Welcome to shallow ML!

Types of Learning → Supervised Learning

3 points of view

Linear Regression

ML prob. LA

↳ Linear Neuron → Classification

Logistic Neuron ← Logistic Regression

→ Assignment IV

Types of Learning

1.) unsupervised learning: descriptive model
unlabeled data; find lower dimensional representation / model of data $P(x|\mathbb{D})$

2.) Supervised learning: predictive model
Mapping from input to output given data $\mathbb{D}$

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

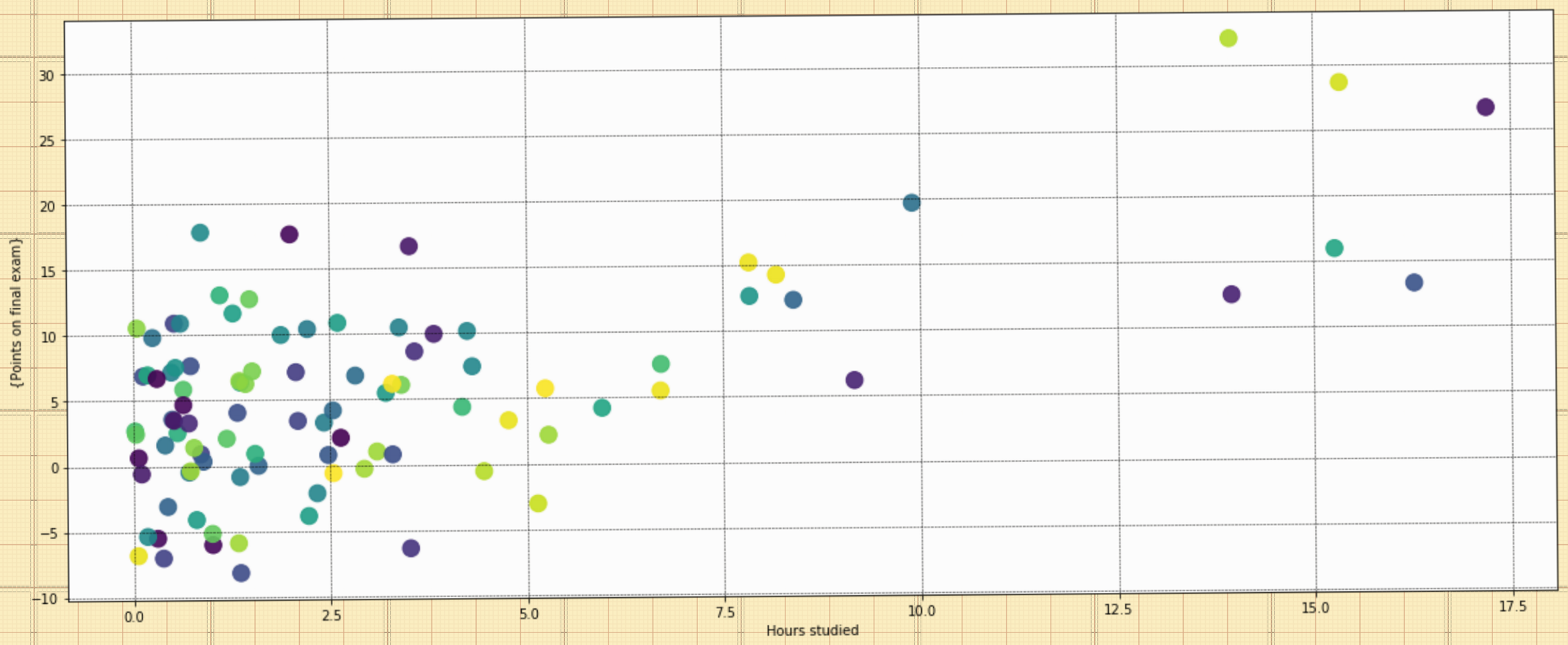
$p(y|x, \mathbb{D})$ or $f(x, \mathbb{D})$ → model as functions

↳ model as a probability distribution

3.) Reinforcement, semi-supervised...

Supervised Learning: Nomenclature

A sample problem: Grade on final test



$x_i \in X$: Input : hours spent studying

$y_i \in Y$: Output : points on test

$f : X \to Y$: Generating function : the truth

$\{(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)\} =: \mathbb{D}$: data

$h : X \to Y$: hypothesis set

θ : parameters : trainable

$h_\theta : X \xrightarrow{\theta} Y$: Model : a specific choice from $\{h\}$

$h_\theta(x^*)$: Prediction

Generating function f (unknown!)

Data $\mathcal{D}$

Hypothesis set $\{h\}$

ML

Final model $h_\theta \approx f$

Model h_θ

$-\ominus$

[Y. S. Abu Mostafa, CalTech]

Linear Regression: 3 Approaches

↳ continuous functions ↔ classification

$f: x \to y$

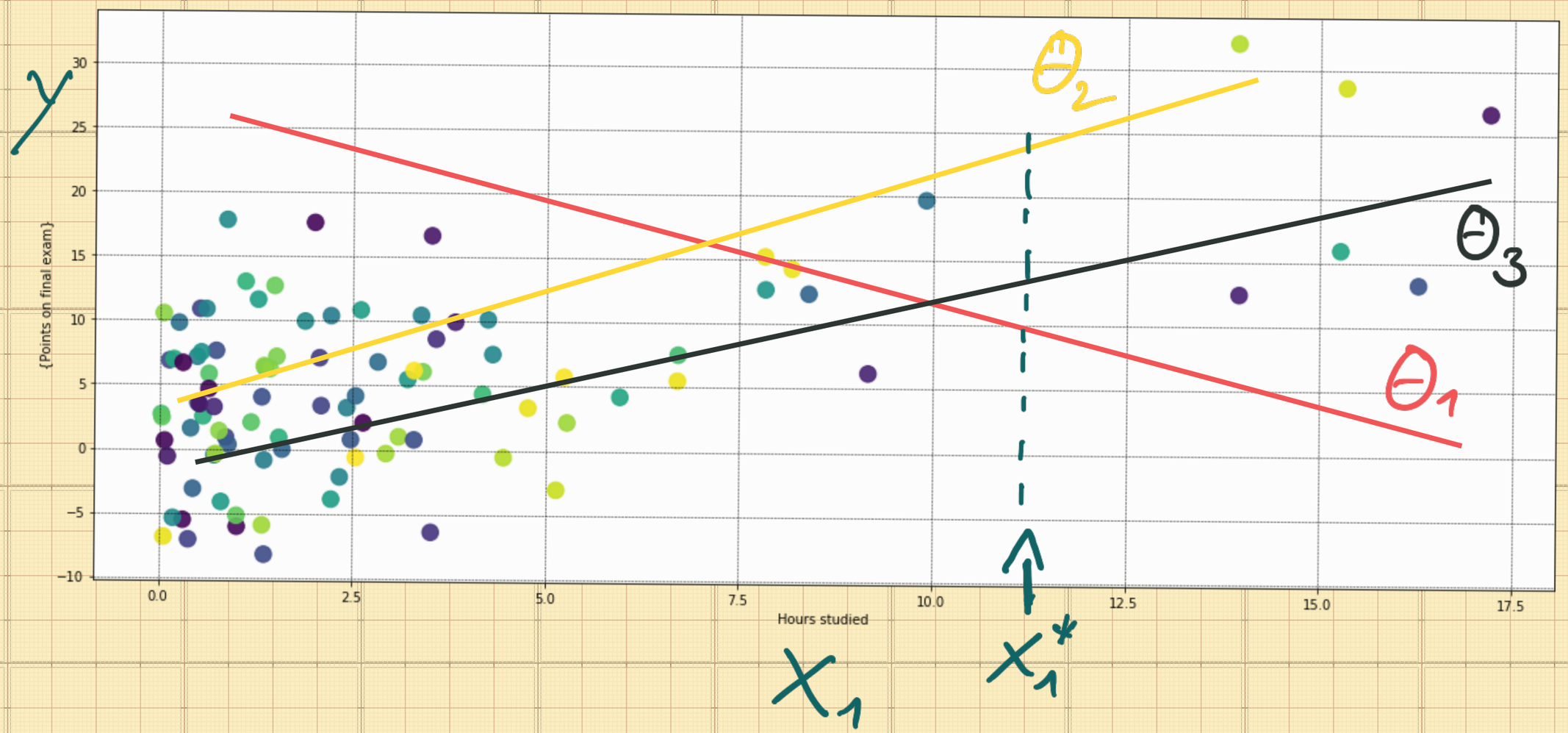
$h: y = x^T\theta = \sum_{i=1}^{p} w_i x_i$

We consider the univariate case here where we have a single feature x_1:

$$h = w_1 x_1 + w_0 x_0 = w_1 x_1 + \underbrace{w_0 \cdot 1}_{\text{bias term}}$$

Linear: refers to the parameters!

$h = w_1 x_1^2 + w_0 x_0$ is still LR (with feature engineering)


y
{Points on final exam}
30
25
20
15
10
5
0
−5
−10
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Hours studied
θ_2
θ_3
θ_1
x_1
x_1^*

Approach 1: Classical Linear Algebra

→ Linear Least Squares → Normal form

Approach 2: Max. Likelihood Estimate

→ Gaussian Noise → Normal form

Approach 3: Iterative Learning

→ Square Loss → approx. LLS

→ Most famous shallow ML algorithm!

The linear algebra perspective

(heavily inspired by Pavel Grinstein)

A): Quadratic Form Minimization:

A function $f: \mathbb{R}^N \to \mathbb{R}$ of the form

$$f(\underline{x}) = \frac{1}{2}\underline{x}^T \underline{\underline{A}}\, \underline{x} - \underline{x}^T \underline{b}$$

with $\underline{\underline{A}}$ square, positive definite

is called the "quadratic form" (with linear extension).

Then, $\min_{\underline{x}} f(\underline{x}) : \underline{\underline{A}}\, \underline{x} = \underline{b}$

One way to show this: $x \in \mathbb{R}^3$, compute $\nabla f = 0$

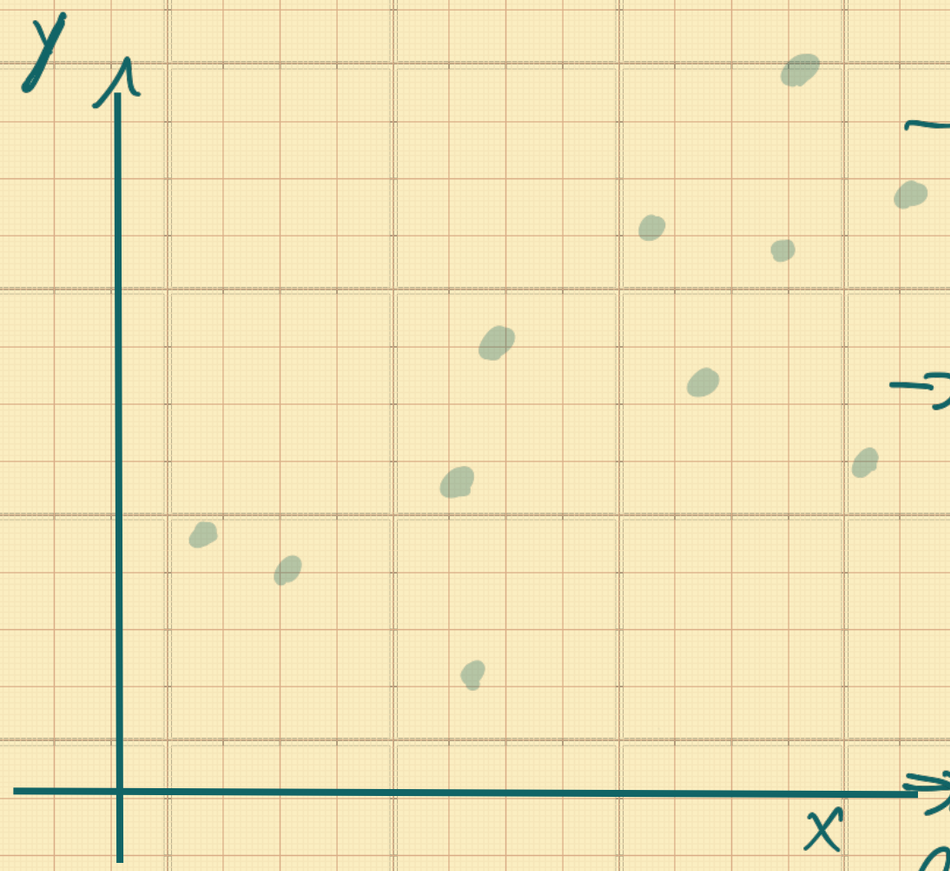
→ solving the linear system $\boxed{\underline{\underline{A}}\,\underline{x} = \underline{b}}$ minimizes the associate quadratic form $\frac{1}{2}\underline{x}^T\underline{\underline{A}}\,\underline{x} - \underline{x}^T\underline{b}$!

