МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №7

по дисциплине «Нейронные сети»

тема: «Нейрокомпьютерные сети основанные на соревновании.

Самоорганизующееся разбиение Кохонена»

Выполнила: ст. группы МИВТ-201

Браткова Ирина Олеговна

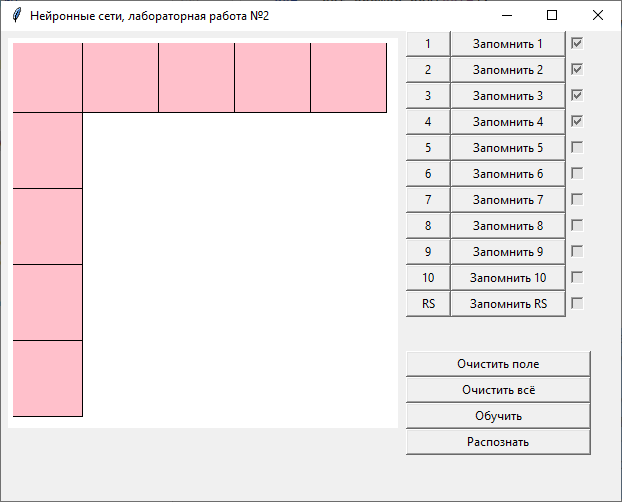
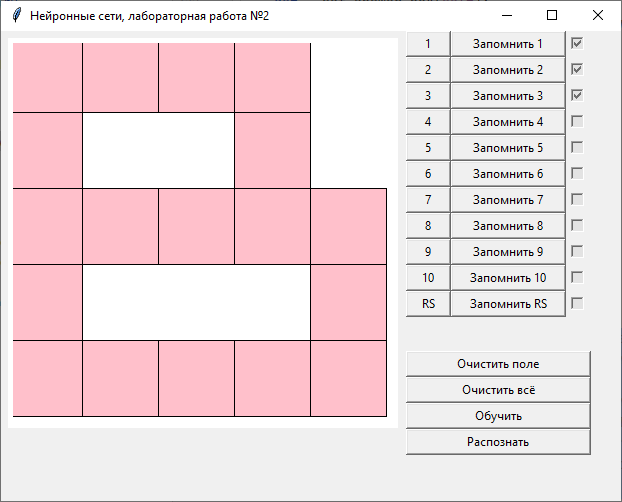
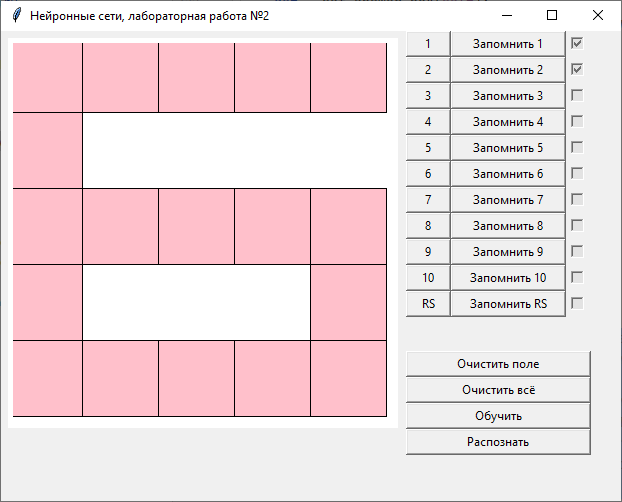
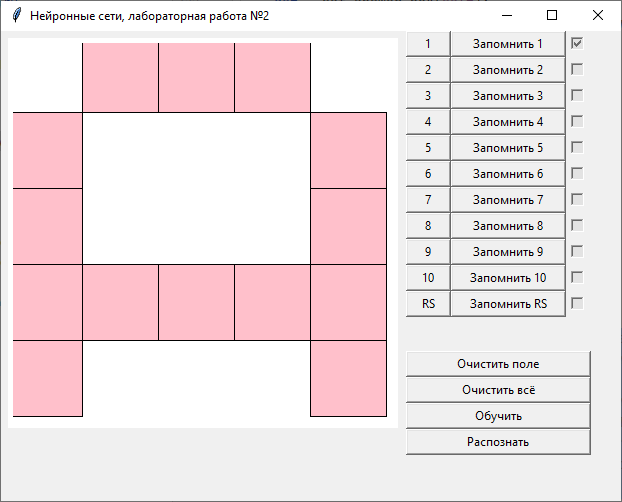
Белгород 2020 г

