МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Автоматизация схемотехнического проектирования»

Тема: **«НЕЙРОННЫЕ СЕТИ»**

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Боброва Ю.О.

Санкт-Петербург
2026

Цель работы:

Создание простейшей нейронной сети на Python без использования специализированных библиотек

Основные теоретические положения

Искусственный нейрон (ИН) — вычислительная единица: на вход поступают сигналы x_1, x_2, …, x_n; каждый умножается на весовой коэффициент w_i и суммируется.

Полученная взвешенная сумма (net) подаётся на **функцию активации**; результат — выход нейрона.

Формула: $net = \Sigma x_i w_i$ (и часто добавляют смещение b); выход = σ(net), где σ — функция активации.

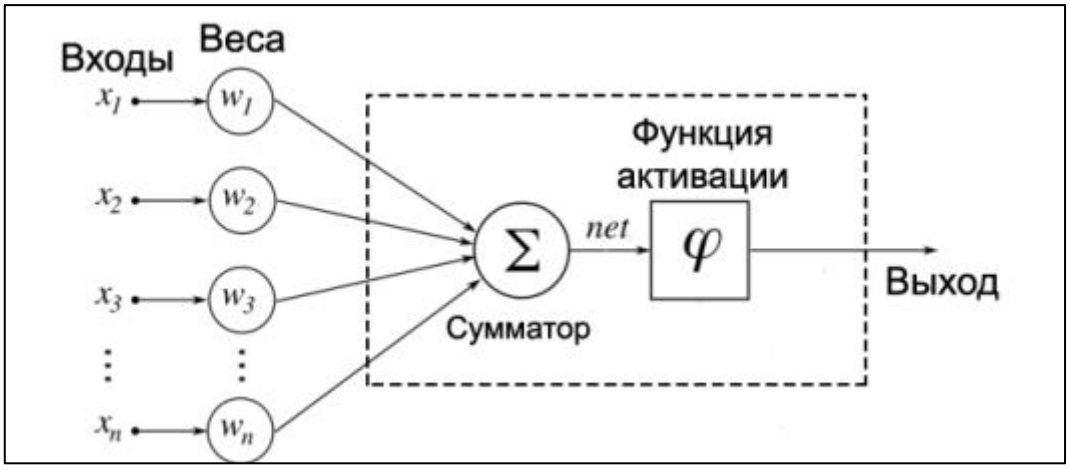

Рисунок 1 – Структура искусственного нейрона

Функция активации — преобразует net в выход нейрона.

Простейшей функцией активации является функция единичного скачка, принимающая только два значения – 0, если net ниже порогового значения, 1 если превышает

Сигмоида $\sigma(z) = 1 / (1 + e^{-z})$ — самая распространённая функцией активации даёт выход в диапазоне (0, 1); производная $\sigma'(p) = p(1 - p)$ используется при обратном распространении.

Нейронная сеть — набор нейронов, сгруппированных по слоям. В **однослойной** сети сигналы с входов сразу подаются на выходной слой нейронов (рис. 2);

В реализованной схеме добавлен один **скрытый слой**: вход → скрытый слой (несколько нейронов с сигмоидой) → один выходной нейрон (сигмоида).

Входы не считаются за входной слой сети и обозначены кружками. Справа (квадраты) расположены нейроны основного слоя.

