15. Backpropagation

EEC454 Introduction to
Machine Learning Systems

2023 Fall, Jaeho Lee

Optimizing Neural Networks

- **Today.** How to optimize the parameters of neural networks.

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

  - Here, the parameters are **weights & biases**:

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

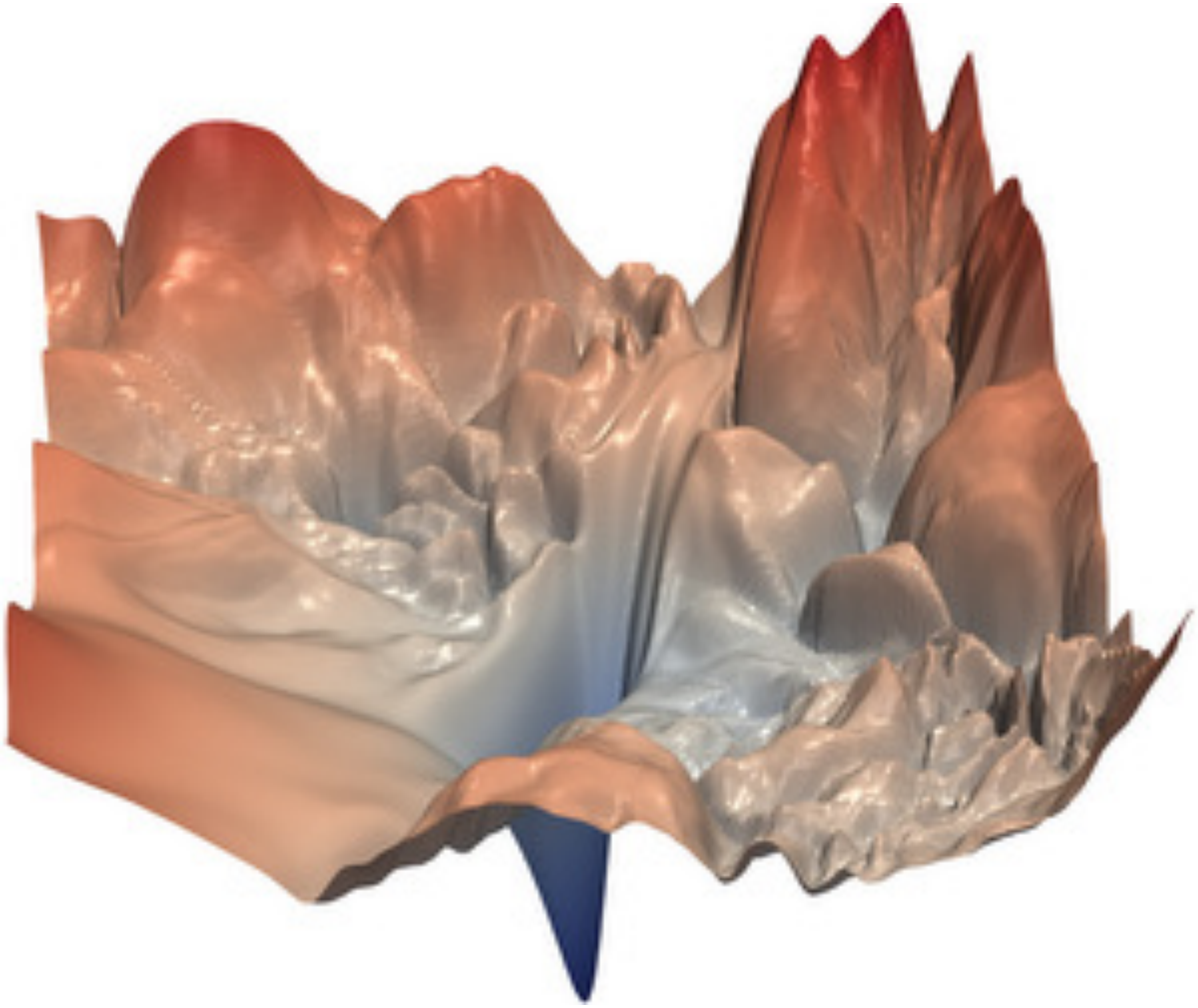
  - Again, the goal is to minimize the *empirical risk*:

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

Image source: Li et al., “Visualizing the Loss Landscape of Neural Nets,” NeurIPS 2018

Problem

- The loss landscape $L(\theta)$ is too irregular!



Recap: Gradient Descent

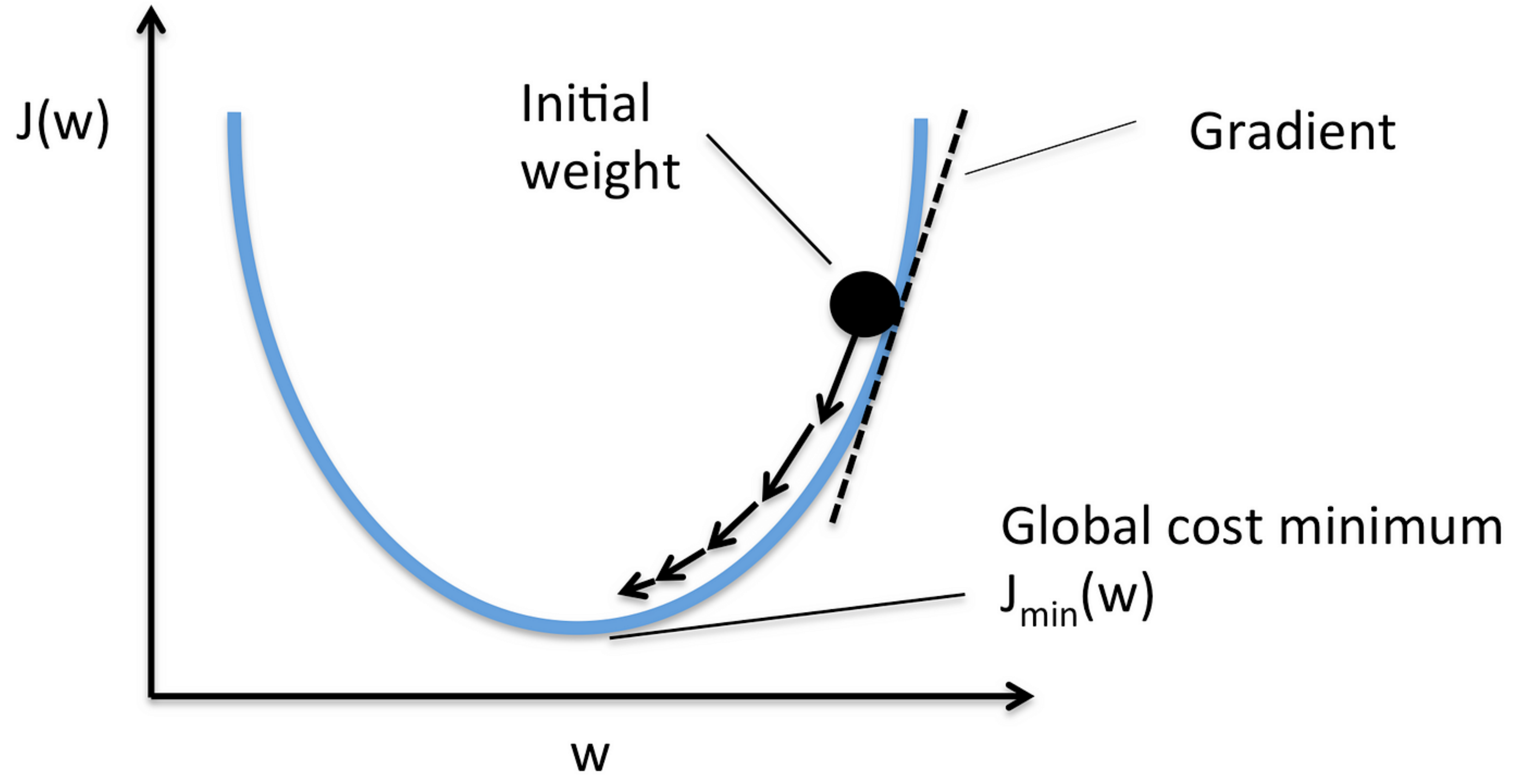
Gradient Descent

- Fortunately, simply performing **gradient descent** works well.
- **Idea.** Iteratively update θ in a direction the loss decreases.

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

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

Direction of **fastest increase**



SGD

- Computing $\nabla_\theta L(\theta)$ is expensive!
  - Need to look at the whole dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$

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

Recall the **chain rule**:

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

  - Requires computation of the per-sample gradient

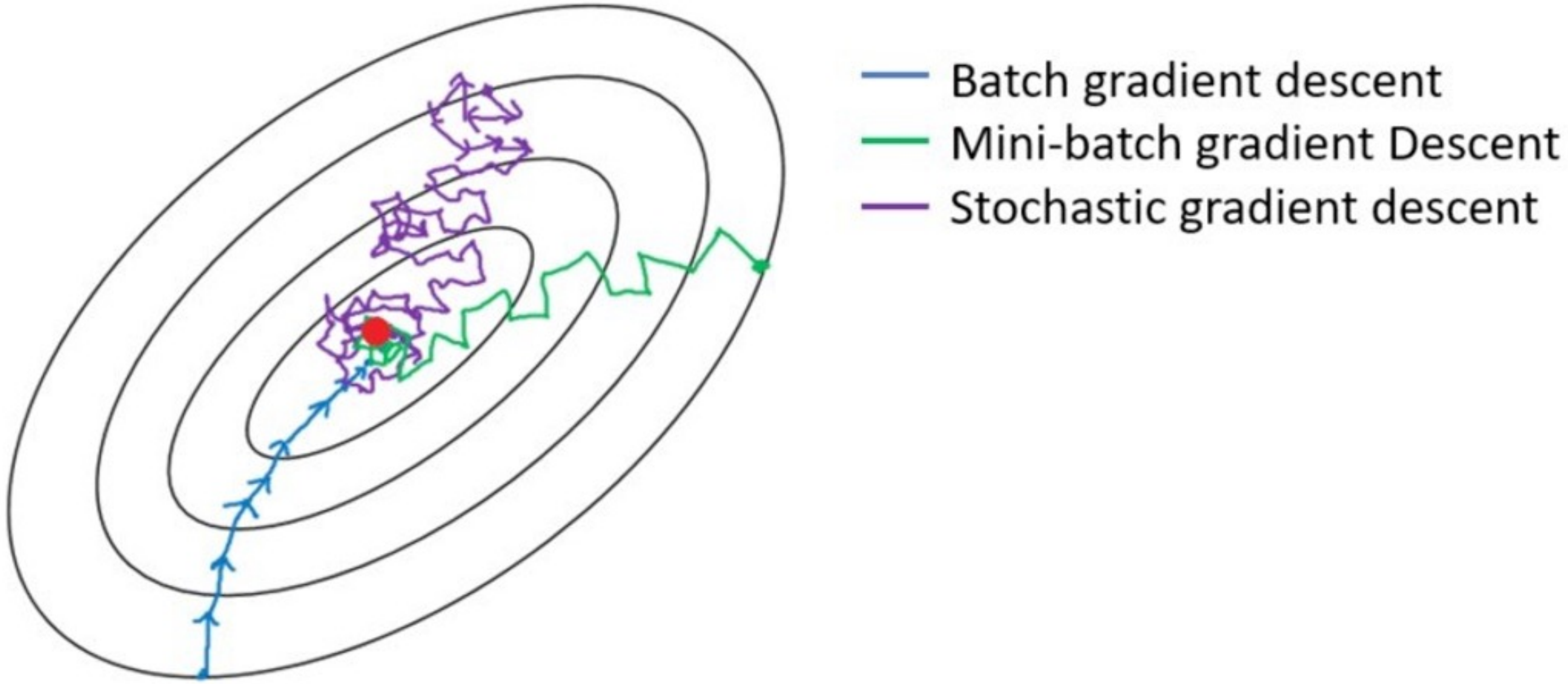
$$\nabla_\theta\Big(\ell(y, f_\theta(\mathbf{x}))\Big) = \frac{\partial \ell(y,z)}{\partial z}(f_\theta(\mathbf{x})) \cdot \nabla_\theta f_\theta(\mathbf{x})$$

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

Prediction gradient (heavy!)

