## Backpropagation Algorithm

### Gradient Computation ¶

The following equation is the cost function that we saw previously:

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

What we'd like to do is try to find parameters theta to try to minimize $J(\Theta)$.

goal: $\min_\Theta J(\Theta)$

In order to use either gradient descent or one of the advance optimization algorithms. What we need to do therefore is to write code that takes the input the parameters theta and computes $J(\Theta)$ and the partial derivative terms $J(\Theta)$ with respect $\Theta_{ij}^{(l)}$ (we remember that $\Theta_{ij}^{(l)} \in \mathbb{R}$).

Need to compute:

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

Let's start by an example with the case of when we have only one training example, and let's step through the sequence of calculations we would do with this one training example.

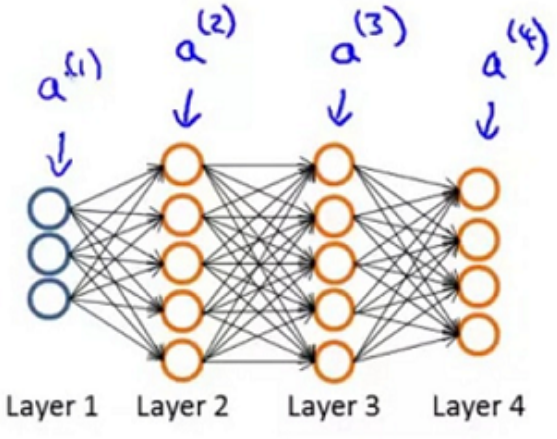
Given one training example (x, y):

The first thing we do is we apply **forward propagation** in order to compute whether a hypotheses actually outputs given the input. Concretely, we rememeber that we called to $a(1)$ the activation values of the first layer.

Given one training example $(x, y)$:

Forward propagation:

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

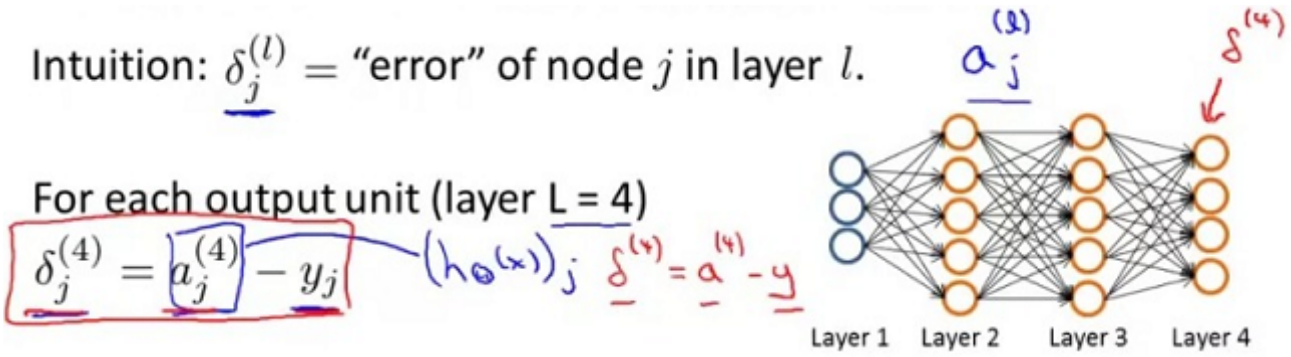



So this is our vectorized implementation of forward propagation and it allows us to compute the activation values for all of the neurons in our neural network.

In order to compute the derivatives, we're going to use an algorithm called back propagation.

## Gradient computation: Backpropagation algorithm

The intuition of the back propagation algorithm is that for each node we're going to compute the term $\delta_j^{(l)}$ that's going to somehow represent the error of node $j$ in the layer $l$, this delta term is in some sense going to capture our error in the activation of that neural duo (The only "real" value we have is our actual classification (our $y$ value) - so that's where we start).




For each output unit, we're going to compute the delta term. So, delta for the $j$ unit in the $4^{th}$ ($a_j^{(4)}$ is can also be written $h_\Theta(x)_j$) layer is equal to just the activation of that unit minus what was the actual value in our training example ($y_j$ whereas $y_j$ is the $j$ element of the vector value $y$ in our labeled training set).

We can do a vectorized implementation of this delta term: $\delta^{(4)} = a^{(4)} - y$. Where, each of the $\delta^{(4)} = a^{(4)} - y$, each of these is a vector whose dimension is equal to the number of output units in our network. What we are going to do next is compute the delta terms for the earlier layers in our network.

With $\delta^{(4)}$ calculated, we can determine the error terms for the other layers as follows:

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

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

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

- $\Theta^3$ is the vector of parameters for the $3$ -> $4$ layer mapping
- $\delta^4$ is (as calculated) the error vector for the $4^{th}$ layer
- $g'(z^3)$ is the first derivative of the activation function $g$ evaluated by the input values given by $z^3$

And finally, there is no $\delta^{(1)}$ term, because the first layer corresponds to the input layer and that's just the feature we observed in our training sets, so that doesn't have any error associated with that, and we don't really want to try to change those values. And so we have delta terms only for layers $2, 3$ in this example.

