















Cost Function and Backpropagation

	Cost Function	6 min
	Cost Function	4 min
	Backpropagation Algorithm	11 min
	Backpropagation Algorithm	10 min
	Backpropagation Intuition	12 min
	Backpropagation Intuition	4 min

Backpropagation in Practice

	Implementation Note: Unrolling Parameters	7 min
	Implementation Note: Unrolling Parameters	3 min
	Gradient Checking	11 min
	Gradient Checking	3 min
	Random Initialization	6 min
	Random Initialization	3 min
	Putting It Together	13 min
	Putting It Together	4 min

Application of Neural Networks

Review

Backpropagation Intuition

Note: [4:39, the last term for the calculation for z_1^3 (three-color handwritten formula) should be a_2^2 Instead of a_1^2 . 6:08 - the equation for cost(l) is incorrect. The first term is missing parentheses for the log() function, and the second term should be $(1 - y^{(i)})\log(1 - h_{\Theta}(x^{(i)}))$. 8:50 - $\delta^{(4)} = y - a^{(4)}$ is incorrect and should be $\delta^{(4)} = a^{(4)} - y$.]

Recall that the cost function for a neural network is:

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

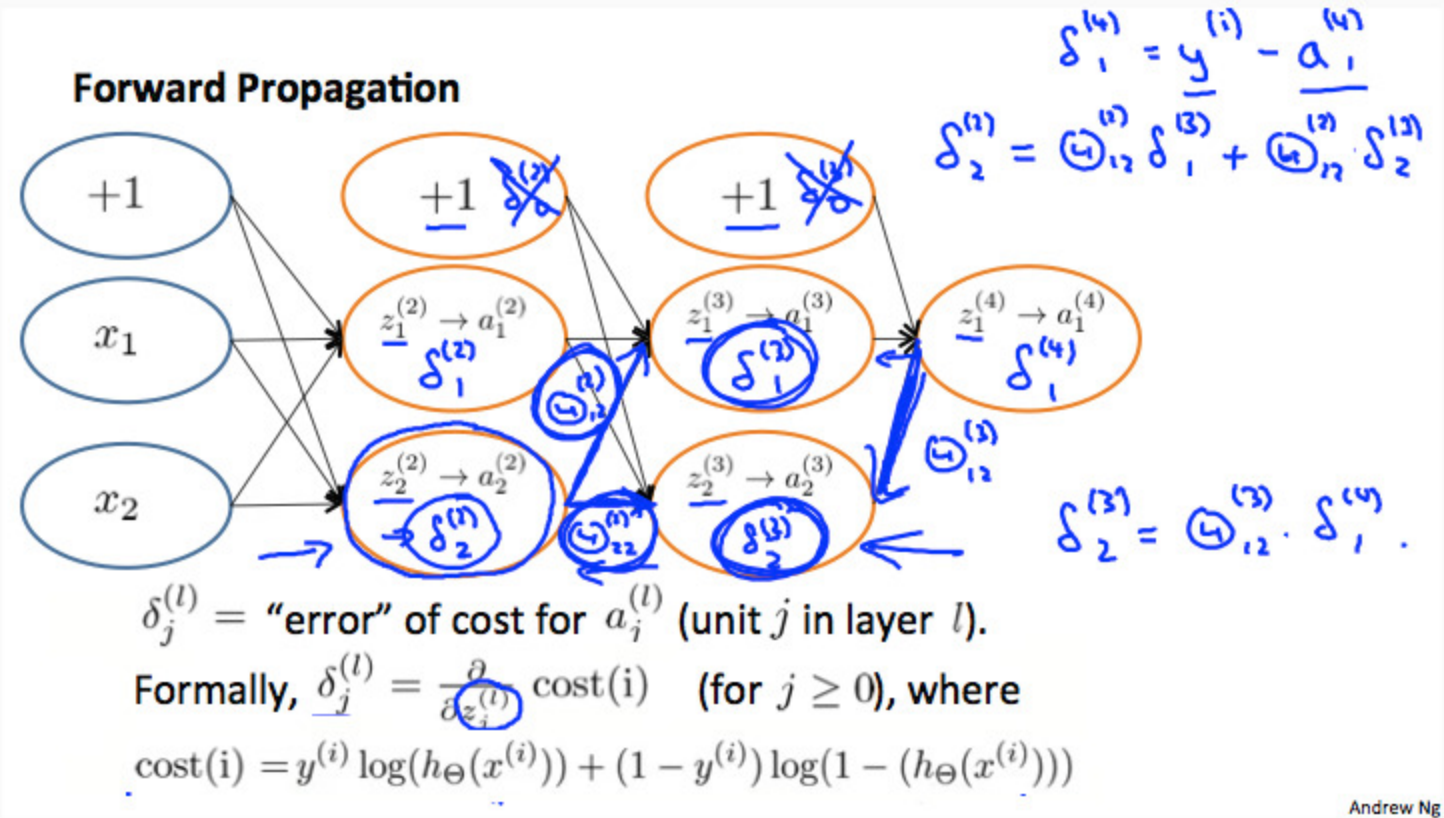
If we consider simple non-multiclass classification (k = 1) and disregard regularization, the cost is computed with:

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

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:

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

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 $\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$.

✔ Completado