CS163: Deep Learning for Computer Vision

Lecture 6: Backpropagation

Courtesy of Prof. Justin Johnson at Umich and the team of Stanford CS231n for the slides

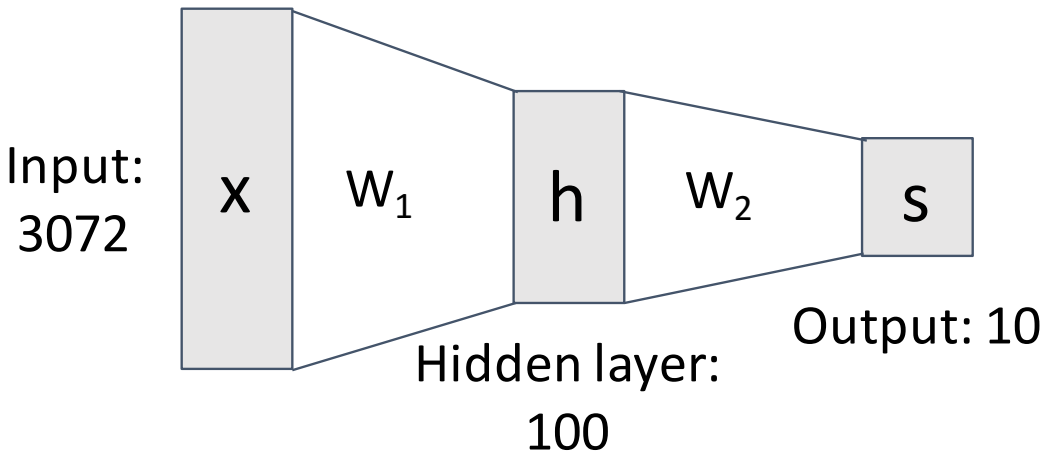
Announcement

- TA Tutorial on Course Project
- Deep Learning Hardware and Software

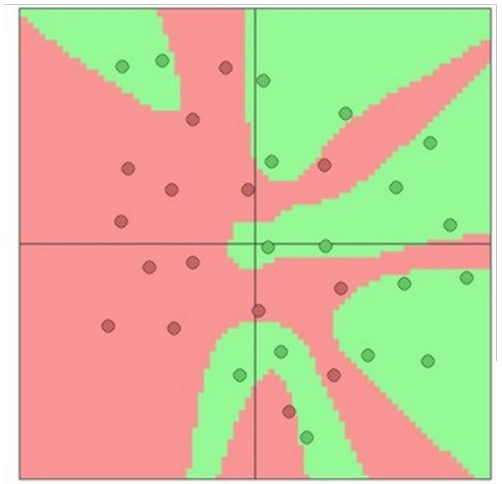
Last time: Neural Networks

From linear classifiers to fully-connected networks

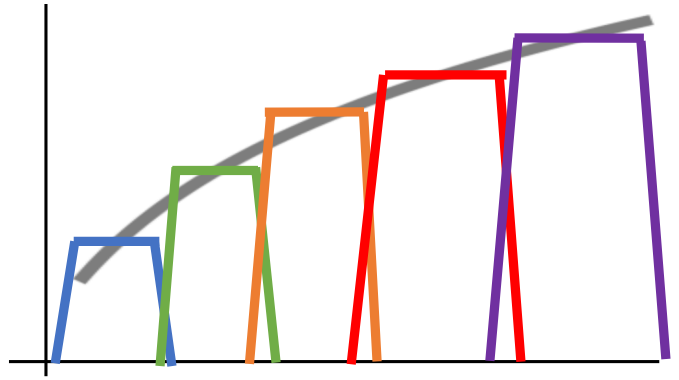
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



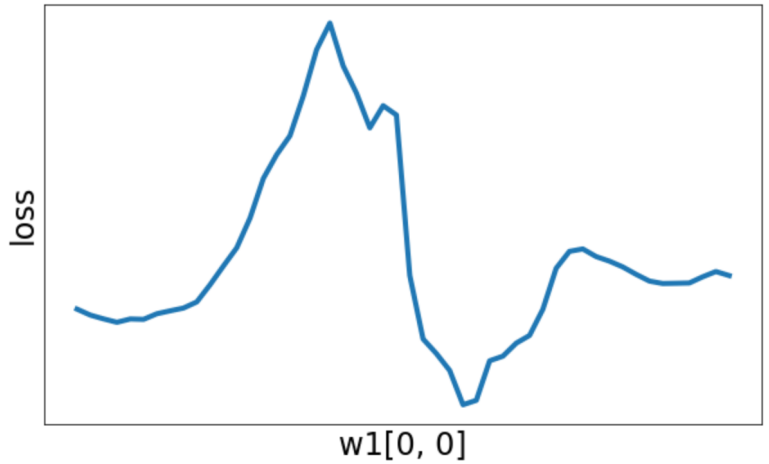
Space Warping



Universal Approximation



Nonconvex



Problem: How to compute gradients?

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

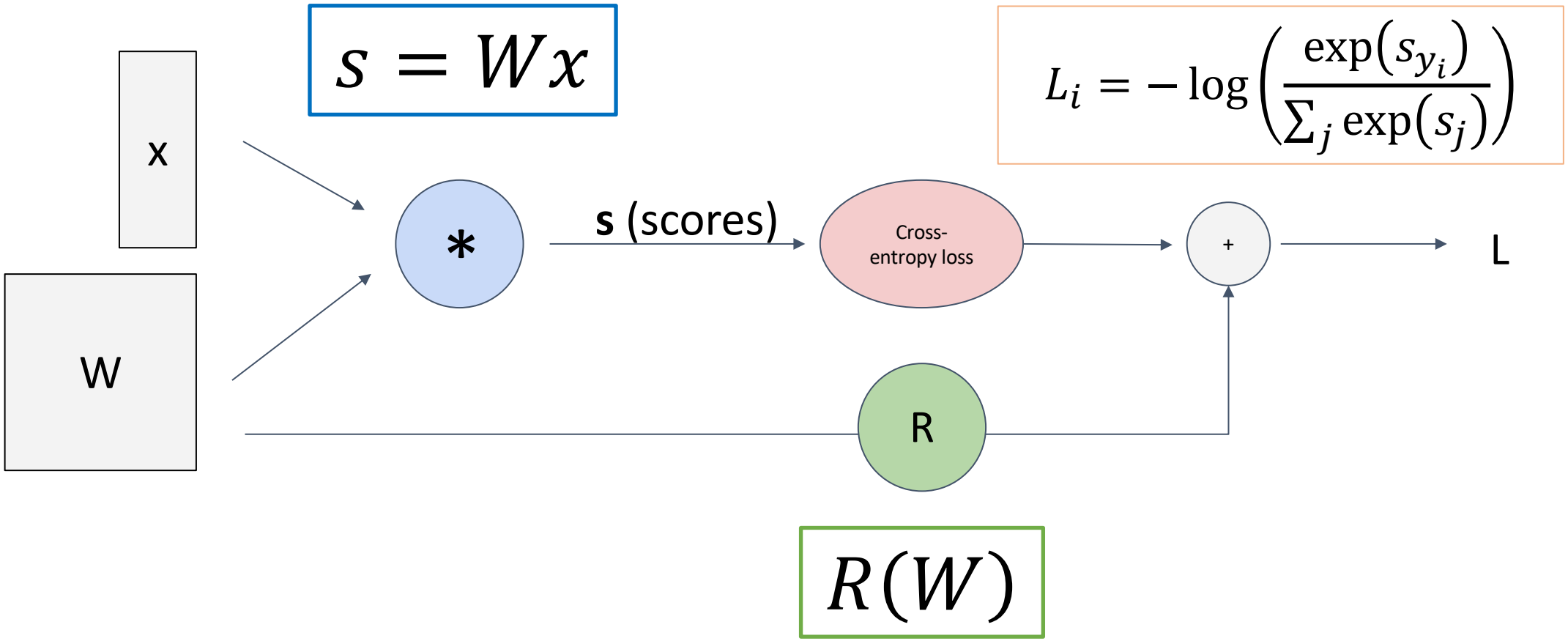
$L_i = -\log\left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}\right)$ Softmax loss

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

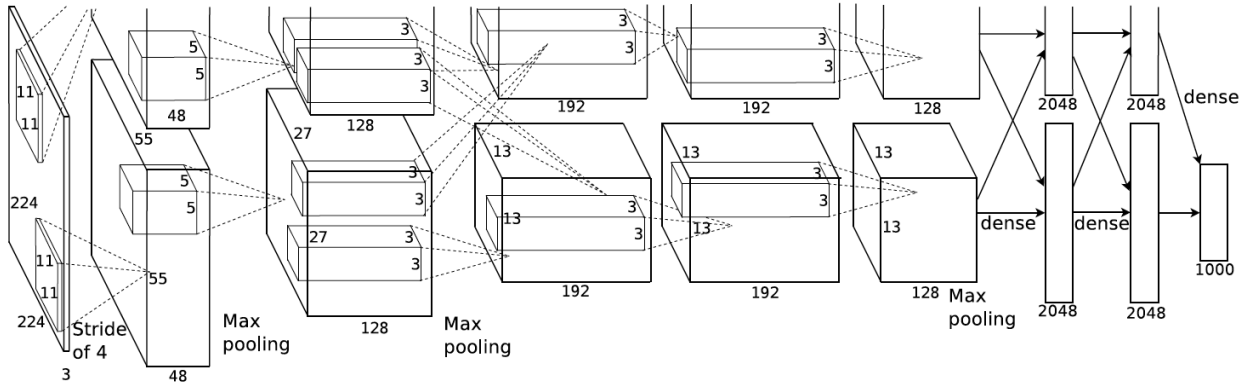
If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}$ then we can optimize with SGD

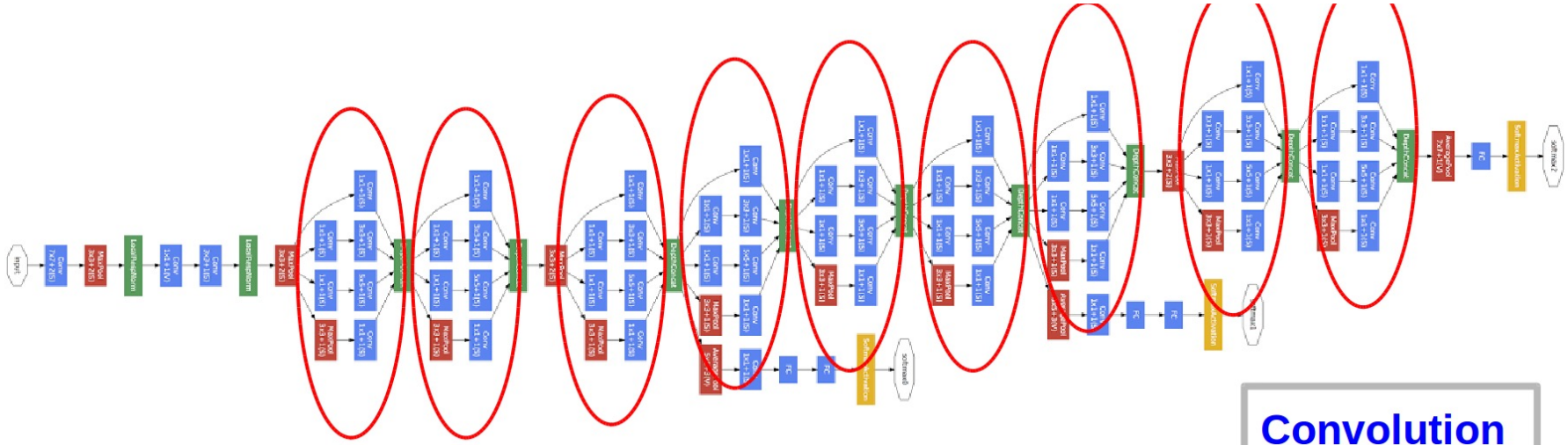
Computational Graphs



Simplified computational graph to denote neural networks

AlexNet



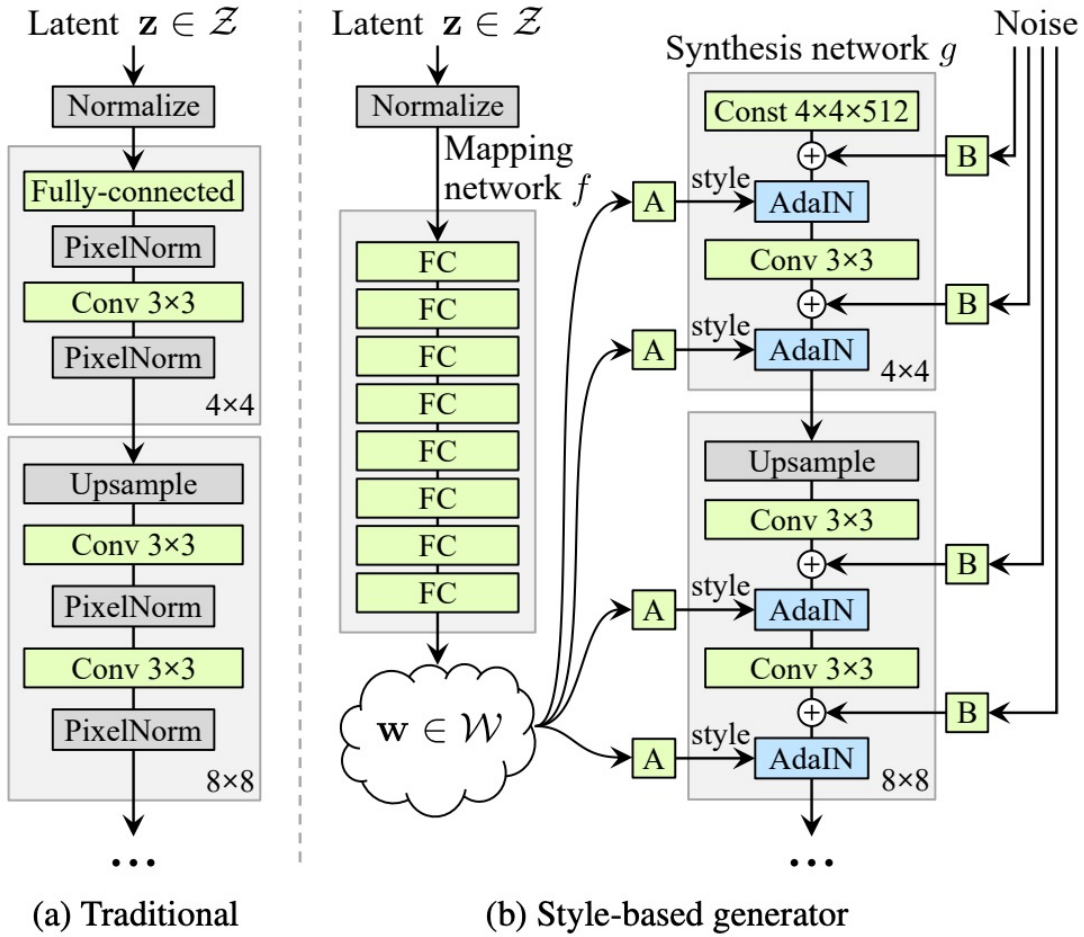


GoogleNet

Convolution
Pooling
Softmax
Concat/Normalize

Simplified computational graph to denote neural networks

StyleGAN



Transformer

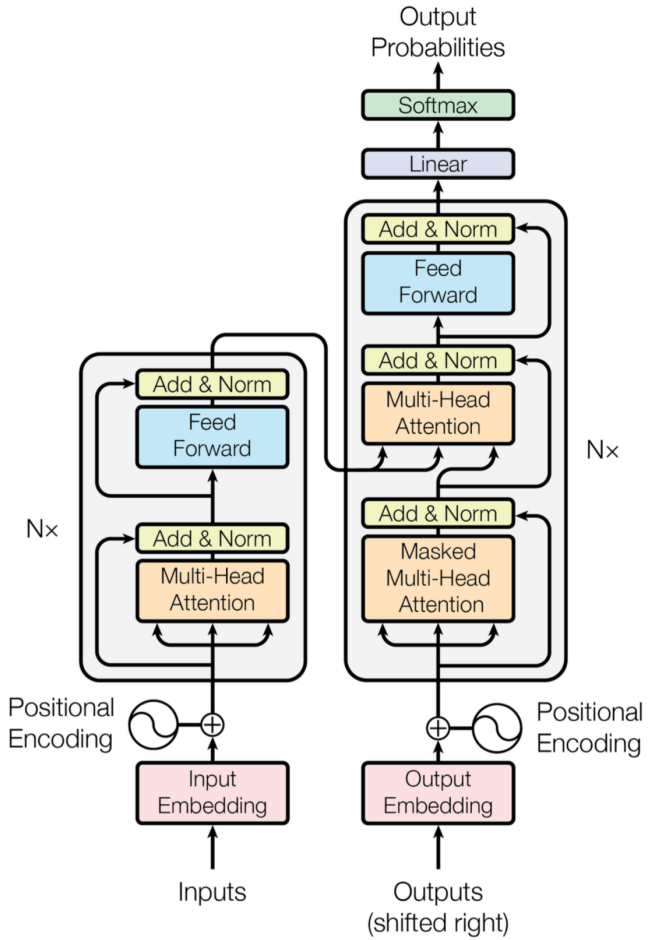
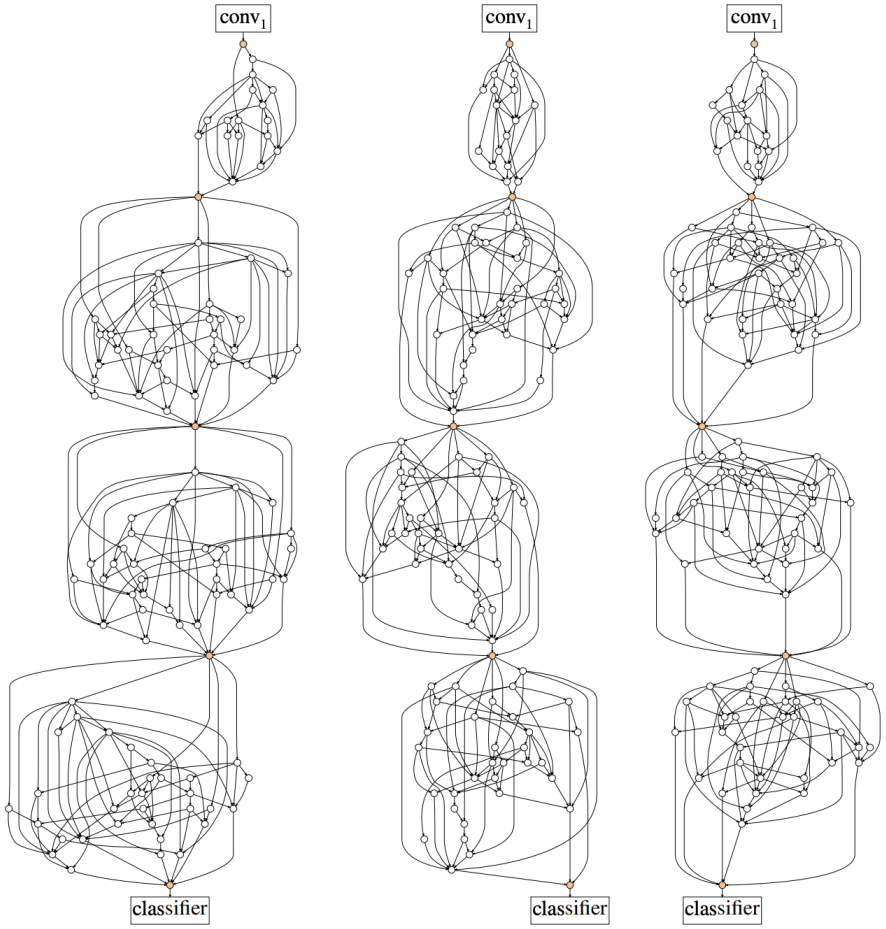


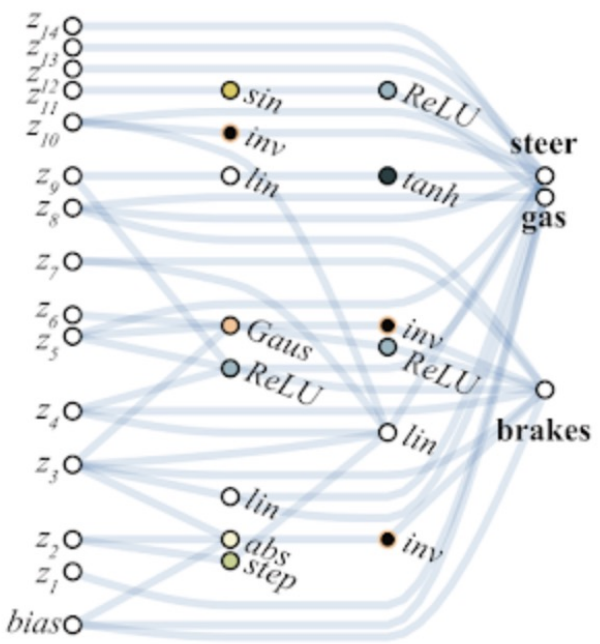
Figure 1: The Transformer - model architecture.

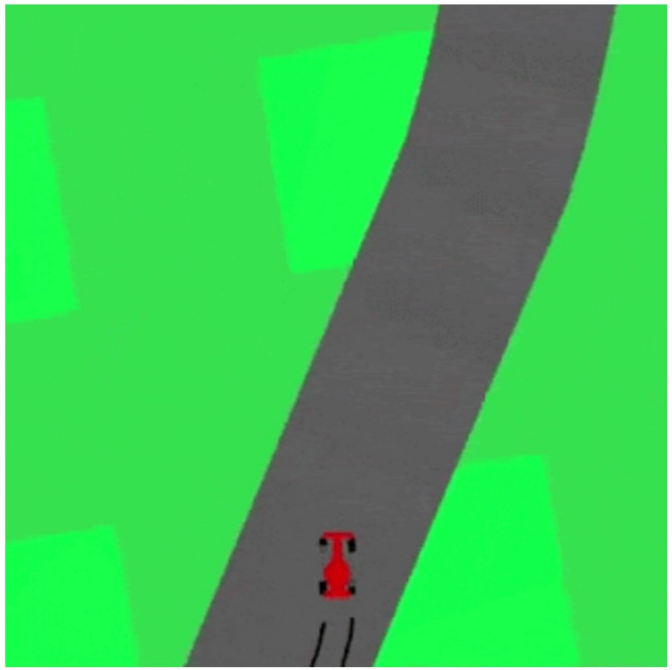
Simplified computational graph to denote neural networks



Xie et al. Random Wired Networks

Could be messy….





Weight agnostic neural networks by David Ha

Chain Rule

Suppose that functions $y = f(u)$ and $u = g(x)$ are both differentiable, then the chain rule states that

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

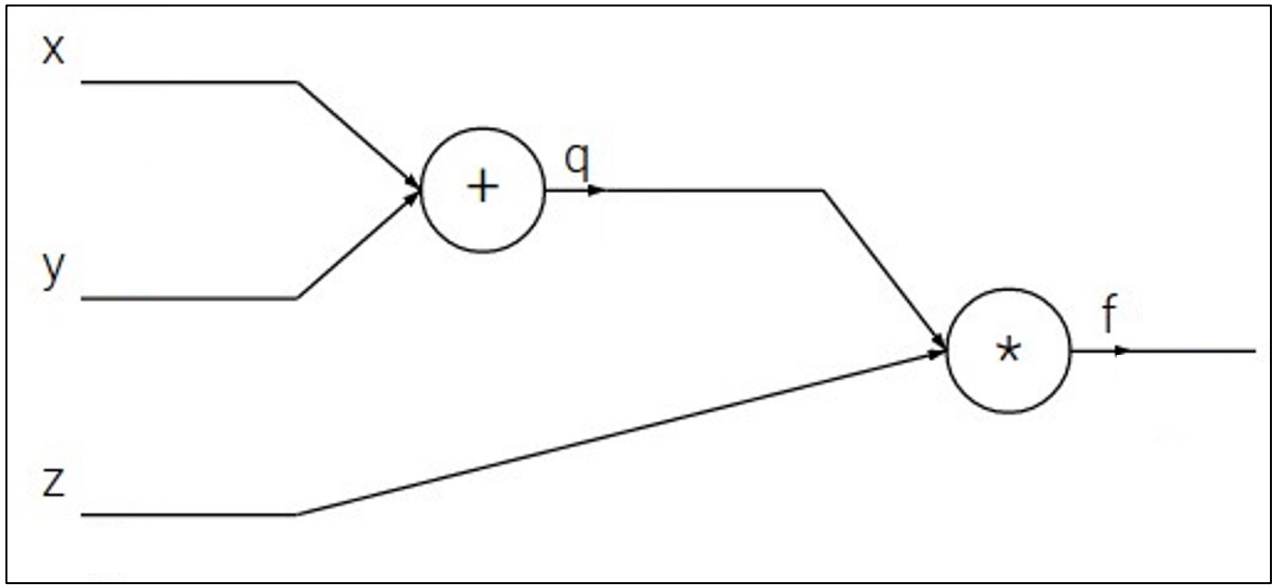
In more general scenario, suppose that y has variables $u_1, u_2, \dots, u_m$, where each differentiable function u_i has variables $x_1, x_2, \dots, x_n$. Then the chain rule gives

$$\frac{dy}{dx_i} = \frac{dy}{du_1}\frac{du_1}{dx_i} + \frac{dy}{du_2}\frac{du_2}{dx_i} + \dots + \frac{dy}{du_m}\frac{du_m}{dx_i}$$

for any $i = 1,2, \dots, n$

Backpropagation: Simple Example

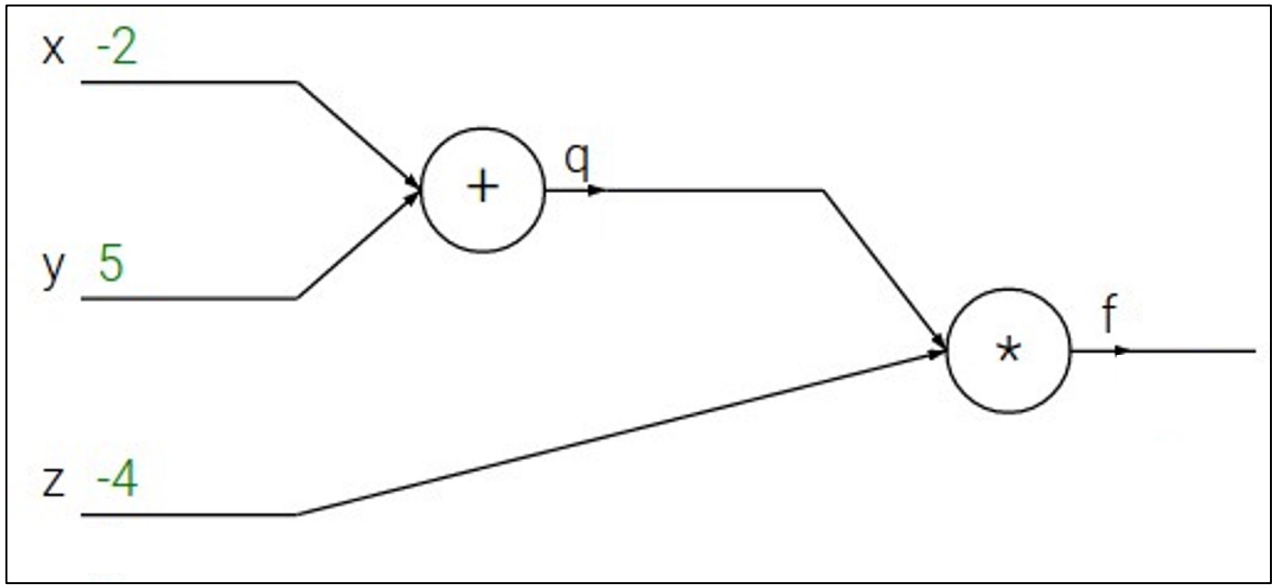
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

