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
Макарский А. Э.
Проверил:
Крощенко А. А.

Брест 2025

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

Вариант 10

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 10 Прогнозирование уровня дохода
• Adult Census Income
• Задача: предсказать, превышает ли доход $50 тыс. в год (бинарная классификация).
• Архитектура:
o входной слой;
o один скрытый слой с 32 нейронами (ReLU);
o 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
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, f1_score, classification_report
data = pd.read_csv('adult.csv')
data = data.replace('?', np.nan)
data = data.dropna()

categorical_columns = ['workclass', 'education', 'marital.status', 'occupation',
                       'relationship', 'race', 'sex', 'native.country']

label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

le_income = LabelEncoder()
```

```
data['income'] = le_income.fit_transform(data['income'])


X = data.drop('income', axis=1)

y = data['income']


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_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).unsqueeze(1)

y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).unsqueeze(1)

class SimpleMLP(nn.Module):

    def __init__(self, input_size):

        super(SimpleMLP, self).__init__()

        self.layer1 = nn.Linear(input_size, 32)

        self.relu = nn.ReLU()

        self.output = nn.Linear(32, 1)

        self.sigmoid = nn.Sigmoid()


    def forward(self, x):

        x = self.relu(self.layer1(x))

        x = self.sigmoid(self.output(x))

        return x


class DeepMLP(nn.Module):

    def __init__(self, input_size):

        super(DeepMLP, self).__init__()

        self.layer1 = nn.Linear(input_size, 16)

        self.layer2 = nn.Linear(16, 16)

        self.relu = nn.ReLU()

        self.output = nn.Linear(16, 1)

        self.sigmoid = nn.Sigmoid()


    def forward(self, x):
```

```
            x = self.relu(self.layer1(x))
            x = self.relu(self.layer2(x))
            x = self.sigmoid(self.output(x))
            return x


    def train_and_evaluate(model, model_name, X_train, y_train, X_test, y_test,
epochs=100):
        print(f"\n=== Обучение модели: {model_name} ===")

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

        train_losses = []
        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()

            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)
            y_pred = (y_pred_proba > 0.5).float()

            accuracy = accuracy_score(y_test.numpy(), y_pred.numpy())
            f1 = f1_score(y_test.numpy(), y_pred.numpy())

            print(f"\nРезультаты для {model_name}:")
            print(f"Accuracy: {accuracy:.4f}")
            print(f"F1-Score: {f1:.4f}")
```

```
        print("\nClassification Report:")
        print(classification_report(y_test.numpy(), y_pred.numpy()))
    return accuracy, f1, train_losses
input_size = X_train.shape[1]
simple_model = SimpleMLP(input_size)
acc_simple, f1_simple, losses_simple = train_and_evaluate(
    simple_model, "Простая модель (1 слой, 32 нейрона)",
    X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor
)
deep_model = DeepMLP(input_size)
acc_deep, f1_deep, losses_deep = train_and_evaluate(
    deep_model, "Глубокая модель (2 слоя, 16 нейронов)",
    X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor
)
print("\n" + "=" * 50)
print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ")
print("=" * 50)
print(f"Простая модель (1 слой, 32 нейрона):")
print(f"  Accuracy: {acc_simple:.4f}, F1-Score: {f1_simple:.4f}")
print(f"Глубокая модель (2 слоя, 16 нейронов):")
print(f"  Accuracy: {acc_deep:.4f}, F1-Score: {f1_deep:.4f}")
if acc_simple > acc_deep:
    best_model = "Простая модель"
    best_acc = acc_simple
else:
    best_model = "Глубокая модель"
    best_acc = acc_deep
print(f"\nЛучшая модель: {best_model} с Accuracy: {best_acc:.4f}")
print(f"\nИнформация о данных:")
print(f"Размерность признаков: {input_size}")
print(f"Размер обучающей выборки: {X_train.shape[0]}")
print(f"Размер тестовой выборки: {X_test.shape[0]}")
print(f"Распределение классов в целевой переменной:")
print(f"  <=50K: {sum(y == 0)} samples")
print(f"  >50K: {sum(y == 1)} samples")
```

```
==================================================
СРАВНЕНИЕ РЕЗУЛЬТАТОВ
==================================================
Простая модель (1 слой, 32 нейрона):
  Accuracy: 0.8019, F1-Score: 0.4444
Глубокая модель (2 слоя, 16 нейронов):
  Accuracy: 0.7623, F1-Score: 0.0901

Лучшая модель: Простая модель с Accuracy: 0.8019

Информация о данных:
Размерность признаков: 14
Размер обучающей выборки: 24129
Размер тестовой выборки: 6033
Распределение классов в целевой переменной:
  <=50K: 22654 samples
  >50K: 7508 samples
```

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