Refining the Parameters of Training Classifier

In this lesson, we will learn how to refine our parameters by updating their values based on the error value. To do this, we first need to identify a relationship between A and E.

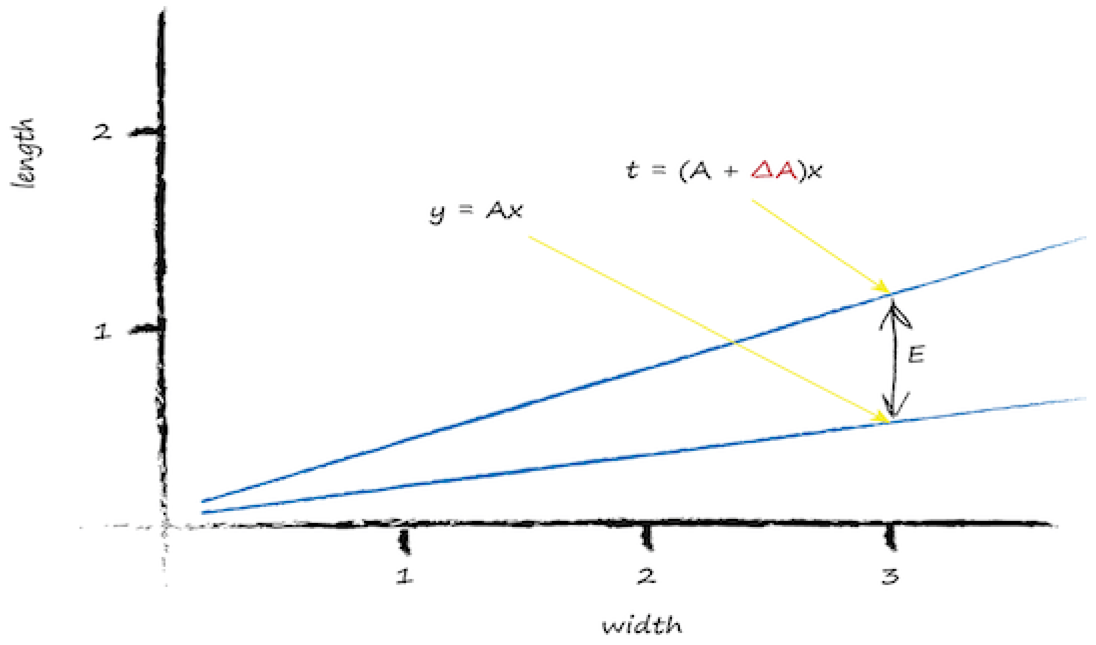
Continuing the discussing from the previous lesson, what do we do with this **E** to guide us to a better-refined parameter **A**? That's the important question. Let's take a step back from this task and think again. We want to use the error in *y*, which we call **E**, to inform the required change in parameter **A**. To do this, we need to know how the two are related. How is **A** related to **E**? If we can know this, then we can understand how changing one affects the other. Let's start with the linear function for the classifier:

$$y = Ax$$

We know that for initial guesses of A this gives the wrong answer for *y*, which should be the value given by the training data. Let's call the correct desired value, *t* for the target value. To get that value *t*, we need to adjust **A** by a small amount. Mathematicians use the delta symbol Δ to mean "a small change in". Let's write that out:

$t = (A + \Delta A)x$

Let's picture this to make it easier to understand. You can see the new slope (A + ΔA).



Remember the error **E** was the difference between the correct value and the one we calculate based on our current guess for **A**. That is, **E** was $t - y$. Let's write that out to make it clear:

t - y = (A + ΔA)x - Ax

Expanding out the terms and simplifying:

E = t - y = Ax + (ΔA)x - Ax E = (ΔA)x

That's remarkable! The error **E** is related to **ΔA** is a very simple way. It's so simple that I thought it must be wrong — but it was indeed correct. Anyway, this simple relationship makes our job much easier. It's easy to get lost or distracted by that algebra. Let's remind ourselves of what we wanted to get out of all this, in plain English. We wanted to know how much to adjust **A** to improve the slope of the line, so it is a better classifier, being informed by the error **E**. To do this, we simply re-arrange that last equation to put **ΔA** on its own:

ΔA = E / x

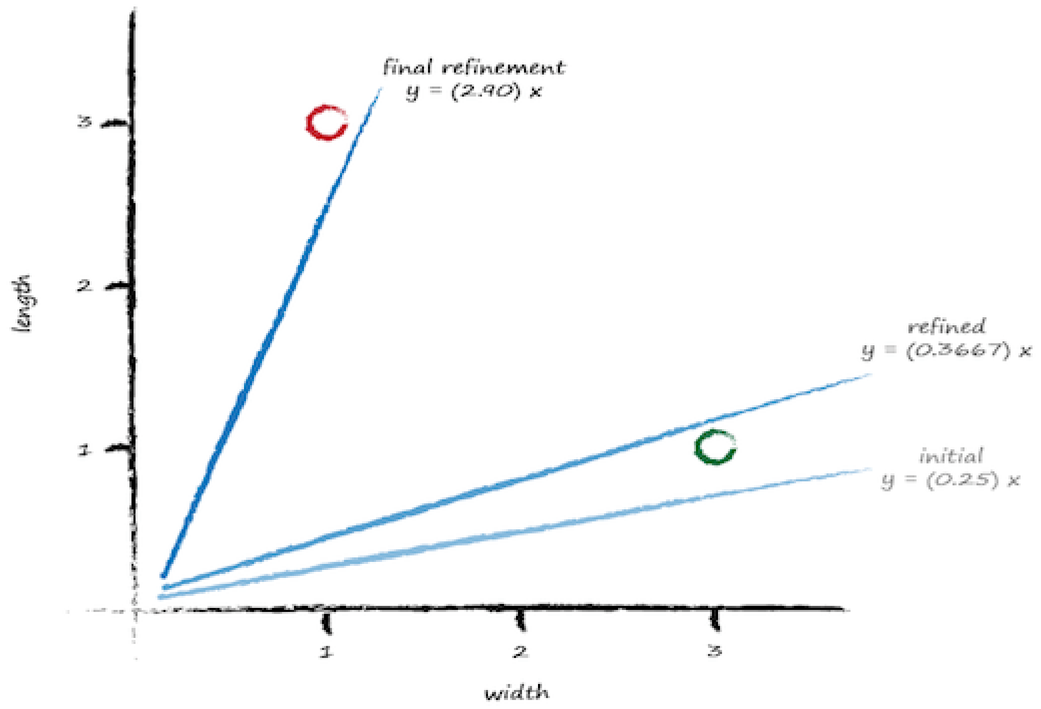
That's it! That's the magic expression we've been looking for. We can use the error **E** to refine the slope **A** of the classifying line by an amount **ΔA**. Let's do it! Let's update that initial slope. The error was 0.35 and the *x* was 3.0. That gives ΔA = E / x as $0.35/3.0 = 0.1167$. That means we need to change the current $A = 0.25$ by 0.1167. That means the new improved value for **A** is (**A**

+ Δ**A**) which is $0.25 + 0.1167 = 0.3667$. As it happens, the calculated value of *y* with this new **A** is 1.1 as you'd expect — it's the desired target value.

Phew! We did it! All that work and we have a method for refining that parameter **A**, informed by the current error. Let's press on. Now we're done with one training example, let's learn from the next one. Here we have a known true pairing of $x = 1.0$ and $y = 3.0$. Let's see what happens when we put $x = 1.0$ into the linear function which is now using the updated $A = 0.3667$. We get $y = 0.3667 * 1.0 = 0.3667$. That's not very close to the training example with $y = 3.0$ at all.

Using the same reasoning as before that we want the line to not cross the training data but instead be just above or below it, we can set the desired target value at 2.9. This way the training example of a caterpillar is just above the line, not on it. The error **E** is $(2.9 - 0.3667) = 2.5333$. That's a bigger error than before, but if you think about it, all we've had so far for the linear function to learn from is a single training example, which clearly biases the line towards that single example.

Let's update the **A** again, just like we did before. The Δ**A** is E/x which is $2.5333/1.0 = 2.5333$. That means the even newer **A** is $0.3667 + 2.5333 = 2.9$. That means for $x = 1.0$ the function gives 2.9 as the answer, which is what the desired value was. That's a fair amount of working out so let's pause again and visualize what we've done. The following plot shows the initial line, the line updated after learning from the first training example, and the final line after learning from the second training example.



Wait! What's happened! Looking at that plot, we don't seem to have improved the slope in the way we had hoped. It hasn't divided neatly the region between ladybirds and caterpillars. Well, we got what we asked for. The line updates to give each desired value for *y*.

What's wrong with that? Well, if we keep doing this, updating for each training data example, all we get is that the final update simply matches the last training example closely. We might as well have not bothered with all previous training examples. In effect, we are throwing away any learning that previous training examples might give us and just learn from the last one. How do we fix this? Let's find out in the next lesson!