Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

ЛабоРАТОРНАЯ РАБОТА №4

«Нейронные сети»

Выполнил: Яловчук Валерий Валерьевич

магистрант кафедры информатики

группа №858641

Проверил: доцент, кандидат технических наук Стержанов Максим Валерьевич

Минск 2019

ХОД РАБОТЫ

**Задание.**

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).

**Выполнение:**

1. Загрузите данные ex4data1.mat из файла:

file\_path = 'ex4data1.mat'

data = sio.loadmat(file\_path)

y = data.get('y')

X = data.get('X')

2. Загрузите веса нейронной сети из файла ex4weights.mat, который содержит две матрицы Θ(1) (25, 401) и Θ(2) (10, 26). Какова структура полученной нейронной сети?

file\_path = 'ex4weights.mat'

weights = sio.loadmat(file\_path)

theta1 = weights.get('Theta1')

theta2 = weights.get('Theta2')

nn\_params = np.hstack((theta1.ravel(order='F'), theta2.ravel(order='F'))) # unroll parameters

# neural network hyperparameters

hidden\_layer\_size = len(theta2[0]) - 1

lmbda = 1

print("hidden\_layer\_size: ", hidden\_layer\_size)

Input layer: 400 + 1

Hidden layer: 25 + 1

Output layer: 10.

3. Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации:

def sigmoid(z):

return 1 / (1 + np.exp(-z))

def h0x(X, theta):

m = len(X)

ones = np.ones((m, 1))

a1 = np.hstack((ones, X))

a2 = sigmoid(np.dot(a1, theta[0].T))

a2 = np.hstack((ones, a2))

h = sigmoid(np.dot(a2, theta[1].T))

return h

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

pred = np.argmax(pred, axis=1) + 1

predictions = 0

for i in range(len(pred)):

if pred[i] == y[i][0]:

predictions += 1

print("Accuracy: ", predictions / len(y) \* 100)

Результат выполнения: 97.52%. У логистической регрессией этот показатель был 95.12. Можно сделать вывод, что нейронная сеть справилась лучше с поставленной задачей, т. к. итоговый показатель правильно распознанных образов выше примерно на 2.5%.

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

y\_one\_hot = pd.get\_dummies(y.flatten())

Класс преобразуется в вектор: 4 - [0 0 0 1 0 0 0 0 0 0].

6-7. Реализуйте функцию стоимости для данной нейронной сети. Добавьте L2-регуляризацию в функцию стоимости:

def cost\_func(nn\_params, hidden\_layer\_size, X, y\_d, lmbda):

input\_layer\_size = len(X[0]) # 400

num\_labels = len(y\_one\_hot.values[0]) # 10

m = len(X)

theta1 = np.reshape(nn\_params[:hidden\_layer\_size\*(input\_layer\_size+1)], (hidden\_layer\_size, input\_layer\_size+1), 'F')

theta2 = np.reshape(nn\_params[hidden\_layer\_size\*(input\_layer\_size+1):], (num\_labels, hidden\_layer\_size+1), 'F')

h = h0x(X, [theta1, theta2])

temp1 = np.multiply(y\_d, np.log(h))

temp2 = np.multiply(1 - y\_d, np.log(1-h))

temp3 = np.sum(temp1 + temp2)

sum1 = np.sum(np.sum(np.power(theta1[:, 1:], 2), axis=1))

sum2 = np.sum(np.sum(np.power(theta2[:, 1:], 2), axis=1))

return np.sum(temp3 / (-m)) + (sum1 + sum2) \* lmbda / (2\*m)

8. Реализуйте функцию вычисления производной для функции активации:

def sigmoid\_derivative(z):

return np.multiply(sigmoid(z), 1-sigmoid(z))

9. Инициализируйте веса небольшими случайными числами:

def rand\_weights(L\_in, L\_out):

epi = (6 \*\* 1/2) / (L\_in + L\_out) \*\* 1/2

W = np.random.rand(L\_out, L\_in + 1) \* (2 \* epi) - epi

return W

input\_layer\_size = len(X[0]) # 400

num\_labels = len(y\_one\_hot.values[0]) # 10

initial\_Theta1 = rand\_weights(input\_layer\_size, hidden\_layer\_size)

initial\_Theta2 = rand\_weights(hidden\_layer\_size, num\_labels)

initial\_nn\_params = np.append(initial\_Theta1.flatten(), initial\_Theta2.flatten())

10. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети:

def back\_propagation(nn\_params, hidden\_layer\_size, X, y\_d, lmbda):

input\_layer\_size = len(X[0]) # 400

num\_labels = len(y\_d.values[0]) # 10

initial\_theta1 = np.reshape(nn\_params[:hidden\_layer\_size\*(input\_layer\_size+1)], (hidden\_layer\_size, input\_layer\_size+1), 'F')

initial\_theta2 = np.reshape(nn\_params[hidden\_layer\_size\*(input\_layer\_size+1):], (num\_labels, hidden\_layer\_size+1), 'F')

delta1 = np.zeros(initial\_theta1.shape)

delta2 = np.zeros(initial\_theta2.shape)

m = len(y\_d)

for i in range(X.shape[0]):

ones = np.ones(1)

a1 = np.hstack((ones, X[i]))

z2 = np.dot(a1, initial\_theta1.T)

a2 = np.hstack((ones, sigmoid(z2)))

z3 = np.dot(a2, initial\_theta2.T)

a3 = sigmoid(z3)

d3 = a3 - y\_d.iloc[i, :][np.newaxis, :]

z2 = np.hstack((ones, z2))

d2 = np.multiply(np.dot(initial\_theta2.T, d3.T), sigmoid\_derivative(z2).T[:,np.newaxis])

delta1 = delta1 + np.dot(d2[1:, :], a1[np.newaxis, :])

delta2 = delta2 + np.dot(d3.T, a2[np.newaxis, :])

delta1 /= m

delta2 /= m

delta1[:, 1:] = delta1[:, 1:] + initial\_theta1[:, 1:] \* lmbda / m

delta2[:, 1:] = delta2[:, 1:] + initial\_theta2[:, 1:] \* lmbda / m

return np.hstack((delta1.ravel(order='F'), delta2.ravel(order='F')))

backprop\_params = back\_propagation(initial\_nn\_params, hidden\_layer\_size, X, y\_one\_hot, 0)

11. Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 10-4:

def gradient\_check(nn\_initial\_params, nn\_backprop\_params, hidden\_layer\_size, X, y\_d, lmbda):

myeps = 0.0001

flattened = nn\_initial\_params

flattenedDs = nn\_backprop\_params

n\_elems = len(flattened)

# Pick ten random elements, compute numerical gradient, compare to respective D's

for i in range(10):

x = int(np.random.rand()\*n\_elems)

epsvec = np.zeros((n\_elems, 1))

epsvec[x] = myeps

cost\_high = cost\_func(flattened + epsvec.flatten(), hidden\_layer\_size, X, y\_d, lmbda)

cost\_low = cost\_func(flattened - epsvec.flatten(), hidden\_layer\_size, X, y\_d, lmbda)

mygrad = (cost\_high - cost\_low) / float(2\*myeps)

print("Element: {0}. Numerical Gradient = {1:.9f}. BackProp Gradient = {2:.9f}."

.format(x, mygrad, flattenedDs[x]))

gradient\_check(initial\_nn\_params, backprop\_params, hidden\_layer\_size, X, y\_one\_hot, 0)

12-13. Добавьте L2-регуляризацию в процесс вычисления градиентов. Проверьте полученные значения градиента:

backprop\_params = back\_propagation(initial\_nn\_params, hidden\_layer\_size, X, y\_one\_hot, lmbda)

gradient\_check(initial\_nn\_params, backprop\_params, hidden\_layer\_size, X, y\_one\_hot, lmbda)

Результат выполнения:

Element: 986. Numerical Gradient = -0.000000734. BackProp Gradient = -0.000000734.

Element: 1521. Numerical Gradient = 0.000000711. BackProp Gradient = 0.000000711.

Element: 2965. Numerical Gradient = 0.000085671. BackProp Gradient = 0.000085671.

Element: 3249. Numerical Gradient = -0.000711440. BackProp Gradient = -0.000711440.

Element: 9632. Numerical Gradient = 0.000002351. BackProp Gradient = 0.000002351.

Element: 8061. Numerical Gradient = -0.000000715. BackProp Gradient = -0.000000715.

Element: 3879. Numerical Gradient = -0.003488895. BackProp Gradient = -0.003488895.

Element: 7718. Numerical Gradient = 0.000508230. BackProp Gradient = 0.000508230.

Element: 4952. Numerical Gradient = -0.001355993. BackProp Gradient = -0.001355993.

Element: 6969. Numerical Gradient = 0.000061777. BackProp Gradient = 0.000061777.

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

theta\_opt = opt.fmin\_cg(maxiter=30, f=cost\_func, x0=initial\_nn\_params, fprime=back\_propagation, args=(hidden\_layer\_size, X, y\_one\_hot, lmbda))

print(theta\_opt)

theta1\_opt = np.reshape(theta\_opt[:hidden\_layer\_size\*(input\_layer\_size+1)], (hidden\_layer\_size, input\_layer\_size+1), 'F')

theta2\_opt = np.reshape(theta\_opt[hidden\_layer\_size\*(input\_layer\_size+1):], (num\_labels, hidden\_layer\_size+1), 'F')

15. Вычислите процент правильных классификаций на обучающей выборке:

pred = h0x(X, [theta1\_opt, theta2\_opt])

pred = np.argmax(pred, axis=1) + 1

predictions = 0

for i in range(len(pred)):

if pred[i] == y[i][0]:

predictions += 1

print("Accuracy: ", predictions / len(y) \* 100)

Результат выполнения:

'Accuracy: ', 93.84

16. Визуализируйте скрытый слой обученной сети:

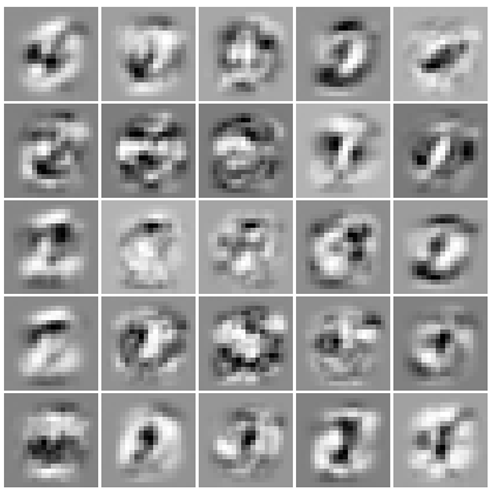


Рисунок 1 – визуализация скрытого слоя при lambda=1

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

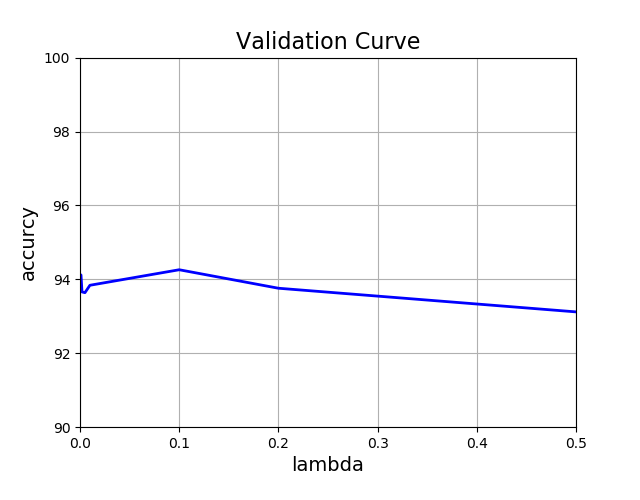


Рисунок 2 – точность предсказаний при различных lambda (на рисунке видно, что наибольшая точность достигается при lambda=0.1)

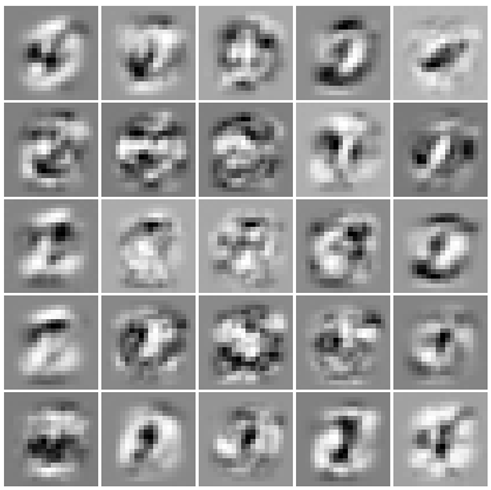


Рисунок 3 – визуализация скрытого слоя при lambda=0.1 (optimal)

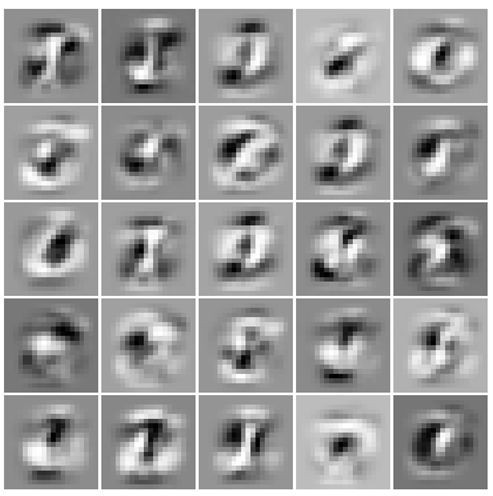


Рисунок 4 – визуализация скрытого слоя при lambda=100 (high bias)