Linear Regression: the ML perspective

Let us now treat the regression problem from an ML perspective! We have seen that closed form solutions to the LLS exist, but we pretend that they are not available. Instead, we formulate and solve via ML. The steps we are taking and the concepts we introduce are part of (almost) all supervised learning method, in particular Neural Networks → in fact, the algorithm we show here will result in the innermost part of a NN, a linear neuron.

Recap: $\mathbb{D} = \{(x_i, y_i)_{i=1}^{N}\}$: data

$f: x \to y$: generating function

$h: x \to y$: hypothesis

h_θ : model (a specific hypothesis for parameters θ)

Supervised learning: given $\mathbb{D}$, find optimal $\hat{\theta}$ for your choice of h } "training stage"

2.

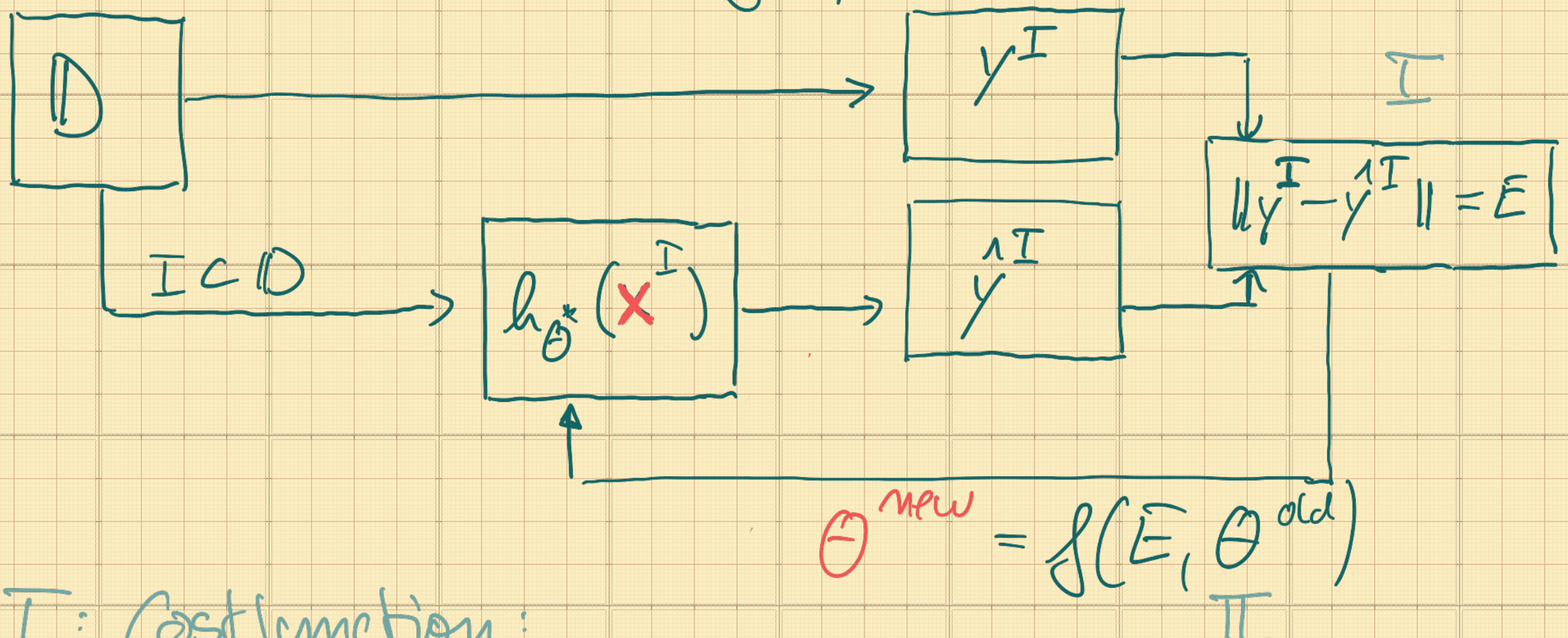
For an $(x_j, y_j) \in \mathbb{P} \cap \mathbb{D} = \emptyset$, $h_{\hat{\theta}}(x_j) = \hat{y}_j$

"inference"

"prediction stage"

- validation data (y_j known)
- test data ($y_j = ?$)

The Supervised Learning cycle: Fair better!

