МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з модульної контрольної роботи з дисципліни

«Методи та технології штучного інтелекту»

Перевірив:
Шимкович В.М.

Виконав:
студент 3 курсу
групи ІП-11 ФІОТ
Прищепа В.С.

Київ-2023

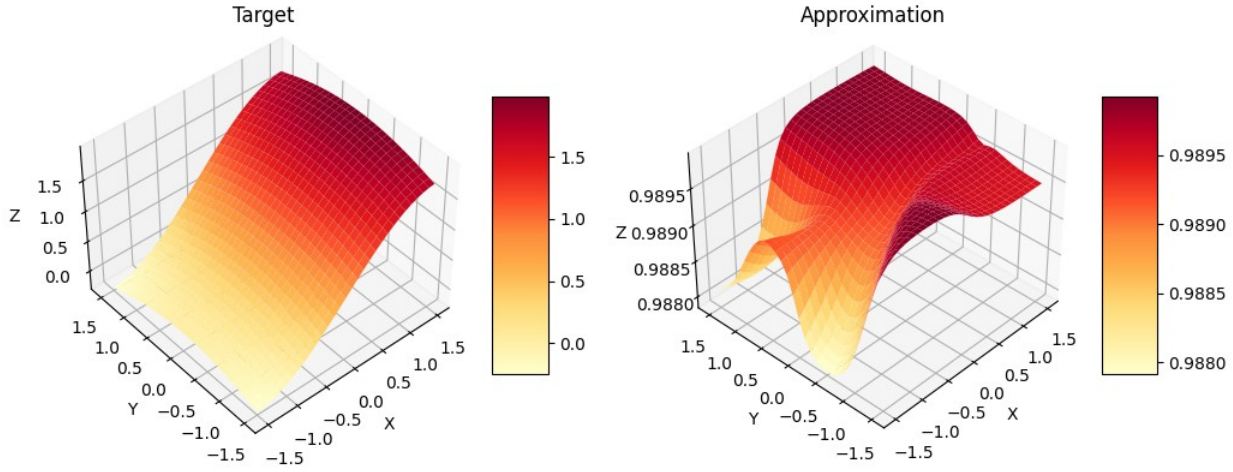
Модульна контрольна робота

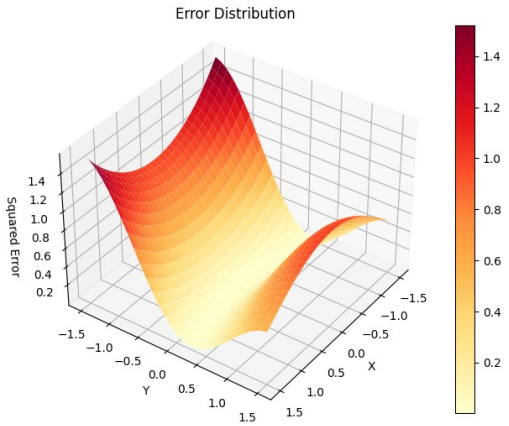
Мета роботи: Реалізувати нейронну мережу прямого поширення, навчити нейронну мережу генетичним алгоритмом моделювати функцію двох змінних. При реалізації використати будь-яку зручну для Вас мову програмування.

Завдання:

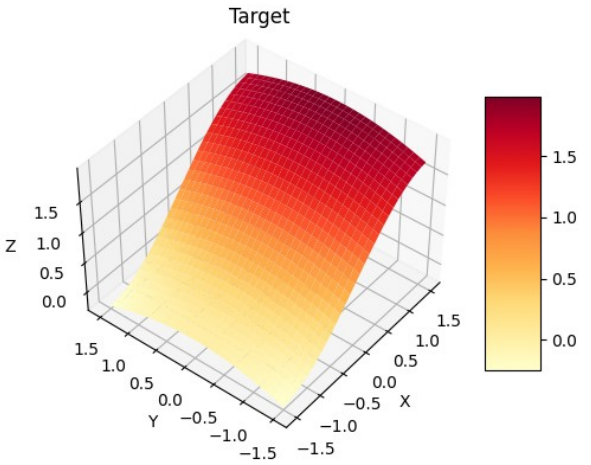
Прищепа Владислав Станіславович	2-4-8-12-8-8-1	$z = \sin(x) + \cos(y/2)$

