МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТк

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №1

Специальность ИИ26(з)

Выполнил
Д.А.Турич,
студент группы ИИ-26

Проверил
К.В. Андренко,
ст. преп. кафедры ИИТ,
«___»_____2026 г.

Брест 2026

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

Задание 1. Для заданного массива вещественных чисел найти среднее значение и стандартное отклонение.

Задачи лабораторной работы:
Реализовать алгоритм обучения однослойной нейронной сети с использованием MSE в качестве функции ошибки.
Провести обучение сети с разными значениями шага обучения и построить график зави- симости MSE от номера эпохи.
Выполнить визуализацию результатов классификации:
исходные точки обучающей выборки,
разделяющую линию (границу между двумя классами).
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 ВСТАВИТЬ СВОЙ ДИАПАЗОН, например -0.5≤x1,x2≤1.5
Написать вывод по выполненной работе.
Допускается применение математических и графических библиотек
ML-библиотеки и ML-фреймворки использовать нельзя (например: scikit-learn, TensorFlow, PyTorch - запрещены)

x_1	x_2	e
3	6	0
-3	6	0
3	-6	1
-3	-6	1

Код программы:

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

X_raw = np.array([
    [3.0, 6.0],
    [-3.0, 6.0],
    [3.0, -6.0],
    [-3.0, -6.0],
], dtype=float)

E = np.array([0.0, 0.0, 1.0, 1.0], dtype=float)
```

```
x_scale = np.max(np.abs(X_raw), axis=0)
X = X_raw / x_scale


def step_01(s: float, threshold: float = 0.5) -> int:
    return 1 if s >= threshold else 0


def mse_stable(y_lin: np.ndarray, e: np.ndarray) -> float:
    diff = np.clip(e - y_lin, -1e6, 1e6)
    return float(np.mean(diff * diff))


class SingleLayerNet:
    def __init__(self, lr: float = 0.1, seed: int = 42, w_clip: float = 50.0):
        rng = np.random.default_rng(seed)
        self.w = rng.uniform(-0.5, 0.5, size=2).astype(float)
        self.b = float(rng.uniform(-0.5, 0.5))
        self.lr = float(lr)
        self.w_clip = float(w_clip)

    def forward_linear(self, x: np.ndarray) -> float:
        return float(np.dot(self.w, x) + self.b)

    def predict_class(self, x: np.ndarray, threshold: float = 0.5) -> int:
        return step_01(self.forward_linear(x), threshold)

    def train_epoch(self, X: np.ndarray, E: np.ndarray, shuffle: bool = True, seed: int = 0) -> float:
        idx = np.arange(len(X))
        if shuffle:
            rng = np.random.default_rng(seed)
            rng.shuffle(idx)

        for i in idx:
            x = X[i]
            e = E[i]
            y_lin = self.forward_linear(x)
            err = e - y_lin

            self.w += self.lr * err * x
            self.b += self.lr * err

            self.w = np.clip(self.w, -self.w_clip, self.w_clip)
```

```
            self.b = float(np.clip(self.b, -self.w_clip, self.w_clip))

        y_all = np.array([self.forward_linear(x) for x in X], dtype=float)
        return mse_stable(y_all, E)


def plot_mse_histories(histories: dict):
    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111)

    for lr, hist in histories.items():
        ax.plot(range(1, len(hist) + 1), hist, label=f"lr={lr}", linewidth=2)

    ax.set_xlabel("Номер эпохи", fontsize=12)
    ax.set_ylabel("MSE", fontsize=12)
    ax.set_title("Зависимость MSE от номера эпохи", fontsize=14)
    ax.grid(True, alpha=0.3)
    ax.legend(fontsize=10)
    ax.set_yscale('log')

    fig.tight_layout()
    fig.savefig("mse_plot.png", dpi=200)
    plt.show()


def plot_points_and_boundary(model: SingleLayerNet,
                             X_raw: np.ndarray,
                             E: np.ndarray,
                             x_scale: np.ndarray,
                             user_points=None,
                             xlim=(-7, 7),
                             ylim=(-7, 7),
                             threshold: float = 0.5):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111)

    cls0 = X_raw[E == 0]
    cls1 = X_raw[E == 1]

    if len(cls0):
        ax.scatter(cls0[:, 0], cls0[:, 1], marker="o", s=150,
                   color='blue', alpha=0.7, label="Класс 0 (e=0)", edgecolors='black')
    if len(cls1):
        ax.scatter(cls1[:, 0], cls1[:, 1], marker="^", s=150,
```

