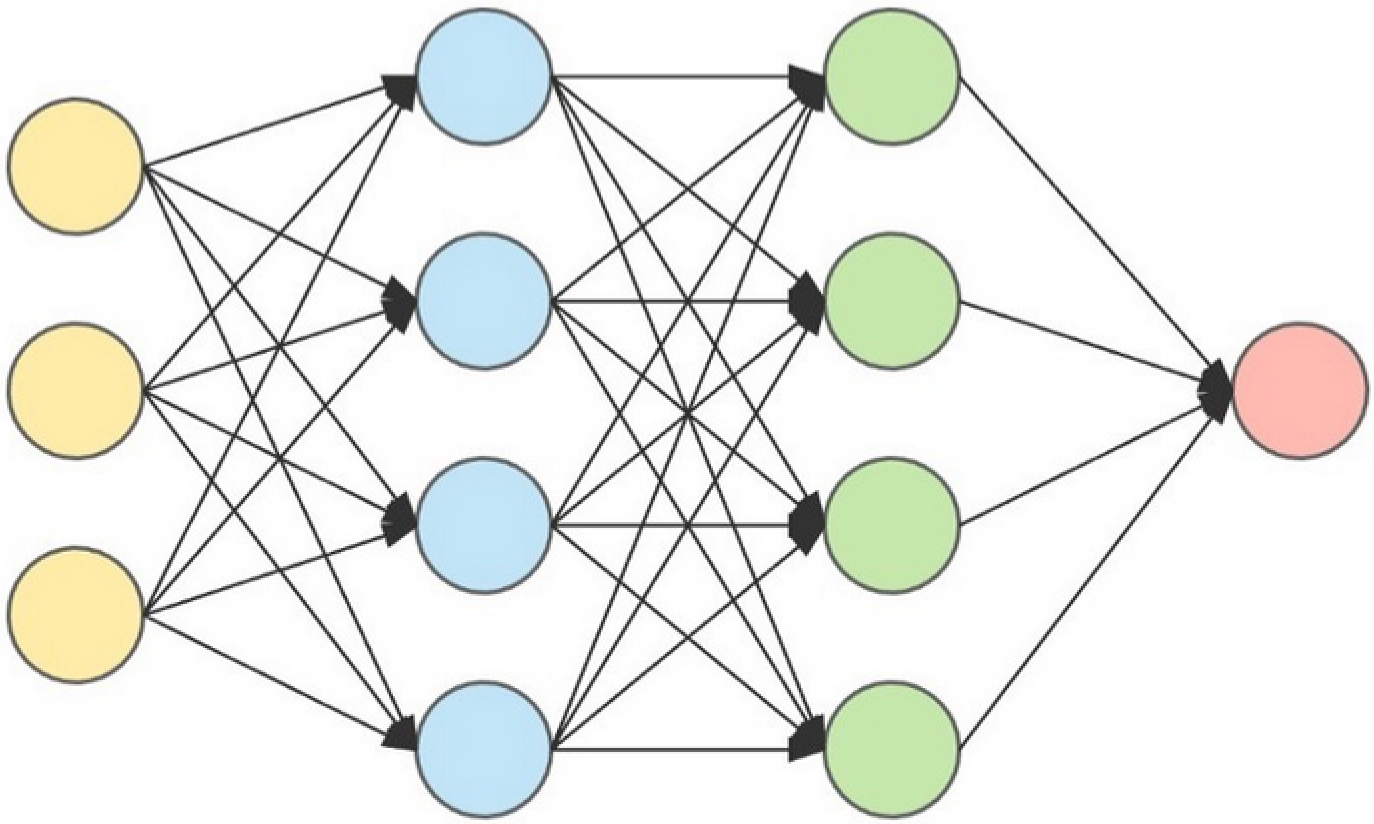
Title

How to update Neural Network’s link weights?



First, we got to the point where all the error are back propagated to each layer of the network. Why did we do this? We did this to improve the link weights based on the error and improve the overall answer of the neural network. This is basically going back to the simple linear classifier and how we adjusted its slope.

How can we adjust the link weights when the nodes are sophisticated as they are the weighted sum of weights and then input into the sigmoid function. Can we can use some fancy algebra ? We can't do algebra because maths is too hard and their are too many combinations of weights and function of functions.

