Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «Модели решения задач в интеллектуальных системах»

Тема: «Бинарная классификация»

Выполнил:

Студент 3 курса

Группы ИИ-26(1)

Пасевич К.Ю.

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).

Ход 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, например $-0.5 \leq x_1, x_2 \leq 1.5$.

5. Написать вывод по выполненной работе.

- Допускается применение математических и графических библиотек
- ML-библиотеки и ML-фреймворки использовать нельзя (например: scikit-learn, TensorFlow, PyTorch - запрещены).

Вариант 9

9.

x_1	x_2	e
2	1	0
-2	1	1
2	-1	0
-2	-1	0

x_1, x_2 - входные данные сети, e - эталонные значения

Код программы:

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

data = [
```

```
    {'a': 2, 'b': 1, 'e': 0},
    {'a': -2, 'b': 1, 'e': 1},
    {'a': 2, 'b': -1, 'e': 0},
    {'a': -2, 'b': -1, 'e': 0}
]

class Neuron:
    def __init__(self, rate=0.1):
        self.w1 = random.random() - 0.5
        self.w2 = random.random() - 0.5
        self.b = random.random() - 0.5
        self.r = rate

    def step(self, s):
        return 1 if s >= 0 else 0

    def fit(self, data):
        e_sum = 0
        for item in data:
            s = (item['a'] * self.w1) + (item['b'] * self.w2) + self.b
            pred = self.step(s)
            err = item['e'] - pred

            if err != 0:
                self.w1 += self.r * err * item['a']
                self.w2 += self.r * err * item['b']
                self.b += self.r * err

            e_sum += err ** 2

        return e_sum / len(data)

class Plotter:
    def __init__(self):
        self.fig, (self.ax1, self.ax2) = plt.subplots(1, 2,
figsize=(12, 5))

    def show(self, mse_hist, model, data, user=None):
        self.ax1.clear()
        self.ax1.plot(mse_hist, color="#FF6B6B", linewidth=2.5)
        self.ax1.set_title("Training Error Over Time")
        self.ax1.set_xlabel("Iteration")
        self.ax1.set_ylabel("Error Rate")
        self.ax1.grid(True, alpha=0.3, color='#333333')

        self.ax2.clear()
        self.ax2.set_title("Classification Boundary")
        self.ax2.axhline(0, color='#888888', linewidth=0.8,
linestyle='--', alpha=0.5)
        self.ax2.axvline(0, color='#888888', linewidth=0.8,
```

```
linestyle='--', alpha=0.5)

        x_vals = np.linspace(-3, 3, 100)
        if model.w2 != 0:
            y_vals = (-model.w1 * x_vals - model.b) / model.w2
            self.ax2.plot(x_vals, y_vals, color='#9B59B6',
linewidth=2.5, label='Separating Line')

        for p in data:
            if p['e'] == 1:
                color = '#2ECC71'
            else:
                color = '#F39C12'
            self.ax2.scatter(p['a'], p['b'], color=color, s=120,
edgecolors='#1E1E1E', linewidth=1.5, zorder=3)
            self.ax2.text(p['a'] + 0.2, p['b'],
f"({p['a']},{p['b']})", fontsize=8, color='#1E1E1E')

        if user:
            ux, uy, ue = user['a'], user['b'], user['e']
            self.ax2.scatter(ux, uy, c='#E74C3C', marker='D', s=140,
edgecolors='#1E1E1E', linewidth=1.5, label=f"Test Point: Group {ue}",
zorder=4)
            self.ax2.text(ux + 0.2, uy, f"Group {ue}", fontsize=9,
fontweight='bold', color='#E74C3C')

        self.ax2.set_xlim(-4, 4)
        self.ax2.set_ylim(-4, 4)
        self.ax2.legend(loc='upper right', framealpha=0.9)
        self.ax2.grid(True, linestyle='--', alpha=0.3,
color='#333333')
        plt.tight_layout()
        plt.show()

model = Neuron(rate=0.05)
plot = Plotter()
mse_hist = []

print("Training dataset:")
for i, point in enumerate(data):
    print(f"Point {i + 1}: X1={point['a']}, X2={point['b']},
Label={point['e']}")

for epoch in range(500):
    mse = model.fit(data)
    mse_hist.append(mse)
    print(f"Epoch {epoch}: MSE = {mse:.6f}")
    if mse == 0:
        print(f"Training finished early at epoch: {epoch}")
        break
```

