Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №4**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «MLP. Задача исключающего или»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2024

**Цель:** Написать нейронную сеть (multilayer perceptron с одним скрытым слоем) решающую задачу исключающее или. Датасет сгенерировать самостоятельно. Использовать Сигмоидные функции активации

В обучении использовать алгоритм обратного распространения ошибки и batch\_size > 1

**Ход работы**

**Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание**import numpy as np  
  
class MLP:  
 def \_\_init\_\_(self, input\_size, hide\_size, output\_size, alpha):  
 self.input\_size = input\_size  
 self.output\_size = output\_size  
 self.hide\_size = hide\_size  
 self.Wh = np.random.uniform(size=(input\_size, hide\_size))  
 self.Wo = np.random.uniform(size=(hide\_size, output\_size))  
 self.Th = np.random.uniform(size=(1, hide\_size))  
 self.To = np.random.uniform(size=(1, output\_size))  
 self.alpha = alpha  
  
 def sigmoid(self, x):  
 return 1 / (1 + np.exp(-x))  
  
 def sigmoid\_derivative(self, x):  
 return x \* (1 - x)  
  
 def train\_online(self,x,e):  
 #forward  
 yh = self.sigmoid(np.dot(x,self.Wh)+self.Th)  
 yo = self.sigmoid(np.dot(yh,self.Wo)+self.To)  
 #back  
 error = e - yo  
 deltao = error\*self.sigmoid\_derivative(yo)  
 hidden\_error = deltao.dot(self.Wo.T)  
 deltah = hidden\_error\*self.sigmoid\_derivative(yh)  
 #upd  
 self.Wo += yh.T.dot(deltao) \* self.alpha  
 self.To += np.sum(deltao) \* self.alpha  
 self.Wh += x.T.dot(deltah) \* self.alpha  
 self.Th += np.sum(deltah) \* self.alpha  
  
 def train(self, x, e, batch\_size):  
 num\_samples = x.shape[0]  
  
 for start in range(0, num\_samples, batch\_size):  
 end = start + batch\_size  
 x\_batch = x[start:end]  
 e\_batch = e[start:end]  
  
 yh = self.sigmoid(np.dot(x\_batch, self.Wh) + self.Th)  
 yo = self.sigmoid(np.dot(yh, self.Wo) + self.To)  
  
 error = e\_batch - yo  
 deltao = error \* self.sigmoid\_derivative(yo)  
 hidden\_error = deltao.dot(self.Wo.T)  
 deltah = hidden\_error \* self.sigmoid\_derivative(yh)  
  
 self.Wo += yh.T.dot(deltao) \* self.alpha / batch\_size  
 self.To += np.sum(deltao, axis=0, keepdims=True) \* self.alpha / batch\_size  
 self.Wh += x\_batch.T.dot(deltah) \* self.alpha / batch\_size  
 self.Th += np.sum(deltah, axis=0, keepdims=True) \* self.alpha / batch\_size  
  
 def test(self, x, e):  
 yh = self.sigmoid(np.dot(x, self.Wh) + self.Th)  
 yo = self.sigmoid(np.dot(yh, self.Wo) + self.To)  
 print(yo)  
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  
y = np.array([[0], [1], [1], [0]])  
model = MLP(input\_size=2, hide\_size=2, output\_size=1, alpha=0.2)  
batch\_size = 2   
for epoch in range(10000):  
 model.train(x, y, batch\_size)  
model.test(x, y

**Вывод:** в ходе лабораторной работы я научился реализовывать однослойную нейронную сеть.