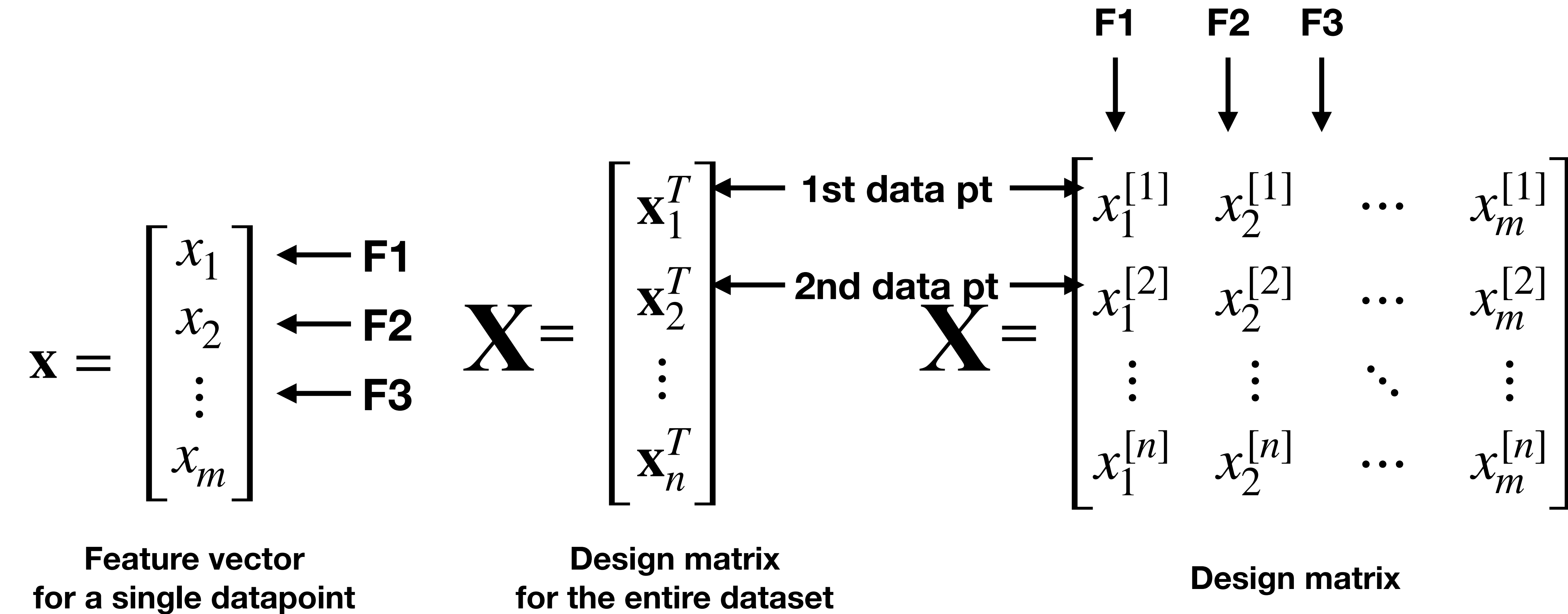


Lecture 3 pre-video

The learning part of ML requires
vector calculus*

* because learning is a lot like rolling down a grassy hill
when you were 7 🤪

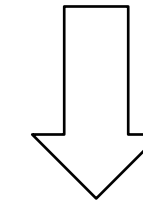
Data Representation



Representing data in a matrix

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad y_i \in \{-1, +1\}$$

	age	male or female	weight (lb)	height (cm)
$y_1 = -1$ (negative)	$x_{11} = 22$	$x_{12} = M$	$x_{13} = 160$	$x_{14} = 180$
$y_2 = +1$ (positive)	$x_{21} = 51$	$x_{22} = M$	$x_{23} = 190$	$x_{24} = 175$
$y_3 = +1$ (positive)	$x_{31} = 43$	$x_{32} = F$	$x_{33} = 120$	$x_{34} = 165$

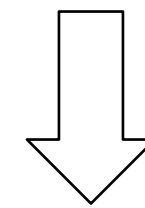


$$\mathbf{X} = \begin{pmatrix} 22 & 1 & 0 & 160 & 180 \\ 51 & 1 & 0 & 190 & 175 \\ 43 & 0 & 1 & 120 & 165 \end{pmatrix}$$

Predicting outputs

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad y_i \in \{-1, +1\}$$

	age	male or female	weight (lb)	height (cm)
$y_1 = -1$ (negative)	$x_{11} = 22$	$x_{12} = M$	$x_{13} = 160$	$x_{14} = 180$
$y_2 = +1$ (positive)	$x_{21} = 51$	$x_{22} = M$	$x_{23} = 190$	$x_{24} = 175$
$y_3 = +1$ (positive)	$x_{31} = 43$	$x_{32} = F$	$x_{33} = 120$	$x_{34} = 165$



$$\mathbf{y} = \begin{pmatrix} -1 \\ +1 \\ +1 \end{pmatrix}$$

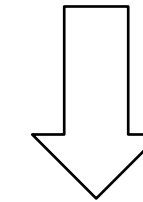
$$\mathbf{X} = \begin{pmatrix} 22 & 1 & 0 & 160 & 180 \\ 51 & 1 & 0 & 190 & 175 \\ 43 & 0 & 1 & 120 & 165 \end{pmatrix}$$

Lets assume $\mathbf{w} = \begin{pmatrix} 0.075 \\ 0 \\ 0 \\ -0.007 \\ -0.008 \end{pmatrix}$

Predicting outputs

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad y_i \in \{-1, +1\}$$

	age	male or female	weight (lb)	height (cm)
$y_1 = -1$ (negative)	$x_{11} = 22$	$x_{12} = M$	$x_{13} = 160$	$x_{14} = 180$
$y_2 = +1$ (positive)	$x_{21} = 51$	$x_{22} = M$	$x_{23} = 190$	$x_{24} = 175$
$y_3 = +1$ (positive)	$x_{31} = 43$	$x_{32} = F$	$x_{33} = 120$	$x_{34} = 165$

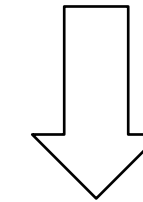


$$\mathbf{X} = \begin{pmatrix} 22 & 1 & 0 & 160 & 180 \\ 51 & 1 & 0 & 190 & 175 \\ 43 & 0 & 1 & 120 & 165 \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} 0.075 \\ 0 \\ 0 \\ -0.007 \\ -0.008 \end{pmatrix} \quad \hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \begin{pmatrix} -0.91 \\ +1.095 \\ +0.65 \end{pmatrix}$$

How wrong were the predictions?

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad y_i \in \{-1, +1\}$$

	age	male or female	weight (lb)	height (cm)
$y_1 = -1$ (negative)	$x_{11} = 22$	$x_{12} = M$	$x_{13} = 160$	$x_{14} = 180$
$y_2 = +1$ (positive)	$x_{21} = 51$	$x_{22} = M$	$x_{23} = 190$	$x_{24} = 175$
$y_3 = +1$ (positive)	$x_{31} = 43$	$x_{32} = F$	$x_{33} = 120$	$x_{34} = 165$



$$\mathbf{y} = \begin{pmatrix} -1 \\ +1 \\ +1 \end{pmatrix} \quad \hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \begin{pmatrix} -0.91 \\ +1.095 \\ +0.65 \end{pmatrix}$$

$$\mathcal{L} = \mathbf{y} - f(\hat{\mathbf{y}})$$

$$\mathcal{L}(\mathbf{w}) = \mathbf{y} - f(\mathbf{X}\mathbf{w})$$

🔍 machine learning is just [Search Google](#)

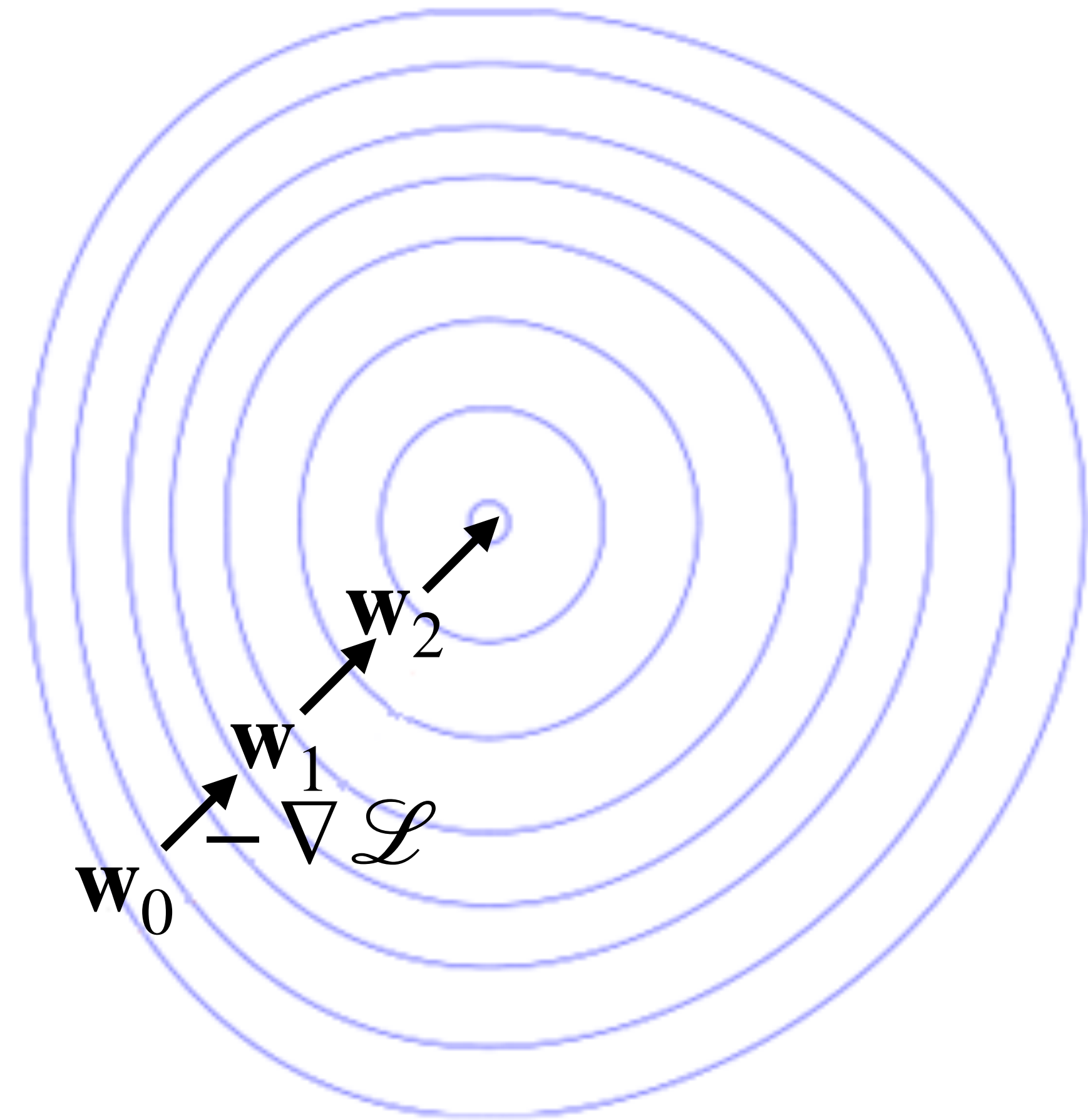
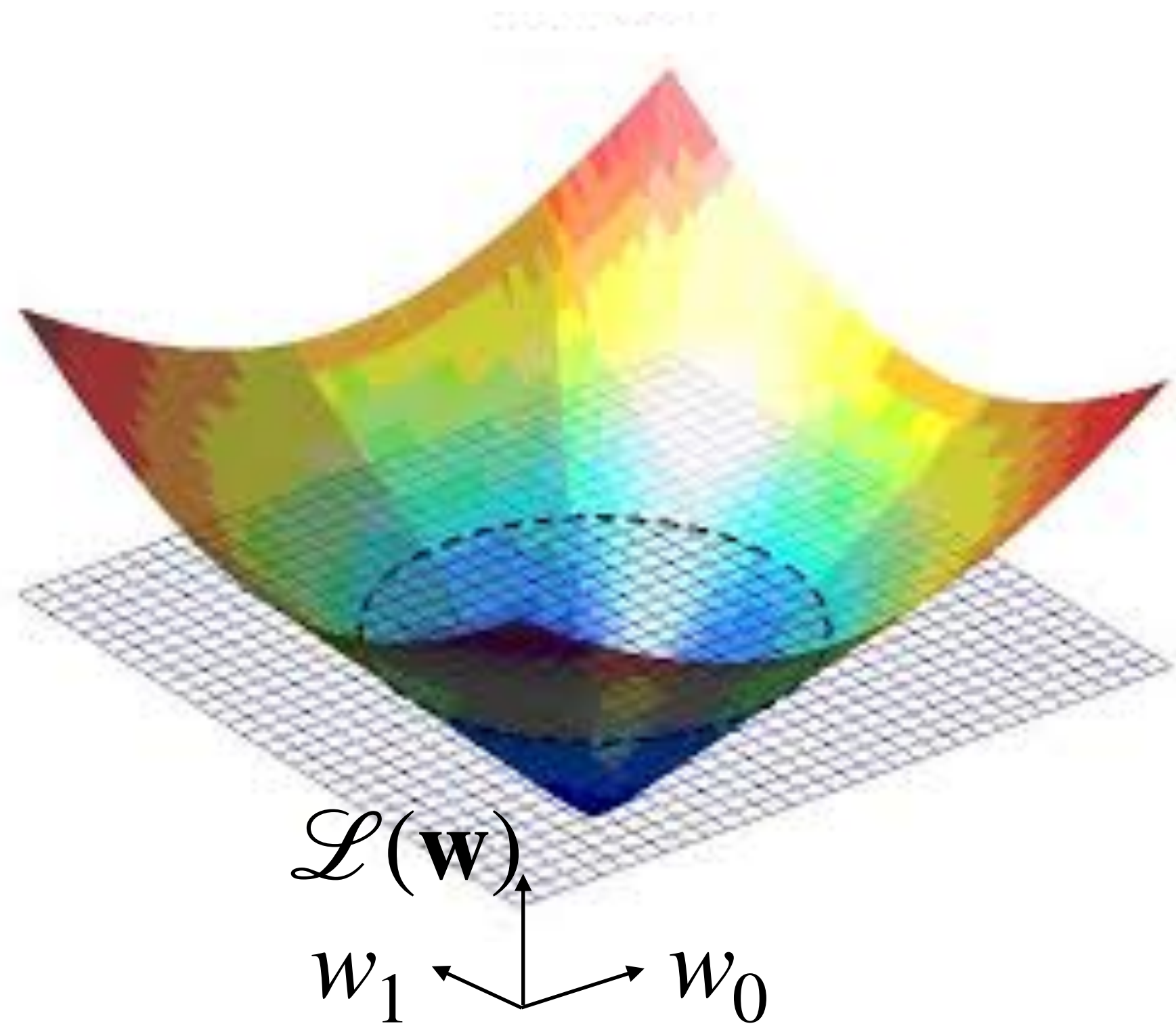
Google Suggestions

🔍 **Machine learning is just picking an appropriate model and then minimizing a loss function**

- 🔍 machine learning is just statistics
- 🔍 machine learning is just if statements
- 🔍 machine learning is just a bunch of if statements
- 🔍 machine learning is just curve fitting

$$\mathcal{L}(\mathbf{w}) = g(\mathbf{y} - f(\mathbf{X}\mathbf{w}))$$

$$\nabla \mathcal{L} \equiv \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$$

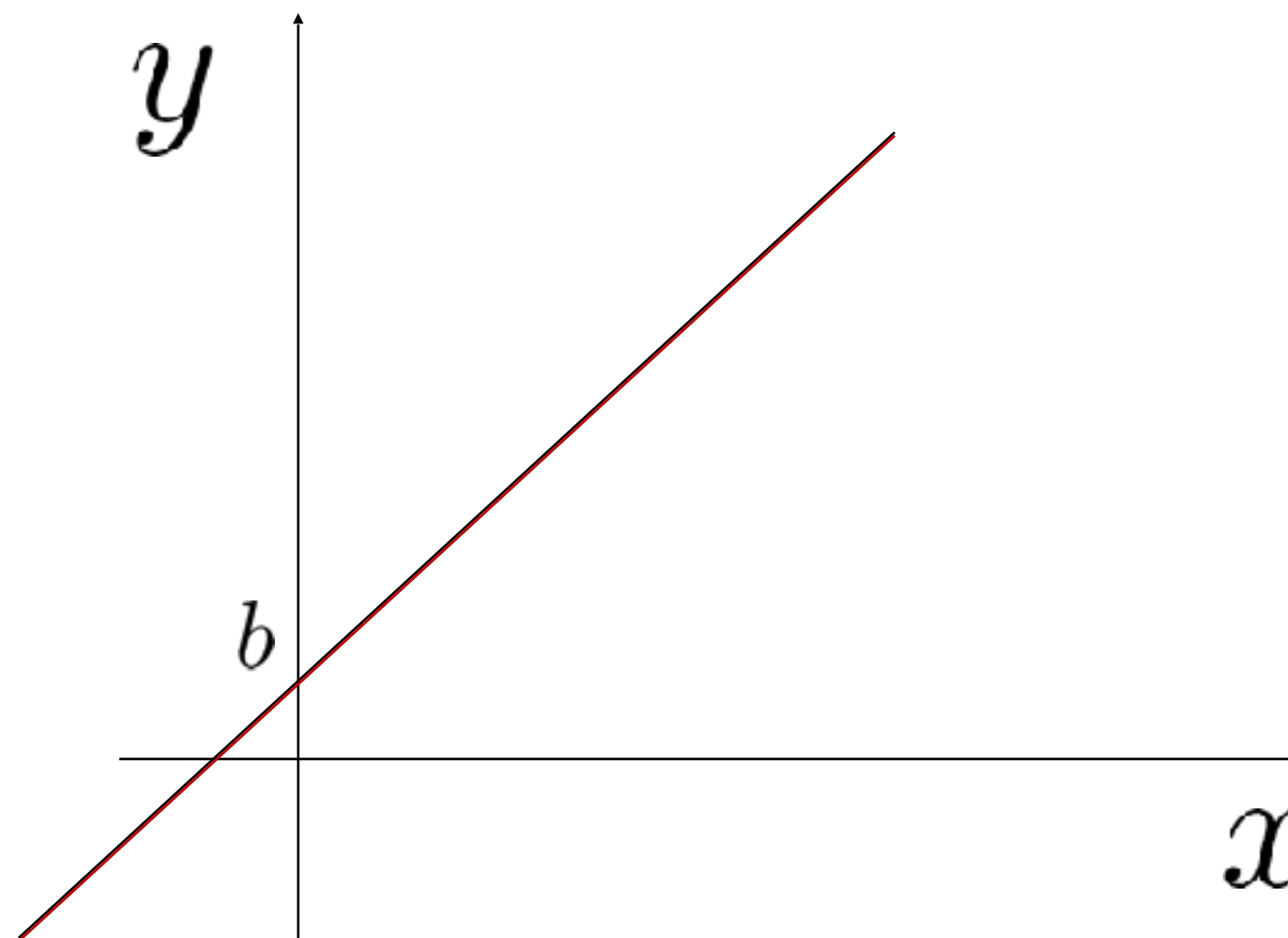


Calculus

Scalar:

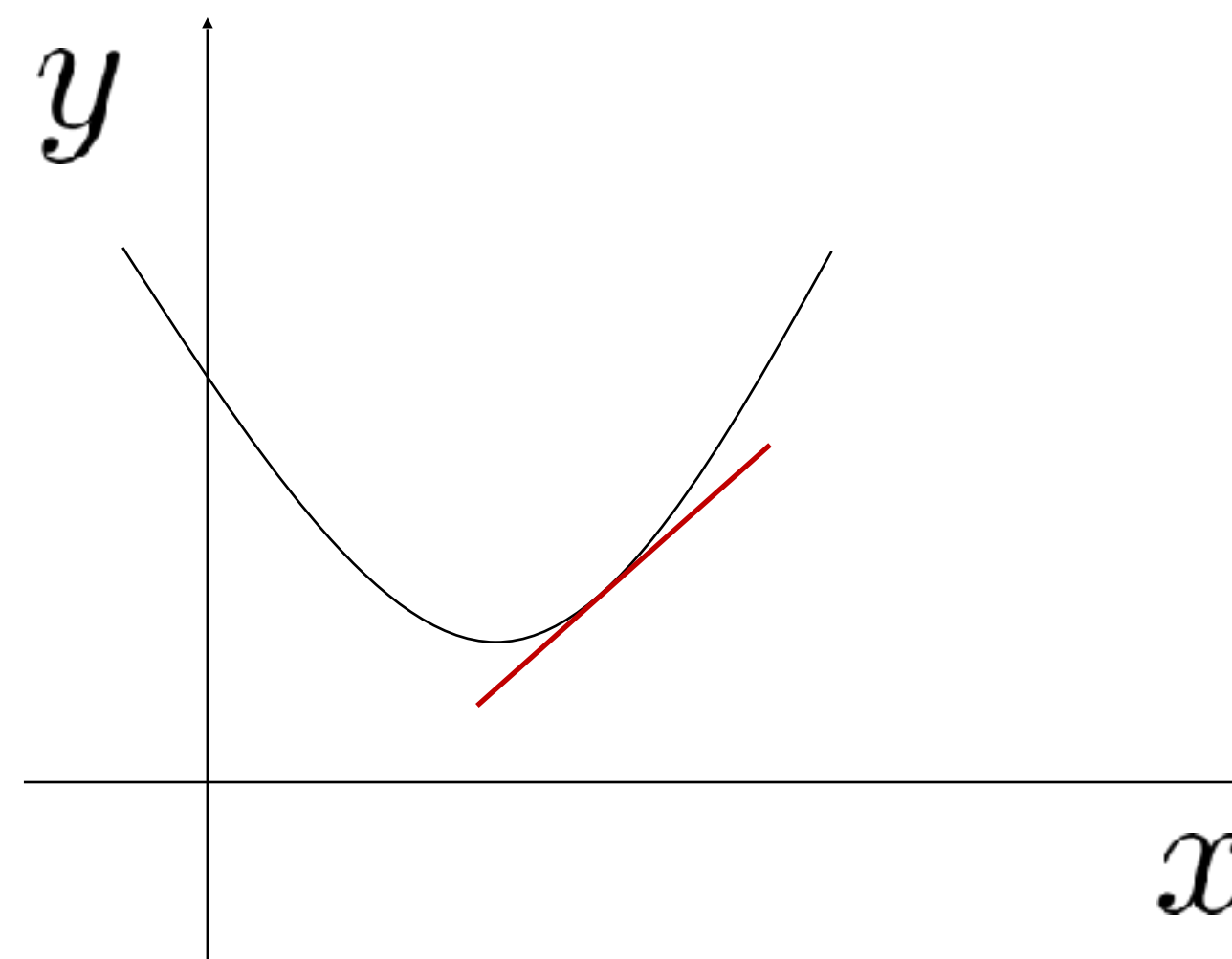
$$y = ax + b$$

$$\frac{dy}{dx} = a$$



$$y = ax^2 + bx + c$$

$$\frac{dy}{dx} = 2ax + b$$



Vector calculus

Vector derivatives

Vector-by-scalar

$$\mathbf{y}(x) = \begin{pmatrix} y_1(x) & y_2(x) & y_3(x) \end{pmatrix}$$

$$\frac{d\mathbf{y}(x)}{dx} = \begin{pmatrix} \frac{dy_1(x)}{dx} & \frac{dy_2(x)}{dx} & \frac{dy_3(x)}{dx} \end{pmatrix}$$

Vector-by-vector

$$\mathbf{y}(\mathbf{x}) = \begin{pmatrix} y_1(\mathbf{x}) & , \dots , & y_m(\mathbf{x}) \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x_1 & , \dots , & x_n \end{pmatrix}$$

$$\frac{d\mathbf{y}(\mathbf{x})}{d\mathbf{x}} = \begin{pmatrix} \frac{dy_1(\mathbf{x})}{dx_1} & , \dots , & \frac{dy_m(\mathbf{x})}{dx_1} \\ \cdot & \cdot & \cdot \\ \frac{dy_1(\mathbf{x})}{dx_n} & , \dots , & \frac{dy_m(\mathbf{x})}{dx_n} \end{pmatrix}$$

Vector-by-vector

$$A = \begin{pmatrix} a_{11} & , \dots , & a_{1m} \\ . & . & . \\ a_{n1} & , \dots , & a_{nm} \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ . \\ x_m \end{pmatrix}$$

Vector calculus

$$\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A$$

numerator form

$$\frac{\partial \mathbf{x}^T A^T}{\partial \mathbf{x}} = A$$

denominator form

Vector calculus

Identities: vector-by-vector $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$			
Condition	Expression	Numerator layout, i.e. by \mathbf{y} and \mathbf{x}^\top	Denominator layout, i.e. by \mathbf{y}^\top and \mathbf{x}
\mathbf{a} is not a function of \mathbf{x}	$\frac{\partial \mathbf{a}}{\partial \mathbf{x}} =$	$\mathbf{0}$	
	$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} =$	\mathbf{I}	
\mathbf{A} is not a function of \mathbf{x}	$\frac{\partial \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} =$	\mathbf{A}	\mathbf{A}^\top
\mathbf{A} is not a function of \mathbf{x}	$\frac{\partial \mathbf{x}^\top \mathbf{A}}{\partial \mathbf{x}} =$	\mathbf{A}^\top	\mathbf{A}
a is not a function of \mathbf{x} , $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$\frac{\partial a \mathbf{u}}{\partial \mathbf{x}} =$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	
$a = a(\mathbf{x})$, $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$\frac{\partial a \mathbf{u}}{\partial \mathbf{x}} =$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \mathbf{u} \frac{\partial a}{\partial \mathbf{x}}$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial a}{\partial \mathbf{x}} \mathbf{u}^\top$
\mathbf{A} is not a function of \mathbf{x} , $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$\frac{\partial \mathbf{A} \mathbf{u}}{\partial \mathbf{x}} =$	$\mathbf{A} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \mathbf{A}^\top$
$\mathbf{u} = \mathbf{u}(\mathbf{x})$, $\mathbf{v} = \mathbf{v}(\mathbf{x})$	$\frac{\partial (\mathbf{u} + \mathbf{v})}{\partial \mathbf{x}} =$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	
$\mathbf{u} = \mathbf{u}(\mathbf{x})$	$\frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{x}} =$	$\frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}}$

Matrix-by-scalar

$$Y(x) = \begin{pmatrix} y_{11}(x) & , \dots , & y_{1m}(x) \\ \cdot & \cdot & \cdot \\ y_{n1}(x) & , \dots , & y_{nm}(x) \end{pmatrix}$$

Matrix calculus

$$\frac{dY(x)}{dx} = \begin{pmatrix} \frac{dy_{11}(x)}{dx} & , \dots , & \frac{dy_{1m}(x)}{dx} \\ \frac{dy_{n1}(x)}{dx} & , \dots , & \frac{dy_{nm}(x)}{dx} \end{pmatrix}$$

Matrix calculus

Matrix-by-vector

$$A = \begin{pmatrix} a_{11} & , \dots , & a_{1n} \\ . & . & . \\ a_{n1} & , \dots , & a_{nn} \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ . \\ x_n \end{pmatrix}$$

$$\frac{\partial \mathbf{x}^T A \mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{x}^T A$$

Matrix calculus

Condition	Expression	Numerator layout, i.e. by \mathbf{x}^\top ; result is row vector	Denominator layout, i.e. by \mathbf{x} ; result is column vector
\mathbf{a} is not a function of \mathbf{x}	$\frac{\partial(\mathbf{a} \cdot \mathbf{x})}{\partial \mathbf{x}} = \frac{\partial(\mathbf{x} \cdot \mathbf{a})}{\partial \mathbf{x}} =$ $\frac{\partial \mathbf{a}^\top \mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^\top \mathbf{a}}{\partial \mathbf{x}} =$	\mathbf{a}^\top	\mathbf{a}
\mathbf{A} is not a function of \mathbf{x} \mathbf{b} is not a function of \mathbf{x}	$\frac{\partial \mathbf{b}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{b}^\top \mathbf{A}$	$\mathbf{A}^\top \mathbf{b}$
\mathbf{A} is not a function of \mathbf{x}	$\frac{\partial \mathbf{x}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} =$	$\mathbf{x}^\top (\mathbf{A} + \mathbf{A}^\top)$	$(\mathbf{A} + \mathbf{A}^\top) \mathbf{x}$
\mathbf{A} is not a function of \mathbf{x} \mathbf{A} is <i>symmetric</i>	$\frac{\partial \mathbf{x}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} =$	$2\mathbf{x}^\top \mathbf{A}$	$2\mathbf{A} \mathbf{x}$
\mathbf{A} is not a function of \mathbf{x}	$\frac{\partial^2 \mathbf{x}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}^2} =$	$\mathbf{A} + \mathbf{A}^\top$	
\mathbf{A} is not a function of \mathbf{x} \mathbf{A} is <i>symmetric</i>	$\frac{\partial^2 \mathbf{x}^\top \mathbf{A} \mathbf{x}}{\partial \mathbf{x}^2} =$	$2\mathbf{A}$	

Vector and Matrix Calculus

One way to master vector+matrix calculus is by **simplifying** the formulation into **scalar** cases, which can be understood more easily.

Then to scale up to vector/matrix, always think about the matrix math operations will do, and what the shape of outputs will look like as you progress through the formula

Convexity and optimization

Jason G. Fleischer, Ph.D.

Asst. Teaching Professor

Department of Cognitive Science, UC San Diego

jfleischer@ucsd.edu



@jasongfleischer

<https://jgfleischer.com>

Slides in this presentation are from material kindly provided by
Zhuowen Tu

One hot encoding

Hot or not?

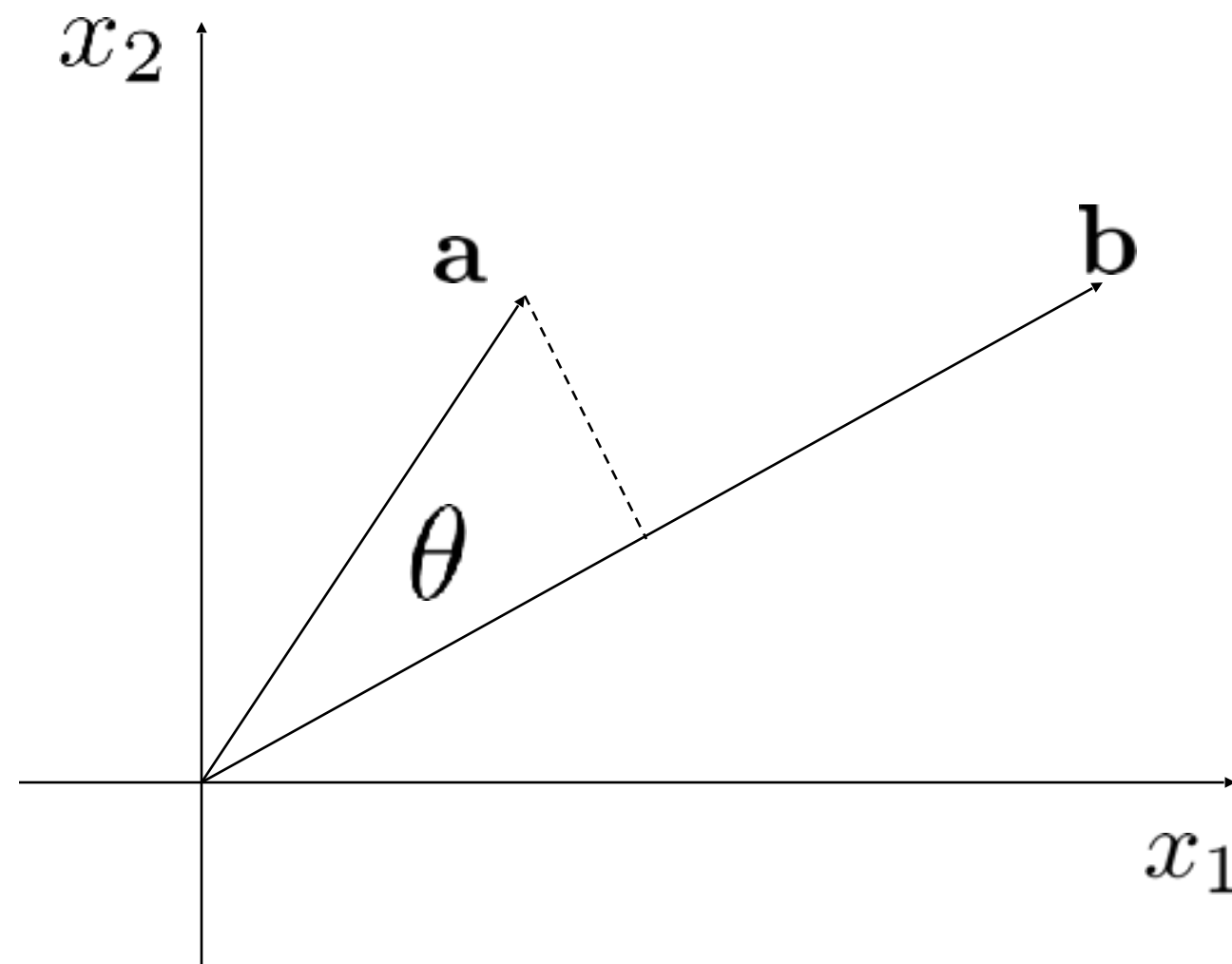
Human-Readable

Machine-Readable

Pet	Cat	Dog	Turtle	Fish
Cat	1	0	0	0
Dog	0	1	0	0
Turtle	0	0	1	0
Fish	0	0	0	1
Cat	1	0	0	0

Orthogonal!
No ordinality!

Vector projection: Inner product



$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\langle \mathbf{a}, \mathbf{b} \rangle \equiv \mathbf{a} \cdot \mathbf{b} \equiv \mathbf{a}^T \mathbf{b} \equiv a_1 b_1 + a_2 b_2 + a_3 b_3$$

It's a scalar!

It's the projection of **a** onto **b**!

It's not normalized!

(what happens when **a** gets longer?)

A normalized projection we can use...

$$\cos(\theta) = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\|_2 \times \|\mathbf{b}\|_2}$$

The “**cosine similarity**” above can be used to measure the “**similarity**” between two vectors (data samples) that are not normalized (non-unit).

Cosine similarity

$$\cos(\theta) = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\|_2 \times \|\mathbf{b}\|_2}$$

A cosine similarity value of 0 indicates two vectors are



A. the least similar

B. the most similar

C. the most uncertain

D. the least uncertain

Cosine similarity

	fly?	laying eggs?	weight (lb)
sparrow	yes	yes	0.087
chipmunk	no	no	0.19
bat	yes	no	0.09

Feature representation (one-hot encoded).

$$\text{sparrow} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0.087 \end{pmatrix} \quad \text{chipmunk} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0.19 \end{pmatrix} \quad \text{bat} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0.09 \end{pmatrix}$$

$$\frac{\text{sparrow} \cdot \text{chipmunk}}{\|\text{sparrow}\|_2 \times \|\text{chipmunk}\|_2} = 0.0082$$

\cdot refers to the dot product between two vectors;

$$\frac{\text{sparrow} \cdot \text{bat}}{\|\text{sparrow}\|_2 \times \|\text{bat}\|_2} = 0.502$$

$\|\cdot\|_2$ refers to the L2 norm of a vector;

$$\frac{\text{chipmunk} \cdot \text{bat}}{\|\text{chipmunk}\|_2 \times \|\text{bat}\|_2} = 0.503$$

\times refers to the multiplication of two scalar values.

Feature scaling is another factor

Now we purposely **stretch** one particular feature dimension by a large factor. Let's see what will happen.

$$\text{sparrow} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 87 \end{pmatrix} \quad \text{chipmunk} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 190 \end{pmatrix} \quad \text{bat} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 90 \end{pmatrix}$$

$$\frac{\text{sparrow} \cdot \text{chipmunk}}{\|\text{sparrow}\|_2 \times \|\text{chipmunk}\|_2} = 0.99984$$

$$\frac{\text{sparrow} \cdot \text{bat}}{\|\text{sparrow}\|_2 \times \|\text{bat}\|_2} = 0.99987$$

$$\frac{\text{chipmunk} \cdot \text{bat}}{\|\text{chipmunk}\|_2 \times \|\text{bat}\|_2} = 0.99990$$

Now, the concept of similarity diminishes.

Conclusion: The **relative scaling** of the individual features is also important.

In practice, we often **normalize** the individual features to $[0, 1]$ to make them directly comparable.

Measuring similarity

Two methods: Dot product, Cosine Similarity

- Dot product: cares about vector length and direction
 - Max value when parallel: product of the magnitude of the vectors
 - Min value when orthogonal: 0
- Cosine similarity: cares only about direction, not length
 - Max value when parallel: 1
 - Min value when orthogonal: 0
- Both methods can have their answers changed by an arbitrary rescaling of one of the features in the vector
- To measure similarity for non-ordinal categorical data you should almost certainly use one-hot encoding

🔍 machine learning is just [Search Google](#)

Google Suggestions

🔍 **Machine learning is just picking an appropriate model and then minimizing a loss function**

- 🔍 machine learning is just statistics
- 🔍 machine learning is just if statements
- 🔍 machine learning is just a bunch of if statements
- 🔍 machine learning is just curve fitting

