МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТк

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ26(з)

Выполнил
Т.В.
Данилюк,
студент группы ИИ26

Проверила
К.В. Андренко,
ст. преп. кафедры ИИТ,
«___k_____2026 г.

Брест 2026

Цель работы: изучить алгоритм оптимизации градиентного спуска с использованием адаптивного шага обучения. Реализовать модифицированный персептрон, в котором параметр скорости обучения t вычисляется на основе минимизации квадратичной формы ошибки для каждой итерации. Сравнить скорость сходимости с классическим алгоритмом из Лабораторной работы №1. Вариант сохраняется.

Код программы:

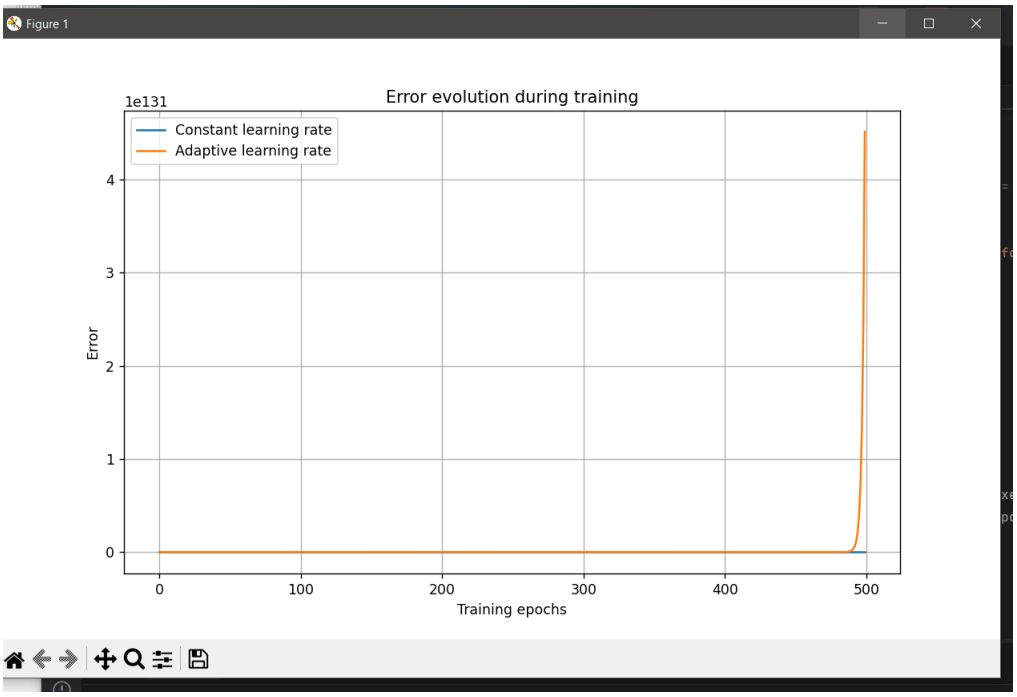
Содержимое в файле main.py

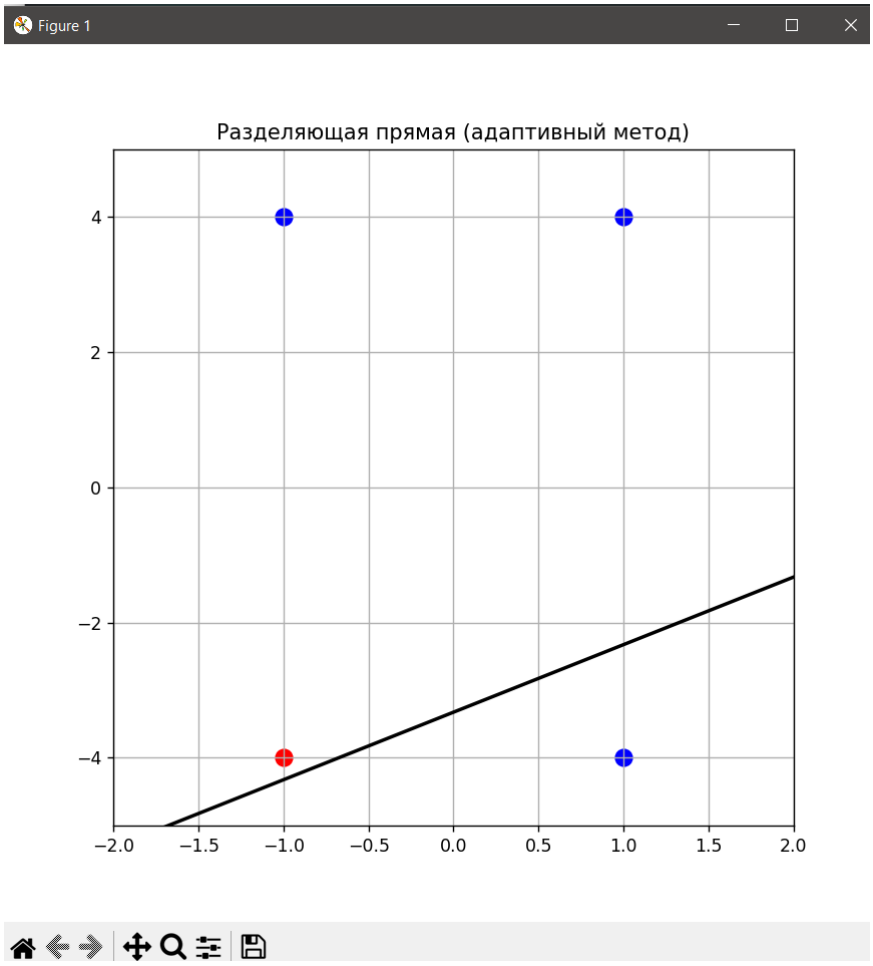
```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
    [1, 4],
    [-1, 4],
    [1, -4],
    [-1, -4]
], dtype=float)

E = np.array([0, 0, 0, 1], dtype=float)

E_e = 0.01
epochs_max = 500


def forward(w, T, x):
    S = np.dot(w, x) - T
    return S


def train_fixed_eta(X, E, eta, E_e, epochs_max):
    w = np.random.randn(X.shape[1])
    T = np.random.randn()
    mse_curve = []

    for epoch in range(epochs_max):
        for x, e in zip(X, E):
            y = forward(w, T, x)
            error = e - y

            # --- исправлено ---
            w += eta * error * x
            T += eta * error

        E_sum = sum((E[i] - forward(w, T, X[i]))**2 for i in range(len(X)))
        mse_curve.append(0.5 * E_sum)

        if mse_curve[-1] <= E_e:
            break

    return w, T, mse_curve


def train_adaptive_eta(X, E, E_e, epochs_max):
    w = np.random.randn(X.shape[1])
```

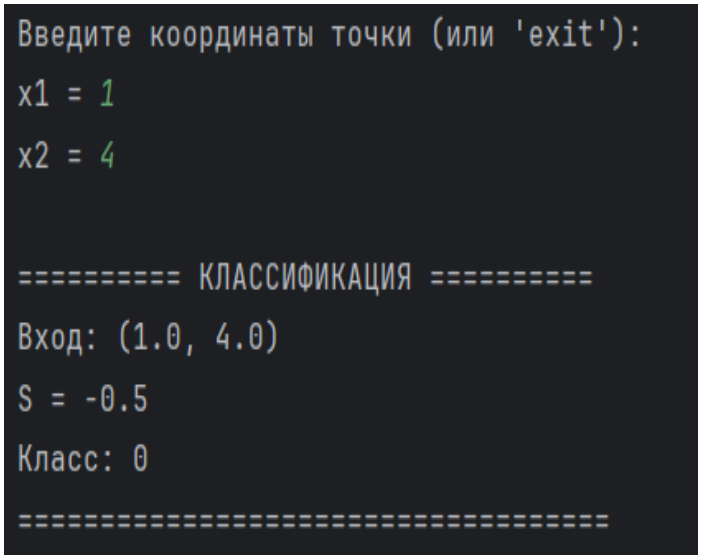
```
    T = np.random.randn()
    mse_curve = []

    for epoch in range(epochs_max):
        for x, e in zip(X, E):
            y = forward(w, T, x)
            error = e - y

            alpha_t = 1 / (1 + np.dot(x, x))

            # --- исправлено ---
            w += alpha_t * error * x
            T += alpha_t * error

        E_sum = sum((E[i] - forward(w, T, X[i]))**2 for i in range(len(X)))
        mse_curve.append(0.5 * E_sum)

        if mse_curve[-1] <= E_e:
            break

    return w, T, mse_curve


# ==== Обучение ====
eta_fixed = 0.01
w_fix, T_fix, curve_fix = train_fixed_eta(X, E, eta_fixed, E_e, epochs_max)
w_ad, T_ad, curve_ad = train_adaptive_eta(X, E, E_e, epochs_max)

print("Фиксированный шаг: эпох =", len(curve_fix))
print("Адаптивный шаг: эпох =", len(curve_ad))
print("Final Es (fixed eta):", curve_fix[-1])
print("Final Es (adaptive):", curve_ad[-1])


# ==== График ошибки ====
plt.figure(figsize=(10, 6))
plt.plot(curve_fix, label="Constant learning rate")
plt.plot(curve_ad, label="Adaptive learning rate")
plt.xlabel("Training epochs")
plt.ylabel("Error")
plt.title("Error evolution during training")
plt.grid()
plt.legend()
plt.show()


# ==== Разделяющая прямая (адаптивный метод) ====
plt.figure(figsize=(7, 7))

for i in range(len(X)):
    color = "red" if E[i] == 1 else "blue"
    plt.scatter(X[i, 0], X[i, 1], color=color, s=90)

x_line = np.linspace(-2, 2, 300)
y_line = (T_ad - w_ad[0] * x_line) / w_ad[1]
plt.plot(x_line, y_line, 'k', linewidth=2)

plt.xlim(-2, 2)
```

```
plt.ylim(-5, 5)
plt.grid()
plt.title("Разделяющая прямая (адаптивный метод)")
plt.show()


# ==== Классификация новой точки ====
def classify_point(x1, x2):
    S = w_ad[0] * x1 + w_ad[1] * x2 - T_ad
    y = 1 if S > 0 else 0

    print("\n========== КЛАССИФИКАЦИЯ ==========")
    print(f"Вход: ({x1}, {x2})")
    print(f"S = {S}")
    print(f"Класс: {y}")
    print("====================================\n")

    plt.figure(figsize=(7, 7))

    for i in range(len(X)):
        color = "red" if E[i] == 1 else "blue"
        plt.scatter(X[i, 0], X[i, 1], color=color, s=90)

    plt.plot(x_line, y_line, 'k')
    plt.scatter(x1, x2, color="green", s=150, marker="x")

    plt.xlim(-2, 2)
    plt.ylim(-5, 5)
    plt.grid()
    plt.title("Классификация пользовательской точки")
    plt.show()


while True:
    print("Введите координаты точки (или 'exit'):")
    x1 = input("x1 = ")

    if x1 == "exit":
        break

    x2 = input("x2 = ")
    if x2 == "exit":
        break

    try:
        classify_point(float(x1), float(x2))
    except:
        print("Ошибка: нужно вводить числа!")
```

Результата работы программы:







Вывод: В ходе лабораторной работы была реализована и исследована работа линейного нейрона для задачи бинарной классификации двухмерных входных данных. Проведено обучение с фиксированным и адаптивным шагом обучения.