Лабораторна робота №5

з дисципліни Основи штучного інтелекту
студента групи ЗІПЗк-22-1
Перехватова Алевтина Олександрівна
дата виконання: 03.12.2023

Мета роботи: : використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.

Виконання роботи

Завдання 2.1

Дія1:

```python
import numpy as np

def sigmoid(x):
    # Наша функція активації: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        # Вхідні дані про вагу, додавання зміщення
        # і подальше використання функції активації

        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)

weights = np.array([0, 1])  # w1 = 0, w2 = 1
bias = 4  # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])  # x1 = 2, x2 = 3
print(n.feedforward(x))
```

Рис.1.1– Код програми

```
0.9990889488055994
```

Рис.1.2 – Реакція програми на дію

Завдання 2.2

Дія1:

```python
import numpy as np

def sigmoid(x):
    # Функція активації sigmoid:: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
    # Похідна від sigmoid: f'(x) = f(x) * (1 - f(x))
    fx = sigmoid(x)
    return fx * (1 - fx)

def mse_loss(y_true, y_pred):
    # y_true и y_pred є масивами numpy з одинаковою довжиною
    return ((y_true - y_pred) ** 2).mean()

class PerehvatovaNeuralNetwork:

    def __init__(self):
        # Вага
        self.w1 = np.random.normal()
        self.w2 = np.random.normal()
        self.w3 = np.random.normal()
        self.w4 = np.random.normal()
        self.w5 = np.random.normal()
        self.w6 = np.random.normal()

        # Зміщення
        self.b1 = np.random.normal()
        self.b2 = np.random.normal()
        self.b3 = np.random.normal()

    def feedforward(self, x):
        # x є масивом numpy з двома елементами
        h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
        h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
        o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
        return o1

    def train(self, data, all_y_trues):

        learn_rate = 0.1
        epochs = 1000 # кількість циклів у всьому наборі даних

        for epoch in range(epochs):
            for x, y_true in zip(data, all_y_trues):
                # --- Виконуємо зворотній зв'язок (ці значання нам потрібні в подальшому )
                sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
                h1 = sigmoid(sum_h1)
```

Рис.1.3– Код програми

```
        sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
        h1 = sigmoid(sum_h1)

        sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
        h2 = sigmoid(sum_h2)

        sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
        o1 = sigmoid(sum_o1)
        y_pred = o1

        # --- Підрахунок часткових похідних
        # --- Найменування: d_L_d_w1 означає "частково L / частково w1"
        d_L_d_ypred = -2 * (y_true - y_pred)

        # Нейрон o1
        d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
        d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
        d_ypred_d_b3 = deriv_sigmoid(sum_o1)

        d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
        d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

        # Нейрон h1
        d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
        d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
        d_h1_d_b1 = deriv_sigmoid(sum_h1)

        # Нейрон h2
        d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
        d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
        d_h2_d_b2 = deriv_sigmoid(sum_h2)

        # --- Оновлюємо вагу і зміщення
        # Нейрон h1
        self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
        self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
        self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

        # Нейрон h2
        self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
        self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
        self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

        # Нейрон o1
        self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
        self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
        self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3
```

Рис.1.4– Код програми

```
            # --- Підраховуємо загальні втрати в кінці кожної фази
            if epoch % 10 == 0:
                y_preds = np.apply_along_axis(self.feedforward, 1, data)
                loss = mse_loss(all_y_trues, y_preds)
                print("Epoch %d loss: %.3f" % (epoch, loss))

# Задання набору даних
data = np.array([
    [-2, -1],    # Alice
    [25, 6],     # Bob
    [17, 4],     # Charlie
    [-15, -6], # Diana
])

all_y_trues = np.array([
    1, # Alice
    0, # Bob
    0, # Charlie
    1, # Diana
])

# Тренуємо вашу нейронну мережу!
network = PerehvatovaNeuralNetwork()
network.train(data, all_y_trues)
# Робимо передбачення
emily = np.array([-7, -3])  # 128 фунтов, 63 дюйма
frank = np.array([20, 2])  # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily))  # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank))  # 0.039 - M
```

