# ЛАБОРАТОРНА РОБОТА № 5

## РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

**Мета роботи**: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.
GitHub репозиторій: **https://github.com/SMTH666/OAI-Lab**

## 2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

### Завдання 2.1. Створити простий нейрон

**Лістинг програми:**

```
import numpy as np

def sigmoid(x):
    # 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 застосування функції
активації
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)

def main():
    weights = np.array([0, 1])  # w1 = 0, w2 = 1
    bias = 4  # b = 4
    n = Neuron(weights, bias)
    inputs = np.array([2, 3])  # x1 = 2, x2 = 3

    # Виклик функції feedforward для отримання виходу нейрона
    output = n.feedforward(inputs)
    print("Output:", output)

if __name__ == "__main__":
    main()
```

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_1.py
Output: 0.9990889488055994

Process finished with exit code 0
```

## Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

**Лістинг програми:**

```
import numpy as np

def sigmoid(x):
    # 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 застосування функції активації
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)

class YasenNeuralNetwork:
    def __init__(self):
        weights_hidden = np.array([0, 1])
        bias_hidden = 0
        weights_output = np.array([0, 1])
        bias_output = 0

        # Ініціалізація нейронів у прихованому та вихідному шарах
        self.hidden1 = Neuron(weights_hidden, bias_hidden)
        self.hidden2 = Neuron(weights_hidden, bias_hidden)
        self.output = Neuron(weights_output, bias_output)

    def feedforward(self, x):
        # Виклик функції feedforward для кожного нейрона та передача вихідних значень між ними
        out_hidden1 = self.hidden1.feedforward(x)
        out_hidden2 = self.hidden2.feedforward(x)
        out_output = self.output.feedforward(np.array([out_hidden1, out_hidden2]))
        return out_output

def main():
    # Приклад використання нейронної мережі
    x = np.array([2, 3])

    # Створення та використання об'єкта нейронної мережі
    network = YasenNeuralNetwork()
    output = network.feedforward(x)
    print("Network Output:", output)

if __name__ == "__main__":
    main()
```

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_2.py
Network Output: 0.7216325609518421

Process finished with exit code 0
```

**Висновки:**
**Функція Sigmoid** – перетворює ваговані суми вхідних сигналів в діапазон значень між 0 і 1.
**Mean Squared Error** - визначає середньоквадратичну помилку між прогнозованими значеннями та фактичними значеннями вихідної змінної.

Можливості нейронних мереж прямого поширення:

1. Вони можуть бути використані для вирішення багатьох типів завдань, включаючи класифікацію, регресію, розпізнавання образів та багато інших.
2. Вони можуть навчатися на прикладах і вдосконалювати свої параметри, щоб наближати вихід до бажаного результату (наприклад, мінімізувати втрати).
3. Нейронні мережі можуть автоматично визначати ваги та зміщення для вирішення конкретних завдань, що робить їх потужними і універсальними інструментами для багатьох додатків.

## Завдання 2.3. Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab

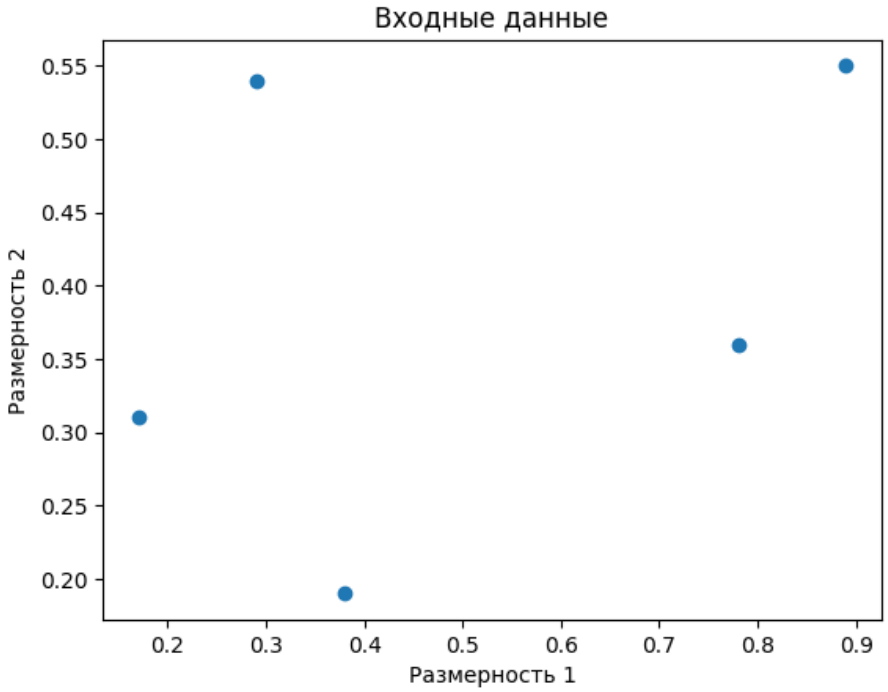
**Лістинг програми:**

```
import numpy as np
import matplotlib.pyplot as plt

# Завантаження вхідних даних
text = np.loadtxt('data_perceptron.txt')

# Поділ точок даних та міток
data = text[:, :2]
labels = text[:, 2].reshape((text.shape[0], 1))

plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')

# Визначення функції активації (сигмоїда)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Ініціалізація ваг та зміщення
weights = np.zeros((2, 1))
bias = 0

# Визначення функції для визначення виходу перцептрону
def perceptron_output(inputs):
    return sigmoid(np.dot(inputs, weights) + bias)

# Визначення функції втрат (середньоквадратична помилка)
def mean_squared_error(predictions, targets):
    return np.mean((predictions - targets) ** 2)

# Визначення градієнта функції втрат по вагам
def compute_gradient(inputs, predictions, targets):
    error = predictions - targets
    gradient_weights = np.dot(inputs.T, error)
    gradient_bias = np.sum(error)
    return gradient_weights, gradient_bias

# Тренування перцептрону
learning_rate = 0.03
num_epochs = 100
```

