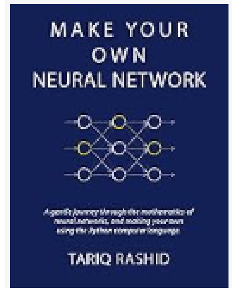
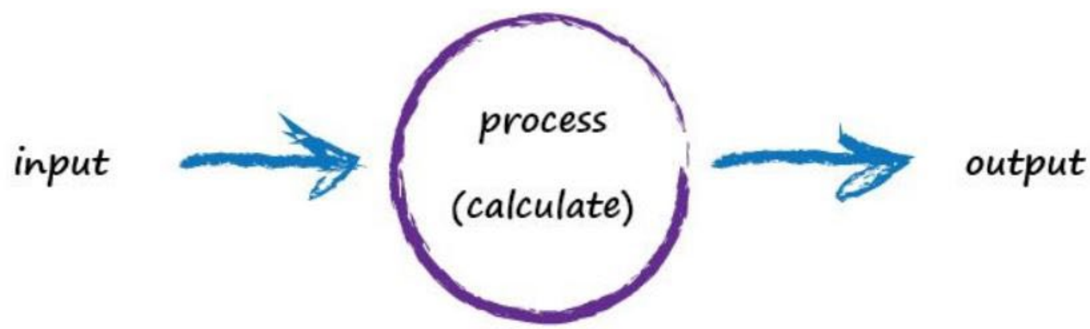
<u>“SIMPLE” EXPLANATION: HOW A NEURAL NET LEARNS TO SOLVE A “REGRESSION” PROBLEM</u>

(From “Make your Own Neural Network”, T. Rashid, 2016)



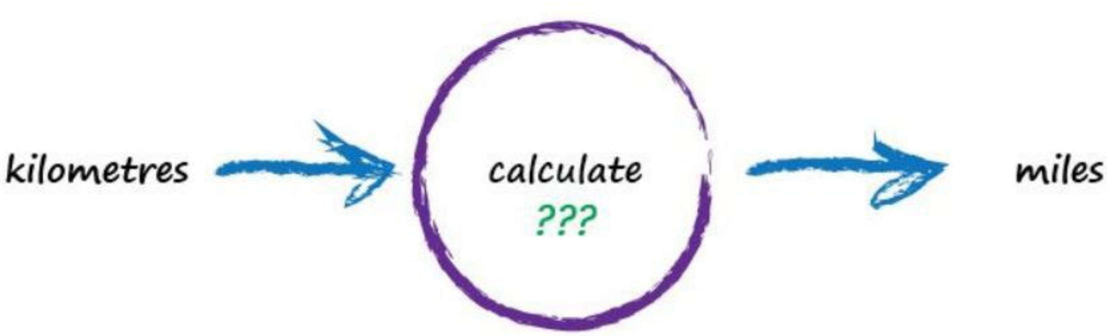
<u>Humans:</u>

<u>(“Traditional”) Computer:</u>



A computer (traditionally) must be given the “formula” (to calculate), and then, when an input is provided, an output is generated. BUT, NEURAL NETWORKS (<u>implemented in computers</u>) may be able to “learn” to provide answers WITHOUT EXPLICITELY PROVIDING THE “FORMULA”:

EXAMPLE - Determine how many miles correspond to a certain number of kms :

