# Praca domowa 4
# Sieci neuronowe

Damian Jankowski s188597

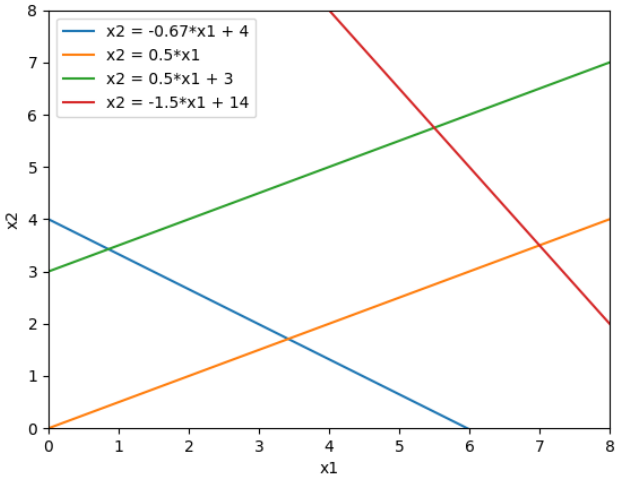
18 maja 2023

## 1 Wstęp

Celem pracy domowej było zapoznanie się z działaniem sieci neuronowych. Zadaniem było zbudowanie sieci neuronowej, która będzie rozpoznawać czy dany punkt jest wewnątrz jakiegoś obszaru czy na zewnątrz oraz porównanie działania sieci neuronowej z gotowymi rozwiązaniami.

Wybrałem trapez opisany za pomocą tych czterech równań:

$$\begin{cases} x_2 = -\frac{2}{3}x_1 + 4 \\ x_2 = \frac{1}{2}x_1 \\ x_2 = \frac{1}{2}x_1 + 3 \\ x_2 = -\frac{3}{2}x_1 + 14 \end{cases} \tag{1}$$




Rysunek 1: Wykres trapezu wykorzystywanego do tego zadania

## 2 Opis budowy sieci neuronowej

Sieć składa się z 2 warstw. Pierwsza warstwa ma 4 neurony odpowiadające czterem równaniom, które opisują trapez. Druga warstwa ma jeden neuron, który ma za zadanie zwrócić 1 jeśli punkt jest wewnątrz trapezu lub 0 jeśli punkt jest na zewnątrz.

Pojedynczy neuron można opisać równaniem:

$$\sigma(w_1x_1 + w_2x_2 + w_3) \tag{2}$$

gdzie $w_i$ to wagi, $x_i$ to wejścia, a $\sigma$ to funkcja aktywacji.

W tym przypadku jako funkcję aktywacji użyłem funkcji stepu:

$$\sigma(x) = \begin{cases} 1 & \text{dla } x > 0 \\ 0 & \text{dla } x \leq 0 \end{cases} \tag{3}$$

Dla tego zadania kolejne neurony są opisane następującymi równaniami:

1. $\frac{2}{3}x_1 + x_2 - 4 = 0$
2. $-\frac{1}{2}x_1 + x_2 = 0$
3. $\frac{1}{2}x_1 - x_2 + 3 = 0$
4. $-\frac{3}{2}x_1 - x_2 + 14 = 0$

Piąty neuron jest opisany równaniem:

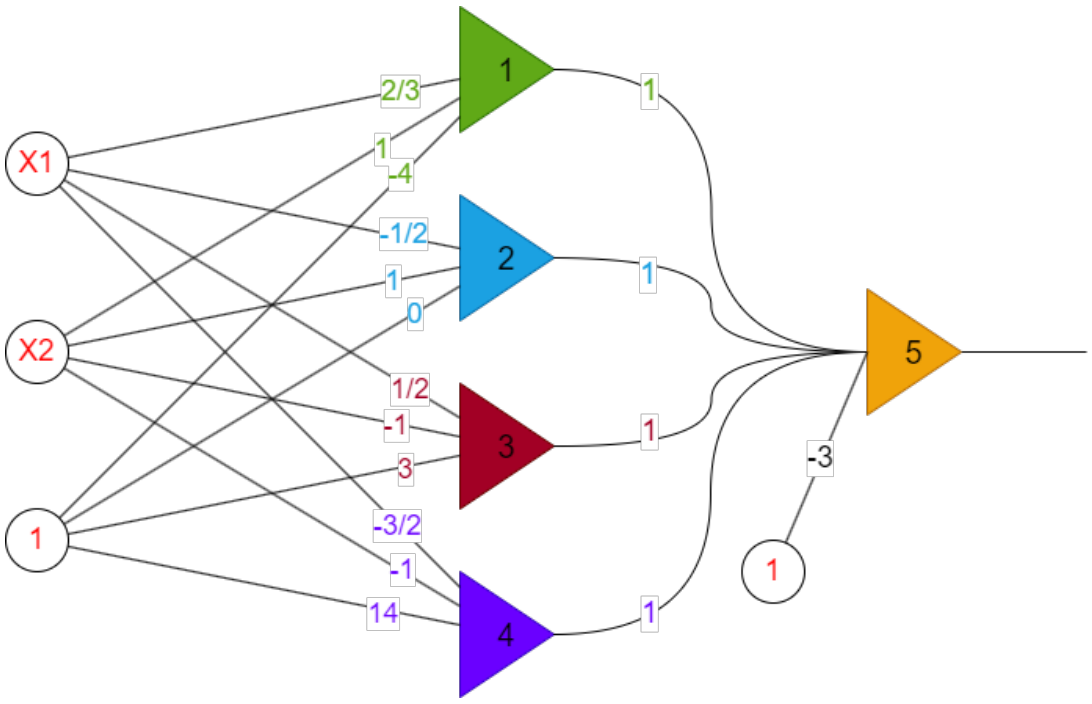
$$\sigma(w_1N_1 + w_2N_2 + w_3N_3 + w_4N_4 - w_5 \cdot 1) \tag{4}$$

gdzie $N_i$ to wyjścia z neuronów pierwszej warstwy, a $w_i$ to wagi.

Wagi zostały dobrane w taki sposób, by każdy neuron pierwszej warstwy zwracał 1, gdy punkt znajduje się po poprawnej stronie prostej. Wyglądają następująco:

$$w_1 = w_2 = w_3 = w_4 = 1 \quad w_5 = -3 \tag{5}$$

Waga biasa została ustawiona jako -3, ponieważ gdy wszystkie neurony pierwszej warstwy zwrócą 1 (punkt znajduje się wewnątrz trapezu), to piąty neuron również zwróci 1. W każdym innym przypadku zwróci 0.




Rysunek 2: Schemat sieci neuronowej

# 3 Zasada działania modelu

Zasadę działania modelu można opisać następująco dla przykładowcych punktów:

- $P_1 = (4, 3)$
- $P_2 = (7, 2)$

Dla punktu $P_1$ neurony zwracają następujące wartości:

- $N_1 = \sigma(\frac{2}{3} \cdot 4 + 3 - 4) = 1$
- $N_2 = \sigma(-\frac{1}{2} \cdot 4 + 3) = 1$
- $N_3 = \sigma(\frac{1}{2} \cdot 4 - 3 + 3) = 1$
- $N_4 = \sigma(-\frac{3}{2} \cdot 4 - 3 + 14) = 1$
- $N_5 = \sigma(1 + 1 + 1 + 1 - 3) = 1$

Według tej sieci punkt $P_1$ znajduje się wewnątrz trapezu.

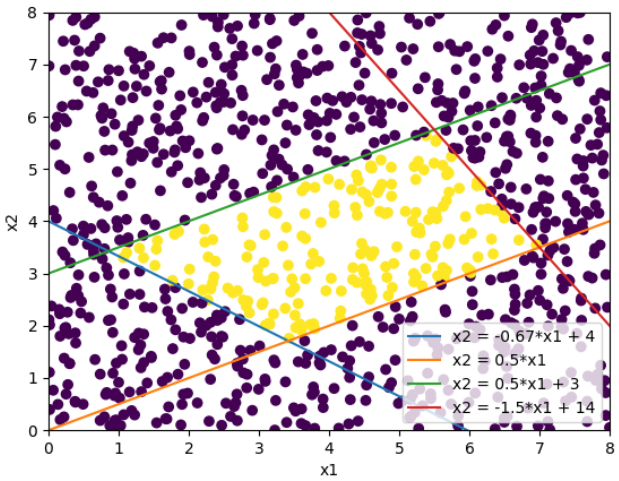
Natomiast dla punktu $P_2$:

- $N_1 = \sigma(\frac{2}{3} \cdot 7 + 2 - 4) = 1$
- $N_2 = \sigma(-\frac{1}{2} \cdot 7 + 2) = 0$
- $N_3 = \sigma(\frac{1}{2} \cdot 7 - 2 + 3) = 1$
- $N_4 = \sigma(-\frac{3}{2} \cdot 7 - 2 + 14) = 1$
- $N_5 = \sigma(1 + 0 + 1 + 1 - 3) = 0$

Według tej sieci punkt $P_2$ znajduje się na zewnątrz trapezu.

# 4 Test modelu teoretycznego

Do przetestowania modelu teoretycznego, wykorzystałem bibliotekę *keras*. Wylosowałem 1000 punktów z przedziału $[0, 8]$ i przetestowałem je na sieci, której wagi są wyznaczone przez równania prostych. Wyniki przedstawia poniższy wykres:

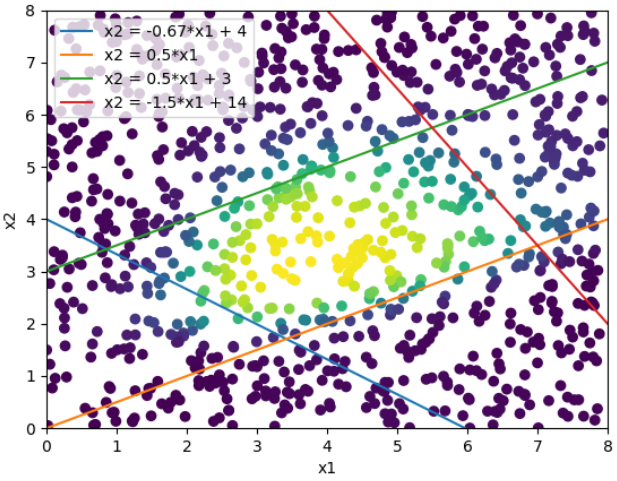



Rysunek 3: Wykres punktów z zaznaczoną predykcją dla modelu teoretycznego

# 5 Porównanie modelu teoretycznego z gotowym modelem

Do porównania modelu teoretycznego z gotowym modelem wykorzystałem ponownie bibliotekę *keras*. Ponownie również wylosowałem 1000 punktów z przedziału $[0, 8]$, które służyły do trenowania modelu.

Jako funkcję aktywacji wybrałem funkcję *sigmoid*, natomiast jako funkcję straty wybrałem *mse*. Struktura sieci jest identyczna jak w modelu teoretycznym. Ilość epok wynosi 100.




Rysunek 4: Wykres punktów testowych z zaznaczoną predykcją dla gotowego modelu

Dla warstwy pierwszej:
Model zdecydował o wybraniu następujących wag:

$$\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.6038228 & 2.0212314 & 0.9918629 & -1.7553812 \\ 0.79492724 & -2.2487426 & -3.4824367 & -0.0563702 \end{bmatrix}$$

Oraz następujących wag biasa:

$$\begin{bmatrix} w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix} = \begin{bmatrix} -6.95771 & 4.133127 & 3.4448078 & 3.7257853 \end{bmatrix}$$

Natomiast dla drugiej warstwy:
Model zdecydował o wybraniu następujących wag:

$$\begin{bmatrix} w_{51} \\ w_{52} \\ w_{53} \\ w_{54} \end{bmatrix} = \begin{bmatrix} -6.9642315 \\ 2.9435923 \\ -7.6793256 \\ -5.9134836 \end{bmatrix}$$

Oraz następujących wag biasa:

$$\begin{bmatrix} w_{55} \end{bmatrix} = \begin{bmatrix} 0.9833915 \end{bmatrix}$$

Dokładność modelu wynosła 95.70%, natomiast funkcja straty wyniosła 0.0385.

# 6 Wnioski

Jak widać na wykresach, oba modele dobrze radzą sobie z klasyfikacją, jednakże model teoretyczny jest bardziej dokładny. Wynika to z faktu, że model teoretyczny korzysta już ze znanych równań prostych, natomiast model gotowy musi nauczyć się tych równań samodzielnie.

Ważnym czynnikiem w przypadku modelu gotowego jest również wybór odpowiedniej funkcji aktywacji oraz funkcji straty, jak również ilość epok, które znacząco wpływają na dokładność modelu.

# 7 Kod programu

```
import keras.optimizers
import matplotlib.pyplot as plt
import numpy as np
from keras import Sequential, layers, backend as K


def wygeneruj_trapez():
    _x = np.linspace(0, 8, 100)
    _y1 = -0.67 * _x + 4
    _y2 = 0.5 * _x
    _y3 = 0.5 * _x + 3
    _y4 = -1.5 * _x + 14
    plt.plot(_x, _y1, label='x2 = -0.67*x1 + 4')
    plt.plot(_x, _y2, label='x2 = 0.5*x1')
    plt.plot(_x, _y3, label='x2 = 0.5*x1 + 3')
    plt.plot(_x, _y4, label='x2 = -1.5*x1 + 14')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.ylim(0, 8)
    plt.xlim(0, 8)

    plt.legend()


# Funkcja aktywacji step
def step(x):
    return K.cast(K.greater(x, 0), K.floatx())


def testuj_model_teor():
    weights1 = np.array([
        [2 / 3, -1 / 2, 1 / 2, -3 / 2],
        [1, 1, -1, -1]])

    biases1 = np.array([-4, 0, 3, 14])

    weights2 = np.array([[1, 1, 1, 1]]).T

    biases2 = np.array([-3])

    # Definicja modelu sieci neuronowej
    model = Sequential([
        layers.Dense(4, activation=step, input_shape=(
            2,), weights=[weights1, biases1]),
        layers.Dense(1, activation=step, weights=[weights2, biases2])
    ])

    # Wygenerowanie 1000 punktow z przedzialu [0, 8]
    points = np.random.uniform(low=0, high=8, size=(1000, 2))
    predictions = model.predict(points)

    # Wygenerowanie wykresu z punktami
    plt.scatter(points[:, 0], points[:, 1], c=predictions.flatten())
    wygeneruj_trapez()
    plt.savefig('punkty_teor.png')

    plt.show()


# Funkcja, ktora sprawdza, czy punkt nalezy do obszaru trapezu
```

```
def is_in_trapezoid(x1, x2):
    condition1 = (2 / 3 * x1 + x2 - 4 >= 0)
    condition2 = (x2 - 0.5 * x1 >= 0)
    condition3 = (x2 - 0.5 * x1 - 3 <= 0)
    condition4 = (1.5 * x1 + x2 - 14 <= 0)
    return condition1 and condition2 and condition3 and condition4


def testuj_model():
    # Definicja modelu sieci neuronowej
    model = Sequential([
        layers.Dense(4, activation='sigmoid', input_shape=(2,)),
        layers.Dense(1, activation='sigmoid')
    ])

    # Kompilacja modelu
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01),
                  loss='mse', metrics=['accuracy'])

    # Wygenerowanie 1000 punktow z przedzialu [0, 8]
    points = np.random.uniform(low=0, high=8, size=(1000, 2))

    # Wygenerowanie etykiet dla punktow
    labels = np.array([is_in_trapezoid(x1, x2) for x1, x2 in points])

    # Trenowanie modelu
    model.fit(points, labels, epochs=100, batch_size=32)

    # Wypisz wagi i biasy
    print(*model.layers[0].get_weights())
    print(*model.layers[1].get_weights())

    # Predykcja dla punktow testowych

    test_points = np.random.uniform(low=0, high=8, size=(1000, 2))

    predictions = model.predict(test_points)

    # Wygenerowanie wykresu z punktami
    plt.scatter(test_points[:, 0], test_points[:, 1], c=predictions.flatten())
    wygeneruj_trapez()
    plt.savefig('punkty.png')

    plt.show()


if __name__ == "__main__":
    wygeneruj_trapez()
    plt.savefig('trapez.png')
    plt.show()

    testuj_model_teor()

    testuj_model()
```