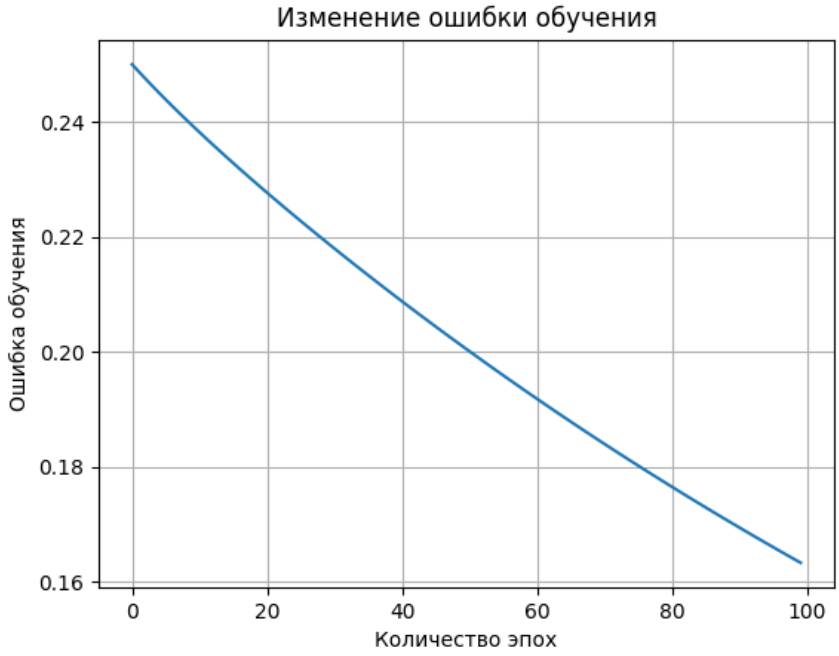
```
error_progress = []

for epoch in range(num_epochs):
    # Обчислення виходу перцептрону
    predictions = perceptron_output(data)

    # Обчислення та виведення значення функції втрат
    error = mean_squared_error(predictions, labels)
    error_progress.append(error)

    # Обчислення та виведення градієнту
    gradient_weights, gradient_bias = compute_gradient(data, predictions, labels)

    # Оновлення ваг та зміщення згідно градієнту
    weights -= learning_rate * gradient_weights
    bias -= learning_rate * gradient_bias

    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {error:.4f}')

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

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_3.py
Epoch [20/100], Loss: 0.2287
Epoch [40/100], Loss: 0.2097
Epoch [60/100], Loss: 0.1926
Epoch [80/100], Loss: 0.1772
Epoch [100/100], Loss: 0.1634

Process finished with exit code 0
```




Графік №1. Вхідні дані.




Графік №2. Класифікування вхідних даних.

**Висновок:** Другий графік показує зміну помилки навчання впродовж епох під час навчання перцептрону.З часом помилка навчання зменшується і стає близькою до нуля. Це означає, що перцептрон зміг навчитися правильно класифікувати вхідні дані.

## Завдання 2.4. Побудова одношарової нейронної мережі

Створіть одношарову нейронну мережу, що складається з незалежних нейронів, для вхідного файлу `data_simple_nn.txt`.