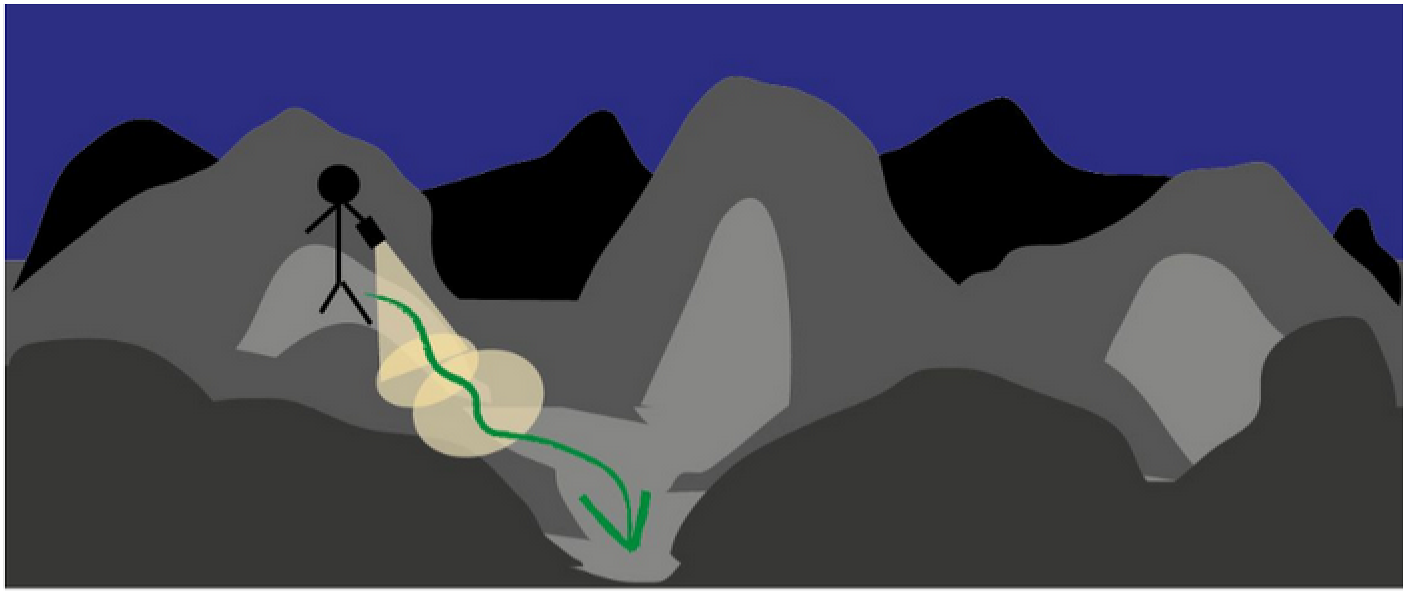
Think about a neural network with 3 layers and 3 neurons in each layer, like we had above. How would you tweak a weight for a link between the first input node and the second hidden node so that the third output node increased its output by, say, 0.5? Even if we did get lucky, the effect could be ruined by tweaking another weight to improve a different output node. You can see this isn't trivial at all.

Instead of being too clever ,lets try the brute force approach and try all the combinations. You can't be serious , its just not feasible to try all the combinations for neural networks. Only for a 3 layer 3 node network with -1 and +1 having 1000 possibilities like 0.934, -0.523, 0.231 etc has 18000 possible combinations. With 500 nodes in each layer the possibilities are 500 million so you see that brute force approach to try all the weights is not at all practicable at all.

