Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра интеллектуально-информационных технологий

Лабораторная работа №4
По дисциплине «Интеллектуальный анализ данных»

Тема: «Предобучение нейронных сетей с использованием RBM»

Выполнила:
студентка 4 курса
группы ИИ-24
Коцуба Е.М.
Проверила:
Андренко К.В.

Брест 2025

Цель работы: научиться осуществлять предобучение нейронных сетей с помощью RBM.

Вариант 5

№	Выборка	Тип задачи	Целевая переменная
5	https://archive.ics.uci.edu/dataset/193/cardiotocography	классификация	CLASS/NSP

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.

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from ucimlrepo import fetch_ucirepo
import warnings
warnings.filterwarnings("ignore")

plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
figsize = (10, 6)

def prepare_data(X, y, test_size=0.2, random_state=42,
normalize_to_01=False):
    mask = ~np.isnan(X).any(axis=1)
    X, y = X[mask], y[mask]
```

```
    X_tr, X_te, y_tr, y_te = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y
    )
    scaler = StandardScaler()
    X_tr = scaler.fit_transform(X_tr)
    X_te = scaler.transform(X_te)

    if normalize_to_01:
        X_tr = (X_tr - X_tr.min()) / (X_tr.max() - X_tr.min() + 1e-8)
        X_te = (X_te - X_te.min()) / (X_te.max() - X_te.min() + 1e-8)

    X_tr_t = torch.FloatTensor(X_tr)
    X_te_t = torch.FloatTensor(X_te)
    y_tr_t = torch.LongTensor(y_tr)
    y_te_t = torch.LongTensor(y_te)

    loader = DataLoader(TensorDataset(X_tr_t, y_tr_t), batch_size=32,
shuffle=True)
    return X_tr_t, X_te_t, y_tr_t, y_te_t, loader

def create_model(in_dim, out_dim):
    class Net(nn.Module):
        def __init__(self):
            super().__init__()
            self.fc1 = nn.Linear(in_dim, 128)
            self.fc2 = nn.Linear(128, 64)
            self.fc3 = nn.Linear(64, 32)
            self.fc4 = nn.Linear(32, 16)
            self.fc5 = nn.Linear(16, out_dim)

        def forward(self, x):
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = F.relu(self.fc3(x))
            x = F.relu(self.fc4(x))
            return self.fc5(x)
    return Net()

class RBM(nn.Module):
    def __init__(self, n_visible, n_hidden):
        super().__init__()
        self.W = nn.Parameter(torch.randn(n_hidden, n_visible) * 0.01)
        self.h_bias = nn.Parameter(torch.zeros(n_hidden))
        self.v_bias = nn.Parameter(torch.zeros(n_visible))

    def sample_h(self, v):
        wx = torch.mm(v, self.W.t())
        activation = wx + self.h_bias
        p_h = torch.sigmoid(activation)
        return p_h, torch.bernoulli(p_h)

    def sample_v(self, h):
        wx = torch.mm(h, self.W)
        activation = wx + self.v_bias
        p_v = torch.sigmoid(activation)
        return p_v, torch.bernoulli(p_v)


def train_rbm(rbm, data_loader, epochs=50, lr=0.01, k=1):
    optimizer = optim.Adam(rbm.parameters(), lr=lr)
    rbm.train()
    losses = []
    for ep in range(epochs):
        epoch_loss = 0.0
```

```
        for x, _ in data_loader:
            x = x.clone()
            batch_size = x.size(0)

            v = x
            h = torch.bernoulli(torch.sigmoid(torch.mm(v, rbm.W.t()) +
rbm.h_bias))
            for _ in range(k):
                v = torch.bernoulli(torch.sigmoid(torch.mm(h, rbm.W) +
rbm.v_bias))
                h = torch.bernoulli(torch.sigmoid(torch.mm(v, rbm.W.t()) +
rbm.h_bias))

            pos_h = torch.sigmoid(torch.mm(x, rbm.W.t()) + rbm.h_bias)
            neg_h = torch.sigmoid(torch.mm(v, rbm.W.t()) + rbm.h_bias)

            dW = (torch.mm(x.t(), pos_h) - torch.mm(v.t(), neg_h)) /
batch_size
            dv = (x - v).sum(0) / batch_size
            dh = (pos_h - neg_h).sum(0) / batch_size

            optimizer.zero_grad()
            rbm.W.grad = -dW.t()
            rbm.v_bias.grad = -dv
            rbm.h_bias.grad = -dh
            optimizer.step()

            recon_loss = F.mse_loss(v, x)
            epoch_loss += recon_loss.item()

        avg_loss = epoch_loss / len(data_loader)
        losses.append(avg_loss)
        if (ep + 1) % 20 == 0:
            print(f'    RBM Эпоха {ep+1}/{epochs}, Recon Loss:
{avg_loss:.6f}')
    return losses

def pretrain_with_rbm(model, X_tr, layers, rbm_epochs=20):
    print('  Предобучение с помощью RBM...')
    input_data = X_tr.clone()
    all_rbm_losses = []

    for i, (in_sz, hid_sz) in enumerate(layers):
        print(f'  Обучение RBM {i+1}: {in_sz} → {hid_sz}')
        rbm = RBM(n_visible=in_sz, n_hidden=hid_sz)
        loader = DataLoader(TensorDataset(input_data,
torch.zeros(len(input_data))), batch_size=32, shuffle=True)
        losses = train_rbm(rbm, loader, epochs=rbm_epochs, lr=0.01, k=1)
        all_rbm_losses.append(losses)

        fc = getattr(model, f'fc{i+1}')
        with torch.no_grad():
            fc.weight.data = rbm.W.clone()
            fc.bias.data   = rbm.h_bias.clone()

        with torch.no_grad():
            p_h, _ = rbm.sample_h(input_data)
            input_data = p_h

    print('  Предобучение RBM завершено.')
    return all_rbm_losses

def train_autoencoder(in_sz, hid_sz, loader, epochs=50):
    class AE(nn.Module):
```

