Neural Computation

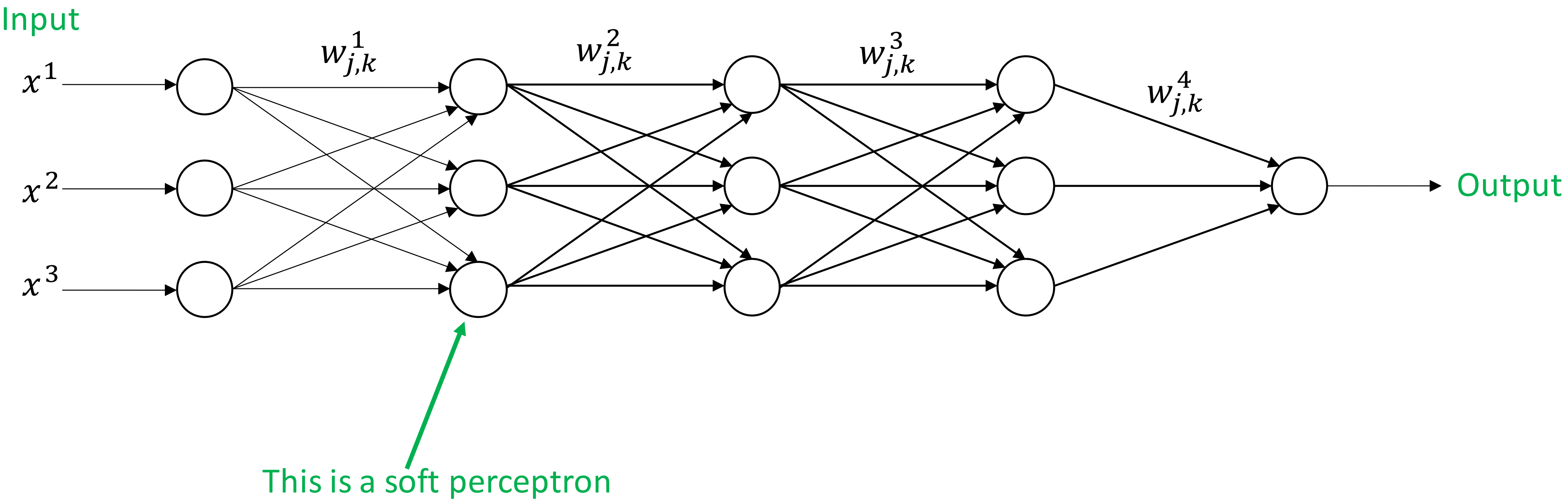
Neural Networks,

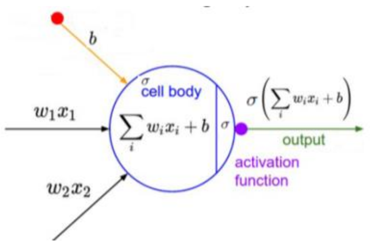
Computation Graphs

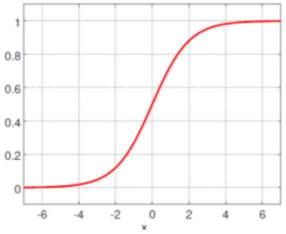
and

Backpropagation

Multi-Layer-Perceptron (MLP) aka Feed-Forward-Net

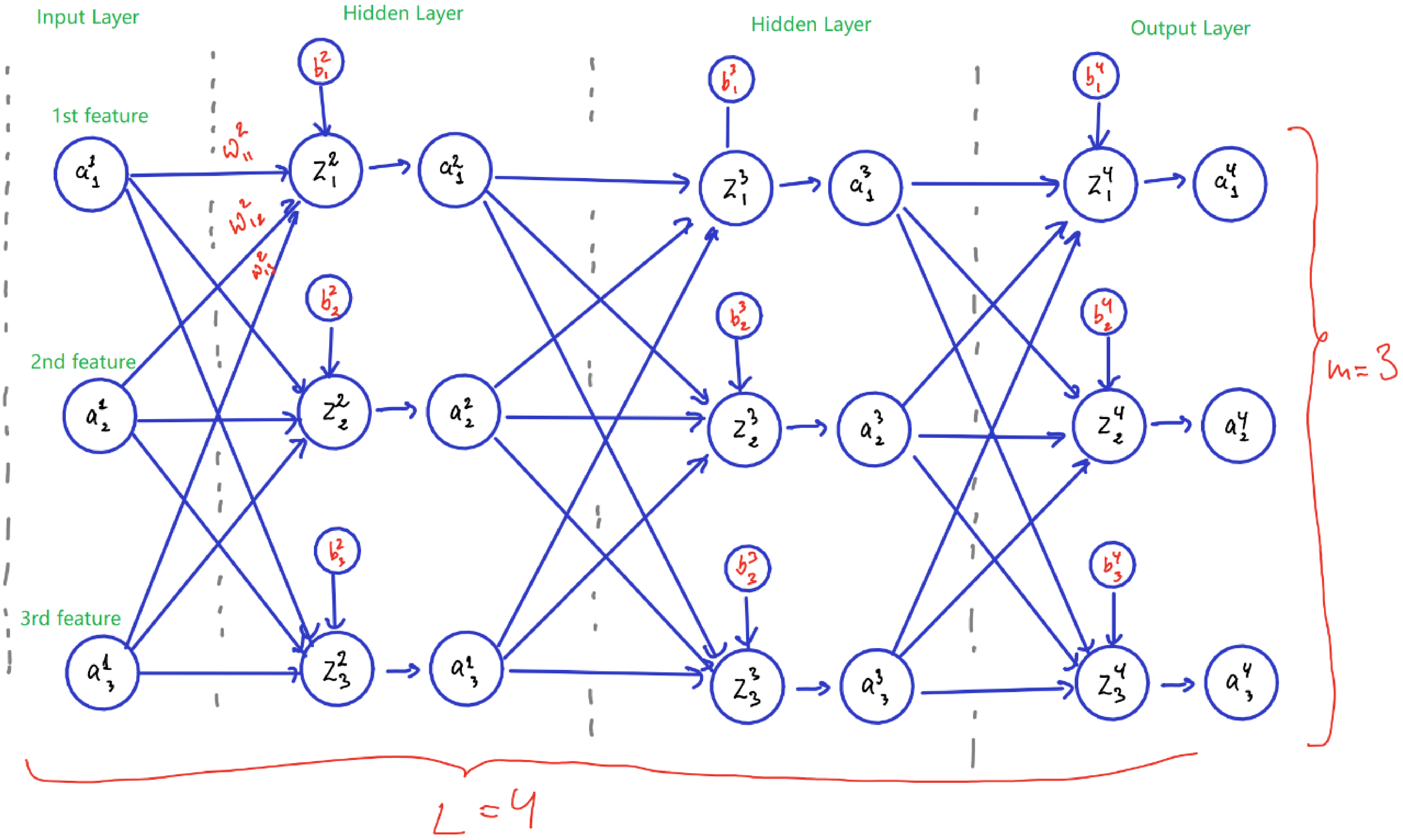






In the this video:

- Compute derivatives for MLP
- Efficient algorithm (backpropagation)



L: number of layers (superscript 1 is "input layer", superscript L is "output layer")

m: "width" of network (can vary between layers)

ω^{ℓ}_{jk}: "weight" of connection between k-th unit in layer $\ell-1$, to j-th unit in layer l

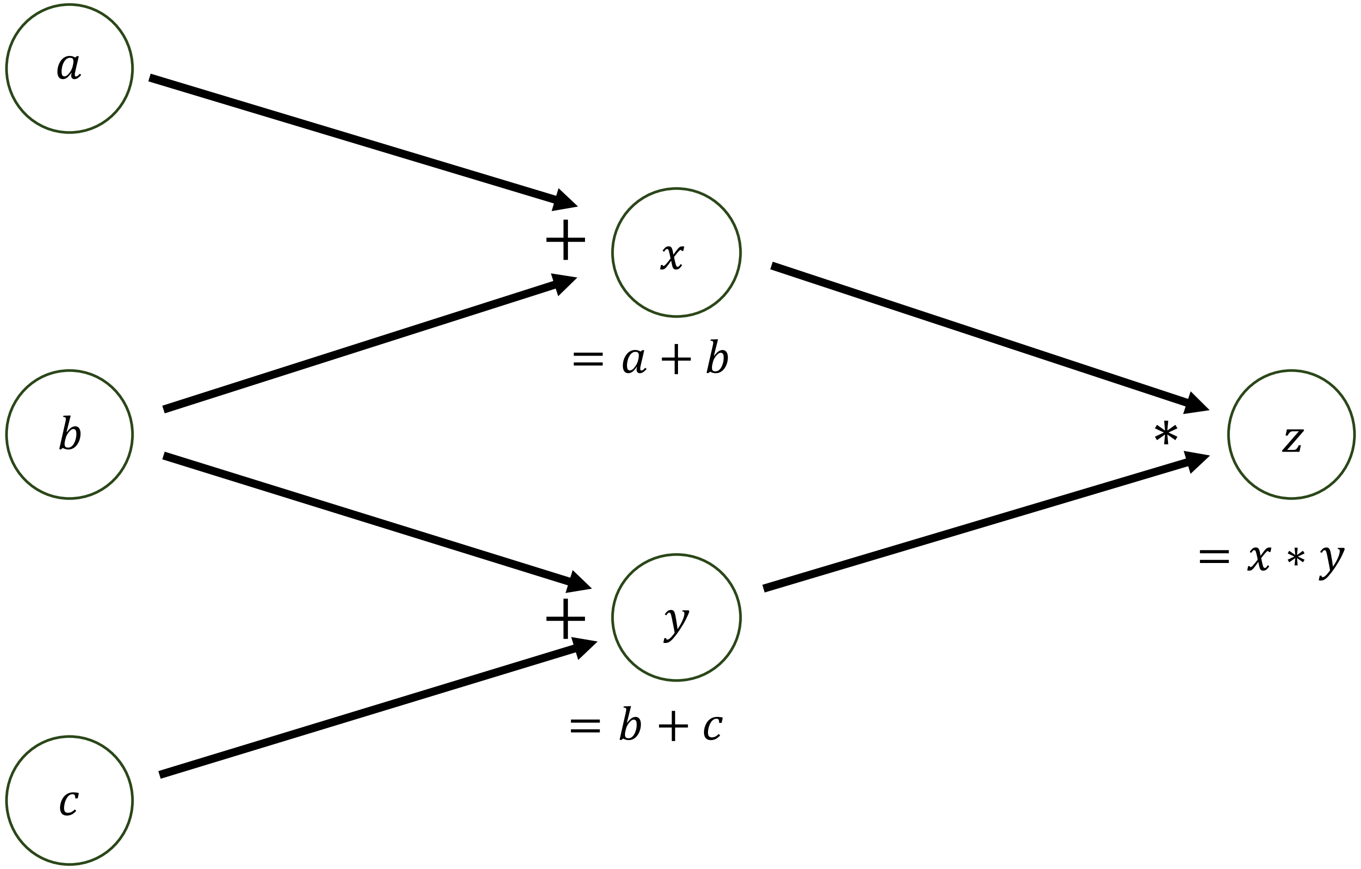
b^{ℓ}_{j}: "bias" of j-th unit in layer l

$z^{\ell}_{j} = \sum_k \omega^{\ell}_{jk} a^{\ell-1}_{k} + b^{\ell}_{j}$: weighted input to unit j in layer ℓ

$a^{\ell}_{j} = \sigma(z^{\ell}_{j})$: "activation" of unit j in layer ℓ, where σ is an "activation function"

Computation Graphs

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$



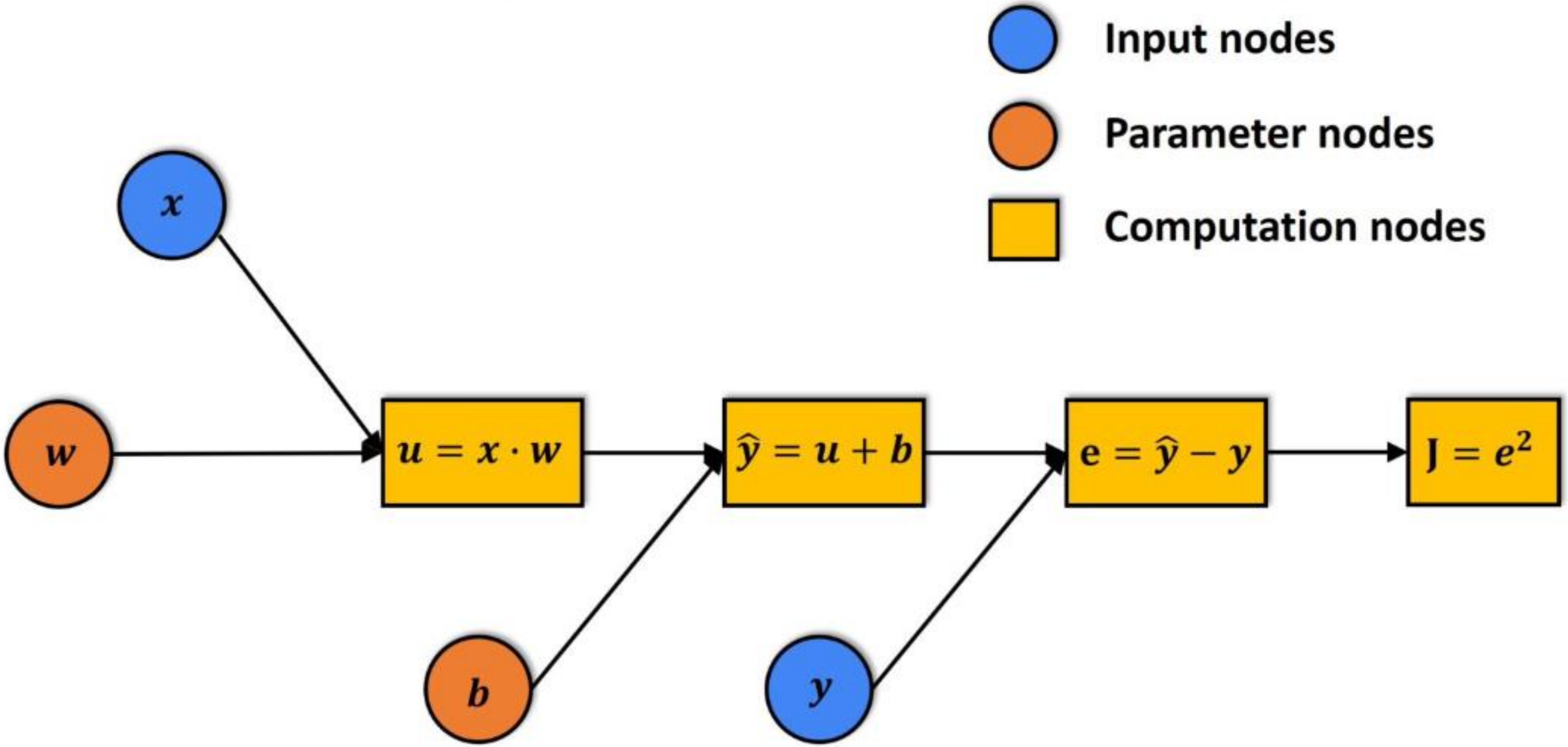
Directed acyclic graph
Nodes:

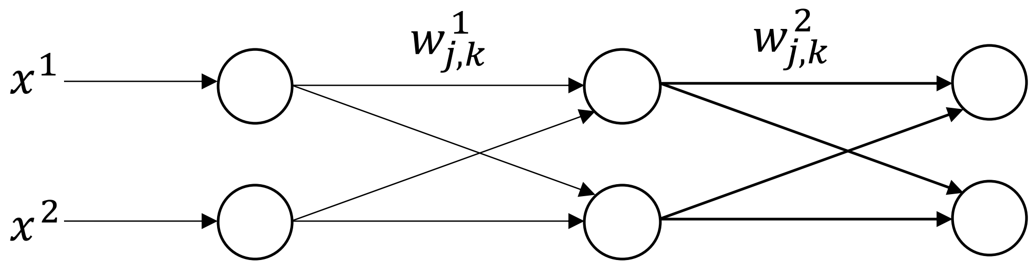
- Inputs, outputs or intermediate results

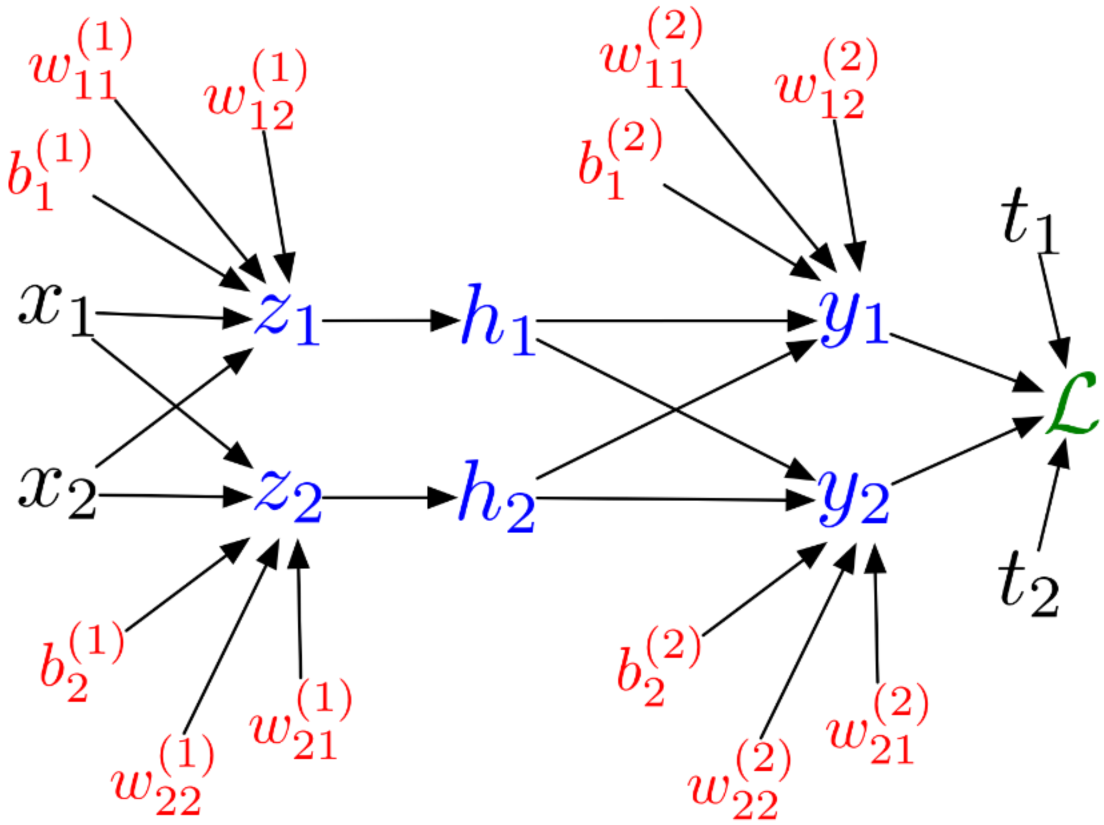
Edges:

- Indicate computation

Computation Graph for Linear Regression







Roger Grosse: CSC 311 Spring 2020: Introduction to Machine Learning

Chain Rule in Computation Graph

- Consider

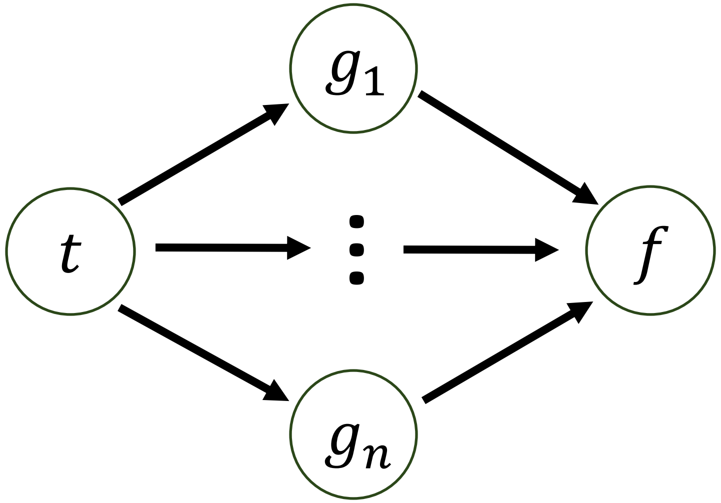
$$f = f(g_1, \dots, g_n),$$

where all

$g_i = g_i(t)$ are functions of t

- We can compute the derivative as

$$\boxed{\frac{\partial f}{\partial t} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial t}}$$



Chain Rule in Computation Graph

- Consider

$$f = f(g_1, \dots, g_n),$$

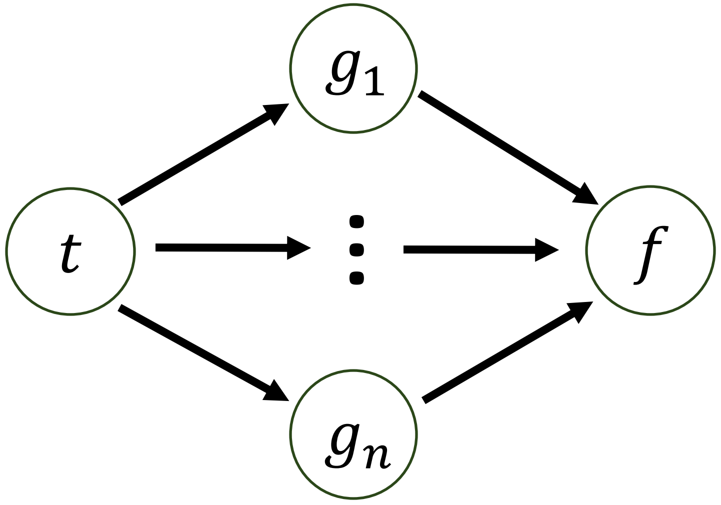
where all

$g_i = g_i(t)$ are **functions** of t

(aka **successors**)

- We can compute the derivative as

$$\boxed{\frac{\partial f}{\partial t} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial t}}$$



Chain Rule in Computation Graph

- Consider

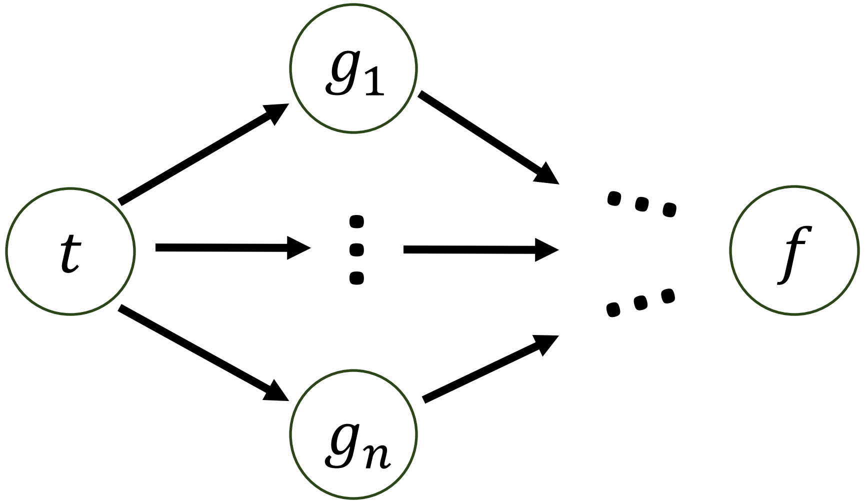
$$f = f(g_1, \dots, g_n),$$

where all

$g_i = g_i(t)$ are **successors** of t

- We can compute the derivative as

$$\boxed{\frac{\partial f}{\partial t} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial t}}$$

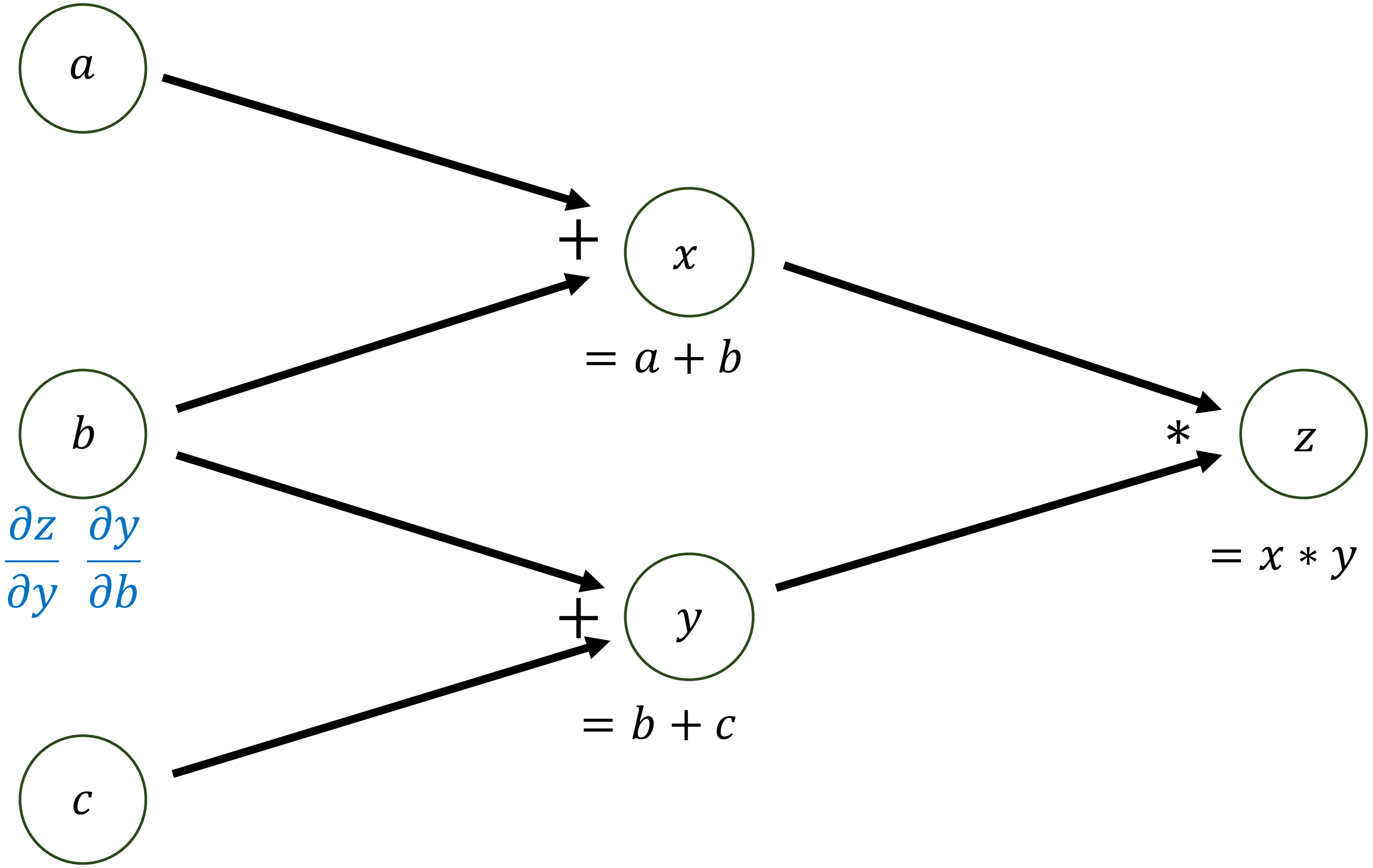


Computation Graphs

$$\frac{\partial f}{\partial t} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial t}$$

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$$\frac{\partial z}{\partial a} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial a}$$



$$\frac{\partial z}{\partial b} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial b} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial b}$$

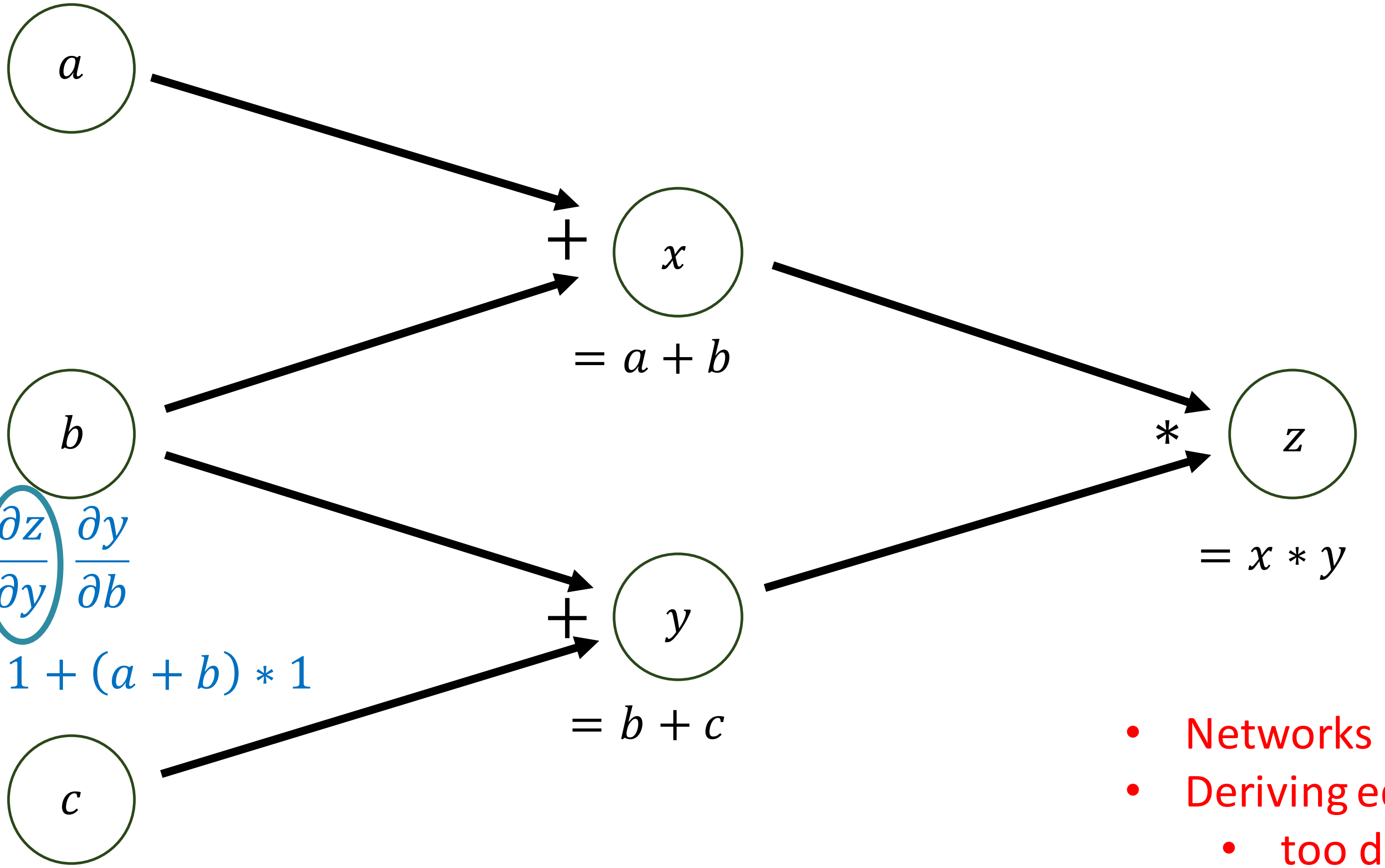
$$\frac{\partial z}{\partial c} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial c}$$

Computation Graphs

$$\frac{\partial f}{\partial t} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial t}$$

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$$\frac{\partial z}{\partial a} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial a} = (b+c) * 1$$



$$\frac{\partial z}{\partial b} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial b} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial b}$$
$$= (b+c) * 1 + (a+b) * 1$$

$$\frac{\partial z}{\partial c} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial c} = (a+b) * 1$$

- Networks have millions to billions of parameters
- Deriving equation for each parameter is
  - too difficult by hand
  - computationally inefficient
- Need efficient algorithm to compute gradient

Backpropagation Algorithm

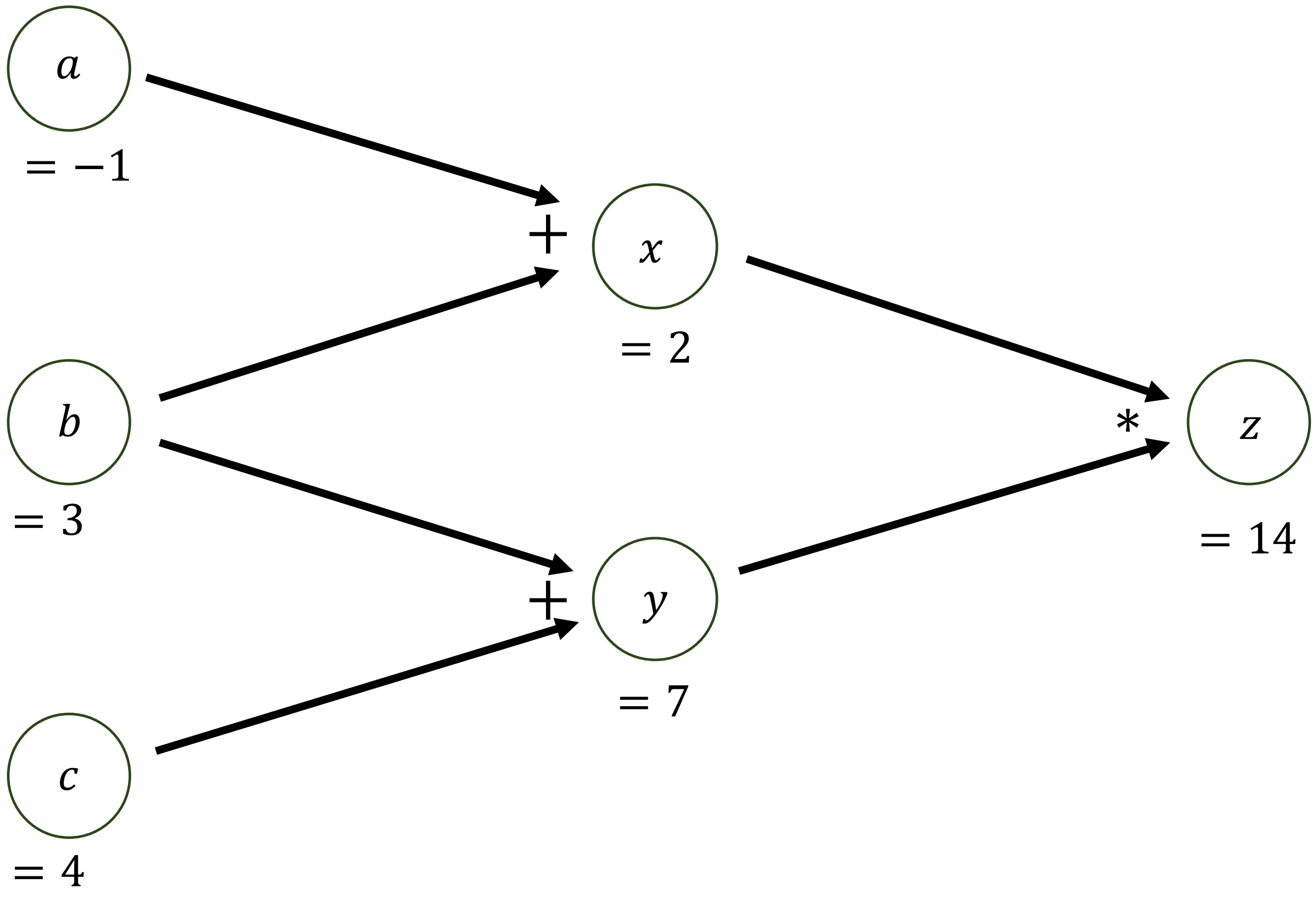
- Forward pass:
  - Move forward through graph to compute all intermediate results

- Backward pass:
  - Move backward through graph compute all gradients
    - Use result from successor nodes.

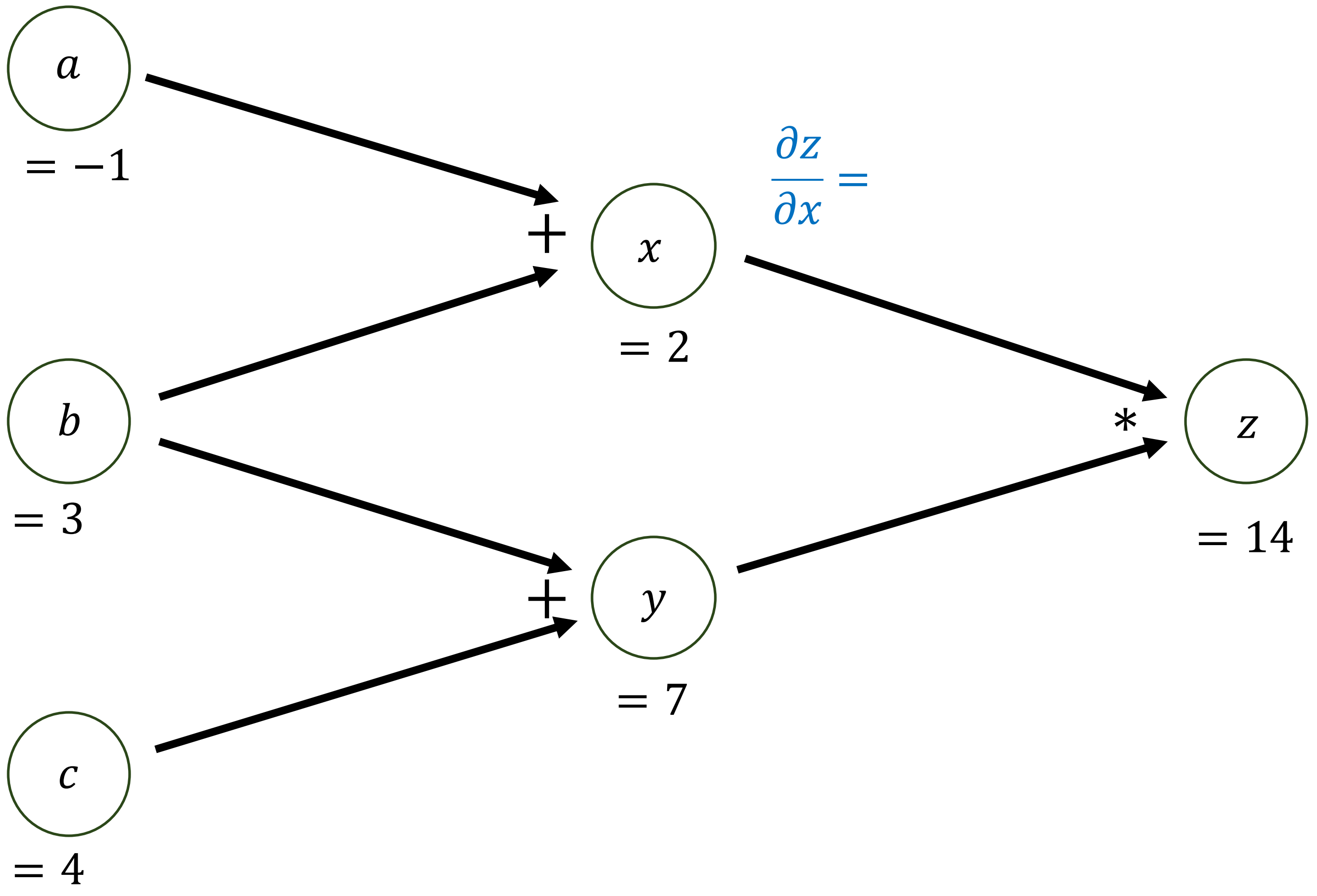
Forward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$



Backward Pass



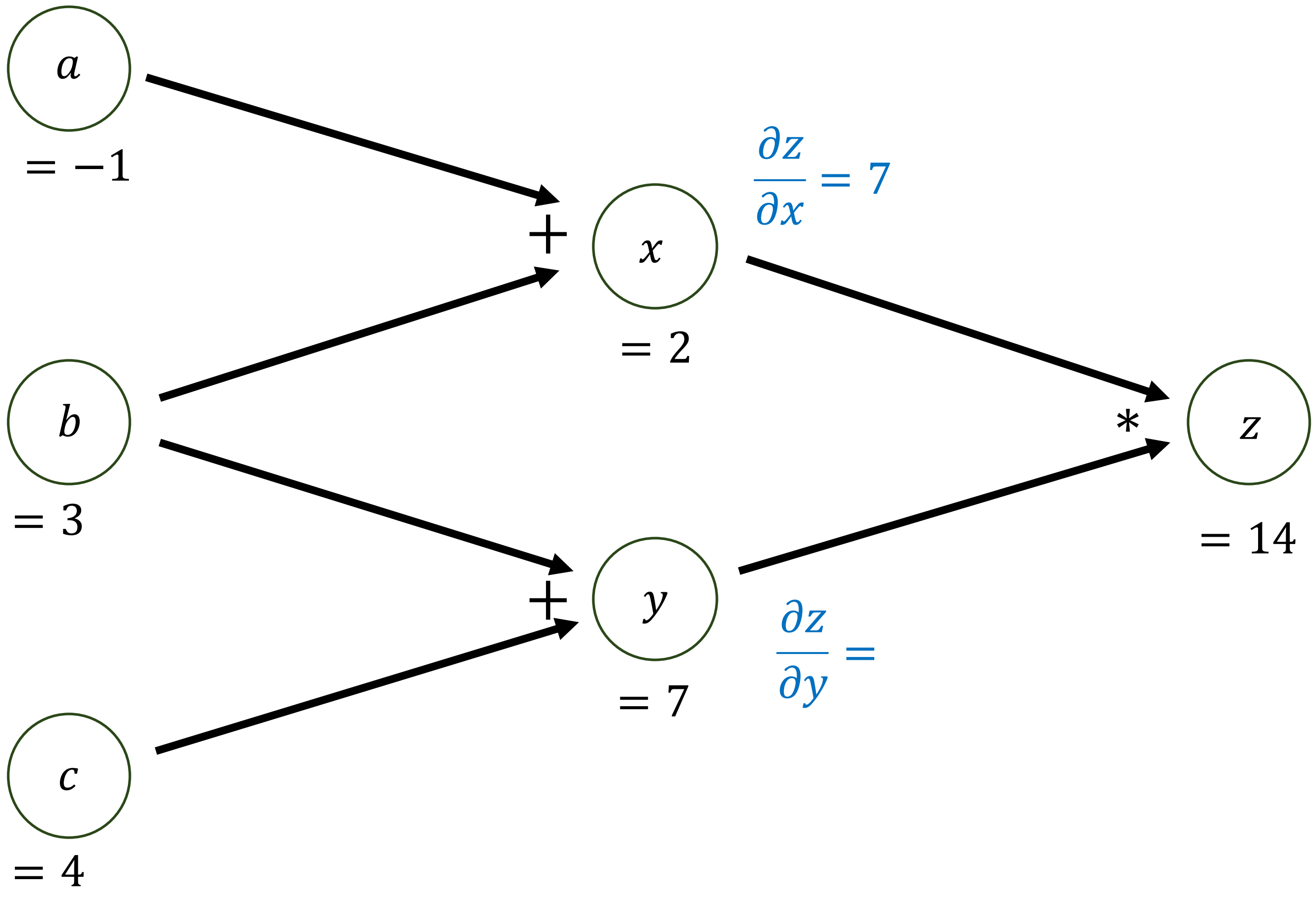
$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$

Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

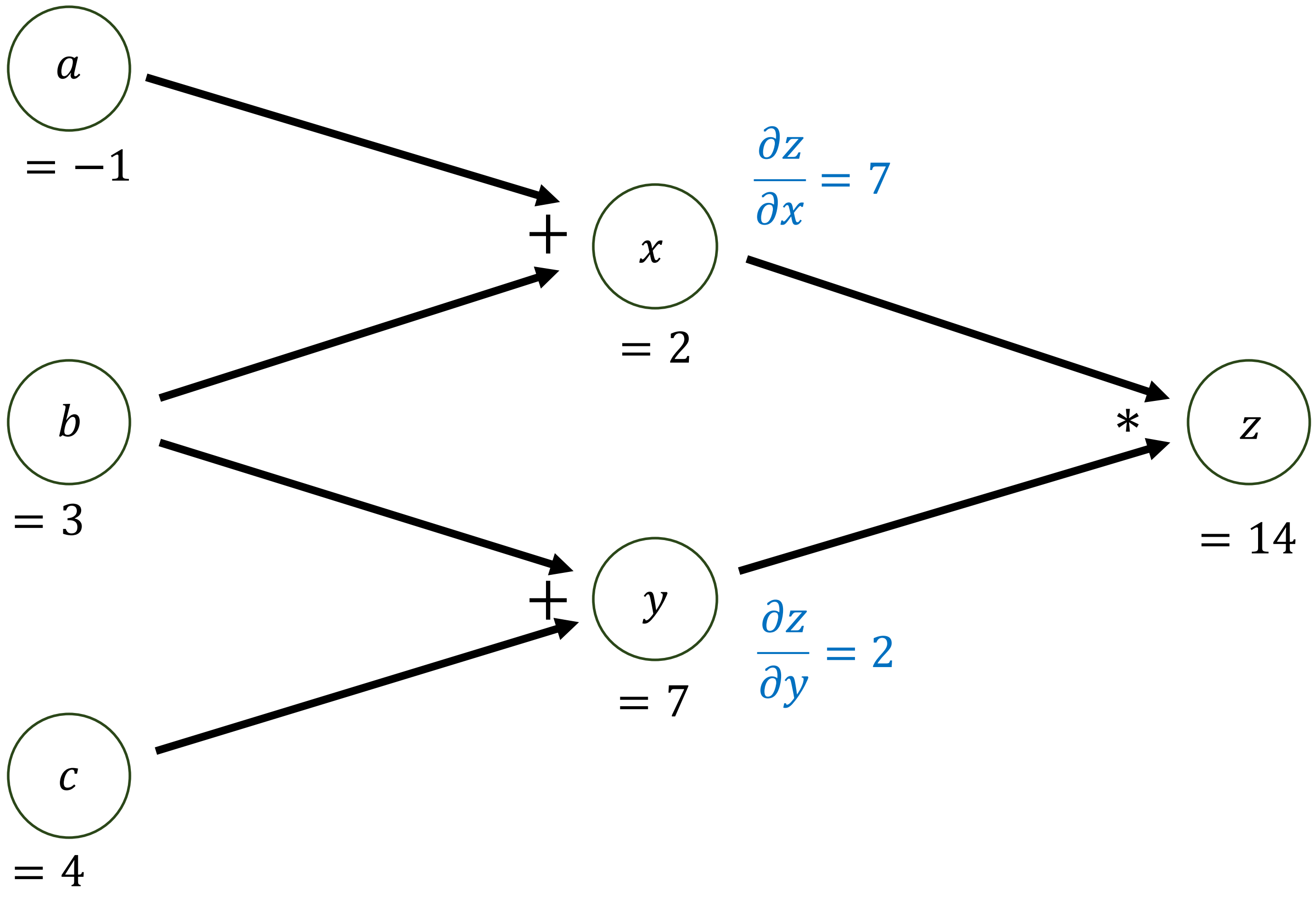
$a = -1 \quad b = 3 \quad c = 4$



Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

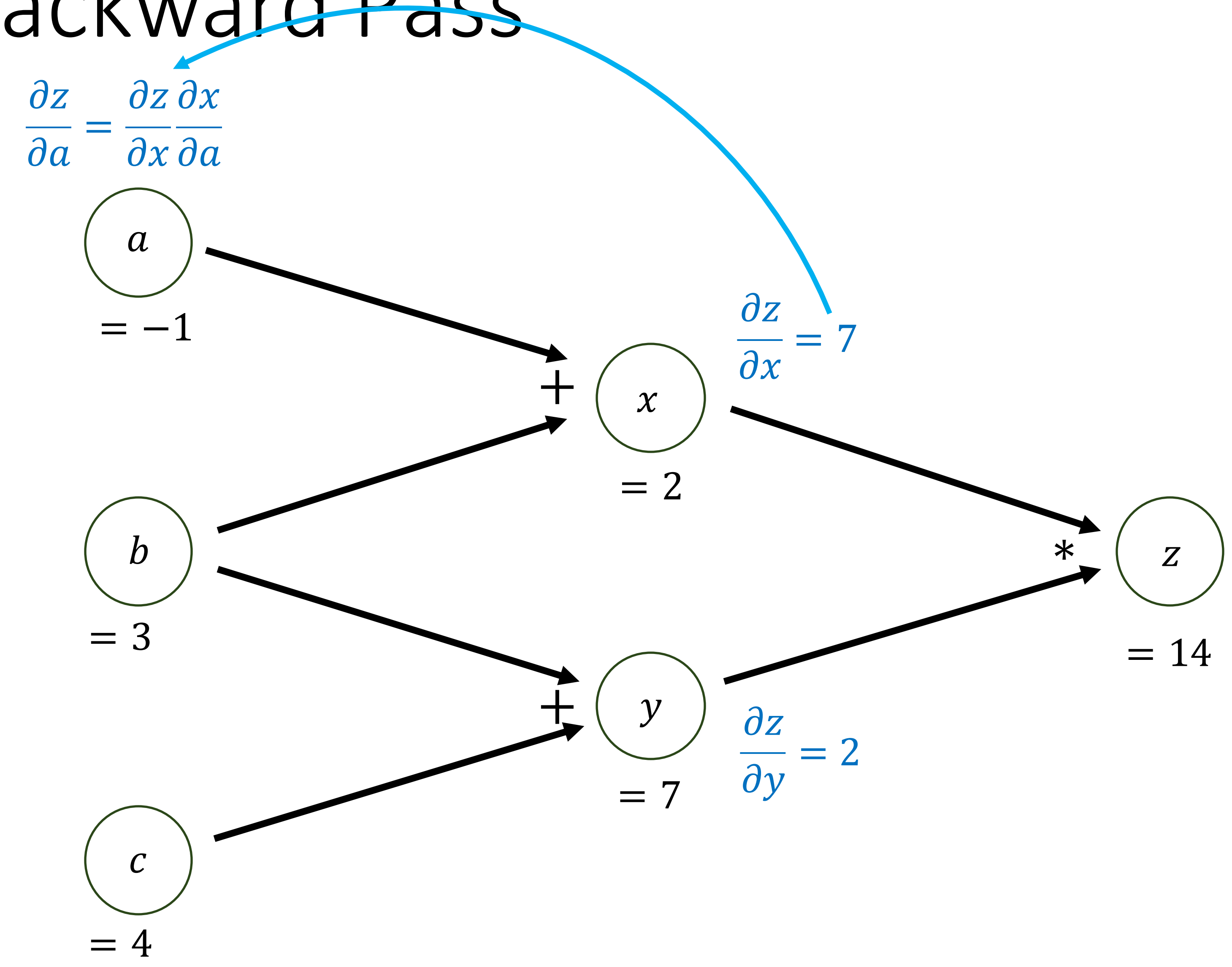
$a = -1 \qquad b = 3 \qquad c = 4$



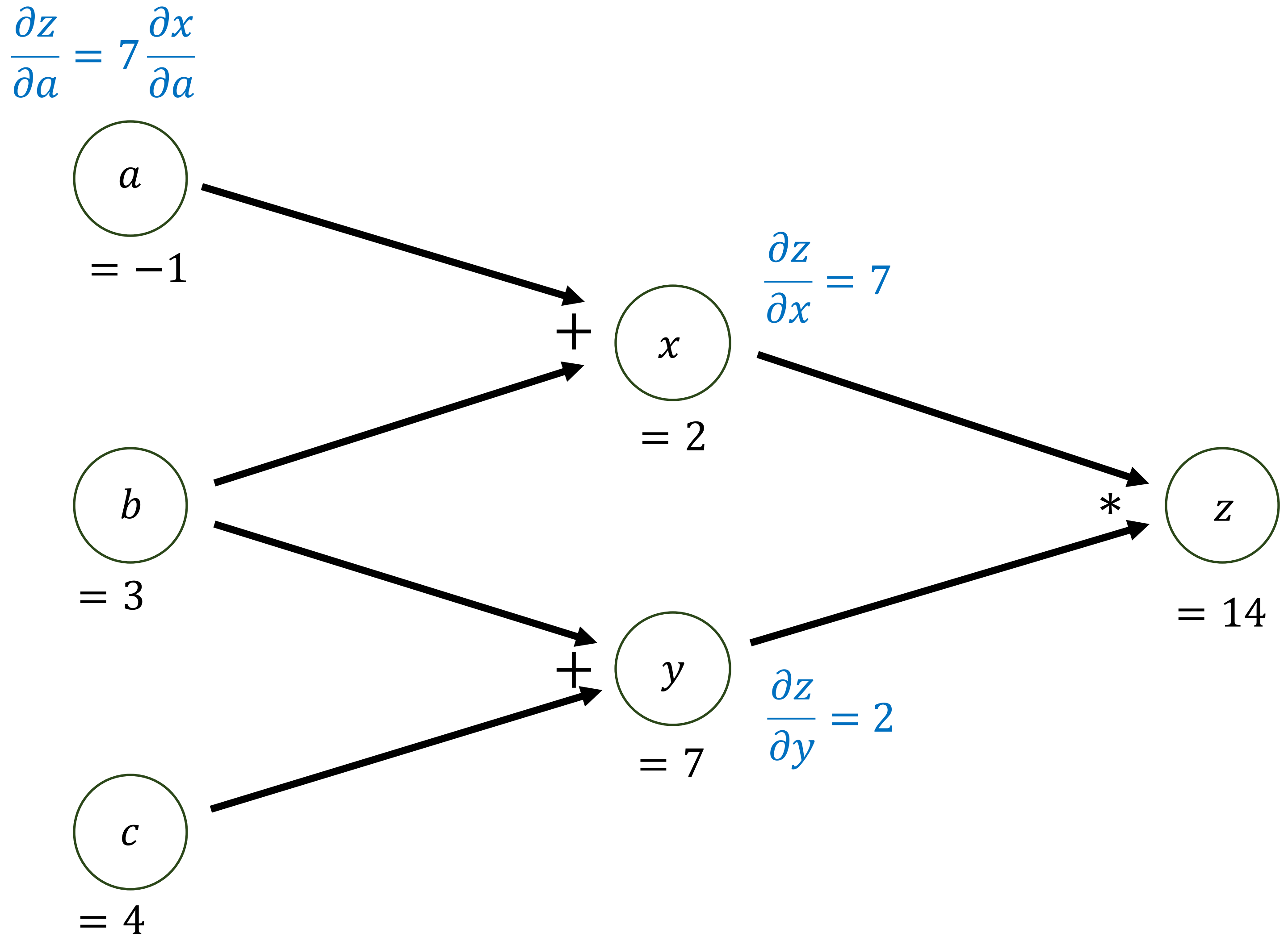
Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$



Backward Pass



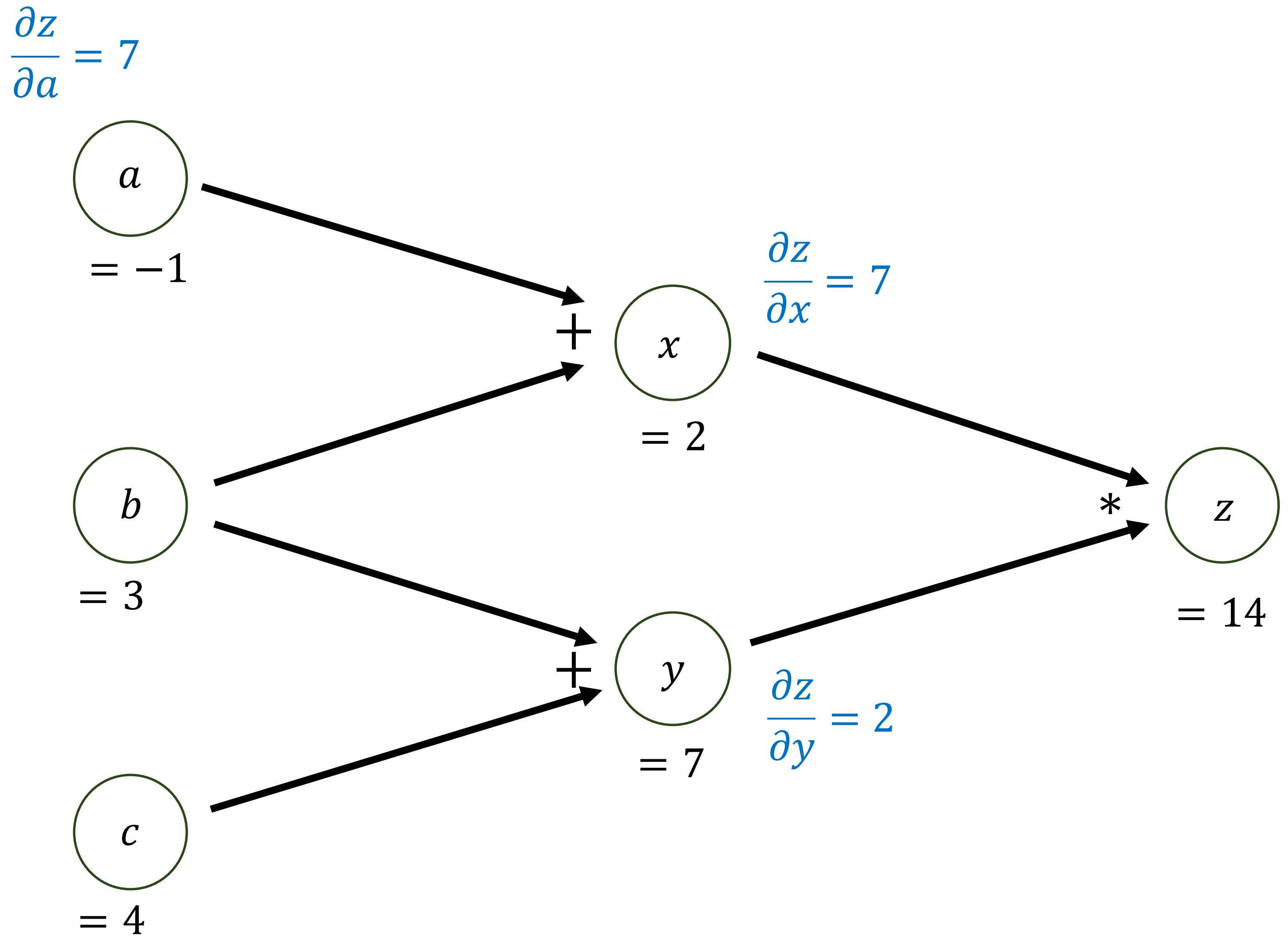
$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$

Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

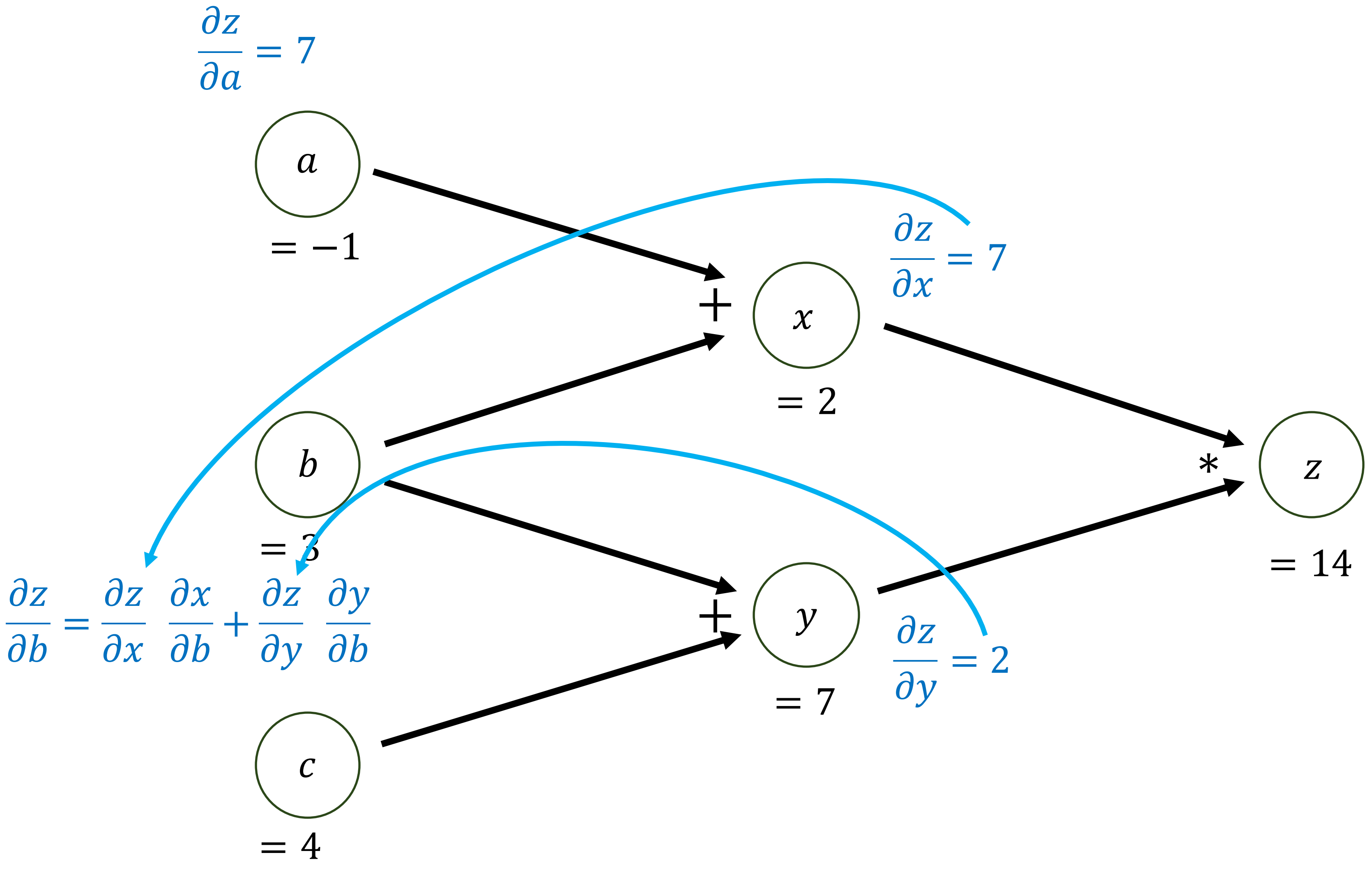
$a = -1 \qquad b = 3 \qquad c = 4$



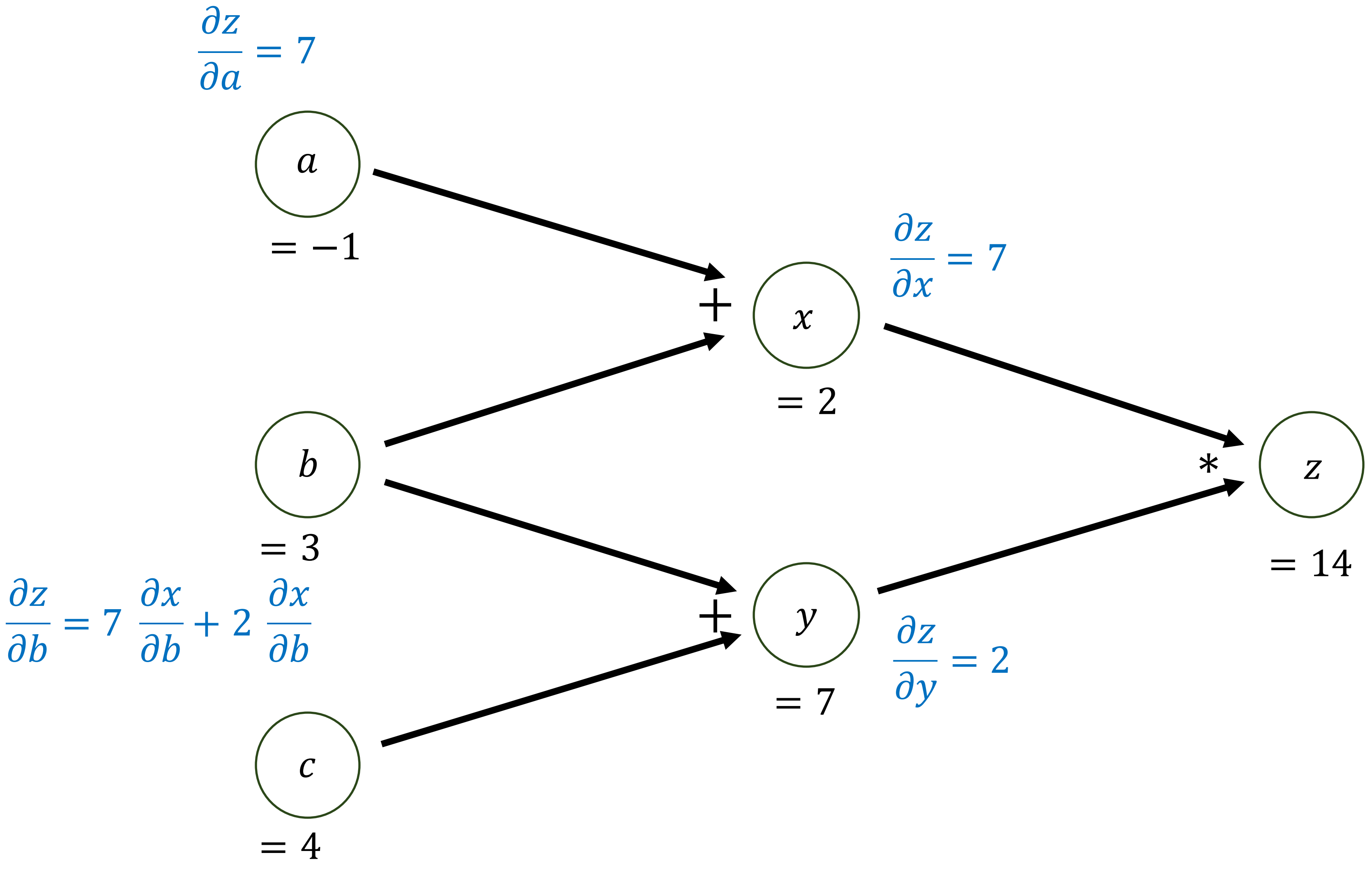
Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$



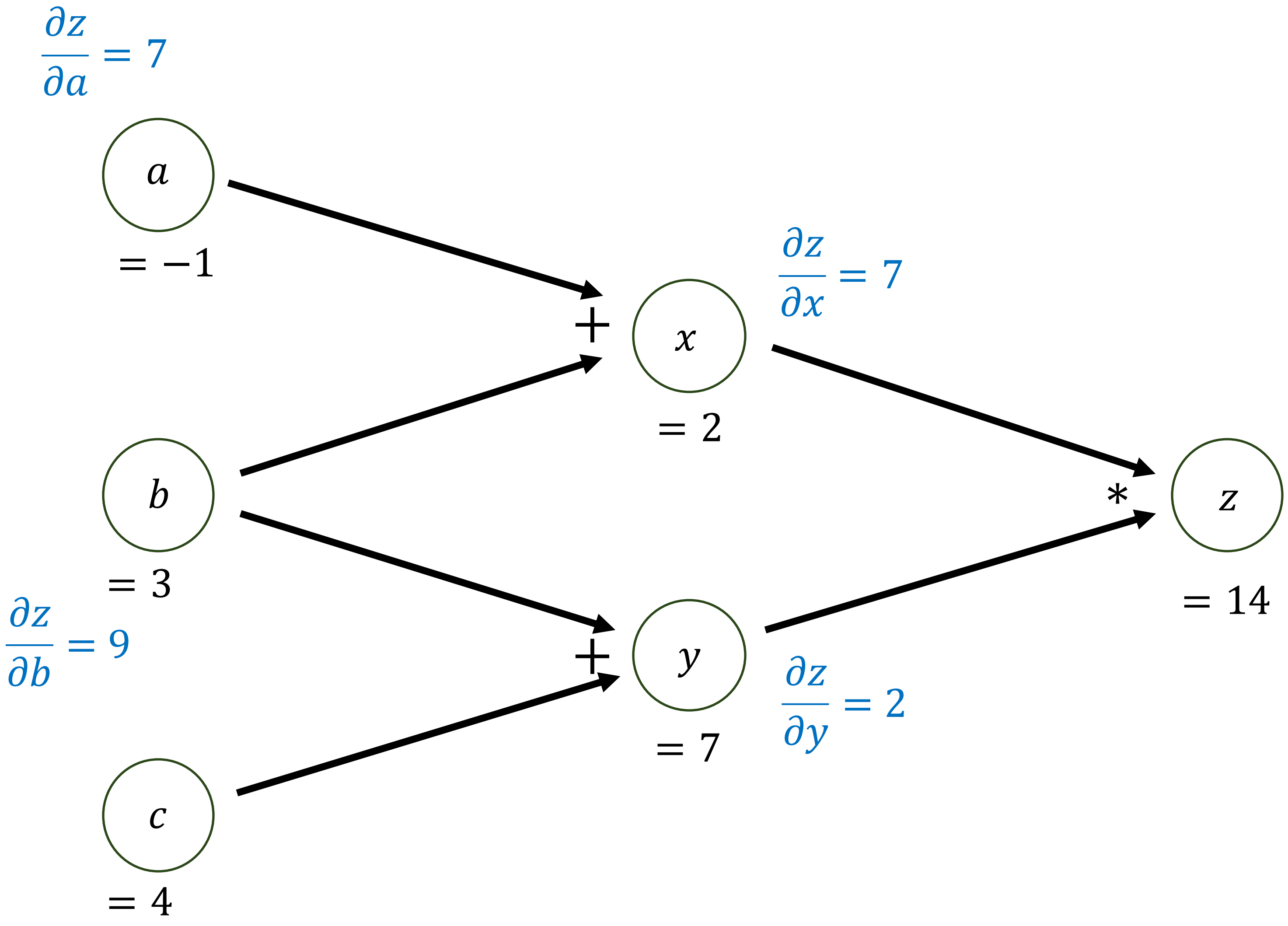
Backward Pass



$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$

Backward Pass



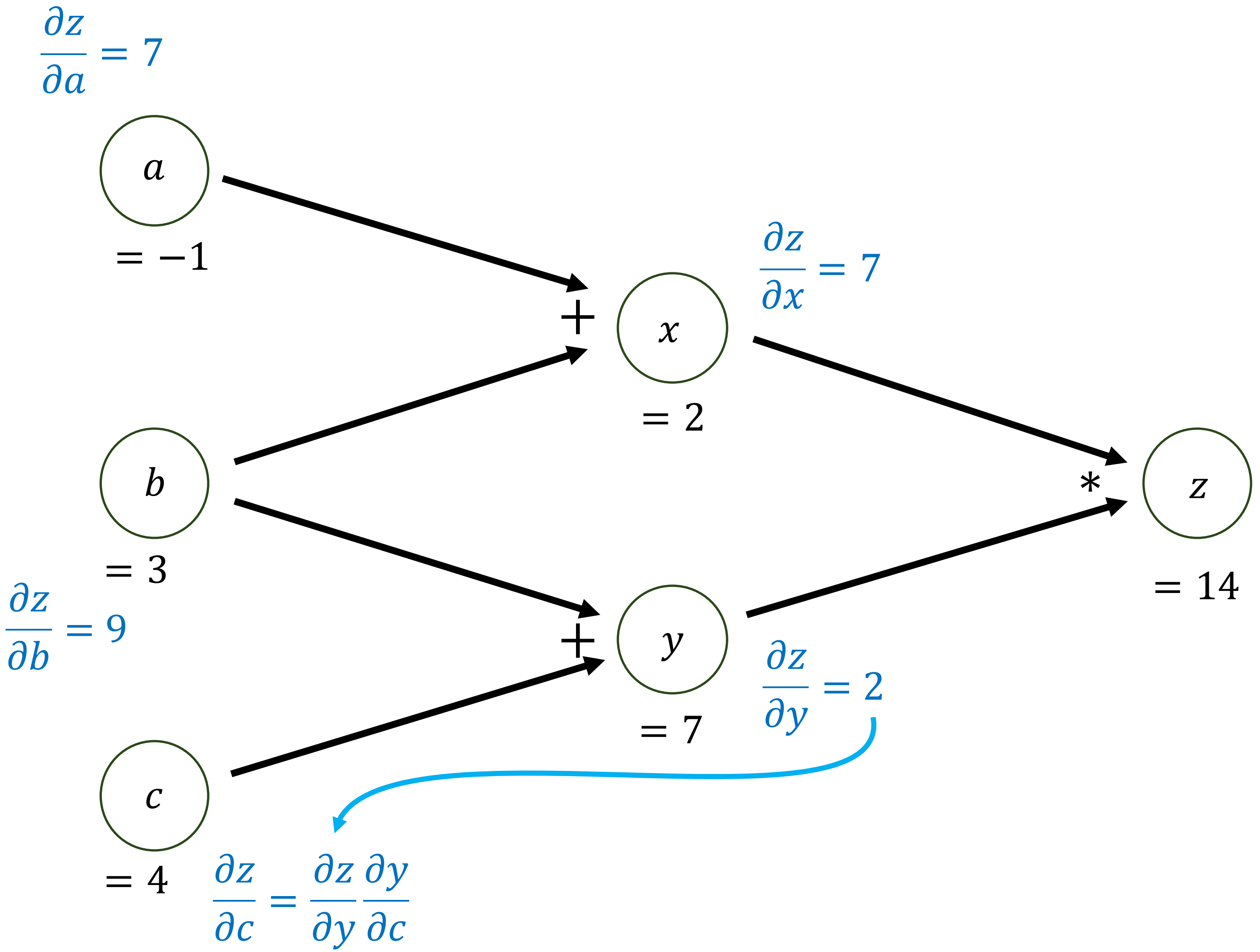
$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$

Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

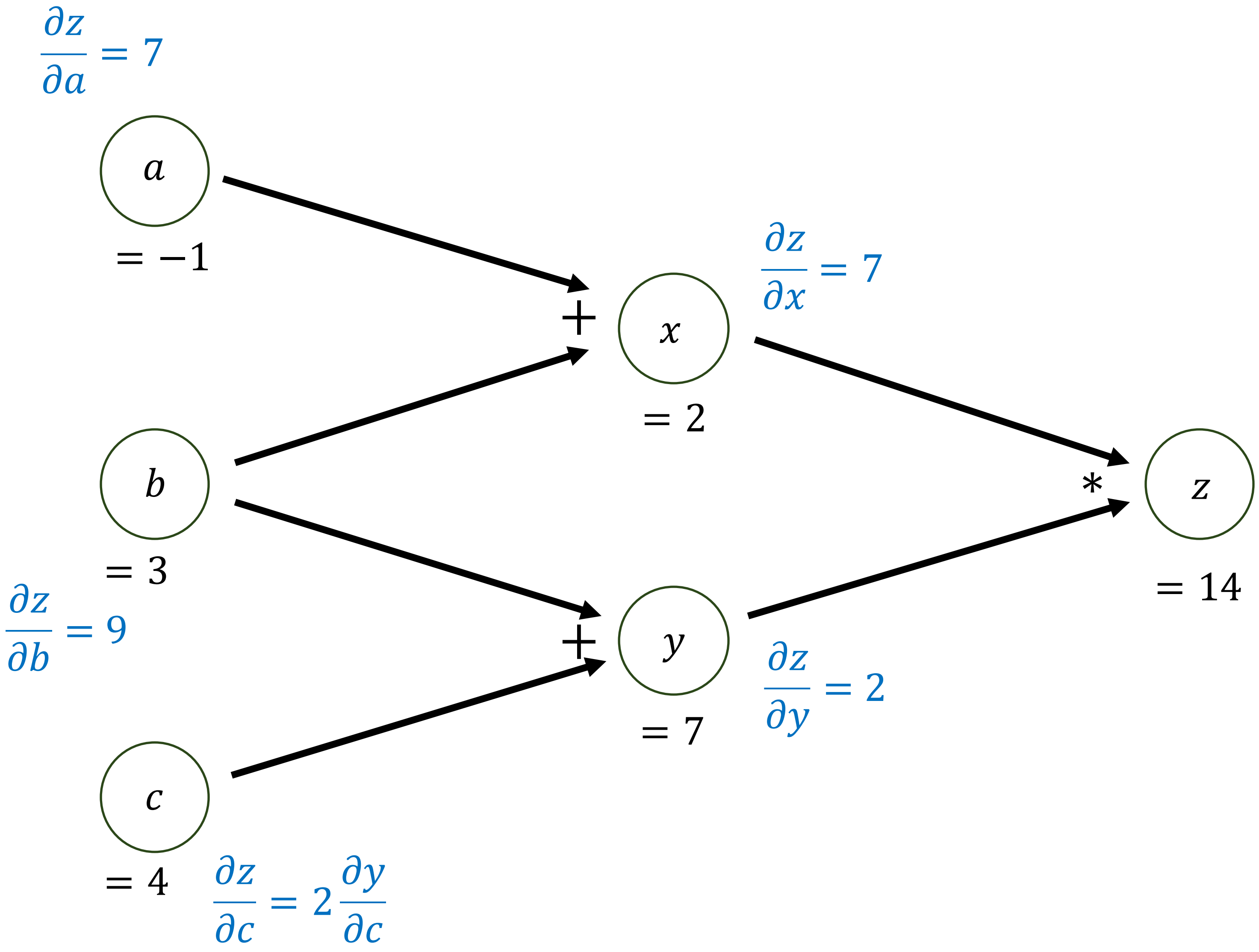
$a = -1$ $b = 3$ $c = 4$



Backward Pass

$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

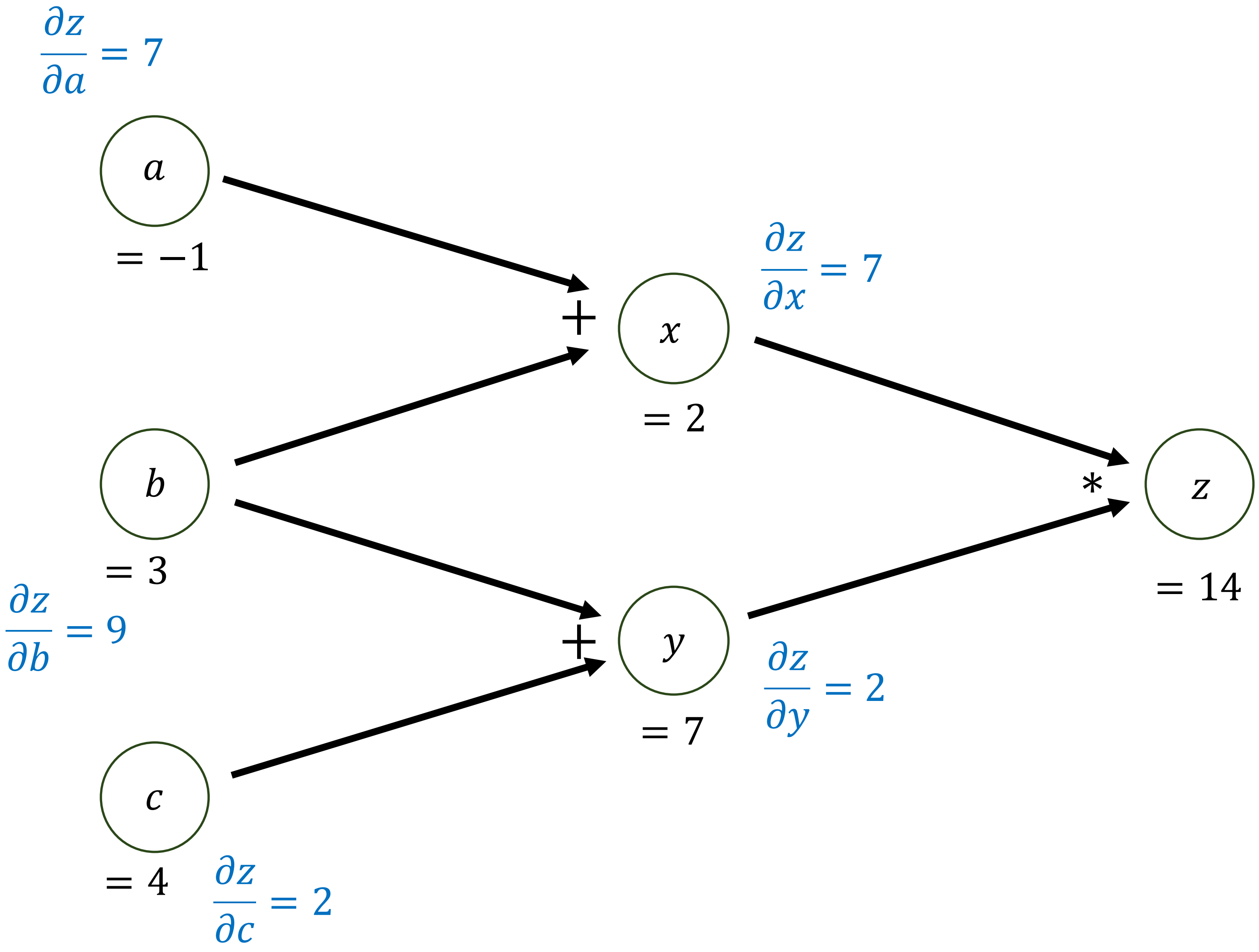
$a = -1 \qquad b = 3 \qquad c = 4$



Backward Pass

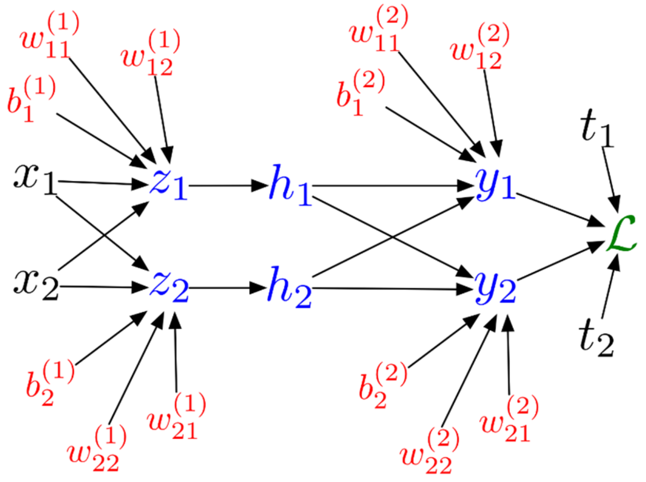
$$z = \underbrace{(a+b)}_{x} * \underbrace{(b+c)}_{y}$$

$a = -1 \qquad b = 3 \qquad c = 4$



Summary

- Computation Graphs
  - Directed and acyclic
  - Nodes: variables
  - Edges: computation
  - Enable calc. of gradients



Roger Grosse: CSC 311 Spring 2020: Introduction to Machine Learning

- Backpropagation
  - Efficient computation of gradients
  - Forward pass to compute values
  - Backward pass to compute gradients
    - Use successor results