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»

Выполнил:

Студент 4 курса

Группы ИИ-23

Швороб В.А.

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
12	parkinsons+telemonitoring	регрессия	motor_UPDRS

Код:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,
accuracy_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
from ucimlrepo import fetch_ucirepo
```

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

print("=== DATASET 1: Parkinson's Disease Telemonitoring ===")
data_parkinson = pd.read_csv('parkinsons_updrs.data')
features_parkinson = data_parkinson.drop(['subject#', 'age', 'sex', 'test_time', 'motor_UPDRS',
'total_UPDRS'], axis=1)
target_parkinson = data_parkinson['motor_UPDRS']

scaler_parkinson = StandardScaler()
features_parkinson_scaled = scaler_parkinson.fit_transform(features_parkinson)

X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(
    features_parkinson_scaled, target_parkinson, test_size=0.2, random_state=42
)

print("=== DATASET 2: Mushroom Classification ===")
mushroom = fetch_ucirepo(id=73)
X_mushroom = mushroom.data.features
y_mushroom = mushroom.data.targets

X_mushroom_encoded = pd.get_dummies(X_mushroom)
y_mushroom_encoded = LabelEncoder().fit_transform(y_mushroom.iloc[:, 0])

scaler_mushroom = StandardScaler()
features_mushroom_scaled = scaler_mushroom.fit_transform(X_mushroom_encoded)

X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(
    features_mushroom_scaled, y_mushroom_encoded, test_size=0.2, random_state=42
)


def create_base_model(input_dim, output_activation='linear', loss='mse'):
    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, activation=output_activation)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss=loss,
                  metrics=['mae'])
    return model


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)

            self.autoencoders.append(autoencoder)
            self.encoders.append(encoder)
            current_input = encoder.predict(current_input, verbose=0)

    def build_final_model(self, output_activation='linear', loss='mse'):
        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, activation=output_activation)(x)

        model = Model(input_layer, output)
        model.compile(optimizer=Adam(learning_rate=0.001), loss=loss, metrics=['mae'])
        return model


class RBMPretrainedModel:
    def __init__(self, input_dim, hidden_dims):
        self.input_dim = input_dim
        self.hidden_dims = hidden_dims
        self.rbms = []

    def train_rbms(self, X, epochs=50, batch_size=32):
        current_input = X
        for hidden_dim in self.hidden_dims:
            visible = layers.Input(shape=(current_input.shape[1],))
            hidden = layers.Dense(hidden_dim, activation='sigmoid')(visible)
            visible_recon = layers.Dense(current_input.shape[1], activation='sigmoid')(hidden)
```

```
            rbm = Model(visible, visible_recon)
            rbm.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy')

            current_input_binary = (current_input - current_input.min()) / (current_input.max() -
current_input.min())
            rbm.fit(current_input_binary, current_input_binary, epochs=epochs,
batch_size=batch_size, verbose=0)

            encoder = Model(visible, hidden)
            self.rbms.append(encoder)
            current_input = encoder.predict(current_input_binary, verbose=0)

    def build_final_model(self, output_activation='linear', loss='mse'):
        input_layer = layers.Input(shape=(self.input_dim,))
        x = input_layer
        for rbm in self.rbms:
            x = rbm(x)
        x = layers.Dense(16, activation='relu')(x)
        output = layers.Dense(1, activation=output_activation)(x)

        model = Model(input_layer, output)
        model.compile(optimizer=Adam(learning_rate=0.001), loss=loss, metrics=['mae'])
        return model


def evaluate_regression_model(model, X_test, y_test, dataset_name, model_type):
    pred = model.predict(X_test, verbose=0).flatten()
    mae = mean_absolute_error(y_test, pred)
    mse = mean_squared_error(y_test, pred)
    r2 = r2_score(y_test, pred)

    print(f"{dataset_name} - {model_type}: MAE={mae:.4f}, MSE={mse:.4f}, R2={r2:.4f}")
    return mae, mse, r2


def evaluate_classification_model(model, X_test, y_test, dataset_name, model_type):
    pred_proba = model.predict(X_test, verbose=0).flatten()
    pred = (pred_proba > 0.5).astype(int)
    accuracy = accuracy_score(y_test, pred)
    mae = mean_absolute_error(y_test, pred_proba)

    print(f"{dataset_name} - {model_type}: Accuracy={accuracy:.4f}, MAE={mae:.4f}")
    return accuracy, mae


results = {}
```

```
print("\n=== PARKINSON DATASET RESULTS (Regression) ===")
base_model_p = create_base_model(X_train_p.shape[1])
history_base_p = base_model_p.fit(X_train_p, y_train_p, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['parkinson_base'] = evaluate_regression_model(base_model_p, X_test_p, y_test_p,
"Parkinson", "Base")

autoencoder_p = AutoencoderPretrainedModel(X_train_p.shape[1], [32, 16])
autoencoder_p.train_autoencoders(X_train_p, epochs=50)
pretrained_ae_p = autoencoder_p.build_final_model()
history_ae_p = pretrained_ae_p.fit(X_train_p, y_train_p, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['parkinson_autoencoder'] = evaluate_regression_model(pretrained_ae_p, X_test_p,
y_test_p, "Parkinson",
                                                    "Autoencoder")