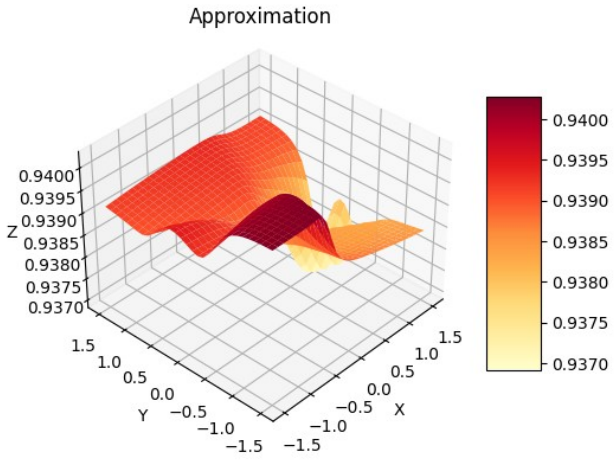
Виконання:

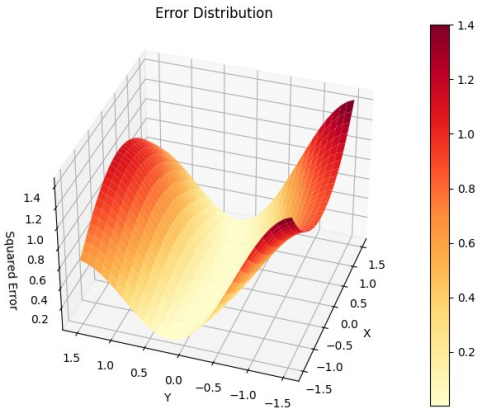




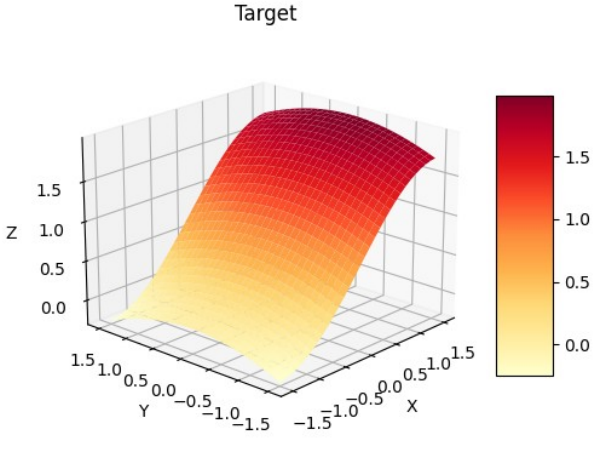

```
Results:
Input Parameters: (-1.5, -1.5), Squared Error: 1.576375999926116;
Input Parameters: (-1.4, -1.4), Squared Error: 1.4944085945599803;
Input Parameters: (-1.3, -1.3), Squared Error: 1.419120078318569;
Input Parameters: (-1.2, -1.2), Squared Error: 1.3503574876755529;
Input Parameters: (-1.1, -1.1), Squared Error: 1.2879540427610414;
Input Parameters: (-1.0, -1.0), Squared Error: 1.2317347427751957;
Input Parameters: (-0.9, -0.9), Squared Error: 1.1815220380340063;
Input Parameters: (-0.8, -0.8), Squared Error: 1.1371408321262493;
Input Parameters: (-0.7, -0.7), Squared Error: 1.098422758947729;
Input Parameters: (-0.6, -0.6), Squared Error: 1.0652097974821084;
Input Parameters: (-0.5, -0.5), Squared Error: 1.0373572631456862;
Input Parameters: (-0.4, -0.4), Squared Error: 1.0147362046363295;
Input Parameters: (-0.3, -0.3), Squared Error: 0.9972353766830074;
Input Parameters: (-0.2, -0.2), Squared Error: 0.9847635155290477;
Input Parameters: (-0.1, -0.1), Squared Error: 0.9772537136116276;
Input Parameters: (0.0, 0.0), Squared Error: 0.9746712282533085;
Input Parameters: (0.1, 0.1), Squared Error: 0.9770189446443014;
Input Parameters: (0.2, 0.2), Squared Error: 0.9843290103558804;
Input Parameters: (0.3, 0.3), Squared Error: 0.9966484106392522;
Input Parameters: (0.4, 0.4), Squared Error: 1.0140378643195236;
Input Parameters: (0.5, 0.5), Squared Error: 1.0365781328227937;
Input Parameters: (0.6, 0.6), Squared Error: 1.064371885491278;
Input Parameters: (0.7, 0.7), Squared Error: 1.097542024134156;
Input Parameters: (0.8, 0.8), Squared Error: 1.1362293759215933;
Input Parameters: (0.9, 0.9), Squared Error: 1.180590176258631;
Input Parameters: (1.0, 1.0), Squared Error: 1.2307931500784302;
Input Parameters: (1.1, 1.1), Squared Error: 1.287016197354147;
Input Parameters: (1.2, 1.2), Squared Error: 1.3494427836322263;
Input Parameters: (1.3, 1.3), Squared Error: 1.4182581222517332;
Input Parameters: (1.4, 1.4), Squared Error: 1.4936452072849145;
Input Parameters: (1.5, 1.5), Squared Error: 1.5757807418057035;
Press any key to continue . . .
```