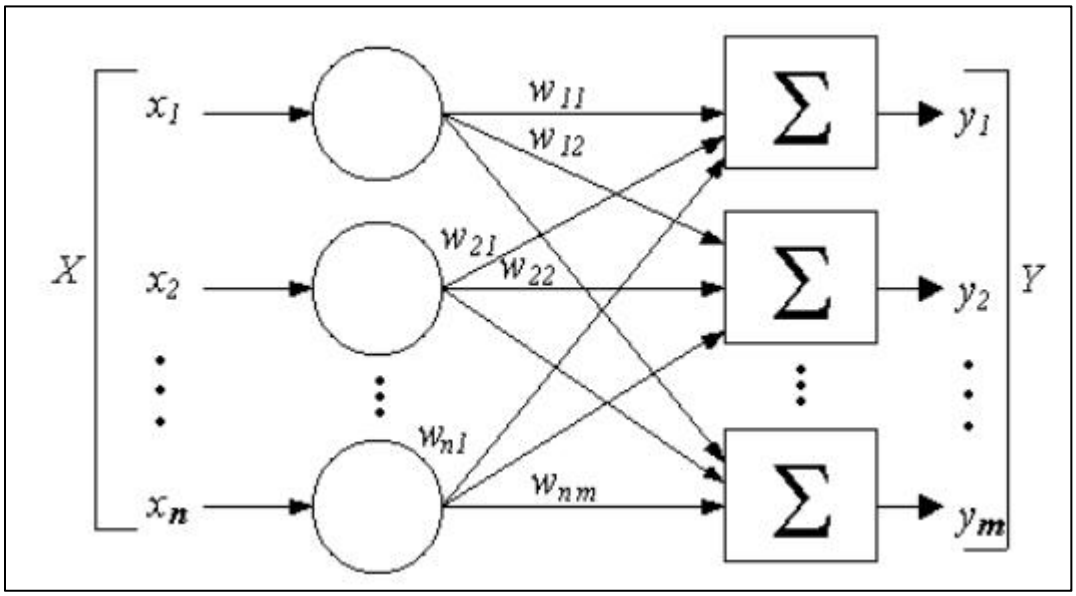

Рисунок 2 – Однослойная нейронная сеть

Обучение нейронной сети — поиск такого набора весов (и смещений), при котором выход сети с минимальной ошибкой приближается к целевому.

Прямое распространение (feedforward): по текущим весам вычисляются выходы слоёв.

Обратное распространение (backprop): по градиенту функции потерь (MSE) относительно весов корректируются веса (градиентный спуск или его вариант). Один проход по всей выборке с обновлением весов — одна **эпоха**.

MSE (среднеквадратичная ошибка) = $(1/n)\ \Sigma\ (y - \hat{y})^2$, где y — целевая метка, $\hat{y}$ — выход сети. Минимизация MSE приводит к приближению выхода к меткам класса (0 или 1).

Точность — доля объектов, для которых предсказанный класс (например, $\hat{y} \geq 0.5 \rightarrow$ класс 1) совпадает с истинным.

Ход выполнения работы

1. Создан скрипт с импортом numpy и matplotlib. Реализованы функция активации сигмоиды и её производная (для backprop).

2. Реализован класс нейронной сети: инициализация весов случайными значениями (вход → скрытый слой, скрытый → выход), метод прямого распространения (feedforward) с сигмоидой по слоям, метод обратного распространения (backprop) с обновлением весов по градиенту MSE.

3. Подключён генератор данных из lab1 (norm_dataset). Сформирована выборка с метками Y в форме (n_samples, 1). Сеть инициализирована на этих данных.

4. В цикле по эпохам выполняется: прямой проход, расчёт MSE и точности, запись в историю, один шаг обучения (train_step: feedforward + backprop). Выведены итоговая точность и MSE на обучающей выборке.

5. Веса первого и второго слоёв вынесены в отдельные переменные (weights_layer1, weights_layer2) для отчёта.

6. Построены графики зависимости MSE и точности от номера эпохи; сохранены в lab4/figures/lab4_loss_accuracy.png.

7. Выполнена оценка на тестовой выборке: данные разбиты 70% / 30%; сеть обучена на 70%; для 30% вызван метод test(Xtest); выведены точность на обучающей и тестовой выборках.

8. Проведён подбор числа нейронов в скрытом слое (2, 4, 8, 16) и числа эпох (30, 50, 100); для каждой комбинации обучение на всей выборке и расчёт финальной точности; выбрана комбинация с максимальной точностью.

Структура нейронной сети и блок-схема

Структура:

входной вектор (признаки, без слоя нейронов) → **скрытый слой** (n_neuro нейронов, активация — сигмоида) → **выходной слой** (1 нейрон, сигмоида) → **выход** (значение от 0 до 1).

