**Лабораторна робота №6**

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

***Мета роботи:*** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовуватипрості нейронні мережі***.***

**Завдання на лабораторну роботу**

Завдання 2.1. Створити простий нейрон

Лістинг програми

*import* numpy *as* np

def sigmoid(x):

*return* 1 / (1 + np.exp(-x))

class Neuron:

    def \_\_init\_\_(self, weights, bias):

*self*.weights = weights

*self*.bias = bias

    def feedforward(self, inputs):

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

*return* sigmoid(total)

*if* \_\_name\_\_ == "\_\_main\_\_":

    weights = np.array([0, 1])

    bias = 4

    n = Neuron(weights, bias)

    x = np.array([2, 3])

    print(n.feedforward(x))

Результат виконання



Завдання 2.2. Створити просту нейронну мережу для передбачення статі людини

Лістинг програми

*import* numpy *as* np

*from* lab6\_task1 *import* Neuron, sigmoid

def deriv\_sigmoid(x):

    fx = sigmoid(x)

*return* fx \* (1 - fx)

def mse\_loss(y\_true, y\_pred):

*return* ((y\_true - y\_pred) \*\* 2).mean()

class MischenchukNeuralNetwork:

    def \_\_init\_\_(self):

*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()

*self*.b1 = np.random.normal()

*self*.b2 = np.random.normal()

*self*.b3 = np.random.normal()

    def feedforward(self, x):

        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

*for* epoch *in* range(epochs):

*for* x, y\_true *in* zip(data, all\_y\_trues):

                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*

                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

*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))

*if* \_\_name\_\_ == "\_\_main\_\_":

    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*

    ])

    network = MischenchukNeuralNetwork()

    network.train(data, all\_y\_trues)

*# Робимо передбачення*

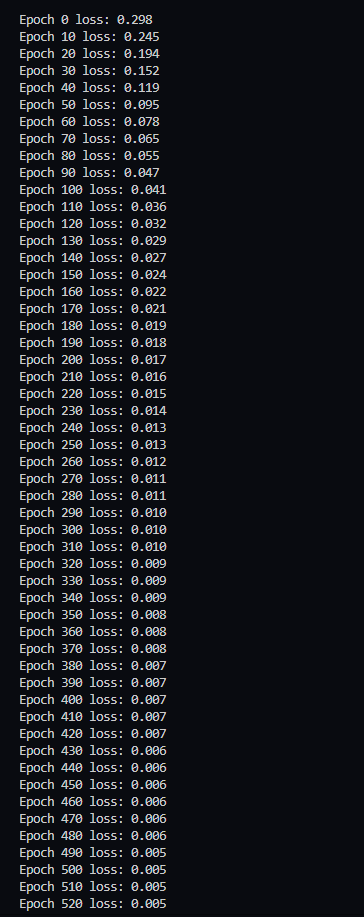
    emily = np.array([-7, -3])  *# 128 фунтов, 63 дюйма*

    frank = np.array([20, 2])  *# 155 фунтов, 68 дюймів*

    print("Emily: %.3f" % network.feedforward(emily))  *# +-0.966 - F*

    print("Frank: %.3f" % network.feedforward(frank))  *# +-0.038 - M*

Результат виконання



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

Лістинг програми

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* neurolab *as* nl

text = np.loadtxt('data\_perceptron.txt')

data = text[:, :2]

labels = text[:, 2].reshape((text.shape[0], 1))

plt.figure()

plt.scatter(data[:, 0], data[:, 1])

plt.xlabel('Розмірність 1')

plt.ylabel('Розмірність 2')

plt.title('Вхідні дані')

plt.show()

dim1\_min, dim1\_max, dim2\_min, dim2\_max = 0, 1, 0, 1

num\_output = labels.shape[1]

dim1 = [dim1\_min, dim1\_max]

dim2 = [dim2\_min, dim2\_max]

perceptron = nl.net.newp([dim1, dim2], num\_output)

error\_progress = perceptron.train(data, labels, epochs=100, show=20, lr=0.03)

plt.figure()

plt.plot(error\_progress)

plt.xlabel('Кількість епох')