```
        def __init__(self):
            super().__init__()
            self.enc = nn.Linear(in_sz, hid_sz)
            self.dec = nn.Linear(hid_sz, in_sz)
        def forward(self, x):
            return self.dec(F.relu(self.enc(x)))

    ae = AE()
    crit = nn.MSELoss()
    opt = optim.Adam(ae.parameters(), lr=0.001)
    for _ in range(epochs):
        ae.train()
        for x, _ in loader:
            opt.zero_grad()
            loss = crit(ae(x), x)
            loss.backward()
            opt.step()
    return ae.enc


def pretrain_model(model, X_tr, layers):
    print('  Предобучение автоэнкодерами...')
    loader = DataLoader(TensorDataset(X_tr, torch.zeros(len(X_tr))),
batch_size=32, shuffle=True)
    enc = train_autoencoder(layers[0][0], layers[0][1], loader)
    model.fc1.weight.data = enc.weight.data
    model.fc1.bias.data   = enc.bias.data

    hidden = X_tr
    for i in range(1, len(layers)):
        with torch.no_grad():
            if i == 1: hidden = F.relu(model.fc1(hidden))
            elif i == 2: hidden = F.relu(model.fc2(hidden))
            elif i == 3: hidden = F.relu(model.fc3(hidden))
        loader = DataLoader(TensorDataset(hidden, torch.zeros(len(hidden))),
batch_size=32, shuffle=True)
        enc = train_autoencoder(layers[i][0], layers[i][1], loader)
        fc = getattr(model, f'fc{i+1}')
        fc.weight.data = enc.weight.data
        fc.bias.data   = enc.bias.data
    print('  Предобучение завершено.')

def train_supervised(model, loader, epochs=50, lr=0.001, label=''):
    crit = nn.CrossEntropyLoss()
    opt = optim.Adam(model.parameters(), lr=lr)
    every = 20 if epochs >= 50 else 10
    model.train()
    losses = []
    for ep in range(epochs):
        epoch_loss = 0.0
        for x, y in loader:
            opt.zero_grad()
            output = model(x)
            loss = crit(output, y)
            loss.backward()
            opt.step()
            epoch_loss += loss.item()
        avg_loss = epoch_loss / len(loader)
        losses.append(avg_loss)
        if (ep + 1) % every == 0:
            print(f'  [{label}] Эпоха {ep+1}/{epochs}, Loss: {avg_loss:.6f}')
    return losses

def evaluate(model, X_te, y_te):
```

```
    model.eval()
    with torch.no_grad():
        pred = model(X_te).argmax(dim=1).cpu().numpy()
    y_true = y_te.cpu().numpy()
    acc = accuracy_score(y_true, pred)
    f1 = f1_score(y_true, pred, average='weighted')
    cm = confusion_matrix(y_true, pred)
    return acc, f1, cm

def plot_rbm_losses(all_rbm_losses, layers):
    plt.figure(figsize=figsize)
    for i, losses in enumerate(all_rbm_losses):
        plt.plot(losses, label=f'RBM {i+1}: {layers[i][0]}→{layers[i][1]}')
    plt.title('RBM Reconstruction Loss по слоям')
    plt.xlabel('Эпоха')
    plt.ylabel('MSE Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()


def plot_supervised_losses(losses_dict):
    plt.figure(figsize=figsize)
    for label, losses in losses_dict.items():
        plt.plot(losses, label=label, marker='o', markevery=5)
    plt.title('Cross-Entropy Loss во время финального обучения')
    plt.xlabel('Эпоха')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()


def plot_metrics_comparison(results):
    methods = list(results.keys())
    accs = [r[0] for r in results.values()]
    f1s = [r[1] for r in results.values()]

    x = np.arange(len(methods))
    width = 0.35

    fig, ax = plt.subplots(figsize=figsize)
    bars1 = ax.bar(x - width/2, accs, width, label='Accuracy',
color='skyblue')
    bars2 = ax.bar(x + width/2, f1s, width, label='F1-score',
color='lightcoral')

    ax.set_ylabel('Score')
    ax.set_title('Сравнение Accuracy и F1-score')
    ax.set_xticks(x)
    ax.set_xticklabels(methods)
    ax.legend()
    ax.set_ylim(0.8, 1.0)

    for bar in bars1 + bars2:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height + 0.001,
                f'{height:.4f}', ha='center', va='bottom', fontsize=9)