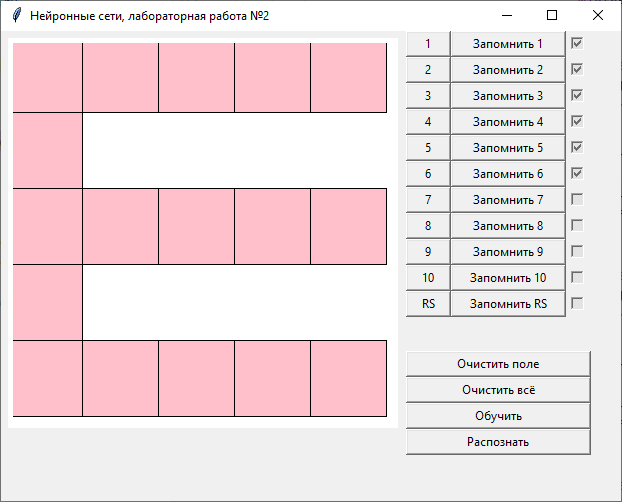
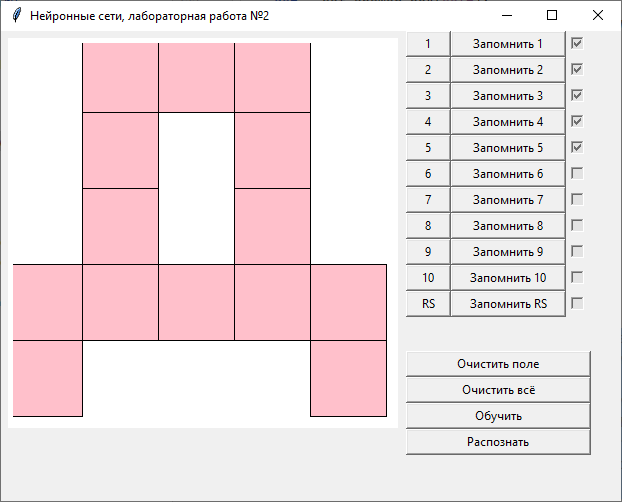
***Цель работы:*** *приобретение и закрепление знаний, получение практических навыков работы с нейрокомпьютерными сетями, основанными на соревновании, исследование сети Хемминга.*

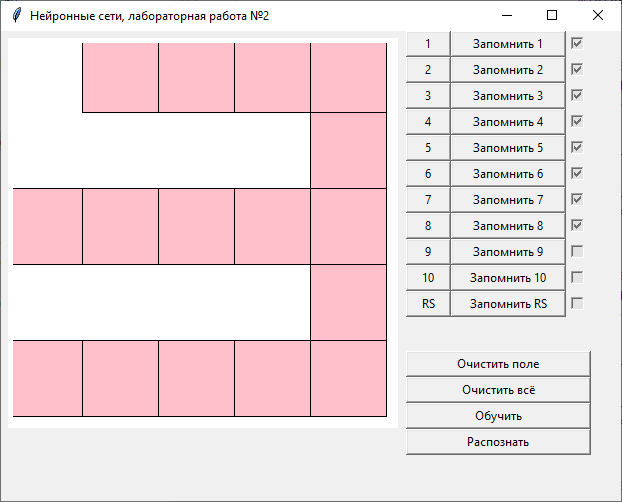
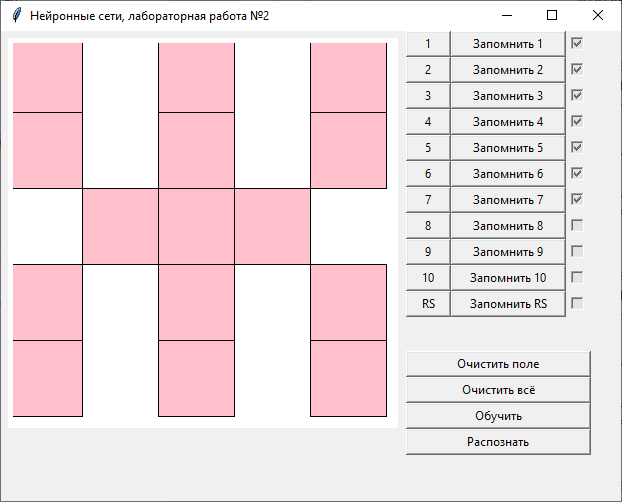
***Индивидуальные задания:***

