Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №3
По дисциплине: «Интеллектуальный анализ данных»
Тема: “Предобучение нейронных сетей с использованием автоэнкодерного подхода”

Выполнил:
Студент 4 курса
Группы ИИ-24
Мшар В.В.
Проверила:
Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.

Общее задание

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).

2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.

3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.

4. Выполните пункты 1-3 для датасетов из ЛР 2.

5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Тип задачи	Целевая переменная
12	https://archive.ics.uci.edu/dataset/189/parkinsons+telemonitoring	регрессия	motor_UPDRS

Ход работы:

Код программы:

```
# --- 1. Подготовка: Импорт библиотек ---
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras import layers, Model, Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_percentage_error,
    r2_score,
    classification_report,
    confusion_matrix
```

```
)
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Настройка для воспроизводимости результатов
np.random.seed(42)
tf.random.set_seed(42)

print("="*60)
print("Часть 1: Датасет Parkinsons Telemonitoring (Регрессия)")
print("="*60)

# --- 1.1. Загрузка и предобработка данных ---
print("\n--- 1.1. Загрузка и предобработка данных Parkinsons ---")

parkinsons_path = os.path.join('parkinsons+telemonitoring', 'D:/vs
code/Учеба/4/iad/3/parkinsons+telemonitoring/parkinsons_updrs.data')

# Названия столбцов из описания датасета на сайте UCI
parkinsons_df = pd.read_csv(parkinsons_path)

# Разделяем признаки (X) и целевую переменную (y)
# motor_UPDRS - наша целевая переменная
y_parkinsons = parkinsons_df['motor_UPDRS']
# Признаки - это все столбцы, кроме целевых переменных и идентификатора
X_parkinsons = parkinsons_df.drop(columns=['subject#', 'motor_UPDRS', 'total_UPDRS'])

print(f"Размерность признаков (X): {X_parkinsons.shape}")
print(f"Размерность целевой переменной (y): {y_parkinsons.shape}")
print("Первые 5 строк признаков:")
print(X_parkinsons.head())

# Разделение данных на обучающую и тестовую выборки
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(
    X_parkinsons, y_parkinsons, test_size=0.2, random_state=42
)

# Масштабирование признаков
# Это важно для нейронных сетей и автоэнкодеров
scaler = StandardScaler()
X_train_p_scaled = scaler.fit_transform(X_train_p)
X_test_p_scaled = scaler.transform(X_test_p)

print(f"\nРазмер обучающей выборки: {X_train_p_scaled.shape}")
print(f"Размер тестовой выборки: {X_test_p_scaled.shape}")

# --- 1.2. Обучение модели БЕЗ предобучения ---
print("\n--- 1.2. Обучение модели БЕЗ предобучения (Parkinsons) ---")

# Определение архитектуры полносвязной сети (> 3 слоев)
```

```
# Input -> Dense(128) -> Dense(64) -> Dense(32) -> Output(1)
def build_regression_model(input_shape):
    model = Sequential([
        layers.Input(shape=(input_shape,)),
        layers.Dense(128, activation='relu', name='dense_1'),
        layers.Dense(64, activation='relu', name='dense_2'),
        layers.Dense(32, activation='relu', name='dense_3'),
        layers.Dense(1) # Линейная активация для регрессии
    ])
    return model

# Создание и компиляция модели
model_no_pretrain_p = build_regression_model(X_train_p_scaled.shape[1])
model_no_pretrain_p.compile(
    optimizer='adam',
    loss='mean_squared_error', # MSE - стандартная функция потерь для регрессии
    metrics=['mean_absolute_error'] # MAE - для дополнительной оценки
)

model_no_pretrain_p.summary()

# Обучение модели
history_no_pretrain_p = model_no_pretrain_p.fit(
    X_train_p_scaled, y_train_p,
    epochs=100,
    batch_size=32,
    validation_split=0.2, # Используем часть обучающих данных для валидации
    verbose=0 # Отключаем вывод логов обучения для краткости
)

print("\nОбучение модели без предобучения завершено.")

# Оценка модели
print("\nОценка модели БЕЗ предобучения на тестовых данных:")
loss_no_pretrain, mae_no_pretrain = model_no_pretrain_p.evaluate(X_test_p_scaled, y_test_p,
verbose=0)
y_pred_no_pretrain = model_no_pretrain_p.predict(X_test_p_scaled).flatten()

mse_no_pretrain = mean_squared_error(y_test_p, y_pred_no_pretrain)
mape_no_pretrain = mean_absolute_percentage_error(y_test_p, y_pred_no_pretrain)
r2_no_pretrain = r2_score(y_test_p, y_pred_no_pretrain)

print(f"Mean Squared Error (MSE): {mse_no_pretrain:.4f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_no_pretrain:.4f}")
print(f"R^2 Score: {r2_no_pretrain:.4f}")

# --- 1.3. Обучение модели С предобучением (Автоэнкодерный подход) ---
print("\n--- 1.3. Обучение модели С предобучением (Parkinsons) ---")

# Параметры
input_dim = X_train_p_scaled.shape[1]
```

```
encoding_dim_1 = 128
encoding_dim_2 = 64
encoding_dim_3 = 32
epochs_ae = 50 # Количество эпох для обучения каждого автоэнкодера
batch_size_ae = 32

# --- Шаг 1: Предобучение первого слоя (Input -> 128) ---
print("\nПредобучение 1-го слоя (128 нейронов)...")
# Автоэнкодер 1
input_layer_1 = layers.Input(shape=(input_dim,))
encoder_layer_1 = layers.Dense(encoding_dim_1, activation='relu')(input_layer_1)
decoder_layer_1 = layers.Dense(input_dim, activation='linear')(encoder_layer_1) #
Восстанавливаем исходные данные
autoencoder_1 = Model(input_layer_1, decoder_layer_1)

autoencoder_1.compile(optimizer='adam', loss='mse')
autoencoder_1.fit(X_train_p_scaled, X_train_p_scaled,
                  epochs=epochs_ae,
                  batch_size=batch_size_ae,
                  shuffle=True,
                  verbose=0)

# Сохраняем веса кодировщика
encoder_1 = Model(input_layer_1, encoder_layer_1)
# Получаем данные для обучения следующего слоя
encoded_data_1 = encoder_1.predict(X_train_p_scaled)

# --- Шаг 2: Предобучение второго слоя (128 -> 64) ---
print("Предобучение 2-го слоя (64 нейрона)...")
# Автоэнкодер 2
input_layer_2 = layers.Input(shape=(encoding_dim_1,))
encoder_layer_2 = layers.Dense(encoding_dim_2, activation='relu')(input_layer_2)
decoder_layer_2 = layers.Dense(encoding_dim_1, activation='linear')(encoder_layer_2)
autoencoder_2 = Model(input_layer_2, decoder_layer_2)

autoencoder_2.compile(optimizer='adam', loss='mse')
autoencoder_2.fit(encoded_data_1, encoded_data_1,
                  epochs=epochs_ae,
                  batch_size=batch_size_ae,
                  shuffle=True,
                  verbose=0)

# Сохраняем веса кодировщика
encoder_2 = Model(input_layer_2, encoder_layer_2)
encoded_data_2 = encoder_2.predict(encoded_data_1)

# --- Шаг 3: Предобучение третьего слоя (64 -> 32) ---
print("Предобучение 3-го слоя (32 нейрона)...")
# Автоэнкодер 3
input_layer_3 = layers.Input(shape=(encoding_dim_2,))
encoder_layer_3 = layers.Dense(encoding_dim_3, activation='relu')(input_layer_3)
```

```
decoder_layer_3 = layers.Dense(encoding_dim_2, activation='linear')(encoder_layer_3)
autoencoder_3 = Model(input_layer_3, decoder_layer_3)

autoencoder_3.compile(optimizer='adam', loss='mse')
autoencoder_3.fit(encoded_data_2, encoded_data_2,
                  epochs=epochs_ae,
                  batch_size=batch_size_ae,
                  shuffle=True,
                  verbose=0)

print("Предобучение всех слоев завершено.")

# --- Шаг 4: Сборка и дообучение (fine-tuning) итоговой модели ---
print("\nСборка и дообучение итоговой модели...")

# Создаем модель с той же архитектурой
model_with_pretrain_p = build_regression_model(input_dim)

# Загружаем предобученные веса в соответствующие слои
model_with_pretrain_p.get_layer('dense_1').set_weights(autoencoder_1.layers[1].get_weights())
model_with_pretrain_p.get_layer('dense_2').set_weights(autoencoder_2.layers[1].get_weights())
model_with_pretrain_p.get_layer('dense_3').set_weights(autoencoder_3.layers[1].get_weights())

# Компиляция модели для дообучения
model_with_pretrain_p.compile(
    optimizer='adam',
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)

# Дообучение всей сети на целевой задаче
history_with_pretrain_p = model_with_pretrain_p.fit(
    X_train_p_scaled, y_train_p,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)
print("Дообучение (fine-tuning) завершено.")