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

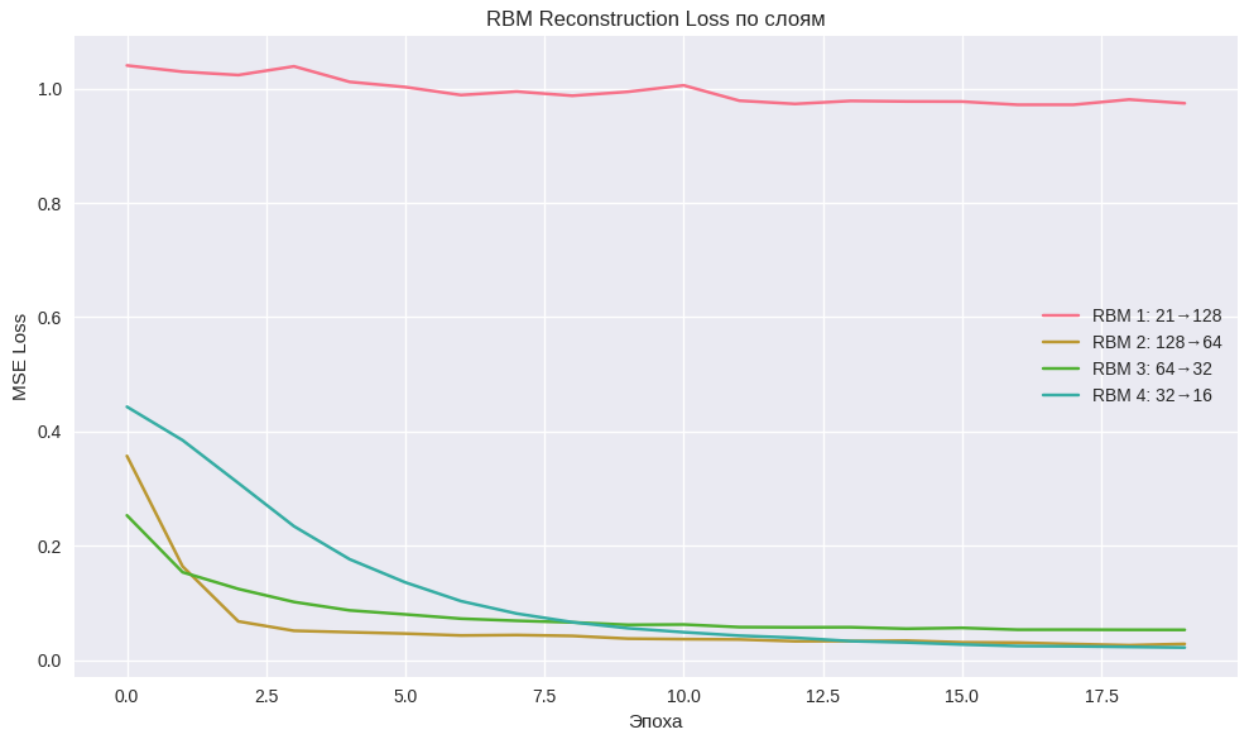
```
def plot_confusion_matrices(cms, labels, class_names):
    fig, axes = plt.subplots(1, 3, figsize=(15, 4))
    for i, (cm, label) in enumerate(zip(cms, labels)):
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[i],
                    xticklabels=class_names, yticklabels=class_names)
        axes[i].set_title(f'Confusion Matrix: {label}')
        axes[i].set_xlabel('Предсказано')
        axes[i].set_ylabel('Истинно')
    plt.tight_layout()
    plt.show()

def run_experiment(dataset_name, dataset_id, target_col=None,
class_names=None, normalize_to_01=False):
    print(f"\n{'='*20} {dataset_name} {'='*20}")

    data = fetch_ucirepo(id=dataset_id)
    X = data.data.features.values
    if target_col:
        y = (data.data.targets[target_col] - 1).values if dataset_id == 193
else data.data.targets.values.flatten()
    else:
        y = data.data.targets.values.flatten()

    X_tr_t, X_te_t, y_tr_t, y_te_t, train_loader = prepare_data(X, y,
normalize_to_01=normalize_to_01)
    in_dim = X.shape[1]
    n_cls = len(np.unique(y))
    print(f"  Признаков: {in_dim}, Классов: {n_cls}, Объектов: {len(y)}")

    layers = [(in_dim, 128), (128, 64), (64, 32), (32, 16)]
    epochs_no = 100
    epochs_pre = 30
    rbm_epochs = 20

    results = {}
    cms = {}
    supervised_losses = {}

    print("\n1. Обучение без предобучения")
    model_no = create_model(in_dim, n_cls)
    loss_no = train_supervised(model_no, train_loader, epochs=epochs_no,
lr=0.001, label="Без")
    acc_no, f1_no, cm_no = evaluate(model_no, X_te_t, y_te_t)
    results["Без"] = (acc_no, f1_no)
    cms["Без"] = cm_no
    supervised_losses["Без"] = loss_no

    print("\n2. Обучение с предобучением (автоэнкодеры)")
    model_ae = create_model(in_dim, n_cls)
    pretrain_model(model_ae, X_tr_t, layers)
    loss_ae = train_supervised(model_ae, train_loader, epochs=epochs_pre,
lr=0.001, label="AE")
    acc_ae, f1_ae, cm_ae = evaluate(model_ae, X_te_t, y_te_t)
    results["AE"] = (acc_ae, f1_ae)
    cms["AE"] = cm_ae
    supervised_losses["AE"] = loss_ae

    print("\n3. Обучение с предобучением (RBM)")
    model_rbm = create_model(in_dim, n_cls)
    rbm_losses = pretrain_with_rbm(model_rbm, X_tr_t, layers,
rbm_epochs=rbm_epochs)
    loss_rbm = train_supervised(model_rbm, train_loader, epochs=epochs_pre,
lr=0.001, label="RBM")
```