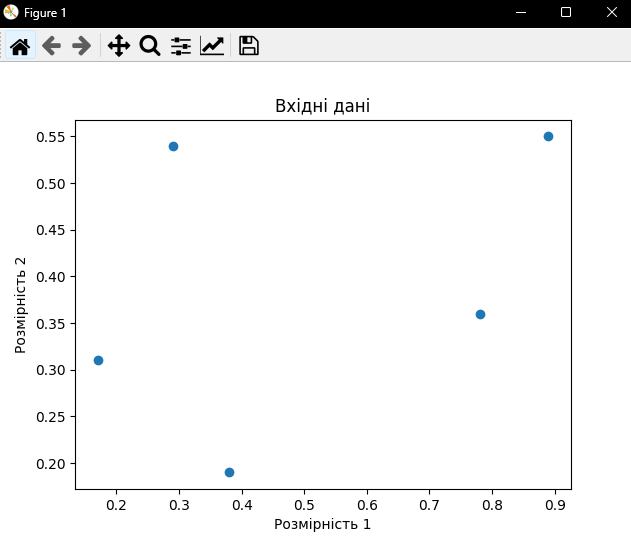
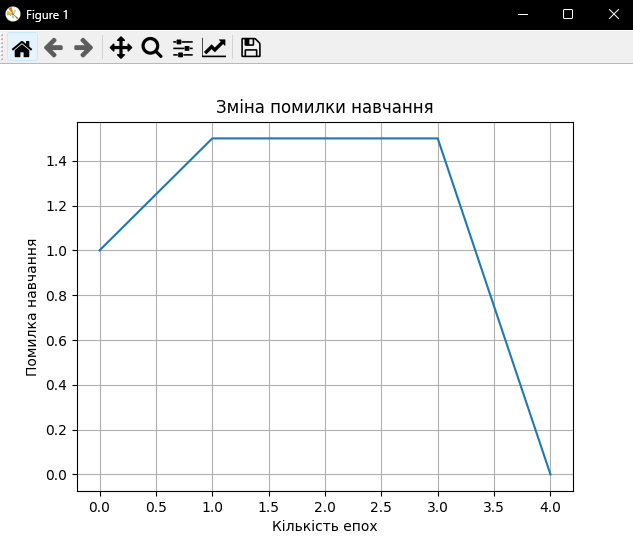
plt.ylabel('Помилка навчання')

plt.title('Зміна помилки навчання')

plt.grid()

plt.show()

Результат виконання

**Завдання 2.4. Побудова одношарової нейронної мережі**

Лістинг програми

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* neurolab *as* nl

text = np.loadtxt('data\_simple\_nn.txt')

data = text[:, 0:2]

labels = text[:, 2:]

plt.figure()

plt.scatter(data[:, 0], data[:, 1])

plt.xlabel('Розмірність 1')

plt.ylabel('Розмірність 2')

plt.title('Вхідні дані')

plt.show()

*# 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()

num\_output = labels.shape[1]

dim1 = [dim1\_min, dim1\_max]

dim2 = [dim2\_min, dim2\_max]

nn = nl.net.newp([dim1, dim2], num\_output)

error\_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

*# Побудова графіка просування процесу навчання*

plt.figure()

plt.plot(error\_progress)

plt.xlabel('Кількість епох')

plt.ylabel('Помилка навчання')

plt.title('Прогрес помилки навчання')

plt.grid()

plt.show()

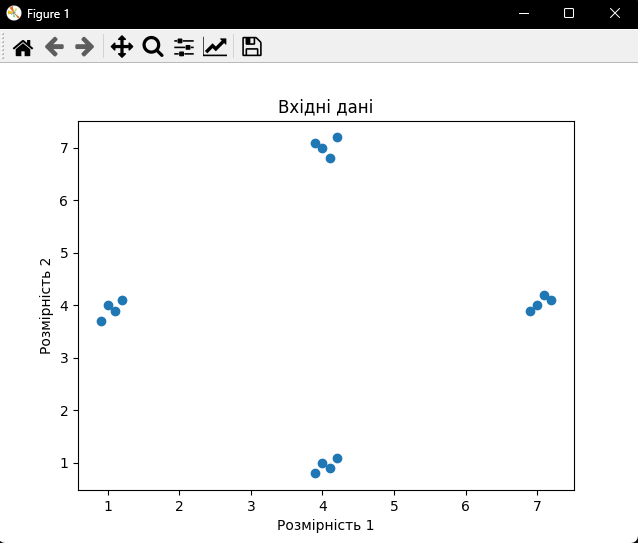
print('\nTest results:')

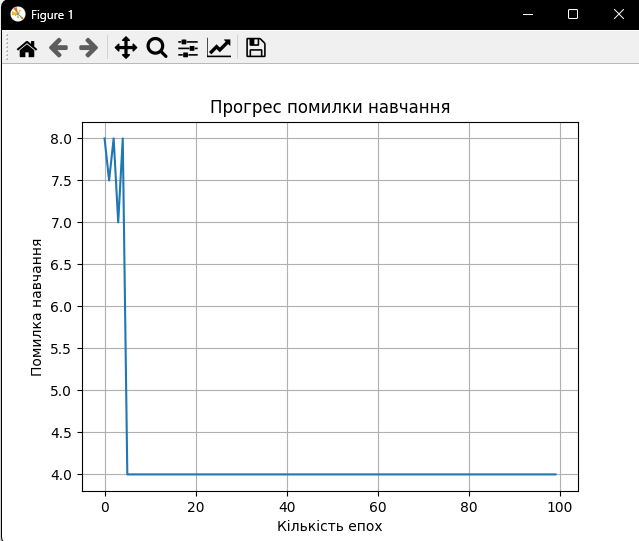
data\_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]

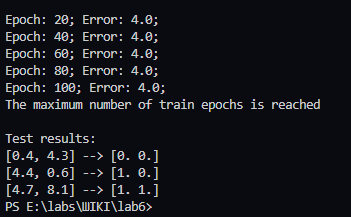
*for* item *in* data\_test:

    print(item, '-->', nn.sim([item])[0])

Результат виконання







**Завдання 2.5. Побудова багатошарової нейронної мережі**

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 = 3 \* np.square(x) + 5

y /= np.linalg.norm(y)

data = x.reshape(num\_points, 1)

labels = y.reshape(num\_points, 1)

plt.figure()

plt.scatter(data, labels)

plt.xlabel('Розмірність 1')

plt.ylabel('Розмірність 2')

plt.title('Вхідні дані')

nn = nl.net.newff([[min\_val, max\_val]], [10, 6, 1])

nn.trainf = nl.train.train\_gd

error\_progress = nn.train(data, labels, epochs=2000, show=100, goal=0.01)

output = nn.sim(data)

y\_pred = output.reshape(num\_points)

plt.figure()

plt.plot(error\_progress)

plt.xlabel('Кількість епох')

plt.ylabel('Помилка навчання')

plt.title('Прогрес помилки навчання')

plt.grid()

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()

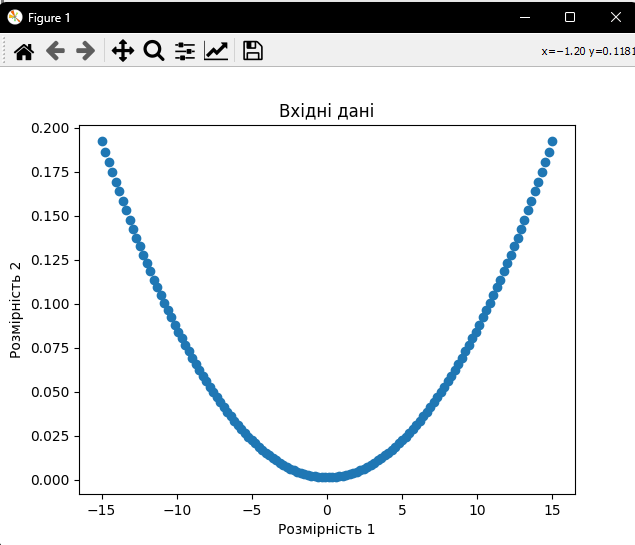
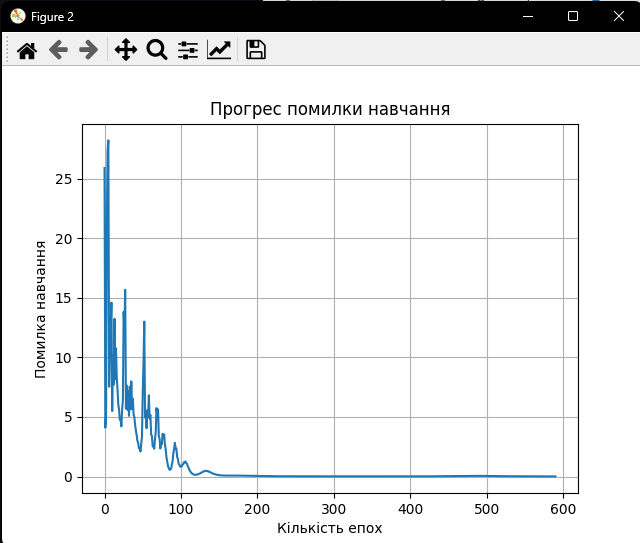
plt.plot(x\_dense, y\_dense\_pred, '-', x, y, '.', x, y\_pred, 'p')