SGD

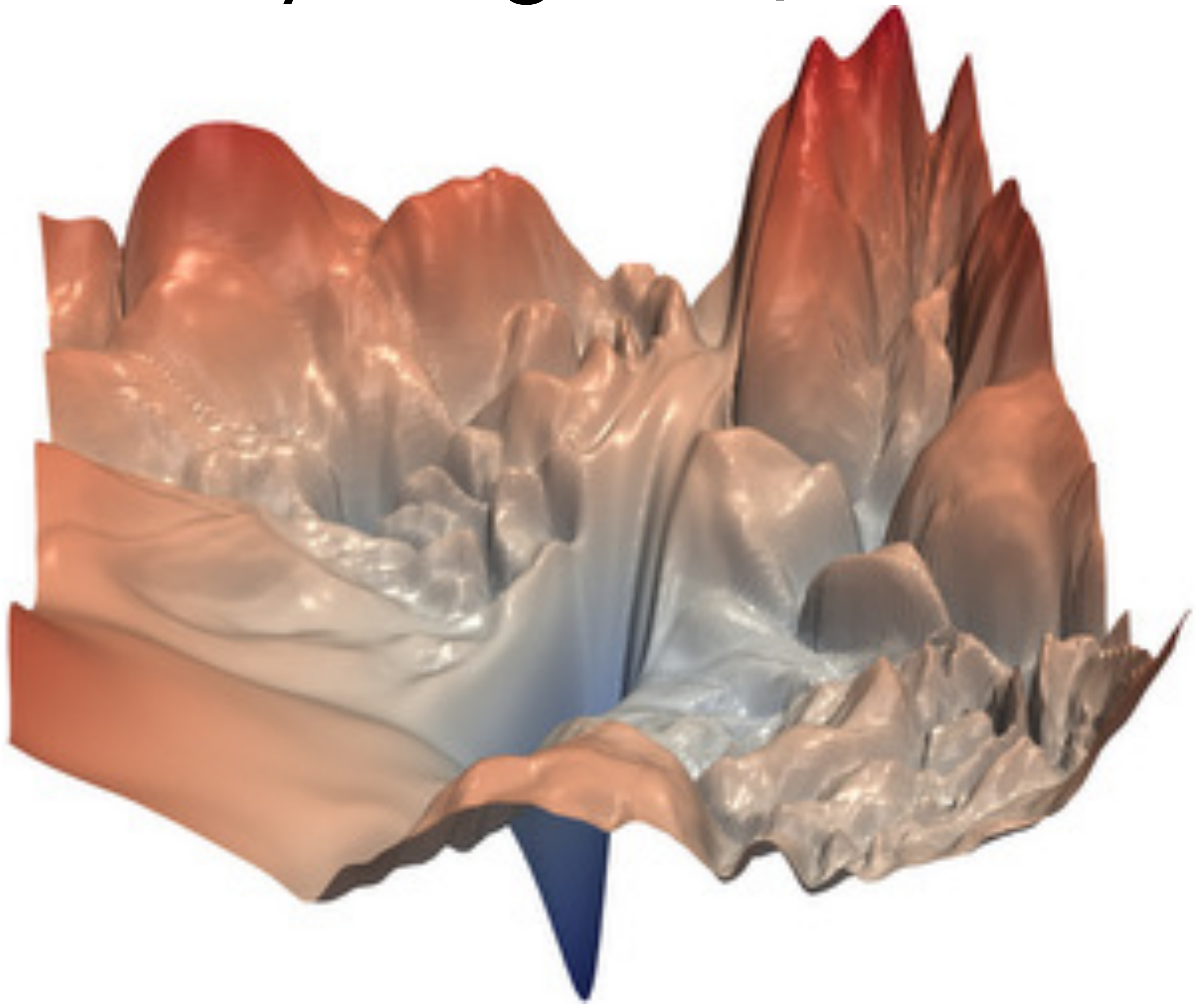
- **Stochastic GD.** Reduces this cost by looking at small number of randomly drawn samples at each time step.
  - **SGD (narrow).** Single sample at a time.
  - **Mini-Batch GD.** A batch of samples.



Computing $\nabla_\theta f_\theta(\mathbf{x})$

Gradient computation

- **Goal.** Compute $\nabla_\theta f_\theta(\mathbf{x})$ for some $\mathbf{x}, \theta$...... but how?
  - The parameter θ is high-dimensional (billions~trillions)
  - The function $f_\theta(\mathbf{x})$ is very irregular (continuous but nonconvex)



Method 1. Numerical Method

- **Idea.** The gradient is defined as a collection of derivatives:

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

Each derivative can be numerically computed as

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

Method 1. 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

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

Method 1. Numerical Method

current W:	W + h (second dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + 0.0001,	0.6,
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 1.25353	

(1.25353 - 1.25347)/0.0001=0.6

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

Method 1. 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

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

Method 1. Numerical Method

- **Pros.**
  - Easy to implement.
- **Cons.**
  - Is only **approximate**
    - Cannot send $\epsilon \to 0$, due to finite precision.
  - Very **slow**
    - Requires at least $d + 1$ evaluations of $f_\theta(\mathbf{x})$ for $\theta \in \mathbb{R}^d$.

Method 2. Analytic Method

- **Idea.** Derive an analytic expression of the gradient.
  - For example, if

$$g(x) = \sin(5 \cdot \exp(x))$$

we know that the gradient will be

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

(how? we'll see more soon)

Method 2. Analytic Method

- **Pros.**
  - Exact.
  - Fast.
- **Cons.**
  - Needs careful implementation for complicated functions.
    - Need to check the correctness, using the numerical method (called gradient check)

Backpropagation

Chain rule

- **Q.** How to analytically derive $\nabla_\theta f_\theta(\mathbf{x})$ for complicated functions?

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

Chain rule

- **Q.** How to analytically derive $\nabla_\theta f_\theta(\mathbf{x})$ for complicated functions?

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

- **A.** 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.
  - We can use chain rule to combine these.

Chain rule: Example

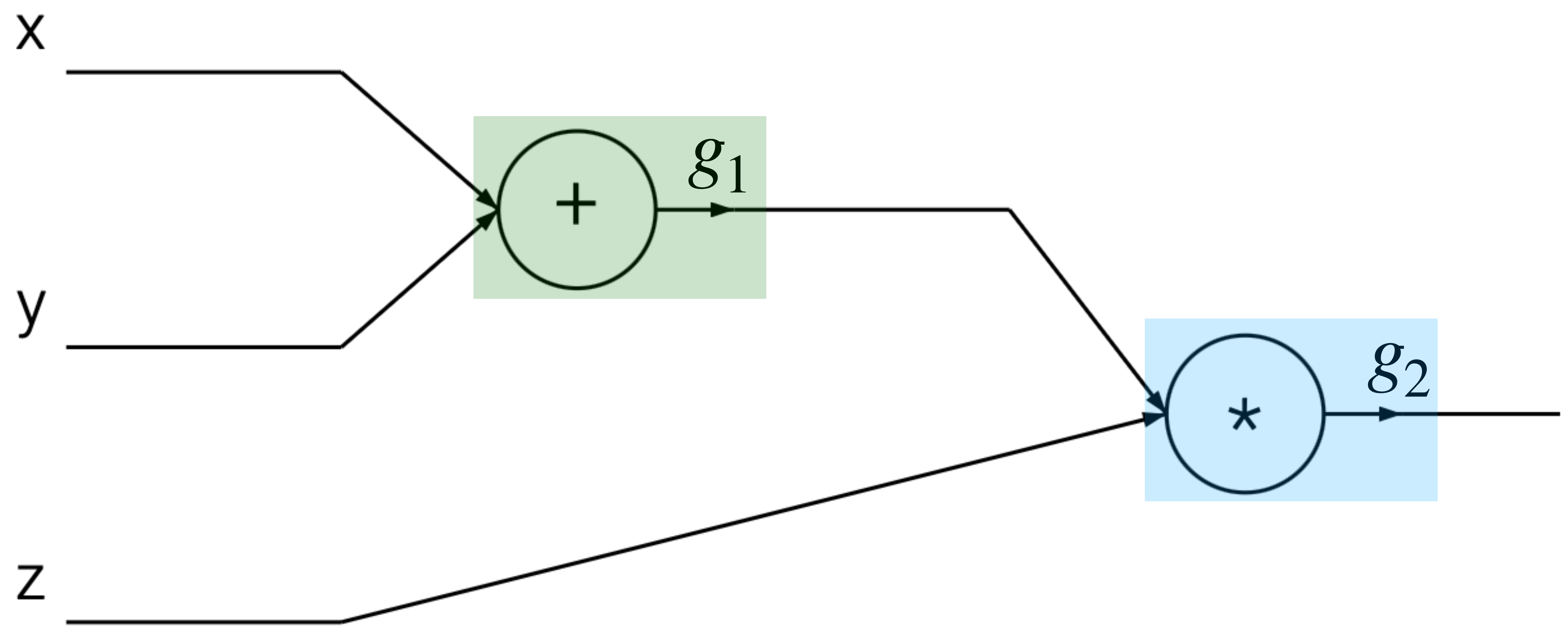
- Consider a function

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

- This can be viewed as a composition of two **elementary operations**:

