Wykonanie: Grzegorz Denert, Michał Dorosz

**Zadanie 1.**

Zaimplementować dwa zaprezentowane na wykładzie przykłady rozwiązania problemu XOR.

**Podejście I**

Obraz zawierający diagram, linia, krąg

Opis wygenerowany automatycznie

Rysunek . Rozwiązanie problemu XOR. Podejście I.

import numpy as np

def calculate\_neuron(weights, neuron\_input, threshold ):

    output = np.dot(weights, neuron\_input) + threshold

    return 1 if output >= 0 else 0

def XOR(inputs):

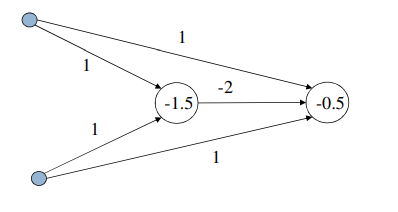
    y1 = calculate\_neuron([1,-1], inputs, -0.5)

    y2 = calculate\_neuron([-1,1], inputs, -0.5)

    result = calculate\_neuron([1,1], [y1,y2], -0.5)

    return result

**Podejście II**

****

Rysunek . Rozwiązanie problemu XOR. Podejście II.

import numpy as np

def calculate\_neuron(weights,neuron\_input, threshold ):

    output = np.dot(weights, neuron\_input) + threshold

    return 1 if output >= 0 else 0

def XOR(inputs):

    y1 = calculate\_neuron([1,1],inputs, -1.5)

    result = calculate\_neuron([1,-2,1],[inputs[0],y1,inputs[1]], -0.5)

    return result

Przeanalizować działanie poszczególnych neuronów wchodzących w skład pierwszej, jak i drugiej warstwy. Zaprezentować graficznie ich działanie.

**Podejście I**

def main():

    inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

    x1 = np.linspace(-1, 2, 100)

    x2 = x1 - 0.5

    plt.plot(x1, x2)

    plt.xlim(-0.25, 1.25)

    plt.ylim(-0.25, 1.25)

    x1 = np.linspace(-1, 2, 100)

    x2 = x1 + 0.5

    plt.xlabel('x1')

    plt.ylabel('x2')

    plt.plot(x1, x2)

    for point in inputs:

        plt.scatter(point[0], point[1], color='red')

    plt.figure()

    x3 = np.linspace(-1, 2, 100)

    x4 = -1\*x3+0.5

    plt.xlabel('y1')

    plt.ylabel('y2')

    plt.plot(x3, x4)

    plt.xlim(-0.25, 1.25)

    plt.ylim(-0.25, 1.25)

    for point in inputs:

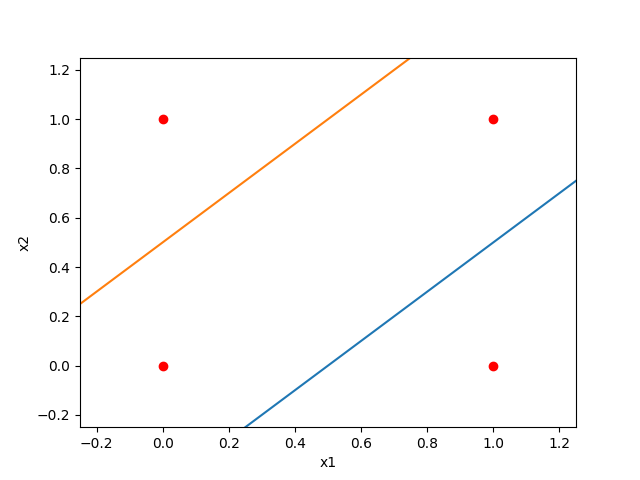
        y1 = calculate\_neuron([1, -1], point, 0.5)

        y2 = calculate\_neuron([-1, 1], point, 0.5)

        plt.scatter(y1, y2, color='green')

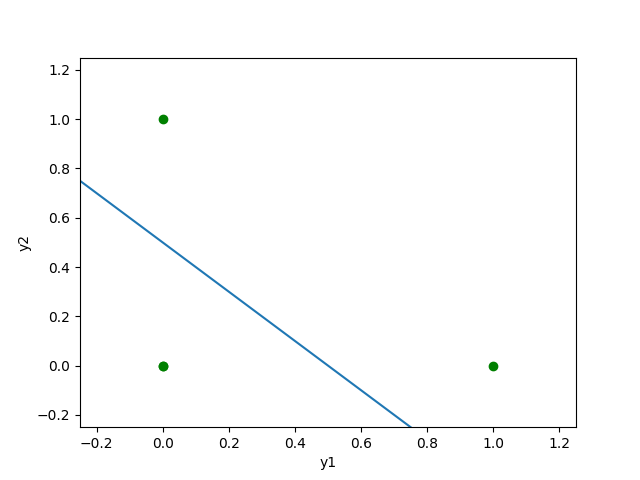
    plt.show()

main()



Rysunek 3. Działanie neuronów warstwy ukrytej. Podejście I.

