Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №1

За шестой семестр

По дисциплине: «Модели решения задач в интеллектуальных системах»

Тема: «Бинарная классификация»

Выполнил:
Студент 3 курса
Группы ИИ-26(1)
Прокопюк А.Д.
Проверила:
Андренко К.В.

Брест 2026

Цель работы: Изучить принципы бинарной классификации и реализовать однослойную нейронную сеть (персептрон) для решения задачи классификации с использованием пороговой функции активации, а также исследовать процесс обучения модели с применением среднеквадратичной ошибки (MSE).

Задачи лабораторной работы:

1. Реализовать алгоритм обучения однослойной нейронной сети с использованием **MSE** в качестве функции ошибки.
2. Провести обучение сети с **разными значениями шага обучения** и построить **график зависимости** MSE от номера эпохи.
3. Выполнить визуализацию результатов классификации:
   - исходные точки обучающей выборки,
   - разделяющую линию (границу между двумя классами).
4. Реализовать режим функционирования сети:
   - пользователь задаёт произвольный входной вектор,
   - сеть вычисляет выходной класс,
   - соответствующая точка отображается на графике,
   - для корректной визуализации рекомендуется выбирать значения из диапазона $-1.5 \leq x1, x2 \leq 1.5$.

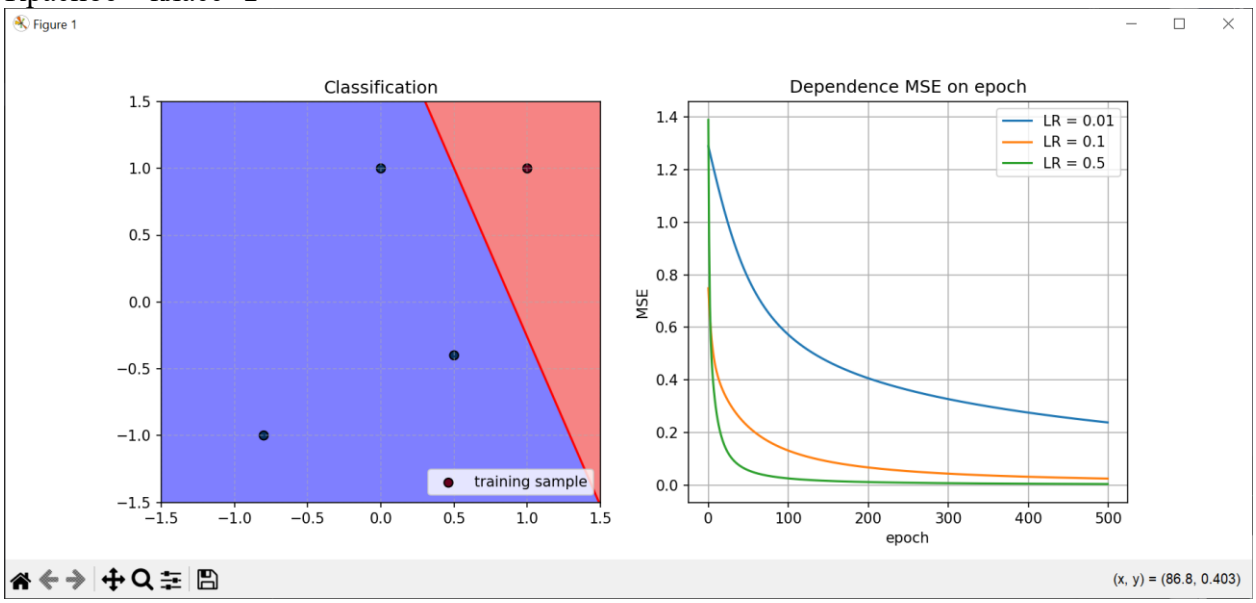
Код программы:

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

class Perceptron:
    def __init__(self, input_size=0, learning_rate=0.1):
        self.X = np.array([])
        self.w = np.random.uniform(-1.0, 1.0, input_size + 1)# threshold w[0]
        self.learning_rate = learning_rate
        self.target = np.array([])

    def set_X(self, X: np.array) -> None:
        X = np.array(X)

        if X.ndim == 1:
            X = X.reshape(1, -1)
        elif X.ndim != 2:
            print("X must be 1D or 2D array")
            return

        self.X = X
        self.X = np.insert(self.X, 0, -1, axis=1)

    def set_w(self, w: np.array) -> None:
        self.w = w          # the first one is a threshold

    def set_target(self, target: np.array) -> None:
        if self.X.ndim != 2:
            print("X not setted!")
            return

        if len(target) != len(self.X):
            print(f"Invalid size of target vector. It must have length = {len(self.X)}. Now =
{len(target)}")
            return

        self.target = target

    def activate(self, arr_wsum: np.array) -> np.array:
        return 2 / (1 + np.exp(-arr_wsum)) - 1

    def der_activate(self, y_pred: np.array) -> np.array:
        return 0.5 * (1 - y_pred**2)