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

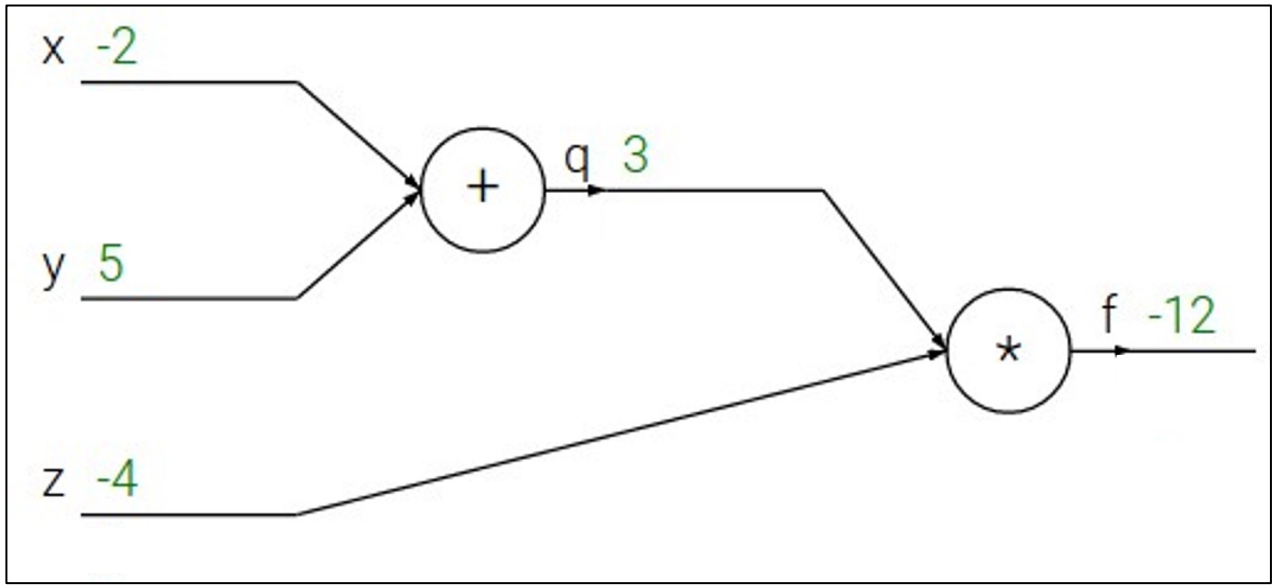
e.g. x = -2, y = 5, z = -4



Backpropagation: Simple Example

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

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



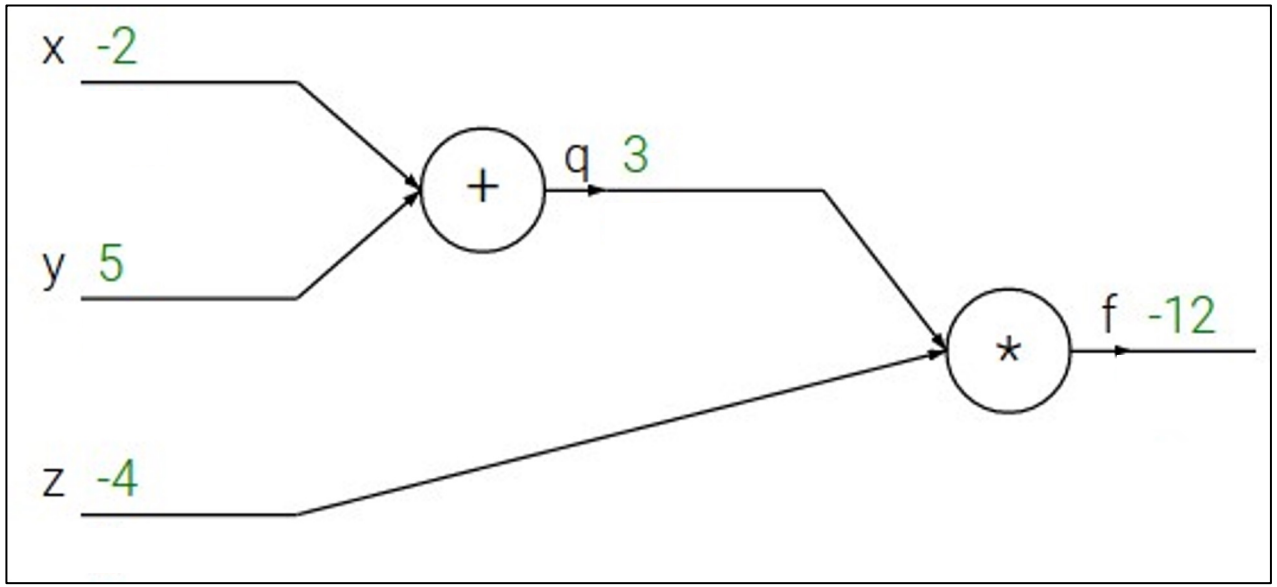
1. Forward pass: Compute outputs

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

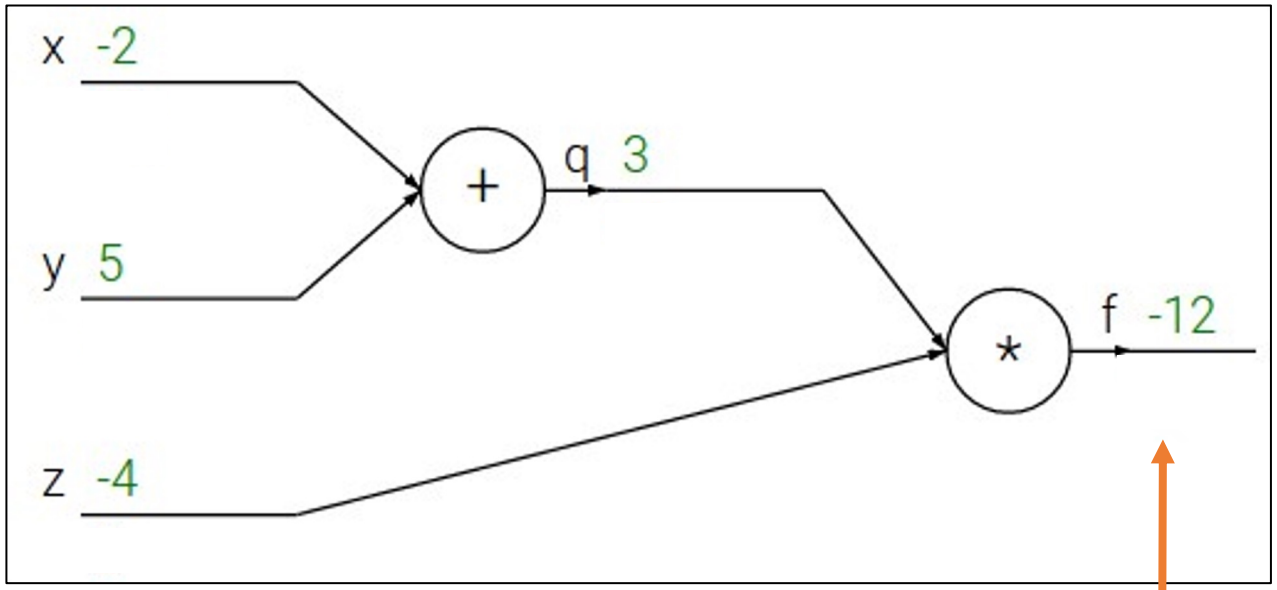
2. Backward pass: Compute derivatives

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

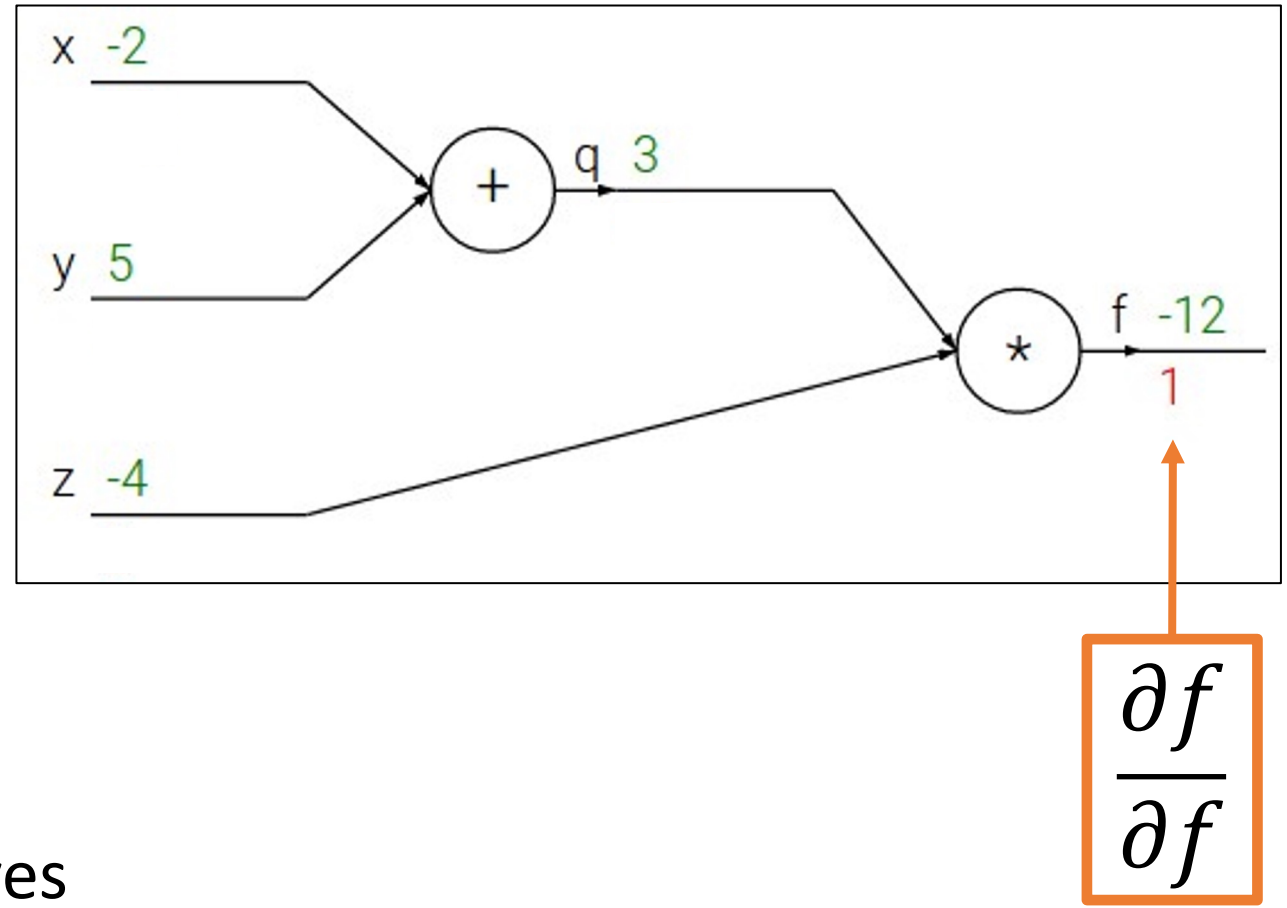
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

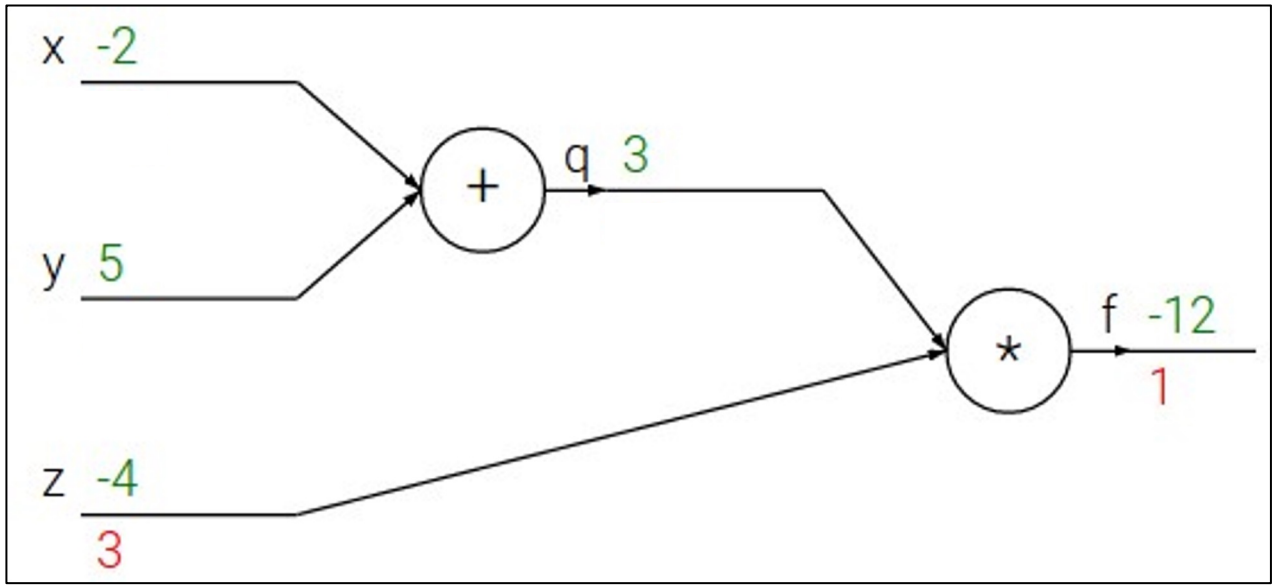
2. Backward pass: Compute derivatives

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

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

2. Backward pass: Compute derivatives

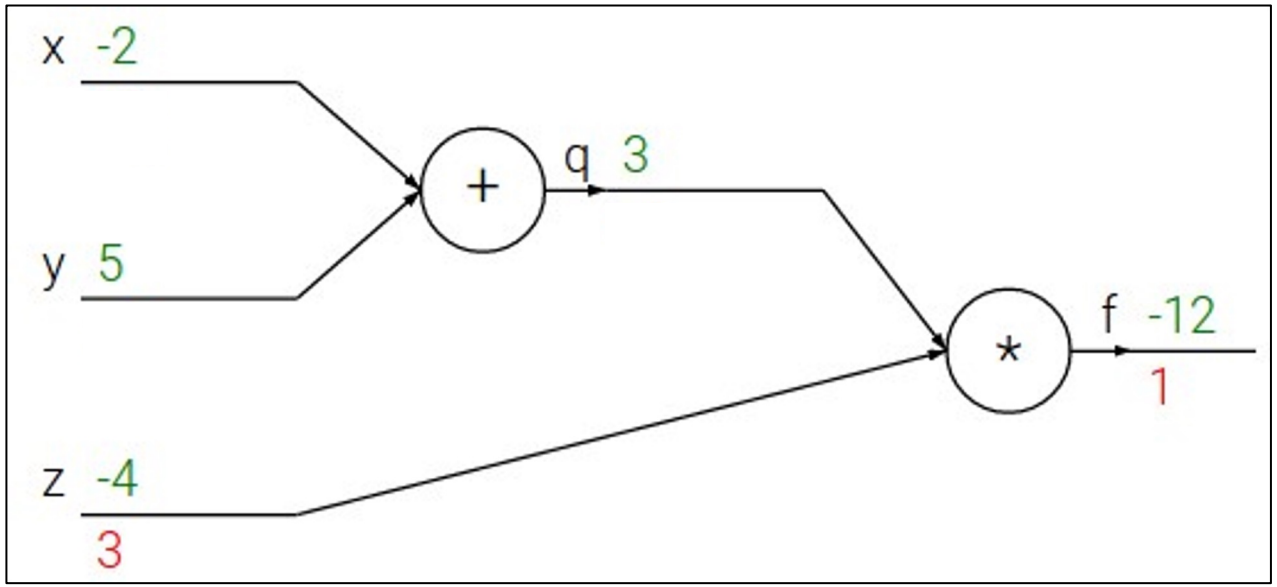
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

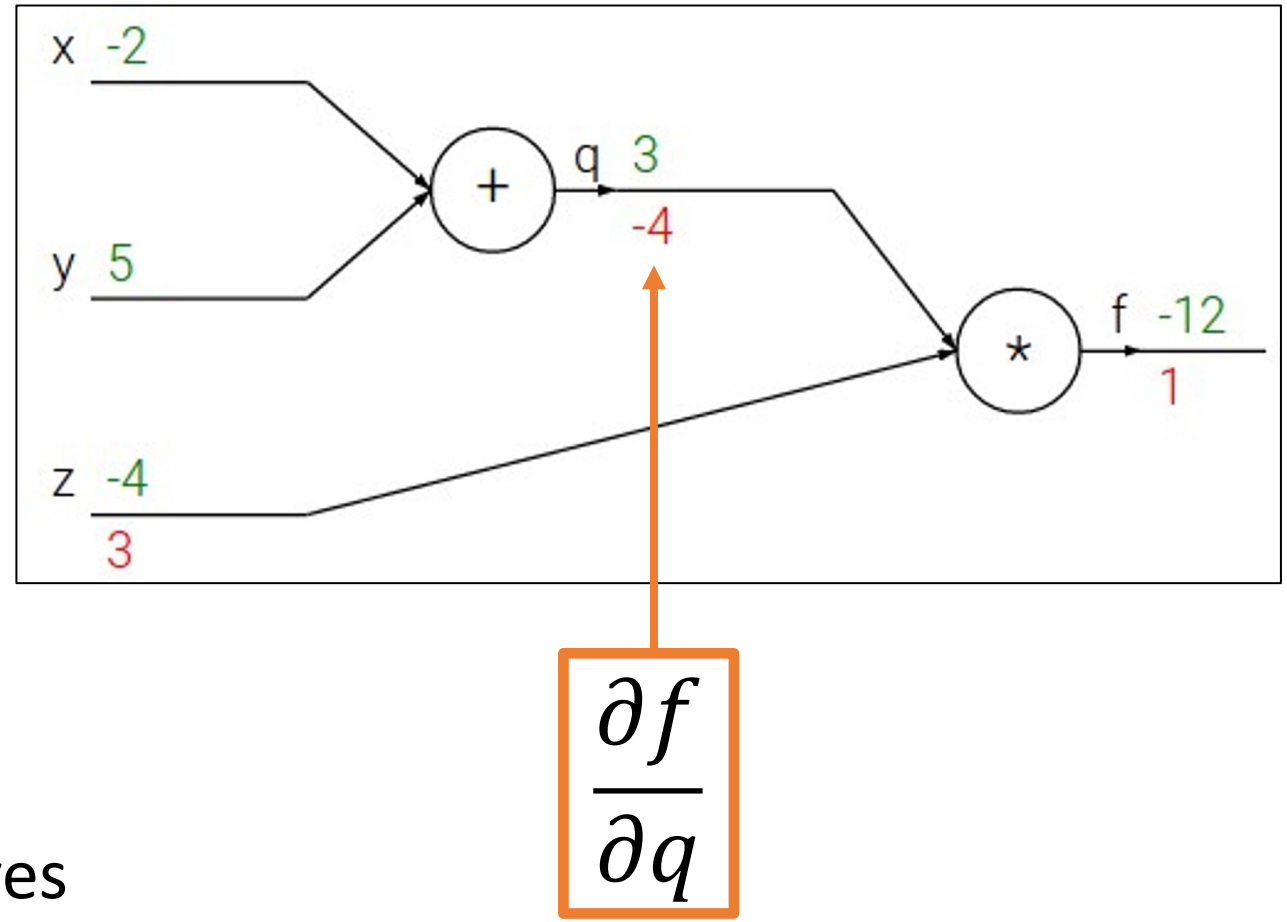
$$\frac{\partial f}{\partial z} = q$$

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

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

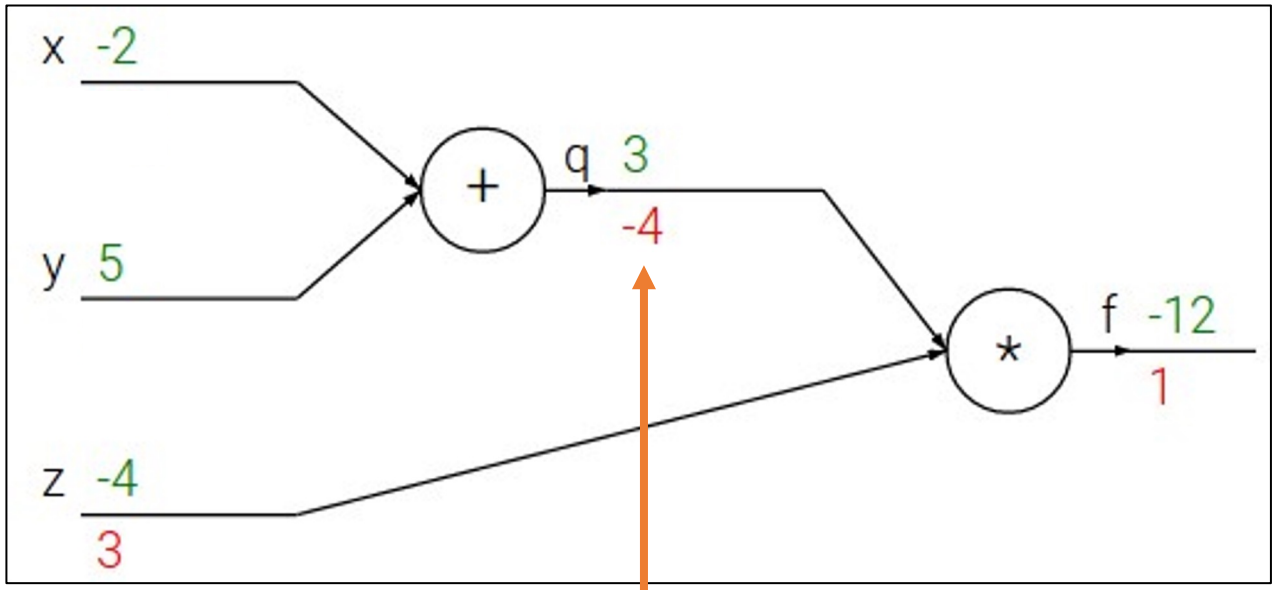
2. Backward pass: Compute derivatives

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

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

2. Backward pass: Compute derivatives

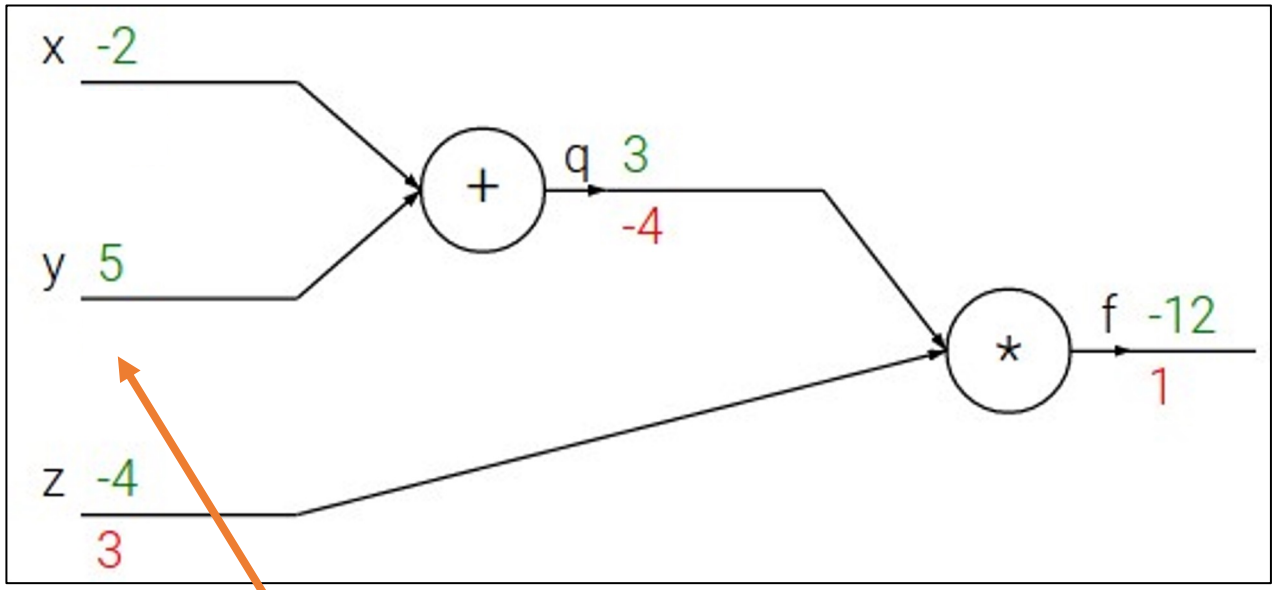
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

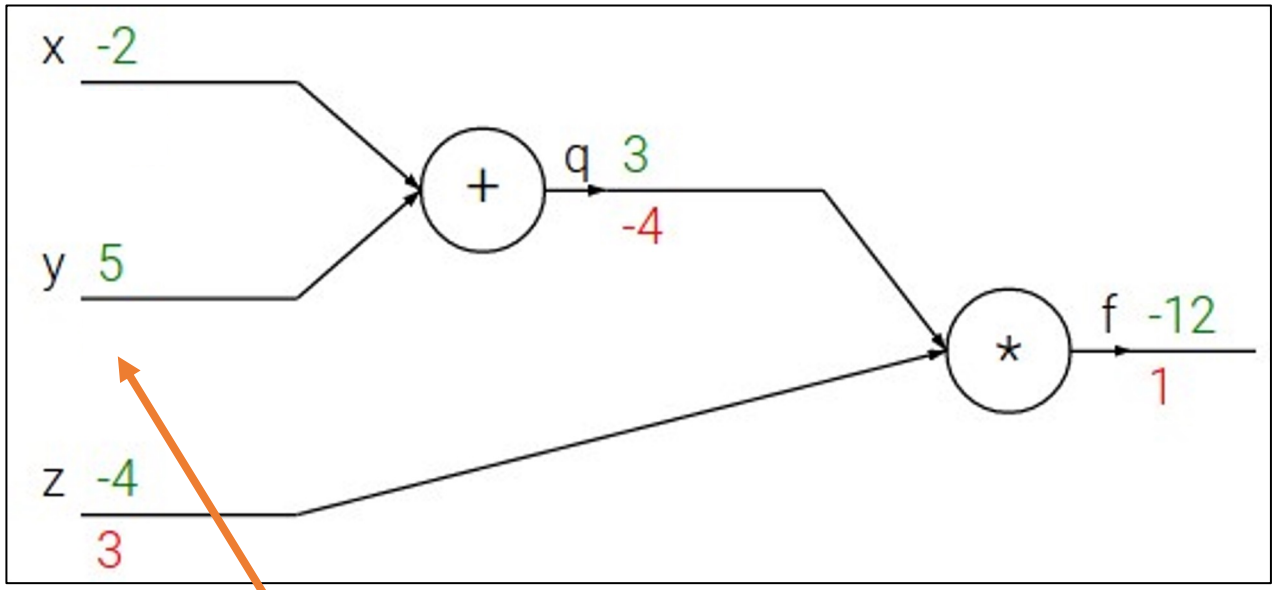
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

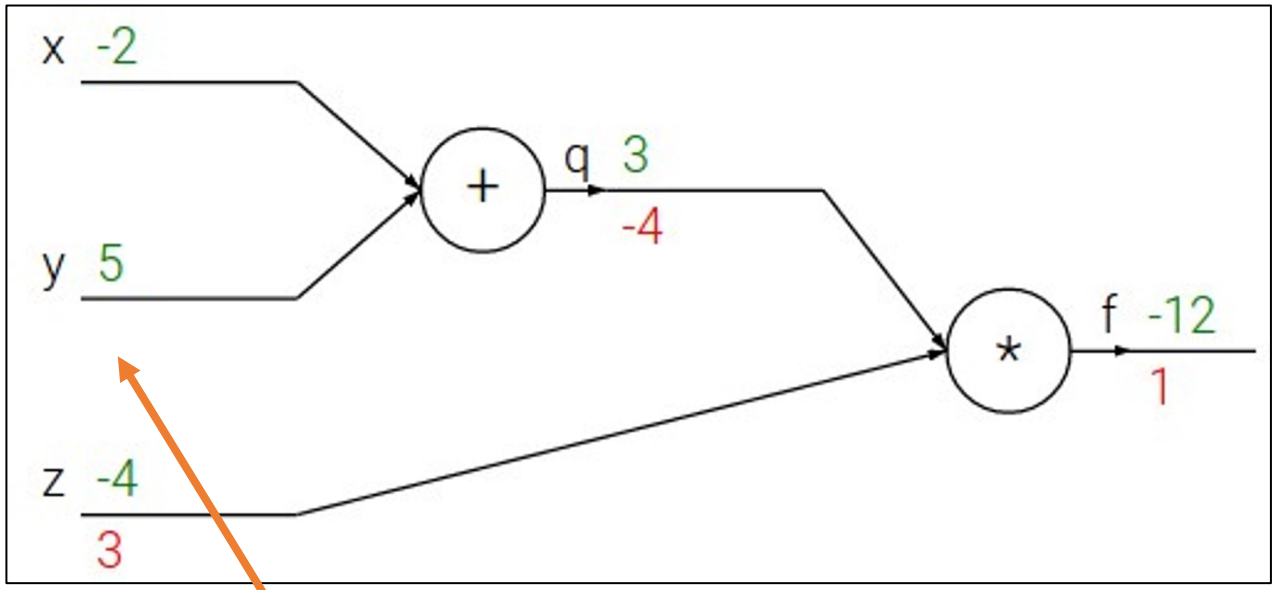
Chain Rule

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

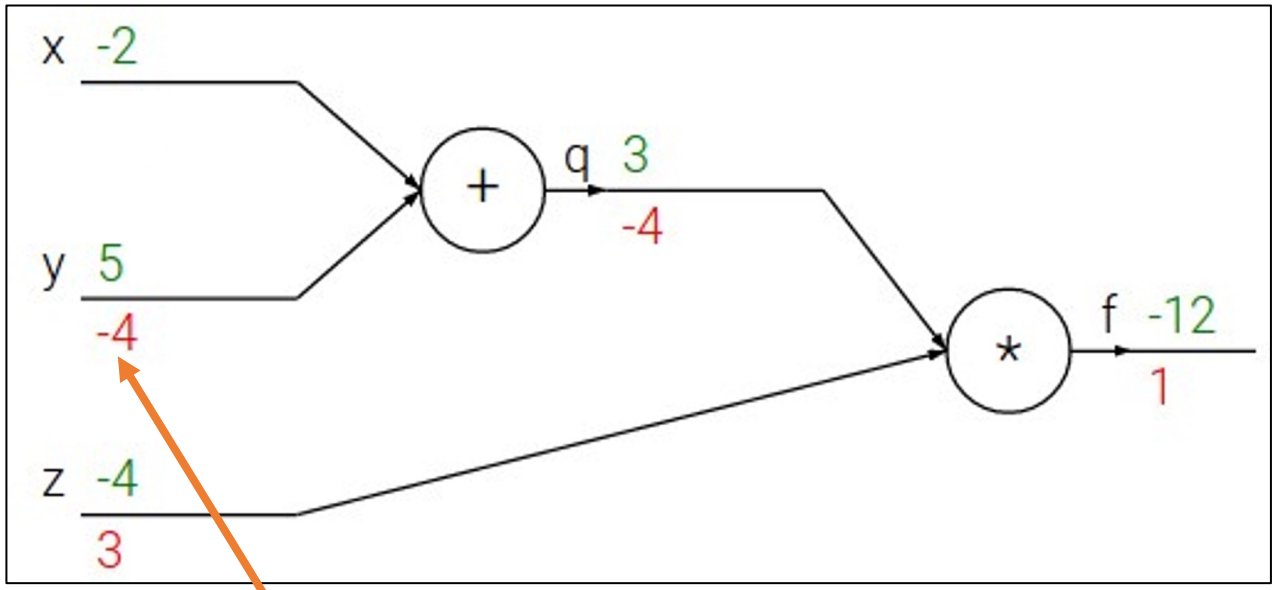
$$\frac{\partial q}{\partial y} = 1$$

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

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

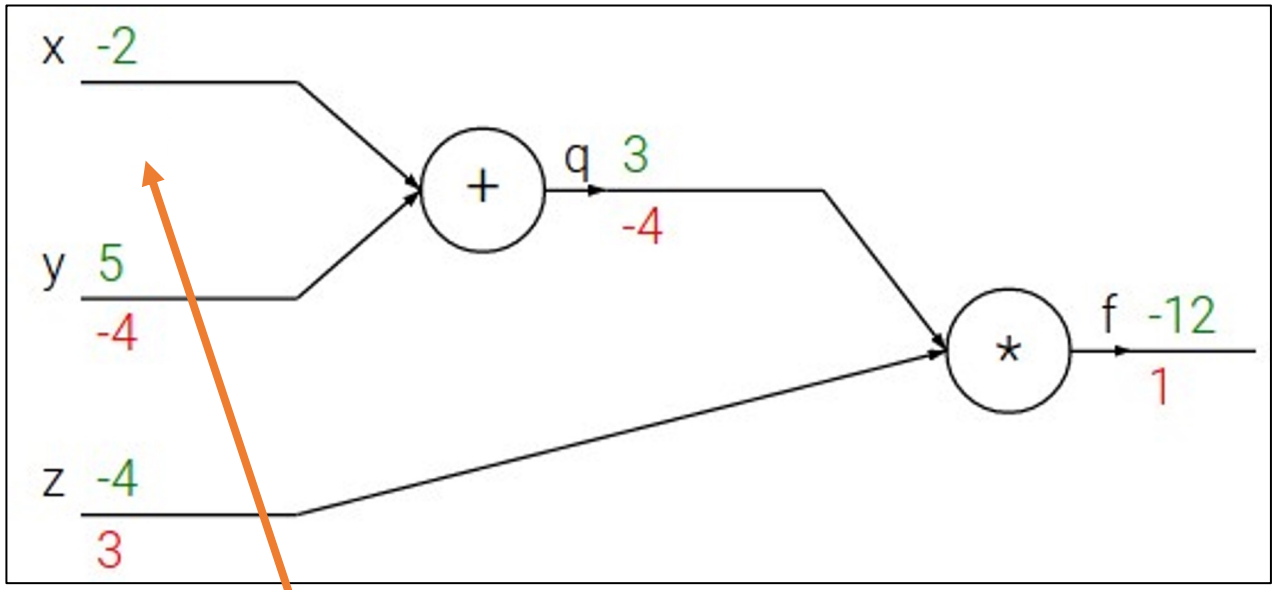
$$\frac{\partial q}{\partial y} = 1$$

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

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

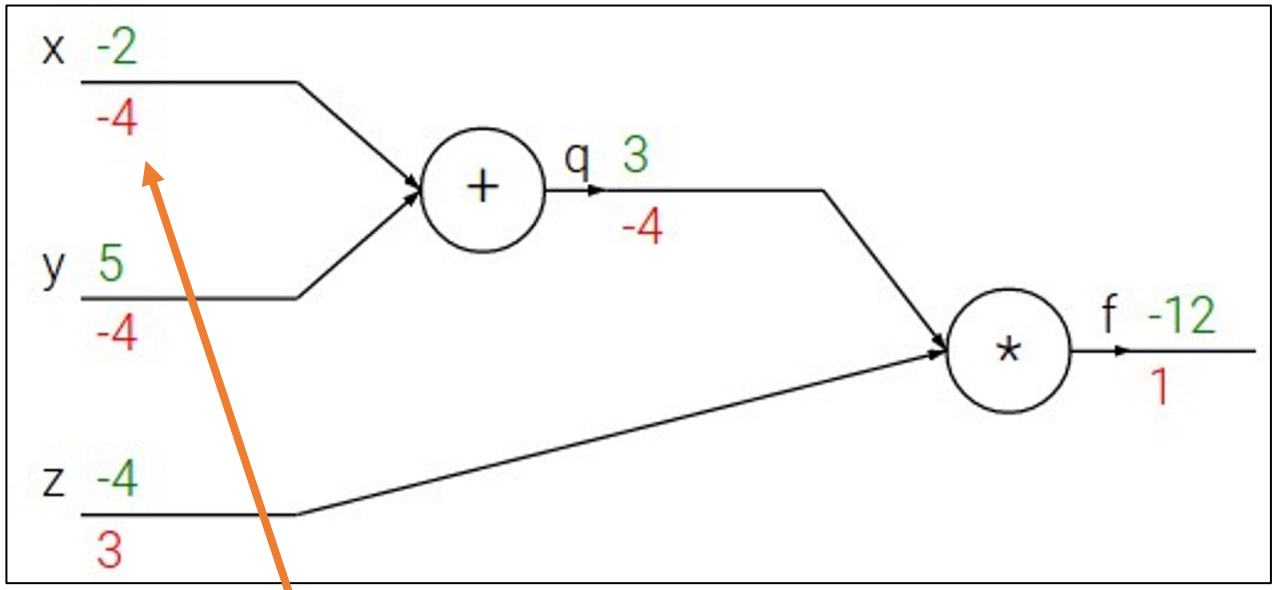
$$\frac{\partial q}{\partial x} = 1$$

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

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

2. Backward pass: Compute derivatives

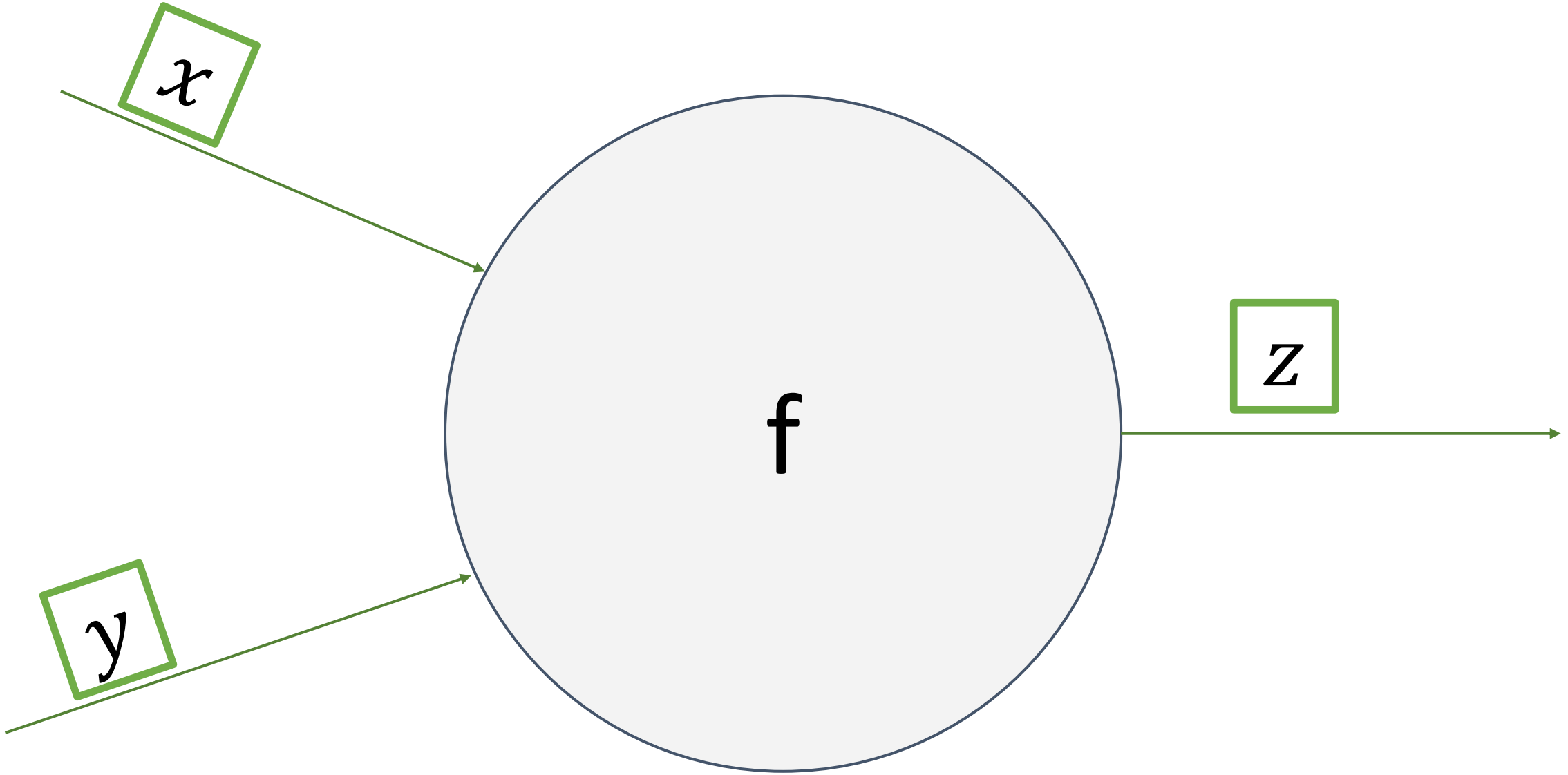
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

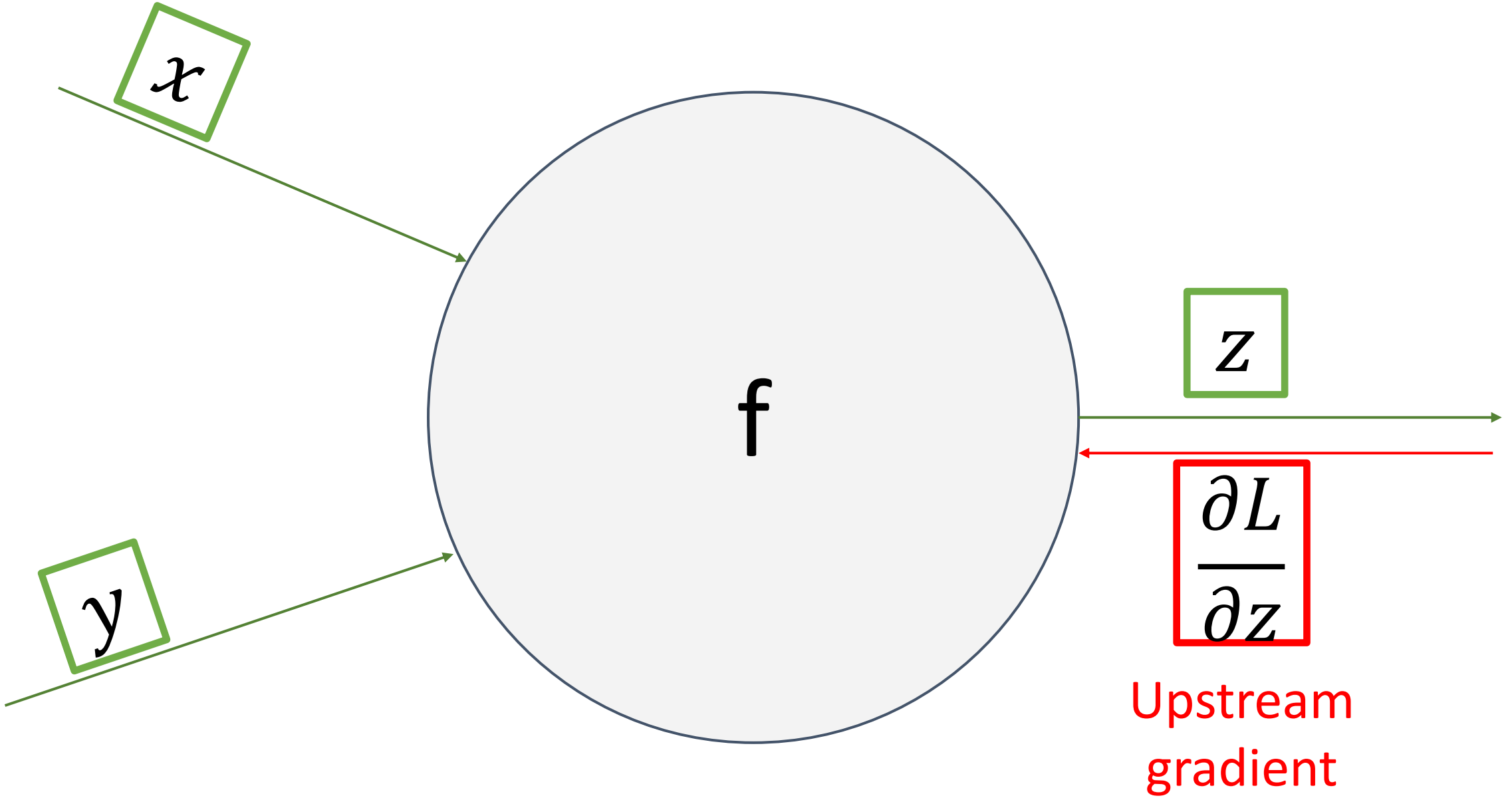
Chain Rule

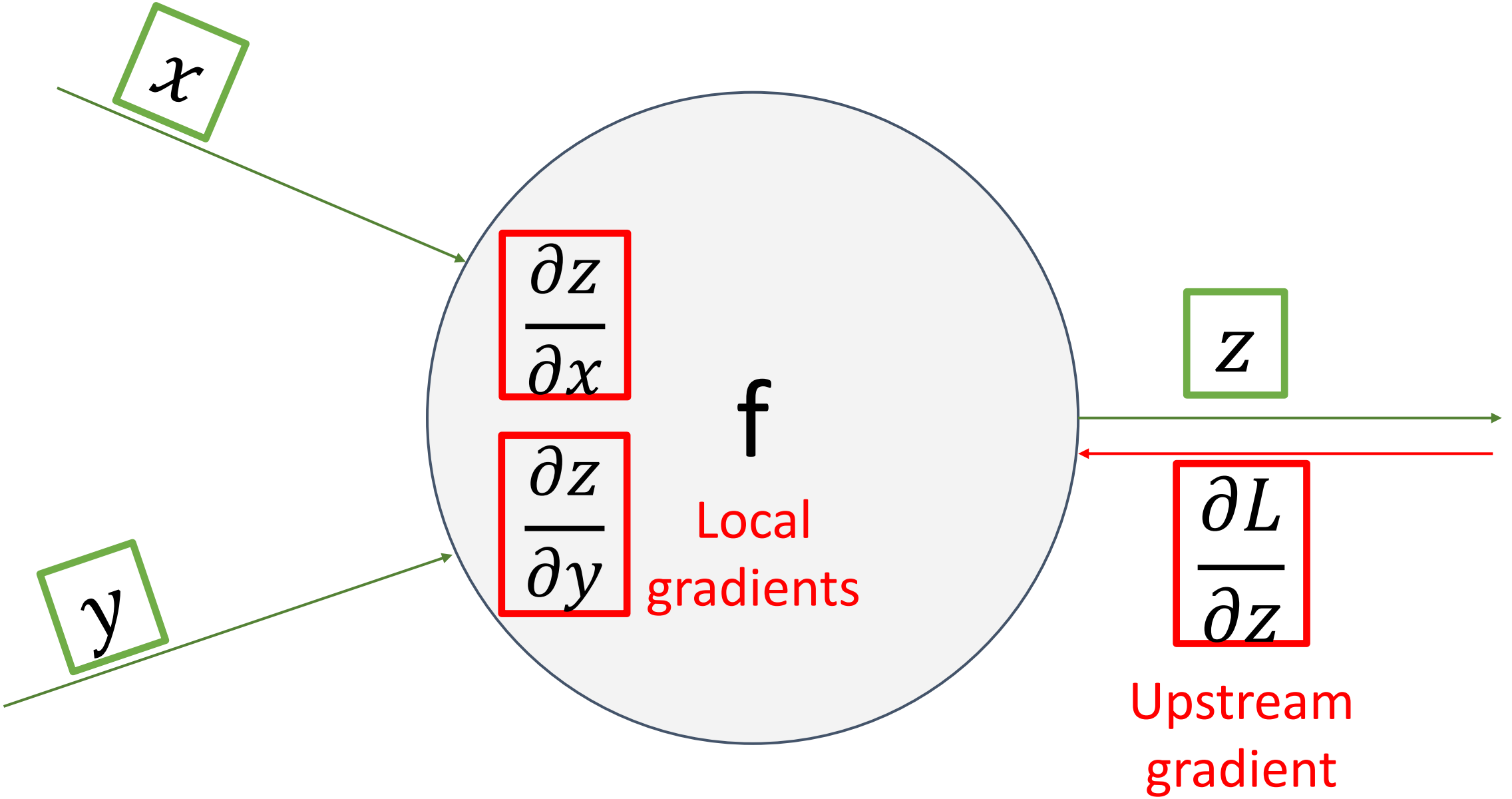
$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

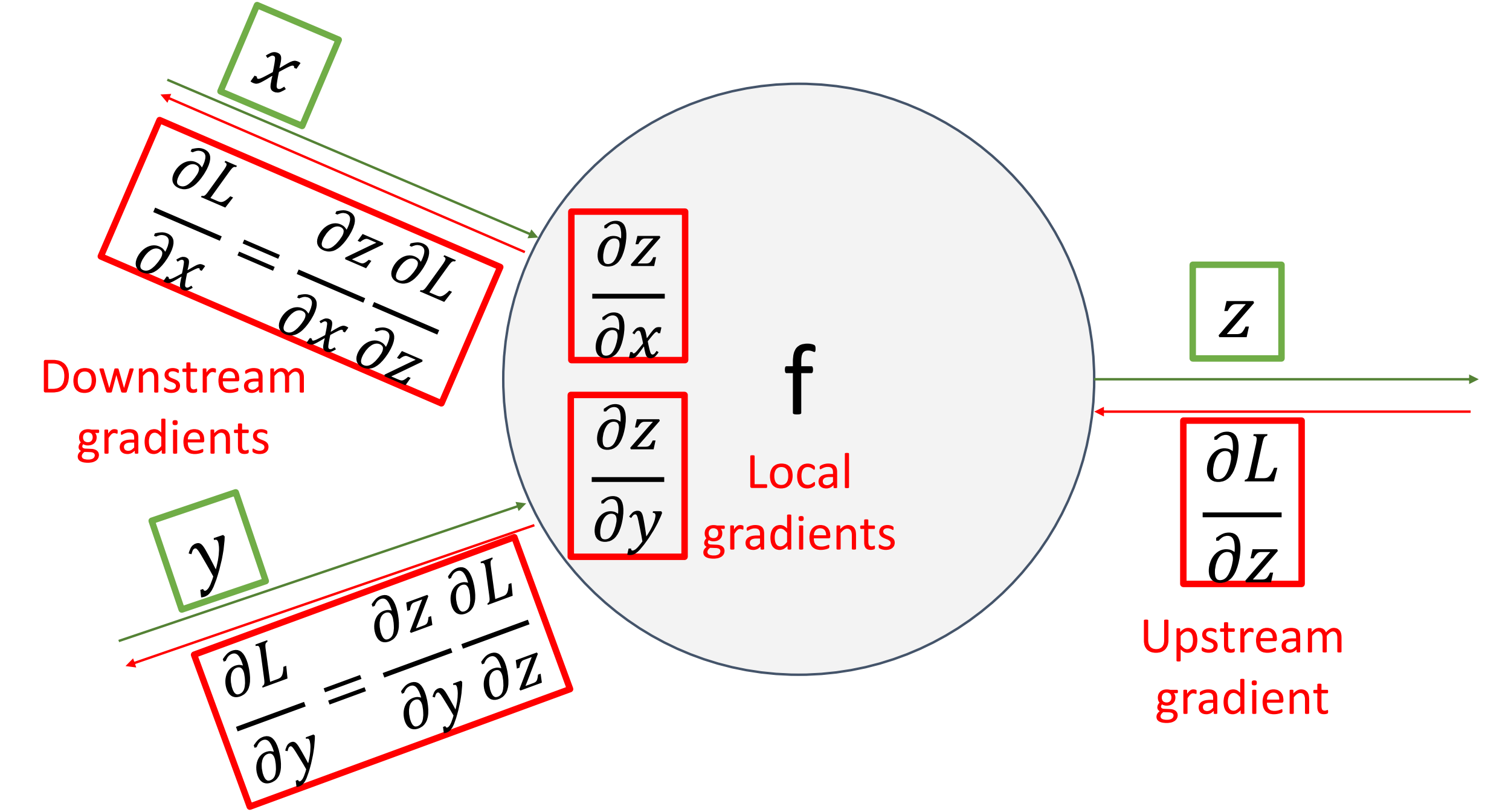
$$\frac{\partial q}{\partial x} = 1$$

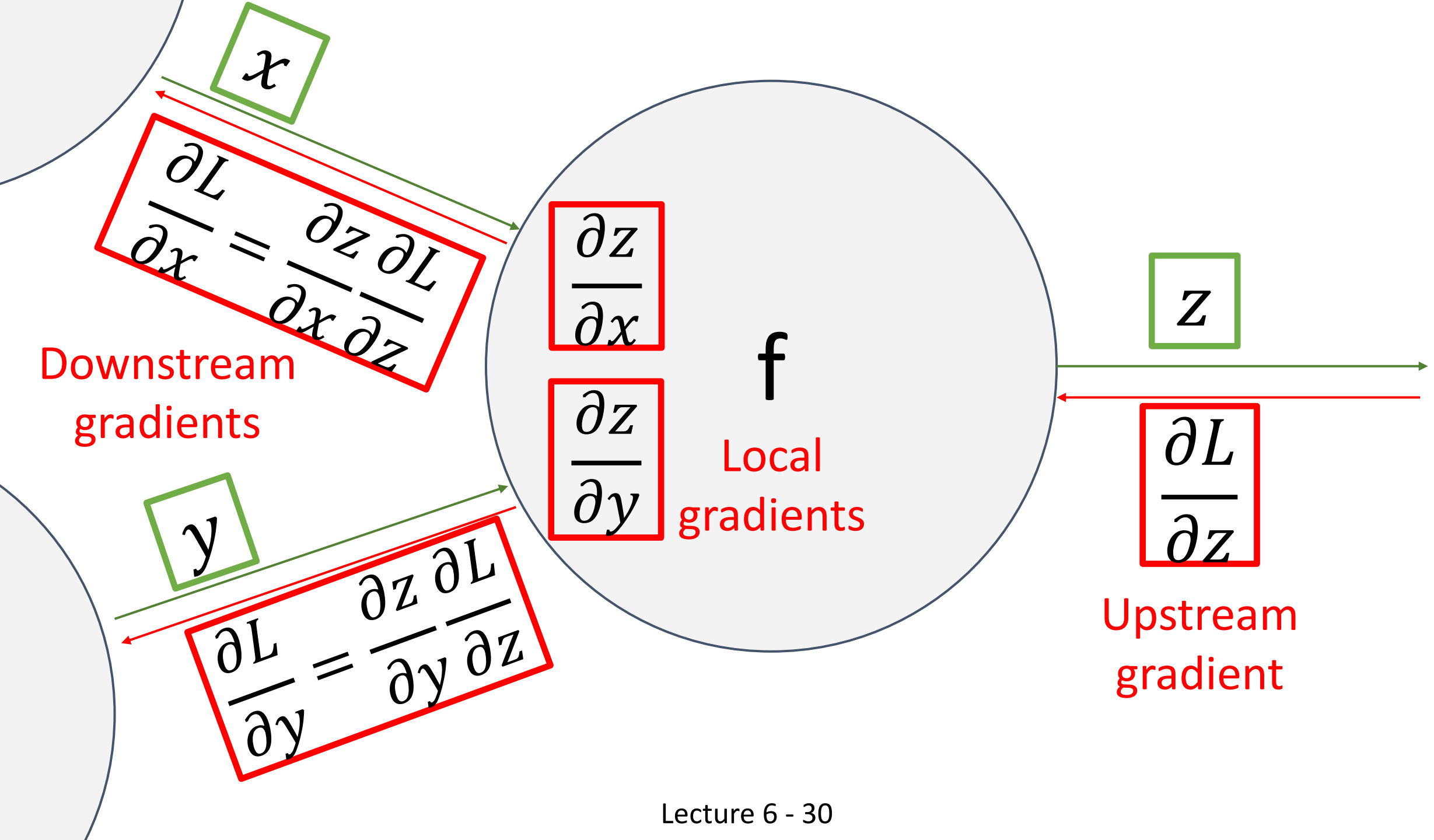
Downstream Gradient **Local Gradient** **Upstream Gradient**


x
y
f
z

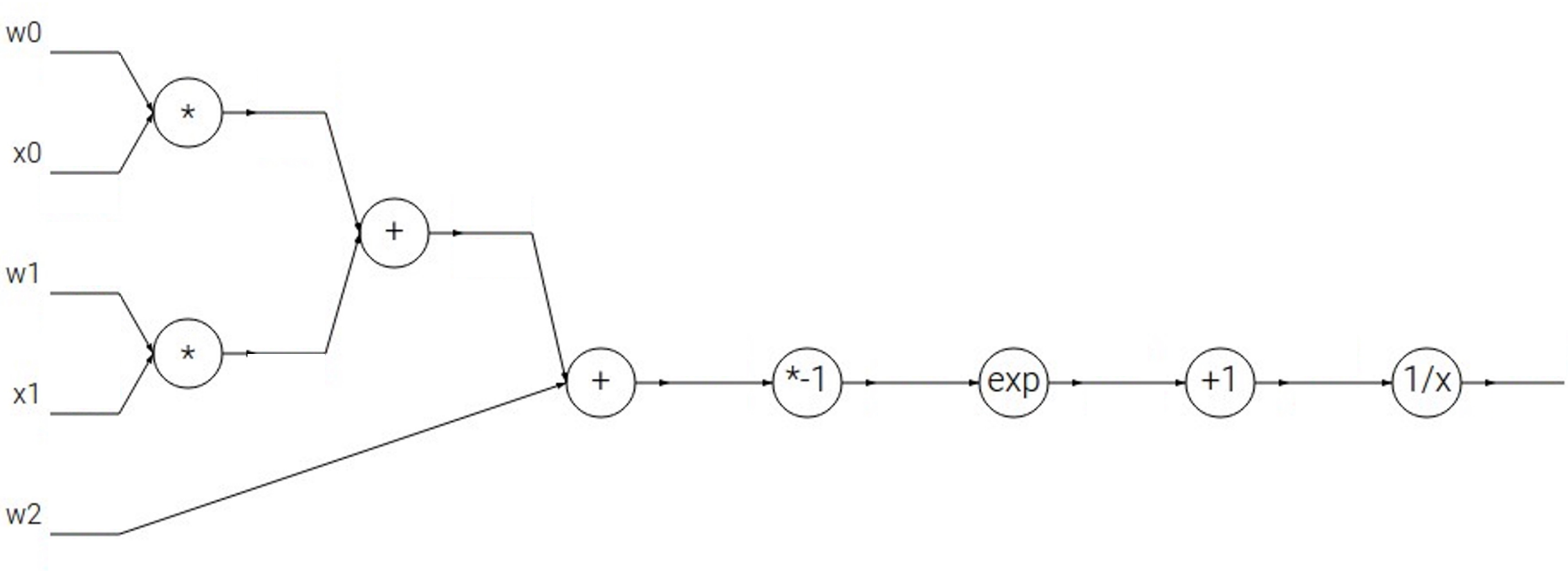

x
y
f
z
$\frac{\partial L}{\partial z}$
Upstream gradient


x
y
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient


x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient

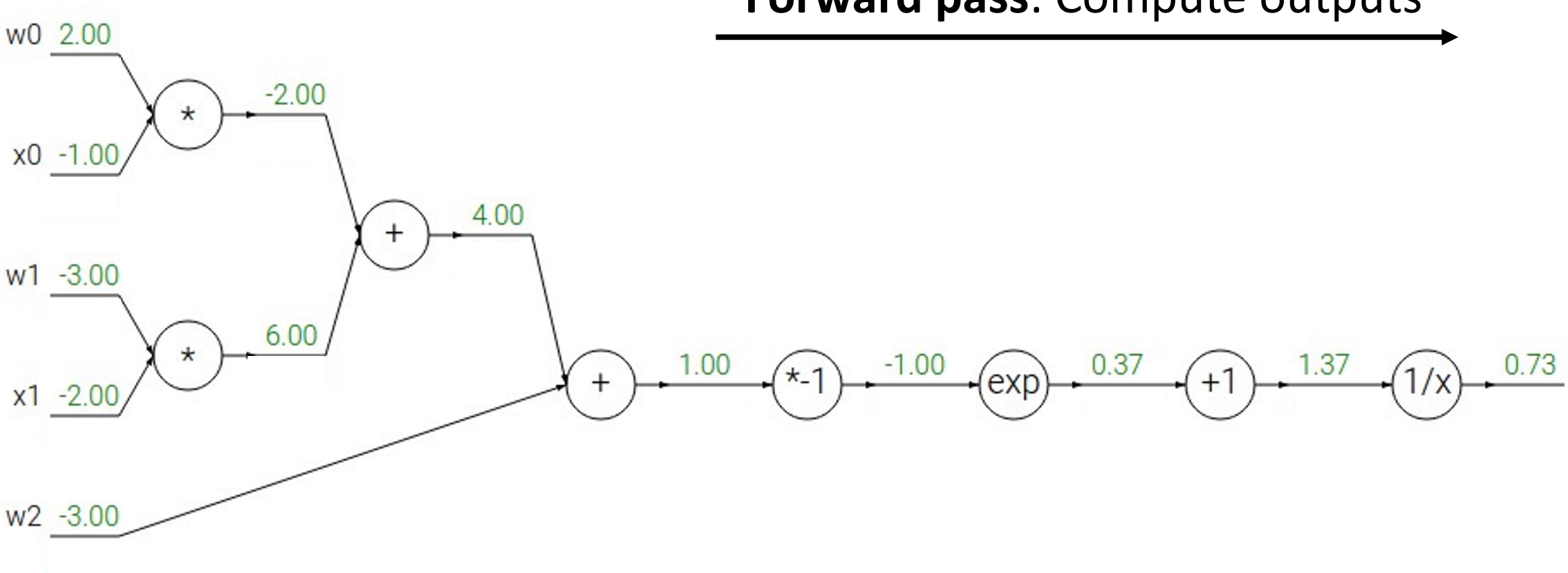

x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient

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



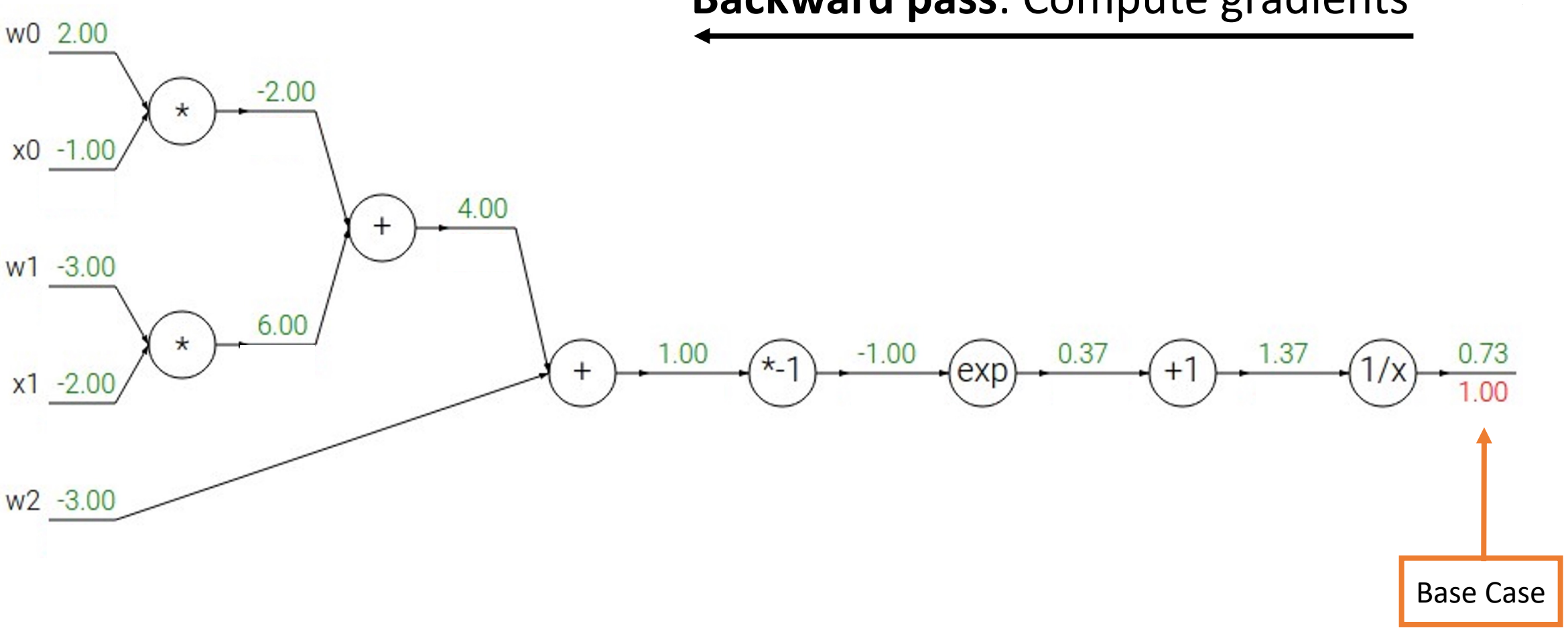
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Forward pass: Compute outputs



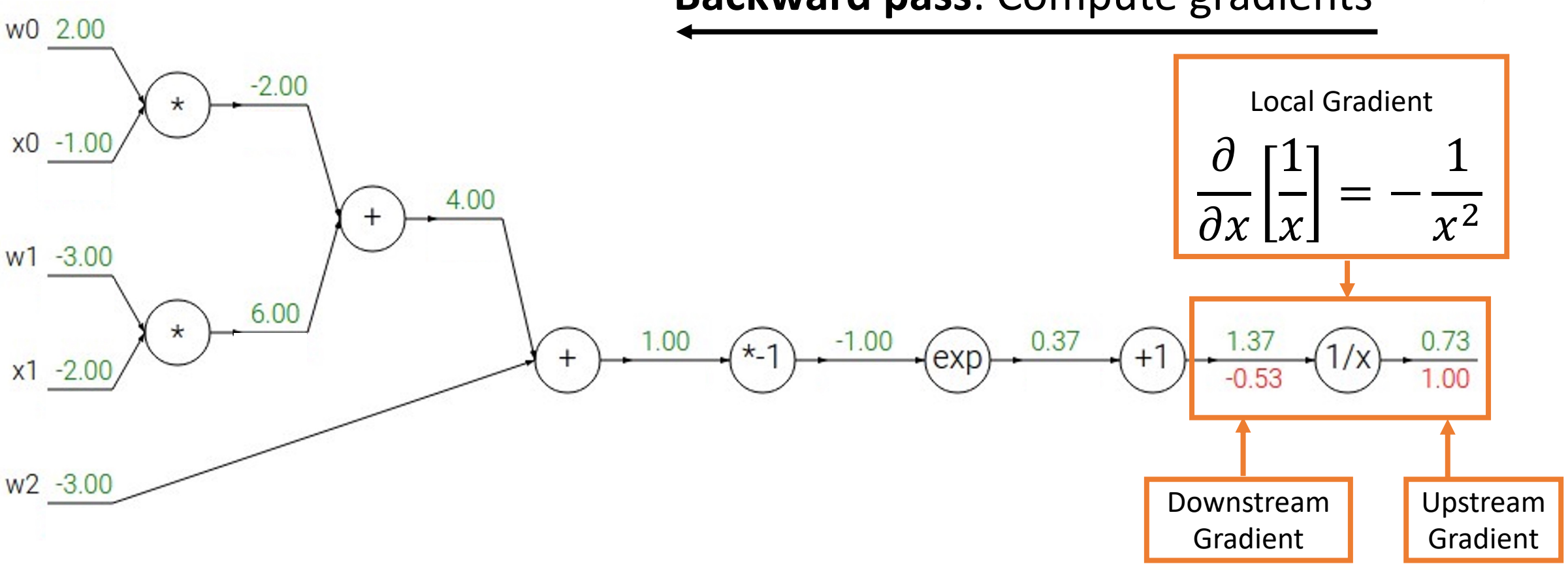
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



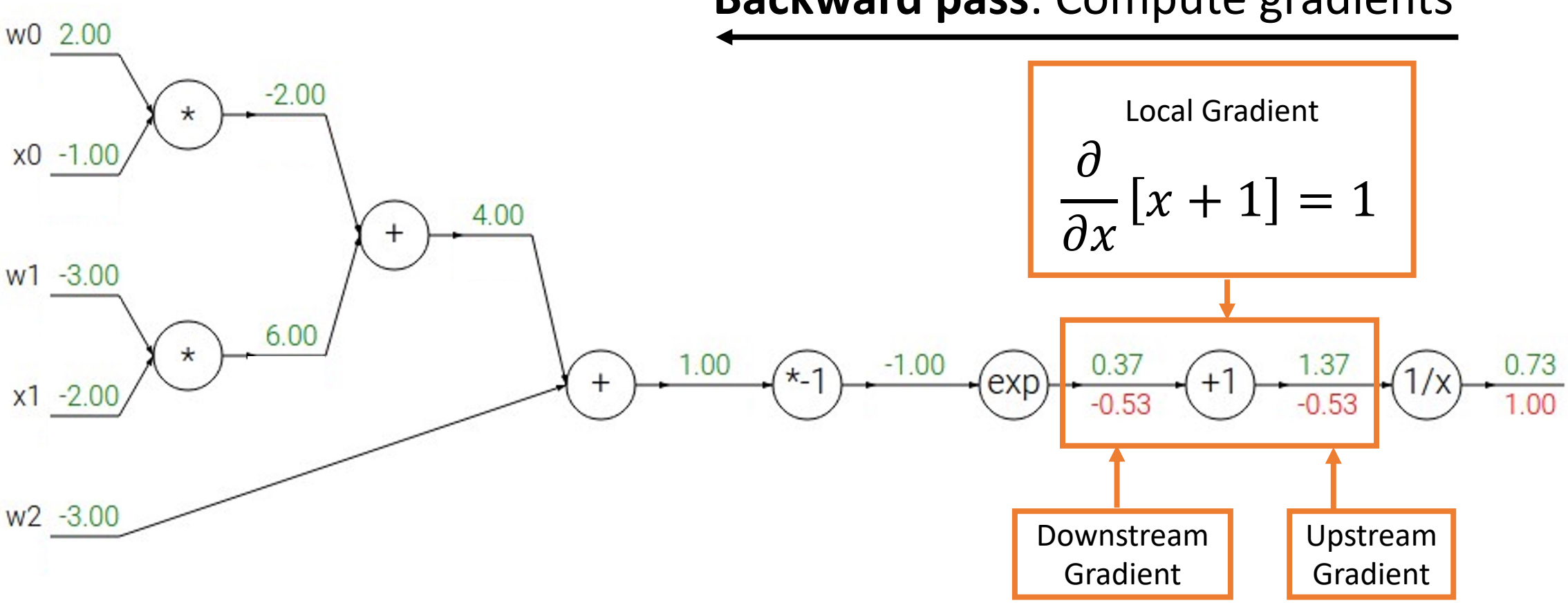
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



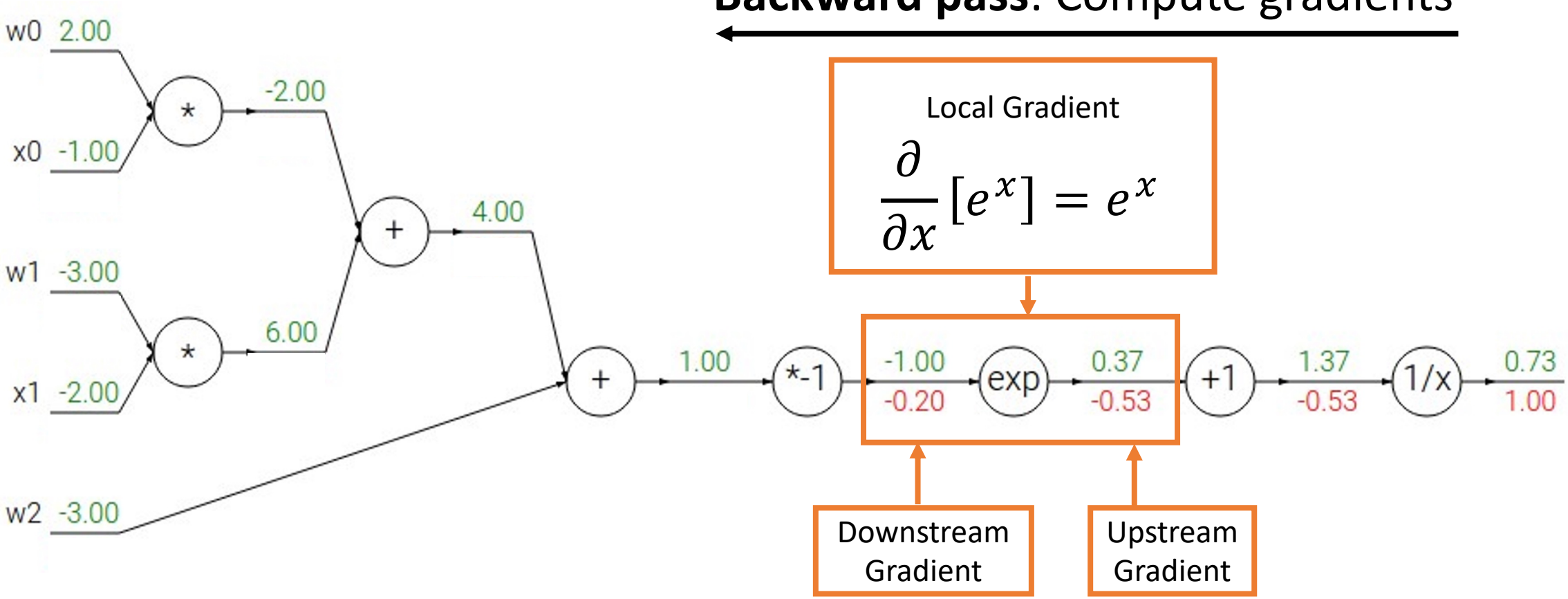
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