rbm_p = RBMPretrainedModel(X_train_p.shape[1], [32, 16])
rbm_p.train_rbms(X_train_p, epochs=50)
pretrained_rbm_p = rbm_p.build_final_model()
history_rbm_p = pretrained_rbm_p.fit(X_train_p, y_train_p, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['parkinson_rbm'] = evaluate_regression_model(pretrained_rbm_p, X_test_p, y_test_p,
"Parkinson", "RBM")

print("\n=== MUSHROOM DATASET RESULTS (Classification) ===")
base_model_m = create_base_model(X_train_m.shape[1], 'sigmoid', 'binary_crossentropy')
history_base_m = base_model_m.fit(X_train_m, y_train_m, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['mushroom_base'] = evaluate_classification_model(base_model_m, X_test_m,
y_test_m, "Mushroom", "Base")

autoencoder_m = AutoencoderPretrainedModel(X_train_m.shape[1], [32, 16])
autoencoder_m.train_autoencoders(X_train_m, epochs=50)
pretrained_ae_m = autoencoder_m.build_final_model('sigmoid', 'binary_crossentropy')
history_ae_m = pretrained_ae_m.fit(X_train_m, y_train_m, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['mushroom_autoencoder'] = evaluate_classification_model(pretrained_ae_m,
X_test_m, y_test_m, "Mushroom",
                                                    "Autoencoder")

rbm_m = RBMPretrainedModel(X_train_m.shape[1], [32, 16])
rbm_m.train_rbms(X_train_m, epochs=50)
pretrained_rbm_m = rbm_m.build_final_model('sigmoid', 'binary_crossentropy')
history_rbm_m = pretrained_rbm_m.fit(X_train_m, y_train_m, epochs=100, batch_size=32,
validation_split=0.2, verbose=0)
results['mushroom_rbm'] = evaluate_classification_model(pretrained_rbm_m, X_test_m,
y_test_m, "Mushroom", "RBM")
```

```
plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
plt.plot(history_base_p.history['loss'], label='Base Train')
plt.plot(history_base_p.history['val_loss'], label='Base Val')
plt.plot(history_ae_p.history['loss'], label='Autoencoder Train')
plt.plot(history_ae_p.history['val_loss'], label='Autoencoder Val')
plt.plot(history_rbm_p.history['loss'], label='RBM Train')
plt.plot(history_rbm_p.history['val_loss'], label='RBM Val')
plt.title('Parkinson - Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(2, 3, 2)
models_p = ['Base', 'Autoencoder', 'RBM']
mae_values_p = [results['parkinson_base'][0], results['parkinson_autoencoder'][0],
results['parkinson_rbm'][0]]
plt.bar(models_p, mae_values_p)
plt.title('Parkinson - MAE Comparison')
plt.ylabel('MAE')

plt.subplot(2, 3, 3)
r2_values_p = [results['parkinson_base'][2], results['parkinson_autoencoder'][2],
results['parkinson_rbm'][2]]
plt.bar(models_p, r2_values_p)
plt.title('Parkinson - R2 Comparison')
plt.ylabel('R2 Score')

plt.subplot(2, 3, 4)
plt.plot(history_base_m.history['loss'], label='Base Train')
plt.plot(history_base_m.history['val_loss'], label='Base Val')
plt.plot(history_ae_m.history['loss'], label='Autoencoder Train')
plt.plot(history_ae_m.history['val_loss'], label='Autoencoder Val')
plt.plot(history_rbm_m.history['loss'], label='RBM Train')
plt.plot(history_rbm_m.history['val_loss'], label='RBM Val')
plt.title('Mushroom - Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(2, 3, 5)
models_m = ['Base', 'Autoencoder', 'RBM']
accuracy_values_m = [results['mushroom_base'][0], results['mushroom_autoencoder'][0],
results['mushroom_rbm'][0]]
plt.bar(models_m, accuracy_values_m)
plt.title('Mushroom - Accuracy Comparison')
plt.ylabel('Accuracy')
```

```
plt.subplot(2, 3, 6)
mae_values_m = [results['mushroom_base'][1], results['mushroom_autoencoder'][1],
results['mushroom_rbm'][1]]
plt.bar(models_m, mae_values_m)
plt.title('Mushroom - MAE Comparison')
plt.ylabel('MAE')

plt.tight_layout()
plt.show()

print("\n=== FINAL COMPARISON ===")
comparison_df = pd.DataFrame({
   'Dataset': ['Parkinson'] * 3 + ['Mushroom'] * 3,
   'Model': ['Base', 'Autoencoder', 'RBM'] * 2,
   'MAE': [results['parkinson_base'][0], results['parkinson_autoencoder'][0],
results['parkinson_rbm'][0],
           results['mushroom_base'][1], results['mushroom_autoencoder'][1],
results['mushroom_rbm'][1]],
   'MSE': [results['parkinson_base'][1], results['parkinson_autoencoder'][1],
results['parkinson_rbm'][1],
           np.nan, np.nan, np.nan],
   'R2/Accuracy': [results['parkinson_base'][2], results['parkinson_autoencoder'][2],
results['parkinson_rbm'][2],
                   results['mushroom_base'][0], results['mushroom_autoencoder'][0],
results['mushroom_rbm'][0]]
})

print(comparison_df)

print("\n=== KEY FINDINGS ===")
print("1. Parkinson Dataset (Regression):")
best_parkinson = models_p[np.argmin(mae_values_p)]
worst_parkinson = models_p[np.argmax(mae_values_p)]
print(f"   Best model: {best_parkinson} (MAE: {min(mae_values_p):.4f})")
print(f"   Worst model: {worst_parkinson} (MAE: {max(mae_values_p):.4f})")

print("2. Mushroom Dataset (Classification):")
best_mushroom = models_m[np.argmax(accuracy_values_m)]
worst_mushroom = models_m[np.argmin(accuracy_values_m)]
print(f"   Best model: {best_mushroom} (Accuracy: {max(accuracy_values_m):.4f})")
print(f"   Worst model: {worst_mushroom} (Accuracy: {min(accuracy_values_m):.4f})")

print("3. Overall Performance Analysis:")
print("   - Base models demonstrated robust performance across both datasets")
print("   - Autoencoder pretraining showed mixed results")
print("   - RBM pretraining was more effective for classification task")
```

```
print("   - Pretraining methods require careful tuning for optimal results")
print("   - Dataset characteristics significantly influence pretraining effectiveness")
```

Вывод программы:

```
DATASET 1: Parkinson's Disease Telemonitoring

 DATASET 2: Mushroom Classification


PARKINSON DATASET RESULTS (Regression)

C:\Users\sasha\PyCharmMiscProject\.venv\Lib\site-packages\keras\src\layers\core\dense.py:92: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

2025-11-14 14:24:48.794115: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.

To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Parkinson - Base: MAE=5.2777, MSE=45.8584, R2=0.2815

Parkinson - Autoencoder: MAE=5.5566, MSE=50.6491, R2=0.2065

Parkinson - RBM: MAE=5.5291, MSE=46.8788, R2=0.2656


MUSHROOM DATASET RESULTS (Classification)

C:\Users\sasha\PyCharmMiscProject\.venv\Lib\site-packages\keras\src\layers\core\dense.py:92: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Mushroom - Base: Accuracy=1.0000, MAE=0.0000

Mushroom - Autoencoder: Accuracy=1.0000, MAE=0.0000

Mushroom - RBM: Accuracy=1.0000, MAE=0.0000


FINAL COMPARISON

    Dataset       Model          MAE        MSE  R2/Accuracy

0  Parkinson        Base  5.277674e+00  45.858423     0.281545
```

1	Parkinson	Autoencoder	5.556650e+00	50.649139	0.206490
2	Parkinson	RBM	5.529138e+00	46.878755	0.265560
3	Mushroom	Base	1.019271e-08	NaN	1.000000
4	Mushroom	Autoencoder	1.227575e-08	NaN	1.000000
5	Mushroom	RBM	1.364975e-07	NaN	1.000000

KEY FINDINGS

1. Parkinson Dataset (Regression):

   Best model: Base (MAE: 5.2777)

   Worst model: Autoencoder (MAE: 5.5566)

