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»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Лящук А. 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.

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
from sklearn.metrics import confusion_matrix, classification_report, f1_score

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

```
# 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)

# 1. ИСПРАВЛЕННАЯ ОБРАБОТКА СТОЛБЦА С ДАТАМИ
print("Исправленная обработка столбца с датами...")

date_column = X.columns[0]
print(f"Столбец с датами: {date_column}")


def parse_date_fixed(date_str):
    """Исправленная функция для преобразования строк дат."""
    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:
            print(f"Неожиданный формат даты: {date_str} (длина:
{len(date_str)})")
            return pd.NaT
    except Exception as e:
        print(f"Ошибка преобразования даты '{date_str}': {e}")
        return pd.NaT


# Применяем исправленную функцию
X[date_column] = X[date_column].apply(parse_date_fixed)

# Проверяем результат преобразования
print(f"Уникальные значения дат: {X[date_column].nunique()}")
print(f"Диапазон дат: от {X[date_column].min()} до {X[date_column].max()}")
print(f"Количество NaT после преобразования: {X[date_column].isna().sum()}")

# 2. РАЗДЕЛЕНИЕ ДАТЫ НА ЧАСТИ
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:
    print("Внимание: все даты являются NaT! Создаем базовые временные
признаки...")
    X['time_index'] = range(len(X))
    X['period_of_day'] = (X['time_index'] % 6)

# Удаляем исходный столбец с датой
X = X.drop(columns=[date_column])
```

```
# 3. ПРОВЕРКА И ОЧИСТКА ДАННЫХ
print("Проверка данных после обработки:")
print(f"Форма X: {X.shape}")
print(f"Пропущенные значения в X: {X.isna().sum().sum()}")

# Заполняем пропуски только в числовых столбцах
numeric_columns = X.select_dtypes(include=[np.number]).columns
X[numeric_columns] = X[numeric_columns].fillna(X[numeric_columns].mean())

# Проверяем целевую переменную
print(f"Пропущенные значения в y: {y.isna().sum().sum()}")
y = y.fillna(y.mean())

# 4. 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
)

# 5. 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)

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


# 6. АВТОЭНКОДЕРЫ ДЛЯ ВИЗУАЛИЗАЦИИ ГЛАВНЫХ КОМПОНЕНТ
def create_visualization_autoencoder_2d(input_dim):
    """Создает автоэнкодер для визуализации в 2D пространстве"""
    encoder = models.Sequential([
        layers.Dense(128, activation='relu', input_shape=(input_dim,)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(2, activation='linear', name='bottleneck_2d')  # 2
нейрона для 2D визуализации
    ])

    decoder = models.Sequential([
        layers.Dense(32, activation='relu', input_shape=(2,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(input_dim, activation='linear')
    ])

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


def create_visualization_autoencoder_3d(input_dim):
    """Создает автоэнкодер для визуализации в 3D пространстве"""
    encoder = models.Sequential([
        layers.Dense(128, activation='relu', input_shape=(input_dim,)),
```

```
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(3, activation='linear', name='bottleneck_3d')  # 3
нейрона для 3D визуализации
    ])

    decoder = models.Sequential([
        layers.Dense(32, activation='relu', input_shape=(3,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(input_dim, activation='linear')
    ])

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


# 7. ВИЗУАЛИЗАЦИЯ С ПОМОЩЬЮ АВТОЭНКОДЕРОВ :cite[1]
print("=" * 60)
print("ВИЗУАЛИЗАЦИЯ ГЛАВНЫХ КОМПОНЕНТ С АВТОЭНКОДЕРОМ")
print("=" * 60)

input_dim = X_train_scaled.shape[1]

# Создаем и обучаем автоэнкодер для 2D визуализации
print("Обучение автоэнкодера для 2D визуализации...")
autoencoder_2d, encoder_2d = create_visualization_autoencoder_2d(input_dim)
autoencoder_2d.compile(optimizer='adam', loss='mse', metrics=['mae'])

history_2d = autoencoder_2d.fit(
    X_train_scaled, X_train_scaled,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    verbose=0,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True)]
)

# Создаем и обучаем автоэнкодер для 3D визуализации
print("Обучение автоэнкодера для 3D визуализации...")
autoencoder_3d, encoder_3d = create_visualization_autoencoder_3d(input_dim)
autoencoder_3d.compile(optimizer='adam', loss='mse', metrics=['mae'])

history_3d = autoencoder_3d.fit(
    X_train_scaled, X_train_scaled,
    epochs=50,
    batch_size=128,
    validation_split=0.2,
    verbose=0,
    callbacks=[EarlyStopping(patience=10, restore_best_weights=True)]
)

# Получаем кодированные представления
encoded_2d = encoder_2d.predict(X_train_scaled)
encoded_3d = encoder_3d.predict(X_train_scaled)

# 8. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ АВТОЭНКОДЕРА
print("Создание визуализаций автоэнкодера...")
```

```
# Для цветового кодирования создаем категории на основе квартилей целевой
переменной
y_train_categories = pd.cut(y_train.iloc[:, 0], bins=4, labels=[0, 1, 2, 3])

# 2D визуализация
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
scatter = plt.scatter(encoded_2d[:, 0], encoded_2d[:, 1],
c=y_train_categories,
                      cmap='viridis', alpha=0.7)
plt.colorbar(scatter, label='Категория энергии')
plt.xlabel('Главная компонента 1')
plt.ylabel('Главная компонента 2')
plt.title('Автоэнкодер: 2D проекция данных')

# 3D визуализация
ax = plt.subplot(1, 3, 2, projection='3d')
scatter_3d = ax.scatter(encoded_3d[:, 0], encoded_3d[:, 1], encoded_3d[:, 2],
                        c=y_train_categories, cmap='viridis', alpha=0.7)
plt.colorbar(scatter_3d, label='Категория энергии')
ax.set_xlabel('Главная компонента 1')
ax.set_ylabel('Главная компонента 2')
ax.set_zlabel('Главная компонента 3')
ax.set_title('Автоэнкодер: 3D проекция данных')

# 9. ВИЗУАЛИЗАЦИЯ С ПОМОЩЬЮ t-SNE :cite[3]:cite[7]
print("Применение t-SNE для визуализации...")

# t-SNE в 2D
tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=300)
X_tsne_2d = tsne_2d.fit_transform(X_train_scaled)

# t-SNE в 3D
tsne_3d = TSNE(n_components=3, random_state=42, perplexity=30, max_iter=300)
X_tsne_3d = tsne_3d.fit_transform(X_train_scaled)

# Визуализация t-SNE
plt.subplot(1, 3, 3)
scatter_tsne = plt.scatter(X_tsne_2d[:, 0], X_tsne_2d[:, 1],
                           c=y_train_categories, cmap='plasma', alpha=0.7)
plt.colorbar(scatter_tsne, label='Категория энергии')
plt.xlabel('t-SNE компонента 1')
plt.ylabel('t-SNE компонента 2')
plt.title('t-SNE: 2D проекция данных')

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

# 10. 3D ВИЗУАЛИЗАЦИЯ T-SNE ОТДЕЛЬНО
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_tsne_3d[:, 0], X_tsne_3d[:, 1], X_tsne_3d[:, 2],
                     c=y_train_categories, cmap='plasma', alpha=0.7)
plt.colorbar(scatter, label='Категория энергии')
ax.set_xlabel('t-SNE компонента 1')
ax.set_ylabel('t-SNE компонента 2')
ax.set_zlabel('t-SNE компонента 3')
ax.set_title('t-SNE: 3D проекция данных')
plt.savefig('tsne_3d_visualization.png', dpi=300, bbox_inches='tight')
```

```
plt.show()

# 11. СРАВНЕНИЕ МЕТОДОВ ВИЗУАЛИЗАЦИИ
print("=" * 60)
print("СРАВНЕНИЕ МЕТОДОВ ВИЗУАЛИЗАЦИИ")
print("=" * 60)


# Вычисляем дисперсию для каждого метода
def explained_variance_ratio(projected_data):
    """Вычисляет долю объясненной дисперсии для спроецированных данных"""
    total_var = np.var(projected_data, axis=0).sum()
    return total_var


ae_2d_variance = explained_variance_ratio(encoded_2d)
ae_3d_variance = explained_variance_ratio(encoded_3d)
tsne_2d_variance = explained_variance_ratio(X_tsne_2d)
tsne_3d_variance = explained_variance_ratio(X_tsne_3d)

print(f"Дисперсия в 2D автоэнкодере: {ae_2d_variance:.4f}")
print(f"Дисперсия в 3D автоэнкодере: {ae_3d_variance:.4f}")
print(f"Дисперсия в 2D t-SNE: {tsne_2d_variance:.4f}")
print(f"Дисперсия в 3D t-SNE: {tsne_3d_variance:.4f}")


# 12. УЛУЧШЕННЫЕ АРХИТЕКТУРЫ МОДЕЛЕЙ ДЛЯ ОСНОВНОЙ ЗАДАЧИ
def create_base_model(input_dim):
    """Создает улучшенную базовую модель для регрессии"""
    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(input_dim):
    """Создает улучшенный автоэнкодер с регуляризацией"""
    # Энкодер
    encoder = models.Sequential([
        layers.Dense(256, activation='relu', input_shape=(input_dim,),
                     kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu',
                     kernel_regularizer=keras.regularizers.l2(0.001)),
        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')
    ])

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


def create_pretrained_model(encoder, input_dim):
    """Создает модель с предобученным энкодером"""
    model = models.Sequential()

    # Добавляем предобученные слои энкодера (замораживаем первые слои)
    for i, layer in enumerate(encoder.layers[:-1]):
        if i < len(encoder.layers) - 2:
            layer.trainable = False
        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


# 13. ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ
print("=" * 60)
print("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']
)

early_stop = EarlyStopping(monitor='val_loss', patience=15,
restore_best_weights=True)

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
)

# 14. ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ
print("=" * 60)
print("ПРЕДОБУЧЕНИЕ АВТОЭНКОДЕРОМ")
print("=" * 60)
```

```
autoencoder, encoder = create_autoencoder(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
)

# 15. ОБУЧЕНИЕ С ПРЕДОБУЧЕНИЕМ
print("=" * 60)
print("ОБУЧЕНИЕ С ПРЕДОБУЧЕНИЕМ")
print("=" * 60)

model_pretrained = create_pretrained_model(encoder, input_dim)
model_pretrained.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='mse',
    metrics=['mae']
)

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

# 16. ОЦЕНКА ЭФФЕКТИВНОСТИ МОДЕЛЕЙ
print("=" * 60)
print("ОЦЕНКА ЭФФЕКТИВНОСТИ МОДЕЛЕЙ")
print("=" * 60)


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
    }


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

results_pretrained = evaluate_model(
    model_pretrained, X_test_scaled, y_test_scaled, scaler_y,
    "МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ"
)

# 17. ВИЗУАЛИЗАЦИЯ СРАВНЕНИЯ МОДЕЛЕЙ
plt.figure(figsize=(15, 5))

# Графики обучения
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_autoencoder.history['loss'], label='Обучение')
plt.plot(history_autoencoder.history['val_loss'], label='Валидация')
plt.title('Автоэнкодер - Потери')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.legend()

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

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

# 18. ФИНАЛЬНОЕ СРАВНЕНИЕ РЕЗУЛЬТАТОВ
print("=" * 60)
print("ФИНАЛЬНОЕ СРАВНЕНИЕ РЕЗУЛЬТАТОВ")
print("=" * 60)

# 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']
    ],
    'С предобучением': [
        results_pretrained['MAE'],
        results_pretrained['MSE'],
        results_pretrained['RMSE'],
        results_pretrained['R2']
    ]
})

print(comparison_df)

# Визуализация сравнения метрик
plt.figure(figsize=(10, 6))
metrics = ['MAE', 'RMSE', 'R²']
values_no_pretrain = [results_no_pretrain['MAE'],
results_no_pretrain['RMSE'], results_no_pretrain['R2']]
values_pretrained = [results_pretrained['MAE'], results_pretrained['RMSE'],
results_pretrained['R2']]

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

plt.bar(x - width / 2, values_no_pretrain, width, label='Без предобучения',
alpha=0.8)
plt.bar(x + width / 2, values_pretrained, width, label='С предобучением',
alpha=0.8)

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

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

# 19. СОХРАНЕНИЕ МОДЕЛЕЙ
model_no_pretrain.save('model_no_pretrain.keras')
model_pretrained.save('model_pretrained.keras')
autoencoder.save('autoencoder.keras')
autoencoder_2d.save('autoencoder_2d.keras')
autoencoder_3d.save('autoencoder_3d.keras')

print("Все модели сохранены в формате .keras")

print("=" * 60)
print("ЛАБОРАТОРНАЯ РАБОТА ЗАВЕРШЕНА")
print("=" * 60)
```

Вывод:

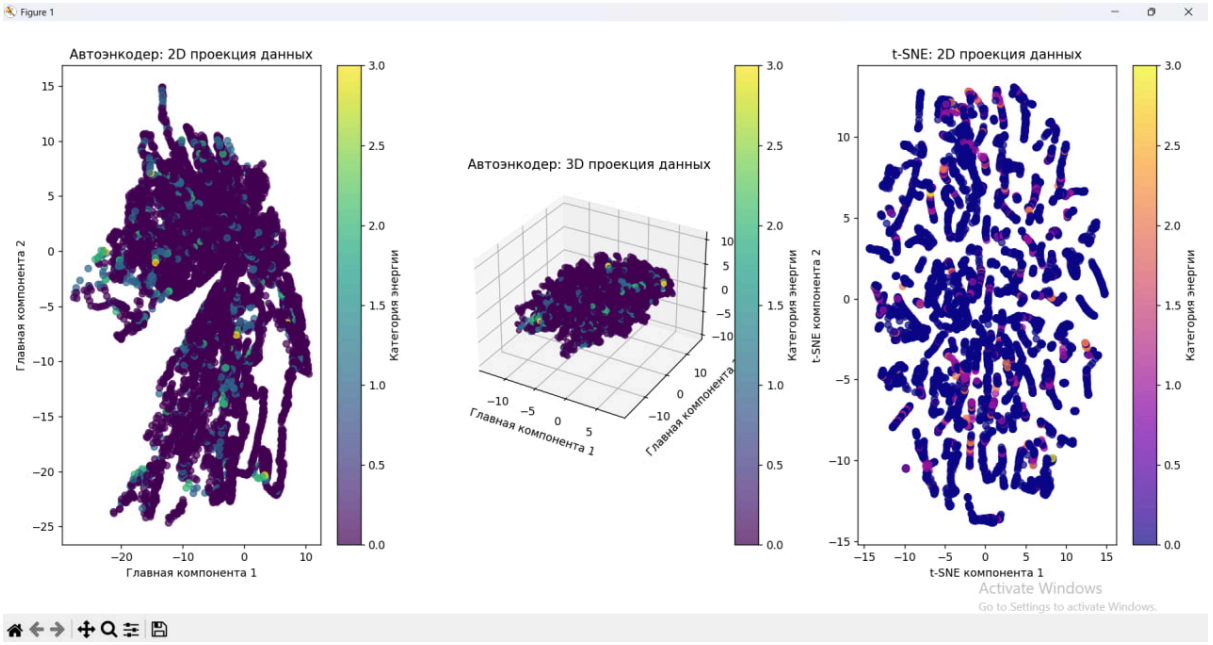


Figure 1
Автоэнкодер: 2D проекция данных
Главная компонента 1
Главная компонента 2
Категория энергии
Автоэнкодер: 3D проекция данных
Главная компонента 1
Главная компонента 2
Категория энергии
t-SNE: 2D проекция данных
t-SNE компонента 1
t-SNE компонента 2
Категория энергии
Activate Windows
Go to Settings to activate Windows.

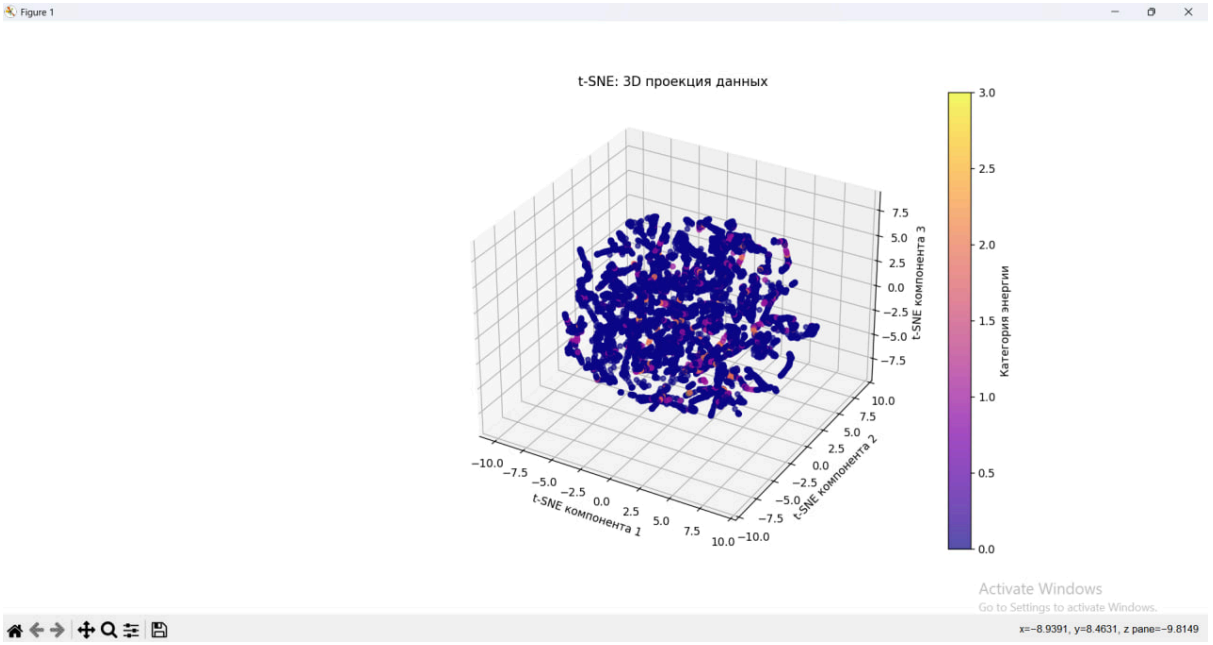
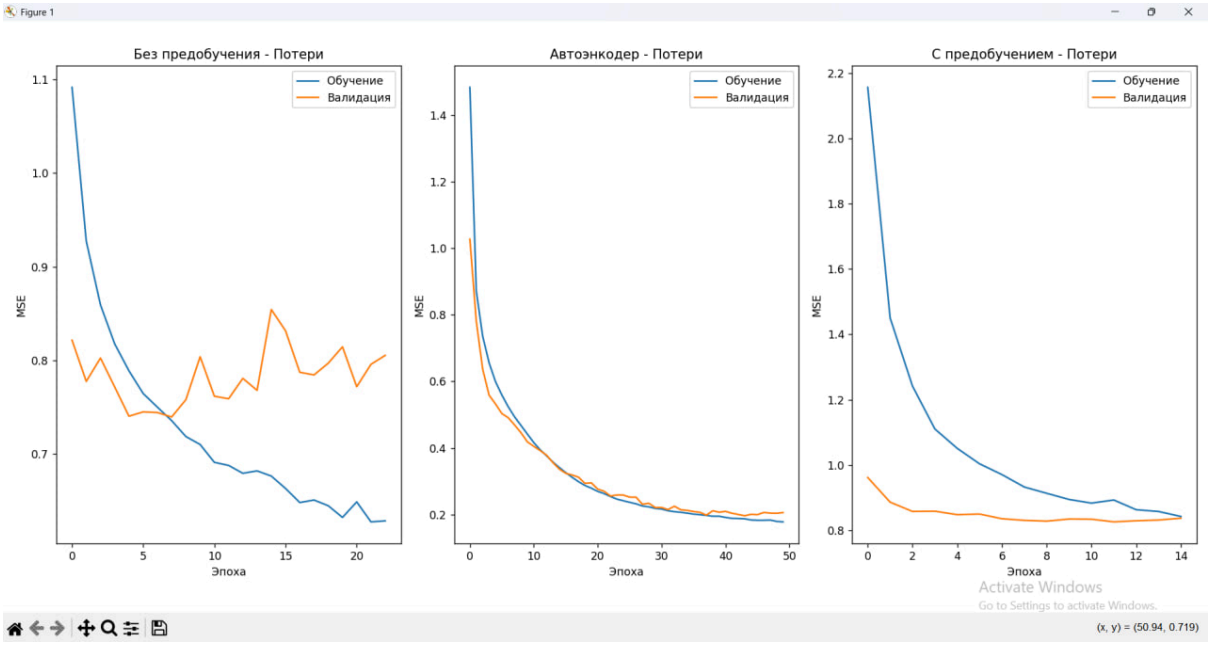
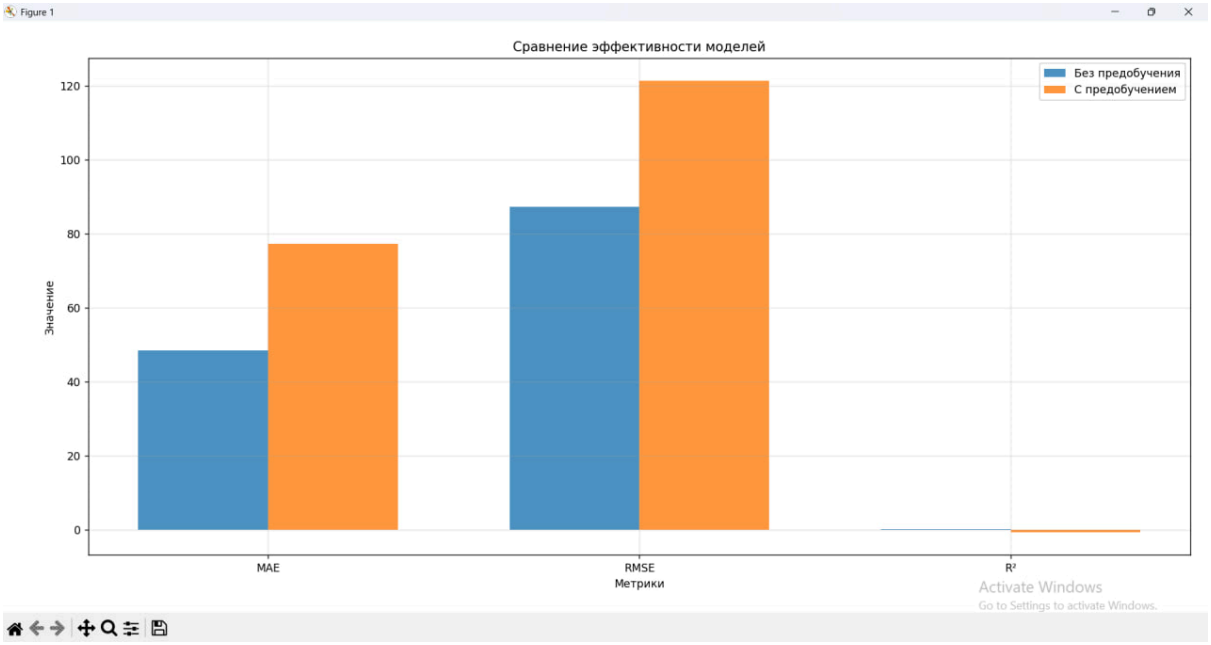


Figure 1
t-SNE: 3D проекция данных
t-SNE компонента 1
t-SNE компонента 2
t-SNE компонента 3
Категория энергии
Activate Windows
Go to Settings to activate Windows.
x=−8.9391, y=8.4631, z pane=−9.8149





Вывод: научилась осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода