Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Модульна контрольна робота

з дисципліни «Методи та технології штучного інтелекту»

Виконала:
студентка групи ІС-12
Павлова Софія

Викладач:
Шимкович В. М.

Київ – 2023

1. Постановка задачі

Завдання:

Реалізувати нейронну мережу прямого поширення, навчити нейронну мережу генетичним алгоритмом моделювати функцію двох змінних.

* +10 – до навчання нейронної мережі генетичним алгоритмом додати ініціалізацію хромосом початкової популяції за допомогою модифікованого методу Нгуєна-Відроу, що враховує апріорну інформацію про значущість ознак при ініціалізації нейронної мережі.

108	Павлова Софія Олегівна	2-4-6-10-6-2-1	$z = \cos(x/2) + y \cdot \cos(x)$

Рисунок 1 – завдання за варіантом

2. Виконання

Нейронна мережа:

Для вирішення задачі навчання нейронної мережі моделювати функцію двох змінних, реалізуємо нейронну мережу прямого поширення для зручного і ефективного налаштовування архітектури нейронної мережі та ваги її шарів для навчання на конкретній функції.

Створимо нейронну мережу прямого поширення з **6 шарами** і відповідною кількістю **нейронів: 2-4-6-10-6-2-1**.

Додамо ініціалізацію хромосом початкової популяції за допомогою модифікованого **методу Нгуєна-Відроу**. Метод Нгуєна-Відроу враховує апріорну інформацію про значущість ознак при ініціалізації ваг нейронної мережі.

Лістинг:

```
import torch
import torch.nn as nn
import torch.nn.init as init
import numpy as np
import random
import matplotlib.pyplot as plt
import warnings

# Вимкнення варнінгів PyTorch
warnings.filterwarnings("ignore", category=UserWarning)

# Функція Z
def z_func(x, y):
    return np.cos(x / 2) + y * np.cos(x)

# Архітектура нейронної мережі
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(2, 4)
        self.layer2 = nn.Linear(4, 6)
        self.layer3 = nn.Linear(6, 10)
        self.layer4 = nn.Linear(10, 6)
        self.layer5 = nn.Linear(6, 2)
        self.layer6 = nn.Linear(2, 1)

        # Модифікований метод Нгуєна-Відроу для ініціалізації ваг
        init.xavier_uniform_(self.layer1.weight, gain=init.calculate_gain('relu'))
        init.xavier_uniform_(self.layer2.weight, gain=init.calculate_gain('relu'))
        init.xavier_uniform_(self.layer3.weight, gain=init.calculate_gain('relu'))
        init.xavier_uniform_(self.layer4.weight, gain=init.calculate_gain('relu'))
        init.xavier_uniform_(self.layer5.weight, gain=init.calculate_gain('relu'))
        init.xavier_uniform_(self.layer6.weight, gain=init.calculate_gain('linear'))

    # Прямий прохід
    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = torch.relu(self.layer3(x))
        x = torch.relu(self.layer4(x))
        x = torch.relu(self.layer5(x))
        x = self.layer6(x)

        return x
```

Генетичний алгоритм:

Розробимо генетичний алгоритм призначений для **оптимізації параметрів нейронної мережі** з метою наближення до значень функції двох змінних.