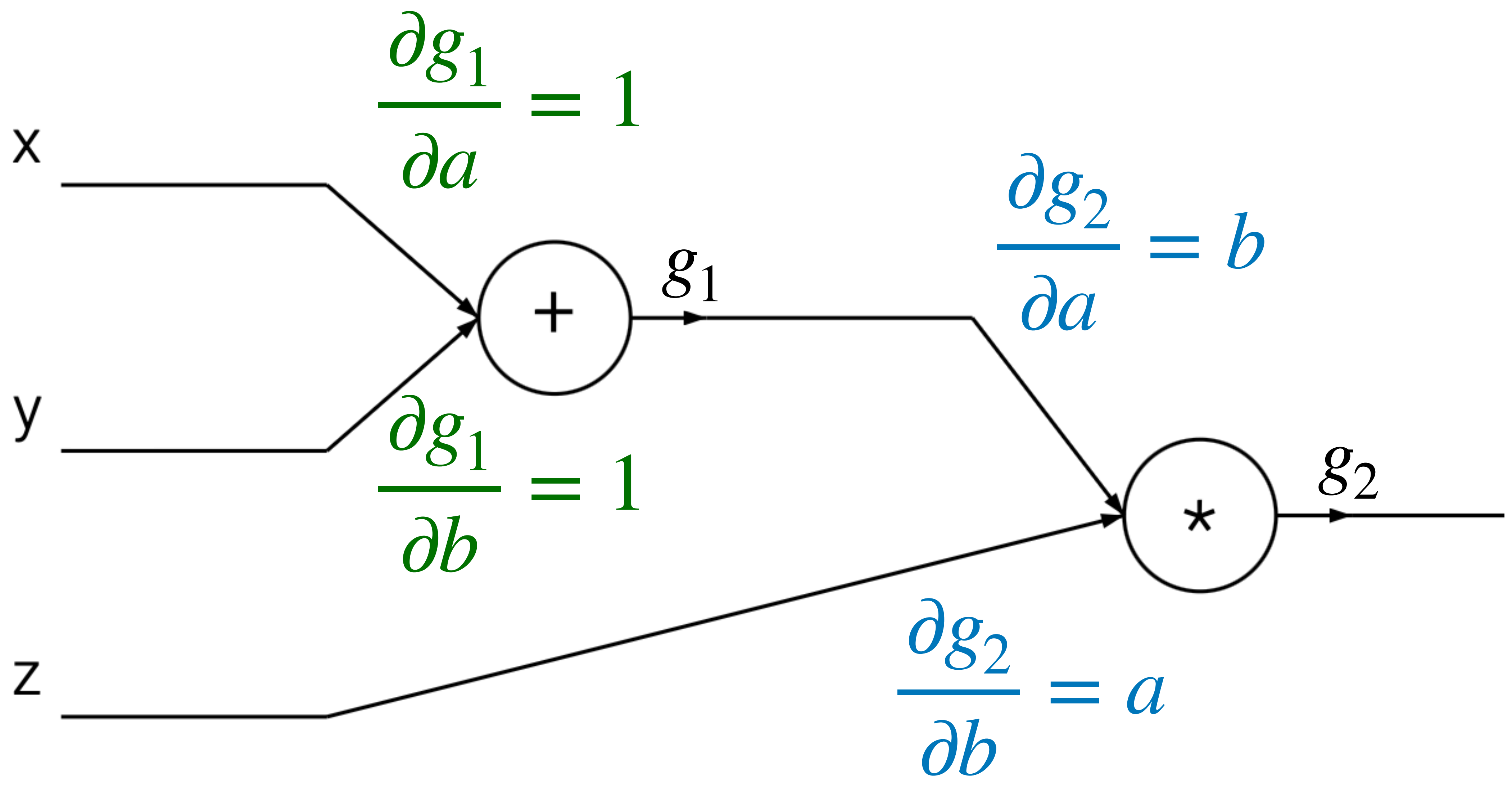
$$g(x, y, z) = g_2(g_1(x, y), z)$$

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



Chain rule: Example

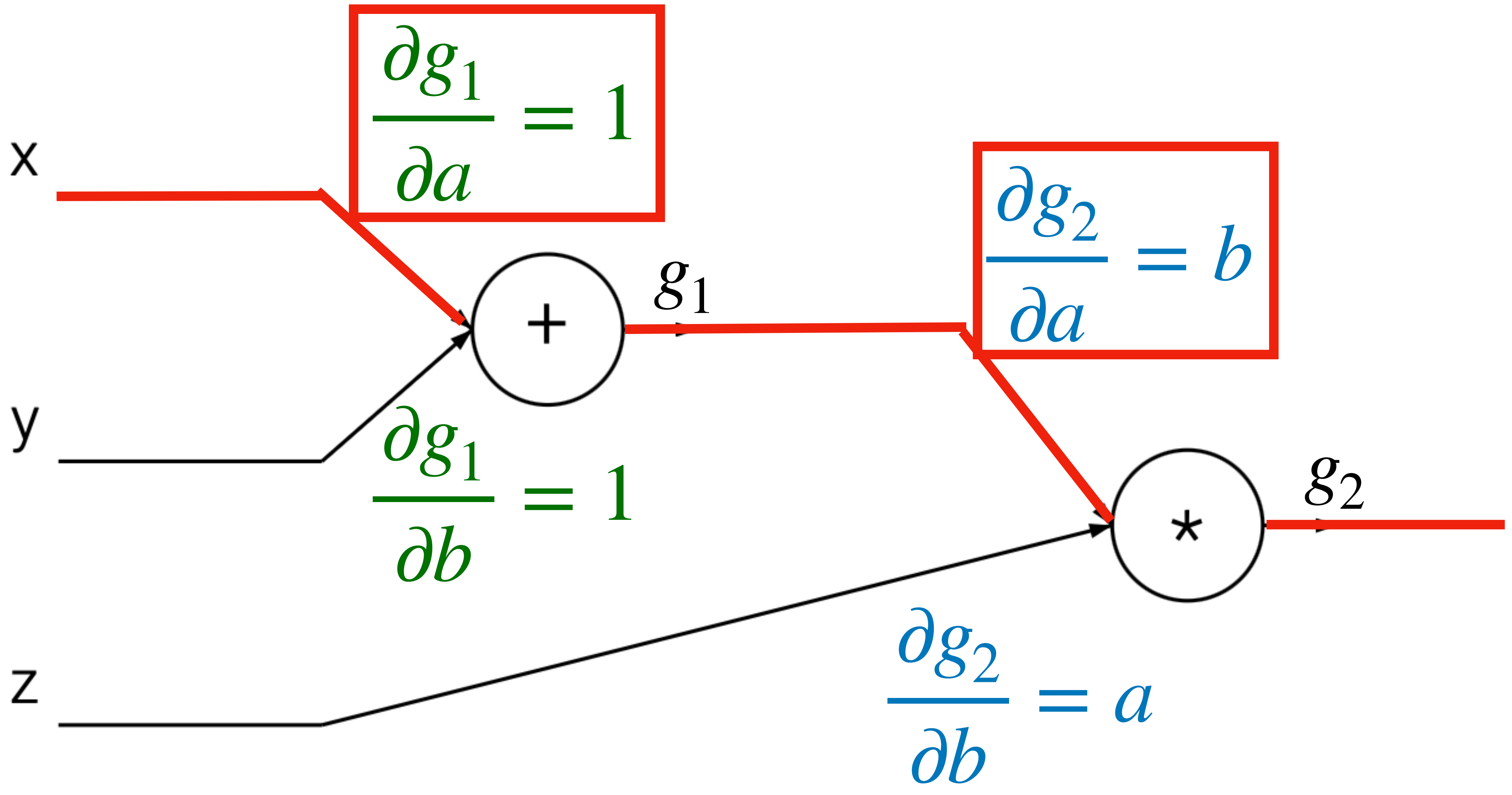
- Each elementary operation has easy-to-write gradients:
  - $\frac{\partial g_1}{\partial a} = 1, \frac{\partial g_1}{\partial b} = 1$
  - $\frac{\partial g_2}{\partial a} = b, \frac{\partial g_1}{\partial b} = a$



Chain rule: Example

- Chain rule tells you that:

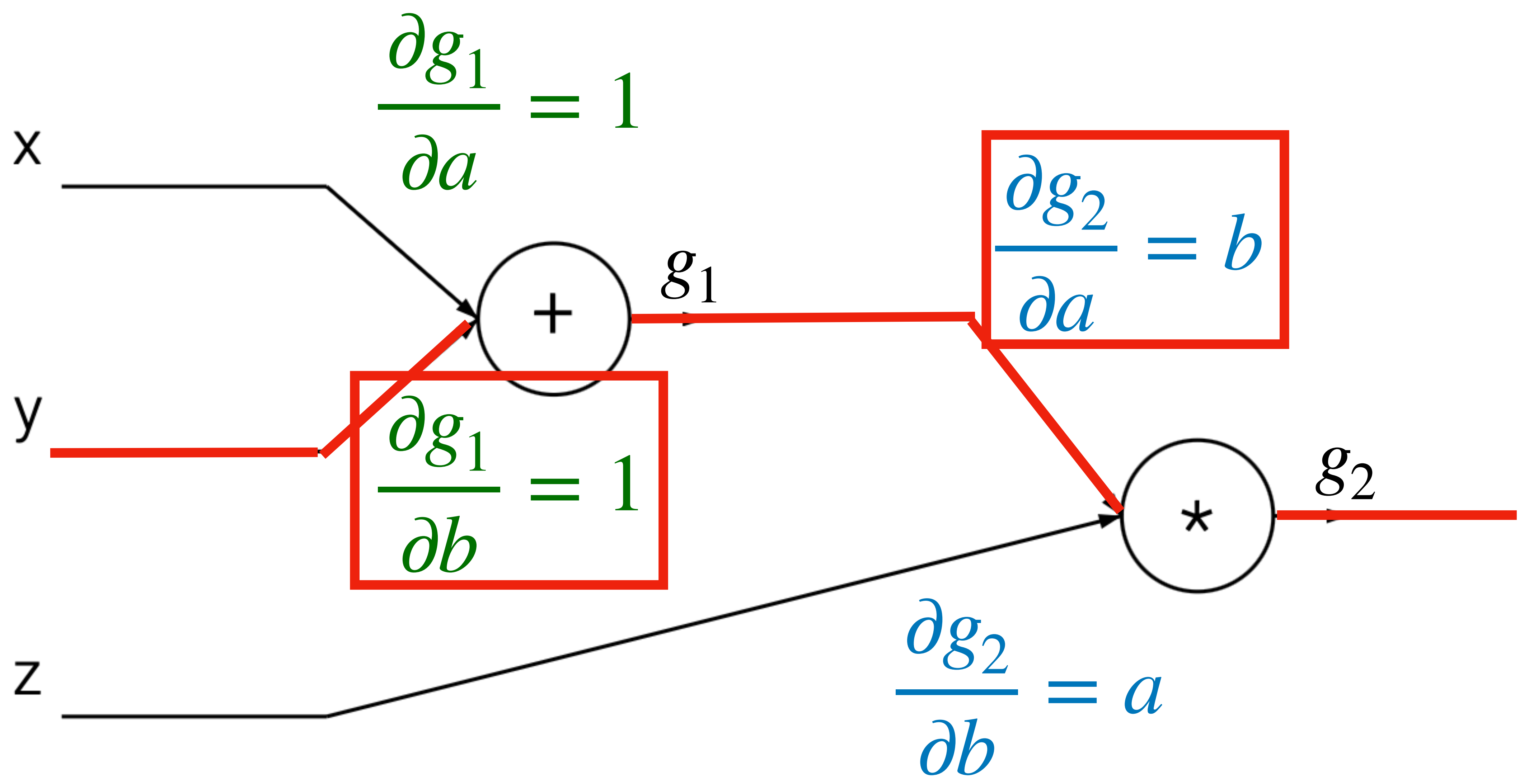
$$\frac{\partial g}{\partial x}(x, y, z) = \underbrace{\frac{\partial g_2}{\partial a}(g_1(x, y), z)}_{= z} \cdot \underbrace{\frac{\partial g_1}{\partial a}(x, y)}_{= 1}$$



Chain rule: Example

- Chain rule tells you that:

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

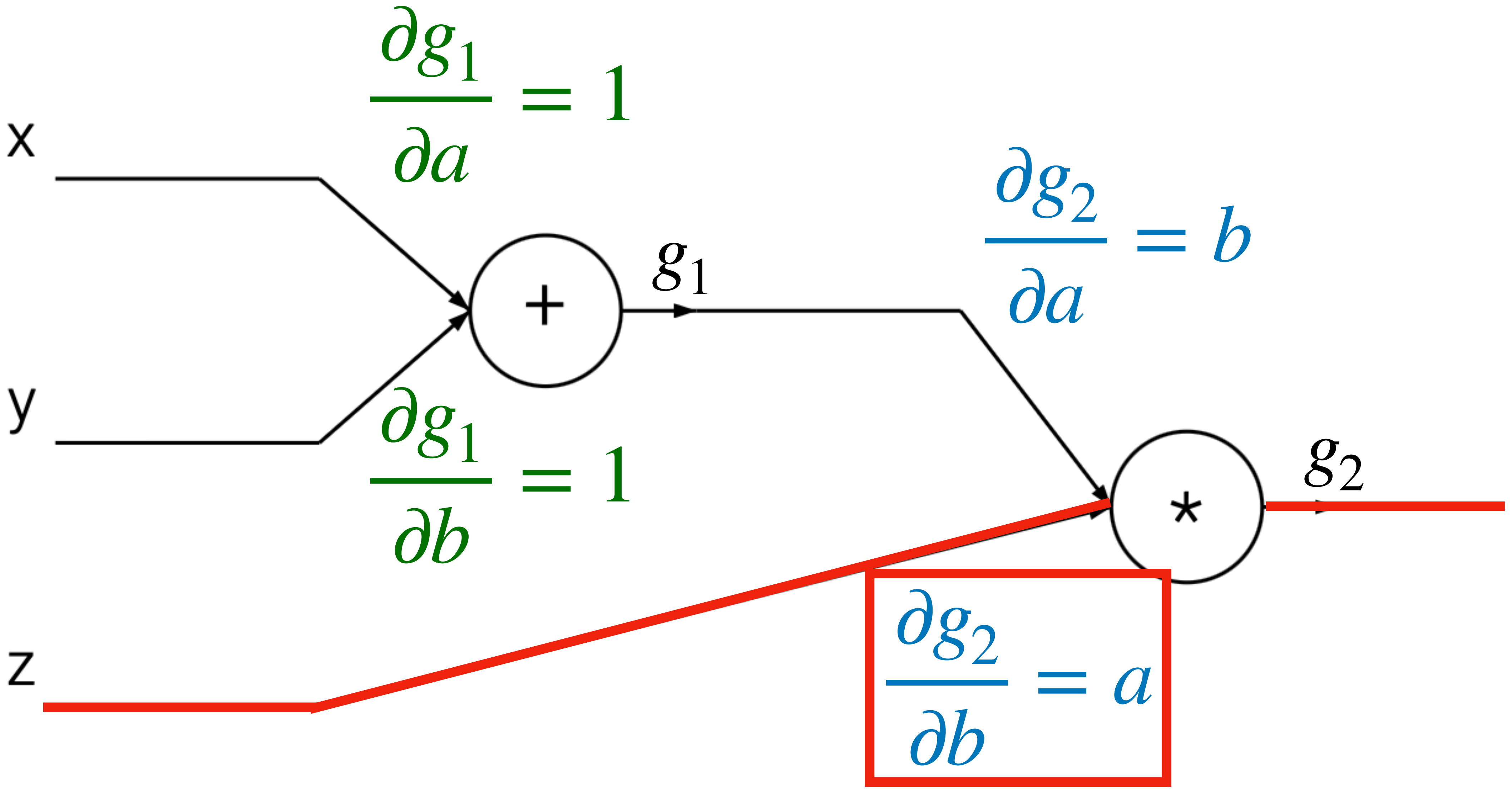


Chain rule: Example

- Chain rule tells you that:

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

$$= g_1(x, y)$$



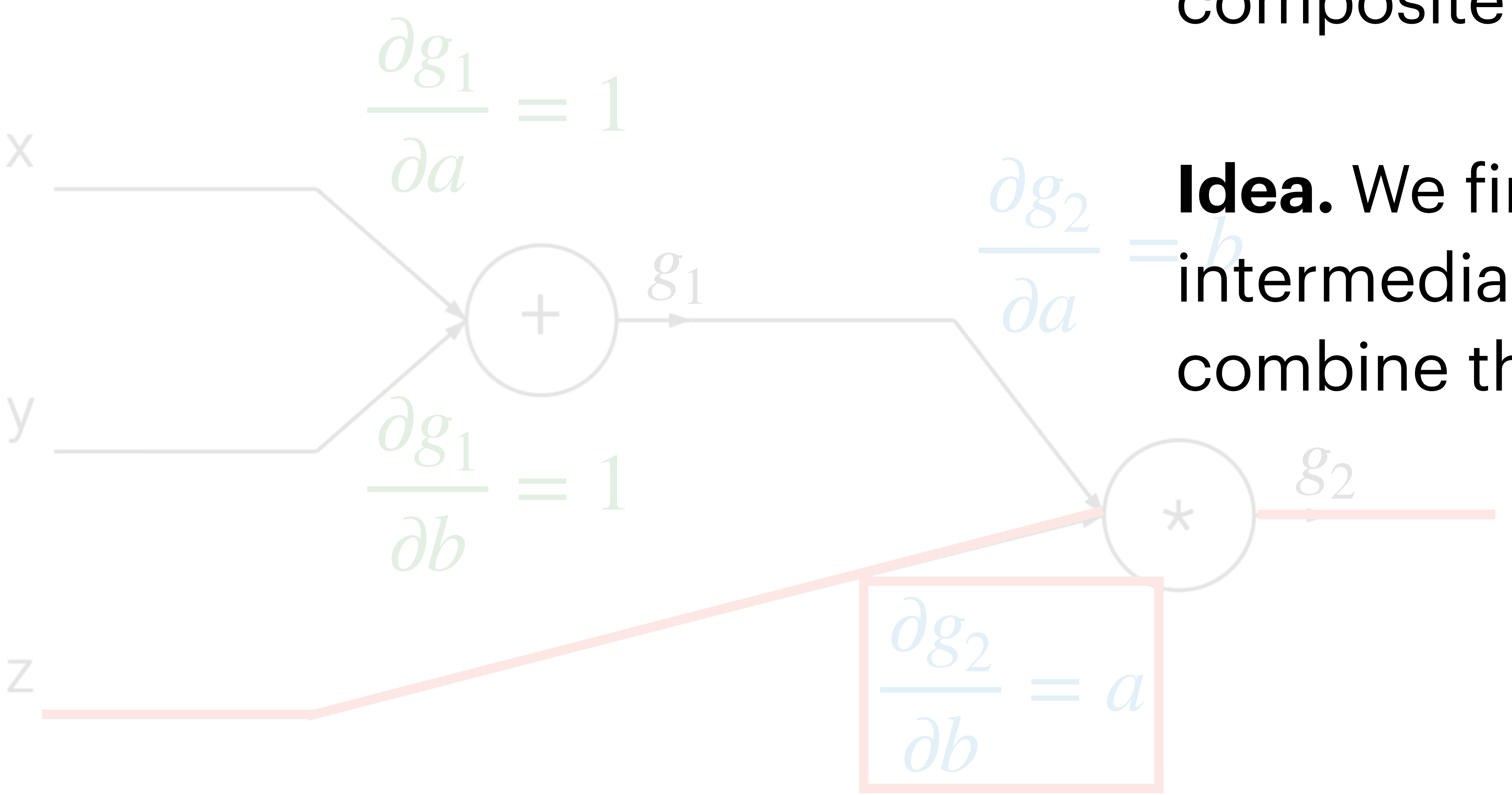
Chain rule: Example

- Chain rule tells you that:

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

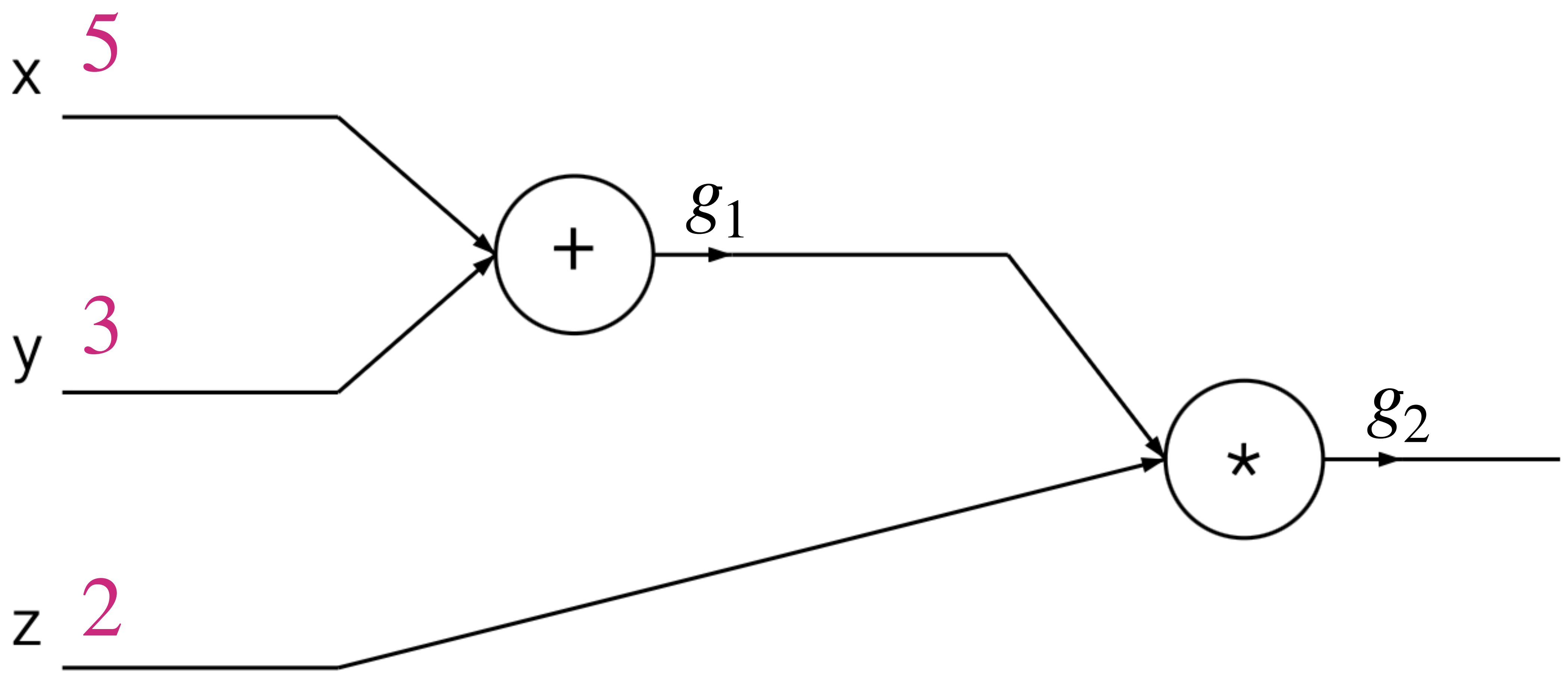
Observation. Computing gradients requires intermediate values of the composite function.