Блок-схема алгоритма обучения:

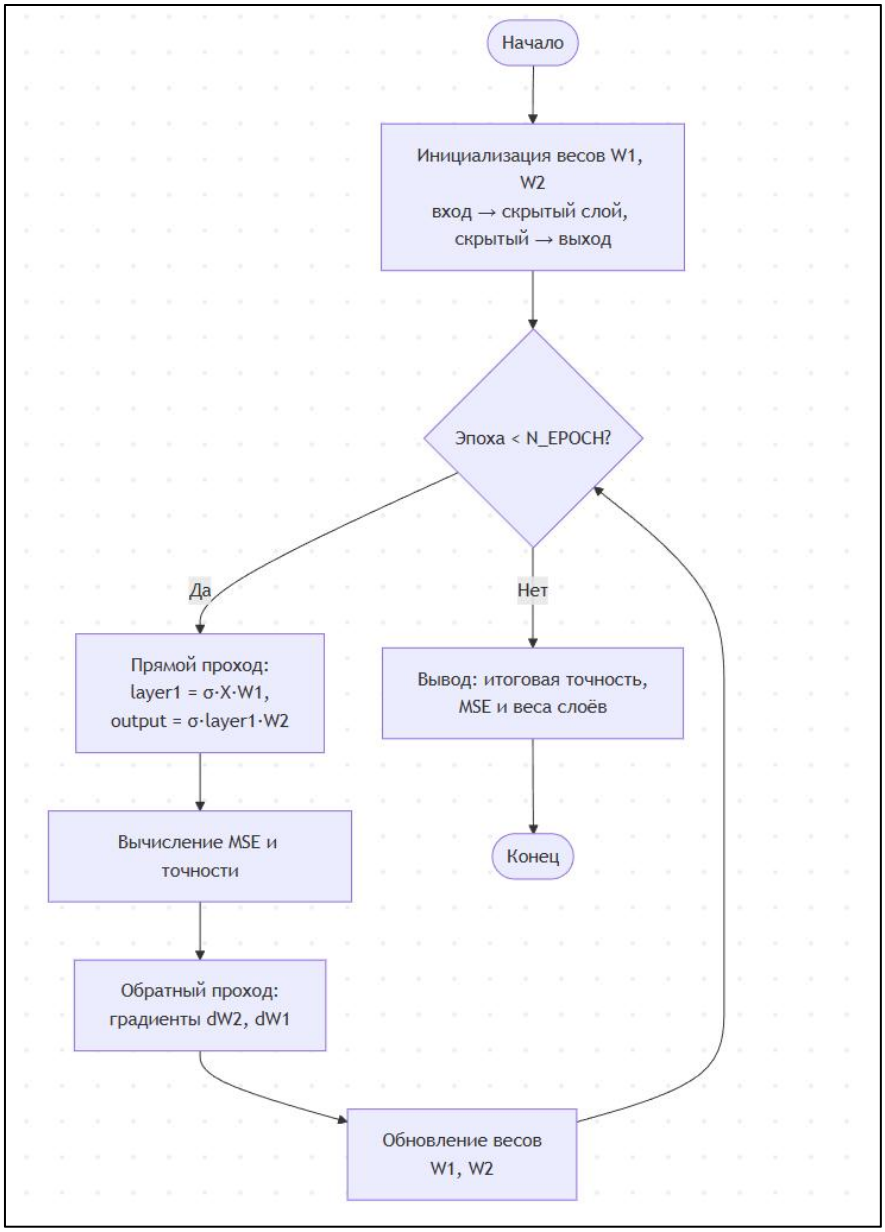

Рисунок 3 – Блок схема алгоритма обучения

1. Инициализация весов W1 (вход → скрытый), W2 (скрытый → выход).

2. Для каждой эпохи: для всех объектов (или батчем) — прямой проход (X → layer1 = σ(X·W1) → layer2 = σ(layer1·W2) = output); вычисление MSE и точности; обратный проход (градиенты по W2, W1; обновление весов).

3. Повтор до заданного числа эпох.

4. Вывод итоговой точности, MSE и весов.