Лістинг:

```
# Генетичний алгоритм для навчання нейронної мережі
class GeneticAlgorithm:
    def __init__(self, population_size, generations, mutation_rate):
        self.population_size = population_size
        self.generations = generations
        self.mutation_rate = mutation_rate
        self.population = [NeuralNetwork() for _ in range(population_size)]

    # Розрахунок пристосованості індивідів
    def calculate_fitness(self, model):
        # Розрахунок середньоквадратичної помилки (MSE) для моделі
        inputs = torch.tensor(np.random.rand(100, 2), dtype=torch.float32)
        outputs = model(inputs)
        targets = torch.tensor(z_func(inputs[:, 0], inputs[:, 1]),
dtype=torch.float32).view(-1, 1)
        loss = nn.MSELoss()(outputs, targets)

        return 1 / (loss.item() + 1e-6)

    # Операція кросовера для створення нащадка
    def crossover(self, parent1, parent2):
        child = NeuralNetwork()
        for param, param1, param2 in zip(child.parameters(), parent1.parameters(),
parent2.parameters()):
            if random.random() < 0.5:
                param.data.copy_(param1.data)
            else:
                param.data.copy_(param2.data)

        return child

    # Операція мутації
    def mutate(self, model):
        for param in model.parameters():
            if random.random() < self.mutation_rate:
                param.data += torch.randn_like(param.data) * 0.1

        return model

    # Вибір індивіда з популяції на основі значень функції
    def select(self):
        fitness_scores = [self.calculate_fitness(model) for model in self.population]
        probabilities = np.array(fitness_scores) / sum(fitness_scores)
        selected_indices = np.random.choice(range(self.population_size),
size=self.population_size, p=probabilities)
        selected_models = [self.population[i] for i in selected_indices]

        return selected_models

    # Тренування нейромережі за допомогою генетичного алгоритму
    def train(self, start, end, step):
        x_values = np.arange(start, end, step)
        y_values = np.arange(start, end, step)
        x, y = np.meshgrid(x_values, y_values)
        z = z_func(x, y)

        # Виведення графіка функції Z у 3D на початку
        plot_3d(x, y, z, 'Реальні значення функції Z')

        print('\nАрхітектура гібридної нейромережі:')
        print(self.population[0])
```

```
        print('\nНавчання нейромережі:')
        fitness_history = []
        loss_history = []

        for generation in range(self.generations):
            selected_models = self.select()


            # Створення нового покоління
            new_population = []
            for _ in range(self.population_size):
                parent1, parent2 = random.sample(selected_models, 2)
                child = self.crossover(parent1, parent2)
                child = self.mutate(child)
                new_population.append(child)

            self.population = new_population

            # Обчислення функції втрат для найкращої моделі
            best_model = max(self.population, key=self.calculate_fitness)
            fitness_history.append(self.calculate_fitness(best_model))

            # Розрахунок середньоквадратичної помилки (MSE) для моделі
            inputs = torch.tensor(np.random.rand(100, 2), dtype=torch.float32)
            outputs = best_model(inputs)
            targets = torch.tensor(z_func(inputs[:, 0], inputs[:, 1]),
dtype=torch.float32).view(-1, 1)
            loss = nn.MSELoss()(outputs, targets)
            loss_history.append(loss.item())

            # Виведення результатів на кожному поколінні
            print(f"Generation {generation + 1},\tFitness: {fitness_history[-1]}, \tLoss:
{loss_history[-1]}")

        # Виведення графіків функції втрат та фітнесу
        self.plot_2d(fitness_history, 'Зміна функції пристосованості', 'Пристосованість')
        self.plot_2d(loss_history, 'Зміна функції втрат', 'Втрати')

        return best_model

    # Графіки функції втрат та фітнесу
    def plot_2d(self, history, title, y_lable):
        plt.plot(range(1, self.generations + 1), history, label=y_lable)
        plt.title(title)
        plt.xlabel('Покоління')
        plt.ylabel(y_lable)
        plt.legend()
        plt.show()
```

Використання:

Потренуємо нейронну мережу з параметрами: ***population_size = 500, generations = 1000, start = -5, end = 5***.

Виведемо графіки функції втрат та пристосованості та порівняємо графіки функції двох змінних з передбаченнями нейромережі.

Лістинг:

```
# Графік функції Z
def plot_3d(x, y, z, title):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z, cmap='viridis')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title(title)
    plt.show()

# Функція для порівняння передбачених та реальних значень на відрізку
def predict(model, start, end, step):
    x_values = np.arange(start, end, step)
    y_values = np.arange(start, end, step)
    x, y = np.meshgrid(x_values, y_values)
    inputs = torch.tensor(np.column_stack((x.flatten(), y.flatten())),
dtype=torch.float32)
    z_pred = model(inputs).detach().numpy().reshape(x.shape)
    plot_3d(x, y, z_pred, 'Передбачені значення функції Z')

# Головні виклики
if __name__ == "__main__":
    # Параметри
    population_size = 500
    generations = 1000
    mutation_rate = 0.3
    start = -5
    end = 5
    step = 0.1

    # Навчання нейронної мережі генетичним алгоритмом
    genetic_algorithm = GeneticAlgorithm(population_size, generations, mutation_rate)
    best_model = genetic_algorithm.train(start, end, step)

    # Передбачення значень функції
    predict(best_model, start, end, step)
```

Результат:

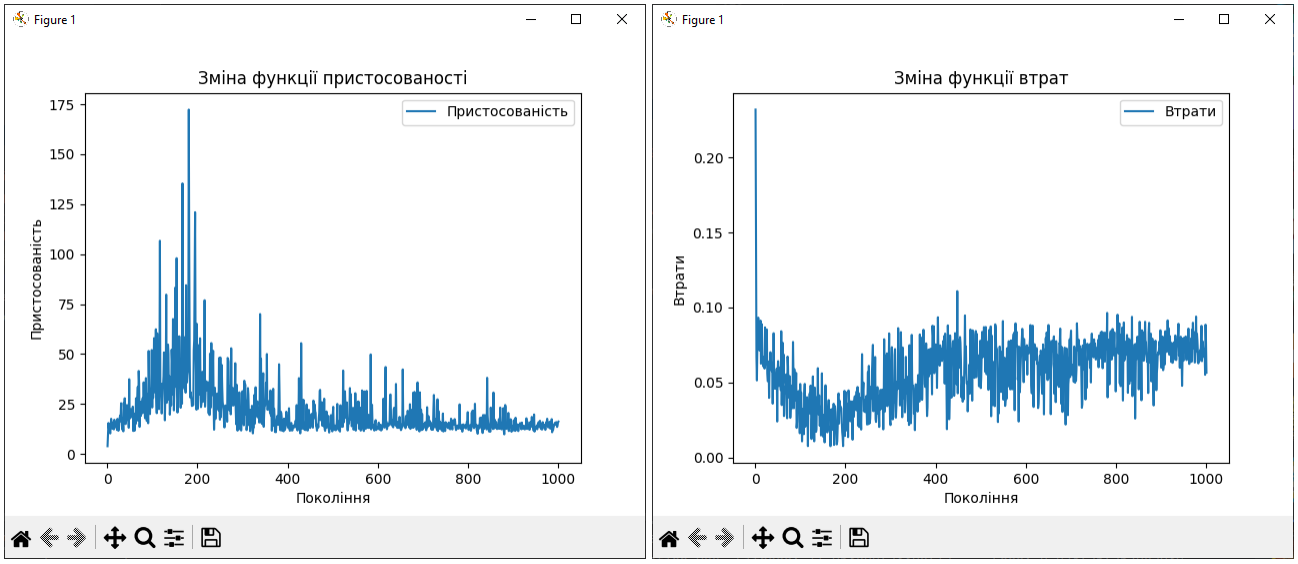

Рисунок 2 – Значення функцій пристосованості та втрат

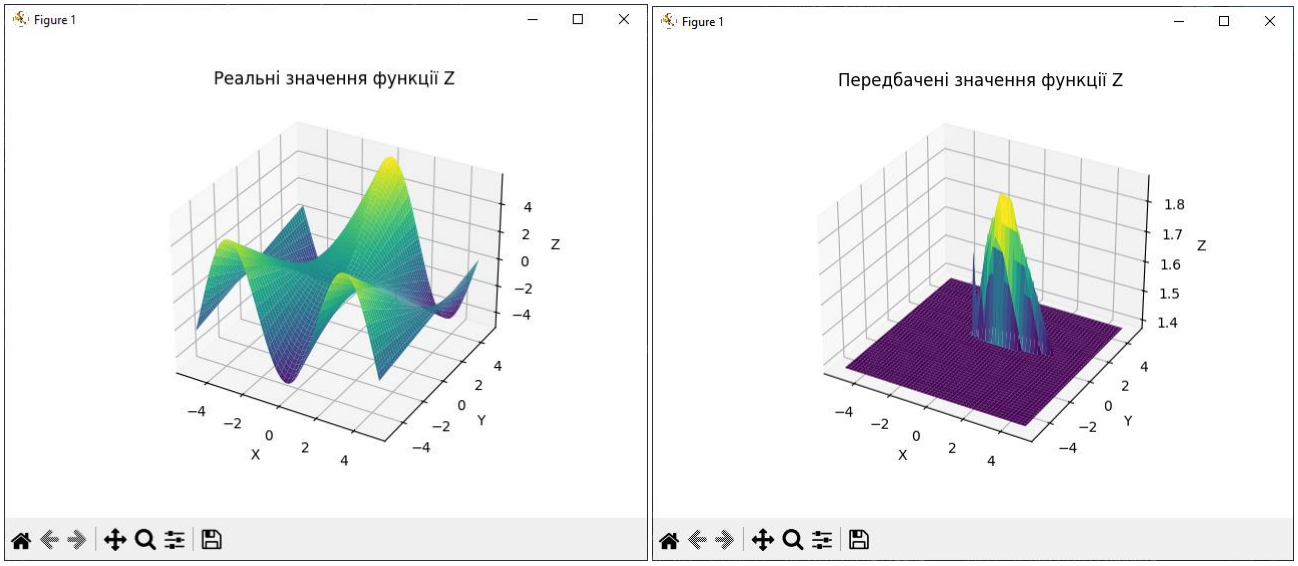

Рисунок 3 – Порівняння результатів функції та нейромережі

З графіків функцій пристосованості та втрат, бачимо, що відбулось перенавчання нейромережі і варто зупинись на 200 епохах.

Потренуємо нейронну мережу з параметрами: ***population_size = 500, generations = 200, start = -5, end = 5***.

Результат:

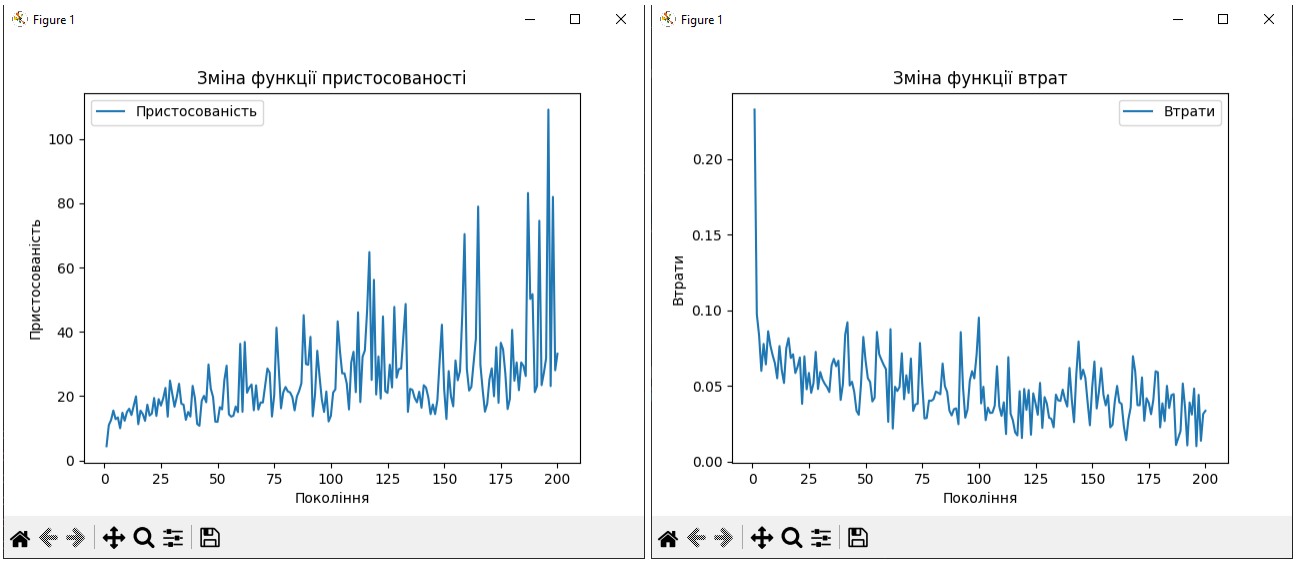

Рисунок 4 – Значення функцій пристосованості та втрат

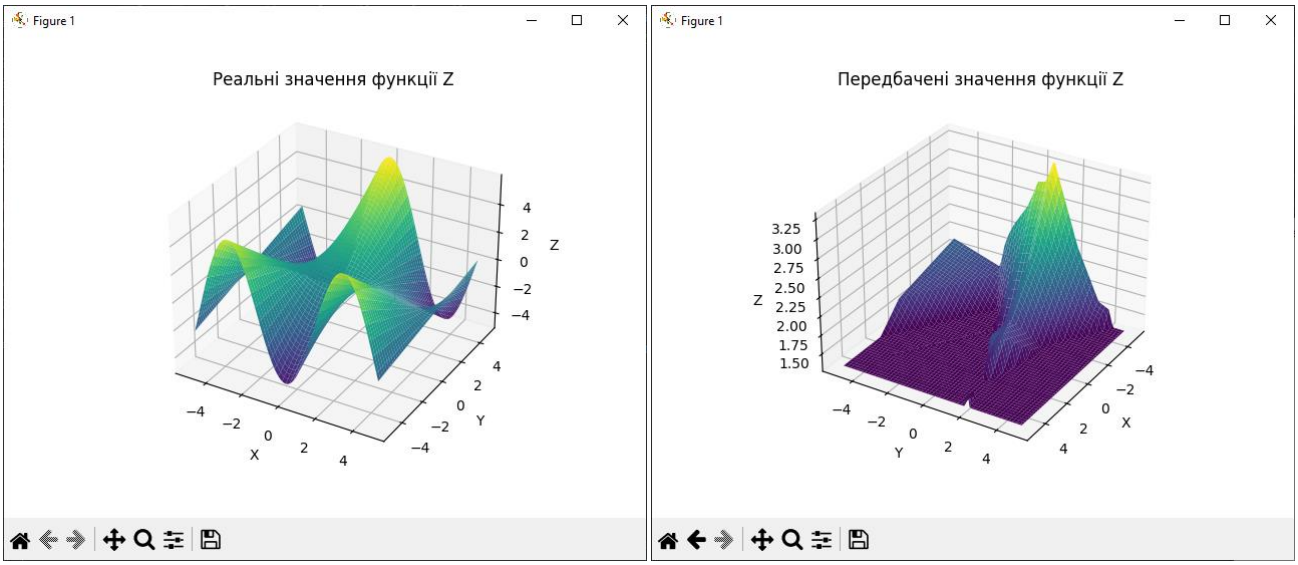

Рисунок 5 – Порівняння результатів функції та нейромережі

З графіку порівняння бачимо, що нейромережа вже значно краще апроксимує функцію.

Висновок:

Під час виконання лабораторної роботи було розроблено генетичний алгоритм для навчання нейронної мережі прямого поширення вирішувати функцію Z(x, y) та імплементовано ініціалізацію хромосом початкової популяції за допомогою модифікованого методу Нгуєна-Відроу.

Розроблена нейромережа, навчена на генетичному алгоритмі гнучка до набору параметрів і в певній мірі апроксимує значення функції двох змінних.

З результатів тестування видно, що нейромережа бачить максимуми і мінімуми функції, утім має тенденцію зриватися з локальних мінімумів при навчанні та зіштовхуватися з проблемою "мертвих" нейронів, коли вихід завжди нуль для певних входів. Для вирішення описаних проблем варто змінити архітектуру нейромережі.