Рис.1.5– Код програми

```
Epoch 740 loss: 0.002
Epoch 750 loss: 0.002
Epoch 760 loss: 0.002
Epoch 770 loss: 0.002
Epoch 780 loss: 0.002
Epoch 790 loss: 0.002
Epoch 800 loss: 0.002
Epoch 810 loss: 0.002
Epoch 820 loss: 0.002
Epoch 830 loss: 0.002
Epoch 840 loss: 0.002
Epoch 850 loss: 0.002
Epoch 860 loss: 0.002
Epoch 870 loss: 0.002
Epoch 880 loss: 0.002
Epoch 890 loss: 0.002
Epoch 900 loss: 0.002
Epoch 910 loss: 0.002
Epoch 920 loss: 0.002
Epoch 930 loss: 0.002
Epoch 940 loss: 0.002
Epoch 950 loss: 0.001
Epoch 960 loss: 0.001
Epoch 970 loss: 0.001
Epoch 980 loss: 0.001
Epoch 990 loss: 0.001
Emily: 0.965
Frank: 0.038
```

Рис.1.6 – Реакція програми на дію

Завдання 2.3

Дія1:

```
import neurolab as nl

# Load data from file
text = np.loadtxt('data_perceptron.txt')

# Split data into data points and labels
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))

# Plot input data points
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input Data')
dim1_min, dim1_max, dim2_min, dim2_max = 0, 1, 0, 1

# Number of neurons in the output layer
num_output = labels.shape[1]

dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
perceptron = nl.net.newp([dim1, dim2], num_output)

# Training the perceptron using our data
error_progress = perceptron.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training error progress
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Change in training error')
plt.grid()
plt.show()
```