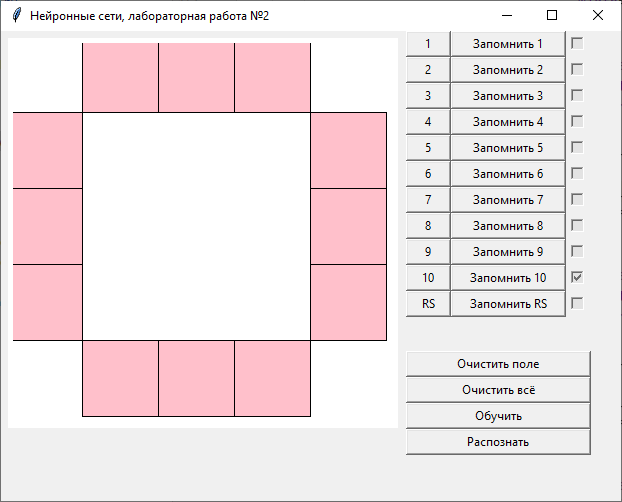
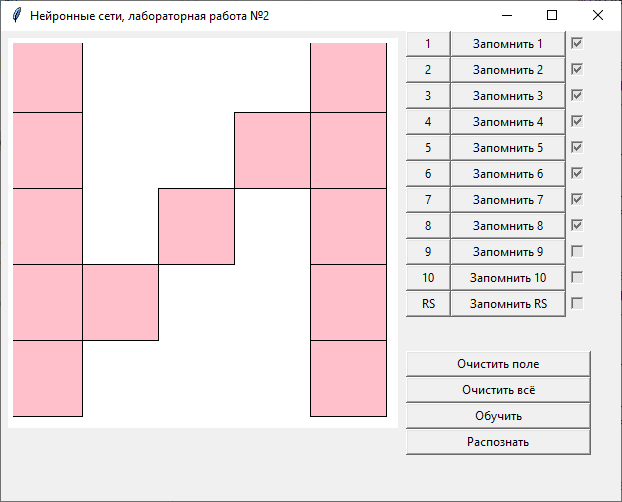
* разработать по два черно-белых изображения пяти первых букв Вашей фамилии (буквы должны быть разными);
* разработать архитектуру нейрокомпьютерной сети Хемминга;
* реализовать алгоритм функционирования сети Хемминга;
* исследовать возможности сети Хемминга по распознаванию изображений;
* исследовать подсеть MaxNet: влияние коэффициента скорости обучения на процесс выявления «победителя»;
* исследовать работу сеть Хемминга, при распознавании изображения, имеющего с двумя и более эталонными одинаковые максимальные меры близости.

***Выполнение***









***Основные модули программы***

* mainwindow.py – модуль, отвечающий за отображение экранной формы;
* controller.py – модуль, отвечающий за связь экранной формы с модулем вычислений;
* calculating.py – класс, выполняющий вычисления;

***Calculating.py***

class CalculatingMor:  
 def \_\_init\_\_(self):  
 self.letter\_1 = [1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1]  
 self.letter\_2 = [1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]  
 self.letter\_3 = [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1]  
 self.letter\_4 = [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]  
 self.letter\_5 = [0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]  
 self.letter\_6 = [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]  
 self.letter\_7 = [1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]  
 self.letter\_8 = [1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]  
 self.letter\_9 = [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1]  
 self.letter\_0 = [1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1]  
 self.letter\_rs = [1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1]  
 self.m = None  
 self.alpha = None  
 self.k = 0.5  
 self.w\_matrix = None  
 self.recognize\_matrix = None  
 self.a\_list = None  
 self.letters\_list = None  
 self.eps = 0.000001  
 self.dictionary = None  
  
