**Лабораторна робота №1**

**Нейронна реалізація логічних функцій AND, OR, XOR**

*Мета:* Дослідити математичну модель нейрона.

**Хід роботи**

Репозиторій GITHUB: https://github.com/AnatoliiYarmolenko/SMI\_1S4C

**Завдання:**

Реалізувати обчислювальний алгоритм для функції xor(x1, x2) через функції or(x1, x2) і and(x1, x2) в програмному середовищі (Python).

Зобразити двохслойний персептрон для функції xor(x1, x2) та скласти відповідне рівняння розділяючої прямої, використовуючи теоретичний матеріал даної лабораторної роботи.

**Виконання:**

Для початку представимо алгоритм для функції xor з поясненням:

def neuron\_OR(*x1*, *x2*):

    """

    Реалізує логіку нейрона OR.

    Ваги: w1=1, w2=1. Поріг: -0.5.

    """

    result = (1 \* x1) + (1 \* x2) - 0.5

*return* 1 *if* result > 0 *else* 0

def neuron\_AND(*x1*, *x2*):

    """

    Реалізує логіку нейрона AND.

    Ваги: w1=1, w2=1. Поріг: -1.5.

    """

    result = (1 \* x1) + (1 \* x2) - 1.5

*return* 1 *if* result > 0 *else* 0

def neuron\_XOR(*x1*, *x2*):

    """

    Реалізує логіку нейрона XOR, використовуючи нейрони OR та AND.

    """

*# Перший шар нейронів*

    y1 = neuron\_OR(x1, x2)

    y2 = neuron\_AND(x1, x2)

*# "Головний" нейрон (другий шар)*

*# Ваги: w1=1, w2=-1. Поріг: -0.5.*

*# Ця комбінація спрацьовує лише для (y1=1, y2=0)*

    result = (1 \* y1) + (-1 \* y2) - 0.5

*return* 1 *if* result > 0 *else* 0

Ось ділянка коду, яка відповідає за це завдання. Функція XOR викликає дві інші функції, OR та AND, щоб отримати кінцевий результат. На табл. 1 показаний принцип роботи цієї функції.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x1 | x2 | OR(x1, x2) | AND(x1, x2) | Результат (OR - AND) | Правильний XOR |
| 0 | 0 | 0 | 0 | 0 - 0 = 0 | 0 |
| 0 | 1 | 1 | 0 | 1 - 0 = 1 | 1 |
| 1 | 0 | 1 | 0 | 1 - 0 = 1 | 1 |
| 1 | 1 | 1 | 1 | 1 - 1 = 0 | 0 |

**Табл. 1** Значення функції

Перевіремо правильність, нанісши випадкові точки до графіку та розділимо їх відповідно до результатів функції:

*#* x\_vals = np.linspace(0, 1, 100)

ax.plot(x\_vals, 0.5 - x\_vals, 'r--', *label*="g₁(x): x1 + x2 = 0.5 (OR)")

ax.plot(x\_vals, 1.5 - x\_vals, 'b--', *label*="g₂(x): x1 + x2 = 1.5 (AND)")

*# Розфарбуємо точки за трьома зонами логіки:*

*#   зона 1: OR=0 (x1 + x2 < 0.5)*

*#   зона 2: OR=1, AND=0 (0.5 <= x1 + x2 < 1.5)*

*#   зона 3: AND=1 (x1 + x2 >= 1.5)*

s = x1 + x2

mask\_or0 = s < 0.5

mask\_or1\_only = (s >= 0.5) & (s < 1.5)

