Цель работы.

Целью работы является разработка, обучение и проверка модели нейронной сети типа многослойного перцептрона (MLP) для распознавания изображений дорожных знаков. В качестве исходных данных используется датасет, сформированный в ИДЗ-2, включая каталог изображений и файл разметки labels.csv.

Задание.

1. Используя датасет дорожных знаков, подготовленный в ИДЗ-2, реализовать модель нейронной сети для классификации изображений.
2. Выполнить обучение модели на обучающей выборке.
3. Провести оценку точности классификации на тестовой выборке.
4. Реализовать сохранение и загрузку обученных весов.
5. Обеспечить возможность распознавания отдельного изображения.
6. Проанализировать процесс обучения и полученные результаты.

Исходные данные

В качестве исходных данных используется датасет дорожных знаков, сформированный в ИДЗ-2. Датасет содержит 1000 изображений размером 30×30 пикселей и файл разметки labels.csv, в котором для каждого изображения задана метка класса.

Количество классов — 10; каждый класс представлен одинаковым числом изображений. Изображения предварительно нормированы и хранятся в каталоге dataset/ рядом с файлом разметки.

Теоретические положения.

Для решения задачи распознавания образов используется многослойный перцептрон (MLP). MLP представляет собой последовательность полносвязных слоёв, в которых каждый нейрон следующего слоя получает взвешенную сумму выходов предыдущего слоя и значение смещения (bias).

Поскольку изображения имеют фиксированный размер 30×30, входные данные удобно представлять в виде вектора признаков длиной 900. Перед обучением выполняется нормализация яркости пикселей в диапазон [0; 1], что стабилизирует обучение и ускоряет сходимость алгоритма.

Обучение сети выполняется методом обратного распространения ошибки (backpropagation) с градиентным спуском. Для многоклассовой классификации используется выходной слой размерности 10; прогноз определяется по максимальному значению выходного вектора. Для снижения переобучения применяется L2-регуляризация (штраф на величину весов).

Ход работы

Работа выполнялась на основе датасета, полученного в ИДЗ-2. Основная программа реализована в виде консольного приложения, поддерживающего обучение модели, проверку точности на тестовой выборке и распознавание отдельного изображения по заданному пути.

Общий алгоритм работы программного приложения приведён на рисунке 3.

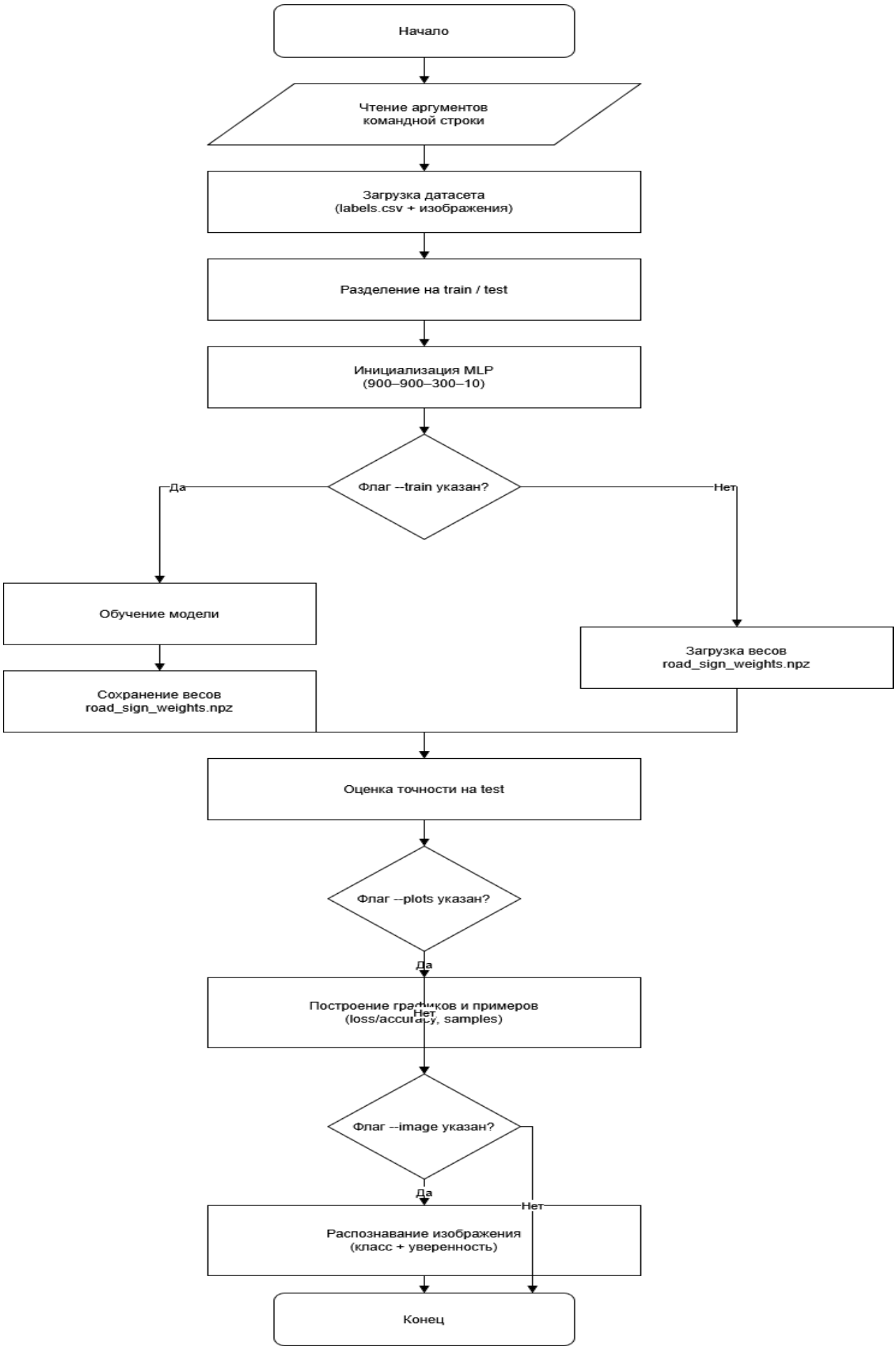

Рисунок 3 — Общий алгоритм работы программы распознавания

1. Загрузка разметки и подготовка списка файлов

На первом этапе считывается файл labels.csv. Для каждой строки извлекаются имя файла и метка класса. Пустые и некорректные значения отбрасываются, после чего формируется список пар (filename, class), который далее используется для загрузки изображений.

Листинг 1 — Загрузка labels.csv

```
1. def load_labels_csv(csv_path: str):
2.     rows = []
3.     with open(csv_path, 'r', encoding='utf-8', newline='') as f:
4.         reader = csv.DictReader(f)
5.         for r in reader:
6.             fn = (r.get('filename') or '').strip()
7.             cls = (r.get('class') or '').strip()
8.             if fn and cls:
9.                 rows.append((fn, cls))
10.    return rows
```

1.1 Формирование отображения 'класс → индекс' (детерминированно)

Для обучения нейронной сети метки классов переводятся в числовые индексы 0…9. Порядок классов фиксируется детерминированно (через сортировку), чтобы при каждом запуске программы соответствие между выходными нейронами и классами не изменялось. Это критично для корректной загрузки сохранённых весов.

Листинг 2 — Построение class_to_idx / idx_to_class

```
1. uniq = sorted(set(labels), key=class_sort_key)
2. class_to_idx = {c: i for i, c in enumerate(uniq)}
3. idx_to_class = {i: c for c, i in class_to_idx.items()}
```

1.2 Предварительная обработка изображений

Каждое изображение приводится к формату, используемому моделью: перевод в оттенки серого, масштабирование до 30×30 пикселей, нормализация яркости и преобразование в вектор признаков длиной 900 элементов.

Листинг 3 — Чтение изображения и преобразование в вектор

```
1. def read_image_as_vector(image_path: str) -> np.ndarray:
2.     img = Image.open(image_path).convert('L')
3.     img = img.resize((30, 30))
4.     arr = (np.array(img, dtype=np.float32) / 255.0).reshape(-1)
5.     return arr
```

1.3 Формирование массива признаков X и массива меток y

После предобработки всех изображений формируется матрица признаков X размерности (N, 900) и вектор меток y_idx. Для обучения дополнительно формируется one-hot представление целевых классов.

Листинг 4 — Формирование X и y_idx

```
X = np.zeros((len(rows), 900), dtype=np.float32)
y_idx = np.zeros((len(rows),), dtype=np.int64)
for i, (fn, cls) in enumerate(rows):
    X[i, :] = read_image_as_vector(os.path.join(dataset_dir, fn))
    y_idx[i] = class_to_idx[cls]
```

2. Разделение датасета на обучающую и тестовую выборки

Для оценки качества классификации выполняется разделение данных на обучающую и тестовую части. Разбиение выполняется стратифицированно, чтобы сохранить равномерность классов.

3. Инициализация и параметры нейронной сети

Используется MLP с архитектурой 900–900–300–10. Параметры обучения (число итераций, скорость обучения и коэффициент регуляризации) задаются через аргументы командной строки, что упрощает повторение экспериментов.

4. Обучение модели и сохранение весов

Обучение запускается при использовании флага --train. В процессе обучения вычисляются значения функции потерь и точности, которые сохраняются для построения графиков. По завершении обучения веса сохраняются в файл road_sign_weights.npz.

Листинг 5 — Запуск обучения и сохранение весов

```
model.train(X_train, y_train_oh, X_val=X_test, Y_val=y_test_oh, iterations=args.iterations)
model.save_weights(args.weights)
```

Алгоритм обучения многослойного перцептрона реализован в виде итерационного процесса и приведён на рисунке 4.

