**ЛАБОРАТОРНА РОБОТА № 5**

РОЗРОБКА ПРОСТИХ НЕЙРОННИХ МЕРЕЖ

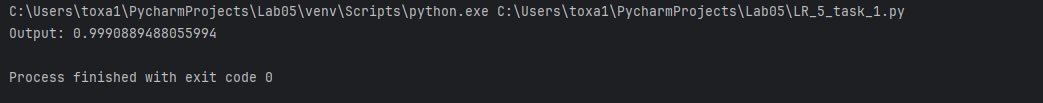
**Мета роботи**: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати та застосовувати прості нейронні мережі.

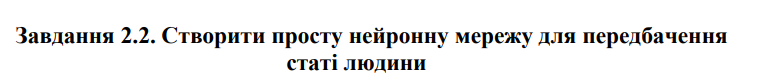
GitHub репозиторій: **https://github.com/SMTH666/OAI-Lab**



**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()

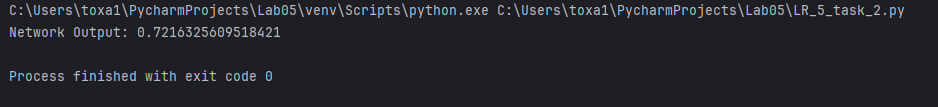
**Результат виконання:  
**

****

**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:**

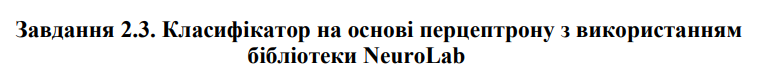
****

**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.



**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:**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание**

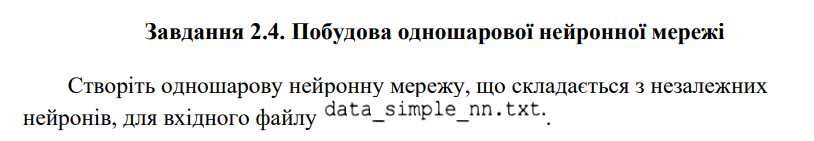
Графік №1. Вхідні дані.

**Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание**

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.

****

**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 - predictions), 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:**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание**

Графік №1. Вхідні дані.

**Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание**

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.

**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:**

**Изображение выглядит как снимок экрана, текст, черный

Автоматически созданное описание**

**Изображение выглядит как текст, График, диаграмма, снимок экрана

Автоматически созданное описание** **Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, диаграмма, График

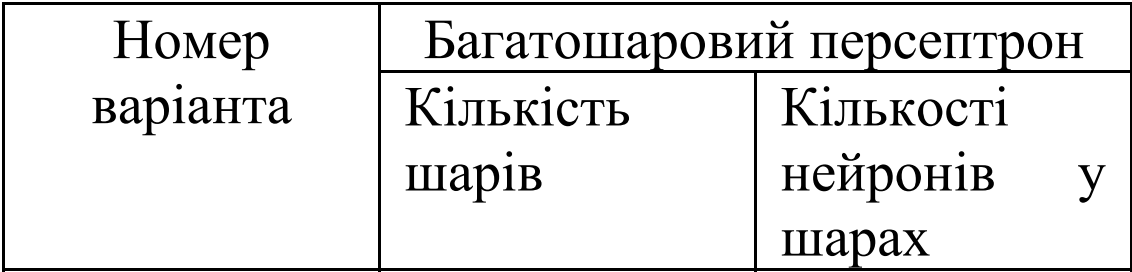
Автоматически созданное описание**

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











**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:**

**Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание**

Изображение выглядит как текст, диаграмма, График, линия

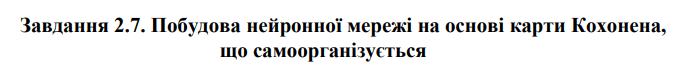
Автоматически созданное описание Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**Изображение выглядит как диаграмма, График, линия, текст

Автоматически созданное описание**

**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).



**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:**

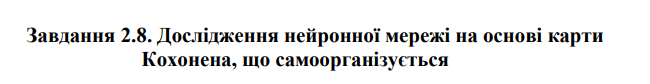
**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание**

**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 похибка змінюється під час навчання нейромережі.









**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:**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание**

****

**Результат виконання:  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание**

**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.