Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: « PCA и автоэнкодеры»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2025

Цель работы: В рамках данной работы необходимо реализовать PCA и автоэнкодер и сравнить время и качество их обучения(для сравнения использовать соответствующие метрики).  
Для реализации автоэнкодера необходимо использовать ML-фреймворки.

**Ход работы**

Изображение выглядит как снимок экрана, Красочность, текст, диаграмма

Автоматически созданное описаниеPCA.py:

import numpy as np  
  
class PCA:  
 def \_\_init\_\_(self, n\_components):  
 self.n\_components = n\_components  
 self.components = None  
 self.mean = None  
  
 def fit(self, X):  
 self.mean = np.mean(X, axis=0)  
 X\_centered = X - self.mean  
 cov\_matrix = np.cov(X\_centered, rowvar=False)  
 eigenvalues, eigenvectors = np.linalg.eigh(cov\_matrix)  
 idx = np.argsort(eigenvalues)[::-1]  
 eigenvectors = eigenvectors[:, idx]  
 self.components = eigenvectors[:, :self.n\_components]  
  
 def transform(self, X):   
 X\_centered = X - self.mean  
 return np.dot(X\_centered, self.components)  
  
 def fit\_transform(self, X):  
 self.fit(X)  
 return self.transform(X)  
  
 def inverse\_transform(self, X):  
 return np.dot(X, self.components.T) + self.mean

Изображение выглядит как текст, снимок экрана, Красочность, диаграмма

Автоматически созданное описаниеautoencoder.py:

Изображение выглядит как линия, текст, График, диаграмма

Автоматически созданное описаниеimport torch  
import torch.nn as nn  
import torch.optim as optim  
import matplotlib.pyplot as plt  
class Autoencoder(nn.Module):  
 def \_\_init\_\_(self, input\_dim, encoding\_dim):  
 super(Autoencoder, self).\_\_init\_\_()  
 self.encoder = nn.Sequential(  
 nn.Linear(input\_dim, 64),  
 nn.ReLU(),  
 nn.Linear(64, encoding\_dim),  
 nn.ReLU()  
 )  
 self.decoder = nn.Sequential(  
 nn.Linear(encoding\_dim, 64),  
 nn.ReLU(),  
 nn.Linear(64, input\_dim),  
 nn.Sigmoid()  
 )  
 def forward(self, x):  
 encoded = self.encoder(x)  
 decoded = self.decoder(encoded)  
 return decoded  
def train\_autoencoder(model, train\_loader, epochs=50, lr=0.001):  
 criterion = nn.MSELoss()  
 optimizer = optim.Adam(model.parameters(), lr=lr)  
 loss\_history = []  
 for epoch in range(epochs):  
 epoch\_loss = 0  
 batch\_count = 0  
 for batch in train\_loader:  
 inputs = batch[0]  
 optimizer.zero\_grad()  
 outputs = model(inputs)  
 loss = criterion(outputs, inputs)  
 loss.backward()  
 optimizer.step()  
 epoch\_loss += loss.item()  
 batch\_count += 1  
 avg\_loss = epoch\_loss / batch\_count  
 loss\_history.append(avg\_loss)  
 print(f'Epoch [{epoch + 1}/{epochs}], Loss: {avg\_loss:.4f}')  
 plt.plot(range(1, epochs + 1), loss\_history, marker='o', linestyle='-')  
 plt.xlabel('Epoch')  
 plt.ylabel('Loss')  
 plt.title('Training Loss Over Epochs')  
 plt.grid()  
 plt.show()



**Вывод:** в ходе лабораторной работы я научился реализовывать PCA и автоэнкодеры.