 def teaching(self, flag=True):  
 print(**'>> ОБУЧЕНИЕ....'**)  
 self.\_\_create\_start\_data\_\_(flag)  
 flag = False  
 matrix\_a = copy.copy(self.w\_matrix)  
 alpha = copy.copy(self.alpha)  
 iterate = 1  
 while not flag:  
 print(**'**\n**==============Итерация №'** + str(iterate) +**'=============='**, end=**''**)  
 tmp\_matrix = copy.deepcopy(matrix\_a)  
 for i in range(len(self.letters\_list)):  
 j = i + 1  
 print(**'**\n**Символ №'** + str(j) + **': '** + str(self.letters\_list[i]))  
 win\_index = self.\_\_calc\_d\_\_(self.letters\_list[i], tmp\_matrix)  
 tmp\_matrix = self.\_\_calc\_w\_new\_\_(win\_index, tmp\_matrix, self.letters\_list[i], alpha)  
 print(**'**\n**Веса: '**)  
 for el in tmp\_matrix:  
 print(el)  
 alpha = alpha \* self.k  
 matrix\_b = copy.copy(tmp\_matrix)  
 flag = self.\_\_sub\_eps\_\_(matrix\_a, matrix\_b)  
 iterate += 1  
 print(**'**\n**ЗНАЧЕНИЕ ВЕСОВ В КОНЦЕ ЭТОЙ ИТЕРАЦИИ: '**)  
 for el in tmp\_matrix:  
 print(el)  
 print(**'Значение alpha после коррекции, aplha = '** + str(alpha))  
 matrix\_a = copy.copy(matrix\_b)  
 print(**'====================================='**)  
 res\_matr = copy.copy(matrix\_a)  
 print(**'**\n**Последняя матрица весов (с округлением до 2 символов): '**)  
 res\_matr2 = [[round(r, 2) for r in row] for row in res\_matr]  
 for el in res\_matr2:  
 print(el)  
 self.w\_matrix = copy.deepcopy(matrix\_a)  
 self.classified()  
  
 def classified(self):  
 print(**'**\n**>> КЛАССИФИКАЦИЯ....'**)  
 print(**'нумерация с 0'**)  
 dictionary = {}  
 classes = []  
 tmp\_matrix = copy.deepcopy(self.w\_matrix)  
 i = 0  
 for letter in self.letters\_list:  
 a = nym.array([letter])  
 b = nym.array(tmp\_matrix)  
 total = a.dot(b)  
 array = total.tolist()  
 array0 = array[0]  
 index = array0.index(max(array0))  
 print(**'Изображение '** + str(i) + **': '** + str(letter) + **', отнесено к классу: '** + str(index))  
 if index not in classes:  
 classes.append(index)  
 if index not in dictionary:  
 dictionary[index] = []  
 dictionary[index].append(i)  
 else:  
 dictionary[index].append(i)  
 i += 1  
 classes.sort()  
 print(dictionary)  
 self.dictionary = dictionary  
 print(**'Обнаружены классы: '** + str(classes))  
 del\_classes = []  
 for i in range(len(self.letters\_list)):  
 if i not in classes:  
 del\_classes.append(i)  
 del\_classes.reverse()  
 print(**'Не обнаруженные классы: '** + str(del\_classes))  
 for index in del\_classes:  
 for row in tmp\_matrix:  
 del row[index]  
 res\_matrix = [[round(r, 2) for r in row] for row in tmp\_matrix]  
 for el in res\_matrix:  
 print(el)  
 print()  
 sorted(self.dictionary.keys())  
 self.recognize\_matrix = copy.deepcopy(tmp\_matrix)  
  
 def recognize(self):  
 print(**'**\n**>> РАСПОЗНАВАНИЕ....'**)  
 self.\_\_calculate\_gemini()  
 a = nym.array([self.letter\_rs])  
 b = nym.array(self.recognize\_matrix)  
 total = a.dot(b)  
 array = total.tolist()  
 array = array[0]  
 print(self.dictionary)  
 print(array)  
  
 def \_\_calculate\_gemini(self):  
 print(**'Схожесть... '**)  
 maximum = 0  
 index = 0  
 i = 0  
 for key, value in self.dictionary.items():  
 s = **'Класс №'** + str(key) + **': '** summa = 0  
 for el in value:  
 letter = self.letters\_list[el]  
 summa += self.\_\_help\_gemini(self.letter\_rs, letter)  
 s += str(summa)  
 if summa >= maximum:  
 maximum = summa  
 index = key  
 print(s)  
 i += 1  
 print(**'Предположительно rs класс №'** + str(index))  
  
 def \_\_help\_gemini(self, letter\_a, letter\_b):  
 t = 0  
 for i in range(len(letter\_a)):  
 if letter\_b[i] == letter\_a[i]:  
 t += 1  
 return t  
  