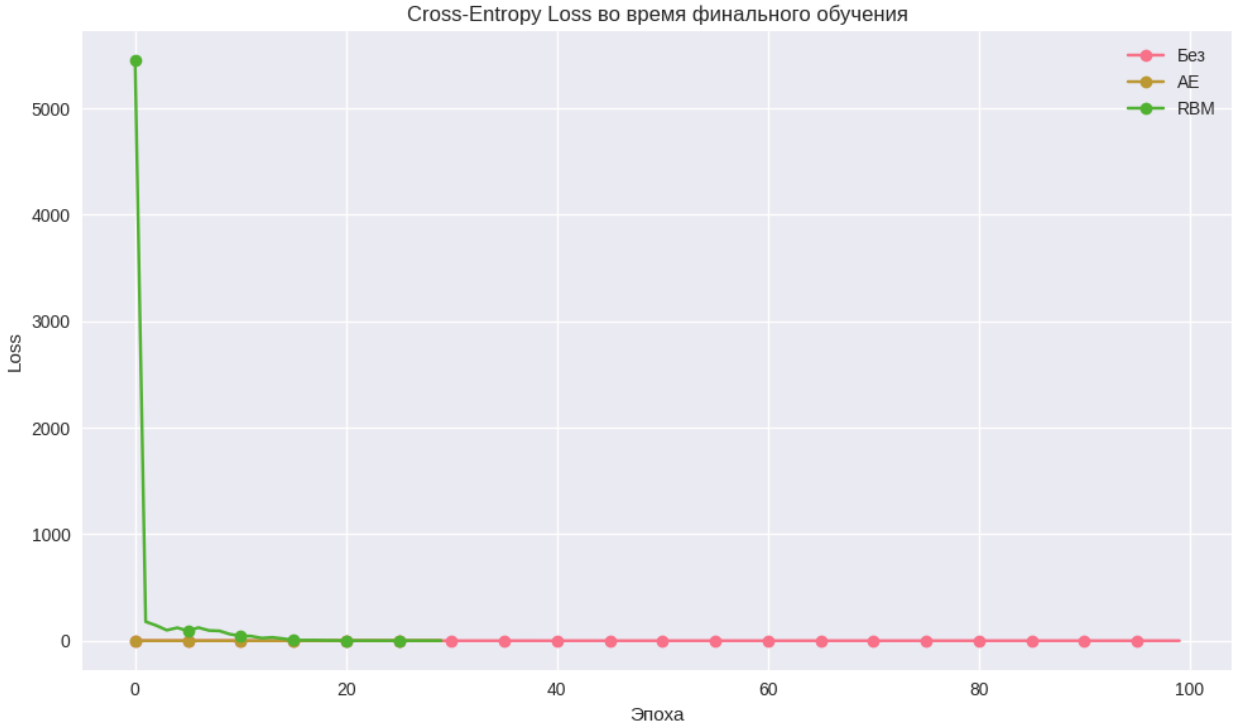
```
    acc_rbm, f1_rbm, cm_rbm = evaluate(model_rbm, X_te_t, y_te_t)
    results["RBM"] = (acc_rbm, f1_rbm)
    cms["RBM"] = cm_rbm
    supervised_losses["RBM"] = loss_rbm

    print("\n" + "="*80)
    print(f"Итоговая таблица: {dataset_name}")
    print("="*80)
    print(f"{'Метод':<25} {'Accuracy':<10} {'F1-score':<10} {'ΔAcc':<8}
{'ΔF1'}")
    print("-"*80)
    base_acc, base_f1 = results["Без"]
    print(f"{'Без предобучения':<25} {base_acc:.4f}     {base_f1:.4f}     {'—
':<8} {'—'}")
    for method in ["AE", "RBM"]:
        acc, f1 = results[method]
        print(f"{'+ ' + method:<24} {acc:.4f}     {f1:.4f}     {acc-
base_acc:+.4f}   {f1-base_f1:+.4f}")
    print("="*80)

    plot_rbm_losses(rbm_losses, layers)
    plot_supervised_losses(supervised_losses)
    plot_metrics_comparison(results)
    plot_confusion_matrices([cms["Без"], cms["AE"], cms["RBM"]], ["Без",
"AE", "RBM"], class_names or [str(i) for i in range(n_cls)])

    return results

results_cardio = run_experiment(
    dataset_name="Cardiotocography (NSP)",
    dataset_id=193,
    target_col='NSP',
    class_names=['N', 'S', 'P'],
    normalize_to_01=False
)

results_digits = run_experiment(
    dataset_name="Optical Recognition of Handwritten Digits",
    dataset_id=80,
    target_col=None,
    class_names=[str(i) for i in range(10)],
    normalize_to_01=True
)
```

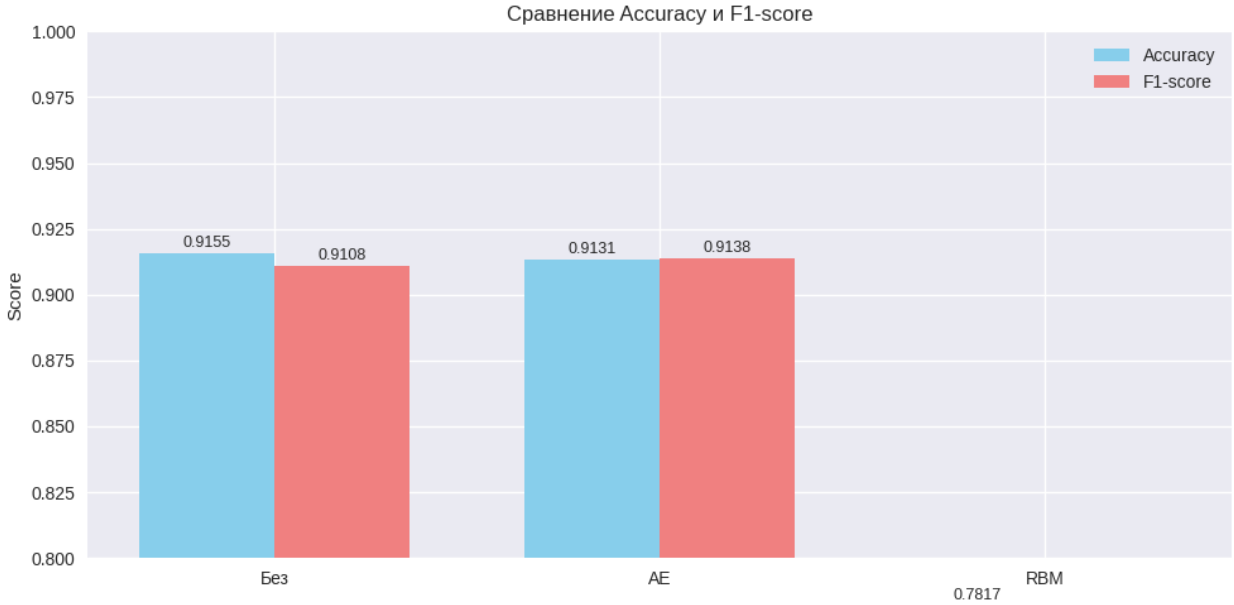
Результат работы программы:

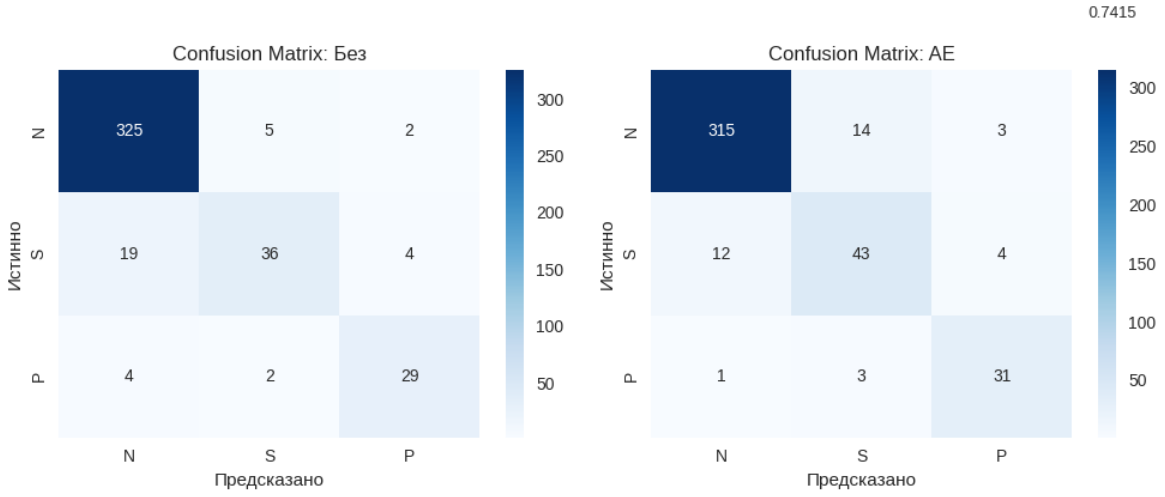
```
  Cardiotocography (NSP)
  Признаков: 21, Классов: 3, Объектов: 2126

1. Обучение без предобучения
  [Без] Эпоха 20/100, Loss: 0.096392
  [Без] Эпоха 40/100, Loss: 0.049392
  [Без] Эпоха 60/100, Loss: 0.033221
  [Без] Эпоха 80/100, Loss: 0.024033
  [Без] Эпоха 100/100, Loss: 0.012229

2. Обучение с предобучением (автоэнкодеры)
  Предобучение автоэнкодерами...
  Предобучение завершено.
  [AE] Эпоха 10/30, Loss: 0.183603
  [AE] Эпоха 20/30, Loss: 0.124219
  [AE] Эпоха 30/30, Loss: 0.106021
```

```
3. Обучение с предобучением (RBM)
  Предобучение с помощью RBM...
  Обучение RBM 1: 21 → 128
    RBM Эпоха 20/20, Recon Loss: 0.974276
  Обучение RBM 2: 128 → 64
    RBM Эпоха 20/20, Recon Loss: 0.028108
  Обучение RBM 3: 64 → 32
    RBM Эпоха 20/20, Recon Loss: 0.052773
  Обучение RBM 4: 32 → 16
    RBM Эпоха 20/20, Recon Loss: 0.021836
  Предобучение RBM завершено.
  [RBM] Эпоха 10/30, Loss: 60.959370
  [RBM] Эпоха 20/30, Loss: 2.568607
  [RBM] Эпоха 30/30, Loss: 0.758146

================================================================================
Итоговая таблица: Cardiotocography (NSP)
================================================================================
Метод                       Accuracy   F1-score   ΔAcc     ΔF1
--------------------------------------------------------------------------------
Без предобучения             0.9155     0.9108     —        —
+ AE                        0.9131     0.9138     -0.0023   +0.0030
+ RBM                       0.7817     0.7415     -0.1338   -0.1693
================================================================================
```

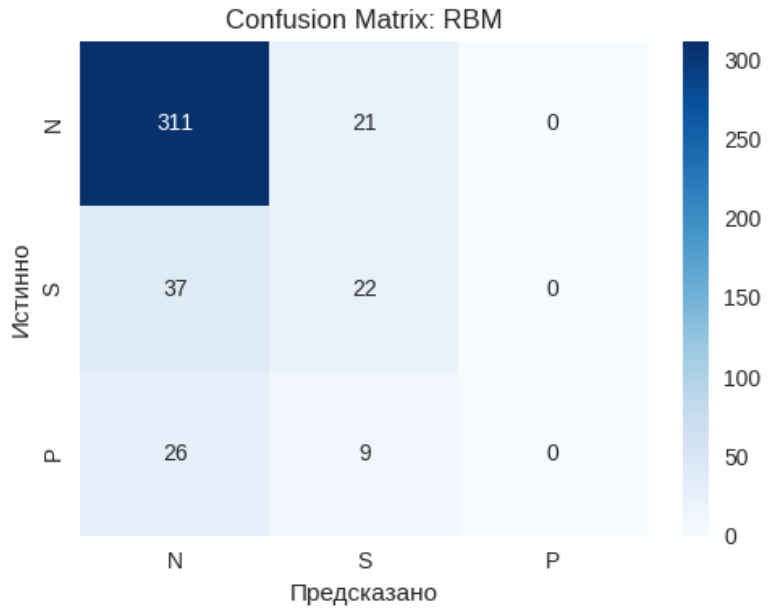



Cross-Entropy Loss во время финального обучения
Без
AE
RBM
Loss
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
Эпоха

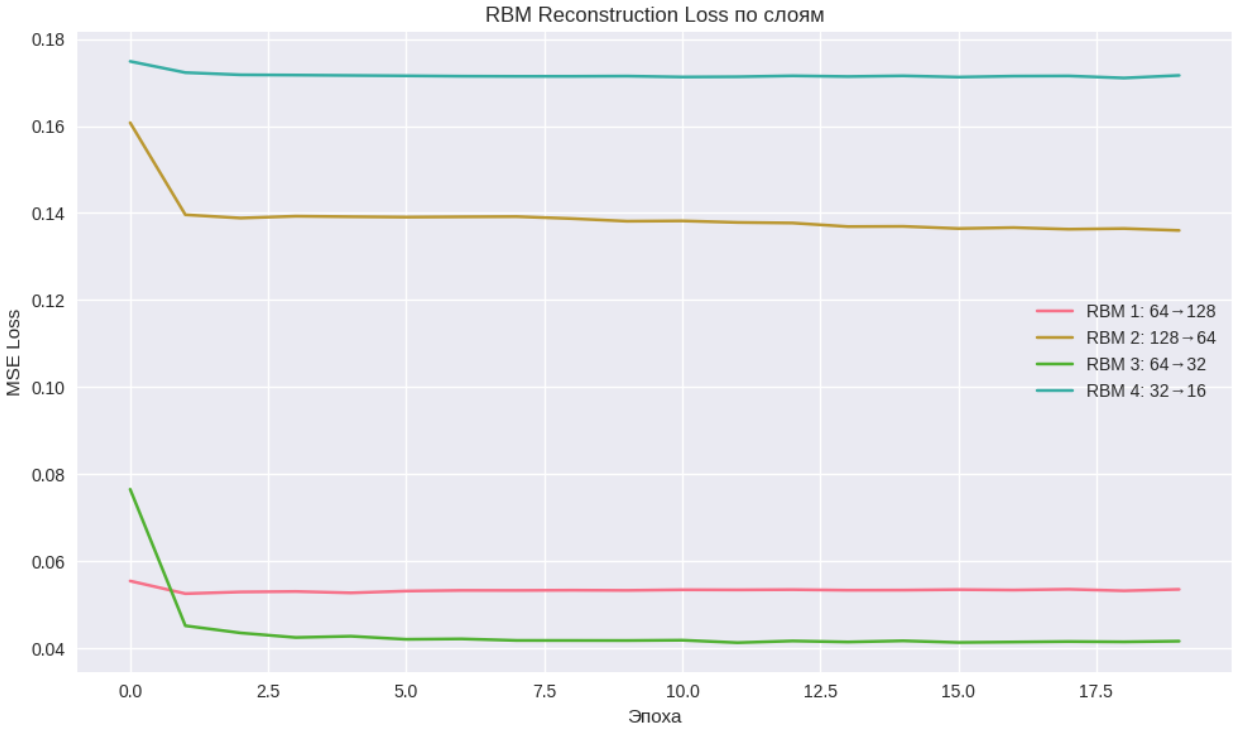

Сравнение Accuracy и F1-score
Accuracy
F1-score
Score
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.9155
0.9108
0.9131
0.9138
Без
AE
RBM
0.7817

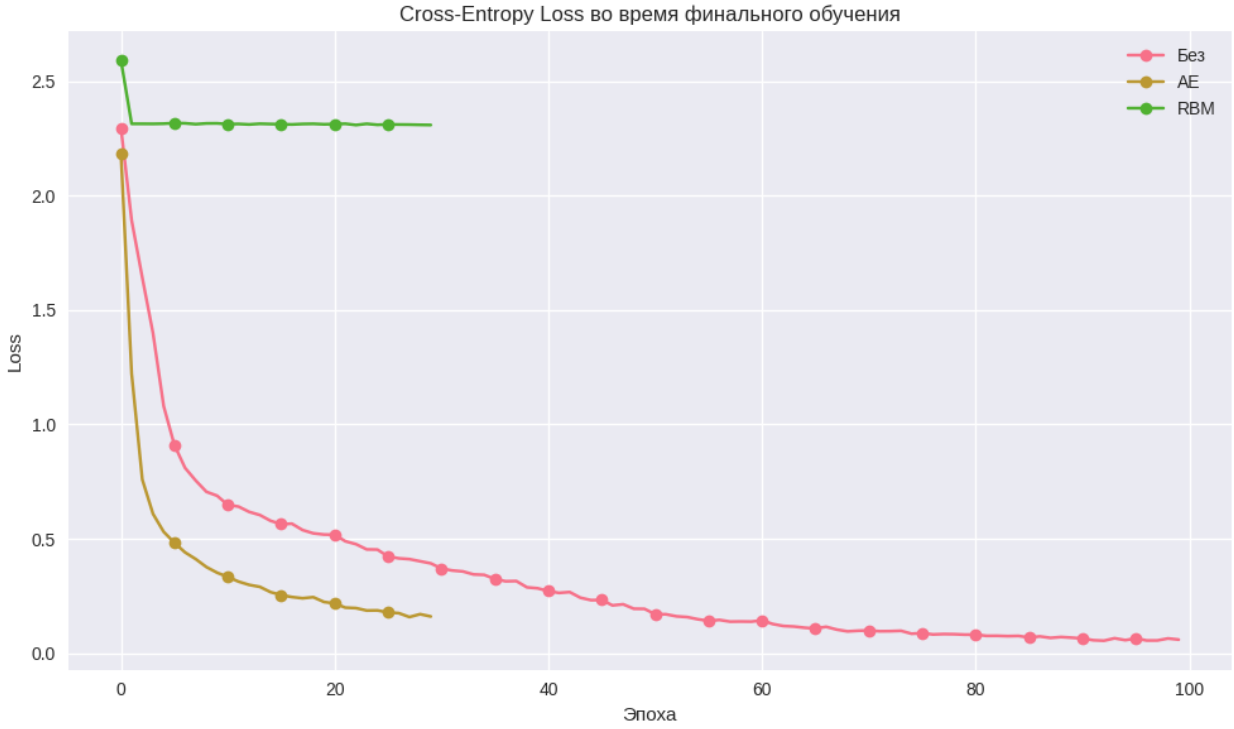

Confusion Matrix: Без
Истинно
N
S
P
325
5
2
19
36
4
4
2
29
300
250
200
150
100
50
N
S
P
Предсказано