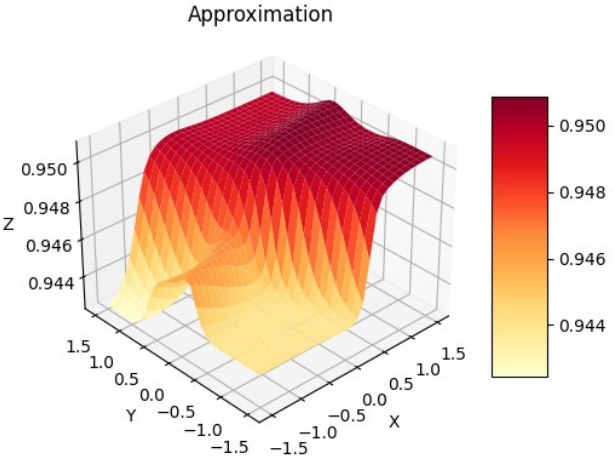


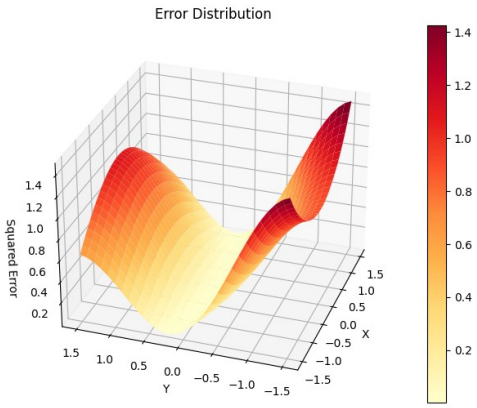





```
Results:
Input Parameters: (-1.5, -1.5), Squared Error: 1.454574701139629;
Input Parameters: (-1.4, -1.4), Squared Error: 1.3757055465000787;
Input Parameters: (-1.3, -1.3), Squared Error: 1.3033960370213968;
Input Parameters: (-1.2, -1.2), Squared Error: 1.237461836660591;
Input Parameters: (-1.1, -1.1), Squared Error: 1.1777129888972275;
Input Parameters: (-1.0, -1.0), Squared Error: 1.123958353726815;
Input Parameters: (-0.9, -0.9), Squared Error: 1.0760104683448413;
Input Parameters: (-0.8, -0.8), Squared Error: 1.0336907103998914;
Input Parameters: (-0.7, -0.7), Squared Error: 0.9968265267136343;
Input Parameters: (-0.6, -0.6), Squared Error: 0.9651883378854756;
Input Parameters: (-0.5, -0.5), Squared Error: 0.9383512255205365;
Input Parameters: (-0.4, -0.4), Squared Error: 0.9160784968552894;
Input Parameters: (-0.3, -0.3), Squared Error: 0.8987681427324071;
Input Parameters: (-0.2, -0.2), Squared Error: 0.8865680861023748;
Input Parameters: (-0.1, -0.1), Squared Error: 0.8793028881098163;
Input Parameters: (0.0, 0.0), Squared Error: 0.8768277194359494;
Input Parameters: (0.1, 0.1), Squared Error: 0.8790773685367921;
Input Parameters: (0.2, 0.2), Squared Error: 0.8860413946795936;
Input Parameters: (0.3, 0.3), Squared Error: 0.8977477895284162;
Input Parameters: (0.4, 0.4), Squared Error: 0.9142550962570961;
Input Parameters: (0.5, 0.5), Squared Error: 0.9356481858237253;
Input Parameters: (0.6, 0.6), Squared Error: 0.9620353648440214;
Input Parameters: (0.7, 0.7), Squared Error: 0.9935458431397284;
Input Parameters: (0.8, 0.8), Squared Error: 1.0303271592798844;
Input Parameters: (0.9, 0.9), Squared Error: 1.0725424037143698;
Input Parameters: (1.0, 1.0), Squared Error: 1.1203671859549271;
Input Parameters: (1.1, 1.1), Squared Error: 1.1739863418944865;
Input Parameters: (1.2, 1.2), Squared Error: 1.2335904015033177;
Input Parameters: (1.3, 1.3), Squared Error: 1.2993718494348159;
Input Parameters: (1.4, 1.4), Squared Error: 1.3715212175768452;
Input Parameters: (1.5, 1.5), Squared Error: 1.4502230520674522;
