**ЛАБОРАТОРНА РОБОТА № 5**

**РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ**

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

**Хід роботи:**

**Завдання №1:** Створити простий нейрон.

Лістинг програми:

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 – Результутат виконання програми.

**Завдання №2:** Створити просту нейронну мережу для передбачення статі людини.

Лістинг програми:

import numpy as np  
from LR\_5\_task\_1 import Neuron, sigmoid  
  
  
def derivative\_sigmoid(x):  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
  
def mse\_loss(y\_true, y\_pred):  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
  
class OleksiichukNeuralNetwork:  
 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 = 100  
  
 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  
  
 d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)  
  
 # Neuron o1  
 d\_ypred\_d\_w5 = h1 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_w6 = h2 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_b3 = derivative\_sigmoid(sum\_o1)  
  
 d\_ypred\_d\_h1 = self.w5 \* derivative\_sigmoid(sum\_o1)  
 d\_ypred\_d\_h2 = self.w6 \* derivative\_sigmoid(sum\_o1)  
  
 # Neuron h1  
 d\_h1\_d\_w1 = x[0] \* derivative\_sigmoid(sum\_h1)  
 d\_h1\_d\_w2 = x[1] \* derivative\_sigmoid(sum\_h1)  
 d\_h1\_d\_b1 = derivative\_sigmoid(sum\_h1)  
  
 # Neuron h2  
 d\_h2\_d\_w3 = x[0] \* derivative\_sigmoid(sum\_h2)  
 d\_h2\_d\_w4 = x[1] \* derivative\_sigmoid(sum\_h2)  
 d\_h2\_d\_b2 = derivative\_sigmoid(sum\_h2)  
  
 # Update weights and biases  
 # Neuron 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  
  
 # Neuron 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  
  
 # Neuron 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 = OleksiichukNeuralNetwork()  
 network.train(data, all\_y\_trues)  
  
 emily = np.array([-7, -3]) # 128 pounds, 63 inches  
 frank = np.array([20, 2]) # 155 pounds, 68 inches  
 print("Emily: %.3f" % network.feedforward(emily)) # +-0.96 - F  
 print("Frank: %.3f" % network.feedforward(frank)) # +-0.039 - M

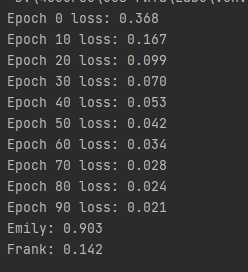


Рис.2 – Результутат виконання програми.

**Завдання №3:** Класифікатор на основі перцептрону з використанням бібліотеки NeuroLab.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
# Завантаження вхідних даних  
text = np.loadtxt('data\_perceptron.txt')  
# Поділ даних на точки даних та мітки  
data = text[:, :2]  
labels = text[:, 2].reshape((text.shape[0], 1))  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data[:, 0], data[:, 1])  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
# Визначення максимального та мінімального значень для кожного виміру  
dim1\_min, dim1\_max, dim2\_min, dim2\_max = 0, 1, 0, 1  
# Кількість нейронів у вихідному шарі  
num\_output = labels.shape[1]  
# Визначення перцептрону з двома вхідними нейронами (оскільки  
# Вхідні дані - двовимірні)  
dim1 = [dim1\_min, dim1\_max]  
dim2 = [dim2\_min, dim2\_max]  
perceptron = nl.net.newp([dim1, dim2], num\_output)  
# Тренування перцептрону з використанням наших даних  
error\_progress = perceptron.train(data, labels, epochs=100, show=20, lr=0.03)  
# Побудова графіка 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()

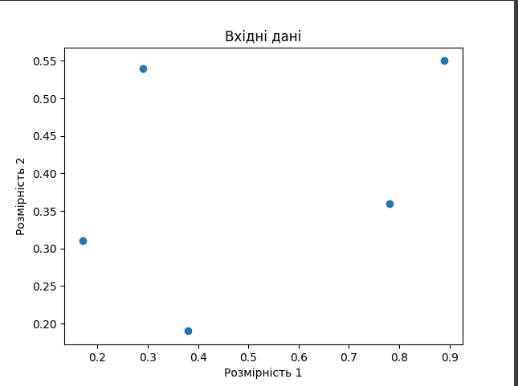


Рис.3 – Результутат виконання програми.

