An Explanation of Machine Learning, And Multivariable Regression

There's a lot of data in the real world, and it's often used by machine learning algorithms, to build all sorts of different prediction models. In this project, I'll be using machine learning, to build a "Multivariable Linear Regression" (MLR) model, from scratch, one of the more basic and popular models. I'll also be explaining what's going on, as I go.

Let's get started.

First, let's say we had some data. Some simple data, just to make the explanation easier to understand.

The Data		
x_1	x_2	y_a
2	31	240
5	56	430
9	72	562

As we can see from the data, we've only got two independent variables/features (x_1, x_2), and then there's our dependent variable, y_a.

So, there's some sort of mathematical relationship/model between our two independent variables, and the dependent variable. This model could be ANYTHING, but we'll just assume it's linear (since we're using MLR here), and thus the model looks like this:

$$y_a \;=\; ?_1\, x_1 \;+\; ?_2\, x_2 \;+\; ?_3$$

So, we want to figure out what the question marks are, but how do we do that? Well we can use machine learning! First, we'll randomly guess what the question marks are (conventionally called weights/bias, or coefficients/constants, we'll call them weights/bias from now on):

Let's say we guess…

$$w_1 = 1, \qquad w_2 = 2, \qquad b = 3$$

That means, that we think the model that fits the data, is this:

$$y = 1x_1 + 2x_2 + 3$$

Okay well, is this model accurate? How do we tell? Well we can simply see what it predicts for each instance of data we have, and compare it to what the actual value is. So:

$$prediction_1 = y_{p1} = 1 * 2 + 2 * 31 + 3 = 67$$
$$prediction_2 = y_{p2} = 1 * 5 + 2 * 56 + 3 = 120$$
$$prediction_3 = y_{p3} = 1 * 9 + 2 * 72 + 3 = 156$$

Now, let’s see how far off we were:

$$diff_1 = y_{p1} - y_{a1} = 67 - 240 = -173$$
$$diff_2 = y_{p2} - y_{a2} = 120 - 430 = -310$$
$$diff_3 = y_{p3} - y_{a3} = 156 - 562 = -406$$

Unfortunately, it looks like our prediction model came up quite short in all cases. Now, this was only for 3 instances of data, if we had 300 instances, we’d have 300 differences to look at (that’s a lot of scrolling), so we’ll just add up all the differences and divide by the number of differences, to see how off we were on average.

However! Unfortunately we can’t just add up all the differences directly, because there is the possibility (not in this case, but in others), that sometimes the model predicts ABOVE the correct answer, thus leading to a POSITIVE difference, and sometimes that same model will predict BELOW the correct answer, leading to a NEGATIVE difference.

If we add together a bunch of positive and and negative differences, they’ll cancel each other out, and our model will deceptively look FAR more accurate, than it actually is.

So what do we do? Well we could add together the ABSOLUTE VALUE, of all differences, so we’d have this:

$$average\ difference = \frac{|\ y_{p1} - y_{a1}\ |\ +\ |\ y_{p2} - y_{a2}\ |\ +\ |\ y_{p3} - y_{a3}\ |}{3}$$

And this seems to work nicely, so we’ve got a good way to figure out/summarise how accurate a model is on average, and thus we could just call this the cost.

Unfortunately, however, as we'll soon see, we're gonna need to figure out the gradient of cost, and if our cost has absolute values in it, then it won't be differentiable/smooth, at any point where an absolute value equals 0, (think of how the graph of $y = |x|$, is sharp at $x = 0$) so the gradient of cost will break down at those points.

So, we can't use the absolute value… well then what else could we use? Well squaring also always results in a positive, AND it's differentiable/smooth everywhere, so we can use that! Perfect!

$$cost = \frac{\left(y_{p1} - y_{a1}\right)^2 + \left(y_{p2} - y_{a2}\right)^2 + \left(y_{p3} - y_{a3}\right)^2}{3}$$

Now, the above is what cost looks like in our particular case, where there's only 3 instances of data (and thus only 3 differences). However, normally, there's far more instances of data, so cost is expressed in an equivalent, but somewhat more complex looking form, shown below:

(n represents the number of instances)

$$cost = \frac{\sum_{i=1}^{n} \left(y_{pi} - y_{ai}\right)^2}{2 * n}$$

This expression uses sigma notation (Σ symbol), to sum up all differences, no matter how many there are. Additionally, there is a new factor of $1/2$, presented as a 2 in the denominator, which is the only actual difference to our original cost expression.

(The factor of $1/2$ is put into cost, to cancel out the factor of 2, that will pop up in every partial derivative of cost later on, when calculating the gradient)

So, now that we've properly defined cost (it's literally just the average SQUARED difference, hence the name: "mean squared error", with a factor of $1/2$ attached), we can see how bad our current model is:

$$cost = \frac{\sum_{i=1}^{3} \left(y_{pi} - y_{ai}\right)^2}{2 * 3} = \frac{(-173)^2 + (-310)^2 + (-406)^2}{6} = 48{,}477.5$$

Okay, so, the current weights/bias of the model, have a cost of: 48,477.5 (which is just the average-squared-difference, halved).

That seems pretty bad, so we should change the weights/bias, so that cost GOES DOWN. But how do we figure out what to change the weights/bias by, so that cost does go down (and not up)?

Well, we need to know HOW cost changes, as a weight/bias changes. So we can simply take the DERIVATIVE of cost, with respect to that weight/bias, and the resulting expression, will tell us EXACTLY how cost changes, as that weight/bias changes.

So, for example, figuring out exactly how cost changes, as w_1 changes:

(The math gets a bit hairy, but is manageable if familiar with calculus:)

$$\frac{dC}{dw_1} \quad = \quad \frac{d\,\frac{\sum_{i=1}^{n}\left(y_{pi}-y_{ai}\right)^2}{2*n}}{dw_1}$$

We can simply pull the constant $\frac{1}{2n}$ right out:

$$\frac{dC}{dw_1} \quad = \quad \frac{1}{2n}*\frac{d\sum_{i=1}^{n}\left(y_{pi}-y_{ai}\right)^2}{dw_1}$$

We can also pull the summation operation out of the derivative (this comes from the sum rule of derivatives: the derivative of a sum of functions, is equal to the sum of their derivatives):

$$\frac{dC}{dw_1} \quad = \quad \frac{1}{2n}*\sum_{i=1}^{n}\frac{d\left(y_{pi}-y_{ai}\right)^2}{dw_1}$$

Now, all we have to do is simplify the derivative within the summation, and to do that, we need to substitute y_{pi} for an expression where we can acutally see w_1:

(m represents the number of independent variables)

$$y_{pi} \quad = \quad w_1x_{1i}+w_2x_{2i}+w_3x_{3i}+\cdots+b \quad = \quad \left(\sum_{j=1}^{m}w_jx_{ji}\right)+b$$

Thus that derivative within the summation can be rewritten as:

$$\frac{d\left(y_{pi}-y_{ai}\right)^2}{dw_1} \quad = \quad \frac{d\left(\left(\sum_{j=1}^{m}w_jx_{ji}\right)+b-y_{ai}\right)^2}{dw_1}$$

The above expression looks pretty messy, so we can simplify it by using a dummy variable:

$$h = \left(\sum_{j=2}^{m} w_j x_{ji}\right) + b - y_{ai}$$

So, our derivative will now look like this:

$$\frac{d\left(y_{pi} - y_{ai}\right)^2}{dw_1} \quad = \quad \frac{d(w_1 x_{1i} + h)^2}{dw_1}$$

We can just expand this out and figure out the derivative quite easily:

$$\frac{d(w_1 x_{1i} + h)^2}{dw_1} \quad = \quad \frac{d(w_1^2 x_{1i}^2 + 2 * w_1 x_{1i} h + h^2)}{dw_1}$$

Now, using the sum rule of derivatives again:

$$\frac{d(w_1 x_{1i} + h)^2}{dw_1} \quad = \quad \frac{dw_1^2 x_{1i}^2}{dw_1} + \frac{d2w_1 x_{1i} h}{d_{w1}} + \frac{dh^2}{dw_1}$$

Constants can be pulled out of the individual derivative terms, thus:

$$\frac{d(w_1 x_{1i} + h)^2}{dw_1} \quad = \quad x_{1i}^2 * \frac{dw_1^2}{dw_1} \ + \ 2x_{1i} h * \frac{dw_1}{dw_1} \ + \ h^2 * \frac{d1}{dw_1}$$

$$\frac{d(w_1 x_{1i} + h)^2}{dw_1} \quad = \quad 2w_1 x_{1i}^2 \ + \ 2x_{1i} h \quad = \quad 2x_{1i}(w_1 x_{1i} + h)$$

Now, we can actually simplify $w_1 x_{1i} + h$, if we just substitute h…

$$w_1 x_{1i} + h \quad = \quad w_1 x_{1i} \quad + \quad \left(\sum_{j=2}^{m} w_j x_{ji}\right) + b - y_{ai}$$

$$w_1x_{1i} + h \quad = \quad \left(\sum_{\boldsymbol{j=1}}^{m} w_jx_{ji}\right) + b - y_{ai}$$

Thus, that derivative within the summation becomes:

$$\frac{d(w_1x_{1i} + h)^2}{dw_1} = 2x_{1i}\left(\left(\sum_{\boldsymbol{j=1}}^{m} w_jx_{ji}\right) + b - y_{ai}\right)$$