mask\_and1 = s >= 1.5

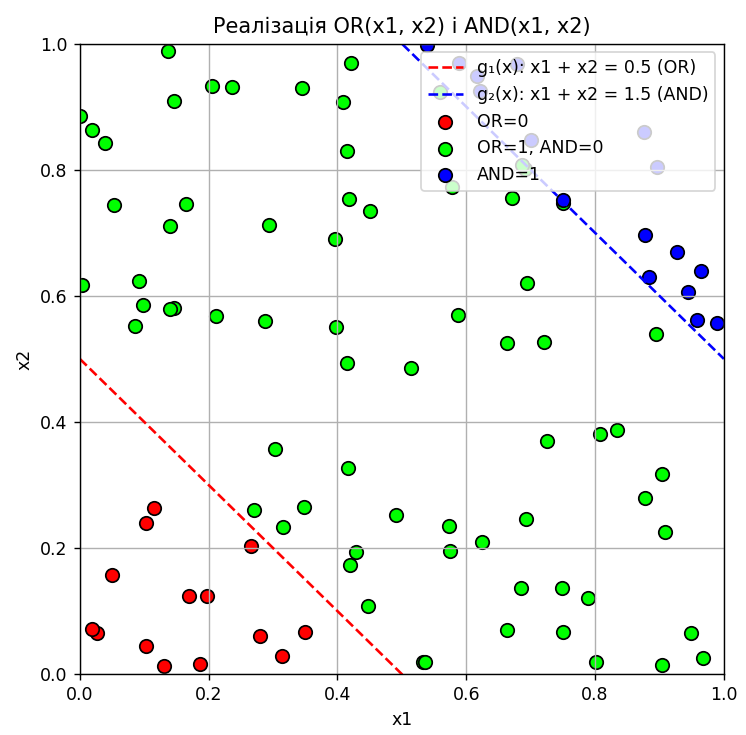
ax.scatter(x1[mask\_or0], x2[mask\_or0], *c*='red', *s*=60, *edgecolors*='black', *label*='OR=0')

ax.scatter(x1[mask\_or1\_only], x2[mask\_or1\_only], *c*='lime', *s*=60, *edgecolors*='black', *label*='OR=1, AND=0')

ax.scatter(x1[mask\_and1], x2[mask\_and1], *c*='blue', *s*=60, *edgecolors*='black', *label*='AND=1')

ax.legend(*loc*="upper right")

В результаті отримаємо графік рис 1. На ньому видно видвід двох функції та значення точок. Очевидно, що шуканий нами xor(x1, x2) лежить поміж цими двома прямими:



**Рис. 1** Спільний графік для Or та And

Наступним нашим кроком буде виведення двохшарового персептрону для функції xor(x1, x2) з рівнянням розділяючої прямої.

*# Розділяюча пряма для XOR у просторі (y1, y2): y2 = y1 - 0.5*

*# Вона відокремлює точку (1,0) від інших (0,0) та (1,1)*

ax2.plot(x\_vals, x\_vals - 0.5, 'm--', *label*="g(x): y2 = y1 - 0.5")

*# Розфарбуємо за розташуванням відносно лінії y2 = y1 - 0.5*

*# Нижче лінії (y2 < y1 - 0.5) — клас XOR=1, вище/на лінії — XOR=0*

mask\_xor1 = y2r < (y1r - 0.5)

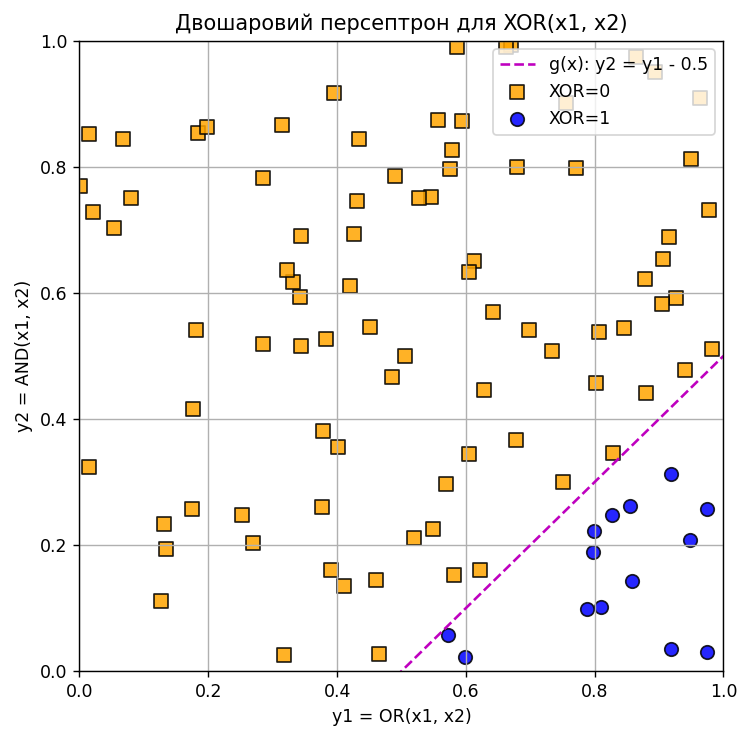
mask\_xor0 = ~mask\_xor1

ax2.scatter(y1r[mask\_xor0], y2r[mask\_xor0], *c*='orange', *s*=60, *edgecolors*='black', *alpha*=0.85, *marker*='s', *label*='XOR=0')

ax2.scatter(y1r[mask\_xor1], y2r[mask\_xor1], *c*='blue', *s*=60, *edgecolors*='black', *alpha*=0.85, *marker*='o', *label*='XOR=1')

ax2.legend(*loc*="upper right")

В результаті отримаємо наступний графік з прмою y2 = y1 - 0.5, де у2 – відповідає за And, а у1 відповідно за Or:



**Рис. 2** Двохшаровий персептрон для функції xor

**Загальний лістинг програми:**

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*# --- Нейрони ---*

def neuron\_OR(*x1*, *x2*):

    """

    Реалізує логіку нейрона OR.

    Ваги: w1=1, w2=1. Поріг: -0.5.

    """

    result = (1 \* x1) + (1 \* x2) - 0.5

*return* 1 *if* result > 0 *else* 0

def neuron\_AND(*x1*, *x2*):

    """

    Реалізує логіку нейрона AND.

    Ваги: w1=1, w2=1. Поріг: -1.5.

    """

    result = (1 \* x1) + (1 \* x2) - 1.5

*return* 1 *if* result > 0 *else* 0

def neuron\_XOR(*x1*, *x2*):

    """

    Реалізує логіку нейрона XOR, використовуючи нейрони OR та AND.

    """

*# Перший шар нейронів*

    y1 = neuron\_OR(x1, x2)

    y2 = neuron\_AND(x1, x2)

*# "Головний" нейрон (другий шар)*

*# Ваги: w1=1, w2=-1. Поріг: -0.5.*

*# Ця комбінація спрацьовує лише для (y1=1, y2=0)*

    result = (1 \* y1) + (-1 \* y2) - 0.5

*return* 1 *if* result > 0 *else* 0

*# --- Генеруємо випадкові точки ---*

np.random.seed(1)

N = 100

x1 = np.random.rand(N)

x2 = np.random.rand(N)

*# --- Обчислюємо виходи (векторизація нейронів) ---*

v\_or = np.vectorize(neuron\_OR)

v\_and = np.vectorize(neuron\_AND)

v\_xor = np.vectorize(neuron\_XOR)

y\_or = v\_or(x1, x2)

y\_and = v\_and(x1, x2)

y\_xor = v\_xor(x1, x2)

*# --- Підготовка полотна з двома графіками ---*

fig, axes = plt.subplots(1, 2, *figsize*=(12, 6))

*# ------------------------------*

*# ПЕРШИЙ ГРАФІК: OR і AND у просторі (x1, x2)*

*# ------------------------------*

ax = axes[0]

ax.set\_title("Реалізація OR(x1, x2) і AND(x1, x2)")

ax.set\_xlabel("x1")

ax.set\_ylabel("x2")

ax.set\_xlim(0, 1)

ax.set\_ylim(0, 1)

ax.grid(True)

