How to Transform the Output into Error Function?

Gradient descent is a really good way of working out the minimum of a function, and it really works well when that function is so complex and difficult. In this lesson, we will learn how to design that function from our Neural Net, which we can then pass to the gradient descent algorithm.

The output of a neural network is a difficult complex function with many parameters, the link weights, which influence its output. So we can use gradient descent to work out the right weights? Yes, as long as we pick the right error function. The output function of a neural network itself isn't an error function. But we know we can turn it into one easily because the error is the difference between the target training values and the actual output values.

There's something to watch out for here. Look at the following table of training and actual values for three output nodes, together with candidates for an error function.

Network Output	**Target Output**	**Error (target - actual)**	**Error |target - actual|**	**Error** $(target - actual)^2$
0.4	0.5	0.1	0.1	0.01
0.8	0.7	-0.1	0.1	0.01
1.0	1.0	0	0	0
sum colspan="2"		0	0.2	0.02

The first candidate for an error function is simply the $(target - actual)$. That seems reasonable enough, right? Well if you look at the sum over the

nodes to get an overall figure for how well the network is trained, you'll see the sum is zero!

What happened?

Clearly, the network isn't perfectly trained because the first two node outputs are different to the target values. The sum of zero suggests there is no error. This happens because the positive and negative errors cancel each other out. Even if they didn't cancel out completely, you could see this is a bad measure of error.

Let's correct this by taking the *absolute* value of the difference. That means ignoring the sign and is written $(target - actual)$. That could work because nothing can ever cancel out. The reason this isn't popular because the slope isn't continuous near the minimum and this makes gradient descent not work so well because we can bounce around the V-shaped valley that this error function has.

The slope doesn't get smaller closer to the minimum, so our steps don't get smaller, which means they risk overshooting. The third option is to take the square of the difference $(target - actual)^2$.