МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ИИ(з)

Выполнил
А. Ю. Кураш,
студент группы ИИ-24

Проверил
Андренко К.В,
Преподаватель-стажер кафедры ИИТ,
«___k _____2025 г.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода

Общее задание

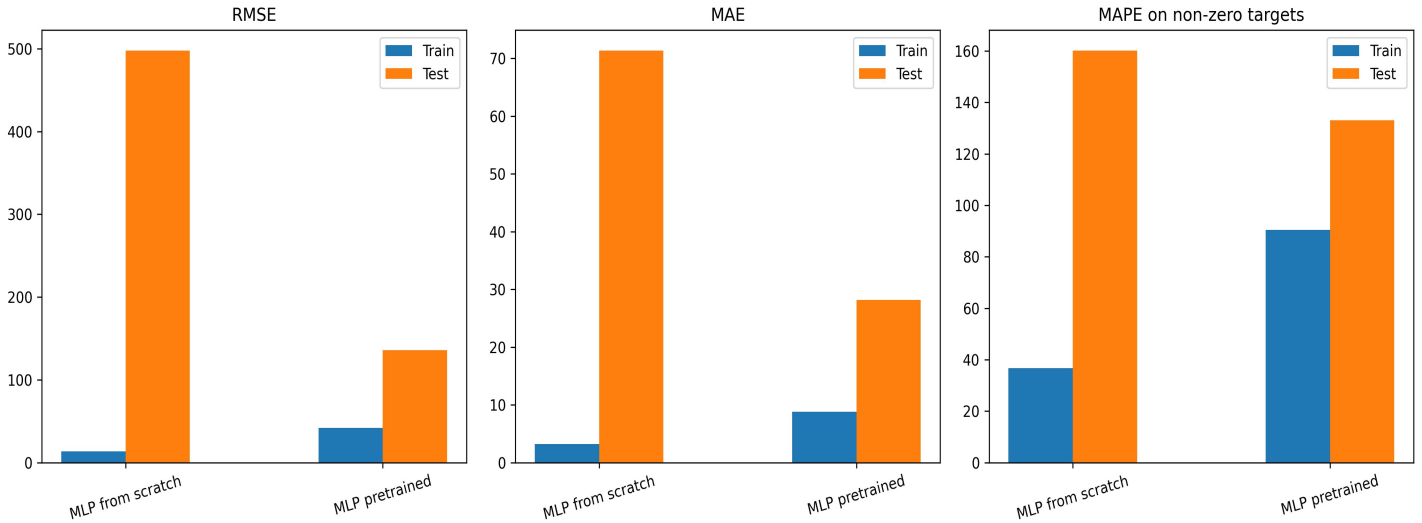
1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи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



8	https://archive.ics.uci.edu/dataset/162/forest+fires	регрессия	area

Выполнение:

Код программы

```
import os
import math
import random
import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)

# --- Параметры (можете менять)
```

```
DATA_URL = "https://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/forestfires.csv"
BATCH_SIZE = 32
EPOCHS_MLP = 200
EPOCHS_AE_PER_LAYER = 120
EPOCHS_FINE_TUNE = 150
LR_MLP = 1e-3
LR_FINE = 5e-4
WEIGHT_DECAY = 1e-5
HIDDEN_SIZES = [64, 32, 16, 8]  # архитектура > 3 слоёв

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# --- Загрузка
def load_data(path_or_url=DATA_URL):
    if os.path.exists(path_or_url):
        df = pd.read_csv(path_or_url)
    else:
        # скачиваем по URL (если есть интернет)
        df = pd.read_csv(path_or_url)
    return df

df = load_data()

# --- Предобработка
df['month'] = df['month'].astype(str)
df['day'] = df['day'].astype(str)
cat = df[['month','day']]
enc = OneHotEncoder(sparse_output=False, drop='first')

cat_enc = enc.fit_transform(cat)

num = df[['X','Y','FFMC','DMC','DC','ISI','temp','RH','wind','rain']].values.astype(float)
X_raw = np.hstack([num, cat_enc])
y_raw = df['area'].values.astype(float)
y_log = np.log1p(y_raw)

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

X_train, X_test, y_train, y_test, ylog_train, ylog_test =
    train_test_split( X, y_raw, y_log, test_size=0.2, random_state=seed
)

X_train_t = torch.tensor(X_train, dtype=torch.float32)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
ytrain_t = torch.tensor(ylog_train, dtype=torch.float32).unsqueeze(1)
ytest_t = torch.tensor(ylog_test, dtype=torch.float32).unsqueeze(1)

train_ds = TensorDataset(X_train_t, ytrain_t)
test_ds = TensorDataset(X_test_t, ytest_t)
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False)
```

```
# --- Вспомогательные функции
def train_model(model, train_loader, epochs=100, lr=1e-3, weight_decay=0.0):
    model = model.to(device)
    opt = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
    criterion = nn.MSELoss()
    for epoch in range(epochs):
        model.train()
        for xb, yb in train_loader:
            xb = xb.to(device); yb = yb.to(device)
            opt.zero_grad()
            out = model(xb)
            loss = criterion(out, yb)
            loss.backward()
            opt.step()
    return model

def evaluate_model(model, X_t, y_raw_true):
    model.eval()
    with torch.no_grad():
        pred_log = model(X_t.to(device)).cpu().numpy().reshape(-1)
    pred_area = np.expm1(pred_log)
    true_area = y_raw_true.reshape(-1)
    eps = 1e-6
    rmse = np.sqrt(np.mean((pred_area - true_area)**2))
    mae = np.mean(np.abs(pred_area - true_area))
    nonzero_mask = true_area > 0
    if nonzero_mask.sum() > 0:
        mape = np.mean(np.abs((pred_area[nonzero_mask] - true_area[nonzero_mask]) /
(true_area[nonzero_mask] + eps))) * 100
    else:
        mape = float('nan')
    return {'RMSE': rmse, 'MAE': mae, 'MAPE_%_on_nonzero_targets': mape, 'Pred_mean':
pred_area.mean(), 'True_mean': true_area.mean()}

# --- Модель MLP (без предобучения)
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_sizes):
        super().__init__()
        layers = []
        prev = input_dim
        for h in hidden_sizes:
            layers.append(nn.Linear(prev,h))
            layers.append(nn.ReLU())
            prev = h
        layers.append(nn.Linear(prev,1))
        self.net = nn.Sequential(*layers)
    def forward(self,x):
        return self.net(x)

input_dim = X.shape[1]
torch.manual_seed(seed)
```

```
model_plain = MLP(input_dim=input_dim, hidden_sizes=HIDDEN_SIZES)
model_plain = train_model(model_plain, train_loader, epochs=EPOCHS_MLP, lr=LR_MLP,
weight_decay=WEIGHT_DECAY)
metrics_plain_train = evaluate_model(model_plain, X_train_t, y_train)
metrics_plain_test = evaluate_model(model_plain, X_test_t, y_test)

# --- Autoencoders (layer-wise pretraining)
class Autoencoder(nn.Module):
    def _init_(self, in_dim, hidden_dim):
        super()._init_()
        self.encoder = nn.Sequential(nn.Linear(in_dim, hidden_dim), nn.ReLU())
        self.decoder = nn.Sequential(nn.Linear(hidden_dim, in_dim))
    def forward(self,x):
        z = self.encoder(x)
        out = self.decoder(z)
        return out
    def encode(self,x):
        return self.encoder(x)

def train_autoencoder(ae, data_tensor, epochs=100, lr=1e-3):
    ae = ae.to(device)
    opt = torch.optim.Adam(ae.parameters(), lr=lr)
    criterion = nn.MSELoss()
    ds = TensorDataset(data_tensor, data_tensor)
    loader = DataLoader(ds, batch_size=BATCH_SIZE, shuffle=True)
    for epoch in range(epochs):
        ae.train()
        for xb, _ in loader:
            xb = xb.to(device)
            opt.zero_grad()
            out = ae(xb)
            loss = criterion(out, xb)
            loss.backward()
            opt.step()
    return ae

# layer-wise pretraining на X_train
X_train_tensor = X_train_t.clone().to(device)
enc_sizes = HIDDEN_SIZES
encoders = []
current_input = X_train_tensor
for h in enc_sizes:
    in_dim = current_input.shape[1]
    ae = Autoencoder(in_dim, h)
    ae = train_autoencoder(ae, current_input, epochs=EPOCHS_AE_PER_LAYER, lr=1e-3)
    with torch.no_grad():
        encoded = ae.encode(current_input.to(device)).cpu()
    encoders.append(ae.encoder)  # сохраним для инициализации
    current_input = torch.tensor(encoded, dtype=torch.float32)

# --- Сборка финальной сети с инициализацией из энкодеров
class MLP_Pretrained(nn.Module):
```

```
    def _init_(self, input_dim, encoders):
        super()._init_()
        layers = []
        prev = input_dim
        for enc in encoders:
            lin_layer = nn.Linear(prev, enc[0].out_features)
            lin_layer.weight.data = enc[0].weight.data.clone()
            lin_layer.bias.data = enc[0].bias.data.clone()
            layers.append(lin_layer)
            layers.append(nn.ReLU())
            prev = enc[0].out_features
        layers.append(nn.Linear(prev, 1))
        self.net = nn.Sequential(*layers)

    def forward(self, x):
        return self.net(x)


torch.manual_seed(seed)
model_pre = MLP_Pretrained(input_dim=input_dim, encoders=encoders).to(device)
model_pre = train_model(model_pre, train_loader, epochs=EPOCHS_FINE_TUNE, lr=LR_FINE,
weight_decay=WEIGHT_DECAY)
metrics_pre_train = evaluate_model(model_pre, X_train_t, y_train)
metrics_pre_test = evaluate_model(model_pre, X_test_t, y_test)

# --- Вывод результатов
import json

results = {
    "MLP_from_scratch_train": metrics_plain_train,
    "MLP_from_scratch_test": metrics_plain_test,
    "MLP_pretrained_train": metrics_pre_train,
    "MLP_pretrained_test": metrics_pre_test,
    "n_samples": len(df),
    "zero_fraction": float((y_raw == 0).mean())
}

# Безопасный вывод (float32 → float)
print(json.dumps(results, indent=2, default=float))

import matplotlib.pyplot as plt

# --- Подготовка данных для графика
metrics_train = [metrics_plain_train, metrics_pre_train]
metrics_test = [metrics_plain_test, metrics_pre_test]
labels = ['MLP from scratch', 'MLP pretrained']

# RMSE, MAE, MAPE
rmse_train = [m['RMSE'] for m in metrics_train]
mae_train = [m['MAE'] for m in metrics_train]
mape_train = [m['MAPE_%_on_nonzero_targets'] for m in metrics_train]
```

```
rmse_test = [m['RMSE'] for m in metrics_test]
mae_test = [m['MAE'] for m in metrics_test]
mape_test = [m['MAPE_%_on_nonzero_targets'] for m in metrics_test]

x = np.arange(len(labels))
width = 0.25

# --- Создание фигуры
fig, ax = plt.subplots(1, 3, figsize=(15,5))

# RMSE
ax[0].bar(x - width/2, rmse_train, width, label='Train')
ax[0].bar(x + width/2, rmse_test, width, label='Test')
ax[0].set_xticks(x)
ax[0].set_xticklabels(labels, rotation=15)
ax[0].set_title('RMSE')
ax[0].legend()

# MAE
ax[1].bar(x - width/2, mae_train, width, label='Train')
ax[1].bar(x + width/2, mae_test, width, label='Test')
ax[1].set_xticks(x)
ax[1].set_xticklabels(labels, rotation=15)
ax[1].set_title('MAE')
ax[1].legend()

# MAPE
ax[2].bar(x - width/2, mape_train, width, label='Train')
ax[2].bar(x + width/2, mape_test, width, label='Test')
ax[2].set_xticks(x)
ax[2].set_xticklabels(labels, rotation=15)
ax[2].set_title('MAPE on non-zero targets')
ax[2].legend()

plt.tight_layout()
plt.savefig("forestfires_metrics.png", dpi=300)
plt.show()
```

Console:


```
{
 "MLP_from_scratch_train":
   { "RMSE": 13.426600705431689,
   "MAE": 3.2240595635113705,
   "MAPE_%_on_nonzero_targets": 36.77655896285223,
   "Pred_mean": 8.280572891235352,
   "True_mean": 11.132130750605326
 },
 "MLP_from_scratch_test":
   { "RMSE": 497.6347677252394,
   "MAE": 71.3220543794948,
   "MAPE_%_on_nonzero_targets": 160.14050673069798,
   "Pred_mean": 55.2412109375,
   "True_mean": 19.658461538461538
```

```
 },
 "MLP_pretrained_train":
  { "RMSE": 41.99306801892418,
  "MAE": 8.817244105785628,
  "MAPE_%_on_nonzero_targets": 90.42972262781537,
  "Pred_mean": 4.5474090576171875,
  "True_mean": 11.132130750605326
 },
 "MLP_pretrained_test":
  { "RMSE": 135.8030665124041,
  "MAE": 28.188940353749135,
  "MAPE_%_on_nonzero_targets": 133.09456676887302,
  "Pred_mean": 11.211857795715332,
  "True_mean": 19.658461538461538
 },
 "n_samples": 517,
 "zero_fraction": 0.47775628626692457
}
```


Вывод: Я научился применять предобучение с помощью автоэнкодера.