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.
Мельникова К.О.

Перевірив:
Шимкович В. М.

Київ – 2023

Завдання: реалізувати нейронну мережу прямого поширення, навчити нейронну мережу генетичним алгоритмом моделювати функцію двох змінних. При реалізації використати будь-яку зручну для Вас мову програмування.

Варіант:

104	Мельникова Катерина Олександрівна	2-4-8-8-6-6-1	$z = \cos\|y\| + \sin(x + y)$

Хід роботи:

Для початку створимо два класи `MyNN` та `GeneticAlgorithm` що відповідно представлятимуть нейронну мережу та генетичний алгоритм:

```python
from matplotlib import pyplot as plt
import numpy as np
import random
from sklearn.metrics import mean_squared_error

def activation_function(x):
    return 1 / (1 + np.exp(-x))

def function_z(x, y):
    return np.cos(abs(y)) + np.sin(x + y)

zero_layer_nn, first_layer_nn, second_layer_nn, third_layer_nn, fourth_layer_nn, fifth_layer_nn, last_layer_nn = 2, 4, 8, 8, 6, 6,

class MyNN:
    def __init__(self):
        self.weights = [
            np.random.rand(zero_layer_nn, first_layer_nn),
            np.random.rand(first_layer_nn, second_layer_nn),
            np.random.rand(second_layer_nn, third_layer_nn),
            np.random.rand(third_layer_nn, fourth_layer_nn),
            np.random.rand(fourth_layer_nn, fifth_layer_nn),
            np.random.rand(fifth_layer_nn, last_layer_nn)
        ]

    def forward_propagate(self, inputs):
        layers = [inputs]
        for i in range(len(self.weights)):
            layers.append(activation_function(np.dot(layers[i], self.weights[i])))
        return layers[-1]
```

```python
    def learn_errors(self, x_train, y_train):
        errors = []
        for x in x_train:
            for y in y_train:
                z = function_z(x, y)
                prediction = self.forward_propagate(np.array([x, y]))
                error = mean_squared_error(np.array([z]), prediction)
                errors.append(error)
        return errors
```

```python
class GeneticAlgorithm:
    def __init__(self, population_size, mutation_rate, generations_number):
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.generations_number = generations_number
        self.population = self._generate_population(population_size)

    def _generate_population(self, population_size):
        return [MyNN() for _ in range(population_size)]

    def fitness(self, x, y, z): # Оцінка пристосованості кожної особини в популяції
        fitness_scores = []
        for network in self.population:
            output = network.forward_propagate(np.array([x, y]))
            error = np.mean(np.square(output - z))
            fitness_scores.append(1 / (error + 1e-6))
        return fitness_scores

    def select_parents(self, fitness_scores):
        return random.choices(self.population, weights=fitness_scores, k=2)

    def crossover(self, parent1, parent2):
        child = MyNN()
        for i in range(len(child.weights)):
            crossover_point = random.randint(0, len(parent1.weights[i]))
            child.weights[i][:crossover_point] = parent1.weights[i][:crossover_point]
            child.weights[i][crossover_point:] = parent2.weights[i][crossover_point:]
        return child

    def mutate(self, network):
        for i in range(len(network.weights)):
            mutation_mask = np.random.rand(*network.weights[i].shape) < self.mutation_rate
            network.weights[i] += np.random.randn(*network.weights[i].shape) * mutation_mask * 0.1
        return network
```

```python
    def one_gen_cycle(self, x, y, z):
        # Одна ітерація генетичного алгоритму
        fitness_scores = self.fitness(x, y, z)
        new_population = []
        for _ in range(self.population_size):
            parent1, parent2 = self.select_parents(fitness_scores)
            child = self.crossover(parent1, parent2)
            child = self.mutate(child)
            new_population.append(child)
        self.population = new_population
        return min(fitness_scores)

    def main_cycle(self, x_train, y_train):
        # Основний цикл генетичного алгоритму
        print("Виконання генетичного алгоритму...")
        for gen in range(self.generations_number):
            min = []
            for x in x_train:
                for y in y_train:
                    z = function_z(x, y)
                    min.append(self.one_gen_cycle(x, y, z))
            print(f"Кількість пройдених поколінь: {gen+1}/100", end='\r')
        print()
```

```
[119]  ✓ 0.0s                                                    Python
```

Визначимо параметри, навчимо нейронну мережу та оцінимо отримані середньоквадратичні похибки:

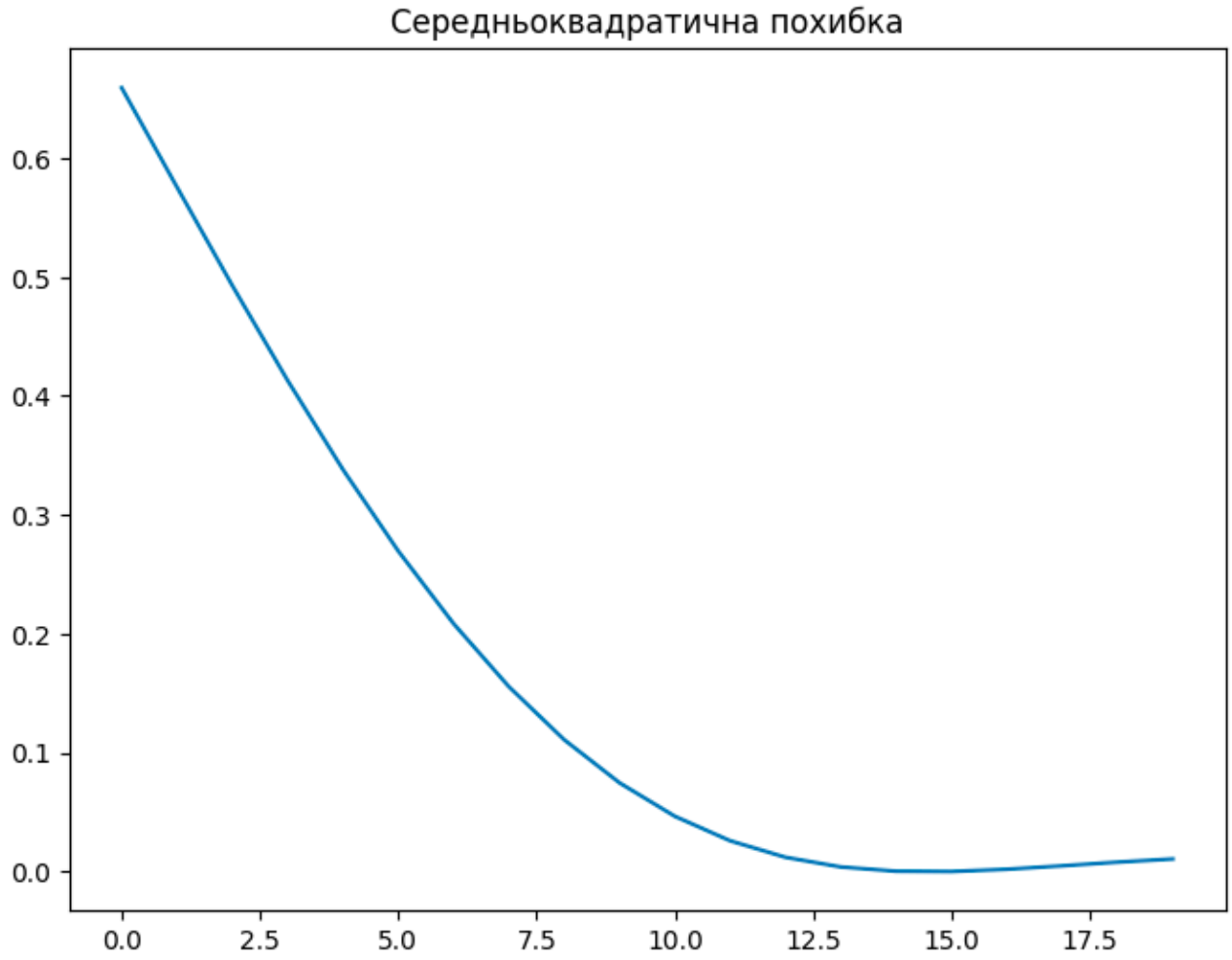
```python
individuals_number = 10
mutation_rate = 0.01
generations_number = 100
genetic_algorithm = GeneticAlgorithm(individuals_number, mutation_rate, generations_number)
x_train = np.linspace(-1, 1, 20)
y_train = np.linspace(-1, 1, 20)

genetic_algorithm.main_cycle(x_train, y_train)

network = genetic_algorithm.population[0]
errors = network.learn_errors(x_train, y_train)
errors_middle = []

print("Оцінка отриманих значень:")
for i, (x, y) in enumerate(zip(x_train, y_train)):
    print(f"Значення X: {round(x, 1)}, Значення Y: {round(y, 1)}, Середньоквадратична похибка: {errors[i]}")
    errors_middle.append(errors[i])
```