```
print(f"Training completed! Final MSE: {mse_hist[-1]:.6f}")
print(f"Weights: w1 = {model.w1:.4f}, w2 = {model.w2:.4f}, bias =
{model.b:.4f}")

def predict(x1, x2):
    s = x1 * model.w1 + x2 * model.w2 + model.b
    e = model.step(s)
    print(f"Prediction for point ({x1}, {x2}) -> Class: {e}")
    return {'a': x1, 'b': x2, 'e': e}

user_pt = predict(0, -3)
plot.show(mse_hist, model, data, user_pt)

print("\nTesting trained network on original data:")
for point in data:
    s = point['a'] * model.w1 + point['b'] * model.w2 + model.b
    pred = model.step(s)
    check = '+' if pred == point['e'] else '-'
    print(f"Point ({point['a']:2}, {point['b']:2}):
Label={point['e']}, Prediction={pred} {check}")
```

Результат тестирования:

Training dataset:

Point 1: X1=2, X2=1, Label=0

Point 2: X1=-2, X2=1, Label=1

Point 3: X1=2, X2=-1, Label=0

Point 4: X1=-2, X2=-1, Label=0

Epoch 0: MSE = 0.250000

Epoch 1: MSE = 0.500000

Epoch 2: MSE = 0.250000

Epoch 3: MSE = 0.750000

Epoch 4: MSE = 0.000000

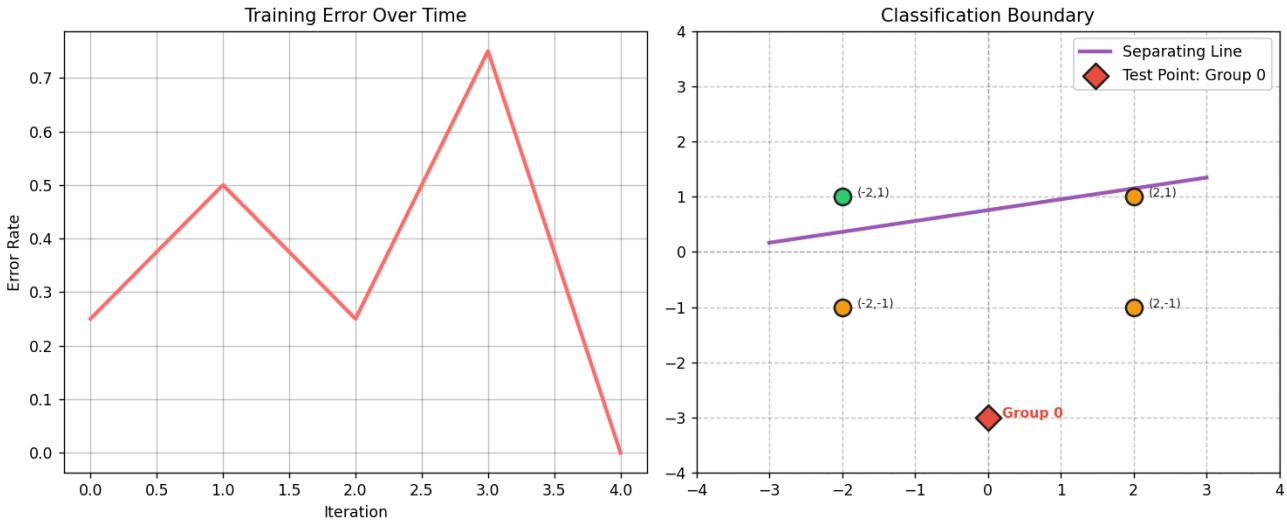
Training finished early at epoch: 4

Training completed! Final MSE: 0.000000

Weights: w1 = -0.0202, w2 = 0.1030, bias = -0.0779

Prediction for point (0, -3) -> Class: 0

График среднеквадратичной ошибки и график с разделяющей поверхностью:



Вывод: в процессе выполнения работы был реализован персептрон для задачи бинарной классификации, освоен принцип его обучения на основе пороговой функции активации и выполнена оценка сходимости с помощью среднеквадратичной ошибки.