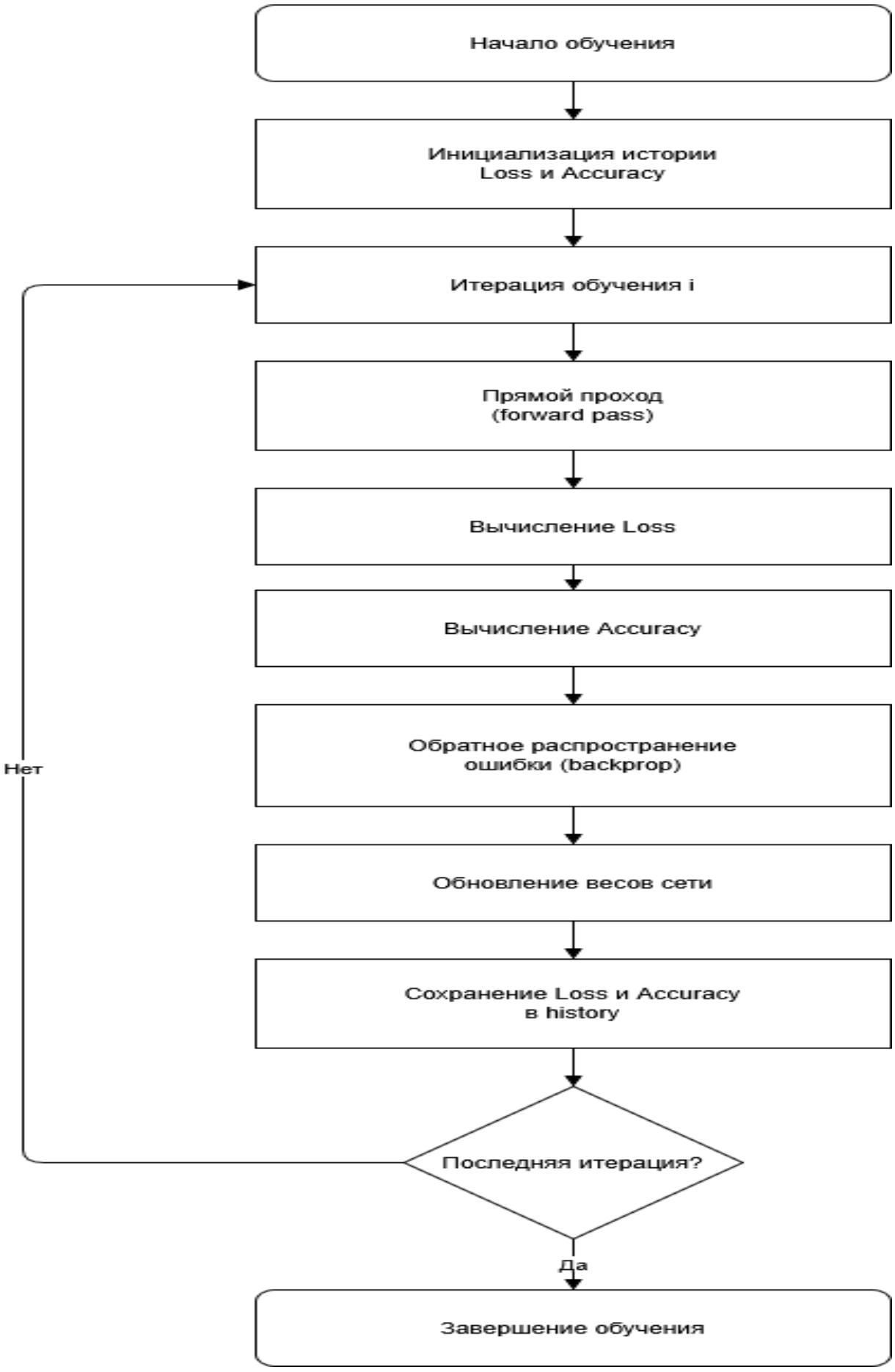

Рисунок 4 — Алгоритм обучения многослойного перцептрона

5. Загрузка весов и оценка точности

При повторных запусках (без --train) выполняется загрузка сохранённых весов, после чего рассчитывается точность на тестовой выборке. Для получения предсказаний выбирается класс с максимальной вероятностью на выходе сети.

6. Построение графиков обучения

Для анализа сходимости обучения строятся графики изменения Loss и Accuracy по итерациям. Построение выполняется при включении режима визуализации (например, флаг --plots).

7. Распознавание отдельного изображения

В режиме распознавания отдельного изображения (флаг --image) файл проходит ту же предобработку, что и обучающая выборка. После вычисления вектора вероятностей определяется наиболее вероятный класс и выводится значение уверенности.

Результаты

В результате обучения многослойного перцептрона на датасете дорожных знаков (1000 изображений, 10 классов) достигнута точность классификации на тестовой выборке 100%.

В процессе обучения фиксировались значения функции потерь (Loss) и точности (Accuracy) по итерациям. Сохранение истории обучения используется для последующего анализа сходимости и построения графиков.

После завершения обучения веса сети сохраняются в файл road_sign_weights.npz; при повторном запуске без режима обучения выполняется загрузка весов и повторная проверка качества без переобучения.

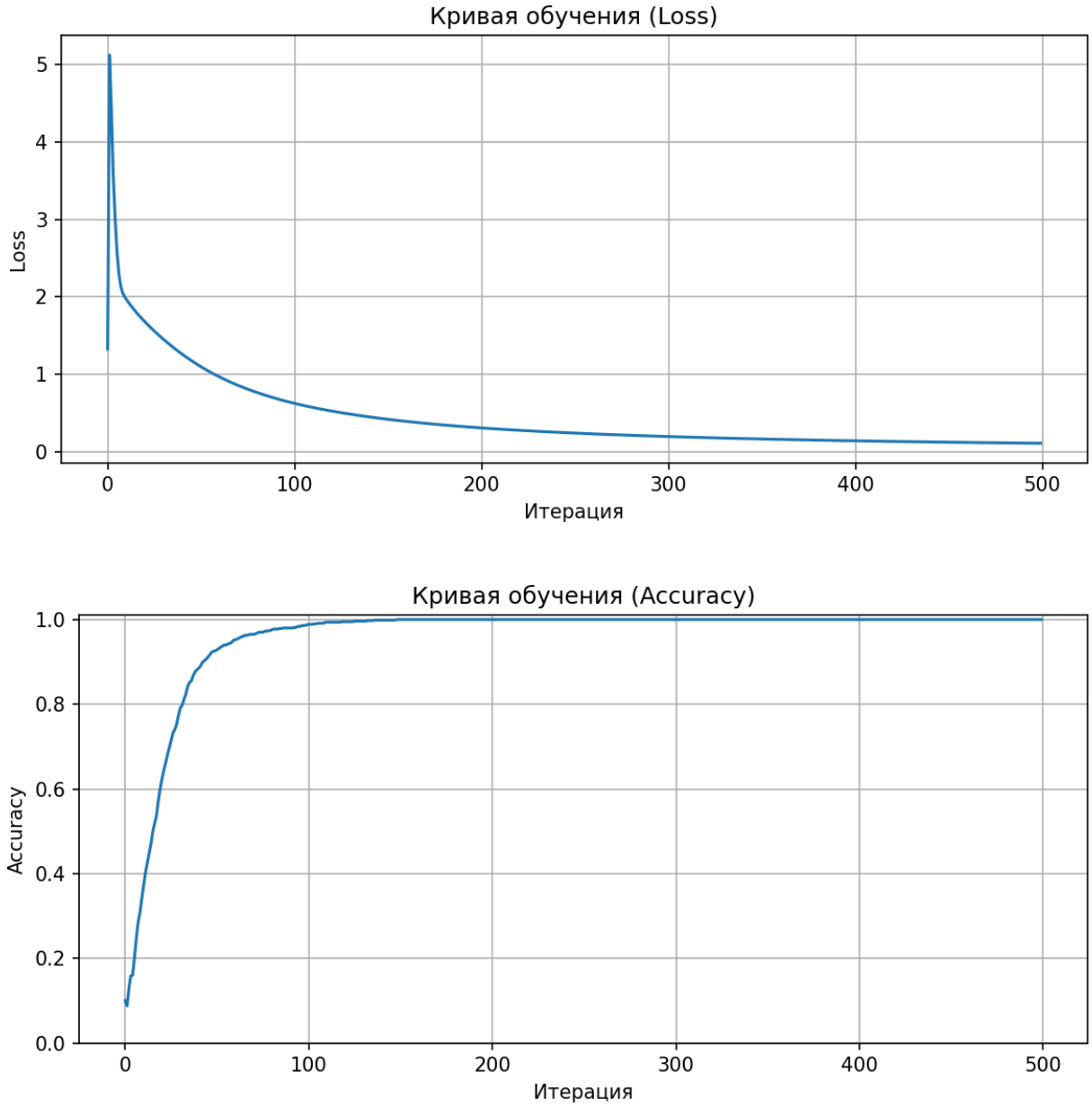

Рисунок 1 — Кривые обучения (Loss и Accuracy)

На рисунке 1 видно, что значение Loss монотонно уменьшается, а Accuracy быстро растёт и выходит на плато, что соответствует устойчивой сходимости при выбранных параметрах обучения.

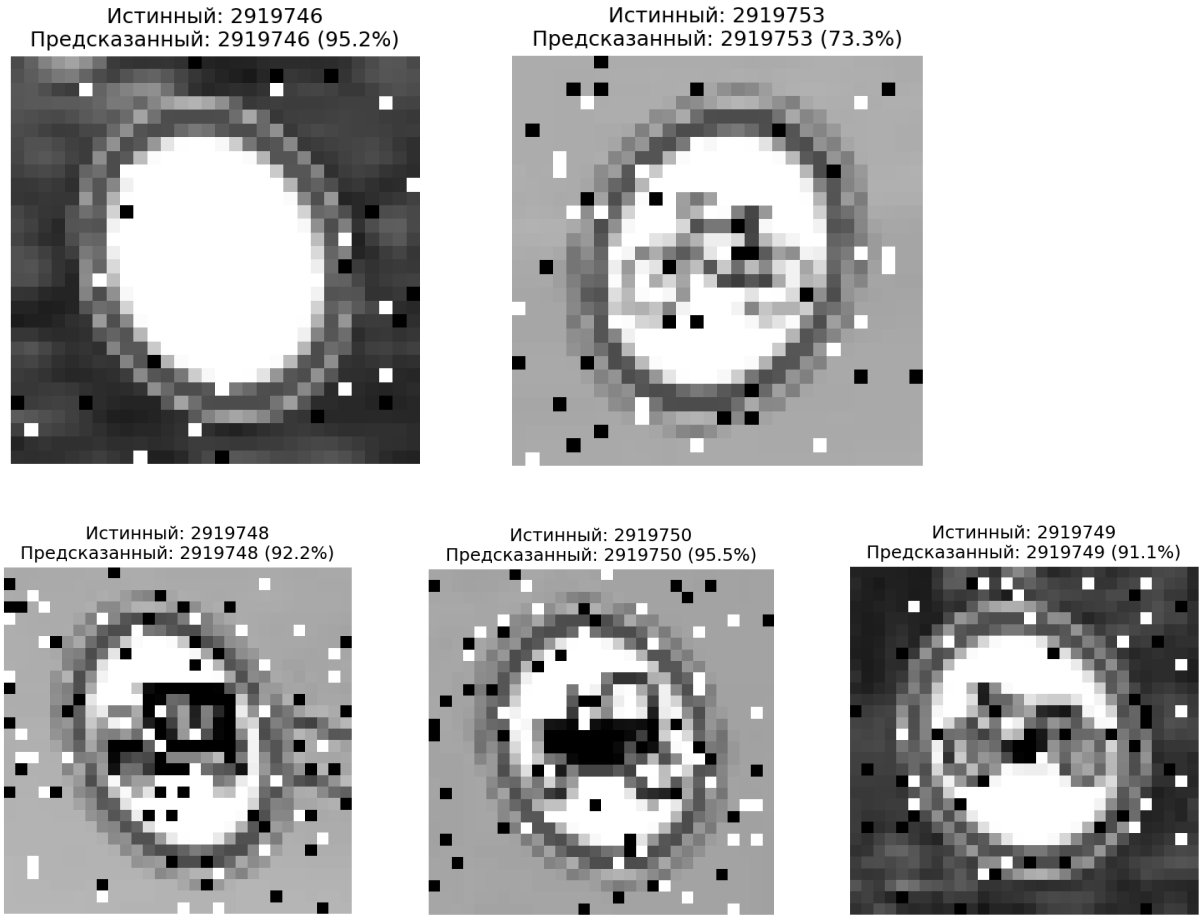

Рисунок 2 — Примеры распознавания изображений (истинный/предсказанный класс и уверенность)

На рисунке 2 приведены примеры классификации изображений из тестовой выборки и отдельных файлов. Для каждого примера отображены истинная метка, предсказанный класс и оценка уверенности модели.

ВЫВОД

В ходе выполнения индивидуального задания № 3 была разработана и протестирована модель нейронной сети для распознавания дорожных знаков на основе датасета, сформированного в ИДЗ-2. Реализованы этапы загрузки и подготовки данных, обучение многослойного перцептрона, оценка точности на тестовой выборке, а также сохранение и загрузка весов модели.

