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

Выполнил:
студент 4 курса
группы ИИ-24
Крупич Д. Д.
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
7	https://archive.ics.uci.edu/dataset/503/hepatitis+c+virus+hcv+for+egyptian+patients	классификация	Baselinehistological staging

Код программы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, f1_score, accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F
import warnings

warnings.filterwarnings('ignore')

np.random.seed(42)
torch.manual_seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed(42)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
print(f"Using device: {device}")

def load_mushroom_data():
    print("Загрузка данных 'Mushroom'...")
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data"
    columns = ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
               'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
               'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
               'stalk-surface-below-ring', 'stalk-color-above-ring',
               'stalk-color-below-ring', 'veil-type', 'veil-color',
               'ring-number', 'ring-type', 'spore-print-color',
               'population', 'habitat']
    try:
        df = pd.read_csv(url, names=columns)
    except Exception as e:
        print(f"Ошибка загрузки данных Mushroom: {e}")
        return np.array([]), np.array([])

    X = df.drop('class', axis=1)
    y = df['class']

    for col in X.columns:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))

    le_y = LabelEncoder()
    y = le_y.fit_transform(y)

    print(f"Mushroom: X={X.shape}, y={y.shape}, classes={len(np.unique(y))}")
    return X.values, y

def load_hcv_data():
    print("Загрузка данных 'HCV (ID 503)'...")
    try:
        from ucimlrepo import fetch_ucirepo
        hcv_data = fetch_ucirepo(id=503)

        X = hcv_data.data.features
        y_df = hcv_data.data.targets

        target_col = 'Baselinehistological staging'

        if target_col not in y_df.columns:
            print(f"Ошибка: Целевая колонка '{target_col}' не найдена в данных.")
            return np.array([]), np.array([])

        df = pd.concat([X, y_df], axis=1)

        df = df.dropna(subset=[target_col])

        df = df.dropna()

        if df.empty:
            print("Ошибка: Нет данных после удаления NaN.")
            return np.array([]), np.array([])

        y = df[target_col]
        X = df.drop(columns=[target_col])

        for col in X.select_dtypes(include=['object', 'category']).columns:
            le = LabelEncoder()
            X[col] = le.fit_transform(X[col].astype(str))
```

```
        X = X.astype(float).values

        le_y = LabelEncoder()
        y = le_y.fit_transform(y)

        print(f"HCV (ID 503): X={X.shape}, y={y.shape}, classes={len(np.unique(y))}")
        return X, y

    except Exception as e:
        print(f"Ошибка загрузки данных HCV (ID 503) через ucimlrepo: {e}")
        return np.array([]), np.array([])

class GaussianBernoulliRBM(nn.Module):
    def __init__(self, n_visible, n_hidden):
        super(GaussianBernoulliRBM, self).__init__()
        self.W = nn.Parameter(torch.randn(n_hidden, n_visible) * 0.1)
        self.b = nn.Parameter(torch.zeros(n_visible))
        self.c = nn.Parameter(torch.zeros(n_hidden))

    def p_h_given_v(self, v):
        pre_sigmoid = torch.mm(v, self.W.t()) + self.c
        return torch.sigmoid(pre_sigmoid)

    def p_v_given_h(self, h):
        return torch.mm(h, self.W) + self.b

    def sample_bernoulli(self, p):
        return F.relu(torch.sign(p - torch.rand(p.size(), device=p.device)))

    def free_energy(self, v):
        v_bias_term = torch.sum(0.5 * (v - self.b)**2, dim=1)
        wx_c = torch.mm(v, self.W.t()) + self.c
        hidden_term = torch.sum(F.softplus(wx_c), dim=1)
        return torch.mean(v_bias_term - hidden_term)

def train_rbm(rbm, train_loader, epochs=50, lr=0.01, k=1):
    rbm.train()
    optimizer = optim.Adam(rbm.parameters(), lr=lr)

    for epoch in range(epochs):
        epoch_loss = 0
        for batch_x, _ in train_loader:
            batch_x = batch_x.to(device)

            v0 = batch_x
            h0_prob = rbm.p_h_given_v(v0)
            h0_sample = rbm.sample_bernoulli(h0_prob)

            vk_sample = h0_sample.detach()
            for _ in range(k):
                v_sample = rbm.p_v_given_h(vk_sample)
                hk_prob = rbm.p_h_given_v(v_sample)
                vk_sample = rbm.sample_bernoulli(hk_prob)

            vk = rbm.p_v_given_h(vk_sample)

            loss = rbm.free_energy(v0) - rbm.free_energy(vk.detach())

            epoch_loss += loss.item()

