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-21

Литвинюк Т. В.

**Проверил:**

Туз И. С.

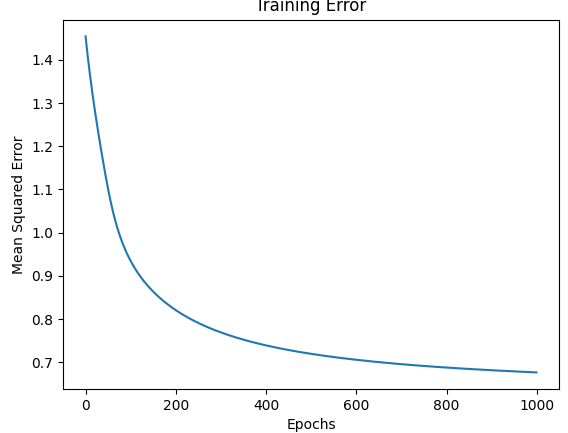
Брест 2024

**Цель:** реализовать и обучить автоэнкодер.

**Ход работы**

**Вариант 7**

****

**OJA:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import StandardScaler

class Autoencoder():

def \_\_init\_\_(self, input\_neuron, encoding\_neuron, learning\_rate):

self.input\_neuron = input\_neuron

self.encoding\_neuron = encoding\_neuron

self.learning\_rate = learning\_rate

self.weights\_input\_encoding = np.random.rand(input\_neuron, encoding\_neuron)

self.weights\_encoding\_input = np.random.rand(encoding\_neuron, input\_neuron)

self.bias\_encoding = np.random.rand(1, encoding\_neuron)

self.bias\_input = np.random.rand(1, input\_neuron)

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(self, 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):

# Кодирование ошибки и обновление весов/смещений для декодирующего слоя

encoding\_error = inputs - self.input\_layer\_output

encoding\_delta = encoding\_error \* self.sigmoid\_derivative(self.input\_layer\_output)

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)

# Правильное обновление с использованием правила Ойи

# Необходимо выполнить операцию для каждого нейрона в кодирующем слое

for i in range(self.encoding\_neuron):

# Корректировка выражения для oja\_delta

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)

# Вычисление и сохранение среднеквадратичной ошибки

mse = np.mean((inputs - self.input\_layer\_output) \*\* 2)

self.errors.append(mse)

def train(self, inputs, epochs=10):

self.errors = []

for \_ in range(epochs):

self.encode(inputs)

self.decode(self.encoding\_layer\_output)

self.backward\_pass(inputs)

def encode\_data(self, inputs\_data):

self.encode(inputs\_data)

return self.encoding\_layer\_output

def decode\_data(self, encoded\_data):

self.decode(encoded\_data)

return self.input\_layer\_output

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv(r"MRZIS/lab2/Seed\_data.csv", header=None)

X = data.iloc[:, :-1].values

scaler = StandardScaler()

X = scaler.fit\_transform(X)

autoencoder = Autoencoder(input\_neuron=X.shape[1], encoding\_neuron=5, learning\_rate=0.093)

autoencoder.train(X, epochs=1000)

X\_encode\_data = autoencoder.encode\_data(X)

X\_decode\_data = autoencoder.decode\_data(X\_encode\_data)

print("Mean Squared Error:", np.mean((X - X\_decode\_data) \*\* 2))

# Построение графика ошибок

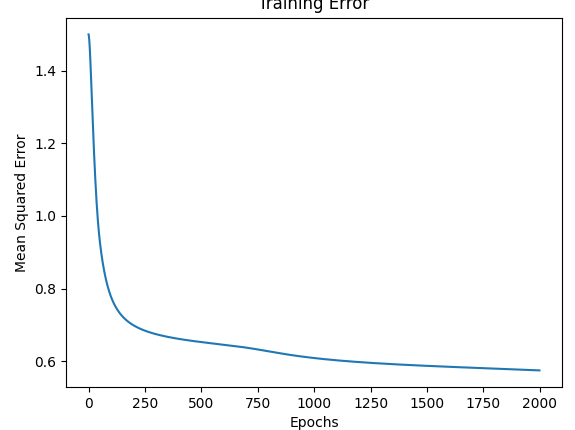
plt.plot(autoencoder.errors)

plt.xlabel('Epochs')

plt.ylabel('Mean Squared Error')

plt.title('Training Error')

plt.show()

Кумулятивное дельта правило:

from matplotlib import pyplot as plt

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

import pandas as pd

class Autoencoder():

    def \_\_init\_\_(*self*, *input\_neuron*, *encoding\_neuron*, *learning\_rate*, *alpha*):

        self.input\_neuron = input\_neuron

        self.encoding\_neuron = encoding\_neuron

        self.learning\_rate = learning\_rate

        self.alpha = alpha

        self.weights\_input\_encoding = np.random.rand(input\_neuron, encoding\_neuron)

        self.weights\_encoding\_input = np.random.rand(encoding\_neuron, input\_neuron)

        self.bias\_encoding = np.random.rand(1, encoding\_neuron)

        self.bias\_input = np.random.rand(1, input\_neuron)

        self.prev\_delta\_weights\_input\_encoding = np.zeros((input\_neuron, encoding\_neuron))

        self.prev\_delta\_weights\_encoding\_input = np.zeros((encoding\_neuron, input\_neuron))

        self.prev\_delta\_bias\_encoding = np.zeros((1, encoding\_neuron))

        self.prev\_delta\_bias\_input = np.zeros((1, input\_neuron))

        self.errors = []

    def sigmoid(*self*, *x*):

        return 1 / (1 + np.exp(-x))

    def sigmoid\_derivative(*self*, *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*):

        encoding\_error = inputs - self.input\_layer\_output

        encoding\_delta = encoding\_error \* self.sigmoid\_derivative(self.input\_layer\_output)

        # Обновление весов от кодировщика к декодировщику

        delta\_weights\_encoding\_input = self.learning\_rate \* np.dot(self.encoding\_layer\_output.T, encoding\_delta) / inputs.shape[0] + 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

        # Обновление весов от декодировщика к кодировщику

        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

        # Сохранение текущих изменений весов для следующей итерации

        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

        mse = np.mean((inputs - self.input\_layer\_output) \*\* 2)

        self.errors.append(mse)

    def train(*self*, *inputs*, *epochs*=10):

        for \_ in range(epochs):

            self.encode(inputs)

            self.decode(self.encoding\_layer\_output)

            self.backward\_pass(inputs)

    def encode\_data(*self*, *inputs\_data*):

        self.encode(inputs\_data)

        return self.encoding\_layer\_output

    def decode\_data(*self*, *encoded\_data*):

        self.decode(encoded\_data)

        return self.input\_layer\_output

if \_\_name\_\_ == "\_\_main\_\_":

    data = pd.read\_csv(r"MRZIS/lab2/Seed\_data.csv", *header*=None)

    X = data.iloc[:, :-1].values

    scaler = StandardScaler()

    X = scaler.fit\_transform(X)

    autoencoder = Autoencoder(*input\_neuron*=X.shape[1], *encoding\_neuron*=4, *learning\_rate*=0.03, *alpha*=0.9)

    autoencoder.train(X, *epochs*=2000)

    X\_encode\_data = autoencoder.encode\_data(X)

    X\_decode\_data = autoencoder.decode\_data(X\_encode\_data)

    print(mean\_squared\_error(X, X\_decode\_data))

    plt.plot(autoencoder.errors)

    plt.xlabel('Epochs')

    plt.ylabel('Mean Squared Error')

    plt.title('Training Error')

    plt.show()

Таким образом, кумулятивное дельта правило работает с меньшей ошибкой.

**Вывод:** в ходе лабораторной работы я реализовал два метода обучения автоенкодера.