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-65
Гуща И.В.
Вариант 3
Проверил:
Крощенко А.А.

Брест 2025

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

Ход работы

Вариант 3

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 (напрмер, 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 3 Классификация качества вина

- Набор данных: Wine Quality
- Задача: классифицировать вино на "хорошее" (оценка >= 7) и "обычное" (бинарная классификация).
- Архитектура:

o входной слой;
o два скрытых слоя по 12 нейронов в каждом (ReLU);
o выходной слой с 1 нейроном (Sigmoid).
• Эксперимент: увеличьте количество эпох обучения в два раза и посмотрите, привело ли это к улучшению F1 -score.

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, preci-
sion_score,recall_score
from sklearn.utils.class_weight import compute_class_weight
import pandas as pd
import numpy as np

data = pd.read_csv('winequality-white.csv', sep=';')
data['good'] = (data['quality'] >= 7).astype(int)

X = data.drop(['quality', 'good'], axis=1).values
y = data['good'].values

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

# Разделение данных на train/test
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
# 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.reshape(-1, 1), dtype=torch.float32)
y_test = torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)
class WineQualityNet(nn.Module):
    def __init__(self, input_dim):
        super(WineQualityNet, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 12),
            nn.ReLU(),
            nn.Linear(12, 12),
            nn.ReLU(),
            nn.Linear(12, 1)
)

    def forward(self, x):
        return self.model(x)
class_weights = compute_class_weight('balanced', classes=np.unique(y), y=y)
pos_weight = torch.tensor(class_weights[1] / class_weights[0],
dtype=torch.float32)
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
```

```
model = WineQualityNet(X_train.shape[1])
optimizer = optim.Adam(model.parameters(), lr=0.001)
def train_model(model, X_train, y_train, epochs=50):
    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) % 10 == 0:
            print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

train_model(model, X_train, y_train, epochs=50)
```

```
Epoch [10/50], Loss: 1.0824
Epoch [20/50], Loss: 1.0703
Epoch [30/50], Loss: 1.0575
Epoch [40/50], Loss: 1.0430
Epoch [50/50], Loss: 1.0263
```

```
def evaluate_model(model, X_test, y_test):
    model.eval()
    with torch.no_grad():
        logits = model(X_test)
        probs = torch.sigmoid(logits)
        preds = (probs > 0.5).float()

    acc = accuracy_score(y_test.numpy(), preds.numpy())
    f1 = f1_score(y_test.numpy(), preds.numpy())
    precision = precision_score(y_test.numpy(), preds.numpy())
    recall = recall_score(y_test.numpy(),preds.numpy())
    print(f"Accuracy: {acc:.4f}, F1-score: {f1:.4f}, Precision: { precision:
.4f}, Recall: { recall: .4f}  ")
    return acc, f1

print("Оценка после 50 эпох:")
evaluate_model(model, X_test, y_test)
```

```
Оценка после 50 эпох:
Accuracy: 0.7224, F1-score: 0.5018, Precision:  0.4102, Recall:  0.6462
(0.7224489795918367, 0.5018315018315018)
```

```
model2 = WineQualityNet(X_train.shape[1])
optimizer = optim.Adam(model2.parameters(), lr=0.001)

train_model(model2, X_train, y_train, epochs=100)

print("Оценка после 100 эпох:")
evaluate_model(model2, X_test, y_test)
```

```
Epoch [10/100], Loss: 1.0641
Epoch [20/100], Loss: 1.0466
Epoch [30/100], Loss: 1.0260
Epoch [40/100], Loss: 1.0016
Epoch [50/100], Loss: 0.9756
Epoch [60/100], Loss: 0.9496
Epoch [70/100], Loss: 0.9249
Epoch [80/100], Loss: 0.9023
Epoch [90/100], Loss: 0.8823
Epoch [100/100], Loss: 0.8650
Оценка после 100 эпох:
Accuracy: 0.7378, F1-score: 0.5196, Precision:  0.4303, Recall:  0.6557
(0.7377551020408163, 0.5196261682242991)
```

Вывод: в ходе работы была построена и обучена нейросеть для определения качества вина. При увеличении количества эпох в два раза возрастают показатели метрик для оценки нейросети.