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
А.В. Семёнов,
студент группы ИИ-26

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

Задачи лабораторной работы:

1. Модифицировать алгоритм последовательного обучения (из Лаб №1) таким образом, чтобы на каждой итерации t значение α вычислялось автоматически на основе текущего входного вектора x по формул (см раздел 2.9).
2. Применить вычисленный α(t) для обновления весов ij и порогов Tj согласно дельта-правилу.
3. Используя данные своего варианта, провести два эксперимента:

- Обучение с фиксированным шагом (например, α=0.1 или α=1p).
- Обучение с адаптивным шагом по Теореме 2.1 (формула 2.36).

Критерий остановки в обоих случаях – достижение заданной суммарной ошибки Es≤Ee.

4. Построить графики обучения Es(p), где p – номер эпохи, для обоих экспериментов на одних осях координат.

5. Выполнить графическую визуализацию разделяющей линии для адаптивного метода.

6. Реализовать режим функционирования сети:

- пользователь задаёт произвольный входной вектор,
- сеть вычисляет выходной класс,
- соответствующая точка отображается на графике,

7. Написать вывод по выполненной работе. Оценить, насколько адаптивный шаг сокращает количество эпох обучения по сравнению с фиксированным.

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)

def sse(y, e):
    y = np.asarray(y, dtype=float).reshape(-1)
    e = np.asarray(e, dtype=float).reshape(-1)
    return float(np.sum((y - e) ** 2))

class SingleLayerNet:

    def __init__(self, seed=42, w_clip=50.0):
        rng = np.random.default_rng(seed)
        self.w = rng.uniform(-0.5, 0.5, size=(2,))
        self.T = 0.0
        self.w_clip = float(w_clip)

    def forward(self, x):
        return float(np.dot(self.w, x) - self.T)

    def predict_class(self, x, threshold=0.5):
        return 1 if self.forward(x) >= threshold else 0

    def update_delta_rule(self, x, e, alpha):

        y = self.forward(x)
        err = (y - e)

        self.w = self.w - alpha * err * x
        self.T = self.T + alpha * err

        self.w = np.clip(self.w, -self.w_clip, self.w_clip)
        self.T = float(np.clip(self.T, -self.w_clip, self.w_clip))


def alpha_adaptive(x):

    return 1.0 / (1.0 + float(np.sum(x ** 2)))


def train_sequential(X, E, *, mode="fixed", alpha_fixed=0.1, Ee=1e-6, max_epochs=2000,
shuffle=True, seed=123):

    model = SingleLayerNet(seed=42, w_clip=50.0)
```

```
    rng = np.random.default_rng(seed)

    hist_Es = []
    n = X.shape[0]

    for ep in range(max_epochs):
        idx = np.arange(n)
        if shuffle:
            rng.shuffle(idx)

        for i in idx:
            x = X[i]
            e = E[i]

            if mode == "fixed":
                alpha = float(alpha_fixed)
            elif mode == "adaptive":
                alpha = alpha_adaptive(x)
            else:
                raise ValueError("mode must be 'fixed' or 'adaptive'")

            model.update_delta_rule(x, e, alpha)

        y_all = np.array([model.forward(x) for x in X], dtype=float)
        Es = sse(y_all, E)
        hist_Es.append(Es)

        if Es <= Ee:
            break

    return model, np.array(hist_Es, dtype=float)


def plot_learning(hist_fixed, hist_adapt):
    plt.figure()
    plt.plot(np.arange(1, len(hist_fixed) + 1), hist_fixed, label="фиксированный шаг")
    plt.plot(np.arange(1, len(hist_adapt) + 1), hist_adapt, label="адаптивный шаг")
    plt.xlabel("номер эпохи p")
    plt.ylabel("Es(p) = Σ(y-e)^2")
    plt.title("графики обучения Es(p) на одних осях")
    plt.grid(True)
    plt.legend()


def plot_points_and_boundary(model, X_raw, E, x_scale, threshold=0.5, user_points=None):

    plt.figure()
```

```
    c1 = X_raw[E == 1]
    c0 = X_raw[E == 0]
    plt.scatter(c1[:, 0], c1[:, 1], marker="o", label="класс 1 (e=1)")
    plt.scatter(c0[:, 0], c0[:, 1], marker="s", label="класс 0 (e=0)")

    if user_points:
        up = np.array(user_points, dtype=float)
        plt.scatter(up[:, 0], up[:, 1], marker="x", label="введённые точки")

    w1, w2 = model.w
    T = model.T

    x1_min, x1_max = X_raw[:, 0].min() - 2, X_raw[:, 0].max() + 2
    xs = np.linspace(x1_min, x1_max, 200)

    if abs(w2) < 1e-12:
        x_const = ((T + threshold) * x_scale) / w1 if abs(w1) > 1e-12 else 0.0
        plt.axvline(x=x_const, linestyle="--", label="разделяющая линия")
    else:
        ys = (((T + threshold) * x_scale) - w1 * xs) / w2
        plt.plot(xs, ys, linestyle="--", label="разделяющая линия")