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



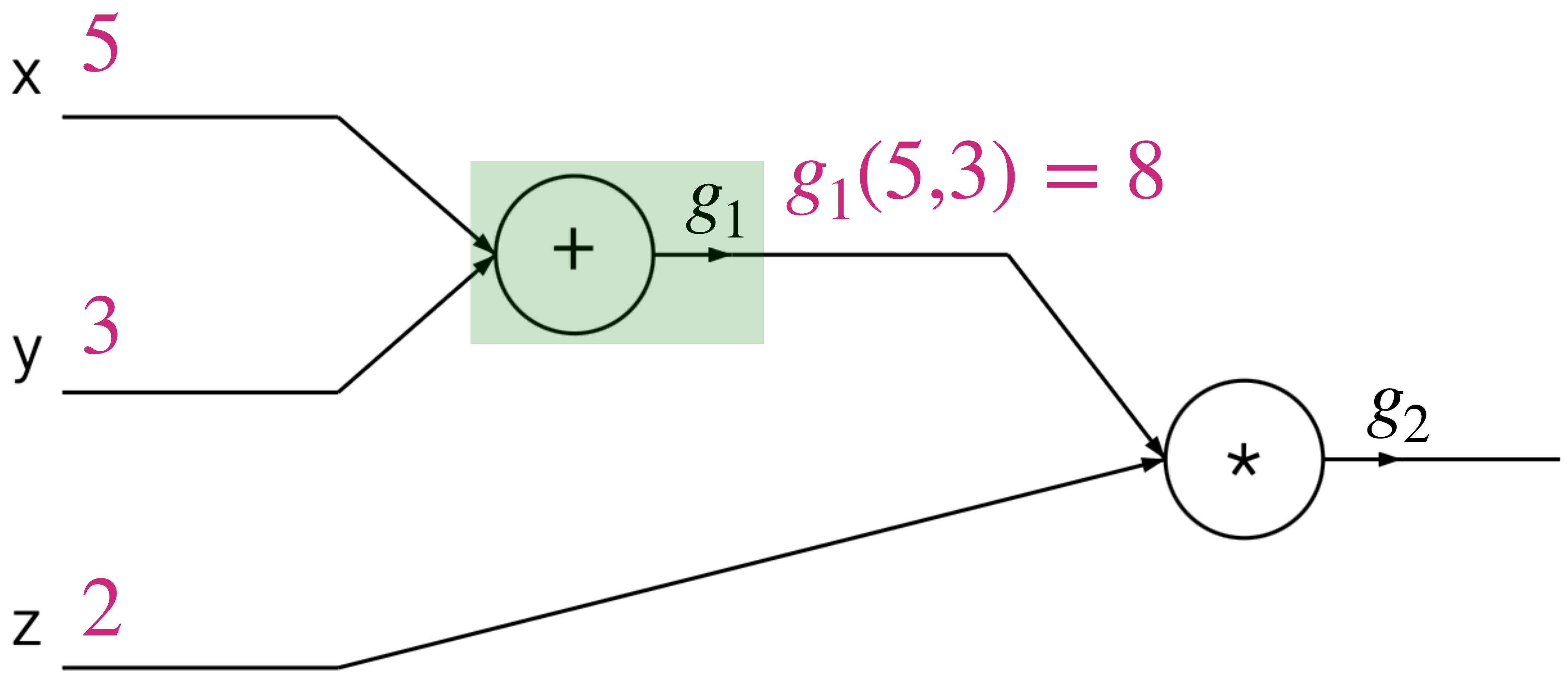
Neural Network Training

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



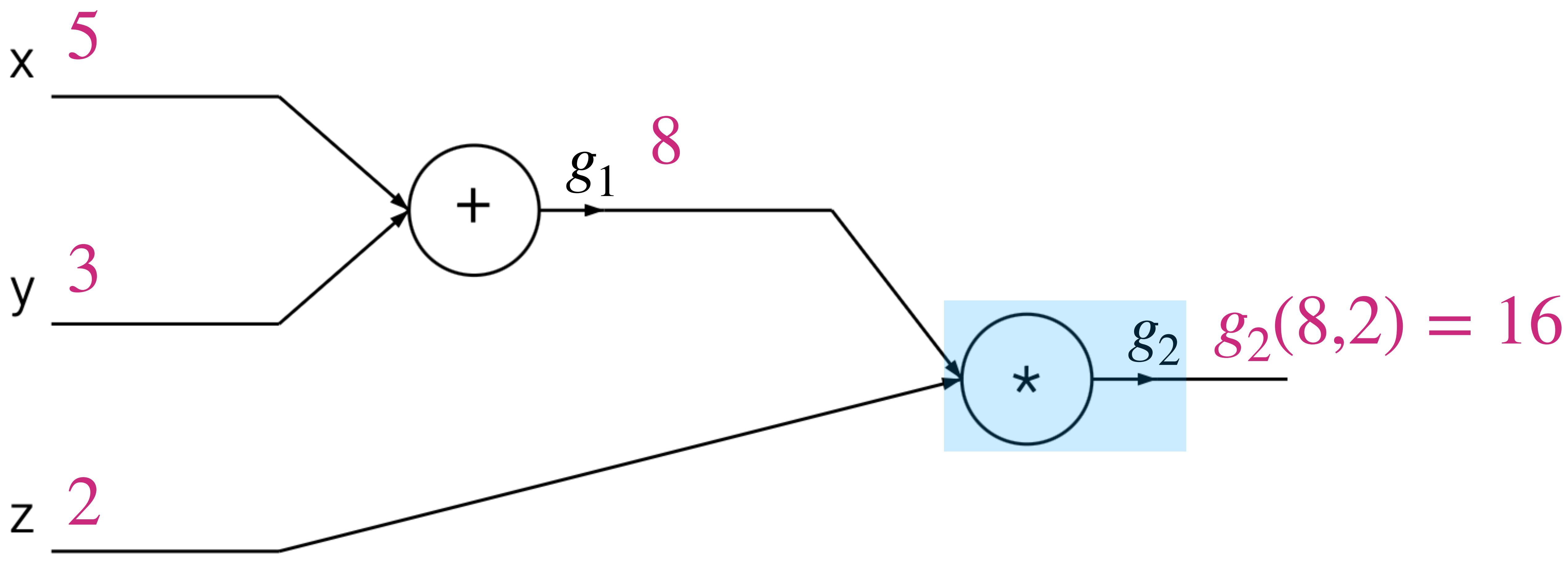
Neural Network Training

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



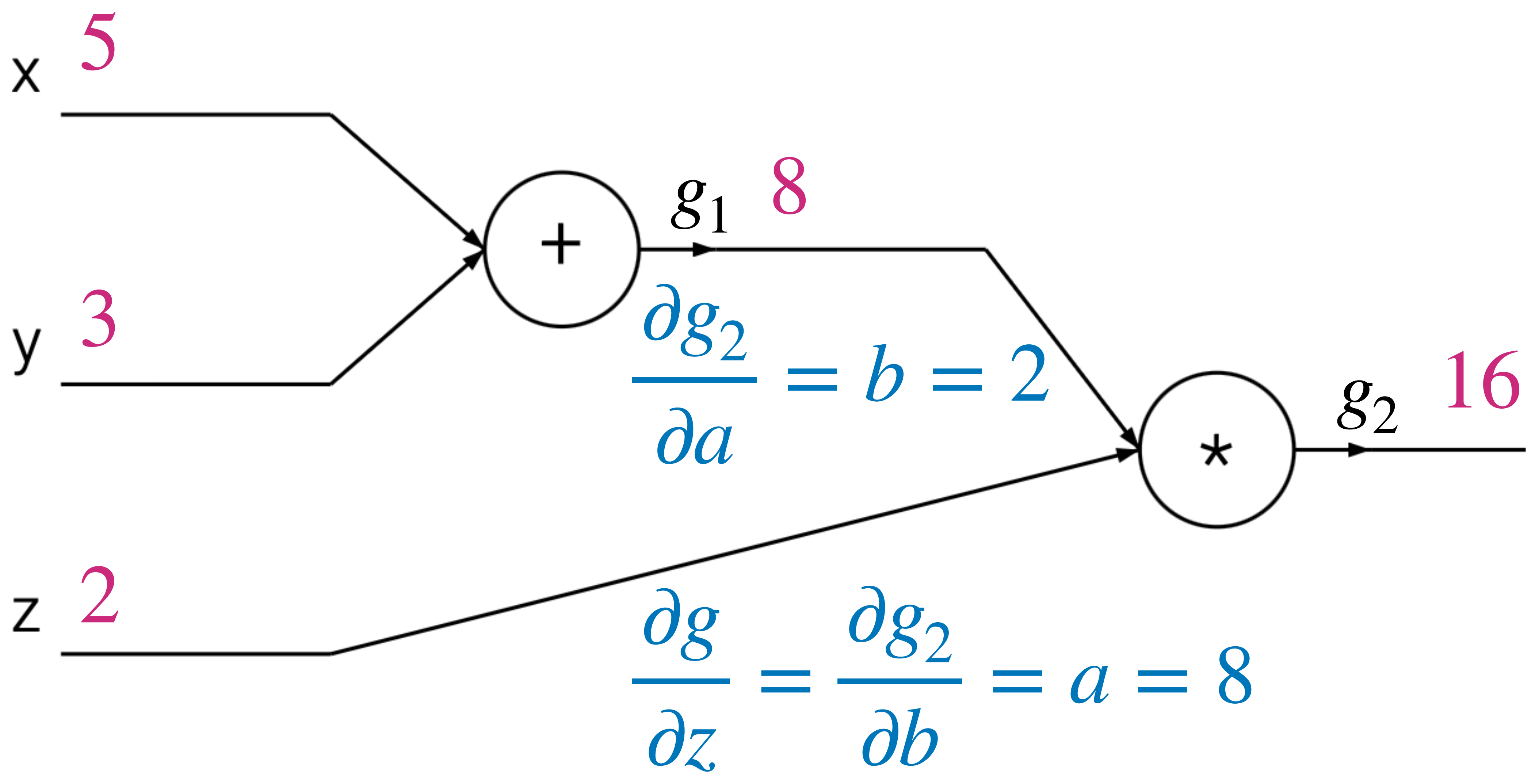
Neural Network Training

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



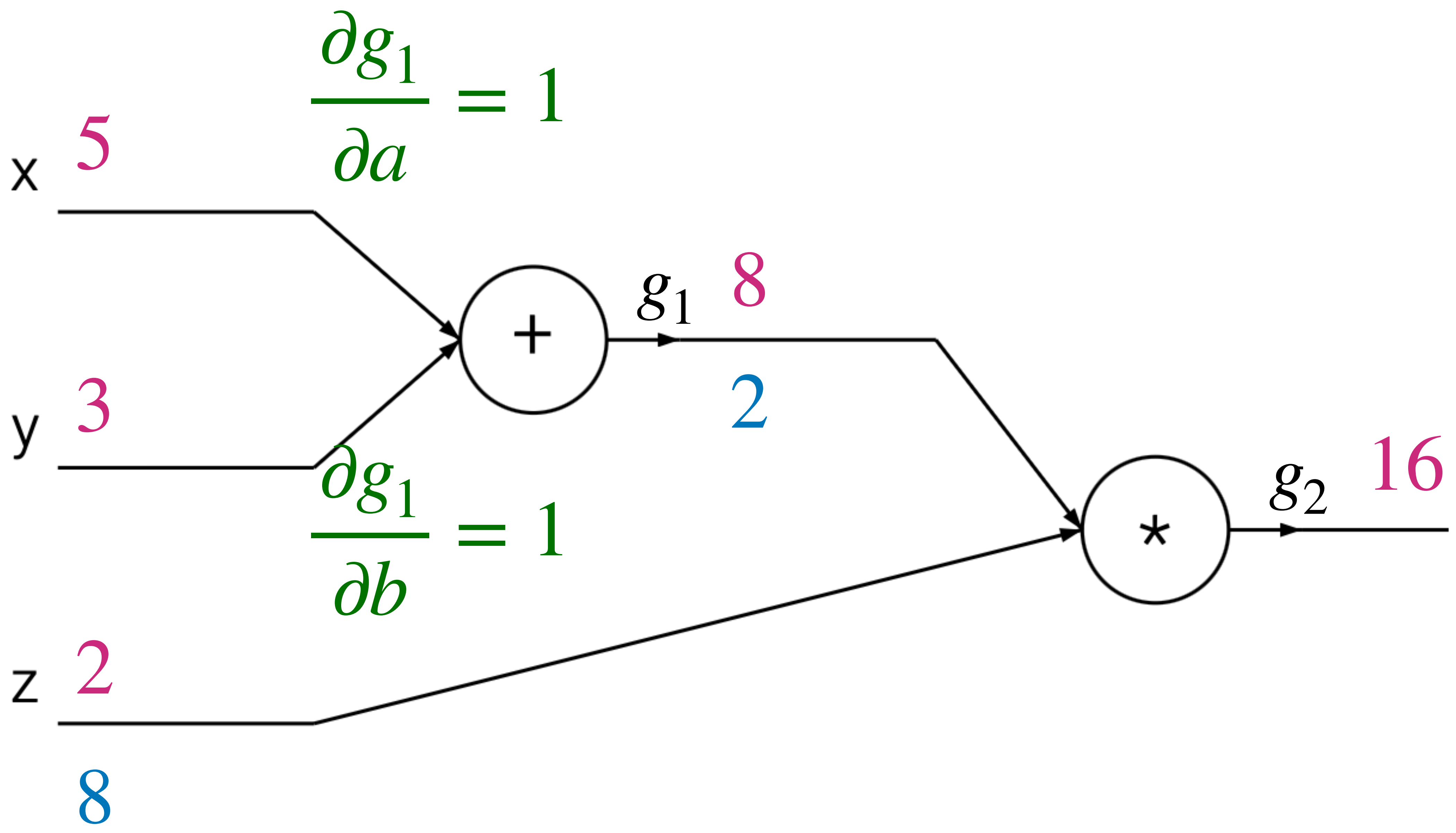
Neural Network Training

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



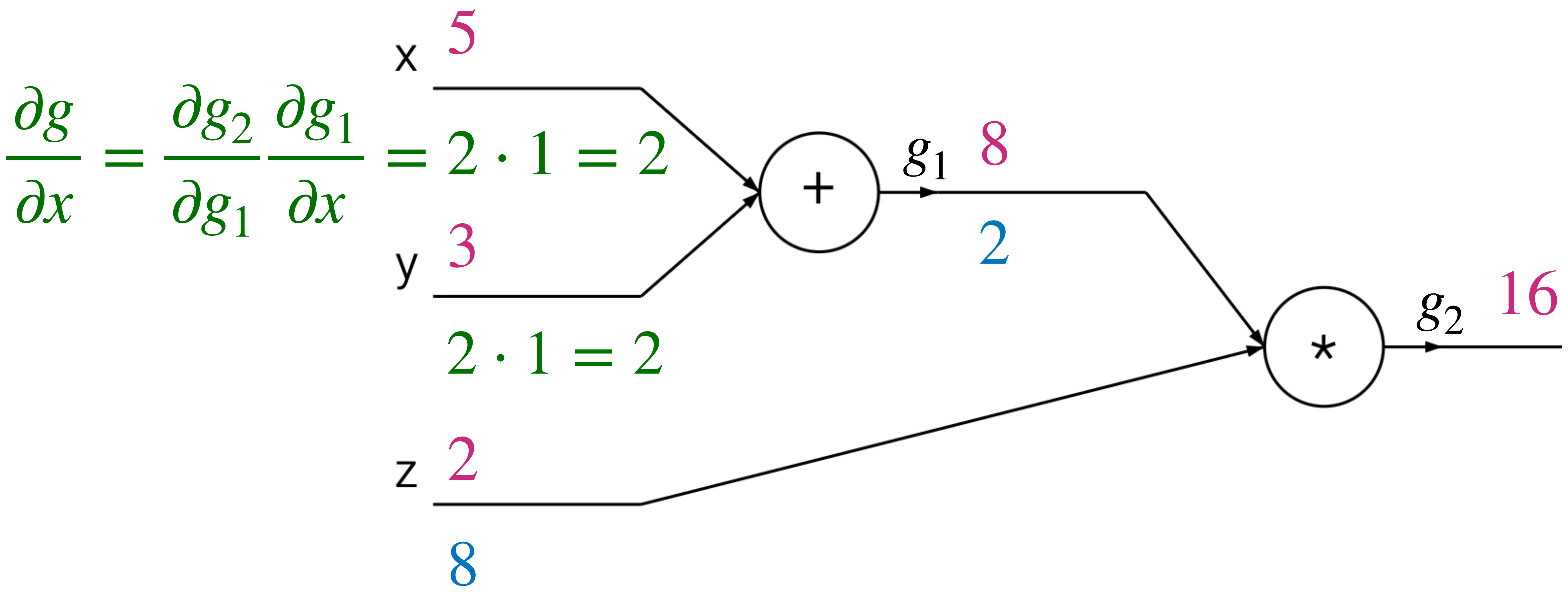
Neural Network Training

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



Neural Network Training

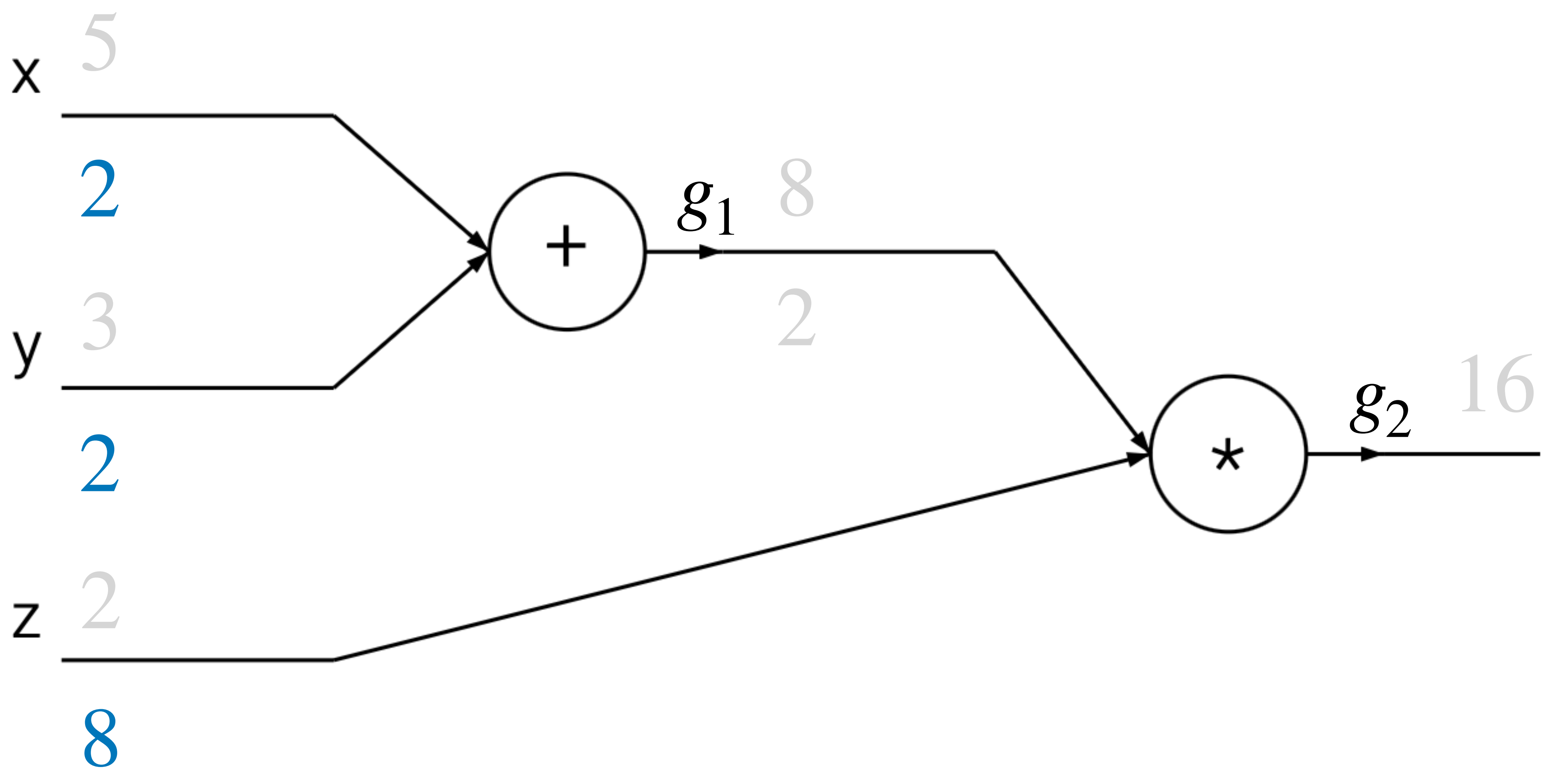
- Iteratively applies three steps:
  - **(2) Backward Pass.** Compute the gradient using stored values.
    - From output to input.



Neural Network Training

- Iteratively applies three steps:
  - **(3) GD.** Update the parameters.

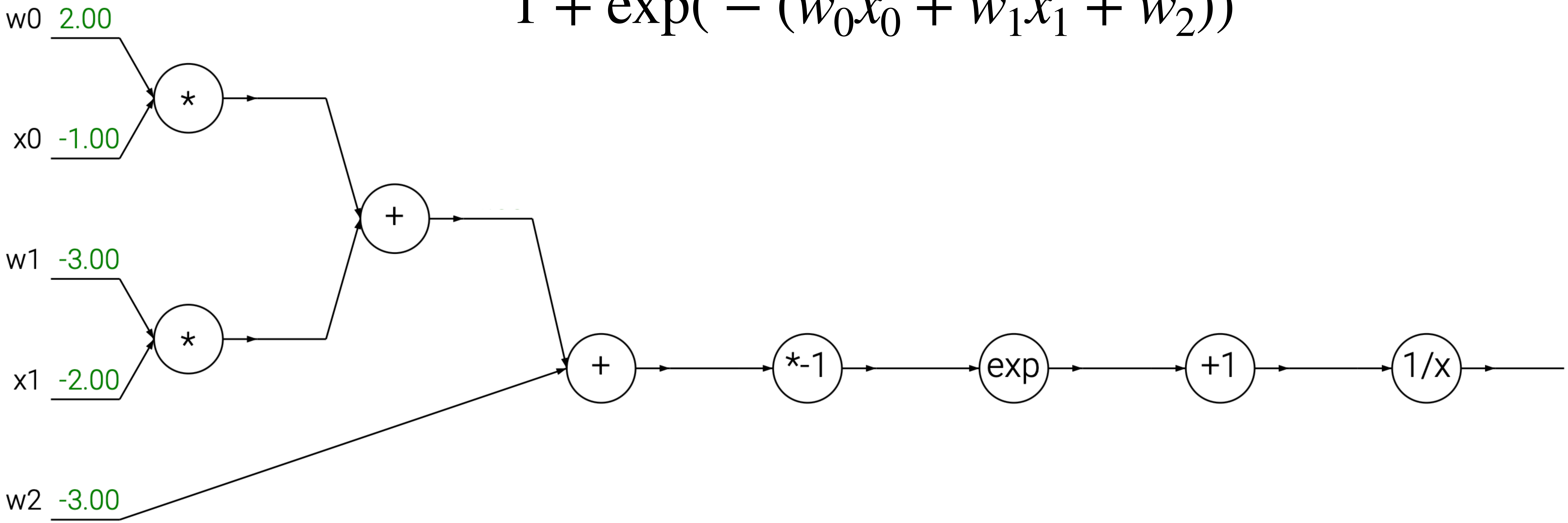
$$x \leftarrow x - \eta \cdot 2, \qquad y \leftarrow y - \eta \cdot 2, \qquad z \leftarrow z - \eta \cdot 8$$



Another example

- Consider a function

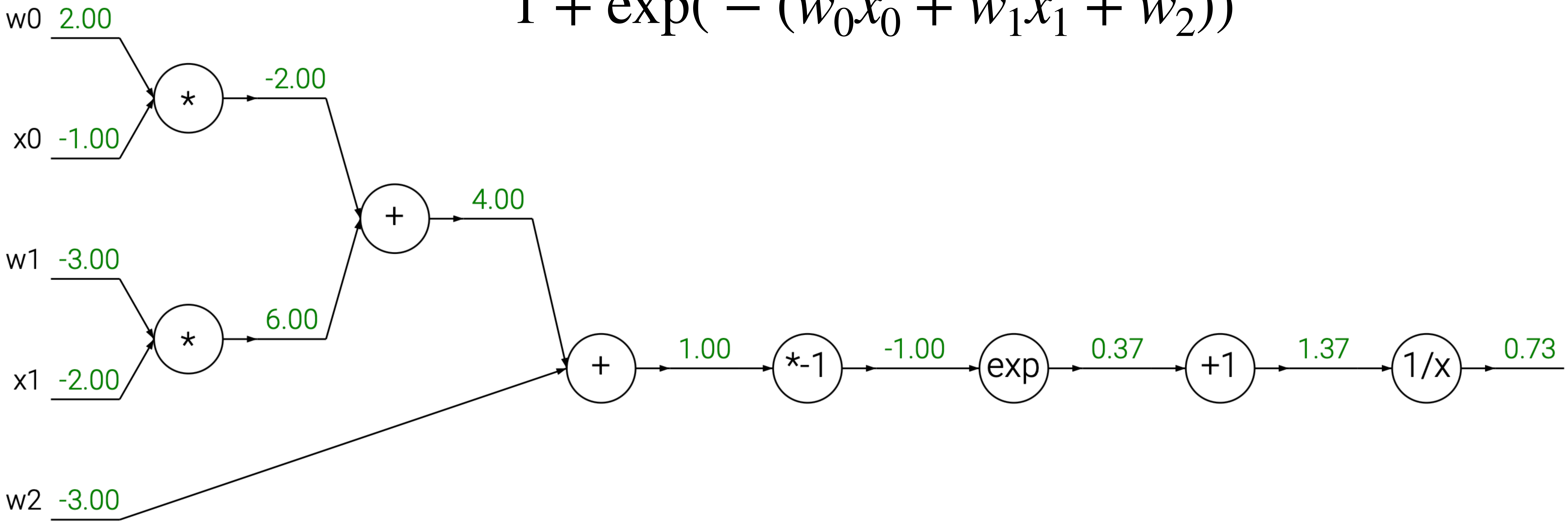
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

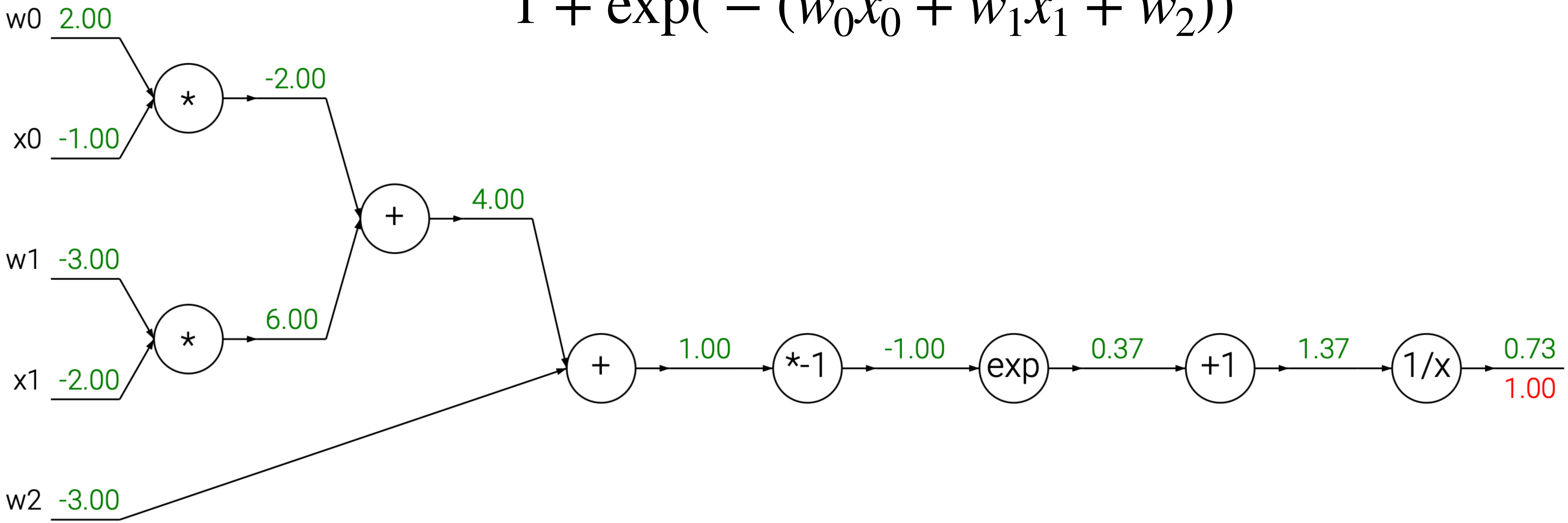
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

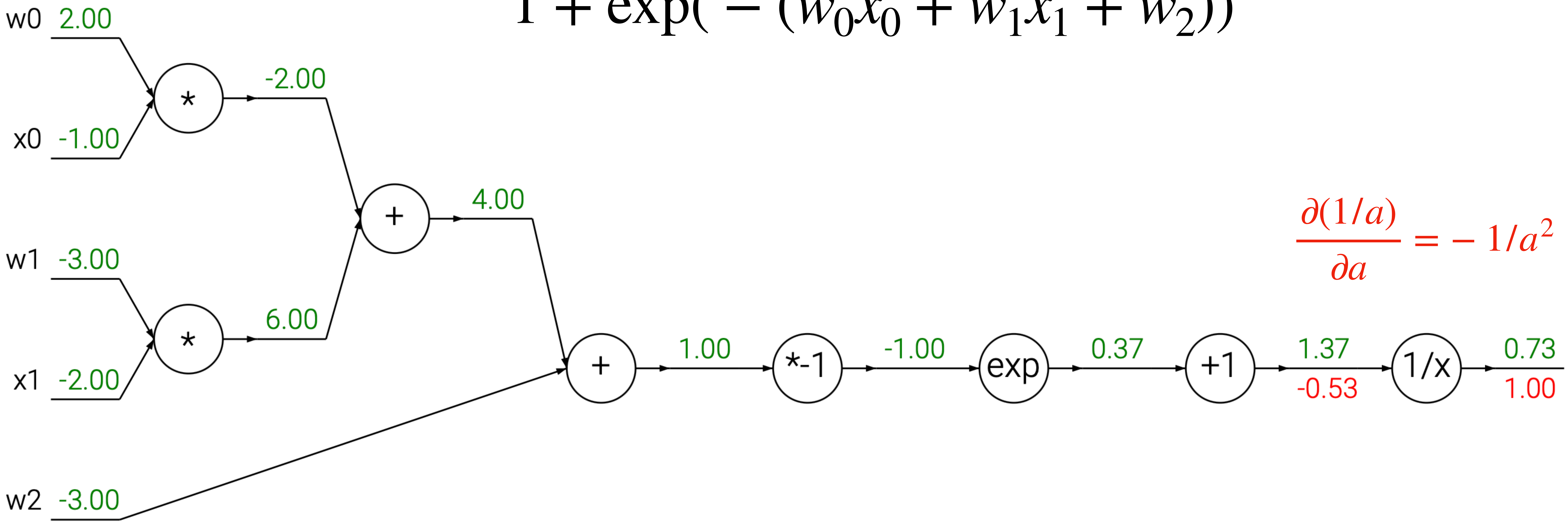
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0x_0 + w_1x_1 + w_2))}$$



Another example

- Consider a function

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

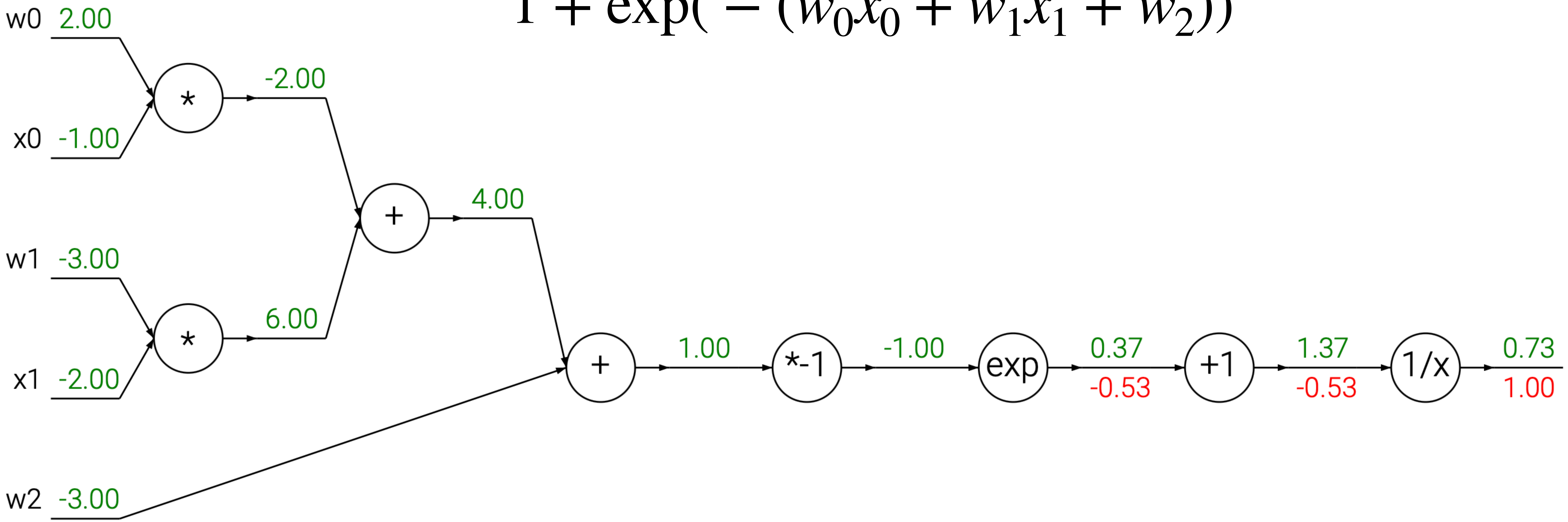


$$\frac{\partial(1/a)}{\partial a} = -1/a^2$$

Another example

- Consider a function

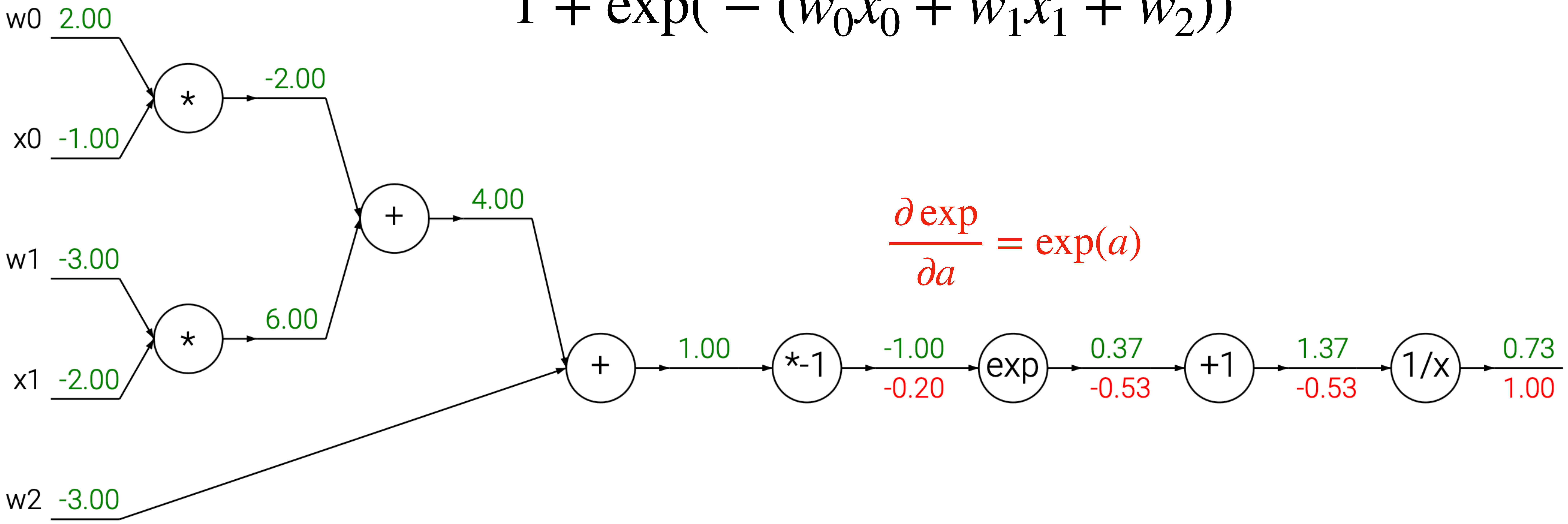
$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$



Another example

- Consider a function

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

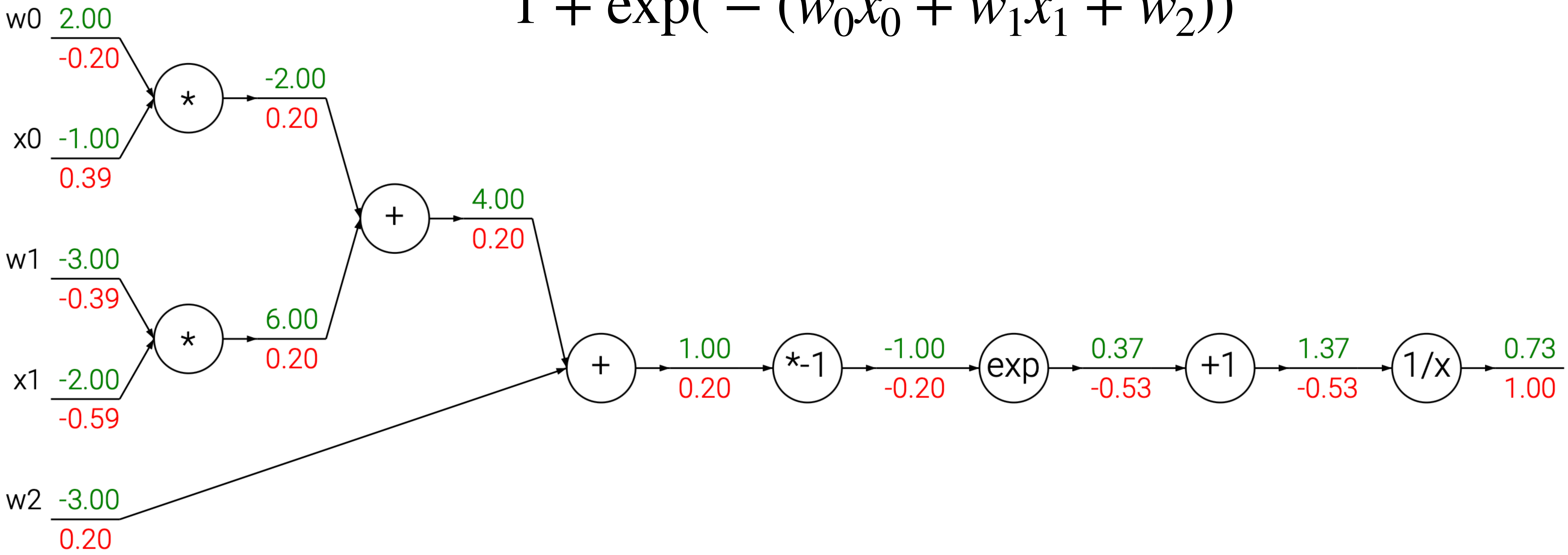


$$\frac{\partial \exp}{\partial a} = \exp(a)$$

Another example

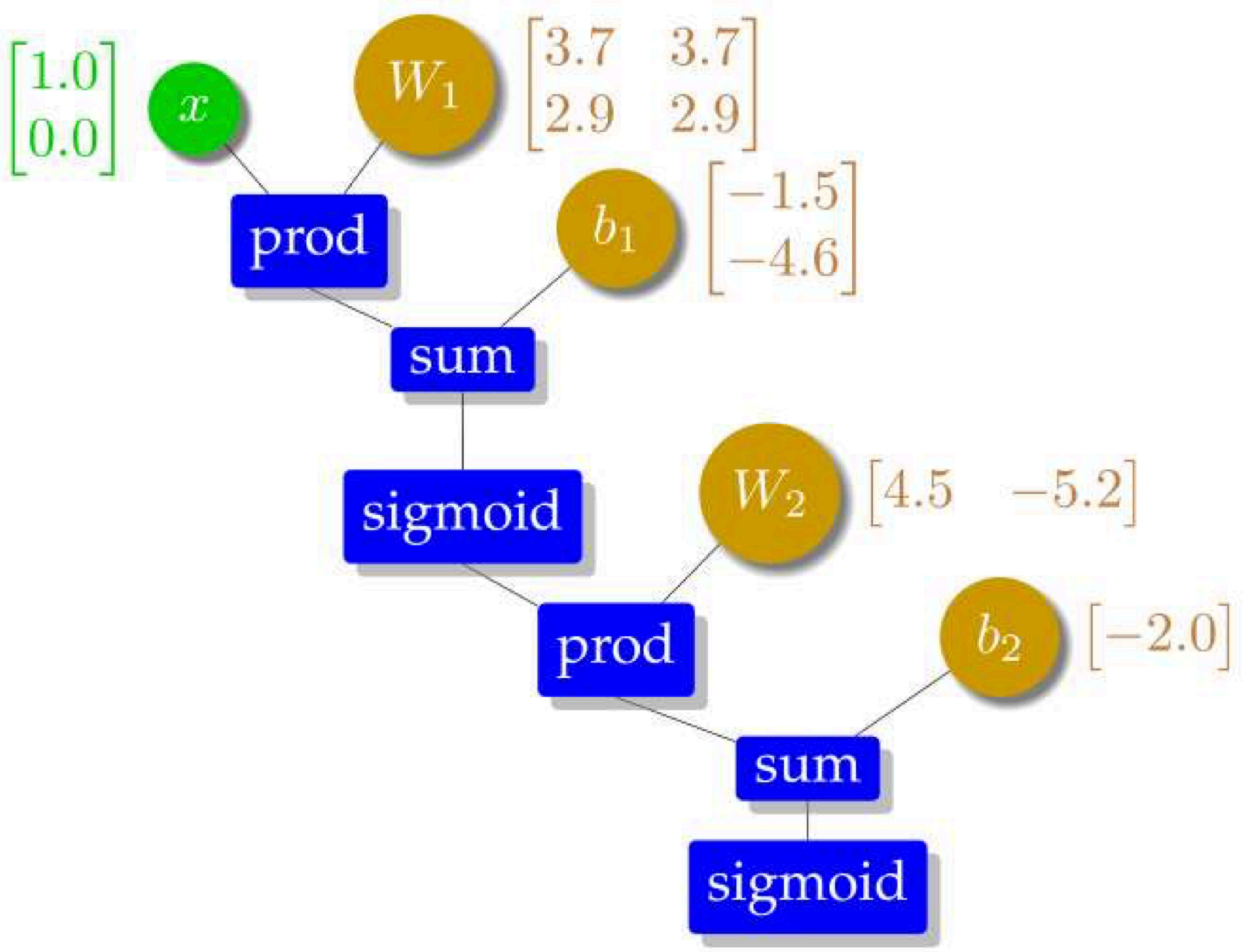
- Consider a function

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



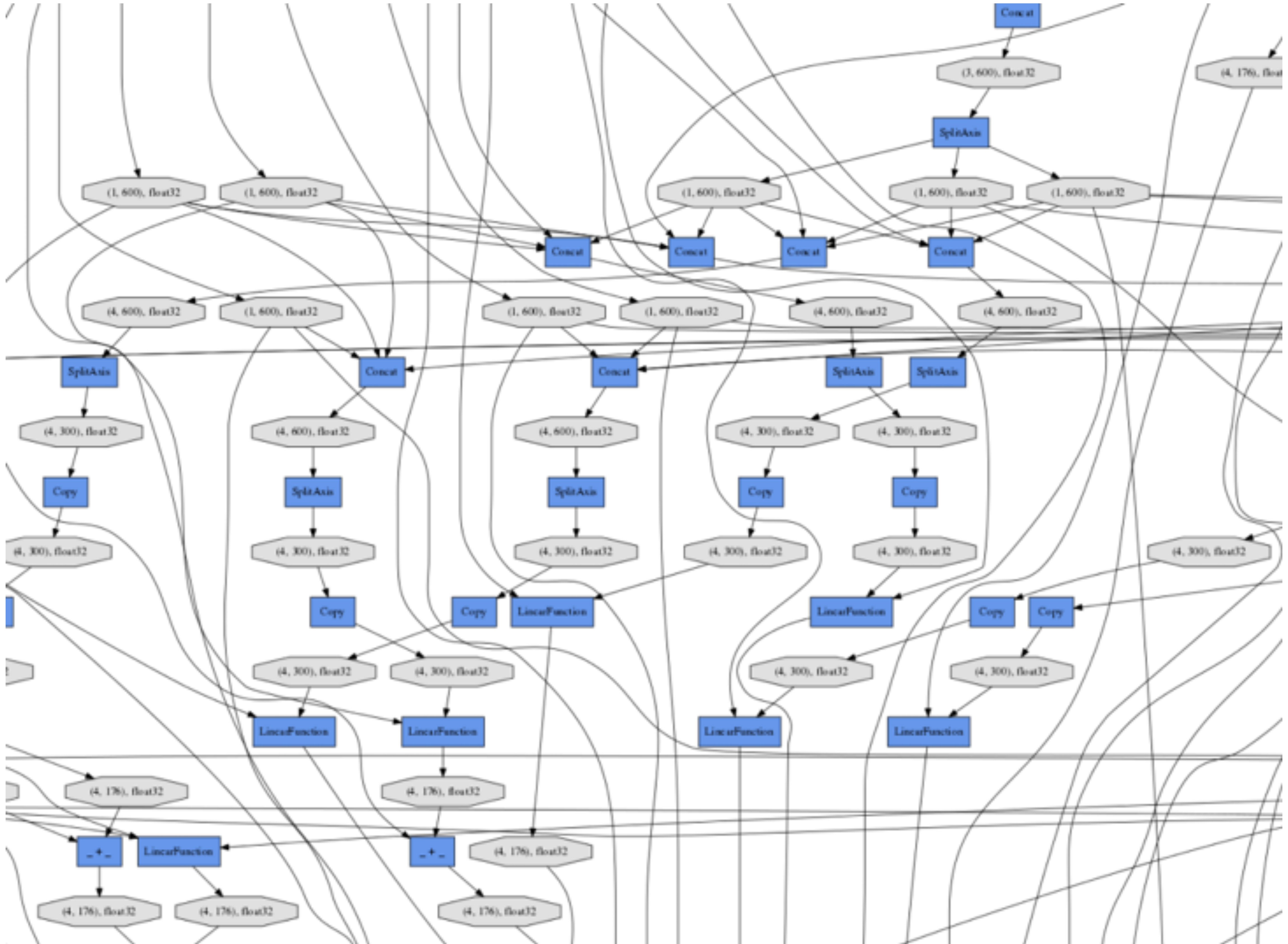
Computation Graphs of NNs

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



Computation Graphs of NNs

- For larger models the computation graph will be:
    - But still, they will be **DAG** (directed acyclic graphs)



Concluding Remarks

- Training neural networks require 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.
  - **ReLU.** has 0/1 gradient… very cheap to store and compute.
- If interested, ask “Automatic Differentiation” to GPT.
  - or this paper: https://arxiv.org/abs/1502.05767

Cheers

- *Next up.* Strategies for Neural Network Training