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
Шлейхер А. С.
Проверил:
Крощенко А. А.

Брест 2025

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

Вариант 10

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.

Вариант 10 Прогнозирование уровня дохода

- Adult Census Income
- **Задача:** предсказать, превышает ли доход $50 тыс. в год (бинарная классификация).
- **Архитектура:**
  - входной слой;
  - один скрытый слой с 32 нейронами (ReLU);
  - выходной слой с 1 нейроном (Sigmoid).
- **Эксперимент:** сравните производительность с более глубокой моделью: два скрытых слоя по 16 нейронов.

Код программы:

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


# 1. Импорт библиотек и подготовка данных
df = pd.read_csv("adult.csv")

df.replace("?", pd.NA, inplace=True)
df.dropna(inplace=True)

df['income'] = df['income'].apply(lambda x: 1 if x.strip() == ">50K" else 0)

cat_cols = df.select_dtypes(include=['object']).columns
encoder = LabelEncoder()
for col in cat_cols:
    df[col] = encoder.fit_transform(df[col])


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

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_t = torch.tensor(X_train_scaled, dtype=torch.float32)
X_test_t = torch.tensor(X_test_scaled, dtype=torch.float32)
y_train_t = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
y_test_t = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
```

```
# 2. Определение архитектуры нейронной сети
class MLP_1(nn.Module): # 32x1
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 32)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))

class MLP_2(nn.Module): # 16x2
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 16)
        self.fc2 = nn.Linear(16, 16)
        self.relu = nn.ReLU()
        self.fc3 = nn.Linear(16, 1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        return self.fc3(x)

input_dim = X_train_t.shape[1]


# 3. Инициализация модели, функции потерь и оптимизатора
def train_and_evaluate(model, name, epochs=1000, lr=0.001):
    print(f"\nобучение модели: {name}")

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


    # 4. Написание цикла обучения (Training Loop)
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()

        logits = model(X_train_t)
        loss = criterion(logits, y_train_t)
        loss.backward()
        optimizer.step()

        if (epoch + 1) % 200 == 0:
            print(f"Epoch {epoch+1}/{epochs}, Loss = {loss.item():.4f}")


    # 5. Оценка модели (Evaluation)
    model.eval()
    with torch.no_grad():
        test_logits = model(X_test_t)
        pred = (torch.sigmoid(test_logits) > 0.5).float()
```

```
    acc = accuracy_score(y_test, pred.numpy())
    prec = precision_score(y_test, pred.numpy())
    f1 = f1_score(y_test, pred.numpy())

    print(f"Accuracy:  {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"F1-score:  {f1:.4f}")
    return acc, prec, f1


results = {}
results["MLP_1"] = train_and_evaluate(MLP_1(input_dim), "один скрытый слой с
32 нейронами")
results["MLP_2"] = train_and_evaluate(MLP_2(input_dim), "два скрытых слоя по
16 нейронов")
print("\nсравнение:")
for name, (acc, prec, f1) in results.items():
    print(f"{name}: Accuracy={acc:.4f}, Precision={prec:.4f}, F1={f1:.4f}")
```

```
обучение модели: один скрытый слой с 32 нейронами
Epoch 200/1000, Loss = 0.4126
Epoch 400/1000, Loss = 0.3688
Epoch 600/1000, Loss = 0.3427
Epoch 800/1000, Loss = 0.3344
Epoch 1000/1000, Loss = 0.3314
Accuracy:  0.8516
Precision: 0.7553
F1-score:  0.6674

обучение модели: два скрытых слоя по 16 нейронов
Epoch 200/1000, Loss = 0.3773
Epoch 400/1000, Loss = 0.3365
Epoch 600/1000, Loss = 0.3306
Epoch 800/1000, Loss = 0.3268
Epoch 1000/1000, Loss = 0.3238
Accuracy:  0.8487
Precision: 0.7373
F1-score:  0.6672
```

```
сравнение:
MLP_1: Accuracy=0.8516, Precision=0.7553, F1=0.6674
MLP_2: Accuracy=0.8487, Precision=0.7373, F1=0.6672
```

Добавление второго скрытого слоя не улучшило качество классификации. Наоборот, небольшое снижение точности и precision говорит о том, что более глубокая модель в данном случае не даёт преимуществ. Это может быть связано с тем, что данных и так достаточно для успешного обучения более простой архитектуры, и усложнение модели не приводит к росту обобщающей способности.

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