

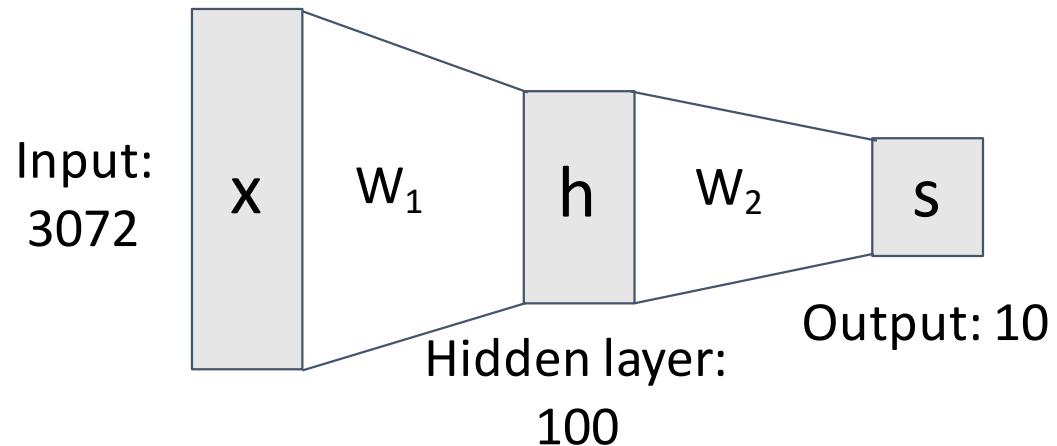
# Lecture 4: Backpropagation



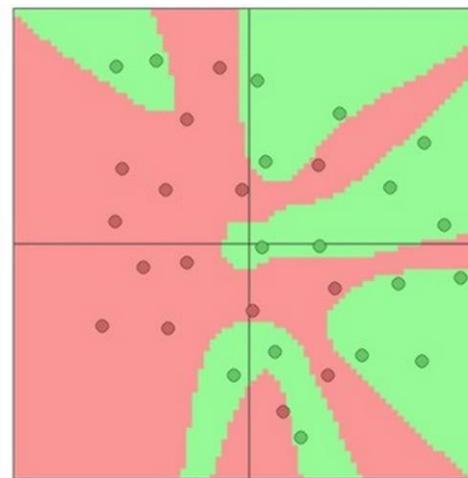
# Last time: Neural Networks

From linear classifiers to  
fully-connected networks

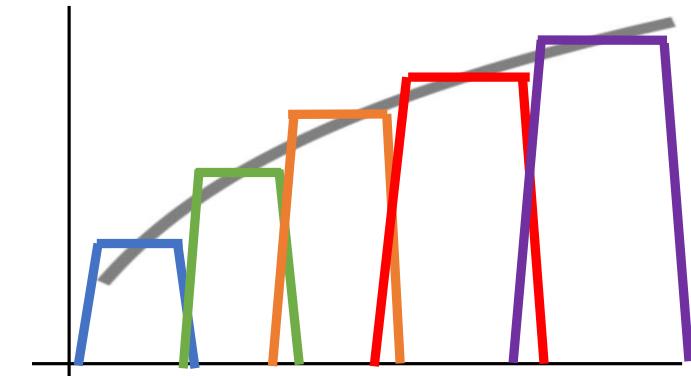
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



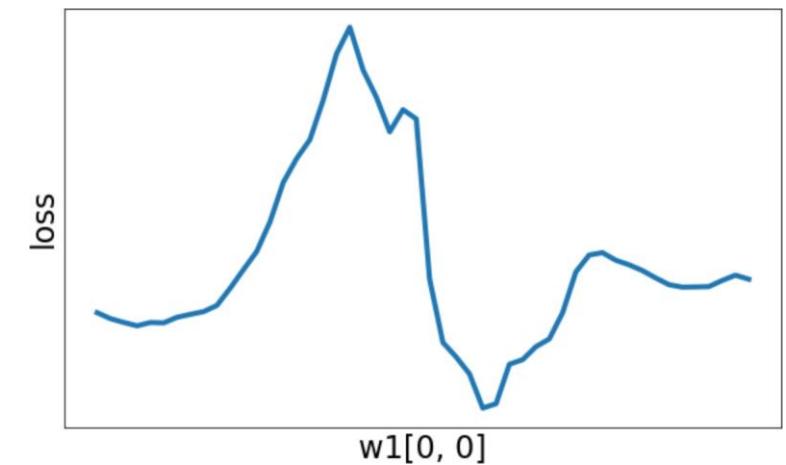
Space Warping



Universal Approximation



Nonconvex



# Problem: How to compute gradients?

$$s = W_2 \max(0, W_1 x + b_1) + b_2 \quad \text{Nonlinear score function}$$

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

$$R(W) = \sum_k W_k^2 \quad \text{L2 Regularization}$$

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

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

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

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

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

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

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

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

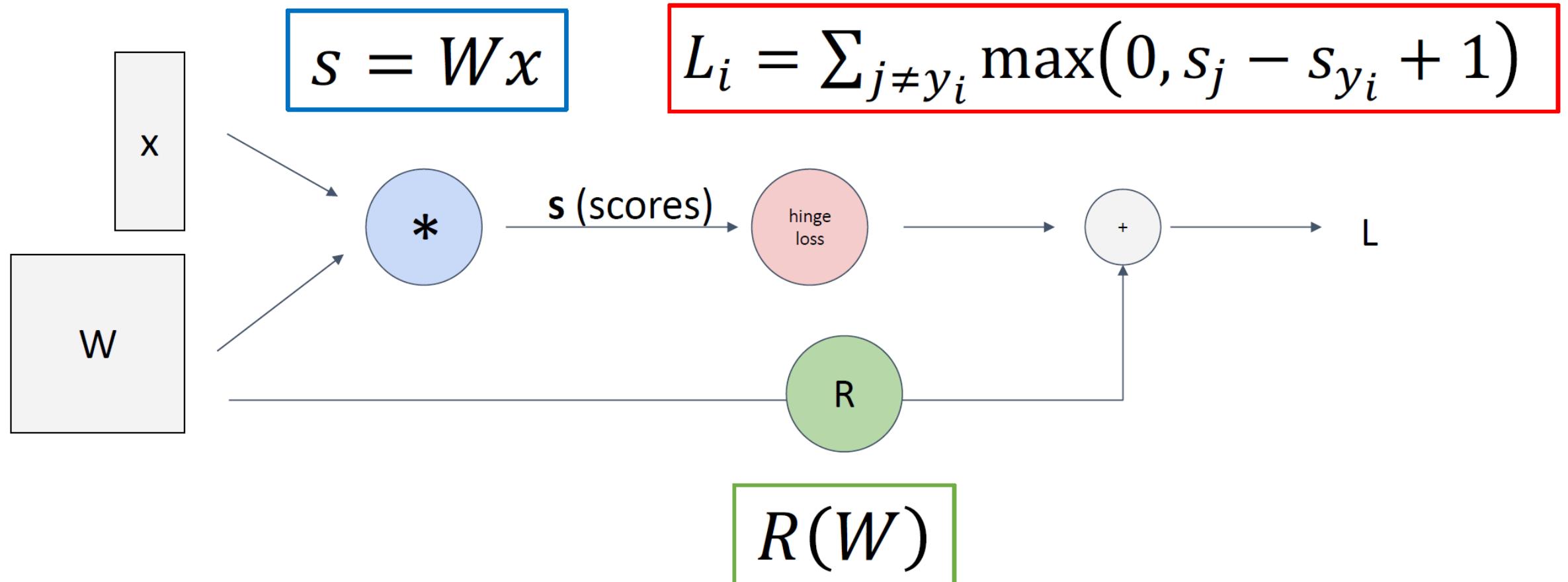
$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

**Problem:** Very tedious: Lots of matrix calculus, need lots of paper

**Problem:** What if we want to change 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

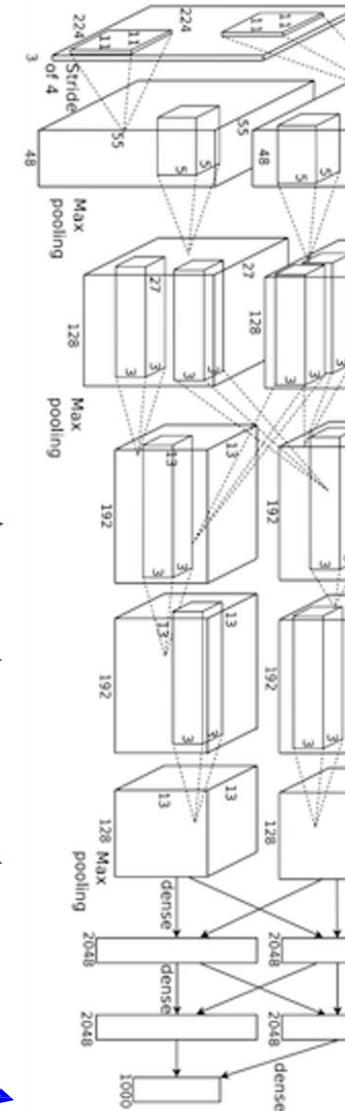


# Deep Network (AlexNet)

input image

weights

loss



# Neural Turing Machine

input image

loss

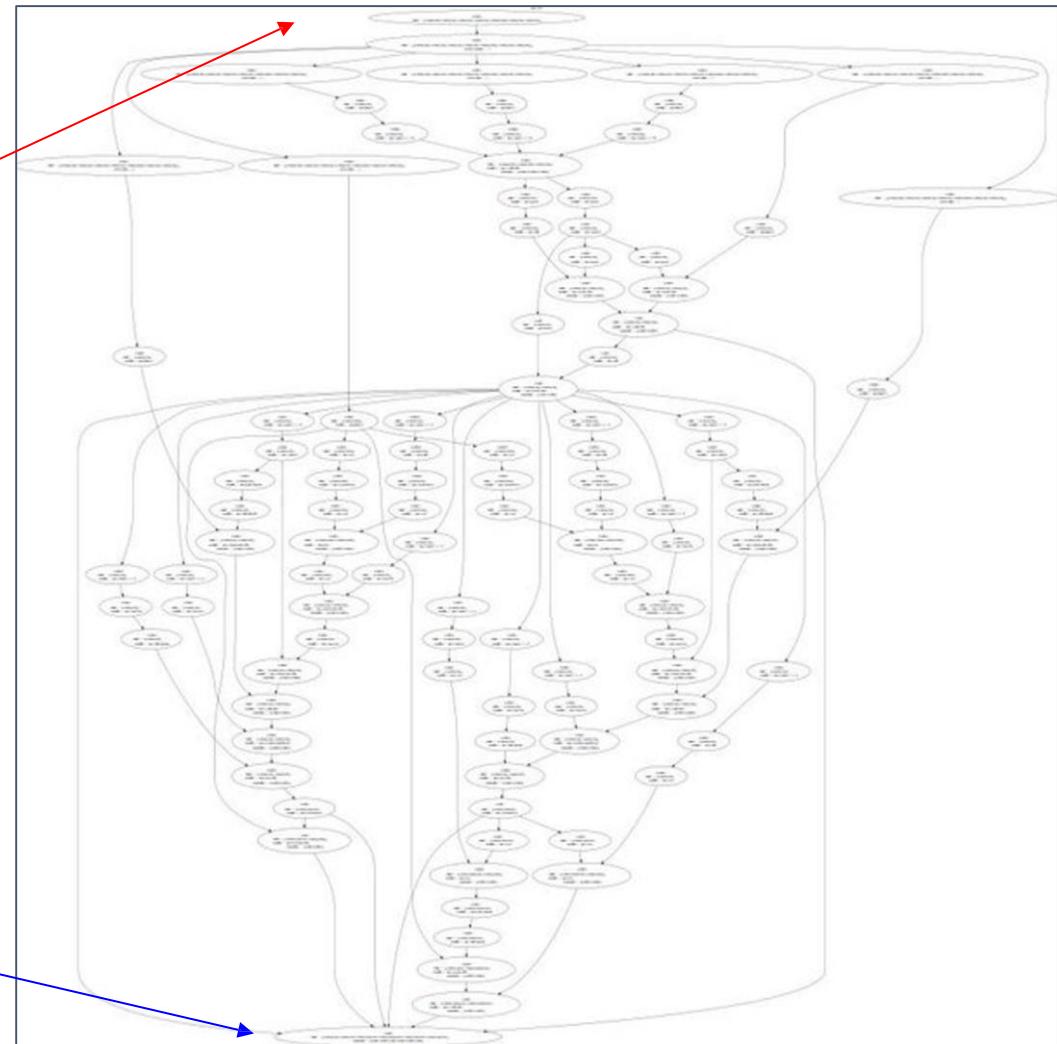
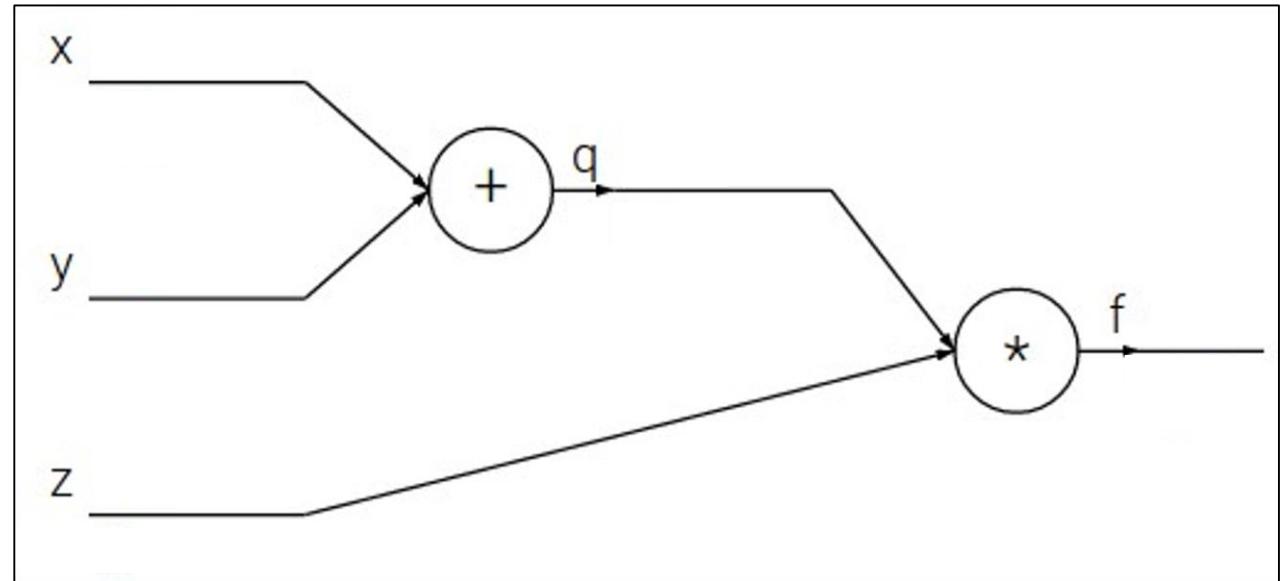


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

# Backpropagation: Simple Example

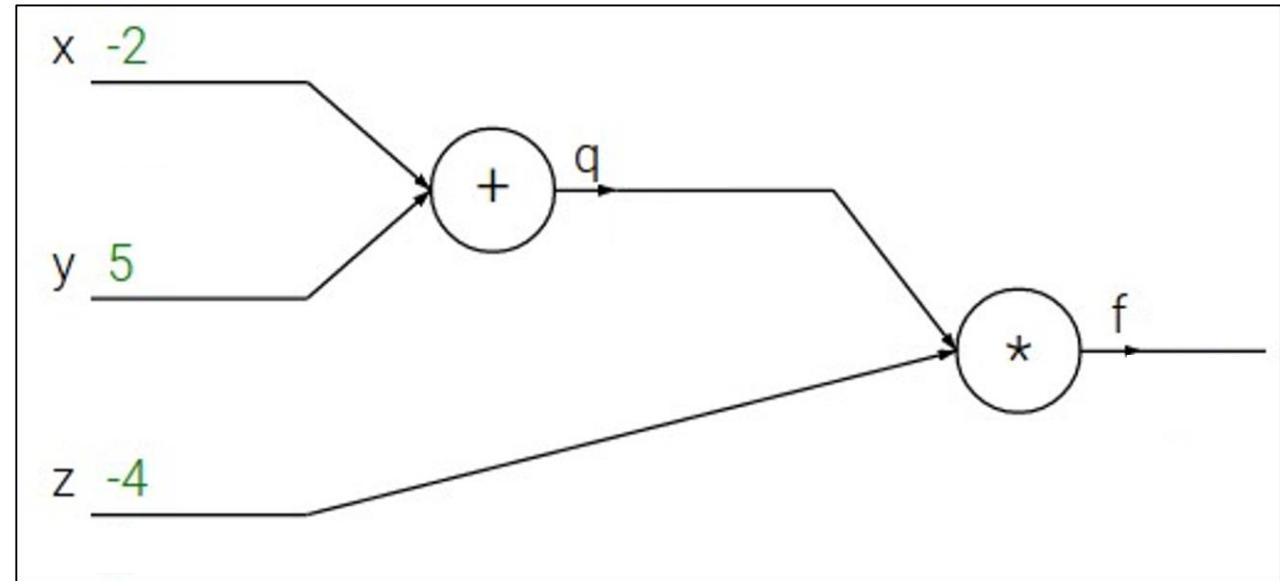
$$f(x, y, z) = (x + y) \cdot z$$



# Backpropagation: Simple Example

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

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



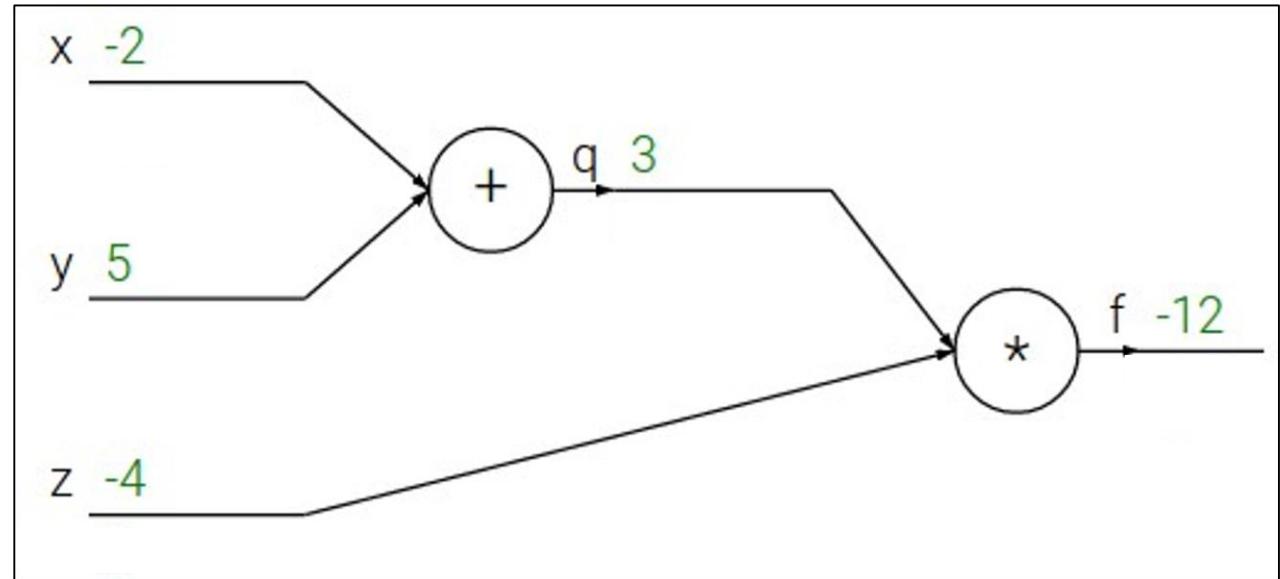
# Backpropagation: Simple Example

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

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

**1. Forward pass:** Compute outputs

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



# Backpropagation: Simple Example

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

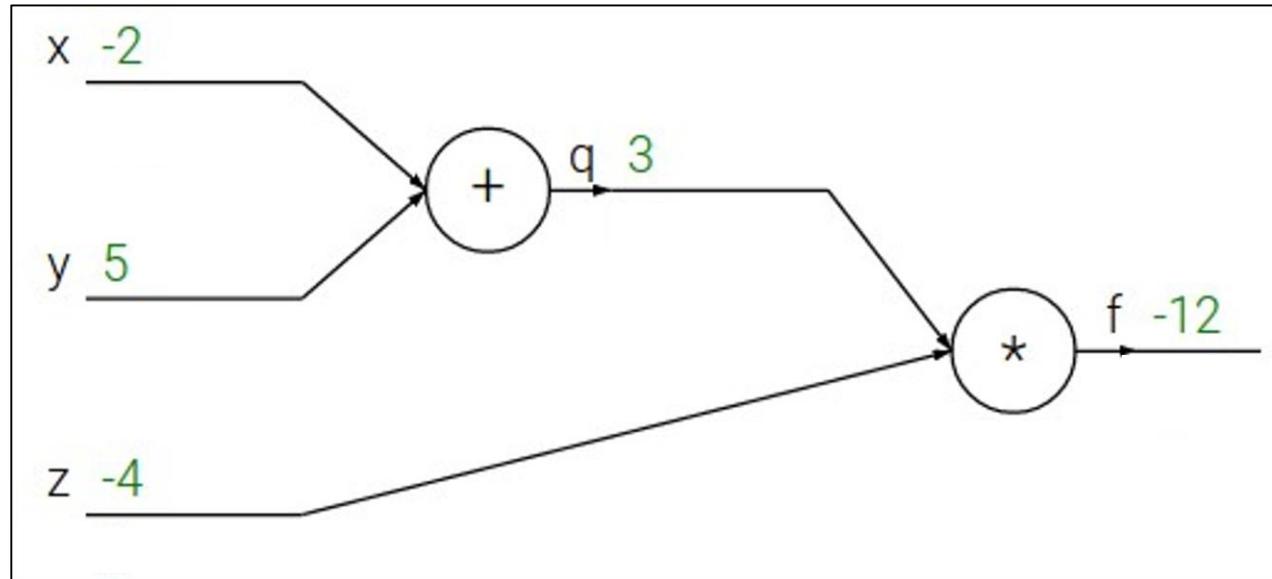
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

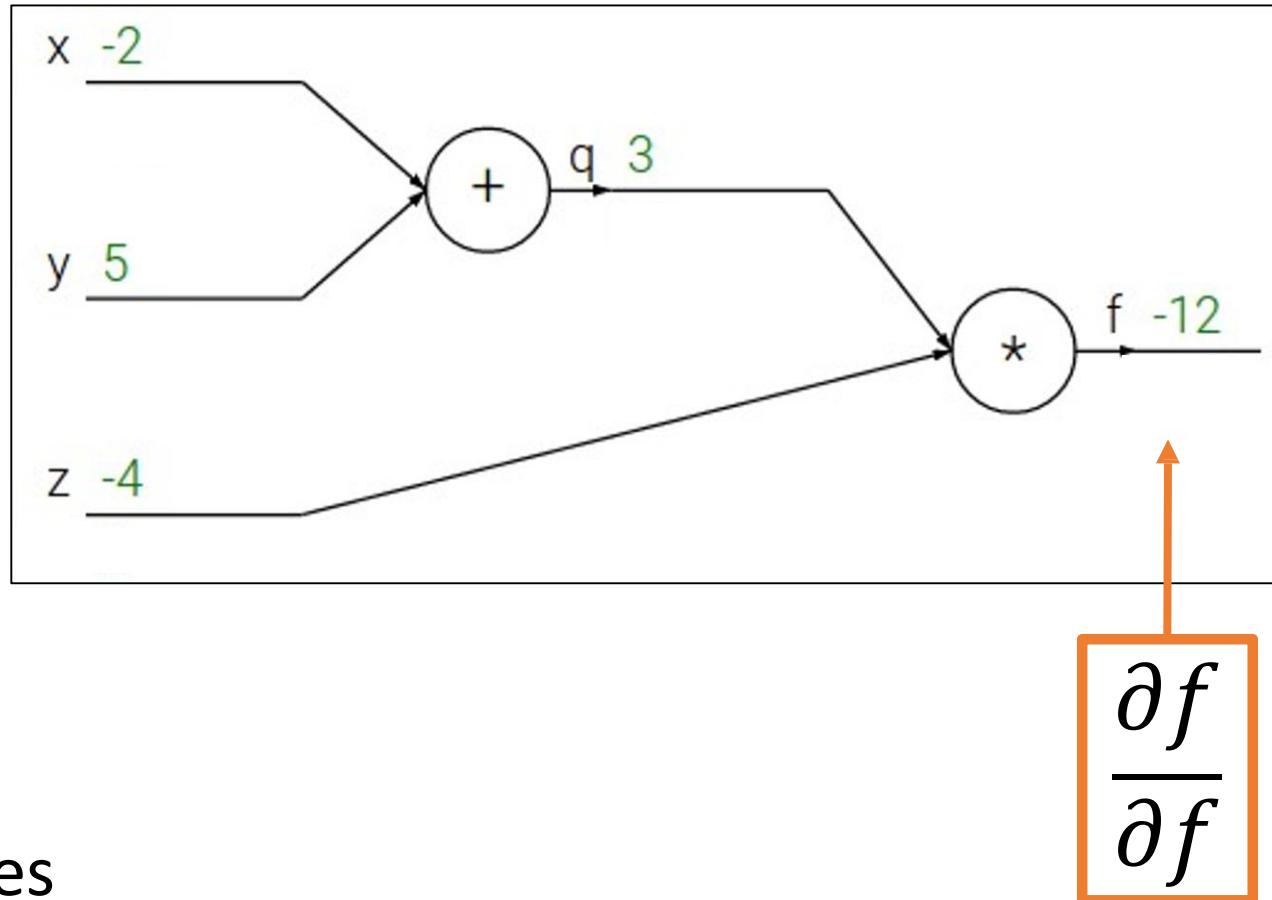
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

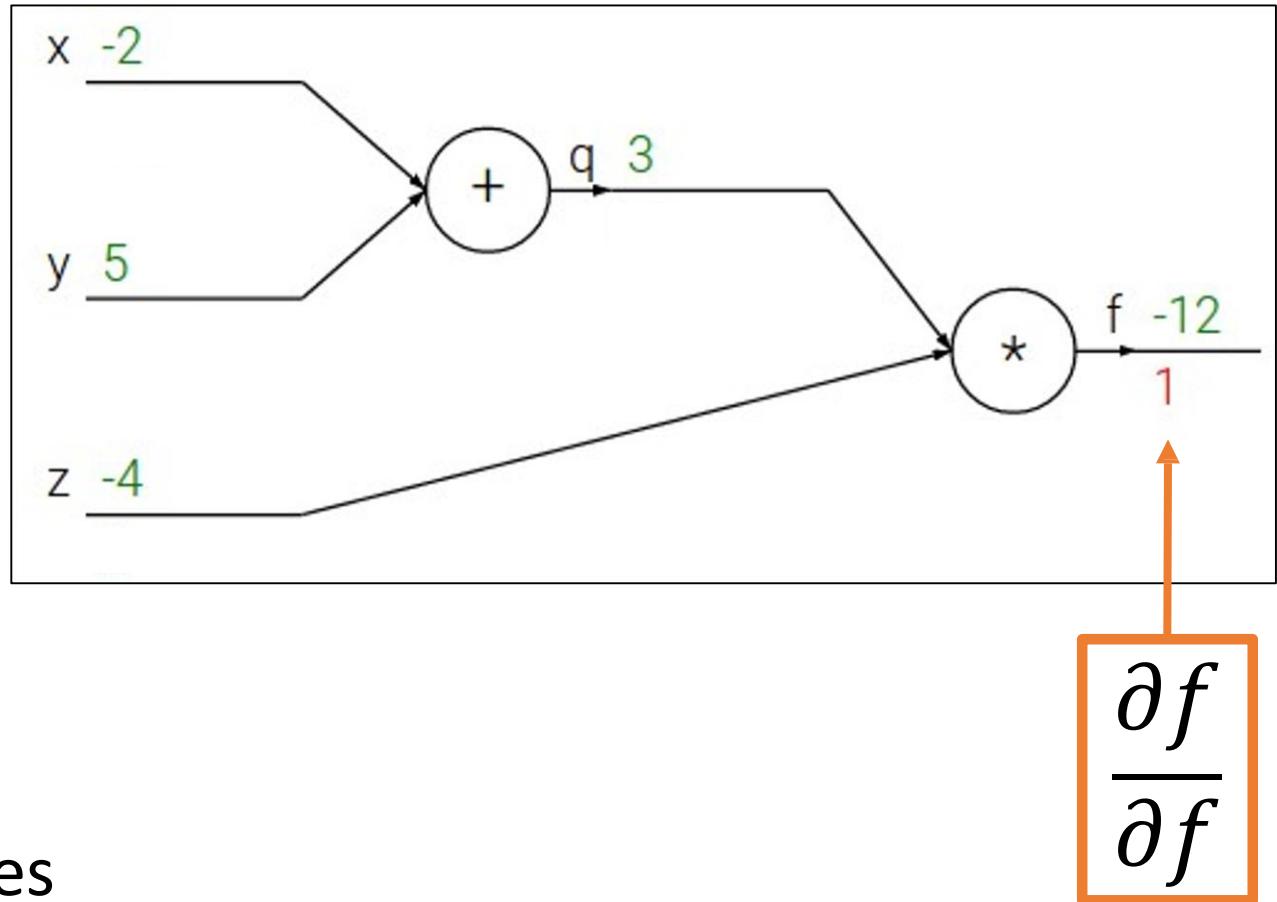
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

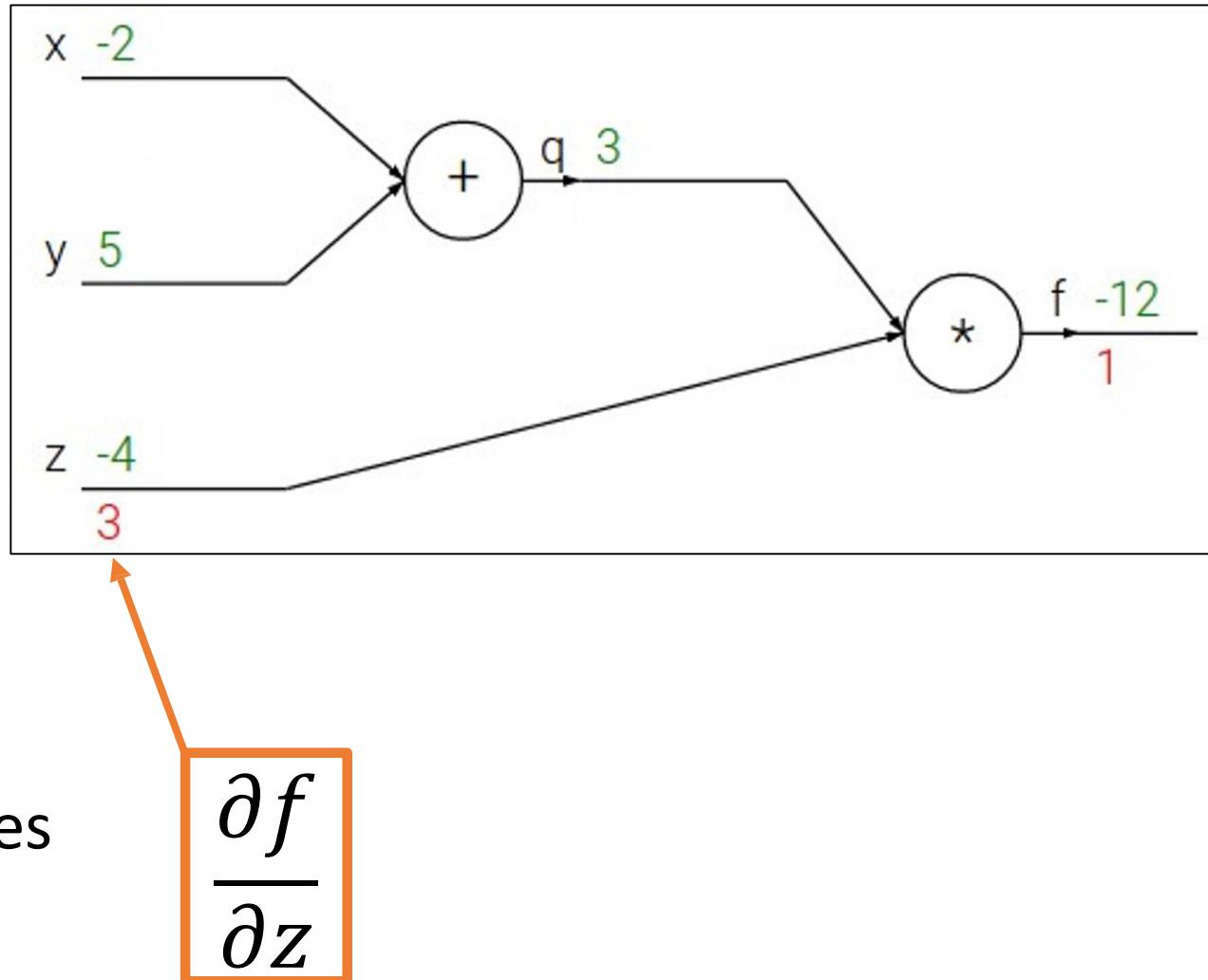
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