    def forward(self, X_input=None) -> np.array:
        X = self.X if X_input is None else X_input
```

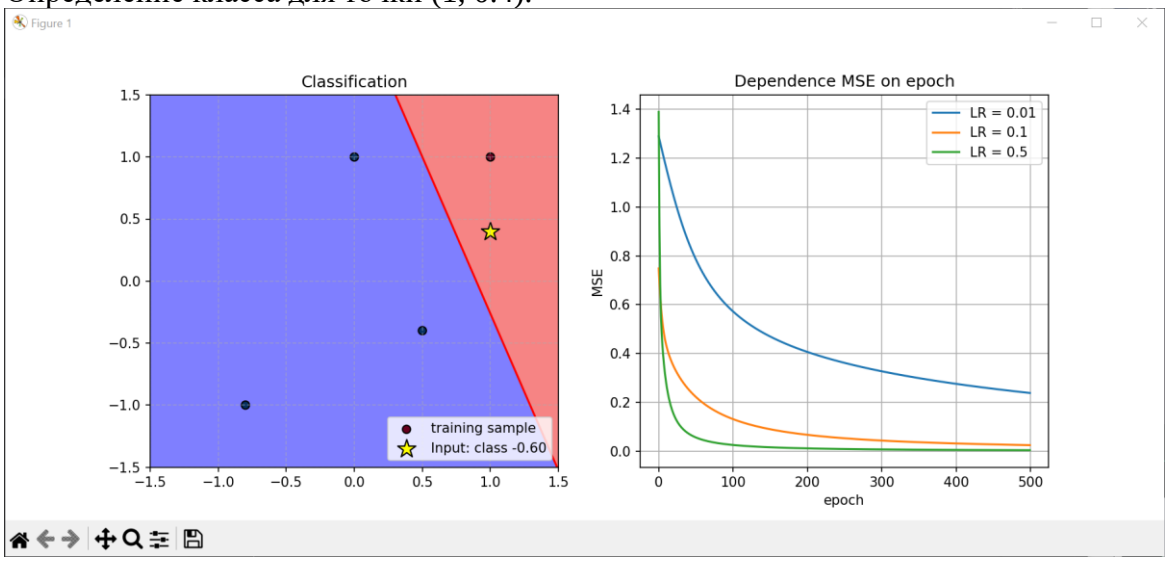
```
        if X.size == 0:
            print("Input X vector not set!")
            return None

        wsum = np.dot(X, self.w)
        y = self.activate(wsum)

        return y

    def train(self, epochs = 1000) -> list:
        mse_history = []
        for epoch in range(epochs):
            y_pred = self.forward()

            error = y_pred - self.target # gamma

            mse = np.mean(error**2)
            mse_history.append(mse)

            self.w = self.w - np.dot(self.learning_rate * error * self.der_activate(y_pred), self.X)

        return mse_history

X_train = np.array([[1, 1.0], [0.5, -0.4], [-0.8, -1], [0, 1]])
Y_targets = np.array([-1, 1, 1, 1])

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 2)
for lr in [0.01, 0.1, 0.5]:
    test_p = Perceptron(input_size=2, learning_rate=lr)
    test_p.set_X(X_train)
    test_p.set_target(Y_targets)
    history = test_p.train(epochs=500)
    plt.plot(history, label=f'LR = {lr}')

plt.title("Dependence MSE on epoch")
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.legend()
plt.grid(True)

p = Perceptron(input_size=2, learning_rate=0.1)
p.set_X(X_train)
p.set_target(Y_targets)
p.train(epochs=1000)

def plot_current_state(user_point=None, user_class=None):
    plt.subplot(1, 2, 1)
    plt.cla()

    xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
    grid_points = np.c_[np.ones(xx.ravel().shape) * -1, xx.ravel(), yy.ravel()]
    Z = p.forward(grid_points).reshape(xx.shape)

    plt.contourf(xx, yy, Z, levels=0, colors=["#ef0a0a", "#0000ff"], alpha=0.5)
    plt.contour(xx, yy, Z, levels=[0], colors='r')
    plt.scatter(X_train[:,0], X_train[:,1], c=Y_targets, cmap='RdBu', edgecolors='k', label='training
sample')

    if user_point is not None:
        plt.scatter(user_point[0],     user_point[1],     color='yellow',     marker='*',     s=200,
edgecolors='black', label=f'Input: class {user_class:.2f}')

    plt.xlim([-1.5, 1.5])
    plt.ylim([-1.5, 1.5])
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.title("Classification")
    plt.pause(0.1)

plot_current_state()
plt.show(block=False)

print("Input coordinates of point (x1, x2) from [-1.0, 0.1]")
try:
    while True:
        line = input("Input x1 x2 (through space or type'exit' for exit): ")
```

```
        if line.lower() == 'exit': break

        coords = list(map(float, line.split()))

        user_x = np.array([[-1, coords[0], coords[1]]])
        prediction = p.forward(user_x)[0]

        print(f"Result: {prediction:.4f} (Class {'1' if prediction > 0 else '-1'})")

        plt.subplot(1, 2, 1)
        plot_current_state(coords, prediction)
        plt.draw()
except ValueError:
    print("Exit...")

plt.show()
```

Результат:
Синее – класс 1
Красное – класс -1



Определение класса для точки (1, 0.4):



Вывод: Изучил принципы бинарной классификации и реализовал однослойную нейронную сеть (персептрон) для решения задачи классификации с использованием пороговой функции активации, а также исследовать процесс обучения модели с применением среднеквадратичной ошибки (MSE).