Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине: «Основы машинного обучения»
Тема**: «Лабораторная работа №4 Введение в нейронные сети: построение многослойного перцептрона»**

Выполнил:
3-го курса
Группы АС-65
Кисель М. С.
Проверил:
Крощенко А.А.

Брест 2025

Цель работы: построить, обучить и оценить **многослойный перцептрон (MLP)** для решения задачи классификации

Ход работы

Общий план для всех вариантов:

1. **Импорт библиотек и подготовка данных**

• импортируйте torch, torch.nn, torch.optim, а также sklearn для загрузки данных и их предобработки;

• загрузите датасет, выполните **стандартизацию** (StandardScaler) и **кодирование** признаков;

• разделите данные на обучающую и тестовую выборки;

• **преобразуйте данные (признаки и метки) в тензоры PyTorch: torch.tensor(X_train, dtype=torch.float32)**.

2. **Определение архитектуры нейронной сети**

• создайте класс, наследуемый от torch.nn.Module;

• в методе __init__ определите все слои, которые будете использовать (например, nn.Linear, nn.ReLU, nn.Dropout);

• в методе forward опишите последовательность применения слоев к входным данным.

3. **Инициализация модели, функции потерь и оптимизатора**

• создайте экземпляр вашей модели: model = MLP();

• определите функцию потерь. Для бинарной классификации используйте nn.BCEWithLogitsLoss, для многоклассовой – nn.CrossEntropyLoss;

• определите оптимизатор: optimizer = torch.optim.Adam(model.parameters(), lr=0.001).

4. **Написание цикла обучения (Training Loop)**

• запустите цикл на определенное количество эпох;

• внутри цикла:

1. переведите модель в режим обучения: model.train();
2. сделайте предсказание (forward pass): y_pred = model(X_train);
3. рассчитайте потери (loss): loss = criterion(y_pred, y_train);
4. обнулите градиенты: optimizer.zero_grad();
5. выполните обратное распространение ошибки: loss.backward();
6. сделайте шаг оптимизации: optimizer.step().

5. **Оценка модели (Evaluation)**

• переведите модель в режим оценки: model.eval();Основы машинного обучения, 2025, Крощенко А.А.

• используйте with torch.no_grad():, чтобы отключить расчет градиентов;

• сделайте предсказания на тестовых данных;

• преобразуйте выходные данные (логиты) в предсказанные классы (например, с помощью torch.argmax или проверки порога > 0);

• рассчитайте метрики (accuracy, f1-score и т.д.), используя sklearn.metrics.

Вариант 8 Классификация семян пшеницы

• Seeds

• **Задача:** классифицировать семена на три сорта (3 класса).Основы машинного обучения, 2025, Крощенко А.А.

• **Архитектура:**

о входной слой;

о один скрытый слой с 7 нейронами (ReLU);

о выходной слой с 3 нейронами (Softmax).

• **Эксперимент:** увеличьте количество нейронов в скрытом слое до 14 и оцените, как это повлияло на точность.

1. **Импорт библиотек и подготовка данных**

```
# Импорт библиотек и загрузка данных
import os
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score
import random
import numpy as np

# Для воспроизводимости результатов
torch.manual_seed(42)
random.seed(42)
np.random.seed(42)
torch.backends.cudnn.deterministic = True

# Путь к файлу
os.chdir("d:/Универ/ОМО/ОМО2025/Лаба4/")

# Названия столбцов
columns = [
    'area', 'perimeter', 'compactness',
    'length_of_kernel', 'width_of_kernel',
    'asymmetry_coefficient', 'length_of_kernel_groove', 'class'
]

# Загружаем данные
df = pd.read_csv("seeds_dataset.txt", delim_whitespace=True, names=columns)
print(df.isnull().sum())
# удалить строки с NaN
df = df.dropna().reset_index(drop=True)

# Разделяем признаки и метки
X = df.drop('class', axis=1).values
y = df['class'].values - 1   # классы из {1,2,3} → {0,1,2}

# Разделяем на train и test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Стандартизация признаков
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Преобразуем в тензоры PyTorch
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

print("Данные успешно подготовлены.")
print(f"Размер обучающей выборки: {X_train.shape}")
print(f"Размер тестовой выборки: {X_test.shape}")
```

2. **Определение архитектуры нейронной сети**

```
# Определение архитектуры нейронной сети
class MLP(nn.Module):
    def __init__(self, input_size=7, hidden_size=7, output_size=3):
        super(MLP, self).__init__()#получает доступ к родительскому классу nn.Module через текущий экземпляр self
        # Линейный слой: вход - скрытый. Каждый вход соединён с каждым нейроном скрытого слоя
        self.fc1 = nn.Linear(input_size, hidden_size)
        # Функция активации ReLU
        self.relu = nn.ReLU()#Добавляем нелинейность, для того сеть могла учиться сложным зависимостям
        # Линейный слой: скрытый - выход
        self.fc2 = nn.Linear(hidden_size, output_size)
        # Softmax преобразует выходные значения в вероятности (только для вывода)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        # Прямое распространение
        out = self.fc1(x)       # вход - скрытый слой
        out = self.relu(out)    # активация ReLU
        out = self.fc2(out)     # скрытый - выходной слой
        return out
```

3. **Инициализация модели, функции потерь и оптимизатора**

```
# Функция для обучения модели
def train_model(model, X_train, y_train, num_epochs=100, model_name="Модель"):
    print(f"\nОбучение {model_name}")

    # Функция потерь: кросс-энтропия для многоклассовой классификации
    criterion = nn.CrossEntropyLoss()#Сравнивает предсказания модели с истинными метками

    # Оптимизатор: Adam - современный, адаптивный вариант градиентного спуска
    optimizer = optim.Adam(model.parameters(), lr=0.001)#обновляет веса, чтобы минимизировать ошибку.
```

4. **Написание цикла обучения (Training Loop)**

```
# Обучение модели (Training Loop)
    train_losses = []

    for epoch in range(num_epochs):
        model.train()  # режим обучения (включает обновление весов)
        # Прямой проход (forward pass)
        y_pred = model(X_train)
        # Вычисляем ошибку (loss)
        loss = criterion(y_pred, y_train)

        # Обнуляем старые градиенты (иначе они накапливаются)
        optimizer.zero_grad()

        # Обратное распространение ошибки
        loss.backward()

        # Обновляем веса
        optimizer.step()

        train_losses.append(loss.item())
```

```
        # Каждые 25 эпох выводим информацию о процессе
        if (epoch + 1) % 25 == 0:
            print(f"{model_name}: Эпоха [{epoch+1}/{num_epochs}], Потери (Loss):
{loss.item():.4f}")

    return train_losses
```

5. **Оценка модели (Evaluation)**

```
# Функция для оценки модели
def evaluate_model(model, X_test, y_test, model_name="Модель"):
    model.eval()  # режим оценки (без обновления весов)
    with torch.no_grad():  # отключаем подсчёт градиентов
        y_test_pred = model(X_test)
        y_pred_classes = torch.argmax(y_test_pred, dim=1)  # выбираем класс с наибольшей
вероятностью

    # Вычисляем метрики
    acc = accuracy_score(y_test, y_pred_classes)
    f1 = f1_score(y_test, y_pred_classes, average='weighted')

    print(f"\n{model_name} - Результаты на тестовой выборке:")
    print(f"Точность (Accuracy): {acc:.3f}")
    print(f"F1-мера: {f1:.3f}")

    return acc, f1, y_pred_classes
```

6. **Эксперимент:** увеличьте количество нейронов в скрытом слое до 14 и оцените, как это повлияло на точность.

```
# МОДЕЛЬ 1: С 7 нейронами в скрытом слое (базовая архитектура)
print("\n")
print("МОДЕЛЬ 1: 7 нейронов в скрытом слое")

# Инициализация модели с 7 нейронами
model_7 = MLP(input_size=7, hidden_size=7, output_size=3)
print(f"Архитектура модели: {model_7}")

# Обучение модели с 7 нейронами
train_losses_7 = train_model(model_7, X_train, y_train, num_epochs=100,
                             model_name="Модель с 7 нейронами")

# Оценка модели с 7 нейронами
acc_7, f1_7, pred_7 = evaluate_model(model_7, X_test, y_test,
                                     model_name="Модель с 7 нейронами")


# МОДЕЛЬ 2: С 14 нейронами в скрытом слое (эксперимент)
print("\n")
print("МОДЕЛЬ 2: 14 нейронов в скрытом слое (ЭКСПЕРИМЕНТ)")

# Инициализация модели с 14 нейронами
model_14 = MLP(input_size=7, hidden_size=14, output_size=3)
print(f"Архитектура модели: {model_14}")

# Обучение модели с 14 нейронами
train_losses_14 = train_model(model_14, X_train, y_train, num_epochs=100,
                              model_name="Модель с 14 нейронами")
```

```python
# Оценка модели с 14 нейронами
acc_14, f1_14, pred_14 = evaluate_model(model_14, X_test, y_test,
                                        model_name="Модель с 14 нейронами")
```

```
МОДЕЛЬ 1: 7 нейронов в скрытом слое
Архитектура модели: MLP(
  (fc1): Linear(in_features=7, out_features=7, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=7, out_features=3, bias=True)
  (softmax): Softmax(dim=1)
)

Обучение Модель с 7 нейронами
Модель с 7 нейронами: Эпоха [25/100], Потери (Loss): 1.1036
Модель с 7 нейронами: Эпоха [50/100], Потери (Loss): 1.0208
Модель с 7 нейронами: Эпоха [75/100], Потери (Loss): 0.9380
Модель с 7 нейронами: Эпоха [100/100], Потери (Loss): 0.8480

Модель с 7 нейронами - Результаты на тестовой выборке:
Точность (Accuracy): 0.881
F1-мера: 0.879


МОДЕЛЬ 2: 14 нейронов в скрытом слое (ЭКСПЕРИМЕНТ)
Архитектура модели: MLP(
  (fc1): Linear(in_features=7, out_features=14, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=14, out_features=3, bias=True)
  (softmax): Softmax(dim=1)
)

Обучение Модель с 14 нейронами
Модель с 14 нейронами: Эпоха [25/100], Потери (Loss): 0.9677
Модель с 14 нейронами: Эпоха [50/100], Потери (Loss): 0.8435
Модель с 14 нейронами: Эпоха [75/100], Потери (Loss): 0.7128
Модель с 14 нейронами: Эпоха [100/100], Потери (Loss): 0.5840

Модель с 14 нейронами - Результаты на тестовой выборке:
Точность (Accuracy): 0.833
F1-мера: 0.815
```

Сравнение результатов эксперимента:

Сравнение моделей на тестовой выборке:

Метрика	7 нейронов	14 нейронов	Изменение
Точность (Accuracy)	0.881	0.833	-0.048
F1-мера	0.879	0.815	-0.064

Анализ эксперимента:

Увеличение количества нейронов с 7 до 14 понизило точность на 5.4%

Финальные результаты:

- Лучшая модель: 7 нейронов
- Лучшая точность: 0.881
- Лучшая F1-мера: 0.879