

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**З В І Т**

**Лабораторна робота №1  
з дисципліни  
«Сучасні методи та моделі  
інтелектуальних систем керування»**

Виконавець:

аспірант групи АКІТР-23-1а

Косей М.П.

Керівник:

викладач

Тиханський М. П.

2024

# Лабораторна робота №1

Тема: Моделювання простих нейронних мереж

Мета: Одержані основні навички побудови моделей штучних нейромереж

## ХІД РОБОТИ

### 1) Ознайомитись з теоретичними відомостями до лабораторної роботи

#### НЕЙРОННІ МЕРЕЖІ

Нейронні мережі є математичними моделями, які використовуються в машинному навчанні для розв'язання різноманітних завдань, таких як класифікація, регресія, синтез зображенень тощо. Вони були названі "нейронними" мережами по аналогії з біологічними нейронними системами, які є основою функціонування мозку людей та інших живих організмів.

Нейрони в біологічних нервових системах взаємодіють між собою, передаючи сигнали у вигляді електричних імпульсів, через спеціалізовані з'єднання, відомі як синапси. Analogія між біологічними нейронами і нейронами мережі полягає в тому, що нейронні мережі складаються зі штучних нейронів, або вузлів, які мають вхідні дані (подібно до дендритів біологічних нейронів), ваги, які відповідають силі з'єднань між нейронами (подібно до синапсів), функції активації, які можна розглядати як процес передачі сигналу нейрону, та вихідні дані (подібно до аксонів біологічних нейронів), які передають сигнал до наступних нейронів у мережі.

Біологічний нейрон складається з таких основних елементів (рисунок 1.1):

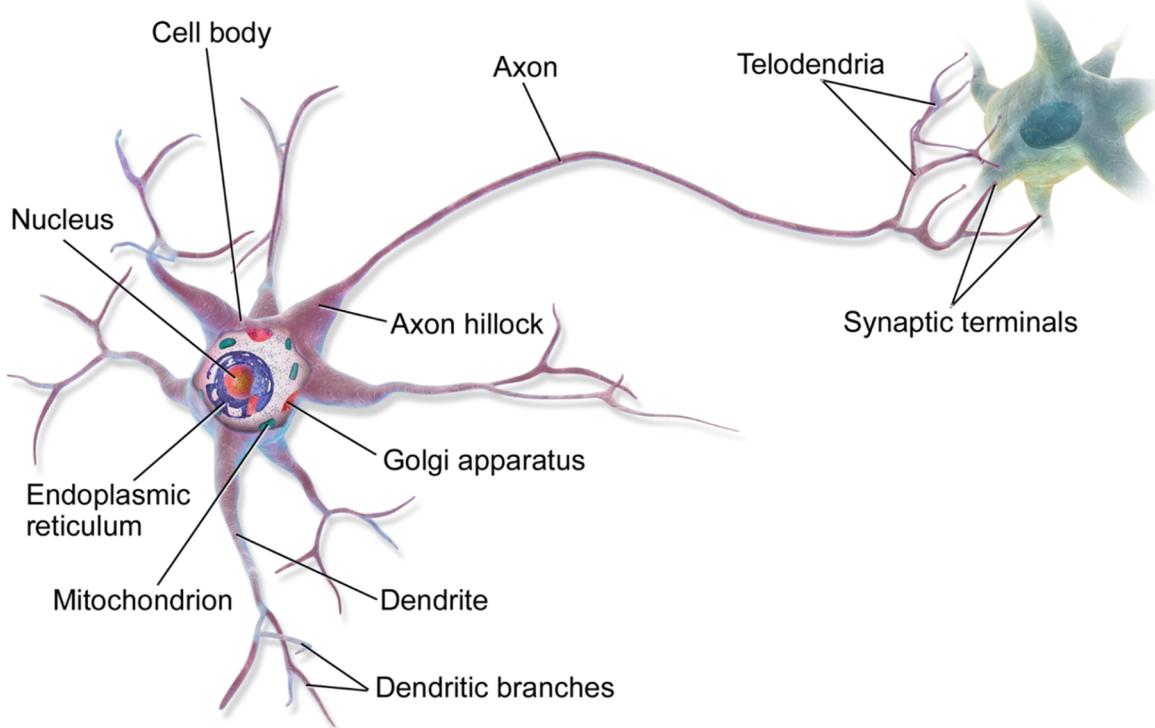


Рисунок 1.1 - Біологічний нейрон

Сома: це тіло клітини, що містить ядро, де знаходиться генетична інформація.

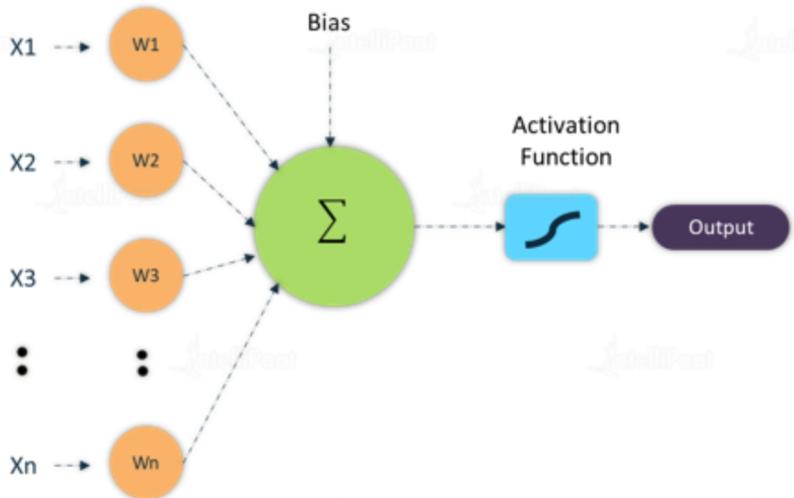
Дендрити: це відростки, які виходять з соми і служать для прийому вхідних

сигналів від інших нейронів.

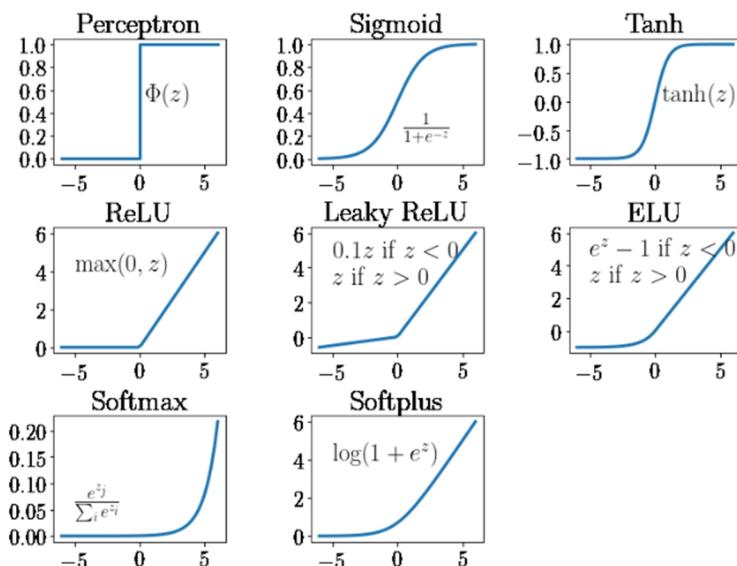
Аксон: це відрізок, який виходить з соми і передає вихідні сигнали від нейрона до інших нейронів або до ефекторних клітин (таких як м'язи або залози).

Синапси: це точки зв'язку між аксоном одного нейрона та дендритами іншого нейрона, де відбувається передача сигналів за допомогою хімічних реакцій.

Задача нейронної мережі полягає в тому, щоб вчитися визначати оптимальні ваги на основі вхідних даних, з метою виконання конкретного завдання.



**Рисунок 1.2 - Штучний нейрон**



**Рисунок 1.3 - Функції активації**

Процес роботи штучного нейрона (рисунок 1.2) включає декілька етапів: спершу він приймає вхідні дані, які можуть бути у формі чисел або векторів ( $X_1, X_2, \dots, X_n$ ). Ці дані потім множаться на ваги ( $W_1, W_2, \dots, W_n$ ), і результати сумуються.

До цієї суми додається зсув (bias), далі suma пропускається через функцію активації.

Вихідне значення функції активації, яке визначає вихідний сигнал нейрона, може потім передаватися до наступних нейронів у мережі.

Функції активації можуть бути різними (рисунок 1.3), такими як сигмоїда, ReLU (Rectified Linear Unit), тангенс гіперболічний (tanh) та інші. Вони відповідають за нелінійність штучного нейрона, що дозволяє нейронним мережам виконувати складні не лінійні завдання.

Штучні нейрони та їх ваги можуть бути оптимізовані під час процесу навчання нейронної мережі, де алгоритми оптимізації змінюють значення ваг та зсувів, щоб нейронна мережа навчалась краще вирішувати поставлені завдання.

Один з методів навчання нейромереж - зворотнє поширення (**Backpropagation**) - це один з найпоширеніших методів навчання нейронних мереж, який використовується для тренування штучних нейронних мереж з вчителем (supervised learning). Він використовується для покращення ваг (внутрішніх параметрів) мережі на основі помилок між прогнозованими виходами мережі та відповідними цільовими виходами (мітками) у навчальному наборі даних.

Основна ідея за методом зворотного поширення полягає в пошуку градієнту функції втрати (loss function) відносно ваг мережі, і використання цього градієнту для оновлення ваг таким чином, щоб зменшити помилки прогнозів мережі на навчальному наборі даних. Оновлення ваг відбувається за допомогою алгоритму градієнтного спуску (gradient descent), де ваги оновлюються у напрямку, протилежному до градієнту функції втрати, з метою мінімізації функції втрати.

## РЕАЛІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

Нейронні мережі можуть використовуватись для моделювання логічних функцій, таких як логічні вирази або булеві функції.

Моделювання логічних функцій нейронними мережами на основі таблиці істинності відноситься до задач класифікації.

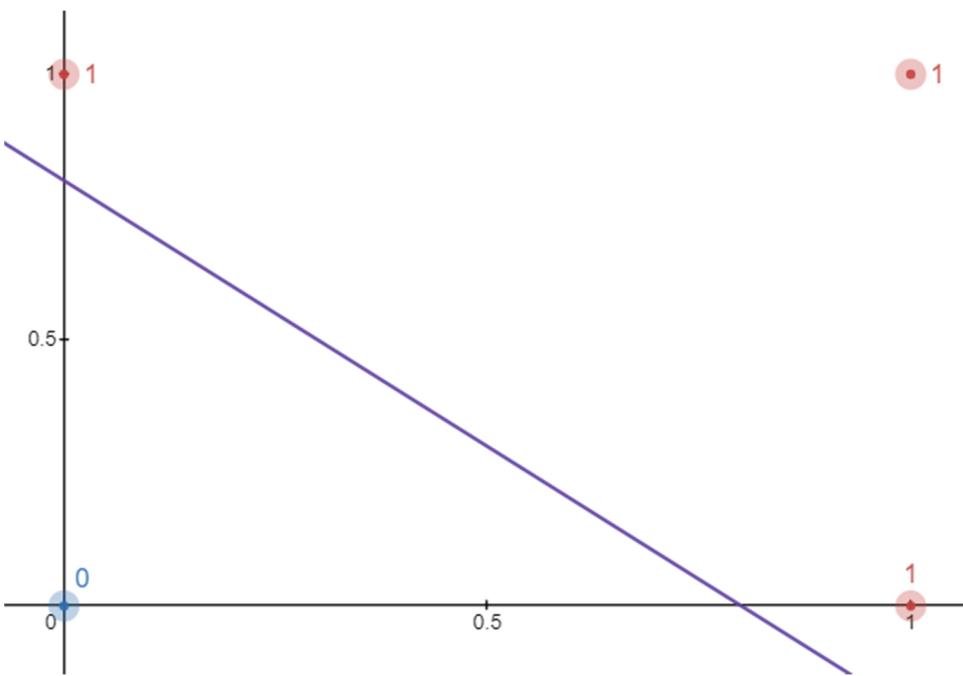
Нейронна мережа навчається класифікувати різні комбінації вхідних значень таблиці істинності на відповідні вихідні значення таблиці істинності.

Розглянемо моделювання функції “**АБО**” (**OR**  $\wedge$ ), “**І**” (**AND**  $\vee$ ), “**ВИКЛЮЧНЕ АБО**” (**XOR**  $\oplus$ ), з використання одного нейрона.

Таблиця істинності для логічної функції "АБО" (**OR**), з двома аргументами виглядає наступним чином:

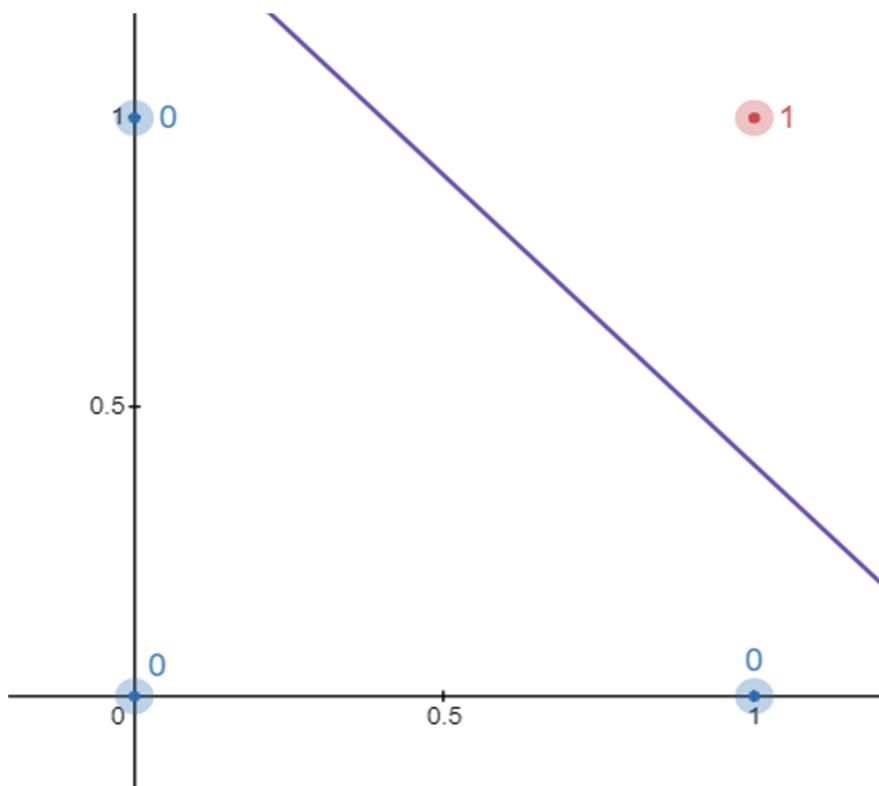
A	B	АБО (OR $\wedge$ )
0	0	0
0	1	1
1	0	1
1	1	1

