Optimizing neural nets: SGD & Backpropagation

Recap: Neural networks

- Consider the case of **supervised learning** with neural nets
- We are performing the usual optimization

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) =: \min_{\theta} \; L(\theta)$$

  - Predictor is the neural network

$$f_\theta(\mathbf{x}) = \mathbf{W}_L \sigma(\mathbf{W}_{L-1} \sigma(\cdots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_{L-1}) + \mathbf{b}_L$$

  - Parameters are weights & biases of each layer

$$\theta = \{(\mathbf{W}_l, \mathbf{b}_l)\}_{l=1}^{L}$$

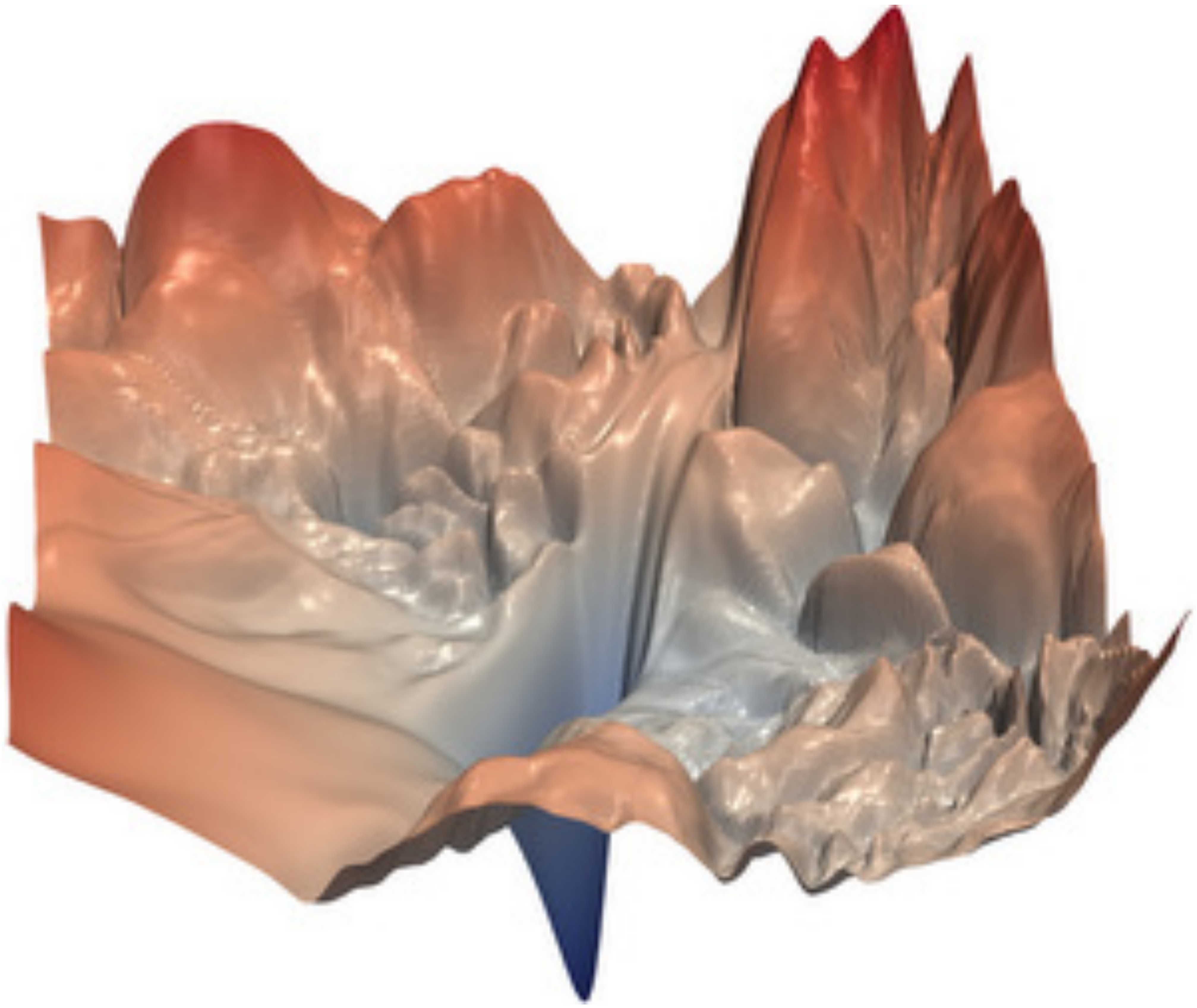
Today

- We focus on: How do we solve the **optimization** proble

$$\min_{\theta} L(\theta), \qquad f_\theta(\mathbf{x}) = \mathbf{W}_L \sigma(\cdots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_L$$

- This is very difficult
  - **Critical point.** Too complicated
  - **Convexity.** Does not hold

- The loss landscape looks like —>



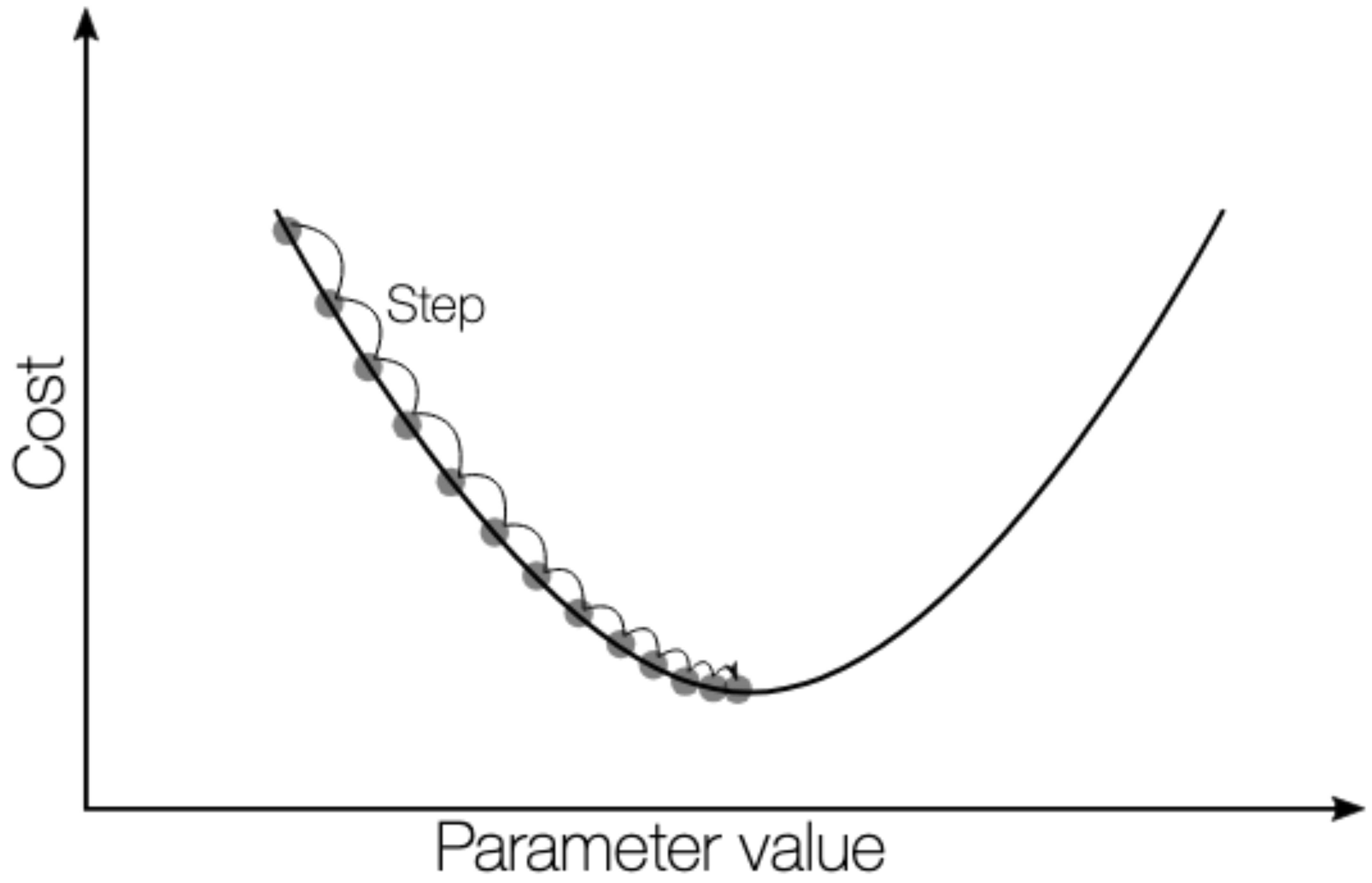
Gradient Descent

- **Solution.** Gradient Descent
  - Iteratively update θ in a direction that the loss decreases the fastest

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta L(\theta)$$

Step size (a.k.a., learning rate)

Direction of fastest increase



Gradient Descent

- Note that the gradient is the average of per-sample loss gradients:

$$\nabla_\theta L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(y_i, f_\theta(\mathbf{x}_i))$$

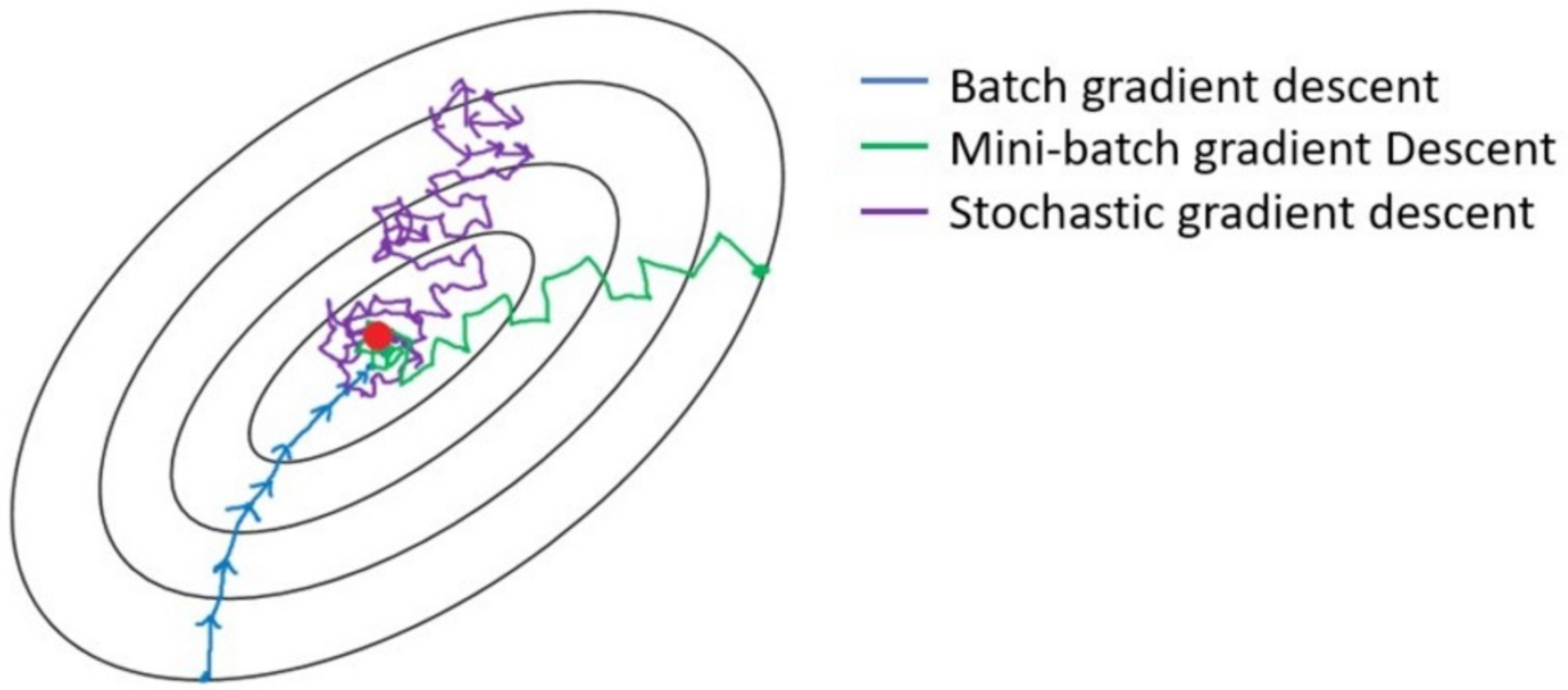
- **Problem.** Datasets for deep learning involves million—trillion-scale data
    - Examples.
        - ImageNet (Image). 1 million samples
        - Common Crawl (Text). 410 billion tokens
    - Thus, computing gradient of all data at each GD step is expensive

Gradient Descent

- **Solution.** Stochastic Gradient Descent (broad)
  - Use gradients of only a few, randomly drawn samples at each step
  - Mini-batch GD. Draw a batch $\mathscr{B}$ of samples and compute

