WEEK-5

⇒ L = total no of layers in network

⇒ K = no of output units

⇒ S_L = no of units in layer L (not counting bias unit)

⇒ **COST FUNCTION** → For Neural Networks

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-h_\Theta(x^{(i)})\right)_k\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

→ we dont sum over bias unit.

⇒ The double sum simply adds the logistic regression costs calculated for each cell in the output layer.

⇒ The triple sum adds up the squares of all the individual Θs in the entire network.

⇒ $\delta_j^{(l)}$ = "error" of node j in layer l

↳ $\delta_j^4 = a_j^4 - y_j$ → Back Propagation (way of finding $\frac{\partial}{\partial\Theta} J(\Theta)$)

$\Rightarrow \quad \delta^3 = (\theta^{(3)})^T \delta^4 .* g'(z^{(3)})$

↳ $a^{(3)} .* (1 - a^{(3)})$

↳ derivative of $g(z^3)$

⇒ So, using back propagation, we can compute gradient and find value of θ.

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)} \quad \left(\text{ignoring } \lambda, \text{ if } \lambda = 0\right)$$

⇒ Back propagation Algorithm (BP)

Step 1⇒ Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

S-2⇒ set $\Delta_{ij}^{(l)} = 0$, for all l, i, j (done for every $x^{(i)}, y^{(i)}$)

S-3⇒ For $i = 1$ to m :-

set $a^{(1)} = x^{(i)}$

→ Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots L$ (last layer)

→ using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

→ using BP, compute $\delta^{(L-1)}, \delta^{(L-2)} \ldots \delta^2$ ($\delta^{(1)}$ → is not calculated)

→ $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

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

S-4 ⇒ $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \theta_{ij}^{(l)}$ if $j \neq 0$

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

$\therefore \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$

Vectorized ⇒ theta_grad = (1/m) * del2 * a1;

⇒ Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$ (for $j \geq 0$)

where $cost(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$

⇒ $\delta_2^{(2)} = \theta_{12}^{(2)} \delta_1^{(3)} + \theta_{22}^{(2)} \delta_2^{(3)}$

$\delta_2^{(3)} = \theta_{12}^{(3)} \delta_1^{(4)}$

⇒ for example, if $s_1 = 10$, $s_2 = 10$, $s_3 = 1$

⇒ $\Theta^1 \Rightarrow \mathbb{R}^{10 \times 11}$, $\Theta^2 \Rightarrow \mathbb{R}^{10 \times 11}$, $\Theta^{(3)} = 1 \times 11$

⇒ $D^{(1)} \Rightarrow \mathbb{R}^{10 \times 11}$, $D^{(2)} \Rightarrow \mathbb{R}^{10 \times 11}$, $D^{(3)} = 1 \times 11$

⇒ thetaVec = [theta1(:); theta2(:); theta3(:)];

converts these vectors in one long vector

⇒ To redo it & get thetaVec → Also called Gradient Checking

thetaVec = reshape(thetaVec(1:110), 10, 11);

⇒ to use these in optimization f^n's like fminunc() → we unroll them into one big long vector.

⇒ gradApprox = $\frac{(J(theta + epsilon) - J(theta - epsilon))}{2 \times epsilon}$

Way to check BP

⇒ if $\theta = [\theta_1, \theta_2, \theta_3, \ldots \theta_n]$

⇒ $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \ldots \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3 \ldots \theta_n)}{2\epsilon}$

$\epsilon = 10^{-4}$

⇒ $\frac{\partial}{\partial \theta_2} J(\theta)$ for every θ.

⇒ ——————→ (Gradient Checking Code)

⇒ for i = 1:n,
thetaPlus = theta;
thetaPlus(i) = thetaPlus(i) + epsilon
thetaMinus = theta;
thetaMinus(i) = thetaMinus(i) - epsilon;
gradApprox(i) = $\frac{(J(thetaPlus) - J(thetaMinus))}{(2 \times epsilon)}$
end;

⇒ if (gradApprox ≈ DVec) → then BP is rightly used

⇒ Gradient checking is very slow as compared to BP. (so only 1 iteration)

⇒ If $\theta_{ij}^{(l)} = 0$ for all i, j, l

$\therefore a_1^{(2)} = a_2^{(2)}$, $\delta_1^{(2)} = \delta_2^{(3)}$

$$\frac{\partial}{\partial \theta_{01}^{(1)}} J(\theta) = \frac{\partial}{\partial \theta_{02}^{(1)}} J(\theta) \quad , \quad \theta_{01}^{(1)} = \theta_{02}^{(1)}$$

⇒ So we can't initialize the theta (parameters) with 0 (as in logistic regression), therefore we use random values

(Symmetry Breaking)

$\therefore$ Initialize each $\theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$

Code

To create a random 10×11 matrix (values b/w 0 & 1)

```
theta1 = rand(10,11) * (2 * init_epsilon)
         - init_epsilon)

theta2 = rand(1,11) * (2 * init_epsilon)
         - init_epsilon
```

to keep the value in $[-\epsilon, \epsilon]$ range

(this epsilon is unrelated to epsilon in gradient checking)

⇒ No. of input units ⇒ Dimension of features ($x^{(i)}$)

⇒ No of output units ⇒ Number of classes.

⇒ For hidden layers ⇒ 1 or if >1 hidden layers, should have same no. of hidden units or nodes in every layer.

⇒ TRAINING A Neural Network

1) = Randomly initialize weights

2) = Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$

3) = Implement code to compute cost f^n ⇒ $J(\theta)$

4) = Implement backdrop to compute partial derivatives → $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta)$.

for $i = 1 : m$
→ Perform forward P & BP using example $(x^{(i)}, y^{(i)})$
→ Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots L$.

5) = Use gradient checking to compare $\frac{\partial}{\partial \Theta} J(\theta)$ computed using BP V.S. using numerical estimate of gradient of $J(\theta)$. Disable GC

6)= Use gradient descent or advanced optimization method with BP to try to minimize $J(\theta)$ as a fn of θ.

⇒ For NN, $J(\theta)$ is non-convex.