Neural Network classification problem

training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots (x^{(m)}, y^{(m)})\}$

L: total no. of layers in the network

s_l: no. of units (exclude bias unit) in layer l

binary classification: $y = 0$ or 1 — 1 output unit

Multi-class classification (K classes) $\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix} \cdots \begin{bmatrix}0\\0\\\vdots\\0\\1\end{bmatrix}$ K output units

logistic regression

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

$h_\Theta(x) \in \mathbb{R}^K$ $\quad (h_\Theta(x))_i$: i^{th} output

Neural Networks: Backpropagation algorithm

Gradient Computation: need $J(\Theta)$ $\quad \frac{\partial J(\Theta)}{\partial \Theta_{ji}^{(l)}}$

e.g. 1 training example, 4 layers, forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)} \qquad a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)} \qquad a^{(4)} = h_\Theta(x) = g(z^{(4)})$$

Backpropagation algorithm

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

e.g. for layer 4: $\delta_j^{(4)} = a_j^{(4)} - y_j$

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

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

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

Mathematically: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignoring λ if $\lambda = 0$)

Backpropagation Algorithm

Training set $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$

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

for $i = 1$ to m:

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

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

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

↳ Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots \delta^{(2)}$

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

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

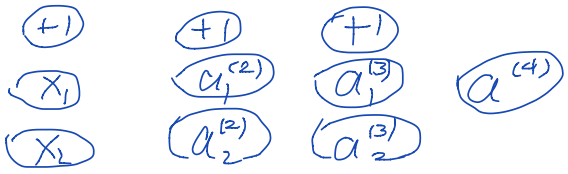
$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

→ use for gradient decent or other algorithms

Backpropagation intuition

forward propagation



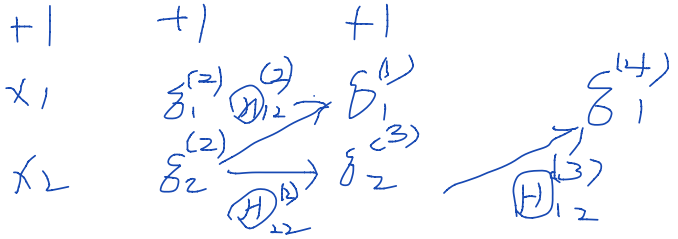
$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)}$$

backpropagation

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

$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l)

formally: $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$



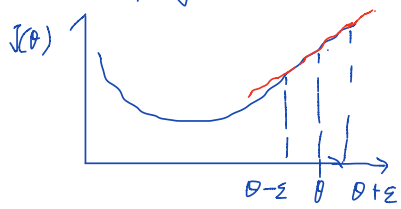
$$\delta^{(4)} = y^{(i)} - a_1^{(4)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta^{(4)}$$

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

Gradient Checking

Numerical estimation of gradients



$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

$$\varepsilon \sim 10^{-4}$$

Parameter vector θ

$\theta \in \mathbb{R}^n$ (unrolled version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$)

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

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

$$\frac{\partial}{\partial\theta_2}J(\theta) \approx \cdots$$

$$\vdots$$

$$\frac{\partial}{\partial\theta_n}J(\theta) \approx \cdots$$

check that above approx $\approx \frac{\partial}{\partial\theta_j^{(l)}}J(\theta)$ from back propagation

Implementation Note

1) Implement backprop. to compute DVec (unrolled $D^{(1)}, D^{(2)}, \cdots$)
2) Implement numerical gradient check to compute gradApprox
3) Make sure they give similar values
4) Turn off gradient checking, using backprop code for learning.

Random Initialization

Initial value of θ

☆. Initialize all parameters to 0 when training a

Neural network is NOT OK !

(after each update, parameters corresponding to inputs going into each of two hidden units are identical

This prevents your neural network from learning anything interesting)

Random Initialization: Symmetry breaking

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

Put it together

1) Pick a network architecture

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

No. of output units: number of classes

No. of hidden layers:

reasonable default: 1 hidden layer

or if >1 hidden layer, have same number of hidden units in each layer (usually the more the better)

2) Randomly initialize weights Θ

3) Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$

4) Implement code to compute cost function $J(\Theta)$

5) Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

for $i = 1 : m$ $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$

perform forward prop and backprop using example $(x^{(i)}, y^{(i)})$. get activations $a^{(l)}$ and delta $\delta^{(l)}$

for $l = 2, \cdots, L$

6) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ from backprop and numerical estimation, then disable the gradient checking

7) Use gradient decent or adv. method with backprop to try to minimize $J(\Theta)$ as a function of parameters Θ