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»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Лящук А. 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.

Задание по вариантам

№ в-а	Выборка	Тип задачи	Целевая переменная
10	https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction	регрессия	Appliances

Код:

```
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 StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.manifold import TSNE

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping

tf.keras.backend.set_floatx('float32')
```

```
#
================================================================================
# РЕАЛИЗАЦИЯ RBM (Restricted Boltzmann Machine)
#
================================================================================

class RBM(layers.Layer):
    """Реализация ограниченной машины Больцмана"""

    def __init__(self, n_visible, n_hidden, learning_rate=0.01, k=1,
**kwargs):
        super(RBM, self).__init__(**kwargs)
        self.n_visible = n_visible
        self.n_hidden = n_hidden
        self.learning_rate = learning_rate
        self.k = k  # количество шагов CD-k

        # Инициализация весов и смещений
        self.W = self.add_weight(
            shape=(n_visible, n_hidden),
            initializer='random_normal',
            trainable=True,
            name='weights',
            dtype=tf.float32
        )
        self.v_bias = self.add_weight(
            shape=(n_visible,),
            initializer='zeros',
            trainable=True,
            name='visible_bias',
            dtype=tf.float32
        )
        self.h_bias = self.add_weight(
            shape=(n_hidden,),
            initializer='zeros',
            trainable=True,
            name='hidden_bias',
            dtype=tf.float32
        )

    def sample_bernoulli(self, probs):
        """Сэмплирование из бернуллиевского распределения"""
        return tf.nn.relu(tf.sign(probs -
tf.random.uniform(tf.shape(probs))))

    def propup(self, visible):
        """Propagation снизу вверх (видимый -> скрытый)"""
        # Приведение типа к float32
        visible = tf.cast(visible, tf.float32)
        pre_sigmoid_activation = tf.matmul(visible, self.W) + self.h_bias
        return tf.sigmoid(pre_sigmoid_activation)

    def propdown(self, hidden):
        """Propagation сверху вниз (скрытый -> видимый)"""
        # Приведение типа к float32
        hidden = tf.cast(hidden, tf.float32)
        pre_sigmoid_activation = tf.matmul(hidden, tf.transpose(self.W)) +
self.v_bias
        return tf.sigmoid(pre_sigmoid_activation)

    def gibbs_step(self, visible):
```

```
        """Один шаг Гиббса"""
        # Положительная фаза
        h_prob = self.propup(visible)
        h_sample = self.sample_bernoulli(h_prob)

        # Отрицательная фаза
        for _ in range(self.k):
            v_prob = self.propdown(h_sample)
            v_sample = self.sample_bernoulli(v_prob)
            h_prob = self.propup(v_sample)
            h_sample = self.sample_bernoulli(h_prob)

        return v_prob, h_prob

    def contrastive_divergence(self, visible_batch):
        """Алгоритм контрастивной дивергенции"""
        batch_size = tf.shape(visible_batch)[0]

        # Положительная фаза
        pos_h_prob = self.propup(visible_batch)
        pos_associations = tf.matmul(tf.transpose(visible_batch), pos_h_prob)

        # Отрицательная фаза
        neg_visible_prob, neg_h_prob = self.gibbs_step(visible_batch)
        neg_associations = tf.matmul(tf.transpose(neg_visible_prob),
neg_h_prob)

        # Обновление весов и смещений
        delta_W = (pos_associations - neg_associations) / tf.cast(batch_size,