```
                color='red', alpha=0.7, label="Класс 1 (e=1)", edgecolors='black')

    a1 = model.w[0] / x_scale[0]
    a2 = model.w[1] / x_scale[1]
    b = model.b

    xs = np.linspace(xlim[0], xlim[1], 400)
    if abs(a2) > 1e-12:
        ys = (threshold - b - a1 * xs) / a2
        ax.plot(xs, ys, 'g-', linewidth=2.5, label="Разделяющая линия")
    else:
        if abs(a1) > 1e-12:
            x0 = (threshold - b) / a1
            ax.axvline(x0, color='g', linewidth=2.5, label="Разделяющая линия")

    if user_points:
        up = np.array(user_points, dtype=float)
        ax.scatter(up[:, 0], up[:, 1], marker="s", s=200,
                   color='purple', alpha=0.7, label="Точки пользователя",
                   edgecolors='black', zorder=5)

    ax.axhline(0, color='black', linewidth=1, alpha=0.5)
    ax.axvline(0, color='black', linewidth=1, alpha=0.5)
    ax.set_xlim(*xlim)
    ax.set_ylim(*ylim)
    ax.set_xlabel("x1", fontsize=12)
    ax.set_ylabel("x2", fontsize=12)
    ax.set_title("Точки обучающей выборки и граница классов", fontsize=14)
    ax.grid(True, alpha=0.3)
    ax.legend(fontsize=11, loc='upper right')

    fig.tight_layout()
    fig.savefig("decision_boundary.png", dpi=200)
    plt.show()


def main():
    learning_rates = [0.01, 0.05, 0.1, 0.2, 0.5, 0.8]
    max_epochs = 100
    threshold = 0.5

    histories = {}
    best_model = None
    best_lr = None
```

```
best_final_mse = float("inf")

print("=" * 60)
print("ОБУЧЕНИЕ ОДНОСЛОЙНОЙ НЕЙРОННОЙ СЕТИ")
print("=" * 60)
print(f"Данные: {X_raw.tolist()}")
print(f"Целевые значения: {E.tolist()}")
print("-" * 60)

for lr in learning_rates:
    model = SingleLayerNet(lr=lr, seed=42, w_clip=50.0)
    hist = []

    for ep in range(max_epochs):
        cur_mse = model.train_epoch(X, E, shuffle=True, seed=1000 + ep)
        hist.append(cur_mse)
        if cur_mse < 1e-8:
            break

    histories[lr] = hist
    print(f"lr = {lr:4.2f}: итоговая MSE = {hist[-1]:.2e}, эпох = {len(hist)}")

    if hist[-1] < best_final_mse:
        best_final_mse = hist[-1]
        best_model = model
        best_lr = lr

print("-" * 60)
print("ЛУЧШИЙ РЕЗУЛЬТАТ")
print(f"  lr = {best_lr}")
print(f"  w1 = {best_model.w[0]:.6f}, w2 = {best_model.w[1]:.6f}")
print(f"  b  = {best_model.b:.6f}")
print(f"  final MSE = {best_final_mse:.2e}")
print("-" * 60)

print("\nПРОВЕРКА КЛАССИФИКАЦИИ ОБУЧАЮЩИХ ТОЧЕК")
print("-" * 40)
for x_raw, e in zip(X_raw, E):
    x_norm = x_raw / x_scale
    y = best_model.predict_class(x_norm, threshold=threshold)
    result = "✓" if y == e else "✗"
    print(f"x=[{x_raw[0]:3.0f}, {x_raw[1]:3.0f}] -> предсказание: {y} (цель: {int(e)}) {result}")

print("\nПостроение графиков...")
```

```
    plot_mse_histories(histories)
    plot_points_and_boundary(best_model, X_raw, E, x_scale, user_points=None, threshold=threshold)

    print("\n" + "=" * 60)
    print("РЕЖИМ ФУНКЦИОНИРОВАНИЯ")
    print("=" * 60)
    print("Введите координаты x1 x2 для классификации")
    print("Для выхода введите 'q'")
    print("-" * 40)

    user_points = []
    while True:
        s = input("x1 x2 > ").strip()
        if s.lower() in ("q", "quit", "exit"):
            break

        try:
            parts = s.replace(",", " ").split()
            if len(parts) != 2:
                print("Ошибка: введите два числа через пробел")
                continue

            x1_str, x2_str = parts
            x_user_raw = np.array([float(x1_str), float(x2_str)], dtype=float)
        except ValueError:
            print("Ошибка: введите корректные числа")
            continue
        except Exception:
            print("Ошибка: два числа через пробел или 'q' для выхода")
            continue

        x_user_norm = x_user_raw / x_scale
        y_class = best_model.predict_class(x_user_norm, threshold=threshold)
        print(f"Класс сети: {y_class}")

        user_points.append([x_user_raw[0], x_user_raw[1]])

        plot_points_and_boundary(best_model, X_raw, E, x_scale,
                                 user_points=user_points, threshold=threshold)


if __name__ == "__main__":
    main()
```

Результат:

ОБУЧЕНИЕ ОДНОСЛОЙНОЙ НЕЙРОННОЙ СЕТИ

Данные: [[3.0, 6.0], [-3.0, 6.0], [3.0, -6.0], [-3.0, -6.0]]

Целевые значения: [0.0, 0.0, 1.0, 1.0]

lr = 0.01: итоговая MSE = 8.54e-05, эпох = 100

lr = 0.05: итоговая MSE = 8.50e-09, эпох = 40

lr = 0.10: итоговая MSE = 4.00e-09, эпох = 19

lr = 0.20: итоговая MSE = 9.95e-09, эпох = 7

lr = 0.50: итоговая MSE = 3.68e-09, эпох = 13

lr = 0.80: итоговая MSE = 2.97e+03, эпох = 100

ЛУЧШИЙ РЕЗУЛЬТАТ

lr = 0.5

w1 = -0.000023, w2 = -0.499944

b = 0.500007

final MSE = 3.68e-09

ПРОВЕРКА КЛАССИФИКАЦИИ ОБУЧАЮЩИХ ТОЧЕК

x=[3, 6] -> предсказание: 0 (цель: 0)

x=[-3, 6] -> предсказание: 0 (цель: 0)

x=[3, -6] -> предсказание: 1 (цель: 1)

x=[-3, -6] -> предсказание: 1 (цель: 1)

Введите координаты x1 x2 для классификации

Для выхода введите 'q'

x1 x2 > 3 6

Класс сети: 0

x1 x2 > -3 6

Класс сети: 0

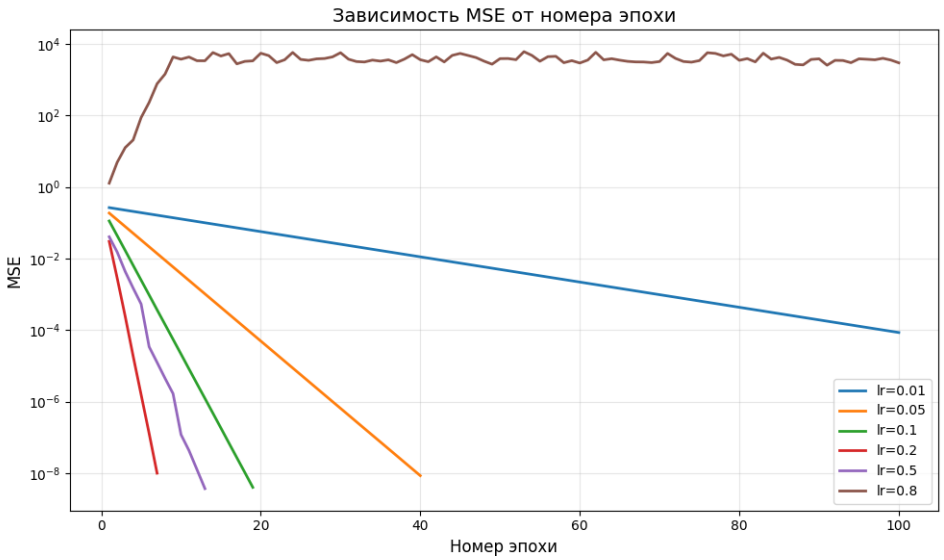
x1 x2 > 3 -6

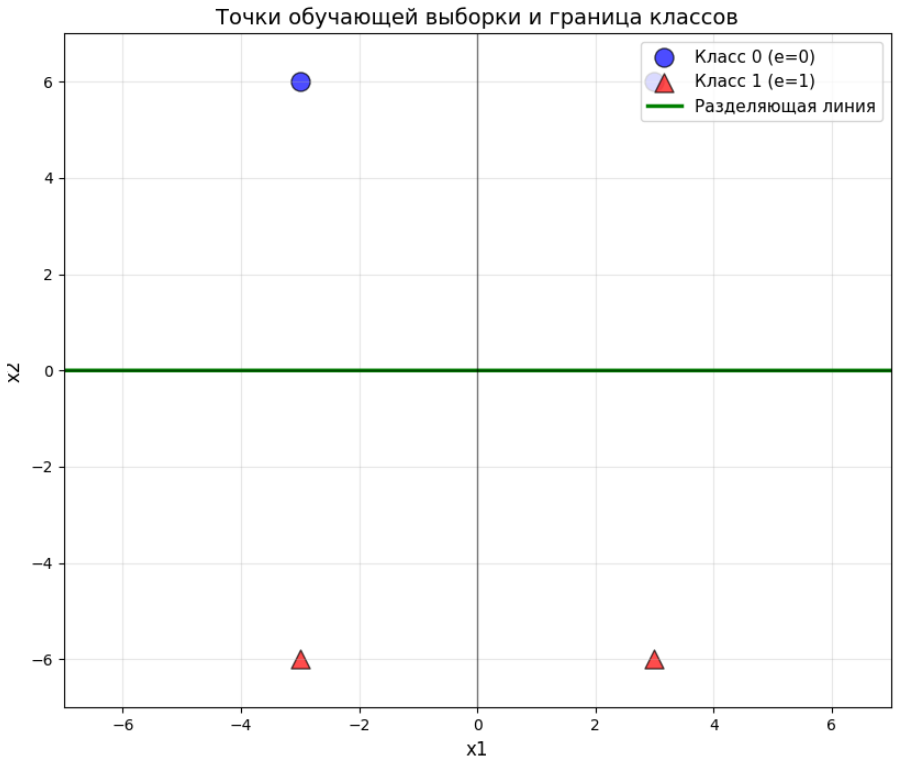
Класс сети: 1

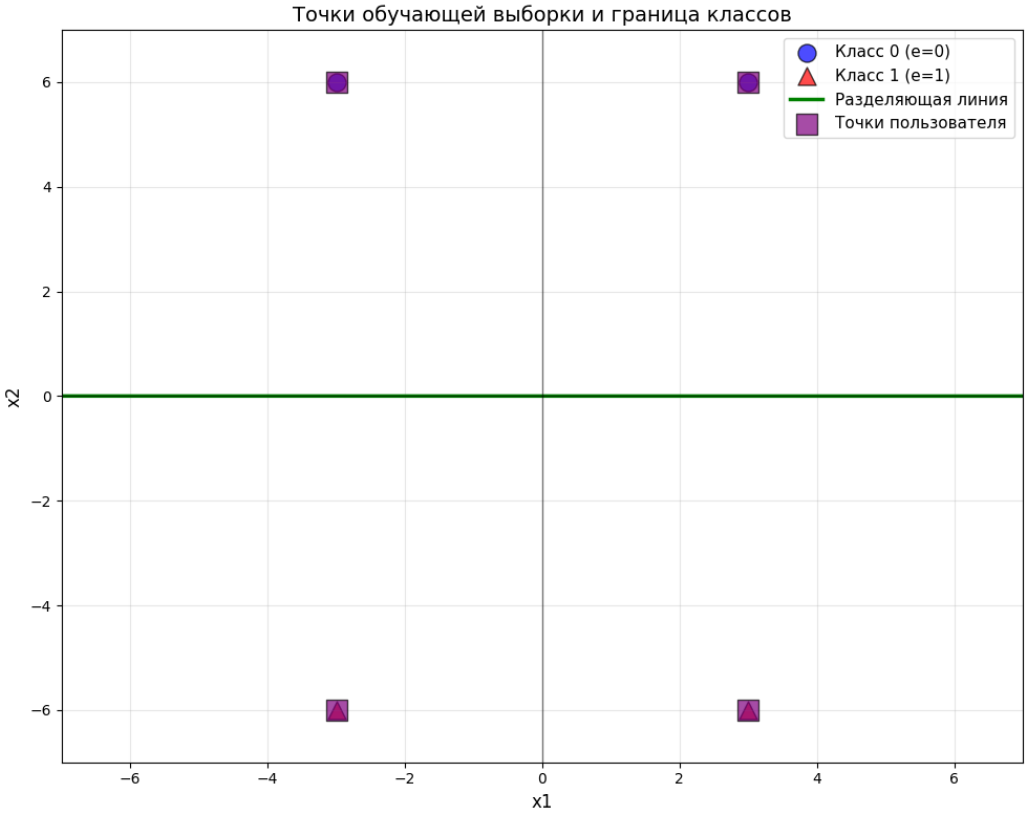
x1 x2 > -3 -6

Класс сети: 1

Рисунки с результатами работы программы







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