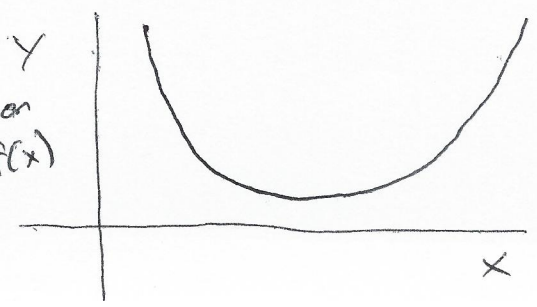
Cost function anatomy 101: Big picture categorization (1)

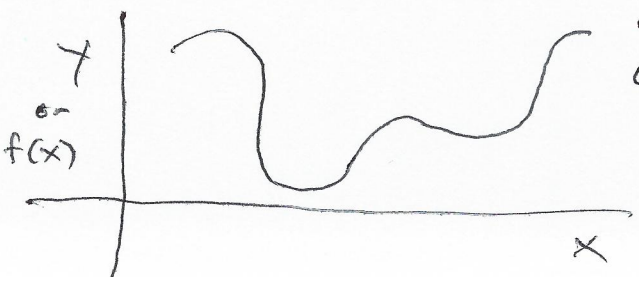
- The cost functions we looked at for examples of linear regression + classification were 3-d surfaces, but here we will examine 2-d cost functions (since they're easier to draw)!

- But no worries — everything discussed here generalizes easily to any dimension you want

- (cost) functions come in all shapes and sizes, but we can <u>generally</u> categorize them into two categories:

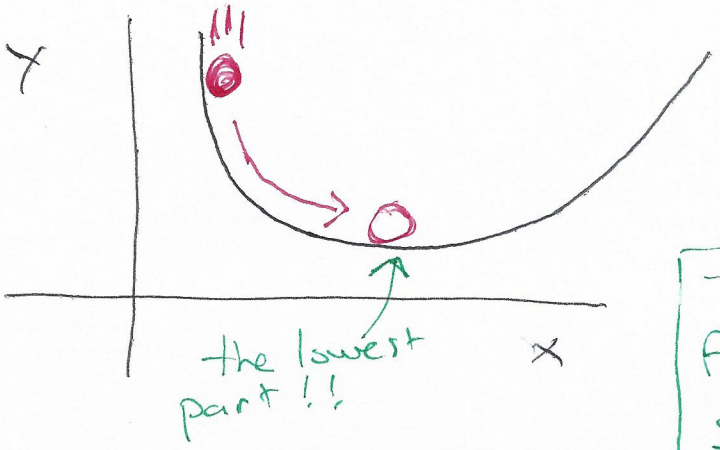


"convex"



"non convex"

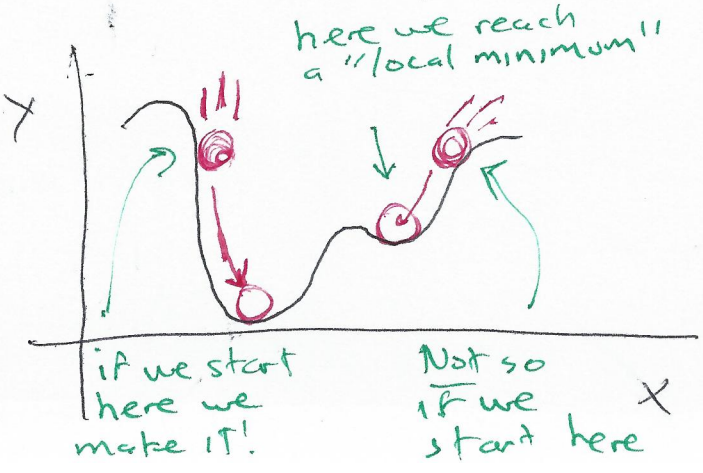
- Convex functions look like bowls or half-pipes
  - If you rolled a marble down any side of a convex function it would eventually reach the lowest point



(as shown in our simulator over here)

The lowest point on a function is called it's global minimum
jargon time.

- A non-convex function is like a mountain range, it goes up and down a bunch, and has multiple hills and/or valleys

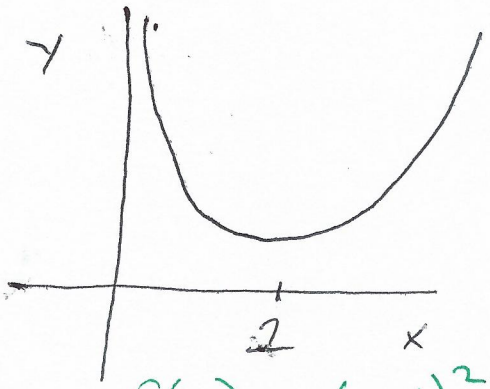


- roll a marble down a side of a non-convex function and you might reach the global minimum - but you might not! Depends on where you start!

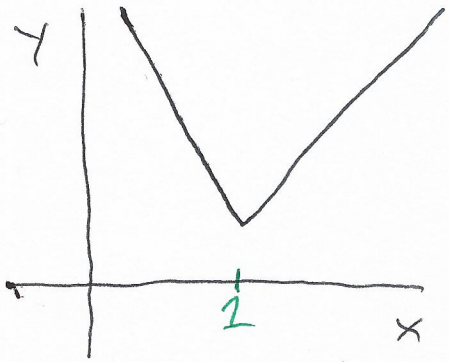
- Now don't go thinking that all convex functions look like the one I drew, they can be quite diverse looking. Examples

minima = many minimum

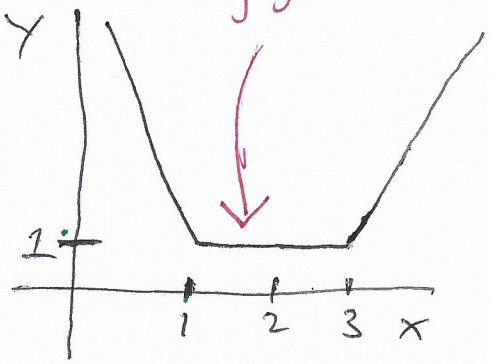
many global minima!



$f(x) = (x-1)^2$

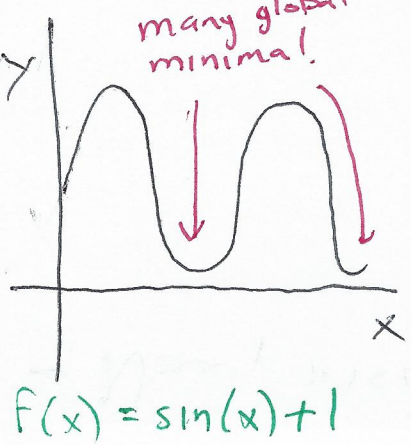


$f(x) = |x-1|$

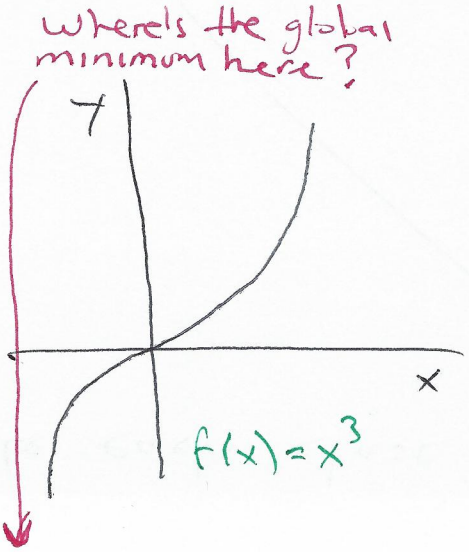


$$f(x) = \begin{cases} -x+2 & x < 1 \\ 1 & 1 \le x \le 3 \\ x-3 & x > 3 \end{cases}$$

