1. Загрузите данные **ex4data1.mat** из файла.

```
load('ex4data1.mat');
```

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?

```
load('ex4weights.mat');
```

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

```
function p = predict(Theta1, Theta2, X)

m = size(X, 1);
num_labels = size(Theta2, 1);

p = zeros(size(X, 1), 1);

X = [ones(m, 1) X];
t1 = sigmoid(X * Theta1');
t1 = [ones(m, 1) t1];

t2 = sigmoid( t1 * Theta2');

[~, p] = max(t2, [], 2);

end
```

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

```
pred = predict(Theta1, Theta2, X);
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

```
Training Set Accuracy: 97.520000
```

```
difference = mean(double(pred == y)) * 100 - 94.78
```

```
difference = 2.7400
```

Результат лучше на 2.74%

6. Реализуйте функцию стоимости для данной нейронной сети.

7. Добавьте L2-регуляризацию в функцию стоимости.

```
function [J grad] = nnCostFunction(nn_params, ...
input_layer_size, ...
hidden_layer_size, ...
num_labels, ...
X, y, lambda)

Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
```

```
hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
num_labels, (hidden_layer_size + 1));

m = size(X, 1);
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));


X = [ones(m,1) X];
z2 = Theta1 * X';
a2 = sigmoid(z2);

a2 = [ones(m,1) a2'];
z3 = Theta2 * a2';
h_theta = sigmoid(z3);

y_new = zeros(num_labels, m);
for i=1:m,
y_new(y(i),i)=1;
end

J = (1/m) * sum ( sum ( (-y_new) .* log(h_theta) - (1-y_new) .* log(1-h_theta) ));

t1 = Theta1(:,2:size(Theta1,2));
t2 = Theta2(:,2:size(Theta2,2));

Reg = lambda  * (sum( sum ( t1.^ 2 )) + sum( sum ( t2.^ 2 ))) / (2*m);

J = J + Reg;


for t=1:m
a1 = X(t,:);
a1 = a1';
z2 = Theta1 * a1;
a2 = sigmoid(z2);
a2 = [1 ; a2];
z3 = Theta2 * a2;
a3 = sigmoid(z3);
delta_3 = a3 - y_new(:,t);
z2=[1; z2];
delta_2 = (Theta2' * delta_3) .* sigmoidGradient(z2);
delta_2 = delta_2(2:end);

Theta2_grad = Theta2_grad + delta_3 * a2';
Theta1_grad = Theta1_grad + delta_2 * a1';
end;

Theta2_grad = (1/m) * Theta2_grad;
Theta1_grad = (1/m) * Theta1_grad;


Theta1_grad(:, 2:end) = Theta1_grad(:, 2:end) + ((lambda/m) * Theta1(:, 2:end)); % for j >= 1
Theta2_grad(:, 2:end) = Theta2_grad(:, 2:end) + ((lambda/m) * Theta2(:, 2:end)); % for j >= 1
```

```
grad = [Theta1_grad(:) ; Theta2_grad(:)];

end
```

8. Реализуйте функцию вычисления производной для функции активации.

```
function g = sigmoidGradient(z)

g = zeros(size(z));
g = sigmoid(z) .* (1 - sigmoid(z));

end
```

9. Инициализируйте веса небольшими случайными числами.

```
initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);
```

```
initial_Theta2 = 10×26
   -0.0648    0.0942    0.0642   -0.0031   -0.0347    0.1164    0.0471    0.0826 ···
    0.1190   -0.0022    0.0795    0.0875   -0.1112   -0.0429    0.0051    0.0871
   -0.1054    0.1127    0.0766   -0.0693    0.0856    0.1166   -0.0595    0.1093
   -0.0351    0.0509   -0.0543    0.0510   -0.0384   -0.0752   -0.0356    0.0902
   -0.0522    0.0916    0.0736   -0.0375    0.0572   -0.0480   -0.0768   -0.0231
   -0.0156    0.0209   -0.0135   -0.0829   -0.0102    0.1132    0.0449   -0.0649
   -0.0357   -0.0202   -0.0184    0.0481   -0.0844    0.0943   -0.0748   -0.0067
    0.0783    0.0319    0.0570    0.1018    0.0665    0.0391    0.0886   -0.0119
    0.0005   -0.0184    0.1131    0.0668   -0.0442    0.0369   -0.0348   -0.0050
   -0.0374   -0.0480    0.0509    0.0960   -0.0938   -0.0350    0.0543   -0.0775
```

```
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];
```

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.

```
lambda = 3;
checkNNGradients(lambda);
```

```
   -0.0093   -0.0093
    0.0089    0.0089
   -0.0084   -0.0084
    0.0076    0.0076
   -0.0067   -0.0067
   -0.0168   -0.0168
    0.0394    0.0394
    0.0593    0.0593
    0.0248    0.0248
   -0.0327   -0.0327
   -0.0602   -0.0602
   -0.0320   -0.0320
```

```
    0.0249    0.0249
    0.0598    0.0598
    0.0386    0.0386
   -0.0174   -0.0174
   -0.0576   -0.0576
   -0.0452   -0.0452
    0.0091    0.0091
    0.0546    0.0546
    0.3145    0.3145
    0.1111    0.1111
    0.0974    0.0974
    0.1187    0.1187
    0.0000    0.0000
    0.0337    0.0337
    0.2040    0.2040
    0.1171    0.1171
    0.0755    0.0755
    0.1257    0.1257
   -0.0041   -0.0041
    0.0170    0.0170
    0.1763    0.1763
    0.1131    0.1131
    0.0862    0.0862
    0.1323    0.1323
   -0.0045   -0.0045
    0.0015    0.0015

The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your backpropagation implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 2.31407e-11
```

```
debug_J  = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, num_labels, X
fprintf('Cost at (fixed) debugging parameters (w/ lambda = 3): %f', debug_J);
```

```
Cost at (fixed) debugging parameters (w/ lambda = 3): 0.576051
```

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

15. Вычислите процент правильных классификаций на обучающей выборке.

```
options = optimset('MaxIter', 50);
lambda = 1;
costFunction = @(p) nnCostFunction(p, input_layer_size, hidden_layer_size, num_labels,
[nn_params, ~] = fmincg(costFunction, initial_nn_params, options);
```

```
Iteration     1 | Cost: 3.309932e+00
Iteration     2 | Cost: 3.242441e+00
Iteration     3 | Cost: 3.192405e+00
Iteration     4 | Cost: 2.798920e+00
Iteration     5 | Cost: 2.487892e+00
Iteration     6 | Cost: 2.399977e+00
Iteration     7 | Cost: 2.198364e+00
Iteration     8 | Cost: 1.839534e+00
Iteration     9 | Cost: 1.703839e+00
Iteration    10 | Cost: 1.645288e+00
Iteration    11 | Cost: 1.458893e+00
```

```
Iteration    12 | Cost: 1.390983e+00
Iteration    13 | Cost: 1.206240e+00
Iteration    14 | Cost: 1.097810e+00
Iteration    15 | Cost: 1.064724e+00
Iteration    16 | Cost: 9.644210e-01
Iteration    17 | Cost: 9.224633e-01
Iteration    18 | Cost: 8.689206e-01
Iteration    19 | Cost: 8.369293e-01
Iteration    20 | Cost: 8.131476e-01
Iteration    21 | Cost: 7.794444e-01
Iteration    22 | Cost: 7.654357e-01
Iteration    23 | Cost: 7.576890e-01
Iteration    24 | Cost: 7.486774e-01
Iteration    25 | Cost: 7.286269e-01
Iteration    26 | Cost: 7.187079e-01
Iteration    27 | Cost: 7.110450e-01
Iteration    28 | Cost: 6.817537e-01
Iteration    29 | Cost: 6.764549e-01
Iteration    30 | Cost: 6.714192e-01
Iteration    31 | Cost: 6.556075e-01
Iteration    32 | Cost: 6.388721e-01
Iteration    33 | Cost: 5.913383e-01
Iteration    34 | Cost: 5.636870e-01
Iteration    35 | Cost: 5.515274e-01
Iteration    36 | Cost: 5.407204e-01
Iteration    37 | Cost: 5.334392e-01
Iteration    38 | Cost: 5.263517e-01
Iteration    39 | Cost: 5.161682e-01
Iteration    40 | Cost: 5.110124e-01
Iteration    41 | Cost: 5.093536e-01
Iteration    42 | Cost: 5.035048e-01
Iteration    43 | Cost: 5.010456e-01
Iteration    44 | Cost: 4.987265e-01
Iteration    45 | Cost: 4.896873e-01
Iteration    46 | Cost: 4.844379e-01
Iteration    47 | Cost: 4.816494e-01
Iteration    48 | Cost: 4.739312e-01
Iteration    49 | Cost: 4.689900e-01
Iteration    50 | Cost: 4.673365e-01
```

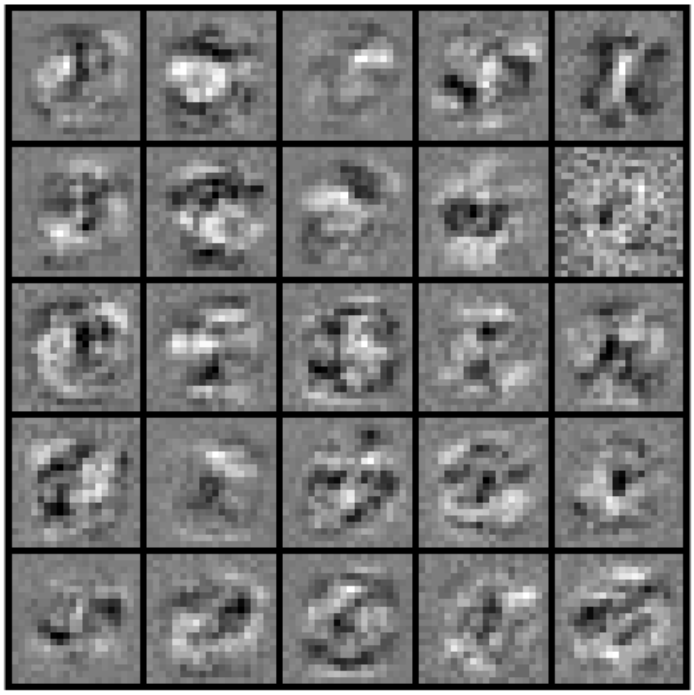
```
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), hidden_layer_
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), num

pred = predict(Theta1, Theta2, X);
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

```
Training Set Accuracy: 95.480000
```

16. Визуализируйте скрытый слой обученной сети.

```
displayData(Theta1(:, 2:end));
```



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

```
lambda = 0.3;
MaxIter = 50;
options = optimset('MaxIter', MaxIter);
costFunction = @(p) nnCostFunction(p,input_layer_size, hidden_layer_size, num_labels, X
[nn_params, ~] = fmincg(costFunction, initial_nn_params, options);
```

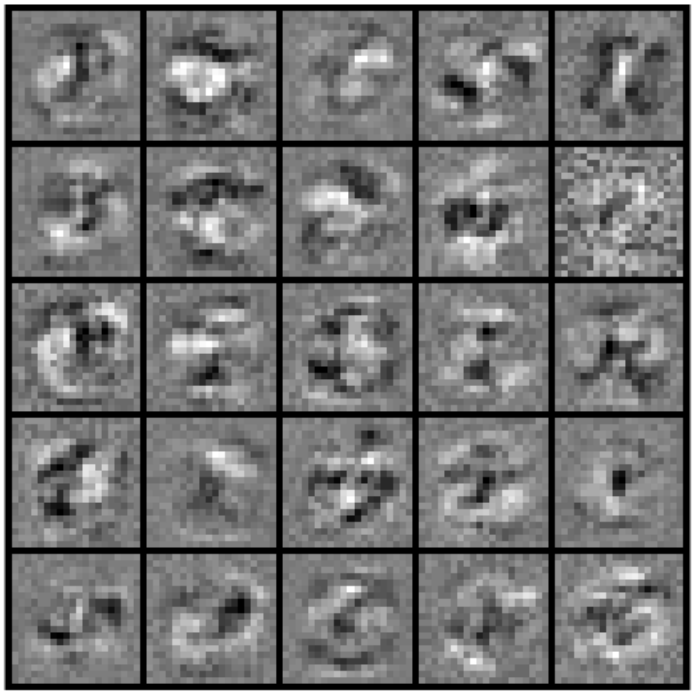
```
Iteration
```

```
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), hidden_layer_
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), num

pred = predict(Theta1, Theta2, X);
```

```
Training Set Accuracy: 94.720000
```

```
displayData(Theta1(:, 2:end));
```



```
lambda = 3;
MaxIter = 50;
options = optimset('MaxIter', MaxIter);
costFunction = @(p) nnCostFunction(p,input_layer_size, hidden_layer_size, num_labels, X
[nn_params, ~] = fmincg(costFunction, initial_nn_params, options);

Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), hidden_layer_
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), num

pred = predict(Theta1, Theta2, X);
displayData(Theta1(:, 2:end));
lambda = 30;
MaxIter = 50;
options = optimset('MaxIter', MaxIter);
costFunction = @(p) nnCostFunction(p,input_layer_size, hidden_layer_size, num_labels, X
[nn_params, ~] = fmincg(costFunction, initial_nn_params, options);

Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), hidden_layer_
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), num

pred = predict(Theta1, Theta2, X);
displayData(Theta1(:, 2:end));
```