

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-23
Швороб В.А

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
12	parkinsons+telemonitoring	регрессия	motor_UPDRS

Код:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

np.random.seed(42)
tf.random.set_seed(42)

data = pd.read_csv('parkinsons_updrs.data')
print(data.info())
```

```

print(data.head())

features = data.drop(['subject#', 'age', 'sex', 'test_time', 'motor_UPDRS', 'total_UPDRS'], axis=1)
target = data['motor_UPDRS']

print(features.columns.tolist())

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

X_train, X_test, y_train, y_test = train_test_split(
    features_scaled, target, test_size=0.2, random_state=42
)

def create_base_model(input_dim):
    model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=(input_dim,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(1)
    ])
    return model

base_model = create_base_model(X_train.shape[1])
base_model.compile(optimizer=Adam(learning_rate=0.001),
                    loss='mse',
                    metrics=['mae'])

history_base = base_model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)

base_pred = base_model.predict(X_test)
base_mae = mean_absolute_error(y_test, base_pred)
base_mse = mean_squared_error(y_test, base_pred)
base_r2 = r2_score(y_test, base_pred)

print(f"Base Model - MAE: {base_mae:.4f}, MSE: {base_mse:.4f}, R2: {base_r2:.4f}")

class AutoencoderPretrainedModel:
    def __init__(self, input_dim, encoding_dims):
        self.input_dim = input_dim
        self.encoding_dims = encoding_dims
        self.autoencoders = []
        self.encoders = []

    def train_autoencoders(self, X, epochs=50):
        current_input = X

        for i, encoding_dim in enumerate(self.encoding_dims):
            input_layer = layers.Input(shape=(current_input.shape[1],))
            encoded = layers.Dense(encoding_dim, activation='relu')(input_layer)
            decoded = layers.Dense(current_input.shape[1], activation='linear')(encoded)

            autoencoder = Model(input_layer, decoded)

```

```

encoder = Model(input_layer, encoded)

autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

autoencoder.fit(
    current_input, current_input,
    epochs=epochs,
    batch_size=32,
    verbose=0,
    shuffle=True
)

self.autoencoders.append(autoencoder)
self.encoders.append(encoder)

current_input = encoder.predict(current_input)

def build_final_model(self):
    input_layer = layers.Input(shape=(self.input_dim,))

    x = input_layer
    for encoder in self.encoders:
        x = encoder(x)

    x = layers.Dense(16, activation='relu')(x)
    output = layers.Dense(1)(x)

    model = Model(input_layer, output)
    return model

autoencoder_model = AutoencoderPretrainedModel(
    input_dim=X_train.shape[1],
    encoding_dims=[32, 16]
)

autoencoder_model.train_autoencoders(X_train, epochs=50)

pretrained_model = autoencoder_model.build_final_model()
pretrained_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

history_pretrained = pretrained_model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)

pretrained_pred = pretrained_model.predict(X_test)
pretrained_mae = mean_absolute_error(y_test, pretrained_pred)
pretrained_mse = mean_squared_error(y_test, pretrained_pred)
pretrained_r2 = r2_score(y_test, pretrained_pred)

print(f'Pretrained Model - MAE: {pretrained_mae:.4f}, MSE: {pretrained_mse:.4f}, R2: {pretrained_r2:.4f}')

plt.figure(figsize=(15, 5))

```

```

plt.subplot(1, 3, 1)
plt.plot(history_base.history['loss'], label='Base Train')
plt.plot(history_base.history['val_loss'], label='Base Val')
plt.plot(history_pretrained.history['loss'], label='Pretrained Train')
plt.plot(history_pretrained.history['val_loss'], label='Pretrained Val')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 3, 2)
plt.scatter(y_test, base_pred, alpha=0.5, label=f'Base (R2: {base_r2:.3f})')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Base Model Predictions')
plt.legend()