[120] ✓ 21.6s Python

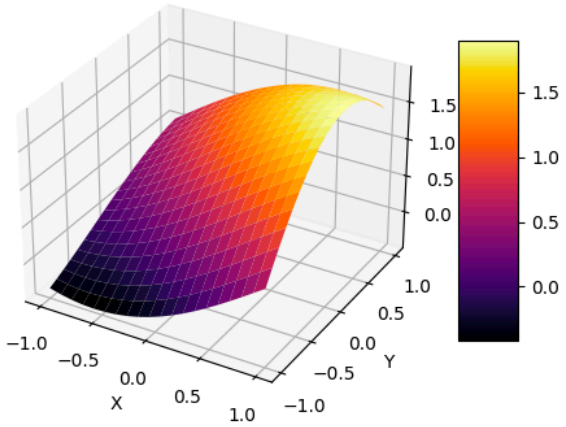
```
Виконання генетичного алгоритму...
Кількість пройдених поколінь: 100/100
Оцінка отриманих значень:
Значення X: -1.0, Значення Y: -1.0, Середньоквадратична похибка: 0.6592386353827527
Значення X: -0.9, Значення Y: -0.9, Середньоквадратична похибка: 0.5757654930368196
Значення X: -0.8, Значення Y: -0.8, Середньоквадратична похибка: 0.4929695432633734
Значення X: -0.7, Значення Y: -0.7, Середньоквадратична похибка: 0.41315838670001026
Значення X: -0.6, Значення Y: -0.6, Середньоквадратична похибка: 0.33825296337875455
Значення X: -0.5, Значення Y: -0.5, Середньоквадратична похибка: 0.26973040409092913
Значення X: -0.4, Значення Y: -0.4, Середньоквадратична похибка: 0.20863647988858416
Значення X: -0.3, Значення Y: -0.3, Середньоквадратична похибка: 0.15562691971553136
Значення X: -0.2, Значення Y: -0.2, Середньоквадратична похибка: 0.11100975616783976
Значення X: -0.1, Значення Y: -0.1, Середньоквадратична похибка: 0.07477908811639968
Значення X: 0.1, Значення Y: 0.1, Середньоквадратична похибка: 0.046641114712962194
Значення X: 0.2, Значення Y: 0.2, Середньоквадратична похибка: 0.02603708370720139
Значення X: 0.3, Значення Y: 0.3, Середньоквадратична похибка: 0.01216860142333031
Значення X: 0.4, Значення Y: 0.4, Середньоквадратична похибка: 0.004031228729130833
Значення X: 0.5, Значення Y: 0.5, Середньоквадратична похибка: 0.000464062930056711
Значення X: 0.6, Значення Y: 0.6, Середньоквадратична похибка: 0.00022492649330343534
Значення X: 0.7, Значення Y: 0.7, Середньоквадратична похибка: 0.002095739304145278
Значення X: 0.8, Значення Y: 0.8, Середньоквадратична похибка: 0.005001393247978387
Значення X: 0.9, Значення Y: 0.9, Середньоквадратична похибка: 0.008097319308939609
Значення X: 1.0, Значення Y: 1.0, Середньоквадратична похибка: 0.010783271776704269
```

Побудуємо графіки:

```python
# Побудова графіка навчання в лінійному масштабі
plt.figure(figsize=(8, 6))
plt.plot(errors_middle)
plt.title('Середньоквадратична похибка')
plt.show()

Z_predicted_for_graph = np.zeros((20, 20))
x_values = np.linspace(-1, 1, 20)
y_values = np.linspace(-1, 1, 20)
z_values = np.array([[function_z(x, y) for x in x_values] for y in y_values])

for i, x in enumerate(x_values):
    for j, y in enumerate(y_values):
        prediction = network.forward_propagate(np.array([x, y]))
        Z_predicted_for_graph[i, j] = prediction[0]

figure = plt.figure(figsize=(12, 6))
X, Y = np.meshgrid(x_values, y_values)
actual_func = figure.add_subplot(121, projection='3d')
surface1 = actual_func.plot_surface(X, Y, z_values, cmap='inferno', edgecolor='none')
actual_func.set_title('Початкова функція z = cos(|y|) + sin(x + y)')
actual_func.set_xlabel('X')
actual_func.set_ylabel('Y')
actual_func.set_zlabel('Z')
figure.colorbar(surface1, ax=actual_func, shrink=0.5, aspect=5)

predicted_func = figure.add_subplot(122, projection='3d')
surface2 = predicted_func.plot_surface(X, Y, Z_predicted_for_graph, cmap='inferno', edgecolor='none')
predicted_func.set_title('Змодульована функція')
predicted_func.set_xlabel('X')
predicted_func.set_ylabel('Y')
predicted_func.set_zlabel('Z')
figure.colorbar(surface2, ax=predicted_func, shrink=0.5, aspect=5)
plt.show()
```



Початкова функція z = cos(|y|) + sin(x + y)



Змодульована функція