МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №4

Специальность АС-66

Выполнил
А. С. Рогожин,
студент группы АС-66

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«___» _____2025 г.

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 10 Прогнозирование уровня дохода

• Adult Census Income

• Задача: предсказать, превышает ли доход $50 тыс. в год (бинарная классификация).

• Архитектура:

о входной слой;

о один скрытый слой с 32 нейронами (ReLU);

о выходной слой с 1 нейроном (Sigmoid).

• Эксперимент: сравните производительность с более глубокой моделью: два скрытых слоя по 16 нейронов.

```
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, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, f1_score, classification_report
import pandas as pd
import numpy as np
import json

CSV_NAME = "adult.csv"
OUT_MODEL_1 = "best_model_mlp1.pt"
OUT_MODEL_2 = "best_model_mlp2.pt"
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", DEVICE)
script_dir = os.path.dirname(os.path.abspath(__file__))
csv_path = os.path.join(script_dir, CSV_NAME)
if not os.path.exists(csv_path):
    raise FileNotFoundError(f"CSV not found: {csv_path}")

df = pd.read_csv(csv_path)
df['income'] = df['income'].astype(str).str.strip()
df['income'] = df['income'].replace({'>50K.': '>50K', '<=50K.': '<=50K'})
df['target'] = (df['income'] == '>50K').astype(int)
df = df.drop(columns=['income'])

X = df.drop(columns=['target'])
y = df['target']

num_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
cat_cols = X.select_dtypes(include=['object']).columns.tolist()

preprocessor = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    ("cat", OneHotEncoder(handle_unknown='ignore', sparse_output=True), cat_cols)
])
X_prepared = preprocessor.fit_transform(X)
if hasattr(X_prepared, "toarray"):
    X_prepared = X_prepared.toarray()

X_train, X_test, y_train, y_test = train_test_split(
    X_prepared, y.values, test_size=0.2, random_state=42, stratify=y.values
)

X_train_t = torch.tensor(X_train, dtype=torch.float32)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.float32).view(-1,1)
```

```
y_test_t = torch.tensor(y_test, dtype=torch.float32).view(-1,1)
input_size = X_train_t.shape[1]

class MLP_1Hidden(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_size, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )
    def forward(self, x):
        return self.net(x)

class MLP_2Hidden(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_size, 16),
            nn.ReLU(),
            nn.Linear(16, 16),
            nn.ReLU(),
            nn.Linear(16, 1)
        )
    def forward(self, x):
        return self.net(x)

def train_mlp(model, X_train_tensor, y_train_tensor,
              X_valid_tensor, y_valid_tensor,
              X_test_tensor, y_test_tensor,
              epochs=500, lr=1e-3, batch_size=128, patience=40,
              out_path='best_model.pt'):

    model = model.to(DEVICE)
    y_train_np = y_train_tensor.cpu().numpy().ravel()
    pos = (y_train_np == 1).sum()
    neg = (y_train_np == 0).sum()
    pos_weight = torch.tensor([neg / (pos + 1e-12)],
dtype=torch.float32).to(DEVICE)
    criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max',
factor=0.5, patience=8)

    best_val_f1 = -1.0
    epochs_no_improve = 0
    best_epoch = 0

    X_valid_device = X_valid_tensor.to(DEVICE)
    y_valid_device = y_valid_tensor.to(DEVICE)
    X_test_device = X_test_tensor.to(DEVICE)

    N = X_train_tensor.shape[0]

    for epoch in range(1, epochs+1):
        model.train()
        batch_losses = []
        pos_idx = torch.where(y_train_tensor.view(-1)==1)[0]
        neg_idx = torch.where(y_train_tensor.view(-1)==0)[0]
        pos_perm = pos_idx[torch.randperm(len(pos_idx))]
        neg_perm = neg_idx[torch.randperm(len(neg_idx))]

        for start in range(0, N, batch_size):
            pos_batch_idx = pos_perm[start//2 : start//2 + batch_size//2]
            neg_batch_idx = neg_perm[start//2 : start//2 + batch_size//2]
```

```
            batch_idx = torch.cat([pos_batch_idx, neg_batch_idx])
            if len(batch_idx) == 0:
                continue
            Xb = X_train_tensor[batch_idx].to(DEVICE)
            yb = y_train_tensor[batch_idx].to(DEVICE)
            optimizer.zero_grad()
            logits = model(Xb)
            loss = criterion(logits, yb)
            loss.backward()
            optimizer.step()
            batch_losses.append(loss.item())

        epoch_loss = np.mean(batch_losses)

        # Validation
        model.eval()
        with torch.no_grad():
            logits_val = model(X_valid_device)
            probs_val = torch.sigmoid(logits_val).cpu().numpy().ravel()
            preds_val = (probs_val > 0.5).astype(int)
            val_f1 = f1_score(y_valid_tensor.cpu().numpy().ravel(), preds_val,
zero_division=0)

        scheduler.step(val_f1)

        if val_f1 > best_val_f1 + 1e-6:
            best_val_f1 = val_f1
            epochs_no_improve = 0
            best_epoch = epoch
            torch.save(model.state_dict(), out_path)
        else:
            epochs_no_improve += 1

        if epoch % 10 == 0 or epoch == 1 or epoch == epochs:
            lr_now = optimizer.param_groups[0]['lr']
            print(f"Epoch {epoch}/{epochs} | loss={epoch_loss:.4f} |
val_f1={val_f1:.4f} | best_val_f1={best_val_f1:.4f} | lr={lr_now:.2e}")

        if epochs_no_improve >= patience:
            print(f"Early stopping at epoch {epoch} (best_epoch={best_epoch},
best_val_f1={best_val_f1:.4f})")
            break

    # Test evaluation
    model.load_state_dict(torch.load(out_path, map_location=DEVICE))
    model.eval()
    with torch.no_grad():
        logits_test = model(X_test_device)
        probs_test = torch.sigmoid(logits_test).cpu().numpy().ravel()
        preds_test = (probs_test > 0.5).astype(int)
        y_test_np = y_test_tensor.cpu().numpy().ravel()
        test_f1 = f1_score(y_test_np, preds_test, zero_division=0)
        test_acc = accuracy_score(y_test_np, preds_test)
        report = classification_report(y_test_np, preds_test, digits=4,
zero_division=0)

    print("\nFinal test metrics:")
    print(f"Test accuracy: {test_acc:.4f}, Test F1: {test_f1:.4f}")
    print(report)

    return {'test_acc': test_acc, 'test_f1': test_f1, 'report': report,
'best_epoch': best_epoch}

X_tr_sub, X_val_sub, y_tr_sub, y_val_sub = train_test_split(
    X_train, y_train, test_size=0.10, random_state=42, stratify=y_train
```

```
)
X_tr_sub_t = torch.tensor(X_tr_sub, dtype=torch.float32)
X_val_sub_t = torch.tensor(X_val_sub, dtype=torch.float32)
y_tr_sub_t = torch.tensor(y_tr_sub, dtype=torch.float32).view(-1,1)
y_val_sub_t = torch.tensor(y_val_sub, dtype=torch.float32).view(-1,1)

print("\n=== Training model 1 (1 hidden, 32) ===")
model1 = MLP_1Hidden(input_size)
res1 = train_mlp(model1, X_tr_sub_t, y_tr_sub_t, X_val_sub_t, y_val_sub_t,
X_test_t, y_test_t,
                 epochs=500, lr=1e-3, batch_size=128, patience=40,
out_path=OUT_MODEL_1)

