

## Лабораторная работа №4 “Нейронные сети”

Долматович Алина, 858641

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).

```
In [1]: 1 import pandas
        2 from scipy.io import loadmat
        3 import numpy as np
        4 import random
        5 import scipy.optimize as optimize
        6 import matplotlib.pyplot as pyplot
```

Загрузите данные ex4data1.mat из файла.

```
In [2]: 1 data = loadmat('ex4data1.mat')
        2 y = data["y"]
        3 x = data["X"]
        4 print(x.shape)
        5 print(y)
```

```
(5000, 400)
[[10]
 [10]
 [10]
 ...,
 [ 9]
 [ 9]
 [ 9]]
```

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?

```
In [3]: 1 weightsData = loadmat('ex4weights.mat')
        2 theta1 = weightsData["Theta1"]
        3 theta2 = weightsData["Theta2"]
        4 print(theta1.shape)
        5 print(theta2.shape)
        6
```

```
(25, 401)
```

```
(10, 26)
```

Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации.

```
In [4]: 1 def sigmoid(x):
        2     return 1 / (1 + np.exp(-x))
        3
        4 def getSigmoidData(ones, x, theta):
        5     if x.shape[1] != theta.shape[1]:
        6         x = np.hstack((ones, x))
        7     z = np.dot(x, theta.T)
        8     return sigmoid(z)
        9
        10 def h(theta1, theta2, x):
        11     m = len(x)
        12     ones = np.ones((m, 1))
        13     x = getSigmoidData(ones, x, theta1)
        14     return x, getSigmoidData(ones, x, theta2)
        15
        16 l1, l2 = h(theta1, theta2, x)
        17 print(l1.shape, l2.shape, x.shape)
```

```
((5000, 25), (5000, 10), (5000, 400))
```

Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный результат с логистической регрессией.

```
In [5]: 1 def predictionPercentValue(resultLayer, y):
2         predictions = np.argmax(resultLayer, axis=1) + 1
3         predictionsCount = 0
4         for predictionValue, realValue in zip(predictions, y):
5             if predictionValue == realValue:
6                 predictionsCount += 1
7         percentValue = float(predictionsCount) / len(y) * 100
8         return percentValue
9
10 predictionPercentValue(l2, y)
```

Out[5]: 97.52

Перекодируйте исходные метки классов по схеме one-hot.

```
In [6]: 1 oneHot = pandas.get_dummies(y.squeeze())
2 print(oneHot)
```

	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	1
18	0	0	0	0	0	0	0	0	0	1
19	0	0	0	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	0	0	0	1
22	0	0	0	0	0	0	0	0	0	1
23	0	0	0	0	0	0	0	0	0	1
24	0	0	0	0	0	0	0	0	0	1
25	0	0	0	0	0	0	0	0	0	1
26	0	0	0	0	0	0	0	0	0	1
27	0	0	0	0	0	0	0	0	0	1
28	0	0	0	0	0	0	0	0	0	1
29	0	0	0	0	0	0	0	0	0	1
...	..	..	..	..	..	..	..	..	..	..

```

4970  0  0  0  0  0  0  0  0  1  0
4971  0  0  0  0  0  0  0  0  1  0
4972  0  0  0  0  0  0  0  0  1  0

4973  0  0  0  0  0  0  0  0  1  0
4974  0  0  0  0  0  0  0  0  1  0
4975  0  0  0  0  0  0  0  0  1  0
4976  0  0  0  0  0  0  0  0  1  0
4977  0  0  0  0  0  0  0  0  1  0
4978  0  0  0  0  0  0  0  0  1  0
4979  0  0  0  0  0  0  0  0  1  0
4980  0  0  0  0  0  0  0  0  1  0
4981  0  0  0  0  0  0  0  0  1  0
4982  0  0  0  0  0  0  0  0  1  0
4983  0  0  0  0  0  0  0  0  1  0
4984  0  0  0  0  0  0  0  0  1  0
4985  0  0  0  0  0  0  0  0  1  0
4986  0  0  0  0  0  0  0  0  1  0
4987  0  0  0  0  0  0  0  0  1  0
4988  0  0  0  0  0  0  0  0  1  0
4989  0  0  0  0  0  0  0  0  1  0
4990  0  0  0  0  0  0  0  0  1  0
4991  0  0  0  0  0  0  0  0  1  0
4992  0  0  0  0  0  0  0  0  1  0
4993  0  0  0  0  0  0  0  0  1  0
4994  0  0  0  0  0  0  0  0  1  0
4995  0  0  0  0  0  0  0  0  1  0
4996  0  0  0  0  0  0  0  0  1  0
4997  0  0  0  0  0  0  0  0  1  0
4998  0  0  0  0  0  0  0  0  1  0
4999  0  0  0  0  0  0  0  0  1  0

```

```
15000 rows x 10 columns
```

Реализуйте функцию стоимости для данной нейронной сети.

```

In [7]: 1 def costFunction(h, y, sumParameter=False):
        2     costs = []
        3     for hI, yI in zip(h, y):
        4         cost = sum((hI - yI) ** 2)
        5         costs.append(cost)
        6     return sum(costs) if sumParameter == True else np.array(costs)
        7
        8 oneHot = np.array(oneHot)
        9 cost = costFunction(l2, oneHot, sumParameter=True)
       10 print(cost)

```

```
304.66188263
```

Добавьте L2-регуляризацию в функцию стоимости.

```
In [8]: 1 def costFunctionL2(theta1, theta2, y, lambda):
2         l1, l2 = h(theta1, theta2, x)
3         return sum(sum((l2 - y) ** 2) * lambda / (2*len(y)))
4
5 lambda = 10000
6 cost = costFunctionL2(theta1, theta2, oneHot, lambda)
7 print(cost)
```

304.66188263

Реализуйте функцию вычисления производной для функции активации.

```
In [9]: 1 def sigmoidDerivate(x):
2         return np.exp(-x) / ((1 + np.exp(-x)) ** 2)
3
```

Инициализируйте веса небольшими случайными числами.

```
In [16]: 1 def generateWeights(shape):
2         weights = []
3         for i in range(shape[0]):
4             line = []
5             for j in range(shape[1]):
6                 value = random.random()
7                 line.append(value)
8             weights.append(line)
9         return np.array(weights)
10
11
12 shape1 = (25, 401)
13 weights1 = generateWeights(shape1)
14 shape2 = (10, 26)
15 weights2 = generateWeights(shape2)
16 print(weights1.shape)
17 print(weights2.shape)
```

(25, 401)  
(10, 26)

Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.

```
In [11]: 1 for j in xrange(100):
2         l0 = x
3         l1, l2 = h(weights1, weights2, l0)
4         l2error = y - l2
5         if (j% 100) == 0:
6             print "Error:" + str(np.mean(np.abs(l2error)))
7         l2delta = l2error*sigmoidDerivate(l2)
8         l1error = l2delta.dot(weights2)
9         l1delta = l1error * sigmoidDerivate(l1)
10        weights1 += np.dot(l0.T, l1delta).T
11        weights2 += np.dot(l1.T, l2delta).T
12
```

Error:4.50000740655

Для того, чтобы удостовериться в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром  $\epsilon = 10^{-4}$ .

```
In [12]: 1 def gradient(weights, x, lambda=0, alpha=10e-4):
2         weights1, weights2 = weights[0], weights[1]
3         for j in xrange(100):
4             layer1 = sigmoid(np.dot(x, weights1.T))
5             layer2 = sigmoid(np.dot(layer1, weights2.T))
6             layer2delta = (layer2 - y) * (layer2 * (1-layer2))
7             layer1delta = np.dot(layer2delta, weights2) * (layer1 * (1-
8             l2Parameter = 1 - lambda/len(x)
9             weights2 = weights2 * l2Parameter - alpha * np.dot(layer1.
10            weights1 = weights1 * l2Parameter - alpha * np.dot(x.T, la
11            return np.array([weights1, weights2])
12
13 gradient([weights1, weights2], x)
```

```
Out[12]: array([ array([[ 9.44371080e-01,  1.20548325e-01,  3.41841525e-01
, ...,
-3.92934393e+05, -1.67927210e+04,  4.74412197e-01]],
[ 5.11302892e-01,  4.30194307e-01,  3.14657778e-01, ...,
-3.92934293e+05, -1.67935690e+04,  2.37830986e-01]],
[ 9.35294885e-01,  2.42716537e-02,  2.87289177e-01, ...,
-3.92933405e+05, -1.67927256e+04,  8.57284128e-01]],
...,
[ 3.27888681e-01,  4.66125296e-01,  3.84507094e-01, ...,
-3.92934279e+05, -1.67931734e+04,  7.49216550e-01]],
[ 6.04701939e-01,  5.91448105e-01,  6.71805820e-01, ...,
-3.92934465e+05, -1.67935775e+04,  9.69054377e-01]],
[ 5.22585685e-01,  8.25504367e-01,  9.09169621e-01, ...,
-3.92933617e+05, -1.67933137e+04,  5.50517327e-01]]),
array([[ 442377.33780564,  442376.95245053,  442377.60335074,
```

```
442377.33565965, 442377.81521366, 442377.44951588,
442377.45913141, 442377.38040217, 442377.05217094,
442377.22384933, 442377.05017671, 442377.62599652,
442377.15116521, 442377.87446756, 442377.19714859,
442377.73183692, 442377.44500665, 442376.98706223,
442377.21133876, 442377.33869702, 442377.07418048,
442376.89038705, 442376.9672255 , 442377.6902827 ,
442377.62075839],
[ 442377.6760308 , 442377.11424624, 442376.87667255,
442376.97064523, 442377.37828497, 442377.4744 ,
442377.02321059, 442377.40733642, 442377.53913507,
442377.49170005, 442377.07520374, 442377.7306359 ,
442377.83414677, 442377.20887069, 442377.19957571,
442377.265434 , 442377.44490618, 442377.48418849,
442377.0506541 , 442376.98787085, 442377.7220329 ,
442377.77087753, 442377.39832498, 442377.5736704 ,
442377.07579233],
[ 442377.27550228, 442377.66816328, 442377.10706844,
442377.12853179, 442377.1725485 , 442376.91628131,
442377.81804318, 442377.60920538, 442377.2987184 ,
442377.18046373, 442377.60279035, 442377.37459615,
442377.38169338, 442377.01873317, 442377.32729088,
442377.56736738, 442377.34262235, 442377.6821326 ,
442377.82471859, 442377.11496578, 442377.83794139,
442377.74310231, 442376.94399424, 442377.84623017,
442376.95869942],
[ 442377.76698935, 442376.93471252, 442377.13580026,
442376.93301228, 442377.68504964, 442377.6166221 ,
442377.02517106, 442377.28454848, 442377.48760976,
442377.58584865, 442377.59863045, 442377.31377653,
442377.44139643, 442377.44832442, 442377.78035808,
442377.09205903, 442377.27706399, 442377.21785764,
442377.67371141, 442377.01202416, 442377.03701648,
442376.97479044, 442377.20857641, 442377.58018586,
442377.34167761],
[ 442377.29103425, 442377.77511616, 442377.40892544,
442377.79026345, 442376.92638041, 442377.11205145,
442376.89851517, 442377.30002413, 442377.13439084,
442377.7856424 , 442377.1499268 , 442377.45126139,
442377.44755624, 442377.26257803, 442377.33316284,
442377.73131356, 442377.4759038 , 442377.39424736,
442377.50009672, 442377.59807165, 442377.52935838,
442377.67382872, 442377.49486791, 442377.41653313,
442377.31335238],
[ 442377.00147951, 442377.5586365 , 442376.98796604,
442377.0439275 , 442377.70305626, 442377.0979745 ,
442377.70811253, 442377.54058751, 442377.63926048,
442377.23110192, 442377.42382972, 442377.68499201,
442377.51458761, 442377.18686374, 442376.96389657,
442376.89631382, 442377.12267309, 442376.98761175,
442377.74333788, 442377.54755945, 442377.2104273 ,
442377.52740871, 442377.63191973, 442377.14107192,
442377.35826125],
```

```
[ 442377.78900595, 442376.96445023, 442376.9265066 ,
 442376.89046506, 442377.32427345, 442377.14896192,
 442377.24537871, 442377.62100195, 442377.6734643 ,
 442377.31847503, 442377.61144755, 442376.86758536,
 442377.04997626, 442377.64644552, 442377.25163395,
 442377.46186808, 442377.84519275, 442377.07145138,
 442377.60825459, 442377.72982291, 442377.7665389 ,
 442377.68559776, 442377.15892529, 442376.98559043,
 442377.80138378],
[ 442377.25779219, 442377.61903693, 442377.50553423,
 442377.72265892, 442377.23255179, 442377.72092837,
 442377.53004176, 442377.23608555, 442377.43122659,
 442377.4806274 , 442377.42265489, 442376.89021444,
 442377.103507 , 442377.71148675, 442377.08252423,
 442377.3194131 , 442377.10493532, 442377.25880792,
 442376.89869127, 442377.45705822, 442377.2275894 ,
 442377.71284119, 442377.47701446, 442377.76092498,
 442377.4678679 ],
[ 442377.41973464, 442377.24990433, 442376.9989195 ,
 442377.70962283, 442377.40675501, 442377.33215072,
 442377.53954564, 442377.60507623, 442377.07991909,
 442377.84709308, 442377.59854959, 442377.34381736,
 442377.66958752, 442377.70865045, 442376.86074572,
 442377.02624263, 442377.11394893, 442377.70440682,
 442377.59033433, 442377.07600733, 442377.32524666,
 442377.1404052 , 442377.27548268, 442377.69043851,
 442377.77419077],
[ 442376.9416131 , 442377.43215474, 442377.07609868,
 442377.73189374, 442377.37866791, 442377.103618 ,
 442377.13387306, 442377.61890375, 442377.66908269,
 442377.27459689, 442377.146192 , 442377.38535729,
 442377.14784724, 442377.0858367 , 442377.13692537,
 442377.48577152, 442377.59204845, 442377.09185438,
 442377.85682265, 442377.16250027, 442377.36812611,
 442377.48432931, 442377.41682913, 442377.28500618,
 442377.66453424]]), dtype=object)
```

Добавьте L2-регуляризацию в процесс вычисления градиентов.

```
In [13]: 1 weights = gradient([weights1, weights2], x, 1)
          2 weights1, weights2 = weights[0], weights[1]
```

Проверьте полученные значения градиента.

```
In [14]: 1 l1, l2 = h(weights1, weights2, x)
          2 print(l1.shape, l2.shape, x.shape)

((5000, 25), (5000, 10), (5000, 400))
```



Обучите нейронную сеть с использованием градиентного спуска или других более эффективных методов оптимизации.

```
In [17]: 1 y = data["y"]
2 x = data["X"]
3 trainWeights1 = generateWeights((25, 401))
4 trainWeights2 = generateWeights((10, 26))
5 trainWeights = gradient([trainWeights1, trainWeights2], x, 0.00000
6 trainWeights1, trainWeights2 = trainWeights[0], trainWeights[1]
7 l1, l2 = h(trainWeights1, trainWeights2, x)
```

Вычислите процент правильных классификаций на обучающей выборке.

```
In [19]: 1 predictionPercentValue(l2, y)
```

Out[19]: 97.52

Визуализируйте скрытый слой обученной сети.

```
In [20]: 1 def plotL1(l1):
2     count = 25
3     _, axis = pyplot.subplots(1, count)
4
5     for j in range(count):
6         matrix = l1.T[j].reshape(50, 100, order="F")
7         axis[j].imshow(matrix)
8         axis[j].axis("off")
9
10    pyplot.show()
11
12    plotL1(l1)
13
```



Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимости от данного параметра?

In [25]:

```
1 lmbdas = [1000000000, 1000, 0, 0.00001]
2
3 for lambda in lmbdas:
4     trainWeights = gradient([trainWeights1, trainWeights2], x, lmb
5     trainWeights1, trainWeights2 = trainWeights[0], trainWeights[1
6     l1, l2 = h(trainWeights1, trainWeights2, x)
7     print(lambda)
8     plotL1(l1)
9
```

1000000000



1000



0



1e-05



## Вывод

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными применениями нейронных сетей является:

- Классификация
- Предсказание
- Распознавание Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной. Также есть нейрон смещения и контекстный нейрон о которых мы поговорим в следующей статье. В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию,  $n$  скрытых слоев, которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона:  $input=output$ . В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации (пока что просто представим ее  $f(x)$ ) и попадает в поле output.