МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №4

Специальность АС-66

Выполнила
Е. С. Неруш,
студент группы АС-66

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«___» _____2025 г.

Брест 2025

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

Задачи:

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 10

• Adult Census Income

• Задача: предсказать, превышает ли доход $50 тыс. в год (бинарная классификация).

• Архитектура:

- о входной слой;
- о один скрытый слой с 32 нейронами (ReLU);
- о выходной слой с 1 нейроном (Sigmoid).

• Эксперимент: сравните производительность с более глубокой моделью: два скрытых слоя по 16 нейронов.

```
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, LabelEncoder
from sklearn.metrics import accuracy_score, f1_score
import pandas as pd

# 1. Загрузка и подготовка данных
data = pd.read_csv("E:/Projects/ml_as66/reports/Nerush/lab3/src/adult.csv")

# Целевая переменная: >50K = 1, <=50K = 0
data["income"] = LabelEncoder().fit_transform(data["income"])
y = data["income"].values
X = data.drop("income", axis=1)

# Кодируем категориальные признаки
for col in X.select_dtypes(include="object").columns:
    X[col] = LabelEncoder().fit_transform(X[col])

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

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Преобразование в тензоры
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
y_test = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)

# 2. Определение архитектур
class MLP_Base(nn.Module):
    def __init__(self, input_dim):
```

```
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)

class MLP_Deep(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 16),
            nn.ReLU(),
            nn.Linear(16, 16),
            nn.ReLU(),
            nn.Linear(16, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)

# 3. Инициализация моделей
input_dim = X_train.shape[1]
model_base = MLP_Base(input_dim)
model_deep = MLP_Deep(input_dim)

criterion = nn.BCELoss()
optimizer_base = optim.Adam(model_base.parameters(), lr=0.001)
optimizer_deep = optim.Adam(model_deep.parameters(), lr=0.001)

# 4. Цикл обучения
def train_model(model, optimizer, X_train, y_train, epochs=20):
    for epoch in range(epochs):
        model.train()
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (epoch+1) % 5 == 0:
            print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

print("=== Обучение базовой модели ===")
train_model(model_base, optimizer_base, X_train, y_train)

print("\n=== Обучение глубокой модели ===")
train_model(model_deep, optimizer_deep, X_train, y_train)
```

```
Обучение базовой модели
Epoch 5, Loss: 0.7226
Epoch 10, Loss: 0.7017
Epoch 15, Loss: 0.6820
Epoch 20, Loss: 0.6635

Обучение глубокой модели
Epoch 5, Loss: 0.7002
Epoch 10, Loss: 0.6878
Epoch 15, Loss: 0.6759
Epoch 20, Loss: 0.6641
```

```
# 5. Оценка моделей
def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test)
        y_pred_classes = (y_pred > 0.5).int()
        acc = accuracy_score(y_test, y_pred_classes)
        f1 = f1_score(y_test, y_pred_classes)
    return acc, f1

acc_base, f1_base = evaluate_model(model_base, X_test, y_test)
acc_deep, f1_deep = evaluate_model(model_deep, X_test, y_test)

print("\nРезультаты")
print(f"Базовая модель (1 слой, 32 нейрона): Accuracy = {acc_base:.4f}, F1 =
{f1_base:.4f}")
print(f"Глубокая модель (2 слоя по 16 нейронов): Accuracy = {acc_deep:.4f}, F1
= {f1_deep:.4f}"
```

```
Результаты
Базовая модель (1 слой, 32 нейрона): Accuracy = 0.7634, F1 = 0.0128
Глубокая модель (2 слоя по 16 нейронов): Accuracy = 0.3952, F1 = 0.3607
```

Вывод: Многослойный перцептрон успешно классифицировал уровень дохода на основе демографических данных, а более глубокая модель показала улучшенные метрики точности и полноты по сравнению с базовой архитектурой.