 def \_\_calc\_d\_\_(self, letter, matrix):  
 ds = []  
 for j in range(self.m): *#по кол-ву A-элементов* d = sum([((matrix[i][j] - letter[i]) \*\* 2) for i in range(len(letter))])  
 ds.append(d)  
 index = ds.index(min(ds))  
 jndex = copy.copy(index) + 1  
 print(str(ds) + **'**\n**Победил элемент №'** + str(jndex) + **' = '** + str(ds[index]))  
 return index  
  
 def \_\_calc\_w\_new\_\_(self, index, matrix, letter, alpha):  
 tmp\_matrix = copy.deepcopy(matrix)  
 for i in range(len(letter)):  
 w\_old = float(tmp\_matrix[i][index])  
 letter\_w\_old = letter[i] - w\_old  
 letter\_w\_old \*= alpha  
 w\_new = w\_old + letter\_w\_old  
 tmp\_matrix[i][index] = copy.deepcopy(w\_new)  
 return tmp\_matrix  
  
 def \_\_sub\_eps\_\_(self, matrix\_a, matrix\_b):  
 a = nym.array(matrix\_a)  
 b = nym.array(matrix\_b)  
 c = a - b  
 for el in c:  
 for e in el:  
 if nym.fabs(e) > self.eps:  
 return False  
 return True  
  
 def \_\_sub\_matrix\_\_(self, a, b):  
 pass  
  
  
 def \_\_make\_letters\_list(self):  
 self.letters\_list = []  
 if self.letter\_1 is not None:  
 self.letters\_list.append(self.letter\_1)  
 if self.letter\_2 is not None:  
 self.letters\_list.append(self.letter\_2)  
 if self.letter\_3 is not None:  
 self.letters\_list.append(self.letter\_3)  
 if self.letter\_4 is not None:  
 self.letters\_list.append(self.letter\_4)  
 if self.letter\_5 is not None:  
 self.letters\_list.append(self.letter\_5)  
 if self.letter\_6 is not None:  
 self.letters\_list.append(self.letter\_6)  
 if self.letter\_7 is not None:  
 self.letters\_list.append(self.letter\_7)  
 if self.letter\_8 is not None:  
 self.letters\_list.append(self.letter\_8)  
 if self.letter\_9 is not None:  
 self.letters\_list.append(self.letter\_9)  
 if self.letter\_0 is not None:  
 self.letters\_list.append(self.letter\_0)  
  
 def \_\_create\_start\_data\_\_(self, flag):  
 if not flag:  
 self.m = 10  
 self.\_\_make\_letters\_list()  
 self.w\_matrix = [[random.uniform(0.0, 1.0) for j in range(self.m)] for i in range(len(self.letters\_list[0]))]  
 else:  
 self.letter\_0 = [0, 0, 0, 1]  
 self.letter\_1 = [0, 0, 1, 1]  
 self.letter\_2 = [1, 0, 0, 0]  
 self.letter\_3 = [1, 1, 0, 0]  
 self.letter\_4 = None  
 self.letter\_5 = None  
 self.letter\_6 = None  
 self.letter\_7 = None  
 self.letter\_8 = None  
 self.letter\_9 = None  
 self.m = 2  
 self.a\_list = [0 for \_ in range(self.m)]  
 self.\_\_make\_letters\_list()  
 self.w\_matrix = [  
 [0.7, 0.6],  
 [0.4, 0.1],  
 [0.5, 0.5],  
 [0.2, 0.9] ]  
 self.alpha = 0.6  
 print(**'Буквы: '**)  
 for el in self.letters\_list:  
 print(el)  
 print(**'**\n**Веса: '**)  
 for el in self.w\_matrix:  
 print(el)

**Пример логов программы (на тестовом примере из теории, итерации 2-8 специально исключены для отчета):**

>> ОБУЧЕНИЕ....