We can now substitute this derivative back into $\frac{dC}{dw_1}$:

$$\frac{dC}{dw_1} \; = \frac{1}{2n} * \sum_{i=1}^{n} 2x_{1i}\left(\left(\sum_{\boldsymbol{j=1}}^{m} w_jx_{ji}\right) + b - y_{ai}\right)$$

This looks pretty messy, so we'll pull the constant 2, out of the first summation, (the 2's cancel out)

$$\frac{dC}{dw_1} \quad = \quad \frac{1}{n} * \sum_{i=1}^{n} x_{1i}\left(\left(\sum_{\boldsymbol{j=1}}^{m} w_jx_{ji}\right) + b - y_{ai}\right)$$

Now, $\frac{dC}{dw_1}$ still LOOKS quite complex, but it's actually quite simple, remember:

$$\left(\sum_{j=1}^{m} w_jx_{ji}\right) + b - y_{ai} \quad = \quad (w_1x_{1i} + w_2x_{2i} + \dots + b) - y_{ai} \quad = y_{pi} - y_{ai} \quad = \quad diff_i$$

$$\frac{dC}{dw_1} \quad = \quad \frac{1}{n}\sum_{i=1}^{n} x_{1i} * diff_i$$

So, to get the derivative of cost, with respect to w_1, all we have to do, is scale up each difference, by the corresponding value in the independent variable attached to w_1. Then sum together all those scaled up differences, and divide the sum by the number of differences.

Perfect! We now know exactly how cost changes, as w_1 changes.

But what about the other weights?

Well, from here, we can actually apply the exact same deduction to w_2, or w_3, or whatever weight, because the mathematics doesn't change. There is nothing special about w_1 compared to any of the other weights, as this is multivariable linear regression, so all independent variables (x_1, x_2, etc), are only in linear form, and thus all weights (w_1, w_2, etc), act only as simple coefficients/multipliers.

So, we can simply say:

$$\frac{dC}{dw_n} = \frac{1}{n}\sum_{i=1}^{n} x_{ni} * diff_i$$

Fantastic! We now know how cost changes, as ANY/ALL of the weights change. Now, we just need to figure out how cost changes, as the BIAS changes. Well, can we apply our deduction to $\frac{dC}{db}$? Let's take a look at our model again to figure that out:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + b$$

As we can see, the bias (b) IS just a variable of the model, able to be tweaked/changed just like any other weight, for the sake of improving the accuracy of the model, but it isn't attached to anything, so what would x_{ni} be, when calculating the derivative?

Well, we can multiply the bias by 1, as $b = b * 1$, therefore, we can rewrite the model as:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + b * 1$$

Therefore, for the purpose of calculating $\frac{dC}{db}$ only, we can simply TREAT the bias as just another weight, that has an independent variable (whose value is ALWAYS 1) attached to it. We'll call this fake independent variable, z, thus:

$$\frac{dC}{db} = \frac{1}{n}\sum_{i=1}^{n} z_i * diff_i$$

z is always equal to one, so the derivative simplifies to:

$$\frac{dC}{db} \quad = \quad \frac{1}{n}\sum_{i=1}^{n} 1 * diff_i \quad = \quad \frac{1}{n}\sum_{i=1}^{n} diff_i$$

So the derivative of cost with respect to the bias, is actually just the AVERAGE/MEAN difference. (This makes perfect sense, as scaling each difference by a factor of 1, does nothing)

Now we know exactly how the cost changes, as all the weights AND the bias change!

So going back to our example, we can now calculate the partial derivatives (how the cost changes as the weights/bias change), and then from there, change our weights/bias, to REDUCE the cost, and thus improve the accuracy of our model!

$$\frac{dC}{dw_1} \quad = \quad \frac{1}{3} * \sum_{i=1}^{3} x_{1i} * diff_i \quad = \quad \frac{1}{3}(2 * -173 + 5 * -310 + 9 * -406) \quad = \quad -1850$$

$$\frac{dC}{dw_2} \quad = \quad \frac{1}{3} * \sum_{i=1}^{3} x_{2i} * diff_i \quad = \quad \frac{1}{3}(31 * -173 + 56 * -310 + 72 * -406) \quad \cong \quad -17318$$

$$\frac{dC}{db} \quad = \quad \frac{1}{3} * \sum_{i=1}^{3} diff_i \quad = \quad \frac{1}{3}(-173 + -310 + -406) \quad \cong \quad -296$$

So, $\frac{dC}{dw_1}$ means, if we change our weight by a very small amount (technically an infinitesimal), then our cost will change by NEGATIVE 1850 times that amount. So for example, roughly speaking, if w_1 changes by +0.001, then cost will change by -1.85. If w_2 changes by +0.001, then cost will change by -17.318, and if b changes by +0.001, then cost will change by -0.296.