    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title("разделяющая линия (адоптивный метод)")
    plt.grid(True)
    plt.legend()


def main():
    x_scale = float(np.max(np.abs(X_raw)))  # 4
    X = X_raw / x_scale

    threshold = 0.5

    alpha_fixed = 0.1
    Ee = 1e-6
    max_epochs = 2000

    model_fixed, hist_fixed = train_sequential(
        X, E, mode="fixed", alpha_fixed=alpha_fixed, Ee=Ee, max_epochs=max_epochs
    )

    model_adapt, hist_adapt = train_sequential(
        X, E, mode="adaptive", alpha_fixed=alpha_fixed, Ee=Ee, max_epochs=max_epochs
    )

    p_fixed = len(hist_fixed)
```

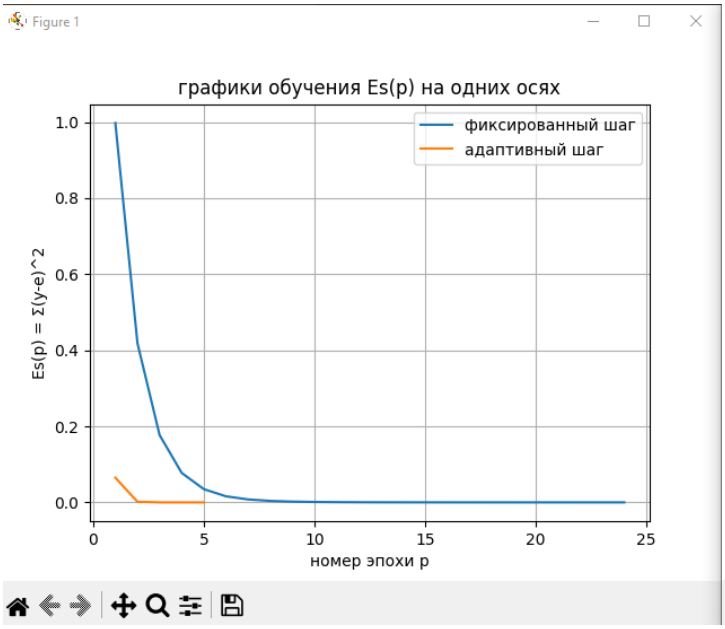
```
p_adapt = len(hist_adapt)
accel = (p_fixed / p_adapt) if p_adapt > 0 else float("inf")
saved_pct = (p_fixed - p_adapt) / p_fixed * 100.0 if p_fixed > 0 else 0.0

print("фиксир шаг")
print(f"Эпох: {p_fixed} | финальный Es: {hist_fixed[-1]:.6e}")
for x_r, x, e in zip(X_raw, X, E):
    print(f"x={x_r} -> class={model_fixed.predict_class(x, threshold)} (e={int(e)})")

print("\nодаптир шаг")
print(f"Эпох: {p_adapt} | финальный Es: {hist_adapt[-1]:.6e}")
for x_r, x, e in zip(X_raw, X, E):
    print(f"x={x_r} -> class={model_adapt.predict_class(x, threshold)} (e={int(e)})")

print("\nоценка ускарения")
print(f"сокращение эпох: {p_fixed} -> {p_adapt}")
print(f"ускарение: {accel:.2f} раза")
print(f"экономия эпох: {saved_pct:.1f}%")

plot_learning(hist_fixed, hist_adapt)

plot_points_and_boundary(model_adapt, X_raw, E, x_scale, threshold=threshold)

plt.show()

print("\nрежим функцианирования")
print("вводи x1 x2 . Для выхода жмакай 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("ашибка , введи два числа через пробел или q.")
        continue

    x_user = x_user_raw / x_scale
    y_class = model_adapt.predict_class(x_user, threshold=threshold)
    print(f"класс сети: {y_class}")

    user_points.append([x_user_raw[0], x_user_raw[1]])
    plot_points_and_boundary(model_adapt, X_raw, E, x_scale, threshold=threshold,
user_points=user_points)
```

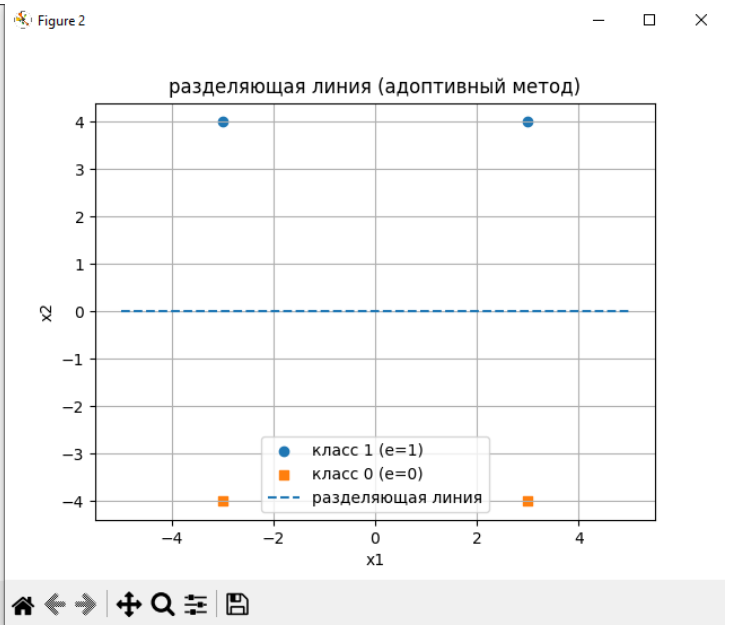
```
    plt.show()


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

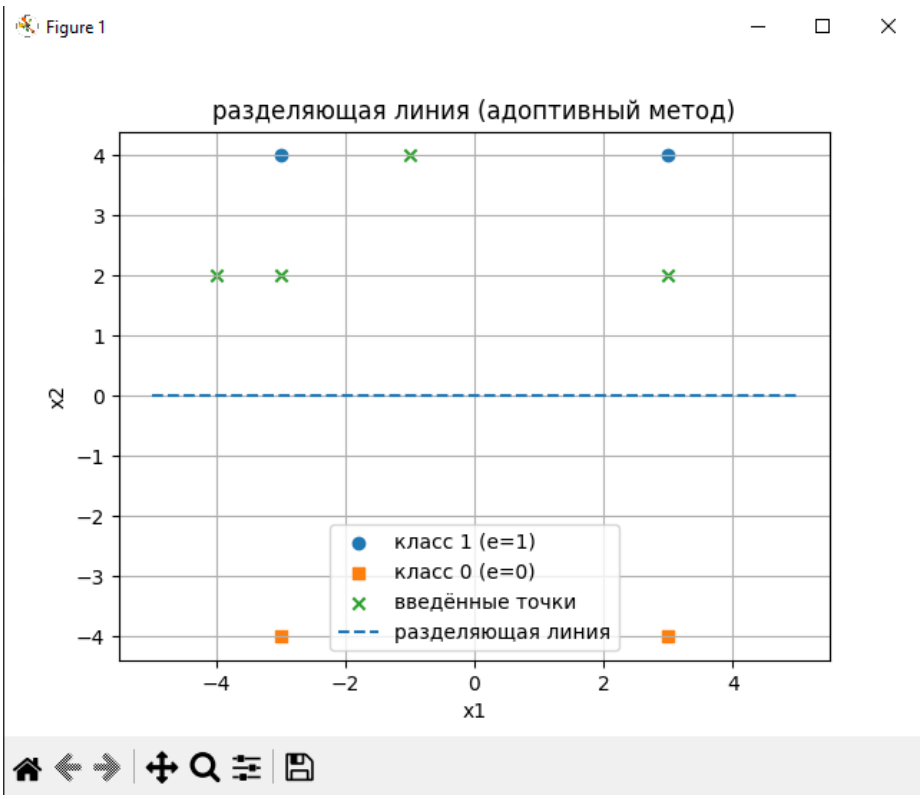
Результат:

```
C:\Users\сеня\lab2\.venv\Scripts\python.exe C:\Users\сеня\lab2\.venv\lab2.py
фиксир шаг
Эпох: 24 | финальный Es: 6.739932e-07
x=[3. 4.] -> class=1 (e=1)
x=[-3.  4.] -> class=1 (e=1)
x=[ 3. -4.] -> class=0 (e=0)
x=[-3. -4.] -> class=0 (e=0)

одаптир шаг
Эпох: 5 | финальный Es: 3.108042e-08
x=[3. 4.] -> class=1 (e=1)
x=[-3.  4.] -> class=1 (e=1)
x=[ 3. -4.] -> class=0 (e=0)
x=[-3. -4.] -> class=0 (e=0)

оценка ускарения
сокращение эпох: 24 -> 5
ускарение: 4.80 раза
экономия эпох: 79.2%

режим функцианирования
вводи x1 x2 . Для выхода жмакай q
x1 x2 > -3 2
класс сети: 1
x1 x2 > -4 2
класс сети: 1
x1 x2 > -1 4
класс сети: 1
x1 x2 > 3 2
класс сети: 1
```

Рисунки с результатами работы программы







Вывод: изучил алгоритм оптимизации градиентного спуска с использованием адаптивного шага обучения. Реализовал модифицированный персептрон, в котором параметр скорости обучения t вычисляется на основе минимизации квадратичной формы ошибки для каждой итерации. Сравнил скорость сходимости с классическим алгоритмом.