

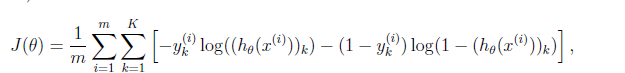
In the previous exercise, you implemented **feedforward** propagation for neu-

ral networks and used it to predict handwritten digits with the weights we

provided. In this exercise, you will implement the **backpropagation** algorithm

to learn the parameters for the neural network.

1. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image.
2. Since the images are of size 20 x 20, this gives us 400 input layer units
3. You have been provided with a set of network parameters Theta (1); Theta (2)) already trained by us. These are stored in ex4weights.mat and will be loaded by ex4.m into Theta1 and Theta2.
4. the cost function –



1. 

**Feedforward and cost function(Try and Test Already done in ex3)**

1. Now you will implement the cost function and gradient for the neural network. First, complete the code in nnCostFunction.m to return the cost.

* (unregularized) neural network.
* gradient for the regularized neural network.

**Backpropagation(Actual use this )**

1. Implement the backpropagation algorithm to compute the gradient for the neural network cost function. You will need to complete the nnCostFunction.m so that it returns an appropriate value for grad.

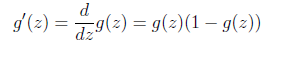
* backpropagation algorithm to compute the gradients for the parameters for the (unregularized) neural network.
* gradient for the regularized neural network.

1. Once you have computed the gradient, you will be able to train the neural network by minimizing the cost function J(Theta) using an advanced optimizer such as fmincg.

**Sigmoid gradient.m(Use to calculate the delta in activation func) and sigmoid.m**



When z = 0, the gradient should be exactly 0.25.



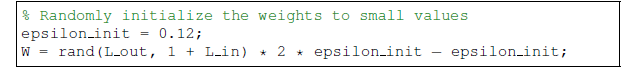
Formula Convert it into Octave Code

**Random initialization**

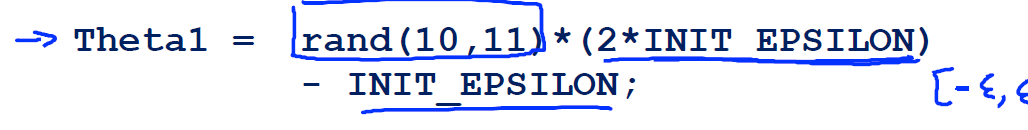
When training neural networks, it is important to randomly initialize the parameters for symmetry breaking. One effective strategy for random initialization is to randomly select values for (l) uniformly in the range 

You should use –e init = 0:12.2 This range of values ensures

Fornula



where Lin = sl and Lout = sl+1 are the number of units in the layers adjacent to (l).

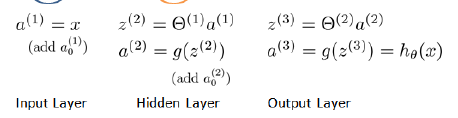


**nnCostFunction.m (function [J grad] To calculate Motto)**

**Part 1(Refer from ex3 pdf: To Calculate Cost Function Theta1 and Theta2 and Minimize Cost Function using Gradient \_ Descent Theta\_grad1 , Theta\_grad2 ) :**

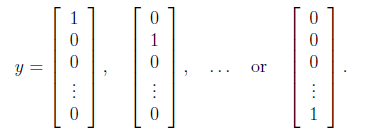
* Given a training example (x(t); y(t)), we will first run a “**forward pass"** to compute all the activations throughout the network, including the output value of the hypothesis h(x) and Cost Function .

Step 1 **:** Convert this representation into Octave Formula



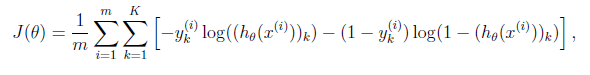
Step 2:

Covert this into Octave Formula



Step 3:

Convert this into Octave Formula



**Part 2: Implement the back propagation algorithm to compute the gradients Theta1\_grad and Theta2\_grad**

**Back propagation**

* Then, for each node j in layer l, we would like to compute an “**error term"** that measures how much that node was “responsible" for any errors in our output.

Concretely, you should implement a for-loop for t = 1:m and place steps 1-4 below insidethe for-loop, with the t th iteration performing the calculation on the t th training example

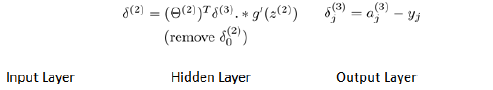
(x(t); y(t)).

Step 1 :

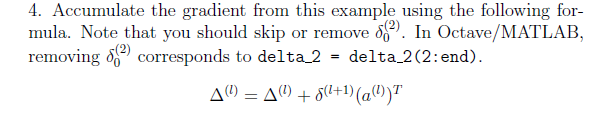
Set the input layer's values (a(1)) to the t-th training example x(t). Perform a feedforward pass (Figure 2), computing the activations (z(2); a(2); z(3); a(3)) for layers 2 and 3. Note that you need to add a +1 term to ensure that the vectors of activations for layers a(1) and a(2) also include the bias unit.

Convert this into logic

Step 2 and Step 3: Convert this into logic

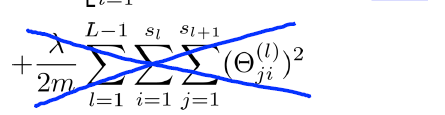


Step 4 : Convert this into logic

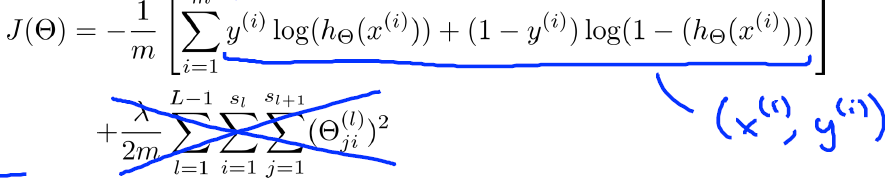


**Part 3: Adding Regularisation term in J and Theta\_grad**

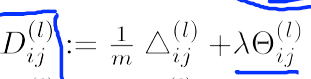
Step 1:



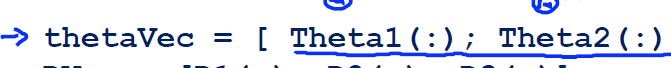
Step 2(J(Theta) was already calculated using Feed Forward Propagation ) :



Step 3 (Delata was calculated using BackProgation ):



Step4 :



O1N2d2s3cGFTeq7a