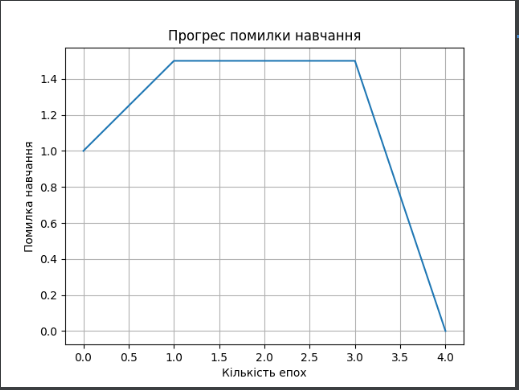


Рис.4 – Результутат виконання програми.

**Завдання №4:** Побудова одношарової нейронної мережі.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
import neurolab as nl  
# Завантаження вхідних даних  
text = np.loadtxt('data\_simple\_nn.txt')  
# Поділ даних на точки даних та мітки  
data = text[:, 0:2]  
labels = text[:, 2:]  
# Побудова графіка вхідних даних  
plt.figure()  
plt.scatter(data[:, 0], data[:, 1])  
plt.xlabel('Розмірність 1')  
plt.ylabel('Розмірність 2')  
plt.title('Вхідні дані')  
# 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 одношарової нейронної мережі  
dim1 = [dim1\_min, dim1\_max]  
dim2 = [dim2\_min, dim2\_max]  
nn = nl.net.newp([dim1, dim2], num\_output)  
# Навчання нейронної мережі  
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()  
# Виконання класифікатора на тестових точках даних  
print('\nРезультати тесту:')  
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])

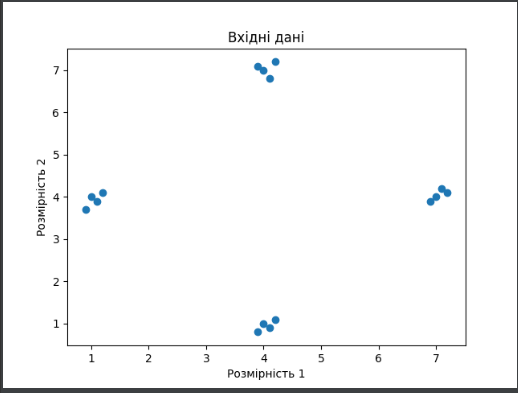


Рис.5 – Результутат виконання програми.

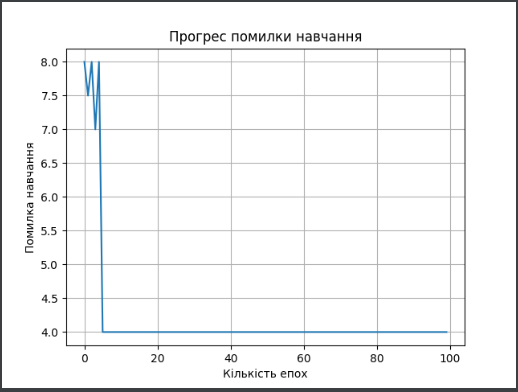


Рис.6 – Результутат виконання програми.

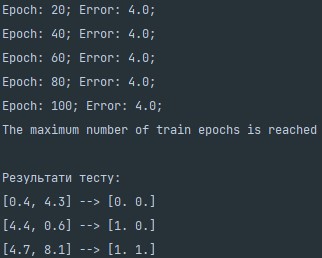


Рис.7 – Результутат виконання програми.

**Завдання №5:** Побудова багатошарової нейронної мережі.

Лістинг програми:

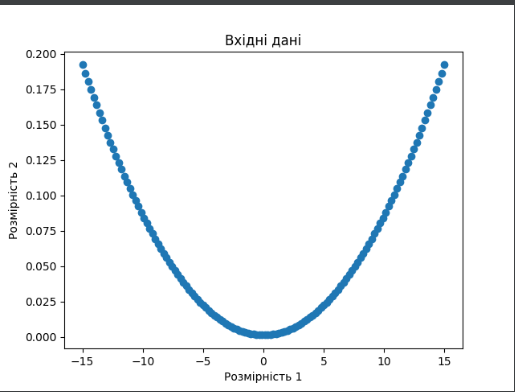


Рис.8 – Результутат виконання програми.