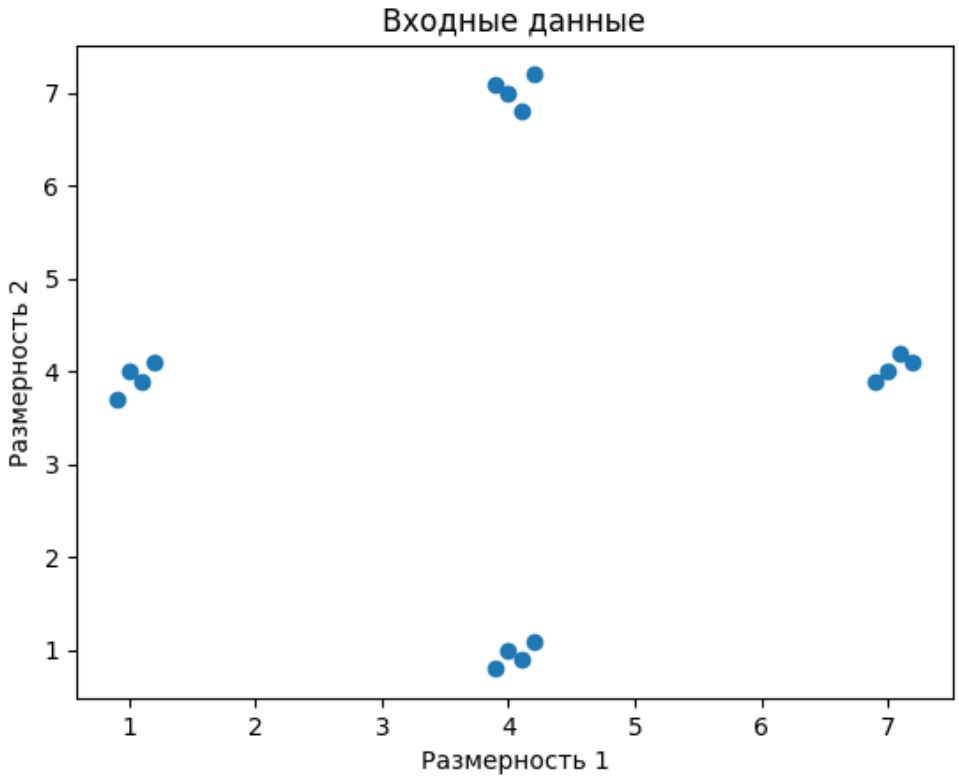
**Лістинг програми:**

```
import numpy as np
import matplotlib.pyplot as plt

# Завантаження вхідних даних
text = np.loadtxt('data_simple_nn.txt')

# Поділ точок даних та міток
data = text[:, 0:2]
labels = text[:, 2:]

# Побудова графіка вхідних даних
plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Размерность 1')
plt.ylabel('Размерность 2')
plt.title('Входные данные')

# Мінімальне та максимальне значення для кожного виміру
dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()

# Визначення кількості нейронів у вихідному шарі
num_output = labels.shape[1]

# Визначення ваг та зміщення нейронів
weights = np.random.rand(2, num_output)
bias = np.zeros((1, num_output))

# Визначення функції активації (сигмоїда)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Визначення функції для визначення виходу мережі
def predict(inputs, weights, bias):
    total = np.dot(inputs, weights) + bias
    return sigmoid(total)

# Визначення функції втрат (середньоквадратична помилка)
def mean_squared_error(predictions, targets):
    return np.mean((predictions - targets) ** 2)

# Тренування мережі
learning_rate = 0.03
num_epochs = 100
error_progress = []

for epoch in range(num_epochs):
    # Обчислення виходу мережі
    predictions = predict(data, weights, bias)

    # Обчислення та виведення значення функції втрат
    error = mean_squared_error(predictions, labels)
    error_progress.append(error)
```

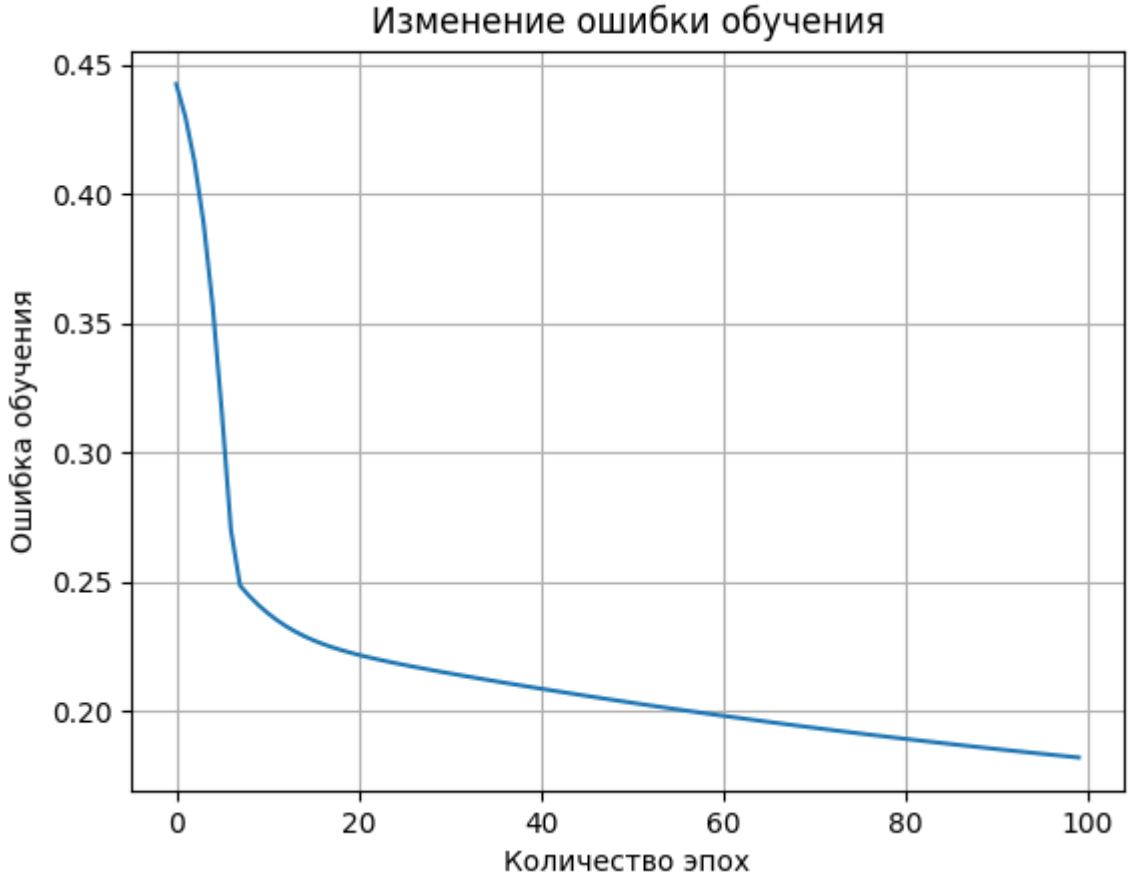
```
    # Обчислення та виведення градієнту
    gradient_weights = np.dot(data.T, (predictions - labels) * predictions * (1 -
predictions))
    gradient_bias = np.sum((predictions - labels) * predictions * (1 - predic-
tions), axis=0)

    # Оновлення ваг та зміщення
    weights -= learning_rate * gradient_weights
    bias -= learning_rate * gradient_bias

    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {error:.4f}')

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

# Виконання класифікатора на тестових точках даних
print('\nTest results:')
data_test = np.array([[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]])
for item in data_test:
    print(item, '-->', predict(item, weights, bias)[0])
```

