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 №5

по дисциплине «Нейронные сети»

тема: «Исследование метода обратного распространения ошибки»

Выполнила: ст. группы МИВТ-201

Браткова Ирина Олеговна

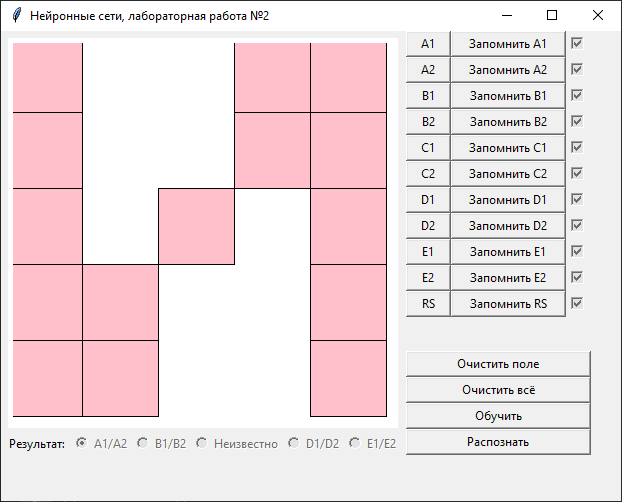
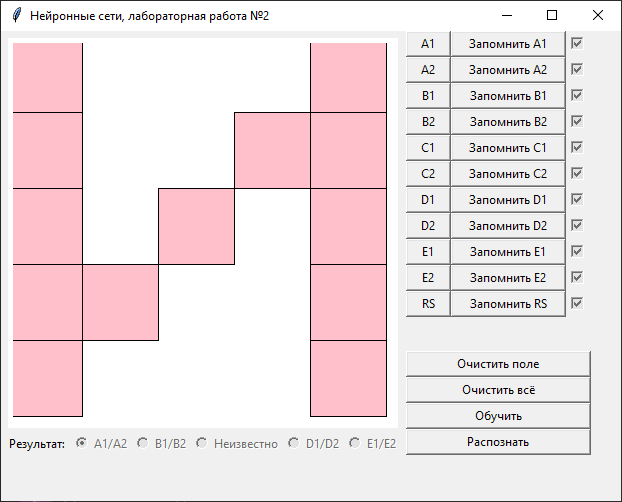
Белгород 2020 г

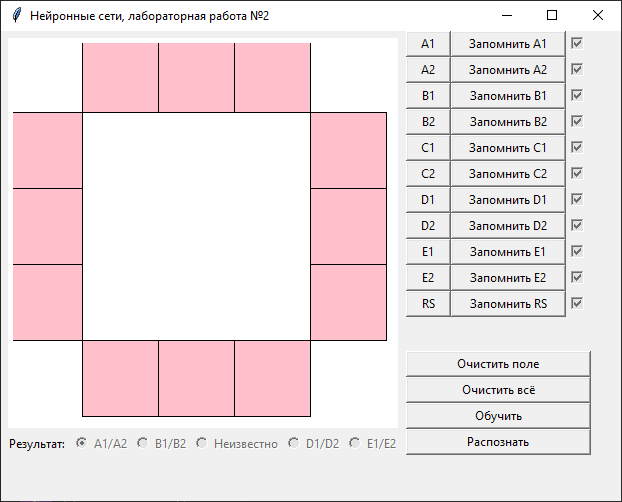
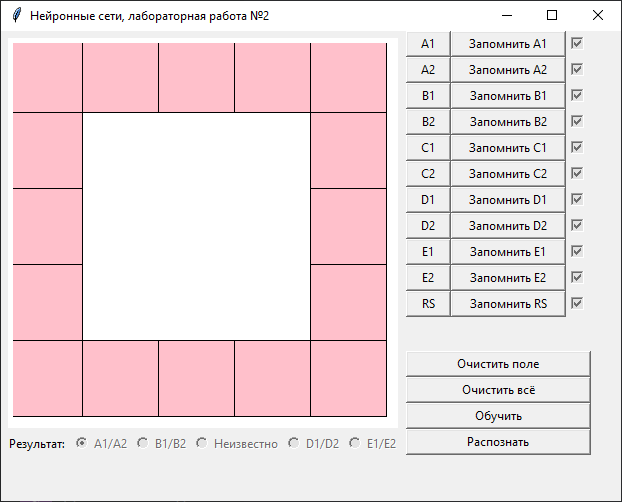
***Цель работы:*** *закрепление теоретических знаний, получение практических навыков использования метода обратного распространения (МОР) ошибки.*

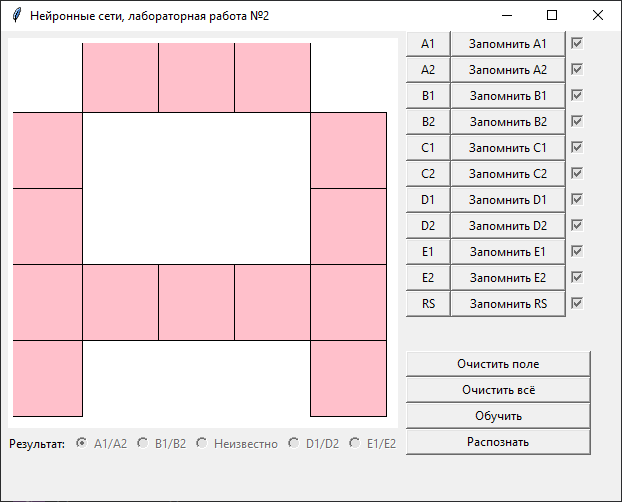
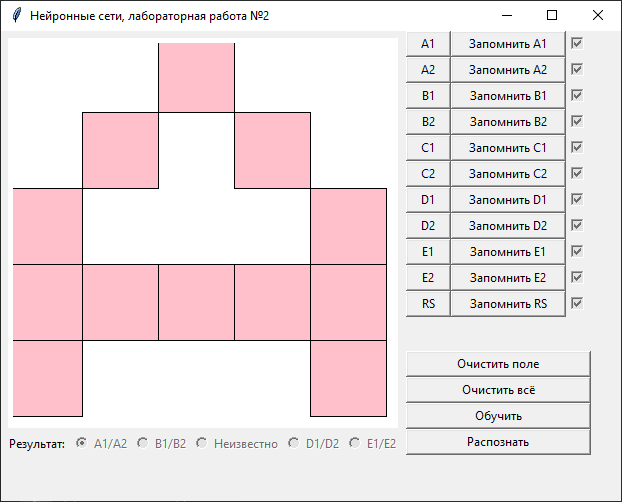
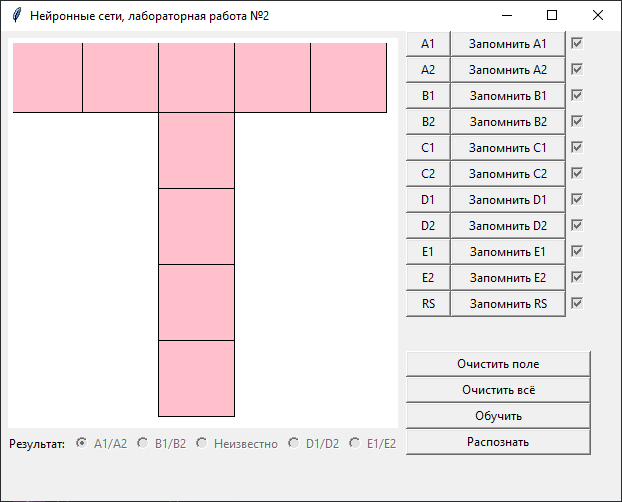
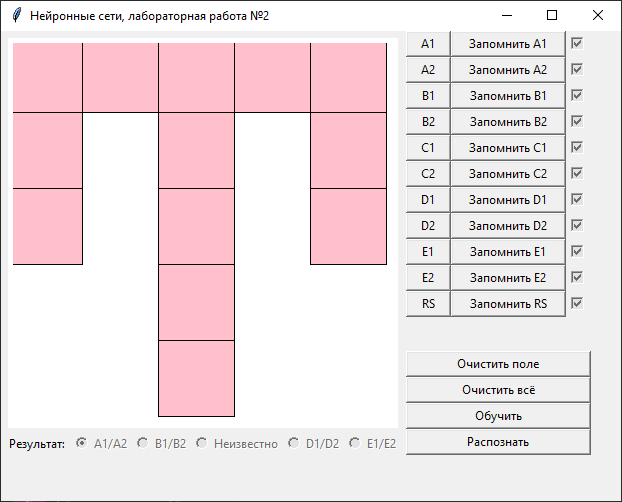
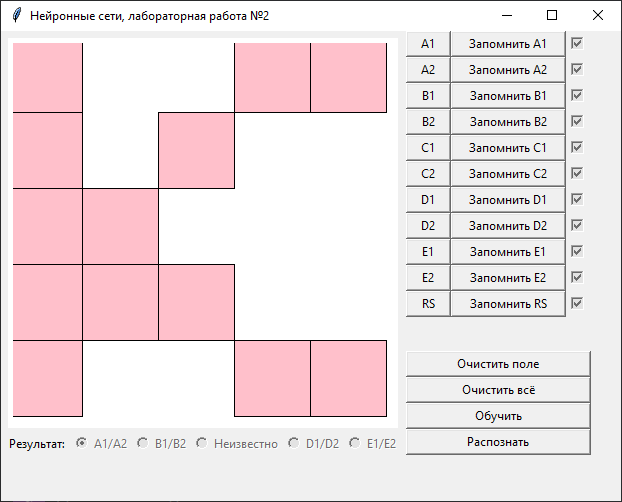
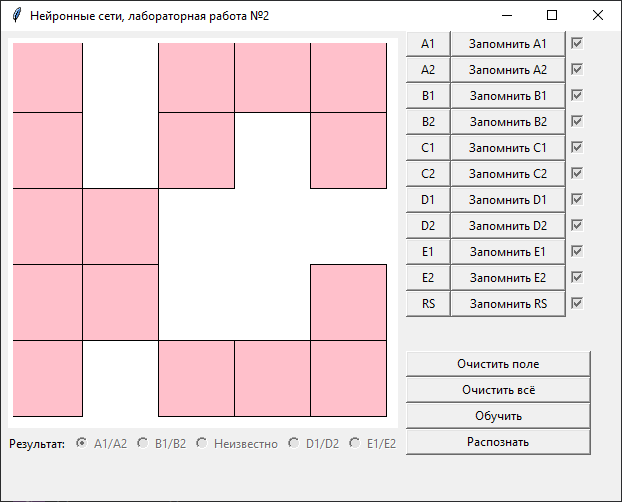
***Индивидуальные задания:***

* разработать по два черно-белых изображения первых букв Вашего имени и отчества (буквы должны быть разными);
* разработать структуру элементарного перцептрона для обучения разработанным классам изображений;
* реализовать и исследовать алгоритм МОР ошибки для элементарного перцептрона;
* разработать 5 черно-белых изображений первых букв Вашей фамилии (буквы должны быть разными);
* разработать структуру многослойного перцептрона с двумя слоями А-элементов для обучения разработанным классам изображений;
* реализовать и исследовать алгоритм МОР ошибки для разработанного многослойного перцептрона. При исследовании, так же, проанализировать зависимость количества итераций от требуемой погрешности.

***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;
* neuron.py – класс «Нейрон»;
* neuronweb.py – класс «Нейронная сеть».

***Calculating.py***

class CalculatingMor:  
 def \_\_init\_\_(self):  
 self.letter\_a1 = [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\_a2 = [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\_b1 = [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\_b2 = [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\_c1 = [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\_c2 = [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\_d1 = [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\_d2 = [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\_e1 = [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\_e2 = [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\_list = None  
 self.neuron\_web = None  
 self.rs = None  
  
 def recognize(self):  
 print(**'**\n**>> РАСПОЗНАВАНИЕ...'**)  
 print(**'**\n**>> Буква для распознавания'**)  
 self.\_\_make\_rs\_list\_binary()  
 print(**'**\n**>> Подсчет схожести'**)  
 self.\_\_calculate\_gemini()  
 self.neuron\_web.recognize(self.rs)  
 return [1]  
  
 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 \_\_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 \_\_make\_rs\_list\_binary(self):  
 if self.letter\_rs is not None:  
 rs = copy\_and\_insert\_one(self.letter\_rs)  
 print(rs) self.rs = rs  
  
 def teaching(self, count\_hide=2):  
 print(**'**\n**>> ОБУЧЕНИЕ...'**)  
 self.\_\_make\_letters\_list()  
 print(**'**\n**>> Создание обучающих выборок'**)  
 self.m\_list = self.\_\_make\_all\_m()  
 print(**'**\n**>> Создание НС'**)  
 self.neuron\_web = NeuronWebMor(len(self.letter\_a1), len(self.letters\_list), count\_hide)  
 print(self.neuron\_web)  
 print(**'**\n**>> НАЧАЛО ОБУЧЕНИЯ С ПОМОЩЬЮ МОР...'**)  
 self.neuron\_web.teaching(self.m\_list, 0.1)  
  
 def \_\_make\_letters\_list(self):  
 self.letters\_list = [  
 (self.letter\_a1, self.letter\_a2),  
 (self.letter\_b1, self.letter\_b2)]  
 if self.letter\_c1 is not None and self.letter\_c2 is not None:  
 self.letters\_list.append((self.letter\_c1, self.letter\_c2))  
 if self.letter\_d1 is not None and self.letter\_d2 is not None:  
 self.letters\_list.append((self.letter\_d1, self.letter\_d2))  
 if self.letter\_e1 is not None and self.letter\_e2 is not None:  
 self.letters\_list.append((self.letter\_e1, self.letter\_e2))  
  
 def \_\_make\_all\_m(self):  
 def \_\_upd\_binary\_list(letters\_list):  
 result = []  
 for el in letters\_list:  
 tmp1 = copy\_and\_insert\_one(el[0])  
 tmp2 = copy\_and\_insert\_one(el[1])  
 result.append((tmp1, tmp2))  
 return result  
 letters = \_\_upd\_binary\_list(self.letters\_list)  
 y = [0 for \_ in range(len(letters))]  
 res = []  
 for i in range(len(letters)):  
 new\_y = copy.copy(y)  
 new\_y[i] = 1  
 res.append((letters[i][0], new\_y))  
 res.append((letters[i][1], new\_y))  
 return res

***Neuron.py***

class NeuronMor:  
 def \_\_init\_\_(self, count\_input, count\_output, w\_range):  
 self.count\_input = count\_input  
 self.count\_output = count\_output  
 self.w\_list = [random.uniform(w\_range[0], w\_range[1]) for \_ in range(count\_output)]  
 self.potential = None  
 self.exi = None self.error = None  
  
 def \_\_str\_\_(self):  
 s = **'Нейрон (**\n\t\t**входов = '** + str(self.count\_input) + **', '** s += **'выходов = '** + str(self.count\_output) + **', '** s += **'**\n\t\t**веса на выходе = '** + str(self.w\_list) + **', '** s += **'**\n\t\t**выходное значение = '** + str(self.exi) +**', '** s += **'**\n\t\t**потенциалы = '** + str(self.potential) + **'**\n**)'** return s  
  
 def calc\_u\_vh(self, x\_lst, ws\_lst):  
 pot = 0.0  
 for i in range(len(ws\_lst)):  
 pot += x\_lst[i] \* ws\_lst[i]  
 return pot  
  
 def calc\_u\_output(self, x, ws\_lst=None):  
 if ws\_lst is None:  
 self.potential = x  
 self.exi = x  
 return self.potential, self.exi  
 else:  
 potential = self.calc\_u\_vh(x, ws\_lst)  
 self.potential = potential  
 exi = self.activation\_function(potential)  
 self.exi = exi  
 return potential, exi  
  
 def calc\_error\_signal(self, sign):  
 if type(sign) is int or type(sign) is float:  
 self.error = sign - self.exi  
 return self.error  
 else:  
 error\_sing = 0.0  
 for i in range(len(sign)):  
 error = sign[i].error  
 error \*= sign[i].exi  
 error \*= (1.0 - sign[i].exi)  
 error\_sing += self.w\_list[i] \* error  
 self.error = error\_sing  
 return error\_sing  
  
 def activation\_function(self, p):  
 return 1.0 / (1.0 + math.exp(-p))  
  
 def product\_activation\_function(self, p):  
 return self.activation\_function(p) \* (1 - self.activation\_function(p))  
 def ws\_correction(self, nu, neurons):  
 for i in range(len(neurons)):  
 tmp = nu \* neurons[i].error \* self.product\_activation\_function(neurons[i].potential) \* self.exi  
 self.w\_list[i] += tmp

***NeuronWeb.py***

class NeuronWebMor:  
 def \_\_init\_\_(self, count\_input\_neurons, count\_output, count\_hide\_layers, w\_range=(-0.5, 0.5)):   
 self.count\_input\_neurons = count\_input\_neurons  
 self.count\_output\_neurons = count\_output  
 self.count\_hide\_layers = count\_hide\_layers  
 self.count\_neurons\_in\_hide, self.hide\_neurons = self.\_\_make\_neurons\_in\_hide\_\_(count\_input\_neurons, count\_hide\_layers, count\_output, w\_range)  
 self.input\_neurons = [  
 NeuronMor(1, self.count\_neurons\_in\_hide[0], w\_range) for \_ in range(count\_input\_neurons)  
 ]  
 self.output\_neurons = [  
 NeuronMor(len(self.hide\_neurons[-1]), 1, w\_range) for \_ in range(count\_output)  
 ]  
 self.list\_patterns = None  
  
 def \_\_str\_\_(self):  
 def \_\_lst\_neurons\_to\_string(lst, tab=**'**\t**'**):  
 s = **''** for l in range(len(lst)):  
 k = l + 1  
 if l == len(lst) - 1:  
 s += tab + str(k) + **'-'** + str(lst[l])  
 else:  
 s += tab + str(k) + **'-'** + str(lst[l]) + **'**\n**'** return s  
  
 def \_\_matrix\_neurons\_to\_string(matrix):  
 s = **''** for m in range(len(matrix)):  
 k = m + 1  
 if m == len(matrix) - 1:  
 s += **'**\n\t**Слой №'** + str(k) + **'**\n**'** + \_\_lst\_neurons\_to\_string(matrix[m], **'**\t\t**'**)  
 else:  
 s += **'**\n\t**Слой №'** + str(k) + **'**\n**'** + \_\_lst\_neurons\_to\_string(matrix[m], **'**\t\t**'**) + **'**\n**'** return s  
 s = **'Сеть.**\n**Входной слой = {**\n**'** t = \_\_lst\_neurons\_to\_string(self.input\_neurons)  
 s += str(t) + **'**\n**}**\n**'** s += **'**\n**Скрытые слои = {**\n**'** t = \_\_matrix\_neurons\_to\_string(self.hide\_neurons)  
 s += str(t) + **'**\n**}**\n**'** s += **'**\n**Выходной слой = {**\n**'** t = \_\_lst\_neurons\_to\_string(self.output\_neurons)  
 s += str(t) + **'**\n**}**\n**'** return s  
  
 def \_\_make\_neurons\_in\_hide\_\_(self, count\_input, count\_hide\_layers, count\_output, w\_range):  
 def \_\_calc\_count():  
 if count\_input > 8:  
 res = []  
 first = int(count\_input / 2)  
 lastt = int(count\_input / 4)  
 for i in range(count\_hide\_layers): *# по количеству слоёв* if i == 0:  
 res.append(first)  
 elif i == count\_hide\_layers - 1: res.append(lastt)  
 else:  
 first2 = res[i - 1] - 2  
 t = first2 if first2 > lastt else lastt  
 res.append(t)  
 return res  
 else:  
 return [2, 2]  
  
 def \_\_make\_neurons():  
 res = [] length = len(count\_hide\_neurons)  
 for i in range(length):  
 if i == 0:  
 t = [NeuronMor(count\_input, count\_hide\_neurons[i + 1], w\_range) for \_ in  
 range(count\_hide\_neurons[i])]  
 res.append(t)  
 elif i == length - 1:  
 t = [NeuronMor(count\_hide\_neurons[i - 1], count\_output, w\_range) for \_ in  
 range(count\_hide\_neurons[i])]  
 res.append(t)  
 else:  
 t = [NeuronMor(count\_hide\_neurons[i - 1], count\_hide\_neurons[i + 1], w\_range) for \_ in  
 range(count\_hide\_neurons[i])]  
 res.append(t)  
 return res  
  
 count\_hide\_neurons = \_\_calc\_count()  
 return count\_hide\_neurons, \_\_make\_neurons()  
  
 def teaching(self, patterns, ny):  
 self.list\_patterns = patterns  
 indexes = [i for i in range(len(patterns))]  
 count\_epoch = 0  
 flag = False  
 while not flag:  
 exis = []  
 for i in range(len(patterns)):  
 if i in indexes:  
 self.direct\_way(patterns[i][0])  
 self.reverse\_way(patterns[i][1])  
 self.error\_correct(ny)  
 exi = patterns[i][1]  
 exis.append(exi)  
  
 flag, indexes = self.\_\_ost(patterns, exis, indexes)  
 count\_epoch += 1  
 print(**'**\n**Потребовалось эпох: '**+str(count\_epoch))  
 for o in self.output\_neurons:  
 print(o)  
  
 def recognize(self, letter):  
 self.direct\_way(letter)  
 exi = [e.exi for e in self.output\_neurons]  
 print(**'Выходы: '**)  
 print(exi)  
  
 def \_\_ost(self, patterns, exis1, last\_idexes, eps=0.1):  
 exis2 = []  
 for i in range(len(patterns)):  
 if i in last\_idexes:  
 self.direct\_way(patterns[i][0])  
 exi2 = copy.copy([e.exi for e in self.output\_neurons])  
 exis2.append(exi2)  
 indexes = []  
 for i in range(len(exis2)):  
 e1 = exis1[i]  
 e2 = exis2[i]  
 for j in range(len(e2)):  
 if math.fabs(e1[j] - e2[j]) < eps:  
 continue  
 else:   
 indexes.append(i)  
 return (False, indexes) if len(indexes) != 0 else (True, [])  
  
 def direct\_way(self, pattern):  
 for i in range(self.count\_input\_neurons):  
 self.input\_neurons[i].calc\_u\_output(pattern[i])  
  
 potentials = [neuron.exi for neuron in self.input\_neurons]  
 for i in range(len(self.hide\_neurons[0])):  
 ws\_lst = [n.w\_list[i] for n in self.input\_neurons]  
 self.hide\_neurons[0][i].calc\_u\_output(potentials, ws\_lst)  
  
 potentials = [neuron.exi for neuron in self.hide\_neurons[0]]  
 for i in range(1, self.count\_hide\_layers):  
 for j in range(len(self.hide\_neurons[i])):  
 ws\_lst = [n.w\_list[j] for n in self.hide\_neurons[i - 1]]  
 self.hide\_neurons[i][j].calc\_u\_output(potentials, ws\_lst)  
  
 potentials = [neuron.exi for neuron in self.hide\_neurons[-1]]  
 for i in range(self.count\_output\_neurons):  
 ws\_lst = [n.w\_list[i] for n in self.hide\_neurons[-1]]  
 self.output\_neurons[i].calc\_u\_output(potentials, ws\_lst)  
  
 def reverse\_way(self, pattern):  
 for i in range(self.count\_output\_neurons):  
 self.output\_neurons[i].calc\_error\_signal(pattern[i])  
  
 for i in range(len(self.hide\_neurons[-1])):  
 self.hide\_neurons[-1][i].calc\_error\_signal(self.output\_neurons)  
  
 for i in range(0, self.count\_hide\_layers - 1):  
 for j in range(len(self.hide\_neurons[i])):  
 self.hide\_neurons[i][j].calc\_error\_signal(self.hide\_neurons[i + 1])  
  
 def error\_correct(self, ny=0.5):  
 for i in range(self.count\_input\_neurons):  
 self.input\_neurons[i].ws\_correction(ny, self.hide\_neurons[0])  
  
 for i in range(0, self.count\_hide\_layers - 1):  
 for j in range(len(self.hide\_neurons[i])):  
 self.hide\_neurons[i][j].ws\_correction(ny, self.hide\_neurons[i+1])  
  
 for i in range(len(self.hide\_neurons[-1])):  
 self.hide\_neurons[-1][i].ws\_correction(ny, self.output\_neurons)

Класс «Нейронная сеть» получает на вход количество входных нейронов, выходных, количество скрытых слоёв и диапазон заполнения весовыми коэффициентами.

Количество скрытых слоёв по умолчанию – 2, а количество нейронов в них формируется по правилу «в первом скрытом слое количество нейронов равно ½ нейронов входного слоя, а во втором скрытом слое – ½ нейронов первого скрытого слоя». Таким образом, для пяти образов нейронная сеть имеет вид: 25-12-6-5.

Правило останова: если разница между полученным выходом нейрона выходного слоя и эталонным значением больше эпсилон, данное изображение запоминается для того, чтобы заново обучить сеть на нём. Итерации выполняются до тех пор, пока ранее упомянутая разница для каждого изображения не станет меньше эпсилон.

**Обучение на примере из лекции**

Нейронная сеть после одной эпохи (итерации) обучения.

Входной слой = {

1-Нейрон ( входов = 1, выходов = 2,

веса на выходе = [0.1999703194887195, 0.30000737062221494],

выходное значение = 0.5, потенциалы = 0.5)

2-Нейрон ( входов = 1, выходов = 2,

веса на выходе = [-0.30001780830676833, 0.20000442237332897],

выходное значение = 0.3, потенциалы = 0.3)

}

Скрытые слои = {

Слой №1

1-Нейрон ( входов = 2, выходов = 2,

веса на выходе = [-0.09940391928810158, 0.19970144718936358],

выходное значение = 0.502499979166875, потенциалы = 0.010000000000000009)

2-Нейрон ( входов = 2, выходов = 2,

веса на выходе = [-0.0993448356107954, -0.3003281454439332],

выходное значение = 0.5523079095743253, потенциалы = 0.21)

Слой №2

1-Нейрон ( входов = 2, выходов = 1,

веса на выходе = [0.40563427303556676],

выходное значение = 0.473656088837, потенциалы = -0.1054807887412005) 2-Нейрон ( входов = 2, выходов = 1,

веса на выходе = [-0.19424613786526898],

выходное значение = 0.483707675592, потенциалы = -0.06519237703892256)

}

**Выходной слой = {**

**1-Нейрон (**

**входов = 2, выходов = 1, веса на выходе = [0.0],**

**выходное значение = 0.5231634464270264,**

**потенциалы = 0.09272015512509502)**

**}**

Значения на выходе совпадают со значениями из лекции.

**Обучение на пяти классах изображений (по два изображения в классе)**

|  |  |  |  |
| --- | --- | --- | --- |
| ε | η | Диапазон весов | Потребовалось итераций (эпох) |
| 0.1 | 0.1 | -0,5 ; 0.5 | 1264 |
| 955 |
| 998 |
| -0.3 ; 0.3 | 1248 |
| 1046 |
| 1445 |
| 0.5 | -0,5 ; 0.5 | 184 |
| 218 |
| 194 |
| -0.3 ; 0.3 | 258 |
| 273 |
| 235 |
| 1 | -0.5 ; 0.5 | 107 |
| 128 |
| 99 |
| -0.3 ; 0.3 | 128 |
| 112 |
| 155 |
| 0.01 | 0.1 | -0.5 ; 0.5 | 39724 |
| 36400 |
| 30872 |
| -0.3 ; 0.3 | 55801 |
| 37298 |
| 30276 |
| 0.5 | -0.5 ; 0.5 | 4713 |
| 6897 |
| 6111 |
| -0.3 ; 0.3 | 7626 |
| 11427 |
| 7607 |
| 1 | -0.5 ; 0.5 | 3266 |
| 3652 |
| 3932 |
| -0.3 ; 0.3 | 3537 |
| 3982 |
| 3575 |

Красным цветом выделен худший результат по количеству эпох. Это связано с тем, что при малом значении эпсилон и небольшом значении коэффициента обучения требуется большее количество итераций для достижения заданной точности. Увеличение коэффициента обучения в этом случае помогает решить проблему, однако, снижает точность.

Оранжевым цветом выделены самые быстрые «ученики» среди сетей. Для достижения заданной точности при таком коэффициенте обучение коррекция, действительно, происходит довольно быстро, однако уменьшается ее точность.

Параметры обучения, представленные черным текстом, имеют место быть, однако скорость работы значительно ниже.

Для проведения опытов по распознаванию был выбран набор параметров ε = 0.1, η = 0.1, веса = (-0,5 – 0,5).

**Распознавание**

|  |  |
| --- | --- |
| **Изображение** | **Лог программы** |
| **Распознавание эталонных изображений** | |
|  | >> Буква для распознавания  [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]  >> Подсчет схожести  Буква №1: 25, 23  Буква №2: 16, 12  Буква №3: 16, 16  Буква №4: 9, 13  Буква №5: 15, 15  Предположительно rs похожа на букву №1  Выходы:  **[0.9010801992313408,**  0.001196678769309776,  0.0703993880994949,  0.018070379688281065,  0.08683305952212537]  Комментарий: наибольшее значение на первом нейроне, это правильно – данная буква относится к первому классу. |
|  | >> Буква для распознавания  [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]  >> Подсчет схожести  Буква №1: 12, 14  Буква №2: 21, 25  Буква №3: 11, 17  Буква №4: 12, 16  Буква №5: 10, 12  Предположительно rs похожа на букву №2  Выходы:  [0.0009217424895051011,  **0.9243224485847613,**  0.07781439814006193,  0.08219902940595412,  0.00046292107780792745]  Комментарий: наибольшее значение на втором нейроне, это правильно. Буква О – второй класс изображений. |
|  | >> Буква для распознавания  [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]  >> Подсчет схожести  Буква №1: 16, 14  Буква №2: 11, 11  Буква №3: 25, 19  Буква №4: 8, 8  Буква №5: 12, 10  Предположительно rs похожа на букву №3  Выходы:  [0.15555233691118156,  0.10030092598872334,  **0.8833445618953741**,  0.0006606274837377894,  0.0017617501395287888]  Комментарий: верно. |
|  | >> Буква для распознавания  [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]  >> Подсчет схожести  Буква №1: 9, 9  Буква №2: 12, 12  Буква №3: 8, 10  Буква №4: 25, 21  Буква №5: 13, 13  Предположительно rs похожа на букву №4  Выходы:  [0.010839263772651313,  0.11264407433017995,  0.00012509948058555425,  **0.8893551180716964,**  0.06552589549454713] |
|  | >> Подсчет схожести  Буква №1: 16, 16  Буква №2: 19, 15  Буква №3: 11, 15  Буква №4: 12, 14  Буква №5: 20, 25  Предположительно rs похожа на букву №5  Выходы:  [0.06059036042829072, 0.05433861434740262, 0.027194446872125796, 0.04456056358125213, **0.8600042668460914]** |
| **Распознавание иных символов** | |
|  | >> Буква для распознавания  [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1]  >> Подсчет схожести  Буква №1: 15, 17  Буква №2: 18, 18  Буква №3: 18, 24  Буква №4: 9, 13  Буква №5: 13, 14  Предположительно rs похожа на букву №3  Выходы:  [0.02734848292859375,  0.00941728104692872,  **0.8018864008118997,**  0.01704340497008083,  0.058421727383734676] |
|  | >> Подсчет схожести  Буква №1: 17, 19  Буква №2: 20, 16  **Буква №3: 16, 22**  Буква №4: 11, 15  Буква №5: 15, 16  Предположительно rs похожа на букву №3  Выходы:  [0.08553688278183967,  0.033408191595713815,  **0.33995925647300107,**  0.06216399571732109,  0.018760594642345616] |
|  | >> Подсчет схожести  Буква №1: 16, 18  Буква №2: 19, 17  Буква №3: 17, 23  Буква №4: 10, 14  Буква №5: 14, 15  Предположительно rs похожа на букву №3  Выходы:  [0.0411291163962036,  0.01292075471672113,  **0.7045394727592481**,  0.029732477281603754,  0.03742165078194464] |
|  | >> Подсчет схожести  Буква №1: 15, 17  Буква №2: 20, 18  Буква №3: 16, 22  Буква №4: 11, 15  Буква №5: 13, 16  Предположительно rs похожа на букву №3  Выходы:  [0.01906672901594561,  0.057326522978376156,  **0.4776455195675092,**  0.08113043737633294,  0.019343307314951246] |
|  | >> Подсчет схожести  Буква №1: 14, 16  **Буква №2: 21, 19**  Буква №3: 15, 21  Буква №4: 10, 14  Буква №5: 14, 17  Предположительно rs похожа на букву №2  Выходы:  [0.010477127303534969,  **0.41631930840356884,**  0.12378551519172488,  0.03295340439223071,  0.08026661172592099] |
|  | >> Подсчет схожести  Буква №1: 13, 15  Буква №2: 20, 18  Буква №3: 14, 20  Буква №4: 11, 15  Буква №5: 15, 18  Предположительно rs похожа на букву №2  Выходы:  [0.009206831763059671,  **0.4286596327959871**,  0.11801454842799437,  0.03191052372671633,  0.08376727425272058] |
|  | >> Подсчет схожести  Буква №1: 14, 16  Буква №2: 19, 17  Буква №3: 13, 19  Буква №4: 12, 16  Буква №5: 14, 17  Предположительно rs похожа на букву №2  Выходы:  [0.015897913805512823,  **0.3516807468994172**,  0.08013494589718867,  0.1663195387996734,  0.012440418228559864] |
|  | >> Подсчет схожести  Буква №1: 13, 15  Буква №2: 18, 18  Буква №3: 12, 18  Буква №4: 13, 17  Буква №5: 13, 16  Предположительно rs похожа на букву №2  Выходы:  [0.018371040277045413,  **0.38936363613288877,**  0.06544811052028607,  0.25484090093965023,  0.00873929076579116] |
|  | >> Подсчет схожести  Буква №1: 12, 14  Буква №2: 17, 19  Буква №3: 11, 17  Буква №4: 14, 18  Буква №5: 12, 15  Предположительно rs похожа на букву №2  Выходы:  [0.031818028479987465,  **0.3668292992424634,**  0.04947658862667357,  0.4630445518613203,  0.0038600888213051686] |
|  | >> Подсчет схожести  Буква №1: 11, 13  Буква №2: 16, 18  Буква №3: 10, 16  Буква №4: 15, 19  Буква №5: 11, 14  Предположительно rs похожа на букву №4  Выходы:  [0.0671942746380566,  0.1996632947142723,  0.055637199073090365,  **0.7846103478196103,**  0.0008762134215653416] |
|  | >> Подсчет схожести  Буква №1: 10, 12  Буква №2: 17, 19  Буква №3: 9, 15  Буква №4: 16, 20  Буква №5: 10, 13  Предположительно rs похожа на букву №4  Выходы:  [0.05900844823687734,  0.2623459424730759,  0.041971929811703494,  **0.8297173946589446,**  0.000779283838469895] |

На основе проведенных исследований можно сделать вывод, что нейронная сеть с обратным распространением ошибки является мощным средством для распознавания образов (в данном случае нейронная сеть отработала правильно во всех случаях).