→ scalar case: $f(x) = \frac{1}{2}ax^2 - bx$,

$f_x = ax - b \doteq 0$

$\Rightarrow \boxed{ax = b}$

B) Linear least squares



hypothesis space

→ Find a best linear fit to the data $\{(x_i, y_i)\}_{i=1}^{N} =: \mathbb{D}$

→ $h_\Theta(x) = w_1 x + w_0 \cdot 1$

$\Theta = (w_1, w_0)^T$

Basis functions

→ how to fit the model $h_\Theta(x)$ on the data $\mathbb{D}$?

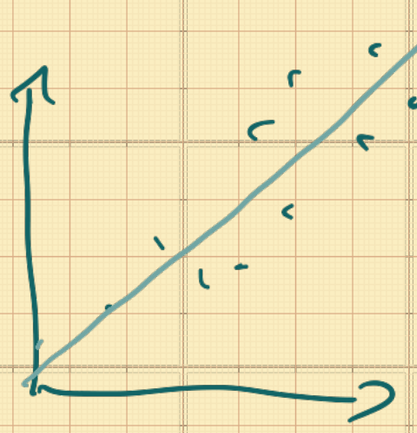
→ Interpolation leads to an overdetermined system:

$$y_i \overset{!}{=} w_1 x_i + w_0 \cdot 1 \qquad \forall i = 1, \dots, N$$

2 unknowns, N constraints

in matrix vector form with $x = \phi_1$, $1 = \phi_0$:

$$\begin{pmatrix} \phi_1(x_1) & \phi_0(x_1) \\ \phi_1(x_2) & \vdots \\ \phi_1(x_3) & \vdots \\ \vdots & \vdots \\ \phi_1(x_N) & \phi_0(x_N) \end{pmatrix} \begin{pmatrix} w_1 \\ w_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ \vdots \\ y_N \end{pmatrix}$$



↳ VanderMonde-Matrix

$$\underline{\underline{A}} \quad \underline{x} = \underline{b}$$

→ cannot be inverted exactly!

→ What can we say about the error?

$$\underline{r} = \underline{b} - \underline{\underline{A}}\,\underline{x}$$

→ need norm for $\underline{r}$ → inner product!

$$\underline{r}^T\underline{r} = (\underline{b} - \underline{\underline{A}}\,\underline{x})^T(\underline{b} - \underline{\underline{A}}\,\underline{x})$$

$$= \underline{x}^T\underline{\underline{A}}^T\underline{\underline{A}}\,\underline{x} - \underline{x}^T\underline{\underline{A}}^T\underline{b} - \underline{b}^T\underline{\underline{A}}\,\underline{x} + \underline{b}^T\underline{b}$$

$$\underline{b}^T(\underline{x}^T\underline{\underline{A}}^T)^T = \underline{b}^T(\underline{\underline{A}}\,\underline{x})$$

$$= 2\left(\frac{1}{2}\underline{x}^T\underline{\underline{A}}^T\underline{\underline{A}}\,\underline{x} - \underline{x}^T\underline{\underline{A}}^T\underline{b}\right) + \underline{b}^T\underline{b}$$

$\underline{\underline{A}}^T\underline{\underline{A}}$ = "$\tilde{A}$"

$\underline{\underline{A}}^T\underline{b}$ = "$\tilde{b}$"

$\underline{b}^T\underline{b}$ = const.

$$= 2\cdot\left(\frac{1}{2}\underline{x}^T\underline{\underline{\tilde{A}}}\,\underline{x} - \underline{x}^T\underline{\tilde{b}}\right) + \text{const.}$$

→ The term in brackets is a <u>quadratic form!</u>
(the position of the minimum does not change through addition of a constant or scaling)

→ This error becomes minimal if we thus solve $\underline{\underline{\tilde{A}}}\,\underline{x} = \underline{\tilde{b}}$

or $\underline{\underline{A}}^T\underline{\underline{A}}\,\underline{x} = \underline{\underline{A}}^T\underline{b}$

$$\Rightarrow \underline{x} = \frac{\underline{\underline{A}}^T\underline{b}}{\underline{\underline{A}}^T\underline{\underline{A}}}$$

$$\begin{pmatrix} w_1 \\ w_0 \end{pmatrix} = \Theta^T$$

→ The famous normal equation of linear least squares!

What did we learn from this?

- Solving a linear system minimizes the associated quadratic form
- Thus, for a linear hypothesis, a quadratic error is natural choice
- A closed form solution to the linear least squares problem is given by the normal form
- This is thus a nice test case for learning algorithms!

- The normal form corresponds to a discrete L_2-projection onto P_1 with standard inner product:

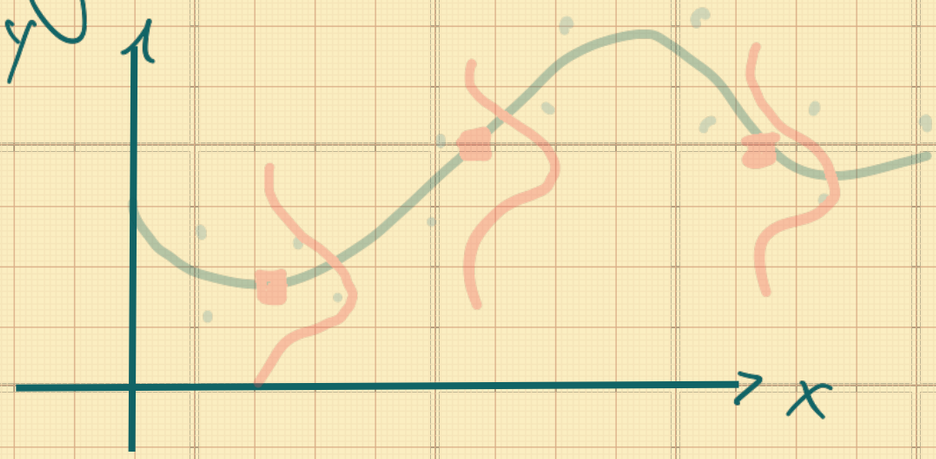
Linear Least Squares

Probalistic p.o.v.: signal/data contains noise, e.g. measurement error

→ regression with likelihood:

$$p(y|x) = \mathcal{N}(y \mid f(x), \sigma^2)$$

here, x is again the input, y is the output/target and we have $y = f(x) + \varepsilon$, where ε is an i.i.d. Gaussian noise with zero μ and variance σ^2



- Note: $f(x)$ is the hidden generating function, that we can only observe through samplings
- Note: we assume that σ^2 is known and constant
- Linear regression means linear in the parameters ☺! (features can be combined non-linear(y):

$$f(x) \approx w_1 x + w_0 \cdot 1 = x^T \theta$$

$$= \begin{pmatrix} x \\ 1 \end{pmatrix}^T \begin{pmatrix} w_1 \\ w_0 \end{pmatrix}$$

features → x, parameters → θ

so: $p(y|x,\theta) = \mathcal{N}(y|x^T\theta, \sigma^2)$, *

$y = f(x) + \varepsilon = x^T\theta + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$

linear in θ

Remark: if $\sigma^2 \to 0$: $p(y|x,\theta) \to \delta$

How is (*) useful in finding a model?

=> Classic estimator: MLE (see lecture statistics II)

maximizing the likelihood / prediction of the training data $\mathbb{D}$ given the unknown, but fixed parameters θ:

$$\theta_{MLE} := \arg\max_{\theta} p(\mathbb{D}, \theta)$$

so more specifically:

$$\theta_{MLE} = \arg\max_{\theta} p(\underline{Y} | \underline{X}, \theta)$$

feature

set of N training samples $\{$ $(y_1, x_{21} \dots y_N)^T$ $\rightarrow$ $\begin{pmatrix} x_1^0 & x_1^1 \\ x_2^0 & x_2^1 \\ \vdots & \\ x_N^0 & x_N^1 \dots \end{pmatrix}$

Since we assume the samples to be i.i.d., we can factorize the likelihood as:

$$p(\underline{Y} | \underline{X}, \theta) = \prod_{n=1}^{N} p(y_n | x_n, \theta)$$

$$= \prod_{n=1}^{N} \mathcal{N}(y_n | x_n^T \theta, \sigma^2) \quad **$$

Remark: $p(\underline{Y} | \underline{X}, \Theta)$ is not a pdf in Θ, but it is one in $\underline{Y}$

Instead of maximizing (**), it is numerically easier to minimize the log-likelihood. Since $\log(x)$ is strictly monotone, the extrema of x are identical to those of $\log(x)$!

[MMLB, 9.2]

NLL: $$NLL(\Theta) := -\log p(\underline{Y} | \underline{X}, \Theta)$$
$$= -\log \prod_{n=1}^{N} p(y_n | x_n, \Theta)$$
$$= -\sum_{n=1}^{N} \log p(y_n | x_n, \Theta)$$

since we have $p(y_n | x_n, \theta) = \mathcal{N}(y_n \; x^T\theta, \sigma^2)$:

$$NLL(\theta) = -\sum_{n=1}^{N} \log \mathcal{N}(y_n | x_n^T \theta, \sigma^2)$$

$$= -\sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \left(\frac{(y_n - \underbrace{x_n^T\theta}_{\mu})^2}{2\sigma^2} \right)$$

$$= -\sum_{n=1}^{N} \log \exp \left(\frac{-(y_n - x_n^T\theta)^2}{2\sigma^2} \right) + \text{const.}$$

↓ no influence on minimum location!

$$= \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - x_n^T\theta)^2 + \text{cons}'$$

so, we need to find the minimum of

$$NLL^*(\theta) = \sum_{n=1}^{N} (y_n - x_n^T \theta)^2$$

This is simply an L_2-error / quadratic form minimization as we have seen before!

To make the connection clearer, rewrite NLL^* in vector form:

$$NLL^*(\theta) = \underbrace{(\bar{Y} - \bar{X}\theta)^T (\bar{Y} - \bar{X}\theta)}_{r^T r}$$

We have seen before that this can be seen as a quadratic form, for which the minimum is found by solving the associated linear system, which leads again to the

normal equation of LLS:

$$\theta_{MLE} = \frac{\bar{X}^T \bar{y}}{\bar{X}^T \bar{X}}$$

→ see talk on normal form.

→ More on MLE and MAP estimator + regularized least squares: Bishop Sec 3.1.

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.

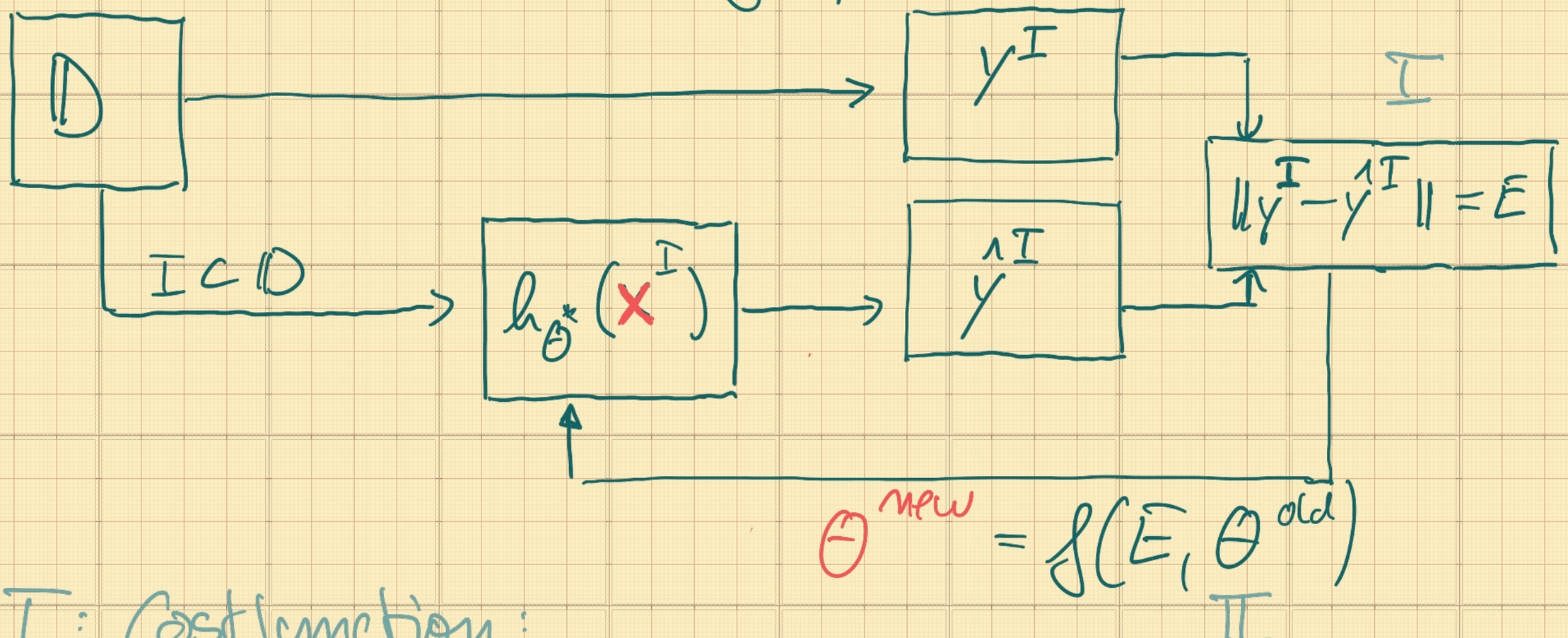
For an $(x_j, y_j) \in \mathbb{P} \cap \mathbb{D} = \emptyset$, $h_{\hat{\theta}}(x_j) = \hat{y}_j$

"inference"

"prediction stage"

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

The Supervised Learning cycle: Fair better!

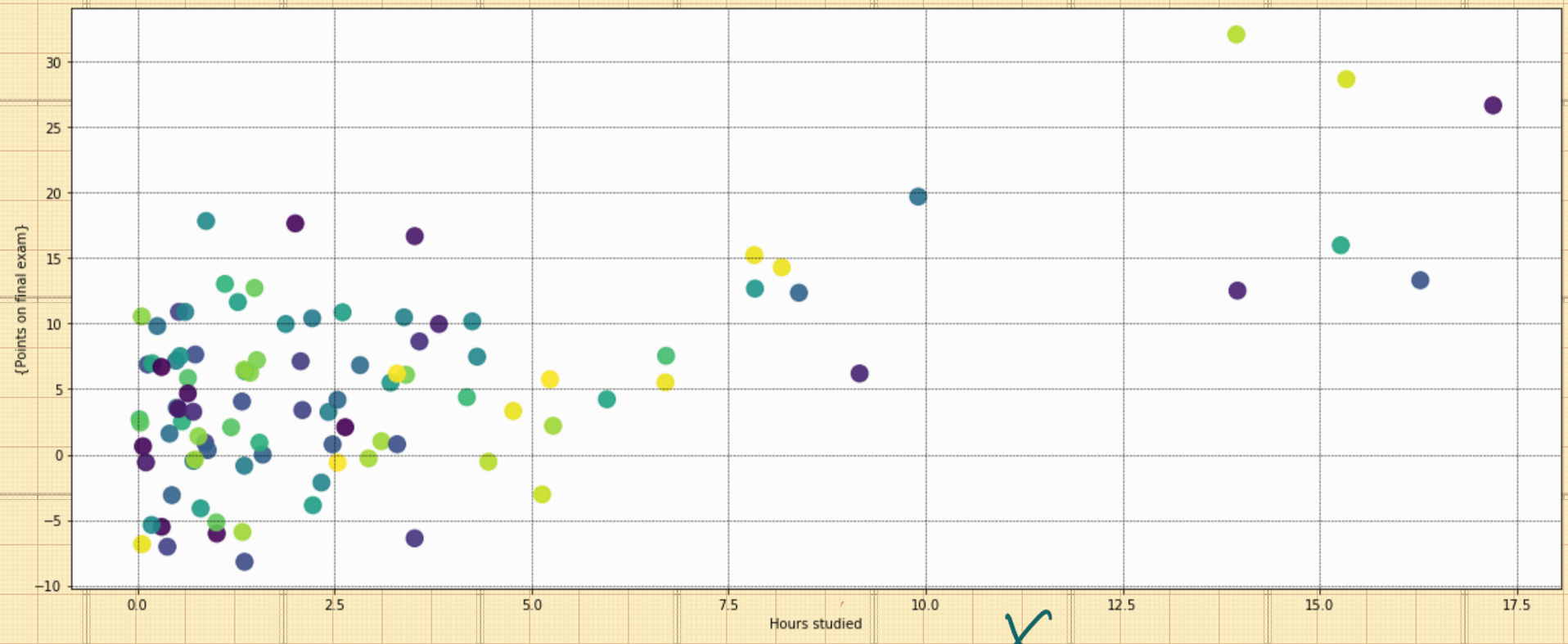


I: Cost function:
How to compute the error

II: How to use knowledge from I to improve θ

Univariate case:

y



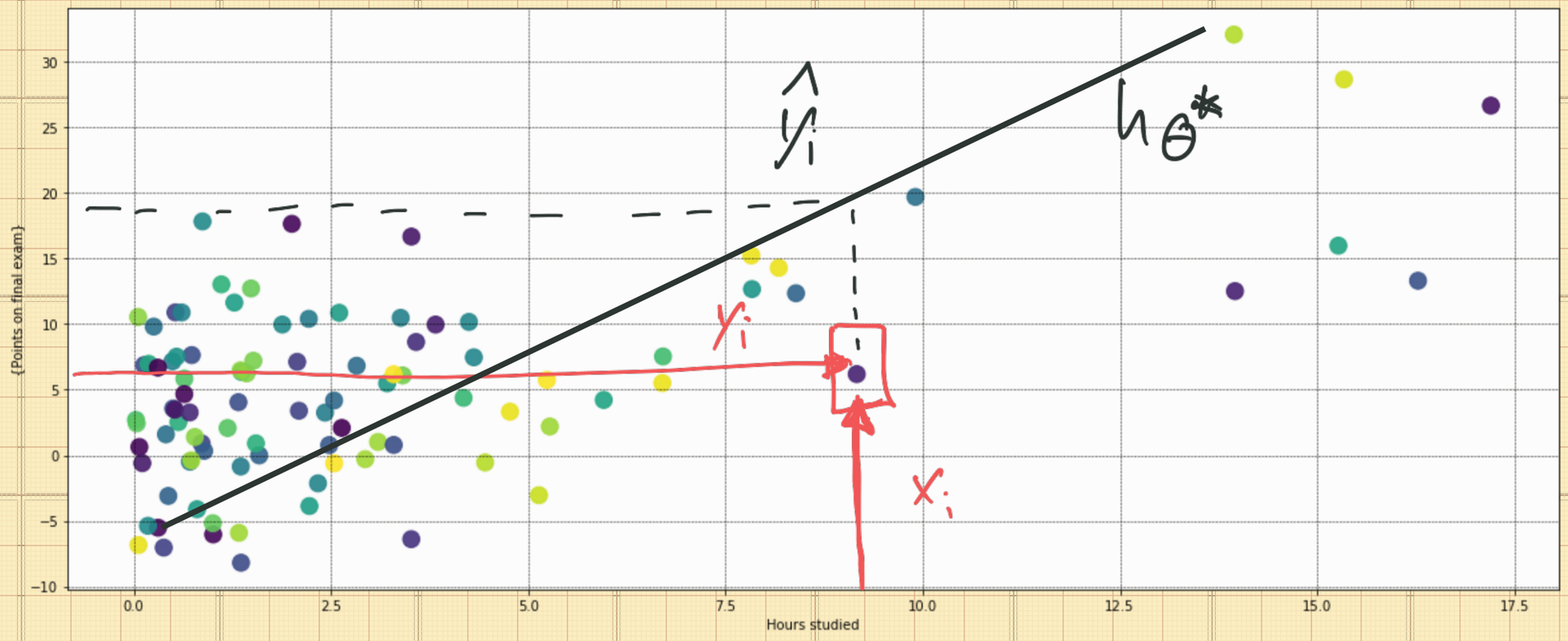
x_1

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

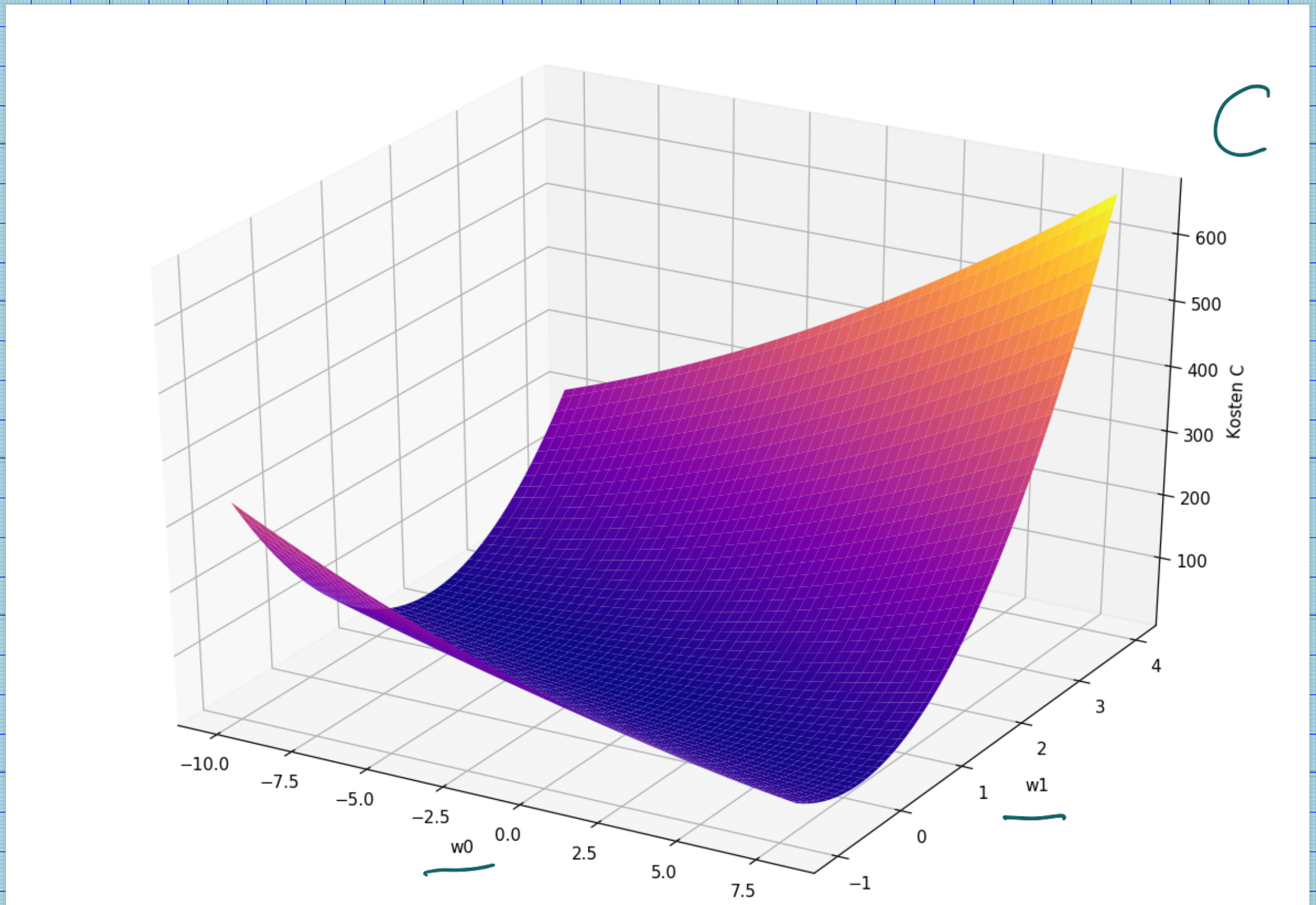
$$C = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

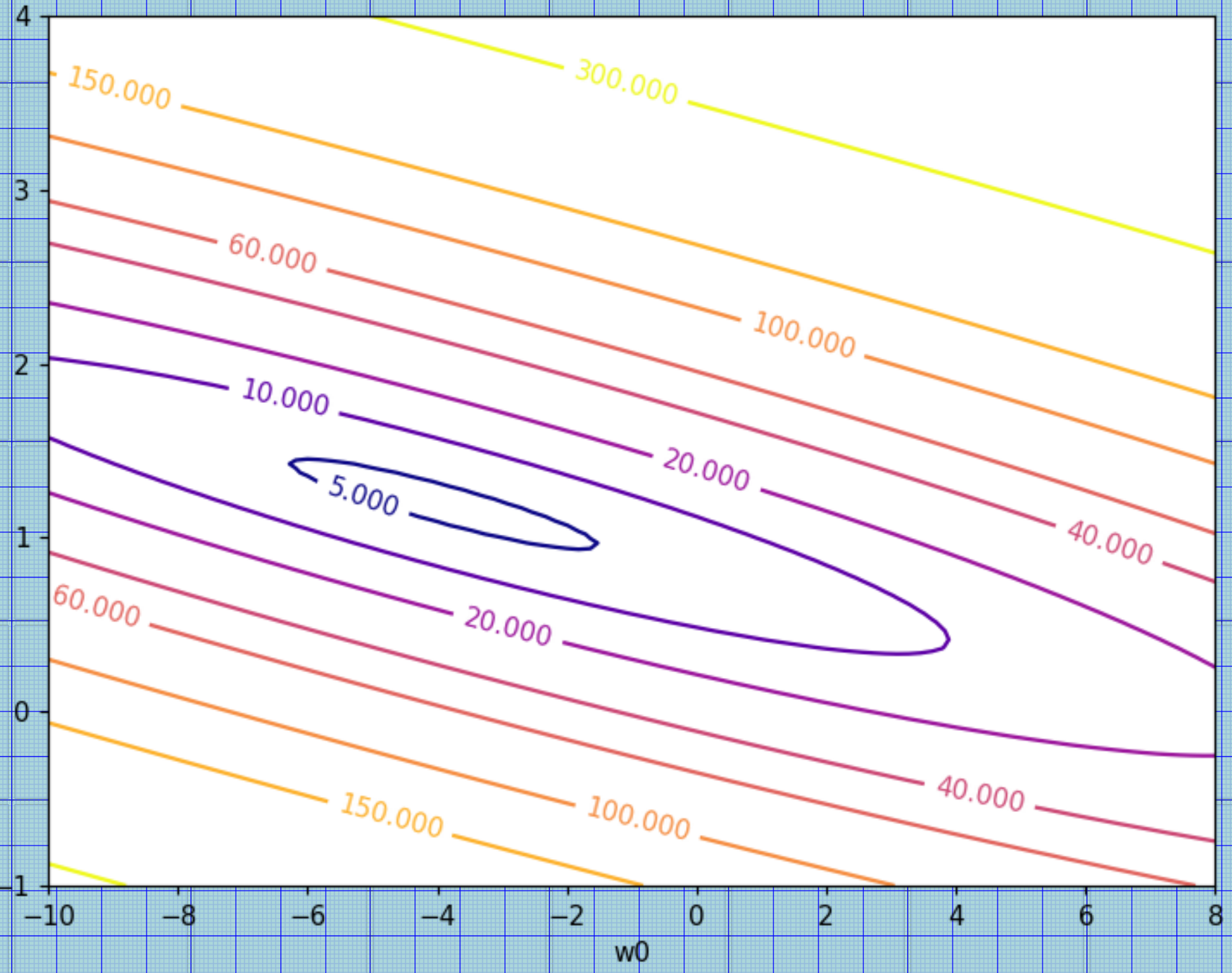
$\frac{dx^2}{dx} = 2x$

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:

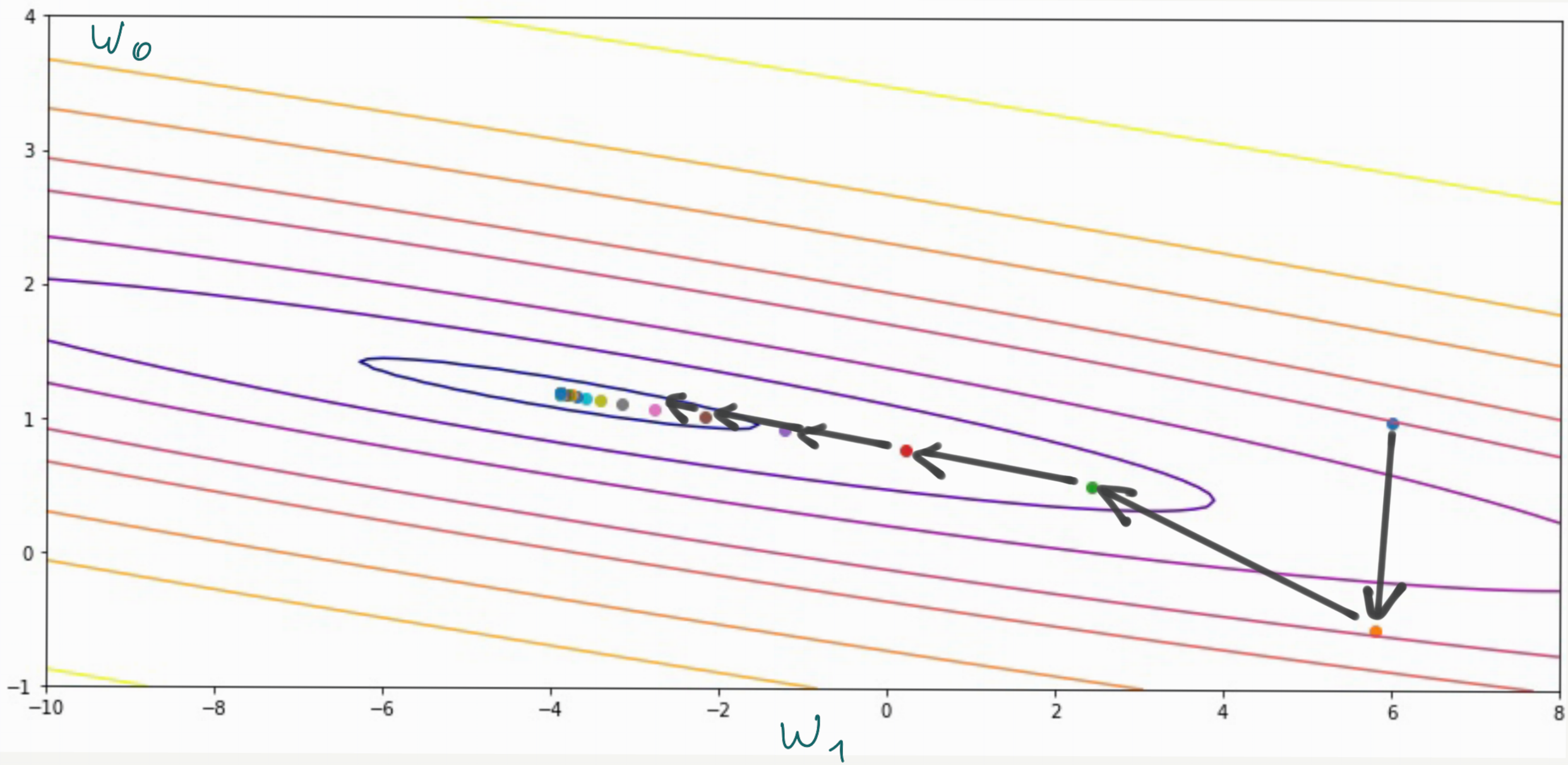
$$w_0^{next} = w_0^{old} - \alpha \frac{\partial C}{\partial w_0}$$

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

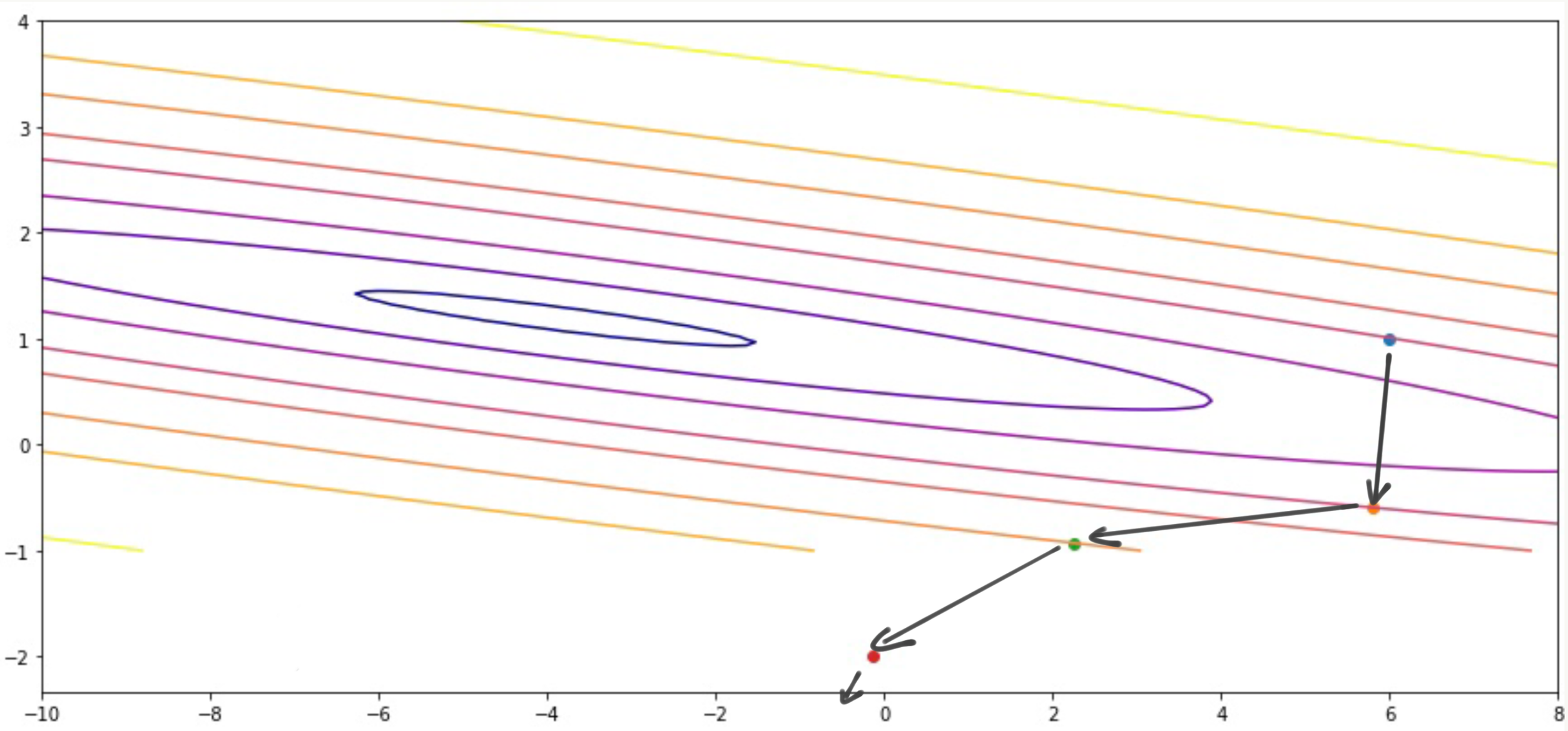
step size: "learning rate"
⇒ hyperparameter

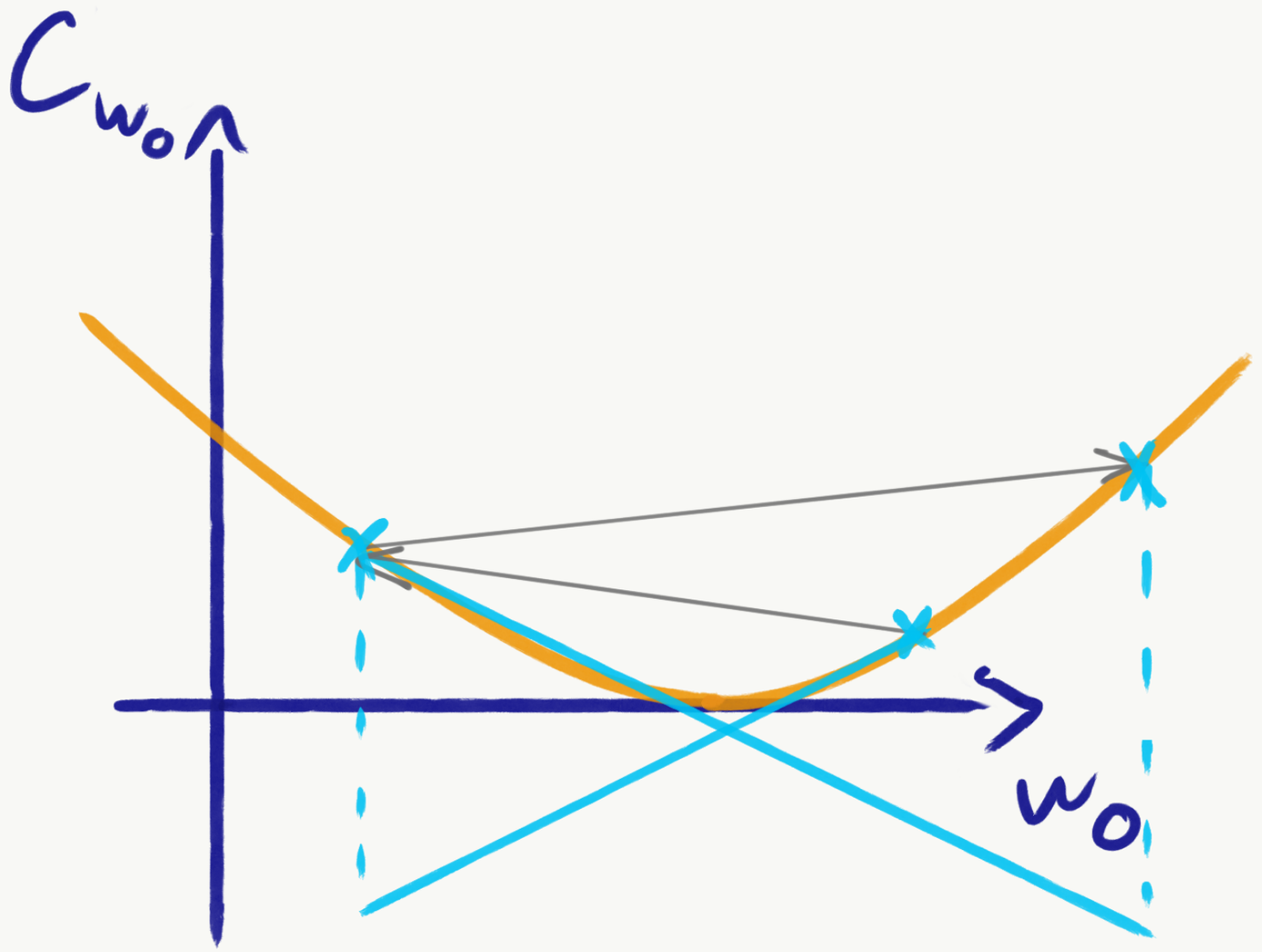
iterate and update!

