Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине: «Интеллектуальный анализ данных»
Тема: “Предобучение нейронных сетей с использованием RBM”

Выполнил:
Студент 4 курса
Группы ИИ-24
Мшар В.В.
Проверила:
Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью RBM.

Общее задание

1. Взять за основу нейронную сеть из лабораторной работы №3. Выполнить обучение с предобучением, используя стек ограниченных машин Больцмана (RBM – Restricted Boltzmann Machine), алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев как RBM выбрать самостоятельно.

2. Сравнить результаты, полученные при
- обучении без предобучения (ЛР 3);
- обучении с предобучением, используя автоэнкодерный подход (ЛР3);
- обучении с предобучением, используя RBM.

3. Обучить модели на данных из ЛР 2, сравнить результаты по схеме из пункта 2;

4. Сделать выводы, оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Тип задачи	Целевая переменная
12	https://archive.ics.uci.edu/dataset/189/parkinsons+telemonitoring	регрессия	motor_UPDRS

Ход работы:

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score, f1_score
from ucimlrepo import fetch_ucirepo
import warnings
import numpy as np

# Подавляем лишние предупреждения для чистоты вывода
warnings.filterwarnings('ignore')
```

```
#
================================================================================
# 1. ОБЩИЕ КОМПОНЕНТЫ: RBM и MLP
#
================================================================================

class RBM(nn.Module):
    def __init__(self, n_visible, n_hidden):
        super(RBM, self).__init__()
        self.W = nn.Parameter(torch.randn(n_hidden, n_visible) * 0.1)
        self.v_bias = nn.Parameter(torch.zeros(n_visible))
        self.h_bias = nn.Parameter(torch.zeros(n_hidden))
        self.k = 1

    def v_to_h(self, v):
        p_h = torch.sigmoid(torch.matmul(v, self.W.t()) + self.h_bias)
        sample_h = torch.bernoulli(p_h)
        return p_h, sample_h

    def h_to_v(self, h):
        p_v = torch.sigmoid(torch.matmul(h, self.W) + self.v_bias)
        sample_v = torch.bernoulli(p_v)
        return p_v, sample_v

    def update_weights(self, v0, vk, h0_prob, hk_prob, lr):
        self.W.data += lr * (torch.matmul(h0_prob.t(), v0) - torch.matmul(hk_prob.t(), vk))
        self.v_bias.data += lr * torch.mean(v0 - vk, dim=0)
        self.h_bias.data += lr * torch.mean(h0_prob - hk_prob, dim=0)

def train_rbm(rbm, data_loader, epochs=25, lr=0.01):
    rbm.train()
    print(f"  Обучение RBM ({rbm.W.shape[1]} -> {rbm.W.shape[0]})")
    for epoch in range(epochs):
        for data, _ in data_loader:
            v0 = data
            h0_prob, _ = rbm.v_to_h(v0)

            vk = v0
            for _ in range(rbm.k):
                _, hk = rbm.v_to_h(vk)
                _, vk = rbm.h_to_v(hk)
```

```
            hk_prob, _ = rbm.v_to_h(vk)
            rbm.update_weights(v0, vk, h0_prob, hk_prob, lr)
    return rbm

# Модель для регрессии
class RegressionMLP(nn.Module):
    def __init__(self, n_in, n_h1, n_h2, n_h3, n_out):
        super(RegressionMLP, self).__init__()
        self.layer1, self.layer2, self.layer3 = nn.Linear(n_in, n_h1), nn.Linear(n_h1, n_h2),
nn.Linear(n_h2, n_h3)
        self.output_layer = nn.Linear(n_h3, n_out)
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.relu(self.layer1(x)); x = self.relu(self.layer2(x)); x = self.relu(self.layer3(x))
        return self.output_layer(x)

# Модель для классификации
class ClassificationMLP(nn.Module):
    def __init__(self, n_in, n_h1, n_h2, n_h3, n_out):
        super(ClassificationMLP, self).__init__()
        self.layer1, self.layer2, self.layer3 = nn.Linear(n_in, n_h1), nn.Linear(n_h1, n_h2),
nn.Linear(n_h2, n_h3)
        self.output_layer = nn.Linear(n_h3, n_out)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = self.relu(self.layer1(x)); x = self.relu(self.layer2(x)); x = self.relu(self.layer3(x))
        return self.sigmoid(self.output_layer(x))

#
==========================================================================
====================
# 2. АНАЛИЗ ДАТАСЕТА PARKISONS (РЕГРЕССИЯ)
#
==========================================================================
====================

def run_parkinsons_analysis():
    print("\n" + "="*80)
    print("Часть 1: Датасет Parkinsons Telemonitoring (Регрессия)")
    print("="*80)

    # 1. Загрузка и подготовка
    parkinsons = fetch_ucirepo(id=189)
    X = parkinsons.data.features
```

```
    y = parkinsons.data.targets['motor_UPDRS']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train); X_test_scaled =
scaler.transform(X_test)
    X_train_tensor = torch.FloatTensor(X_train_scaled); y_train_tensor =
torch.FloatTensor(y_train.values).view(-1, 1)
    X_test_tensor = torch.FloatTensor(X_test_scaled); y_test_tensor =
torch.FloatTensor(y_test.values).view(-1, 1)
    train_loader = DataLoader(TensorDataset(X_train_tensor, y_train_tensor),
batch_size=64, shuffle=True)
    print("✓ Данные Parkinsons подготовлены.")

    # 2. Параметры и предобучение RBM
    n_features, n_h1, n_h2, n_h3, n_out = X.shape[1], 128, 64, 32, 1
    print("\n[Предобучение RBM...]")
    rbm1 = train_rbm(RBM(n_features, n_h1), train_loader)
    with torch.no_grad(): hidden1, _ = rbm1.v_to_h(X_train_tensor)
    rbm2 = train_rbm(RBM(n_h1, n_h2), DataLoader(TensorDataset(hidden1,
y_train_tensor), batch_size=64))
    with torch.no_grad(): hidden2, _ = rbm2.v_to_h(hidden1)
    rbm3 = train_rbm(RBM(n_h2, n_h3), DataLoader(TensorDataset(hidden2,
y_train_tensor), batch_size=64))

    # 3. Дообучение (Fine-Tuning)
    model = RegressionMLP(n_features, n_h1, n_h2, n_h3, n_out)
    model.layer1.weight.data, model.layer1.bias.data = rbm1.W.data, rbm1.h_bias.data
    model.layer2.weight.data, model.layer2.bias.data = rbm2.W.data, rbm2.h_bias.data
    model.layer3.weight.data, model.layer3.bias.data = rbm3.W.data, rbm3.h_bias.data
    optimizer, criterion = optim.Adam(model.parameters(), lr=0.001), nn.MSELoss()
    print("\n[Дообучение (Fine-Tuning)...]")
    for epoch in range(100):
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad(); outputs = model(batch_X); loss = criterion(outputs,
batch_y)
            loss.backward(); optimizer.step()

    # 4. Оценка
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test_tensor)
        mse_rbm = criterion(y_pred, y_test_tensor).item()
        ss_res = torch.sum((y_test_tensor - y_pred)**2); ss_tot = torch.sum((y_test_tensor -
torch.mean(y_test_tensor))**2)
```

```
        r2_rbm = (1 - ss_res / ss_tot).item()

    print("\n--- Итоговое сравнение (Parkinsons) ---")
    print(f"| {'Метод':<35} | {'MSE (↓)':<10} | {'R² (↑)':<10} |")
    print(f"|{'-'*37}|{'-'*12}|{'-'*12}|")
    print(f"| {'Без предобучения (из ЛР №3)':<35} | {9.7132:<10.4f} | {0.8876:<10.4f}
|")
    print(f"| {'С предобучением Автоэнкодером (ЛР №3)':<35} | {8.8149:<10.4f} |
{0.8981:<10.4f} |")
    print(f"| {'С предобучением RBM (ЛР №4)':<35} | {mse_rbm:<10.4f} |
{r2_rbm:<10.4f} |")

#
==========================================================================
====================
# 3. АНАЛИЗ ДАТАСЕТА RICE (КЛАССИФИКАЦИЯ)
#
==========================================================================
====================

def run_rice_analysis():
    print("\n" + "="*80)
    print("Часть 2: Датасет Rice (Cammeo and Osmancik) (Классификация)")
    print("="*80)