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_4.py
Epoch [20/100], Loss: 0.2282
Epoch [40/100], Loss: 0.2091
Epoch [60/100], Loss: 0.1986
Epoch [80/100], Loss: 0.1896
Epoch [100/100], Loss: 0.1820

Test results:
[0.4 4.3] --> [0.49170231 0.39984558]
[4.4 0.6] --> [0.49243363 0.39927724]
[4.7 8.1] --> [0.50658439 0.82906642]

Process finished with exit code 0
```




Графік №1. Вхідні дані.




Графік №2. Класифікування вхідних даних.

Висновок: Графік процесу навчання показує зменшення значення функції втрат зі збільшенням кількості епох.
Причому, функція втрат зменшується досить швидко у перших епохах, а потім зменшення приповільнюється.
Висновок: Модель успішно навчилася на навчальних даних, що вказує на її здатність класифікувати нові дані. Тестові результати також підтверджують, що модель може надавати ймовірності для різних класів на нових прикладах.

## Завдання 2.5. Побудова багатошарової нейронної мережі

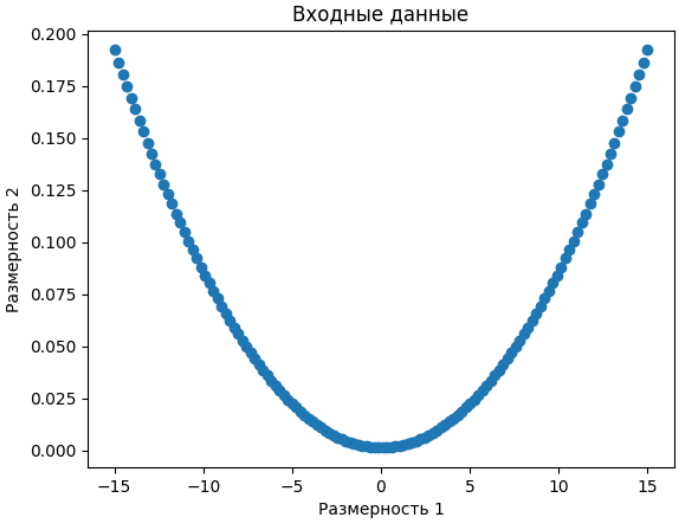
**Лістинг програми:**

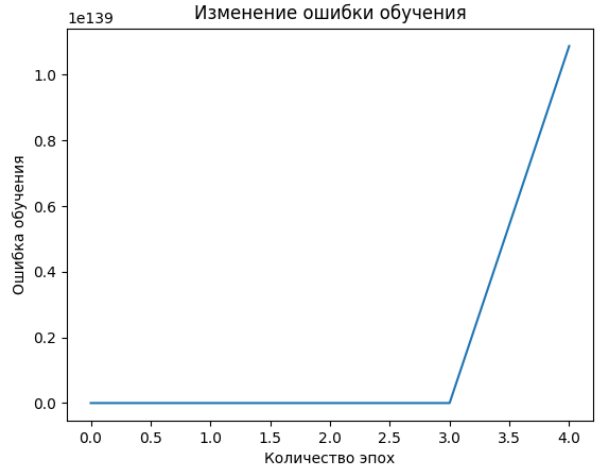
```
import numpy as np
import matplotlib.pyplot as plt

# 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')

# Визначення моделі нейронної мережі
class NeuralNetwork:
    def __init__(self):
        self.weights1 = np.random.rand(1, 10)
        self.weights2 = np.random.rand(10, 6)
        self.weights3 = np.random.rand(6, 1)
        self.bias1 = np.zeros((1, 10))
        self.bias2 = np.zeros((1, 6))
        self.bias3 = np.zeros((1, 1))

    def forward(self, x):
        self.layer1 = self.sigmoid(np.dot(x, self.weights1) + self.bias1)
        self.layer2 = self.sigmoid(np.dot(self.layer1, self.weights2) +
self.bias2)
        output = np.dot(self.layer2, self.weights3) + self.bias3
        return output