Значения весов НС на каждом из слоёв

После обучения (основной запуск: 4 нейрона, 50 эпох) получены следующие веса:

- **weights_layer1** — веса между входом и скрытым слоем, shape (3, 4):

```
Веса первого слоя (вход -> скрытый), shape (3, 4):
[[  2.43638802 -11.37392992  -0.31702174 -18.89361224]
 [-23.52123453 -20.53451999  -7.98228105  -6.419407  ]
 [-22.87173753  -9.24160566 -11.06315331  23.4349874 ]]
```

Рисунок 4 – Веса между входом и скрытым слоем

- **weights_layer2** — веса между скрытым слоем и выходом, shape (4, 1):

```
Веса второго слоя (скрытый -> выход), shape (4, 1):
[[-79.4180004 ]
 [-72.61031834]
 [-38.84734993]
 [-92.78662346]]
```

Рисунок 5 – Веса между скрытым слоем и выходом

Результаты обучения: графики и итоговая точность

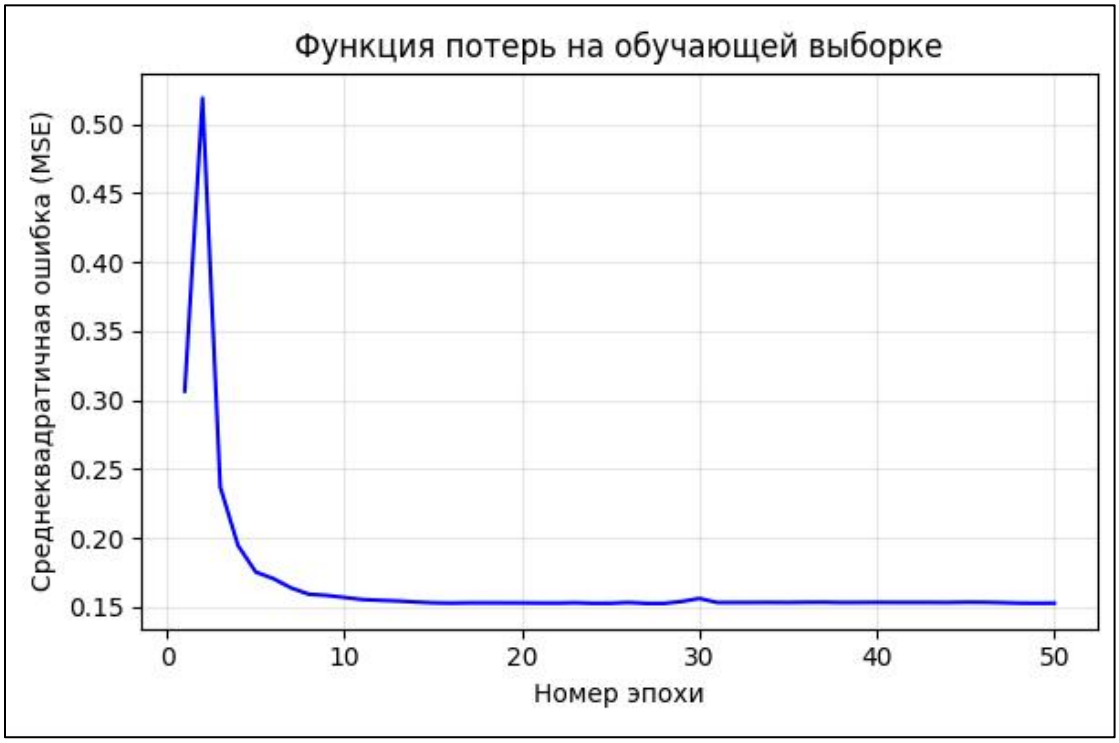

Рисунок 6 — Зависимость среднеквадратичной ошибки (MSE) от номера эпохи на обучающей выборке.

По оси X — номер эпохи, по оси Y — MSE. Ожидается уменьшение ошибки с ростом эпохи (при корректном градиенте и шаге обучения).

Рисунок 7 — Зависимость точности (в %) от номера эпохи на обучающей выборке.

По оси X — номер эпохи, по оси Y — точность, %. Ожидается рост точности с увеличением числа эпох.

