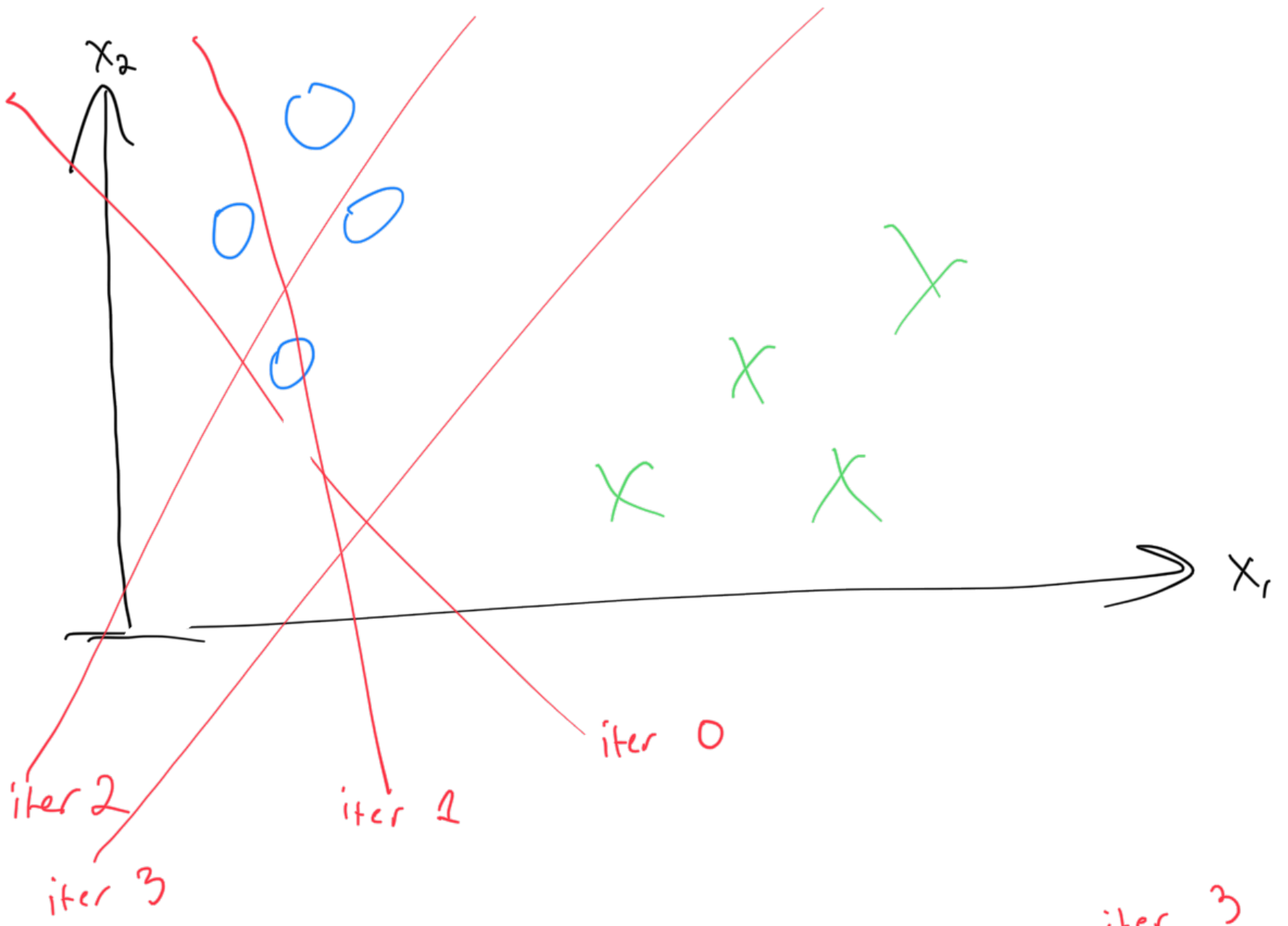
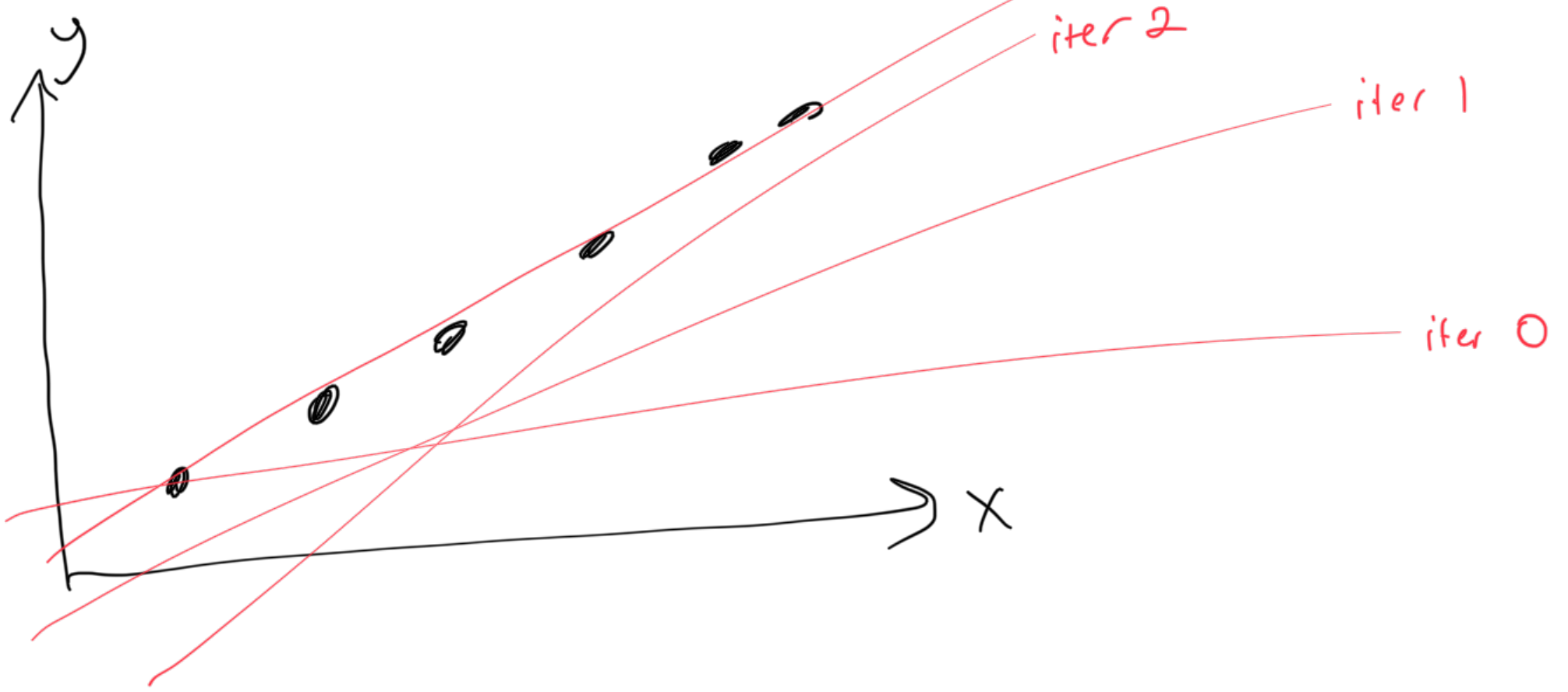
Day 8: Gradient Descent

