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

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

Выполнил:
Студент 3 курса
Группы АС-66
Гончерёнок К.А.
Проверил:
Крощенко А. А.

Брест 2025

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

Вариант 2

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();

• 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 2 Диагностика рака груди
• Breast Cancer Wisconsin
• Задача: определить, является ли опухоль злокачественной (бинарная классификация).
• Архитектура:
о входной слой;
о один скрытый слой с 16 нейронами (ReLU);
о выходной слой с 1 нейроном (Sigmoid).
• Эксперимент: обучите модель с 32 нейронами в скрытом слое. Как изменились метрики precision и recall?

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
import matplotlib.pyplot as plt
df = pd.read_csv('breast_cancer.csv')
df = df.drop('Unnamed: 32', axis=1)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
X = df.drop(['id', 'diagnosis'], axis=1)
y = df['diagnosis']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
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.values, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
class BreastCancerNN(nn.Module):
    def __init__(self, input_size, hidden_size=16):
        super(BreastCancerNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
```

```
            x = self.sigmoid(x)
            return x
    def train_and_evaluate_model(hidden_size, model_name):
        print(f"=== Модель с {hidden_size} нейронами ===")
        model = BreastCancerNN(input_size=X_train_tensor.shape[1],
hidden_size=hidden_size)
        criterion = nn.BCELoss()
        optimizer = optim.Adam(model.parameters(), lr=0.001)
        epochs = 100
        train_losses = []
        for epoch in range(epochs):
            model.train()
            y_pred = model(X_train_tensor)
            loss = criterion(y_pred, y_train_tensor)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_losses.append(loss.item())
            if (epoch + 1) % 20 == 0:
                print(f'Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}')
        model.eval()
        with torch.no_grad():
            y_pred_proba = model(X_test_tensor)
            y_pred = (y_pred_proba > 0.5).float()
            y_pred_np = y_pred.numpy().flatten()
            y_test_np = y_test_tensor.numpy().flatten()
            accuracy = accuracy_score(y_test_np, y_pred_np)
            precision = precision_score(y_test_np, y_pred_np)
            recall = recall_score(y_test_np, y_pred_np)
            f1 = f1_score(y_test_np, y_pred_np)
            print(f"Accuracy: {accuracy:.4f}")
            print(f"Precision: {precision:.4f}")
            print(f"Recall: {recall:.4f}")
            print(f"F1-Score: {f1:.4f}")
            print(classification_report(y_test_np, y_pred_np,
target_names=['Доброкачественная', 'Злокачественная']))
        return accuracy, precision, recall, f1, train_losses
    acc_16, prec_16, rec_16, f1_16, losses_16 = train_and_evaluate_model(16, "16
нейронов")
    acc_32, prec_32, rec_32, f1_32, losses_32 = train_and_evaluate_model(32, "32
нейрона")
    print("\n" + "=" * 40)
    print("ФИНАЛЬНЫЙ РЕЗУЛЬТАТ:")
    print("=" * 40)
    print(f"16 нейронов: P={prec_16:.3f} R={rec_16:.3f} F1={f1_16:.3f}")
    print(f"32 нейрона:  P={prec_32:.3f} R={rec_32:.3f} F1={f1_32:.3f}")
    print(f"Δ: P={prec_32 - prec_16:+.3f} R={rec_32 - rec_16:+.3f}")
    if rec_32 > rec_16:
        print("✅ Recall вырос - меньше пропусков рака")
    else:
        print("❌ Recall упал")
```

```
16 нейронов: P=0.968 R=0.953 F1=0.961
32 нейрона:  P=0.968 R=0.938 F1=0.952
Δ: P=-0.001 R=-0.016
❌ Recall упал
```

Вывод: построил, обучил и оценил многослойный перцептрон (MLP) для решения задачи классификации.