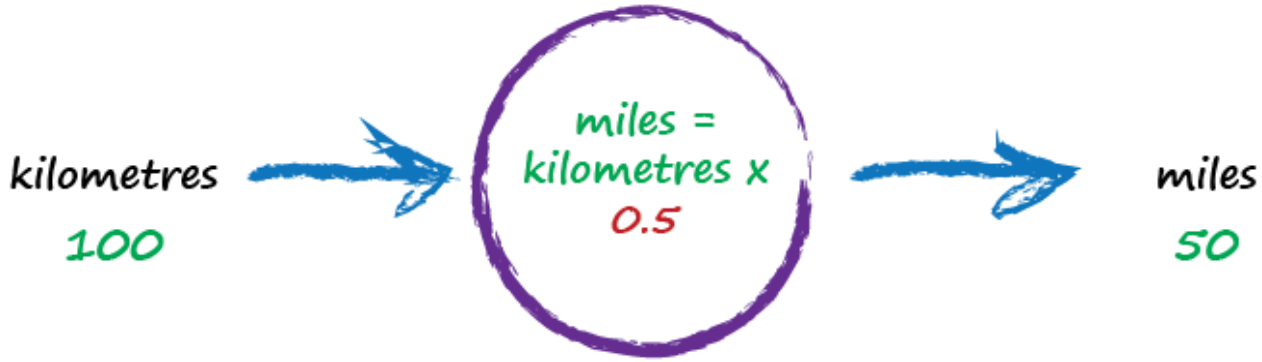
Estimating the Constant "c" Iteratively

Use "Hit and Trial" technique to estimate a constant and refine the model by adjusting other parameters based on the original output.

In the previous lesson, we tried to come up with a value which can work as a constant in the distance to miles formula. So what should we do to work out that missing constant “c”? Let’s just pluck a value at random and give it a go! Let’s try $c = 0.5$ and see what happens.



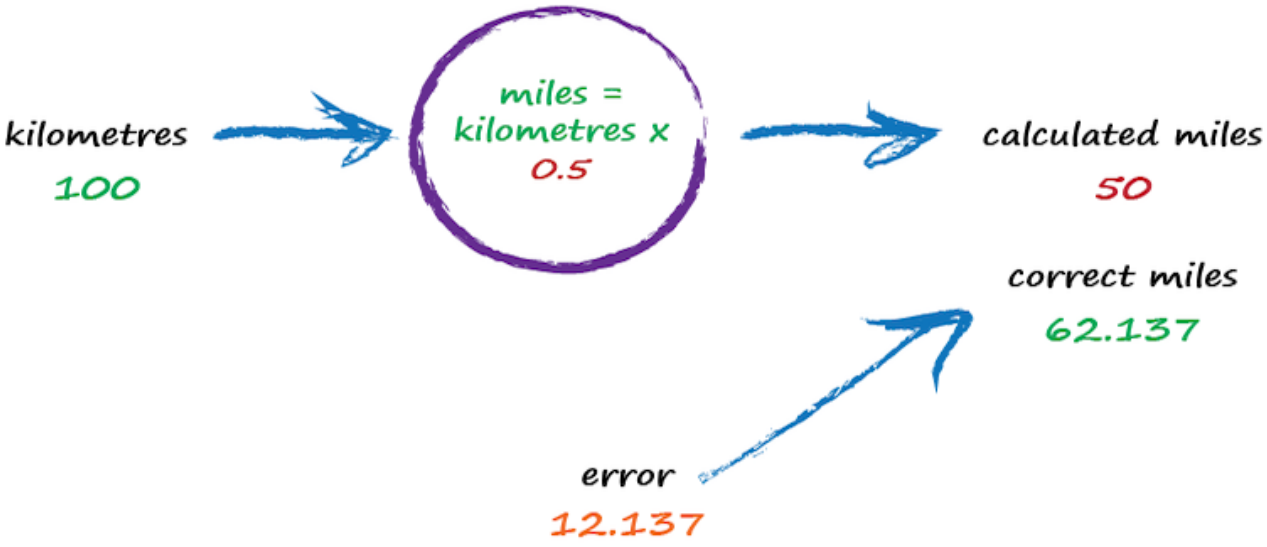
Here we have $miles = kilometers * c$, where kilometers is 100 and c is our current guess at 0.5. That gives 50 miles. Okay. That’s not bad at all given we chose $c = 0.5$ at random! But we know it’s not exactly right because our truth example number 2 tells us the answer should be 62.137.

We are wrong by 12.137. That’s the error, the difference between our calculated answer and the actual truth from our list of examples. That is,

$$error = truth - calculated$$

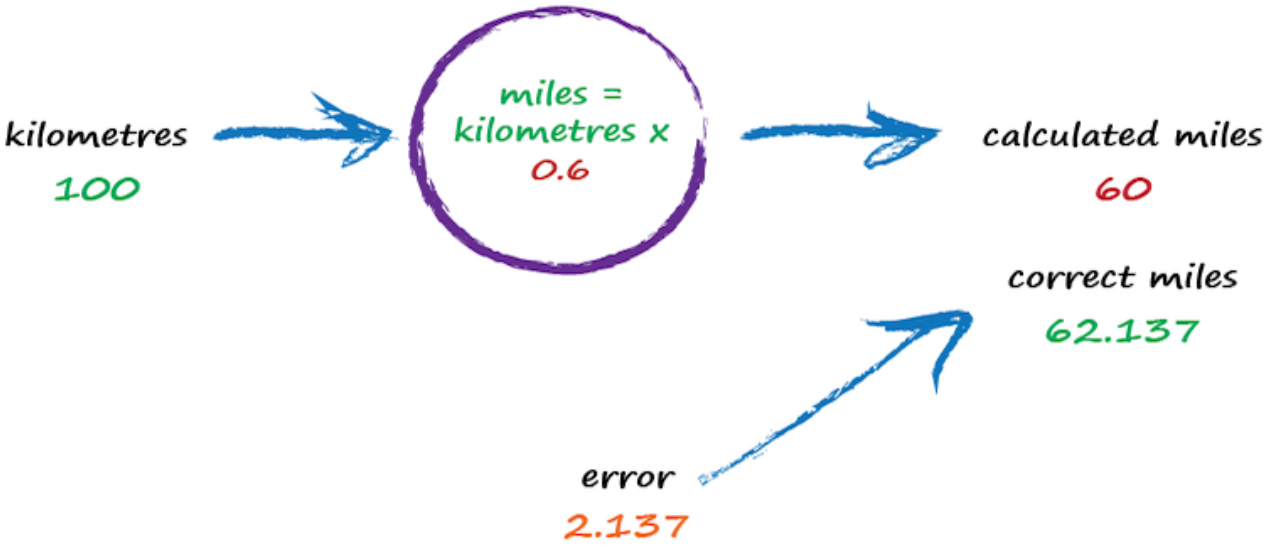
$$= 62.137 - 50$$

$$= 12.137$$



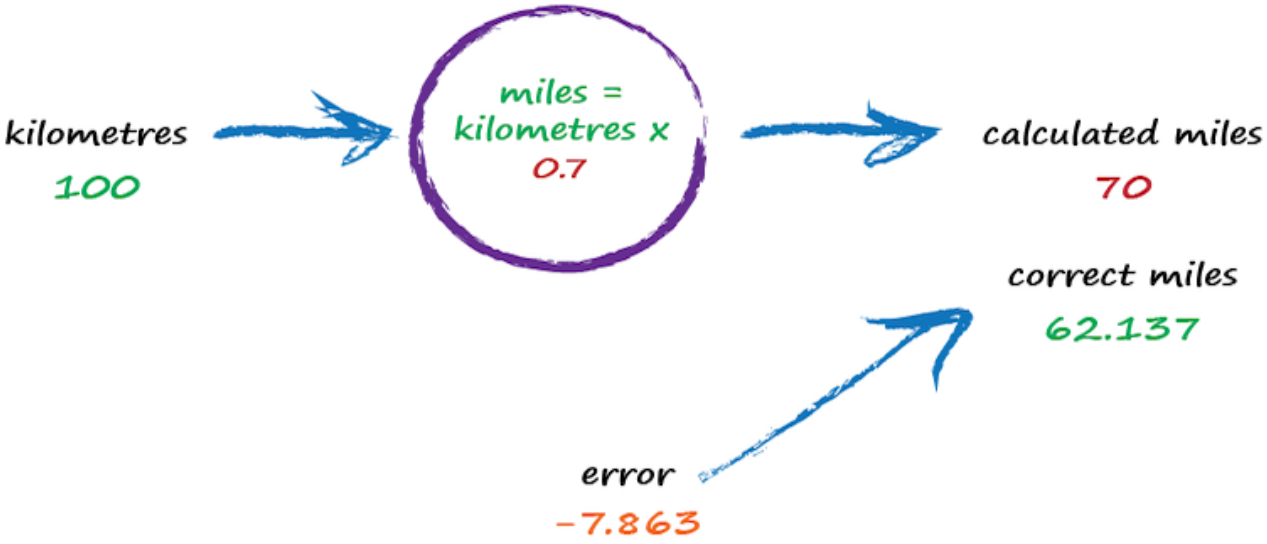
So what next? We know we are wrong, and by how much. Instead of being a reason to despair, we use this error to guide a second, better, guess at c. Look at that error again. We were short by 12.137. Because the formula for converting kilometers to miles is linear, $miles = kilometersxc$, we know that increasing c will increase the output.

Let's nudge c up from 0.5 to 0.6 and see what happens. With c now set to 0.6, we get $miles = kilometres * c = 100 * 0.6 = 60$. That's better than the previous answer of 50. We are clearly making progress! Now the error is a much smaller 2.137. It might even be an error we're happy to live with.



The important point here is that we used the error to guide how we nudged the value of c. We wanted to increase the output from 50 so we increased c a little bit.

Rather than try to use algebra to work out the exact amount c needs to change, let's continue with this approach of refining c. If you're not convinced and think it's easy enough to work out the exact answer, remember that many more interesting problems won't have simple mathematical formulae relating the output and input. That's why we need more sophisticated methods - like neural networks. Let's do this again. The output of 60 is still too small. Let's nudge the value of c up again from 0.6 to 0.7.



Oh no! We have gone too far and overshot the known correct answer. Our previous error was 2.137, but now it's -7.863. The minus sign simply says we overshot rather than undershot, remember the error is (correct value – calculated value).

Ok so $c = 0.6$ was way better than $c = 0.7$. We could be happy with the small error from $c = 0.6$ and end this exercise now. But let's go on for just a bit longer. Why don't we nudge c up by just a tiny amount, from 0.6 to 0.61?

That's much much better than before. We have an output value of 61 which is only wrong by 1.137 from the correct 62.137. So that last effort taught us that we should moderate how much we nudge the value of c. If the outputs are getting close to the correct answer — that is, the error is getting smaller — then don't nudge the changeable bit so much. That way we avoid overshooting the right value as we did earlier.

Again without getting too distracted by exact ways of working out c, and to remain focused on this idea of successively refining it, we could suggest that the correction is a fraction of the error. That's intuitively right — a big error means a bigger correction is needed, and a tiny error means we need the teeniest of nudges to c.

What we have just done, believe it or not, is walked through the very core process of learning in a neural network — we have trained the machine to get better and better at giving the right answer. It is worth pausing to reflect on that — we have not solved a problem exactly in one step like we often do in school maths or science problems. Instead, we have taken a very different approach by trying an answer and improving it repeatedly. Some use the term *iterative* and it means repeatedly improving an answer bit by bit.