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

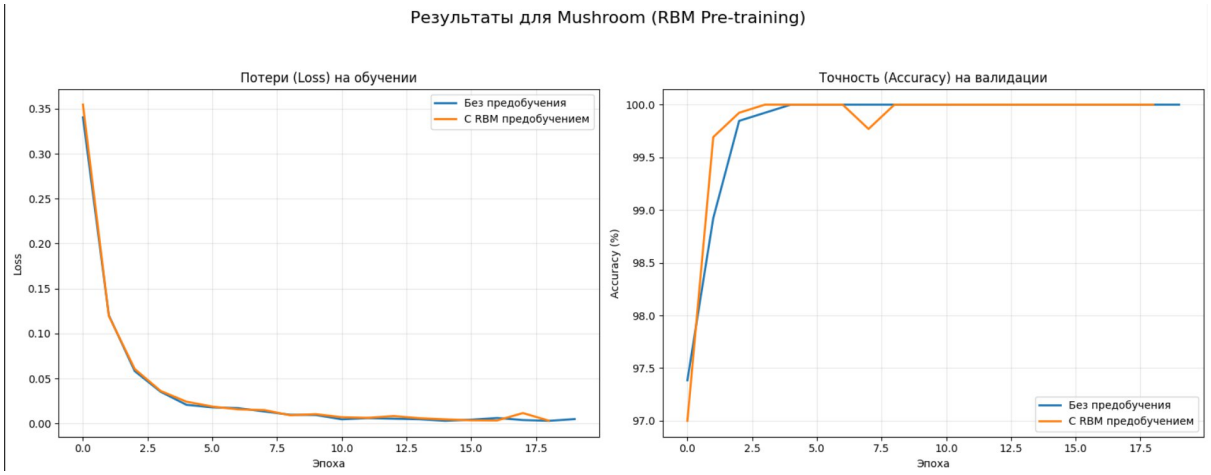
```
    return rbm

def pretrain_layers_rbm(X_train, hidden_dims, epochs_per_layer=50, k=1, lr=0.01):
    pretrained_weights = []
    current_input = X_train.clone().to(device)

    for i, hidden_dim in enumerate(hidden_dims):
        print(f"Pre-training RBM Layer {i+1}: {current_input.shape[1]} -> {hidden_dim}")

        rbm = GaussianBernoulliRBM(current_input.shape[1], hidden_dim).to(device)
        dataset = TensorDataset(current_input, torch.zeros(current_input.shape[0]))
        loader = DataLoader(dataset, batch_size=32, shuffle=True)

        rbm = train_rbm(rbm, loader, epochs=epochs_per_layer, lr=lr, k=k)

        pretrained_weights.append({
            'weight': rbm.W.data.clone(),
            'bias': rbm.c.data.clone()
        })

        rbm.eval()
        with torch.no_grad():
            current_input = rbm.p_h_given_v(current_input)

    return pretrained_weights

class ImprovedDeepNN(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim, dropout_rate=0.3):
        super(ImprovedDeepNN, self).__init__()
        layers = []
        prev_dim = input_dim
        for hidden_dim in hidden_dims:
            layers.extend([
                nn.Linear(prev_dim, hidden_dim),
                nn.BatchNorm1d(hidden_dim),
                nn.ReLU(),
                nn.Dropout(dropout_rate)
            ])
            prev_dim = hidden_dim
        layers.append(nn.Linear(prev_dim, output_dim))
        self.network = nn.Sequential(*layers)

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

def initialize_with_pretrained_weights(model, pretrained_weights):
    layer_idx = 0
    for module in model.network:
        if isinstance(module, nn.Linear) and layer_idx < len(pretrained_weights):
            print(f"Инициализация Linear слоя {layer_idx} весами RBM.")
            module.weight.data = pretrained_weights[layer_idx]['weight'].clone()
            module.bias.data = pretrained_weights[layer_idx]['bias'].clone()
            layer_idx += 1

def train_model(model, train_loader, val_loader, epochs=150, lr=0.001, patience=15):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', factor=0.5, patience=5)

    train_losses = []
    val_accuracies = []
    best_val_acc = 0
```

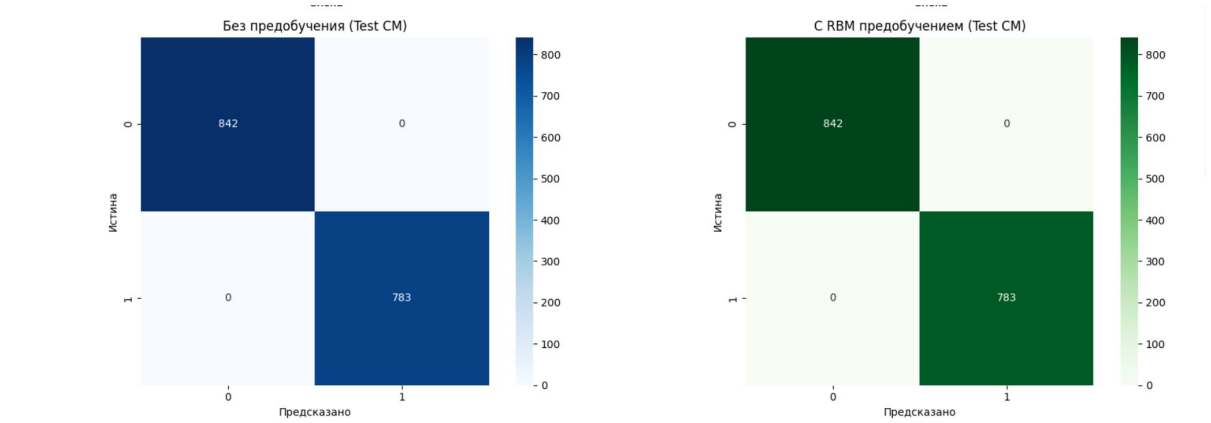
```
    patience_counter = 0

    for epoch in range(epochs):
        model.train()
        total_loss = 0
        for batch_x, batch_y in train_loader:
            batch_x, batch_y = batch_x.to(device), batch_y.to(device)

            optimizer.zero_grad()
            outputs = model(batch_x)
            loss = criterion(outputs, batch_y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for batch_x, batch_y in val_loader:
                batch_x, batch_y = batch_x.to(device), batch_y.to(device)
                outputs = model(batch_x)
                _, predicted = torch.max(outputs.data, 1)
                total += batch_y.size(0)
                correct += (predicted == batch_y).sum().item()

        val_acc = 100 * correct / total
        avg_loss = total_loss / len(train_loader)
        train_losses.append(avg_loss)
        val_accuracies.append(val_acc)

        scheduler.step(val_acc)

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping at epoch {epoch+1}")
            break

    return train_losses, val_accuracies

def evaluate_model(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_x = batch_x.to(device)
            outputs = model(batch_x)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(batch_y.numpy())

    accuracy = accuracy_score(all_labels, all_preds)
    f1_macro = f1_score(all_labels, all_preds, average='macro', zero_division=0)
    f1_weighted = f1_score(all_labels, all_preds, average='weighted', zero_division=0)
    cm = confusion_matrix(all_labels, all_preds)

    return {
```

```
        'accuracy': accuracy,
        'f1_macro': f1_macro,
        'f1_weighted': f1_weighted,
        'confusion_matrix': cm,
        'predictions': all_preds,
        'labels': all_labels
    }

def visualize_results(results_no_pretrain, results_pretrain,
                      train_losses_no_pretrain, train_losses_pretrain,
                      val_acc_no_pretrain, val_acc_pretrain, dataset_name):

    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    fig.suptitle(f'Результаты для {dataset_name} (RBM Pre-training)', fontsize=16, y=0.995)

    axes[0, 0].plot(train_losses_no_pretrain, label='Без предобучения', linewidth=2)
    axes[0, 0].plot(train_losses_pretrain, label='С RBM предобучением', linewidth=2)
    axes[0, 0].set_title('Потери (Loss) на обучении')
    axes[0, 0].set_xlabel('Эпоха')
    axes[0, 0].set_ylabel('Loss')
    axes[0, 0].legend()
    axes[0, 0].grid(True, alpha=0.3)

    axes[0, 1].plot(val_acc_no_pretrain, label='Без предобучения', linewidth=2)
    axes[0, 1].plot(val_acc_pretrain, label='С RBM предобучением', linewidth=2)
    axes[0, 1].set_title('Точность (Accuracy) на валидации')
    axes[0, 1].set_xlabel('Эпоха')
    axes[0, 1].set_ylabel('Accuracy (%)')
    axes[0, 1].legend()
    axes[0, 1].grid(True, alpha=0.3)

    sns.heatmap(results_no_pretrain['confusion_matrix'], annot=True, fmt='d', cmap='Blues',
                ax=axes[1, 0], square=True)
    axes[1, 0].set_title('Без предобучения (Test CM)')
    axes[1, 0].set_xlabel('Предсказано')
    axes[1, 0].set_ylabel('Истина')

    sns.heatmap(results_pretrain['confusion_matrix'], annot=True, fmt='d', cmap='Greens',
                ax=axes[1, 1], square=True)
    axes[1, 1].set_title('С RBM предобучением (Test CM)')
    axes[1, 1].set_xlabel('Предсказано')
    axes[1, 1].set_ylabel('Истина')

    plt.tight_layout(rect=[0, 0.03, 1, 0.97])
    filename = f'{dataset_name.replace(" ", "_").replace("(", "").replace(")", "")}_rbm_results.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()

    print(f"\n--- Итоги для {dataset_name} ---")
    print(f"Без предобучения:  Test Acc={results_no_pretrain['accuracy']:.4f}, Test F1 (Macro)={results_no_pretrain['f1_macro']:.4f}")
    print(f"С RBM предобучением: Test Acc={results_pretrain['accuracy']:.4f}, Test F1 (Macro)={results_pretrain['f1_macro']:.4f}")

def run_experiment(X, y, dataset_name, hidden_dims=[128, 64, 32]):

    if X.size == 0 or y.size == 0:
        print(f"Пропуск эксперимента {dataset_name}: нет данных.")
        return None, None

    try:
        X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
        X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42, stratify=y_temp)
    except ValueError:
        print(f"Стратификация не удалась для {dataset_name}. Используется обычное разделение.")
```

```
        X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    X_train_tensor = torch.FloatTensor(X_train)
    y_train_tensor = torch.LongTensor(y_train)
    X_val_tensor = torch.FloatTensor(X_val)
    y_val_tensor = torch.LongTensor(y_val)
    X_test_tensor = torch.FloatTensor(X_test)
    y_test_tensor = torch.LongTensor(y_test)

    train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
    val_dataset = TensorDataset(X_val_tensor, y_val_tensor)
    test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

    input_dim = X_train.shape[1]
    output_dim = len(np.unique(y))

    print(f"--- {dataset_name}: Обучение модели БЕЗ предобучения ---")
    model_no_pretrain = ImprovedDeepNN(input_dim, hidden_dims, output_dim).to(device)
    train_losses_no_pretrain, val_acc_no_pretrain = train_model(model_no_pretrain, train_loader, val_loader)
    results_no_pretrain = evaluate_model(model_no_pretrain, test_loader)

    print(f"\n--- {dataset_name}: Запуск RBM предобучения ---")
    pretrained_weights = pretrain_layers_rbm(X_train_tensor, hidden_dims, epochs_per_layer=50, k=1, lr=0.01)

    model_pretrain = ImprovedDeepNN(input_dim, hidden_dims, output_dim).to(device)

    initialize_with_pretrained_weights(model_pretrain, pretrained_weights)

    print(f"--- {dataset_name}: Дообучение (fine-tuning) модели с RBM ---")
    train_losses_pretrain, val_acc_pretrain = train_model(model_pretrain, train_loader, val_loader)
    results_pretrain = evaluate_model(model_pretrain, test_loader)

    visualize_results(results_no_pretrain, results_pretrain,
                      train_losses_no_pretrain, train_losses_pretrain,
                      val_acc_no_pretrain, val_acc_pretrain,
                      dataset_name)

    return results_no_pretrain, results_pretrain

def main():
    hidden_dims = [128, 64, 32]

    X_mushroom, y_mushroom = load_mushroom_data()
    run_experiment(X_mushroom, y_mushroom, "Mushroom", hidden_dims=hidden_dims)

    X_hcv, y_hcv = load_hcv_data()

    run_experiment(X_hcv, y_hcv, "HCV (ID 503)", hidden_dims=hidden_dims)


if __name__ == "__main__":
    main()
```

Вывод программы:






```
--- Итоги для Mushroom ---
Без предобучения:  Test Acc=1.0000, Test F1 (Macro)=1.0000
С RBM предобучением: Test Acc=1.0000, Test F1 (Macro)=1.0000
```

Сравнение результатов:

Лаба 3 (Автоэнкодер): На датасете HCV (ID 571) предобучение ухудшило результат. Модель "с нуля" была точнее (93.2%), чем модель с автоэнкодером (92.4%).

Лаба 4 (RBM): На датасете HCV (ID 503) предобучение улучшило результат. Модель "с нуля" провалилась (20.9% — хуже случайного), а модель с RBM показала 24.9%.

Mushroom: На этом легком датасете обе модели (AE и RBM) всегда показывали 100%.

Вывод: я научился осуществлять предобучение нейронных сетей с помощью RBM