Lecture 6: Backpropagation

Reminder: A2

- Use SGD to train linear classifiers and fully-connected networks
- Today's lecture can help you compute derivatives in A2
- Due Friday January 28, 11:59pm ET

Autograder

- A1: 10 submissions / day
- A2: 5 submissions / day
- A3+: 3 submissions / day

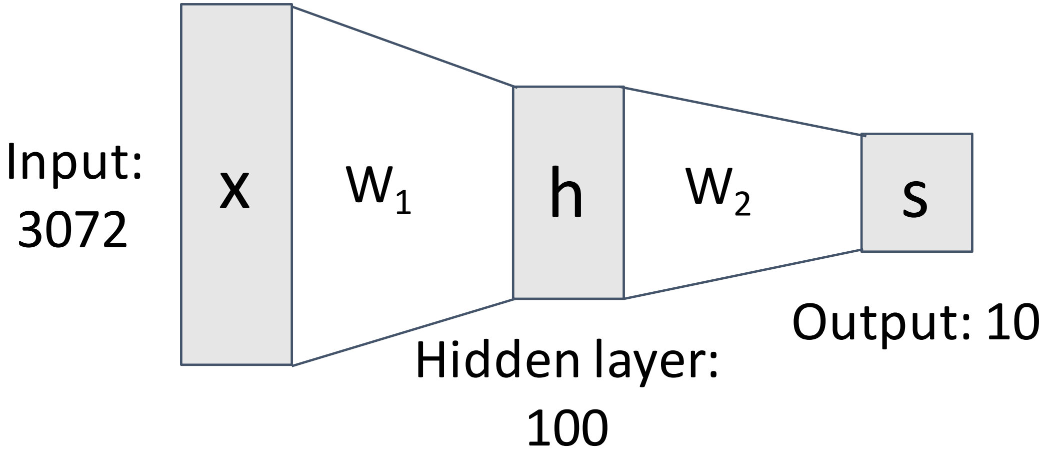
Autograder is a val set, not a training set! You shouldn't be using autograder to debug; you should be testing your code on the side

Extra incentive to start early

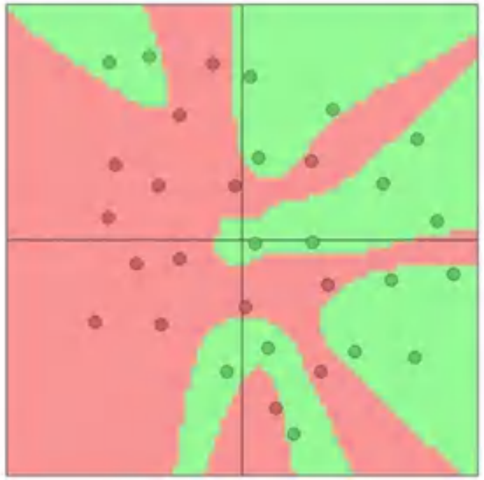
Last time: Neural Networks

From linear classifiers to
fully-connected networks

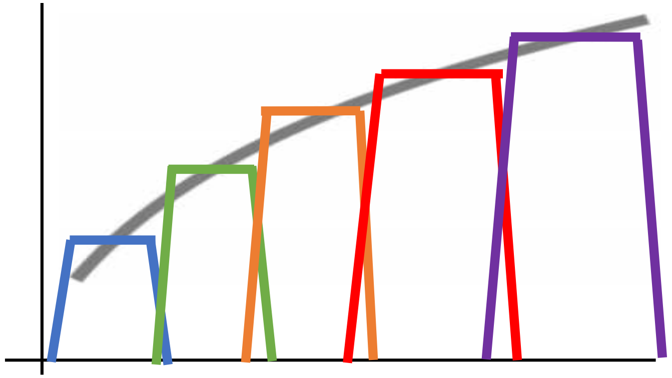
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



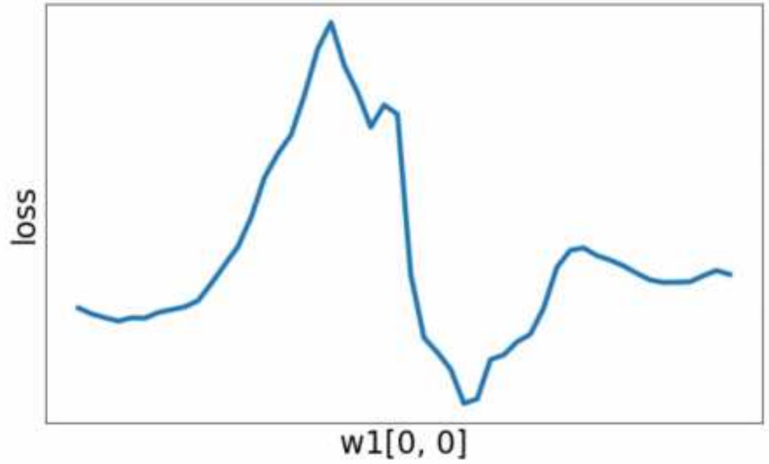
Space Warping



Universal Approximation



Nonconvex



Problem: How to compute gradients?

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

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

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

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

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

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

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

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

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

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

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

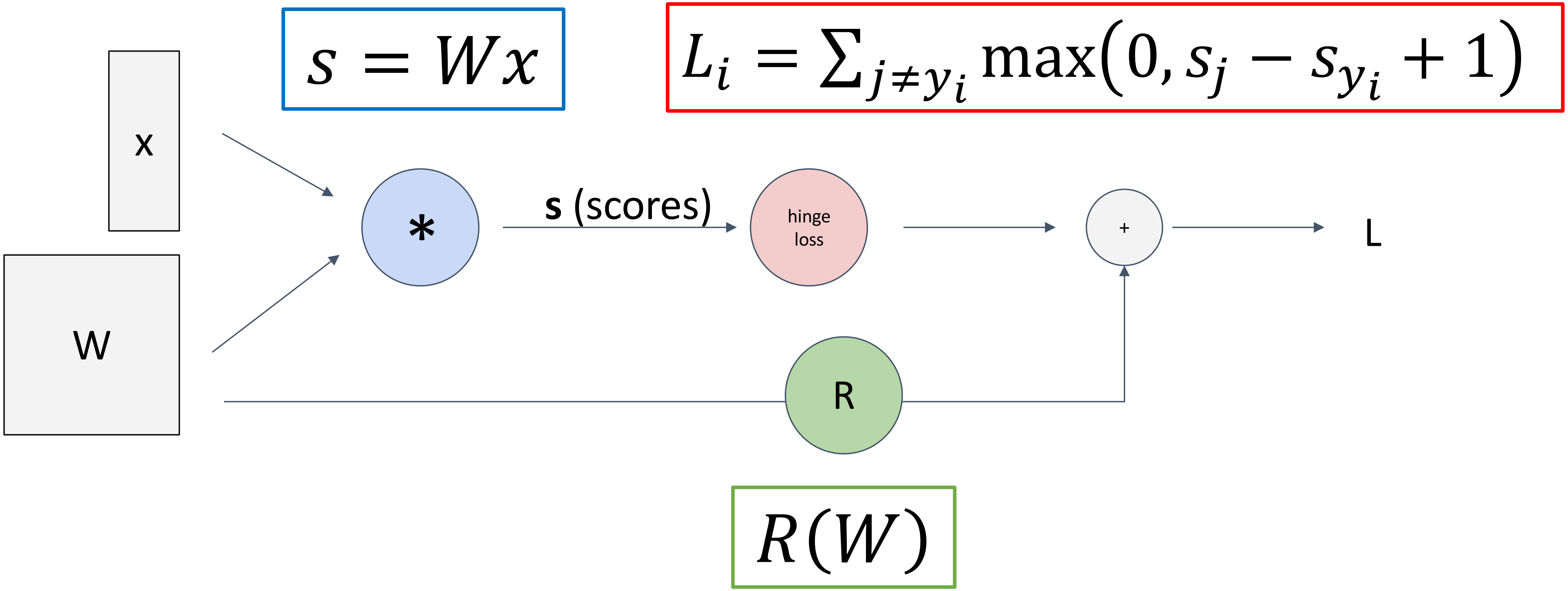
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

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

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

Problem: Not feasible for very complex models!

Better Idea: **Computational Graphs**

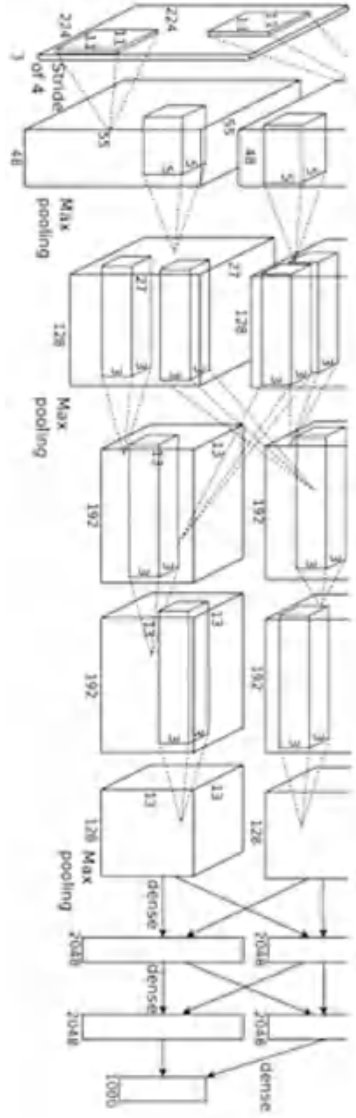


Deep Network (AlexNet)

input image

weights

loss



Neural Turing Machine

input image

loss

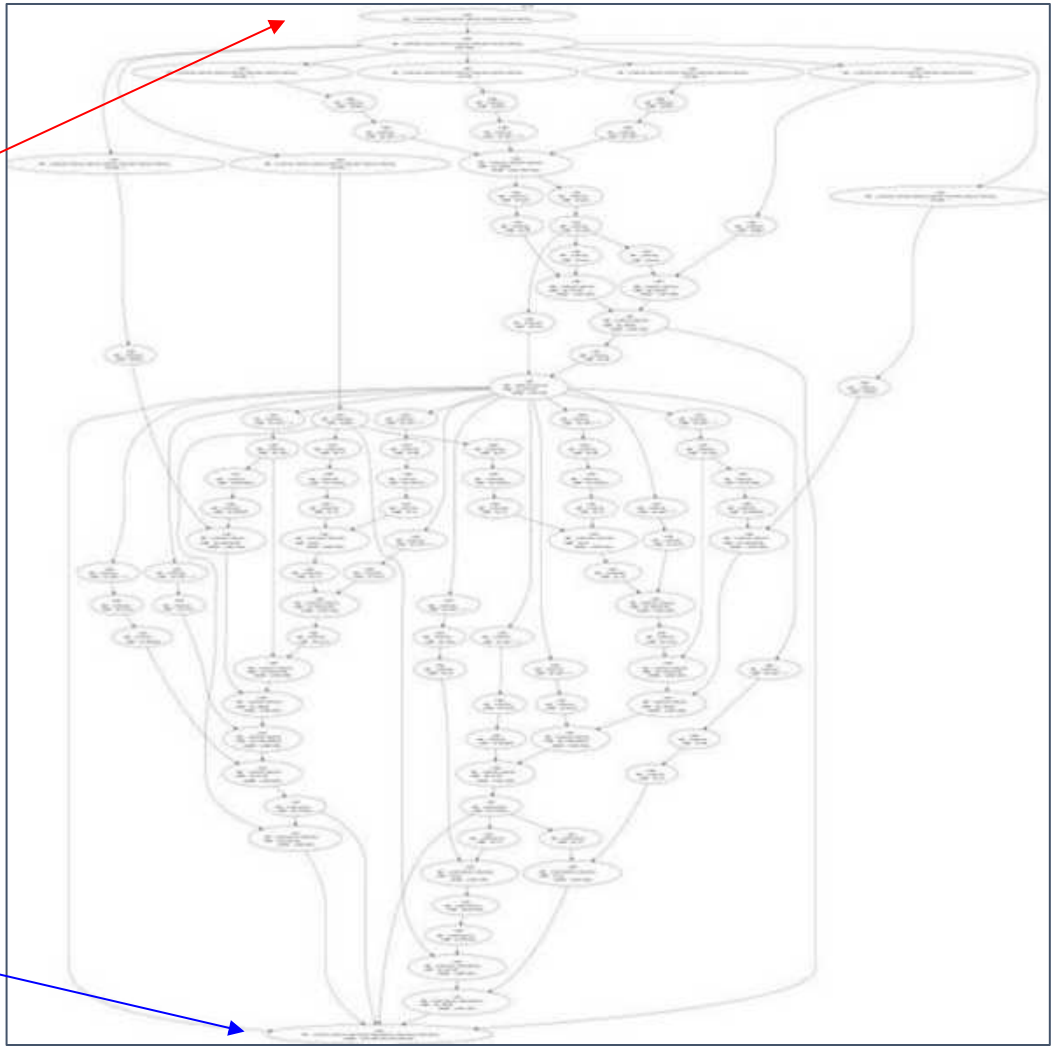
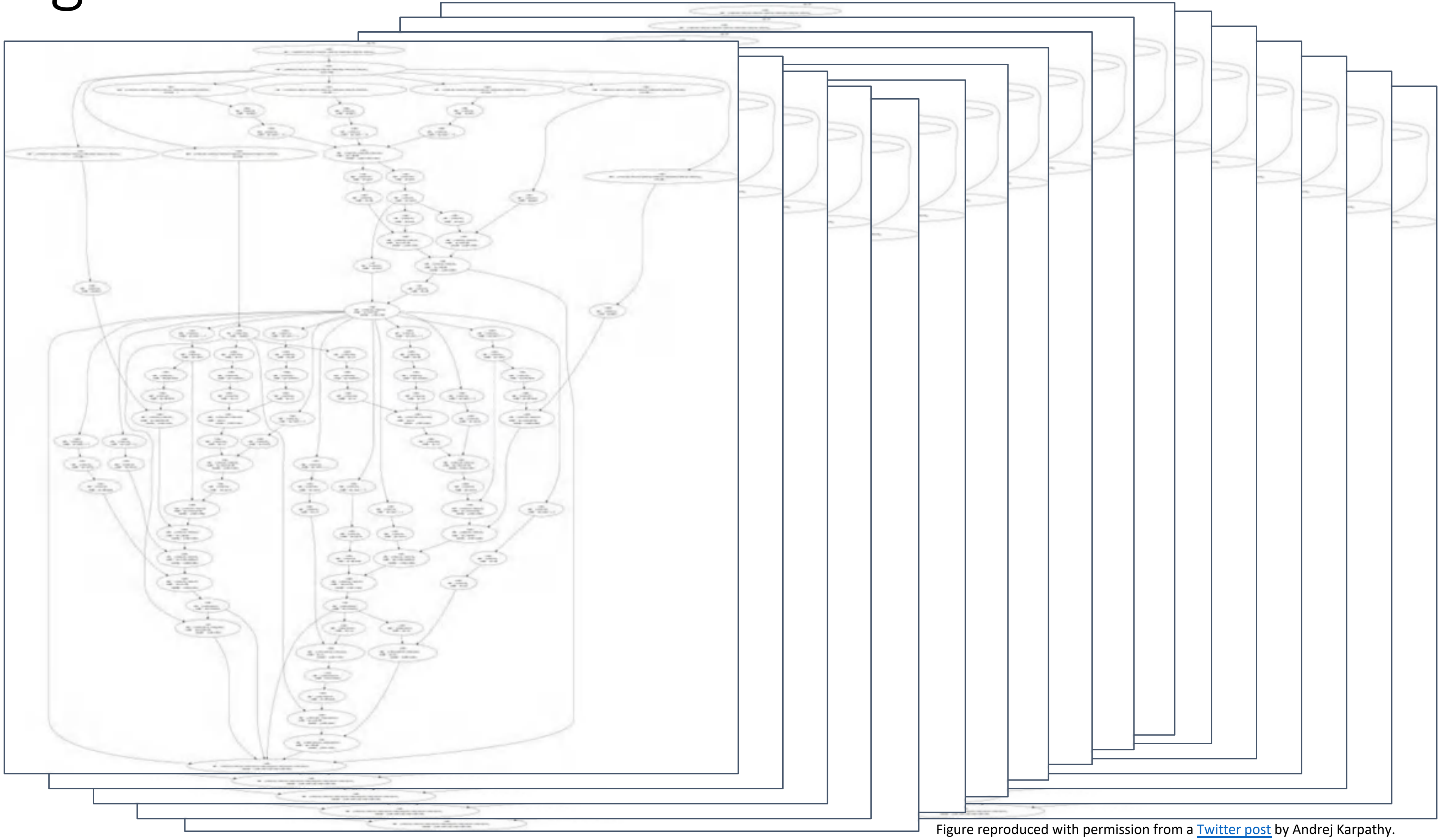


Figure reproduced with permission from a [Twitter post]() by Andrej Karpathy.

Graves et al, arXiv 2014

Neural Turing Machine

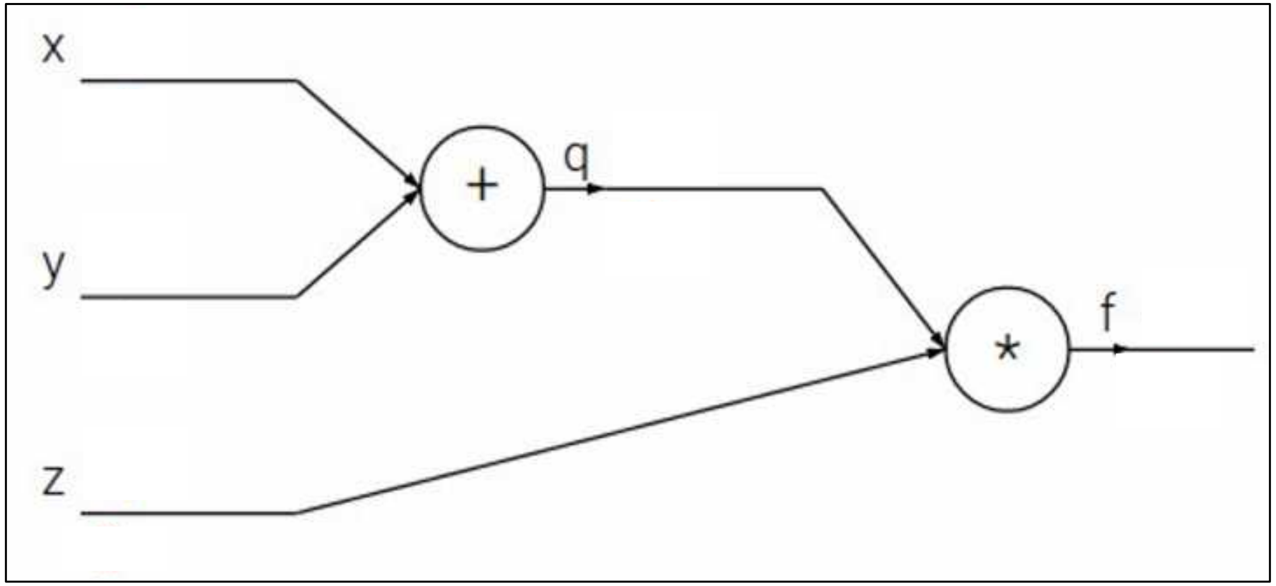


Graves et al, arXiv 2014

Figure reproduced with permission from a [Twitter post](https://twitter.com/karpathy) by Andrej Karpathy.

Backpropagation: Simple Example

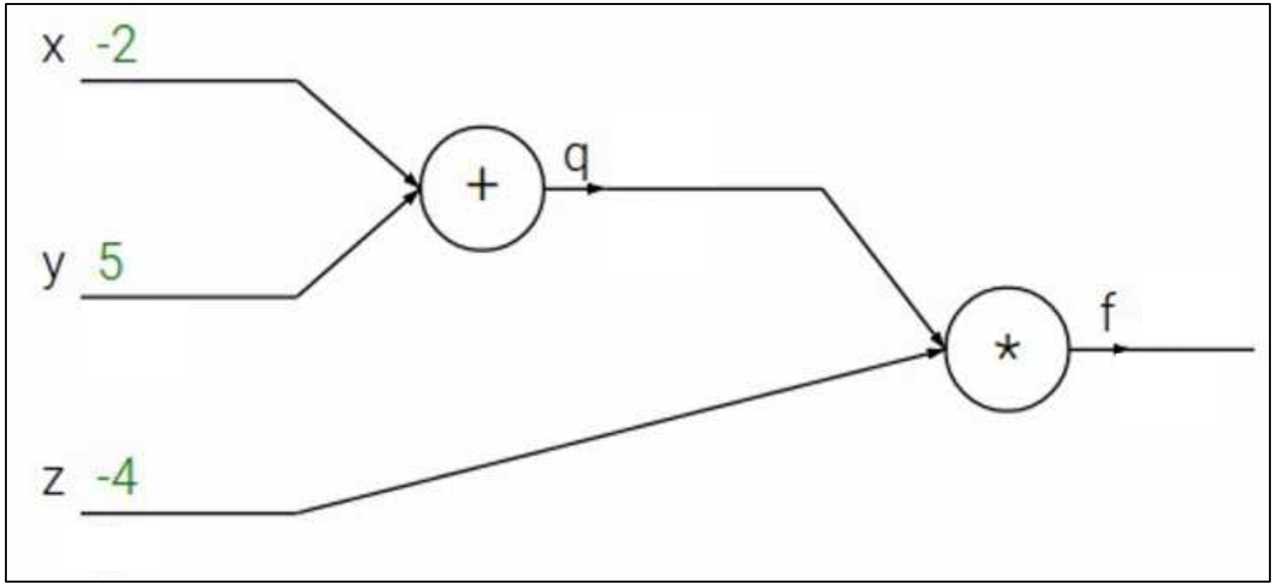
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

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

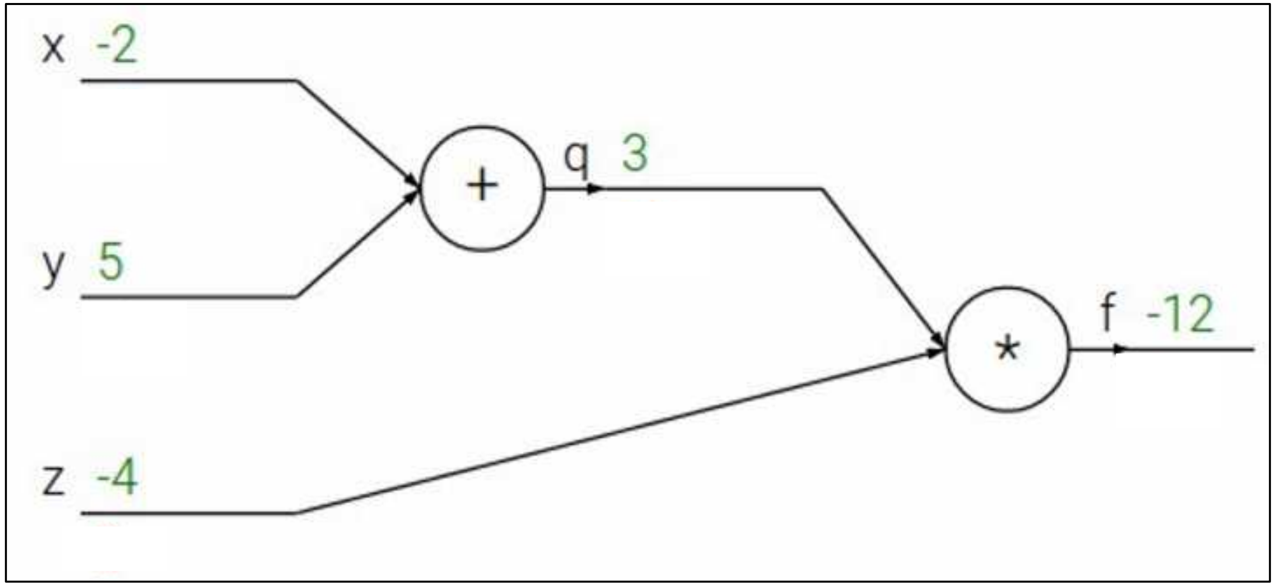
e.g. x = -2, y = 5, z = -4



Backpropagation: Simple Example

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

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



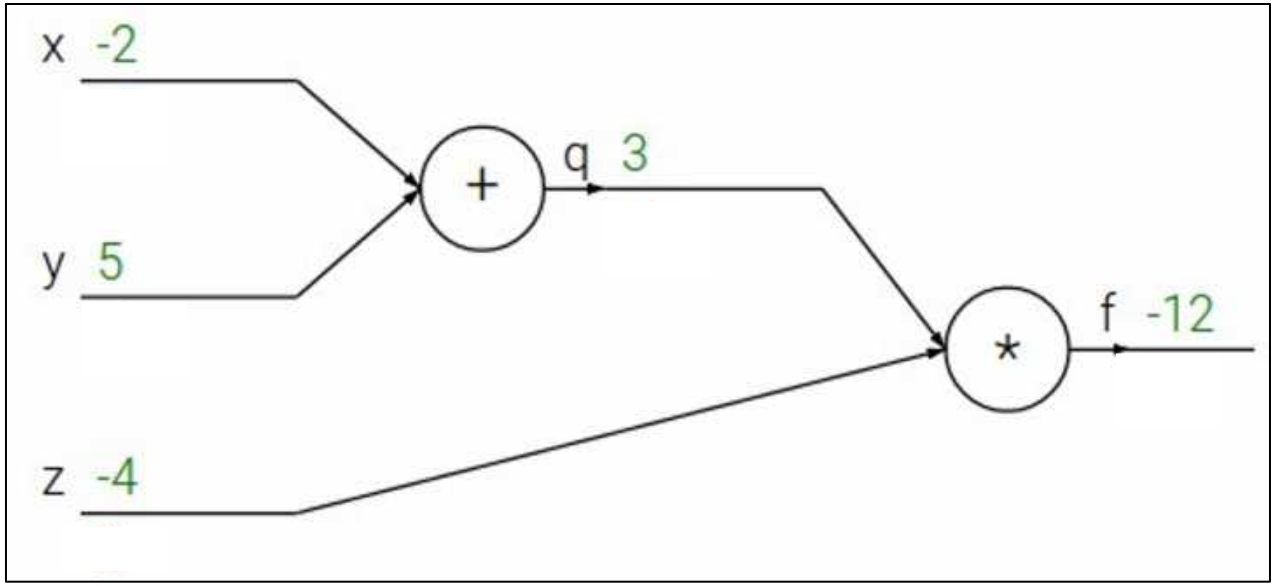
1. Forward pass: Compute outputs

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

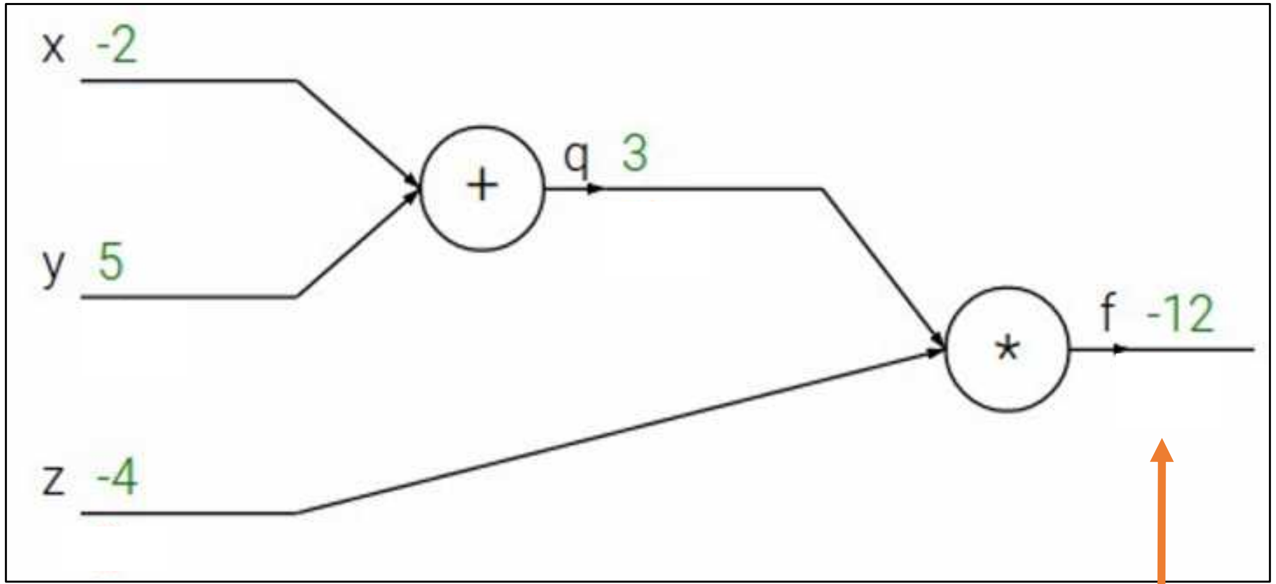
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



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

1. Forward pass: Compute outputs

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

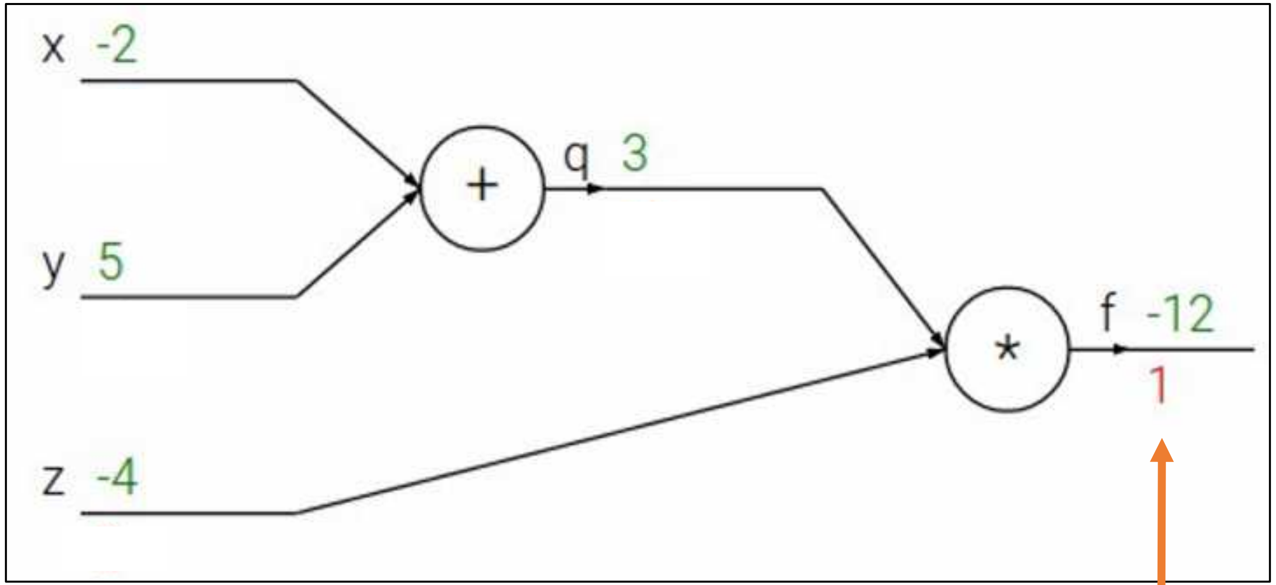
2. Backward pass: Compute derivatives

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

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

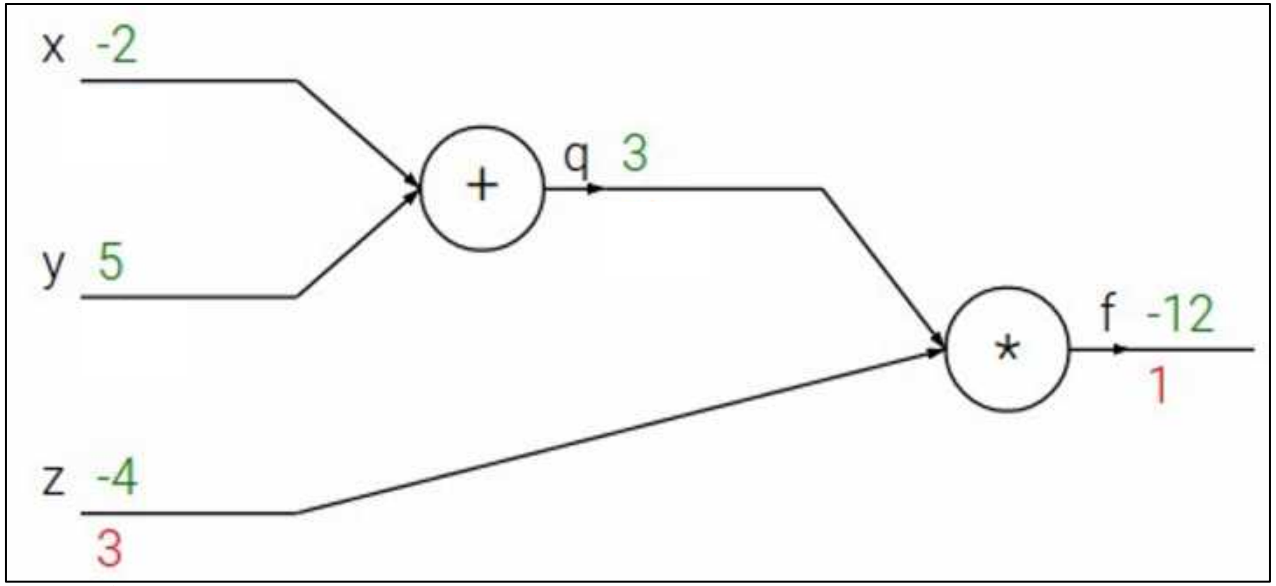
$$\frac{\partial f}{\partial f}$$

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

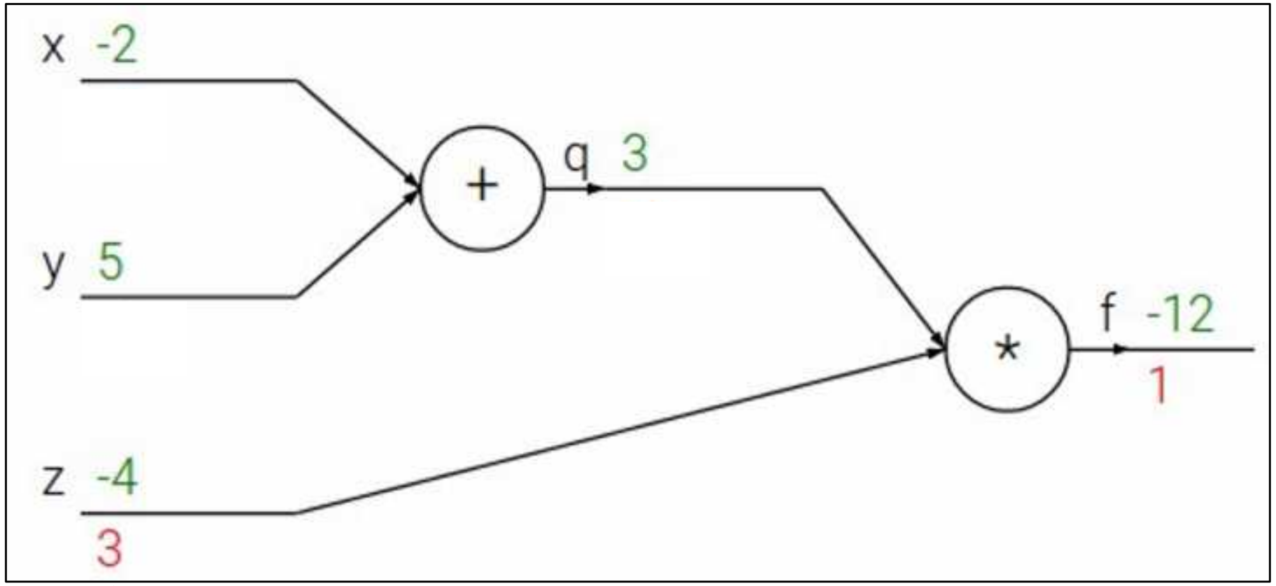
Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

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

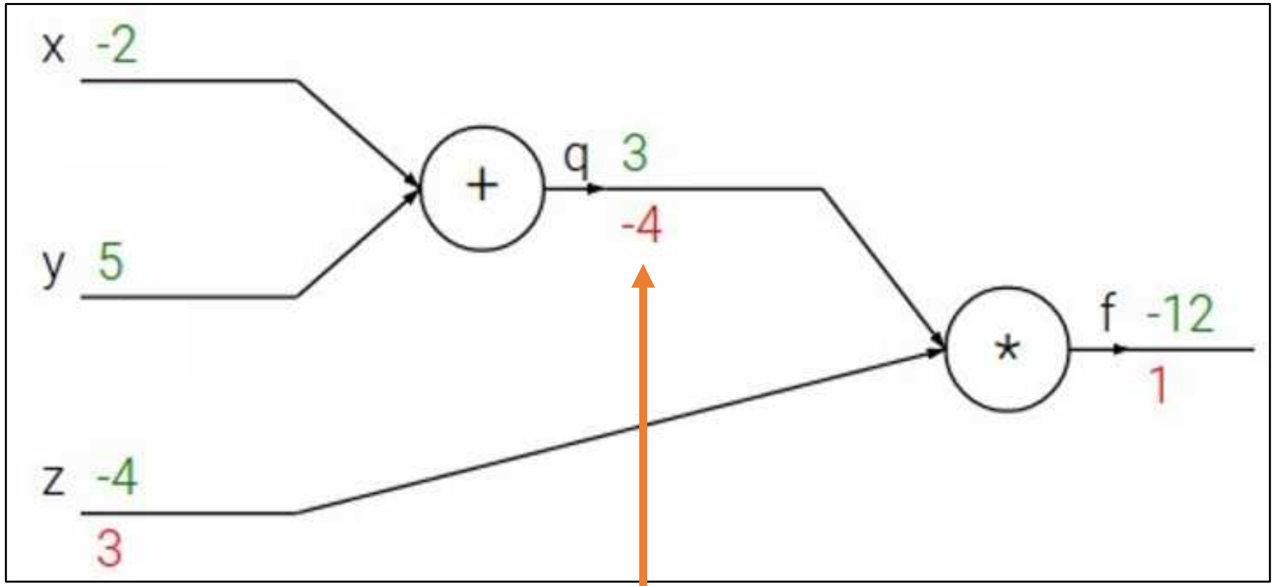
2. Backward pass: Compute derivatives

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

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

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

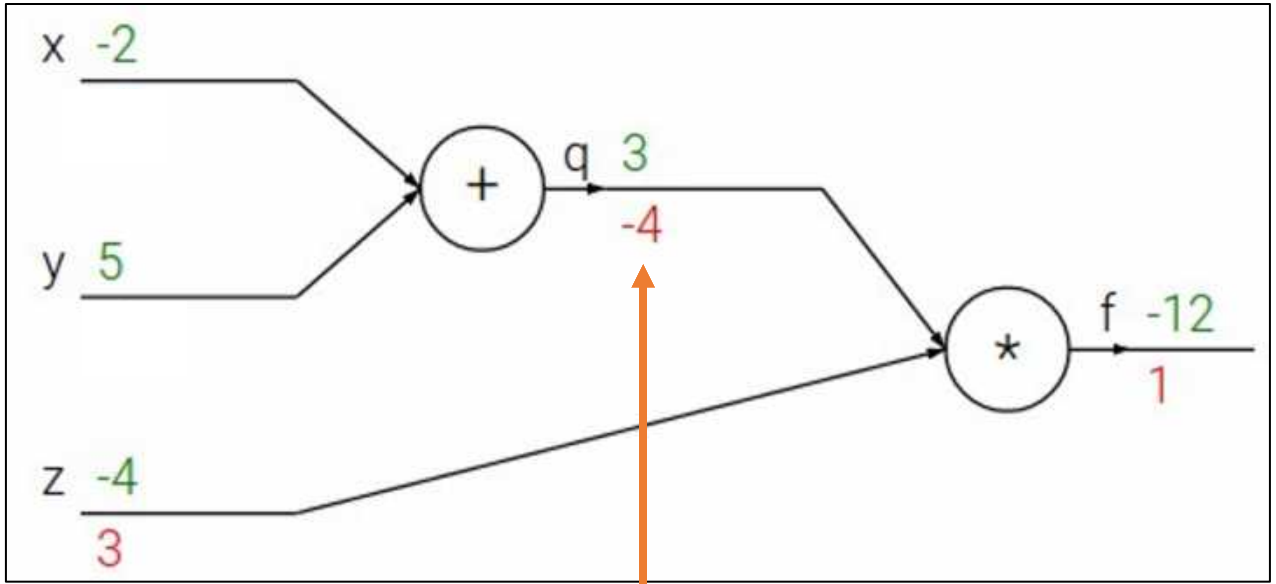
2. Backward pass: Compute derivatives

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



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

1. Forward pass: Compute outputs

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

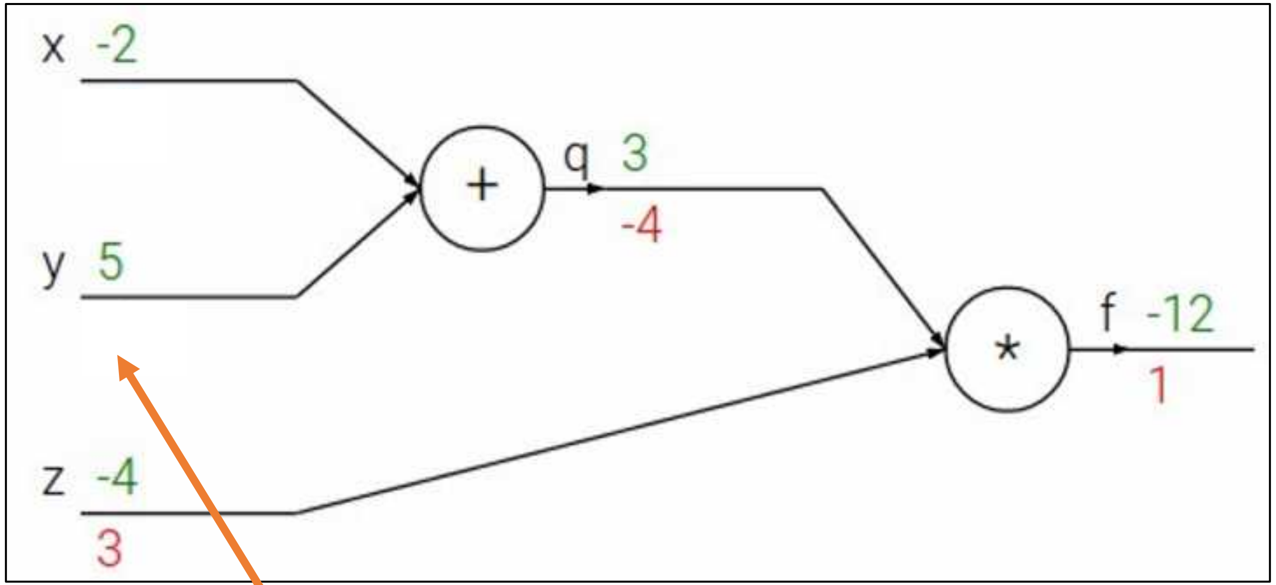
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



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

1. Forward pass: Compute outputs

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

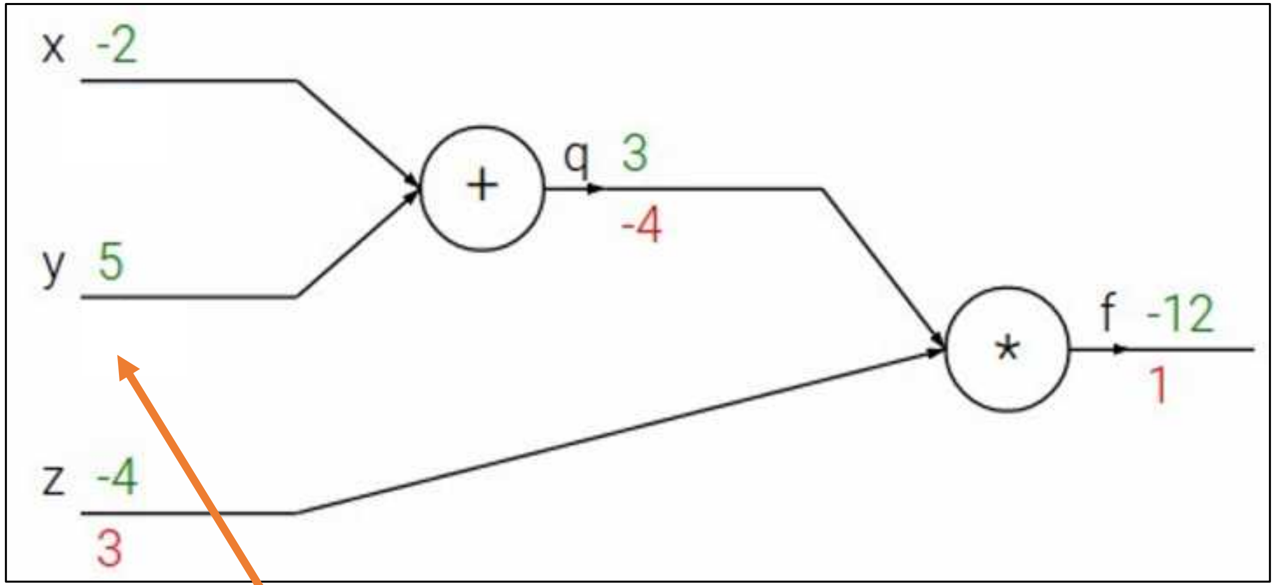
2. Backward pass: Compute derivatives

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

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

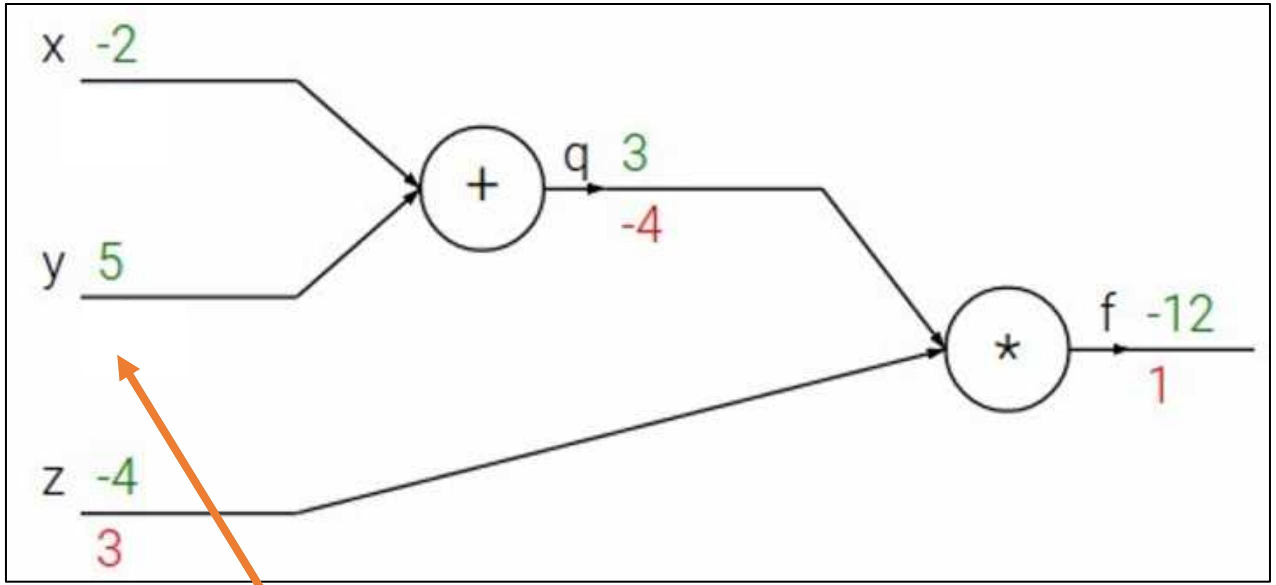
Chain Rule

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

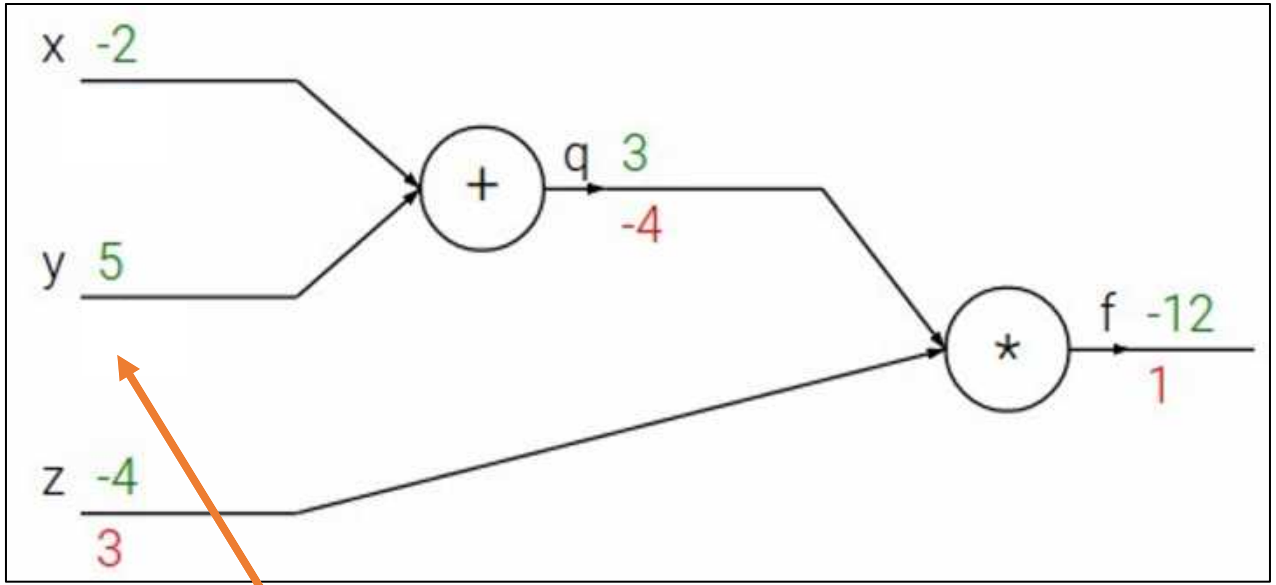
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}}$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

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

Backpropagation: Simple Example

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

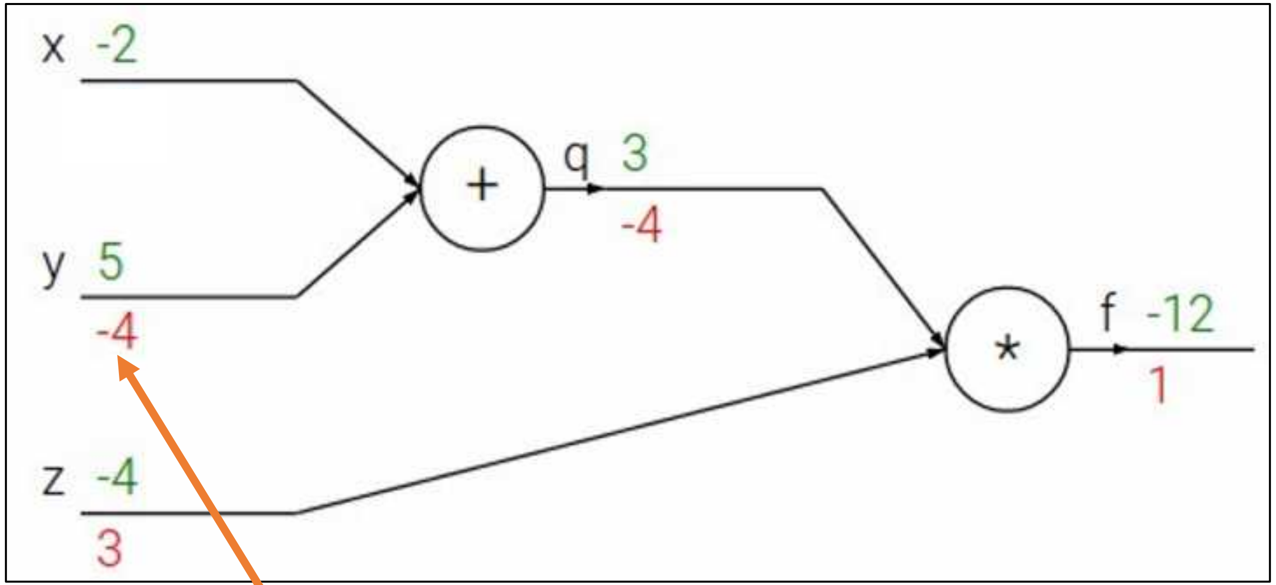
e.g. x = -2, y = 5, z = -4

1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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



Chain Rule

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

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

Backpropagation: Simple Example

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

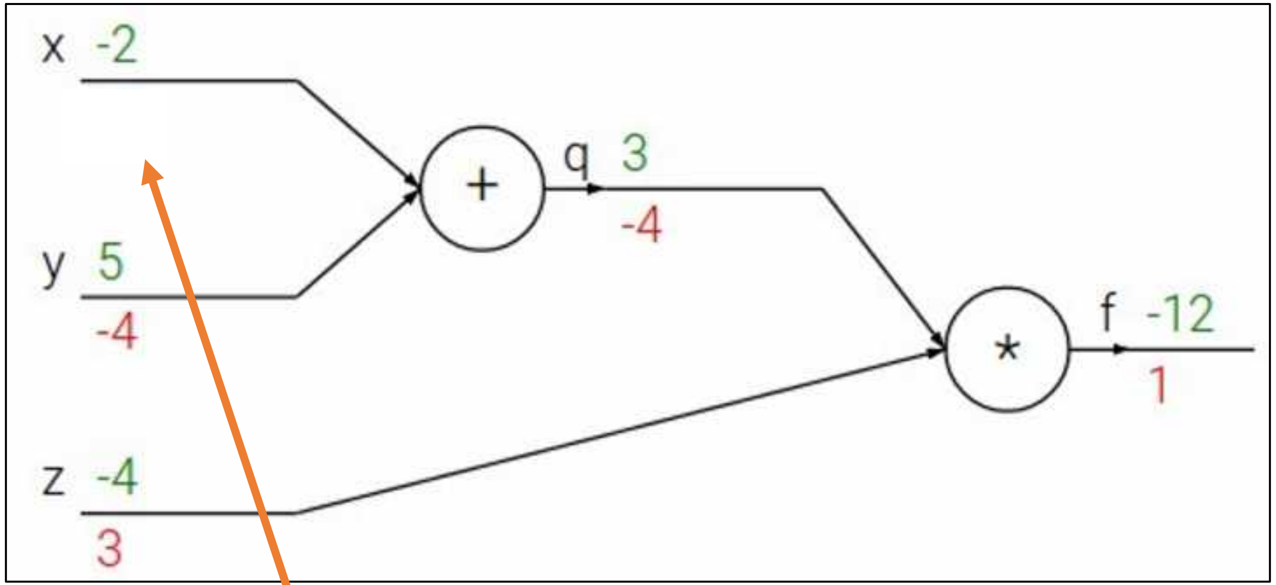
e.g. x = -2, y = 5, z = -4

1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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



Chain Rule

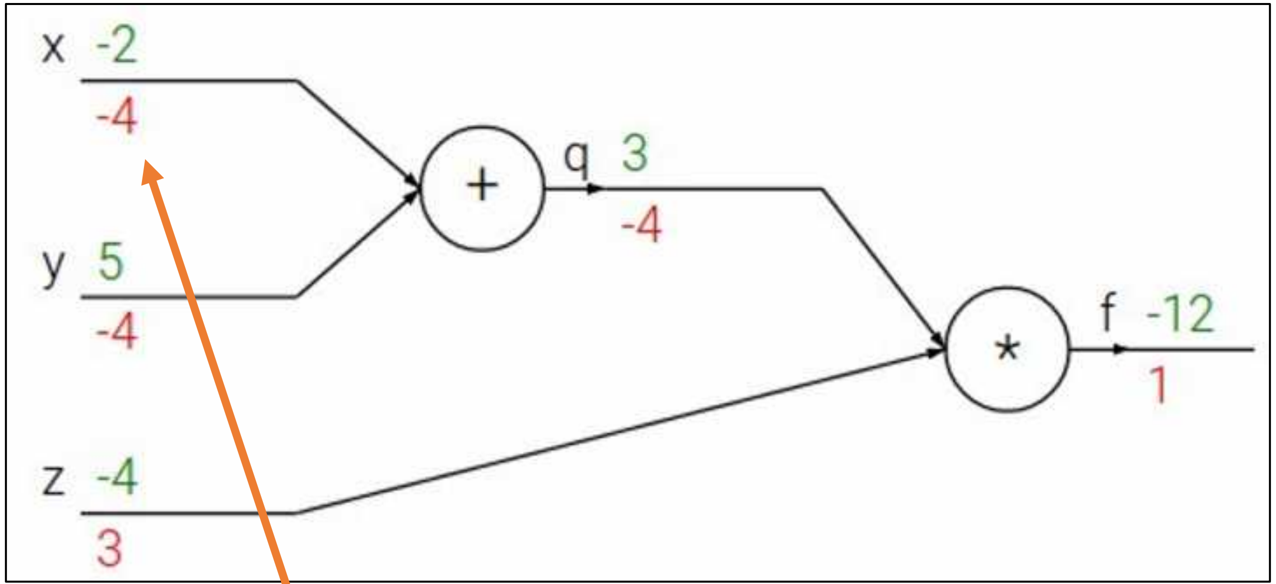
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

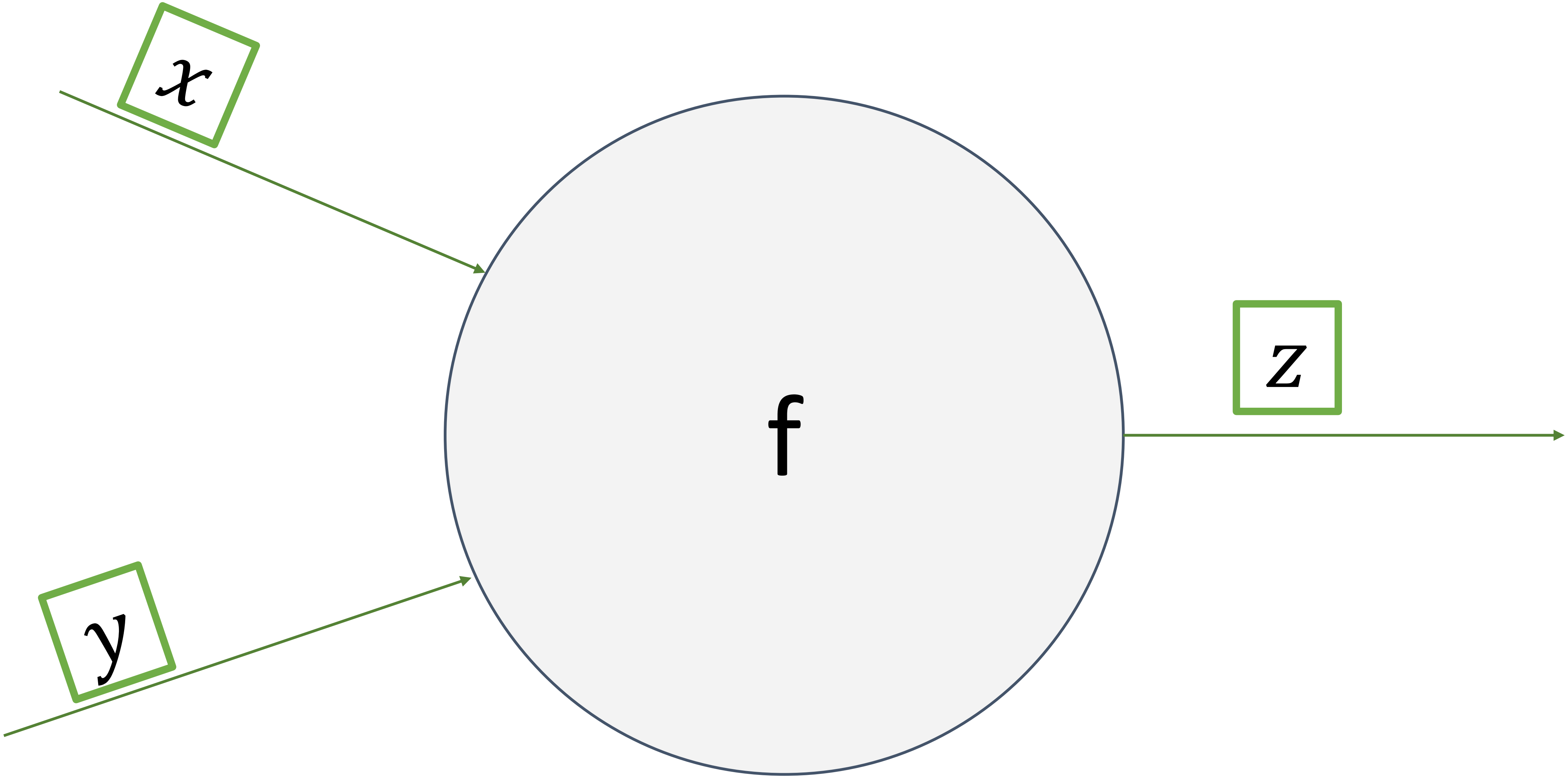
2. Backward pass: Compute derivatives

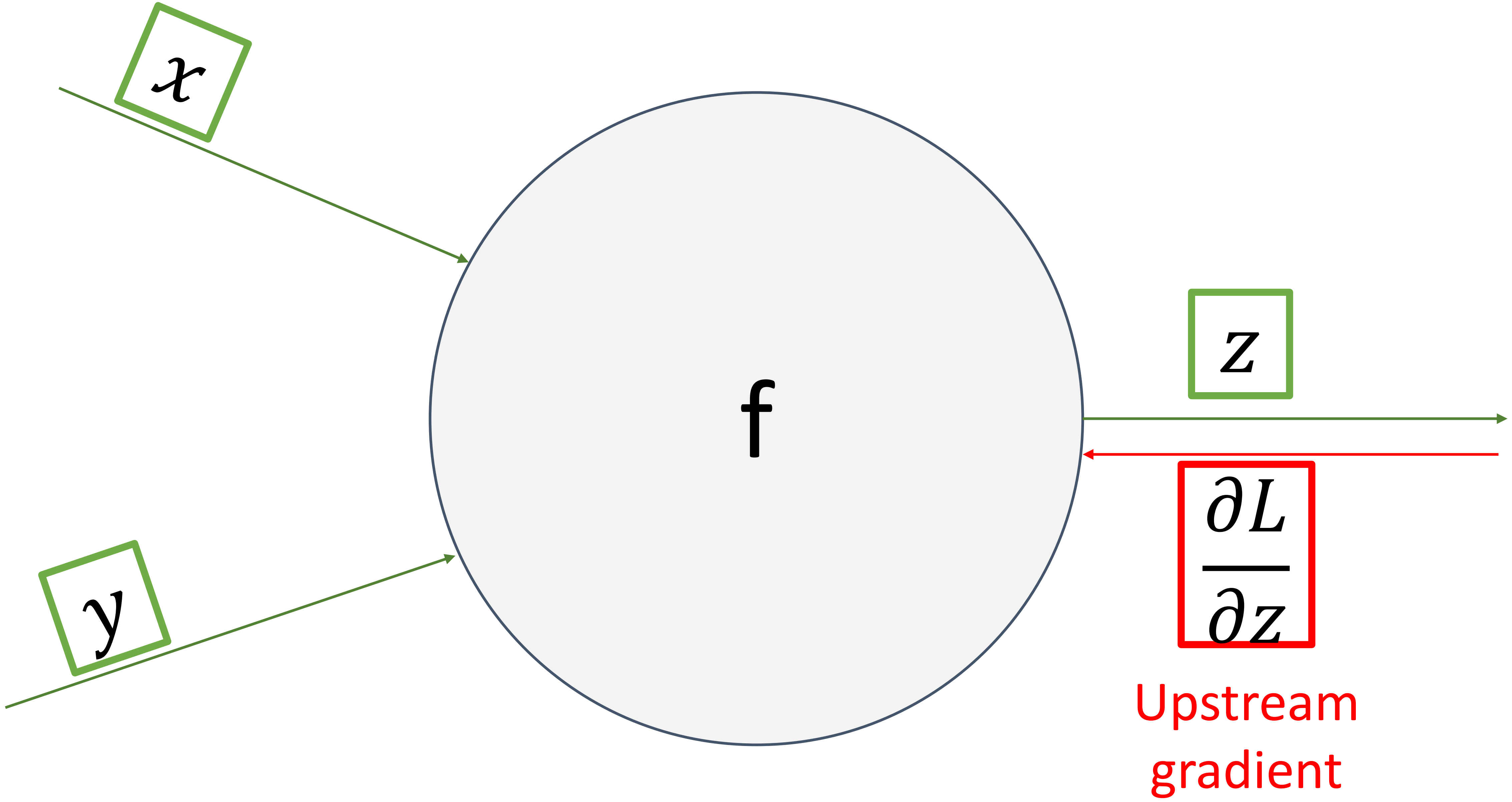
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule

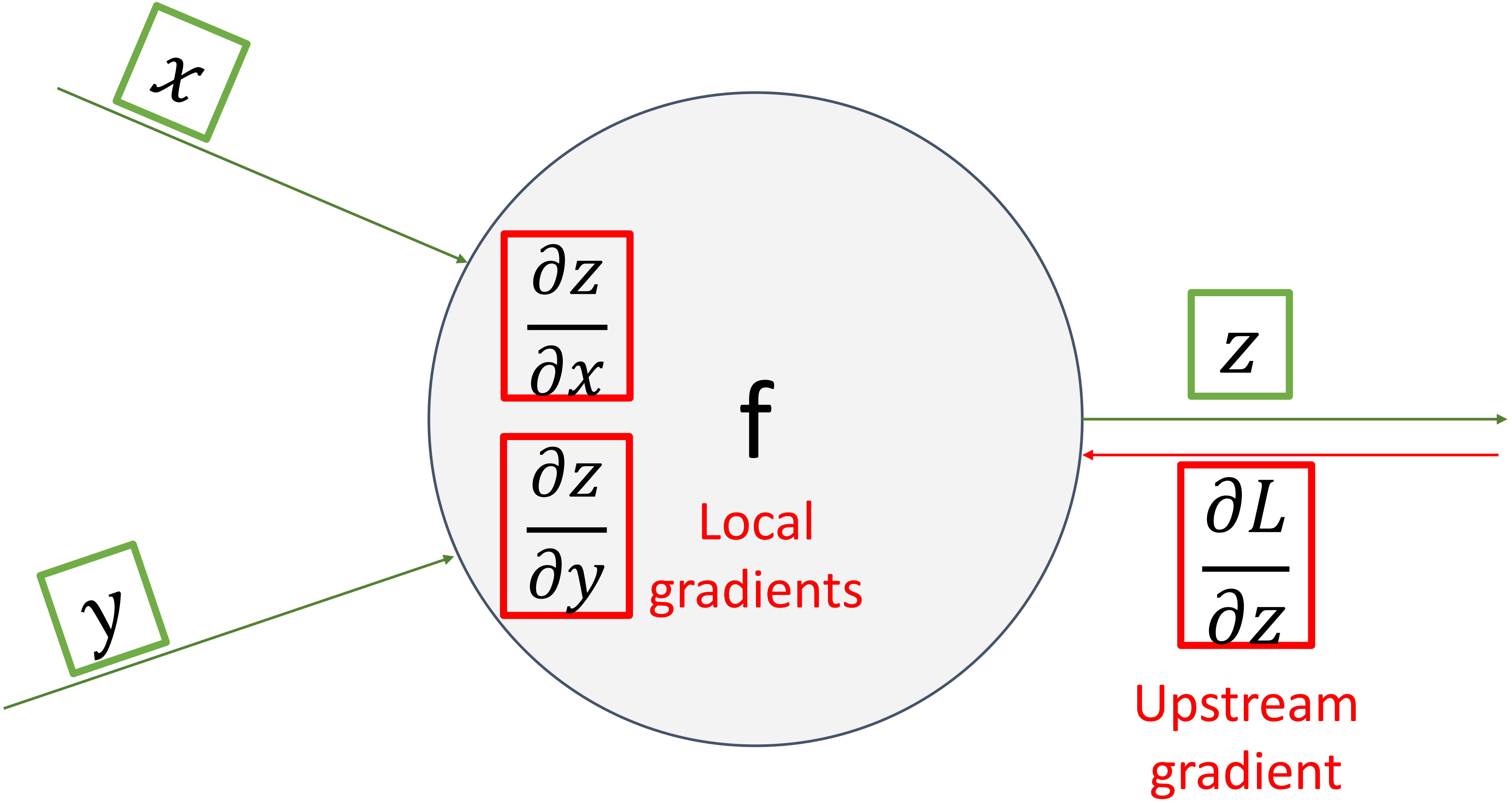
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

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

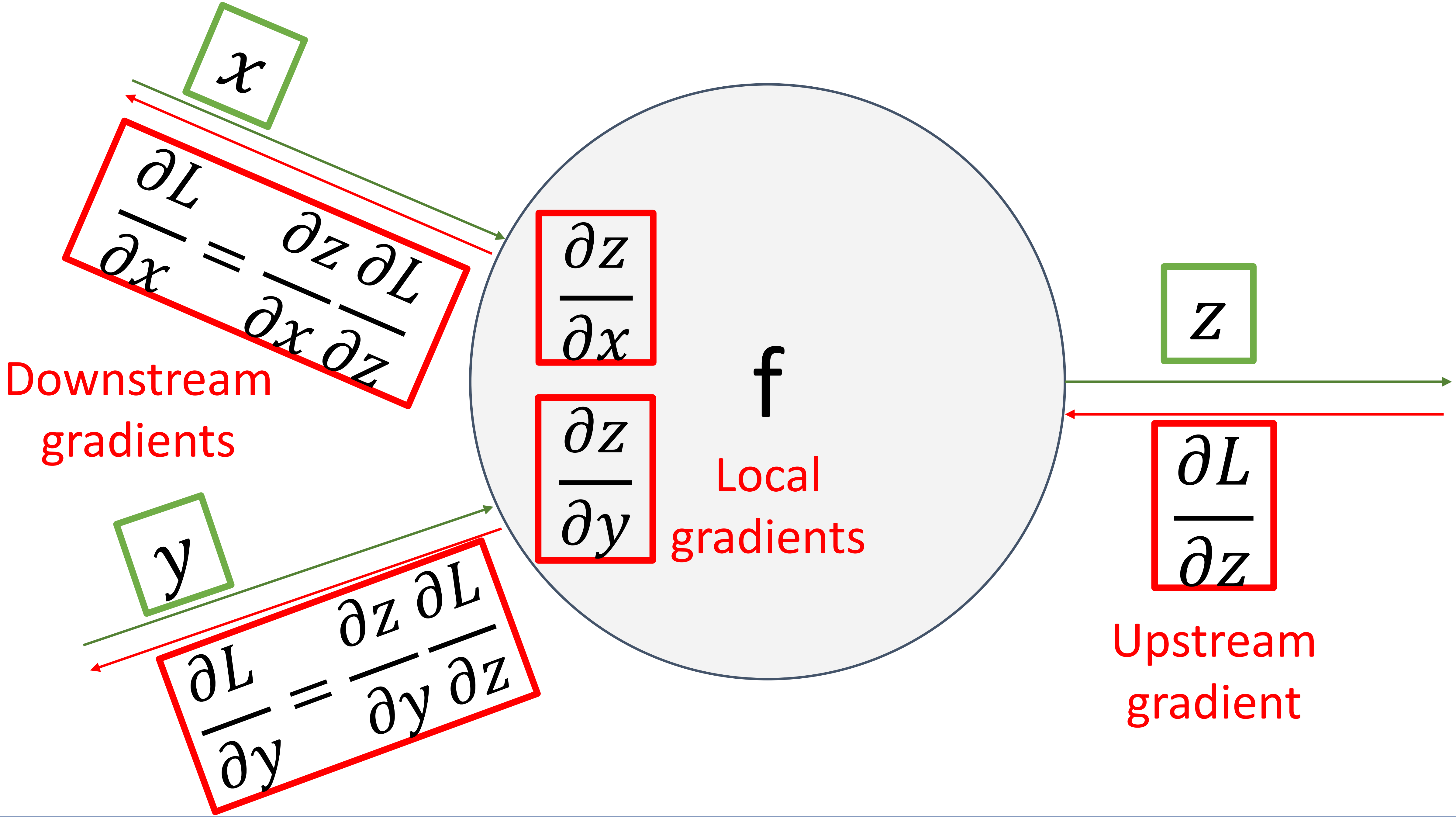

x
y
f
z



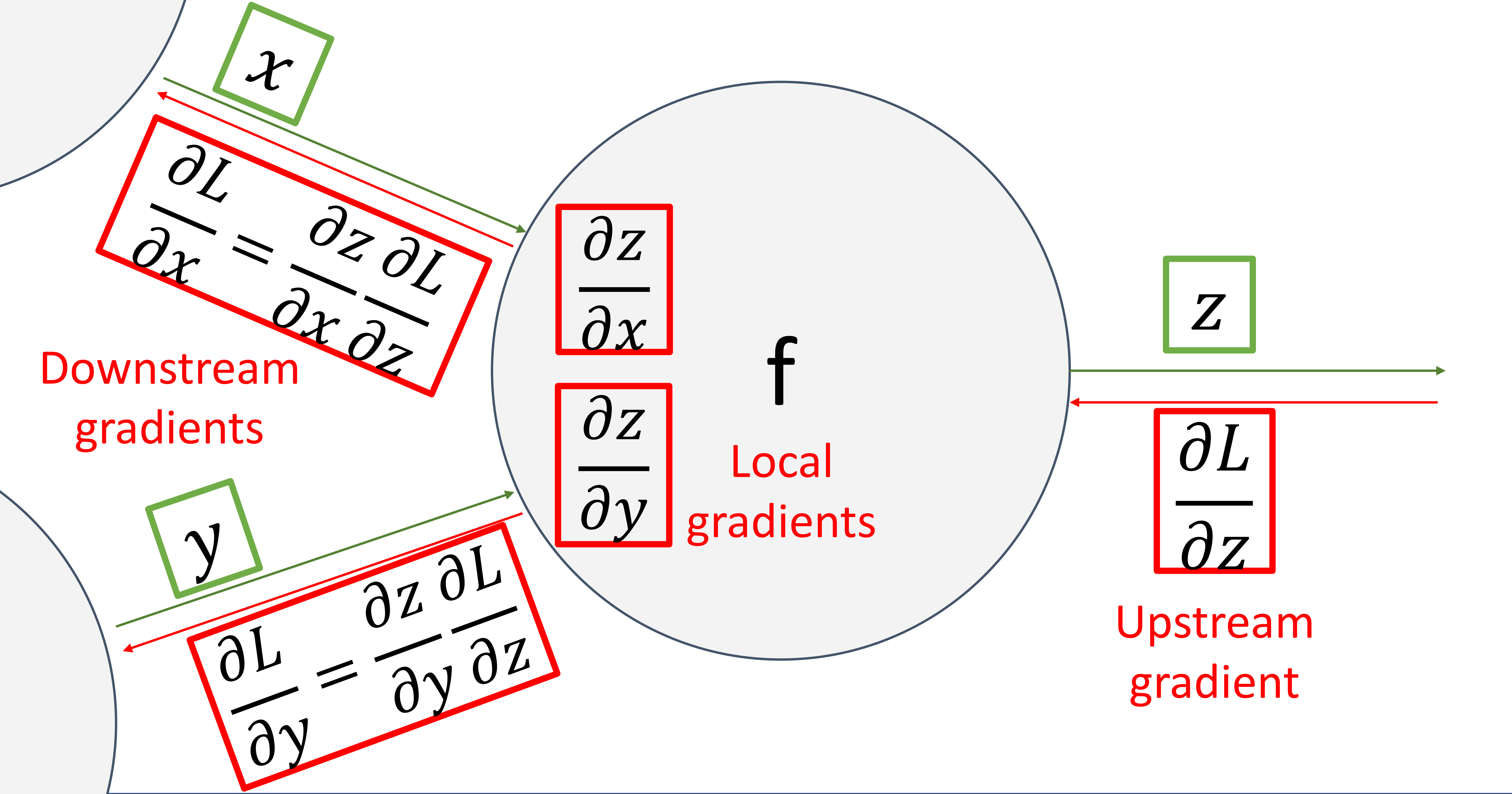
x
y
f
z
$\frac{\partial L}{\partial z}$
Upstream gradient



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

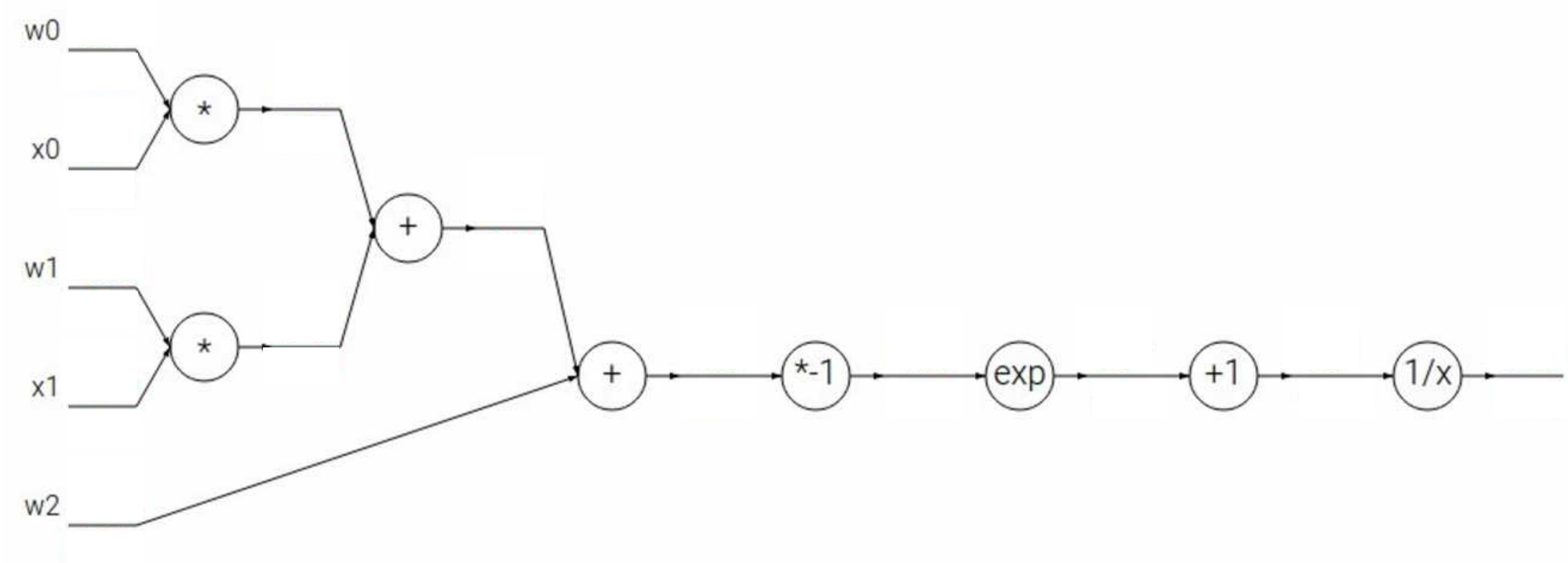


x
y
z
f
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
Downstream gradients
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
Local gradients
$\frac{\partial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



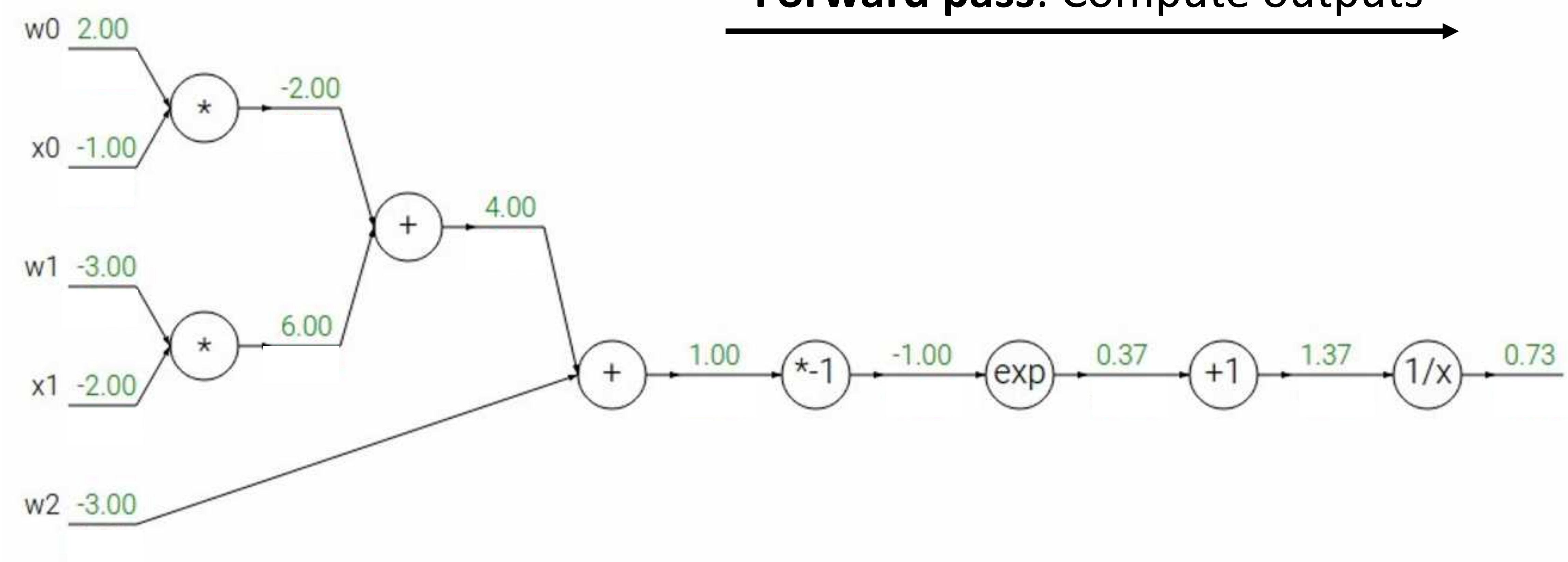
x
y
z
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
Downstream gradients
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
$\frac{\partial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

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



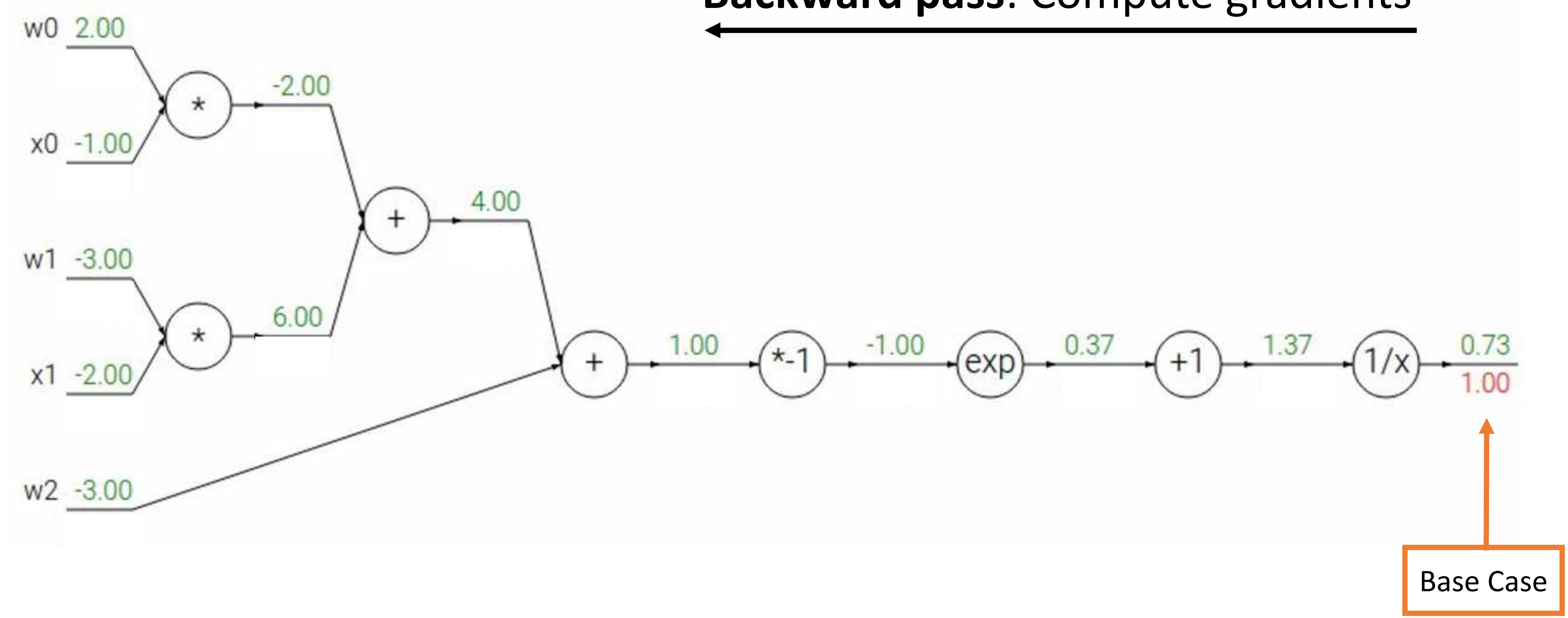
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Forward pass: Compute outputs



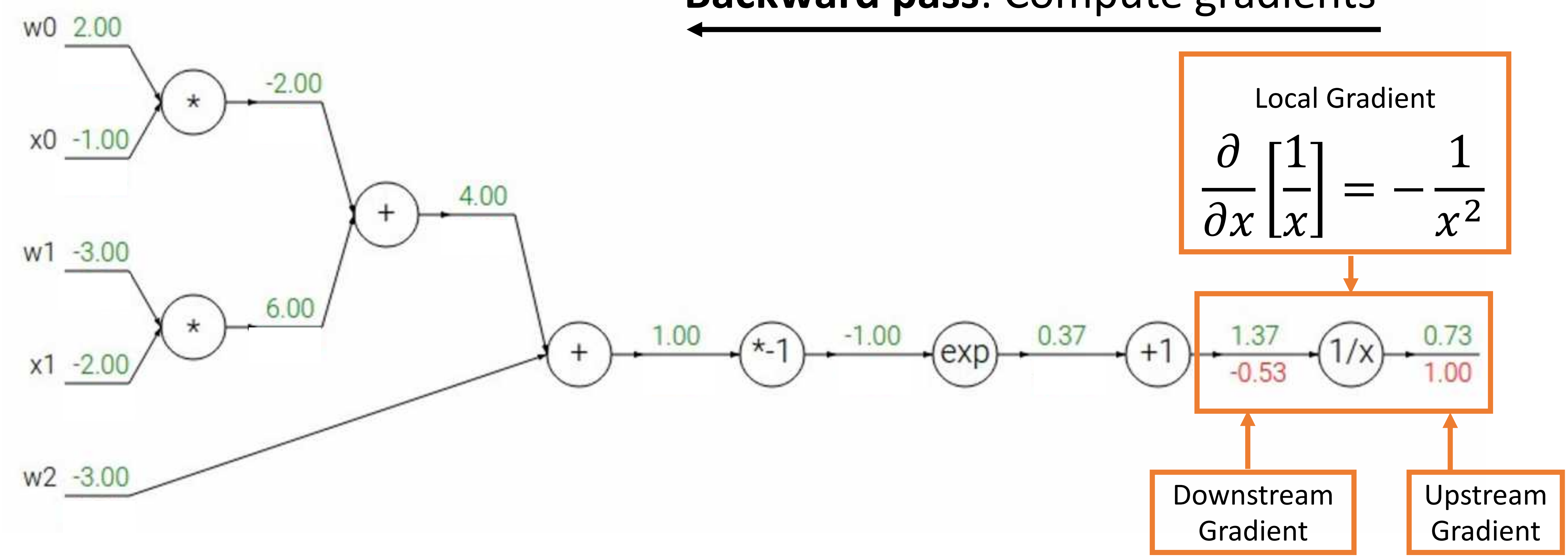
Another Example $f(x,w) = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

Backward pass: Compute gradients



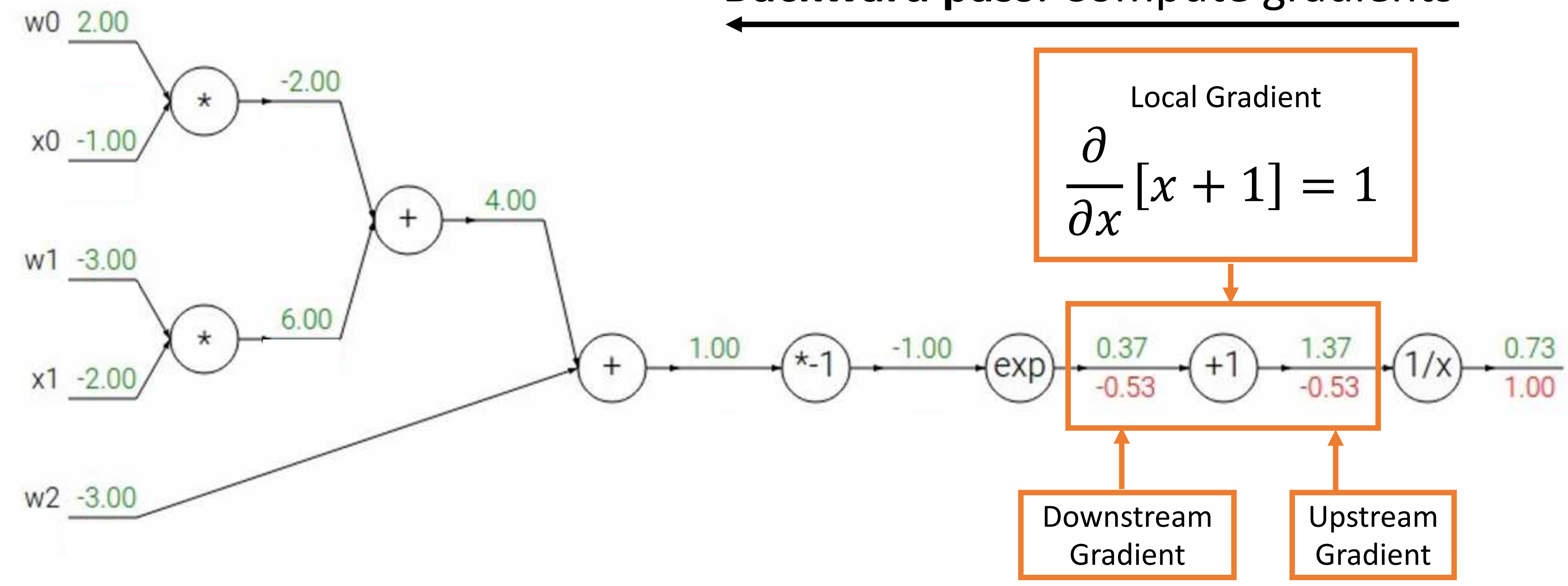
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



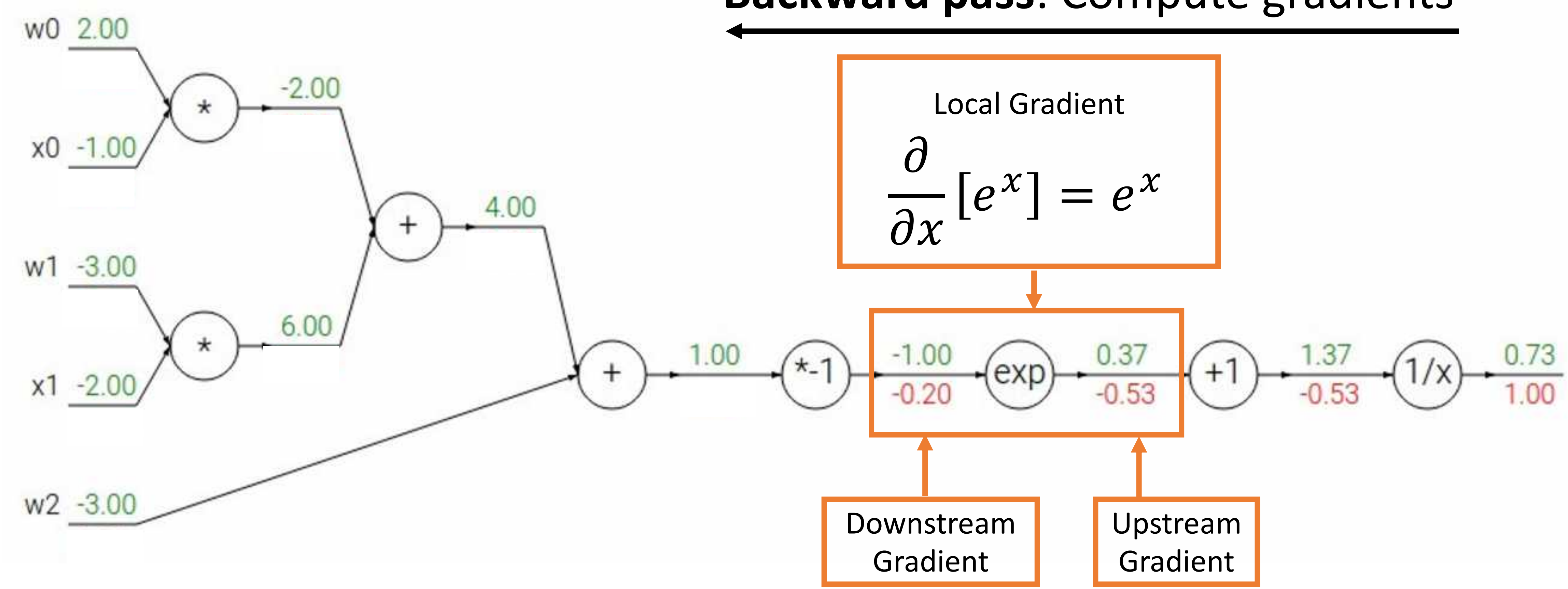
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



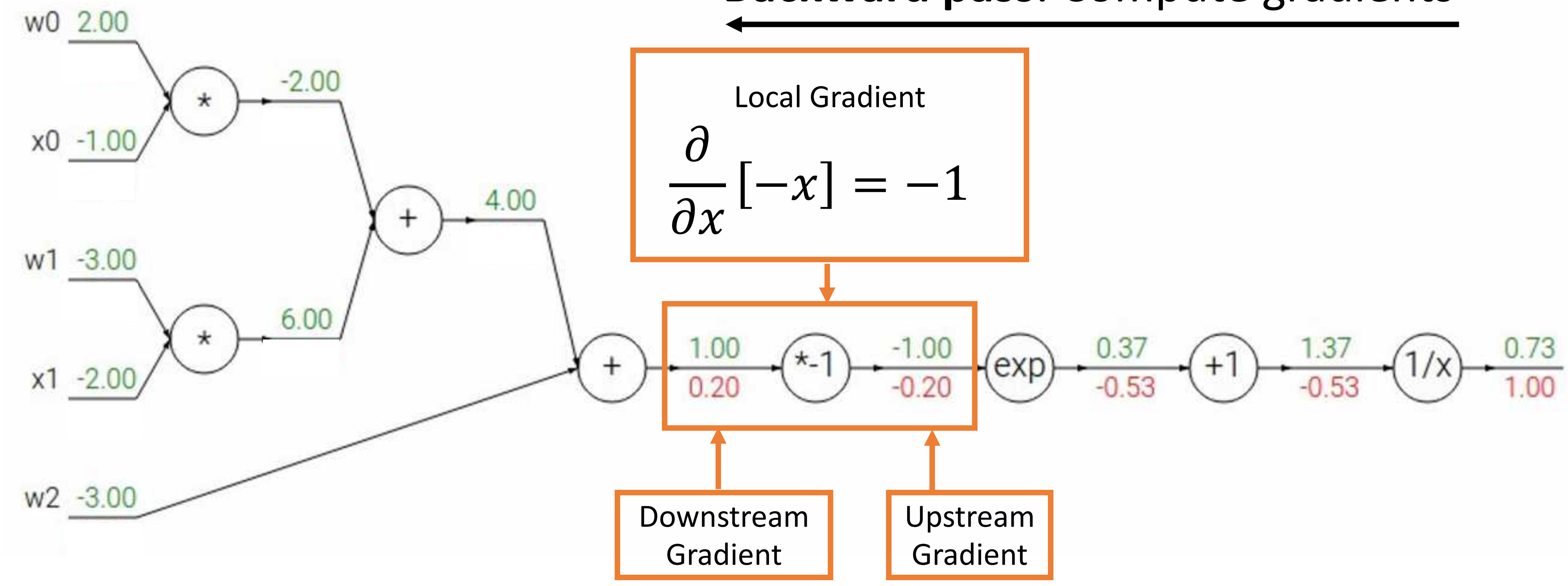
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



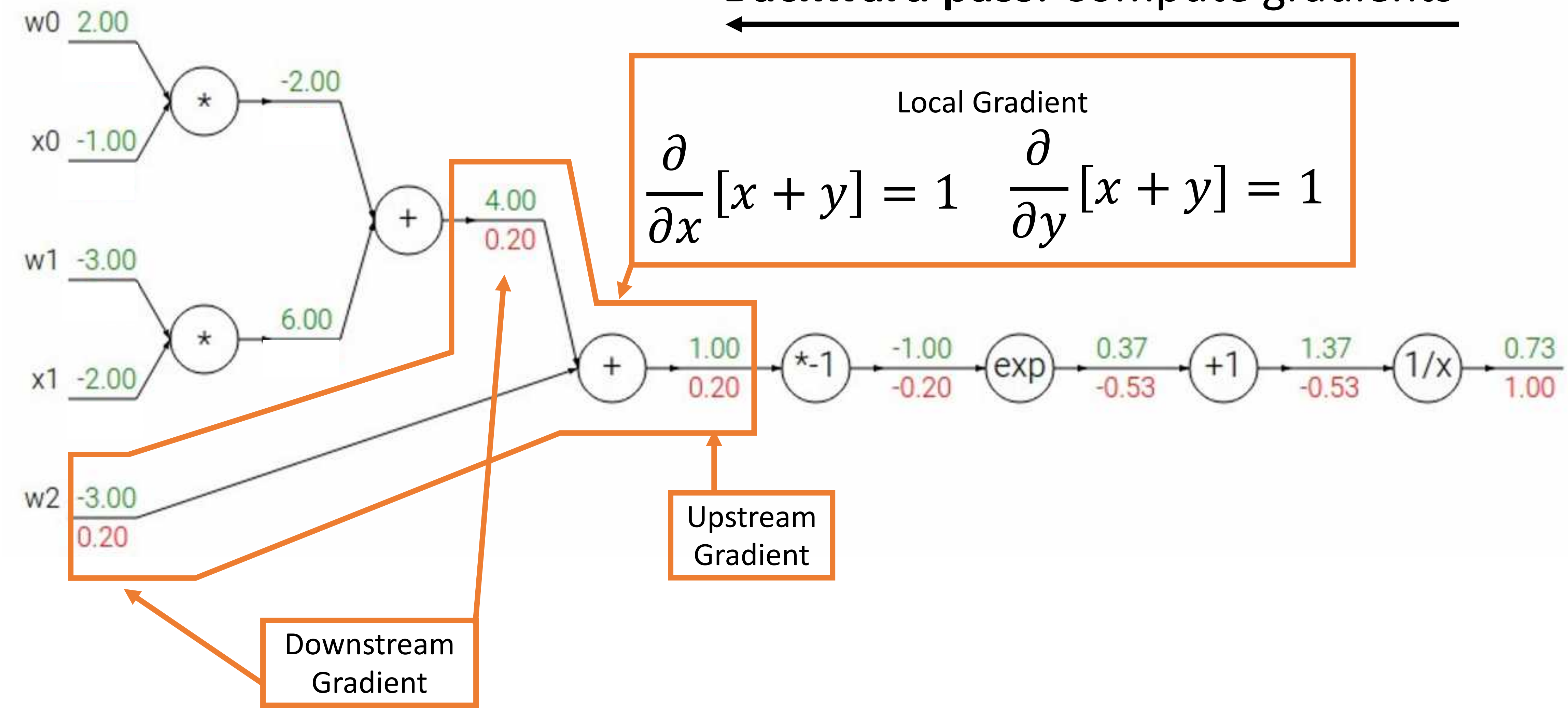
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



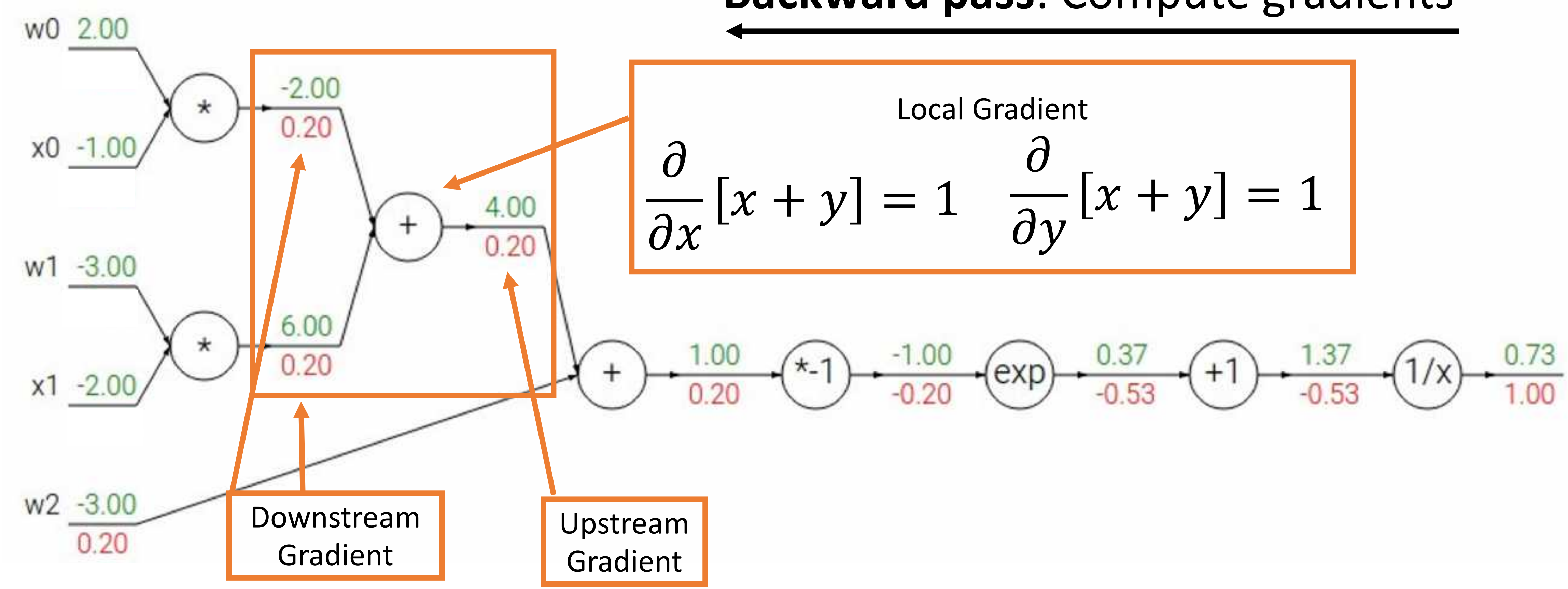
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

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



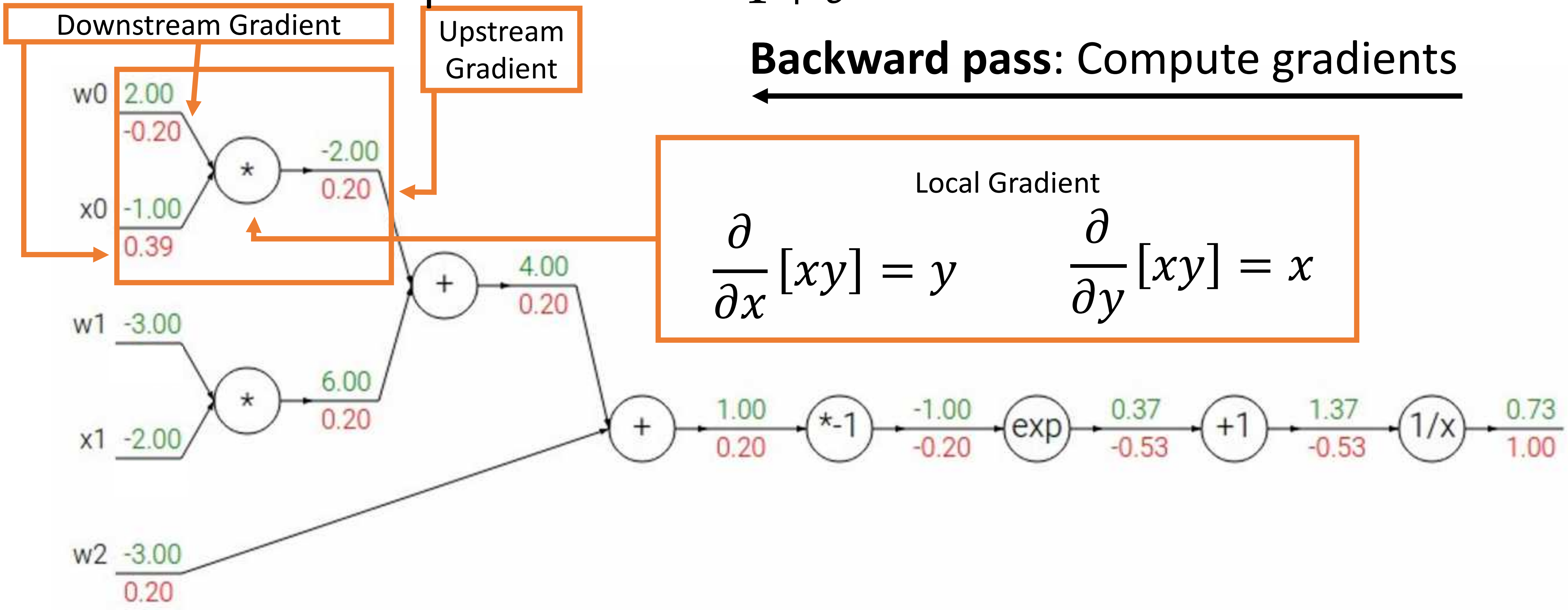
Local Gradient

$$\frac{\partial}{\partial x}[x+y] = 1 \quad \frac{\partial}{\partial y}[x+y] = 1$$

Another Example

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

Backward pass: Compute gradients



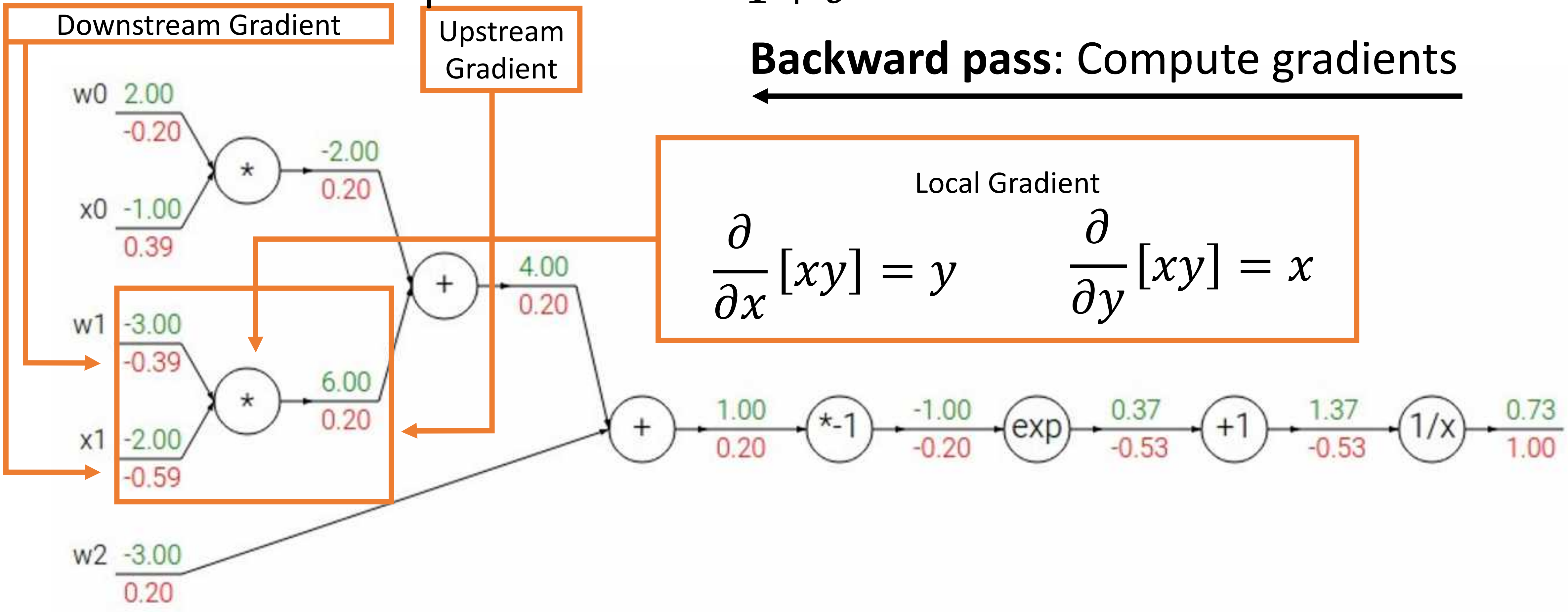
Local Gradient

$$\frac{\partial}{\partial x}[xy] = y \qquad \frac{\partial}{\partial y}[xy] = x$$

Another Example

$$f(x,w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

Backward pass: Compute gradients



Local Gradient

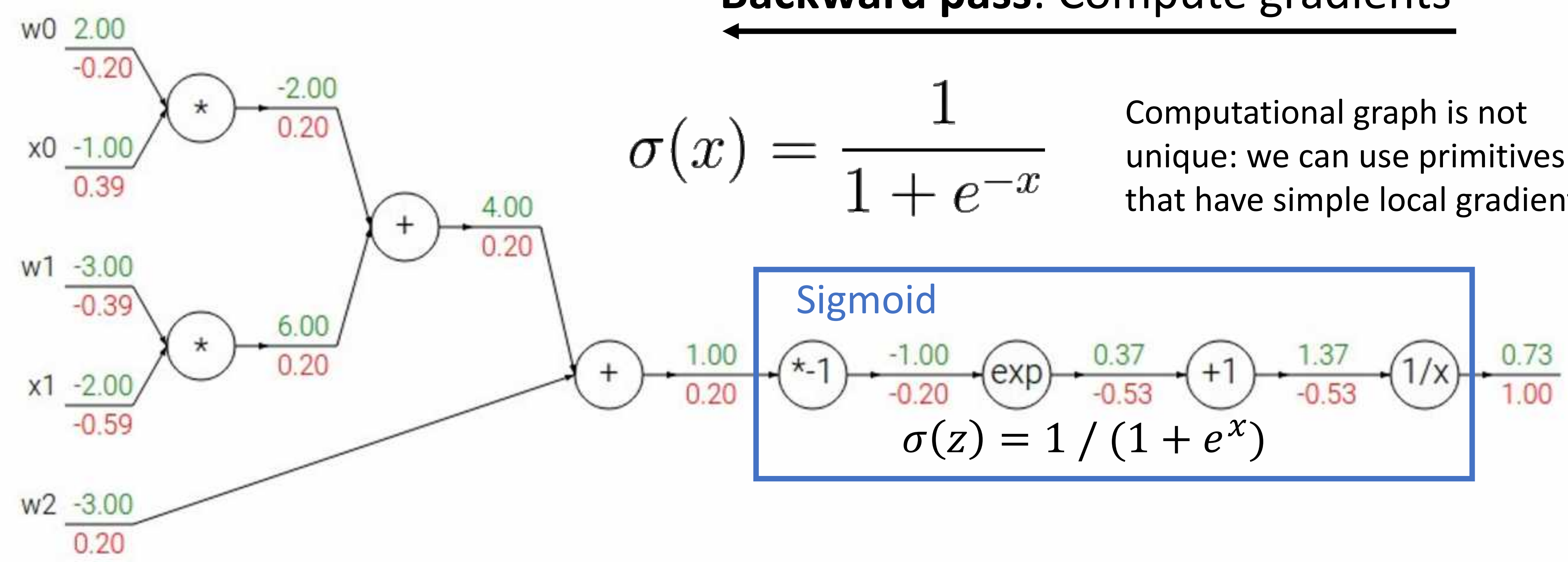
$$\frac{\partial}{\partial x}[xy] = y \qquad \frac{\partial}{\partial y}[xy] = x$$

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



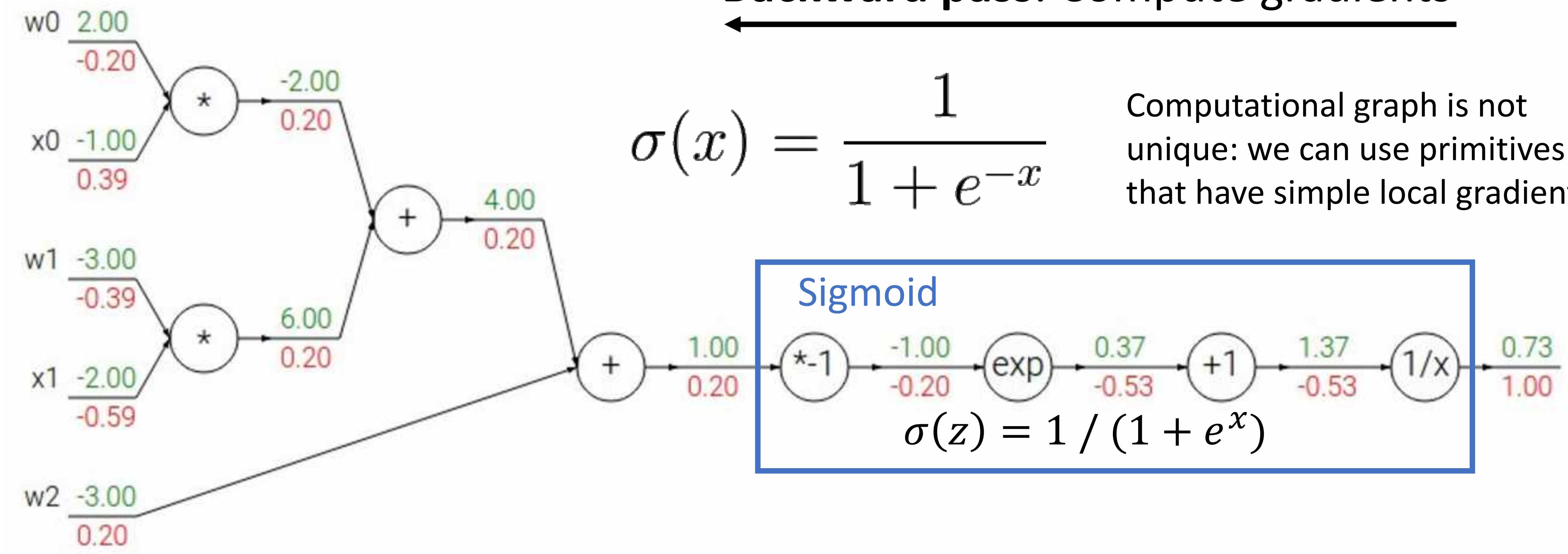
$\sigma(z) = 1 \;/\; (1+e^{x})$

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

Backward pass: Compute gradients

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

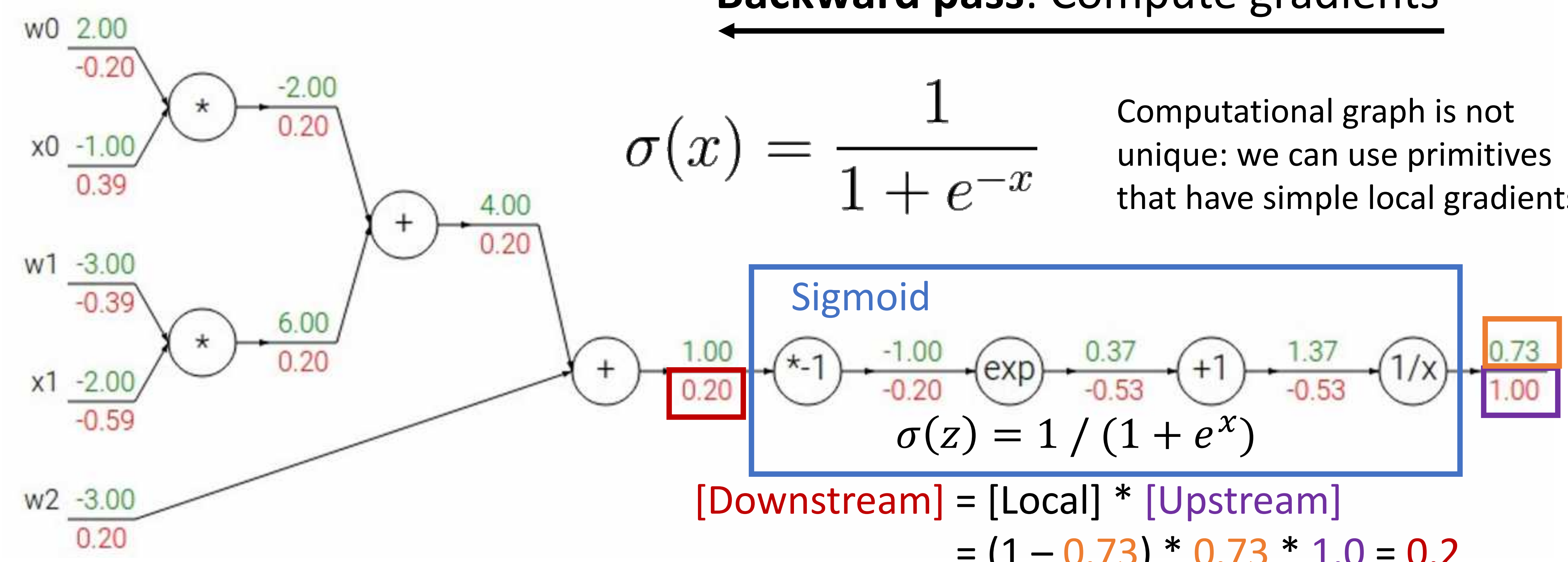
Computational graph is not unique: we can use primitives that have simple local gradients



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

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

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

$\sigma(z) = 1 \,/\, (1+e^{x})$

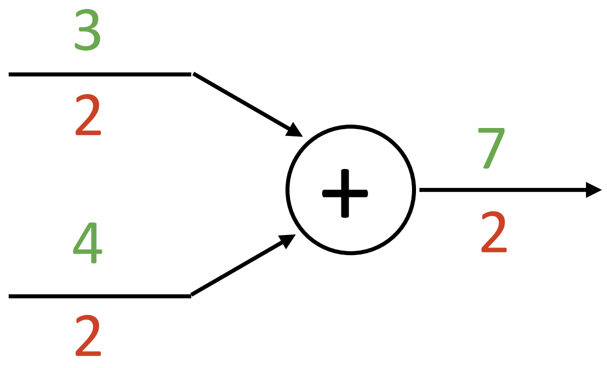
[Downstream] = [Local] * [Upstream]

= (1 – 0.73) * 0.73 * 1.0 = 0.2

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

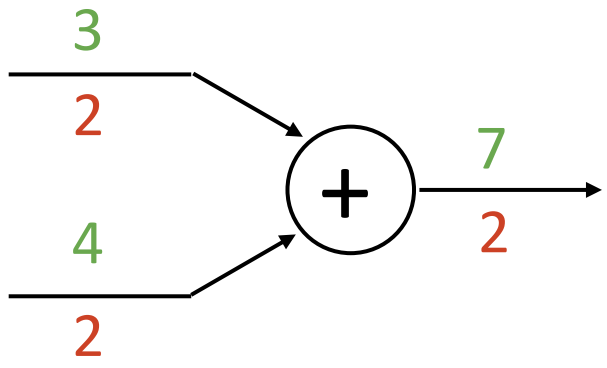
Patterns in Gradient Flow

add gate: gradient distributor

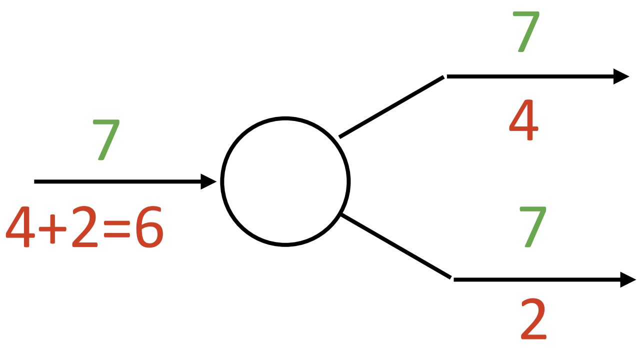


Patterns in Gradient Flow

add gate: gradient distributor

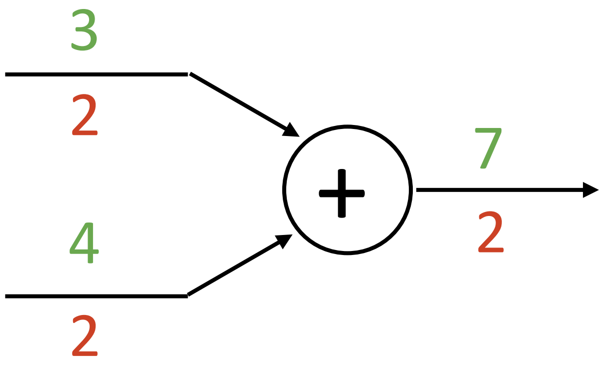


copy gate: gradient adder

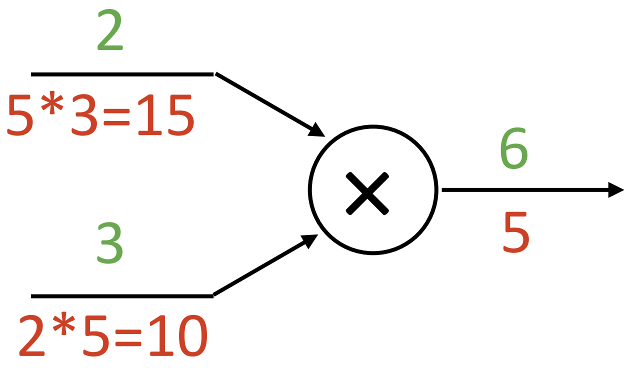


Patterns in Gradient Flow

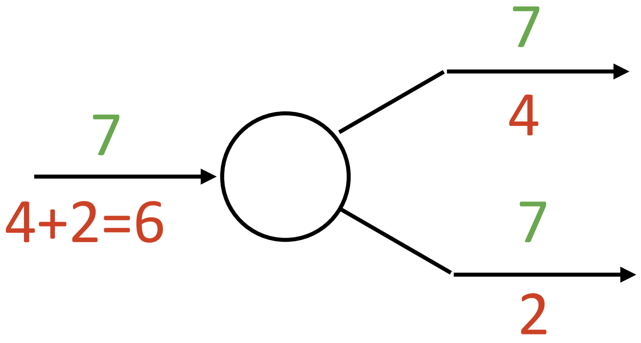
add gate: gradient distributor



mul gate: “swap multiplier”

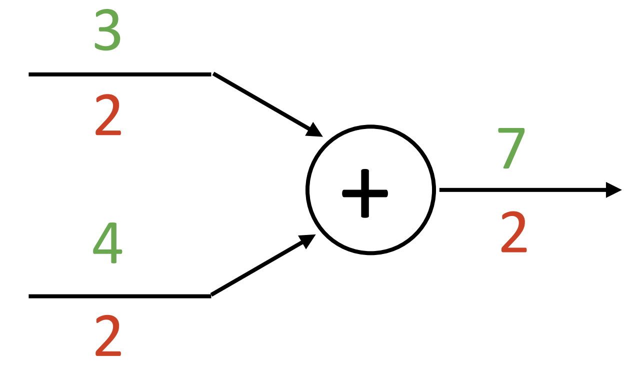


copy gate: gradient adder

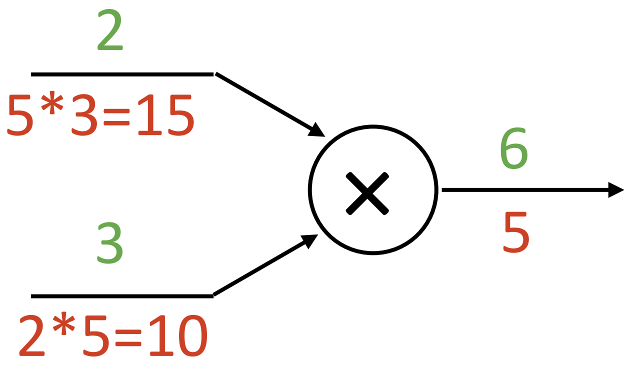


Patterns in Gradient Flow

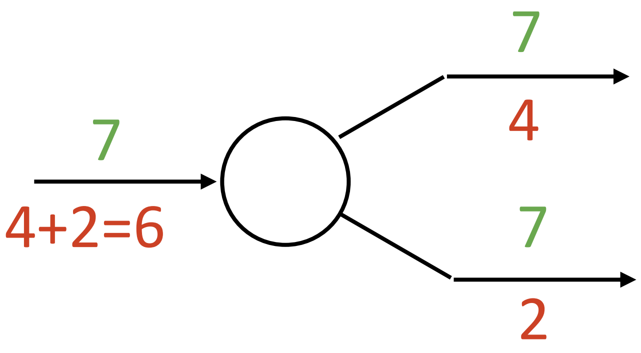
add gate: gradient distributor



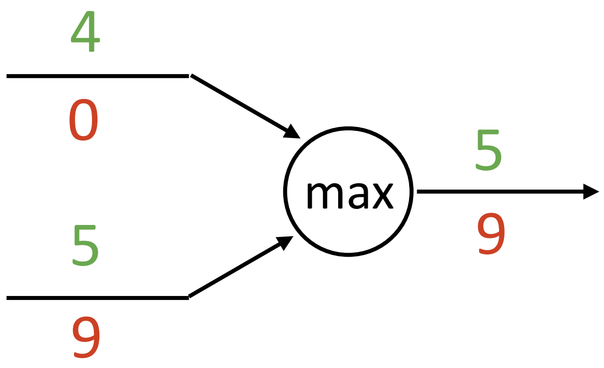
mul gate: "swap multiplier"



copy gate: gradient adder



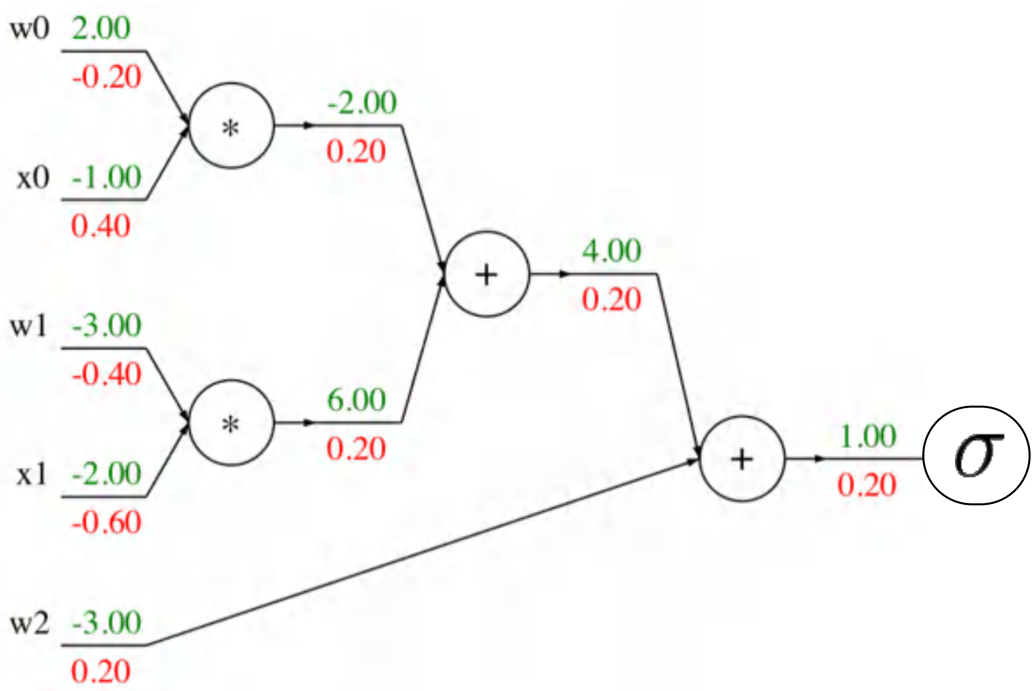
max gate: gradient router



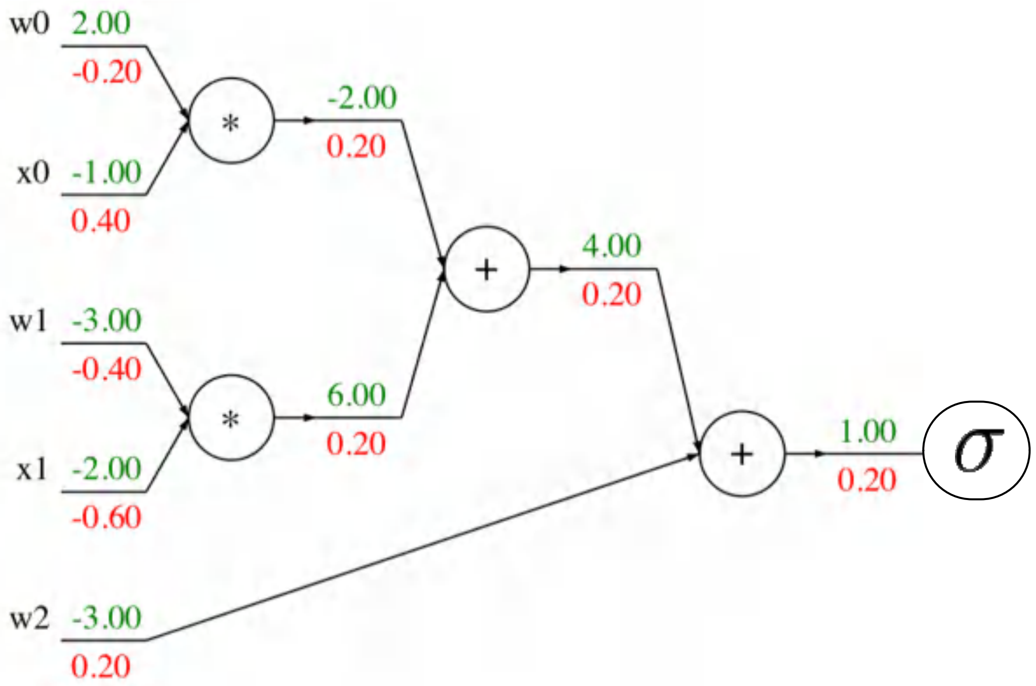
Backprop Implementation: ”Flat” gradient code:

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:



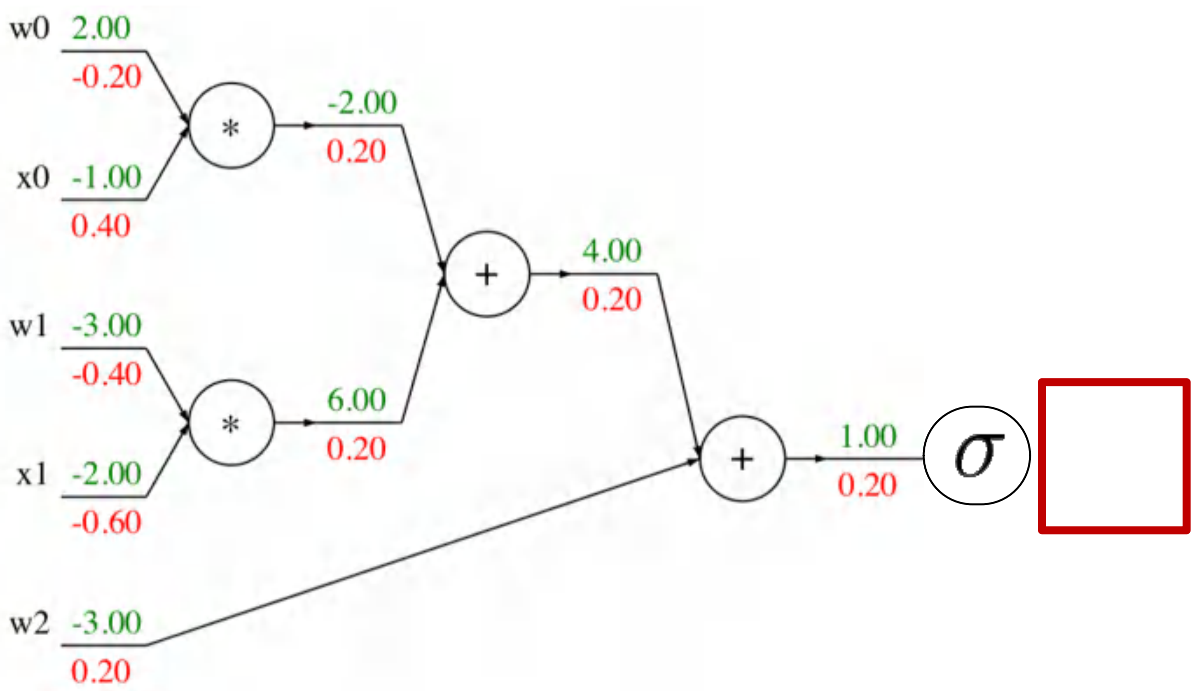
Forward pass: Compute output

```
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:



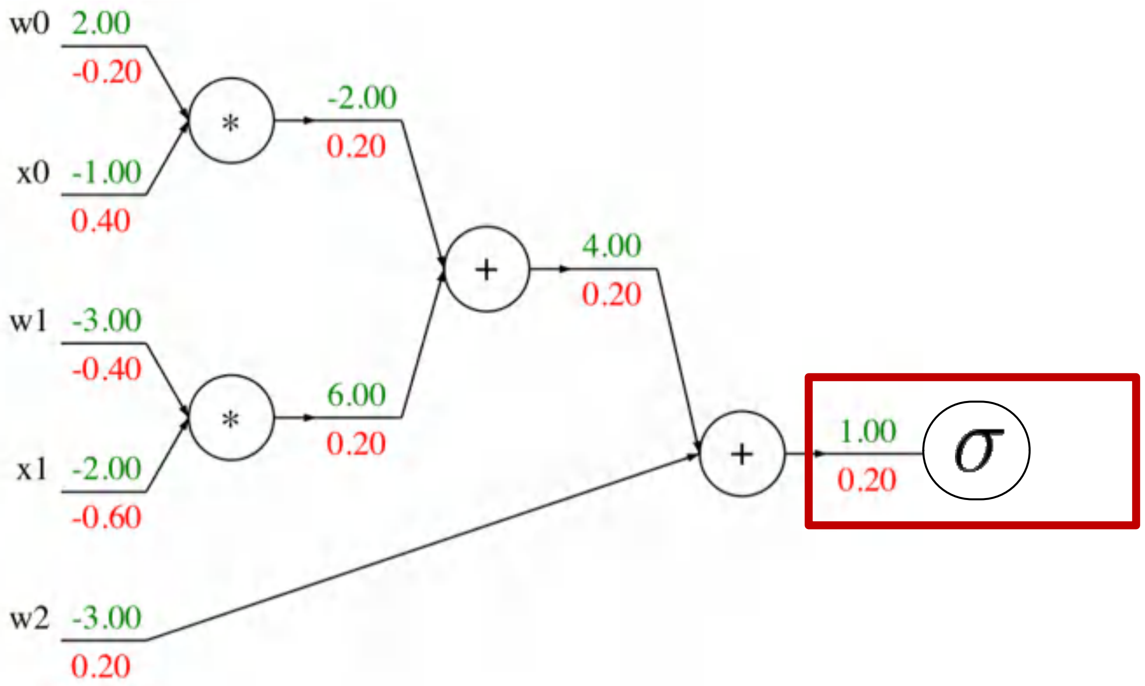
Forward pass: Compute output

Base case

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

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

Backprop Implementation: "Flat" gradient code:



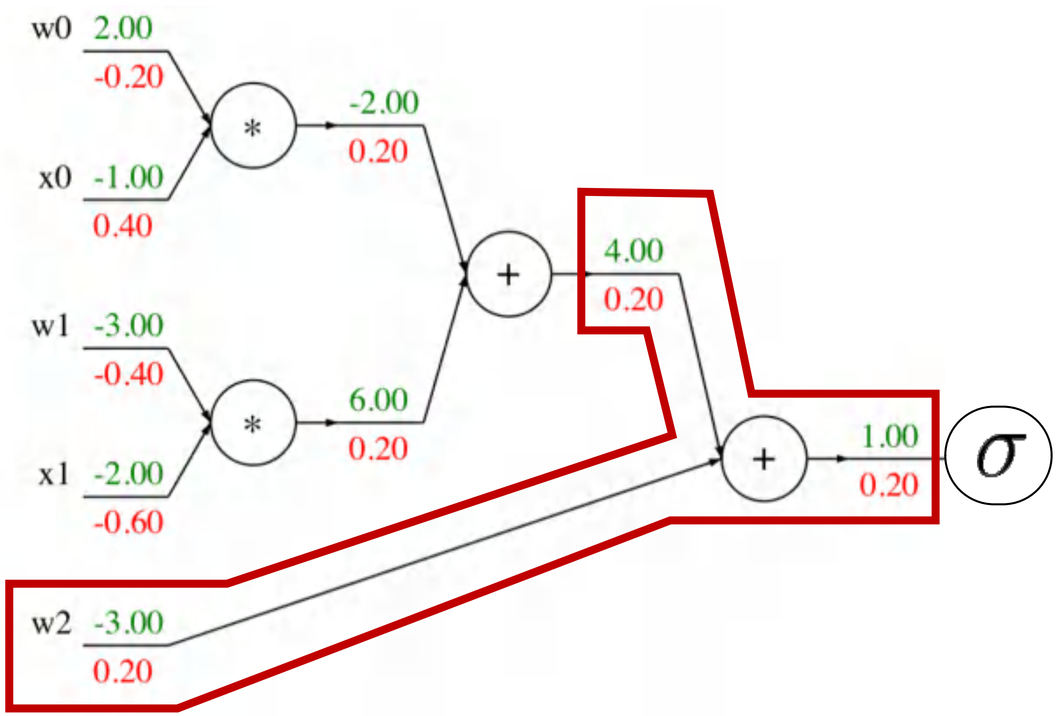
Forward pass: Compute output

```
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
```

Sigmoid

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
```

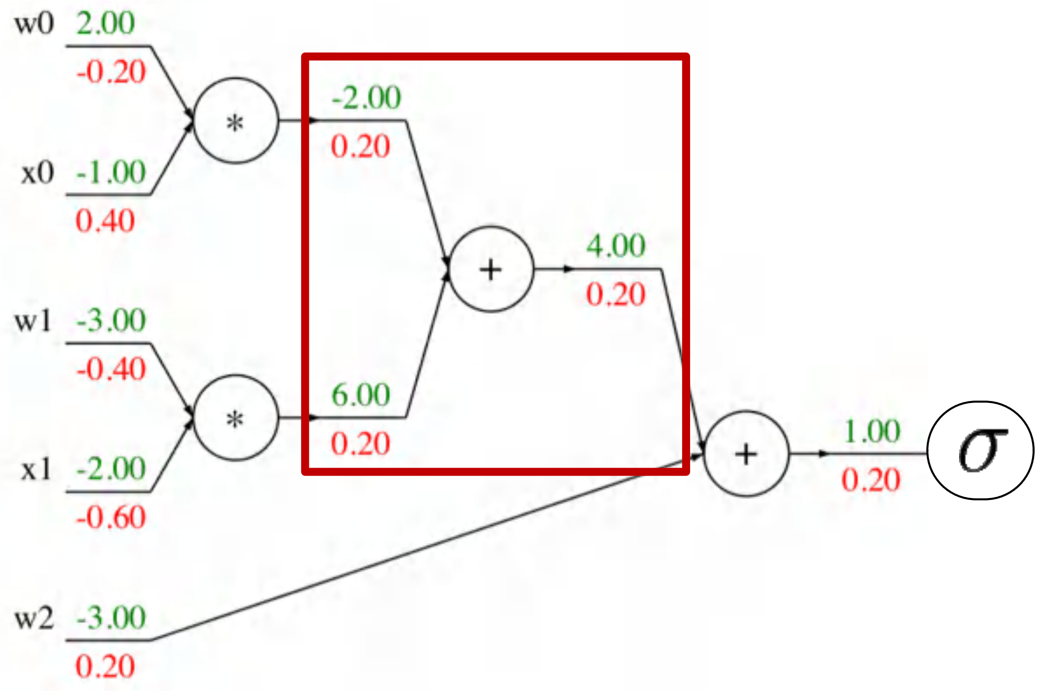
Backprop Implementation: "Flat" gradient code:

Forward pass:
Compute output

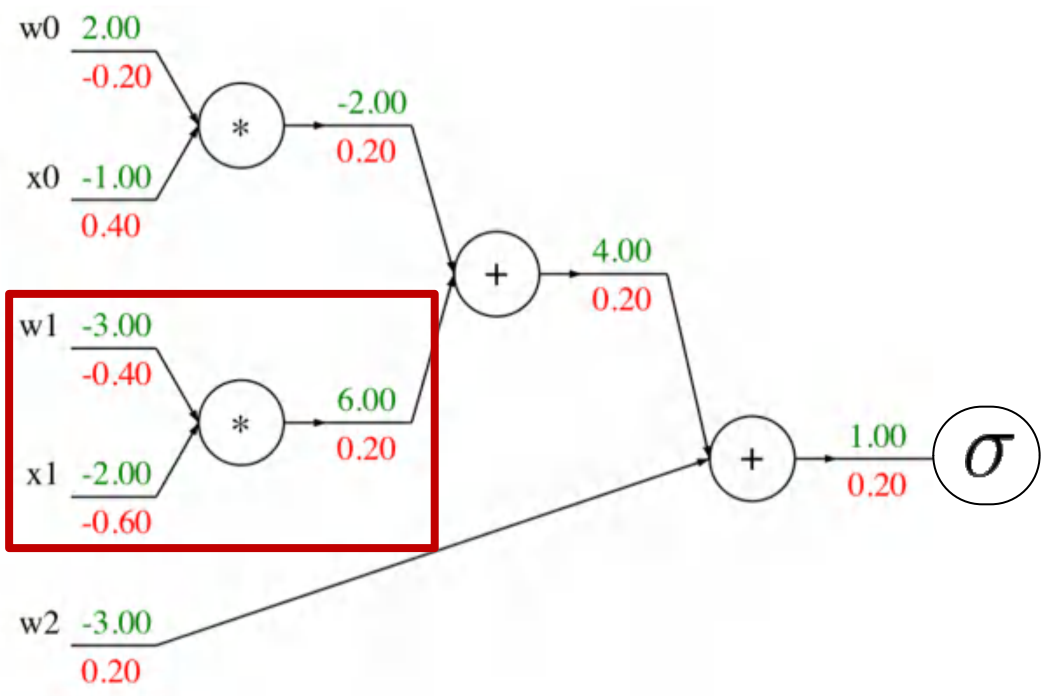
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  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
```

Add



Backprop Implementation: ”Flat” gradient code:



Forward pass: Compute output

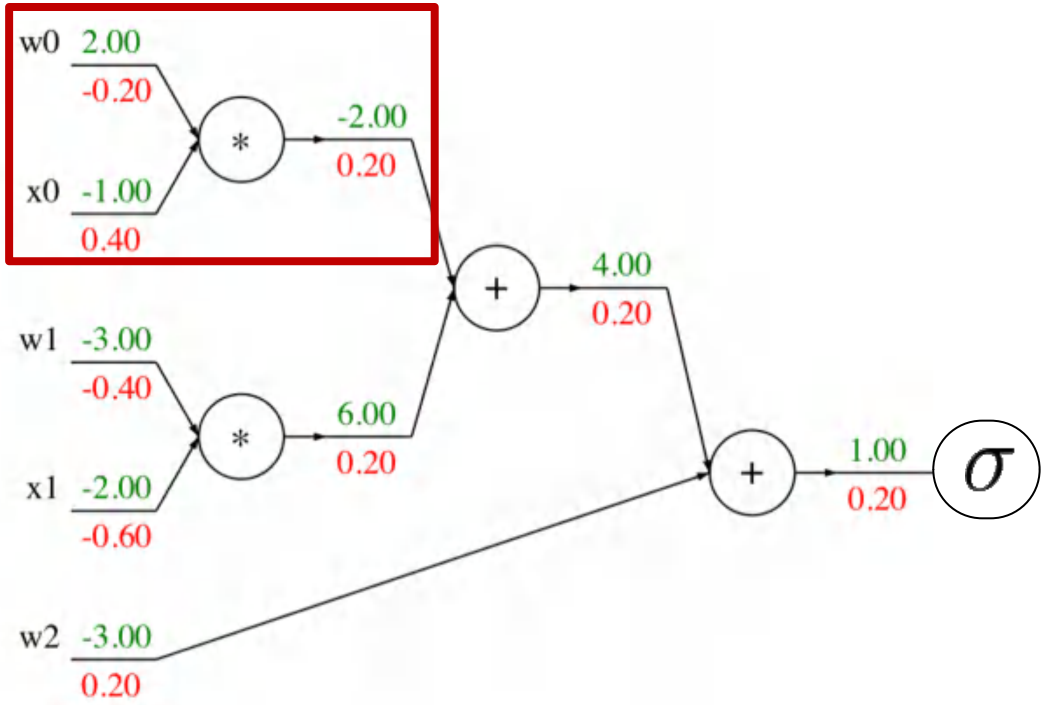
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

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

Multiply

Backprop Implementation: "Flat" gradient code:

Forward pass:
Compute output




```
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
```

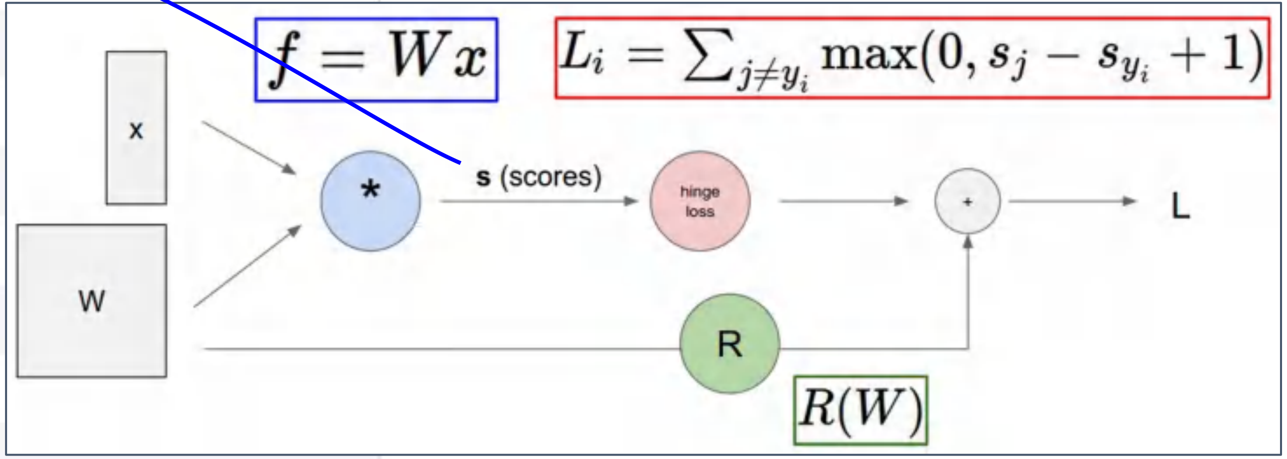
Multiply

“Flat” Backprop: Do this for Assignment 2!

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

E.g. for the SVM:

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



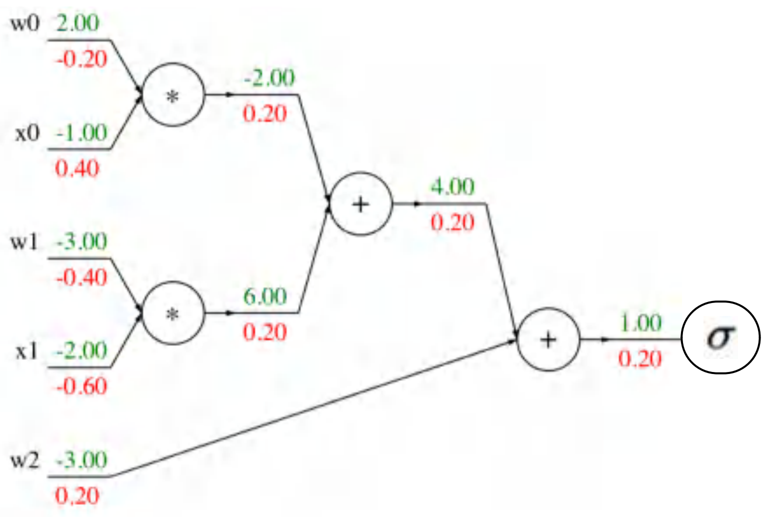
“Flat” Backprop: Do this for Assignment 2!

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

E.g. for two-layer neural net:

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

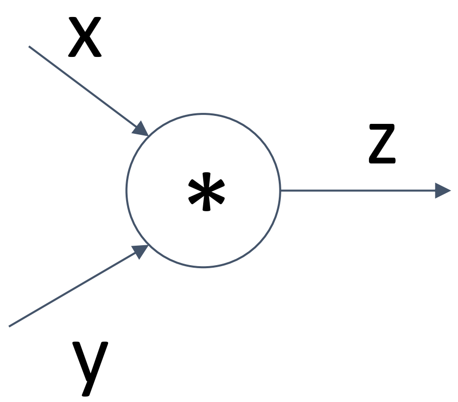
Backprop Implementation: Modular API



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

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

Example: PyTorch Autograd Functions



(x,y,z are scalars)

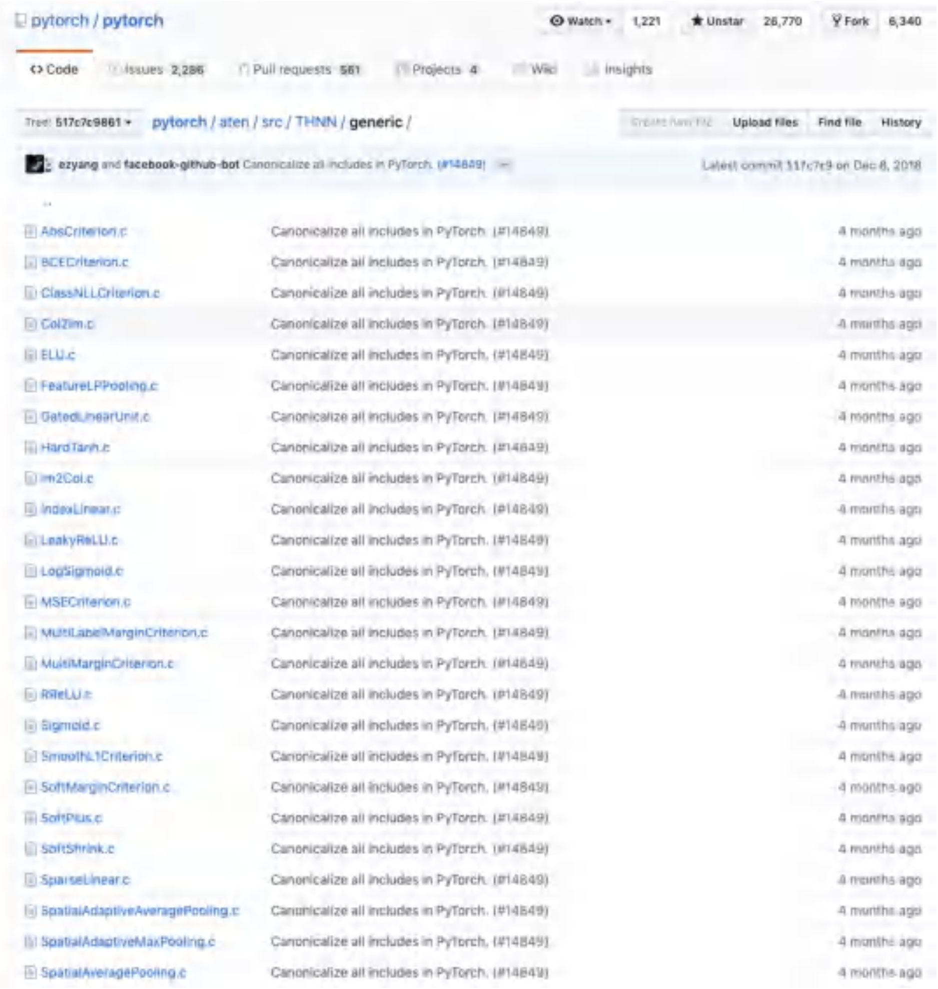
```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz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

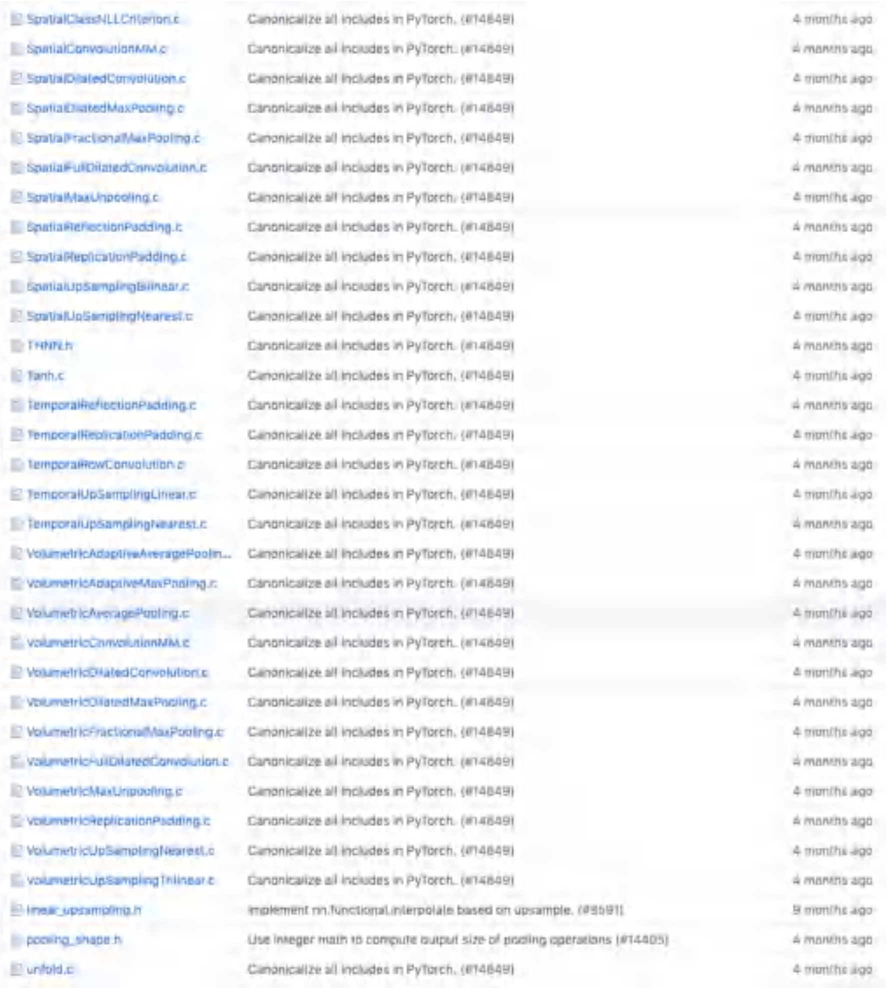
Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Example: PyTorch operators





PyTorch sigmoid layer

```c
#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

```c
#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

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

Source

PyTorch sigmoid layer

```c
#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

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

```cpp
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...

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

Source

PyTorch sigmoid layer

```c
#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

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**:

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

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

Recap: Vector Derivatives

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

Regular derivative:

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

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

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

Derivative is **Gradient**:

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

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

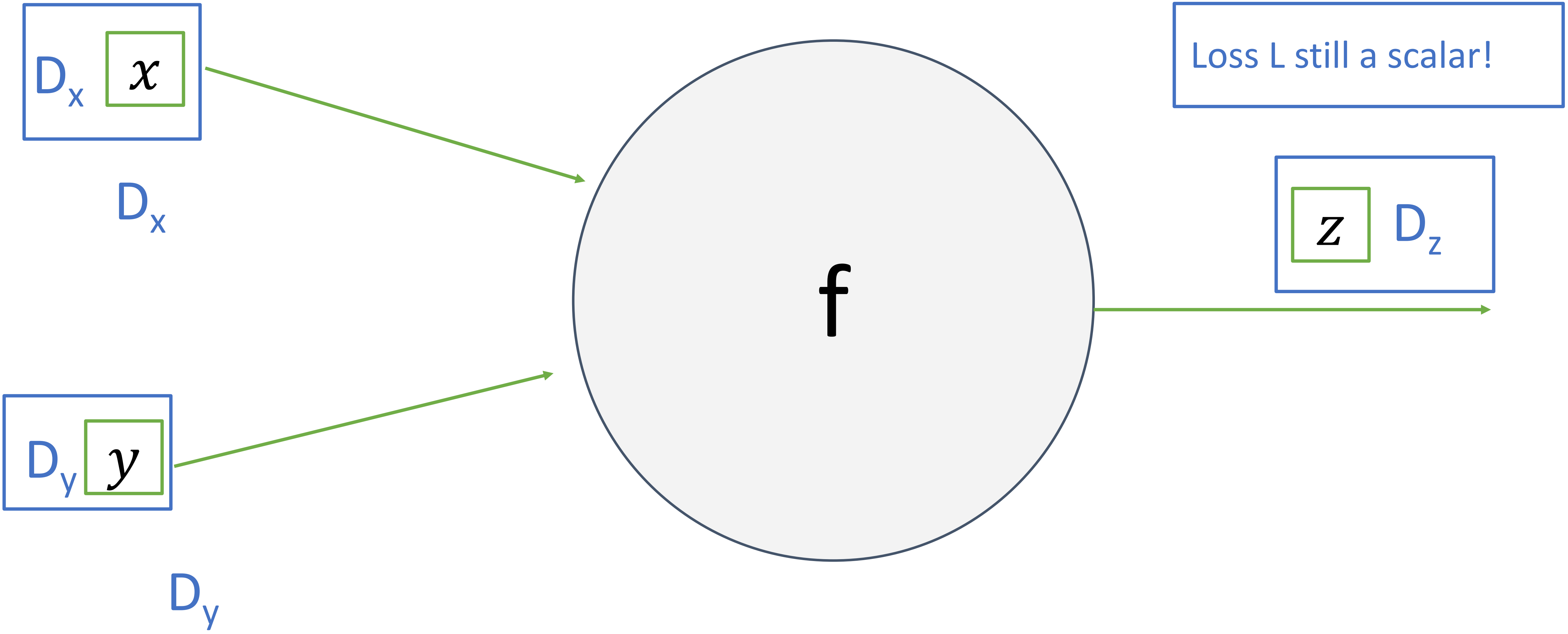
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

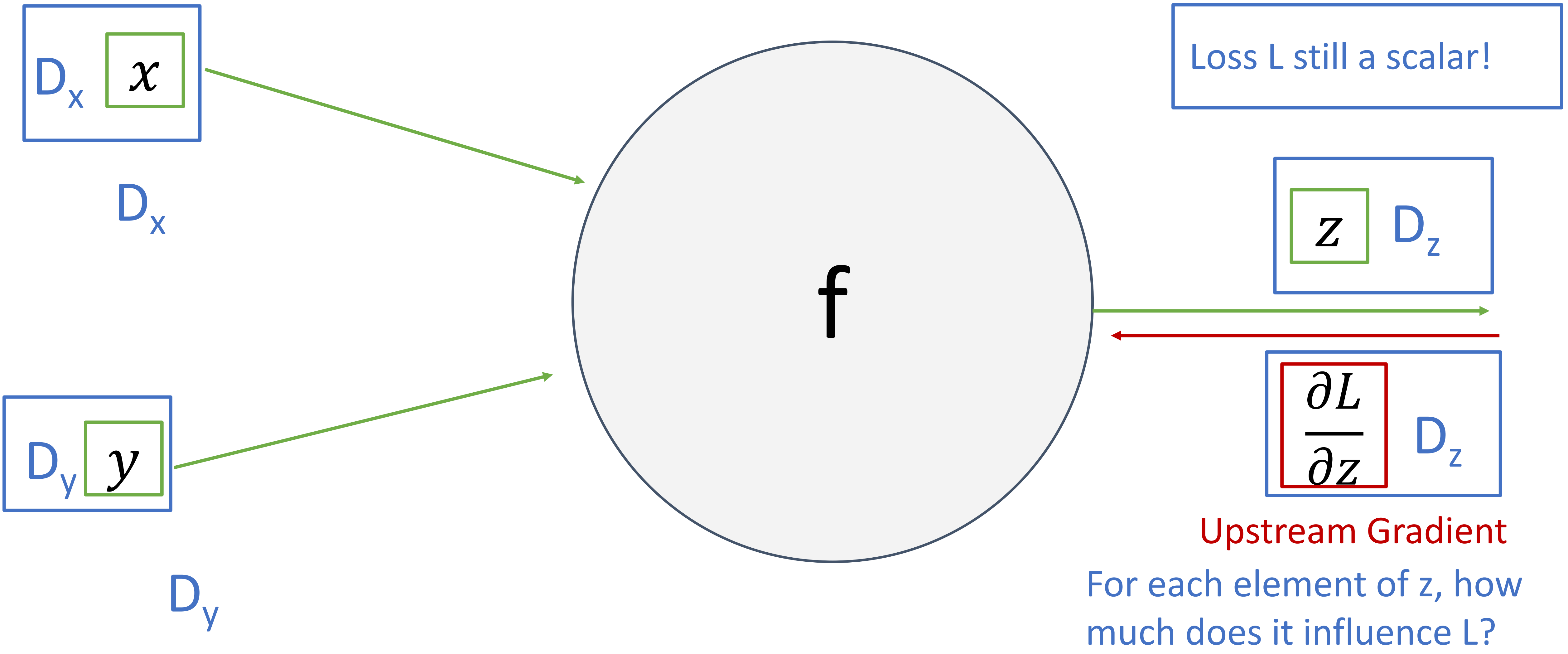
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

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

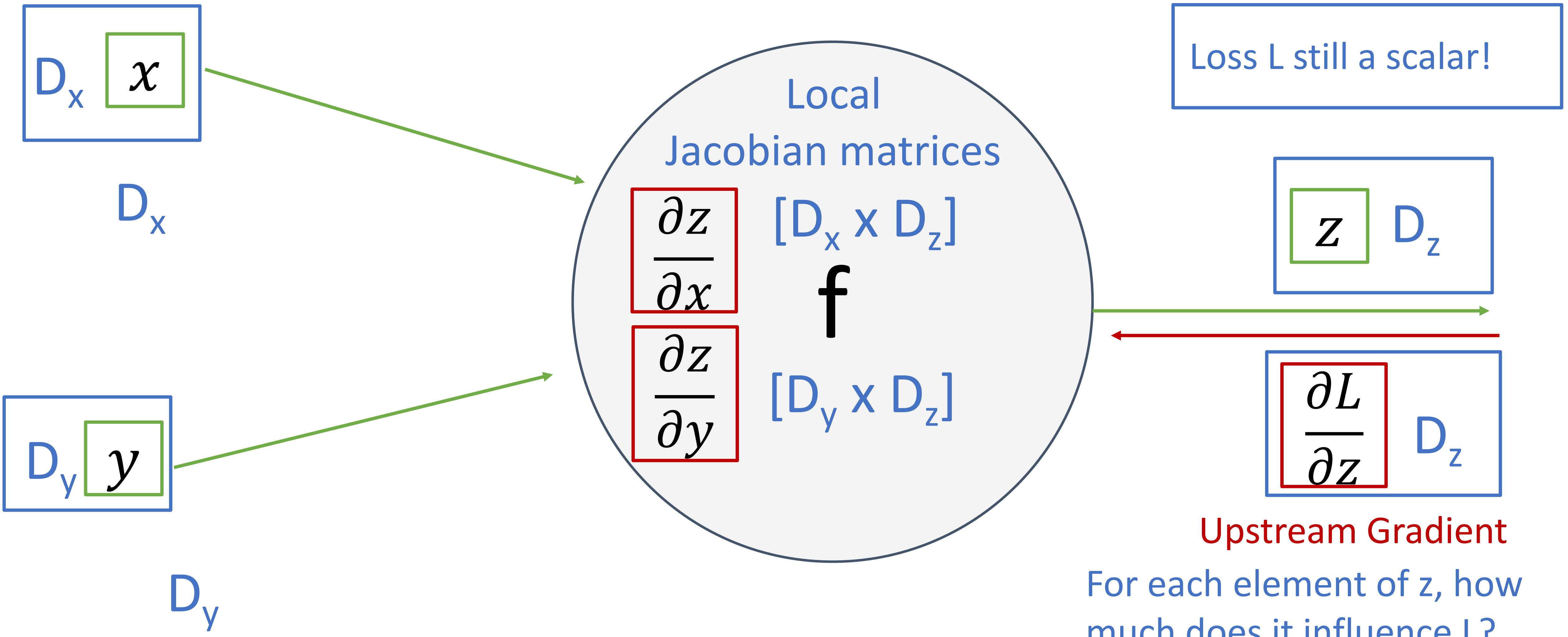
Backprop with Vectors



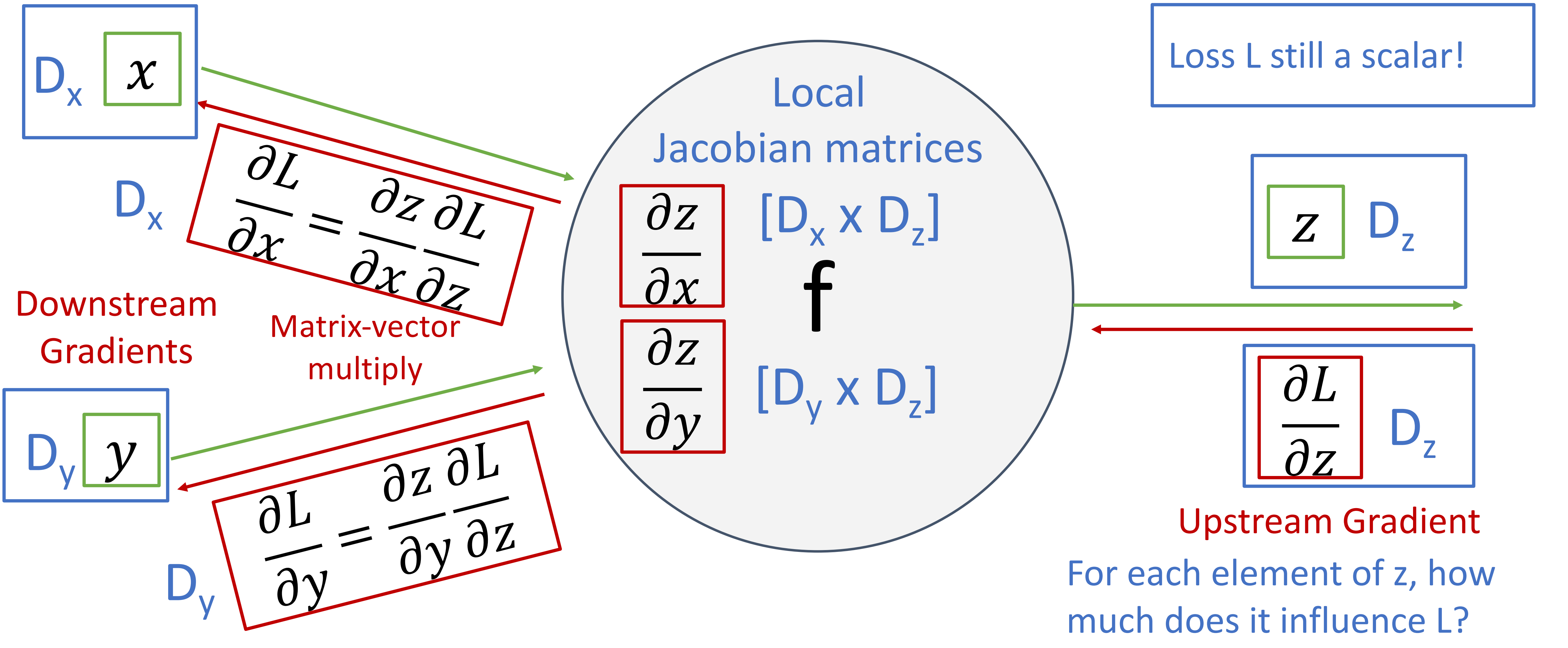
Backprop with Vectors



Backprop with Vectors



Backprop with Vectors



Backprop with Vectors

4D input x:

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

4D output y:

Backprop with Vectors

4D input x:

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

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

4D output y:

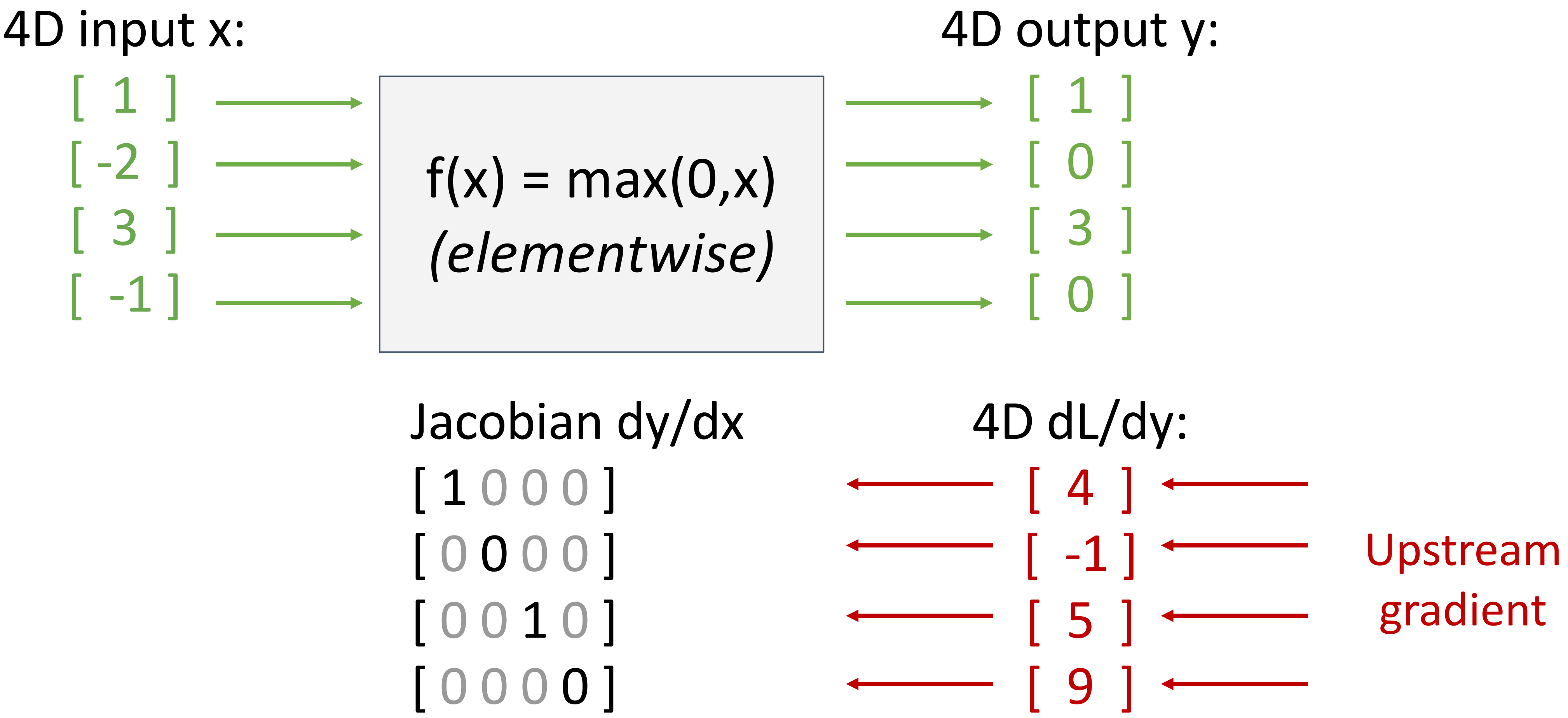
[1]
[0]
[3]
[0]

4D dL/dy:

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

Upstream gradient

Backprop with Vectors



Backprop with Vectors

4D input x:

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

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

4D output y:

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

[dy/dx] [dL/dy]

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

4D dL/dy:

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

Upstream gradient

Backprop with Vectors

4D input x:		4D output y:
[1]		[1]
[-2]	f(x) = max(0,x)	[0]
[3]	*(elementwise)*	[3]
[-1]		[0]

4D dL/dx:	[dy/dx] [dL/dy]	4D dL/dy:	
[4]	[1 0 0 0] [4]	[4]	
[0]	[0 0 0 0] [-1]	[-1]	Upstream gradient
[5]	[0 0 1 0] [5]	[5]	
[0]	[0 0 0 0] [9]	[9]	

Backprop with Vectors

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

4D input x:

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

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

4D output y:

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

4D dL/dx:

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

[dy/dx] [dL/dy]

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

4D dL/dy:

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

Upstream gradient

Backprop with Vectors

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

4D input x:

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

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

4D output y:

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

4D dL/dx:

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

[dy/dx] [dL/dy]

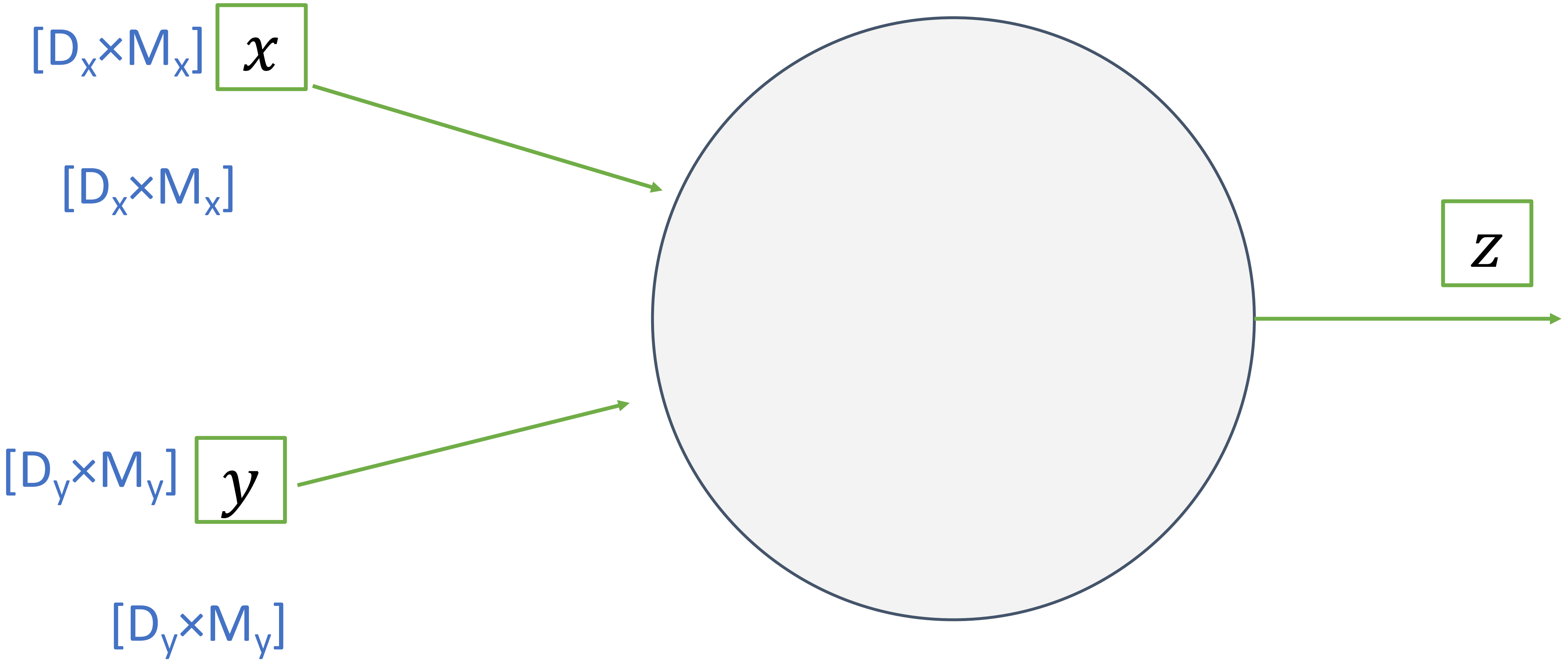
$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial y}\right)_i, & if\ x_i > 0 \\ 0, & otherwise \end{cases}$$

4D dL/dy:

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

Upstream gradient

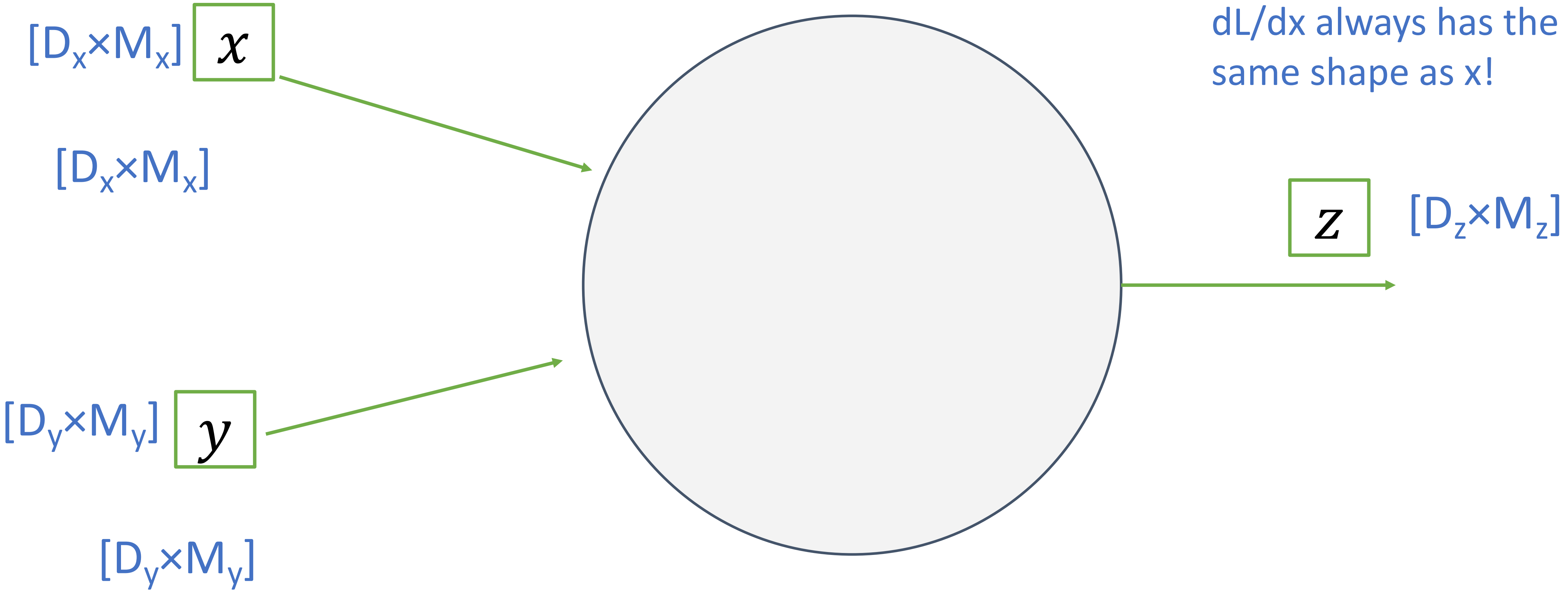
Backprop with Matrices (or Tensors):



Backprop with Matrices (or Tensors):

Loss L still a scalar!

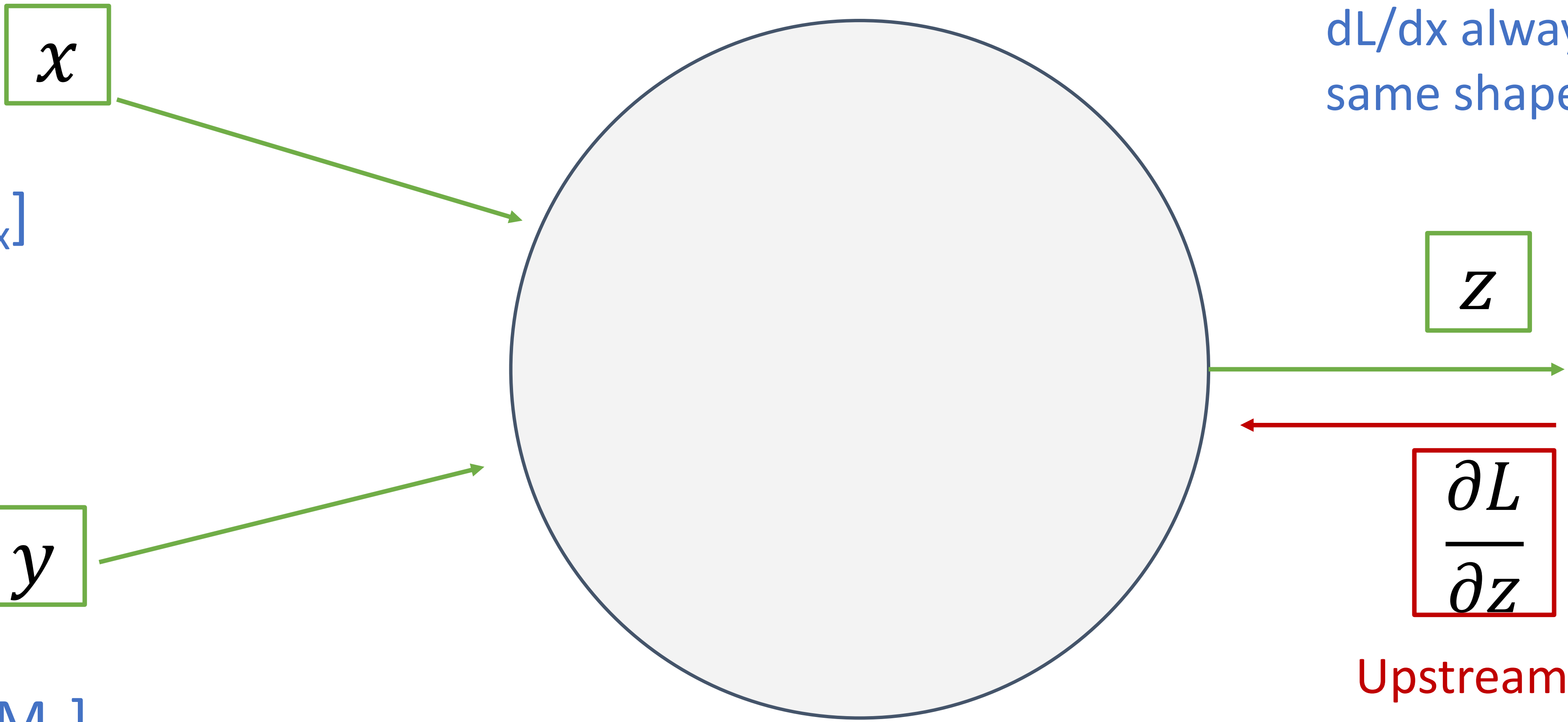
dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors):

Loss L still a scalar!

dL/dx always has the same shape as x!



$[D_x \times M_x]$

$[D_x \times M_x]$

$[D_y \times M_y]$

$[D_y \times M_y]$

$[D_z \times M_z]$

$[D_z \times M_z]$

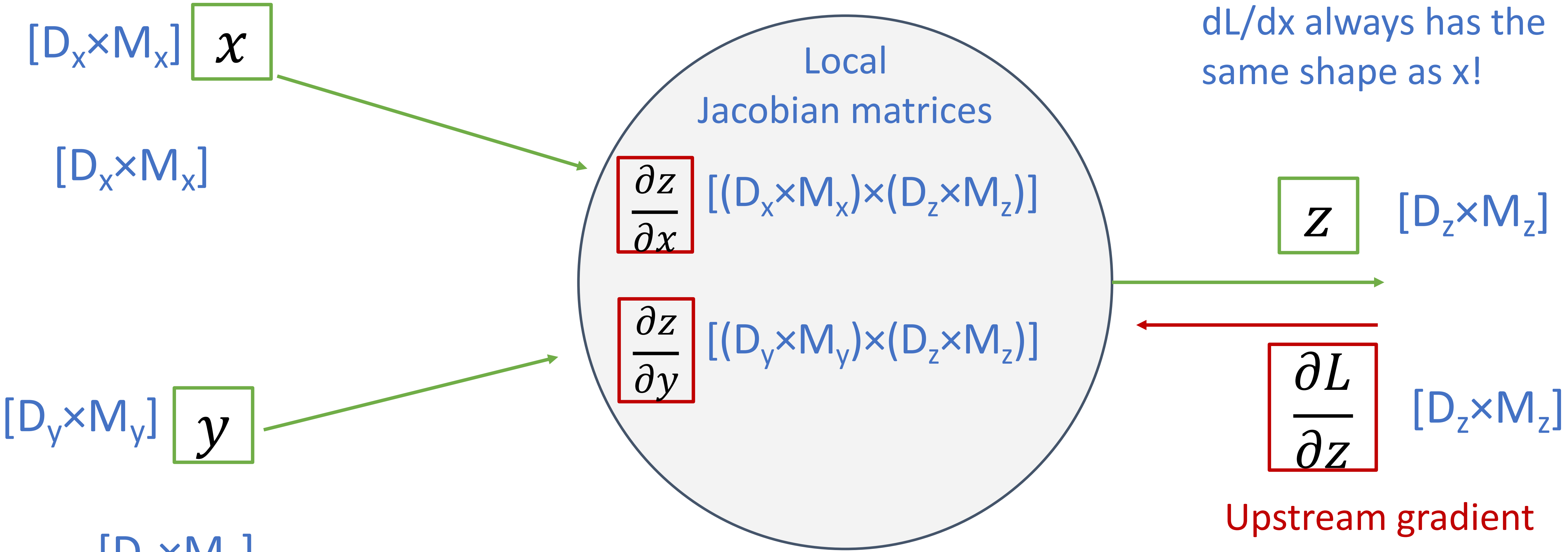
Upstream gradient

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?

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!

$[D_x \times M_x]$ $\boxed{x}$

$[D_x \times M_x]$ $\boxed{\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}}$

Matrix-vector multiply

$[D_y \times M_y]$ $\boxed{y}$

$[D_y \times M_y]$ $\boxed{\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}}$

Local Jacobian matrices

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

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

$\boxed{z}$ $[D_z \times M_z]$

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

Upstream gradient

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

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

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

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply *y = xw*

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Jacobians:

dy/dx: [(N×D)×(N×M)]

dy/dw: [(D×M)×(N×M)]

For a neural net we may have

N=64, D=M=4096

Each Jacobian takes 256 GB of memory! Must work with them implicitly!

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

$dL/dx_{1,1}$

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply *y = xw*

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

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

$dL/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

$dL/dx_{1,1}$

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

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

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

Matrix Multiply $y = xw$

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

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

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

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

Local Gradient Slice:

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

$= (w_{1,:}) \cdot (dL/dy_{1,:})$

$= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0$

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{2,3}$

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

$dL/dx_{2,3}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{2,3}$

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

$dL/dx_{2,3}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

$dL/dx_{i,j}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply *y = xw*

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

$$\underset{[N \times D]}{dL/dx} = \underset{[N \times M]}{(dL/dy)}\ \underset{[M \times D]}{w^T}$$

$dL/dx_{i,j}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

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

$$\underset{[N \times D]}{dL/dx} = \underset{[N \times M]}{(dL/dy)}\ \underset{[M \times D]}{w^T}$$

$$\underset{[D \times M]}{dL/dw} = \underset{[D \times N]}{x^T}\ \underset{[N \times M]}{(dL/dy)}$$

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

Backpropagation: Another View

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

D_0 D_1 D_2 D_3 scalar

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

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

D_0 D_1 D_2 D_3 scalar

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

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

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

D_0 D_1 D_2 D_3 scalar

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

Chain rule

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

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

Reverse-Mode Automatic Differentiation

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

D_0 D_1 D_2 D_3 scalar

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

Chain rule

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

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

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

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

a: scalar; x_0: D_0; x_1: D_1; x_2: D_2; x_3: D_3

Chain rule

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

Forward-Mode Automatic Differentiation

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

scalar $\quad D_0 \quad D_1 \quad D_2 \quad D_3$

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

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

scalar $\quad D_0 \quad D_1 \quad D_2 \quad D_3$

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

Beta implementation in PyTorch! https://pytorch.org/tutorials/intermediate/forward_ad_usage.html

Chain rule

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

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

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

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L$$

D_0 D_1 scalar

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

$[D_0 \times D_0]$

Hessian matrix
H of second derivatives.

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L$$

D_0 D_1 scalar

Hessian / vector multiply

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

Hessian matrix
H of second derivatives.

$[D_0 \text{ x } D_0]$

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

$[D_0 \text{ x } D_0]$ $[D_0]$

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L$$

D_0 D_1 scalar

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

Hessian matrix
H of second derivatives.

$[D_0 \times D_0]$

Hessian / vector multiply

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

(if v doesn't depend on x_0)

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

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\cdot v} (dL/dx_0) \cdot v$$

D_0 D_1 scalar D_1 D_0 scalar

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

Hessian matrix
H of second derivatives.

$[D_0 \text{ x } D_0]$

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)

$[D_0 \text{ x } D_0]$ $[D_0]$

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\cdot v} (dL/dx_0) \cdot v$$

D_0 D_1 scalar D_1 D_0 scalar

Backprop!

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

$[D_0 \times D_0]$

Hessian matrix
H of second derivatives.

Hessian / vector multiply

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

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

(if v doesn't depend on x_0)

Backprop: Higher-Order Derivatives

$$\underset{D_0}{x_0} \xrightarrow{f_1} \underset{D_1}{x_1} \xrightarrow{f_2} \underset{\text{scalar}}{L} \xrightarrow{f'_2} \underset{D_1}{dL/dx_1} \xrightarrow{f'_1} \underset{D_0}{dL/dx_0} \xrightarrow{\cdot v} \underset{\text{scalar}}{(dL/dx_0) \cdot v}$$

Backprop!

This is implemented in PyTorch / Tensorflow!

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

$[D_0 \times D_0]$

Hessian matrix H of second derivatives.

Hessian / vector multiply

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

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

(if v doesn't depend on x_0)

Backprop: Higher-Order Derivatives

$$x_0 \xrightarrow{f_1} x_1 \xrightarrow{f_2} L \xrightarrow{f'_2} dL/dx_1 \xrightarrow{f'_1} dL/dx_0 \xrightarrow{\text{norm}} |dL/dx_0|^2$$

D_0 D_1 scalar D_1 D_0 scalar

Backprop!

This is implemented in PyTorch / Tensorflow!

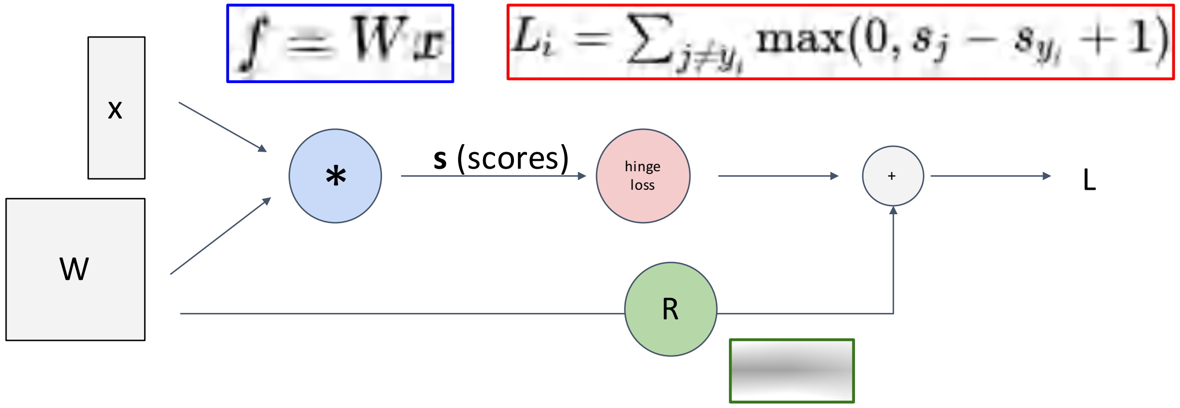
Example: Regularization to penalize the norm of the gradient

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

Gulrajani et al, “Improved Training of Wasserstein GANs”, NeurIPS 2017

Summary

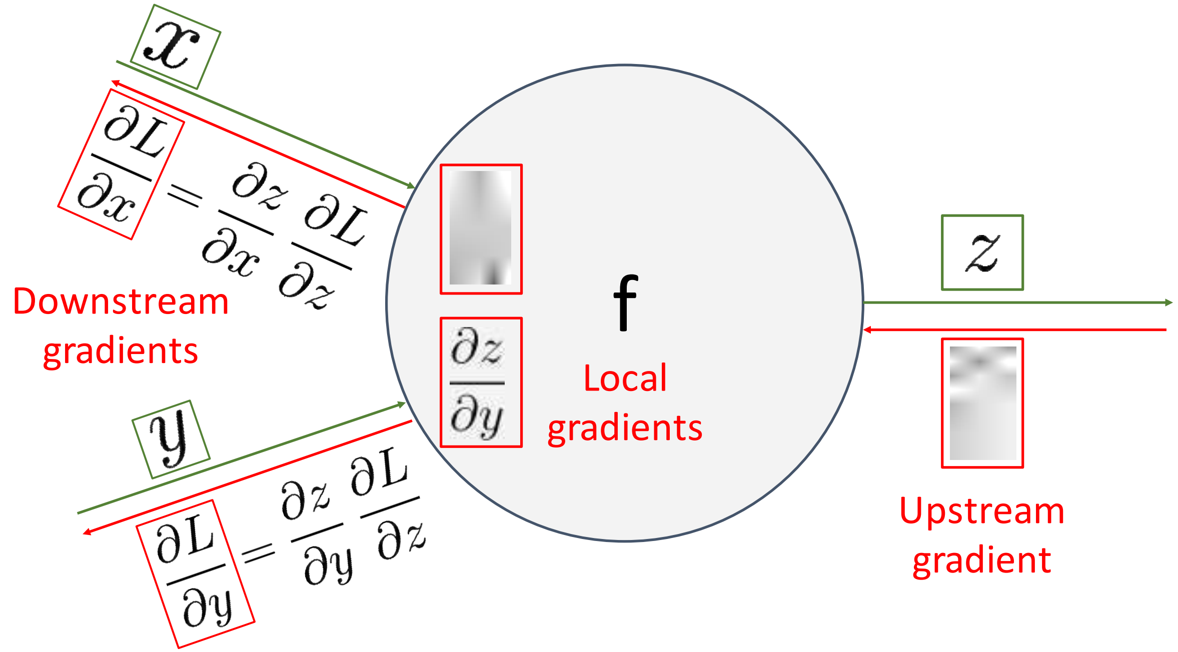
Represent complex expressions as **computational graphs**



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!)

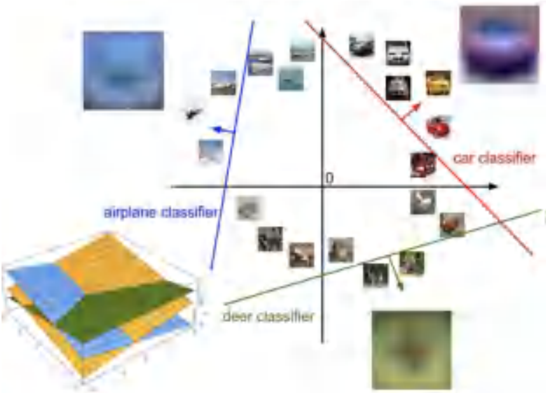
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  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!)

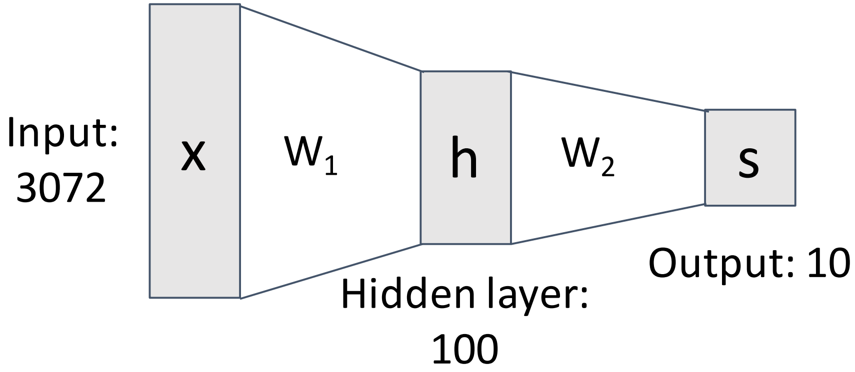
```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz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

$f(x,W) = Wx$

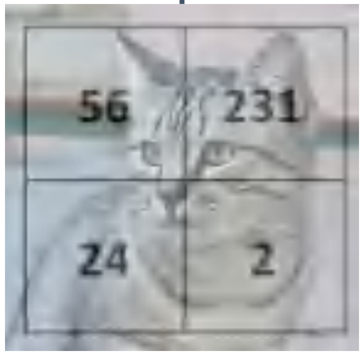


Problem: So far our classifiers don't respect the spatial structure of images!

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



Stretch pixels into column



Input image
(2, 2)

56
231
24
2

(4,)

Next time:
Convolutional Neural Networks