Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$
$$= qz \text{ where } q = (x + y)$$

Clearly the partial derivatives of the subexpressions are trivial:

$$\partial f/\partial z = q \qquad \partial f/\partial q = z$$
$$\partial q/\partial x = 1 \qquad \partial q/\partial y = 1$$

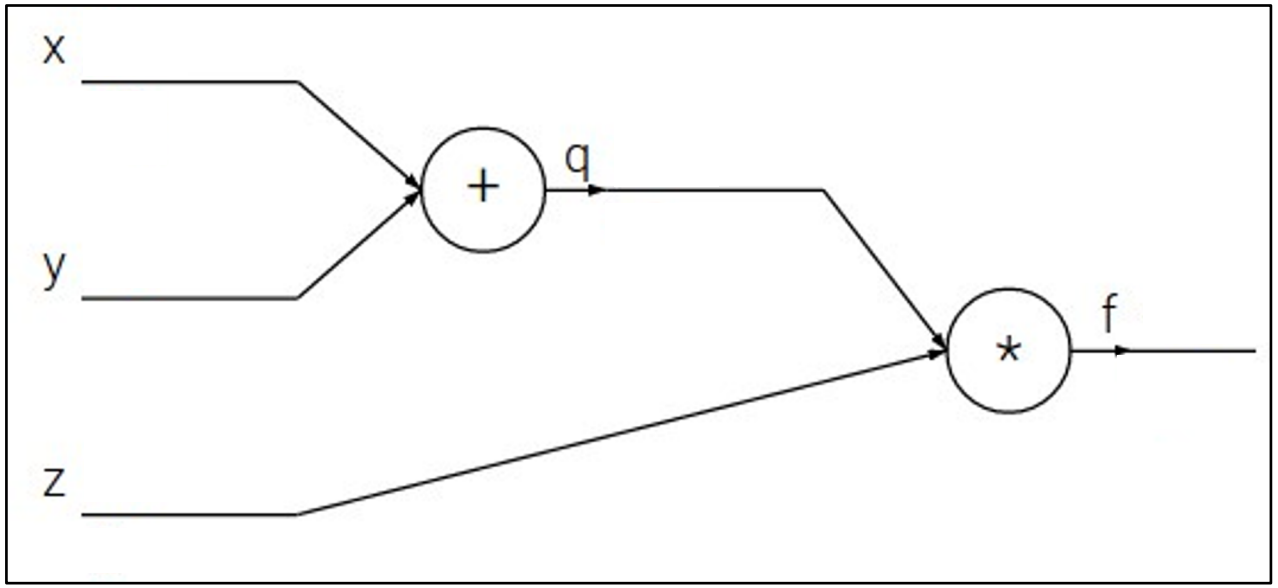
and the chain rule tells us how to combine these:

$$\partial f/\partial x = \partial f/\partial q \; \partial q/\partial x = z$$
$$\partial f/\partial y = \partial f/\partial q \; \partial q/\partial y = z$$

$$\nabla_{[x,y,z]} f = [z, z, q]$$

Backpropagation: Simple Example

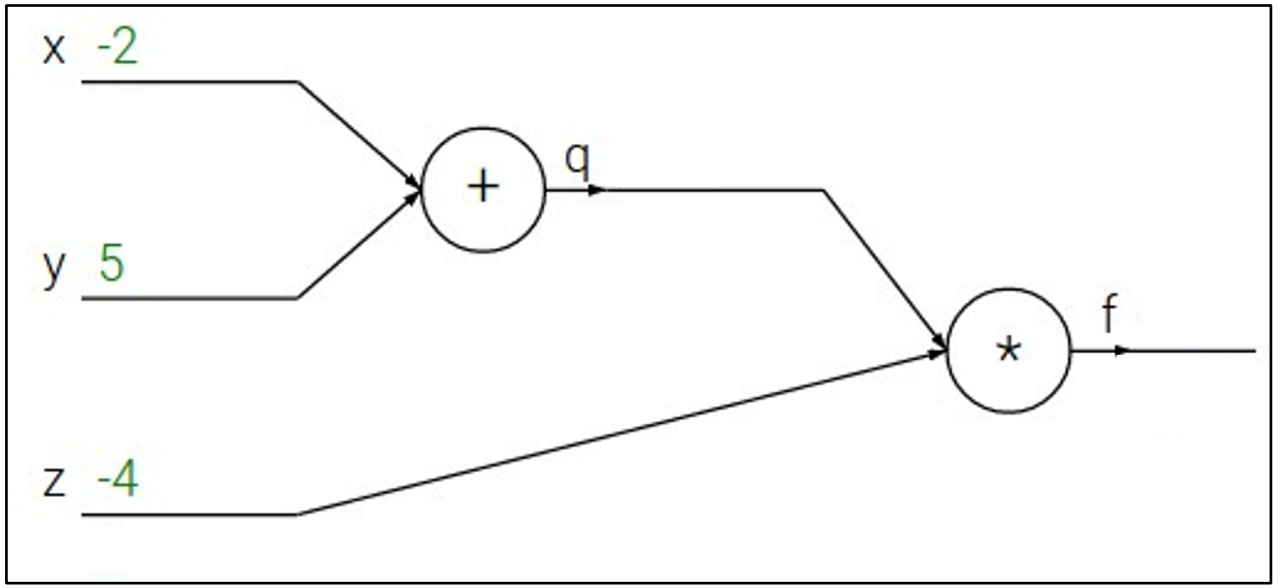
$$f(x, y, z) = (x + y)z$$



Backpropagation: Simple Example

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

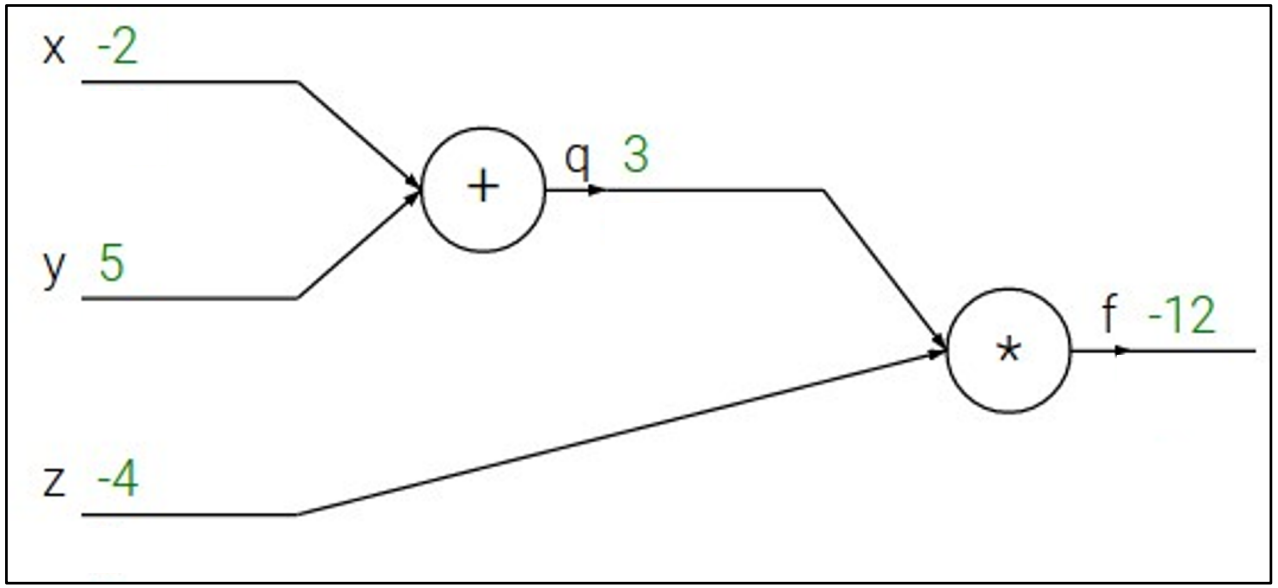
e.g. x = -2, y = 5, z = -4



Backpropagation: Simple Example

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

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



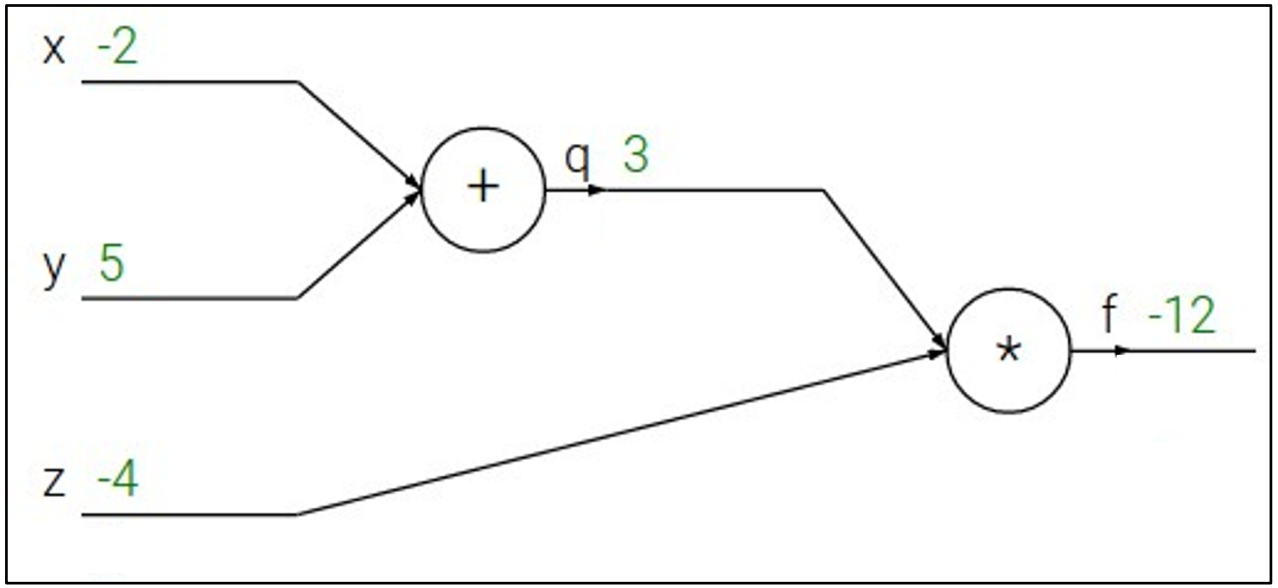
1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

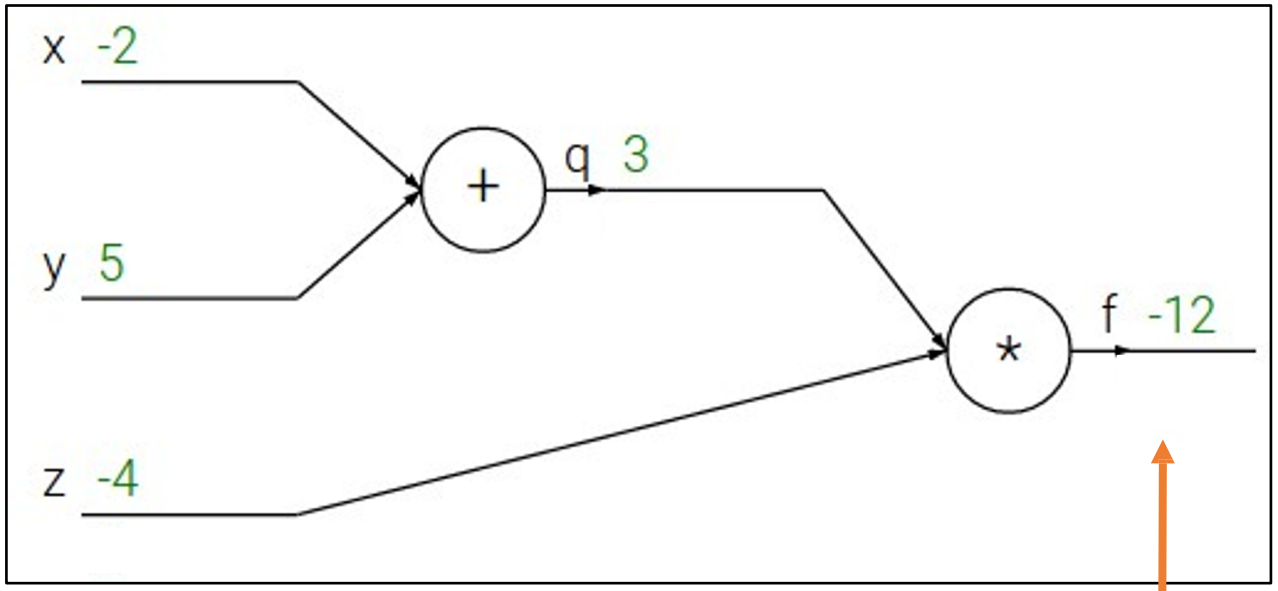
2. **Backward pass**: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

2. **Backward pass**: Compute derivatives

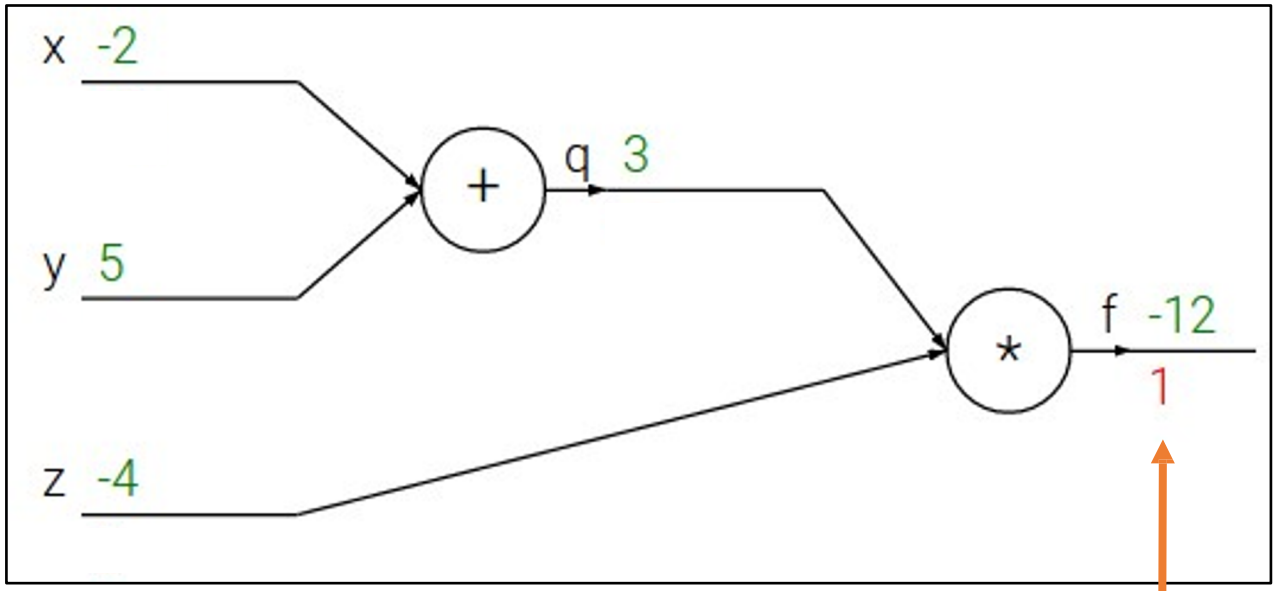
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

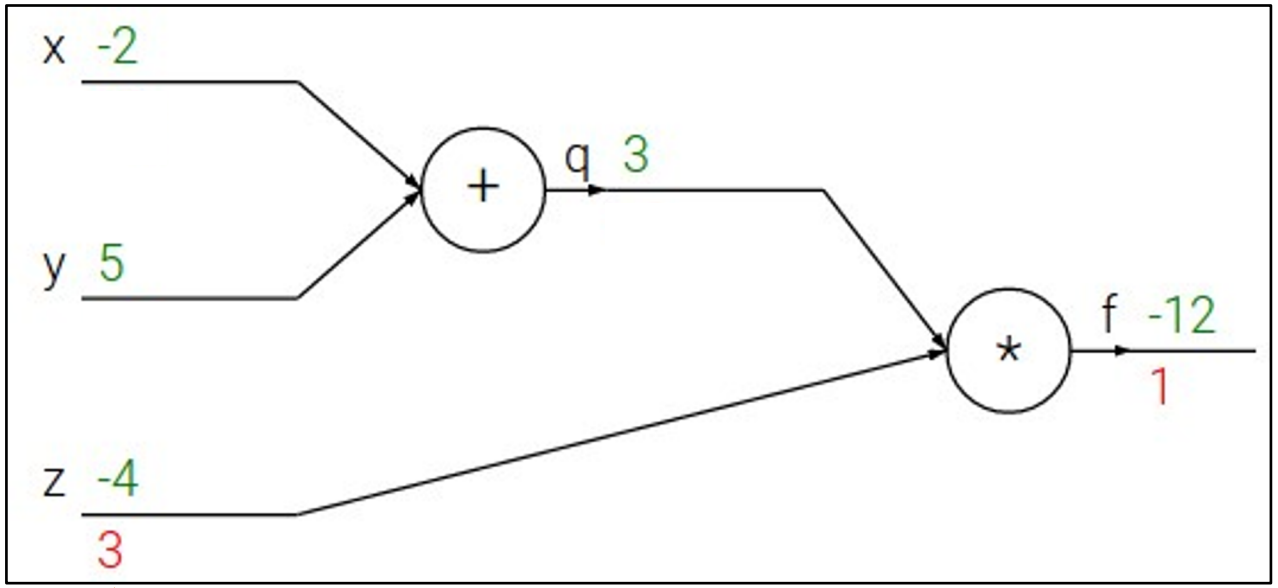
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

2. **Backward pass**: Compute derivatives

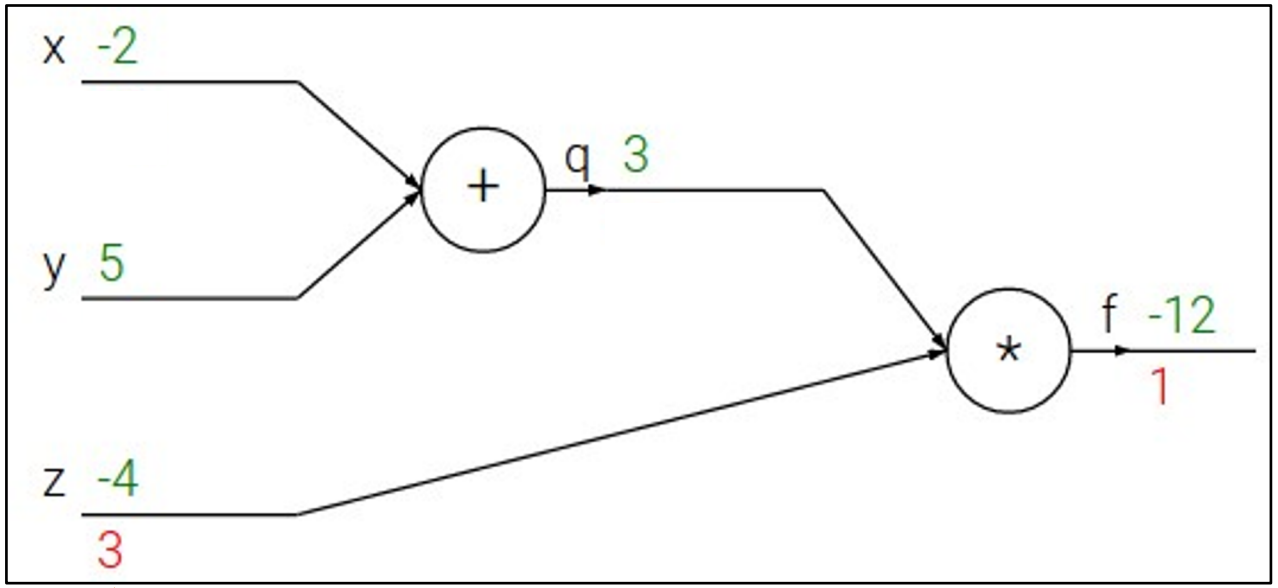
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

2. **Backward pass**: Compute derivatives

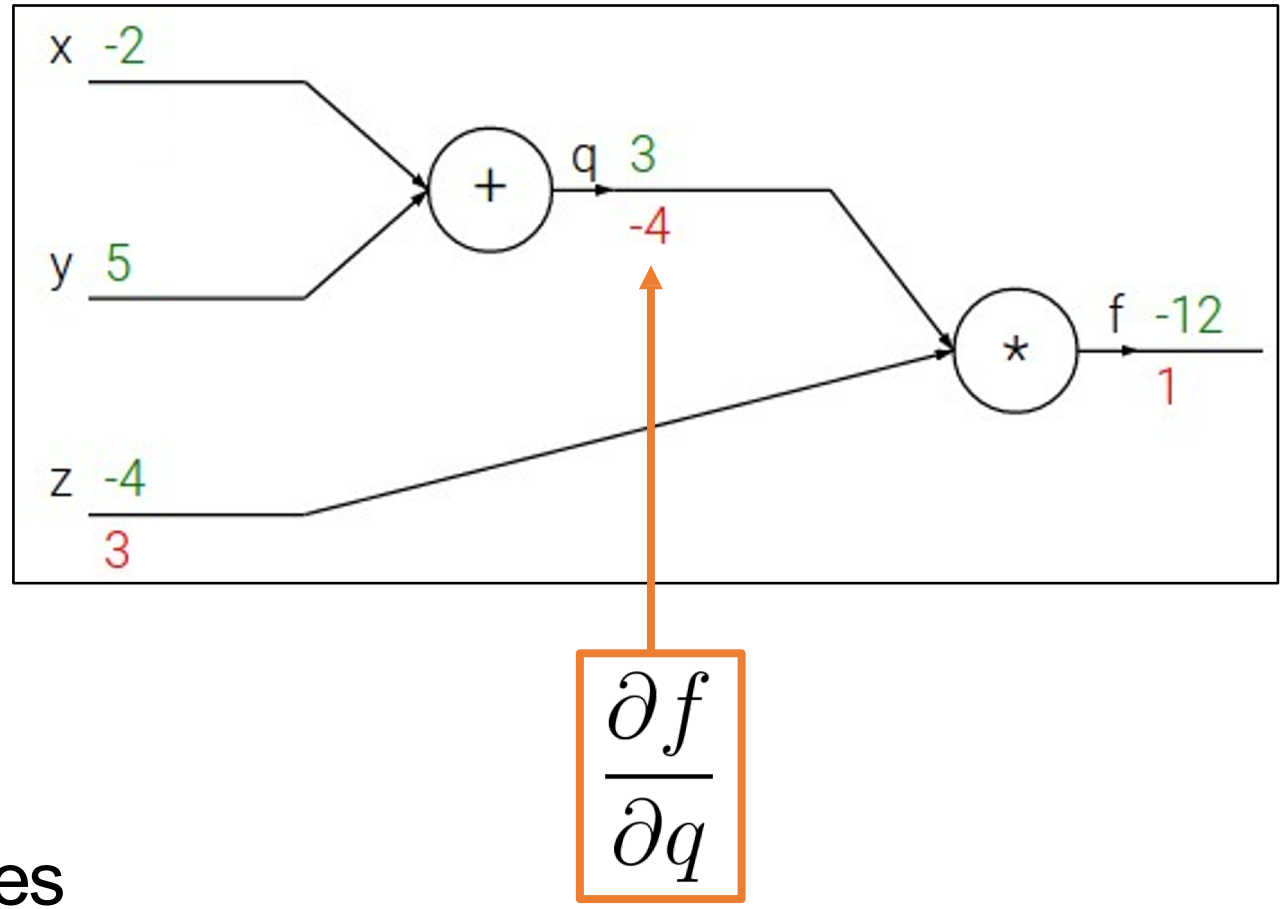
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

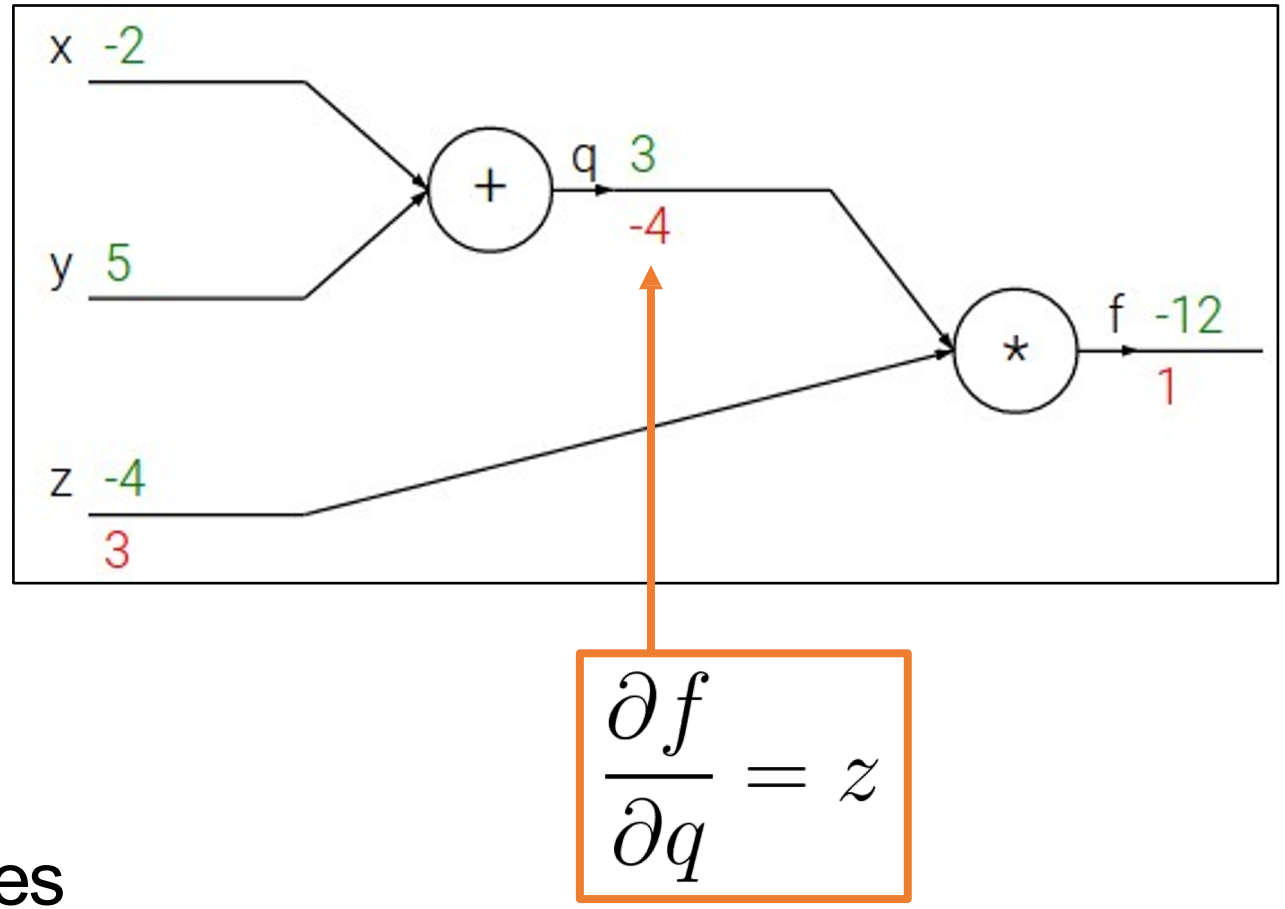
2. **Backward pass**: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

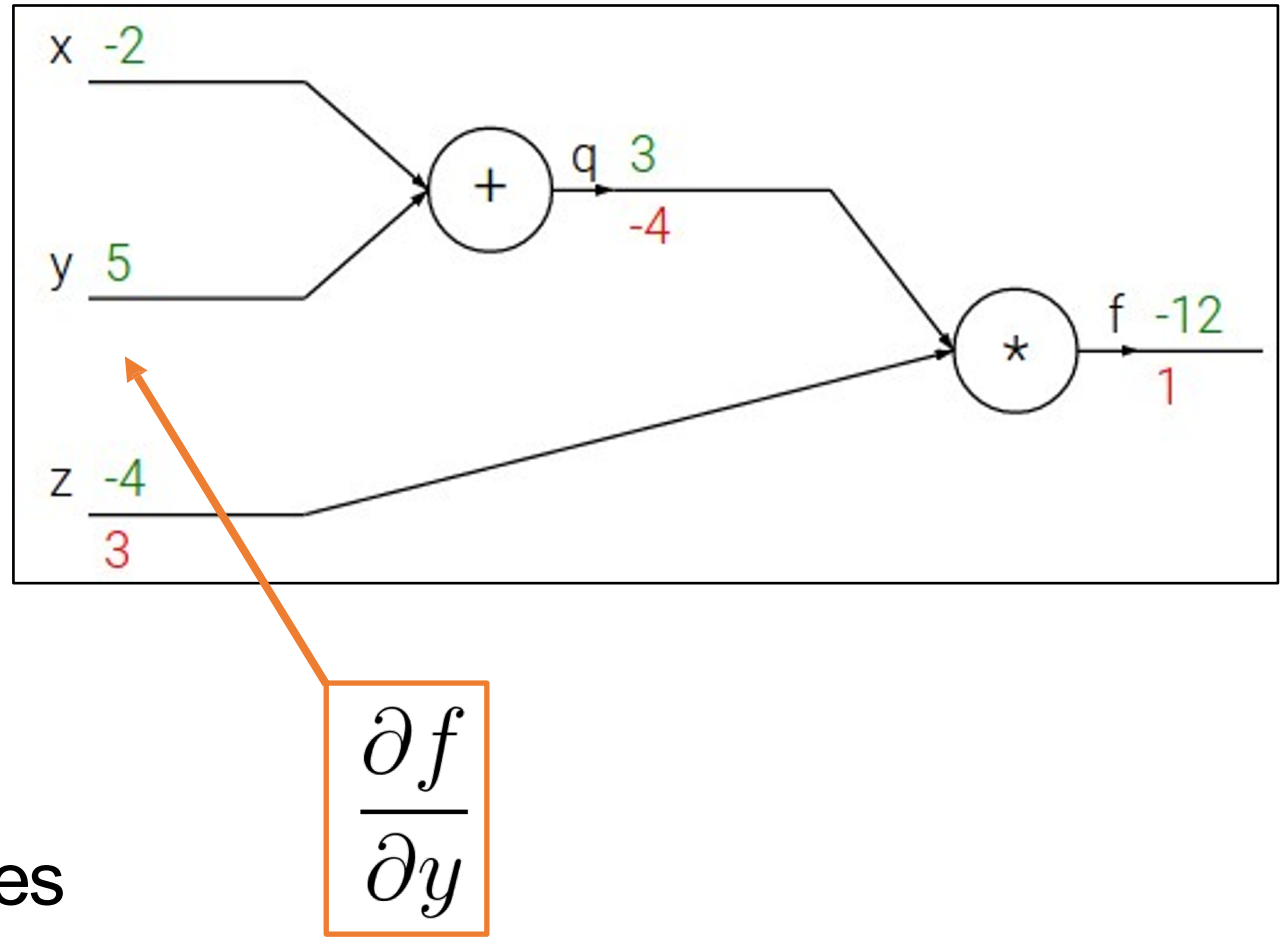
2. **Backward pass**: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. **Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

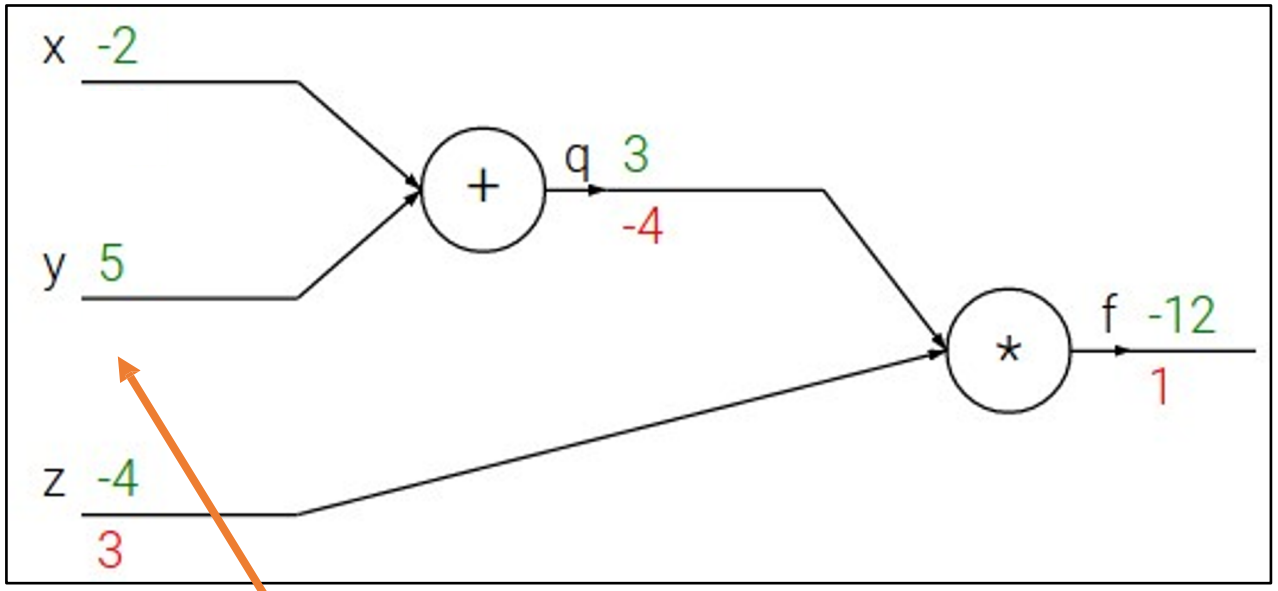
2. **Backward pass**: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

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

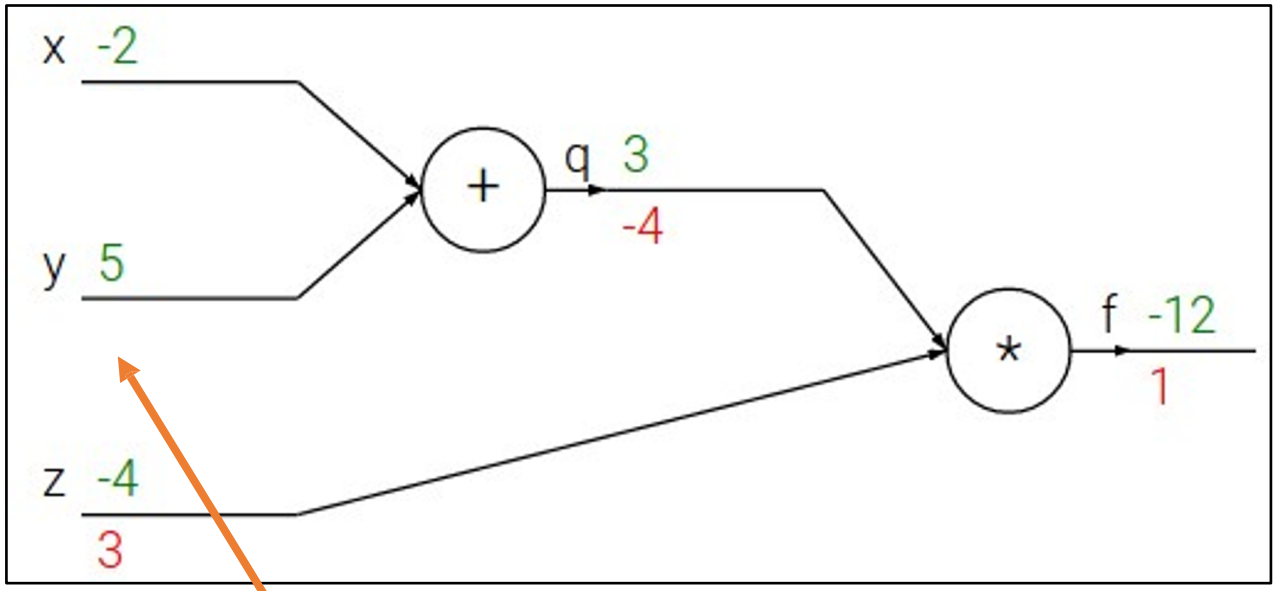
Chain Rule

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

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

Chain Rule

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

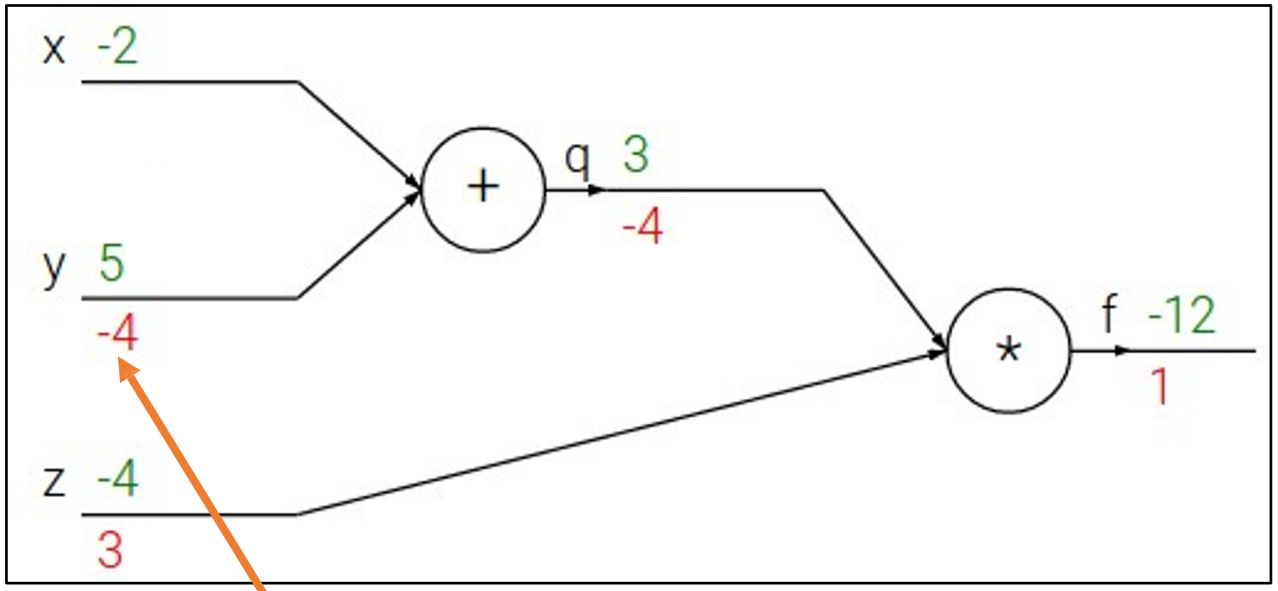
$$\frac{\partial q}{\partial y} = 1$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

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

Chain Rule

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

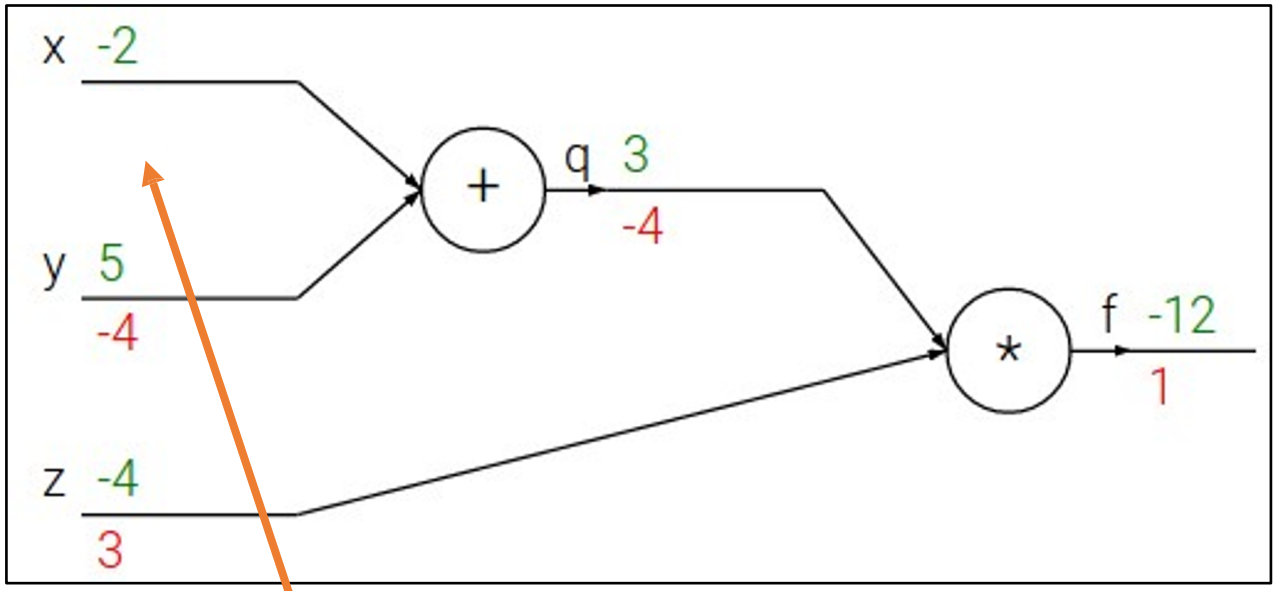
$$\frac{\partial q}{\partial y} = 1$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

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

Chain Rule

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

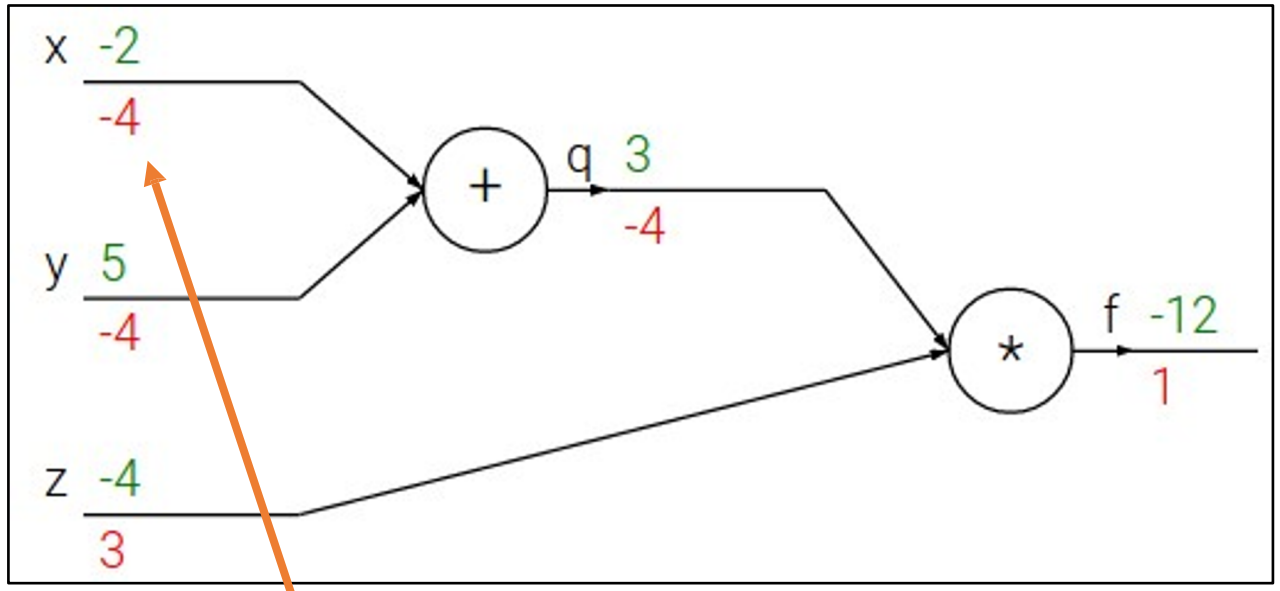
$$\frac{\partial q}{\partial x} = 1$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

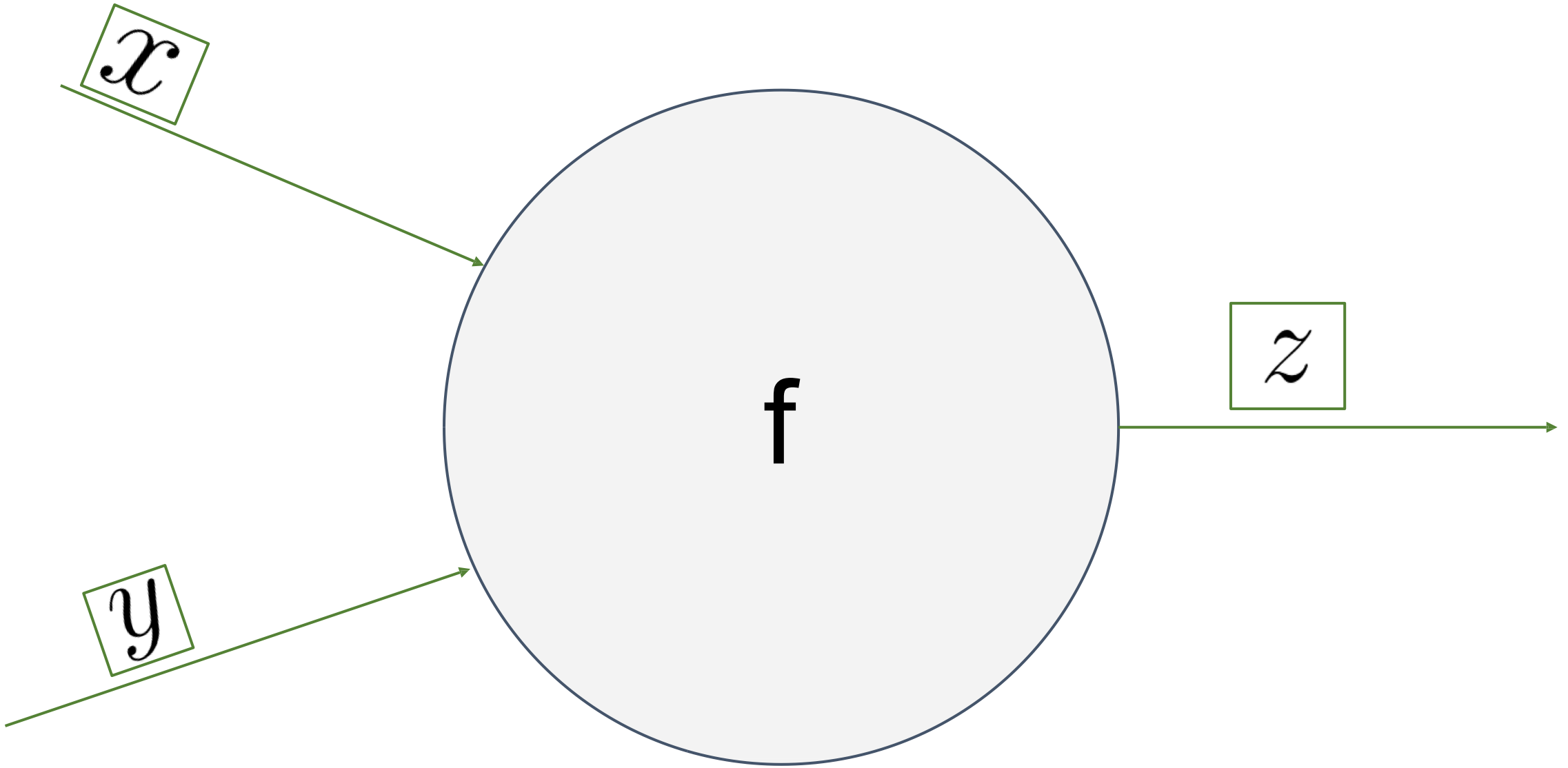
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

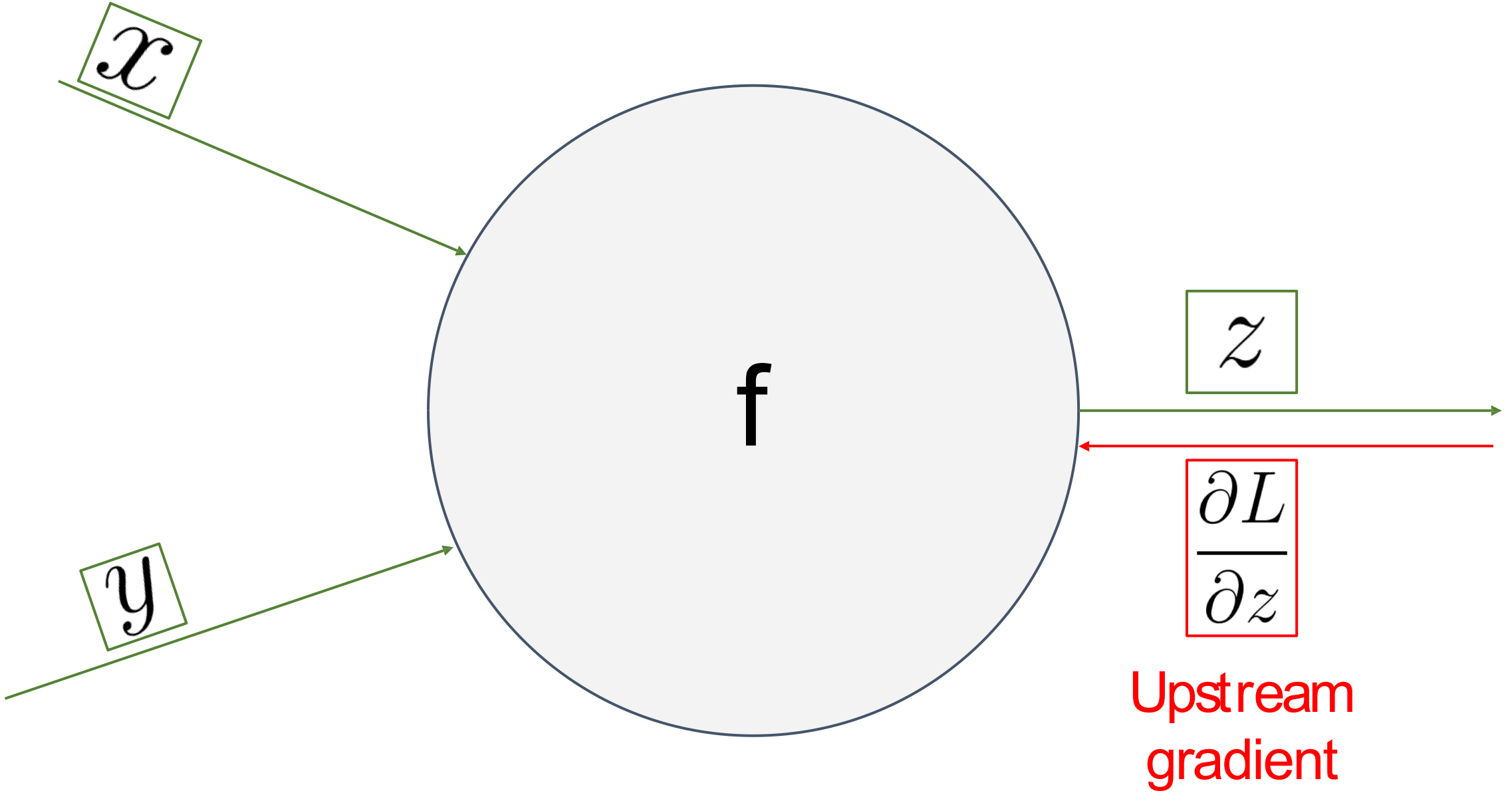
Chain Rule

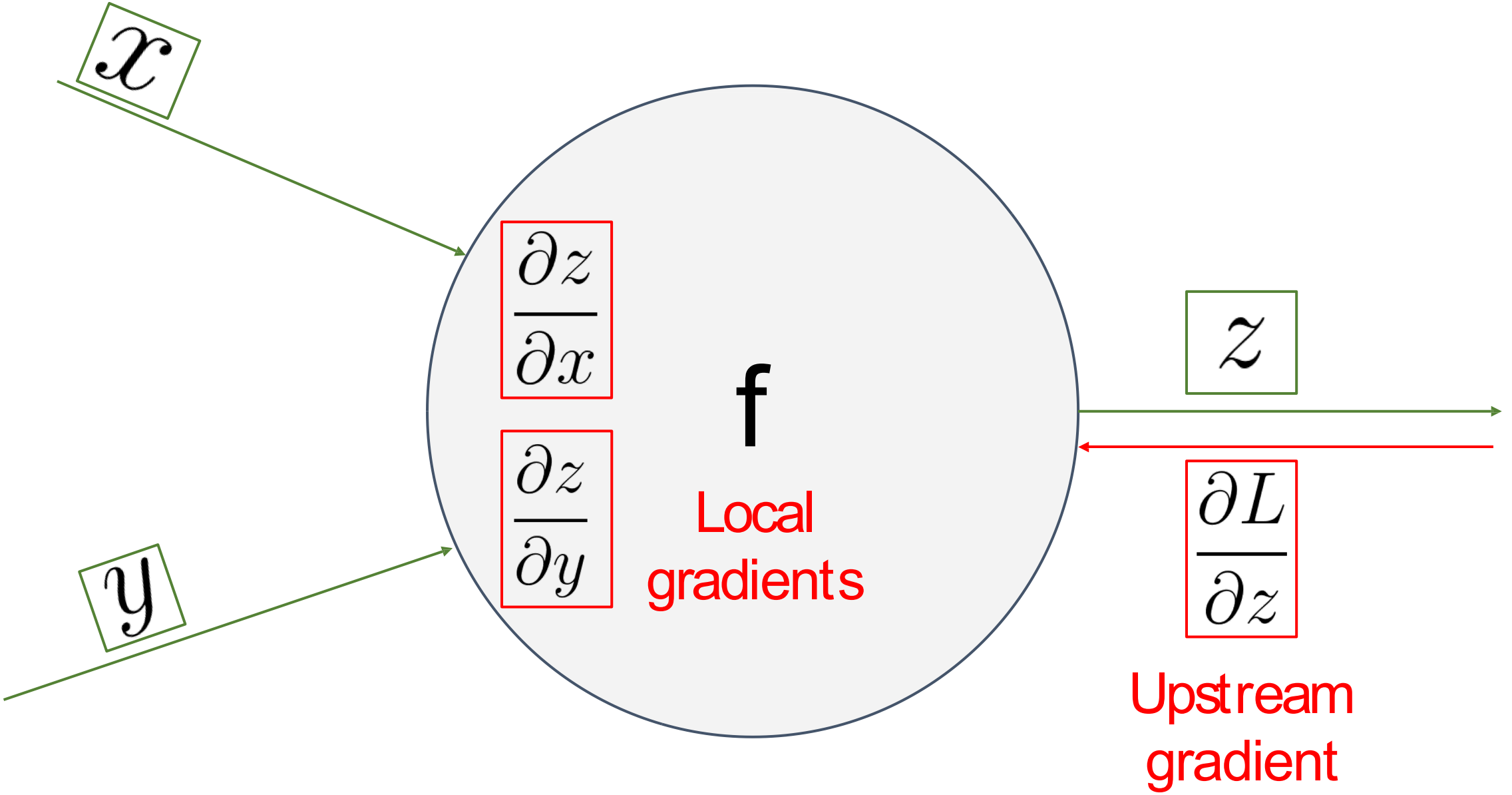
$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

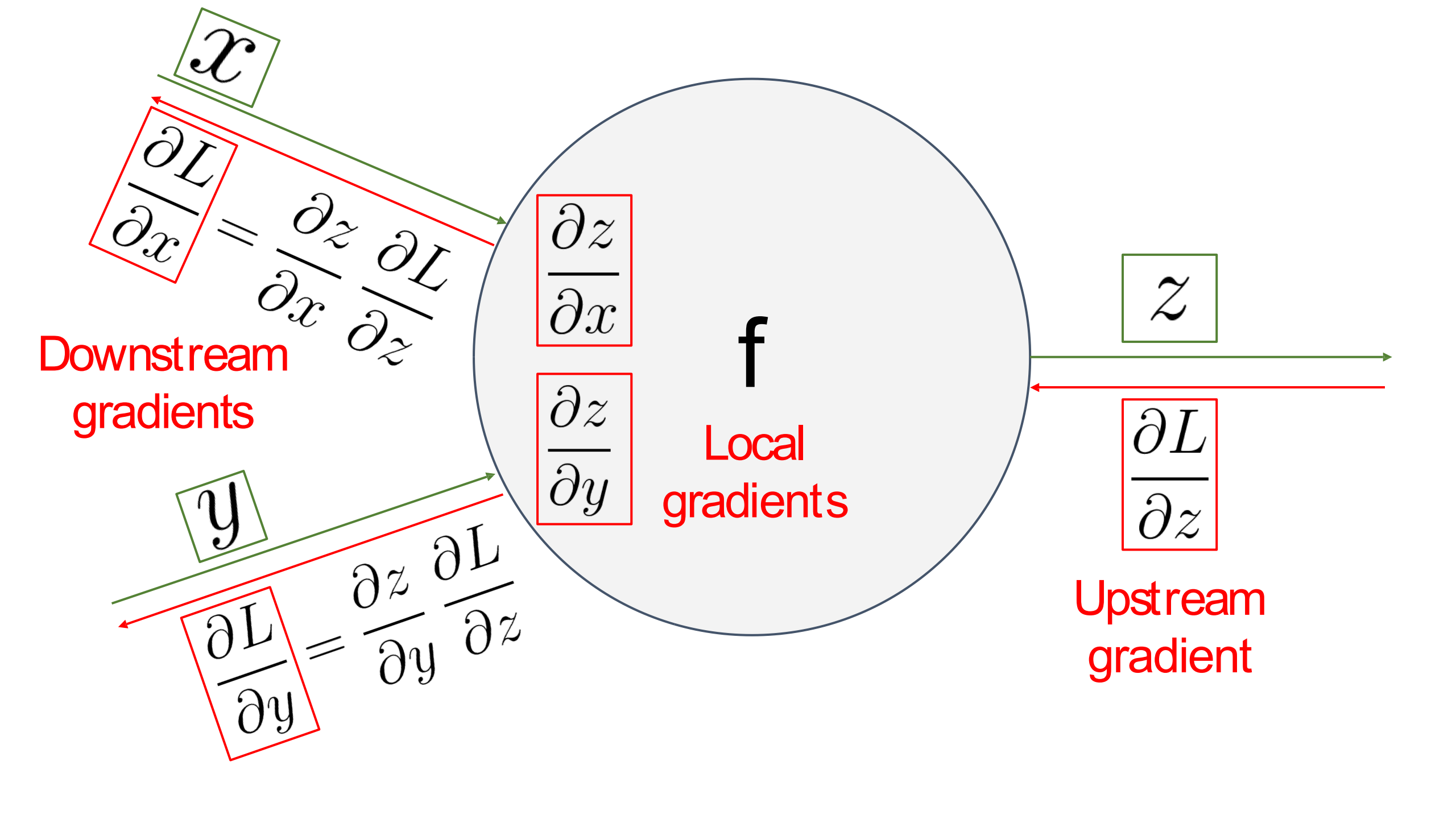
$$\frac{\partial q}{\partial x} = 1$$

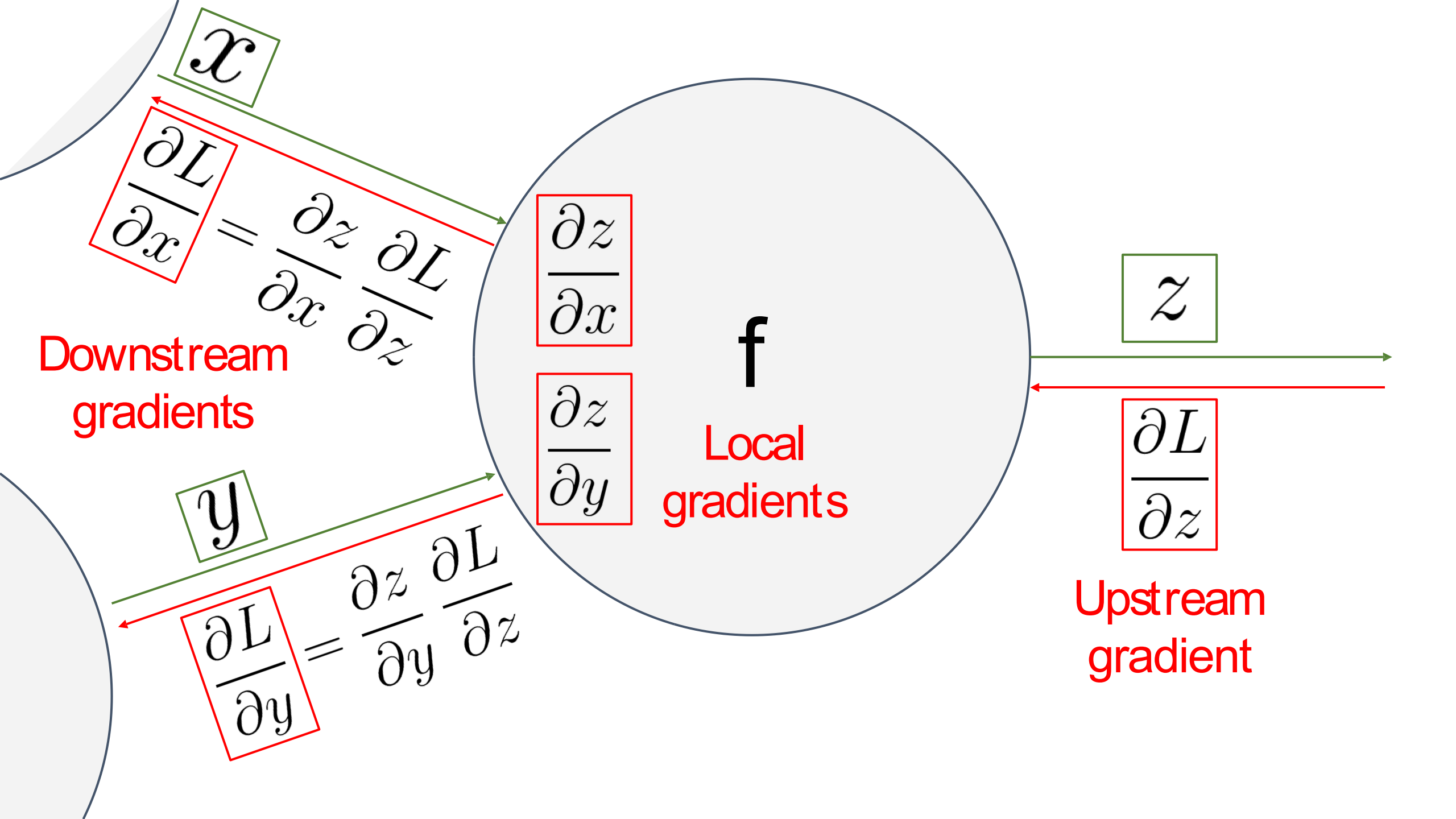
Downstream Gradient **Local Gradient** **Upstream Gradient**


x
f
z
y

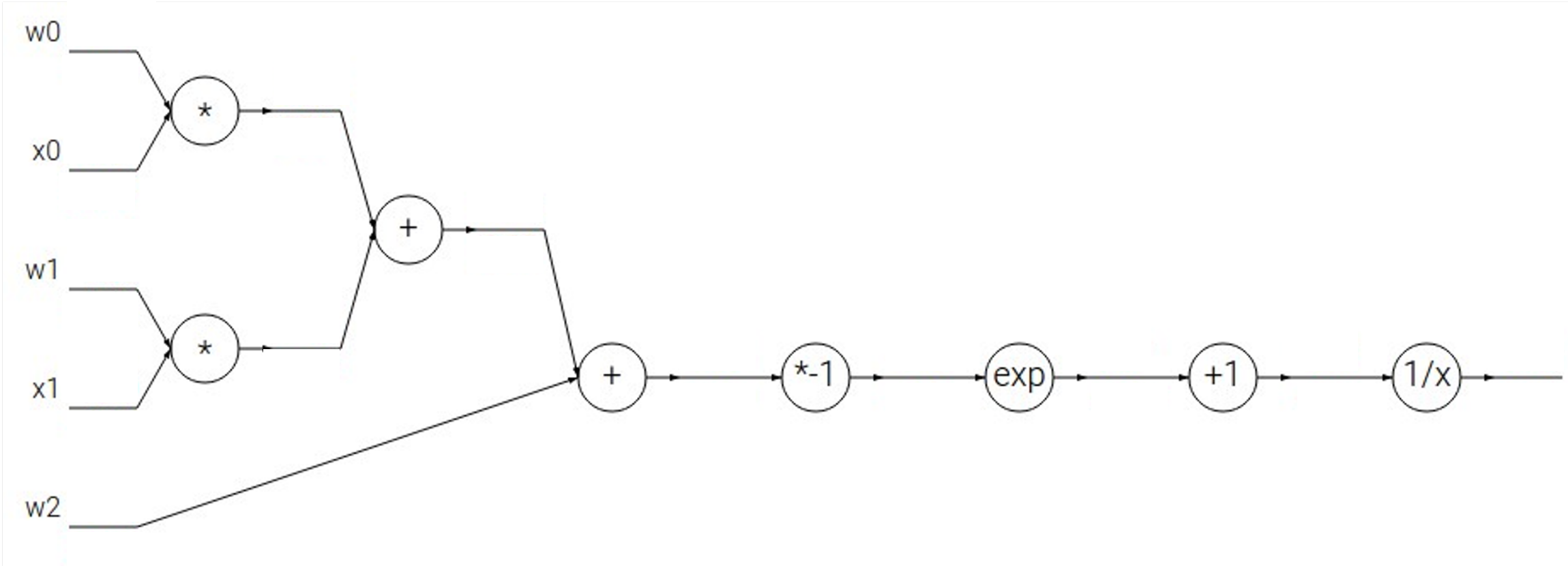

x
y
f
z
$\frac{\partial L}{\partial z}$
Upstream gradient


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

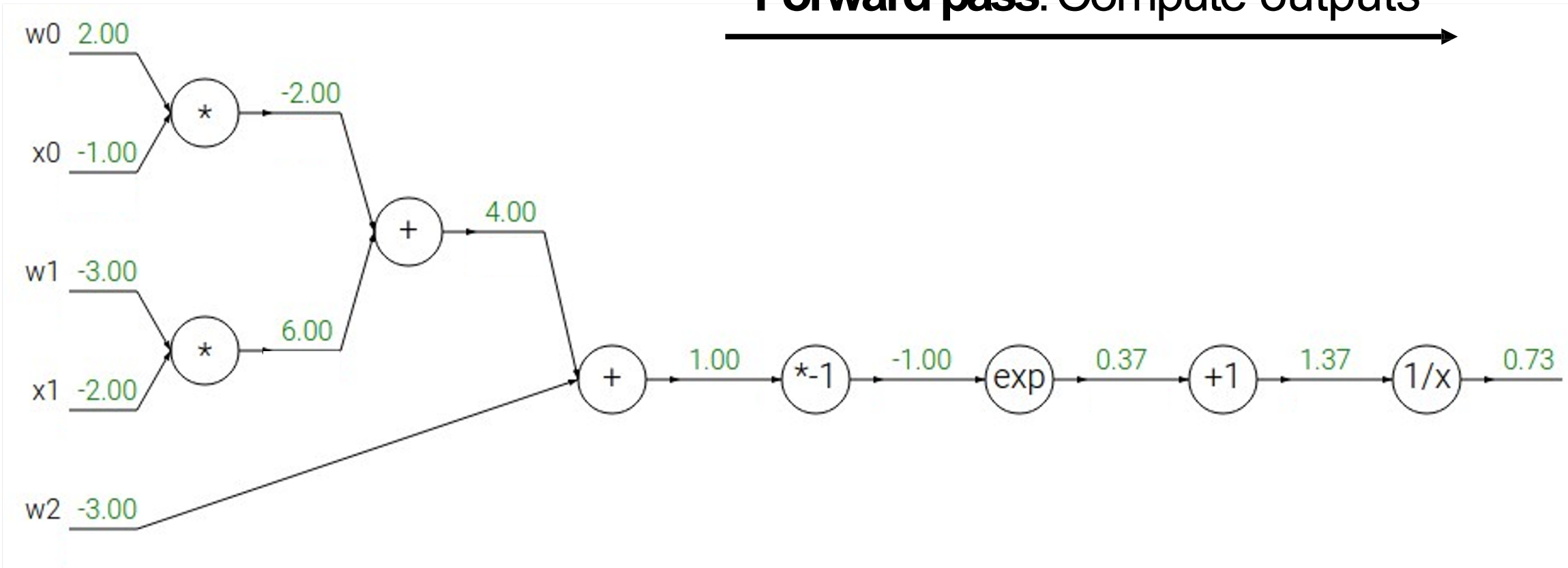

x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial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

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



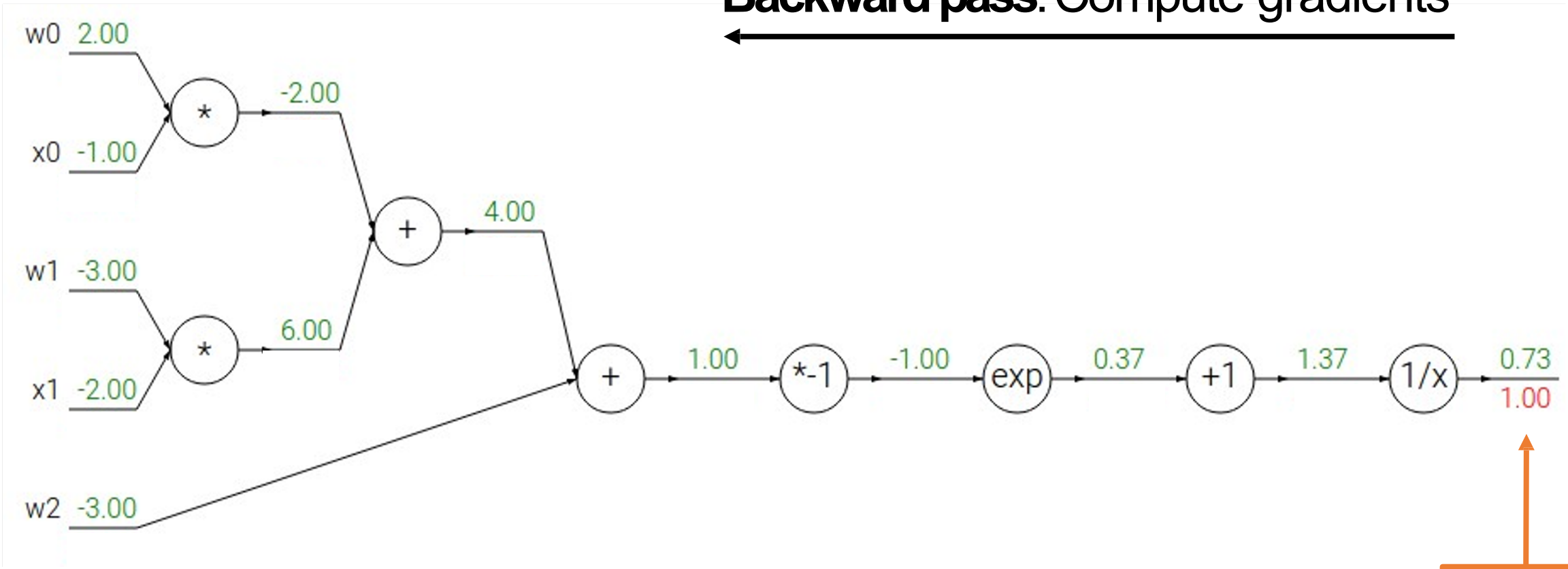
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Forward pass: Compute outputs



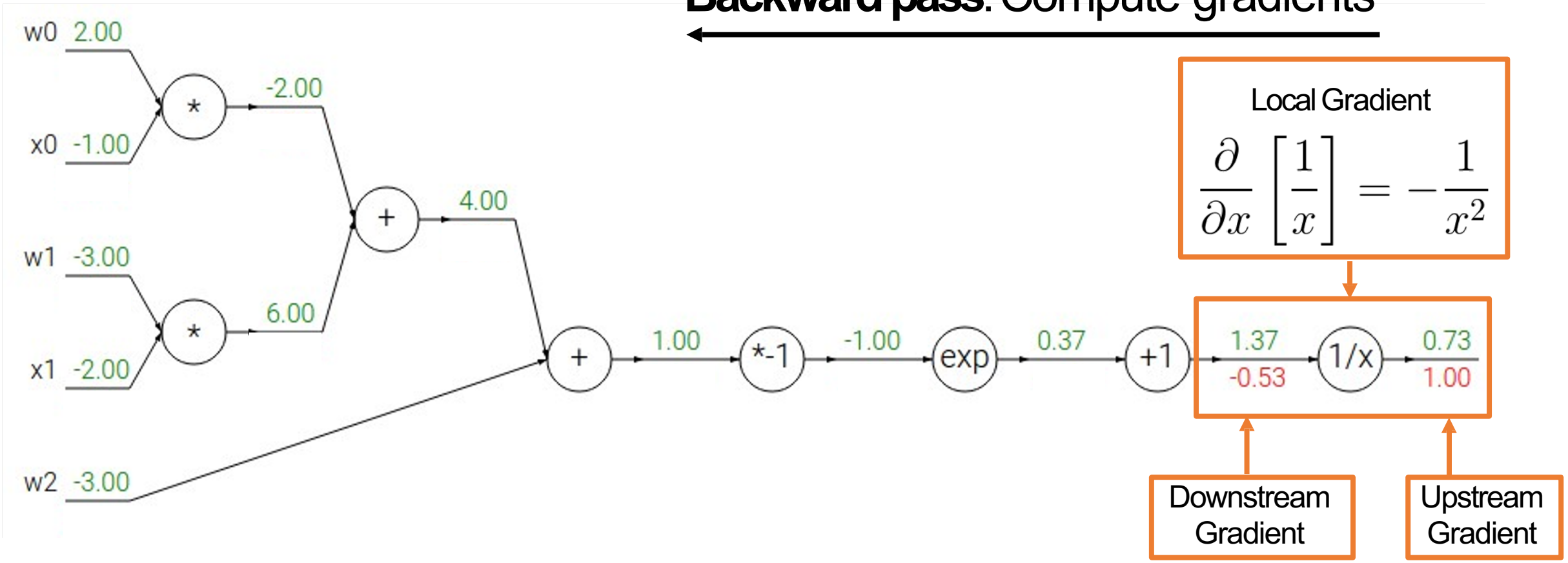
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



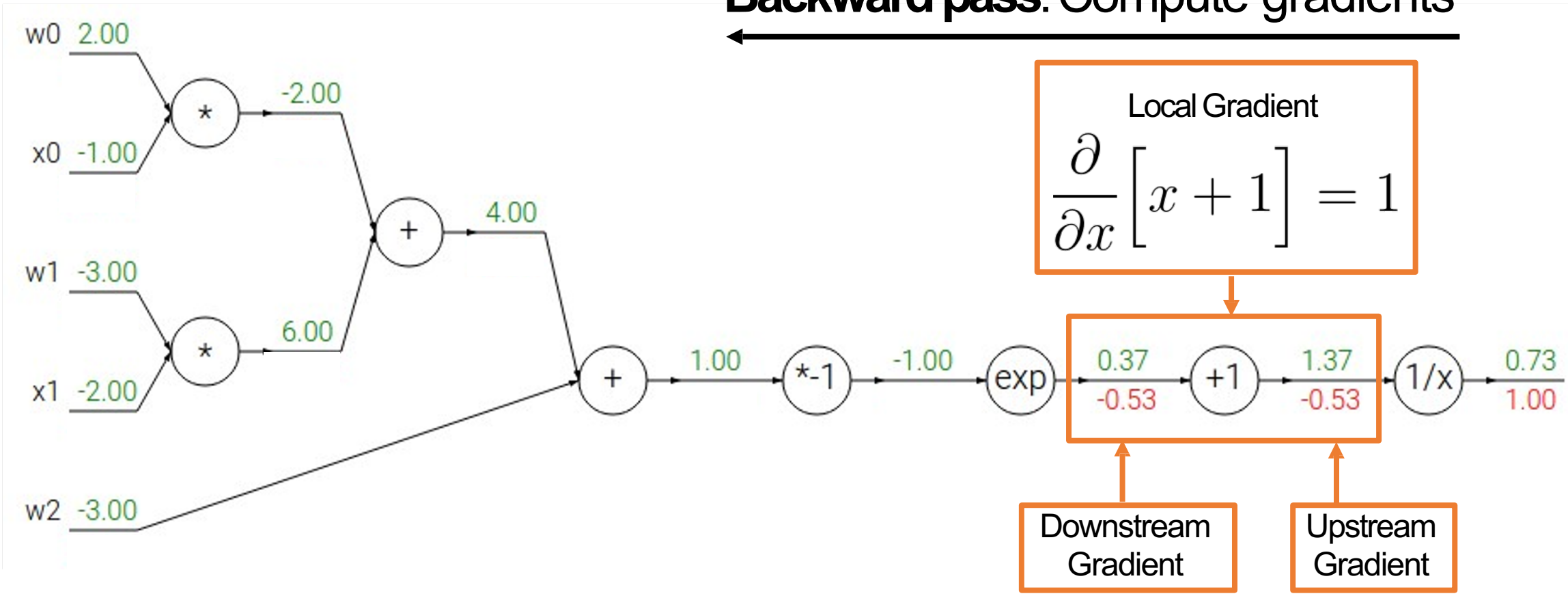
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



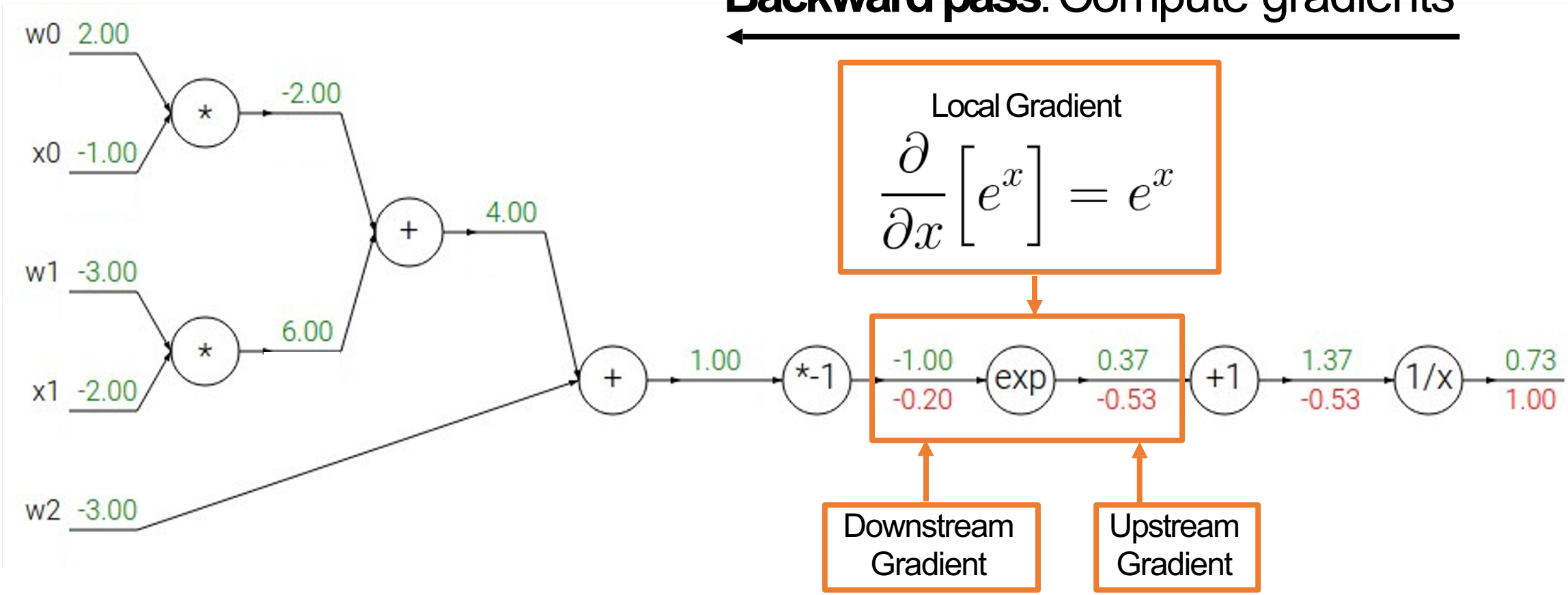
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



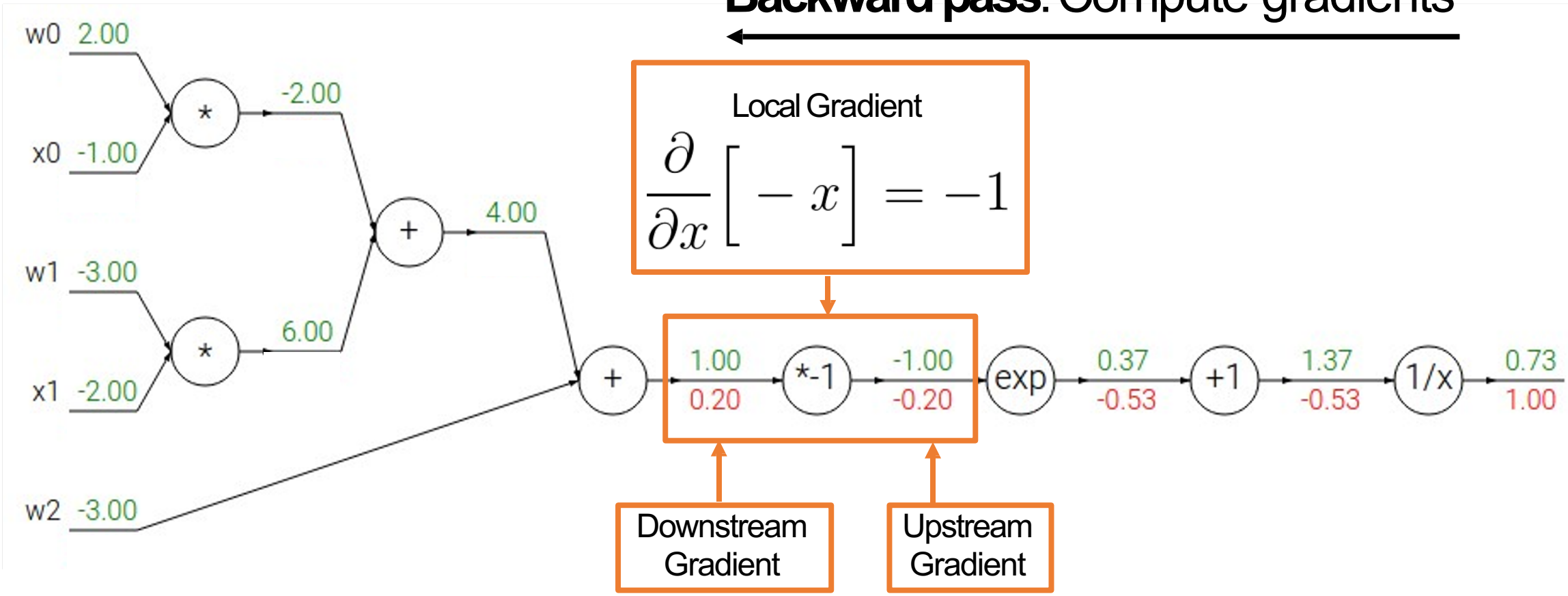
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



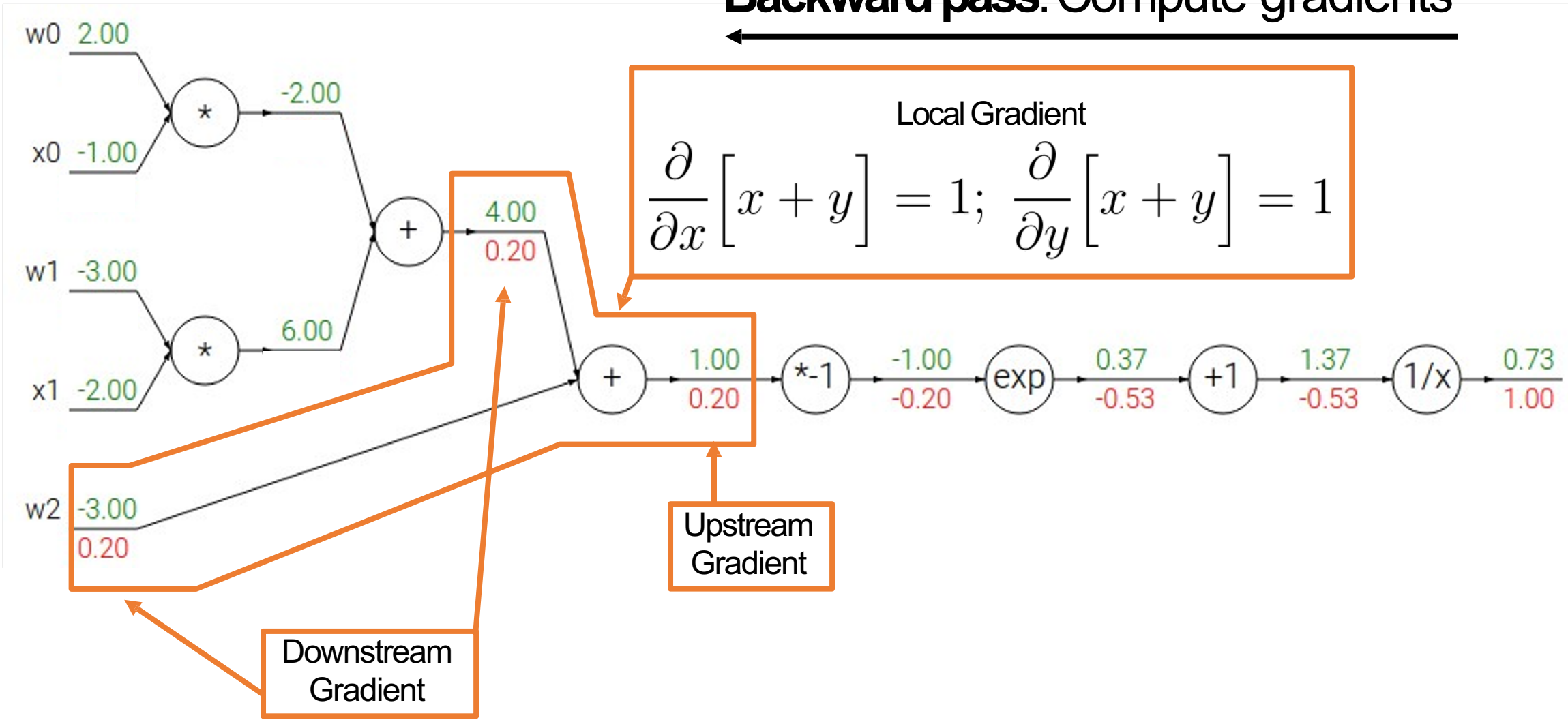
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



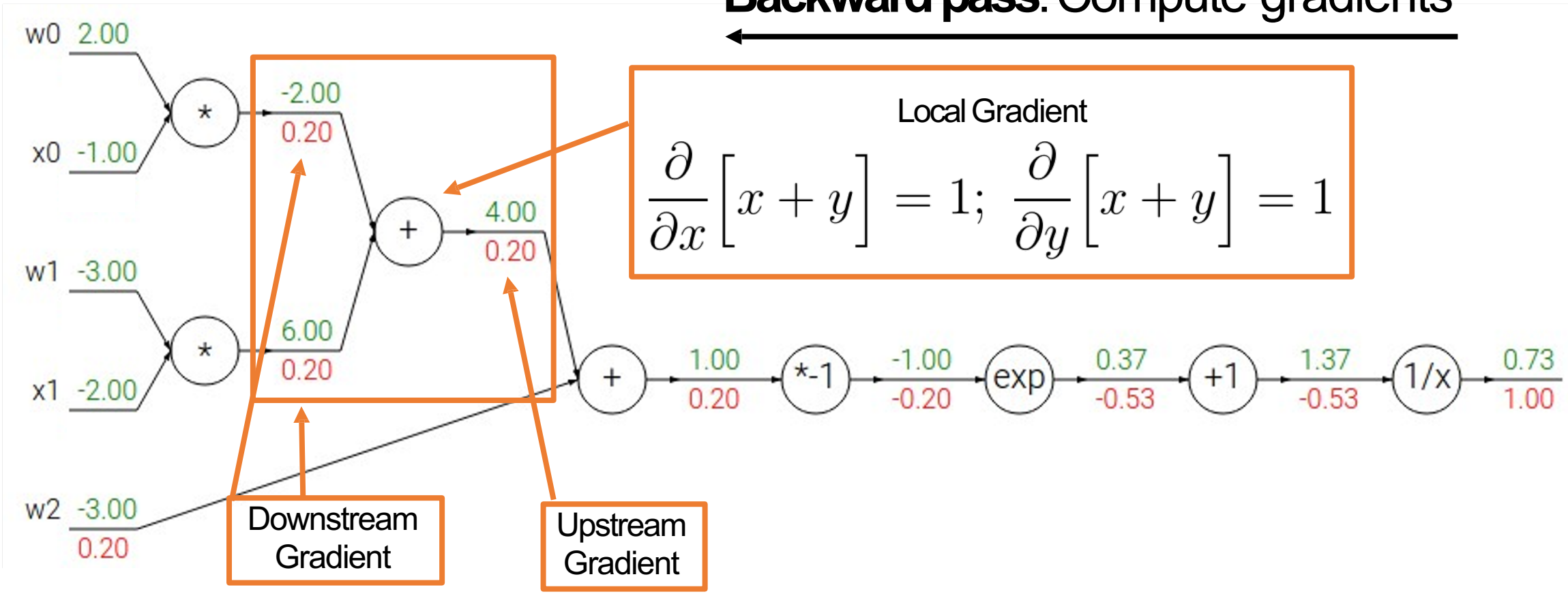
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



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

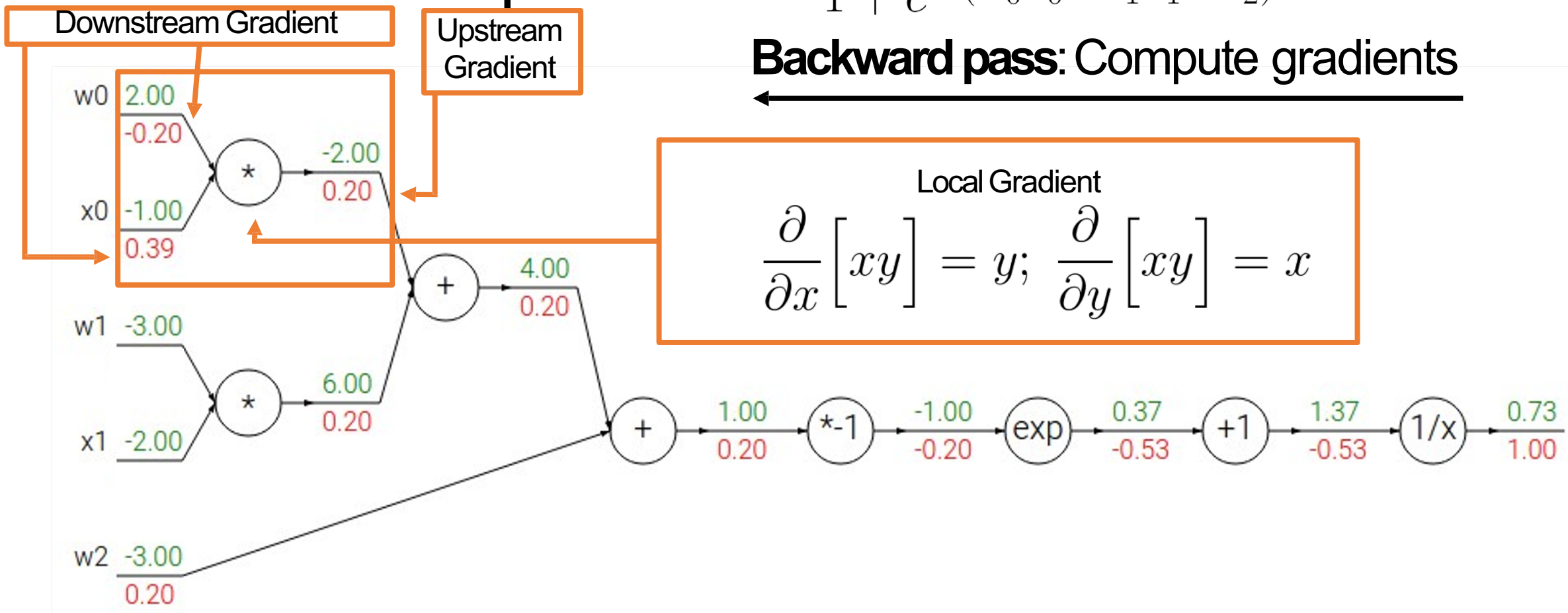
Backward pass: Compute gradients



Another Example

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

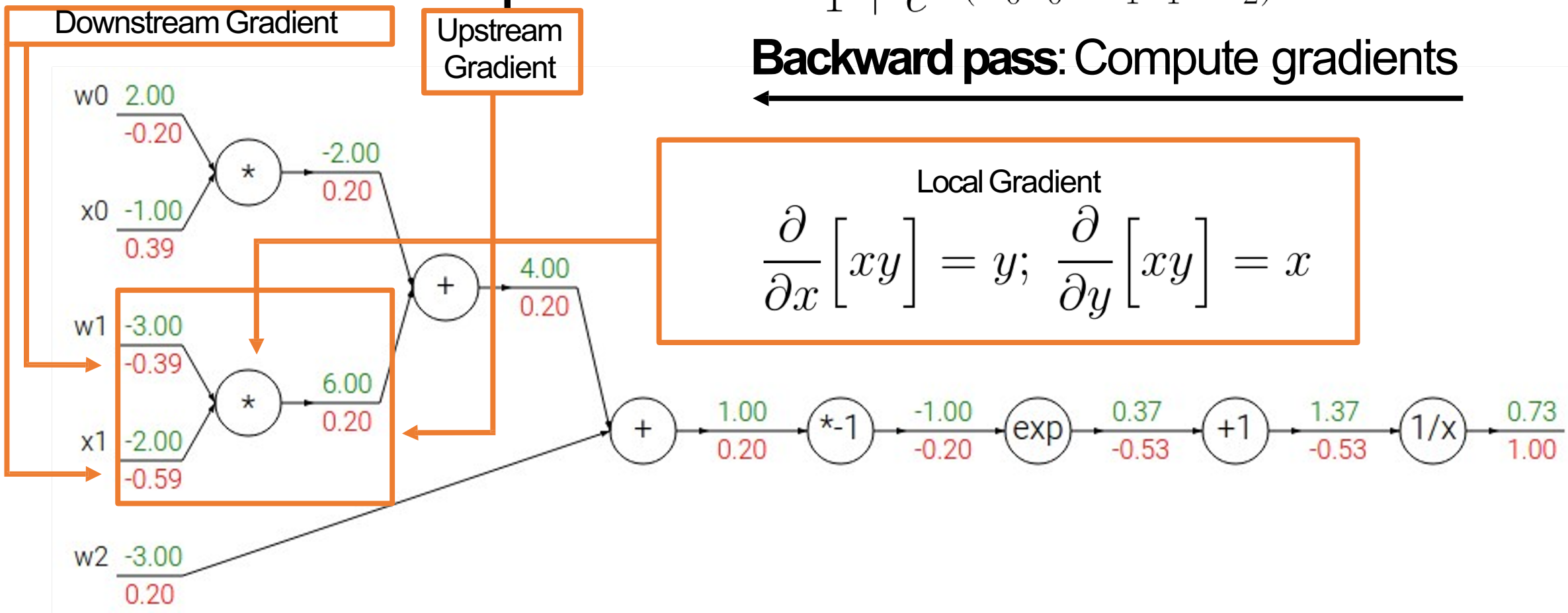
Backward pass: Compute gradients



Another Example

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

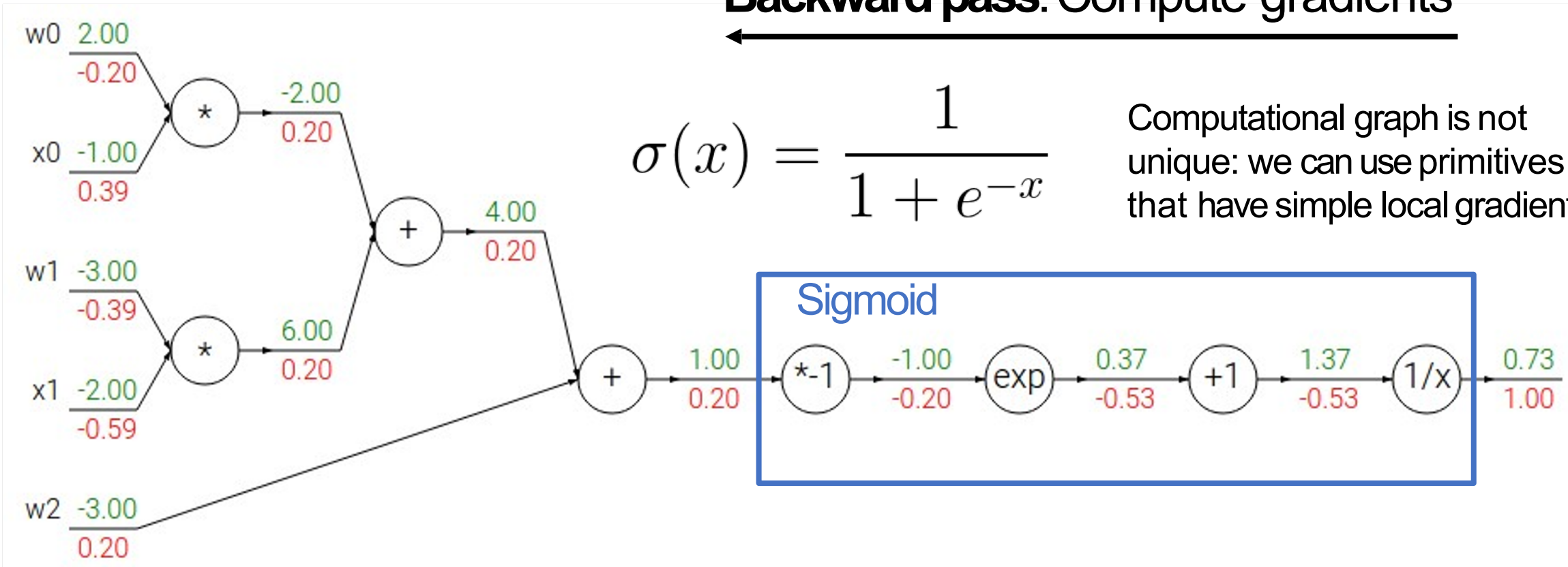
Backward pass: Compute gradients



Another Example

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

Backward pass: Compute gradients



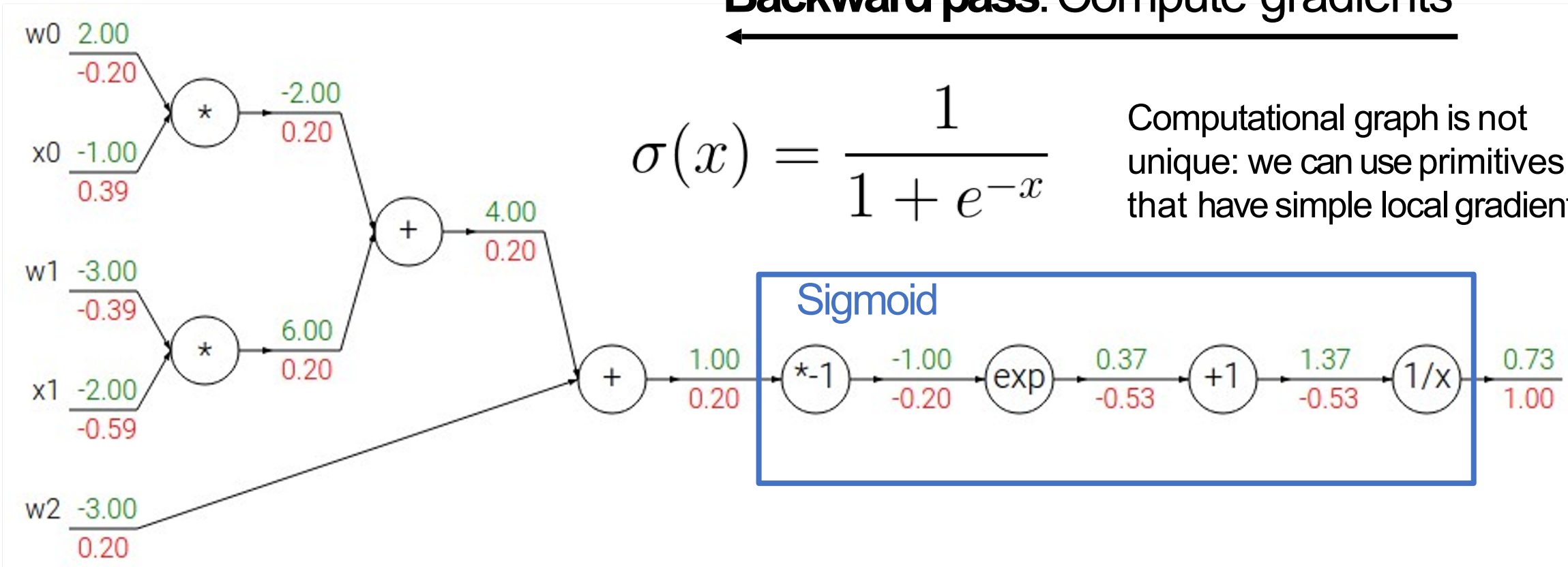
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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

Another Example

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

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

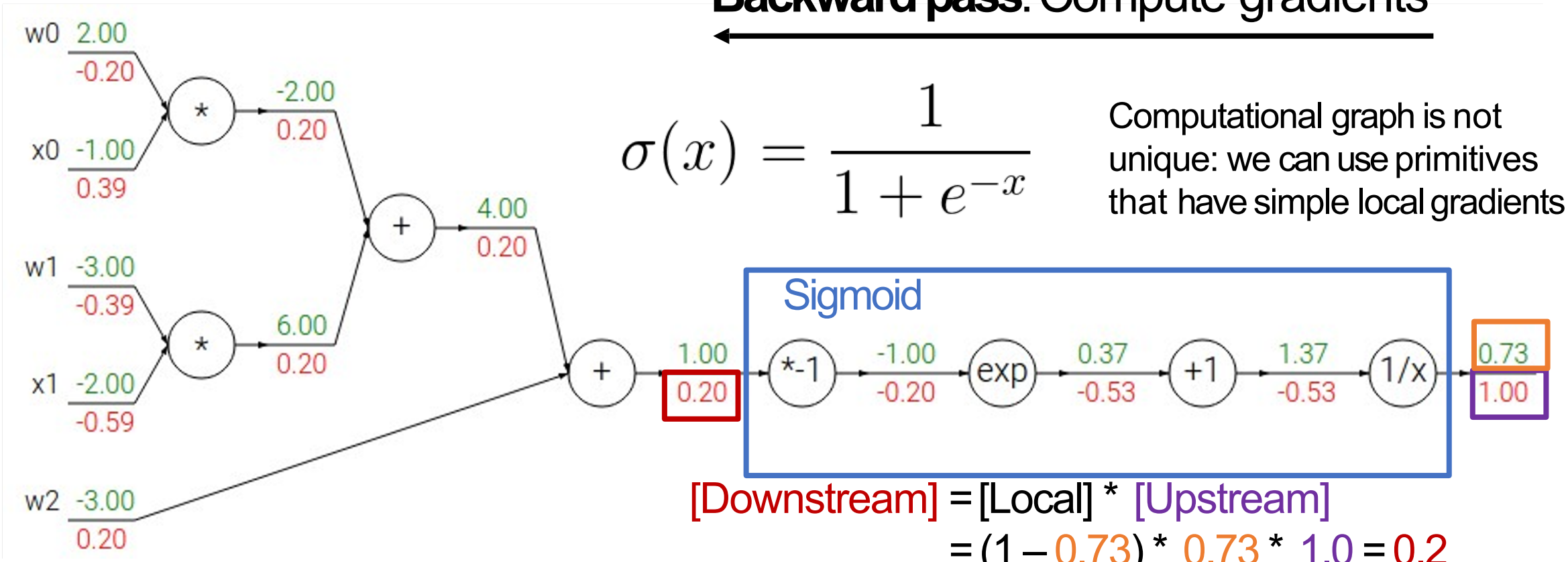
Sigmoid local gradient:

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

Another Example

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

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]

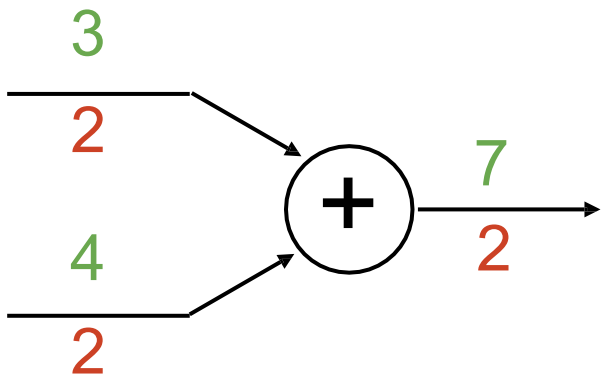
= (1 – 0.73) * 0.73 * 1.0 = 0.2

Sigmoid local gradient:

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

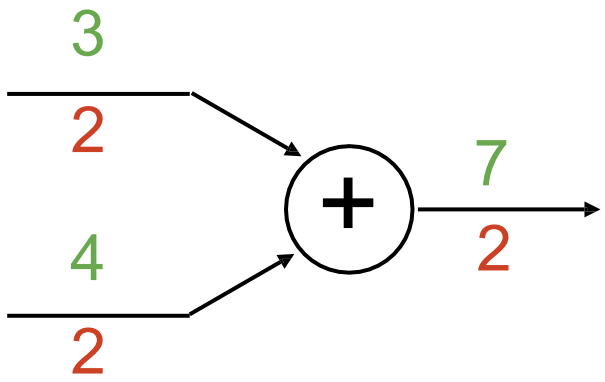
Patterns in Gradient Flow

add gate: gradient distributor

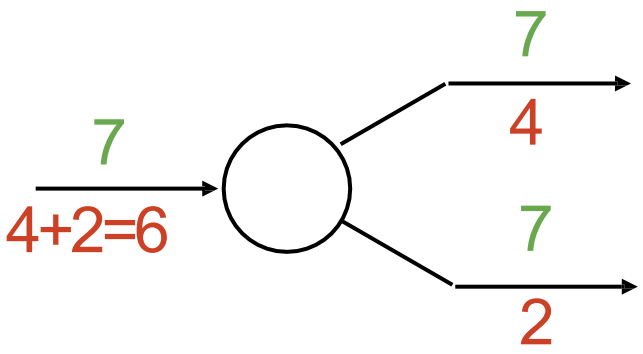


Patterns in Gradient Flow

add gate: gradient distributor

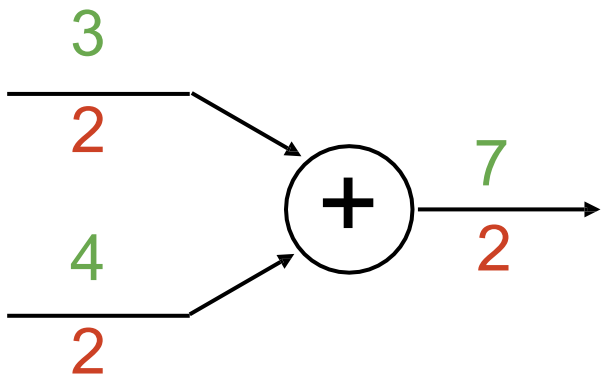


copy gate: gradient adder

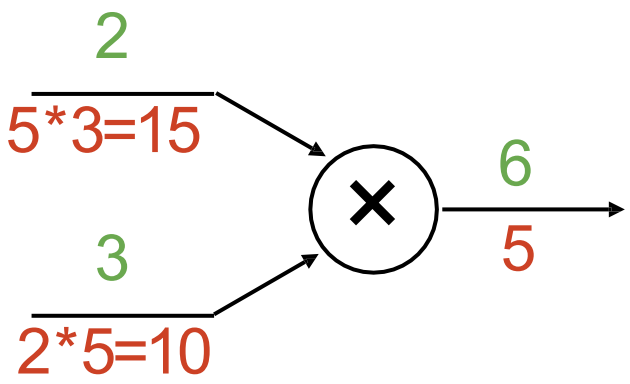


Patterns in Gradient Flow

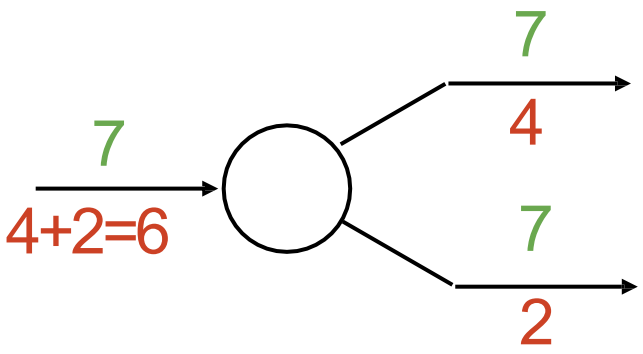
add gate: gradient distributor



mul gate: "swap multiplier"

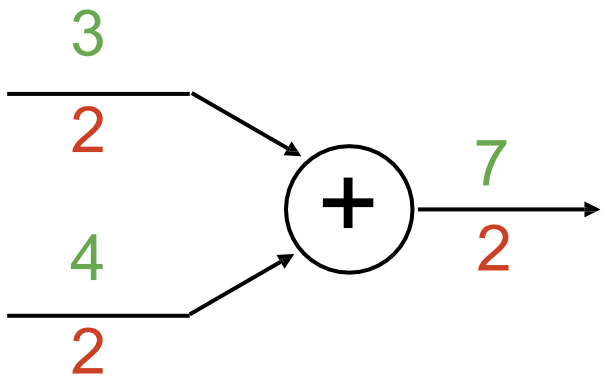


copy gate: gradient adder

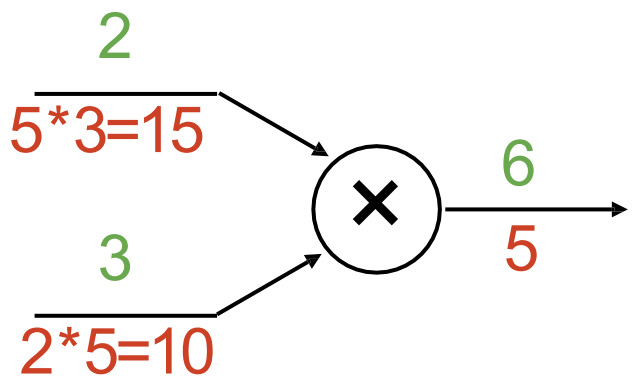


Patterns in Gradient Flow

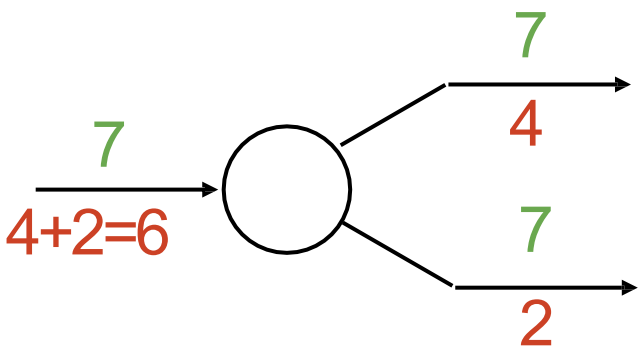
add gate: gradient distributor



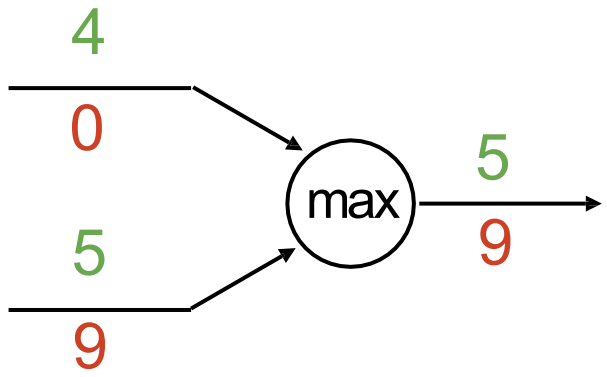
mul gate: "swap multiplier"



copy gate: gradient adder



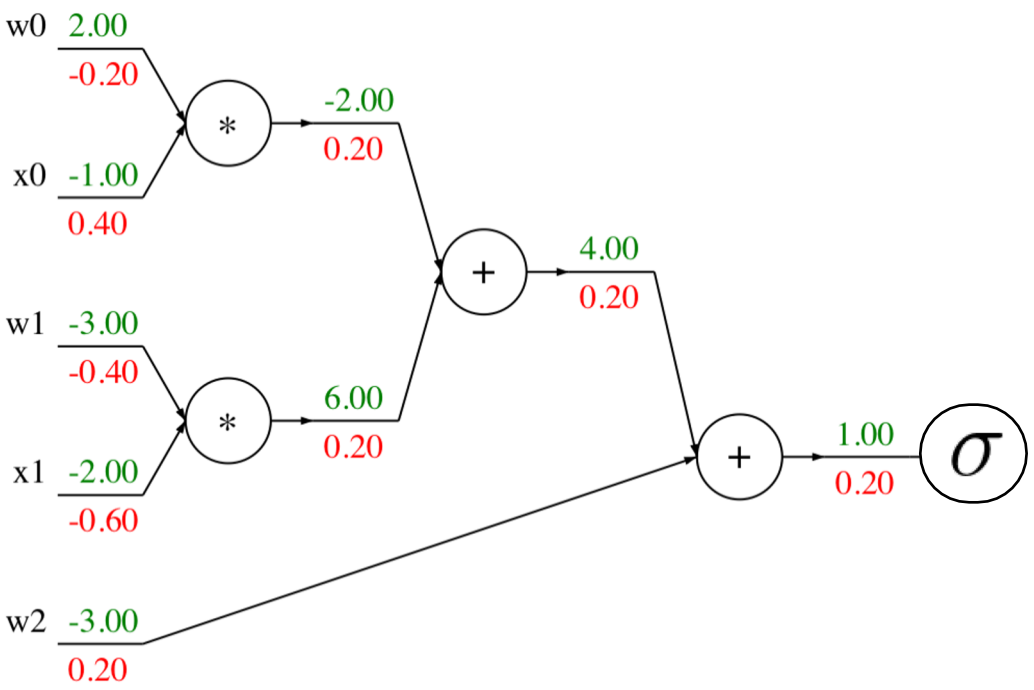
max gate: gradient router



Backprop Implementation: "Flat" gradient code:

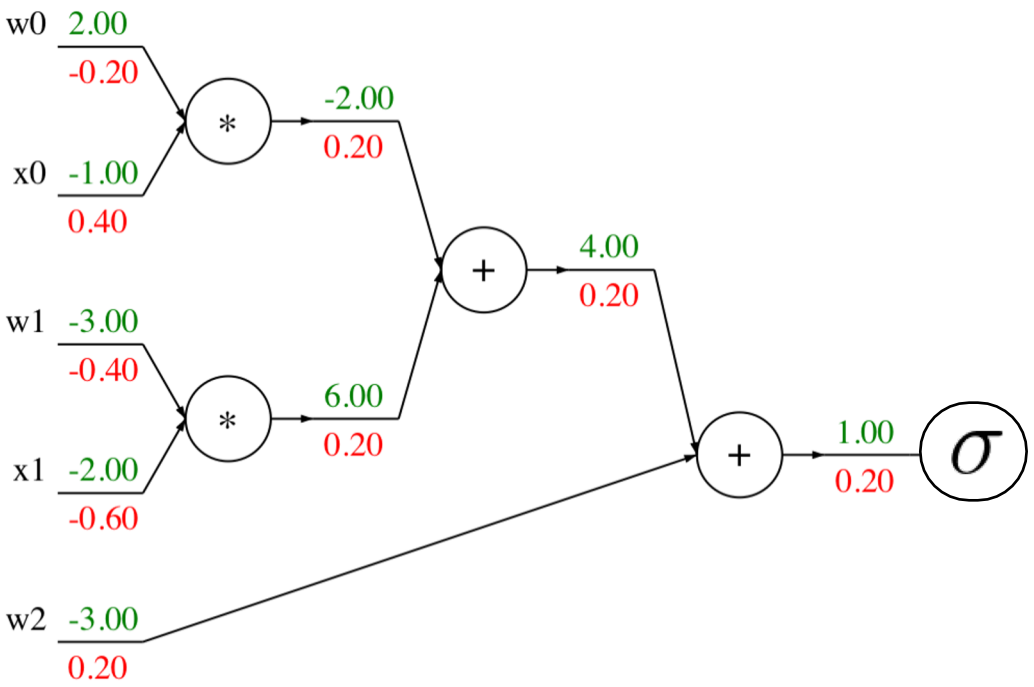
Forward pass: Compute output

```
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

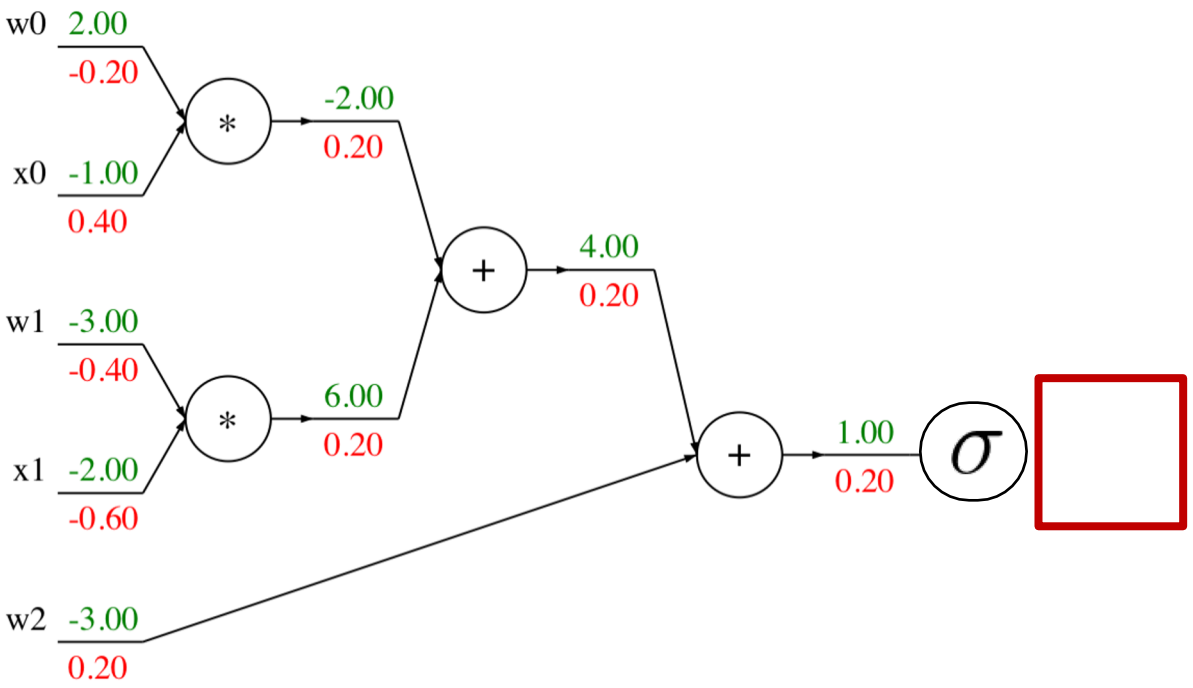



```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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

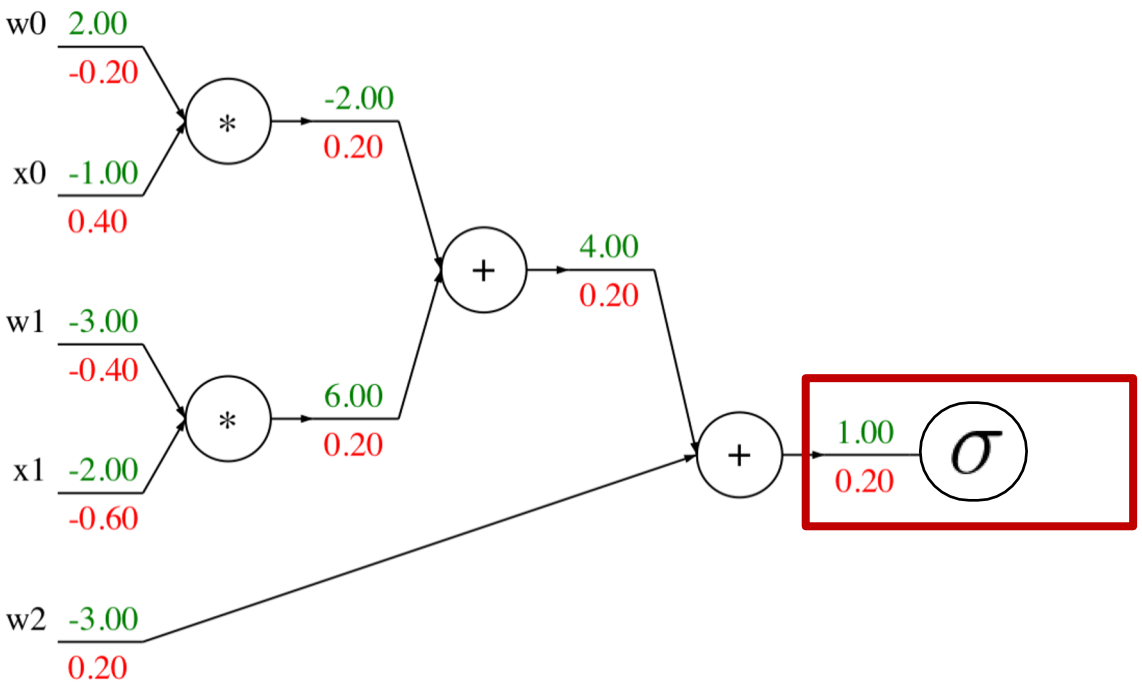
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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

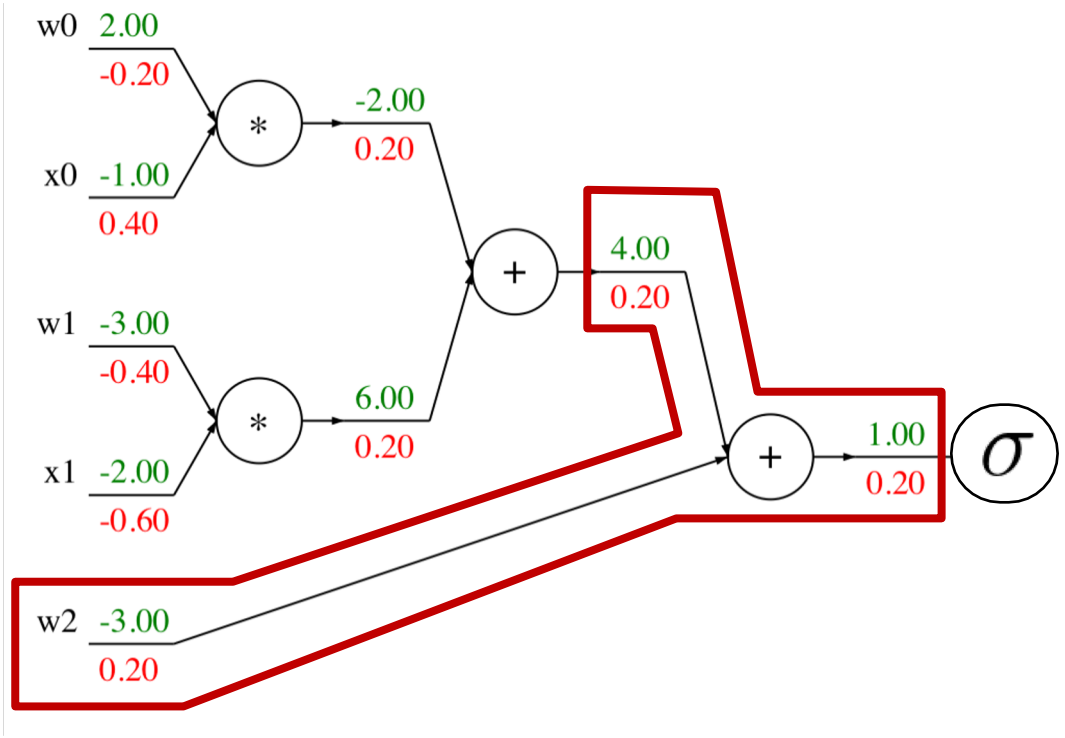



```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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

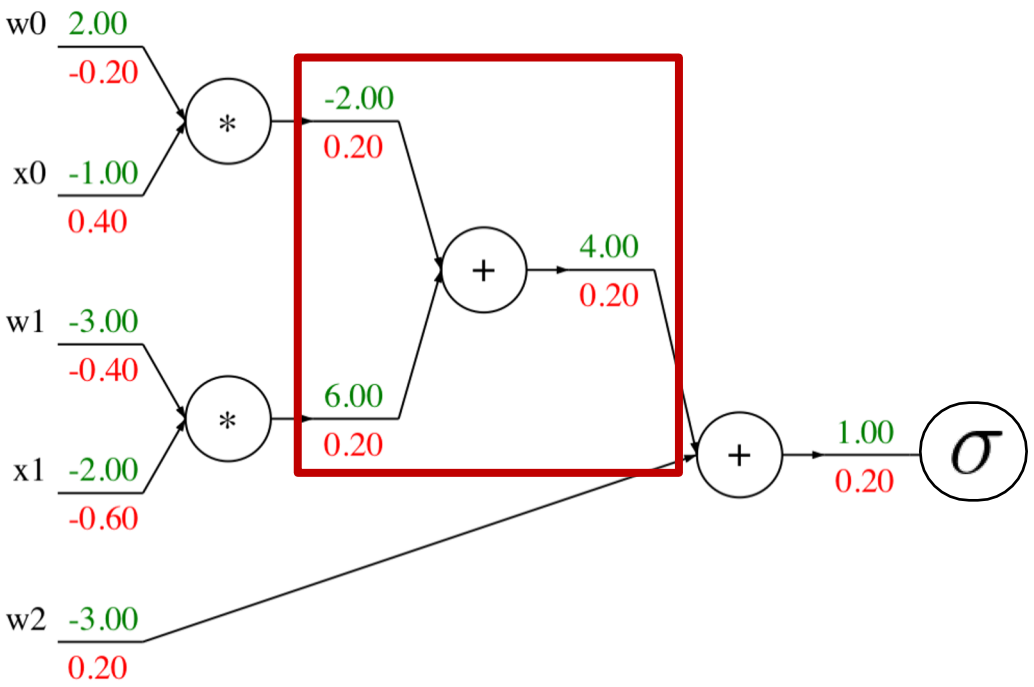
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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



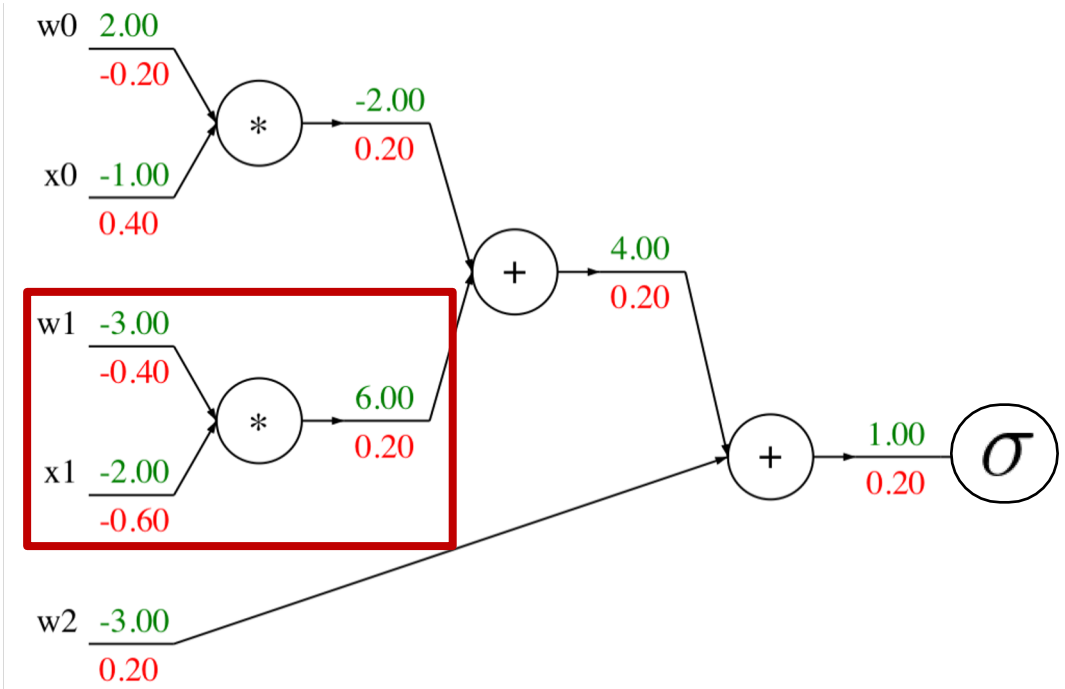

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

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

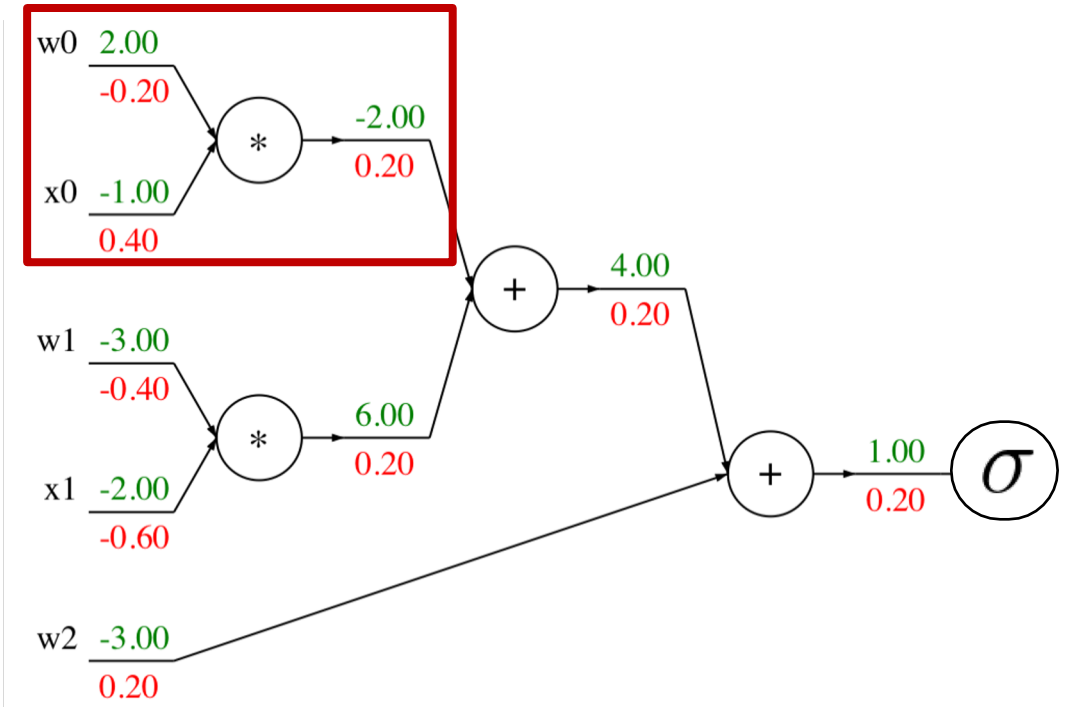



```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  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:



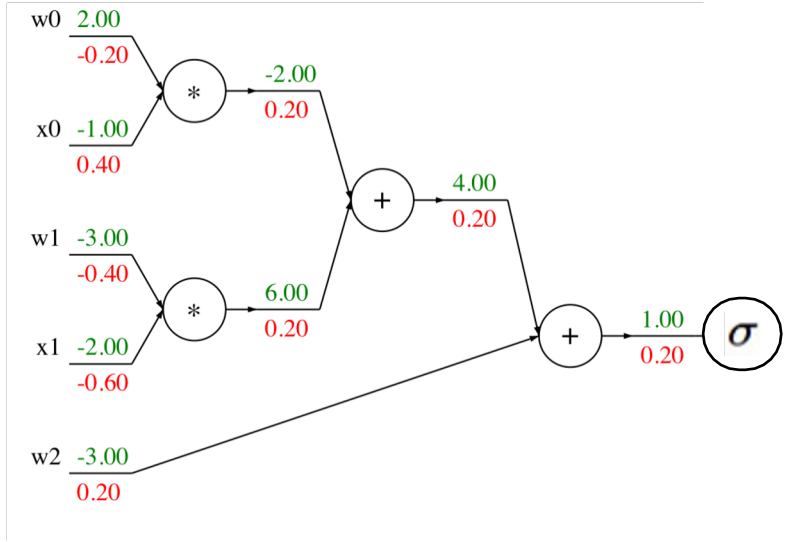
Forward pass: Compute output

```
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

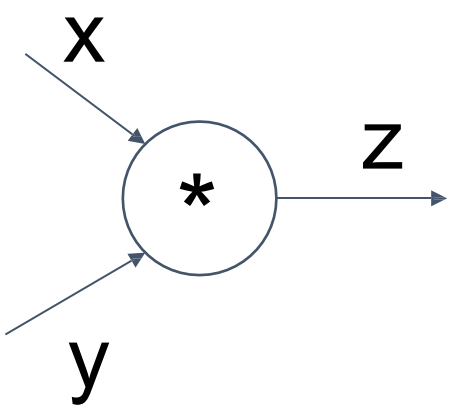
Backprop Implementation: Modular API



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

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
```

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

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

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;
        });
  });
}
```

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

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

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

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

Backpropagation with scalars so far

What about vector-valued functions?

Review: 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?

Review: Vector Derivatives

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

Regular derivative:

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

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

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

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

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

Review: Vector Derivatives

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

Regular derivative:

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

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

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

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

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

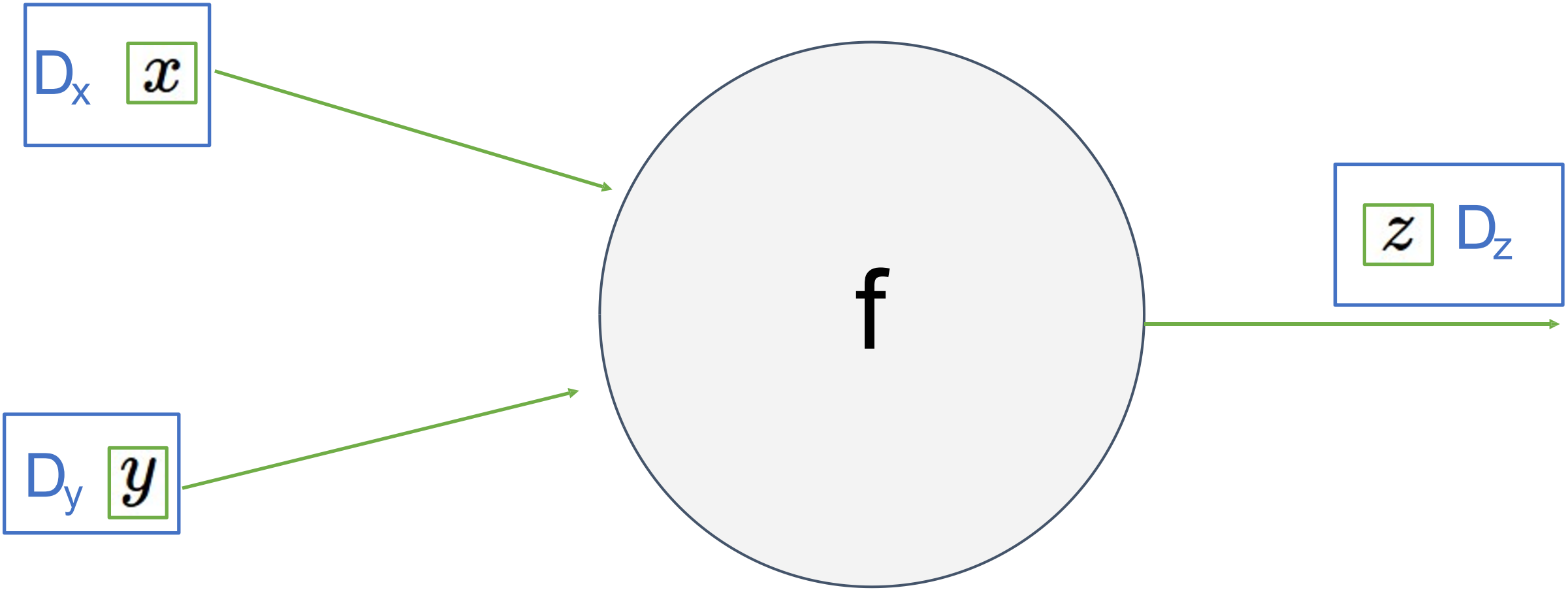
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

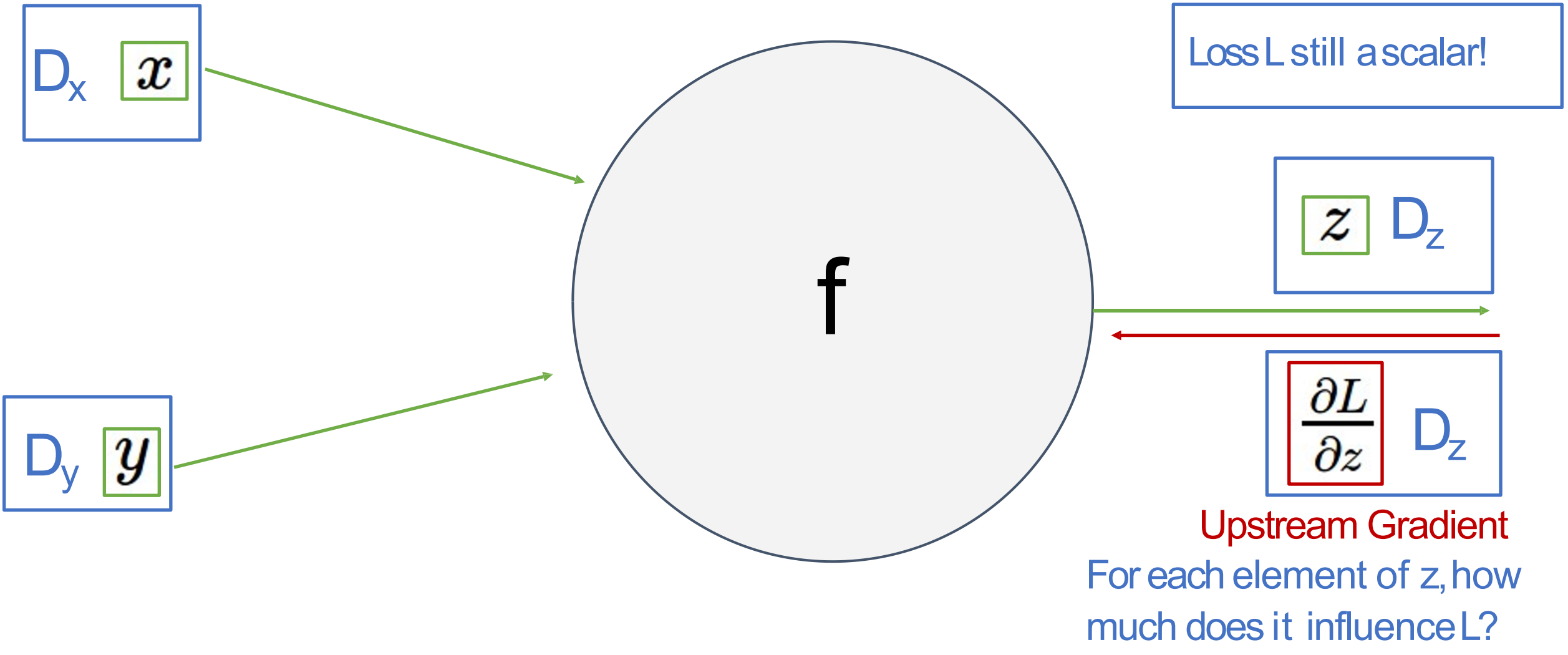
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

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

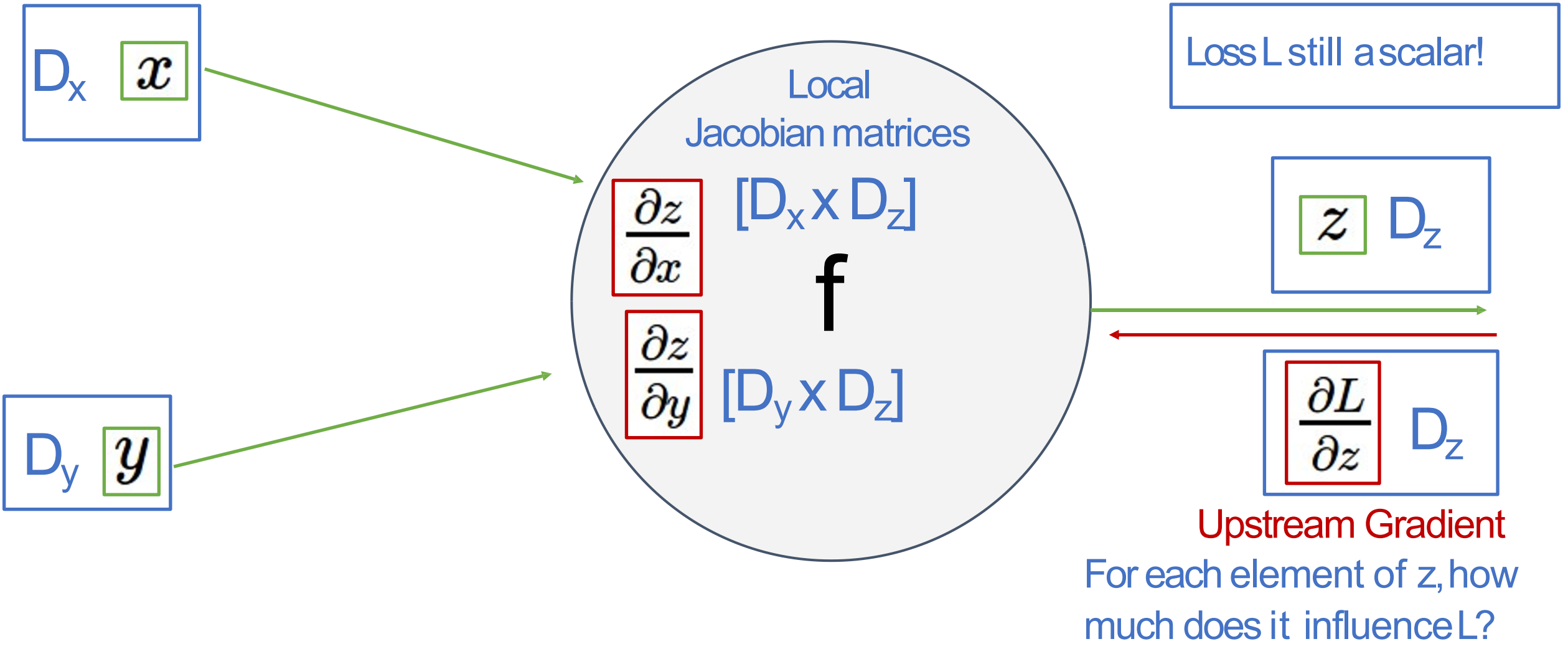
Backprop with Vectors



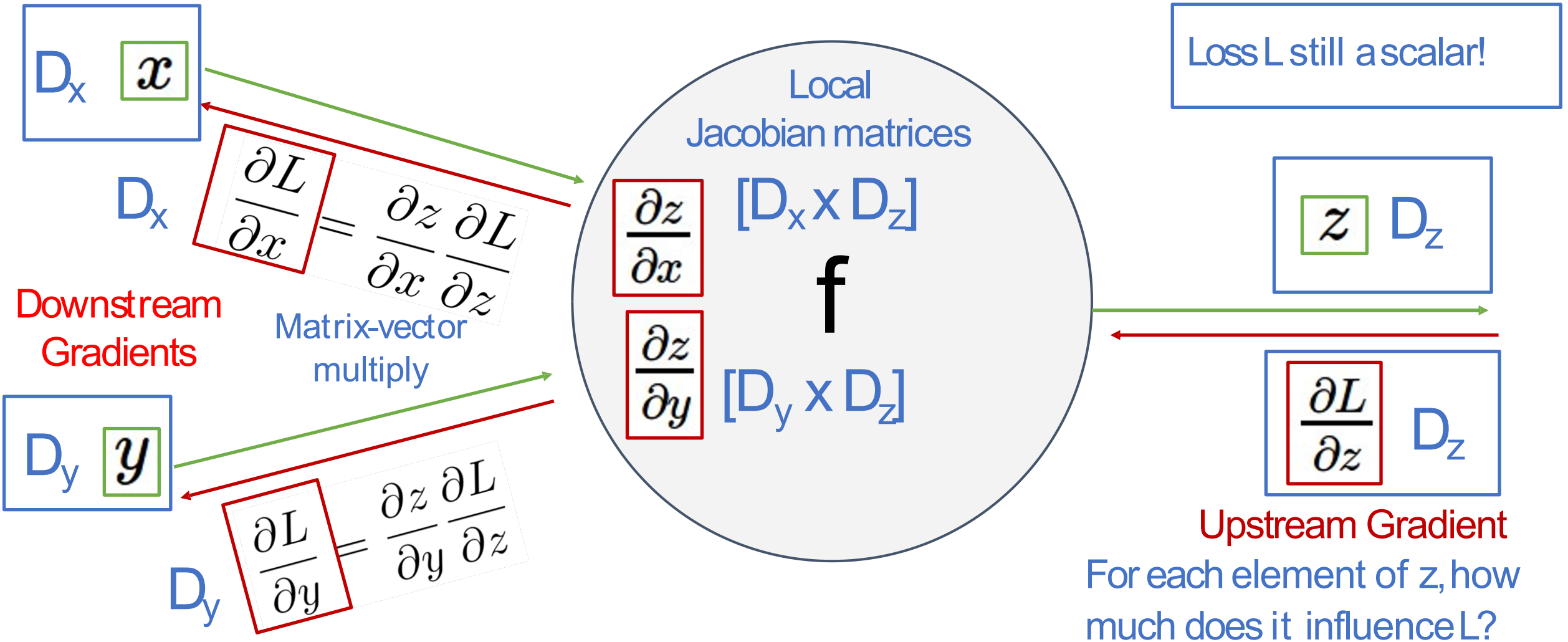
Backprop with Vectors



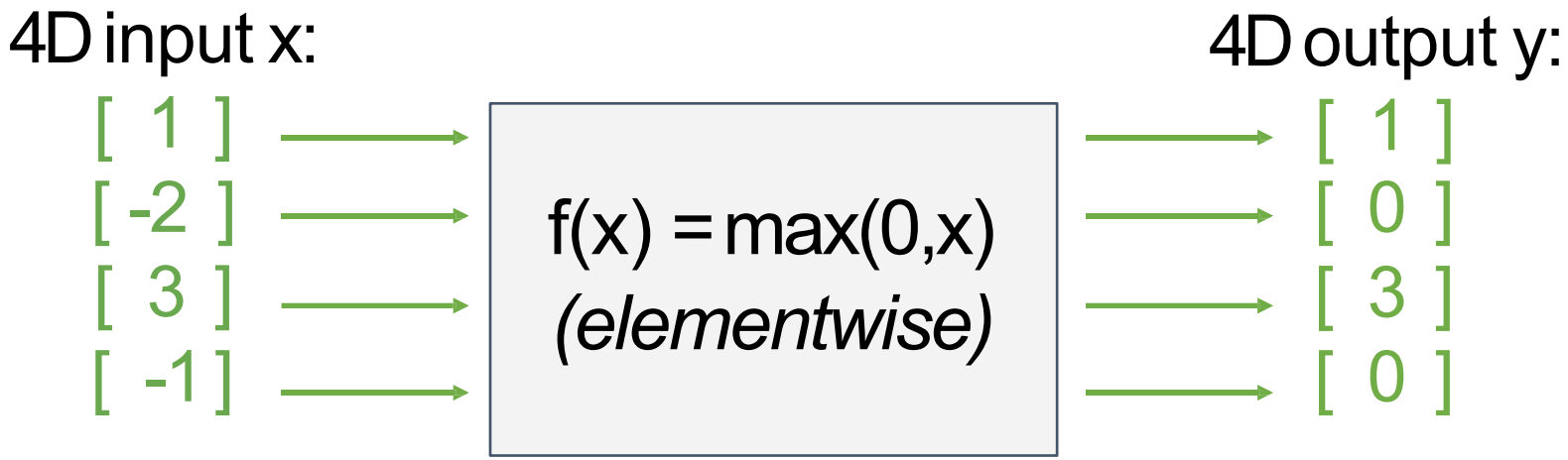
Backprop with Vectors



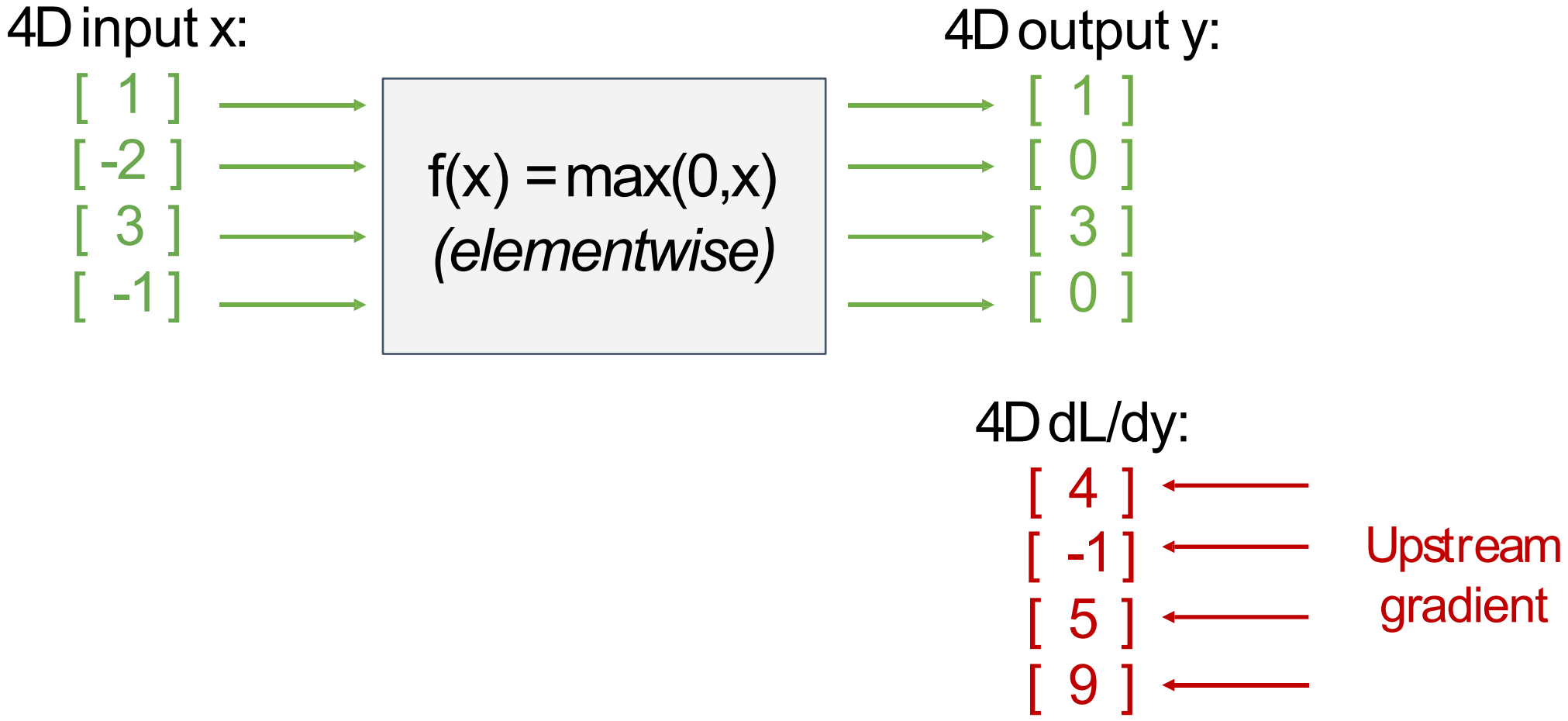
Backprop with Vectors



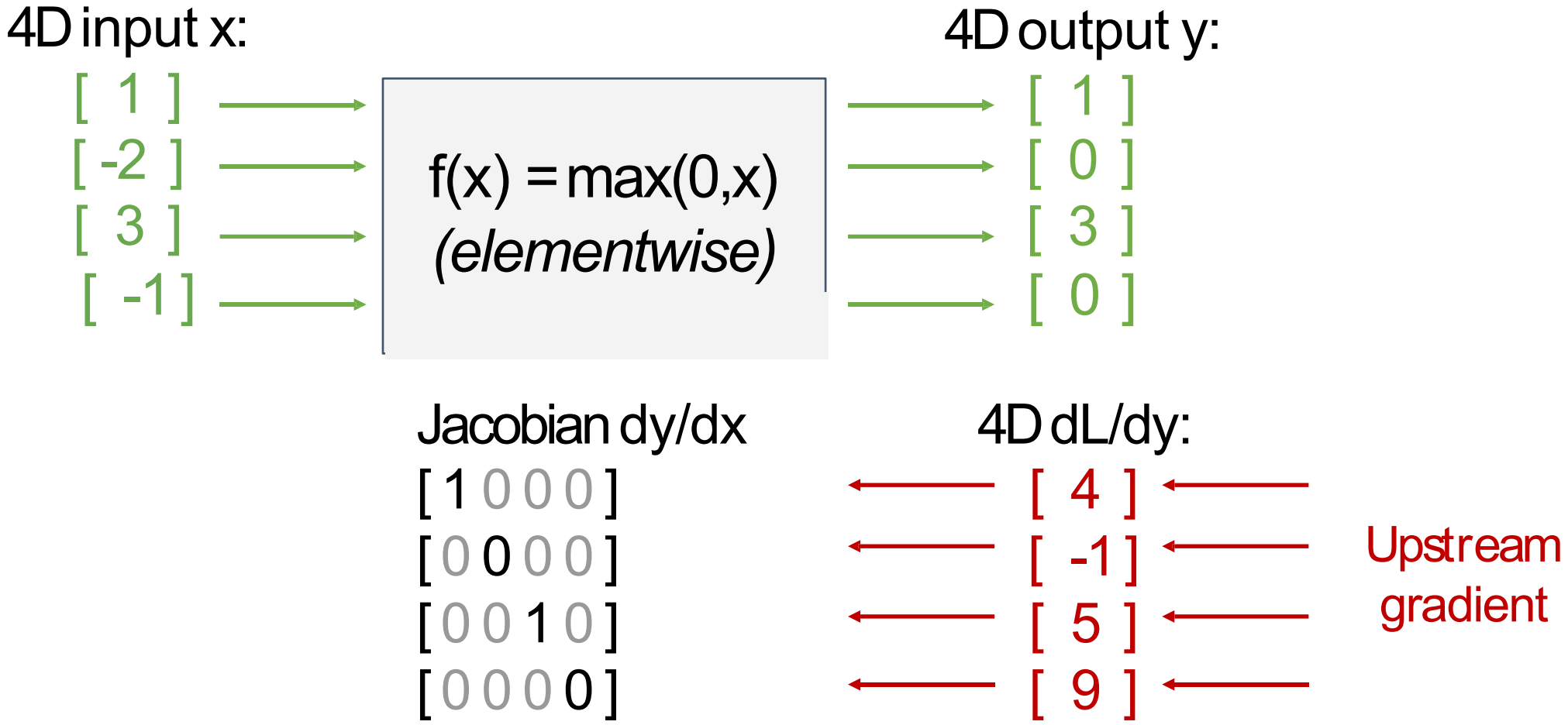
Backprop with Vectors



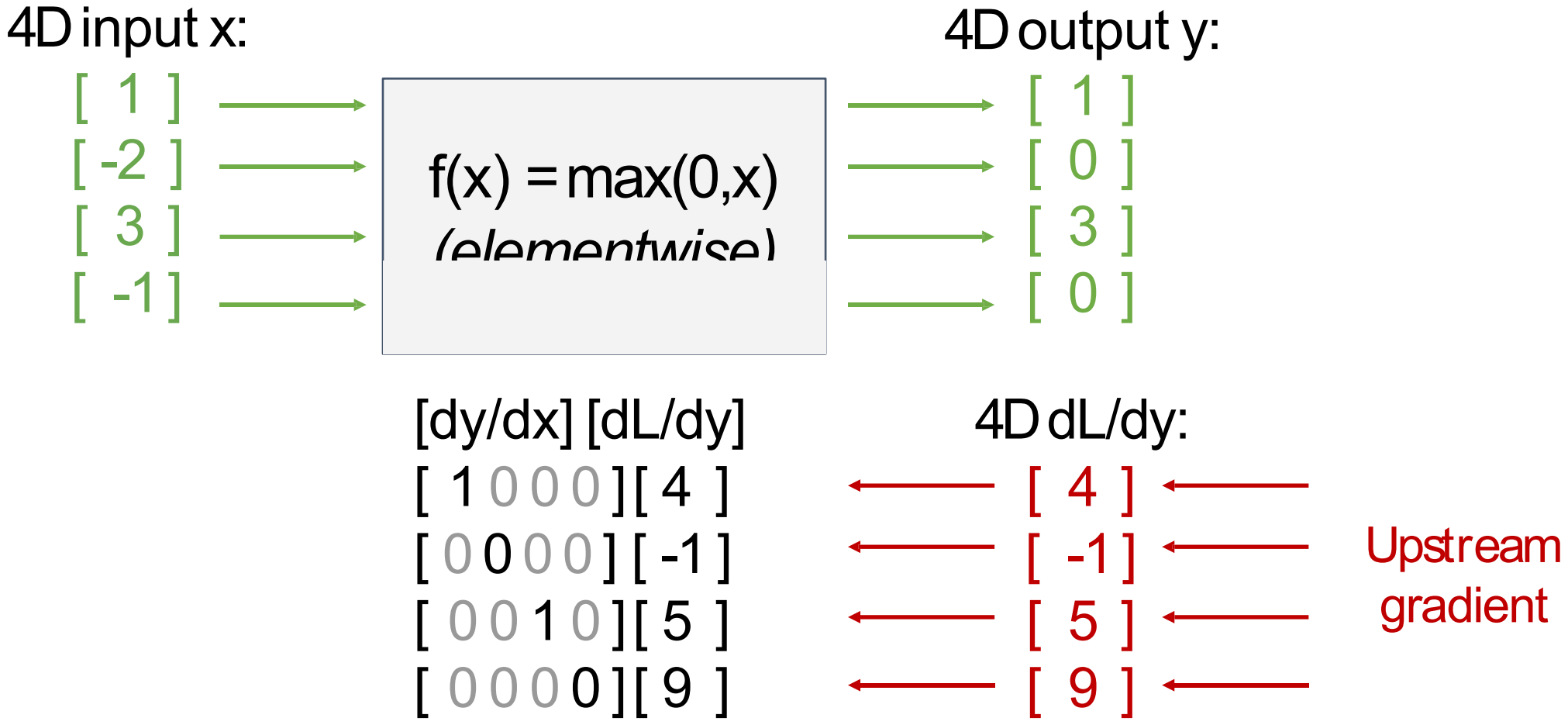
Backprop with Vectors



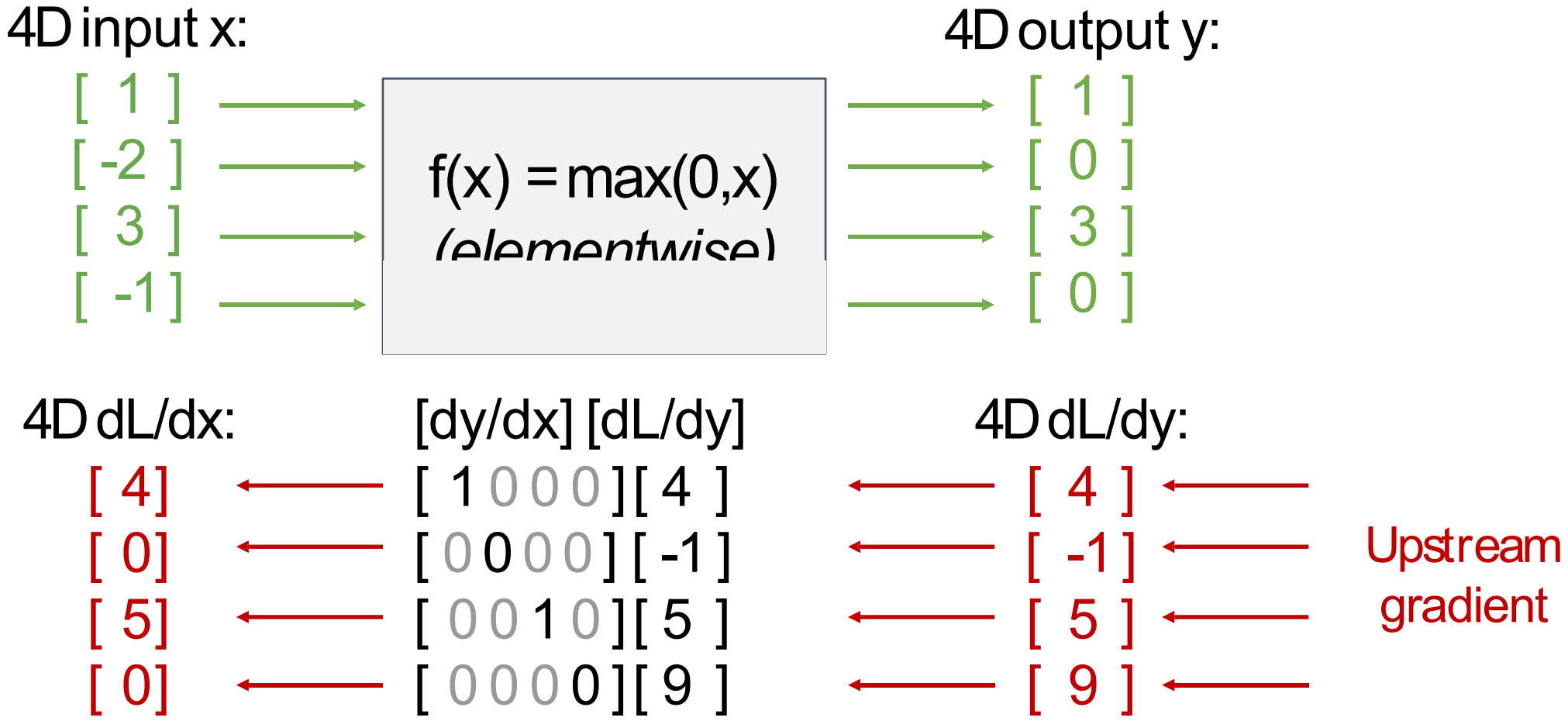
Backprop with Vectors



Backprop with Vectors

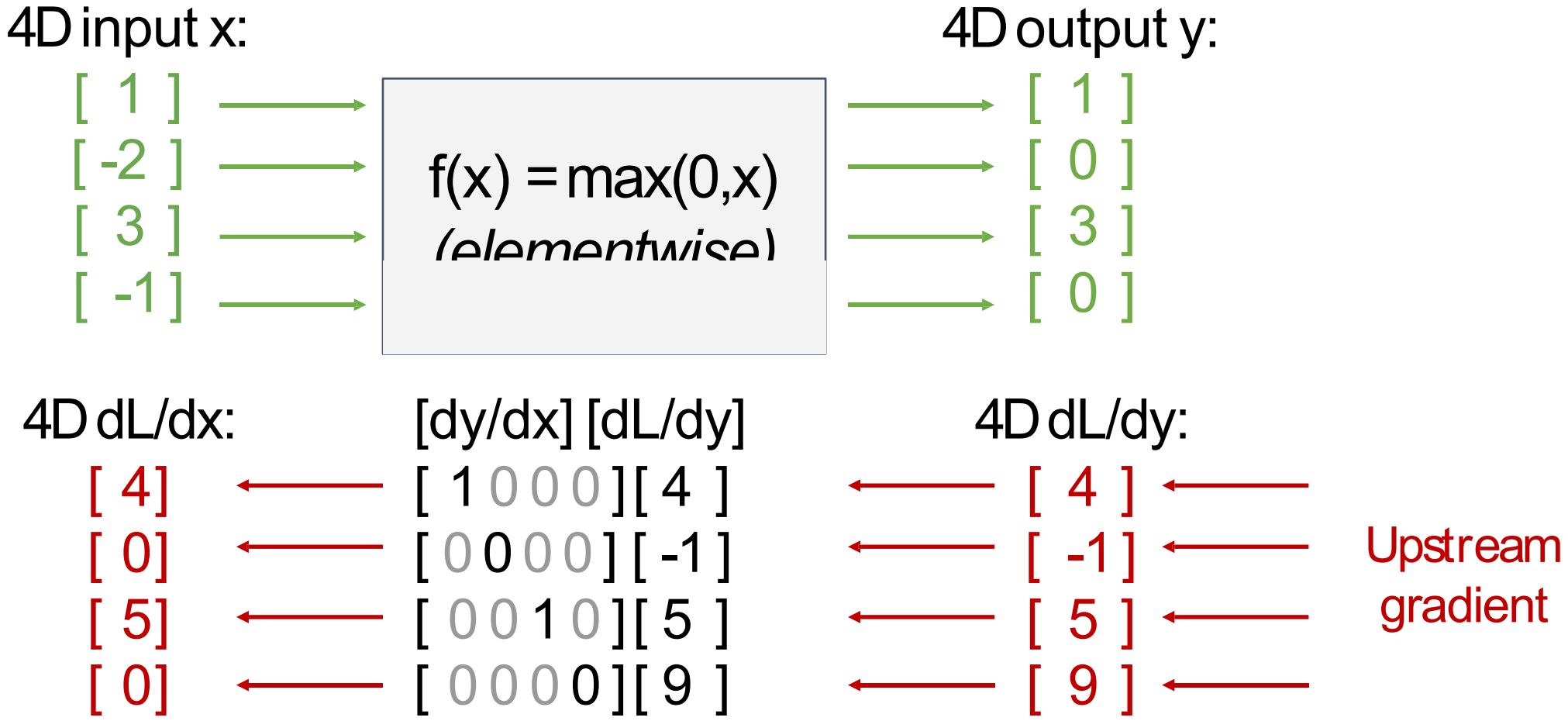


Backprop with Vectors



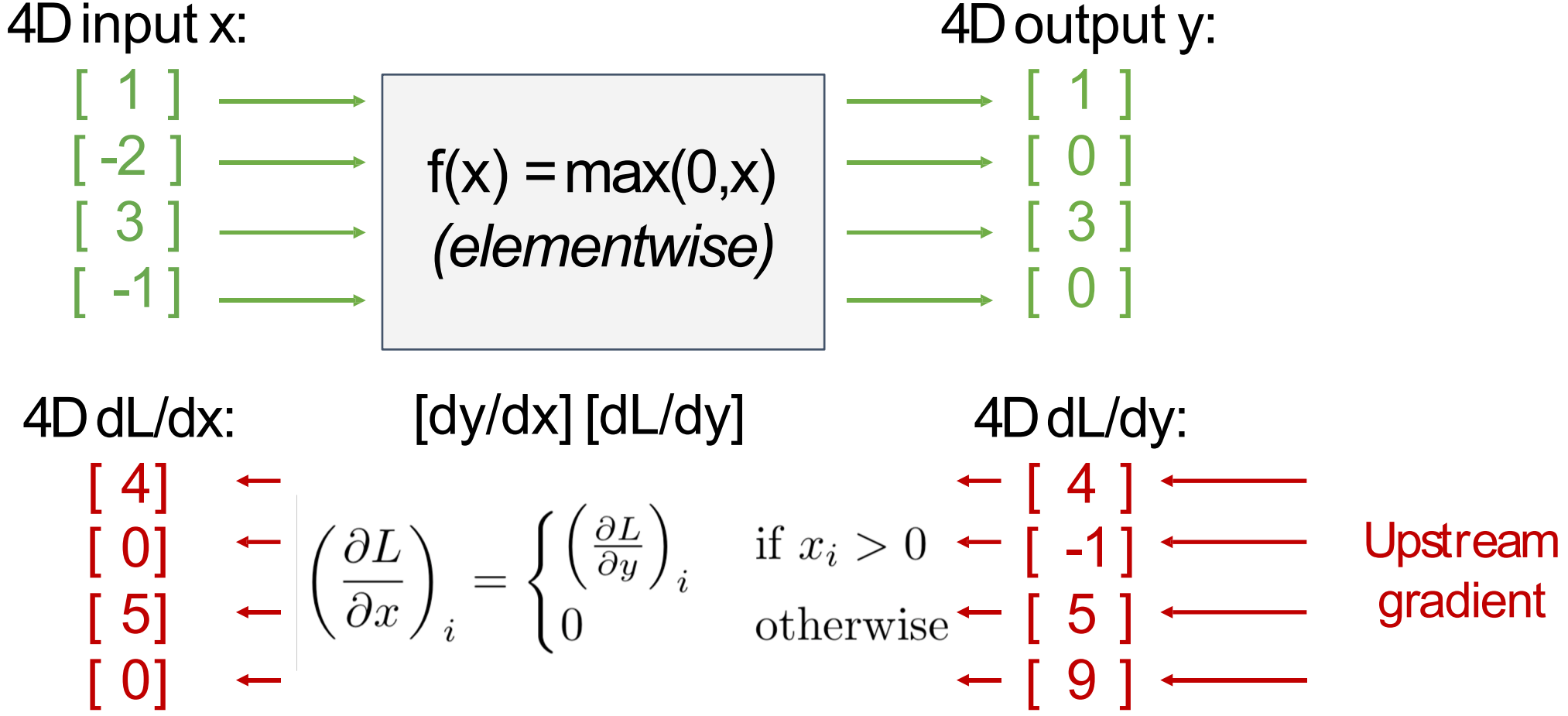
Backprop with Vectors

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

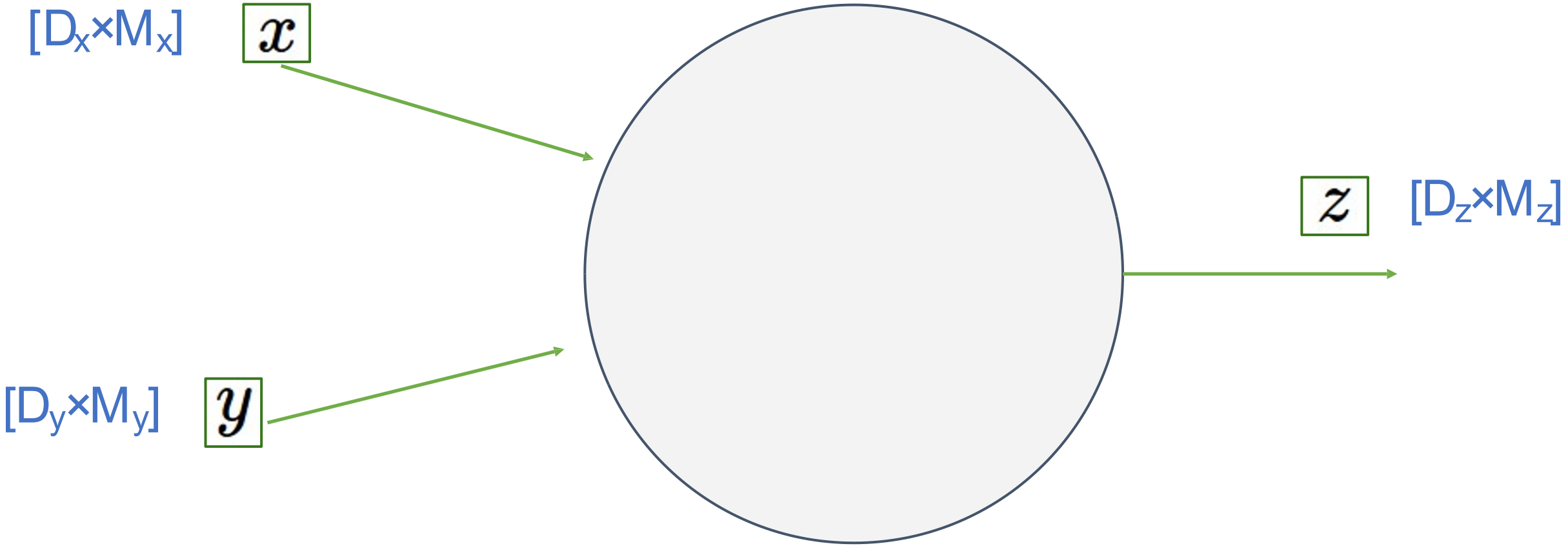


Backprop with Vectors

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



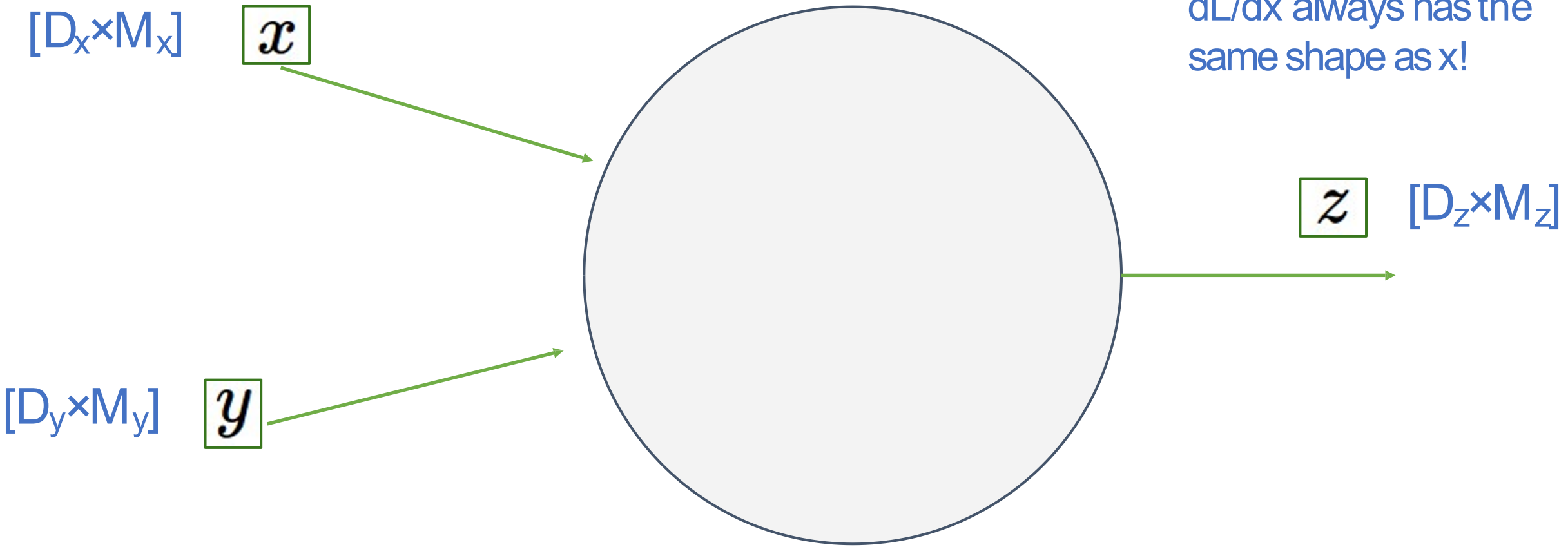
Backprop with Matrices (or Tensors):



Backprop with Matrices (or Tensors):

Loss L still a scalar!

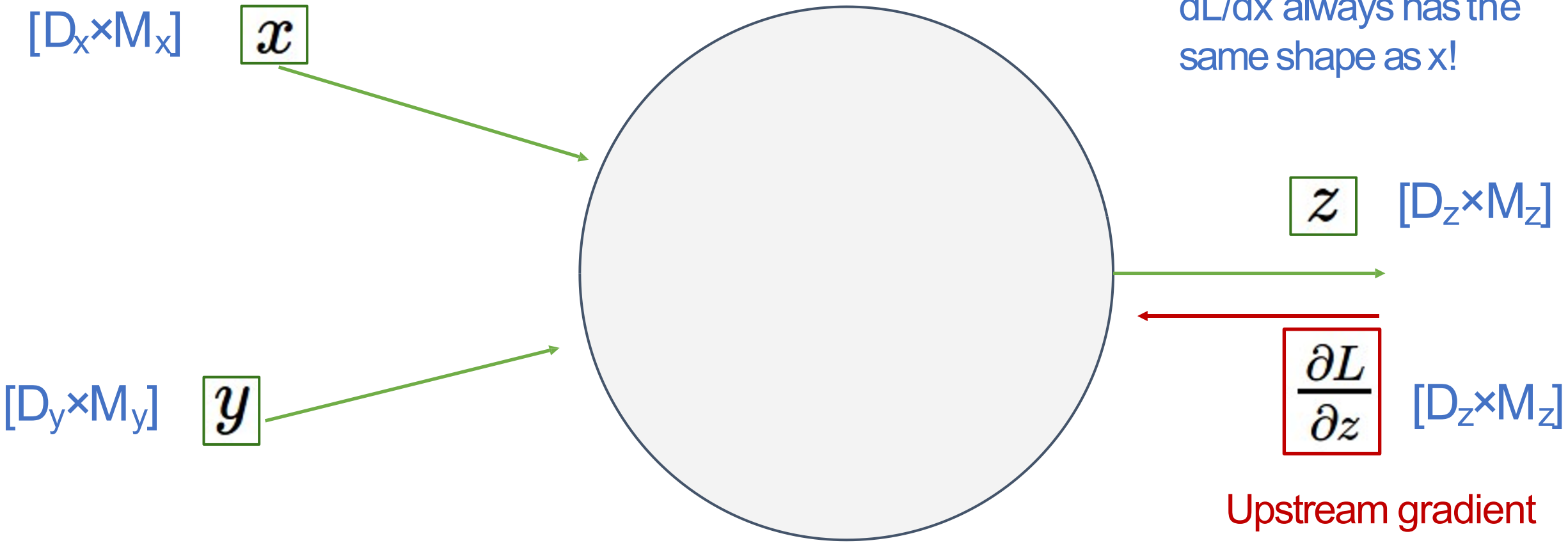
dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors):

Loss L still a scalar!

dL/dx always has the same shape as x!



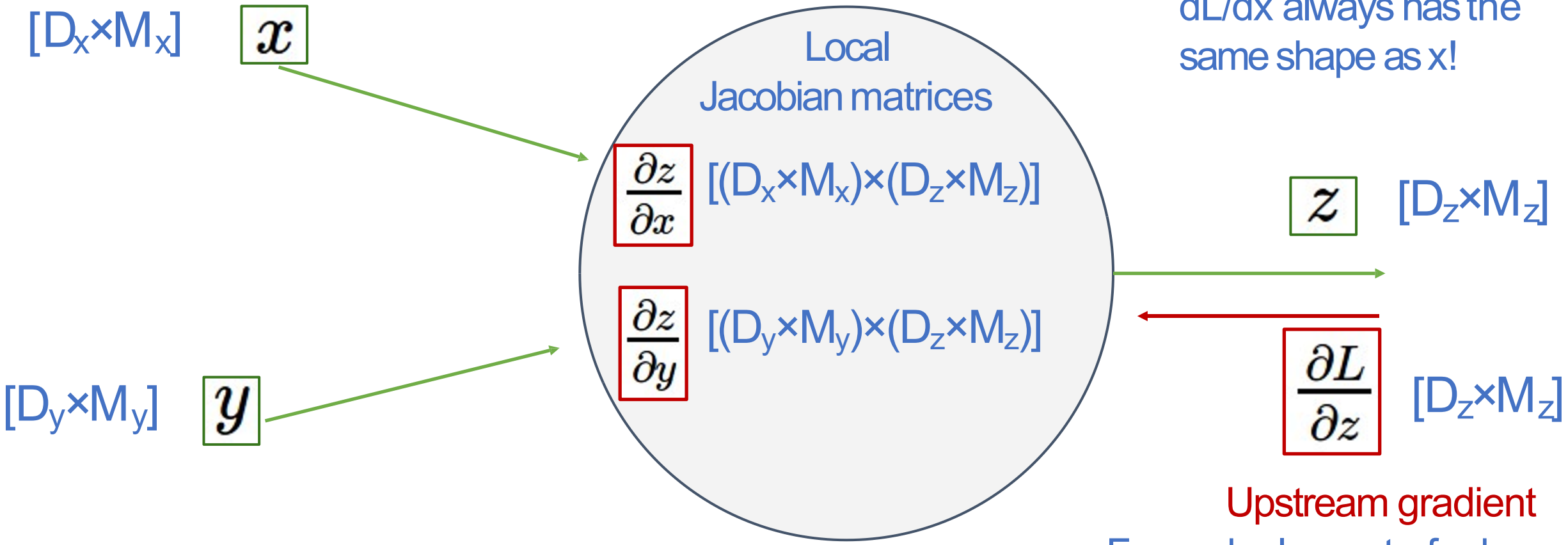
Upstream gradient

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?

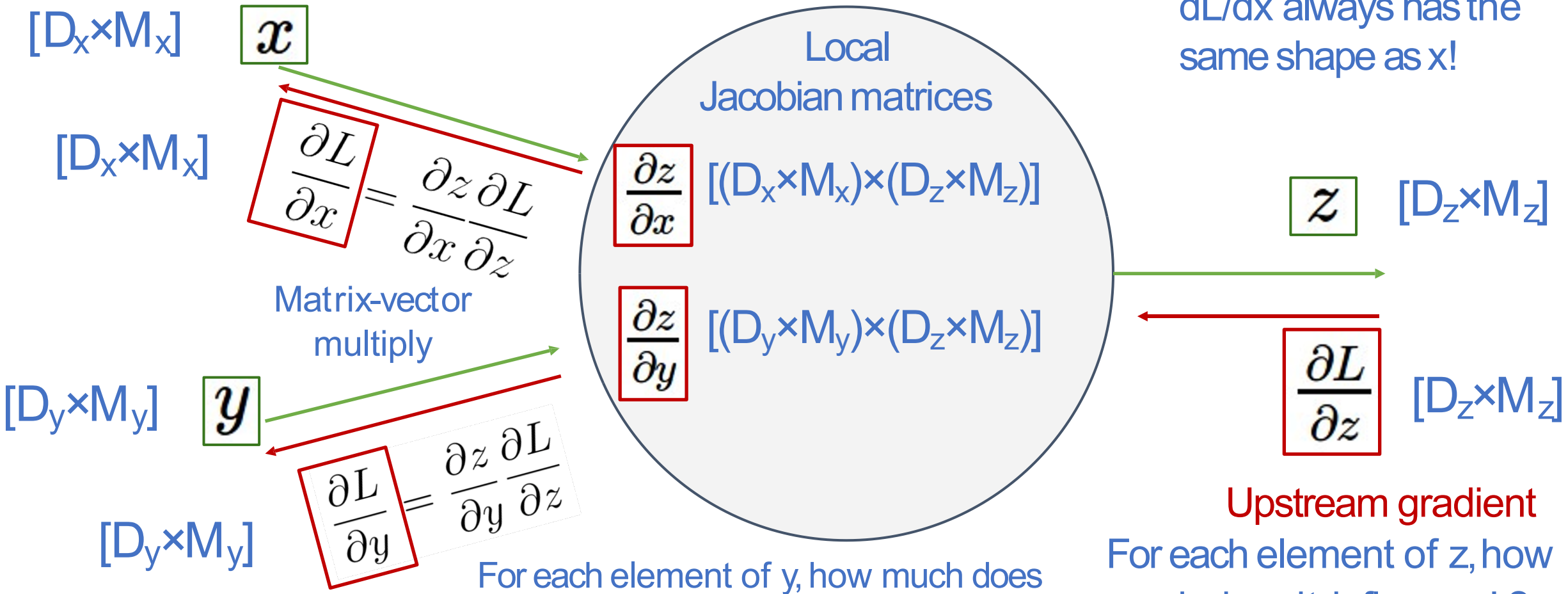
Upstream gradient

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?

Upstream gradient

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]

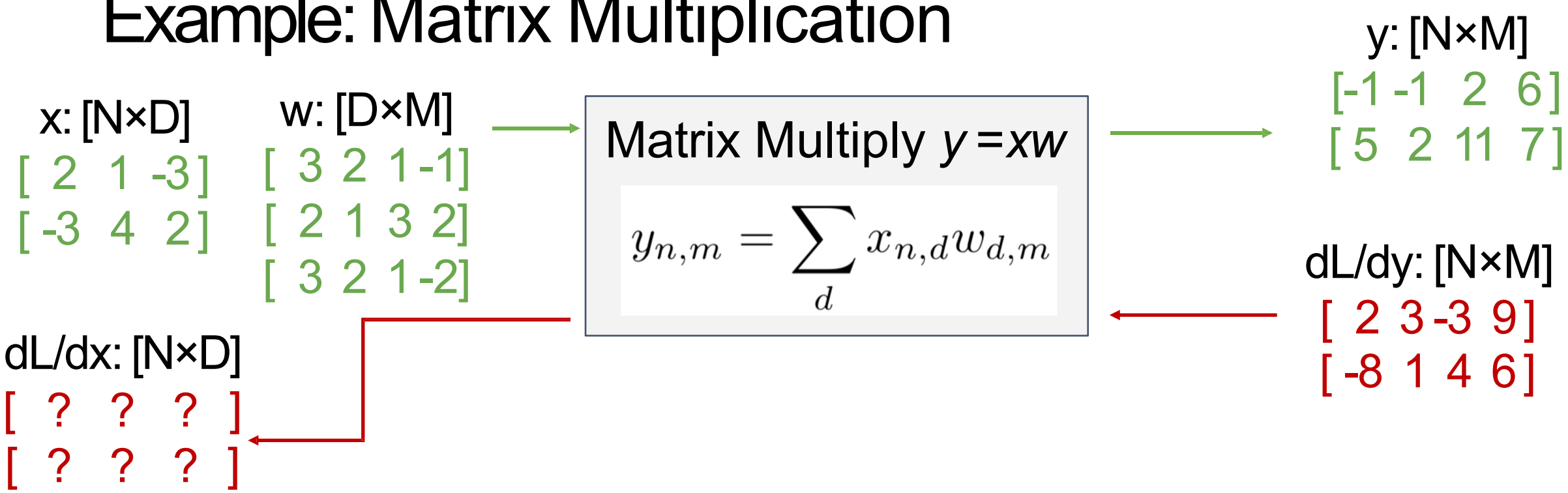
Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

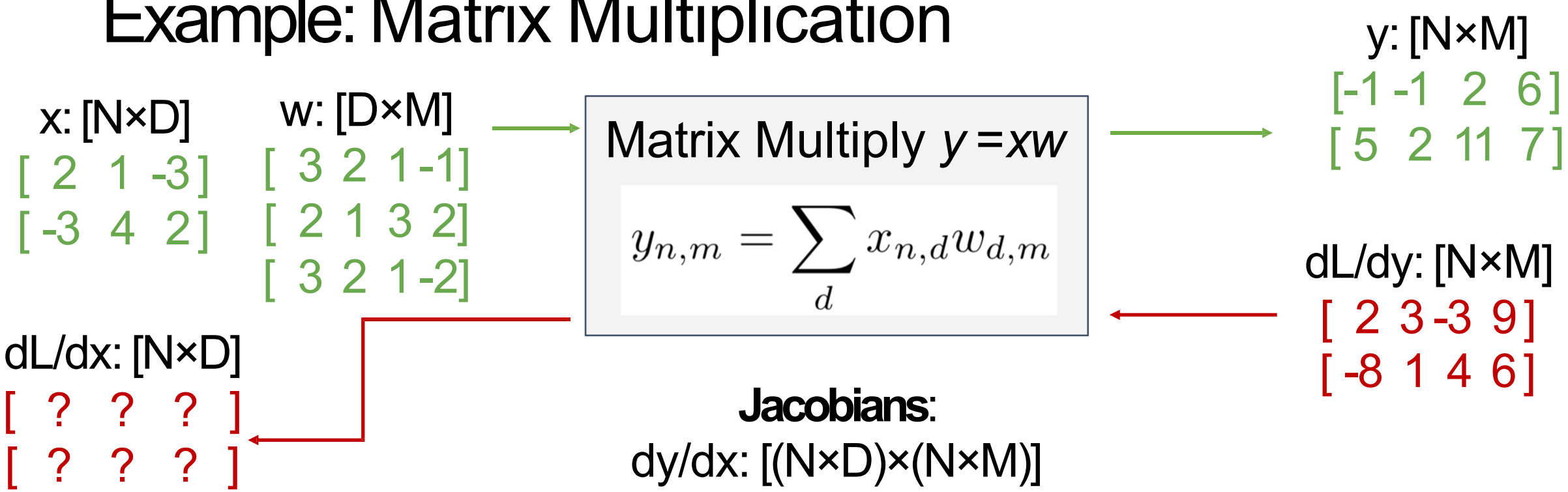
y: [N×M]

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

Example: Matrix Multiplication



Example: Matrix Multiplication



Jacobians:

dy/dx: [(N×D)×(N×M)]

dy/dw: [(D×M)×(N×M)]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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:

$dy/dx_{1,1}$

[? ? ? ?]
[? ? ? ?]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

[? ? ?]
[? ? ?]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

[? ? ?]
[? ? ?]

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

Local Gradient Slice:

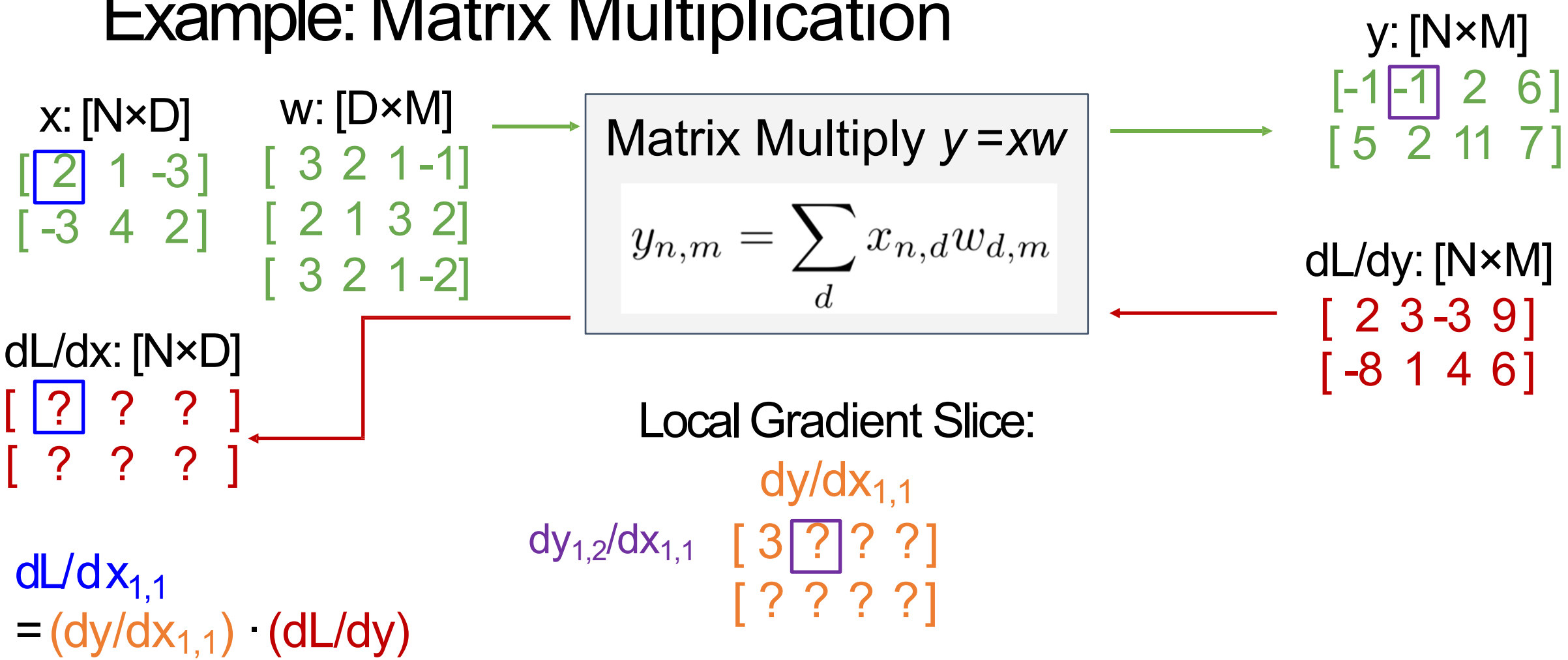
$dy/dx_{1,1}$

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

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 $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

[? ? ?]
[? ? ?]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

[? ? ?]
[? ? ?]

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

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 $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]
[? ? ? ?]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]
[? ? ? ?]

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

[? ? ?]
[? ? ?]

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

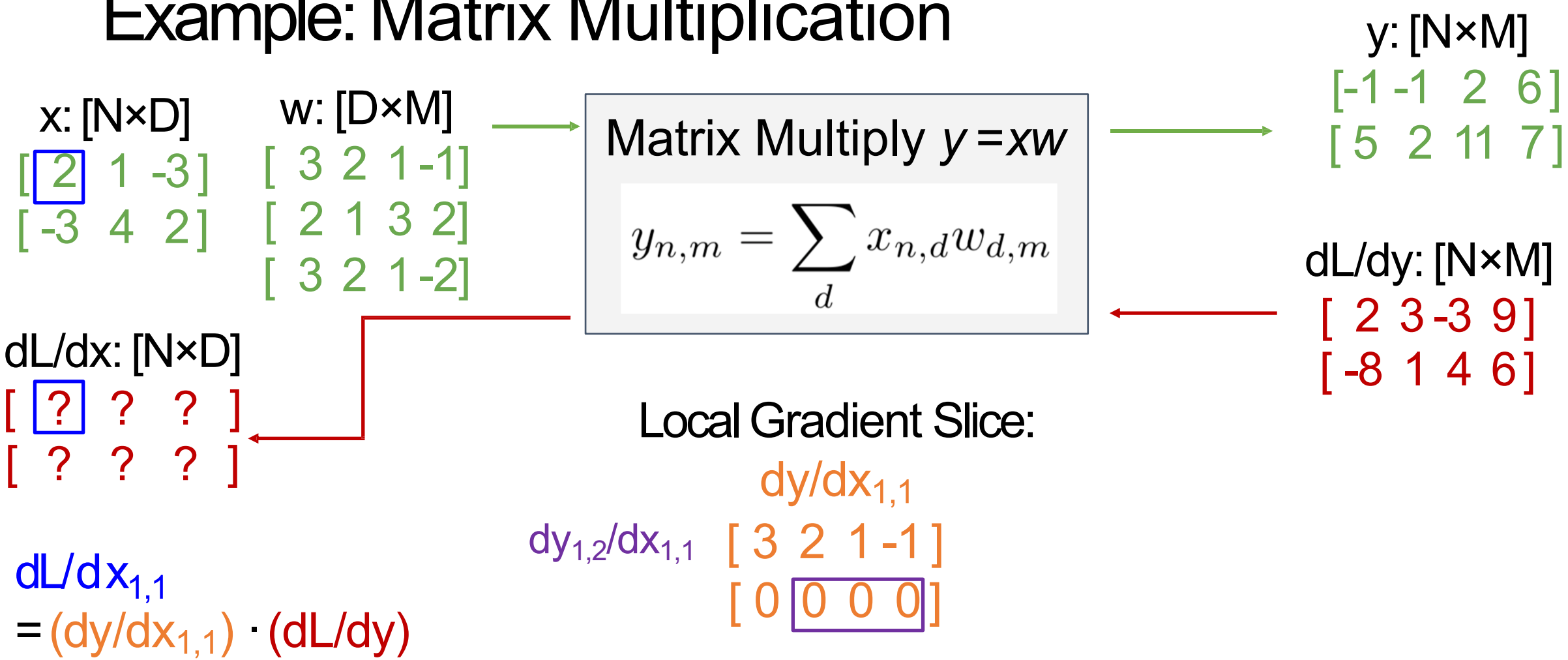
Local Gradient Slice:

$dy/dx_{1,1}$

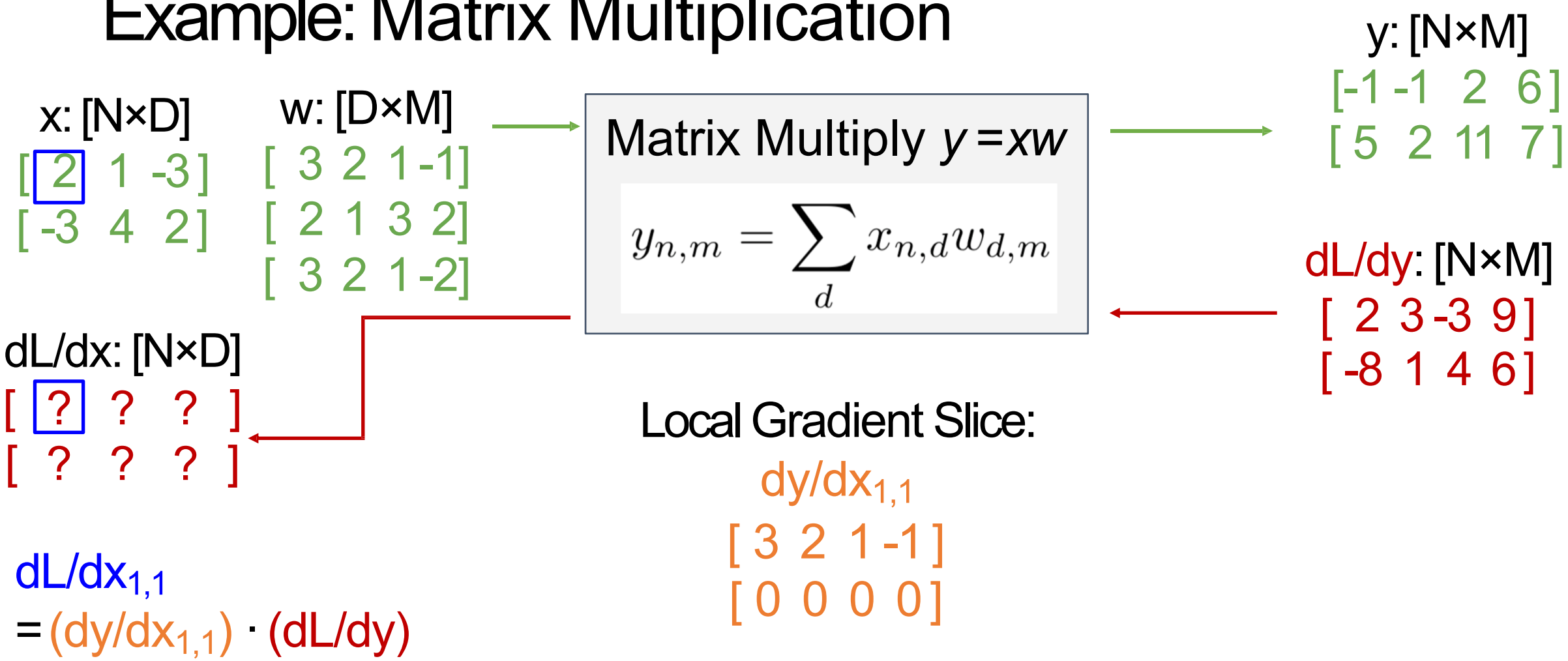
$dy_{1,2}/dx_{1,1}$ [3 2 1 -1]
[0 ? ? ?]

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

Example: Matrix Multiplication



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 $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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 ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

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]

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

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

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

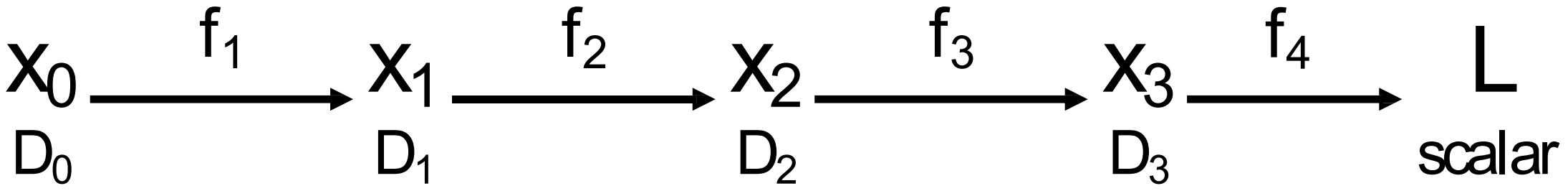
dL/dx: [N×D]
[0 16 -9]
[-24 9 -30]

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

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

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

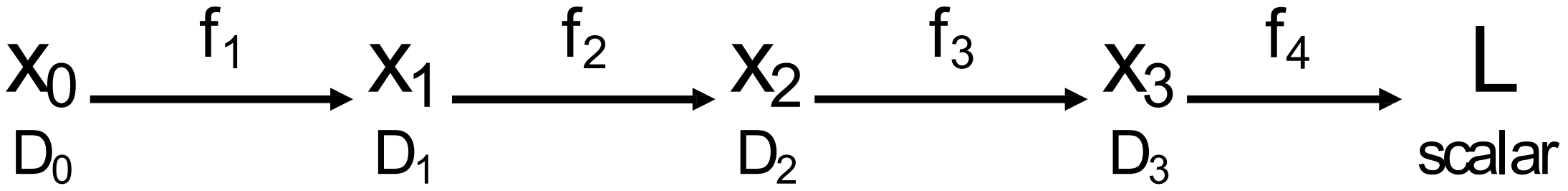
Backpropagation: Another View



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



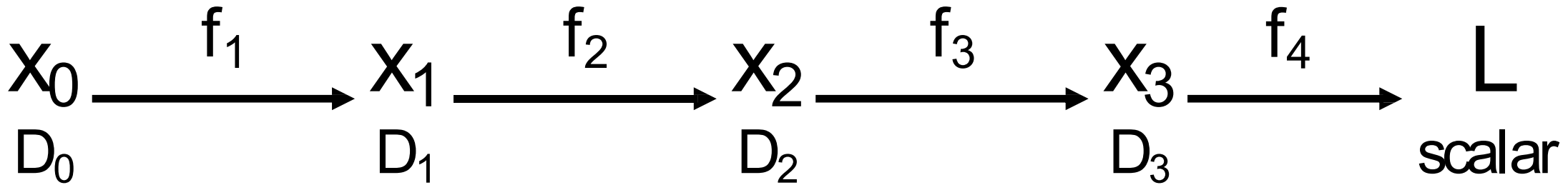
Matrix multiplication is **associative**: we can compute products in any order

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

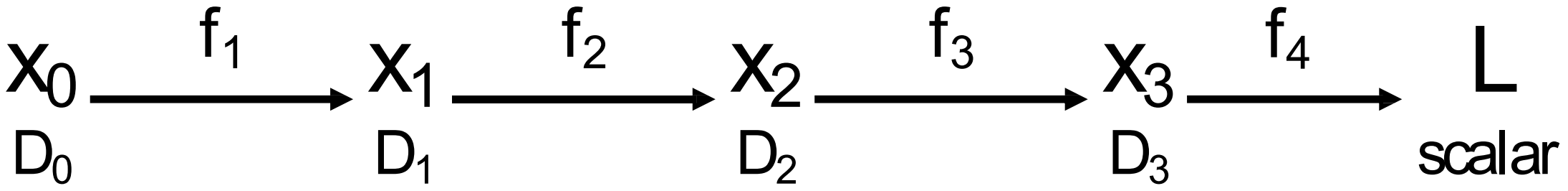
Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

Chain rule

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

$D_0 \times D_1$ $D_1 \times D_2$ $D_2 \times D_3$ 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

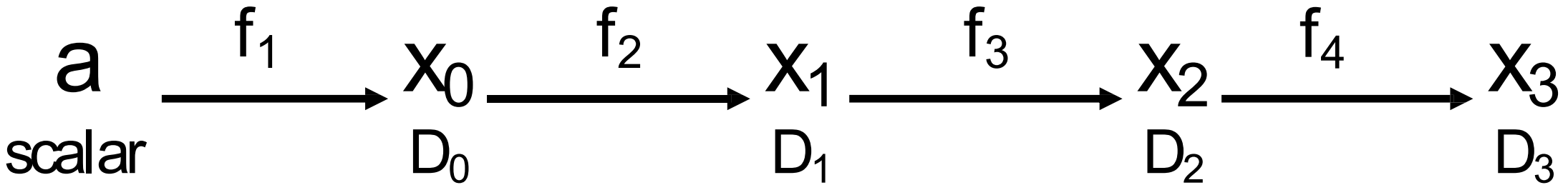
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

$D_0 \times D_1$ $D_1 \times D_2$ $D_2 \times D_3$ D_3

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

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

Forward-Mode Automatic Differentiation

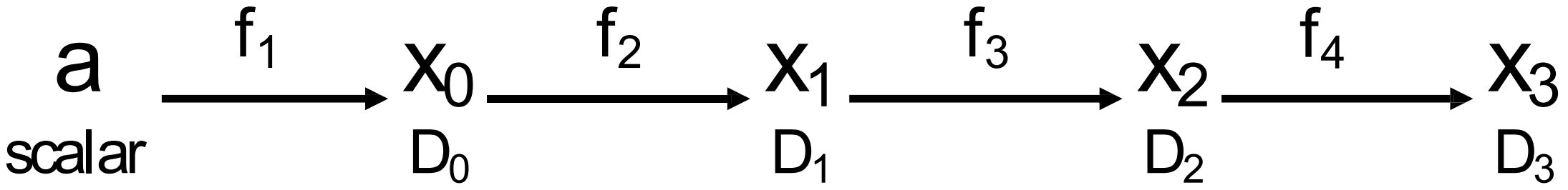


Chain rule

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

D_0 $D_0 \times D_1$ $D_1 \times D_2$ $D_2 \times D_3$

Forward-Mode Automatic Differentiation



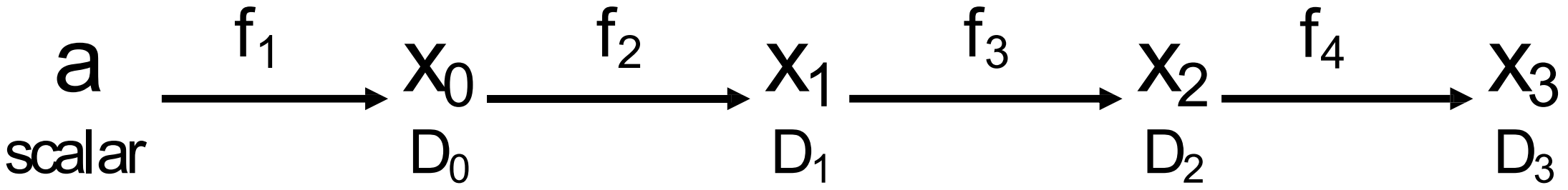
Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

Chain rule

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

D_0 $D_0 \times D_1$ $D_1 \times D_2$ $D_2 \times D_3$

Forward-Mode Automatic Differentiation



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

Not implemented in PyTorch / TensorFlow

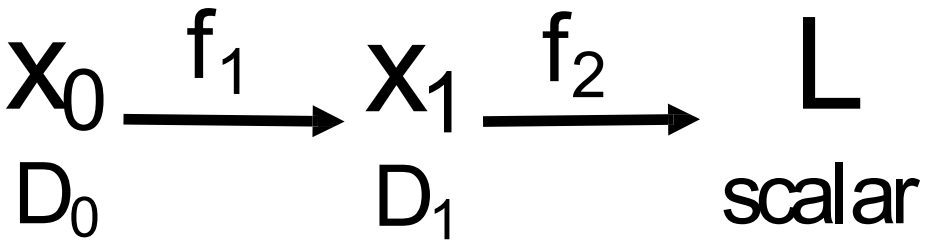
Chain rule

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

D_0 $D_0 \times D_1$ $D_1 \times D_2$ $D_2 \times D_3$

But you can implement forward-mode AD using two calls to reverse-mode AD! (Inefficient but elegant)

Backprop: Higher-Order Derivatives

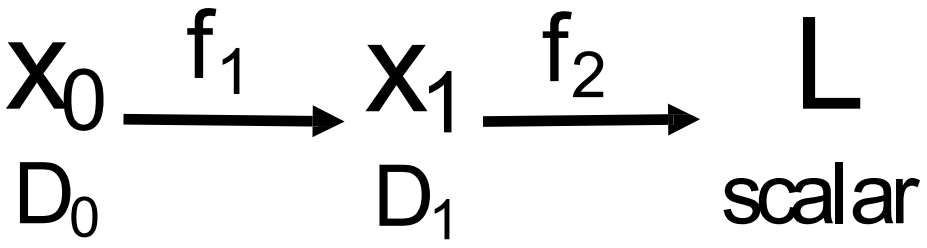


$$\frac{\partial^2 L}{\partial x_0^2}$$

$D_0 \times D_0$

Hessian matrix H of second derivatives.

Backprop: Higher-Order Derivatives



Hessian / vector multiply

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

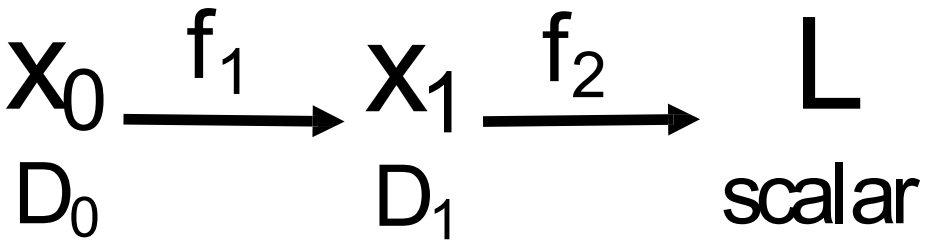
Hessian matrix
H of second derivatives.

$D_0 \times D_0$

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

$D_0 \times D_0 \quad D_0$

Backprop: Higher-Order Derivatives



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

$D_0 \times D_0$

Hessian matrix
H of second derivatives.

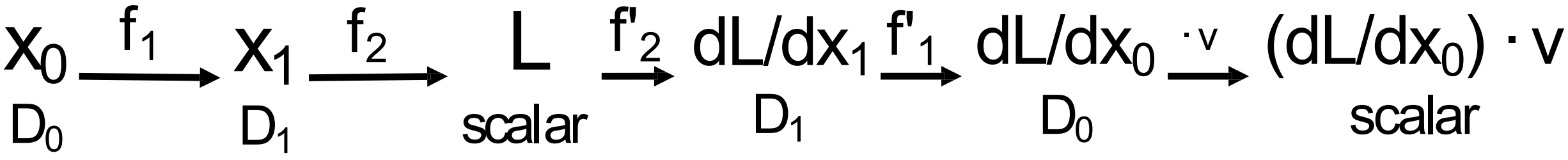
Hessian / vector multiply

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

$D_0 \times D_0 \quad D_0$

(if v doesn't depend on x_0)

Backprop: Higher-Order Derivatives



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

Hessian matrix
H of second derivatives.

$D_0 \times D_0$

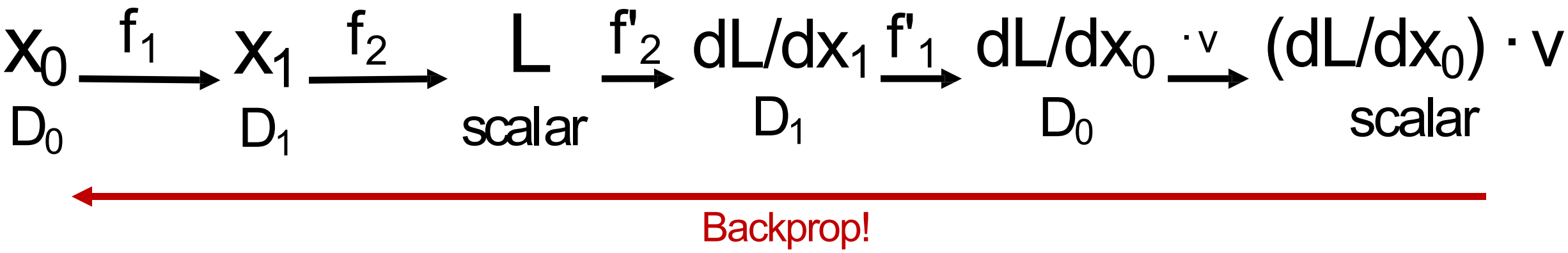
Hessian / vector multiply

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

(if v doesn't depend on x_0)

$D_0 \times D_0 \quad D_0$

Backprop: Higher-Order Derivatives



Hessian / vector multiply

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

$D_0 \times D_0$

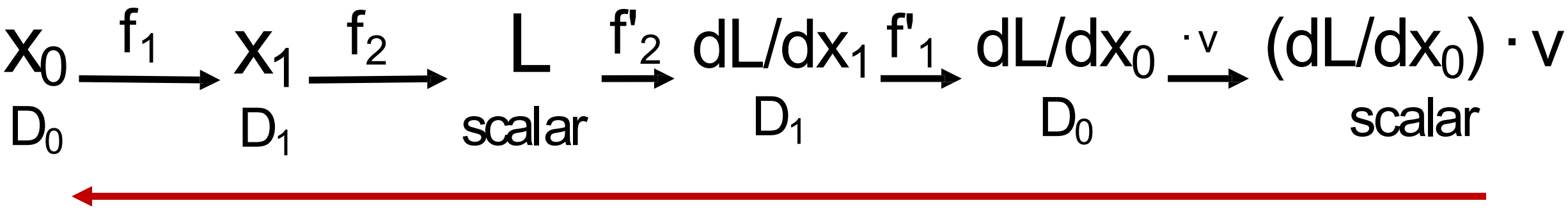
Hessian matrix
H of second derivatives.

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

$D_0 \times D_0$ D_0

(if v doesn't depend on x_0)

Backprop: Higher-Order Derivatives



This is implemented in PyTorch / Tensorflow!

Hessian / vector multiply

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

Hessian matrix H of second derivatives.

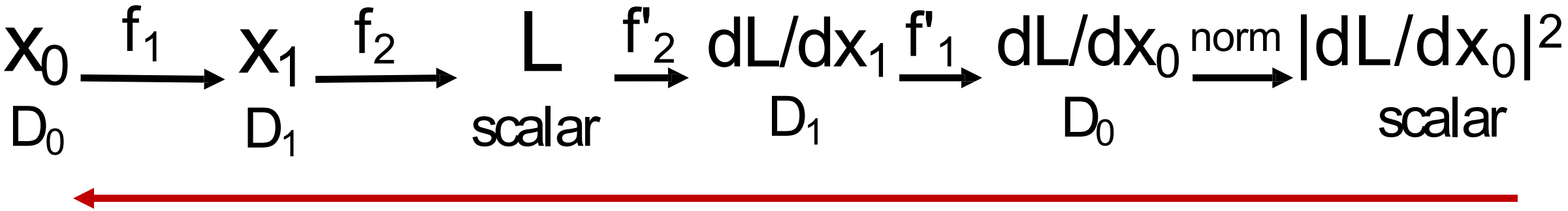
$D_0 \times D_0$

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

(if v doesn't depend on x_0)

$D_0 \times D_0 \quad D_0$

Backprop: Higher-Order Derivatives



This is implemented in PyTorch / Tensorflow!

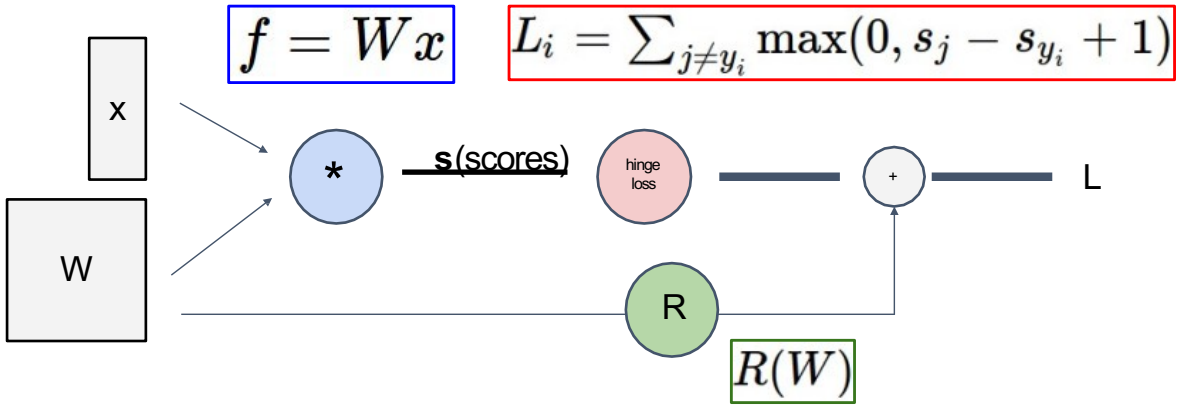
Example: Regularization to penalize the norm of the gradient

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

Gulrajani et al, "Improved Training of Wasserstein GANs", NeurIPS 2017

Summary

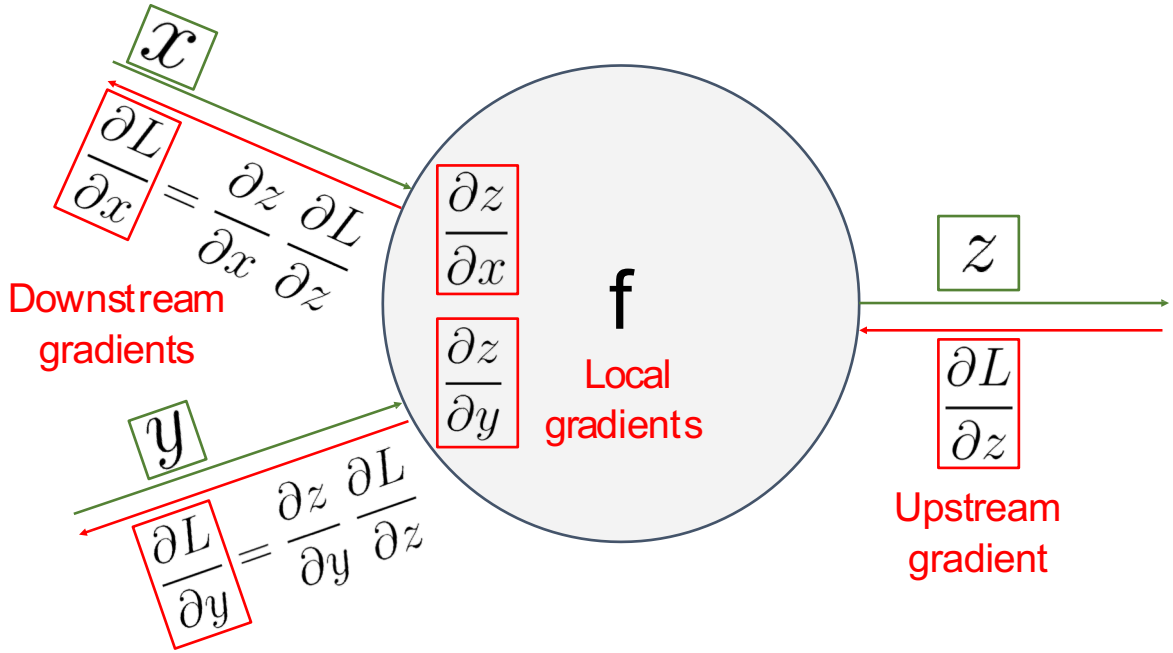
Represent complex expressions as **computational graphs**



Forward pass computes outputs

Backward pass computes gradients

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



Summary

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

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

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

Backprop can be implemented with a modular API, as a set of paired forward/backward functions (We will do this on A3!)

```
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
```