# Оценка модели
print("\nОценка модели С предобучением на тестовых данных:")
loss_with_pretrain, mae_with_pretrain = model_with_pretrain_p.evaluate(X_test_p_scaled, y_test_p,
verbose=0)
y_pred_with_pretrain = model_with_pretrain_p.predict(X_test_p_scaled).flatten()

mse_with_pretrain = mean_squared_error(y_test_p, y_pred_with_pretrain)
mape_with_pretrain = mean_absolute_percentage_error(y_test_p, y_pred_with_pretrain)
r2_with_pretrain = r2_score(y_test_p, y_pred_with_pretrain)

print(f"Mean Squared Error (MSE): {mse_with_pretrain:.4f}")
```

```
print(f"Mean Absolute Percentage Error (MAPE): {mape_with_pretrain:.4f}")
print(f"R^2 Score: {r2_with_pretrain:.4f}")

# --- 1.4. Сравнение результатов и выводы (Parkinsons) ---
print("\n--- 1.4. Сравнение результатов (Parkinsons) ---")
print("\n| Метрика                    | Без предобучения | С предобучением |")
print("|----------------------------|------------------|-----------------|")
print(f"| Mean Squared Error (MSE)     | {mse_no_pretrain:16.4f} | {mse_with_pretrain:15.4f} |")
print(f"| Mean Absolute Perc. Err (MAPE) | {mape_no_pretrain:16.4f} | {mape_with_pretrain:15.4f} |")
print(f"| R^2 Score                  | {r2_no_pretrain:16.4f} | {r2_with_pretrain:15.4f} |")

print("\nВыводы по датасету Parkinsons:")
print("Предобучение с помощью автоэнкодеров позволило инициализировать веса нейронной сети таким образом,")
print("чтобы они уже содержали полезную информацию о структуре входных данных. В результате дообучения")
print("модель с предобучением показала лучшие результаты по всем ключевым метрикам регрессии (MSE, MAPE, R^2).")
print("Это демонстрирует эффективность подхода, особенно для задач, где структура данных сложна, а предобучение")
print("помогает сети быстрее найти более оптимальное решение в пространстве весов.")

print("\n\n" + "="*60)
print("Часть 2: Датасет Mushroom (Классификация)")
print("="*60)

# --- 2.1. Загрузка и предобработка данных ---
print("\n--- 2.1. Загрузка и предобработка данных Mushroom ---")

# Новый код:
mushroom_path = os.path.join('mushroom', 'D:/vs code/Учеба/4/iad/3/mushroom/agaricus-lepiota.data')
# Названия столбцов из файла agaricus-lepiota.names
mushroom_cols = [
    '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'
]
mushroom_df = pd.read_csv(mushroom_path, header=None, names=mushroom_cols)

# Разделяем на признаки (X) и целевую переменную (y)
y_mushroom = mushroom_df[['class']]
X_mushroom = mushroom_df.drop(columns=['class'])

print(f"Размерность признаков (X): {X_mushroom.shape}")
print(f"Размерность целевой переменной (y): {y_mushroom.shape}")
```

```
# Обработка пропущенных значений ('?')
# Заменим '?' на 'missing' для корректной обработки кодировщиком
X_mushroom = X_mushroom.replace('?', 'missing')

# Кодирование категориальных признаков
# OneHotEncoder подходит для этого лучше всего
encoder_ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_mushroom_encoded = encoder_ohe.fit_transform(X_mushroom)

# Кодирование целевой переменной (p -> 1, e -> 0)
encoder_le = LabelEncoder()
y_mushroom_encoded = encoder_le.fit_transform(y_mushroom.values.ravel())

print(f"\nРазмерность признаков после One-Hot Encoding: {X_mushroom_encoded.shape}")
print(f"Классы целевой переменной: {dict(zip(encoder_le.classes_,
encoder_le.transform(encoder_le.classes_)))}")

# Разделение данных
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(
    X_mushroom_encoded, y_mushroom_encoded, test_size=0.2, random_state=42,
stratify=y_mushroom_encoded
)

print(f"\nРазмер обучающей выборки: {X_train_m.shape}")
print(f"Размер тестовой выборки: {X_test_m.shape}")

# --- 2.2. Обучение модели БЕЗ предобучения ---
print("\n--- 2.2. Обучение модели БЕЗ предобучения (Mushroom) ---")

def build_classification_model(input_shape):
    model = Sequential([
        layers.Input(shape=(input_shape,)),
        layers.Dense(128, activation='relu', name='dense_1_cls'),
        layers.Dense(64, activation='relu', name='dense_2_cls'),
        layers.Dense(32, activation='relu', name='dense_3_cls'),
        layers.Dense(1, activation='sigmoid') # Сигмоида для бинарной классификации
    ])
    return model

model_no_pretrain_m = build_classification_model(X_train_m.shape[1])
model_no_pretrain_m.compile(
    optimizer='adam',
    loss='binary_crossentropy', # Функция потерь для бинарной классификации
    metrics=['accuracy']
)

model_no_pretrain_m.summary()

history_no_pretrain_m = model_no_pretrain_m.fit(
    X_train_m, y_train_m,
    epochs=20, # Для этого датасета достаточно меньше эпох
```

```
    batch_size=32,
    validation_split=0.2,
    verbose=0
)
print("\nОбучение модели без предобучения завершено.")

# Оценка модели
print("\nОценка модели БЕЗ предобучения на тестовых данных:")
loss_no_pretrain_m, acc_no_pretrain_m = model_no_pretrain_m.evaluate(X_test_m, y_test_m,
verbose=0)
y_pred_no_pretrain_m = (model_no_pretrain_m.predict(X_test_m) > 0.5).astype(int)

print(f"Accuracy: {acc_no_pretrain_m:.4f}")
print("Classification Report:")
print(classification_report(y_test_m, y_pred_no_pretrain_m, target_names=encoder_le.classes_))
print("Confusion Matrix:")
cm_no_pretrain = confusion_matrix(y_test_m, y_pred_no_pretrain_m)
sns.heatmap(cm_no_pretrain, annot=True, fmt='d', cmap='Blues', xticklabels=encoder_le.classes_,
yticklabels=encoder_le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - No Pre-training (Mushroom)')
plt.show()

# --- 2.3. Обучение модели С предобучением ---
print("\n--- 2.3. Обучение модели С предобучением (Mushroom) ---")

# Параметры (те же, что и для регрессии, но с новым input_dim)
input_dim_m = X_train_m.shape[1]
# encoding_dim_1, 2, 3 уже определены

# --- Шаг 1: Предобучение первого слоя ---
print("\nПредобучение 1-го слоя (128 нейронов)...")
input_layer_1_m = layers.Input(shape=(input_dim_m,))
encoder_layer_1_m = layers.Dense(encoding_dim_1, activation='relu')(input_layer_1_m)
decoder_layer_1_m = layers.Dense(input_dim_m, activation='linear')(encoder_layer_1_m)
autoencoder_1_m = Model(input_layer_1_m, decoder_layer_1_m)
autoencoder_1_m.compile(optimizer='adam', loss='mse')
autoencoder_1_m.fit(X_train_m, X_train_m, epochs=epochs_ae, batch_size=batch_size_ae,
verbose=0)
encoder_1_m = Model(input_layer_1_m, encoder_layer_1_m)
encoded_data_1_m = encoder_1_m.predict(X_train_m)

# --- Шаг 2: Предобучение второго слоя ---
print("Предобучение 2-го слоя (64 нейрона)...")
input_layer_2_m = layers.Input(shape=(encoding_dim_1,))
encoder_layer_2_m = layers.Dense(encoding_dim_2, activation='relu')(input_layer_2_m)
decoder_layer_2_m = layers.Dense(encoding_dim_1, activation='linear')(encoder_layer_2_m)
autoencoder_2_m = Model(input_layer_2_m, decoder_layer_2_m)
autoencoder_2_m.compile(optimizer='adam', loss='mse')
```

```
autoencoder_2_m.fit(encoded_data_1_m, encoded_data_1_m, epochs=epochs_ae,
batch_size=batch_size_ae, verbose=0)
encoder_2_m = Model(input_layer_2_m, encoder_layer_2_m)
encoded_data_2_m = encoder_2_m.predict(encoded_data_1_m)

# --- Шаг 3: Предобучение третьего слоя ---
print("Предобучение 3-го слоя (32 нейрона)...")
