BACKPROP

Gradient descent requires being able to compute the partial derivatives of the Cost function C with respect to the weights. But C is given in terms of the activations at the output layer (and in terms of input values at layer 0, and the desired target value y), not in terms of weights at all. Yet the output activations clearly are dependent on the weights; so the calculus chain rule is called for. We will proceed by referring back at times to the simple example in the previous section, while at the same time developing the general method.

We write C(**w**), where **w** is a list of all the w's. [There are five in our earlier example: three from layer 0 to layer 1, and two from layer 1 to layer 2. The illustrated earlier equation (*) for one of the outputs, and its companion (not shown) for the other output value, are what C will ultimately compare to the desired result y for a given input $x = (a_0^0, a_1^0, a_2^0) = (1, a_1^0, a_2^0)$.] The sigmoid function of course is involved, over and over; and MeanSquaredError (MSE, for the Cost, described below) at the end. But the only variables of interest are the w's (and others defined in terms of them, such as the z's and a's); everything else is considered fixed while gradient descent is being performed.

But since the w's are deeply embedded inside a rather complicated expression, we must use the chain rule. Fortunately, the derivative of the sigmoid is very nice:

$$\sigma' = \sigma(1 - \sigma)$$

but we will not even need to know that for the general development.

Let's get to work. As noted, we will use MSE for our comparator/cost function C. MSE (and hence C) is defined as follows:

$$MSE = \frac{1}{n}\sum_{k=1}^{n} \|\boldsymbol{a_k} - \boldsymbol{y_k}\|^2$$

where for each k, $\boldsymbol{a_k}$ and $\boldsymbol{y_k}$ are the vectors of the output values and the desired values, respectively, k running over all training pairs **(**n being the number of such pairs). (In our earlier notation above we would write **a** as $\boldsymbol{a^L}$ where L is the output layer.) The y-values are fixed (and independent of the w's); it is the a-values that depend on the w's. So we could compute the partial derivatives of C (i.e., MSE) with respect to the output a's, and then the partials of those a's wrt the w's as from equation (*).

To keep things simple (and also because this is sometimes done in practical applications) we will focus attention on just one training pair (**x,y**) at a time (i.e., one value of k – one can imagine that there is only one training pair). In such a case, C becomes

$$\sum(\mathrm{a_i} - y_i)^2$$

where i runs over the output-layer units (other than the bias unit). [Question: how does one get this expression from the one above when n=1?]

As noted in the previous section, computing the partials with respect to all the w's directly from calculus definitions (difference quotients) presents a computational nightmare, compared to just the "local" partials, with respect to one layer below. It turns out that this can be done without invoking difference quotients at all – in fact without making estimates – but instead with purely formal methods.

So here we go, for our example in the output layer L=2:

$$C = \sum (\sigma(z_i^2) - y_i)^2$$

Then:

$$\frac{\partial C}{\partial z_i^2} = 2(\sigma(z_i^2) - y_i) \cdot \sigma'(z_i^2)$$

$$= 2(\sigma(z_i^2) - y_i) \cdot \sigma(z_i^2)(1 - \sigma(z_i^2))$$

Now to find how C depends on the weights connected to the output layer, i.e., to find $\partial C/\partial w_{ij}^1$, for instance, we can use the chain rule as follows:

$$\frac{\partial C}{\partial w_{ij}^1} = \sum_{k=1}^{2} \frac{\partial C}{\partial z_k^2} \cdot \frac{\partial z_k^2}{\partial w_{ij}^1}$$

But $\partial z_k^2/\partial w_{ij}^1$ is just a_i^1 when k=i, and 0 otherwise. So

$$\frac{\partial C}{\partial w_{ij}^1} = \frac{\partial C}{\partial z_j^2} \cdot a_i^1$$

Let's call $\frac{\partial C}{\partial z_i^2}$ the "error" to the input in unit i of layer 2, and more generally we introduce the notation $\delta_u^l = \partial C/\partial z_u^l$ and refer to it as the error to the input to unit u in layer l. Why the error? Well, for the same reason that gradient descent makes sense. Any nonzero value of this partial derivative corresponds to a possible replacement value of z (recall this is the total weighted input to u) that makes C smaller (by adding minus the partial to z).

So, where are we? Above we saw an expression for $\frac{\partial C}{\partial z_i^2}$ that can be computed straightforwardly; a related – and recursive – result holds for other partials of C wrt z's at earlier layers. This is quite general, and not dependent on our simple example.

So we now present the four fundamental equations that allow efficient computation of the needed partials of C wrt the w's. This involves the a's, the z's, and the delta's, as well as the w's.

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$$

$$\delta^l = ((W^l)^T \delta^{l+1}) \odot \sigma'(z^L)$$

$$\frac{\partial C}{\partial w^l_{0j}} = \delta^{l+1}_j$$

$$\frac{\partial C}{\partial w^l_{ij}} = a^l_i \, \delta^{l+1}_j$$

The bull's-eye symbol is the so-called Hadamard (vector) product, and is simply the vector whose elements consist of all the products of the corresponding separate elements in the two vectors being "multiplied". So it's just a shorthand for writing out those many products. That is, for instance, the Hadamard product of (a,b,c),(d,e,f) is (ad,be,cf).

The second equation is highly illustrative of the underlying ideas: it says the "errors" at layer l are determined by those at the layer l+1 just above. The matrix W^l is *transposed* since in this equation we are computing how information at layer l+1 tells us something about layer l – the opposite of what W^l itself tells us (namely how to find activations at layer l+1 from layer l). In addition, sigma (or rather its derivative) is being applied here to *all* the elements z^L_u of the vector z^L, one by one and themselves then collected into a vector; this is again just another shorthand.

Also, the last equation (when j=0) subsumes the third one (since the a-values for bias units are always just 1). So strictly speaking that third equation is redundant; but since it deals specifically with bias units, it is often given separately.

Let's prove the *first equation*, written out in terms of the u-th element on both sides:

$$\delta^L_u = \frac{\partial C}{\partial a^L_u} \sigma'(z^L_u)$$

Starting on the LHS, we have (by definition)

$$\delta^L_u = \frac{\partial C}{\partial z^L_u}$$

and so by the chain rule

$$\delta^L_u = \sum_i \frac{\partial C}{\partial a^L_i} \frac{\partial a^L_i}{\partial z^L_u}$$
$$= \sum_i \frac{\partial C}{\partial a^L_i} \frac{\partial \sigma(z^L_i)}{\partial z^L_u}$$
$$= \partial C / \partial a^L_u \sigma'(z^L_u)$$

since for all i other than u, z^L_i has no dependence on z^L_u.

We turn now to the *second equation*, again written in component terms:

$$\delta_u^l = \sum_k w_{uk}^l \delta_k^{l+1} \sigma'(z_u^l)$$

[Note that I have switched the subscripts on w, so u comes first; as k varies we are looking at different units at level l+1 receiving input from unit u below. That is, these w-values do not form the column vector of weights going up to one unit from all the units one level down, but rather to a row vector of weights coming from the single unit u and all going up to different units k above it. This is the sense of the transpose in the abstract version of the equation.]

We use the chain rule on the LHS:

$$\delta_u^l = \frac{\partial C}{\partial z_u^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_u^l}$$

$$= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_u^l}$$

$$= \sum_k \delta_k^{l+1} \partial / \partial z_u^l \sum_j a_j^l w_{jk}^l$$

$$= \sum_k \delta_k^{l+1} \partial / \partial z_u^l \sum_j \sigma(z_j^l) w_{jk}^l$$

and since the 2nd sum is zero except when j=u, we then get

$$= \sum_k \delta_k^{l+1} \frac{\partial}{\partial z_u^l} \sigma(z_u^l) w_{uk}^l$$

$$= \sum_k w_{uk}^l \delta_k^{l+1} \sigma'(z_u^l)$$

Unlike the first two equations – which provide a way to compute the *delta* values (starting at the output layer and working down) – the third and fourth equations provide a way to compute partial derivatives of C wrt all the weights (using the deltas, but still working down from output layer – which is where C after all is defined in the first place – to input layer).

We will prove the *third equation* now (leaving the proof of the fourth as an exercise); we need to show:

$$\frac{\partial C}{\partial w_{0j}^{l}} = \delta_j^{l+1}$$

Working on the LHS, we have by the chain rule:

$$\frac{\partial C}{\partial w_{0j}^{l}} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \left(\frac{\partial z_k^{l+1}}{\partial w_{0j}^{l}}\right)$$

$$= \sum_k \delta_k^{l+1} \; \partial \left[\sum_i a_i^l w_{ik}^l\right] / \partial w_{0j}^l$$

$$= \delta_j^{l+1} \frac{\partial a_0^l w_{0j}^l}{\partial w_{0j}^l}$$

(since the inner sum has zero derivative wrt w^l_0j except when i=0 and k=j; this also uses the fact that activations a^l depend on the w's in layer l-1, not those in layer l)

$$= \delta_j^{l+1} a_0^l = \delta_j^{l+1}$$