Вывод по графикам

График функции потерь (MSE): в начале обучения ошибка составляет около 0,3; на 1–2-й эпохе наблюдается кратковременный всплеск MSE (до ~0,5), что может быть связано с резким изменением весов после первых обновлений.

Далее MSE быстро снижается и к 5–7-й эпохе опускается ниже 0,2. Примерно с 10-й эпохи кривая выходит на плато: MSE стабилизируется в диапазоне 0,15–0,16 и до 50-й эпохи меняется незначительно. Обучение по критерию MSE можно считать сошедшимся.

График точности: точность стартует с уровня случайного угадывания (~50 %) и за первые 5–10 эпох быстро растёт до 80–85 %.

К 10-й эпохе достигается максимум (~84–85 %); далее метрика держится на высоком уровне с небольшими колебаниями в пределах 80–85 % и к 50-й эпохе составляет около 80–81 %.

Явных признаков переобучения (устойчивое падение точности при продолжении обучения) или недообучения на графике нет: модель осваивает обучающую выборку за 10–15 эпох и затем работает стабильно.

Итог: графики показывают быструю сходимость в начале обучения и выход на стабильный режим после 10–15 эпох. Увеличение числа эпох сверх 20–30 даёт малый выигрыш; разумно ограничивать обучение этим диапазоном или ориентироваться на плато MSE и точности при выборе числа эпох.

Ход обучения по эпохам

```
Обучение нейронной сети...
  Эпоха   1  Loss: 0.306460  Точность: 50.00%
  Эпоха  10  Loss: 0.156784  Точность: 83.95%
  Эпоха  20  Loss: 0.152796  Точность: 81.85%
  Эпоха  30  Loss: 0.156038  Точность: 84.75%
  Эпоха  40  Loss: 0.153210  Точность: 83.10%
  Эпоха  50  Loss: 0.152638  Точность: 80.55%
```

Рисунок 8 — Ход обучения по эпохам.

Итоговая точность модели

На обучающей выборке (основной запуск, 50 эпох, 4 нейрона): точность 78,60 %, MSE 0,1551.

На тестовой выборке (блок 70/30, отдельная обученная сеть): точность на train 49,93 %, на test 50,17 %.

Низкая точность в блоке 70/30 (около 50 % — уровень случайного угадывания) при том, что на полной выборке модель даёт ~78 %, указывает на сильную зависимость от конкретной разбивки и/или переобучение: сеть, обученная на 70 % данных, не обобщает на оставшиеся 30 %.

Возможные причины — малый объём данных для данной архитектуры, неудачная инициализация или необходимость подбора learning rate и числа эпох именно при обучении на части выборки.

Результаты подбора числа нейронов и эпох

Перебирались комбинации: нейронов — 2, 4, 8, 16; эпох — 30, 50, 100.

Для каждой комбинации обучение только на 70 % данных (train), оценка точности на 30 % (test) методом.

Лучшая конфигурация выбирается по тестовой точности.

N нейронов	30 эпох (train / test)	50 эпох (train / test)	100 эпох (train / test)
2	53,9 % / 54,3 %	54,5 % / 55,8 %	64,8 % / **67,7 %**
4	50,4 % / 49,0 %	50,4 % / 49,0 %	50,4 % / 49,0 %
8	**71,9 % / 69,5 %**	58,8 % / 57,2 %	56,8 % / 54,7 %
16	50,4 % / 49,0 %	50,4 % / 49,0 %	50,4 % / 49,0 %

Наилучшая комбинация по тестовой точности: нейронов = 8, эпох = 30, test = 69,50 % (train 71,86 %).

Выводы по подбору: При оценке по тестовой выборке картина меняется по сравнению с обучением на всей выборке.

8 нейронов, 30 эпох дают наилучшую тестовую точность (69,50 %) и небольшой разрыв с обучающей (71,86 %) — приемлемое обобщение.

2 нейрона — недостаточная ёмкость, но с ростом эпох качество растёт (до 67,67 % на тесте при 100 эпохах).

4 и 16 нейронов в данном запуске при обучении на 70 % данных не обучаются: точность на train и test остаётся около 50 % (сеть «застревает» в плохом минимуме из-за инициализации или размера выборки).

Таким образом, оптимальная конфигурация при честной оценке по тесту — 8 нейронов, 30 эпох; выбор лучшей архитектуры по обучающей или полной выборке может приводить к другим результатам и не гарантирует лучшего качества на новых данных.

Выводы по работе

В работе реализована простейшая нейронная сеть с одним скрытым слоем и сигмоидой без использования библиотек глубокого обучения. Реализованы прямое распространение и обратное распространение по градиенту MSE; обучение проводится на данных, сгенерированных генератором из lab1.

По результатам экспериментов:

Сходимость обучения (на полной выборке): MSE снижается в первые эпохи и выходит на плато (~0,15); точность на обучающей выборке при 4 нейронах достигает 80–84 %. Дальнейшее увеличение числа эпох (до 50) не даёт устойчивого прироста.

Подбор архитектуры — два сценария:

Обучение на всей выборке, оценка на ней же:

Лучший результат давали 4 нейрона, 30 эпох (83,60 %); при 8 и 16 нейронах точность падала (переобучение или плохой минимум).

Обучение на 70 %, оценка на 30 % теста:

Лучшая тестовая точность — 8 нейронов, 30 эпох (69,50 %). При 4 и 16 нейронах в этом запуске сеть не обучается (train и test ≈ 50 %); при 2 нейронах точность растёт с числом эпох (до 67,67 % на тесте при 100 эпохах).

Оценка на отложенной выборке (70/30): Сеть с 4 нейронами, обученная только на 70 % данных, в данном запуске дала ~50 % на train и test (не обучается). Подбор по тестовой точности даёт рабочую конфигурацию (8 нейронов, 30 эпох, 69,50 % на тесте) и показывает, что выбор гиперпараметров по обучающей или полной выборке может быть misleading — нужна оценка по отложенным данным.

Практические аспекты: Графики MSE и точности по эпохам полезны для выбора числа эпох. Метод test() позволяет оценивать модель на новых данных. Сохранение весов достаточно для воспроизведения сети в другом приложении.

Итог: Реализация соответствует заданию. Эксперименты показывают, что оптимальная конфигурация зависит от способа оценки: при оценке по тестовой выборке (обучение на 70 %) лучший результат — 8 нейронов, 30 эпох (69,50 % на тесте); при этом часть конфигураций (4, 16 нейронов) на уменьшенной выборке может не обучаться. Честная оценка по отложенной выборке необходима для выбора модели.

Код программы:

Файл: main.py

```
"""
Лабораторная работа 4. Нейронные сети.
"""
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

from lab4.model import NeuralNetwork
from lab4.utils import accuracy, mse_loss, split_70_30

FIGURES_DIR = Path(__file__).resolve().parent / "figures"
FIGURES_DIR.mkdir(exist_ok=True)

try:
    import lab1.DataGenerator as dg
except ImportError:
    import sys

    _root = Path(__file__).resolve().parent.parent
    if str(_root) not in sys.path:
        sys.path.insert(0, str(_root))
    import lab1.DataGenerator as dg


# ---------- Параметры данных ----------
N = 1000
mu0 = [0, 2, 3]
mu1 = [3, 5, 1]
sigma0 = [2, 1, 2]
sigma1 = [1, 2, 1]
col = len(mu0)
mu = [mu0, mu1]
sigma = [sigma0, sigma1]

# Генерация выборки и приведение Y к форме (n_samples, 1)
X, Y_flat, class0, class1 = dg.norm_dataset(mu, sigma, N)
n_samples = X.shape[0]
Y = np.reshape(Y_flat.astype(np.float64), (n_samples, 1))

# Параметры сети и обучения
N_NEURONS = 4
N_EPOCH = 50
LEARNING_RATE = 0.5
SEED = 42

# Инициализация сети
NN = NeuralNetwork(X, Y, n_neuro=N_NEURONS, learning_rate=LEARNING_RATE,
seed=SEED)

# История потерь и точности по эпохам
history_loss = []
history_acc = []

print("Обучение нейронной сети...")
for i in range(N_EPOCH):
```

```
    pred = NN.feedforward()
    loss = mse_loss(Y, pred)
    acc = accuracy(Y, pred)
    history_loss.append(loss)
    history_acc.append(acc)
    if (i + 1) % 10 == 0 or i == 0:
        print(f"  Эпоха {i + 1:3d}  Loss: {loss:.6f}  Точность:
{acc*100:.2f}%")
    NN.train_step(X, Y)

# Итоговая точность
pred_final = NN.feedforward()
final_accuracy = accuracy(Y, pred_final)
final_loss = mse_loss(Y, pred_final)
print(f"\nИтоговая точность на обучающей выборке:
{final_accuracy*100:.2f}%")
print(f"Итоговая MSE: {final_loss:.6f}")

# Вынесение весов в отдельные переменные (п.5 самостоятельной)
weights_layer1 = NN.weights1.copy()
weights_layer2 = NN.weights2.copy()
print(f"\nВеса первого слоя (вход -> скрытый), shape
{weights_layer1.shape}:")
print(weights_layer1)
print(f"\nВеса второго слоя (скрытый -> выход), shape
{weights_layer2.shape}:")
print(weights_layer2)

# ---------- Графики зависимости потерь и точности от номера эпохи --------
--
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(range(1, N_EPOCH + 1), history_loss, "b-")
ax1.set_xlabel("Номер эпохи")
ax1.set_ylabel("Среднеквадратичная ошибка (MSE)")
ax1.set_title("Функция потерь на обучающей выборке")
ax1.grid(True, alpha=0.3)

ax2.plot(range(1, N_EPOCH + 1), [a * 100 for a in history_acc], "g-")
ax2.set_xlabel("Номер эпохи")
ax2.set_ylabel("Точность, %")
ax2.set_title("Точность на обучающей выборке")
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(FIGURES_DIR / "lab4_loss_accuracy.png")
plt.close()
print(f"\nГрафики сохранены в папке: {FIGURES_DIR}")

# ---------- Доп. задание: оценка на тестовой выборке (метод test) --------
--
# Используем конфигурацию, оптимальную по тестовой точности: 8 нейронов, 30
эпох
print("\n--- Оценка на тестовой выборке (train/test 70/30, 8 нейронов, 30
эпох) ---")
Xtrain, Ytrain, Xtest, Ytest = split_70_30(X, Y)
NN_split = NeuralNetwork(Xtrain, Ytrain, n_neuro=8,
learning_rate=LEARNING_RATE, seed=SEED)
for _ in range(30):
```

```
    NN_split.train_step(Xtrain, Ytrain)
pred_test = NN_split.test(Xtest)
acc_test = accuracy(Ytest, pred_test)
acc_train_split = accuracy(Ytrain, NN_split.feedforward())
print(f"  Точность на обучающей выборке: {acc_train_split*100:.2f}%")
print(f"  Точность на тестовой выборке (метод test): {acc_test*100:.2f}%")

# ---------- Подбор числа нейронов и эпох (оценка по тестовой выборке) ----
------
print("\n--- Подбор числа нейронов и эпох (обучение на 70%, оценка на 30%
теста) ---")
best_test_acc = -1
best_nn, best_ep = None, None
for n_neur in [2, 4, 8, 16]:
    for n_ep in [30, 50, 100]:
        nn = NeuralNetwork(Xtrain, Ytrain, n_neuro=n_neur,
learning_rate=LEARNING_RATE, seed=SEED)
        for _ in range(n_ep):
            nn.train_step(Xtrain, Ytrain)
        acc_train = accuracy(Ytrain, nn.feedforward())
        acc_test_val = accuracy(Ytest, nn.test(Xtest))
        print(f"  нейронов={n_neur:2d}, эпох={n_ep:3d} -> train
{acc_train*100:.2f}%, test {acc_test_val*100:.2f}%")
        if acc_test_val > best_test_acc:
            best_test_acc = acc_test_val
            best_nn, best_ep = n_neur, n_ep
print(f"  Оптимально по тестовой точности: нейронов={best_nn},
эпох={best_ep}, test={best_test_acc*100:.2f}%")
```