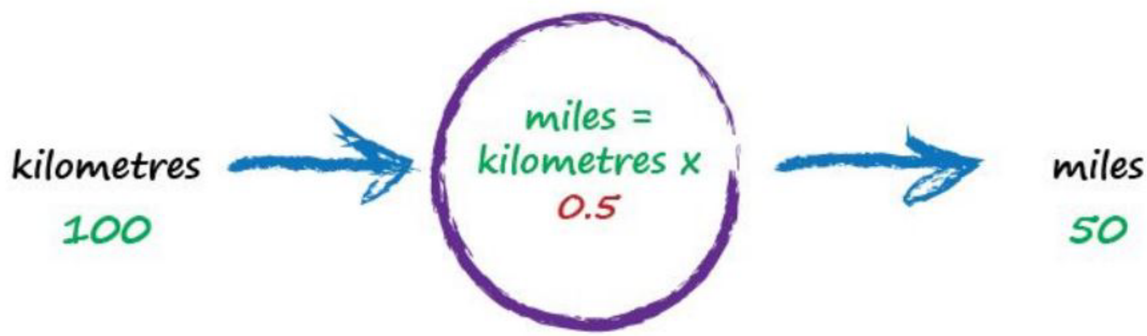


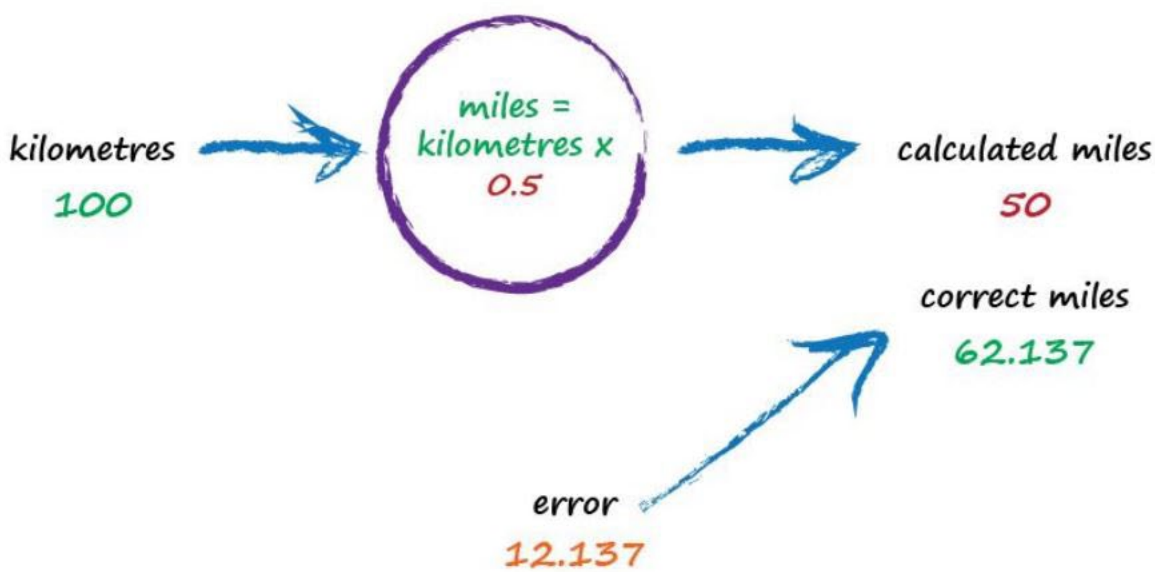
INSTEAD OF "DIRECTLY PROGRAMMING" THE "FORMULA" in the computer, MULTIPLE EXAMPLES (for which correct answers – called "targets" – are known) will be "presented to the NN. We realize that the "format" of the solution is:

Kilometers x " c " = miles

Truth Example	Kilometres	Miles
1	0	0
2	100	62.137

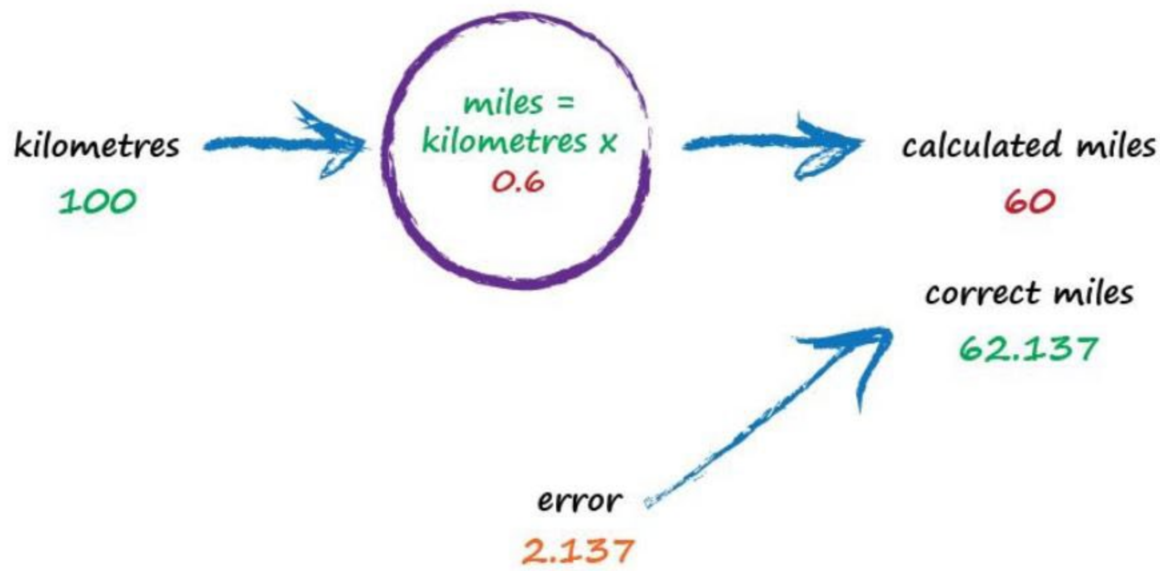
What should we do to work out that missing constant c? Let's just pluck a value at **random** and give it a go! Let's try c = 0.5 and see what happens.





