Cost Function:

L : total number of layers in the network.

S_l : number of units (not counting bias unit) in layer l.

K : number of output units / classes.

m : number of training samples.

$h_\theta(x)_k$: The hypothesis that results in the k^{th} output.

二分类问题 : Binary classification.

$h_\theta(x) \in \mathbb{R}$

$S_L = 1$ $\quad y = 0$ or 1 $\quad$ S_L : 输出层的 node 数量.

Multi-class classification (K classes).

$h_\theta(x) \in \mathbb{R}^K$

$S_L = K \quad (K \geq 3)$ $\quad$ output $= \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\\vdots\\0\end{bmatrix}$

We use the same cost function as logistic regression:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

For Neural Networks:

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

$\Theta_{ij}^{(l)} \in \mathbb{R}$ $\quad + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\Theta_{j,i}^{(l)})^2$ → 排除 θ_0 后，j 循环所有的行，i 循环所有的列，由 S_l 层的激活单元数所决定.

Notice : ① Theta matrix 的列数 = 当前网络层节点的数量 (包含 bias 单元).

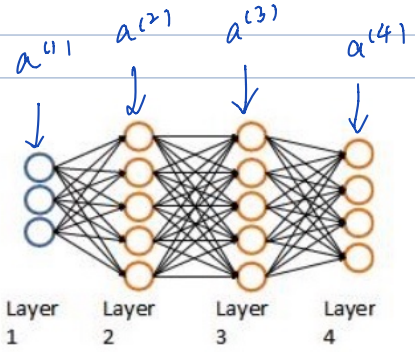
② Theta matrix 的行数 = 下一层网络的节点的数量 (不包含 bias 单元)

Backpropagation Algorithms :

$\min_\theta J(\theta)$

Need code to compute :

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta)$ ← gradients.



Given one training example (x, y):

Forward propagation:

$a^{(1)} = x$.

$z^{(2)} = \theta^{(1)}a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$) ← 先计激活函数，再添加偏置

$z^{(3)} = \theta^{(2)}a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \theta^{(3)}a^{(3)}$

$a^{(4)} = g(z^{(4)})$ (add $a_0^{(4)}$)

For each output unit (layer $L = 4$)

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

$\delta_j^{(4)} = a_j^{(4)} - y_j = (h_\theta(x))_j - y_j$ $\quad \underline{\delta}^{(4)} = \underline{a}^{(4)} - \underline{y}$

$\underline{\delta}^{(3)} = (\underline{\theta}^{(3)})^T \underline{\delta}^{(4)} .* g'(\underline{z}^{(3)})$, element-wise

$\underline{\delta}^{(2)} = (\underline{\theta}^{(2)})^T \underline{\delta}^{(3)} .* g'(\underline{z}^{(2)})$

$g'(z^{(2)}) = g(z^{(2)}) * (1 - g(z^{(2)}))$

BP 时，后一层的误差对前一层的影响.

因为第一层是输入层，不存在误差。当我们有了所有的误差表达式后，便可以计算 $J(\theta)$ 的偏导数了：

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

Note: l: 表示目前所计算的是第 l 层；
j: 目前计算层中激活单元的下标，也将是下一层的第 j 个输入变量的下标；
i: 下一层中误差单元的下标，是受到权重矩阵 (θ) 中第 i 行影响的下一层中的误差单元的下标。

Training set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$. ← m个训练样本.

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j) as an accumulator to compute $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$.

$\Delta_{ij}^{(l)}$ 表示误差矩阵，表示第 l 层的第 i 个 unit 受到第 j 个参数影响的误差。

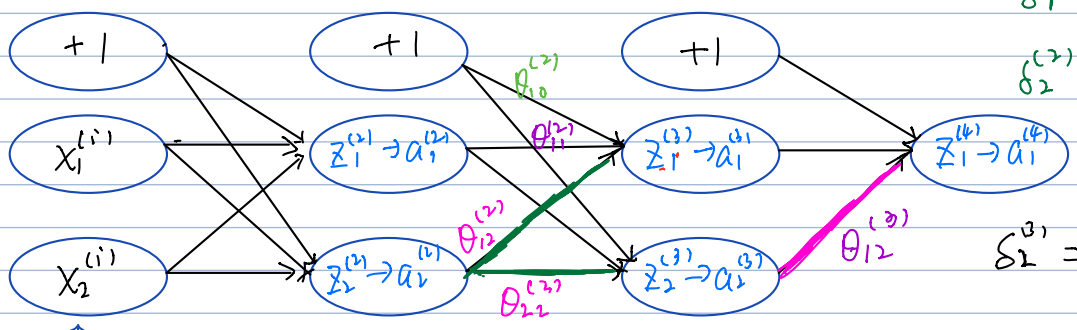
↑第 l 层的误差矩阵.

For $i = 1$ to m:
Set $a^{(1)} = x^{(i)}$
Forward propagation: $a^{(l)}$, $l = 2, 3, \cdots, L$.
Using $y^{(i)}$ compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$
$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ($\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$)

$D_{ij}^{(l)} := \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)})$ if $j \neq 0$ ← multi-class

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

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



$(x^{(i)}, y^{(i)})$

$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$

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

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

$z_1^{(3)} = \theta_{10}^{(2)} \times 1 + \theta_{11}^{(2)} \cdot a_1^{(2)} + \theta_{12}^{(2)} \cdot a_2^{(2)}$

↳未过激活函数.

$$J(\theta) = -\frac{1}{m}\left[\sum y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{S_l} \sum_{j=1}^{S_{l+1}} (\theta_{ij}^{(l)})^2$$

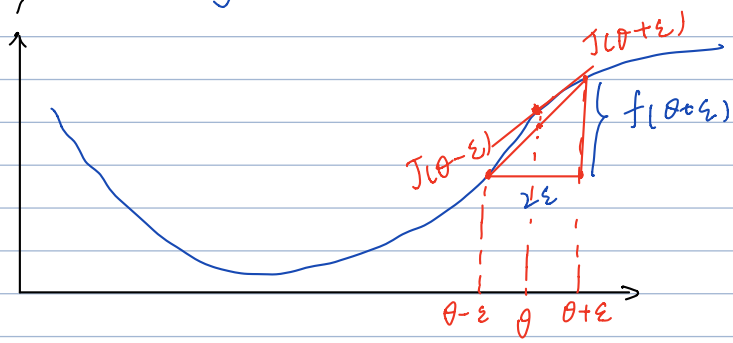
we focus on single example $(x^{(i)}, y^{(i)})$, $\lambda = 0$, 1 output unit

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

Think of $\text{cost}(i) \approx (h_\theta(x^{(i)}) - y^{(i)})^2$

$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$, Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ (for $j \geq 0$)

Gradient checking:



$f(\theta+\varepsilon) - f(\theta-\varepsilon)$ $\theta \in \mathbb{R}$

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$ ← two-side difference.

↑ more accurate.

$\theta \in \mathbb{R}^n$ (i.e. θ is "unrolled" version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$).

$\theta = [\theta_1, \theta_2, \cdots, \theta_n]$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \cdots, \theta_n) - J(\theta_1-\varepsilon, \theta_2, \cdots, \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\varepsilon, \cdots, \theta_n) - J(\theta_1, \theta_2-\varepsilon, \cdots, \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \cdots, \theta_n+\varepsilon) - J(\theta_1, \theta_2, \cdots, \theta_n-\varepsilon)}{2\varepsilon}$$

} numerically estimate gradient of $J(\theta)$ w.r.t. to θ_j.

In practical: gradApprox ≈ Dvec

gradApprox ↓ estimation

Dvec ↓ From Back Prop ...

Implementation Note:
- Implement backprop to compute Dvec
- Implement numerical gradient check to compute gradApprox.
- Make sure they give similar values.
- Turn off gradient checking before using backprop code for training.

Random Initialization:
如果全部初始为0，hidden layer所有nodes都会学到一样的值.

Random Initialization: Symmetry breaking

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

eg. $\theta_1 = \text{rand}(10*(1)) * (2\varepsilon) - \varepsilon$.

Training a Neural Network:

1. Randomly initialize the weights;
2. Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$.
3. Implement the cost function $J(\theta)$
4. Implement Backprop to compute partial derivatives
5. Use gradient checking to confirm that your Backprop works. Then disable it
6. use gradient descent or other built-in optimization function to minimize the cost function. with the weights in theta.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$