= Same goes for non-convex functions: they are even more diverse in shape



$f(x) = \sin(x)+1$



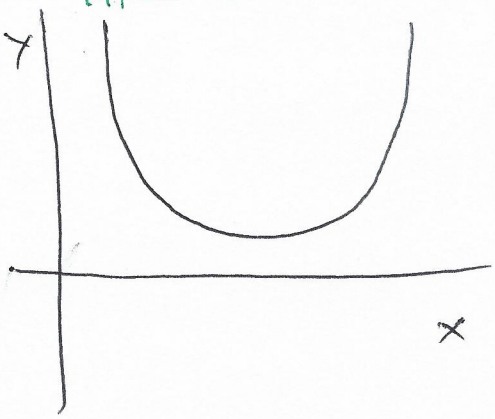
$f(x) = x^3$

- However for machine learning we generally only see smoothly curving cost functions with a fixed lower bound (i.e. where it's global minima are not $-\infty$)

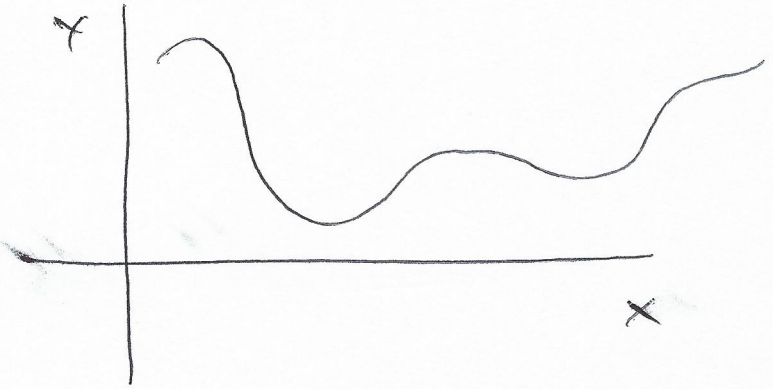
  ← not always but very often

- So from our lineup, we typically see

Convex function like this



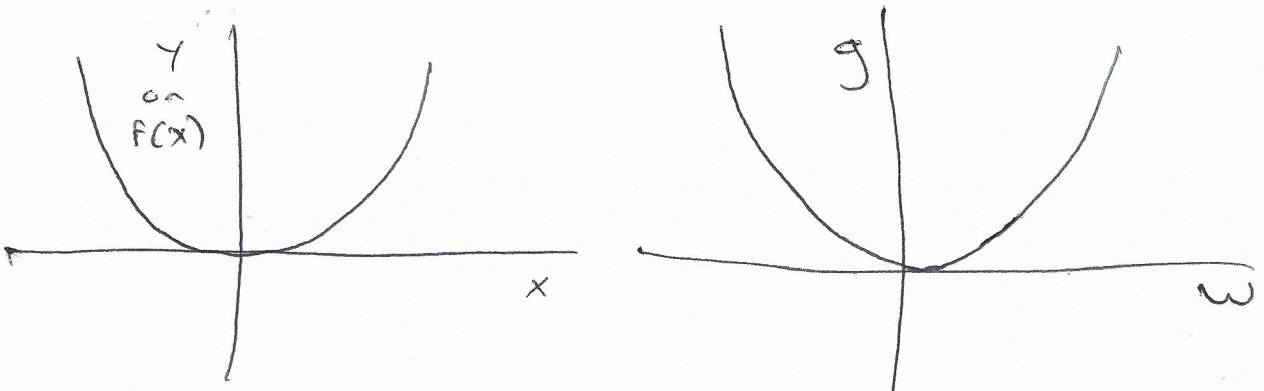
Non-convex function like this



Cost function anatomy 102: the derivative and linear approximation

- We're going to make a slight change of notation: since we are talking about cost functions of machine learning models that reflect a parameter or wieght tuning, we will use w as our input from now on ("w" is short for "weight")

e.g.



$f(x) = x^2$

$g(w) = w^2$

↖ this says $g(w) = w^2$

Same thing, different notation

$x \longmapsto w$

$f \longmapsto g$

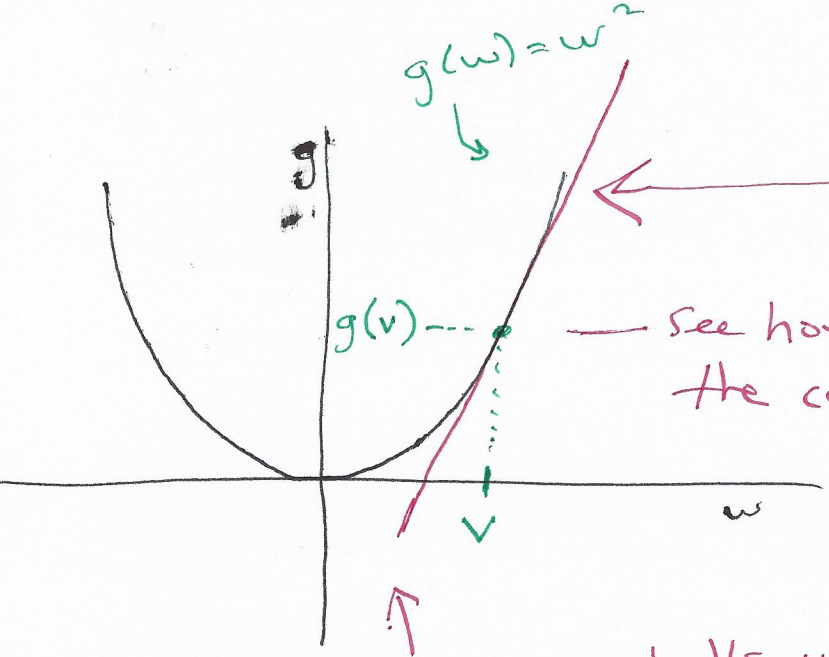
- Great, now with that out of the way we can talk about

The derivative

- What is it? The derivative of a ~~function~~ function $g(w)$ at a point v — denoted $g'(v)$ — (that's a "prime") (you say this "g prime v") is the slope of the line tangent to g at v

..... wha? Picture time.

Say this is g, $g'(v)$ is the slope of this line

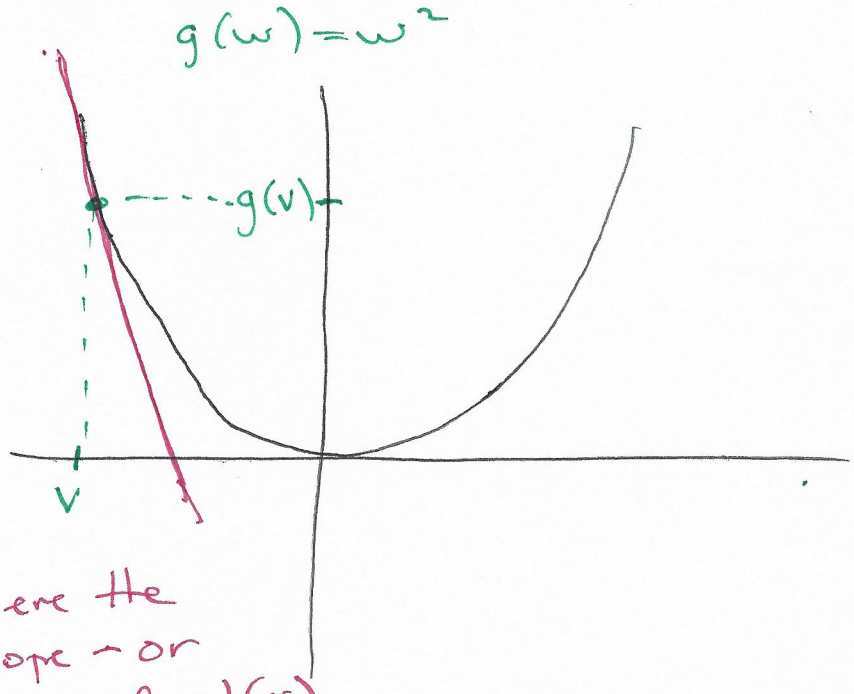


— see how that line is just touching the curve? That's how you know it's a tangent line! If it just touches.

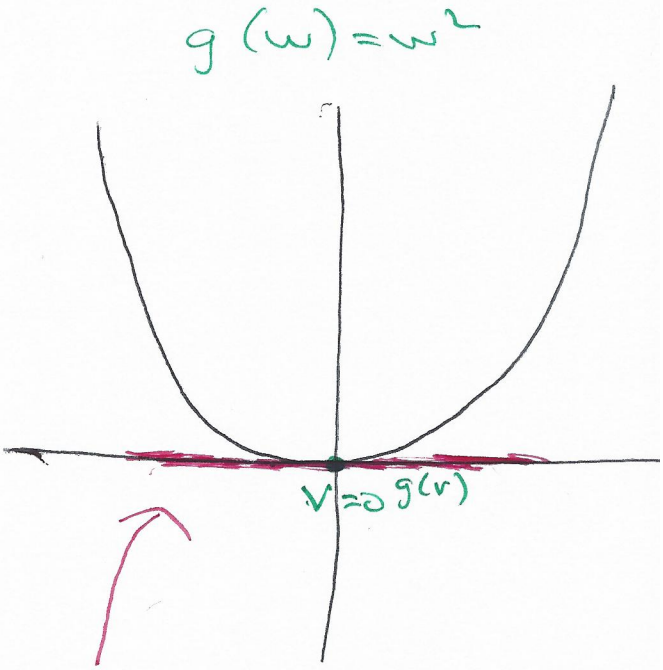
Using our eye-balls we can see the slope — or value of $g'(v)$ — is positive

That is $g'(v) > 0$

How about if v is here?



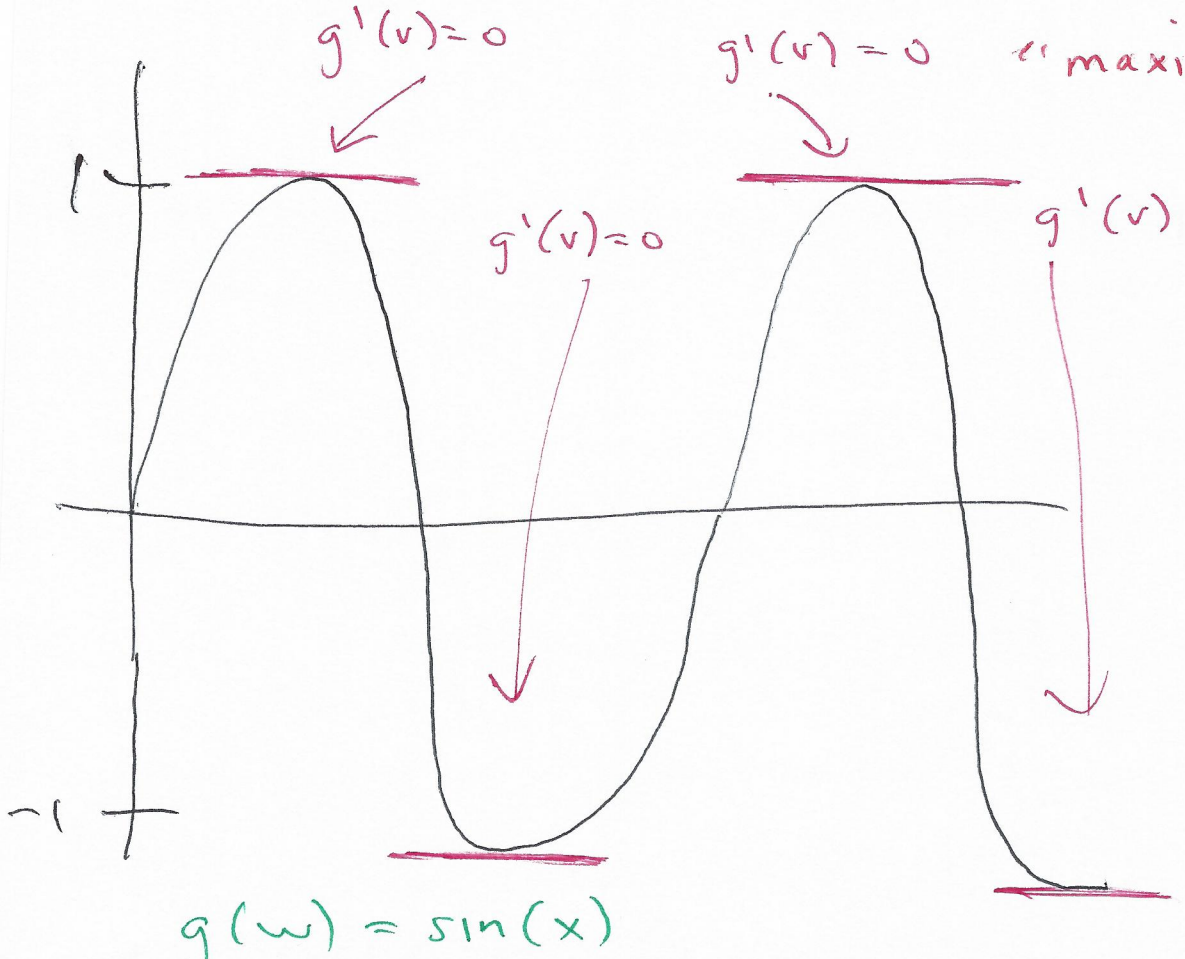
Here the slope - or value of g'(v) - is negative → so g'(v) < 0



Here g'(v) = 0

- Notice: g'(v) = 0 only when v was the global minimum of the function
- Does that generalize? Is it true
  - for convex functions ✓✓✓ YOU BET!!
  - for non-convex functions ✗ NOPE

How come? Cause this ....

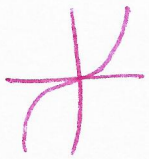


$g(w) = \sin(x)$

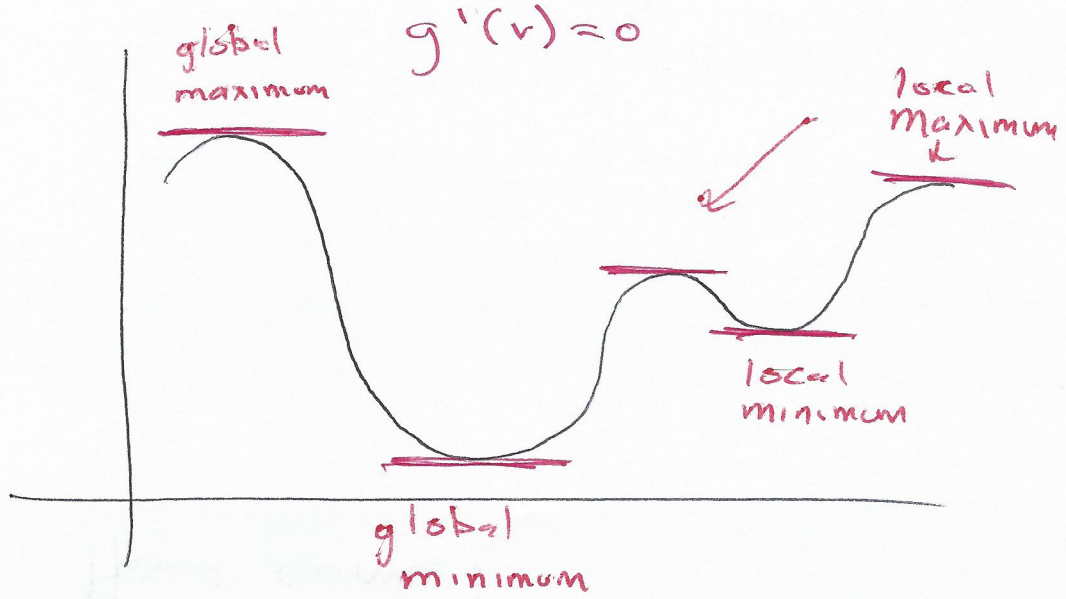
Aside

There are some other points to that satisfy $g'(v)=0$ that are not maxima or minima. These are called "saddle points"

E.g. $g(w)=w^3$ at $v=0$



and this



Please!

Make it stop!

So many $g'(v)=0$ that aren't global minima!!

- Look, chill out, its cool. Even though it is not the case for non-convex functions that $g'(v) = 0$ only when v is a global minimum, this condition or test

$$g'(v) = 0$$

is extremely useful. Using this we can design an algorithm that can practically find the global minima of a generic convex or non-convex cost function

> **Jargon time**
> Any point satisfying this is called a "critical" or "stationary" point

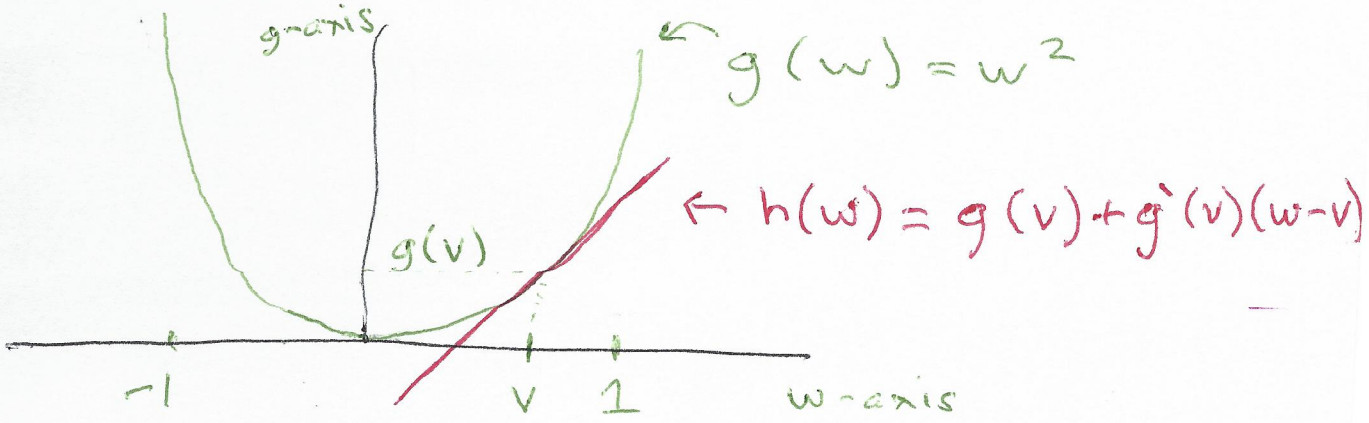
> **Reminder:** we want to find the global minima of machine learning cost functions since they correspond to best ~~fit~~ fitting regression / classification schemes.

To Do:

Write a short function in Python that produces the following items:

- A plot of the cost function $g(w) = w^2$ over the region of input $[-1, 1]$
- A plot of the linear approximation to g ~~for~~ at a point v where v can be any point in $(-1, 1)$

The following picture - minus the equations - is what you're after

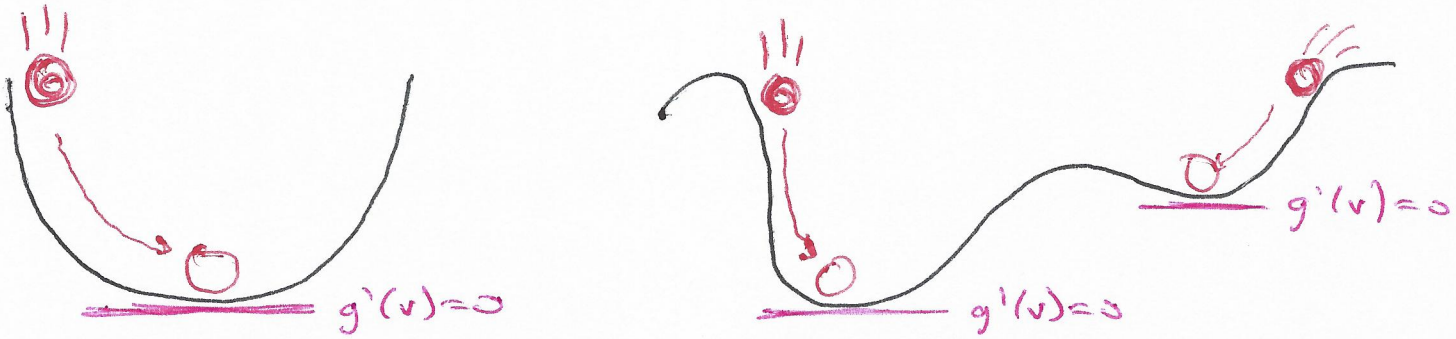


y is $h(w) = g(v) + g'(v)(w-v)$ the equation
e tangent line at v?

So, how can we use this condition

$$g'(v) = 0$$

to find minima? Back to our marble simulator....



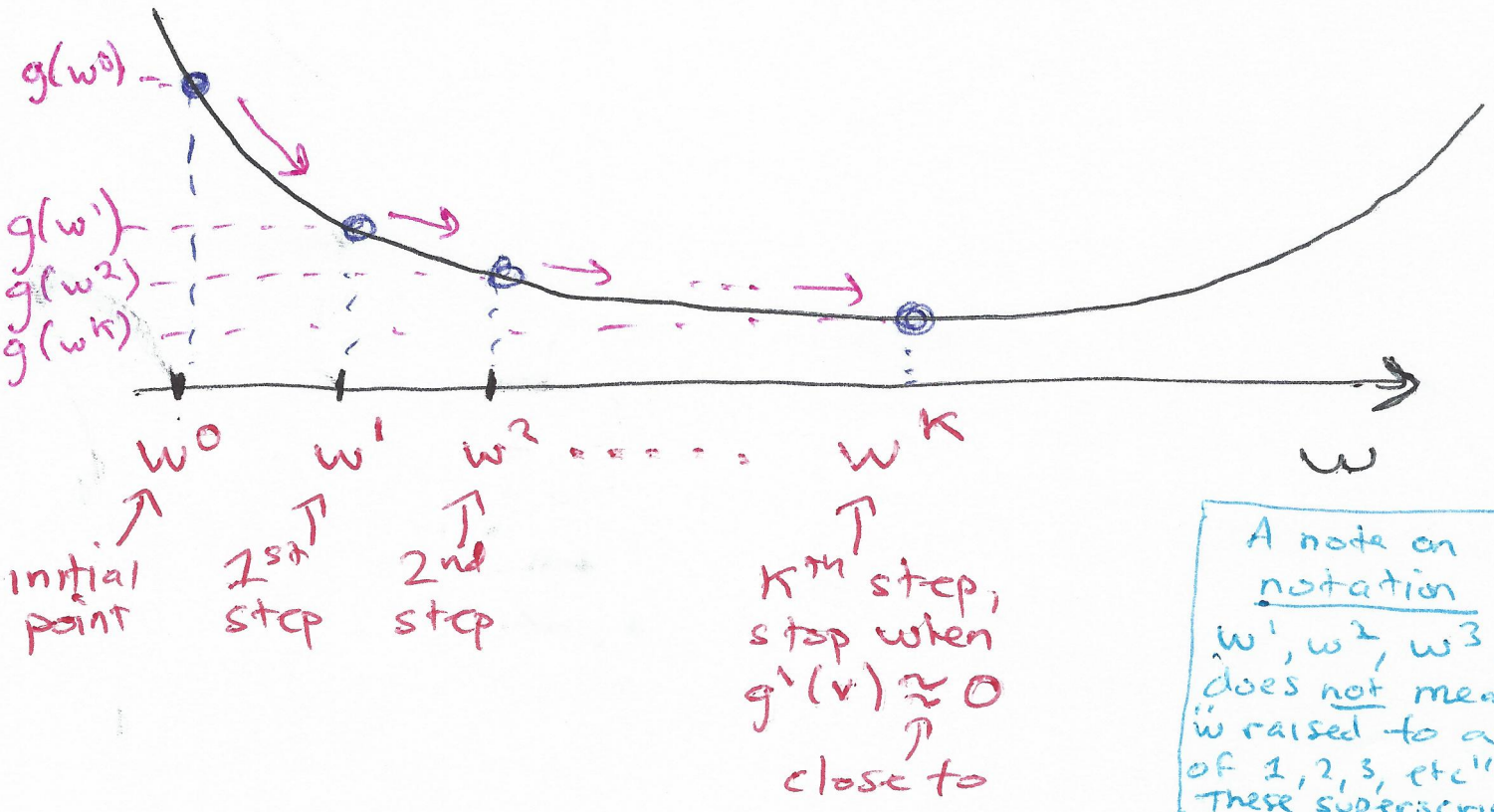
Mathematical optimization algorithms work like our marble simulator - you roll down hill until you reach a point where

$$g'(v) = 0$$ a stationary pt.!!

then you stop!!

- These algorithms sequentially decrease in cost function value on their way to a stationary point of g

Looks like this on a convex cost



A note on notation

$w^1, w^2, w^3 \ldots$ does not mean "w raised to a power of 1, 2, 3, etc" These superscripts are indicies!!

- You get down sequentially i.e. by a sequence of steps

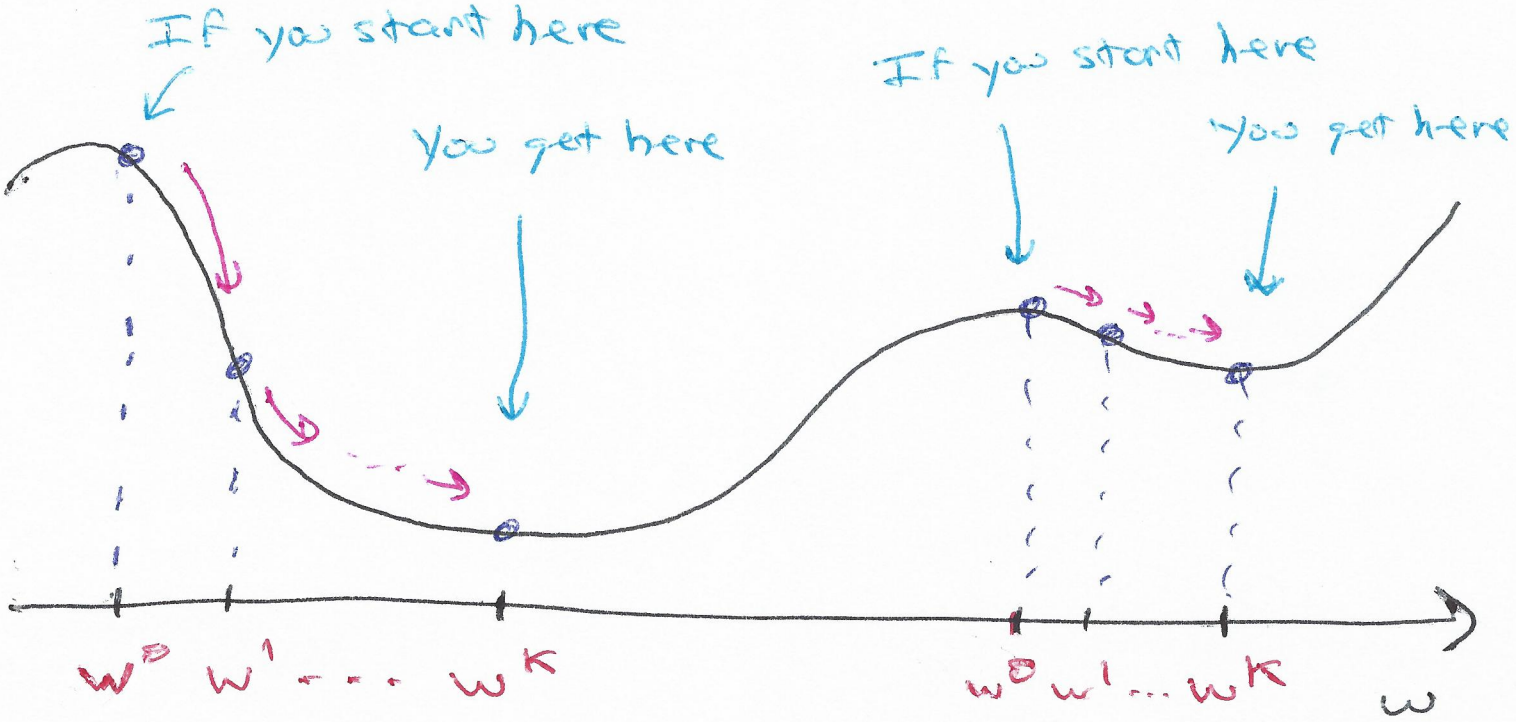
$$w^0, w^1, w^2, \ldots, w^K$$

and at each step you (typically) are decreasing the cost function value

$$g(w^0) > g(w^1) > g(w^2) > \ldots > g(w^K)$$

until at some w^K $g(w^K) \approx 0$

Same sort of deal with non-convex costs, but where we end up depends on where we begin. E.g.,



- Ok, fine, but how does it work? How can we program an algo to do this?

we need this!!

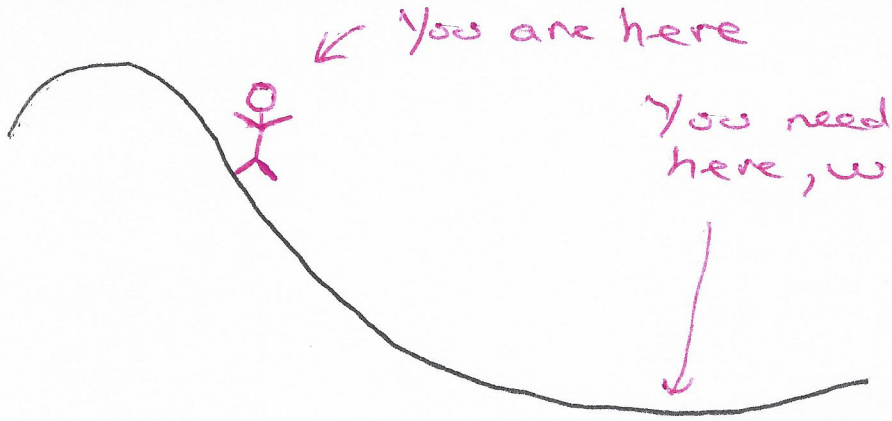
Remember: in our pictures these cost functions have only 1 weight/parameter. So we can see everything that's going on. In practice machine learning costs can have hundreds, thousands, or even millions of weights! These costs are REALLY high dimensional!! You can't visualize them!

- How do you know which way is down on a cost function that lives in 10, 100, or 10^6 dimensional space?

- Remember - you need to imagine you're completely in the dark - you can't "see" the function

- It's like if you were somewhere near the top of a mountain [redacted]. It's pitch-black out. No moon, no stars, no light whatsoever. You need to get to the bottom of the mountain. How do you do it?

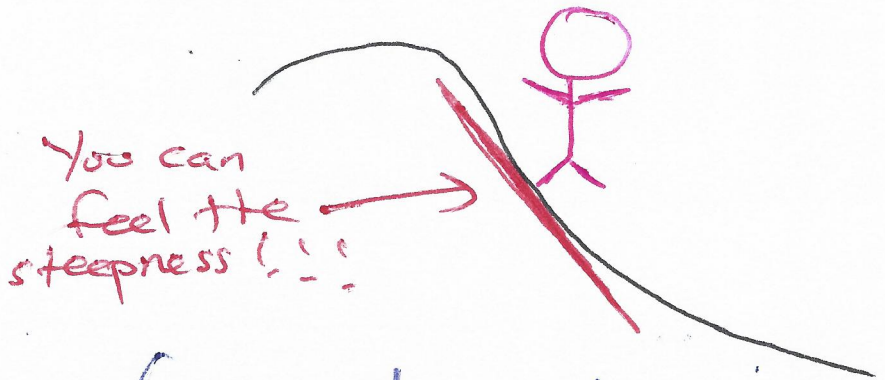
- You got no fancy equipment (no ropes, grapling hooks, sleds etc). You have to walk: step-by-step



- Remember you can't <u>see</u> anything...

  but you can <u>feel</u> something.

- You can feel the <u>steepness</u> of the ground beneath your feet!!



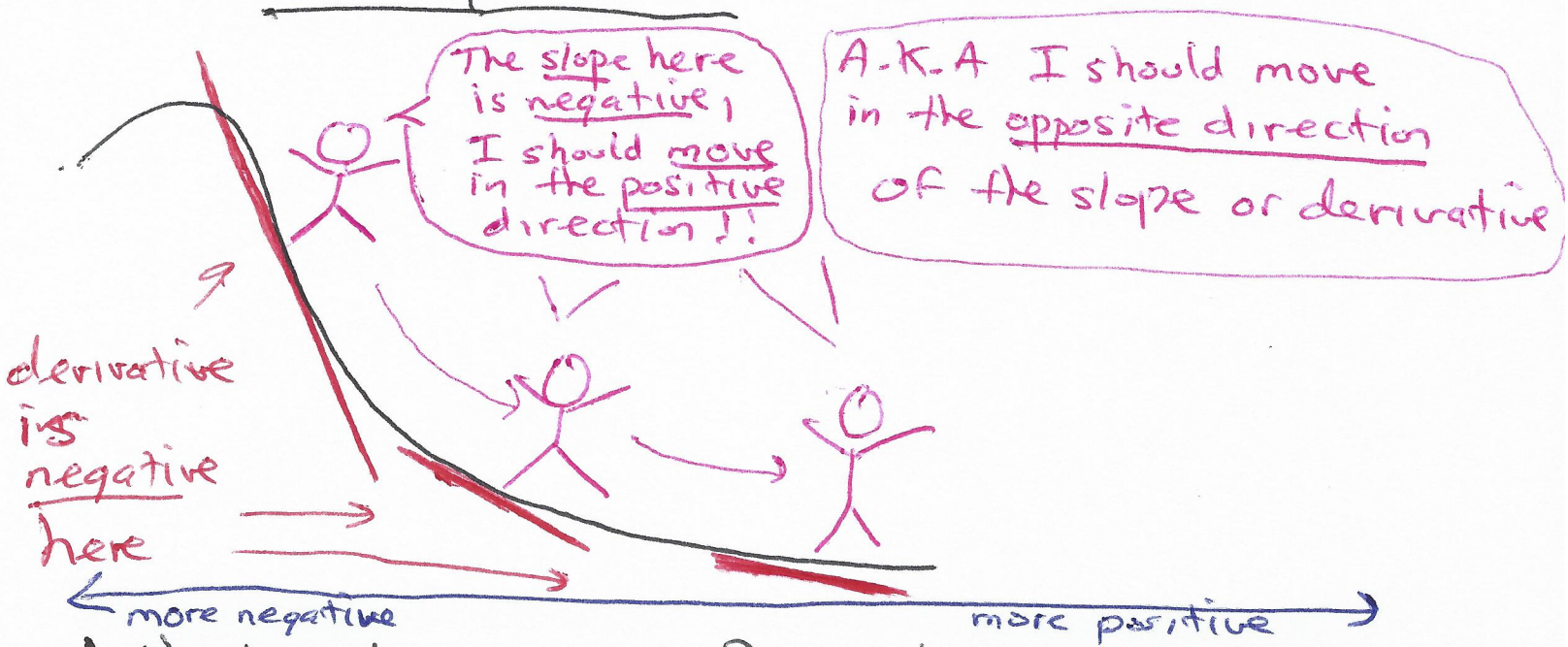
(zoomed version of you on a mountain)

- In other words - you can feel the <u>slope</u> of the line tangent to where you are standing

  i.e. the <u>derivative there</u>

- Imagine that - you can <u>feel</u> the steepness under your feet. You can feel the slope of the tangent line — the derivative.

- Using that knowledge you can easily take <u>a step down</u> the mountain

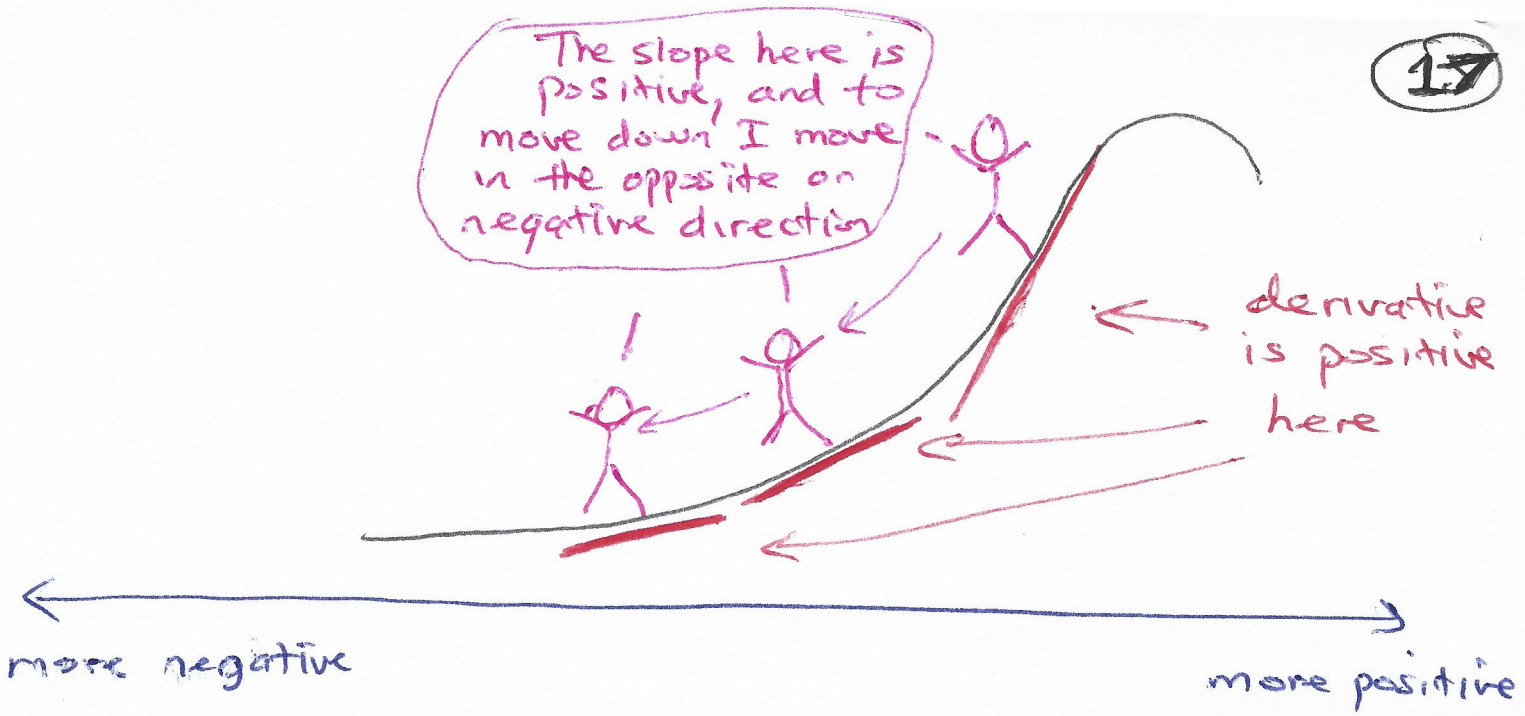


- What do you do? You take a step in the <u>opposite direction of the derivative</u>

opposite of derivative at v $= -g'(v)$

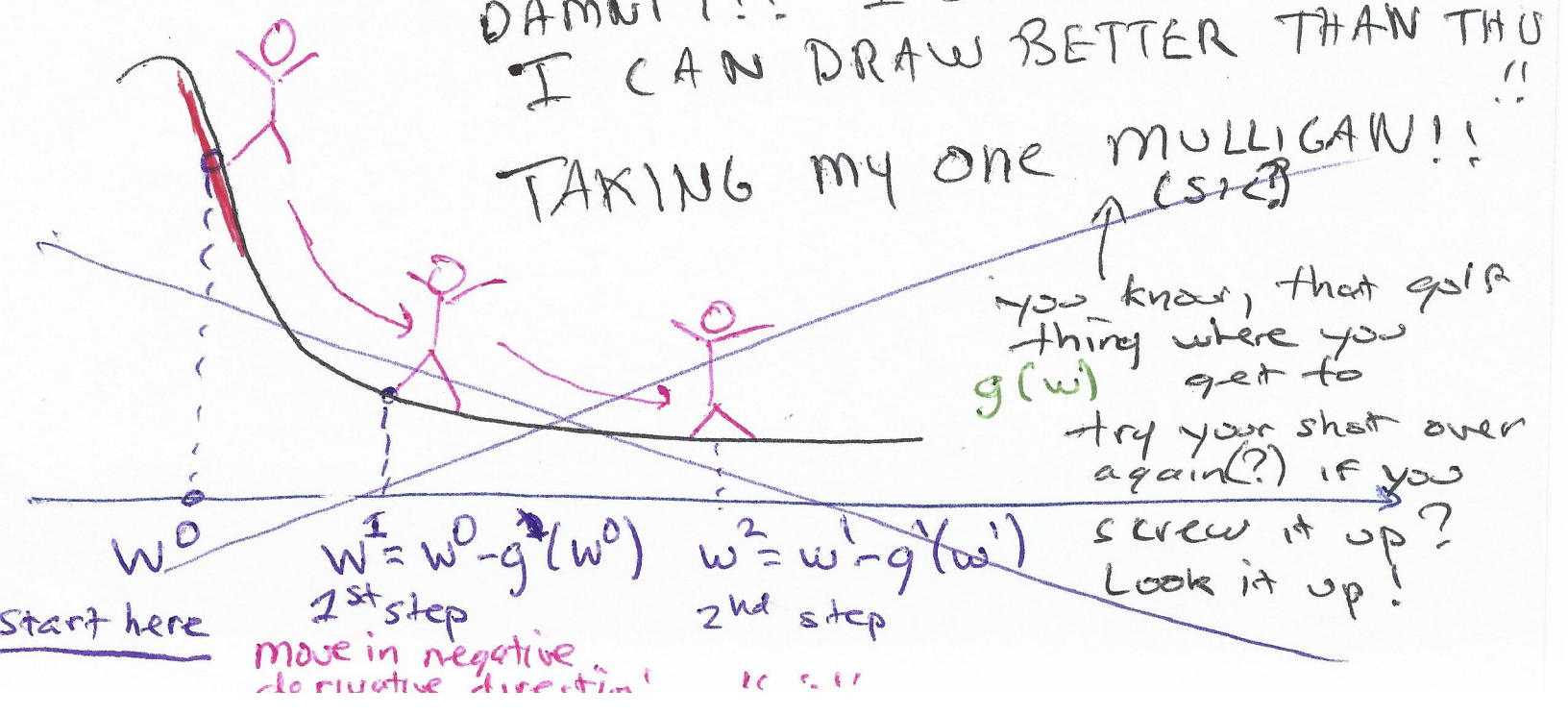
or negative

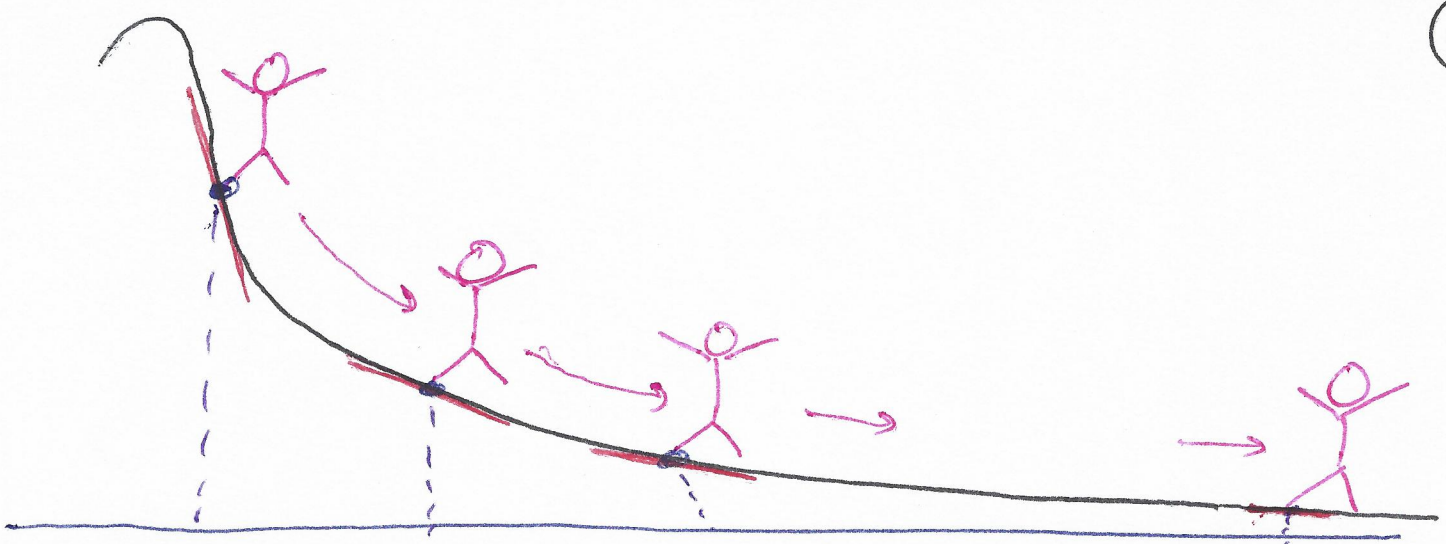
- Notice how this works if you start on the right side instead of the left.



- Still happening: to go down hill take steps in opposite (or negative) derivative direction

- Adding back in the math notation we have





w^0 start here

$w^1 = w^0 - g'(w^0)$ move away from w^0 in negative derivative direction

$w^2 = w^1 - g'(w^1)$ " " " w^1 " "

$w^k = w^{k-1} - g'(w^{k-1})$ " " " w^{k-1} " "

- So to go down hill we take steps of the form

$$w^j = w^{j-1} - g'(w^{j-1})$$

- One caveat: how big should each step be? We can control the stride-length by multiplying the direction $-g'(w^{j-1})$ by a tunable parameter α → we set this either once at the beginning of our journey or at each step.

- So our strategy for step taking becomes

$$w^j = w^{j-1} - \alpha\, g'(w^{j-1})$$

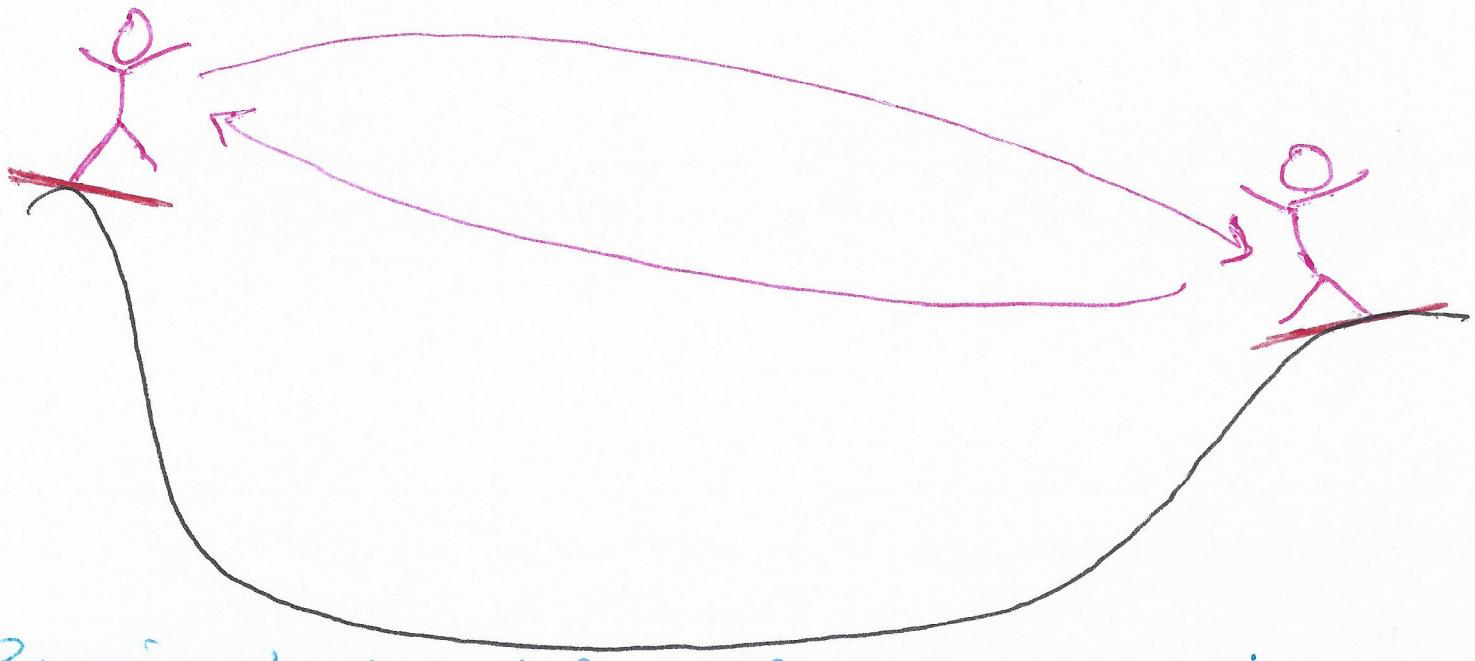
- Why add this <u>step-length</u> adjusting parameter? <u>So we don't overshoot our objective</u> as $-g'(w^{j-1})$ gives us ~~the~~ the <u>direction</u> to travel, but doesn't tell us how big each step should be.

- Could be the case that just doing

$$w^j = w^{j-1} - g'(w^{j-1})$$

this is like if $\alpha = 1$

we might take too big of steps. So we need to control their magnitude



Bouncing back and forth forever → we know the right direction at each step, but haven't made the <u>step length</u> small enough.

- So we need that step length parameter, so our scheme can always be made to descend if we make it (step length) small enough

$$w^j = w^{j-1} - \alpha\, g'(w^{j-1})$$

- This is called the Gradient Descent Algorithm

- Jargon: α is sometimes called the "learning rate", often called the step length - which makes more sense and is less hocus-pocus-y

- Why called "Gradient Descent"?

The gradient is just the derivative of a multivariable input function.

We have 1 input in our examples so far.

So we could just call it Derivative Descent, which makes sense: we are descending in the negative direction of the derivative