



Backpropagation Algorithm

"Backpropagation" is neural-network terminology for minimizing our cost function, just like what we were doing with gradient descent in logistic and linear regression. Our goal is to compute:

$$\min_{\Theta} J(\Theta)$$

That is, we want to minimize our cost function J using an optimal set of parameters in Θ . In this section we'll look at the equations we use to compute the partial derivative of $J(\Theta)$:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$$

To do so, we use the following algorithm:

Backpropagation algorithm

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

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

For $i = 1$ to $m \leftarrow (x^{(i)}, y^{(i)})$.

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

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

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

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

$\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ *$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$*

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

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

$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)}$

Back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$

- Set $\Delta_{i,j}^{(l)} := 0$ for all (l, i, j) , (hence you end up having a matrix full of zeros)

For training example $t=1$ to m :

- Set $a^{(1)} := x^{(t)}$
- Perform forward propagation to compute $a^{(l)}$ for $l=2, 3, \dots, L$

Gradient computation