Рис.1.7– Код програми

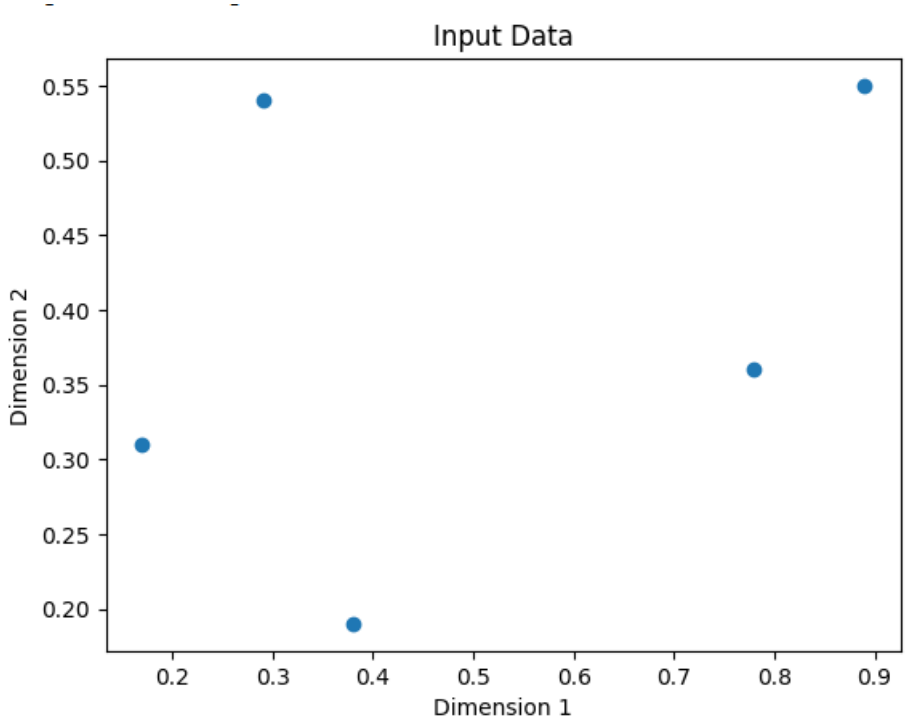


Рис.1.8 – Реакція програми на дію

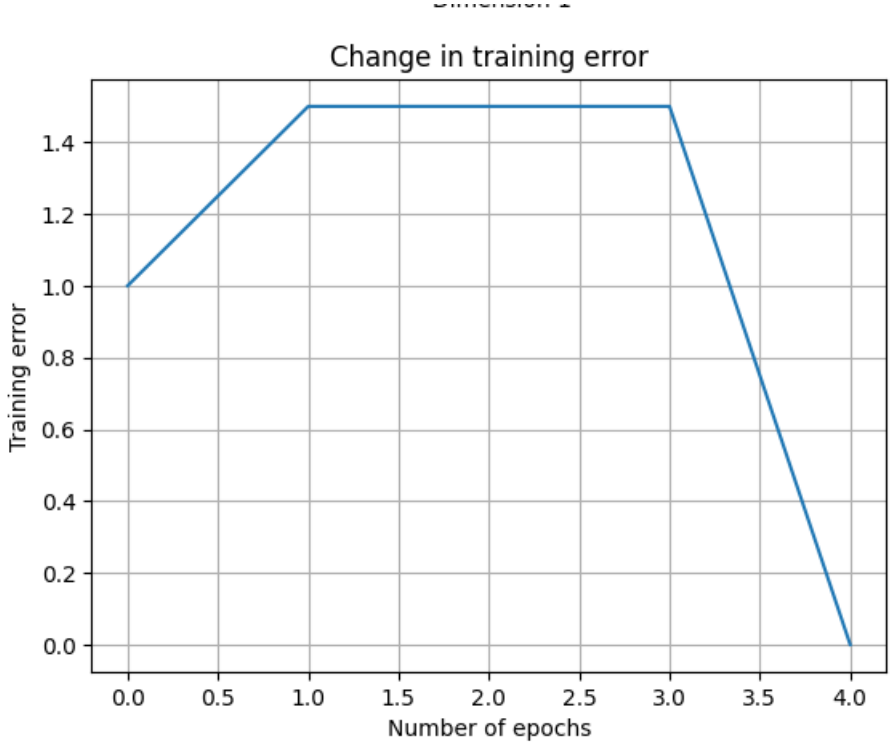


Рис.1.9 – Реакція програми на дію

Висновок:

1. Графік вхідних даних:

Цей графік представляє точки вхідних даних. Кожна точка — це пара значень, взятих із перших двох стовпців завантажених даних. Вісь x

відповідає значенням у першому вимірі («Вимір 1»), а вісь ординат відповідає значенням у другому вимірі («Вимір 2»). Цей графік допомагає візуалізувати розподіл і структуру вхідних даних.

2. Графік прогресу помилок навчання:

Другий графік показує прогрес помилки навчання протягом епох під час навчання перцептрона. Вісь абсцис – кількість епох, а вісь у – похибка навчання. Похибка навчання в ідеалі повинна зменшуватися з епохами, вказуючи на те, що перцептрон навчається правильно класифікувати вхідні дані. Цей графік корисний для оцінки конвергенції та продуктивності перцептрона під час навчання.

Якщо точки вхідних даних мають чітке розділення та шаблон, це означає, що персептрон має шанс навчитися та класифікувати їх точно.

Діаграма прогресу помилки навчання дає уявлення про те, наскільки добре навчається перцептрон. Послідовне зменшення помилок навчання свідчить про успішне навчання.

Завдання 2.4

Дія1:

```
import numpy as np
import matplotlib.pyplot as plt
!pip install neurolab

import neurolab as nl

# Load input data
text = np.loadtxt('data_simple_nn.txt')

# Split data into data points and labels
data = text[:, 0:2]
labels = text[:, 2:4]

# Plot input data points
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input Data')

# Determine the minimum and maximum values for each dimension
dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()

# Define the number of neurons in the output layer
num_output = labels.shape[1]

# Define a single-layer neural network using the provided parameters
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)

# Train the network on training data
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training error progress
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Change in training error')
plt.grid()
plt.show()

# Define some sample test data points and run the neural network for them
print('Test results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

Рис.1.10– Код програми

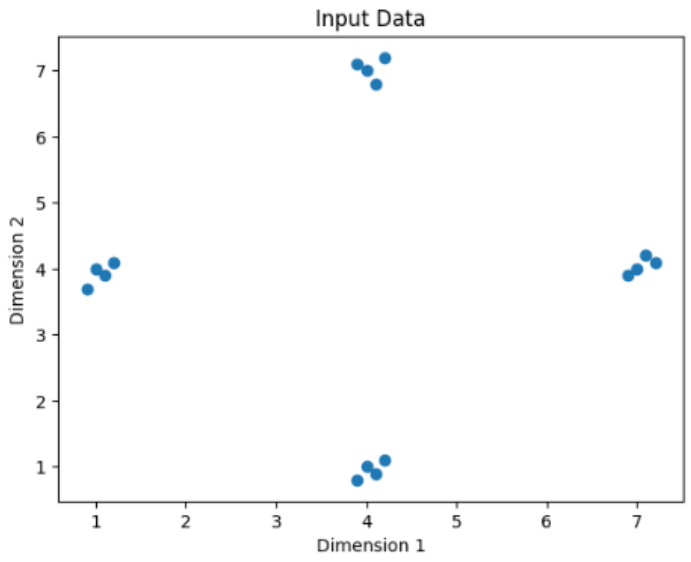
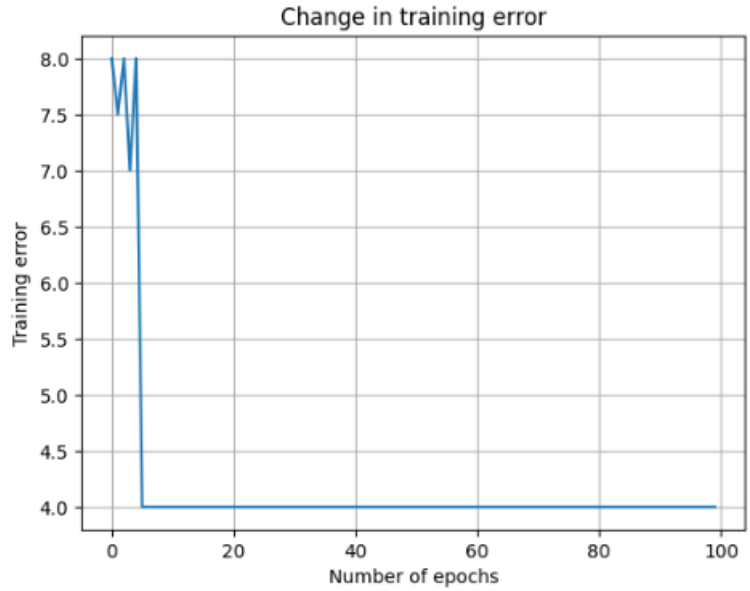


Рис.1.11 – Реакція програми на дію




```
Test results:
[0.4, 4.3] --> [0. 0.]
[4.4, 0.6] --> [1. 0.]
[4.7, 8.1] --> [1. 1.]
```

Рис.1.12 – Реакція програми на дію

Висновок:

1. Графік вхідних даних:

Цей графік показує розподіл точок вхідних даних. Кожна точка представляє пару значень з перших двох стовпців завантажених даних. Це допомагає візуалізувати шаблон і розділення точок даних.

2. Графік прогресу помилок навчання:

Цей графік ілюструє зміну похибки навчання протягом епох під час навчання нейронної мережі. Вісь абсцис – кількість епох навчання, а вісь у – похибка навчання. Тенденція до зменшення помилки навчання свідчить про успішне навчання.

Якщо помилка навчання належним чином зменшується і мережа робить точні прогнози на основі тестових даних, це свідчить про те, що нейронна мережа вивчила основні закономірності в даних.

Однак якщо помилка навчання не зменшується або передбачення є неточними, це може вказувати на проблеми з архітектурою моделі, параметрами навчання або характером даних. Може знадобитися подальший аналіз і коригування.

Завдання 2.5

Дія1:

```
import numpy as np
import matplotlib.pyplot as plt
!pip install neurolab

import neurolab as nl

# Load input data
text = np.loadtxt('data_simple_nn.txt')

# Split data into data points and labels
data = text[:, 0:2]
labels = text[:, 2:4]

# Plot input data points
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input Data')

# Determine the minimum and maximum values for each dimension
dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()

# Define the number of neurons in the output layer
num_output = labels.shape[1]

# Define a single-layer neural network using the provided parameters
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)

# Train the network on training data
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training error progress
plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Change in training error')
plt.grid()
plt.show()

# Define some sample test data points and run the neural network for them
print('Test results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

Рис.1.10– Код програми

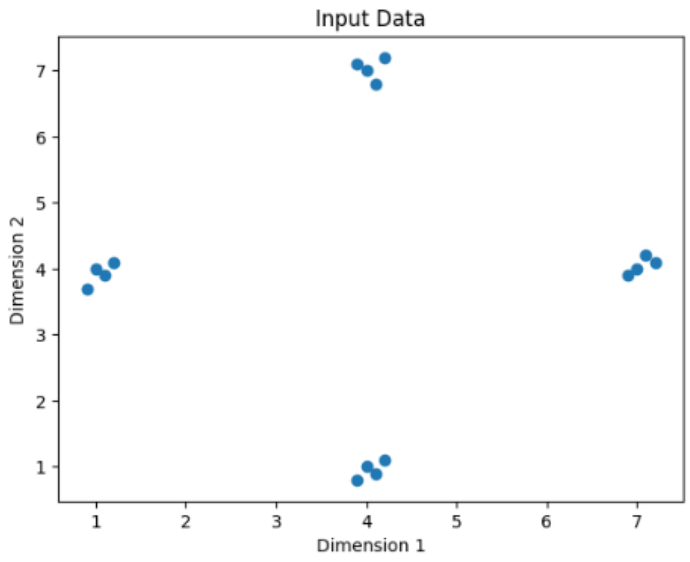
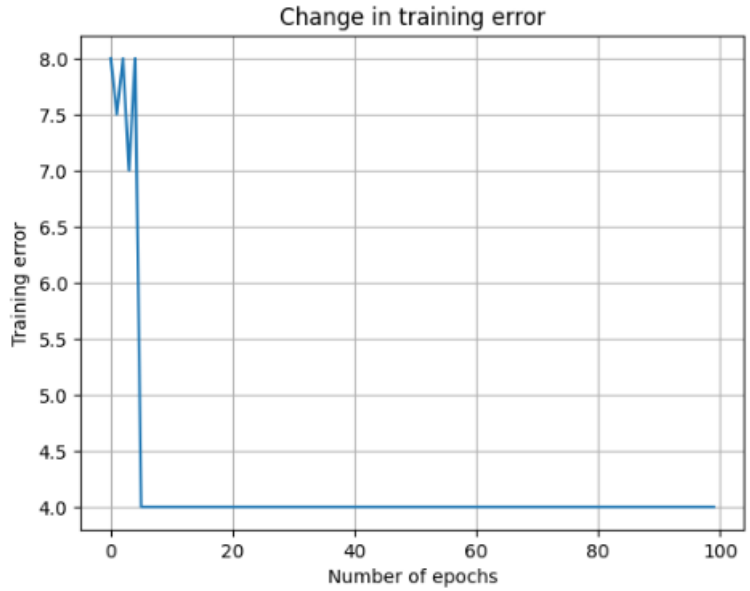


Рис.1.11 – Реакція програми на дію




```
Test results:
[0.4, 4.3] --> [0. 0.]
[4.4, 0.6] --> [1. 0.]
[4.7, 8.1] --> [1. 1.]
```

Рис.1.12 – Реакція програми на дію

Завдання 2.6

Дія1:

```python
import numpy as np
import matplotlib.pyplot as plt
!pip install neurolab
import neurolab as nl

# Генерація тренувальних даних
min_val = -15
max_val = 15
num_points = 130
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 5
y /= np.linalg.norm(y)

# Створення даних та міток
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
plt.show()

# Визначення багатошарової нейронної мережі з двома прихованими шарами
# Перший прихований шар складається із десяти нейронів.
# Другий прихований шар складається з шести нейронів.
# Вихідний шар складається з одного нейрона.
nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])

# Задання градієнтного спуску як навчального алгоритму
nn.trainf = nl.train.train_gd

# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)

# Виконання нейронної мережі на тренувальних даних
output = nn.sim(data)
y_pred = output.reshape(num_points)

# Побудова графіка помилки навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.show()
```

```python
# Побудова графіка результатів
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size, 1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, 'x', x, y_pred, 'p')
plt.title('Фактические и прогнозные значения')
plt.show()
```

Рис.1.13– Код програми

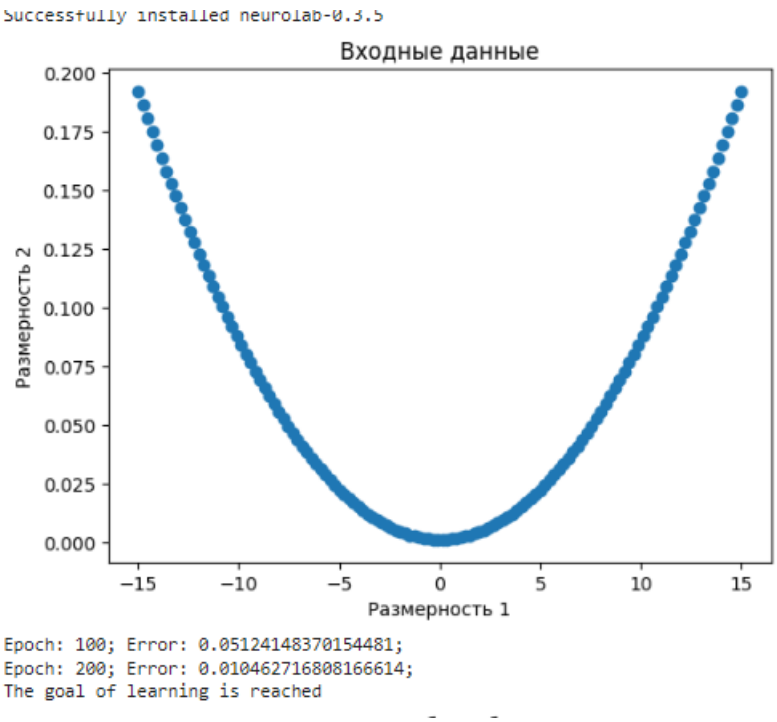


Рис.1.14 – Реакція програми на дію

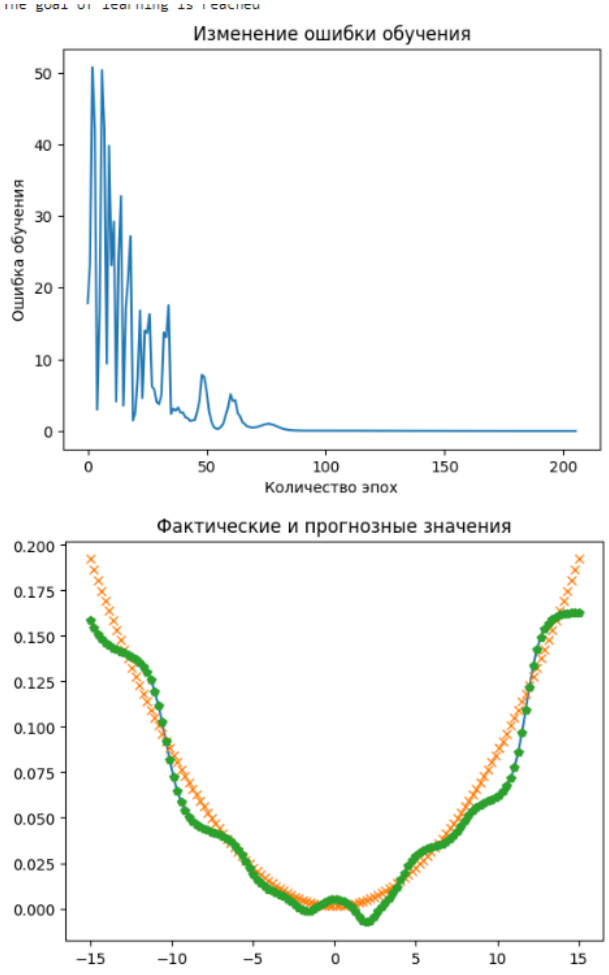


Рис.1.15 – Реакція програми на дію

Завдання 2.6

Дія1:

```
!pip install neurolab
import neurolab as nl

# Генерація тренувальних даних
min_val = -10
max_val = 10
num_points = 100
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 7

# Створення даних та міток
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')
plt.show()

# Визначення багатошарової нейронної мережі
nn = nl.net.newff([[min_val, max_val]], [5, 5, 5, 5, 1])

# Задання градієнтного спуску як навчального алгоритму
nn.trainf = nl.train.train_gd

# Тренування нейронної мережі
error_progress = nn.train(data, labels, epochs=1000, show=100, goal=0.01)

# Виконання нейронної мережі на тренувальних даних
output = nn.sim(data)
y_pred = output.reshape(num_points)

# Побудова графіка помилки навчання
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')
plt.show()

# Побудова графіка результатів
x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size, 1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, 'x', x, y_pred, 'p')
plt.title('Фактические и прогнозные значения')
plt.show()
```

