

Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Кафедра информатики

Лабораторная работа №4  
«Нейронные сети.»

Выполнил: Чёрный Родион Павлович  
магистрант кафедры информатики  
группа №858642

Проверил: доцент кафедры информатики  
Стержанов Максим Валерьевич

Минск 2019

## Постановка задачи

Набор данных **ex4data1.mat** (такой же, как в лабораторной работе №2) представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 20x20 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 400 элементов. В результате загрузки набора данных должна быть получена матрица 5000x400. Далее расположены метки классов изображений от 1 до 9 (соответствуют цифрам от 1 до 9), а также 10 (соответствует цифре 0).

1. Загрузите данные **ex4data1.mat** из файла.
2. Загрузите веса нейронной сети из файла **ex4weights.mat**, который содержит две матрицы  $\Theta^{(1)}$  (25, 401) и  $\Theta^{(2)}$  (10, 26). Какова структура полученной нейронной сети?
3. Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации.
4. Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный результат с логистической регрессией.
5. Перекодируйте исходные метки классов по схеме one-hot.
6. Реализуйте функцию стоимости для данной нейронной сети.
7. Добавьте L2-регуляризацию в функцию стоимости.
8. Реализуйте функцию вычисления производной для функции активации.
9. Инициализируйте веса небольшими случайными числами.
10. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.
11. Для того, чтобы удостовериться в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром  $\epsilon = 10^{-4}$ .
12. Добавьте L2-регуляризацию в процесс вычисления градиентов.
13. Проверьте полученные значения градиента.
14. Обучите нейронную сеть с использованием градиентного спуска или других более эффективных методов оптимизации.
15. Вычислите процент правильных классификаций на обучающей выборке.
16. Визуализируйте скрытый слой обученной сети.
17. Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимости от данного параметра?
18. Ответы на вопросы представьте в виде отчета.

## Описание реализации

1. Загрузим из файла ex4data1.mat набор из 5000 изображений и метки их классов.

```
mat = scipy.io.loadmat('ex4data1.mat')
X = mat['X']
Y = mat['y']
```

2. Загрузим из файла ex4weights.mat веса нейронной сети.

```
trained_weights = scipy.io.loadmat("ex4weights.mat")
theta1 = trained_weights["Theta1"]
theta2 = trained_weights["Theta2"]
```

Матрица  $\Theta^{(1)}$  имеет размерность (25, 401), а матрица -  $\Theta^{(2)}$  (10, 26).

Структура нейронной сети следующая:

1. Слой входных данных длины 400 (+ нулевой элемент со значением 1), предназначенный для входного изображения 20x20 пикселей.
2. Скрытый слой длины 25 (+ нулевой элемент со значением 1).
3. Выходной слой длины 10, который отвечает за определение вероятности принадлежности входного изображения одному из 10 классов.

3. Реализация функции прямого распространения с сигмоидом в качестве функции активации:

```
def sigmoid(v):
    return 1.0 / (1 + np.exp(-v))
```

```
class SigmoidLayer:
    def __init__(self, weights, is_final_layer=False):
        self.weights = weights
        self.is_final_layer = is_final_layer

    def activate(self, input_vector):
        self.output = sigmoid(np.dot(self.weights, input_vector))
        if not self.is_final_layer:
            self.output = np.insert(self.output, 0, 1.0)
        return self.output
```

```
class NeuralNetwork:
    def __init__(self, layer_weights):
        self.layers = []
        self.layer_weights = layer_weights
        for layer_weight in layer_weights:
            self.layers.append(SigmoidLayer(layer_weight))
        self.layers[-1].is_final_layer = True
```

```
def propagate_forward(self, input_vector):
    input_vector = np.insert(input_vector, 0, 1.0)
    for layer in self.layers:
        output = layer.activate(input_vector)
        input_vector = output
    return input_vector
```

4. Процент правильных классификаций на обучающей выборке составляет 97.52%. При этом используя логистическую регрессию удавалось достичь 94%.

5. Перекодируем исходные метки классов по схеме one-hot:

```
def convert_to_one_hot(y_vector):
    result = []
    for y in y_vector:
        one_hot = np.zeros(10)
        one_hot[y - 1] = 1
        result.append(one_hot)
    return np.array(result)

y_one_hot = convert_to_one_hot(Y)
```

6, 7. Реализация функции стоимости для нашей нейронной сети с добавленной L2 регуляризацией

```
def cost(self, x, y):
    total_sum = 0
    m = len(x)
    for x_i, y_i in zip(x, y):
        result = self.propagate_forward(x_i)
        temp_sum = 0
        for output, y_i_k in zip(result, y_i):
            temp_sum += (y_i_k * np.log(output) + (1 - y_i_k) * np.log(1 - output))
        total_sum += temp_sum
    reg = (self.reg_coeff / 2) * np.sum([np.sum(theta**2) for theta in self.layer_weights])
    return (- total_sum - reg) / m
```

8. Функция вычисления производной от функции активации (сигмоида):

```
def sigmoid_derivative(z):
    return np.multiply(sigmoid(z), 1-sigmoid(z))
```

9. Функция инициализации весов небольшими случайными числами, лежащими в диапазоне от -eps до eps. Параметр eps возьмем равным 0.0001

```

def new(sizes):
    eps = 0.0001
    weights = []
    for i in range(len(sizes) - 1):
        in_size = sizes[i]
        out_size = sizes[i + 1]
        weight_matrix = np.random.rand(out_size, in_size + 1) * (2 * eps) - eps
        weights.append(weight_matrix)

```

10. Реализация алгоритма обратного распространения ошибки для текущей конфигурации нейронной сети:

```

def back_propagate(self, x, y):
    delta1, delta2 = self.compute_gradient(x, y)
    self.layer_weights[0] -= self.learning_coeff * delta1
    self.layers[0].weights = self.layer_weights[0]
    self.layer_weights[1] -= self.learning_coeff * delta2
    self.layers[1].weights = self.layer_weights[1]

def compute_gradient(self, x, y):
    delta1 = np.zeros(self.layer_weights[0].shape)
    delta2 = np.zeros(self.layer_weights[1].shape)
    m = len(y)
    for x_i, y_i in zip(x, y):
        ones = np.ones(1)
        a1 = np.hstack((ones, x_i))
        z2 = np.dot(a1, self.layer_weights[0].T)
        a2 = np.hstack((ones, sigmoid(z2)))
        z3 = np.dot(a2, self.layer_weights[1].T)
        a3 = sigmoid(z3)
        d3 = a3 - y_i
        d2 = np.multiply(np.dot(self.layer_weights[1].T, d3.T), np.multiply(a2, 1 - a2))
        delta1 = delta1 + np.outer(d2[1:], a1)
        delta2 = delta2 + np.outer(d3, a2)
    delta1 /= m
    delta2 /= m
    return delta1, delta2

```

11. Для того, чтобы убедиться, правильно ли вычисляется градиент, реализуем метод проверки градиента. Параметр `eps` возьмем равным 0.0001.

```
def check_gradient(self, gradient, x, y):
    eps = 1e-4
    layer_num = 1
    layer = self.layer_weights[layer_num]
    rows = len(layer)
    cols = len(layer[0])
    for i in range(rows):
        for j in range(cols):
            minus_layer = np.copy(layer)
            minus_layer[i][j] -= eps

            plus_layer = np.copy(layer)
            plus_layer[i][j] += eps

            plus_weights = list(self.layer_weights)
            plus_weights[layer_num] = plus_layer

            minus_weights = list(self.layer_weights)
            minus_weights[layer_num] = minus_layer

            plus_network = NeuralNetwork(minus_weights)
            minus_network = NeuralNetwork(plus_weights)

            approximation = (minus_network.cost(x, y) - plus_network.cost(x, y)) / (2 *
eps)
            diff = approximation - gradient[layer_num][i][j]
            print("Diff is " + str(diff))
```

12. L2 регуляризация уже добавлена в функцию вычисления градиента.

13. Проинициализируем веса малыми случайными величинами, вычислим значение градиента в текущей точке и проверим его правильность. Сделаем 5 проверок, меняя значение веса в случайных компонентах матрицы весов. Вычислим разницу между полученное значение компоненты градиента и тем, что мы высчитали ранее. Результаты вычисления:

```
Diff is -1.3741950286028093e-12
Diff is 1.6532421590574609e-12
Diff is -7.11869148697919e-13
Diff is 1.6929507925741993e-11
Diff is 6.407940367561538e-12
```

Такая малая разница дает нам основания считать, что градиент вычисляется корректно.

14. Проинициализируем веса малыми случайными величинами и произведем 30 итераций метода сопряженных градиентов. Воспользуемся реализацией метода сопряженных градиентов из пакета `sklearn`. Коэффициент регуляризации возьмем равным 0:

```
def train(self, x, y, reg_coeff=0):
    weights = self.initialize_weights(self.sizes)
    weights = np.append(weights [0].flatten(), weights [1].flatten())
    optimal_weights = opt.fmin_cg(
        maxiter=30,
        f=self.cost_func,
        x0=weights,
        fprime=self.back_propagate_func,
        args=(x, y, 1))
    self.layer_weights = self.reshape_weights(optimal_weights)
```

15. После обучения процент правильных классификаций на обучающей выборке составил 86.9%:

```
Warning: Maximum number of iterations has been exceeded.
Current function value: 0.903592
Iterations: 30
Function evaluations: 73
Gradient evaluations: 73
Correct predictions: 4345
Incorrect predictions: 655
Percent of correct predictions is 86.9
```

16. Скрытый слой нейронной сети изображен на рисунке 1.

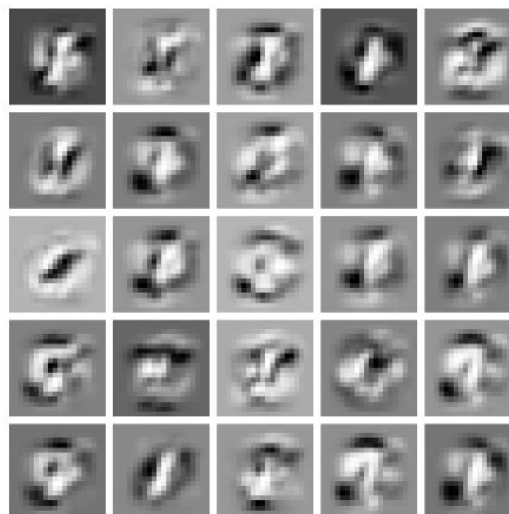


Рисунок 1 - Визуализация скрытого слоя нейронной сети

17. Подберем параметр регуляризации, перебирая его значения и сравнивая процент правильных классификаций после обучения модели. Также будем сравнивать визуализации скрытого слоя после каждого обучения. Результаты сравнения указаны в таблице 1.

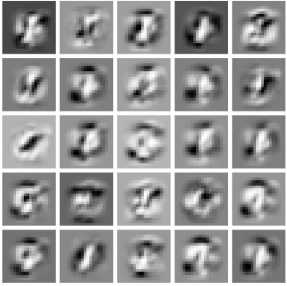
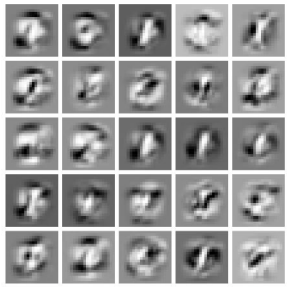

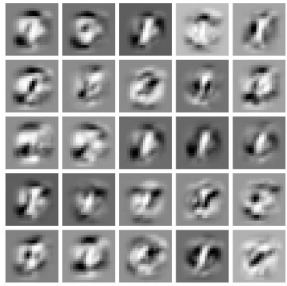
Коэффициент регуляризации	Значение функции стоимости	Процент правильных классификация	Визуализация скрытого слоя
0	0.932422	85.12	
0.0001	0.873	87.48	
0.001	0.818	88.46	
0.002	0.842	87.56	

Таблица 1 - Сравнение влияния коэффициента регуляризации на нейронную сеть



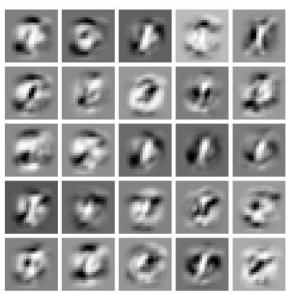
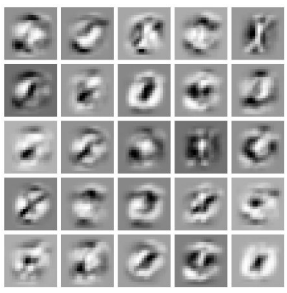
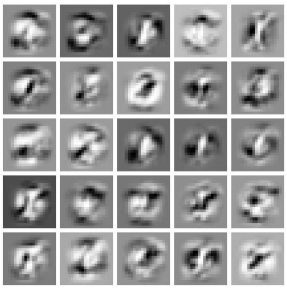
0.1	0.738	89.68	
0.2	0.957	84.72	
0.5	0.595	90.2	

Таблица 1 - Сравнение влияния коэффициента регуляризации на нейронную сеть

Несмотря на то, что при ненулевом коэффициенте регуляризации значение функции стоимости уменьшается (ввиду упрощения модели), процент правильных классификаций возрастает. Лучший результат - 90.2% правильных классификаций, имеем при коэффициенте регуляризации 0.5.

Узоры, если их можно так назвать, на изображениях внутреннего слоя не меняются. Изменение коэффициента регуляризации приводит к незначительному изменению яркости отдельных изображений.

## Выводы

Нейронная сеть с сигмной функцией активации, при достаточной натренированности, показывает лучшие результаты классификации чем логистическая регрессия. Но, при этом, требует больше временных затрат на ее обучение.

Когда мы имеем дело с большим количеством характеристик (например, для изображения 300x300px их число будет 90000) целесообразнее использовать именно нейронную сеть, так как она сможет находить более сложные закономерности - как раз из-за наличия скрытого слоя.