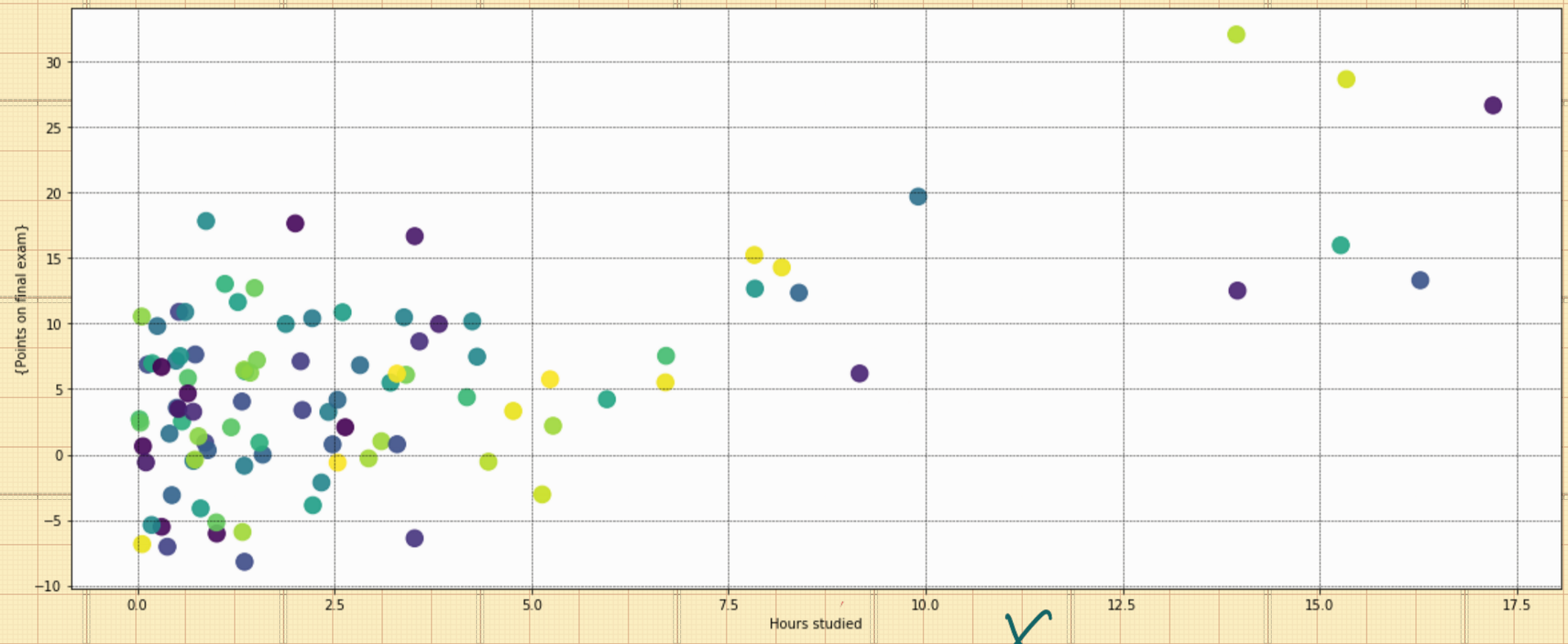


I: Cost function:
How to compute the error

II: How to use knowledge from I to improve θ

Univariate case:

y



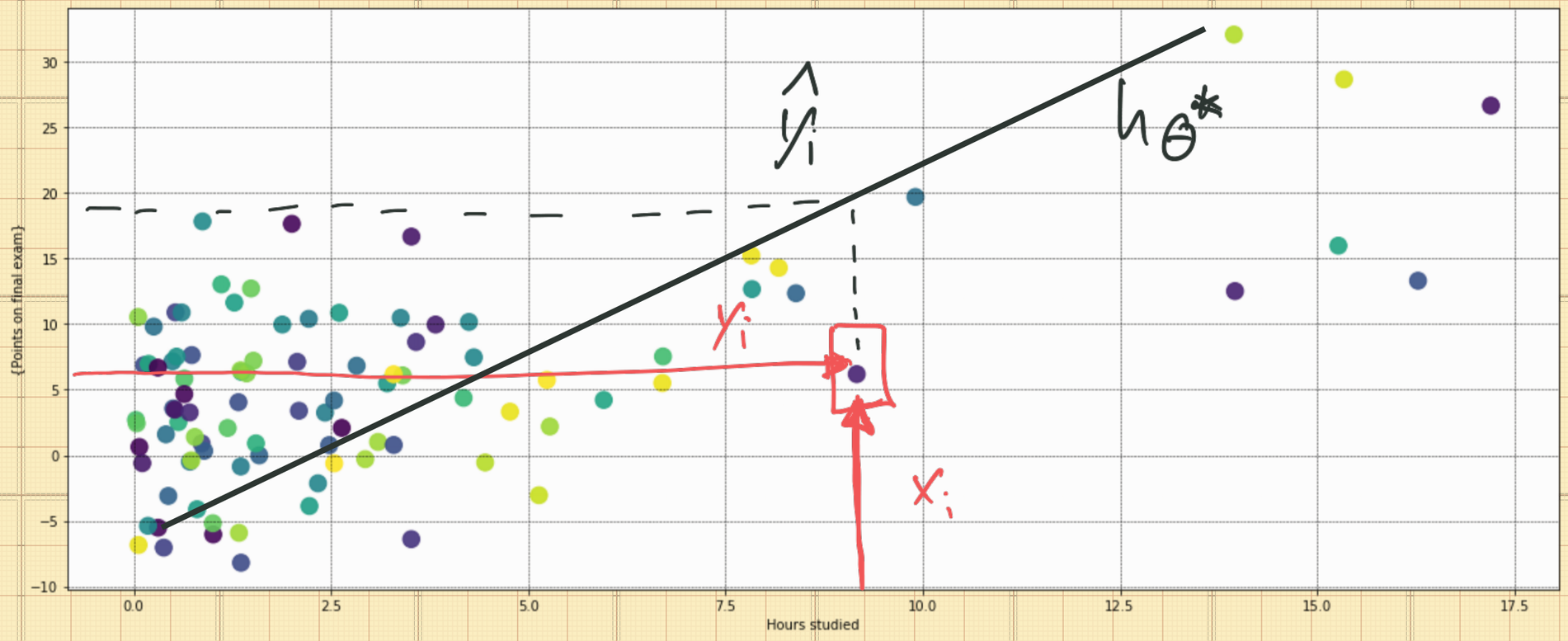
x_1

h: univariate linear model

h_Θ: $y = x^T w = w_1 x_1 + w_0 x_0$ → $= 1$

initial guess for w_0, w_1: random!