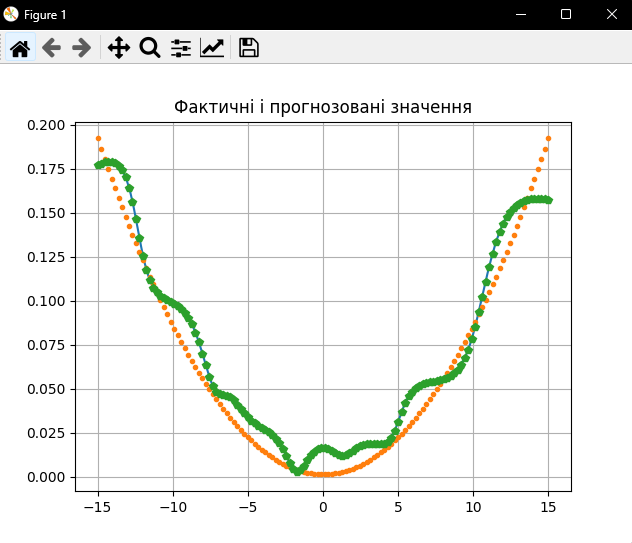
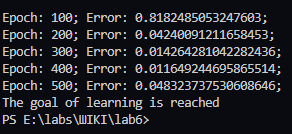
plt.title('Фактичні і прогнозовані значення')

plt.grid()

plt.show()

Результат виконання

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

Лістинг програми

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* neurolab *as* nl

min\_val = -15

max\_val = 15

num\_points = 130

x = np.linspace(min\_val, max\_val, num\_points)

y = 2 \* np.square(x) + 8

y /= np.linalg.norm(y)

data = x.reshape(num\_points, 1)

labels = y.reshape(num\_points, 1)

plt.figure()

plt.scatter(data, labels)

plt.xlabel('Розмірність 1')

plt.ylabel('Розмірність 2')

plt.title('Вхідні дані')

nn = nl.net.newff([[min\_val, max\_val]], [5, 1])

nn.trainf = nl.train.train\_gd

*# Тренування нейронної мережі*

error\_progress = nn.train(data, labels, epochs=20000, show=1000, goal=0.01)

output = nn.sim(data)

y\_pred = output.reshape(num\_points)

plt.figure()

plt.plot(error\_progress)

plt.xlabel('Кількість епох')

plt.ylabel('Помилка навчання')

plt.title('Прогрес помилки навчання')

plt.grid()

plt.show()

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()

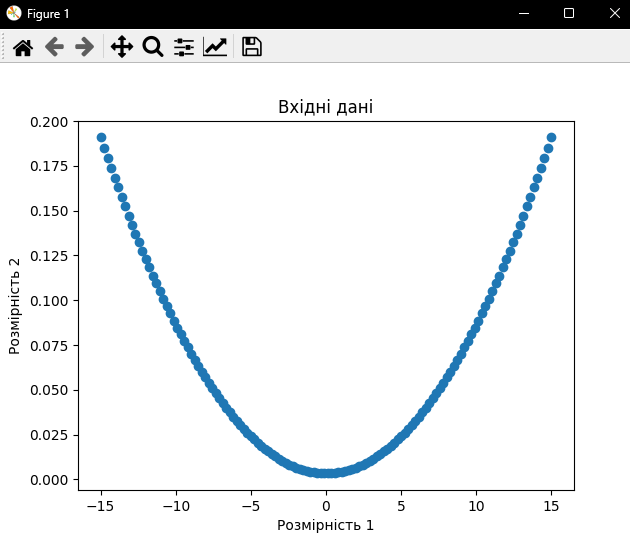
plt.plot(x\_dense, y\_dense\_pred, '-', x, y, '.', x, y\_pred, 'p')

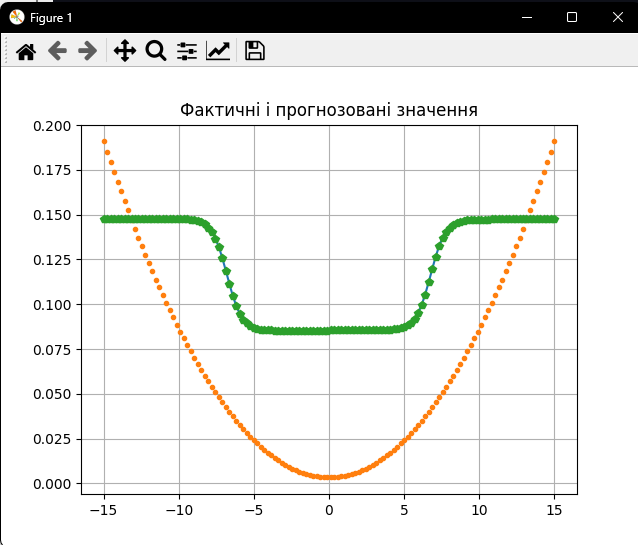
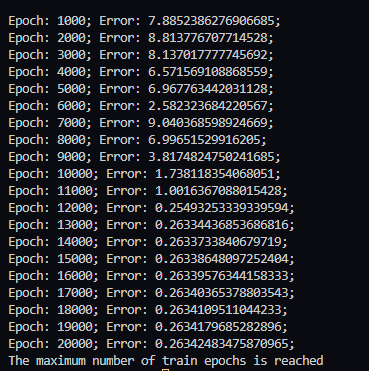
plt.title('Фактичні і прогнозовані значення')

plt.grid()

plt.show()