Файл: model.py

```
"""
Модель нейронной сети: активационные функции и класс NeuralNetwork.
"""
import numpy as np


def sigmoid(Z):
    Z = np.clip(Z, -500, 500)
    return 1 / (1 + np.exp(-Z))


def sigmoid_derivative(p):
    """Производная сигмоиды в точке p: sigma'(p) = p * (1 - p)."""
    return p * (1 - p)


class NeuralNetwork:
    """
    Однослойная нейронная сеть: входной слой -> скрытый слой (n_neuro
нейронов) -> выход (1 нейрон).
    Обучение: прямое распространение (feedforward) + обратное
распространение (backprop).
    """

    def __init__(self, x, y, n_neuro=4, learning_rate=0.5, seed=None):
        """
        Инициализация весов случайными значениями.
```

```
        x: (n_samples, n_features), y: (n_samples, 1).
        """
        if seed is not None:
            np.random.seed(seed)
        self.input = x
        n_inp = self.input.shape[1]
        self.n_neuro = n_neuro
        self.lr = learning_rate
        self.weights1 = np.random.rand(n_inp, n_neuro)
        self.weights2 = np.random.rand(n_neuro, 1)
        self.y = y
        self.output = np.zeros(y.shape)

    def feedforward(self, X=None):
        """
        Прямое распространение: выходы слоёв по сигмоиде.
        Если X задан — используется он (для предсказания на тестовых
данных), иначе self.input.
        """
        inp = X if X is not None else self.input
        self.layer1 = sigmoid(np.dot(inp, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2

    def backprop(self):
        """
        Обратное распространение: коррекция весов по градиенту MSE.
        .T — транспонирование для согласования размерностей при градиенте
по весам.
        """
        d_weights2 = np.dot(
            self.layer1.T,
            2 * (self.y - self.output) * sigmoid_derivative(self.output),
        )
        d_weights1 = np.dot(
            self.input.T,
            np.dot(
                2 * (self.y - self.output) *
sigmoid_derivative(self.output),
                self.weights2.T,
            )
            * sigmoid_derivative(self.layer1),
        )
        self.weights1 += self.lr * d_weights1
        self.weights2 += self.lr * d_weights2

    def train_step(self, X, y):
        """Один шаг обучения: прямой проход и обновление весов."""
        self.input = X
        self.y = y
        self.output = self.feedforward()
        self.backprop()

    def test(self, X):
        """
        Предсказание на новых данных X без изменения весов.
        Возвращает вероятности (0..1); для меток класса: (pred >
0.5).astype(int).
```

```
        """
        return self.feedforward(X)
```

Файл: utils.py

```
"""
Вспомогательные функции: метрики и разбиение выборки.
"""
import numpy as np


def accuracy(y_true, pred_proba, threshold=0.5):
    """Точность: доля правильных предсказаний. pred_proba — выход НС
(0..1)."""
    y_true = np.asarray(y_true).ravel()
    pred_proba = np.asarray(pred_proba).ravel()
    pred_class = (pred_proba >= threshold).astype(np.int64)
    y_int = (
        (np.asarray(y_true) != 0).astype(np.int64)
        if y_true.dtype == bool
        else np.asarray(y_true, dtype=np.int64)
    )
    return np.mean(pred_class == y_int)


def mse_loss(y_true, pred):
    """Среднеквадратичная ошибка."""
    y_true = np.asarray(y_true).ravel()
    pred = np.asarray(pred).ravel()
    return np.mean((y_true - pred) ** 2)


def split_70_30(X, Y, seed=42):
    """Разбиение на обучающую (70%) и тестовую (30%) выборки."""
    n = len(Y)
    rng = np.random.default_rng(seed)
    idx = rng.permutation(n)
    X, Y = X[idx], Y[idx]
    train_count = round(0.7 * n)
    return X[:train_count], Y[:train_count], X[train_count:],
Y[train_count:]
```