Нейрон як лінійний елемент за допомогою лінійного розділення вирішує задачу класифікації вхідних на два класи: **0** та **1**.



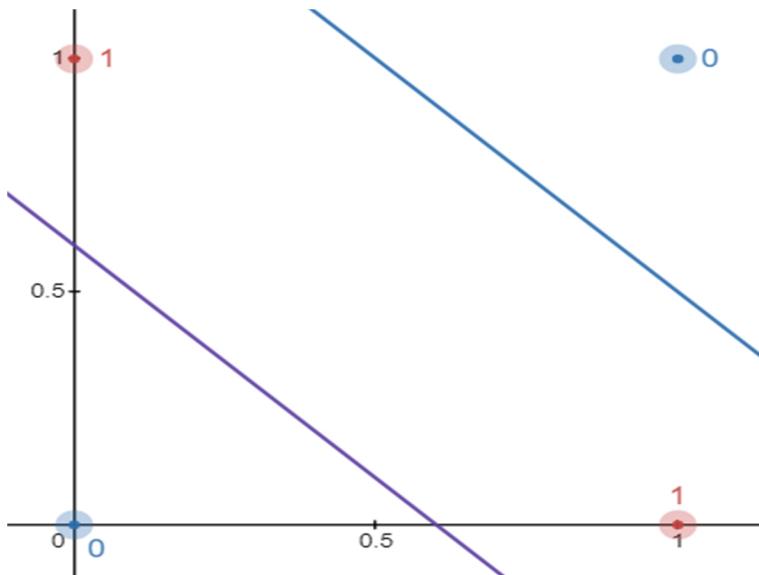
Аналогічно для функції “I” (AND):

A	B	I (AND v)
0	0	0
0	1	0
1	0	0
1	1	1

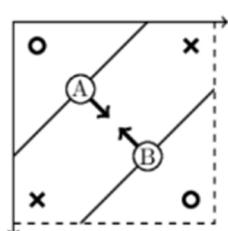


Аналогічно для функції “ВИКЛЮЧНЕ АБО” (XOR),

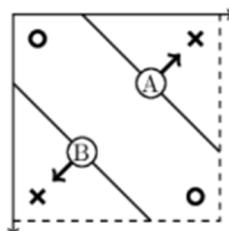
A	B	ВИКЛЮЧНЕ АБО (XOR $\oplus$ )
0	0	0
0	1	1
1	0	1
1	1	0



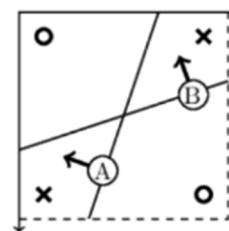
Як видно за допомогою одного нейрона неможливо реалізувати функцію “ВИКЛЮЧНЕ АБО” (XOR  $\oplus$ ), але можливо реалізувати за допомогою більшої кількості нейронів чотирма реалізаціями:



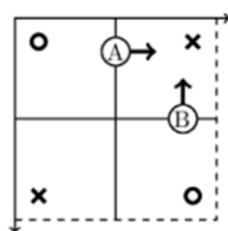
$$A \& B$$



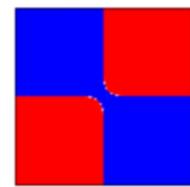
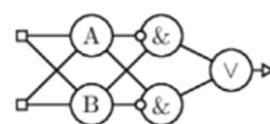
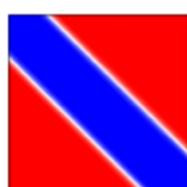
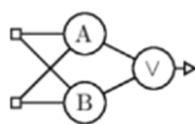
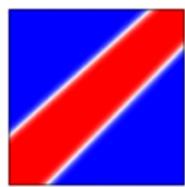
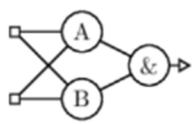
$$A \vee B$$



$$(A \& B) \vee (\bar{A} \& \bar{B})$$



$$(A \vee B) \vee (\bar{A} \vee \bar{B})$$



Кожну логічну функцію можна привести до канонічного вигляду, використовуючи логічні закони та алгоритми спрощення виразів.

Канонічний вигляд логічної функції - це такий вираз, в якому використовуються лише базові операції кон'юнкції, диз'юнкції та заперечення, і вираз містить мінімальну кількість операторів та операцій.

Враховуючи раніше згадане, можна зробити висновок, що нейронна мережа здатна змоделювати будь-яку логічну функцію, якщо вона має достатню кількість прихованих шарів та нейронів у цих шарах, при чому один нейрон може реалізовувати одну з функцій “АБО” (OR  $\Lambda$ ) , “І” (AND  $\vee$ ) .

## АПРОКСИМАЦІЯ НЕПЕРЕРВНИХ ФУНКЦІЙ БАГАТЬОХ ЗМІННИХ

Також нейроні мережі можуть апроксимувати функції.

Згідно з універсальною теоремою апроксимації доведеної Джоржем Цибенко у 1989 році, яка стверджує що штучна нейронна мережа прямого зв'язку (англ. feed-forward; у яких зв'язки не утворюють циклів) з одним прихованим шаром і сигмоїдною функцією активацією може апроксимувати будь-яку неперервну функцію багатьох змінних з будь-якою точністю.

Умовами є достатня кількість нейронів прихованого шару, вдалий підбір вагових коефіцієнтів зв'язків нейронів.

### 2) Практична частина – моделювання нейромереж.

Для моделювання нейронних мереж також можна використовувати наступні інструменти:

1) **MATLAB** – популярне середовище для математичних обчислень, яке має вбудовані інструменти для побудови та навчання нейронних мереж. MATLAB зручний для наукових досліджень, а його модулі для нейронних мереж полегшують процес моделювання без необхідності глибокого занурення в програмування.

2) **Python** – універсальна мова програмування з багатим набором бібліотек для роботи з нейронними мережами (наприклад, TensorFlow, PyTorch, Keras). Python є найпоширенішою мовою для машинного навчання завдяки простоті та багатофункціональноті.

3) **Kaggle** – платформа для змагань з машинного навчання, де можна знайти готові набори даних, різноманітні приклади використання нейронних мереж, а також навчальні матеріали. Kaggle також має вбудоване середовище для запуску Jupyter Notebook, що спрощує експерименти з моделями.

4) **Jupyter Lab** – інтегроване середовище для обчислень, яке дозволяє створювати та виконувати Python-код у вигляді нотаток. Ідеально підходить для моделювання нейронних мереж, тестування моделей, аналізу результатів та візуалізації.

Використовуємо мову програмування **Python** та проект [\*\*JupyterLab\*\*](#).



## РЕАЛІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

Lab-1.ipynb

Markdown Python 3 (ipykernel)

### РЕАЛІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

```
[8]: # Імпортуємо необхідні бібліотеки
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense
import numpy as np
import matplotlib.pyplot as plt
```

Таблиця істинності

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

```
# Визначення навчальних даних для логічної функції AND
# Вхідні дані (X) та вихідні дані (y)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])
y = np.array([[0],
              [0],
              [0],
              [1]])
```

Для реалізації логічної функції "I" достатньо одного нейрона

```
# Створення моделі нейронної мережі з одного нейрона
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(1, activation='hard_sigmoid', input_dim=2))

# Компільування моделі
from tensorflow.keras.optimizers import Adam
adam_optimizer = Adam(learning_rate=0.02)
model.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import Callback

# Створення кастомного колбеку для зупинки при досягненні точності 1
class StopWhenAccuracyIsOne(Callback):
    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs.get('accuracy')
        if accuracy == 1.0:
            print(f'\nДосягнуто точності 1.0 на епосі {epoch+1}. Зупинка навчання.')
            self.model.stop_training = True

# Навчання моделі з кастомним колбеком
model.fit(X, y, epochs=1000, callbacks=[StopWhenAccuracyIsOne()], verbose=1)
```

```
Epoch 28/1000
1/1 0s 76ms/step - accuracy: 0.7500 - loss: 0.6153
Epoch 29/1000
1/1 0s 75ms/step - accuracy: 0.7500 - loss: 0.6119
Epoch 30/1000
1/1 0s 81ms/step - accuracy: 0.7500 - loss: 0.6087
Epoch 31/1000
1/1 0s 87ms/step - accuracy: 0.7500 - loss: 0.6056
Epoch 32/1000
1/1 0s 87ms/step - accuracy: 0.7500 - loss: 0.6026
Epoch 33/1000
1/1 0s 82ms/step - accuracy: 1.0000 - loss: 0.5998
Досягнуто точності 1.0 на епосі 33. Зупинка навчання.
1/1 0s 90ms/step - accuracy: 1.0000 - loss: 0.5998
```

```
<keras.src.callbacks.history.History at 0x1ba4e6dc1c0>
```

Використовуємо функцію активації

The screenshot shows the TensorFlow API documentation for the `tf.keras.activations.hard_sigmoid` function. The page is titled "TensorFlow > API > TensorFlow v2.16.1 > Python". It includes a "View source on GitHub" button and a "Was this helpful?" feedback link. The function description states it's a Hard sigmoid activation function. Below the description is a code snippet showing the function call: `tf.keras.activations.hard_sigmoid(x)`. A note below the code explains the function's definition: "The hard sigmoid activation is defined as:" followed by a list of three items: "0 if  $x \leq -3$ ", "1 if  $x \geq 3$ ", and " $(x/6) + 0.5$  if  $-3 < x < 3$ ". A note at the bottom states it's a faster, piecewise linear approximation of the sigmoid activation. The "Args" section shows the parameter `x` as "Input tensor".

## Реалізація логічної функції "АБО" (OR $\wedge$ ) ¶

Таблиця істинності

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

```
# Визначення навчальних даних для логічної функції OR
# Вхідні дані (X) та вихідні дані (y)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])
y = np.array([[0],
              [1],
              [1],
              [1]])

# Створення моделі нейронної мережі з одного нейрона
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(1, activation='hard_sigmoid', input_dim=2))

# Компільування моделі
from tensorflow.keras.optimizers import Adam
adam_optimizer = Adam(learning_rate=0.02)
model.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import Callback

# Створення кастомного колбеку для зупинки при досягненні точності 1
class StopWhenAccuracyIsOne(Callback):
    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs.get('accuracy')
        if accuracy == 1.0:
            print(f"\nДосягнуто точності 1.0 на епосі {epoch+1}. Зупинка навчання.")
            self.model.stop_training = True

# Навчання моделі з кастомним колбеком
model.fit(X, y, epochs=1000, callbacks=[StopWhenAccuracyIsOne()], verbose=1)
```

```

Epoch 310/1000
1/1 0s 55ms/step - accuracy: 0.7500 - loss: 0.2078
Epoch 311/1000
1/1 0s 50ms/step - accuracy: 0.7500 - loss: 0.2071
Epoch 312/1000
1/1 0s 44ms/step - accuracy: 0.7500 - loss: 0.2064
Epoch 313/1000
1/1 0s 40ms/step - accuracy: 0.7500 - loss: 0.2058
Epoch 314/1000
1/1 0s 37ms/step - accuracy: 1.0000 - loss: 0.2051
Досягнуто точності 1.0 на епосі 314. Зупинка навчання.
1/1 0s 42ms/step - accuracy: 1.0000 - loss: 0.2051
<keras.src.callbacks.history.History at 0x1ba4fbdb8970>

```

```

# Тестування моделі
print("Тестування моделі:")
predictions = model.predict(X)
for i, pred in enumerate(predictions):
    print(f"Вхід: {X[i]} => Прогноз: {np.round(pred[0])}, Фактичний: {y[i][0]}")

```

Тестування моделі:

```

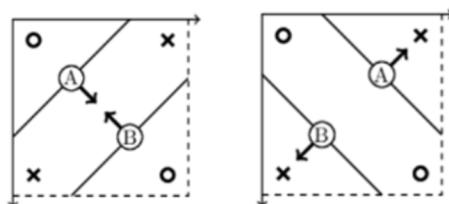
1/1 0s 127ms/step
Вхід: [0 0] => Прогноз: 0.0, Фактичний: 0
Вхід: [0 1] => Прогноз: 1.0, Фактичний: 1
Вхід: [1 0] => Прогноз: 1.0, Фактичний: 1
Вхід: [1 1] => Прогноз: 1.0, Фактичний: 1

```

### Реалізація логічної функції "ВИКЛЮЧНЕ АБО" (XOR $\oplus$ )

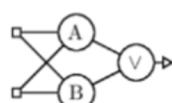
Таблиця істинності		
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Для реалізації логічної функції "ВИКЛЮЧНЕ АБО" недостатньо одного нейрона потрібно мінімум 3 нейрона

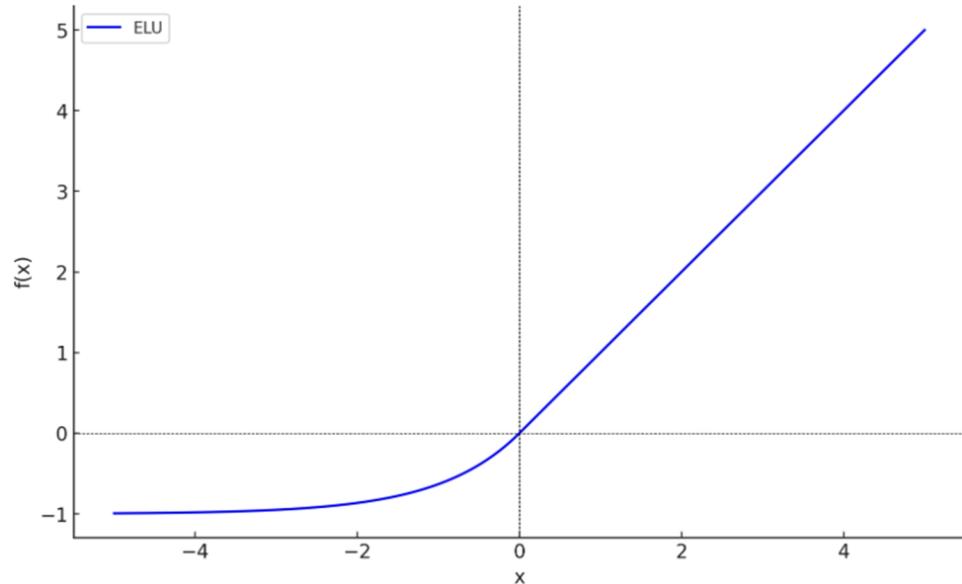


$A \& B$

$A \vee B$



Для першого шару з 2-х нейронів використовуємо функцію активації ELU



```
# Визначення навчальних даних для логічної функції XOR
# Вхідні дані (X) та вихідні дані (y)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])
y = np.array([[0],
              [1],
              [1],
              [0]])

# Створення моделі нейронної мережі з 3-х нейронів
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

model = Sequential()
model.add(Dense(2, activation='elu', input_dim=2))
model.add(Dense(1, activation='hard_sigmoid', input_dim=2))

# Компільування моделі
from tensorflow.keras.optimizers import Adam
optimizer = Adam(learning_rate=0.025)

model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import Callback

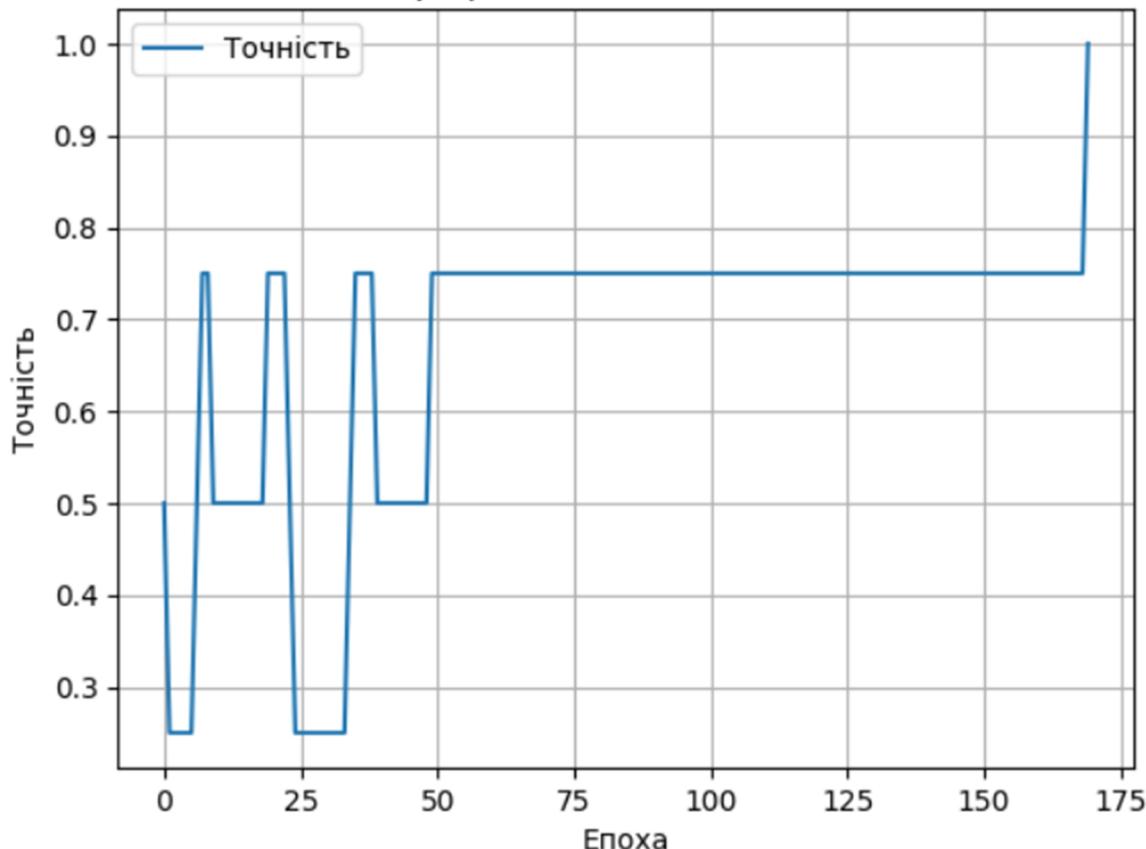
# Створення кастомного колбеку для зупинки при досягненні точності 1
class StopWhenAccuracyIsOne(Callback):
    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs.get('accuracy')
        if accuracy == 1.0:
            print(f"\nДосягнуто точності 1.0 на епосі {epoch+1}. Зупинка навчання.")
            self.model.stop_training = True

# Навчання моделі з кастомним колбеком
history = model.fit(X, y, epochs=1000, callbacks=[StopWhenAccuracyIsOne()], verbose=1)

# Графік точності
plt.plot(history.history['accuracy'], label='Точність')
plt.title('Графік точності навчання')
plt.xlabel('Епоха')
plt.ylabel('Точність')
plt.legend()
plt.grid()
plt.show()
```

```
Epoch 169/1000
1/1 - 0s - accuracy: 0.7500 - loss: 0.1977
Epoch 170/1000
1/1 - 0s - accuracy: 1.0000 - loss: 0.1835
Досягнуто точності 1.0 на епосі 170. Зупинка навчання.
1/1 - 0s - accuracy: 1.0000 - loss: 0.1835
```

Графік точності навчання



```
# Виведення архітектури моделі
model.summary()
```

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 2)	6
dense_6 (Dense)	(None, 1)	3

Total params: 29 (120.00 B)

Trainable params: 9 (36.00 B)

