## Cost Function:

For a neural Network:

$L$ = total number of layers in the network

$S_l$ = number of units in layer $l$ (input) (Not counting bias Unit)

$K$ = Number of output units/classes

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{m}\sum_{j=1}^{S_{l+1}}\left(\Theta_{j,i}^{(l)}\right)^2$$

## Backpropagation Algorithm:

Given Training set = $\{(x^1, y^1), \ldots (x^m, y^m)\}$

Set, $\Delta_{i,j}^{(l)} = 0$ → Error Matrix

For training example $t = 1$ to $m$:

layer number

- → set $a^{(1)} := x^{(t)}$
- → Perform Forward propagation $a^l$ for $l = 1, 2, \ldots L$
- → Using $y^t$, compute $\delta^{(L)} = a^L - y^t$

→ Compute $\delta^{L-1}, \ldots \delta^{2}$ using,

$$\delta^{(l)} = \left( (\Theta^{(l)})^T \delta^{(l+1)} \right) .* \underbrace{a^{(l)} .* (1 - a^{(l)})}_{g'(z^l)}$$

$$\rightarrow \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

$$D_{ij}^{(l)} := \frac{1}{m} \left( \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \right) \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

## Unroll:

thetavec = [Theta1(:); Theta2(:); Theta(:);]

→ Theta1 = reshape(thetavec(1:110), 10, 11)

→ Theta2 = reshape(thetavec(111:220), 10, 11)

→ Theta3 = reshape(thetavec(221:231), 1, 11)

## Gradient Checking (Numerically):

```
epsilon = 1e-4;

for i=1 : n,
   thetaplus = theta;
   thetaplus(i)+= epsilon;
   thetaminus = theta;
   thetaminus(i)-= epsilon;
   gradApprox(i)= (J(thetaplus)-J (thetaminus))/
                        (2* epsilon);
end;
```

check gradApprox ≈ DeltaVector

## Random Initialization:

Initializing all thetas weight to ZERO does not work with neural network. Because in backpropagation, all nodes will be updated with same value repeatedly.

To breake symmetry:

Theta1 = rand (10, 11) * (2 * EPS) - EPS;
Theta2 = rand (1, 11) * (2 * EPS) - EPS;

Big Picture of Neural Network:

Pick a network architecture.

→ # Input Units = dimension of features $x^{(i)}$

→ # Output Units = number of classes

→ # hidden units per layer = the more the better

→ Defalut 1 hidden layer, but again the more the better. [Some # of hidden units in all layer]

## Training a Neural Network:

→ Randomly Initialized the weights ($\Theta$)
→ Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$

→ Implement the cost Function

→ Implement back propagation to compute partial derivative

→ Use gradient checking to confirm that back propagation works, then disable it.

→ Use gradient descent or a built-in optimization function to minimize the cost function with weights in theta.