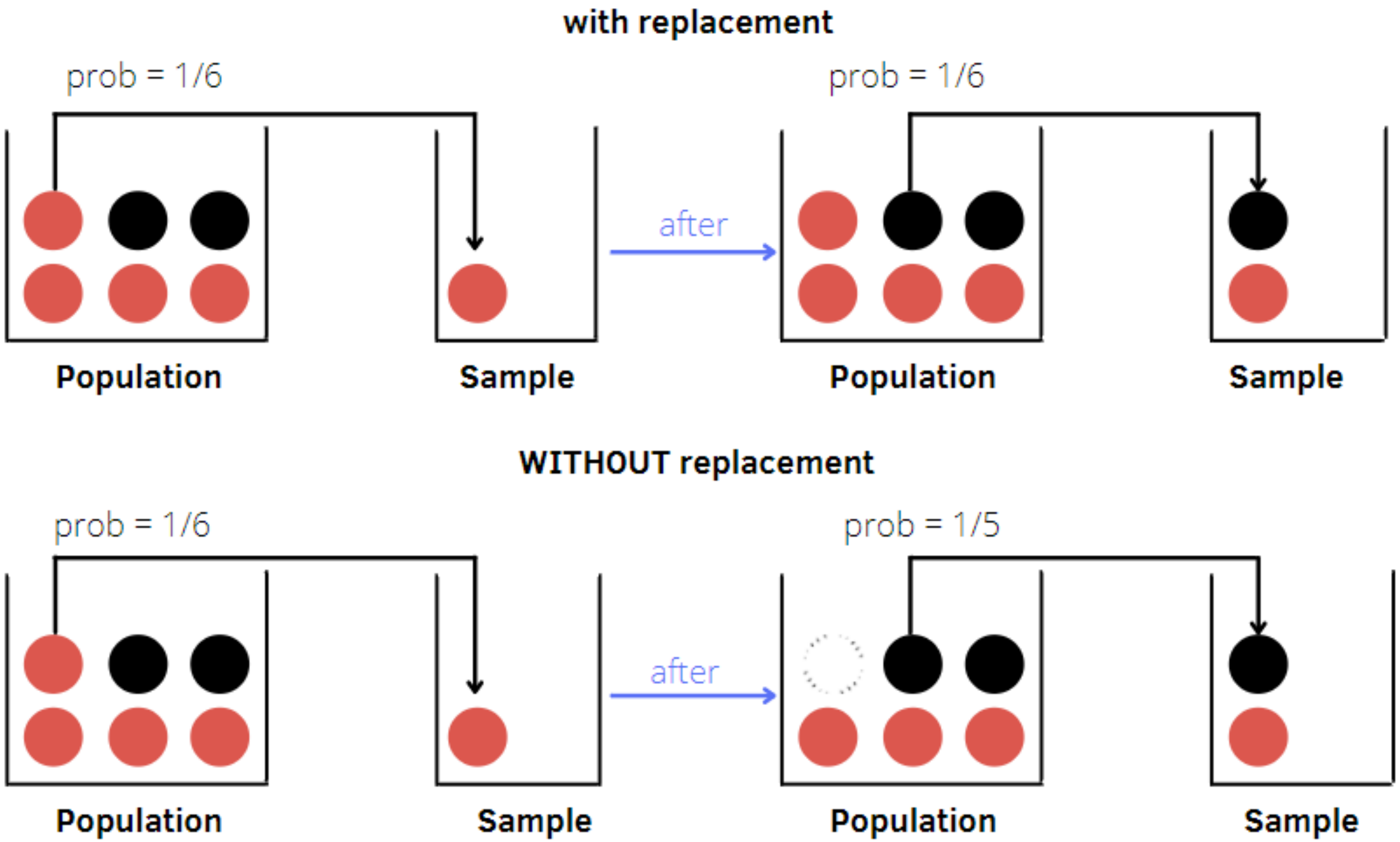
$$\hat{\nabla}_\theta L(\theta) = \frac{1}{|\mathscr{B}|} \sum_{i \in \mathscr{B}} \nabla_\theta \ell(y_i, f_\theta(\mathbf{x}_i))$$

  - SGD (narrow). Mini-batch GD with a single example



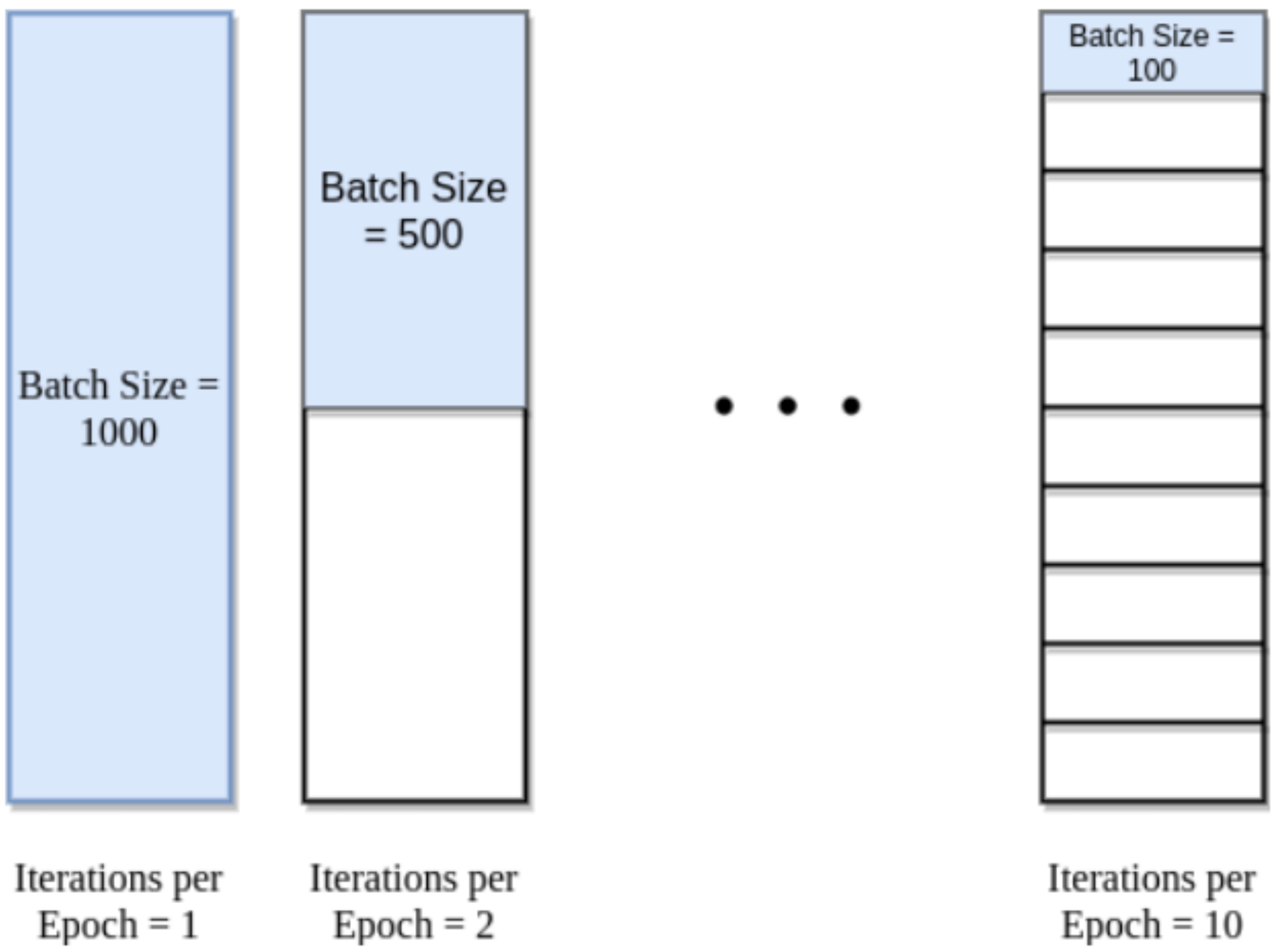
Gradient Descent

- Typically, we draw samples without replacement
  - i.e., never use a sample twice unless no sample has been never used



Gradient Descent

- **Epoch.** A set of iterations until every sample has been used once
  - Example. If we use the batch size of 64 for a dataset of size 32,000, we need 500 steps for a single epoch
  - Batch size and learning rate are key hyperparameters of SGD



Computing per-sample Gradients

Computing Gradients

- The sample-wise loss gradient is a product of
(1) the derivative of the loss function, and
(2) the gradient w.r.t. the predictor

$$\nabla_\theta\Big(\ell(y, f_\theta(\mathbf{x}))\Big) = \frac{\partial \ell(y,z)}{\partial z}(f_\theta(\mathbf{x})) \cdot \nabla_\theta f_\theta(\mathbf{x})$$

loss derivative, evaluated at prediction $f_\theta(\mathbf{x})$

Predictor gradient

- **Why?** Recall the chain rule:

$$\frac{\partial}{\partial x} g(f(x)) = g'(f(x)) \cdot f'(x)$$

Computing Gradients

$$\nabla_\theta \left(\ell(y, f_\theta(\mathbf{x})) \right) = \boxed{\frac{\partial \ell(y,z)}{\partial z}(f_\theta(\mathbf{x}))} \cdot \nabla_\theta f_\theta(\mathbf{x})$$

- The **loss derivative** is typically easy to compute

- **Example.** For squared loss $\ell(y,z) = (y-z)^2$, the loss derivative will be:

$$2(y - f_\theta(\mathbf{x}))$$

  - Simply do (1) pass the data through the predictor
    (2) measure the error
    (3) multiply 2

Computing Gradients

$$\nabla_\theta \Big(\ell(y, f_\theta(\mathbf{x})) \Big) = \frac{\partial \ell(y,z)}{\partial z}(f_\theta(\mathbf{x})) \cdot \boxed{\nabla_\theta f_\theta(\mathbf{x})}$$

- The **predictor gradient** is much tricker to compute
  - The parameter θ is high-dimensional

$$\nabla_\theta g(\theta) = \left[\frac{\partial}{\partial \theta_1} g(\theta), \ldots, \frac{\partial}{\partial \theta_d} g(\theta) \right]$$

  - How do we compute this, for a very complicated function like...?

$$g(\theta) = \mathbf{W}_L \sigma(\cdots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_L$$

Computing Gradients: Numerical Method

$$\nabla_\theta g(\theta) = \left[\frac{\partial}{\partial \theta_1} g(\theta), \ldots, \frac{\partial}{\partial \theta_d} g(\theta)\right]$$

- One possible way is the **numerical method**
  - Note that

$$\frac{\partial}{\partial x} g(x) = \lim_{\epsilon \to 0} \frac{g(x+\epsilon) - g(x)}{\epsilon}$$

  - Make a very small perturbation on the current parameter
    - Do this for the first entry θ_1
    - Do this for the second entry θ_2
    - …

Computing Gradients: Numerical Method

current W:	W + h (first dim):	gradient dW:
[0.34,	[0.34 + 0.0001,	[-2.5,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	loss 1.25322	

(1.25322 - 1.25347)/0.0001 = -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Computing Gradients: Numerical Method

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim):

[0.34,
-1.11 + 0.0001,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,…]

(1.25353 - 1.25347)/0.0001 = 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Computing Gradients: Numerical Method

current W:	W + h (third dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + 0.0001,	0,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	loss 1.25347	

(1.25347 - 1.25347)/0.0001 = 0

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Computing Gradients: Numerical Method

- **Pros.**
  - Easy to implement
  - Can use for black-box models
- **Cons.**
  - Only gives you approximate
    - cannot take the limit $\epsilon \to 0$, due to the finite precision
  - Very slow <—
    - Requires at least $d + 1$ model inferences

Computing Gradients: Analytic Method

- The most popular method is the **analytic method**

- **Example.** Consider the function

$$g(\theta_1, \theta_2) = \sin(5 \cdot \exp(\theta_1) + 2\cos(\theta_2))$$

  - Then, we know that the gradient will have the formula:

$$\nabla_{\theta_1} g(\theta_1, \theta_2) = 5 \cdot \cos(5 \cdot \exp(\theta_1) + 2 \cdot \cos(\theta_2)) \cdot \exp(\theta_1)$$

$$\nabla_{\theta_2} g(\theta_1, \theta_2) = -2 \cdot \cos(5 \cdot \exp(\theta_1) + 2 \cdot \cos(\theta_2)) \cdot \sin(\theta_2)$$

  - We can simply evaluate these functions

Computing Gradients: Analytic Method

- **Pros.**
  - Exact
- **Cons.**
  - Requires deriving the gradient formula for all parameters
  - Still needs computing the gradients for each parameters

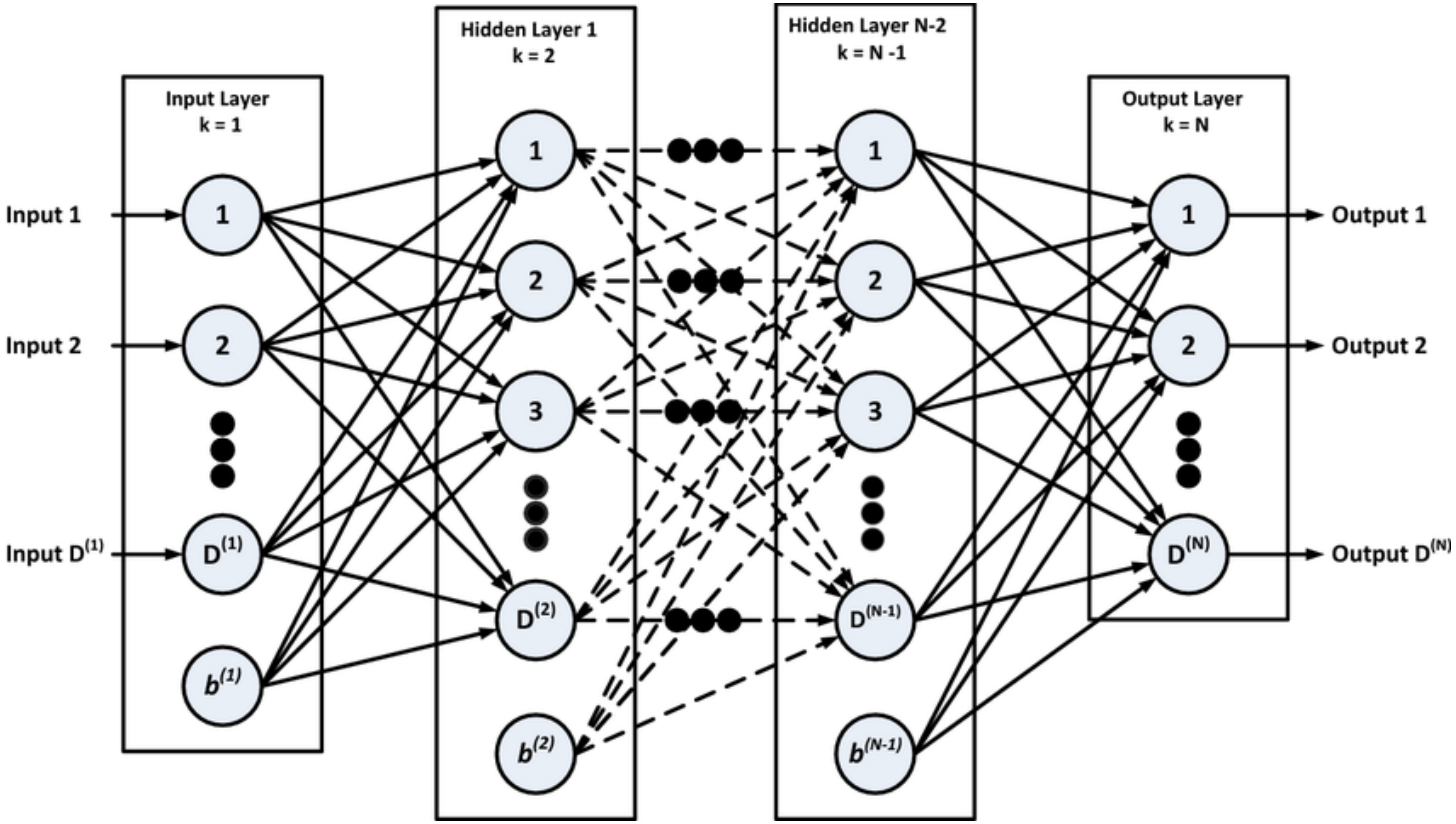
- Luckily, for neural nets, the cons become easy to solve
  - Derivation can be automatized
  - Computing the gradients can be grouped and simplified

Backpropagation

Analytic Form of Gradients

- **Question.** How do we derive an analytic form of $\nabla_\theta f_\theta(\mathbf{x})$, for…?

$$f_\theta(\mathbf{x}) = \mathbf{W}_L\sigma(\mathbf{W}_{L-1}\sigma(\cdots\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\cdots + \mathbf{b}_{L-1}) + \mathbf{b}_L$$



Analytic Form of Gradients

- **Idea.** View this as a composition of elementary operations

$$f_\theta(\mathbf{x}) = f_{\mathbf{b}_L} \circ f_{\mathbf{W}_L} \circ f_{\sigma_L} \circ \cdots \circ f_{\mathbf{W}_1}(\mathbf{x})$$

  - $f_{\mathbf{W}_i}(\mathbf{x}) = \mathbf{W}_i\mathbf{x}$
  - $f_{\mathbf{b}_i}(\mathbf{x}) = \mathbf{x} + \mathbf{b}_i$
  - $f_\sigma(\mathbf{x}) = \sigma(\mathbf{x})$

- Then:
  - Derivatives of each elementary op can be hard-coded
  - Use chain rule to combine these

Example

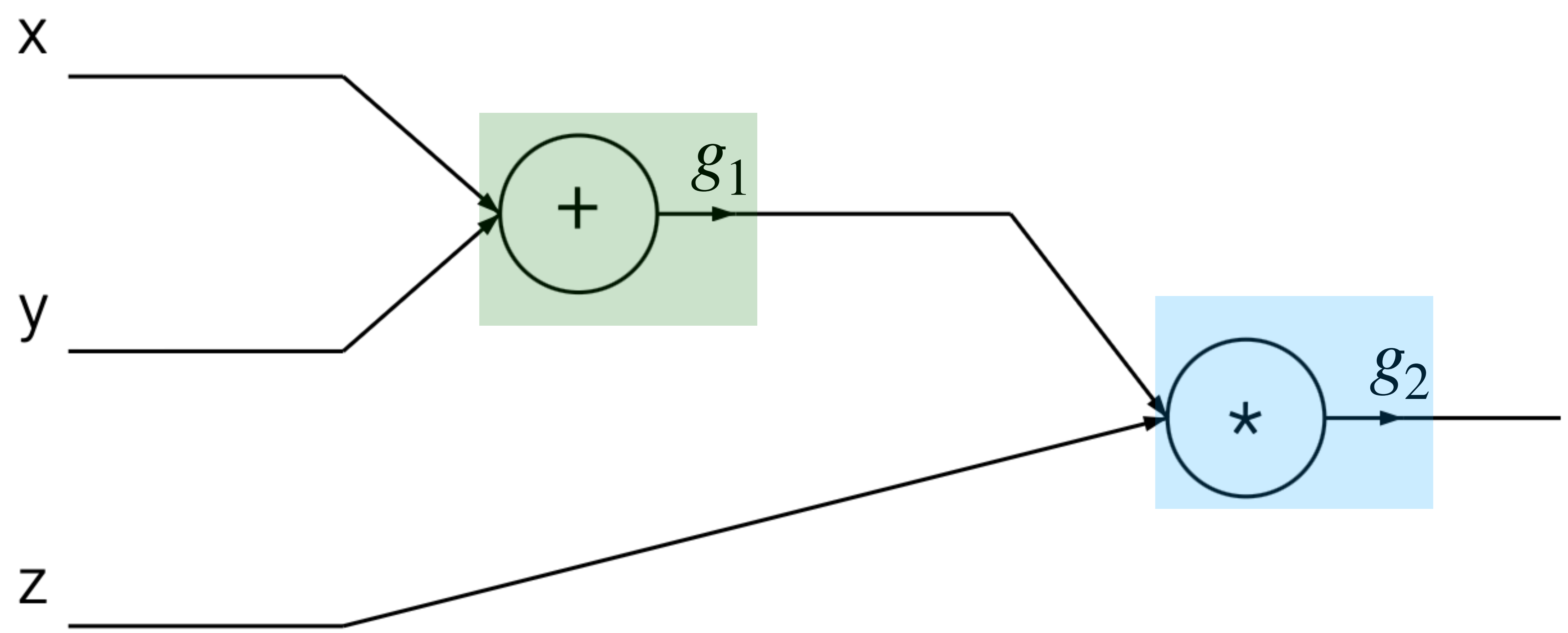
- Consider a function

$$g(x, y, z) = (x + y) \cdot z$$

- This is a composition of two elementary operations

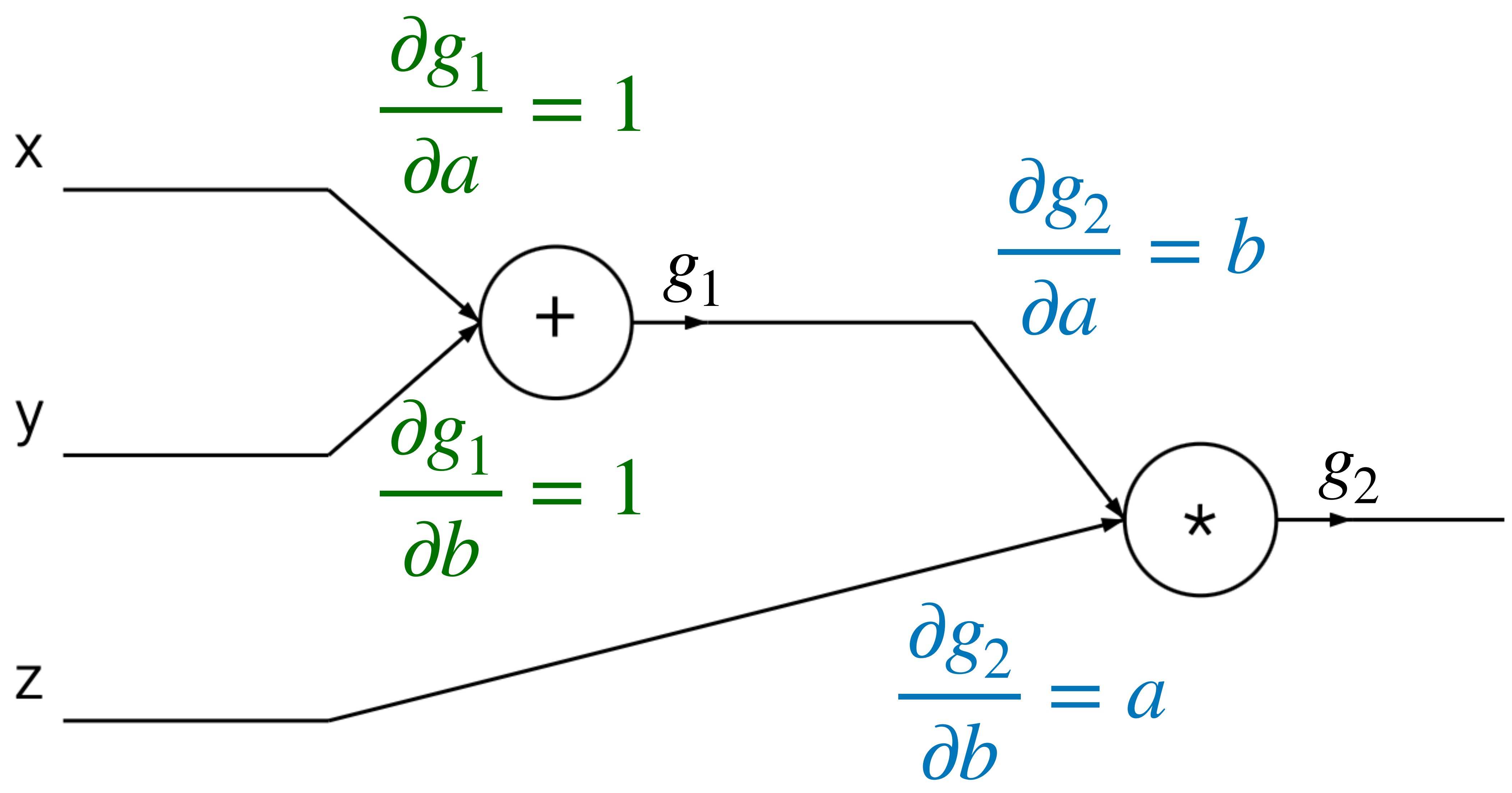
$$g(x, y, z) = g_2(g_1(x, y), z)$$

  - Addition: $g_1(a, b) = a + b$
  - Multiplication: $g_2(a, b) = ab$



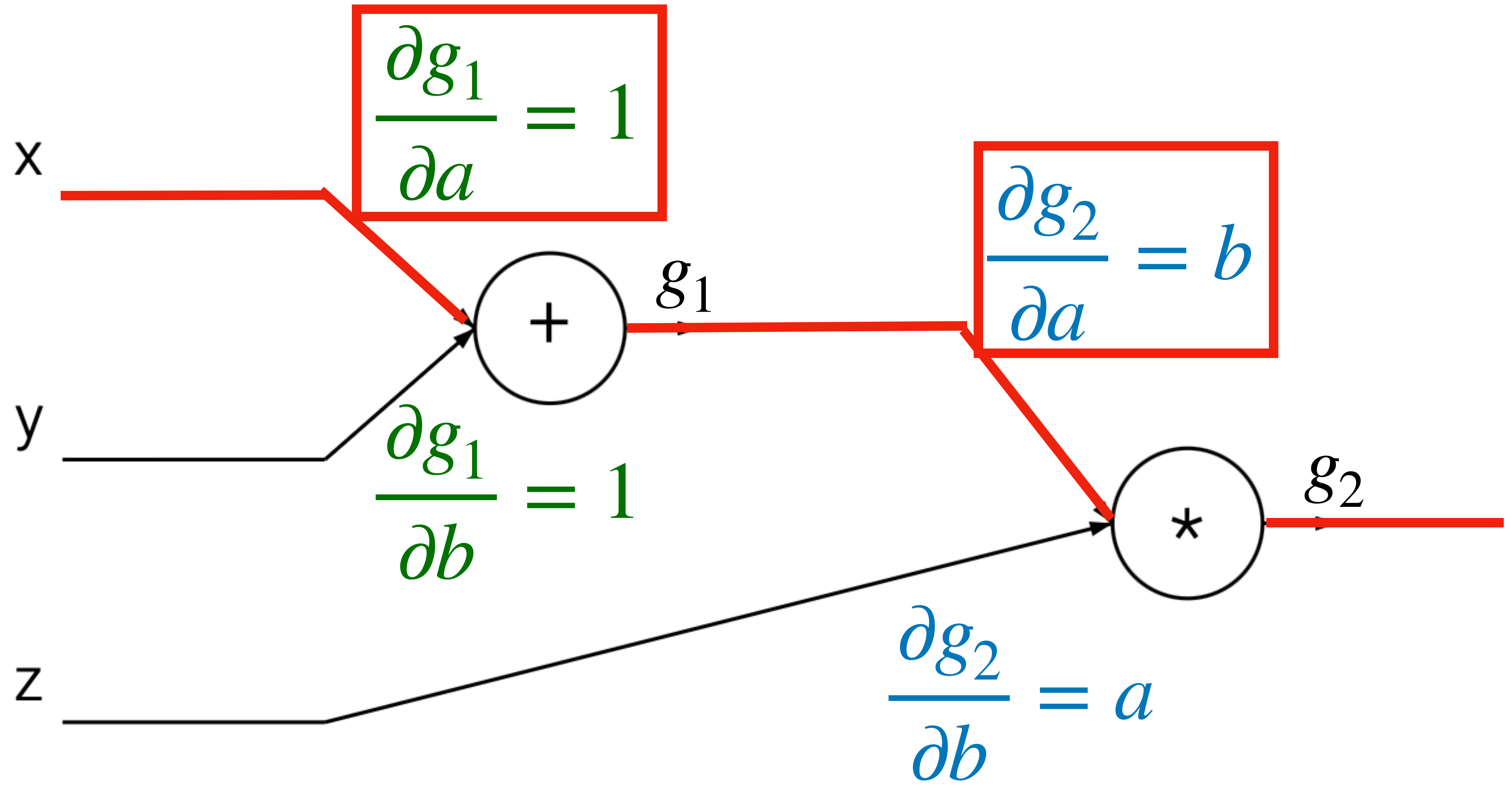
Example

- Each elementary operation has an easy-to-write gradient
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$



Example

- Each elementary operation has an easy-to-write gradient
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$

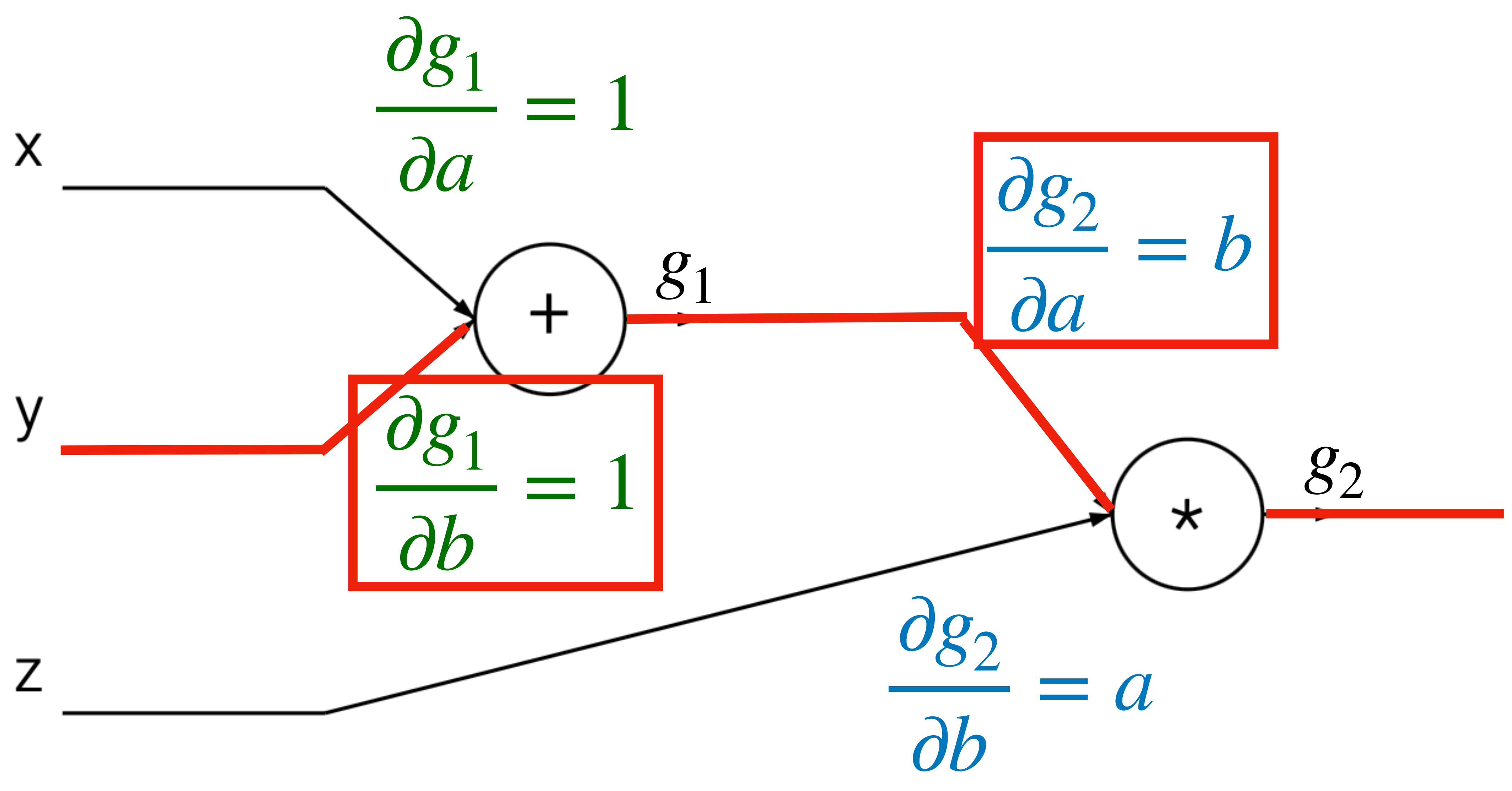


- Chain rule tells you that:

$$\frac{\partial g}{\partial x}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial a}(g_1(x, y), z)}_{= z} \cdot \underbrace{\frac{\partial g_1}{\partial a}(x, y)}_{= 1}$$

Example

- Each elementary operation has an easy-to-write gradient
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$



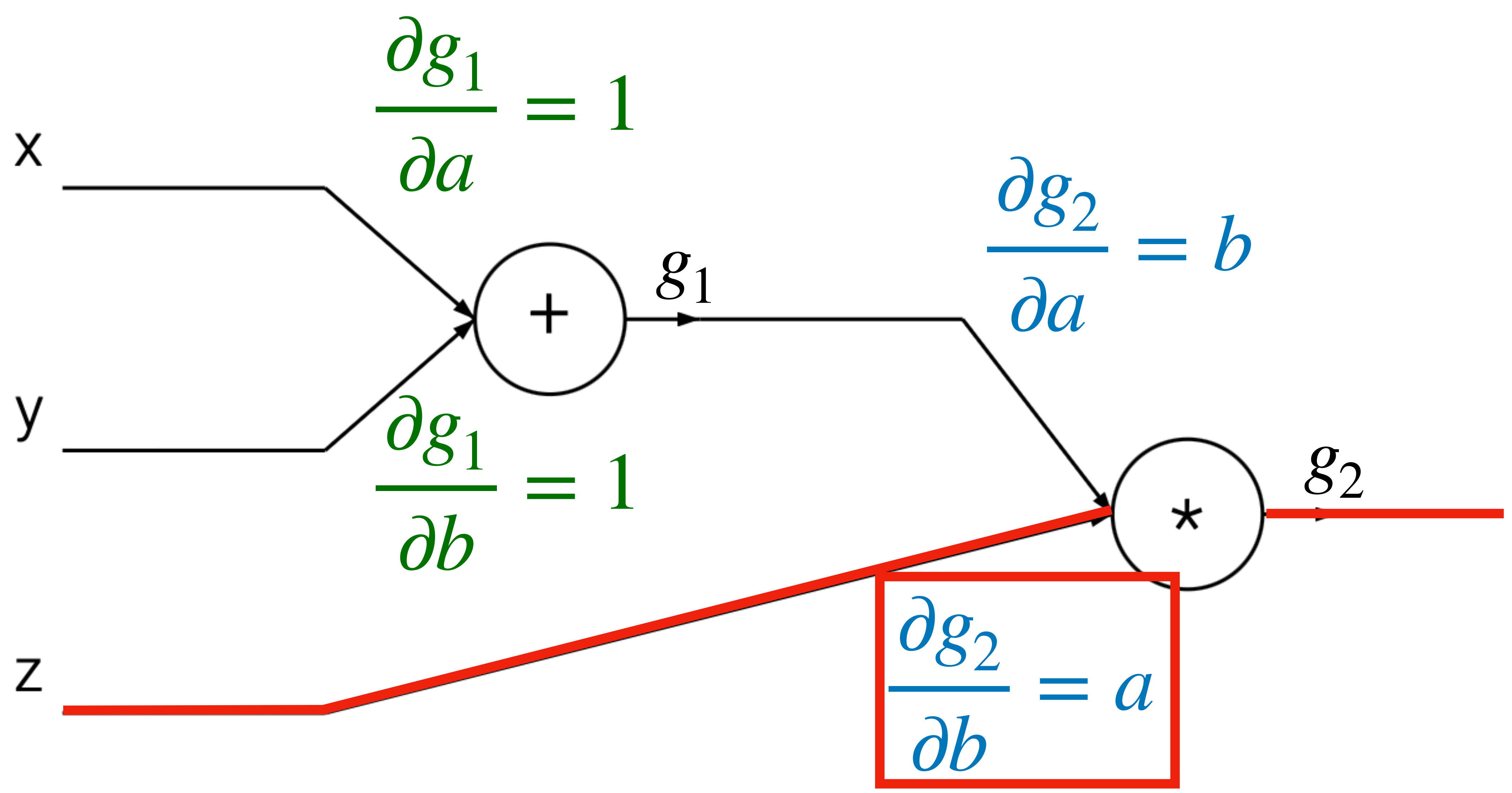
- Chain rule tells you that:

$$\frac{\partial g}{\partial x}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial a}(x, y)$$

$$\frac{\partial g}{\partial y}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial a}(g_1(x, y), z)}_{= z} \cdot \underbrace{\frac{\partial g_1}{\partial b}(x, y)}_{= 1}$$

Example

- Each elementary operation has an easy-to-write gradient
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$



- Chain rule tells you that:

$$\frac{\partial g}{\partial x}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial a}(x, y)$$

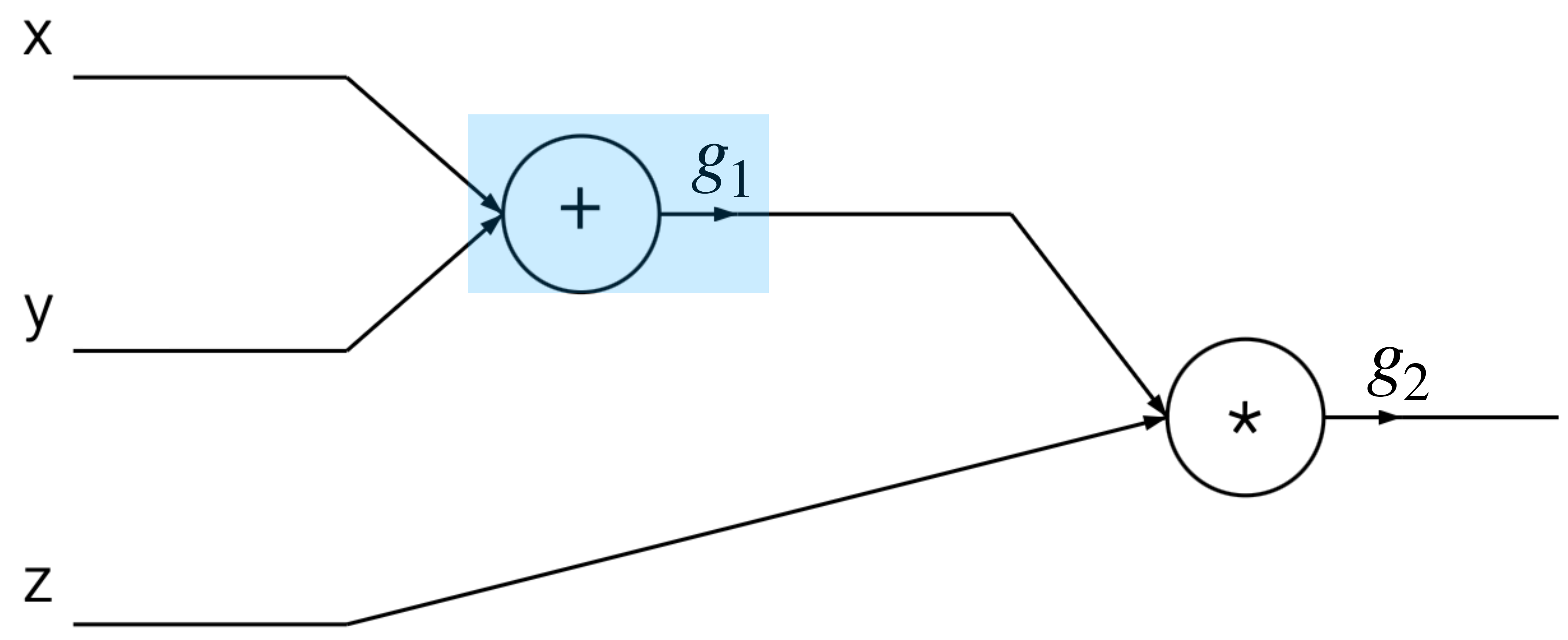
$$\frac{\partial g}{\partial y}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial b}(x, y) \quad \frac{\partial g}{\partial z}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial b}(g_1(x, y), z)}_{= g_1(x, y)}$$

Example

$$\frac{\partial g}{\partial x}(x,y,z) = \frac{\partial g_2}{\partial a}(g_1(x,y),z) \cdot \frac{\partial g_1}{\partial a}(x,y)$$

$$\frac{\partial g}{\partial y}(x,y,z) = \frac{\partial g_2}{\partial a}(g_1(x,y),z) \cdot \frac{\partial g_1}{\partial b}(x,y) \quad \frac{\partial g}{\partial z}(x,y,z) = \frac{\partial g_2}{\partial b}(g_1(x,y),z)$$

- **Observation 1.** Computing gradients involves intermediate states of the composite function
  - **Idea.** Compute all intermediate states and store them. Later, we can combine these intermediate values

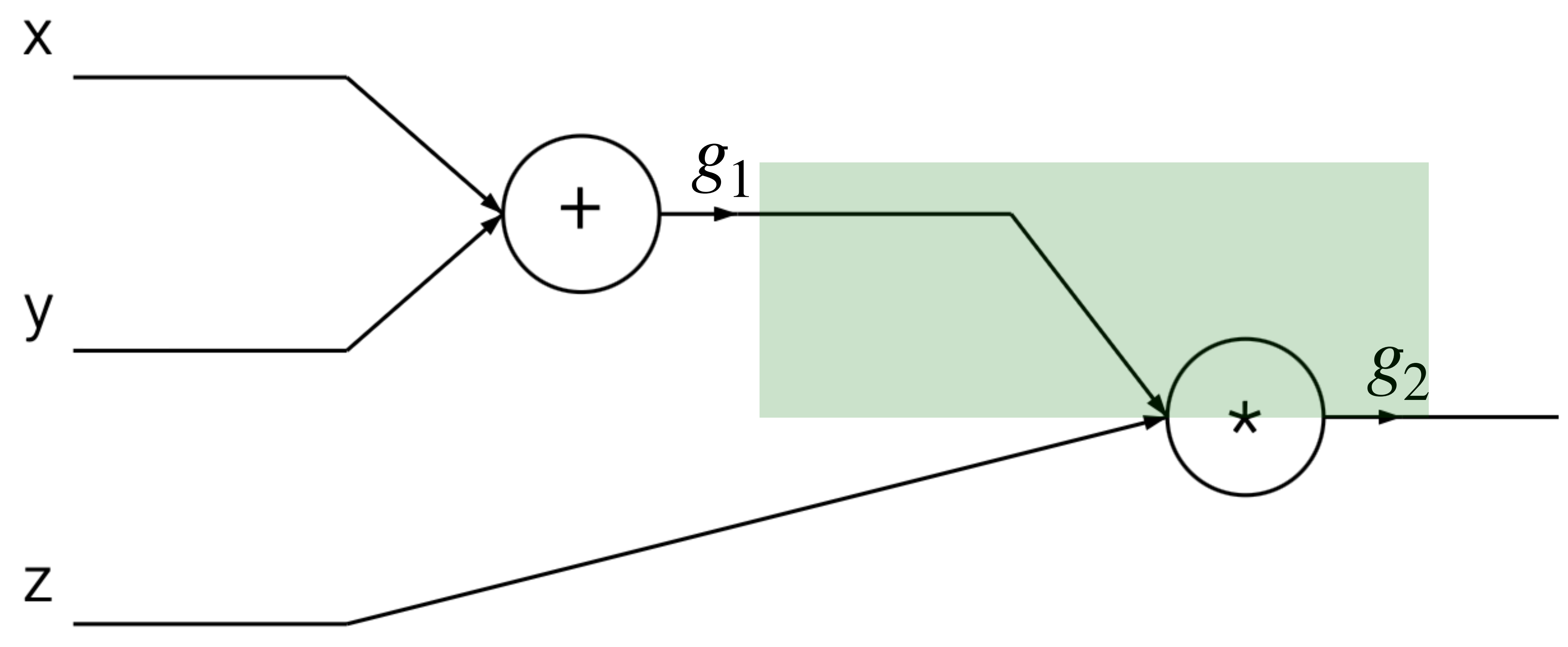


Example

$$\frac{\partial g}{\partial x}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial a}(x, y)$$

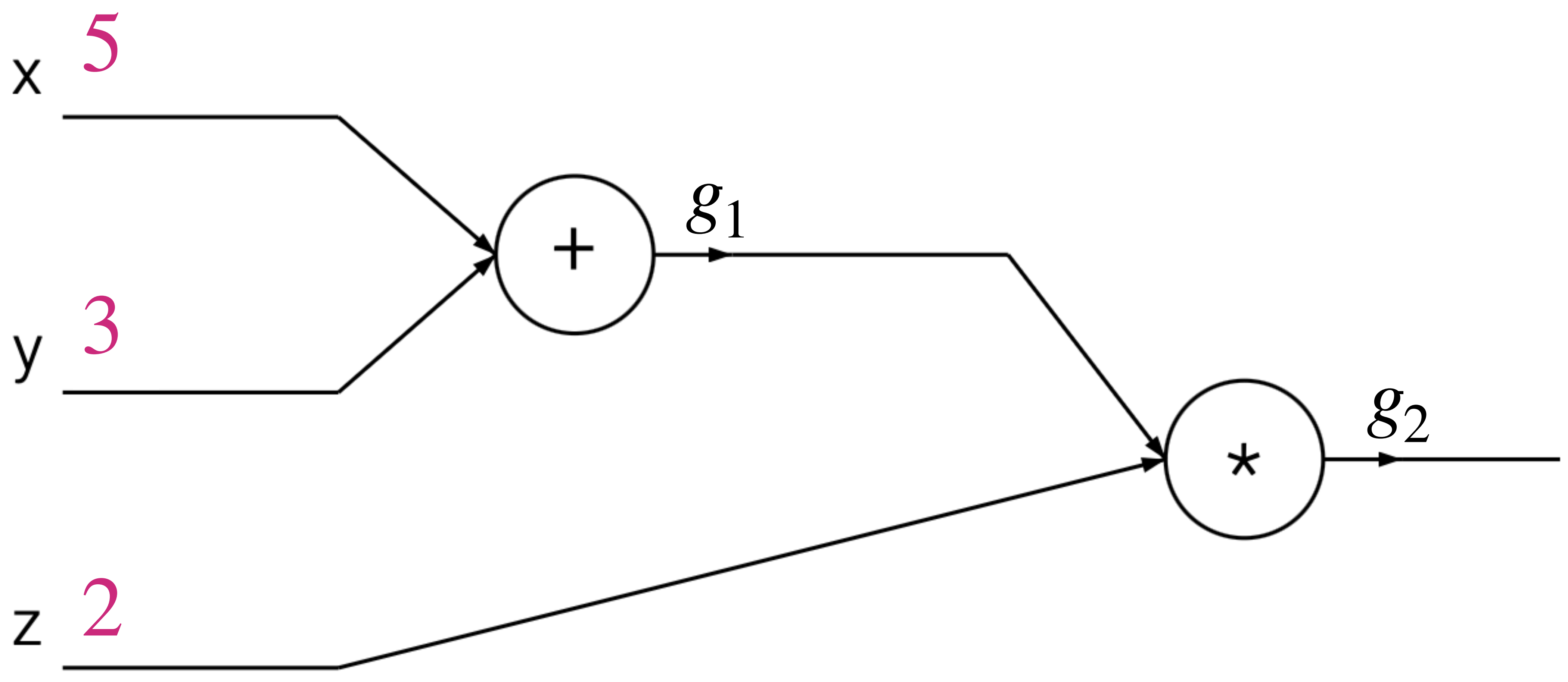
$$\frac{\partial g}{\partial y}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial b}(x, y) \quad \frac{\partial g}{\partial z}(x, y, z) = \frac{\partial g_2}{\partial b}(g_1(x, y), z)$$

- **Observation 2.** The computed gradients themselves can be reused
  - In particular, the gradient of the later block is used for computing the gradients of earlier-block parameters



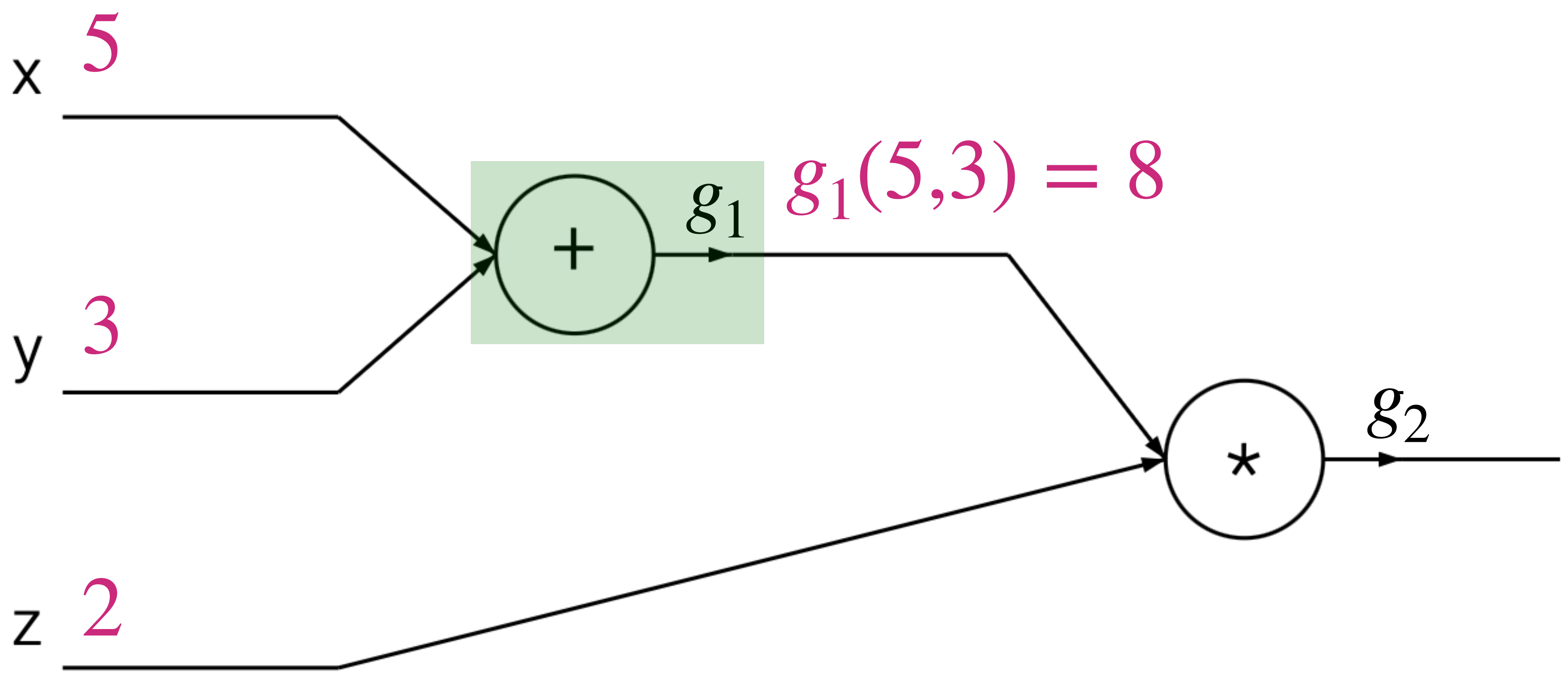
Example: Backpropagation

- Inspired by these, we can think of a three-step algorithm
- **1. Forward Pass.** Compute the output, storing all intermediate states in the memory
    - From input to output



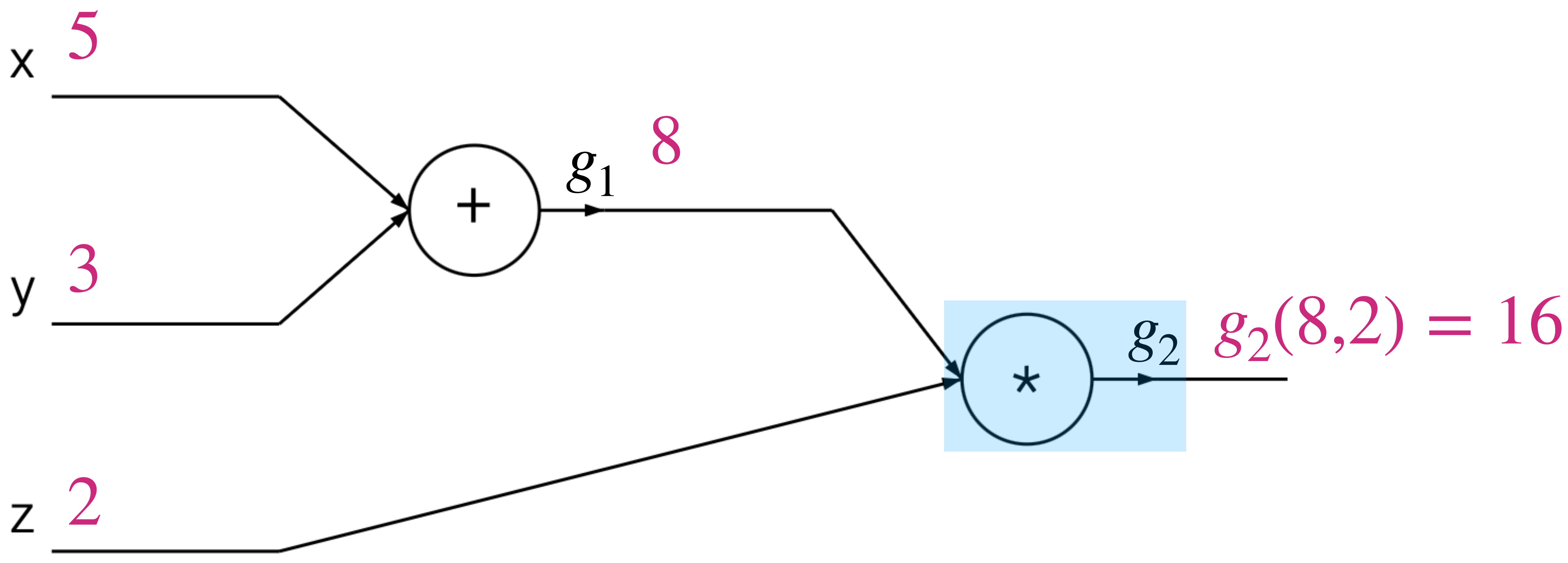
Example: Backpropagation

- Inspired by these, we can think of a three-step algorithm
- **1. Forward Pass.** Compute the output, storing all intermediate states in the memory
  - From input to output



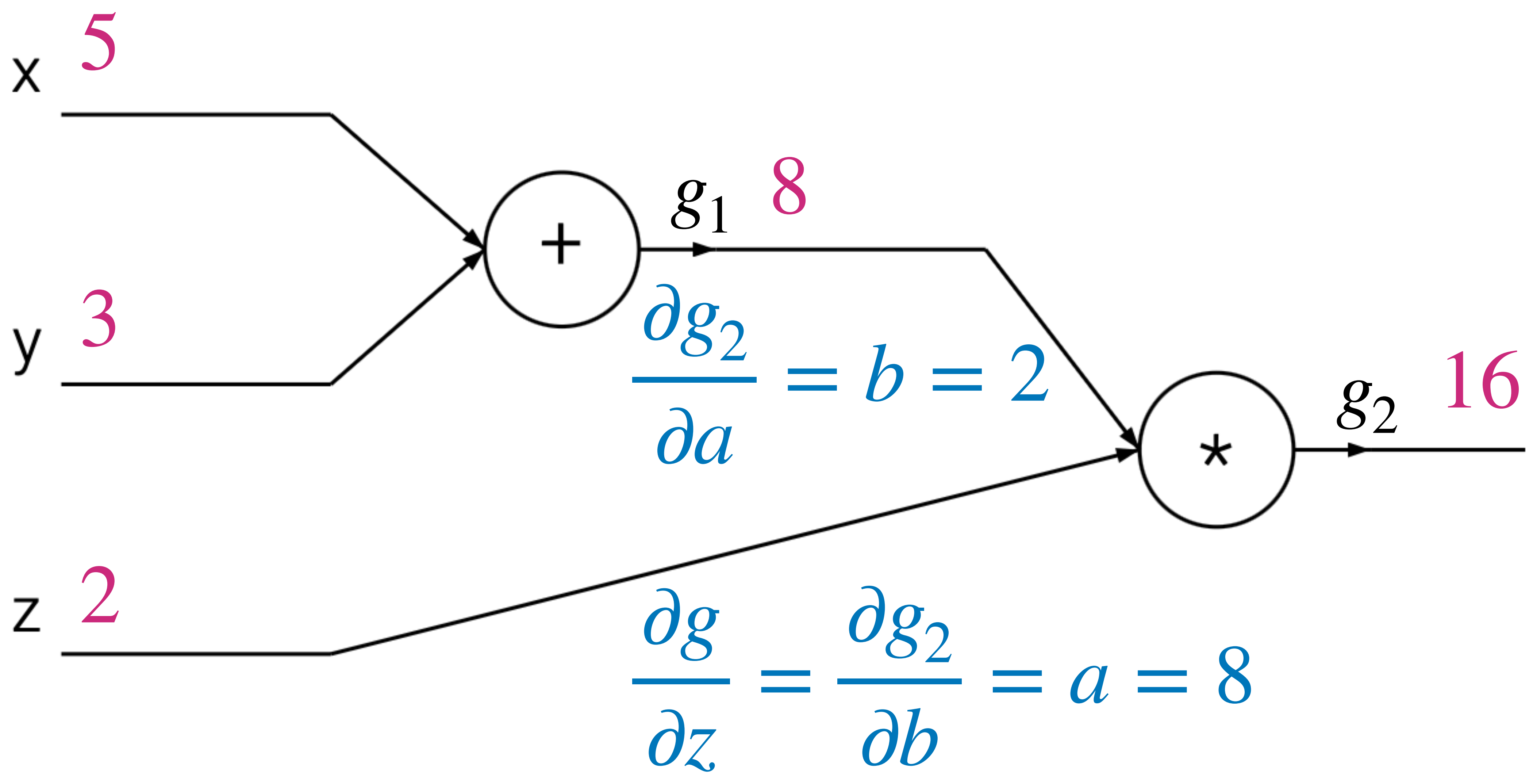
Example: Backpropagation

- Inspired by these, we can think of a three-step algorithm
- **1. Forward Pass.** Compute the output, storing all intermediate states in the memory
  - From input to output



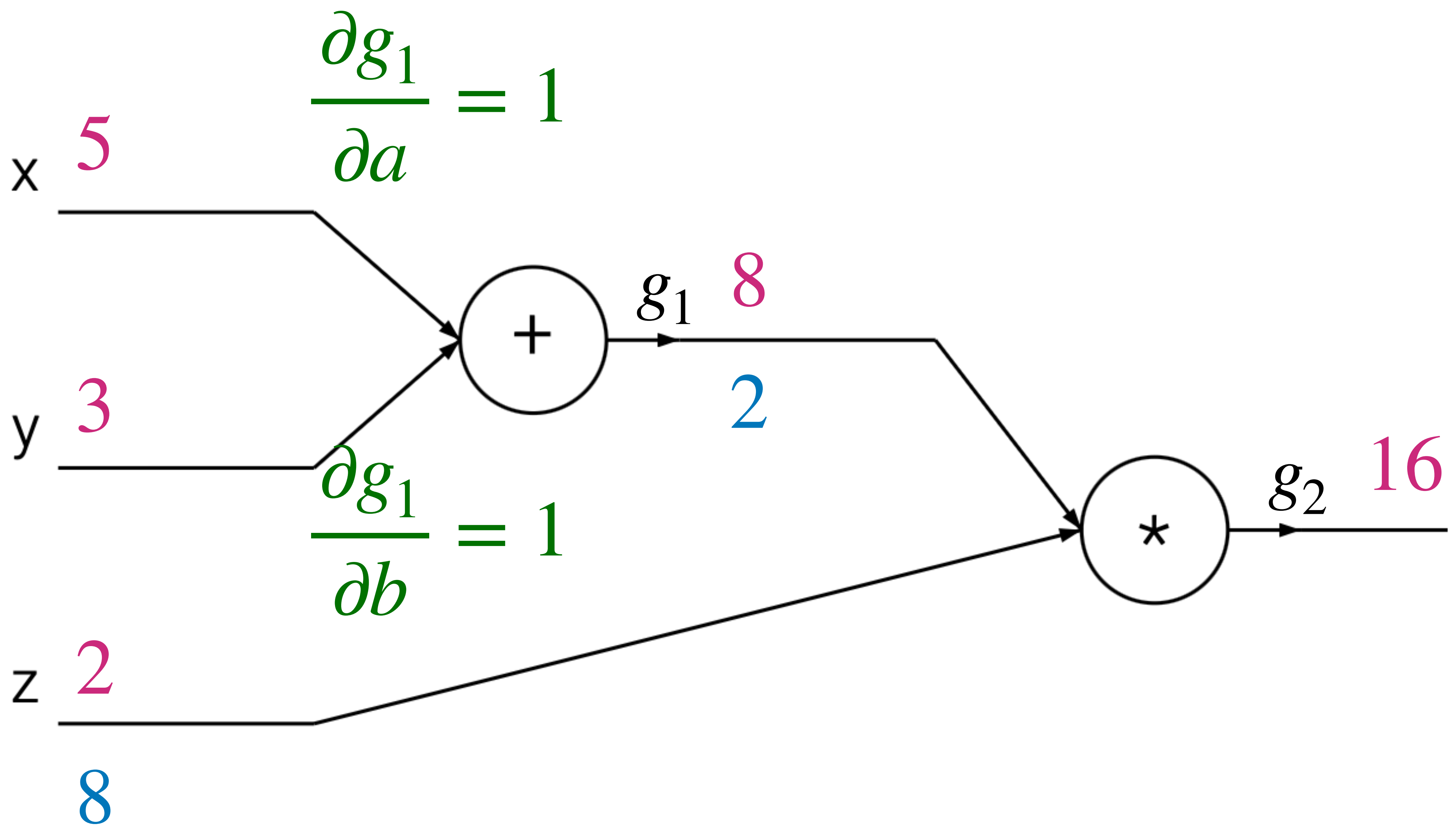
Example: Backpropagation

- **2. Backward Pass.** Compute the gradient using stored states
  - From output to input



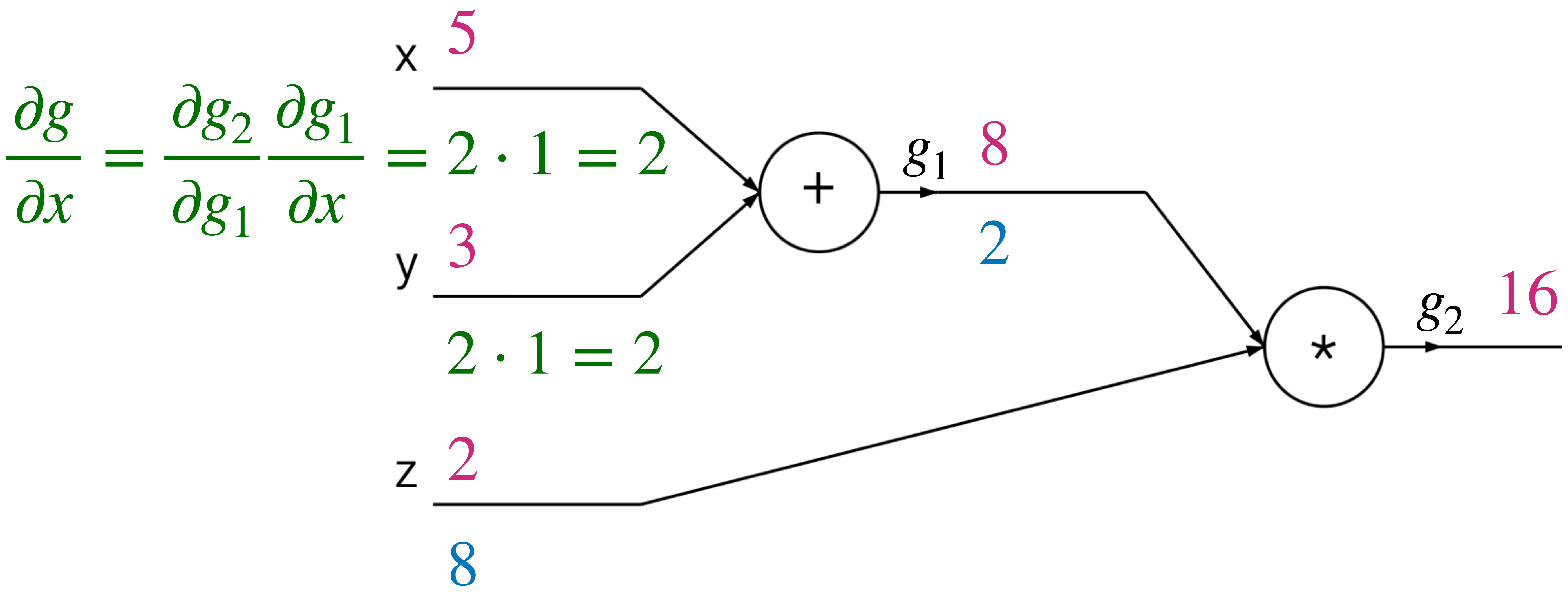
Example: Backpropagation

- **2. Backward Pass.** Compute the gradient using stored states
  - From output to input



Example: Backpropagation

- **2. Backward Pass.** Compute the gradient using stored states
    - From output to input

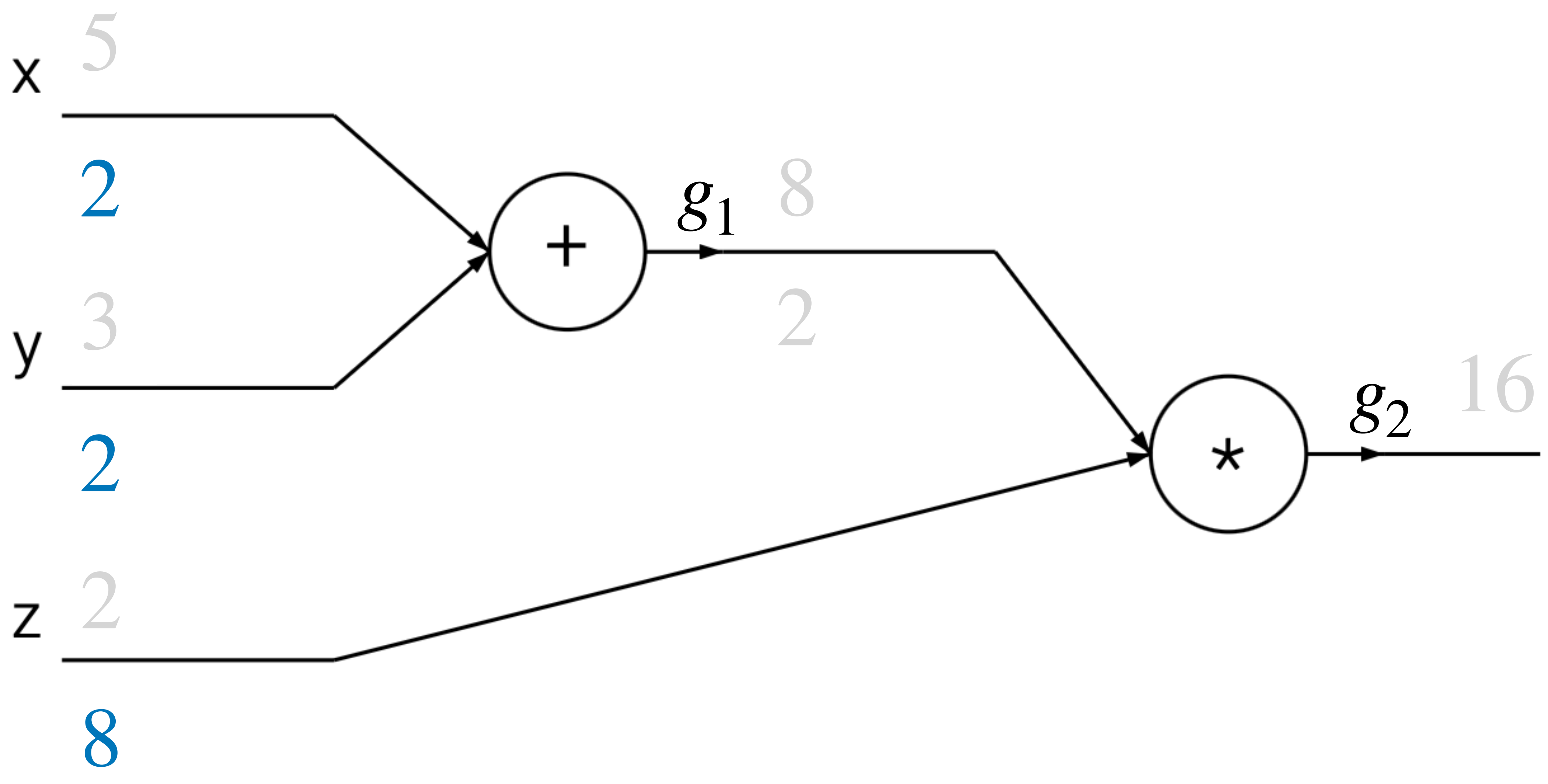


Example: Backpropagation

- **3. GD.** Update the parameters

$$x \leftarrow x - \eta \cdot 2, \qquad y \leftarrow y - \eta \cdot 2, \qquad z \leftarrow z - \eta \cdot 8$$

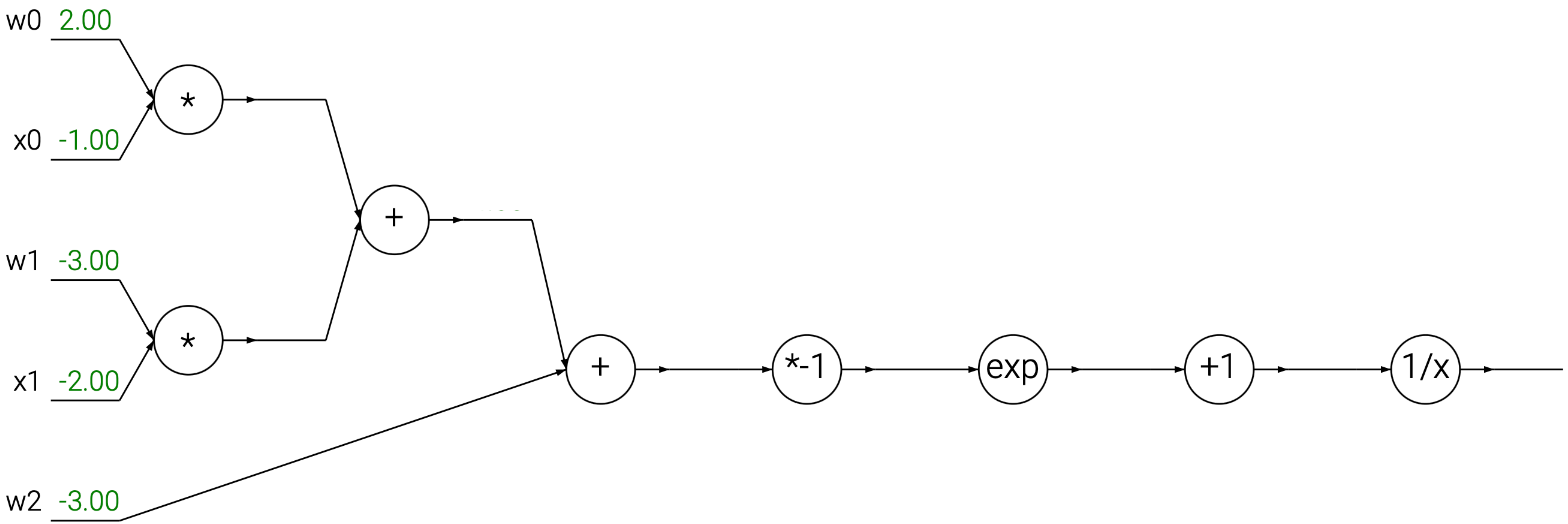
  - Then, repeat 1—3 over and over…



Another example

- Consider a function

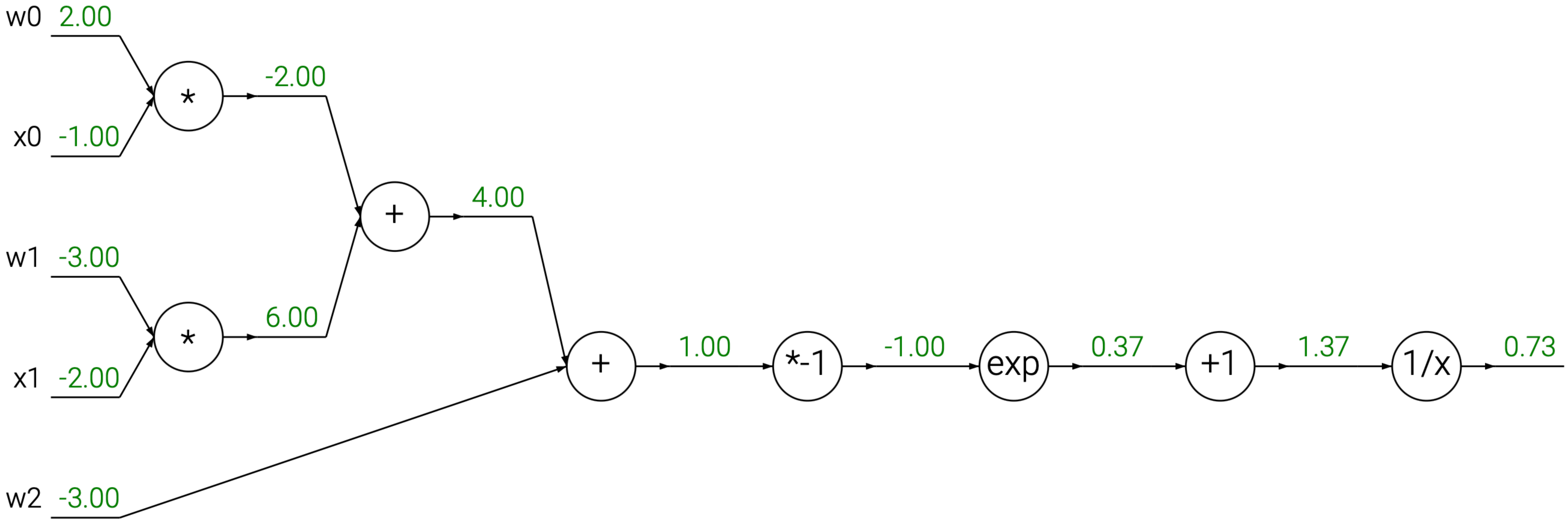
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

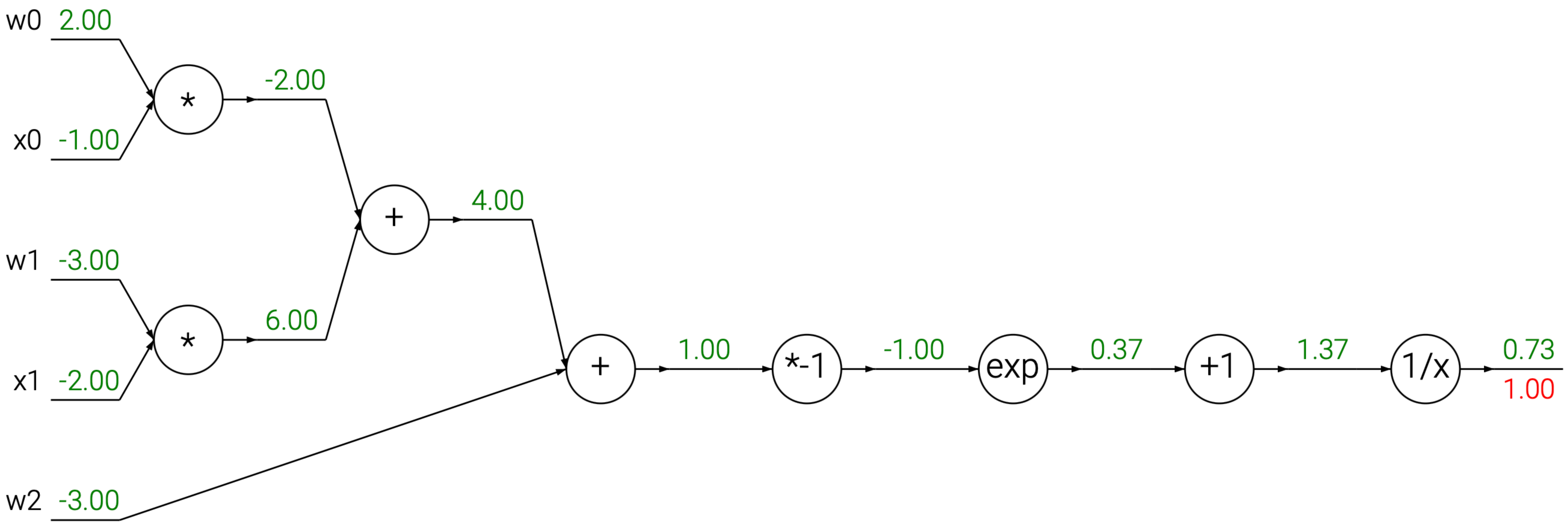
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

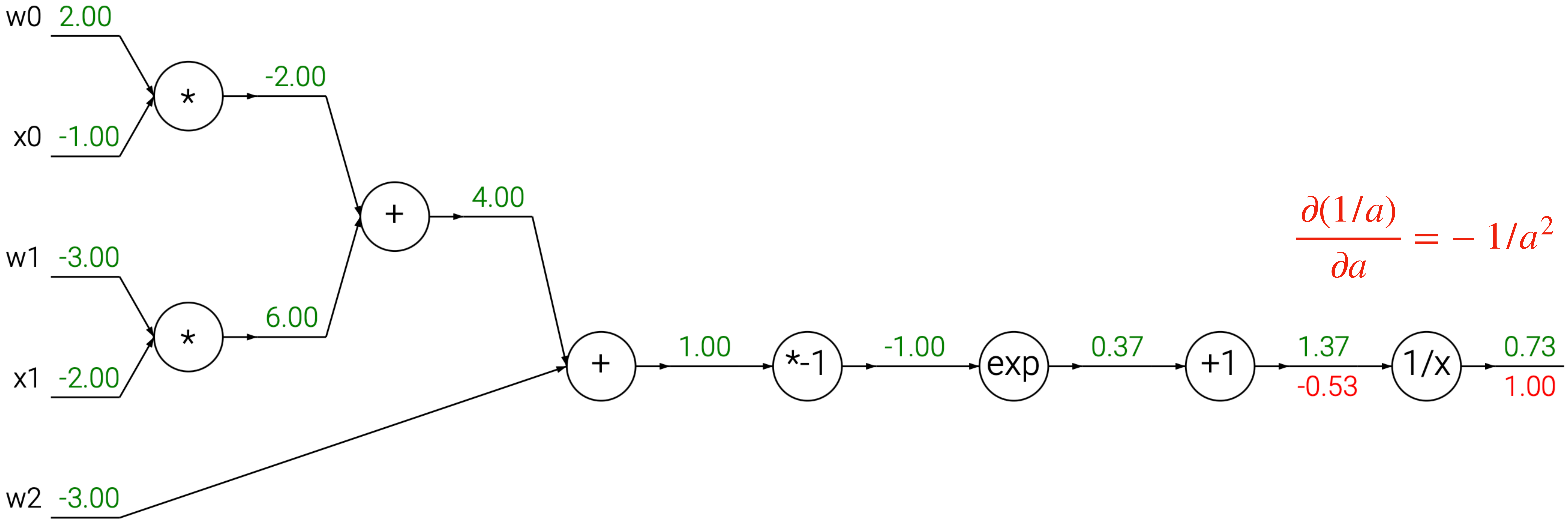
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0x_0 + w_1x_1 + w_2))}$$

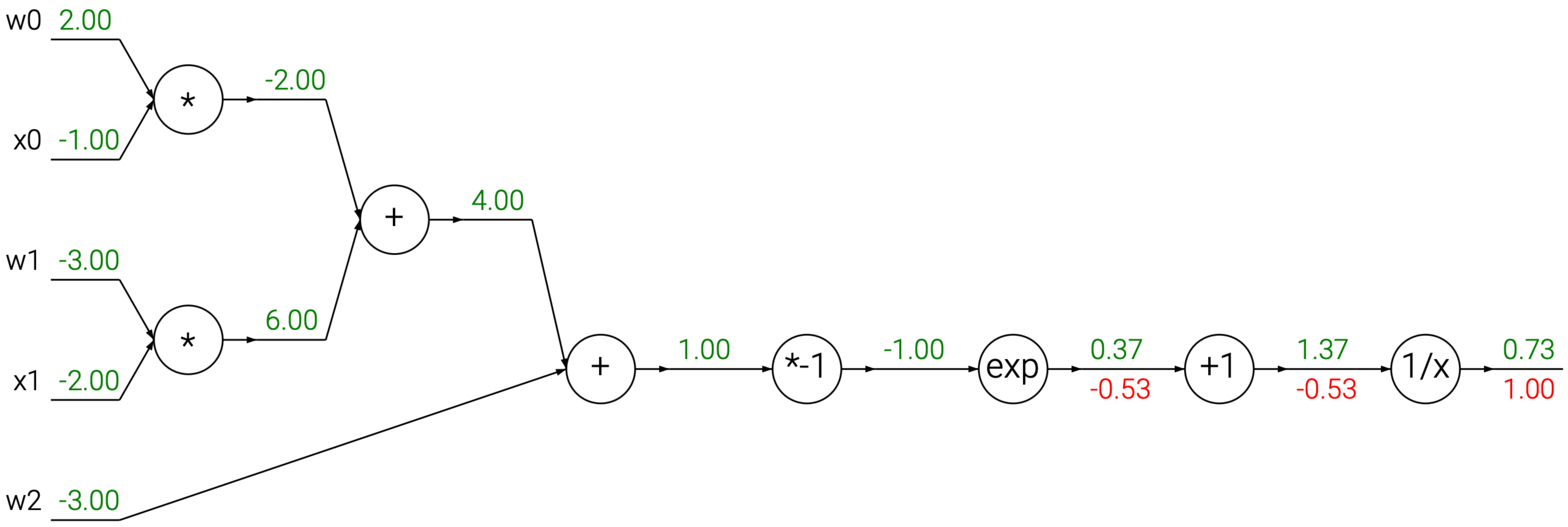


$$\frac{\partial(1/a)}{\partial a} = -1/a^2$$

Another example

- Consider a function

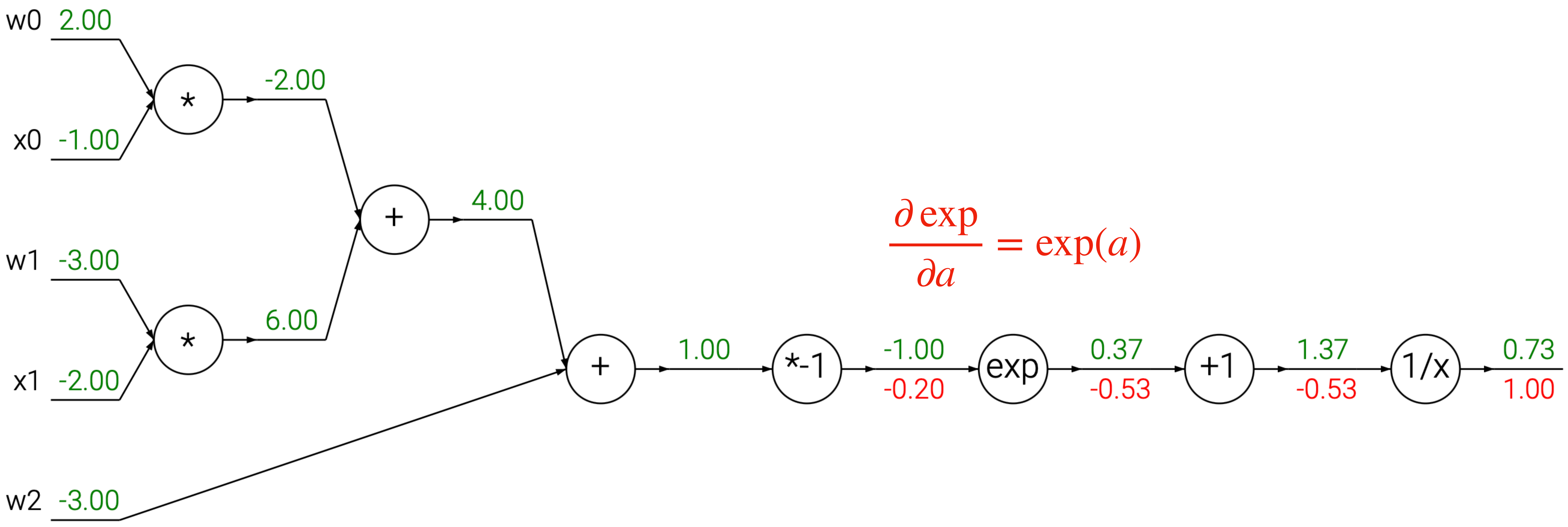
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

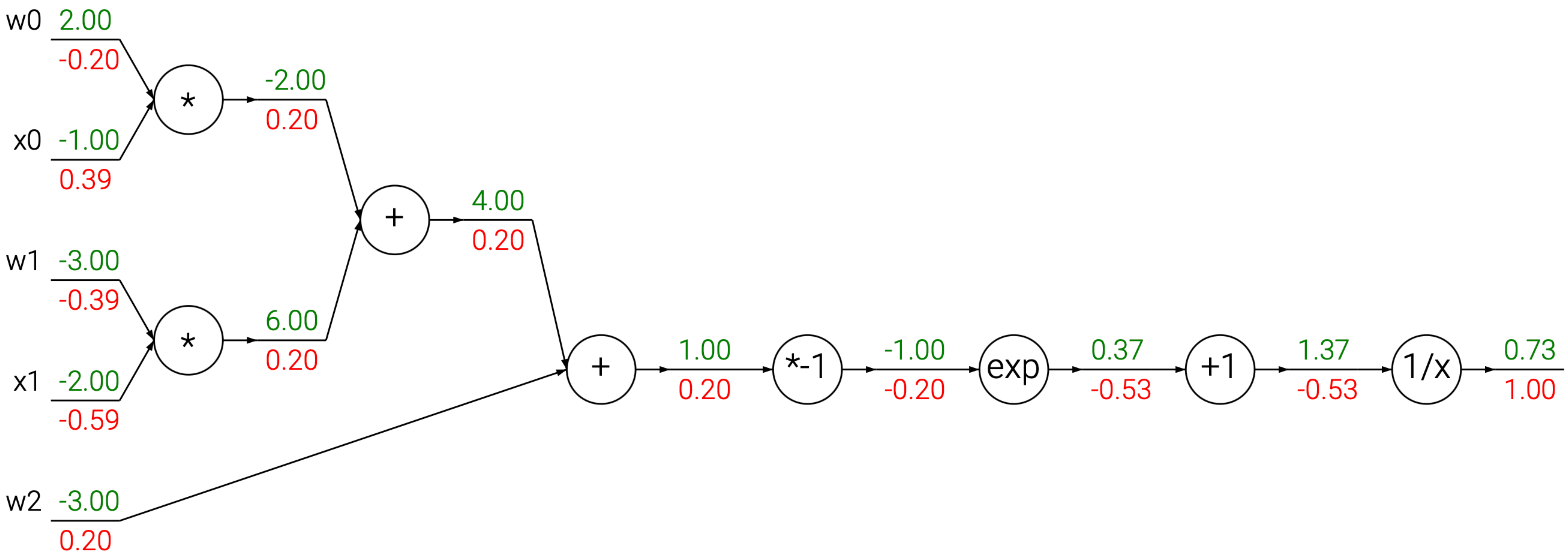
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

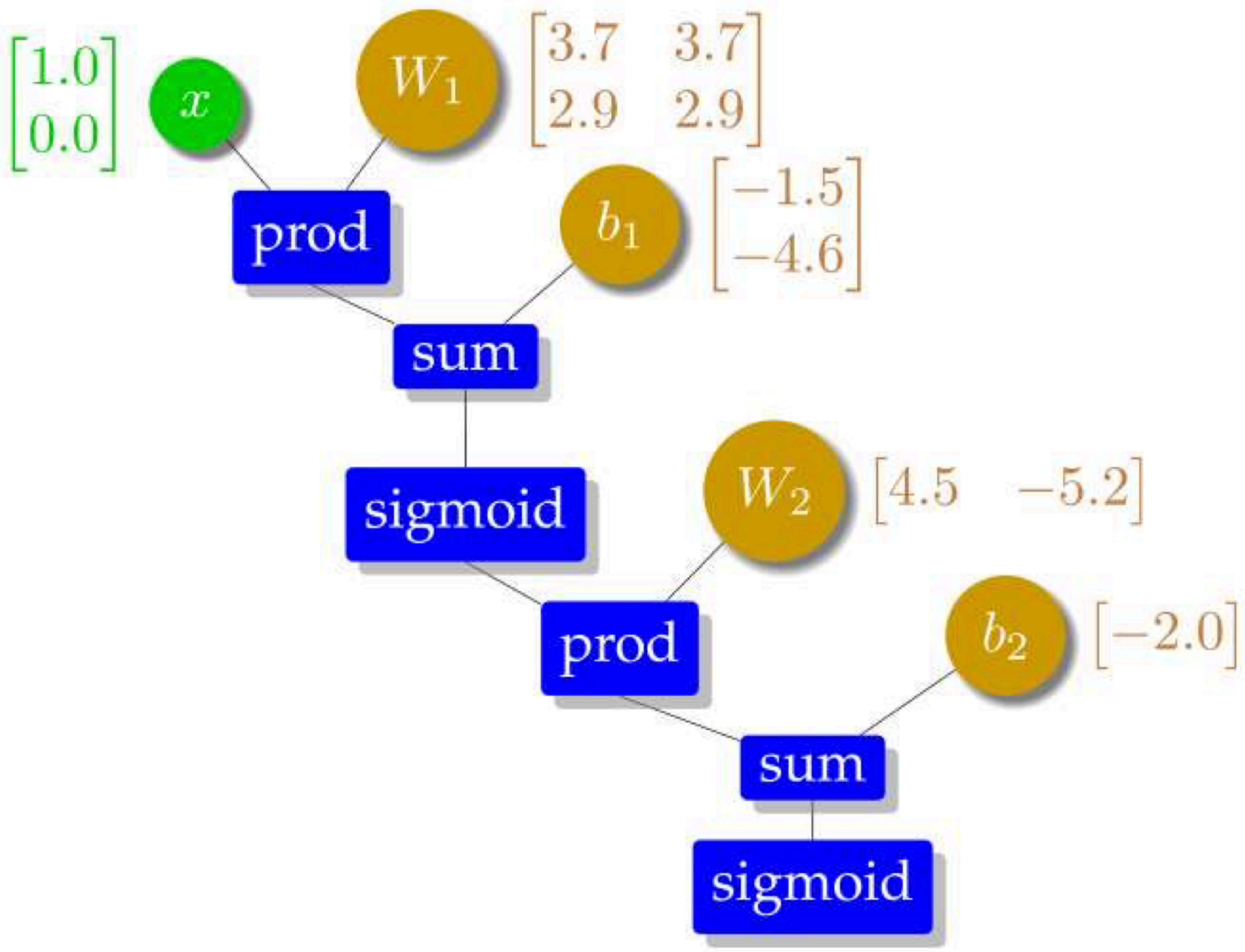
- Consider a function

$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



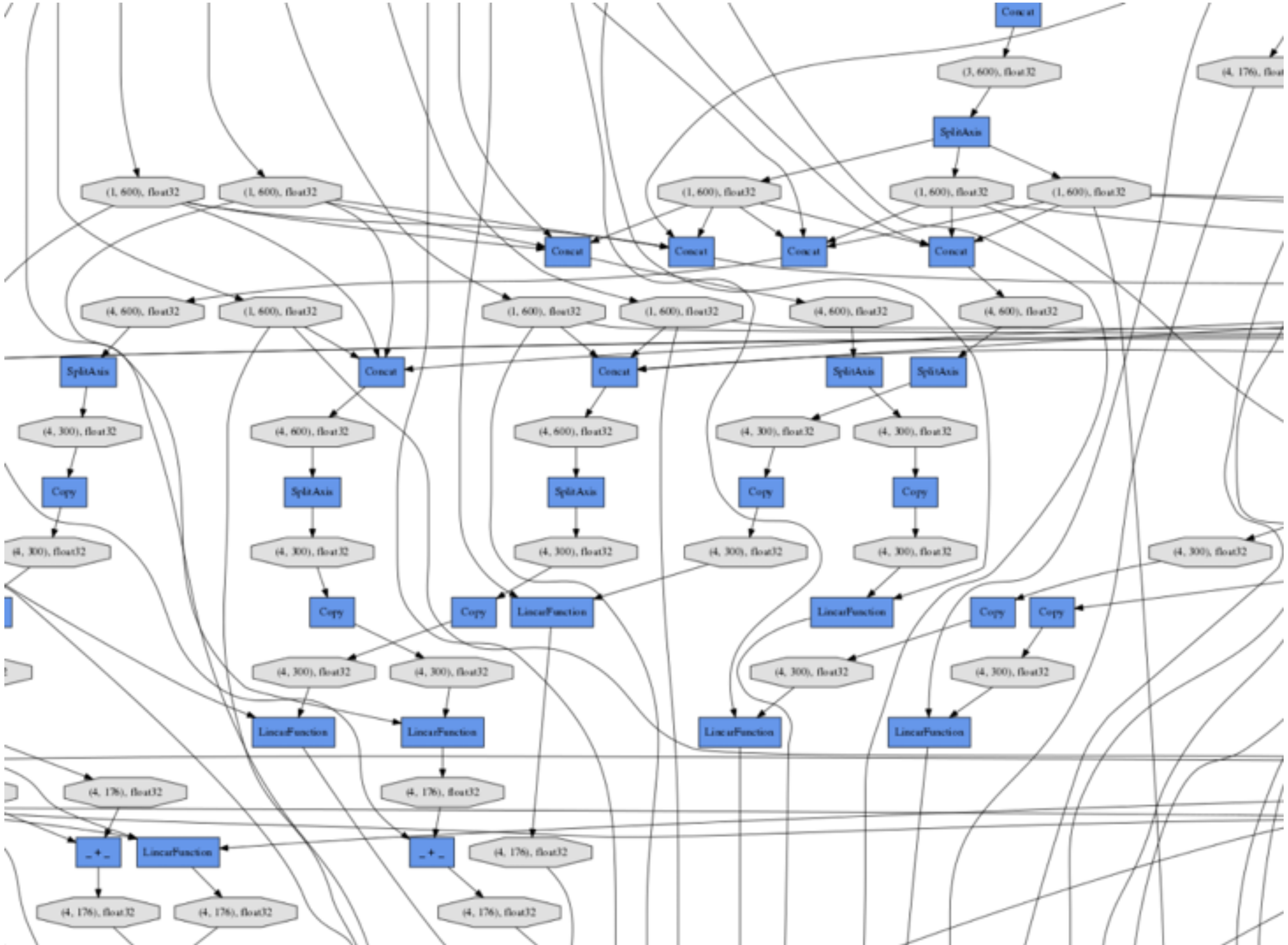
Computational Graph

- For simple neural networks, the computation graph will look like:



Computational Graph

- For larger models, the computation graph will be like:



Computational Graph

- Fortunately, deep learning frameworks will automatically construct the computational graph for you
    - PyTorch
    - TensorFlow

Remarks

- **Computation.** Backpropagation requires a lot of memory!
  - Additional memory needed is typically twice the model size (keep the gradients & intermediate states)
    - Sometimes, we discard the intermediate states (activations) and rematerialize them whenever needed
  - Gradients of some activation functions are cheaper to compute/store
    - e.g., ReLU

Next up

- More about optimization
  - Advanced optimizers
  - Training strategies
  - Network initialization

</lecture 12>