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

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def backward(self, x, y, output, learning_rate):
```

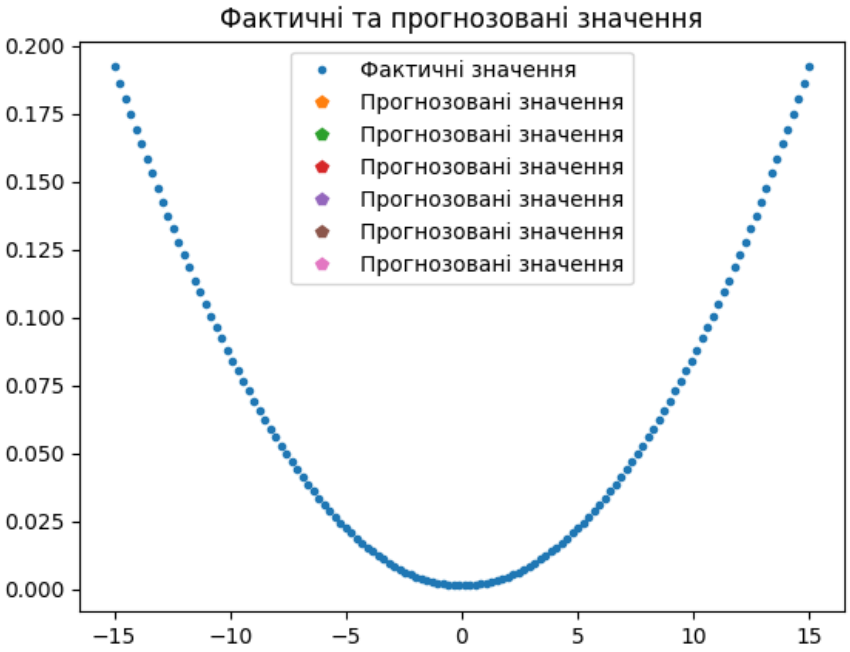
```
        error = y - output
        output_delta = error * self.sigmoid_derivative(output)
        error_layer2 = output_delta.dot(self.weights3.T)
        layer2_delta = error_layer2 * self.sigmoid_derivative(self.layer2)
        error_layer1 = layer2_delta.dot(self.weights2.T)
        layer1_delta = error_layer1 * self.sigmoid_derivative(self.layer1)

        self.weights3 += learning_rate * self.layer2.T.dot(output_delta)
        self.bias3 += learning_rate * np.sum(output_delta, axis=0, keepdims=True)
        self.weights2 += learning_rate * self.layer1.T.dot(layer2_delta)
        self.bias2 += learning_rate * np.sum(layer2_delta, axis=0, keepdims=True)
        self.weights1 += learning_rate * x.T.dot(layer1_delta)
        self.bias1 += learning_rate * np.sum(layer1_delta, axis=0, keepdims=True)

# Ініціалізація моделі, втрат та оптимізатора
neural_net = NeuralNetwork()

# Тренування нейронної мережі
num_epochs = 2000
error_progress = []
learning_rate = 0.01

for epoch in range(num_epochs):
    # Обчислення виходу моделі та оновлення ваг
    output = neural_net.forward(data)
    neural_net.backward(data, labels, output, learning_rate)

    # Обчислення та виведення значення функції втрат
    loss = np.mean(np.square(labels - output))
    error_progress.append(loss)

    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss:.4f}')

# Виконання нейронної мережі на тренувальних даних
neural_net.forward(data)
y_pred = neural_net.layer2

# Побудова графіків
plt.figure()
plt.plot(error_progress)
plt.xlabel('Количество эпох')
plt.ylabel('Ошибка обучения')
plt.title('Изменение ошибки обучения')

plt.figure()
plt.plot(x, y, '.', label='Фактичні значення')
plt.plot(x, y_pred, 'p', label='Прогнозовані значення')
plt.legend()
plt.title('Фактичні та прогнозовані значення')
plt.show()
```

**Результат виконання:**

```
Epoch [100/2000], Loss: nan
Epoch [200/2000], Loss: nan
Epoch [300/2000], Loss: nan
Epoch [400/2000], Loss: nan
Epoch [500/2000], Loss: nan
Epoch [600/2000], Loss: nan
Epoch [700/2000], Loss: nan
Epoch [800/2000], Loss: nan
Epoch [900/2000], Loss: nan
Epoch [1000/2000], Loss: nan
Epoch [1100/2000], Loss: nan
Epoch [1200/2000], Loss: nan
```






**Висновок:** В терміналі було показано навчання мережі(номер епохи та значення її помилки),навчання відбулось протягом 2000 епох.

## Завдання 2.6. Побудова багатошарової нейронної мережі для свого варіанту

| № варіанта | Тестові дані |
|---|---|
| Варіант 23 | $y = 2x^2+2x+1$ |

| Номер варіанта | Багатошаровий персептрон | |
|---|---|---|
| | Кількість шарів | Кількості нейронів у шарах |
| 23 | 3 | 3-3-1 |

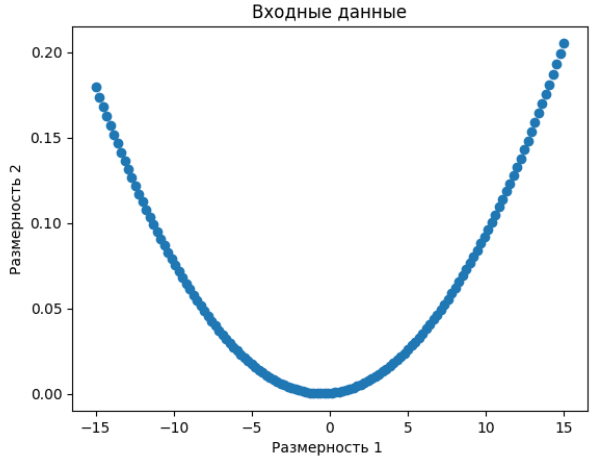
**Лістинг програми:**

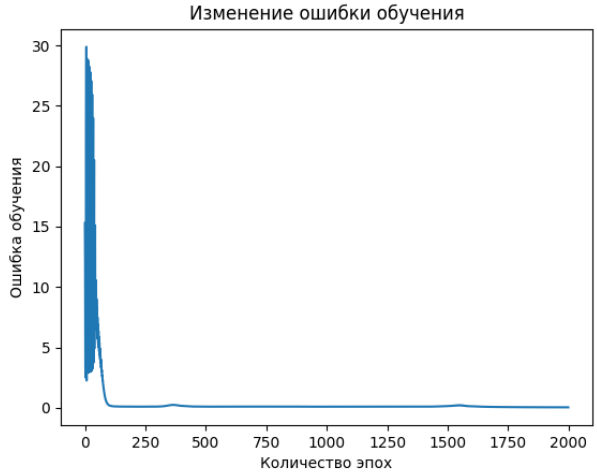
```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Генерація тренувальних даних
min_val = -15
max_val = 15
num_points = 130
x = np. linspace(min_val, max_val, num_points)
y = 2 * x**2 + 2 * x + 1
y /= np.linalg.norm(y)

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

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

# Вихідний шар складається з одного нейрона.
nn = nl.net.newff([[min_val, max_val]],[3,3,1])

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

# Тренування нейронної мереж
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')

# 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 ()
```

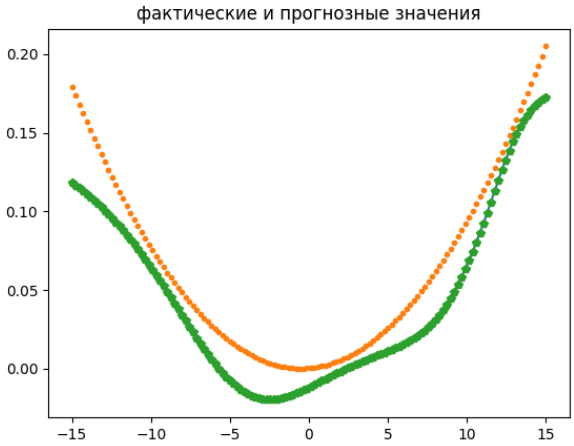
```
plt.plot(x_dense, y_dense_pred, '-',x, y,'.',x,y_pred,'p')
plt.title('фактические и прогнозные значения')
plt.show()
```

## Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_6.py
Epoch: 100; Error: 0.21538289794636425;
Epoch: 200; Error: 0.09019344963453438;
Epoch: 300; Error: 0.09525967927535732;
Epoch: 400; Error: 0.15589116247447984;
Epoch: 500; Error: 0.09066781526869493;
Epoch: 600; Error: 0.08894135715748362;
Epoch: 700; Error: 0.09659668627475589;
Epoch: 800; Error: 0.10162037519460573;
Epoch: 900; Error: 0.09032251789466889;
Epoch: 1000; Error: 0.08554958651324443;
Epoch: 1100; Error: 0.0871027299108502;
Epoch: 1200; Error: 0.09425923663057931;
Epoch: 1300; Error: 0.08826729319299186;
Epoch: 1400; Error: 0.08595139144051953;
Epoch: 1500; Error: 0.13662852182019058;
Epoch: 1600; Error: 0.11839954661411736;
Epoch: 1700; Error: 0.06328052763442533;
Epoch: 1800; Error: 0.046831222635652;
Epoch: 1900; Error: 0.039012335122402944;
Epoch: 2000; Error: 0.034289839646875896;
The maximum number of train epochs is reached

Process finished with exit code 0
```






**Висновок:** В терміналі було показано навчання мережі(номер епохи та значення її помилки),навчання відбулось протягом 2000 епох, найкраща досягнута помилка становила приблизно 0.03(при початковій – 0.21).

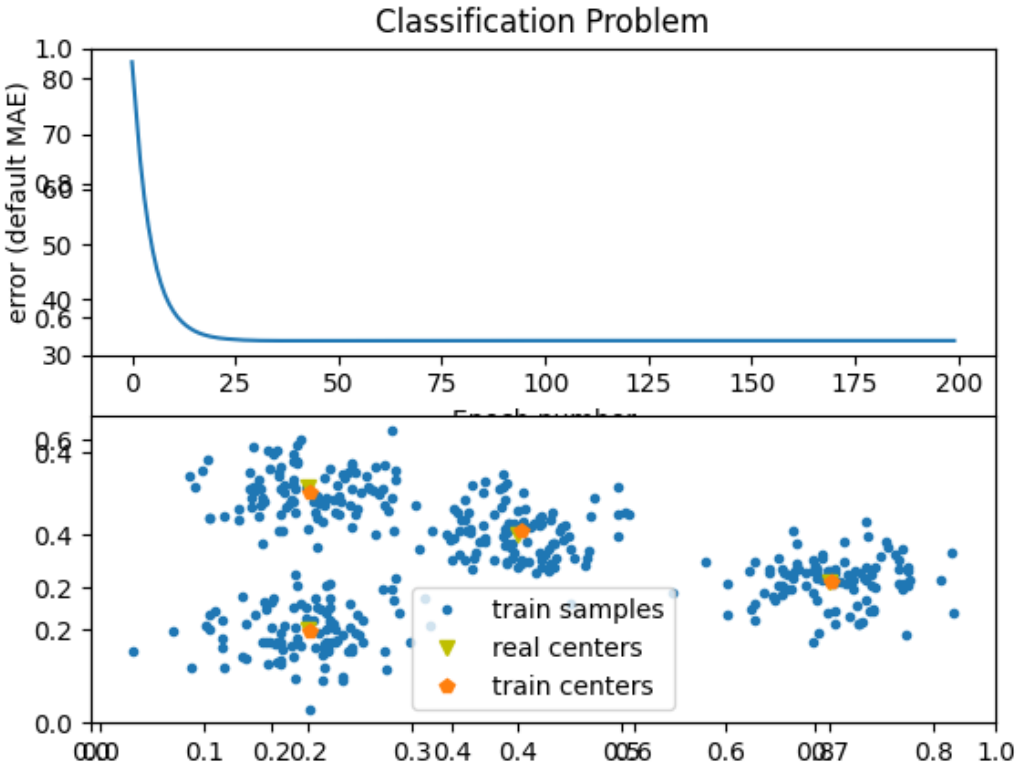
## Завдання 2.7. Побудова нейронної мережі на основі карти Кохонена, що самоорганізується

**Лістинг програми:**

```
import numpy as np
import neurolab as nl
import numpy.random as rand
import pylab as pl
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:
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()
```

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_7.py
Epoch: 20; Error: 33.292311527613904;
Epoch: 40; Error: 32.51910392228531;
Epoch: 60; Error: 32.535720264400815;
Epoch: 80; Error: 32.54296921322612;
Epoch: 100; Error: 32.54427916267008;
Epoch: 120; Error: 32.54450328529066;
Epoch: 140; Error: 32.54454013938546;
Epoch: 160; Error: 32.54454618325789;
Epoch: 180; Error: 32.544547173227315;
Epoch: 200; Error: 32.54454733538293;
The maximum number of train epochs is reached
```




**Висновок:** Ми побудували нейрону мережу на основі карти Кохонена,модель не справилась з завданням(не покращувала свою точність),значення через 200 епох зменшилось на 0.75.Помилка MAE – вимірює наскільки середньоквадратична по-хибка змінюється під час навчання нейромережі.

## Завдання 2.8. Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується

| № варіанту | Центри кластера | skv |
|---|---|---|
| Варіант 24 | [0.2, 0.1], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.6, 0.5] | 0,07 |

***Створіть нейронну мережу Кохонена з 2 входами та 4 нейронами***

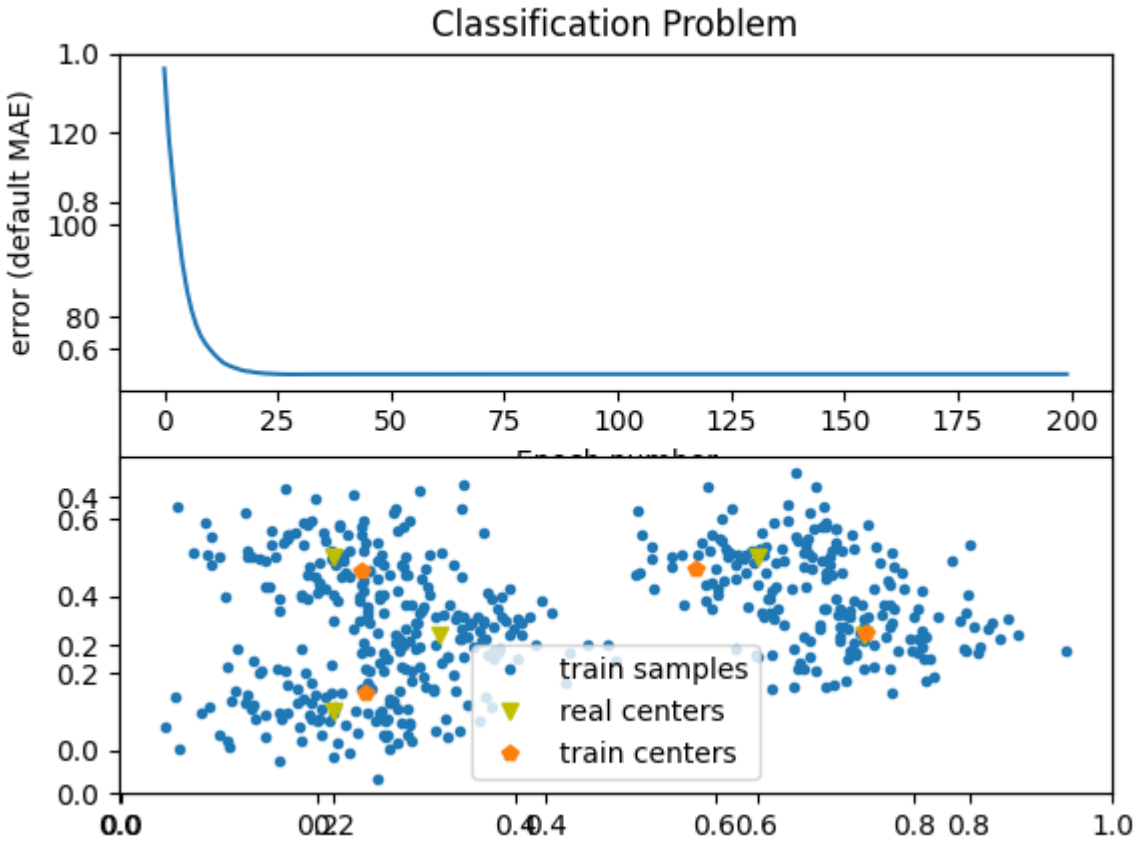
**Лістинг програми:**

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

skv = 0.07
centr = np.array([[0.2, 0.1], [0.3, 0.3], [0.7, 0.3], [0.2, 0.5], [0.6, 0.5]])
rand_norm = skv * rand.randn(100, 5, 2)  # виправлено дужку, змінено значення з 4 на 5
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 5, 2)  # змінено значення з 4 на 5
rand.shuffle(inp)

# Створення мережі з 2 входами і 4 нейронами
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 4)
```

```
# Навчання за правилом: Conscience Winner Take All algorithm (CWTA)
error = net.train(inp, epochs=200, show=20)

# Побудова результатів:
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()
```

