Optimizing neural networks: SGD & Backpropagation

EECE454 Intro. to Machine Learning Systems

Fall 2024

Recap: Neural networks

- **Deep learning (supervised).**
  Performing the usual optimization

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

  where the parameters are weights & biases of each layers

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

  and the predictor is the neural network

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

Today

- **Question.** How do we solve the optimization problem?

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

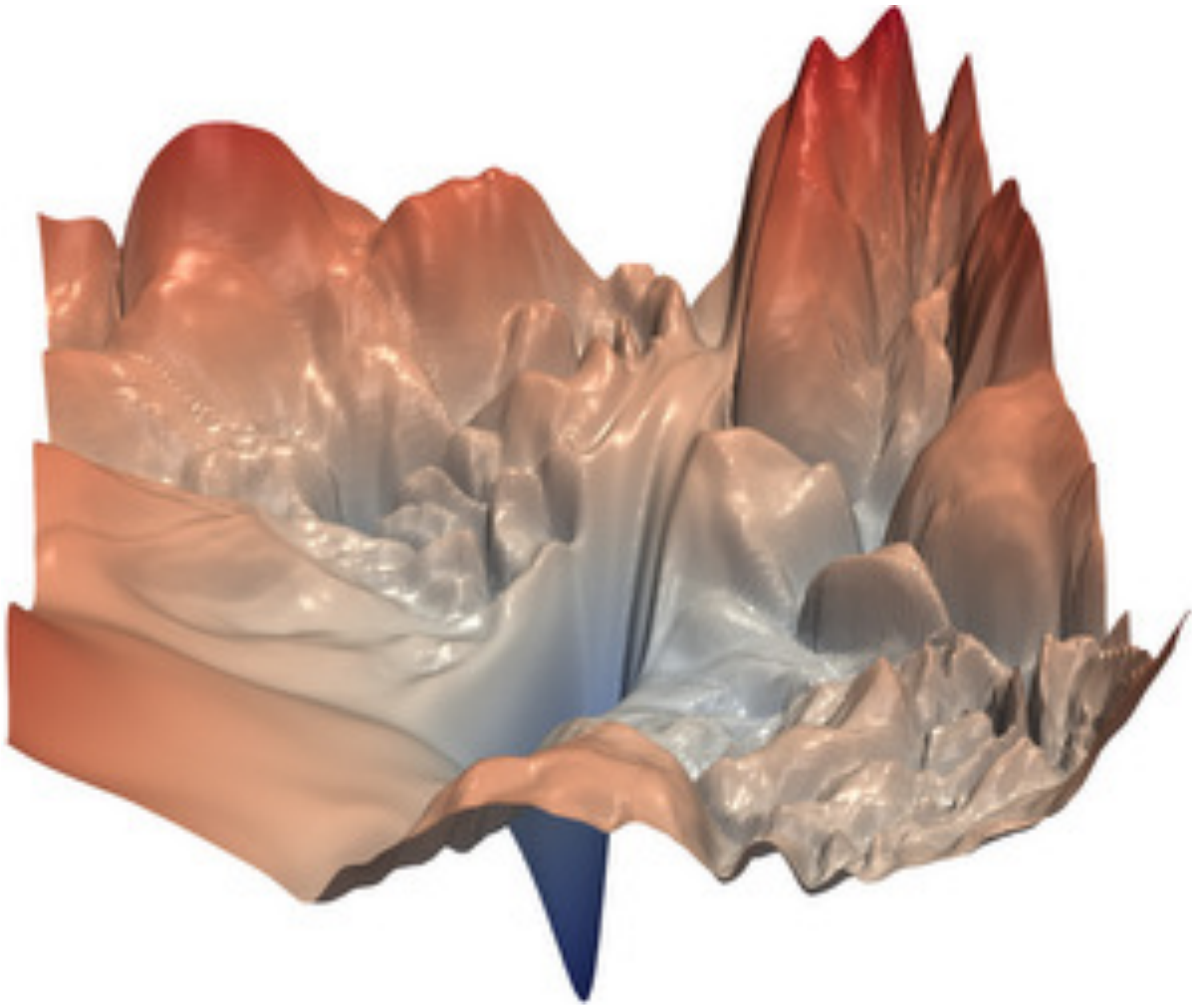
Today

- **Question.** How do we solve the optimization problem?

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

- Convex? Not really

- Critical point analysis? Too complicated



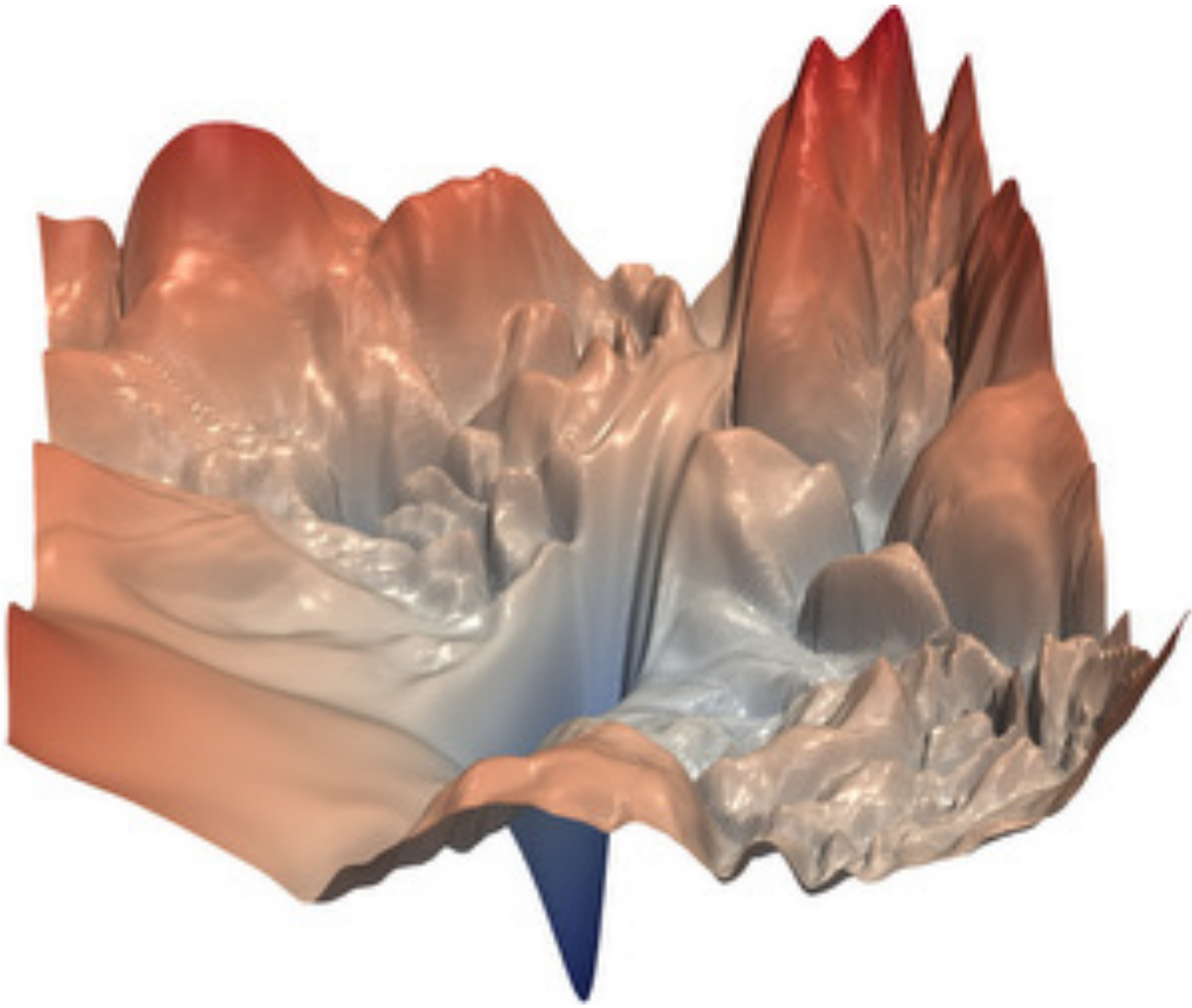
Today

- **Question.** How do we solve the optimization problem?

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

  - Convex? Not really
  - Critical point analysis? Too complicated

- **Answer.** Use a heuristic; the gradient descent



Gradient descent and stochastic gradient descent

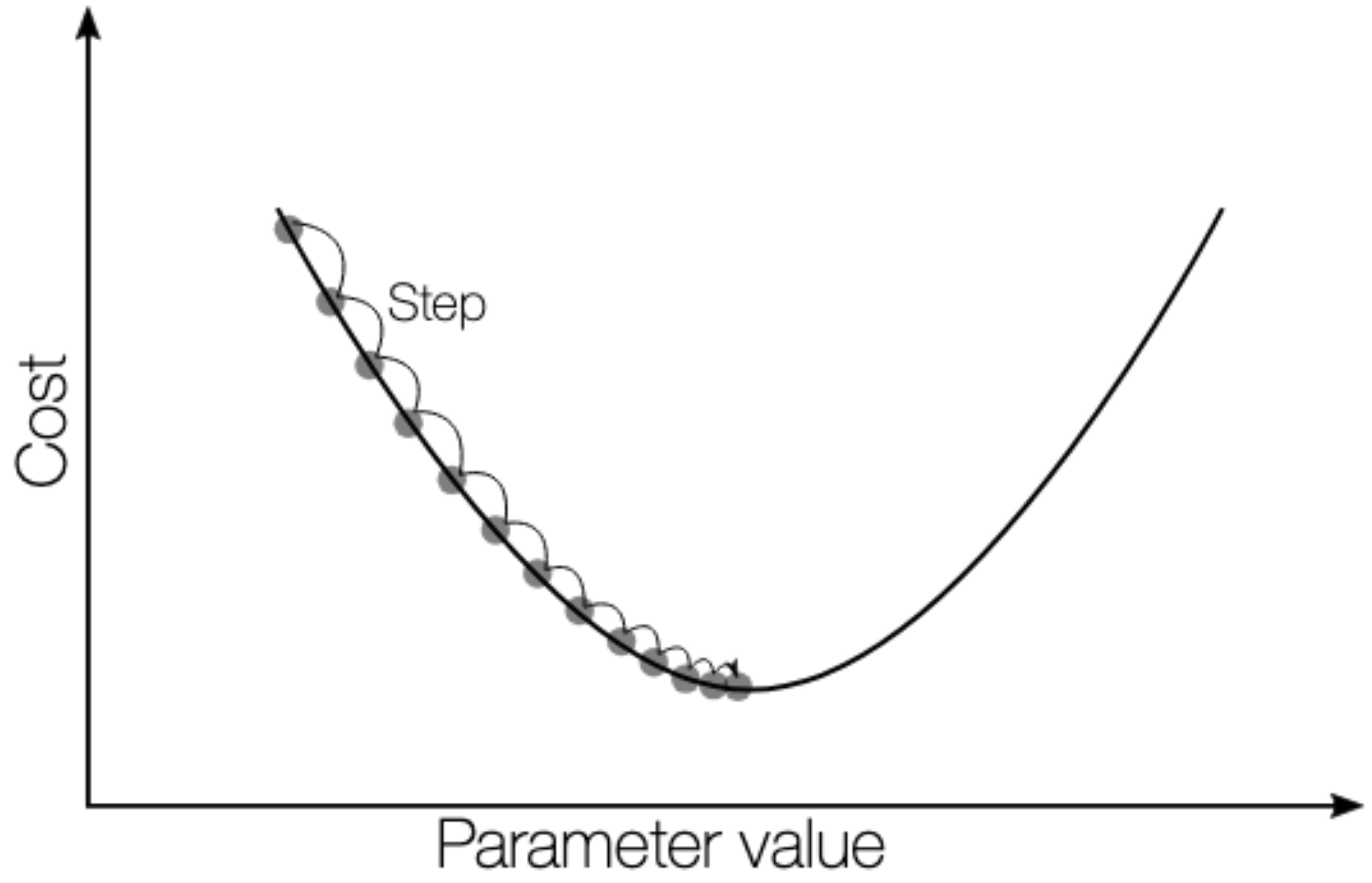
Recap: Gradient descent

- **Idea.** 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



Recap: Gradient descent

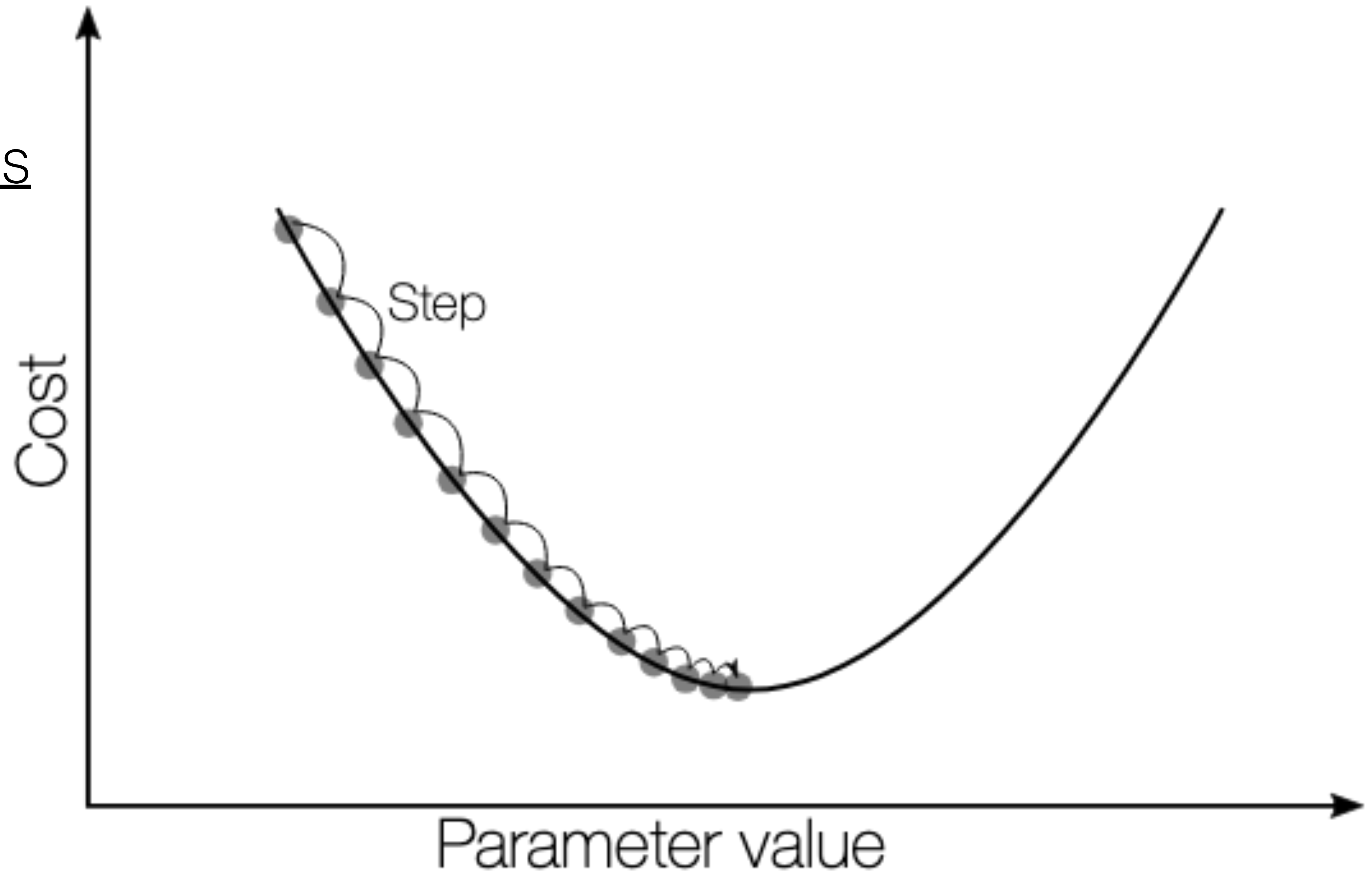
- **Idea.** Iteratively update θ in a direction that the loss decreases the fastest

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

- **Problem.** To evaluate gradients, we need to look at all data samples at each iteration.
  - Gradient is the average of per-sample gradients

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

  - But there are typically quite many samples! (e.g., ImageNet has millions of samples)

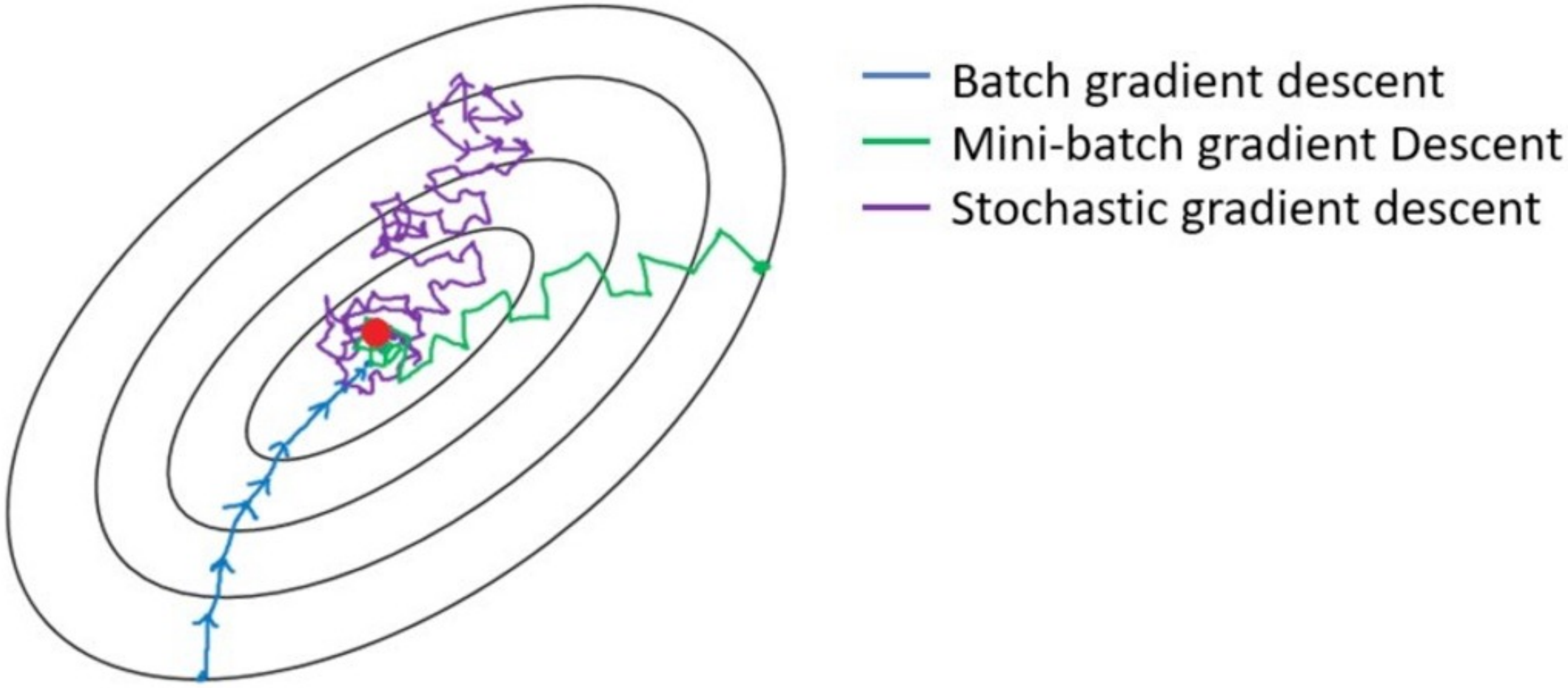


Stochastic gradient descent (SGD)

- **SGD (wide).** Look at only a few, randomly drawn samples at a time.
  - Mini-batch GD. Draw a batch $\mathscr{B}$ of samples, and compute

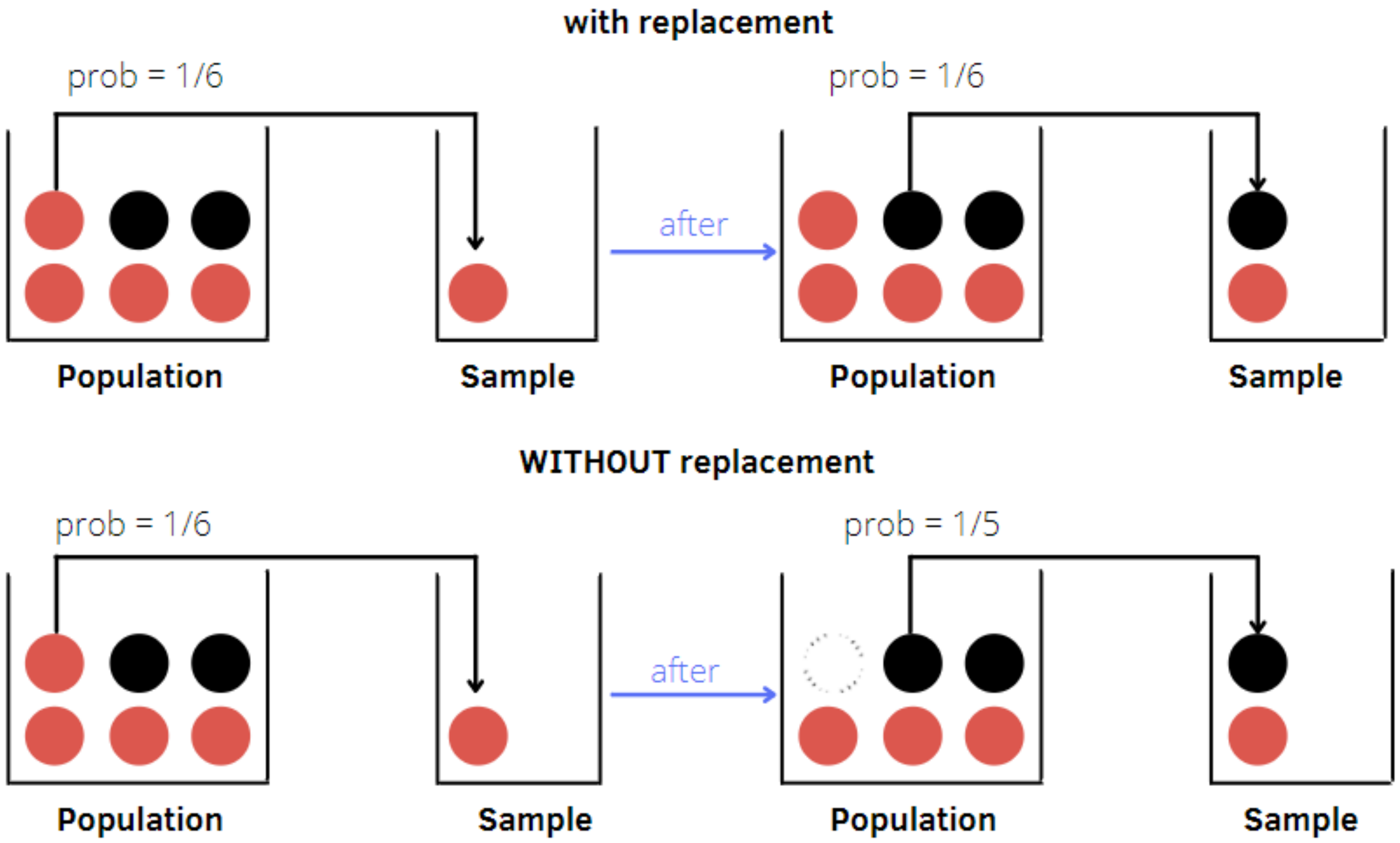
$$\hat{\nabla}_\theta L(\theta) = \frac{1}{|\mathscr{B}|} \sum_{i \in \mathscr{B}} \nabla_\theta \ell(y_i, f_\theta(\mathbf{x}_i))$$

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



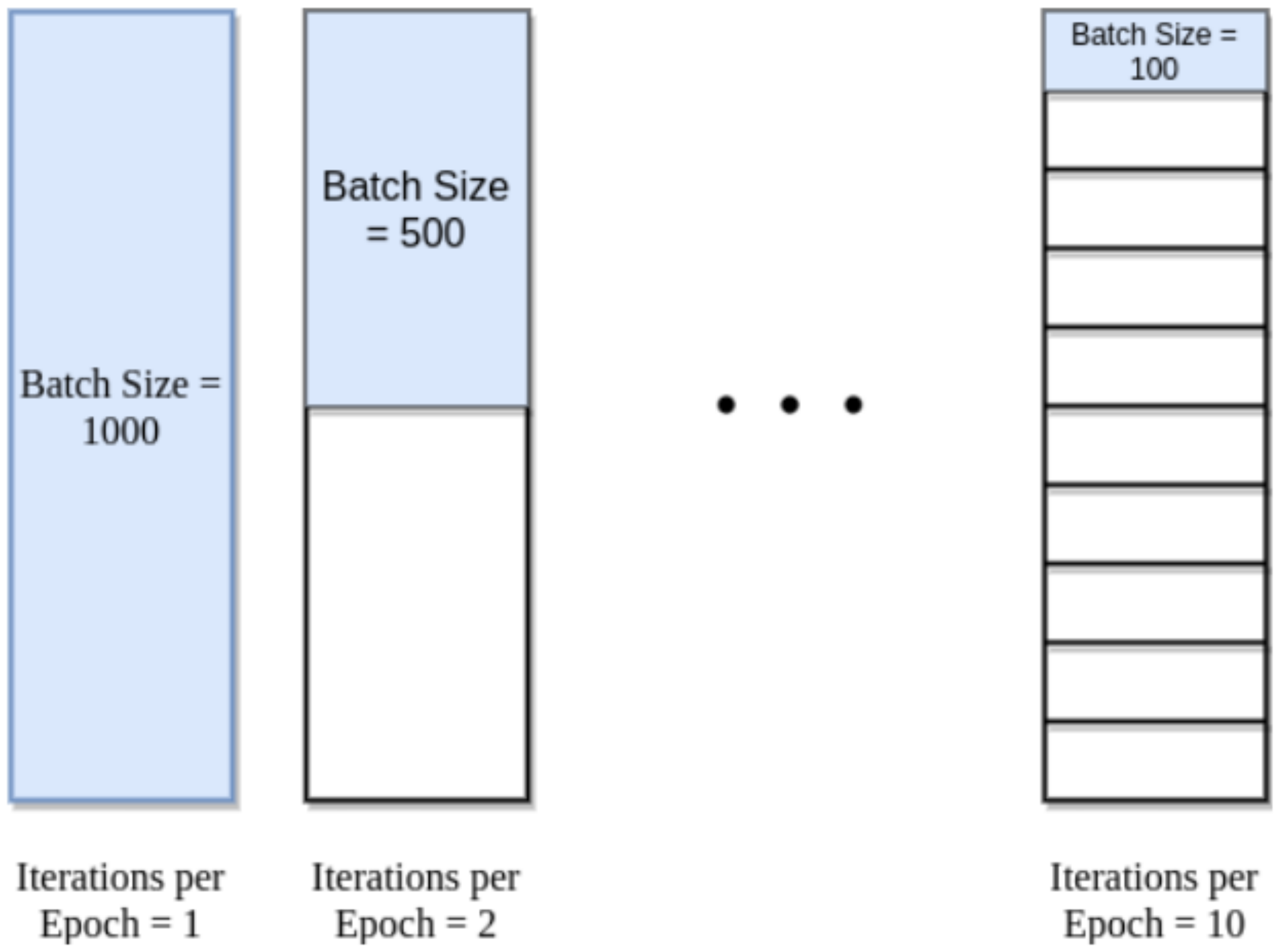
Stochastic gradient descent (SGD)

- It is typical to draw samples without replacement
    - i.e., never use a sample twice, if there exists a sample that has never been used



Stochastic gradient descent (SGD)

- It is typical to draw samples without replacement
  - i.e., never use a sample twice, if there exists a sample that has never been used
- **Epoch.** A set of iterations until every samples has been used once.
  - Example. If we use a batch size 64 for a dataset of size 32,000, we need 500 steps for a single epoch.
- **Note.** The learning rate and the batch size are the most important hyperparameters



Computing the gradients

Evaluating gradients

- **(per-sample) gradient.** A product of loss derivative and the predictor gradient

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

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

Predictor gradient

Recall the **chain rule**:

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

Evaluating gradients

- **(per-sample) gradient.** A product of loss derivative and the predictor gradient

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

- **Loss derivative.** 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 pass the data through the predictor, measure the error, and multiply 2.