What happens when we call

model.fit(X_train, y_train) ?

[for linear/logistic/other regressions, neural networks, ...]





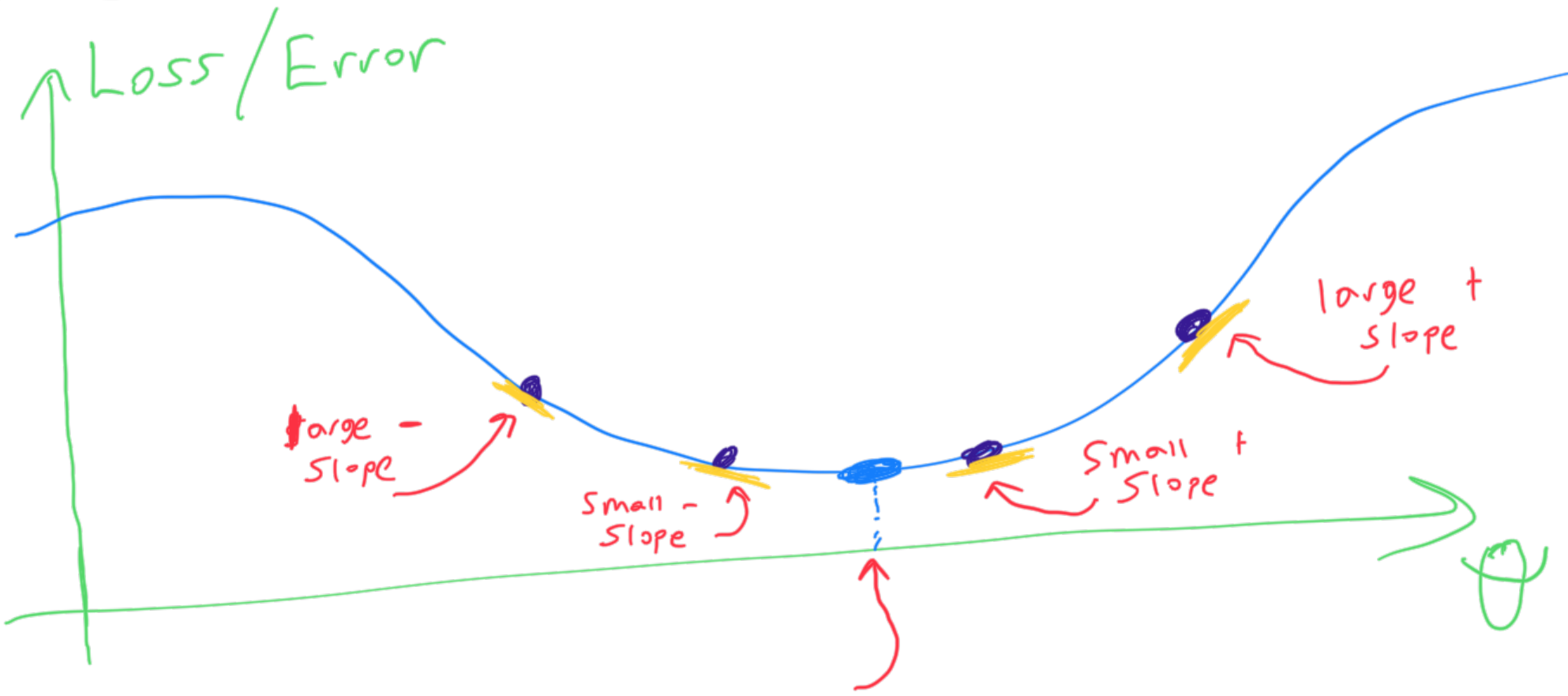
Gradient Descent Steps

1. randomly initialize θ
2. calculate loss with current θ

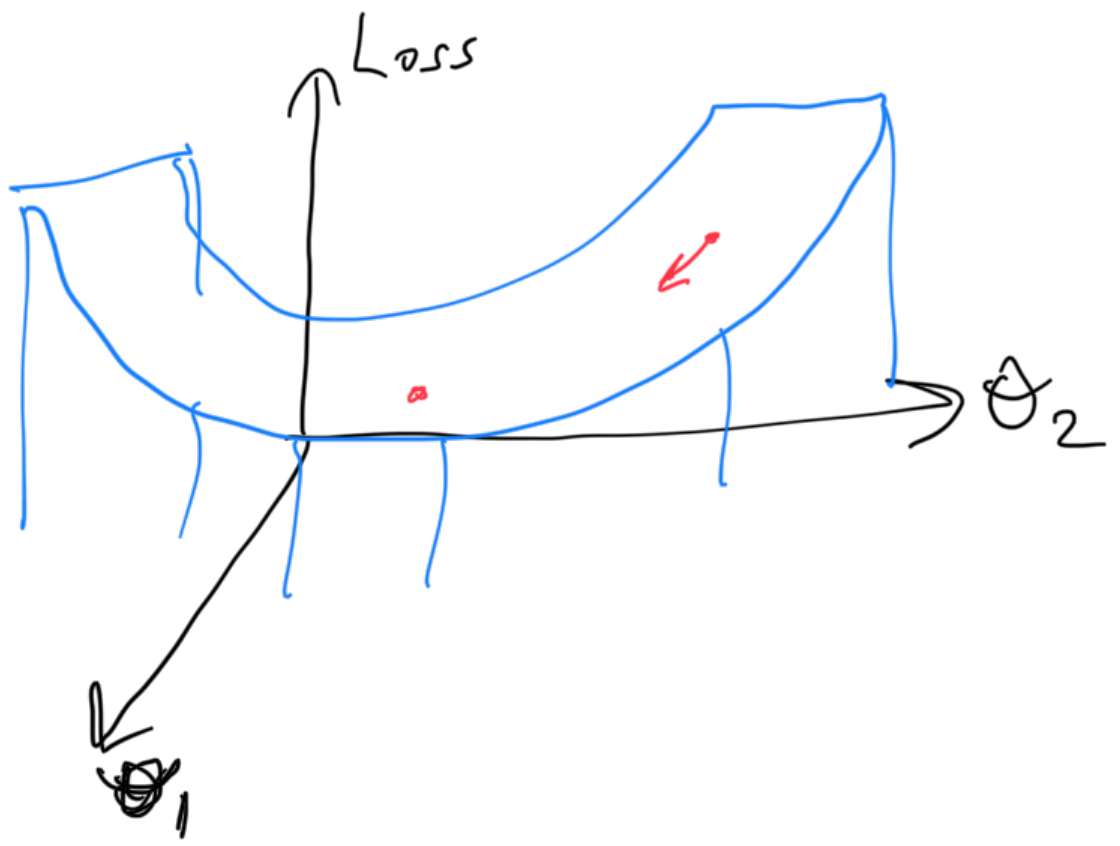
3. update θ in the direction of lower loss using calculus

4. return to step 2 unless stopping criteria are met

Intuition on step 3



We want to move in the direction opposite of the gradient



Equation for how θ are updated:

how much ... at each time step

$$\theta_{i+1,j} = \theta_{i,j} - \alpha \frac{\partial Loss}{\partial \theta_j}$$