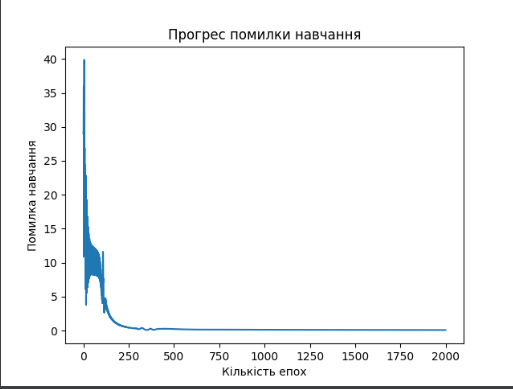


Рис.9 – Результутат виконання програми.

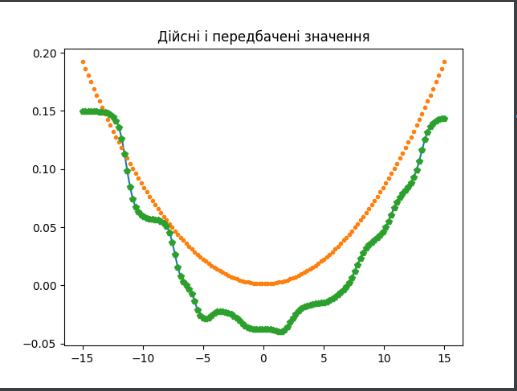


Рис.10 – Результутат виконання програми.

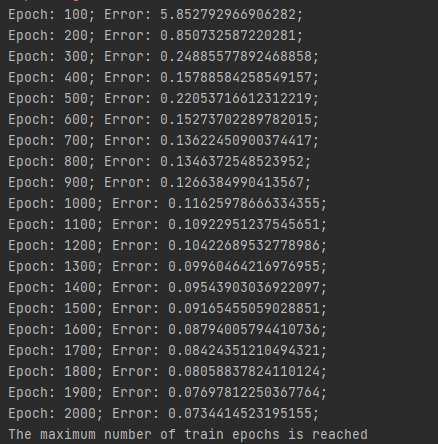


Рис.11 – Результутат виконання програми.

**Завдання №6:** Побудова багатошарової нейронної мережі для свого варіанту.

Табл. 1

|  |  |
| --- | --- |
| № варіанта | Тестові дані |
| Варіант 21 | y = 6x2+2х+8 |

Табл. 2

|  |  |  |
| --- | --- | --- |
| Номер варіанта | Багатошаровий персептрон | |
| Кількість шарів | Кількості нейронів у шарах |
| 21 | 2 | 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 = 6 \* np.square(x) + 2\*x + 1

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('Розмірність 1')

plt.ylabel('Розмірність 2')

plt.title('Вхідні дані')

# Визначення багатошарової нейронної мережі з двома прихованими

# шарами. Перший прихований шар складається із трьох нейронів.

# Другий прихований шар складається з одного нейрона.

# Вихідний шар складається з одного нейрона

nn = nl.net.newff([[min\_val, max\_val]], [2, 3, 1])

# Завдання 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('Кількість епох')

plt.ylabel('Помилка навчання')

plt.title('Прогрес помилки навчання')

# 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('Дійсні і передбачені значення')

plt.show()

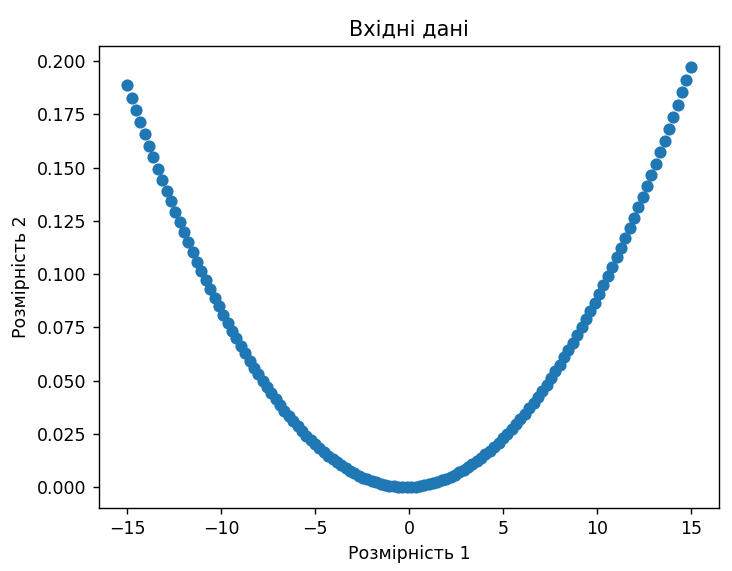


Рис.12 – Результутат виконання програми.

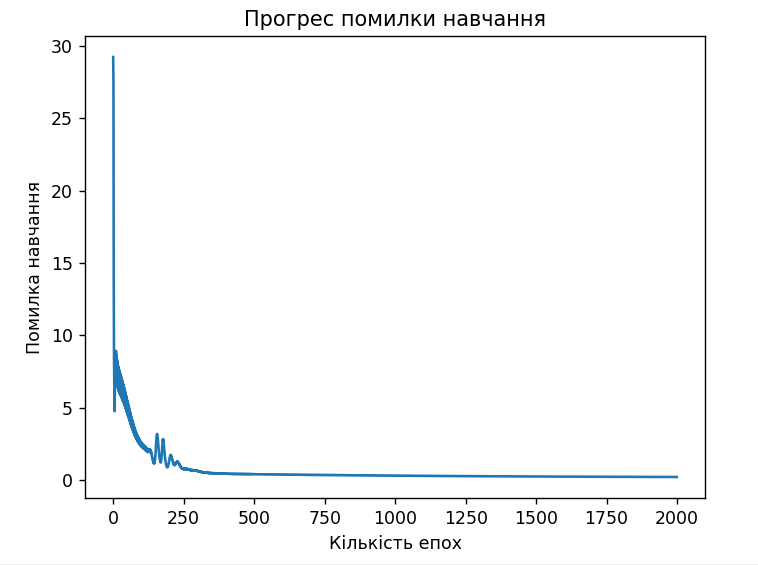


Рис.13 – Результутат виконання програми.

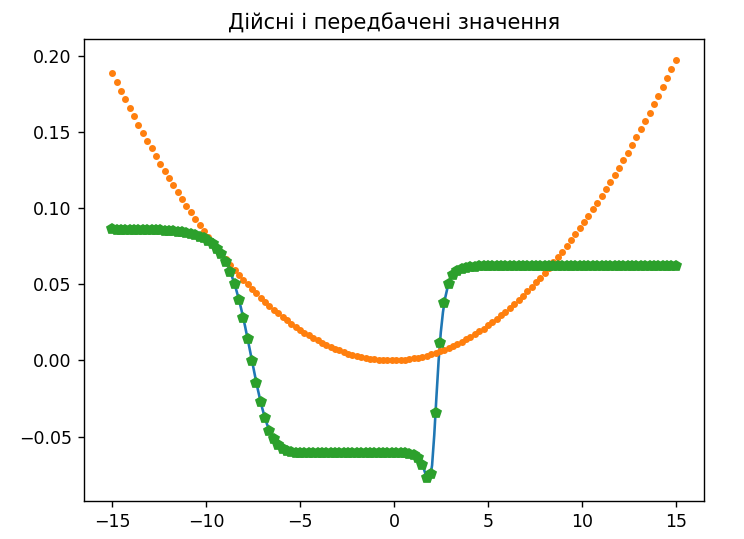


Рис.14 – Результутат виконання програми.

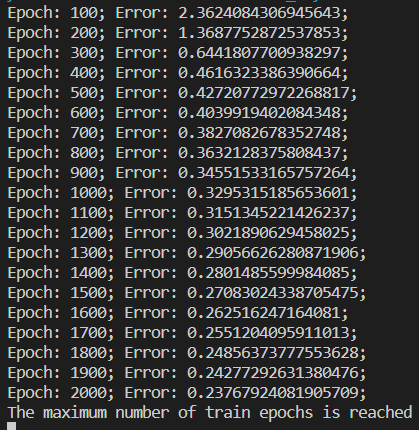


Рис.15 – Результутат виконання програми.

**Завдання №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  
center = 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([center + 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:  
fig, axs = pl.subplots(2)  
fig.suptitle('Classification Problem')  
axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')  
axs[1].plot(error)  
w = net.layers[0].np['w']  
axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],  
w[:, 1], 'p')  
axs[0].legend(['train samples', 'real centers', 'train centers'], loc='upper left')  
pl.show()

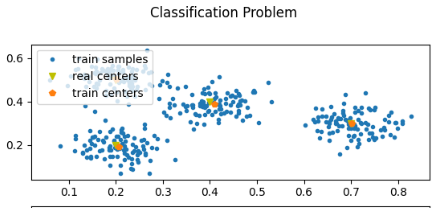


Рис.16 – Результутат виконання програми.

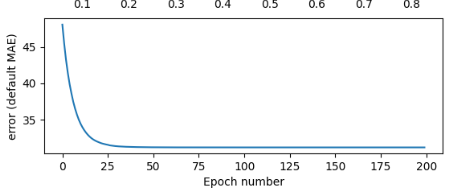


Рис.17 – Результутат виконання програми.

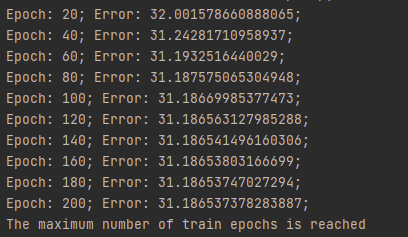


Рис.18 – Результутат виконання програми.

Помилка MAE (Mean Absolute Error) - це середня абсолютна помилка, яка визначається як середня абсолютна різниця між фактичними та прогнозованими значеннями. Вона є лінійною мірою помилки, що означає, що кожна помилка має однаковий вплив на загальну оцінку помилки, незалежно від того, наскільки велика вона.

По результатам можна зробити висновок, що мережа навчається, так як помилка МАЕ зменшується і чим більше поколінь, тим точніше будуть кінцеві дані. Так само і з центрами кластерів – чим більше поколінь, тим точніші дані. Це означає, що мережа навчається.

**Завдання №8:** Дослідження нейронної мережі на основі карти Кохонена, що самоорганізується.

Табл. 3

|  |  |  |
| --- | --- | --- |
| № варіанту | Центри кластера | skv |
| Варіант 21 | [0.3, 0.3], [0.4, 0.4], [0.5, 0.3], [0.2, 0.6], [0.4, 0.5] | 0,07 |

Лістинг програми:

import numpy as np

import neurolab as nl

import numpy.random as rand

import pylab as pl

skv = 0.07

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

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

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

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

rand.shuffle(inp)

# Create net with 2 inputs and 5 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:

fig, axs = pl.subplots(2)

fig.suptitle('Classification Problem')

axs.flat[1].set(xlabel='Epoch number', ylabel='error (default MAE)')

axs[1].plot(error)

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

axs[0].plot(inp[:, 0], inp[:, 1], '.', center[:, 0], center[:, 1], 'yv', w[:, 0],

w[:, 1], 'p')

axs[0].legend(['train samples', 'real centers', 'train centers'],loc='upper left')

pl.show()

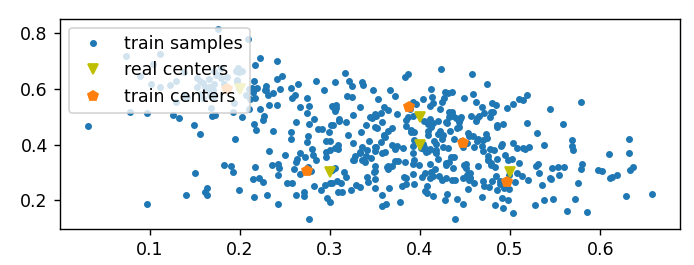


Рис.19 – Результутат виконання програми.

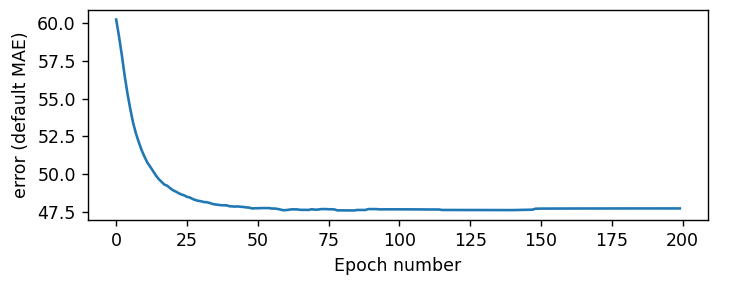


Рис.20 – Результутат виконання програми.

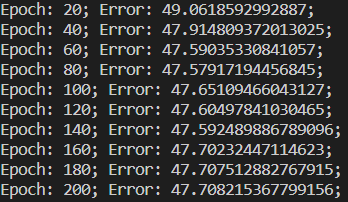


Рис.21 – Результутат виконання програми.

**Висновки:** були досліджені прості нейронні мережі, використовуючи спеціалізовані бібліотеки та мову програмування Python. Створено одношарову та багатошарову нейронні мережі. Отримані знання про помилку МАЕ, за що вона відповідає та її відображення на графіках.

Посилання на репозиторій: <https://github.com/RomanMatskevich/SHI.git>