DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota

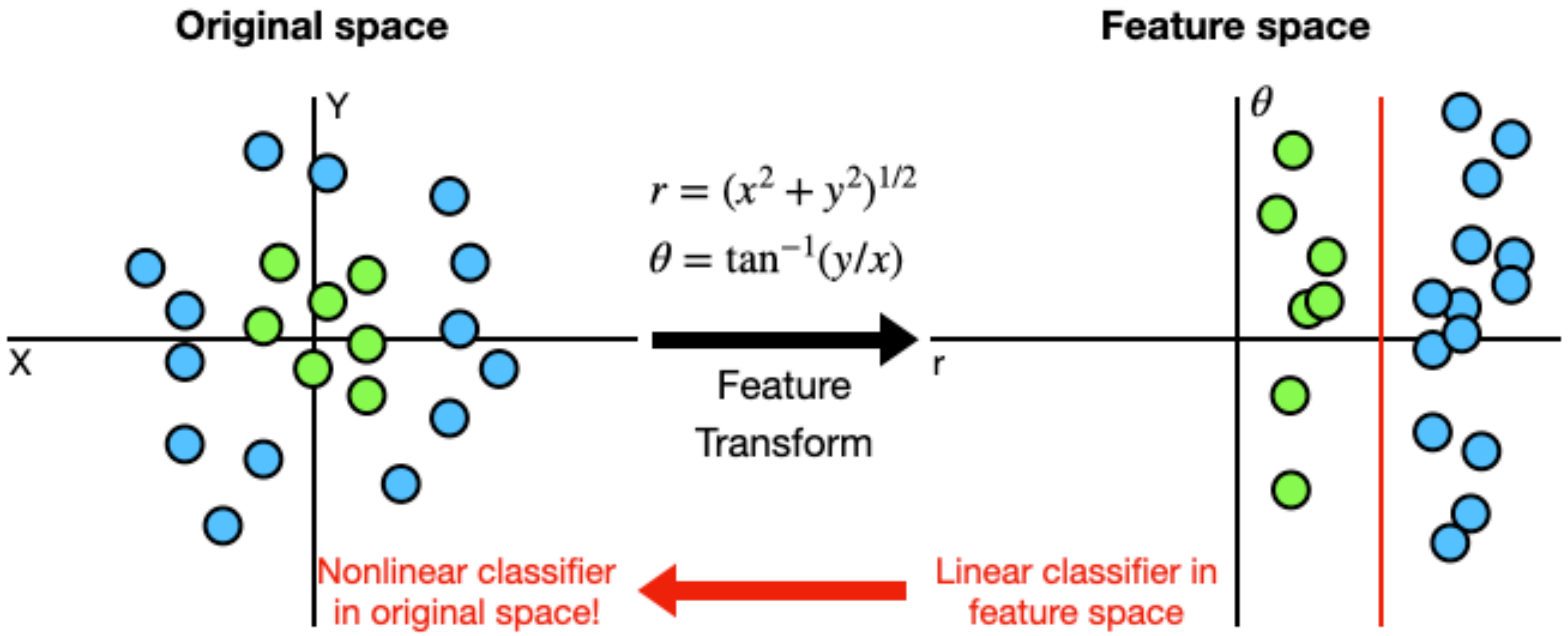
$$\frac{\partial L}{\partial W_{\ell_1}} \leftarrow \frac{\partial L}{\partial W_{\ell_2}} \leftarrow \frac{\partial L}{\partial W_{\ell_3}} \leftarrow \frac{\partial L}{\partial W_{\ell_4}} \leftarrow \frac{\partial L}{\partial W_{\ell_5}} \leftarrow \frac{\partial L}{\partial \text{Out}}$$

Project 1—Reminder

- Instructions and code available on the website
  - Here: deeprob.org/projects/project1/
- Uses Python, PyTorch and Google Colab
- Implement KNN, linear SVM, and linear softmax classifiers
- **Autograder is online and updated**
- **Due Thursday, January 26th 11:59 PM EST**

Recap from Previous Lecture

Feature transform + Linear classifier allows nonlinear decision boundaries

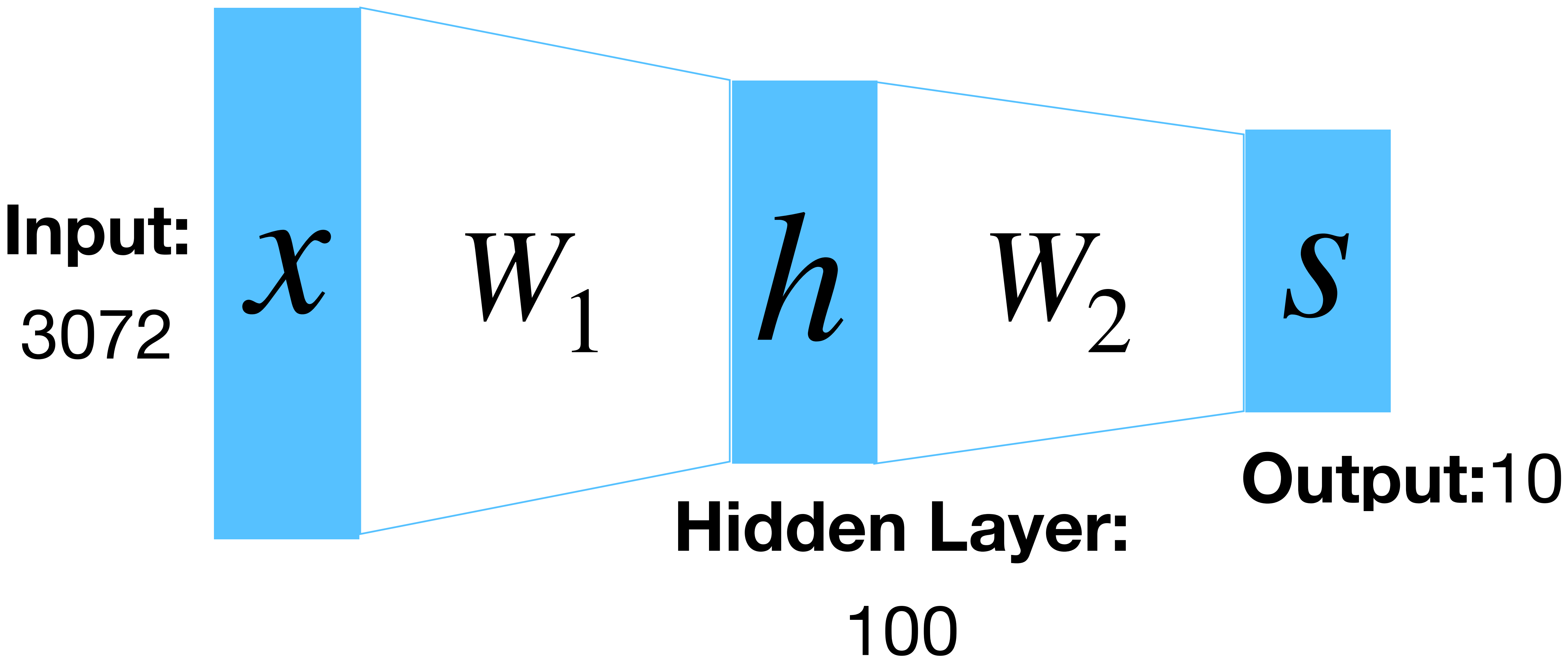


Neural Networks as learnable feature transforms

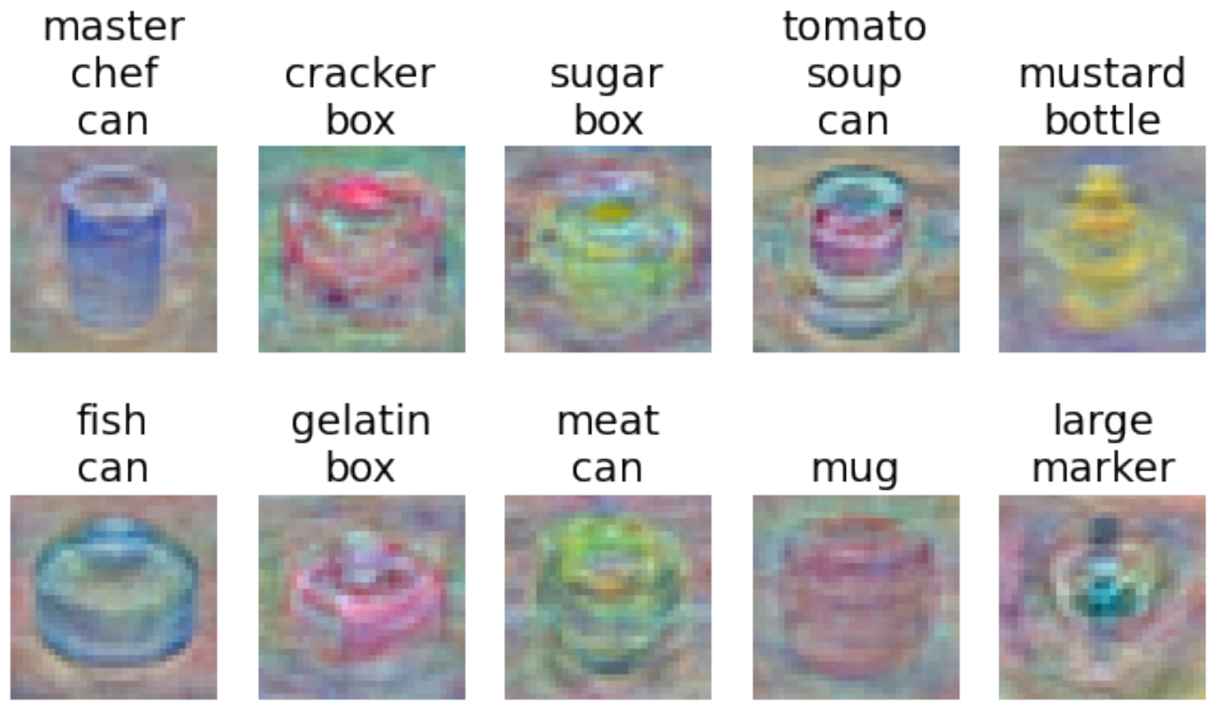
Recap from Previous Lecture

From linear classifiers to
fully-connected networks

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



Linear classifier: One template per class

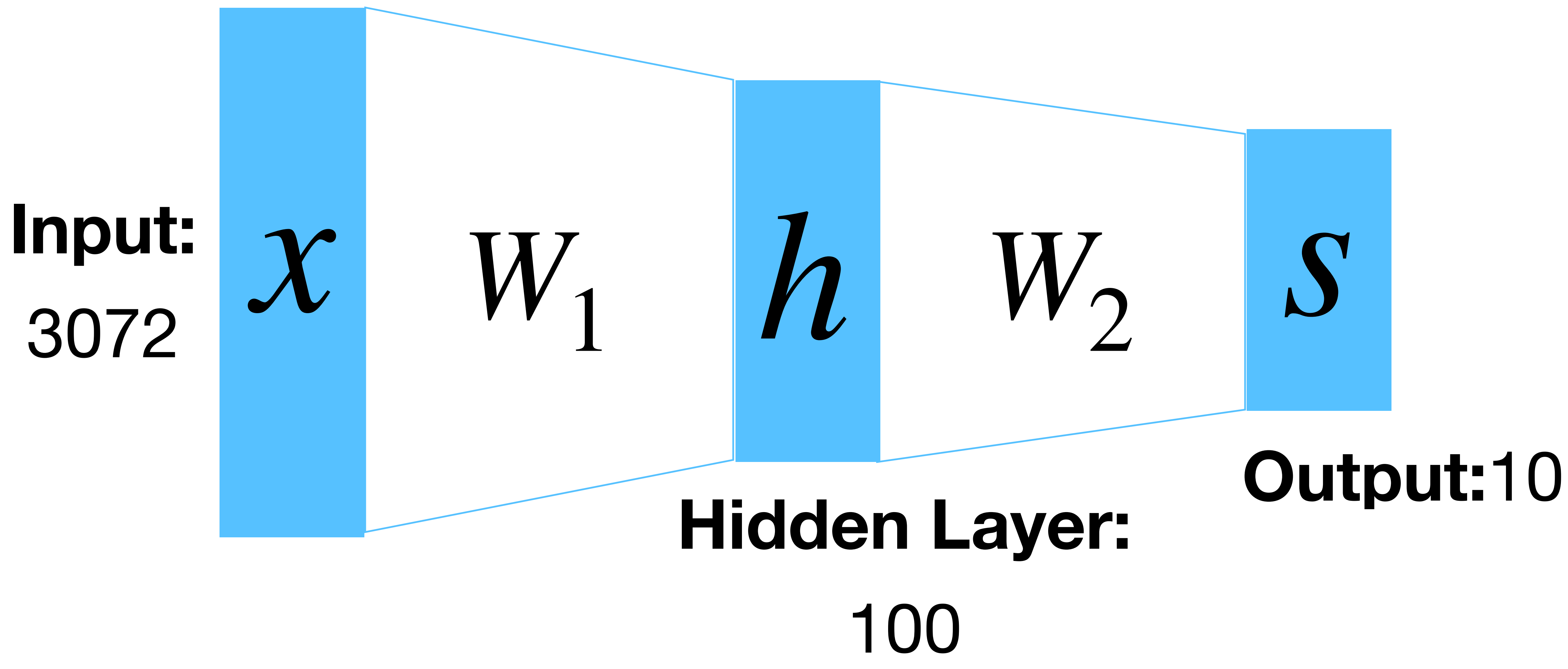


Neural networks: Many reusable templates

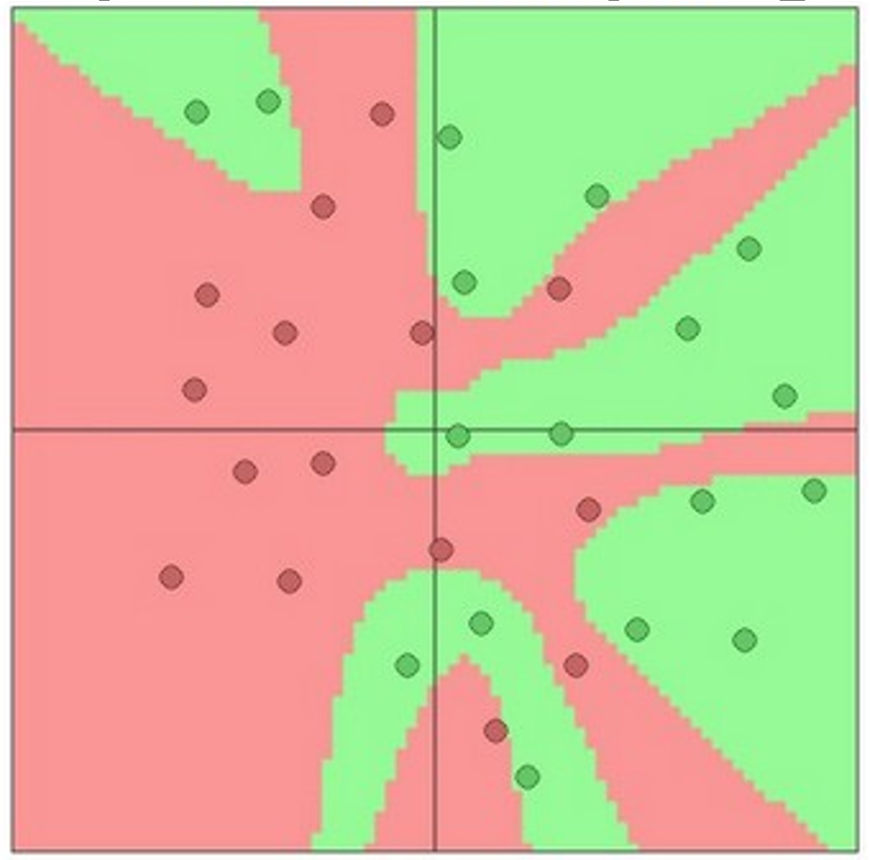
Recap from Previous Lecture

From linear classifiers to fully-connected networks

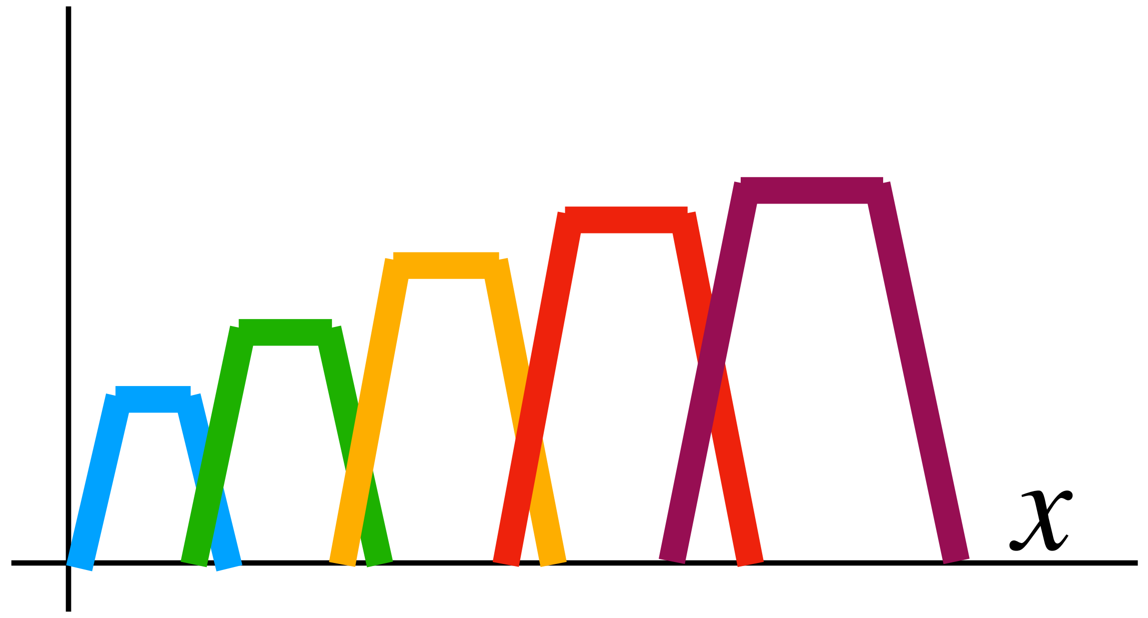
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



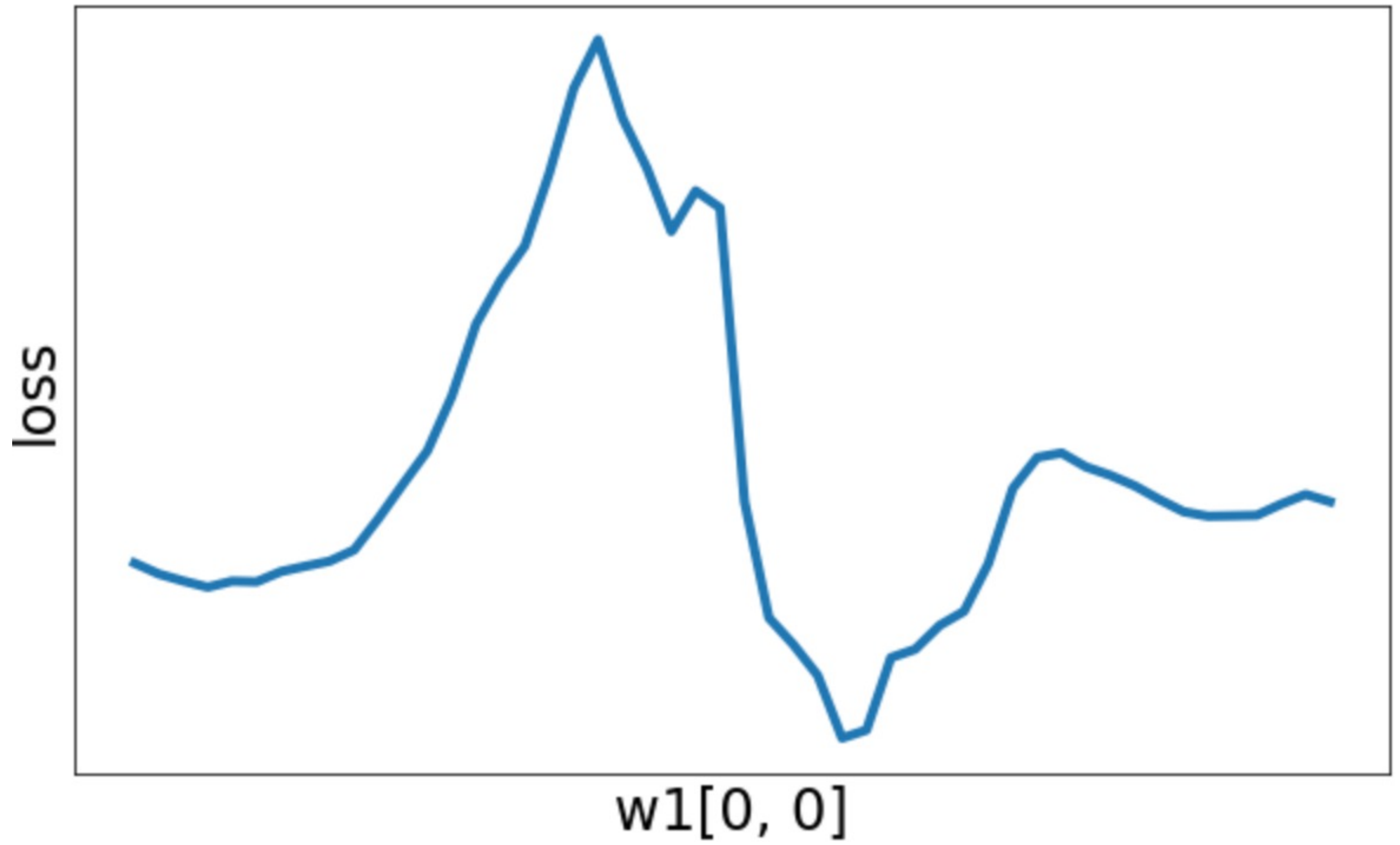
Space Warping



Universal approximation



Nonconvex



Problem: How to compute gradients?

$s = W_2 \max(0, W_1 x + b_!) + b_2$ Nonlinear score function

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$ Per-element data loss

$R(W) = \sum_k W_k^2$ L2 regularization

$L(W_1, W_2, b_1, b_2) = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$ Total loss

If we can compute $\frac{\delta L}{\delta W_1}, \frac{\delta L}{\delta W_2}, \frac{\delta L}{\delta b_1}, \frac{\delta L}{\delta b_2}$ then we can optimize with SGD

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x - W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x - W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

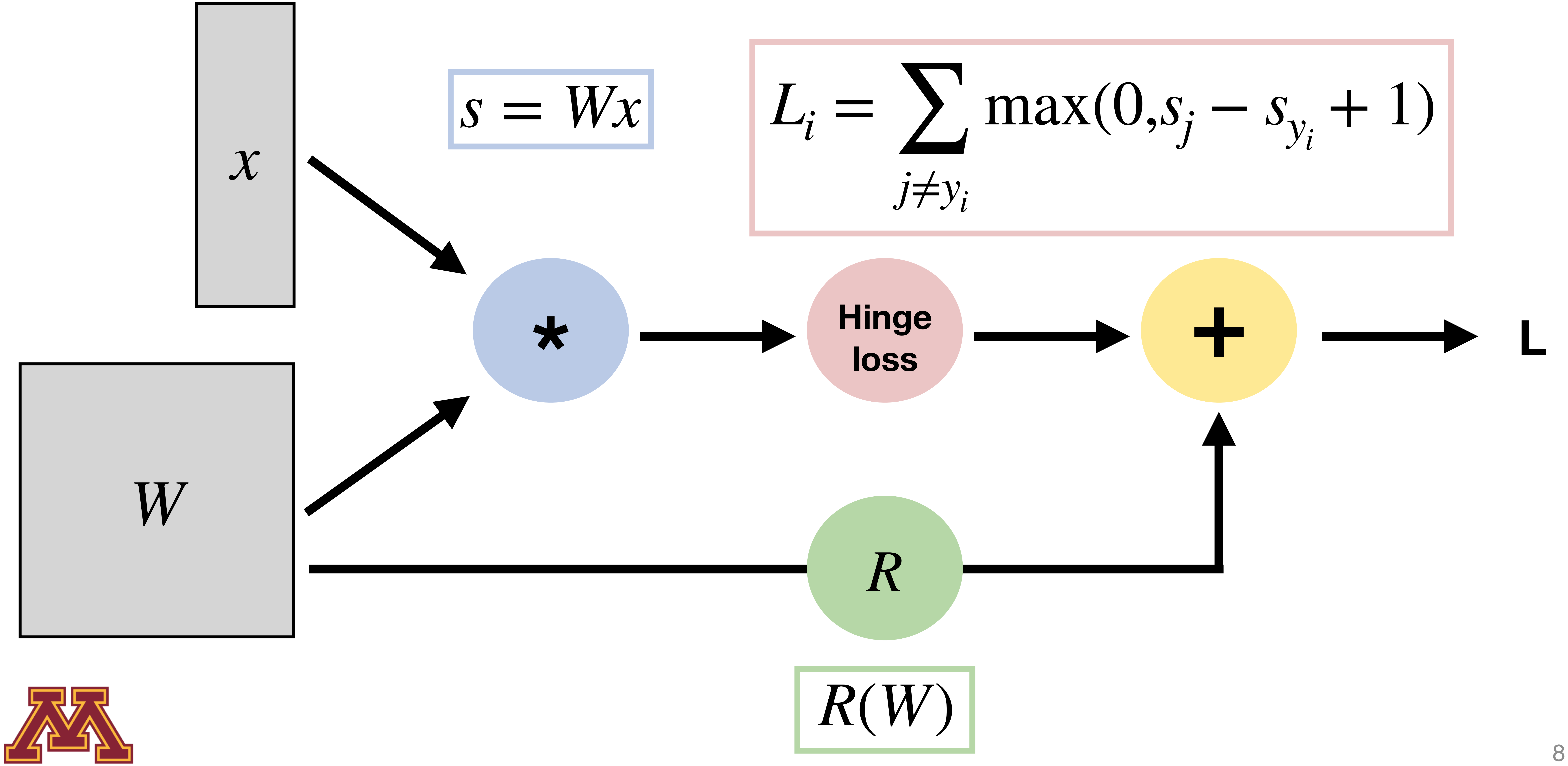
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x - W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

Problem: Very tedious with lots of matrix calculus

Problem: What if we want to change the loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

Problem: Not feasible for very complex models!

Better Idea: Computational Graphs

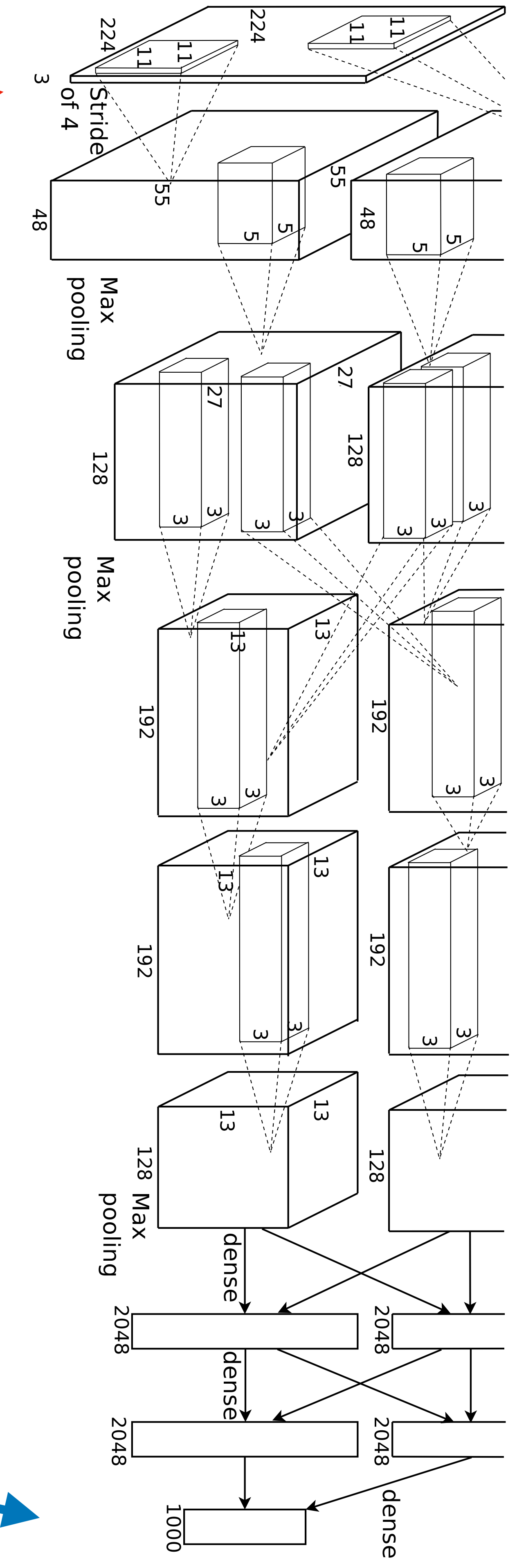


Deep Network (AlexNet)

Input image

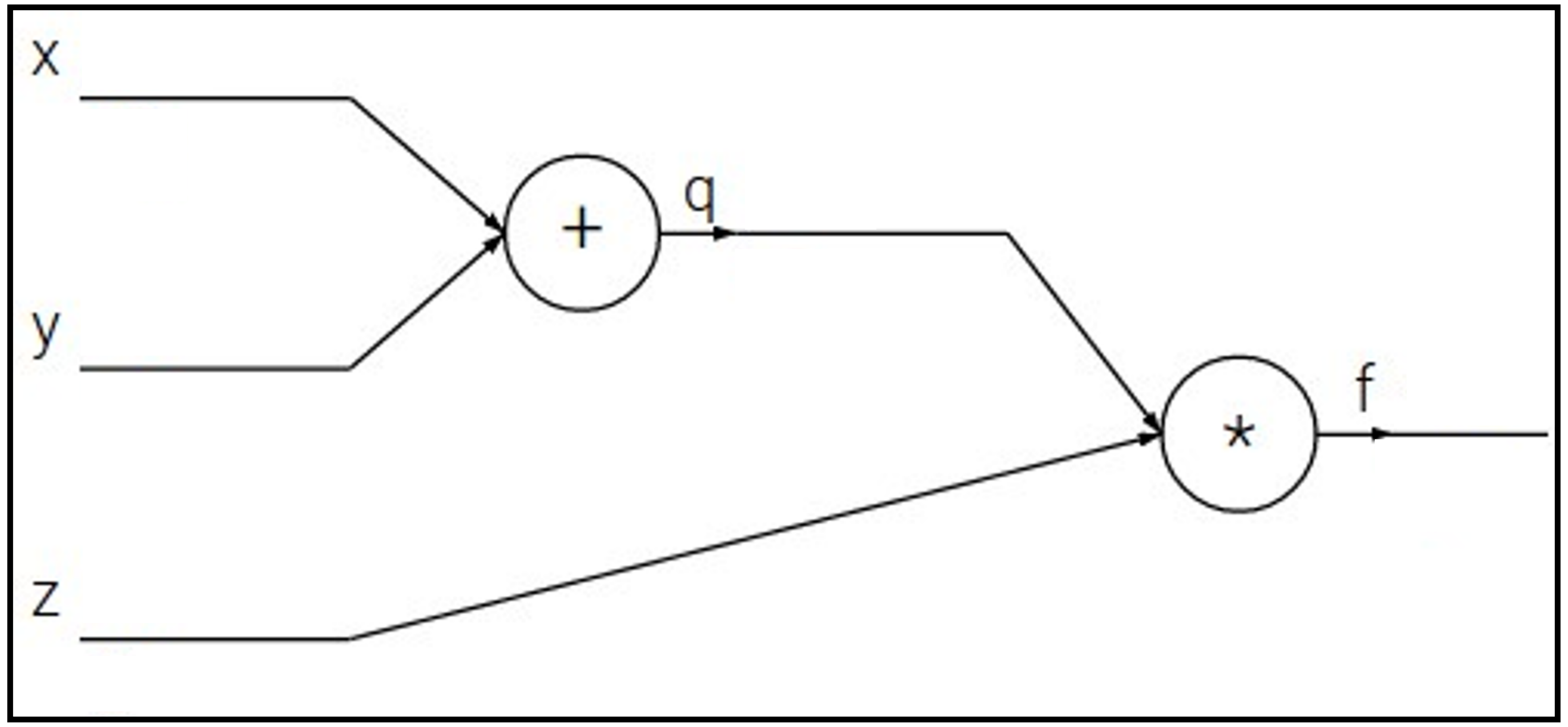
Weights

Loss



Backpropagation: Simple Example

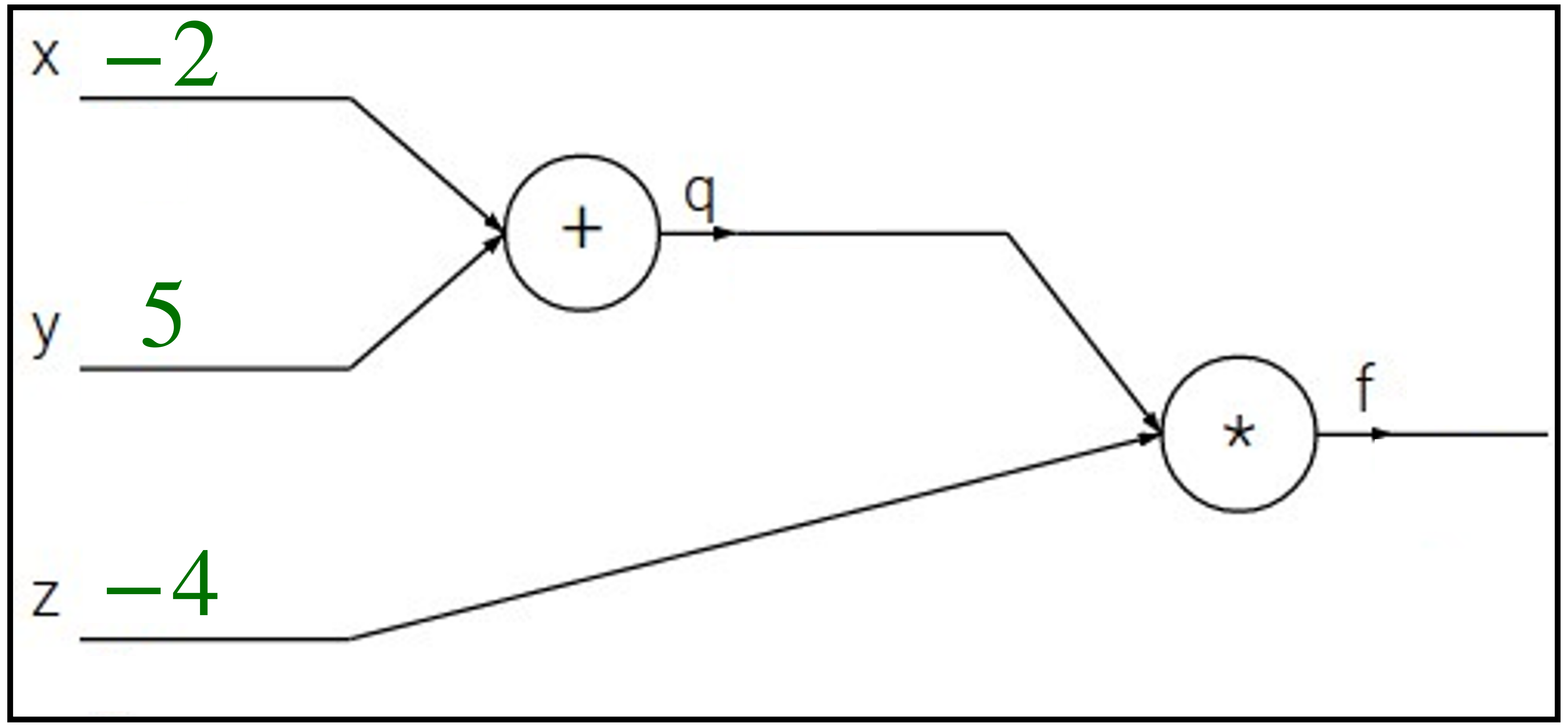
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

