**Лущик Никита КС-44**

**Нейронная сеть Кохонена**

**Входные данные**

Массив из 20ти обучающих примеров

InputArray = [[1.00, 0.00, 0.17, 0.52, 0.35, 0.13, 0.00], *# 1*

[1.00, 0.00, 0.17, 0.58, 0.35, 0.00, 0.00], *# 2*

[0.00, 0.00, 0.17, 0.58, 0.35, 0.70, 0.60], *# 3*

[1.00, 1.00, 1.00, 0.77, 0.84, 0.75, 1.00], *# 4*

[0.00, 1.00, 0.33, 0.77, 0.70, 0.71, 0.71], *# 5*

[0.00, 1.00, 0.17, 0.77, 0.90, 0.87, 0.63], *# 6*

[0.00, 1.00, 0.00, 0.78, 0.65, 0.74, 0.81], *# 7*

[1.00, 0.00, 0.00, 0.52, 0.58, 0.59, 0.63], *# 8*

[1.00, 0.00, 0.00, 0.57, 0.24, 0.68, 0.49], *# 9*

[1.00, 1.00, 0.17, 0.78, 0.70, 0.77, 0.68], *# 10*

[0.00, 1.00, 1.00, 0.90, 0.99, 1.00, 1.00], *# 11*

[0.00, 1.00, 0.17, 0.89, 0.88, 0.70, 0.63], *# 12*

[1.00, 0.00, 0.00, 0.61, 0.00, 0.05, 0.49], *# 13*

[0.00, 1.00, 0.83, 0.83, 0.72, 0.77, 0.81], *# 14*

[1.00, 0.00, 0.00, 0.00, 0.03, 0.03, 0.49], *# 15*

[0.00, 1.00, 0.17, 0.65, 0.66, 0.68, 0.49], *# 16*

[1.00, 1.00, 0.67, 1.00, 1.00, 0.89, 1.00], *# 17*

[0.00, 1.00, 1.00, 0.85, 0.94, 0.92, 0.81], *# 18*

[1.00, 1.00, 0.83, 0.52, 0.58, 0.74, 0.49], *# 19*

[1.00, 0.00, 0.00, 0.57, 0.35, 0.03, 0.63]] *# 20*

Начальные значения весовых коэффициентов

W = [[0.20, 0.20, 0.30, 0.40, 0.40, 0.20, 0.50],*# claster 1*

[0.20, 0.80, 0.70, 0.80, 0.70, 0.70, 0.80], *# claster 2*

[0.80, 0.20, 0.50, 0.50, 0.40, 0.40, 0.40], *# claster 3*

[0.80, 0.80, 0.60, 0.70, 0.70, 0.60, 0.70]] *# claster 4*

Начальная скорость обучения 0.3, уменьшается с каждой эпохой; в начале эпохи массив примеров перемешивается.

**Листинг Kohonen.py**

**from** math **import** sqrt

**from** random **import** shuffle

**import** Common

**def** Kohonen(InputArray, W, V, cluster, Eras): *# Self-learning # Input array rows count must be divisible to cluster count*

Vdecrement = V/Eras

R = []

era = 0

**while**(era < Eras):

**print**("**\n**Era", era, "Learn speed:", V, "**\n**-----------------------------------------------")

**print**("Input array:**\n**")

Common.NumerizedPrint(InputArray)

**print**("**\n**Start W:**\n**")

Common.FloatNumerizedPrint(W, 2)

**for** e **in** range(len(InputArray)):

Rtmparray = []

**for** j **in** range(cluster):

Rtmp = 0

**for** i **in** range(len(InputArray[0])):

Rtmp += (InputArray[e][i]-W[j][i])\*\*2 *# R*

Rtmp = sqrt(Rtmp)

Rtmparray.append(Rtmp)

WWinnerIndex = Rtmparray.index(min(Rtmparray))

**print**("**\n**Step", e)

**print**("R current: ", Rtmparray)

**print**("Winner index: ", WWinnerIndex)

**for** i **in** range(len(W[0])):

W[WWinnerIndex][i] = W[WWinnerIndex][i] + V\*(InputArray[e][i] - W[WWinnerIndex][i])

R.append(Rtmparray.copy())

**print**("New W: ")

Common.FloatNumerizedPrint(W, 2)

V -= Vdecrement

era += 1

shuffle(InputArray)

**def** main():

InputArray = [[1.00, 0.00, 0.17, 0.52, 0.35, 0.13, 0.00], *# 10*

[1.00, 0.00, 0.17, 0.58, 0.35, 0.00, 0.00], *# 2*

[0.00, 0.00, 0.17, 0.58, 0.35, 0.70, 0.60], *# 3*

[1.00, 1.00, 1.00, 0.77, 0.84, 0.75, 1.00], *# 4*

[0.00, 1.00, 0.33, 0.77, 0.70, 0.71, 0.71], *# 5*

[0.00, 1.00, 0.17, 0.77, 0.90, 0.87, 0.63], *# 6*

[0.00, 1.00, 0.00, 0.78, 0.65, 0.74, 0.81], *# 7*

[1.00, 0.00, 0.00, 0.52, 0.58, 0.59, 0.63], *# 8*

[1.00, 0.00, 0.00, 0.57, 0.24, 0.68, 0.49], *# 9*

[1.00, 1.00, 0.17, 0.78, 0.70, 0.77, 0.68], *# 1*

[0.00, 1.00, 1.00, 0.90, 0.99, 1.00, 1.00], *# 11*

[0.00, 1.00, 0.17, 0.89, 0.88, 0.70, 0.63], *# 12*

[1.00, 0.00, 0.00, 0.61, 0.00, 0.05, 0.49], *# 13*

[0.00, 1.00, 0.83, 0.83, 0.72, 0.77, 0.81], *# 14*

[1.00, 0.00, 0.00, 0.00, 0.03, 0.03, 0.49], *# 15*

[0.00, 1.00, 0.17, 0.65, 0.66, 0.68, 0.49], *# 16*

[1.00, 1.00, 0.67, 1.00, 1.00, 0.89, 1.00], *# 17*

[0.00, 1.00, 1.00, 0.85, 0.94, 0.92, 0.81], *# 18*

[1.00, 1.00, 0.83, 0.52, 0.58, 0.74, 0.49], *# 19*

[1.00, 0.00, 0.00, 0.57, 0.35, 0.03, 0.63]] *# 20*

W = [[0.20, 0.20, 0.30, 0.40, 0.40, 0.20, 0.50], *# claster 1*

[0.20, 0.80, 0.70, 0.80, 0.70, 0.70, 0.80], *# claster 2*

[0.80, 0.20, 0.50, 0.50, 0.40, 0.40, 0.40], *# claster 3*

[0.80, 0.80, 0.60, 0.70, 0.70, 0.60, 0.70]] *# claster 4*

Kohonen(InputArray, W, 0.3, 4, 10)

 main()

**Листинг Common.py**

**def** NumerizedPrint(array, end = "**\n**"):

**for** i **in** range(len(array)):

**print**("%3d) %s" % (i, array[i]), end=end)

**def** FloatNumerizedPrint(array, roundsigns):

**for** i **in** range(len(array)):

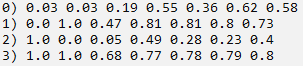
**print**("%3d)" % i, end=" ")

**for** j **in** range(len(array[0])):

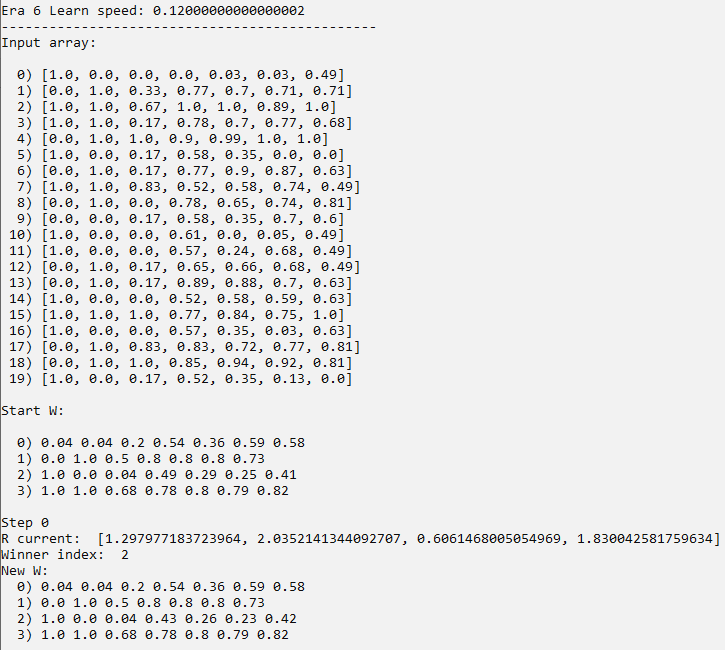
**print**(round(array[i][j], roundsigns), end=" ")

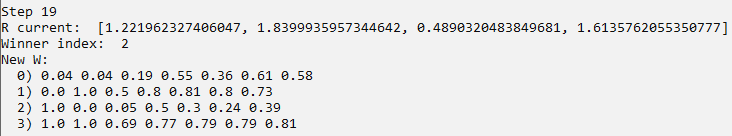
**print**("**\n**", end="")

**Итоговые значения весовых коэффициентов нейронной сети Кохонена**



**Фрагмент вывода программы**

****

****

**Вывод**

Все четыре кластера оказались заполнены. В 1-й кластер попал только пример № 3; во 2-ой – примеры № 5, 7, 11, 12, 14, 16, 18; в 3-й кластер – примеры № 2, 8, 9, 10, 13, 15, 20; в 4-й – № 1, 4, 17, 19. Успешность обучения также проверена подачей на вход примера, не участвовавшего в процессе обучения. В результате проверки можно сделать вывод, что нейронная сеть обучена и может использоваться для получения характеристики любого другого примера, для которого имеется тот же набор входных данных.