The name back propagation comes from the fact that we start by computing the delta term for the output layer and then we go back a layer and compute the delta terms for the third hidden layer and then we go back another step to compute delta 2 and so, we're sort of back propagating the errors from the output layer to layer $3$ to their to hence the name back propagation.

## Backpropagation algorithm with large training set

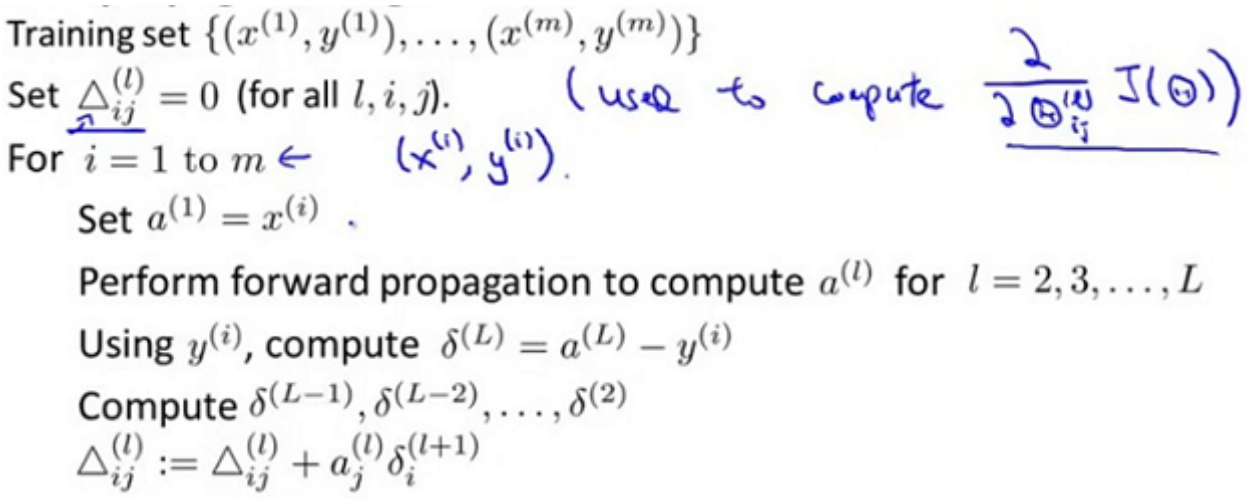
The following is a training set with $m$ examples.

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

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

The triangle is capital delta. We're gonna set this equal to zero for all values $(l, i, j)$. Eventually, the capital delta will be used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

Concretely, we're going to loop through our training set.




We'll say for $i = 1$ through $m$ and so for the $i$ iteration, we're going to working with the training example $(x^{(i)}, y^{(i)})$. We're going to use the capital delta term to accumulate the partial derivative terms.

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

Note here: $l$ = layer, $j$ = node in that layer and $i$ = the error of the affected node in the target layer

Finally, after executing the body of the four-loop we then go outside the four-loop and we compute the following. We compute $\Delta$ as follows and we have two separate cases for $j = 0$ and $j \neq 0$.

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

The case of $j = 0$ corresponds to the bias term so when $j = 0$ that's why we're missing the extra regularization term. The capital-delta matrix $D$ is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get:

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

**Video Question:** Suppose you have two training examples $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

- FP using $x^{(1)}$ followed by FP using $x^{(2)}$. Then BP using $y^{(1)}$ followed by BP using $y^{(2)}$.
- FP using $x^{(1)}$ followed by BP using $y^{(2)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(1)}$.
- BP using $y^{(1)}$ followed by FP using $x^{(1)}$. Then BP using $y^{(2)}$ followed by FP using $x^{(2)}$.

> FP using $x^{(1)}$ followed by BP using $y^{(1)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(2)}$.

## Summary

"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 theta. 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 $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

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

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

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

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

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

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

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

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

**Back propagation Algorithm**

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

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

For training example $t = 1$ to $m$:

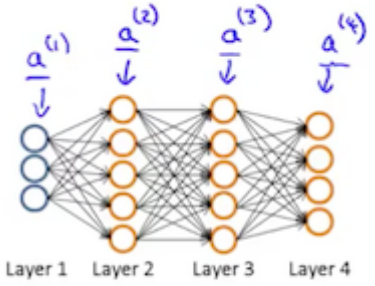
1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

**Gradient computation**

Given one training example $(x, y)$:

Forward propagation:

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




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

Where $L$ is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in $y$. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \, .* \, a^{(l)} \, .* \, (1 - a^{(l)})$

The delta values of layer $l$ are calculated by multiplying the delta values in the next layer with the theta matrix of layer $l$. We then element-wise multiply that with a function called $g'$, or g-prime, which is the derivative of the activation function $g$ evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:

$$g'(z^{(l)}) = a^{(l)} \, .* \, (1 - a^{(l)})$$

5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

Hence we update our new $\Delta$ matrix.

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

The capital-delta matrix $D$ is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$