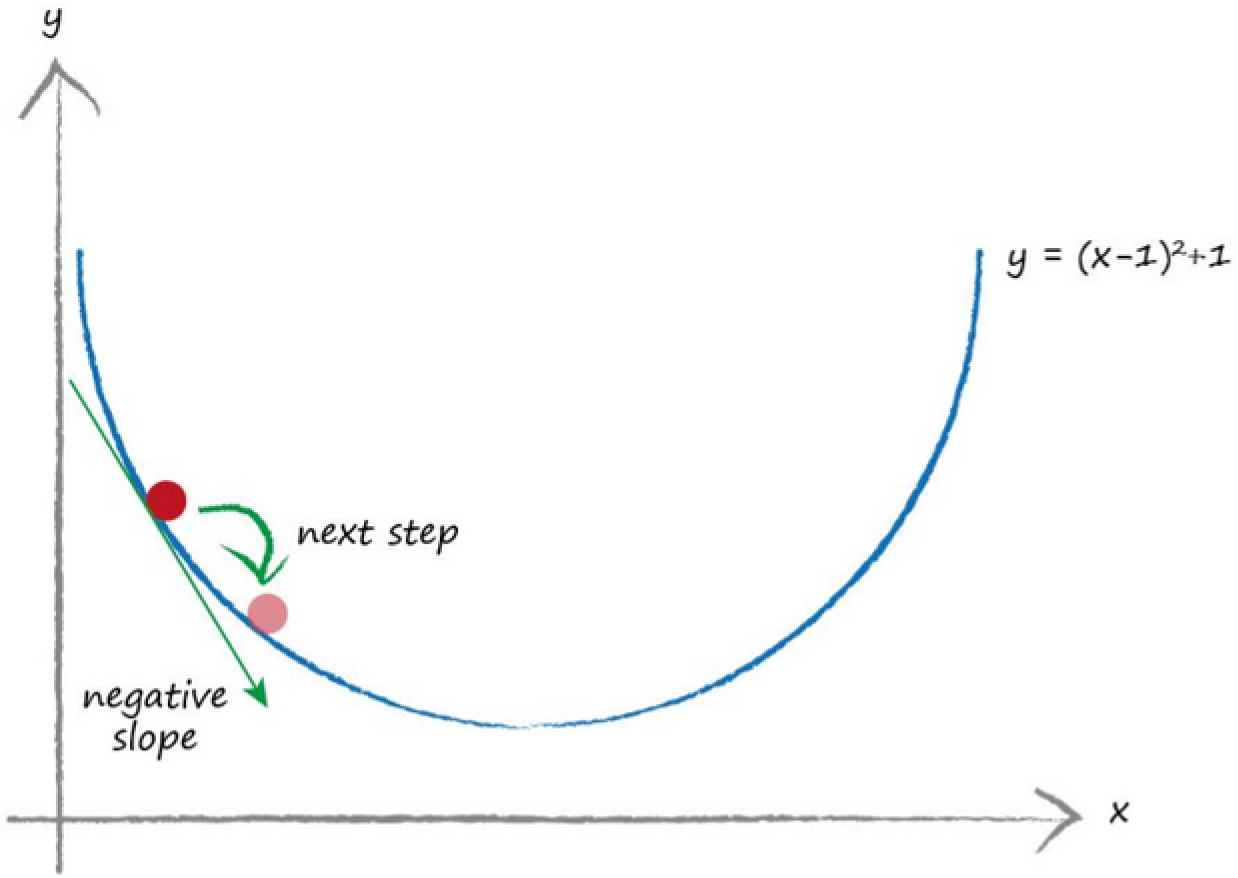
Then what can we do? Believe it or not you have got the answer. The first thing we have to do is to embrace **pessimism**.We have to be realistic, there are just too many combinations to check. and the data might also be flawed or we might not have many nodes to model the whole truth. so we got that out of the way that there's no perfect answer. Now that into account these limitation, we can find an approach that works out a flawed but better solution.

Let’s illustrate what we mean by this. Imagine a very complex landscape with peaks and troughs, and hills with treacherous bumps and gaps. It’s dark and you can’t see anything. You know you’re on the side of a hill and you need to get to the bottom. You don’t have an accurate map of the entire landscape. You do have a torch.

