Back Propagation

Back propagation is how neural networks *learn*. They take the final output at the output layer and work it's way back to layer 2.

In back propagation we want to capture the error, for each unit, of how far off it was when computing the hypothesis during forward propagation. To find the error of each node in each layer we start at the last layer and progress to the the second layer (non-input layer) and capture the error for each unit in each layer. The error term is represented as $\delta_j^{(l)}$ (delta) and computed by simply subtracting the hypotheses at each unit $a_j^{(l)}$ by the correct value y_j from our labeled training data.

For and example with a 4 layer network ($L = 4$), back propagation would start at layer 4 with the below where j is the index of each output unit:

> $\delta_j^{(4)} = a_j^{(4)} - y_j$

Which can also be written as:

> $\delta_j^{(4)} = h_\Theta(x)_j - y_j$

This can easily be vectorized as the following where a and y are vectors with dimensions equal to the number of output units in the network (in this instance 4):

> $\delta^{(4)} = a_j^{(4)} - y$

For the remainder of the layers, the computation is a little different. In using vectors for each layer, we take the error from the previous layer and multiply that by the Θ values for that current layer. Then, perform *element wise multiplication* $\odot$ by the *prime* of the activation units for that layer:

> **Layer 3**
>
> $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \odot g\prime(z^{(3)})$
>
> **Layer 2**
>
> $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \odot g\prime(z^{(2)})$

> **NOTE:** Elementwise multiplication of matrices is called the *Hadamard product* or *Schur product*.
>
> **NOTE:** $\delta_j^{(i)}$ is actually the partial derivative with respect to $z_j^{(l)}$ of the cost function $J(i)$ where $J(i) = y_k^{(i)} \cdot log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \cdot log(1 - h_\Theta(x^{(i)}))_k)$ which is the NN cost function.

Here $g\prime$ (g prime) is the *derivative* of the specific activation function which can be derived by using Calculus. For example, if we were using *Sigmoid* functions for each unit in a particular layer:

> $g(z) = \frac{1}{1+e^{-z}}$

The derivative of *Sigmoid* would be:

> $g\prime(z) = g(z) \cdot (1 - g(z))$

What the derivative is actually giving us is the gradients for that layer with our objective being finding the global minimum.

If there is no regularization involved (ignoring λ or $\lambda = 0$) in back propagation then the partial derivative terms you want are exactly given by activations and delta terms:

> $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_1^{(l+1)}$

Back Propagation With a Large Training Set

If we have a large training set such as:

> $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

We will initialize all values to zero:

> $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

The Δ term will be used to compute the partial derivative with respect to:

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

Below is the basic logic for forward and back propagation:

> for ($i = 1$ to m)
>
> set $a^{(1)} = x^{(i)}$ // Set the activations of the input layer
>
> $a^{(L)} =$ forward_prop() // Perform forward propagation to compute the activations for all layers
>
> $\delta^{(L)} = a^{(L)} - y^{(i)}$ // Compute the error term for the output layer
>
> $[\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}] =$ back_prop() // Perform back propagation
>
> $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ // Accumulate all the partial derivatives
>
> $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$ // Or accumulate using a vectorized implementation
>
> // Compute the derivative with respect to each of the parameters
>
> $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$
>
> $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

Computing the derivative $D_{ij}^{(l)}$ can be simply expressed as:

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

These derivatives $D_{ij}^{(l)}$ can then be used in *Gradient Decent* or other more advanced algorithms for finding the minimum.