Sygnał wejściowy zostaje przekazany dwóm neuronom z pierwszej warstwy. Wartości które zwracają można interpretować jako oraz . Na wykresie widzimy jedynie trzy punkty, ponieważ wartości oraz dla wartości wejściowych , równych (0, 0) oraz (1, 1) są identyczne. Aby uzyskać rozwiązanie problemu XOR wystarczy następnie podać wartości y1 oraz y2 na wejście kolejnego neuronu, gdyż problem reprezentowany na rysunku 3. jest liniowo separowalny. Ostateczne działanie sieci jest zaprezentowane na rysunku 4.



Rysunek 4. Wyjście sieci neuronowej. Podejście I.

Neuron ten rozwiązuje problem AND. Na wyjściu tego neuronu pojawia się „1” jedynie w sytuacji gdy wartości wejściowe są równe: = 1 oraz = 1.

**Podejście II**

def main():

    inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

    x1 = np.linspace(-1, 2, 100)

    x2 = -1\*x1+1.5

    plt.xlabel('x1')

    plt.ylabel('x2')

    plt.xlim(-0.25, 1.25)

    plt.ylim(-0.25, 1.25)

    plt.plot(x1, x2)

    for point in inputs:

        plt.scatter(point[0], point[1], color='red')

    plt.figure()

    ax = plt.axes(projection='3d')

    tuple\_list = [(2, -2, -2), (-2, 2, -2), (-2, -2, 2)]

    x1, x2, \_ = zip(tuple\_list)

    x1, x2 = np.meshgrid(x1, x2)

    plt.xlabel('x1')

    plt.ylabel('x2')

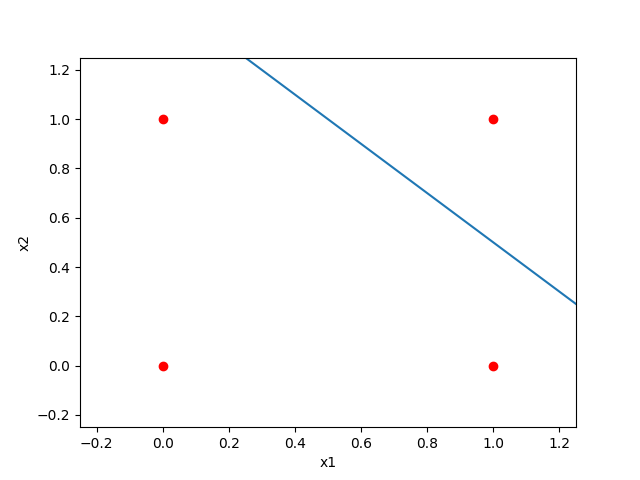
    ax.set\_zlabel('y')

    z = 0.5\*x1 + 0.5\*x2 - 0.25

    ax.plot\_surface(x1, x2, z)

    plt.show()

main()

****

Rysunek 5. Działanie neuronu warstwy ukrytej. Podejście II.

Ponieważ neuron wyjściowy przyjmuje trzy wartości, aby zaprezentować graficznie jego działanie potrzebny jest wykres trójwymiarowy (rysunek 6.). Wartość wyjściową 0 neuron osiągnie jedynie dla punktów leżących na prezentowanej płaszczyźnie.

*Obraz zawierający diagram, linia, design, kostka/sześcian

Opis wygenerowany automatycznie*

Rysunek 6. Wyjście sieci neuronowej. Podejście II.

**Zadanie 2.**

Zaproponować rozwiązanie problemu parzystości dla N=4 (wymiar danych wejściowych).

Obraz zawierający linia, diagram, Czcionka, design

Opis wygenerowany automatycznie

Rysunek 7. Przykładowe rozwiązanie problemy parzystości dla N=4

Rysunek . Propozycja rozwiązania problemu parzystości dla N=4 z wykorzystaniem trzech odpowiednio połączonych sieci neuronowych rozwiązujących problem XOR.

**Implementacja**

import numpy as np

def calculate\_neuron(weights,neuron\_input, threshold):

    output = np.dot(weights, neuron\_input) + threshold

    return 1 if output >= 0 else 0

def XOR(neuron\_inputs):

    y1 = calculate\_neuron([1, 1],neuron\_inputs, -1.5)

    result = calculate\_neuron([1, 2, 1], [neuron\_inputs[0], y1, neuron\_inputs[1]], -0.5)

    return result

def parity(inputs):

    y1 = XOR([inputs[0], inputs[1]])

    y2 = XOR([inputs[2], inputs[3]])

    output = XOR([y1, y2])

    return output

def main():

    inputs = [1, 0, 1, 0]

    y = parity(inputs)

    print(y)

main()

Przeanalizować działanie poszczególnych neuronów w zaproponowanej sieci.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x** | **x2** | **x3** | **x4** | **XOR1** | **XOR2** | **XOR3** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Tabela 1. Wartości poszczególnych segmentów przykładowej sieci neuronowej

Tabela 1. pokazuje, że zaproponowane rozwiązanie problemu parzystości N=4 jest prawidłowe. Opisana schematem z rysunku 7. sieć neuronowa zwraca na wyjściu wartość 0, gdy liczba podanych na wejściu zer i jedynek jest parzysta. W przeciwnym wypadku, na wyjściu pojawia się 1.