Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №4

Выполнил
В.Д.Головкина,
студент группы АС66
Проверил
А. А. Крощенко,
доц. кафедры ИИТ,
« __ » _____2025 г.

Брест 2025

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

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();
• 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 1 Классификация ирисов
• Iris
• Задача: Определить вид ириса (3 класса).
• Архитектура:
о входной слой;
о один скрытый слой с 10 нейронами (ReLU);
о выходной слой с 3 нейронами (Softmax).
• Эксперимент: попробуйте добавить второй скрытый слой с 5 нейронами и сравните точность.

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score

file_path = r"D:\y--6a\3 kurs\omo\4\iris.csv"

df = pd.read_csv(file_path)

# пусть последний столбец — это метка класса
X = df.iloc[:, :-1].values
y_raw = df.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y_raw)  # 3 класса: 0, 1, 2

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran-
dom_state=42)

#тензоры PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

class MLP_1Hidden(nn.Module):
    def __init__(self):
        super(MLP_1Hidden, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(4, 10),
            nn.ReLU(),
            nn.Linear(10, 3)
        )

    def forward(self, x):
        return self.model(x)
```

```python
class MLP_2Hidden(nn.Module):
    def __init__(self):
        super(MLP_2Hidden, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(4, 10),
            nn.ReLU(),
            nn.Linear(10, 5),
            nn.ReLU(),
            nn.Linear(5, 3)
        )

    def forward(self, x):
        return self.model(x)

model = MLP_2Hidden()#нужную модель выбрать MLP_1Hidden()/MLP_2Hidden() и запустить.

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

n_epochs = 100 # если плохая обучаемость то повысить на 200-300
for epoch in range(n_epochs):
    model.train()
    y_pred = model(X_train_tensor)
    loss = criterion(y_pred, y_train_tensor)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f"Epoch {epoch+1}/{n_epochs}, Loss: {loss.item():.4f}")

model.eval()#Оценка
with torch.no_grad():
    y_test_logits = model(X_test_tensor)
    y_test_pred = torch.argmax(y_test_logits, dim=1)

acc = accuracy_score(y_test_tensor, y_test_pred)
f1 = f1_score(y_test_tensor, y_test_pred, average='macro')

print(f"\nAccuracy: {acc:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
PS D:\y--6a\3 kurs\omo\1> & C:/Users/
лаб4.py"
Epoch 10/100, Loss: 1.0079
Epoch 20/100, Loss: 0.9728
Epoch 30/100, Loss: 0.9396
Epoch 40/100, Loss: 0.9073
Epoch 50/100, Loss: 0.8754
Epoch 60/100, Loss: 0.8437
Epoch 70/100, Loss: 0.8117
Epoch 80/100, Loss: 0.7796
Epoch 90/100, Loss: 0.7478
Epoch 100/100, Loss: 0.7160

Accuracy: 0.9333
F1-score: 0.9306
PS D:\y--6a\3 kurs\omo\1> & C:/Users/
лаб4.py"
Epoch 10/100, Loss: 1.1456
Epoch 20/100, Loss: 1.1266
Epoch 30/100, Loss: 1.1078
Epoch 40/100, Loss: 1.0894
Epoch 50/100, Loss: 1.0700
Epoch 60/100, Loss: 1.0458
Epoch 70/100, Loss: 1.0156
Epoch 80/100, Loss: 0.9814
Epoch 90/100, Loss: 0.9422
Epoch 100/100, Loss: 0.8981

Accuracy: 0.6333
F1-score: 0.5402
PS D:\y--6a\3 kurs\omo\1>
```

Вывод: я изучила и на практике построила, обучила и оценила многослойный перцептрон (MLP) для решения задачи классификации.