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-65
Нестюк Н. С.
Проверил:
Крощенко А. А.

Брест 2025

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

Вариант 2

Ход работы:

Задание:

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:
o входной слой;
o один скрытый слой с 16 нейронами (ReLU);
o 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
from sklearn.datasets import load_breast_cancer
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
import numpy as np

data = load_breast_cancer()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

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).view(-1, 1)
y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

class MLP(nn.Module):
    def __init__(self, input_size=30, hidden_size=16):
        super(MLP, 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.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x

model_16 = MLP(hidden_size=16)
criterion = nn.BCELoss()
```

```
optimizer_16 = optim.Adam(model_16.parameters(), lr=0.001)

epochs = 100
for epoch in range(epochs):
    model_16.train()
    optimizer_16.zero_grad()
    outputs = model_16(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer_16.step()

def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test)
        y_pred_class = (y_pred > 0.5).float()

        accuracy = accuracy_score(y_test, y_pred_class)
        precision = precision_score(y_test, y_pred_class)
        recall = recall_score(y_test, y_pred_class)
        f1 = f1_score(y_test, y_pred_class)

    return accuracy, precision, recall, f1

acc_16, prec_16, rec_16, f1_16 = evaluate_model(model_16, X_test, y_test)

print("Модель с 16 нейронами:")
print(f"Accuracy: {acc_16:.4f}, Precision: {prec_16:.4f}, Recall: {rec_16:.4f}, F1-
score: {f1_16:.4f}")

model_32 = MLP(hidden_size=32)
optimizer_32 = optim.Adam(model_32.parameters(), lr=0.001)

for epoch in range(epochs):
    model_32.train()
    optimizer_32.zero_grad()
    outputs = model_32(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer_32.step()

acc_32, prec_32, rec_32, f1_32 = evaluate_model(model_32, X_test, y_test)

print("\nМодель с 32 нейронами:")
print(f"Accuracy: {acc_32:.4f}, Precision: {prec_32:.4f}, Recall: {rec_32:.4f}, F1-
score: {f1_32:.4f}")

print("\nСравнение метрик:")
print(f"Precision: {prec_16:.4f} -> {prec_32:.4f} (изменение: {prec_32 -
prec_16:+.4f})")
print(f"Recall: {rec_16:.4f} -> {rec_32:.4f} (изменение: {rec_32 - rec_16:+.4f})")
```

```
Модель с 16 нейронами:
Accuracy: 0.9737, Precision: 0.9857, Recall: 0.9718, F1-score: 0.9787

Модель с 32 нейронами:
Accuracy: 0.9737, Precision: 0.9857, Recall: 0.9718, F1-score: 0.9787

Сравнение метрик:
Precision: 0.9857 -> 0.9857 (изменение: +0.0000)
Recall: 0.9718 -> 0.9718 (изменение: +0.0000)
```

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