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

Интеллектуальный анализ данных
Лабораторная работа №4
Предобучение нейронных сетей с использованием RBM

Выполнила:
студентка 4 курса
группы ИИ-24
Алешко А. В.
Проверила:
Андренко К. В.

Брест-2025

Цель работы: 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.

№	Выборка	Тип задачи	Целевая переменная
1	https://archive.ics.uci.edu/dataset/27/credit+approval	классификация	+/-

Код программы(вариант 1):

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Credit Approval ds
def load_credit_data():
    url =
'https://archive.ics.uci.edu/static/public/27/data.csv'
```

```
    data = pd.read_csv(url)
    data = data.replace('?', np.nan)
    X = data.drop('A16', axis=1)
    y = data['A16'].map({'+': 1, '-': 0})
    categorical_cols = ['A1', 'A4', 'A5', 'A6', 'A7', 'A9',
'A10', 'A12', 'A13']
    continuous_cols = ['A2', 'A3', 'A8', 'A11', 'A14', 'A15']
    cat_imputer = SimpleImputer(strategy='most_frequent')
    cont_imputer = SimpleImputer(strategy='median')
    X[categorical_cols] =
cat_imputer.fit_transform(X[categorical_cols])
    X[continuous_cols] =
cont_imputer.fit_transform(X[continuous_cols])
    encoder = OneHotEncoder(sparse_output=False,
handle_unknown='ignore')
    X_cat_encoded = encoder.fit_transform(X[categorical_cols])
    X_cat_encoded = pd.DataFrame(X_cat_encoded,
columns=encoder.get_feature_names_out(categorical_cols))
    X = pd.concat([X[continuous_cols], X_cat_encoded], axis=1)
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    return X, y.values

# Breast Cancer ds
def load_breast_cancer_data():
    url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/wdbc.data'
    column_names = ['ID', 'Diagnosis'] + [f'feature_{i}' for i
in range(1, 31)]
    data = pd.read_csv(url, header=None, names=column_names)
    data = data.drop('ID', axis=1)
    y = data['Diagnosis'].map({'M': 1, 'B': 0}).values
    X = data.drop('Diagnosis', axis=1)
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    return X, y

# NNA(4 уровня)
class ClassificationNet(nn.Module):
    def __init__(self, input_size, hidden_sizes,
output_size=1):
        super(ClassificationNet, self).__init__()
        self.layers = nn.ModuleList()
        prev_size = input_size
        for h_size in hidden_sizes:
            self.layers.append(nn.Linear(prev_size, h_size))
            prev_size = h_size
        self.output = nn.Linear(prev_size, output_size)

    def forward(self, x):
        for layer in self.layers:
```

```
            x = torch.relu(layer(x))
        x = torch.sigmoid(self.output(x))
        return x

# Autoencoder
class Autoencoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Linear(input_size, hidden_size)
        self.decoder = nn.Linear(hidden_size, input_size)

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

# RBM
class RBM(nn.Module):
    def __init__(self, visible_size, hidden_size):
        super(RBM, self).__init__()
        self.W = nn.Parameter(torch.randn(visible_size,
hidden_size) * 0.01)
        self.b_v = nn.Parameter(torch.zeros(visible_size))
        self.b_h = nn.Parameter(torch.zeros(hidden_size))

    def sample_h_given_v(self, v):
        p_h = torch.sigmoid(torch.matmul(v, self.W) +
self.b_h)
        h = torch.bernoulli(p_h)
        return p_h, h

    def sample_v_given_h(self, h):
        mean_v = torch.matmul(h, self.W.t()) + self.b_v
        v = mean_v
        return mean_v, v

# Autoencoder предобучение
def pretrain_ae_layers(input_data, hidden_sizes, epochs=50,
lr=0.01):
    pretrained_weights = []
    current_input = input_data
    for h_size in hidden_sizes:
        ae = Autoencoder(current_input.shape[1], h_size)
        optimizer = optim.Adam(ae.parameters(), lr=lr)
        criterion = nn.MSELoss()
        dataset = TensorDataset(current_input, current_input)
        loader = DataLoader(dataset, batch_size=32,
shuffle=True)
        for epoch in range(epochs):
            for data, target in loader:
                optimizer.zero_grad()
```

```
                output = ae(data)
                loss = criterion(output, target)
                loss.backward()
                optimizer.step()
        with torch.no_grad():
            current_input =
torch.relu(ae.encoder(current_input))

pretrained_weights.append((ae.encoder.weight.data.clone(),
ae.encoder.bias.data.clone()))
    return pretrained_weights

# RBM предобучение
def pretrain_rbm_layers(input_data, hidden_sizes, epochs=50,
lr=0.01, k=1):
    pretrained_weights = []
    current_input = input_data.clone()
    for h_size in hidden_sizes:
        rbm = RBM(current_input.shape[1], h_size)
        optimizer = optim.Adam(rbm.parameters(), lr=lr)
        dataset = TensorDataset(current_input)
        loader = DataLoader(dataset, batch_size=32,
shuffle=True)
        for epoch in range(epochs):
            for batch in loader:
                v0 = batch[0]
                ph0, h0 = rbm.sample_h_given_v(v0)
                mean_vk, vk = rbm.sample_v_given_h(h0)
                phk, _ = rbm.sample_h_given_v(mean_vk)
                rbm.W.grad = (torch.matmul(v0.t(), ph0) -
torch.matmul(mean_vk.t(), phk)) / v0.size(0)
                rbm.b_v.grad = torch.mean(v0 - mean_vk, dim=0)
                rbm.b_h.grad = torch.mean(ph0 - phk, dim=0)
                optimizer.step()
        with torch.no_grad():
            p_h = torch.sigmoid(torch.matmul(current_input,
rbm.W) + rbm.b_h)
            current_input = p_h
        pretrained_weights.append((rbm.W.data.t().clone(),
rbm.b_h.data.clone()))
    return pretrained_weights

def init_with_pretrain(net, pretrained_weights):
    for i, (w, b) in enumerate(pretrained_weights):
        net.layers[i].weight.data = w
        net.layers[i].bias.data = b

# Обучение
def train_model(net, X_train, y_train, X_test, y_test,
epochs=100, lr=0.001, batch_size=32):
    criterion = nn.BCELoss()
```

```
    optimizer = optim.Adam(net.parameters(), lr=lr)
    train_dataset = TensorDataset(X_train, y_train)
    train_loader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
    losses = []
    for epoch in range(epochs):
        net.train()
        epoch_loss = 0
        for data, target in train_loader:
            optimizer.zero_grad()
            output = net(data).squeeze()
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        losses.append(epoch_loss / len(train_loader))
    net.eval()
    with torch.no_grad():
        y_pred = (net(X_test).squeeze() >
0.5).float().cpu().numpy()
        f1 = f1_score(y_test.cpu().numpy(), y_pred)
        cm = confusion_matrix(y_test.cpu().numpy(), y_pred)
    return f1, cm, losses

def process_dataset(dataset_name, load_data_func):
    print(f"\n   {dataset_name} Dataset ")
    X, y = load_data_func()
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    X_train_tensor = torch.tensor(X_train,
dtype=torch.float32)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train,
dtype=torch.float32)
    y_test_tensor = torch.tensor(y_test, dtype=torch.float32)
    input_size = X_train.shape[1]
    hidden_sizes = [64, 32, 16]
    output_size = 1

    # 1. Без предобучения
    net_no_pretrain = ClassificationNet(input_size,
hidden_sizes, output_size)
    f1_no, cm_no, losses_no = train_model(net_no_pretrain,
X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor)
    print("\nWithout Pretraining:")
    print("F1 Score:", f1_no)
    print("Confusion Matrix:\n", cm_no)

    # 2. Autoencoder предобучение
    pretrained_weights_ae = pretrain_ae_layers(X_train_tensor,
hidden_sizes, epochs=50, lr=0.01)
```

```
    net_ae = ClassificationNet(input_size, hidden_sizes,
output_size)
    init_with_pretrain(net_ae, pretrained_weights_ae)
    f1_ae, cm_ae, losses_ae = train_model(net_ae,
X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor)
    print("\nWith Autoencoder Pretraining:")
    print("F1 Score:", f1_ae)
    print("Confusion Matrix:\n", cm_ae)

    # 3. RBM предобучение
    pretrained_weights_rbm =
pretrain_rbm_layers(X_train_tensor, hidden_sizes, epochs=50,
lr=0.01)
    net_rbm = ClassificationNet(input_size, hidden_sizes,
output_size)
    init_with_pretrain(net_rbm, pretrained_weights_rbm)
    f1_rbm, cm_rbm, losses_rbm = train_model(net_rbm,
X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor)
    print("\nWith RBM Pretraining:")
    print("F1 Score:", f1_rbm)
    print("Confusion Matrix:\n", cm_rbm)

    print("\nComparison:")
    print(f"F1 without pretrain: {f1_no:.4f} | F1 with AE:
{f1_ae:.4f} | F1 with RBM: {f1_rbm:.4f}")
    plt.figure(figsize=(10, 5))
    plt.plot(losses_no, label='No Pretrain')
    plt.plot(losses_ae, label='AE Pretrain')
    plt.plot(losses_rbm, label='RBM Pretrain')
    plt.title(f'Loss Curves - {dataset_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
    sns.heatmap(cm_no, annot=True, fmt='d', ax=axes[0],
cmap='Blues')
    axes[0].set_title(f'No Pretrain ({dataset_name})')
    sns.heatmap(cm_ae, annot=True, fmt='d', ax=axes[1],
cmap='Blues')
    axes[1].set_title(f'AE Pretrain ({dataset_name})')
    sns.heatmap(cm_rbm, annot=True, fmt='d', ax=axes[2],
cmap='Blues')
    axes[2].set_title(f'RBM Pretrain ({dataset_name})')
    plt.show()

    return f1_no, f1_ae, f1_rbm

if __name__ == "__main__":
    # Credit Approval ds
```

```
    f1_no_credit, f1_ae_credit, f1_rbm_credit =
process_dataset("Credit Approval", load_credit_data)
    # Breast Cancer ds
    f1_no_breast, f1_ae_breast, f1_rbm_breast =
process_dataset("Breast Cancer Wisconsin",
load_breast_cancer_data)

    print("\n    Final Comparison Across Datasets ")
    print(f"Credit Approval - F1 without: {f1_no_credit:.4f} |
AE: {f1_ae_credit:.4f} | RBM: {f1_rbm_credit:.4f}")
    print(f"Breast Cancer - F1 without: {f1_no_breast:.4f} |
AE: {f1_ae_breast:.4f} | RBM: {f1_rbm_breast:.4f}")
```

Результат работы программы:

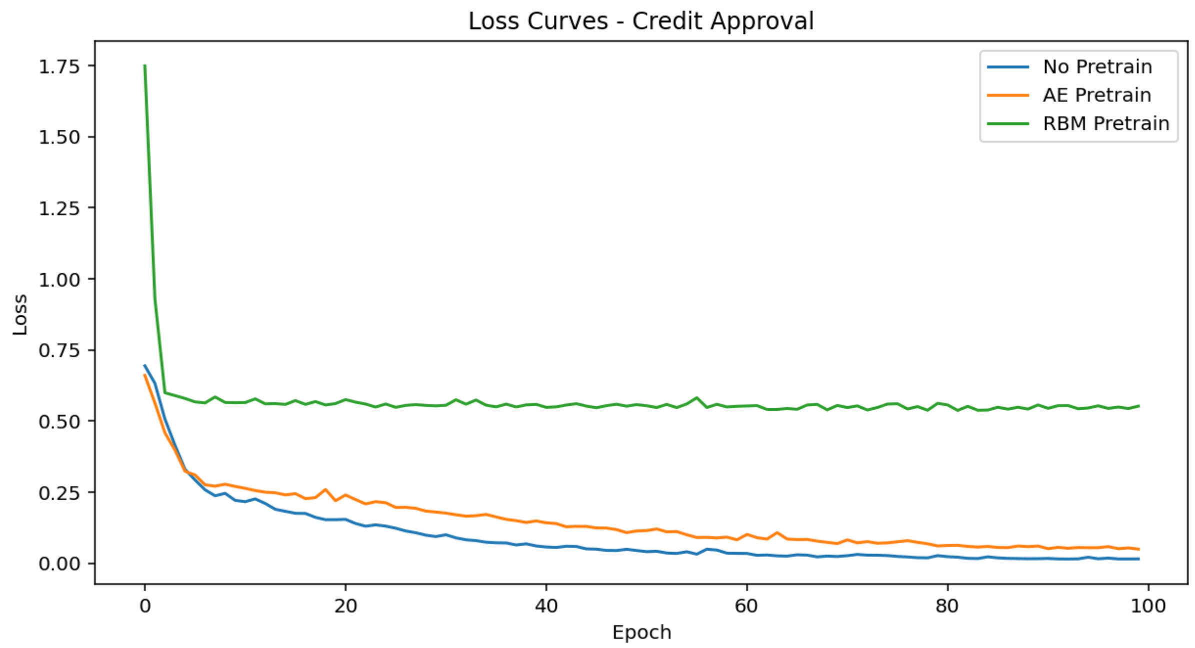
Credit Approval Dataset

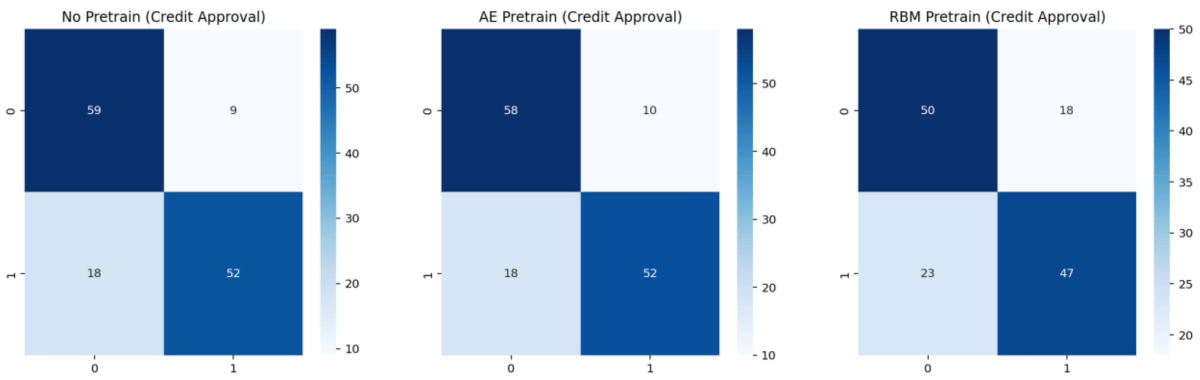
Without Pretraining:
F1 Score: 0.7938931297709924
Confusion Matrix:
[[59 9]
[18 52]]

With Autoencoder Pretraining:
F1 Score: 0.7878787878787878
Confusion Matrix:
[[58 10]
[18 52]]

With RBM Pretraining:
F1 Score: 0.6962962962962963
Confusion Matrix:
[[50 18]
[23 47]]

Comparison:
F1 without pretrain: 0.7939 | F1 with AE: 0.7879 | F1 with RBM: 0.6963





Breast Cancer Wisconsin Dataset

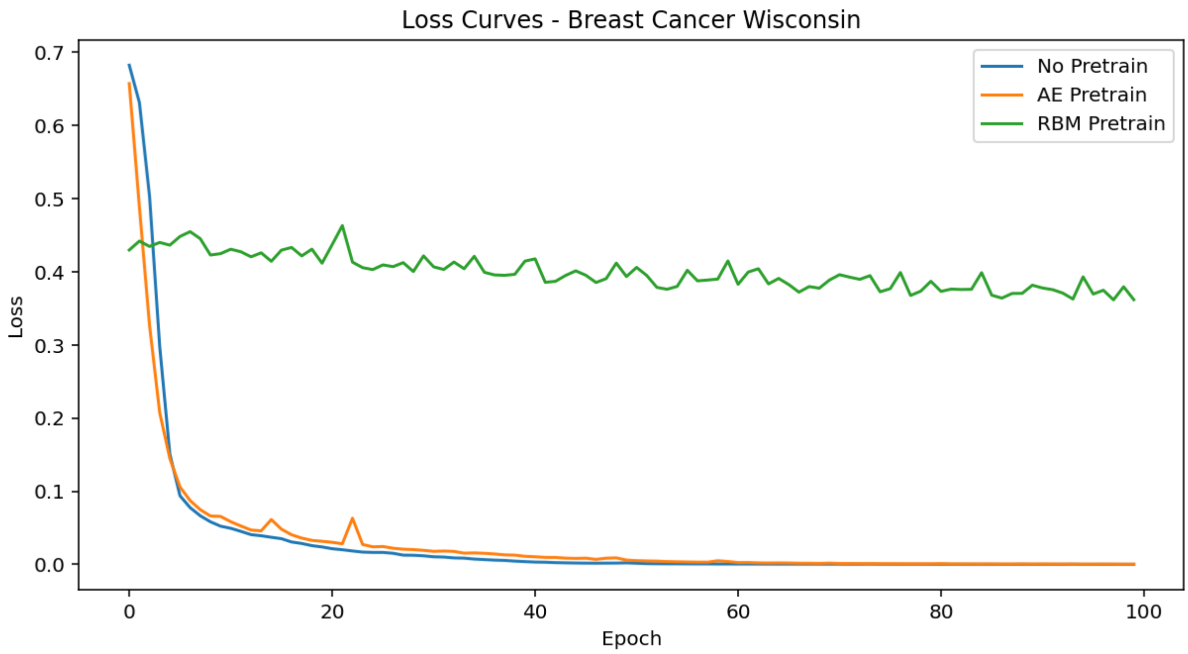
Without Pretraining:
F1 Score: 0.9655172413793104
Confusion Matrix:
[[69 2]
[1 42]]

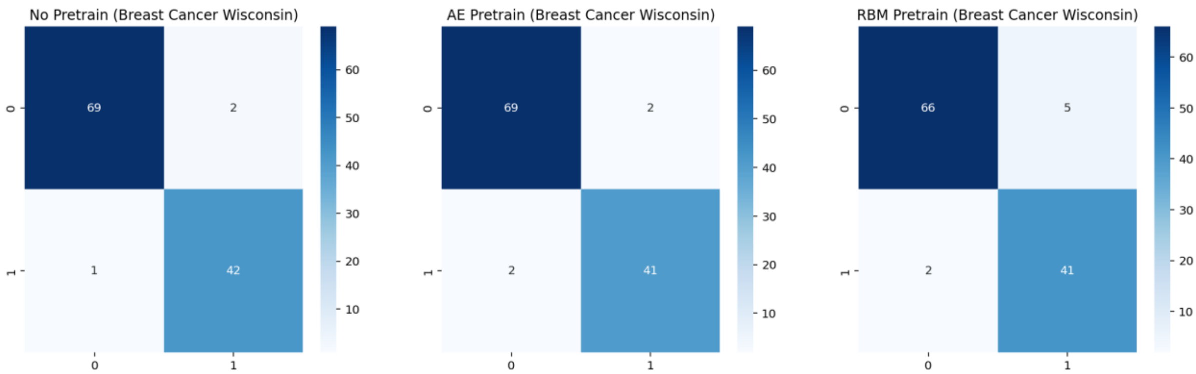
With Autoencoder Pretraining:
F1 Score: 0.9534883720930233
Confusion Matrix:
[[69 2]
[2 41]]

With RBM Pretraining:
F1 Score: 0.9213483146067416

Confusion Matrix:
[[66 5]
[2 41]]

Comparison:
F1 without pretrain: 0.9655 | F1 with AE: 0.9535 | F1 with RBM: 0.9213





Final Comparison Across Datasets
Credit Approval - F1 without: 0.7939 | AE: 0.7879 | RBM: 0.6963
Breast Cancer - F1 without: 0.9655 | AE: 0.9535 | RBM: 0.9213

Обучение без предобучения показало наилучшие результаты для обоих датасетов (F1 = 0.7939 для Credit Approval и 0.9655 для Breast Cancer). Предобучение с автоэнкодерами дало близкие, но слегка худшие результаты (F1 = 0.7879 и 0.9535 соответственно), тогда как RBM-предобучение значительно снизило производительность (F1 = 0.6963 и

0.9213). Breast Cancer оказался проще для классификации благодаря однородным данным, в то время как Credit Approval сложнее из-за смешанных признаков. Для повышения эффективности RBM требуется более тщательная настройка гиперпараметров, а стандартное обучение остается наиболее эффективным подходом..

Вывод: научилась осуществлять предобучение нейронных сетей с помощью RBM.