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
А.В. Семёнов,
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:
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 ВСТАВИТЬ СВОЙ ДИАПАЗОН, например $-0.5 \leq x1, x2 \leq 1.5$

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)

3.

x_1	x_2	e
3	4	1
-3	4	1
3	-4	0
-3	-4	0

Код программы:

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

X_raw = np.array([
    [ 3.0,  4.0],
    [-3.0,  4.0],
    [ 3.0, -4.0],
    [-3.0, -4.0],
], dtype=float)

E = np.array([1.0, 1.0, 0.0, 0.0], 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()
    ax = fig.add_subplot(111)
    for lr, hist in histories.items():
        ax.plot(range(1, len(hist) + 1), hist, label=f"lr={lr}")
    ax.set_xlabel("Номер эпохи")
    ax.set_ylabel("MSE")
    ax.set_title("Зависимость MSE от номера эпохи")
    ax.grid(True)
    ax.legend()
    fig.tight_layout()
    fig.savefig("mse_plot.png", dpi=200)

def plot_points_and_boundary(model: SingleLayerNet,
                             X_raw: np.ndarray,
                             E: np.ndarray,
                             x_scale: np.ndarray,
                             user_points=None,
                             xlim=(-6, 6),
                             ylim=(-6, 6),
                             threshold: float = 0.5):

    fig = plt.figure()
    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=90, label="Класс 0 (e=0)")
    if len(cls1):
        ax.scatter(cls1[:, 0], cls1[:, 1], marker="^", s=90, label="Класс 1 (e=1)")

    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, label="Разделяющая линия")
    else:
        if abs(a1) > 1e-12:
            x0 = (threshold - b) / a1
            ax.axvline(x0, label="Разделяющая линия")

    if user_points:
        up = np.array(user_points, dtype=float)
        ax.scatter(up[:, 0], up[:, 1], marker="s", s=120, label="Точки пользователя")
```

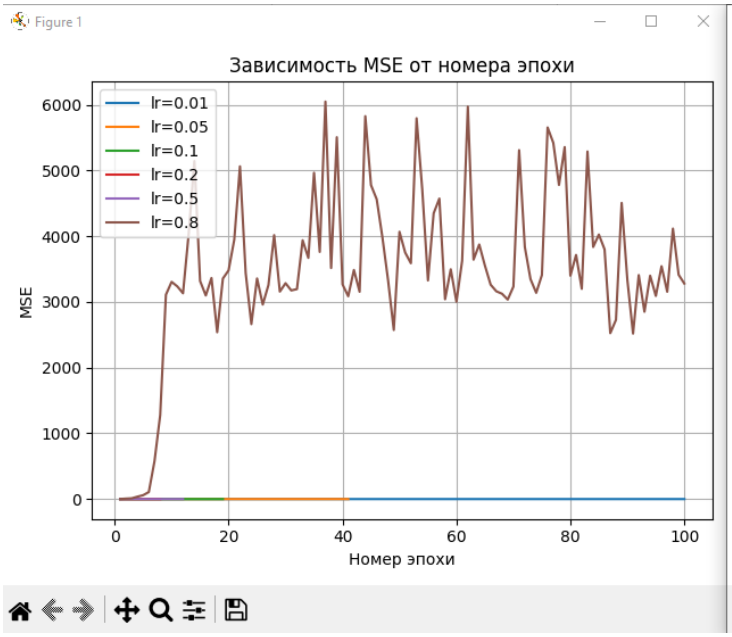
```
    ax.axhline(0, linewidth=1)
    ax.axvline(0, linewidth=1)
    ax.set_xlim(*xlim)
    ax.set_ylim(*ylim)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("Точки обучающей выборки и граница классов")
    ax.grid(True)
    ax.legend()
    fig.tight_layout()
    fig.savefig("decision_boundary.png", dpi=200)

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")

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

    print("ЛУЧШИЙ РЕЗУЛЬТАТ MSE")
    print(f"lr = {best_lr}")
    print(f"w = {best_model.w}, b = {best_model.b}")
    print(f"final MSE = {best_final_mse}\n")

