Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №4
«Нейронные сети»

БГУИР 1-40 81 04

Магистрант:
гр. 858642
Кукареко А.В.

Проверил:
Стержанов М. В.

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).

1. Загрузите данные ex4data1.mat из файла.
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?
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 удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 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.

Результат выполнения:

1. Загрузите данные ex4data1.mat из файла.

```
img_data = scipy.io.loadmat('ex4data1.mat')

X, Y = img_data['X'], img_data['y']
```

```
print(f'X.shape = {X.shape}')
print(f'Y.shape = {Y.shape}')
```

```
X.shape = (5000, 400)
Y.shape = (5000, 1)
```

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?

Загрузка весов:

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

print(f'theta1.shape = {theta1.shape}')
print(f'theta2.shape = {theta2.shape}')
```

```
theta1.shape = (25, 401)
theta2.shape = (10, 26)
```

Структура нейросети состоит из 3х слоев:

- входной слой - 400 нейронов + 1 bias:
- скрытый слой – 25 нейронов + 1 bias;
- выходной слой – 10 нейронов.

Количество нейронов выходного слоя равно количеству классов.

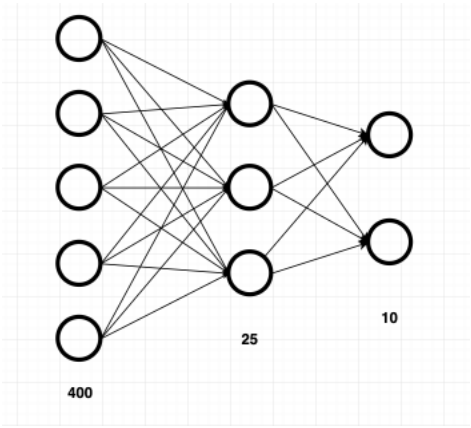

Рисунок 1 – структура нейросети.

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

Функция сигмоида:

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

Вспомогательные функции:

```
def add_bias_vec(a):
    return np.insert(a,0,1,axis=1)

def rm_bias(input):
    return input[:, 1:]
```

Функцию прямого распространения:

```
def forward_prop_vec_all(thetas, X):
    a1 = add_bias_vec(X)
    z2 = np.dot(a1, thetas[0].T)
    a2 = sigmoid(z2)

    a2 = add_bias_vec(a2)
    z3 = np.dot(a2, thetas[1].T)
    a3 = sigmoid(z3)

    return {'a1': a1, 'z2': z2, 'a2': a2, 'z3': z3, 'a3': a3}

def forward_prop_vec(thetas, X):
    return forward_prop_vec_all(thetas, X)['a3']
```

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

Функция предсказания:

```
def predict(thetas, x):
    if x.ndim == 1:
        x = x.reshape(1, -1)

    fp_res = forward_prop_vec(thetas, x)

    return np.argmax(fp_res[0]) + 1
```

Функция подсчета “accuracy”:

```
def calc_accuracy(thetas, X, Y):
    m = X.shape[0]
    correct = 0

    for i in range(m):
        if predict(thetas, X[i]) == Y[i]:
            correct += 1

    return correct/m
```

```
print("Accuracy: %0.1f%%"%(100*calc_accuracy(thetas, X, Y)))
```

```
Accuracy: 97.5%
```

Нейросеть дала точность - 99.0%, что на несколько процентов больше, чем логистическая регрессия, которая дала - 97.2%.

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

Функция кодирования в «one-hot»:

```
def one_hot(labels):
    m = len(labels)
    uniq_labels = np.unique(labels)

    return (labels == uniq_labels).astype(int)
```

Результат работы:

```
Y_oh = one_hot(Y)
Y_oh[0]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
```

6. Реализуйте функцию стоимости для данной нейронной сети.

Функция стоимости:

```
def J(thetas, X, Y_one_hot, lmb = 0.):
    m = len(X)
    h = forward_prop_vec(thetas, X)

    e1 = np.multiply(Y_one_hot, np.log(h))
    e2 = np.multiply((1 - Y_one_hot), np.log(1 - h))

    regularization = 0
    cost = (-1 / m) * np.sum(e1 + e2)

