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 №1

по дисциплине «Нейронные сети»

тема: «Использование одного нейрона. Функции активации»

Выполнила: ст. группы МИВТ-201

Браткова Ирина Олеговна

Белгород 2020 г

***Цель работы***: закрепление теоретических знаний, получение практических навыков использования одного нейрона, исследование правила Хебба, исследование функций активации.

***Индивидуальные задания:***

* разработать по два черно-белых изображения первых букв Вашего имени и отчества (буквы должны быть разными);
* разработать нейрокомпьютерную сеть для обучения разработанным классам изображений;
* реализовать алгоритм обучения нейрона на основе правила Хебба, при этом использовать как бинарную так и биполярную функции активации;
* исследовать возможности нейрона по распознаванию изображений.

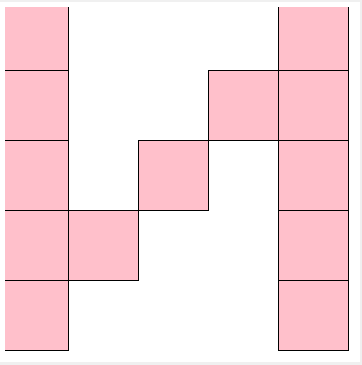
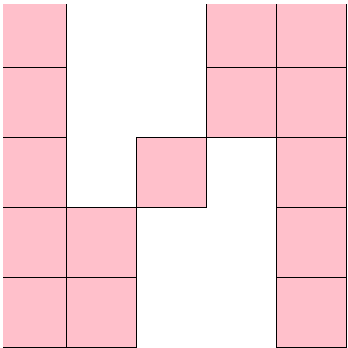
***Дополнительные задания (не обязательные):***

* исследовать возможности использования в нейроне других функций активации;
* предложить и исследовать способы «улучшения» правила Хебба.

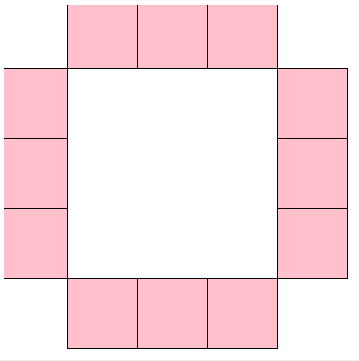
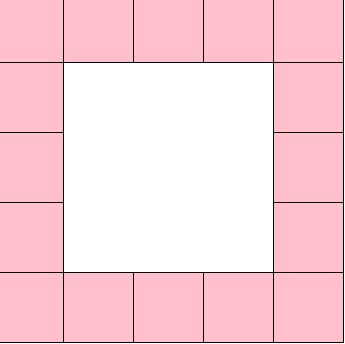
***Выполнение***

Для выполнения работы были использованы буквы «И» и «О».

Буквы «И» – первый класс изображений (А).

Буквы «О» – второй класс изображений (B).

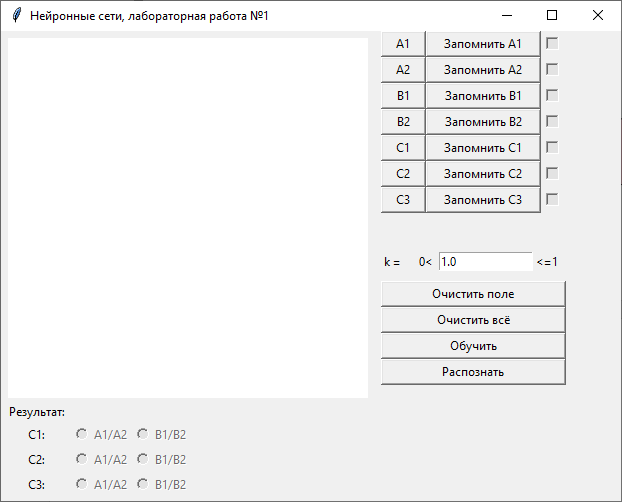
 

Расчет значений (в Excel)

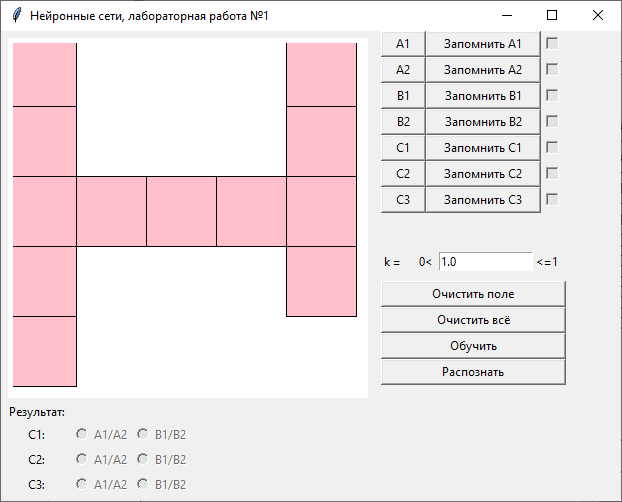
    

(в программе будут использованы изображения, для которых был проведён данный расчёт)

Для разработки был выбран язык Python, разработан следующий интерфейс:



На скриншоте видно поле для отображения/создания букв:



При запуске кнопки «А1», «А2», «В1» и «В2» инициализированы буквами «И» и «О», соответственно. 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 «А1», «А2», «В1» или «В2» (после запоминания) отобразит на поле сохраненную букву.

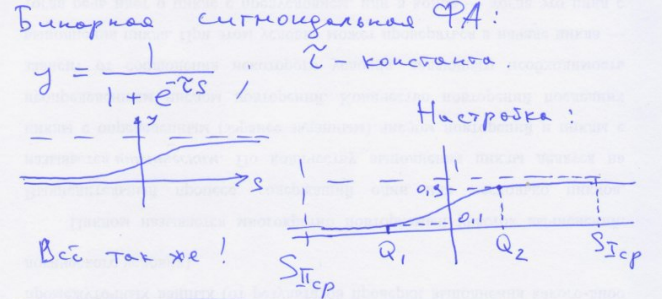
Работа с буквами С1-С3 ведётся аналогичным образом, однако, в отличии от буков А1-А2, В1-В2, в обучении они не участвуют.

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

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 буков С1-С3.

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

***Функцией варианта является бинарная сигмоидальная функция активации.***



***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»;
* functions.py – класс, содержащий функции активации.

Первый и второй модуль не представляют интереса для данной лабораторной работы, поскольку их реализация полностью зависит от языка программирования и необходима для вывода информации на экран.

***Functions.py***

В конструкторе инициализируются значения функций активации – бинарной, биполярной и сигмоидальной. Функция – структура, состоящая из названия, самой функции, а также правила Хебба.

class ActivationFunctions(object):  
  
 def \_\_init\_\_(self):  
 self.binary\_function = (**'Бинарная'**, self.binary, self.hebb\_for\_binary)  
 self.bipolar\_function = (**'Биполярная'**, self.bipolar, self.hebb\_for\_bipolar)  
 self.sigmoid\_function = (**'Сигмоидальная'**, self.sigmoid, self.hebb\_for\_binary)  
  
 def sigmoid(self, s, t):  
 return 1 / (1 + math.pow(math.e, -t \* s))  
  
 def bipolar(self, s, div):  
 return (1, **'A'**) if s > div else (-1, **'B'**)

def binary(self, s, div):  
 return (1, **'A'**) if s > div else (0, **'B'**)  
  
 def hebb\_for\_bipolar(self, w, x, y, k=1.0):  
 return w + x \* y \* k  
  
 def hebb\_for\_binary(self, w, x, y, k=1.0):  
 delta\_w = 0  
 if x == 1 and y == 1:  
 delta\_w = 1  
 elif x == 0:  
 delta\_w = 0  
 elif x != 0 and y == 0:  
 delta\_w = -1  
 return w + delta\_w \* k

***Neuron.py***

Данный класс инициализируется функцией активации, количеством входов и количеством выходов (по умолчанию – 1). В конструкторе также происходит обнуление весовых коэффициентов.

class Neuron:  
 def \_\_init\_\_(self, function, count\_input, count\_output=1):  
 self.count\_input = count\_input  
 self.count\_output = count\_output  
 self.w\_list = [0 for \_ in range(count\_input)]  
 self.function = function  
  
 *# Возвращает значение входного суммарного сигнала для x\_list на основе весовых коэффициентов w\_list* def calculate\_s(self, x\_list):  
 res = 0  
 for i in range(self.count\_input):  
 res += x\_list[i] \* self.w\_list[i]  
 return res  
  
 *# Корректировка весовых коэффициентов для x\_list со значением y и коэффициентом плавности k* def correction\_w\_list(self, x\_list, y, k):  
 for i in range(len(self.w\_list)):  
 res = self.\_\_correction\_w(self.w\_list[i], x\_list[i], y, k)  
 self.w\_list[i] = copy.copy(res)  
 print(self.w\_list)  
  
 *# Подсчет значения w для x и y с указанным k с помощью заранее определенной функции активации* def \_\_correction\_w(self, w, x, y, k):  
 return self.function[2](w, x, y, k)

***Calculating.py***

Класс, в котором ведется основная работа по обучению и распознаванию.

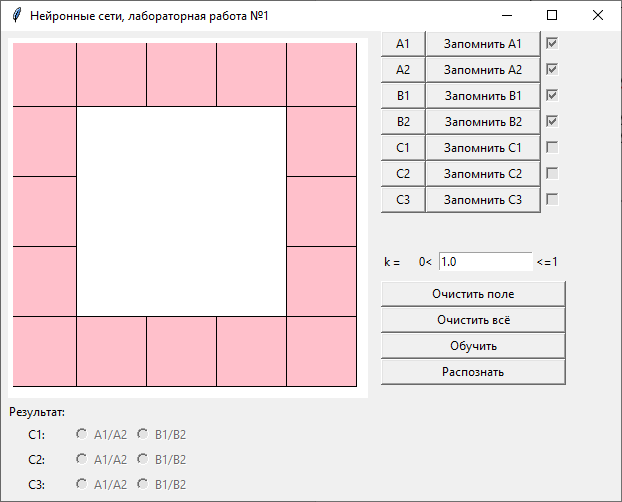
class Calculating:  
 def \_\_init\_\_(self):  
 *# Буквы* self.letter\_a1 = None  
 self.letter\_a2 = None  
 self.letter\_b1 = None  
 self.letter\_b2 = None  
 self.letter\_c1 = None  
 self.letter\_c2 = None  
 self.letter\_c3 = None  
  
 *# Кол-во входов* self.input\_count = None  
  
 *# Обучающие выборки* self.m\_binary\_list = None  
 self.m\_bipolar\_list = None  
  
 *# Нейроны* self.binary\_neuron = None  
 self.bipolar\_neuron = None  
 self.sigmoid\_neuron = None  
  
 *# Листы C1-C3* self.c\_list\_binary = None  
 self.c\_list\_bipolar = None  
  
 *# Коэффициенты* self.w\_binary\_list = None self.w\_bipolar\_list = None  
  
 *# Коэффициент плавности* self.k = None  
  
 *# Лист суммарных входных сигналов* self.s\_binary\_list = []  
 self.s\_bipolar\_list = []  
 self.s\_c\_binary\_list = []  
  
 *# Значения Q1 и Q2 для бинарного/биполярного метода* self.q\_binary = None  
 self.q\_bipolar = None  
  
 *# Делители* self.divider\_binary = None  
 self.divider\_bipolar = None  
  
 *# T (для функции варианта)* self.t = None  
  
 *# Средние значения* self.avg\_binary = None  
 self.avg\_bipolar = None

def teaching(self):  
 print(**'**\n**>> ОБУЧЕНИЕ...'**)  
 self.input\_count = len(self.letter\_a1) + 1  
 self.\_\_make\_ms()  
  
 print(**'**\n**Созданые обучающие выборки: '**)  
 print(**'Бинарная: '**, end=**''**)  
 print(self.m\_binary\_list)  
 print(**'Биполярная: '**, end=**''**)  
 print(self.m\_bipolar\_list)  
  
 print(**'**\n**Созданы два нейрона на основе биполярной и бинарной функций '**)  
 self.\_\_make\_neuron()  
 print(**'**\n**Обучение бинарного нейрона: '**)  
 self.w\_binary\_list = self.\_\_calculate\_w(self.binary\_neuron, self.m\_binary\_list)  
 print(**'**\n**Обучение биполярного нейрона: '**)  
 self.w\_bipolar\_list = self.\_\_calculate\_w(self.bipolar\_neuron, self.m\_bipolar\_list)  
  
 print(**'**\n**Суммарные входные сигналы:'**)  
 self.\_\_calculate\_ss()  
 self.avg\_binary = (self.s\_binary\_list[0][0] + self.s\_binary\_list[1][0] + self.s\_binary\_list[2][0] +  
 self.s\_binary\_list[3][0]) / 4  
 self.w\_binary\_list[0] -= self.avg\_binary  
 print(**'**\n**avg (бинарное) = '** + str(self.avg\_binary))  
  
 self.avg\_bipolar = (self.s\_bipolar\_list[0][0] + self.s\_bipolar\_list[1][0] + self.s\_bipolar\_list[2][0] +  
 self.s\_bipolar\_list[3][0]) / 4  
 self.w\_bipolar\_list[0] -= self.avg\_bipolar  
 print(**'**\n**avg (биполярное) = '** + str(self.avg\_bipolar))  
 self.\_\_calculate\_ss()  
  
 print(**'**\n**Q1 и Q2:'**)  
 self.\_\_calculate\_qs()  
  
 print(**'**\n**!Далее работа продолжается с сигмоидальной бинарной функцией активации!'**)  
 t1 = self.\_\_calculate\_t(0.1, self.q\_binary[0])  
 t2 = self.\_\_calculate\_t(0.9, self.q\_binary[1])  
 print(**'**\n**Значение t1 = '** + str(t1))  
 print(**'Значение t2 = '** + str(t2))  
 self.t = t1

def recognition(self):  
 print(**'**\n**>> РАСПОЗНАВАНИЕ...'**)  
 print(**'**\n**!Работа ведётся с сигмоидальной функцией активации!'**)  
  
 self.c\_list\_binary = self.\_\_make\_c\_lists\_binary()  
 if not self.c\_list\_binary:  
 print(**'Не введено ни одной буквы С'**)  
 print(**'**\n**Буква(ы) С в бинарном представлении: '**)  
 print(self.c\_list\_binary)  
  
 print(**'**\n**>> Сходство: '**, end=**''**)  
 self.\_\_calculate\_gemini(self.c\_list\_binary, self.m\_binary\_list)  
  
 self.sigmoid\_neuron = copy.deepcopy(self.binary\_neuron)  
 self.sigmoid\_neuron.function = ActivationFunctionConst().sigmoid\_function  
 print(**'**\n**>> Выбрана функция варианта - '** + self.sigmoid\_neuron.function[0])  
  
 print(**'**\n**>> Входной(ые) суммарные сигналы С: '**)  
 self.s\_c\_binary\_list = self.\_\_calculate\_c\_s()  
  
 print(**'**\n**>> Согласно функции активации, изображение(я) принадлежит(ат) классу(ам): '**)  
 result = self.\_\_classify(self.\_\_get\_image\_class())  
 print(result)  
 return result  
  
def \_\_calculate\_c\_s(self):  
 res = self.\_\_calculate\_s(self.c\_list\_binary, self.sigmoid\_neuron)  
 return res  
  
def \_\_classify(self, list):  
 result = []  
 for r in list:  
 tmp = self.binary\_neuron.function[1](r[0], 0)  
 result.append((tmp, r[1]))  
 return result  
  
def \_\_get\_image\_class(self):  
 result = []  
 for i in range(len(self.c\_list\_binary)):  
 res = self.sigmoid\_neuron.function[1](self.s\_c\_binary\_list[i][0], self.t)  
 result.append((round(res), self.s\_c\_binary\_list[i][1]))  
 return result  
  
def \_\_calculate\_gemini(self, list\_c, list\_letters):  
 for row in list\_c:  
 print(**'**\n**'** + row[1] + **': '**)  
 for letter in list\_letters:  
 tmp = self.\_\_help\_gemini(row[0], letter[0])  
 print(letter[1] + **' =>> '** + str(tmp))  
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 copy\_and\_insert\_one(self, list):  
 result = copy.deepcopy(list)  
 result.insert(0, 1)  
 return result  
def \_\_make\_ms(self):  
 self.m\_binary\_list = self.\_\_make\_m\_by\_binary()  
 self.m\_bipolar\_list = self.\_\_make\_m\_by\_bipolar()  
def \_\_make\_m\_by\_binary(self):  
 a1 = self.copy\_and\_insert\_one(self.letter\_a1)  
 a2 = self.copy\_and\_insert\_one(self.letter\_a2)  
 b1 = self.copy\_and\_insert\_one(self.letter\_b1)  
 b2 = self.copy\_and\_insert\_one(self.letter\_b2)  
 return [(a1, **'A1'**, 1), (a2, **'A2'**, 1), (b1, **'B1'**, 0), (b2, **'B2'**, 0)]  
def \_\_make\_m\_by\_bipolar(self):  
 a1 = [-1 if not el else 1 for el in self.letter\_a1]  
 a1.insert(0, 1)  
 a2 = [-1 if not el else 1 for el in self.letter\_a2]  
 a2.insert(0, 1)  
 b1 = [-1 if not el else 1 for el in self.letter\_b1]  
 b1.insert(0, 1)  
 b2 = [-1 if not el else 1 for el in self.letter\_b2]  
 b2.insert(0, 1)  
 return [(a1, **'A1'**, 1), (a2, **'A2'**, 1), (b1, **'B1'**, -1), (b2, **'B2'**, -1)]  
def \_\_make\_c\_lists(self):  
 self.c\_list\_binary = self.\_\_make\_c\_lists\_binary()  
 self.c\_list\_bipolar = self.\_\_make\_c\_lists\_bipolar()  
def \_\_make\_c\_lists\_bipolar(self):  
 result\_list = []  
 if self.letter\_c1 is not None:  
 c1 = [-1 if not el else 1 for el in self.letter\_c1]  
 c1.insert(0, 1)  
 result\_list.append((c1, **'C1'**))  
 if self.letter\_c2 is not None:  
 c2 = [-1 if not el else 1 for el in self.letter\_c2]  
 c2.insert(0, 1)  
 result\_list.append((c2, **'C2'**))  
 if self.letter\_c3 is not None:  
 c3 = [-1 if not el else 1 for el in self.letter\_c3]  
 c3.insert(0, 1)  
 result\_list.append((c3, **'C3'**))  
 return result\_list  
def \_\_make\_c\_lists\_binary(self):  
 result\_list = []  
 if self.letter\_c1 is not None:  
 c1 = self.copy\_and\_insert\_one(self.letter\_c1) *# copy.copy(self.letter\_c1)* result\_list.append((c1, **'C1'**))  
 if self.letter\_c2 is not None:  
 c2 = self.copy\_and\_insert\_one(self.letter\_c2) *# copy.copy(self.letter\_c2)* result\_list.append((c2, **'C2'**))  
 if self.letter\_c3 is not None:  
 c3 = self.copy\_and\_insert\_one(self.letter\_c3) *# copy.copy(self.letter\_c3)* result\_list.append((c3, **'C3'**))  
 return result\_list  
def \_\_make\_neuron(self):  
 functions = ActivationFunctionConst()  
 self.binary\_neuron = Neuron(functions.binary\_function, self.input\_count)  
 self.bipolar\_neuron = Neuron(functions.bipolar\_function, self.input\_count)  
def \_\_calculate\_w(self, neuron, m\_list):  
 for i in range(len(m\_list)):  
 j = i + 1  
 y = copy.copy(m\_list[i][2])  
 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 \_\_calculate\_ss(self):  
 print(**'Бинарные: '**)  
 self.s\_binary\_list = self.\_\_calculate\_s(self.m\_binary\_list, self.binary\_neuron)  
 print(**'Биполярные: '**)  
 self.s\_bipolar\_list = self.\_\_calculate\_s(self.m\_bipolar\_list, self.bipolar\_neuron)  
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, m\_list[i][1]))  
 print(m\_list[i][1] + **', s = '** + str(s))  
 return s\_list  
  
def \_\_calculate\_qs(self):  
 self.q\_binary, self.divider\_binary = self.\_\_calculate\_q1\_q1(self.s\_binary\_list)  
 print(**'Бинарные: '**, end=**''**)  
 print(**'s2(cp) = '** + str(self.divider\_binary[1]), end=**''**)  
 print(**' | q1 = '** + str(self.q\_binary[0]), end=**''**)  
 print(**' | q2 = '** + str(self.q\_binary[1]), end=**''**)  
 print(**' | s1(cp) = '** + str(self.divider\_binary[0]), end=**''**)  
 self.q\_bipolar, self.divider\_bipolar = self.\_\_calculate\_q1\_q1(self.s\_bipolar\_list)  
 print(**'**\n**Биполярные: '**, end=**''**)  
 print(**'s2(cp) = '** + str(self.divider\_bipolar[1]), end=**''**)  
 print(**' | q1 = '** + str(self.q\_bipolar[0]), end=**''**)  
 print(**' | q2 = '** + str(self.q\_bipolar[1]), end=**''**)  
 print(**' | s1(cp) = '** + str(self.divider\_bipolar[0]), end=**''**)  
  
def \_\_calculate\_q1\_q1(self, s\_list):  
 divider\_a = (s\_list[0][0] + s\_list[1][0]) / 2 *# s1ср* divider\_b = (s\_list[2][0] + s\_list[3][0]) / 2 *# s1ср* result = divider\_a - divider\_b  
 q = result / 3  
 center = (divider\_b + divider\_a) / 2  
 tmp = q / 2  
 q1 = center - tmp  
 q2 = center + tmp  
 return (q1, q2), (divider\_a, divider\_b)  
  
def \_\_calculate\_t(self, y, q): ln = -math.log((1 / y) - 1) return ln / q

***Результат работы программы на примере буков «И» и «О»***

\*для удобства ниже представлены не экранные копии, а текстовые копии работы программы (лог))



>> ОБУЧЕНИЕ...

Созданые обучающие выборки:

Бинарная:

[([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], 'A1', 1),

([1, 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], 'A2', 1),

([1, 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], 'B1', 0),

([1, 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], 'B2', 0)]

Биполярная:

[([1, 1, -1, -1, -1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 1, 1, 1, -1, -1, 1, 1, -1, -1, -1, 1], 'A1', 1), ([1, 1, -1, -1, 1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 1, 1, 1, -1, -1, 1, 1, 1, -1, -1, 1], 'A2', 1), ([1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, -1, 1, 1, 1, -1], 'B1', -1), ([1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, 1], 'B2', -1)]

Созданы два нейрона на основе биполярной и бинарной функций

Обучение бинарного нейрона:

w[1]:

[0.8, 0.8, 0.0, 0.0, 0.0, 0.8, 0.8, 0.0, 0.0, 0.8, 0.8, 0.8, 0.0, 0.8, 0.0, 0.8, 0.8, 0.8, 0.0, 0.0, 0.8, 0.8, 0.0, 0.0, 0.0, 0.8]

w[2]:

[1.6, 1.6, 0.0, 0.0, 0.8, 1.6, 1.6, 0.0, 0.0, 1.6, 1.6, 1.6, 0.0, 1.6, 0.0, 1.6, 1.6, 1.6, 0.0, 0.0, 1.6, 1.6, 0.8, 0.0, 0.0, 1.6]

w[3]:

[0.8, 1.6, -0.8, -0.8, 0.0, 1.6, 0.8, 0.0, 0.0, 1.6, 0.8, 0.8, 0.0, 1.6, 0.0, 0.8, 0.8, 1.6, 0.0, 0.0, 0.8, 1.6, 0.0, -0.8, -0.8, 1.6]

w[4]:

[0.0, 0.8, -1.6, -1.6, -0.8, 0.8, 0.0, 0.0, 0.0, 1.6, 0.0, 0.0, 0.0, 1.6, 0.0, 0.0, 0.0, 1.6, 0.0, 0.0, 0.0, 0.8, -0.8, -1.6, -1.6, 0.8]

Обучение биполярного нейрона:

w[1]:

[1.0, 1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, -1.0, 1.0]

w[2]:

[2.0, 2.0, -2.0, -2.0, 0.0, 2.0, 2.0, -2.0, -2.0, 2.0, 2.0, 2.0, -2.0, 2.0, -2.0, 2.0, 2.0, 2.0, -2.0, -2.0, 2.0, 2.0, 0.0, -2.0, -2.0, 2.0]

w[3]:

[1.0, 3.0, -3.0, -3.0, -1.0, 3.0, 1.0, -1.0, -1.0, 3.0, 1.0, 1.0, -1.0, 3.0, -1.0, 1.0, 1.0, 3.0, -1.0, -1.0, 1.0, 3.0, -1.0, -3.0, -3.0, 3.0]

w[4]:

[0.0, 2.0, -4.0, -4.0, -2.0, 2.0, 0.0, 0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 2.0, -2.0, -4.0, -4.0, 2.0]

Суммарные входные сигналы:

Бинарные: A1, s = 10.0 A2, s = 8.0 B1, s = -10.0 B2, s = -6.0

Биполярные: A1, s = 40.0 A2, s = 32.0 B1, s = -40.0 B2, s = -24.0

avg (бинарное) = 0.5

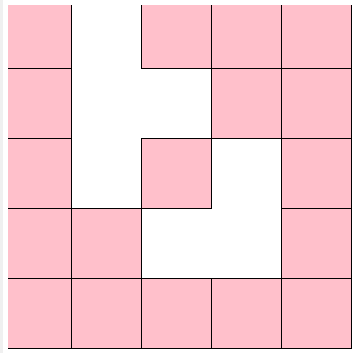
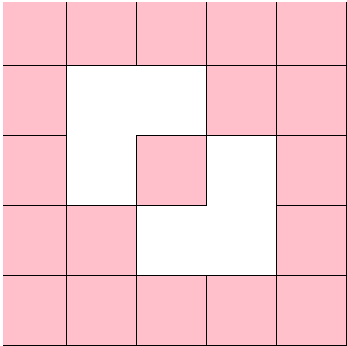
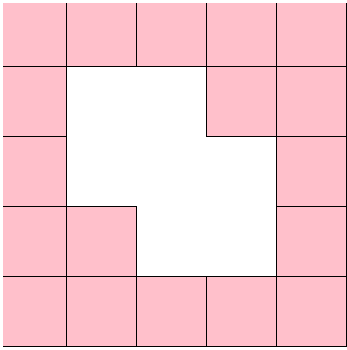
Q1 и Q2:

Бинарные: s2(cp) = -8.5 | q1 = -2.8333333333333335 | q2 = 2.8333333333333335 | s1(cp) = 8.5

Биполярные: s2(cp) = -34.0 | q1 = -11.333333333333334 | q2 = 11.333333333333334 | s1(cp) = 34.0

!Далее работа продолжается с бинарной функцией активации!

Символы, подготовленные для распознавания:

>> РАСПОЗНАВАНИЕ...

!Работа ведётся с сигмоидальной функцией!

Буква(ы) С в бинарном представлении:

[([1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1], 'C1'), ([1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1], 'C2'), ([1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1], 'C3')]

>> Сходство:

C1: A1 =>> 21 A2 =>> 23 B1 =>> 18 B2 =>> 22

C2: A1 =>> 20 A2 =>> 22 B1 =>> 19 B2 =>> 23

C3: A1 =>> 19 A2 =>> 21 B1 =>> 20 B2 =>> 24

>> Выбрана функция варианта - Сигмоидальная

>> Входной(ые) суммарные сигналы С:

C1, s = 1.5

C2, s = -0.5

C3, s = -2.5

>> Согласно функции активации, изображение(я) принадлежит(ат) классу(ам):

[((1, 'A'), 'C1'), ((0, 'B'), 'C2'), ((0, 'B'), 'C3')]

>> Сравним с функцией - Бинарная

>> Входной(ые) суммарные сигналы С:

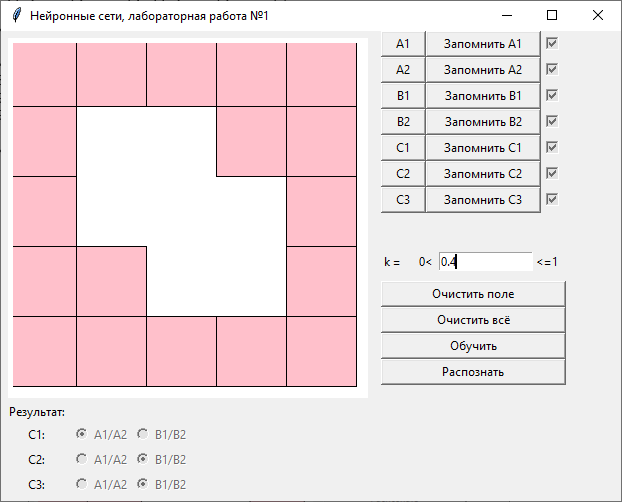
C1, s = 1.5

C2, s = -0.5

C3, s = -2.5

>> Согласно функции активации, изображение(я) принадлежит(ат) классу(ам):

[((1, 'A'), 'C1'), ((0, 'B'), 'C2'), ((0, 'B'), 'C3')]



**Вывод:** при распознавании символов С1-С3 с помощью сигмоидальной и бинарной функций, получились одинаковые результаты (С1 – класс А, С2 – класс В, С3 – класс В).

В расчёте, приведенном в начале выполнения были использованы буквы С1 и С2, результат распознавания сошелся с тем, какой был получен в ходе работы программы.

С1 и С2 различаются одним пикселом, аналогично С2 и С3.

При k=1.0

C1, s = 1.5

C2, s = -0.5

C3, s = -2.5

s2(cp) = -8.5 | q1 = -2.8 | q2 = 2.8 | s1(cp) = 8.5

При k=0.1:

C1, s = 0.15

C2, s = -0.04

C3, s = -0.24

s2(cp) = -0.84 | q1 = -0.28 | q2 = 0.28 | s1(cp) = 0.85

При уменьшении коэффициента k диапазоны стали меньше, однако значения букв по-прежнему находятся в этих диапазонах, таким образом можно сказать, что уменьшение k увеличивает точность.

Результаты распознавания на основе бинарной и сигмоидальной функций дают одинаковый результат, однако использование сигмоидальной бинарной функции позволяет сделать результат более точным, чем при использовании бинарной.