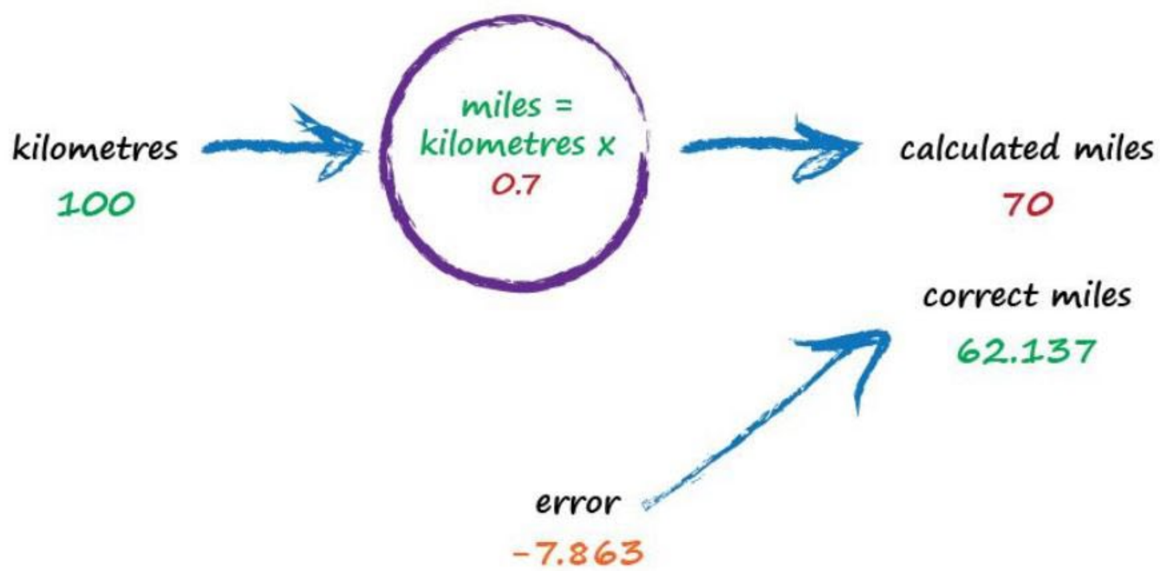
So what next? We know we're wrong, and by how much. Instead of being a reason to despair, we use this error to guide a second, better, guess at **c**.

Because the “error” = CorrectMiles – Calculated Miles, came out to be + 12.137 , we realize we need to “nudge” the value of “ c “ up. For example, with c = 0.6:



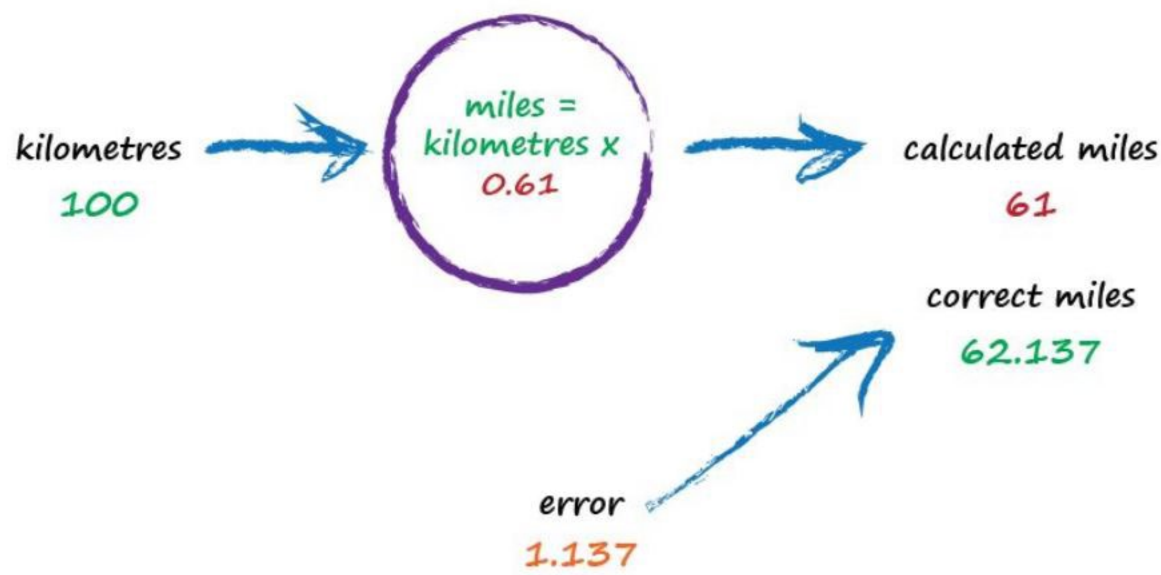
The important point here is that we used the error to guide how we nudged the value of c. We wanted to increase the output from 50 so we increased **c** a little bit.

We are still “short”. Should we increase c further? What would happen with c = 0.7?



Oh no! We’ve 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.



We found that c = 0.61 gets us *"very close"* to the correct conversion values. That is, we managed to, iteratively, arrive at the value of the "model parameter" (c) that "minimizes" the error (although we stopped the process without reaching an error of "0". Such complete elimination of the error may or may not be possible).

##