Backpropagation is “easy”!

Marco Zamboni, May 27, PyconIT

Logistics

- Materials available at:
  - Discord channel #slides-and-resources
  - Github: github.com/ZamboniMarco99/backpropagation-pycon
  - Colab notebook
- Ask questions!
  - Best if you interrupt me
  - Slido if you feel shy (Notice that I might answer them only at the end)

Who am I

- Student in MSc in Data Science @ ETH Zurich
- Data Scientist @ 42Matters
  - App intelligence company
- linkedin.com/in/marco-zamboni

Outline

- Recap on the optimization task for training neural networks
- Modeling functions as computational graphs
- Reverse mode automatic differentiation

- Everything implemented from scratch*

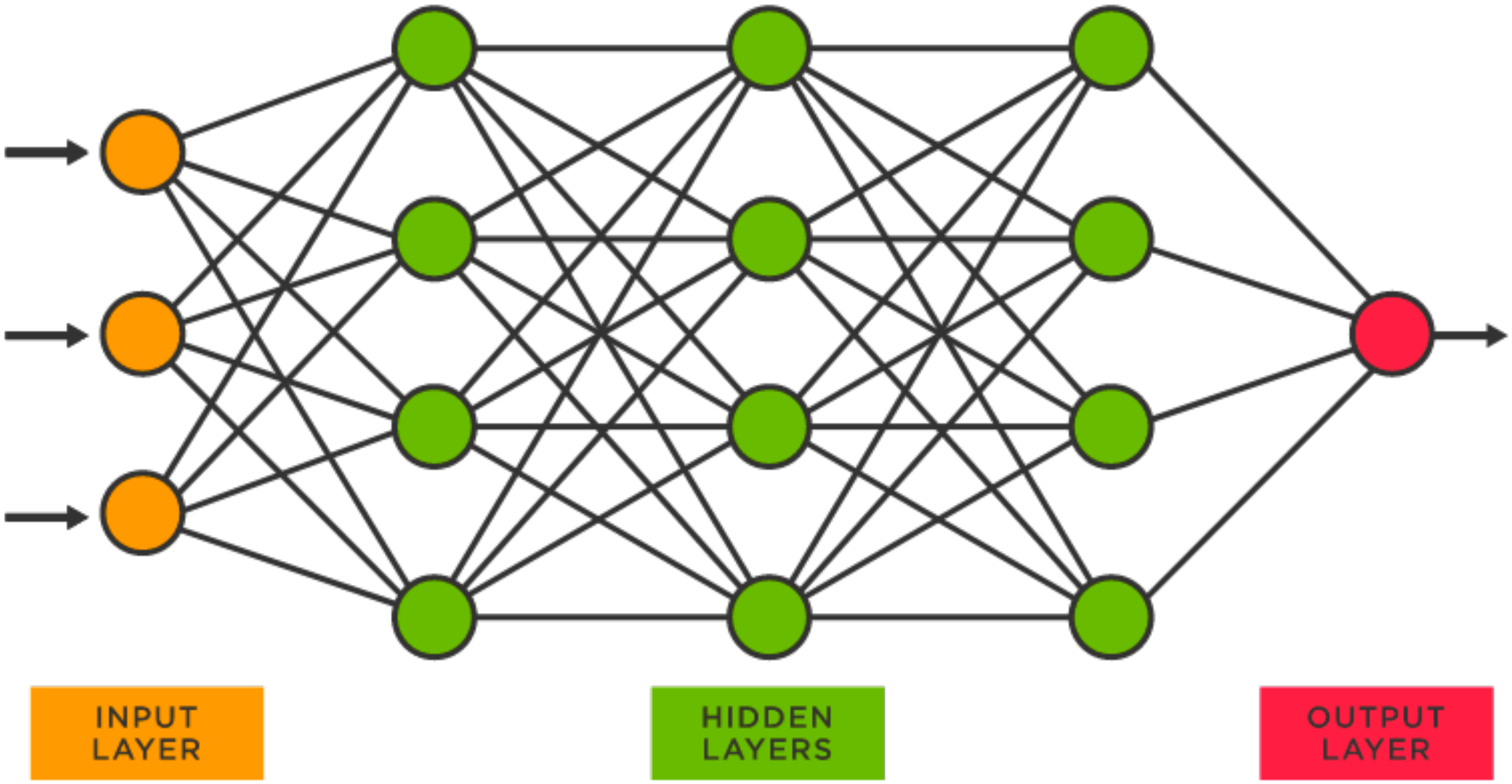
*We'll use a parser as an helper

Motivation & Preamble

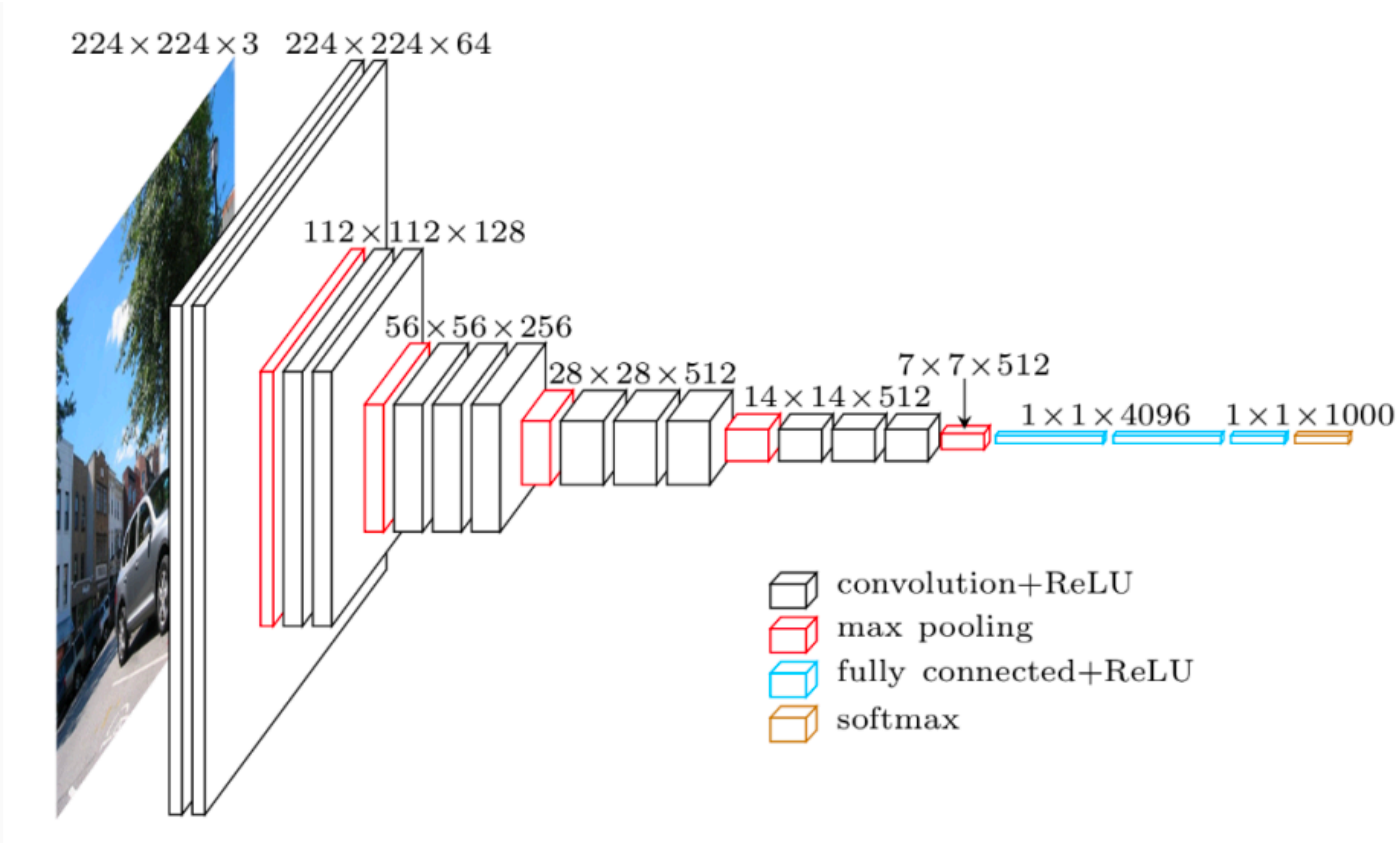
Why this workshop should be interesting

1. Most people are either
    - Scientist that know a lot of statistical properties
    - Engineers that just use the frameworks
2. Knowing what is under the hood makes you a better professional
3. The main focus will be “reverse mode automatic differentiation” which sounds scary, but…
4. …it’s not that complicated, anyone of us here can do it!

Neural Networks



Neural Networks

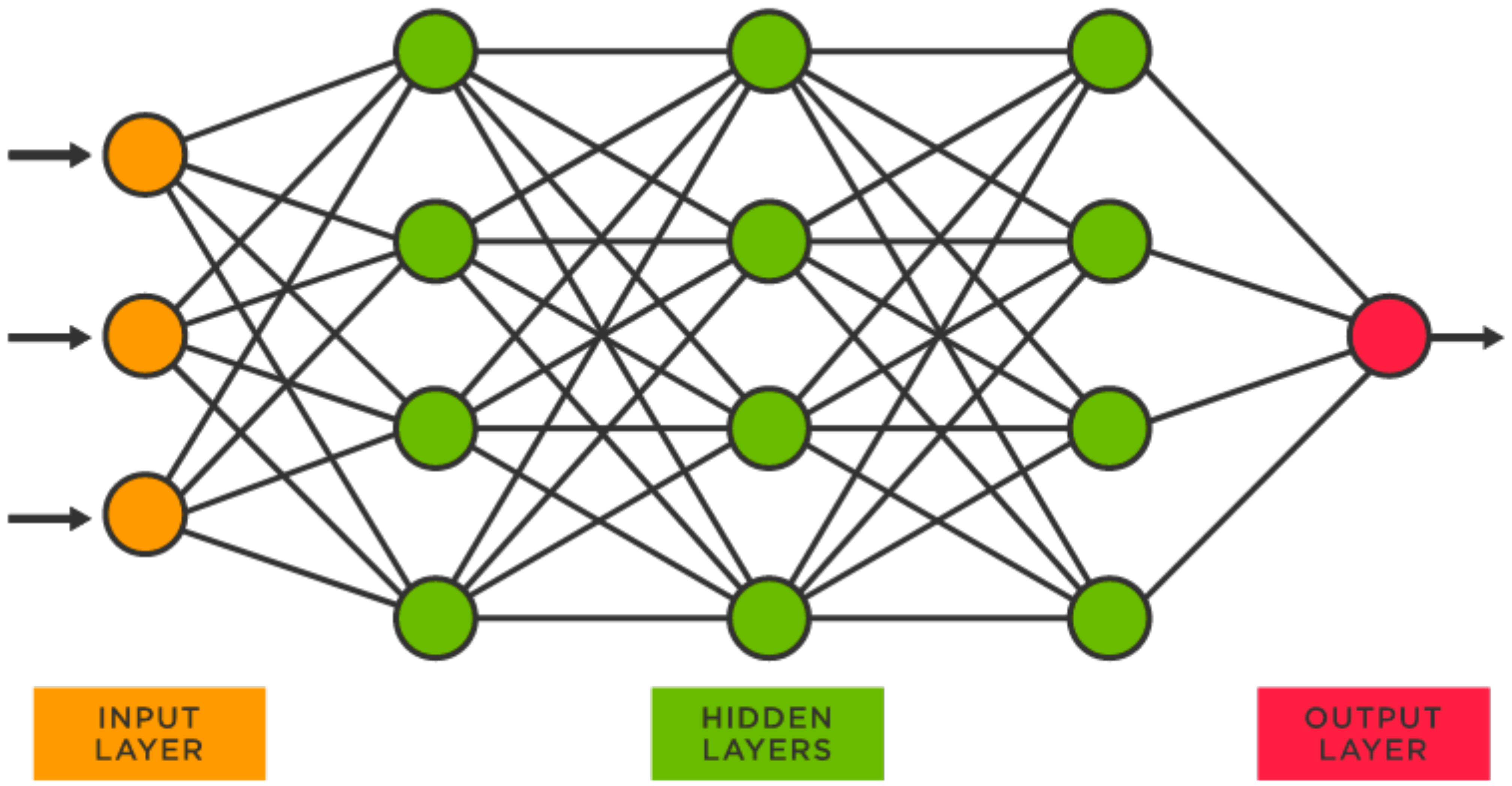


Neural Networks

Mathematical formulation

$$\ell_n \circ \ell_{n-1} \circ \ldots \circ \ell_2 \circ \ell_1$$

$$NN(w, x) =$$
$$softmax(relu(\ldots relu(w_2 relu(w_1 x + b_1) + b_2) \ldots + b_n))$$



$$\ell_i(w_i, x)[n] = \sum_{m=0}^{size(x)} x(m) w_i(n - m)$$

Training a NN
Optimization

$$w^* = min_w \; loss(y, NN(w, x))$$

- Analytical approach
- Linear programming (simplex)
- Gradient methods
- Hessian methods

Training a NN
Optimization

$$w* = min_w \; loss(y, NN(w, x))$$

- ~~Analytical approach~~
- ~~Linear programming (simplex)~~
- **Gradient methods**
  - Gradient Descent
- Hessian methods
  - Newton Method

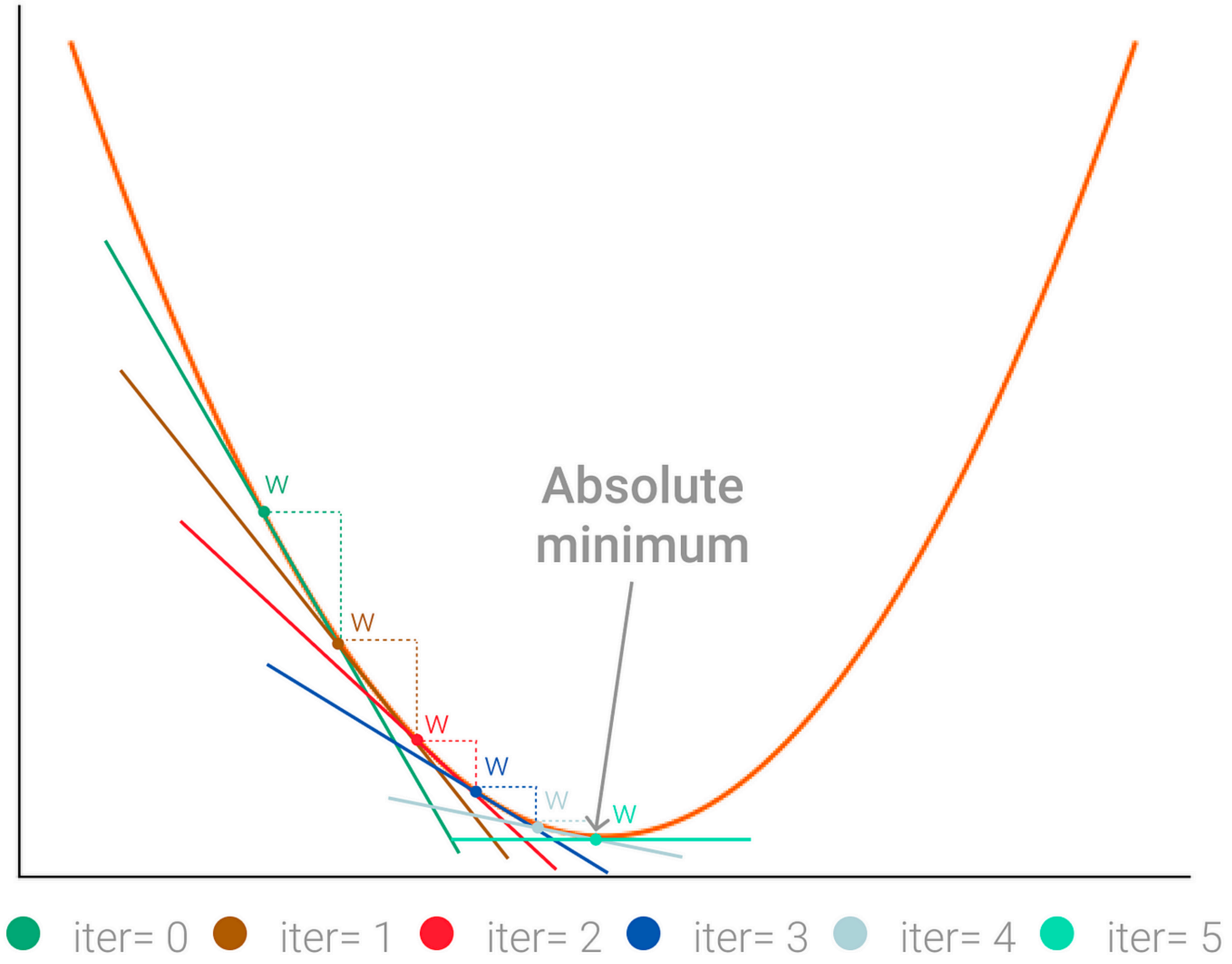
Gradient Descent!

$$w_0 \sim \mathbf{D}(w)$$

$$w_i = w_{i-1} - \eta \nabla_w NN(w, x)$$

Gradient:

$$\nabla f(x, y, z) := [\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}]$$



Insights on gradient descent

- Backed up by statistics when functions are convex
- Neural Networks are not convex
- Gradient descent still works very well to train Neural Networks

- Biological neural networks (our brains) can’t perform gradient optimization

Let’s code!

Modelling a generic function

We can decompose

$f(a, b) = (a + b) * (b + 1)$

- $c = a + b$
- $d = b + 1$
- $e = c * d$
- $f(a, b) = e$

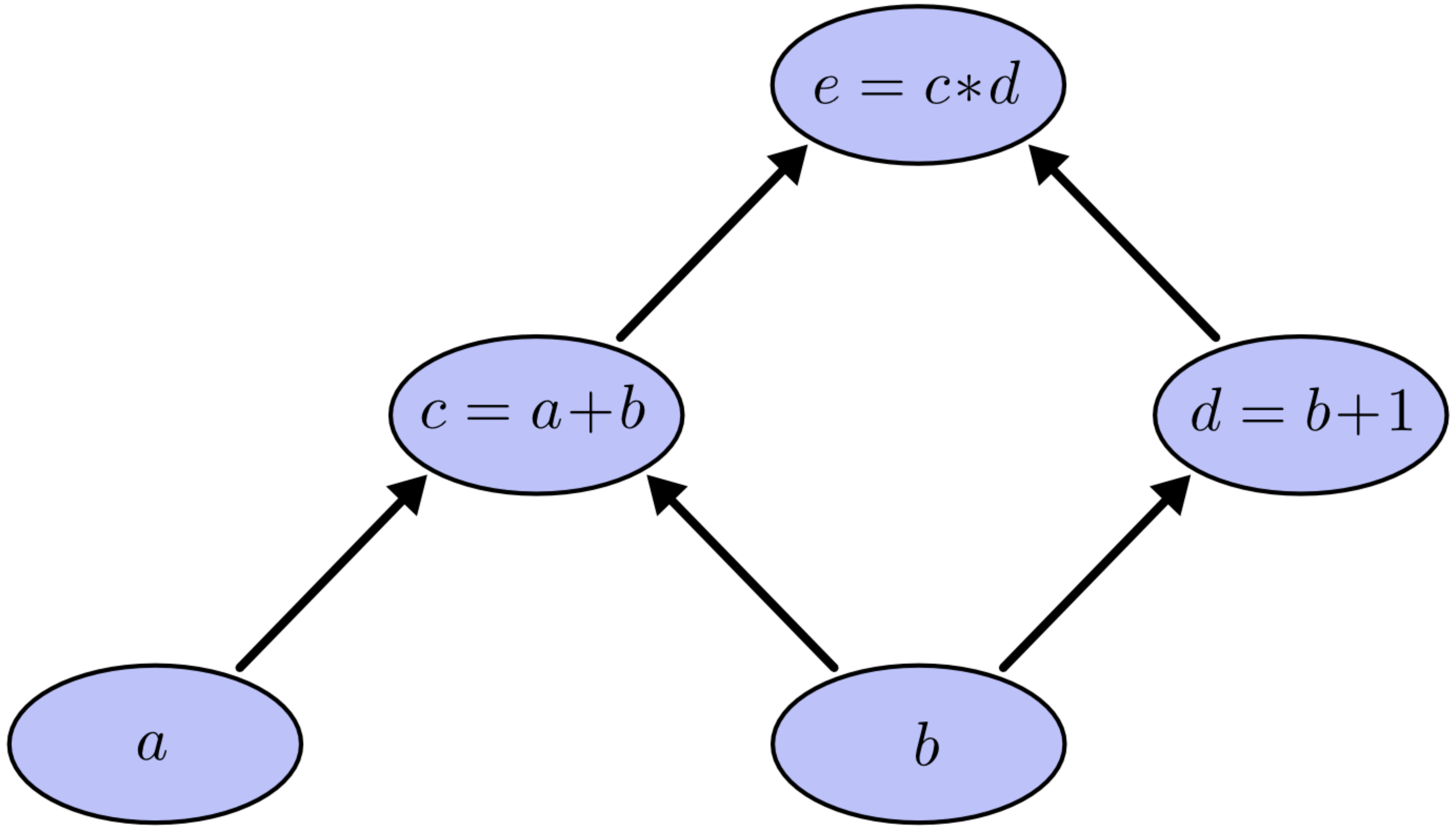
Computational graph

$$f(a, b) = (a + b) * (b + 1)$$

$$f(2,1) = ?$$

$$\nabla f(a, b) = [\frac{df}{da}, \frac{df}{db}]$$

$$\nabla f(2,1) = [?, ?]$$



Thanks to https://colah.github.io/posts/2015-08-Backprop/

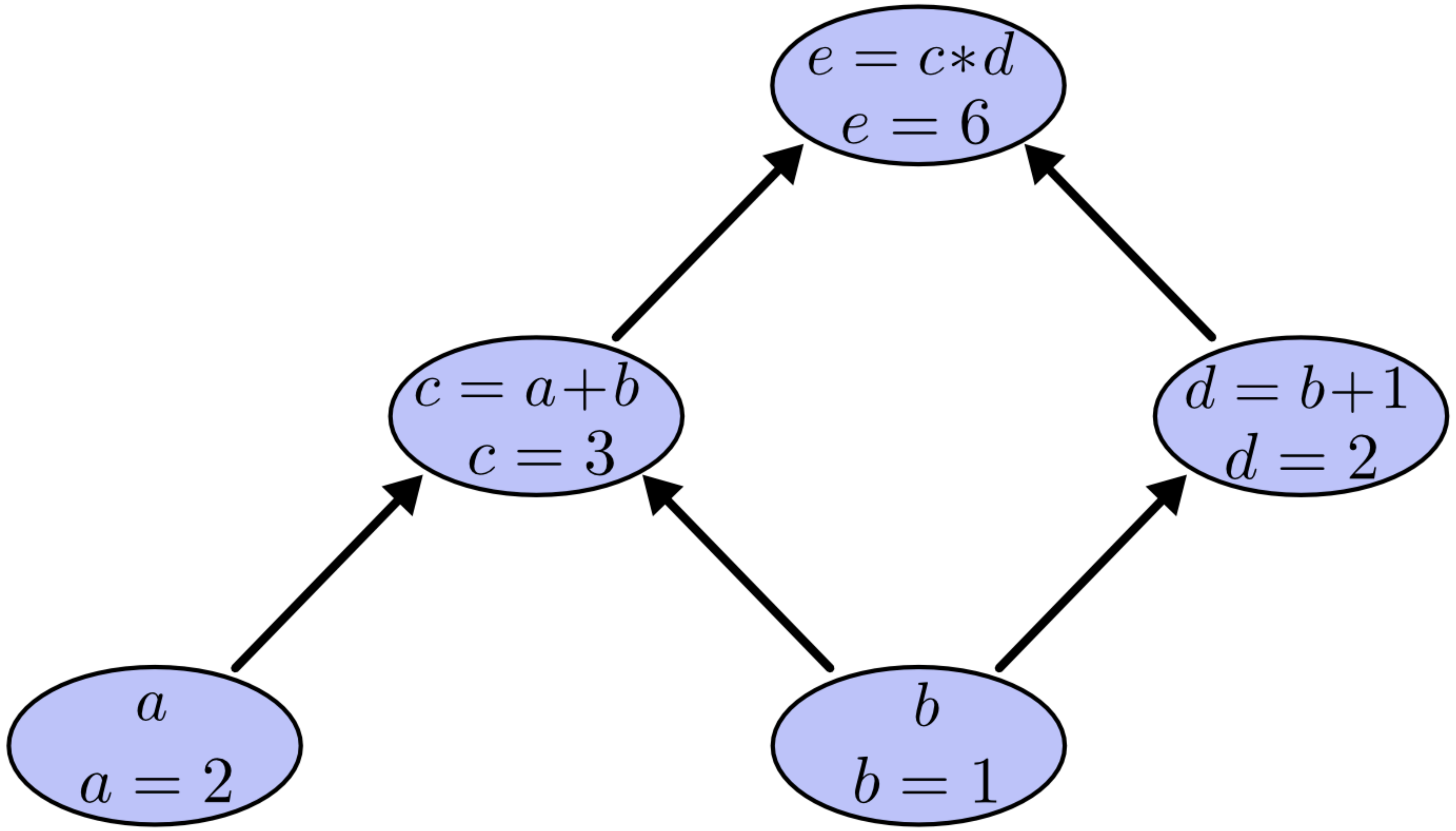
Computational graph

$$f(a,b) = (a+b) * (b+1)$$

$$f(2,1) = 6$$

$$\nabla f(a,b) = [\frac{df}{da}, \frac{df}{db}]$$

$$\nabla f(2,1) = [?,?]$$



Thanks to https://colah.github.io/posts/2015-08-Backprop/

Let’s code!

Automatic differentiation

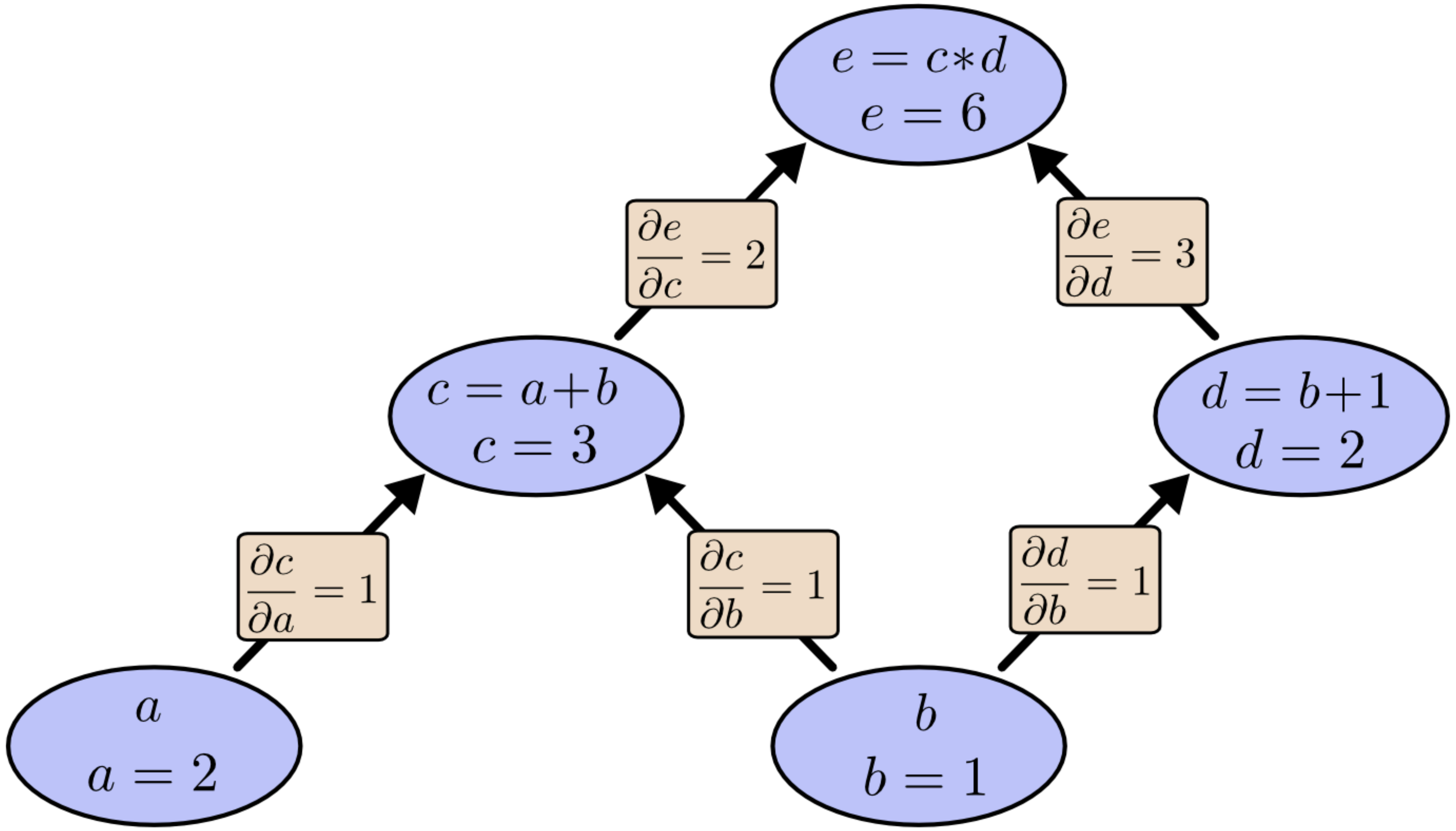
Let's start with the trivials

$$f(a, b) = (a + b) * (b + 1)$$

$$f(2,1) = 6$$

$$\nabla f(a, b) = [\frac{df}{da}, \frac{df}{db}]$$

$$\nabla f(2,1) = [?, ?]$$



Thanks to https://colah.github.io/posts/2015-08-Backprop/

Chain rule

$$f = g \circ h$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

$$f' = (g' \circ f) \cdot f'$$

Follow the paths

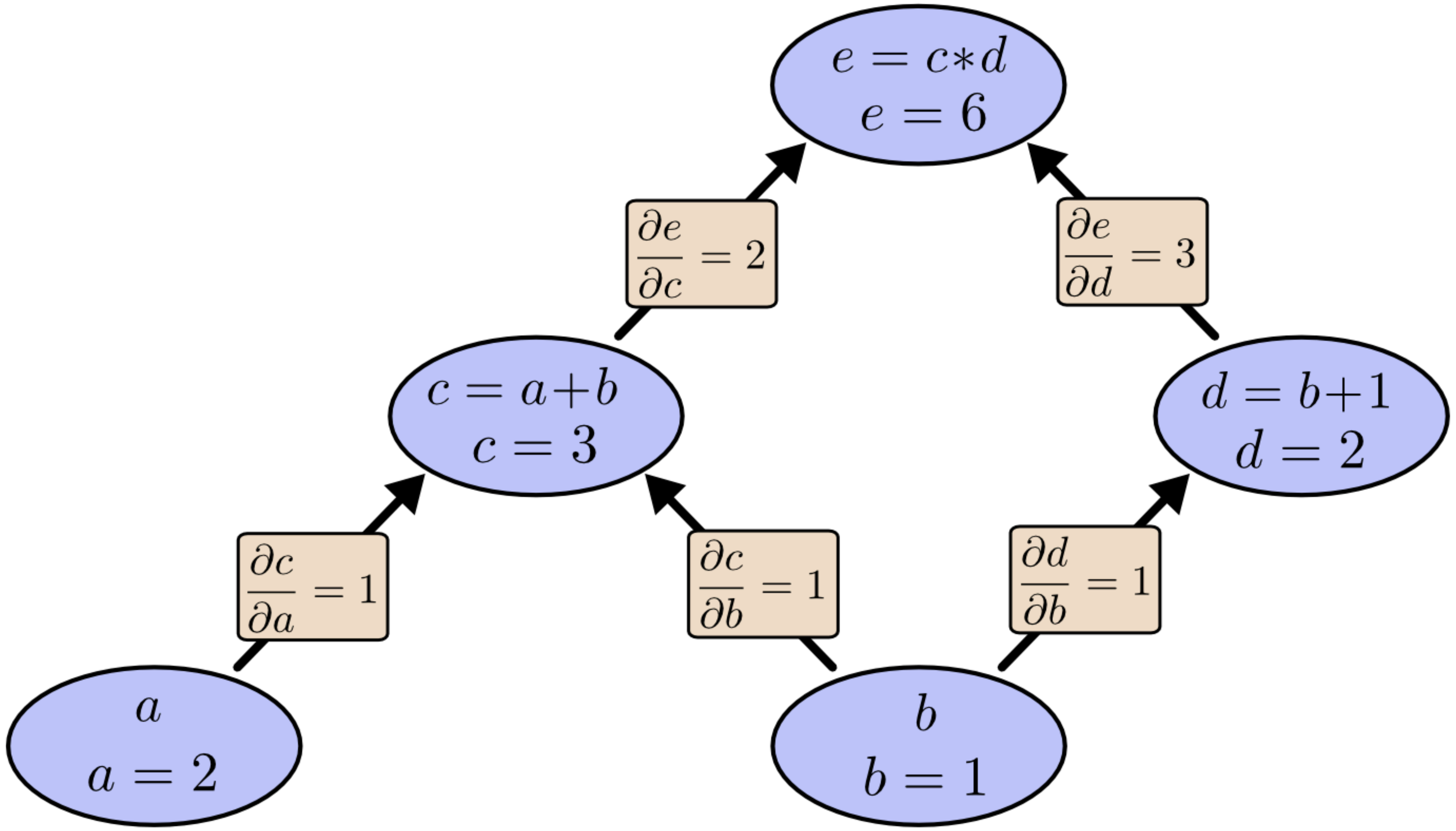
$$f(a,b) = (a+b) * (b+1)$$

$$\nabla f(a,b) = [\frac{df}{da}, \frac{df}{db}]$$

$$\frac{df}{da} = \frac{dc}{da} \cdot \frac{de}{dc} = 2$$

$$\frac{df}{db} = \frac{dc}{db} \cdot \frac{de}{dc} + \frac{dd}{db} \cdot \frac{de}{dd} = 5$$

$$\nabla f(2,1) = [2,5]$$

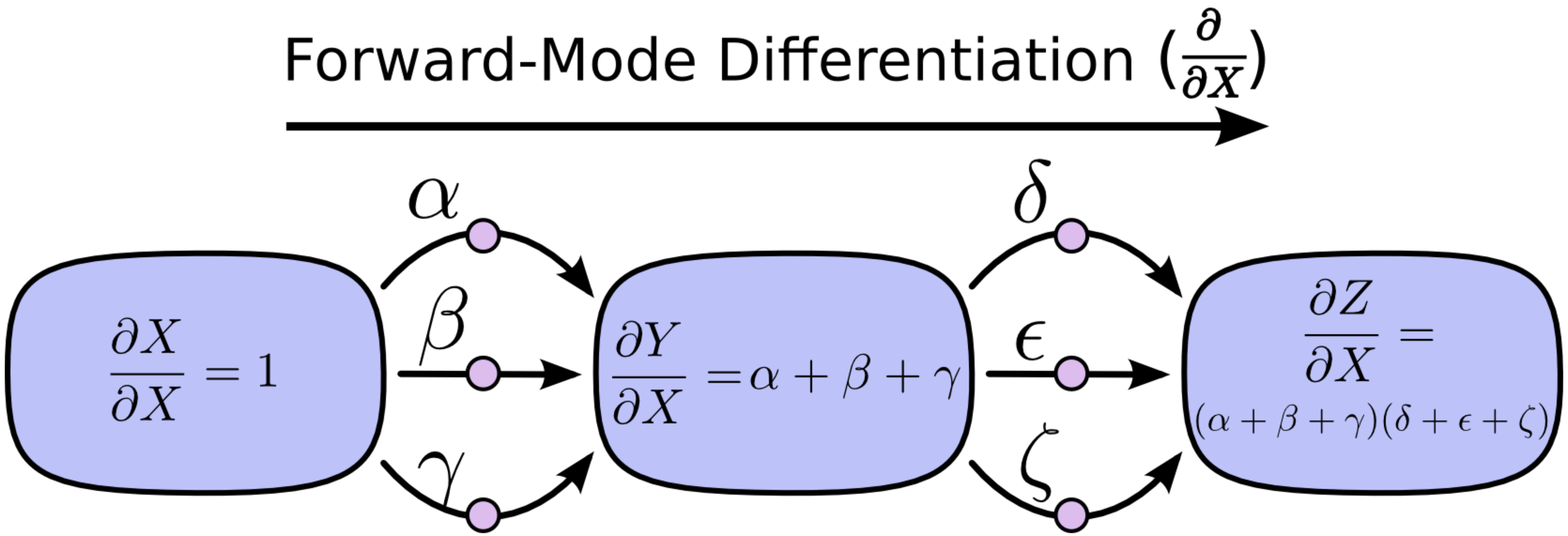


Thanks to https://colah.github.io/posts/2015-08-Backprop/

$O(N!)$ **paths**

Forward mode autodiff

Let's do some dynamic programming



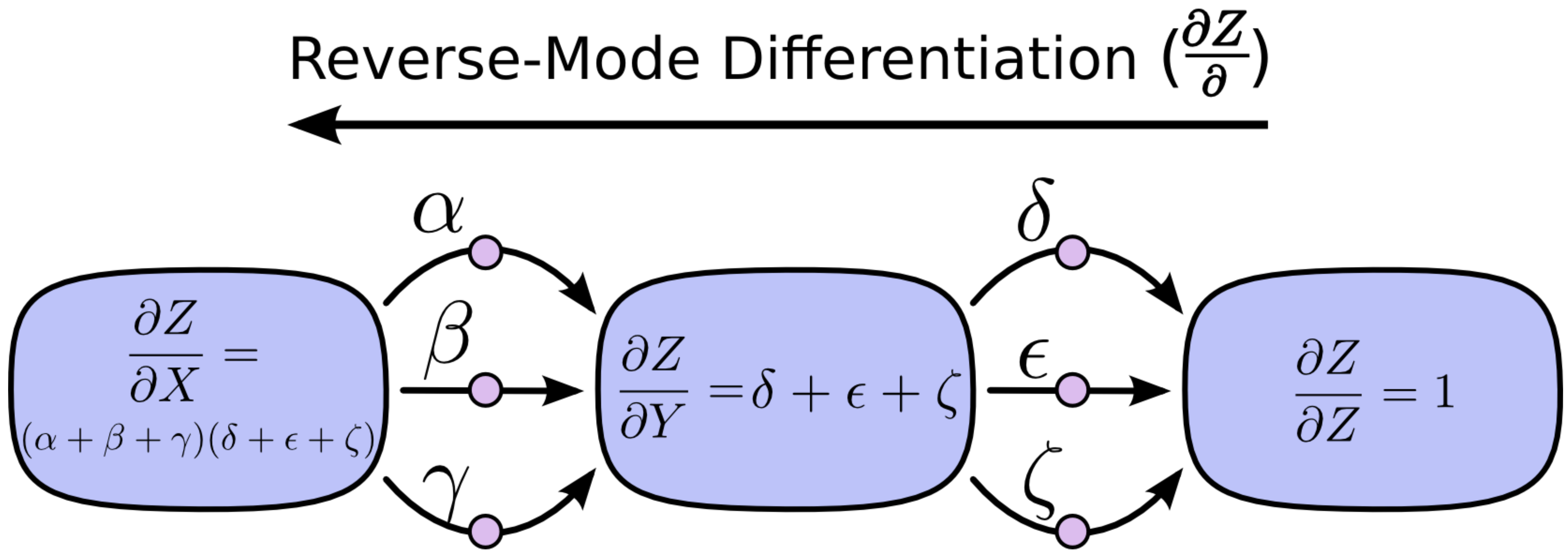
Thanks to https://colah.github.io/posts/2015-08-Backprop/

$$O(N)$$

(But we need to run it for every parameter)

Reverse mode autodiff

Focus on the output



Thanks to https://colah.github.io/posts/2015-08-Backprop/

$$O(N)$$

Let’s code!