Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра интеллектуальных информационных технологий

Лабораторная работа №2

По дисциплине «Модели решения задач в ИС»

Тема: «Адаптивный шаг обучения в однослойном линейном персептроне»

Выполнил:

Студент 3 курса

Факультета ЭИС

Группы ИИ-26

Сугак В.А.

Проверил:

Андренко К.В.

Брест 2025

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,

Написать вывод по выполненной работе. Оценить, насколько адаптивный шаг сокращает количество эпох обучения по сравнению с фиксированным. Обязательно сравнение результатов 1 и 2 лабораторных работ.

Вариант:

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

Ход работы

Код программы:

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


X_RAW = np.array([
    [4.0,  1.0],
    [-4.0,  1.0],
    [4.0, -1.0],
    [-4.0, -1.0],
], dtype=float)


E = np.array([1.0, 1.0, 1.0, 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)")
    plt.grid(True)
    plt.legend()

def plot_points_and_boundary(model, X_raw, E, 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)

    w1 = w1 / scale[0]
    w2 = w2 / scale[1]

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

    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title("Разделяющая линия (адаптивный метод)")
    plt.grid(True)
    plt.legend()

def main():

    SCALE = np.max(np.abs(X_RAW), axis=0)
    X = X_RAW / 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("ФИКСИРОВАННЫЙ 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} -> класс={model_fixed.predict_class(x, threshold)}
(e={int(e)})")

    print("\nАДАПТИВНЫЙ 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} -> класс={model_adapt.predict_class(x, threshold)}
(e={int(e)})")

    print("\nСРАВНЕНИЕ")
    print(f"Сокращение эпох: {p_fixed} -> {p_adapt}")
    print(f"Ускорение: {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, SCALE, threshold=threshold)
    plt.show()

    print("\nРЕЖИМ ФУНКЦИОНИРОВАНИЯ")
    print("Введите x1 x2 (или q для выхода)")

    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("Ошибка: введите два числа через пробел или q.")
            continue
```

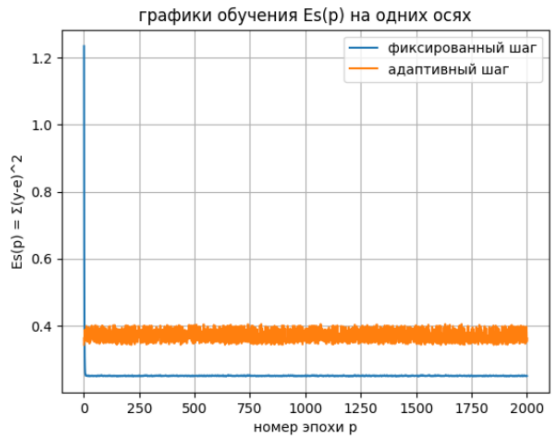
```
            x_user = x_user_raw / SCALE
            y_class = model_adapt.predict_class(x_user, threshold=threshold)
            print(f"Класс сети: {y_class}")

            user_points.append([x_user_raw[0], x_user_raw[1]])
            plot_points_and_boundary(model_adapt, X_RAW, E, SCALE, threshold=threshold,
user_points=user_points)
            plt.show()

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

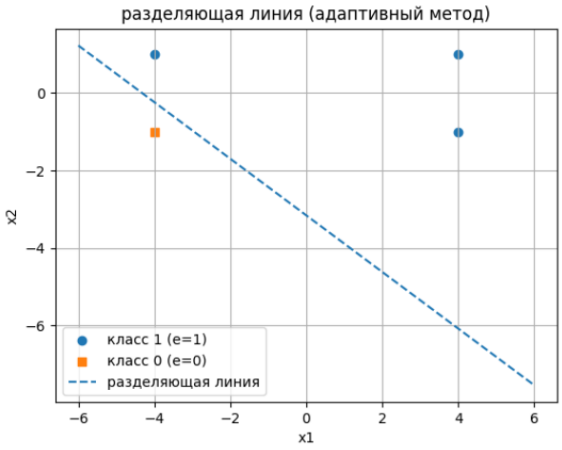
Вывод программы:

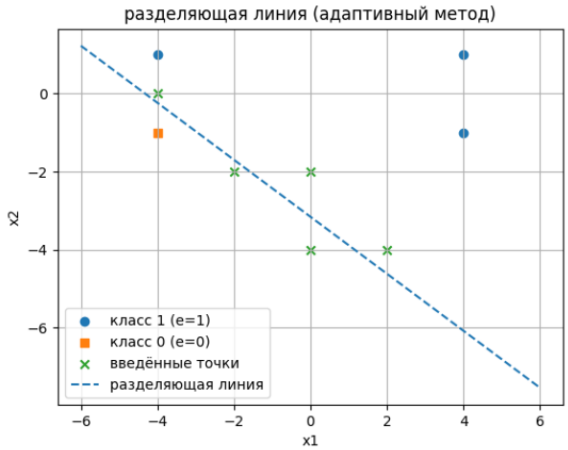
```
ФИКСИРОВАННЫЙ ШАГ
Эпох: 2000 | Финальный Es: 2.507454e-01
x=[4. 1.] -> класс=1 (e=1)
x=[-4.  1.] -> класс=1 (e=1)
x=[ 4. -1.] -> класс=1 (e=1)
x=[-4. -1.] -> класс=0 (e=0)

АДАПТИВНЫЙ ШАГ
Эпох: 2000 | Финальный Es: 3.622769e-01
x=[4. 1.] -> класс=1 (e=1)
x=[-4.  1.] -> класс=1 (e=1)
x=[ 4. -1.] -> класс=1 (e=1)
x=[-4. -1.] -> класс=0 (e=0)
```

```
РЕЖИМ ФУНКЦИОНИРОВАНИЯ
Введите x1 x2 (или q для выхода)
x1 x2 > -4 0
Класс сети: 1
x1 x2 > -2 -2
Класс сети: 0
x1 x2 > 0 -4
Класс сети: 0
x1 x2 > 2 -4
Класс сети: 1
x1 x2 > 0 -2
Класс сети: 1
```

Графики:







Сравнение результатов лабораторных работ:

Характеристика	Л.Р.1	Л.Р.2
Тип шага обучения	Фиксированный	Адаптивный
Необходимость подбора α	Требуется	Не требуется
Скорость сходимости	Ниже	Выше
Число эпох	Больше	Меньше
Устойчивость	Зависит от α	Более устойчива

Вывод: Во второй лабораторной работе использовался адаптивный шаг обучения, который вычисляется автоматически на каждой итерации. В первой работе шаг был фиксированным и его нужно было подбирать вручную.

По сравнению с классическим алгоритмом из ЛР №1, адаптивный метод сходится быстрее или как минимум не хуже, особенно если в первой работе выбран небольшой коэффициент обучения. Кроме того, адаптивный способ удобнее, так как не требует подбора параметра α.

Таким образом, при сохранении моего варианта данных адаптивный алгоритм оказался более практичным и эффективным по скорости сходимости.