    print("Проверка классификации обучающих точек")
    for x_raw, e in zip(X_raw, E):
        x_norm = x_raw / x_scale
        y = best_model.predict_class(x_norm, threshold=threshold)
        print(f"x={x_raw} -> y={y} (e={int(e)})")
```

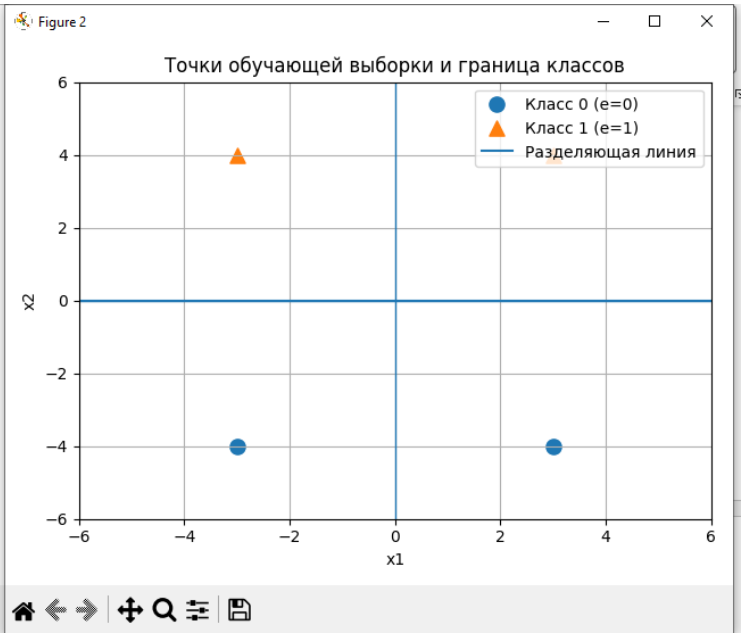
```
    plot_mse_histories(histories)
    plot_points_and_boundary(best_model, X_raw, E, x_scale, user_points=None, threshold=threshold)

    plt.show()

    print("\nРЕЖИМ ФУНКЦИОНИРОВАНИЯ")
    print("Вводи x1 x2. Для выхода: q")

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

        try:
            x1_str, x2_str = s.replace(",", " ").split()
            x_user_raw = np.array([float(x1_str), float(x2_str)], dtype=float)
        except Exception:
            print("два числа через пробел или выход на q.")
            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)
        plt.show()

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

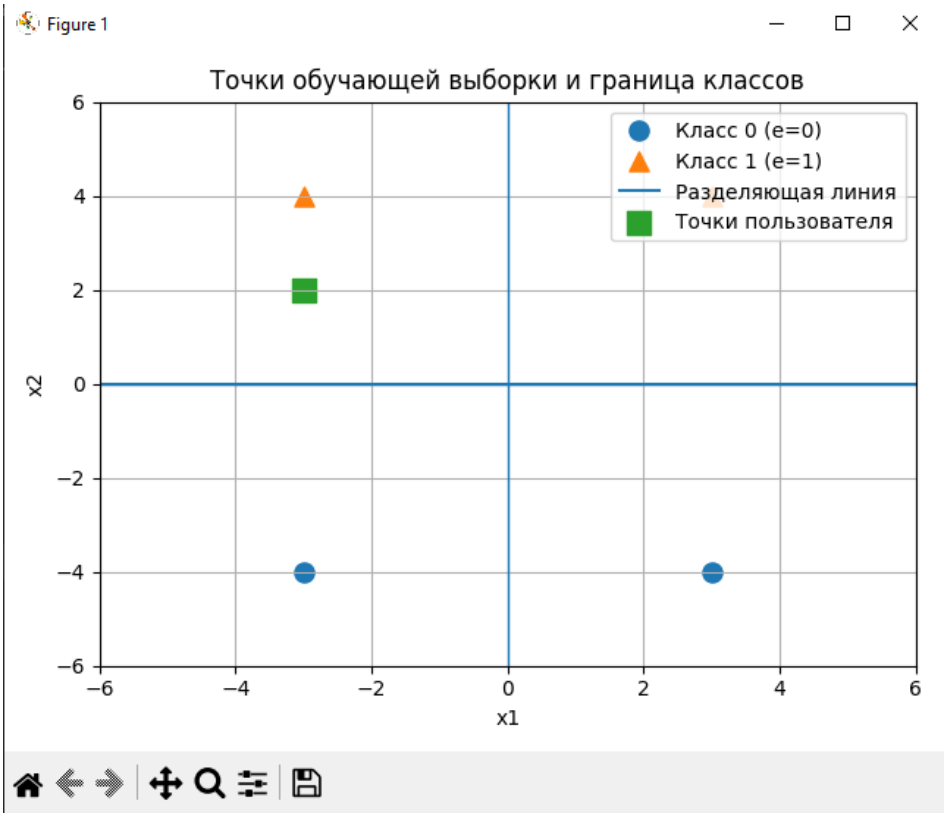
Результат:

```
ЛУЧШИЙ РЕЗУЛЬТАТ MSE
lr = 0.2
w = [2.01751035e-05 4.99976657e-01], b = 0.49998927006985994
final MSE = 1.0670440920480664e-09

Проверка классификации обучающих точек
x=[3. 4.] -> y=1 (e=1)
x=[-3.  4.] -> y=1 (e=1)
x=[ 3. -4.] -> y=0 (e=0)
x=[-3. -4.] -> y=0 (e=0)

РЕЖИМ ФУНКЦИОНИРОВАНИЯ
Вводи x1 x2. Для выхода: q
x1 x2 > -3 2
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).