*# Розділяючі прямі*

x\_vals = np.linspace(0, 1, 100)

ax.plot(x\_vals, 0.5 - x\_vals, 'r--', *label*="g₁(x): x1 + x2 = 0.5 (OR)")

ax.plot(x\_vals, 1.5 - x\_vals, 'b--', *label*="g₂(x): x1 + x2 = 1.5 (AND)")

*# Розфарбуємо точки за трьома зонами логіки:*

*#   зона 1: OR=0 (x1 + x2 < 0.5)*

*#   зона 2: OR=1, AND=0 (0.5 <= x1 + x2 < 1.5)*

*#   зона 3: AND=1 (x1 + x2 >= 1.5)*

s = x1 + x2

mask\_or0 = s < 0.5

mask\_or1\_only = (s >= 0.5) & (s < 1.5)

mask\_and1 = s >= 1.5

ax.scatter(x1[mask\_or0], x2[mask\_or0], *c*='red', *s*=60, *edgecolors*='black', *label*='OR=0')

ax.scatter(x1[mask\_or1\_only], x2[mask\_or1\_only], *c*='lime', *s*=60, *edgecolors*='black', *label*='OR=1, AND=0')

ax.scatter(x1[mask\_and1], x2[mask\_and1], *c*='blue', *s*=60, *edgecolors*='black', *label*='AND=1')

ax.legend(*loc*="upper right")

*# ------------------------------*

*# ДРУГИЙ ГРАФІК: простір (y1=OR, y2=AND) і лінія XOR*

*# ------------------------------*

ax2 = axes[1]

ax2.set\_title("Двошаровий персептрон для XOR(x1, x2)")

ax2.set\_xlabel("y1 = OR(x1, x2)")

ax2.set\_ylabel("y2 = AND(x1, x2)")

ax2.set\_xlim(0, 1)

ax2.set\_ylim(0, 1)

ax2.grid(True)

*# Розділяюча пряма для XOR у просторі (y1, y2): y2 = y1 - 0.5*

*# Вона відокремлює точку (1,0) від інших (0,0) та (1,1)*

ax2.plot(x\_vals, x\_vals - 0.5, 'm--', *label*="g(x): y2 = y1 - 0.5")

*# Згенеруємо окремі випадкові точки у просторі (y1, y2)*

y1r = np.random.rand(N)

y2r = np.random.rand(N)

*# Розфарбуємо за розташуванням відносно лінії y2 = y1 - 0.5*

*# Нижче лінії (y2 < y1 - 0.5) — клас XOR=1, вище/на лінії — XOR=0*

mask\_xor1 = y2r < (y1r - 0.5)

mask\_xor0 = ~mask\_xor1

ax2.scatter(y1r[mask\_xor0], y2r[mask\_xor0], *c*='orange', *s*=60, *edgecolors*='black', *alpha*=0.85, *marker*='s', *label*='XOR=0')

ax2.scatter(y1r[mask\_xor1], y2r[mask\_xor1], *c*='blue', *s*=60, *edgecolors*='black', *alpha*=0.85, *marker*='o', *label*='XOR=1')

ax2.legend(*loc*="upper right")

*# --- Показати обидва графіки ---*

plt.tight\_layout()

plt.show()

**Висновок:** В ході виконання роботи було успішно реалізовано двошаровий персептрон для вирішення нелінійної задачі XOR. Практично продемонстровано, що шляхом перетворення вхідних даних (x1, x2) за допомогою функцій першого шару (OR та AND) у новий простір ознак (y1, y2), задача стає лінійно роздільною. Це дозволило провести єдину розділяючу пряму з рівнянням y2=y1−0.5, яка коректно класифікує всі виходи, що підтверджує здатність багатошарових мереж вирішувати складні, нелінійні проблеми.