По результатам эксперимента модель демонстрирует устойчивую сходимость по функции потерь и рост точности в процессе обучения, что подтверждается графиками. Реализован режим распознавания отдельного изображения с выводом предсказанного класса и величины уверенности. Таким образом, поставленная цель работы достигнута, требования задания выполнены.

Приложение 1. Команды запуска

Ниже приведены команды для запуска из терминала VS Code. Для публикации в открытом доступе пути к пользовательским каталогам не используются; предполагается, что команды выполняются из корня проекта (папка ИДЗ-3).

```
# Создание виртуального окружения (один раз)
python -m venv .venv
# Активация окружения (Windows)
.\.venv\Scripts\activate
# Установка зависимостей
python -m pip install -U pip
python -m pip install numpy pillow scikit-learn matplotlib
# Обучение модели + построение графиков
python "ИДЗ-3.py" --train --plots
# Проверка точности на тестовой выборке (загрузка сохранённых весов)
python "ИДЗ-3.py"
# Распознавание одного изображения
python "ИДЗ-3.py" --image dataset\img_00001.png
# Показ примеров распознавания из тестовой выборки
python "ИДЗ-3.py" --plots --samples 5
```

Приложение 2. Исходный код ИДЗ-3.py

```
# ИДЗ-3.py
# Обучение MLP на датасете из ИДЗ-2.
# Датасет: dataset/labels.csv (filename,class) + изображения.
# Важно: порядок классов должен быть ОДИНАКОВЫМ при обучении и при последующих запусках.
# Поэтому список классов строится детерминированно (без влияния set/hash).

from __future__ import annotations

import os
import csv
import argparse
from typing import List, Tuple, Dict

import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split

# Графики/картинки — это “витрина” результатов.
# На обучение это не влияет, но удобно для отчёта и самопроверки.
import matplotlib.pyplot as plt

from mlp import MLP


# ------------------------
# Константы по ТЗ
# ------------------------
IMG_W, IMG_H = 30, 30
N_INPUT = IMG_W * IMG_H  # 900
N_CLASSES = 10
TEST_SIZE = 200


# ------------------------
# Стабильная сортировка классов
# ------------------------
def class_sort_key(label: str):
    """
    Делает стабильный ключ сортировки для меток классов.
    Поддерживает:
      - "2919753" -> числовой id
      - "3.10"    -> две части через точку
      - остальное -> строковая сортировка
    """
    s = label.strip()

    # 1) Чисто число
    if s.isdigit():
        return (0, int(s), 0, "")

    # 2) Формат A.B (например 3.10)
    if "." in s:
        parts = s.split(".")
        if len(parts) == 2 and parts[0].isdigit() and parts[1].isdigit():
            return (1, int(parts[0]), int(parts[1]), "")

    # 3) Фолбэк: строка
    return (2, 0, 0, s)

