Autodiff

The algorithm that will upend the world (maybe)

Nanyang Technological University

Lim Soon Wei, Daniel

05 March 2025

Norming

(verb) forming shared "norms": expectations, styles, comfort

Questions are welcome!

Ask directly in the chat, raise a "hand", or just unmute.

Enjoy the story

Don't worry about taking notes; you can download these slides and demo code later.

No pressure

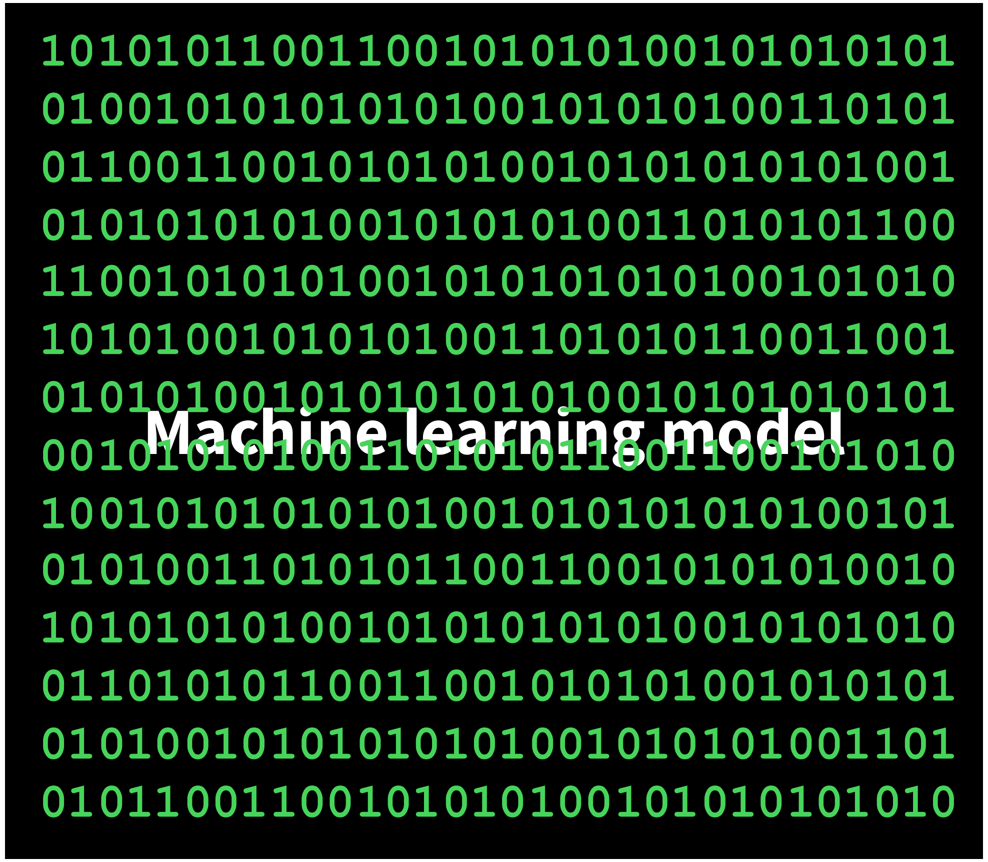
I won't call on you unless you raise a "hand"

Switch on your camera only if you are comfortable!

Have you used:

Gemini deepseek ChatGPT

Machine learning model


Machine learning model

Simplest model

$f(x,y) = a$

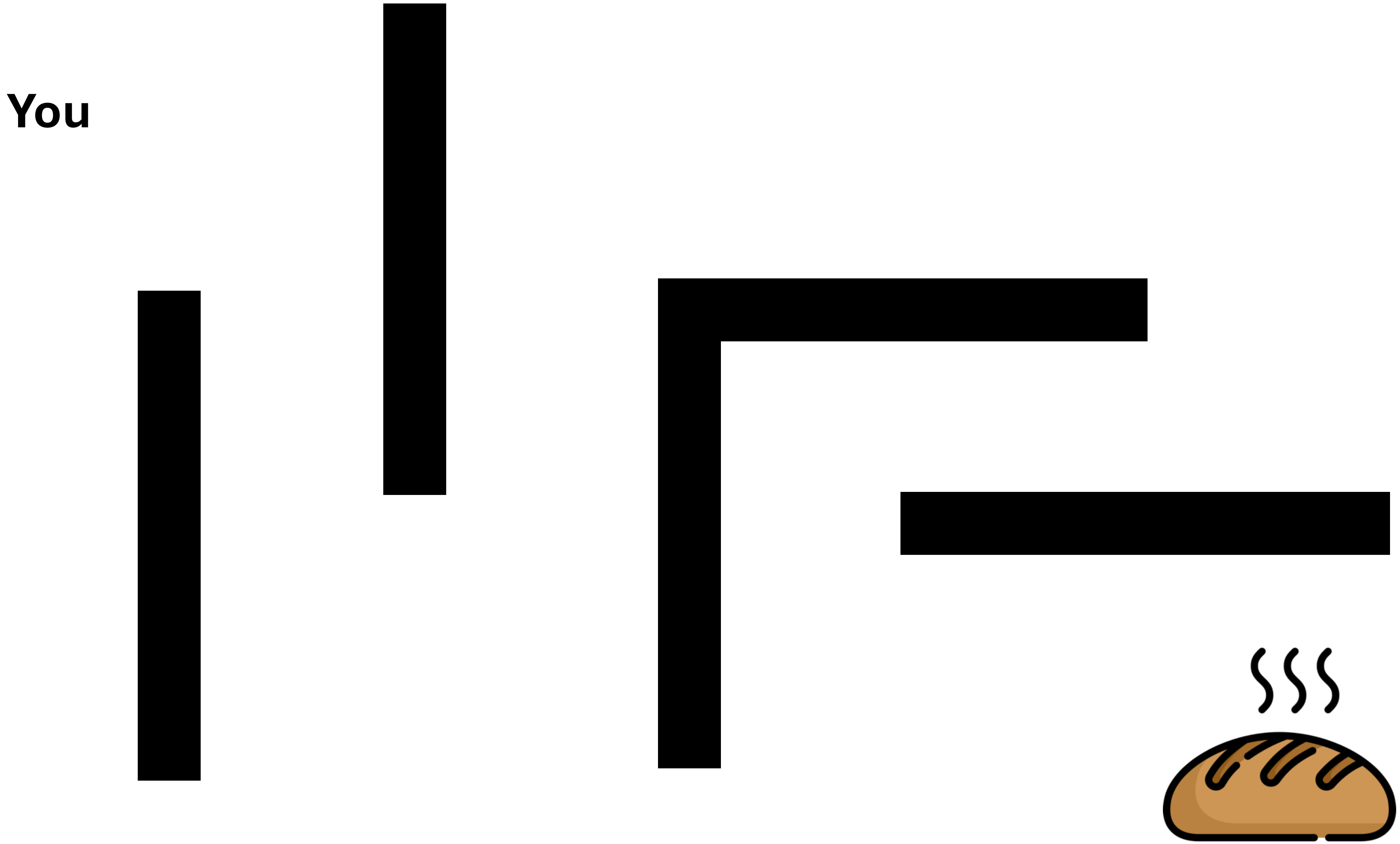
Input (x,y)

Output a

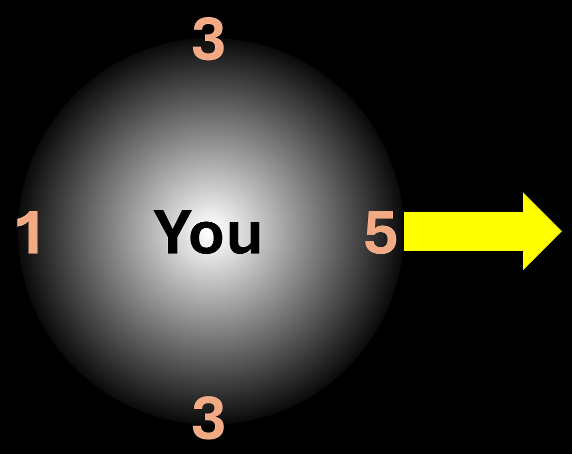
"Quality of model"

Optimization problem: How to pick (x,y) to minimize/maximize a?

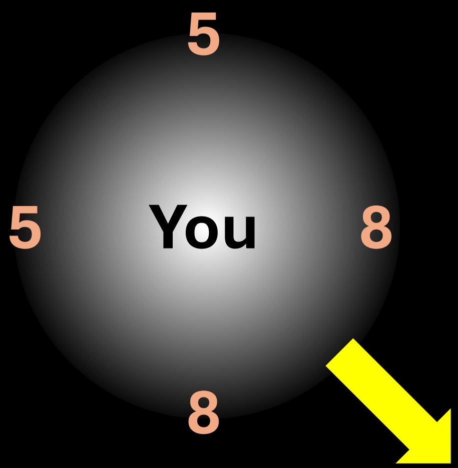
https://nowboarding.changiairport.com/explore-singapore/best-bakeries-in-singapore.html


You

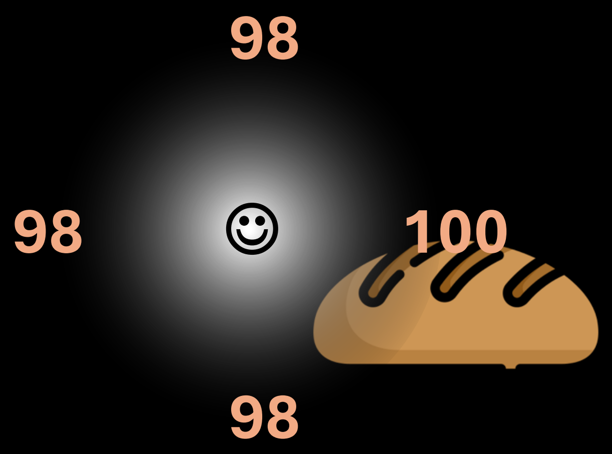
“Aroma” score



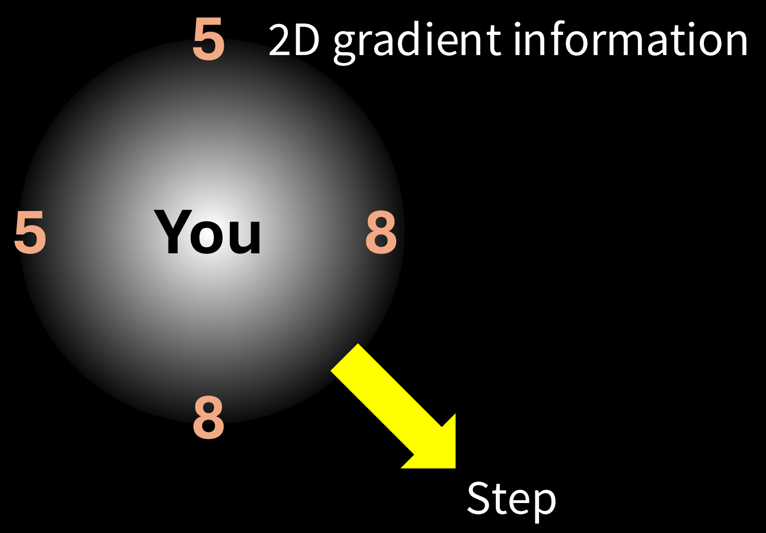
“Aroma” score

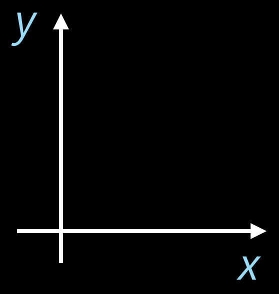


“Aroma” score



Gradient descent



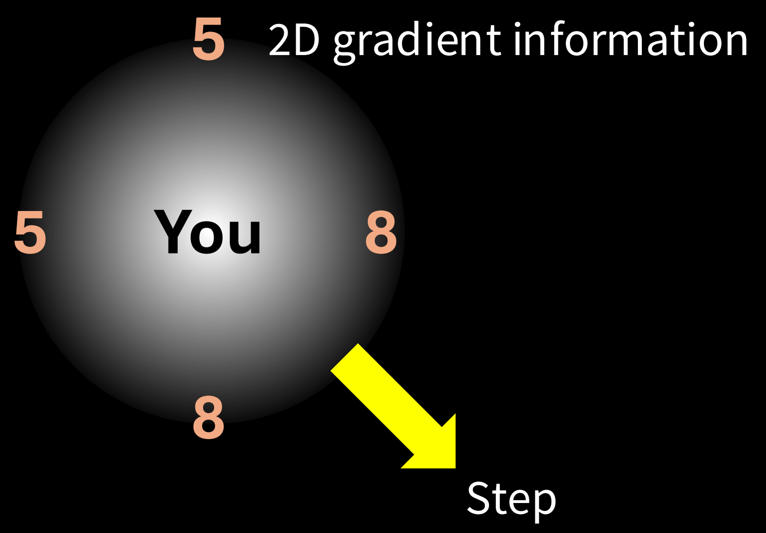


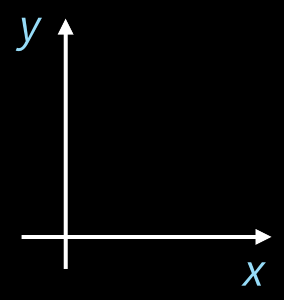
Parameter space

Algorithm

1. Compute gradient
2. Move a step in direction of greatest increase/decrease
3. Good enough? If not, go back to Step 1.

Gradient descent (mathematically)





Parameter space

Algorithm

1. Compute gradient $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$ at current position (x_0, y_0)
2. Move a step: $(x_0, y_0) \mapsto (x_0, y_0) - L\nabla f$, L = step size
3. Check for convergence (e.g., is ∇f small?) . If not, go back to Step 1.

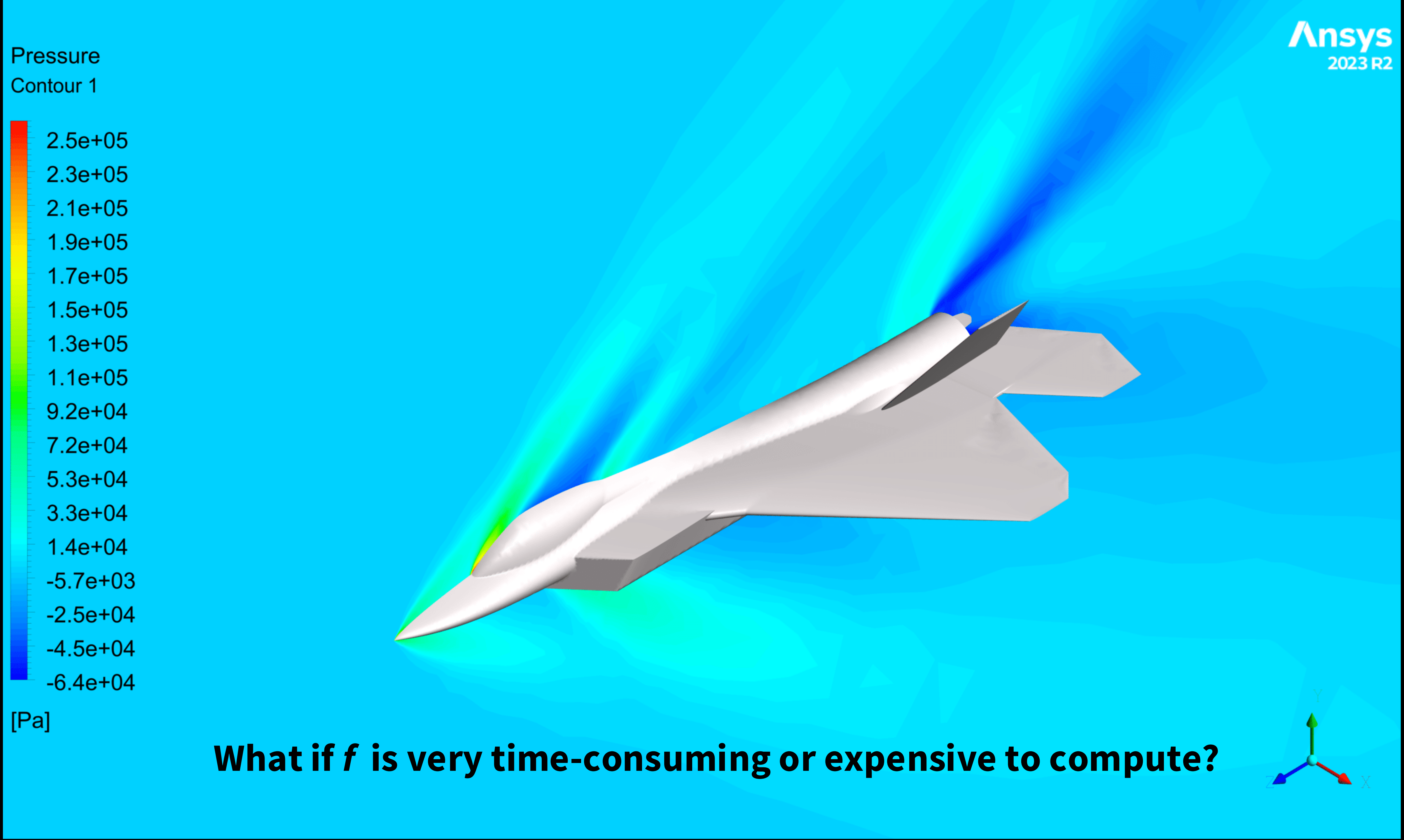
How many steps does it take in finite differences?

In 2D: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$

In 3D: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$

In nD: $\nabla f = \left(\underset{\underset{2}{\uparrow}}{\frac{\partial f}{\partial x_1}}, \underset{\underset{2}{\uparrow}}{\frac{\partial f}{\partial x_2}}, \cdots, \underset{\underset{2}{\uparrow}}{\frac{\partial f}{\partial x_n}}\right)$

Number of steps $\propto$ number of dimensions



What if *f* is very time-consuming or expensive to compute?

What if *n* is very very big?

27 billion (Gemma 2, 2025)

671 billion (R1, 2025)

Billions? Trillions?

Number of parameters *n*

We need a better method of calculating gradients.

Steps taken

In 2D: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$

In 3D: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$

In nD: $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$

2 2 2

Finite differences: Number of *f* calculations $\propto$ number of dimensions

Automatic differentiation will give the gradient **in at most 4x the number of forward pass (function f) operations[1]!**

[1] A. Griewank and A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd ed., SIAM, Philadelphia, 2008

Automatic/algorithmic differentiation

using the information *already present in your code* to calculate the gradient.

Not a new idea:
1952 Master's Thesis from John F. Nolan, Boston University: Analytical Differentiation on a digital computer

$$f(x_1, x_2, x_3) = (x_1 - x_2 + x_2 x_3) * (\cos x_2 x_3)$$

Find $\dfrac{\partial f}{\partial x_1}, \dfrac{\partial f}{\partial x_2}, \dfrac{\partial f}{\partial x_3}$

$$f(x_1, x_2, x_3) = \overbrace{(x_1 - x_2 + x_2x_3)}^{z_1} * \overbrace{(\cos x_2x_3)}^{z_2}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial z_1}{\partial x_1} z_2 + z_1 \frac{\partial z_2}{\partial x_1}$$

By product rule

$$z_1 = \underbrace{x_1 - x_2}_{y_1} + \underbrace{x_2x_3}_{y_2}$$

$$z_2 = \cos \underbrace{x_2x_3}_{y_2}$$

$$\frac{\partial z_1}{\partial x_1} = \frac{\partial y_1}{\partial x_1} + \frac{\partial y_2}{\partial x_1}$$

$$= 1 + 0$$

$$\frac{\partial z_2}{\partial x_1} = -\sin y_2 * \frac{\partial y_2}{\partial x_1}$$

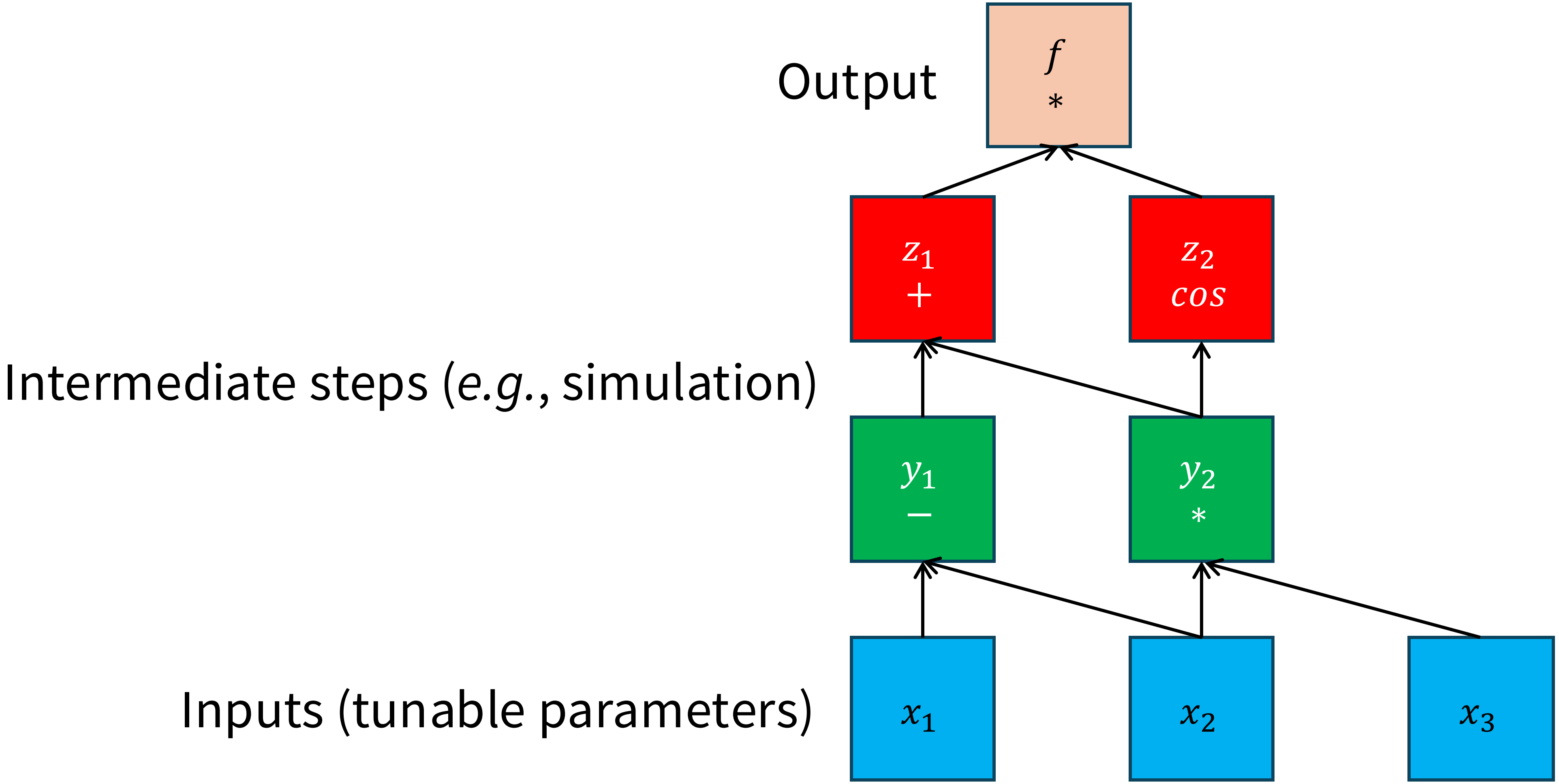
$$= 0$$

By chain rule

$$\frac{\partial f}{\partial x_1} = \cos x_2x_3$$

Equivalent computational tree

$$f(x_1, x_2, x_3) = \overbrace{(\underbrace{x_1 - x_2}_{y_1} + \underbrace{x_2 x_3}_{y_2})}^{z_1} * \overbrace{(\cos \underbrace{x_2 x_3}_{y_2})}^{z_2}$$



$$f(x_1, x_2, x_3) = \underbrace{(x_1 - x_2 + x_2 x_3)}^{z_1} * \underbrace{(\cos x_2 x_3)}^{z_2}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial z_1}{\partial x_1} z_2 + z_1 \frac{\partial z_2}{\partial x_1}$$

By product rule

$$z_1 = \underbrace{x_1 - x_2}_{y_1} + \underbrace{x_2 x_3}_{y_2}$$

$$z_2 = \cos \underbrace{x_2 x_3}_{y_2}$$

$$\frac{\partial z_1}{\partial x_1} = \frac{\partial y_1}{\partial x_1} + \frac{\partial y_2}{\partial x_1}$$
$$= 1 + 0$$

$$\frac{\partial z_2}{\partial x_1} = -\sin y_2 * \frac{\partial y_2}{\partial x_1}$$
$$= 0$$

By chain rule

$$\frac{\partial f}{\partial x_1} = \cos x_2 x_3$$

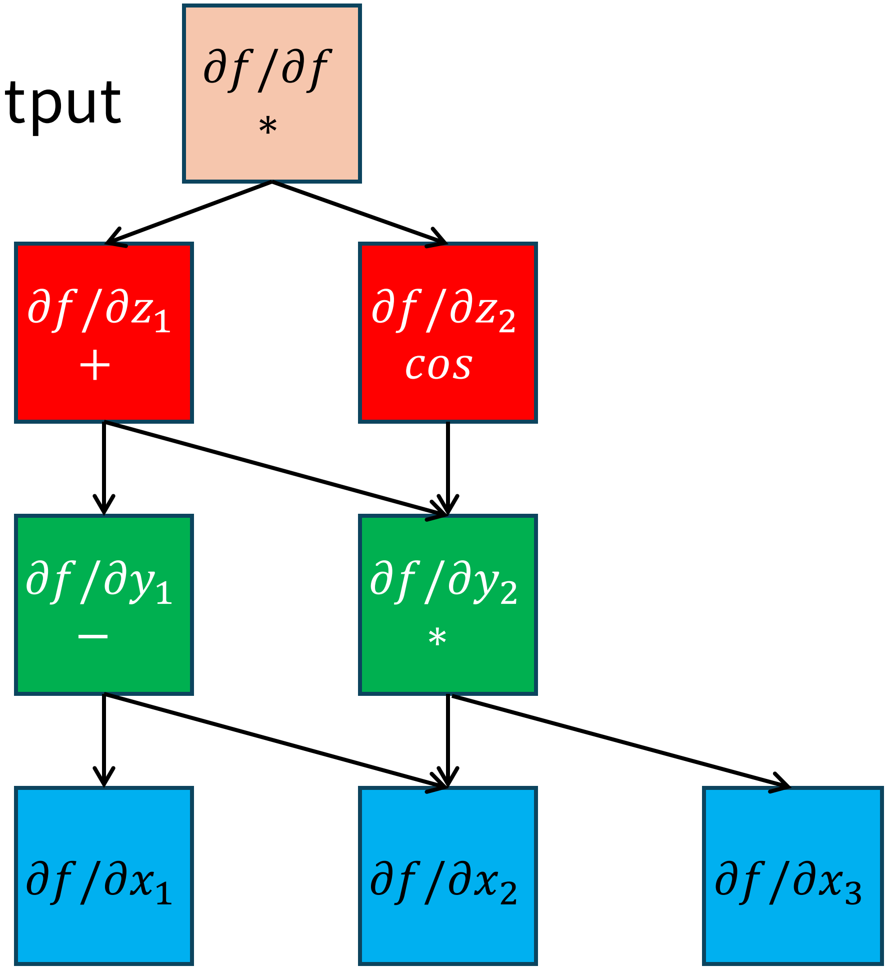
Adjoint computational tree

$$f(x_1, x_2, x_3) = \overbrace{(\underbrace{x_1 - x_2}_{y_1} + \underbrace{x_2 x_3}_{y_2})}^{z_1} * \overbrace{(\cos \underbrace{x_2 x_3}_{y_2})}^{z_2}$$

Output

Intermediate steps (*e.g.*, simulation)

Inputs (tunable parameters)



Key insight for autodiff

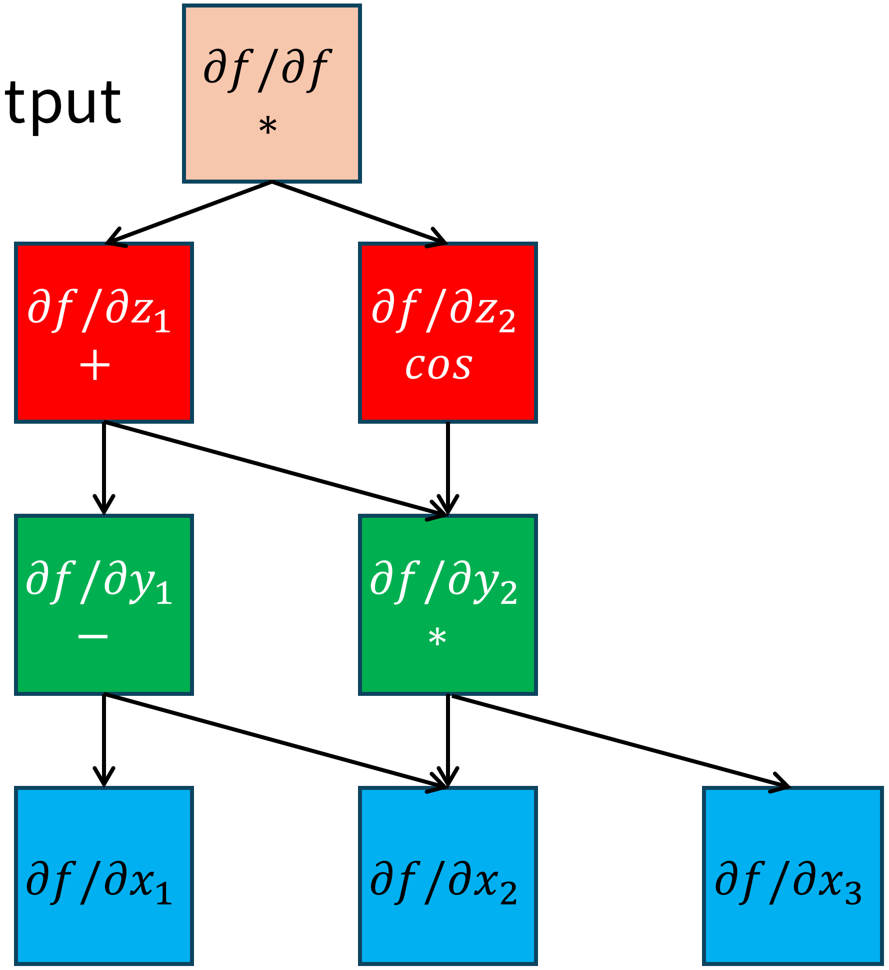
The sensitivity $\frac{\partial f}{\partial j}$ of every node j can be computed by tracing the computational tree backwards.

How does the output change by wiggling this value?

Output

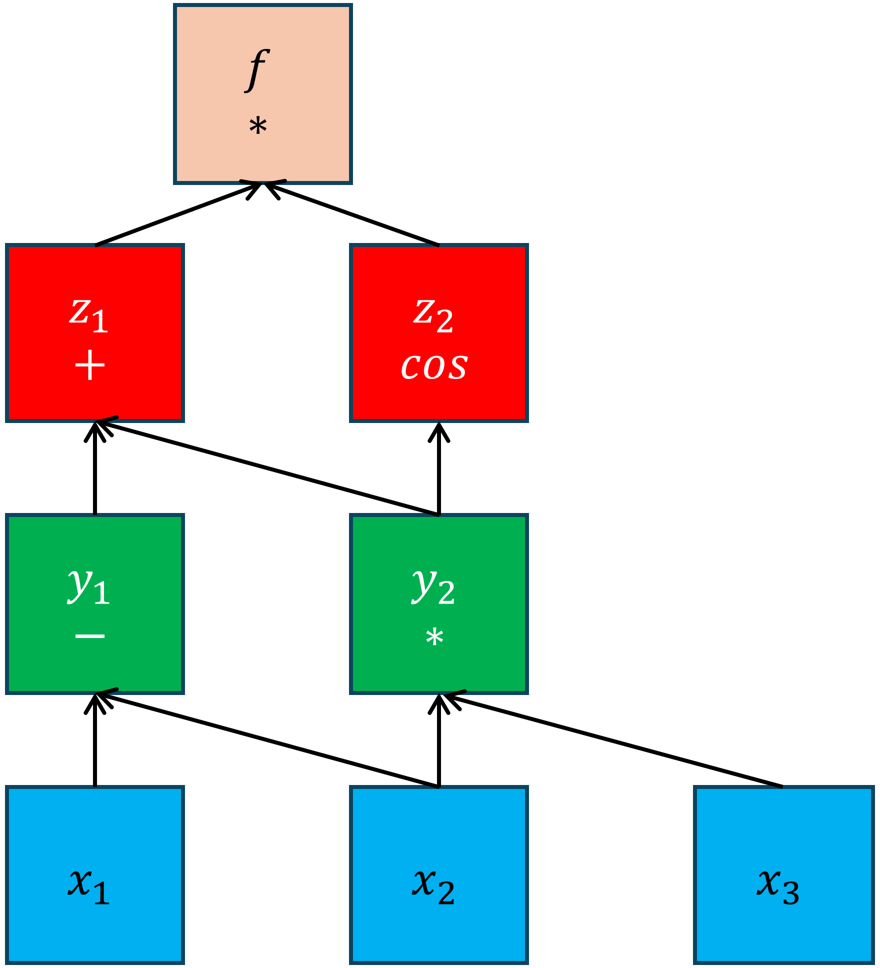
Intermediate steps (*e.g.*, simulation)

Inputs (tunable parameters)

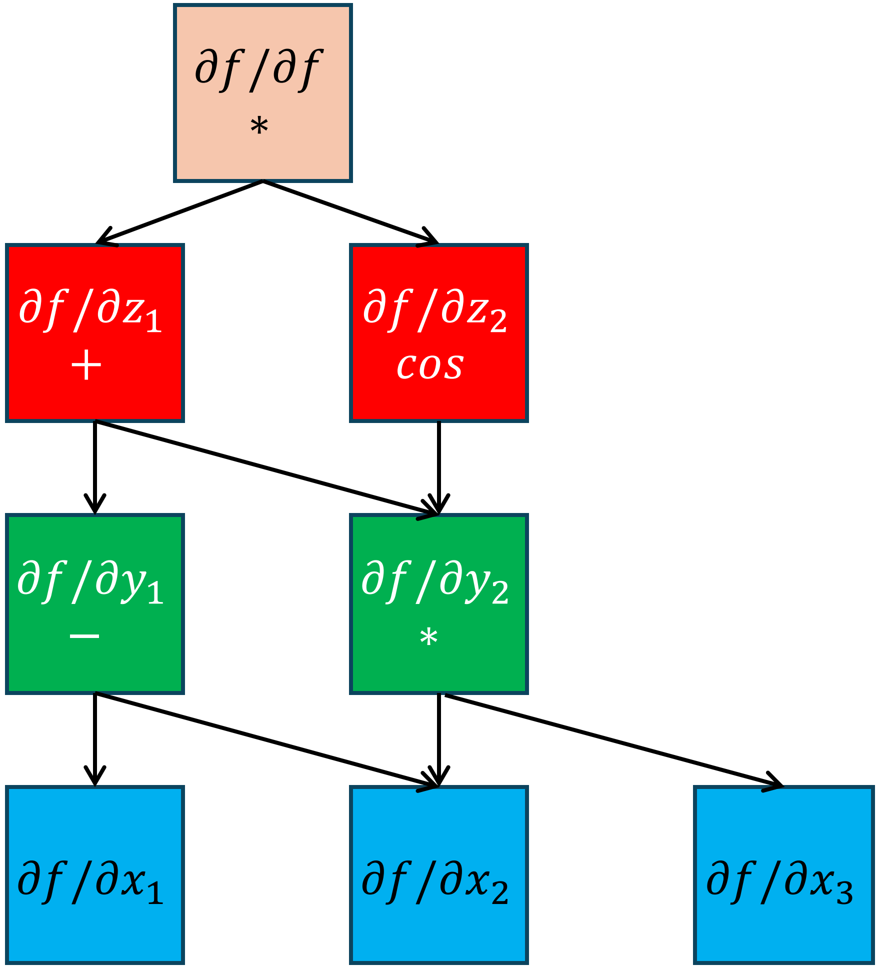


Autodiff protocol

1. Run forward pass. Store computational tree and intermediate values.



2. Run reverse pass. Use stored tree to compute sensitivities.



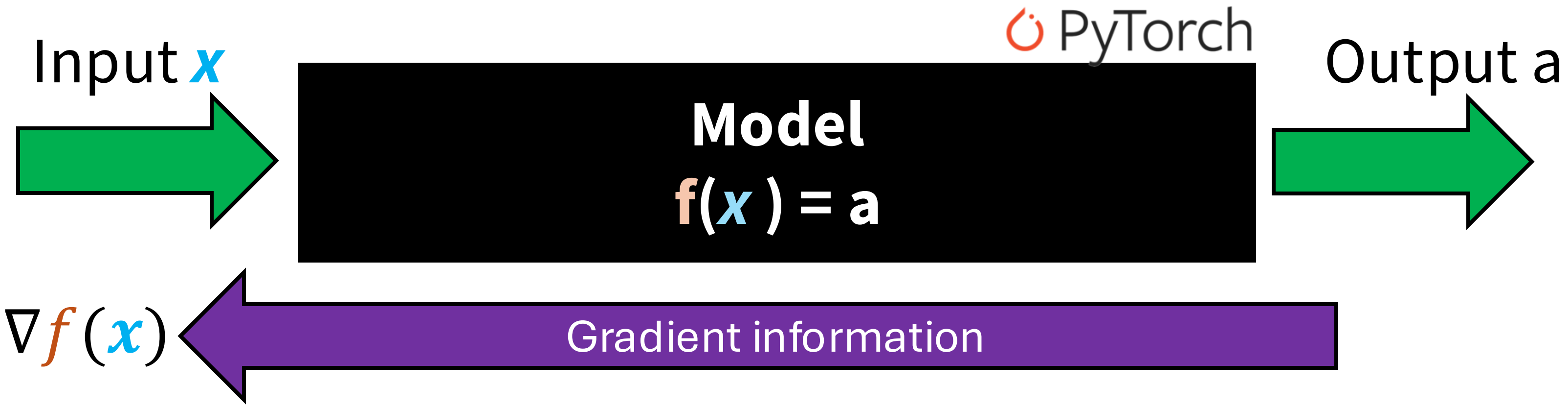
Do you need to know about computational trees to use Autodiff?

No (: Free frameworks build it for you automatically!

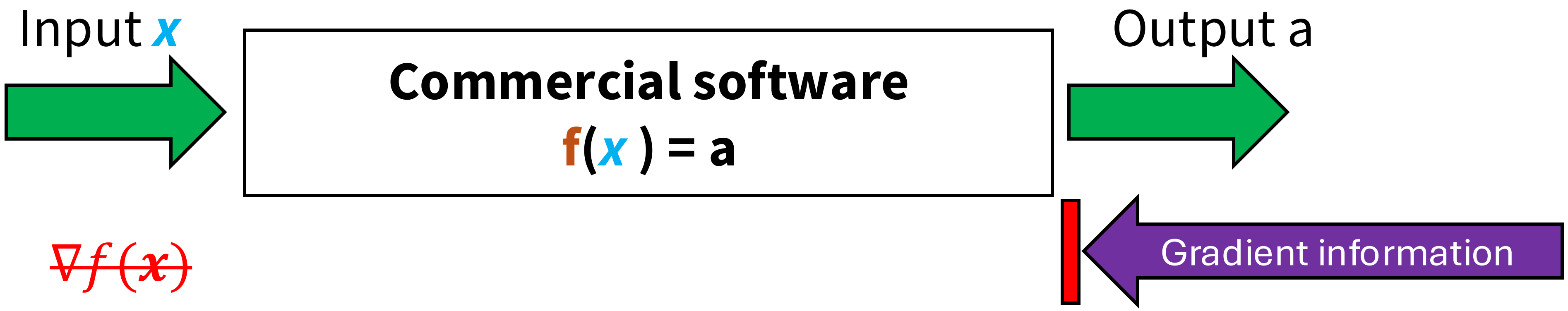
TensorFlow PyTorch

A small catch

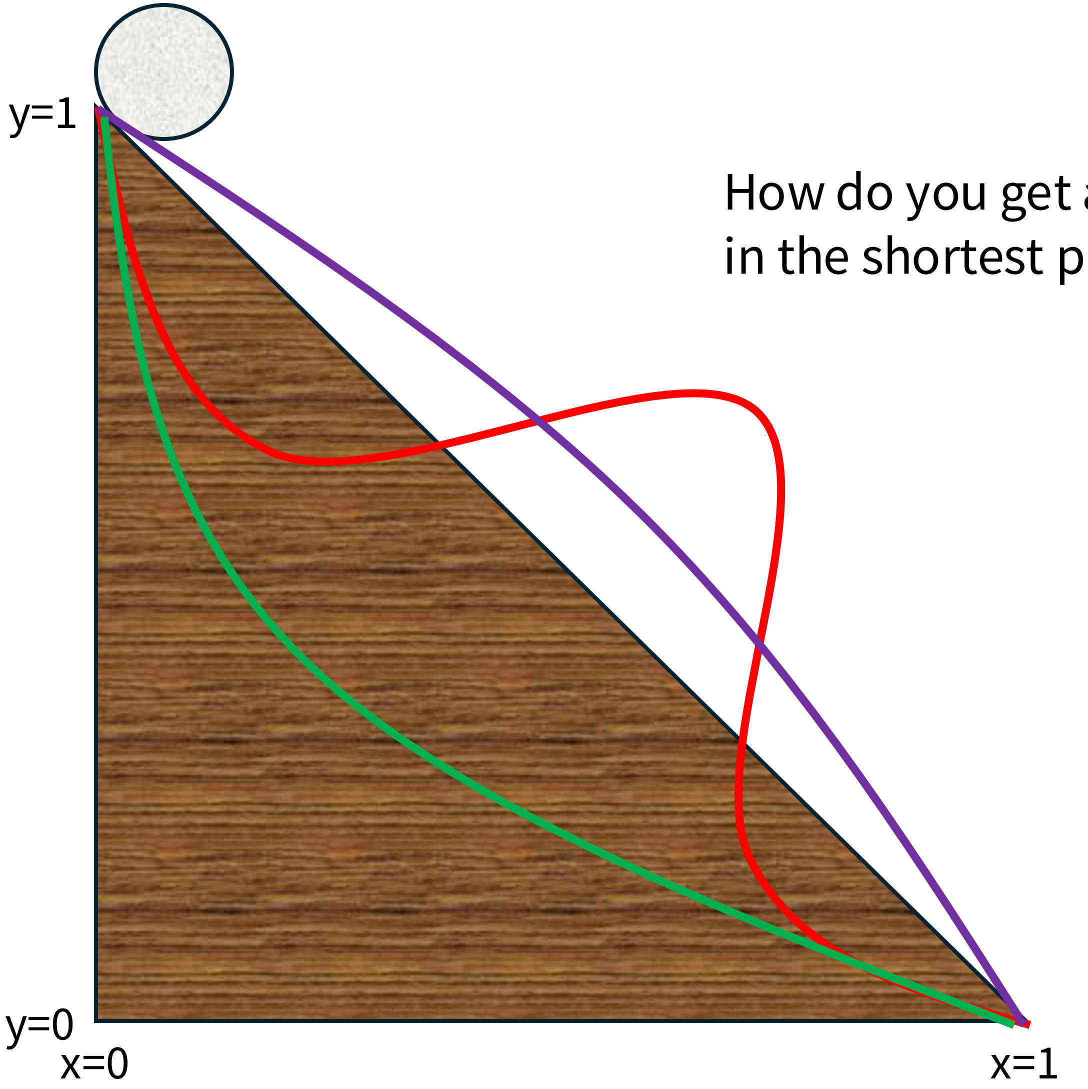
All calculations need to be performed on a *differentiable platform* for the gradients to be backpropagated.



Software lacking the source code are *not differentiable*.

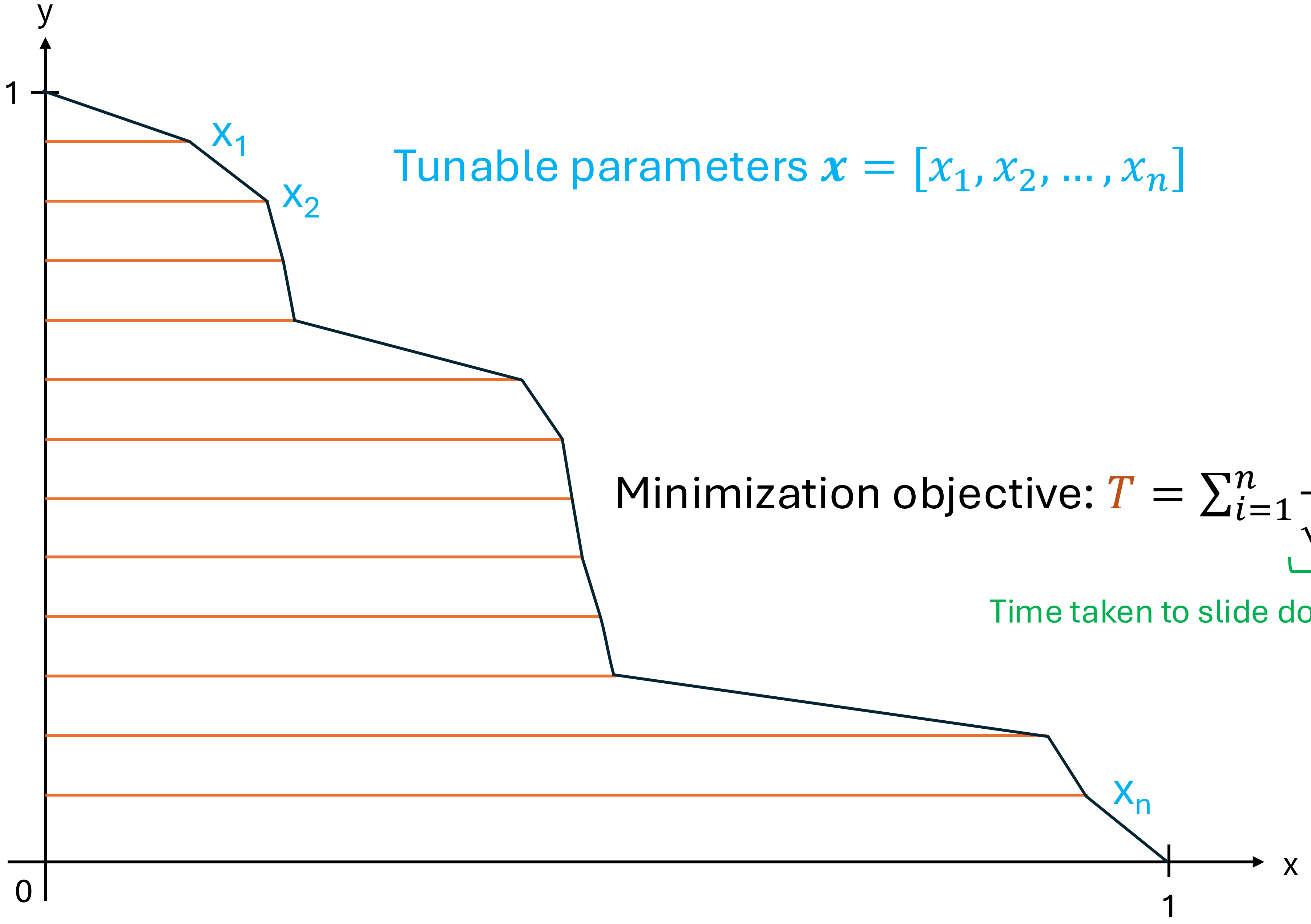


Let's solve a problem



How do you get a particle to slide to the end in the shortest possible time?

Optimization setup



Tunable parameters $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$

Minimization objective: $T = \underbrace{\sum_{i=1}^{n} \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\sqrt{2g(y_{start} - y)}}}_{\text{Time taken to slide down each section}}$

Recap

- The time-intensive step in optimization is often gradient calculation.
- Automatic differentiation enables efficient high-dimensional gradient calculation for any physical or mathematical system.
- To use automatic differentiation, all calculations must be on a *differentiable* platform.

Download these slides and demo code here:

https://danlimsw.com/coursenotes/