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

Лабораторная работа №4
По дисциплине «ОМО»

Выполнил:
Студент 3-го курса
Группы АС-65
Егоренков Н. Д.
Проверил:
Крощенко А.А.

Брест 2025

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

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 5 Определение съедобности грибов

- Mushroom Classification
- **Задача:** определить, является ли гриб ядовитым (бинарная классификация).
- **Архитектура:**
  - входной слой;
  - один скрытый слой с 8 нейронами (ReLU);
  - выходной слой с 1 нейроном (Sigmoid).
- **Эксперимент:** попробуйте использовать tanh в качестве функции активации в скрытом слое вместо ReLU. Сравните итоговую точность.

Ход работы

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classification_report

df = pd.read_csv('mushrooms.csv')

label_encoders = {}
for column in df.columns:
    if df[column].dtype == 'object':
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column])
        label_encoders[column] = le

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

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

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

X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, 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 MushroomClassifier(nn.Module):
    def __init__(self, input_size, hidden_size=8, activation='relu'):
        super(MushroomClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, 1)

        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'tanh':
            self.activation = nn.Tanh()
        else:
            self.activation = nn.ReLU()

        self.sigmoid = nn.Sigmoid()
```

```
    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x


input_size = X_train_tensor.shape[1]

model_relu = MushroomClassifier(input_size=input_size, hidden_size=8,
activation='relu')
criterion = nn.BCELoss()
optimizer_relu = optim.Adam(model_relu.parameters(), lr=0.001)

model_tanh = MushroomClassifier(input_size=input_size, hidden_size=8,
activation='tanh')
optimizer_tanh = optim.Adam(model_tanh.parameters(), lr=0.001)

def train_model(model, optimizer, X_train, y_train, epochs=100):
    model.train()
    losses = []

    for epoch in range(epochs):
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        losses.append(loss.item())

    return losses

losses_relu = train_model(model_relu, optimizer_relu, X_train_tensor, y_train_tensor,
epochs=100)

losses_tanh = train_model(model_tanh, optimizer_tanh, X_train_tensor, y_train_tensor,
epochs=100)


def evaluate_model(model, X_test, y_test):
    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())

        return accuracy, f1, y_pred


print("\nОценка модели с ReLU:")
accuracy_relu, f1_relu, y_pred_relu = evaluate_model(model_relu, X_test_tensor,
y_test_tensor)
print(f"Accuracy: {accuracy_relu:.4f}")
print(f"F1-Score: {f1_relu:.4f}")
```

```
Оценка модели с ReLU:
Accuracy: 0.8954
F1-Score: 0.8818
```

```
print("\nОценка модели с Tanh:")
accuracy_tanh, f1_tanh, y_pred_tanh = evaluate_model(model_tanh, X_test_tensor,
y_test_tensor)
print(f"Accuracy: {accuracy_tanh:.4f}")
print(f"F1-Score: {f1_tanh:.4f}")
```

```
Оценка модели с Tanh:
Accuracy: 0.8745
F1-Score: 0.8625
```

```
print("\nСравнение результатов:")
print(f"ReLU - Accuracy: {accuracy_relu:.4f}, F1-Score: {f1_relu:.4f}")
print(f"Tanh - Accuracy: {accuracy_tanh:.4f}, F1-Score: {f1_tanh:.4f}")
```

```
Сравнение результатов:
ReLU - Accuracy: 0.8954, F1-Score: 0.8818
Tanh - Accuracy: 0.8745, F1-Score: 0.8625
```

```
print("\nДетальный отчет классификации для модели с ReLU:")
print(classification_report(y_test_tensor.numpy(), y_pred_relu.numpy(),
target_names=['Ядовитый', 'Съедобный']))
```

```
Детальный отчет классификации для модели с ReLU:
              precision    recall  f1-score   support

    Ядовитый       0.85      0.98      0.91       842
   Съедобный       0.97      0.81      0.88       783

    accuracy                           0.90      1625
   macro avg       0.91      0.89      0.89      1625
weighted avg       0.90      0.90      0.89      1625
```

```
print("\nДетальный отчет классификации для модели с Tanh:")
print(classification_report(y_test_tensor.numpy(), y_pred_tanh.numpy(),
target_names=['Ядовитый', 'Съедобный']))
```

```
Детальный отчет классификации для модели с Tanh:
              precision    recall  f1-score   support

    Ядовитый       0.85      0.93      0.88       842
   Съедобный       0.91      0.82      0.86       783

    accuracy                           0.87      1625
   macro avg       0.88      0.87      0.87      1625
weighted avg       0.88      0.87      0.87      1625
```

Вывод: мы научились строить, обучать и оценивать многослойный перцептрон