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

Максимович А. И.

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
11	https://archive.ics.uci.edu/dataset/27/credit+approval	классификация	+/-

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, LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score, accuracy_score, classification_report, \
    confusion_matrix
from sklearn.manifold import TSNE
from sklearn.impute import SimpleImputer

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

# Установка единого типа данных
```

```
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

        # 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)"""
        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)"""
        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"""
        current_data = X.astype(np.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
            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:
                    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"""
        input_dim = self.rbms[0].n_visible

        # Создаем модель энкодера
        encoder = models.Sequential()
        encoder.add(layers.InputLayer(input_shape=(input_dim,)))

        # Добавляем слои из RBM
        for i, rbm in enumerate(self.rbms):
            # Создаем Dense слой
```

```
            dense_layer = layers.Dense(
                rbm.n_hidden,
                activation='sigmoid',
                name=f'rbm_pretrained_{i + 1}'
            )
            encoder.add(dense_layer)

        # Сначала нужно построить модель
        dummy_input = tf.ones((1, input_dim))
        _ = encoder(dummy_input)

        # Устанавливаем веса из RBM
        for i, rbm in enumerate(self.rbms):
            weights = [rbm.W.numpy(), rbm.h_bias.numpy()]
            encoder.layers[i].set_weights(weights)

        return encoder


#
================================================================================
# ФУНКЦИИ ДЛЯ ПРЕДОБРАБОТКИ ДАННЫХ
#
================================================================================

def preprocess_credit_data(X, y):
    """Предобработка данных кредитного скоринга"""
    # Копируем данные
    X_processed = X.copy()
    y_processed = y.copy()

    # Определяем числовые и категориальные колонки
    numeric_columns = X_processed.select_dtypes(include=[np.number]).columns
    categorical_columns =
X_processed.select_dtypes(include=['object']).columns

    print(f"Числовые колонки: {len(numeric_columns)}")
    print(f"Категориальные колонки: {len(categorical_columns)}")

    # Заполняем пропуски в числовых колонках
    if len(numeric_columns) > 0:
        numeric_imputer = SimpleImputer(strategy='median')
        X_processed[numeric_columns] =
numeric_imputer.fit_transform(X_processed[numeric_columns])

    # Обрабатываем категориальные колонки
    if len(categorical_columns) > 0:
        categorical_imputer = SimpleImputer(strategy='most_frequent')
        X_processed[categorical_columns] =
categorical_imputer.fit_transform(X_processed[categorical_columns])

        # One-hot encoding для категориальных признаков
        X_processed = pd.get_dummies(X_processed,
columns=categorical_columns, drop_first=True)

    # Кодируем целевую переменную
    label_encoder = LabelEncoder()
    y_processed = label_encoder.fit_transform(y_processed.iloc[:, 0] if
hasattr(y_processed, 'iloc') else y_processed)
```

```
    print(f"Форма данных после обработки: X {X_processed.shape}, y
{y_processed.shape}")
    return X_processed, y_processed


def preprocess_cancer_data(X, y):
    """Предобработка данных рака груди"""
    # Копируем данные
    X_processed = X.copy()
    y_processed = y.copy()

    # Заполняем пропуски
    numeric_imputer = SimpleImputer(strategy='median')
    X_processed = pd.DataFrame(numeric_imputer.fit_transform(X_processed),
                               columns=X_processed.columns)

    # Кодируем целевую переменную (M/B -> 1/0)
    label_encoder = LabelEncoder()
    y_processed = label_encoder.fit_transform(y_processed.iloc[:, 0] if
hasattr(y_processed, 'iloc') else y_processed)

    print(f"Форма данных после обработки: X {X_processed.shape}, y
{y_processed.shape}")
    return X_processed, y_processed


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

def create_base_model(input_dim, output_dim=1,
problem_type='classification'):
    """Создает базовую модель без предобучения"""
    model = models.Sequential([
        layers.Dense(128, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.1)
    ])

    # Выходной слой в зависимости от типа задачи
    if problem_type == 'classification':
        if output_dim == 1:
            model.add(layers.Dense(1, activation='sigmoid'))
        else:
            model.add(layers.Dense(output_dim, activation='softmax'))
    else:  # regression
        model.add(layers.Dense(1))

    return model


def create_autoencoder_pretrained_model(input_dim, encoding_dim=32):
    """Создает модель с предобучением автоэнкодером"""
    # Энкодер
```

```
    encoder = models.Sequential([
        layers.Dense(128, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.BatchNormalization(),
        layers.Dense(encoding_dim, activation='relu', name="bottleneck")
    ])

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

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


def create_rbm_pretrained_model(input_dim, stacked_rbm, output_dim=1,
problem_type='classification'):
    """Создает модель с предобучением RBM"""
    # Получаем предобученный энкодер от RBM
    encoder = stacked_rbm.get_pretrained_encoder()

    # Создаем полную модель
    model = models.Sequential()

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

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

    # Выходной слой
    if problem_type == 'classification':
        if output_dim == 1:
            model.add(layers.Dense(1, activation='sigmoid'))
        else:
            model.add(layers.Dense(output_dim, activation='softmax'))
    else:  # regression
        model.add(layers.Dense(1))

    return model


#
================================================================================
# ФУНКЦИИ ДЛЯ ОЦЕНКИ МОДЕЛЕЙ
#
================================================================================

def evaluate_classification_model(model, X_test, y_test, model_name):
    """Вычисляет метрики для модели классификации"""
```

```
    y_pred = model.predict(X_test)

    if y_pred.shape[1] == 1:  # бинарная классификация
        y_pred_classes = (y_pred > 0.5).astype(int).flatten()
    else:  # многоклассовая классификация
        y_pred_classes = np.argmax(y_pred, axis=1)

    accuracy = accuracy_score(y_test, y_pred_classes)

    print(f"\n{model_name}:")
    print(f"Точность: {accuracy:.4f}")
    print("\nОтчет по классификации:")
    print(classification_report(y_test, y_pred_classes))

    # Матрица ошибок
    cm = confusion_matrix(y_test, y_pred_classes)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Матрица ошибок: {model_name}')
    plt.ylabel('Истинные значения')
    plt.xlabel('Предсказанные значения')
    plt.tight_layout()
    plt.show()

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


#
================================================================================
# ОСНОВНОЙ КОД ДЛЯ ПЕРВОГО ДАТАСЕТА (CREDIT APPROVAL)
#
================================================================================

print("=" * 70)
print("ЛАБОРАТОРНАЯ РАБОТА №4 - ПРЕДОБУЧЕНИЕ С RBM")
print("ДАТАСЕТ: CREDIT APPROVAL")
print("=" * 70)

# Загрузка данных кредитного скоринга
from ucimlrepo import fetch_ucirepo

credit_approval = fetch_ucirepo(id=27)
X_credit = credit_approval.data.features
y_credit = credit_approval.data.targets

# Метаданные
print("Метаданные датасета Credit Approval:")
print(credit_approval.metadata)
print("\nИнформация о переменных:")
print(credit_approval.variables)

# Предобработка данных
X_credit_processed, y_credit_processed = preprocess_credit_data(X_credit,
y_credit)

# Разделение на обучающую и тестовую выборки
```

```
X_train_credit, X_test_credit, y_train_credit, y_test_credit =
train_test_split(
    X_credit_processed, y_credit_processed, test_size=0.2, random_state=42,
stratify=y_credit_processed
)

# Нормализация данных
scaler_credit = StandardScaler()
X_train_credit_scaled = scaler_credit.fit_transform(X_train_credit)
X_test_credit_scaled = scaler_credit.transform(X_test_credit)

X_train_credit_scaled = X_train_credit_scaled.astype(np.float32)
X_test_credit_scaled = X_test_credit_scaled.astype(np.float32)

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

#
================================================================================
# ОБУЧЕНИЕ И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ CREDIT APPROVAL
#
================================================================================

input_dim_credit = X_train_credit_scaled.shape[1]
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# 1. МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ
print("\n" + "=" * 60)
print("1. ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ (CREDIT APPROVAL)")
print("=" * 60)

model_credit_no_pretrain = create_base_model(input_dim_credit, output_dim=1,
problem_type='classification')
model_credit_no_pretrain.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history_credit_no_pretrain = model_credit_no_pretrain.fit(
    X_train_credit_scaled, y_train_credit,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# 2. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРОМ
print("\n" + "=" * 60)
print("2. ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ (CREDIT APPROVAL)")
print("=" * 60)

autoencoder_credit, encoder_ae_credit =
create_autoencoder_pretrained_model(input_dim_credit)
autoencoder_credit.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse',
    metrics=['mae']
)
```

```
history_autoencoder_credit = autoencoder_credit.fit(
    X_train_credit_scaled, X_train_credit_scaled,
    epochs=30,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

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

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

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

history_credit_ae_pretrained = model_credit_ae_pretrained.fit(
    X_train_credit_scaled, y_train_credit,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# 3. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM
print("\n" + "=" * 60)
print("3. ПРЕДОБУЧЕНИЕ RBM (CREDIT APPROVAL)")
print("=" * 60)

# Создаем и обучаем стек RBM
stacked_rbm_credit = StackedRBM(
    layer_sizes=[64, 32, 16],
    learning_rates=[0.1, 0.1, 0.1],
    k=1,
    epochs=20,
    batch_size=32
)

stacked_rbm_credit.pretrain(X_train_credit_scaled)

# Создаем модель с предобучением RBM
model_credit_rbm_pretrained = create_rbm_pretrained_model(
    input_dim_credit, stacked_rbm_credit, output_dim=1,
problem_type='classification'
)
model_credit_rbm_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
history_credit_rbm_pretrained = model_credit_rbm_pretrained.fit(
    X_train_credit_scaled, y_train_credit,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

#
================================================================================
# ОЦЕНКА И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ CREDIT APPROVAL
#
================================================================================

print("\n" + "=" * 70)
print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ ДЛЯ ДАТАСЕТА CREDIT APPROVAL")
print("=" * 70)

# Оценка всех моделей
results_credit_no_pretrain = evaluate_classification_model(
    model_credit_no_pretrain, X_test_credit_scaled, y_test_credit,
    "МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ (CREDIT APPROVAL)"
)

results_credit_ae_pretrained = evaluate_classification_model(
    model_credit_ae_pretrained, X_test_credit_scaled, y_test_credit,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРА (CREDIT APPROVAL)"
)

results_credit_rbm_pretrained = evaluate_classification_model(
    model_credit_rbm_pretrained, X_test_credit_scaled, y_test_credit,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM (CREDIT APPROVAL)"
)

#
================================================================================
# ВТОРОЙ ДАТАСЕТ (BREAST CANCER) - ПУНКТ 3
#
================================================================================

print("\n" + "=" * 70)
print("ОБРАБОТКА ВТОРОГО ДАТАСЕТА (BREAST CANCER WISCONSIN)")
print("=" * 70)

# Загрузка данных рака груди
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
X_cancer = breast_cancer_wisconsin_diagnostic.data.features
y_cancer = breast_cancer_wisconsin_diagnostic.data.targets

print("Метаданные датасета Breast Cancer:")
print(breast_cancer_wisconsin_diagnostic.metadata)
print("\nИнформация о переменных:")
print(breast_cancer_wisconsin_diagnostic.variables)

# Предобработка данных
X_cancer_processed, y_cancer_processed = preprocess_cancer_data(X_cancer,
y_cancer)

# Разделение на обучающую и тестовую выборки
```

```
X_train_cancer, X_test_cancer, y_train_cancer, y_test_cancer =
train_test_split(
    X_cancer_processed, y_cancer_processed, test_size=0.2, random_state=42,
stratify=y_cancer_processed
)

# Нормализация данных
scaler_cancer = StandardScaler()
X_train_cancer_scaled = scaler_cancer.fit_transform(X_train_cancer)
X_test_cancer_scaled = scaler_cancer.transform(X_test_cancer)

X_train_cancer_scaled = X_train_cancer_scaled.astype(np.float32)
X_test_cancer_scaled = X_test_cancer_scaled.astype(np.float32)

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

#
================================================================================
# ОБУЧЕНИЕ И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ BREAST CANCER
#
================================================================================

input_dim_cancer = X_train_cancer_scaled.shape[1]
early_stop_cancer = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# 1. МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ (CANCER)
print("\n" + "=" * 60)
print("1. ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ (BREAST CANCER)")
print("=" * 60)

model_cancer_no_pretrain = create_base_model(input_dim_cancer, output_dim=1,
problem_type='classification')
model_cancer_no_pretrain.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history_cancer_no_pretrain = model_cancer_no_pretrain.fit(
    X_train_cancer_scaled, y_train_cancer,
    epochs=50,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop_cancer],
    verbose=1
)

# 2. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРОМ (CANCER)
print("\n" + "=" * 60)
print("2. ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ (BREAST CANCER)")
print("=" * 60)

autoencoder_cancer, encoder_ae_cancer =
create_autoencoder_pretrained_model(input_dim_cancer)
autoencoder_cancer.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse',
    metrics=['mae']
)
```

```
history_autoencoder_cancer = autoencoder_cancer.fit(
    X_train_cancer_scaled, X_train_cancer_scaled,
    epochs=30,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop_cancer],
    verbose=1
)

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

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

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

history_cancer_ae_pretrained = model_cancer_ae_pretrained.fit(
    X_train_cancer_scaled, y_train_cancer,
    epochs=50,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop_cancer],
    verbose=1
)

# 3. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM (CANCER)
print("\n" + "=" * 60)
print("3. ПРЕДОБУЧЕНИЕ RBM (BREAST CANCER)")
print("=" * 60)

# Создаем и обучаем стек RBM для cancer
stacked_rbm_cancer = StackedRBM(
    layer_sizes=[64, 32, 16],
    learning_rates=[0.1, 0.1, 0.1],
    k=1,
    epochs=20,
    batch_size=16
)

stacked_rbm_cancer.pretrain(X_train_cancer_scaled)

# Создаем модель с предобучением RBM
model_cancer_rbm_pretrained = create_rbm_pretrained_model(
    input_dim_cancer, stacked_rbm_cancer, output_dim=1,
problem_type='classification'
)
model_cancer_rbm_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
history_cancer_rbm_pretrained = model_cancer_rbm_pretrained.fit(
    X_train_cancer_scaled, y_train_cancer,
    epochs=50,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop_cancer],
    verbose=1
)

#
==============================================================================
# ОЦЕНКА И СРАВНЕНИЕ МОДЕЛЕЙ ДЛЯ BREAST CANCER
#
==============================================================================

print("\n" + "=" * 70)
print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ ДЛЯ ДАТАСЕТА BREAST CANCER")
print("=" * 70)

# Оценка всех моделей для cancer
results_cancer_no_pretrain = evaluate_classification_model(
    model_cancer_no_pretrain, X_test_cancer_scaled, y_test_cancer,
    "МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ (BREAST CANCER)"
)

results_cancer_ae_pretrained = evaluate_classification_model(
    model_cancer_ae_pretrained, X_test_cancer_scaled, y_test_cancer,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРА (BREAST CANCER)"
)

results_cancer_rbm_pretrained = evaluate_classification_model(
    model_cancer_rbm_pretrained, X_test_cancer_scaled, y_test_cancer,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ RBM (BREAST CANCER)"
)

#
==============================================================================
# ВИЗУАЛИЗАЦИЯ И ФИНАЛЬНОЕ СРАВНЕНИЕ
#
==============================================================================

# Графики обучения для credit approval
plt.figure(figsize=(18, 12))

# Credit Approval - графики потерь
plt.subplot(2, 3, 1)
plt.plot(history_credit_no_pretrain.history['loss'], label='Обучение')
plt.plot(history_credit_no_pretrain.history['val_loss'], label='Валидация')
plt.title('Credit Approval - Без предобучения\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()

plt.subplot(2, 3, 2)
plt.plot(history_credit_ae_pretrained.history['loss'], label='Обучение')
plt.plot(history_credit_ae_pretrained.history['val_loss'], label='Валидация')
plt.title('Credit Approval - Autoencoder\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()
```

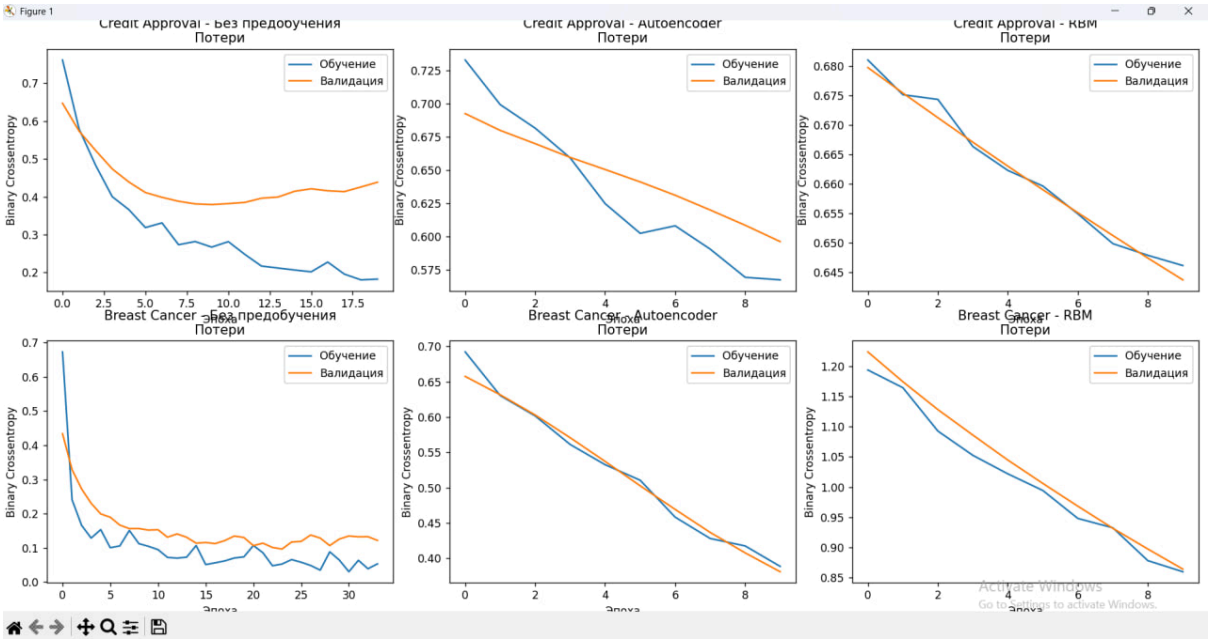
```
plt.subplot(2, 3, 3)
plt.plot(history_credit_rbm_pretrained.history['loss'], label='Обучение')
plt.plot(history_credit_rbm_pretrained.history['val_loss'],
label='Валидация')
plt.title('Credit Approval - RBM\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()

# Breast Cancer - графики потерь
plt.subplot(2, 3, 4)
plt.plot(history_cancer_no_pretrain.history['loss'], label='Обучение')
plt.plot(history_cancer_no_pretrain.history['val_loss'], label='Валидация')
plt.title('Breast Cancer - Без предобучения\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()

plt.subplot(2, 3, 5)
plt.plot(history_cancer_ae_pretrained.history['loss'], label='Обучение')
plt.plot(history_cancer_ae_pretrained.history['val_loss'], label='Валидация')
plt.title('Breast Cancer - Autoencoder\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()

plt.subplot(2, 3, 6)
plt.plot(history_cancer_rbm_pretrained.history['loss'], label='Обучение')
plt.plot(history_cancer_rbm_pretrained.history['val_loss'],
label='Валидация')
plt.title('Breast Cancer - RBM\nПотери')
plt.xlabel('Эпоха')
plt.ylabel('Binary Crossentropy')
plt.legend()

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

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

plt.subplot(1, 2, 1)
for i, history in enumerate(stacked_rbm_credit.histories):
    plt.plot(history, label=f'RBM {i + 1}')
plt.title('Ошибки реконструкции RBM\n(Credit Approval)')
plt.xlabel('Эпоха')
plt.ylabel('Ошибка реконструкции')
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
for i, history in enumerate(stacked_rbm_cancer.histories):
    plt.plot(history, label=f'RBM {i + 1}')
plt.title('Ошибки реконструкции RBM\n(Breast Cancer)')
plt.xlabel('Эпоха')
plt.ylabel('Ошибка реконструкции')
plt.legend()
plt.grid(True, alpha=0.3)
```

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

#
================================================================================
# ФИНАЛЬНОЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ
#
================================================================================

print("\n" + "=" * 70)
print("ФИНАЛЬНОЕ СРАВНЕНИЕ ВСЕХ МОДЕЛЕЙ")
print("=" * 70)

# Сравнение для credit approval
print("\n--- CREDIT APPROVAL DATASET ---")
credit_comparison = pd.DataFrame({
    'Метод': ['Без предобучения', 'Autoencoder', 'RBM'],
    'Точность': [
        results_credit_no_pretrain['accuracy'],
        results_credit_ae_pretrained['accuracy'],
        results_credit_rbm_pretrained['accuracy']
    ]
})
print(credit_comparison)

# Сравнение для breast cancer
print("\n--- BREAST CANCER DATASET ---")
cancer_comparison = pd.DataFrame({
    'Метод': ['Без предобучения', 'Autoencoder', 'RBM'],
    'Точность': [
        results_cancer_no_pretrain['accuracy'],
        results_cancer_ae_pretrained['accuracy'],
        results_cancer_rbm_pretrained['accuracy']
    ]
})
print(cancer_comparison)

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

# График для credit approval
methods = ['Без предобучения', 'Autoencoder', 'RBM']
credit_accuracies = [
    results_credit_no_pretrain['accuracy'],
    results_credit_ae_pretrained['accuracy'],
    results_credit_rbm_pretrained['accuracy']
]

ax1.bar(methods, credit_accuracies, color=['red', 'blue', 'green'],
alpha=0.7)
ax1.set_ylabel('Точность')
ax1.set_title('Сравнение точности для Credit Approval')
ax1.set_ylim(0, 1)
ax1.grid(True, alpha=0.3)

# Добавление значений на столбцы
for i, v in enumerate(credit_accuracies):
    ax1.text(i, v + 0.01, f'{v:.4f}', ha='center', va='bottom')

# График для breast cancer
```

```
cancer_accuracies = [
    results_cancer_no_pretrain['accuracy'],
    results_cancer_ae_pretrained['accuracy'],
    results_cancer_rbm_pretrained['accuracy']
]

ax2.bar(methods, cancer_accuracies, color=['red', 'blue', 'green'],
alpha=0.7)
ax2.set_ylabel('Точность')
ax2.set_title('Сравнение точности для Breast Cancer')
ax2.set_ylim(0, 1)
ax2.grid(True, alpha=0.3)

# Добавление значений на столбцы
for i, v in enumerate(cancer_accuracies):
    ax2.text(i, v + 0.01, f'{v:.4f}', ha='center', va='bottom')

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

#
================================================================================
# СОХРАНЕНИЕ МОДЕЛЕЙ И РЕЗУЛЬТАТОВ
#
================================================================================

# Сохранение моделей для credit approval
model_credit_no_pretrain.save('model_credit_no_pretrain.keras')
model_credit_ae_pretrained.save('model_credit_ae_pretrained.keras')
model_credit_rbm_pretrained.save('model_credit_rbm_pretrained.keras')

# Сохранение моделей для breast cancer
model_cancer_no_pretrain.save('model_cancer_no_pretrain.keras')
model_cancer_ae_pretrained.save('model_cancer_ae_pretrained.keras')
model_cancer_rbm_pretrained.save('model_cancer_rbm_pretrained.keras')

# Сохранение результатов сравнения
comparison_results = pd.DataFrame({
    'Dataset': ['Credit Approval'] * 3 + ['Breast Cancer'] * 3,
    'Method': methods * 2,
    'Accuracy': credit_accuracies + cancer_accuracies
})

comparison_results.to_csv('comparison_results.csv', index=False)
print("\nРезультаты сравнения сохранены в файл 'comparison_results.csv'")

print("\n" + "=" * 70)
print("ВСЕ МОДЕЛИ СОХРАНЕНЫ")
print("=" * 70)

# Вывод итоговых результатов
print("\nИТОГОВЫЕ РЕЗУЛЬТАТЫ:")
print(
    f"Credit Approval - Лучшая модель:
{methods[np.argmax(credit_accuracies)]} (Точность =
{max(credit_accuracies):.4f})")
print(
    f"Breast Cancer - Лучшая модель: {methods[np.argmax(cancer_accuracies)]}
(Точность = {max(cancer_accuracies):.4f})")
```

```
print("\n" + "=" * 70)
print("ЛАБОРАТОРНАЯ РАБОТА №4 ЗАВЕРШЕНА")
print("=" * 70)
```

Вывод:



Вывод: научилась осуществлять предобучение нейронных сетей с помощью RBM