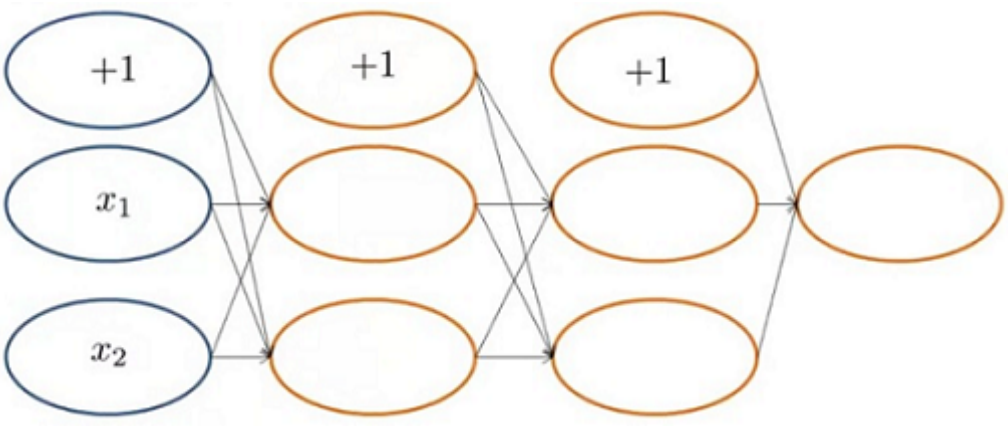
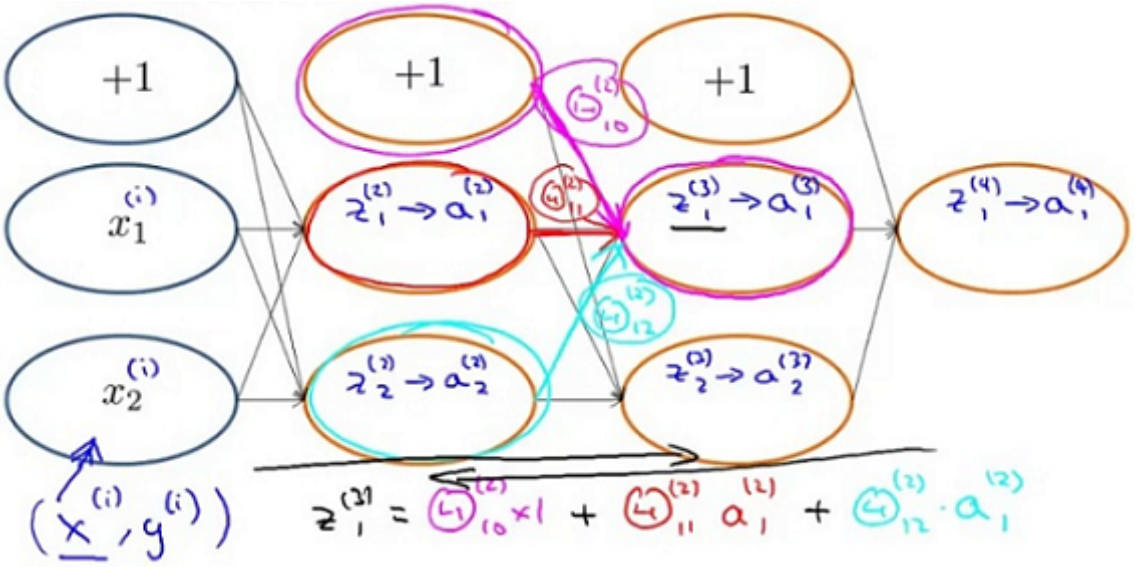
Backpropagation Intuition

Forward Propagation ¶

In order to better understand backpropagation, let's take another closer look at what forward propagation is doing. The following picture is a neural network with two input units that is not counting the bias unit, and two hidden units in the next layer. And then, finally, one output unit.



When performing forward propagation, we might have some particular example. Say some example $(x^{(i)}, y^{(i)})$. And when we forward propagated to the first hidden layer, what we do is compute $z_1^{(2)}$ and $z_2^{(2)}$. So these are the weighted sum of the input units.