Результат виконання

**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

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], '.', \

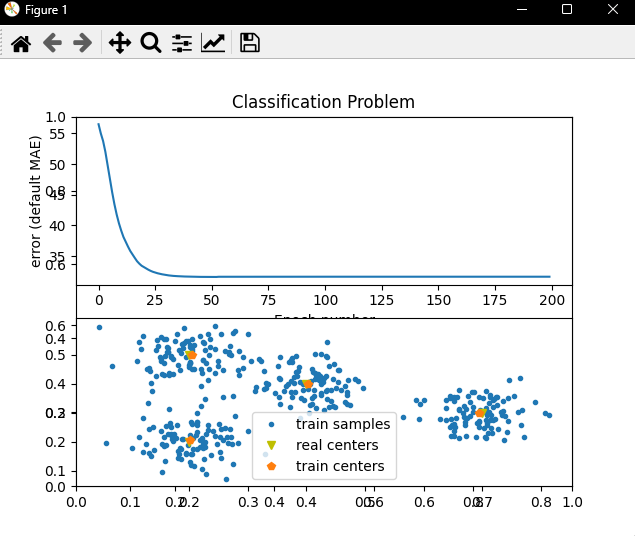
        centr[:,0], centr[:, 1] , 'yv', \

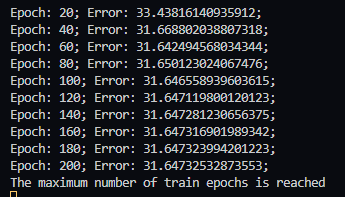
        w[:,0], w[:,1], 'p')

pl.legend(['train samples', 'real centers', 'train centers'])

pl.show()

Результат виконання





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

Лістинг програми

*import* numpy *as* np

*import* neurolab *as* nl

*import* numpy.random *as* rand

skv = 0.03

centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.3, 0.3], [0.2, 0.6], [0.5, 0.7]])

rand\_norm = skv \* rand.randn(100, 5, 2)

inp = np.array([centr + r *for* r *in* rand\_norm])

inp.shape = (100 \* 5, 2)

rand.shuffle(inp)

*# Create net with 2 inputs and 4 neurons*

net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 5)

*# 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], '.', \

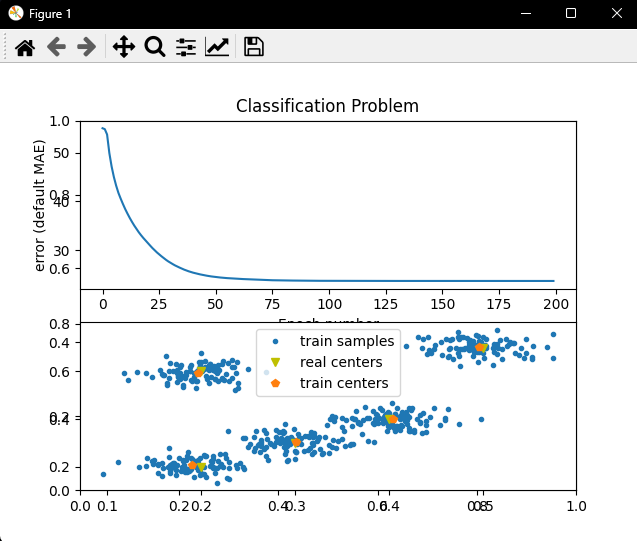
        centr[:,0], centr[:, 1] , 'yv', \

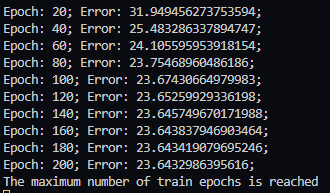
        w[:,0], w[:,1], 'p')

pl.legend(['train samples', 'real centers', 'train centers'])

pl.show()

Результат виконання





Посилання на GitHub - https://github.com/MischenchukMykola/lab6

**Висновок:** виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python навчився створювати та застосовуватипрості нейронні мережі***.***