plt.subplot(1, 3, 3)
plt.scatter(y_test, pretrained_pred, alpha=0.5, label=f'Pretrained (R2: {pretrained_r2:.3f})')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Pretrained Model Predictions')
plt.legend()

plt.tight_layout()
plt.show()

results_comparison = pd.DataFrame({
    'Model': ['Base', 'Pretrained'],
    'MAE': [base_mae, pretrained_mae],
    'MSE': [base_mse, pretrained_mse],
    'R2': [base_r2, pretrained_r2]
})
print(results_comparison)

class AutoencoderPCA:
    def __init__(self, input_dim, encoding_dim):
        self.input_dim = input_dim
        self.encoding_dim = encoding_dim
        self.autoencoder = None
        self.encoder = None

    def build_model(self):
        input_layer = layers.Input(shape=(self.input_dim,))
        encoded = layers.Dense(self.encoding_dim, activation='linear')(input_layer)
        decoded = layers.Dense(self.input_dim, activation='linear')(encoded)

        self.autoencoder = Model(input_layer, decoded)
        self.encoder = Model(input_layer, encoded)

        self.autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

    def train(self, X, epochs=100):
        self.build_model()
        history = self.autoencoder.fit(
            X, X,
            epochs=epochs,
            batch_size=32,

```

```

        shuffle=True,
        verbose=0
    )
    return history

def encode(self, X):
    return self.encoder.predict(X)

features_for_pca = data.drop(['subject#', 'motor_UPDRS', 'total_UPDRS'], axis=1)
scaler_pca = StandardScaler()
features_pca_scaled = scaler_pca.fit_transform(features_for_pca)

autoencoder_pca_2d = AutoencoderPCA(features_pca_scaled.shape[1], 2)
autoencoder_pca_2d.train(features_pca_scaled, epochs=100)
components_2d = autoencoder_pca_2d.encode(features_pca_scaled)

autoencoder_pca_3d = AutoencoderPCA(features_pca_scaled.shape[1], 3)
autoencoder_pca_3d.train(features_pca_scaled, epochs=100)
components_3d = autoencoder_pca_3d.encode(features_pca_scaled)

motor_updrs_bins = pd.cut(data['motor_UPDRS'], bins=3, labels=['Low', 'Medium', 'High'])

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
scatter = plt.scatter(components_2d[:, 0], components_2d[:, 1],
                     c=pd.factorize(motor_updrs_bins)[0],
                     cmap='viridis', alpha=0.6)
plt.colorbar(scatter, label='motor_UPDRS Level')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('2D Autoencoder Projection')

plt.subplot(1, 2, 2)
ax = plt.axes(projection='3d')
scatter = ax.scatter3D(components_3d[:, 0], components_3d[:, 1], components_3d[:, 2],
                      c=pd.factorize(motor_updrs_bins)[0],
                      cmap='viridis', alpha=0.6)
plt.colorbar(scatter, label='motor_UPDRS Level')
ax.set_xlabel('First Principal Component')
ax.set_ylabel('Second Principal Component')
ax.set_zlabel('Third Principal Component')
ax.set_title('3D Autoencoder Projection')

plt.tight_layout()
plt.show()

from sklearn.manifold import TSNE

tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30)
components_tsne_2d = tsne_2d.fit_transform(features_pca_scaled)

tsne_3d = TSNE(n_components=3, random_state=42, perplexity=30)
components_tsne_3d = tsne_3d.fit_transform(features_pca_scaled)

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
scatter = plt.scatter(components_tsne_2d[:, 0], components_tsne_2d[:, 1],
                     c=pd.factorize(motor_updrs_bins)[0],
                     cmap='viridis', alpha=0.6)

```

```

plt.colorbar(scatter, label='motor_UPDRS Level')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('2D t-SNE Visualization')

plt.subplot(1, 2, 2)
ax = plt.axes(projection='3d')
scatter = ax.scatter3D(components_tsne_3d[:, 0], components_tsne_3d[:, 1], components_tsne_3d[:, 2],
                      c=pd.factorize(motor_updrs_bins)[0],
                      cmap='viridis', alpha=0.6)
plt.colorbar(scatter, label='motor_UPDRS Level')
ax.set_xlabel('t-SNE Component 1')
ax.set_ylabel('t-SNE Component 2')
ax.set_zlabel('t-SNE Component 3')
ax.set_title('3D t-SNE Visualization')

plt.tight_layout()
plt.show()

print("Comparison of training approaches:")
print(f"Base model final validation MAE: {history_base.history['val_mae'][-1]:.4f}")
print(f"Pretrained model final validation MAE: {history_pretrained.history['val_mae'][-1]:.4f}")

improvement_mae = ((base_mae - pretrained_mae) / base_mae) * 100
improvement_r2 = ((pretrained_r2 - base_r2) / abs(base_r2)) * 100

print(f"MAE improvement with pretraining: {improvement_mae:.2f}%")
print(f"R2 improvement with pretraining: {improvement_r2:.2f}%")

```

Вывод программы:

```

RangeIndex: 5875 entries, 0 to 5874
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   subject#          5875 non-null    int64  
 1   age               5875 non-null    int64  
 2   sex               5875 non-null    int64  
 3   test_time         5875 non-null    float64
 4   motor_UPDRS       5875 non-null    float64
 5   total_UPDRS       5875 non-null    float64
 6   Jitter(%)         5875 non-null    float64
 7   Jitter(Abs)        5875 non-null    float64
 8   Jitter:RAP         5875 non-null    float64
 9   Jitter:PPQ5         5875 non-null    float64
 10  Jitter:DDP         5875 non-null    float64
 11  Shimmer            5875 non-null    float64
 12  Shimmer(dB)        5875 non-null    float64
 13  Shimmer:APQ3        5875 non-null    float64
 14  Shimmer:APQ5        5875 non-null    float64
 15  Shimmer:APQ11        5875 non-null    float64
 16  Shimmer:DDA         5875 non-null    float64

```

```

17    NHR          5875 non-null   float64
18    HNR          5875 non-null   float64
19    RPDE         5875 non-null   float64
20    DFA          5875 non-null   float64
21    PPE          5875 non-null   float64
dtypes: float64(19), int64(3)
memory usage: 1009.9 KB
None
      subject#  age  sex  test_time  ...    HNR    RPDE    DFA    PPE
0            1   72    0     5.6431  ...  21.640  0.41888  0.54842  0.16006
1            1   72    0    12.6660  ...  27.183  0.43493  0.56477  0.10810
2            1   72    0    19.6810  ...  23.047  0.46222  0.54405  0.21014
3            1   72    0    25.6470  ...  24.445  0.48730  0.57794  0.33277
4            1   72    0    33.6420  ...  26.126  0.47188  0.56122  0.19361
[5 rows x 22 columns]
['Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP',
 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11',
 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE']
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
37/37 ━━━━━━━━ 0s 2ms/step
Base Model - MAE: 5.3052, MSE: 46.2440, R2: 0.2755
147/147 ━━━━━━━━ 0s 741us/step
147/147 ━━━━━━━━ 0s 757us/step
37/37 ━━━━━━━━ 0s 3ms/step
Pretrained Model - MAE: 5.4444, MSE: 46.5019, R2: 0.2715

```

Model Loss

Base Model Predictions

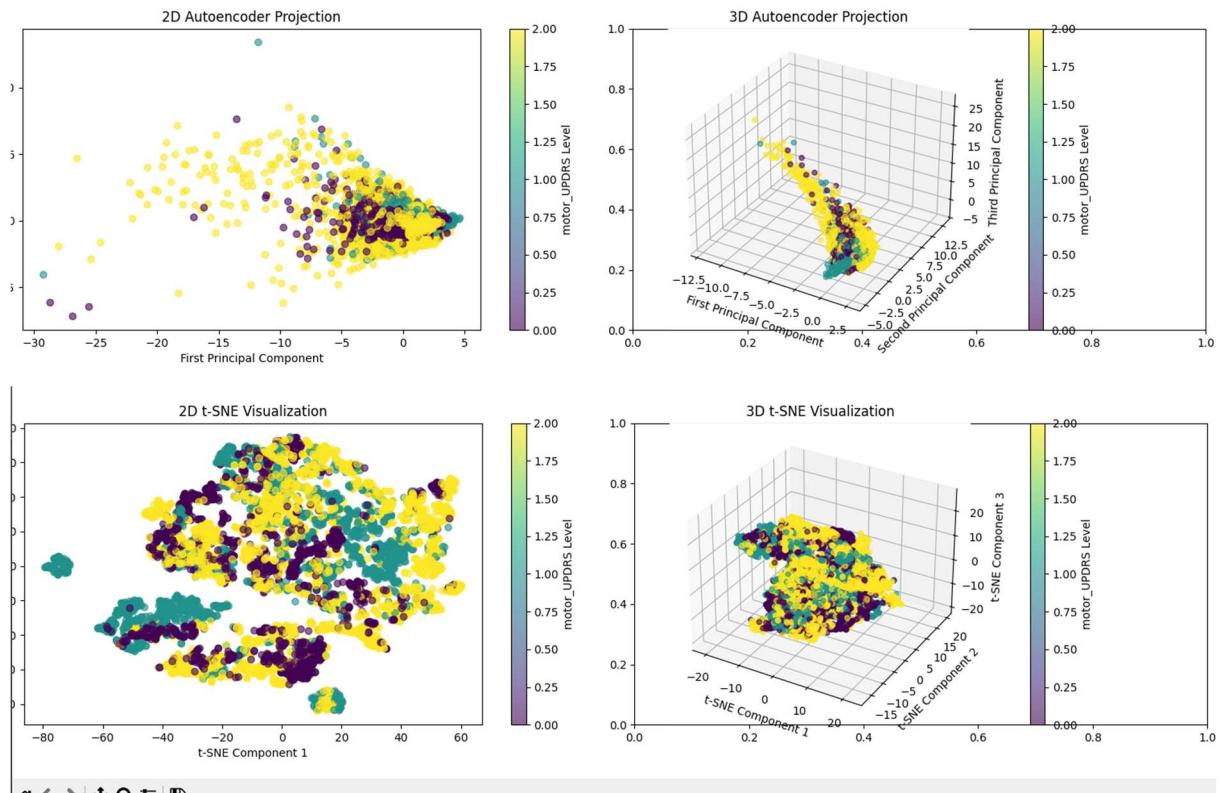
Pretrained Model Predictions

	Model	MAE	MSE	R2
0	Base	5.305245	46.243991	0.275504
1	Pretrained	5.444418	46.501924	0.271463

```

184/184 ━━━━━━━━ 0s 704us/step
184/184 ━━━━━━━━ 0s 717us/step

```



Comparison of training approaches:

Base model final validation MAE: 5.1743

Pretrained model final validation MAE: 5.3655

MAE improvement with pretraining: -2.62%

R2 improvement with pretraining: -1.47%

Выводы по работе:

1. **Предобучение автоэнкодером не улучшило результаты** – базовая модель показала лучшие метрики (MAE: 5.305 vs 5.444, R²: 0.276 vs 0.271)
2. **Автоэнкодерное предобучение может фиксировать субоптимальные веса**, ограничивая дальнейшее обучение регрессора
3. **Визуализации PCA и t-SNE** показали наличие структуры в данных, связанной с целевой переменной
4. Для данного набора биомедицинских данных прямое обучение оказалось более эффективным, чем поэтапное автоэнкодерное предобучение
5. **Качество предсказаний умеренное** ($R^2 \approx 0.27$), что характерно для сложных биомедицинских задач