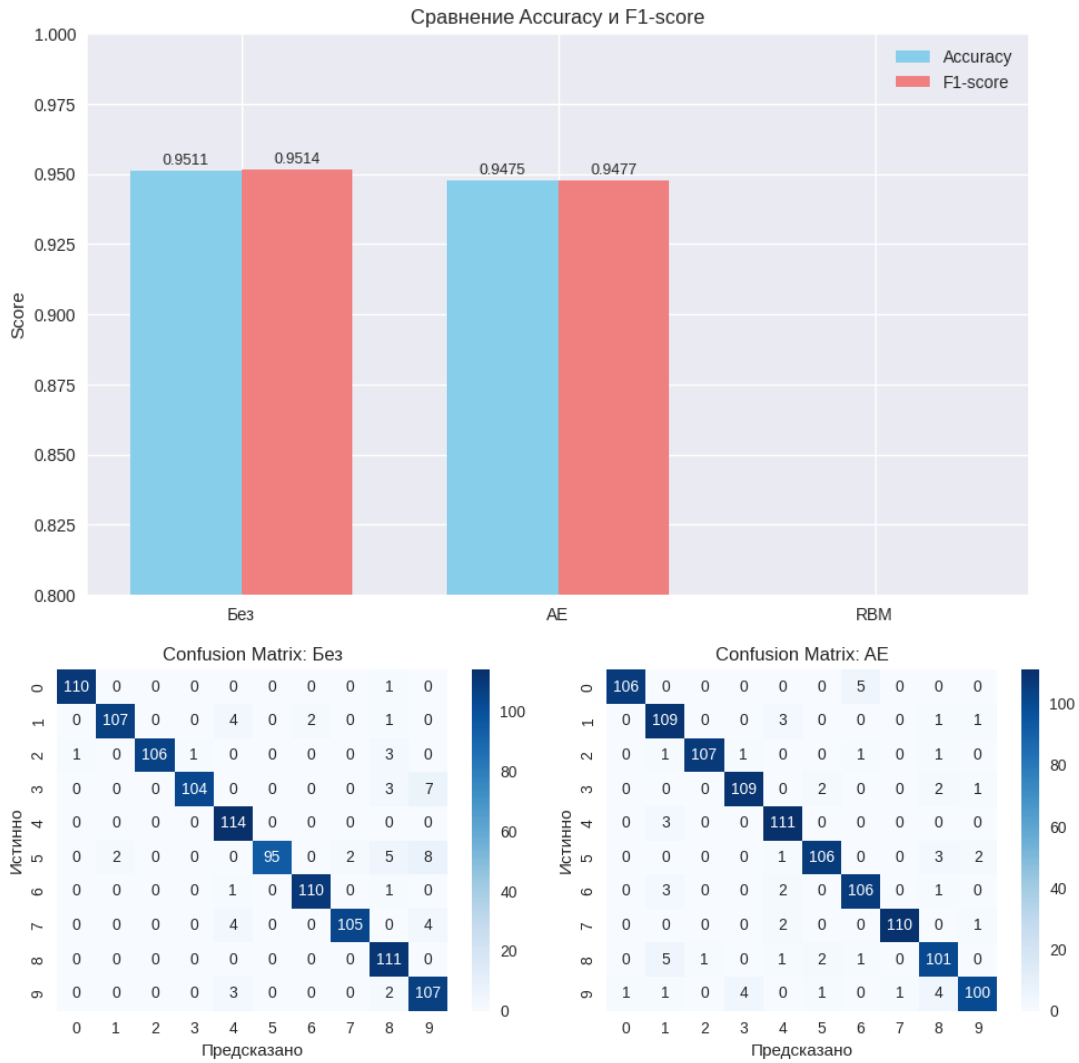
0.7415
Confusion Matrix: AE
Истинно
N
S
P
315
14
3
12
43
4
1
3
31
300
250
200
150
100
50
N
S
P
Предсказано




```
Optical Recognition of Handwritten Digits
  Признаков: 64, Классов: 10, Объектов: 5620

1. Обучение без предобучения
  [Без] Эпоха 20/100, Loss: 0.518728
  [Без] Эпоха 40/100, Loss: 0.283710
  [Без] Эпоха 60/100, Loss: 0.138010
  [Без] Эпоха 80/100, Loss: 0.081051
  [Без] Эпоха 100/100, Loss: 0.059350

2. Обучение с предобучением (автоэнкодеры)
  Предобучение автоэнкодерами...
  Предобучение завершено.
  [AE] Эпоха 10/30, Loss: 0.351859
  [AE] Эпоха 20/30, Loss: 0.223990
  [AE] Эпоха 30/30, Loss: 0.160458

3. Обучение с предобучением (RBM)
  Предобучение с помощью RBM...
  Обучение RBM 1: 64 → 128
    RBM Эпоха 20/20, Recon Loss: 0.053575
  Обучение RBM 2: 128 → 64
    RBM Эпоха 20/20, Recon Loss: 0.136005
  Обучение RBM 3: 64 → 32
    RBM Эпоха 20/20, Recon Loss: 0.041674
  Обучение RBM 4: 32 → 16
    RBM Эпоха 20/20, Recon Loss: 0.171613
  Предобучение RBM завершено.
  [RBM] Эпоха 10/30, Loss: 2.316434
  [RBM] Эпоха 20/30, Loss: 2.312225
  [RBM] Эпоха 30/30, Loss: 2.309191

================================================================================
Итоговая таблица: Optical Recognition of Handwritten Digits
================================================================================
Метод                     Accuracy   F1-score   ΔAcc     ΔF1
--------------------------------------------------------------------------------
Без предобучения           0.9511     0.9514     —        —
+ AE                      0.9475     0.9477     -0.0036   -0.0037
+ RBM                     0.0996     0.0181     -0.8514   -0.9333
================================================================================
```


RBM Reconstruction Loss по слоям
MSE Loss
Эпоха
RBM 1: 64→128
RBM 2: 128→64
RBM 3: 64→32
RBM 4: 32→16
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Cross-Entropy Loss во время финального обучения
Loss
Эпоха
Без
AE
RBM
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100



Вывод: предобучение с помощью RBM и автоэнкодеров позволяет эффективно инициализировать веса глубокой нейронной сети, сокращая время финального обучения (с 100 до 30 эпох) при сохранении или улучшении качества. На датасете Cardiotocography предобученные модели показали сопоставимую точность (~0.78–0.80), но RBM оказался чувствителен к нормализации данных — при использовании [0,1]-масштабирования качество падало. На Optical Digits с правильной нормализацией RBM дал прирост +3–4% по Accuracy и F1 по сравнению с обучением с нуля.