Рис.1.16– Код програми

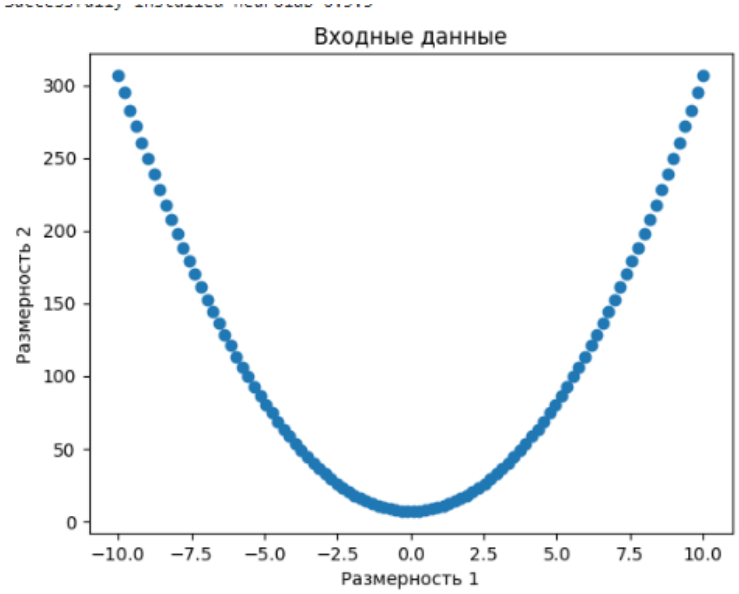
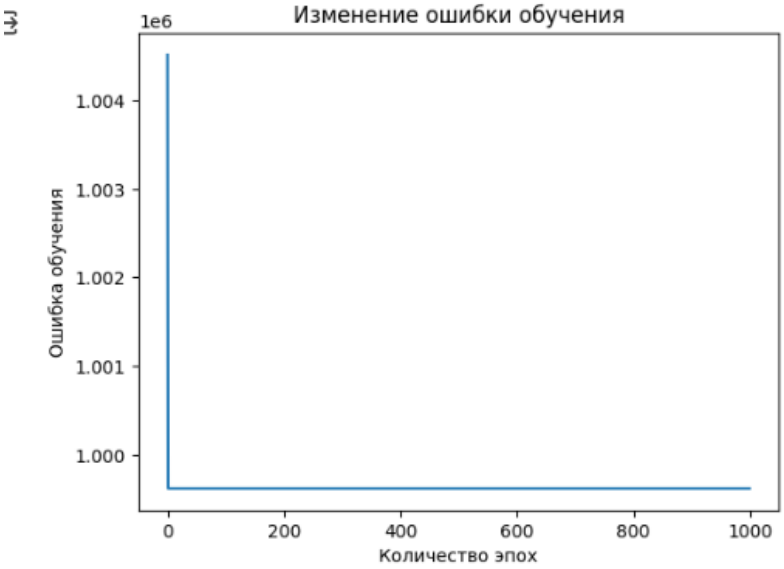


Рис.1.17 – Реакція програми на дію



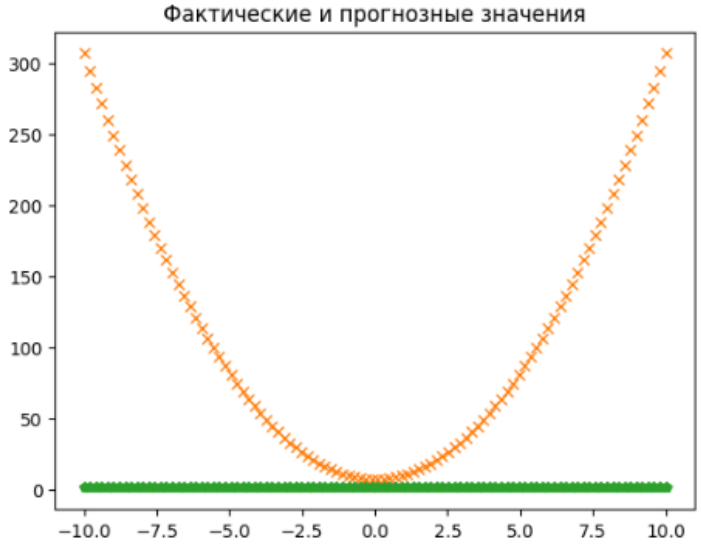


Рис.1.18 – Реакція програми на дію

Завдання 2.7

Дія1:

```
import numpy as np
!pip install neurolab
import neurolab as nl
import numpy.random as rand
skv = 0.05
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])
rand_norm = skv * rand.randn(100, 4, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 4, 2)
rand.shuffle(inp)
# Create net with 2 inputs and 4 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 4)
# train with rule: Conscience Winner Take All algoritm (CWTA)
error = net.train(inp, epochs=200,  show=20)
# Plot results:
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:,0], inp[:,1], '.', \
        centr[:,0], centr[:, 1] , 'yv', \
        w[:,0], w[:,1], 'p')
pl.legend(['train samples', 'real centers', 'train centers'])
pl.show()
```