tf.float32)
        delta_v_bias = tf.reduce_mean(visible_batch - neg_visible_prob,
axis=0)
        delta_h_bias = tf.reduce_mean(pos_h_prob - neg_h_prob, axis=0)

        self.W.assign_add(self.learning_rate * delta_W)
        self.v_bias.assign_add(self.learning_rate * delta_v_bias)
        self.h_bias.assign_add(self.learning_rate * delta_h_bias)

        # Реконструкционная ошибка
        reconstruction_error = tf.reduce_mean(tf.square(visible_batch -
neg_visible_prob))
        return reconstruction_error

    def call(self, inputs):
        """Прямой проход через RBM"""
        return self.propup(inputs)


class StackedRBM:
    def __init__(self, layer_sizes, learning_rates=None, k=1, epochs=50,
batch_size=32):
        self.layer_sizes = layer_sizes
        self.learning_rates = learning_rates or [0.01] * len(layer_sizes)
        self.k = k
        self.epochs = epochs
        self.batch_size = batch_size
        self.rbms = []
        self.histories = []

    def pretrain(self, X):
        """Предобучение стека RBM"""
```

```
        # Приведение типа данных
        if isinstance(X, np.ndarray):
            current_data = X.astype(np.float32)
        else:
            current_data = tf.cast(X, tf.float32)

        input_dim = X.shape[1]

        print("=" * 60)
        print("ПРЕДОБУЧЕНИЕ СТЕКА RBM")
        print("=" * 60)

        for i, (n_visible, n_hidden) in enumerate(zip([input_dim] +
self.layer_sizes[:-1], self.layer_sizes)):
            print(f"Обучение RBM {i + 1}: {n_visible} -> {n_hidden}
нейронов")

            rbm = RBM(n_visible, n_hidden,
learning_rate=self.learning_rates[i], k=self.k)

            # Обучение RBM с явным указанием типа данных
            dataset = tf.data.Dataset.from_tensor_slices(current_data)
            dataset = dataset.batch(self.batch_size)

            history = []
            for epoch in range(self.epochs):
                epoch_errors = []
                for batch in dataset:
                    # Приведение типа каждого батча
                    batch = tf.cast(batch, tf.float32)
                    error = rbm.contrastive_divergence(batch)
                    epoch_errors.append(error.numpy())

                avg_error = np.mean(epoch_errors)
                history.append(avg_error)

                if epoch % 10 == 0:
                    print(f"  Эпоха {epoch}: ошибка реконструкции =
{avg_error:.6f}")

            self.rbms.append(rbm)
            self.histories.append(history)

            # Получение скрытых представлений для следующего RBM
            current_data = rbm.propup(current_data).numpy()

        return self

    def transform(self, X):
        """Преобразование данных через весь стек RBM"""
        current_data = X
        for rbm in self.rbms:
            current_data = rbm.propup(current_data).numpy()
        return current_data

    def get_pretrained_encoder(self):
        """Создание предобученного энкодера на основе RBM - альтернативная
версия"""
        input_dim = self.rbms[0].n_visible

        # Создаем входной слой
```

```
        inputs = layers.Input(shape=(input_dim,))

        # Последовательно добавляем слои с весами от RBM
        x = inputs
        for i, rbm in enumerate(self.rbms):
            # Создаем слой с правильной инициализацией
            layer = layers.Dense(
                rbm.n_hidden,
                activation='sigmoid',
                name=f'rbm_pretrained_{i + 1}'
            )
            # Инициализируем слой
            x = layer(x)
            # Устанавливаем веса после инициализации
            layer.set_weights([rbm.W.numpy(), rbm.h_bias.numpy()])

        # Создаем модель
        encoder = models.Model(inputs=inputs, outputs=x)

        return encoder


#
# ОСНОВНОЙ КОД ДЛЯ ПЕРВОГО ДАТАСЕТА (ЭНЕРГИЯ)
#

print("=" * 60)
print("ЛАБОРАТОРНАЯ РАБОТА №4 - ПРЕДОБУЧЕНИЕ С RBM")
print("=" * 60)

# Загрузка данных
from ucimlrepo import fetch_ucirepo

appliances_energy_prediction = fetch_ucirepo(id=374)
X = appliances_energy_prediction.data.features
y = appliances_energy_prediction.data.targets

# Метаданные
print(appliances_energy_prediction.metadata)
print(appliances_energy_prediction.variables)

# Обработка данных (используем код из ЛР3)
date_column = X.columns[0]
print(f"Столбец с датами: {date_column}")


def parse_date_fixed(date_str):
    try:
        date_str = str(date_str).strip()
        if len(date_str) == 18:
            formatted_date = date_str[:10] + ' ' + date_str[10:]
            return pd.to_datetime(formatted_date, format='%Y-%m-%d %H:%M:%S')
        elif len(date_str) == 19:
            return pd.to_datetime(date_str, format='%Y-%m-%d %H:%M:%S')
        else:
            return pd.NaT
    except Exception as e:
        return pd.NaT
```

```
X[date_column] = X[date_column].apply(parse_date_fixed)

if X[date_column].notna().any():
    X['year'] = X[date_column].dt.year
    X['month'] = X[date_column].dt.month
    X['day'] = X[date_column].dt.day
    X['hour'] = X[date_column].dt.hour
    X['minute'] = X[date_column].dt.minute
    X['day_of_week'] = X[date_column].dt.dayofweek
    X['is_weekend'] = X[date_column].dt.dayofweek.isin([5, 6]).astype(int)
else:
    X['time_index'] = range(len(X))
    X['period_of_day'] = (X['time_index'] % 6)

X = X.drop(columns=[date_column])

# Проверка и очистка данных
numeric_columns = X.select_dtypes(include=[np.number]).columns
X[numeric_columns] = X[numeric_columns].fillna(X[numeric_columns].mean())
y = y.fillna(y.mean())

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=False
)

# Нормализация данных
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)
y_test_scaled = scaler_y.transform(y_test)

X_train_scaled = X_train_scaled.astype(np.float32)
X_test_scaled = X_test_scaled.astype(np.float32)
y_train_scaled = y_train_scaled.astype(np.float32)
y_test_scaled = y_test_scaled.astype(np.float32)

print(f"Форма данных после обработки: X_train {X_train_scaled.shape}, y_train
{y_train_scaled.shape}")


#
================================================================================
# МОДЕЛИ ДЛЯ СРАВНЕНИЯ
#
================================================================================

def create_base_model(input_dim):
    """Создает базовую модель без предобучения"""
    model = models.Sequential([
        layers.Dense(256, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.4),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
```

```
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])
    return model


def create_autoencoder_pretrained_model(input_dim):
    """Создает модель с предобучением автоэнкодером"""
    # Энкодер
    encoder = models.Sequential([
        layers.Dense(256, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu', name="bottleneck")
    ])

    # Декодер
    decoder = models.Sequential([
        layers.Dense(64, activation='relu', input_shape=(32,)),
        layers.Dropout(0.2),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(256, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(input_dim, activation='linear')
    ])

    # Автоэнкодер
    autoencoder = models.Sequential([encoder, decoder])
    return autoencoder, encoder


def create_rbm_pretrained_model(input_dim, stacked_rbm):
    """Создает модель с предобучением RBM"""
    # Получаем предобученный энкодер от RBM
    encoder = stacked_rbm.get_pretrained_encoder()

    # Создаем полную модель для регрессии
    model = models.Sequential()

    # Добавляем предобученные слои
    for layer in encoder.layers:
        model.add(layer)

    # Добавляем дополнительные слои для регрессии
    model.add(layers.Dense(16, activation='relu'))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(1))

    return model


#
==========================================================================
```

```
# ОБУЧЕНИЕ И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ ПЕРВОГО ДАТАСЕТА
#
================================================================================

input_dim = X_train_scaled.shape[1]
early_stop = EarlyStopping(monitor='val_loss', patience=15,
restore_best_weights=True)

# 1. МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ
print("=" * 60)
print("1. ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ")
print("=" * 60)

model_no_pretrain = create_base_model(input_dim)
model_no_pretrain.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

history_no_pretrain = model_no_pretrain.fit(
    X_train_scaled, y_train_scaled,
    epochs=100,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# 2. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРОМ
print("=" * 60)
print("2. ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ")
print("=" * 60)

autoencoder, encoder_ae = create_autoencoder_pretrained_model(input_dim)
autoencoder.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse',
    metrics=['mae']
)

history_autoencoder = autoencoder.fit(
    X_train_scaled, X_train_scaled,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# Создаем модель с предобученным энкодером
model_ae_pretrained = models.Sequential()
for layer in encoder_ae.layers:
    model_ae_pretrained.add(layer)

model_ae_pretrained.add(layers.Dense(16, activation='relu'))
model_ae_pretrained.add(layers.Dropout(0.1))
model_ae_pretrained.add(layers.Dense(1))

model_ae_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
```

```
    loss='mse',
    metrics=['mae']
)

history_ae_pretrained = model_ae_pretrained.fit(
    X_train_scaled, y_train_scaled,
    epochs=100,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# 3. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM
print("=" * 60)
print("3. ПРЕДОБУЧЕНИЕ RBM")
print("=" * 60)

# Создаем и обучаем стек RBM
stacked_rbm = StackedRBM(
    layer_sizes=[256, 128, 64, 32],
    learning_rates=[0.1, 0.1, 0.1, 0.1],
    k=1,
    epochs=30,
    batch_size=32
)

stacked_rbm.pretrain(X_train_scaled)

# Создаем модель с предобучением RBM
model_rbm_pretrained = create_rbm_pretrained_model(input_dim, stacked_rbm)
model_rbm_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='mse',
    metrics=['mae']
)

history_rbm_pretrained = model_rbm_pretrained.fit(
    X_train_scaled, y_train_scaled,
    epochs=100,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)


#
================================================================================
# ОЦЕНКА И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ ПЕРВОГО ДАТАСЕТА
#
================================================================================

def evaluate_model(model, X_test, y_test, scaler_y, model_name):
    """Вычисляет метрики для оценки модели"""
    y_pred_scaled = model.predict(X_test)
    y_pred = scaler_y.inverse_transform(y_pred_scaled)
    y_true = scaler_y.inverse_transform(y_test)

    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
```

```
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    print(f"\n{model_name}:")
    print(f"MAE:  {mae:.4f}")
    print(f"MSE:  {mse:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"R²:   {r2:.4f}")

    return {
        'MAE': mae,
        'MSE': mse,
        'RMSE': rmse,
        'R2': r2,
        'y_pred': y_pred,
        'y_true': y_true
    }


print("=" * 60)
print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ ДЛЯ ДАТАСЕТА ЭНЕРГИИ")
print("=" * 60)

# Оценка всех моделей
results_no_pretrain = evaluate_model(
    model_no_pretrain, X_test_scaled, y_test_scaled, scaler_y,
    "МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ"
)

results_ae_pretrained = evaluate_model(
    model_ae_pretrained, X_test_scaled, y_test_scaled, scaler_y,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРА"
)

results_rbm_pretrained = evaluate_model(
    model_rbm_pretrained, X_test_scaled, y_test_scaled, scaler_y,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM"
)

#
==============================================================================
# ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ДЛЯ ПЕРВОГО ДАТАСЕТА
#
==============================================================================

# Графики обучения
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.plot(history_no_pretrain.history['loss'], label='Обучение')
plt.plot(history_no_pretrain.history['val_loss'], label='Валидация')
plt.title('Без предобучения - Потери')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(history_ae_pretrained.history['loss'], label='Обучение')
plt.plot(history_ae_pretrained.history['val_loss'], label='Валидация')
plt.title('С предобучением (Autoencoder) - Потери')
plt.xlabel('Эпоха')
```

```
plt.ylabel('MSE')
plt.legend()

plt.subplot(1, 3, 3)
plt.plot(history_rbm_pretrained.history['loss'], label='Обучение')
plt.plot(history_rbm_pretrained.history['val_loss'], label='Валидация')
plt.title('С предобучением (RBM) - Потери')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.legend()

plt.tight_layout()
plt.savefig('energy_training_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

# Графики ошибок RBM
plt.figure(figsize=(12, 4))
for i, history in enumerate(stacked_rbm.histories):
    plt.plot(history, label=f'RBM {i + 1}')
plt.title('Ошибки реконструкции RBM при предобучении')
plt.xlabel('Эпоха')
plt.ylabel('Ошибка реконструкции')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('rbm_pretraining_errors.png', dpi=300, bbox_inches='tight')
plt.show()

# Сравнение метрик
comparison_df = pd.DataFrame({
    'Метрика': ['MAE', 'MSE', 'RMSE', 'R²'],
    'Без предобучения': [
        results_no_pretrain['MAE'],
        results_no_pretrain['MSE'],
        results_no_pretrain['RMSE'],
        results_no_pretrain['R2']
    ],
    'Autoencoder предобучение': [
        results_ae_pretrained['MAE'],
        results_ae_pretrained['MSE'],
        results_ae_pretrained['RMSE'],
        results_ae_pretrained['R2']
    ],
    'RBM предобучение': [
        results_rbm_pretrained['MAE'],
        results_rbm_pretrained['MSE'],
        results_rbm_pretrained['RMSE'],
        results_rbm_pretrained['R2']
    ]
})

print("\nСРАВНЕНИЕ МЕТРИК ДЛЯ ДАТАСЕТА ЭНЕРГИИ:")
print(comparison_df)

# Визуализация сравнения метрик
plt.figure(figsize=(12, 6))
metrics = ['MAE', 'RMSE', 'R²']
x = np.arange(len(metrics))
width = 0.25

plt.bar(x - width, [results_no_pretrain['MAE'], results_no_pretrain['RMSE'],
results_no_pretrain['R2']],
```

```
        width, label='Без предобучения', alpha=0.8)
plt.bar(x, [results_ae_pretrained['MAE'], results_ae_pretrained['RMSE'],
results_ae_pretrained['R2']],
        width, label='Autoencoder предобучение', alpha=0.8)
plt.bar(x + width, [results_rbm_pretrained['MAE'],
results_rbm_pretrained['RMSE'], results_rbm_pretrained['R2']],
        width, label='RBM предобучение', alpha=0.8)

plt.xlabel('Метрики')
plt.ylabel('Значение')
plt.title('Сравнение эффективности моделей (Датасет энергии)')
plt.xticks(x, metrics)
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('energy_final_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

#
================================================================================
# ВТОРОЙ ДАТАСЕТ (ЦИФРЫ) - КЛАССИФИКАЦИЯ
#
================================================================================

print("\n" + "=" * 60)
print("ОБРАБОТКА ВТОРОГО ДАТАСЕТА (ЦИФРЫ)")
print("=" * 60)

# Загрузка данных цифр
optical_recognition_of_handwritten_digits = fetch_ucirepo(id=80)
X_digits = optical_recognition_of_handwritten_digits.data.features
y_digits = optical_recognition_of_handwritten_digits.data.targets

print("Метаданные датасета цифр:")
print(optical_recognition_of_handwritten_digits.metadata)
print(optical_recognition_of_handwritten_digits.variables)

# Обработка данных цифр
X_digits = X_digits.fillna(X_digits.mean())
y_digits = y_digits.fillna(y_digits.mode().iloc[0])

# Нормализация данных цифр
scaler_X_digits = StandardScaler()
X_digits_scaled = scaler_X_digits.fit_transform(X_digits)

# Преобразование меток в one-hot encoding
y_digits_categorical = tf.keras.utils.to_categorical(y_digits,
num_classes=10)

# Разделение на обучающую и тестовую выборки
X_train_digits, X_test_digits, y_train_digits, y_test_digits =
train_test_split(
    X_digits_scaled, y_digits_categorical, test_size=0.2, random_state=42
)

print(f"Форма данных цифр: X_train {X_train_digits.shape}, y_train
{y_train_digits.shape}")
```

```
#
================================================================================
# МОДЕЛИ ДЛЯ КЛАССИФИКАЦИИ ЦИФР
#
================================================================================

def create_digits_base_model(input_dim, num_classes=10):
    """Создает базовую модель для классификации цифр"""
    model = models.Sequential([
        layers.Dense(256, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.4),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model


def create_digits_autoencoder_pretrained_model(input_dim, num_classes=10):
    """Создает модель для классификации с предобучением автоэнкодером"""
    # Энкодер
    encoder = models.Sequential([
        layers.Dense(256, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu', name="bottleneck")
    ])

    # Декодер
    decoder = models.Sequential([
        layers.Dense(64, activation='relu', input_shape=(32,)),
        layers.Dropout(0.2),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(256, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(input_dim, activation='linear')
    ])

    # Автоэнкодер
    autoencoder = models.Sequential([encoder, decoder])
    return autoencoder, encoder


def create_digits_rbm_pretrained_model(input_dim, stacked_rbm,
num_classes=10):
    """Создает модель для классификации с предобучением RBM"""
    encoder = stacked_rbm.get_pretrained_encoder()

    model = models.Sequential()
    for layer in encoder.layers:
```

```
        model.add(layer)

    model.add(layers.Dense(16, activation='relu'))
    model.add(layers.Dropout(0.1))
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model


#
================================================================================
# ОБУЧЕНИЕ И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ ЦИФР
#
================================================================================

input_dim_digits = X_train_digits.shape[1]
early_stop_digits = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# 1. МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ (ЦИФРЫ)