So, as we can see from the three partial derivative, they're all currently NEGATIVE, which means if we GROW/INCREASE all the weights and the bias, then the cost will DECAY/DECREASE, and thus the model will become more accurate.

Perfect! We now know to GROW the weights and the bias. Unfortunately however, we don't know by what EXACT AMOUNT (the +0.001 earlier was just an arbitrary example). We also don't know if we should grow them all by the same amount, or grow them by differing amounts relative to each other.

So how on earth would we figure out what EXACT AMOUNT, to grow each weight/bias by?

Well, the answer is in multivariable calculus, specifically the concept of the GRADIENT. In multivariable calculus, we've seen that if you take ALL the partial derivatives of a function/dependent variable (like y in our case), and put them in a VECTOR, then that vector is called the gradient, and it actually points in the DIRECTION OF STEEPEST ASCENT.

So what the gradient of cost would look like in this case, is this:

$$Gradient_{Cost} \quad = \begin{bmatrix} \frac{dC}{dw_1} \\ \frac{dC}{dw_2} \\ \frac{dC}{dw_3} \\ \cdots \\ \frac{dC}{db} \end{bmatrix}$$

And what is meant by "this vector points in the direction of steepest ascent", is if you change all the variables (the weights/bias in this case), by the corresponding amounts specified in the gradient (so change w_1 by $\frac{dC}{dw_1}$, change w_2 by $\frac{dC}{dw_2}$, etc), then you'll be going in the direction of STEEPEST ASCENT, so the dependent variable will be increasing as fast as possible. A good visualisation is provided here:

https://www.youtube.com/watch?v=TEB2z7ZlRAw

So, the gradient vector, basically tells us the exact amount, to change each weight/ bias by, in order to INCREASE cost as fast as possible. Well we don't want that, we want to DECREASE cost as fast as possible, so we want the vector that points in the direction of steepest DESCENT. Well that's simply the OPPOSITE direction, so we can just take the gradient, and multiply it by -1.

Therefore, to DECREASE COST as fast as possible, we can just change the weights and bias, as according to the NEGATIVE GRADIENT:

$$w_1 = w_1 + -\frac{dC}{dw_1}$$

$$w_2 = w_2 + -\frac{dC}{dw_2}$$

$$\cdots$$

$$b = b + -\frac{dC}{db}$$

So, as we can see, we're just changing the weights/bias, as according to the negative gradient, so we're just heading in the direction of steepest descent, and thus cost is dropping as fast as possible!

But there is one problem. HOW FAR, are we going, in the direction of steepest descent? Because for example, lets say:

$$cost \quad = \quad w^2$$

Then the negative gradient is just this:

$$-\frac{dcost}{dw} \quad = \quad -2 * w$$

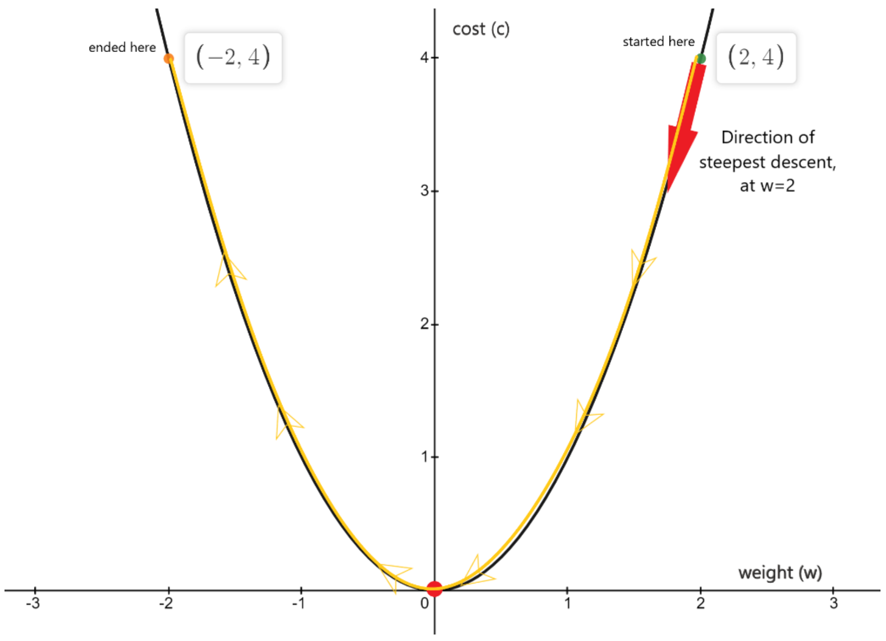
So we know the direction of steepest descent (what exactly to change the weight by, to drop cost as fast as possible).

Now let’s say our weight was initially guessed as 2, so $w = 2$. Then cost is 4, and we want to decrease cost, so we’ll just change the weight using the negative gradient:

$$w \quad = \quad w + -\frac{dcost}{dw} \quad = \quad 2 + -2 * 2 \quad = -2$$

So, we changed our weight (its now -2), with the intention of reducing cost, and thus improving the accuracy of the model, but… if we use this new weight value… $cost = (-2)^2 = 4$… the cost is still the same, our model didn’t get more accurate at all, what happened?

Well basically, we went TOO FAR, in the direction of steepest descent, we SHOT PAST the minimum value of cost, as shown in the graph of cost below:



So, we can't actually change our weights/bias, by the negative gradient directly, as we run the risk of going TOO far in the direction of steepest descent, so we have to SCALE DOWN the negative gradient, by a factor (conventionally called $learning\ rate, alpha$, or α)

So, when we change our weights/bias, it'll look like this:

$$w_1 = w_1 + \alpha * -\frac{dC}{dw_1}$$

$$w_2 = w_2 + \alpha * -\frac{dC}{dw_2}$$

$$\ldots$$

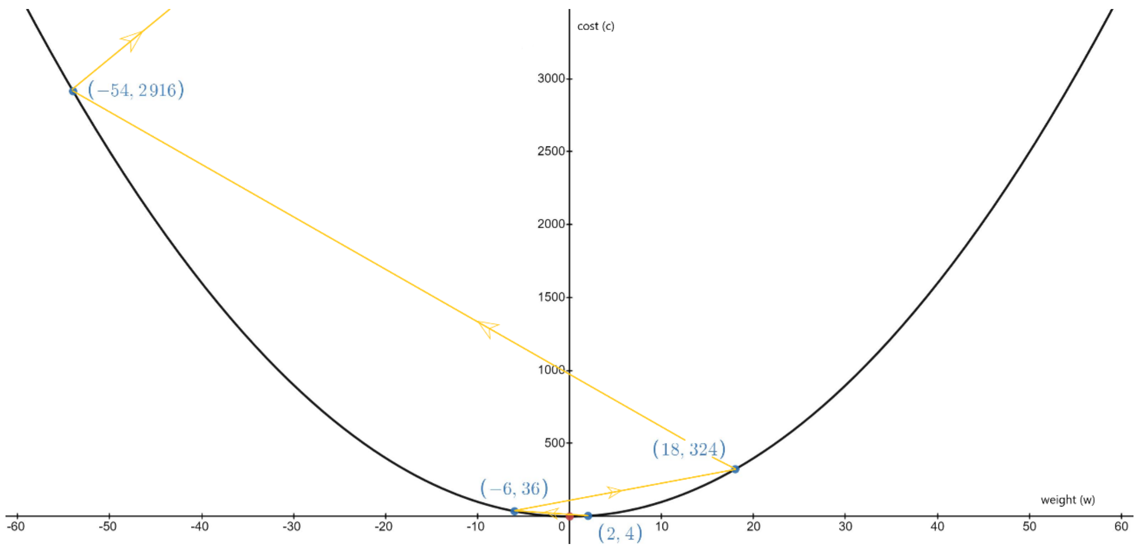
$$b = b + \alpha * -\frac{dC}{db}$$

Now, how do we figure out what α is? Well for this basic project, we'll just guess, by starting at $\frac{1}{10^1}$, and going down by an order of magnitude each time, so $\frac{1}{10^2}$ is next, then $\frac{1}{10^3}$, etc (so we'll be using trial and error to find α).

(There are better ways of figuring of what α is, such as by using adaptive learning rate algorithms, like fuzzy logic, Adam, Adagrad, or using learning rate schedulers, or even just grid search. However we'll be ignoring these, to keep the explanation as simple as possible).

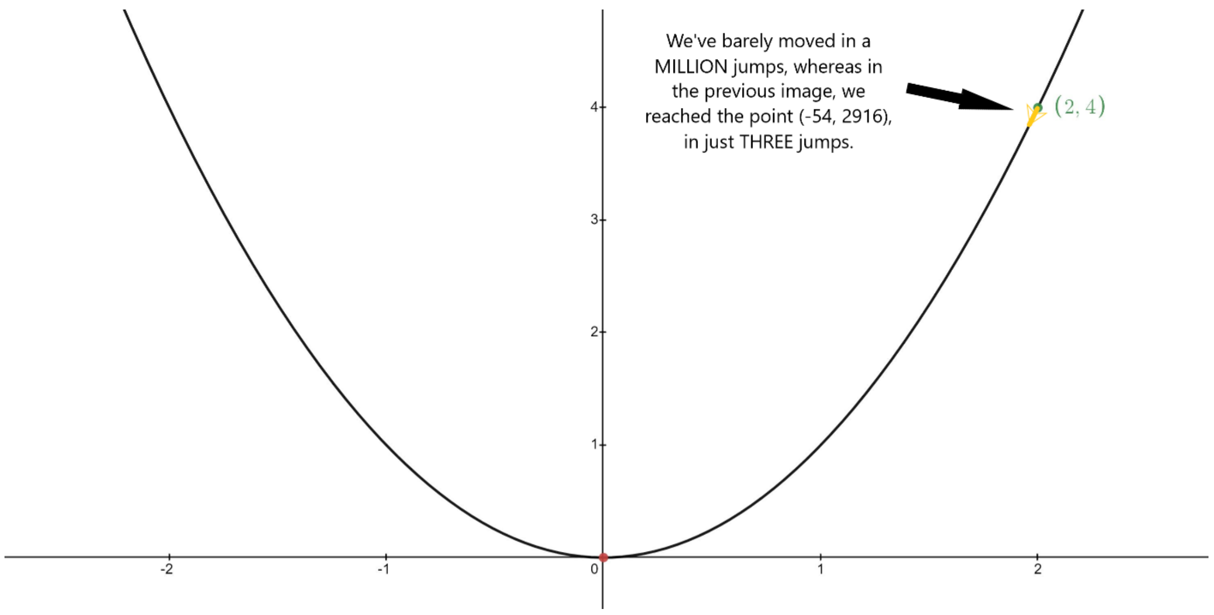
Now, If our α is too large, then we'll end up diverging away from the minimum cost. This is because we'll start off at some point, and be going in the direction of steepest descent far enough, that we'll end up at a new point that has a STEEPER descent, than the previous point. So the partial derivatives here overall will be larger, which means when we go in the new, steeper direction, we'll end up travelling EVEN FURTHER this time, to a newer point with an EVEN STEEPER descent, that will cause us to travel EVEN FURTHER, to an even STEEPER point, and so on, spiralling out of control.

A visualisation of divergence due to a large α, is shown below: ($\alpha = 2$)



If our α is too small, then we do actually CONVERGE on the minimum, it will just take a longer and longer amount of time, as α gets smaller, so optimization ends up getting pretty bad.

A visualisation of incredibly slow convergence, due to a small α, is shown below: ($\alpha = 10^{-30}$)



Thus, we want to find an α that allows us to converge on the minimum, in the smallest amount of time possible.

Now, finding a decent α value, through simple trial and error, is pretty inefficient, and requires a lot of boring manual calculations, so instead of providing a mathematical walkthrough of this inefficient process, I've simply used my own code, and found a decent value for α, which was $\alpha = 10^{-4}$.
(Code used is within a folder called "optional" on the Github repo)

So, now that we have our α, we can actually change our weights and bias!

As a reminder, our weights/bias, cost, and negative gradient of cost, was as follows:

$$w_1 = 1 \qquad w_2 = 2 \qquad b = 3$$

$$-\frac{dC}{dw_1} = 1850 \qquad -\frac{dC}{dw_2} = \frac{51955}{3} \qquad -\frac{dC}{db} = \frac{889}{3}$$

$$cost \;=\; 48{,}477.5$$

Now, changing/improving our weights/bias:

$$w_1 \quad = \quad w_1 + \alpha * -\frac{dC}{dw_1} \quad = \quad 1 + 10^{-4} * 1850 \quad = \quad 1.185$$

$$w_2 \quad = \quad w_2 + \alpha * -\frac{dC}{dw_2} \quad = \quad 2 + 10^{-4} * \frac{51955}{3} \quad \cong \quad 3.73$$

$$b \quad = \quad b + \alpha * -\frac{dC}{db} \quad = \quad 3 + 10^{-4} * \frac{889}{3} \quad \cong \quad 3.03$$

We have our new and improved model! Let's make some predictions with it, and see if cost is better:

$$y = 1.185x_1 + 3.73x_2 + 3.03$$

$$y_{p1} \cong 121 \qquad y_{p2} \cong 218 \qquad y_{p3} \cong 282$$

$$cost \quad \cong \quad \frac{(121-240)^2 + (218-430)^2 + (282-562)^2}{2*3} \quad \cong \quad 22{,}918$$

Excellent! Our cost has dropped from about 48,000, to 23,000, almost halved! So our model has been improved!

If we want to improve it further, all we have to do, is update the gradient of cost (with the new weights/bias values), then change our weights/bias again accordingly, if you do this over and over, the model becomes increasingly accurately, as we're converging on the minimum of the cost.

That's it! That's basically how machine learning works, with multivariable linear regression, to allow you to reasonably predict virtually any quantifiable feature, if given the right raw data!

Note – Feature Scaling

If a MLA wants to create an accurate prediction model, then it will need to "learn/train" off of a dataset, and that can be quite time/resource intensive. Thus, feature scaling is one way of optimizing the "learning" process.

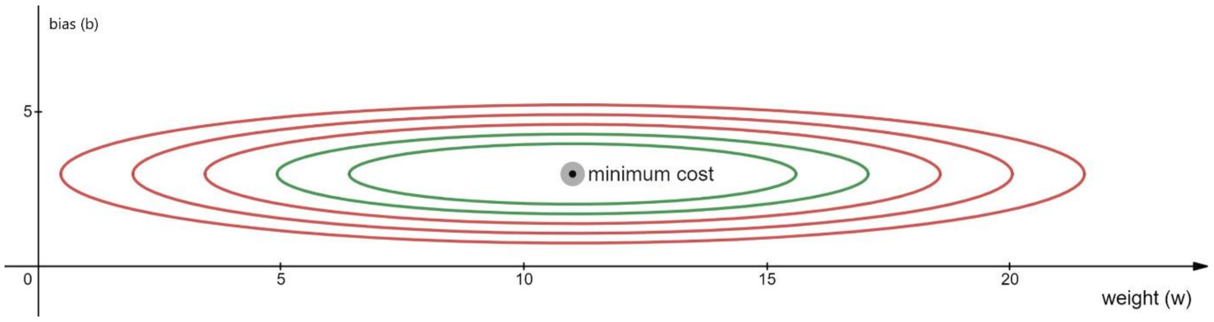
To explain, let's say we had a simple situation, whereby we have only 1 independent variable, so only 1 weight and bias:

$$y = w * x + b$$

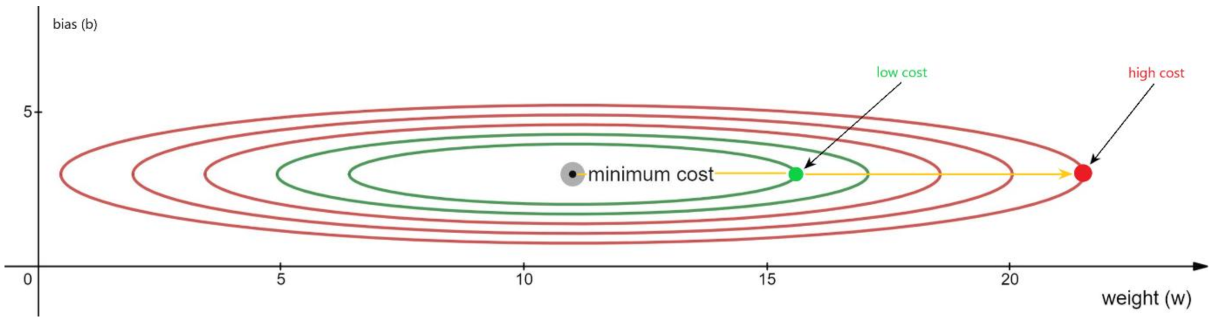
Now, let's say our independent variable (x), only had incredibly small-scale values. So let's say it ranges from billionths to millionths $(10^{-9}\ to\ 10^{-6})$.

This means, that if we were to change our weight and bias, by the same amounts, then what will happen, is the cost will change by far larger amounts as our BIAS changes, compared to if our weight changes.

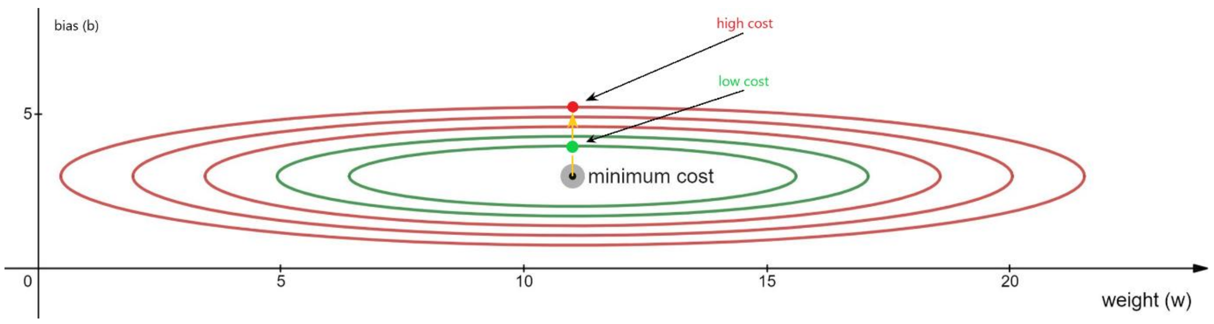
Here's a visualisation of what this looks like (contour graph of cost):



As we can see, as we change w by 12 (from about 11, to 23), we go from the minimum cost to the high cost.