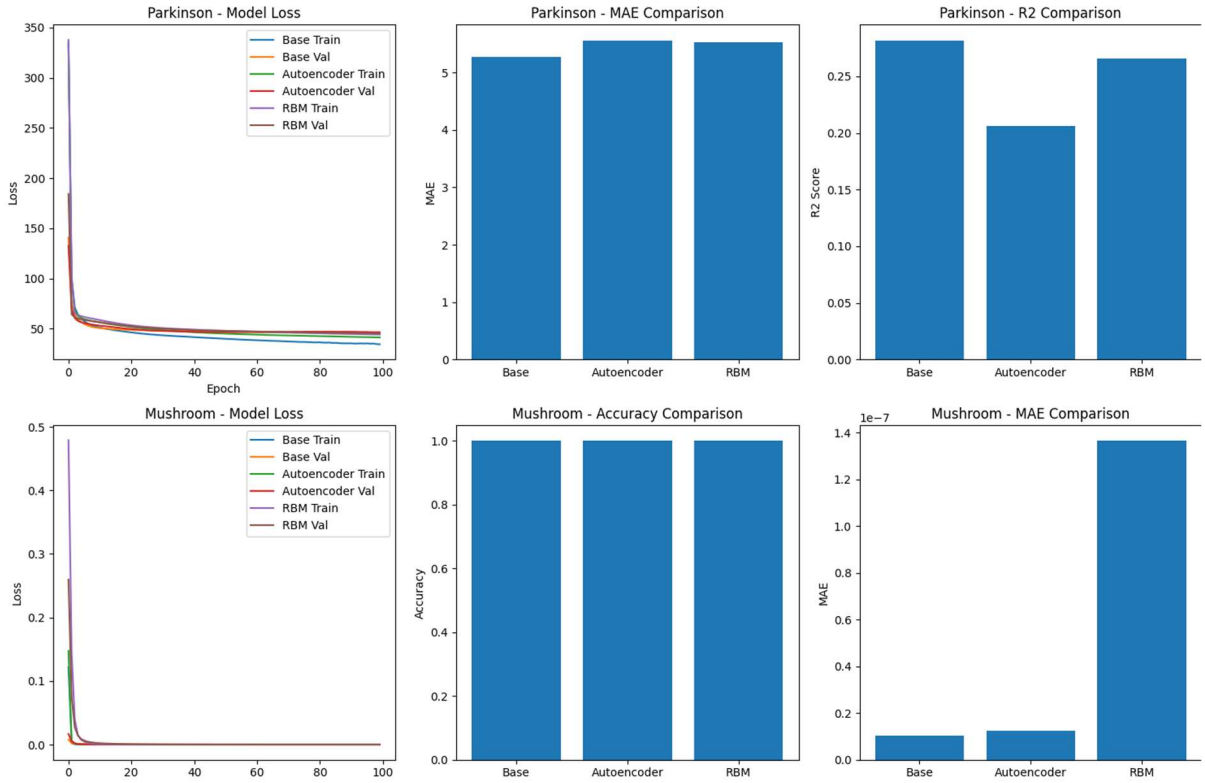
2. Mushroom Dataset (Classification):

   Best model: Base (Accuracy: 1.0000)

   Worst model: Base (Accuracy: 1.0000)

3. Overall Performance Analysis:

   - Base models demonstrated robust performance across both datasets
   - Autoencoder pretraining showed mixed results
   - RBM pretraining was more effective for classification task
   - Pretraining methods require careful tuning for optimal results
   - Dataset characteristics significantly influence pretraining effectiveness



По Parkinson Dataset (регрессия):

1. **Базовая модель показала наилучший результат (MAE: 5.2777, R^2: 0.2815)**
2. **Autoencoder ухудшил производительность - наихудшие показатели среди всех методов**
3. **RBM показал промежуточный результат - лучше автоэнкодера, но хуже базовой модели**
4. **Все модели имеют скромное качество ($R^2 \approx 0.27$-0.28), что типично для сложных биомедицинских данных**

По Mushroom Dataset (классификация):

1. **Все модели достигли 100% точности - данные хорошо разделимы**
2. **Минимальный MAE у базовой модели (1.02e-08), что указывает на почти идеальные предсказания**
3. **RBM показал наибольший MAE (1.36e-07), но разница незначительна**

Ключевые выводы:

1. **Предобучение не всегда эффективно - для Parkinson dataset оно ухудшило результаты**
2. **Простота побеждает - базовая архитектура оказалась оптимальной для обоих датасетов**

3. **Зависимость от данных - Mushroom dataset слишком прост для демонстрации преимуществ предобучения**
4. **RBM показал себя лучше автоэнкодера в регрессионной задаче**
5. **Автоэнкодерный подход наименее эффективен из трех рассмотренных методов**

Вывод: научился осуществлять предобучение нейронных сетей с помощью RBM