Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ЛАБОРАТОРНАЯ РАБОТА №4

«Нейронные сети»

Студент А. Ю. Омельчук

Преподаватель М. В. Стержанов

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 $\varepsilon = 10^{-4}$.
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?

Результат выполнения:

1. Код выгрузки данных из файла представлен ниже:

```
    file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex4data1.mat')
    data = sio.loadmat(file_path)
    y = data.get('y')
    X = data.get('X')
```

2. Код выгрузки весов нейронной сети:

```
    file_path = os.path.join(os.path.dirname(__file__), 'data', '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)
```

У нас получается следующая структура нейронной сети. На входном слое 400 нейронов +1 bias нейрон, в скрытом нейронном слое 25 нейронов и 1 bias, и на выходном слое 10 нейронов, что равняется количеству наших классов - десять цифр от нуля до 9.

3. Код сигмоида и функции прямого распространения:

```
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. Код вычисления:

```
        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. Код реализации:

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

В данном коде каждой цифре присваевается класс, точнее цифра преобразуется в вектор, размера 10x1, который даёт метку классу, т. е. цифра 3 будет перекодирована в следующий вектор [0 0 1 0 0 0 0 0 0 0] – и так для каждой цифры.

6-7. Код реализации:

```
    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. Код реализации:

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

9. Код реализации:

```
        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. Код реализации:

```
    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. Код реализации:

```
    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. Код реализации:

```
    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. Код реализации:

```
        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. Код реализации:

```
        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. График представлен ниже:

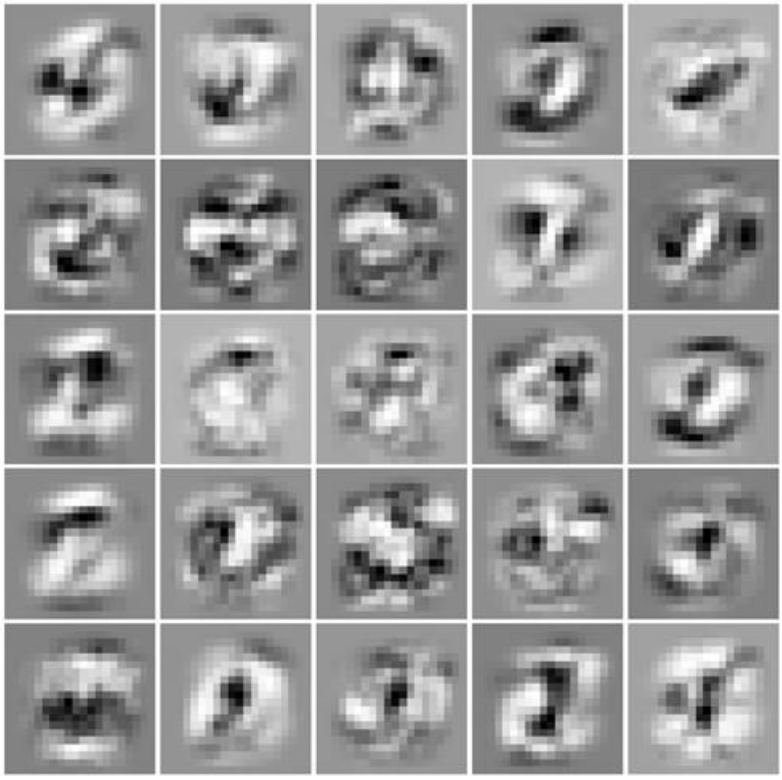

Рисунок 1 – визуализация скрытого слоя при lambda=1

17. График представлен ниже:

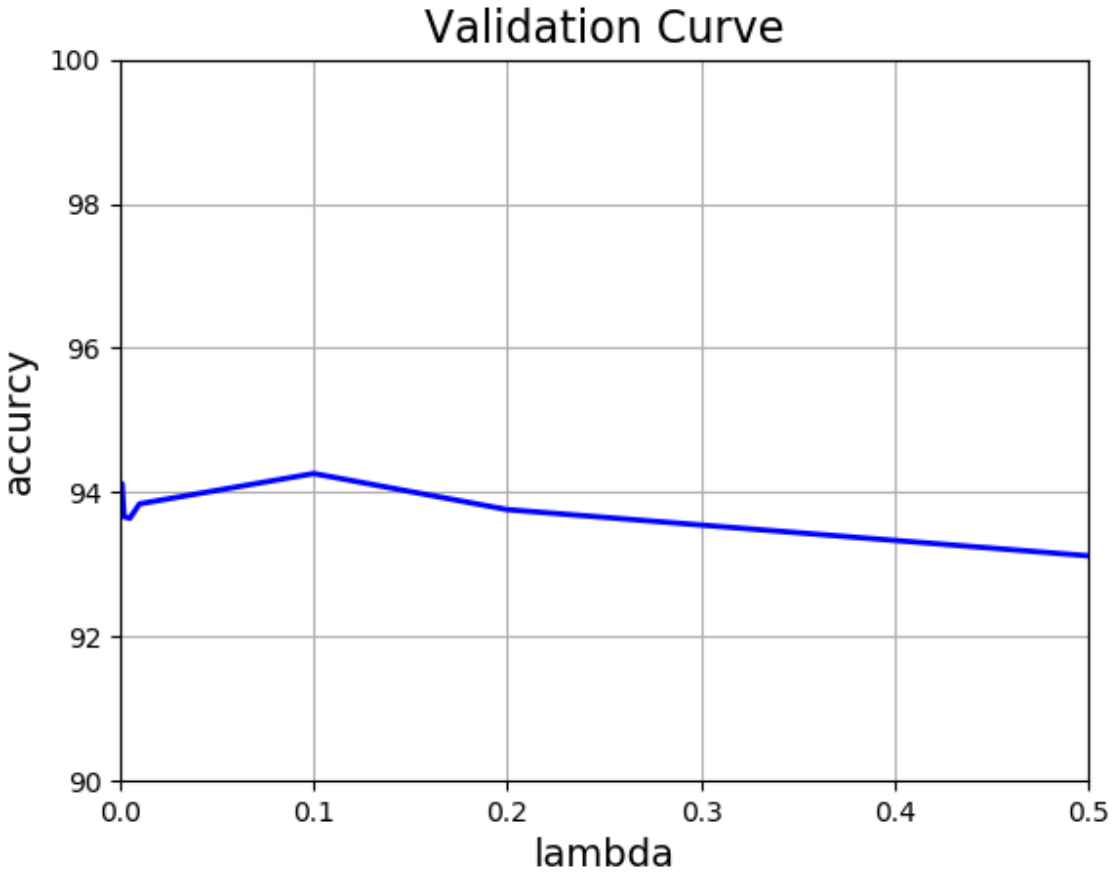

Рисунок 2 – точность предсказаний при различных lambda (на рисунке видно, что наибольшая точность достигается при lambda=0.1)

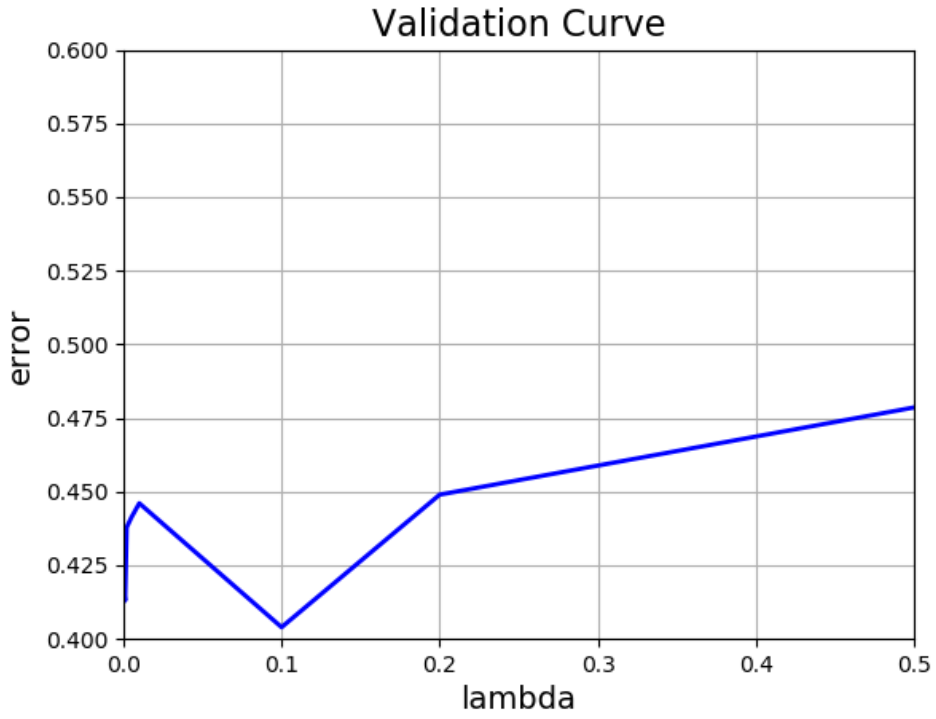

Рисунок 3 – зависимость ошибки от lambda (на рисунке видно, что наименьшая ошибка достигается при lambda=0.1)

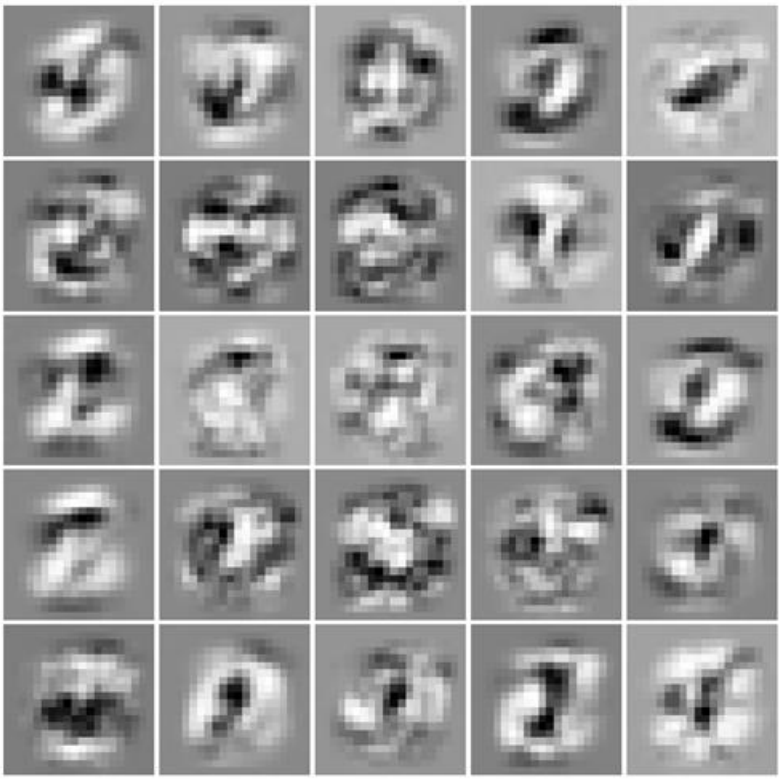

Рисунок 4 – визуализация скрытого слоя при lambda=0.1 (в сравнении с 1 здесь получились изображения с большей чёткостью)

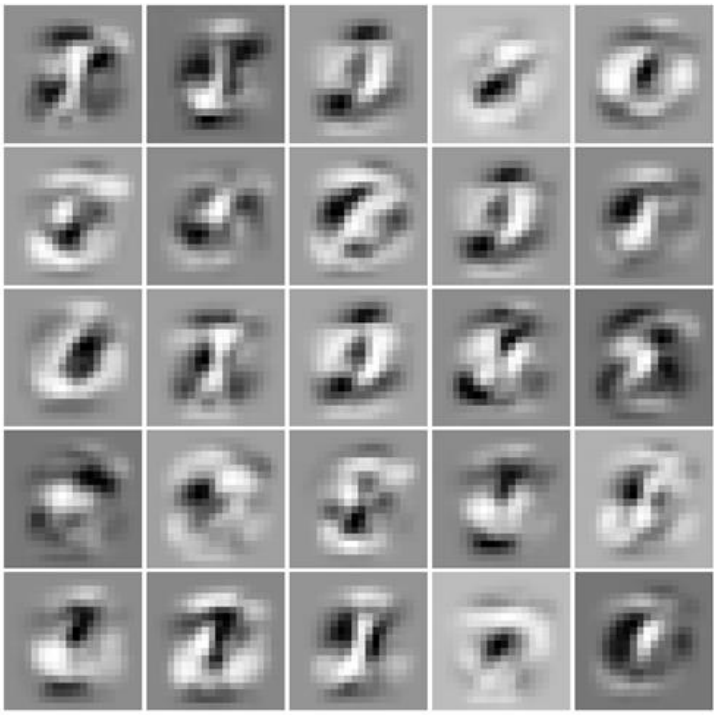

Рисунок 5 – визуализация скрытого слоя при lambda=100 (очень смазанные изображения и непонятно что это за цифры и цифры ли это вообще)

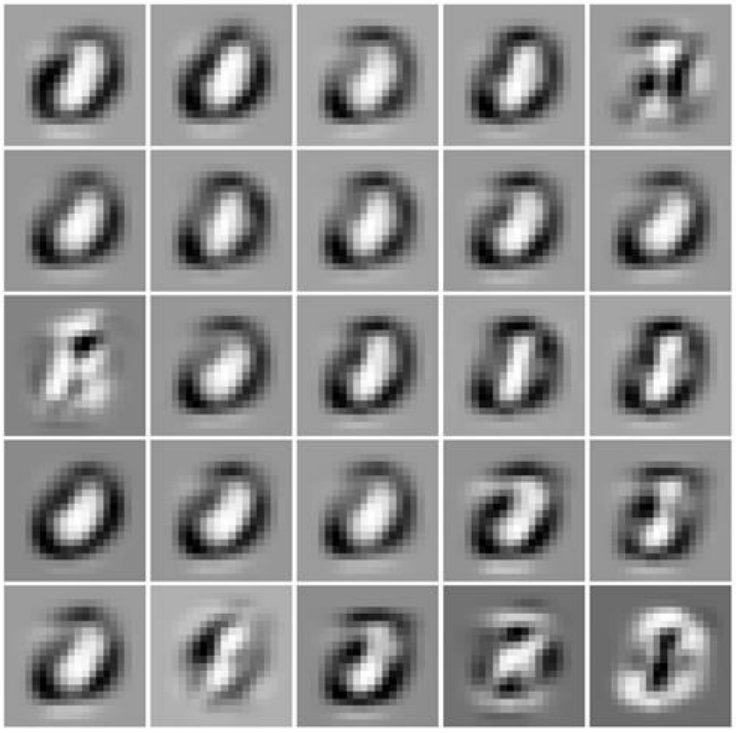

Рисунок 6 – визуализация скрытого слоя при lambda=1000 (почти все изображения стали похожими на картинку с нулём)

Программный код:

```
from __future__ import division
import scipy.io as sio
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from scipy import optimize as opt


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


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)


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


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

```
    def visualize(X, example_width=None, figsize=(10, 10)):
        """
        Displays 2D data stored in X in a nice grid.
        """
        # Compute rows, cols
        if X.ndim == 2:
            m, n = X.shape
        elif X.ndim == 1:
            n = X.size
            m = 1
            X = X[None]  # Promote to a 2 dimensional array
        else:
            raise IndexError('Input X should be 1 or 2 dimensional.')

        example_width = example_width or int(np.round(np.sqrt(n)))
        example_height = n / example_width

        # Compute number of items to display
        display_rows = int(np.floor(np.sqrt(m)))
        display_cols = int(np.ceil(m / display_rows))

        fig, ax_array = plt.subplots(display_rows, display_cols, figsize=figsize)
        fig.subplots_adjust(wspace=0.025, hspace=0.025)

        ax_array = [ax_array] if m == 1 else ax_array.ravel()

        for i, ax in enumerate(ax_array):
            # Display Image
            h = ax.imshow(X[i].reshape(example_width, example_width, order='F'),
                          cmap='Greys', extent=[0, 1, 0, 1])
            ax.axis('off')

        plt.show()


    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')))


    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]))


    if __name__ == '__main__':
        # 1
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex4data1.mat')
        data = sio.loadmat(file_path)
        y = data.get('y')
        X = data.get('X')

        # 2
        file_path = os.path.join(os.path.dirname(__file__), 'data', '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)

        # 3
        print("Sigmoid with x=2:", sigmoid(2), "Approximately: 0.88")
        pred = h0x(X, [theta1, theta2])

        # 4
        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)
        # 95.12 (log) vs 97.52 (nn)

        # 5
        #        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
        y_one_hot = pd.get_dummies(y.flatten())

        # 6
        cost = cost_func(nn_params, hidden_layer_size, X, y_one_hot, 0)
        print("Cost:", cost, "Approximately: 0.287629")

        # 7
        cost = cost_func(nn_params, hidden_layer_size, X, y_one_hot, lmbda)
        print("Cost:", cost, "Approximately: 0.383770")

        # 8
        print("::Sigmoid derivative:: ", "Approximately: sg(2) = 0.1", sigmoid_derivative(2))

        # 9
        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
        backprop_params = back_propagation(initial_nn_params, hidden_layer_size, X, y_one_hot,
0)

        # 11
        gradient_check(initial_nn_params, backprop_params, hidden_layer_size, X, y_one_hot, 0)

        # 12-13
        print("L2 regularization: with lambda = 1")
        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)

        # 14
        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
        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)

        # 16
        visualize(theta1_opt[:, 1:])

        # 17
        print("Optimal theta searching...")
        # TODO: implement the rest part
        lambda_values = [0, 0.0001, 0.001, 0.002, 0.005, 0.01, 0.1, 0.2, 0.5]
        val_err = []
        accur = []
        for lamb in lambda_values:
            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, lamb))
            val_err.append(cost_func(theta_opt, hidden_layer_size, X, y_one_hot, 0))
            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')

            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
            accur.append(predictions / len(y) * 100)
        plt.plot(lambda_values, val_err, c="b", linewidth=2)
        plt.axis([0, 0.5, 0.4, 0.6])
        plt.grid()
        plt.xlabel("lambda", fontsize=14)
        plt.ylabel("error", fontsize=14)
        plt.title("Validation Curve", fontsize=16)
        plt.show()

        plt.plot(lambda_values, accur, c="b", linewidth=2)
        plt.axis([0, 0.5, 90, 100])
        plt.grid()
        plt.xlabel("lambda", fontsize=14)
        plt.ylabel("accurcy", fontsize=14)
        plt.title("Validation Curve", fontsize=16)
        plt.show()

        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, 0.1))
        theta1_opt       =       np.reshape(theta_opt[:hidden_layer_size*(input_layer_size+1)],
(hidden_layer_size, input_layer_size+1), 'F')
        visualize(theta1_opt[:, 1:])

        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, 100))
        theta1_opt       =       np.reshape(theta_opt[:hidden_layer_size*(input_layer_size+1)],
(hidden_layer_size, input_layer_size+1), 'F')
        visualize(theta1_opt[:, 1:])
```