|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** | | | | | |
| БГТУ.СМК-Ф-4.2-К5-01 | | | | | |
| Факультет | |  | И |  | Информационные и управляющие системы |  |
| Кафедра | |  | И5 |  | Информационные системы и программная инженерия |  |
| Дисциплина | |  | Основы искусственного интеллекта | | |  |

ЛАБОРАТОРНАЯ РАБОТА № 1

|  |
| --- |
| ОБУЧЕНИЕ ИСКУСТВЕННОЙ НЕЙРОННОЙ СЕТИ |
| МЕТОДОМ ОБРАТНОГО РАСПРОСТАНЕНИЯ ОШИБКИ |
|  |

Вариант 13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | И973 |
| Прокофьев Е.Д. | | | | | |
| Фамилия И.О. | | | | | |
| **ПРОВЕРИЛ** | | | | | |
| Куликов Д.Б. | |  |  | | |
| Фамилия И.О. Подпись | | | | | |
| Оценка |  | | | |  |
| «\_\_\_\_\_» |  | | | | 2021 г. |

САНКТ-ПЕТЕРБУРГ

2021 г.

**1 Исходные данные**

Обучающая выборка состоит из 4 образцов. Каждый образец – это символ, состоящий из N точек (матрица точек размеров NxM) и состоящий из 2 цветов: черный или белый. Для обработки изображения используется кодирование, где «белая» точка – 0, «черная» точка – 1.

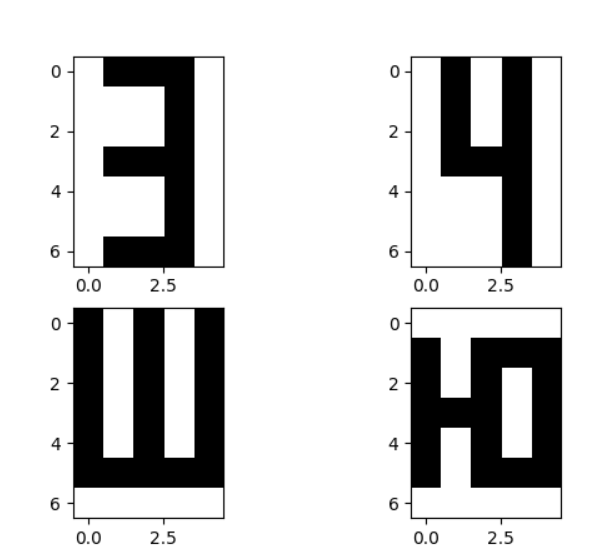
Образцы относятся к одному из четырех классов.

В таблице 1 представлена обучающая выборка из массива образцов.

Таблица 1 – Обучающая выборка

|  |  |  |
| --- | --- | --- |
| Символ | Код | Класс |
| 3 | 00111000010000100111000010000100111 | 1000 |
| 4 | 01010010100101001110000100001000010 | 0100 |
| ю | 10111101011010111101101011010110111 | 0010 |
| ш | 10101101011010110101101011010111111 | 0001 |

На рисунке 1 представлена графическая интерпретация таблицы 1.

  
Рисунок 1 – Обучающая выборка

Для проверки работоспособности обученной сети посредством проверки правильности классификации на примерах «искаженных» образцов.

На рисунке 2 представлена тестовая выборка.

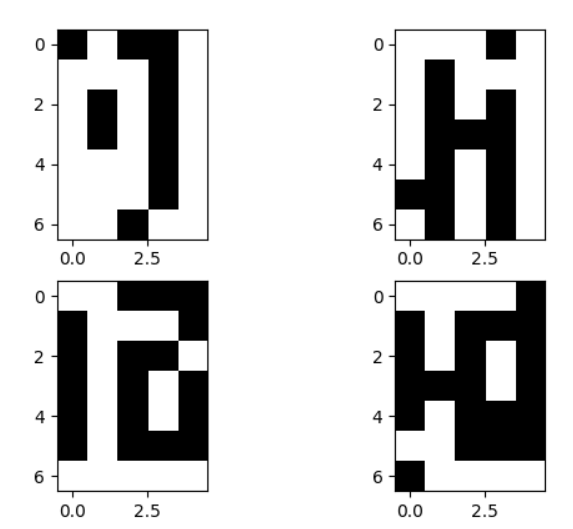
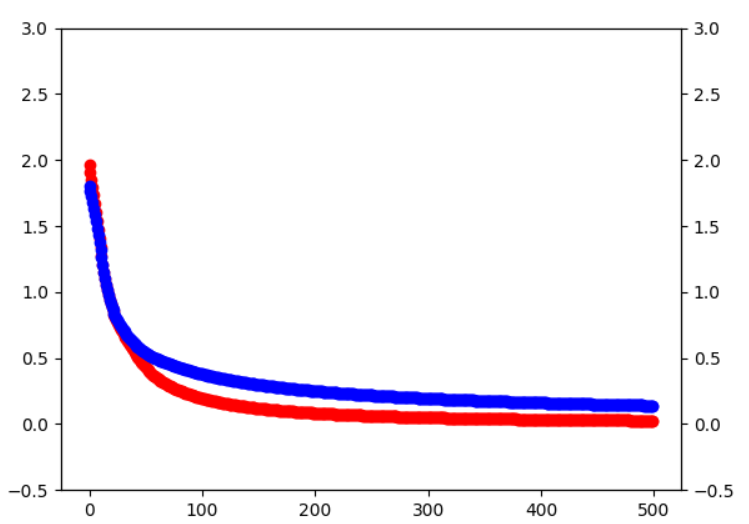


Рисунок 2 - Тестовая выборка

**2 График изменения ошибки**

На рисунке 3 представлен график изменения ошибки обучения(красный) и тестирования(синий):

  
Рисунок 3 – График изменения ошибки от эпох

**3 Фрагменты программного кода**

3.1 Инициализация сети

def init(numInputs=1, numOutputs=1):  
 global weights\_0\_1, weights\_1\_2  
  
 k = math.ceil(math.sqrt(numInputs \* numOutputs)) # количество скрытых нейронов(применяется эврестический метод.  
 # ceil-вовзращает нименьшее целое число, но не меньше чем sqrt(numInputs \* numOutputs)  
  
#для весов инициализируем мылыми случайными величинами  
 weights\_0\_1 = np.random.normal(0.0, 2 \*\* -0.5, (k, numInputs))#матрица кхnuminputs  
 weights\_1\_2 = np.random.normal(0.0, 1, (numOutputs, k))

3.2 Вычисление ошибки обучения

#среднее квадратическое отклонение mean-вычисляет среднее арифметическое значений элементов массива  
def MSE(a, b):  
 return np.mean((a - b) \*\* 2)

3.3 Реализация алгоритма прямого распространения

#функция прямого распространения  
#на каждый нейрон подаем сумму сигналов умноженную на весовую характеристику. Получившиеся значение передаем в сигмоидную функцию,  
# результат функции будет сигнал для выхода  
def predict(inputs):  
 global weights\_0\_1, weights\_1\_2  
 # От первого слоя к скрытому  
 inp = np.dot(weights\_0\_1, inputs)#Функция dot() вычисляет скалярное произведение двух массивов  
 out = sigmoid\_mapper(inp)  
 # От скрытого слоя к третьему  
 inp2 = np.dot(weights\_1\_2, out)  
 out2 = sigmoid\_mapper(inp2)  
 return out2

3.4 Реализация алгоритма обратного распространения ошибки

#функция обратного распространения ошибки  
#проходим по каждому нейрону слоя и считаем ошибку от ожидаемого значения. Высчитываем дельту, значение на сколько  
# будем сдвигать веса и применяем к текущим значениям весов для слоя  
def train(inputs, expected\_predict, learning\_rate):  
 global weights\_0\_1, weights\_1\_2  
  
 # От первого слоя к скрытому. укаждого нейрона есть свой вес. в инпутс1 засовывает матричное произведение входного массива на массив весов. прогоняет через функцию активации. и получает выход от каждого нейрона  
 inputs\_1 = np.dot(weights\_0\_1, inputs)  
 outputs\_1 = sigmoid\_mapper(inputs\_1)  
  
 # От скрытого слоя к третьему  
 inputs\_2 = np.dot(weights\_1\_2, outputs\_1)  
 outputs\_2 = sigmoid\_mapper(inputs\_2)  
  
 # Проходим по всем нейронам третьего слоя  
 for iter in range(len(outputs\_2)):  
 actual\_predict = outputs\_2[iter] # Текущее значение нейрона  
  
 # От третьего слоя к скрытому  
 error\_layer\_2 = np.array([actual\_predict - expected\_predict[iter]]) # Считаем ошибку. текущее значение нейрона минус ожидаемое.предсказанное значение класса - ожидаемое значение класса  
 gradient\_layer\_2 = actual\_predict \* (1 - actual\_predict) # Производная сигмоидной функции. Градиент - направление скорейшего возрастания. Тк мы считаем здесь ошибку, а ее нужно минимализировать, соответствеенно дальше эту ошибку будем дальше отнимать  
 weights\_delta\_layer\_2 = error\_layer\_2 \* gradient\_layer\_2 # Дельта весов. error\*sigmoid(x)dx. чтобы получить значение которое будем отнимать  
 weights\_1\_2[iter] -= (np.dot(weights\_delta\_layer\_2,outputs\_1.reshape(1, len(outputs\_1)))) \* np.array([learning\_rate]) #Сдвигаем веса.weights=weights-outputs\*delta\*скорость обучения  
  
 # От скрытого слоя к первому  
 error\_layer\_1 = weights\_delta\_layer\_2 \* [weights\_1\_2[iter]] # Считаем ошибку. передаем дальше ошибку  
 gradient\_layer\_1 = [outputs\_1 \* (1 - outputs\_1)] # Производная сигмоидной функции  
 weights\_delta\_layer\_1 = error\_layer\_1 \* gradient\_layer\_1 # Дельта весов  
 weights\_0\_1 -= np.dot(inputs.reshape(len(inputs), 1), weights\_delta\_layer\_1).T \* np.array([learning\_rate]) #Сдвигаем веса

3.5 Сигмоидная функция

#сигмоидная функция. вычиселние выходного сигнала нейрона.применяется для сглаживания значений некоторой величины  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))#сигмоида(логическая функция)  
sigmoid\_mapper = np.vectorize(sigmoid) # Для использования векторов  
#массивы будут хранить значения весов  
weights\_0\_1 = []#веса между первым и скрытым слоем  
weights\_1\_2 = []#веса между скрытым и третьем слоем

3.6 Зашумление сигнала

def shum(data): # зашумление  
 count = height \* width - 1  
 data\_out = data.copy()  
 for i in range(count):  
 n = random.random()  
 if n<0.3:  
 if data\_out[i]==1:  
 data\_out[i]=0  
 else:  
 data\_out[i]=1  
 else:  
 data\_out[i]=data\_out[i]  
 return data\_out

**4 Тестирование**

После обучения, сеть способна определять измененные обучающие образы.

Подадим тестовую выборку на сеть. Полученный результат показан на рисунках 4, 5, 6, 7, 8.



Рисунок 4 – Результат работы для цифры 3

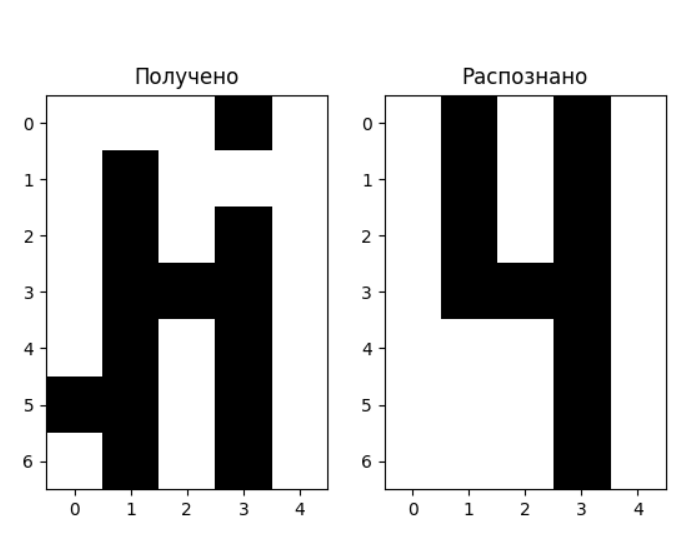


Рисунок 5 – Результат работы для цифры 4

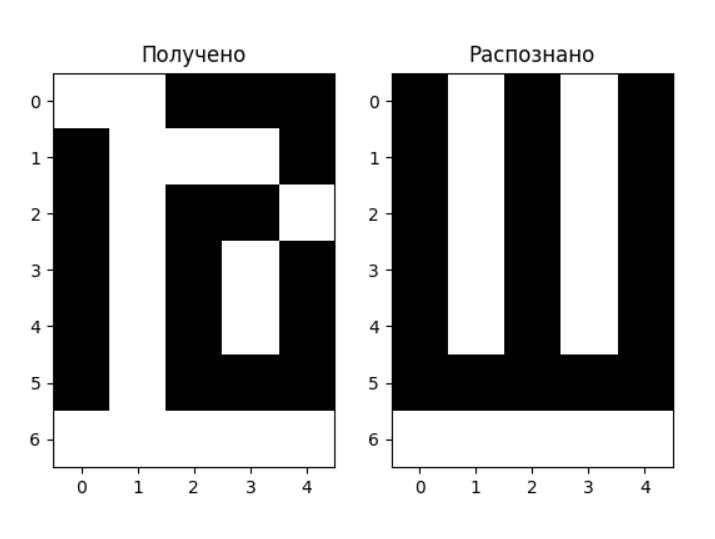


Рисунок 6 – Результат работы для буквы Ш

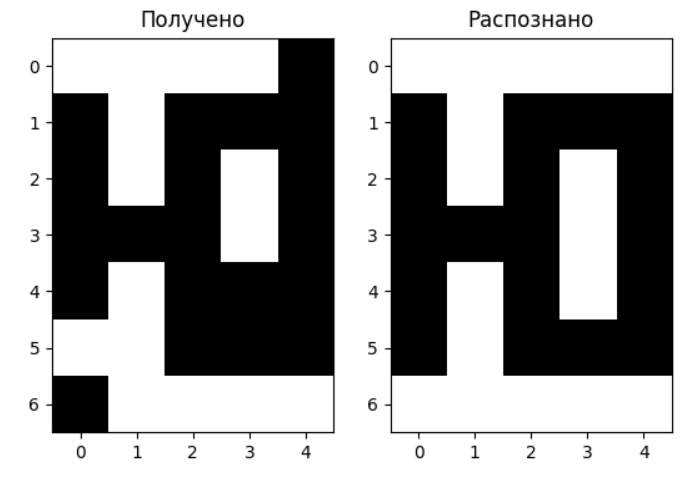


Рисунок 7 – Результат работы для буквы Ю

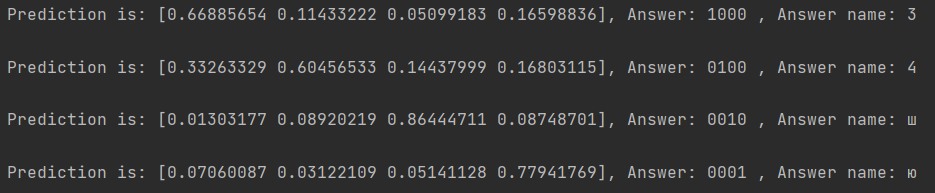


Рисунок 8 – Результат

Вывод: в результате выполнения работы была реализована многослойная нейронная сеть, обучающаяся методом обратного распространения ошибки.

Таким образом, были получены навыки создания простейших нейронных сетей, способных обрабатывать и классифицировать картинки с высоким разрешением.