And then we apply the sigmoid of the logistic function, the sigmoid activation function applied to the z value. And then we forward propagate again to get $z_1^{(3)}$ and $z_2^{(3)}$ (and we apply the activation function to these variables and we get $a_1^{(3)}$ and $a_2^{(3)}$). And similarly, like so until we get $z_1^{(4)}$.

Apply the activation function. This gives us $a_1^{(4)}$, which is the final output value of the neural network. Concretely, the sigmoid function applied to the z values gives the activation values.

It turns out that, what backpropagation is doing, is a process very similar to this. Except that instead of the computations flowing from the left to the right of this network, the computations flows from the right to the left of the network. And using a very similar computation as this.

What is backpropagation doing?

To better understand what backpropagation is doing, let's look at the cost function. It's just the cost function that we had for when we have only one output unit, and we do forward propagation and backpropagation on one example at a time.

So let's just focus on the single example, $(x^{(i)}\ y^{(i)})$ and focus on the case of having one output unit.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{t=i}^{m} y^{(i)}\ \log(h_\Theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - (h_\Theta(x^{(i)})))\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{j,i}^{(l)})^2$$

Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$)

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

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

For the purpose of intuition, we think of the cost function as being the sort of the squared error cost function. And so the $cost(i)$ measures how well is the network doing on correctly predicting example i.

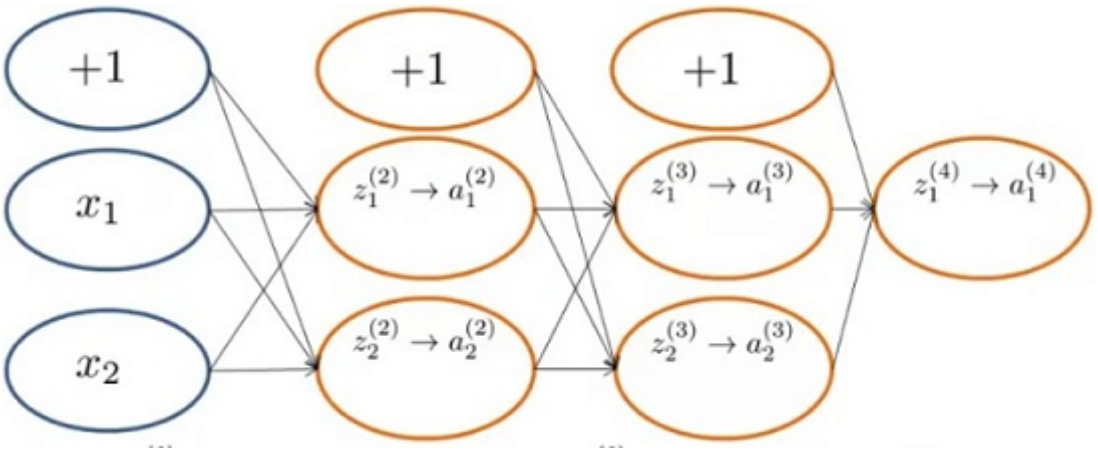
Backpropagation

One useful intuition is that backpropagation is computing the $\delta_j^{(l)}$ terms, and we can think of these as the error of the activation value $a_j^{(l)}$ that we got for unit j in the l^{th} layer.

