МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙГОСУДАРСТВЕННЫЙТЕХНИЧЕСКИЙУНИВЕРСИТЕТ» ФАКУЛЬТЕТ

ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ИИ-22

Выполнила
Леваневская Н.И.
студентка группы ИИ-22

Проверил
А.А. Крощенко,
ст. преп.
кафедры ИИТ,
«—» ———— 2024 г.

Брест 2024

Вариант 11

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

Задания:

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.

Выборка	Тип задачи	Целевая переменная
https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction	регрессия	Appliances

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
import numpy as np
from ucimlrepo import fetch_ucirepo
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')

appliances_energy_prediction = fetch_ucirepo(id=374)

X = appliances_energy_prediction.data.features
y = appliances_energy_prediction.data.targets['Appliances']

X = X.select_dtypes(include=['float64', 'int64'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
```

```
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).ravel()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).ravel()

X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_scaled, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test_scaled, dtype=torch.float32).view(-1, 1)

class Autoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Linear(input_dim, hidden_dim)
        self.decoder = nn.Linear(hidden_dim, input_dim)

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

class RegressionModel(nn.Module):
    def __init__(self, input_dim, hidden1_dim, hidden2_dim, output_dim):
        super(RegressionModel, self).__init__()
        self.f1 = nn.Linear(input_dim, hidden1_dim)
        self.f2 = nn.Linear(hidden1_dim, hidden2_dim)
        self.out = nn.Linear(hidden2_dim, output_dim)

    def forward(self, x):
        x = torch.relu(self.f1(x))
        x = torch.relu(self.f2(x))
        x = self.out(x)
        return x

def train_autoencoder(autoencoder, data, epochs=50, lr=0.001):
    optimizer = optim.Adam(autoencoder.parameters(), lr=lr)
    criterion = nn.MSELoss()
    for epoch in range(epochs):
        autoencoder.train()
        encoded, decoded = autoencoder(data)
        loss = criterion(decoded, data)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    return autoencoder

autoencoder_64 = Autoencoder(X_train.shape[1], 64)
autoencoder_64 = train_autoencoder(autoencoder_64, X_train_tensor, epochs=50)

with torch.no_grad():
    encoded_64, _ = autoencoder_64(X_train_tensor)

autoencoder_32 = Autoencoder(64, 32)
autoencoder_32 = train_autoencoder(autoencoder_32, encoded_64, epochs=50)

model_pretrained = RegressionModel(X_train.shape[1], 64, 32, 1)
model_pretrained.f1.weight.data = autoencoder_64.encoder.weight.data.clone()
```

```
model_pretrained.f1.bias.data = autoencoder_64.encoder.bias.data.clone()
model_pretrained.f2.weight.data = autoencoder_32.encoder.weight.data.clone()
model_pretrained.f2.bias.data = autoencoder_32.encoder.bias.data.clone()

def train_model_with_metrics(model, X_train, y_train, X_test, y_test, epochs=50, lr=0.001,
name="Model"):
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.MSELoss()
    losses, mapes = [], []
    print(f"Начало обучения {name}")
    for epoch in range(epochs):
        model.train()
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        model.eval()
        with torch.no_grad():
            test_pred = model(X_test)
            test_mape = mean_absolute_percentage_error(y_test.numpy(), test_pred.numpy())
            losses.append(loss.item())
            mapes.append(test_mape)

        if (epoch + 1) % 10 == 0:
            print(f"[{name}] Epoch {epoch + 1}/{epochs}, Loss: {loss.item():.4f}, Test MAPE:
{test_mape:.4f}")
    return model, losses, mapes

model_no_pretraining, losses_no_pretraining, mape_no_pretraining = train_model_with_metrics(
    RegressionModel(X_train.shape[1], 64, 32, 1),
    X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor,
    name="Без предобучения"
)

model_pretrained, losses_pretraining, mape_pretraining = train_model_with_metrics(
    model_pretrained,
    X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor,
    name="С предобучением"
)