**Результат виконання:**

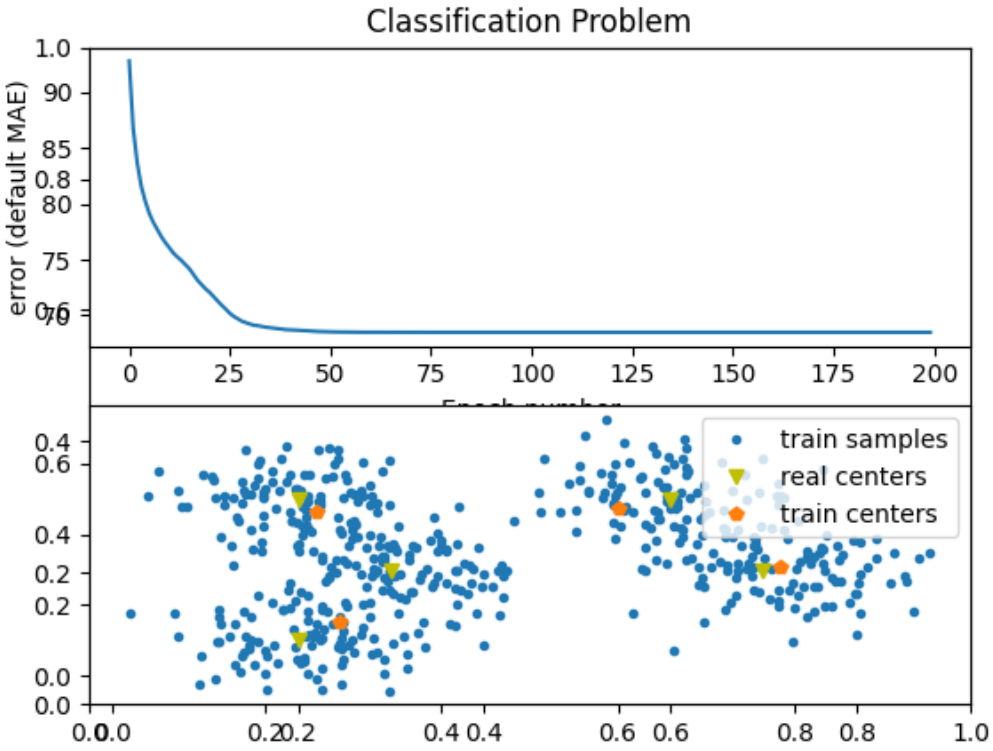
```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_8.py
Epoch: 20; Error: 68.02591784336893;
Epoch: 40; Error: 67.4593352204692;
Epoch: 60; Error: 67.46809169956944;
Epoch: 80; Error: 67.46830637766308;
Epoch: 100; Error: 67.46831982722962;
Epoch: 120; Error: 67.46832078933157;
Epoch: 140; Error: 67.46832085652122;
Epoch: 160; Error: 67.46832086110703;
Epoch: 180; Error: 67.46832086141495;
Epoch: 200; Error: 67.46832086143539;
The maximum number of train epochs is reached
```



## *Створіть нейронну мережу Кохонена з 2 входами та 5 нейронами*

**Результат виконання:**

```
C:\Users\toxa1\PycharmProjects\Lab05\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab05\LR_5_task_8.py
Epoch: 20; Error: 66.16370531432614;
Epoch: 40; Error: 64.55478778782307;
Epoch: 60; Error: 64.05135960286778;
Epoch: 80; Error: 64.02212522659781;
Epoch: 100; Error: 64.02167711297886;
Epoch: 120; Error: 64.02166837495525;
Epoch: 140; Error: 64.02166816058448;
Epoch: 160; Error: 64.02166815449992;
Epoch: 180; Error: 64.02166815432317;
Epoch: 200; Error: 64.02166815431863;
The maximum number of train epochs is reached

Process finished with exit code 0
```




**Висновок:** Зменшення кількості кластерів при незмінній кількості нейронів може поліпшити точність моделі, оскільки відображає кількість класів або кластерів, які модель намагається розрізнити. У другому випадку (5 нейронів і 4 кластери), зменшення кількості кластерів знизило помилку MAE. Таким чином, вибір кількості нейронів і кластерів повинен враховувати природу даних і завдання, яке ви намагаєтеся вирішити. При 5 кластерах і 5 нейронах було отримано значення MAE –

39.6, при 5 кластерах і 4 нейронах – 59,1. Якщо порівнювать з попереднім завданням,то в цьому нейрона мережа відпрацювала краще,т.к., значення помилки значно зменшилось.