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

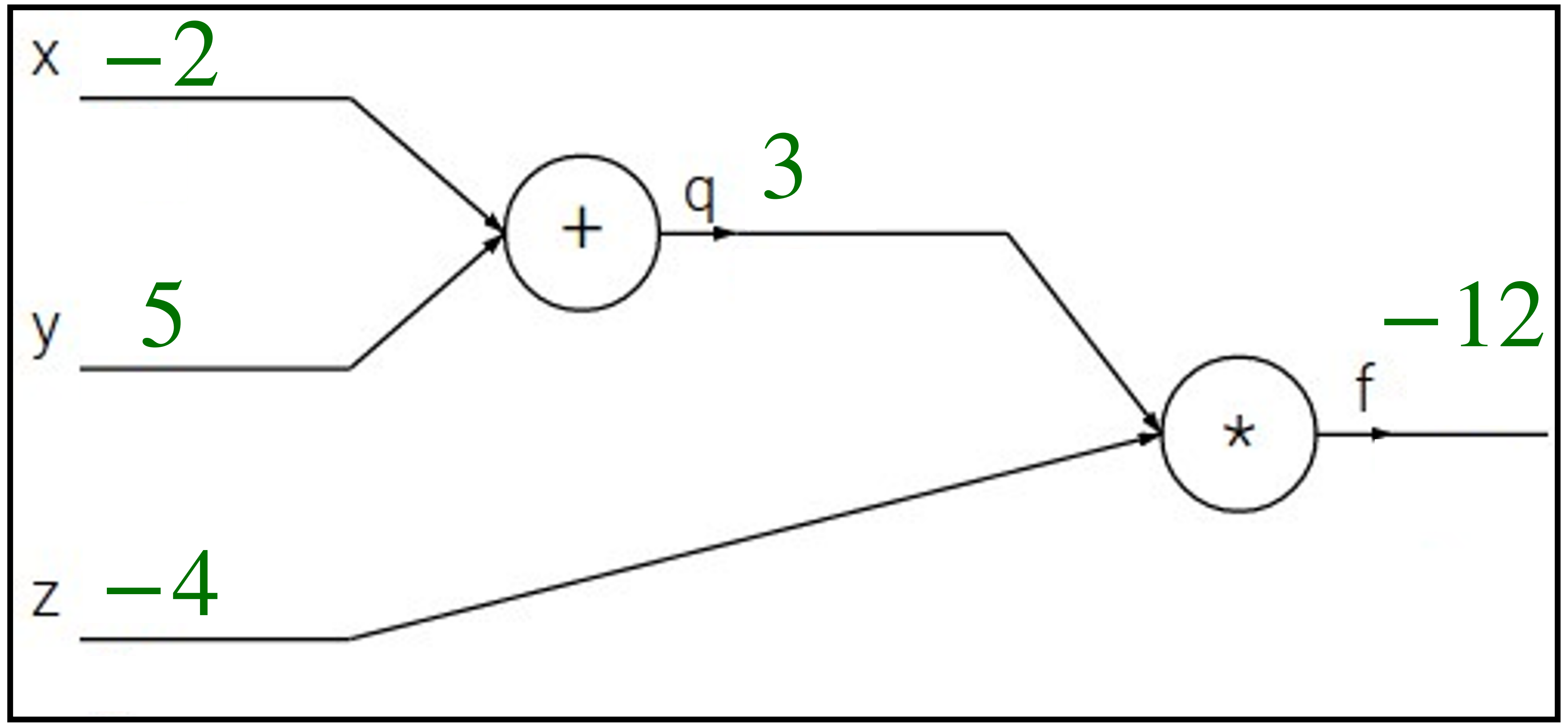
e.g. $x = -2, y = 5, z = -4$



Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



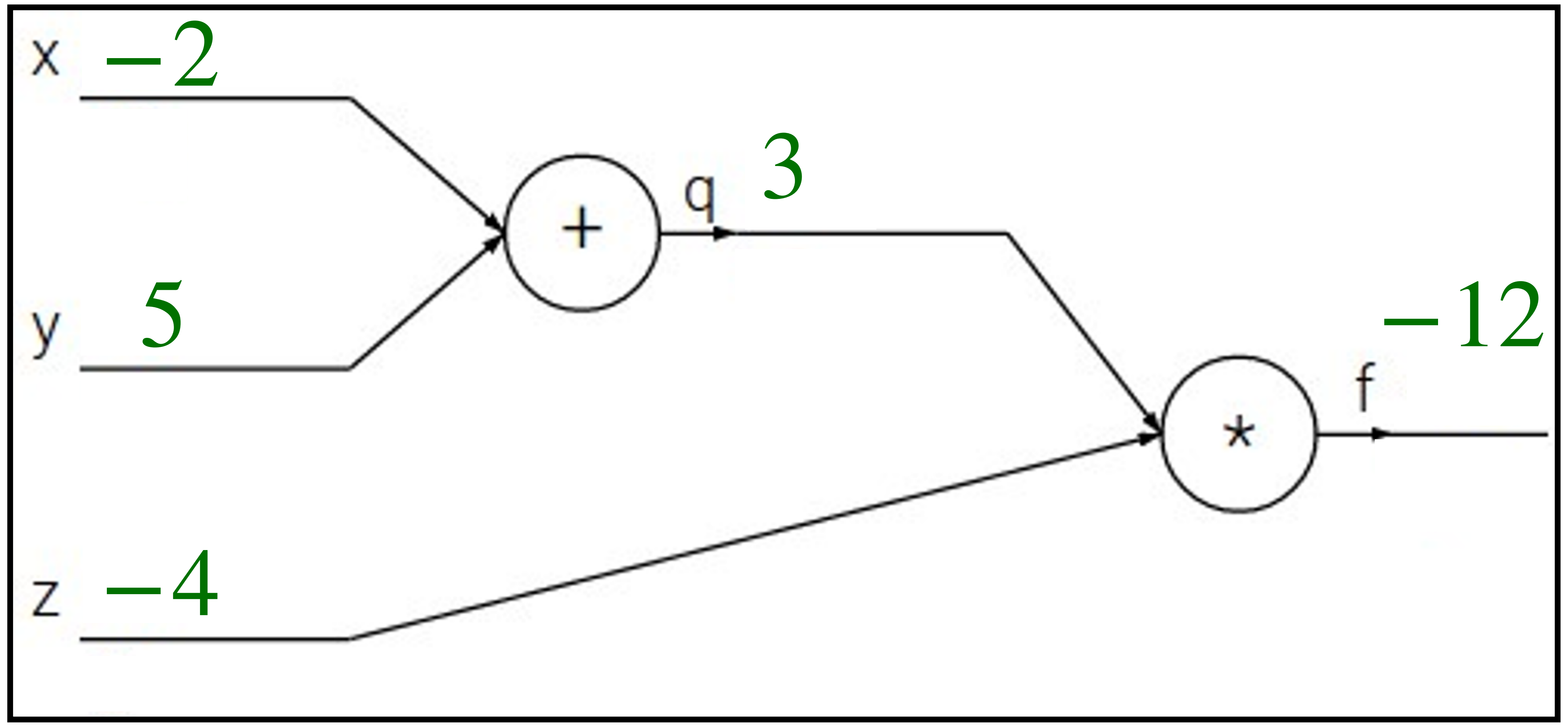
1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = q \cdot z$$

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = q \cdot z$$

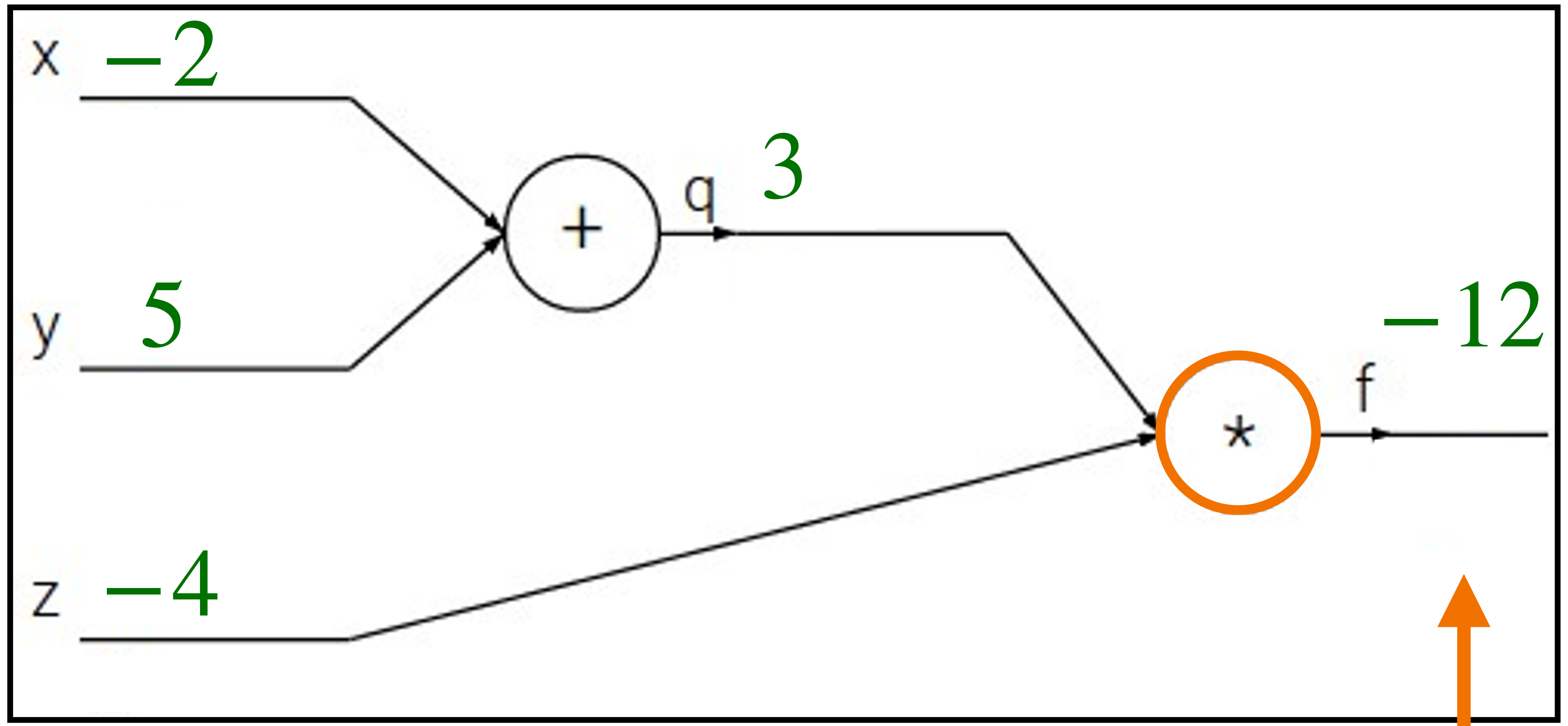
2. **Backward pass**: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial f}}$$

Backpropagation: Simple Example

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

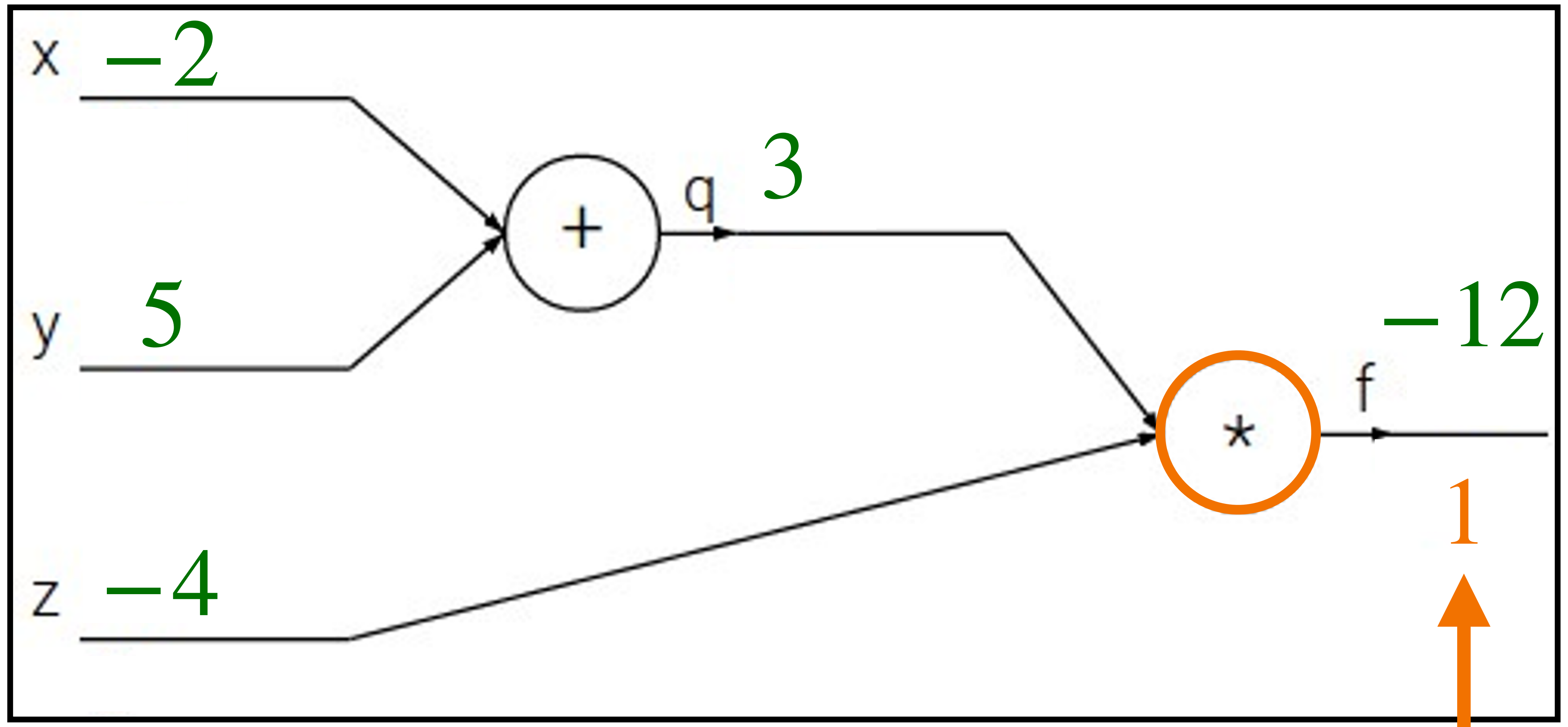
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial f}}$$

Backpropagation: Simple Example

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

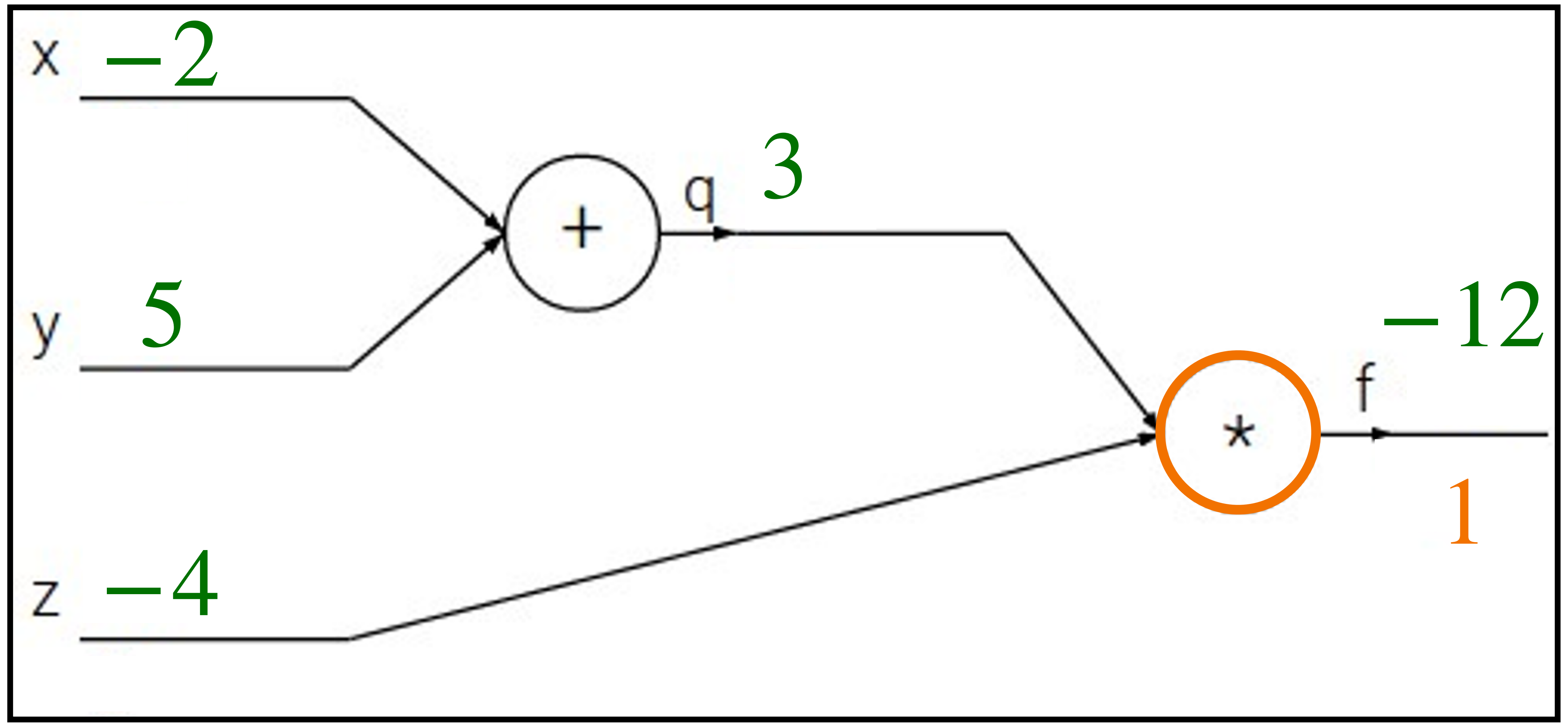
e.g. $x = -2$, $y = 5$, $z = -4$

1. **Forward pass**: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. **Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

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

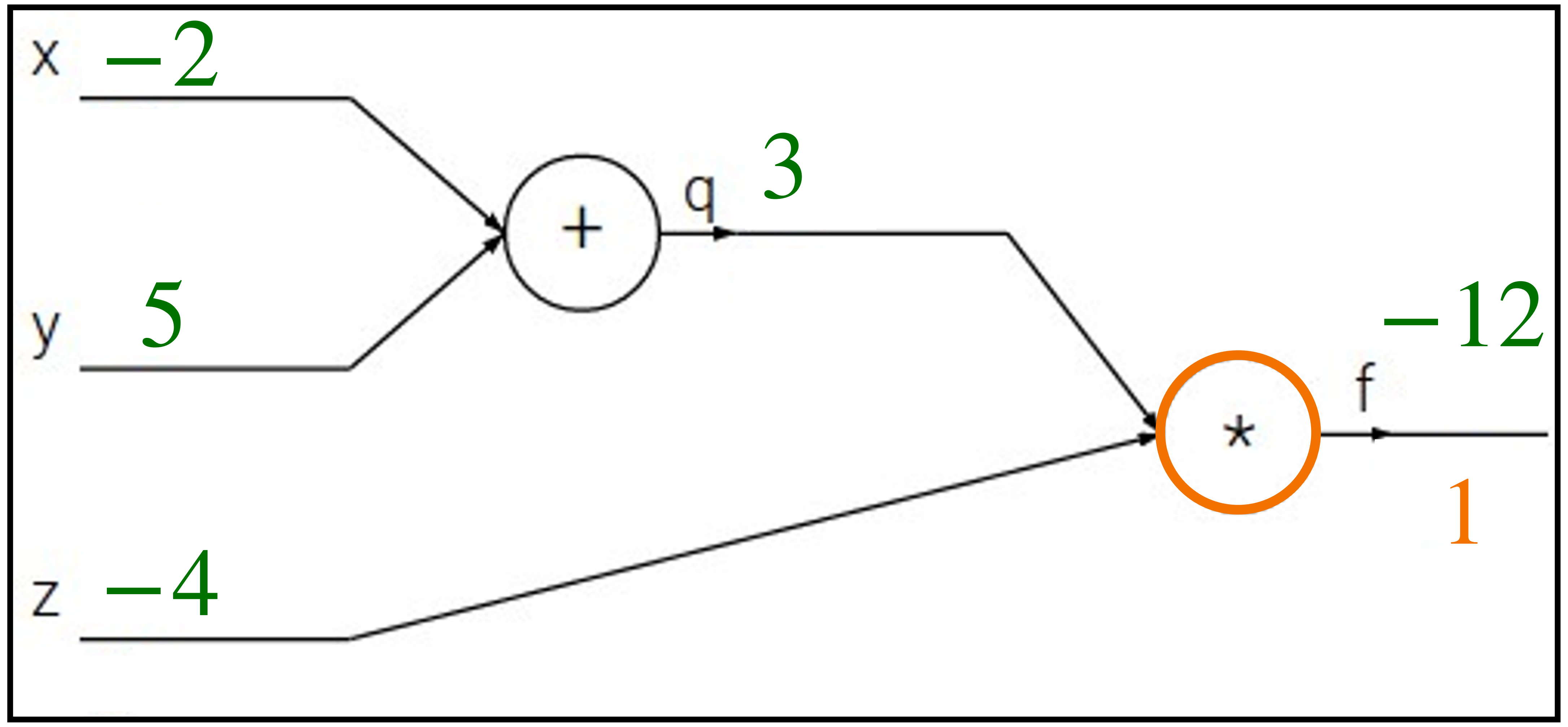
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$q = x + y$ $\quad \boxed{f = q \cdot z}$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$\boxed{\dfrac{\partial f}{\partial z}}$

Backpropagation: Simple Example

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

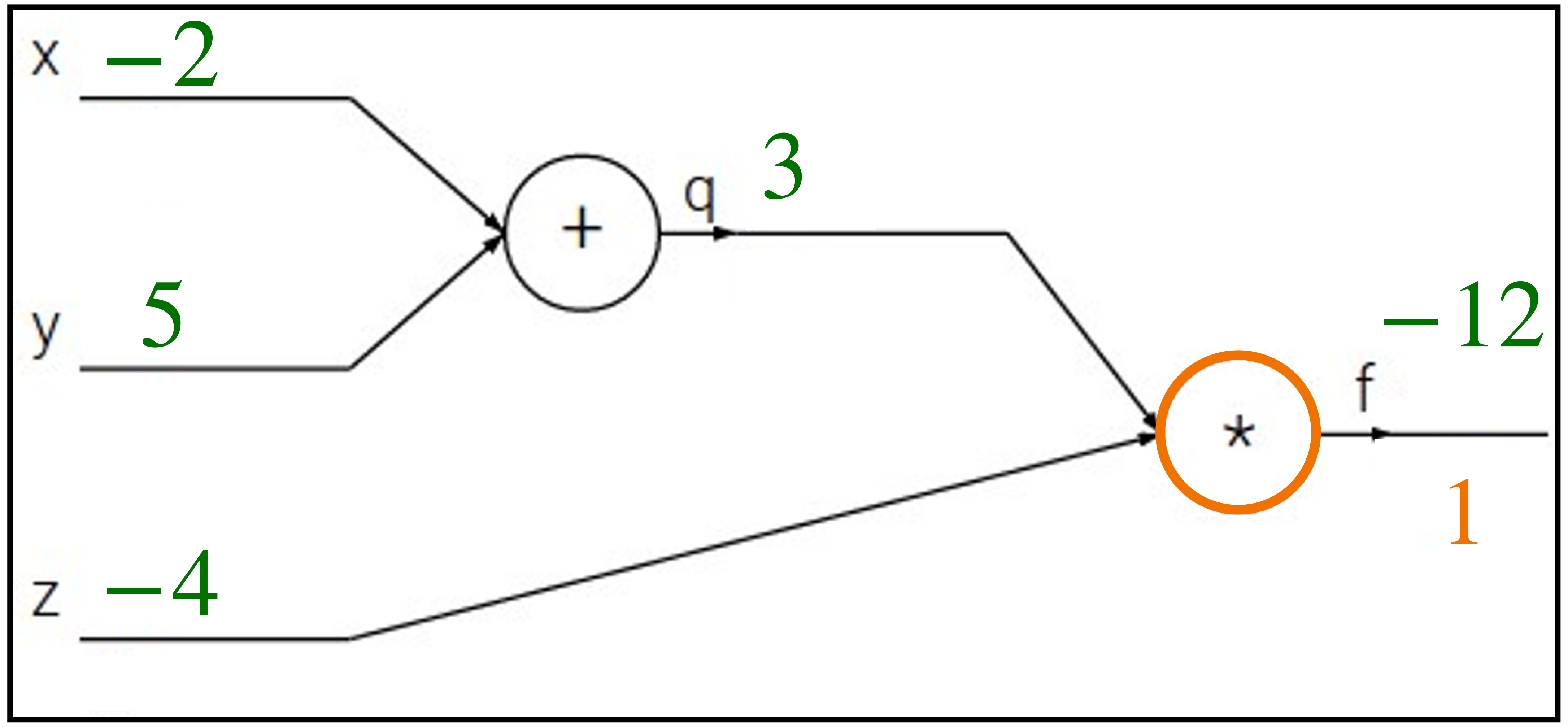
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

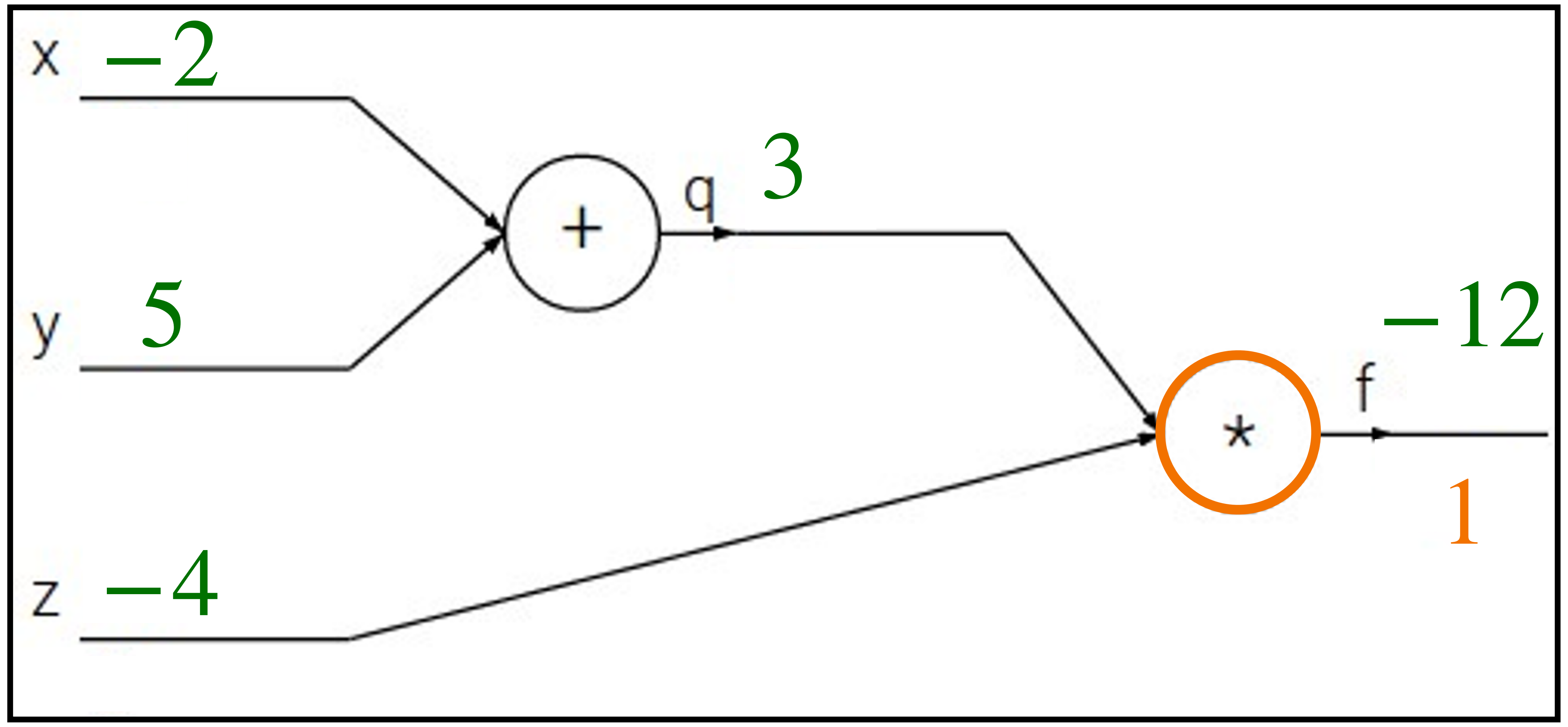


$$\boxed{\frac{\partial f}{\partial z}}$$

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial z} = q}$$

Backpropagation: Simple Example

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

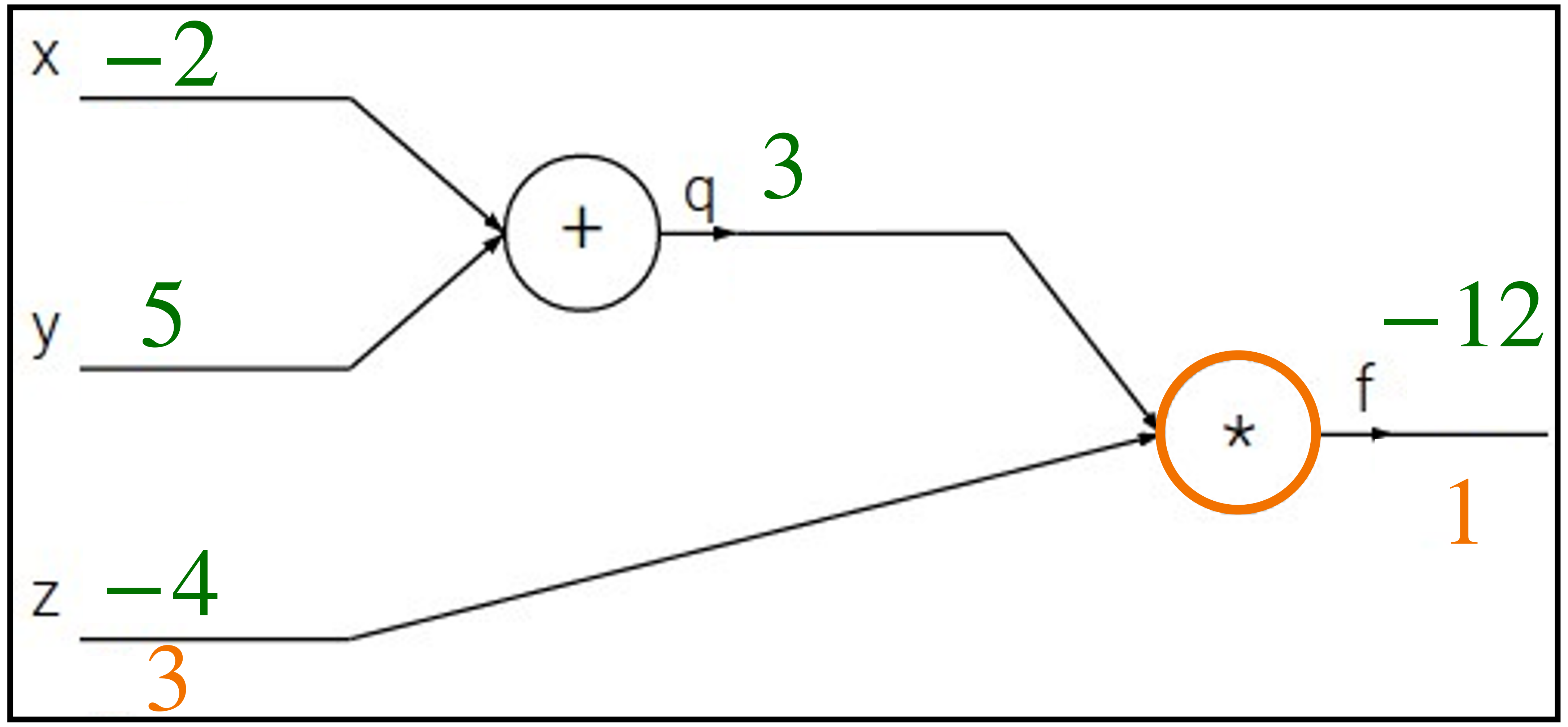
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial z} = q}$$

Backpropagation: Simple Example

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

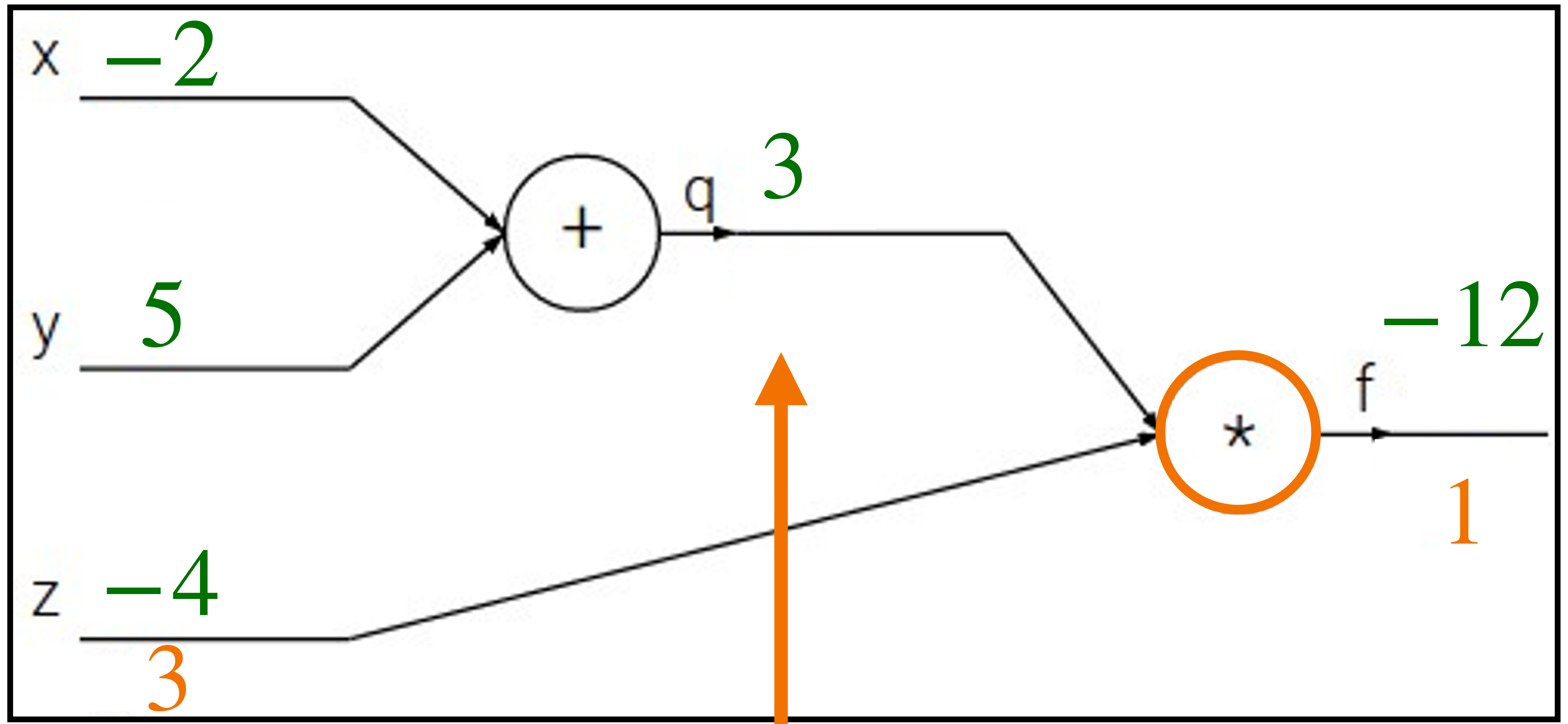
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$q = x + y \qquad \boxed{f = q \cdot z}$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$\boxed{\dfrac{\partial f}{\partial q}}$

Backpropagation: Simple Example

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

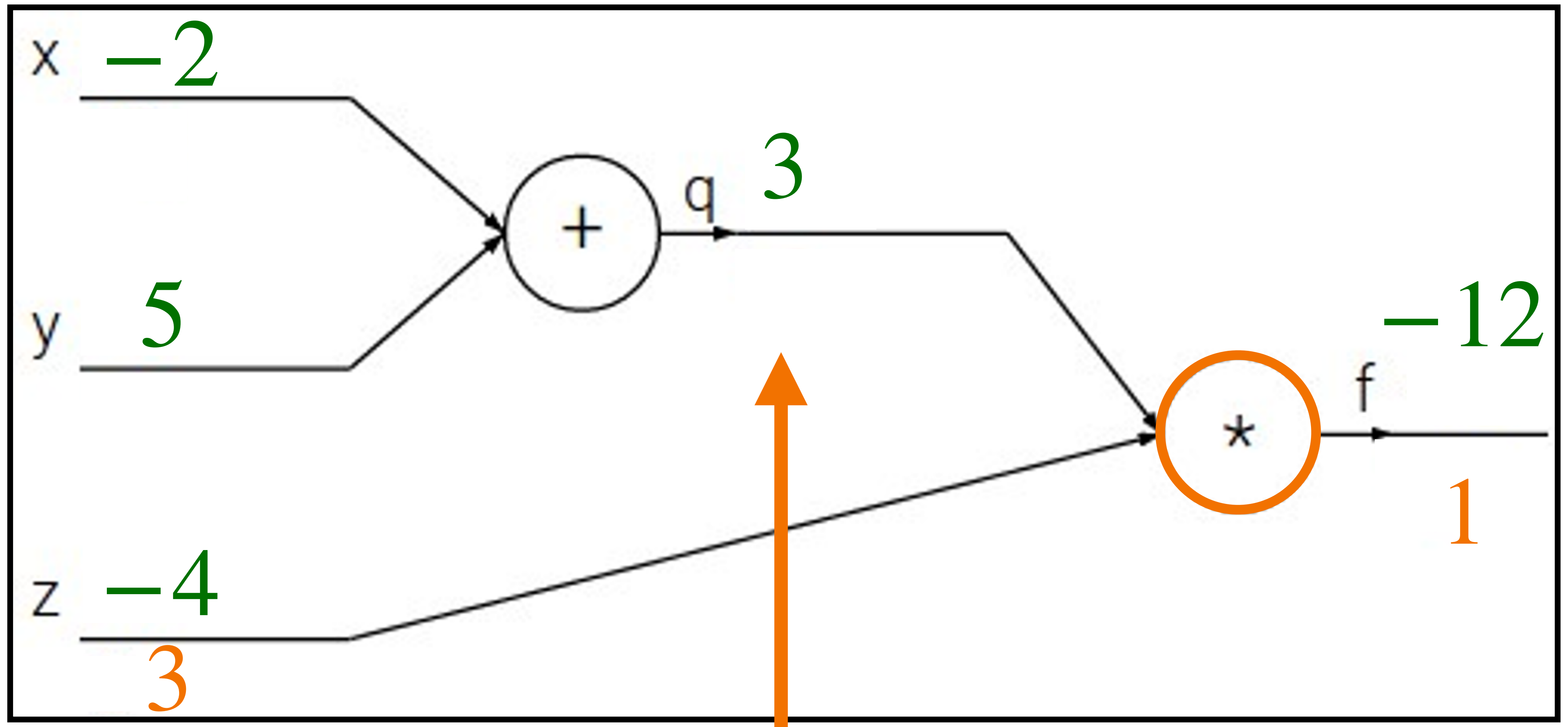
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$q = x + y$ $\quad \boxed{f = q \cdot z}$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q}}$$