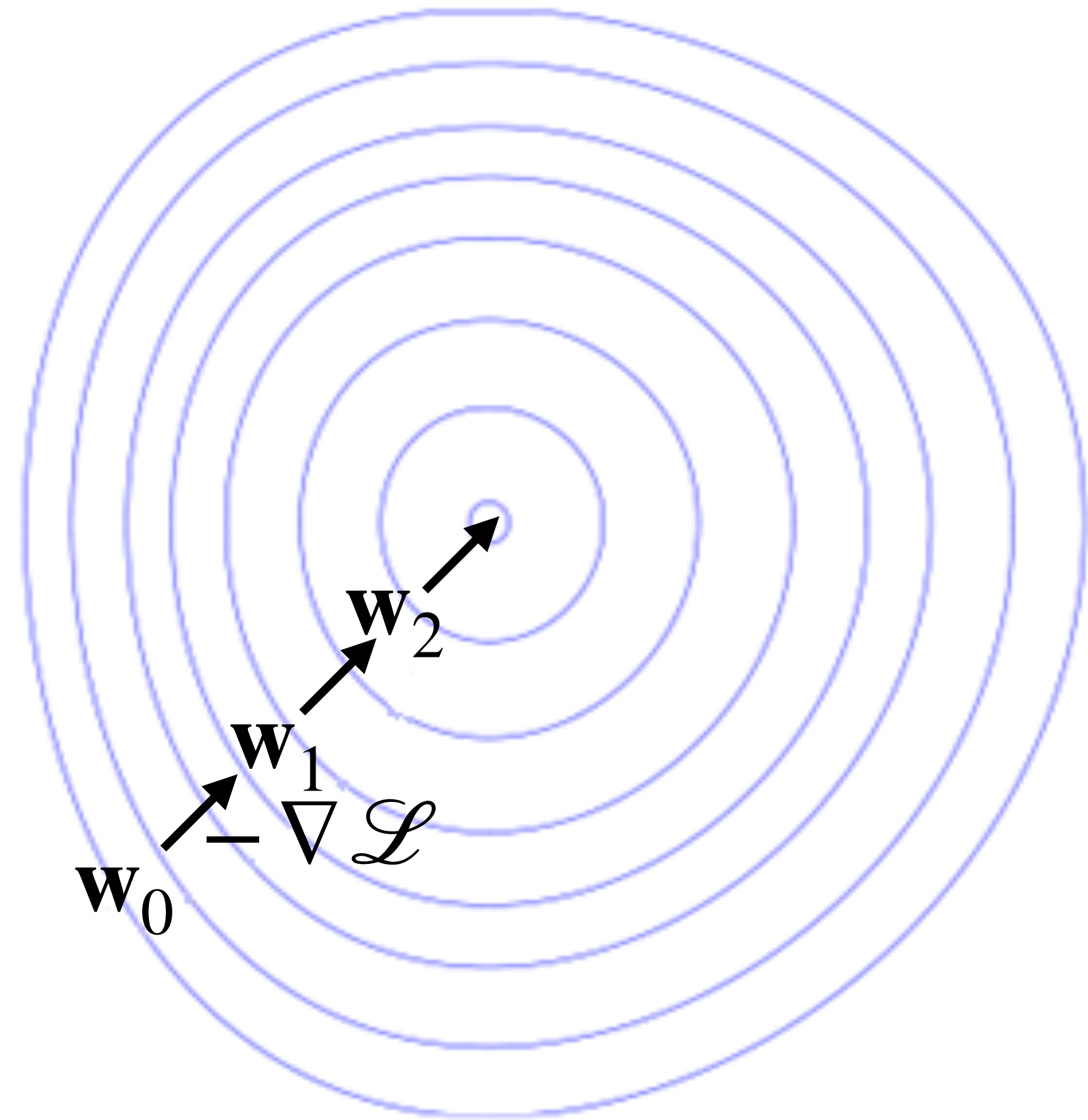
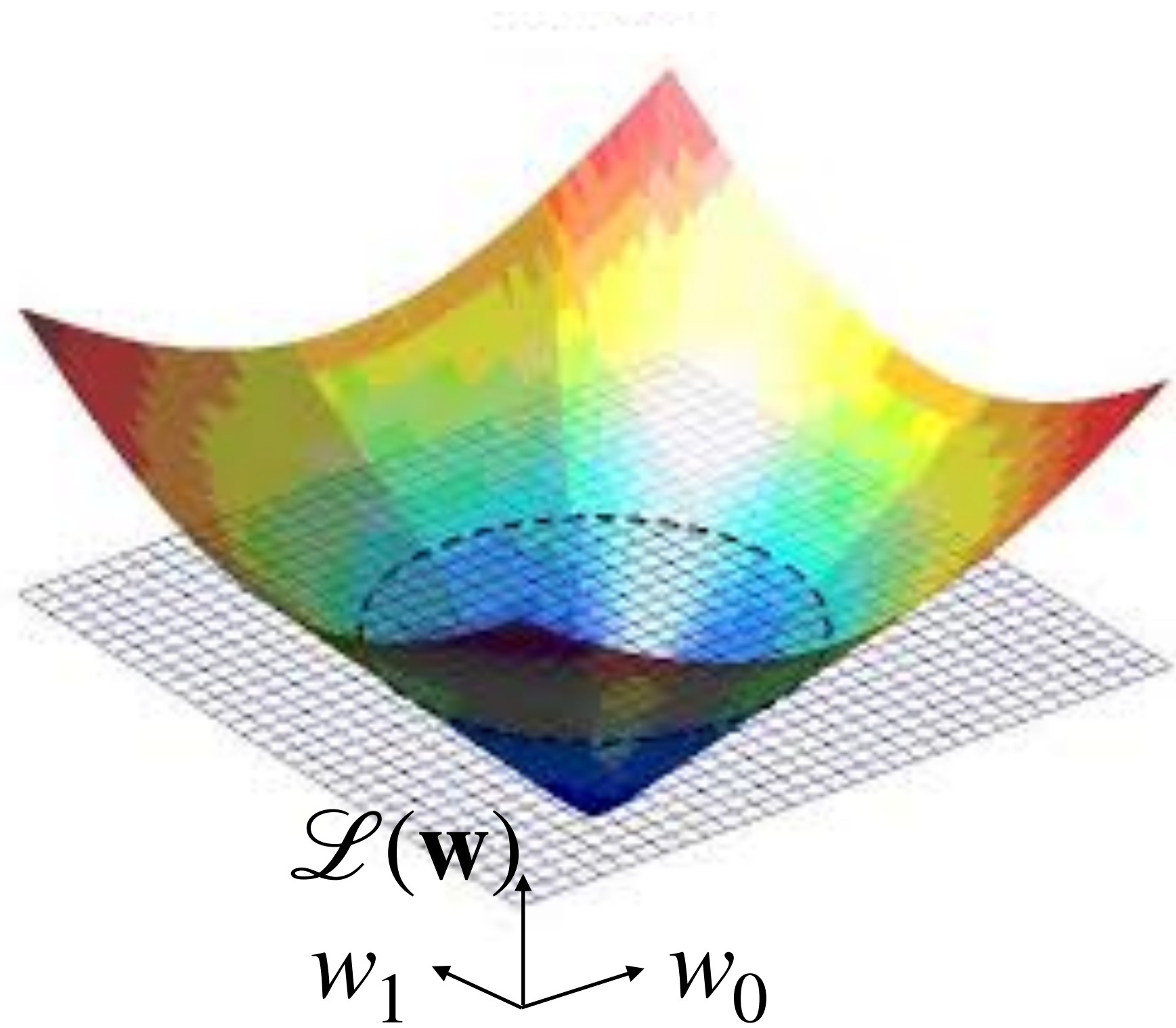
A rose by any other name

When optimizing a function we can call the function:





- Loss function
- Cost function
- Objective function
- Fitness function
- And probably some others

$$\mathcal{L}(\mathbf{w}) = g(\mathbf{y} - f(\mathbf{X}\mathbf{w}))$$

$$\nabla \mathcal{L} \equiv \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$$



Optimization math:
argmin and argmax

ForbesBillionairesInnovationLeadershipMoneyBusinessSmall BusinessLifestyleReal EstateShoppingCoupons									
Filter list by: OLDEST YOUNGEST WOMEN INDUSTRY COUNTRY/TERRITORY									
	RANK	NAME		NET WORTH	CHANGE		AGE		
	1	Jeff Bezos		\$181.5 B	\$0		57		Am
	2	Elon Musk		\$179.2 B	\$0		50		Tex
	3	Bernard Arnault & family		\$146.1 B	▲ \$1.6 B 1.09%		72		LV
	4	Bill Gates		\$120.2 B	▲ \$20 M 0.02%		66		M

$$\max_{\text{person}} \text{NetWorth}(\text{person}) = 181 \text{ billion dollars}$$

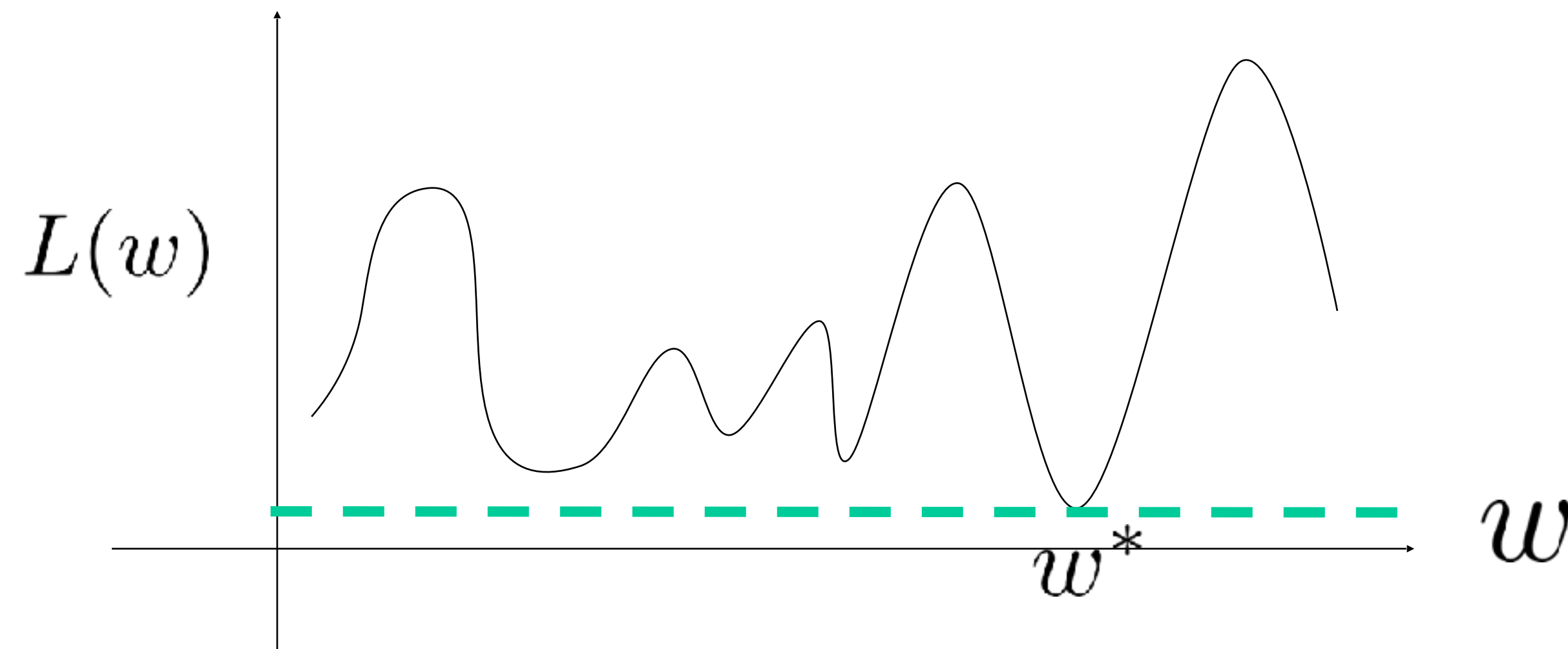
$$\arg \max_{\text{person}} \text{NetWorth}(\text{person}) = \text{Jeff Bezos}$$

Optimization: argmin

$$w^* = \arg \min_w L(w)$$

The operator $\arg \min$ defines the optimal value (in the argument of function $L()$) w^* that minimizes $L(w)$

$\arg \min L(w)$ doesn't return the value of $L(w)$

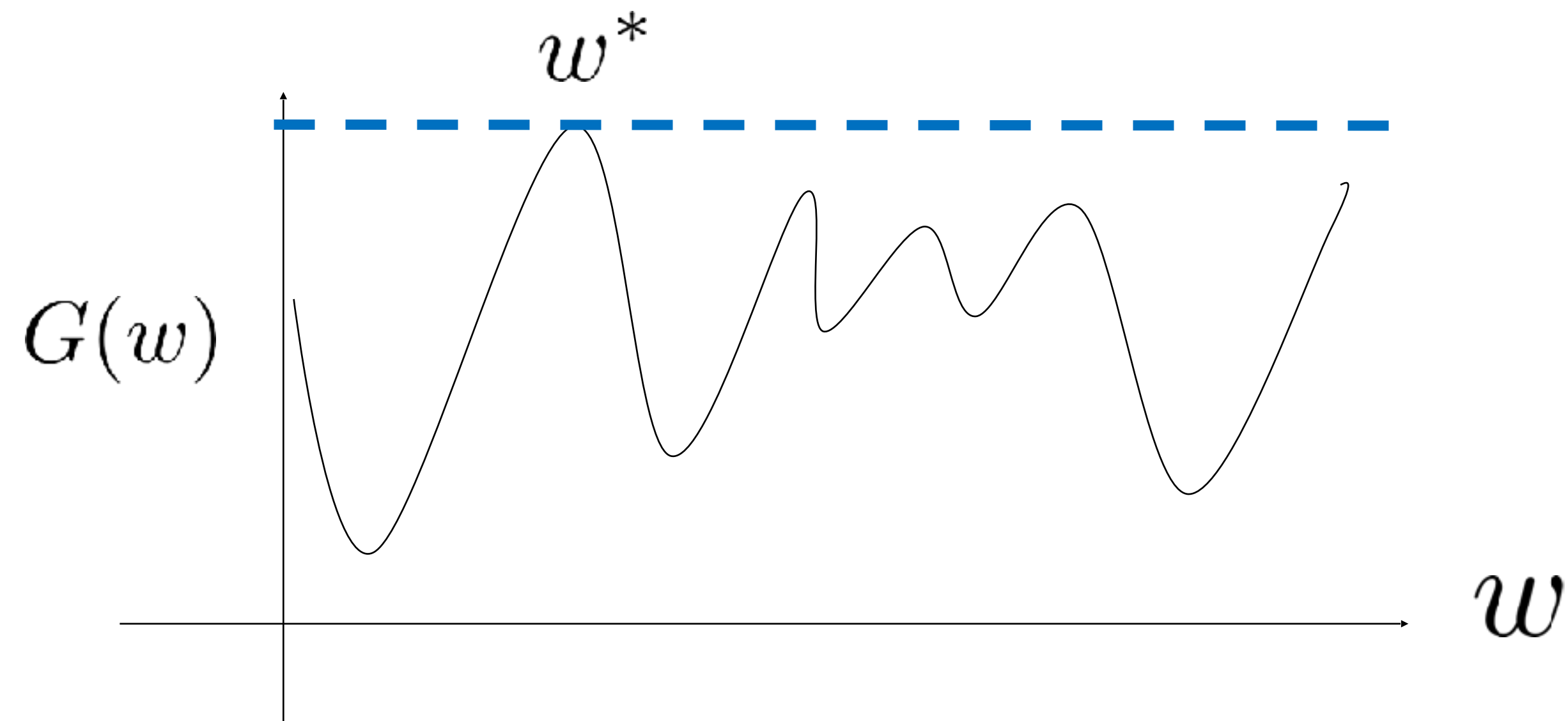


Optimization: argmax

$$w^* = \arg \max_w G(w)$$

The operator $\arg \max$ defines the optimal value (in the argument of function $G()$) w^* that maximizes $G(w)$

$\arg \max G(w)$ doesn't return the value of $G(w)$



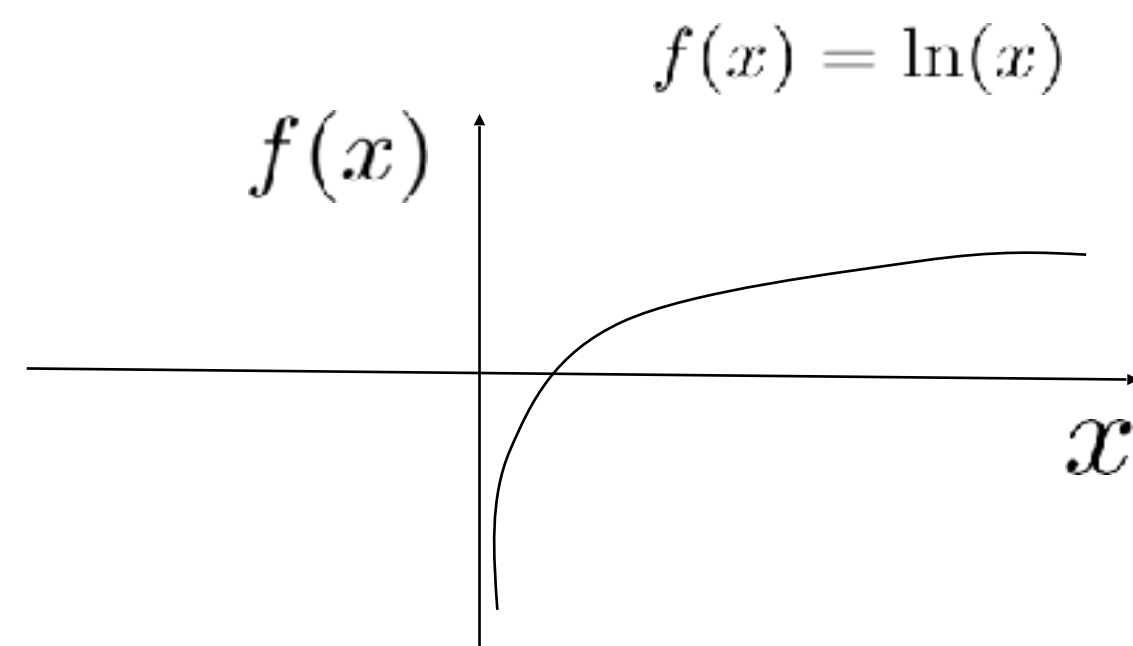
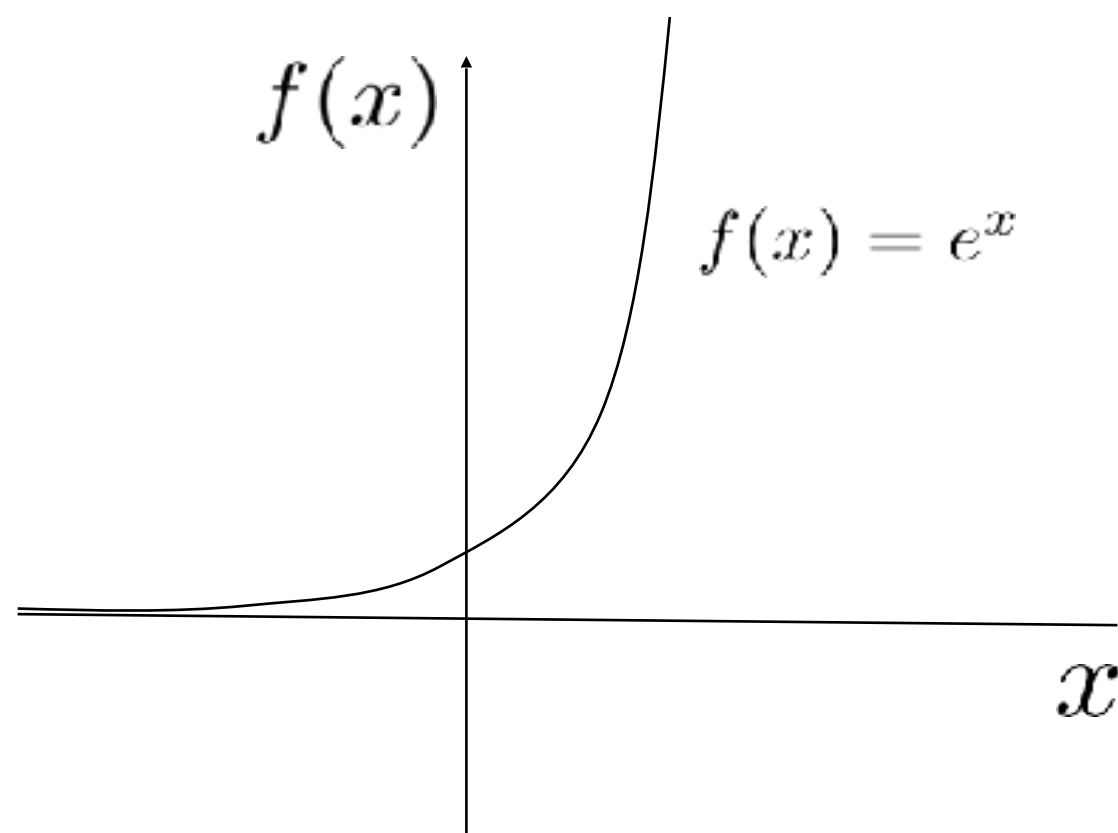
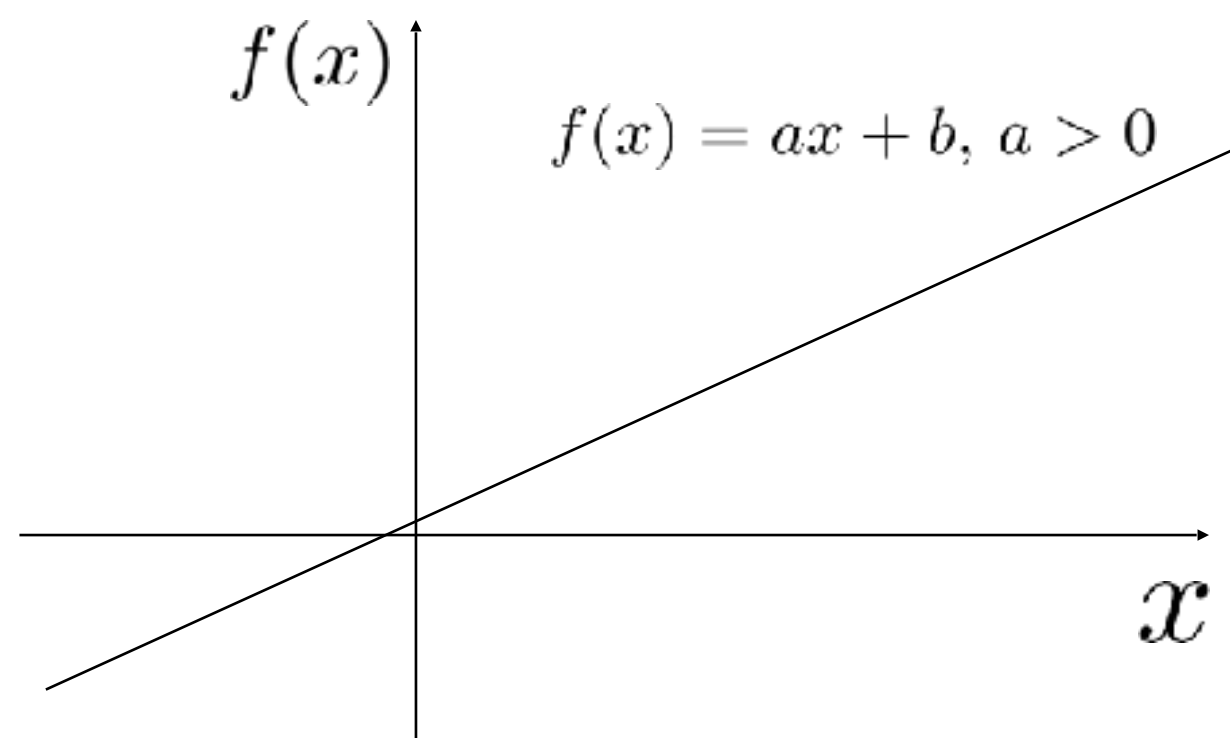
argmin and argmax

$$\arg \max_x f(x) = \arg \min_x -f(x)$$

$$\arg \max_{\text{person}} \text{NetWorth}(\text{person}) = \arg \min_{\text{person}} -\text{NetWorth}(\text{person})$$

**Transforming the loss function to
make optimization easier**

Monotonic functions



For a monotonically increasing
function:

$$f(x)$$

Is $-f(x)$ monotonically decreasing?

A. Yes

B. No

C. It depends

For a monotonically increasing
function:

$$f(x)$$

Is $-f(x)$ monotonically decreasing?

- ★ A. Yes
- B. No
- C. It depends

For a monotonically increasing
function:

$$f(x) \in R^+$$

Is $\ln f(x)$ monotonically increasing?

- A. Yes
- B. No
- C. It depends

Is $\ln f(x)$ monotonically increasing?

★ A. Yes

B. No

C. It depends

For a monotonically increasing
function:

$$f(x) \in R^+$$

For two monotonically
increasing functions:

$f(x)$ and $g(x)$

Is $f(x) + g(x)$ monotonically increasing?

- A. Yes
- B. No
- C. It depends

For two monotonically
increasing functions:

$f(x)$ and $g(x)$

Is $f(x) + g(x)$ monotonically increasing?

- ★ A. Yes
- B. No
- C. It depends

For two monotonically
increasing functions:

$f(x)$ and $g(x)$

Is $f(x) - g(x)$ monotonically increasing?

A. Yes

B. No

C. It depends

For two monotonically
increasing functions:

$f(x)$ and $g(x)$

Is $f(x) - g(x)$ monotonically increasing?

- A. Yes
- B. No
- ★ C. It depends

Monotonic transformation

$$w^* = \arg \min_w L(w)$$

If a function $g(v)$ is monotonic, e.g. $\forall v_1 > v_2$ it is always true that $g(v_1) > g(v_2)$, then:

$$w^* = \arg \min_w L(w) = \arg \min_w g(L(w))$$

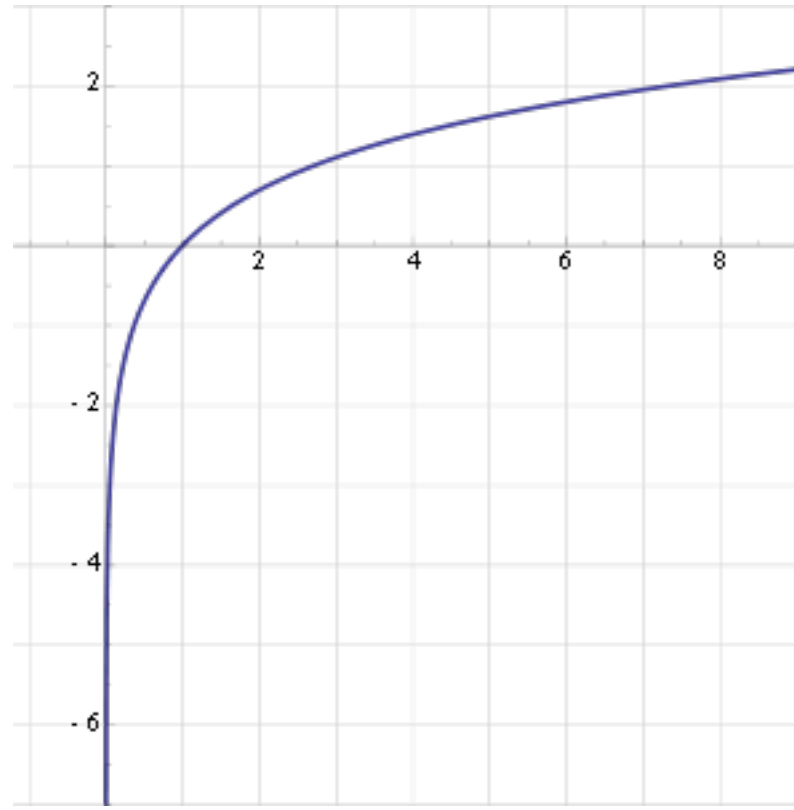
For example,

$$\text{if } g(v) = 2 \times v + 10$$

$$w^* = \arg \min_w L(w) = \arg \min_w 2 \times L(w) + 10$$

argmin and argmax

The function $\ln(v)$ is monotonically increasing, e.g. $\forall v_1 > v_2$ it is always true that $\ln(v_1) > \ln(v_2)$, then:



$$\begin{aligned} w^* &= \arg \max_w G(w) \\ &= \arg \max_w \ln(G(w)) \\ &= \arg \min_w -\ln(G(w)) \end{aligned}$$

Applying a monotonic function to another function does not change its arg min or arg max!

Adding a constant to a function does not change its arg min or arg max!

An actual ML example

Log loss in logistic regression

We want to maximize the probability of correct classification of each datapoint

$$w^* = \arg \max_w \prod_{i=1}^n p(y_i | x_i; w)$$

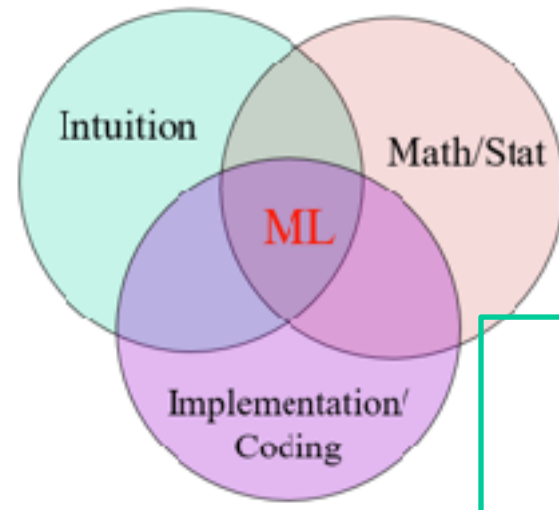
Product because each prob
Is independent

$$= \arg \max_w \ln[\prod_{i=1}^n p(y_i | x_i; w)]$$

Products suck, sums rule: apply a monotonic
transformation

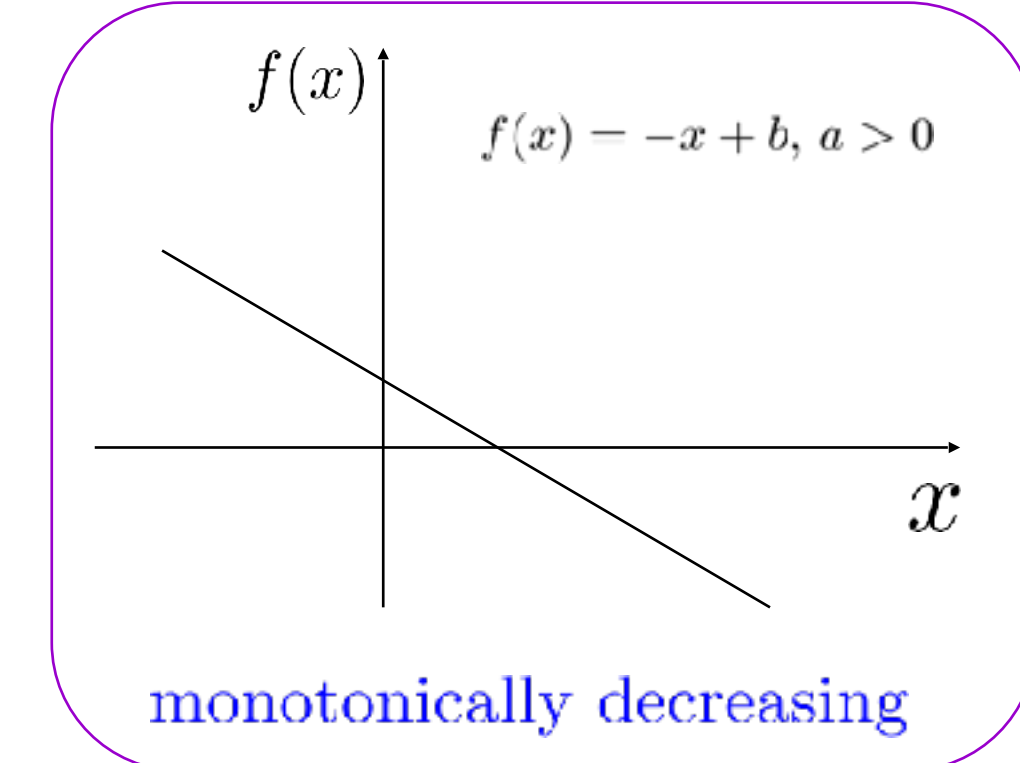
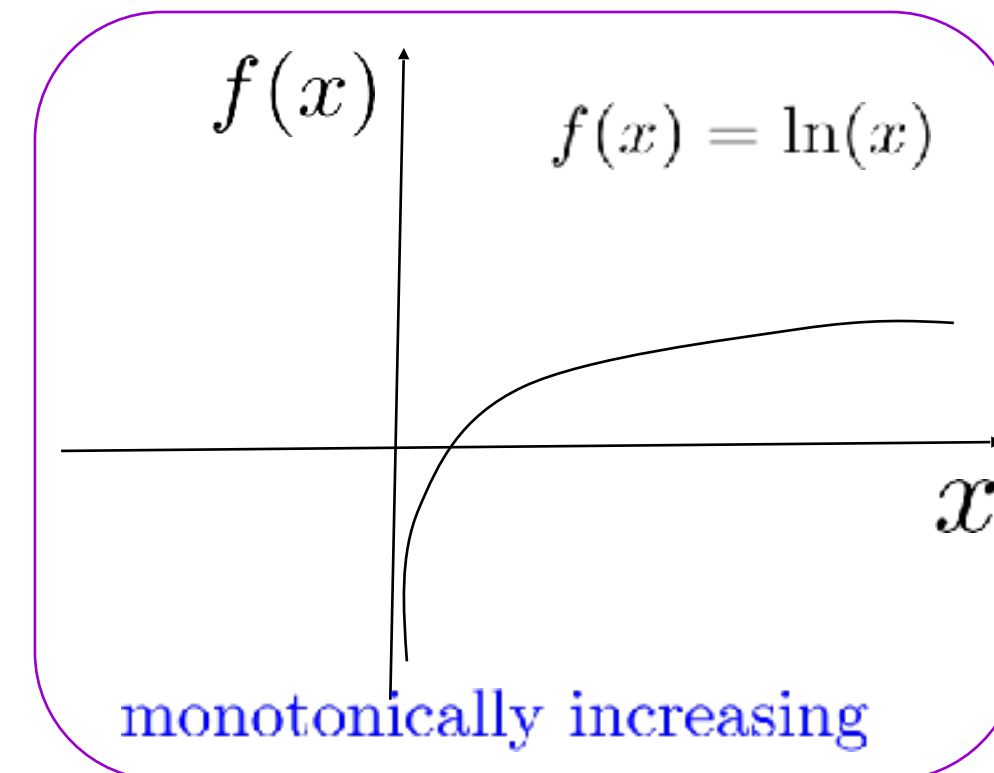
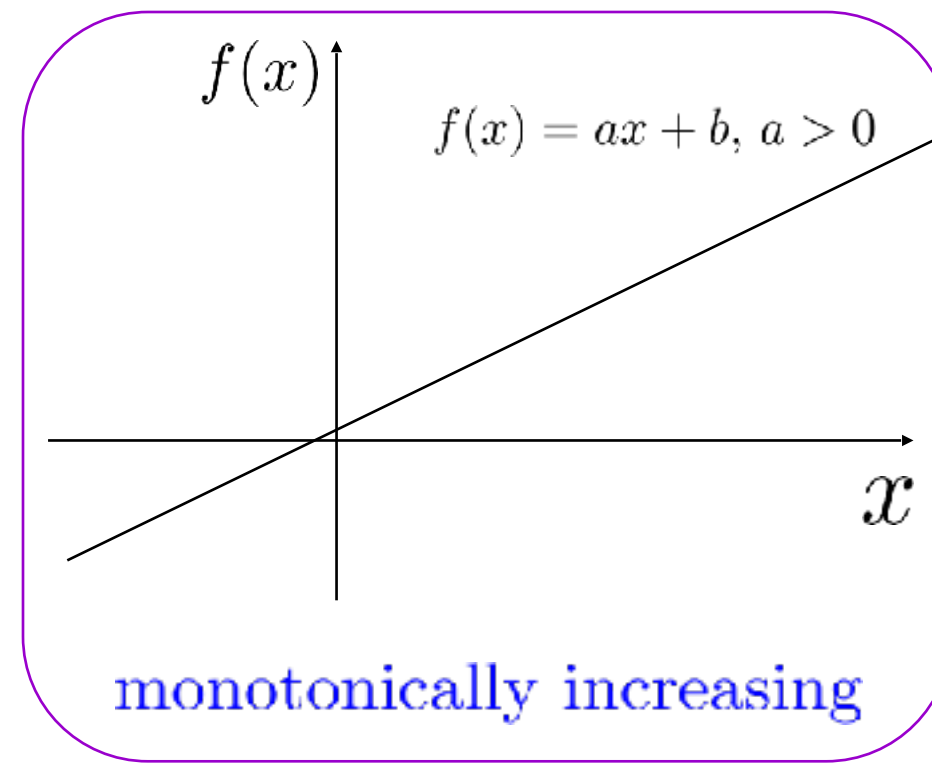
$$= \arg \min_w - \sum_{i=1}^n \ln[p(y_i | x_i; w)]$$

Turn it into a minimization of loss instead of
maximization of probability correct



Recap: Monotonicity

Intuition: In machine learning, we use the monotonicity of functions to help significantly reduce the difficulty/complexity of an **estimation/learning** problem.



Math: If a function $g(v)$ is monotonic, e.g. $\forall v_1 > v_2$ it is always true that $g(v_1) > g(v_2)$, then:

$$w^* = \arg \min_w L(w) = \arg \min_w g(L(w))$$

$\mathbf{1}(f(x))$ means
take value 1 wherever $f(x)$ is true
otherwise take value 0

Machine learning is
optimization

$$e_{testing} = e_{training} + generalization(f)$$

Ideally: minimize $e_{testing}$

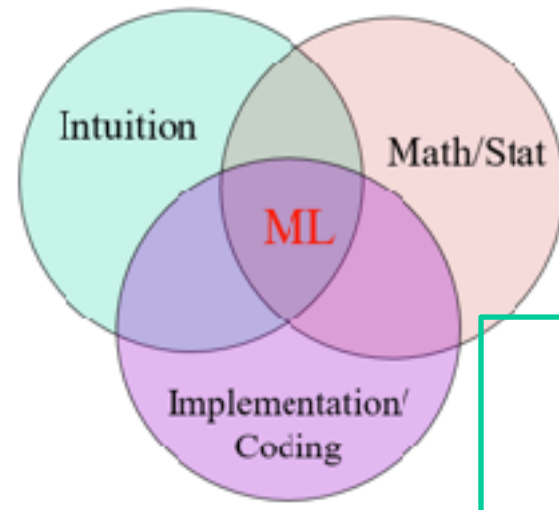
As a proxy : minimize $e_{training}$

$$\text{Minimize } \frac{1}{n} \sum_{i=1}^n \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

In general: $W^* = \arg \min_W \mathcal{L}(W)$, where $\mathcal{L}(W) = e_{training}$
defines a **loss/objective** function in machine learning.

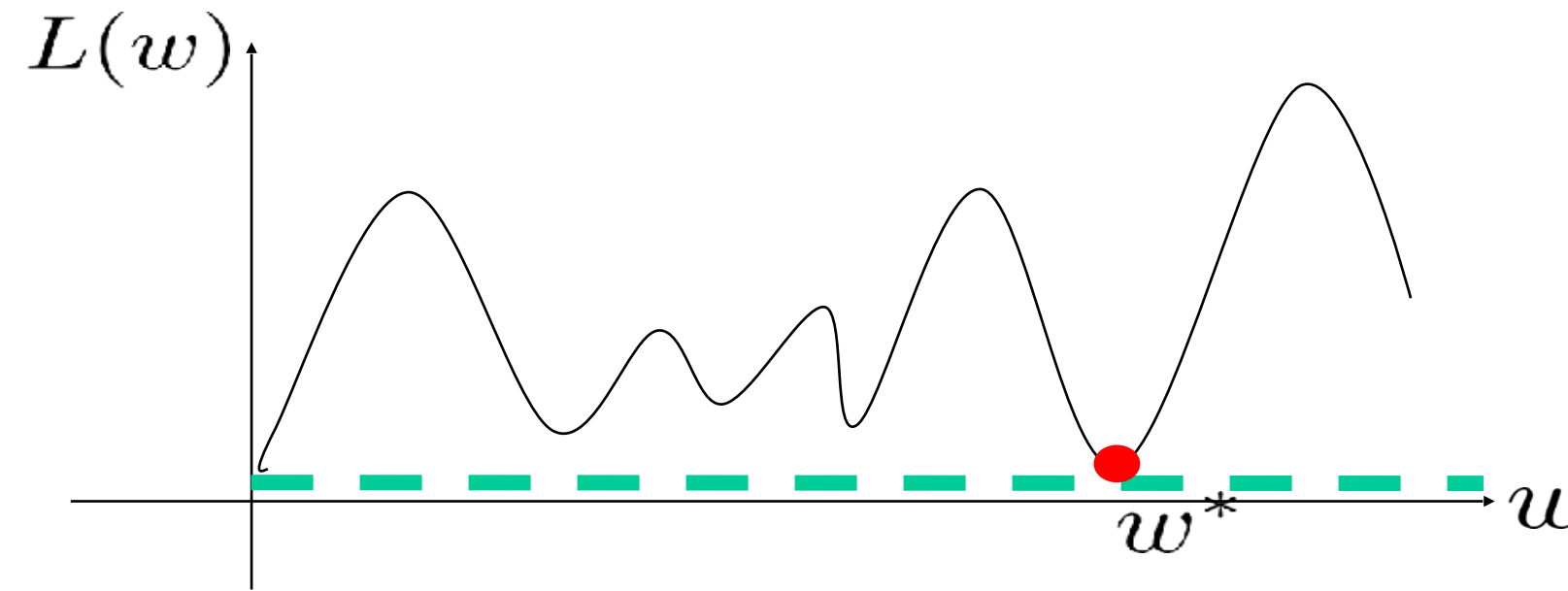
The “model” or “algorithm” consists of both W^* and the loss function!

There are ML algorithms which share the same parameter form,
but different loss functions and vice versa



Recap: Optimization

Intuition: Typically, **optimization** in machine learning refers the process in which an objective function is **minimized/maximized**. A major task in machine learning is to perform training to attain the **optimal parameter** that minimizes/maximizes the corresponding objective function. Note that the optimal model parameter is **not** the minimal/maximal value of the function itself.



Math:

$$w^* = \arg \min_w L(w)$$

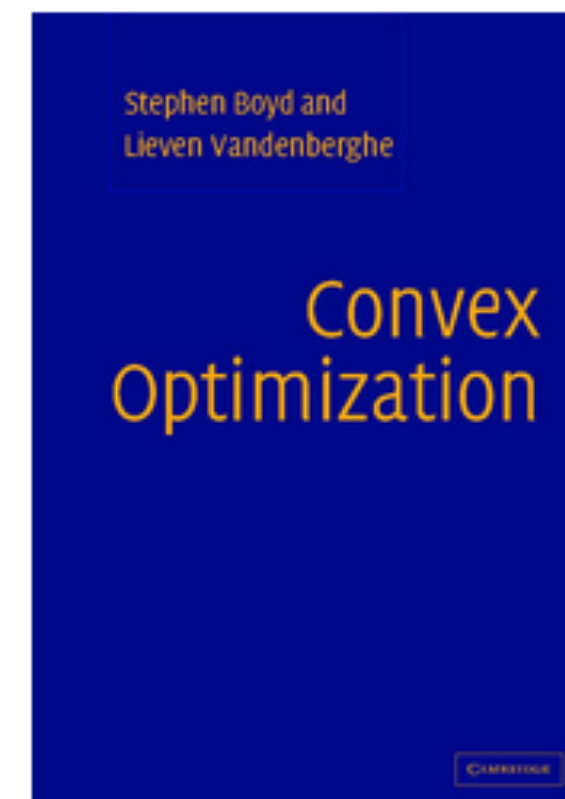
How do we do this?

$$w^* = \arg \min_w L(w)$$

Learning and estimation with convex functions:

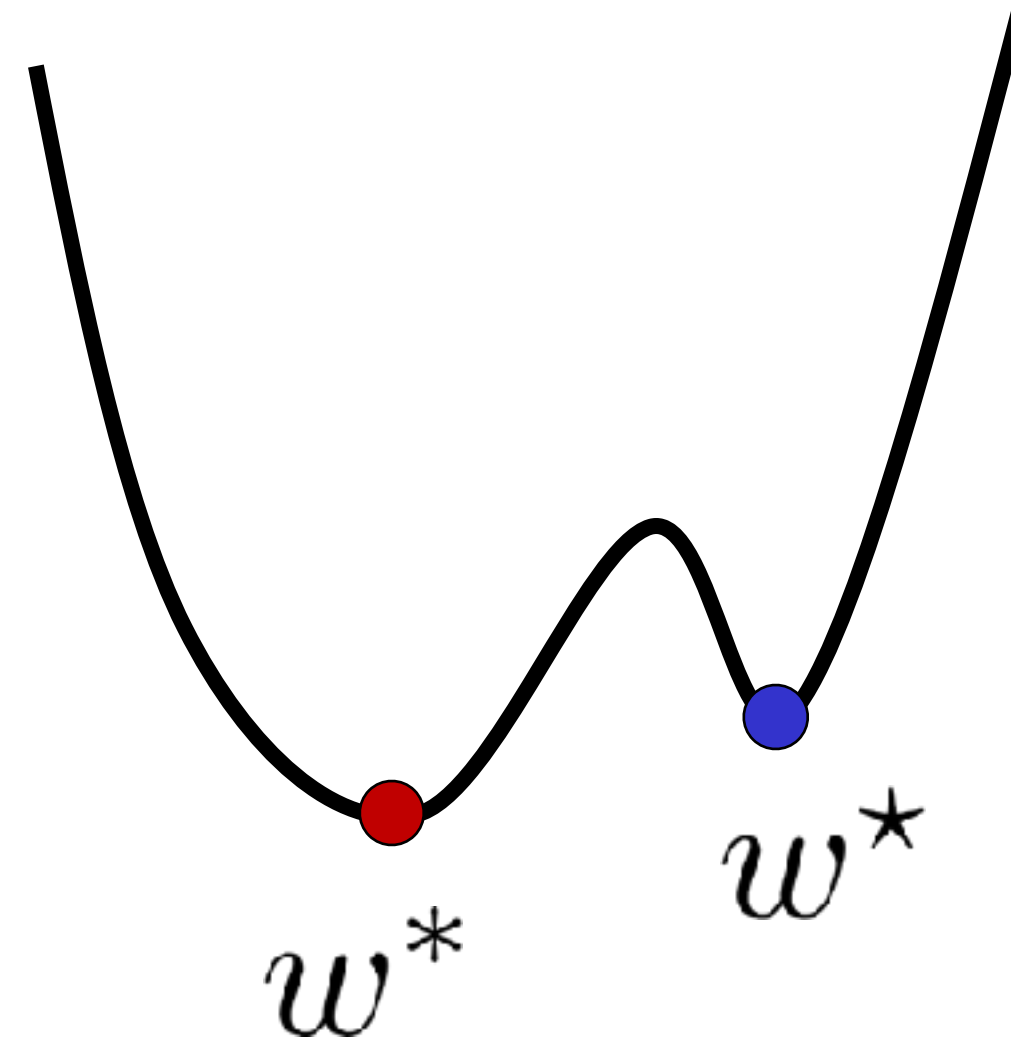


<http://stanford.edu/~boyd/>



Convex Optimization
Stephen Boyd and Lieven Vandenberghe
Cambridge University Press

Optimization

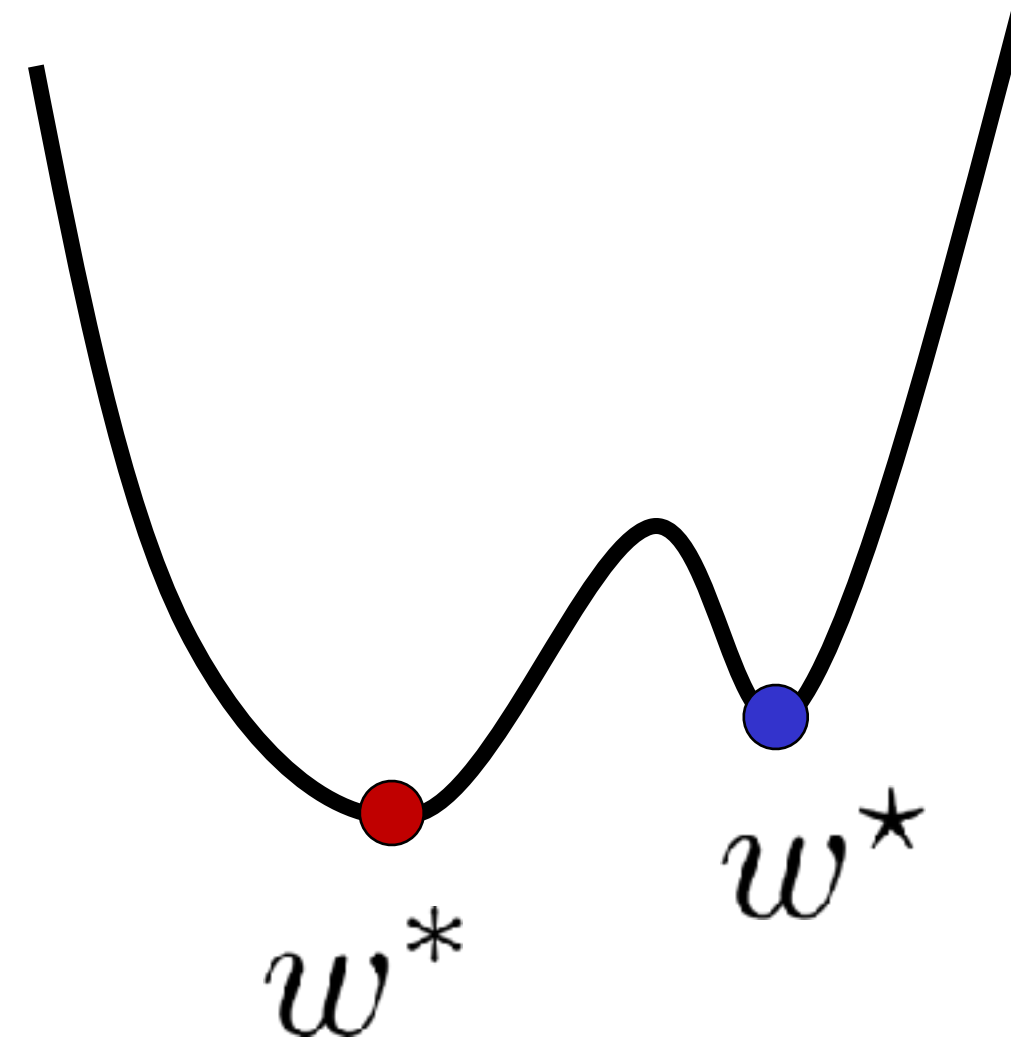


Ω is the domain of w
ie., the set of all possible values
(Or all possible values that could be solutions)

Definition:

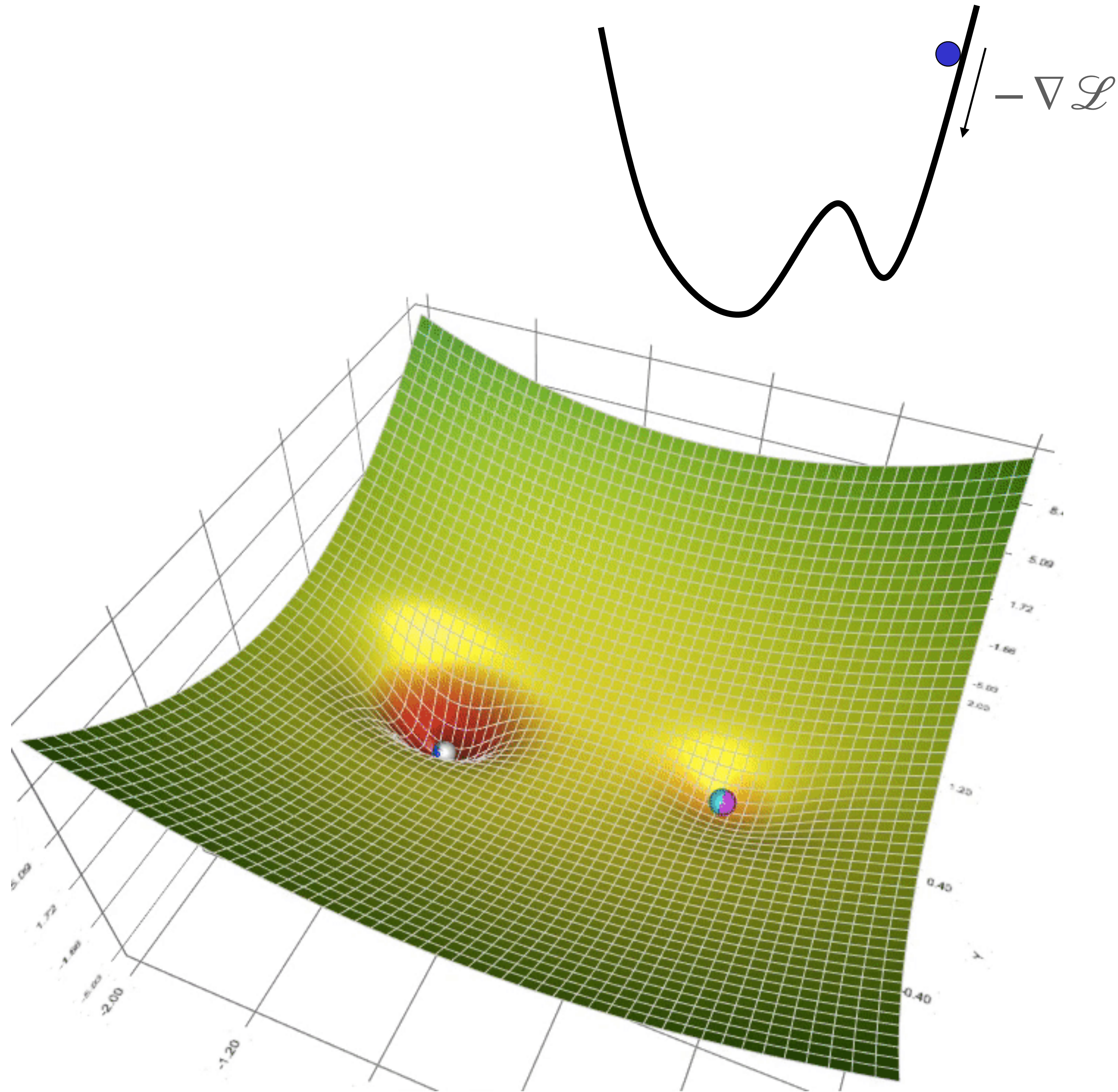
1. w^* is a **globally optimal** solution for $w^* \in \Omega$ and $L(w^*) \leq L(w) \forall w \in \Omega$
2. w^* is a **locally optimal** solution if there is a neighborhood \mathcal{N} around w such that $w^* \in \Omega$, $L(w^*) \leq L(w)$, $\forall w \in \mathcal{N} \cap \Omega$.

Optimization



Things we often need to be able to do to solve optimization problem:

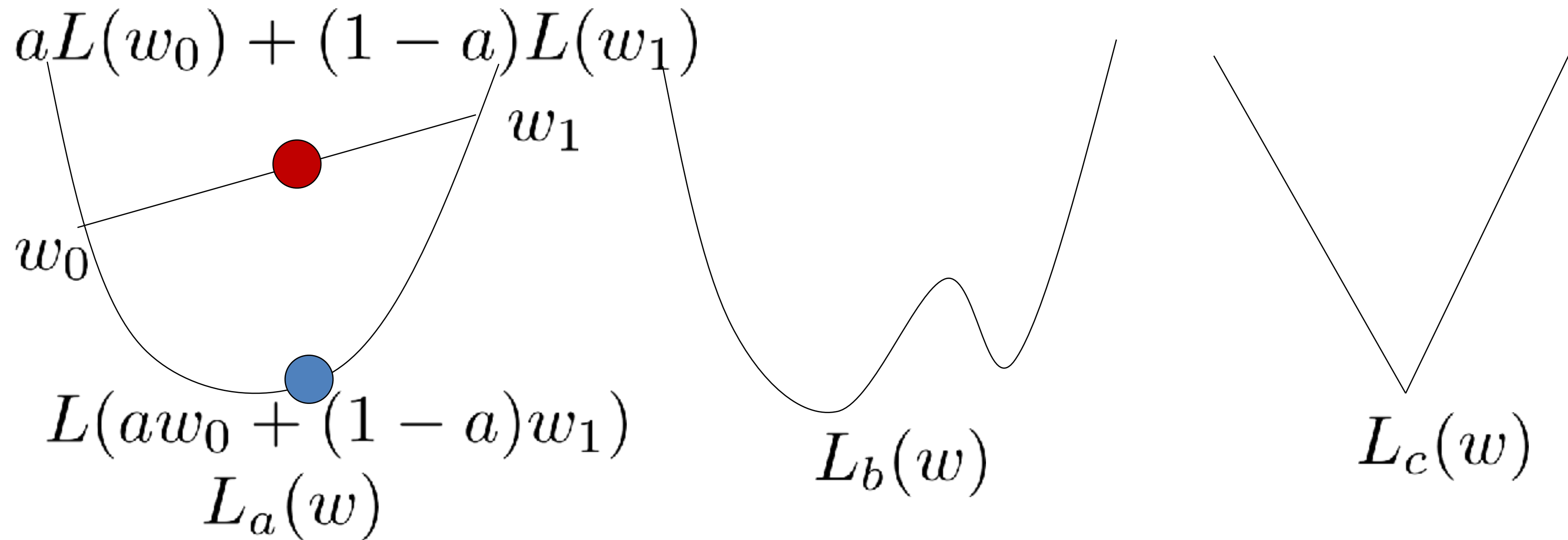
1. $\forall w$, check if $w \in \Omega$?
2. For $\forall w$, computing $L(w)$, $\nabla L(w)$, $\nabla^2 L(w)$.



Convex loss functions make optimization easy!

Convex functions

$$w^* = \arg \min_w L(w)$$



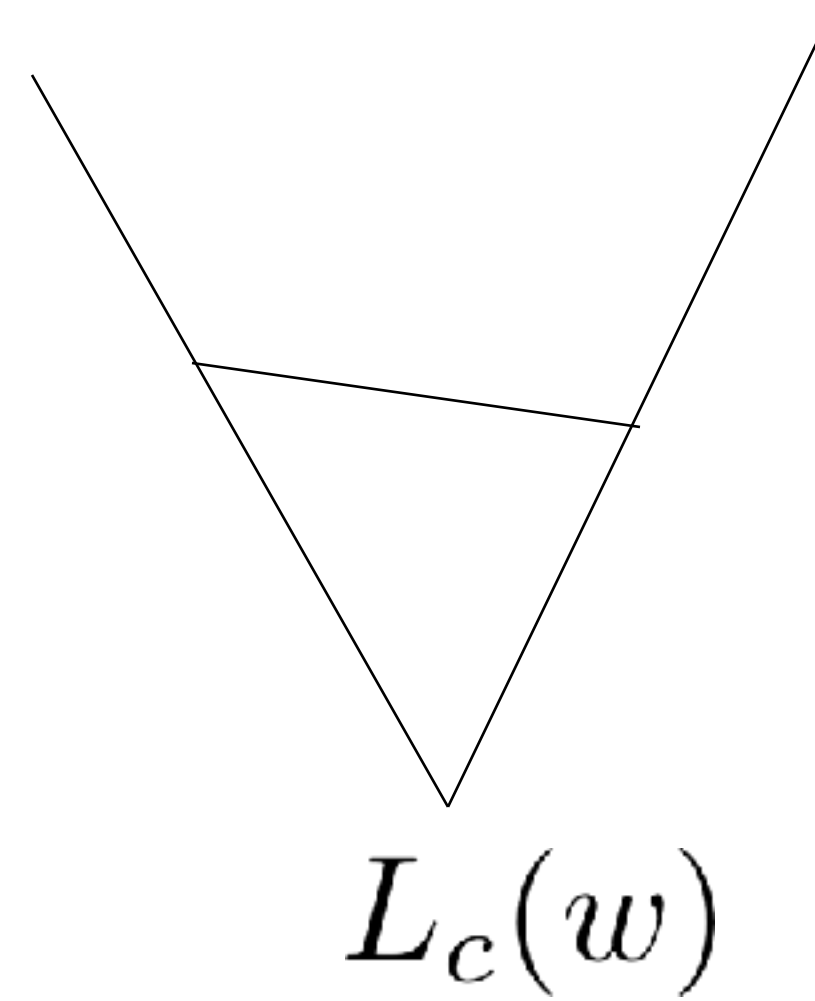
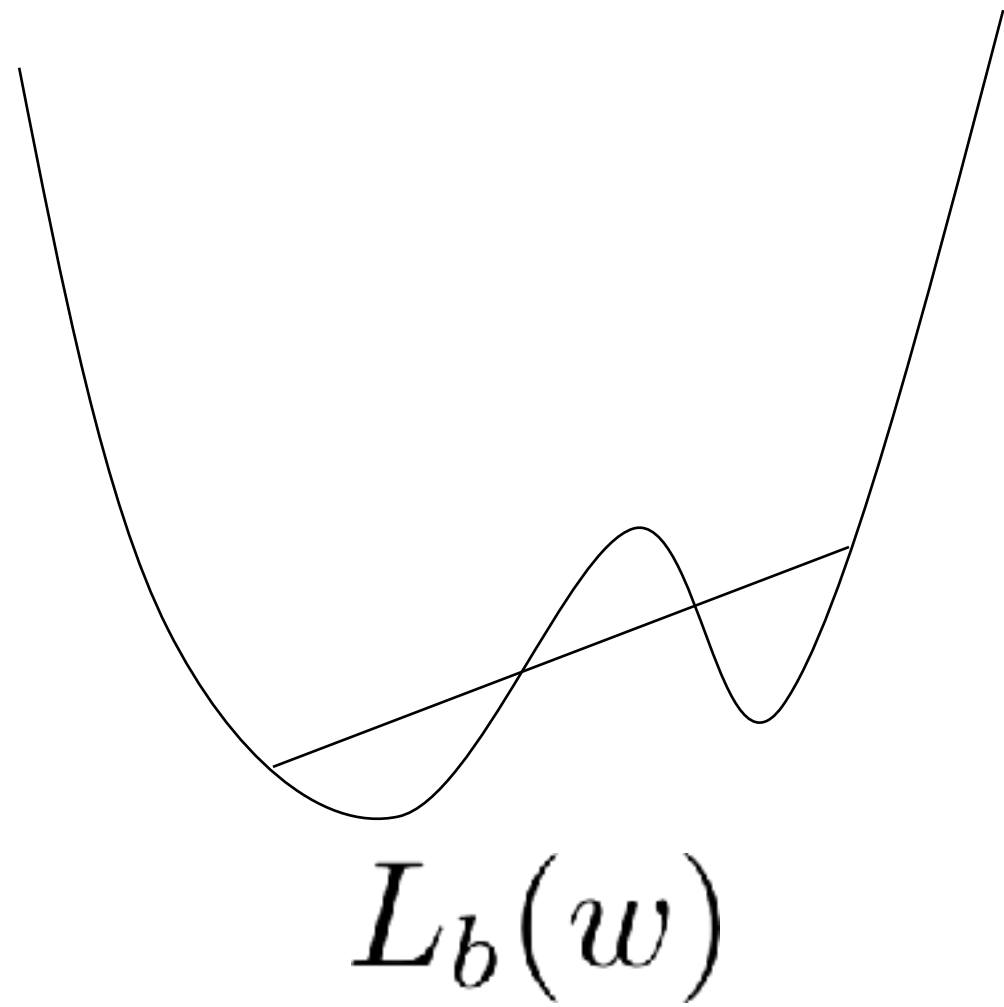
Definition:

$$\forall w_0, w_1, a \in [0, 1]$$

$$aL(w_0) + (1-a)L(w_1) \geq L(aw_0 + (1-a)w_1)$$

Convex functions

$$w^* = \arg \min_w L(w)$$

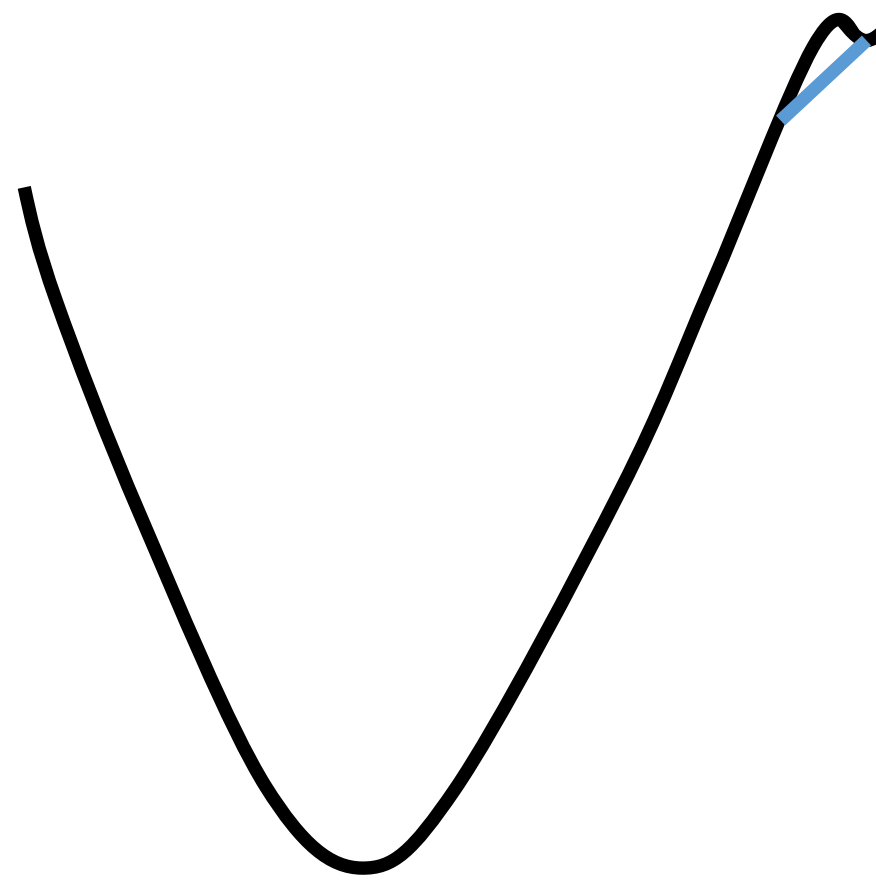


$$\forall w_0, w_1, a \in [0, 1]$$

$$aL(w_0) + (1 - a)L(w_1) \geq L(aw_0 + (1 - a)w_1)$$

Convexity

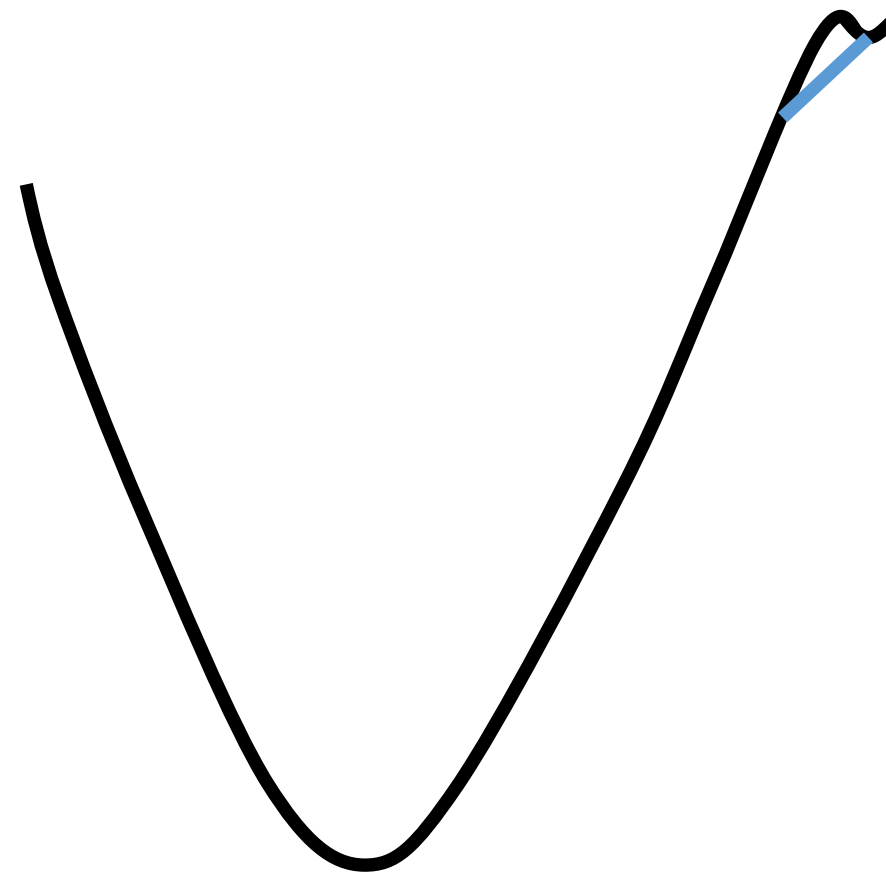
Is this a convex function?



- A. Yes
- B. No
- C. It depends

Convexity

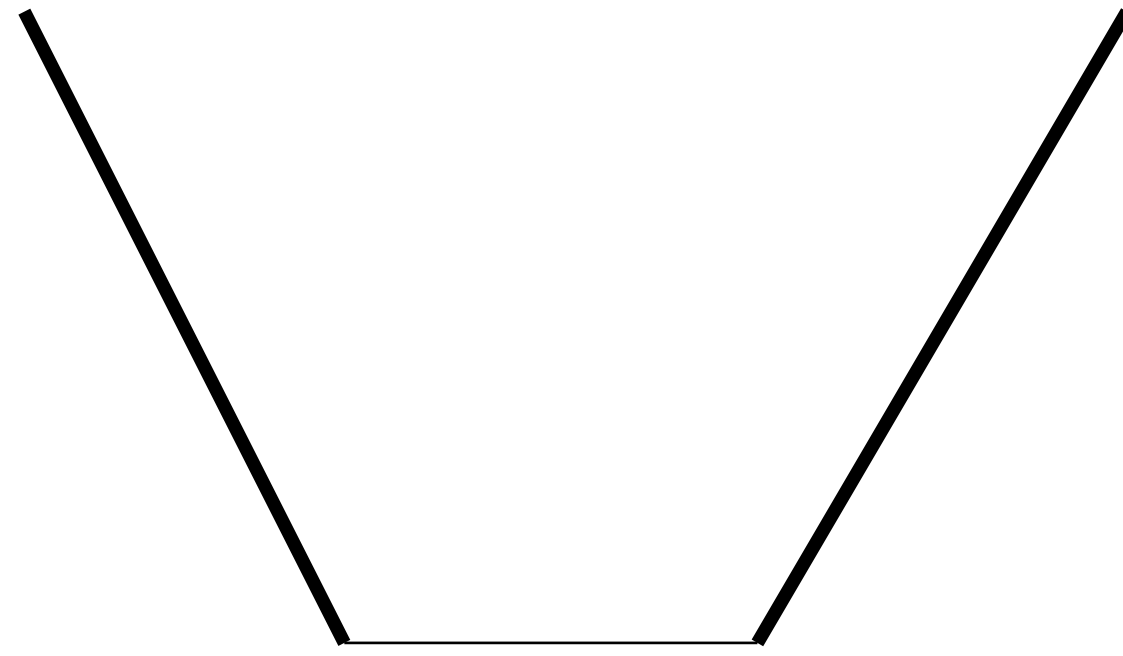
Is this a convex function?



A. Yes

★ B. No

C. It depends



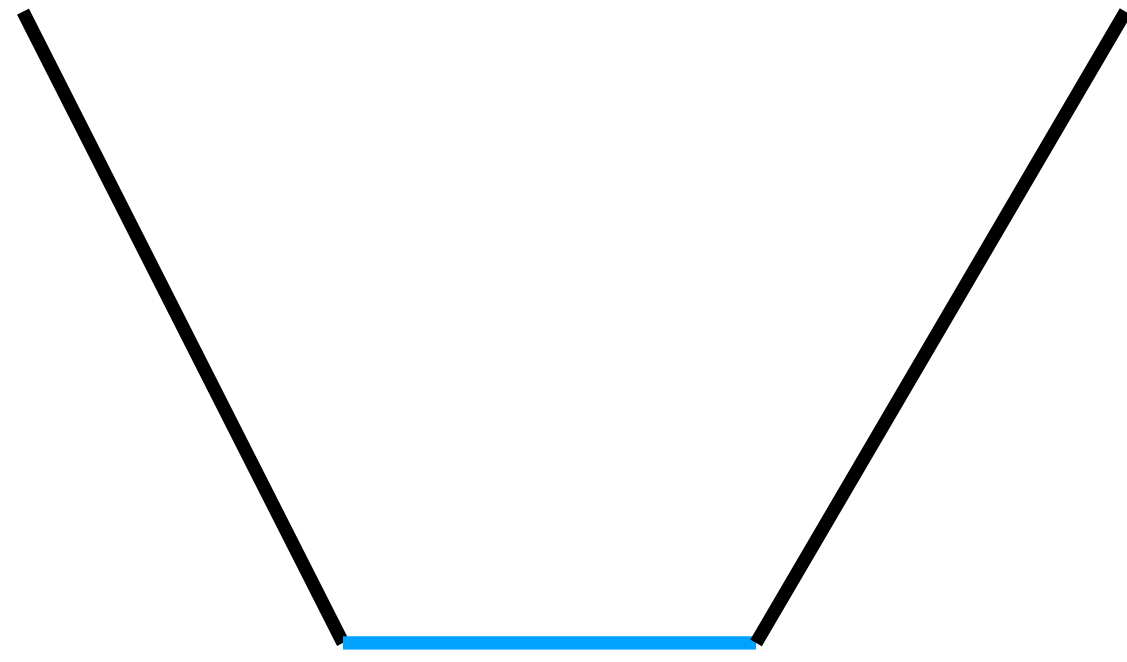
Convexity

Is this a convex function?

- A. Yes
- B. No
- C. It depends

But not strictly convex

$$aL(w_0) + (1 - a)L(w_1) > g(aw_0 + (1 - a)w_1)$$



Convexity

Is this a convex function?

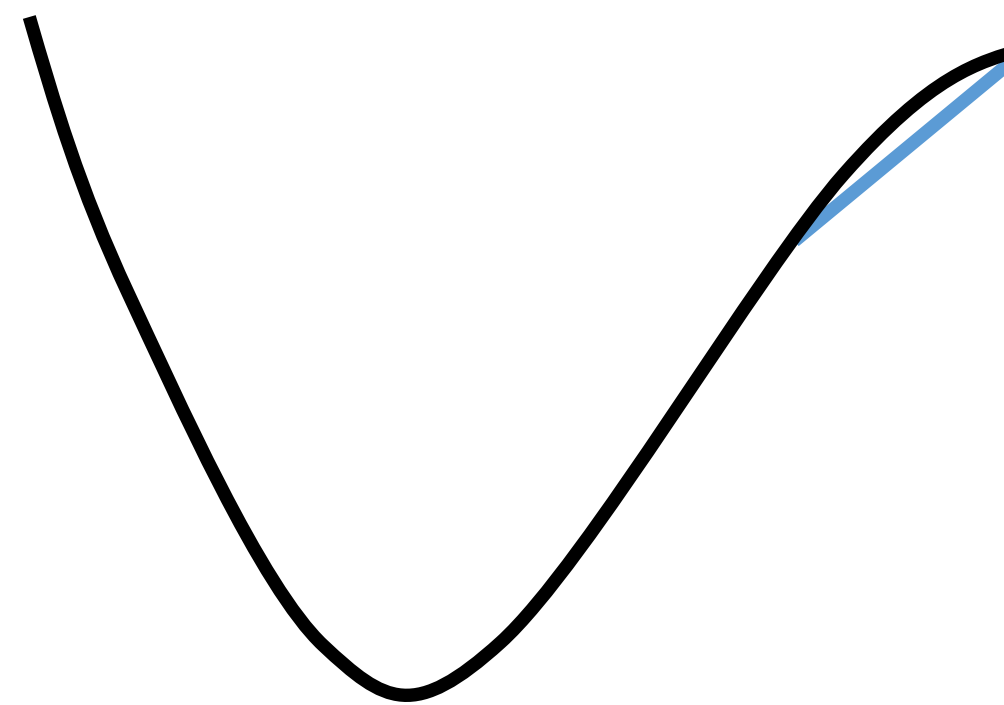
- ★ A. Yes
- B. No
- C. It depends

But not strictly convex

$$aL(w_0) + (1 - a)L(w_1) > L(aw_0 + (1 - a)w_1)$$

Convexity

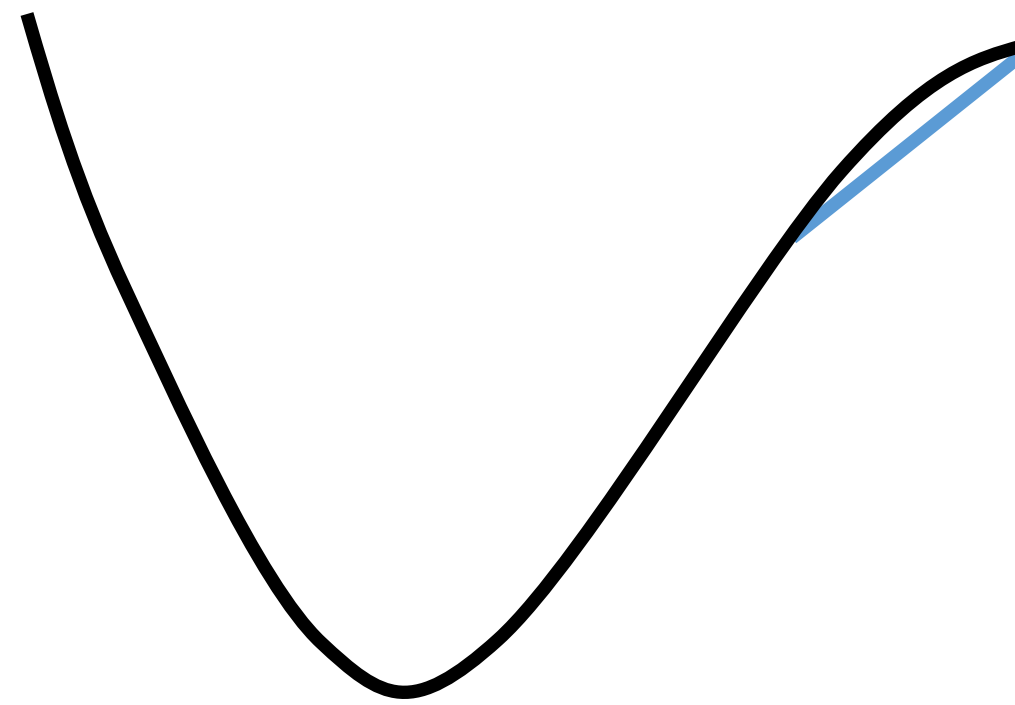
Is this a convex function?



- A. Yes
- B. No
- C. It depends

Convexity

Is this a convex function?



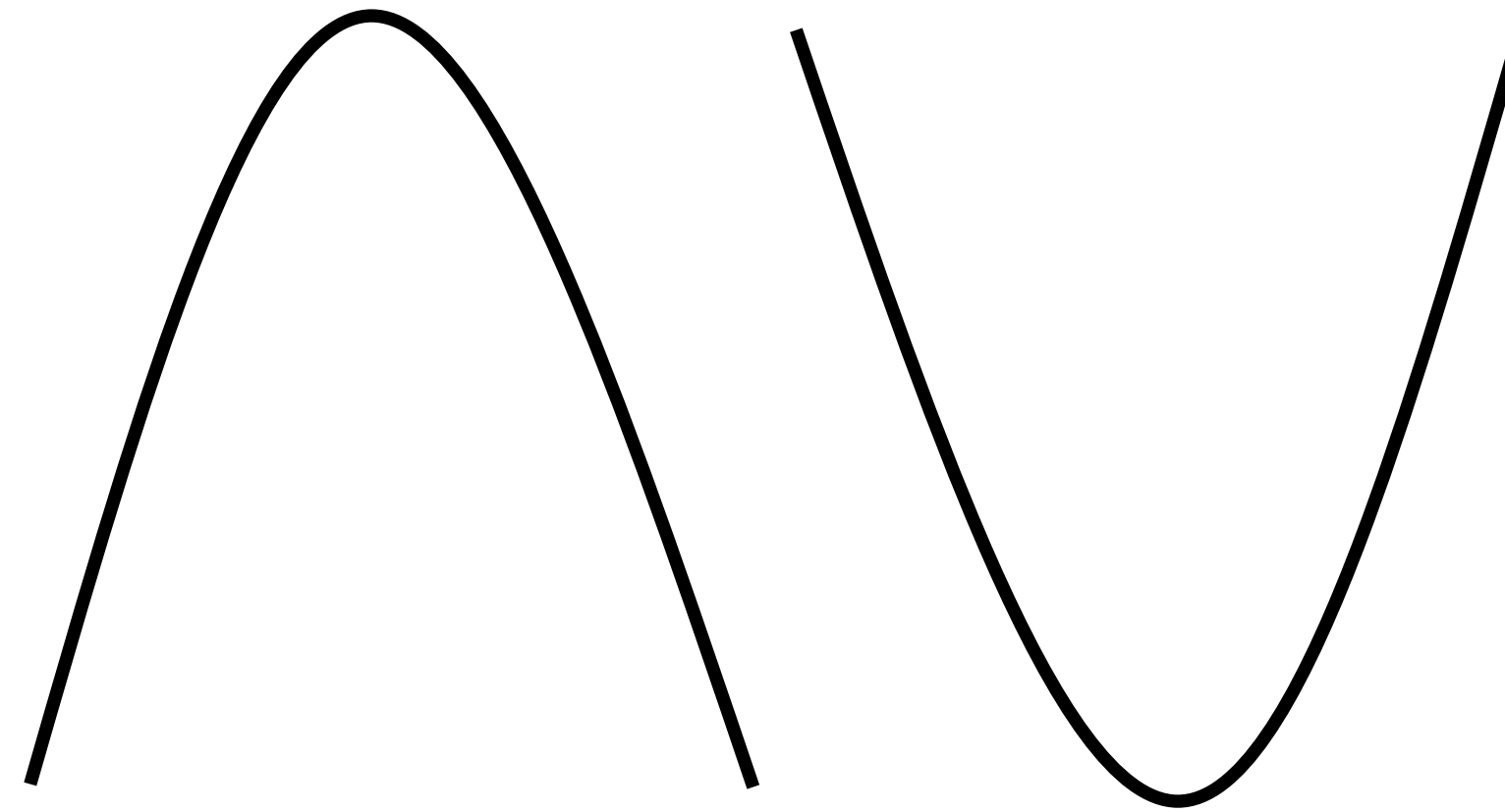
A. Yes

★ B. No

C. It depends

Convexity

Is this a convex function?



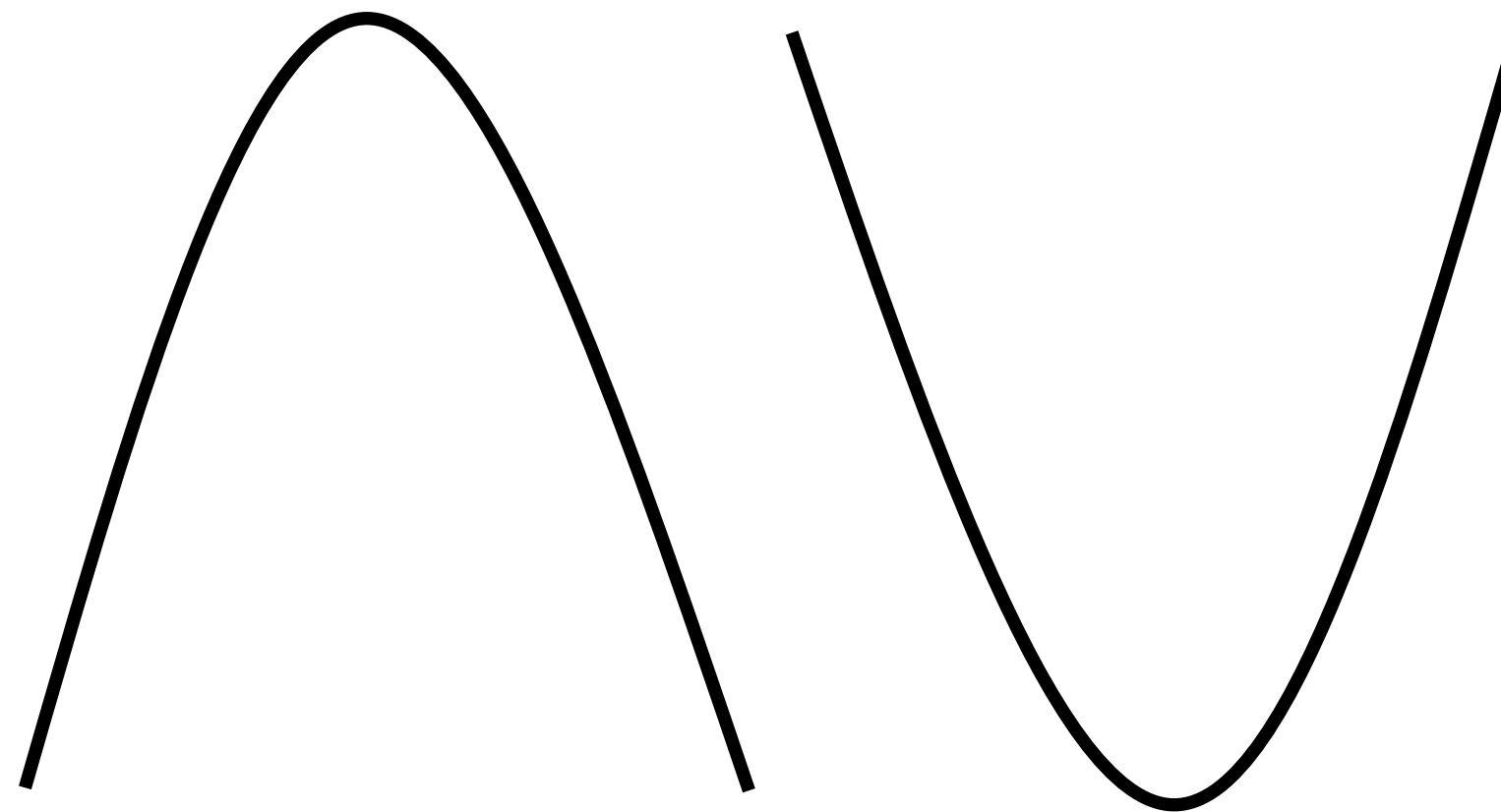
- A. Yes
- B. No
- C. It depends

It is concave! 😊

But for a concave function $L(w)$, $-L(w)$ is convex,
and vice versa.

Convexity

Is this a convex function?

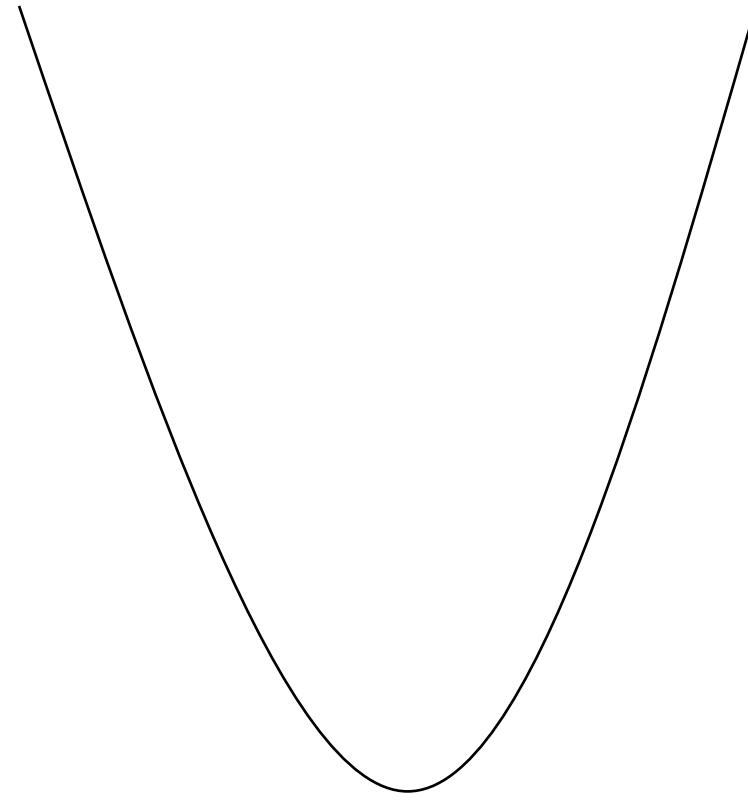


- A. Yes
- ★ B. No
- C. It depends

It is concave! 😊

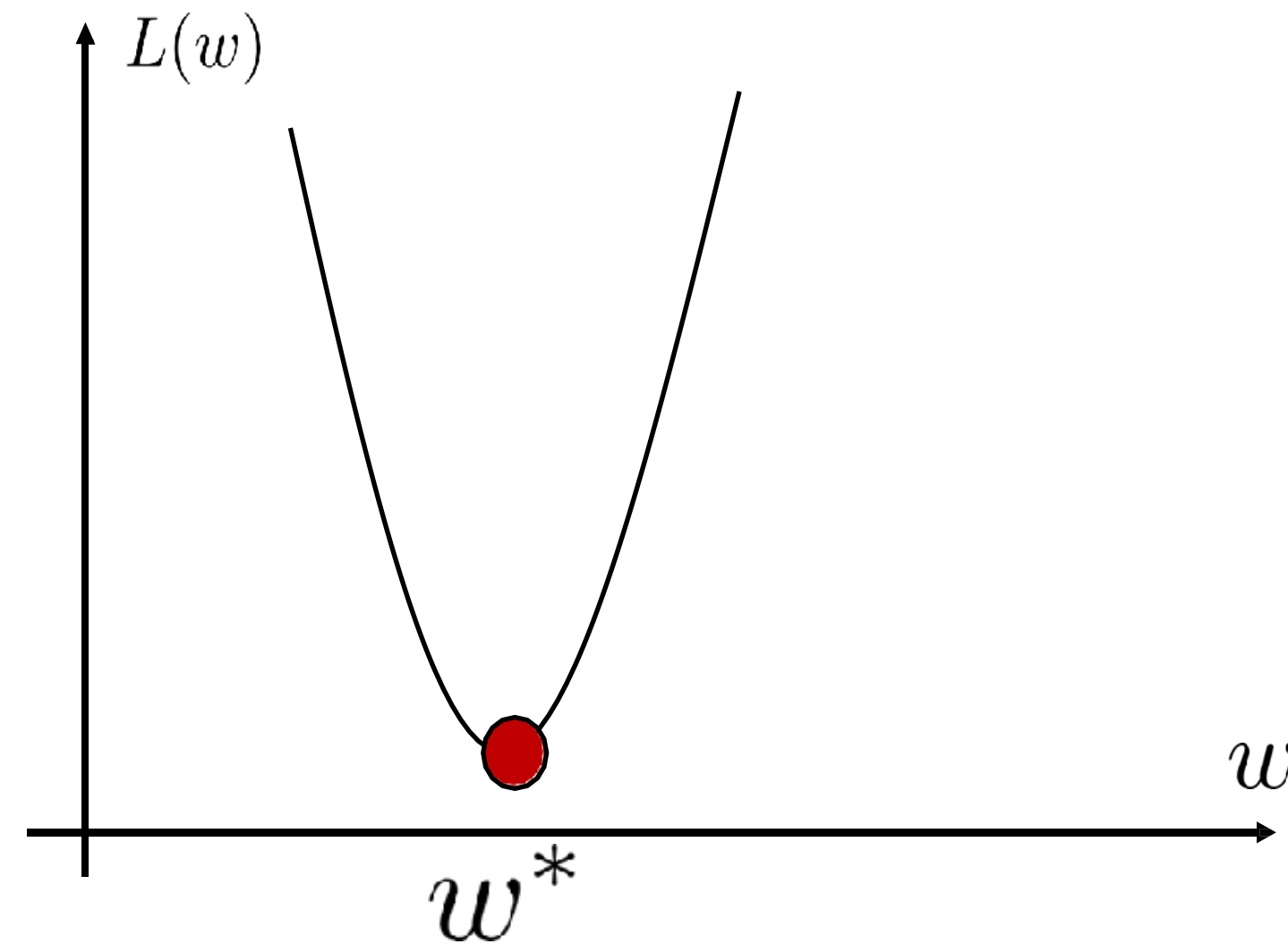
But for a concave function $L(w)$, $-L(w)$ is convex,
and vice versa.

Why study convex functions?



1. It has the globally optimal solution (to learn the best model).
2. Might have a closed form solution, if it is everywhere differentiable and has analytic form (learning accomplished in one-shot).
3. Gradient descent/ascent can be directly applied (iterative steps).

Convex functions that are differentiable

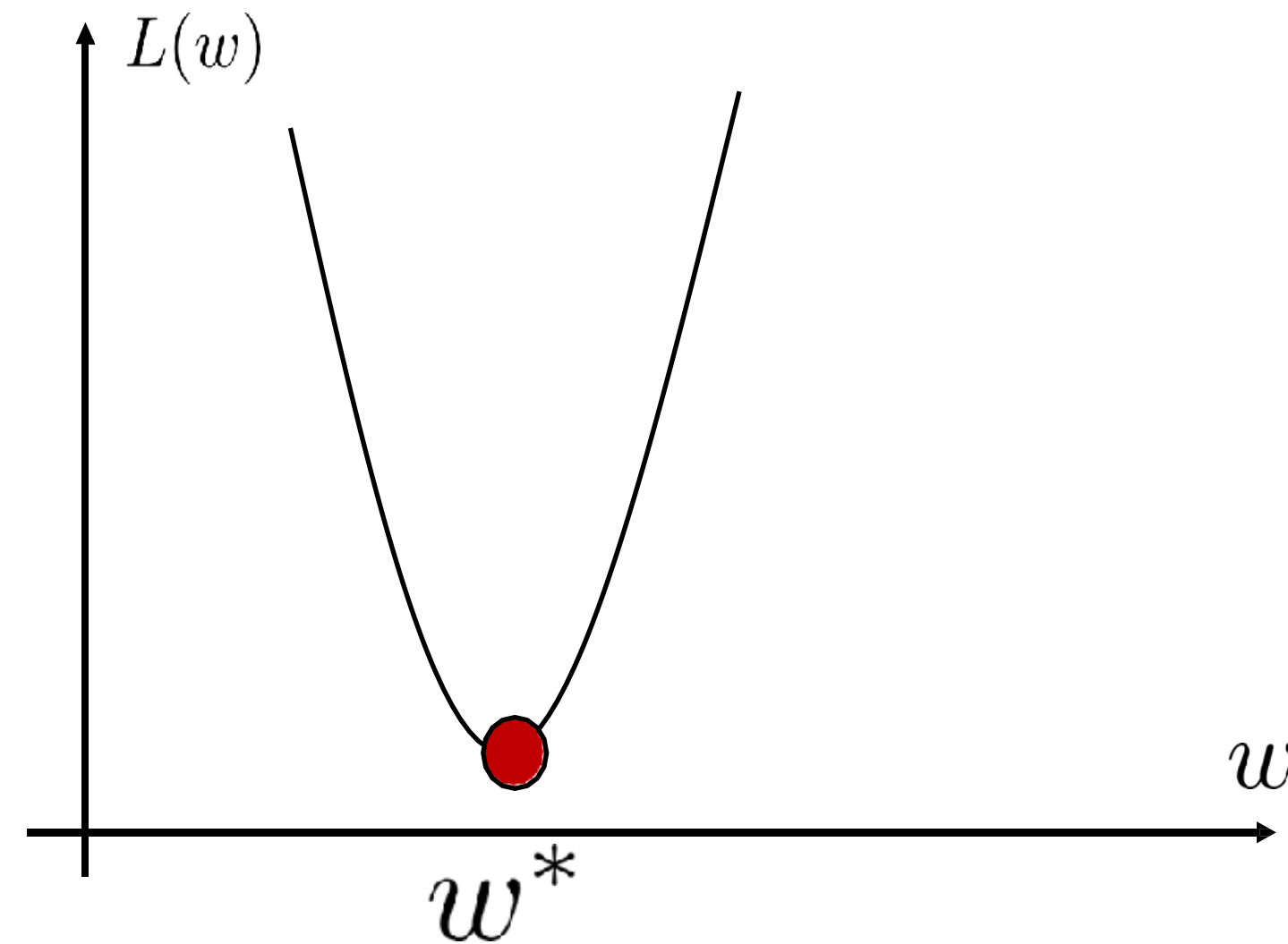


For a convex and differential function $\mathcal{L}(\mathbf{w})$,

its global optimum is achieved at \mathbf{w}^* . To find \mathbf{w}^* we solve for

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = 0$$

Convex functions that are differentiable

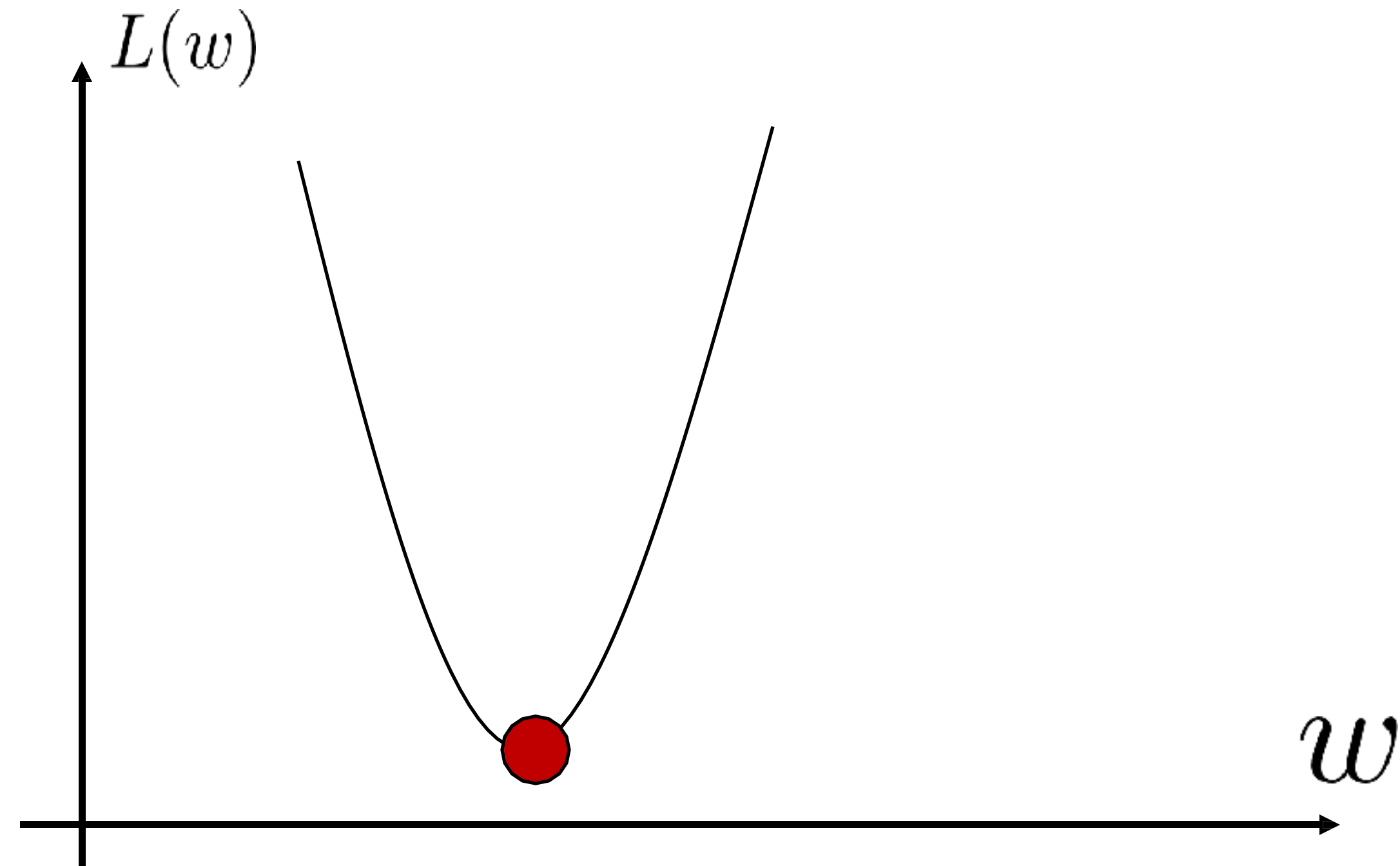


An analytical (closed form)
solution exists!

$w^* = q(X, Y)$ where X and Y consists of your training data
with the corresponding ground-truth labels.

You get your model in 1 shot, no iteration!

Convex function: differentiable



$$w^* = \arg \min_{\theta} L(w)$$

1. (Convex) Function

$$L(w) = (w - 3)^2 + 4$$

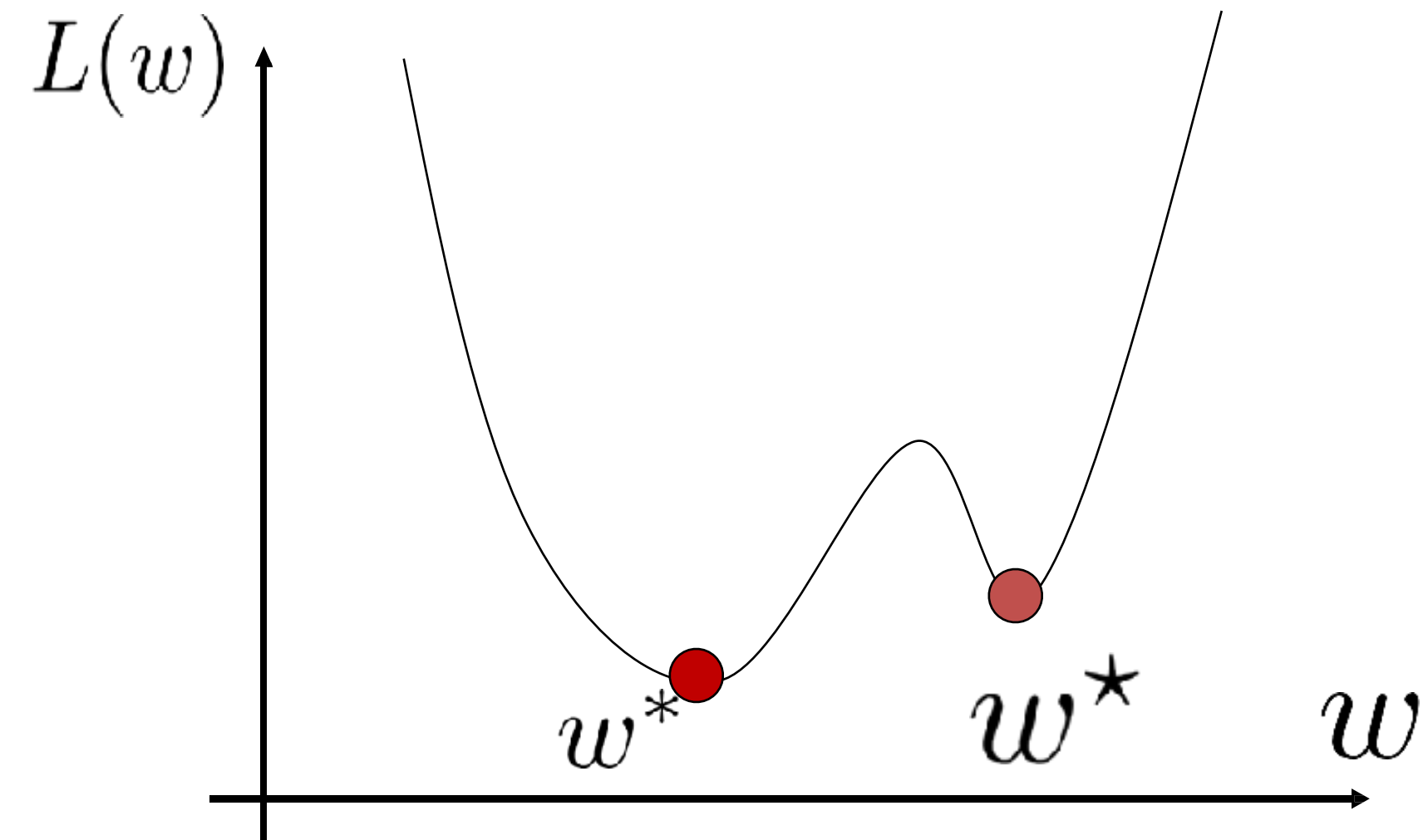
2. Set Derivative to 0

$$\frac{dL(w)}{dw} = 2 \times (w - 3) \quad \frac{dL(w)}{dw} = 0$$

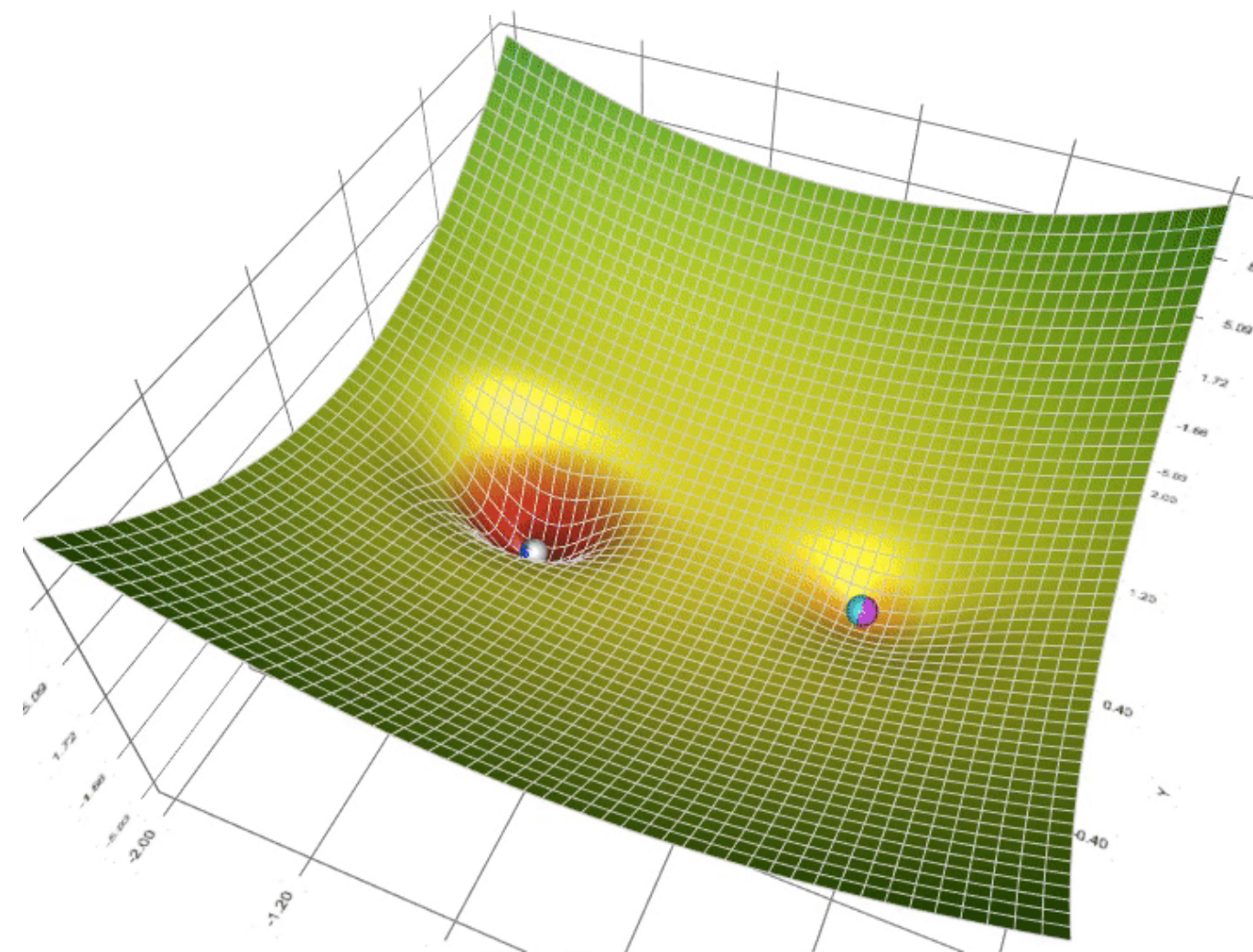
3. Solve for w

$$2 \times (w - 3) = 0 \rightarrow w = 3$$

Non-convex functions that are differentiable



Compute $\frac{\partial L(w)}{\partial w}$ to solve the problem iteratively through gradient descent



How to optimize (courtesy of wikipedia)

Computational optimization techniques [[edit](#)]

To solve problems, researchers may use [algorithms](#) that terminate in a finite number of steps, or [iterative methods](#) that converge to a solution (on some specified class of problems), or [heuristics](#) that may provide approximate solutions to some problems (although their iterates need not converge).

Optimization algorithms [[edit](#)]

See also: *List of optimization algorithms*

- [Simplex algorithm](#) of [George Dantzig](#), designed for [linear programming](#).
- Extensions of the simplex algorithm, designed for [quadratic programming](#) and for [linear-fractional programming](#).
- Variants of the simplex algorithm that are especially suited for [network optimization](#).
- [Combinatorial algorithms](#)
- [Quantum optimization algorithms](#)

Iterative methods [[edit](#)]

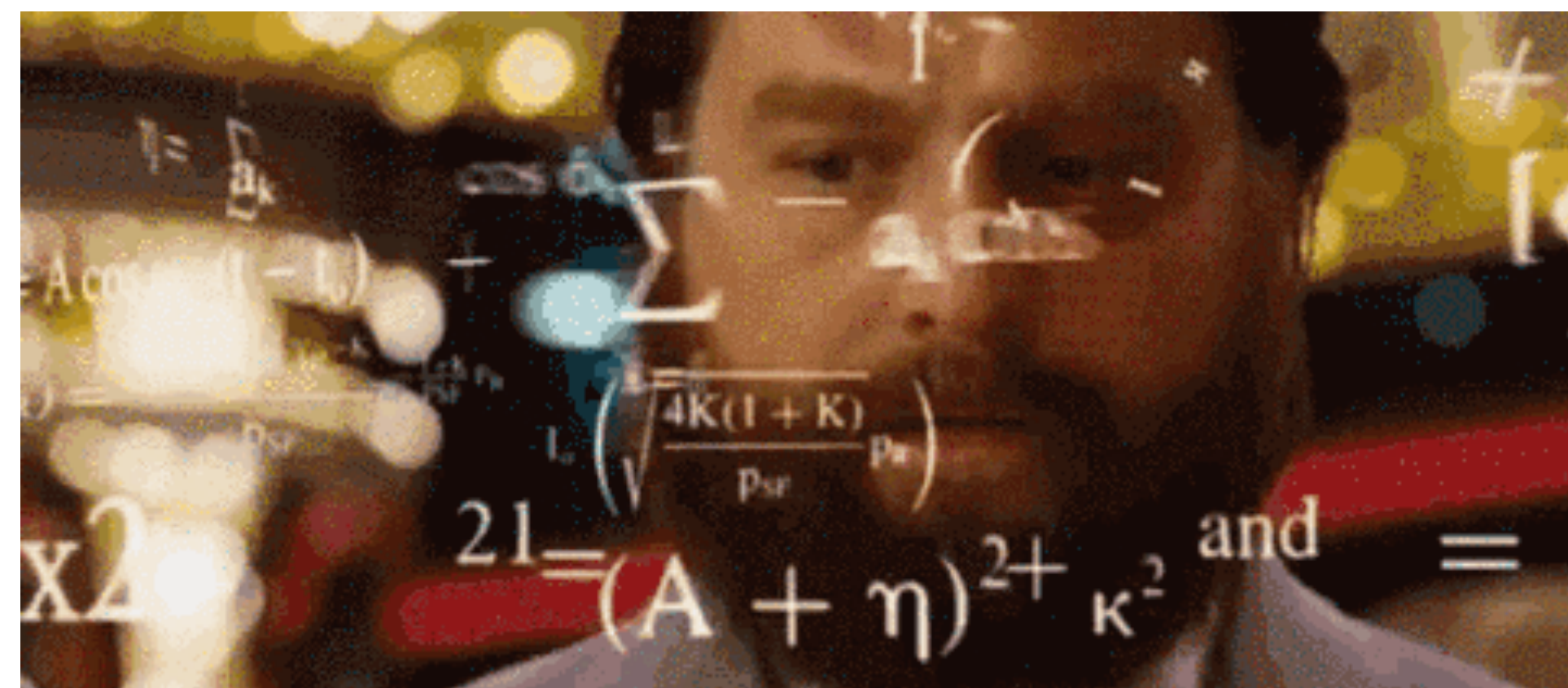
Main article: *[Iterative method](#)*

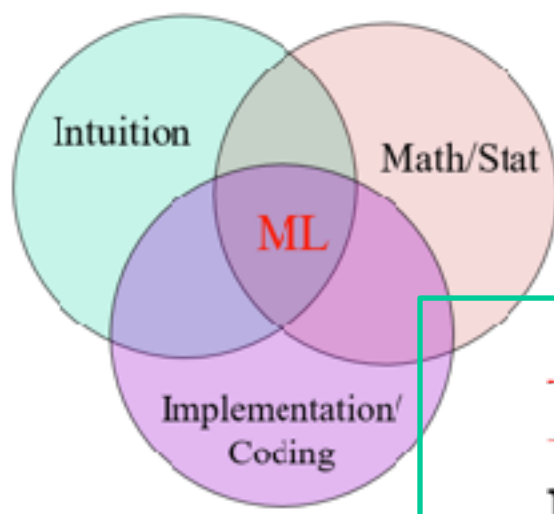
See also: *[Newton's method in optimization](#), [Quasi-Newton method](#), [Finite difference](#), [Approximation theory](#), and [Numerical analysis](#)*

The [iterative methods](#) used to solve problems of [nonlinear programming](#) differ according to whether they [evaluate Hessians](#), gradients, or only function values. While evaluating Hessians (H) and gradients (G) improves the rate of convergence, for functions for which these quantities exist and vary sufficiently smoothly, such evaluations increase the [computational complexity](#) (or computational cost) of each iteration. In some cases, the computational complexity may be excessively high.

One major criterion for optimizers is just the number of required function evaluations as this often is already a large computational effort, usually much more effort than within the optimizer itself, which mainly has to operate over the N variables. The derivatives provide detailed information for such optimizers, but are even harder to calculate, e.g. approximating the gradient takes at least N+1 function evaluations. For approximations of the 2nd derivatives (collected in the Hessian matrix), the number of function evaluations is in the order of N². Newton's method requires the 2nd order derivatives, so for each iteration, the number of function calls is in the order of N², but for a simpler pure gradient optimizer it is only N. However, gradient optimizers need usually more iterations than Newton's algorithm. Which one is best with respect to the number of function calls depends on the problem itself.

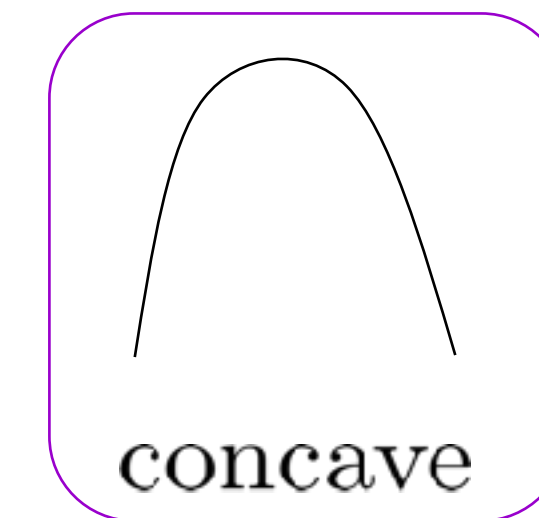
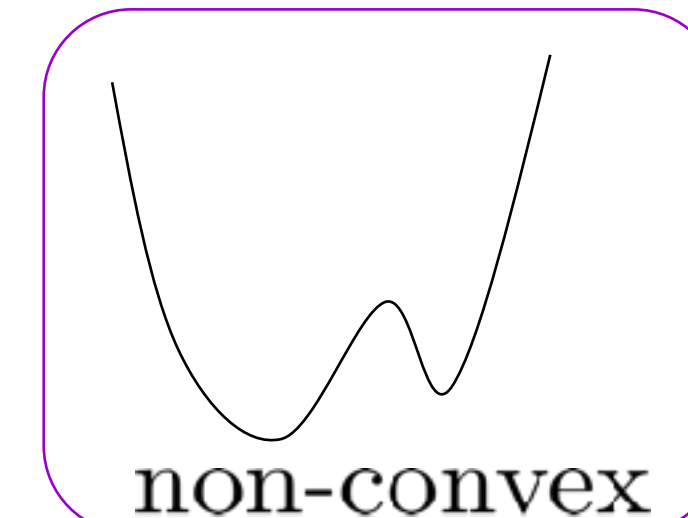
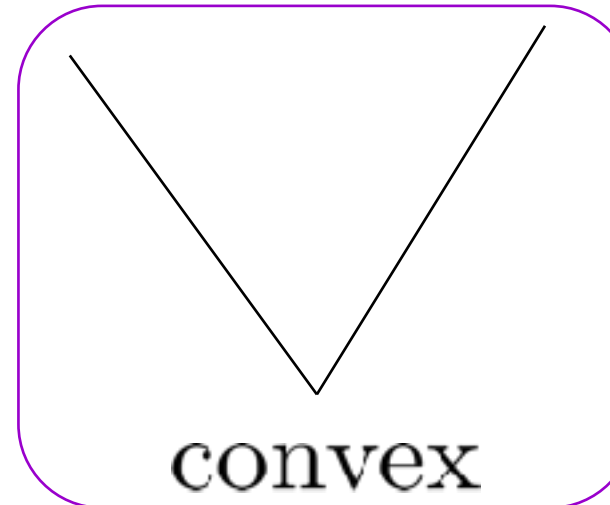
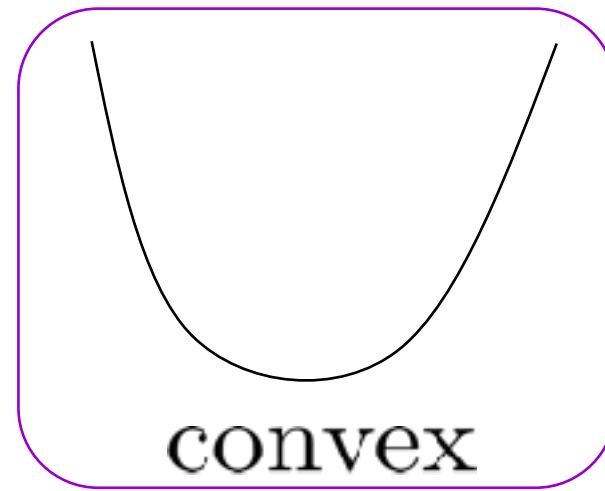
- Methods that evaluate Hessians (or approximate Hessians, using [finite differences](#)):
 - [Newton's method](#)
 - [Sequential quadratic programming](#): A Newton-based method for small-medium scale *constrained* problems. Some versions can handle large-dimensional problems.
 - [Interior point methods](#): This is a large class of methods for constrained optimization. Some interior-point methods use only (sub)gradient information and others of which require the evaluation of Hessians.
- Methods that evaluate gradients, or approximate gradients in some way (or even subgradients):
 - [Coordinate descent](#) methods: Algorithms which update a single coordinate in each iteration
 - [Conjugate gradient methods](#): [Iterative methods](#) for large problems. (In theory, these methods terminate in a finite number of steps with quadratic objective functions, but this finite termination is not observed in practice on finite-precision computers.)
 - [Gradient descent](#) (alternatively, "steepest descent" or "steepest ascent"): A (slow) method of historical and theoretical interest, which has had renewed interest for finding approximate solutions of enormous problems.
 - [Subgradient methods](#) - An iterative method for large [locally Lipschitz functions](#) using [generalized gradients](#). Following Boris T. Polyak, subgradient-projection methods are similar to conjugate-gradient methods.
 - Bundle method of descent: An iterative method for small-medium-sized problems with locally Lipschitz functions, particularly for [convex minimization](#) problems. (Similar to conjugate gradient methods)
 - [Ellipsoid method](#): An iterative method for small problems with [quasiconvex](#) objective functions and of great theoretical interest, particularly in establishing the polynomial time complexity of some combinatorial optimization problems. It has similarities with [Quasi-Newton methods](#).





Recap: Convexity

Intuition: Understanding the convexity of the estimation functions allows us to better design the learning algorithms and allows us to judge the quality (**global vs. local optimal**) of the learned models.



Math:

$$\forall w_0, w_1, a \in [0, 1]$$

$$aL(w_0) + (1 - a)L(w_1) \geq L(aw_0 + (1 - a)w_1)$$

or

Alternatively (for differentiable function):

$$L(w_1) \geq L(w_0) + \langle \nabla L(w_0), w_1 - w_0 \rangle$$