МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-наукового інституту атомної і теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ЗВІТ

ДО ВИКОНАННЯ ПРАКТИЧНОГО ЗАВДАННЯ №2

з дисципліни

«МЕТОДОЛОГІЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ ПРОГРАМ »

Тема: «Розпізнавання образів за допомогою штучних нейронних мереж»

Варіант 16(2)

КИЇВ 2025

Мета: Ознайомитися з методами розпізнавання образів за допомогою штучних нейронних мереж, побудувати, навчити та протестувати нейрону мережу для розпізнавання букв.

Опис дії алгоритму

Для навчання та перевірки роботи нейронної мережі створюються два окремі файли: train.csv та test.csv. У цих файлах містяться зображення розміром 6×6, що представлені у вигляді матриць з нулів і одиниць та відповідають наступним фігурам:

- Коло
- Квадрат
- Ромб
- Еліпс
- Трикутник

У навчальному наборі ці цифри мають чітку, еталонну форму, тоді як у тестовому наборі деякі пікселі (2–3) навмисно змінені), щоб перевірити здатність моделі до узагальнення. Для кожного зображення також наведено правильну відповідь у вигляді трьохбітного бінарного коду. Як приклад — вміст файлу train.csv:

001100
010010
100001
100001
010010
001100
000
000000
011110
011110
011110
011110
000000
100
001000
011100
111110
111110
011100
001000
010
000000
011110
100001
100001
011110
000000

001
100000
110000
111000
111100
111110
111111
111

Для оцінки ефективності роботи нейронної мережі за різних конфігурацій — таких як тип функції активації, кількість прихованих шарів та кількість нейронів у кожному з них — використано об'єктно-орієнтований підхід. Це дозволяє задавати всі зазначені параметри безпосередньо у конструкторі класу нейромережі. Для повноти експерименту реалізовано набір функцій активації (сигмоїда, ReLU, тангенс гіперболічний) разом з їх похідними. Навчання здійснюється за допомогою алгоритму зворотного поширення помилки, а в ролі функції втрат використовуються середньоквадратична похибка (MSE) та середня абсолютна похибка (MAE).

Щоб перевірити стійкість моделі, у тестовий набір внесено незначний шум шляхом зміни кількох пікселів, тоді як тренувальні зображення збережено у початковому, автентичному вигляді.