print("\n=== Training model 2 (2 hidden, 16+16) ===")
model2 = MLP_2Hidden(input_size)
res2 = train_mlp(model2, X_tr_sub_t, y_tr_sub_t, X_val_sub_t, y_val_sub_t,
X_test_t, y_test_t,
                 epochs=500, lr=1e-3, batch_size=128, patience=40,
out_path=OUT_MODEL_2)

print("\n=== Comparison ===")
print(f"Model1 (1 hidden): Test Acc={res1['test_acc']:.4f}, Test
F1={res1['test_f1']:.4f}, best_epoch={res1['best_epoch']}")
print(f"Model2 (2 hidden): Test Acc={res2['test_acc']:.4f}, Test
F1={res2['test_f1']:.4f}, best_epoch={res2['best_epoch']}")

summary = {
    'model1': {'test_acc': res1['test_acc'], 'test_f1': res1['test_f1'],
'best_epoch': res1['best_epoch']},
    'model2': {'test_acc': res2['test_acc'], 'test_f1': res2['test_f1'],
'best_epoch': res2['best_epoch']}
}
with open('training_summary.json', 'w', encoding='utf-8') as f:
    json.dump(summary, f, indent=2, ensure_ascii=False)
print('Saved training_summary.json')
```

```
Using device: cpu

=== Training model 1 (1 hidden, 32) ===
Epoch 1/500 | loss=0.6397 | val_f1=0.0437 | best_val_f1=0.0437 | lr=1.00e-03
Epoch 10/500 | loss=0.5735 | val_f1=0.6587 | best_val_f1=0.6587 | lr=1.00e-03
Epoch 20/500 | loss=0.5578 | val_f1=0.6884 | best_val_f1=0.6884 | lr=1.00e-03
Epoch 30/500 | loss=0.5485 | val_f1=0.6885 | best_val_f1=0.6951 | lr=1.00e-03
Epoch 40/500 | loss=0.5424 | val_f1=0.6900 | best_val_f1=0.6951 | lr=5.00e-04
Epoch 50/500 | loss=0.5312 | val_f1=0.6854 | best_val_f1=0.6951 | lr=2.50e-04
Epoch 60/500 | loss=0.5278 | val_f1=0.6812 | best_val_f1=0.6951 | lr=1.25e-04
Early stopping at epoch 67 (best_epoch=27, best_val_f1=0.6951)

Final test metrics:
Test accuracy: 0.8475, Test F1: 0.6908
              precision    recall  f1-score   support

         0.0     0.9057    0.8920    0.8988      4945
         1.0     0.6750    0.7073    0.6908      1568

    accuracy                         0.8475      6513
   macro avg     0.7904    0.7996    0.7948      6513
weighted avg     0.8502    0.8475    0.8487      6513


=== Training model 2 (2 hidden, 16+16) ===
Epoch 1/500 | loss=0.6638 | val_f1=0.0221 | best_val_f1=0.0221 | lr=1.00e-03
Epoch 10/500 | loss=0.5719 | val_f1=0.6645 | best_val_f1=0.6645 | lr=1.00e-03
Epoch 20/500 | loss=0.5661 | val_f1=0.6810 | best_val_f1=0.6880 | lr=1.00e-03
Epoch 30/500 | loss=0.5534 | val_f1=0.6909 | best_val_f1=0.6909 | lr=5.00e-04
Epoch 40/500 | loss=0.5494 | val_f1=0.6942 | best_val_f1=0.6951 | lr=5.00e-04
Epoch 50/500 | loss=0.5459 | val_f1=0.6866 | best_val_f1=0.6951 | lr=2.50e-04
Epoch 60/500 | loss=0.5335 | val_f1=0.6827 | best_val_f1=0.6951 | lr=1.25e-04
Epoch 70/500 | loss=0.5339 | val_f1=0.6816 | best_val_f1=0.6951 | lr=6.25e-05
Early stopping at epoch 76 (best_epoch=36, best_val_f1=0.6951)
```

```
Final test metrics:
Test accuracy: 0.8326, Test F1: 0.6896
              precision    recall  f1-score   support

         0.0     0.9219    0.8518    0.8854      4945
         1.0     0.6229    0.7723    0.6896      1568

    accuracy                         0.8326      6513
   macro avg     0.7724    0.8120    0.7875      6513
weighted avg     0.8499    0.8326    0.8383      6513


=== Comparison ===
Model1 (1 hidden): Test Acc=0.8475, Test F1=0.6908, best_epoch=27
Model2 (2 hidden): Test Acc=0.8326, Test F1=0.6896, best_epoch=36
Saved training_summary.json
PS D:\Uni>
```

Вывод: был построен, обучен и оценён многослойный перцептрон (MLP) для решения задачи классификации. Провели сравнение производительности с более глубокой моделью: два скрытых слоя по 16 нейронов.