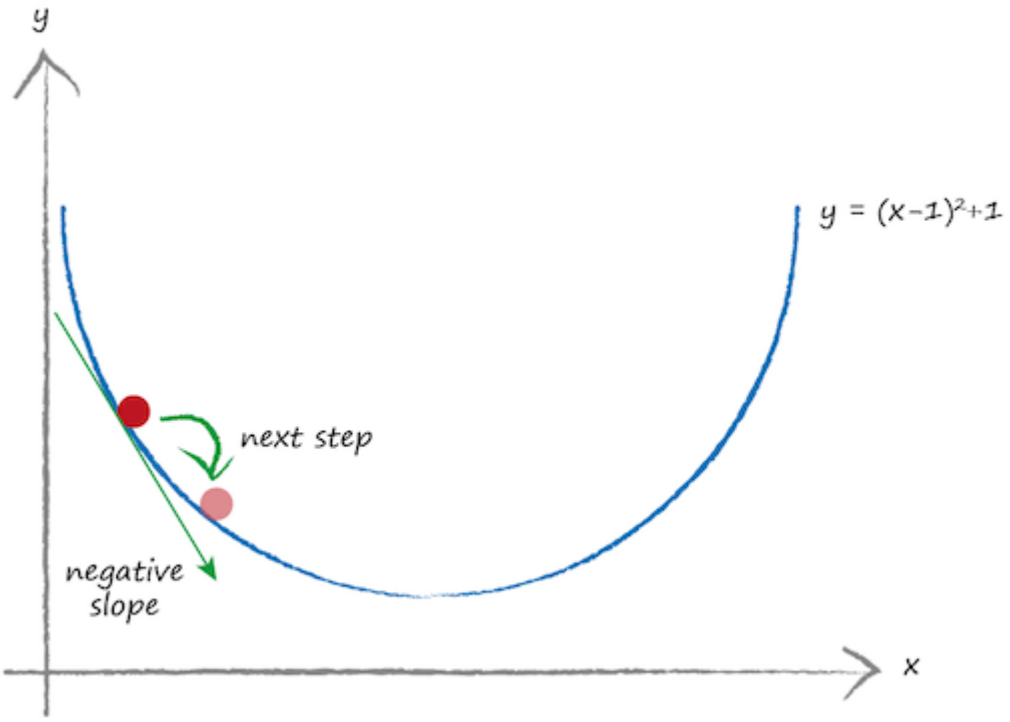
Understanding the Gradient Descent Algorithm

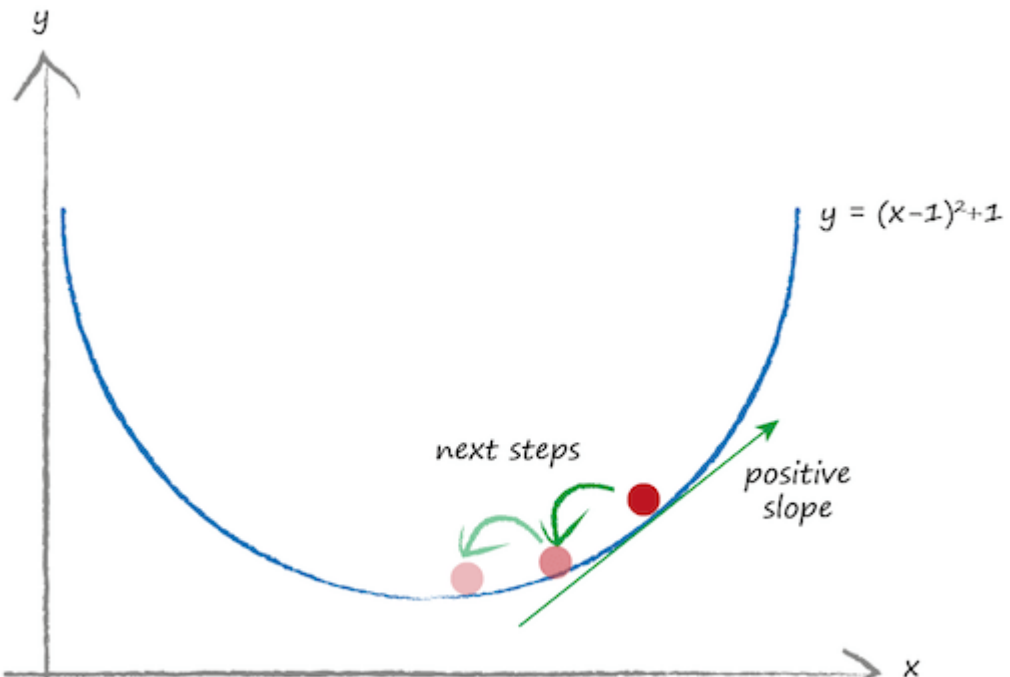
This lesson is a brief introduction to Gradient Descent and how it works. We will also discuss whether how it is an efficient approach to minimize the error in our Neural Network?

Now imagine that complex landscape is a mathematical function. What this gradient descent method gives us is an ability to find the minimum without actually having to understand that complex function enough to work it out mathematically. If a function is so difficult that we can't easily find the minimum using algebra, we can use this method instead. Sure it might not give us the exact answer because we are using steps to approach an answer, improving our position bit by bit.

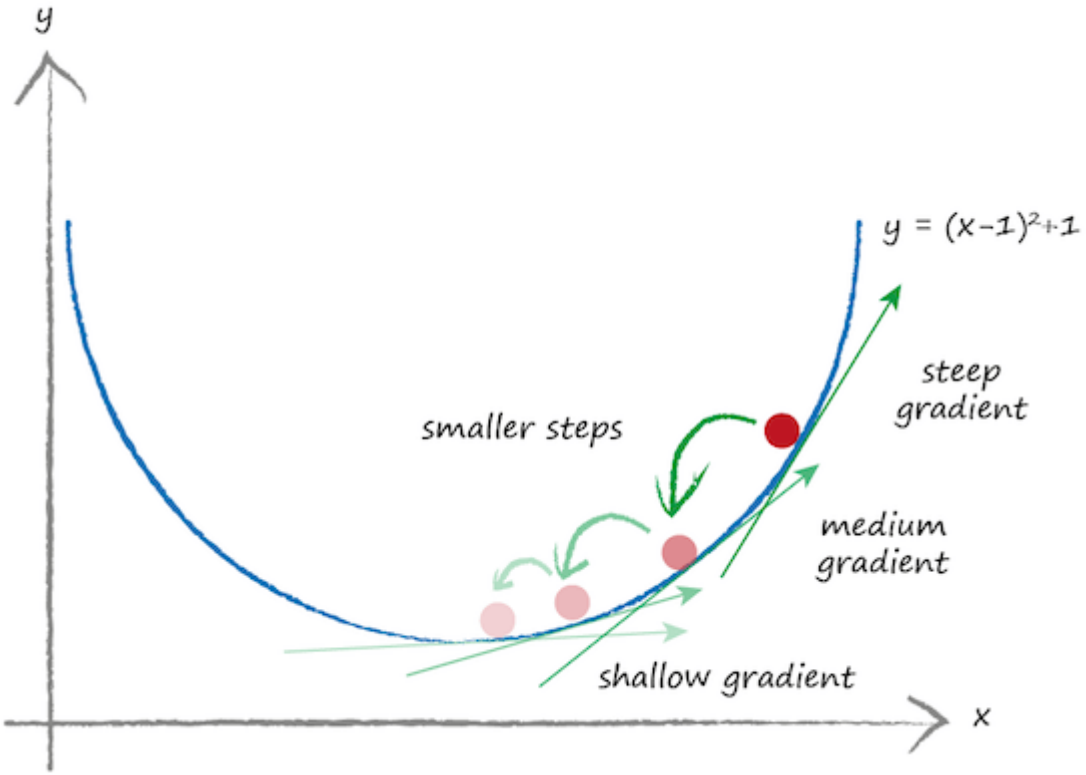
What's the link between this really cool gradient descent method and neural networks? Well, if the difficult, complex function is the error of the network, then going downhill to find the minimum means we are minimizing the error. We are improving the network's output. That's what we want! Let's look at this gradient descent idea with a super simple example so we can understand it properly. The following graph shows a simple function $y = (x - 1)^2 + 1$. If this were a function where y was the error, we would want to find the x which minimizes it. For a moment, pretend this wasn't an easy function but instead a complex difficult one.



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 have improved our position and moved closer to the actual minimum. Let's imagine we started somewhere else, as shown in the next graph.

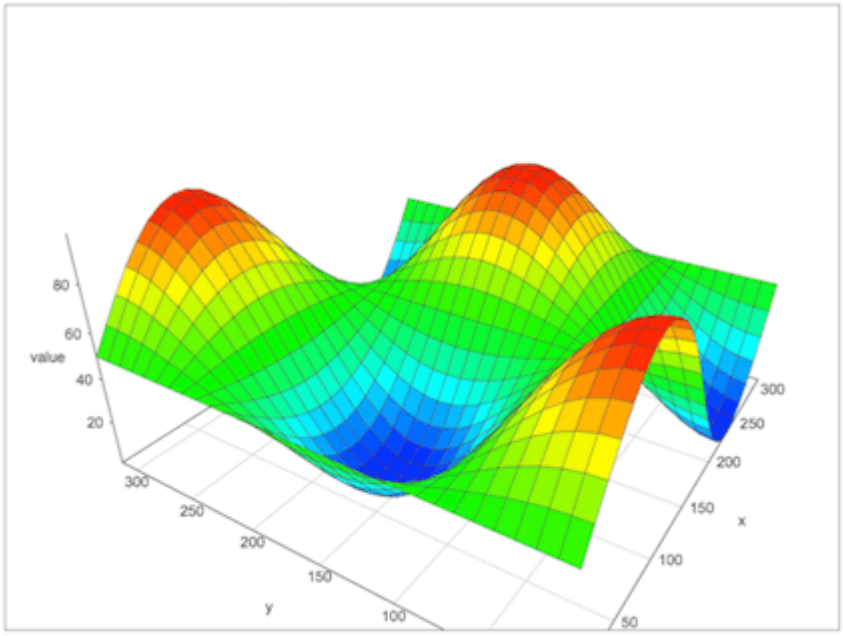


This time, the slope beneath our feet is positive, so we move to the left. That is, we decrease x a little. Again you can see we've improved our position by moving closer to the actual true minimum. We can keep doing this until our improvements are so small that we can be satisfied that we've arrived at the minimum. A necessary refinement is to change the size of the steps we take to avoid overshooting the minimum and forever bouncing around it. You can imagine that if we've arrived 0.5 meters from the true minimum but can only take 2-meter steps, then we're going to keep missing the minimum as every step we take in the direction of the minimum will overshoot. If we moderate the step size, so it is proportionate to the size of the gradient then when we are close, we'll take smaller steps. This assumes that as we get closer to a minimum, the slope does indeed get shallower. That's not a bad assumption at all for most smooth continuous functions. It wouldn't be a good assumption for crazy zig-zag functions with jumps and gaps, which mathematicians call *discontinuities*. The following illustrates this idea of moderating step size as the function gradient gets smaller, which is a good indicator of how close we are to a minimum.



By the way, did you notice that we increase x in the opposite direction to the gradient? A positive gradient means we reduce x. A negative gradient means we increase x. The graphs make this clear, but it is easy to forget and get it the

wrong way around. When we did this gradient descent, we didn't work out the true minimum using algebra because we pretended the function $y = (x - 1)^2 + 1$ was too complex and difficult. Even if we couldn't work out the slope exactly using mathematical precision, we could estimate it, and you can see this would still work quite well in moving us in the correct general direction. This method really shines when we have functions of many parameters. So not just *y* depending on *x*, but maybe y depending on *a*, *b*, *c*, *d*, *e* and *f*. Remember the output function, and therefore the error function, of a neural network depends on many many weight parameters. Often hundreds of them! The following again illustrates gradient descent but with a slightly more complex function that depends on two parameters. This can be represented in three dimensions with the height representing the value of the function.



You may be looking at that 3-dimensional surface and wondering whether gradient descent ends up in that other valley also shown at the right. In fact, thinking more generally, doesn't gradient descent sometimes get stuck in the wrong valley, because some complex functions will have many valleys? What's the wrong valley? It's a valley which isn't the lowest. The answer to this is yes, that can happen. To avoid ending up in the wrong valley, or function *minimum*, we train neural networks several times starting from different points on the hill to ensure we don't always end up in the wrong valley. Different starting points means choosing different starting parameters, and in the case of neural networks, this means choosing different starting link weights. The following illustrates three different goes at gradient descent, with one of them ending up trapped in the wrong valley.

with one of them ending up trapped in the wrong valley.