Preparing Data: Random Initial Weights

In this section, we are going to consider how we might best prepare the initial random weights to give the training process a good chance of working.

The same argument applies here as with the inputs and outputs. We should avoid large initial weights because they cause large signals into an activation function, leading to the saturation we just talked about, and the reduced ability to learn better weights. We could choose initial weights randomly and uniformly from a range -1.0 to $+1.0$. That would be a much better idea than using a very large range, say -1000 to $+1000$.

Can we do better? Probably. Mathematicians and computer scientists have done the maths to work out a rule of thumb for setting the random initial weights given specific shapes of networks and with specific activation functions. That's a lot of "specifics"! Let's carry on anyway.

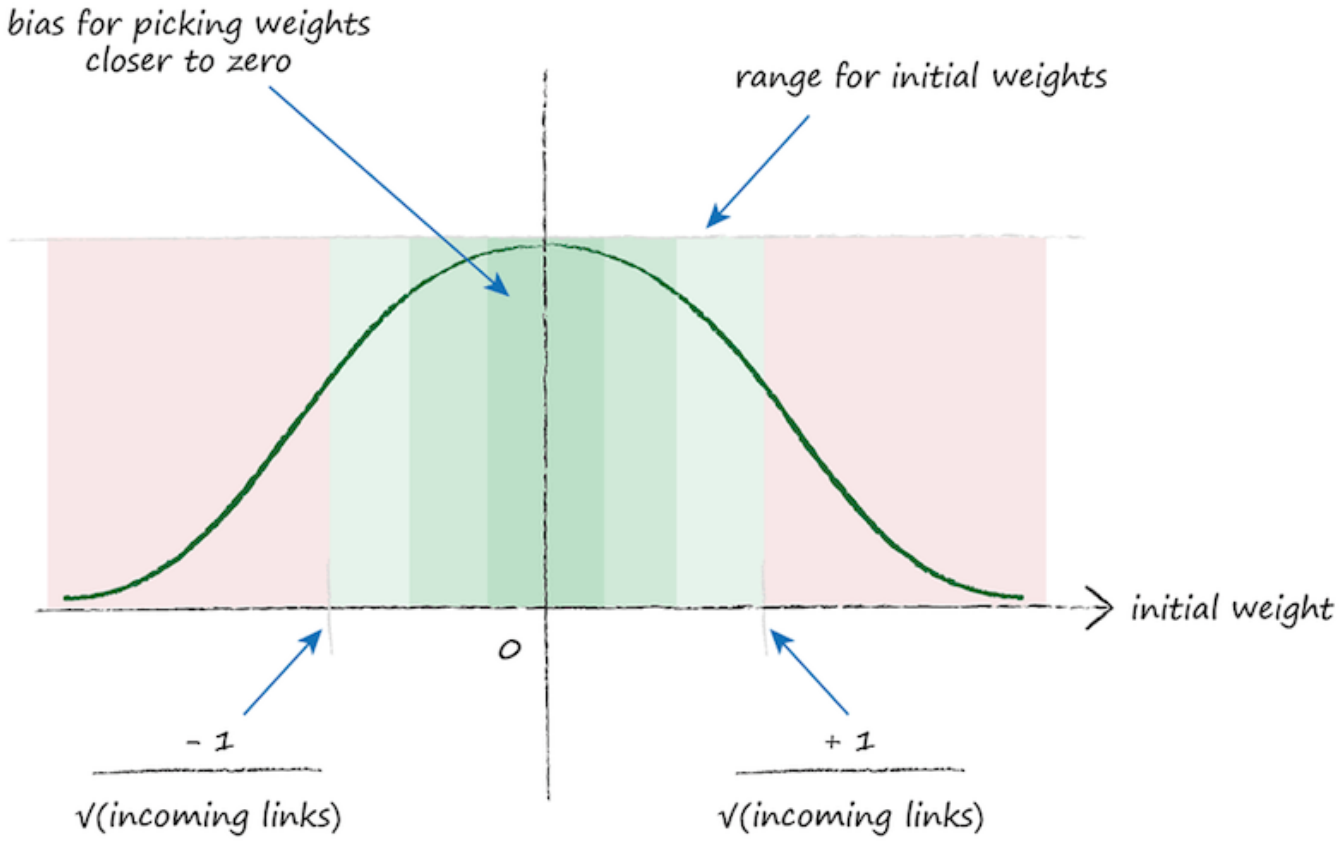
We won't go into the details of that working out but the core idea is that if we have many signals into a node, which we do in a neural network, and that these signals are already well behaved and not too large or crazily distributed, the weights should support keeping those signals well behaved as they are combined and the activation function applied. In other words, we don't want the weights to undermine the effort we put into carefully scaling the input signals.

The rule of thumb these mathematicians arrive at is that the weights are initialized randomly sampling from a range that is roughly the inverse of the square root of the number of links into a node. So if each node has 3 links into it, the initial weights should be in the range from -1/(√3) to +1/(√3), or ±0.577. If each node has 100 incoming links, the weights should be in the range from -1/(√100) to +1/(√100), or ±0.1.

Intuitively this makes sense. Some overly large initial weights would bias the activation function in a biased direction, and very large weights would *saturate* the activation functions. And the more links we have into a node, the

more signals are being added together. So a rule of thumb that reduces the weight range if there are more links makes sense.

If you are already familiar with the idea of sampling from probability distributions, this rule of thumb is actually about sampling from a normal distribution with mean zero and a standard deviation which is the inverse of the square root of the number of links into a node. But let's not worry too much about getting this precisely right because that rule of thumb assumes quite a few things which may not be true, such as an activation function like the alternative $tanh()$ and a specific distribution of the input signals. The following diagram summarises visually both the simple approach and the more sophisticated approach with a normal distribution.



Whatever you do, don't set the initial weights the same constant value, especially not zero. That would be bad! It would be bad because each node in the network would receive the same signal value, and the output out of each output node would be the same. If we then proceeded to update the weights in the network by back-propagating the error, the error would have to be divided equally. You'll remember the error is split in proportion to the weights. That would lead to equal weight updates leading again to another set of equal valued weights. This symmetry is bad because if the properly trained network should have unequal weights (extremely likely for almost all problems) then you'd never get there.

Zero weights are even worse because they kill the input signal. The weight update function, which depends on the incoming signals, is zeroed. That kills the ability to update the weights completely. There are many other things you can do to refine how you prepare your input data, how you set your weights, and how you organize your desired outputs. For this guide, the above ideas are both easy enough to understand and also have a decent effect, so we'll stop there.