Лістинг програми

```
import csv
import math
import random
import matplotlib.pyplot as plt

filepath = "train.csv"

def get_data(filepath):
    X = []
    y = []
    with open(filepath) as f:
        for i, row in enumerate(csv.reader(f)):
            goal_index = i // 7
            row = row[0]
            if len(row) == 6:
                if len(X) <= goal_index:
                    X.append([])
                X[goal_index].extend(map(int, row))
            elif len(row) == 3:
                if len(y) <= goal_index:
                    y.append([])
                y[goal_index].extend(map(int, row))
    return X, y

import math
import random

class NeuralNetwork:
```

```
def __init__(self, layer_sizes, learning_rate=0.1, activation="sigmoid", loss="mse"):
    self.layer_sizes = layer_sizes
    self.learning_rate = learning_rate
    self.activation = activation
    self.loss = loss

    # Ініціалізація ваг і зсувів
    self.weights = [
        [
            [random.uniform(-1, 1) for _ in range(layer_sizes[l])]
            for _ in range(layer_sizes[l + 1])
        ] for l in range(len(layer_sizes) - 1)
    ]
    self.biases = [
        [random.uniform(-1, 1) for _ in range(layer_sizes[l + 1])]
        for l in range(len(layer_sizes) - 1)
    ]

    # Збереження виходів
    self.outputs = [[0.0 for _ in range(size)] for size in layer_sizes]

    # Мапа для виводу результатів
    self.results = {
        (0, 0 ,0): "коло", (1, 0, 0): "квадрат", (0, 1, 0): "ромб",
        (0, 0, 1): "еліпс", (1, 1, 1): "трикутник"
    }

def activate(self, x):
    if self.activation == "sigmoid":
        return 1 / (1 + math.exp(-x))
    elif self.activation == "relu":
        return max(0, x)
    elif self.activation == "tanh":
        return math.tanh(x)
    return x

def activate_derivative(self, activated_x):
    if self.activation == "sigmoid":
        return activated_x * (1 - activated_x)
    elif self.activation == "relu":
        return 1 if activated_x > 0 else 0
    elif self.activation == "tanh":
        return 1 - activated_x ** 2
    return 1

def forward(self, x):
    self.outputs[0] = x[:]
    for l in range(1, len(self.layer_sizes)):
        for j in range(self.layer_sizes[l]):
            weighted_sum = sum(
                self.weights[l - 1][j][k] * self.outputs[l - 1][k]
                for k in range(self.layer_sizes[l - 1])
            ) + self.biases[l - 1][j]
            self.outputs[l][j] = self.activate(weighted_sum)
```

```
        return self.outputs

    def mse_loss(self, y_true, y_pred):
        return sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred)) / len(y_true)

    def mae_loss(self, y_true, y_pred):
        return sum(abs(yt - yp) for yt, yp in zip(y_true, y_pred)) / len(y_true)

    def backward(self, y_true):
        num_layers = len(self.layer_sizes)
        local_gradients = [[0.0 for _ in range(size)] for size in self.layer_sizes[1:]]

        # Градієнти вихідного шару
        for i in range(self.layer_sizes[-1]):
            output = self.outputs[-1][i]
            error = y_true[i] - output
            local_gradients[-1][i] = error * self.activate_derivative(output)

        # Зворотне поширення градієнтів
        for l in reversed(range(len(self.layer_sizes) - 1)):
            for i in range(self.layer_sizes[l + 1]):
                for j in range(self.layer_sizes[l]):
                    delta = self.learning_rate * local_gradients[l][i] * self.outputs[l][j]
                    self.weights[l][i][j] += delta
                self.biases[l][i] += self.learning_rate * local_gradients[l][i]

            if l > 0:
                for j in range(self.layer_sizes[l]):
                    gradient_sum = sum(
                        self.weights[l][i][j] * local_gradients[l][i]
                        for i in range(self.layer_sizes[l + 1])
                    )
                    local_gradients[l - 1][j] = gradient_sum * self.activate_derivative(self.outputs[l][j])

        # Повернення втрати
        if self.loss == 'mse':
            return self.mse_loss(y_true, self.outputs[-1])
        elif self.loss == 'mae':
            return self.mae_loss(y_true, self.outputs[-1])
        return 0

    def train(self, X_train, y_train, epochs=1000):
        for epoch in range(epochs):
            total_loss = 0
            for x, y in zip(X_train, y_train):
                self.forward(x)
                loss = self.backward(y)
                total_loss += loss
            if epoch % 100 == 0:
                print(f"Епоха {epoch}, середня втрата: {total_loss / len(X_train):.6f}")

    def evaluate(self, X_test, y_test):
        correct = 0
        for x, y in zip(X_test, y_test):
```

```
            raw_pred = self.forward(x)[-1]  # беремо тільки вихідний шар
            pred = [1 if p >= 0.5 else 0 for p in raw_pred]
            pred_fig = self.results.get(tuple(pred), '?')
            true_fig = self.results.get(tuple(y), '?')
            if pred_fig == true_fig:
                correct += 1
        accuracy = correct / len(X_test)
        print(f"Точність: {accuracy * 100}%")

    def visualize_predictions(self, X_test, y_test, num_samples=5):
        fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))
        for i in range(num_samples):
            x, y = X_test[i], y_test[i]
            raw_pred = self.forward(x)[-1]  # беремо лише вихід нейромережі
            pred = [1 if p >= 0.5 else 0 for p in raw_pred]  # округлення до 0 або 1
            # Перетворення списку x у 2D-масив 6x6
            image = [x[j*6:(j+1)*6] for j in range(6)]

            axes[i].imshow(image, cmap="Blues")
                        axes[i].set_title(f"Прогноз:    {self.results.get(tuple(pred),    '?')}\nПравильна:
{self.results.get(tuple(y), '?')}")
            axes[i].axis("off")
        plt.show()


X_train, y_train = get_data("train.csv")
X_test, y_test = get_data("test.csv")

nn = NeuralNetwork(layer_sizes=[36, 12, 3], learning_rate=0.1, activation="sigmoid", loss="mse")
nn.train(X_train, y_train, epochs=1000)
nn.evaluate(X_test, y_test)
nn.visualize_predictions(X_test, y_test, num_samples=5)
```

Тестування

Проведемо експериментальне тестування нейронної мережі з наступними чотирма конфігураціями:

- **Конфігурація 1**:
  - Архітектура: 2 шари (вхідний та вихідний)
  - Функція активації: сигмоїда
  - Швидкість навчання: 0.01
- Результат: 100% точність класифікації

- **Конфігурація 2**:
  - Архітектура: та сама, 2 шари
  - Функція активації: ReLU
  - Швидкість навчання: 0.01
- Результат: точність 60%

- **Конфігурація 3**:
  - Архітектура: додано один прихований шар з 12 нейронами

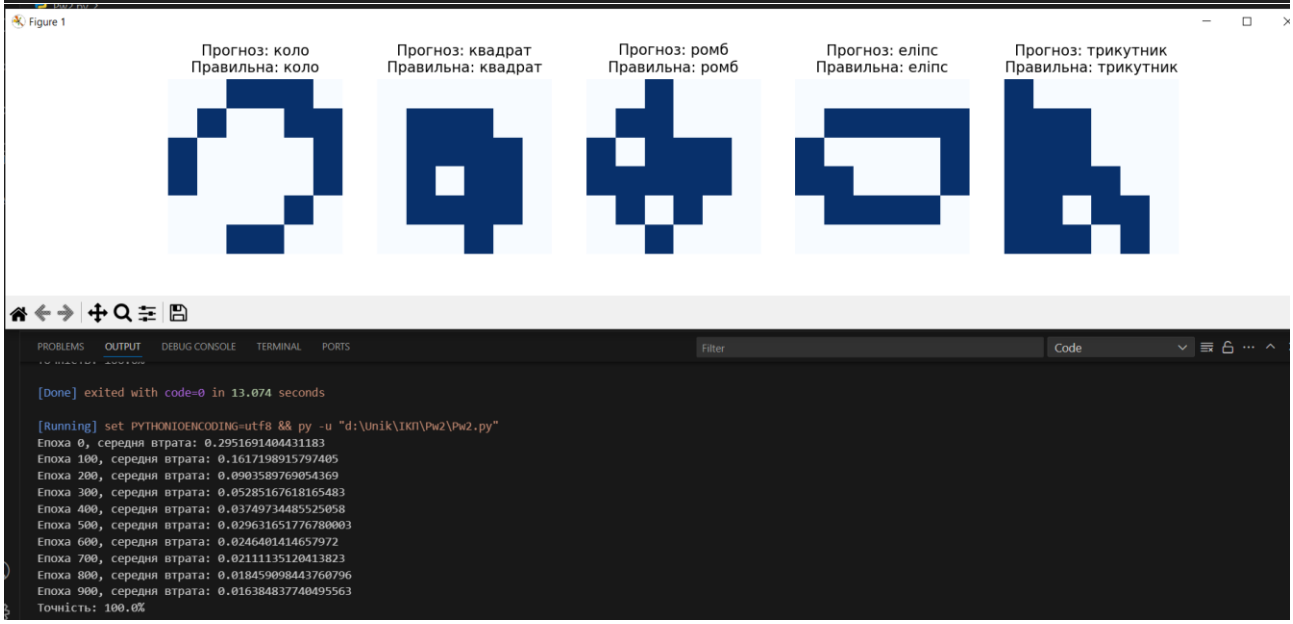
    - Функція активації: сигмоїда
    - Швидкість навчання: 0.1
- Результат: точність знову досягає 100%

- **Конфігурація 4**:
    - Архітектура: додано прихований шар
    - Функція активації: гіперболічний тангенс (tanh)
    - Швидкість навчання: 0.01
- Результат: точність становить 60%

Скріншоти тестування

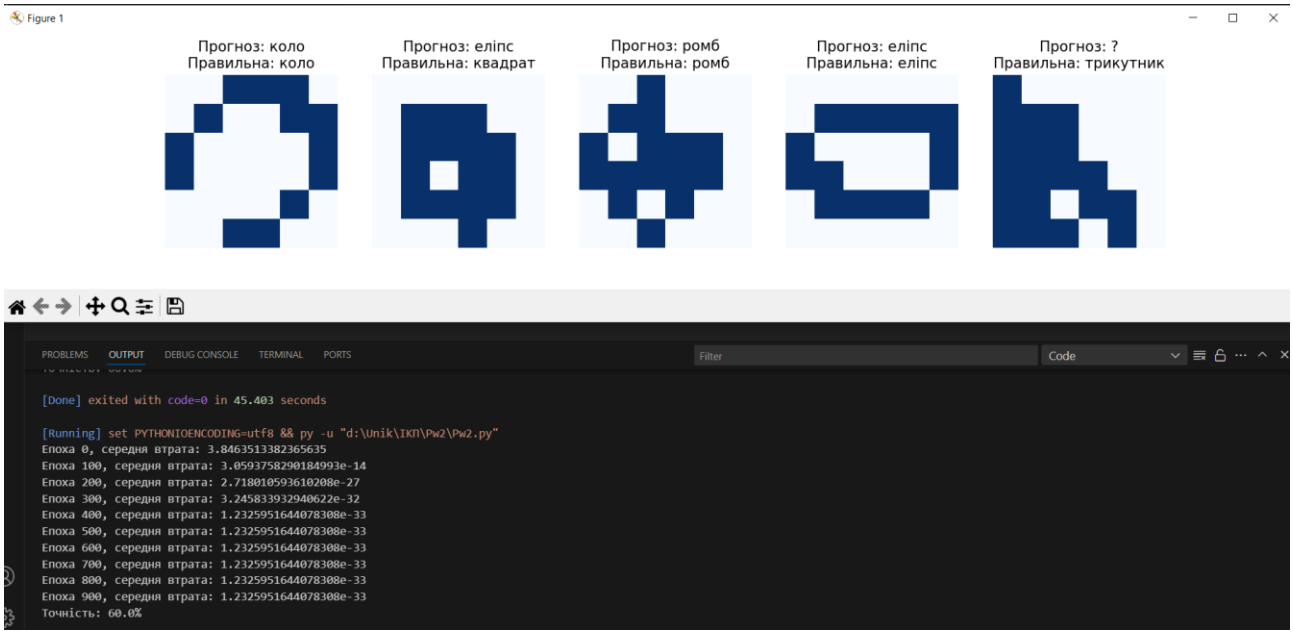
Конфігурація 1:

```
nn = NeuralNetwork(layer_sizes=[36, 3], learning_rate=0.01, activation="sigmoid", loss="mse")
nn.train(X_train, y_train, epochs=1000)
nn.evaluate(X_test, y_test)
nn.visualize_predictions(X_test, y_test, num_samples=5)
```



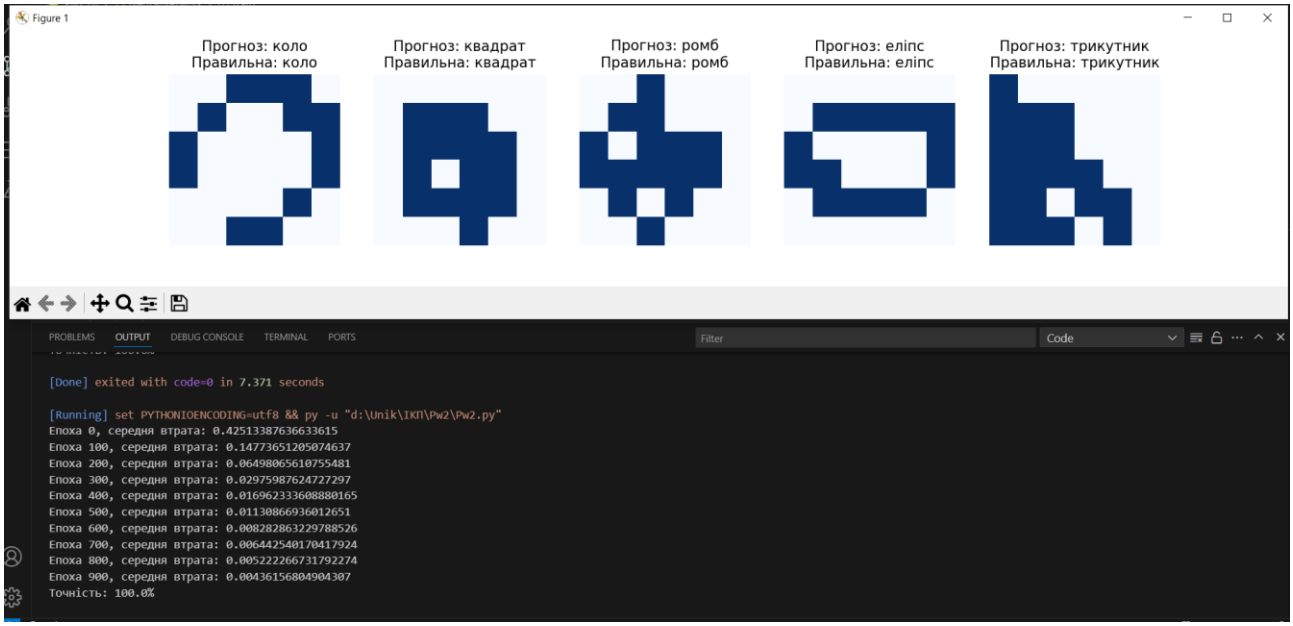
Конфігурація 2:

```
nn = NeuralNetwork(layer_sizes=[36, 3], learning_rate=0.01, activation="ReLu", loss="mse")
nn.train(X_train, y_train, epochs=1000)
nn.evaluate(X_test, y_test)
nn.visualize_predictions(X_test, y_test, num_samples=5)
```



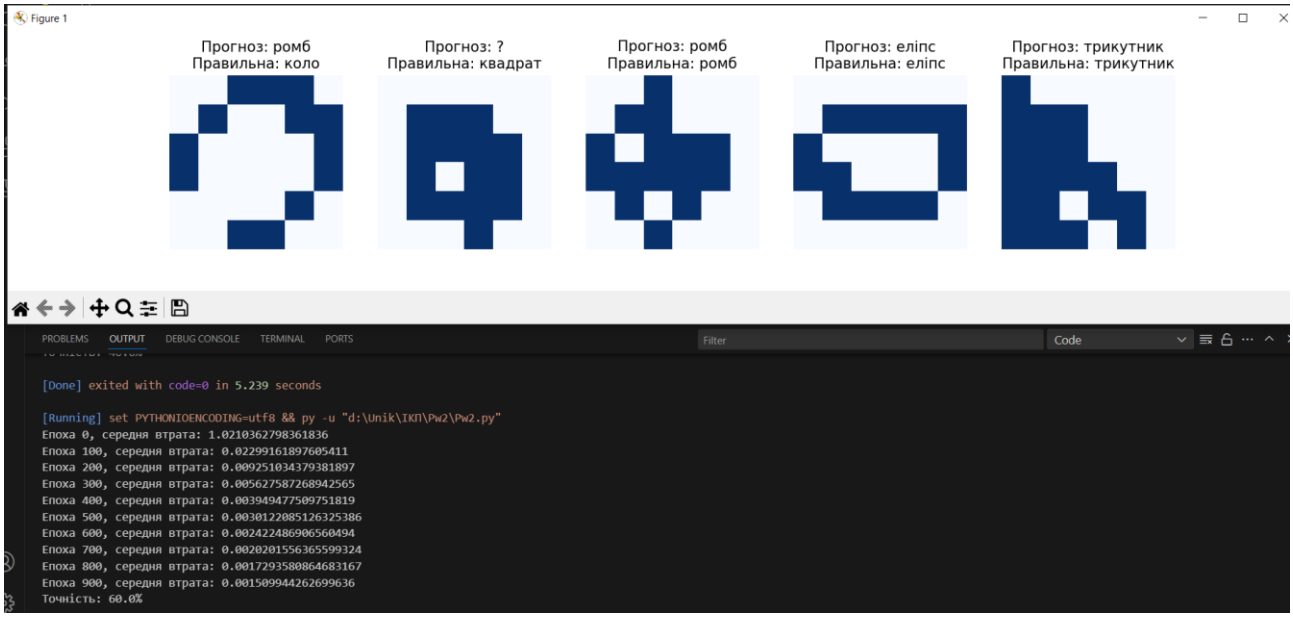
Конфігурація 3:

```
nn = NeuralNetwork(layer_sizes=[36, 12, 3], learning_rate=0.1, activation="sigmoid", loss="mse")
nn.train(X_train, y_train, epochs=1000)
nn.evaluate(X_test, y_test)
nn.visualize_predictions(X_test, y_test, num_samples=5)
```



Конфігурація 4:

```
nn = NeuralNetwork(layer_sizes=[36, 8, 3], learning_rate=0.01, activation="tanh", loss="mse")
nn.train(X_train, y_train, epochs=1000)
nn.evaluate(X_test, y_test)
nn.visualize_predictions(X_test, y_test, num_samples=5)
```



Після проведення тестування з різними комбінаціями вхідних параметрів для побудови нейронної мережі були виявлені такі основні закономірності та тенденції:

1. Найкращі результати класифікації зазвичай демонструють мережі з сигмоїдною функцією активації. Функція ReLU і тангенс дещо поступаються
2. Зменшення швидкості навчання, хоч і не завжди, але часто сприяє підвищенню точності класифікації.
3. Додавання додаткових прихованих шарів позитивно впливає на якість роботи нейронної мережі при використанні ReLU, тоді як для сигмоїдної та тангенсної функцій активації це може навпаки знижувати точність розпізнавання.

Висновок

На даній практичній роботі було досліджено вплив архітектури нейронної мережі, функції активації та швидкості навчання на розпізнавання геометричних фігур з матриць 6 на 6. В нашому випадку сигмоїдальна функція активації показала кращу точність ніж дві інші. Додавання прихованих шарів не сильно впливає на погіршення результатів нейронної мережі