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 №3

по дисциплине «Нейронные сети»

тема: «Однослойная нейрокомпьютерная сеть.

Использование Дельта-правила»

Выполнила: ст. группы МИВТ-201

Браткова Ирина Олеговна

Белгород 2020 г

***Цель работы:*** *приобретение и закрепление знаний, получение практических навыков работы с Дельта-правилом.*

***Индивидуальные задания:***

разработать черно-белые изображения пяти первых букв Вашей фамилии (буквы должны быть разными);

разработать однослойную архитектуру нейронной сети;

реализовать алгоритм обучения нейросети на основе Дельта-правила, при этом использовать биполярные представления входных и выходных сигналов;

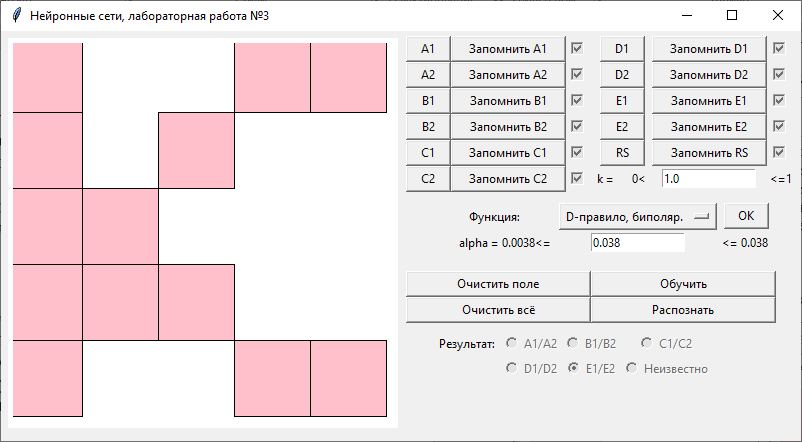
исследовать возможности алгоритма Дельта-правила в однослойной нейросети;

исследовать влияние коэффициента скорости обучения на процесс обучения;

провести сравнительный анализ правила Хэбба и Дельта-правила.

***Выполнение***

Аналогично первой лабораторной работе, при разработке был выбран язык Python. Интерфейс, разработанный для данной лабораторной работы, базируется на интерфейсе первой ЛР, и имеет следующий вид:



При запуске кнопки А1, А2, В1, В2, C1, C2, D1, D2, E1, E2 инициализированы буквами Б, Р, А, Т, К (первые пять букв фамилии).

Нажатие на кнопки «Запомнить <NM>» запоминает текущее состояние поля рисования для буквы N с номером M. Рядом с соответствующей буквой после запоминания устанавливается флажок, указывающий на то, что буква сохранена. Повторное нажатие на кнопки типа «<NM>» (после запоминания) отобразит на поле сохраненную букву.

При желании можно сохранить любые наборы букв, отличные от тех, что предусмотрены в программе.

Набор кнопок для буквы RS имеет аналогичное поведение, однако RS не участвует в обучение, это символ для распознавания.

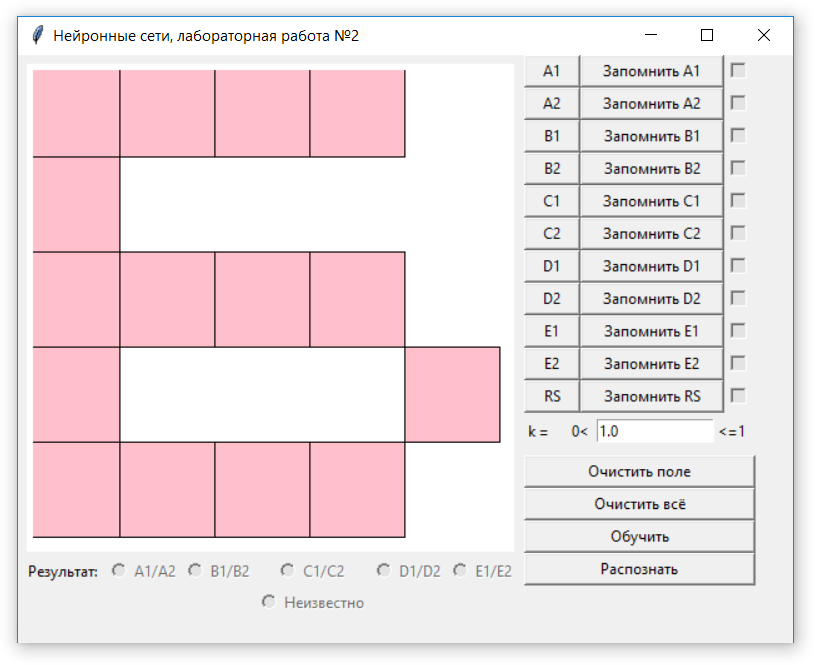
Следующий элемент – коэффициент плавности обучения, по умолчанию установленный в значение 1.0. Используется для обучения по Хеббу.

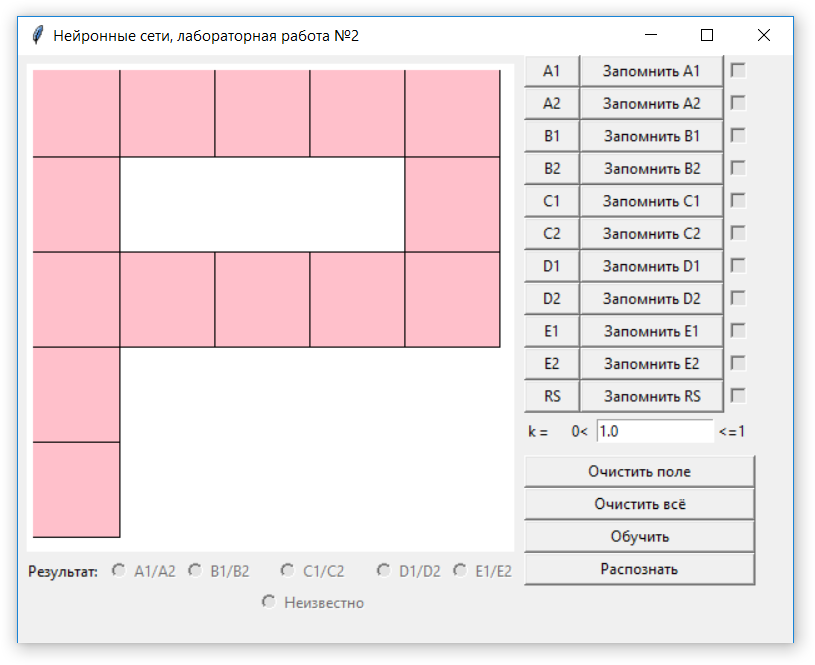
Далее – выбор функции. При использовании D-правила в алгоритме используется коэффициент alpha, который можно ввести в предложенном диапазоне. По умолчанию 0.038.

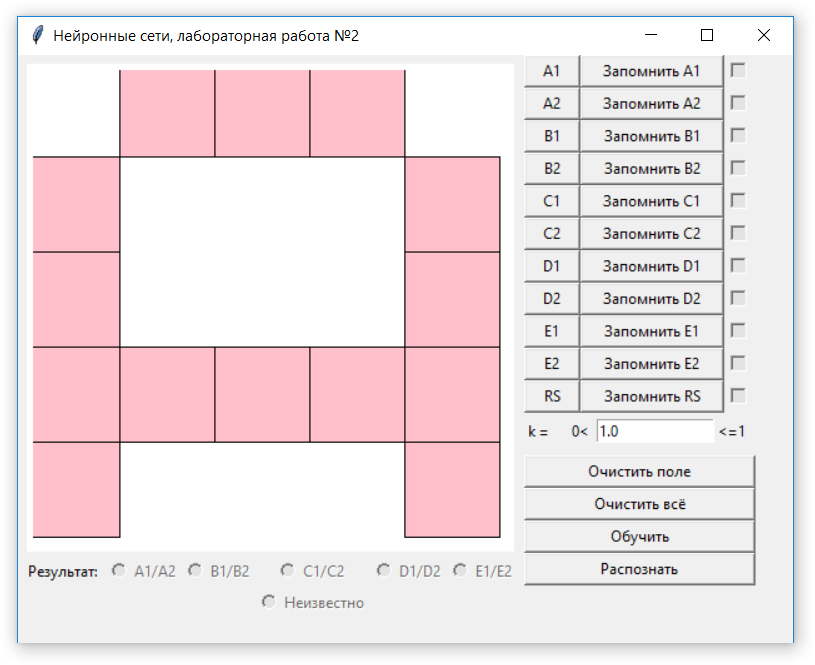
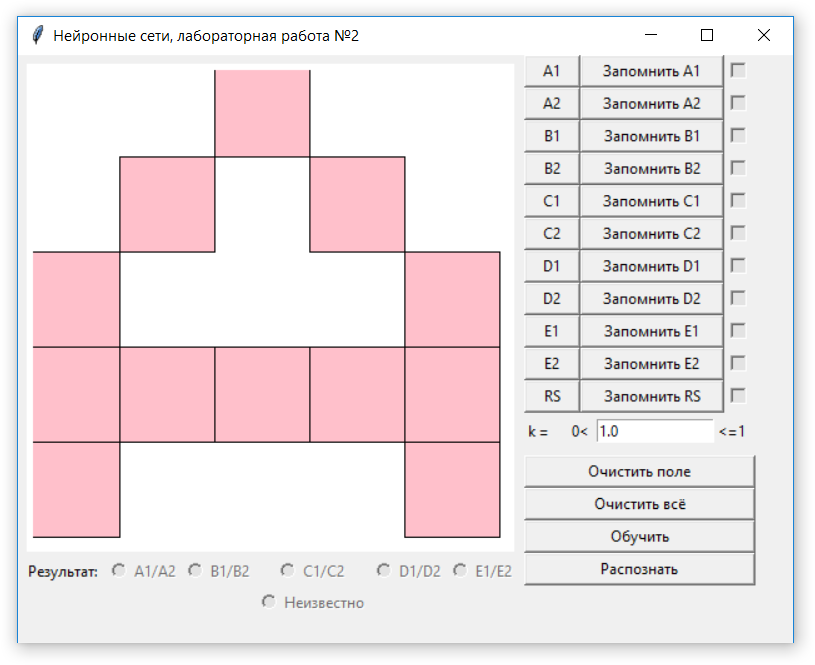
Следующие кнопки отвечают за очистку поля (без очистки информации о буквах), полной очистке поля и буков, хранящихся в памяти (аналог перезапуска программы), а также обучению на основе образов и распознаванию буквы RS.

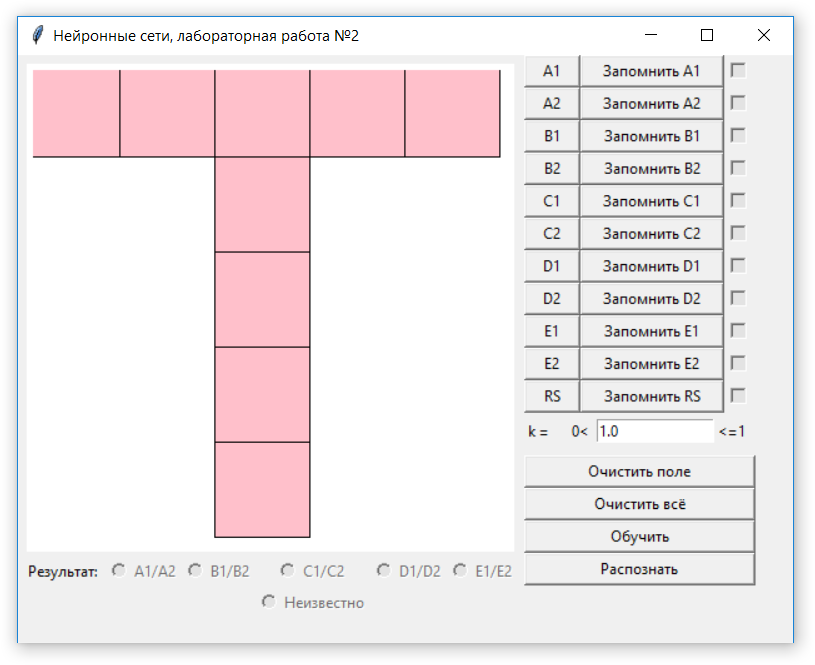
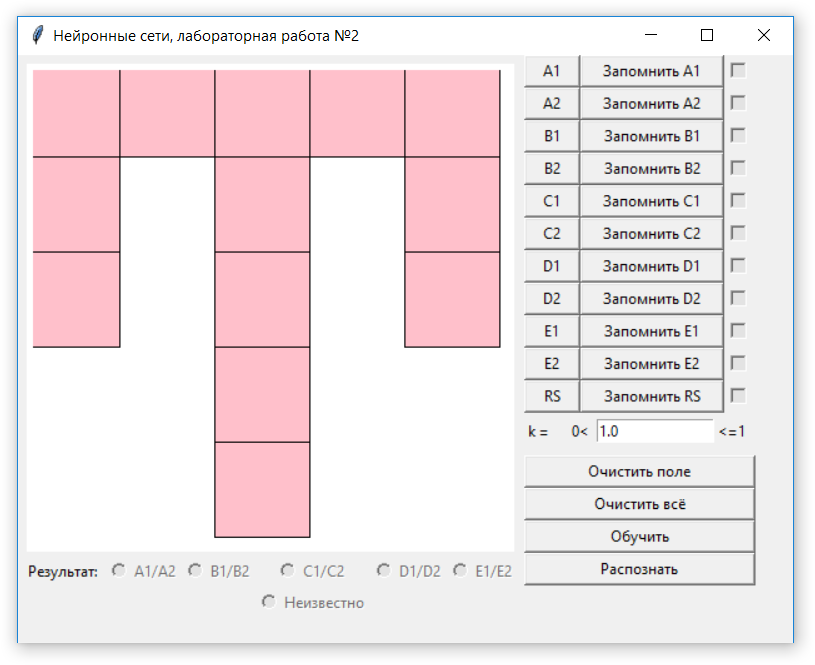
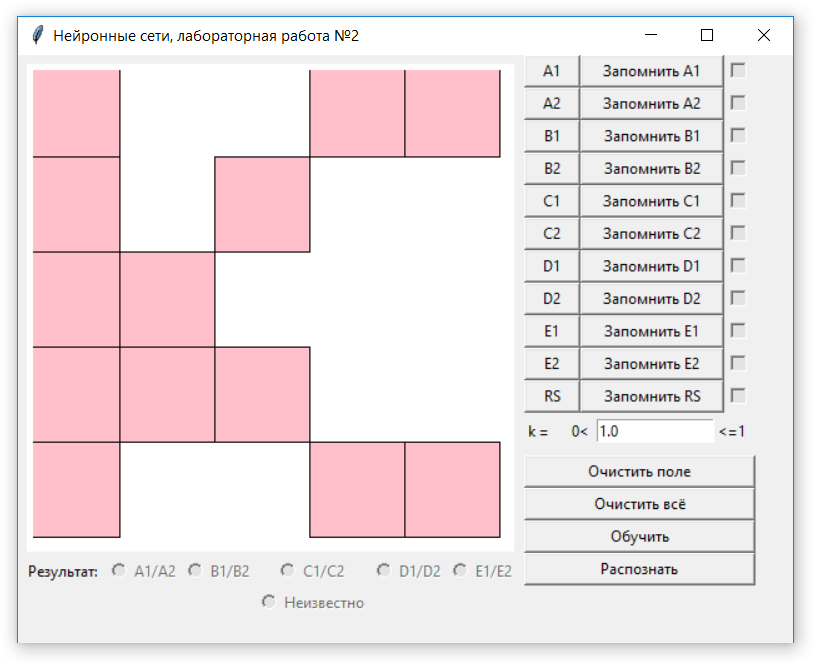
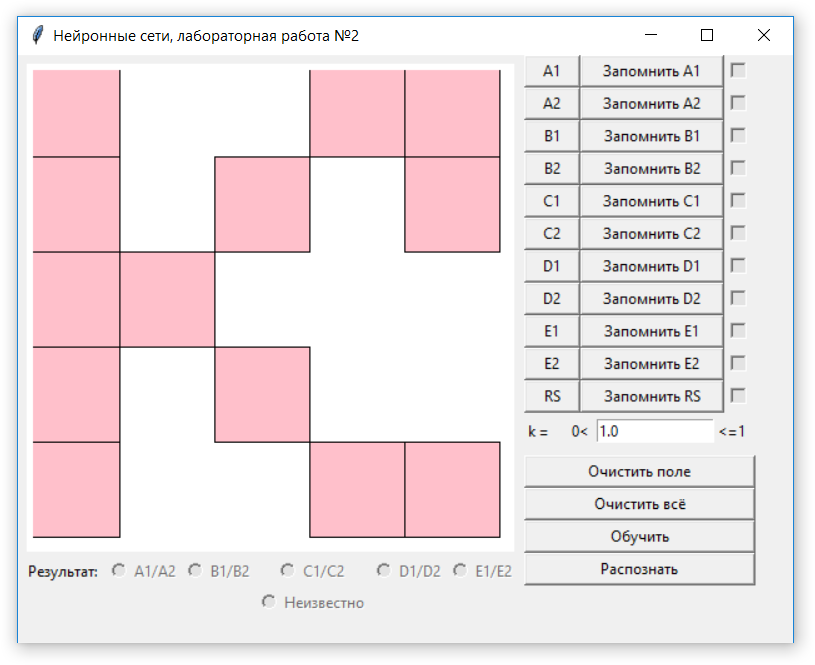
Результат работы программы отображается в блоке «Результат». В зависимости от того, к какому классу принадлежит изображение, устанавливается соответственный флажок.

***Для данной ЛР используются символы, разработанные для ЛР№2:***

******

****** ******



***Расчёты в Excel (файл Браткова\_Л3\_таблицы.xlsx)***

Для изучения материала было принято решение провести расчёты для обучения нейрона на образах букв «Ч» и «П» (первый лист файла).

**alpha = 0.1**, из расчёта, что 0.1 <= n\*alpha <= 1.0, где n – количество входов, включая смещение, n = 10.

**epsilon = 0.01**



Таким образом, для обучения нейрона потребовалось 5 шагов.

Распознавание символа:





**Результат верный.**

Выполнен первый шаг расчётов для пяти букв лабораторной работы, также в excel (второй лист файла):



Из расчёта 0.1 <= n\*alpha <= 1.0, где n = 26, alpha может принимать значения от 0.0038 до 0.038.

Для удобства сверки расчётов было принято решение обнулить весовые коэффициенты на первом шаге, а не задавать им случайные значения в небольшом диапазоне.



Как видно, первого шага в данном случае недостаточно и для достижения заданной точности необходимо провести дополнительные расчёты.

***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;
* neuron.py – класс «Нейрон»;
* adaline.py – наследник класса «Нейрон», работающий по дельта-правилу.
* functions.py – класс, содержащий функции активации.

Структура НС совпадает со структурой НС из лабораторной работы №2.

***Calculating.py***

class Calculator:  
 def \_\_init\_\_(self):  
 self.symbols = [**'A1'**, **'A2'**, **'B1'**, **'B2'**, **'C1'**, **'C2'**, **'D1'**, **'D2'**, **'E1'**, **'E2'**]  
 self.letters\_list = None  
 self.letter\_a1 = None  
 self.letter\_a2 = None  
 self.letter\_b1 = None  
 self.letter\_b2 = None  
 self.letter\_c1 = None  
 self.letter\_c2 = None  
 self.letter\_d1 = None  
 self.letter\_d2 = None  
 self.letter\_e1 = None  
 self.letter\_e2 = None  
 self.letter\_rs = None  
 self.act\_func = None  
 self.neurons = None  
 self.alpha = None  
 self.avgs = None  
 self.rs\_s = None  
 self.rs = None  
 self.ss = None  
 self.m = None  
 self.k = None  
 self.ws = None  
  
 def teaching(self):  
 print(**'**\n**>> ОБУЧЕНИЕ'**)  
 self.\_\_make\_letters\_list()  
 print(**'**\n**>> Создание обучающих выборок'**)  
 self.\_\_make\_all\_m()  
 print(**'**\n**>> Функция для создания нейронов - '** + self.act\_func[0])  
 self.\_\_make\_neurons()  
 print(**'**\n**>> Инициализировано '** + str(len(self.letters\_list)) + **' нейронов'**)  
 self.\_\_work()  
 print(**'**\n**>> Суммарные входные сигналы'**)  
 self.\_\_calculate\_ss()  
 print(**'**\n**>> Среднее арифметическое'**)  
 self.\_\_calculate\_avgs()  
  
 def recognition(self):  
 print(**'**\n**>> РАСПОЗНАВАНИЕ...'**)  
 print(**'**\n**>> Буква для распознавания'**)  
 self.\_\_make\_rs()  
 print(**'**\n**>> Подсчет схожести'**)  
 self.\_\_calculate\_gemini()  
 print(**'**\n**>> Суммарный входной сигнал на разных нейронах'**)  
 self.\_\_calculate\_s\_for\_rs()  
 print(**'**\n**>> Результат'**)  
 return self.\_\_image\_class()  
  
 def \_\_image\_class(self):  
 result = []  
 for i in range(len(self.neurons)):  
 j = i + 1  
 r = self.neurons[i].function[1](self.rs\_s[i], self.avgs[i])  
 tr = r  
 print(str(j) + **': '** + str(tr))  
 result.append(tr)  
 return result  
  
 def \_\_calculate\_s\_for\_rs(self):  
 result = []  
 for i in range(len(self.letters\_list)):  
 j = i + 1  
 s = self.neurons[i].calculate\_s(self.rs)  
 print(str(j) + **'. '** + str(s))  
 result.append(s)  
 self.rs\_s = result  
  
 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 \_\_calculate\_gemini(self):  
 maximum = 0  
 index = 0  
 for i in range(len(self.letters\_list)):  
 j = i + 1  
 s = **'Буква №'** + str(j) + **': '** letter = copy\_and\_insert\_one(self.letters\_list[i][0])  
 tmp1 = self.\_\_help\_gemini(self.rs, letter)  
 s += str(tmp1) + **', '** letter = copy\_and\_insert\_one(self.letters\_list[i][1])  
 tmp2 = self.\_\_help\_gemini(self.rs, letter)  
 s += str(tmp2)  
 max\_tmp = tmp1 + tmp2  
 if max\_tmp >= maximum:  
 maximum = max\_tmp  
 index = i  
 print(s)  
 print(**'Предположительно rs похожа на букву №'** + str(index+1))  
  
 def \_\_make\_rs(self):  
 y = self.act\_func[3]  
 if self.letter\_rs is not None:  
 if y == (1, 0):  
 rs = copy\_and\_insert\_one(self.letter\_rs) self.rs = rs  
 if y == (1, -1):  
 rs = [-1 if not x else 1 for x in self.letter\_rs] rs.insert(0, 1)  
 self.rs = rs  
  
 def \_\_make\_all\_m(self):  
 res = []  
 lst = self.\_\_upd\_by\_functions()  
 for i in range(5):  
 j = i + 1  
 print(**'m-'** + str(j))  
 tmp = self.\_\_make\_m\_by\_function(i, lst)  
 res.append(tmp)  
 self.m = res  
  
 def \_\_make\_m\_by\_function(self, index, lst):  
 result = []  
 y = self.act\_func[3]  
 for i in range(len(lst)):  
 if i == index:  
 tmp1 = lst[i][0], y[0] tmp2 = lst[i][1], y[0]  
 result.append(tmp1) result.append(tmp2)  
 print(tmp1) print(tmp2)  
 else:  
 tmp1 = lst[i][0], y[1] tmp2 = lst[i][1], y[1]  
 result.append(tmp1) result.append(tmp2)  
 print(tmp1) print(tmp2)  
 return result  
  
 def \_\_upd\_by\_functions(self):  
 y = self.act\_func[3]  
 if y == (1, 0): return self.\_\_upd\_binary()  
 if y == (1, -1): return self.\_\_upd\_bipolar()  
  
 def \_\_upd\_binary(self):  
 result = []  
 for el in self.letters\_list:  
 tmp1 = copy\_and\_insert\_one(el[0])  
 tmp2 = copy\_and\_insert\_one(el[1])  
 result.append((tmp1, tmp2))  
 return result  
  
 def \_\_upd\_bipolar(self):  
 result = []  
 for el in self.letters\_list:  
 tmp1 = [-1 if not x else 1 for x in el[0]] tmp1.insert(0, 1)  
 tmp2 = [-1 if not x else 1 for x in el[1]] tmp2.insert(0, 1)  
 result.append((tmp1, tmp2))  
 return result  
  
 def \_\_make\_letters\_list(self):  
 self.letters\_list = [  
 (self.letter\_a1, self.letter\_a2), (self.letter\_b1, self.letter\_b2),  
 (self.letter\_c1, self.letter\_c2), (self.letter\_d1, self.letter\_d2),  
 (self.letter\_e1, self.letter\_e2)]  
  
 def \_\_make\_neurons(self):  
 count\_input = len(self.letter\_a2) + 1  
 res = []  
 tmp = ActivationFunctionConst().bipolar\_delta\_function  
 if self.act\_func[0] != tmp[0]:  
 for i in range(len(self.letters\_list)):  
 neuron = Neuron(self.act\_func, count\_input)  
 res.append(neuron)  
 self.neurons = res  
 else:  
 alpha = copy.copy(self.alpha)  
 for i in range(len(self.letters\_list)):  
 neuron = Adaline(self.act\_func, count\_input, alpha)  
 res.append(neuron)  
 self.neurons = res  
  
 def \_\_work(self):  
 tmp = ActivationFunctionConst().bipolar\_delta\_function  
 if self.act\_func[0] != tmp[0]: self.\_\_work\_with\_simple\_neuron()  
 else: self.\_\_work\_with\_adaline()  
 def \_\_work\_with\_simple\_neuron(self):  
 print(**'**\n**>> Подсчет весовых коэффициентов для нейронов'**)  
 self.\_\_calculate\_ws()  
  
 def \_\_calculate\_avgs(self):  
 res = []  
 tmp = ActivationFunctionConst().bipolar\_delta\_function  
 if self.act\_func[0] != tmp[0]:  
 for i in range(len(self.ss)):  
 j = i + 1  
 tmp = statistics.mean(self.ss[i])  
 res.append(tmp)  
 print(str(j) + **': '**, end=**''**)  
 print(tmp)  
 self.avgs = res  
 else:  
 self.avgs = [0 for \_ in range(5)]  
  
 def \_\_calculate\_ss(self):  
 res = []  
 for i in range(len(self.letters\_list)):  
 j = i + 1  
 print(**'**\n**Нейрон '** + str(j))  
 neuron = self.neurons[i]  
 m\_list = self.m[i]  
 tmp = self.\_\_calculate\_s(m\_list, neuron)  
 res.append(tmp)  
 self.ss = res  
  
 def \_\_calculate\_s(self, m\_list, neuron):  
 s\_list = []  
 for i in range(len(m\_list)):  
 s = neuron.calculate\_s(m\_list[i][0])  
 s\_list.append(s)  
 print(self.symbols[i] + **', s = '** + str(s))  
 return s\_list  
  
 def \_\_calculate\_ws(self):  
 res = []  
 for i in range(len(self.letters\_list)):  
 j = i + 1  
 print(**'**\n**Нейрон '** + str(j) + **' и его весовые коэффициенты:'**)  
 neuron = self.neurons[i]  
 m\_list = self.m[i]  
 tmp = self.\_\_calculate\_w(neuron, m\_list)  
 res.append(tmp)  
 self.ws = res  
  
 def \_\_calculate\_w(self, neuron, m\_list):  
 for i in range(len(m\_list)):  
 j = i + 1  
 y = copy.copy(m\_list[i][1])  
 x\_list = copy.copy(m\_list[i][0])  
 print(**'w['** + str(j) + **']: '**, end=**''**)  
 neuron.correction\_w\_list(x\_list, y, self.k)  
 return neuron.w\_list  
def \_\_work\_with\_adaline(self):  
 print(**'**\n**>> Подсчет весовых коэффициентов для нейронов'**)  
 self.\_\_calculate\_ws\_adaline()  
  
 def \_\_calculate\_ws\_adaline(self):  
 j = 1  
 res = []  
 for i in range(len(self.neurons)):  
 print(**'**\n**Работа с нейроном №'** + str(j) + **'.......'**)  
 neuron = self.neurons[i]  
 m = self.m[i]  
 j += 1  
 self.\_\_help\_adaline(neuron, m)  
 res.append(copy.copy(neuron.w\_list))  
 self.ws = res  
  
 def \_\_help\_adaline(self, neuron, m):  
 flag = False  
 j = 1  
 while not flag:  
 print(**'**\n**Итерация №'** + str(j))  
 neuron.w\_list\_old = copy.copy(neuron.w\_list)  
 for i in range(len(m)):  
 x\_list = m[i][0]  
 y = m[i][1]  
 neuron.correction\_w\_list\_with\_adaline(x\_list, y)  
 flag, s = neuron.check\_stop\_rule()  
 j += 1  
 print(**'**\n**s = '** + str(s) + **'. s <= eps? '** + str(flag))

***Adaline.py***

class Adaline(Neuron):  
 def \_\_init\_\_(self, function, count\_input, alpha=0.1):  
 super().\_\_init\_\_(function, count\_input)   
 self.w\_list = [random.uniform(-0.1, 0.1) for \_ in range(count\_input)]   
 self.w\_list\_old = copy.copy(self.w\_list)   
 self.alpha = alpha  
  
 def calculate\_u\_input(self, x\_list):  
 new\_x\_list = [x\_list[i] for i in range(len(x\_list)) if i != 0]  
 new\_w\_list = [self.w\_list[i] for i in range(len(self.w\_list)) if i != 0]  
 tmp\_list = [new\_x\_list[i] \* new\_w\_list[i] for i in range(len(new\_x\_list))]  
 tmp\_list.insert(0, self.w\_list[0])   
 return sum(tmp\_list)  
  
 def check\_stop\_rule(self, epsilon=0.01):  
 tmp\_list = [math.fabs(self.w\_list[i] - self.w\_list\_old[i]) for i in range(len(self.w\_list))]  
 s = sum(tmp\_list)  
 flag = s <= epsilon  
 return flag, s  
  
 def \_\_correction\_w\_with\_adaline(self, w\_old, u\_in, u\_out, x):  
 w\_old2 = float(copy.copy(w\_old))  
 u\_in2 = float(copy.copy(u\_in))  
 u\_out2 = float(copy.copy(u\_out))  
 x2 = float(copy.copy(x))  
 alpha2 = float(copy.copy(self.alpha))  
 t = u\_out2 - u\_in2  
 m = t \* x2  
 z = m \* alpha2  
 s = z + w\_old2  
 return s  
  
 def correction\_w\_list\_with\_adaline(self, x\_list, u\_out):  
 u\_in = self.calculate\_u\_input(x\_list)  
 print(**'**\n**Uвх = '** + str(u\_in))  
 for i in range(len(self.w\_list)):  
 res = self.\_\_correction\_w\_with\_adaline(self.w\_list[i], u\_in, u\_out, x\_list[i])  
 self.w\_list[i] = copy.copy(float(res))  
 print(self.w\_list)

***Результат работы программы***

Для первого прогона программы было решено задать весовые коэффициенты равными нулю для сравнения с первым шагом расчётов из файла.

>> Подсчет весовых коэффициентов для нейронов

Работа с нейроном №1.......

***Итерация №1***

Uвх = 0

[0.038, 0.038, 0.038, 0.038, 0.038, 0.038, 0.038, -0.038, -0.038, -0.038, -0.038, 0.038, 0.038, 0.038, 0.038, 0.038, 0.038, -0.038, -0.038, -0.038, 0.038, 0.038, 0.038, 0.038, 0.038, 0.038]

Uвх = 0.7600000000000001

[0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.028880000000000003, 0.047119999999999995, -0.047119999999999995, -0.047119999999999995, -0.047119999999999995, -0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.028880000000000003, 0.047119999999999995, -0.047119999999999995, -0.047119999999999995, -0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.047119999999999995, 0.028880000000000003]

Uвх = 0.64144

[-0.015254720000000006, -0.015254720000000006, -0.015254720000000006, -0.015254720000000006, -0.015254720000000006, -0.03349472, -0.015254720000000006, 0.015254720000000006, 0.015254720000000006, 0.015254720000000006, -0.10949471999999999, -0.015254720000000006, -0.015254720000000006, -0.015254720000000006, -0.015254720000000006, -0.03349472, -0.015254720000000006, 0.015254720000000006, 0.015254720000000006, 0.015254720000000006, 0.10949471999999999, -0.015254720000000006, 0.10949471999999999, 0.10949471999999999, 0.10949471999999999, 0.09125472000000001]

Uвх = -0.57825664

[-0.03128096768000001, -0.03128096768000001, -0.03128096768000001, -0.03128096768000001, -0.03128096768000001, -0.04952096768, -0.03128096768000001, 0.03128096768000001, 0.03128096768000001, 0.03128096768000001, -0.12552096767999998, -0.03128096768000001, 0.0007715276799999948, 0.0007715276799999948, 0.0007715276799999948, -0.04952096768, -0.03128096768000001, -0.0007715276799999948, -0.0007715276799999948, -0.0007715276799999948, 0.09346847232, -0.03128096768000001, 0.12552096767999998, 0.12552096767999998, 0.12552096767999998, 0.10728096768]

Uвх = -0.054921661439999936

[-0.06719394454528002, 0.004632009185279999, 0.004632009185279999, -0.06719394454528002, 0.004632009185279999, -0.013607990814719993, 0.004632009185279999, -0.004632009185279999, 0.06719394454528002, -0.004632009185279999, -0.08960799081471998, -0.06719394454528002, 0.03668450454528, 0.03668450454528, 0.03668450454528, -0.08543394454528, -0.06719394454528002, -0.03668450454528, -0.03668450454528, -0.03668450454528, 0.05755549545471999, -0.06719394454528002, 0.16143394454527998, 0.16143394454527998, 0.16143394454527998, 0.07136799081472]

Uвх = -1.1215549497139201

[-0.06257485645615106, 1.292109615103499e-05, 0.009251097274408964, -0.06257485645615106, 0.009251097274408964, -0.018227078903848958, 0.009251097274408964, -0.009251097274408964, 0.06257485645615106, -0.009251097274408964, -0.08498890272559102, -0.06257485645615106, 0.03206541645615104, 0.03206541645615104, 0.03206541645615104, -0.08081485645615104, -0.06257485645615106, -0.03206541645615104, -0.03206541645615104, -0.03206541645615104, 0.06217458354384895, -0.06257485645615106, 0.156814856456151, 0.156814856456151, 0.156814856456151, 0.07598707890384897]

Uвх = 0.005516087205724232

[-0.10078446776996858, -0.03819669021766648, -0.028958514039408554, -0.10078446776996858, -0.028958514039408554, -0.05643669021766648, 0.047460708588226486, 0.028958514039408554, 0.02436524514233354, 0.028958514039408554, -0.046779291411773505, -0.02436524514233354, 0.07027502776996855, -0.006144194857666478, 0.07027502776996855, -0.04260524514233352, -0.02436524514233354, 0.006144194857666478, -0.07027502776996855, 0.006144194857666478, 0.10038419485766648, -0.02436524514233354, 0.1950244677699685, 0.11860524514233349, 0.1950244677699685, 0.11419669021766649]

Uвх = -1.1205119531699594

[-0.09620501354951012, -0.03361723599720802, -0.024379059818950097, -0.09620501354951012, -0.024379059818950097, -0.05185723599720802, 0.052040162808684946, 0.024379059818950097, 0.028944699362792002, 0.024379059818950097, -0.042199837191315044, -0.01978579092187508, 0.0656955735495101, -0.0015647406372080192, 0.0656955735495101, -0.03802579092187506, -0.028944699362792002, 0.0015647406372080192, -0.0656955735495101, 0.0015647406372080192, 0.09580474063720802, -0.028944699362792002, 0.19044501354951004, 0.12318469936279194, 0.19044501354951004, 0.10961723599720803]

Uвх = -0.22420032791020233

[-0.12568540108892243, -0.06309762353662034, 0.005101327720462217, -0.0667246260100978, -0.053859447358362414, -0.08133762353662033, 0.022559775269272633, 0.053859447358362414, -0.0005356881766203116, 0.053859447358362414, -0.01271944965190273, -0.04926617846128739, 0.03621518601009778, 0.027915646902204293, 0.09517596108892241, -0.008545403382462746, -0.05842508690220431, -0.027915646902204293, -0.09517596108892241, 0.031045128176620335, 0.12528512817662033, -0.05842508690220431, 0.21992540108892236, 0.15266508690220426, 0.16096462601009773, 0.08013684845779571]

Uвх = -0.9602980094343195

[-0.1271940767304183, -0.0646062991781162, 0.006610003361958074, -0.06521595036860195, -0.05536812299985827, -0.08284629917811619, 0.021051099627776775, 0.05536812299985827, -0.002044363818116169, 0.05536812299985827, -0.014228125293398588, -0.05077485410278325, 0.03470651036860192, 0.02942432254370015, 0.09668463673041827, -0.007036727740966888, -0.05993376254370017, -0.026406971260708435, -0.09668463673041827, 0.03255380381811619, 0.1267938038181162, -0.05993376254370017, 0.22143407673041823, 0.15417376254370013, 0.15945595036860186, 0.07862817281629986]

***s = 1.784526341216327. s <= eps? False***

\*\*\*\*

***Итерация №13***

Uвх = 1.0002843554731335

[-0.13077658337352557, 0.029865504377628714, 0.04548359509896977, -0.05783477574602602, -0.027458212528529653, -0.012686848225358562, 0.06719640508425614, 0.027458212528529653, -0.03381347803217347, 0.027458212528529653, -0.02484140535292608, -0.03612196576073958, 0.03508922339409435, 0.01638277724339432, 0.10803103102159378, -0.00573251073622742, -0.039128329595325965, -0.018281452340661072, -0.10803103102159378, 0.056559030384105154, 0.22343732354605358, -0.039128329595325965, 0.29765487653547373, 0.20600662275727447, 0.22471306890797446, 0.02151862855383306]

Uвх = 0.9938048730811834

[-0.13054116855061054, 0.030100919200543745, 0.0457190099218848, -0.057599360923110986, -0.027222797705614622, -0.012922263048273595, 0.06743181990717118, 0.027222797705614622, -0.034048892855088504, 0.027222797705614622, -0.025076820175841112, -0.03588655093782455, 0.03532463821700938, 0.01661819206630935, 0.10826644584450881, -0.005967925559142452, -0.038892914772410934, -0.018516867163576103, -0.10826644584450881, 0.05632361556119012, 0.22367273836896862, -0.038892914772410934, 0.2978902913583888, 0.2062420375801895, 0.2249484837308895, 0.02128321373091803]

Uвх = -0.9935154611664128

[-0.13078758102628685, 0.02985450672486743, 0.04547259744620849, -0.0578457733987873, -0.027469210181290938, -0.01316867552394991, 0.06718540743149486, 0.027469210181290938, -0.03380248037941219, 0.027469210181290938, -0.025323232651517428, -0.03613296341350086, 0.03507822574133307, 0.016371779590633035, 0.1080200333688325, -0.006214338034818767, -0.03913932724808725, -0.018270454687899788, -0.1080200333688325, 0.056570028036866435, 0.22391915084464492, -0.03913932724808725, 0.2981367038340651, 0.2064884500558658, 0.2251948962065658, 0.021529626206594345]

Uвх = -1.010464881286036

[-0.1303899155374175, 0.030252172213736796, 0.04587026293507786, -0.05744810790991793, -0.02707154469242157, -0.012771010035080544, 0.06758307292036422, 0.02707154469242157, -0.03420014586828156, 0.02707154469242157, -0.02492556716264806, -0.035735297924631496, 0.0346805602524637, 0.015974114101763668, 0.10762236787996313, -0.0058166725459494, -0.03874166175921788, -0.01787278919903042, -0.10762236787996313, 0.0569676935257358, 0.22431681633351427, -0.03874166175921788, 0.2977390383451957, 0.20609078456699645, 0.22479723071769644, 0.02113196071772498]

Uвх = -1.0074502507283538

[-0.13010680600974006, 0.029969062686059353, 0.04558715340740041, -0.057164998382240484, -0.027354654220099014, -0.013054119562757989, 0.06729996339268678, 0.027354654220099014, -0.034483255395959006, 0.027354654220099014, -0.025208676690325504, -0.03545218839695405, 0.034397450724786255, 0.015691004574086224, 0.10733925835228568, -0.005533563018271955, -0.03845855223154043, -0.017589679671352977, -0.10733925835228568, 0.05725080305341325, 0.2245999258611917, -0.03845855223154043, 0.29745592881751826, 0.20580767503931902, 0.224514121190019, 0.021415070245402422]

Uвх = -0.9888604481098108

[-0.13053010898156725, 0.030392365657886545, 0.04516385043557322, -0.05758830135406767, -0.027777957191926207, -0.012630816590930799, 0.06687666042085959, 0.027777957191926207, -0.03405995242413182, 0.027777957191926207, -0.025631979662152696, -0.03587549136878124, 0.03482075369661344, 0.016114307545913416, 0.10776256132411287, -0.0059568659900991455, -0.03888185520336762, -0.01801298264318017, -0.10776256132411287, 0.05682750008158606, 0.22417662288936452, -0.03888185520336762, 0.29787923178934544, 0.2062309780111462, 0.2249374241618462, 0.02099176727357523]

Uвх = -0.9990356021664245

[-0.13056675609924312, 0.030355718540210675, 0.04512720331789735, -0.05762494847174354, -0.027814604309602076, -0.012667463708606667, 0.06691330753853546, 0.027814604309602076, -0.03409659954180769, 0.027814604309602076, -0.025595332544476827, -0.03583884425110537, 0.03485740081428931, 0.016077660428237547, 0.10779920844178874, -0.0059202188724232765, -0.03884520808569175, -0.0179763355255043, -0.10779920844178874, 0.05686414719926193, 0.2242132700070404, -0.03884520808569175, 0.2979158789070213, 0.20619433089347033, 0.22497407127952207, 0.0210284143912511]

Uвх = -1.000870603484937

[-0.13053367316681552, 0.030388801472638283, 0.04516028625032496, -0.05759186553931593, -0.02778152137717447, -0.01263438077617906, 0.06694639047096307, 0.02778152137717447, -0.034063516609380076, 0.02778152137717447, -0.02556224961204922, -0.035805761318677756, 0.0348243178818617, 0.016110743360665154, 0.10776612550936113, -0.005887135939995668, -0.03887829101811936, -0.018009418457931907, -0.10776612550936113, 0.05683106426683432, 0.2241801870746128, -0.03887829101811936, 0.2978827959745937, 0.20622741382589793, 0.22494098834709447, 0.020995331458823492]

Uвх = -0.9869355575456557

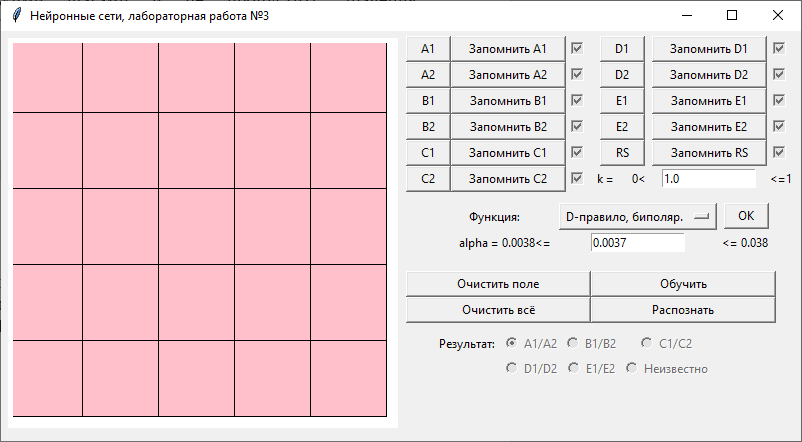
[-0.1310301219800806, 0.0298923526593732, 0.04565673506359005, -0.05709541672605085, -0.028277970190439553, -0.013130829589444145, 0.06644994165769798, 0.028277970190439553, -0.03455996542264516, 0.028277970190439553, -0.025065800798784135, -0.03630221013194284, 0.03432786906859662, 0.016607192173930238, 0.10826257432262622, -0.0053906871267305825, -0.039374739831384446, -0.01850586727119699, -0.10826257432262622, 0.0573275130800994, 0.22467663588787787, -0.039374739831384446, 0.2983792447878588, 0.206723862639163, 0.2244445395338294, 0.02049888264555841]

Uвх = -1.0129630937457221

[-0.13053752441774316, 0.03038495022171064, 0.04516413750125261, -0.057588014288388285, -0.02778537262810211, -0.012638232027106703, 0.06694253922003542, 0.02778537262810211, -0.03406736786030772, 0.02778537262810211, -0.024573203236446694, -0.0358096125696054, 0.034820466630934056, 0.016114594611592797, 0.10776997676028878, -0.005883284689068024, -0.03888214226904701, -0.018998464833534432, -0.10776997676028878, 0.056834915517761965, 0.22418403832554043, -0.03888214226904701, 0.2978866472255214, 0.20623126507682557, 0.22493713709616683, 0.02099148020789585]

***s = 0.008092569824872716. s <= eps? True***

Как видно, значения первой итерации в программе совпали со значениями расчёта из эксель-файла. Всего при значении alpha = 0.038, epsilon = 0.01 потребовалось 13 итераций для обучения одного нейрона, а также близкое количество итераций для других нейронов – 11-13 шагов.

На рисунке представлены результаты экспериментов для разных значений alpha в подходящем диапазоне, значений epsilon, а также для весовых коэффициентов, равных 0 на первом шаге и для тех, что заданы случайным образом. Можно сделать вывод, что чем выше требуется точность, тем больше будет требоваться итераций для её достижения. В свою очередь, чем меньше коэффициент обучения, тем больше итераций будет выполнено для того, чтобы достигнуть заданной точности. Однако, использование таких значений для обучающего коэффициента позволит двигаться к «цели» более маленькими шагами и не пропустить значения, удовлетворяющего условию останова.

Выбор коэффициента обучения меньшего, чем допустимый диапазон – 0.0037 (но очень близкого к его границе) увеличило количество итераций до 267, однако результат был получен. Правда, совершенно неверный. Неверное обучение привело к тому, что символ был опознан сразу несколькими классами.

Кстати, это же изображение было использовано во второй лабораторной работе, где сеть определила его класс, как неизвестный. Обучим сеть при alpha = 0.0038 и epsilon = 0.01 и сравним результаты.

|  |  |
| --- | --- |
| **Лог из лабораторной работы №2** | **Лог из лабораторной работы №3** |
| >> Подсчет схожести  Буква №1: 19, 16  Буква №2: 15, 16  Буква №3: 13, 15  Буква №4: 10, 14  Буква №5: 14, 14  Предположительно rs похожа на букву №1  >> Суммарный входной сигнал на разных нейронах  1. -76.0  2. -84.0  3. -90.0  4. -98.0  5. -90.0  >> Результат  1: 0  2: 0  3: 0  4: 0  5: 0 | >> Подсчет схожести  Буква A: 19, 16  Буква B: 15, 16  Буква C: 13, 15  Буква D: 10, 14  Буква E: 14, 14  Предположительно rs похожа на букву №1  >> Суммарный входной сигнал на разных нейронах  1. 0.9000467774273954  2. -0.3659237864384436  3. -0.8924127124305636  4. -0.5577912570043178  5. -0.4361219619777615  >> Результат  1: 1  2: -1  3: -1  4: -1  5: -1 |

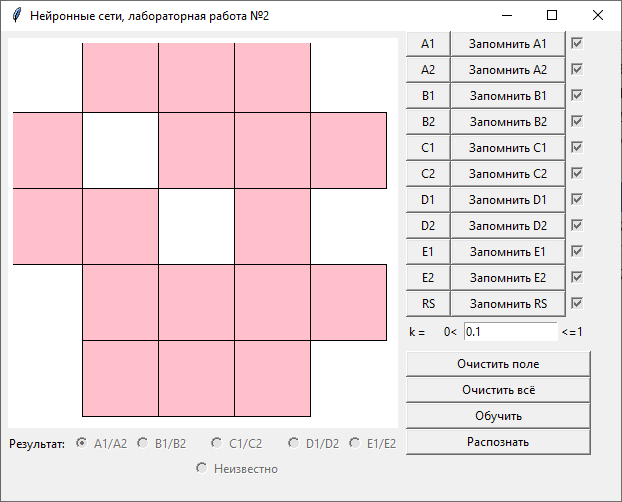
Как видно, в обоих случаях при подсчете схожести предполагалось, что введенный символ будет определен как первый класс символов – класс А, однако при обучении с использованием бинарной функции по правилу Хебба определить класс символа не удалось.

В суммарных входных сигналах тоже можно проследить связь – для первого класса значение максимально в обоих случаях, (-76 и 0.9000467774273954), и хотя в ЛР№2 использовались улучшения правила, можно вспомнить, что для получения верного результата не хватило всего «немного». При среднем значении -53, значение -76 хоть и было ближайшим, но не смогло преодолеть порог.



В свою очередь, в ЛР№3 суммарный входной сигнал введенного символа для класса А оказался достаточно близок к единице, что позволило отнести изображение именно к этому классу.

Ещё один случай изображения из ЛР№2

Можно увидеть довольно схожую картину: по правилу (бинарному в первом случае и биполярному во втором) изображения относятся к 1, 3 и 4 классам.

Однако, обе программы отнесли изображение к классу №1 (подсчёт схожести).

В лабораторной работе №2 значения суммарных входных сигналов достаточно близки и велики, а в ЛР№3, в свою очередь, можно точно сказать, что введенный символ относится к первому классу. На это указывает значение 0.7431171308800344, максимально приближенное к единице.

***Таким образом, использование дельта-правила позволяет дать более точную оценку для распознаваемого изображения.***

|  |  |
| --- | --- |
| **Лог из лабораторной работы №2** | **Лог из лабораторной работы №3** |
| >> Подсчет схожести  Буква №1: 13, 16  Буква №2: 11, 14  Буква №3: 11, 15  Буква №4: 12, 14  Буква №5: 12, 12  Предположительно rs похожа на букву №1  >> Суммарный входной сигнал на разных нейронах  1. -5.0  2. -5.800000000000001  3. -6.000000000000001  4. -6.4  5. -6.2  >> Результат  1: 1  2: 0  3: 1  4: 1  5: 0 | >> Подсчет схожести  Буква №1: 13, 16  Буква №2: 11, 14  Буква №3: 11, 15  Буква №4: 12, 14  Буква №5: 12, 12  Предположительно rs похожа на букву №1  >> Суммарный входной сигнал на разных нейронах  1. 0.7431171308800344  2. -0.5791636212037463  3. 0.180155439616725  4. 0.2082296075839385  5. -0.006775446242550179  >> Результат  1: 1  2: -1  3: 1  4: 1  5: -1 |