For a given sample $\{x_i, y_i\} \in D$, the forward pass with current θ^* gives: $h_{\theta^*}(x_i) = \hat{y}_i$



Measure of failure: L_2 / Square error:

For sample i: $\mathcal{L}_i = (\hat{y}_i - y_i)^2 = J_i$

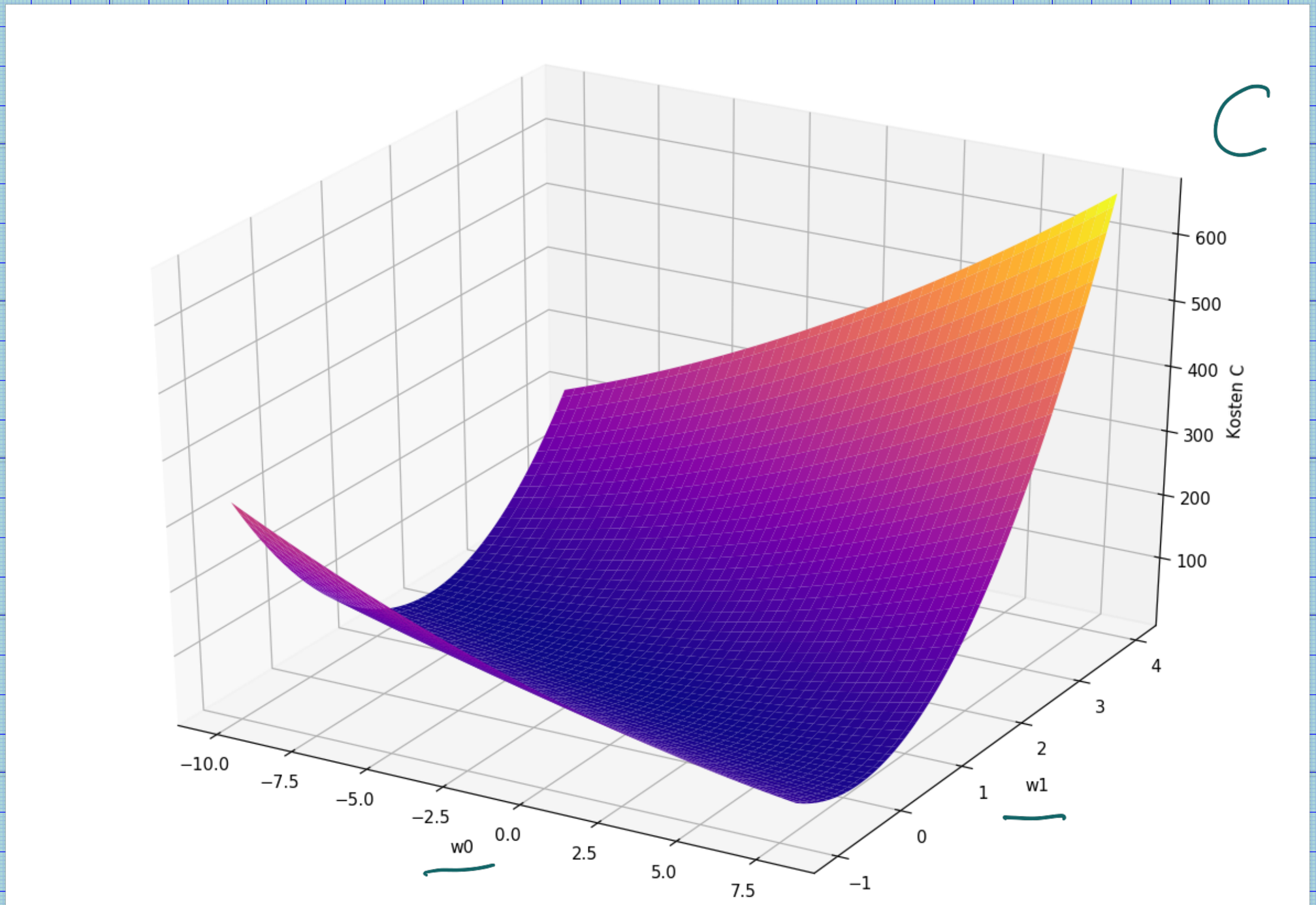
$$C = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

