Why you need to understand the maths behind machine learning

At one point I hated maths too, trust →@lachinemearning

If I showed you this image of the minkowski distance to an 12 year old who just learned about Pythagoras' theorem-

$$D(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

This would scare the living daylight out of them. But if I told the kid:

- Take two numbers on a graph (7 and 3)
- Subtract them from each other (7 - 3 = 4)
- Then multiply the result by 2 (4 * 2 = 8)
- Notice the jump from our original number 4, to the number 8
- The larger the difference between our 2 numbers, the larger the number gets when multiplied by 2
- This is how we punish our 2 numbers for being so far from each other
- Now lets add all of the punishments together

But notice how that's so many words for just saying it's a **generalised formula for descaling a summed penalised difference between n points in a dimension so it's a meaningful distance instead of an arbitrarily inflated number**.

The point I'm trying to make is- understanding maths makes more advanced maths a lot more digestible. Maths notation and problem solving will become so ingrained in you, it'll actually be easier to just write out a bunch of maths jargon to another academic than writing an essay on why you're doing what you're doing.

You don't have to be a maths connoisseur- believe it or not it took a while for me to like maths. I don't believe that anyone is 'maths' smart and not 'maths' smart. Sure, some people are quicker at picking up certain knowledge- but we are ALL capable of being good at maths. The thing that sets us apart is practice.

Higher level maths is all built on preceding knowledge. Someone isn't born knowing how to solve this integration by parts question automatically

$$\int \frac{1}{x^3} \ln x \, dx$$

You need to know how to:

- Manipulate fractions
- Understand exponent rules
- Being able to rewrite expressions in equivalent forms
- What a function is
- What natural log is
- What a derivative is
- What an integrand is
- Knowing the LIATE rule for integration by parts

If you are weak in them, you will be confused by how to answer that question.

You don't have to know how to solve a triple integral in your day to day life as a corporate slave. If you're going into academia/research, yes you're gonna have to be very capable at maths since you'll be working more with maths than computer science at that point. But if you know you want to make that bread spinning up pipelines, don't sweat it.

In machine learning, there are three major branches: Statistics, Linear Algebra, and Calculus- in that order of importance. TLDR: Linear algebra is

how your data is represented, Statistics is about unbiased generalisation, and Calculus is the engine of learning.

Statistics - describing your data

Statistics in machine learning is our tool for trustworthiness. We have so many different tests and techniques to find more information about our data:

- Where my data comes from
- How representative it is
- Does my model perform well because it's good or was my test data I gave…easy?

In a world of messy incoherent data, statistics helps us navigate best ways to have our ground truth. **(You will hear the phrase ground truth a LOT, note it down)**

Because we are humans using human data, we need to make it **unskewed**, **numerical**, and **easy** to **toss** into math functions. Whether statistical tests, or machine learning algorithms (basically math functions that are so general you can put any part of your data to it and it'll come out correct).

Even with LLMs like **chatGPT**, **Claude**, **Gemini**, at the end of the day they are a *massive probabilistic machine*. When you talk to it, it gives you the best guess answer. If you add the words:

- *'Car'*
- *'Automotive'*
- *'Engine'*

in your prompt, the LLM will crank up its probability dial- so it might talk back to you with its internal mind looking at how to respond with words like this:

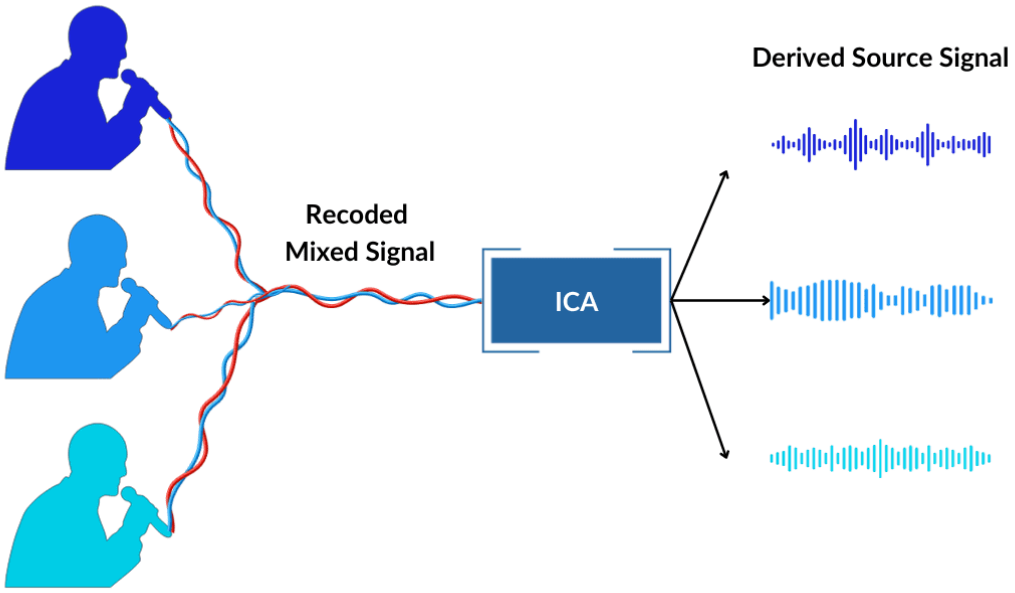
- *'Trucks'* 89%
- *'Water'* 9%

TLDR: LLMs don't process words the way you do, they just know that certain word chunks that we call tokens, are statistically more likely to come after one another.

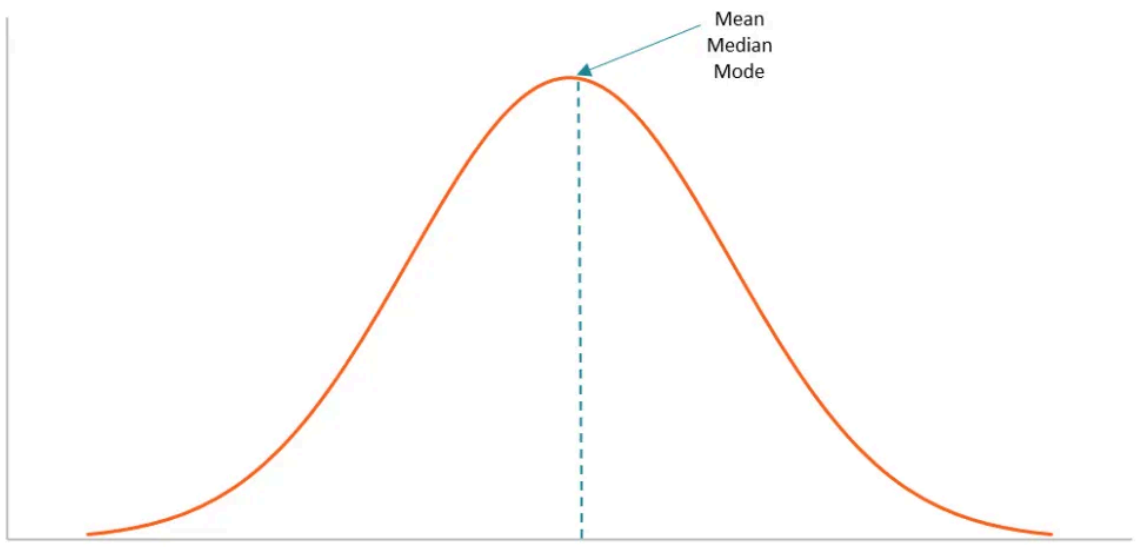
I want to bring up something crucial, what happens if what we are measuring has soooo many different things **blended** together and we need them apart?

Imagine a karaoke bar and your **3** friends have **three** mics in their hands and are all **singing** together drunk. **All** 3 mics are picking up all 3 of their voices at different volumes, so your recordings are **super tangled** nooooo

Independent component analysis, or **ICA**, takes those blended recordings and **splits** their voices back up so you can hear each person **separately**. This is also called *blind source separation* because you are never told who is singing where, **ICA** does the magic of separating their voices for you.



Without too much jargon, if you're familiar with the famous statistical **bell curve** shape (*its okay if you're not*), basically it's the most **boring** and **normal** shape data can take where most data is **clustered** around the **ideal average**.



When your friends' voices are **blended**, it looks more like this **boring bell curve**. What **ICA** does is go **hunting** for any shapes that start to look **different** to this boring shape. Random **spikes**, **flat** areas, anything that looks a bit **funny**.

Usually, anything with a slight personality **peaking** out is almost always **one of** the original singing voices that has been mixed in with the other 2, so **ICA** keeps **poking** and **pulling** at that string until it comes apart.

Linear algebra- structuring and forming your data

Machine learning models understand only numbers. But our data cannot be converted into an arbitrary (random) value.

If I have a house that has **3** *bedrooms,* **2** *bathrooms,* is **1200** *sqft,* **£250,000** *price*, we need to put that information together to represent one house…

Here comes vectors! So we might make our house represented something like

3
2
1200
250000

Imagine this is a vector, sorry for terrible formatting

Now if I have **multiple** houses because I want to **predict** house prices based on their **room number**, how **big** they are..

I will take **different** houses and put them in **one** dataset- this is a **matrix**! So from the very first step, your data IS **linear algebra**!

Now when we want to take **ALL** these houses and find the best **general** maths function that can best explain how these houses correlate with their price, we will put it through a machine learning model!

Now bear with me, this is gonna get a bit more confusing but I will cover this more under deep learning- but when we get this scary equation

$$z = \left(\sum_{i=1}^{n} x_i \times w_i \right) + b$$

All we are doing is

- getting our house attributes (the number of rooms, how big the property is) which is x_i
- we multiply it by a bunch of numbers in a matrix that determine how important that specific attribute may be (w)
- then add one big number to that whole value $(Bias\ vector)$.

That multiplication between your **house attributes** and those sets of **random values** that dial the importance of those attributes, that's **matrix multiplication**!
Congratulations, you just learned that a neural network is just a massive forward chain of vector and matrix multiplication lol

Why we do matrix multiplication is because it lets you take data and transform it- you can:

- squish it
- stretch it

- you can do anything with it

The point is, you squish and stretch the data via matrix multiplication until it gives you something useful! Whether it's a **classification**, a **prediction**, a **generated token** in an LLM response.

How linear algebra is involved in PCA

You may or may not have heard of PCA, and that's okay I'm here to build the intuition for you :)

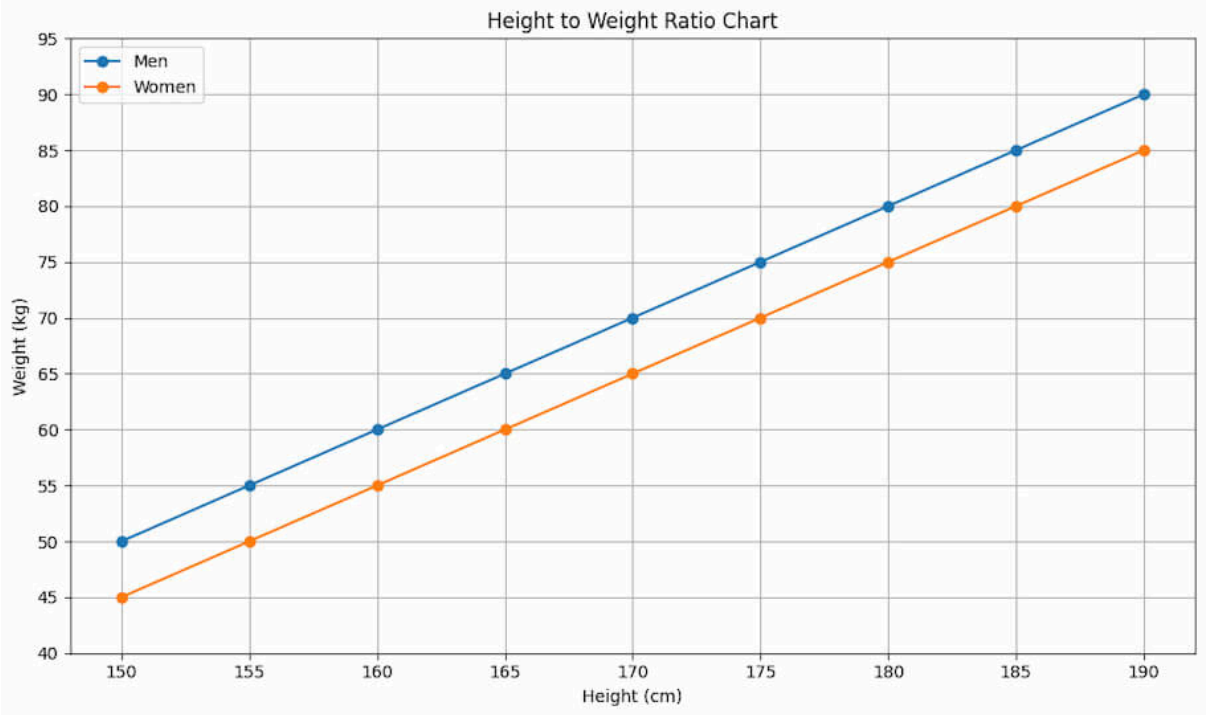
Remember our dataset of houses being a matrix? Our **rows** are **houses**, **columns** are the things we **measure**. Looks like this:

House 1	**2 bedroom**	**2 bathroom**	**900 sqft**	**£200,000**
House 2	**3 bedroom**	**2 bathrooms**	**1500 sqft**	**£350,000**
House 3	**1 bedroom**	**1 bathrooms**	**800 sqft**	**£100,000**

The thing about real data is sometimes, a lot of the columns are basically saying the same thing twice.

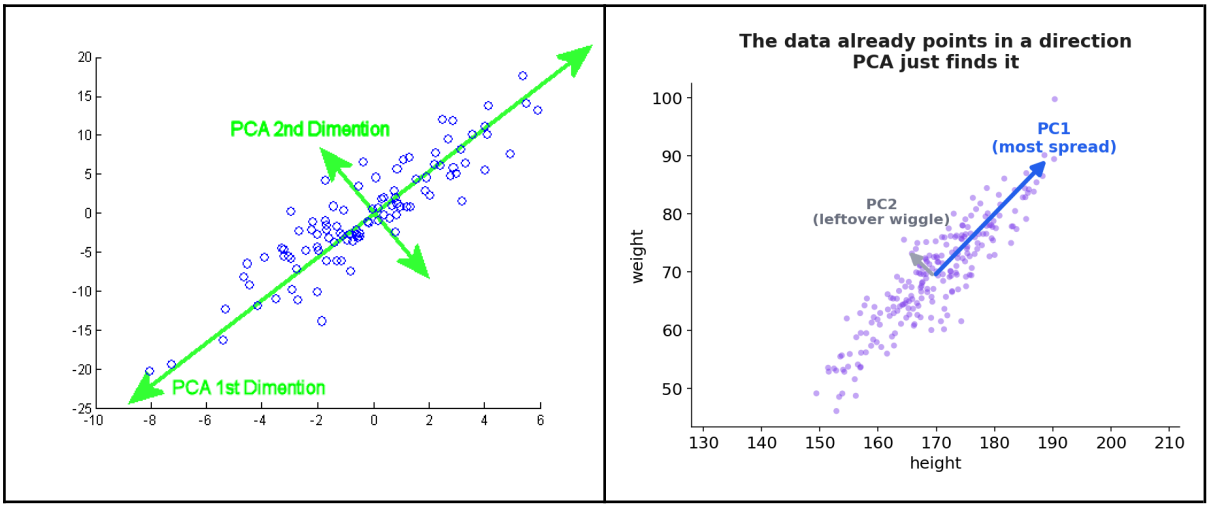
And if they are showing the same pattern of like 2 straight lines going up, the model doesn't have to learn the direction of BOTH lines. It just needs to know that the pattern of the whole data goes up!

You will know this in your gut, take the attributes of height and weight for example- a taller person is usually heavier.



Why would I store two arrows showing that the general pattern of data is that it goes up positively when one would do?- That is the whole idea of **PCA**!

- It hunts for the directions your data spreads out in the most, and lets you describe everything using those directions.



Notice PCA 1 shows the most information about data. Your eyes immediately see the up-and-to-the-right pattern. This is the data's main direction, the **first principal component.*

The PCA 2 which is the little arrow poking out sideways is the **second principal component. It's the leftover wiggle that the **PCA1** arrow couldn't explain.*

PCA just finds the arrows for you and chooses the biggest one. But how do we find the arrows? This is where something called **eigen-decomposition** comes in. It sounds a lot scarier than it is lol.

Quick detour- take the number **12.** You may have learned at ages 13-15 that you can break down **12** into 3 * 2 * 2, also known as its **prime factors** which are the building blocks of numbers.

Eigen-decomposition does the exact same thing, but just to a **matrix!** The intuition should be starting to click. When you slowly simplify what a matrix is, it's broken down into 2 core factors:

- **Eigen-vectors** which are the core directions baked into the matrix (the direct comparison to your prime factors)
- **Eigen-values** which is how strong each of those directions are (how much each building block counts pretty much)

Okay but what are these directions??? Well, when you learn vectors and matrices, you learn that multiplying a vector by a matrix, **STRETCHES** and **ROTATES** the vector at the same time.

What an **eigen**-**vector** is, is the **core direction** in a matrix that just **REFUSES** to rotate. It can only stretch. *It can change in magnitude aka get longer or shorter, not in direction**.***

The **eigen**-**value** is just how much the eigen-vector **stretches** by. If a line gets longer by 2, the line is your vector, and the 2 is your eigen-value.

IRIS WHY IS THIS RELEVANT TO FRICKIN PCA????

Okay okay relax, here- remember when we wanted the arrows that show the direction of how much data spreads out the most! Our arrows are actually just the eigen-vectors lol

We build a table called a **covariance matrix** (*DONT GET SCARED PLEASE STAY*) which is just a table recording how every **feature** moves **together**. We

break down the table into **eigen-vectors** and we get the associating **eigen-values**.

The **largest** eigen-value shows the largest **spread** (*remember how the bigger the eigen value, the bigger the line stretches by, so naturally we want the number that is the largest stretch right?*)

We get the data with the **largest arrow** (eigen-vector with the largest eigen-value) and basically use that to **teach** our model and **delete** all the other parts of the data since they are not contributing to learning more information. I don't expect you to understand PCA in one try, but digesting some parts and reading more about it later will have you thanking me for making this section meow

Congrats, you just learned PCA and eigen decomposition yippee! Now PCA typically uses singular value decomposition, but that's for you to go out into the internet land and learn!! This was to build a quick overview heh

It's easy to think pca and ica are related and are inverses of each other, trust ill explain why

PCA	ICA
● Declutterer ● Two brown strings side by side, you keep one ● Cares about spread	● Unmixer ● Three different coloured strings mixed together, goal is to separate them ● Cares about independence

PCA is run first as a warm-up step for ICA.

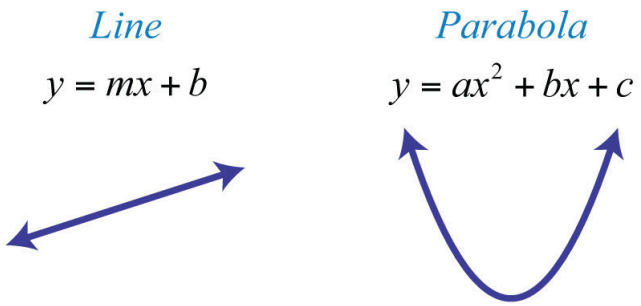
Calculus - optimising the model on the transformed data

Okay sure hopefully you understood what the hell a neural network does…hopefully…

And lets sayyyyyy you take this network to make a prediction but…

shock! The prediction is **wrong**!

How do we tell the network how to become less wrong? **Calculus** solves this problem, I'll explain why hopefully you can understand.



Let's build intuition first.

Look at the left-most graph (**y=mx+b**). You learned in school that if I take two points on the line, I can find the distance between them by **subtracting** them from each other.

But for the right-most line (also called a **parabola**!), the way you find the distance between 2 points in one section may be **different** in how you find it in another section.

The idea of calculus is to study **continuous change**, things that aren't just straight lines. So our rules become a little more **complicated** to handle constant change.

In machine learning, we have a way to measure how wrong our model is from the actual truth…which is what we call the loss **function**.

What it does is spit out a number from the **model's prediction** and the **real answer**. The bigger the number -> the more wrong the model was.

- *Think about the minkowski distance formula at the beginning of this document, remember how it multiplies the difference between two values with a number to punish a larger difference? Our two values are the model's prediction and the ground truth!*

We want to **minimise** the difference, makes sense right?

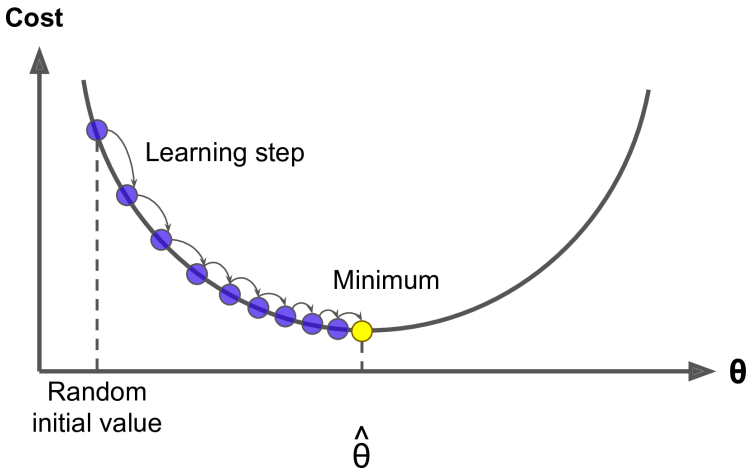
The closer the model's prediction is to the actual truth, the better. Our loss function is basically the minkowski distance, and imagine we plot it on a graph.

Look at the *2D version* image, we want to get to the **lowest** point of the curve, and we try to find the slope/gradient of where we are on the graph to find:

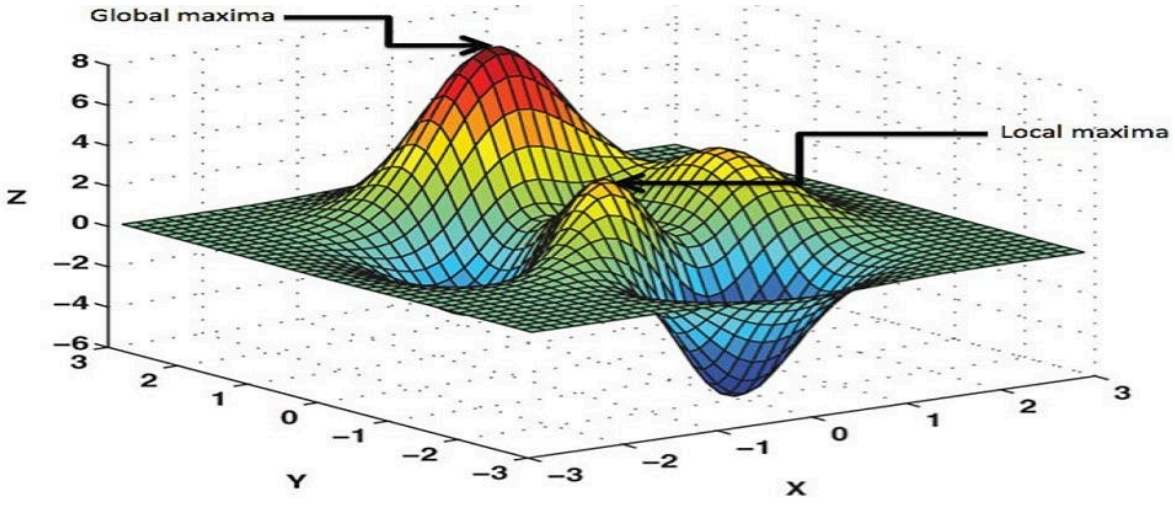
- which direction do we go downhill from where we are standing right now

This is what we call the **derivative**- the **derivative** is just a **slope**/**gradient**. When someone says to find the derivative of a function, they're just asking you to *find the gradient of the function on a graph that isn't a straight line*.

2D version



3D version



The 3d version is just to help you understand a more realistic version of the mathematical landscape. When we find where we are standing right now via our derivative (gradient), we take a step in the direction that helps us go down hill. The cycle looks like this:

- Calculate **derivative**
- Take step **opposite** of the direction given to us
- Recalculate the **derivative**

Congrats you understood how calculus is directly correlated to **gradient descent**. (Literally **descending** the mountain using the **gradient** of where we are now lol)

The way this works with all that weight bias input jargon- is that our **weights** and **bias** values are the only values in the network we can change. We CAN'T change our input values, but we can change the importance of them.

When trying to make the neural network more **accurate**, **backpropagation** is when you use **chain rule** (*a differentiation technique you'll learn later)* repeatedly to change the **weights** and **biases** to values that allow the network to be **more accurate**.

Iris…what the hell is backpropagation and gradient descent.

- **Backpropagation**: finding out how much each weight was the reason that the model was incorrect
- **Gradient descent**: Using the error to update the actual weights to better values that give a more accurate prediction

The calculus chain rule in backpropagation basically says *"if my answer/output depends on a value that can change called x- through different layers of the network, i can find the gradient of my answer/output by multiplying the derivative (gradient) found at every step/layer"*

Hopefully you understood from this why you need the maths, and why its good to start from the ground up! Trust me if you don't know how to multiply powers together, you will NOT know the power rule of calculus.

Take your time, be patient with yourself, and it will all be worth it in the end. Practice khan academy unit tests every so often after you learn the maths to keep your mind fresh and never forget. This is how you become ‘maths’ smart. Go get em tiger!! <3