Backpropagation: Simple Example

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

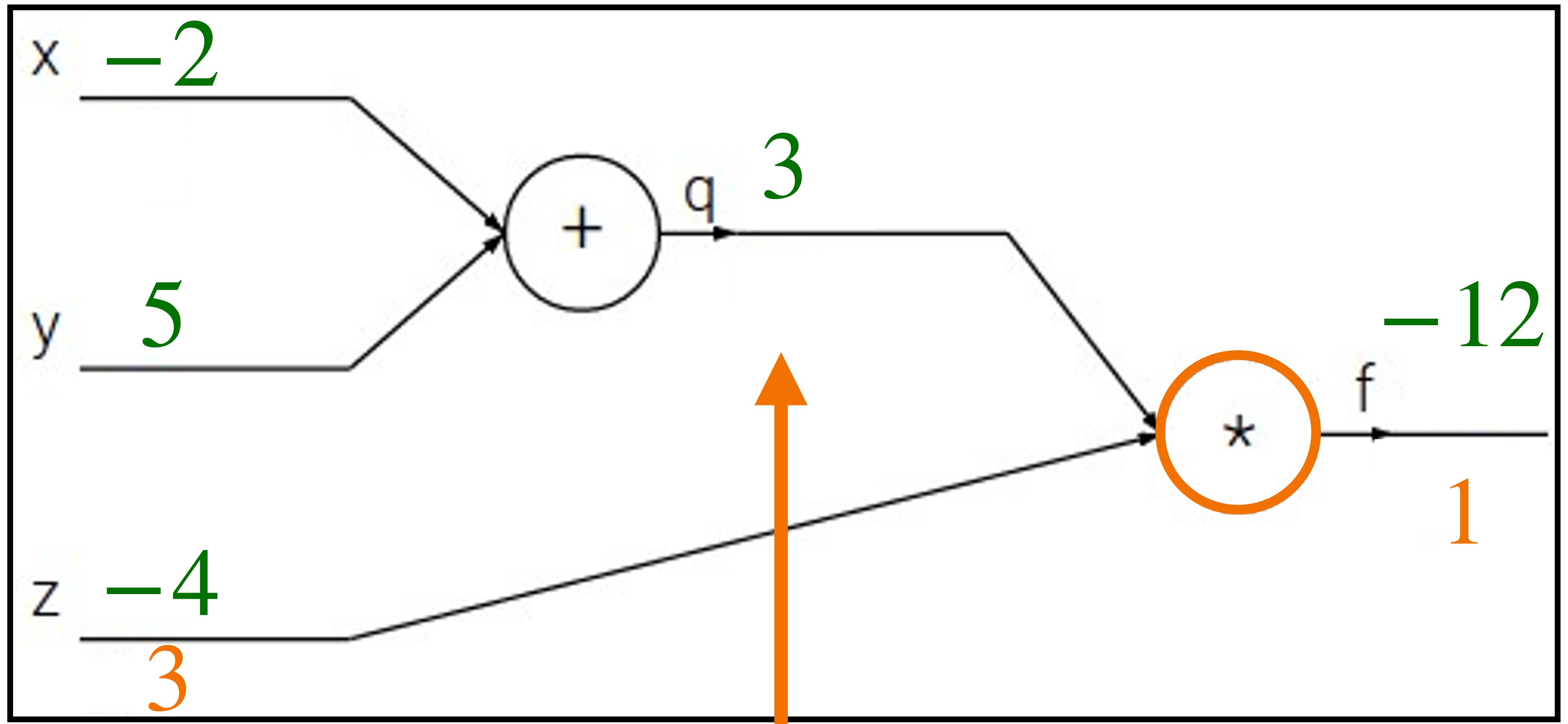
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$q = x + y$ $\quad$ $\boxed{f = q \cdot z}$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q} = z}$$

Backpropagation: Simple Example

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

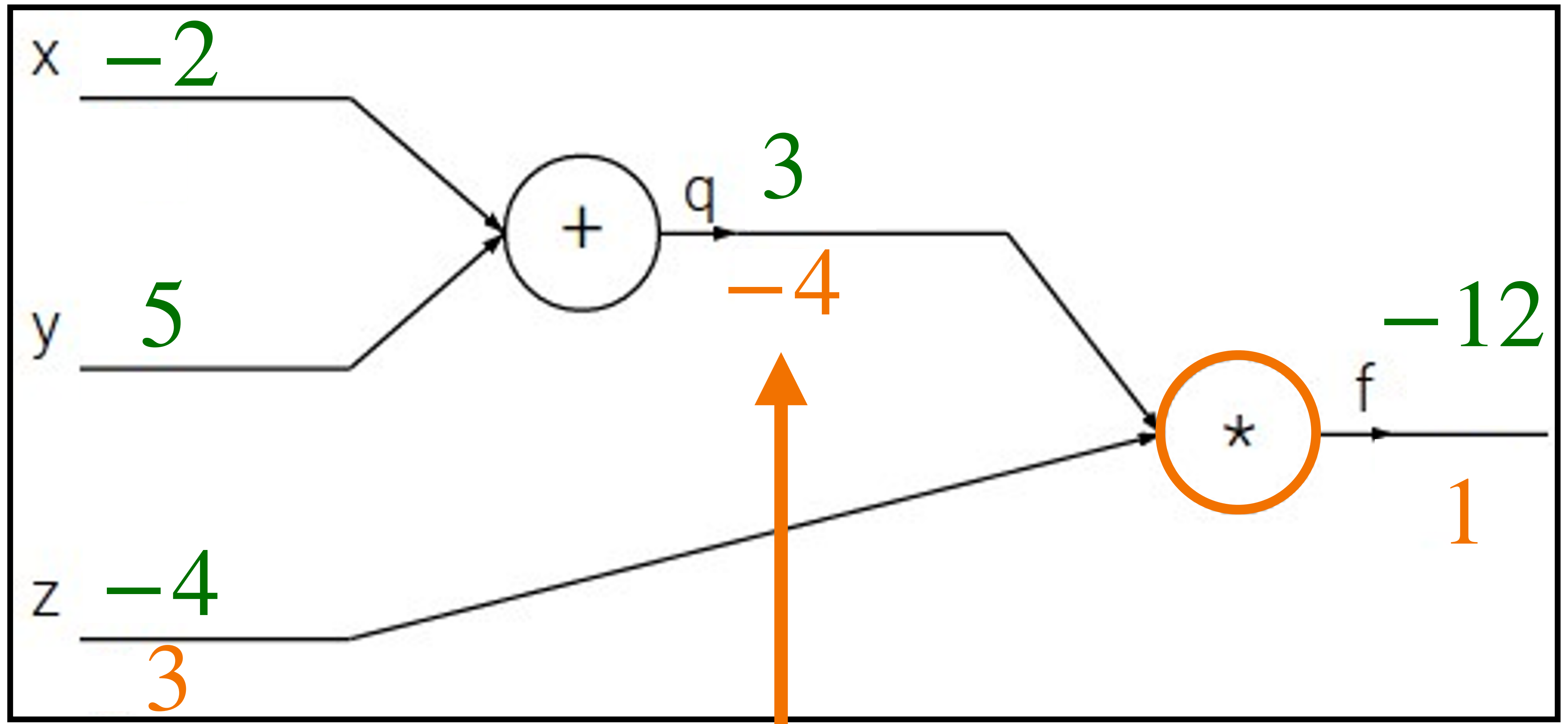
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \qquad \boxed{f = q \cdot z}$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q} = z}$$

Backpropagation: Simple Example

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

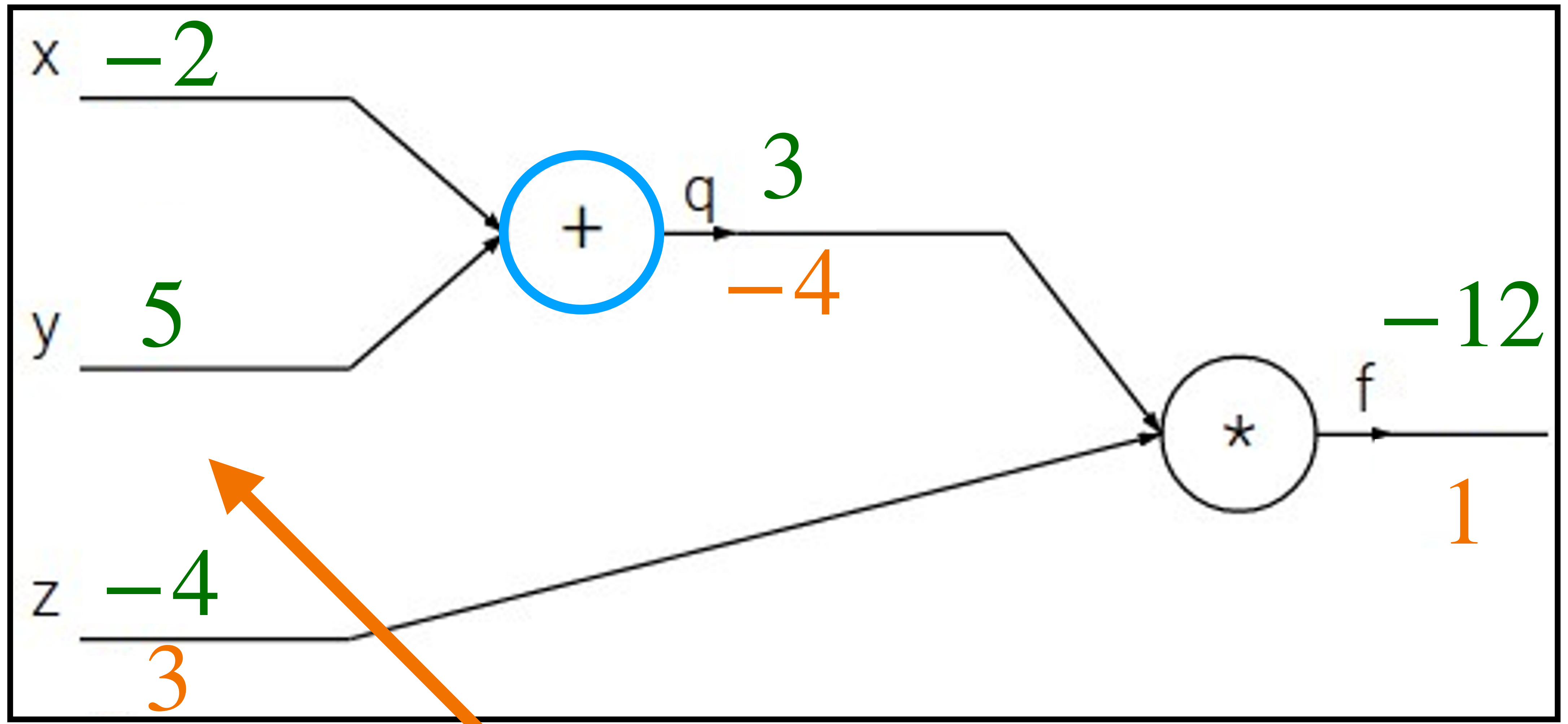
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$\boxed{q = x + y}$ $\quad f = q \cdot z$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\boxed{\frac{\partial f}{\partial y}}$

Backpropagation: Simple Example

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

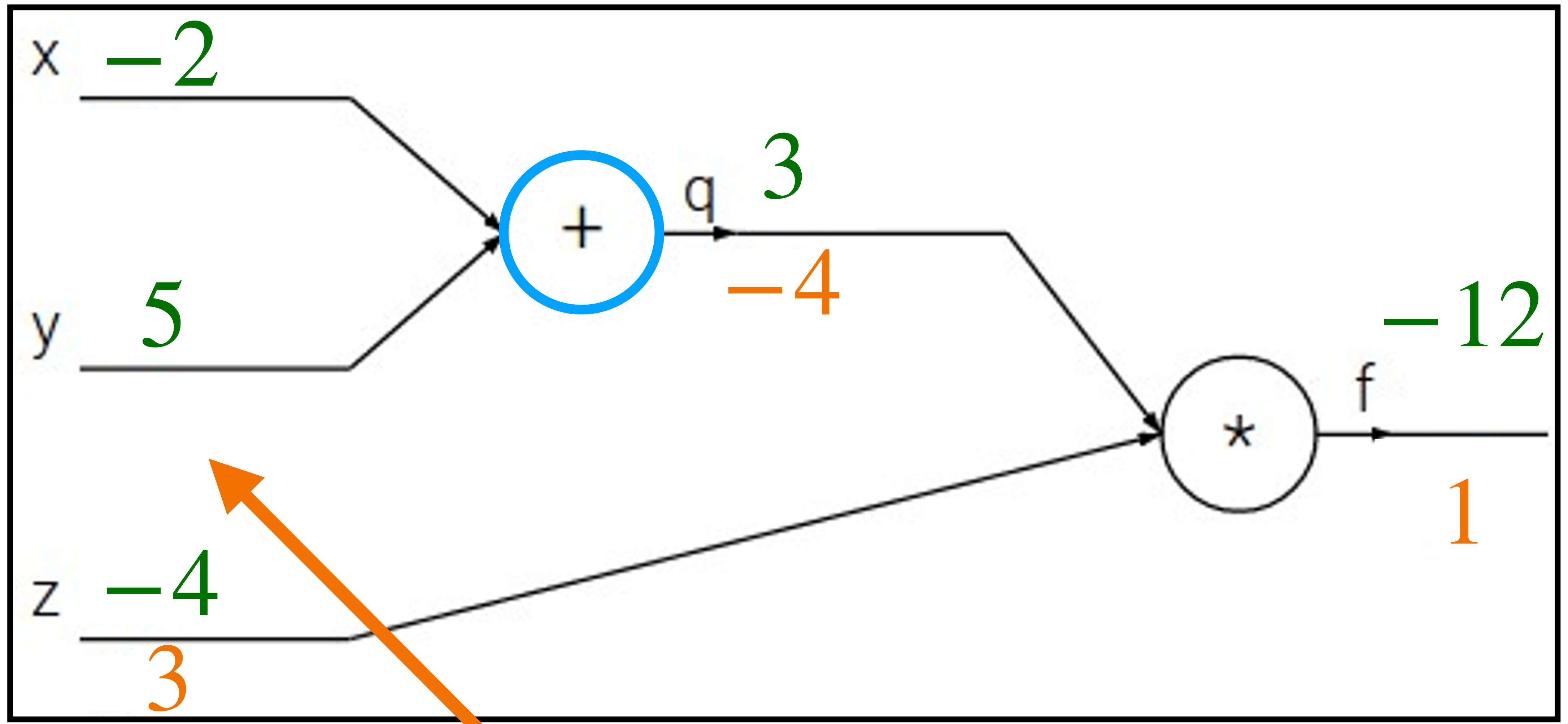
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

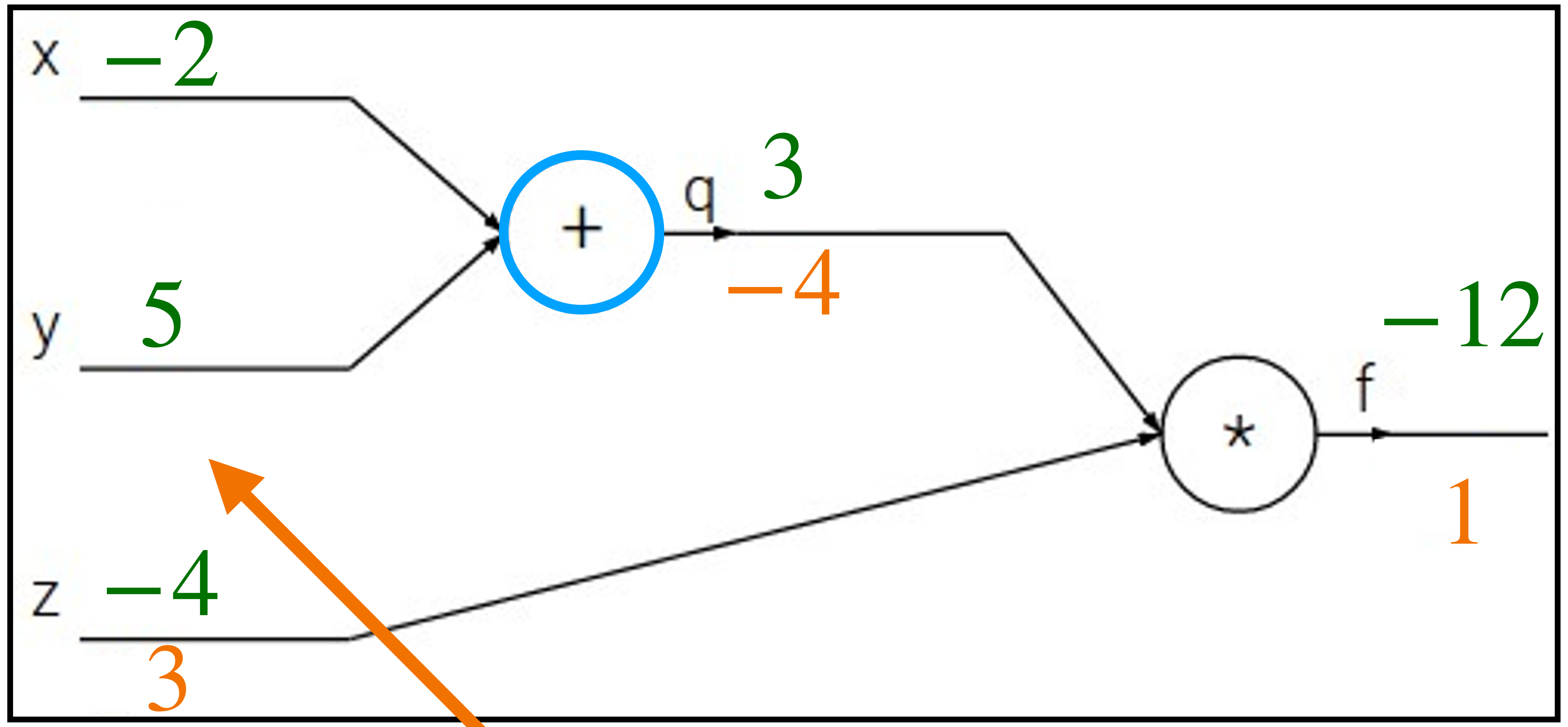


$$\frac{\partial f}{\partial y}$$

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. **Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. **Backward pass**: Compute derivatives

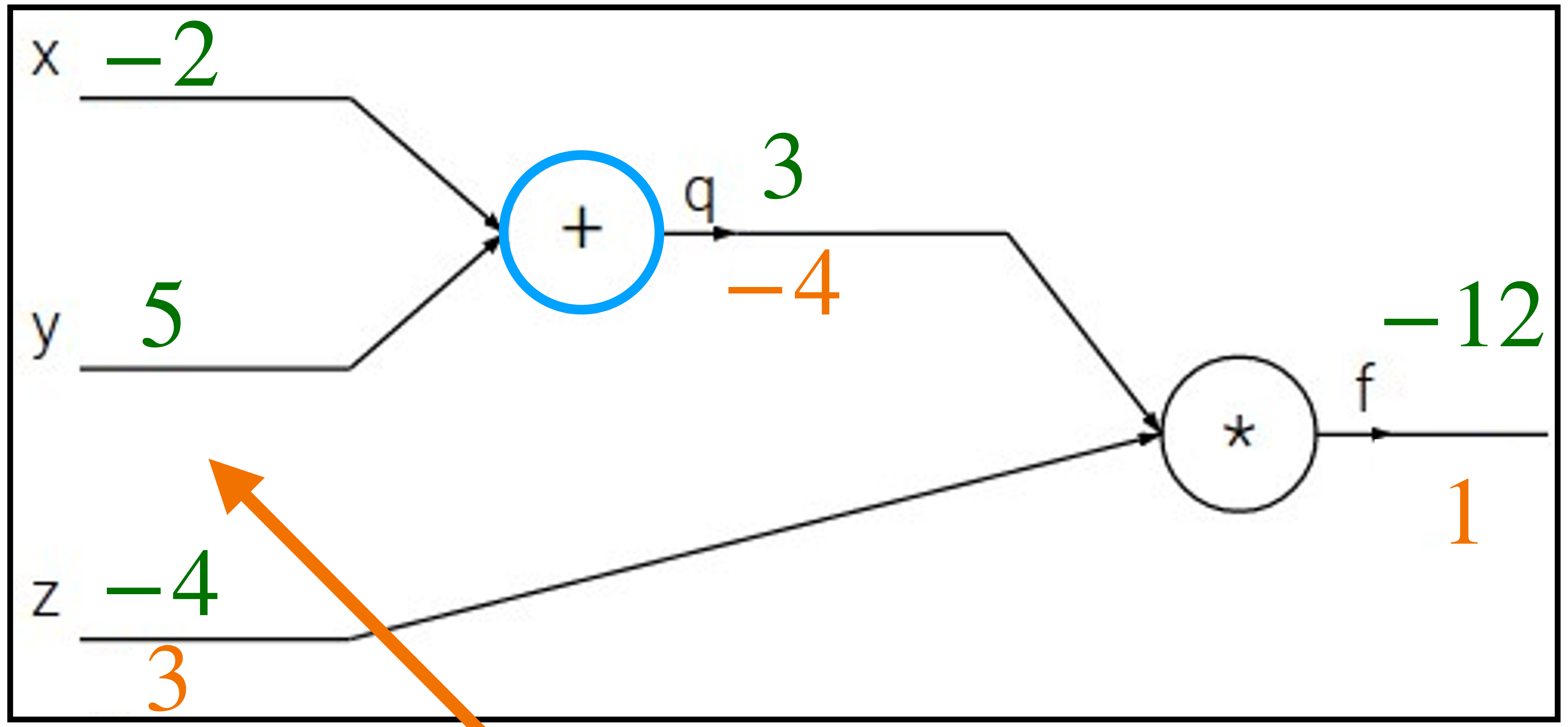
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}}$$

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. Forward pass: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}}$$

Downstream Gradient

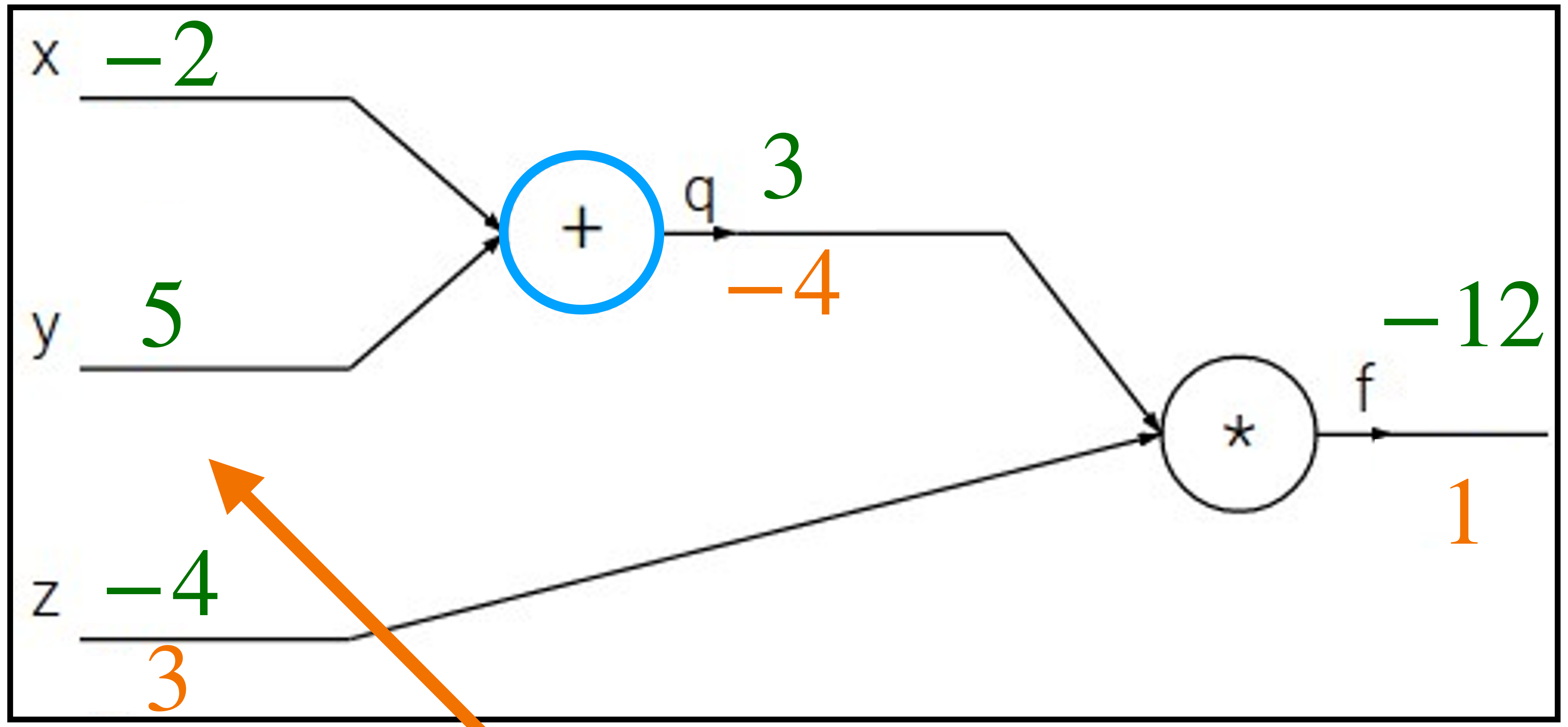
Local Gradient

Upstream Gradient

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. Forward pass: Compute outputs

$\boxed{q = x + y}$ $\quad f = q \cdot z$

2. Backward pass: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

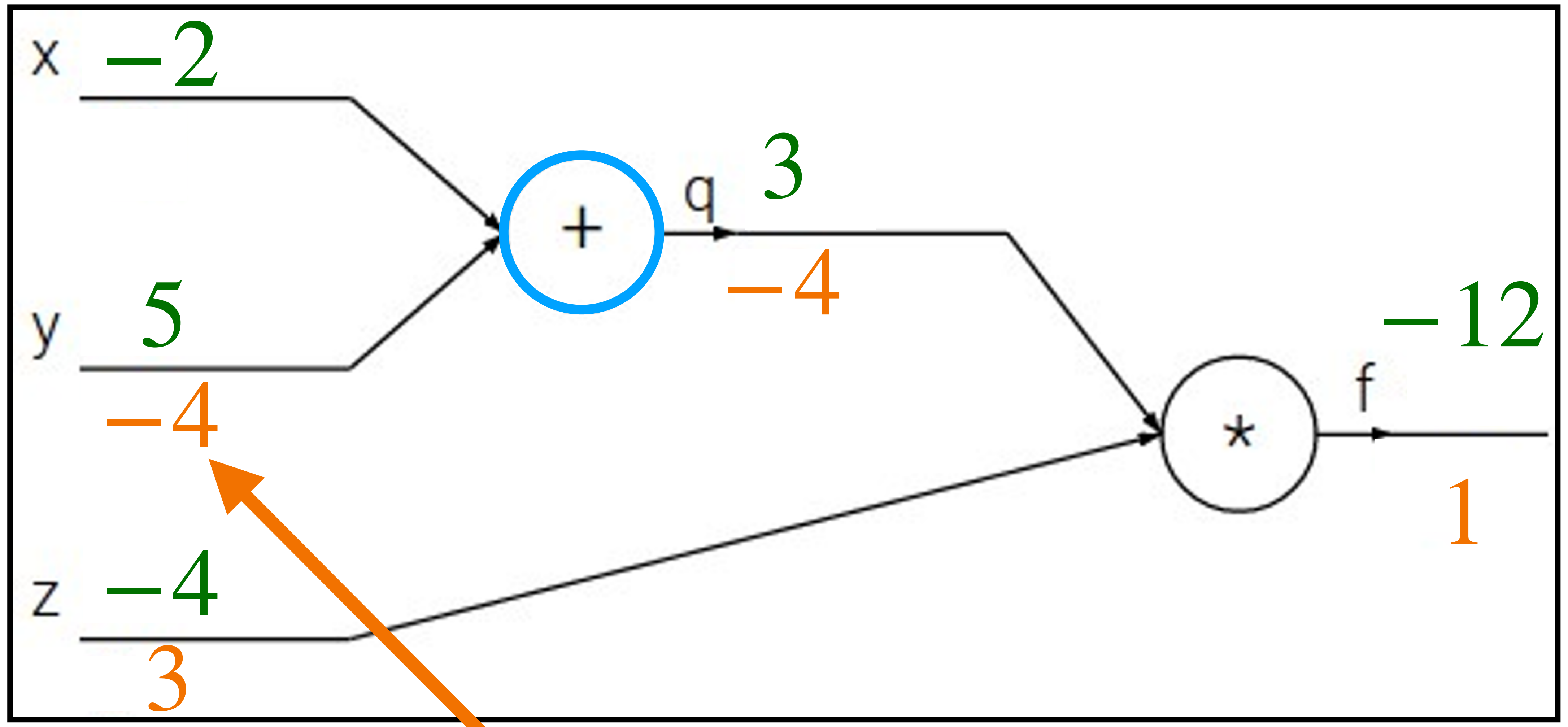
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

Downstream Gradient — Local Gradient — Upstream Gradient

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. **Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. **Backward pass**: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

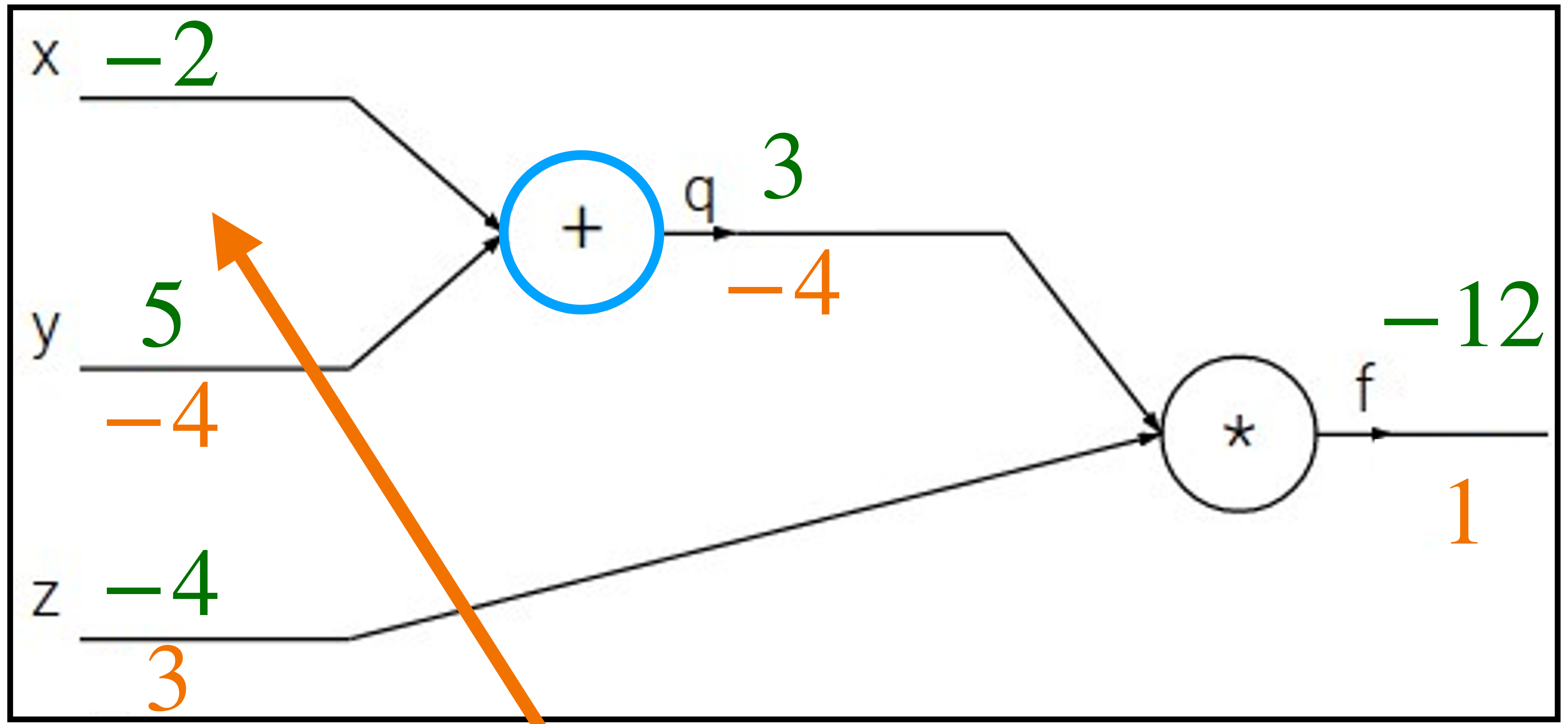
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

Downstream Gradient | Local Gradient | Upstream Gradient

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. Forward pass: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

Downstream Gradient

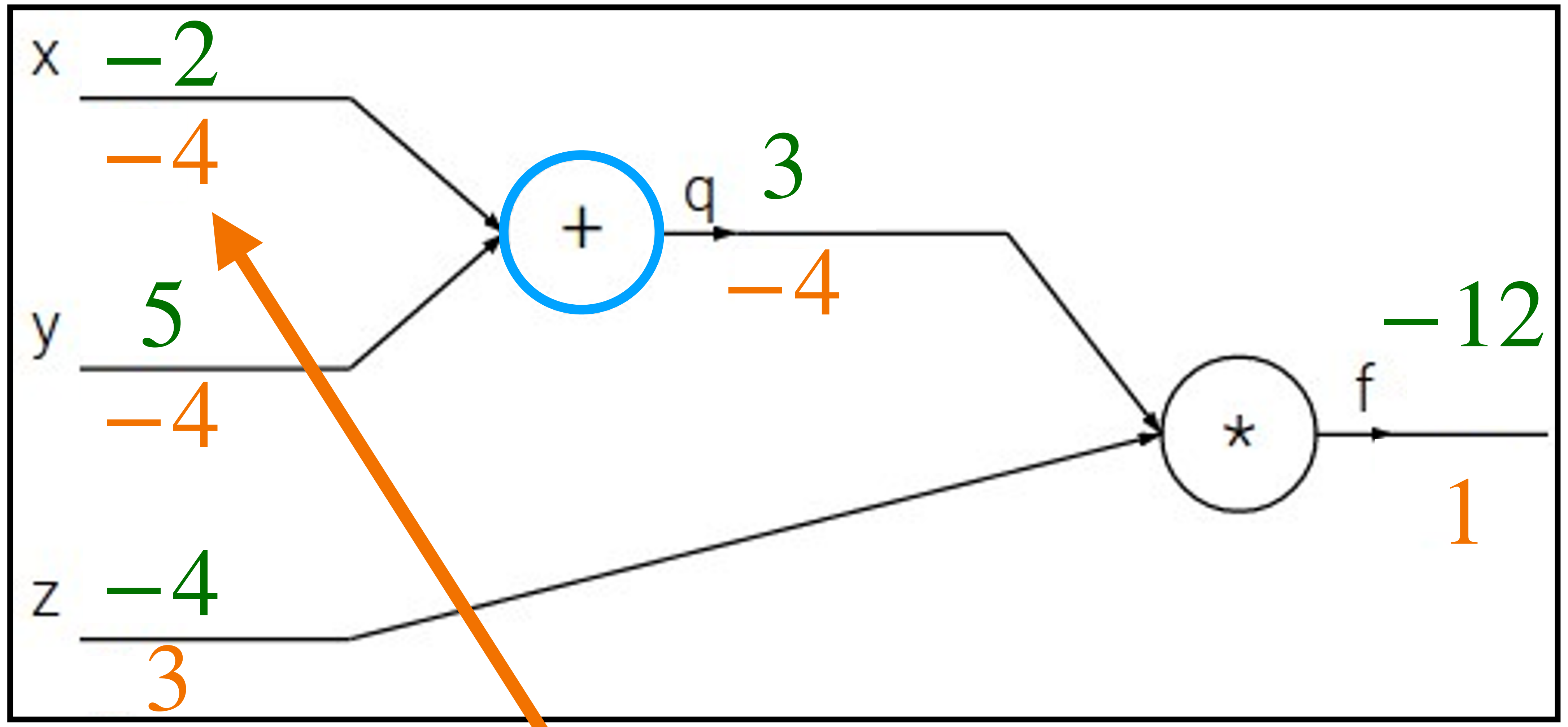
Local Gradient

Upstream Gradient

Backpropagation: Simple Example

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

e.g. $x = -2, y = 5, z = -4$



1. **Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = q \cdot z$$

2. **Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

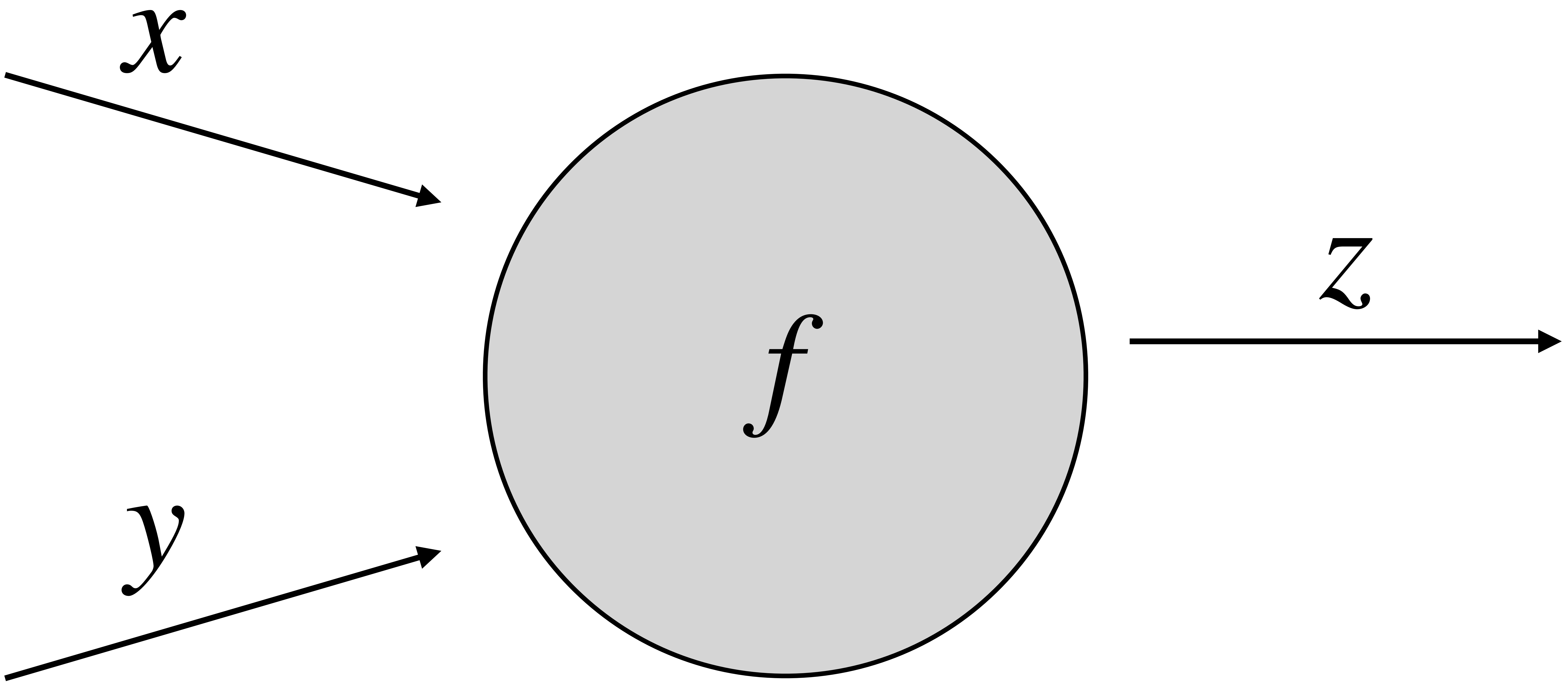
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

Downstream Gradient

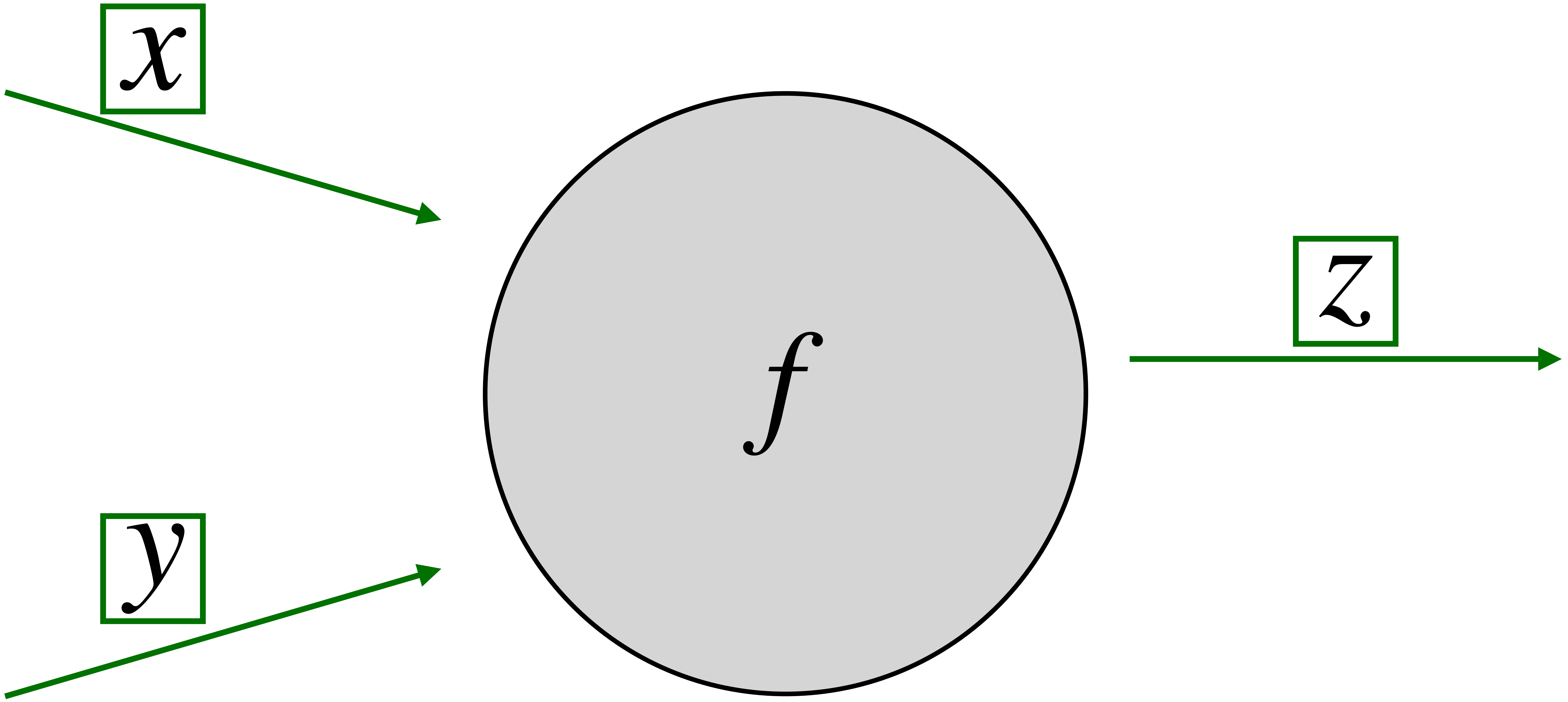
Local Gradient

Upstream Gradient

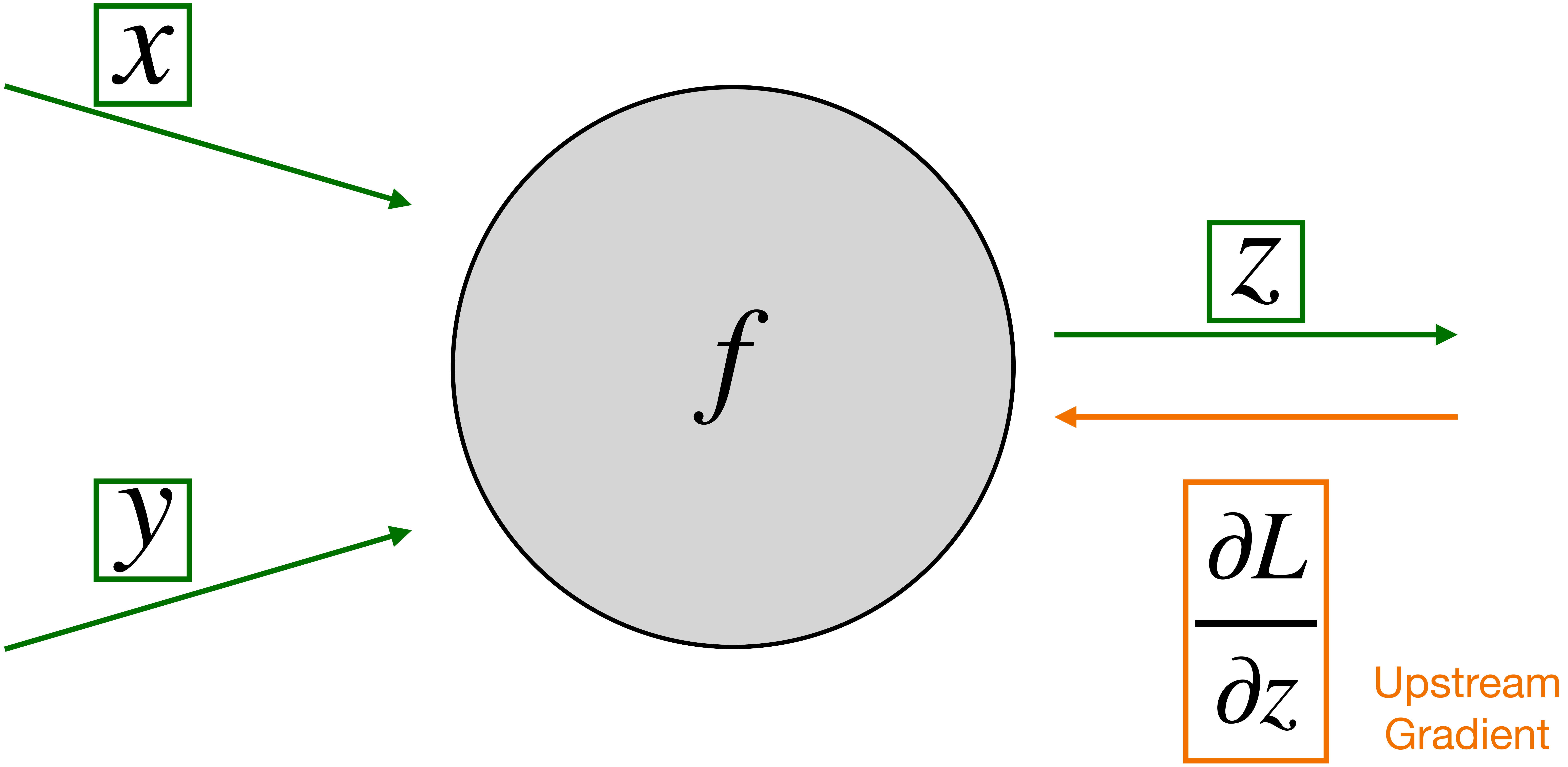
Local Properties of Backpropagation



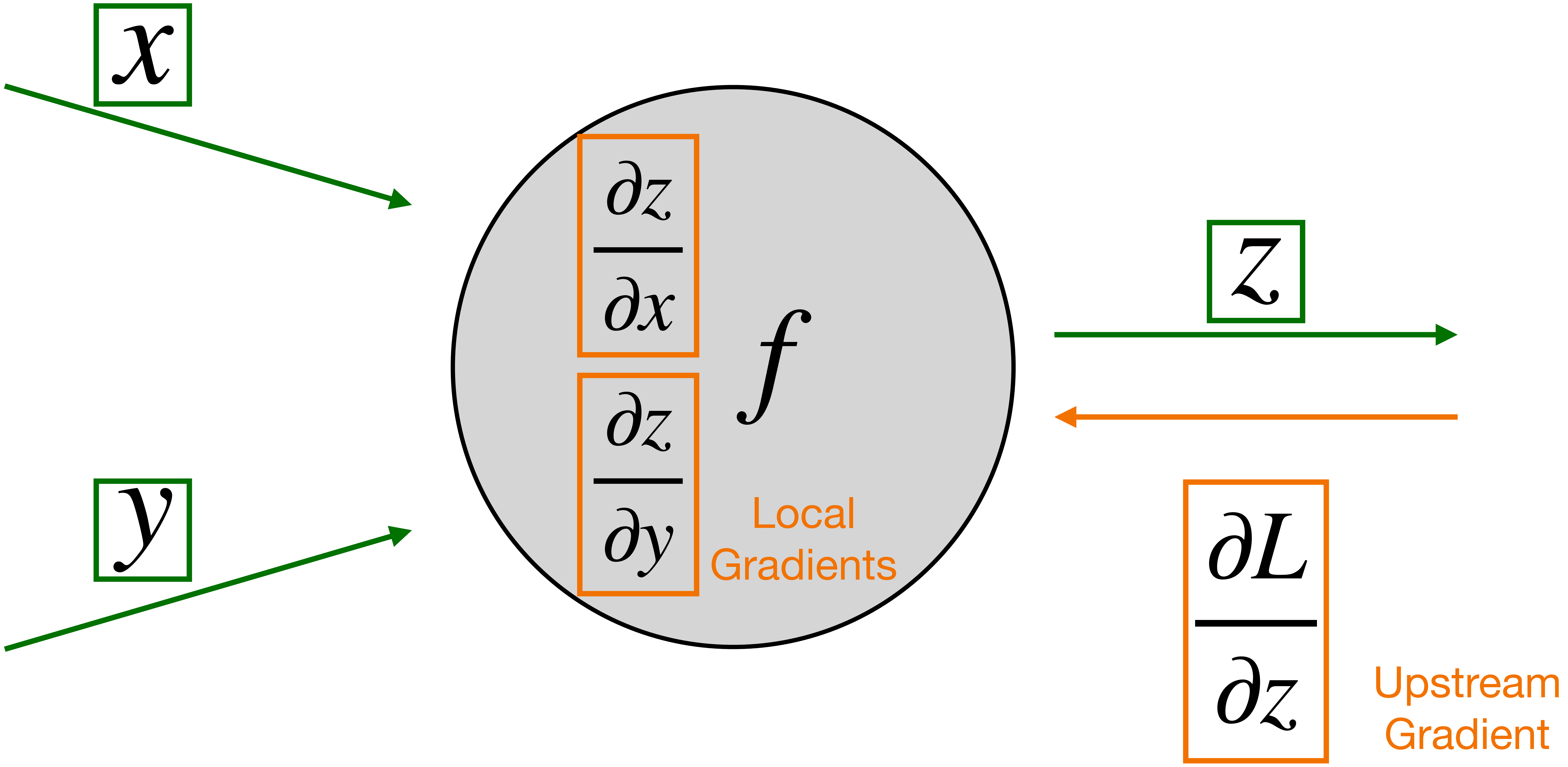
Local Properties of Backpropagation



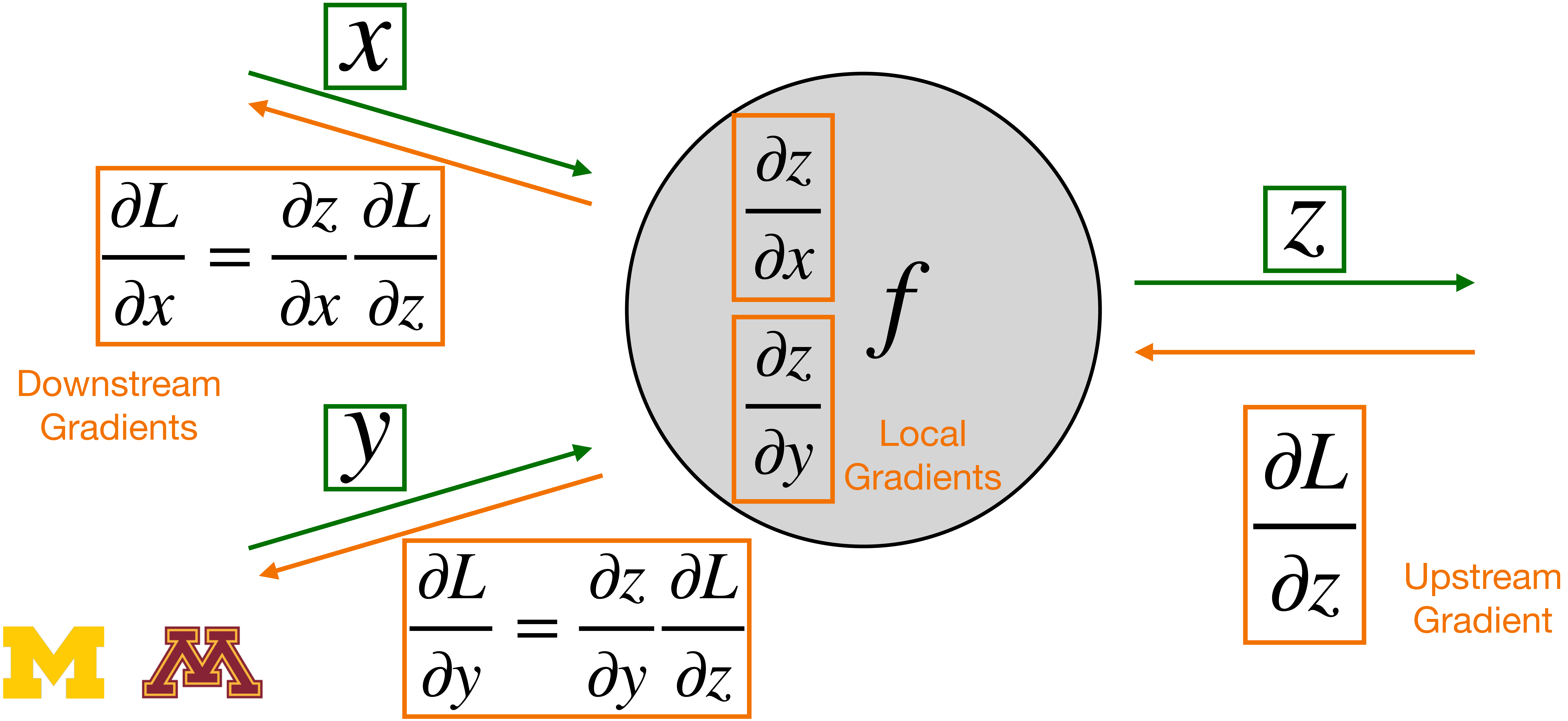
Local Properties of Backpropagation



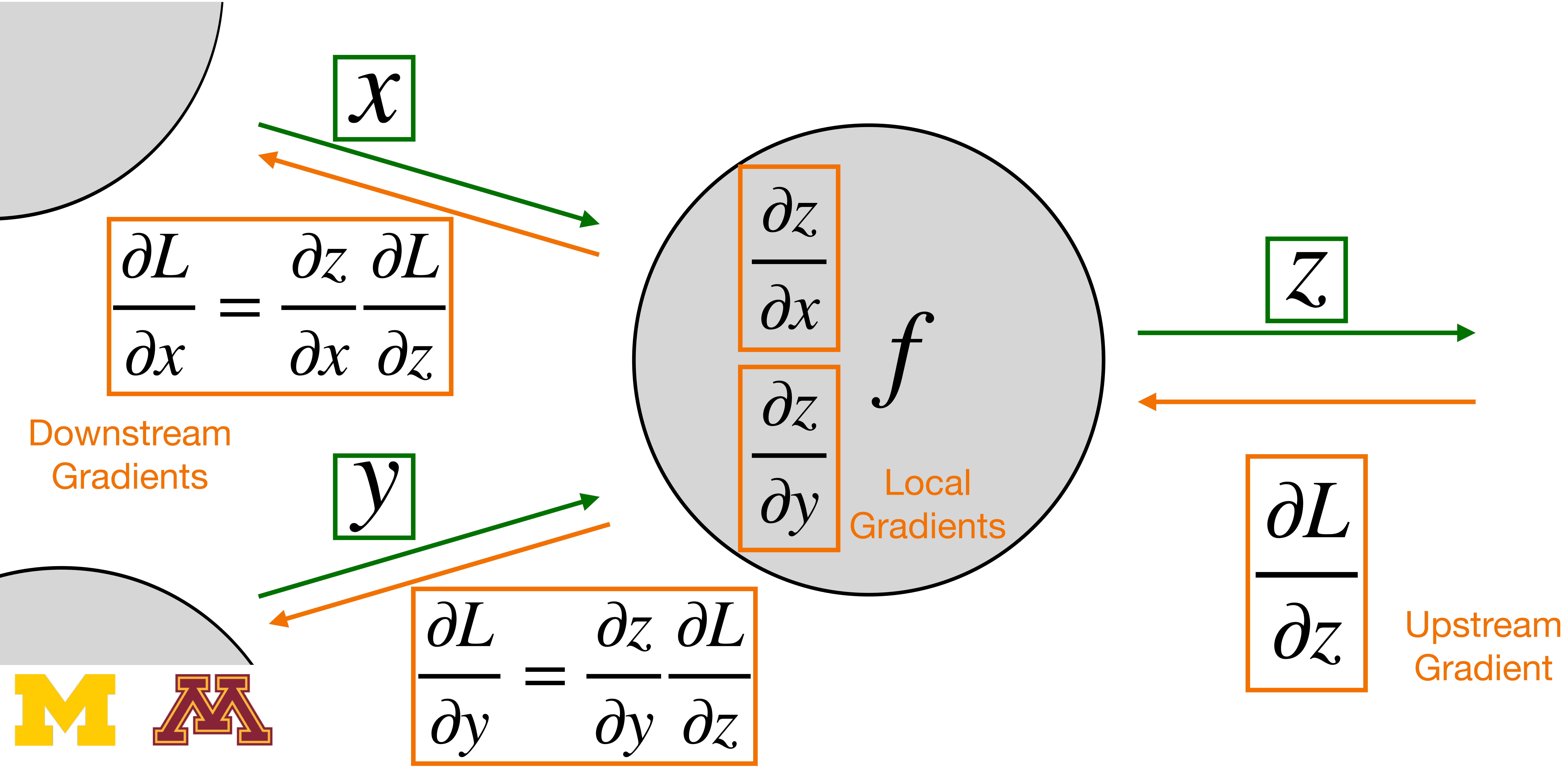
Local Properties of Backpropagation



Local Properties of Backpropagation



Local Properties of Backpropagation

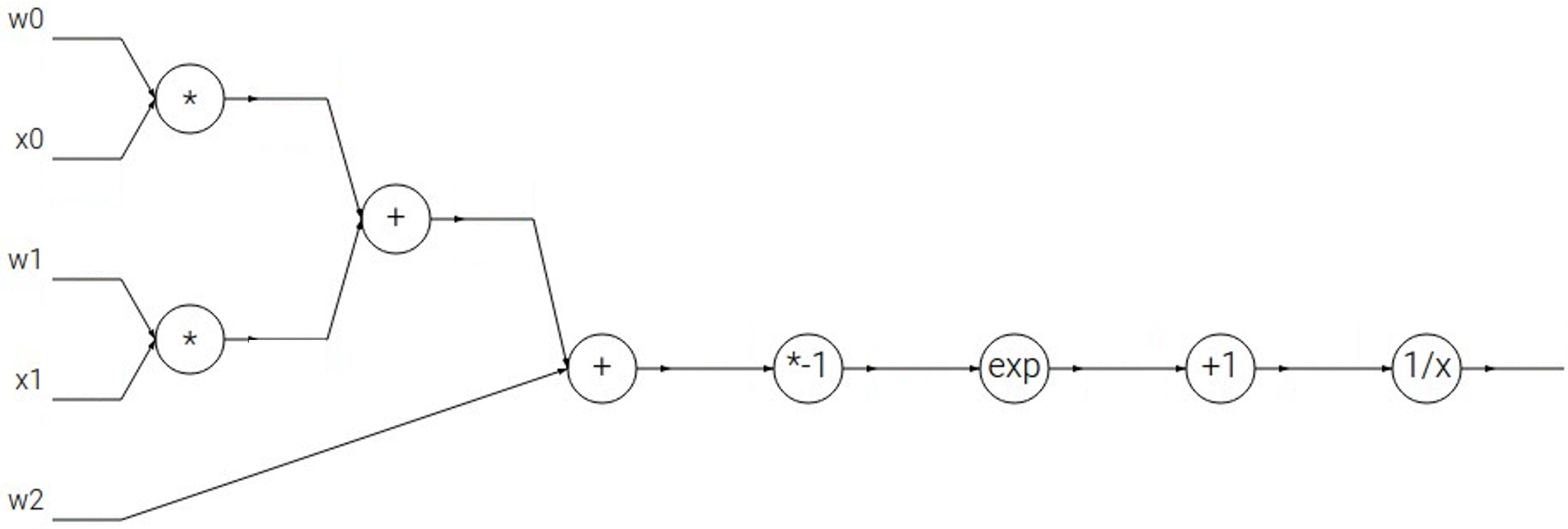


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

Another example

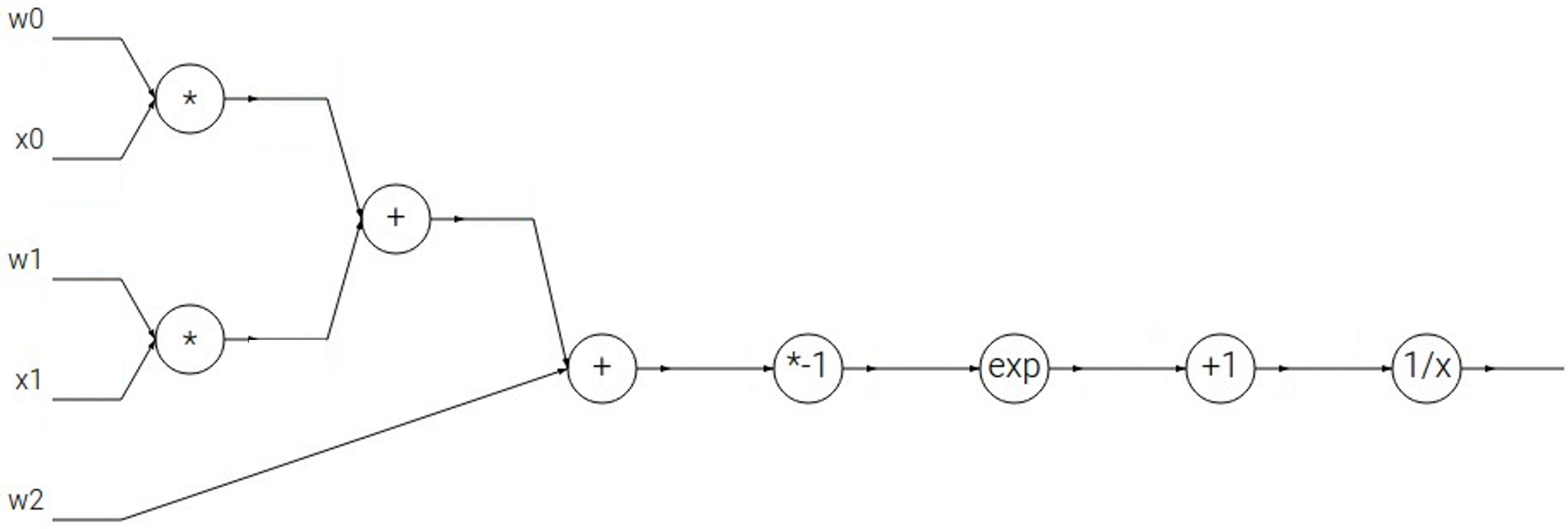
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0+w_1x_1+w2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

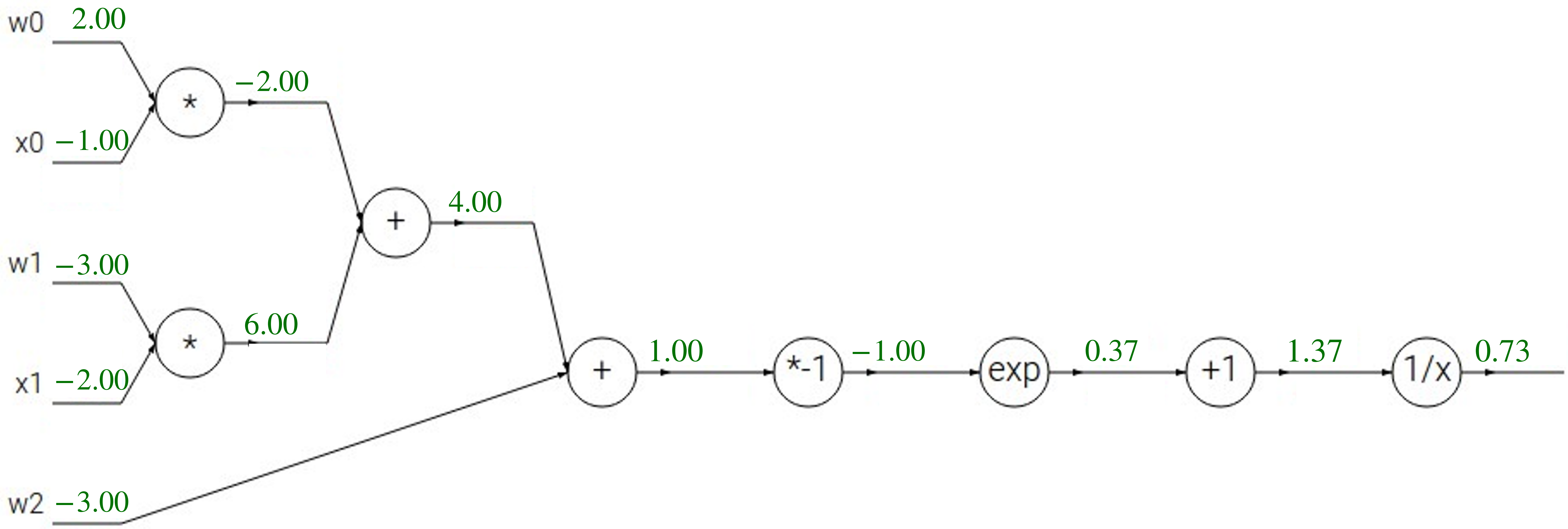
1. Forward pass: Compute outputs



Another example

$$f(x,w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. Forward pass: Compute outputs

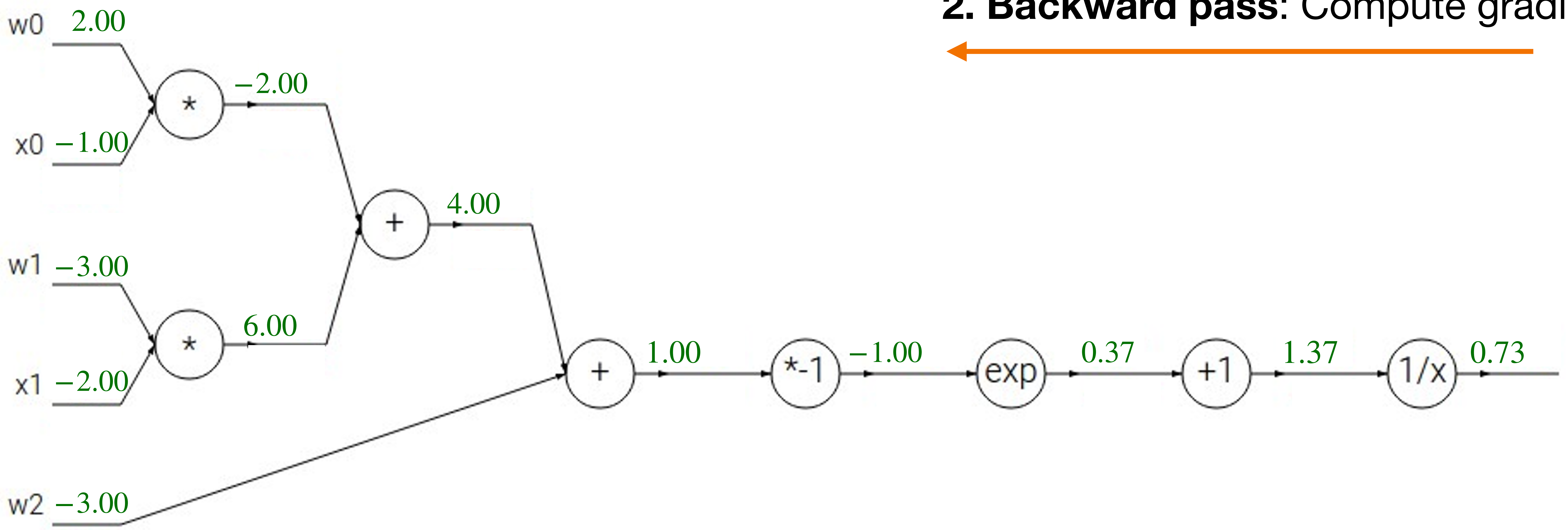


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

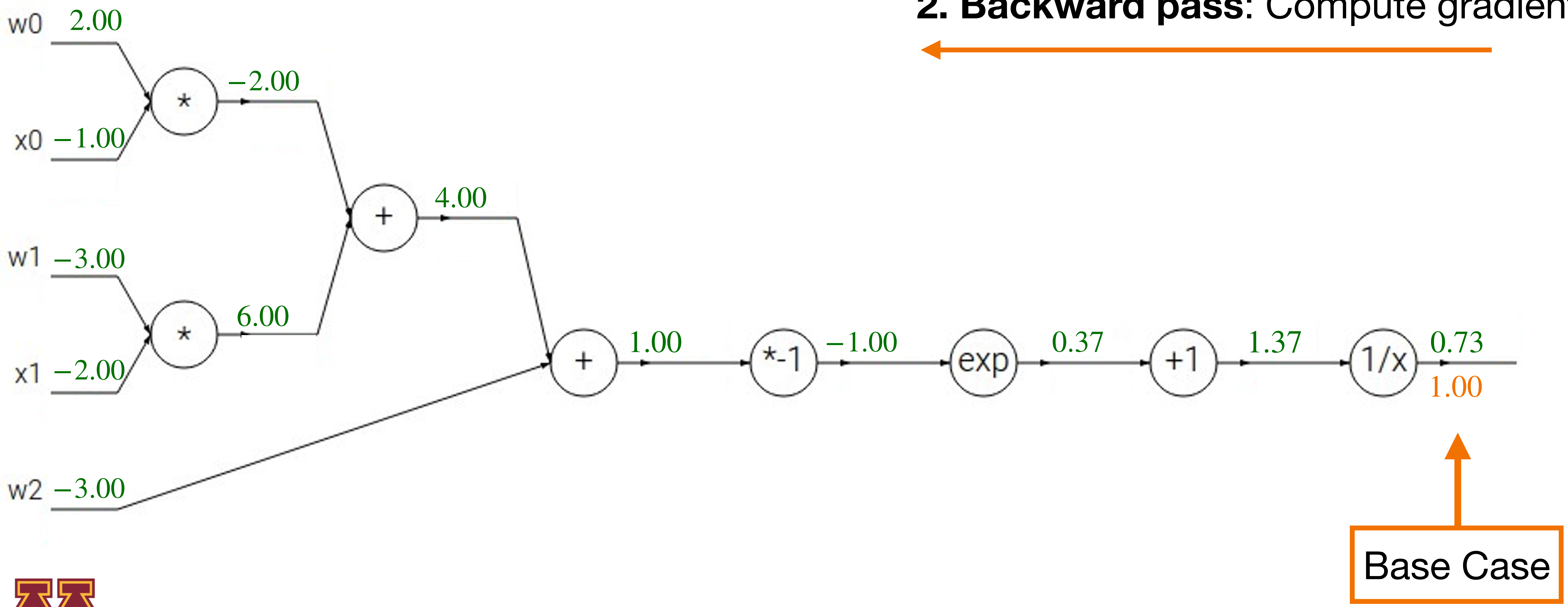


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

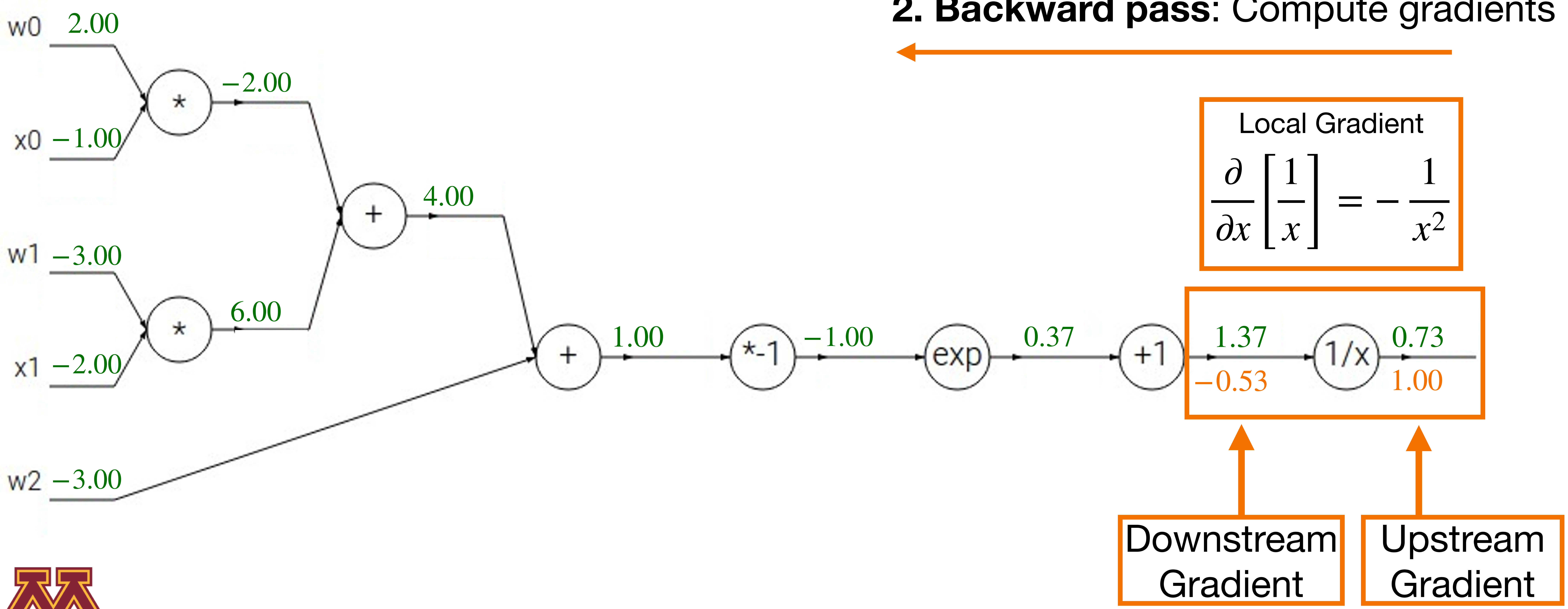


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. Forward pass: Compute outputs

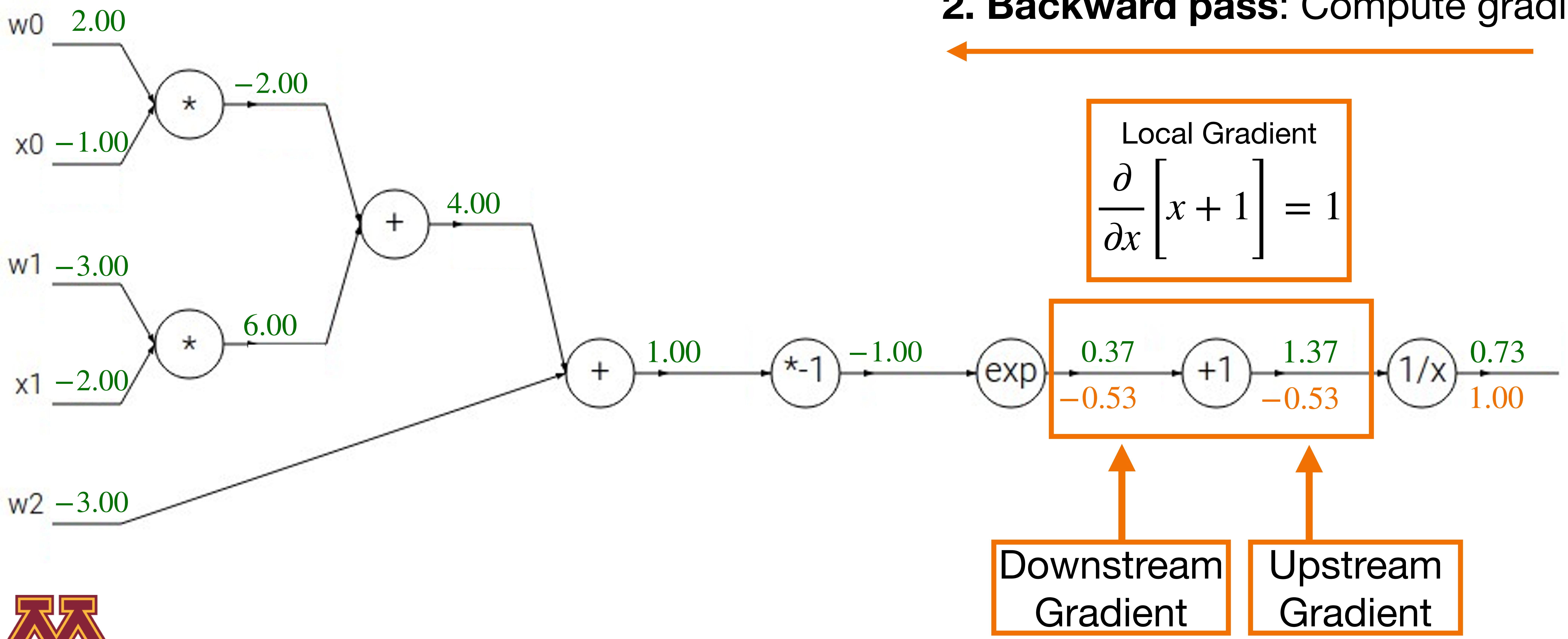
2. Backward pass: Compute gradients



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. **Forward pass**: Compute outputs
2. **Backward pass**: Compute gradients

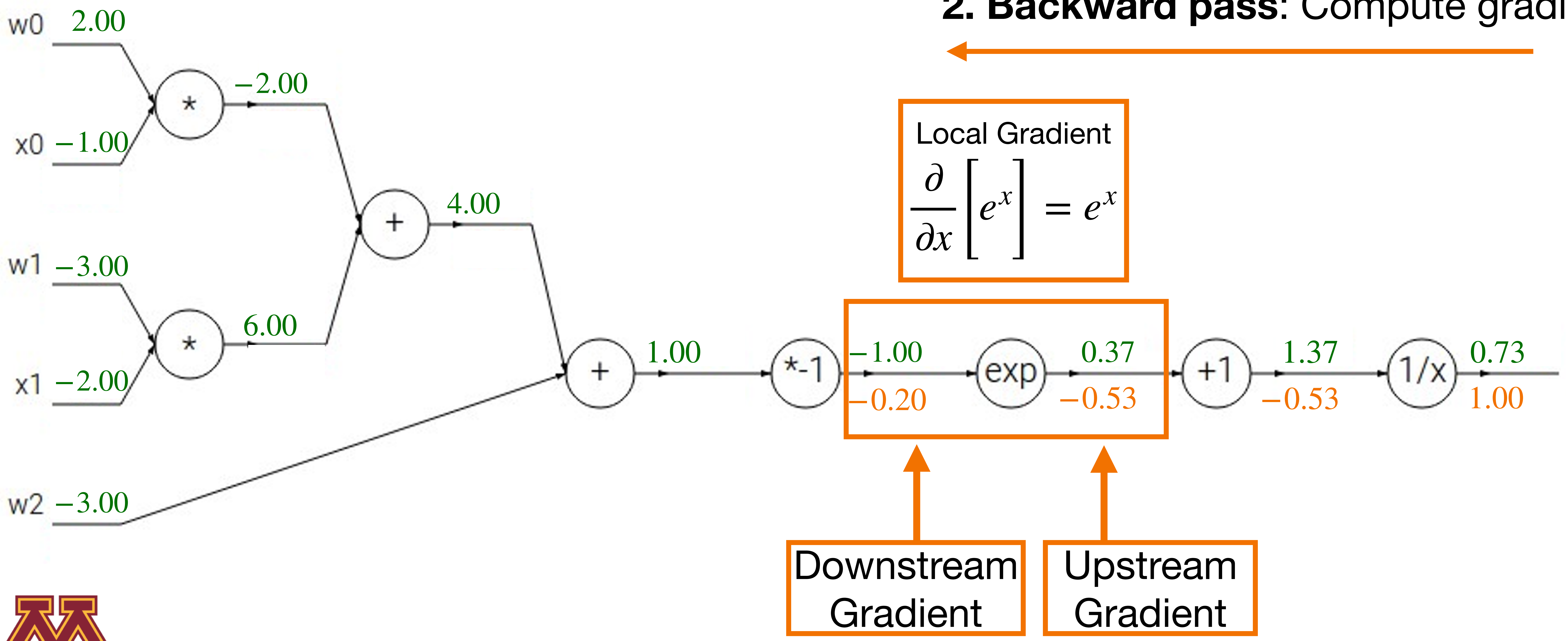


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. **Forward pass**: Compute outputs

2. **Backward pass**: Compute gradients

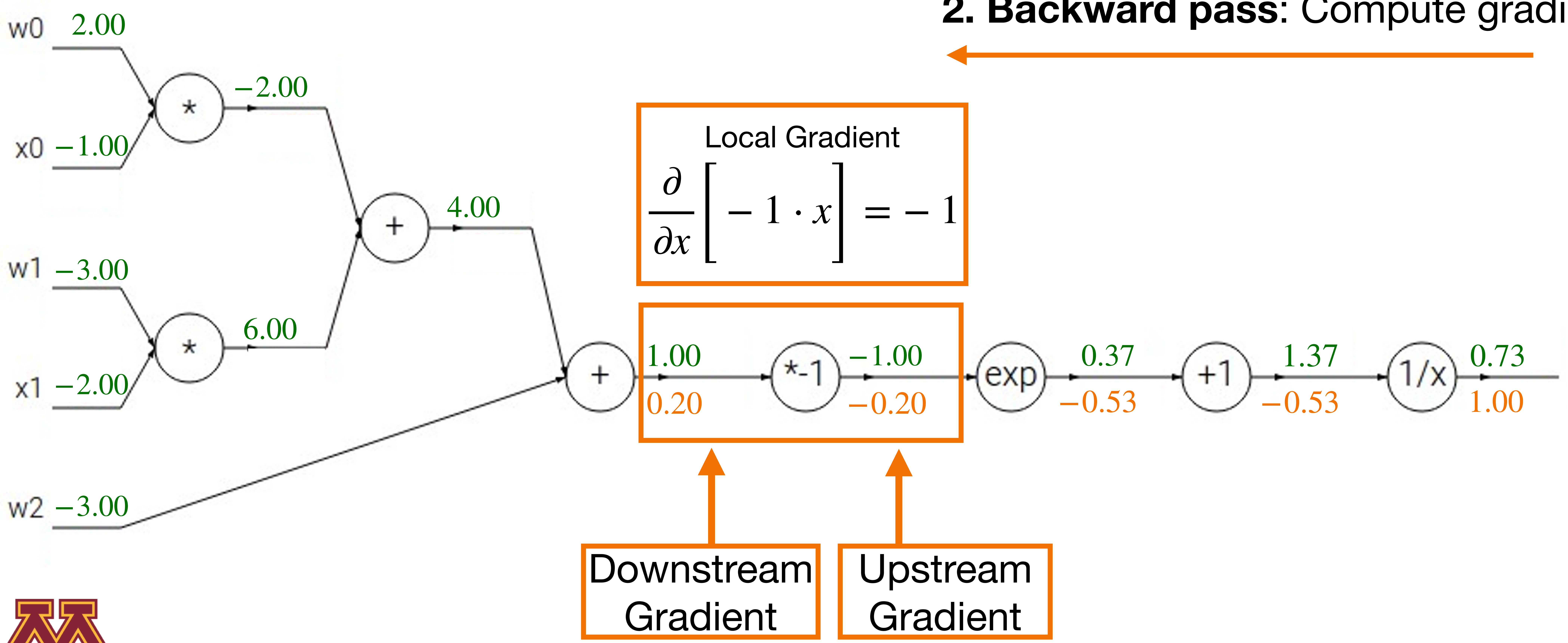


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. Forward pass: Compute outputs

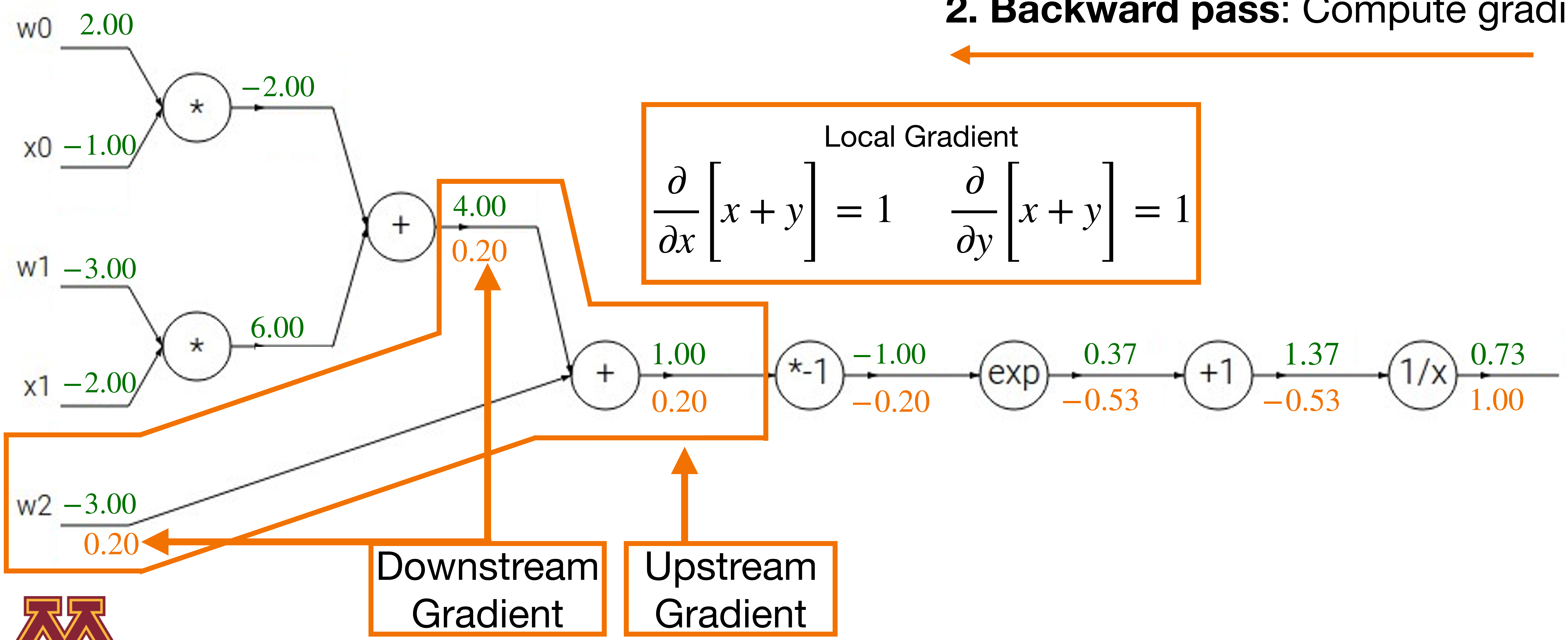
2. Backward pass: Compute gradients



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

1. **Forward pass**: Compute outputs
2. **Backward pass**: Compute gradients

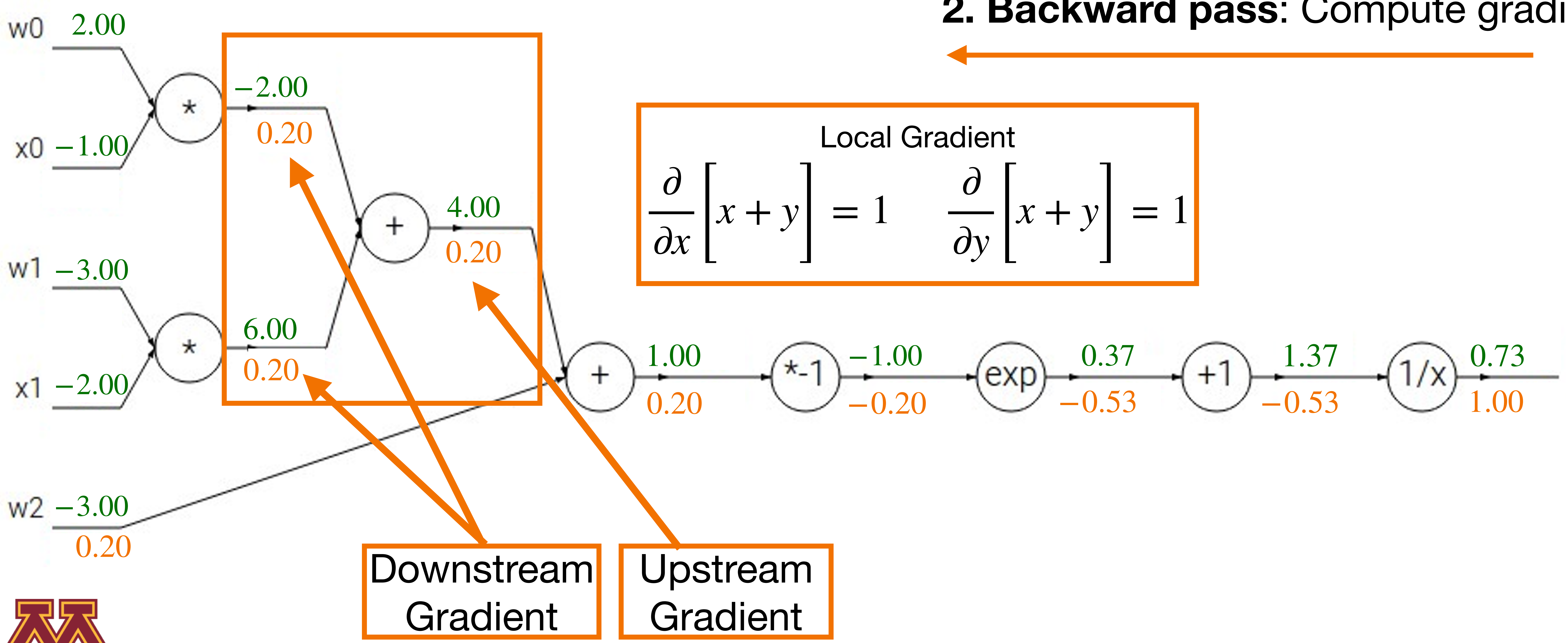


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0+w_1x_1+w2)}}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients



Local Gradient

$$\frac{\partial}{\partial x}\left[x+y\right] = 1 \qquad \frac{\partial}{\partial y}\left[x+y\right] = 1$$

Another example

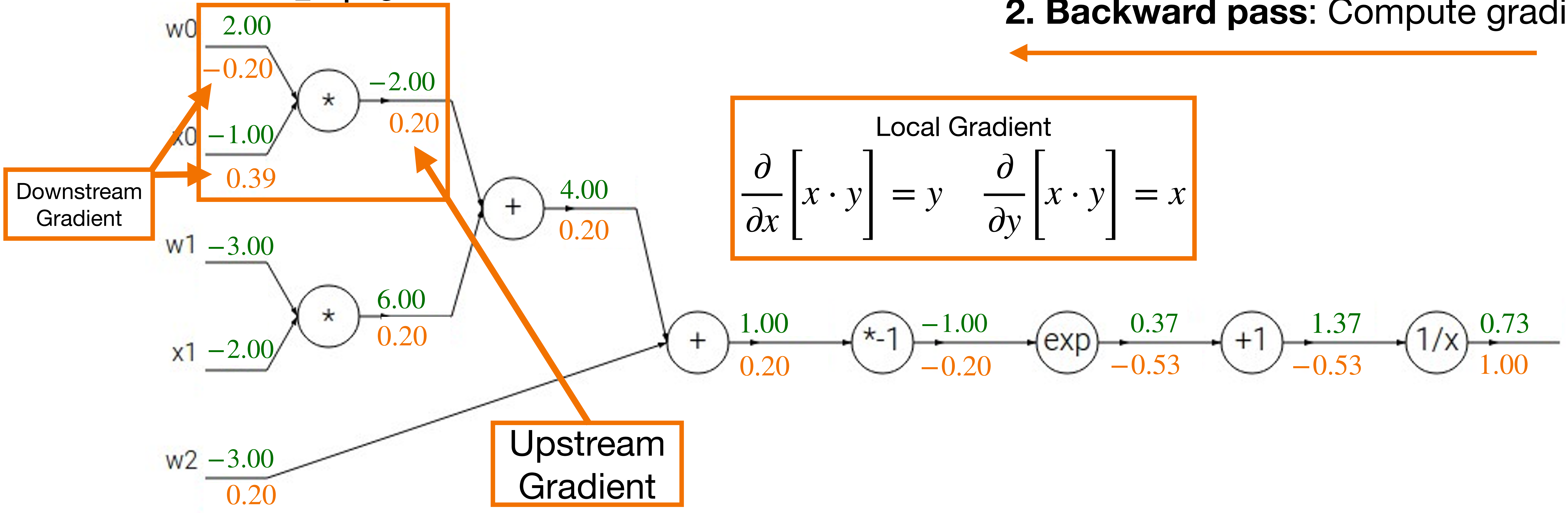
$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w2)}}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Local Gradient

$$\frac{\partial}{\partial x}\left[x \cdot y\right] = y \quad \frac{\partial}{\partial y}\left[x \cdot y\right] = x$$



Another example

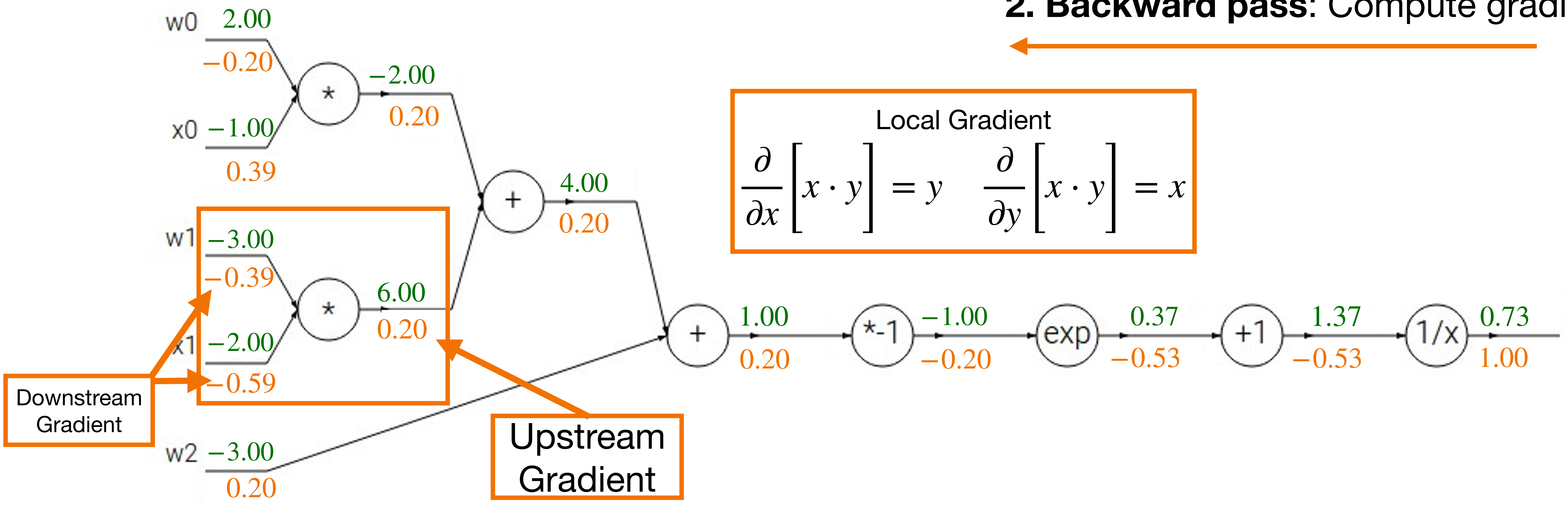
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0+w_1x_1+w2)}}$$

1. **Forward pass**: Compute outputs

2. **Backward pass**: Compute gradients

Local Gradient

$$\frac{\partial}{\partial x}\left[x \cdot y\right] = y \qquad \frac{\partial}{\partial y}\left[x \cdot y\right] = x$$



Another example

$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w2)}}$$

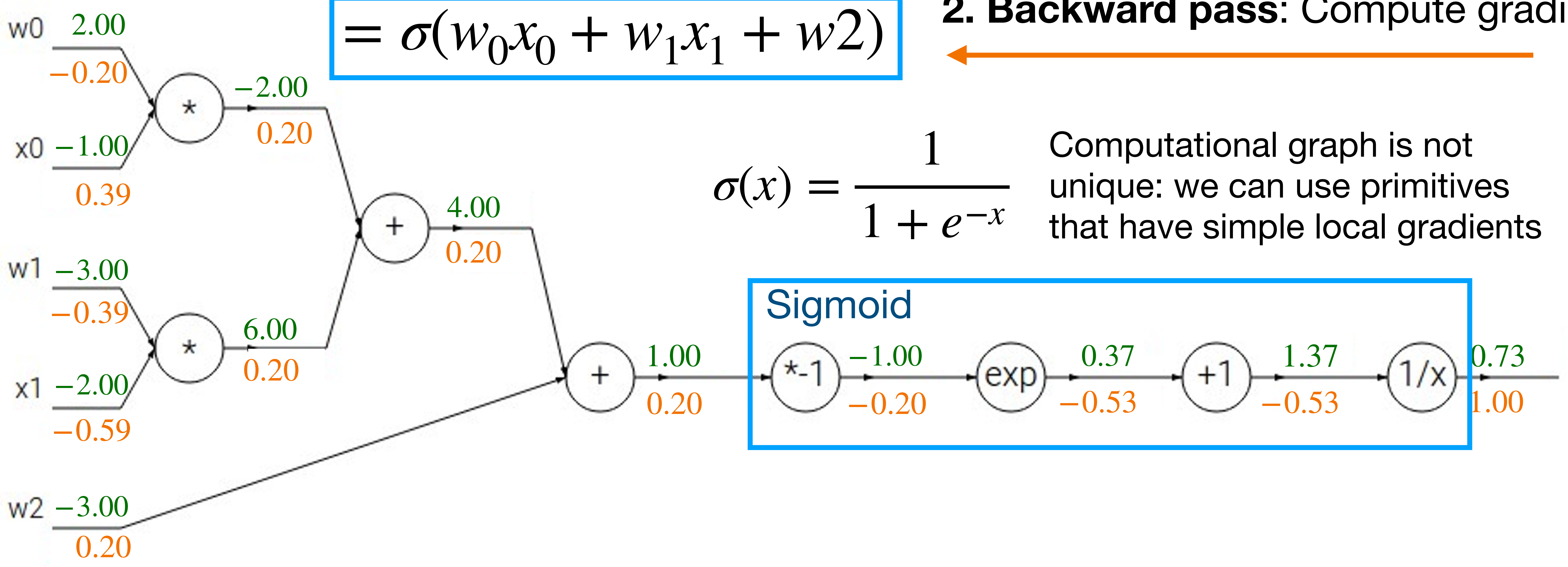
$$\boxed{= \sigma(w_0x_0 + w_1x_1 + w2)}$$

1. **Forward pass**: Compute outputs

2. **Backward pass**: Compute gradients

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Computational graph is not unique: we can use primitives that have simple local gradients



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

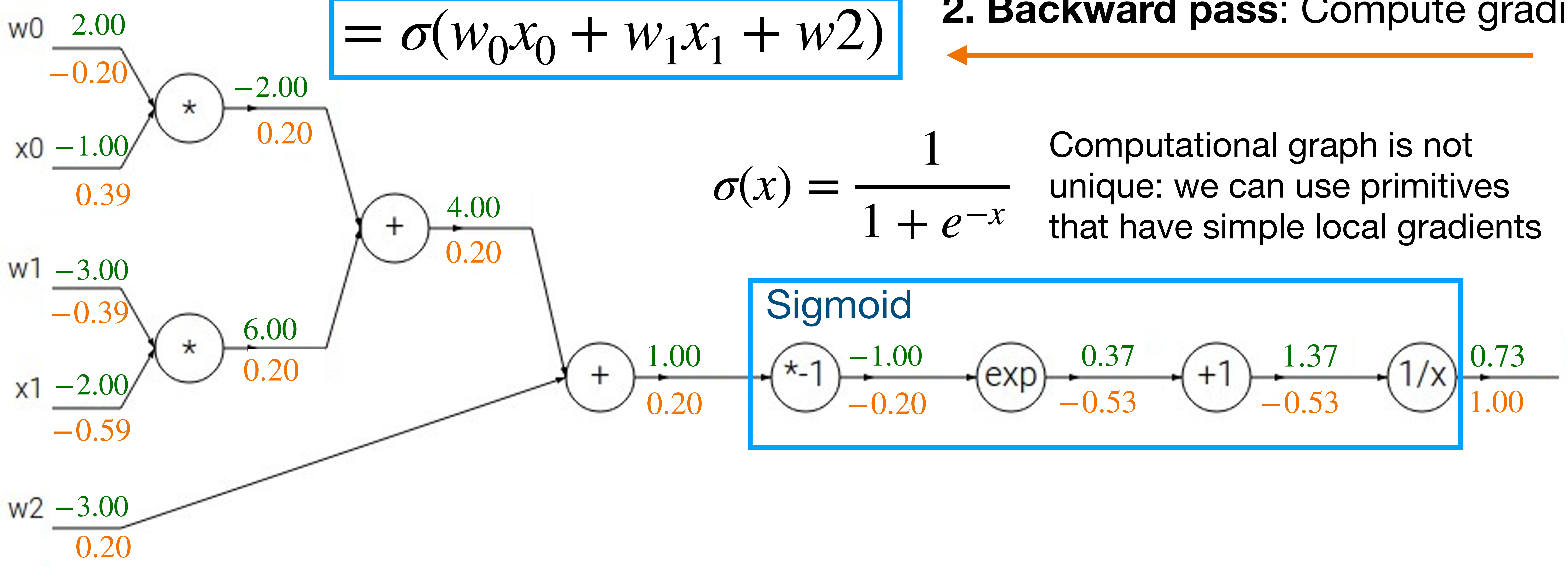
$$\boxed{= \sigma(w_0 x_0 + w_1 x_1 + w2)}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Computational graph is not unique: we can use primitives that have simple local gradients



Sigmoid local gradient: $\frac{\partial}{\partial x}\left[\sigma(x)\right] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right)\left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x))\sigma(x)$

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w2)}}$$

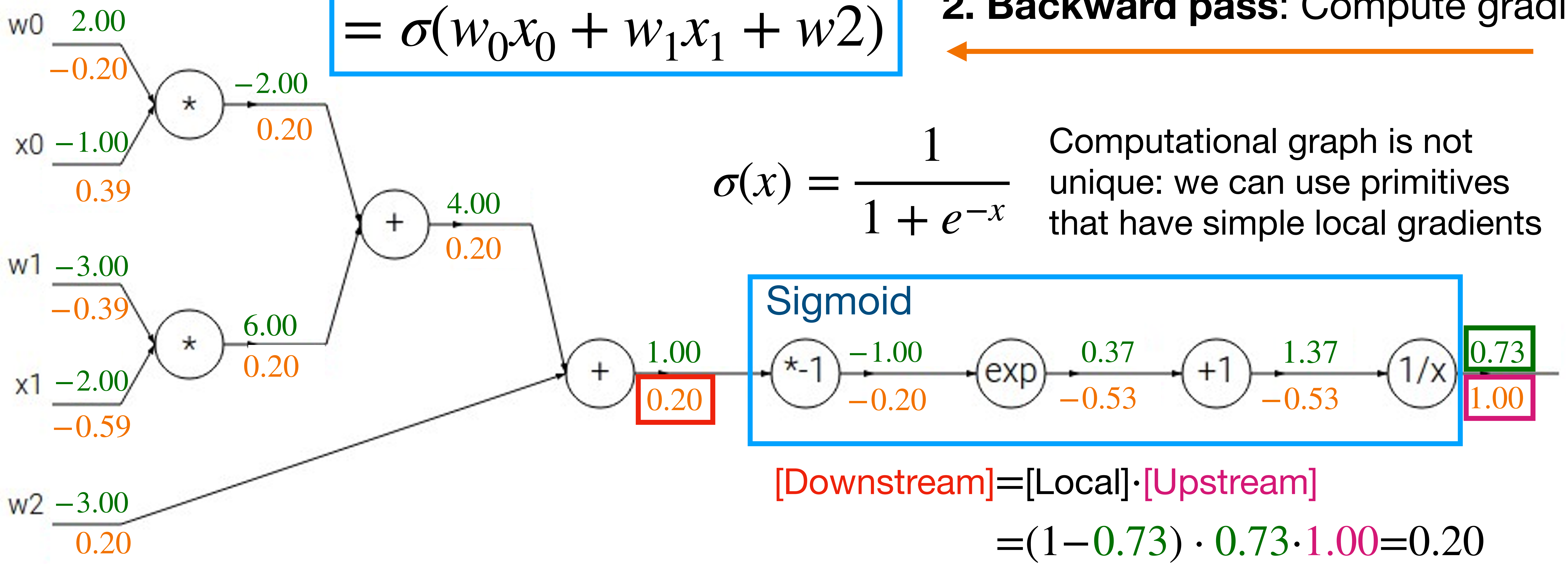
$$\boxed{= \sigma(w_0 x_0 + w_1 x_1 + w2)}$$

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Computational graph is not unique: we can use primitives that have simple local gradients



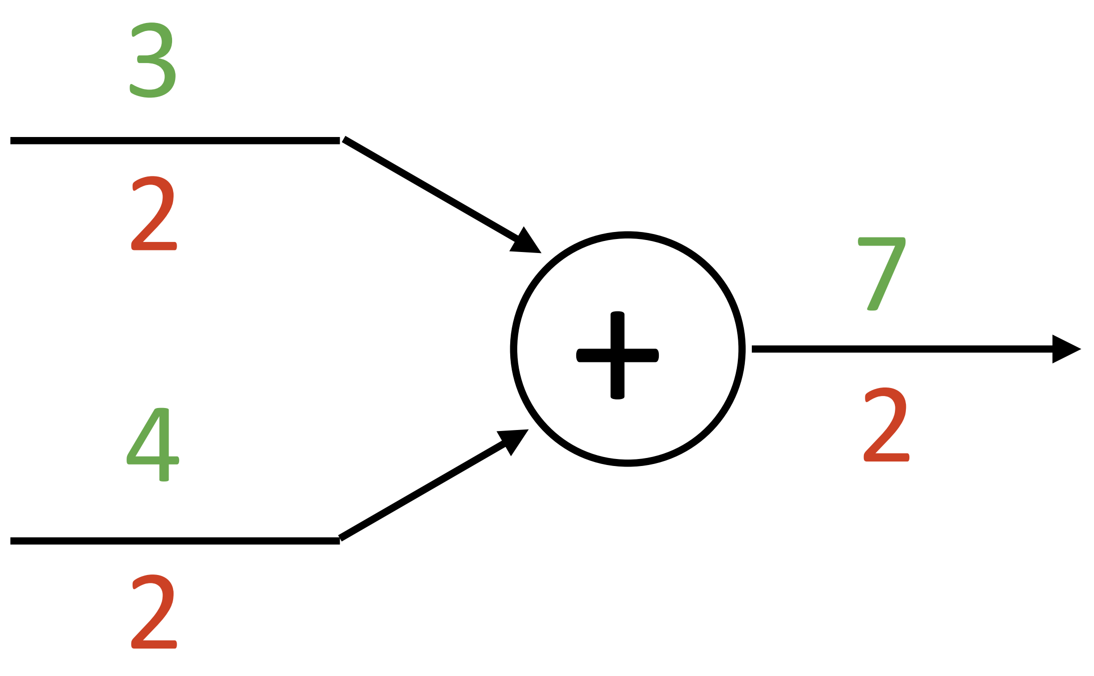
[Downstream]=[Local]·[Upstream]

$$=(1-0.73)\cdot 0.73 \cdot 1.00 = 0.20$$

Sigmoid local gradient: $\frac{\partial}{\partial x}\left[\sigma(x)\right] = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$

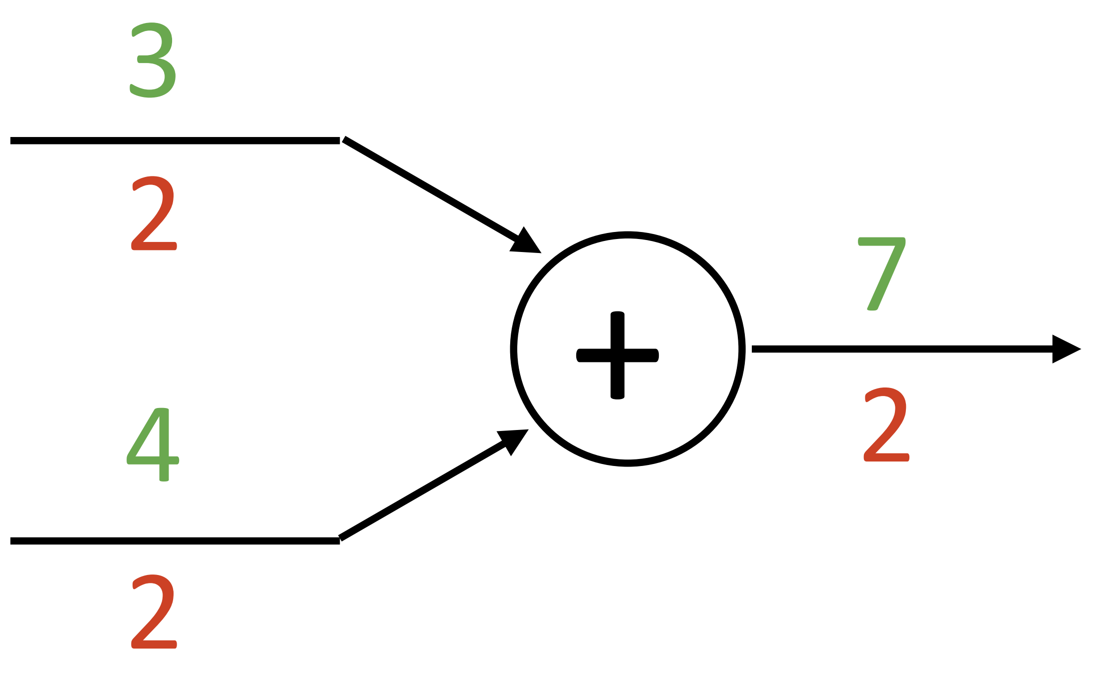
Patterns in Gradient Flow

add gate: gradient distributor

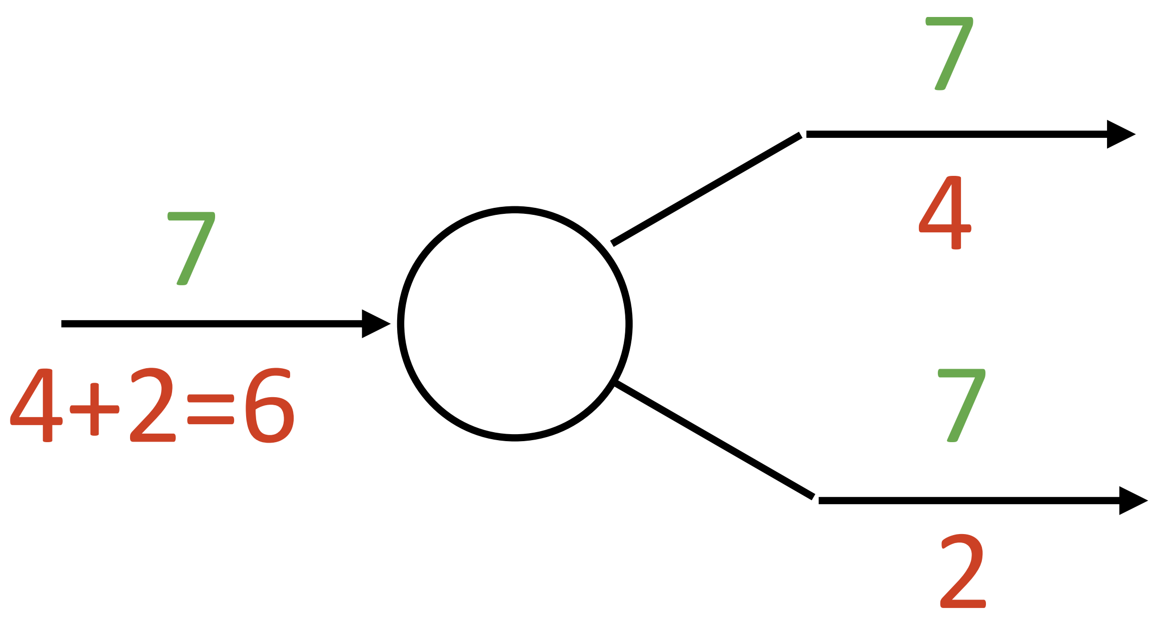


Patterns in Gradient Flow

add gate: gradient distributor

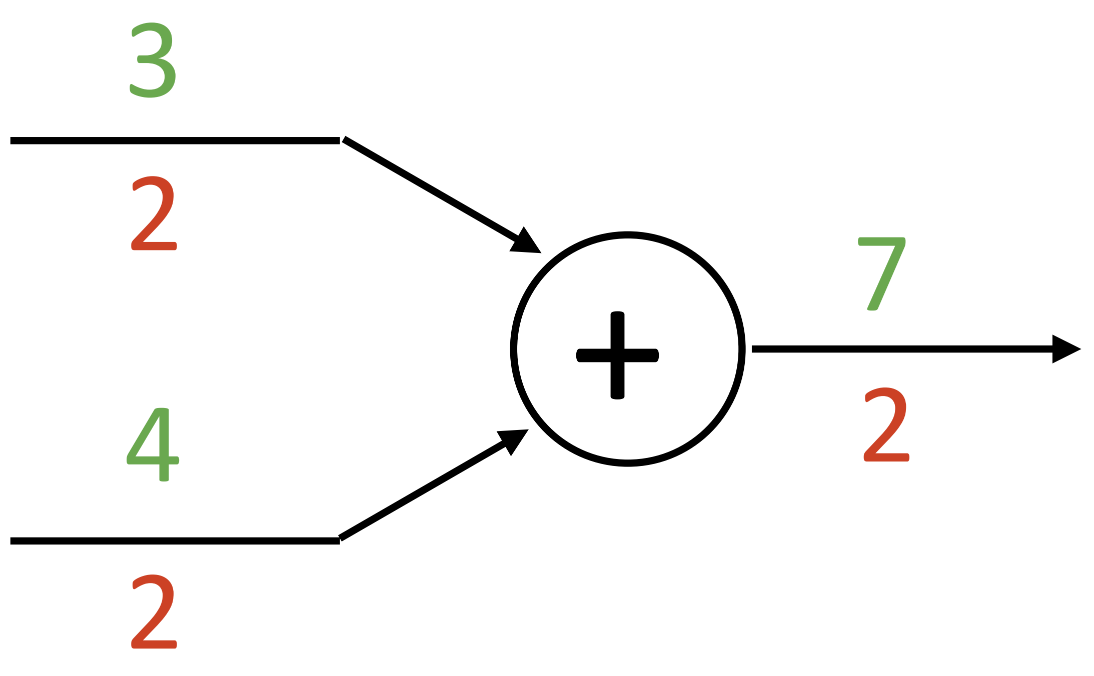


copy gate: gradient adder

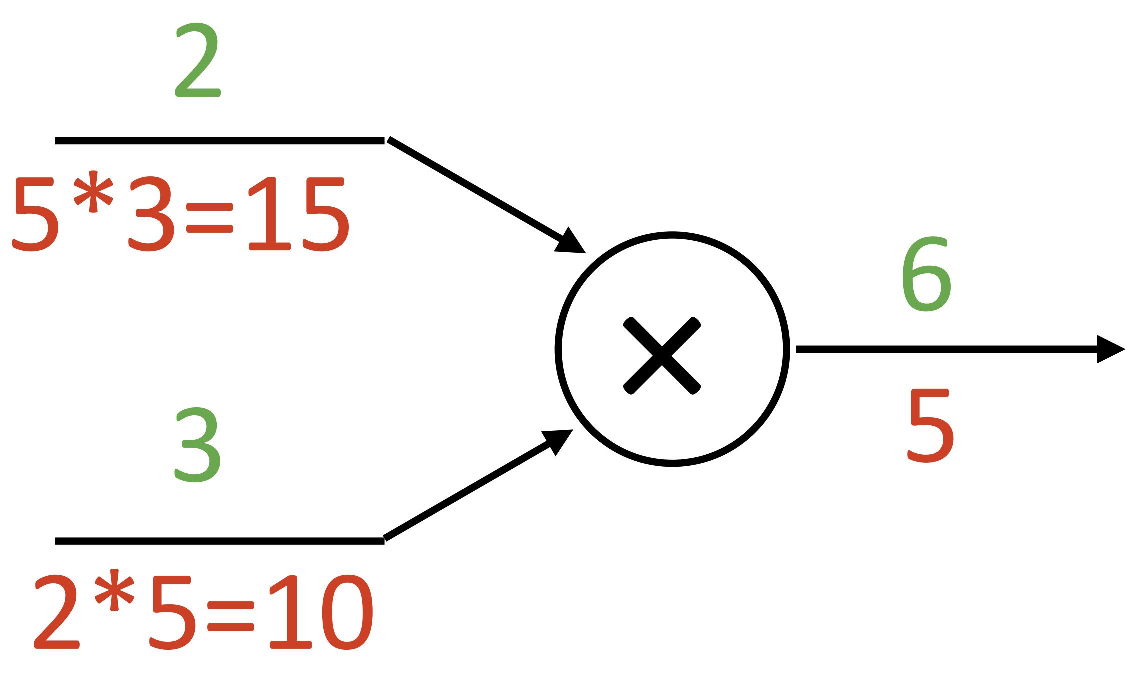


Patterns in Gradient Flow

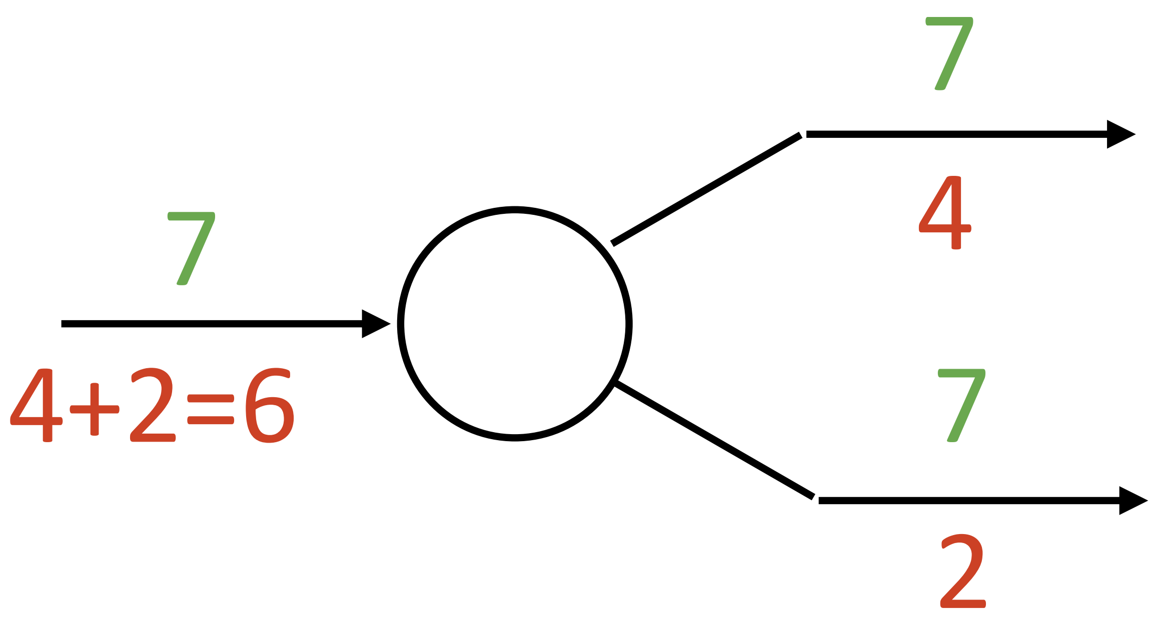
add gate: gradient distributor



mul gate: "swap multiplier"

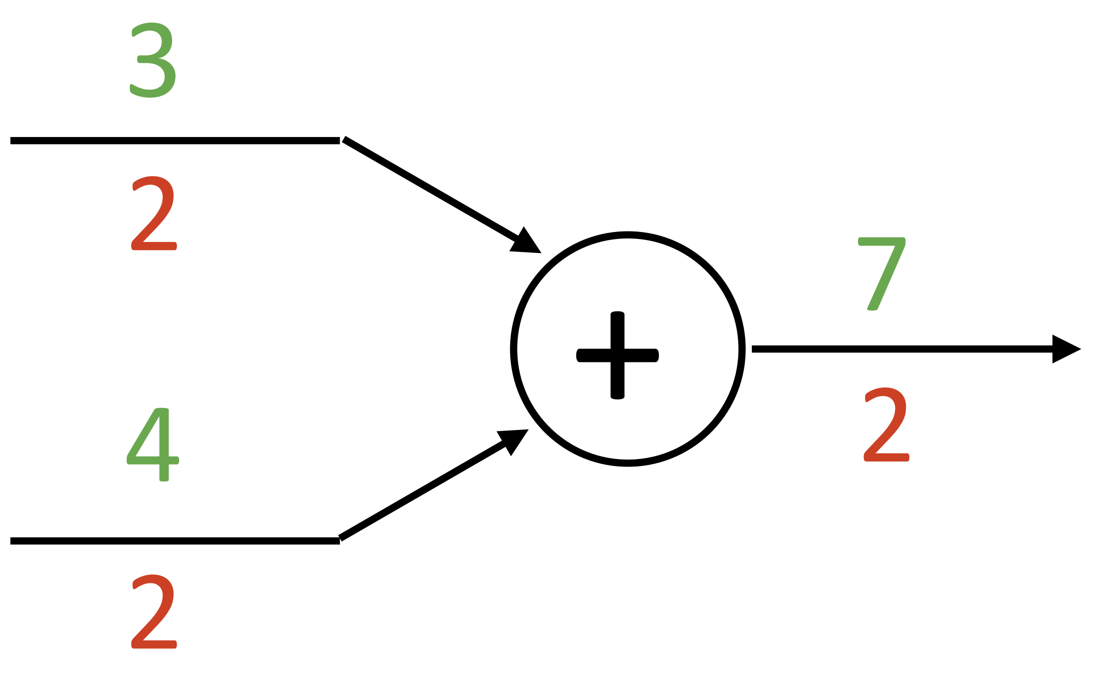


copy gate: gradient adder

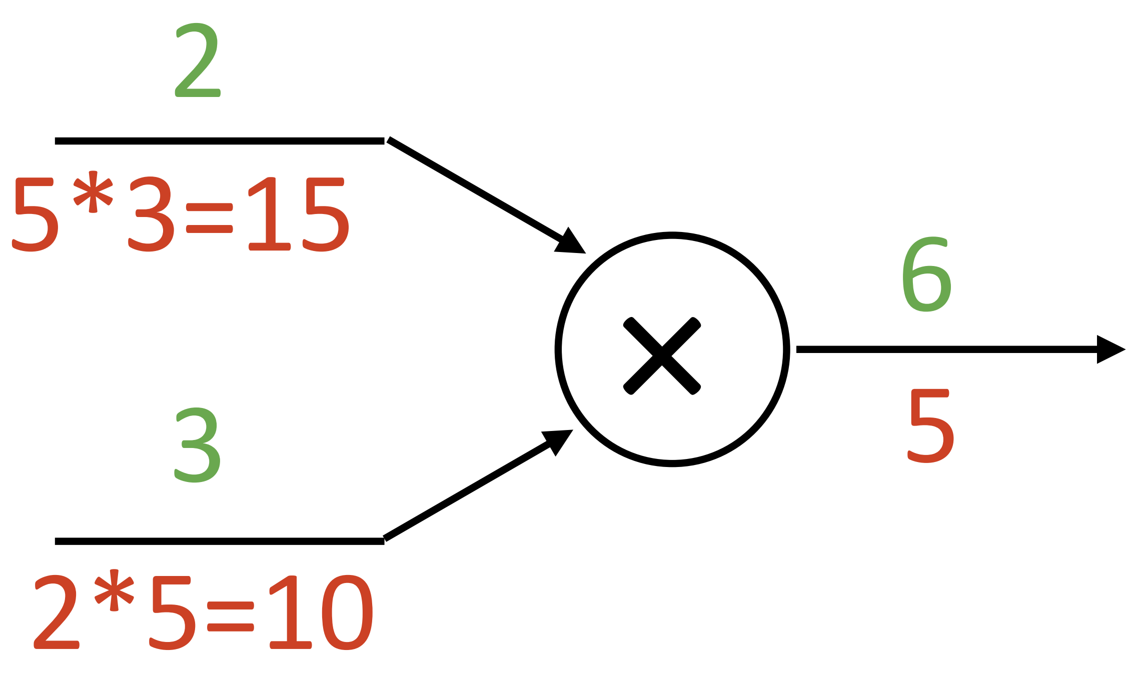


Patterns in Gradient Flow

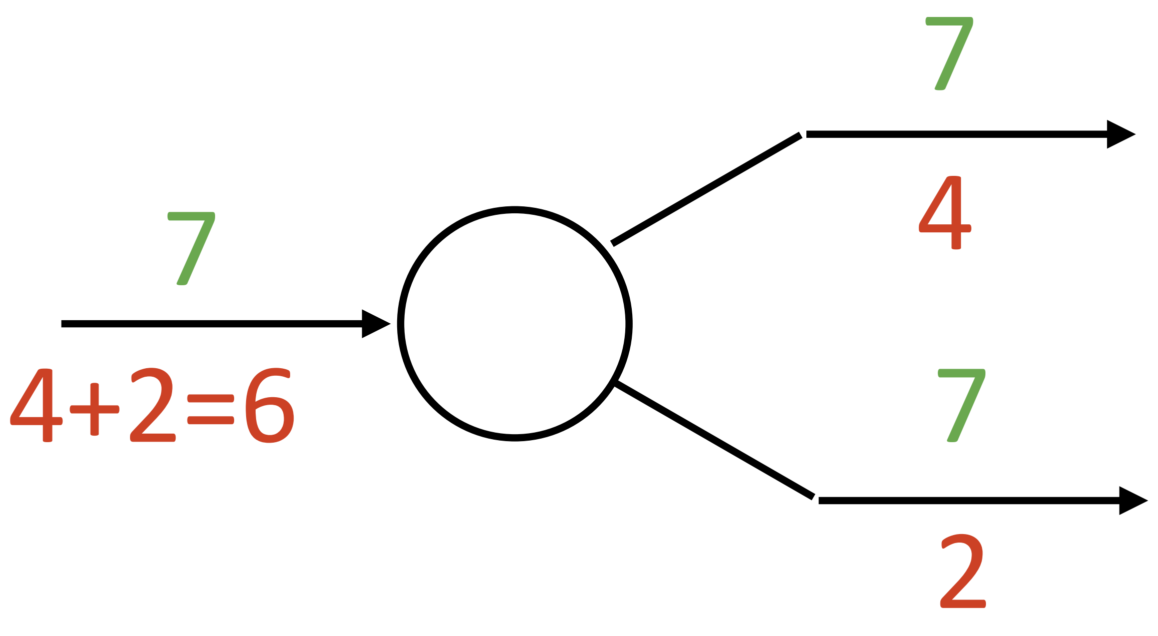
add gate: gradient distributor



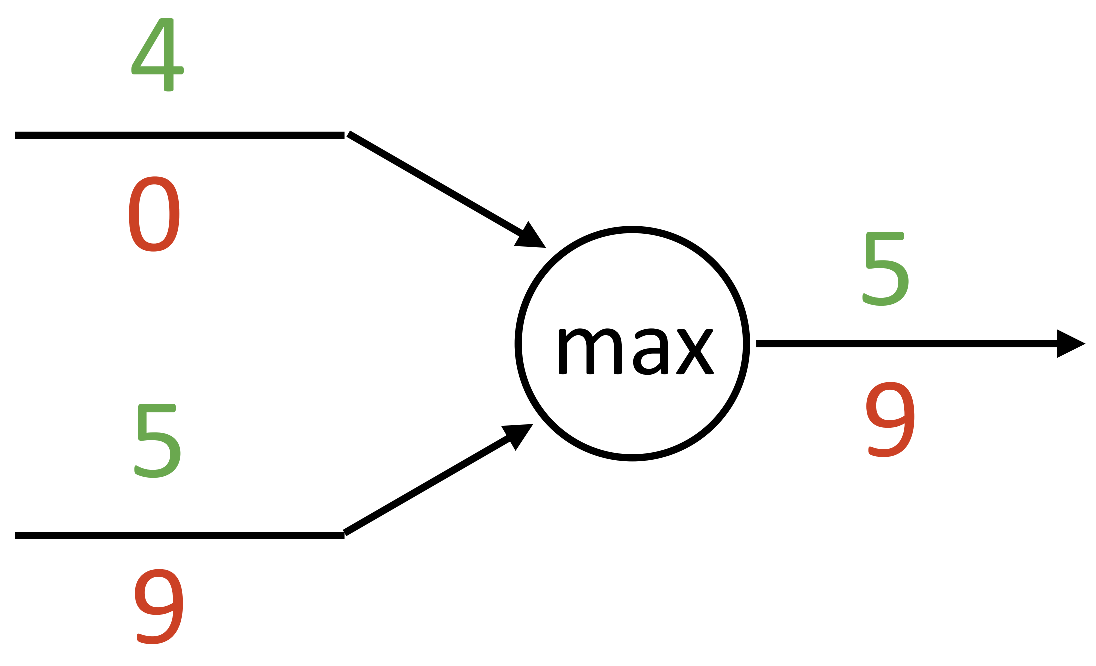
mul gate: "swap multiplier"



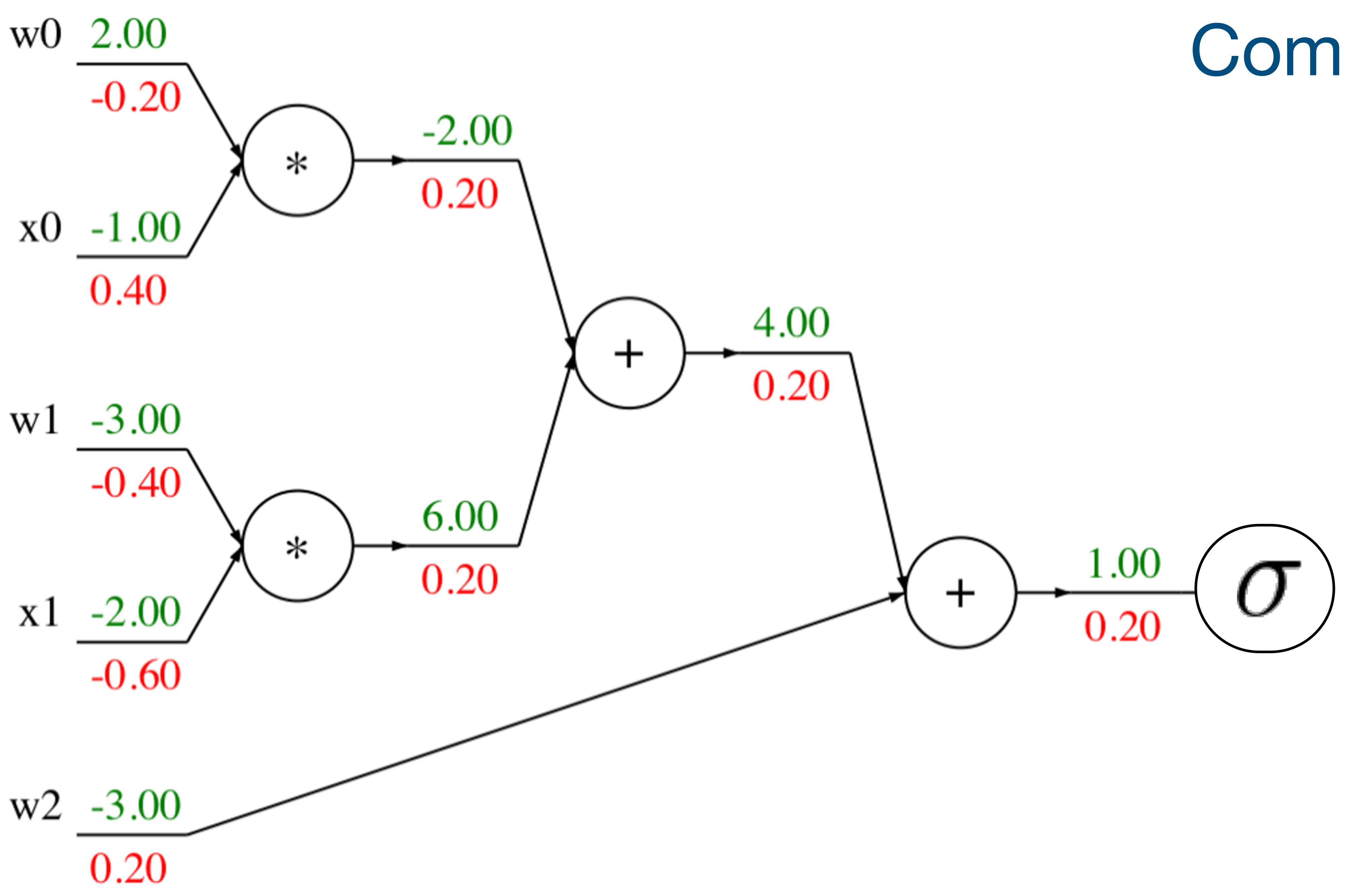
copy gate: gradient adder



max gate: gradient router



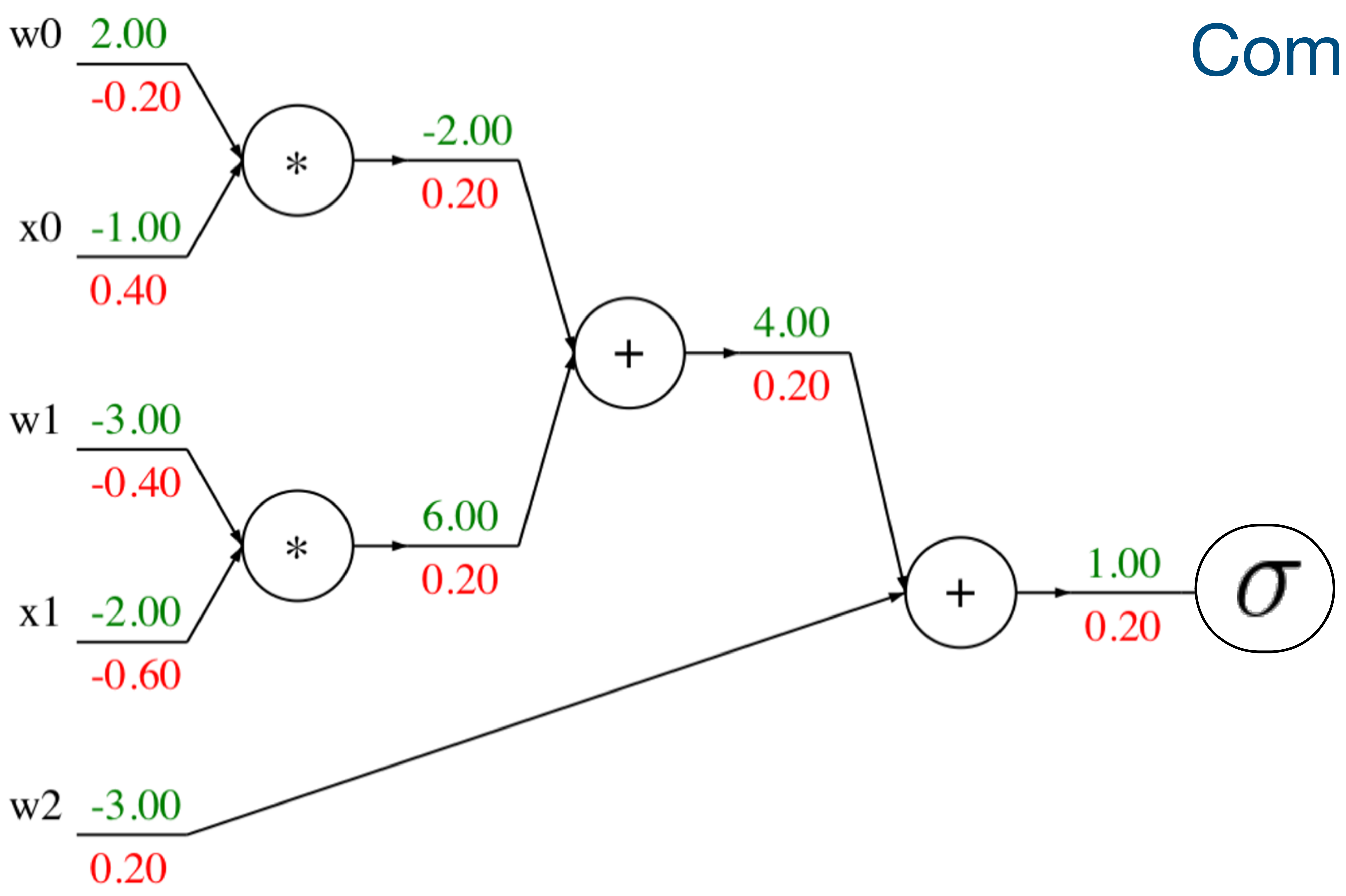
Backprop Implementation: "Flat" gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backprop Implementation: “Flat” gradient code



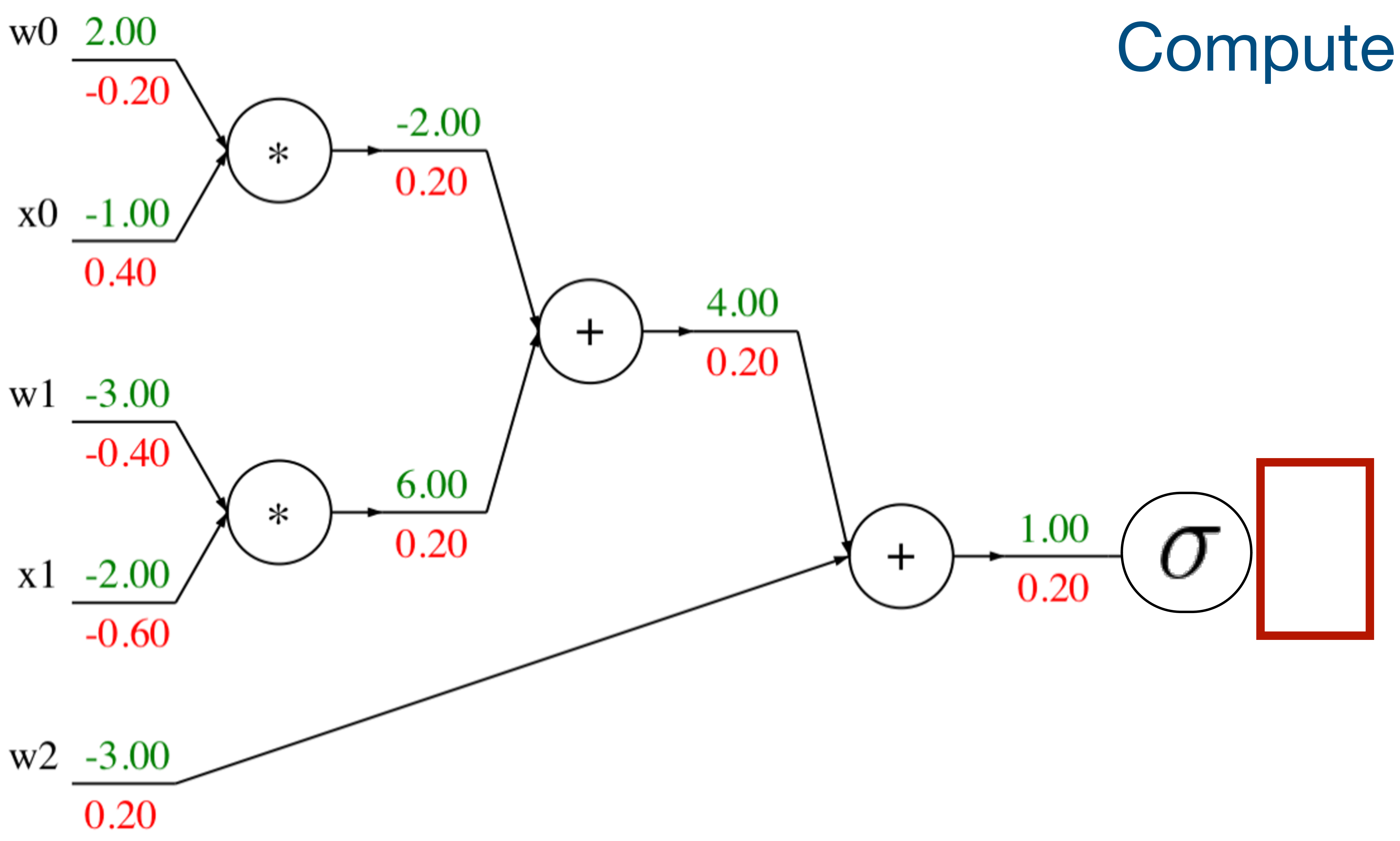
Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute gradients

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backprop Implementation: "Flat" gradient code



Forward pass:
Compute outputs

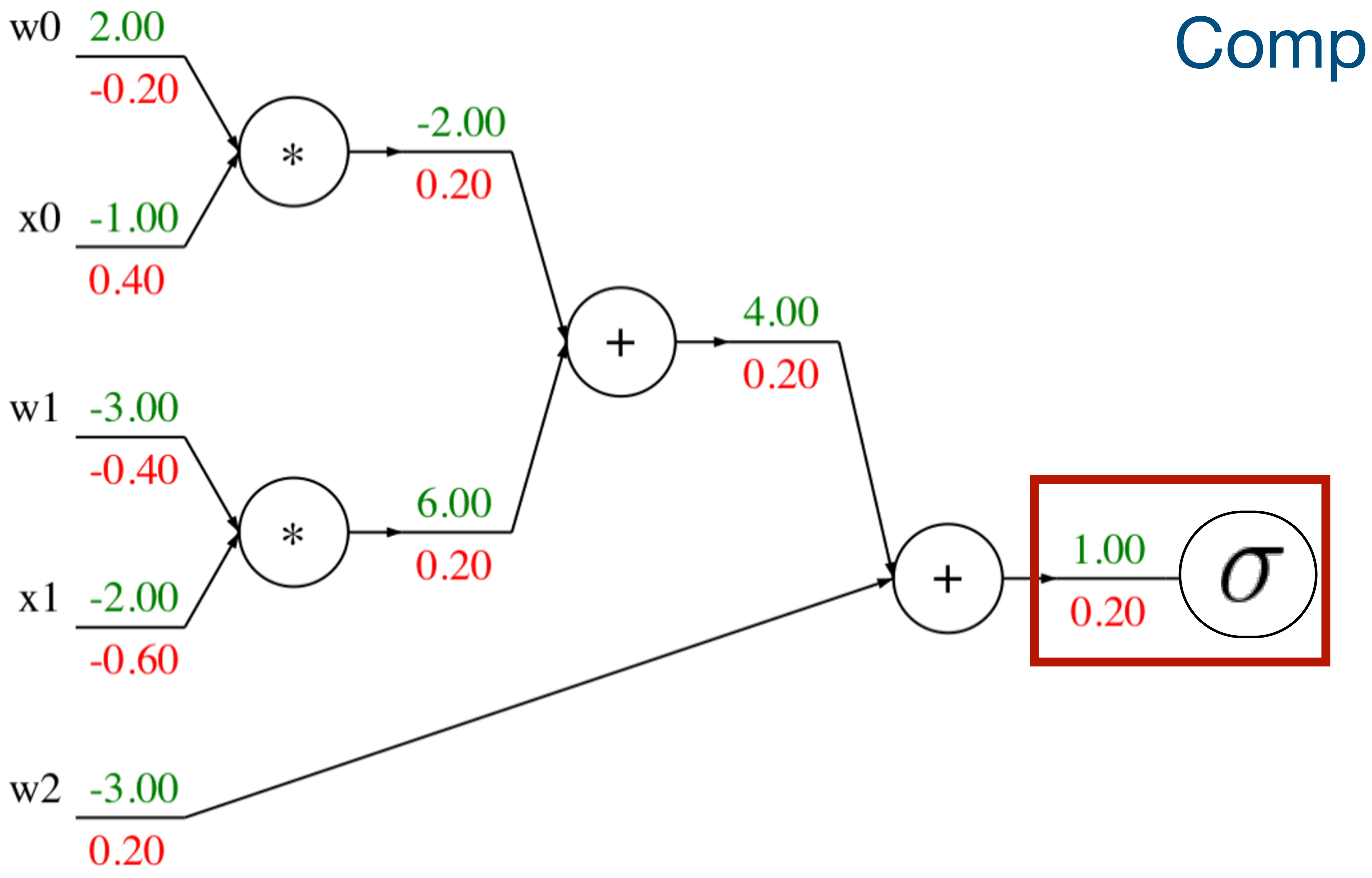
```python
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Base case

```python
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

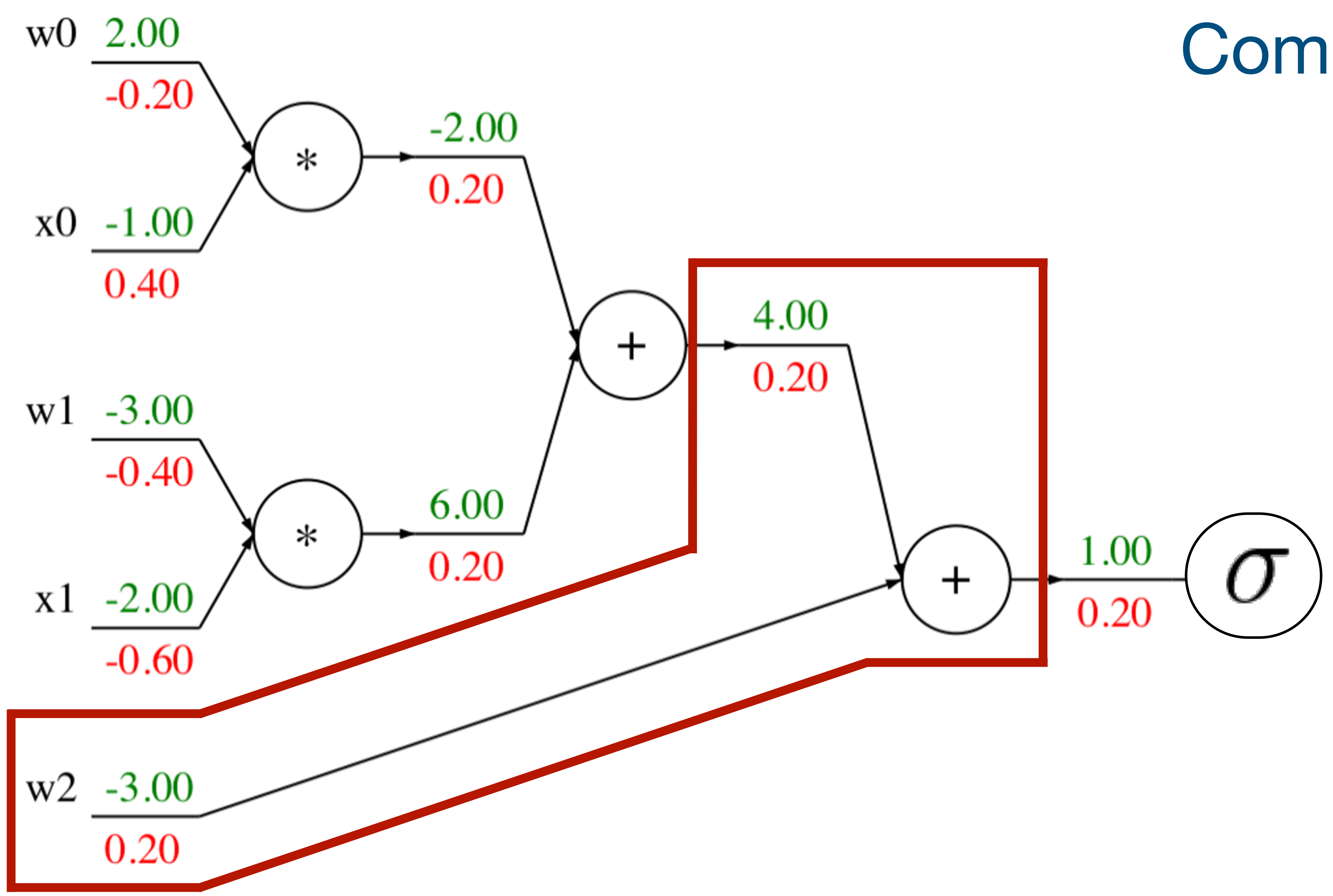
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Sigmoid

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: "Flat" gradient code



Forward pass:
Compute outputs

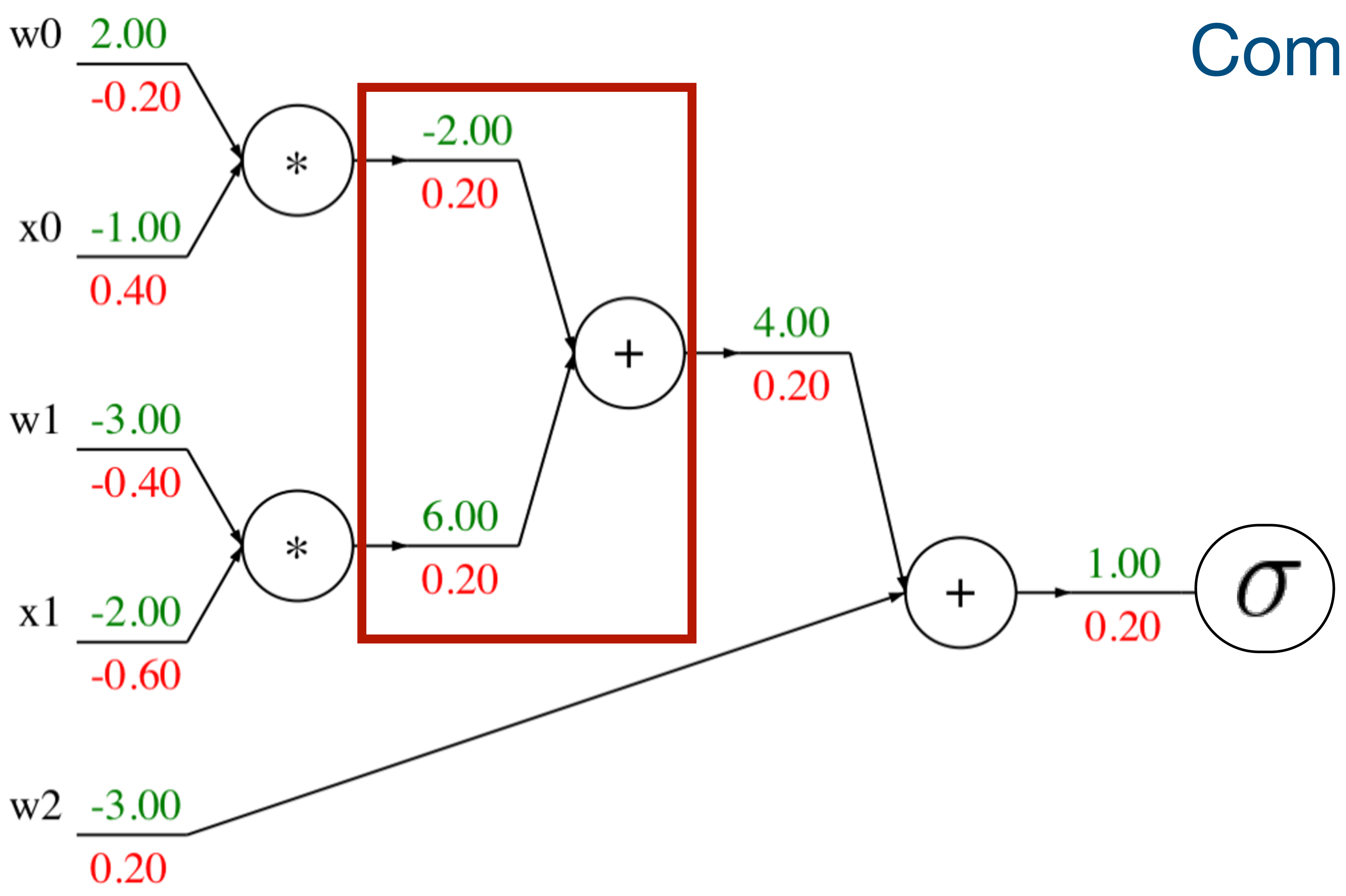
```python
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Add

```python
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: "Flat" gradient code



Forward pass:
Compute outputs

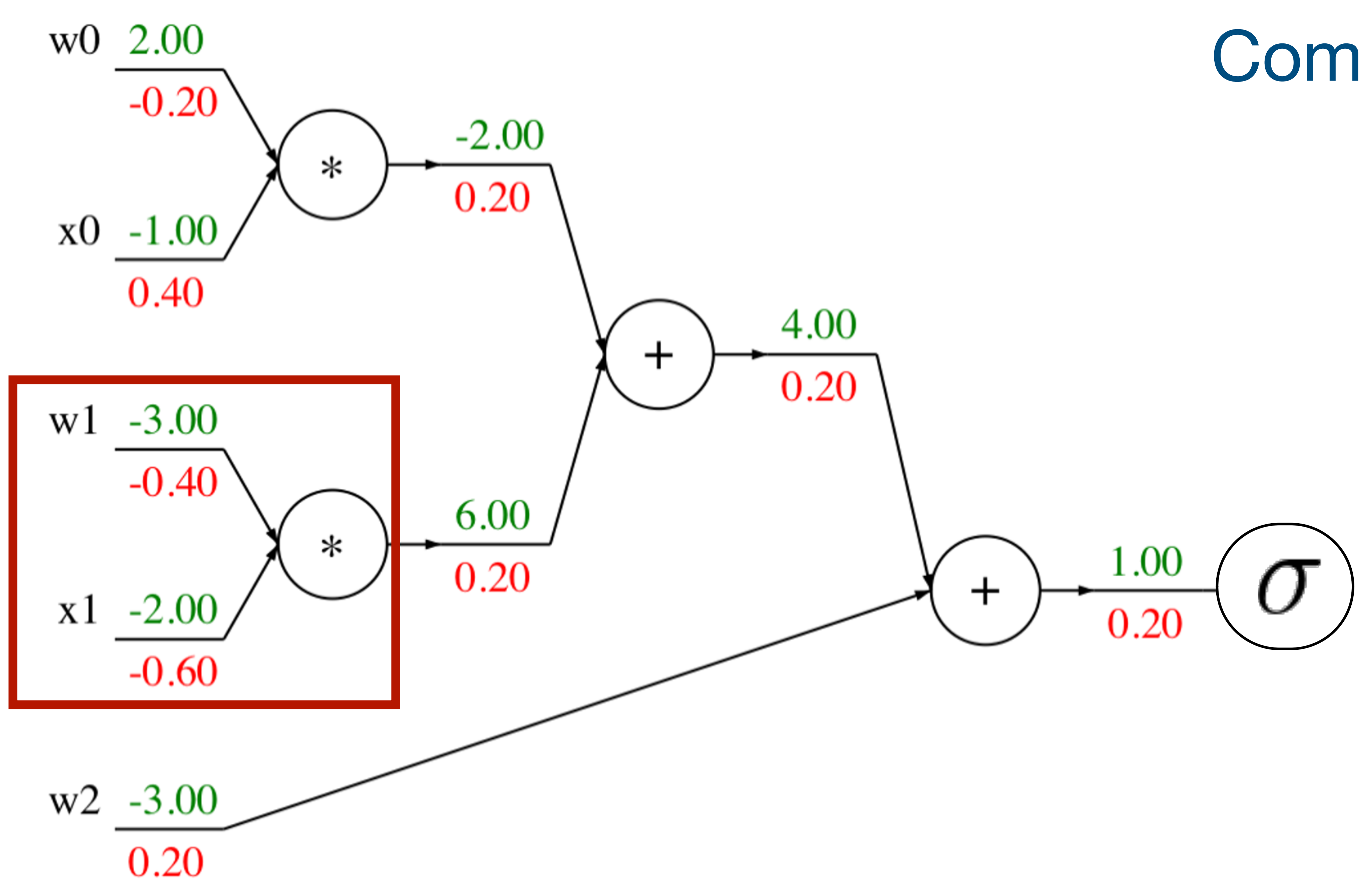
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Add

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

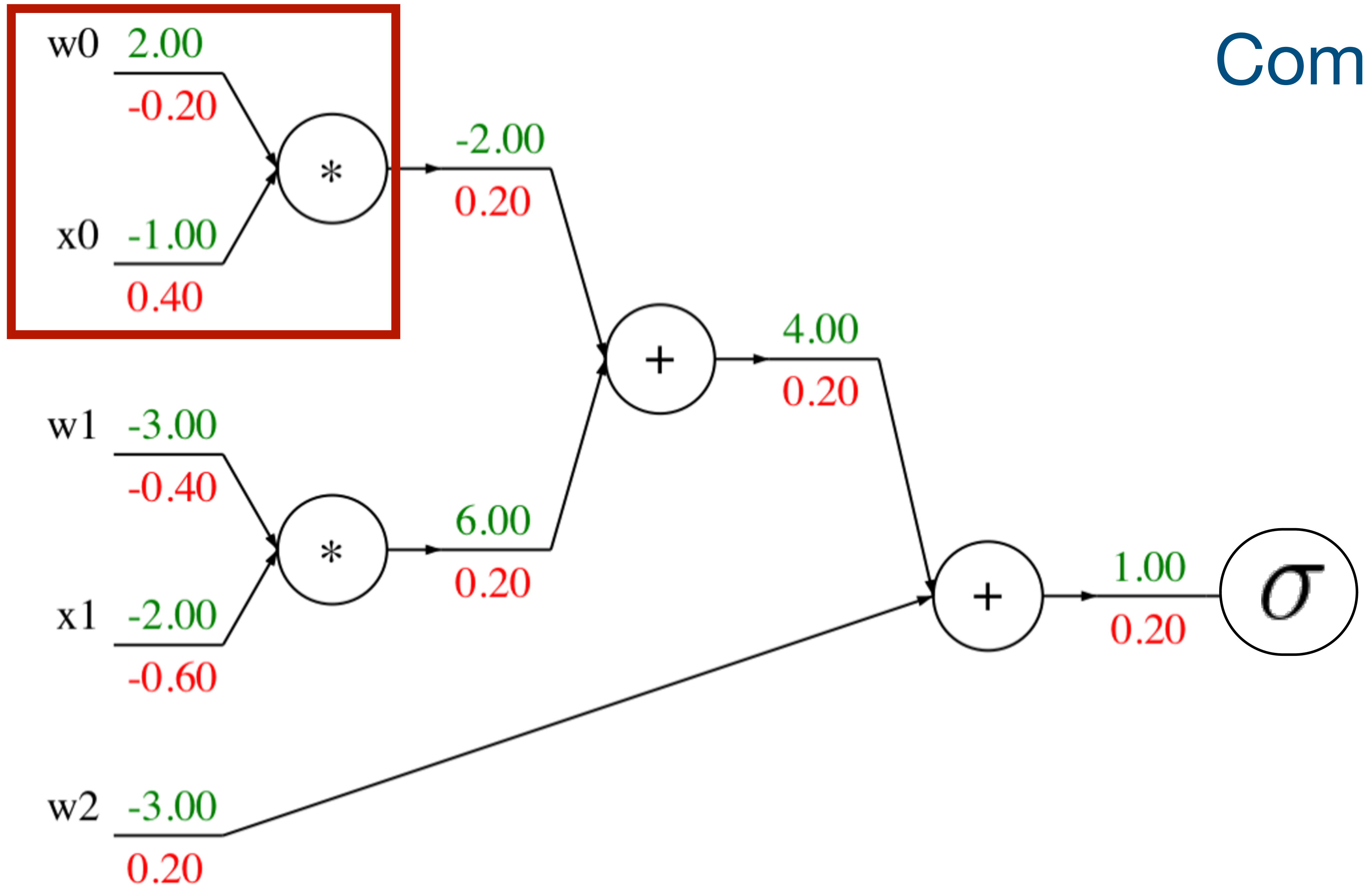
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

```
  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

Multiply

Backward pass:
Compute gradients

Backprop Implementation: "Flat" gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Multiply

Backward pass:
Compute gradients

"Flat" Backprop: Do this for Project 1 & 2

Forward pass:
Compute outputs

```
#########################################################################
# TODO:                                                                 #
# Implement a vectorized version of the structured SVM loss, storing the    #
# result in loss.                                                       #
#########################################################################
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#########################################################################
#                              END OF YOUR CODE                         #
#########################################################################
```

Backward pass:
Compute gradients

```
#########################################################################
# TODO:                                                                 #
# Implement a vectorized version of the gradient for the structured SVM     #
# loss, storing the result in dW.                                       #
#                                                                       #
# Hint: Instead of computing the gradient from scratch, it may be easier    #
# to reuse some of the intermediate values that you used to compute the     #
# loss.                                                                 #
#########################################################################
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#########################################################################
#                              END OF YOUR CODE                         #
#########################################################################
```

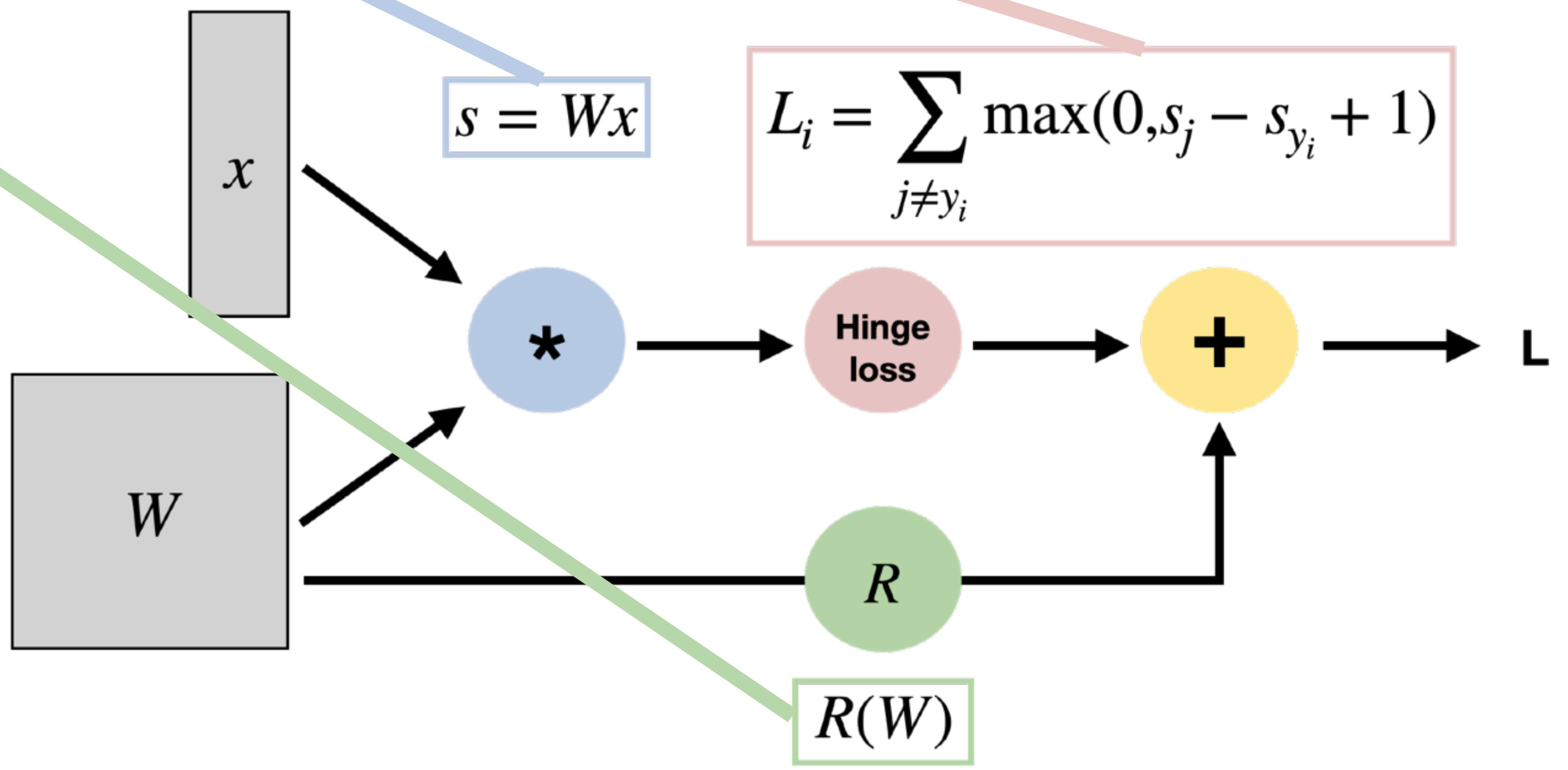
“Flat” Backprop: Do this for Project 1 & 2

Forward pass:
Compute outputs

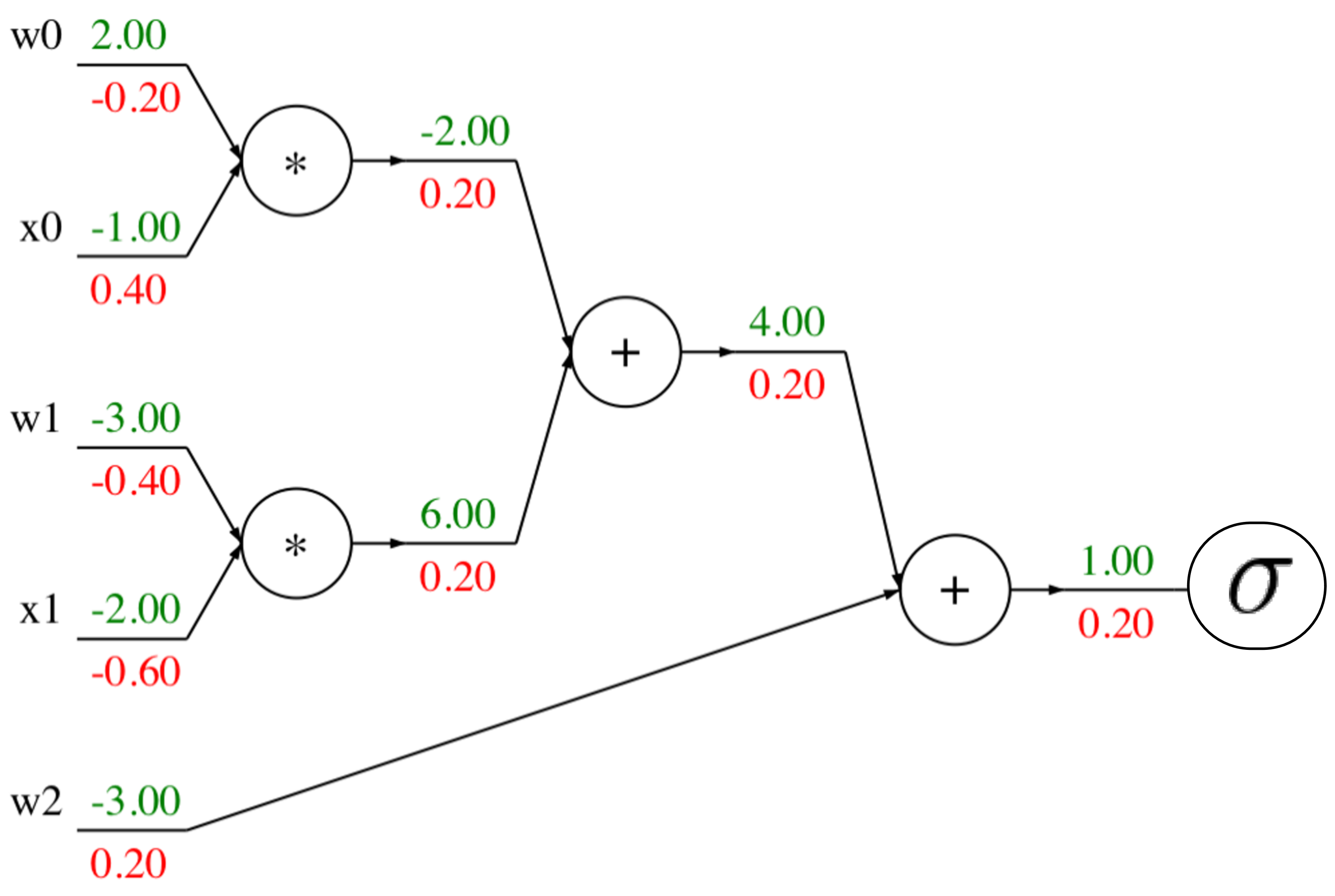
```
#############################################################################
# TODO:                                                                     #
# Implement a vectorized version of the structured SVM loss, storing the    #
# result in loss.                                                           #
#############################################################################
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
```

Backward pass:
Compute gradients

```
#############################################################################
# TODO:                                                                     #
# Implement a vectorized version of the gradient for the structured SVM     #
# loss, storing the result in dW.                                           #
#                                                                           #
# Hint: Instead of computing the gradient from scratch, it may be easier    #
# to reuse some of the intermediate values that you used to compute the     #
# loss.                                                                     #
#############################################################################
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#############################################################################
#                             END OF YOUR CODE                              #
#############################################################################
```



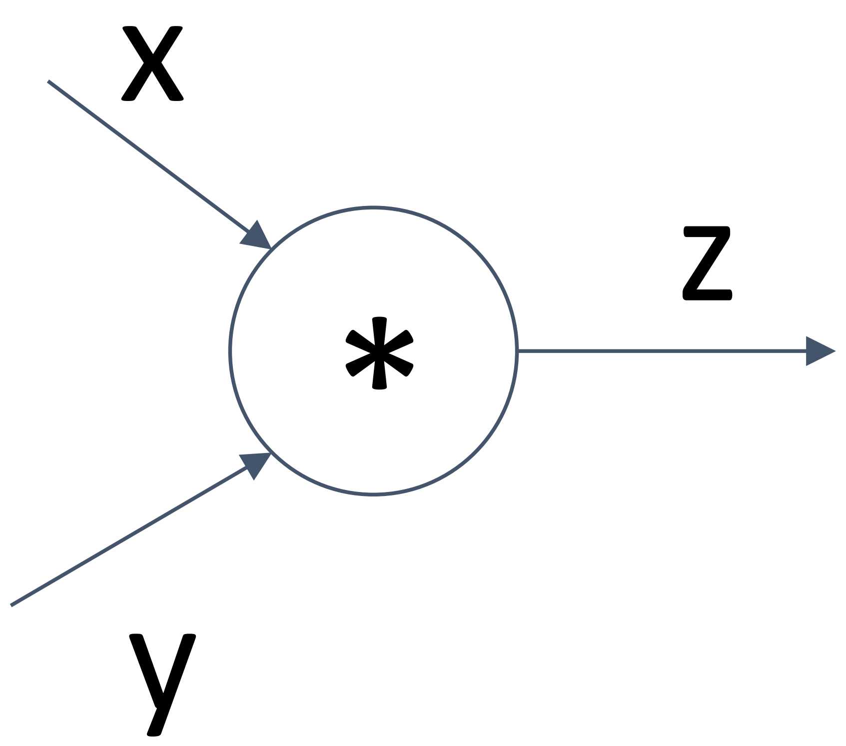
Backprop Implementation: Modular API



Graph (or Net) object *(rough pseudo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Example: PyTorch Autograd Functions



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

So far: backprop with scalars

What about vector-valued functions?

Recap: Vector Derivatives

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Recap: Vector Derivatives

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$x \in \mathbb{R}^N, y \in \mathbb{R}$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N, \left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will y change?

Recap: Vector Derivatives

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$x \in \mathbb{R}^N, y \in \mathbb{R}$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N, \quad \left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will y change?

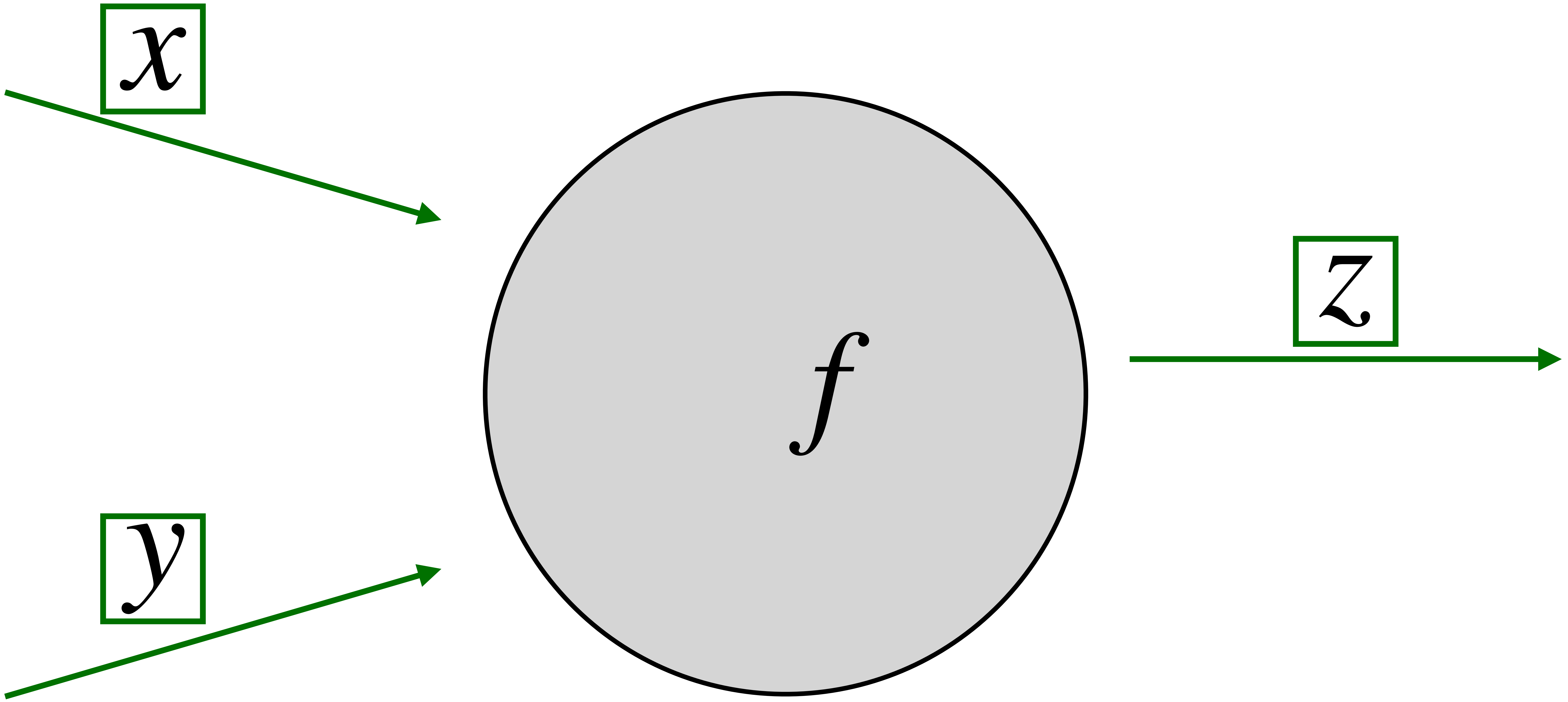
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

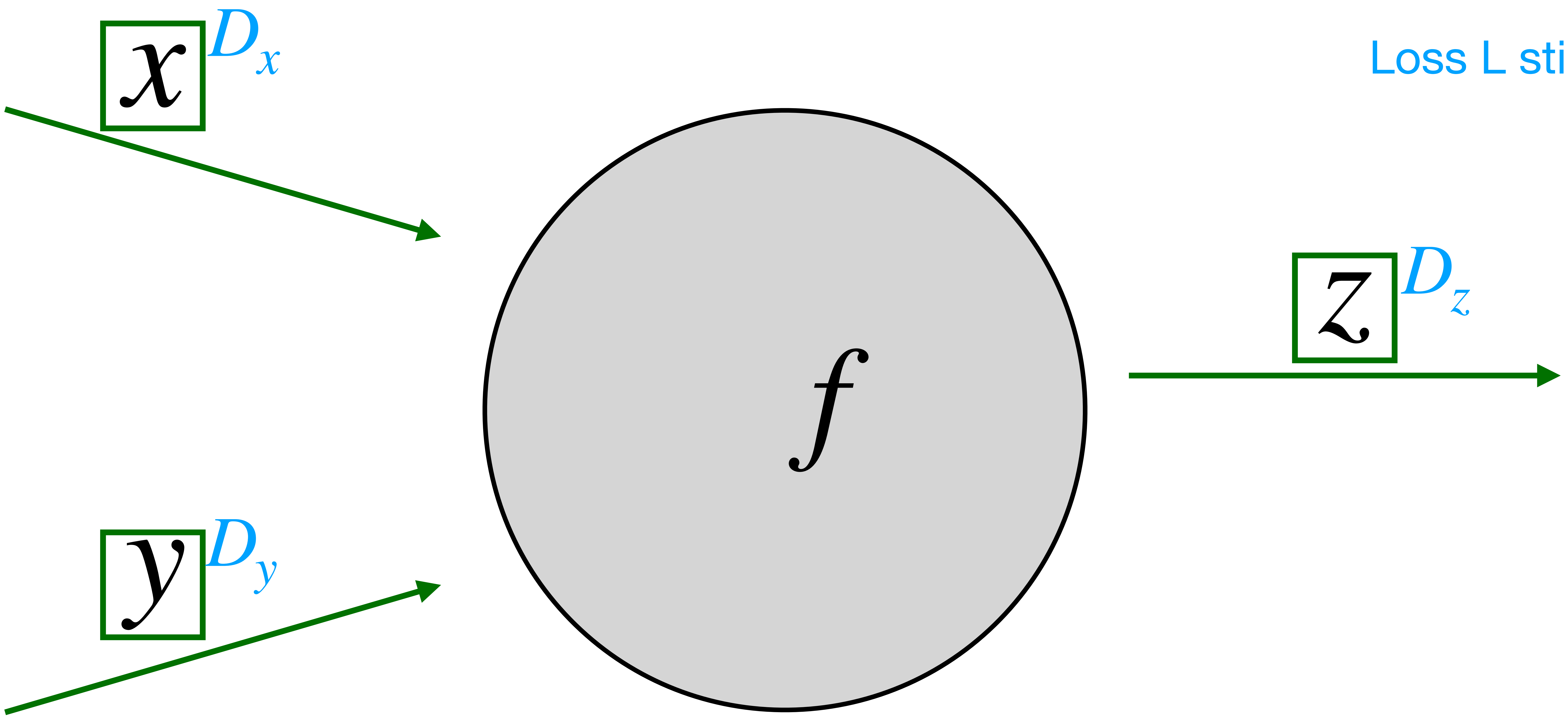
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

Backprop with Vectors

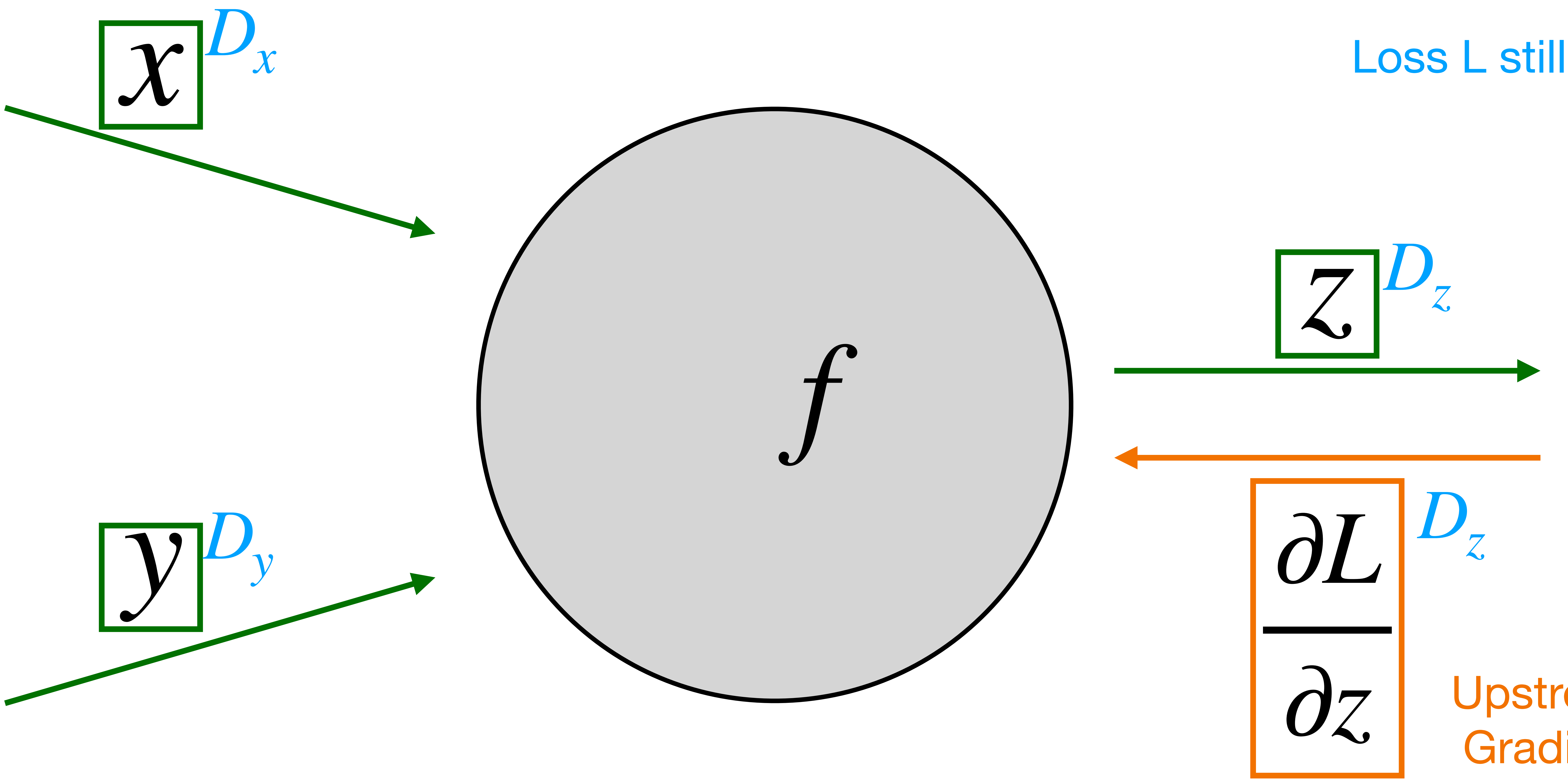


Backprop with Vectors



Loss L still a scalar!

Backprop with Vectors

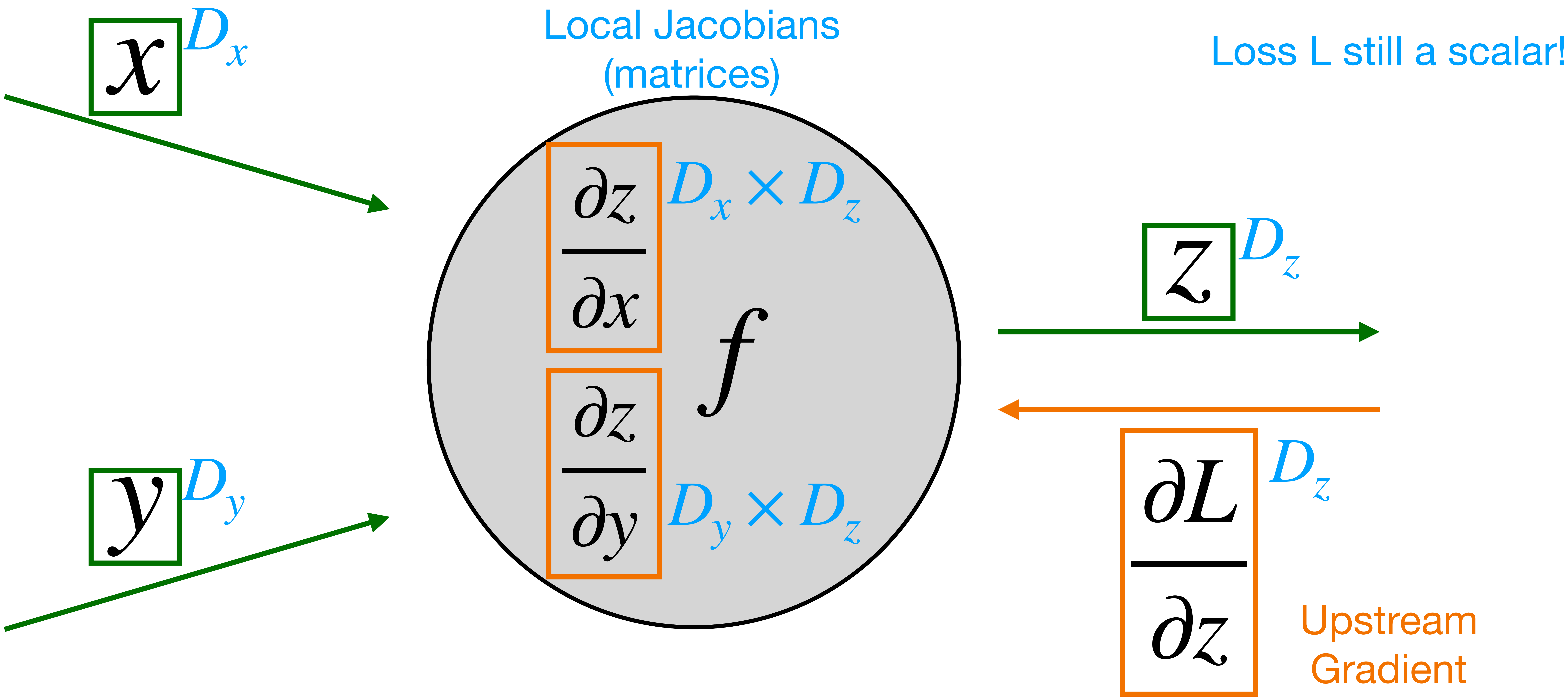


Loss L still a scalar!

Upstream Gradient

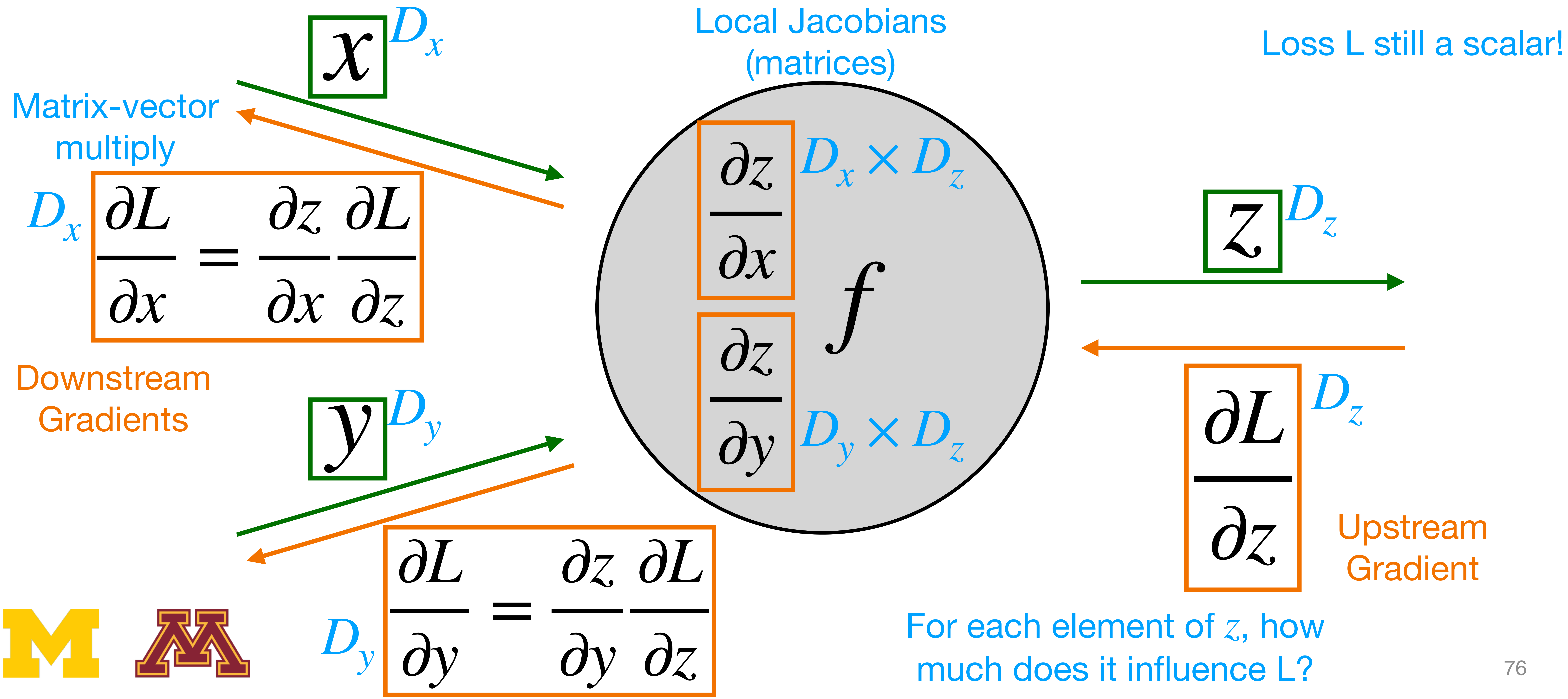
For each element of z, how much does it influence L?

Backprop with Vectors



For each element of z, how much does it influence L?

Backprop with Vectors



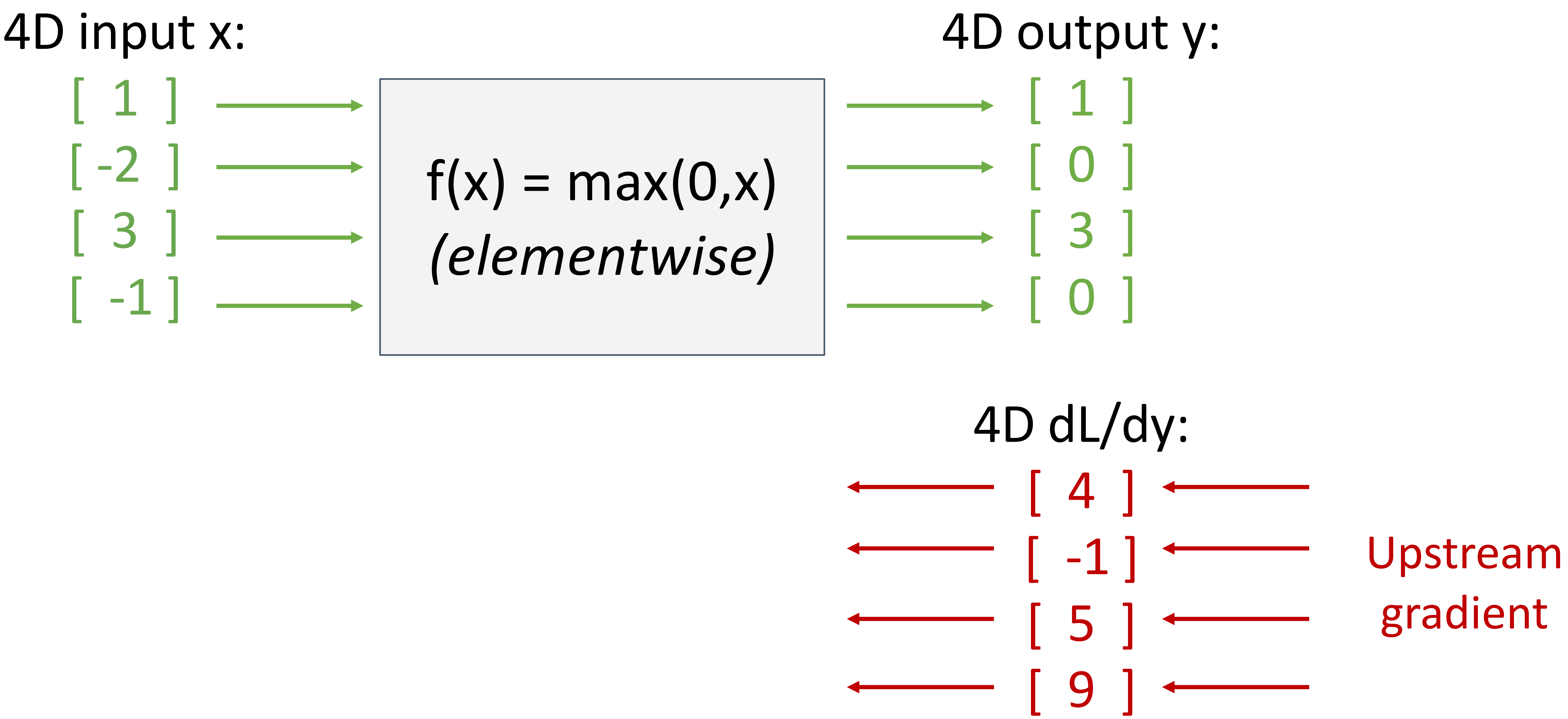
Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow \boxed{\begin{matrix} f(x) = \max(0,x) \\ \textit{(elementwise)} \end{matrix}} \longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D output y:

Backprop with Vectors



Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output y:

[1]
[0]
[3]
[0]

[dy/dx] [dL/dy]

[1 0 0 0] [4]
[0 0 0 0] [-1]
[0 0 1 0] [5]
[0 0 0 0] [9]

4D dL/dy:

[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output y:

[1]
[0]
[3]
[0]

4D dL/dx:

[4]
[0]
[5]
[0]

[dy/dx] [dL/dy]

[1 0 0 0] [4]
[0 0 0 0] [-1]
[0 0 1 0] [5]
[0 0 0 0] [9]

4D dL/dy:

[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix}1\\-2\\3\\-1\end{bmatrix}$$

f(x) = max(0,x)
(elementwise)

4D output y:

$$\begin{bmatrix}1\\0\\3\\0\end{bmatrix}$$

Jacobian is **sparse**: off-diagonal entries all zero! Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx:

$$\begin{bmatrix}4\\0\\5\\0\end{bmatrix}$$

[dy/dx] [dL/dy]

$$\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}4\\-1\\5\\9\end{bmatrix}$$

4D dL/dy:

$$\begin{bmatrix}4\\-1\\5\\9\end{bmatrix}$$

Upstream gradient

Backprop with Vectors

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output y:

[1]
[0]
[3]
[0]

Jacobian is **sparse**: off-diagonal entries all zero! Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx:

[4]
[0]
[5]
[0]

[dy/dx] [dL/dy]

$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial y}\right)_i, & if\ x_i > 0 \\ 0, & otherwise \end{cases}$$

4D dL/dy:

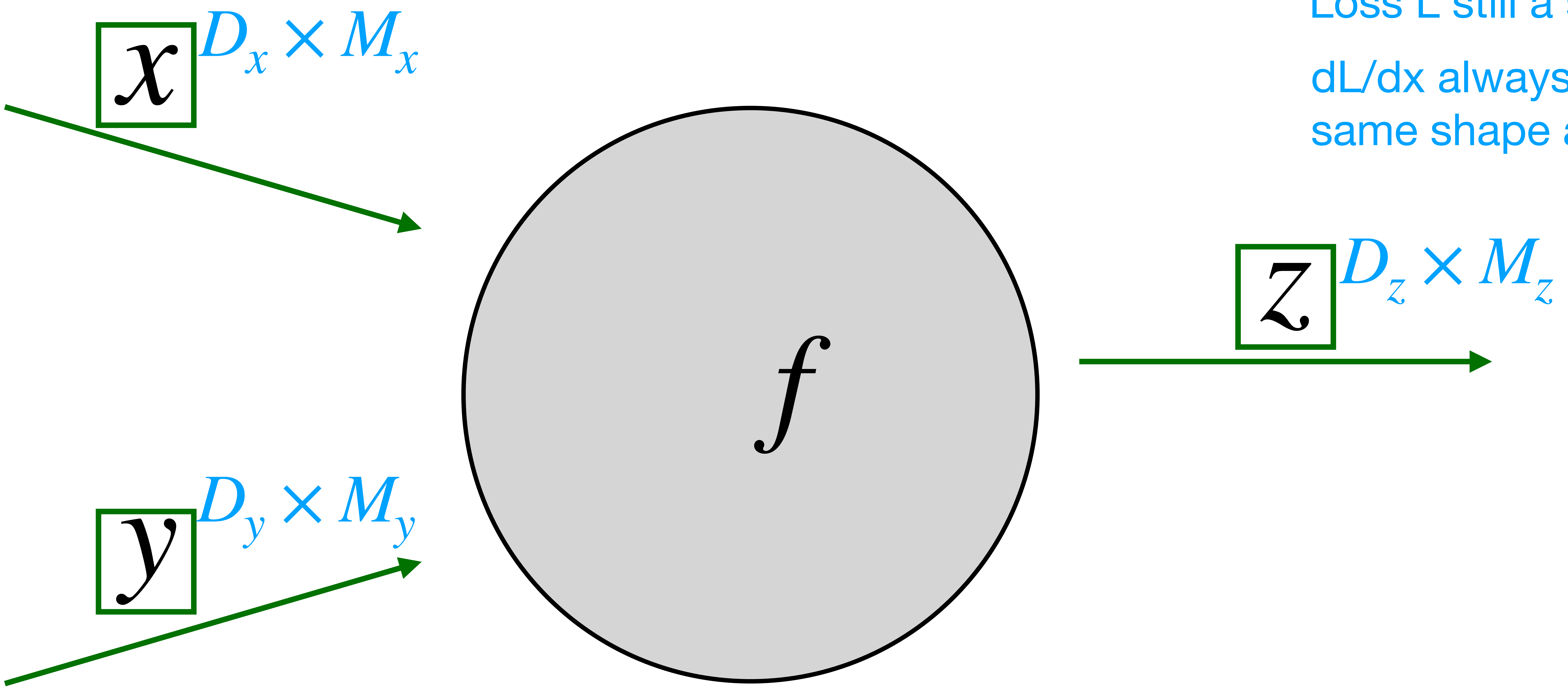
[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Matrices (or Tensors)

Loss L still a scalar!

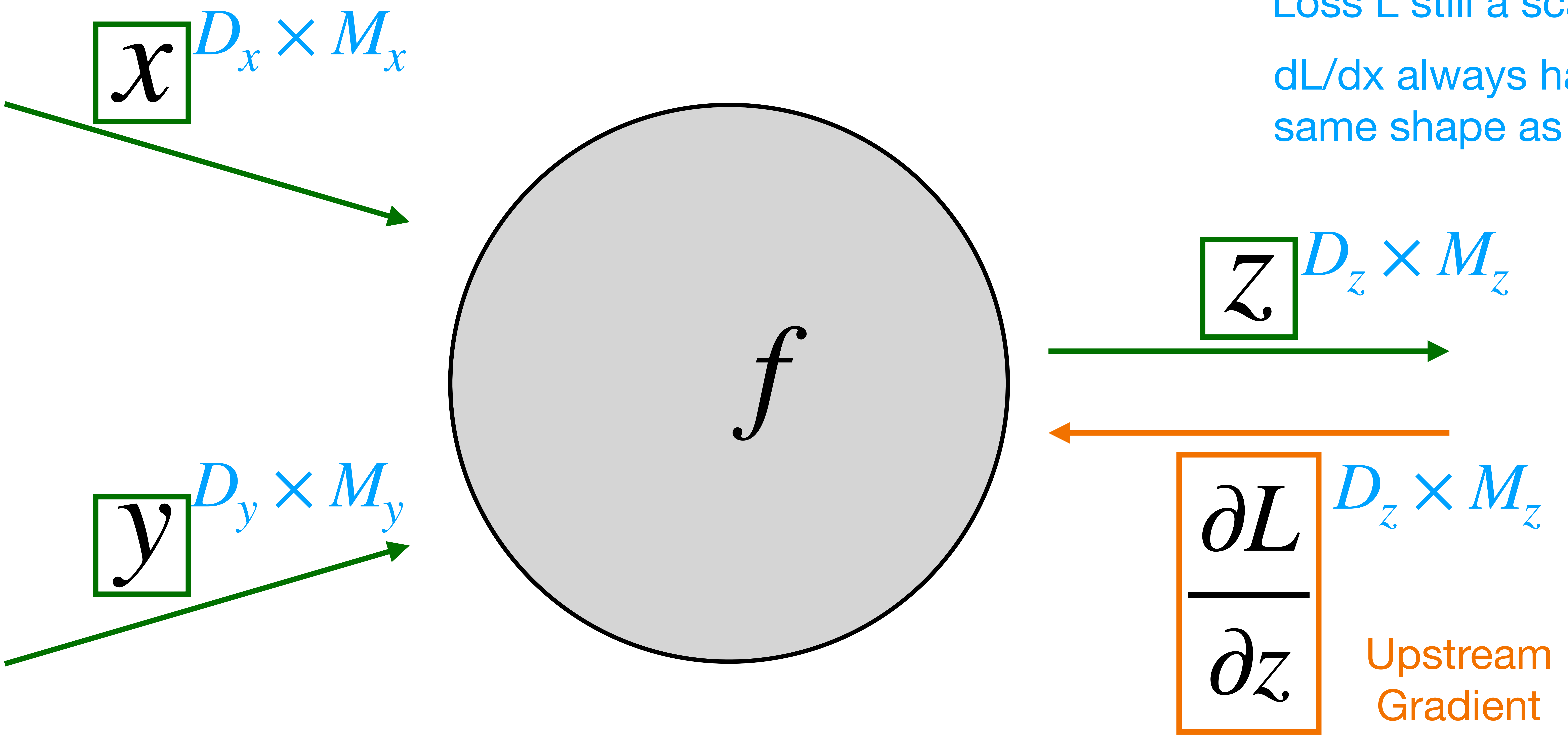
dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!

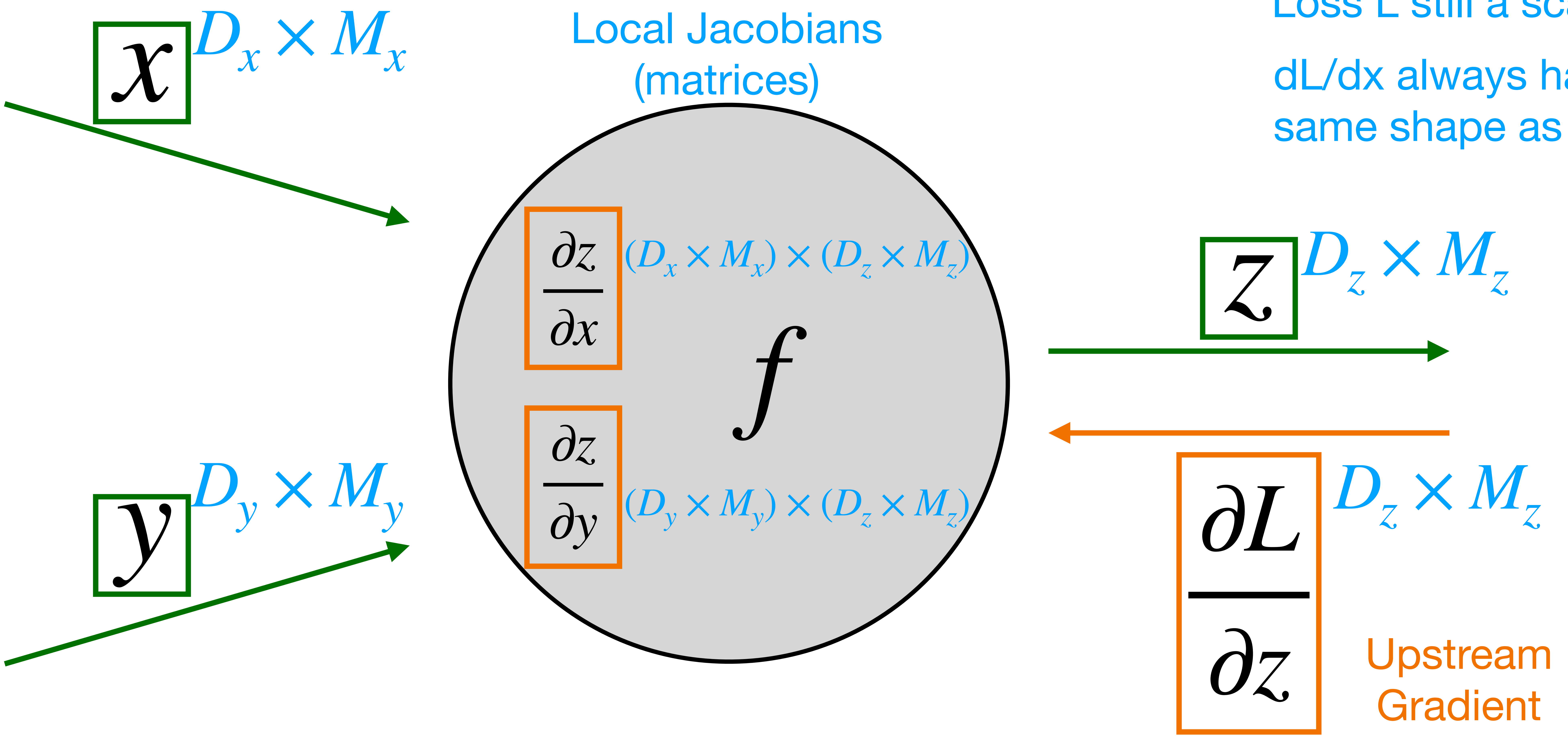


For each element of z, how much does it influence L?

Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!



For each element of z, how much does it influence L?

Backprop with Matrices (or Tensors)

x $D_x \times M_x$

Local Jacobians (matrices)

Loss L still a scalar!

dL/dx always has the same shape as x!

Matrix-vector multiply

$$\boxed{\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}}$$

$D_x \times M_x$

Downstream Gradients

$\boxed{\frac{\partial z}{\partial x}}$ $(D_x \times M_x) \times (D_z \times M_z)$

f

$\boxed{\frac{\partial z}{\partial y}}$ $(D_y \times M_y) \times (D_z \times M_z)$

z $D_z \times M_z$

$\boxed{\frac{\partial L}{\partial z}}$ $D_z \times M_z$

Upstream Gradient

y $D_y \times M_y$

$D_y \times M_y$

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

For each element of z, how much does it influence L?

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

→

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

→

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply *y = xw*

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

y: [N×M]

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Jacobians:
dy/dx: [(N×D)×(N×M)]
dy/dw: [(D×M)×(N×M)]

For a neural net we may have
N=64, D=M=4096
Each Jacobian takes 256 GB of memory! Must work with them implicitly!

Example: Matrix Multiplication

x: [N×D]
$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$

w: [D×M]
$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

y: [N×M]
$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$

dL/dy: [N×M]
$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$

dL/dx: [N×D]
$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$

Local Gradient Slice:

$dy/dx_{1,1}$
$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$
$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} \boxed{-1} & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,1}/dx_{1,1}$ $\begin{bmatrix} \boxed{?} & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} \boxed{-1} & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,1}/dx_{1,1}$ $\begin{bmatrix} \boxed{?} & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

$y_{1,1} = x_{1,1}w_{1,1} + x_{1,2}w_{2,1} + x_{1,3}w_{3,1}$

Example: Matrix Multiplication

x: [N×D]

[2 1 -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,1}/dx_{1,1}$ [3 ? ? ?]
[? ? ? ?]

$dL/dx_{1,1}$
$= (dy/dx_{1,1}) \cdot (dL/dy)$

$y_{1,1} = x_{1,1}\boxed{w_{1,1}} + x_{1,2}w_{2,1} + x_{1,3}w_{3,1}$
$\Rightarrow dy_{1,1}/dx_{1,1} = \boxed{w_{1,1}}$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & \boxed{-1} & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ $\begin{bmatrix} 3 & \boxed{?} & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & \boxed{-1} & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ $\begin{bmatrix} 3 & \boxed{?} & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

$y_{1,2} = x_{1,1}w_{1,2} + x_{1,2}w_{2,2} + x_{1,3}w_{3,2}$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ $\begin{bmatrix} 3 & 2 & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

$y_{1,2} = x_{1,1}\boxed{w_{1,2}} + x_{1,2}w_{2,2} + x_{1,3}w_{3,2}$

$\Rightarrow dy_{1,2}/dx_{1,1} = w_{1,2}$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & \boxed{1 & -1} \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$

$$\begin{bmatrix} 3 & 2 & \boxed{1 & -1} \\ ? & ? & ? & ? \end{bmatrix}$$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ \boxed{5} & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ \boxed{?} & ? & ? & ? \end{bmatrix}$$

$y_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ \boxed{5} & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{cc} & \\ dy_{1,2}/dx_{1,1} & \begin{bmatrix} 3 & 2 & 1 & -1 \\ \boxed{0} & ? & ? & ? \end{bmatrix} \end{array}$$

$y_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$

$\Rightarrow dy_{2,1}/dx_{1,1} = 0$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} \boxed{2} & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} \boxed{?} & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{1,1}$

$$\begin{array}{rc} dy_{1,2}/dx_{1,1} & \begin{bmatrix} 3 & 2 & 1 & -1 \\ 0 & \boxed{0 \quad 0 \quad 0} \end{bmatrix} \end{array}$$

$dL/dx_{1,1}$

$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

[$\boxed{2}$ 1 -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]

[$\boxed{?}$? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

[3 2 1 -1]
[0 0 0 0]

$dL/dx_{1,1}$
$= (dy/dx_{1,1}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[0 ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$
[3 2 1 -1]
[0 0 0 0]

$dL/dx_{1,1}$
$= (dy/dx_{1,1}) \cdot (dL/dy)$
$= (w_{1,:}) \cdot (dL/dy_{1,:})$
$= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0$

Example: Matrix Multiplication

x: [N×D]

[2 1 -3]
[-3 4 $\boxed{2}$]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[$\boxed{3\ 2\ 1\ -2}$]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]

[0 ? ?]
[? ? $\boxed{-30}$]

Local Gradient Slice:

$dy/dx_{2,3}$

[0 0 0 0]
[$\boxed{3\ 2\ 1\ -2}$]

$dL/dx_{2,3}$

$= (dy/dx_{2,3}) \cdot (dL/dy)$

Example: Matrix Multiplication

x: [N×D]

$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & \boxed{2} \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

dL/dx: [N×D]

$$\begin{bmatrix} 0 & ? & ? \\ ? & ? & \boxed{-30} \end{bmatrix}$$

Local Gradient Slice:

$dy/dx_{2,3}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

$dL/dx_{2,3}$

$= (dy/dx_{2,3}) \cdot (dL/dy)$

$= (w_{3,:}) \cdot (dL/dy_{2,:})$

$= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30$

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[0 16 -9]
[-24 9 -30]

$dL/dx_{i,j}$
$= (dy/dx_{i,j}) \cdot (dL/dy)$
$= (w_{j,:}) \cdot (dL/dy_{i,:})$

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[0 16 -9]
[-24 9 -30]

dL/dx = (dL/dy) w^T

[N x D] [N x M] [M x D]

$dL/dx_{i,j}$
$= (dy/dx_{i,j}) \cdot (dL/dy)$
$= (w_{j,:}) \cdot (dL/dy_{i,:})$

Easy way to remember: It's the only way the shapes work out!

Example: Matrix Multiplication

x: [N×D]

[2 1 -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 3 2]
[3 2 1 -2]

Matrix Multiply *y = xw*

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]

[0 16 -9]
[-24 9 -30]

dL/dx = (dL/dy) w^T
[N x D] [N x M] [M x D]

dL/dw = x^T (dL/dy)
[D x M] [D x N] [N x M]

Easy way to remember: It's the only way the shapes work out!

Backpropagation: Another View

$$\underset{D_0}{x_0} \xrightarrow{\quad f_1 \quad} \underset{D_1}{x_1} \xrightarrow{\quad f_2 \quad} \underset{D_2}{x_2} \xrightarrow{\quad f_3 \quad} \underset{D_3}{x_3} \xrightarrow{\quad f_4 \quad} \underset{\text{scalar}}{L}$$

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

Backpropagation: Another View

$$\underset{D_0}{x_0} \xrightarrow{f_1} \underset{D_1}{x_1} \xrightarrow{f_2} \underset{D_2}{x_2} \xrightarrow{f_3} \underset{D_3}{x_3} \xrightarrow{f_4} \underset{\text{scalar}}{L}$$

Matrix multiplication is **associative**: we can compute products in any order

Chain rule

$$\frac{\partial L}{\partial x_0} = \underset{[D_0 \times D_1]}{\left(\frac{\partial x_1}{\partial x_0}\right)} \underset{[D_1 \times D_2]}{\left(\frac{\partial x_2}{\partial x_1}\right)} \underset{[D_2 \times D_3]}{\left(\frac{\partial x_3}{\partial x_2}\right)} \underset{[D_3]}{\left(\frac{\partial L}{\partial x_3}\right)}$$

Reverse-Mode Automatic Differentiation

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} x_2 \xrightarrow{f_3} x_3 \xrightarrow{f_4} L$$

D_0 D_1 D_2 D_3 scalar

Matrix multiplication is **associative**: we can compute products in any order
Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

[D_0 x D_1] [D_1 x D_2] [D_2 x D_3] [D_3]

Reverse-Mode Automatic Differentiation

$$\begin{array}{ccccccccc} x_0 & \xrightarrow{f_1} & x_1 & \xrightarrow{f_2} & x_2 & \xrightarrow{f_3} & x_3 & \xrightarrow{f_4} & L \\ D_0 & & D_1 & & D_2 & & D_3 & & \text{scalar} \end{array}$$

Matrix multiplication is **associative**: we can compute products in any order
Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

$[D_0 \times D_1]$ $[D_1 \times D_2]$ $[D_2 \times D_3]$ $[D_3]$

Compute grad of scalar <u>output</u> w/respect to all vector <u>inputs</u>

What if we want grads of scalar <u>input</u> w/respect to vector <u>outputs</u>?

Forward-Mode Automatic Differentiation

$$\underset{\text{scalar}}{a} \xrightarrow{f_1} \underset{D_0}{x_0} \xrightarrow{f_2} \underset{D_1}{x_1} \xrightarrow{f_3} \underset{D_2}{x_2} \xrightarrow{f_4} \underset{D_3}{x_3}$$

Chain rule

$$\frac{\partial x_3}{\partial a} = \underset{[D_0]}{\left(\frac{\partial x_0}{\partial a}\right)} \underset{[D_0 \times D_1]}{\left(\frac{\partial x_1}{\partial x_0}\right)} \underset{[D_1 \times D_2]}{\left(\frac{\partial x_2}{\partial x_1}\right)} \underset{[D_2 \times D_3]}{\left(\frac{\partial x_3}{\partial x_2}\right)}$$

Forward-Mode Automatic Differentiation

$$\underset{\text{scalar}}{a} \xrightarrow{f_1} \underset{D_0}{x_0} \xrightarrow{f_2} \underset{D_1}{x_1} \xrightarrow{f_3} \underset{D_2}{x_2} \xrightarrow{f_4} \underset{D_3}{x_3}$$

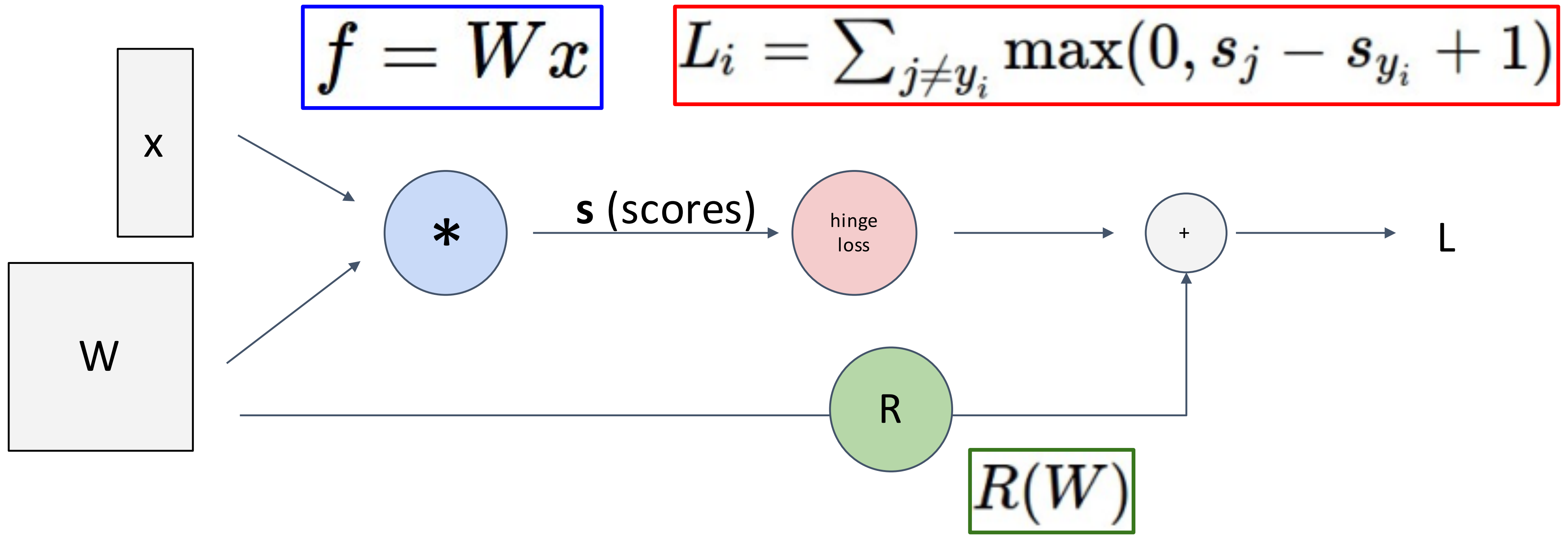
Computing products <u>left-to-right</u> avoids matrix-matrix products; only needs matrix-vector

Chain rule

$$\frac{\partial x_3}{\partial a} = \overrightarrow{\underset{[D_0]}{\left(\frac{\partial x_0}{\partial a}\right)} \underset{[D_0 \times D_1]}{\left(\frac{\partial x_1}{\partial x_0}\right)} \underset{[D_1 \times D_2]}{\left(\frac{\partial x_2}{\partial x_1}\right)} \underset{[D_2 \times D_3]}{\left(\frac{\partial x_3}{\partial x_2}\right)}}$$

Summary

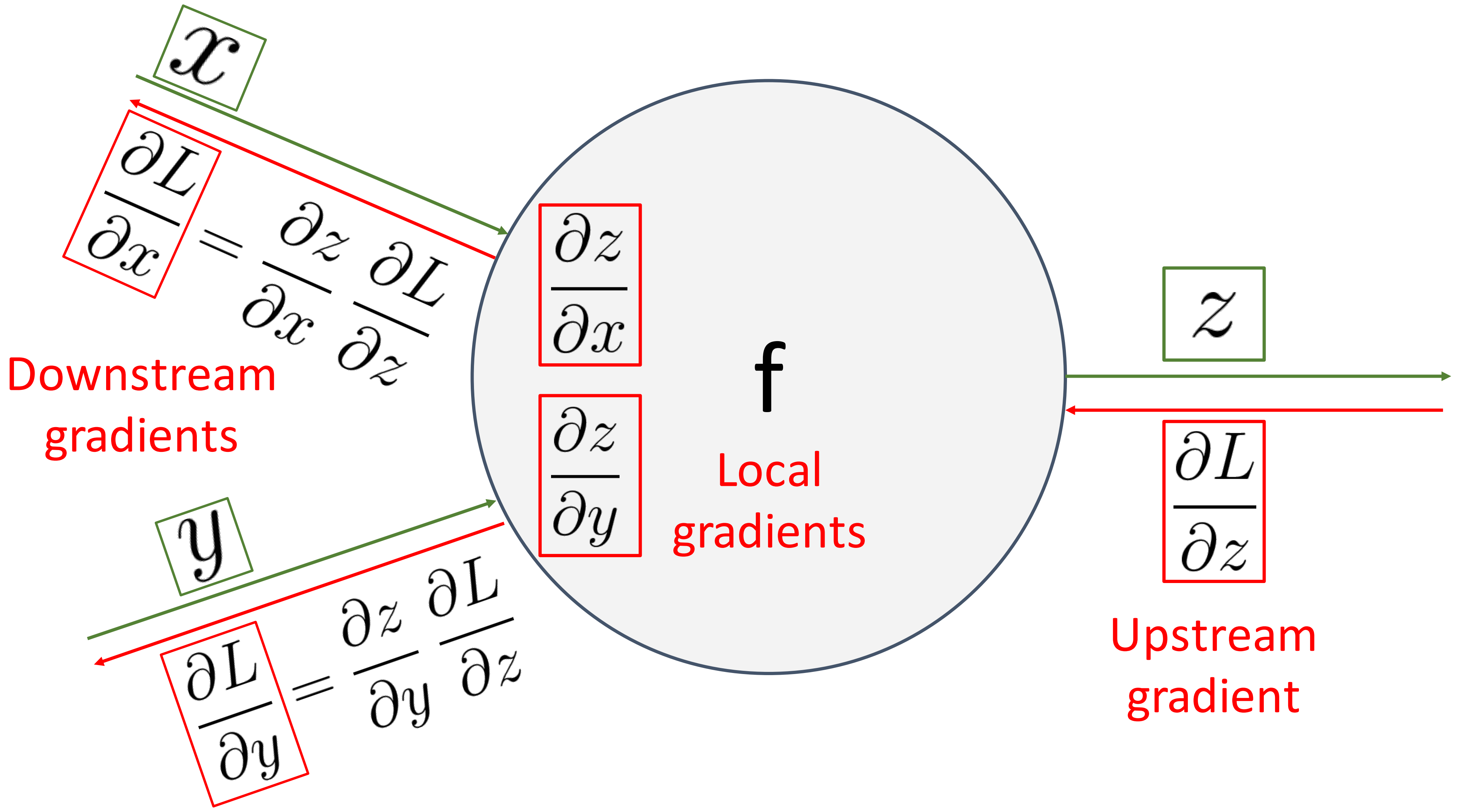
Represent complex expressions as **computational graphs**



Forward pass computes outputs

Backward pass computes gradients

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**



Summary

Backprop can be implemented with “flat” code where the backward pass looks like forward pass reversed (Use this for A2!)

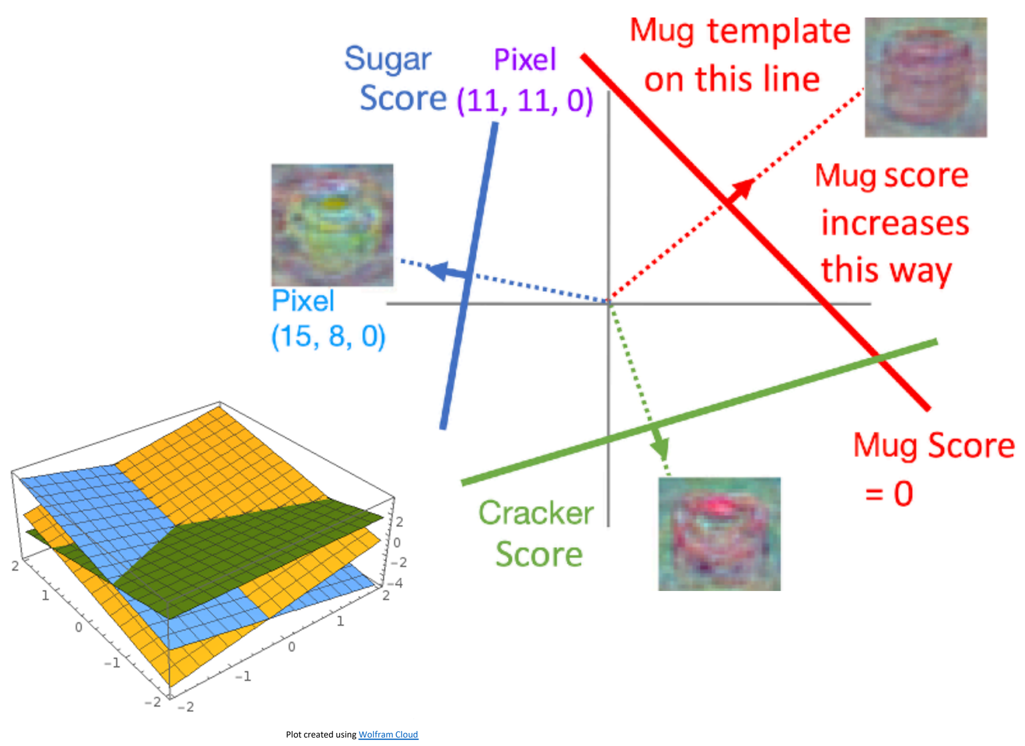
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

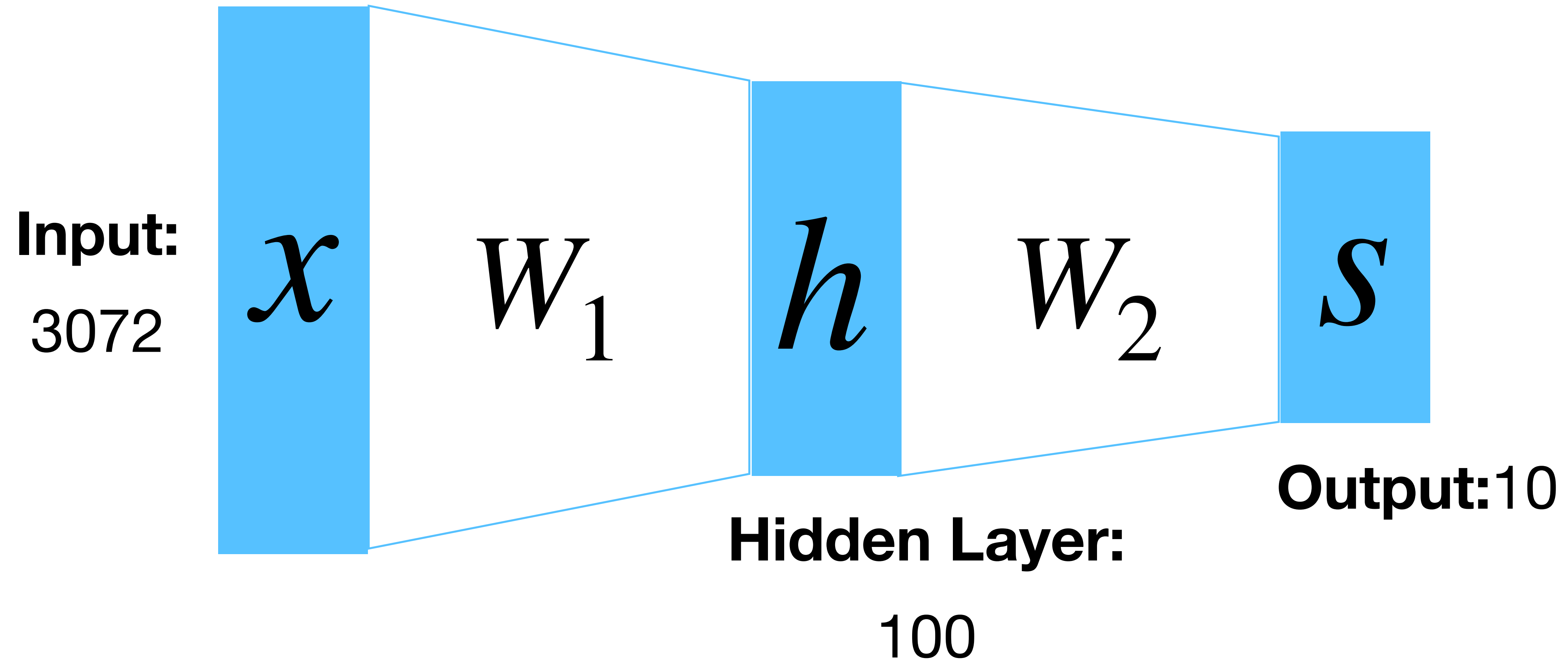
Backprop can be implemented with a modular API, as a set of paired forward/backward functions

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

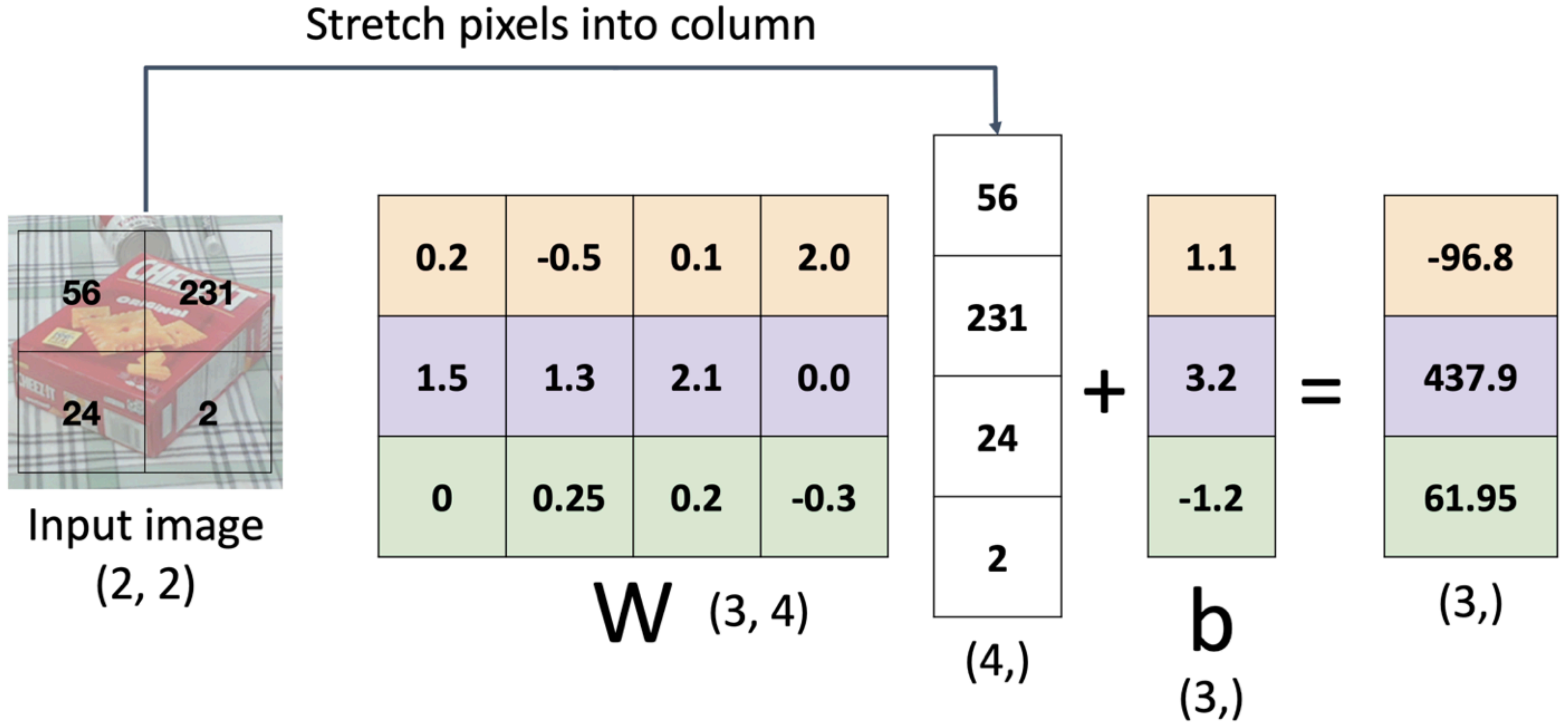
Summary



$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



Problem: So far our classifiers don't respect the spatial structure of images!



Next time: Convolutional Neural Networks

DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota