Neural Network Cost Function

Binary classification	Multi-class classification (K classes)
1 output unit	K output units
$h_\Theta(x) \in \mathbb{R}$	$h_\Theta(x) \in \mathbb{R}^K$
$S_L = 1 \quad K = 1$	$S_L = k \quad (k \geq 3)$

Cost Function

$h_\Theta(x) \in \mathbb{R}^K \qquad (h_\Theta(x))_i = i^{th}$ output $\qquad y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \ldots$

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

Backpropagation Algorithm (for derivatives)

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

$\min_\Theta J(\Theta)$

Need code to compute

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

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

$\delta_j^{(4)} = \boxed{a_j^{(4)}} - y_j \quad (h_\Theta(x))_j \rightarrow$ vectorized $\quad \delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* \underline{g'(z^{(3)})} \quad a^{(3)} .* (1 - a^{(3)})$

$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* \underline{g'(z^{(2)})} \quad a^{(2)} .* (1 - a^{(2)})$

(No $\delta^{(1)}$)

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

Set $\Delta_{ij}^{(l)} = 0$ (for l, i, j) (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m — training example $(x^{(i)}, y^{(i)})$

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

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

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

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ ~~$\delta^{(1)}$~~

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ → vectorized $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

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

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

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

Back propagation Intuition

$\Theta_{ij}^{(l)}$ ← layer
to
from

Implementation note: Unrolling Parameters

Gradient Checking

$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta+\varepsilon) - J(\Theta-\varepsilon)}{2\varepsilon}$ ← 2-sided diff

$\varepsilon = 10^{-4}$

$\frac{J(\Theta+\varepsilon) - J(\Theta)}{\varepsilon}$ ← 1-sided diff

$\theta \in \mathbb{R}^n$

$\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$

$\frac{\partial}{\partial \theta_1} J(\theta) \simeq \frac{J(\theta_1+\varepsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1-\varepsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\varepsilon}$

⋮

Random Initialization

need initial value for θ

Zero initialization, makes $a_1^{(2)} = a_2^{(2)}$ & $\delta_1^{(2)} = \delta_2^{(2)}$ (BAD)

Computes all the same feature

random value in $[-\varepsilon, \varepsilon]$

Putting it Together

input units: Dimension of features $x^{(i)}$

output units: number of classes

Reasonable default: 1 hidden layer, if > 1 hidden layer, → same # of units in each

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 function $J(\theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\theta)$
5. Use gradient checking to compare $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$ computed using backprop vs. using estimate of gradient

   Disable gradient checking code
6. Use gradient descent or advanced optimization method w/ backprop to try to minimize $J(\theta)$

Autonomous Driving