

## Neural Networks II: Backprop

Backprop is the style of gradient descent used to train multilayer neural networks. It's basically a gross application of the chain rule of derivatives:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \times \frac{\partial a}{\partial w} \quad (1)$$

The idea is to first do a forward pass to obtain activations at each hidden unit ( $a_j$ ) and sigmoided-values of these  $z_j = \tanh(a_j)$ . These are finally combined into predictions  $\hat{y}$ . This  $\hat{y}$  is compared to the true  $y$  using the appropriate loss function and a derivative is computed. In the case of squared error, we get  $\delta_y = \hat{y} - y$  – the difference between our predict and the truth. We then propagate these errors back, producing errors at internal nodes:  $\delta_j = (1 - z_j^2)u_j\delta_y$ , where  $u_j$  is the weight connecting the internal node to the output. We finally compute the actual derivatives as:

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i \quad (2)$$

$$\frac{\partial E}{\partial u_j} = \delta_y z_j \quad (3)$$

One nice thing about two-layer networks is that they can learn arbitrary reasonable<sup>1</sup> functions (though they might need lots of units to do so). As a simple example, you can construct a two-layer network that represents xor by essentially having one hidden unit represent “and” and one represent “or” (which we know are linear functions) and then combine them via roughly “or minus and.”

There are several big difficulties in using neural networks that has led to them being less popular recently (though see BT2 for recent efforts on very deep networks). The key difficulties are:

- **Architecture selection:** how many layers, how many hidden units (can partially be addressed by *learning* the structure, but no one seems to do this, so I assume it doesn't work well).
- **Symmetries:** There are lots of different parameters that give rise to the same model; this redundancy makes optimization difficult. This also makes initialization a bit of an art.
- **Non-convexity:** You pay a big price for non-linear models: you get non-convex optimization problems, which means local optima, saddle points and all the things we hate about non-linear models. If you want an argument that non-convexity isn't that terrible, see Yann LeCun's talk on “Who's afraid of non-convex loss functions” at [http://videlectures.net/eml07\\_lecun\\_wia/](http://videlectures.net/eml07_lecun_wia/).

---

<sup>1</sup>eg., continuous, differentiable, Lipschitz, etc.