Evaluating gradients

- **(per-sample) gradient.** A product of loss derivative and the predictor gradient

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

- **Predictor gradient.** Much trickier
  - The parameter θ is high-dimensional (billions — trillions)

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

  - This makes the numerical method very computation-heavy to use.

The difficulty of numerical method

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

- **Numerical method.** Evaluate each partial derivative by taking the limit

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

  - Problem. Cannot take the limit.
    - Approximate by choosing a very small ϵ
  - Requires evaluating both $g(x+\epsilon)$ and $g(x)$... for each parameter dimension!

The difficulty of 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 (first dim):

[0.34 + 0.0001,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,…]

(1.25322 - 1.25347)/0.0001 = -2.5

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

The difficulty of 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

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

The difficulty of 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 (third dim):

[0.34,
-1.11,
0.78 + 0.0001,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?,
?,
?,
?,
?,…]

(1.25347 - 1.25347)/0.0001=0

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

The difficulty of numerical method

- **Pros.**
  - Easy to implement
  - Can use for black-box model
- **Cons.**
  - Only gives you approximate
    - Cannot send $\epsilon \to 0$, due to finite precision
  - Very slow
    - Requires at least $d + 1$ evaluations of $f_\theta(\mathbf{x})$, for d-dimensional parameter θ

Analytic method

- Thus, we mainly use what we call the analytic method
- **Idea.** Derive an analytic expression of the gradient
    - Example. If $g(x) = \sin(5 \cdot \exp(x))$, we know that the gradients will be

$$g'(x) = 5 \cdot \cos(5 \cdot \exp(x)) \cdot \exp(x)$$

Analytic method

- Thus, we mainly use what we call the analytic method
- **Idea.** Derive an analytic expression of the gradient
  - Example. If $g(x) = \sin(5 \cdot \exp(x))$, we know that the gradients will be

$$g'(x) = 5 \cdot \cos(5 \cdot \exp(x)) \cdot \exp(x)$$

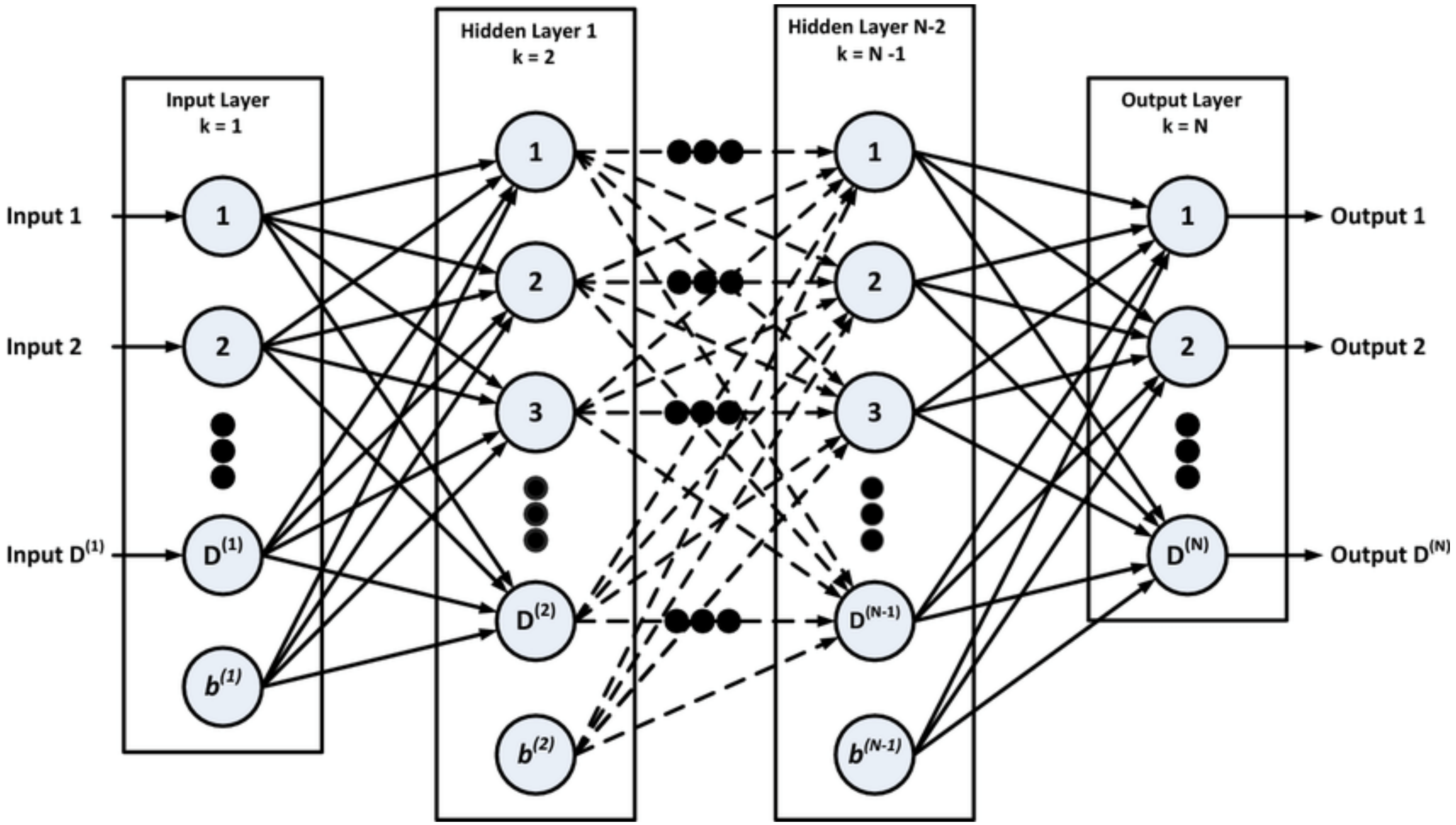
  - **Pros.** Exact & Fast
  - **Cons.** Requires a careful implementation for complicated functions
    - Often check correctness, using the numerical method (called “gradient check”)

Analytic forms of NN gradients, and backpropagation

Analytic form of gradients

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

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



Analytic form of gradients

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

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

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

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

  - Derivatives of elementary operations can be hard-coded
  - Use chain rule to combine these
    - Let us see an example...

