Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «SLP. Классификация»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Макаревич Н.Р.

**Проверил:**

Туз И. С.

Брест 2024

**Цель работы**: Изучить обучение и функционирование SLP при решении задач классификации.

Написать нейронную сеть(single layer perceptron) для решения задачи классификации.

Обучить сеть с использованием константного и адаптивного шага обучения, online-

learning и batch-learning

**y = -2x + 12**

**Ход работы:**

Код нейронной сети:

class SimpleNN:

def \_\_init\_\_(self, x\_len, y\_len):

self.X = np.zeros(x\_len)

self.Y = np.zeros(y\_len)

self.W = np.random.rand(x\_len, y\_len)

self.T = np.zeros(y\_len)

self.alp: float = 0.5

self.adaptive\_flag: bool = False

self.errors\_for\_chart = np.array([])

def forward\_prop(self, image):

self.X = image

return np.dot(image, self.W) - self.T

def activation\_func(self, weighted\_sum):

return 1 / (1 + math.exp(-weighted\_sum))

def activation\_derivative(self, weighted\_sum):

sig = self.activation\_func(weighted\_sum)

return sig \* (1 - sig)

def adjust(self, error, weighted\_sum):

for i in range(self.W.shape[0]):

for j in range(self.W.shape[1]):

self.W[i, j] -= self.alp \* error \* self.X[i] \* self.activation\_derivative(weighted\_sum)

self.T[j] += self.alp \* error \* self.activation\_derivative(weighted\_sum)

def adaptive\_online(self, image):

self.alp = self.activation\_derivative(self.forward\_prop(image)) / (1 + np.sum(image \*\* 2))

def adaptive\_batch(self, batch\_data, batch\_e):

dividend = 0

divider = 0

a = self.helper\_a(batch\_data, batch\_e)

for ind, image in enumerate(batch\_data):

activation = self.activation\_func(self.forward\_prop(image))

delta = activation - batch\_e[ind]

dividend += delta \* activation \* (1 - activation) \* a

divider += a \*\* 2

self.alp = 3 \* dividend / divider

return self.alp

def helper\_a(self, batch\_data, batch\_e):

a = 0

for ind, image in enumerate(batch\_data):

activation = self.activation\_func(self.forward\_prop(image))

a += (activation - batch\_e[ind]) \* (1 + np.sum(image \*\* 2))

return a

def train\_online(self, train\_data, train\_e, test\_data, test\_e):

max\_epoch: int = 100

for epoch in range(max\_epoch):

ind: int = 0

for image in train\_data:

Y = self.forward\_prop(image)

prediction = self.activation\_func(Y)

error = prediction - train\_e[ind]

if self.adaptive\_flag:

self.adaptive\_online(image)

self.adjust(error, Y)

ind += 1

if self.test(test\_data, test\_e):

print(f"Network trained in {epoch} epoches")

self.drawErrorGraph()

break

def test(self, test\_data, test\_e) -> bool:

counter: float = 0

for ind, image in enumerate(test\_data):

Y = self.activation\_func(self.forward\_prop(image))

if (Y >= 0.5 and test\_e[ind] == 1) or (Y < 0.5 and test\_e[ind] == 0):

counter+=1

right\_percent = counter / len(test\_e) \* 100

print(f"Percentage: {right\_percent}")

self.errors\_for\_chart = np.append(self.errors\_for\_chart, right\_percent)

return right\_percent > 95

def train\_batch(self, train\_data, train\_e, test\_data, test\_e, batch\_size):

max\_epoch: int = 100

num\_of\_batches = len(train\_data) / batch\_size

for epoch in range(max\_epoch):

start: int = 0

while start < num\_of\_batches:

batch\_data = train\_data[start: start + batch\_size]

batch\_e = train\_e[start: start + batch\_size]

weight\_updater = np.zeros(2)

batch\_error: float = 0

for ind, image in enumerate(batch\_data):

Y = self.forward\_prop(image)

prediction = self.activation\_func(Y)

error = prediction - batch\_e[ind]

batch\_error += error \* self.activation\_derivative(Y)

weight\_updater += error

weight\_updater = weight\_updater \* image

weight\_updater \*= self.activation\_derivative(Y)

if self.adaptive\_flag:

self.adaptive\_batch(batch\_data, batch\_e)

for i in range(self.W.shape[0]):

for j in range(self.W.shape[1]):

# self.W[i, j] -= self.alp \* error \* self.X[i]

self.W[i, j] -= self.alp \* weight\_updater[i]

self.T[j] += self.alp \* batch\_error

start += batch\_size

if self.test(test\_data, test\_e):

print(f"Network trained in {epoch} epoches")

self.drawErrorGraph()

return

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Adaptive alp** | | **Const alp** | |
| **batch** | **online** | **batch** | **online** |
| **Epochs** | 17 | 12 | 2 | 5 |
| **Error%** | 4 | 2 | 4 | 0 |
| **time** | 0.27 | 0.098 | 0.048 | 0.041 |

**Вывод:** в ходе лабораторной работы научился реализовывать однослойную нейронную сеть для решения задачи классификации