Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра интеллектуально-информационных технологий

Лабораторная работа №3
По дисциплине «Интеллектуальный анализ данных»

Тема: «Предобучение нейронных сетей с использованием автоэнкодерного подхода»

Выполнила:
студентка 4 курса
группы ИИ-24
Коцуба Е.М.
Проверила:
Андренко К.В.

Брест 2025

Цель работы: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.

Вариант 5

№	Выборка	Тип задачи	Целевая переменная
5	https://archive.ics.uci.edu/dataset/193/cardiotocography	классификация	CLASS/NSP

Задание:

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).

2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.

3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.

4. Выполните пункты 1-3 для датасетов из ЛР 2.

5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from ucimlrepo import fetch_ucirepo


def prepare_data(X, y, test_size=0.2, random_state=42):
    mask = ~np.isnan(X).any(axis=1)
    X, y = X[mask], y[mask]

    X_tr, X_te, y_tr, y_te = train_test_split(
```

```
        X, y, test_size=test_size, random_state=random_state, stratify=y
    )
    scaler = StandardScaler()
    X_tr = scaler.fit_transform(X_tr)
    X_te = scaler.transform(X_te)

    X_tr_t = torch.FloatTensor(X_tr)
    X_te_t = torch.FloatTensor(X_te)
    y_tr_t = torch.LongTensor(y_tr)
    y_te_t = torch.LongTensor(y_te)

    loader = DataLoader(TensorDataset(X_tr_t, y_tr_t), batch_size=32,
shuffle=True)
    return X_tr_t, X_te_t, y_tr_t, y_te_t, loader


def create_model(in_dim, out_dim):
    class Net(nn.Module):
        def __init__(self):
            super().__init__()
            self.fc1 = nn.Linear(in_dim, 128)
            self.fc2 = nn.Linear(128, 64)
            self.fc3 = nn.Linear(64, 32)
            self.fc4 = nn.Linear(32, 16)
            self.fc5 = nn.Linear(16, out_dim)

        def forward(self, x):
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = F.relu(self.fc3(x))
            x = F.relu(self.fc4(x))
            return self.fc5(x)
    return Net()


def evaluate(model, X_te, y_te):
    model.eval()
    with torch.no_grad():
        pred = model(X_te).argmax(dim=1).cpu().numpy()
    return (
        accuracy_score(y_te, pred),
        f1_score(y_te, pred, average='weighted'),
        confusion_matrix(y_te, pred)
    )


def train_supervised(model, loader, epochs=50, lr=0.001, label=''):
    crit = nn.CrossEntropyLoss()
    opt = optim.Adam(model.parameters(), lr=lr)
    every = 20 if epochs >= 50 else 10
    for ep in range(epochs):
        model.train()
        for x, y in loader:
            opt.zero_grad()
            loss = crit(model(x), y)
            loss.backward()
            opt.step()
        if (ep + 1) % every == 0:
            print(f'  [{label}] Эпоха {ep+1}/{epochs}')


def train_autoencoder(in_sz, hid_sz, loader, epochs=50):
    class AE(nn.Module):
        def __init__(self):
```

```
            super().__init__()
            self.enc = nn.Linear(in_sz, hid_sz)
            self.dec = nn.Linear(hid_sz, in_sz)
        def forward(self, x):
            return self.dec(F.relu(self.enc(x)))

    ae = AE()
    crit = nn.MSELoss()
    opt = optim.Adam(ae.parameters(), lr=0.001)
    for _ in range(epochs):
        ae.train()
        for x, _ in loader:
            opt.zero_grad()
            loss = crit(ae(x), x)
            loss.backward()
            opt.step()
    return ae.enc


def pretrain_model(model, X_tr, layers):
    print('  Предобучение автоэнкодерами...')
    loader = DataLoader(TensorDataset(X_tr, torch.zeros(len(X_tr))),
batch_size=32, shuffle=True)
    enc = train_autoencoder(layers[0][0], layers[0][1], loader)
    model.fc1.weight.data = enc.weight.data
    model.fc1.bias.data = enc.bias.data

