

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-66
Осовец А.О.
Проверил:
Крощенко А.А.

Брест 2025

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

Вариант 7

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:
 - переведите модель в режим обучения: model.train();
 - сделайте предсказание (forward pass):

```
y_pred = model(X_train);
```

- рассчитайте потери (loss):

```
loss = criterion(y_pred, y_train);
```

- обнулите градиенты: optimizer.zero_grad();
- выполните обратное распространение ошибки:

```
loss.backward();
```

- 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`.

```

import os
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, classification_report
import pandas as pd

# 1. Загрузка датасета
current_dir = os.path.dirname(os.path.abspath(__file__))
project_root = os.path.abspath(os.path.join(current_dir, '../..', '..', '..'))
file_path = os.path.join(project_root, 'Telco-Customer-Churn.csv')

data = pd.read_csv(file_path)

data = data.dropna()
data = data.drop(columns=["customerID"])

y = data["Churn"]
X = data.drop(columns=["Churn"])

for col in X.select_dtypes(include=["object"]).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

y = (y == "Yes").astype(int)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
y_test = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)

input_dim = X_train.shape[1]

# 2. Определение архитектуры нейронной сети (без Dropout)
class MLP(nn.Module):
    def __init__(self, input_dim):
        super(MLP, self).__init__()
        self.hidden = nn.Linear(input_dim, 24)
        self.relu = nn.ReLU()
        self.output = nn.Linear(24, 1)

    def forward(self, x):
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        return x

```

```

# 3. Инициализация модели, функции потерь и оптимизатора
model = MLP(input_dim)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 4. Цикл обучения
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"Эпоха [{epoch + 1}/{epochs}], Потери: {loss.item():.4f}")

# 5. Оценка модели
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test)
    y_pred_classes = (torch.sigmoid(y_pred_test) > 0.5).float()

accuracy = accuracy_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
print("\n==== Результаты без Dropout ===")
print(f"Accuracy: {accuracy:.4f}")
print(f"F1-score: {f1:.4f}")
print(classification_report(y_test, y_pred_classes))

```

Эксперимент: добавляем Dropout 0.2

```

class MLP_Dropout(nn.Module):
    def __init__(self, input_dim):
        super(MLP_Dropout, self).__init__()
        self.hidden = nn.Linear(input_dim, 24)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.2)
        self.output = nn.Linear(24, 1)

    def forward(self, x):
        x = self.hidden(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.output(x)
        return x

model_dropout = MLP_Dropout(input_dim)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model_dropout.parameters(), lr=0.001)

for epoch in range(epochs):
    model_dropout.train()
    y_pred = model_dropout(X_train)
    loss = criterion(y_pred, y_train)

    optimizer.zero_grad()
    loss.backward()

```

```
optimizer.step()

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

model_dropout.eval()
with torch.no_grad():
    y_pred_test = model_dropout(X_test)
    y_pred_classes = (torch.sigmoid(y_pred_test) > 0.5).float()

accuracy_d = accuracy_score(y_test, y_pred_classes)
f1_d = f1_score(y_test, y_pred_classes)
print("\n==== Результаты с Dropout(0.2) ====")
print(f"Accuracy: {accuracy_d:.4f}")
print(f"F1-score: {f1_d:.4f}")
print(classification_report(y_test, y_pred_classes))

print("\n==== Сравнение ====")
print(f"Без Dropout: Accuracy={accuracy:.4f}, F1={f1:.4f}")
print(f"C Dropout: Accuracy={accuracy_d:.4f}, F1={f1_d:.4f}")
if f1_d > f1:
    print("    Dropout помог улучшить результат!")
else:
    print("    Dropout не улучшил результат.")
```

Результат:

```
Эпоха [10/50], Потери: 0.7022
Эпоха [20/50], Потери: 0.6660
Эпоха [30/50], Потери: 0.6344
Эпоха [40/50], Потери: 0.6066
Эпоха [50/50], Потери: 0.5818
```

```
== Результаты без Dropout ==
```

```
Accuracy: 0.7431
```

```
F1-score: 0.4532
```

	precision	recall	f1-score	support
0.0	0.80	0.87	0.83	1035
1.0	0.52	0.40	0.45	374
accuracy			0.74	1409
macro avg	0.66	0.63	0.64	1409
weighted avg	0.73	0.74	0.73	1409

```
Эпоха [10/50] (Dropout), Потери: 0.6281
```

```
Эпоха [20/50] (Dropout), Потери: 0.5991
```

```
Эпоха [30/50] (Dropout), Потери: 0.5730
```

```
Эпоха [40/50] (Dropout), Потери: 0.5488
```

```
Эпоха [50/50] (Dropout), Потери: 0.5280
```

```
== Результаты с Dropout(0.2) ==
```

```
Accuracy: 0.7544
```

```
F1-score: 0.3472
```

	precision	recall	f1-score	support
0.0	0.77	0.94	0.85	1035
1.0	0.59	0.25	0.35	374
accuracy			0.75	1409
macro avg	0.68	0.59	0.60	1409
weighted avg	0.73	0.75	0.72	1409

```
== Сравнение ==
```

```
Без Dropout: Accuracy=0.7431, F1=0.4532
```

```
С Dropout: Accuracy=0.7544, F1=0.3472
```

```
✗ Dropout не улучшил результат.
```

Вывод: На практике сравнили работу нескольких алгоритмов классификации, таких как метод k-ближайших соседей (k-NN), деревья решений и метод опорных векторов (SVM). Научились подбирать гиперпараметры моделей и оценивать их влияние на результат.