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

Лабораторная работа №4
По дисциплине: «ОМО»
Тема:” Введение в нейронные сети:
построение многослойного перцептрона”

Выполнил:
Студент 3-го курса
Группы АС-66
Янчук А.Ю.
Проверил:
Крощенко А.А.

Брест 2025

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

Вариант 13

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 13:
Оценка безопасности автомобиля
• Car Evaluation
• Задача: оценить безопасность автомобиля (4 класса).
• Архитектура:
o входной слой;
o два скрытых слоя: первый с 16 нейронами, второй с 8 (ReLU);
o выходной слой с 4 нейронами (Softmax).
• Эксперимент: обучите модель с одним скрытым слоем на 24 нейрона. Какая архитектура показала себя лучше?

```
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

columns = ["buying","maint","doors","persons","lug_boot","safety","class"]
df = pd.read_csv("car_evaluation.csv", names=columns)

for col in df.columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

X = df.drop("class", axis=1).values
y = df["class"].values

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.long)
y_test  = torch.tensor(y_test, dtype=torch.long)

class MLP_2hidden(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(6, 16)
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, 4)
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
```

```
        x = self.fc3(x)
        return x

class MLP_1hidden(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(6, 24)
        self.fc2 = nn.Linear(24, 4)
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

def train_model(model, X_train, y_train, X_test, y_test, epochs=50):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    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()

    model.eval()
    with torch.no_grad():
        y_pred_test = model(X_test)
        y_pred_classes = torch.argmax(y_pred_test, dim=1)

    acc = accuracy_score(y_test, y_pred_classes)
    f1 = f1_score(y_test, y_pred_classes, average="weighted")
    return acc, f1

modelA = MLP_2hidden()
accA, f1A = train_model(modelA, X_train, y_train, X_test, y_test)

modelB = MLP_1hidden()
accB, f1B = train_model(modelB, X_train, y_train, X_test, y_test)

print("2 скрытых слоя (16→8): Accuracy =", round(accA, 4), "F1 =", round(f1A,
4))
print("1 скрытый слой (24): Accuracy =", round(accB, 4), "F1 =", round(f1B,
4))
```

Результат:

```
2 скрытых слоя (16→8): Accuracy = 0.2312 F1 = 0.2915
1 скрытый слой (24): Accuracy = 0.6676 F1 = 0.5707
```

Вывод: На практике построили, обучили и оценили многослойный перцептрон (MLP) для решения задачи классификации.