Non-trainable params: 0 (0.00 B)

Optimizer params: 20 (84.00 B)

```

# Тестування моделі
print("Тестування моделі:")
predictions = model.predict(X)
for i, pred in enumerate(predictions):
    print(f"Вхід: {X[i]} => Прогноз: {np.round(pred[0])}, Фактичний: {y[i][0]}")

```

Тестування моделі:

```

1/1 _____ 0s 129ms/step
Вхід: [0 0] => Прогноз: 0.0, Фактичний: 0
Вхід: [0 1] => Прогноз: 1.0, Фактичний: 1
Вхід: [1 0] => Прогноз: 1.0, Фактичний: 1
Вхід: [1 1] => Прогноз: 0.0, Фактичний: 0

```

## АПРОКСИМАЦІЯ НЕПЕРЕРВНИХ ФУНКІЙ БАГАТЬОХ ЗМІННИХ

```

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Генерація даних
def generate_data(num_samples):
    x1 = np.random.uniform(-10, 10, num_samples)
    x2 = np.random.uniform(-10, 10, num_samples)
    y = x1**2 + x2
    return np.column_stack((x1, x2)), y

# Параметри
num_samples = 1000
x_train, y_train = generate_data(num_samples)

# Створення моделі
model = keras.Sequential([
    layers.Input(shape=(2,)), # Вхідний шар з 2 входами
    layers.Dense(4, activation='sigmoid'), # Прихований шар 1
    layers.Dense(4, activation='sigmoid'), # Прихований шар 2
    layers.Dense(1) # Вихідний шар
])

# Компіляція моделі
model.compile(optimizer='adam', loss='mean_squared_error')

# Навчання моделі
history = model.fit(x_train, y_train, epochs=200, verbose=1)

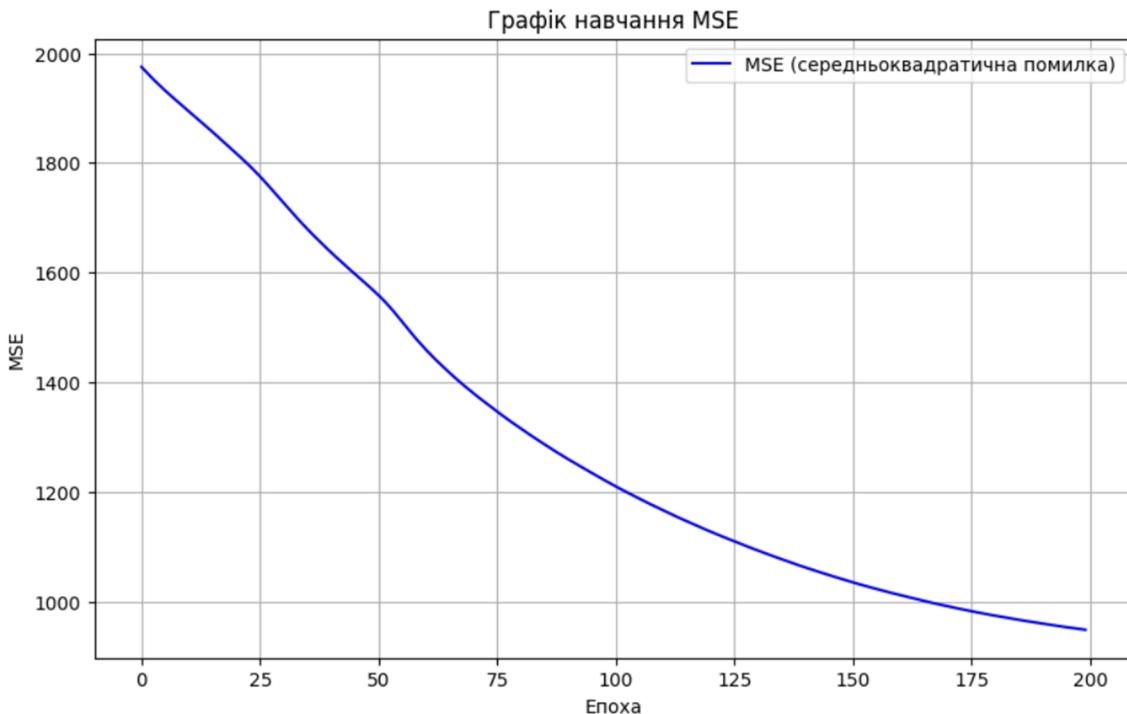
# Тестування моделі
x_test, y_test = generate_data(100)
y_pred = model.predict(x_test)

```

```

# Графік MSE
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='MSE (середньоквадратична помилка)', color='blue')
plt.title('Графік навчання MSE')
plt.xlabel('Епоха')
plt.ylabel('MSE')
plt.legend()
plt.grid()
plt.show()

```



```

# Створення сітки значень для x1 та x2
x1_grid = np.linspace(-10, 10, 100)
x2_grid = np.linspace(-10, 10, 100)
x1_mesh, x2_mesh = np.meshgrid(x1_grid, x2_grid)

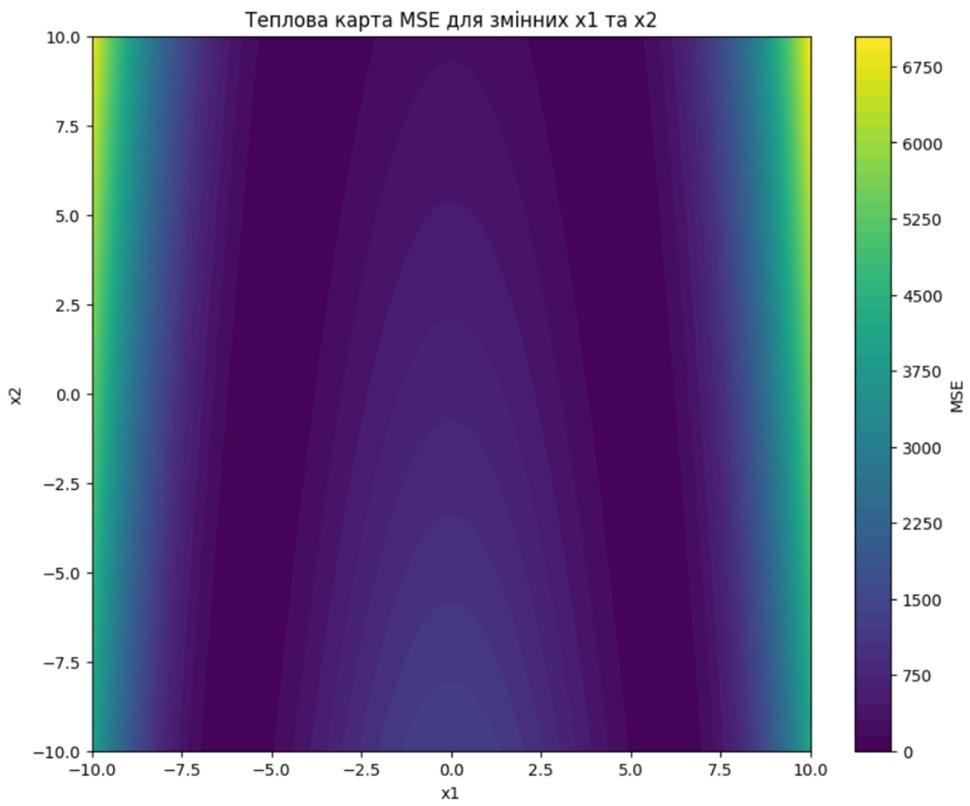
# Обчислення прогнозів моделі для сітки точок
x_test_grid = np.column_stack((x1_mesh.flatten(), x2_mesh.flatten()))
y_true_grid = x_test_grid[:, 0]**2 + x_test_grid[:, 1]
y_pred_grid = model.predict(x_test_grid).flatten()

# Обчислення MSE для кожної точки сітки
mse_grid = (y_true_grid - y_pred_grid) ** 2
mse_grid = mse_grid.reshape(x1_mesh.shape)

# Побудова теплової карти
plt.figure(figsize=(10, 8))
plt.contourf(x1_mesh, x2_mesh, mse_grid, cmap='viridis', levels=50)
plt.colorbar(label='MSE')
plt.title('Теплова карта MSE для змінних x1 та x2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()