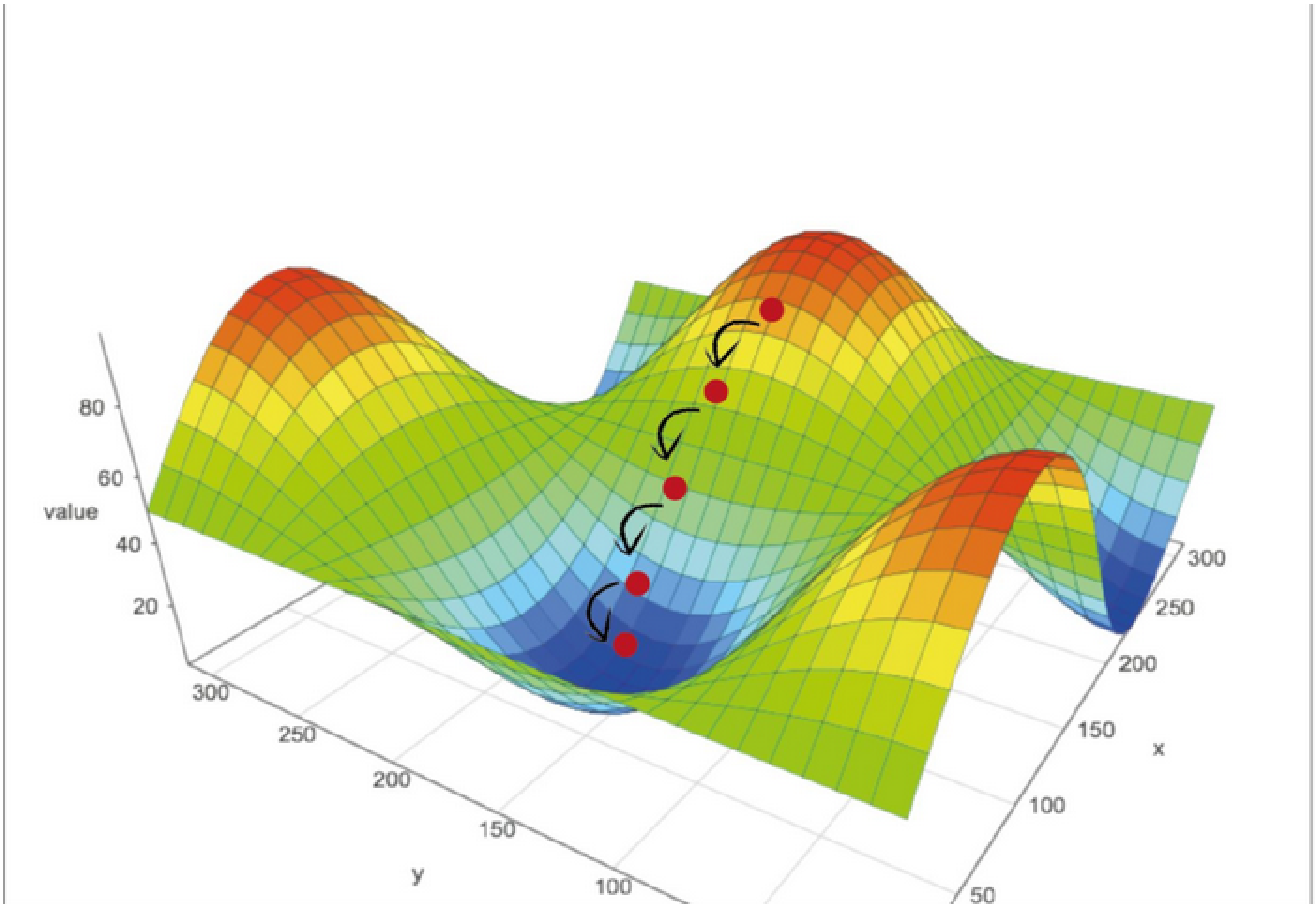
What do you do? You’ll probably use the torch to look at the area close to your feet. You can’t use it to see much further anyway, and certainly not the entire landscape. You can see which bit of earth seems to be going downhill and take small steps in that direction. In this way, you slowly work your way down the hill, step by step, without having a full map and without having worked out a journey beforehand.



The mathematical version of this approach is called **gradient descent.** You can see why after each step we are getting near to the target and descending the slope. The graph of y = $(x-1)^2+1$ looks something like this.



To do gradient descent we have to start somewhere. The graph shows our randomly chosen starting point. Like the hill climber, we look around the place we're standing and see which direction is downwards. The slope is marked on the graph and in this case is a negative gradient. We want to follow the downward direction so we move along x to the right. That is, we increase x a little. That's our hill climber's first step. You can see that we've improved our position and moved closer to the actual minimum.



In 3d it would look something similar.