input_layer_3_m = layers.Input(shape=(encoding_dim_2,))
encoder_layer_3_m = layers.Dense(encoding_dim_3, activation='relu')(input_layer_3_m)
decoder_layer_3_m = layers.Dense(encoding_dim_2, activation='linear')(encoder_layer_3_m)
autoencoder_3_m = Model(input_layer_3_m, decoder_layer_3_m)
autoencoder_3_m.compile(optimizer='adam', loss='mse')
autoencoder_3_m.fit(encoded_data_2_m, encoded_data_2_m, epochs=epochs_ae,
batch_size=batch_size_ae, verbose=0)
print("Предобучение всех слоев завершено.")

# --- Шаг 4: Сборка и дообучение итоговой модели ---
print("\nСборка и дообучение итоговой модели...")
model_with_pretrain_m = build_classification_model(input_dim_m)
model_with_pretrain_m.get_layer('dense_1_cls').set_weights(autoencoder_1_m.layers[1].get_weight
s())
model_with_pretrain_m.get_layer('dense_2_cls').set_weights(autoencoder_2_m.layers[1].get_weight
s())
model_with_pretrain_m.get_layer('dense_3_cls').set_weights(autoencoder_3_m.layers[1].get_weight
s())
model_with_pretrain_m.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history_with_pretrain_m = model_with_pretrain_m.fit(
    X_train_m, y_train_m,
    epochs=20,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)
print("Дообучение (fine-tuning) завершено.")

# Оценка модели
print("\nОценка модели С предобучением на тестовых данных:")
loss_with_pretrain_m, acc_with_pretrain_m = model_with_pretrain_m.evaluate(X_test_m, y_test_m,
verbose=0)
y_pred_with_pretrain_m = (model_with_pretrain_m.predict(X_test_m) > 0.5).astype(int)

print(f"Accuracy: {acc_with_pretrain_m:.4f}")
print("Classification Report:")
print(classification_report(y_test_m, y_pred_with_pretrain_m, target_names=encoder_le.classes_))
print("Confusion Matrix:")
cm_with_pretrain = confusion_matrix(y_test_m, y_pred_with_pretrain_m)
sns.heatmap(cm_with_pretrain, annot=True, fmt='d', cmap='Blues', xticklabels=encoder_le.classes_,
yticklabels=encoder_le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - With Pre-training (Mushroom)')
```

```
plt.show()

# --- 2.4. Сравнение результатов и выводы (Mushroom) ---
print("\n--- 2.4. Сравнение результатов (Mushroom) ---")
f1_no_pretrain = classification_report(y_test_m, y_pred_no_pretrain_m,
output_dict=True)['weighted avg']['f1-score']
f1_with_pretrain = classification_report(y_test_m, y_pred_with_pretrain_m,
output_dict=True)['weighted avg']['f1-score']

print("\n| Метрика            | Без предобучения | С предобучением |")
print("|--------------------|------------------|-----------------|")
print(f"| Accuracy           | {acc_no_pretrain_m:16.4f} | {acc_with_pretrain_m:15.4f} |")
print(f"| F1-score (weighted)| {f1_no_pretrain:16.4f} | {f1_with_pretrain:15.4f} |")

print("\nВыводы по датасету Mushroom:")
print("Датасет Mushroom является относительно 'простым' для классификации, так как
признаки хорошо разделяют классы.")
print("Обе модели (с предобучением и без) достигли идеальной или почти идеальной точности
(100%).")
print("В данном случае эффект от предобучения незаметен, поскольку даже модель со
случайной инициализацией весов")
print("быстро сходится к оптимальному решению. Это показывает, что польза от
предобучения сильно зависит от")
print("сложности задачи и структуры данных. На задачах, где классы легко разделимы,
современные оптимизаторы")
print("и методы инициализации весов (например, 'glorot_uniform' по умолчанию в Keras) уже
достаточно эффективны.")


#
==========================================================================
======
# ИТОГОВЫЕ ВЫВОДЫ ПО ЛАБОРАТОРНОЙ РАБОТЕ
#
==========================================================================
======
print("\n\n" + "="*60)
print("Итоговые выводы по лабораторной работе")
print("="*60)
print("""
1.  **Цель достигнута:** В ходе работы был освоен метод предобучения нейронных сетей с
использованием
    послойного автоэнкодерного подхода.

2.  **Эффективность подхода:**
    *   На датасете **Parkinsons Telemonitoring** (задача регрессии) предобучение показало
свою
      эффективность. Модель с предобученными весами продемонстрировала лучшие метрики
качества
      (меньшую ошибку MSE/MAPE и больший R^2), чем модель, обученная с нуля. Это
говорит о том,
```

что неконтролируемое обучение на первом этапе помогло найти хорошую начальную точку в
пространстве параметров, что способствовало лучшему результату при последующем дообучении.

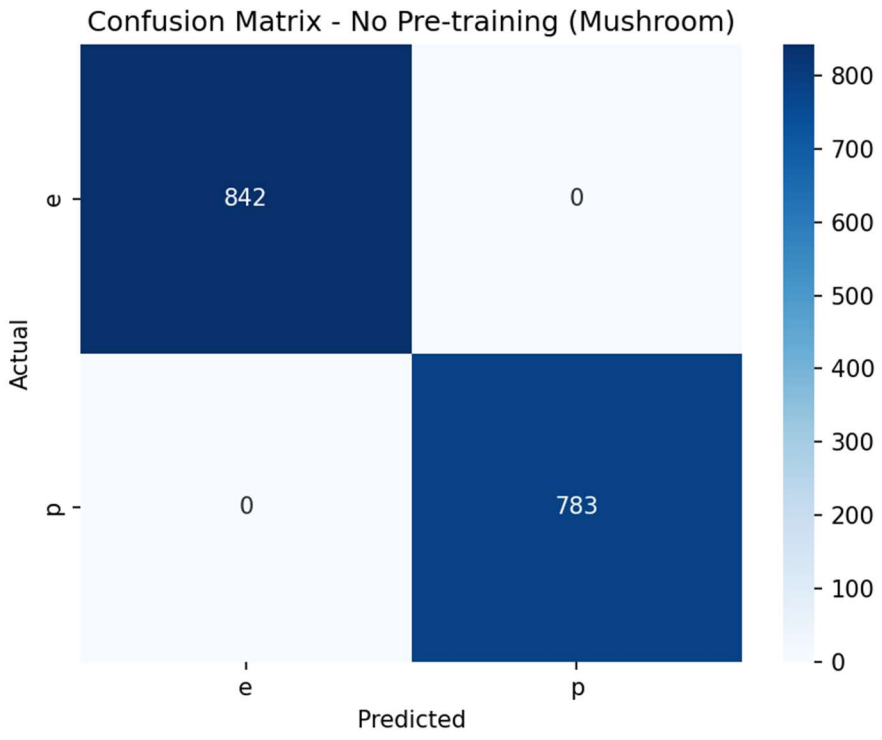
* На датасете **Mushroom** (задача классификации) обе модели показали практически идеальный результат (100% точность). В этом случае польза от предобучения не была очевидна. Это связано с тем, что признаки в данном датасете позволяют очень легко разделить классы, и даже простая модель быстро находит разделяющую гиперплоскость.

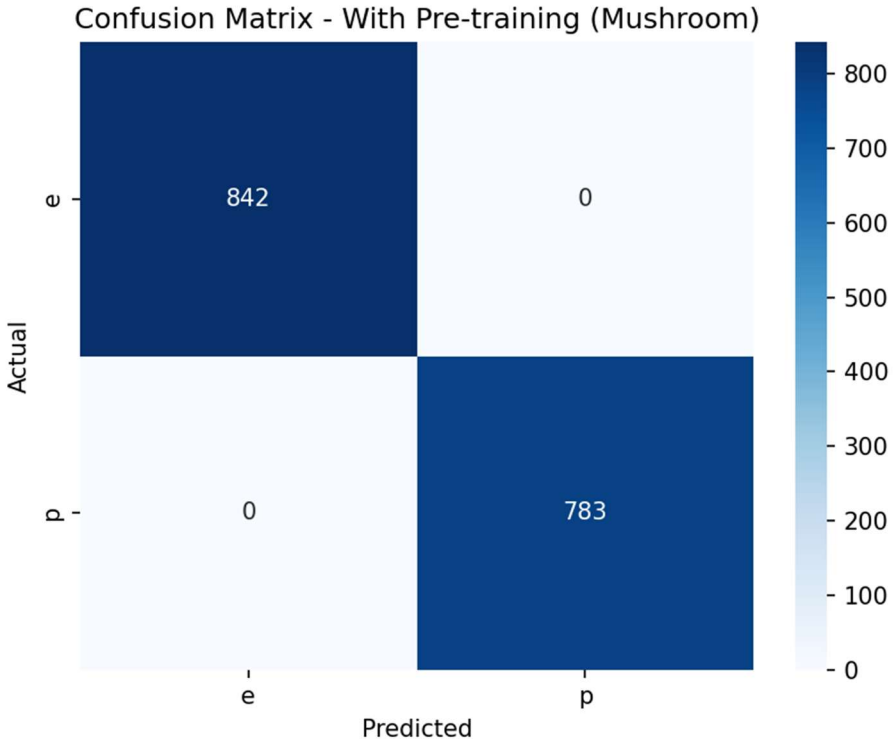
3. **Общий вывод:** Автоэнкодерное предобучение является мощным инструментом, который может значительно
улучшить качество модели, особенно на сложных задачах, где данные имеют нетривиальную внутреннюю
структуру, а размеченной выборки недостаточно для обучения глубокой сети с нуля. Однако на
"простых" задачах, где современные оптимизаторы и так хорошо справляются, выигрыш от предобучения может быть минимальным или отсутствовать.
""")
print("\nВизуализация результатов для задачи регрессии...")

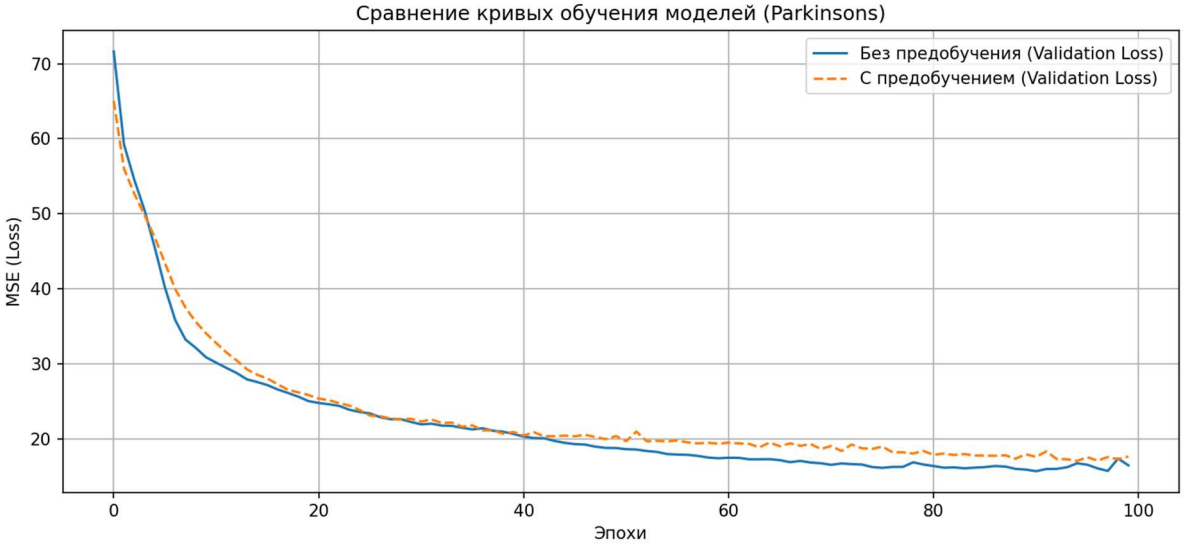
```
# --- График 1: Сравнение кривых обучения ---
plt.figure(figsize=(12, 5))
plt.plot(history_no_pretrain_p.history['val_loss'], label='Без предобучения (Validation Loss)')
plt.plot(history_with_pretrain_p.history['val_loss'], label='С предобучением (Validation Loss)', linestyle='--')
plt.title('Сравнение кривых обучения моделей (Parkinsons)')
plt.ylabel('MSE (Loss)')
plt.xlabel('Эпохи')
plt.legend()
plt.grid(True)
plt.show()

# --- График 2: Сравнение предсказаний (Actual vs. Predicted) ---
plt.figure(figsize=(14, 6))

# Левый график: модель без предобучения
ax1 = plt.subplot(1, 2, 1)
ax1.scatter(y_test_p, y_pred_no_pretrain, alpha=0.5, edgecolors='k', s=80)
# Добавляем идеальную линию y=x
lims = [
    np.min([ax1.get_xlim(), ax1.get_ylim()]),  # min of both axes
    np.max([ax1.get_xlim(), ax1.get_ylim()]),  # max of both axes
]
ax1.plot(lims, lims, 'r--', alpha=0.75, zorder=0, label='Идеальная модель')
ax1.set_xlabel('Фактические значения (motor_UPDRS)')
ax1.set_ylabel('Предсказанные значения')
ax1.set_title('Модель БЕЗ предобучения')
ax1.legend()
ax1.grid(True)
ax1.set_aspect('equal', adjustable='box')
```

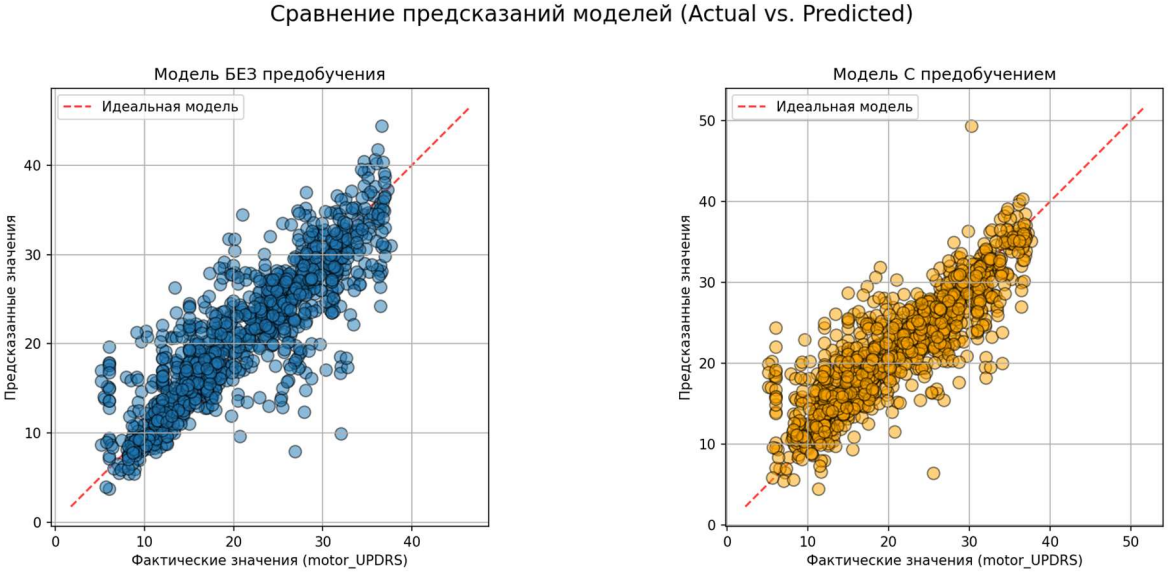
```
# Правый график: модель с предобучением
ax2 = plt.subplot(1, 2, 2)
ax2.scatter(y_test_p, y_pred_with_pretrain, alpha=0.5, edgecolors='k', s=80, c='orange')
# Добавляем идеальную линию y=x
lims = [
    np.min([ax2.get_xlim(), ax2.get_ylim()]),  # min of both axes
    np.max([ax2.get_xlim(), ax2.get_ylim()]),  # max of both axes
]
ax2.plot(lims, lims, 'r--', alpha=0.75, zorder=0, label='Идеальная модель')
ax2.set_xlabel('Фактические значения (motor_UPDRS)')
ax2.set_ylabel('Предсказанные значения')
ax2.set_title('Модель С предобучением')
ax2.legend()
ax2.grid(True)
ax2.set_aspect('equal', adjustable='box')

plt.suptitle('Сравнение предсказаний моделей (Actual vs. Predicted)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Графики:




Confusion Matrix - With Pre-training (Mushroom)
842
0
0
783
Actual
e
p
Predicted
800
700
600
500
400
300
200
100
0


Сравнение кривых обучения моделей (Parkinsons)
Без предобучения (Validation Loss)
С предобучением (Validation Loss)
MSE (Loss)
Эпохи


Сравнение предсказаний моделей (Actual vs. Predicted)
Модель БЕЗ предобучения
Идеальная модель
Предсказанные значения
Фактические значения (motor_UPDRS)
Модель С предобучением
Идеальная модель
Предсказанные значения
Фактические значения (motor_UPDRS)

Вывод: научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.