Press any key to continue . . .
```








```
Results:
Input Parameters: (-1.5, -1.5), Squared Error: 1.4635398825018644;
Input Parameters: (-1.4, -1.4), Squared Error: 1.3844232327070825;
Input Parameters: (-1.3, -1.3), Squared Error: 1.3118802419707798;
Input Parameters: (-1.2, -1.2), Squared Error: 1.2457267742487326;
Input Parameters: (-1.1, -1.1), Squared Error: 1.1857726828858888;
Input Parameters: (-1.0, -1.0), Squared Error: 1.1318258091714986;
Input Parameters: (-0.9, -0.9), Squared Error: 1.0836958369632097;
Input Parameters: (-0.8, -0.8), Squared Error: 1.0411979544186576;
Input Parameters: (-0.7, -0.7), Squared Error: 1.0041562763528575;
Input Parameters: (-0.6, -0.6), Squared Error: 0.9724069884362752;
Input Parameters: (-0.5, -0.5), Squared Error: 0.9458011967568523;
Input Parameters: (-0.4, -0.4), Squared Error: 0.9242075282206393;
Input Parameters: (-0.3, -0.3), Squared Error: 0.9075146958172875;
Input Parameters: (-0.2, -0.2), Squared Error: 0.8956347023709308;
Input Parameters: (-0.1, -0.1), Squared Error: 0.8885086390594068;
Input Parameters: (0.0, 0.0), Squared Error: 0.886120714839427;
Input Parameters: (0.1, 0.1), Squared Error: 0.888536436892807;
Input Parameters: (0.2, 0.2), Squared Error: 0.896005452286771;
Input Parameters: (0.3, 0.3), Squared Error: 0.9092512472569316;
Input Parameters: (0.4, 0.4), Squared Error: 0.9299542762516497;
Input Parameters: (0.5, 0.5), Squared Error: 0.9559361915621065;
Input Parameters: (0.6, 0.6), Squared Error: 0.9841481538806995;
Input Parameters: (0.7, 0.7), Squared Error: 1.0166086354971802;
Input Parameters: (0.8, 0.8), Squared Error: 1.054185007575417;
Input Parameters: (0.9, 0.9), Squared Error: 1.0971696217019569;
Input Parameters: (1.0, 1.0), Squared Error: 1.1457621315985986;
Input Parameters: (1.1, 1.1), Squared Error: 1.200157225189367;
Input Parameters: (1.2, 1.2), Squared Error: 1.2605544150868402;
Input Parameters: (1.3, 1.3), Squared Error: 1.3271541380296494;
Input Parameters: (1.4, 1.4), Squared Error: 1.4001528960010987;
Input Parameters: (1.5, 1.5), Squared Error: 1.4797391239026938;
Press any key to continue . . .
```

Лістинг:

```
from matplotlib import pyplot as plt
import numpy as np
import random

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

def target_function(x, y):
    return np.sin(x) + np.cos(y / 2)

class NeuralNetwork:
    def __init__(self):
        self.weights1 = np.random.rand(2, 4)
        self.weights2 = np.random.rand(4, 8)
        self.weights3 = np.random.rand(8, 12)
        self.weights4 = np.random.rand(12, 8)
        self.weights5 = np.random.rand(8, 8)
        self.weights6 = np.random.rand(8, 1)

    def forward_propagate(self, inputs):
        self.layer1 = sigmoid(np.dot(inputs, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        self.layer3 = sigmoid(np.dot(self.layer2, self.weights3))
        self.layer4 = sigmoid(np.dot(self.layer3, self.weights4))
        self.layer5 = sigmoid(np.dot(self.layer4, self.weights5))
```

```
        self.output = sigmoid(np.dot(self.layer5, self.weights6))
        return self.output

    def calculate_error(self, target):
        return np.mean(np.square(self.output - target))

class GeneticAlgorithm:
    def __init__(self, population_size, mutation_rate):
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.population = [NeuralNetwork() for _ in range(population_size)]

    def calculate_fitness(self, x, y, z):
        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):
        parents = random.choices(self.population, weights=fitness_scores, k=2)
        return parents

    def crossover(self, parent1, parent2):
        child = NeuralNetwork()
        for child_weights, p1_weights, p2_weights in zip([child.weights1,
child.weights2, child.weights3, child.weights4, child.weights5, child.weights6],
[parent1.weights1, parent1.weights2, parent1.weights3, parent1.weights4,
parent1.weights5, parent1.weights6],
[parent2.weights1, parent2.weights2, parent2.weights3, parent2.weights4,
parent2.weights5, parent2.weights6]):
            crossover_point = random.randint(0, len(p1_weights))
            child_weights[:crossover_point] = p1_weights[:crossover_point]
            child_weights[crossover_point:] = p2_weights[crossover_point:]
        return child

    def mutate(self, network):
        for weights in [network.weights1, network.weights2, network.weights3,
network.weights4]:
            mutation_mask = np.random.rand(*weights.shape) < self.mutation_rate
            weights += np.random.randn(*weights.shape) * mutation_mask * 0.1
        return network

    def run_generation(self, x, y, z):
```

```
        fitness_scores = self.calculate_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)

population_size = 10
mutation_rate = 0.01
generations = 100

genetic_algorithm = GeneticAlgorithm(population_size, mutation_rate)
x_for_training = np.linspace(-1.5, 1.5, 31)
y_for_training = np.linspace(-1.5, 1.5, 31)

for generation in range(generations):
    generation_min_fitnesses = []
    for x in x_for_training:
        for y in y_for_training:
            z = target_function(x, y)
            generation_min_fitnesses.append(genetic_algorithm.run_generation(x, y, z))

best_network = genetic_algorithm.population[0]
errors = []
for x in x_for_training:
    for y in y_for_training:
        z = target_function(x, y)
        prediction = best_network.forward_propagate(np.array([x, y]))
        error = np.mean(np.square(prediction - np.array([z])))
        errors.append(error)
print("Results:")
for i, (x, y) in enumerate(zip(x_for_training, y_for_training)):
    print(f"Input Parameters: ({round(x, 1)}, {round(y, 1)}), Squared Error:
{errors[i]};")

Z_nn = np.zeros((31, 31))
x_values = np.linspace(-1.5, 1.5, 31)
y_values = np.linspace(-1.5, 1.5, 31)
z_values = np.array([[target_function(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 = best_network.forward_propagate(np.array([x, y]))
```

```
        Z_nn[i, j] = prediction[0]
fig = plt.figure(figsize=(12, 6))
X, Y = np.meshgrid(x_values, y_values)
ax1 = fig.add_subplot(121, projection='3d')
surf1 = ax1.plot_surface(X, Y, z_values, cmap='YlOrRd', edgecolor='none')
ax1.set_title('Target')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')
fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=5)
ax2 = fig.add_subplot(122, projection='3d')
surf2 = ax2.plot_surface(X, Y, Z_nn, cmap='YlOrRd', edgecolor='none')
ax2.set_title('Approximation')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')
fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=5)
plt.show()
error_matrix = np.reshape(errors, (31, 31))
X, Y = np.meshgrid(x_values, y_values)
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, error_matrix, cmap='YlOrRd')
plt.colorbar(surf)
ax.set_title('Error Distribution')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Squared Error')
plt.show()
```

Висновок: Під час виконання МКР я реалізував ГА, що навчає нейрону мережу. Проаналізувавши результати, можна сказати, що в цілому розроблена нейронна мережа зазвичай видає графік подібний по формі до оригінального, але значення виходять досить не точні. Для того, щоб вона видавала кращі результати, потрібні більший набір даних і кількість поколінь, що напряму впливає на час навчання. Код і результат виконання наведені вище.