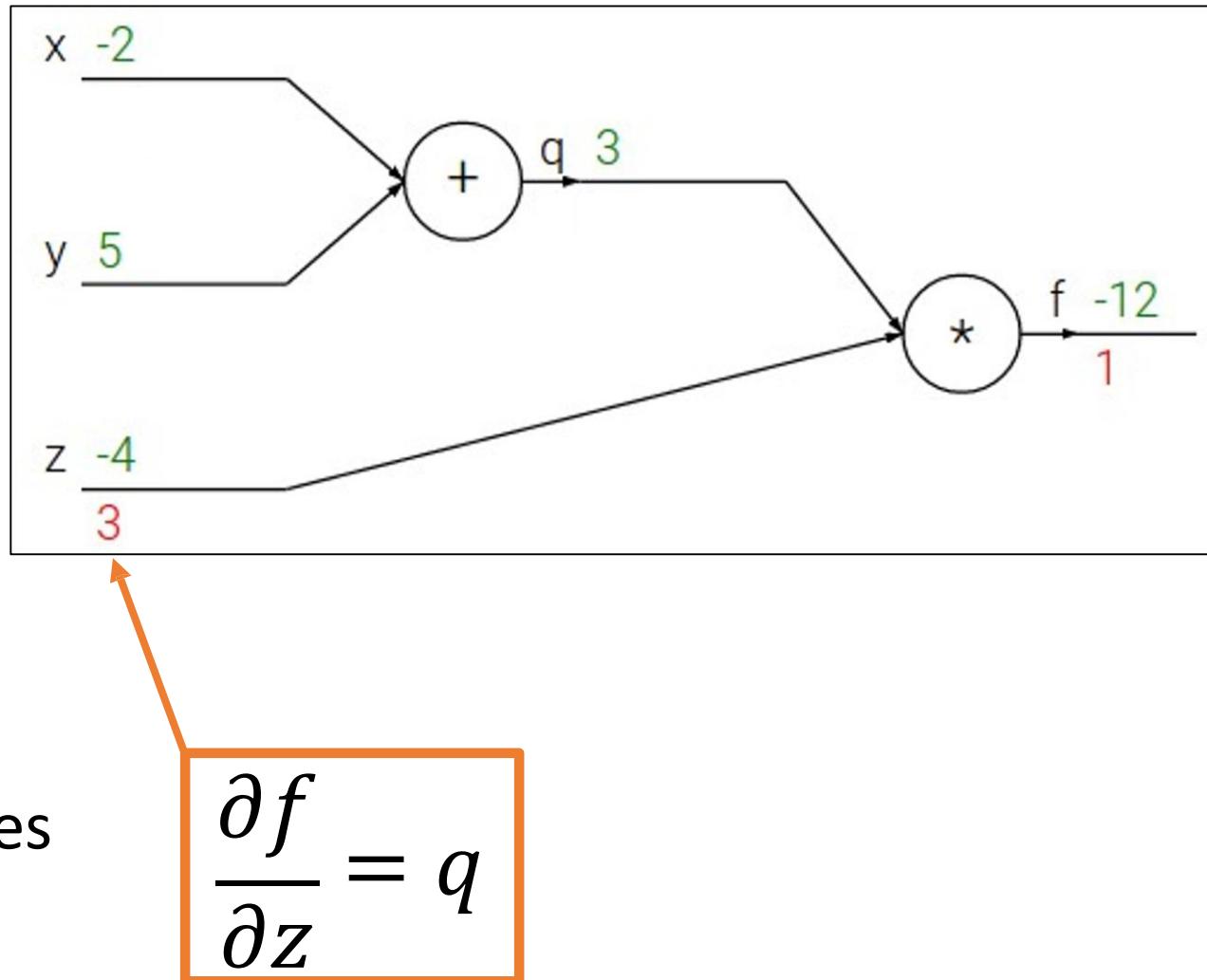
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

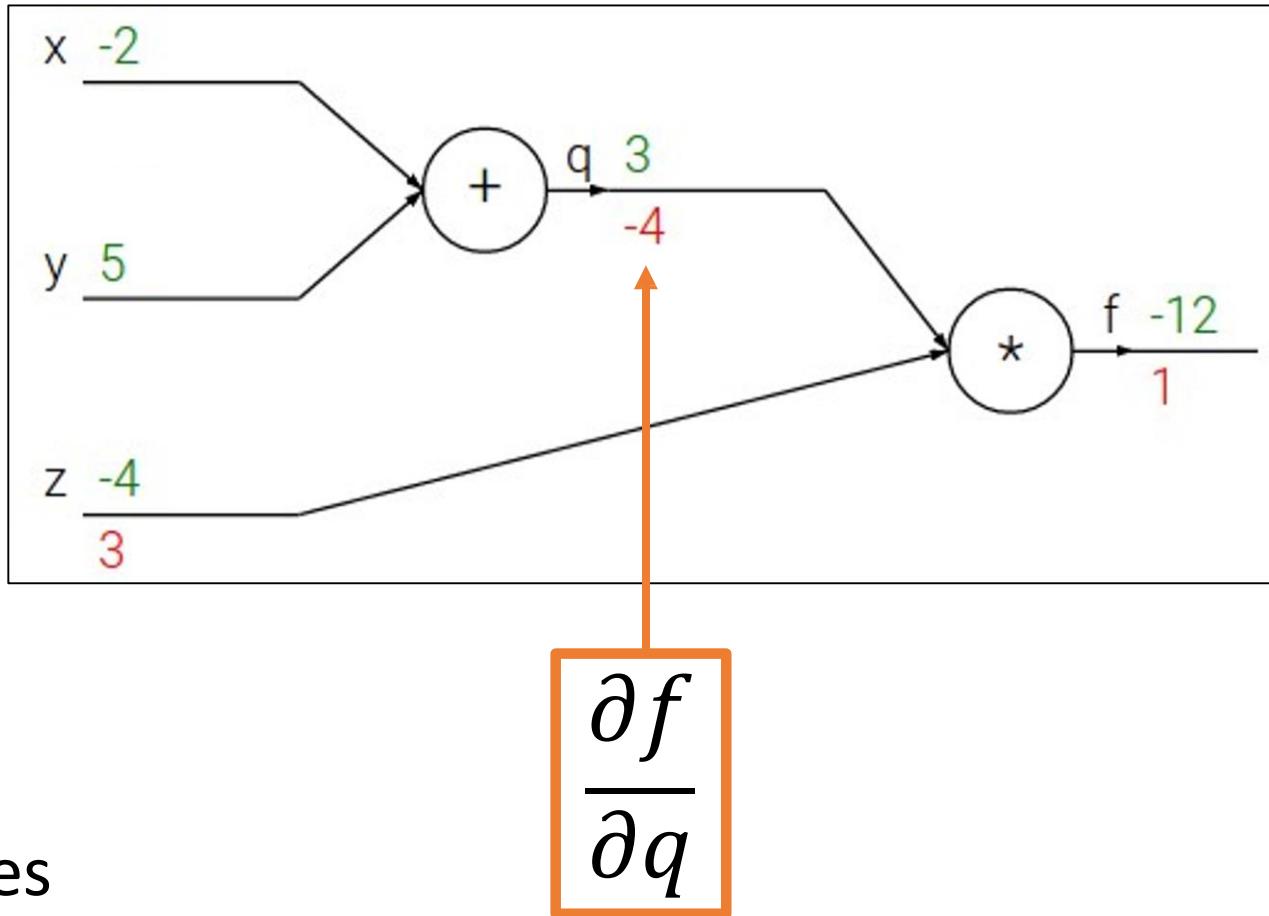
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

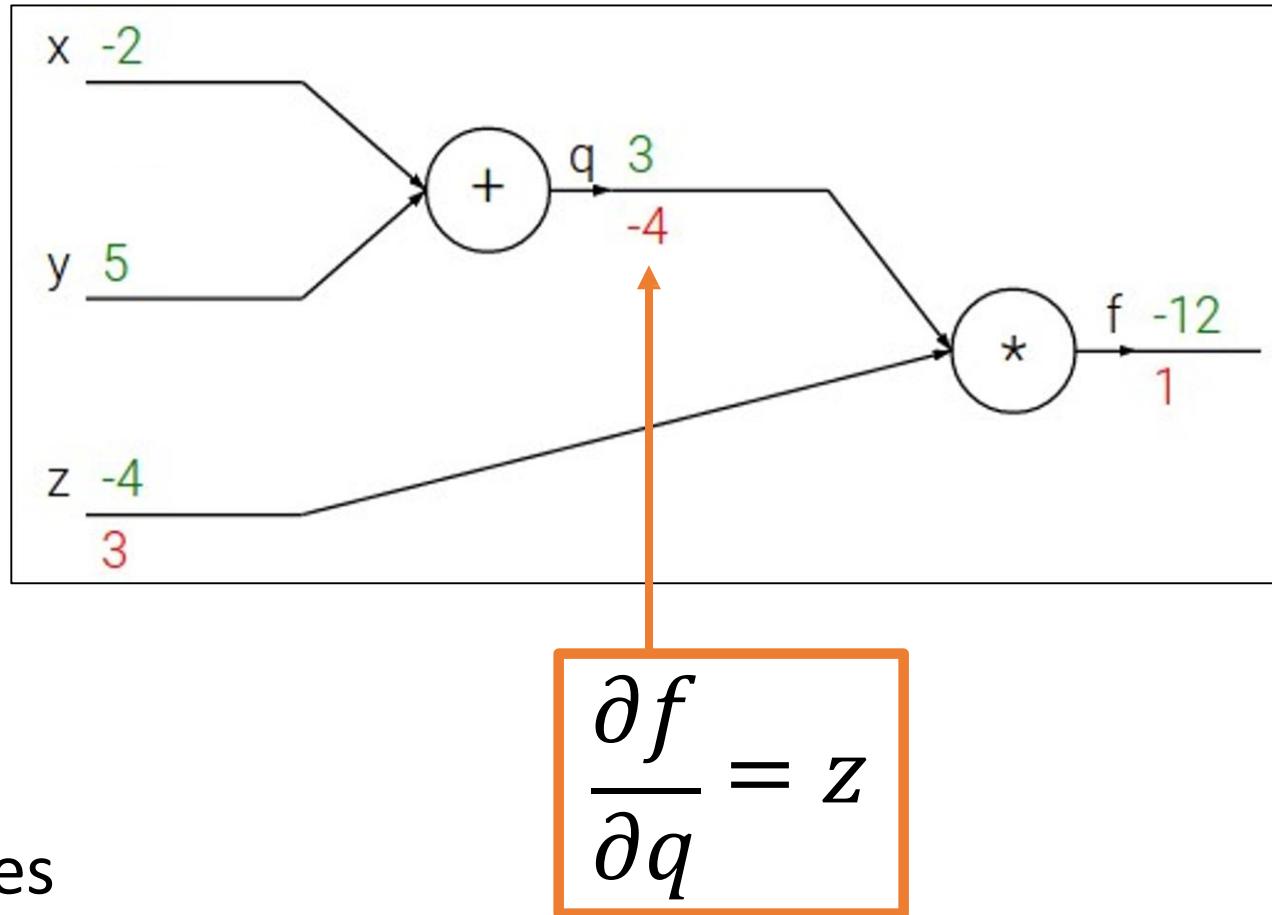
e.g.  $x = -2, y = 5, z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

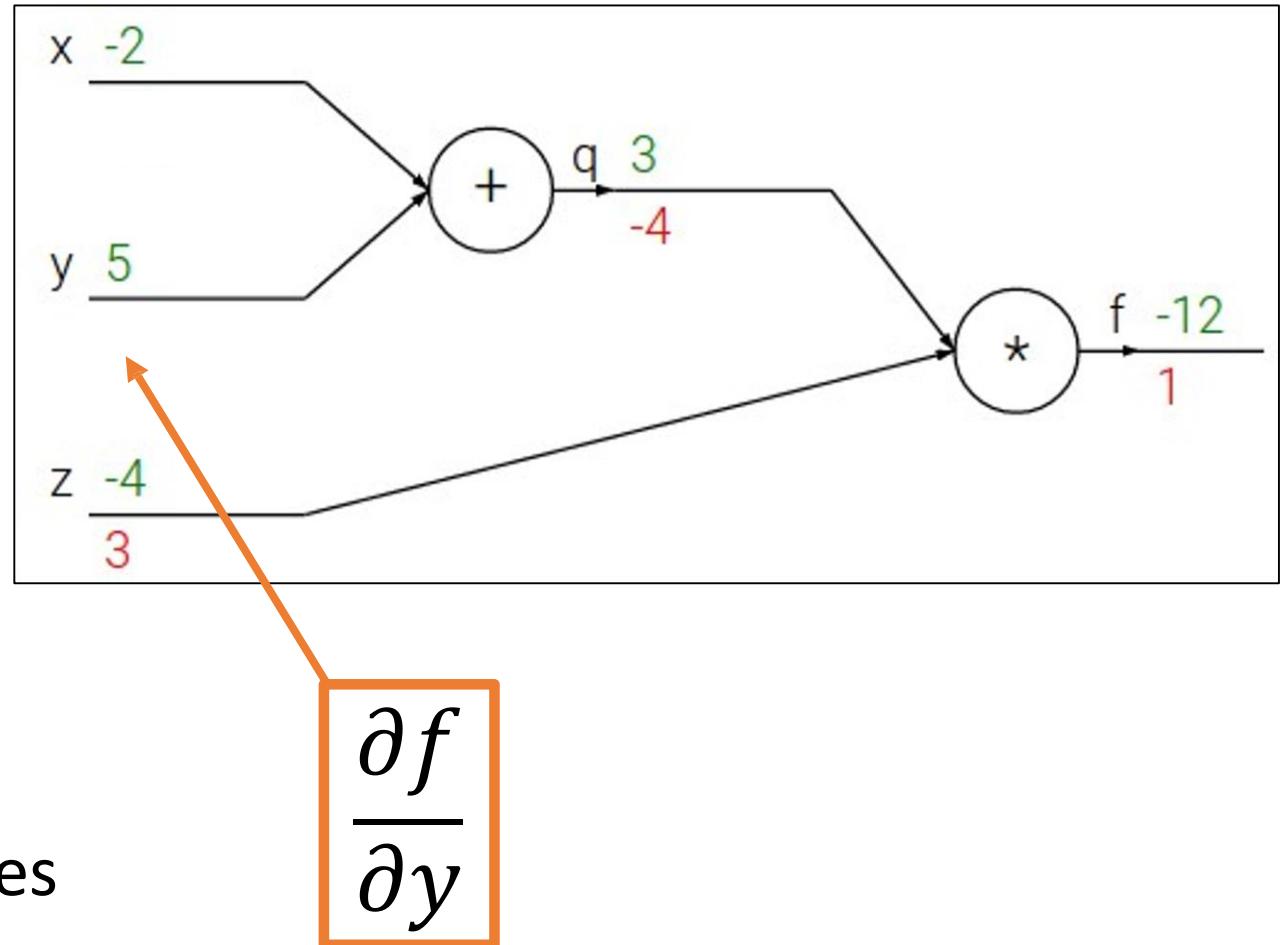
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



# Backpropagation: Simple Example

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

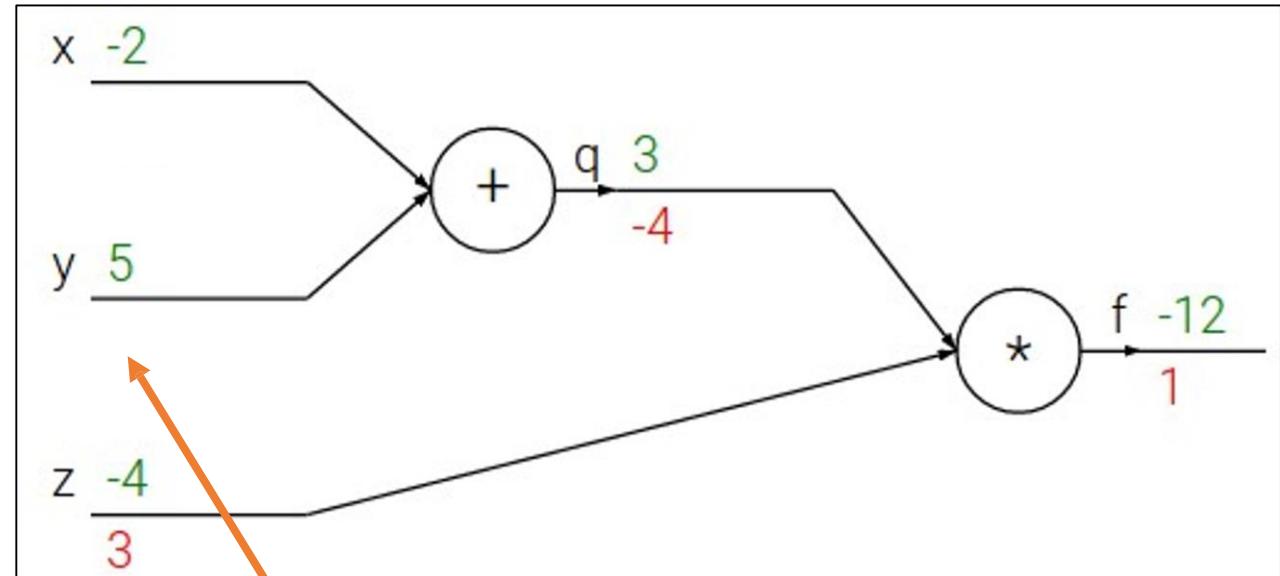
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



**Chain Rule**

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

# Backpropagation: Simple Example

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

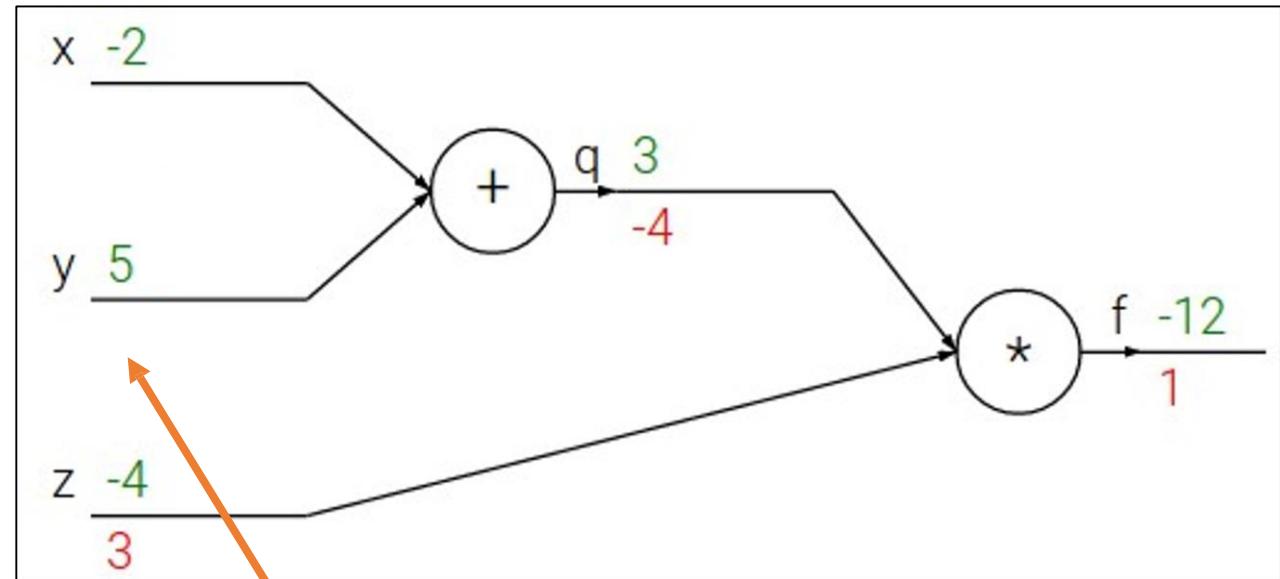
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

**1. Forward pass:** Compute outputs

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

**2. Backward pass:** Compute derivatives

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



**Chain Rule**

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

Downstream  
Gradient

Local  
Gradient

Upstream  
Gradient

$$\frac{\partial q}{\partial y} = 1$$

# Backpropagation: Simple Example

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

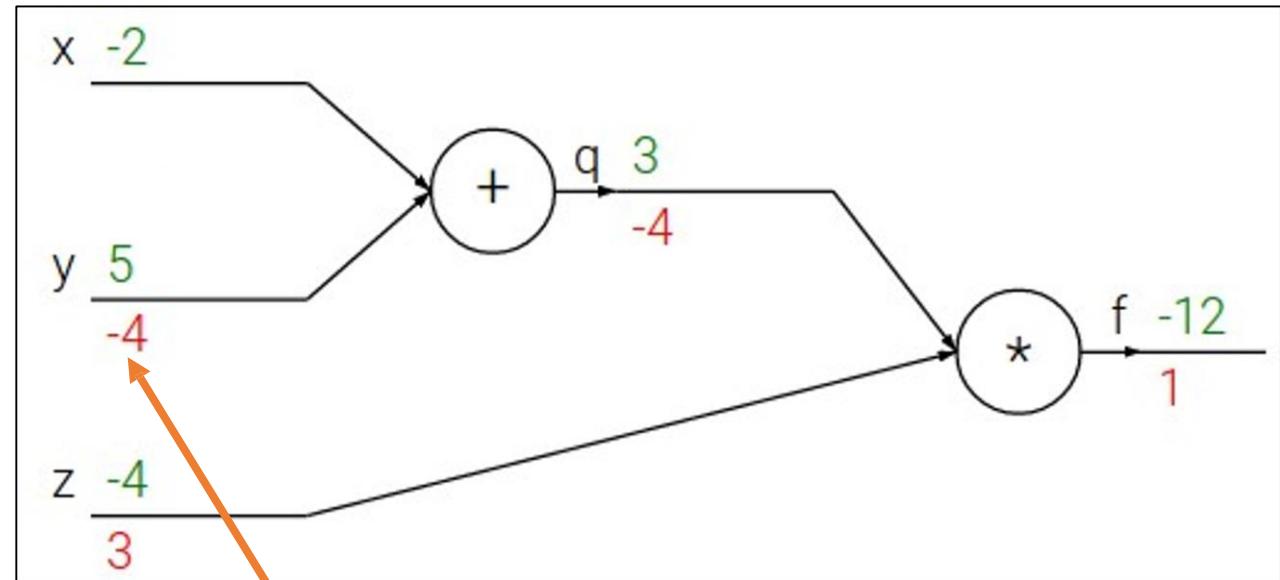
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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



Chain Rule

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

Downstream  
Gradient

Local  
Gradient

Upstream  
Gradient

$$\frac{\partial q}{\partial y} = 1$$

# Backpropagation: Simple Example

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

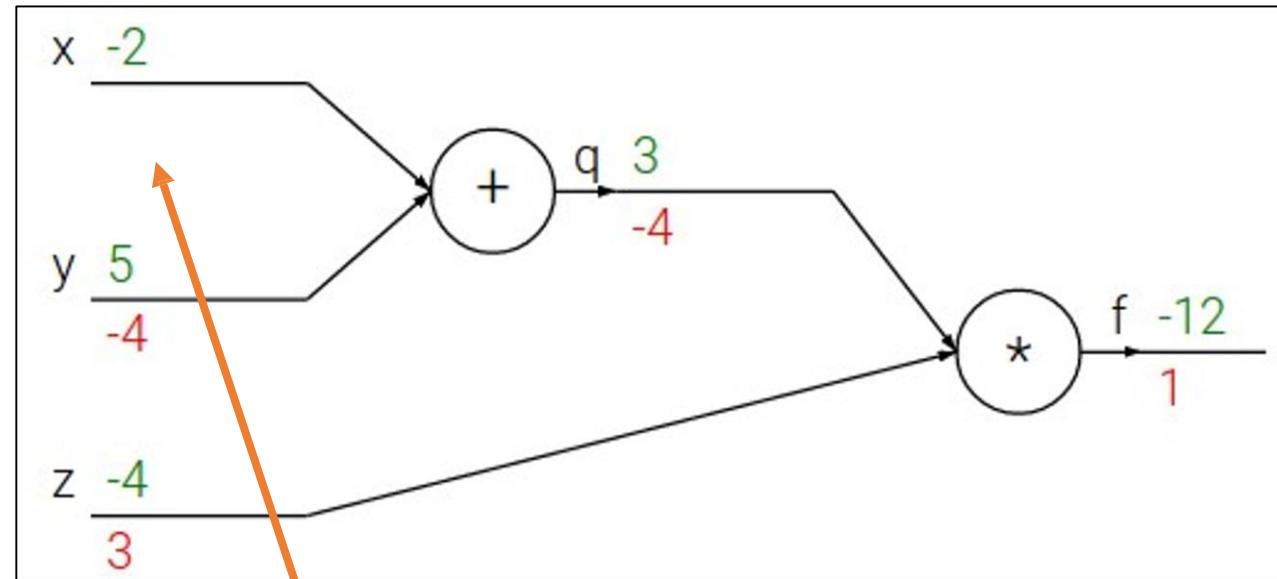
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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



Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

Downstream  
Gradient

Local  
Gradient

Upstream  
Gradient

$$\frac{\partial q}{\partial x} = 1$$

# Backpropagation: Simple Example

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

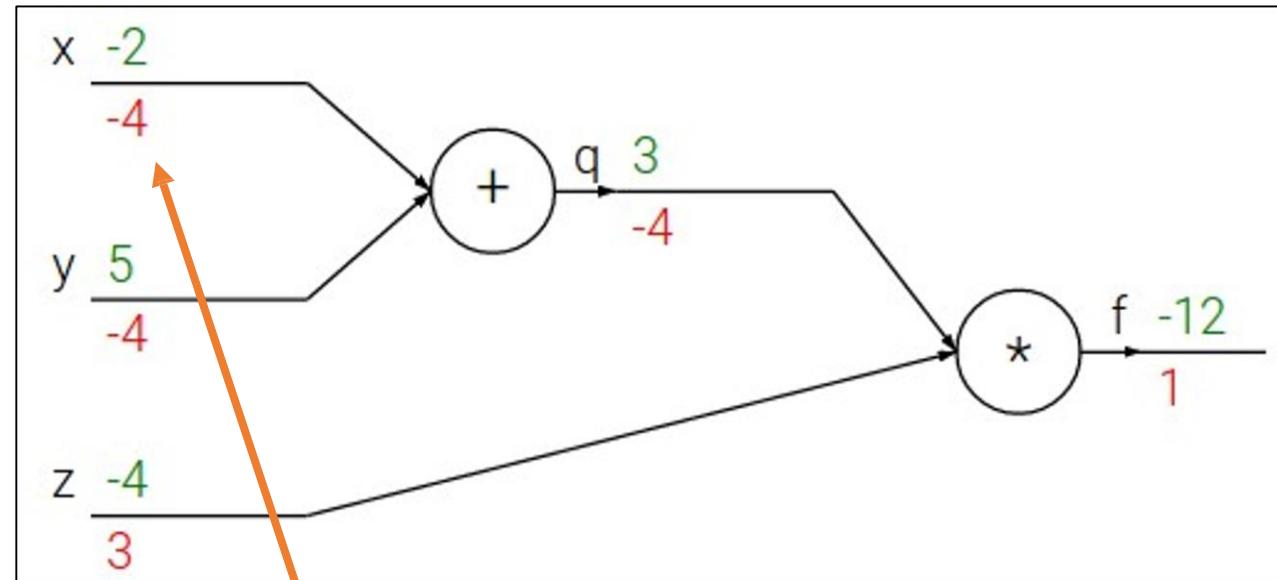
e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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



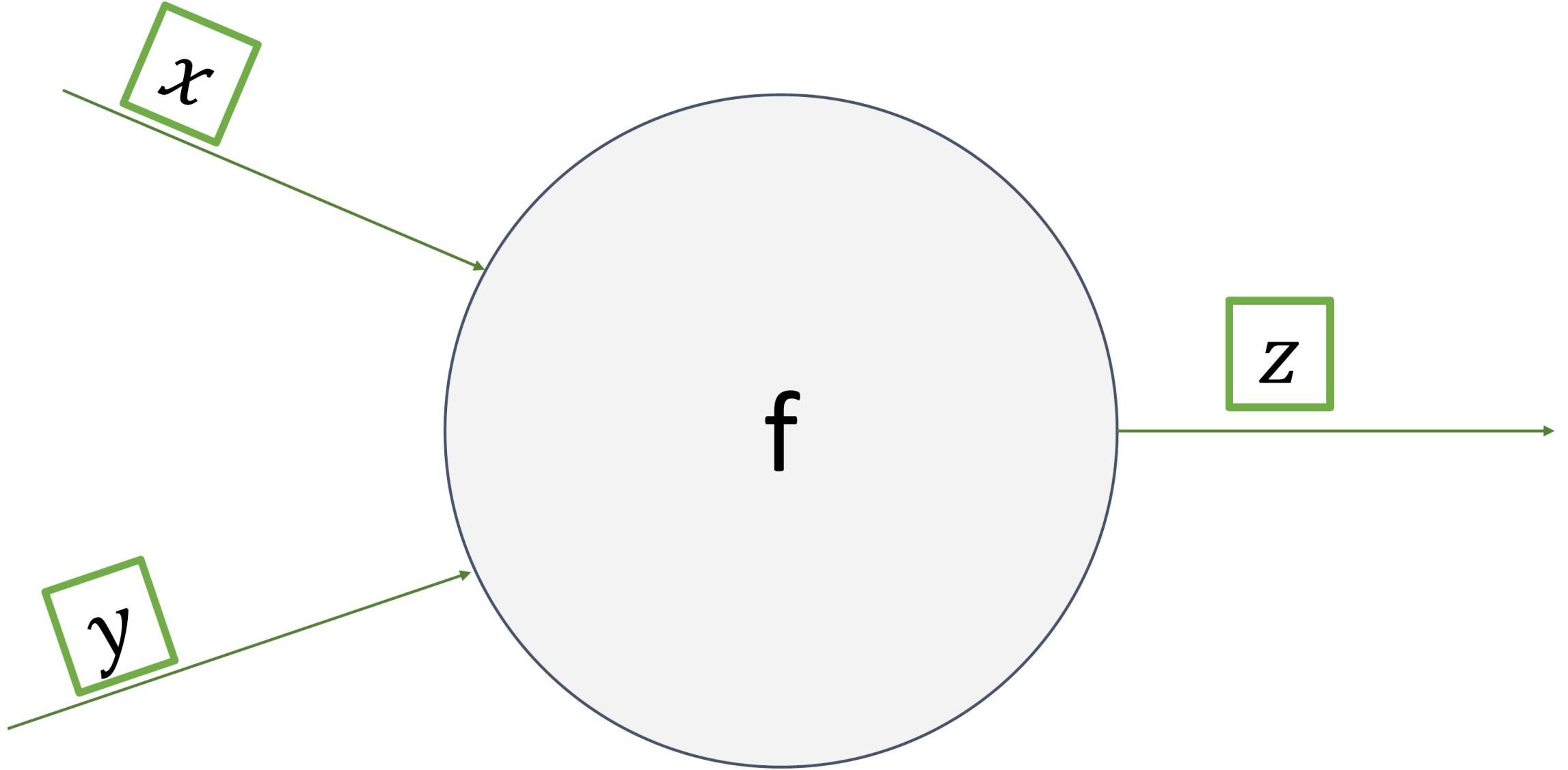
Chain Rule

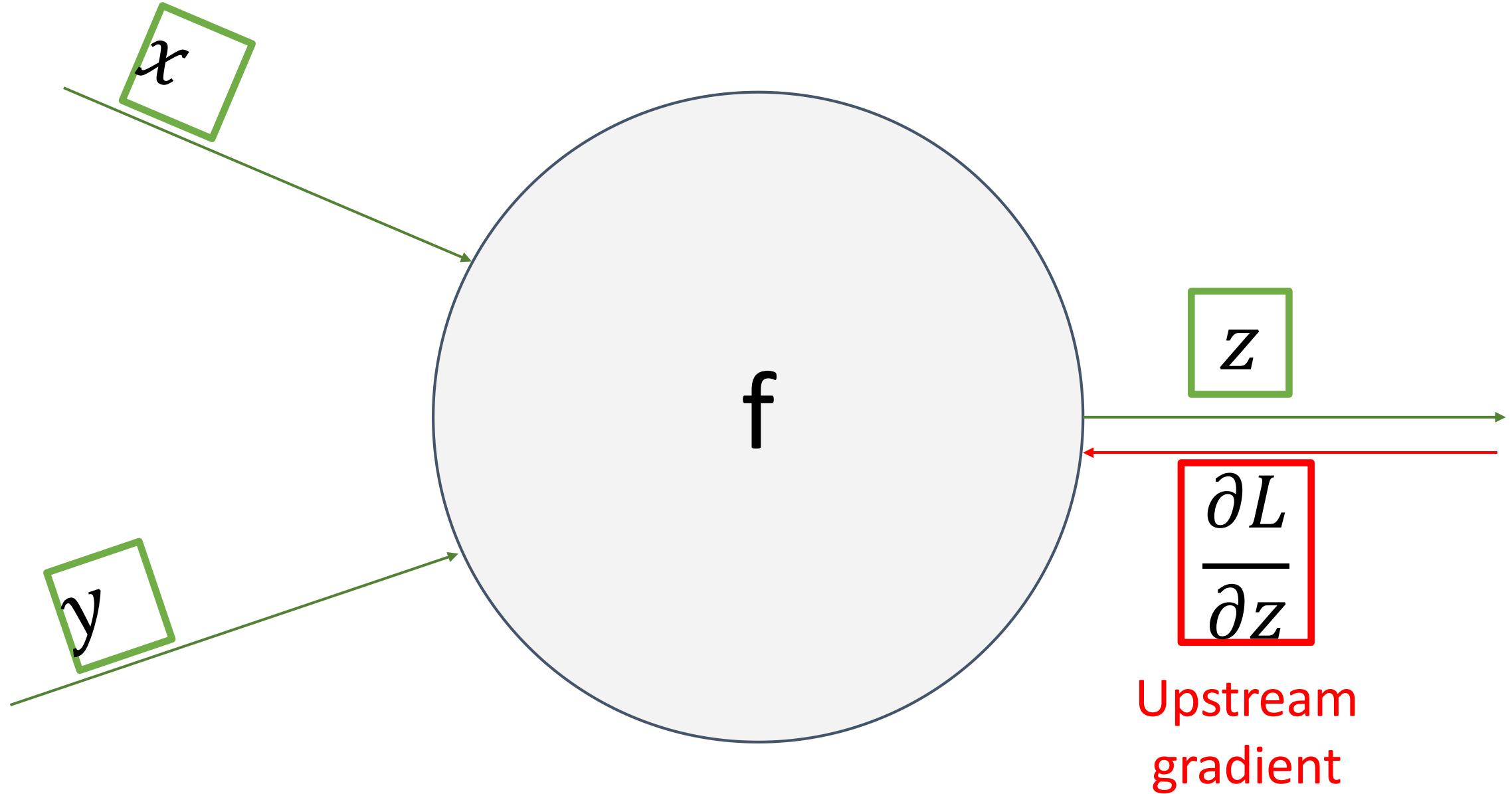
$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

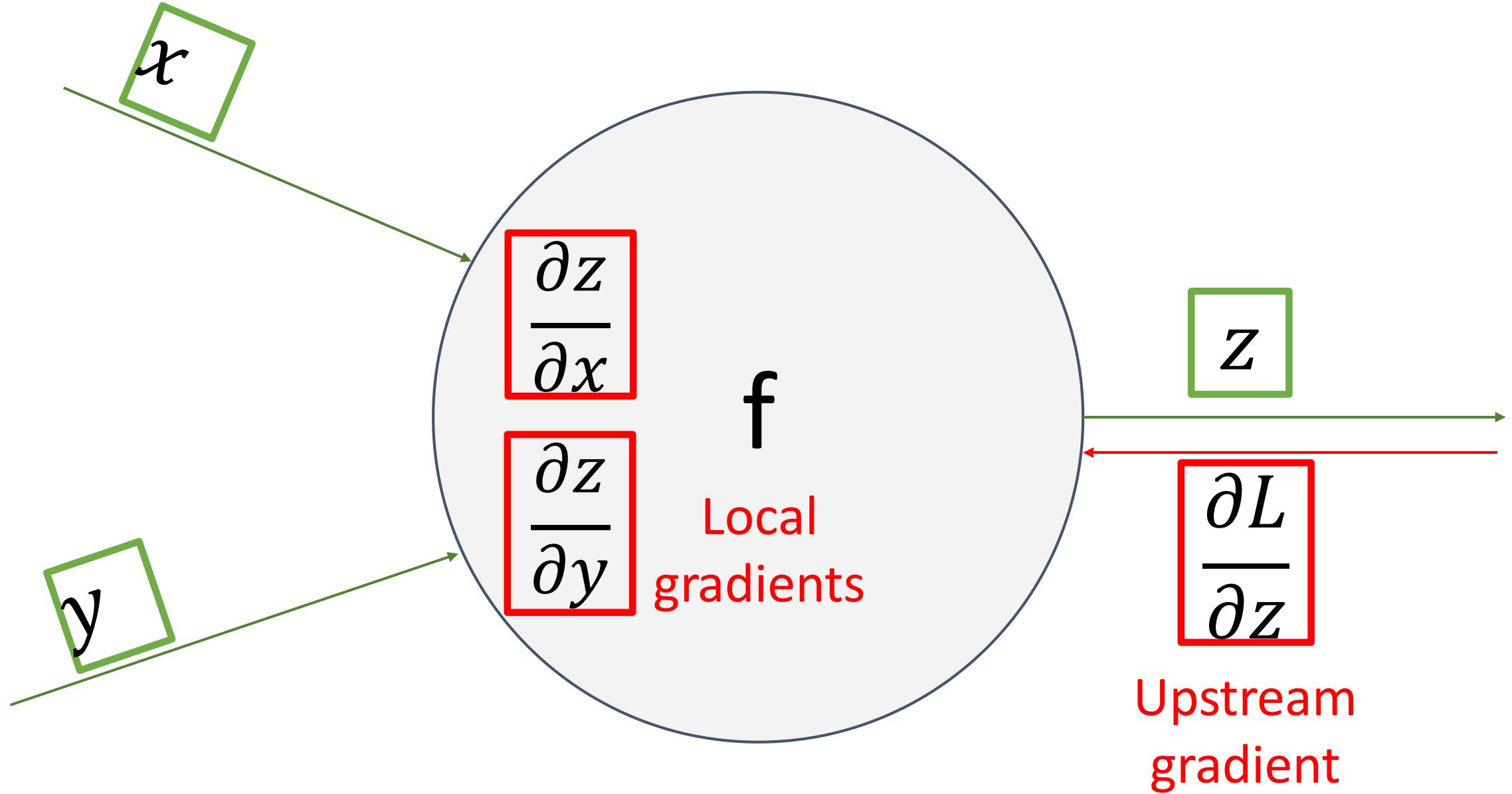
Downstream  
Gradient

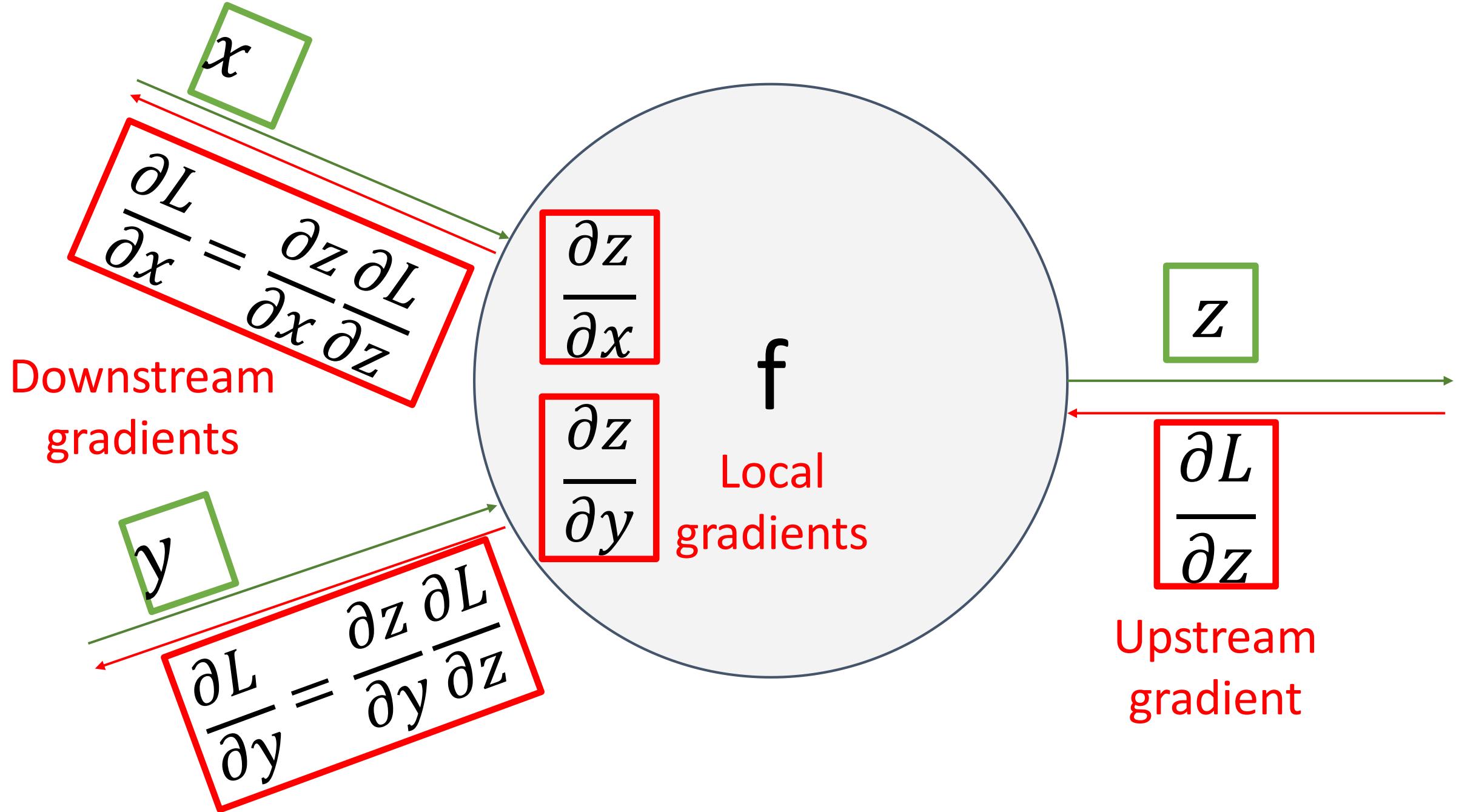
Local  
Gradient

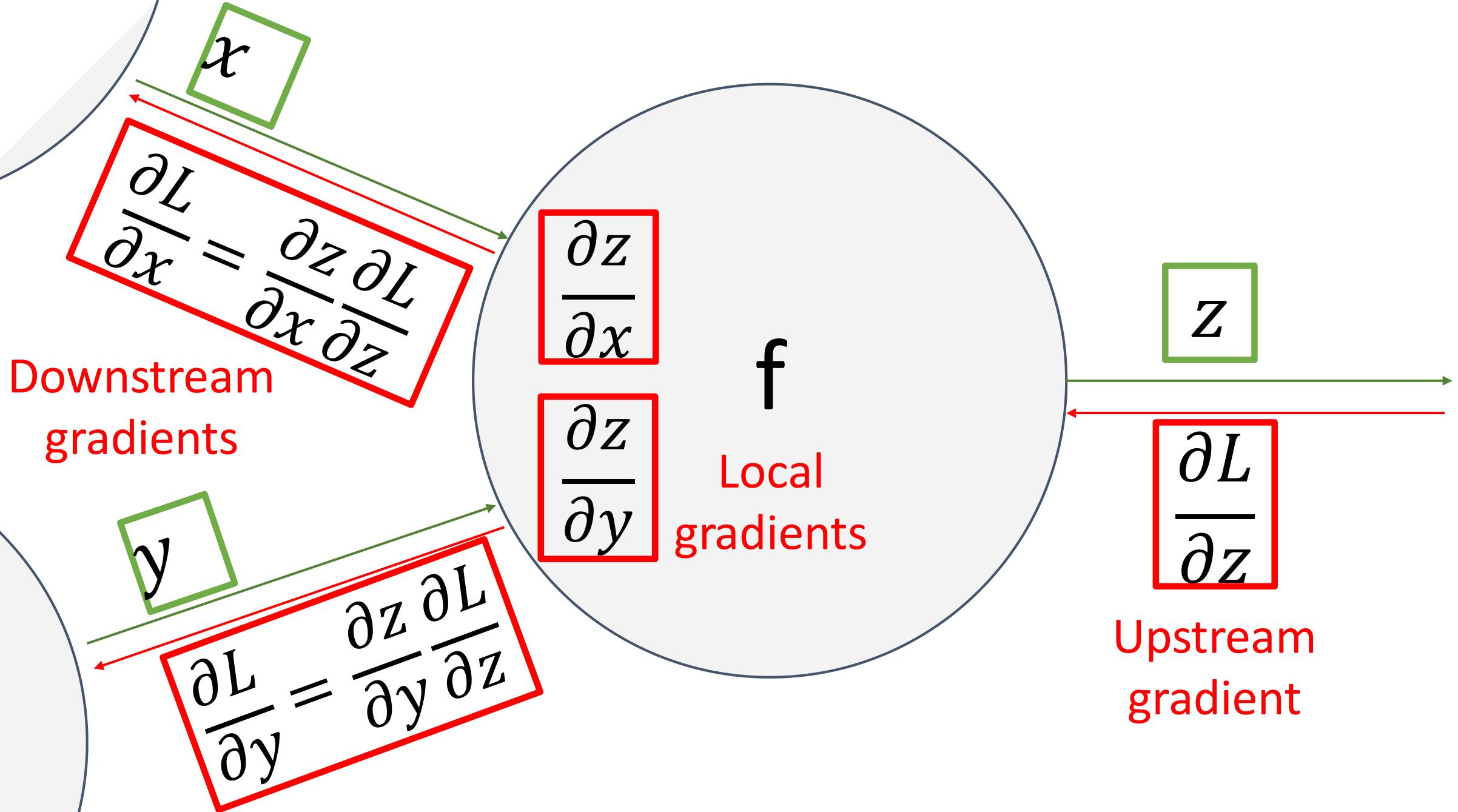
Upstream  
Gradient



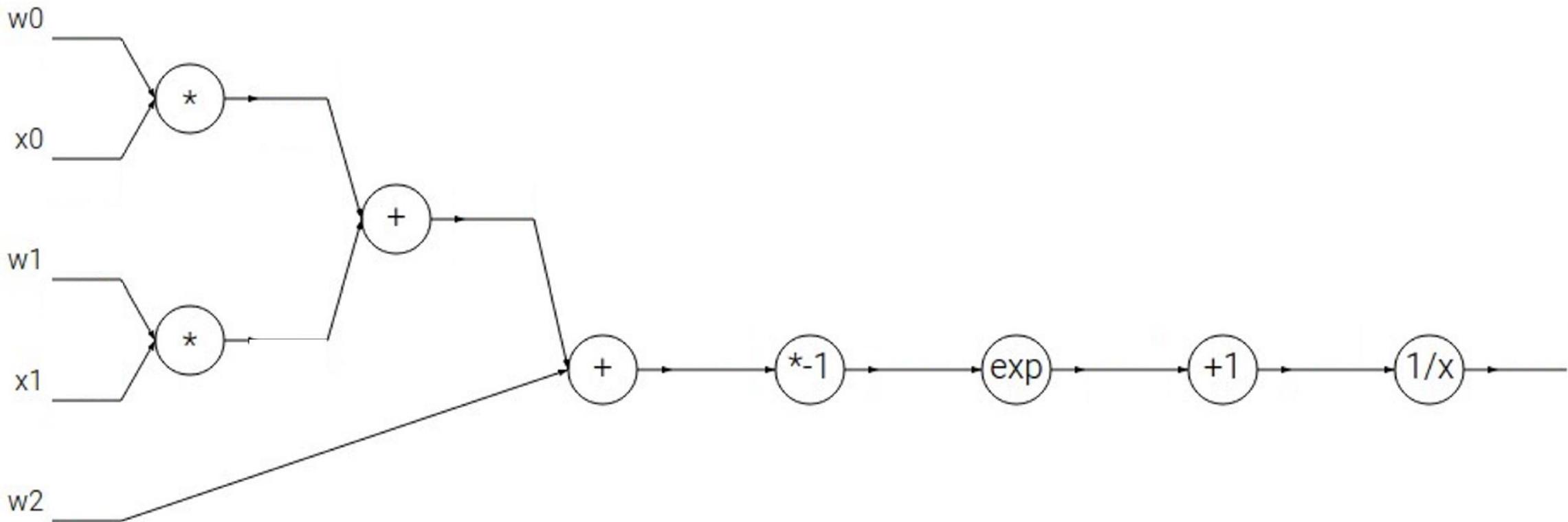








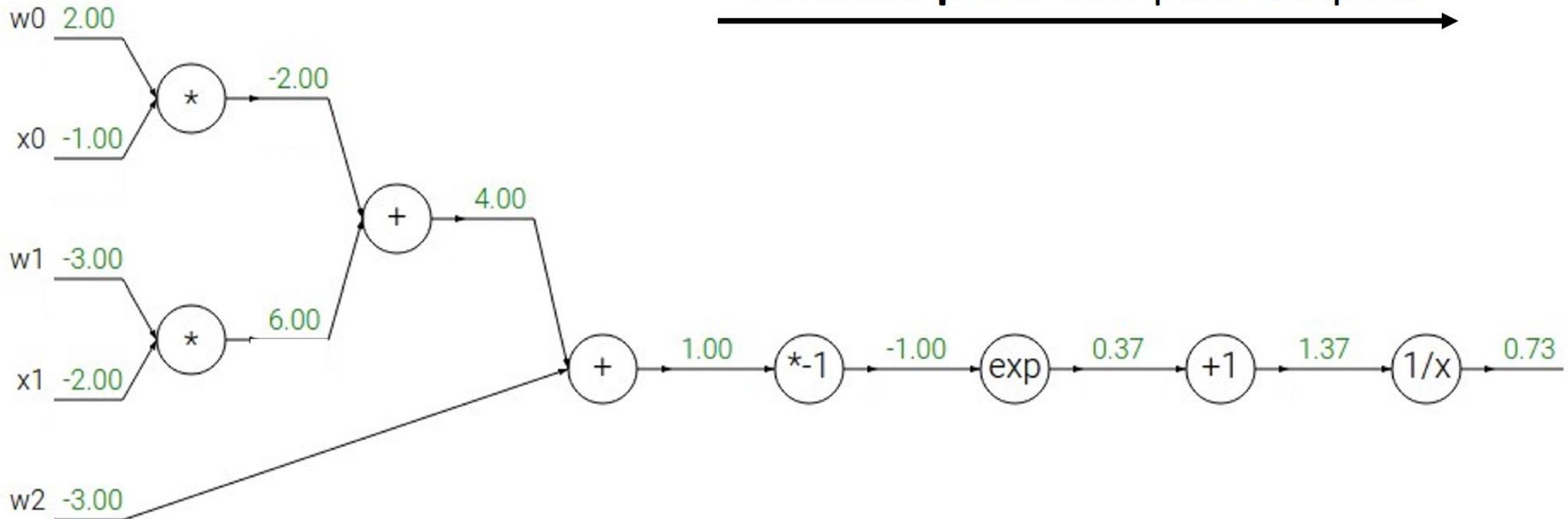
Another Example  $f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

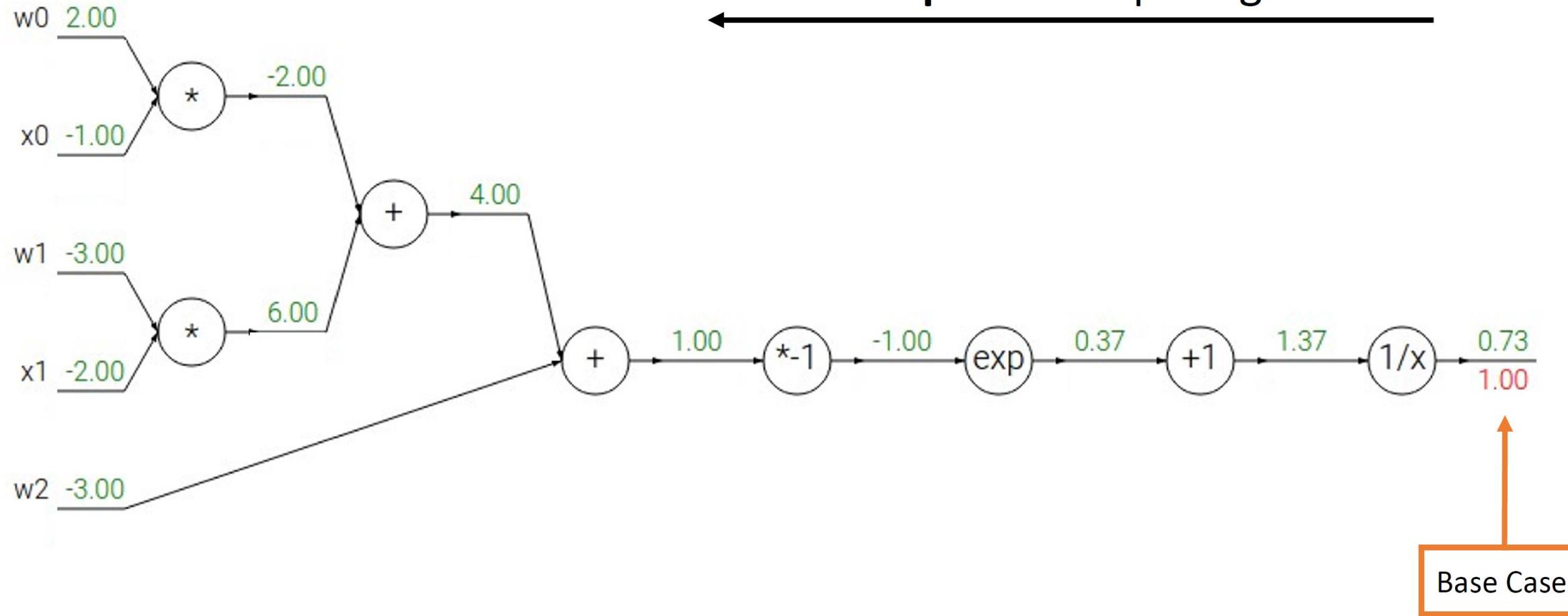
**Forward pass: Compute outputs**



Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

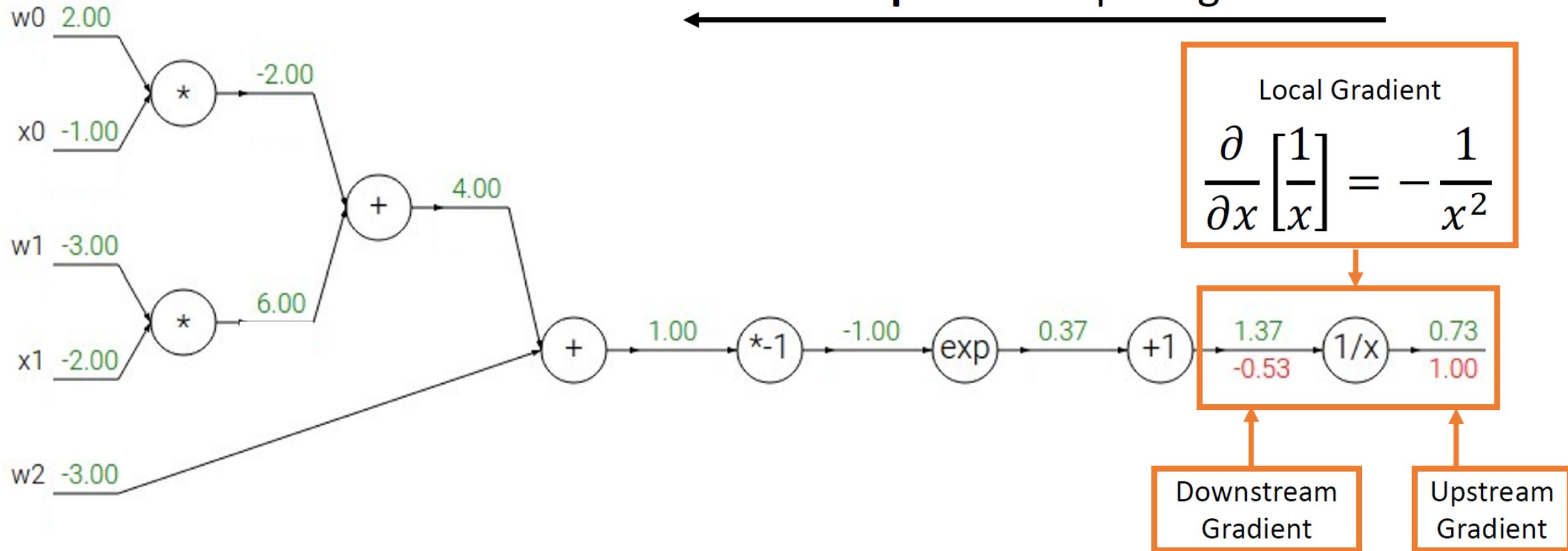
**Backward pass: Compute gradients**



Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

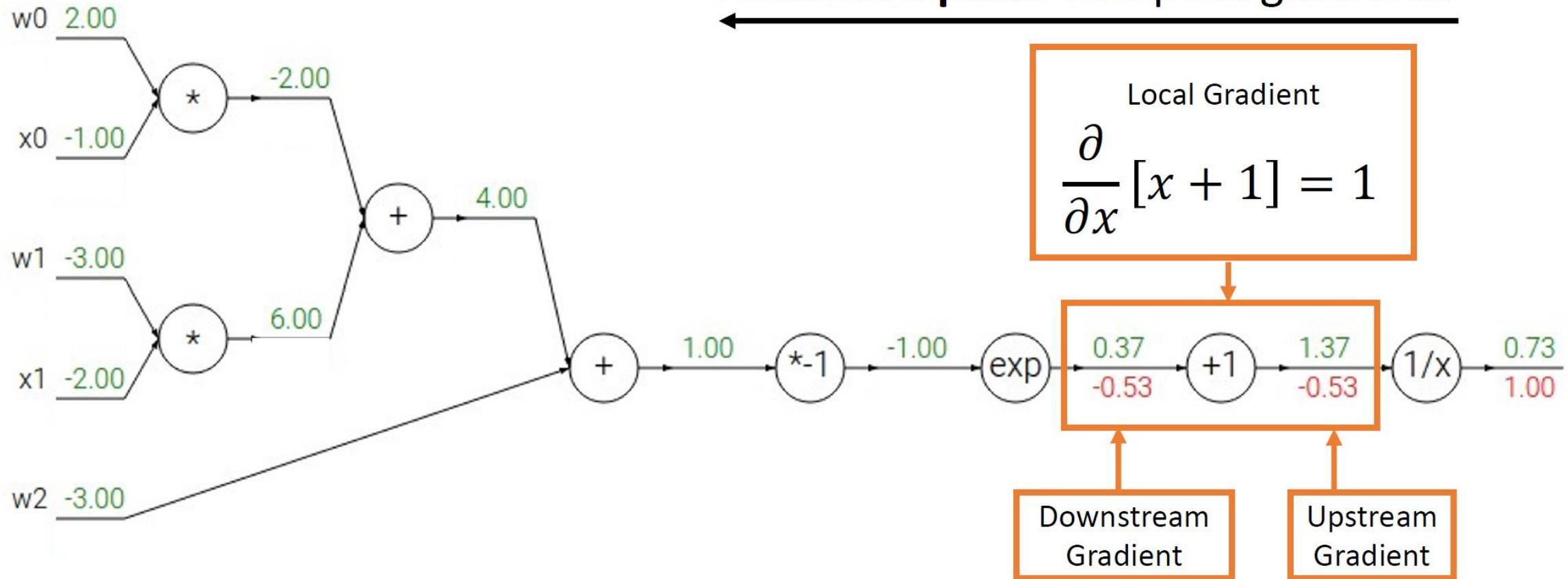
Backward pass: Compute gradients



Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

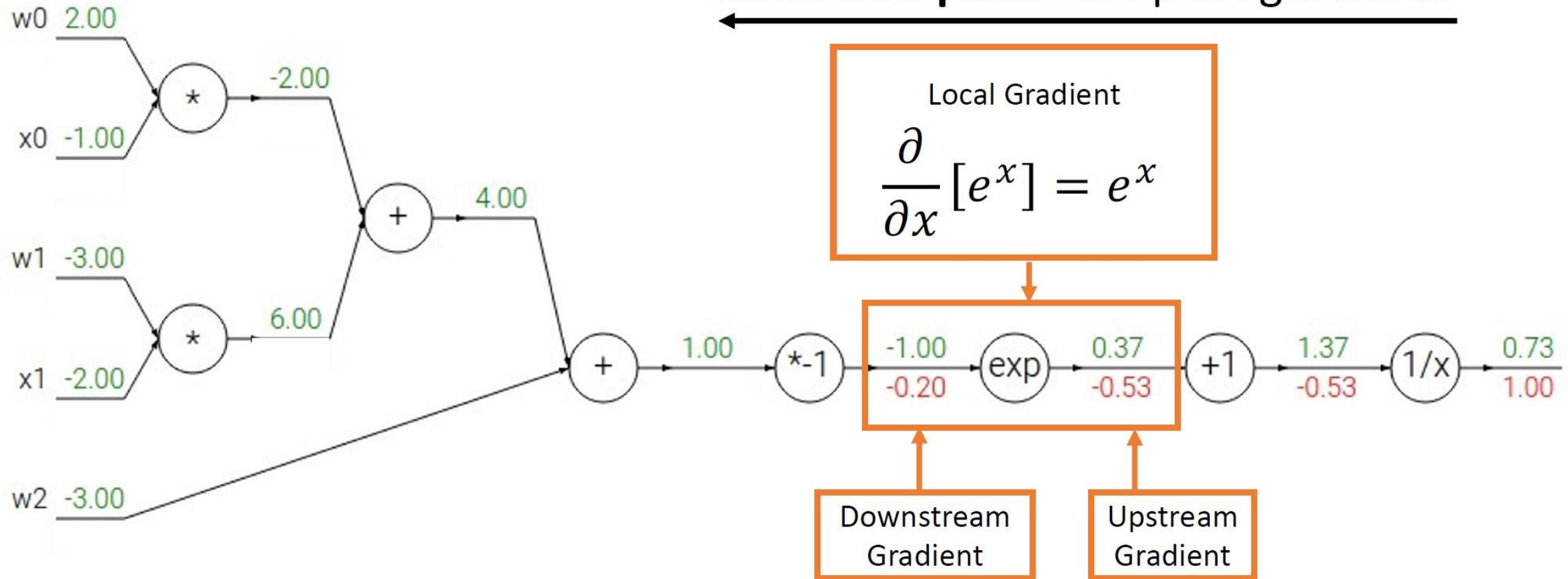
Backward pass: Compute gradients



Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

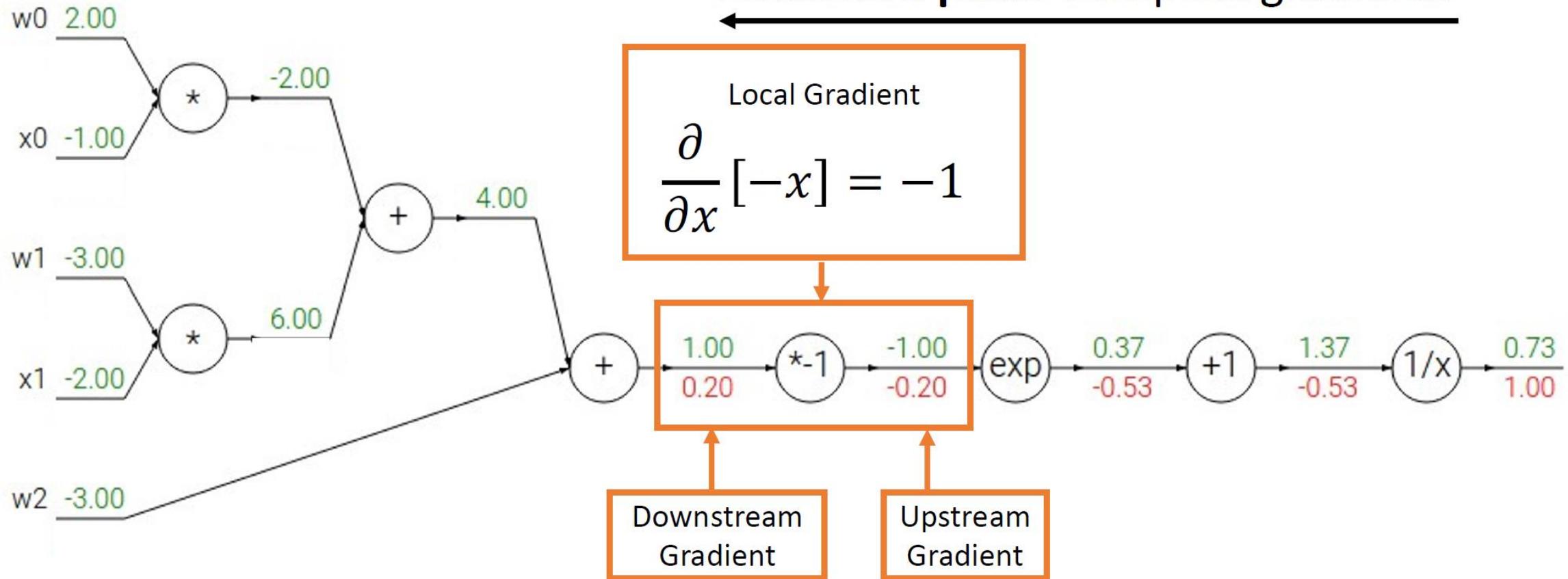
Backward pass: Compute gradients



Another Example

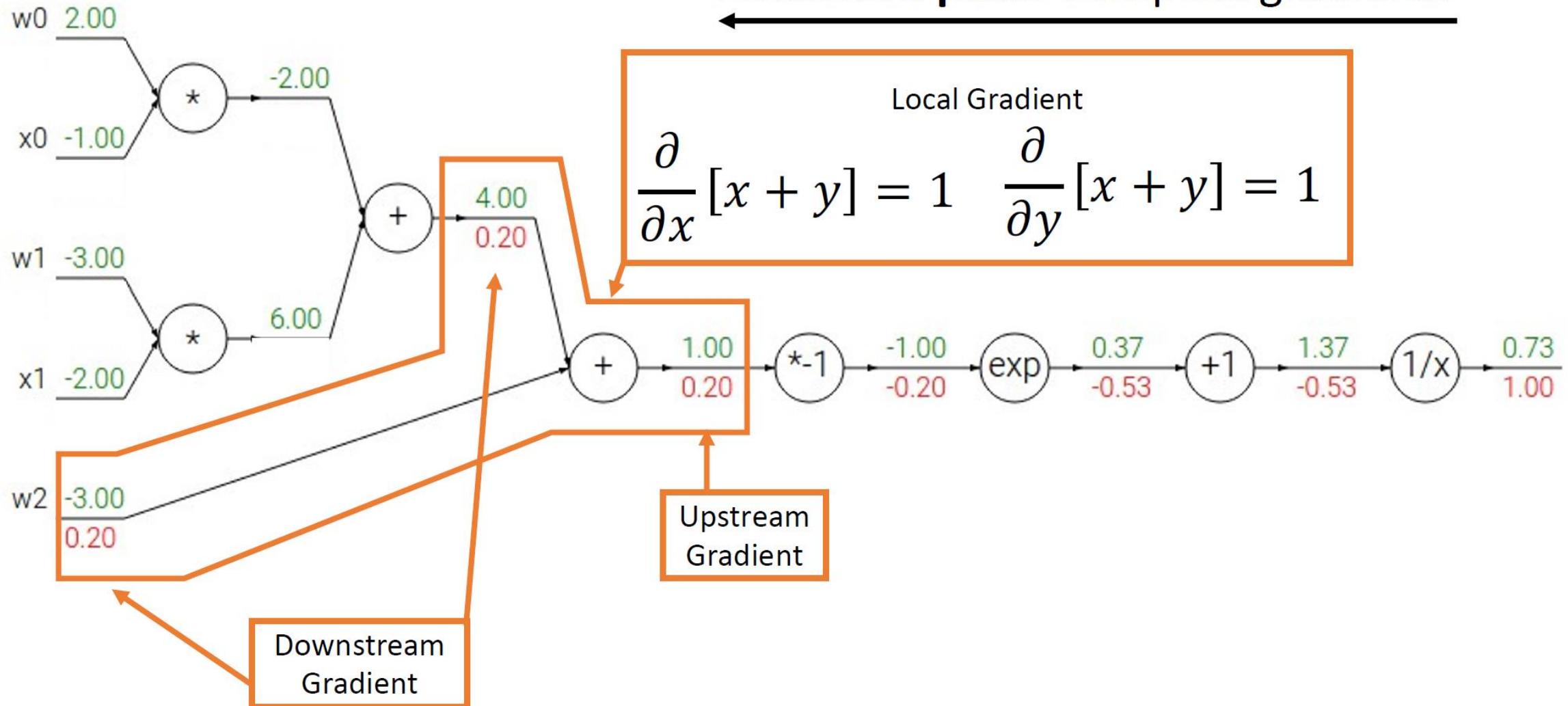
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

**Backward pass: Compute gradients**



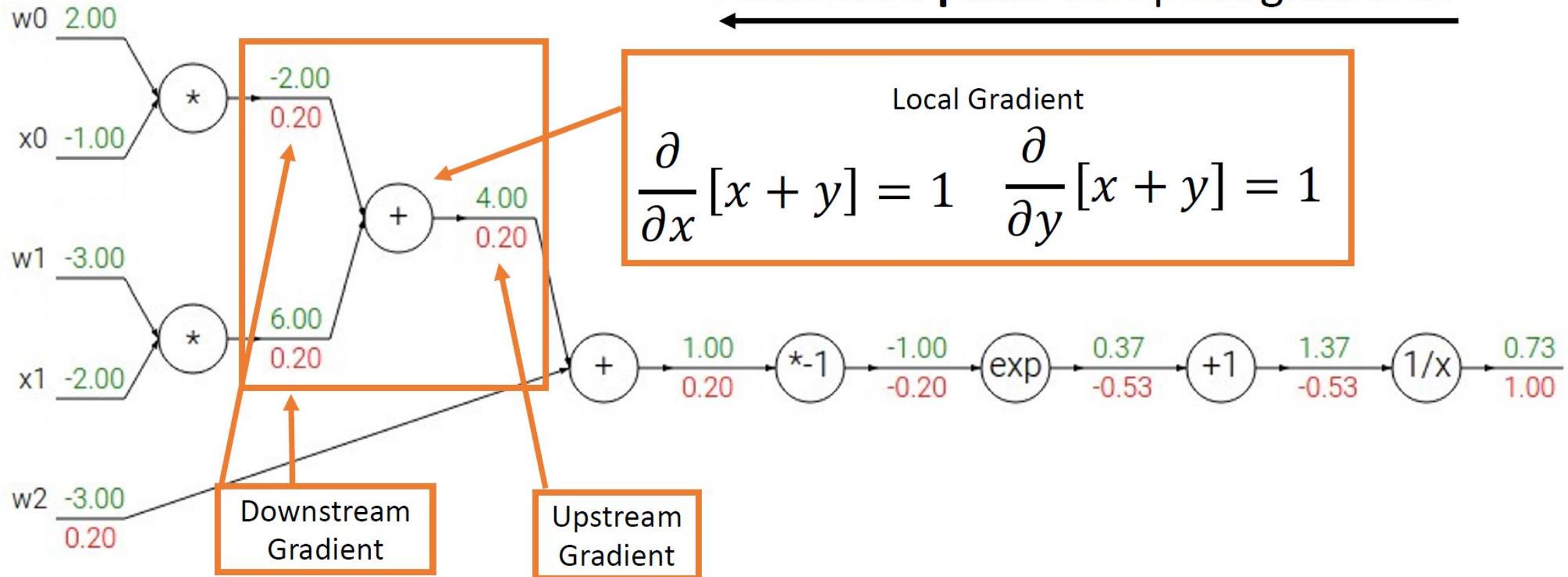
Another Example  $f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$

**Backward pass: Compute gradients**



Another Example  $f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$

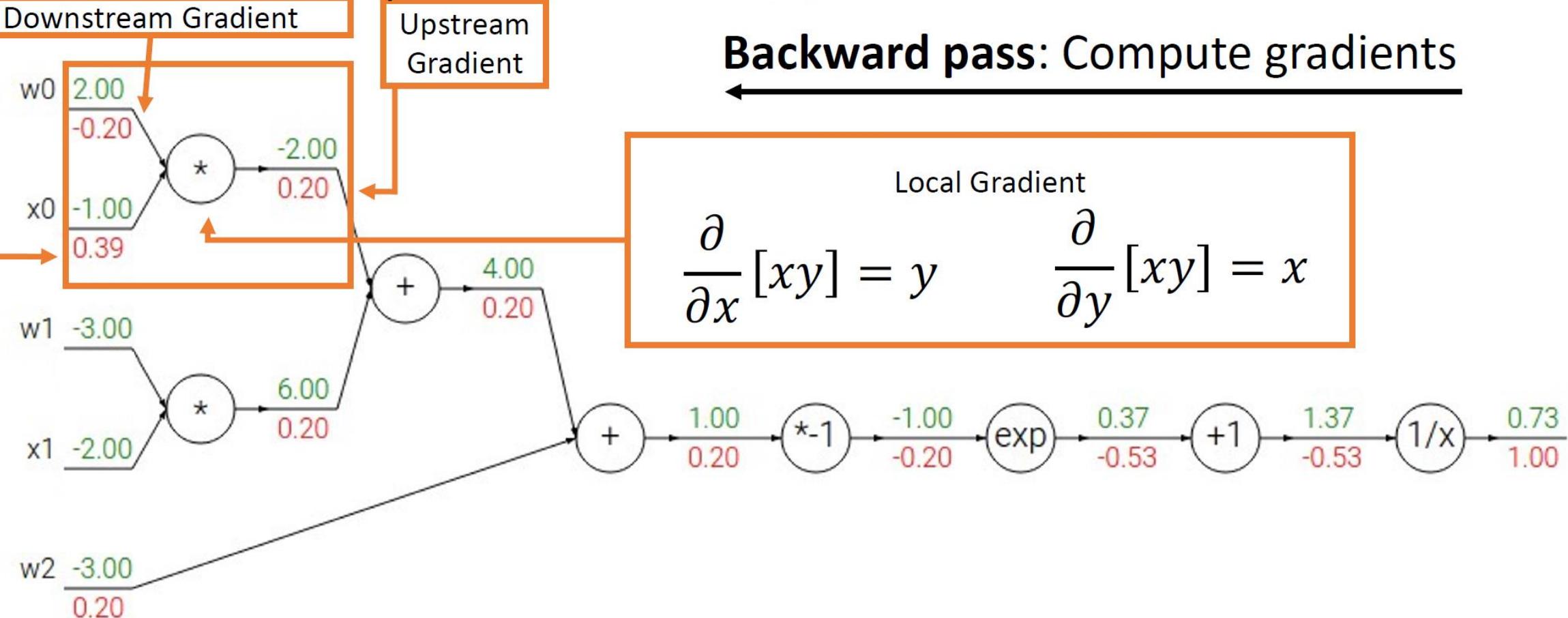
**Backward pass: Compute gradients**



# Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

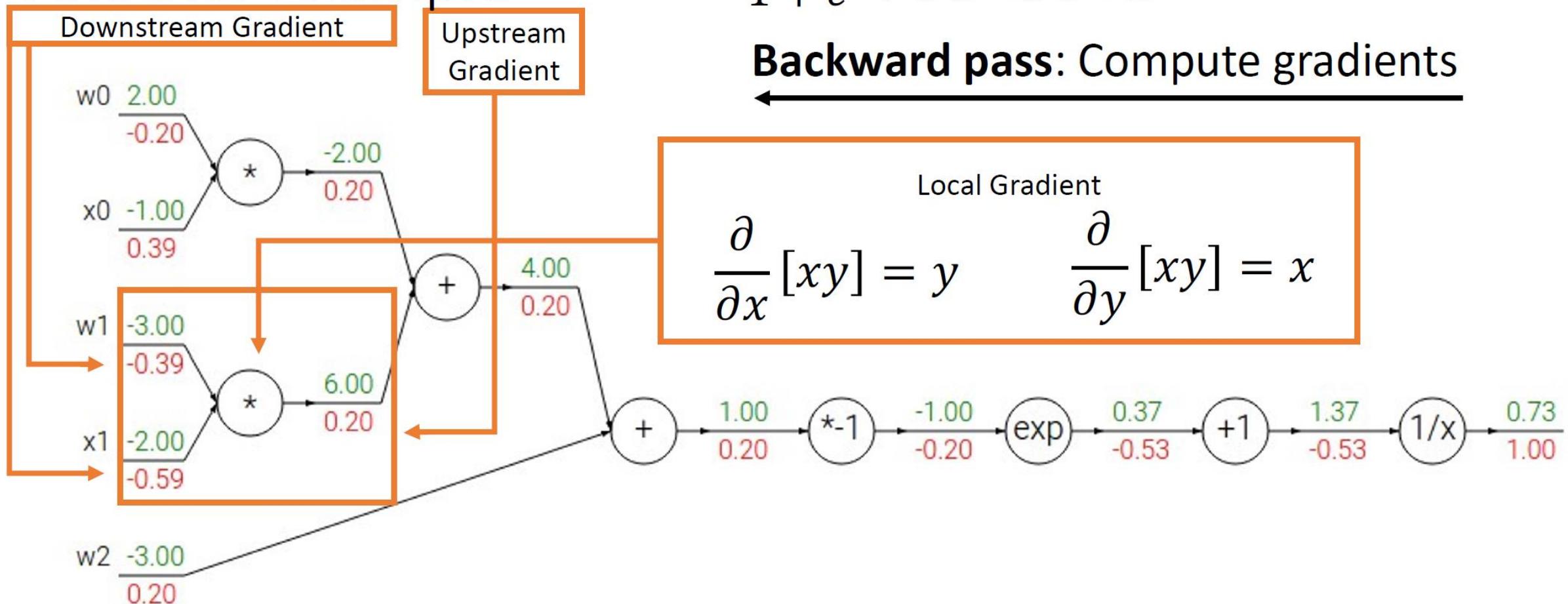
**Backward pass: Compute gradients**



# Another Example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

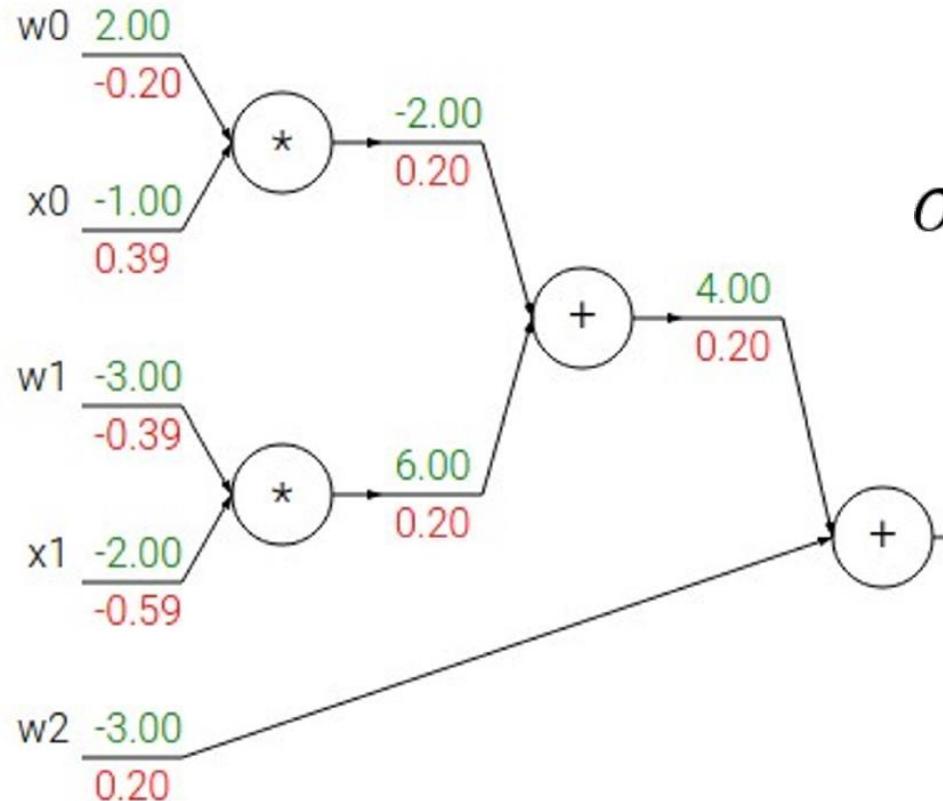
**Backward pass: Compute gradients**



Another Example

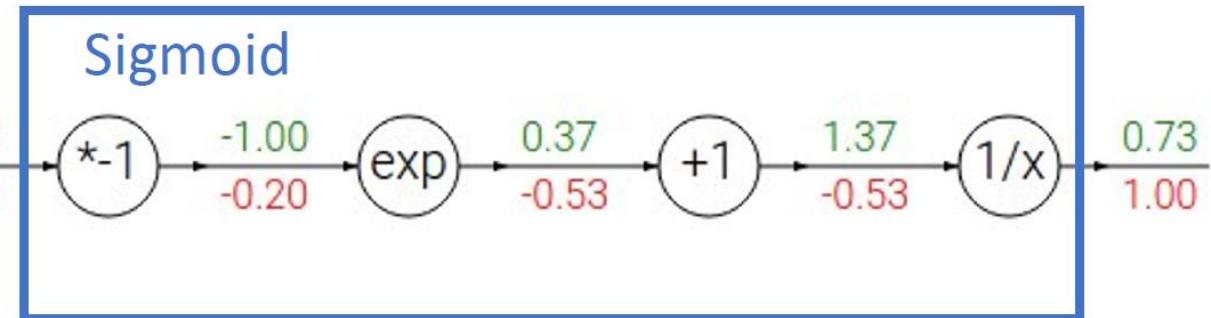
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} = \sigma(w_0x_0 + w_1x_1 + w_2)$$

**Backward pass: Compute gradients**



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

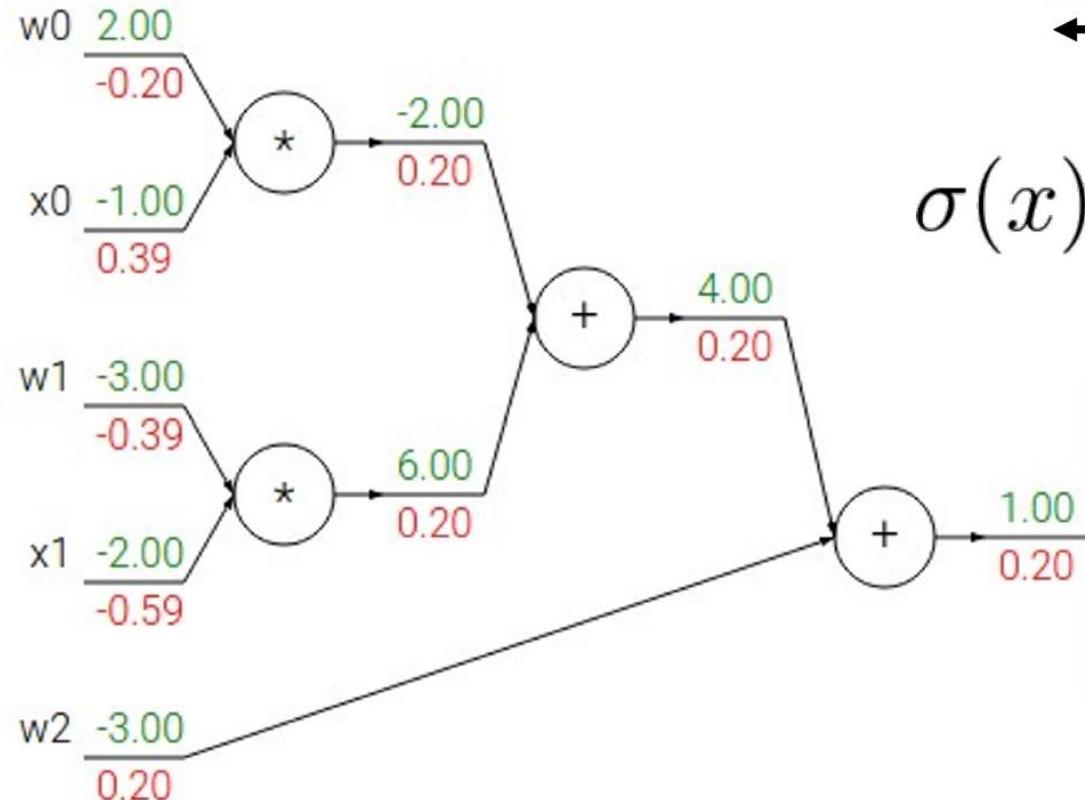
Computational graph is not unique: we can use primitives that have simple local gradients



Another Example

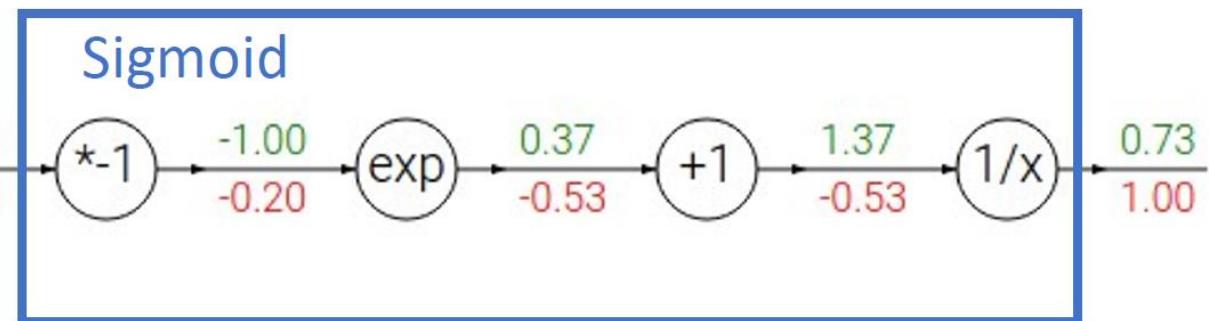
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} = \boxed{\sigma(w_0x_0 + w_1x_1 + w_2)}$$

**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



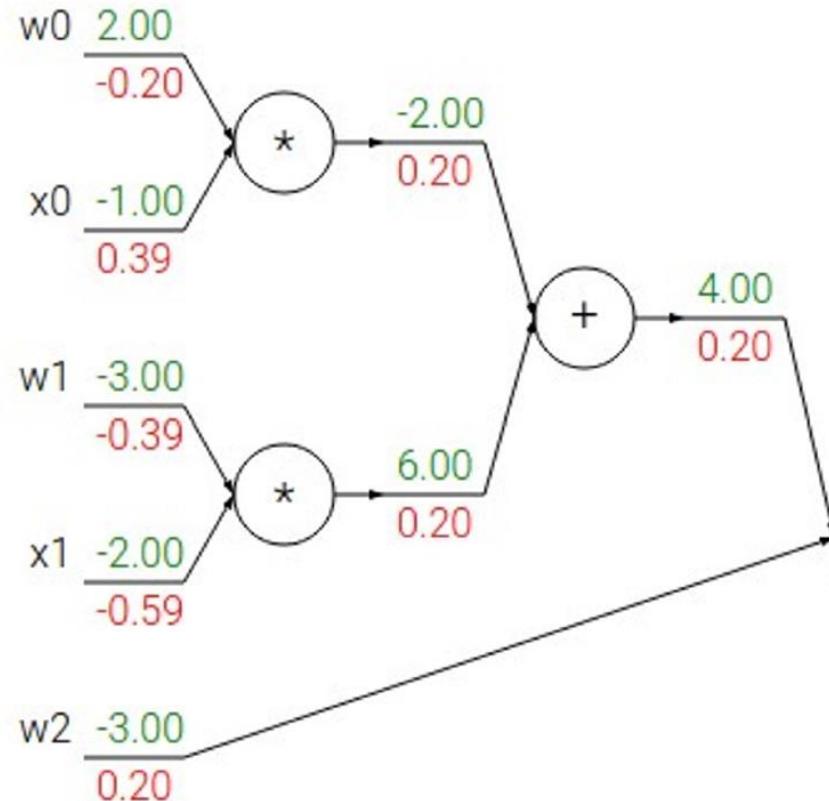
Sigmoid local gradient:

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

Another Example

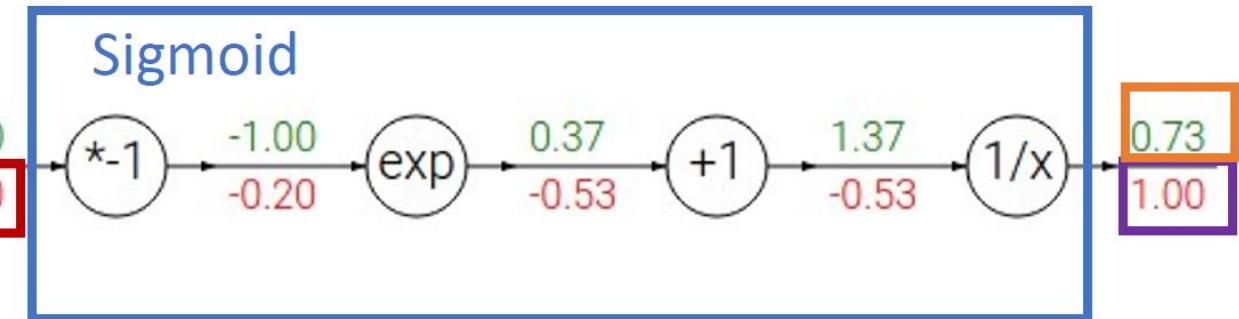
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} = \sigma(w_0x_0 + w_1x_1 + w_2)$$

**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



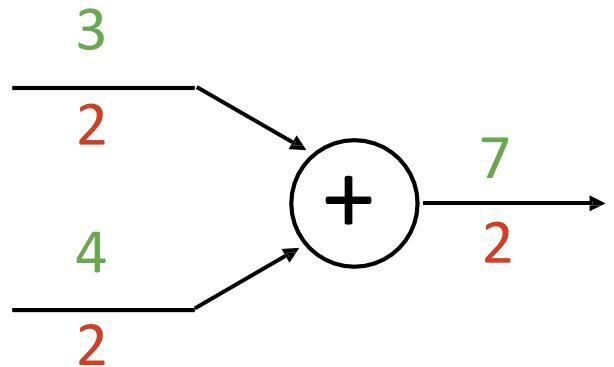
$$\begin{aligned} [\text{Downstream}] &= [\text{Local}] * [\text{Upstream}] \\ &= (1 - 0.73) * 0.73 * 1.0 = 0.2 \end{aligned}$$

Sigmoid local gradient:

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

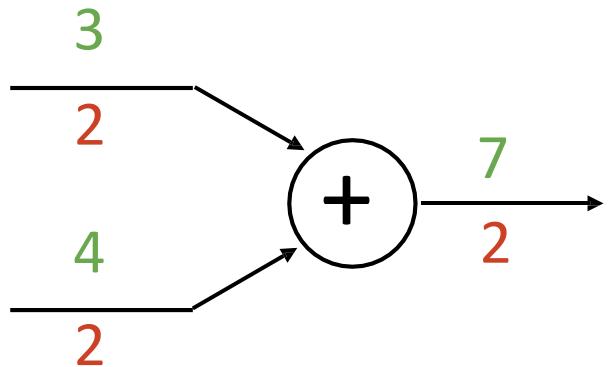
# Patterns in Gradient Flow

**add** gate: gradient distributor

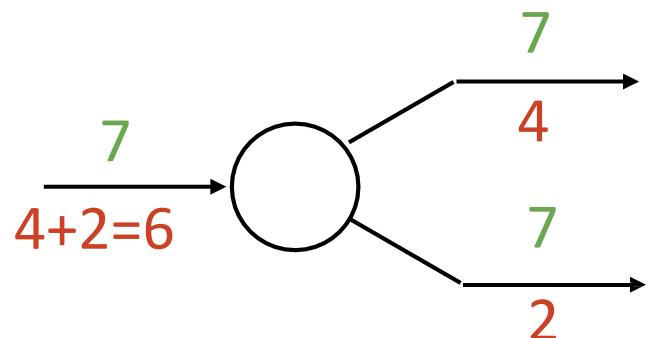


# Patterns in Gradient Flow

**add** gate: gradient distributor

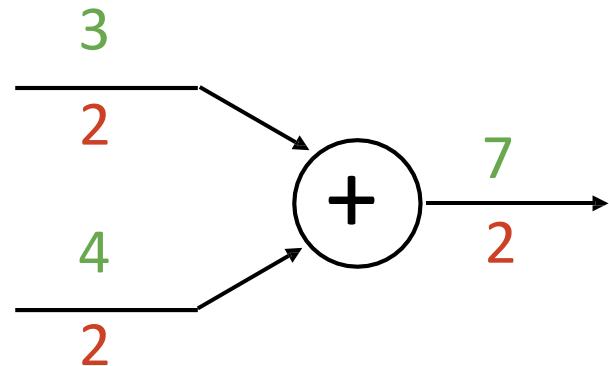


**copy** gate: gradient adder

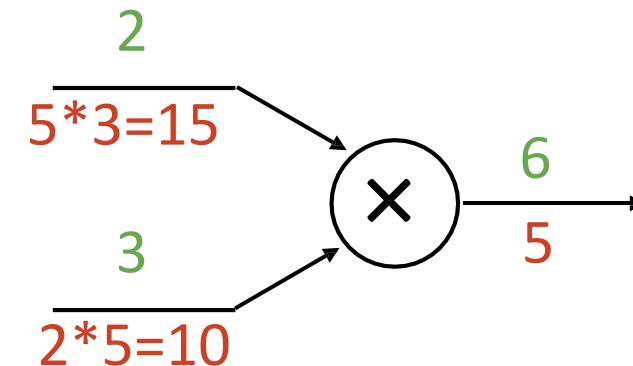


# Patterns in Gradient Flow

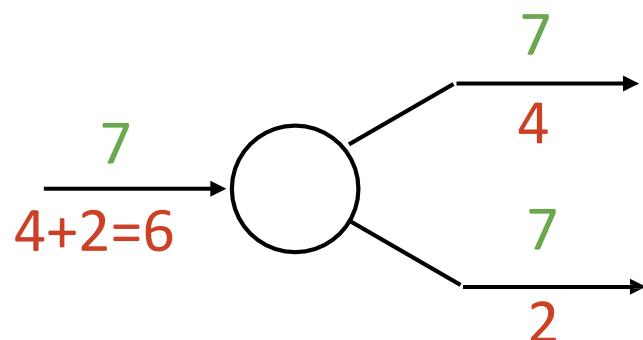
**add** gate: gradient distributor



**mul** gate: “swap multiplier”

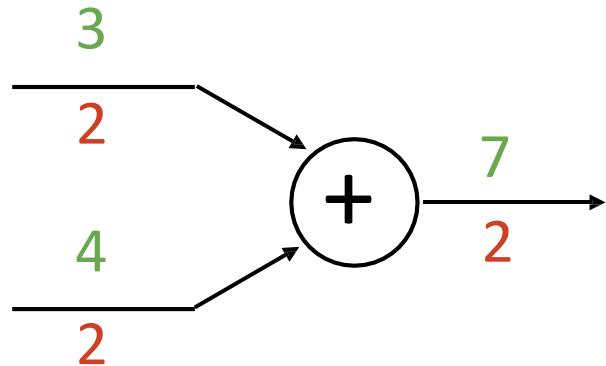


**copy** gate: gradient adder

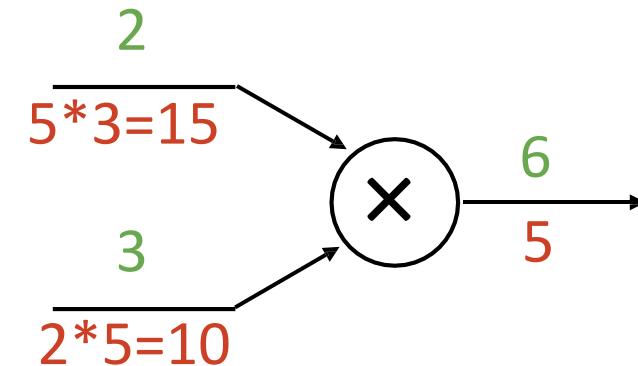


# Patterns in Gradient Flow

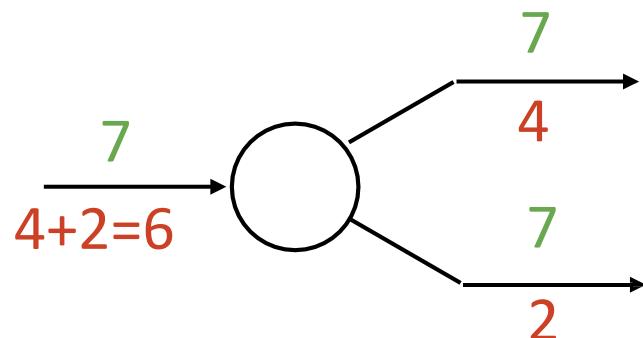
**add** gate: gradient distributor



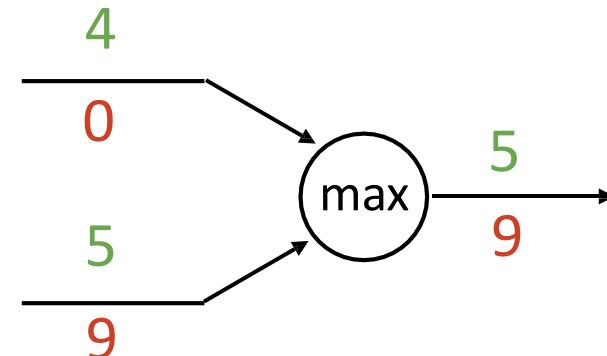
**mul** gate: “swap multiplier”



**copy** gate: gradient adder

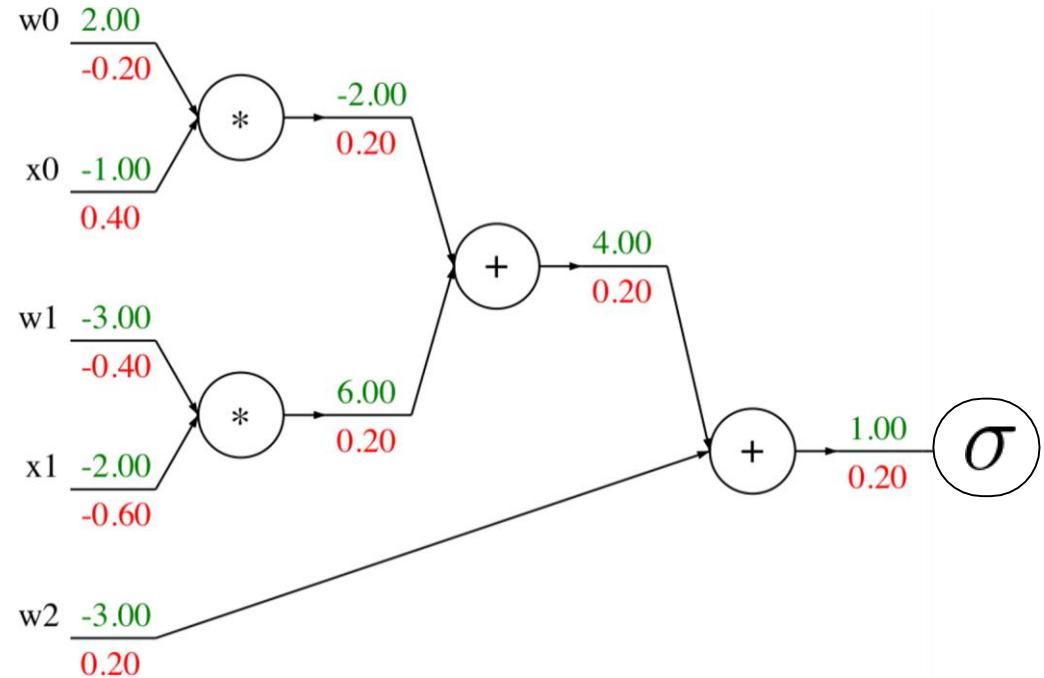


**max** gate: gradient router



# Backprop Implementation: "Flat" gradient code:

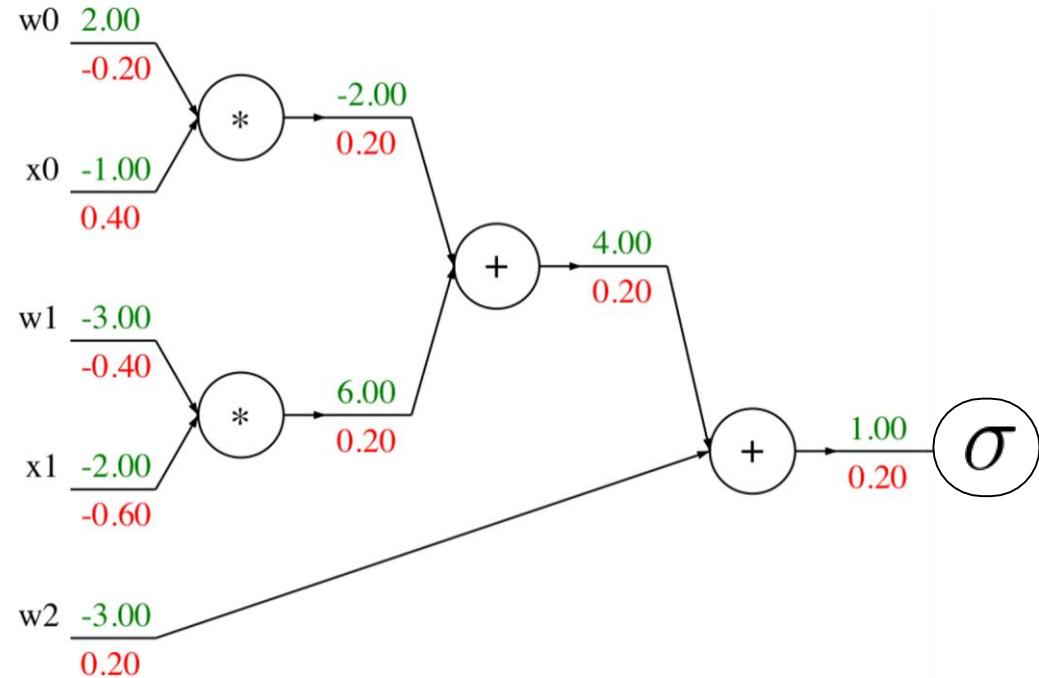
Forward pass:  
Compute output



```
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 output



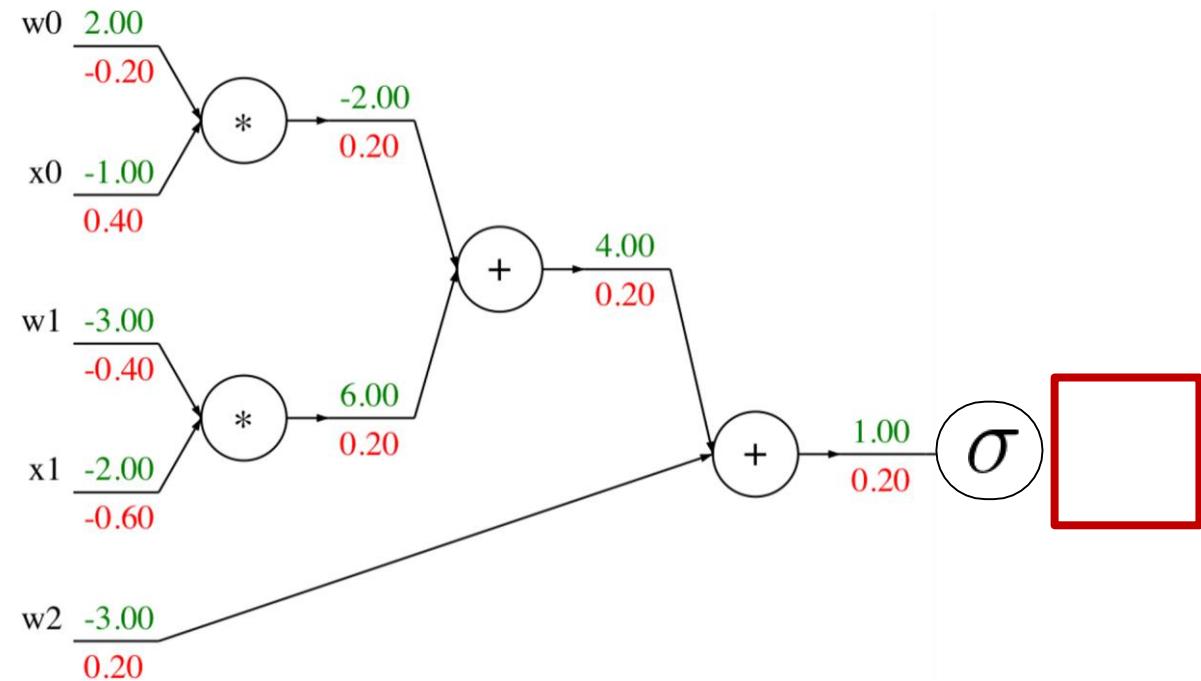
Backward pass:  
Compute grads

```
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 Implementation: "Flat" gradient code:

Forward pass:  
Compute output



Base case

```
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
```

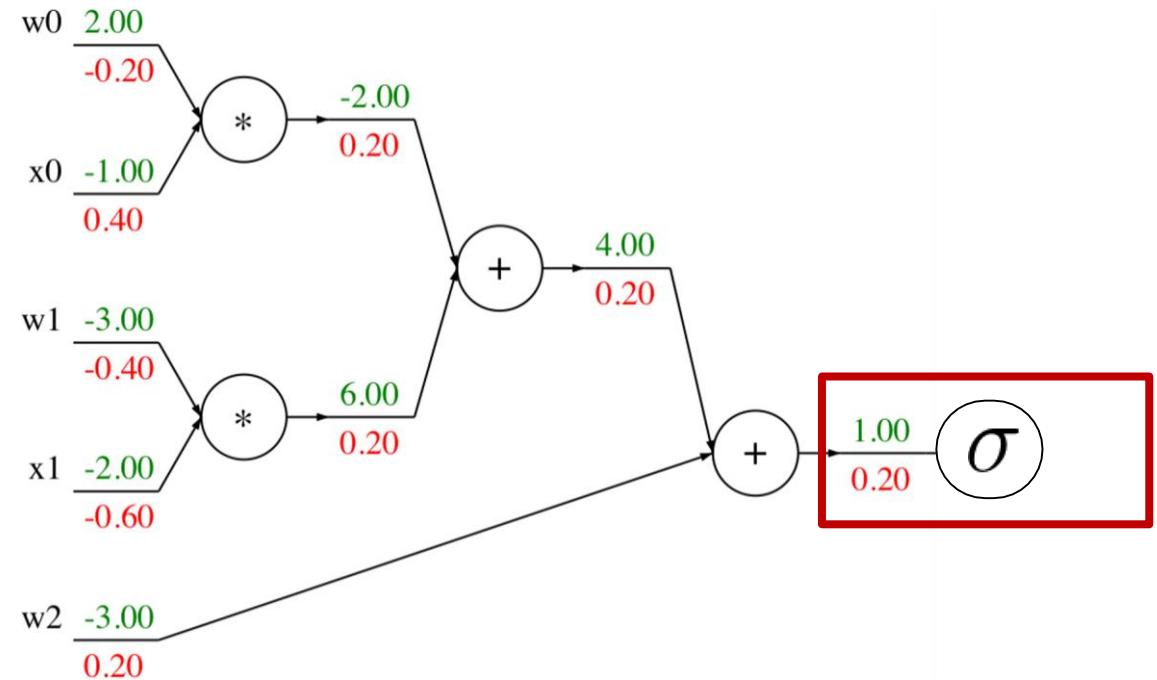
```
grad_x1 = grad_s1 * w1
```

```
grad_w0 = grad_s0 * x0
```

```
grad_x0 = grad_s0 * w0
```

# Backprop Implementation: "Flat" gradient code:

Forward pass:  
Compute output



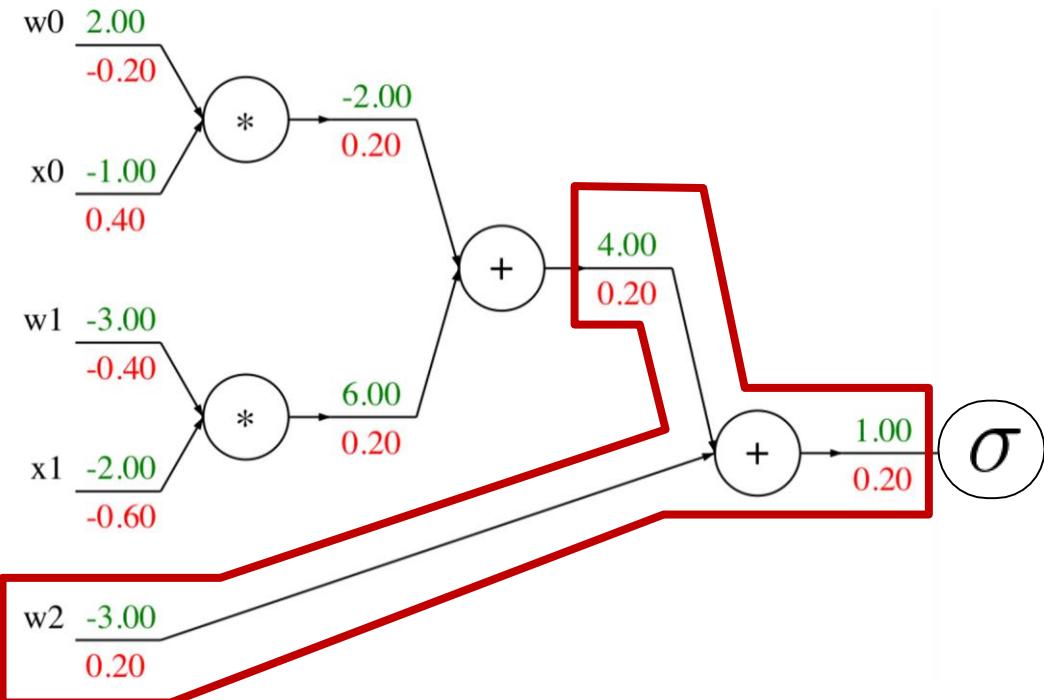
Sigmoid

```
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 Implementation: "Flat" gradient code:

Forward pass:  
Compute output



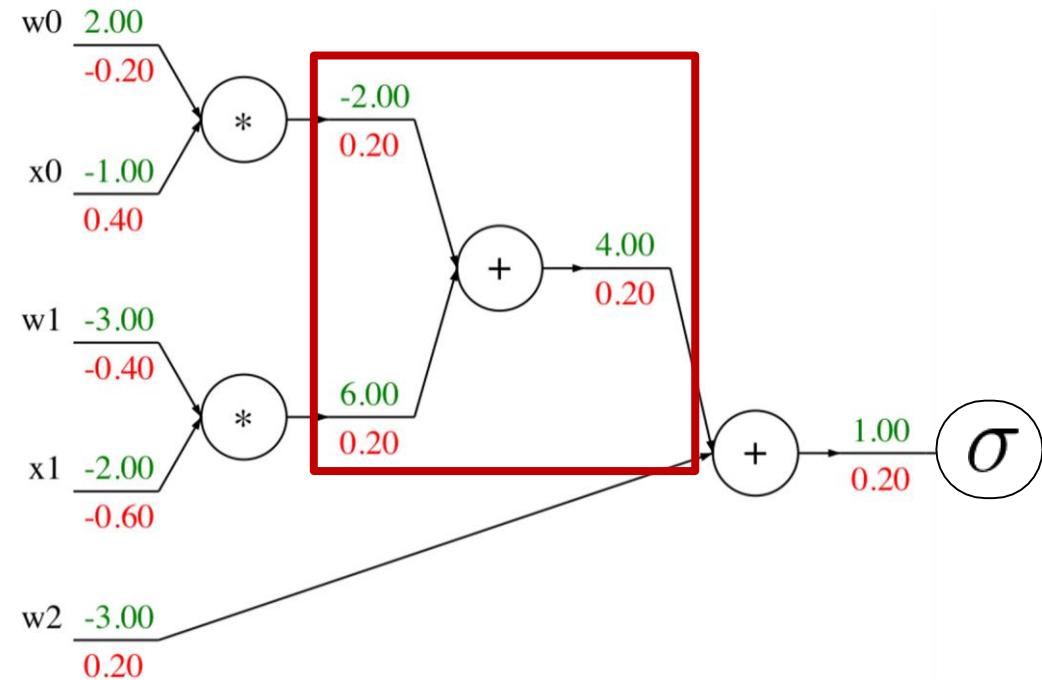
Add

```
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 Implementation: "Flat" gradient code:

Forward pass:  
Compute output



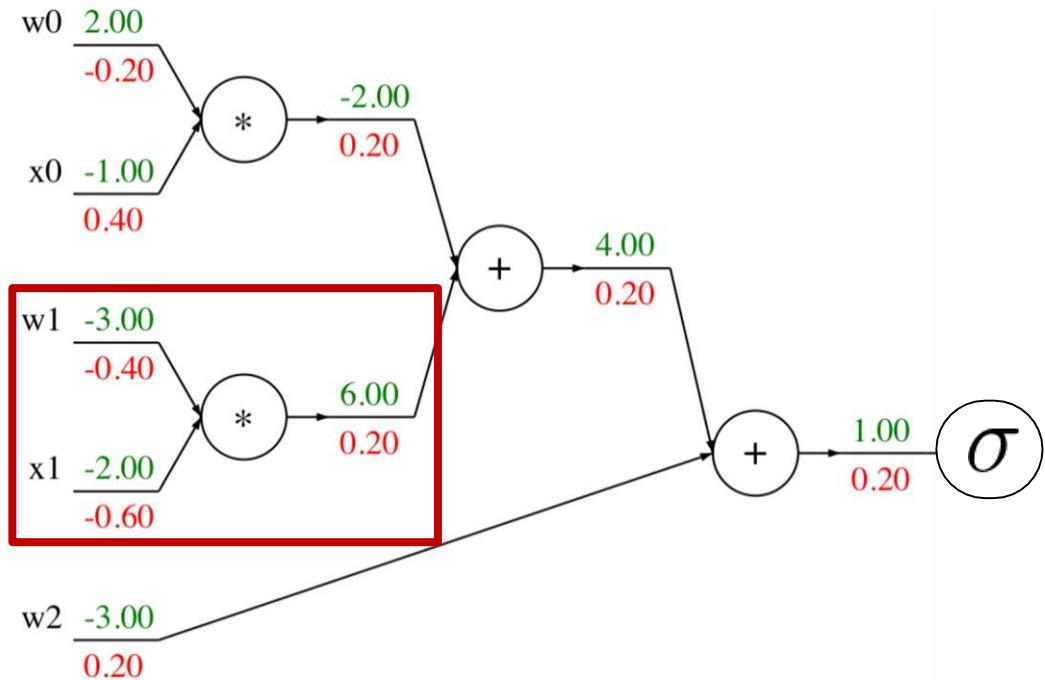
```
def f(w0, x0, w1, x1, w2):  
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

```
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
```

Add

# Backprop Implementation: "Flat" gradient code:

Forward pass:  
Compute output



```
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
```

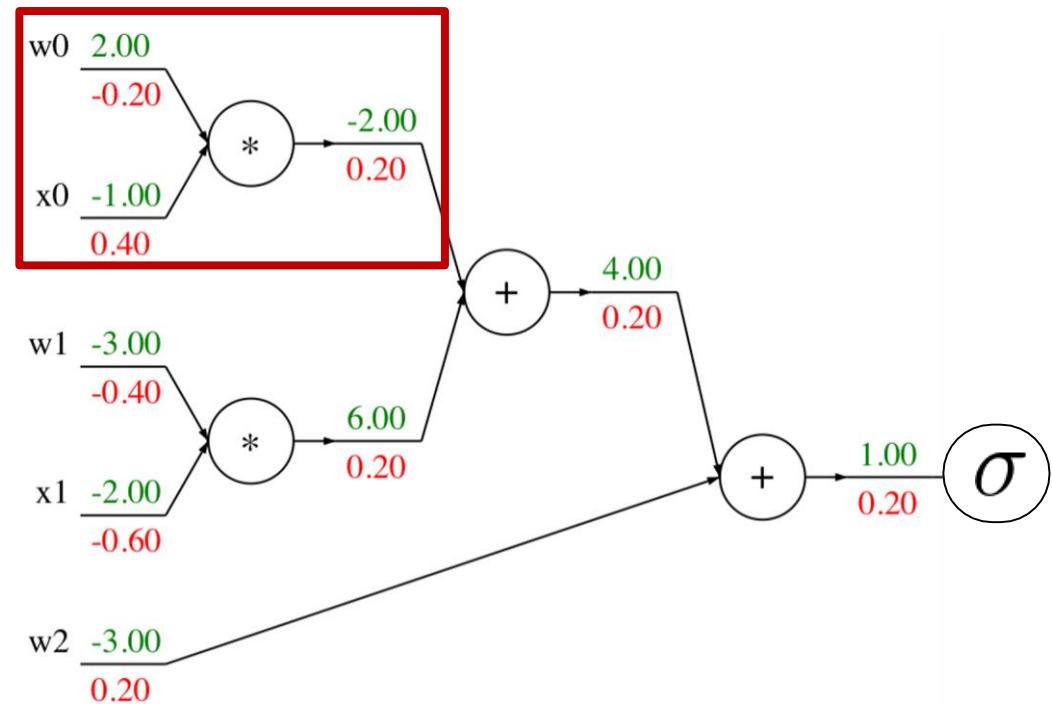
```
grad_w0 = grad_s0 * x0
```

```
grad_x0 = grad_s0 * w0
```

Multiply

# Backprop Implementation: "Flat" gradient code:

Forward pass:  
Compute output



Multiply

```
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
```

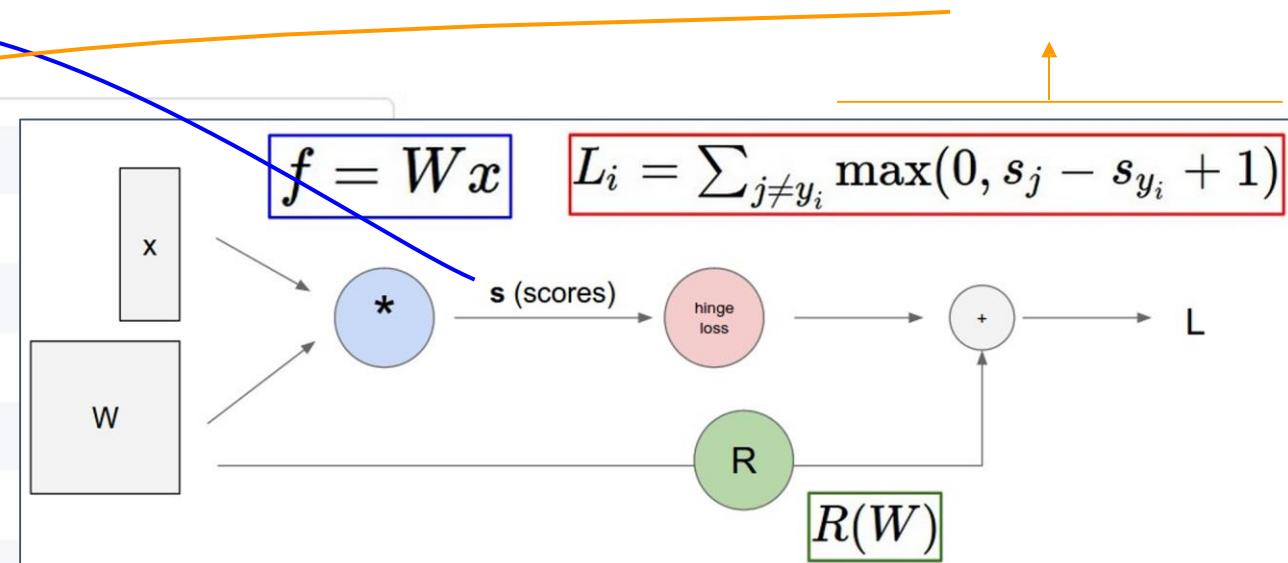
```
grad_x0 = grad_s0 * w0
```

# “Flat” Backprop: Done this for Homework 3!

Your gradient code should look like a “reversed version” of your forward pass!

E.g. for the SVM:

```
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = ...
margins = ...
data_loss = ...
reg_loss = ...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = ... (optionally, we go direct to dscores)
dscores = ...
dW = ...
```



# “Flat” Backprop: Do this for Homework 5!

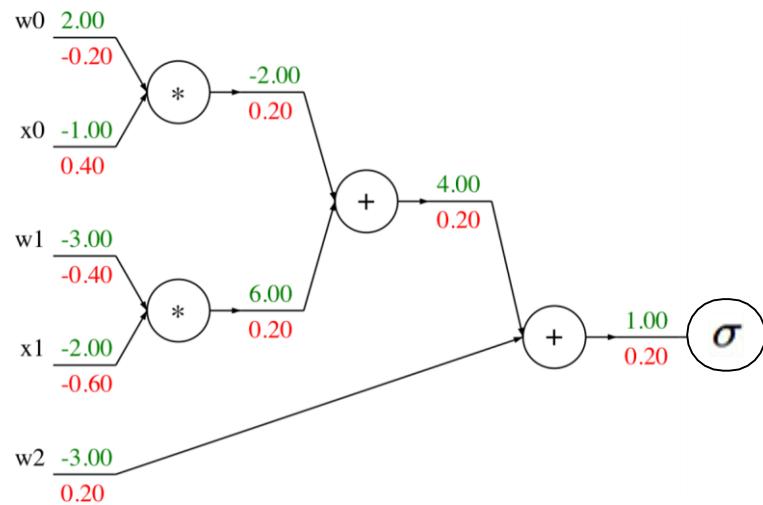
Your gradient code should look like a “reversed version” of your forward pass!

E.g. for two-layer neural net:

```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```

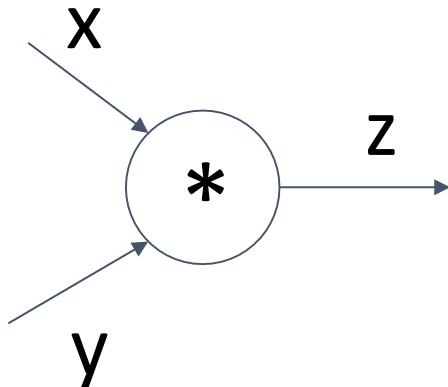
# Backprop Implementation: Modular API

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) ← Need to stash some values for use in backward  
        z = x * y  
        return z  
  
    @staticmethod  
    def backward(ctx, grad_z): ← Upstream gradient  
        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

# Example: PyTorch operators

pytorch / pytorch		
Code Issues Pull requests Projects Wiki Insights		
Tree: 517c7c9861 ▾	pytorch / aten / src / THNN / generic /	Create new file Upload files Find file History
	ezyang and facebook-github-bot Canonicalize all includes in PyTorch. (#14849) ...	Latest commit 517c7c9 on Dec 8, 2018
..		
<a href="#">AbsCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">BCECriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">ClassNLLCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Col2Im.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">ELU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">FeatureLPPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">GatedLinearUnit.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">HardTanh.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Im2Col.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">IndexLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">LeakyReLU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">LogSigmoid.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MSECriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MultiLabelMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MultiMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">RReLU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Sigmoid.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SmoothL1Criterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftPlus.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftShrink.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SparseLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAdaptiveAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAdaptiveMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialClassNLLCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialConvolutionMM.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialDilatedMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialFractionalMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialFullDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialMaxUnpooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialReflectionPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialUpSamplingBilinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">THNN.h</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Tanh.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalReflectionPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalRowConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalUpSamplingLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAdaptiveAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAdaptiveMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricConvolutionMM.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricDilatedMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricFractionalMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricFullDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricMaxUnpooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricUpSamplingTrilinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">linear_upsampling.h</a>	Implement nn.functional.interpolate based on upsample. (#8591)	9 months ago
<a href="#">pooling_shape.h</a>	Use integer math to compute output size of pooling operations (#14405)	4 months ago
<a href="#">unfold.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19    THNN_CHECK_NELEMENT(output, gradOutput);
20    THTensor_(resizeAs)(gradInput, output);
21    TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22                      scalar_t z = *output_data;
23                      *gradInput_data = *gradOutput_data * (1. - z) * z;
24    );
25 }
26
27 #endif
```

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19    THNN_CHECK_NELEMENT(output, gradOutput);
20    THTensor_(resizeAs)(gradInput, output);
21    TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22        scalar_t z = *output_data;
23        *gradInput_data = *gradOutput_data * (1. - z) * z;
24    );
25 }
26
27#endif
```

Forward

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

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19    THNN_CHECK_NELEMENT(output, gradOutput);
20    THTensor_(resizeAs)(gradInput, output);
21    TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22        scalar_t z = *output_data;
23        *gradInput_data = *gradOutput_data * (1. - z) * z;
24    );
25 }
26
27#endif
```

Forward

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

```
static void sigmoid_kernel(TensorIterator& iter) {
    AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
        unary_kernel_vec(
            iter,
            [=](scalar_t a) -> scalar_t {return (1 / (1 + std::exp((-a)))); },
            [=](Vec256<scalar_t> a) {
                a = Vec256<scalar_t>((scalar_t)(0)) - a;
                a = a.exp();
                a = Vec256<scalar_t>((scalar_t)(1)) + a;
                a = a.reciprocal();
                return a;
            });
    });
}
```

Forward actually defined [elsewhere](#)...

`return (1 / (1 + std::exp((-a))));`

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19     THNN_CHECK_NELEMENT(output, gradOutput);
20     THTensor_(resizeAs)(gradInput, output);
21     TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22         scalar_t z = *output_data;
23         *gradInput_data = *gradOutput_data * (1. - z) * z;
24     );
25 }
26
27#endif
```

Forward

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

Backward

$$(1 - \sigma(x)) \sigma'(x)$$

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**:

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

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**:

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

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

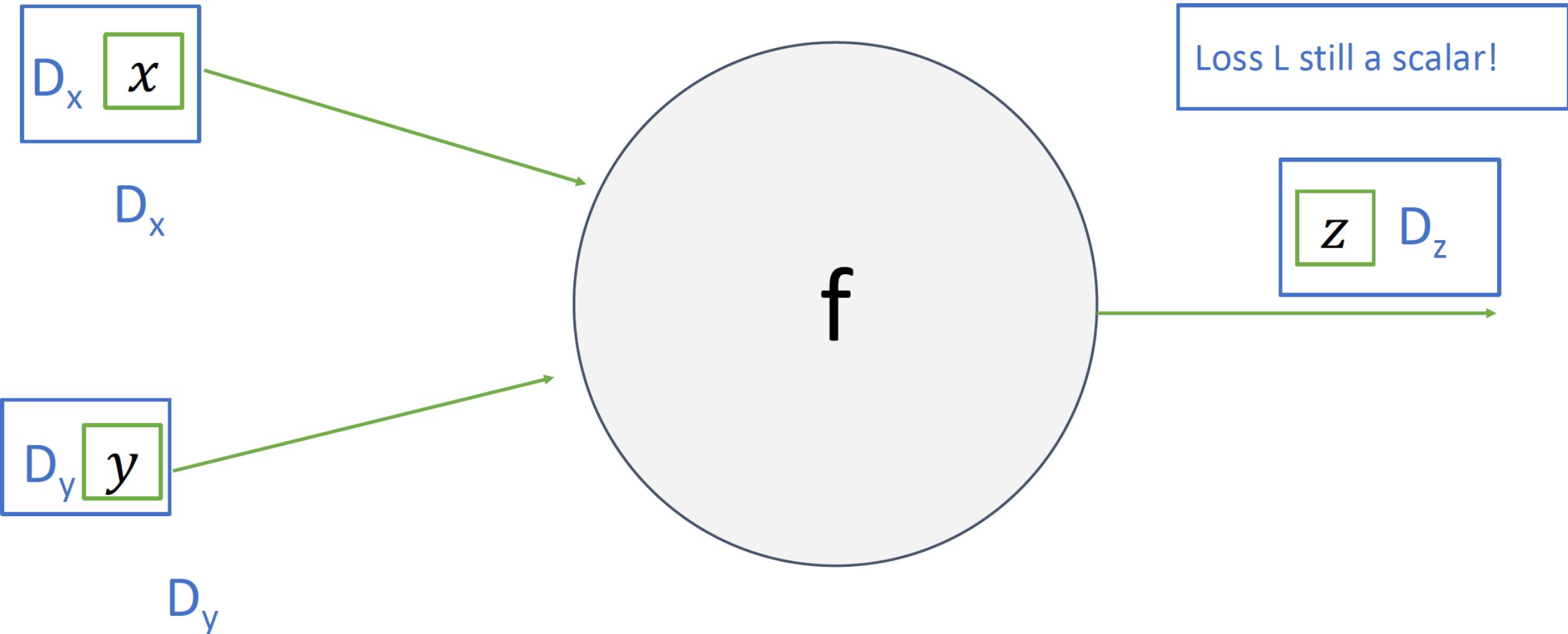
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

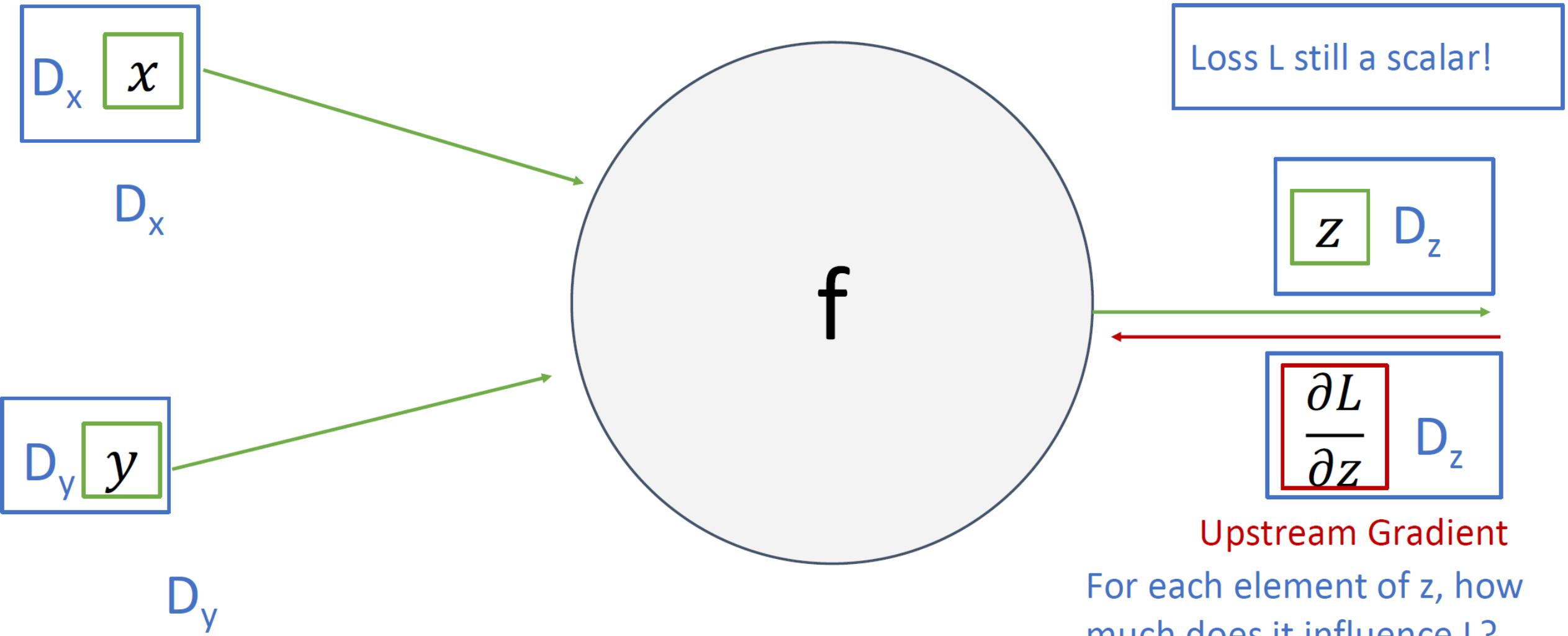
$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^{N \times M} \\ \left(\frac{\partial y}{\partial x}\right)_{i,j} &= \frac{\partial y_j}{\partial x_i}\end{aligned}$$

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

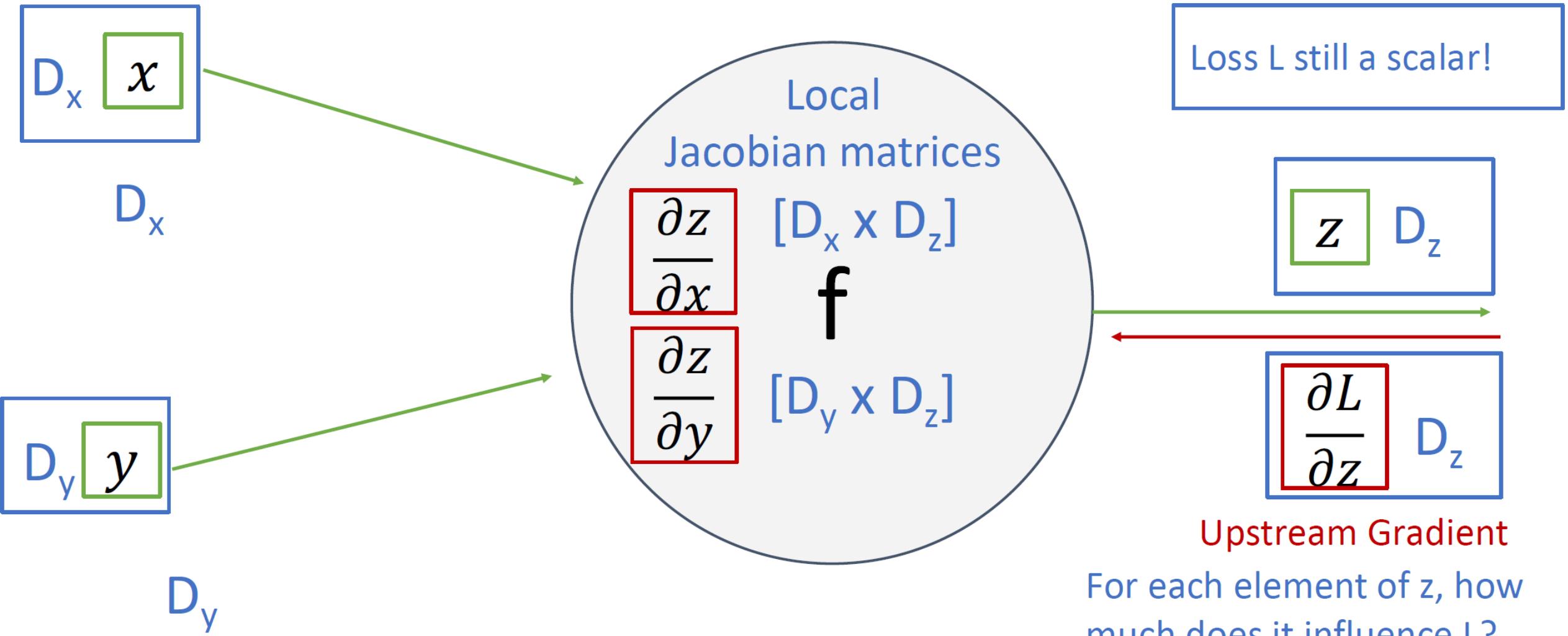
# Backprop with Vectors



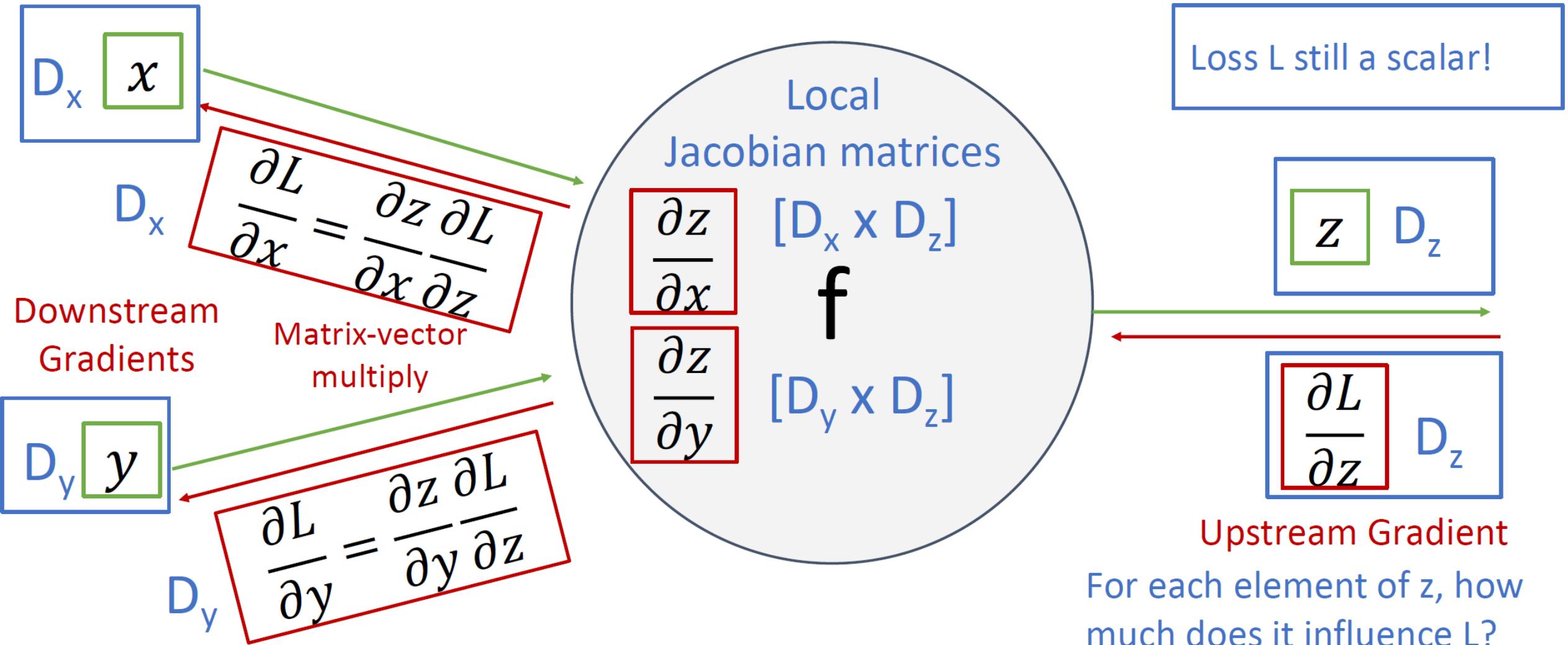
# Backprop with Vectors



# Backprop with Vectors



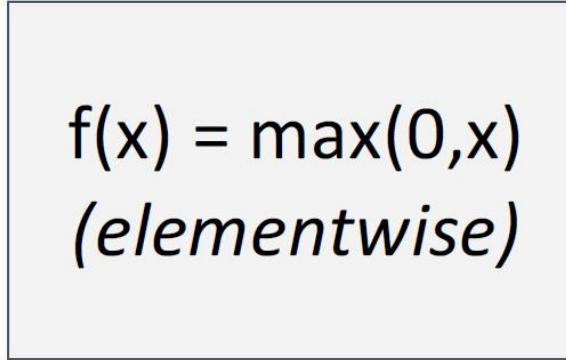
# Backprop with Vectors



# Backprop with Vectors

4D input x:

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



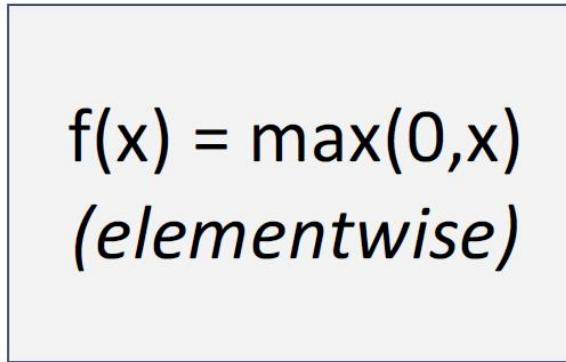
4D output y:

$$\begin{array}{l} \longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix} \\ \longrightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 3 \end{bmatrix} \\ \longrightarrow \begin{bmatrix} 3 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ \longrightarrow \begin{bmatrix} 0 \\ 3 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

# Backprop with Vectors

4D input x:

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



4D output y:

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

4D  $dL/dy$ :

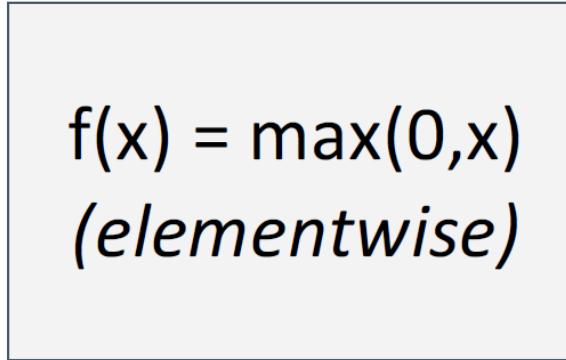
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \longleftarrow$$

Upstream  
gradient

# Backprop with Vectors

4D input x:

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



4D output y:

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

Jacobian  $\frac{dy}{dx}$

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

4D  $\frac{dL}{dy}$ :

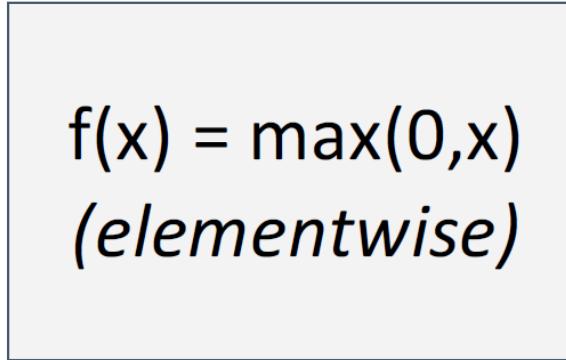
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \xleftarrow{\quad}$$

Upstream  
gradient

# Backprop with Vectors

4D input x:

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



4D output y:

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

$$\begin{bmatrix} dy/dx \\ dL/dy \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 4 & -1 & 5 & 9 \end{bmatrix}$$

4D  $dL/dy$ :

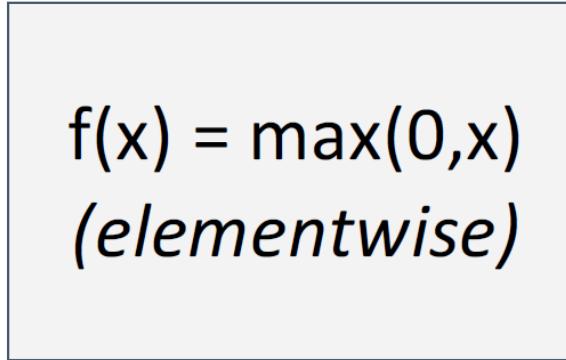
$$\begin{array}{c} \leftarrow \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow \\ \leftarrow \begin{bmatrix} -1 \\ 5 \end{bmatrix} \leftarrow \\ \leftarrow \begin{bmatrix} 5 \end{bmatrix} \leftarrow \\ \leftarrow \begin{bmatrix} 9 \end{bmatrix} \leftarrow \end{array}$$

Upstream  
gradient

# Backprop with Vectors

4D input x:

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



4D output y:

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

4D  $dL/dx$ :

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

$[dy/dx] [dL/dy]$

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

4D  $dL/dy$ :

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

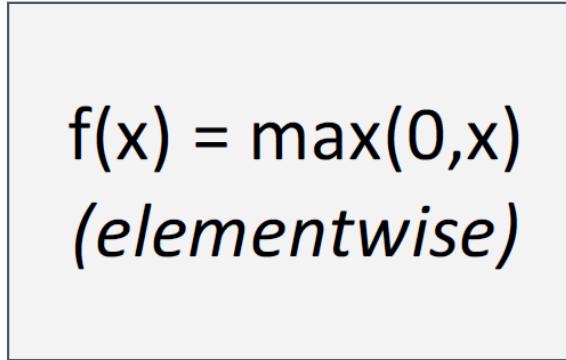
Upstream  
gradient

# Backprop with Vectors

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

4D input  $x$ :

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



4D output  $y$ :

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

4D  $dL/dx$ :

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

$[dy/dx] [dL/dy]$

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

4D  $dL/dy$ :

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

Upstream  
gradient

# Backprop with Vectors

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

4D input  $x$ :

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

$$f(x) = \max(0, x)$$

(elementwise)

4D output  $y$ :

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

4D  $dL/dx$ :

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

$[dy/dx] [dL/dy]$

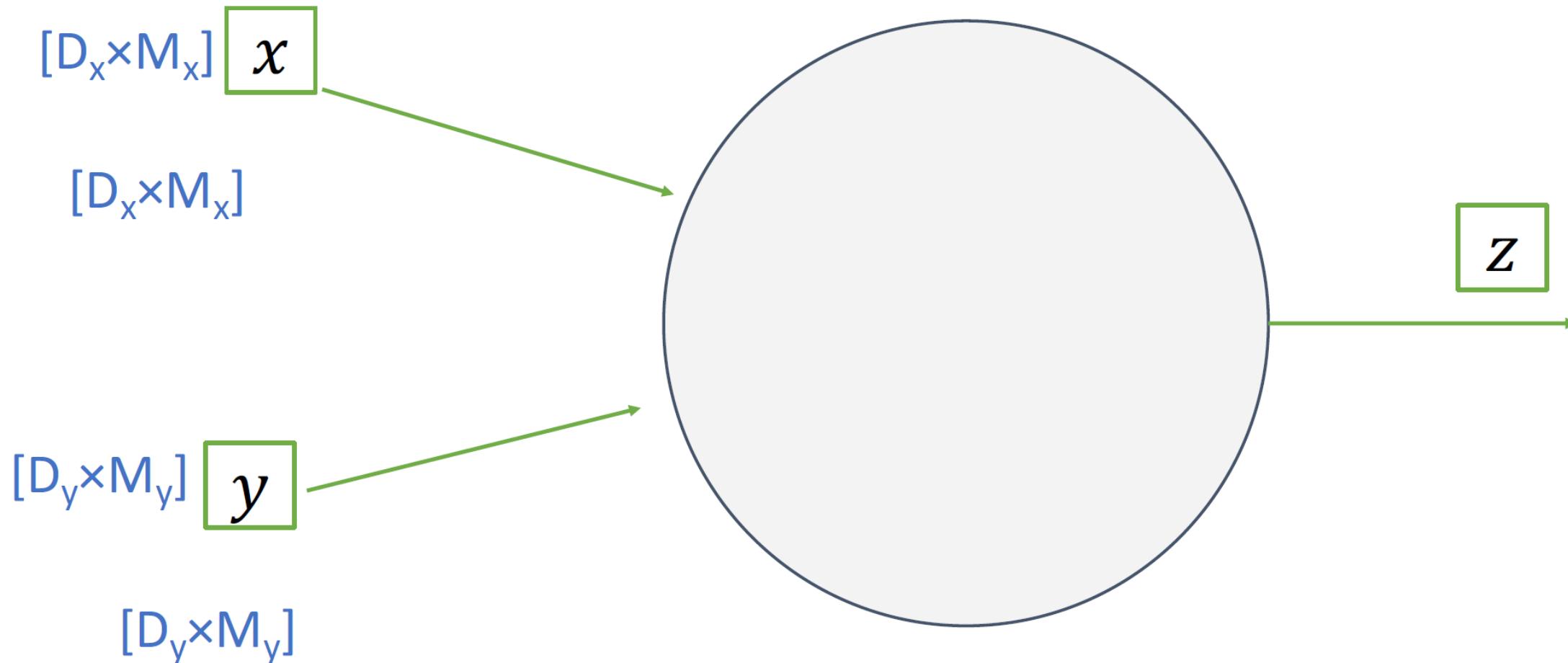
$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial y}\right)_i, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

4D  $dL/dy$ :

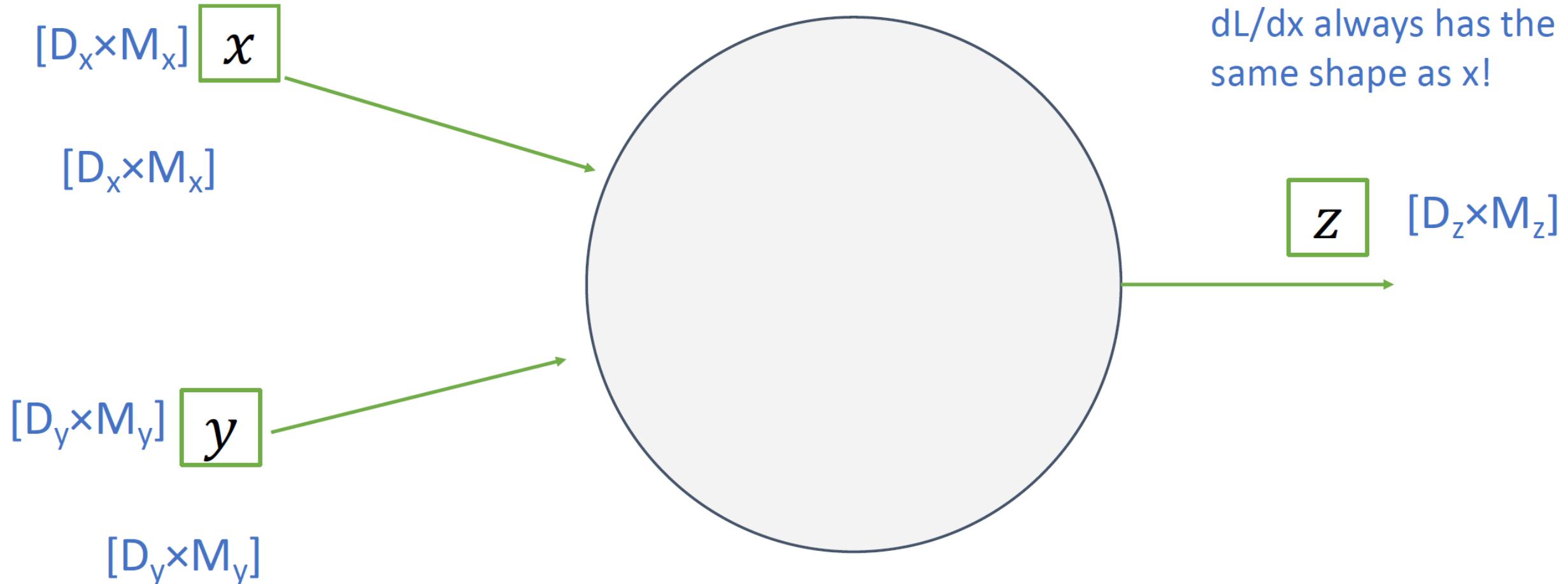
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow$$

Upstream  
gradient

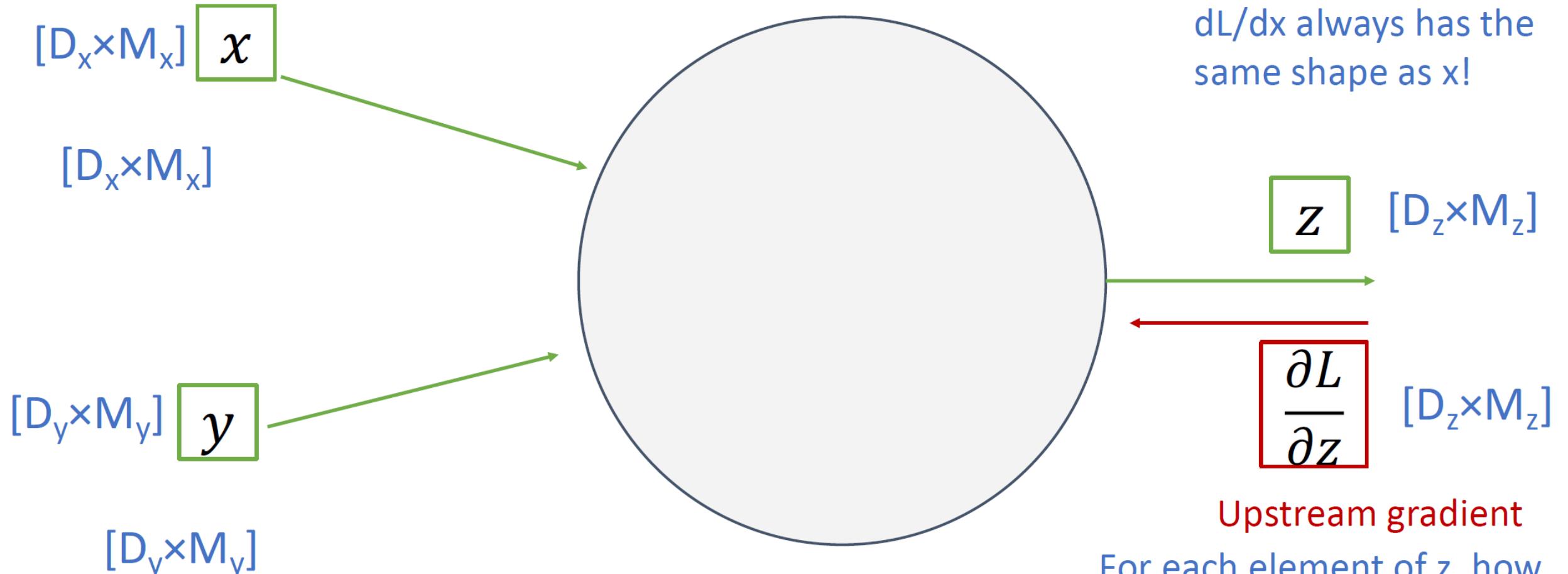
# Backprop with Matrices (or Tensors):



# Backprop with Matrices (or Tensors):



# Backprop with Matrices (or Tensors):



Loss  $L$  still a scalar!

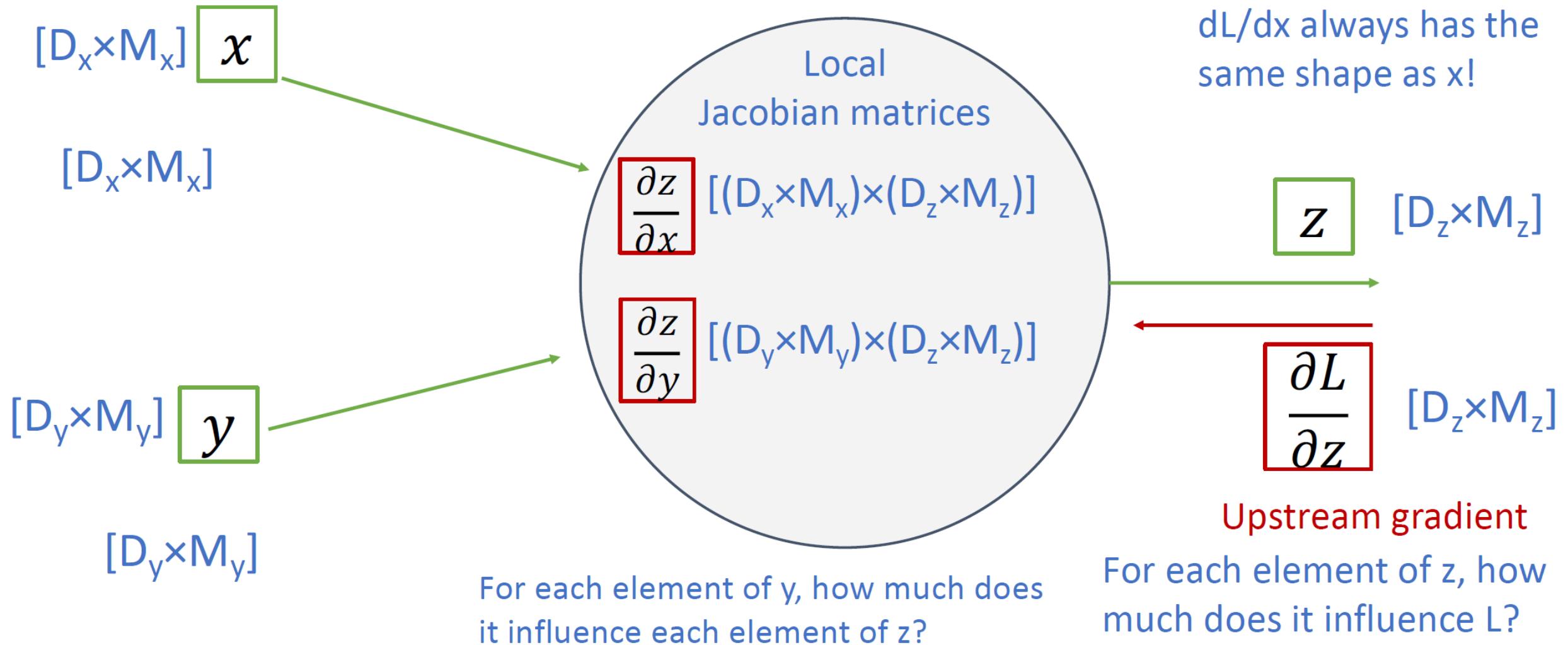
$dL/dx$  always has the same shape as  $x$ !

$$\frac{\partial L}{\partial z} \quad [D_z \times M_z]$$

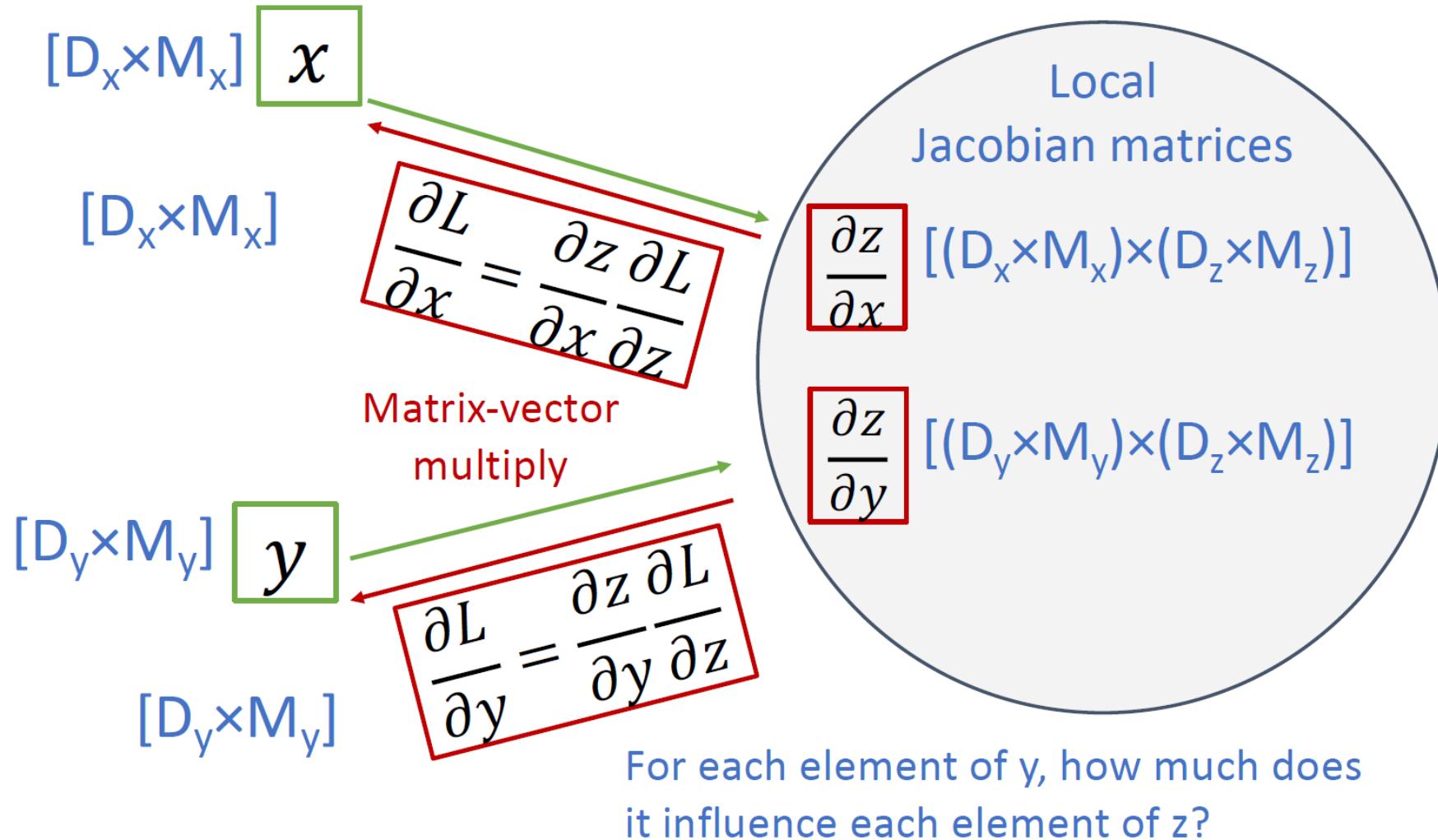
Upstream gradient

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

# Backprop with Matrices (or Tensors):



# Backprop with Matrices (or Tensors):



Loss  $L$  still a scalar!

$dL/dx$  always has the same shape as  $x$ !

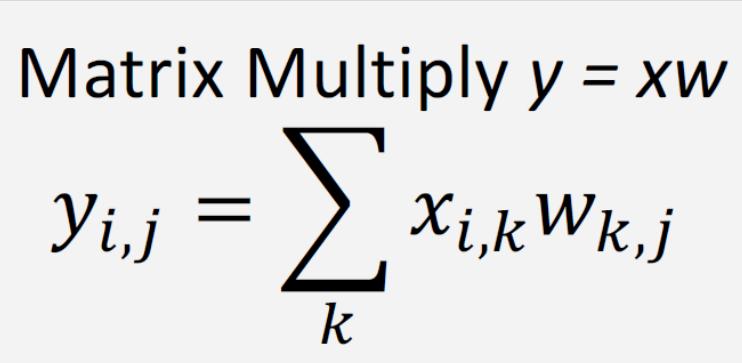
# Example: Matrix Multiplication

x: [NxD]

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

w: [DxM]

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



y: [NxM]

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

# Example: Matrix Multiplication

$x: [N \times D]$

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

$w: [D \times M]$

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



Matrix Multiply  $y = xw$

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



$y: [N \times M]$

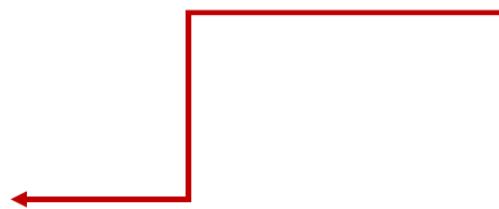
$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

$dL/dy: [N \times M]$

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

$dL/dx: [N \times D]$

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



# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

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

Matrix Multiply  $y = xw$

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

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

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

## Jacobians:

$$dy/dx: [(N \times D) \times (N \times M)]$$
$$dy/dw: [(D \times M) \times (N \times M)]$$

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

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

$$dL/dx: [N \times D]$$

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

$$dL/dx_{1,1}$$

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

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

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

$$dL/dy: [N \times M]$$

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

Local Gradient Slice:

$$dy/dx_{1,1}$$

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

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply  $y = xw$

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

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

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

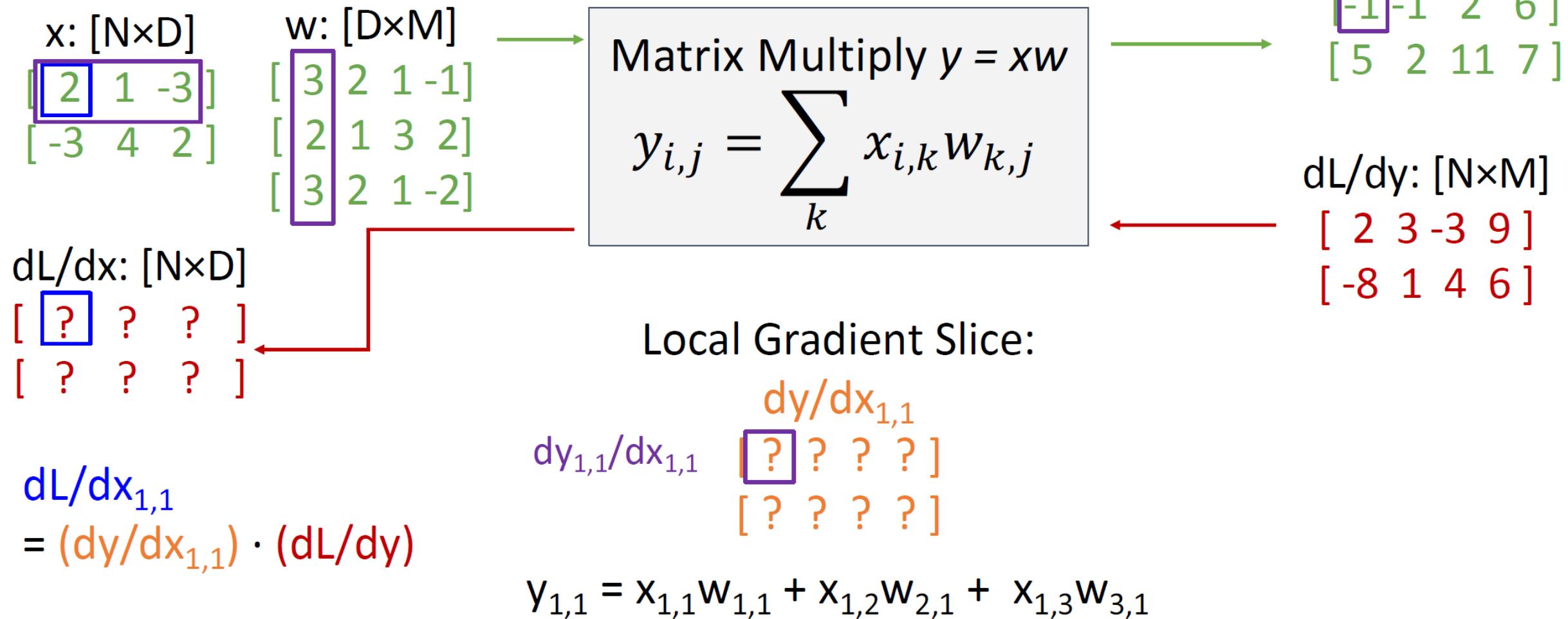
$$dL/dx_{1,1} = (dy/dx_{1,1}) \cdot (dL/dy)$$

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

Local Gradient Slice:

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

# Example: Matrix Multiplication



# Example: Matrix Multiplication

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

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

Matrix Multiply  $y = xw$

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

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

$$dL/dx: [N \times D]$$

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

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

Local Gradient Slice:

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

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply  $y = xw$

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

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

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

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

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

Local Gradient Slice:

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$

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

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

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

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

$$\frac{dL}{dy}: [N \times M]$$

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

$$\frac{dL}{dx}: [N \times D]$$

[ ? ? ? ]
[ ? ? ? ]

$$\begin{aligned} \frac{dL}{dx}_{1,1} \\ = (\text{dy/dx}_{1,1}) \cdot (\frac{dL}{dy}) \end{aligned}$$

Local Gradient Slice:

$$\begin{aligned} \frac{dy}{dx}_{1,1} &= [ 3 ? ? ? ] \\ dy_{1,2}/dx_{1,1} &= [ ? ? ? ? ] \end{aligned}$$

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$

$[2 \quad 1 \quad -3]$	$[3 \quad 2 \quad 1 \quad -1]$
$[-3 \quad 4 \quad 2]$	$[2 \quad 1 \quad 3 \quad 2]$

$$dL/dx: [N \times D]$$

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

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

$[-1 \quad \boxed{-1} \quad 2 \quad 6]$
$[5 \quad 2 \quad 11 \quad 7]$

$$dL/dy: [N \times M]$$

$[2 \quad 3 \quad -3 \quad 9]$
$[-8 \quad 1 \quad 4 \quad 6]$

Local Gradient Slice:

$$\frac{dy}{dx_{1,1}}$$

$$\frac{dy_{1,2}}{dx_{1,1}} \quad [3 \quad \boxed{2} \quad ? \quad ?]$$

$$[? \quad ? \quad ? \quad ?]$$

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

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

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$

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

$$dL/dx: [N \times D]$$

[ ? ? ? ]
[ ? ? ? ]

$$dL/dx_{1,1}$$

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

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

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

$$dL/dy: [N \times M]$$

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

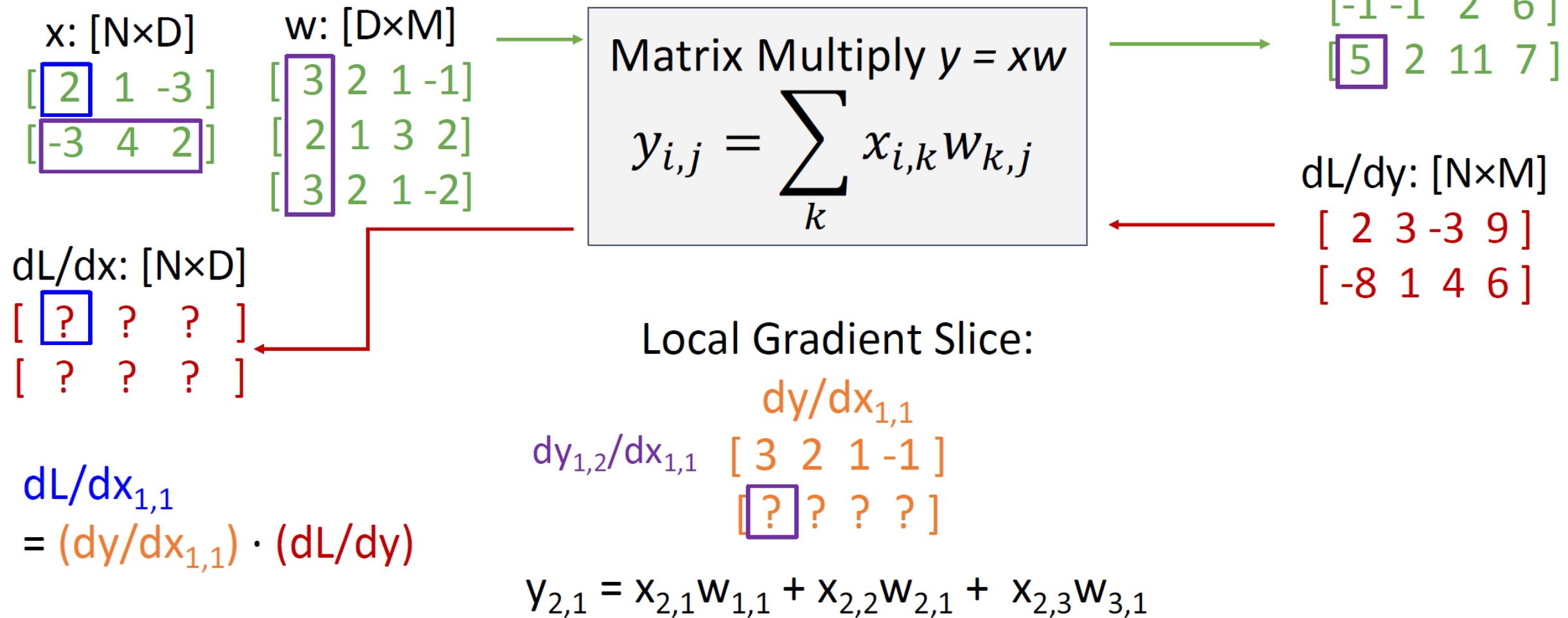
Local Gradient Slice:

$$\frac{dy}{dx}_{1,1}$$

$$dy_{1,2}/dx_{1,1} \quad [ 3 2 1 -1 ]$$

$$[ ? ? ? ? ]$$

# Example: Matrix Multiplication



# Example: Matrix Multiplication

$$x: [N \times D]$$

2	1	-3
-3	4	2

$$w: [D \times M]$$

3	2	1	-1
2	1	3	2
3	2	1	-2

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

-1	-1	2	6
5	2	11	7

$$\frac{dL}{dy}: [N \times M]$$

2	3	-3	9
-8	1	4	6

$$\frac{dL}{dx}: [N \times D]$$

?	?	?
?	?	?

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

Local Gradient Slice:

$$\frac{dy}{dx_{1,1}}$$

$$\frac{dy_{1,2}}{dx_{1,1}} \quad [3 \ 2 \ 1 \ -1]$$

0	?	?	?
---	---	---	---

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

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply  $y = xw$

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

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

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

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

Local Gradient Slice:

$$\frac{dy}{dx_{1,1}}$$
$$\frac{dy_{1,2}}{dx_{1,1}} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

$$dL/dx: [N \times D]$$

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

$$dL/dx_{1,1}$$

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

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

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

$$dL/dy: [N \times M]$$

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

Local Gradient Slice:

$$dy/dx_{1,1}$$

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

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$
$$dL/dx: [N \times D]$$
$$\begin{bmatrix} 0 & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$
$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$
$$dL/dy: [N \times M]$$
$$\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

Local Gradient Slice:

$$dy/dx_{1,1}$$
$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

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

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M]$$
$$\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply  $y = xw$

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

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

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

$$dL/dx: [N \times D]$$
$$\begin{bmatrix} 0 & ? & ? \\ ? & ? & -30 \end{bmatrix}$$

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

Local Gradient Slice:

$$dy/dx_{2,3}$$
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

# Example: Matrix Multiplication

$$x: [N \times D] \quad w: [D \times M] \longrightarrow$$

[ 2 1 -3 ]	[ 3 2 1 -1 ]
[ -3 4 <b>2</b> ]	[ 2 1 3 2 ]

$$dL/dx: [N \times D]$$

[ 0 ? ? ]
[ ? ? <b>-30</b> ]

Matrix Multiply  $y = xw$

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

$$y: [N \times M]$$

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

$$dL/dy: [N \times M]$$

[ 2 3 -3 9 ]
[ <b>-8 1 4 6</b> ]

Local Gradient Slice:

$$\frac{dy}{dx}_{2,3}$$

[ 0 0 0 0 ]
[ <b>3 2 1 -2</b> ]

$$\begin{aligned}
 dL/dx_{2,3} &= (\frac{dy}{dx}_{2,3}) \cdot (dL/dy) \\
 &= (w_{3,:}) \cdot (dL/dy_{2,:}) \\
 &= 3 * (-8) + 2 * 1 + 1 * 4 + (-2) * 6 = -30
 \end{aligned}$$

# Example: Matrix Multiplication

$x: [N \times D]$

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

$w: [D \times M]$

Matrix Multiply  $y = xw$

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

$y: [N \times M]$

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

$dL/dy: [N \times M]$

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

$dL/dx: [N \times D]$

$$\begin{bmatrix} 0 & 16 & -9 \\ -24 & 9 & -30 \end{bmatrix}$$

$dL/dx_{i,j}$

$$= (\frac{dy}{dx_{i,j}}) \cdot (dL/dy)$$

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

# Example: Matrix Multiplication

$x: [N \times D]$

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

$w: [D \times M]$

Matrix Multiply  $y = xw$

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

$y: [N \times M]$

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

$dL/dx: [N \times D]$

$$\begin{bmatrix} 0 & 16 & -9 \\ -24 & 9 & -30 \end{bmatrix}$$

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

$[N \times D] \quad [N \times M] \quad [M \times D]$

$dL/dx_{i,j}$

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

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

$dL/dy: [N \times M]$

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

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

# Example: Matrix Multiplication

$x: [N \times D]$

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

$w: [D \times M]$

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

Matrix Multiply  $y = xw$

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

$y: [N \times M]$

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

$dL/dy: [N \times M]$

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

$dL/dx: [N \times D]$

$$\begin{bmatrix} 0 & 16 & -9 \\ -24 & 9 & -30 \end{bmatrix}$$

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

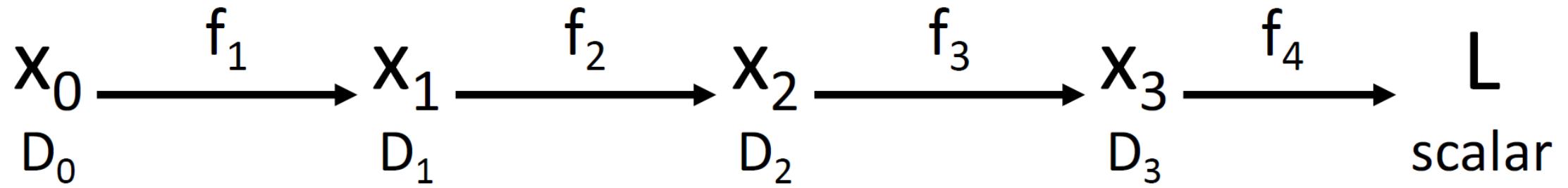
$[N \times D] \quad [N \times M] \quad [M \times D]$

$$dL/dw = x^T (dL/dy)$$

$[D \times M] \quad [D \times N] \quad [N \times M]$

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

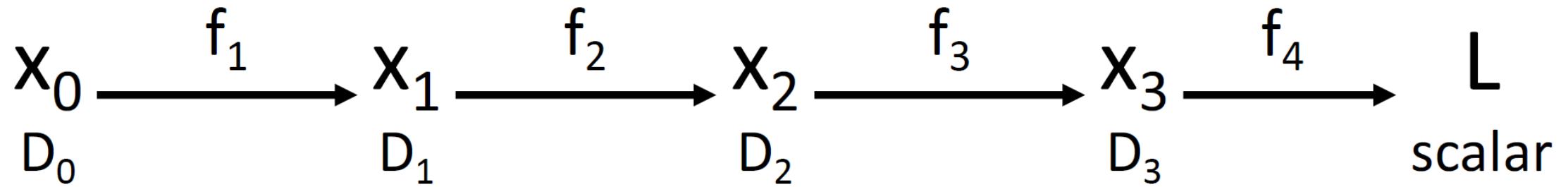
# Backpropagation: Another View



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



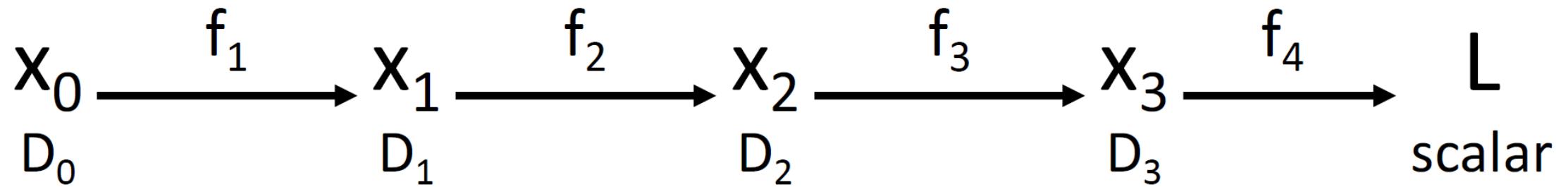
Matrix multiplication is **associative**: we can compute products in any order

Chain  
rule

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

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

# Reverse-Mode Automatic Differentiation

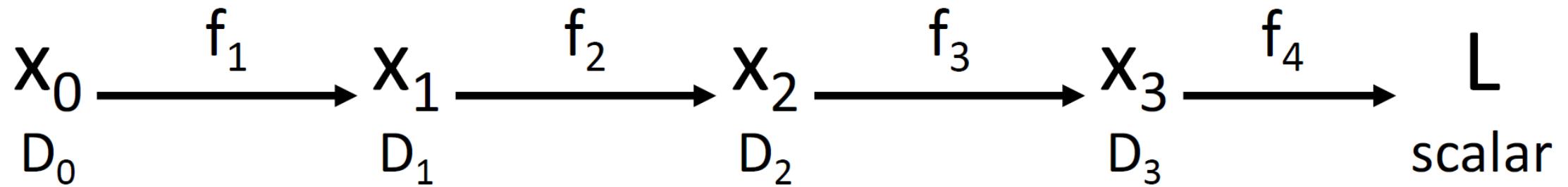


Matrix multiplication is **associative**: we can compute products in any order  
Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule

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

# Reverse-Mode Automatic Differentiation



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

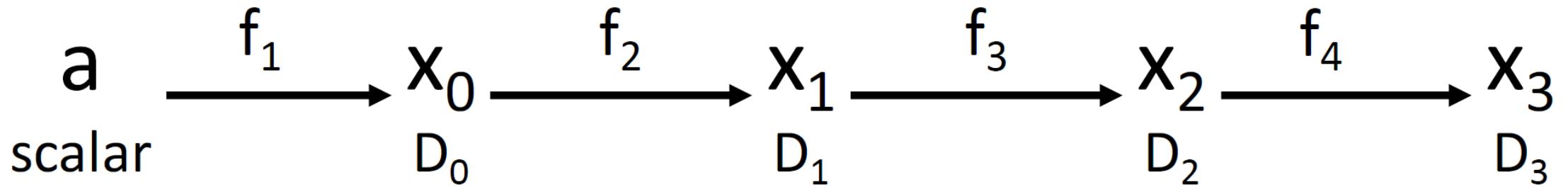
$$\frac{\partial L}{\partial x_0} = \overbrace{\left( \frac{\partial x_1}{\partial x_0} \right) \left( \frac{\partial x_2}{\partial x_1} \right) \left( \frac{\partial x_3}{\partial x_2} \right) \left( \frac{\partial L}{\partial x_3} \right)}^{\longleftarrow}$$

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

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

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

# Forward-Mode Automatic Differentiation

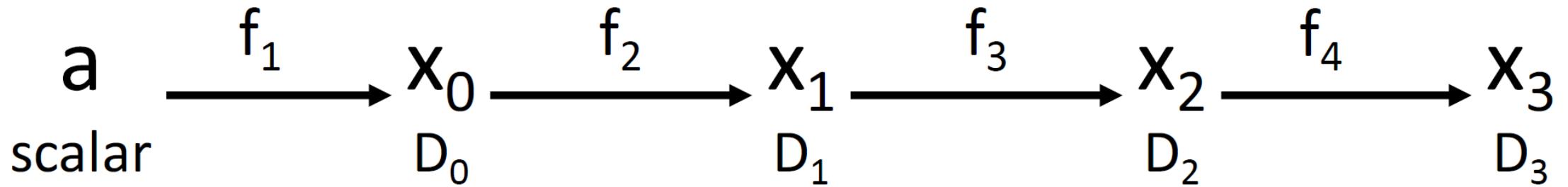


Chain  
rule

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

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

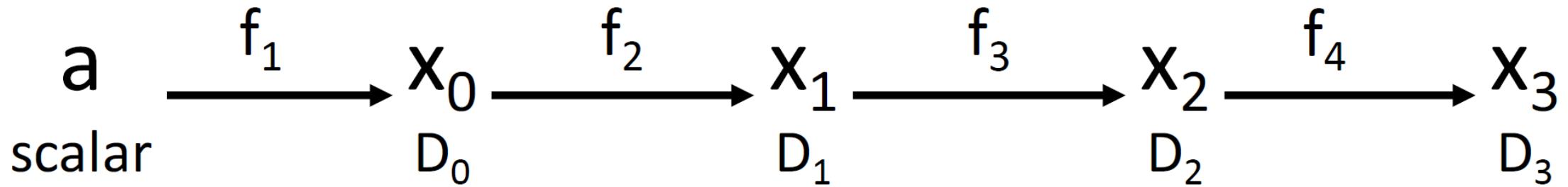
# Forward-Mode Automatic Differentiation



Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

Chain rule 
$$\frac{\partial x_3}{\partial a} = \overbrace{\left( \frac{\partial x_0}{\partial a} \right) \left( \frac{\partial x_1}{\partial x_0} \right) \left( \frac{\partial x_2}{\partial x_1} \right) \left( \frac{\partial x_3}{\partial x_2} \right)}^{\longrightarrow}$$
$$[D_0] \quad [D_0 \times D_1] \quad [D_1 \times D_2] \quad [D_2 \times D_3]$$

# Forward-Mode Automatic Differentiation

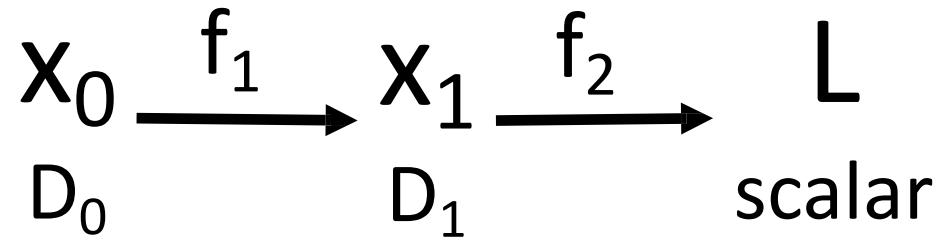


Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector  
Not implemented in PyTorch / TensorFlow =(

Chain rule

$$\frac{\partial x_3}{\partial a} = \overbrace{\left( \frac{\partial x_0}{\partial a} \right) \left( \frac{\partial x_1}{\partial x_0} \right) \left( \frac{\partial x_2}{\partial x_1} \right) \left( \frac{\partial x_3}{\partial x_2} \right)}^{\text{But you can implement forward-mode AD using two calls to reverse-mode AD! (Inefficient but elegant)}}$$
$$[D_0] \quad [D_0 \times D_1] \quad [D_1 \times D_2] \quad [D_2 \times D_3]$$

# Backprop: Higher-Order Derivatives

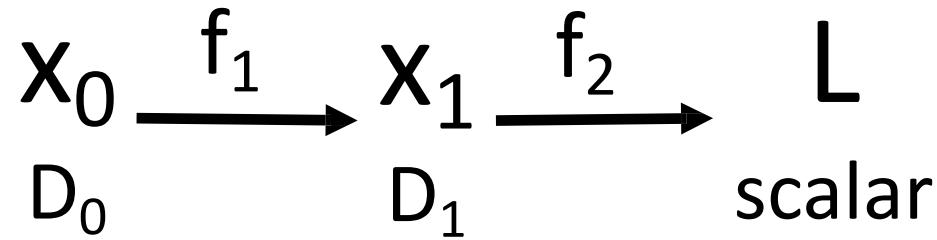


$\frac{\partial^2 L}{\partial x_0^2}$  **Hessian matrix**

$\frac{\partial^2 L}{\partial x_0^2}$  H of second  
derivatives.

$[D_0 \times D_0]$

# Backprop: Higher-Order Derivatives



$$\frac{\partial^2 L}{\partial x_0^2}$$

$[D_0 \times D_0]$

**Hessian matrix**  
H of second  
derivatives.

$$\frac{\partial^2 L}{\partial x_0^2} v$$

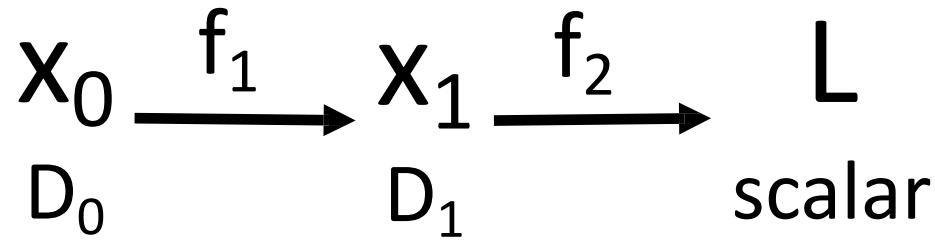
$[D_0 \times D_0] [D_0]$

**Hessian / vector multiply**

$$\frac{\partial^2 L}{\partial x_0^2}$$

$$v$$

# Backprop: Higher-Order Derivatives



$$\frac{\partial^2 L}{\partial x_0^2}$$

**Hessian matrix**  
H of second  
derivatives.

$$[D_0 \times D_0]$$

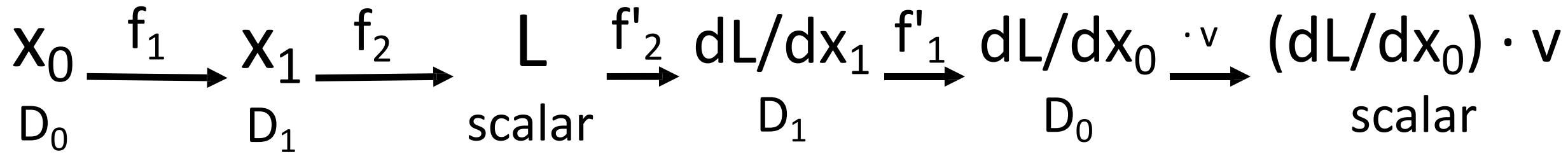
**Hessian / vector multiply**

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right]$$

(if  $v$  doesn't  
depend on  $x_0$ )

$$[D_0 \times D_0] [D_0]$$

# Backprop: Higher-Order Derivatives



$$\frac{\partial^2 L}{\partial x_0^2}$$

**Hessian matrix**  
H of second  
derivatives.

$$[D_0 \times D_0]$$

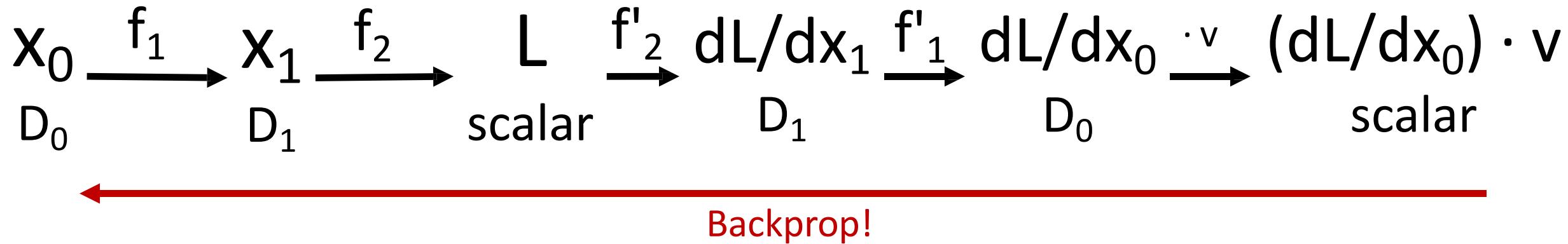
**Hessian / vector multiply**

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right]$$

(if  $v$  doesn't  
depend on  $x_0$ )

$$[D_0 \times D_0] [D_0]$$

# Backprop: Higher-Order Derivatives



$$\frac{\partial^2 L}{\partial x_0^2}$$

**Hessian matrix**  
H of second  
derivatives.

$[D_0 \times D_0]$

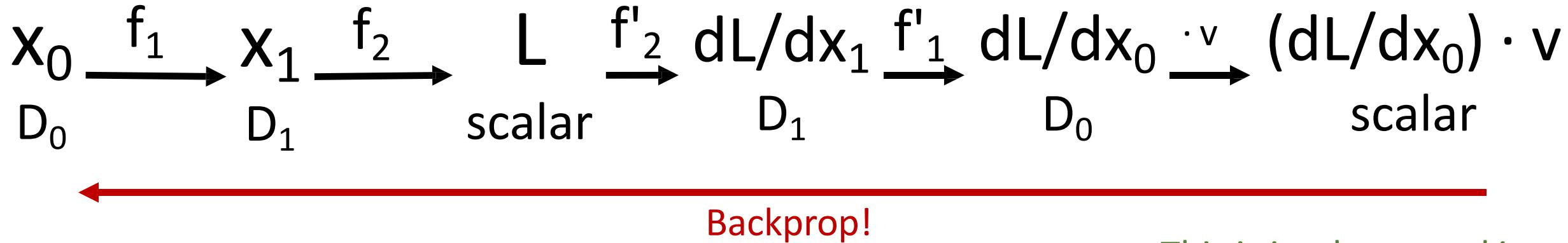
**Hessian / vector multiply**

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right]$$

(if  $v$  doesn't  
depend on  $x_0$ )

$[D_0 \times D_0] [D_0]$

# Backprop: Higher-Order Derivatives



$$\frac{\partial^2 L}{\partial x_0^2}$$

**Hessian matrix**  
H of second  
derivatives.

$[D_0 \times D_0]$

$$\frac{\partial^2 L}{\partial x_0^2} v = \frac{\partial}{\partial x_0} \left[ \frac{\partial L}{\partial x_0} \cdot v \right]$$

$[D_0 \times D_0] [D_0]$

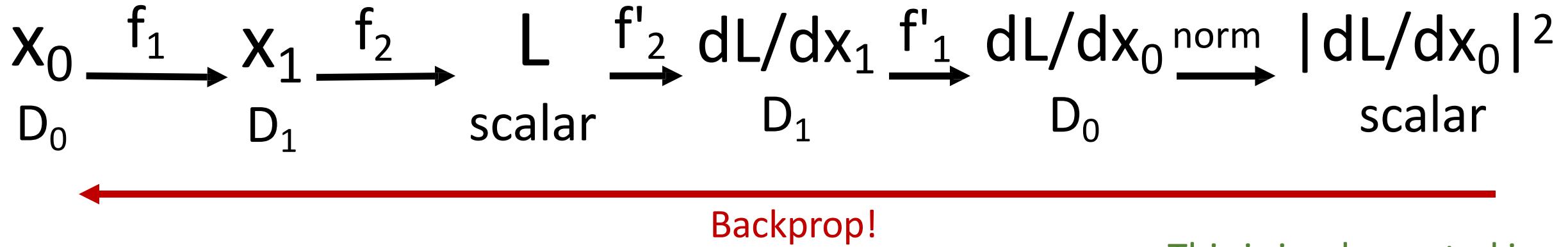
Lecture 4 - 117

**Hessian / vector multiply**

This is implemented in  
PyTorch / Tensorflow!

(if  $v$  doesn't  
depend on  $x_0$ )

# Backprop: Higher-Order Derivatives



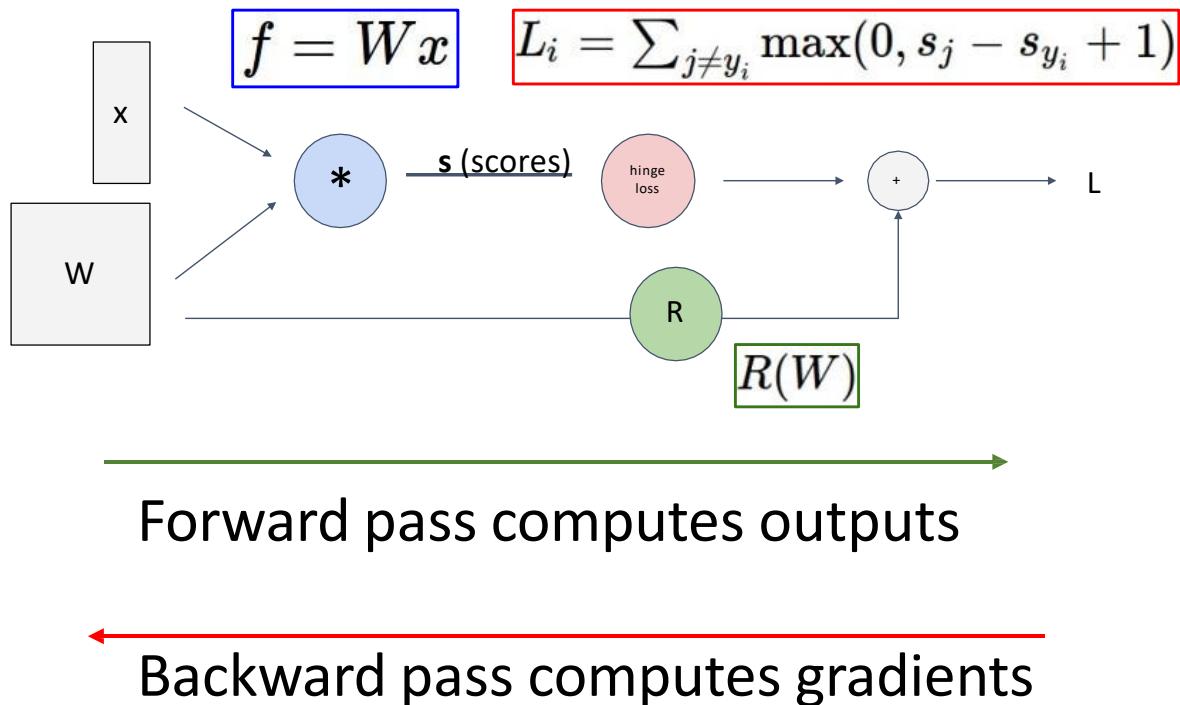
This is implemented in  
PyTorch / Tensorflow!

**Example:** Regularization to penalize the norm of the gradient

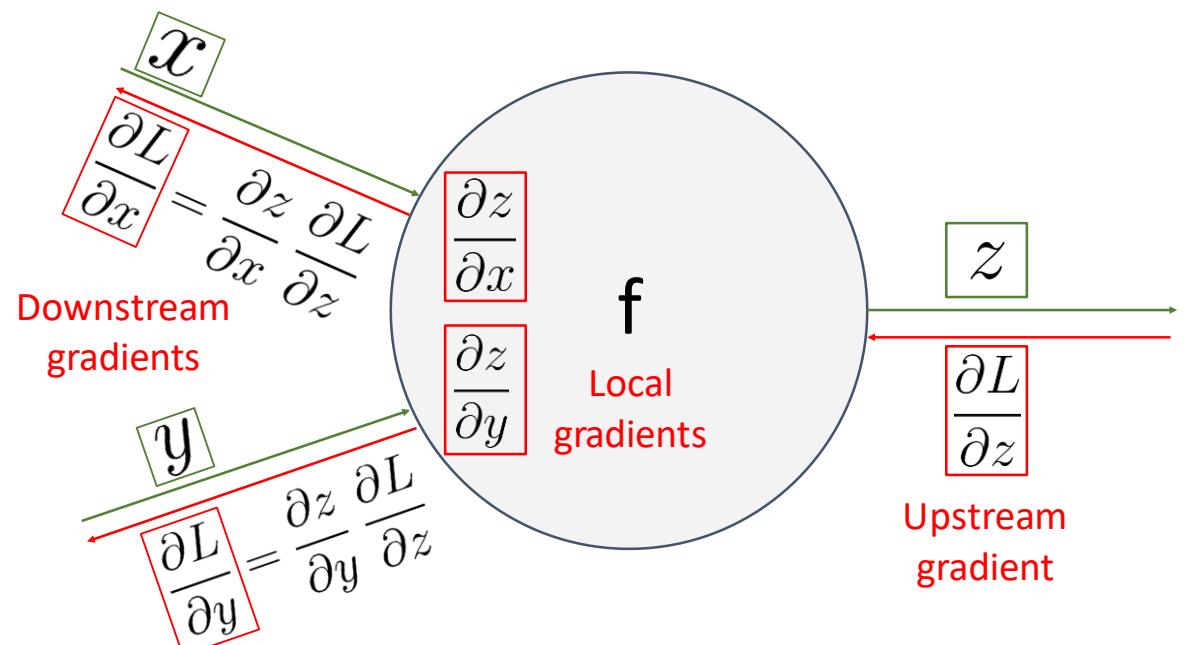
$$R(W) = \left\| \frac{\partial L}{\partial W} \right\|_2^2 = \left( \frac{\partial L}{\partial W} \right) \cdot \left( \frac{\partial L}{\partial W} \right) \quad \frac{\partial}{\partial x_0} [R(W)] = 2 \left( \frac{\partial^2 L}{\partial x_0^2} \right) \left( \frac{\partial L}{\partial x_0} \right)$$

# Summary

Represent complex expressions as **computational graphs**



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 HW3!)

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

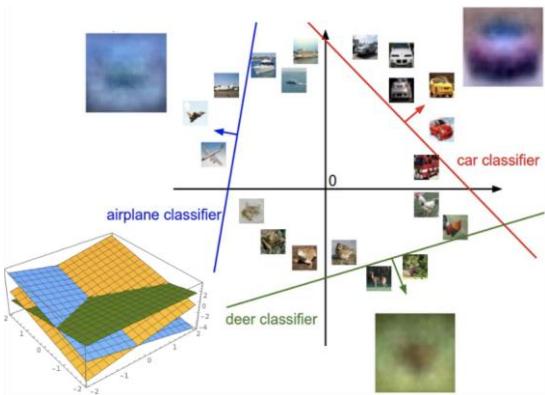
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backprop can be implemented with a modular API, as a set of paired forward/backward functions (We will do this on HW4!)

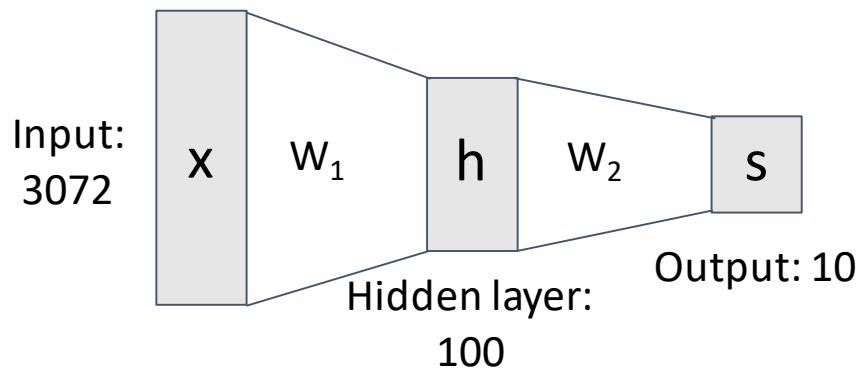
```
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        z = x * y
        return z

    @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
```

$$f(x, W) = Wx$$

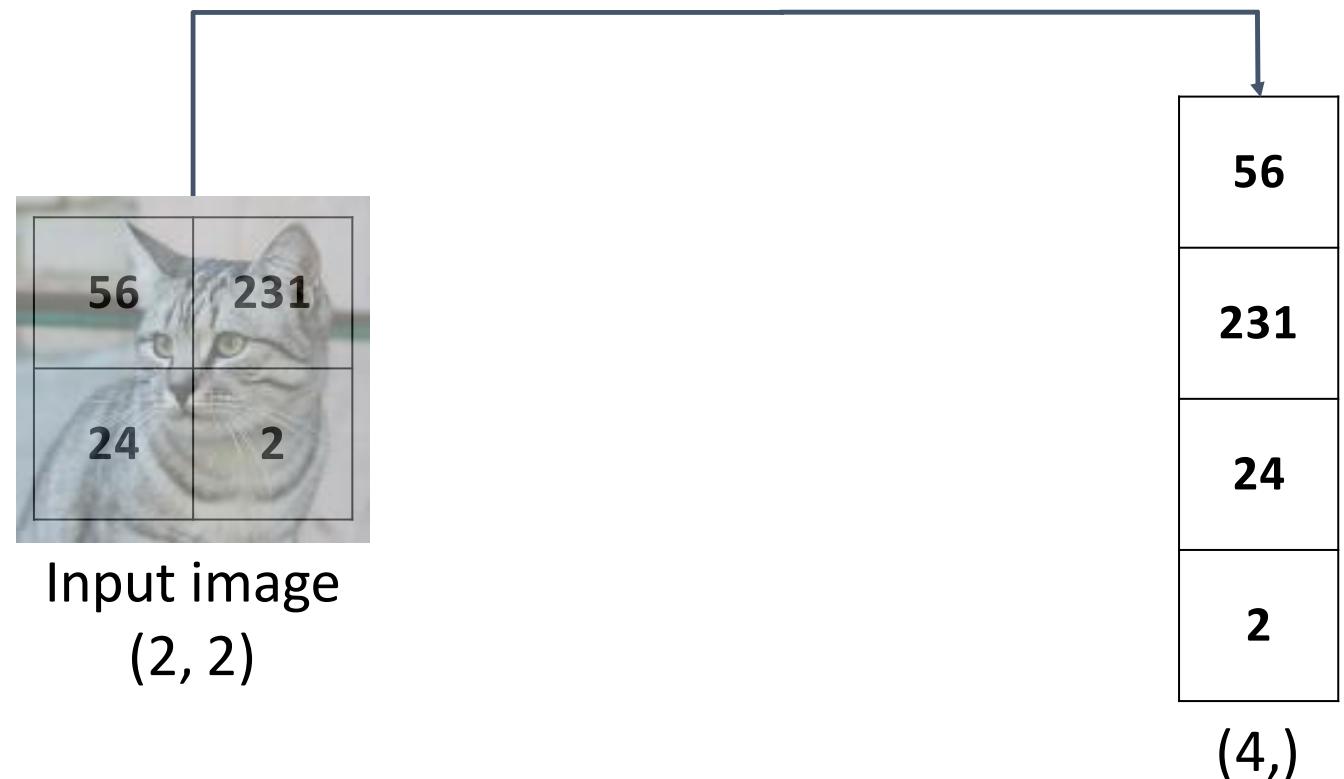


$$f = W_2 \max(0, W_1 x)$$



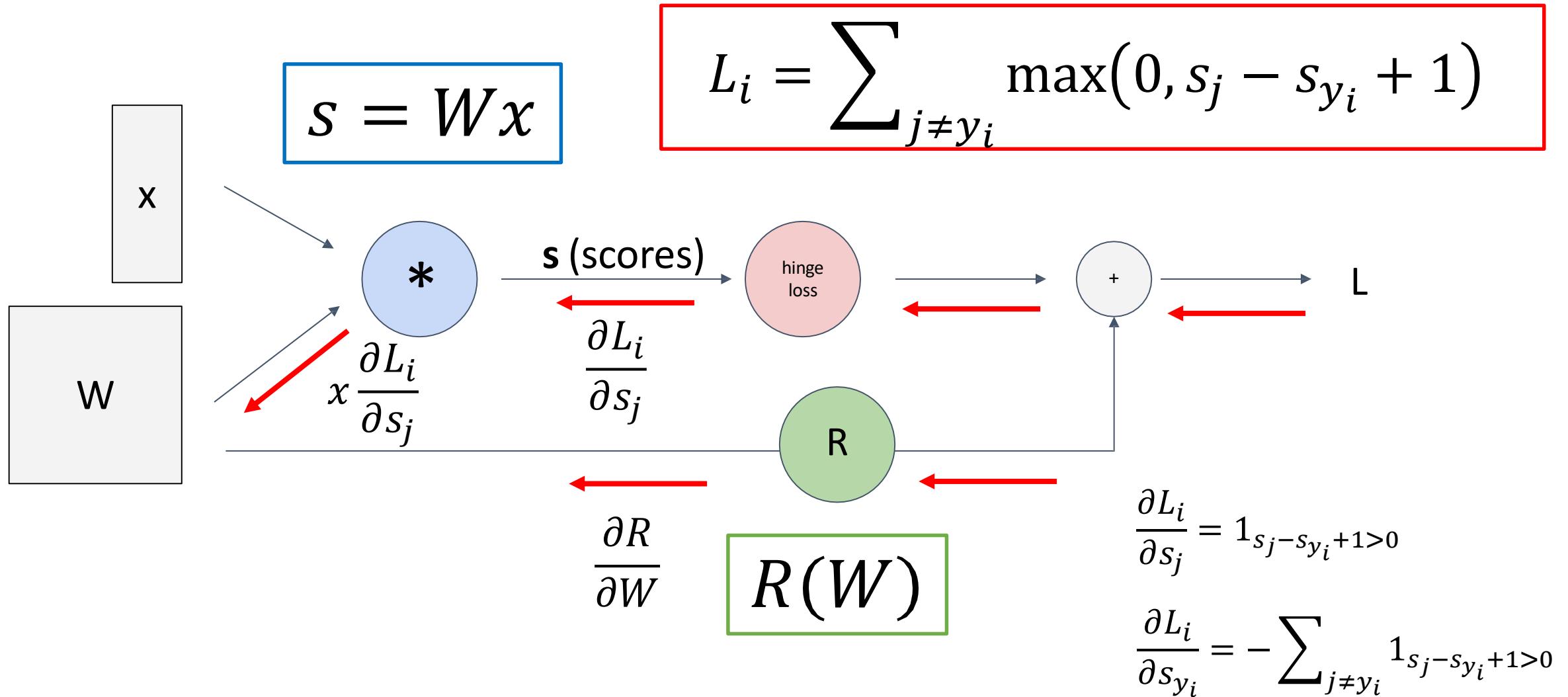
**Problem:** So far our classifiers don't respect the spatial structure of images!

Stretch pixels into column

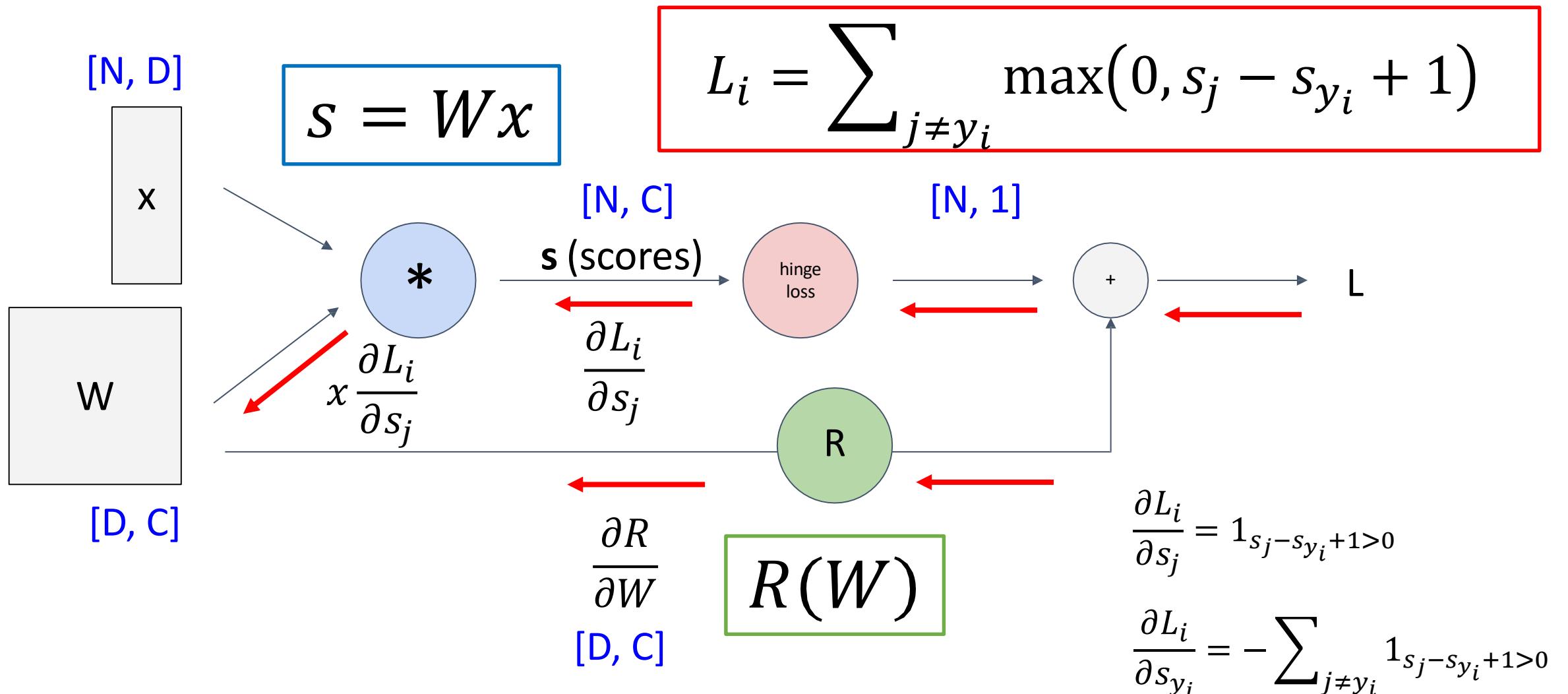


Next time:  
Convolutional Neural Networks

# HW2



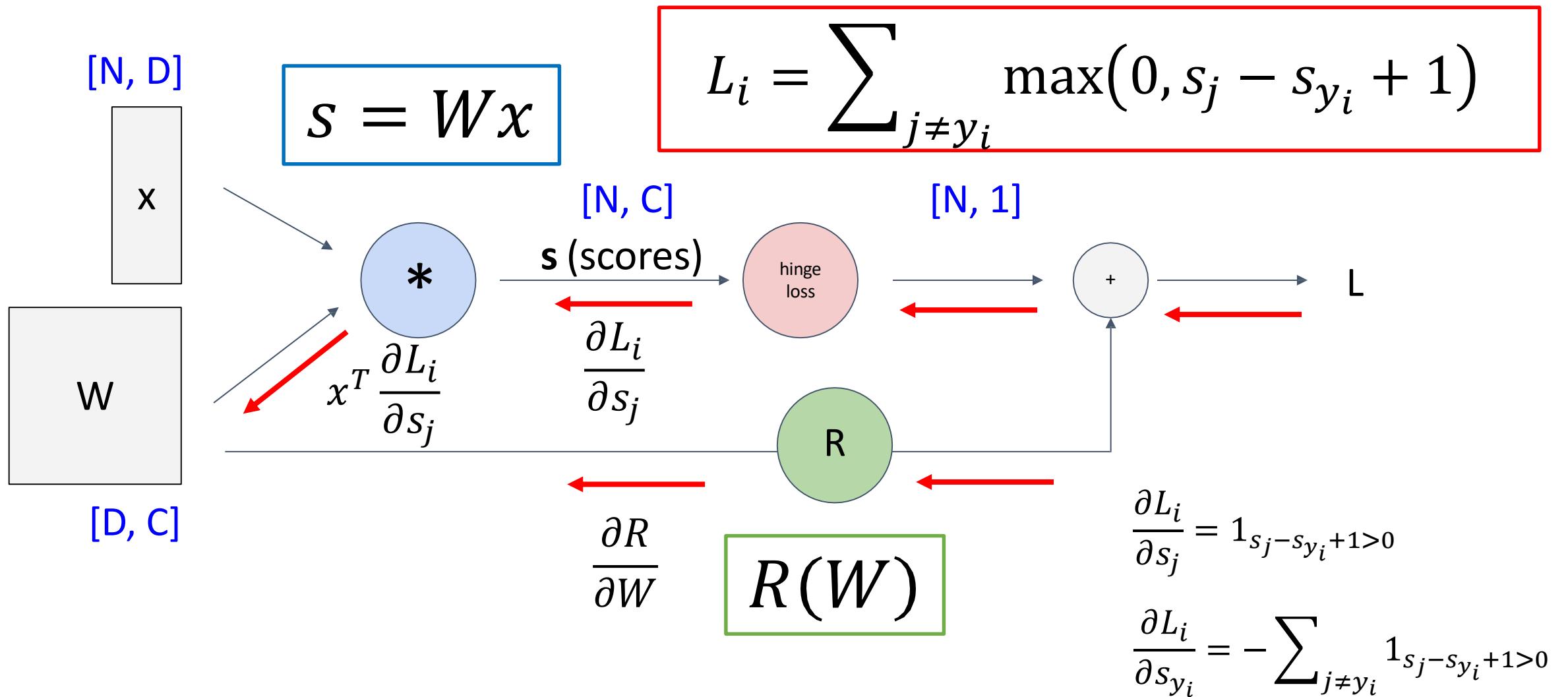
# HW2



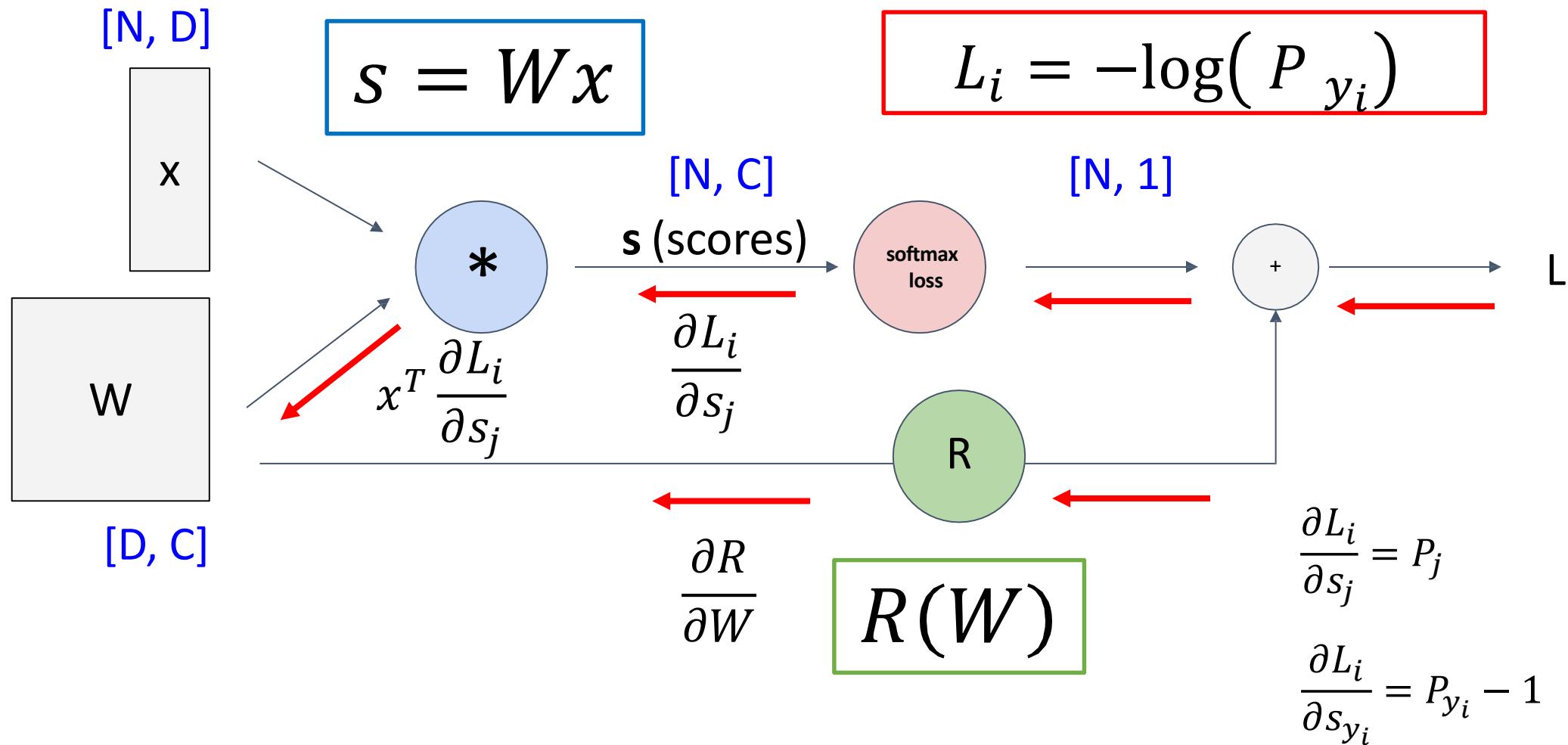
$$\frac{\partial L_i}{\partial s_j} = 1_{s_j - s_{y_i} + 1 > 0}$$

$$\frac{\partial L_i}{\partial s_{y_i}} = - \sum_{j \neq y_i} 1_{s_j - s_{y_i} + 1 > 0}$$

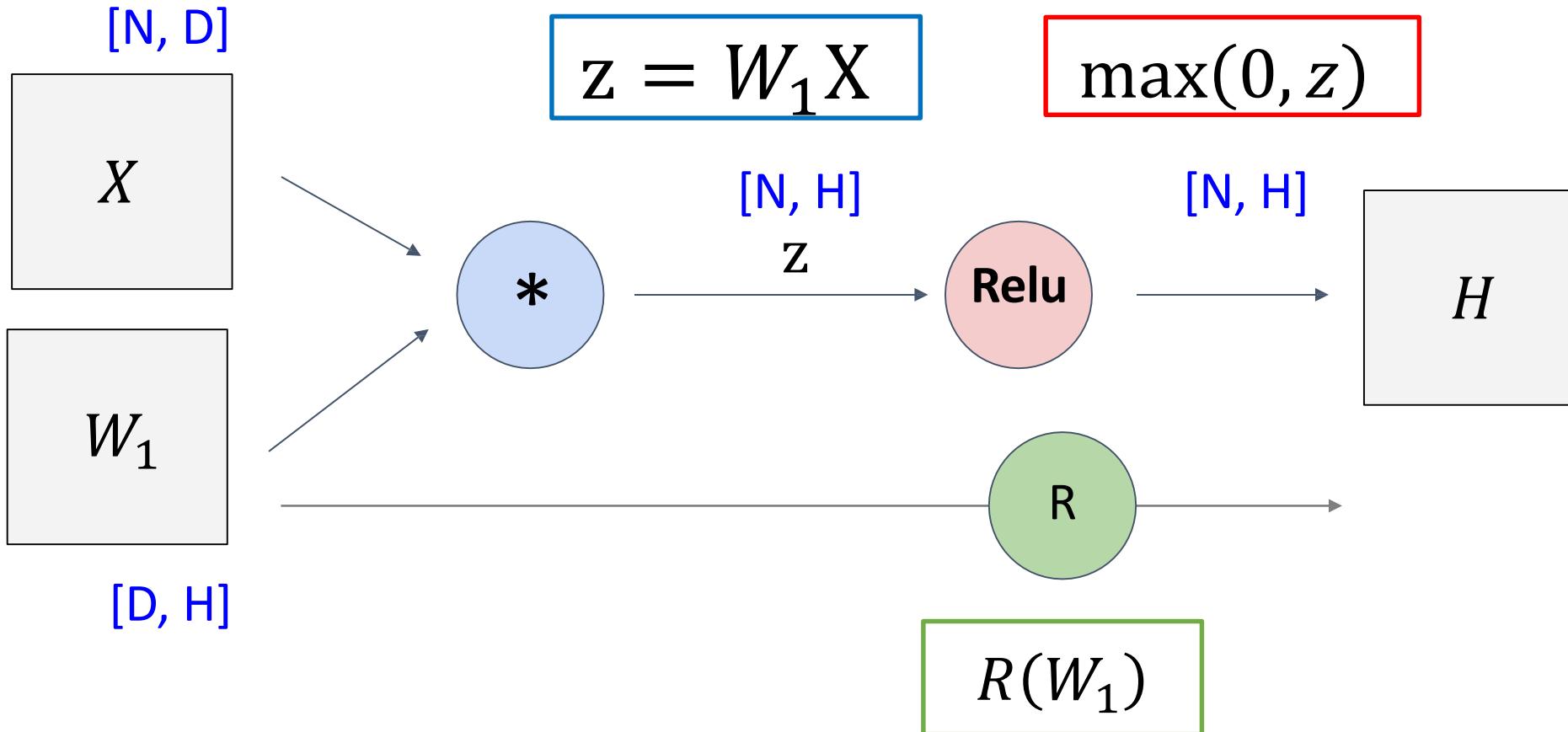
# HW2



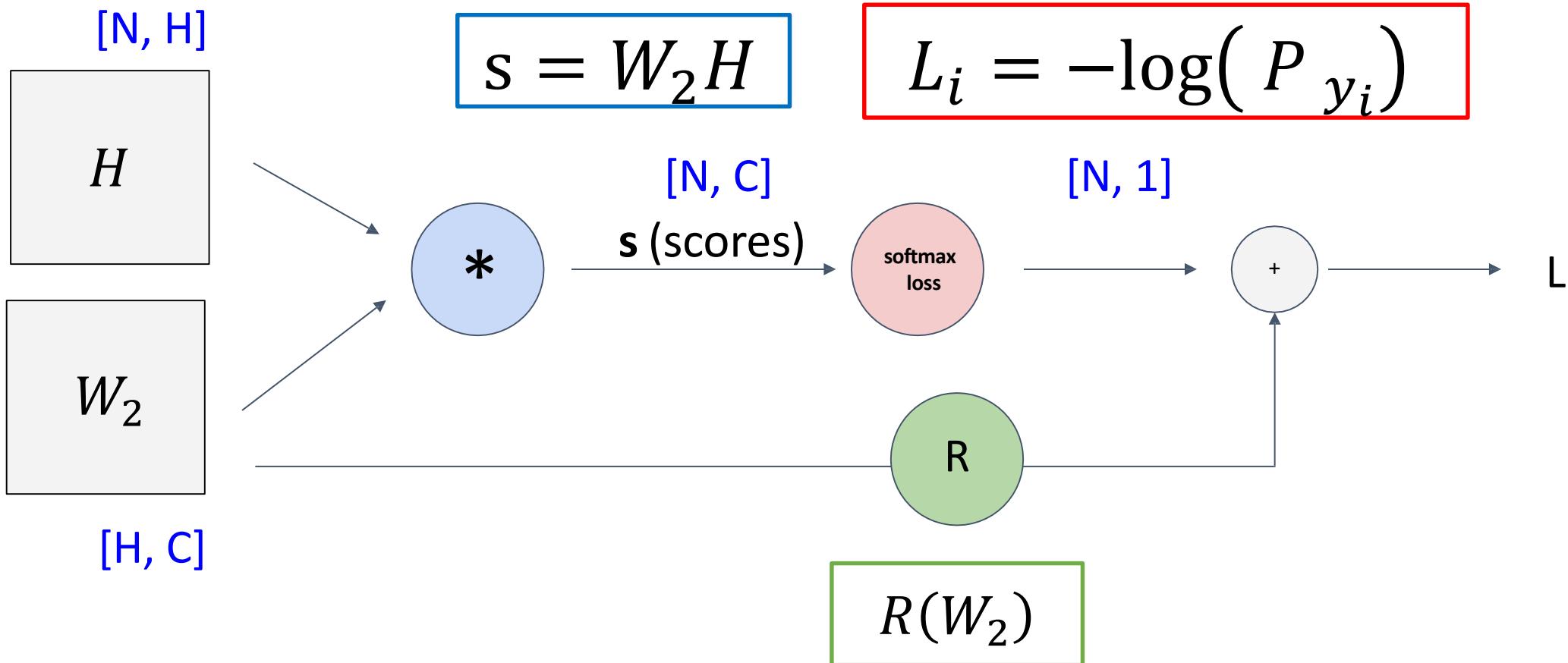
# HW2



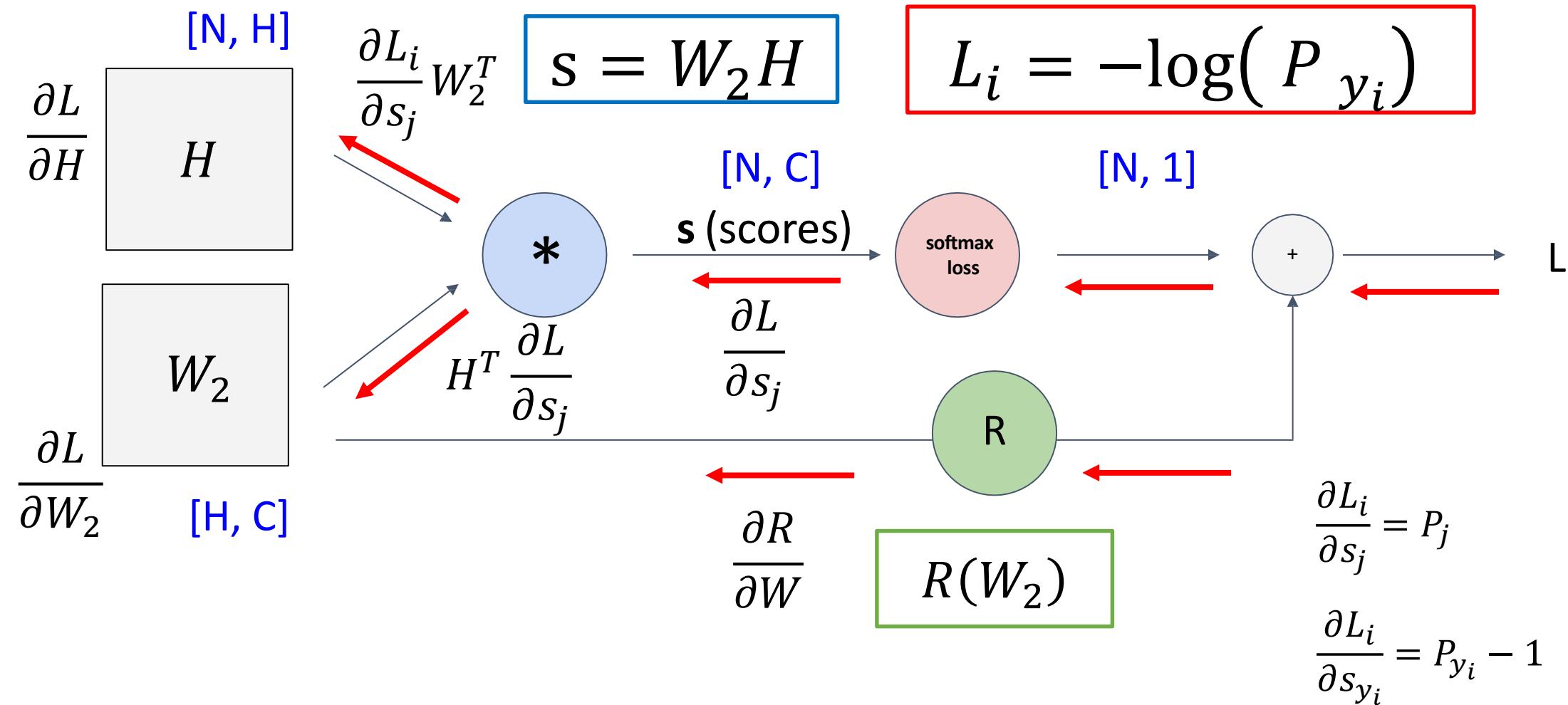
# HW5-Forward



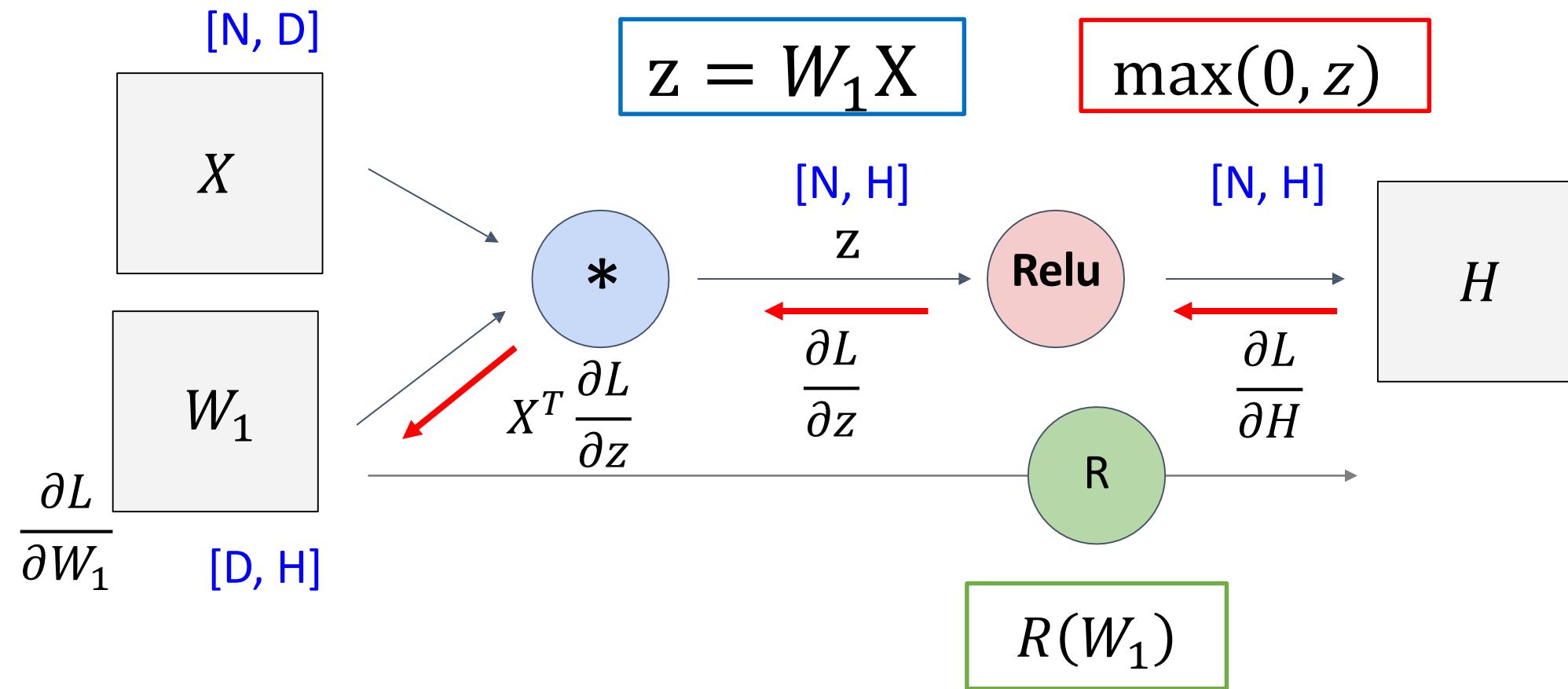
# HW5-Forward



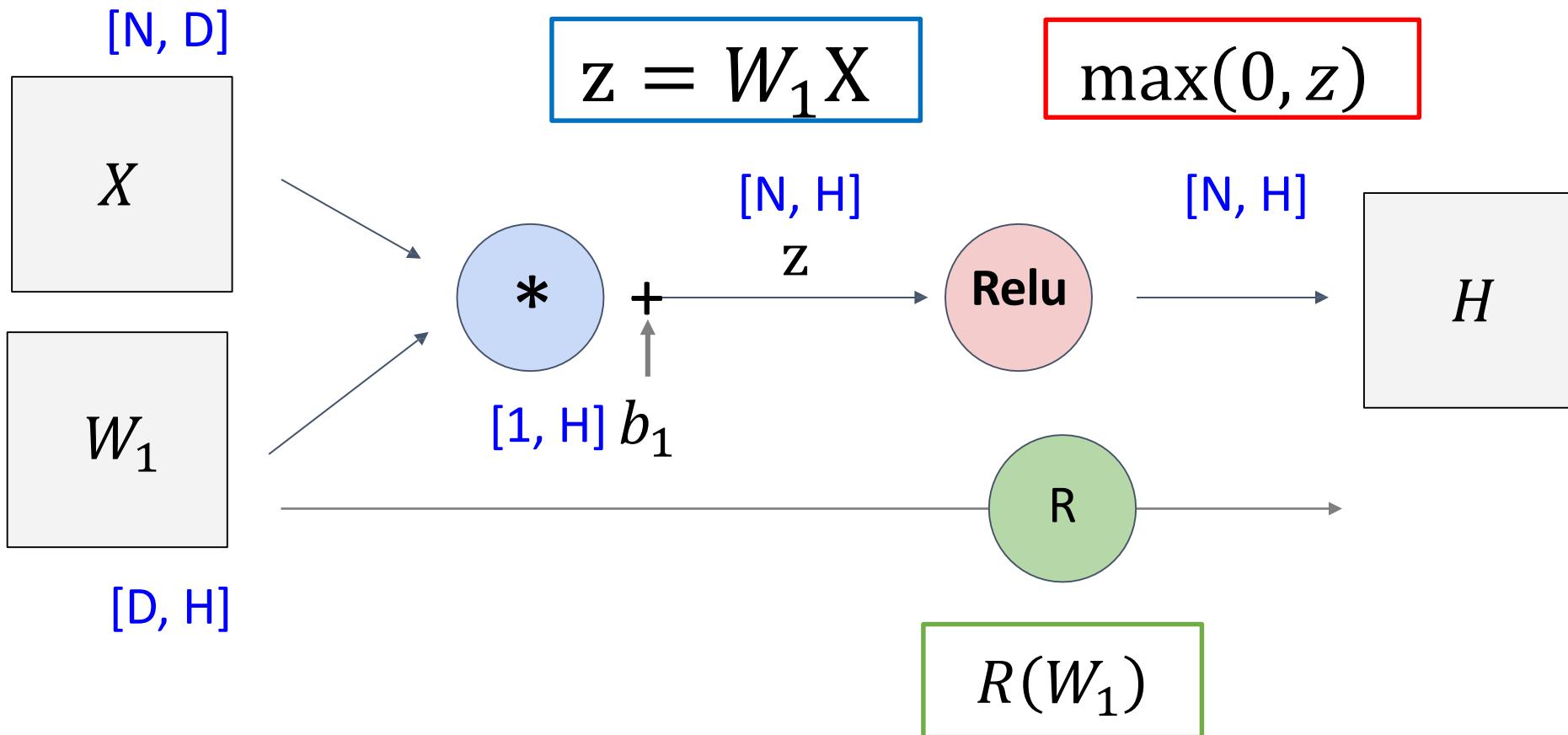
# HW5-Backward



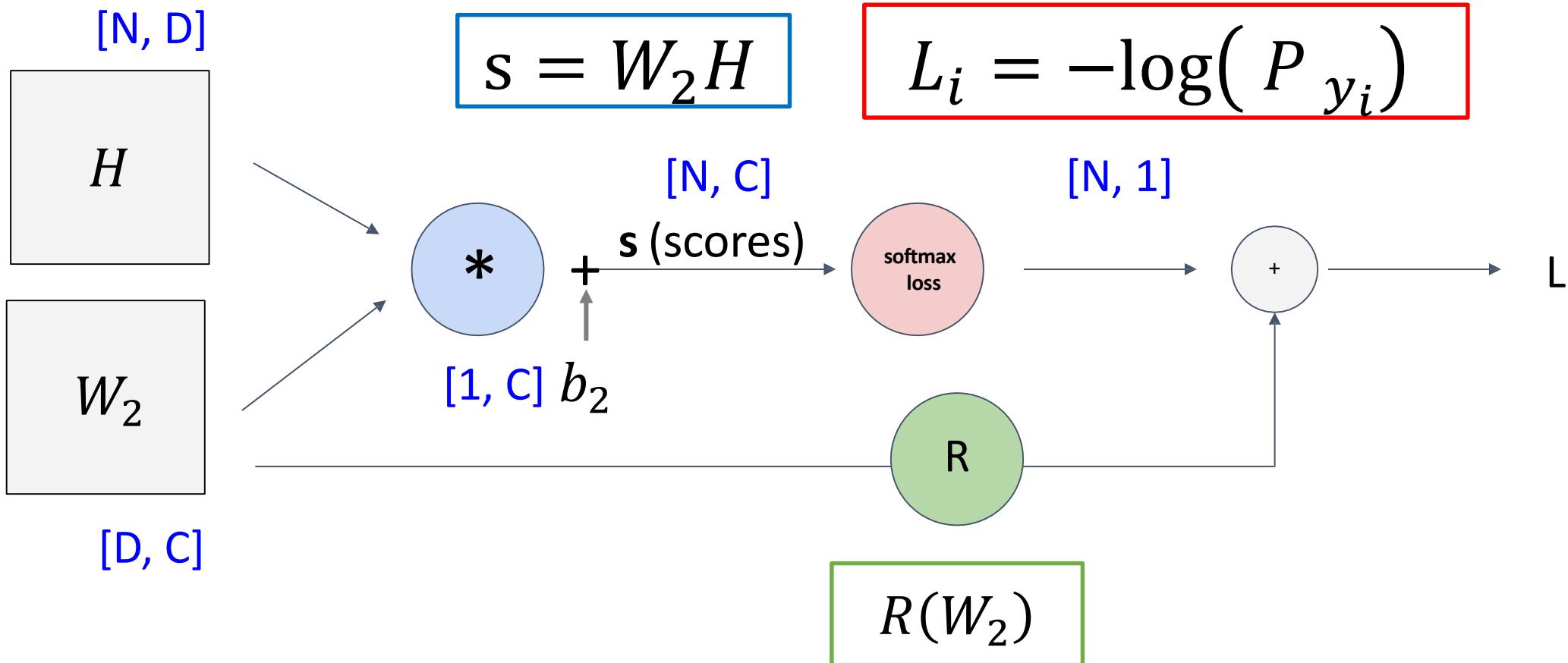
# HW5-Backward



# HW5-Forward



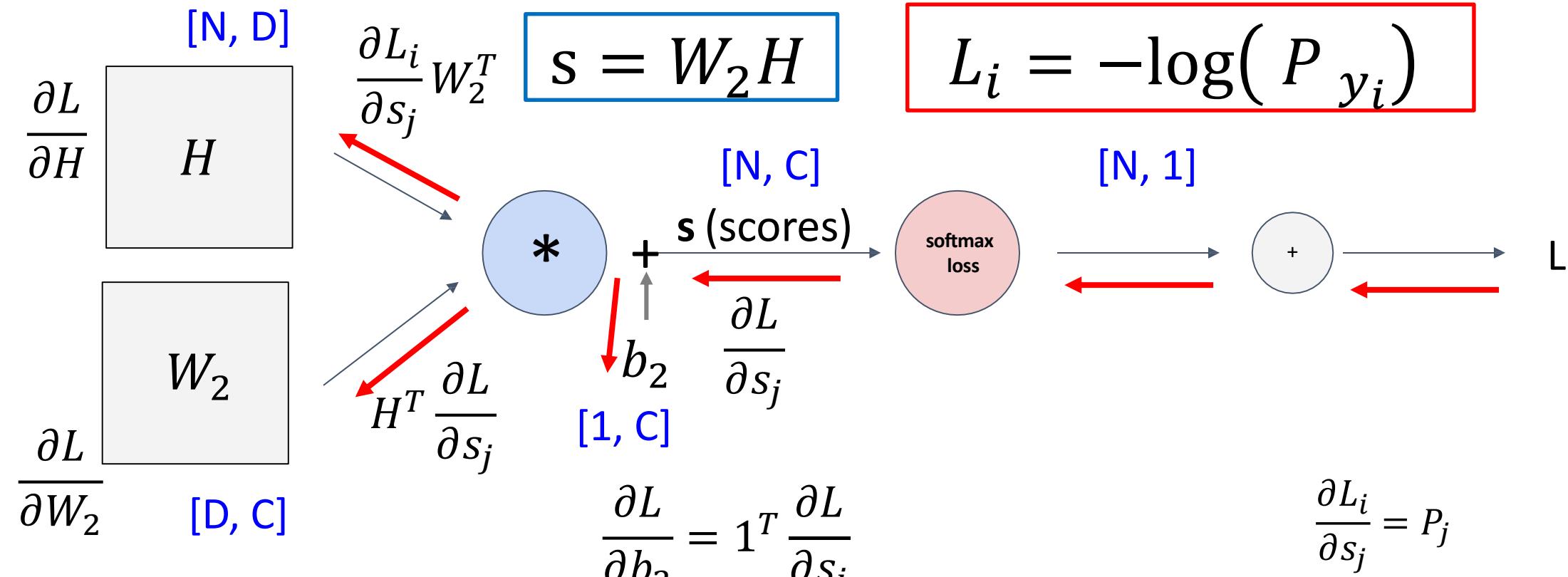
# HW5-Forward



# HW5-Backward

$$s = W_2H + b_2 = [W_2 \quad b_2] \begin{bmatrix} H \\ 1 \end{bmatrix}$$

[D]  
[1]



$$\frac{\partial L}{\partial b_2} = 1^T \frac{\partial L}{\partial s_j}$$

$$\frac{\partial L_i}{\partial s_j} = P_j$$

$$\frac{\partial L_i}{\partial s_{y_i}} = P_{y_i} - 1$$