final_mape_no_pretraining = mape_no_pretraining[-1]
final_mape_pretraining = mape_pretraining[-1]
print(f"MAPE без предобучения: {final_mape_no_pretraining:.4f}")
print(f"MAPE с предобучением: {final_mape_pretraining:.4f}")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, len(mape_no_pretraining) + 1), mape_no_pretraining, label="Без
предобучения", marker='o', color='#540EAD')
plt.plot(range(1, len(mape_pretraining) + 1), mape_pretraining, label="С предобучением",
marker='o', color='#E60042')
plt.title("MAPE на тесте")
```

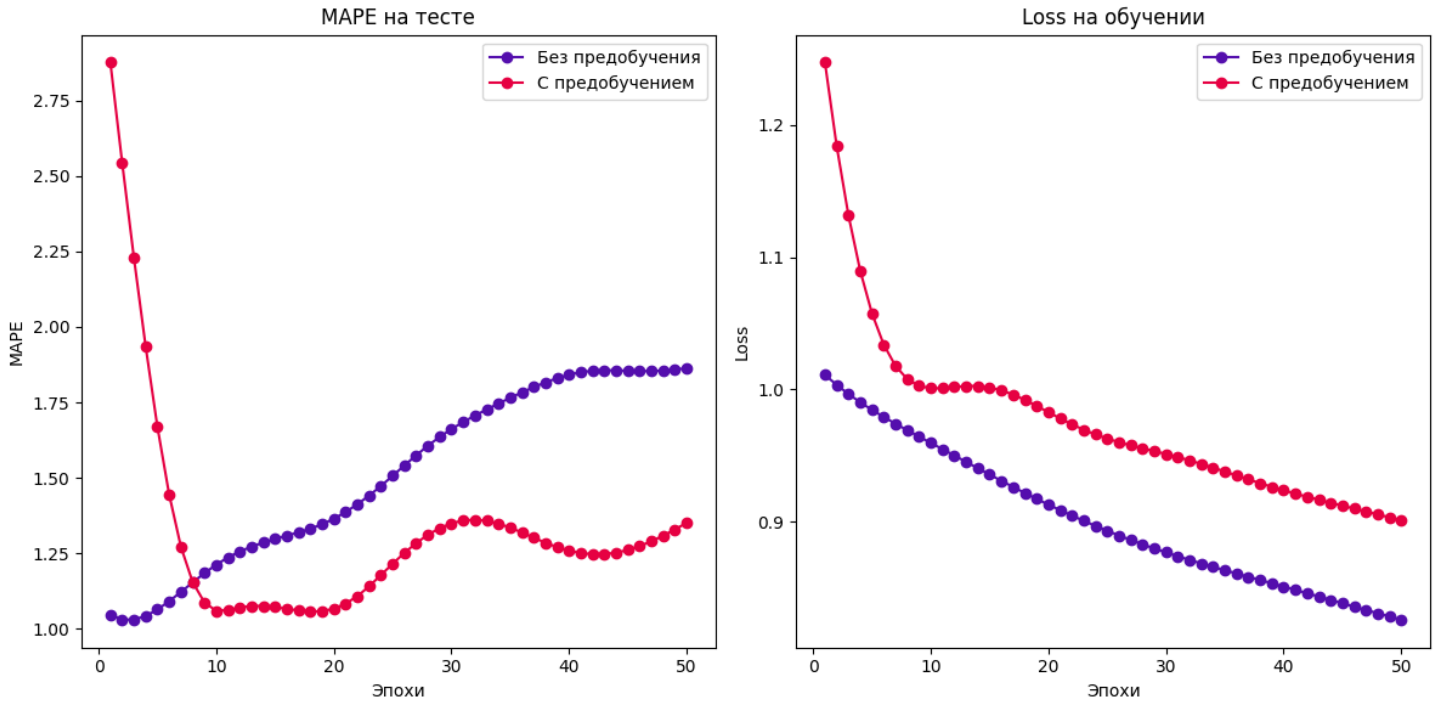
```
plt.xlabel("Эпохи")
plt.ylabel("MAPE")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, len(losses_no_pretraining) + 1), losses_no_pretraining, label="Без
предобучения", marker='o', color='#540EAD')
plt.plot(range(1, len(losses_pretraining) + 1), losses_pretraining, label="С предобучением",
marker='o', color='#E60042')
plt.title("Loss на обучении")
plt.xlabel("Эпохи")
plt.ylabel("Loss")
plt.legend()

plt.tight_layout()
plt.show()
```

Результат работы:

```
Начало обучения Без предобучения
[Без предобучения] Epoch 10/50, Loss: 0.9596, Test MAPE: 1.2130
[Без предобучения] Epoch 20/50, Loss: 0.9129, Test MAPE: 1.3651
[Без предобучения] Epoch 30/50, Loss: 0.8768, Test MAPE: 1.6619
[Без предобучения] Epoch 40/50, Loss: 0.8510, Test MAPE: 1.8420
[Без предобучения] Epoch 50/50, Loss: 0.8259, Test MAPE: 1.8633
Начало обучения С предобучением
[С предобучением] Epoch 10/50, Loss: 1.0009, Test MAPE: 1.0602
[С предобучением] Epoch 20/50, Loss: 0.9828, Test MAPE: 1.0669
[С предобучением] Epoch 30/50, Loss: 0.9512, Test MAPE: 1.3499
[С предобучением] Epoch 40/50, Loss: 0.9237, Test MAPE: 1.2592
[С предобучением] Epoch 50/50, Loss: 0.9011, Test MAPE: 1.3517
MAPE без предобучения: 1.8633
MAPE с предобучением: 1.3517
```



Вывод: на практике научилась осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.