    # 1. Загрузка и подготовка
    rice = fetch_ucirepo(id=545)
    X = rice.data.features
    y = rice.data.targets['Class']
    le = LabelEncoder(); y_encoded = le.fit_transform(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,
random_state=42, stratify=y_encoded)
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train); X_test_scaled =
scaler.transform(X_test)
    X_train_tensor = torch.FloatTensor(X_train_scaled); y_train_tensor =
torch.FloatTensor(y_train).view(-1, 1)
    X_test_tensor = torch.FloatTensor(X_test_scaled); y_test_tensor =
torch.FloatTensor(y_test).view(-1, 1)
    train_loader = DataLoader(TensorDataset(X_train_tensor, y_train_tensor),
batch_size=32, shuffle=True)
    print("✓ Данные Rice подготовлены.")

    # 2. Параметры и предобучение RBM
    n_features, n_h1, n_h2, n_h3, n_out = X.shape[1], 128, 64, 32, 1
```

```
    print("\n[Предобучение RBM...]")
    rbm1 = train_rbm(RBM(n_features, n_h1), train_loader, epochs=50)
    with torch.no_grad(): hidden1, _ = rbm1.v_to_h(X_train_tensor)
    rbm2 = train_rbm(RBM(n_h1, n_h2), DataLoader(TensorDataset(hidden1,
y_train_tensor), batch_size=32), epochs=50)
    with torch.no_grad(): hidden2, _ = rbm2.v_to_h(hidden1)
    rbm3 = train_rbm(RBM(n_h2, n_h3), DataLoader(TensorDataset(hidden2,
y_train_tensor), batch_size=32), epochs=50)

    # 3. Дообучение (Fine-Tuning)
    model = ClassificationMLP(n_features, n_h1, n_h2, n_h3, n_out)
    model.layer1.weight.data, model.layer1.bias.data = rbm1.W.data, rbm1.h_bias.data
    model.layer2.weight.data, model.layer2.bias.data = rbm2.W.data, rbm2.h_bias.data
    model.layer3.weight.data, model.layer3.bias.data = rbm3.W.data, rbm3.h_bias.data
    optimizer, criterion = optim.Adam(model.parameters(), lr=0.001), nn.BCELoss()
    print("\n[Дообучение (Fine-Tuning)...]")
    for epoch in range(50):
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad(); outputs = model(batch_X); loss = criterion(outputs,
batch_y)
            loss.backward(); optimizer.step()

    # 4. Оценка
    model.eval()
    with torch.no_grad():
        y_pred_prob = model(X_test_tensor)
        y_pred = (y_pred_prob > 0.5).int().numpy().flatten()
        y_true = y_test_tensor.int().numpy().flatten()
        acc_rbm = accuracy_score(y_true, y_pred)
        f1_rbm = f1_score(y_true, y_pred, average='weighted')

    acc_no_pre, f1_no_pre = 0.9883, 0.9883
    acc_ae, f1_ae = 0.9921, 0.9921

    print("\n--- Итоговое сравнение (Rice) ---")
    print(f"| {'Метод':<35} | {'Accuracy (↑)':<15} | {'F1-score (↑)':<15} |")
    print(f"|{'-'*37}|{'-'*17}|{'-'*17}|")
    print(f"| {'Без предобучения':<35} | {acc_no_pre:<15.4f} | {f1_no_pre:<15.4f} |")
    print(f"| {'С предобучением Автоэнкодером':<35} | {acc_ae:<15.4f} |
{f1_ae:<15.4f} |")
    print(f"| {'С предобучением RBM (ЛР №4)':<35} | {acc_rbm:<15.4f} |
{f1_rbm:<15.4f} |")
```

```
#
================================================================================
# 4. ЗАПУСК
#
================================================================================
if __name__ == '__main__':
    run_parkinsons_analysis()
    run_rice_analysis()
    print("\n" + "="*80)
    print("Все анализы завершены.")
    print("="*80)
```

Вывод: научился осуществлять предобучение нейронных сетей с помощью RBM.