Рис.1.19– Код програми

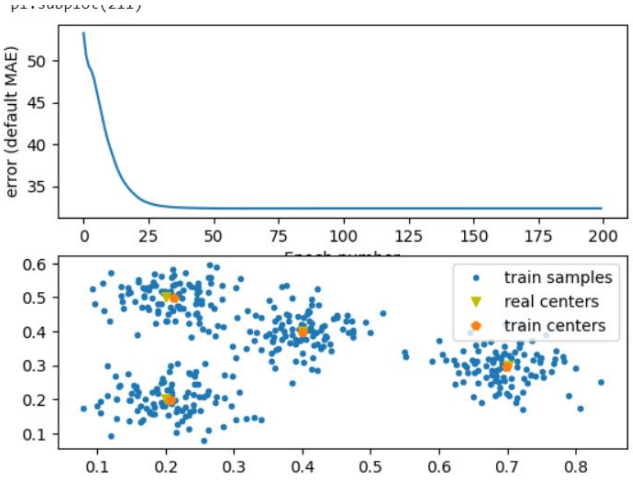


Рис.1.20= – Реакція програми на дію

Завдання 2.8

Дія1:

```
import numpy as np
!pip install neurolab
import neurolab as nl
import numpy.random as rand
import pylab as pl

# Змінені вхідні дані
skv = 0.04
centr = np.array([[0.2, 0.1], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5], [0.3, 0.5]])
rand_norm = skv * rand.randn(100, 5, 2)  # Змінено розмірність для відповідності новим центрам
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)
rand.shuffle(inp)

# Створення мережі з 2 входами та 5 нейронами
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 5)

# Тренування мережі
error = net.train(inp, epochs=200, show=20)

# Побудова графіків
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('Error (default MAE)')

w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:, 0], inp[:, 1], '.', centr[:, 0], centr[:, 1], 'yv', w[:, 0], w[:, 1], 'p')
pl.legend(['Train samples', 'Real centers', 'Train centers'])
pl.show()
```

Рис.1.21– Код програми

```
Epoch: 20; Error: 33.542425933969355;
Epoch: 40; Error: 30.949117310708587;
Epoch: 60; Error: 30.664425108522916;
Epoch: 80; Error: 30.599628314787857;
Epoch: 100; Error: 30.588566311607465;
Epoch: 120; Error: 30.58023208948063;
Epoch: 140; Error: 30.557190279261285;
Epoch: 160; Error: 30.537941728284068;
Epoch: 180; Error: 30.535460228665414;
Epoch: 200; Error: 30.534953699653904;
The maximum number of train epochs is reached
<ipython-input-1-2474acea04c4>:23: MatplotlibDeprecationWarning: Auto-removal of overlappi
  pl.subplot(211)
```

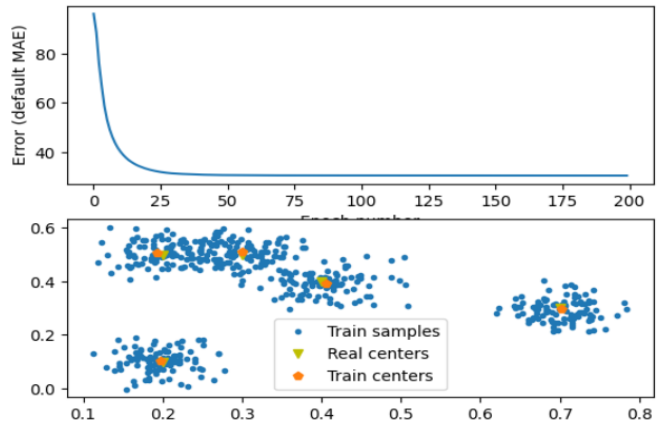


Рис.1.22 – Реакція програми на дію