```

313/313 ━━━━━━ 1s 2ms/step



## КЛАСТЕРИЗАЦІЯ ДАННИХ

```

import numpy as np
import matplotlib.pyplot as plt

# Задаємо перший кластер (центр у точці (2, 0))
x1 = 2 + np.random.rand(50)
y1 = 0 + np.random.rand(50)

# Задаємо другий кластер (центр у точці (-2, 0))
x2 = -2 + np.random.rand(50)
y2 = 0 + np.random.rand(50)

# Задаємо третій кластер (центр у точці (0, 2))
x3 = 0 + np.random.rand(50)
y3 = 2 + np.random.rand(50)

# Задаємо четвертий кластер (центр у точці (0, -2))
x4 = 0 + np.random.rand(50)
y4 = -2 + np.random.rand(50)

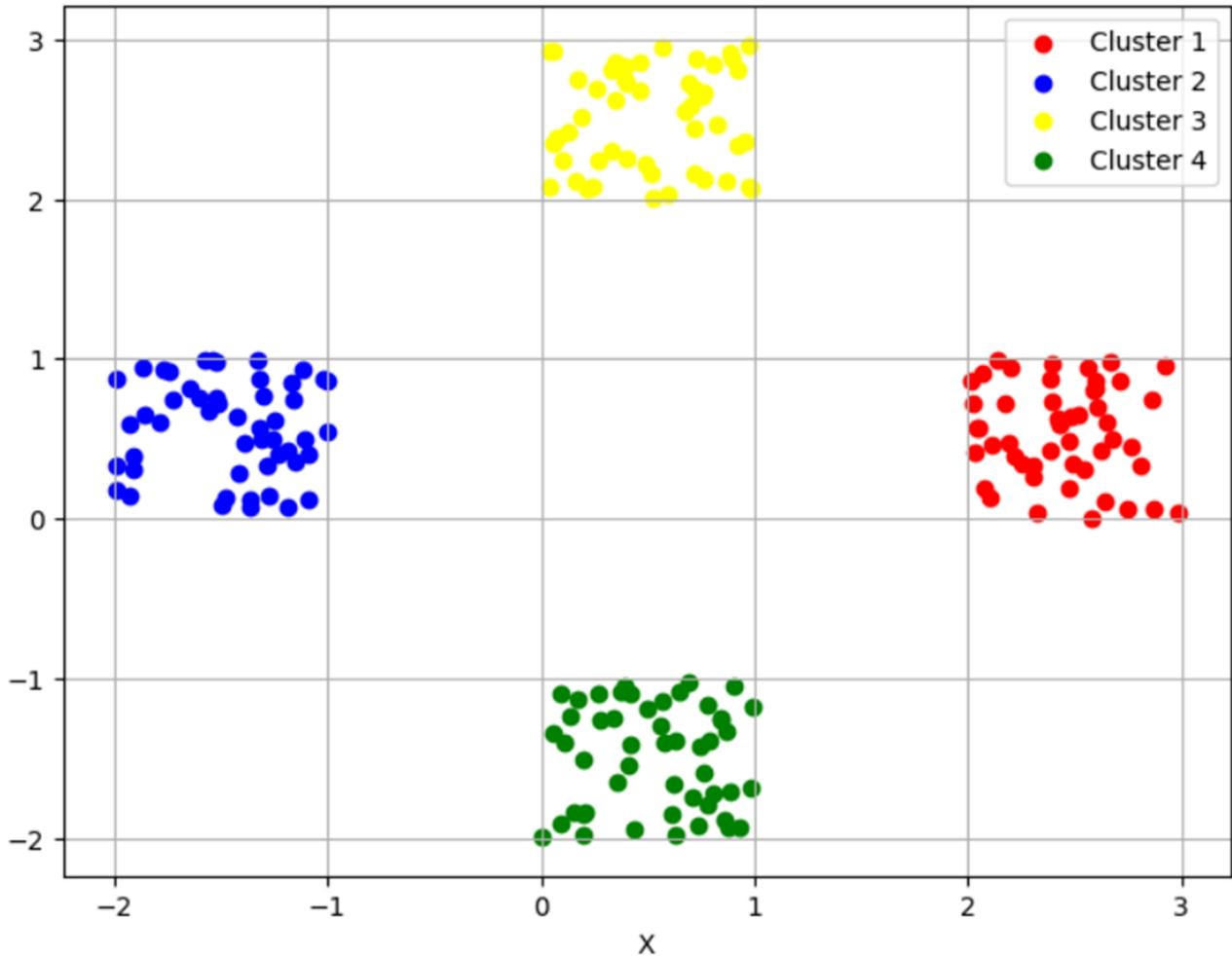
# Побудова графіку
plt.figure(figsize=(8, 6))
plt.scatter(x1, y1, color='red', label='Cluster 1')
plt.scatter(x2, y2, color='blue', label='Cluster 2')
plt.scatter(x3, y3, color='yellow', label='Cluster 3')
plt.scatter(x4, y4, color='green', label='Cluster 4')

# Налаштування графіку
plt.grid(True)
plt.legend()
plt.title('Кластеризація точок')
plt.xlabel('X')
plt.ylabel('Y')

# Відображення графіку
plt.show()