Chain rule: Example

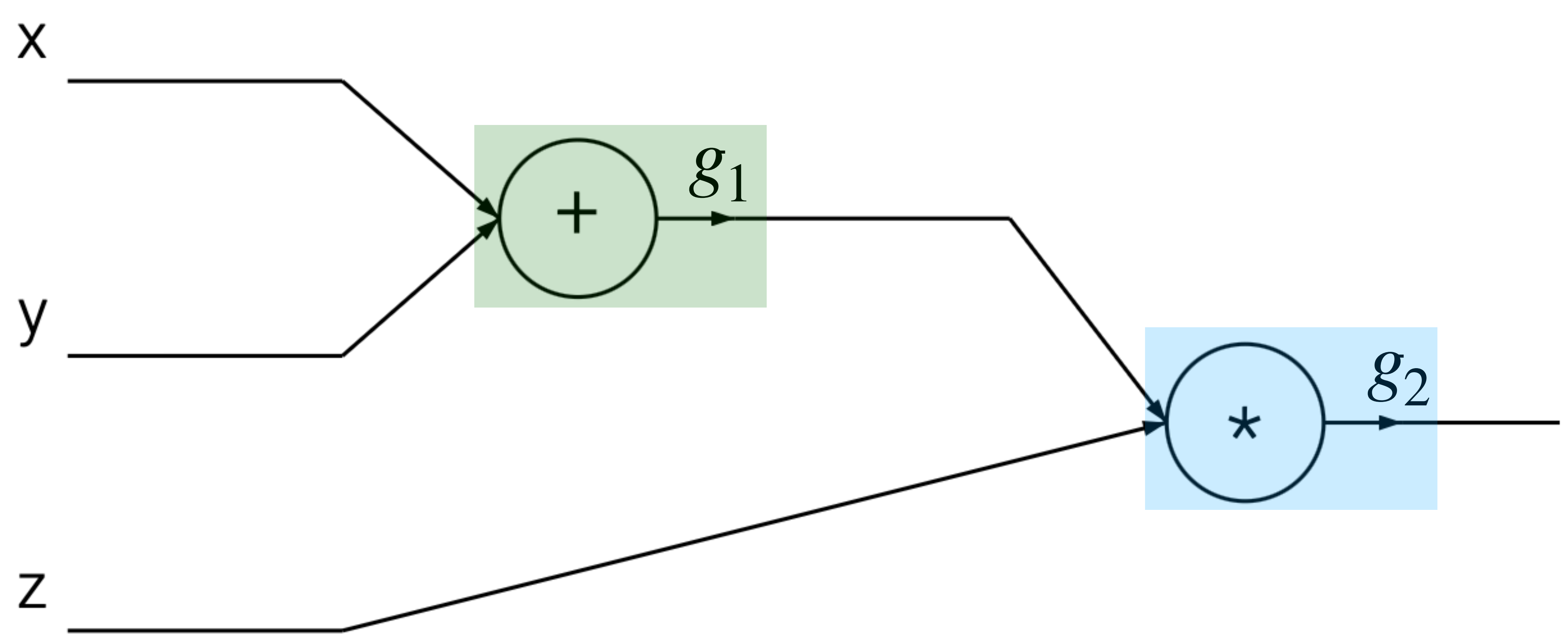
- **Example.** Consider a function

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

  - This can be viewed as a composition of two <u>elementary operations</u>

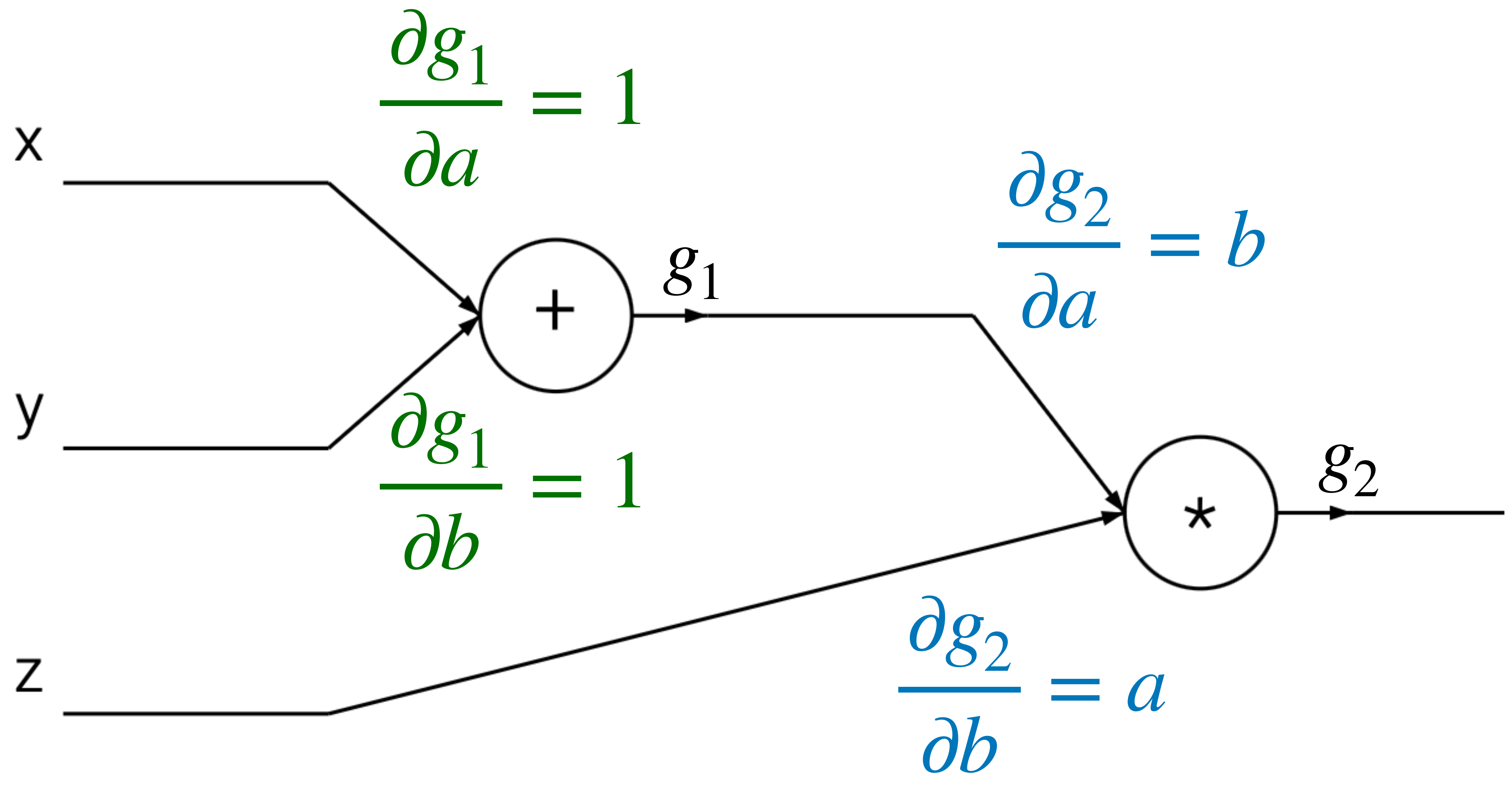
$$g(x, y, z) = g_2(g_1(x, y), z)$$

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



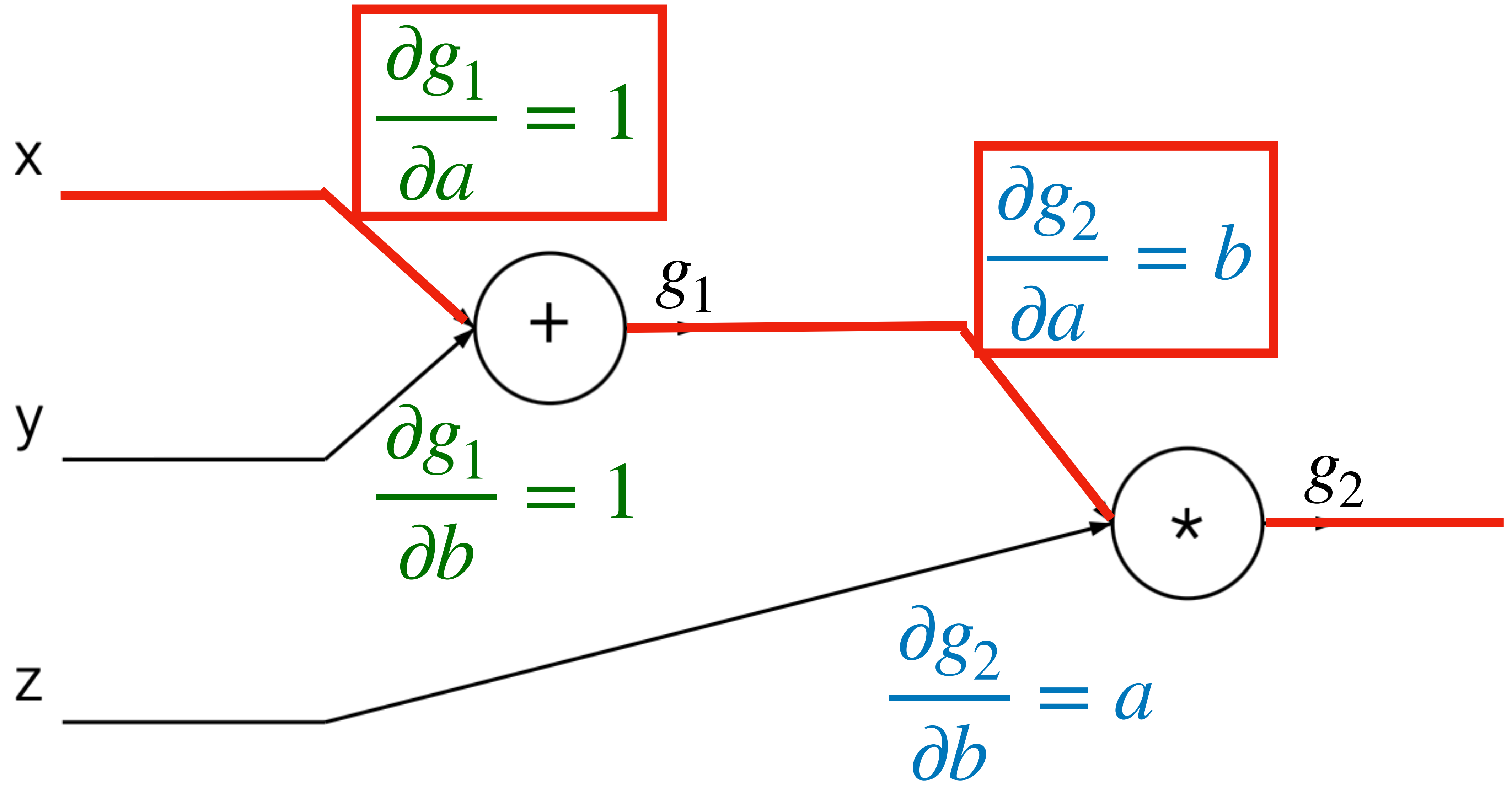
Chain rule: Example

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



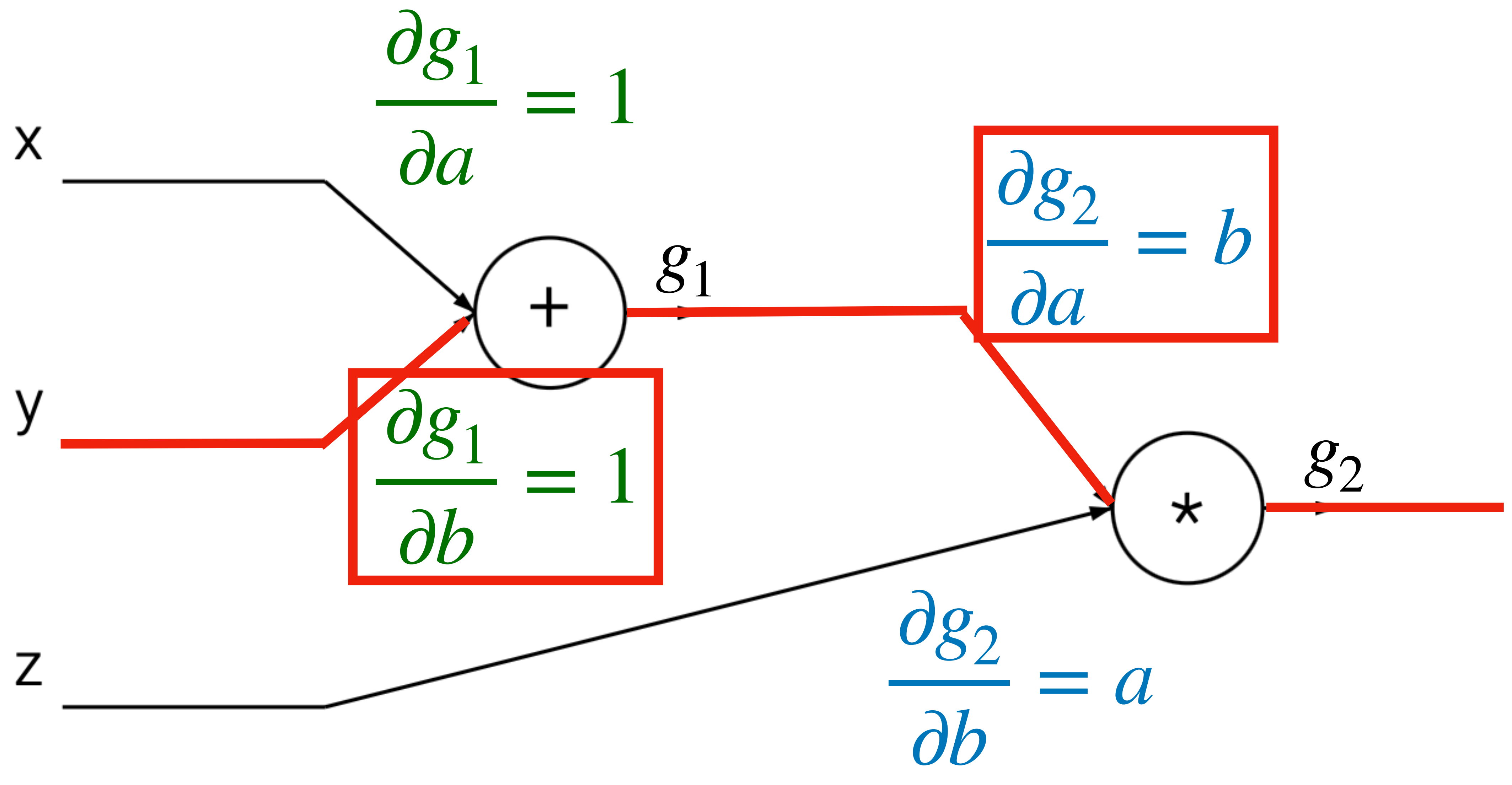
Chain rule: Example

- Each elementary operation has <u>easy-to-write</u> gradients
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$
- **Chain rule** tells you that:
  - $\frac{\partial g}{\partial x}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial a}(g_1(x, y), z)}_{= z} \cdot \underbrace{\frac{\partial g_1}{\partial a}(x, y)}_{= 1}$



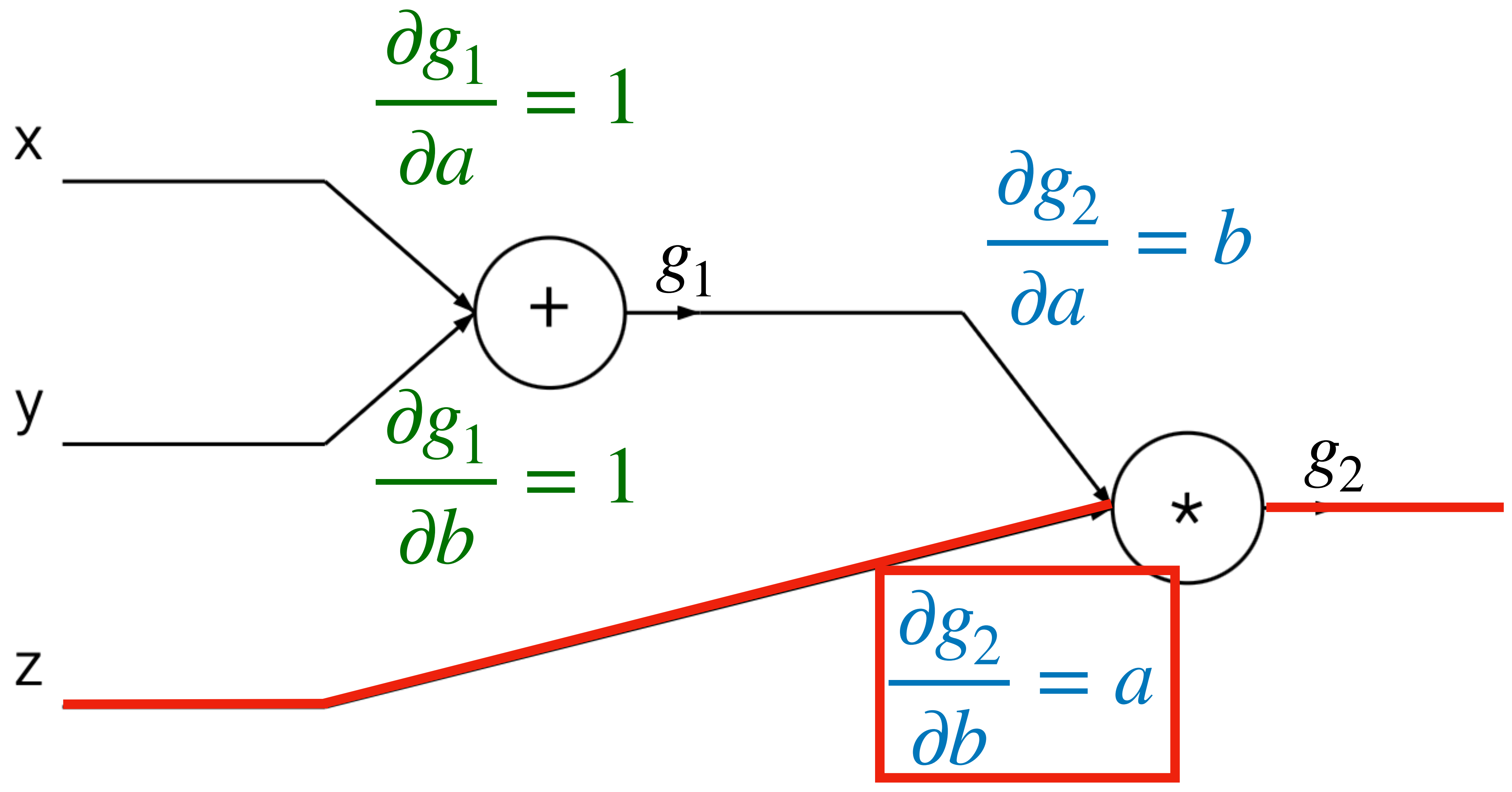
Chain rule: Example

- Each elementary operation has <u>easy-to-write</u> gradients
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$
- **Chain rule** tells you that:
  - $\frac{\partial g}{\partial x}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial a}(x, y)$
  - $\frac{\partial g}{\partial y}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial a}(g_1(x, y), z)}_{= z} \cdot \underbrace{\frac{\partial g_1}{\partial b}(x, y)}_{= 1}$



Chain rule: Example

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



- **Chain rule** tells you that:
  - $\frac{\partial g}{\partial x}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial a}(x, y)$
  - $\frac{\partial g}{\partial y}(x, y, z) = \frac{\partial g_2}{\partial a}(g_1(x, y), z) \cdot \frac{\partial g_1}{\partial b}(x, y)$

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

Chain rule: Example

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

Observation. Computing gradients requires intermediate values of the composite function.

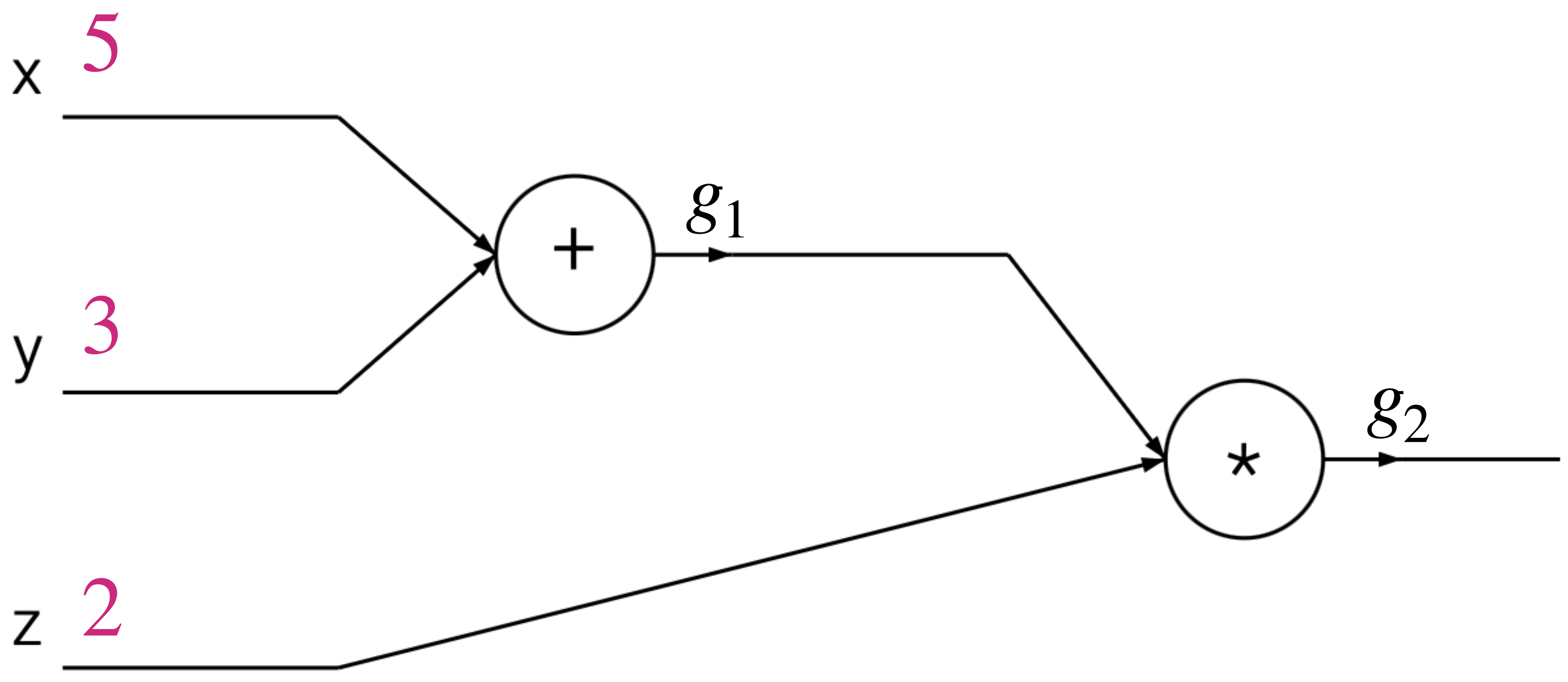
Idea. We first compute all intermediate values, and then combine them to get gradients.

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

$= g_1(x, y)$

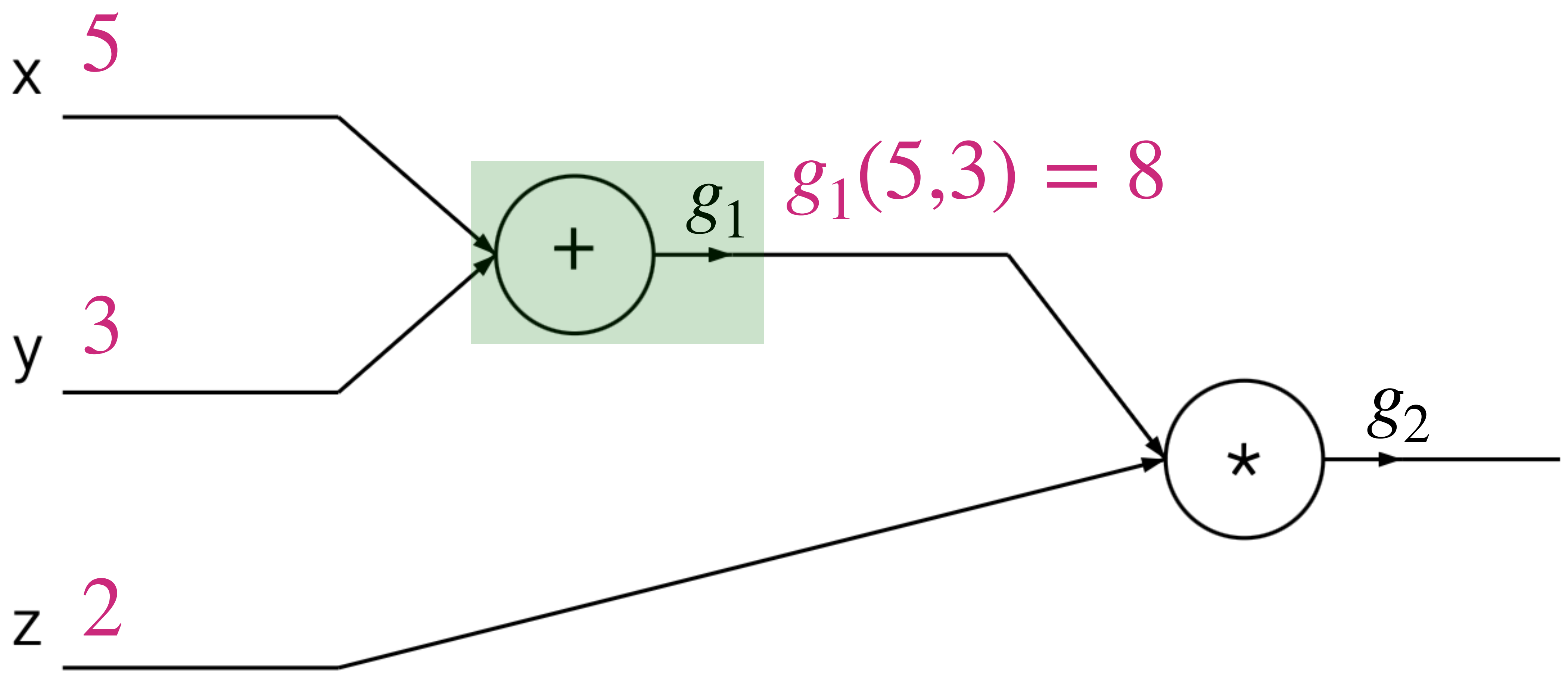
Neural network training

- Iteratively apply three steps:
  - **1. Forward Pass.** Compute the function output, storing all intermediate values on memory
    - From input to output



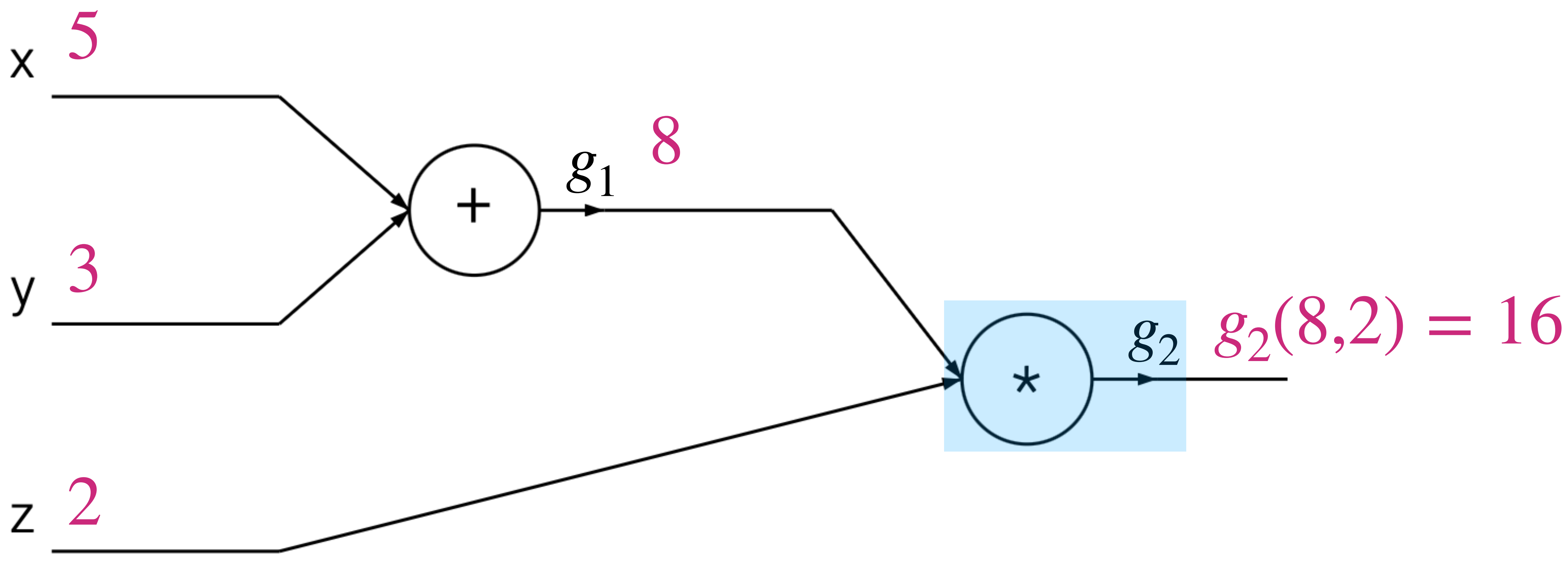
Neural network training

- Iteratively apply three steps:
  - **1. Forward Pass.** Compute the function output, storing all intermediate values on memory
    - From input to output



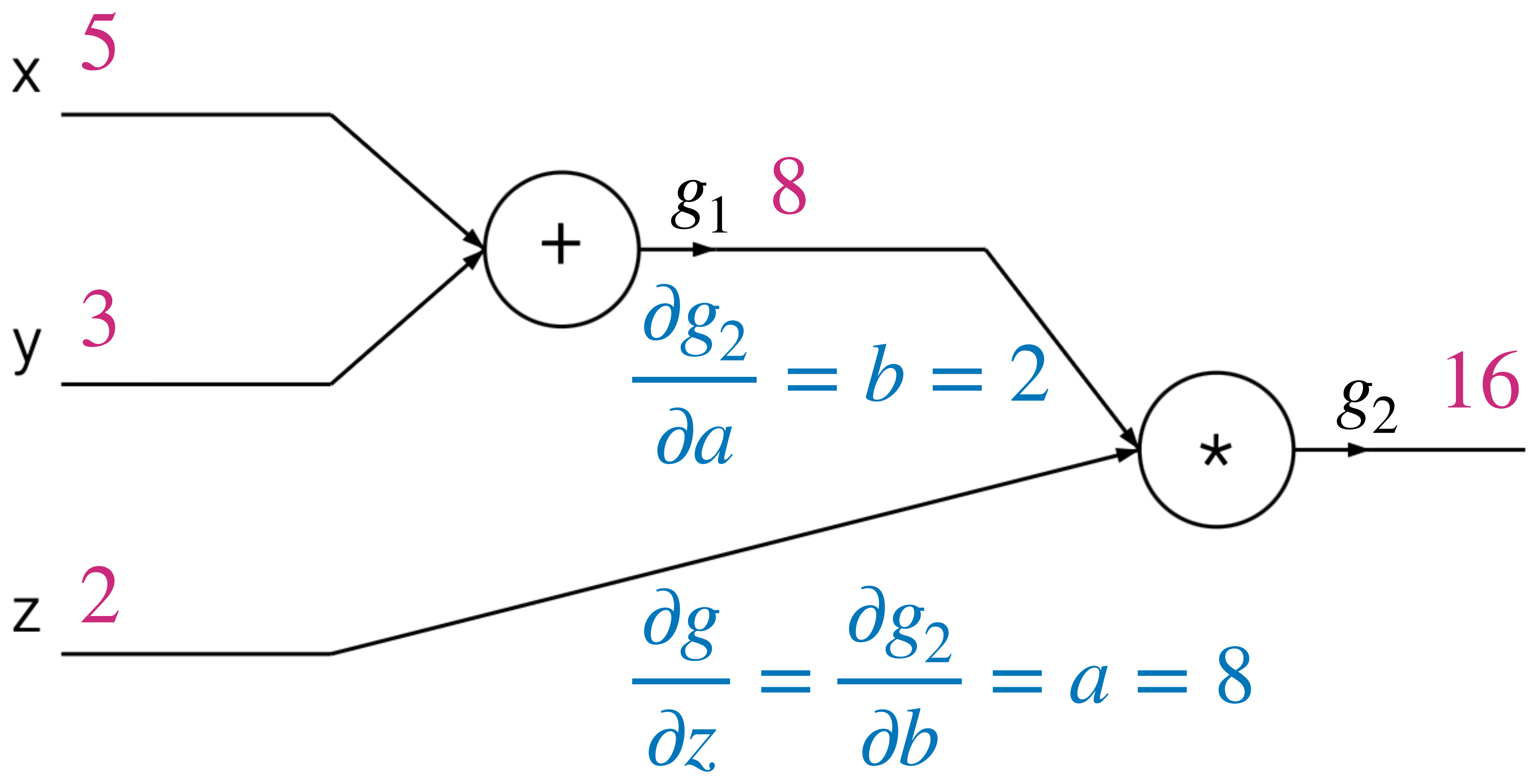
Neural network training

- Iteratively apply three steps:
  - **1. Forward Pass.** Compute the function output, storing all intermediate values on memory
    - From input to output



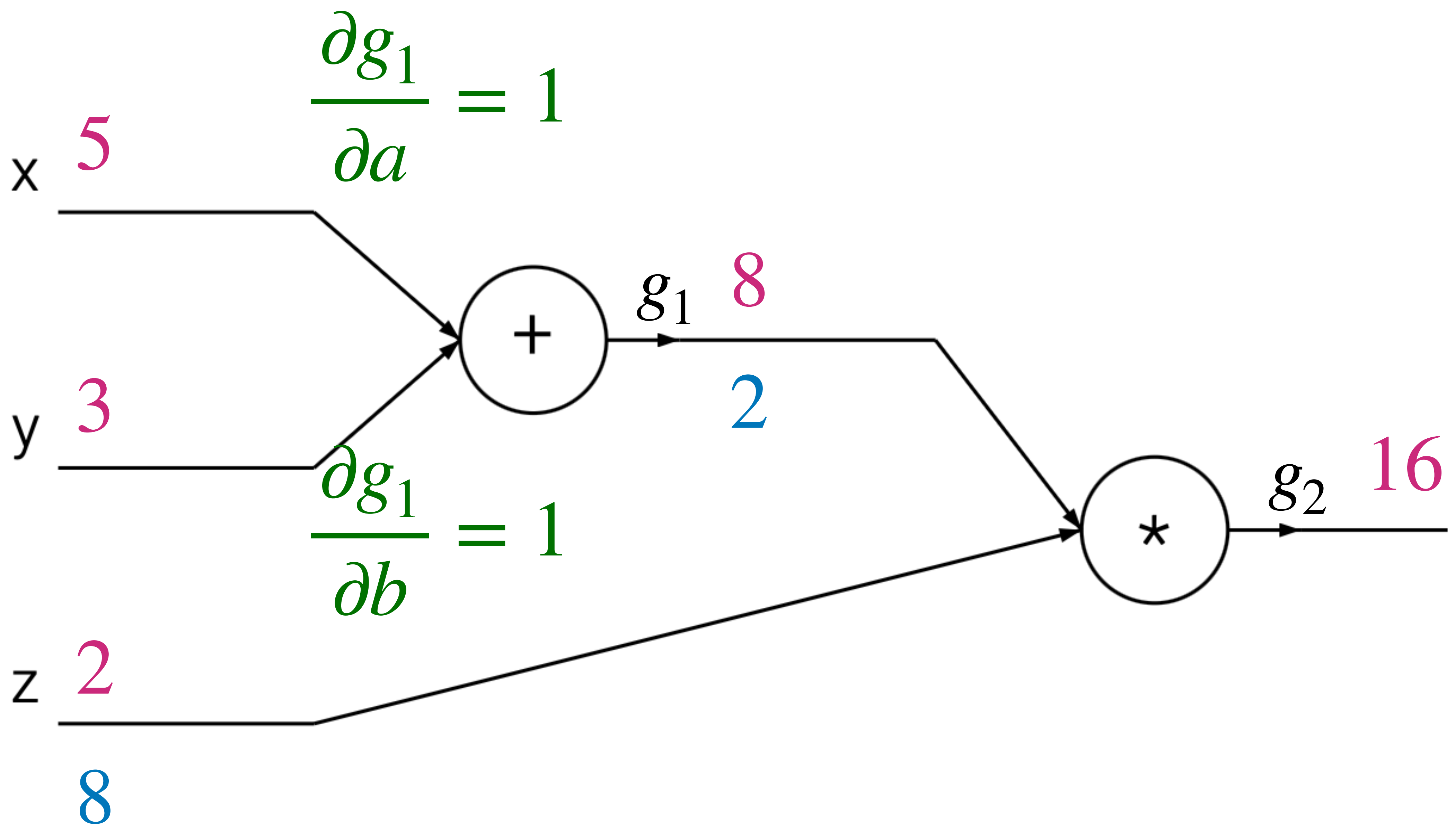
Neural network training

- Iteratively apply three steps:
  - **2. Backward Pass.** Compute the gradient, using stored values
    - From output to input



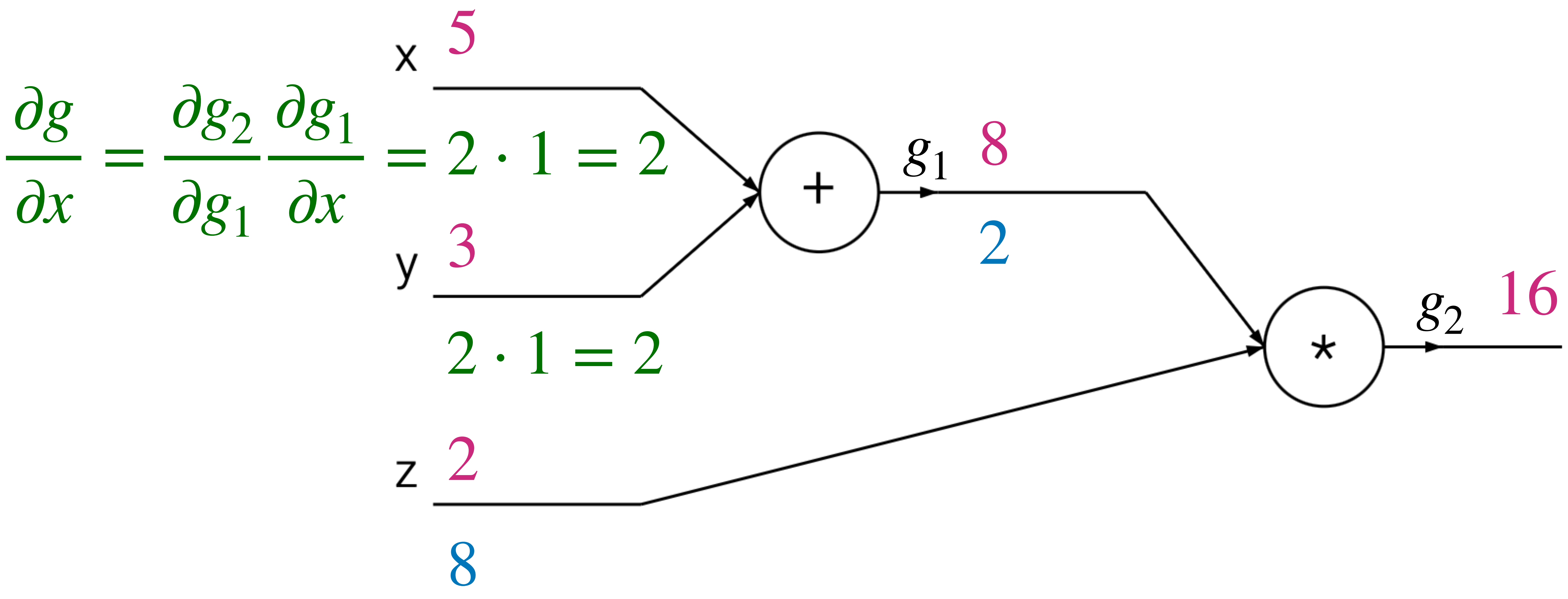
Neural network training

- Iteratively apply three steps:
  - **2. Backward Pass.** Compute the gradient, using stored values
    - From output to input



Neural network training

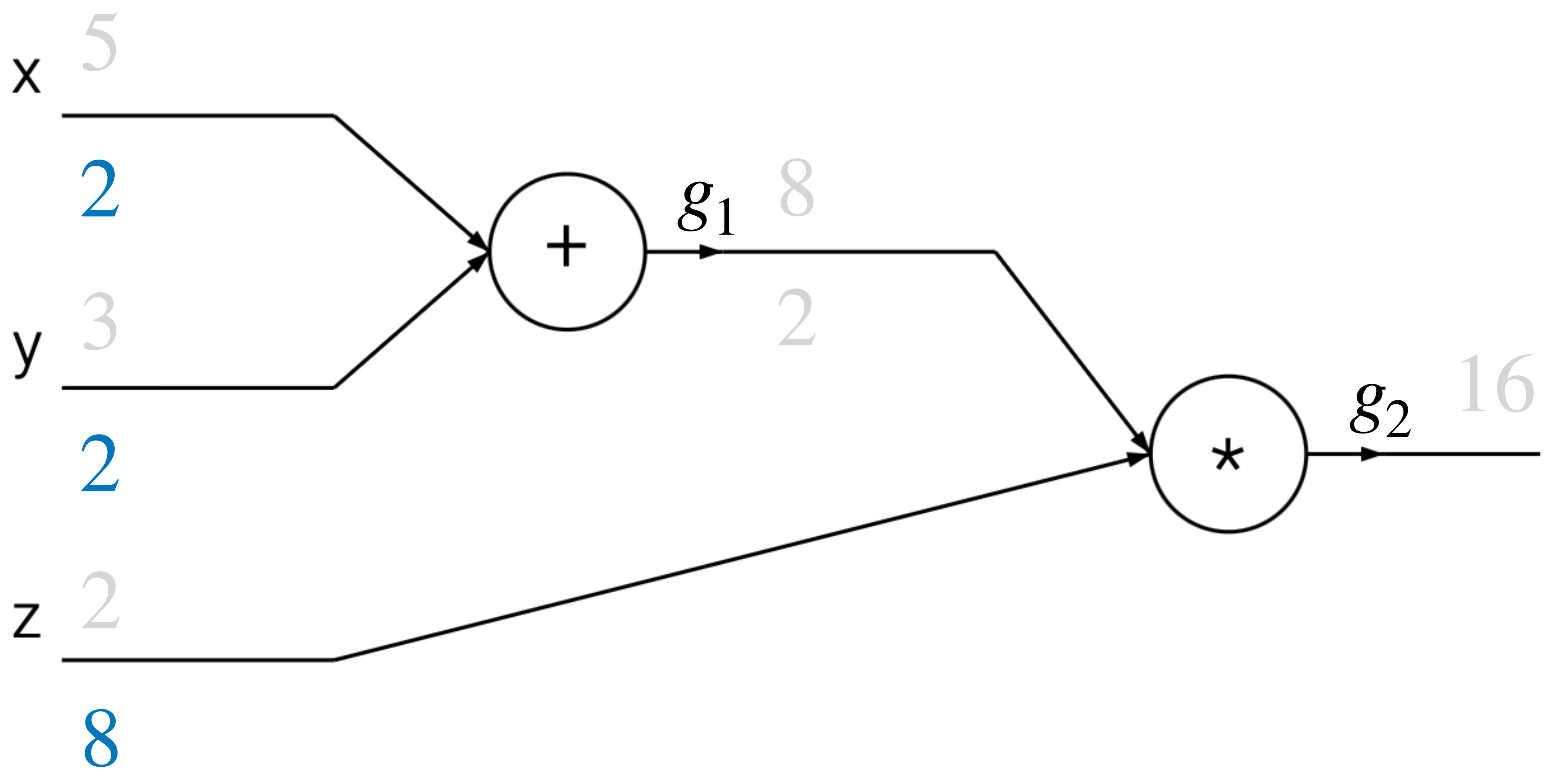
- Iteratively apply three steps:
  - **2. Backward Pass.** Compute the gradient, using stored values
    - From output to input



Neural network training

- Iteratively apply three steps:
  - **3. SGD.** Update the parameters

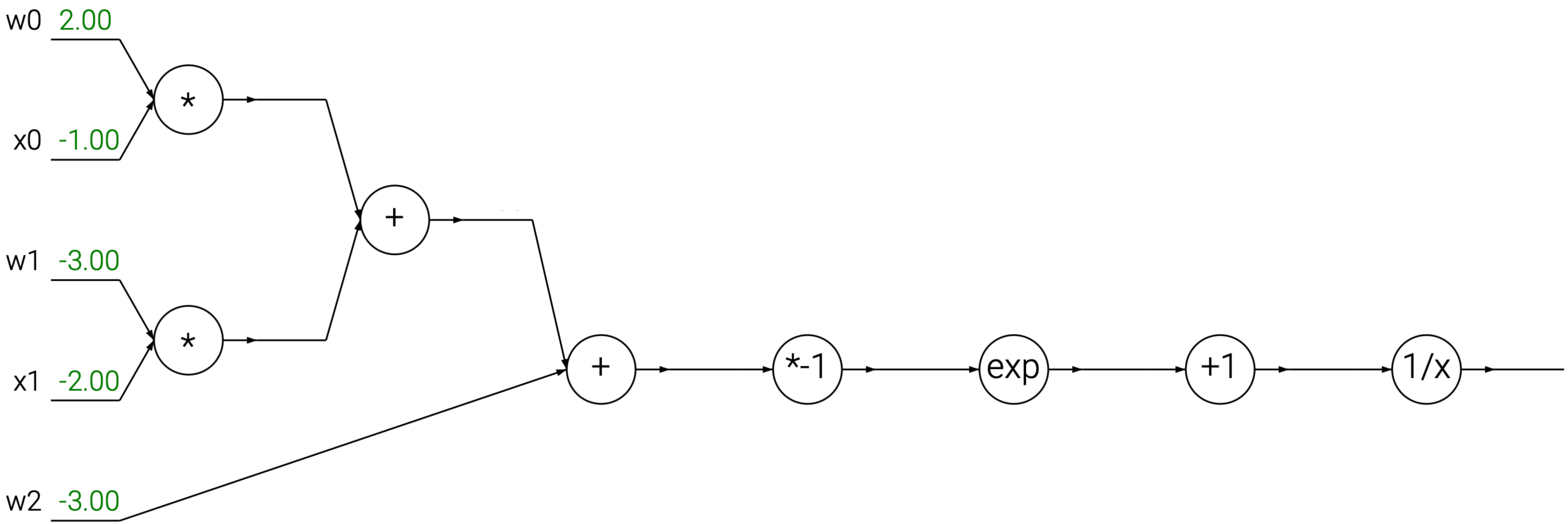
$$x \leftarrow x - \eta \cdot 2, \qquad y \leftarrow y - \eta \cdot 2, \qquad z \leftarrow z - \eta \cdot 8$$



Another example

- Consider a function

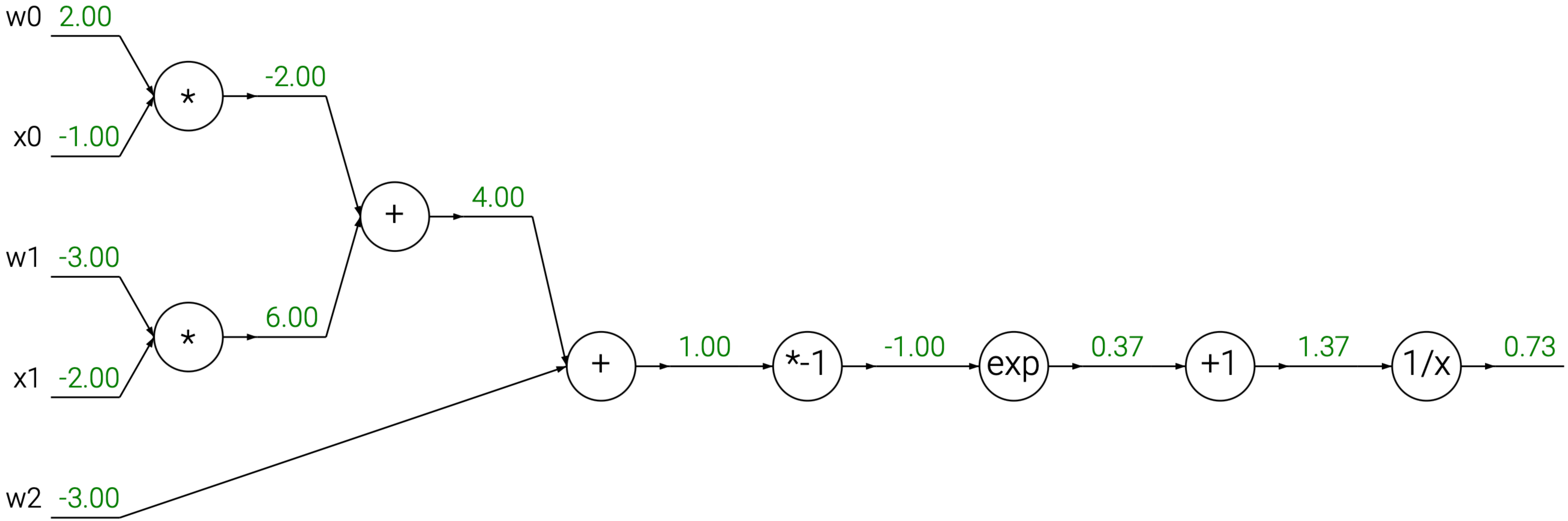
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

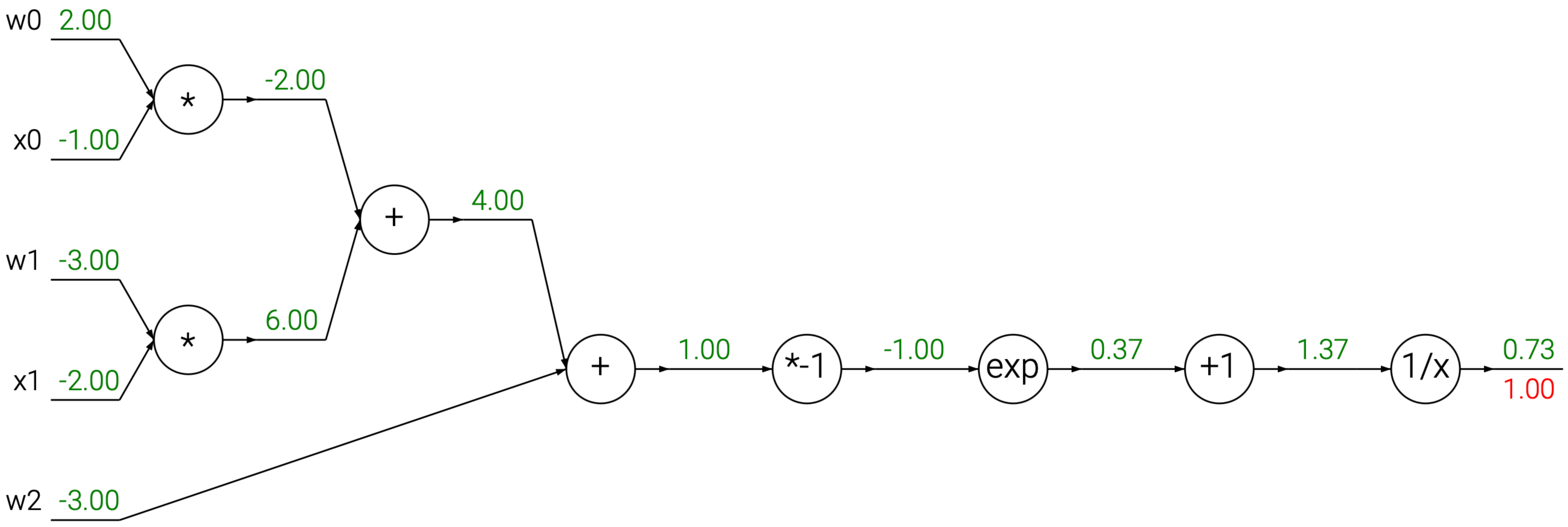
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

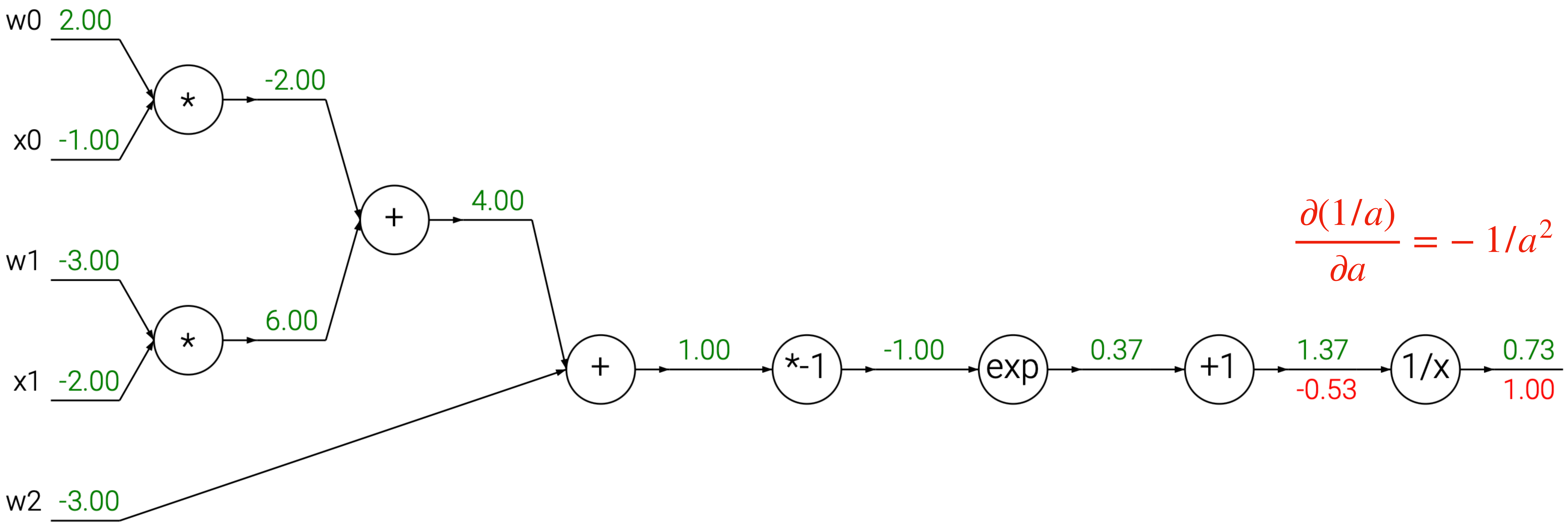
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0x_0 + w_1x_1 + w_2))}$$



Another example

- Consider a function

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

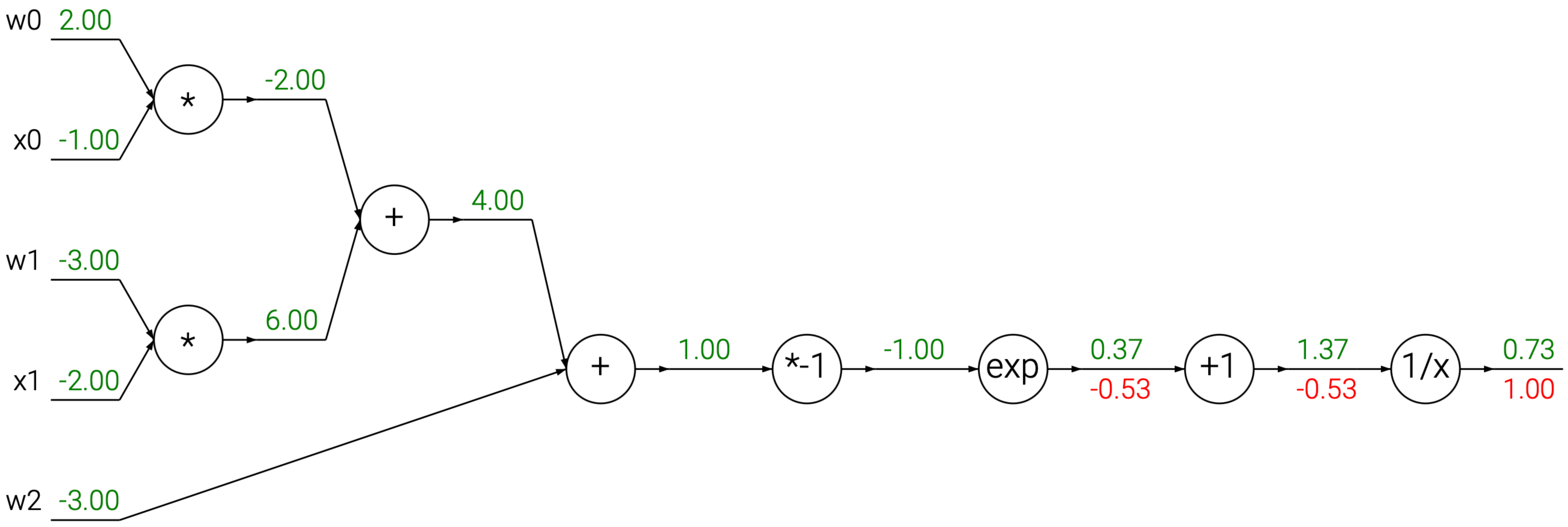


$$\frac{\partial(1/a)}{\partial a} = -1/a^2$$

Another example

- Consider a function

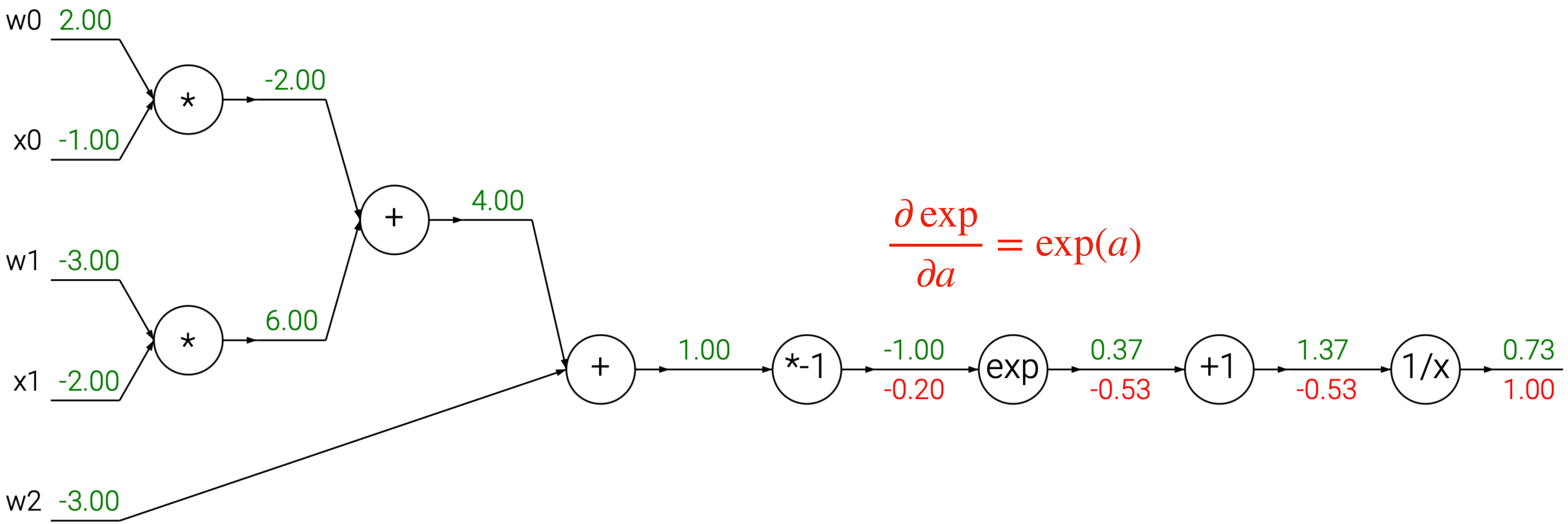
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

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

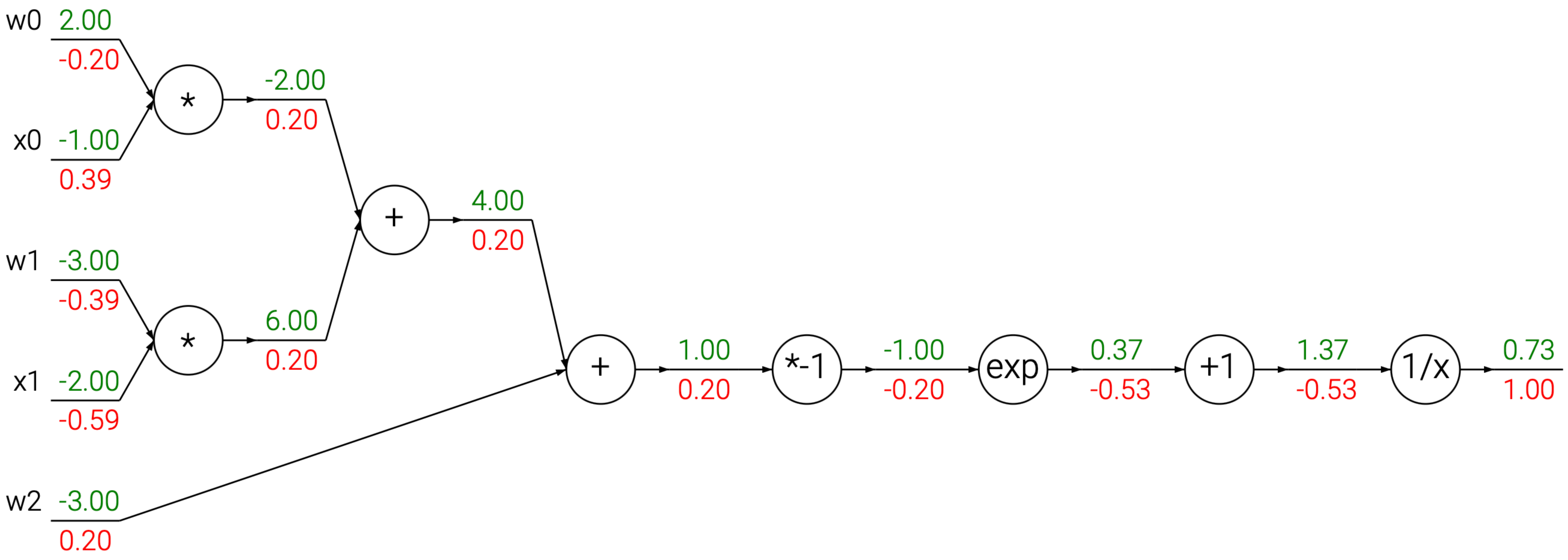


$$\frac{\partial \exp}{\partial a} = \exp(a)$$

Another example

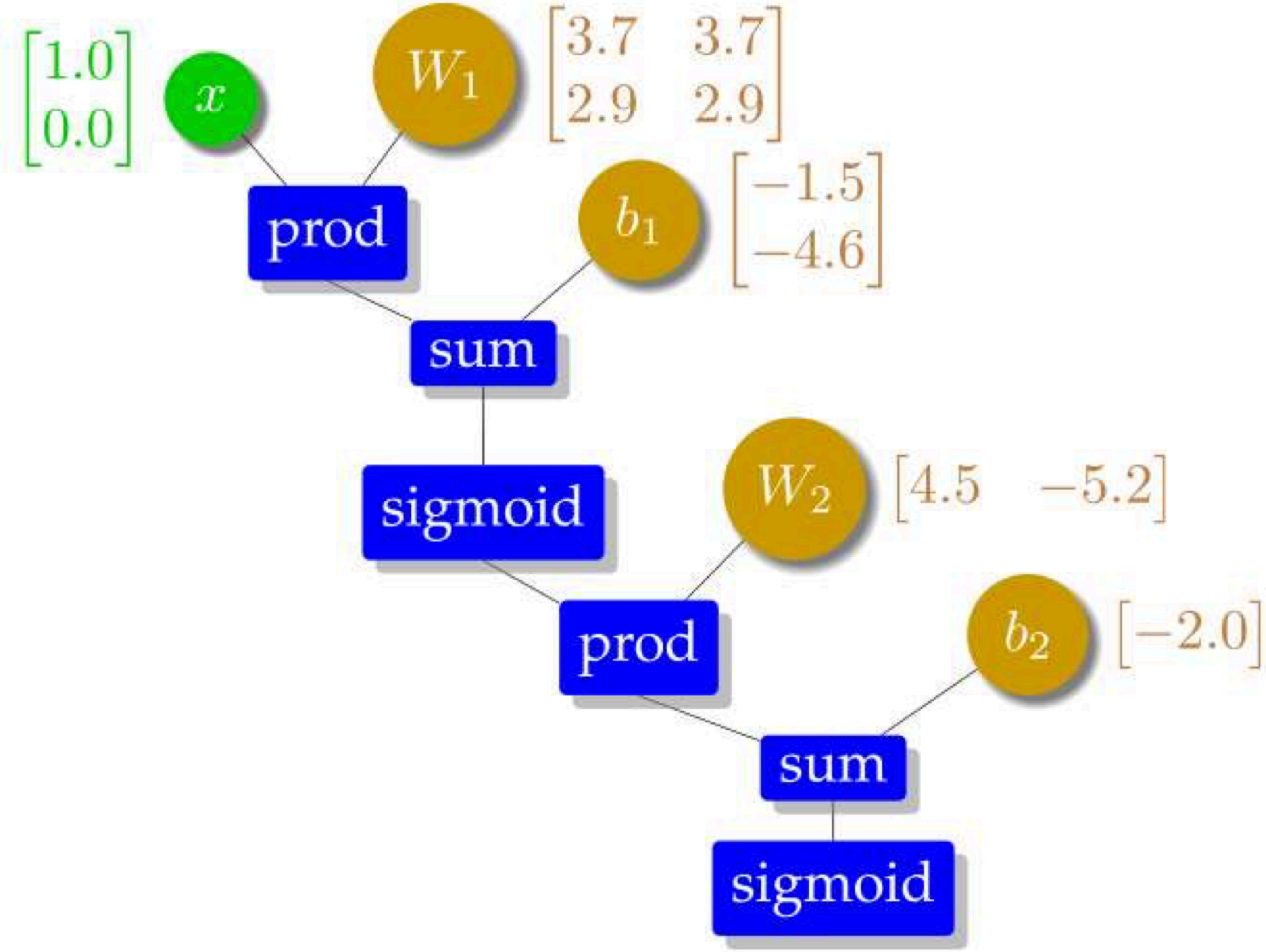
- Consider a function

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



Computational graphs of NNs

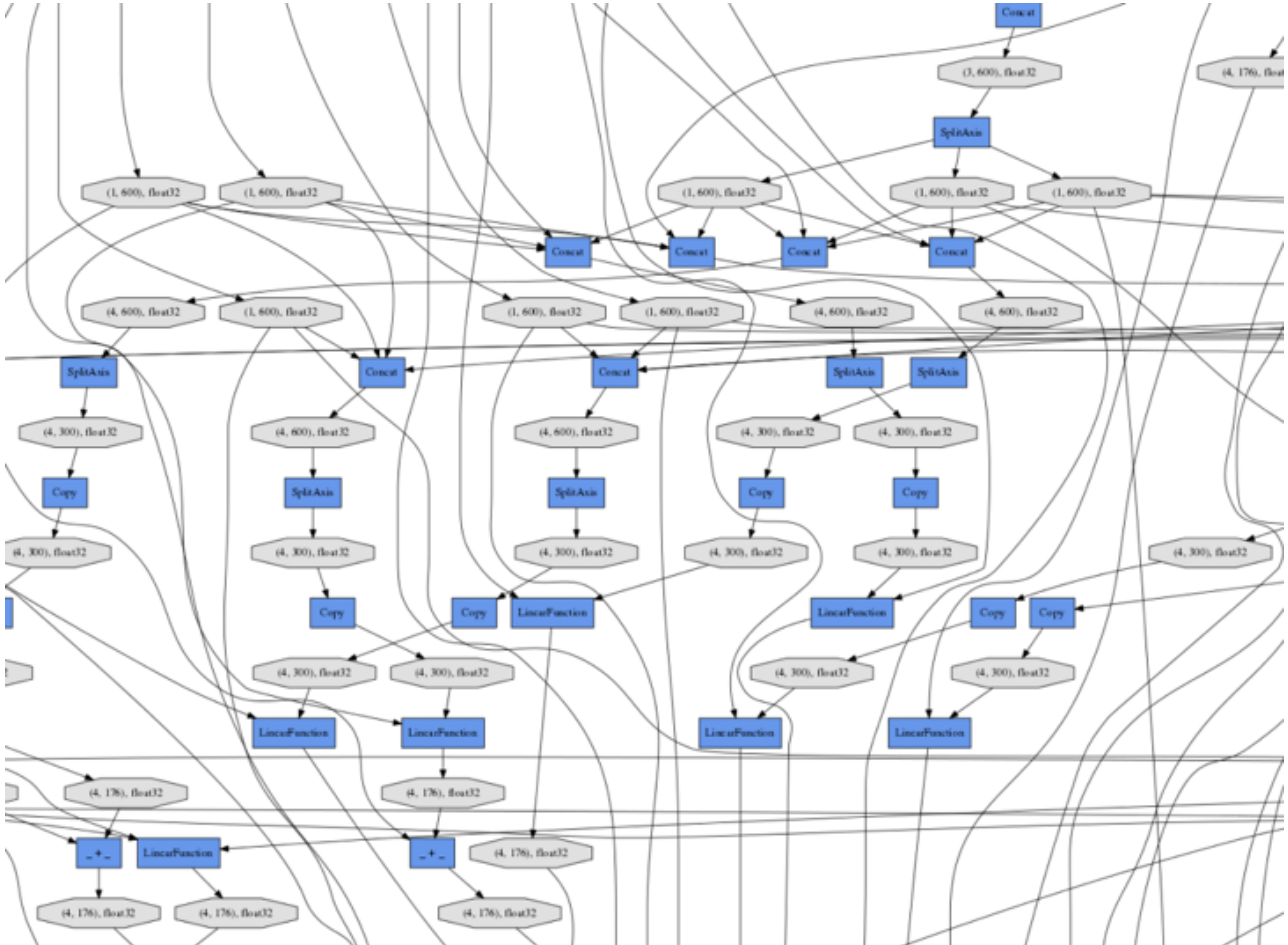
- For simple neural nets, the computation graph will be like:



Computational graphs of NNs

- For simple neural nets, the computation graph will be like:

- For larger models, the computation graph will be like:
    - TensorFlow & PyTorch automatically constructs these for you.
    - Still, these will be DAGs (directed acyclic graphs)



Concluding remarks

- This "backpropagation" requires a lot of memory!
  - **Rule of thumb.** Additional memory $\approx 2 \cdot$ Model size
  - **Gradient Checkpointing.** Re-compute activations when needed
- Gradients of some activations are cheaper to compute/store
  - e.g., ReLU
- If you are interested, check out the keyword "Automatic Differentiation"
  - https://arxiv.org/abs/1502.05767

Next up

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

Cheers