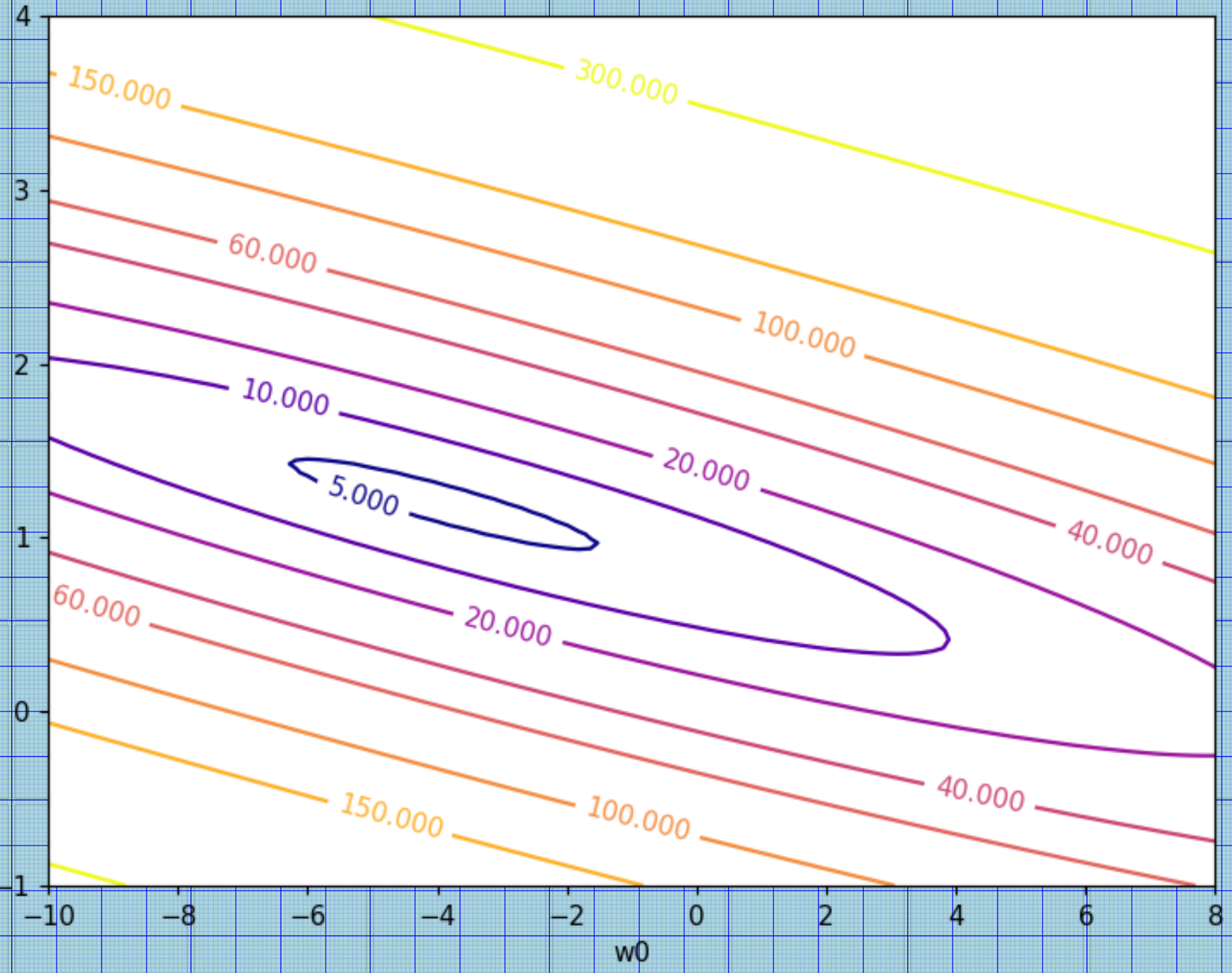
$\frac{dx^2}{dx} = 2x$

no. samples

$$\text{so } C = \frac{1}{2N} \sum_{i=1}^{N} (h_\theta(x_i) - y_i)^2 = C(\theta) = C((w_0, w_1)$$

Cost function, sampled at θ


C
600
500
400
300
200
100
Kosten C
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
w0
−1
0
1
2
3
4
w1


300.000
150.000
100.000
60.000
10.000
20.000
5.000
40.000
60.000
20.000
40.000
150.000
100.000
4
3
2
1
0
−1
−10
−8
−6
−4
−2
0
2
4
6
8
w0
w1

How does the cost function help us to achieve goal II: → Gradient Descent

→ Compute $\nabla_\theta \mathcal{C} = \left(\frac{\partial \mathcal{C}}{\partial w_0}, \frac{\partial \mathcal{C}}{\partial w_1} \right)^T$

$$\frac{\partial \mathcal{C}}{\partial w_j} = \frac{1}{2N} \sum_{i=1}^{N} \frac{\partial}{\partial w_j} \left[\underbrace{\sum_k w_k x_k(x_i)}_{h_\theta(x_i)} - y_i \right]^2$$

linear model: $\frac{\partial w_i x_i}{\partial w_j} = \begin{cases} x_i & : i = j \\ 0 & : i \neq j \end{cases}$

k: Basis functions / features of linear model

j: which parameter / which gradient?

i: sample

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)\, \underbrace{x_j(x_i)}_{\text{j-th basis evaluated a i-th sample}}$$

$\mathcal{L}$: Costfunction, often also called J

Note: This is an average gradient over all samples!

$N = |D|$: Batch Gradient Descent

$N < |D|$: Mini-Batch GD

$N = 1$: Stochastic GD

→ Simple optimization of C w.r.t. Θ:
iterative gradient descent:

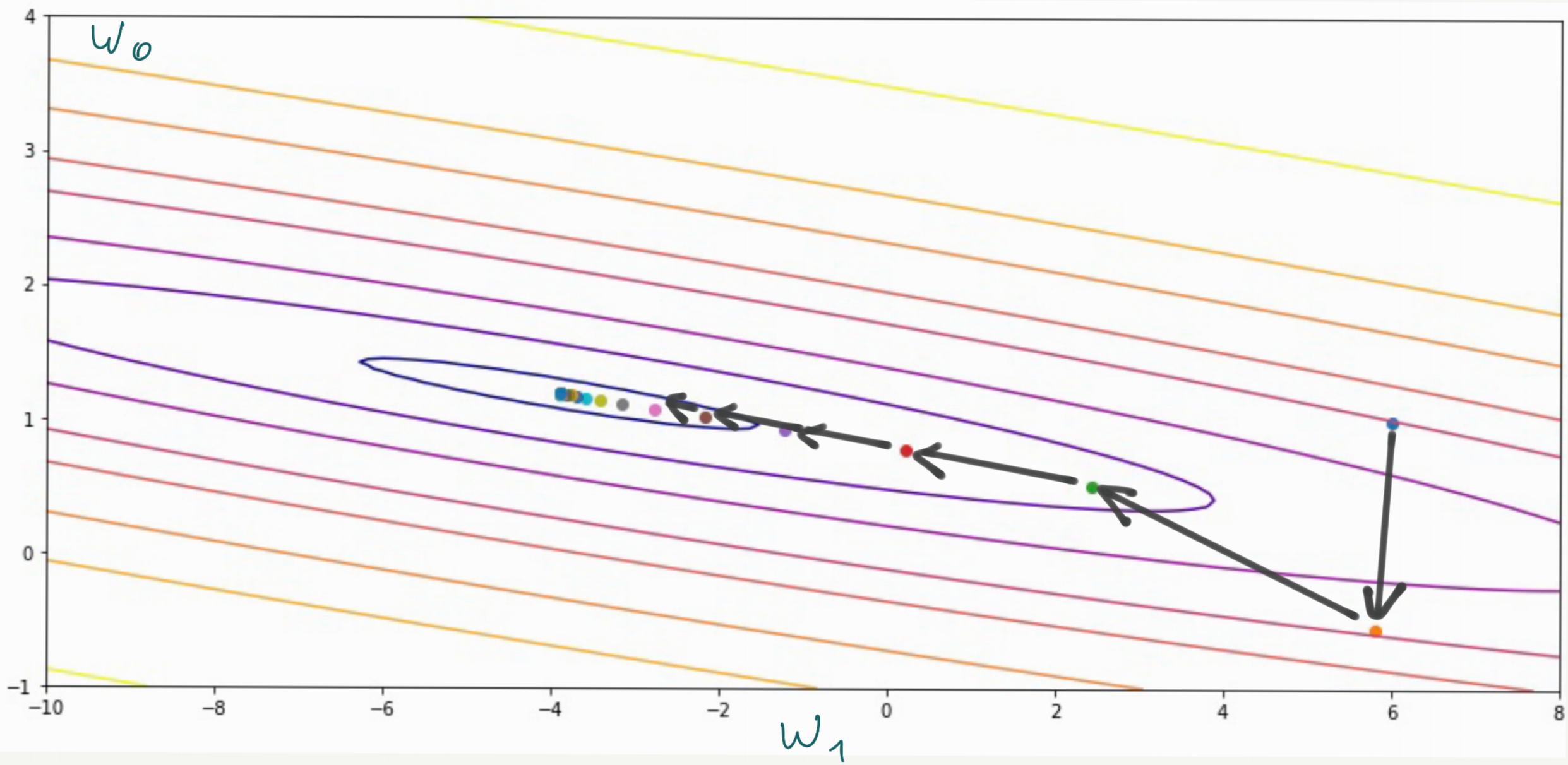
$$w_0^{next} = w_0^{old} - \alpha \frac{\partial C}{\partial w_0}$$

$$w_1^{next} = w_1^{old} - \alpha \frac{\partial C}{\partial w_1}$$

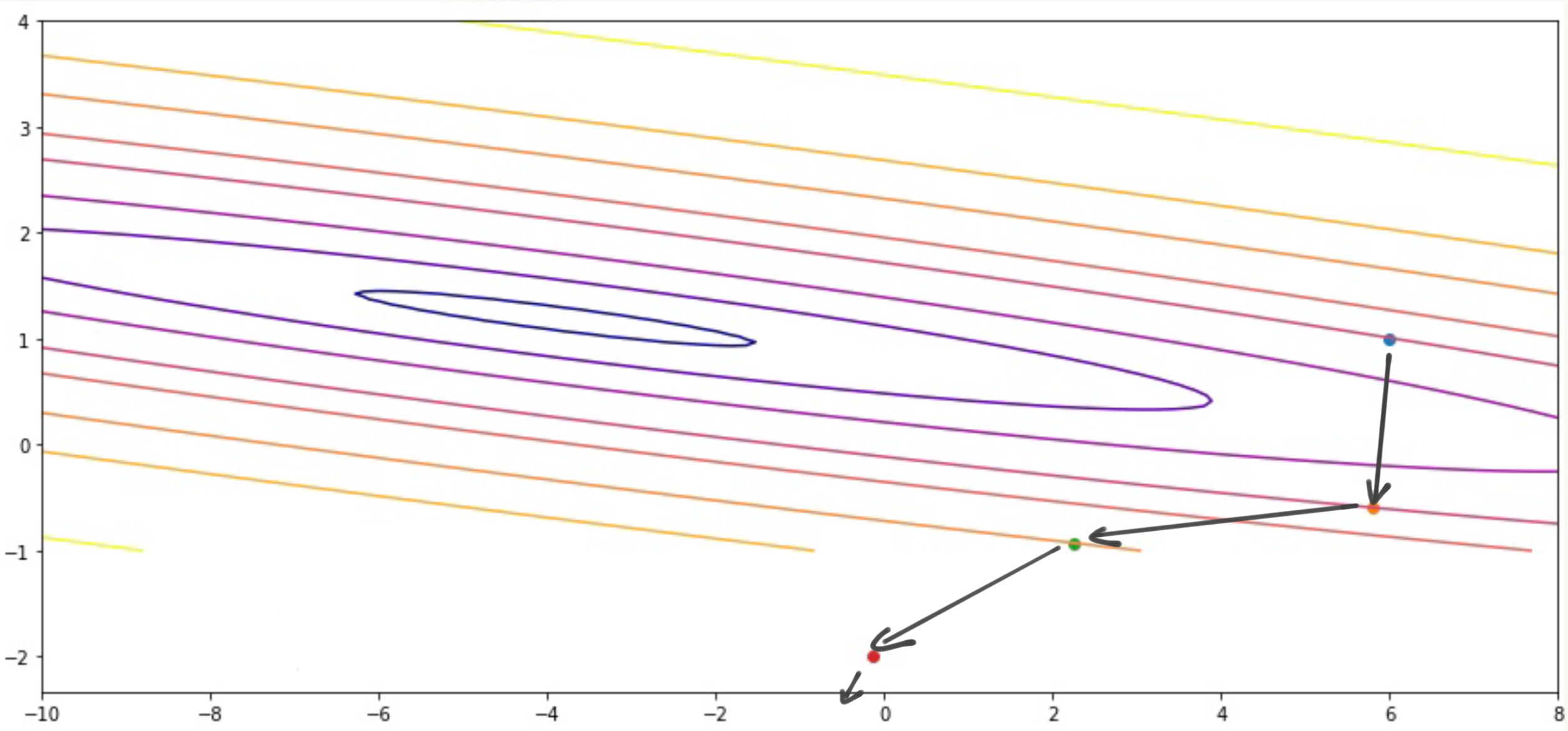
step size: "learning rate"
⇒ hyperparameter

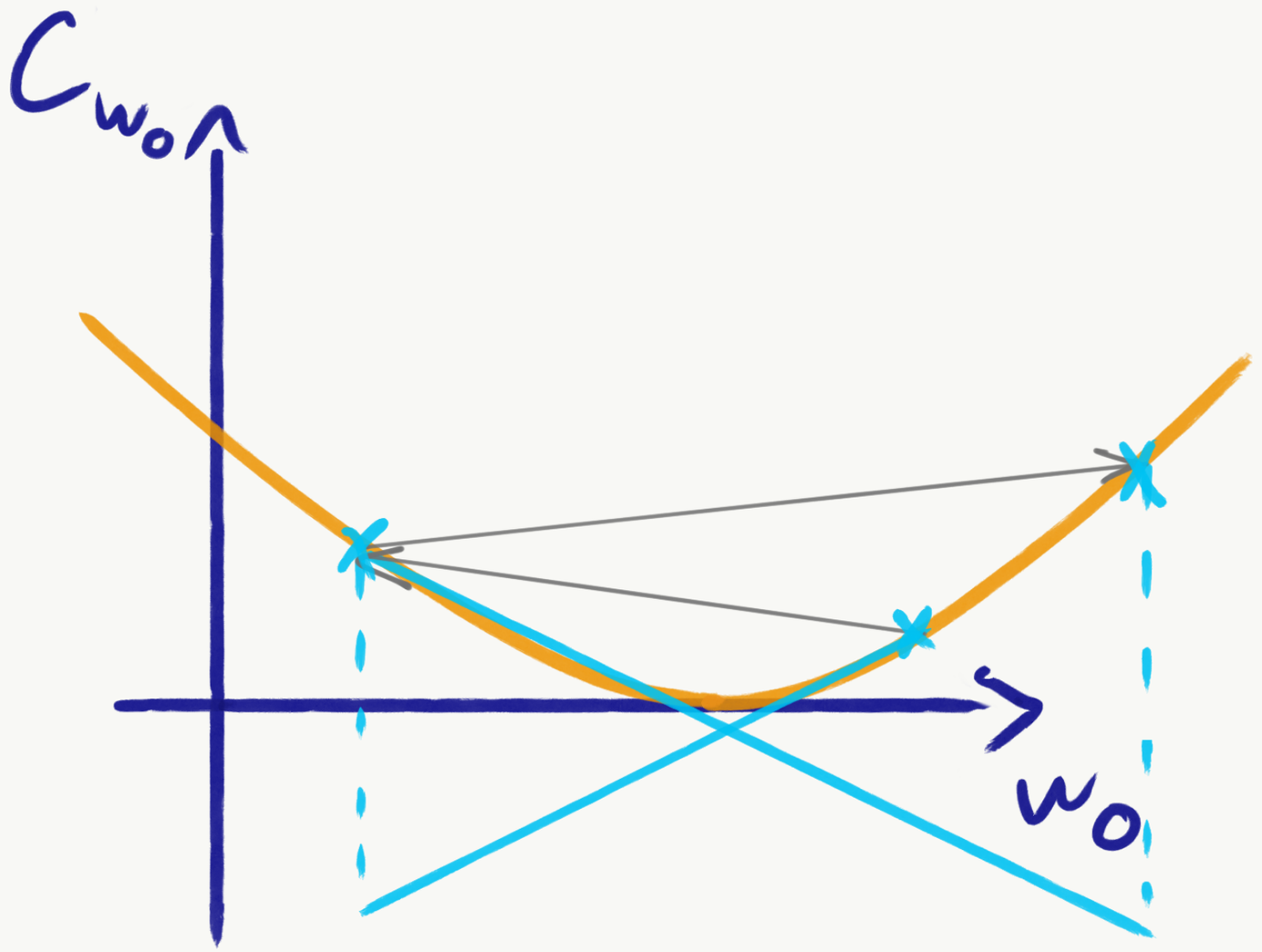
iterate and update!

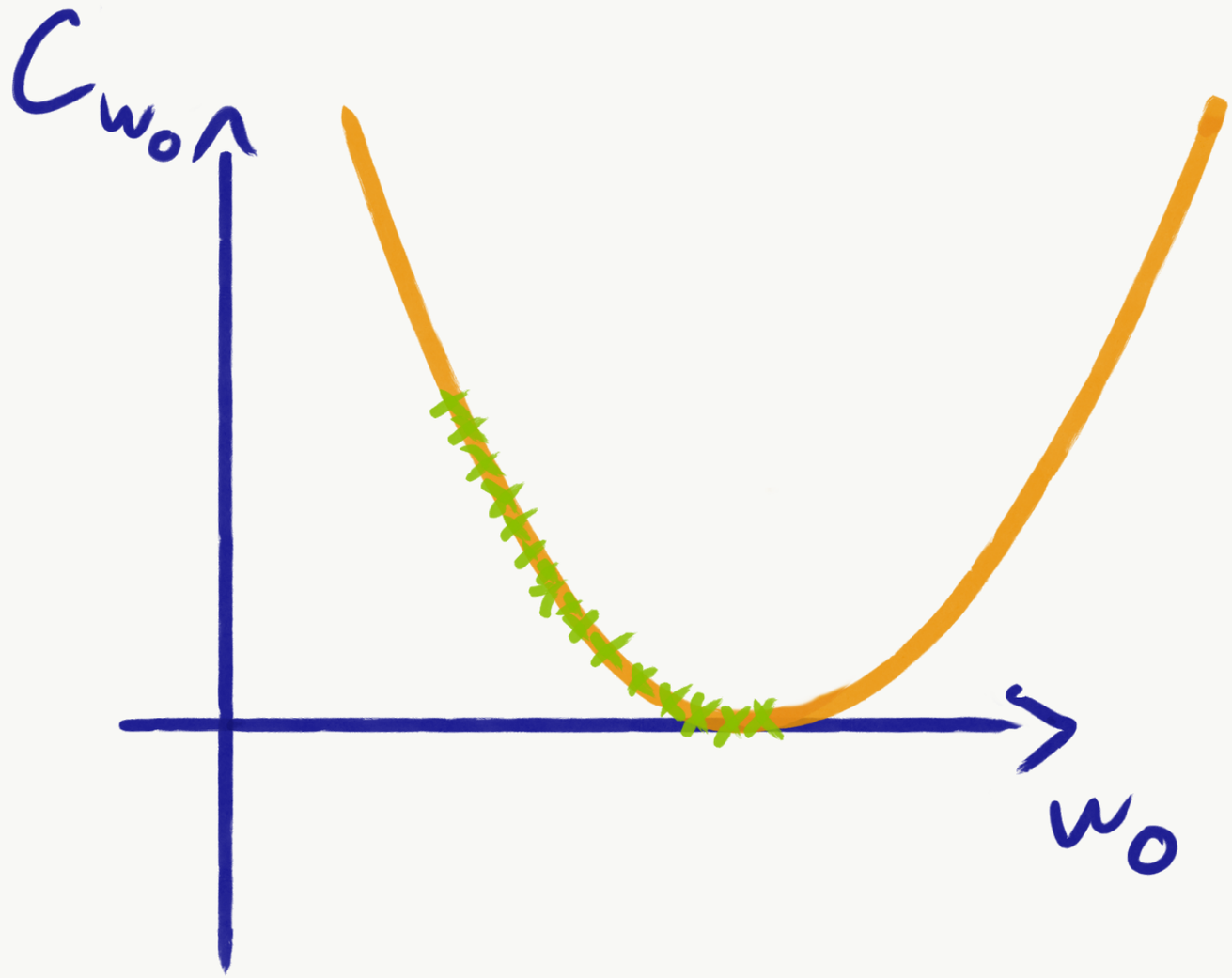
Succesful optimization

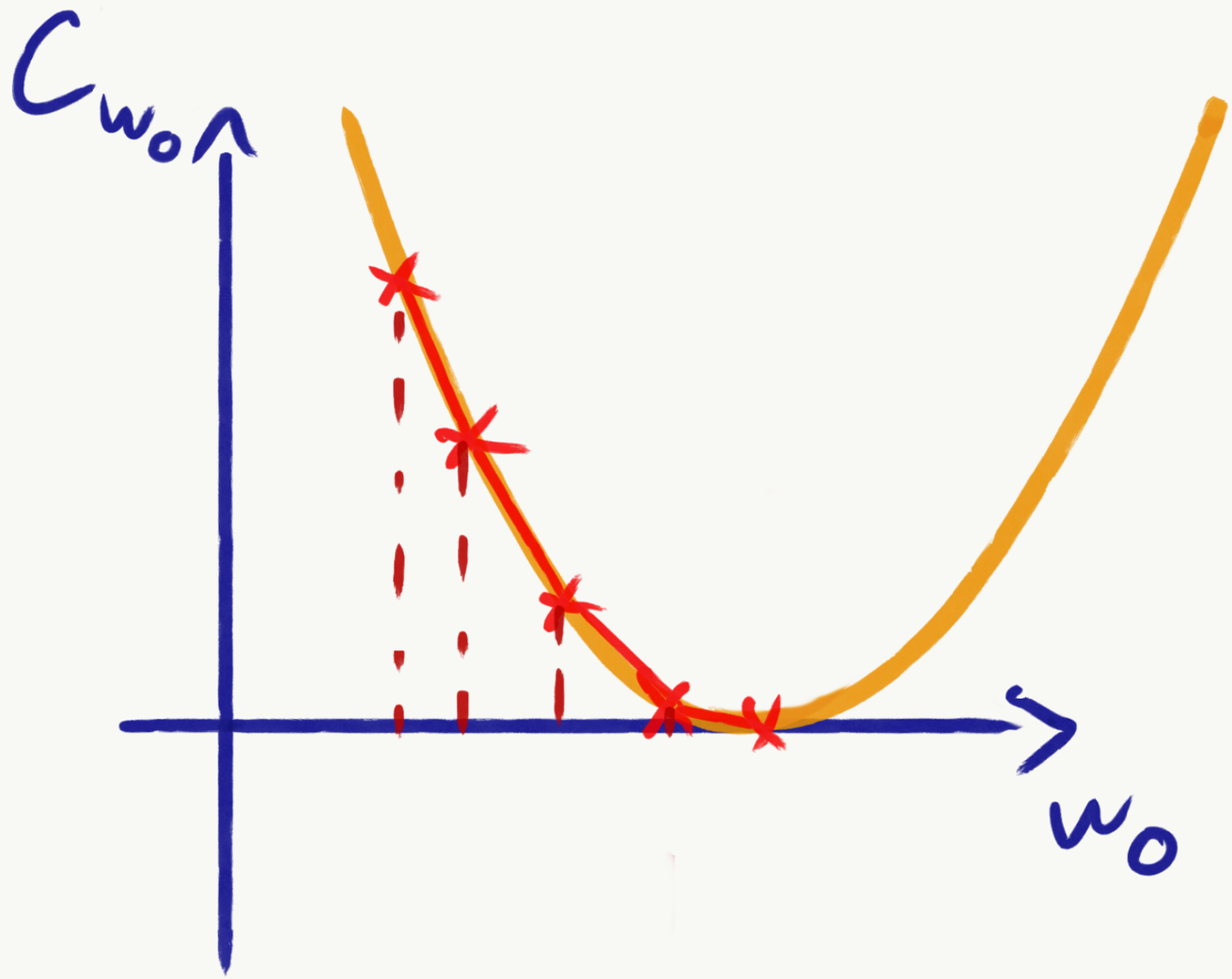


Learning Rate too large → diverging




ω_{on}
ω_o


C_{WOA}
ω_0


C_{WO}
w_0

- Note that if done right, this method converges to the LLS we discussed before!
- We can increase the expressiveness by
  a) more features: $w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots$
  b) high order ansatz: $x^2, x^3, x_1 \cdot x_2, \dots$
- Multivariate LS: nothing new!
- Linear Neuron as the building block of NNs: