Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе #4
«Нейронные сети»

Выполнил: Карп Александр Игоревич
магистрант кафедры информатики
группа №858641

Проверил: магистр технических наук
Стержанов Максим Валерьевич

Минск 2019

1. Загрузите данные ex4data1.mat из файла.

```
#1
data = sio.loadmat('ex4data1.mat')
X = data.get('X')
y = data.get('y')
```

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?

```
#2
weights = sio.loadmat('ex4weights.mat')
theta1 = weights['Theta1']
theta2 = weights['Theta2']
```

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

   Код функции сигмоиды и прямого распространения:

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

def hx(theta1, theta2, X):
    m = len(y)
    ones = np.ones((m, 1))
    a1 = np.hstack((ones, X))
    a2 = sigmoid(a1 @ theta1.T)
    a2 = np.hstack((ones, a2))
    h = sigmoid(a2 @ theta2.T)
    return h
```

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

Код подсчета отношения правильных ответов к общему числу:

```
def accuracy(y_pred, y_true):
    error = 0
    for I in range(len(y_pred)):
        if y_pred[i] != y_true[i][0]:
            error += 1
    return 1 - error / len(y_pred)
```

Точность с использование нейронной сет: 0.9752
С использованием логистической регрессии: 0.9648

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

```
y_d = pd.get_dummies(y.flatten())
```

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

```
def loss(nn_params, input_layer_size, hidden_layer_size,
num_labels, X, y, lmbda):
    m = len(y)
    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')
    y = pd.get_dummies(y.flatten())
    h = hx(theta1, theta2, X)
    temp1 = np.multiply(y, np.log(h))
    temp2 = np.multiply(1 - y, 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 dsig(z):
    return np.multiply(sigmoid(z), 1-sigmoid(z))
```

9. Инициализируйте веса небольшими случайными числами.

```
def randInitializeWeights(L_in, L_out):
    epsilon = 0.12
    return np.random.rand(L_out, L_in+1) * 2 * epsilon
- epsilon

initial_theta1 =
randInitializeWeights(input_layer_size,
hidden_layer_size)
initial_theta2 =
randInitializeWeights(hidden_layer_size, num_labels)
```

10-13. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.
Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 10-4.
Добавьте L2-регуляризацию в процесс вычисления градиентов.
Проверьте полученные значения градиента.

```
def gradient(nn_params, input_layer_size, hidden_layer_size, num_labels, X, y,
lmbda):
    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')
    y_d = pd.get_dummies(y.flatten())
    delta1 = np.zeros(initial_theta1.shape)
    delta2 = np.zeros(initial_theta2.shape)
    m = len(y)

    ones = np.ones((X.shape[0], 1))
    a1 = np.hstack((ones, X))
    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
    z2 = np.hstack((ones, z2))
```

```
    d2 = np.multiply(np.dot(initial_theta2.T,d3.T), dsig(z2).T)
    delta1 = delta1 + np.dot(d2[1:, :], a1)
    delta2 = delta2 + np.dot(d3.T, a2)

    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')))
```

```
def checkGradient(nn_initial_params,nn_backprop_Params,input_layer_size,
hidden_layer_size, num_labels,myX,myy,mylambda=0., e= 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] = e

        cost_high = loss(flattened + epsvec.flatten(),input_layer_size,
hidden_layer_size, num_labels,myX,myy,mylambda)
        cost_low  = loss(flattened - epsvec.flatten(),input_layer_size,
hidden_layer_size, num_labels,myX,myy,mylambda)
        mygrad = (cost_high - cost_low) / float(2*e)
        print("Element: {0}. Numerical Gradient = {1:.9f}. BackProp Gradient =
{2:.9f}.".format(x,mygrad,flattenedDs[x]))
```

Результат проверки градиента при $\lambda = 0$ (без регуляризации)

```
checkGradient(nn_initial_params,nn_backprop_Params,input_layer_size,
hidden_layer_size, num_labels,X,y,0)
```

Element: 6752. Numerical Gradient = -0.015064415. BackProp Gradient = -0.015064415.
Element: 3870. Numerical Gradient = 0.000366763. BackProp Gradient = 0.000366763.
Element: 1728. Numerical Gradient = -0.000023502. BackProp Gradient = -0.000023502.
Element: 5231. Numerical Gradient = -0.001285017. BackProp Gradient = -0.001285017.
Element: 3014. Numerical Gradient = 0.000047936. BackProp Gradient = 0.000047936.
Element: 2954. Numerical Gradient = 0.001852073. BackProp Gradient = 0.001852074.

Element: 9331. Numerical Gradient = -0.000008899. BackProp Gradient = -0.000008899.
Element: 4601. Numerical Gradient = -0.004389783. BackProp Gradient = -0.004389783.
Element: 7498. Numerical Gradient = 0.000050507. BackProp Gradient = 0.000050507.
Element: 2826. Numerical Gradient = -0.004238061. BackProp Gradient = -0.004238061.

Результат проверки градиента при $\lambda = 1$
Element: 5126. Numerical Gradient = -0.008648458. BackProp Gradient = -0.008647272.
Element: 7559. Numerical Gradient = -0.000040537. BackProp Gradient = -0.000030239.
Element: 3223. Numerical Gradient = 0.004780599. BackProp Gradient = 0.004780404.
Element: 1748. Numerical Gradient = 0.000494324. BackProp Gradient = 0.000494796.
Element: 9243. Numerical Gradient = 0.000000213. BackProp Gradient = -0.000004726.
Element: 10052. Numerical Gradient = 0.197726268. BackProp Gradient = 0.197743894.
Element: 7580. Numerical Gradient = -0.000268463. BackProp Gradient = -0.000285402.
Element: 4600. Numerical Gradient = 0.002757815. BackProp Gradient = 0.002757244.
Element: 935. Numerical Gradient = -0.000015960. BackProp Gradient = -0.000001618.
Element: 5388. Numerical Gradient = -0.006800920. BackProp Gradient = -0.006808652.

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

```
theta_opt = opt.fmin_cg(maxiter = 100, f = loss, x0 =
nn_initial_params, fprime = gradient, args =
(input_layer_size, hidden_layer_size, num_labels, X, y,
lmbda))
theta1_opt =
np.reshape(theta_opt[:hidden_layer_size*(input_layer_si
ze+1)], (hidden_layer_size, input_layer_size+1), 'F')
```

```
theta2_opt =
np.reshape(theta_opt[hidden_layer_size*(input_layer_siz
e+1):], (num_labels, hidden_layer_size+1), 'F')
```

15. Вычислите процент правильных классификаций на обучающей выборке.

Функция предсказания результатов:

```
def predict(theta1, theta2, X):
    return np.argmax(hx(theta1,theta2,X), axis = 1) + 1
```

```
y_pred = predict(theta1_opt, theta2_opt, X)
print("Accuracy: ", accuracy(y_pred, y))
```

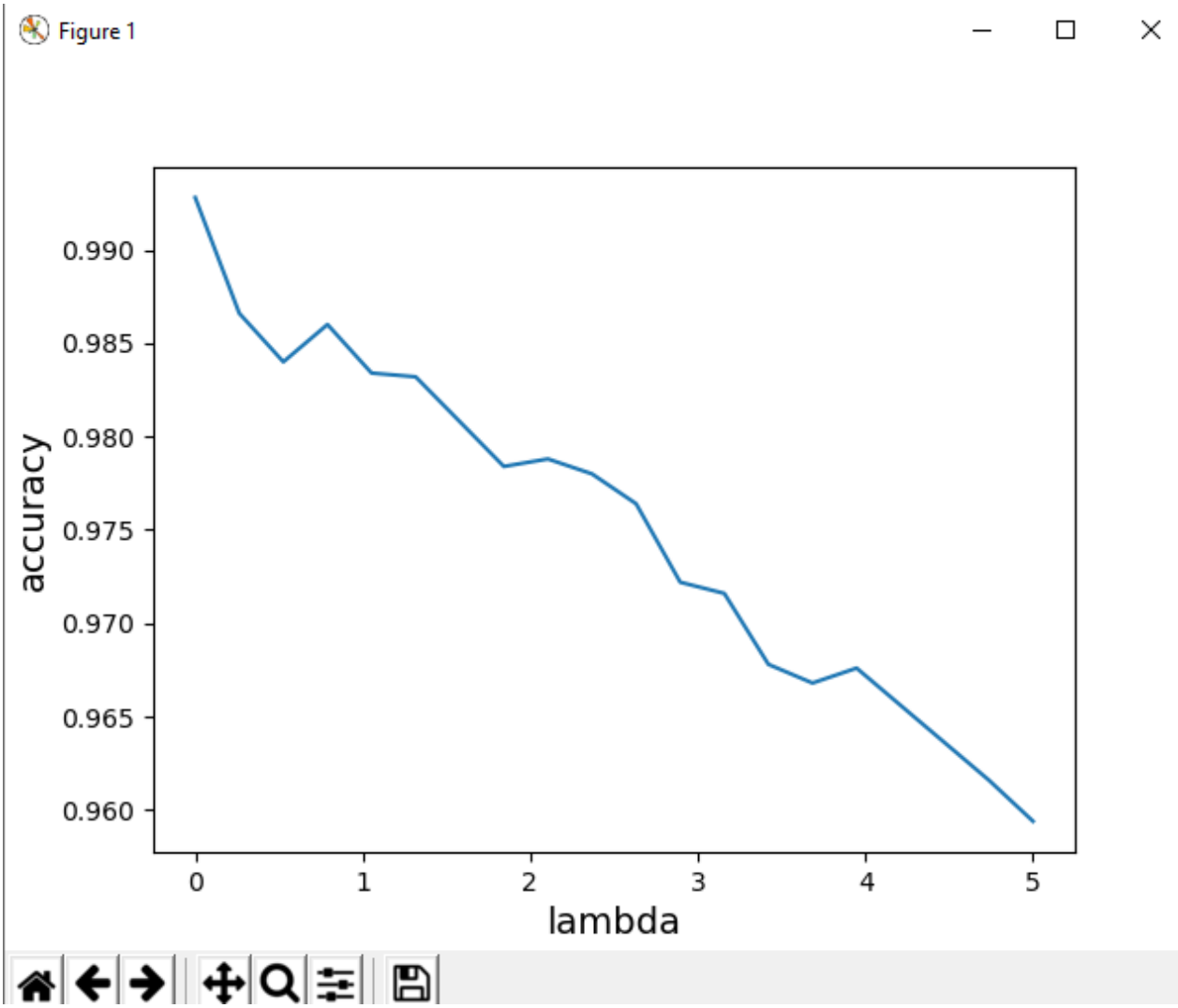
$$Accuracy: 0.986$$

17. Подберите параметр регуляризации.

```
#17

lambdas = np.linspace(0,5, 20)
preds = []
for l in lambdas:
    theta_opt = opt.fmin_cg(maxiter=100, f=loss, x0=nn_initial_params,
fprime=gradient,
                            args=(input_layer_size, hidden_layer_size,
num_labels, X, y, l))
    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')
    y_pred = predict(theta1_opt, theta2_opt, X)
    preds.append(accuracy(y_pred, y))

plt.plot(lambdas, preds)
plt.xlabel("lambda", fontsize=14)
plt.ylabel("accuracy", fontsize=14)
plt.show()
```



Как видно из графика, наилучший результат при $\lambda = 0$
Это обьясняется тем, что мы проверяем точность на тренировочных данных. Точность на тренировочных данных при увеличении параметра регуляризации уменьшается, так как регуляризация сглаживает нашу модель (при этом на валидации точность повышается)

Выводы

Мы изучили двуслойную нейронную сеть с обратным распространением ошибки. Научились строить данную нейронную сеть, настраивать ее параметры, добавлять регуляризацию, обучать сеть и предсказывать с ее помощью.