```

## Кластеризація точок



```
# Конкатенуємо всі дані в один масив
X_train = np.vstack((np.column_stack((x1, y1)),
                     np.column_stack((x2, y2)),
                     np.column_stack((x3, y3)),
                     np.column_stack((x4, y4)),

# Генеруємо мітки для кожного кластера
y_train = np.array([0]*50 + [1]*50 + [2]*50 + [3]*50)

# Перетворюємо мітки в one-hot формат
y_train = keras.utils.to_categorical(y_train, 4)

# Створення моделі нейронної мережі
model = keras.Sequential([
    layers.Dense(10, activation='sigmoid', input_shape=(2,)), # Перший шар: 10 нейронів
    layers.Dense(2, activation='sigmoid'),                      # Другий шар: 2 нейрони
    layers.Dense(4, activation='softmax')                      # Вихідний шар: 4 класи (softmax)
])

# Компіляція моделі
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Створення кастомного колбеку для зупинки при досягненні точності 1
class StopWhenAccuracyIsOne(Callback):
    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs.get('accuracy')
        if accuracy == 1.0:
            print(f"\nДосягнуто точності 1.0 на епосі {epoch+1}. Зупинка навчання.")
            self.model.stop_training = True

# Навчання моделі з використанням EarlyStopping
model.fit(X_train, y_train, epochs=1000, callbacks=[StopWhenAccuracyIsOne()])
```

```
Epoch 120/1000
4/7 ----- 0s 3ms/step - accuracy: 1.0000 - loss: 0.96192
Досягнуто точності 1.0 на епосі 120. Зупинка навчання.
7/7 ----- 0s 4ms/step - accuracy: 1.0000 - loss: 0.9572
<keras.src.callbacks.history.History at 0x2cf91a0280>
```

```
# Генерація тестових даних (аналогічно до тренувальних даних)
x1_test = 2 + np.random.rand(50)
y1_test = 0 + np.random.rand(50)

x2_test = -2 + np.random.rand(50)
y2_test = 0 + np.random.rand(50)

x3_test = 0 + np.random.rand(50)
y3_test = 2 + np.random.rand(50)

x4_test = 0 + np.random.rand(50)
y4_test = -2 + np.random.rand(50)

# Об'єднуємо тестові дані в один масив
X_test = np.vstack((np.column_stack((x1_test, y1_test)),
                     np.column_stack((x2_test, y2_test)),
                     np.column_stack((x3_test, y3_test)),
                     np.column_stack((x4_test, y4_test)))))

# Виконуємо передбачення на тестових даних
y_pred_test = model.predict(X_test)

# Перетворюємо ймовірності softmax у мітки класів
y_pred_classes = np.argmax(y_pred_test, axis=1)

# Візуалізація результатів класифікації на тестових даних
plt.figure(figsize=(8, 6))

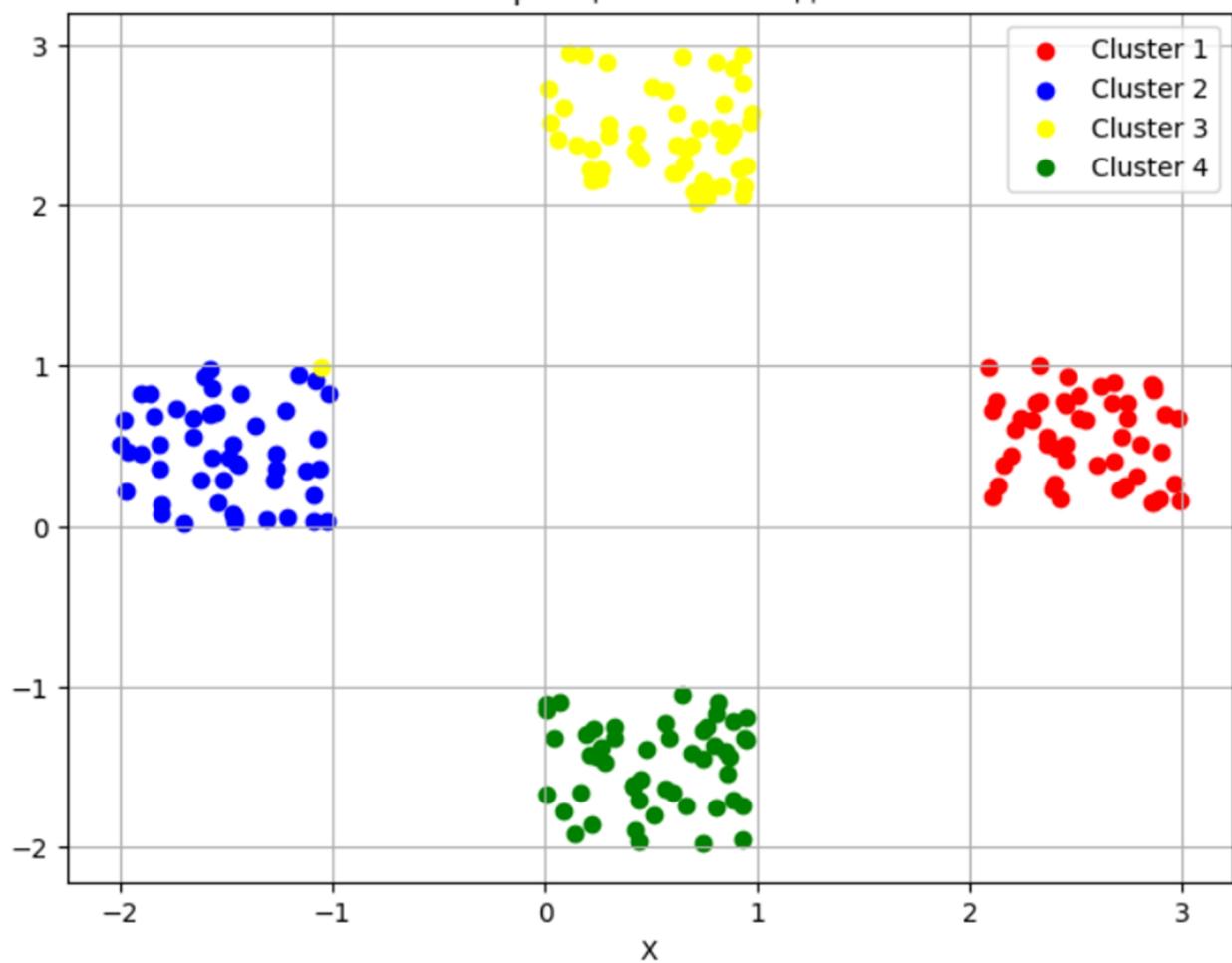
# Масиви для різних кольорів передбачених класів
colors = ['red', 'blue', 'yellow', 'green']
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']

# Розбиваємо передбачені точки по класах і будуємо графік
for i in range(4):
    cluster_points = X_test[y_pred_classes == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], color=colors[i], label=labels[i])

# Налаштування графіку
plt.grid(True)
plt.legend()
plt.title('Класифікація тестових даних')
plt.xlabel('X')
plt.ylabel('Y')

# Відображення графіку
plt.show()
```

### Класифікація тестових даних



## ВИСНОВКИ

В результаті виконаної лабораторної роботи розроблені моделі нейронних мереж для апроксимації функцій.

Усі матеріали викладені у репозіторії GitHub, за посиланням <https://github.com/Max11mus/LAB1-Modern-Methods-and-Models-of-Intelligent-Control-Systems.git>.