Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №2**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Альтернативные методы обучения автоэнкодеров»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2025

Цель работы: В рамках данной работы необходимо реализовать и обучить автоэнкодер. В качестве алгоритмов оптимизации использовать правило Ойя, кумулятивное дельта правило, SGD, Adam.

**Ход работы**

Autoencoder.py:  
class Autoencoder:  
 def \_\_init\_\_(self, input\_dim, encoding\_dim, learning\_rate, alpha=0.0, optimizer\_type="cumulative"):  
 self.input\_dim = input\_dim  
 self.encoding\_dim = encoding\_dim  
 self.learning\_rate = learning\_rate  
 self.alpha = alpha  
 self.optimizer\_type = optimizer\_type  
 self.weights\_input\_encoding = np.random.randn(input\_dim, encoding\_dim)  
 self.weights\_encoding\_input = np.random.randn(encoding\_dim, input\_dim)  
 self.bias\_encoding = np.random.randn(1, encoding\_dim)  
 self.bias\_input = np.random.randn(1, input\_dim)  
 self.prev\_delta\_weights\_input\_encoding = np.zeros((input\_dim, encoding\_dim))  
 self.prev\_delta\_weights\_encoding\_input = np.zeros((encoding\_dim, input\_dim))  
 self.prev\_delta\_bias\_encoding = np.zeros((1, encoding\_dim))  
 self.prev\_delta\_bias\_input = np.zeros((1, input\_dim))  
 self.train\_errors = []  
 self.test\_errors = []  
 if optimizer\_type == "adam":  
 self.model = nn.Sequential(  
 nn.Linear(input\_dim, encoding\_dim),  
 nn.Sigmoid(),  
 nn.Linear(encoding\_dim, input\_dim),  
 nn.Sigmoid()  
 )  
 self.criterion = nn.MSELoss()  
 self.optimizer = optim.Adam(self.model.parameters(), lr=learning\_rate)  
 @staticmethod  
 def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
 @staticmethod  
 def sigmoid\_derivative(x):  
 return x \* (1 - x)  
 def encode(self, inputs):  
 self.encoding\_layer\_input = np.dot(inputs, self.weights\_input\_encoding) + self.bias\_encoding  
 self.encoding\_layer\_output = self.sigmoid(self.encoding\_layer\_input)  
 def decode(self, encoding\_output):  
 self.input\_layer\_input = np.dot(encoding\_output, self.weights\_encoding\_input) + self.bias\_input  
 self.input\_layer\_output = self.sigmoid(self.input\_layer\_input)  
 def backward\_pass(self, inputs):  
 if self.optimizer\_type == "adam":  
 inputs\_tensor = torch.tensor(inputs, dtype=torch.float32)  
 outputs = self.model(inputs\_tensor)  
 loss = self.criterion(outputs, inputs\_tensor)  
 self.optimizer.zero\_grad()  
 loss.backward()  
 self.optimizer.step()  
 self.train\_errors.append(loss.item())  
 return  
 encoding\_error = inputs - self.input\_layer\_output  
 encoding\_delta = encoding\_error \* self.sigmoid\_derivative(self.input\_layer\_output)  
 if self.optimizer\_type == "oja":  
 self.weights\_encoding\_input += self.learning\_rate \* np.dot(self.encoding\_layer\_output.T, encoding\_delta) / inputs.shape[0]  
 self.bias\_input += self.learning\_rate \* np.mean(encoding\_delta, axis=0, keepdims=True)  
 decoding\_error = encoding\_delta.dot(self.weights\_encoding\_input.T)  
 decoding\_delta = decoding\_error \* self.sigmoid\_derivative(self.encoding\_layer\_output)  
 for i in range(self.encoding\_dim):  
 delta\_w = self.learning\_rate \* (self.encoding\_layer\_output[:, i][:, np.newaxis] \* (  
 inputs - self.encoding\_layer\_output[:, i][:, np.newaxis] \* self.weights\_input\_encoding[:, i])).mean(axis=0)  
 self.weights\_input\_encoding[:, i] += delta\_w  
 self.bias\_encoding += self.learning\_rate \* np.mean(decoding\_delta, axis=0, keepdims=True)  
 elif self.optimizer\_type == "cumulative":  
 delta\_weights\_encoding\_input = (  
 self.learning\_rate \* np.dot(self.encoding\_layer\_output.T, encoding\_delta) / inputs.shape[0]  
Изображение выглядит как линия, График, текст, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки. + self.alpha \* self.prev\_delta\_weights\_encoding\_input  
 )  
 delta\_bias\_input = (  
 self.learning\_rate \* np.mean(encoding\_delta, axis=0, keepdims=True)  
 + self.alpha \* self.prev\_delta\_bias\_input  
 )  
 self.weights\_encoding\_input += delta\_weights\_encoding\_input  
 self.bias\_input += delta\_bias\_input  
 decoding\_error = encoding\_delta.dot(self.weights\_encoding\_input.T)  
 decoding\_delta = decoding\_error \* self.sigmoid\_derivative(self.encoding\_layer\_output)  
 delta\_weights\_input\_encoding = (  
 self.learning\_rate \* np.dot(inputs.T, decoding\_delta) / inputs.shape[0]  
 + self.alpha \* self.prev\_delta\_weights\_input\_encoding  
 )  
 delta\_bias\_encoding = (  
 self.learning\_rate \* np.mean(decoding\_delta, axis=0, keepdims=True)  
 + self.alpha \* self.prev\_delta\_bias\_encoding  
 )  
 self.weights\_input\_encoding += delta\_weights\_input\_encoding  
 self.bias\_encoding += delta\_bias\_encoding  
 # Store current deltas for next iteration  
 self.prev\_delta\_weights\_input\_encoding = delta\_weights\_input\_encoding  
 self.prev\_delta\_weights\_encoding\_input = delta\_weights\_encoding\_input  
 self.prev\_delta\_bias\_encoding = delta\_bias\_encoding  
 self.prev\_delta\_bias\_input = delta\_bias\_input  
 elif self.optimizer\_type == "sgd":  
Изображение выглядит как линия, График, текст, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки. self.weights\_encoding\_input += self.learning\_rate \* np.dot(self.encoding\_layer\_output.T, encoding\_delta) / inputs.shape[0]  
 self.bias\_input += self.learning\_rate \* np.mean(encoding\_delta, axis=0, keepdims=True)  
 decoding\_error = encoding\_delta.dot(self.weights\_encoding\_input.T)  
 decoding\_delta = decoding\_error \* self.sigmoid\_derivative(self.encoding\_layer\_output)  
 self.weights\_input\_encoding += self.learning\_rate \* np.dot(inputs.T, decoding\_delta) / inputs.shape[0]  
 self.bias\_encoding += self.learning\_rate \* np.mean(decoding\_delta, axis=0, keepdims=True)  
 mse = np.mean((inputs - self.input\_layer\_output) \*\* 2)  
 self.train\_errors.append(mse)  
 def train(self, train\_data, test\_data, epochs=10):  
 for epoch in range(epochs):

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки. if self.optimizer\_type == "adam":  
 self.backward\_pass(train\_data)  
 else:  
 self.encode(train\_data)  
 self.decode(self.encoding\_layer\_output)  
 self.backward\_pass(train\_data)  
 if self.optimizer\_type == "adam":  
 test\_inputs\_tensor = torch.tensor(test\_data, dtype=torch.float32)  
 test\_outputs = self.model(test\_inputs\_tensor)  
 test\_loss = self.criterion(test\_outputs, test\_inputs\_tensor)  
 self.test\_errors.append(test\_loss.item())  
 else:  
 self.encode(test\_data)  
 self.decode(self.encoding\_layer\_output)  
 test\_mse = np.mean((test\_data - self.input\_layer\_output) \*\* 2)  
 self.test\_errors.append(test\_mse)  
 if (epoch + 1) % 100 == 0:  
 print(f"Epoch {epoch + 1}/{epochs}, Train MSE: {self.train\_errors[-1]:.4f}, Test MSE: {self.test\_errors[-1]:.4f}")

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