    # Не помню, нужно ли в регуляризации учитывать баес? В Лог регрессии - не нужно было
    if lmb !=0:
        reg_sum = np.sum(np.power(rm_bias(thetas[0]), 2)) + np.sum(np.power(rm_bias(thetas[1]), 2))
        regularization = (lmb / (2 * m)) * reg_sum

    return cost + regularization

def J_unroll(unroll_thetas, X, Y_one_hot, lmb = 0.):
    return J(rehape_thetas(unroll_thetas), X, Y_one_hot, lmb)
```

Результат работы функции стоимости:

```
: J(thetas, X, Y_oh)
```

```
: 0.2876291651613189
```

7. Добавьте L2-регуляризацию в функцию стоимости.

Функция стоимости с регуляризацией L2:

```
J(thetas, X, Y_oh, 0.4)
```

```
0.3260854427331608
```

8. Реализуйте функцию вычисления производной для функции активации.

Функция производная от функции активации:

```
def der_sigmoid(a):
    return np.multiply(a, 1 - a)
```

```
der_sigmoid(np.array([1, 2, 3]))
```

```
array([ 0, -2, -6])
```

9. Инициализируйте веса небольшими случайными числами..

Функция инициализации весов случайными числами:

```
def gen_thetas(eps = 0.1):
    t1 = np.random.rand(nn_params['layer_1_output'], nn_params['layer_1_input']) * (2 * eps) - eps
    t2 = np.random.rand(nn_params['layer_2_output'], nn_params['layer_2_input']) * (2 * eps) - eps

    return np.array([t1, t2])
```

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

Функция обратного распространения ошибки:

```
def back_prop_vec(unrolled_thetas, X, Y_one_hot, lmb = 0.):
    #print('call - back_prop_vec')
    m = len(X)
    thetas = rehape_thetas(unrolled_thetas)

    delta_1 = np.zeros(thetas[0].shape)
    delta_2 = np.zeros(thetas[1].shape)

    for i in range(m):
        fp_data = forward_prop_vec_all(thetas, X[i].reshape(1, -1))

        d3 = fp_data['a3'] - Y_one_hot[i] # слой выхода

        d2 = rm_bias(np.multiply(np.dot(d3, thetas[1]), der_sigmoid(fp_data['a2'])))

        delta_1 += np.dot(d2.T, fp_data['a1'])
        delta_2 += np.dot(d3.T, fp_data['a2'])

    delta_1 /= m
    delta_2 /= m

    if lmb !=0:
        lmb_mult = ( lmb / m )
        delta_1[:, 1:] += lmb_mult * rm_bias(thetas[0])
        delta_2[:, 1:] += lmb_mult * rm_bias(thetas[1])

    return unroll_thetas(np.array([delta_1, delta_2]))
```

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

Функция проверки градиента:

```
def gd_check(experiments_count, unrolled_thetas, back_prop_thetas, X, Y_one_hot, lmb = 0.):
    eps = 0.0001
    theta_count = len(unrolled_thetas)

    for i in range(experiments_count):
        idx = int(np.random.rand() * theta_count)

        experiment_thetas = np.copy(unrolled_thetas)
        orig_val = experiment_thetas[idx]

        experiment_thetas[idx] = orig_val + eps
        cost_plus = J_unroll(experiment_thetas, X, Y_one_hot, lmb)

        experiment_thetas[idx] = orig_val - eps
        cost_minus = J_unroll(experiment_thetas, X, Y_one_hot, lmb)

        calc_g = (cost_plus - cost_minus) / (2 * eps)

        print(f'Idx: {idx} check gradient: {calc_g:f}, BP gradient: {back_prop_thetas[idx]:f}')
```

Пример работы функции градиента:

```
gd_check(5, unroll_thetas(thetas), back_prop_thetas, X, Y_oh, 0)
```

```
Idx: 8448 check gradient: 0.000001, BP gradient: 0.000001
Idx: 2533 check gradient: 0.000317, BP gradient: 0.000317
Idx: 801 check gradient: 0.000000, BP gradient: 0.000000
Idx: 1284 check gradient: -0.000000, BP gradient: -0.000000
Idx: 4018 check gradient: -0.000001, BP gradient: -0.000001
```

12. Добавьте L2-регуляризацию в процесс вычисления градиентов.

Функция вычисления градиента с L2-регуляризацией:

```
back_prop_thetas_reg = back_prop_vec(unroll_thetas(thetas), X, Y_oh, 0.5)
back_prop_thetas_reg
```

```
array([ 6.18712766e-05, -1.05624163e-12,  2.19414684e-13, ...,
        7.18308933e-05, -6.29727728e-04,  6.40577831e-04])
```

13. Проверьте полученные значения градиента.

Результат проверки градиента с L2-регуляризацией:

```
gd_check(5, unroll_thetas(thetas), back_prop_thetas_reg, X, Y_oh, 0.5)
```

```
Idx: 929 check gradient: -0.000100, BP gradient: -0.000100
Idx: 9461 check gradient: -0.000216, BP gradient: -0.000216
Idx: 3770 check gradient: 0.000002, BP gradient: 0.000002
Idx: 9094 check gradient: 0.000013, BP gradient: 0.000013
Idx: 804 check gradient: 0.000000, BP gradient: 0.000000
```

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

Функция обучения нейросети:

```
def fit(X, Y, lmb = 0., maxiter = 30):

    rand_thetas = gen_thetas()
    unrolled_thetas = unroll_thetas(rand_thetas)

    Y_one_hot = one_hot(Y)

    back_prop_vec(unroll_thetas(thetas), X, Y_oh, 0.5)
    #result = optimize.minimize(J_unroll, jac=back_prop_vec, x0=unrolled_thetas, args=(X, Y_one_hot, lmb
    #out_thetas = rehape_thetas(result.x)
    result = optimize.fmin_cg(maxiter=maxiter, f=J_unroll, x0=unrolled_thetas, fprime=back_prop_vec,
                              args=(X, Y_one_hot, lmb))
    out_thetas = rehape_thetas(result)

    return out_thetas
```

15. Вычислите процент правильных классификаций на обучающей выборке.

Результат:

```
print("NN accuracy: %0.1f%%"%(100*calc_accuracy(fitted_thetas, X, Y)))
```

```
NN accuracy: 99.0%
```

Accuracy - 99.0% при 75 итерациях.

16. Визуализируйте скрытый слой обученной сети.

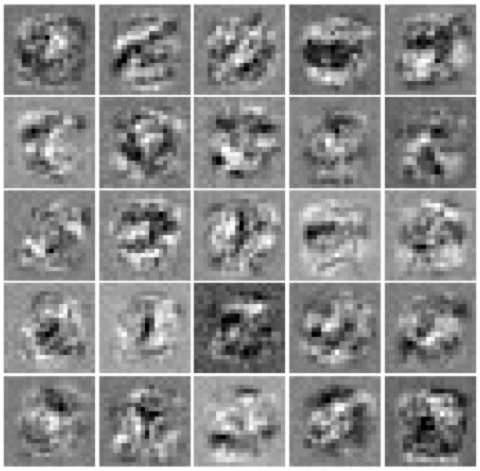

Рисунок 2 – визуализация скрытого слоя нейросети.

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

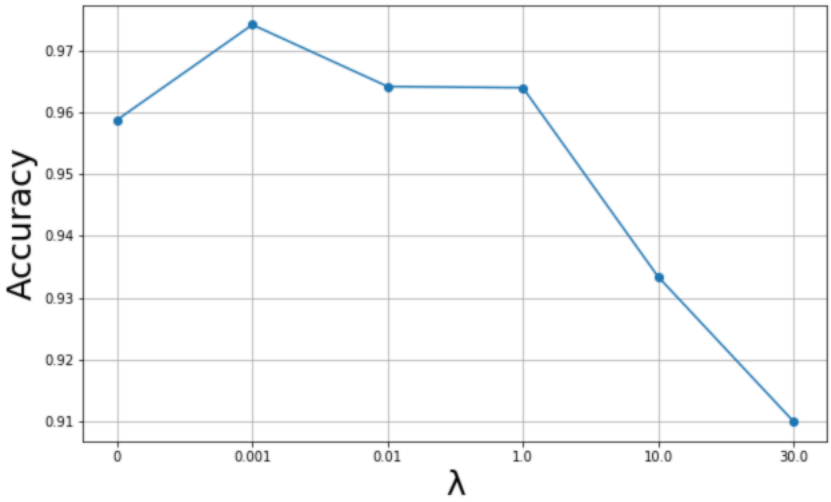

Рисунок 3 – зависимость точности от параметра lambda.

Про рисунку 3 видно, что наибольшая точность получается при параметре lambda = 0.01, отсутствие регуляризации, или использование других коэффициентов lambda понижает точность модели.

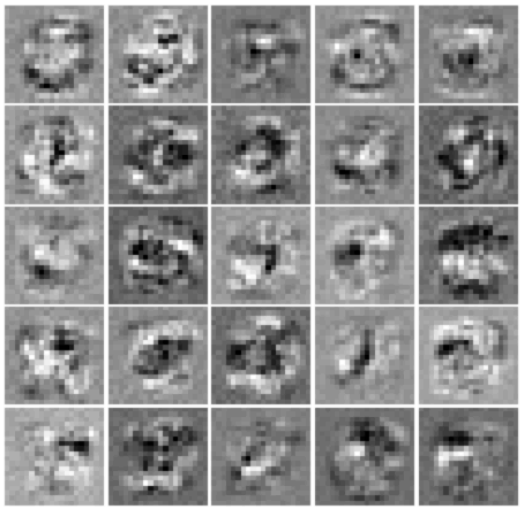

Рисунок 4 – визуализация скрытого слоя при lambda=0.001.

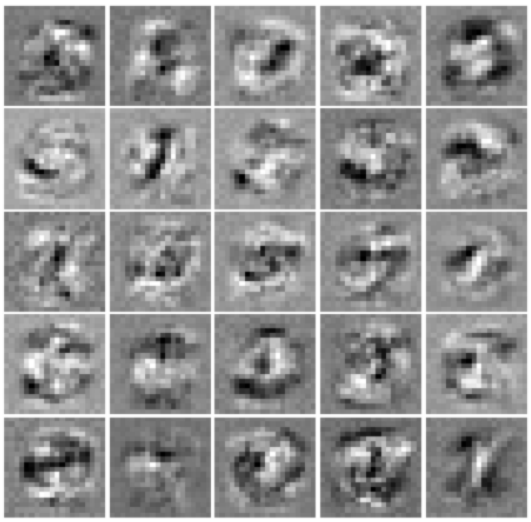

Рисунок 5 – визуализация скрытого слоя при lambda=1.

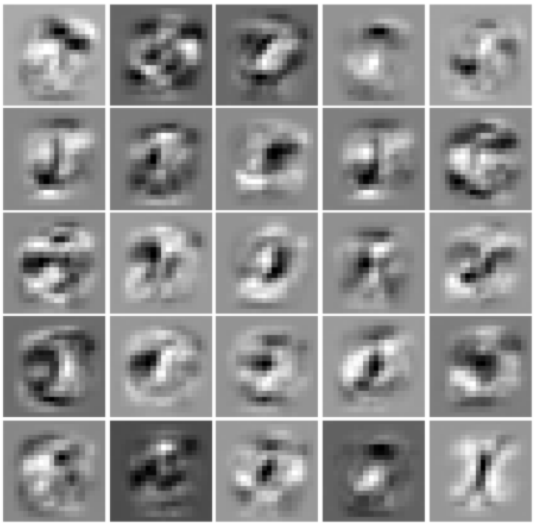

Рисунок 6 – визуализация скрытого слоя при lambda=30.

Если сравнить визуализацию внутреннего слоя при разных lambda, можно заметить, что наиболее четкое изображение получается при lambda = 0.001. Если увеличить коэффициент, то изображение начинает смазываться и размываться. Пример тому визуализация скрытого слоя при lambda=30.

Вывод.

В ходе выполнения лабораторной работы я изучил принципы работы нейронной сети, реализовал алгоритмы: “forward propagation”, “back propagation” и алгоритм проверки корректности градиента.

Применив для классификации рукописных цифр нейронную сеть вместо логистической регрессии удалость повысить параметр accuracy на 1,8% и теперь он составляет 99.0%.