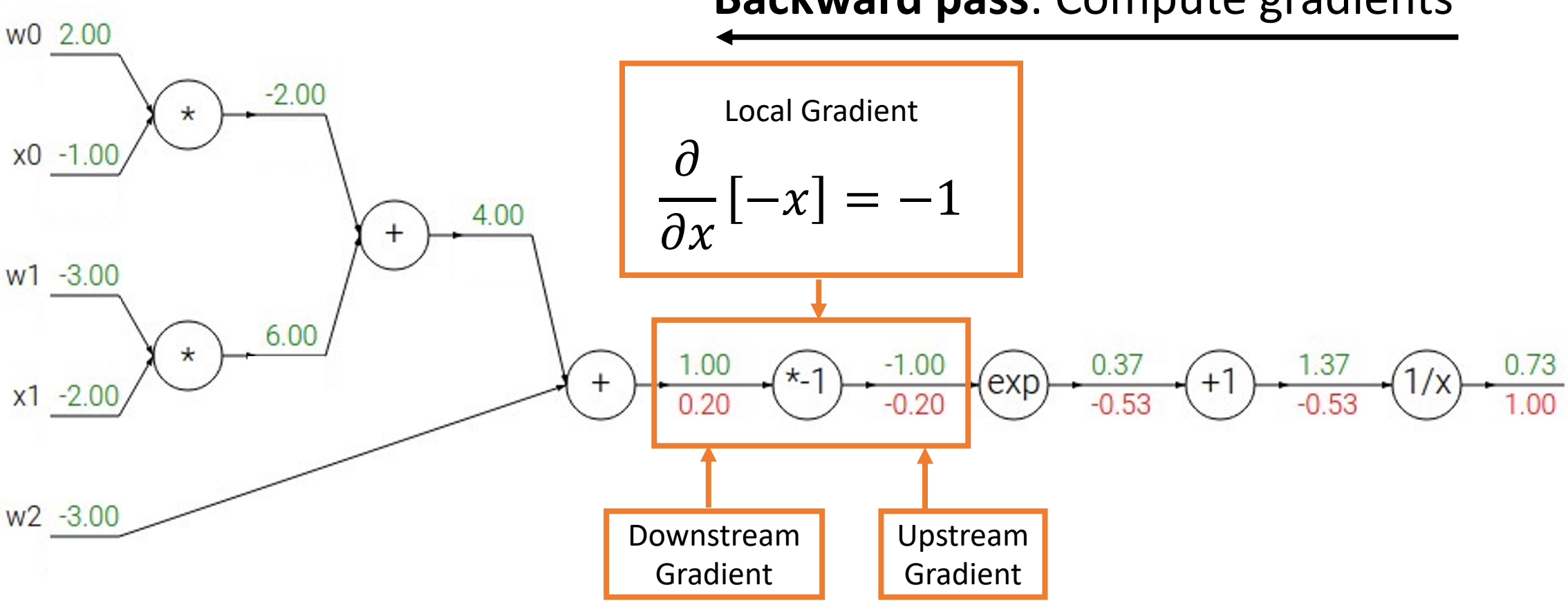
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



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

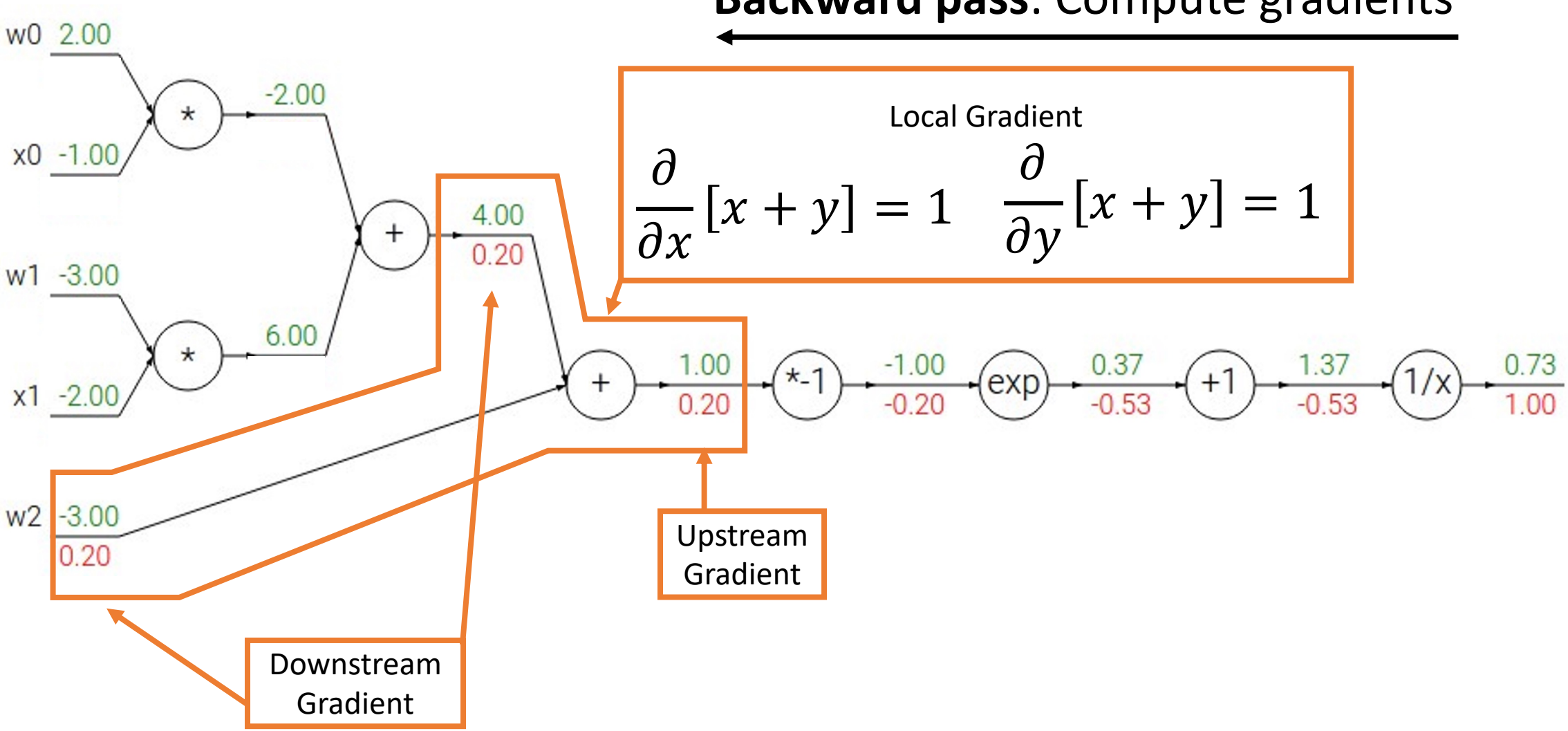
Backward pass: Compute gradients



Another Example

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

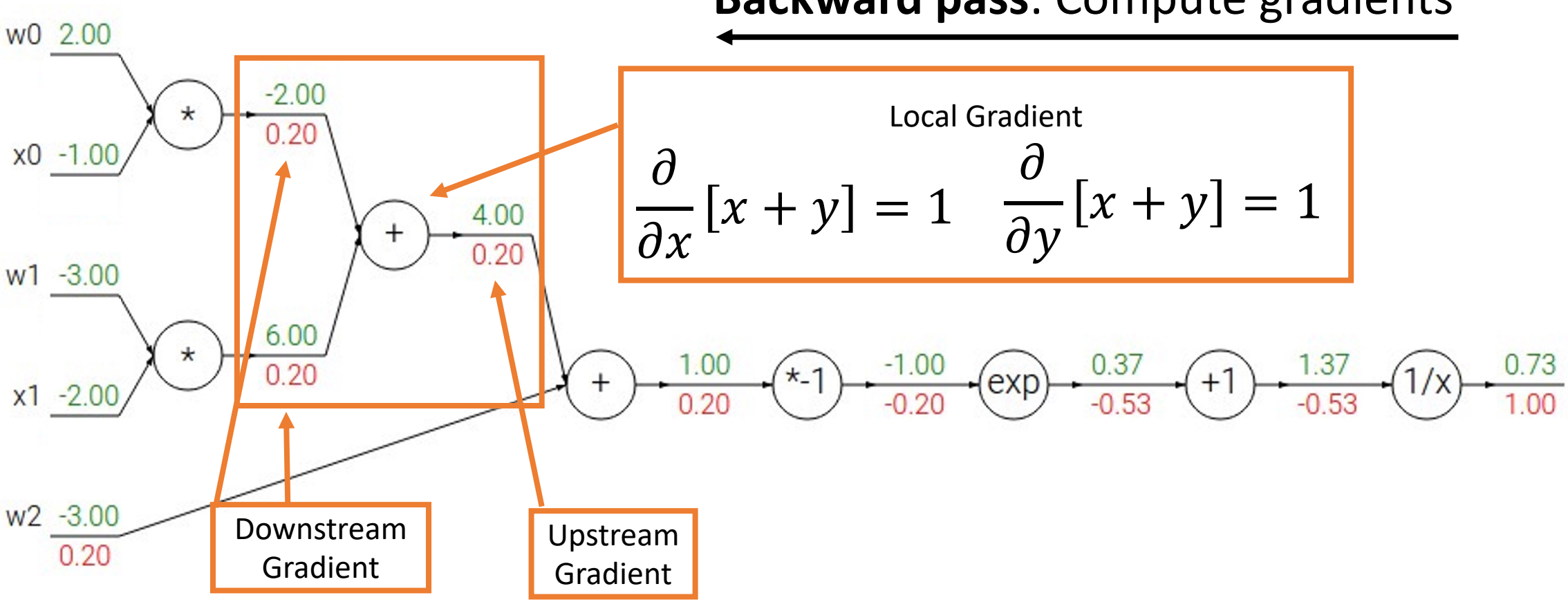
Backward pass: Compute gradients



Another Example

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

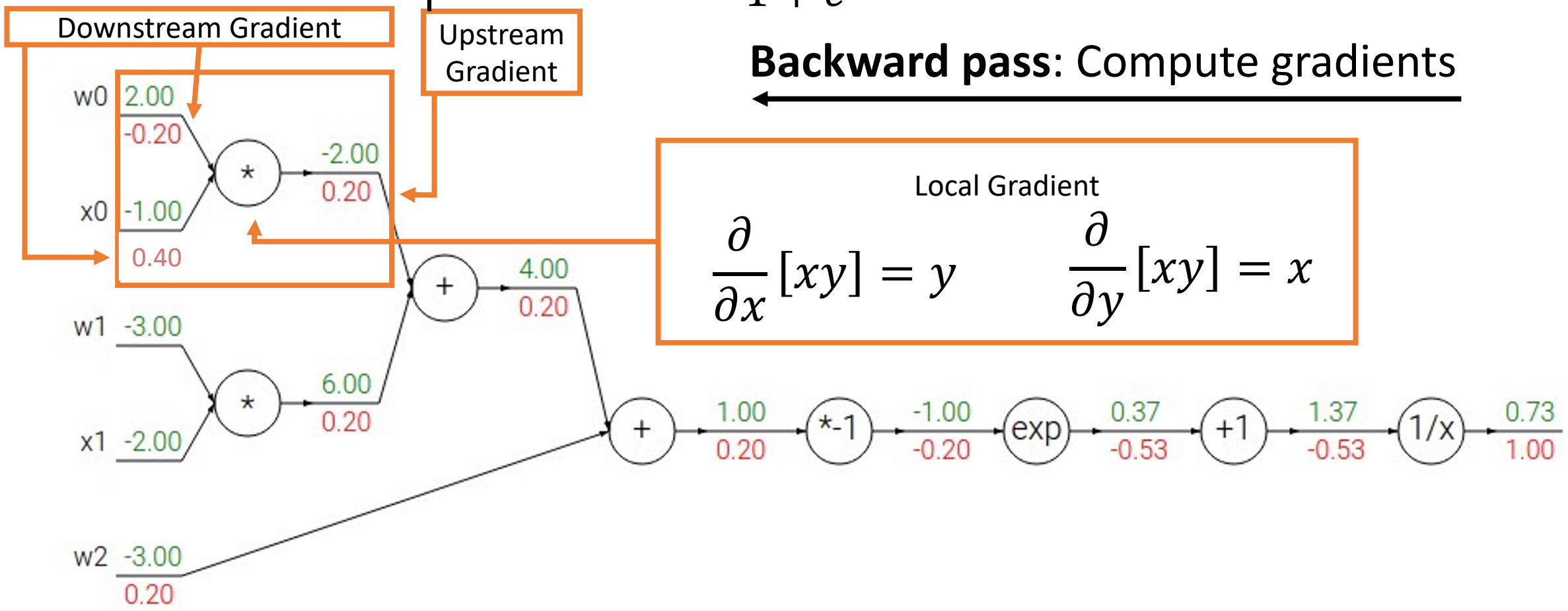
Backward pass: Compute gradients



Another Example

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

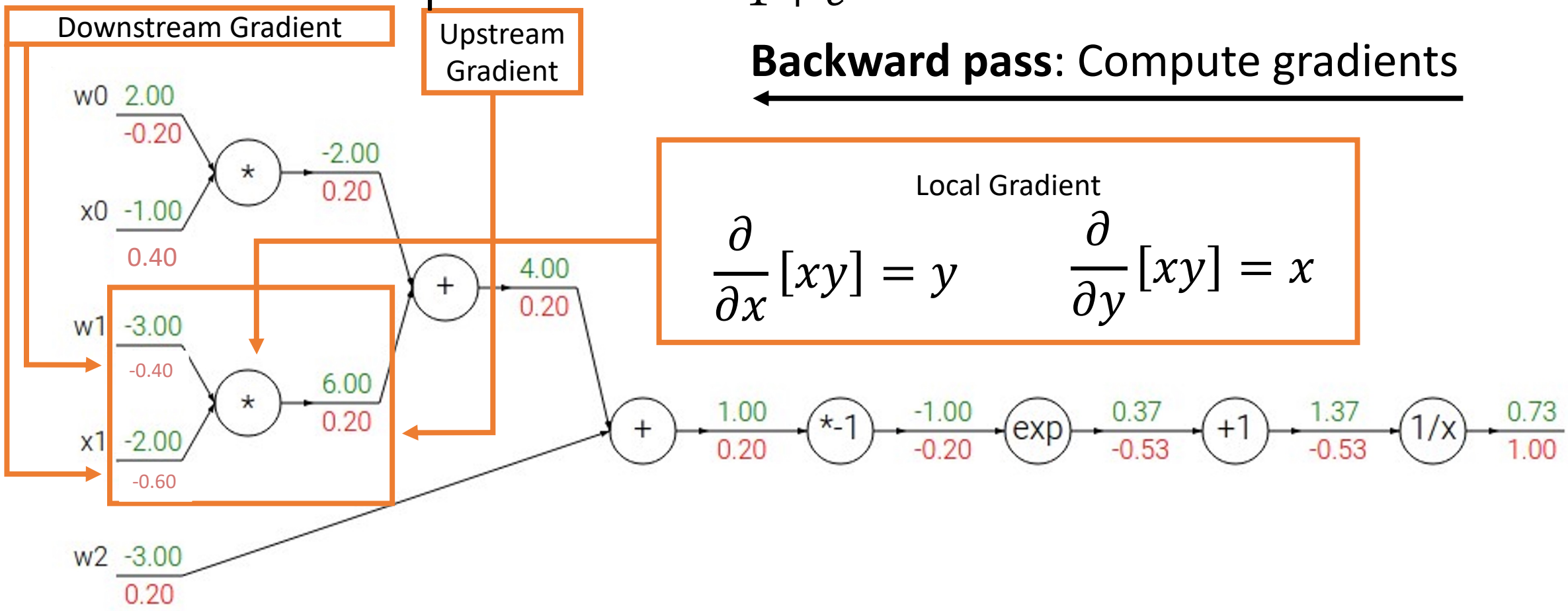
Backward pass: Compute gradients



Another Example

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

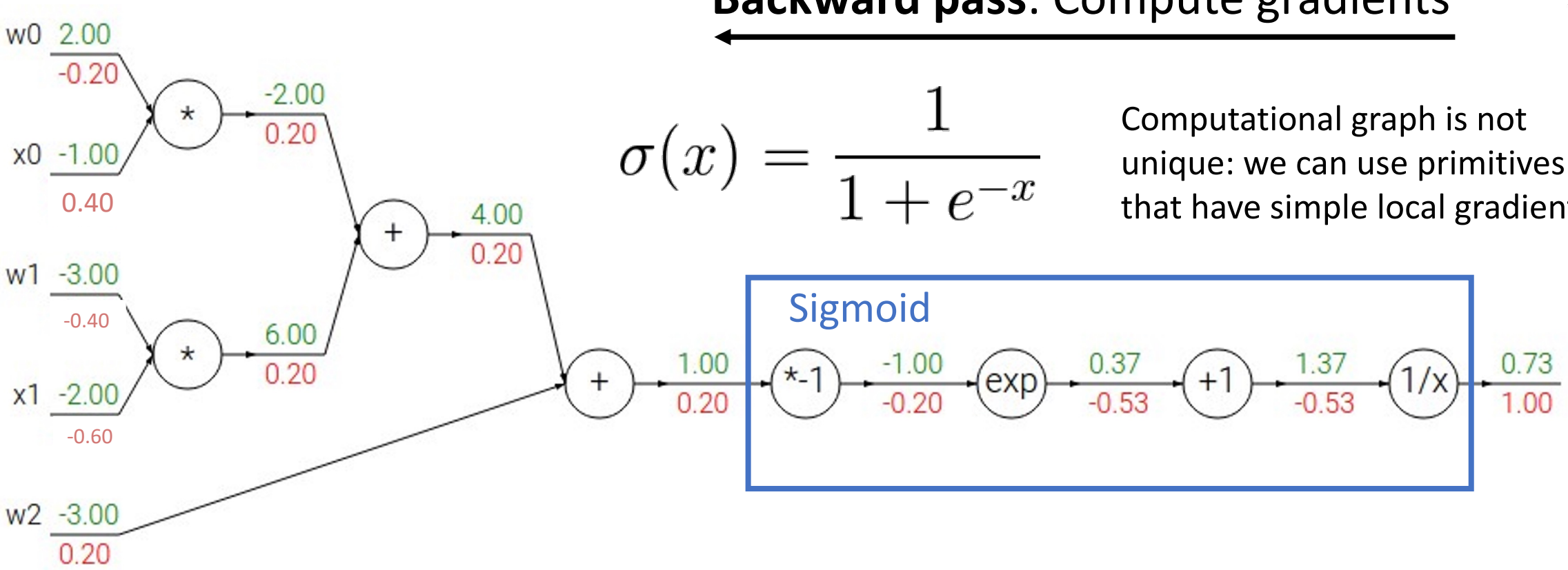
Backward pass: Compute gradients



Another Example

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

Backward pass: Compute gradients



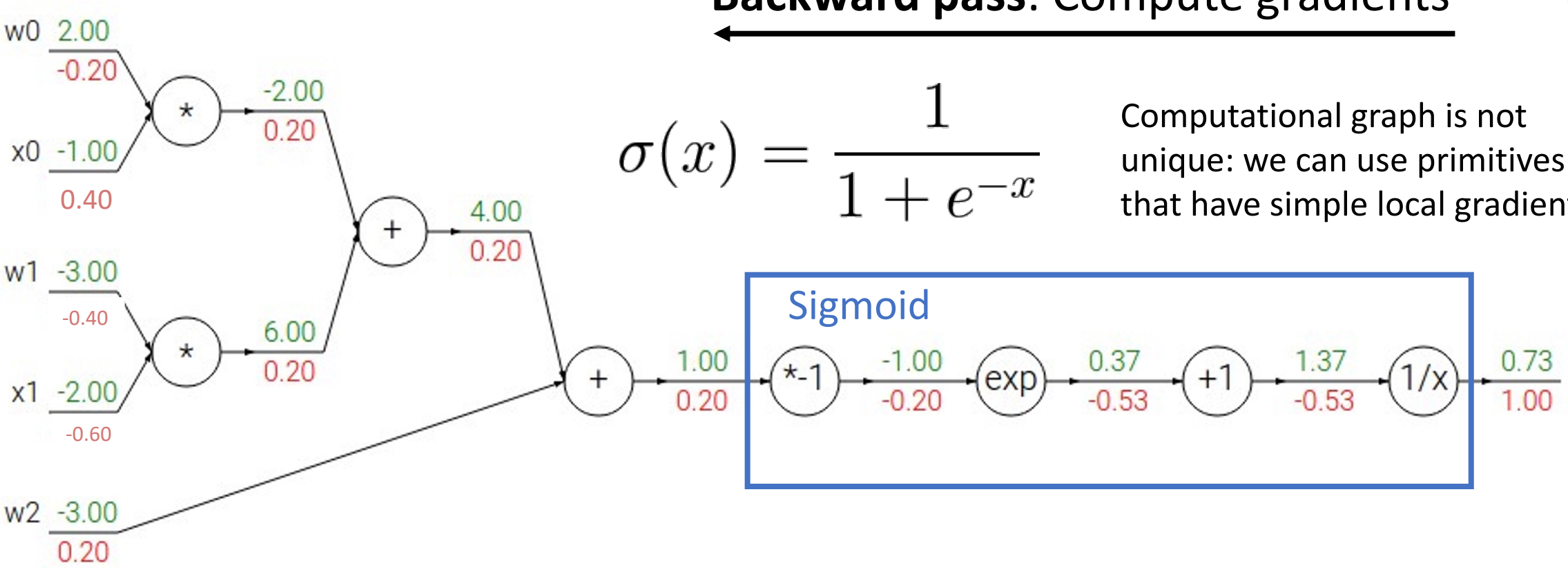
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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

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

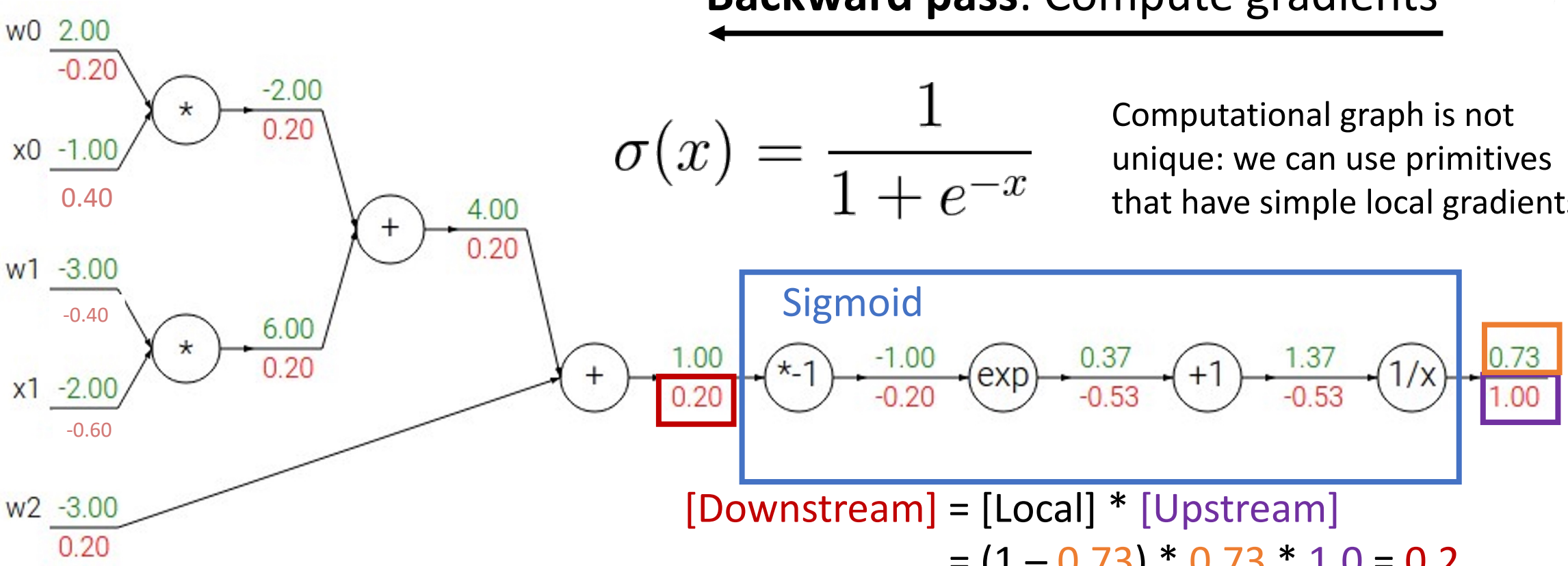
Sigmoid local gradient:

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

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

[Downstream] = [Local] * [Upstream]

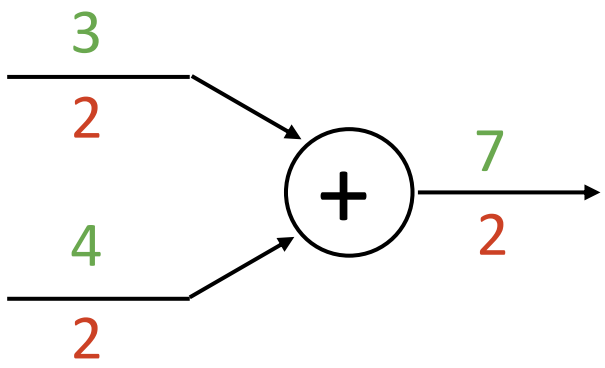
= (1 – 0.73) * 0.73 * 1.0 = 0.2

Sigmoid local gradient:

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

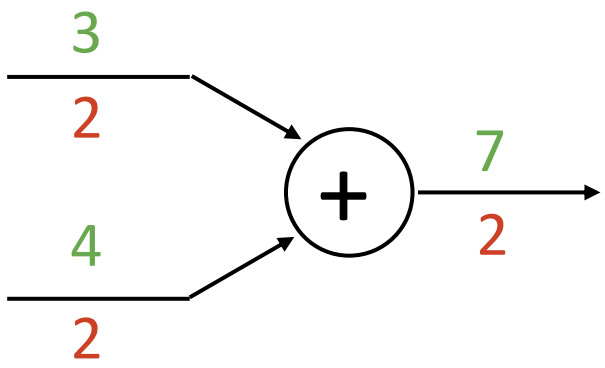
Patterns in Gradient Flow

add gate: gradient distributor

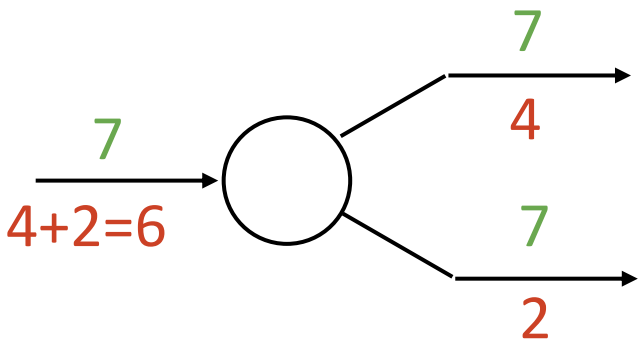


Patterns in Gradient Flow

add gate: gradient distributor

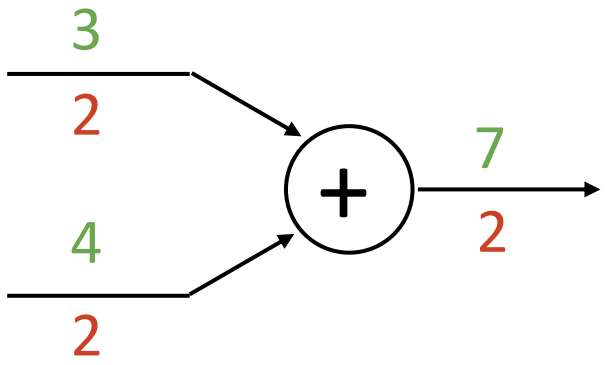


copy gate: gradient adder

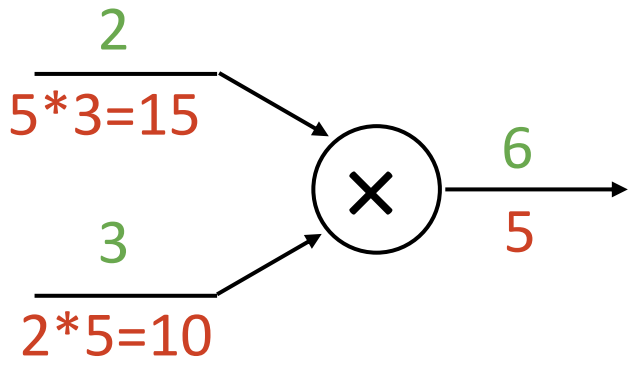


Patterns in Gradient Flow

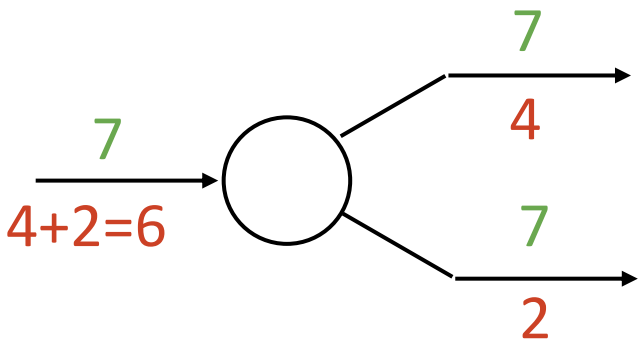
add gate: gradient distributor



mul gate: "swap multiplier"

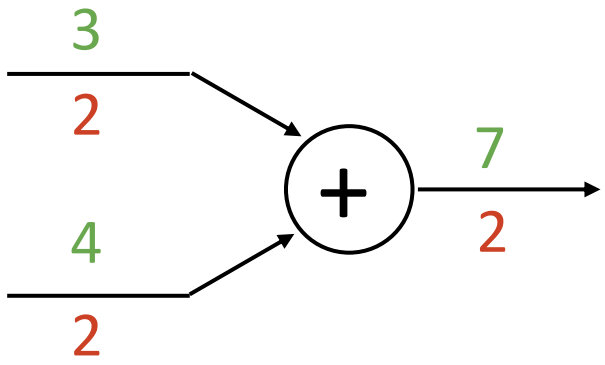


copy gate: gradient adder

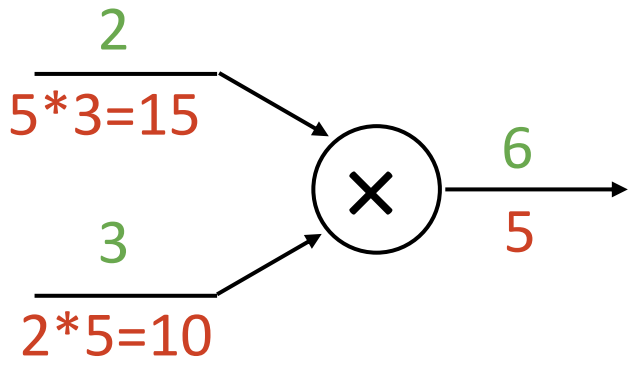


Patterns in Gradient Flow

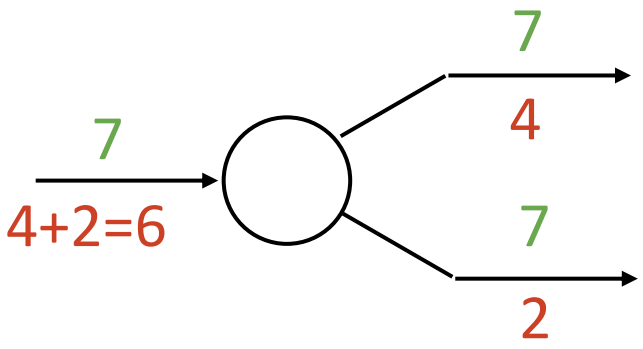
add gate: gradient distributor



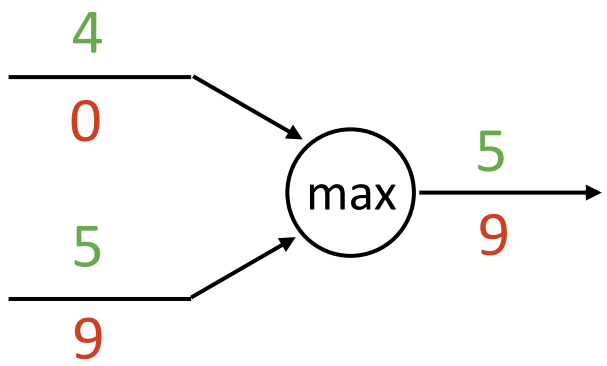
mul gate: "swap multiplier"



copy gate: gradient adder



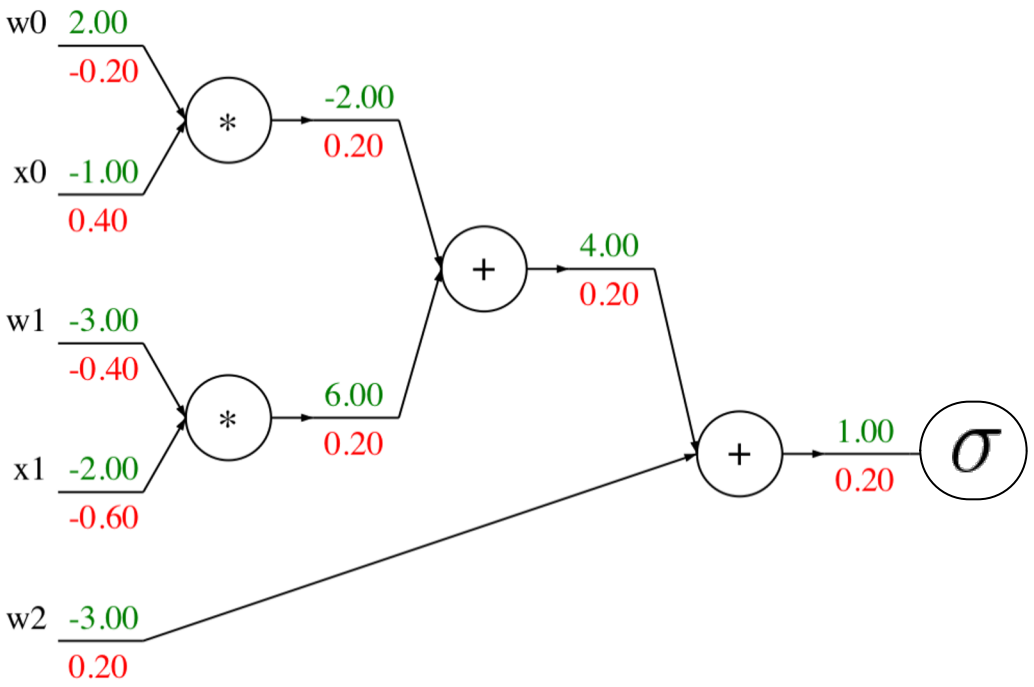
max gate: gradient router



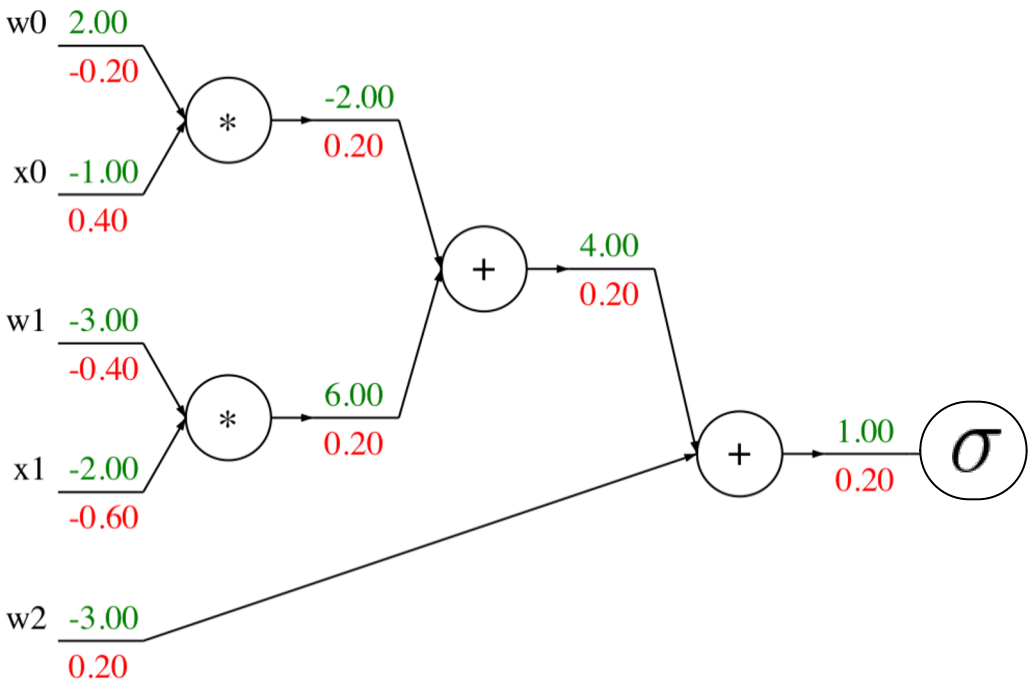
Backprop Implementation: "Flat" gradient code:

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

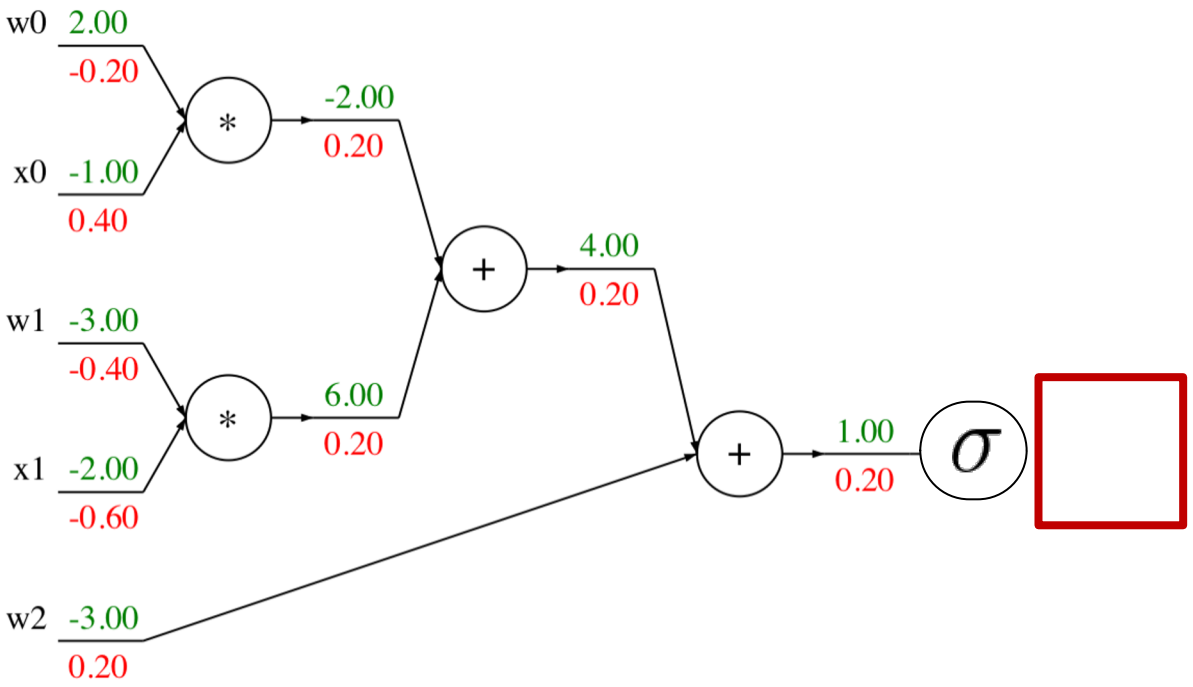
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Backward pass:
Compute grads

```
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



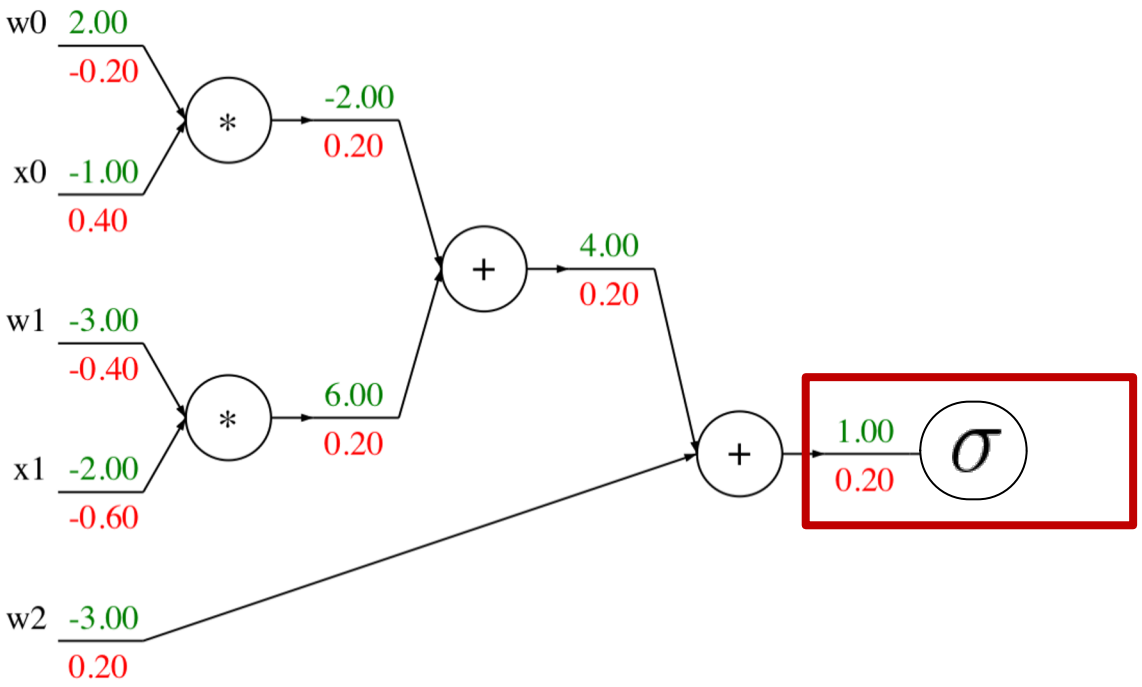

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Base case

```
  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

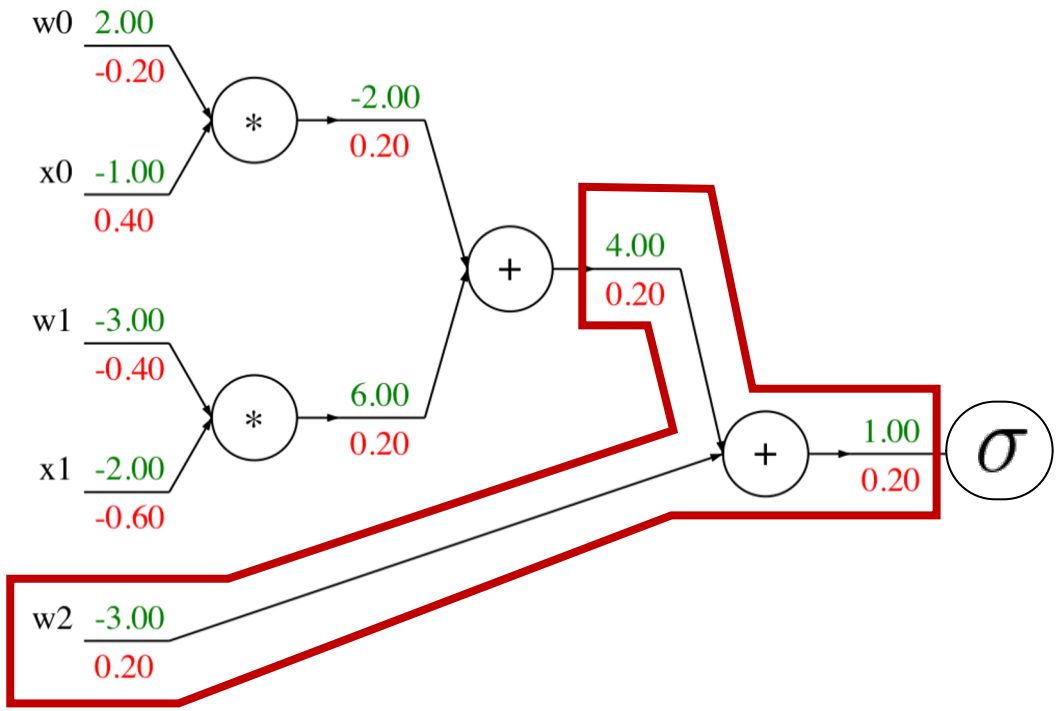



```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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
```

Backprop Implementation: "Flat" gradient code:



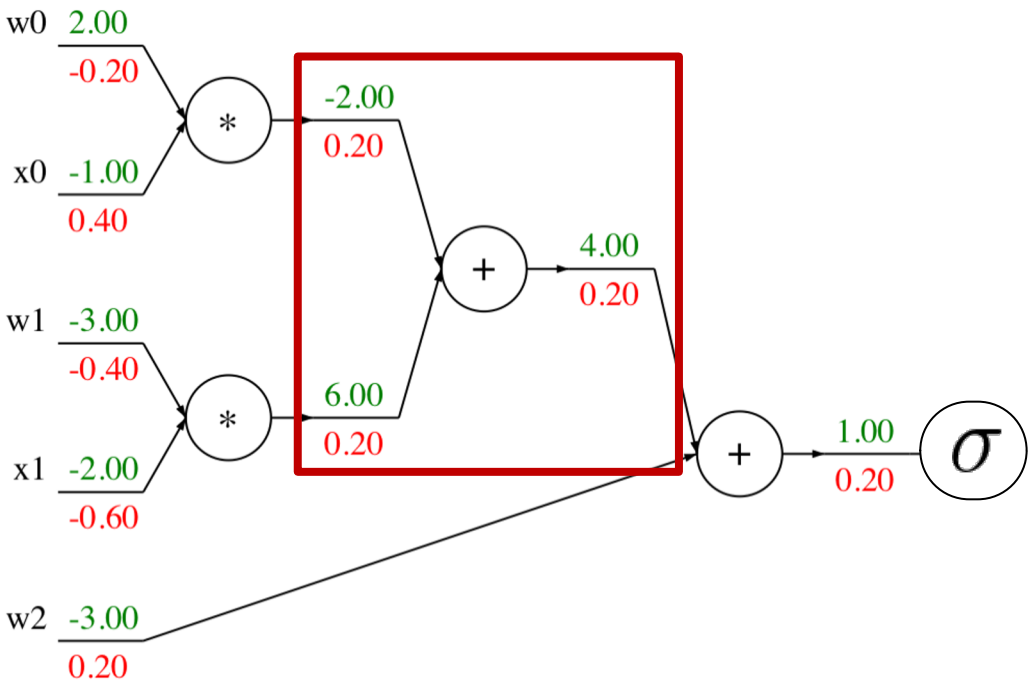
Forward pass:
Compute output

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

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
```

Backprop Implementation: "Flat" gradient code:



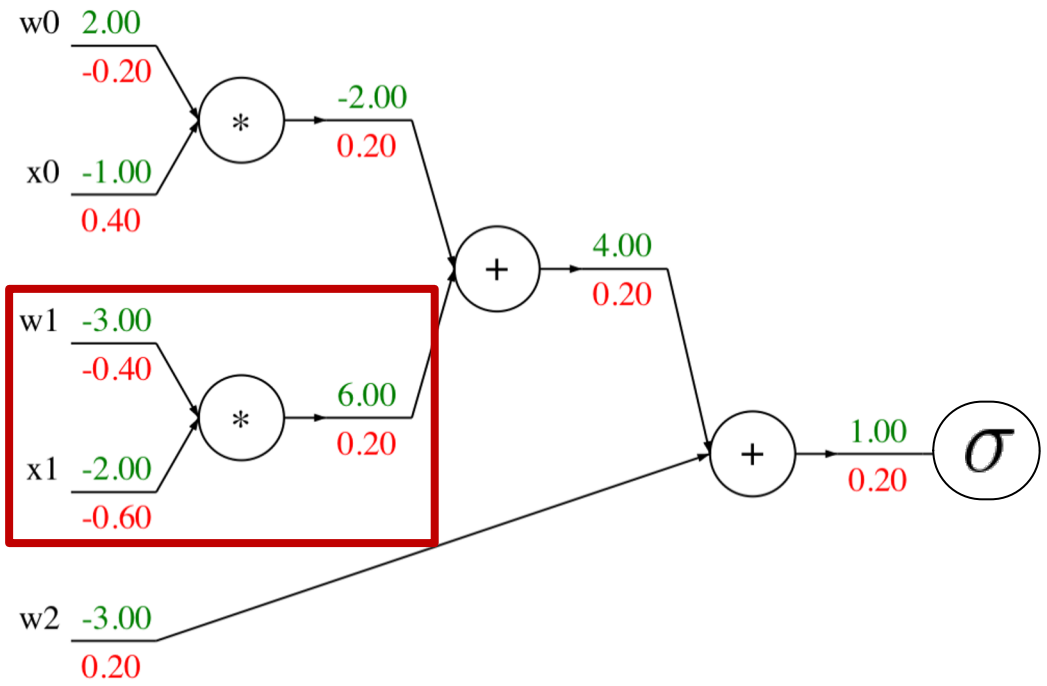
Forward pass:
Compute output

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

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
```

Backprop Implementation: "Flat" gradient code:



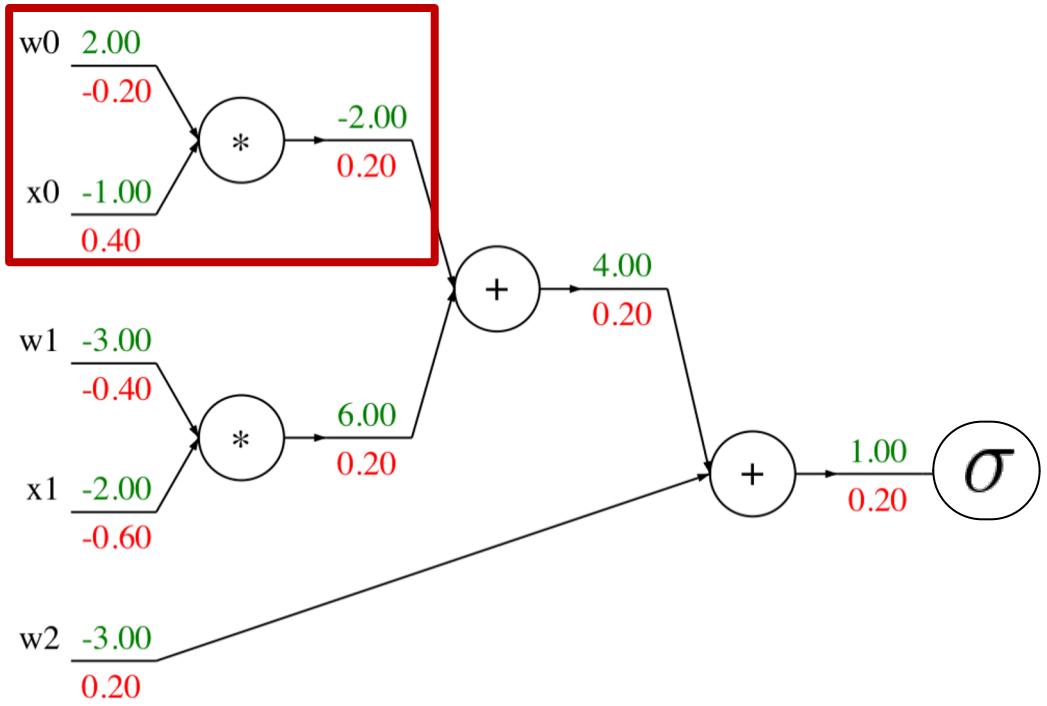
Forward pass: Compute output

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

Multiply

```
  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:



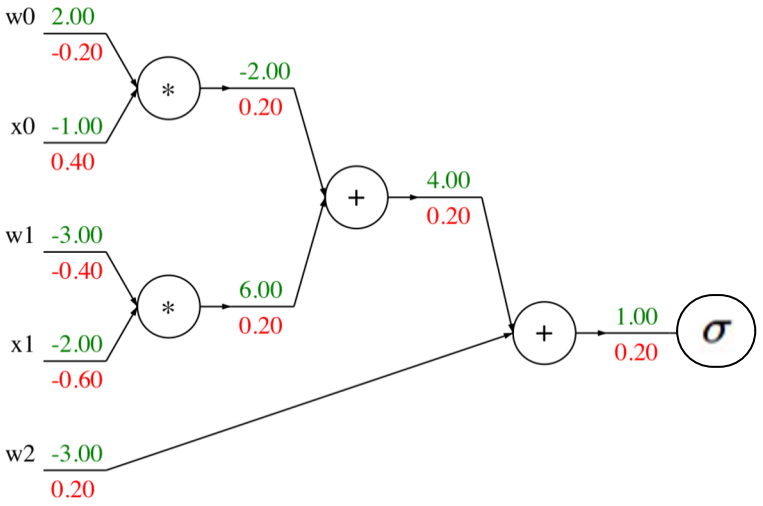
Forward pass: Compute output

```
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
```

Multiply

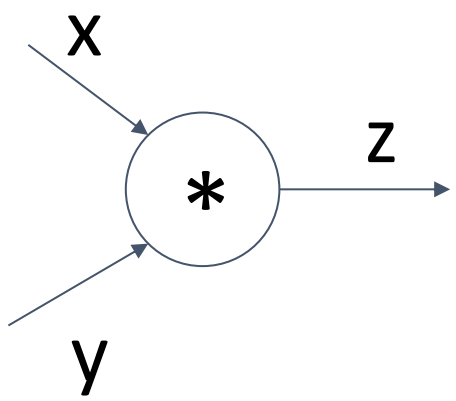
Backprop Implementation: Modular API



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)

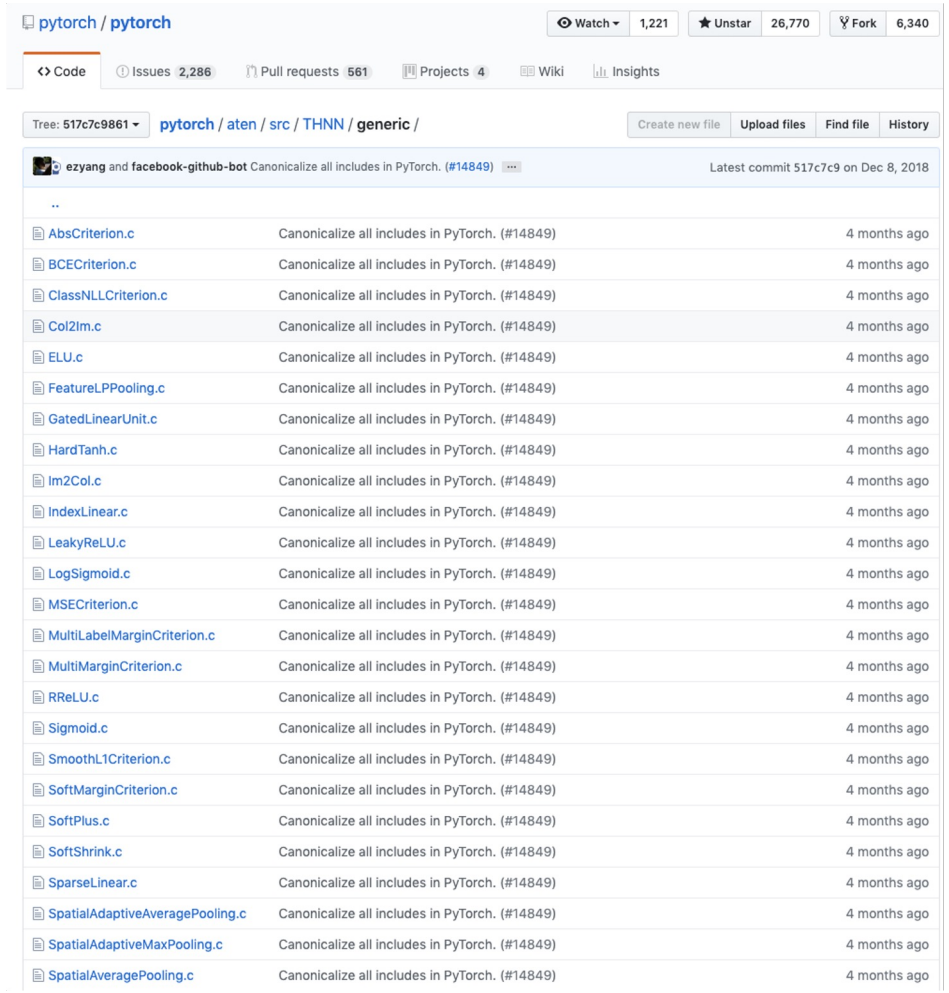
```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz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

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Example: PyTorch operators



pytorch / pytorch

Watch 1,221 | Unstar 26,770 | Fork 6,340

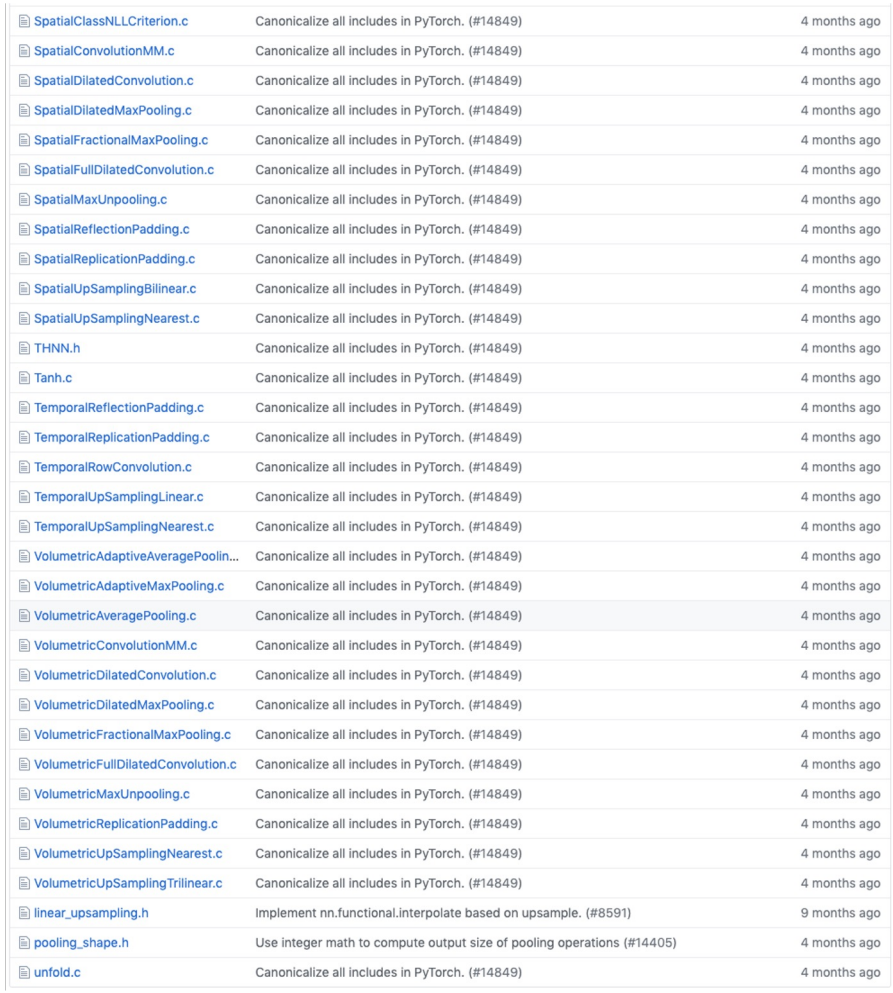
Code | Issues 2,286 | Pull requests 561 | Projects 4 | Wiki | Insights

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

..

AbsCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
BCECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ClassNLLCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Col2Im.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ELU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
FeatureLPPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
GatedLinearUnit.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
HardTanh.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Im2Col.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
IndexLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LeakyReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LogSigmoid.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MSECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiLabelMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
RReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Sigmoid.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SmoothL1Criterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftPlus.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftShrink.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SparseLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAdaptiveAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAdaptiveMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago



SpatialClassNLLCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialConvolutionMM.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialDilatedMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialFractionalMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialFullDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialMaxUnpooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialReflectionPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialUpSamplingBilinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
THNN.h	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Tanh.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalReflectionPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalRowConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalUpSamplingLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAdaptiveAveragePoolin...	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAdaptiveMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricConvolutionMM.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricDilatedMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricFractionalMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricFullDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricMaxUnpooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricUpSamplingTrilinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
linear_upsampling.h	Implement nn.functional.interpolate based on upsample. (#8591)	9 months ago
pooling_shape.h	Use integer math to compute output size of pooling operations (#14405)	4 months ago
unfold.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Source

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

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

Source

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#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;
        });
  });
}
```

Source

PyTorch sigmoid layer

```
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

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

Backward

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

Source

Life is easy in PyTorch (no need to write backpropagation because of Autograd)

```
class NestMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(nn.Linear(20, 64), nn.ReLU(),
                                 nn.Linear(64, 32), nn.ReLU())
        self.linear = nn.Linear(32, 16)

    def forward(self, X):
        return self.linear(self.net(X))

chimera = nn.Sequential(NestMLP(), nn.Linear(16, 20), FixedHiddenMLP())
chimera(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**:

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

$$\left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

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**:

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

$$\left(\frac{\partial y}{\partial x}\right)_i = \frac{\partial y}{\partial x_i}$$

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

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

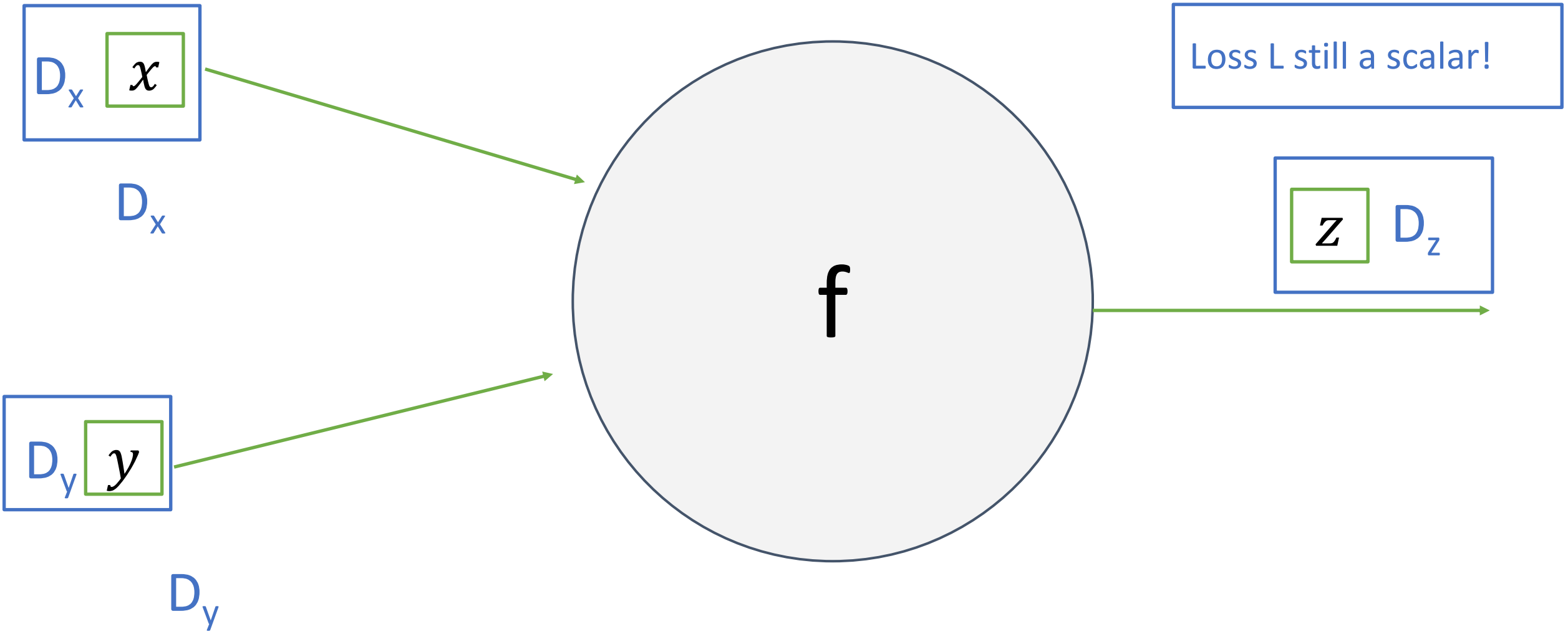
Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{M \times N}$$

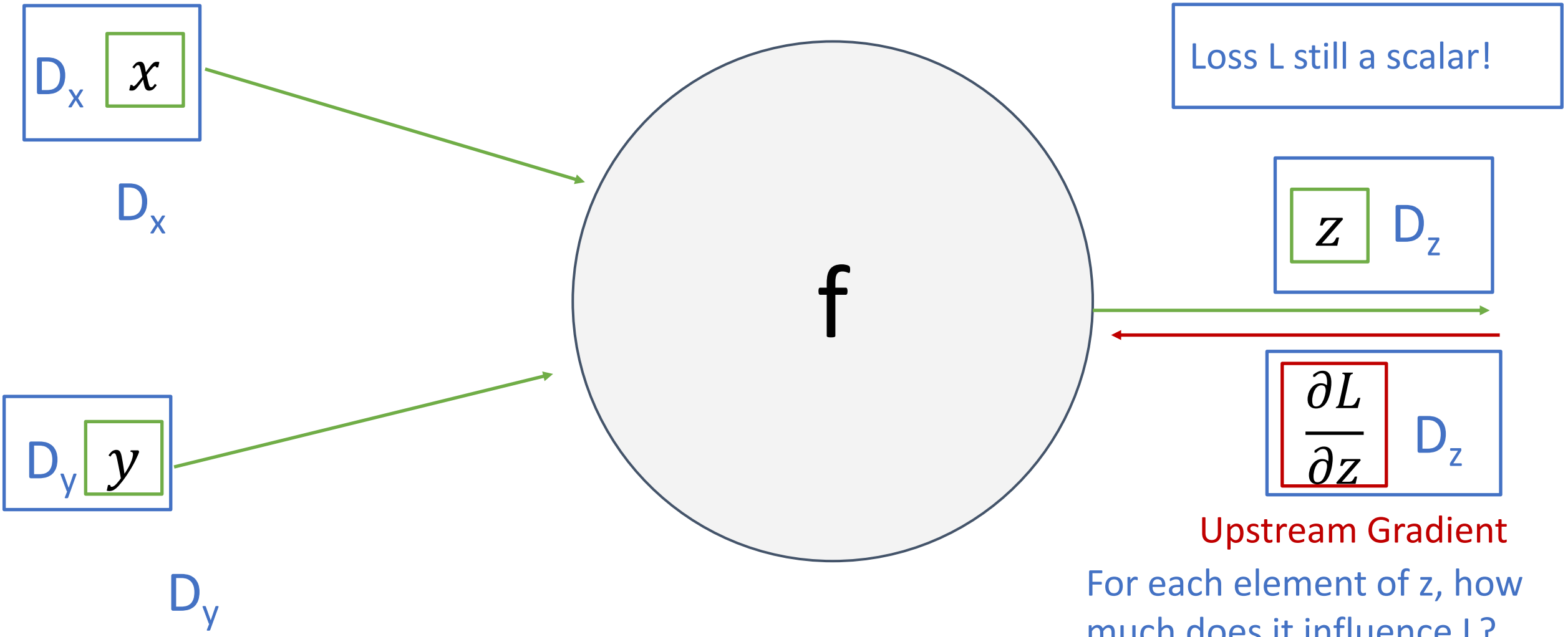
$$\left(\frac{\partial y}{\partial x}\right)_{j,i} = \frac{\partial y_j}{\partial x_i}$$

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

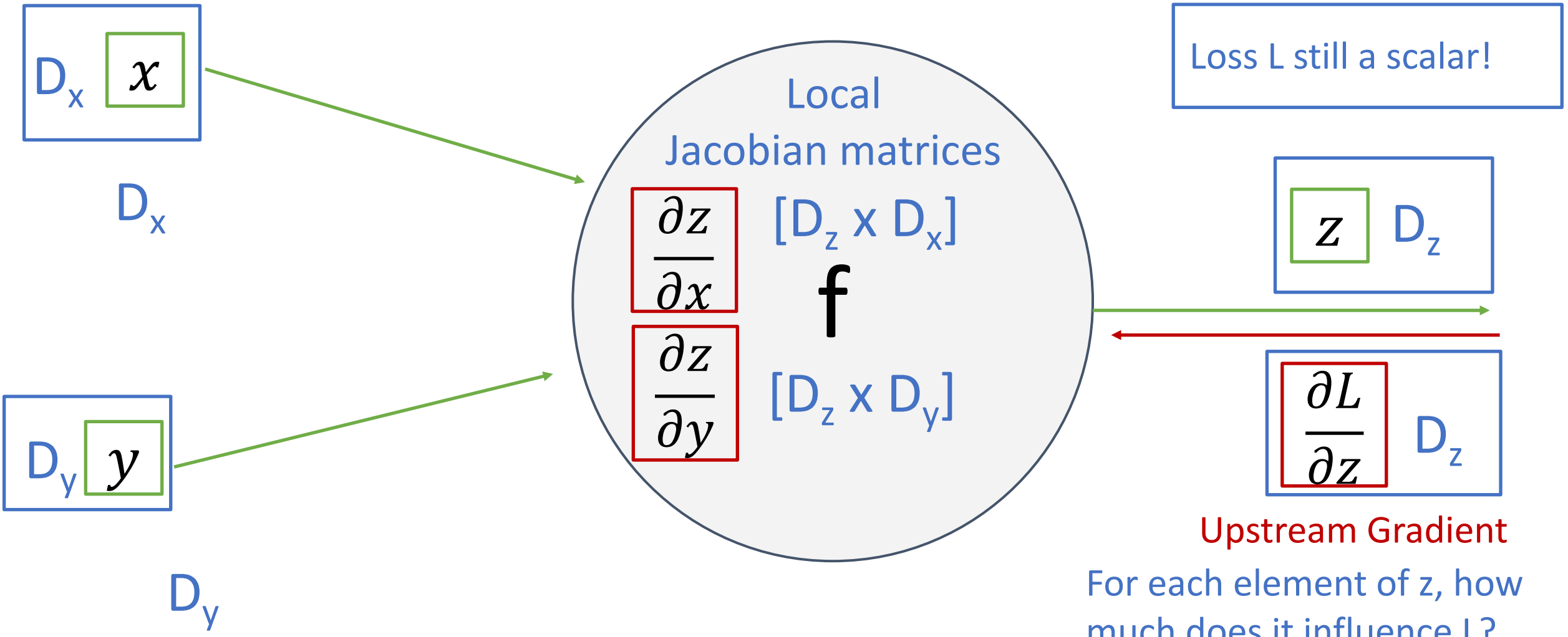
Backprop with Vectors



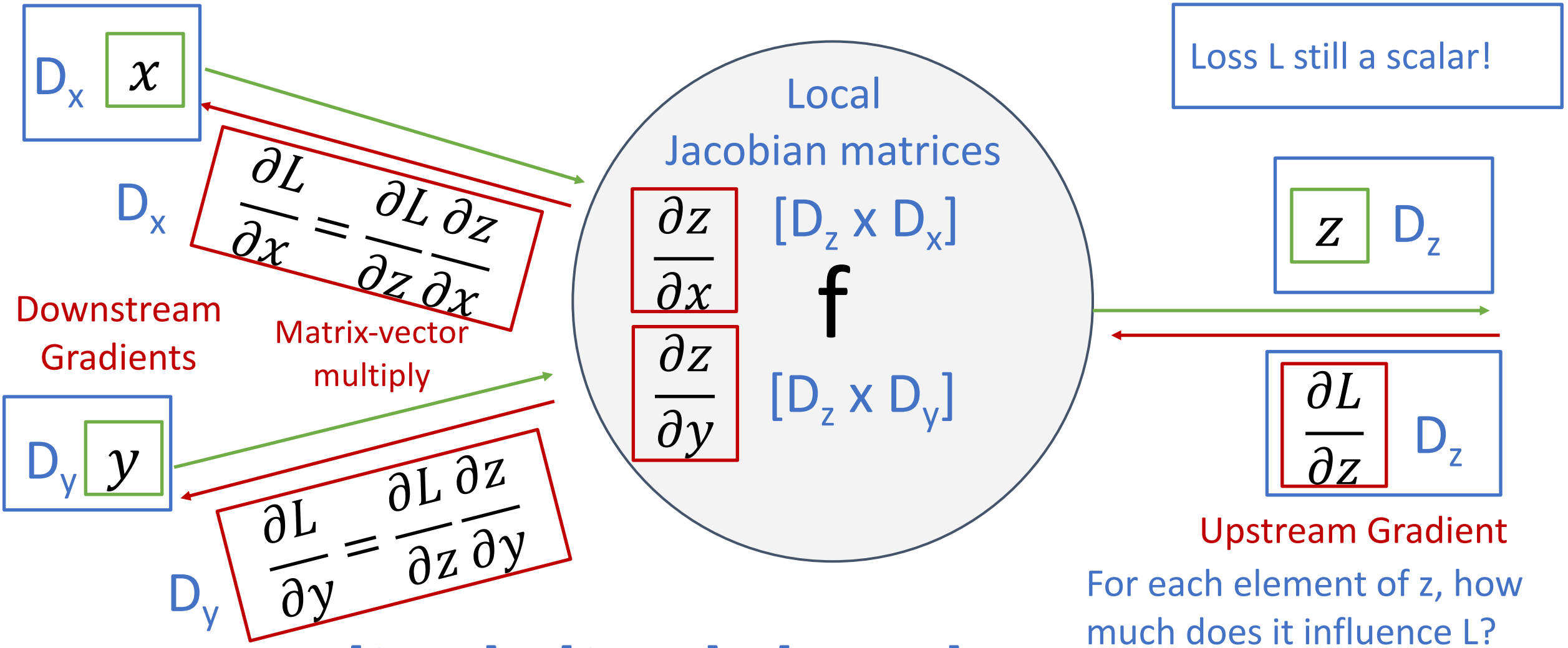
Backprop with Vectors



Backprop with Vectors

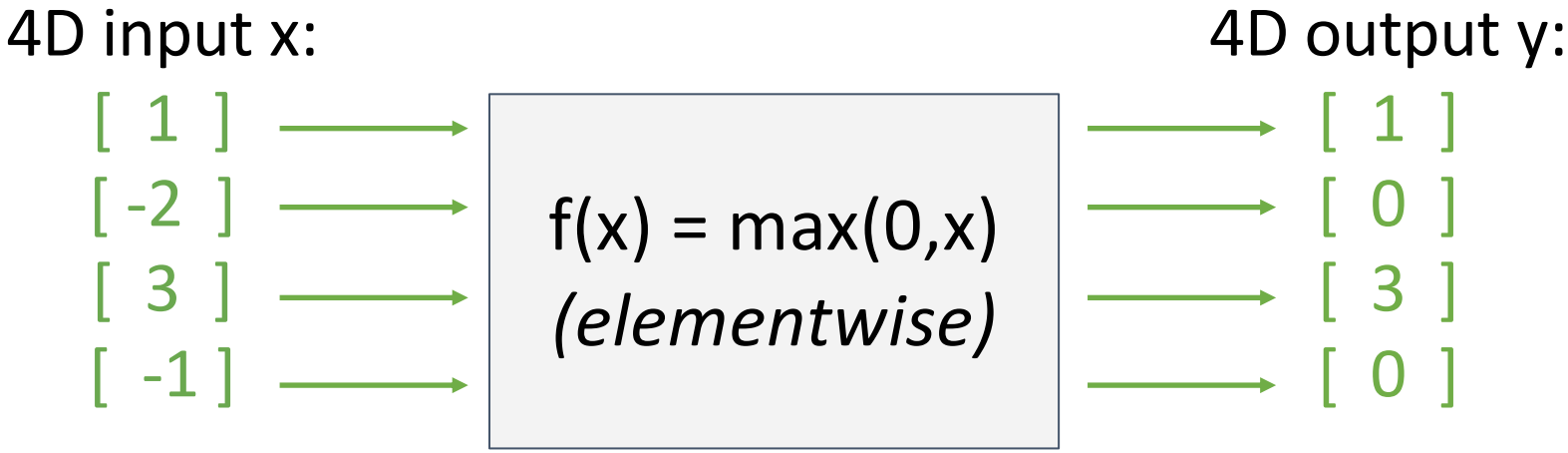


Backprop with Vectors

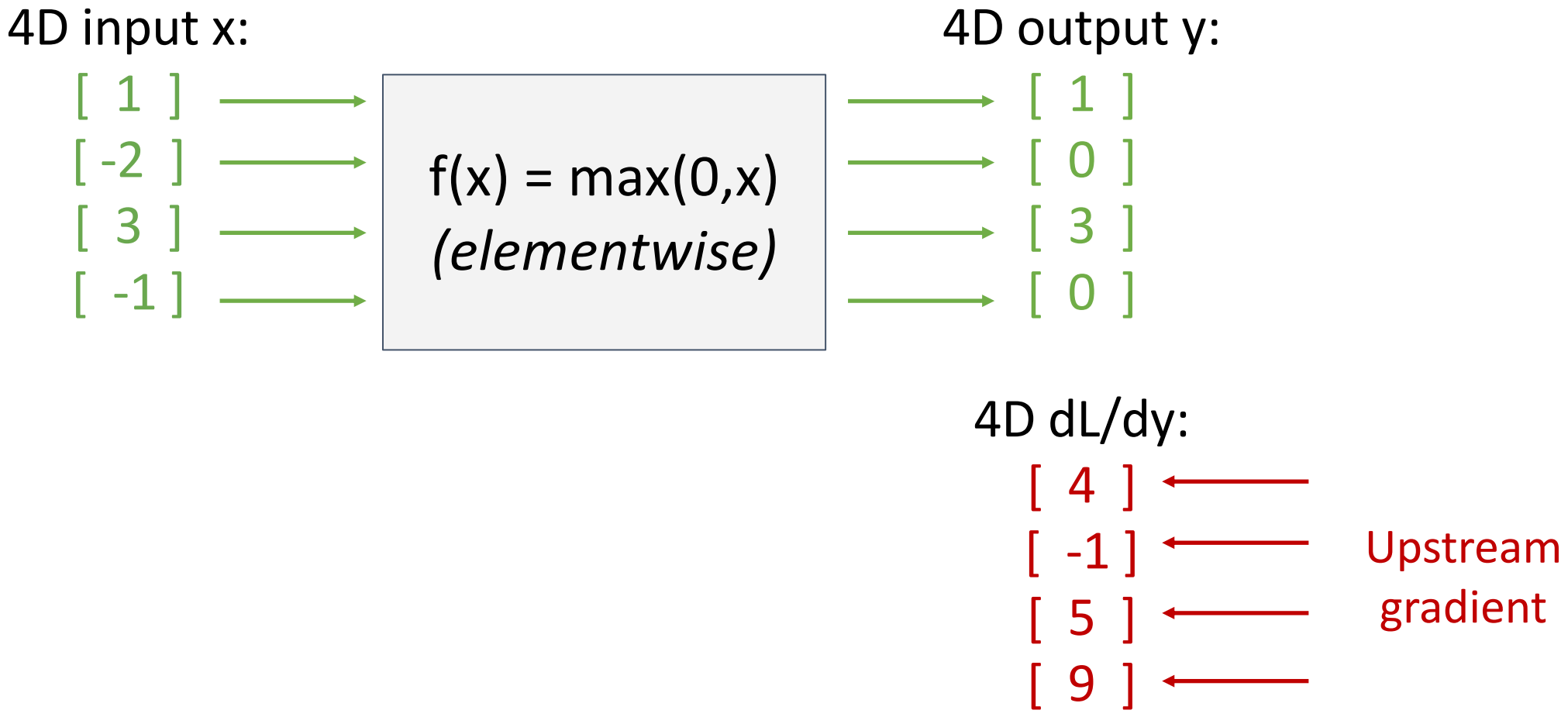


$$[1 \times D_y] = [1 \times D_z] \times [D_z \times D_y]$$

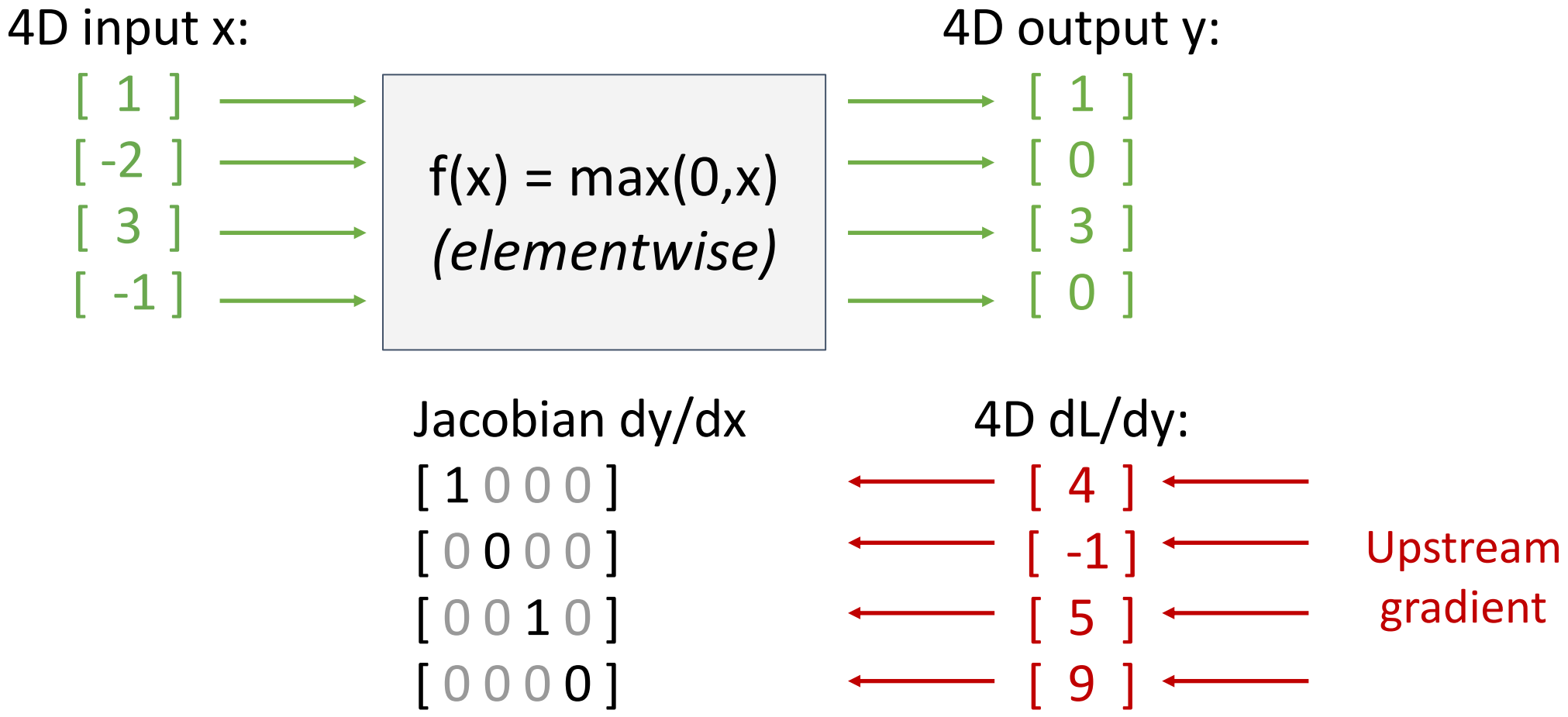
Backprop with Vectors



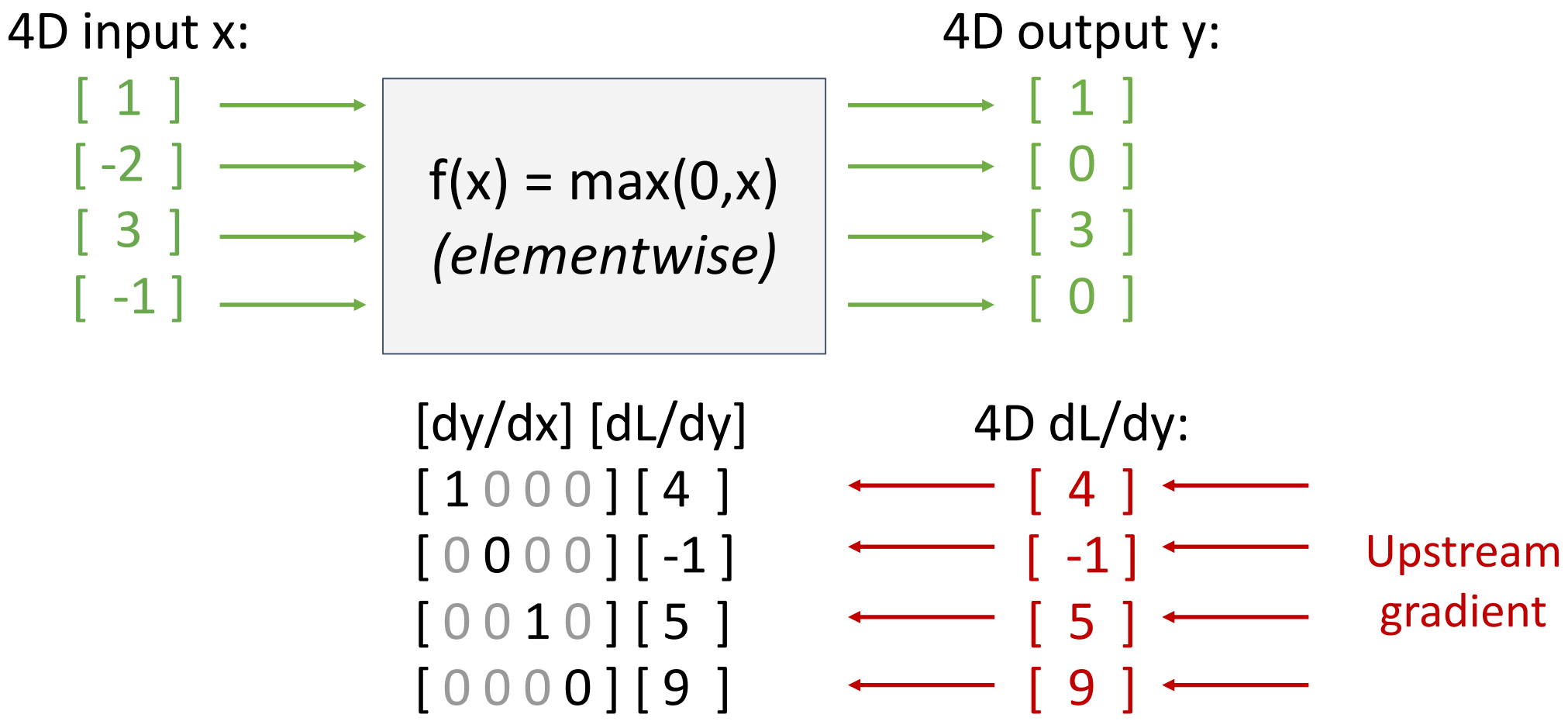
Backprop with Vectors



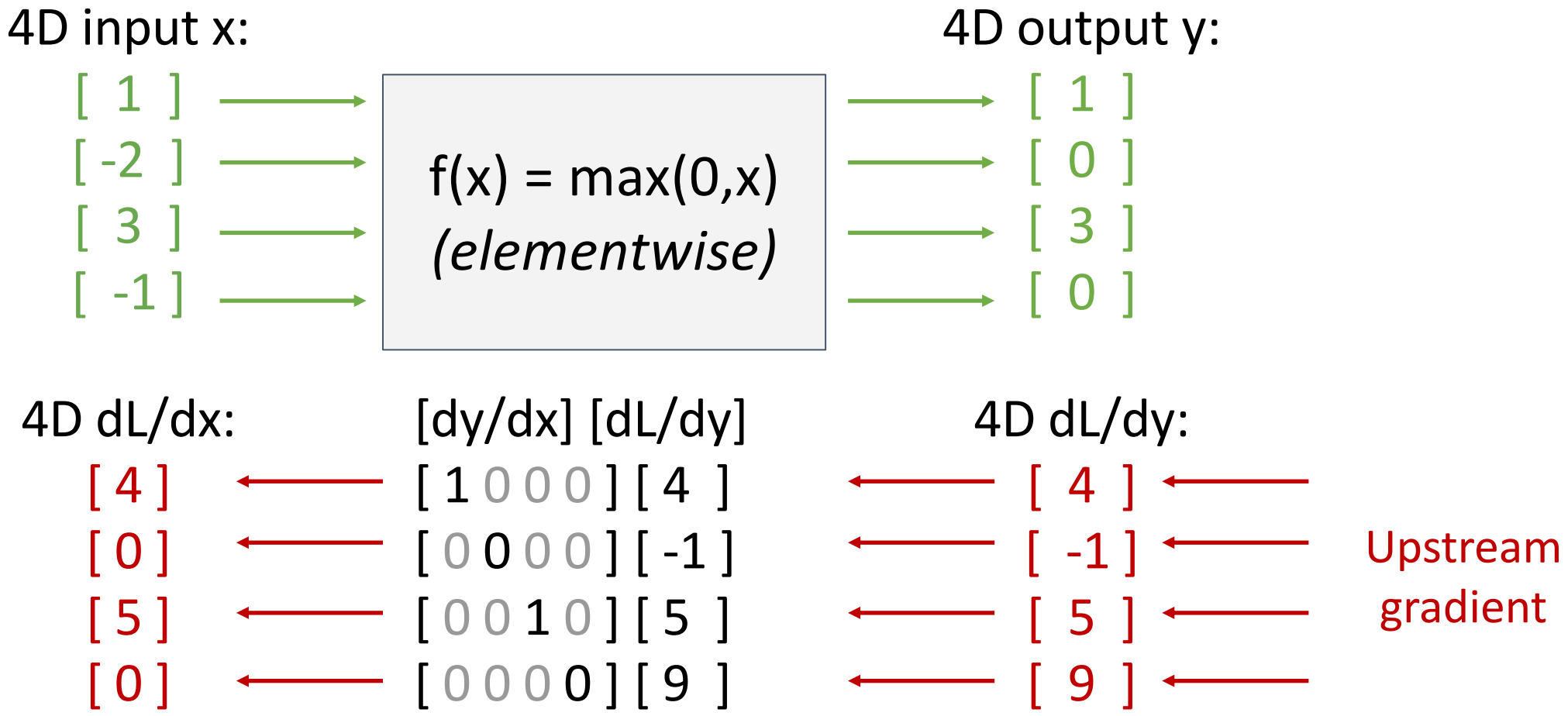
Backprop with Vectors



Backprop with Vectors

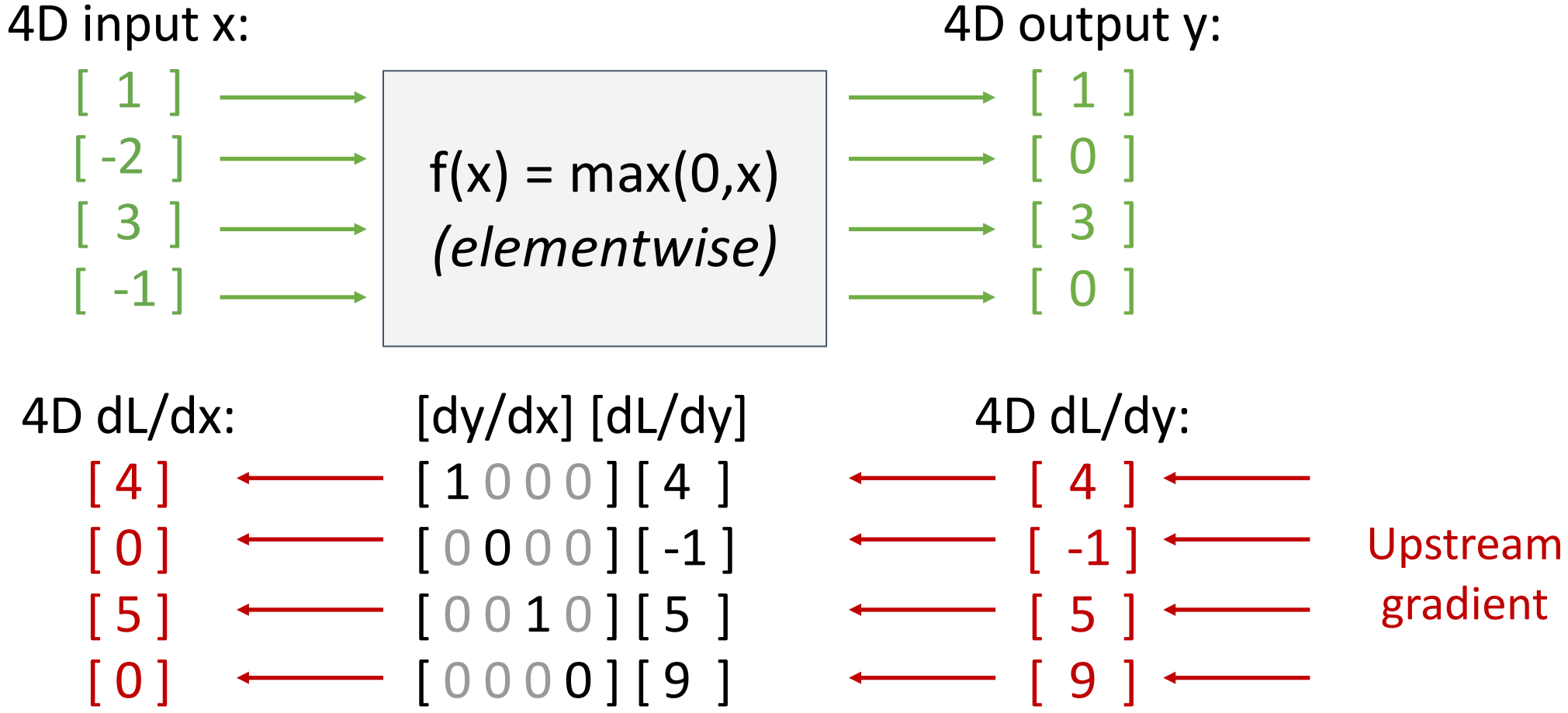


Backprop with Vectors



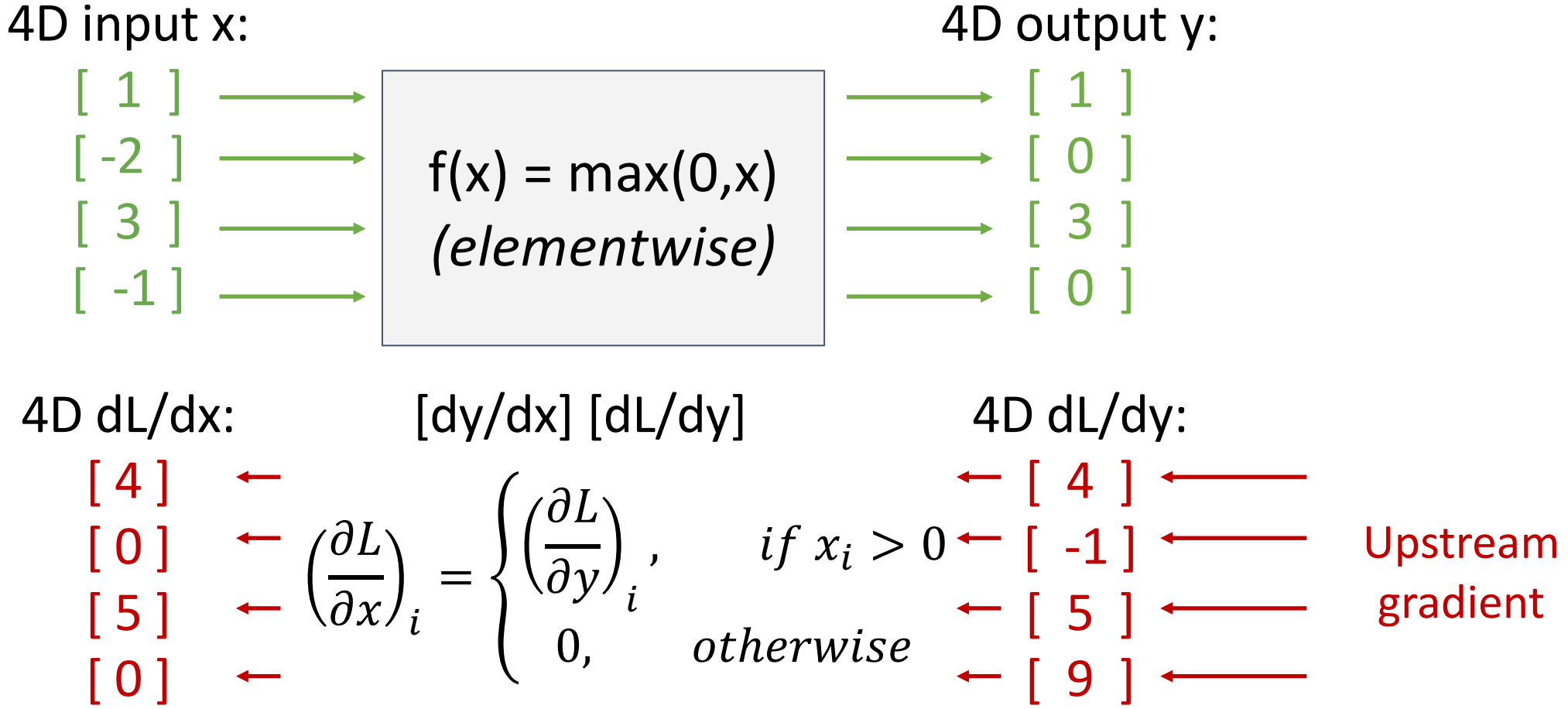
Backprop with Vectors

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

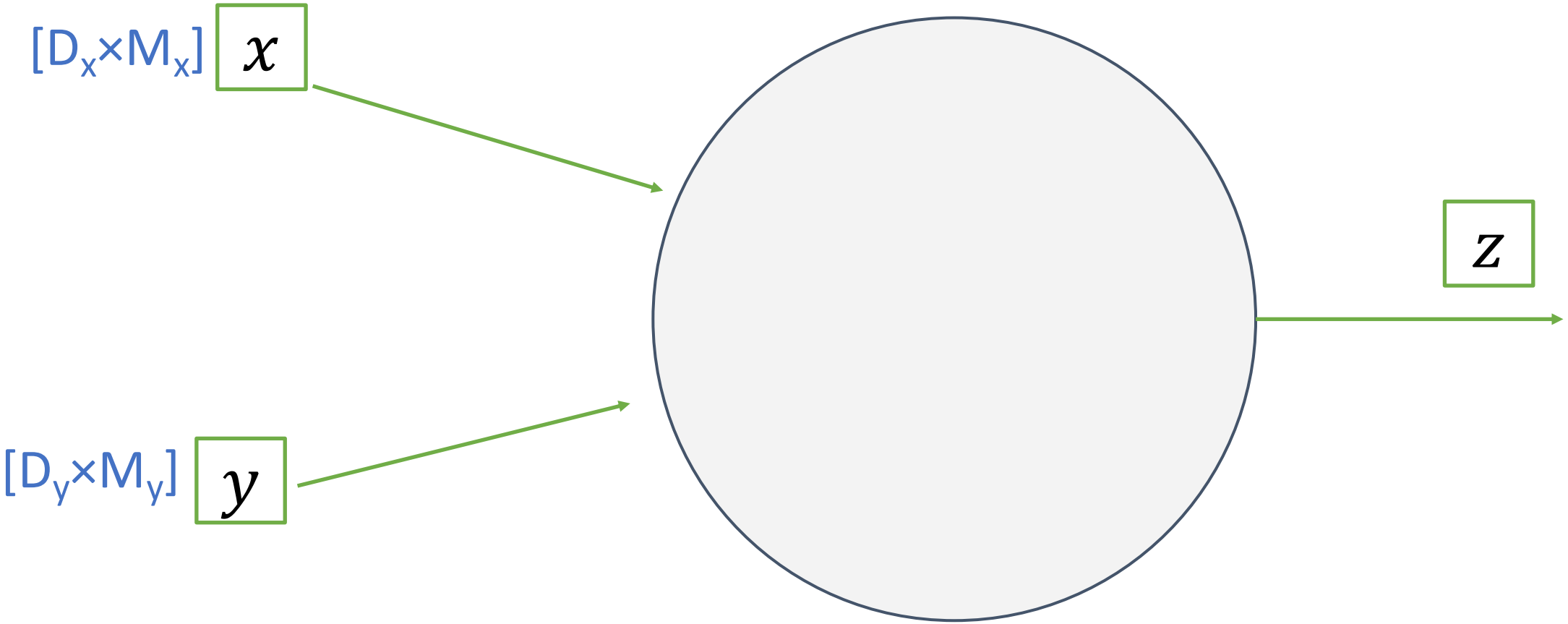


Backprop with Vectors

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



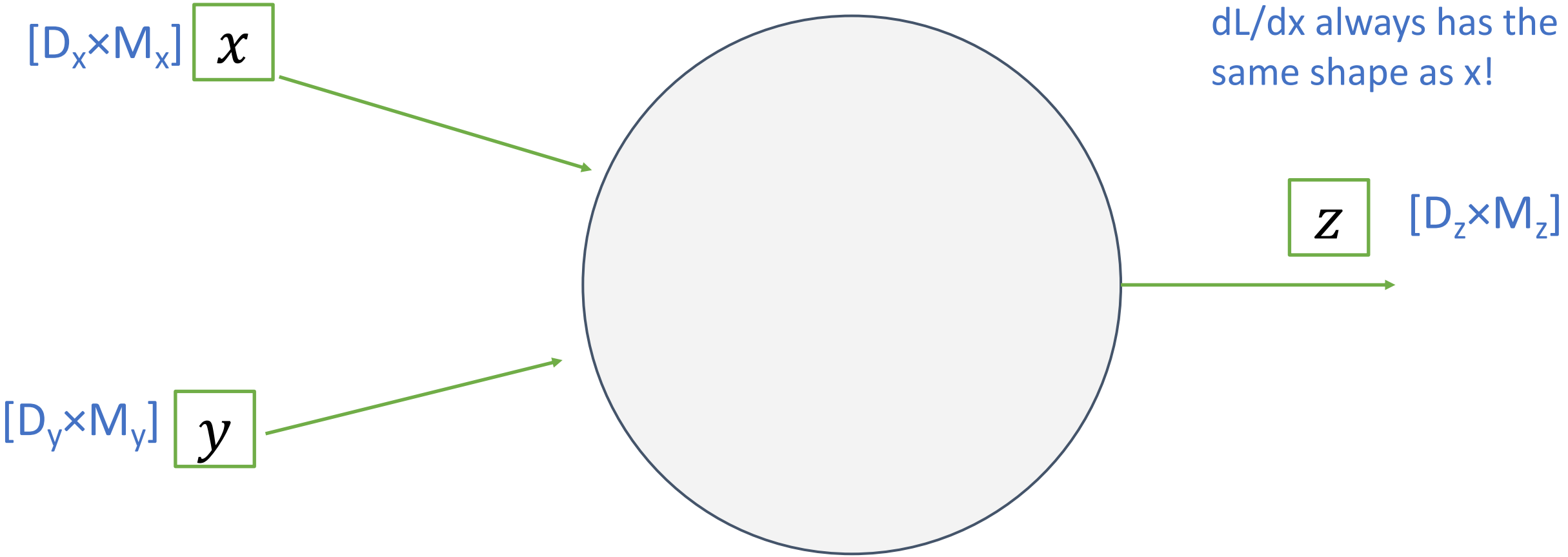
Backprop with Matrices (or Tensors):



Backprop with Matrices (or Tensors):

Loss L still a scalar!

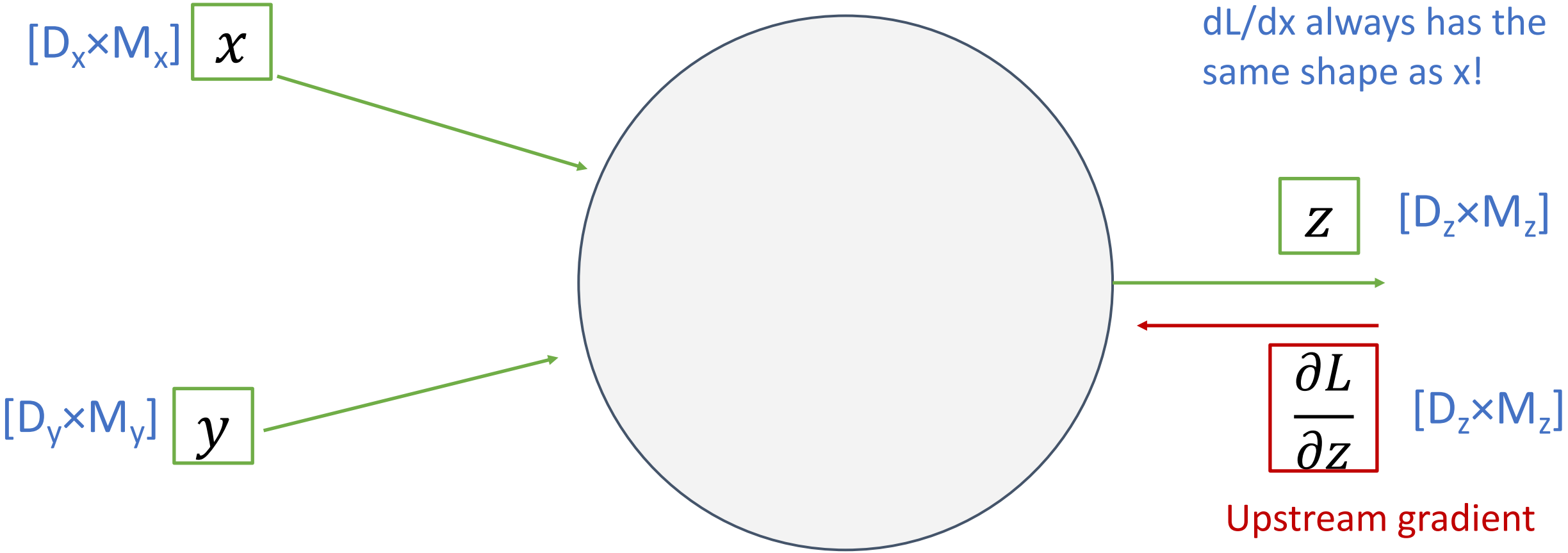
dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors):

Loss L still a scalar!

dL/dx always has the same shape as x!



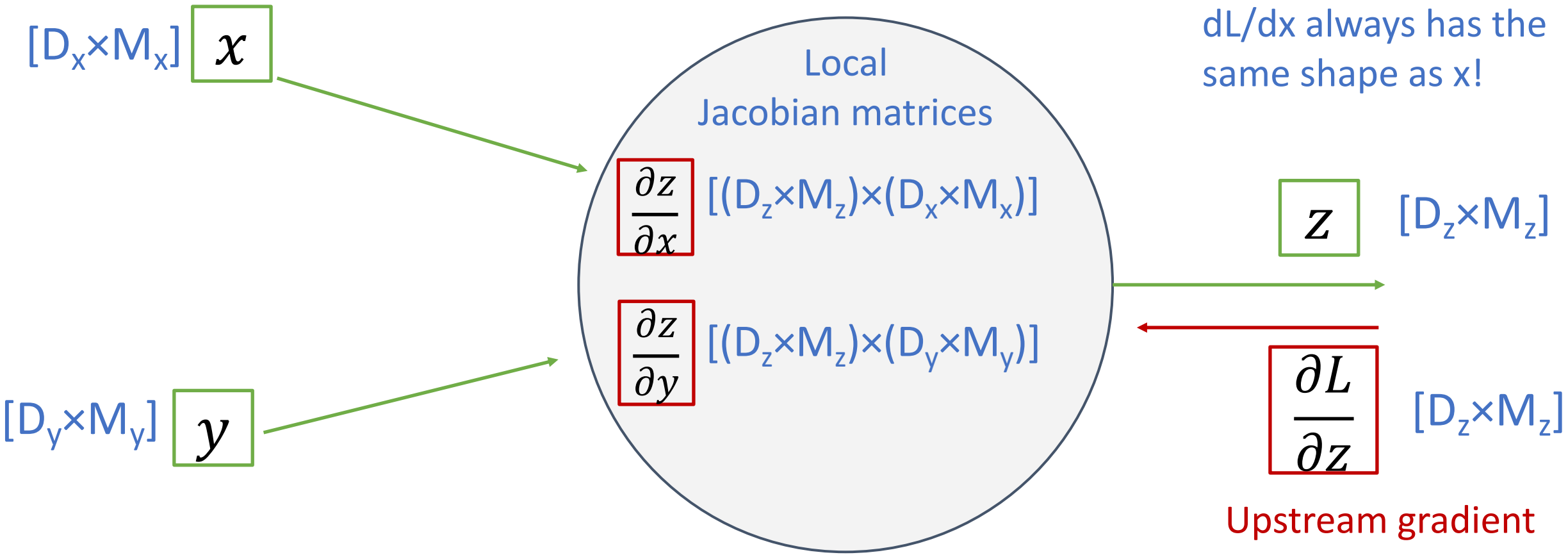
Upstream gradient

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!



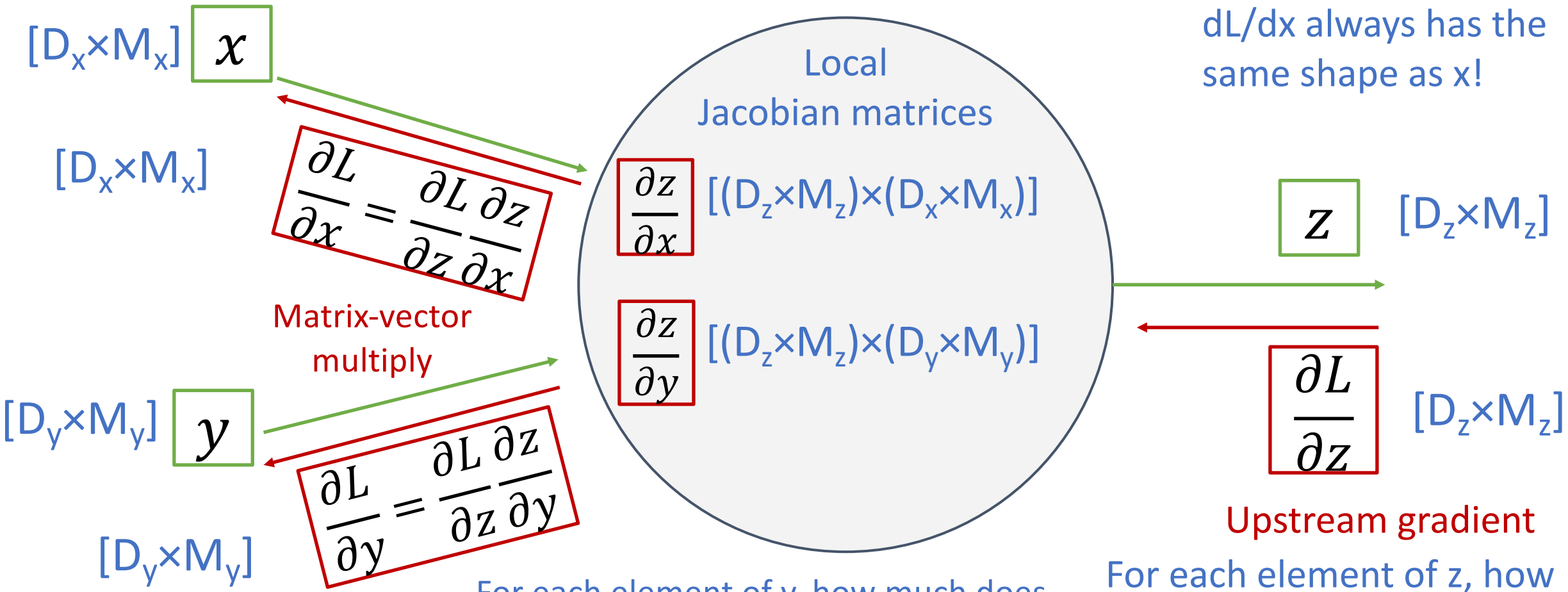
For each element of y, how much does it influence each element of z?

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 y, how much does it influence each element of z?

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

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]

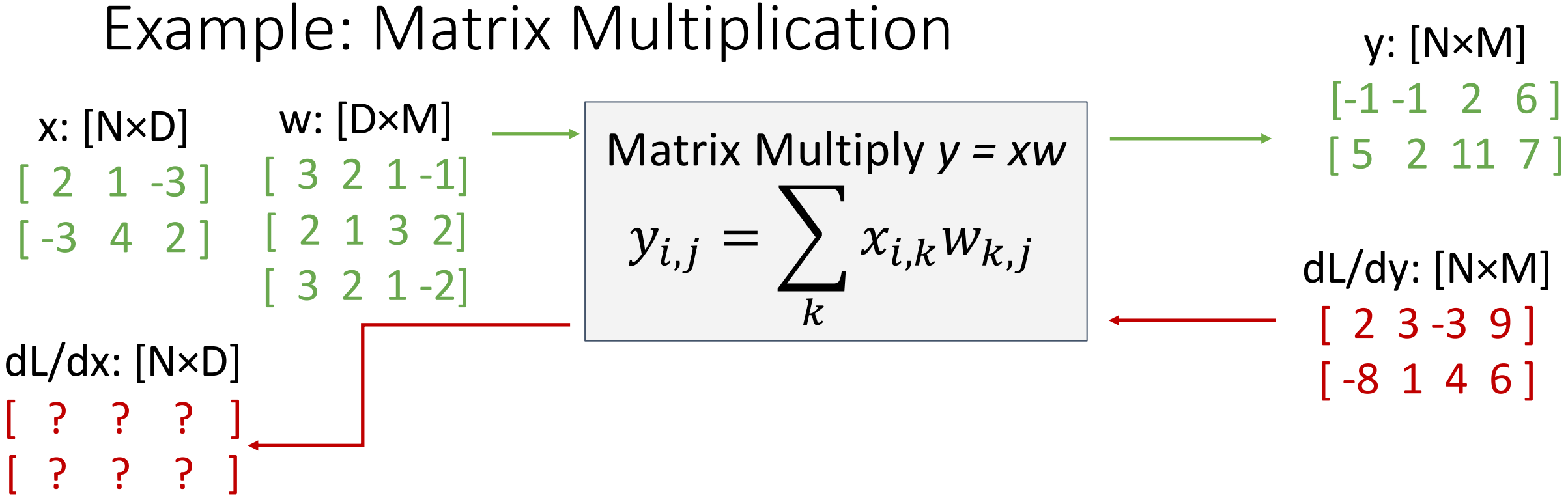
Matrix Multiply $y = xw$

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

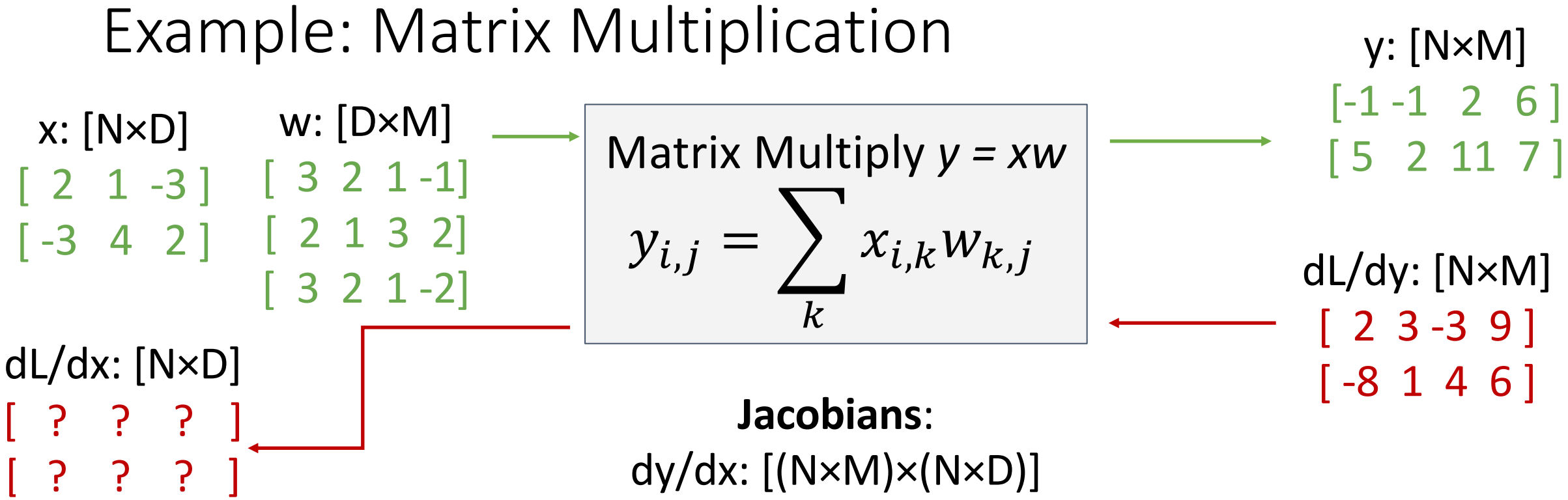
y: [N×M]

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

Example: Matrix Multiplication



Example: Matrix Multiplication



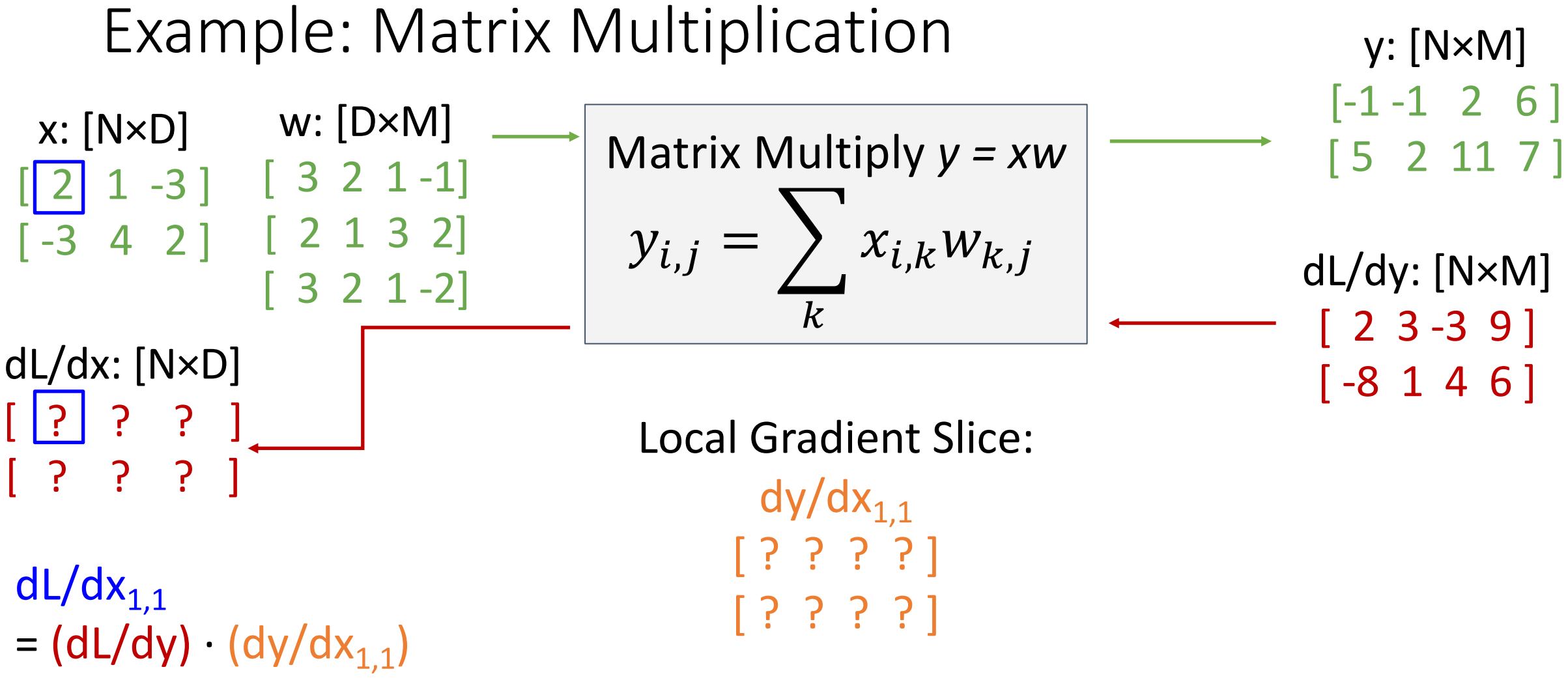
Jacobians:

dy/dx: [(N×M)×(N×D)]

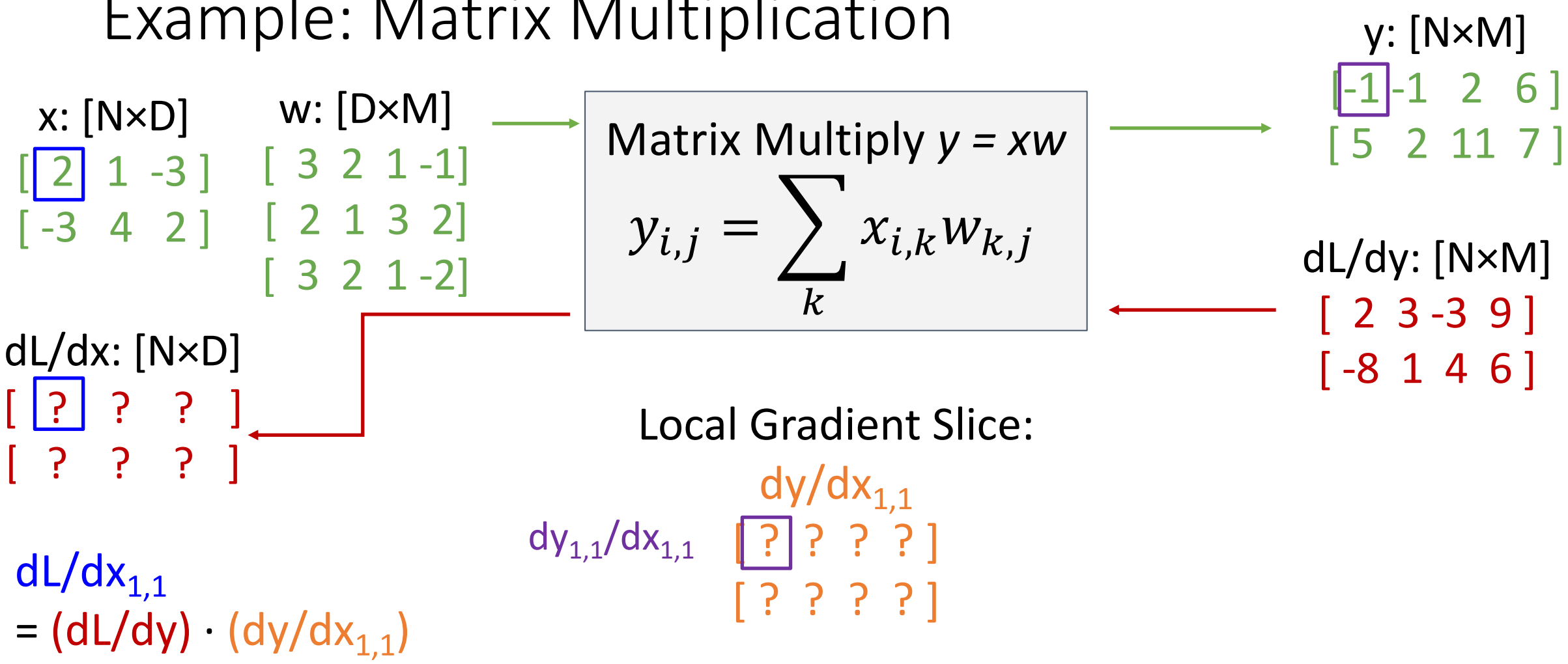
dy/dw: [(N×M)×(D×M)]

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

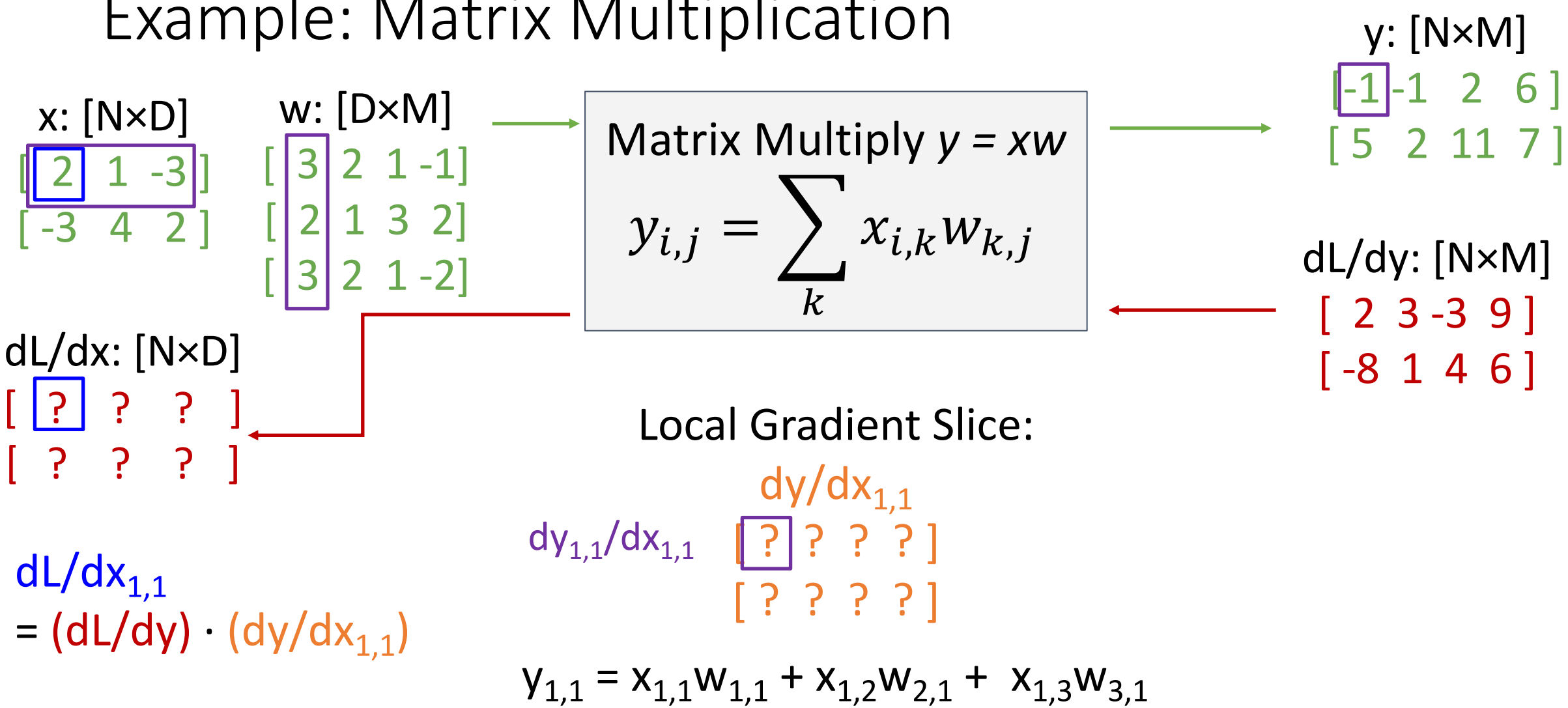
Example: Matrix Multiplication



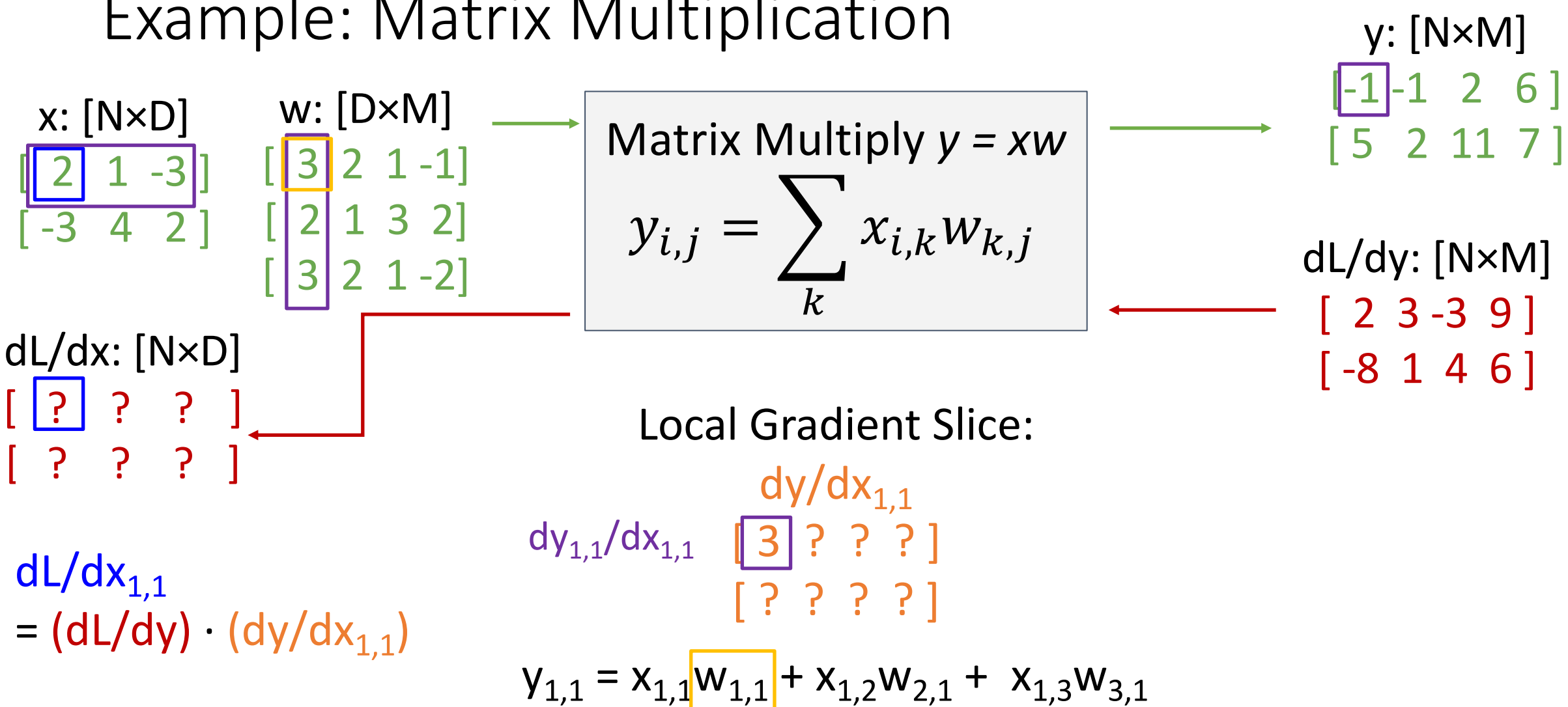
Example: Matrix Multiplication



Example: Matrix Multiplication



Example: Matrix Multiplication



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

Local Gradient Slice:

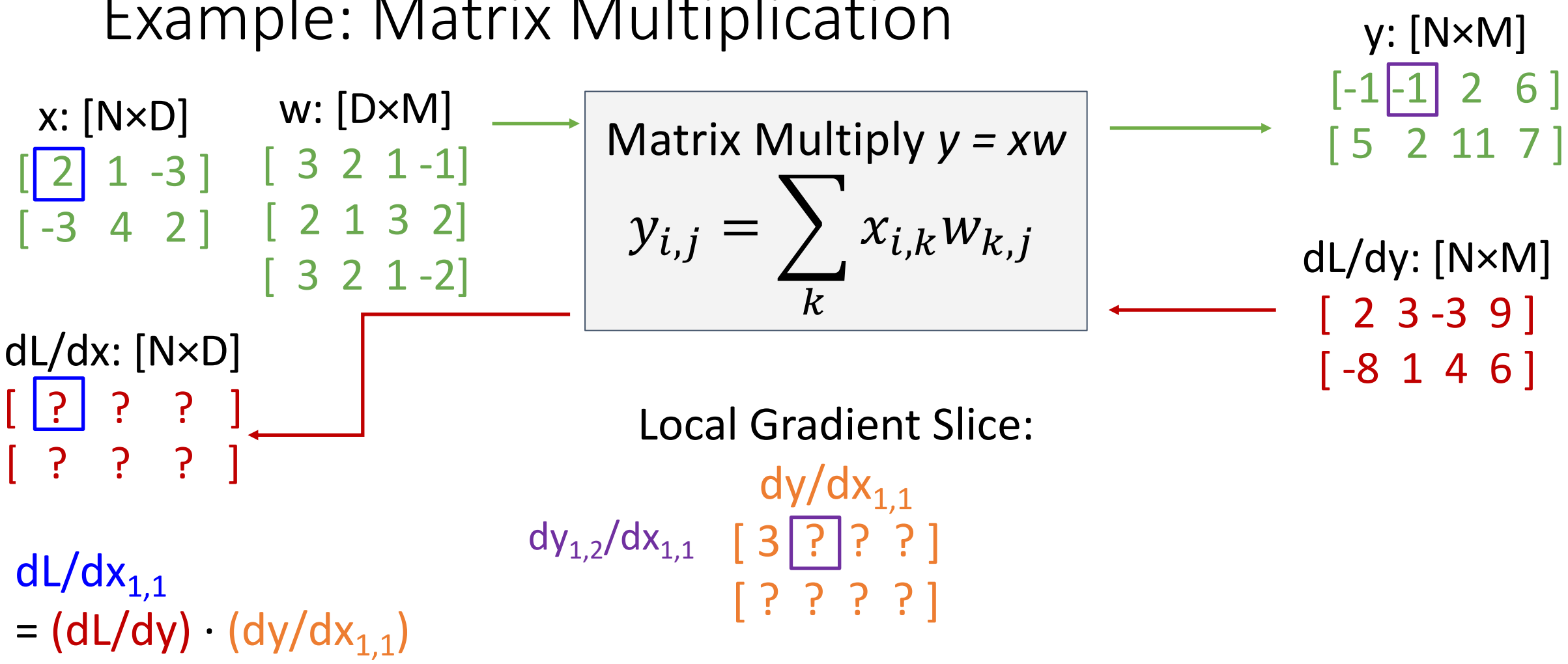
$dy/dx_{1,1}$

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

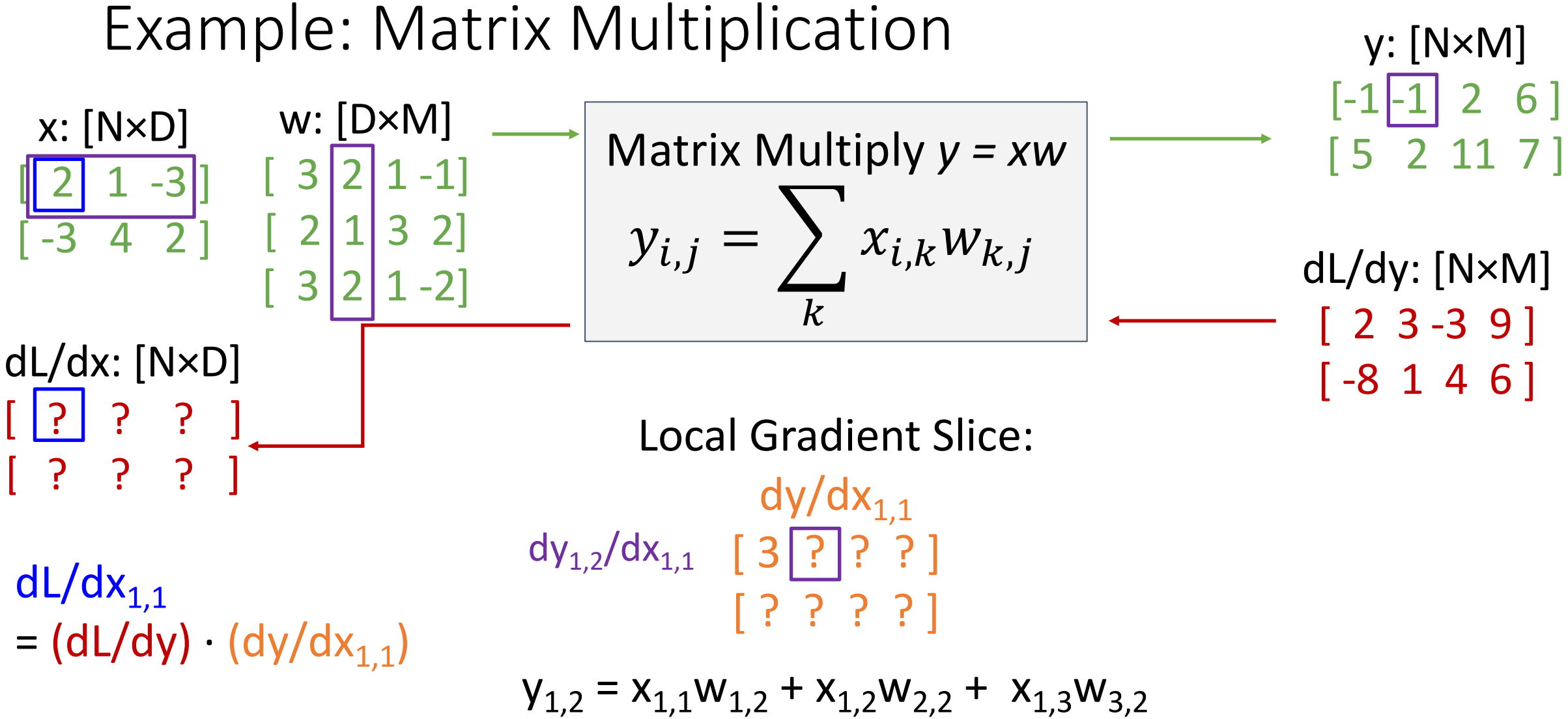
$y_{1,1} = x_{1,1}w_{1,1} + x_{1,2}w_{2,1} + x_{1,3}w_{3,1}$

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

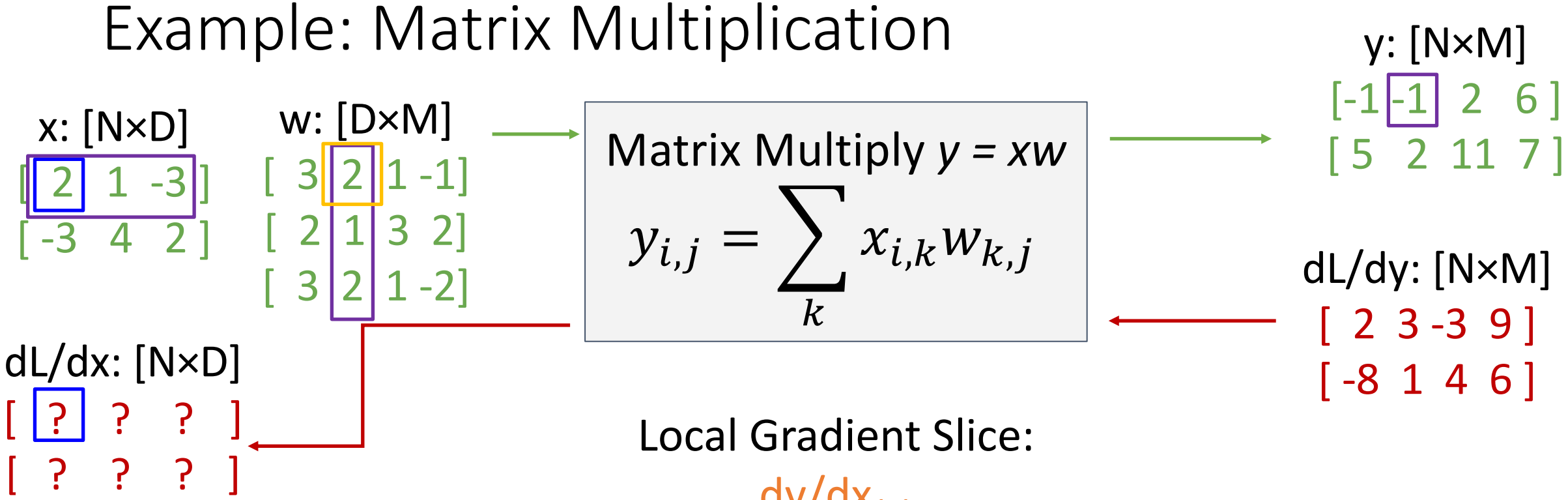
Example: Matrix Multiplication



Example: Matrix Multiplication



Example: Matrix Multiplication



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

Local Gradient Slice:

$dy/dx_{1,1}$

$dy_{1,2}/dx_{1,1}$ [3 2 ? ?]
[? ? ? ?]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

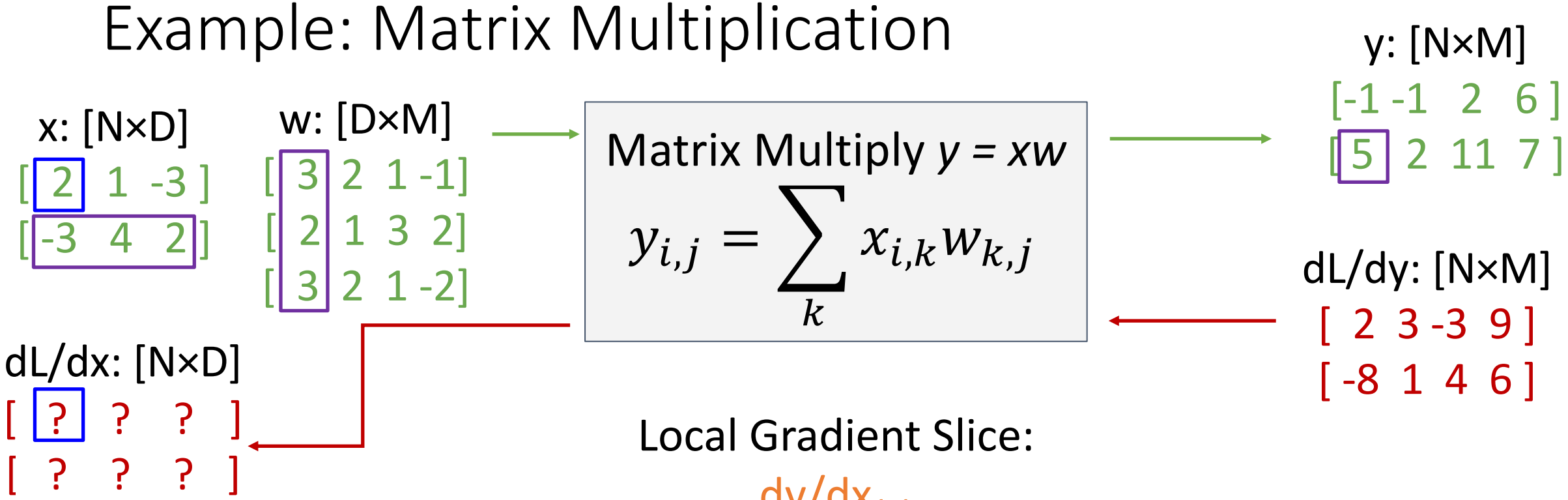
Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication



Local Gradient Slice:

$dy/dx_{1,1}$

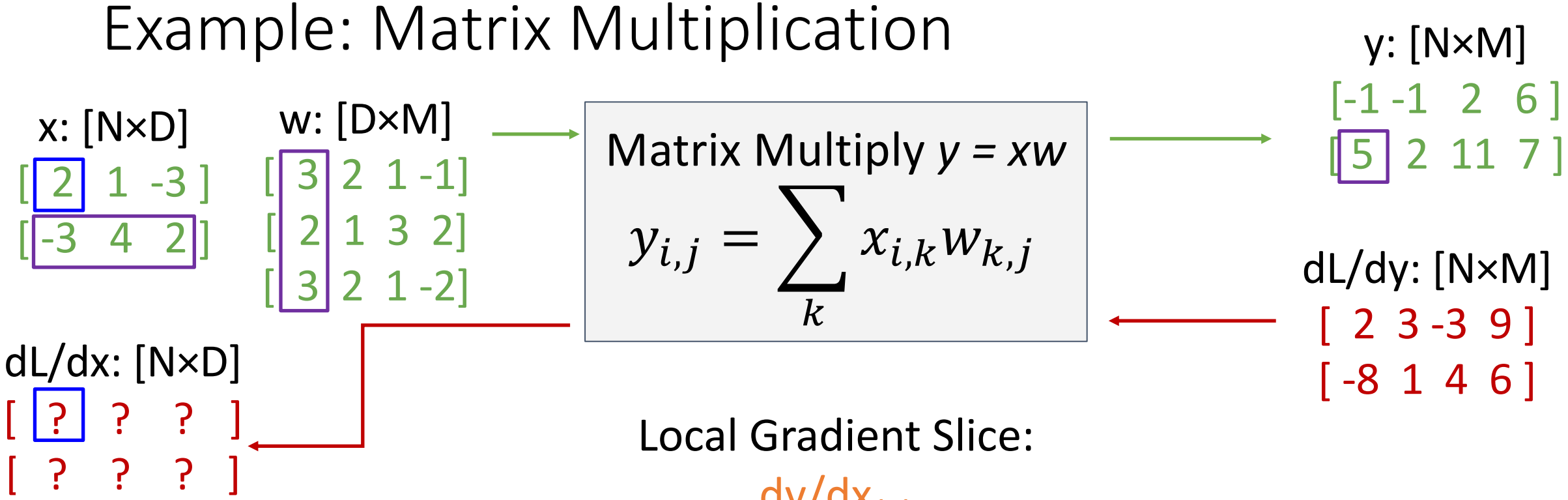
$dy_{1,2}/dx_{1,1}$ [3 2 1 -1]

[? ? ? ?]

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

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

Example: Matrix Multiplication



Local Gradient Slice:

$dy/dx_{1,1}$

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

[0 ? ? ?]

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

$y_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$
$\Rightarrow dy_{2,1}/dx_{1,1} = 0$

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

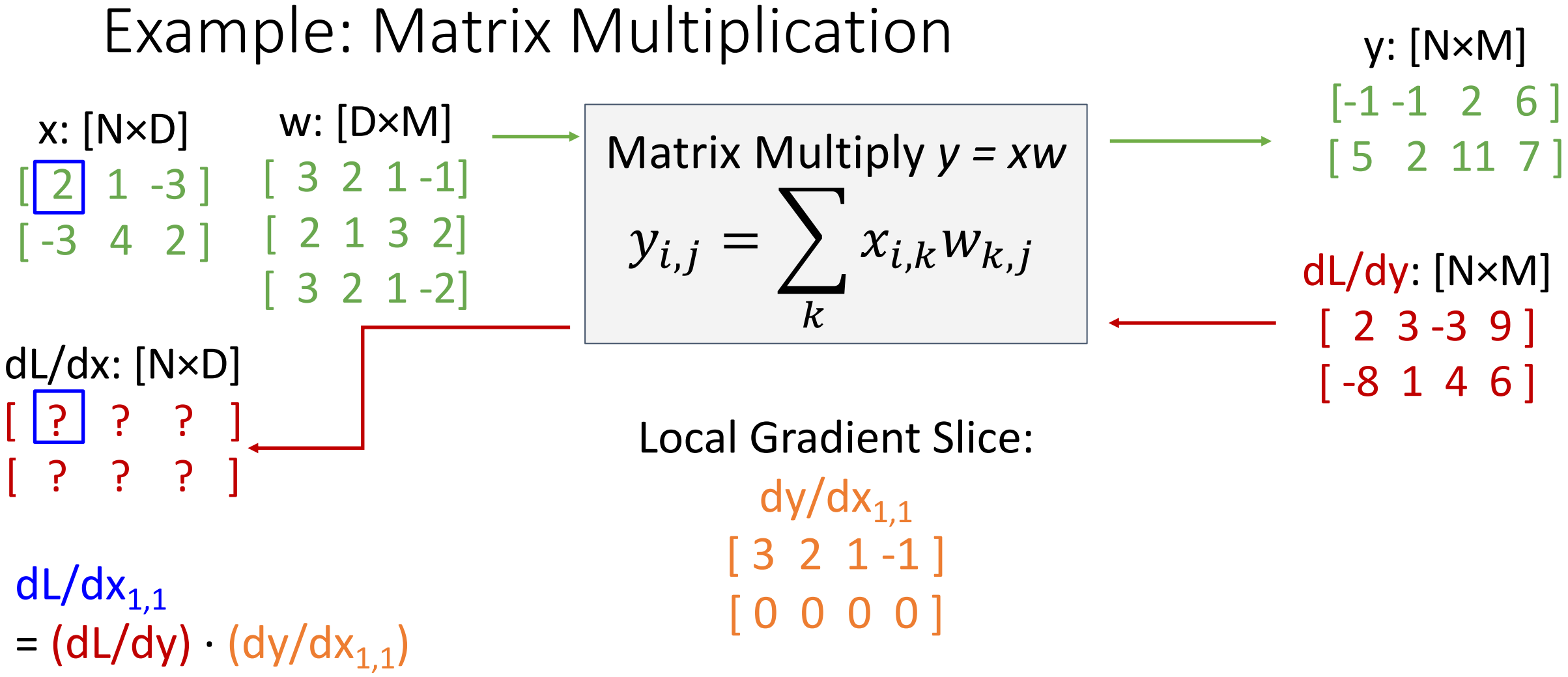
Local Gradient Slice:

$dy/dx_{1,1}$

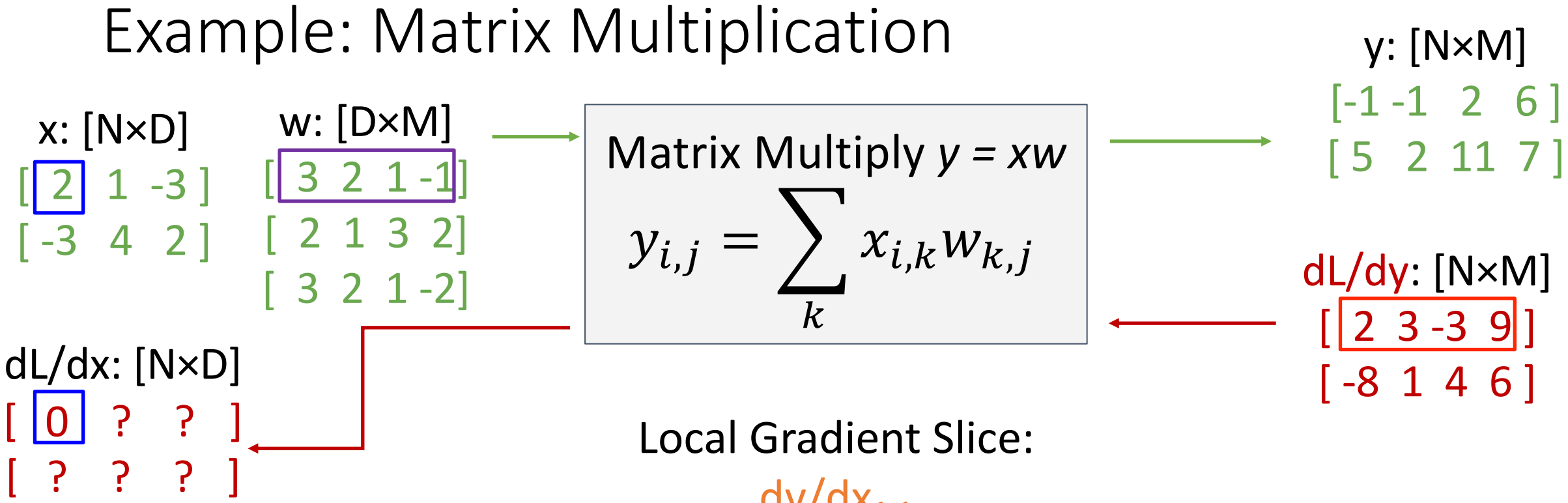
$dy_{1,2}/dx_{1,1}$ [3 2 1 -1]
[0 0 0 0]

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

Example: Matrix Multiplication

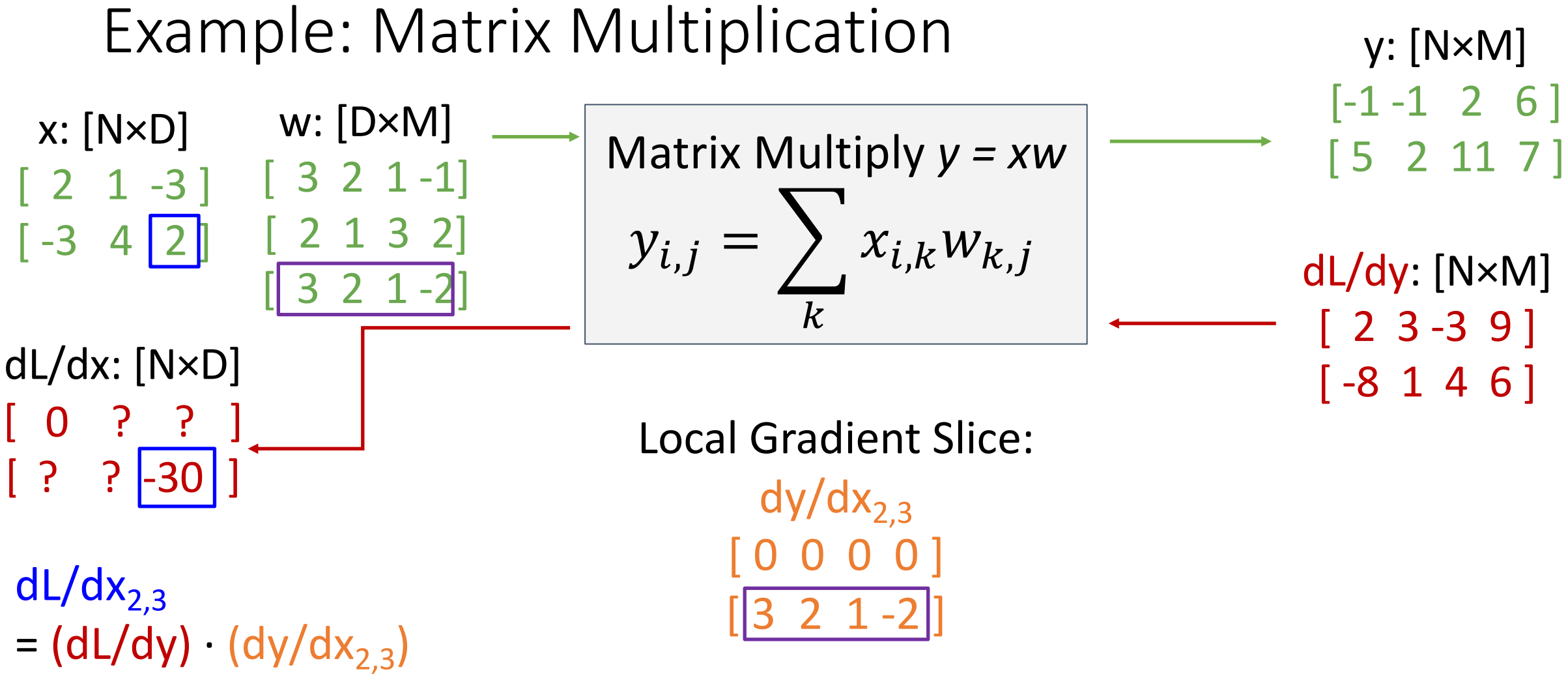


Example: Matrix Multiplication



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

Example: Matrix Multiplication



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 $y = xw$

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

y: [N×M]

[-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 ? ?]
[? ? -30]

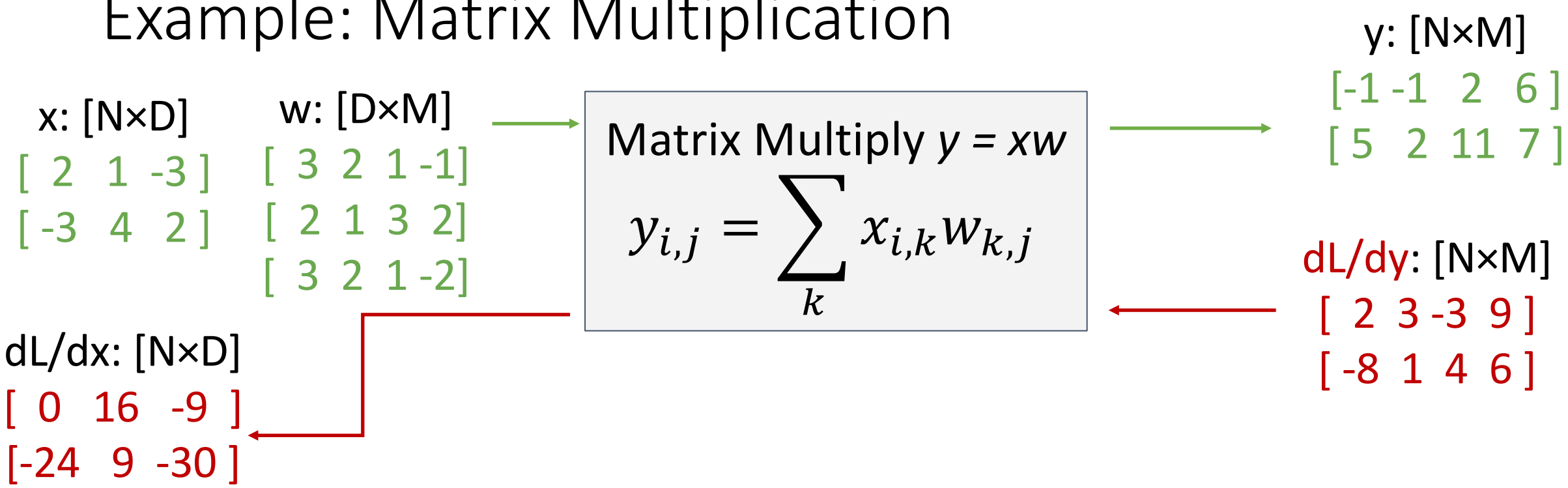
Local Gradient Slice:

$dy/dx_{2,3}$

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

$dL/dx_{2,3}$
$= (dL/dy) \cdot (dy/dx_{2,3})$
$= (dL/dy_{2,:}) \cdot (w_{3,:})$
= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30

Example: Matrix Multiplication



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

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 $y = xw$

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

y: [N×M]
[-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]

It is easy to derive $dy/dx = w^T$

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

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

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

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 $y = xw$

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

y: [N×M]
[-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

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

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

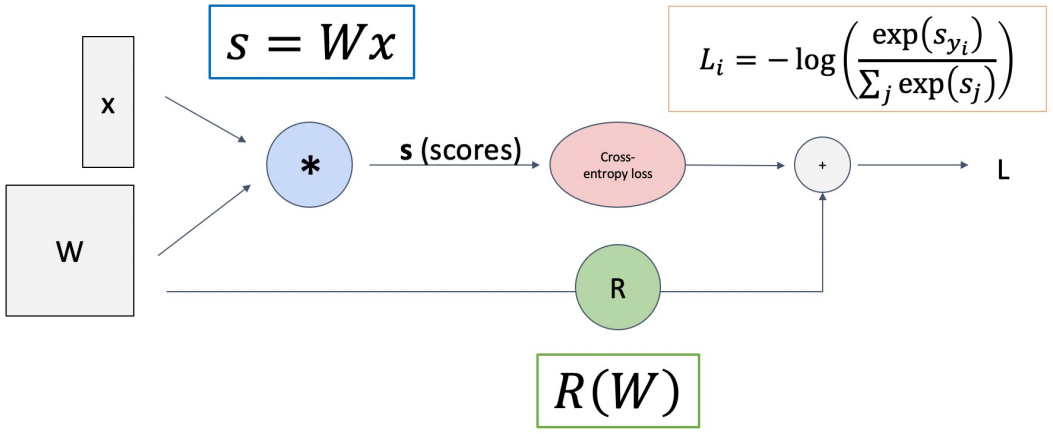
Easy way to remember:
It's the only way the shapes work out!

Reading and Practice

Derive by yourself on the gradient as a practice!
Read D2L textbook (https://d2l.ai/d2l-en.pdf) : Chapter 2.4, Chapter 2.5, Chapter 5.3
Notebooks in D2L textbook: Chapter 5. Multilayer Perceptrons

Summary

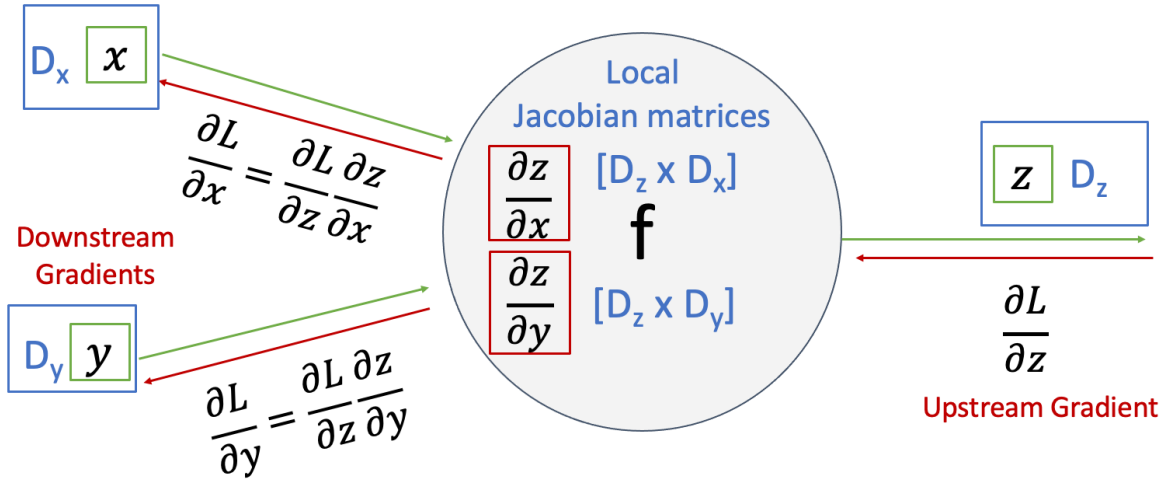
Represent complex expressions as **computational graphs**



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**

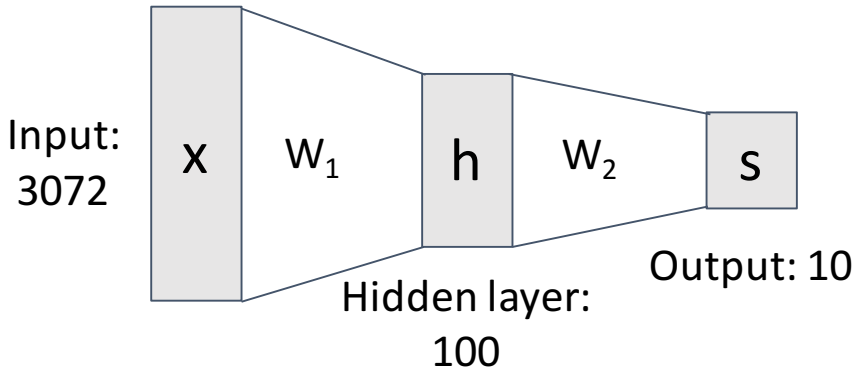


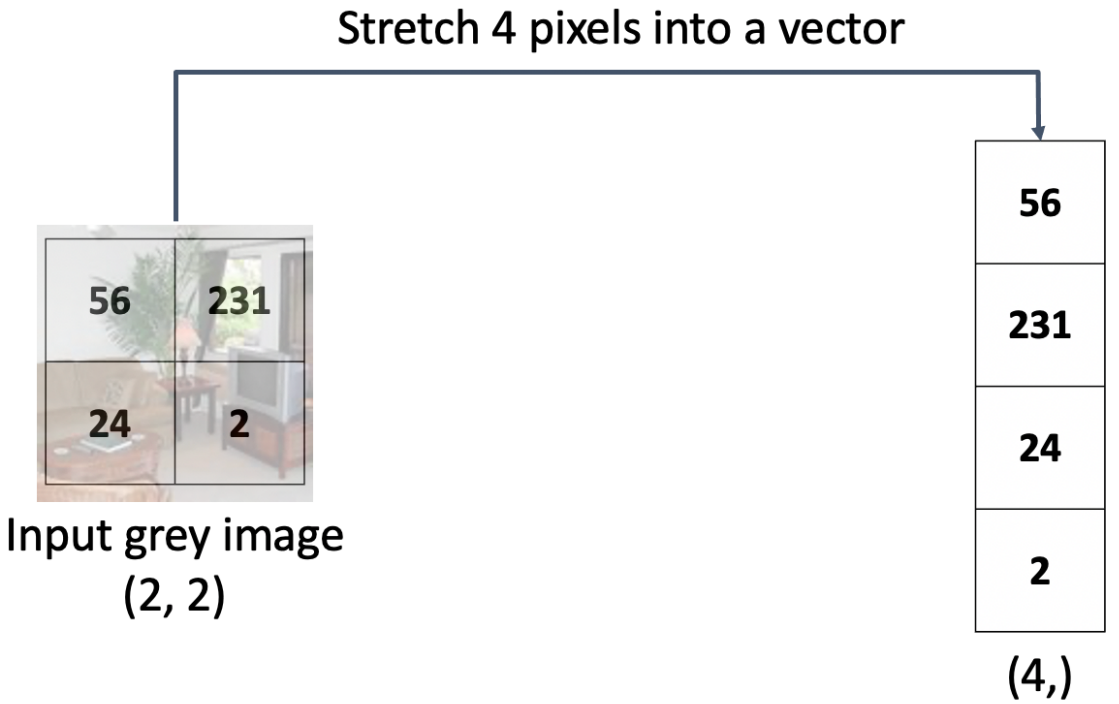
Problem: So far our classifiers don't respect the spatial structure of images!

Linear NN: $f = W_1 x$

Mult-Layer Perceptron:

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





Next time:
Convolutional Neural Networks