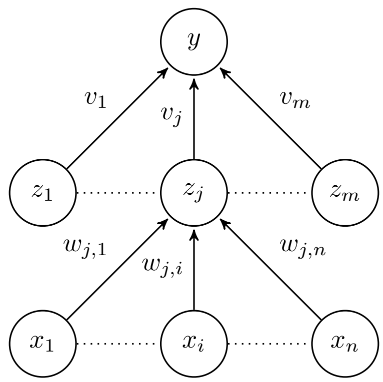
Backpropagation Derivation

Fabio A. González
Universidad Nacional de Colombia, Bogotá

March 21, 2018

Consider the following multilayer neural network, with inputs $x_1, \ldots, x_n$:



The dynamics of the network is given :

$$a_j = \sum_i w_{ji} x_i$$

$$z_j = \sigma(a_j)$$

$$a_y = \sum_j v_j z_j$$

$$y = \sigma(a_y)$$

with

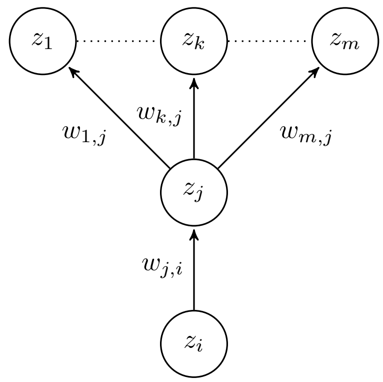
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The loss function to minimize is:

$$E_\ell(w) = -r^\ell \log y^t - (1 - r^\ell) \log(1 - y^t)$$

Since we are going to use gradient descent our problem is to calculate the gradient $\frac{\partial E_\ell}{\partial w_i}$ for every i.

For this we will consider a more general situation for an internal neuron z_j, which is in layer n of a multilayer neural network:



Where

$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j)$$

The strategy is to express the gradient in terms of two quantities that allow an easy and efficient calculation:

$$\frac{\partial E_\ell}{\partial w_{ji}} = \frac{\partial E_\ell}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

This is a direct application of the chain rule. The first term is called:

$$\delta_j = \frac{\partial E_\ell}{\partial a_j}$$

and the second is just z_i since:

$$\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial \sum_i w_{ji} z_i}{\partial w_{ji}} = z_i$$

So

$$\frac{\partial E_\ell}{\partial w_{ji}} = \delta_j z_i \tag{1}$$

z_i is calculated when we forward propagate samples through the net. For calculating δ_j we will derive a rule. Before doing this we first need to remember the generalized (or multidimensional) chain rule (GCR):

$$y = f(u_1, \ldots, u_m)$$

$$\mathbf{u} = g(x_1, \ldots, x_n)$$

$$\frac{\partial y}{\partial x_i} = \sum_{\ell=1}^{m} \frac{\partial y}{\partial u_\ell} \frac{\partial u_l}{\partial x_i}$$

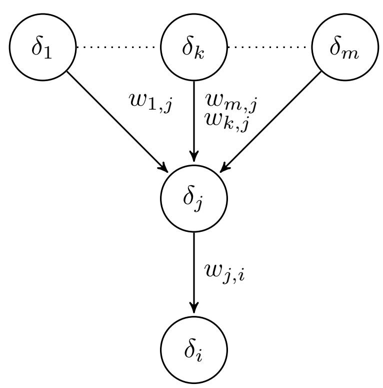
Now we can proceed. We are considering neural networks organized in layers. If neuron z_j is in layer n, the neurons $z_1, \ldots, z_k, \ldots, z_m$ are in layer $n+1$. Notice that in general

$$E_\ell = f(a_1, \ldots, a_k)$$

for some function f. Applying the chain rule:

$$\begin{aligned}
\delta_j = \frac{\partial E_\ell}{\partial a_j} &= \sum_{k=1}^{m} \frac{\partial E_\ell}{\partial a_k} \frac{\partial a_k}{\partial a_j} \\
&= \sum_{k=1}^{m} \delta_k \frac{\partial a_k}{\partial a_j} \\
&= \sum_{k=1}^{m} \delta_k \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \qquad (2) \\
&= \sum_{k=1}^{m} \delta_k w_{kj} h'(a_j) \\
&= h'(a_j) \sum_{k=1}^{m} \delta_k w_{kj}
\end{aligned}$$

This gives us a rule to calculate δ in layer n with base in values from layer $n-1$. This is illustrated by the following diagram:



In this case, the δ values are propagated backwards, or back-propagated. This is where the name of the method comes from.

The backpropagation algorithm is formulated as follows:

1: `Initialize` $\mathbf{w}$
2: `for` n `= 1 to` num epochs
3:     `for all` $x^\ell \in D$
4:         `Forward propagate` x^ℓ `through the network`
           `to calculate the` a_j `and` z_j `values`
5:         `Calculate` $\delta_o = \frac{\partial E_\ell}{\partial a_o}$
           `for all the output neurons`
6:         `Backward propagate` δ_j `values`
           $\delta_j = h'(a_j) \sum_{k=1}^{m} \delta_k w_{kj}$
7:         `for all` $w_{ji} \in \mathbf{w}$
8:             $\Delta w_{ji} \leftarrow \delta_j z_i$
9:             $w_{ji} \leftarrow w_{ji} - \eta_n \Delta w_{ji}$

For our original example:

$$\delta_y = \frac{\partial E_\ell}{\partial a_y} = \sigma(a_y) - r^\ell = y - r^\ell$$

$$\delta_j = \sigma'(a_j)\delta_y v_j = \sigma(a_j)(1 - \sigma(a_j))\delta_y v_j = z_j(1 - z_j)\delta_y v_j$$