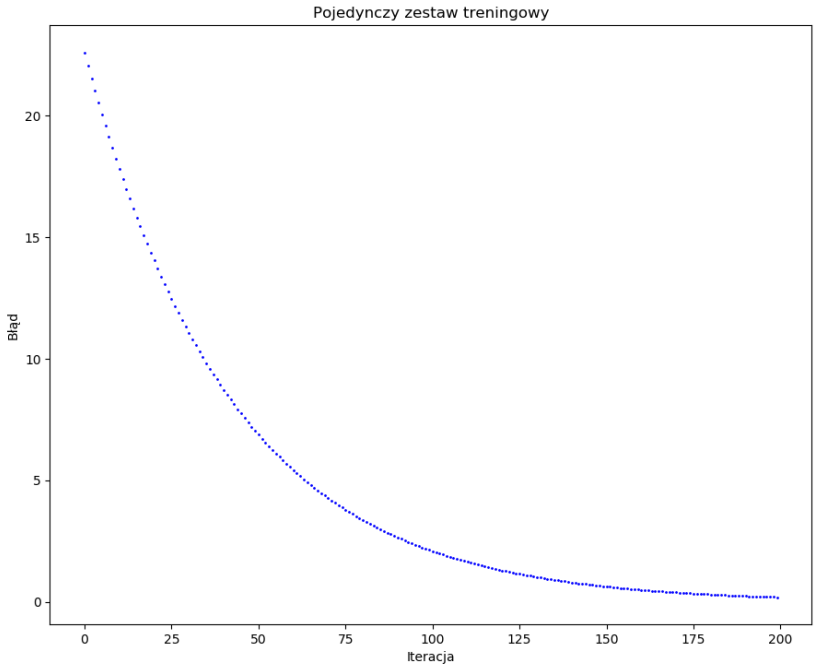
Przypadek 1)

- $Y = 39.99999999988704$
- $W = [0.6965196489675397, 2.508312698813254, 0.5595966229157469]$



Przypadek 2)

- $Y = 39.80101190368577$
- $W = [0.5226469749241156, 2.38488232125833, 1.1699354433833635]$



3.2 Eksperyment nr 2

3.2.1 Założenia

Uczenie neuronu liniowego wieloma zestawami treningowymi. Zbadanie jakości uczenia w przypadku wielu zestawów treningowych. Sprawdzenie wpływu losowania zestawu treningowego na wynik końcowy oraz zmiany błędu w iteracjach.

3.2.2 Przebieg

Wartości wejściowe neuronu dla w przypadku 1) i 2) uczenia wieloma zestawami treningowymi:

- $X = [([1, 15, 3], 40), ([5, 15, 8], 59), ([0, 3, 0], 6), ([2, 5, 0], 12)]$
- $trainingstep = 0.005$
- $K = 200$

Przypadek pierwszy został uruchomiony z losowaniem zestawów treningowych po każdej iteracji. Natomiast przypadek drugi w każdym cyklu zakładał niezmienną kolejność zestawów treningowych.

3.2.3 Rezultat

Wektor wag końcowych oraz błąd po 200 iteracjach dla przypadku z losowaniem:

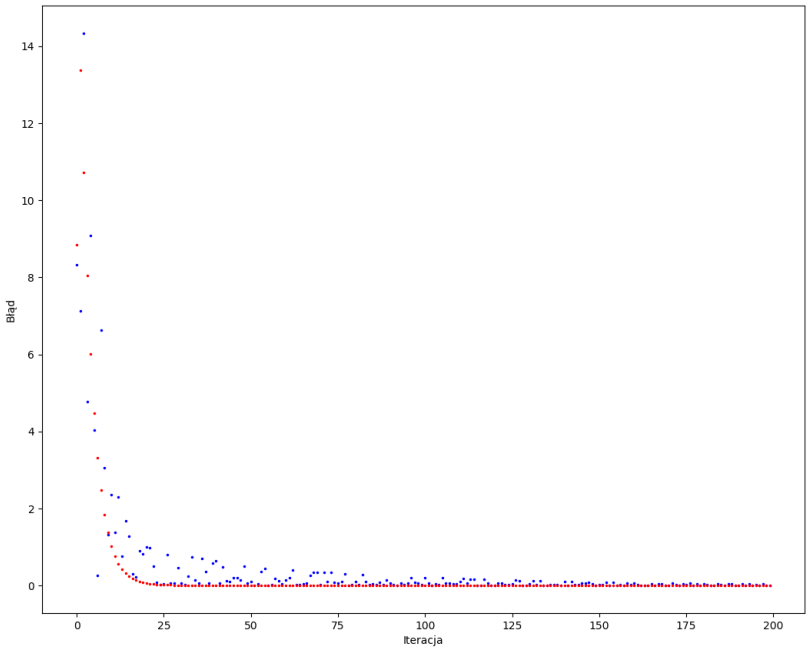
$$w = [1.0151408765753343, 2.001030093932605, 2.989198209278939]$$

$$error = 0.0021323251303613233$$

Wektor wag końcowych oraz błąd po 200 iteracjach dla przypadku bez losowania zestawów treningowych:

$$w = [1.013791503266288, 2.000164600144744, 2.9910934238344504]$$

$$error = 0.00037031375426721524$$



Rysunek 2: Wykres prezentujący zmianę błędu pomiędzy wartością wyjściową, a oczekiwaną.

Niebieskie punkty prezentują aktualną wartość błędu dla przypadku z losowaniem zestawów treningowych. Natomiast punkty czerwone odpowiadają iteracjom bez zmian kolejności zestawów.

4 Wnioski

1. Podnoszenie wartości stopnia treningowego zwiększa szybkość uczenia neuronu, jednakże dokładność obliczeń maleje ze względu na mniej precyzyjne korekty wag.

2. Należy odpowiednio dobrać ilość iteracji względem stopnia treningowego, tak aby wynik był zarówno dokładny, jak i nie prowadził do bardzo drobnych korekt wag w momencie błędu o bardzo niskim rzędzie wielkości.

3. W przypadku uczenia wieloma zestawami treningowymi możliwość tasowania może pozytywnie wpłynąć na wynik, w przypadku jeśli stosujemy metodę uczącą neuron, aż do momentu uzyskania satysfakcjonującego błędu. Można zauważyć, że błąd w takim przypadku drastycznie spadł już w okolicach 10 iteracji. Takie spadki mogą przyspieszyć proces uczenia.

5 Załączniki

```
def train(input_x, input_w, k, training_step, WITH_SHUFFLE, log):
    ERR = []
    for _ in range(k):
        x_temp = input_x
        if WITH_SHUFFLE:
            random.shuffle(x_temp)
        for training_set in x_temp:
            pattern, expected_result = training_set[0], training_set[1]
            output = 0
            for i, x in enumerate(pattern, start=0):
                output += x * input_w[i]

            for i, weight in enumerate(input_w, start=0):
                error = expected_result - output
                input_w[i] = weight + training_step * error * pattern[i]
                if log:
                    print(f"expected: {expected_result} | output: {output}")

        ERR.append(abs(error))
    return input_w, ERR

def main():
    X = [([1, 15, 3], 40), ([5, 15, 8], 59), ([0, 3, 0], 6), ([2, 5, 0], 12)]
    training_step = 0.005
    K = 200
    W = [random.uniform(0, 1) for _ in range(len(X[0][0]))]

    w1, err1 = train(X, W[:], K, training_step, True, False)
    w2, err2 = train(X, W[:], K, training_step, False, False)

    plt.plot(err1, "bo", markersize=1.5)
    plt.plot(err2, "ro", markersize=1.5)
    plt.ylabel("Error")
    plt.xlabel("Iteration")
    plt.show()


if __name__ == "__main__":
    main()
```