Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуально-информационных технологий

Интеллектуальный анализ данных
Лабораторная работа №3
Предобучение нейронных сетей с использованием автоэнкодерного подхода

Выполнила:
студент 4 курса
группы ИИ-24
Лозейко М. А.
Проверила:
Андренко К. В.

Брест-2025

Цель работы: 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 (Wisconsin Diagnostic Breast Cancer (WDBC), класс – 2 признак).
5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№	Выборка	Тип задачи	Целевая переменная
9	https://archive.ics.uci.edu/dataset/850/raisin	классификация	Class

Код программы(вариант 1):

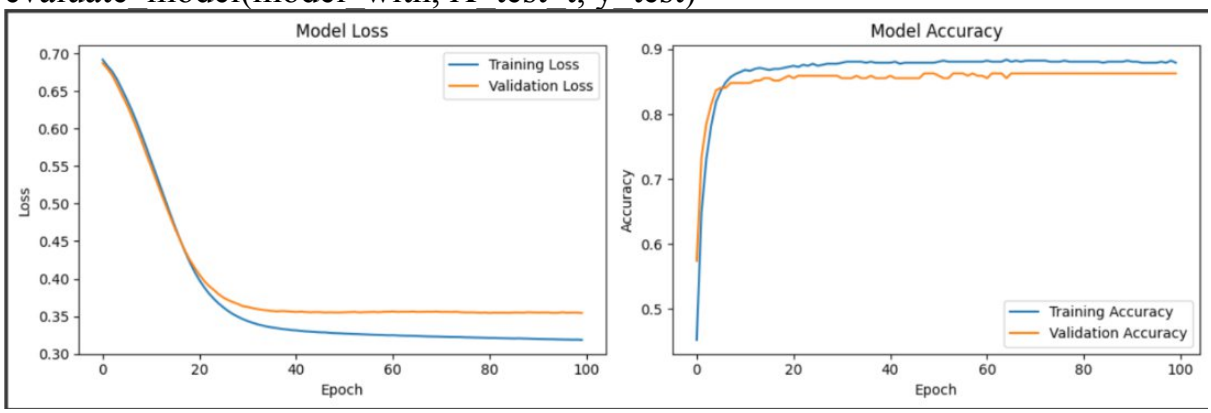
```
from ucimlrepo import fetch_ucirepo
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
```

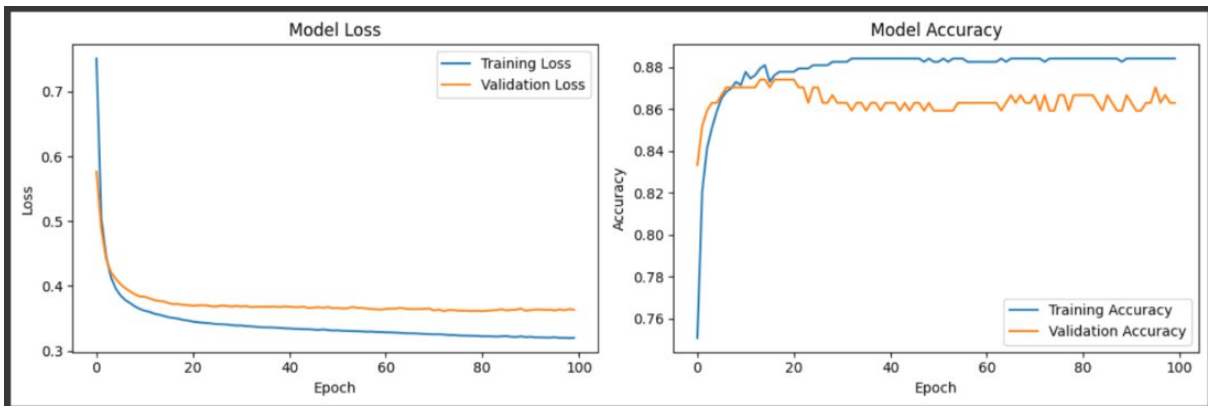
```
import math

# Определяем устройство (GPU, если доступно, иначе CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Загрузка данных
infrared_thermography_temperature = fetch_ucirepo(id=925)
X = infrared_thermography_temperature.data.features
y = infrared_thermography_temperature.data.targets['aveOralM']  # Регрессия по aveOralM

# Предобработка: обработка категориальных фич (one-hot), масштабирование
categorical_cols = ['Gender', 'Age', 'Ethnicity']  # Категориальные
X_encoded = pd.get_dummies(X, columns=categorical_cols, dtype=float)
X = X_encoded.fillna(X_encoded.median())  # Заполнение NaN

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y = y.values.reshape(-1, 1)

# Разделение на train/test (80/20)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Тензоры
X_train_t = torch.FloatTensor(X_train).to(device)
y_train_t = torch.FloatTensor(y_train).to(device)
X_test_t = torch.FloatTensor(X_test).to(device)
y_test_t = torch.FloatTensor(y_test).to(device)

# Модель MLP
class MLP(nn.Module):
    def __init__(self, input_size, layer_sizes=[128, 64, 32], output_size=1):
        super().__init__()
        layers = []
        in_size = input_size
        for out_size in layer_sizes:
            layers.append(nn.Linear(in_size, out_size))
            in_size = out_size
        self.layers = nn.ModuleList(layers)
        self.output_layer = nn.Linear(in_size, output_size)
        self.relu = nn.ReLU()
```

```
    def forward(self, x):
        for layer in self.layers:
            x = self.relu(layer(x))
        x = self.output_layer(x)
        return x

# Функция обучения
def train_model(model, X_train_t, y_train_t, epochs=200, lr=0.001,
batch_size=32):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    dataset = TensorDataset(X_train_t, y_train_t)
    loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    for epoch in range(epochs):
        model.train()
        epoch_loss = 0.0
        for batch_x, batch_y in loader:
            optimizer.zero_grad()
            out = model(batch_x)
            loss = criterion(out, batch_y)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        if (epoch + 1) % 50 == 0:
            print(f'Epoch {epoch+1}/{epochs}, Loss: {epoch_loss/len(loader):.4f}')

def evaluate_model(model, X_test_t, y_test):
    model.eval()
    with torch.no_grad():
        predictions = model(X_test_t).cpu().numpy()

    y_test_np = y_test.cpu().numpy() if isinstance(y_test, torch.Tensor) else
y_test
    mae = mean_absolute_error(y_test_np, predictions)
    rmse = math.sqrt(mean_squared_error(y_test_np, predictions))
    mape = mean_absolute_percentage_error(y_test_np, predictions) * 100

    print(f'MAE: {mae:.4f} °C')
    print(f'RMSE: {rmse:.4f} °C')
    print(f'MAPE: {mape:.4f}%')

# --- Модель без предобучения ---
```

```
print('Без предобучения:')
input_size = X.shape[1]
model_without = MLP(input_size).to(device)
train_model(model_without, X_train_t, y_train_t)
evaluate_model(model_without, X_test_t, y_test)

# Автоэнкодер
class AutoEncoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.encoder = nn.Linear(input_size, hidden_size)
        self.decoder = nn.Linear(hidden_size, input_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        encoded = self.relu(self.encoder(x))
        decoded = self.decoder(encoded)
        return decoded

def train_autoencoder(ae, X_t, epochs=50, lr=0.001, batch_size=32):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(ae.parameters(), lr=lr)
    dataset = TensorDataset(X_t, X_t)
    loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    for epoch in range(epochs):
        ae.train()
        epoch_loss = 0.0
        for batch_x, _ in loader:
            optimizer.zero_grad()
            out = ae(batch_x)
            loss = criterion(out, batch_x)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        if (epoch + 1) % 10 == 0:
            print(f'AE Layer {len(encoders)+1} - Epoch {epoch+1}/{epochs}, Loss:
{epoch_loss/len(loader):.4f}')

# --- Модель с предобучением ---
print('\nПредобучение автоэнкодерами:')
layer_sizes = [128, 64, 32]
current_input = X_train_t
encoders = []
```

```
for hidden_size in layer_sizes:
    ae = AutoEncoder(current_input.shape[1], hidden_size).to(device)
    train_autoencoder(ae, current_input)
    encoders.append(ae.encoder)

    # Получаем представления для следующего слоя
    with torch.no_grad():
        current_input = ae.relu(ae.encoder(current_input))

# Создаем предобученную модель
model_with = MLP(input_size, layer_sizes).to(device)

# Копируем веса из обученных энкодеров
for i, encoder_layer in enumerate(encoders):
    model_with.layers[i].load_state_dict(encoder_layer.state_dict())

print('\nДообучение модели с предобученными весами:')
train_model(model_with, X_train_t, y_train_t)

print('\nОценка модели с предобучением:')
evaluate_model(model_with, X_test_t, y_test)
```





```
Предобучение автоэнкодерами:
AE Layer 1 - Epoch 10/50, Loss: 0.0625
AE Layer 1 - Epoch 20/50, Loss: 0.0221
AE Layer 1 - Epoch 30/50, Loss: 0.0132
AE Layer 1 - Epoch 40/50, Loss: 0.0092
AE Layer 1 - Epoch 50/50, Loss: 0.0070
AE Layer 2 - Epoch 10/50, Loss: 0.0601
AE Layer 2 - Epoch 20/50, Loss: 0.0292
AE Layer 2 - Epoch 30/50, Loss: 0.0211
AE Layer 2 - Epoch 40/50, Loss: 0.0166
AE Layer 2 - Epoch 50/50, Loss: 0.0138
AE Layer 3 - Epoch 10/50, Loss: 0.2807
AE Layer 3 - Epoch 20/50, Loss: 0.2022
AE Layer 3 - Epoch 30/50, Loss: 0.1472
AE Layer 3 - Epoch 40/50, Loss: 0.1133
AE Layer 3 - Epoch 50/50, Loss: 0.0945
```

```
Дообучение модели с предобученными весами:
Epoch 50/200, Loss: 0.2576
Epoch 100/200, Loss: 0.0892
Epoch 150/200, Loss: 0.0554
Epoch 200/200, Loss: 0.0688

Оценка модели с предобучением:
MAE: 0.4532 °C
RMSE: 0.7654 °C
MAPE: 1.2257%
PS C:\Users\uzuma>
```

```
Без предобучения:
Epoch 50/200, Loss: 0.2829
Epoch 100/200, Loss: 0.0861
Epoch 150/200, Loss: 0.1387
Epoch 200/200, Loss: 0.1734
MAE: 0.5893 °C
RMSE: 1.2211 °C
MAPE: 1.5929%
```

Вывод: научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.