```

```
60.
61. # -------------------------
62. # Загрузка разметки и датасета
63. # -------------------------
64. def load_labels_csv(csv_path: str) -> List[Tuple[str, str]]:
65.     if not os.path.isfile(csv_path):
66.         raise FileNotFoundError(f"Не найден файл разметки: {csv_path}")
67.
68.     rows: List[Tuple[str, str]] = []
69.     with open(csv_path, "r", encoding="utf-8", newline="") as f:
70.         reader = csv.DictReader(f)
71.         fieldnames = list(reader.fieldnames or [])
72.         if ("filename" not in fieldnames) or ("class" not in fieldnames):
73.             raise ValueError(
74.                 f"Некорректный заголовок CSV. Нужно: filename,class. Сейчас: {reader.fieldnames}"
75.             )
76.
77.         for r in reader:
78.             fn = (r.get("filename") or "").strip()
79.             cls = (r.get("class") or "").strip()
80.             if fn and cls:
81.                 rows.append((fn, cls))
82.
83.     if not rows:
84.         raise ValueError("labels.csv пустой или не содержит корректных строк.")
85.     return rows
86.
87.
88. def read_image_as_vector(image_path: str) -> np.ndarray:
89.     """
90.     Приведение картинки к 30x30 grayscale и разворот в вектор длиной 900.
91.     """
92.     img = Image.open(image_path).convert("L")
93.     img = img.resize((IMG_W, IMG_H))
94.     arr = (np.array(img, dtype=np.float32) / 255.0).reshape(-1)
95.     if arr.size != N_INPUT:
96.         raise ValueError(f"Неверный размер входа: {arr.size}, ожидалось {N_INPUT}")
97.     return arr
98.
99.
100. def load_dataset(dataset_dir: str, labels_csv: str):
101.     """
102.     Возвращает:
103.       X: (N, 900) float32
104.       y_idx: (N,) int64
105.       y_oh: (N, 10) float32
106.       idx_to_class: список из 10 меток классов (строки) в фиксированном порядке
107.     """
108.     csv_path = os.path.join(dataset_dir, labels_csv)
109.     rows = load_labels_csv(csv_path)
110.
111.     labels = [cls for _, cls in rows]
112.     uniq = sorted(set(labels), key=class_sort_key)
113.
114.     if len(uniq) != N_CLASSES:
115.         raise ValueError(
116.             f"Ожидается {N_CLASSES} классов, найдено: {len(uniq)} -> {uniq}"
117.         )
118.
119.     class_to_idx: Dict[str, int] = {c: i for i, c in enumerate(uniq)}
120.
121.     X = np.zeros((len(rows), N_INPUT), dtype=np.float32)
122.     y_idx = np.zeros((len(rows),), dtype=np.int64)
```

```
123.
124.     for i, (fn, cls) in enumerate(rows):
125.         img_path = os.path.join(dataset_dir, fn)
126.         if not os.path.isfile(img_path):
127.             raise FileNotFoundError(f"Не найден файл изображения из CSV: {img_path}")
128.
129.         # важно: X[i, :] (так и типизатору понятно, и явно что вектор)
130.         X[i, :] = read_image_as_vector(img_path)
131.         y_idx[i] = class_to_idx[cls]
132.
133.     y_oh = np.eye(N_CLASSES, dtype=np.float32)[y_idx]
134.
135.     print(f"Загружено {X.shape[0]} изображений, размер входа: {X.shape[1]}")
136.     return X, y_idx, y_oh, uniq
137.
138.
139. def ensure_dir(path: str) -> None:
140.     # На Windows проще создавать папку явно, чем ловить исключения при сохранении
141.     os.makedirs(path, exist_ok=True)
142.
143.
144. def save_training_plots(history: dict, out_dir: str, show: bool) -> None:
145.     """Сохраняет два графика: Loss и Accuracy."""
146.     ensure_dir(out_dir)
147.
148.     loss = history.get("loss", [])
149.     acc = history.get("accuracy", [])
150.     if not loss or not acc:
151.         print("[WARN] История обучения пустая — графики не строю.")
152.         return
153.
154.     # Loss
155.     plt.figure(figsize=(8, 4))
156.     plt.plot(loss)
157.     plt.title("Кривая обучения (Loss)")
158.     plt.xlabel("Итерация")
159.     plt.ylabel("Loss")
160.     plt.grid(True)
161.     plt.tight_layout()
162.     plt.savefig(os.path.join(out_dir, "training_loss.png"), dpi=150)
163.     if show:
164.         plt.show()
165.     plt.close()
166.
167.     # Accuracy
168.     plt.figure(figsize=(8, 4))
169.     plt.plot(acc)
170.     plt.title("Кривая обучения (Accuracy)")
171.     plt.xlabel("Итерация")
172.     plt.ylabel("Accuracy")
173.     plt.ylim(0.0, 1.01)
174.     plt.grid(True)
175.     plt.tight_layout()
176.     plt.savefig(os.path.join(out_dir, "training_accuracy.png"), dpi=150)
177.     if show:
178.         plt.show()
179.     plt.close()
180.
181.
182. def save_prediction_image(
183.     image_vec: np.ndarray,
184.     title: str,
185.     out_path: str,
```

```
    show: bool,
) -> None:
    """Сохраняет 30x30 картинку с заголовком."""
    plt.figure(figsize=(4, 4))
    plt.imshow(image_vec.reshape(IMG_H, IMG_W), cmap="gray")
    plt.title(title)
    plt.axis("off")
    plt.tight_layout()
    plt.savefig(out_path, dpi=150)
    if show:
        plt.show()
    plt.close()


def export_test_samples(
    model: MLP,
    X_test: np.ndarray,
    y_test: np.ndarray,
    idx_to_class: list[str],
    out_dir: str,
    n_samples: int,
    seed: int,
    show: bool,
) -> None:
    """Сохраняет несколько примеров из тестовой выборки: истинный/предсказанный класс и уверенность."""
    ensure_dir(out_dir)
    rng = np.random.default_rng(seed)

    n = int(min(n_samples, X_test.shape[0]))
    if n <= 0:
        return

    idxs = rng.choice(X_test.shape[0], size=n, replace=False)
    for k, i in enumerate(idxs, start=1):
        x = X_test[i : i + 1]
        proba = model.predict_proba(x)[0]
        pred_idx = int(np.argmax(proba))