    hidden = X_tr
    for i in range(1, len(layers)):
        with torch.no_grad():
            if i == 1: hidden = F.relu(model.fc1(hidden))
            elif i == 2: hidden = F.relu(model.fc2(hidden))
            elif i == 3: hidden = F.relu(model.fc3(hidden))
        loader = DataLoader(TensorDataset(hidden, torch.zeros(len(hidden))),
batch_size=32, shuffle=True)
        enc = train_autoencoder(layers[i][0], layers[i][1], loader)
        fc = getattr(model, f'fc{i+1}')
        fc.weight.data = enc.weight.data
        fc.bias.data = enc.bias.data
    print('  Предобучение завершено.')

datasets = [
    {
        "name": "Cardiotocography (NSP)",
        "id": 193,
        "target": "NSP",
        "adjust_y": lambda y: (y - 1).values
    },
    {
        "name": "Wholesale Customers (Region)",
        "id": 292,
        "target": "Region",
        "adjust_y": lambda y: y.values - 1
    },
    {
        "name": "Optical Digits (tra)",
        "type": "csv",
        "train_url": "https://archive.ics.uci.edu/ml/machine-learning-
databases/optdigits/optdigits.tra",
        "test_url":  "https://archive.ics.uci.edu/ml/machine-learning-
databases/optdigits/optdigits.tes",
        "adjust_y": lambda y: np.array(y)
    }
]
```

```
results = []

for ds in datasets:
    print(f"Датасет: {ds['name']}")

    if ds.get("type") == "csv":
        df_train = pd.read_csv(ds["train_url"], header=None)
        df_test  = pd.read_csv(ds["test_url"],  header=None)

        X_train = df_train.iloc[:, :-1].values
        y_train = df_train.iloc[:, -1].values
        X_test  = df_test.iloc[:, :-1].values
        y_test  = df_test.iloc[:, -1].values

        X = np.concatenate([X_train, X_test], axis=0)
        y = np.concatenate([y_train, y_test], axis=0)
        y = ds["adjust_y"](y)

    else:
        data = fetch_ucirepo(id=ds["id"])
        X = data.data.features.values
        y = ds["adjust_y"](data.data.targets[ds["target"]])

    X_tr_t, X_te_t, y_tr, y_te, train_loader = prepare_data(X, y)
    in_dim = X.shape[1]
    n_cls  = len(np.unique(y))

    print(f"  Признаков: {in_dim}, Классов: {n_cls}, Объектов: {len(y)}")

    layers = [(in_dim, 128), (128, 64), (64, 32), (32, 16)]

    is_optical = "Optical" in ds["name"]
    epochs_no = 100 if is_optical else 50
    epochs_pre = 30 if is_optical else 20

    print("\n Обучение без предобучения")
    model_no = create_model(in_dim, n_cls)
    train_supervised(model_no, train_loader, epochs=epochs_no, lr=0.001,
label="Без")
    acc_no, f1_no, _ = evaluate(model_no, X_te_t, y_te)

    print("\n Обучение с предобучением")
    model_pre = create_model(in_dim, n_cls)
    pretrain_model(model_pre, X_tr_t, layers)
    train_supervised(model_pre, train_loader, epochs=epochs_pre, lr=0.0001,
label="С предоб.")
    acc_pre, f1_pre, _ = evaluate(model_pre, X_te_t, y_te)

    print(f"\n  Результаты:")
    print(f"    Без предобучения → Acc: {acc_no:.4f}, F1: {f1_no:.4f}")
    print(f"    С предобучением  → Acc: {acc_pre:.4f}, F1: {f1_pre:.4f}")
    print(f"    Улучшение: Acc {acc_pre-acc_no:+.4f}, F1 {f1_pre-
f1_no:+.4f}")

    results.append({
        "dataset": ds["name"],
        "acc_no": acc_no, "f1_no": f1_no,
        "acc_pre": acc_pre, "f1_pre": f1_pre
    })

print("\n" + "="*90)
print("Итоговая таблица")
print("="*90)
```

```
print(f"{'Датасет':<38} {'Без Acc':<8} {'С Acc':<8} {'ΔAcc':<7} {'Без F1':<8}
{'С F1':<8} {'ΔF1'}")
print("-"*90)
for r in results:
    name = r["dataset"].split(" (")[0]
    print(f"{name:<38} {r['acc_no']:.4f}   {r['acc_pre']:.4f}
{r['acc_pre']-r['acc_no']:+.4f}  "
          f"{r['f1_no']:.4f}   {r['f1_pre']:.4f}   {r['f1_pre']-
r['f1_no']:+.4f}")
```

Результат работы программы:

Датасет: Cardiotocography (NSP)
Признаков: 21, Классов: 3, Объектов: 2126

Обучение без предобучения
[Без] Эпоха 20/50
[Без] Эпоха 40/50

Обучение с предобучением
Предобучение автоэнкодерами...
Предобучение завершено.
[С предоб.] Эпоха 10/20
[С предоб.] Эпоха 20/20

Результаты:
Без предобучения → Acc: 0.9038, F1: 0.9035
С предобучением → Acc: 0.8873, F1: 0.8850
Улучшение: Acc -0.0164, F1 -0.0185

Датасет: Wholesale Customers (Region)
Признаков: 7, Классов: 3, Объектов: 440

Обучение без предобучения
[Без] Эпоха 20/50
[Без] Эпоха 40/50

Обучение с предобучением
Предобучение автоэнкодерами...
Предобучение завершено.
[С предоб.] Эпоха 10/20
[С предоб.] Эпоха 20/20

Результаты:
Без предобучения → Acc: 0.6932, F1: 0.5862
С предобучением → Acc: 0.7045, F1: 0.5918
Улучшение: Acc +0.0114, F1 +0.0056

Датасет: Optical Digits (tra)
Признаков: 64, Классов: 10, Объектов: 5620

Обучение без предобучения
[Без] Эпоха 20/100
[Без] Эпоха 40/100
[Без] Эпоха 60/100
[Без] Эпоха 80/100

[Без] Эпоха 100/100

Обучение с предобучением
Предобучение автоэнкодерами...
Предобучение завершено.
[С предоб.] Эпоха 10/30
[С предоб.] Эпоха 20/30
[С предоб.] Эпоха 30/30

Результаты:
Без предобучения → Acc: 0.9804, F1: 0.9805
С предобучением → Acc: 0.9698, F1: 0.9697
Улучшение: Acc -0.0107, F1 -0.0108

==========
Итоговая таблица
==========

Датасет	Без Acc	С Acc	ΔAcc	Без F1	С F1	ΔF1
Cardiotocography	0.9038	0.8873	-0.0164	0.9035	0.8850	-0.0185
Wholesale Customers	0.6932	0.7045	+0.0114	0.5862	0.5918	+0.0056
Optical Digits	0.9804	0.9698	-0.0107	0.9805	0.9697	-0.0108

Анализ результатов

1. Wholesale Customers (Region) — положительный эффект +1.14% по Accuracy, +0.56% по F1.

Датасет маленький (440 объектов) и несбалансированный. Предобучение помогло лучше инициализировать веса, особенно в глубоких слоях. Лучший случай для автоэнкодеров — малые данные + глубокая сеть.

2. Cardiotocography (NSP) — небольшое ухудшение −1.64% по Accuracy.

Датасет среднего размера, но сильно несбалансированный (78% — класс N). Предобучение переобучило скрытые представления под доминирующий класс. Fine-tuning не смог компенсировать смещение

Вывод: на несбалансированных данных нужна стратификация или взвешенная потеря

3. Optical Digits (tra) — ухудшение −1.07% по Accuracy.

Датасет большой (5620 объектов), хорошо разделимый. Базовая модель уже достигла 98% — близко к пределу. Предобучение внесло шум в инициализацию, не улучшив обобщение

Вывод: на больших и простых датасетах предобучение не нужно