move in the opposite direction of the gradient

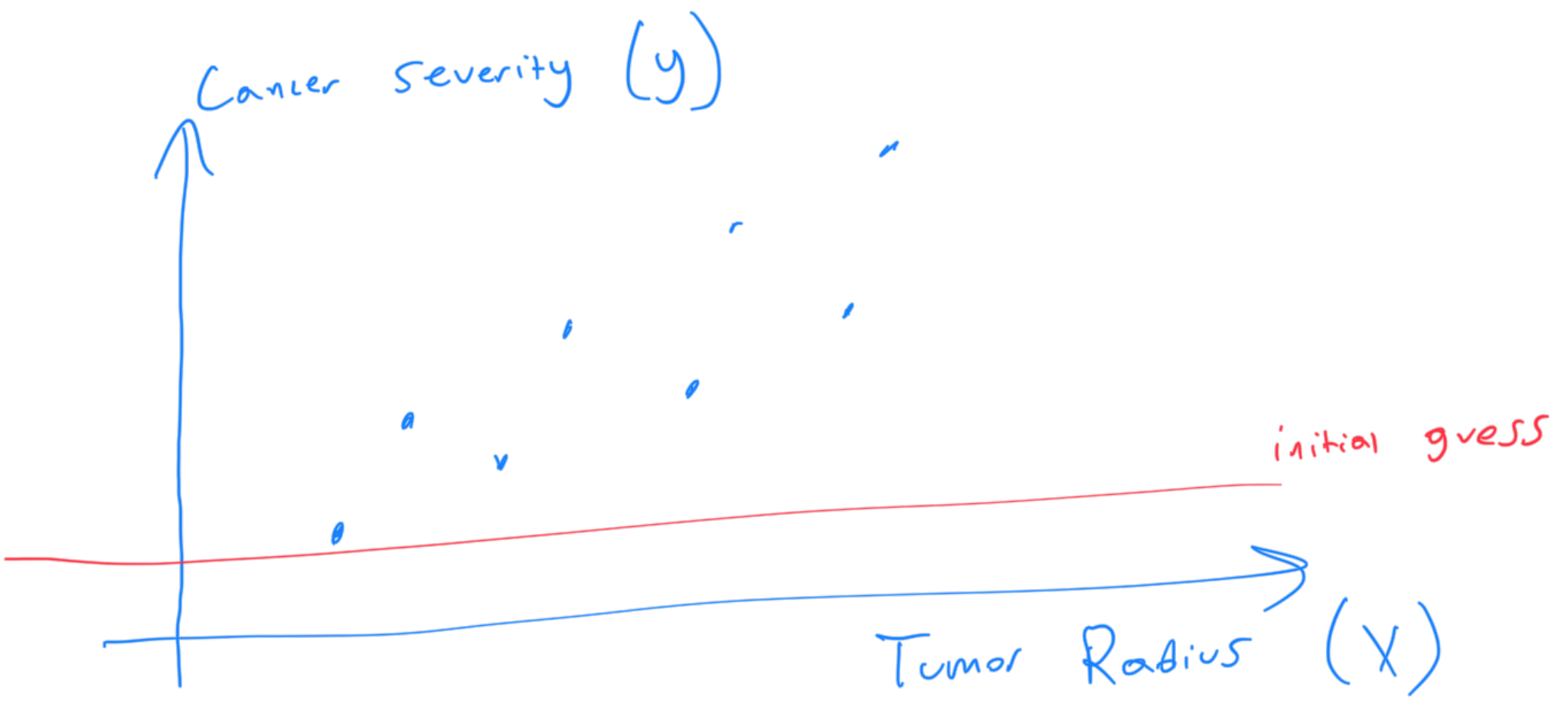
direction of the gradient

$\theta_{i,j}$ = parameter j at iteration/time i

α = "alpha" = learning rate (hyperparameter)

Example:

Step 1:



$$y = \underbrace{0}_{\theta_0} \cdot X + \underbrace{10}_{\theta_1}$$

θ_0 m

θ_1 b

Step 2

(x) Radius	(y) Severity	$\hat{y}$	("square error") SE
0.3	20	10	100
0.6	30	10	400
1.0	35	10	625
1.2	50	10	1600
1.8	38	10	784
2.0	55	10	2025
2.4	60	10	2500
2.6	62	10	2704

$$MSE = \frac{1}{n}\sum_{i=1}^{n} SE_i = 1342.25$$

Step 3

want to calculate:

$$\frac{\partial MSE}{\partial m} \quad \text{and} \quad \frac{\partial MSE}{\partial b}$$

(recall: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$)

$$\frac{\partial MSE}{\partial m} = \frac{-2}{n}\sum_{i=1}^{n} X_i\left(y_i - (mx_i + b)\right)$$

$$\frac{\partial MSE}{\partial b} = \frac{-2}{n}\sum_{i=1}^{n} y_i - (mx_i + b)$$

Plugging in the values from the above table:

$$\frac{\partial MSE}{\partial m} = -120.9$$

$$\frac{\partial MSE}{\partial b} = -67.5$$

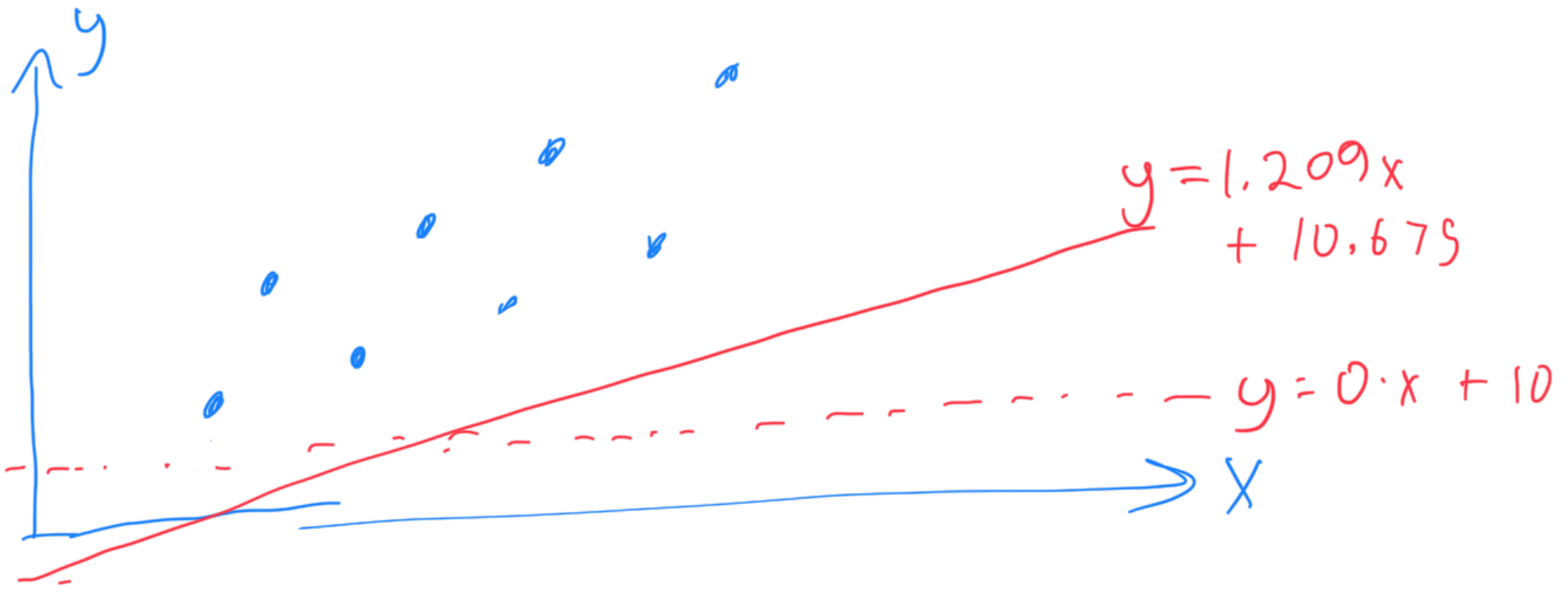
Let's say the ML engineer sets $\alpha = 0.01$:

$$m = m - 0.01(-120.9) = 1.209$$

$$b = b - 0.01(-67.5) = 10.675$$

So now, our updated model is:

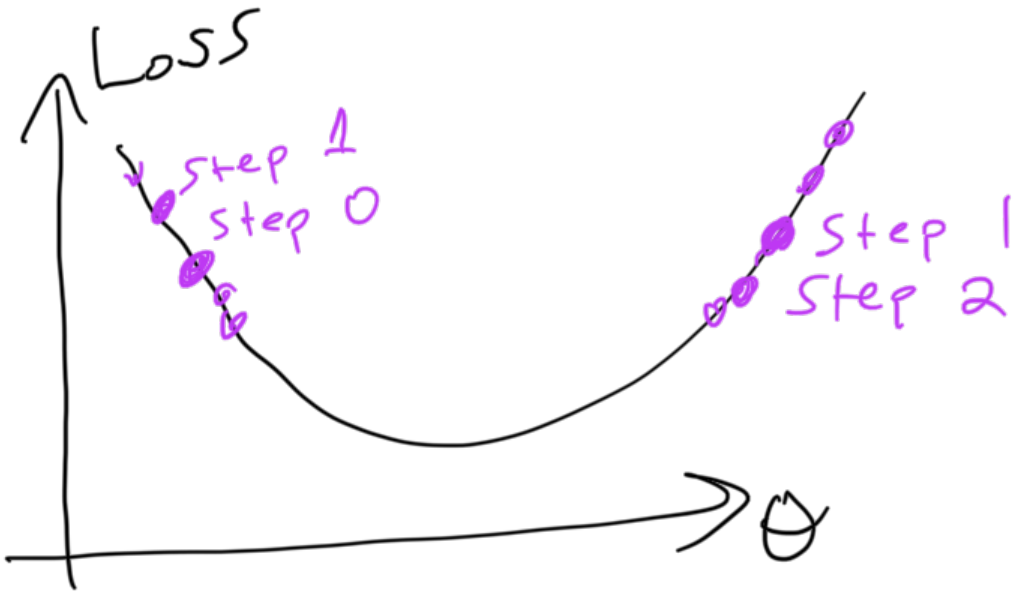
$$y = 1.209x + 10.675$$



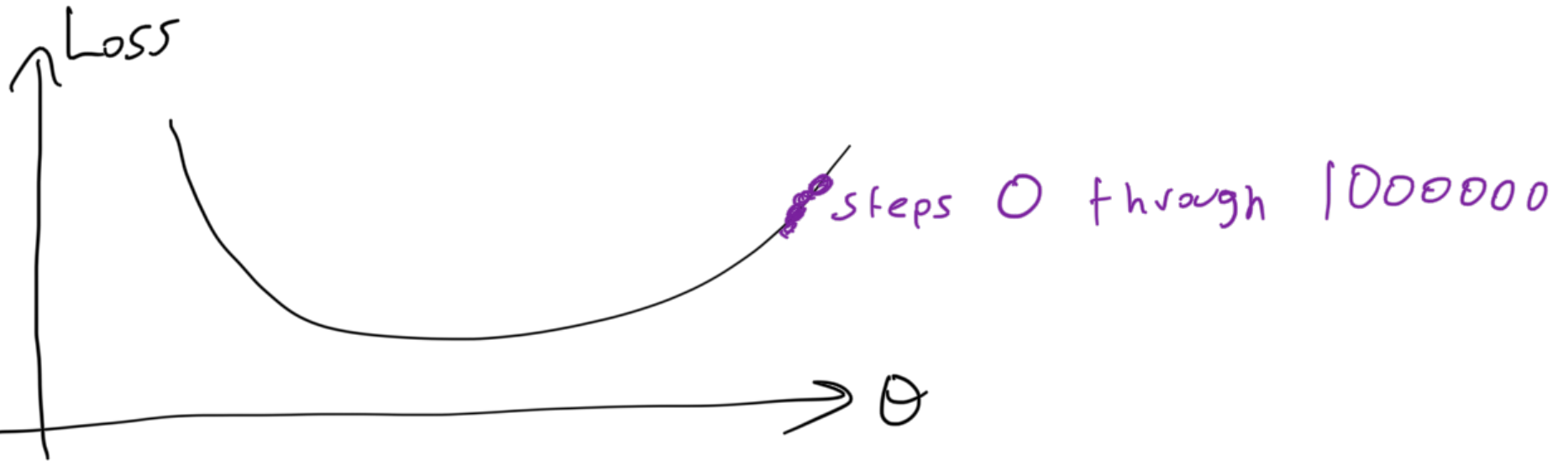
Step 4

Repeat for many iterations until, e.g., loss is low, or num. iterations is too high, or ...

What happens if α is too large?

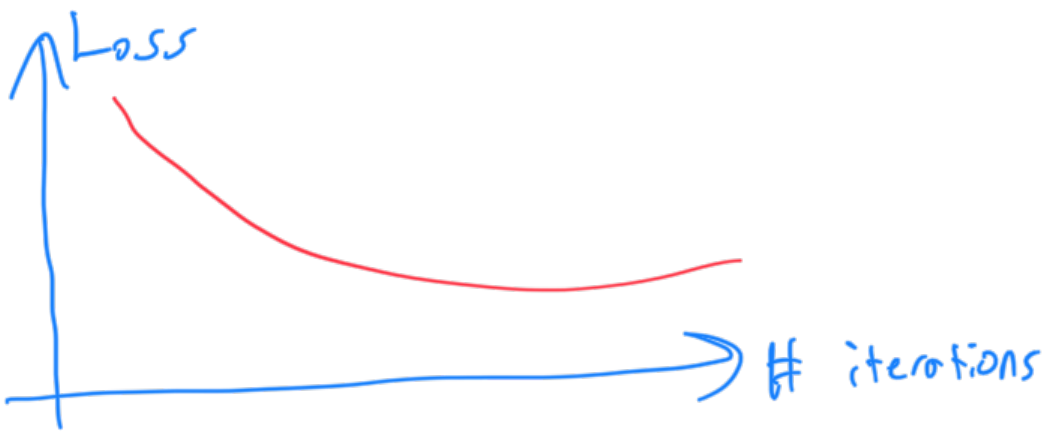


What happens if α is too small?



Gradient Descent Variants

Batch Gradient Descent: use all data when updating θ per iteration

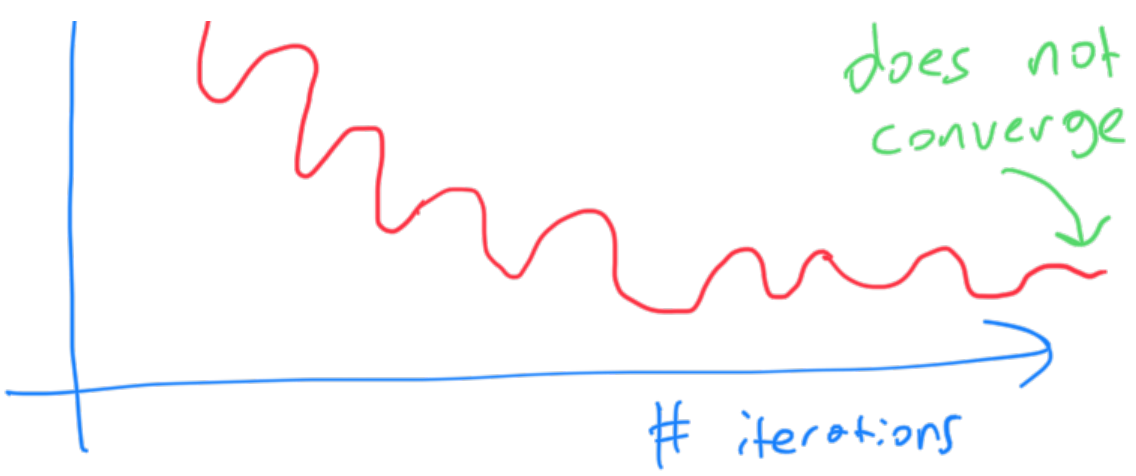


Mini-Batch Gradient Descent: update θ using random subsets ("mini-batches") each iteration

Con:

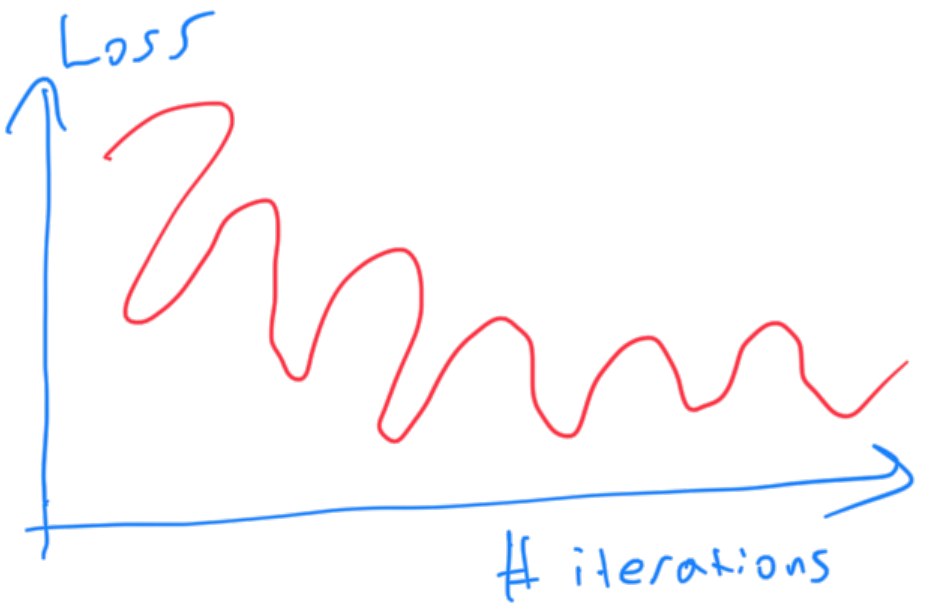
Pros:

* Fast / iterations



* randomness helps reduce overfitting

Stochastic Gradient Descent (SGD): update θ using one data point per iteration

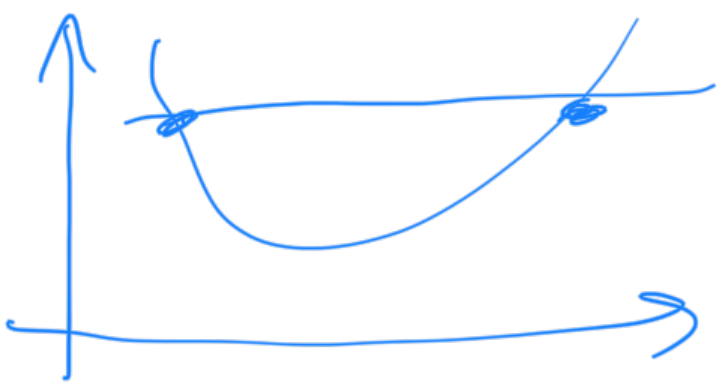


Pros/Cons are same as mini-batch but more extreme

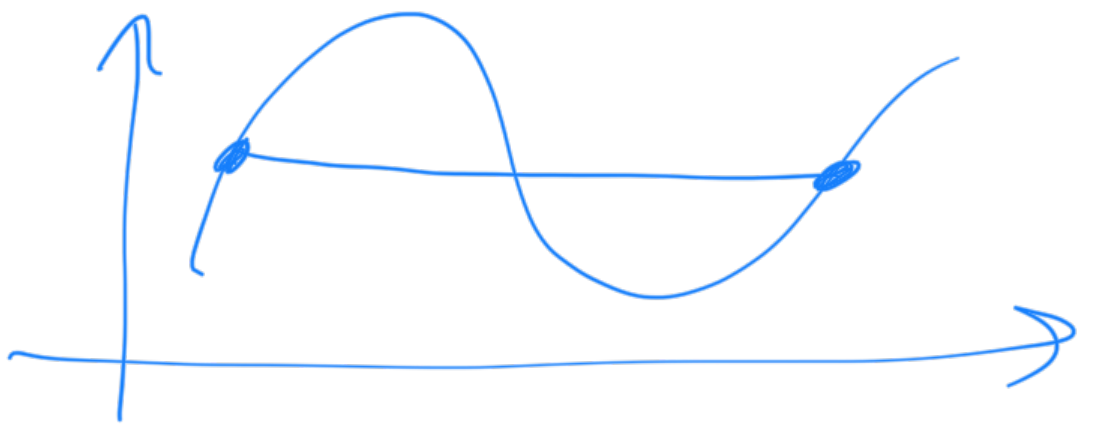
Handling Non-Convexity

Convex Function: line segment between 2 points is above function for all points in between

Convex:

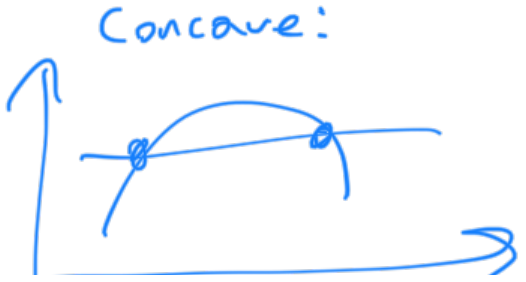


Not Convex or Concave;



Concave Function: same, but below line

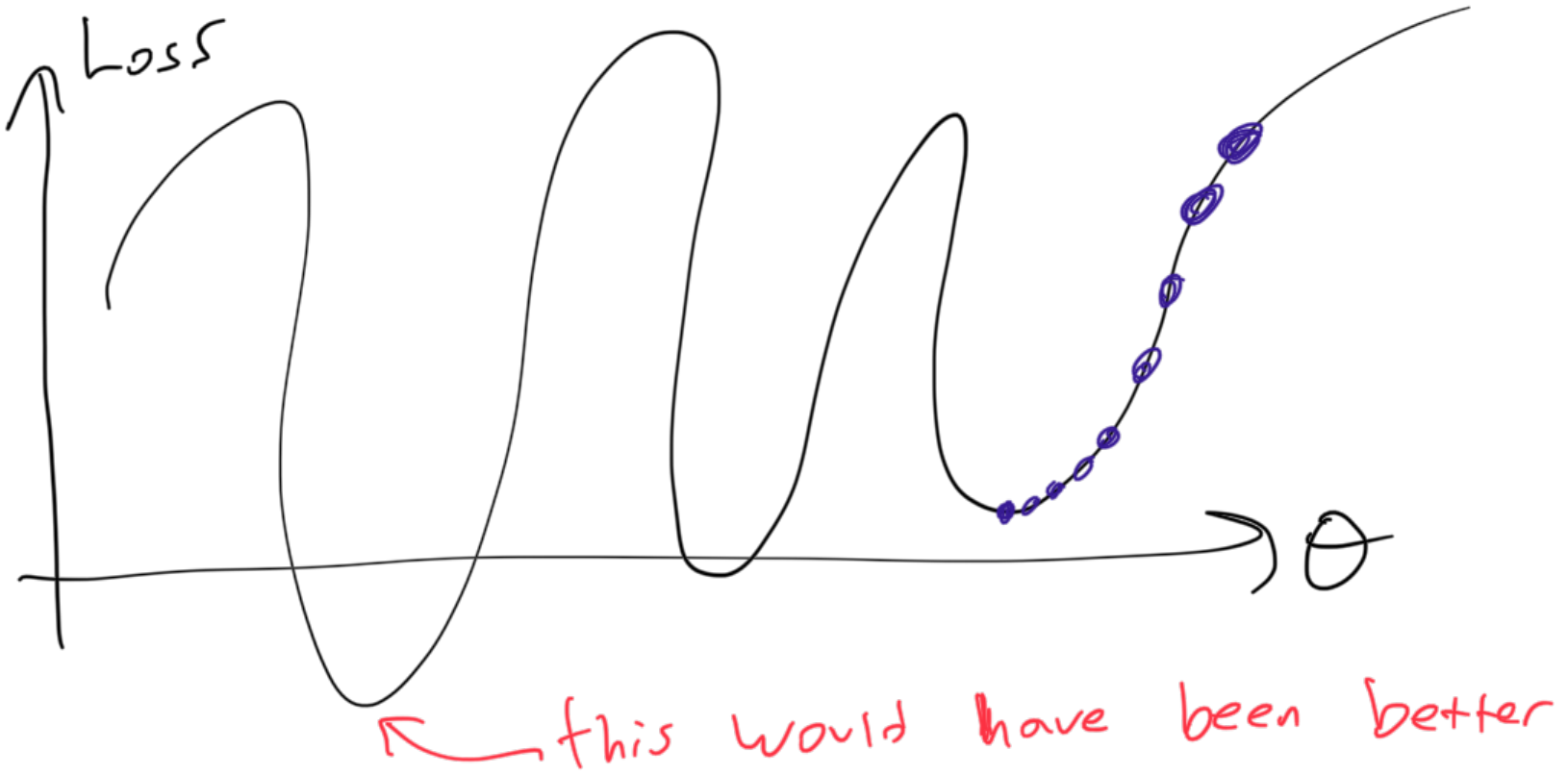
Concave:



In ML, a convex loss function is good

↳ means that we will approach global minimum loss using GD

(Unfortunately, more complex models like Neural networks are not convex)



There are many approaches to try to mitigate this:

momentum:

use history of past θ changes to simulate "momentum"