        conf = float(np.max(proba))

        true_lbl = idx_to_class[int(y_test[i])]
        pred_lbl = idx_to_class[pred_idx]

        title = f"Истинный: {true_lbl}\nПредсказанный: {pred_lbl} ({conf*100:.1f}%)"
        out_path = os.path.join(out_dir, f"test_sample_{k:02d}.png")
        save_prediction_image(X_test[i], title, out_path, show=show)


# -------------------------
# Основной код
# -------------------------
def main():
    ap = argparse.ArgumentParser("ИДЗ-3 Беляев (MLP)")
    ap.add_argument(
        "--dataset_dir",
        default="dataset",
        help="Папка с датасетом (по умолчанию dataset)",
    )
    ap.add_argument(
        "--labels_csv", default="labels.csv", help="CSV разметка внутри dataset_dir"
    )
    ap.add_argument(
        "--weights", default="road_sign_weights.npz", help="Файл весов .npz"
```

```
    )
    ap.add_argument(
        "--train", action="store_true", help="Обучить модель и сохранить веса"
    )
    ap.add_argument(
        "--iterations", type=int, default=500, help="Число итераций обучения"
    )
    ap.add_argument("--lr", type=float, default=0.04, help="learning rate")
    ap.add_argument(
        "--lambda",
        dest="reg_lambda",
        type=float,
        default=0.001,
        help="L2-регуляризация",
    )
    ap.add_argument(
        "--image", default=None, help="Распознать одно изображение (путь к файлу)"
    )
    ap.add_argument("--seed", type=int, default=42, help="seed для воспроизводимости")

    # Доп. опции — чисто для отчёта/проверки
    ap.add_argument(
        "--plots",
        action="store_true",
        help="Сохранить графики обучения и несколько примеров из теста",
    )
    ap.add_argument(
        "--out_dir",
        default="outputs",
        help="Папка для графиков/картинок (по умолчанию outputs)",
    )
    ap.add_argument(
        "--samples",
        type=int,
        default=3,
        help="Сколько примеров из тестовой выборки сохранить (по умолчанию 3)",
    )
    ap.add_argument(
        "--show",
        action="store_true",
        help="Показать графики/картинки на экране (если нужно)",
    )

    args = ap.parse_args()

    # Сделаем пути независимыми от того, откуда запускают скрипт
    script_dir = os.path.dirname(os.path.abspath(__file__))
    os.chdir(script_dir)

    # Данные
    X, y_idx, y_oh, idx_to_class = load_dataset(args.dataset_dir, args.labels_csv)

    if X.shape[0] != 1000:
        # По ТЗ ожидается 1000; если другое число — лучше знать об этом явно
        print(f"[WARN] В датасете {X.shape[0]} изображений (по ТЗ обычно 1000).")

    if X.shape[0] <= TEST_SIZE:
        raise ValueError(
            f"Недостаточно данных для test_size={TEST_SIZE}. Всего: {X.shape[0]}"
        )

    # Разбиение (фиксированное и воспроизводимое)
    X_tr, X_te, y_tr, y_te, oh_tr, oh_te = train_test_split(
```

```
311.         X, y_idx, y_oh, test_size=TEST_SIZE, random_state=args.seed, stratify=y_idx
312.     )
313.
314.     # Модель
315.     model = MLP(
316.         layer_sizes=[900, 900, 300, 10],
317.         learning_rate=args.lr,
318.         reg_lambda=args.reg_lambda,
319.         use_bias=True,
320.         seed=args.seed,
321.     )
322.
323.     # Обучение / загрузка
324.     if args.train:
325.         model.train(X_tr, oh_tr, iterations=args.iterations, verbose_every=50)
326.         model.save_weights(args.weights)
327.         print(f"Веса сохранены в файл {args.weights}")
328.
329.         # Графики обучения актуальны только в режиме train (история появляется при обучении)
330.         if args.plots:
331.             save_training_plots(model.history, args.out_dir, show=args.show)
332.     else:
333.         if not os.path.isfile(args.weights):
334.             raise FileNotFoundError(
335.                 f"Файл весов не найден: {args.weights}. Сначала запусти с флагом --train."
336.             )
337.         model.load_weights(args.weights)
338.         print(f"Веса загружены из файла {args.weights}")
339.
340.     # Точность на тесте
341.     y_pred = model.predict(X_te)
342.     acc = float(np.mean(y_pred == y_te))
343.     print(f"Точность на тестовой выборке: {acc:.2%}")
344.
345.     # Несколько примеров из тестовой выборки — удобно приложить в отчёт
346.     if args.plots:
347.         export_test_samples(
348.             model=model,
349.             X_test=X_te,
350.             y_test=y_te,
351.             idx_to_class=idx_to_class,
352.             out_dir=args.out_dir,
353.             n_samples=args.samples,
354.             seed=args.seed,
355.             show=args.show,
356.         )
357.
358.     # Распознавание одного изображения
359.     if args.image:
360.         x = read_image_as_vector(args.image).reshape(1, -1)
361.         proba = model.predict_proba(x)[0]
362.         k = int(np.argmax(proba))
363.         print(
364.             f"Распознанный класс: {idx_to_class[k]}, уверенность: {float(np.max(proba)):.2%}"
365.         )
366.
367.         # Если включили режим отчёта — сохраняем и эту картинку
368.         if args.plots:
369.             ensure_dir(args.out_dir)
370.             title = f"Класс: {idx_to_class[k]} ({float(np.max(proba))*100:.1f}%)"
371.             out_path = os.path.join(args.out_dir, "single_image_prediction.png")
372.             save_prediction_image(read_image_as_vector(args.image), title, out_path,
show=args.show)
```

```
373.
374.
375. if __name__ == "__main__":
376.     main()
```

Приложение 3. Исходный код mlp.py

```
1. # mlp.py
2. # Многослойный перцептрон (MLP) для ИДЗ-3.
3. # Реализация на numpy, активация sigmoid, обучение backprop.
4. # Веса сохраняются в npz с ключами W0, W1... + метаданные (чтобы не было тихих ошибок при загрузке).
5.
6. from __future__ import annotations
7. import numpy as np
8.
9.
10. class MLP:
11.     def __init__(
12.         self,
13.         layer_sizes: list[int],
14.         learning_rate: float = 0.04,
15.         reg_lambda: float = 0.001,
16.         use_bias: bool = True,
17.         seed: int = 42,
18.     ):
19.         if not isinstance(layer_sizes, (list, tuple)) or len(layer_sizes) < 2:
20.             raise ValueError(
21.                 "layer_sizes должен быть списком, например [900, 900, 300, 10]."
22.             )
23.         if layer_sizes[0] != 900 or layer_sizes[-1] != 10:
24.             raise ValueError("По ТЗ ожидается вход 900 и выход 10.")
25.
26.         self.layer_sizes = list(layer_sizes)
27.         self.learning_rate = float(learning_rate)
28.         self.reg_lambda = float(reg_lambda)
29.         self.use_bias = bool(use_bias)
30.
31.         self.rng = np.random.default_rng(seed)
32.
33.         # веса по слоям: W0 для (900->900), W1 для (900->300), W2 для (300->10)
34.         self.W: list[np.ndarray] = []
35.         self._init_weights()
36.
37.         # для графиков/контроля
38.         self.history = {"loss": [], "accuracy": []}
39.
40.     # ---------- базовая математика ----------
41.
42.     @staticmethod
43.     def _sigmoid(z: np.ndarray) -> np.ndarray:
44.         z = np.clip(z, -60, 60)
45.         return 1.0 / (1.0 + np.exp(-z))
46.
47.     @staticmethod
48.     def _add_bias(X: np.ndarray) -> np.ndarray:
49.         # добавляем единичный столбец слева
50.         return np.concatenate([np.ones((X.shape[0], 1), dtype=X.dtype), X], axis=1)
51.
52.     def _init_weights(self) -> None:
```

```
53.         # Xavier init (нормально подходит для sigmoid)
54.         self.W.clear()
55.         for n_in, n_out in zip(self.layer_sizes[:-1], self.layer_sizes[1:]):
56.             eps = 4.0 * np.sqrt(6.0) / np.sqrt(n_in + n_out)
57.             fan_in = n_in + (1 if self.use_bias else 0)
58.             w = self.rng.uniform(-eps, eps, size=(n_out, fan_in)).astype(np.float32)
59.             self.W.append(w)
60.
61.     # ---------- прямой проход ----------
62.
63.     def _forward(self, X: np.ndarray) -> list[np.ndarray]:
64.         # A[0]=X, A[-1]=выход сети
65.         A: list[np.ndarray] = [X]
66.         a = X
67.         for w in self.W:
68.             a_in = self._add_bias(a) if self.use_bias else a
69.             z = a_in @ w.T
70.             a = self._sigmoid(z)
71.             A.append(a)
72.         return A
73.
74.     def predict_proba(self, X: np.ndarray) -> np.ndarray:
75.         return self._forward(X)[-1]
76.
77.     def predict(self, X: np.ndarray) -> np.ndarray:
78.         return np.argmax(self.predict_proba(X), axis=1)
79.
80.     # ---------- loss и backprop ----------
81.
82.     def _loss_ce(self, Y: np.ndarray, Y_hat: np.ndarray) -> float:
83.         eps = 1e-8
84.         Y_hat = np.clip(Y_hat, eps, 1.0 - eps)
85.         loss = -np.mean(np.sum(Y * np.log(Y_hat), axis=1))
86.
87.         # L2 (bias не регуляризуем)
88.         if self.reg_lambda > 0:
89.             reg = 0.0
90.             for w in self.W:
91.                 reg += float(np.sum((w[:, 1:] if self.use_bias else w) ** 2))
92.             loss += (self.reg_lambda / (2.0 * Y.shape[0])) * reg
93.
94.         return float(loss)
95.
96.     def _backprop(self, A: list[np.ndarray], Y: np.ndarray) -> list[np.ndarray]:
97.         n = Y.shape[0]
98.         grads: list[np.ndarray] = [np.empty_like(w) for w in self.W]
99.
100.          # для sigmoid + CE удобно: delta = y_hat - y
101.          delta = A[-1] - Y
102.
103.          for layer in reversed(range(len(self.W))):
104.              a_prev = A[layer]
105.              a_prev_b = self._add_bias(a_prev) if self.use_bias else a_prev
106.
107.              grad = (delta.T @ a_prev_b) / n
108.
109.              if self.reg_lambda > 0:
110.                  if self.use_bias:
111.                      grad[:, 1:] += (self.reg_lambda / n) * self.W[layer][:, 1:]
112.                  else:
113.                      grad += (self.reg_lambda / n) * self.W[layer]
114.
115.              grads[layer] = grad.astype(np.float32)
```

```
116.
117.             if layer > 0:
118.                 w_no_bias = self.W[layer][:, 1:] if self.use_bias else self.W[layer]
119.                 deriv = A[layer] * (1.0 - A[layer])  # sigmoid'
120.                 delta = (delta @ w_no_bias) * deriv
121.
122.         return grads
123.
124.     # ---------- обучение ----------
125.
126.     def train(
127.         self,
128.         X: np.ndarray,
129.         Y: np.ndarray,
130.         iterations: int = 500,
131.         verbose_every: int = 50,
132.     ) -> None:
133.         self.history = {"loss": [], "accuracy": []}
134.
135.         for it in range(1, iterations + 1):
136.             A = self._forward(X)
137.             Y_hat = A[-1]
138.
139.             loss = self._loss_ce(Y, Y_hat)
140.             acc = float(np.mean(np.argmax(Y_hat, axis=1) == np.argmax(Y, axis=1)))
141.
142.             grads = self._backprop(A, Y)
143.             for i in range(len(self.W)):
144.                 self.W[i] -= self.learning_rate * grads[i]
145.
146.             self.history["loss"].append(loss)
147.             self.history["accuracy"].append(acc)
148.
149.             if (
150.                 it == 1
151.                 or it == iterations
152.                 or (verbose_every and it % verbose_every == 0)
153.             ):
154.                 print(
155.                     f"Итерация {it}/{iterations}, Loss: {loss:.4f}, Точность: {acc:.4f}"
156.                 )
157.
158.     # ---------- сохранение / загрузка ----------
159.
160.     def save_weights(self, filename: str) -> None:
161.         # сохраняем в явном виде W0..Wn + метаданные, чтобы потом можно было проверить
совместимость
162.         payload: dict[str, np.ndarray] = {
163.             "layer_sizes": np.array(self.layer_sizes, dtype=np.int32),
164.             "use_bias": np.array([1 if self.use_bias else 0], dtype=np.int32),
165.         }
166.         for i, w in enumerate(self.W):
167.             payload[f"W{i}"] = w.astype(np.float32)
168.
169.         # важно: file=filename (иначе Pylance иногда путается по сигнатуре)
170.         np.savez(file=str(filename), **payload)  # type: ignore[arg-type]
171.
172.     def load_weights(self, filename: str) -> None:
173.         data = np.load(filename)
174.
175.         file_layer_sizes = data["layer_sizes"].tolist()
176.         file_use_bias = bool(int(data["use_bias"][0]))
177.
```

```
        if file_layer_sizes != self.layer_sizes:
            raise ValueError(
                f"Несовместимые слои. В файле: {file_layer_sizes}, в модели: {self.layer_sizes}"
            )
        if file_use_bias != self.use_bias:
            raise ValueError(
                f"Несовместимый use_bias. В файле: {file_use_bias}, в модели: {self.use_bias}"
            )

        weights: list[np.ndarray] = []
        for i in range(len(self.layer_sizes) - 1):
            key = f"W{i}"
            if key not in data.files:
                raise ValueError(
                    f"В файле весов нет ключа {key}. Файл старого формата или повреждён."
                )
            w = data[key].astype(np.float32)

            expected_in = self.layer_sizes[i] + (1 if self.use_bias else 0)
            expected_out = self.layer_sizes[i + 1]
            if w.shape != (expected_out, expected_in):
                raise ValueError(
                    f"W{i} имеет форму {w.shape}, ожидалось {(expected_out, expected_in)}"
                )

            weights.append(w)

        self.W = weights


# cd "ИДЗ 3"
```