Буквы:

[0, 0, 1, 1]

[1, 0, 0, 0]

[1, 1, 0, 0]

[0, 0, 0, 1]

Веса:

[0.7, 0.6]

[0.4, 0.1]

[0.5, 0.5]

[0.2, 0.9]

==============Итерация №1==============

Символ №1: [0, 0, 1, 1]

[1.54, 0.63]

Победил элемент №2 = 0.63

Веса:

[0.7, 0.24]

[0.4, 0.04000000000000001]

[0.5, 0.8]

[0.2, 0.96]

Символ №2: [1, 0, 0, 0]

[0.54, 2.1408000000000005]

Победил элемент №1 = 0.54

Веса:

[0.88, 0.24]

[0.16000000000000003, 0.04000000000000001]

[0.2, 0.8]

[0.08000000000000002, 0.96]

Символ №3: [1, 1, 0, 0]

[0.7663999999999999, 3.0608000000000004]

Победил элемент №1 = 0.7663999999999999

Веса:

[0.952, 0.24]

[0.664, 0.04000000000000001]

[0.08000000000000002, 0.8]

[0.03200000000000001, 0.96]

Символ №4: [0, 0, 0, 1]

[2.2906239999999998, 0.7008000000000002]

Победил элемент №2 = 0.7008000000000002

Веса:

[0.952, 0.096]

[0.664, 0.016000000000000004]

[0.08000000000000002, 0.32000000000000006]

[0.03200000000000001, 0.984]

ЗНАЧЕНИЕ ВЕСОВ В КОНЦЕ ЭТОЙ ИТЕРАЦИИ:

[0.952, 0.096]

[0.664, 0.016000000000000004]

[0.08000000000000002, 0.32000000000000006]

[0.03200000000000001, 0.984]

Значение alpha после коррекции, aplha = 0.3

=====================================

\*\*\*

==============Итерация №9==============

Символ №1: [0, 0, 1, 1]

[3.2517073876628855, 0.34064618554478604]

Победил элемент №2 = 0.34064618554478604

Веса:

[0.9873916883620387, 0.0251575218151192]

[0.5788709648381597, 0.004192920302519866]

[0.02101385272993511, 0.41829284368315994]

[0.008405541091974047, 0.99580707969748]

Символ №2: [1, 0, 0, 0]

[0.33576279858262614, 2.116936080906471]

Победил элемент №1 = 0.33576279858262614

Веса:

[0.9874212390924402, 0.0251575218151192]

[0.5775142360143203, 0.004192920302519866]

[0.020964601512599323, 0.41829284368315994]

[0.008385840605039732, 0.99580707969748]

Символ №3: [1, 1, 0, 0]

[0.17916228283576824, 3.1085502403014313]

Победил элемент №1 = 0.17916228283576824

Веса:

[0.9874507205633173, 0.0251575218151192]

[0.5785044370236617, 0.004192920302519866]

[0.02091546572780417, 0.41829284368315994]

[0.00836618629112167, 0.99580707969748]

Символ №4: [0, 0, 0, 1]

[2.293501386394504, 0.17563696514174923]

Победил элемент №2 = 0.17563696514174923

Веса:

[0.9874507205633173, 0.025098558873365016]

[0.5785044370236617, 0.004183093145560835]

[0.02091546572780417, 0.4173124698307775]

[0.00836618629112167, 0.995816906854439]

ЗНАЧЕНИЕ ВЕСОВ В КОНЦЕ ЭТОЙ ИТЕРАЦИИ:

[0.9874507205633173, 0.025098558873365016]

[0.5785044370236617, 0.004183093145560835]

[0.02091546572780417, 0.4173124698307775]

[0.00836618629112167, 0.995816906854439]

Значение alpha после коррекции, aplha = 0.001171875

=====================================

Последняя матрица весов (с округлением до 2 символов):

[0.99, 0.03]

[0.58, 0.0]

[0.02, 0.42]

[0.01, 1.0]

>> КЛАССИФИКАЦИЯ....

нумерация с 0

Изображение 0: [0, 0, 1, 1], отнесено к классу: 1

Изображение 1: [1, 0, 0, 0], отнесено к классу: 0

Изображение 2: [1, 1, 0, 0], отнесено к классу: 0

Изображение 3: [0, 0, 0, 1], отнесено к классу: 1

{1: [0, 3], 0: [1, 2]}

Обнаружены классы: [0, 1]

Не обнаруженные классы: [3, 2]

***Результаты совпадают с теми, что были представлены в теории к ЛР.***

**Исследование сети Кохонена на примере из 10 изображений (изображения приведены в начале отчёта)**

**Классы записываются, как {номер класса: [изображение1, …, изображениеN]}**

**Граница весовых коэффициентов [0;1]**

**Alpha = 0.6**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| eps | k | Итераций | Классов | Классы |
| 0.0005 | 0.5 | 12 | 6 | {9: [0, 4], 2: [1, 2, 5, 7], 5: [3], 6: [6], 3: [8], 8: [9]} |
|  |  | 11 | 5 | {6: [0], 9: [1, 2, 3, 5, 7, 9], 5: [4], 1: [6], 7: [8]} |
|  |  | 11 | 6 | {8: [0, 3], 9: [1, 2, 5, 7], 7: [4], 5: [6], 6: [8], 4: [9]} |
|  | 0.1 | 6 | 6 | {7: [0, 4], 0: [1, 2, 5, 7], 4: [3], 1: [6], 5: [8], 6: [9]} |
|  |  | 6 | 5 | {1: [0], 9: [1, 2, 3, 5, 7, 9], 5: [4], 7: [6], 4: [8]} |
|  |  | 6 | 7 | {3: [0], 1: [1, 2, 5, 7], 2: [3], 4: [4], 6: [6], 0: [8], 8: [9]} |
| 0.0001 | 0.5 | 13 | 5 | {4: [0], 9: [1, 2, 3, 5, 7, 9], 6: [4], 3: [6], 8: [8]} |
|  |  | 14 | 5 | {1: [0], 6: [1, 2, 5, 7, 9], 9: [3], 2: [4], 0: [6, 8]} |
|  |  | 13 | 6 | {5: [0, 9], 6: [1, 2, 5, 7], 7: [3], 4: [4], 0: [6], 8: [8]} |

**Alpha = 0.1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| eps | k | Итераций | Классов | Классы |
| 0.0005 | 0.5 | 10 | 4 | {2: [0], 5: [1, 2, 3, 5, 7, 8, 9], 6: [4], 1: [6]} |
|  |  | 10 | 4 | {1: [0, 1, 2, 3, 5, 7, 9], 2: [4], 9: [6], 5: [8]} |
|  |  | 11 | 2 | {0: [0, 3, 4, 8], 4: [1, 2, 5, 6, 7, 9]} |
| 0.0001 | 0.5 | 13 | 5 | {4: [0], 9: [1, 2, 3, 5, 7, 9], 6: [4], 3: [6], 8: [8]} |
|  |  | 13 | 4 | {8: [0], 5: [1, 2, 3, 5, 6, 7, 9], 6: [4], 4: [8]} |
|  |  | 13 | 4 | {8: [0, 4, 9], 3: [1, 2, 3, 5, 7], 1: [6], 0: [8]} |

**Alpha = 0.9**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| eps | k | Итераций | Классов | Классы |
| 0.0001 | 0.1 | 7 | 7 | {8: [0], 1: [1, 2, 5, 7], 5: [3], 7: [4], 4: [6], 2: [8], 3: [9]} |
|  |  | 7 | 6 | {9: [0, 9], 5: [1, 2, 5, 7], 1: [3], 0: [4], 7: [6], 6: [8]} |
|  |  | 7 | 7 | {8: [0], 3: [1, 2, 5, 7], 4: [3], 6: [4], 0: [6], 7: [8], 1: [9]} |
| 0.0005 | 0.1 | 6 | 7 | {7: [0], 5: [1, 2, 5, 7], 8: [3], 4: [4], 0: [6], 2: [8], 1: [9]} |
|  |  | 6 | 6 | {8: [0, 4], 1: [1, 2, 5, 7], 5: [3], 3: [6], 0: [8], 9: [9]} |
|  |  | 7 | 6 | {9: [0, 9], 0: [1, 2, 5, 7], 8: [3], 1: [4], 4: [6], 6: [8]} |

*Как видно из таблицы 1, результаты довольно схожи – изображения 0 и 9, 0 и 8, или 0 и 4 чаще попадают в один класс. Зачастую изображения 0, 4, 8 и 9 представляют собой самостоятельные классы. Также, видно, что изображения 1, 2, 5 и 7 гораздо чаще оказываются в одном классе, это логично – написание буков А и Д похоже, Д также «имеет» схожие биты с Ж и Б.*

*Таблица 2 показывает, что при малом значении параметра альфа для получения точности требуется немного меньшее число итераций, однако результаты классификации хуже – многие буквы попадают в один класс.*

*Наилучшие результаты получились в таблице 3, где при большом значении alpha и малом значении k получается разбиение в среднем на большее число классов, нежели в других случаях.*

**Граница весовых коэффициентов [-0.5;0.5]**

**Alpha = 0.9 (выбраны значения eps-k-alpha, давшие наилучшие результаты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| eps | k | Итераций | Классов | Классы |
| 0.0001 | 0.1 | 7 | 3 | {2: [0, 4], 8: [1, 2, 3, 5, 7, 9], 3: [6, 8]} |
|  |  | 7 | 3 | {8: [0, 4], 9: [1, 2, 3, 5, 7, 9], 5: [6, 8]} |
|  |  | 7 | 3 | {1: [0, 4], 4: [1, 2, 3, 5, 7, 9], 3: [6, 8]} |
| 0.0005 | 0.1 | 6 | 3 | {6: [0, 4], 4: [1, 2, 3, 5, 7, 9], 5: [6, 8]} |
|  |  | 6 | 2 | {4: [0, 4], 2: [1, 2, 3, 5, 6, 7, 8, 9]} |
|  |  | 6 | 2 | {2: [0, 4], 4: [1, 2, 3, 5, 6, 7, 8, 9]} |

**При alpha = 0.1 и остальных аналогичных значениях параметров, количество классов при повторе эксперимента 10 раз дало следующий результат: 8 раз сеть определила все изображения в один класс и дважды – в два класса.**

**При alpha = 0.9, k = 0.1, eps = 0.0001, диапазоне [0, 3] в 6 из 10 опытов получилось 4 класса, а в остальных – 5 классов изображений.**

**В ходе дополнительных экспериментов было принято решение об оптимальном диапазоне [0, 1].**

*Также были проведены опыты по изменению изображений, один из которых представлен ниже:*

**Зададим значение eps = 0.0005, k = 0.5, alpha = 0.9, диапазон [0, 1]**

Организуем изображения следующим образом: каждое изображение содержит в себе на один пиксель больше, чем предыдущее. А также отличается от предыдущего только на один пиксель.

Результаты классификации:

1. {3: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}

Итераций всего 12

1. {6: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}

Итераций всего 12

Во всех последующих случаях (при различных изображениях) результаты были одинаковыми.

Т.е. изображение 0 входит во все последующие изображения, аналогично изображение 1, 2 и т.д. Т.е. существует очень мало различных пикселей. Таким образом, сеть определяет изображения в один класс.

Таким образом, количество классов, на которые может поделить изображения сеть Кохонена определяется величиной коэффициента alpha, коэффициентом k, заданной точностью, а также количеством различных пикселей изображения в разных буквах (за счет изменения букв, чаще попадающих в один класс, можно добиться увеличения количества классов). Кроме того влияние оказывают начальные весовые коэффициенты – это видно в таблицах при использовании одного набора параметров alpha-k-eps для разных экспериментов.

**Пример распознавания**

|  |  |
| --- | --- |
|  | Класс №6: 14 Класс №2: 51  Класс №1: 17 Класс №9: 12  Класс №3: 15 Класс №7: 15 Класс №4: 10  Предположительно rs класс №2  {6: [0], **2: [1, 2, 5, 7**], 1: [3], 9: [4], 3: [6], 7: [8], 4: [9]}  [6: 7.059328051850801,  **2: 8.784145531229221,**  1: 7.844594717357204,  9: 5.1114919585458365,  3: 7.844380191842438,  7: 7.924910442009336,  4: 6.879396008968139] |
|  | Класс №2: 42 Класс №7: 67  Класс №1: 16 Класс №3: 9  Класс №9: 10 Класс №4: 12  Предположительно rs класс №7  {2: [0, 9], **7: [1, 2, 5, 7],** 1: [3], 3: [4], 9: [6], 4: [8]}  [2: 5.966367852516766,  **7: 11.470421081407629**,  1: 4.960396978974933,  3: 5.958599246375203,  9: 10.162635493865544,  4: 6.020478072678172] |
|  | Класс №2: 42 Класс №7: 75  Класс №1: 16 Класс №3: 11  Класс №9: 12 Класс №4: 14  Предположительно rs класс №7  {2: [0, 9], **7: [1, 2, 5, 7],** 1: [3], 3: [4], 9: [6], 4: [8]}  [2: 6.995949725127699,  1: 11.809466737333675,  3: 6.910359616535677,  9: 7.796499160968558,  **7: 12.162617356671149,**  4: 7.884286858134566] |

Таким образом, сеть Кохонена представляет собой удобную для классификации объектов по определенным признакам сеть.