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

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

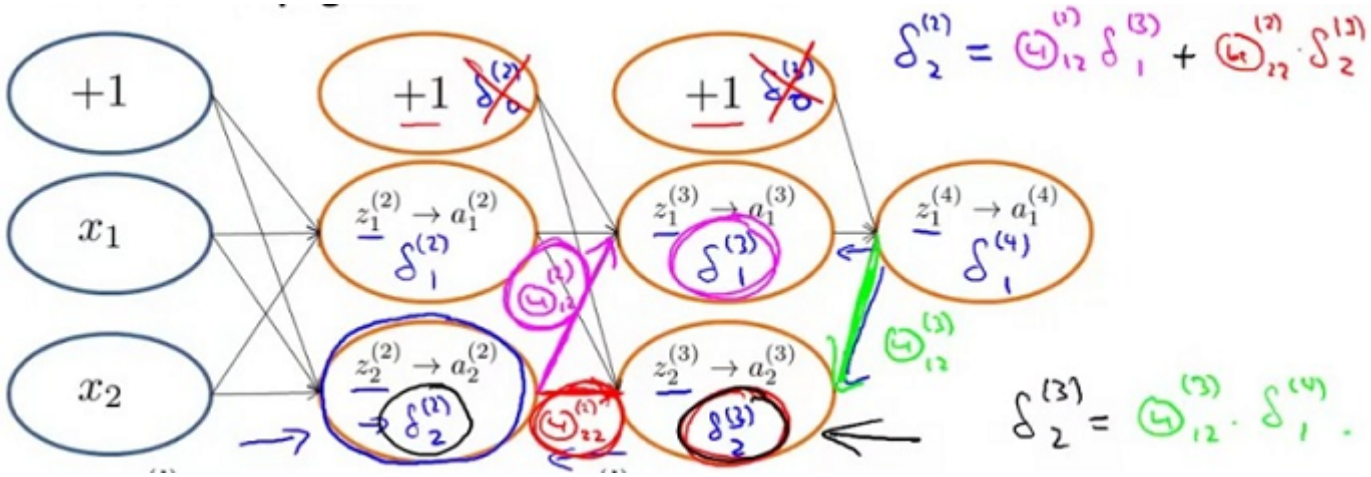
$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$



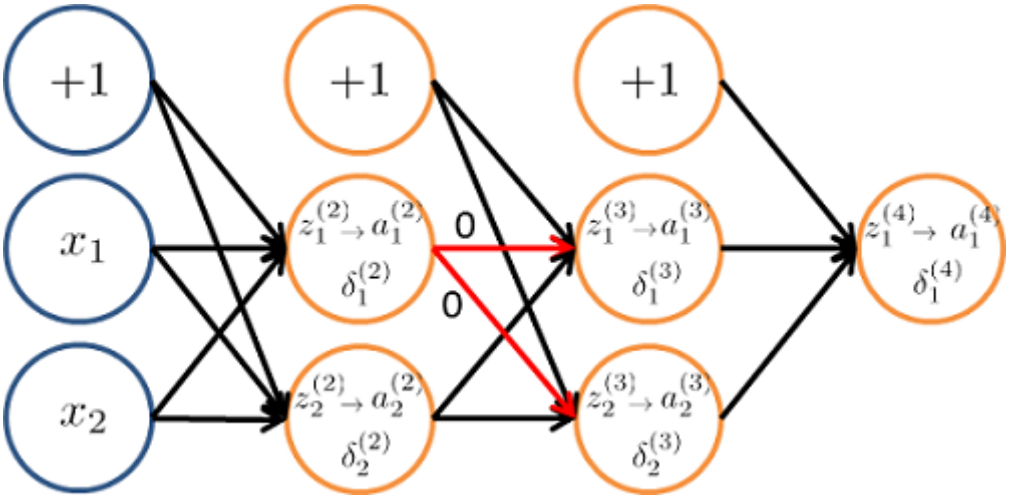
Concretely the Cost function is a function of y value and the hypothesis function

- So - for the output layer, back propagation sets the δ value as [a - y]
- Difference between activation and actual value

The backpropagation calculation is a lot like running the forward propagation algorithm, but doing it backwards.



Video Question: Consider the following neural network:



Suppose both of the weights shown in red ($\Theta_{11}^{(2)}$ and $\Theta_{21}^{(2)}$) are equal to 0. After running backpropagation, what can we say about the value of $\delta_1^{(3)}$?

- $\delta_1^{(3)} > 0$
- $\delta_1^{(3)} = 0$ only if $\delta_1^{(2)} = \delta_2^{(2)} = 0$, but not necessarily otherwise
- $\delta_1^{(3)} \leq 0$ regardless of the values of $\delta_1^{(2)}$ and $\delta_2^{(2)}$

> There is insufficient information to tell

Summary

Recall that the cost function for a neural network is:

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

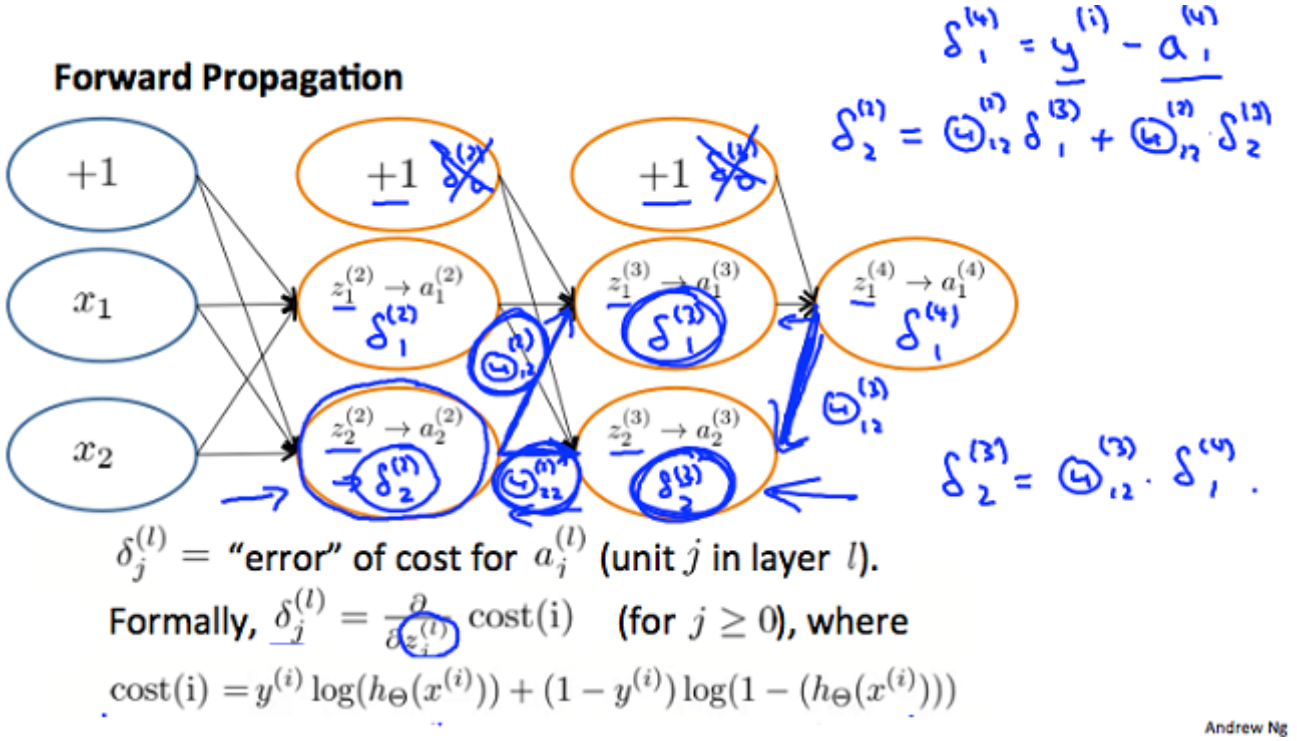
If we consider simple non-multiclass classification ($k = 1$) and disregard regularization, the cost is computed with:

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

Intuitively, $\delta_j^{(l)}$ is the "error" for $a_j^{(l)}$ (unit j in layer l). More formally, the delta values are actually the derivative of the cost function:

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(t)$$

Recall that our derivative is the slope of a line tangent to the cost function, so the steeper the slope the more incorrect we are. Let us consider the following neural network below and see how we could calculate some $\delta_j^{(l)}$:



In the image above, to calculate $\delta_2^{(2)}$, we multiply the weights $\Theta_{12}^{(2)}$ and $\Theta_{22}^{(2)}$ by their respective δ values found to the right of each edge. So we get $\delta_2^{(2)} = \Theta_{12}^{(2)} * \delta_1^{(3)} + \Theta_{22}^{(2)} * \delta_2^{(3)}$. To calculate every single possible $\delta_j^{(l)}$, we could start from the right of our diagram.

We can think of our edges as our Θ_{ij}. Going from right to left, to calculate the value of $\delta_j^{(l)}$, you can just take the over all sum of each weight times the δ it is coming from. Hence, another example would be $\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$.