However, if we change b by only 2 (from about 3, to 5), we also go from the minimum cost, to the same high cost.



As we can see, cost changes very quickly as the bias changes, compared to the weight. Mathematical proof of this concept can also be provided by the gradient of cost that we worked out earlier:

How cost changes as a weight changes, is shown below:

$$\frac{dC}{dw_n} = \frac{1}{n}\sum_{i=1}^{n} x_{ni} * diff_i$$

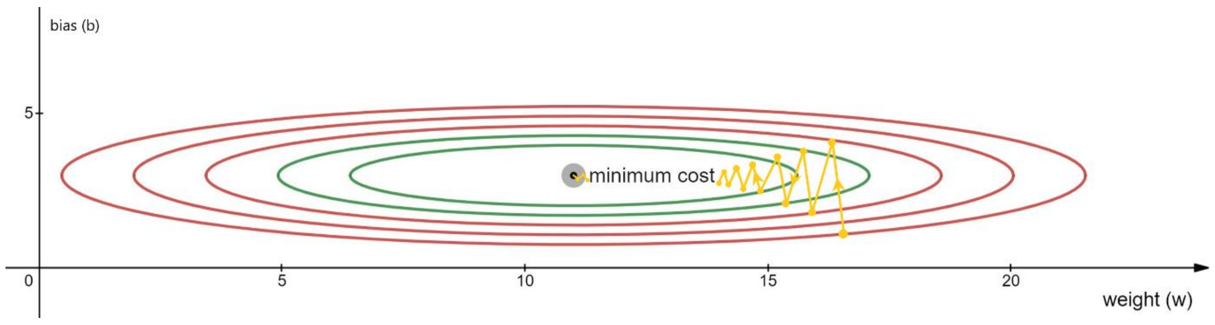
As we can see, there is a factor of x_{ni} (the values of the independent variable), attached to every single difference $(diff_i)$. This means if x incredibly small-scale (say billionths to millionths), then the derivative of cost with respect to the weight, will be full of incredibly small multipliers, scaling all the terms down, and thus the derivative of cost ends up being a very small number.

Meanwhile, how the cost changes, as the BIAS changes:

$$\frac{dC}{db} = \frac{1}{n}\sum_{i=1}^{n} diff_i$$

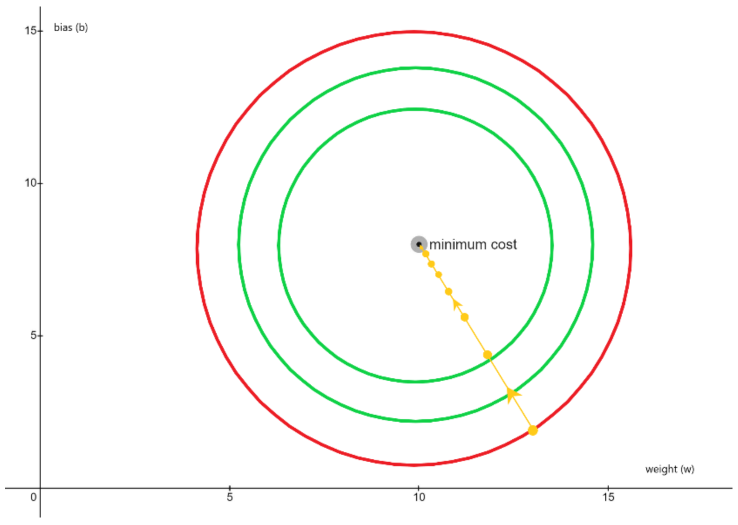
There is no factor of x_{ni} to scale all the terms down, thus the derivative will end up being quite large. (And if you had a weight, attached to a LARGE-SCALE variable, then the derivative of cost will end up being very large).

Now, why this is a problem (having an oval/ellipse shaped cost function), is because when training is undertaken, small steps are repeatedly made in the direction of steepest descent, which will look like this:



The direction of steepest descent is always perpendicular to the contour lines, and because our independent variable hasn't been scaled up, we end up with oval/ellipse shaped contour lines, so the direction of steepest descent at most points, isn't actually pointing at the minimum, but is instead off by some angle (up to 90 degrees). So we have to take A LOT of steps, to get really close to the minimum.

Now, if we scaled up our independent variable, so that $\frac{dC}{dw} \cong \frac{dC}{dB}$, then we'd end up with circular contour lines, where the direction of steepest descent at ANY point, is pointing to the minimum, so we take far less steps, to get really close to the minimum.



Thus, the amount of time the "learning" process takes, decreases!

(Note: This simple example used a single independent variable, but the same applies if there were many independent variables, you'd simply scale them all, to within the same ranges as each other)