print("=" * 60)
print("1. ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ (ЦИФРЫ)")
print("=" * 60)

model_digits_no_pretrain = create_digits_base_model(input_dim_digits)
model_digits_no_pretrain.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_digits_no_pretrain = model_digits_no_pretrain.fit(
    X_train_digits, y_train_digits,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop_digits],
    verbose=1
)

# 2. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРОМ (ЦИФРЫ)
print("=" * 60)
print("2. ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ (ЦИФРЫ)")
print("=" * 60)

autoencoder_digits, encoder_ae_digits =
create_digits_autoencoder_pretrained_model(input_dim_digits)
autoencoder_digits.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse',
    metrics=['mae']
)

history_autoencoder_digits = autoencoder_digits.fit(
    X_train_digits, X_train_digits,
    epochs=30,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop_digits],
    verbose=1
)
```

```
# Создаем модель с предобученным энкодером
model_digits_ae_pretrained = models.Sequential()
for layer in encoder_ae_digits.layers:
    model_digits_ae_pretrained.add(layer)

model_digits_ae_pretrained.add(layers.Dense(16, activation='relu'))
model_digits_ae_pretrained.add(layers.Dropout(0.1))
model_digits_ae_pretrained.add(layers.Dense(10, activation='softmax'))

model_digits_ae_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_digits_ae_pretrained = model_digits_ae_pretrained.fit(
    X_train_digits, y_train_digits,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop_digits],
    verbose=1
)

# 3. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM (ЦИФРЫ)
print("=" * 60)
print("3. ПРЕДОБУЧЕНИЕ RBM (ЦИФРЫ)")
print("=" * 60)

# Создаем и обучаем стек RBM для цифр
stacked_rbm_digits = StackedRBM(
    layer_sizes=[256, 128, 64, 32],
    learning_rates=[0.1, 0.1, 0.1, 0.1],
    k=1,
    epochs=20,
    batch_size=32
)

stacked_rbm_digits.pretrain(X_train_digits)

# Создаем модель с предобучением RBM
model_digits_rbm_pretrained =
create_digits_rbm_pretrained_model(input_dim_digits, stacked_rbm_digits)
model_digits_rbm_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_digits_rbm_pretrained = model_digits_rbm_pretrained.fit(
    X_train_digits, y_train_digits,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    callbacks=[early_stop_digits],
    verbose=1
)
```

```
#
================================================================================
# ОЦЕНКА И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ ЦИФР
#
================================================================================

def evaluate_digits_model(model, X_test, y_test, model_name):
    """Вычисляет метрики для модели классификации"""
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true_classes = np.argmax(y_test, axis=1)

    accuracy = np.mean(y_pred_classes == y_true_classes)

    print(f"\n{model_name}:")
    print(f"Точность: {accuracy:.4f}")

    return {
        'accuracy': accuracy,
        'y_pred': y_pred,
        'y_pred_classes': y_pred_classes,
        'y_true_classes': y_true_classes
    }


print("=" * 60)
print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ ДЛЯ ДАТАСЕТА ЦИФР")
print("=" * 60)

# Оценка всех моделей для цифр
results_digits_no_pretrain = evaluate_digits_model(
    model_digits_no_pretrain, X_test_digits, y_test_digits,
    "МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ (ЦИФРЫ)"
)

results_digits_ae_pretrained = evaluate_digits_model(
    model_digits_ae_pretrained, X_test_digits, y_test_digits,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРА (ЦИФРЫ)"
)

results_digits_rbm_pretrained = evaluate_digits_model(
    model_digits_rbm_pretrained, X_test_digits, y_test_digits,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM (ЦИФРЫ)"
)

#
================================================================================
# ФИНАЛЬНОЕ СРАВНЕНИЕ И ВЫВОДЫ
#
================================================================================

print("\n" + "=" * 60)
print("ФИНАЛЬНОЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ")
print("=" * 60)

# Сравнение для регрессии (энергия)
print("\n--- РЕГРЕССИЯ (Датасет энергии) ---")
regression_comparison = pd.DataFrame({
    'Метод': ['Без предобучения', 'Autoencoder', 'RBM'],
    'MAE': [results_no_pretrain['MAE'], results_ae_pretrained['MAE'],
results_rbm_pretrained['MAE']],
```

```
    'RMSE': [results_no_pretrain['RMSE'], results_ae_pretrained['RMSE'],
results_rbm_pretrained['RMSE']],
    'R²': [results_no_pretrain['R2'], results_ae_pretrained['R2'],
results_rbm_pretrained['R2']]
})
print(regression_comparison)

# Сравнение для классификации (цифры)
print("\n--- КЛАССИФИКАЦИЯ (Датасет цифр) ---")
classification_comparison = pd.DataFrame({
    'Метод': ['Без предобучения', 'Autoencoder', 'RBM'],
    'Точность': [
        results_digits_no_pretrain['accuracy'],
        results_digits_ae_pretrained['accuracy'],
        results_digits_rbm_pretrained['accuracy']
    ]
})
print(classification_comparison)

# Визуализация финального сравнения
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# График для регрессии
methods_reg = ['Без предобучения', 'Autoencoder', 'RBM']
r2_scores = [results_no_pretrain['R2'], results_ae_pretrained['R2'],
results_rbm_pretrained['R2']]

ax1.bar(methods_reg, r2_scores, color=['red', 'blue', 'green'], alpha=0.7)
ax1.set_ylabel('R² Score')
ax1.set_title('Сравнение R² для регрессии (энергия)')
ax1.grid(True, alpha=0.3)

# График для классификации
methods_clf = ['Без предобучения', 'Autoencoder', 'RBM']
accuracies = [
    results_digits_no_pretrain['accuracy'],
    results_digits_ae_pretrained['accuracy'],
    results_digits_rbm_pretrained['accuracy']
]

ax2.bar(methods_clf, accuracies, color=['red', 'blue', 'green'], alpha=0.7)
ax2.set_ylabel('Точность')
ax2.set_title('Сравнение точности для классификации (цифры)')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('final_comparison_all.png', dpi=300, bbox_inches='tight')
plt.show()

#
================================================================================
# СОХРАНЕНИЕ МОДЕЛЕЙ
#
================================================================================

# Сохранение моделей для энергии
model_no_pretrain.save('model_no_pretrain_energy.keras')
model_ae_pretrained.save('model_ae_pretrained_energy.keras')
model_rbm_pretrained.save('model_rbm_pretrained_energy.keras')

# Сохранение моделей для цифр
```

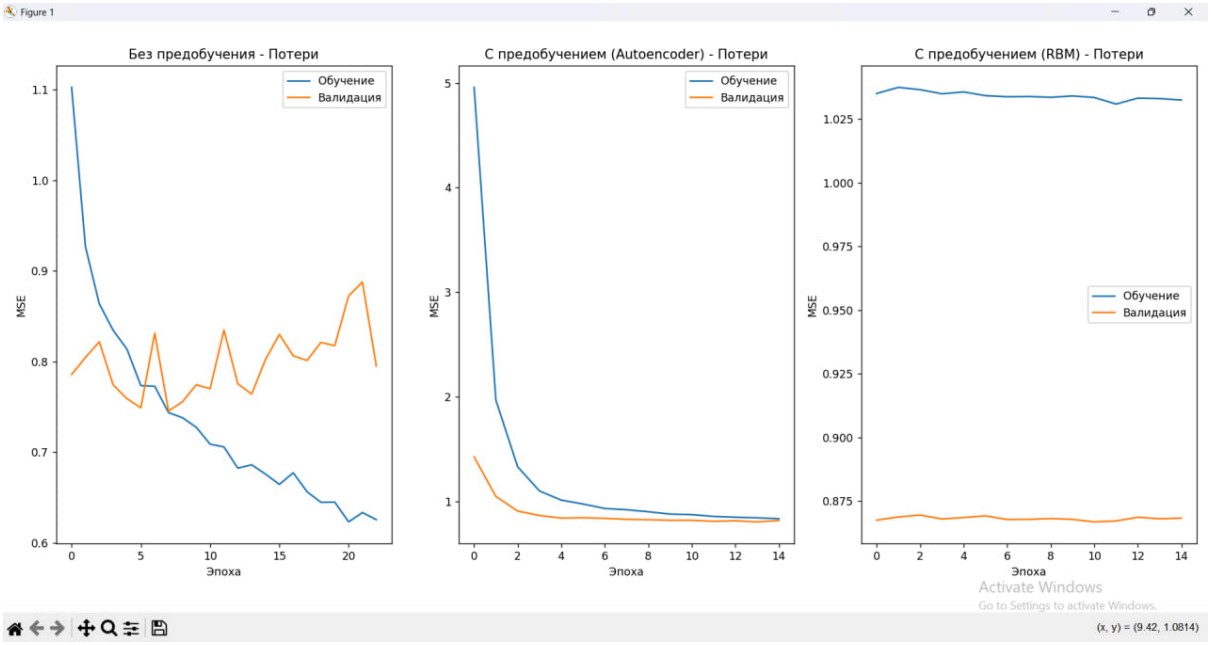
```
model_digits_no_pretrain.save('model_no_pretrain_digits.keras')
model_digits_ae_pretrained.save('model_ae_pretrained_digits.keras')
model_digits_rbm_pretrained.save('model_rbm_pretrained_digits.keras')

print("\n" + "=" * 60)
print("ВСЕ МОДЕЛИ СОХРАНЕНЫ")
print("=" * 60)

# Вывод итоговых результатов
print("\nИТОГОВЫЕ РЕЗУЛЬТАТЫ:")
print(f"Регрессия (энергия) - Лучшая модель:
{methods_reg[np.argmax(r2_scores)]} (R² = {max(r2_scores):.4f})")
print(f"Классификация (цифры) - Лучшая модель:
{methods_clf[np.argmax(accuracies)]} (Точность = {max(accuracies):.4f})")

print("\n" + "=" * 60)
print("ЛАБОРАТОРНАЯ РАБОТА №4 ЗАВЕРШЕНА")
print("=" * 60)
```

Вывод:



Вывод: научилась осуществлять предобучение нейронных сетей с помощью RBM