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»**

Выполнила:
Студентка 3 курса
Группы АС-65
Степанова Д. А.
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:
    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 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.

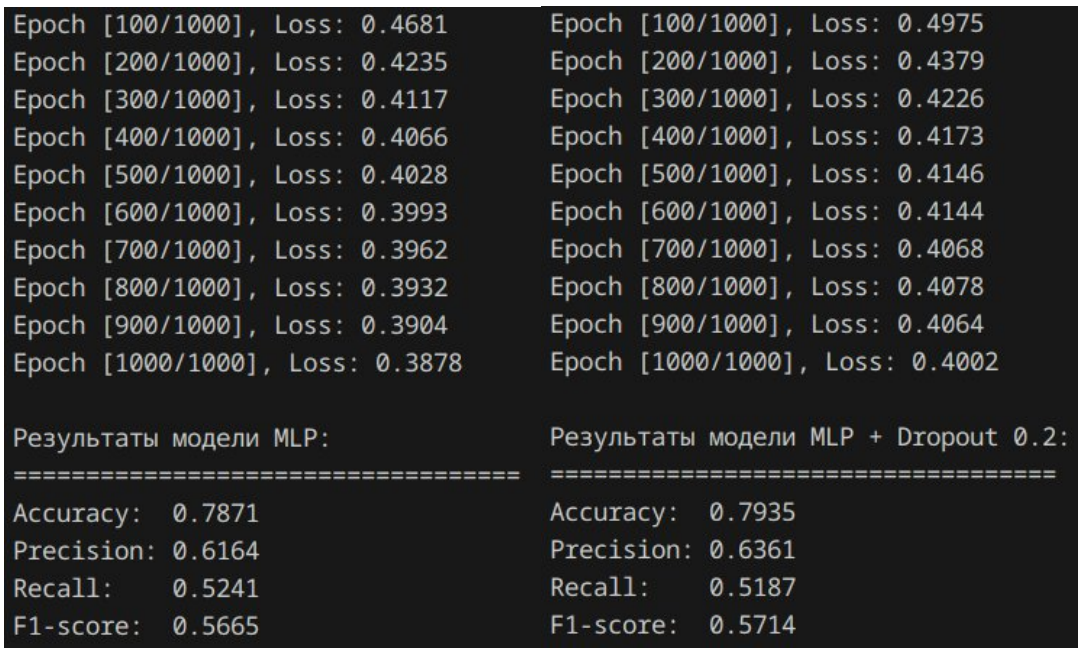
Прогнозирование оттока клиентов

- Telco Customer Churn
- **Задача:** предсказать, уйдет ли клиент (бинарная классификация).
- **Архитектура:**
  - входной слой;
  - один скрытый слой с 24 нейронами (ReLU);
  - выходной слой с 1 нейроном (Sigmoid).
- **Эксперимент:** добавьте в модель слой Dropout с коэффициентом 0.2 после скрытого слоя и сравните результаты. Помогло ли это?

Код программы:

```
import pandas as pd
import numpy as np
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, precision_score, recall_score,
f1_score

# 1. Импорт библиотек и подготовка данных
df = pd.read_csv("Telco-Customer-Churn.csv")

df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")
df["TotalCharges"] = df["TotalCharges"].fillna(df["TotalCharges"].median())

df["Churn"] = df["Churn"].map({"Yes": 1, "No": 0})
df = df.drop("customerID", axis=1)

X = df.drop("Churn", axis=1)
Y = df["Churn"]

X = pd.get_dummies(X, drop_first=True)

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_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)

input_dim = X_train.shape[1]


# 2. Определение архитектуры нейронной сети
class MLP_NoDropout(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(input_dim, 24)
        self.relu = nn.ReLU()
        self.output = nn.Linear(24, 1)
```

```
    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.output(x)
        return x

class MLP_Dropout(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = 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.relu(self.layer1(x))
        x = self.dropout(x)
        x = self.output(x)
        return x

# 3. Инициализация модели, функции потерь и оптимизатора
criterion = nn.BCEWithLogitsLoss()

def train_and_evaluate(model, name, epochs=1000):
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # 4. Написание цикла обучения (Training Loop)
    for epoch in range(1, epochs + 1):
        model.train()
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)

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

        if epoch % 100 == 0:
            print(f"Epoch [{epoch}/{epochs}], Loss: {loss.item():.4f}")

    # 5. Оценка модели (Evaluation)
    model.eval()
    with torch.no_grad():
        logits = model(X_test).numpy()
        preds = (torch.sigmoid(torch.tensor(logits)) >
0.5).numpy().astype(int)

    acc = accuracy_score(y_test, preds)
    prec = precision_score(y_test, preds)
    rec = recall_score(y_test, preds)
    f1 = f1_score(y_test, preds)

    print(f"\nРезультаты модели {name}:")
    print("==================================")
    print(f"Accuracy:  {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall:    {rec:.4f}")
    print(f"F1-score:  {f1:.4f}\n")

    return acc, prec, rec, f1


model1 = MLP_NoDropout()
results1 = train_and_evaluate(model1, "MLP")

model2 = MLP_Dropout()
results2 = train_and_evaluate(model2, "MLP + Dropout 0.2")
```



После добавления Dropout (0.2) метрики модели немного улучшились:

Метрика	**Было**	**Стало**	**Разница (%)**
Accuracy	0.7871	0.7935	+ 0.81
Precision	0.6164	0.6361	+ 3.20
Recall	0.5241	0.5187	- 1.03
F1-score	0.5665	0.5714	+ 0.86

Dropout позволил повысить общую точность и качество классификации (F1-score вырос). Это говорит о небольшом снижении переобучения и более устойчивой работе модели. Улучшение хоть и незначительное, но Dropout оказал положительный эффект.

Вывод: построила, обучила и оценила многослойный перцептрон (MLP) для решения задачи классификации.