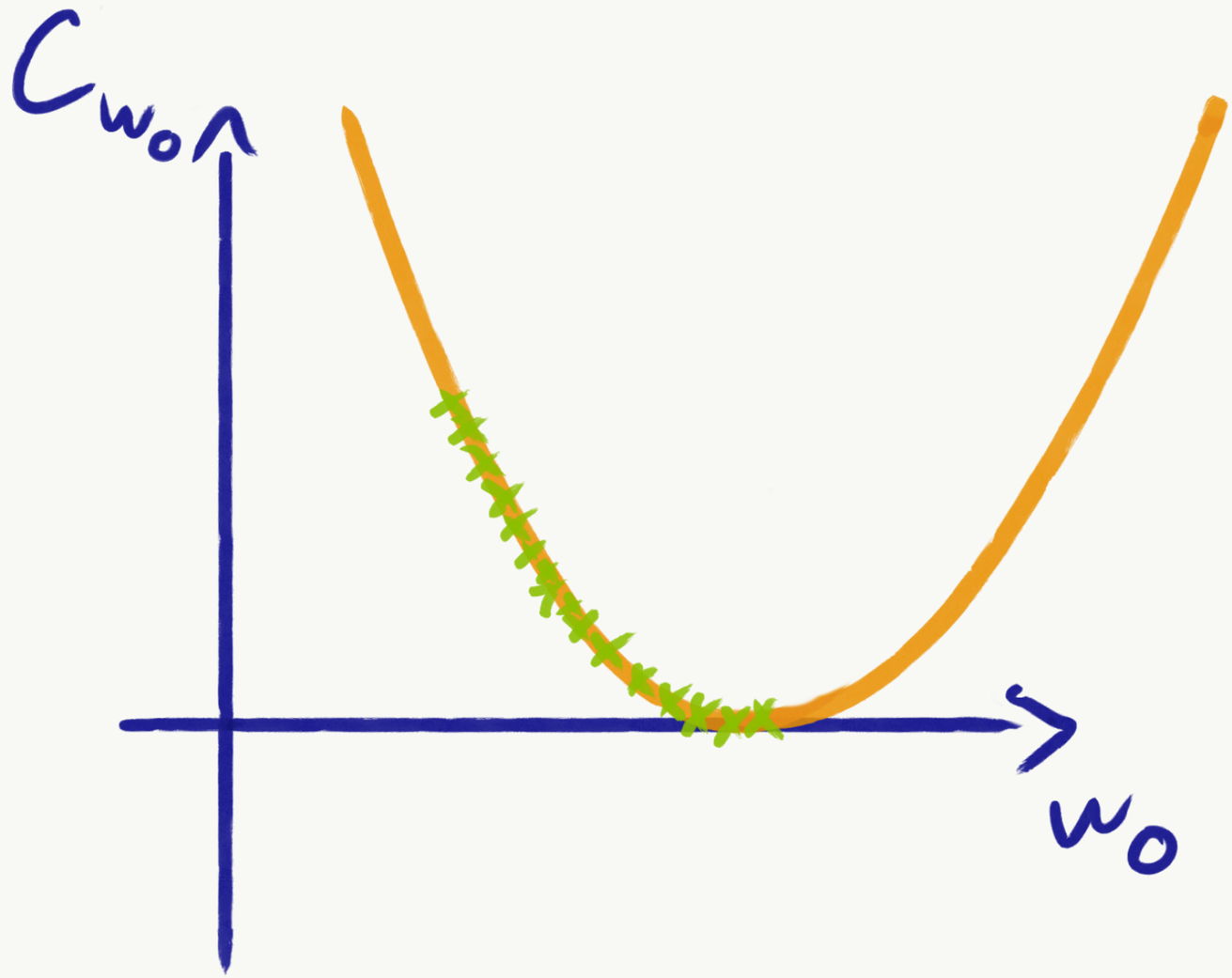
Succesful optimization

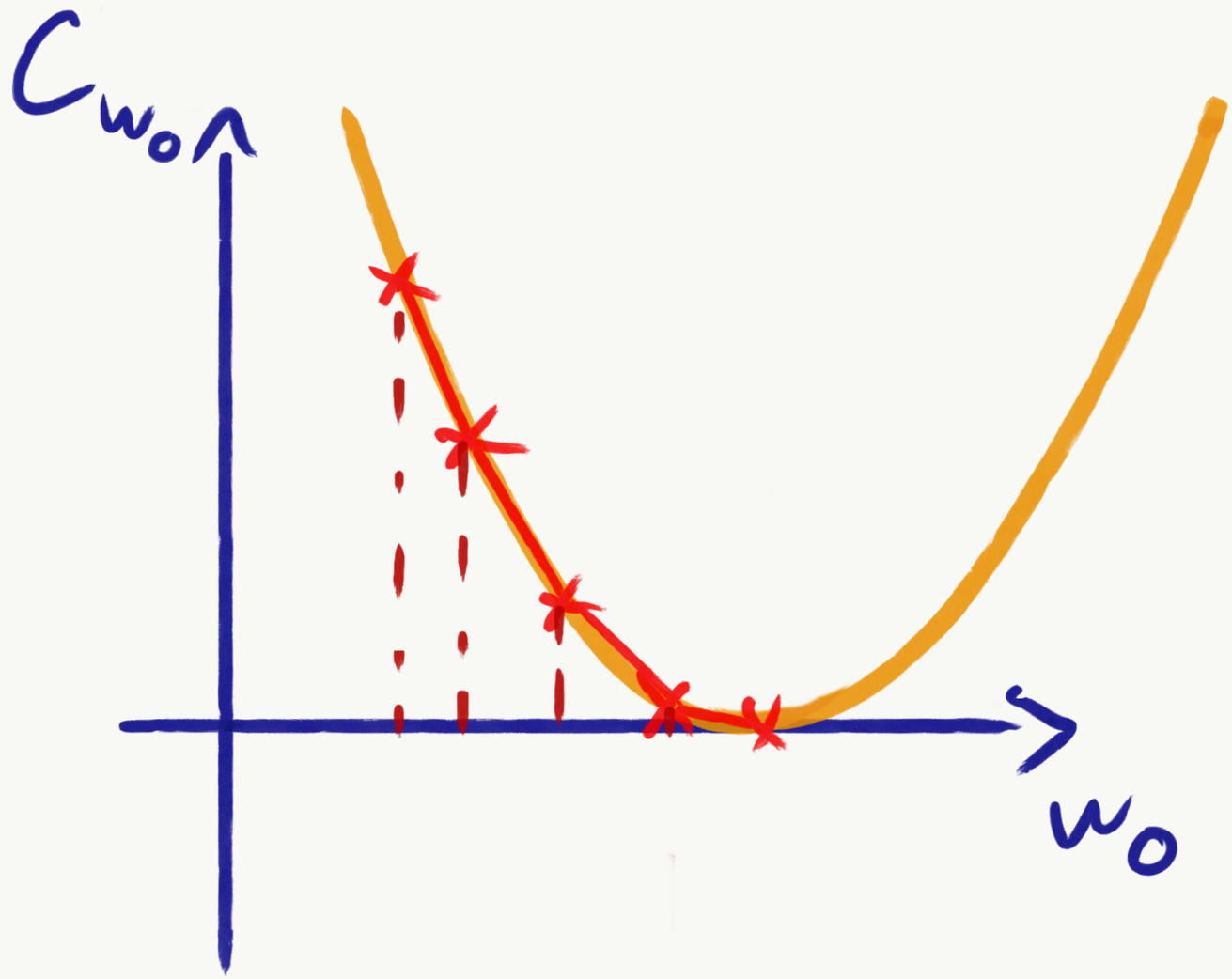


Learning Rate too large → diverging

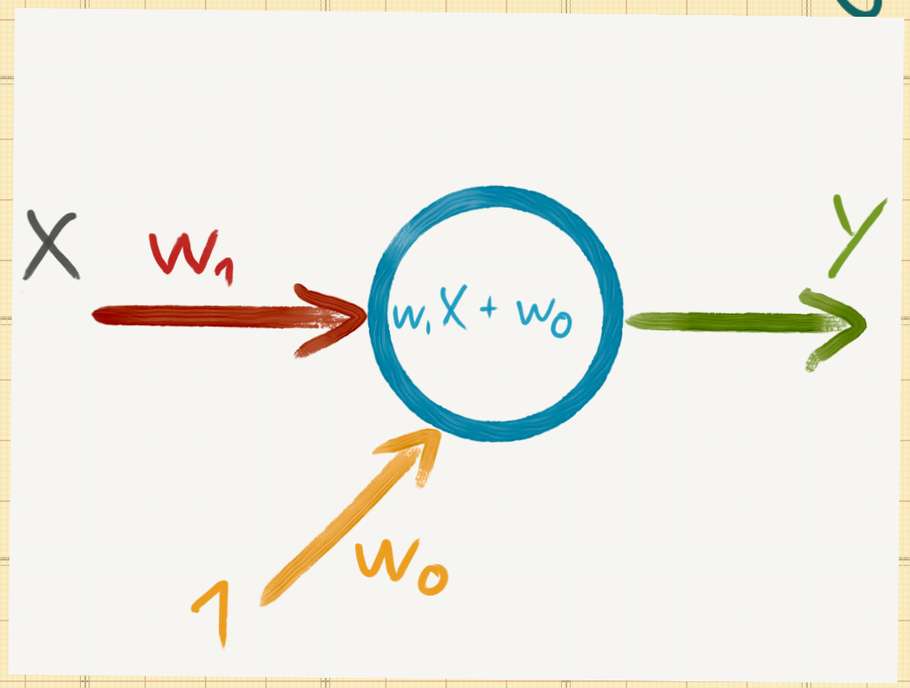


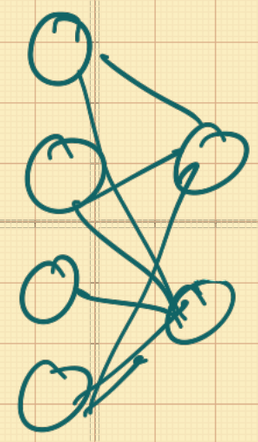

ω_{on}
ω_o


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:

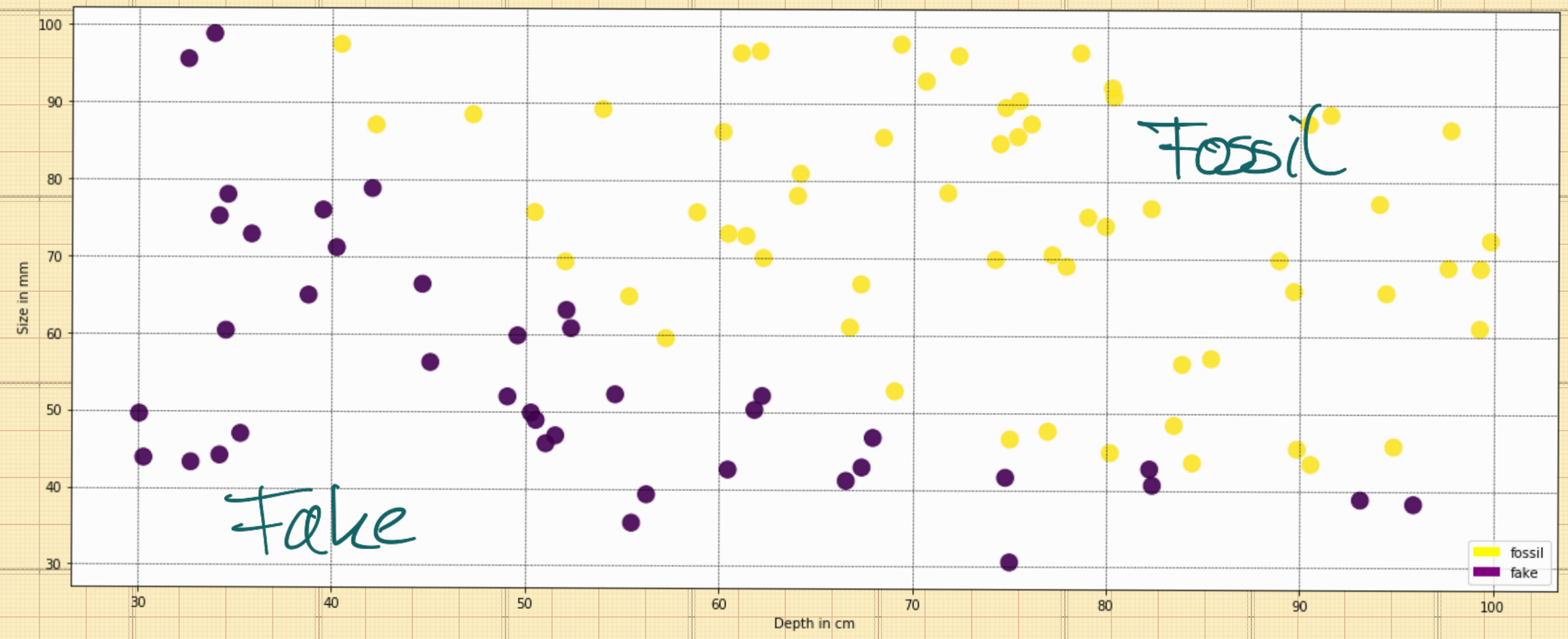




Logistic Regression (a Classifier!)

Lin. Reg: regression; Log. Reg: Classification

→ we consider binary classification: pass/fail; spam/no spam; $y = \{0, 1\}$



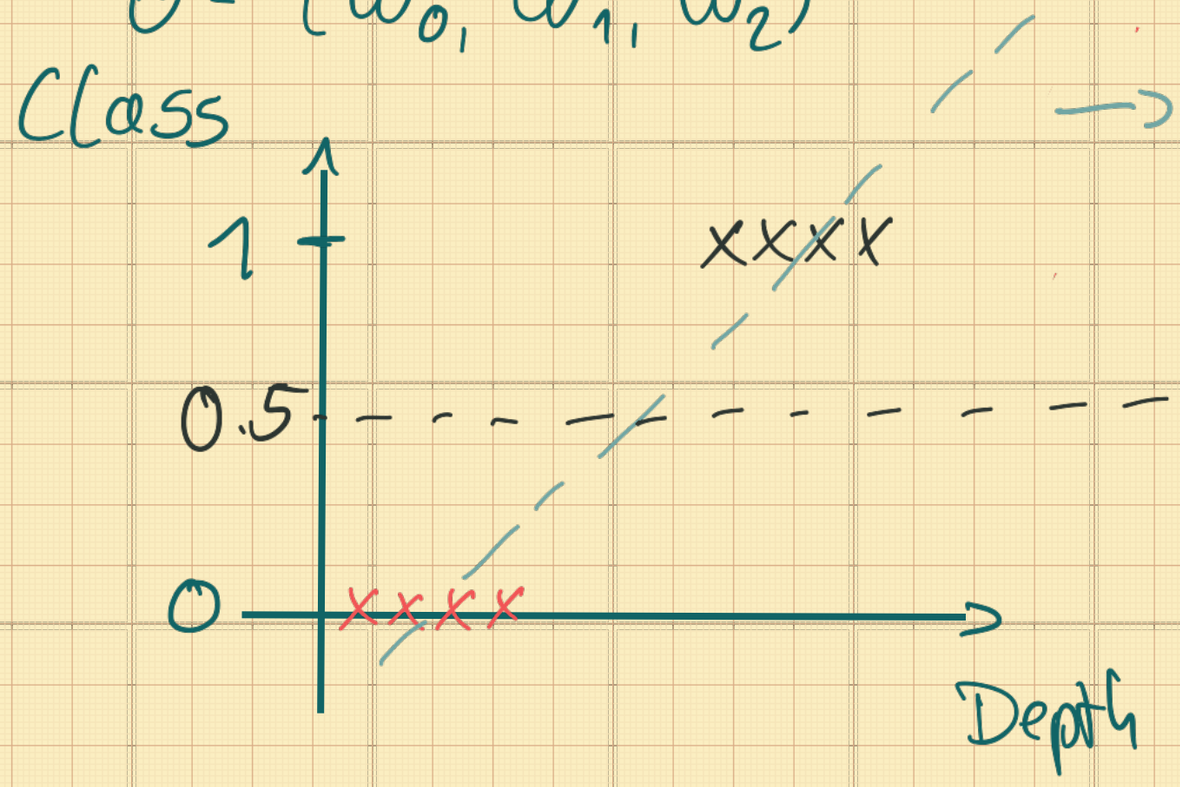
→ train an ML method with inputs x_1: depth, x_2: size to predict 0/1 : fake / fossil

→ let us try the linear regression:

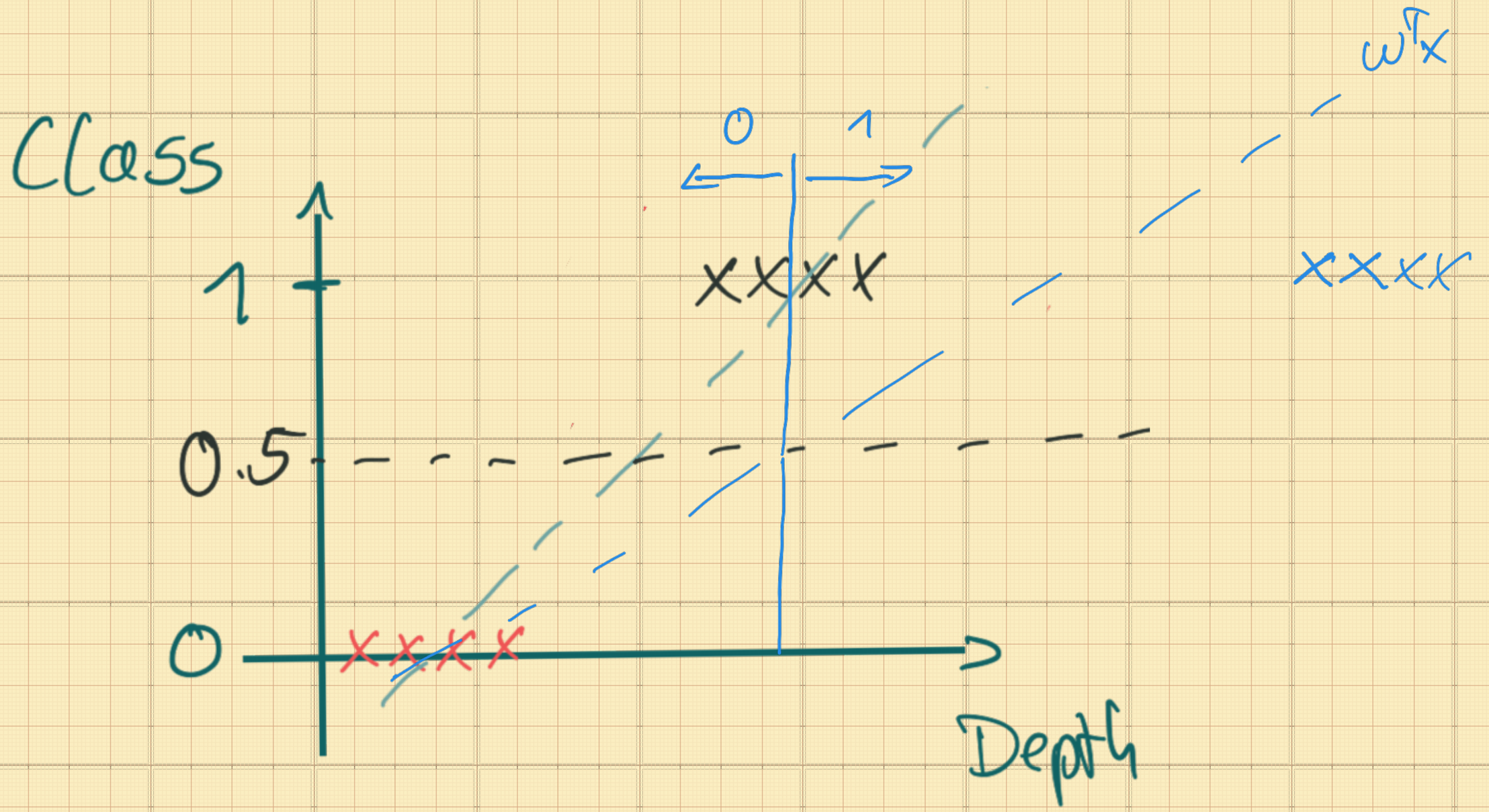
$$h_\theta = w_0 + w_1 x + w_2 x_2$$

$$\theta = (w_0, w_1, w_2)^T$$

if $\hat{y} > 0.5 \rightarrow 1$
if $\hat{y} \leq 0.5 \rightarrow 0$

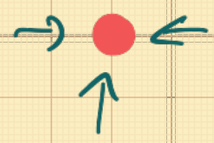


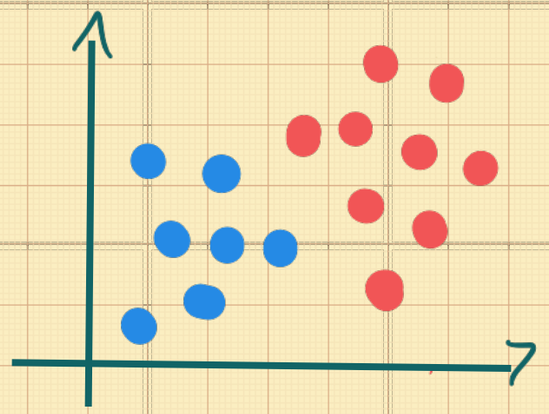
→ Linreg: $h = w^T x$, minimizes MSE


Class
1
0.5
0
Depth
$w^T x$
0
1

→ linear model not well suited for classification!

→ outlies move the optimum strongly





→ output of $x^T w$ is unbounded, not $0 \leq x^T w \leq 1$

→ Probabilistic p.o.v: $p(y \mid x, \theta) = \text{Ber}(y \mid \mu(x))$

→ MLE → cost function: "cross entropy"

→ non-linear w.r.t. θ

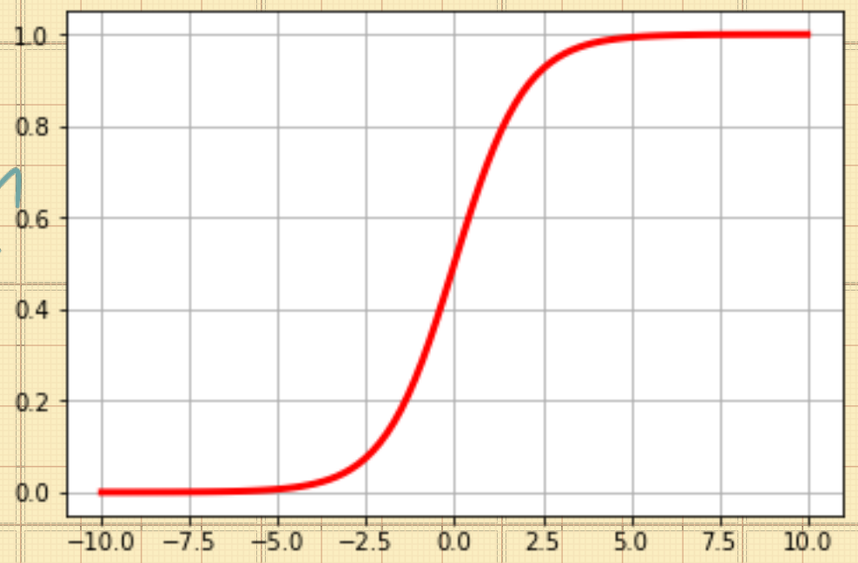
→ optimization more difficult

→ practical approach!

→ extend linear hypothesis by

non-linear activation function

→ "sigmoid" function:



↓ induced by

Bernoulli distr.

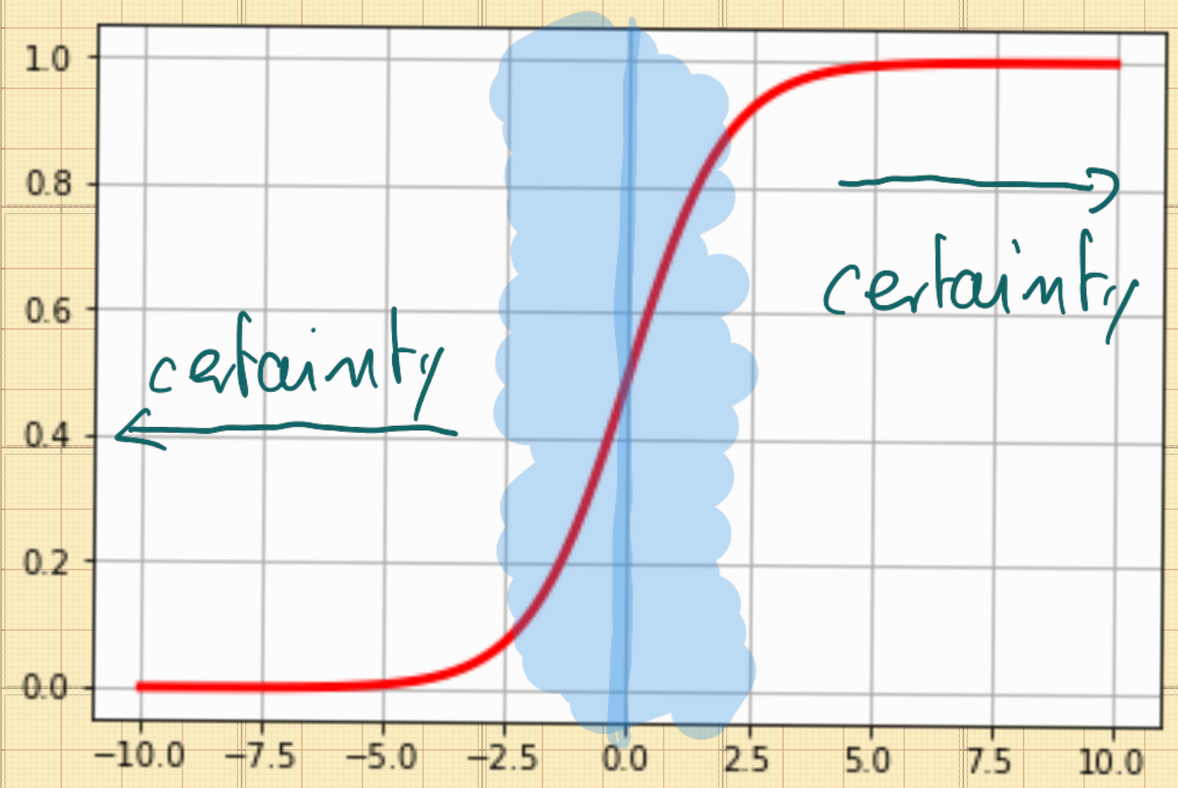
$$G(z) = \frac{1}{1+\exp(-z)}$$

$z \to \infty : G(z) \to 1$

$z \to -\infty : G(z) \to 0$

$$\to \overset{\log}{h_\theta} = G(\overset{\text{lin}}{h_\theta}(x)) = G(w_0 + w_1 x_1 + w_2 x_2) \in (0,1)$$

→ We can now interpret $h_\theta^{\log}(x)$ as the probabilities $p(y=1|x)$: "How likely is class 1?"

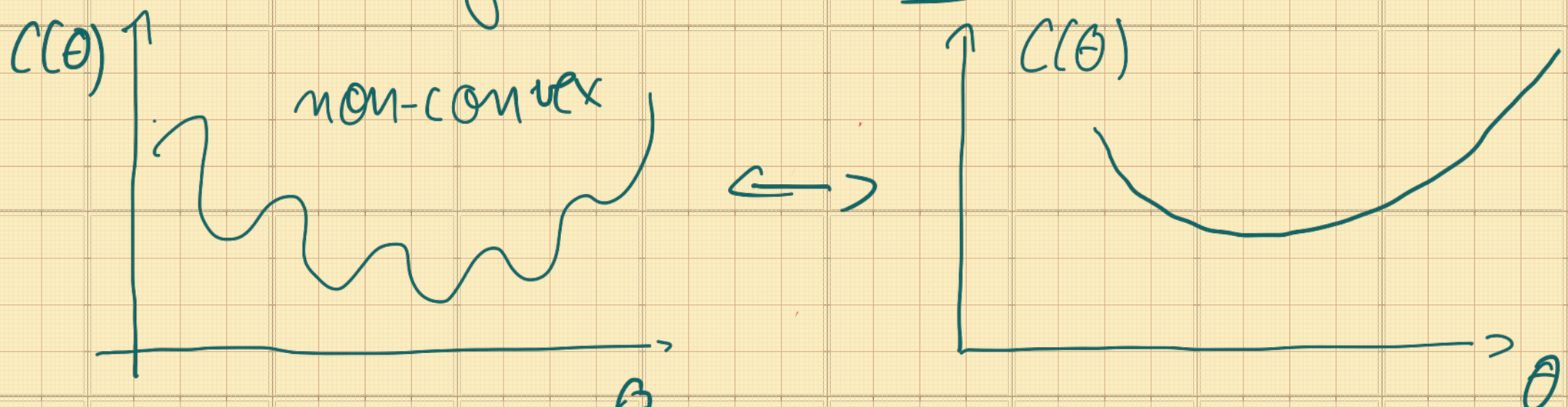


→ note that now all points are equally important in the cost function (outliers do not skew the cost)

$h_\theta^{log}(x) = 0.5$: decision boundary

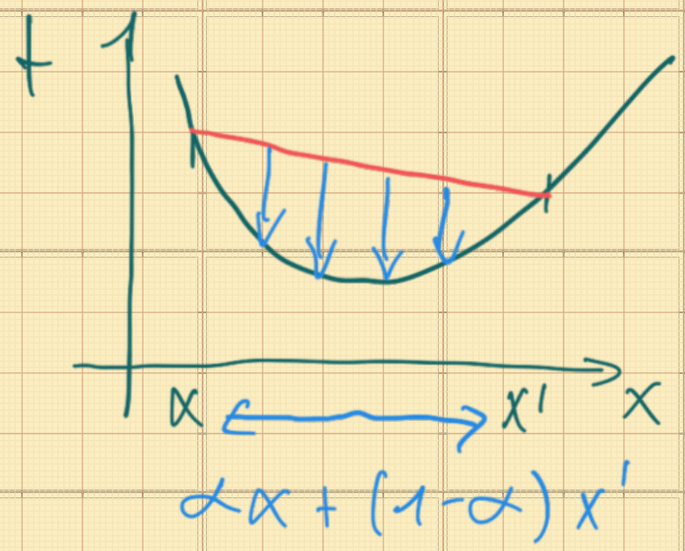
$$C = \frac{-1}{N} \sum_{n=1}^{N} \left[y_i \log(h(x_i)) + (1-y_i) \log(1-h(x_i)) \right]$$

(a MSE cost function is not convex w.r.t. θ)



Def.: $0 \leq \alpha \leq 1$: if $f(\alpha x + (1-\alpha)x') \leq \alpha f(x) + (1-\alpha) f(x')$

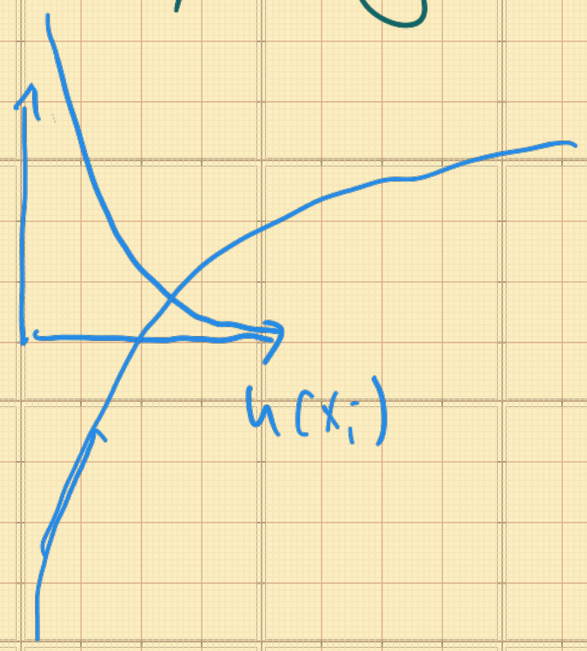
linear blending

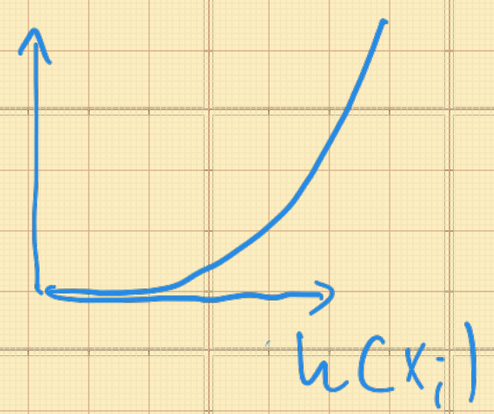


convex

non-convex

$$C' = -y_i \log(h_i) - (1-y_i)\log(1-h_i)$$

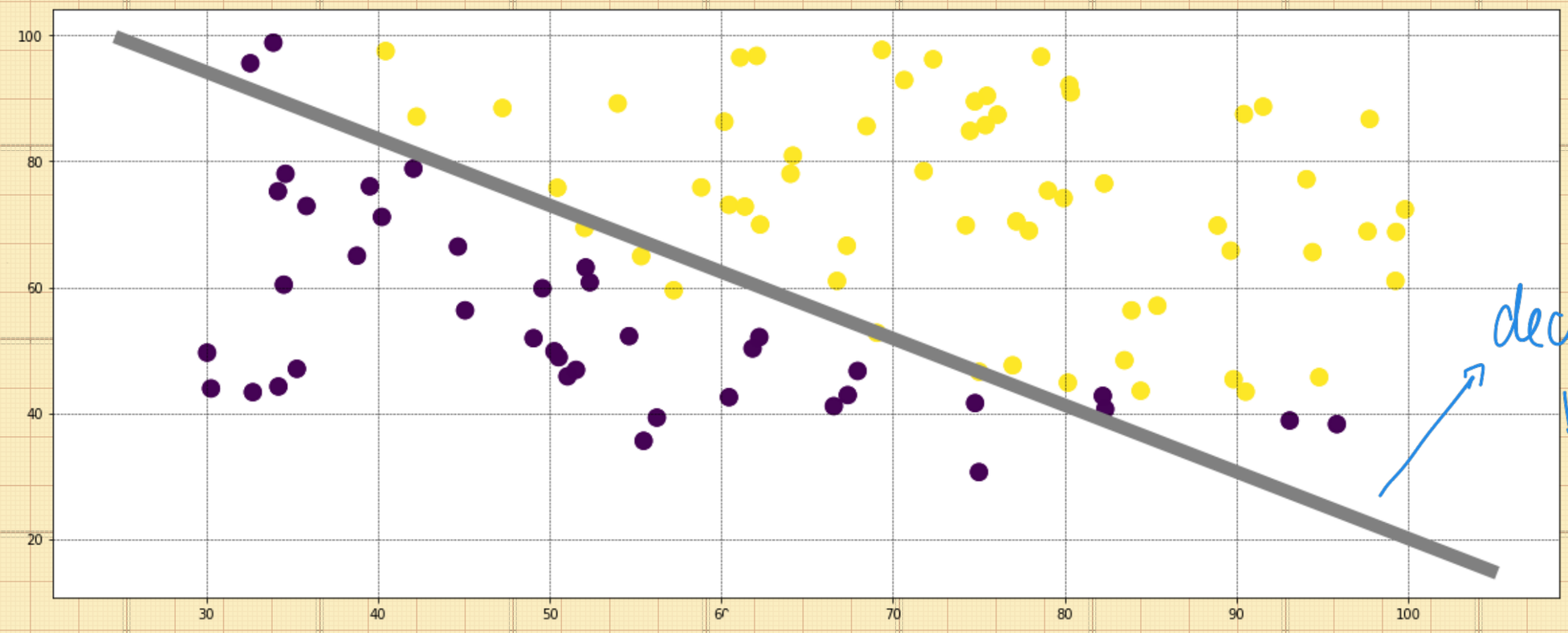




$$\nabla_\theta \mathcal{L}^i = \frac{\partial \mathcal{L}^i}{\partial w_j} = (h_i - y_i)\, x_{ij}$$

$$\text{e.g. } w_1^{new} = w_1^{old} - \frac{LR}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}^n}{\partial w_1}$$

j: feature index
$(w_0 x_0 + w_1 x_1 + w_2 x_2)$
i: which sample

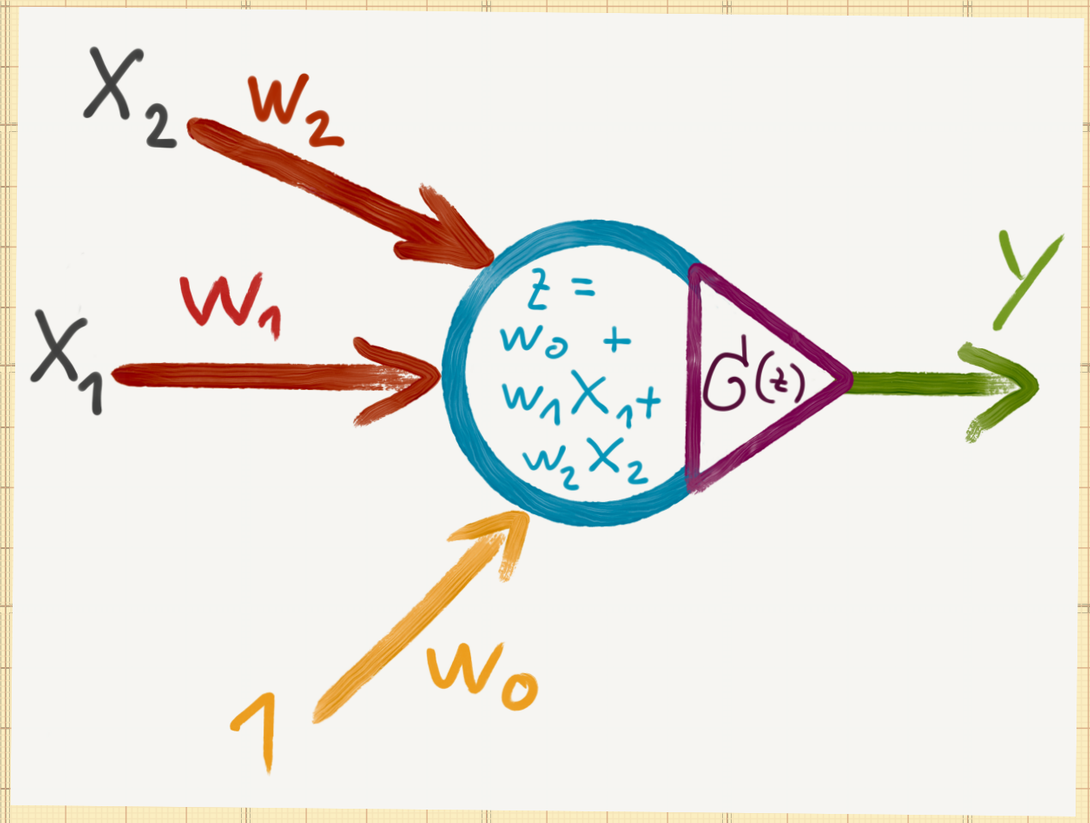


Problem: linearly seperable data

Logistic neuron:

linear neuron

→ activation function



Sigmoid, tanh, ReLU, ...

"Perceptron": Rosenblatt ~~1926~~ 1962 : $\sigma(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$

→ Book recommendation:

Pattern recognition & machine learning, Ch. Bishop

Springer