

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 перцептрана»

Выполнил:
Студент 3-го курса
Группы АС-65
Осовец М. М.
Проверил:
Крощенко А. А.

Брест 2025

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

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 (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 3 Классификация качества вина

- Набор данных: Wine Quality
- Задача: классифицировать вино на "хорошее" (оценка ≥ 7) и "обычное" (бинарная классификация).
- Архитектура:
 - о входной слой;
 - о два скрытых слоя по 12 нейронов в каждом (ReLU);
 - о выходной слой с 1 нейроном (Sigmoid).
- Эксперимент: увеличьте количество эпох обучения в два раза и посмотрите, привело ли это к улучшению F1-score

Импорт библиотек и подготовка данных

```
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  
from sklearn.metrics import accuracy_score, f1_score, classification_report  
import pandas as pd  
import numpy as np  
  
# Загрузка датасета
```

```

data = pd.read_csv("winequality-white.csv", sep=";")

# Создаем бинарную целевую переменную: 1 - хорошее (>=7), 0 - обычное (<7)
data["good"] = (data["quality"] >= 7).astype(int)

X = data.drop(columns=["quality", "good"])
y = data["good"]

# Делим данные на train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Стандартизация признаков
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Преобразуем в тензоры
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)

print("Форма обучающих данных:", X_train_tensor.shape)
print("Форма тестовых данных:", X_test_tensor.shape)

```

Форма обучающих данных: torch.Size([3918, 11])
Форма тестовых данных: torch.Size([980, 11])

Определение архитектуры нейронной сети

```

class WineMLP(nn.Module):
    def __init__(self, input_dim):
        super(WineMLP, self).__init__()
        self.hidden1 = nn.Linear(input_dim, 12)
        self.hidden2 = nn.Linear(12, 12)
        self.output = nn.Linear(12, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.hidden1(x))
        x = self.relu(self.hidden2(x))
        x = self.sigmoid(self.output(x))
        return x

input_dim = X_train.shape[1]
model = WineMLP(input_dim)
Model
WineMLP(
  (hidden1): Linear(in_features=11, out_features=12, bias=True)
  (hidden2): Linear(in_features=12, out_features=12, bias=True)
  (output): Linear(in_features=12, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
)

```

Инициализация модели, функции потерь и оптимизатора

```

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

```

Обучение модели

EPOCHS = 50

```

for epoch in range(EPOCHS):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f"Эпоха [{epoch+1}/{EPOCHS}], Потери: {loss.item():.4f}")

Эпоха [10/50], Потери: 0.6981
Эпоха [20/50], Потери: 0.6807
Эпоха [30/50], Потери: 0.6646
Эпоха [40/50], Потери: 0.6479
Эпоха [50/50], Потери: 0.6294

```

Оценка модели

```

model.eval()
with torch.no_grad():
    y_pred = model(X_test_tensor)
    y_pred_cls = (y_pred > 0.5).float()

acc = accuracy_score(y_test_tensor, y_pred_cls)
f1 = f1_score(y_test_tensor, y_pred_cls)

print("\n--- Результаты модели ---")
print(f"Accuracy: {acc:.4f}")
print(f"F1-score: {f1:.4f}")
print(classification_report(y_test_tensor, y_pred_cls, digits=4, zero_division=0))

--- Результаты модели ---
Accuracy: 0.7837
F1-score: 0.0000
      precision    recall   f1-score   support
0.0      0.7837    1.0000    0.8787     768
1.0      0.0000    0.0000    0.0000     212

      accuracy          0.7837     980
      macro avg      0.3918    0.5000    0.4394     980
  weighted avg      0.6141    0.7837    0.6886     980

```

Эксперимент: увеличение числа эпох в 2 раза

```

model2 = WineMLP(input_dim)
optimizer2 = optim.Adam(model2.parameters(), lr=0.001)
EPOCHS2 = EPOCHS * 2

for epoch in range(EPOCHS2):
    model2.train()
    optimizer2.zero_grad()
    outputs = model2(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer2.step()

model2.eval()
with torch.no_grad():
    y_pred2 = model2(X_test_tensor)
    y_pred2_cls = (y_pred2 > 0.5).float()

acc2 = accuracy_score(y_test_tensor, y_pred2_cls)

```

```
f12 = f1_score(y_test_tensor, y_pred2_cls)

print("\n--- Эксперимент (эпохи x2) ---")
print(f"Accuracy: {acc2:.4f}")
print(f"F1-score: {f12:.4f}")
print(classification_report(y_test_tensor, y_pred2_cls, digits=4, zero_division=0))
```

```
--- Эксперимент (эпохи x2) ---
Accuracy: 0.7837
F1-score: 0.0000
      precision    recall  f1-score   support
0.0      0.7837  1.0000   0.8787     768
1.0      0.0000  0.0000   0.0000     212

accuracy                           0.7837     980
macro avg                           0.3918     0.5000   0.4394     980
weighted avg                         0.6141   0.7837   0.6886     980
```

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