18. Backpropagation

STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

May {14, 16}, 2024

** Slides adapted from EECS498/598 @ Univ. of Michigan by Justin Johnson*

연세대학교
YONSEI UNIVERSITY

Assignment 4

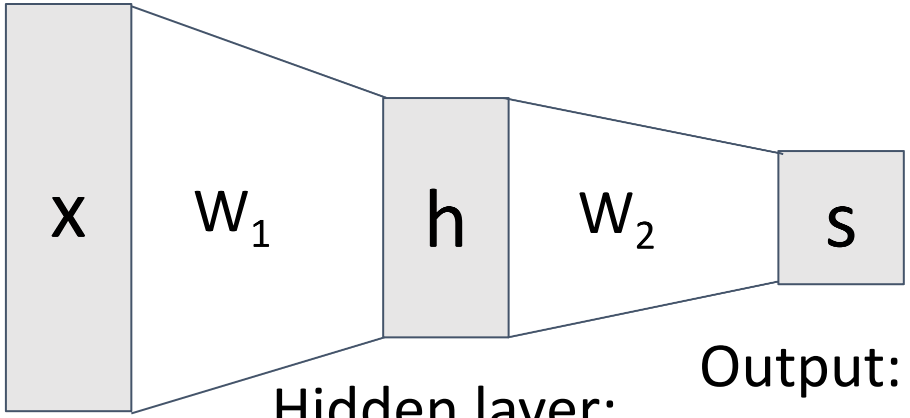
- Due **Friday 5/17, 11:59pm**
- Topics
  - K-Means and Gaussian Mixture Models
  - Principal Component Analysis
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take option 2.

Recap: Neural Networks

From linear classifiers to
fully-connected networks

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

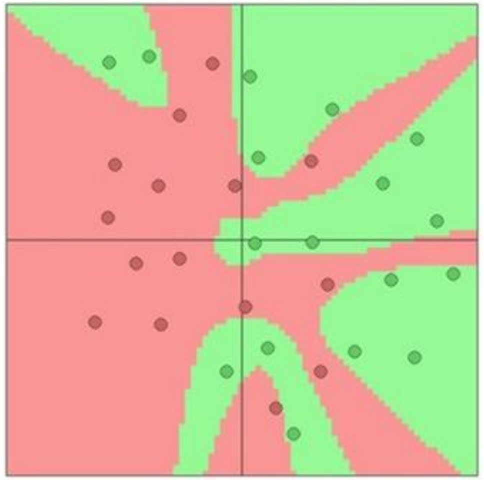
Input:
D



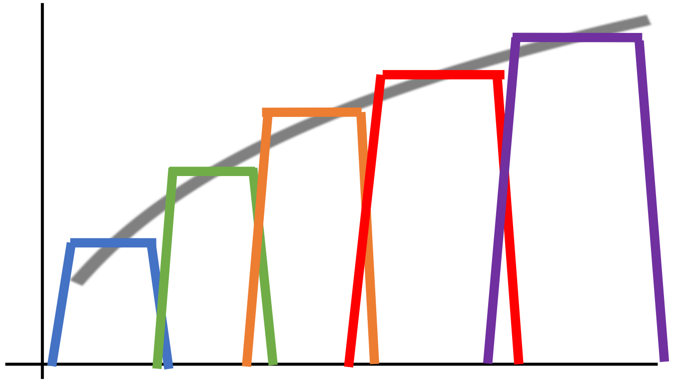
Hidden layer:
H

Output: C

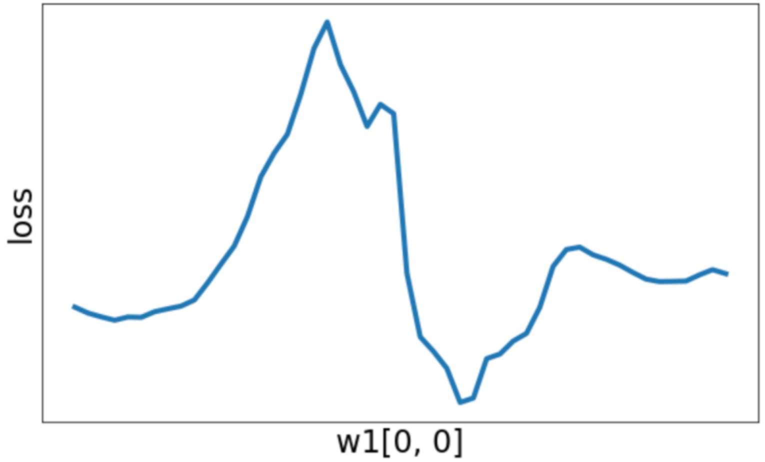
Space Warping



Universal Approximation



Non-convex



Problem: How to compute (complex) gradients?

$s = W_2\, g(W_1 x + b_1) + b_2$ 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: Backpropagation by **Chain Rule**

$$x \xrightarrow{W_1} h_1 \xrightarrow{W_2} h_2 \xrightarrow{W_3} h_3 \xrightarrow{W_4} L$$

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

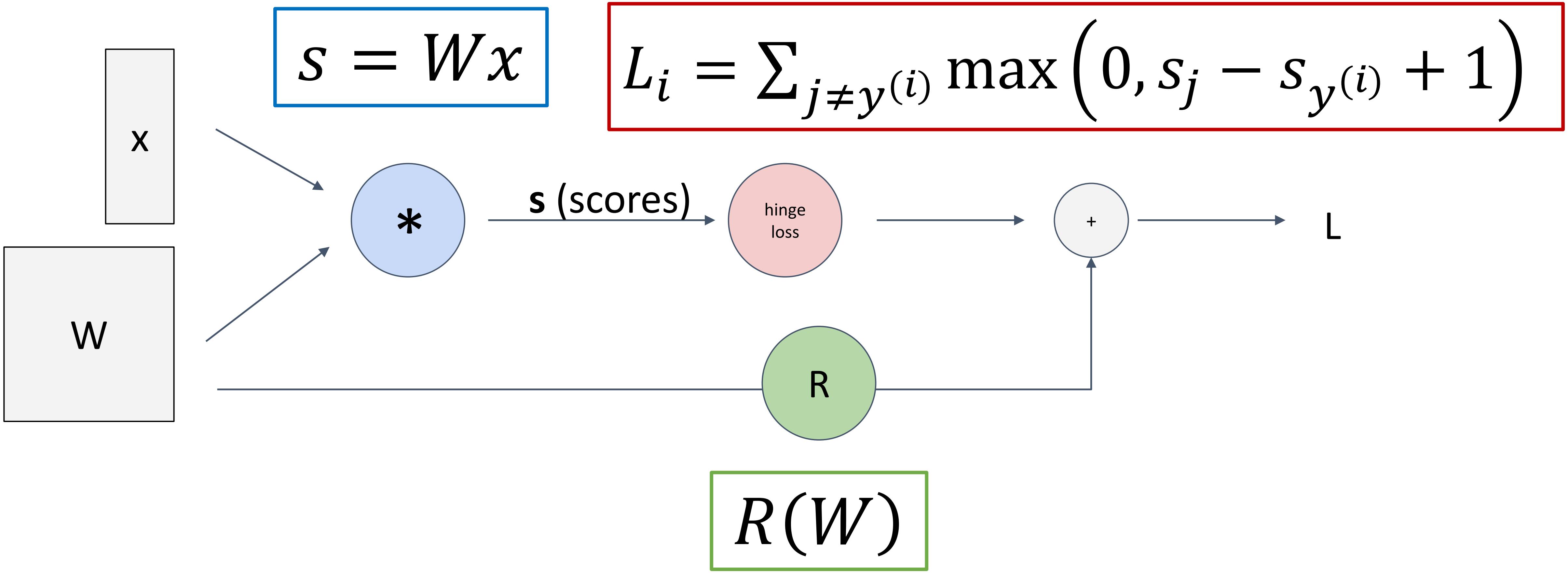
Chain rule

$$\frac{\partial L}{\partial x} = \underbrace{\left(\frac{\partial h_1}{\partial x}\right)}_{[D_0 \text{ x } D_1]} \underbrace{\left(\frac{\partial h_2}{\partial h_1}\right)}_{[D_1 \text{ x } D_2]} \underbrace{\left(\frac{\partial h_3}{\partial h_2}\right)}_{[D_2 \text{ x } D_3]} \underbrace{\left(\frac{\partial L}{\partial h_3}\right)}_{[D_3]}$$

e.g., (i,j)-th element in W_2

$$\frac{\partial L}{\partial W_2(i,j)} = \underbrace{\left(\frac{\partial h_2}{\partial W_2(i,j)}\right)}_{[D_2]} \underbrace{\left(\frac{\partial h_3}{\partial h_2}\right)}_{[D_2 \text{ x } D_3]} \underbrace{\left(\frac{\partial L}{\partial h_3}\right)}_{[D_3]}$$

Better Idea: **Computational Graphs**

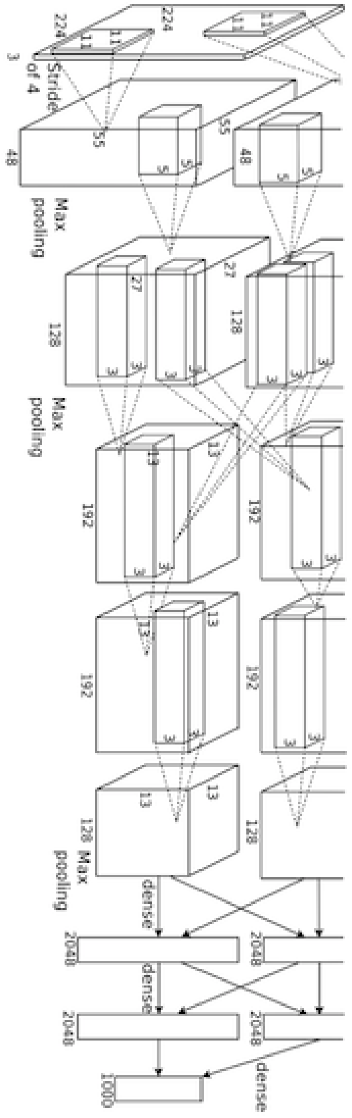


Deep Network (AlexNet)

input image

weights

loss



Neural Turing Machine

input image

loss

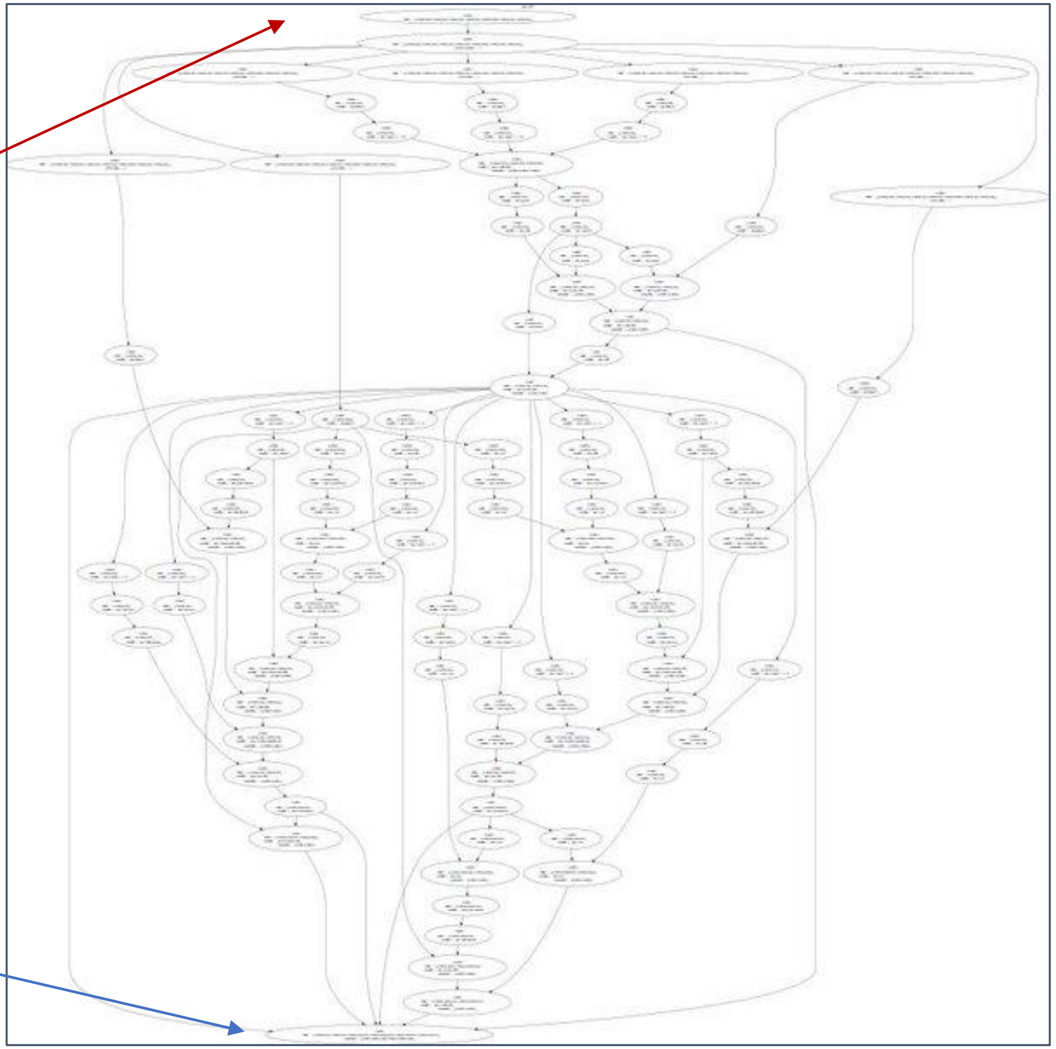


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

Graves et al, arXiv 2014

Neural Turing Machine

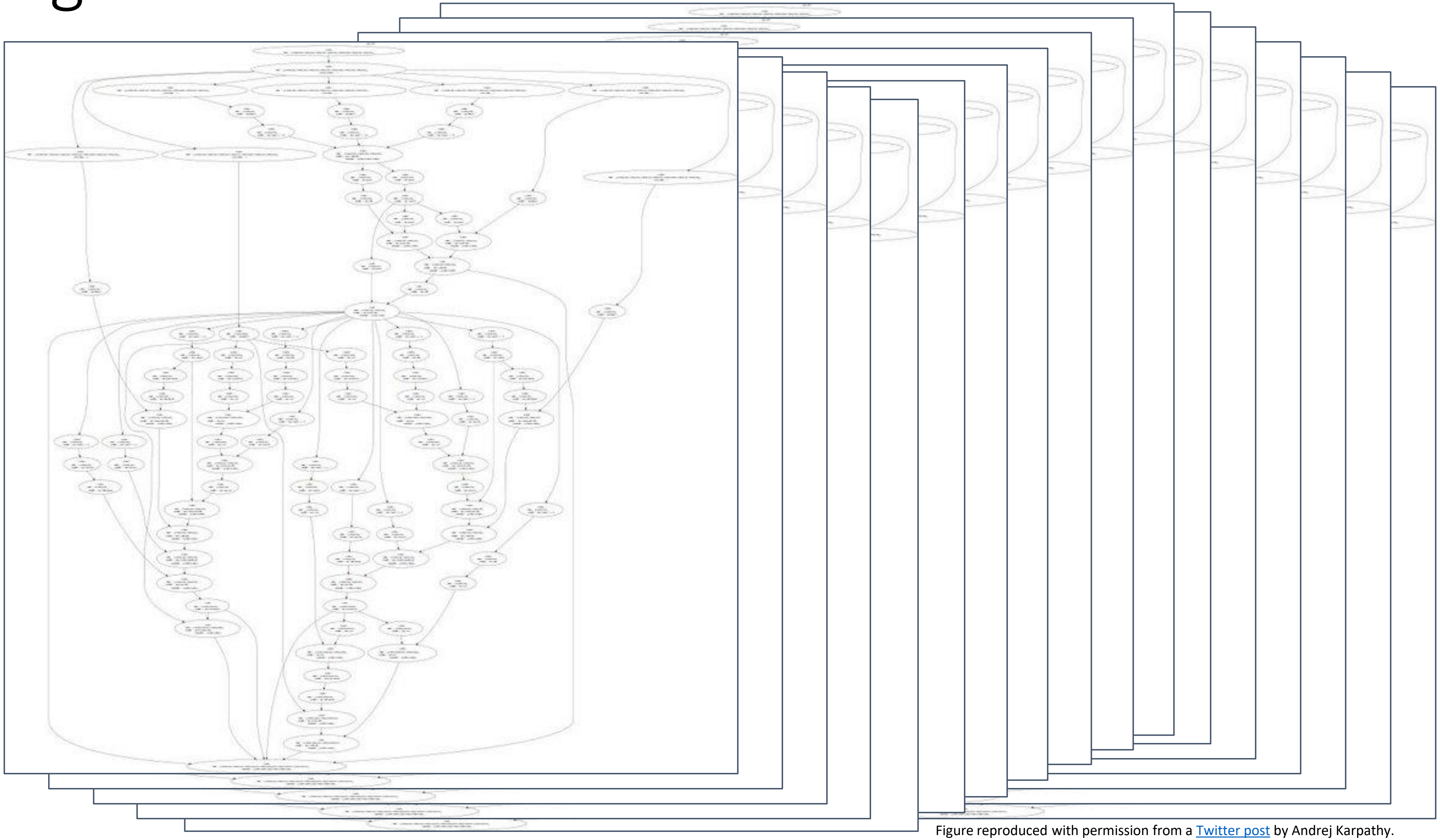
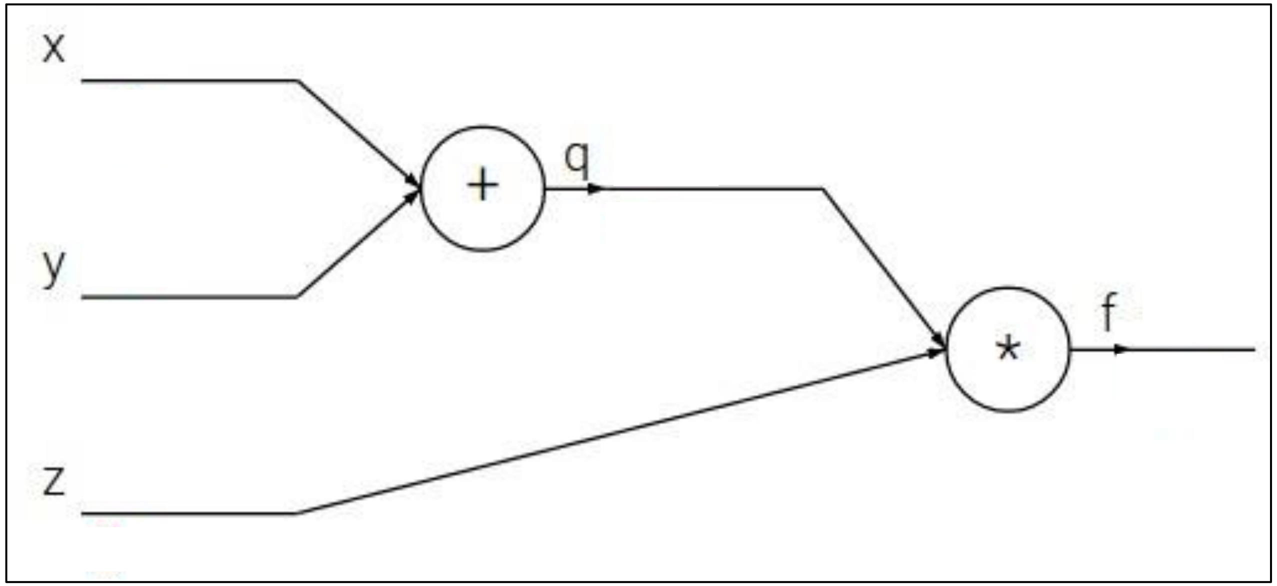


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

Graves et al, arXiv 2014

Backpropagation: Simple Example

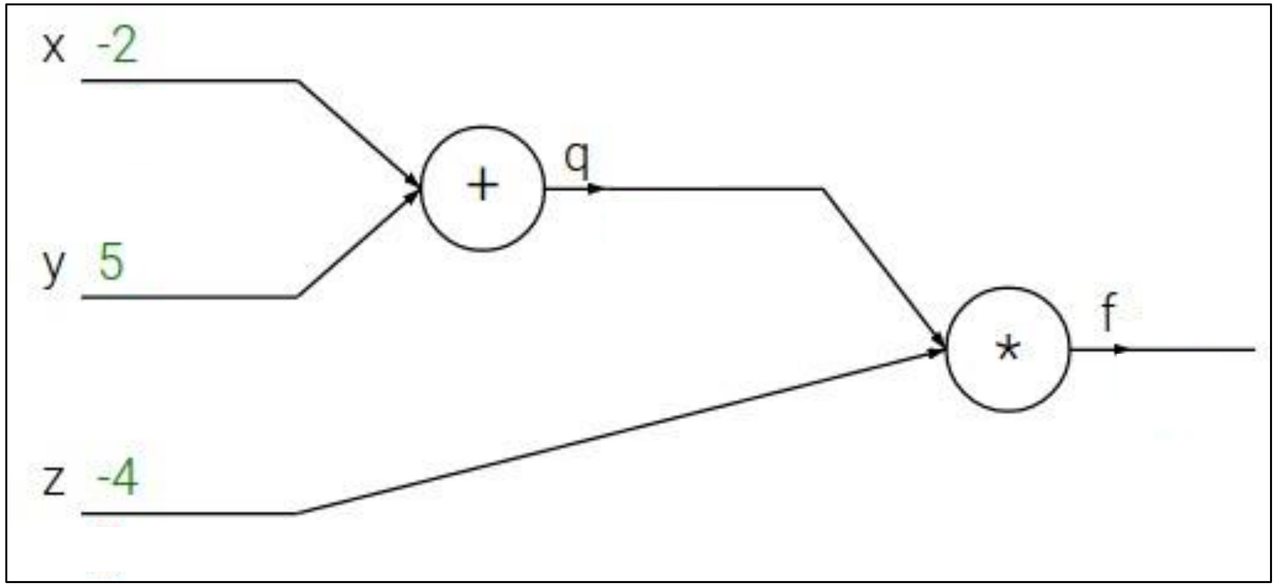
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

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

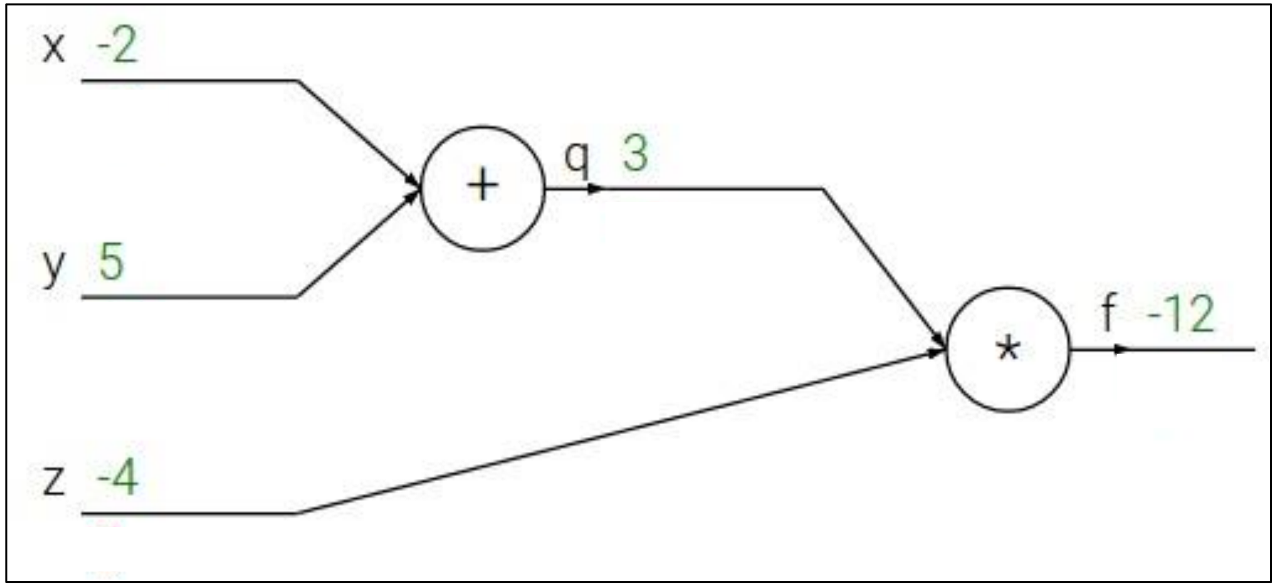
e.g., x = -2, y = 5, z = -4



Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

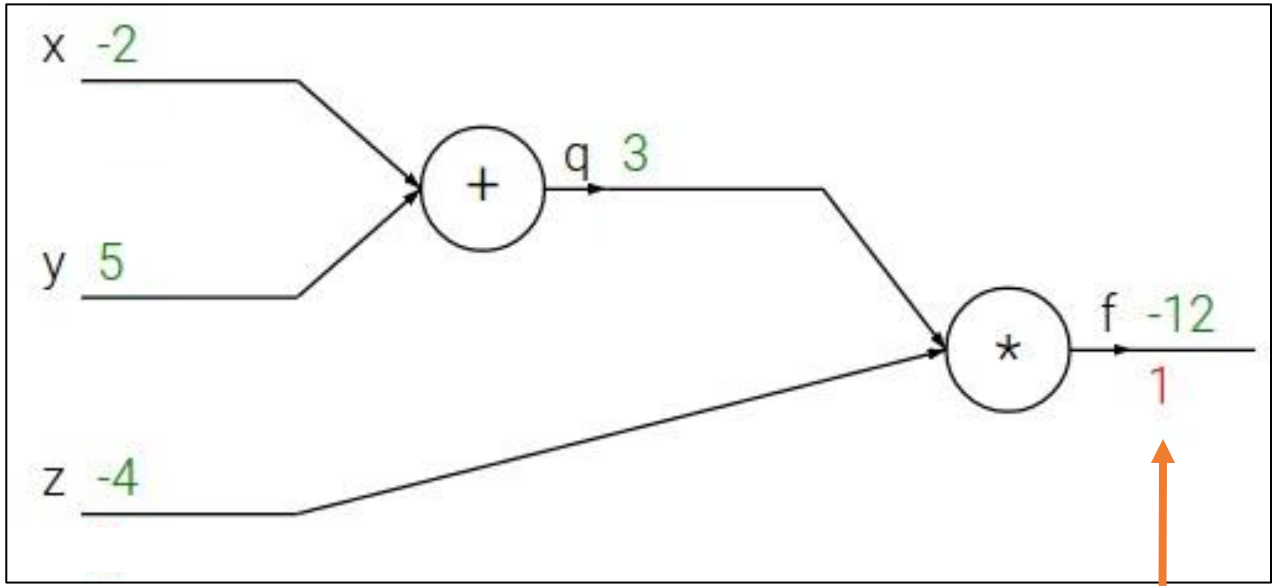
2. Backward pass: Compute derivatives

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



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

1. Forward pass: Compute outputs

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

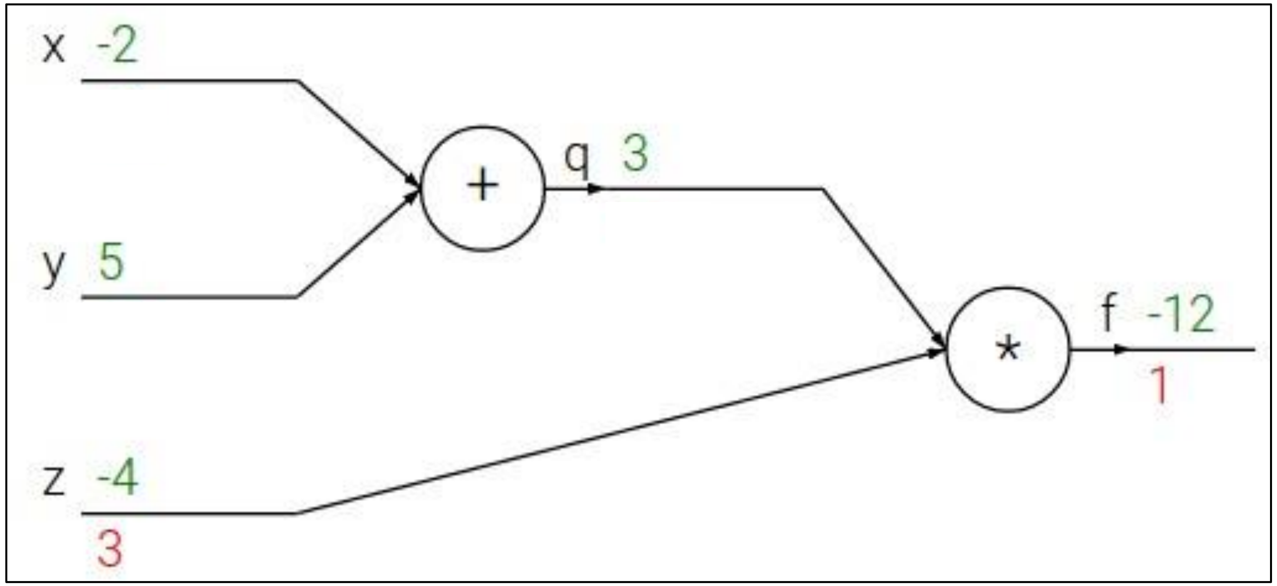
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

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

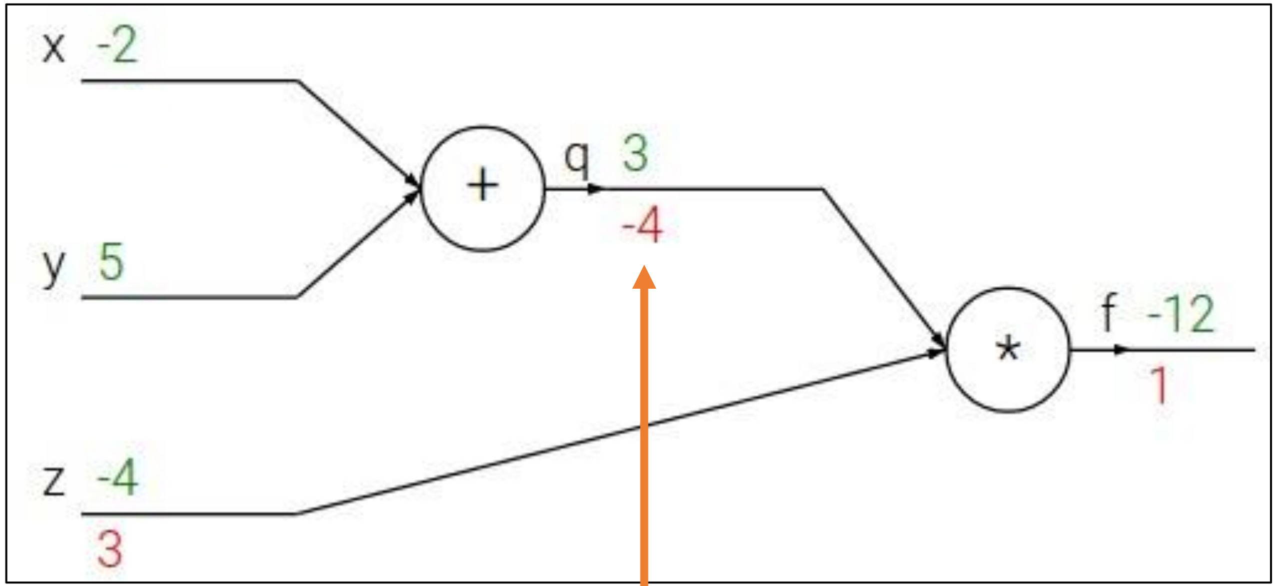
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

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

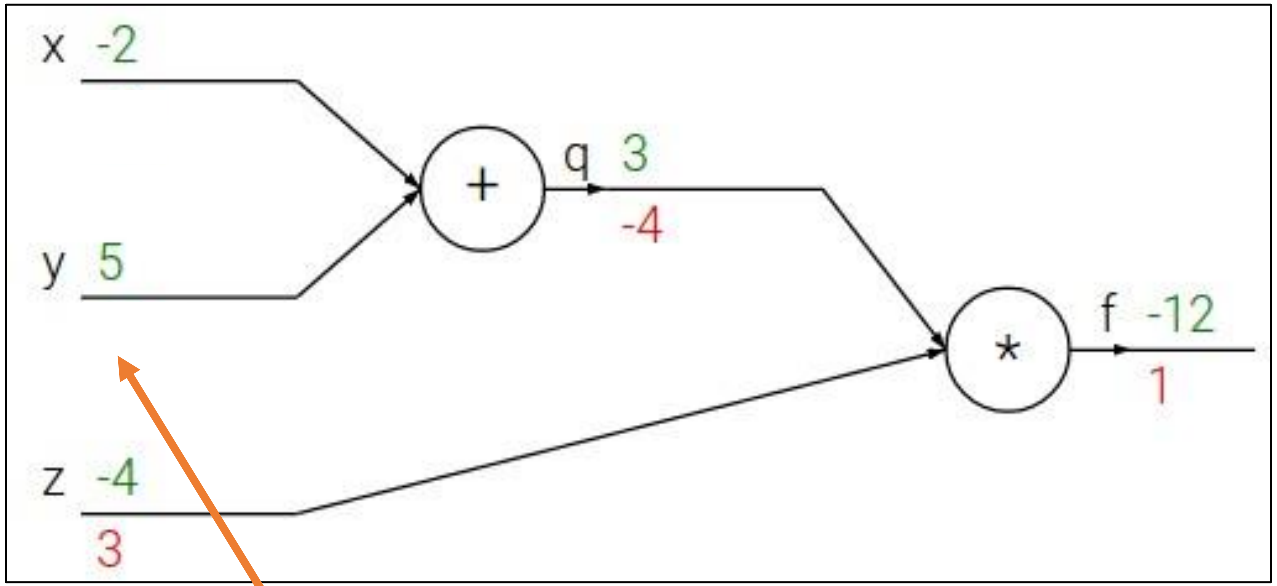
2. Backward pass: Compute derivatives

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



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

1. Forward pass: Compute outputs

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

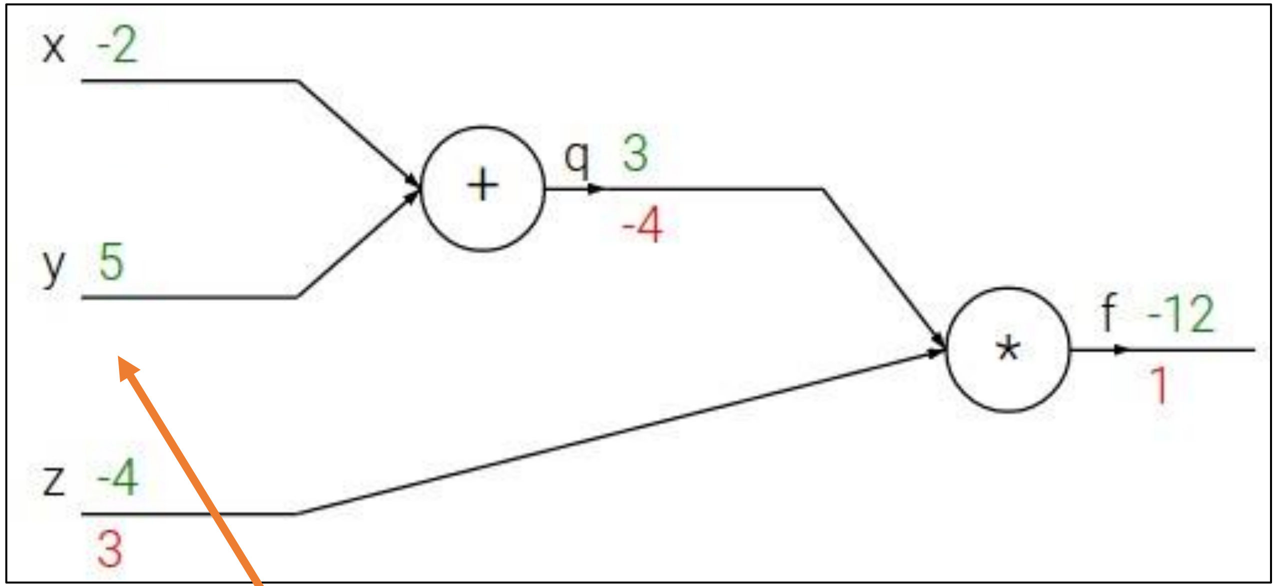
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

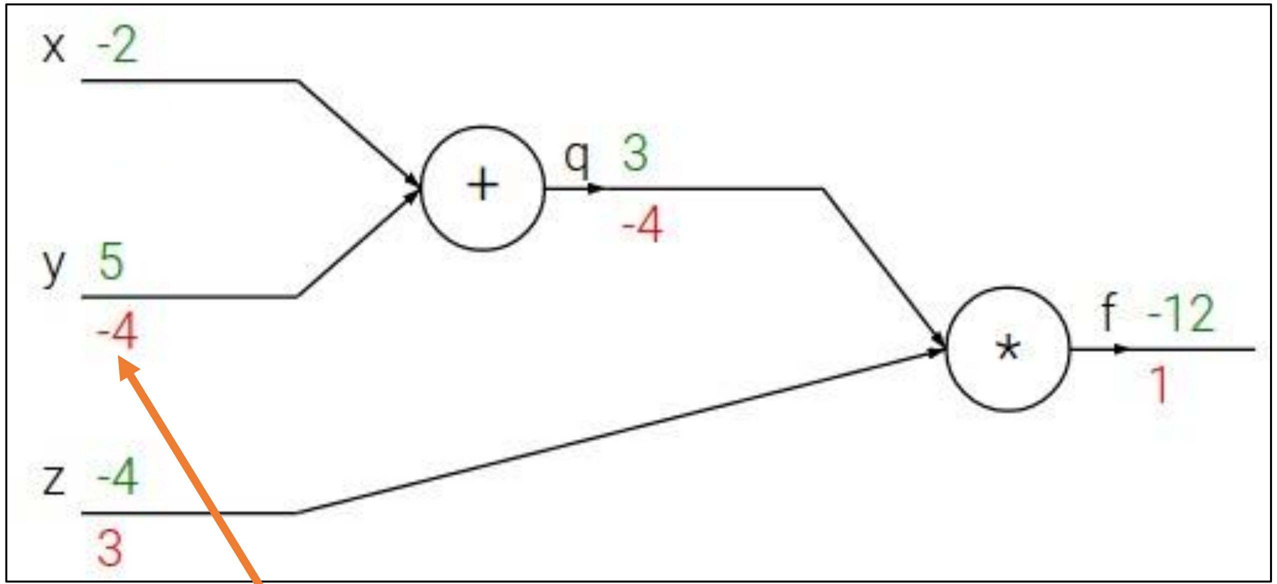
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

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

Chain Rule

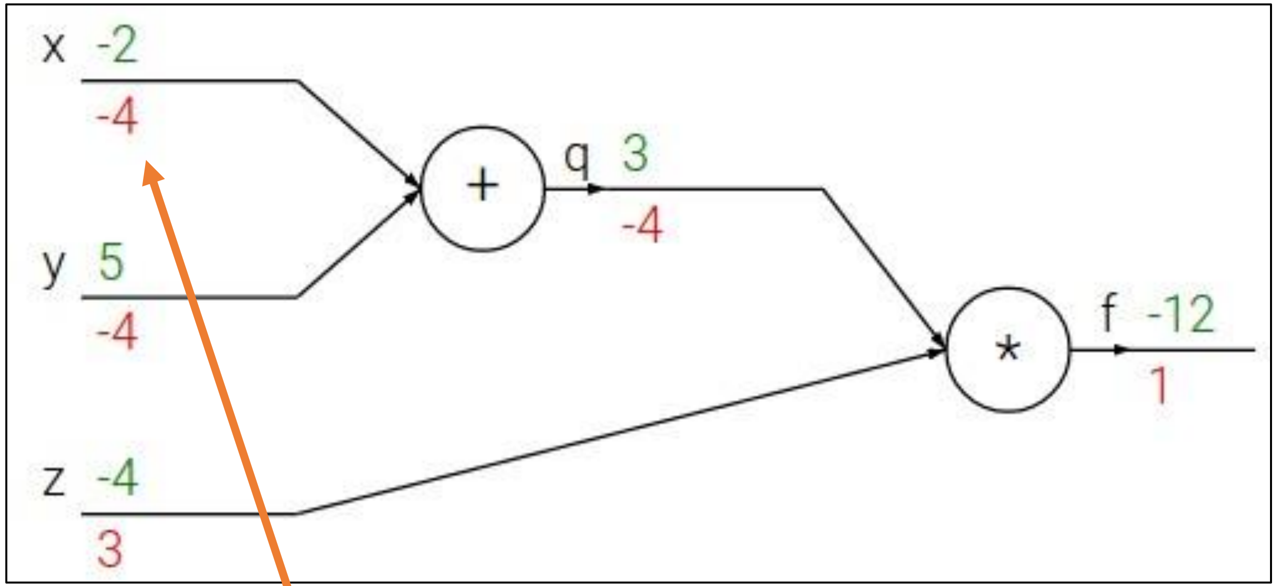
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

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

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

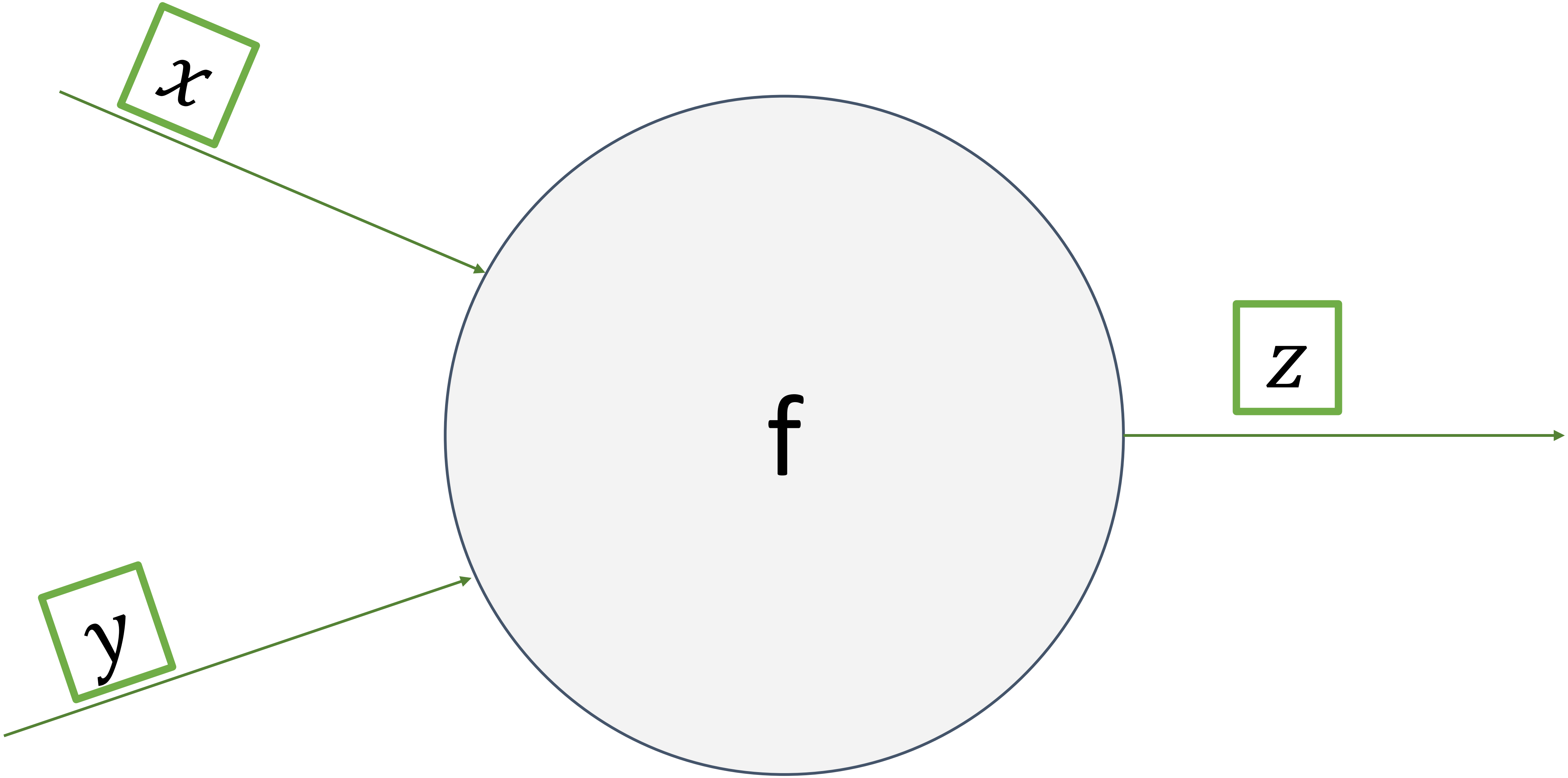
2. Backward pass: Compute derivatives

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

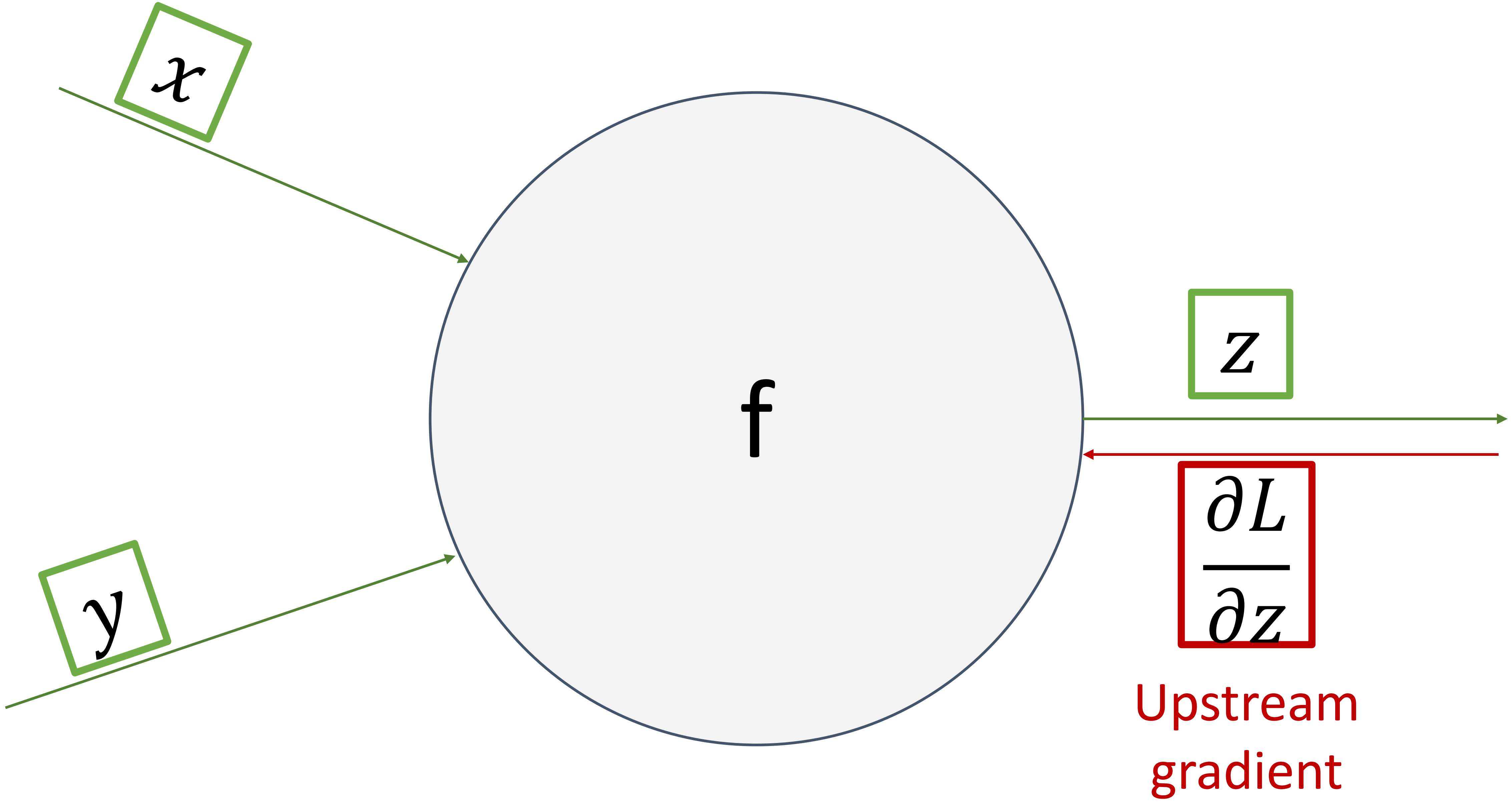
Chain Rule

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

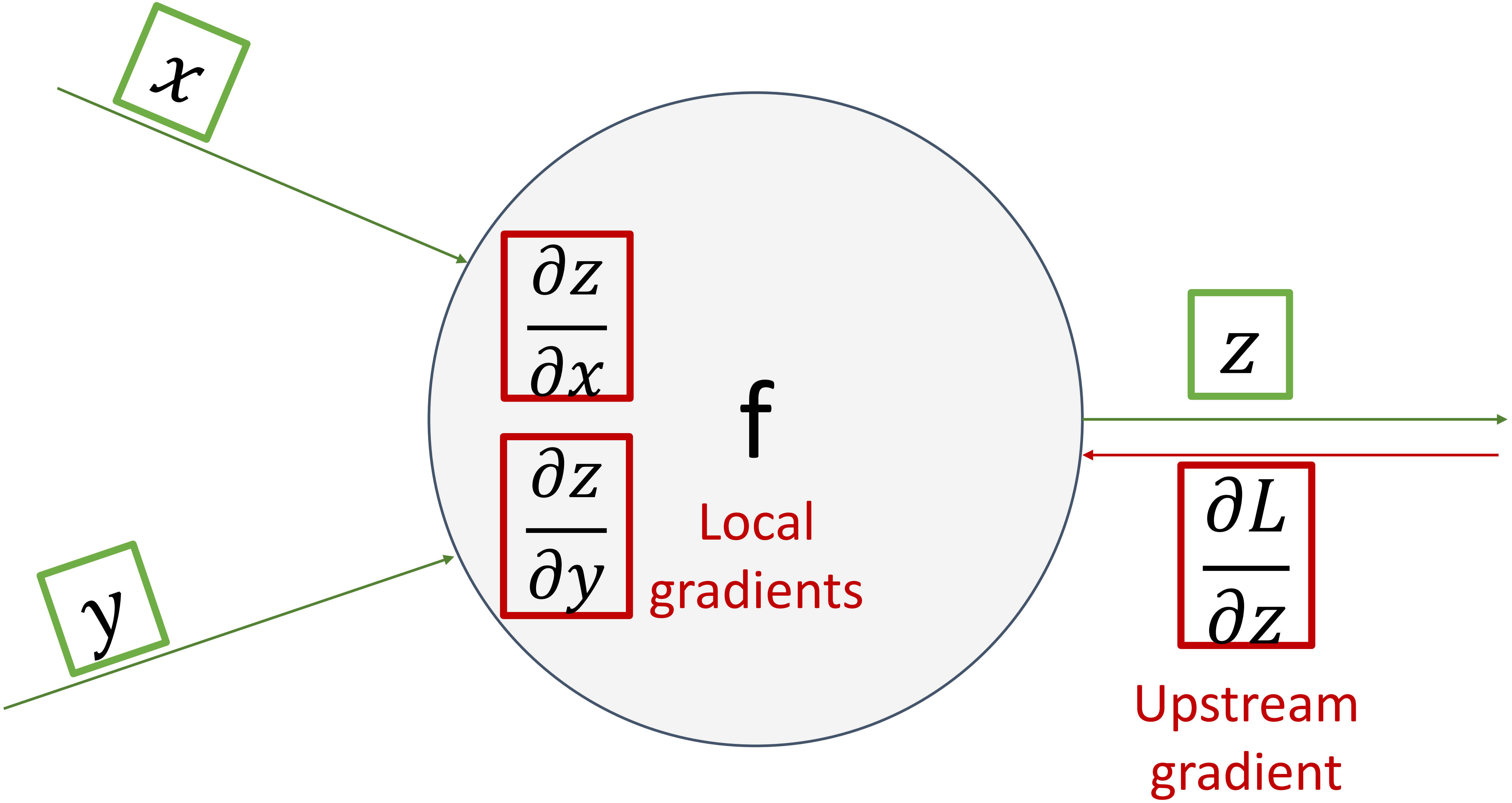
Downstream Gradient **Local Gradient** **Upstream Gradient**



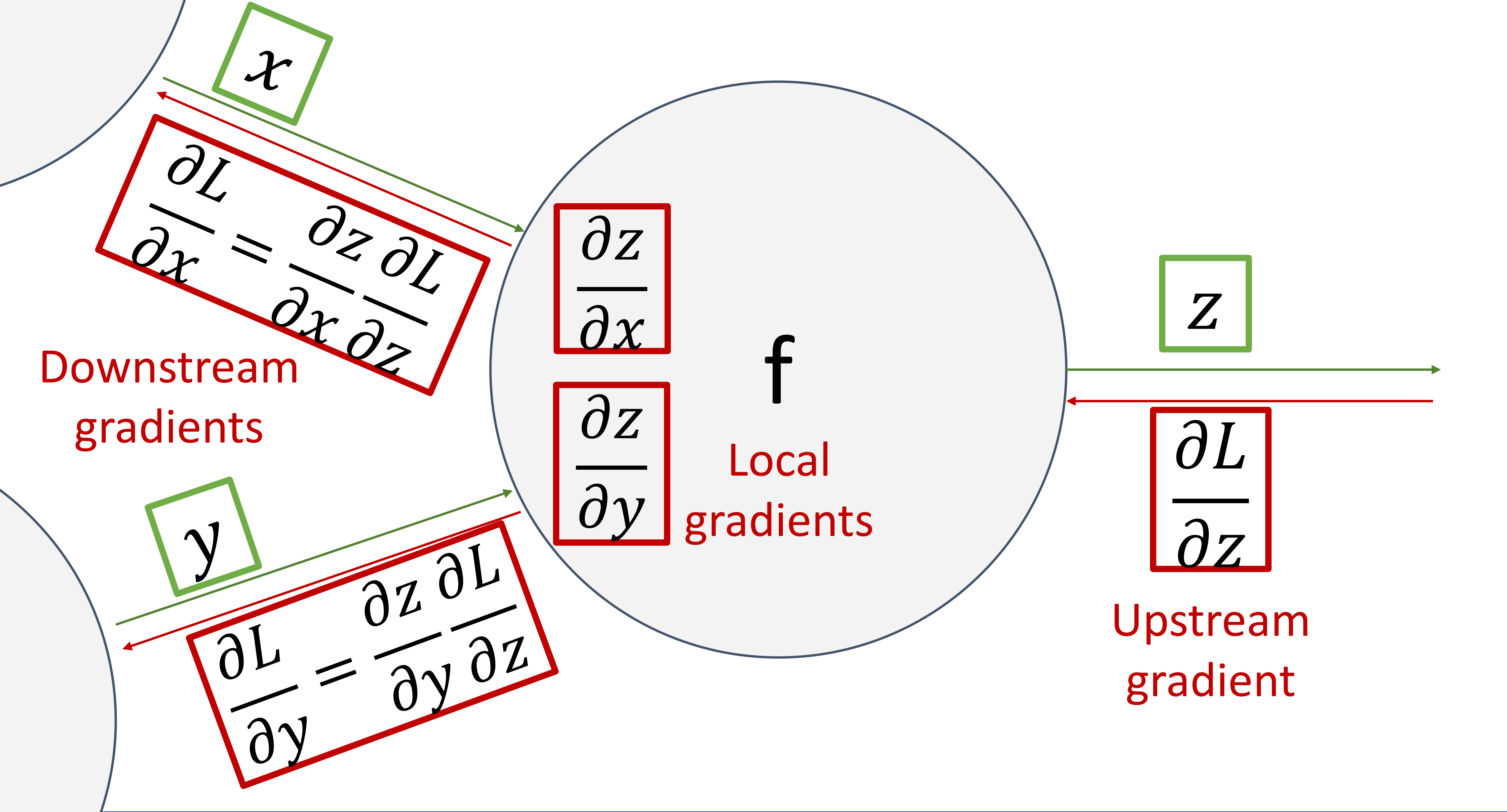
x
y
f
z



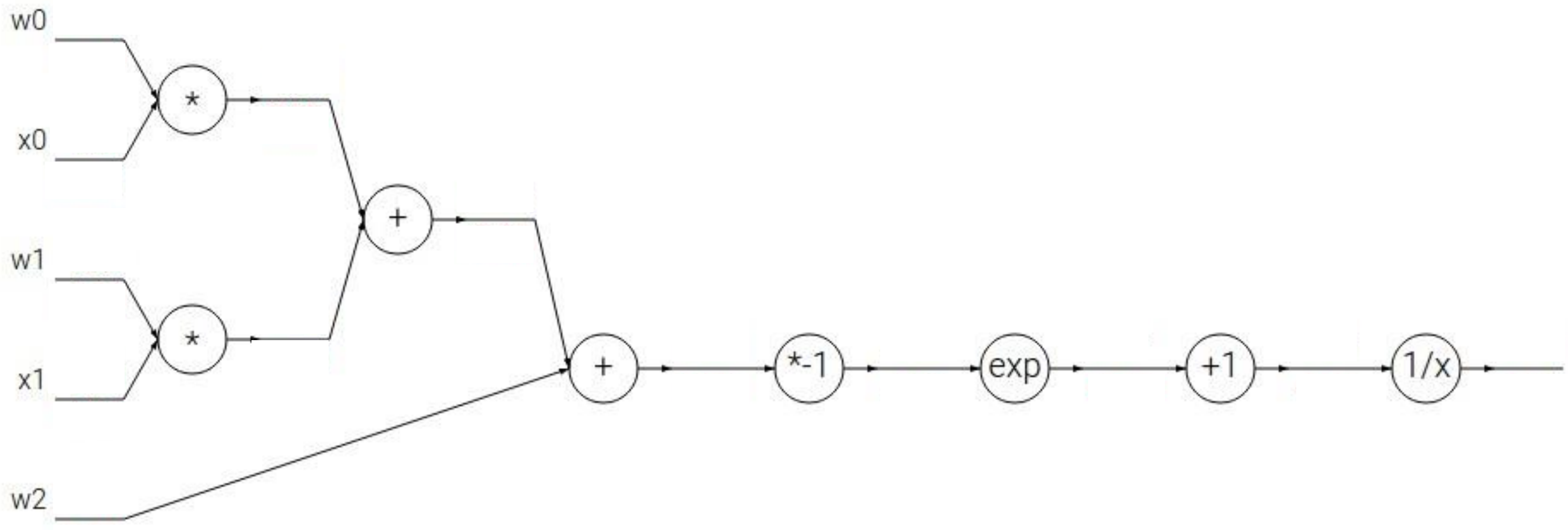
x
y
f
z
$\frac{\partial L}{\partial z}$
Upstream gradient



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

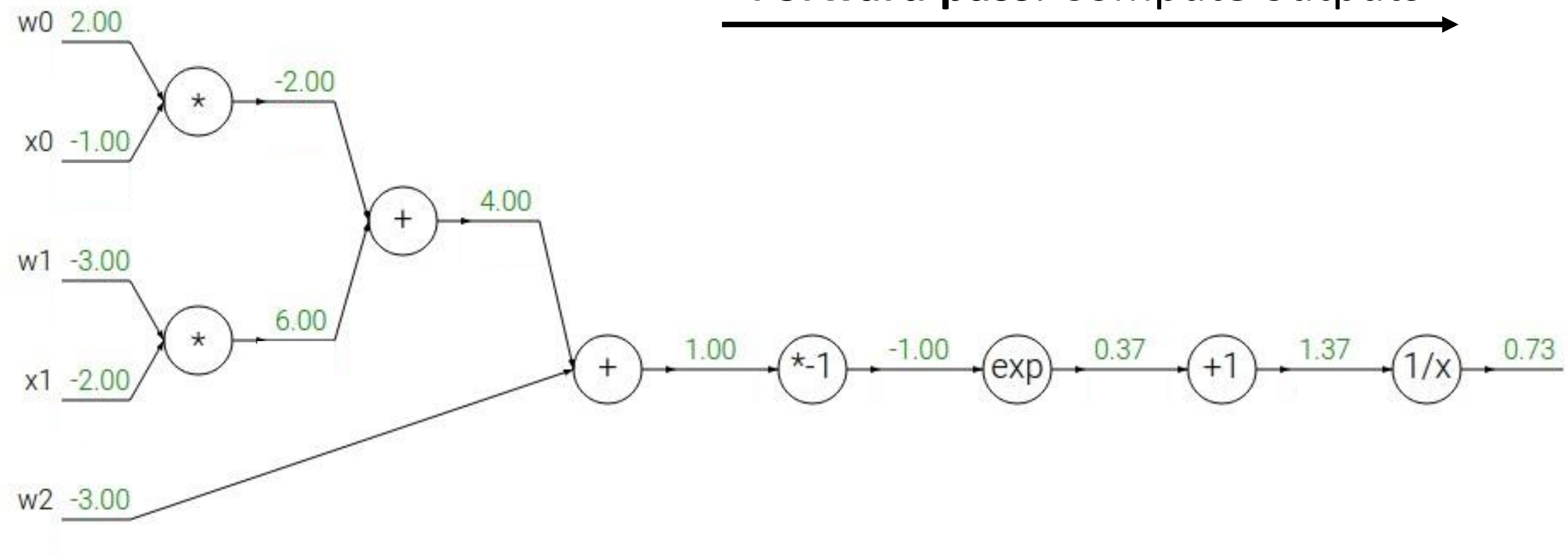

x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
Local gradients
z
$\frac{\partial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

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



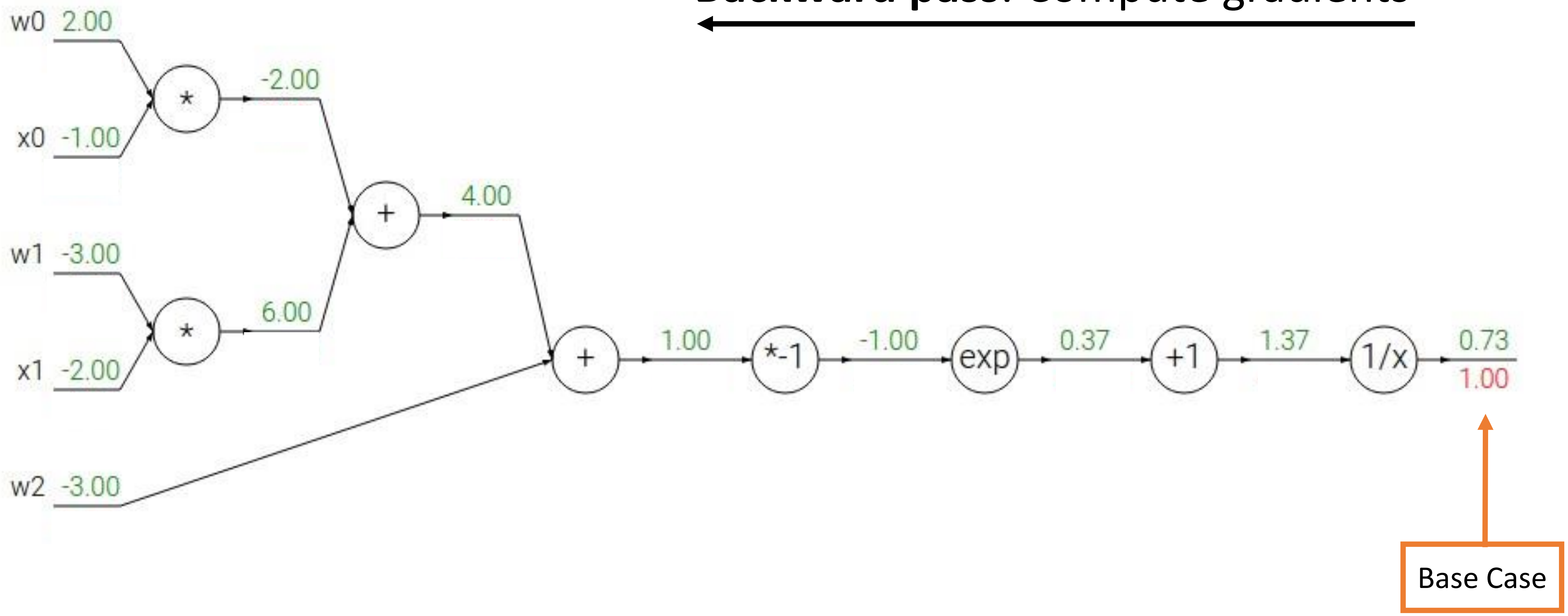
Another Example $f(x,w) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Forward pass: Compute outputs



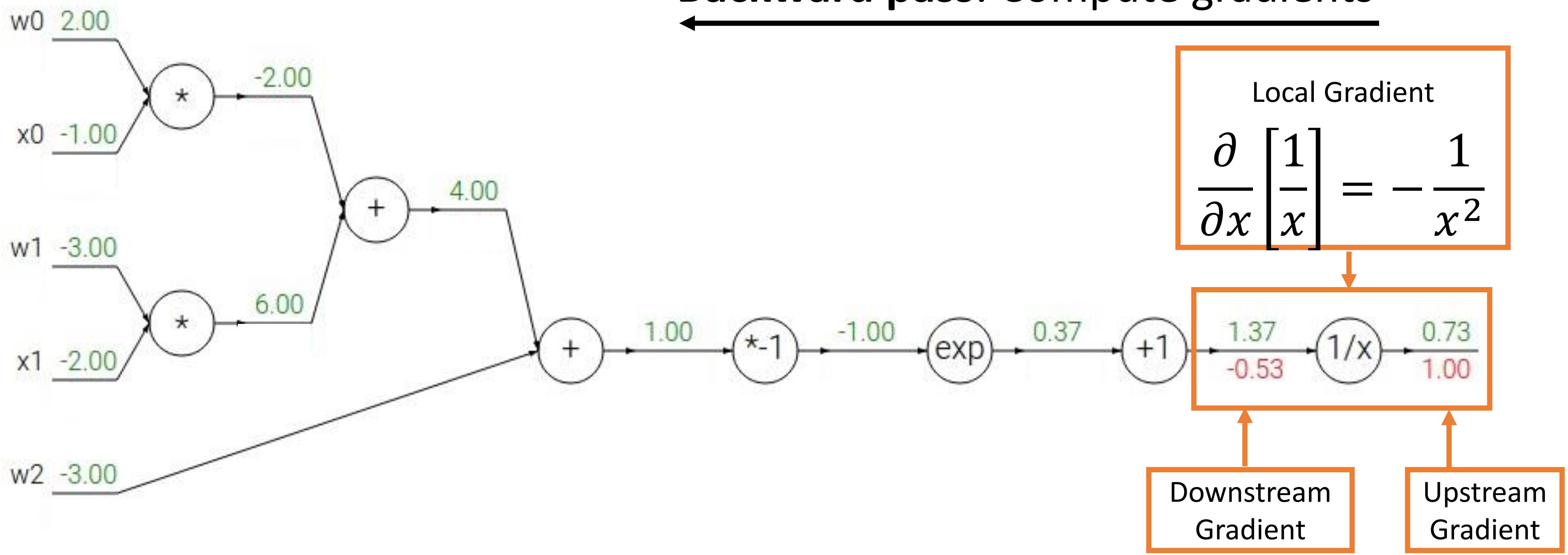
Another Example $f(x,w) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



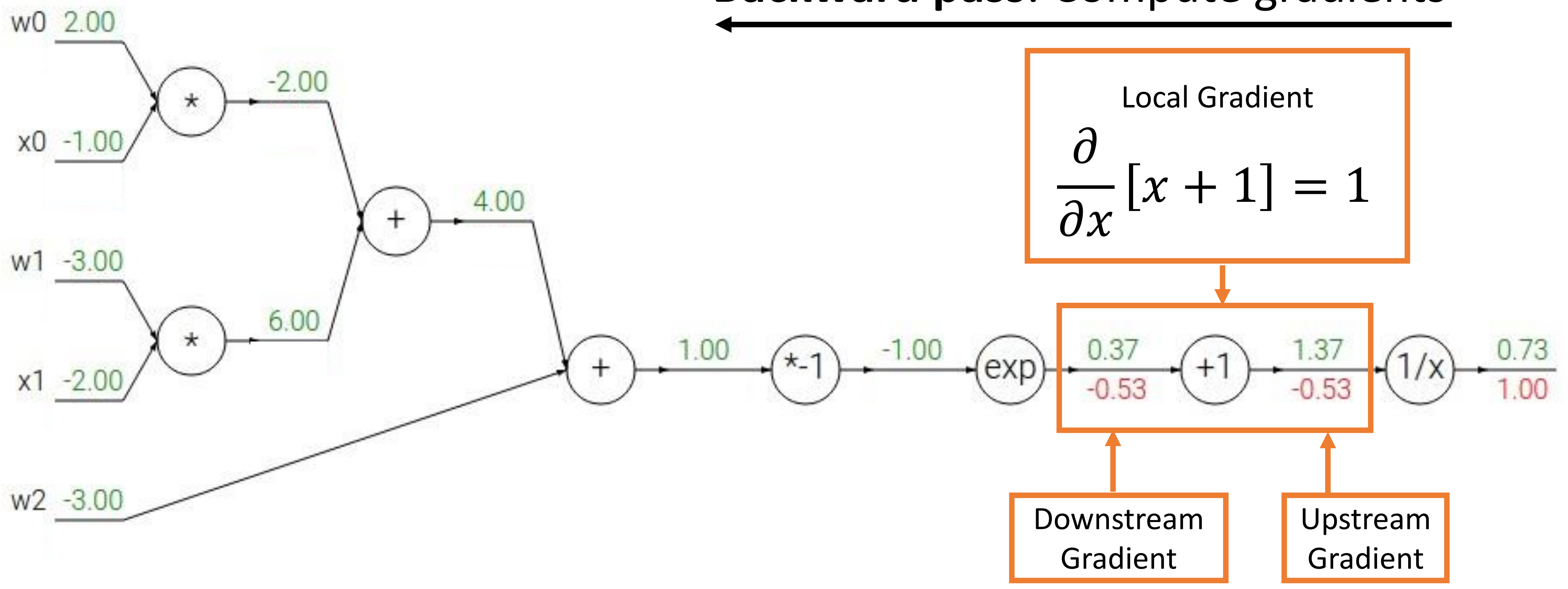
Another Example $f(x,w) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



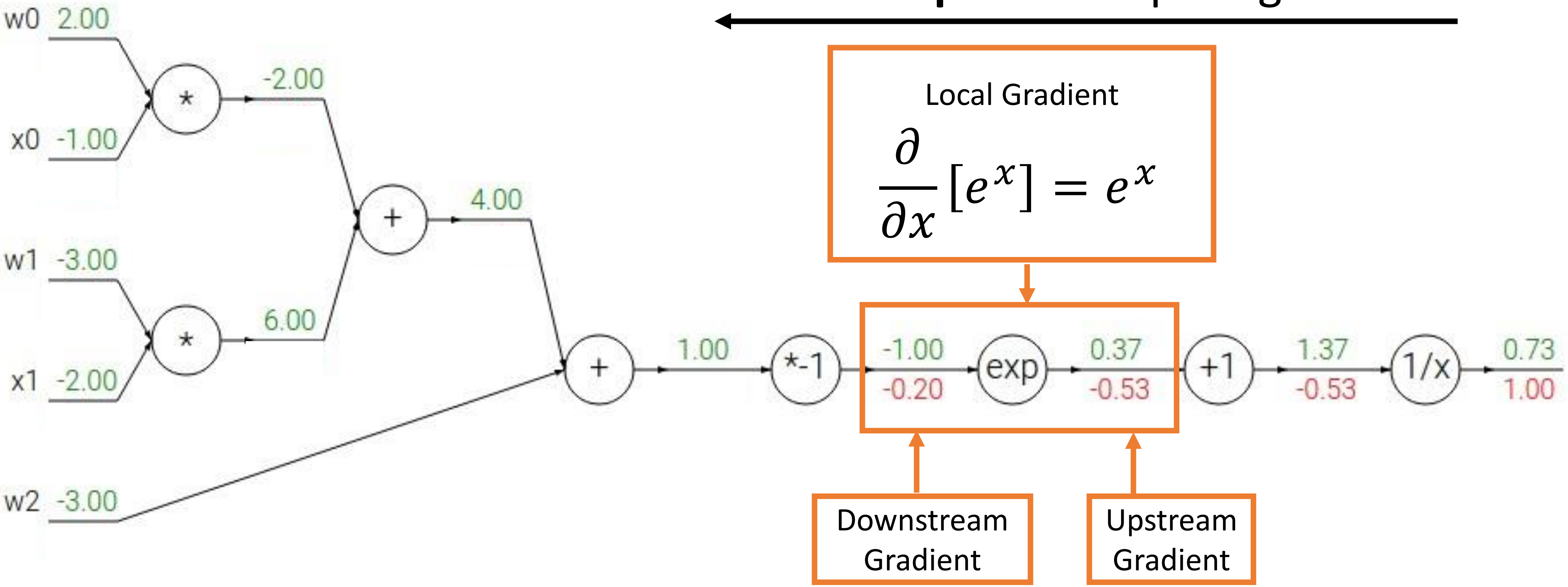
Another Example $f(x,w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

Backward pass: Compute gradients



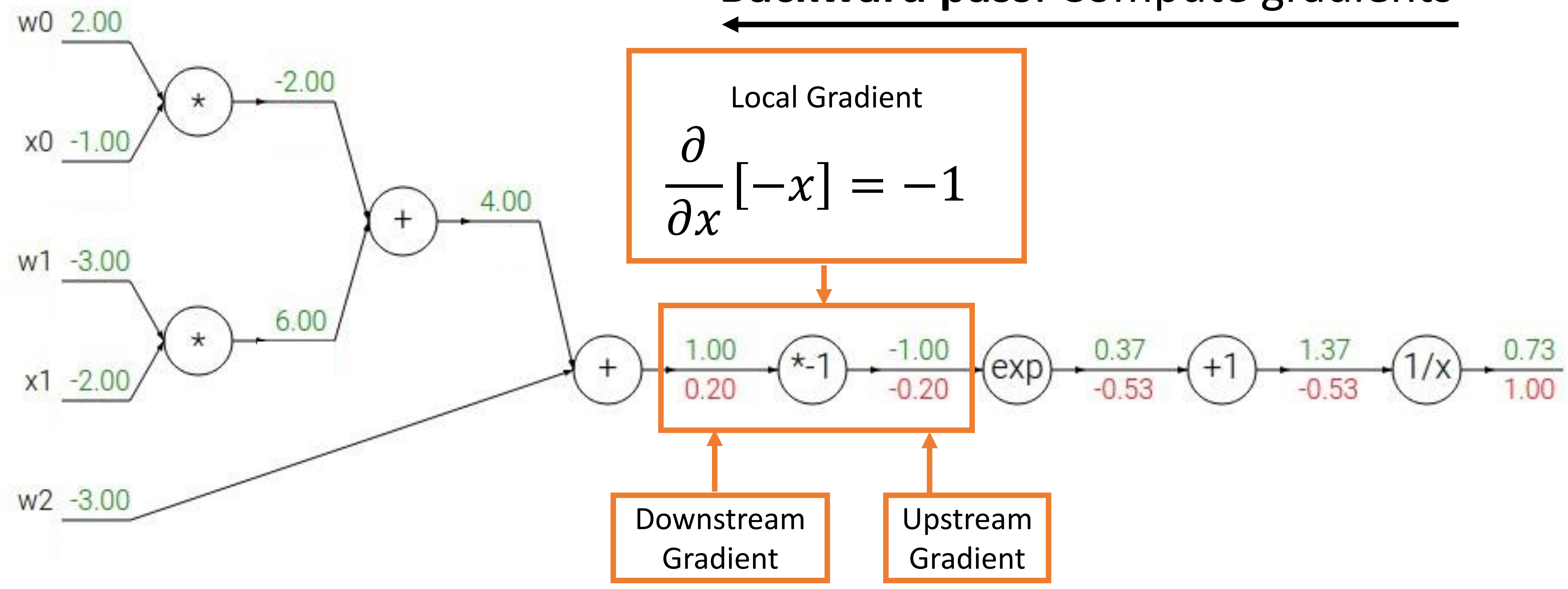
Another Example $f(x,w) = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

Backward pass: Compute gradients



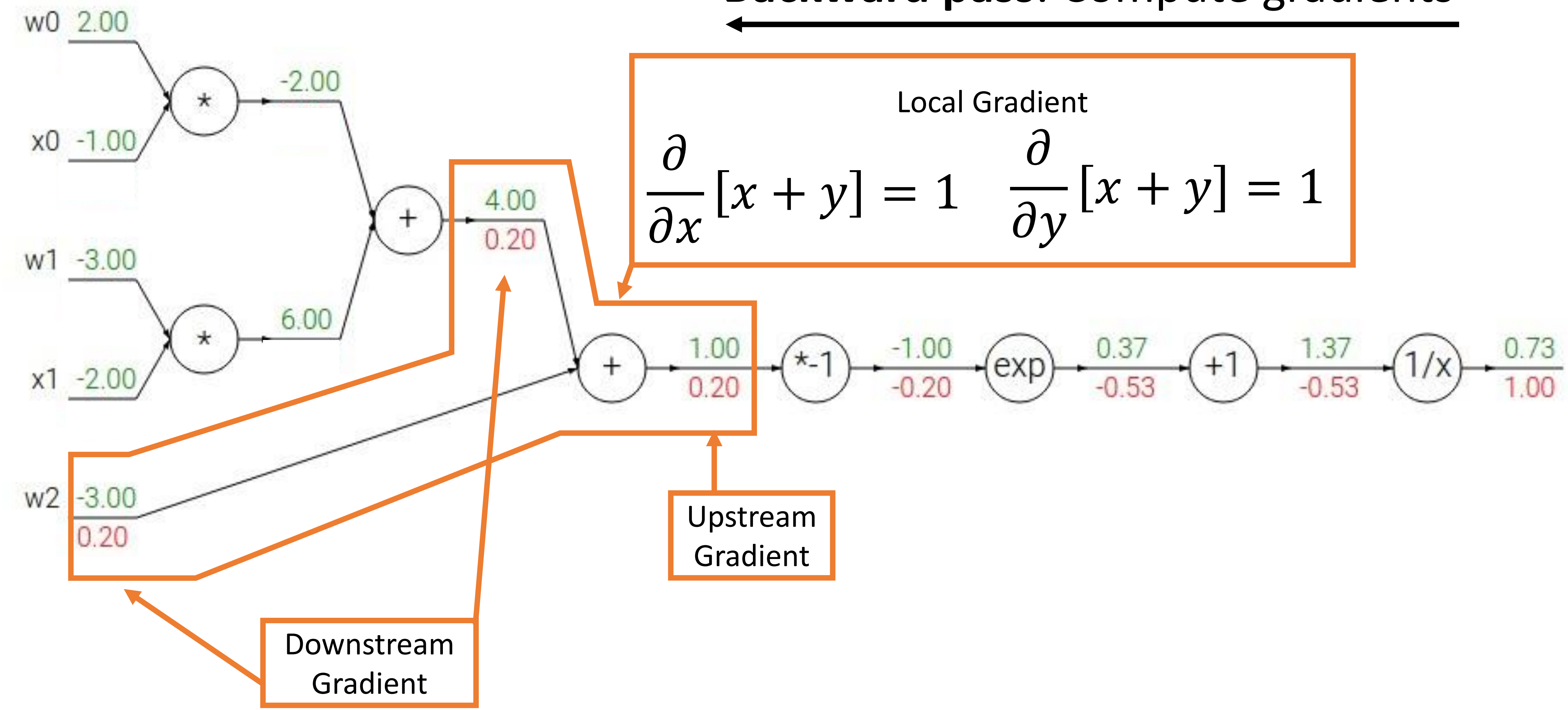
Another Example $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Backward pass: Compute gradients



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

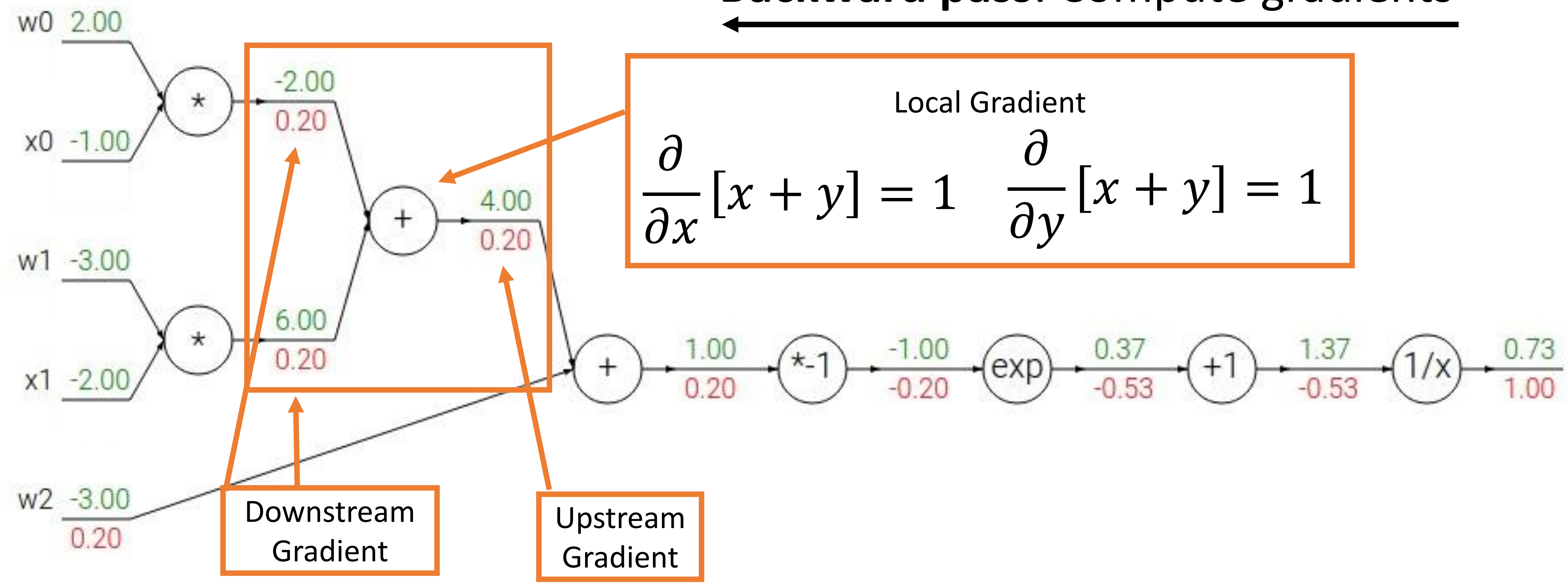
Backward pass: Compute gradients



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

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

Backward pass: Compute gradients



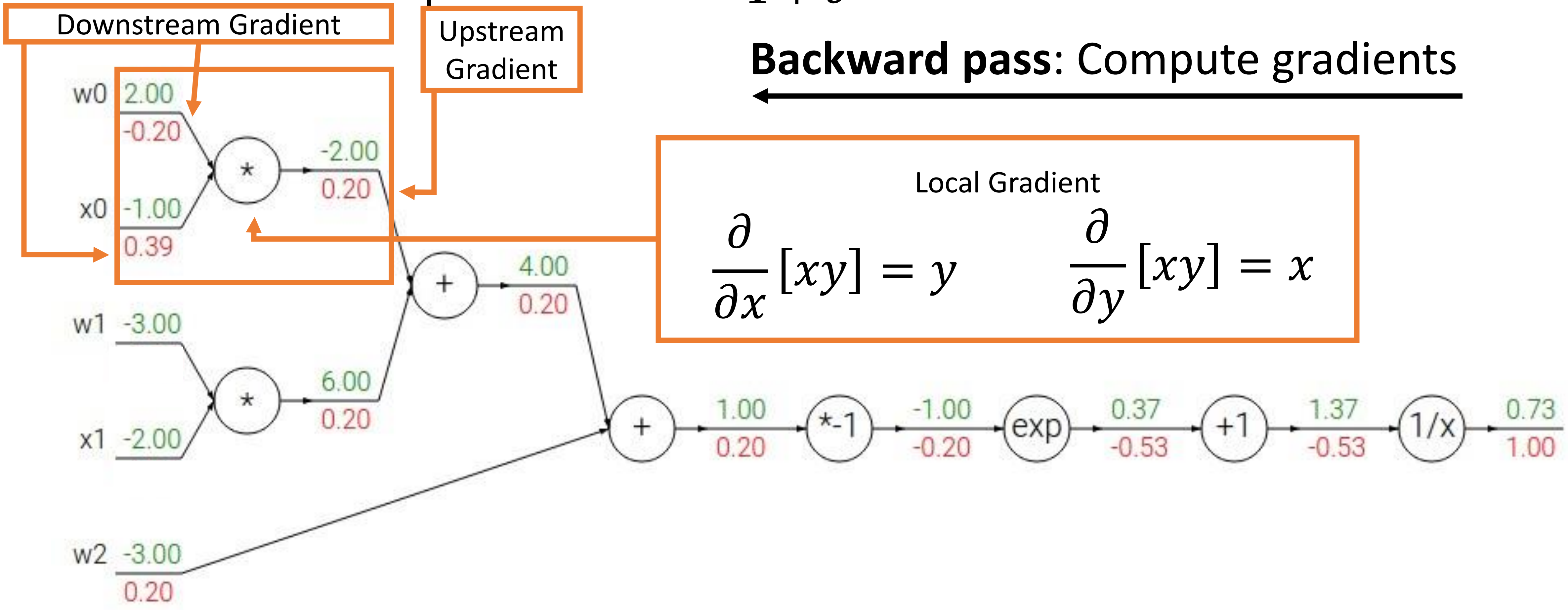
Local Gradient

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

Another Example

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

Backward pass: Compute gradients

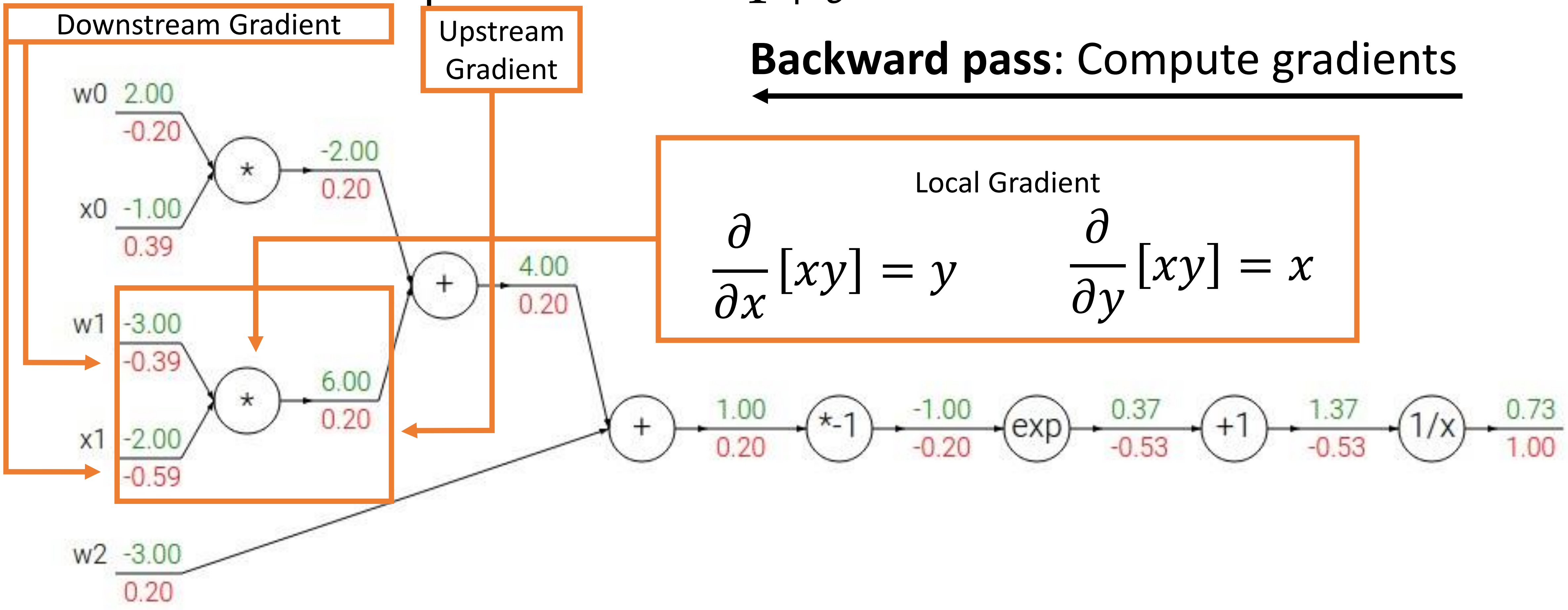


$$\frac{\partial}{\partial x}[xy] = y \qquad \frac{\partial}{\partial y}[xy] = x$$

Another Example

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

Backward pass: Compute gradients



Local Gradient

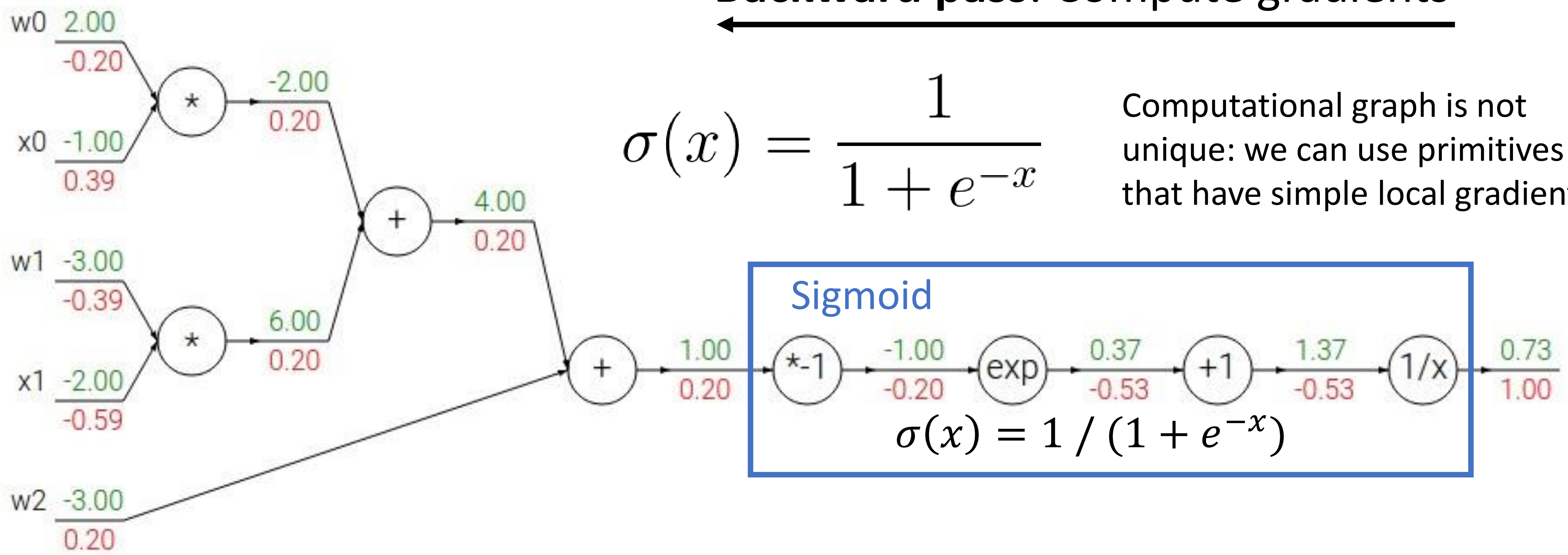
$$\frac{\partial}{\partial x}[xy] = y \qquad \frac{\partial}{\partial y}[xy] = x$$

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

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

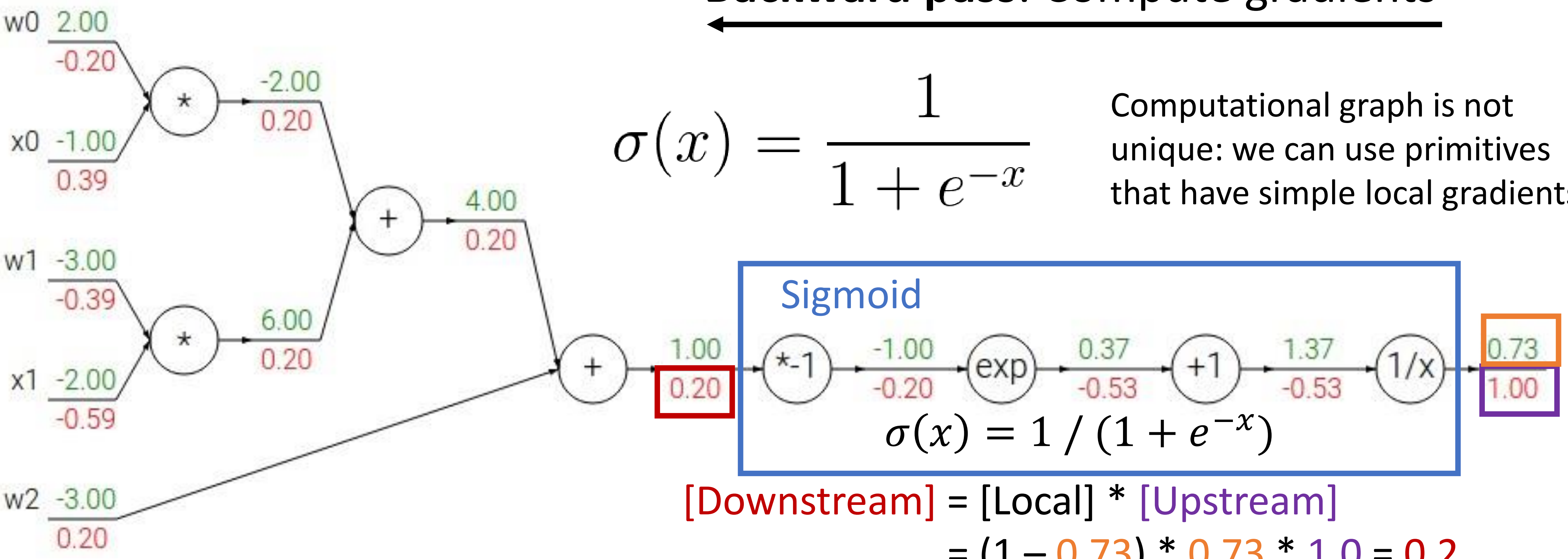
Computational graph is not unique: we can use primitives that have simple local gradients



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

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

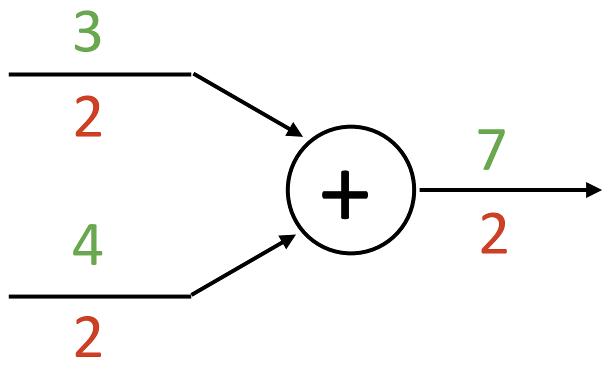
[Downstream] = [Local] * [Upstream]

= (1 – 0.73) * 0.73 * 1.0 = 0.2

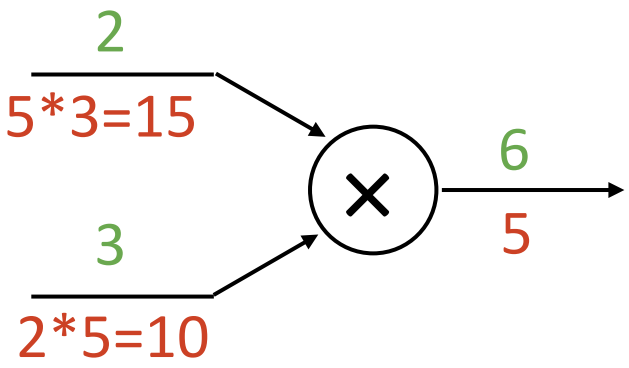
Sigmoid local gradient: $\frac{\partial}{\partial x}[\sigma(x)] = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$

Patterns in Gradient Flow

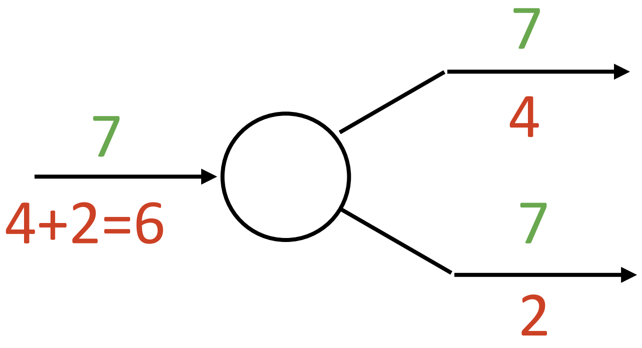
add gate: gradient distributor



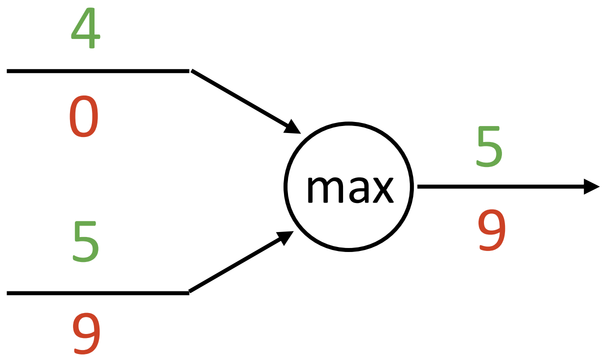
mul gate: “swap multiplier”



copy gate: gradient adder



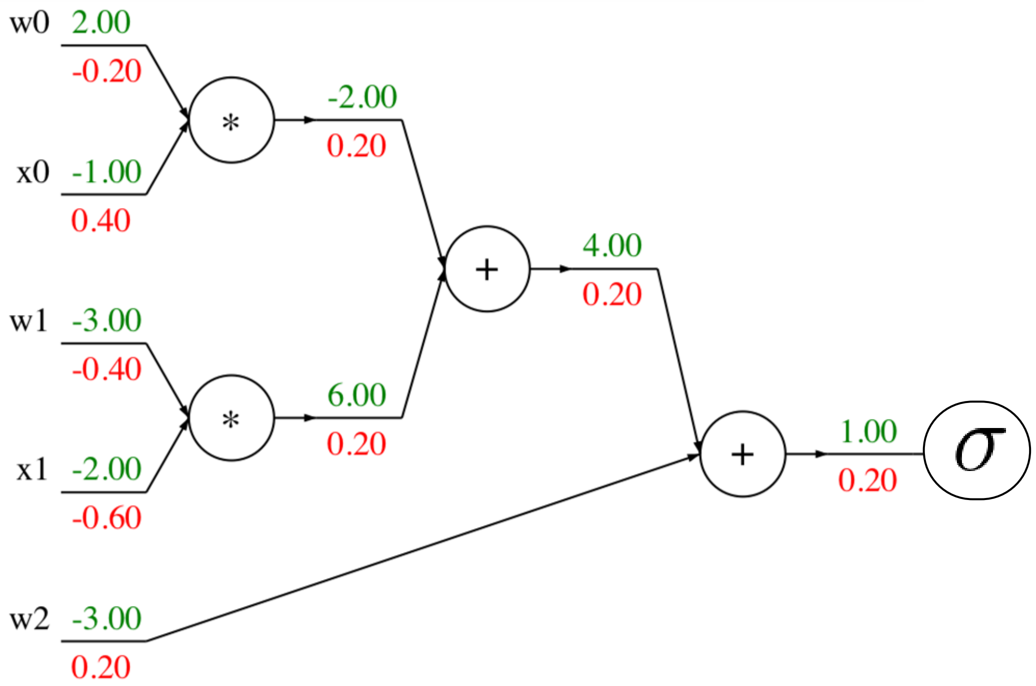
max gate: gradient router



Backprop Implementation: "Flat" gradient code:

Forward pass:
Compute output

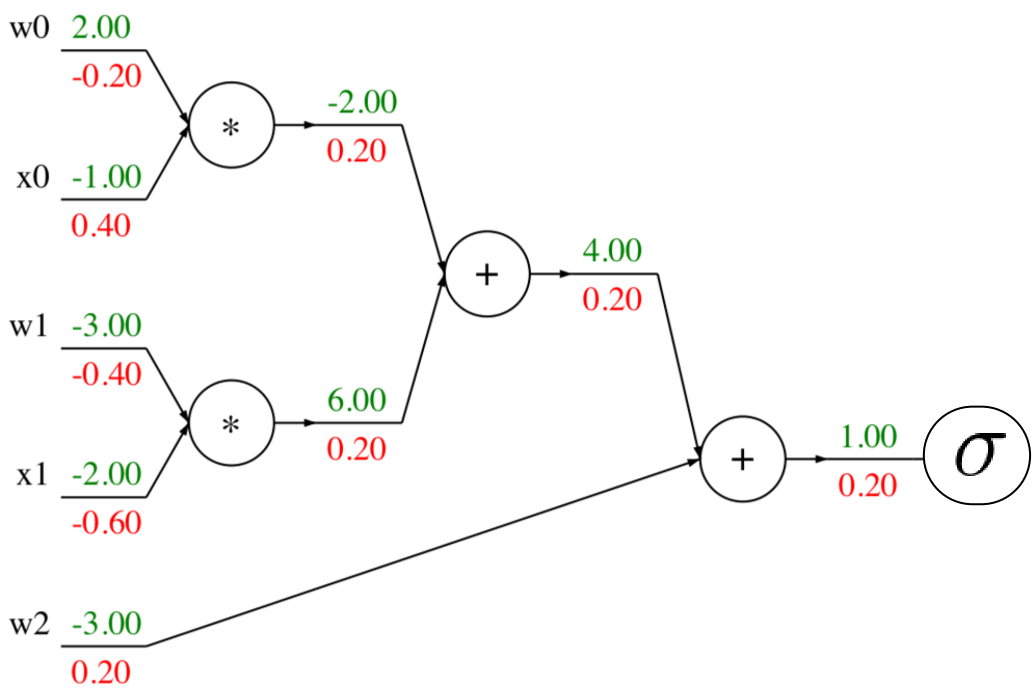
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```



Backprop Implementation: "Flat" gradient code:

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

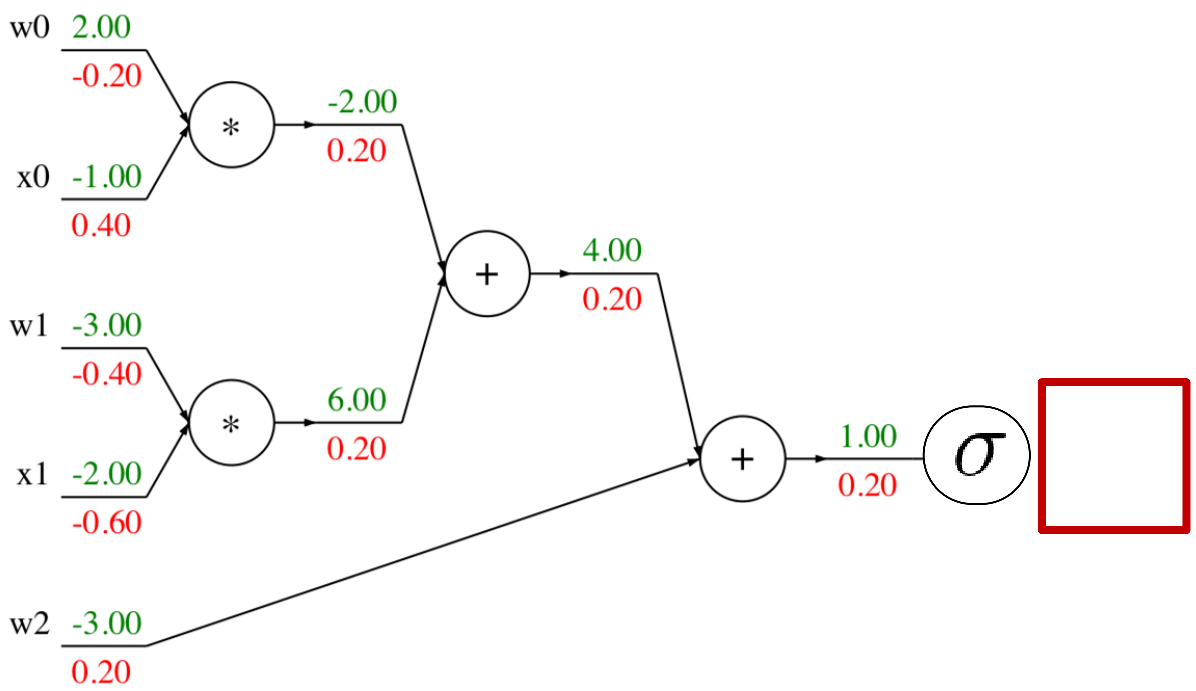
Forward pass: Compute output



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:



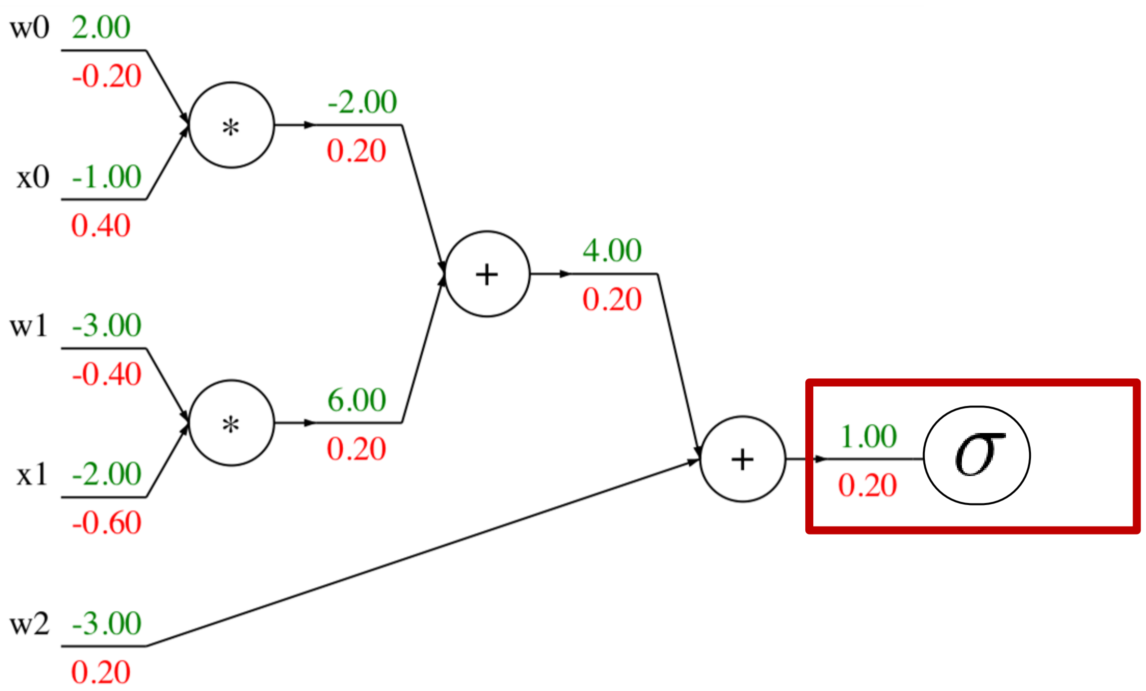
Forward pass:
Compute output

```
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:



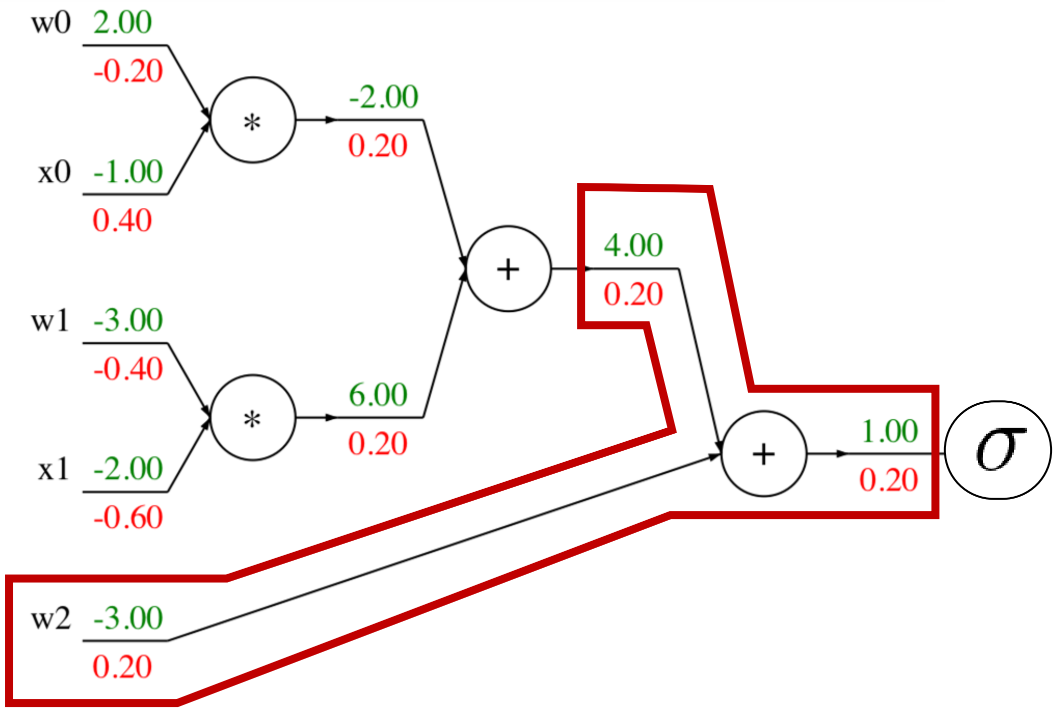
Forward pass: Compute output

```
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:



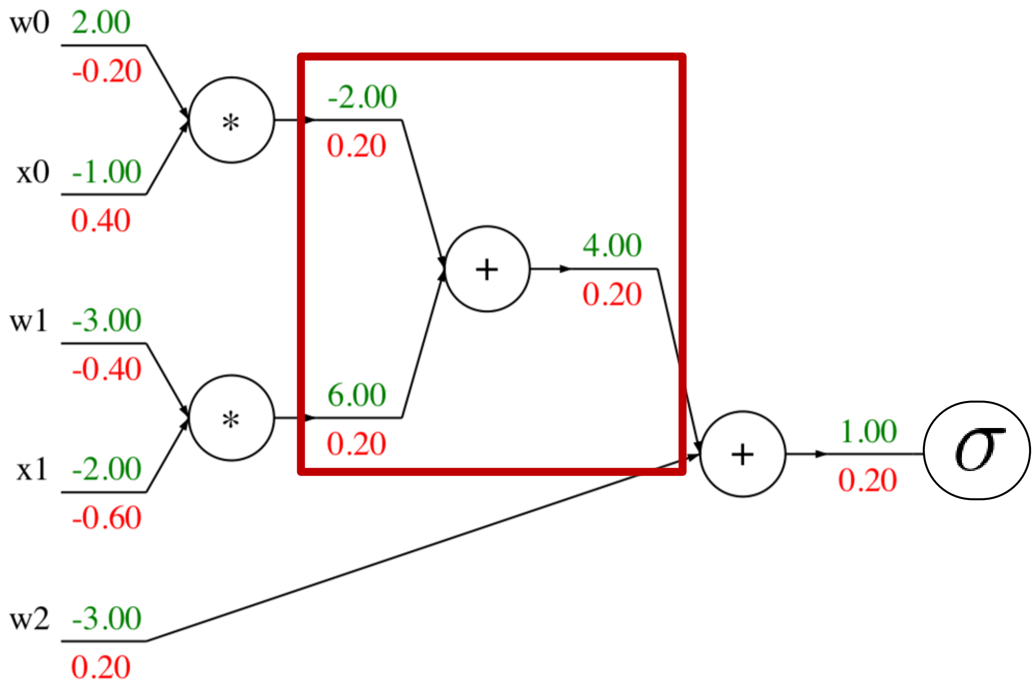
Forward pass: Compute output

```
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:



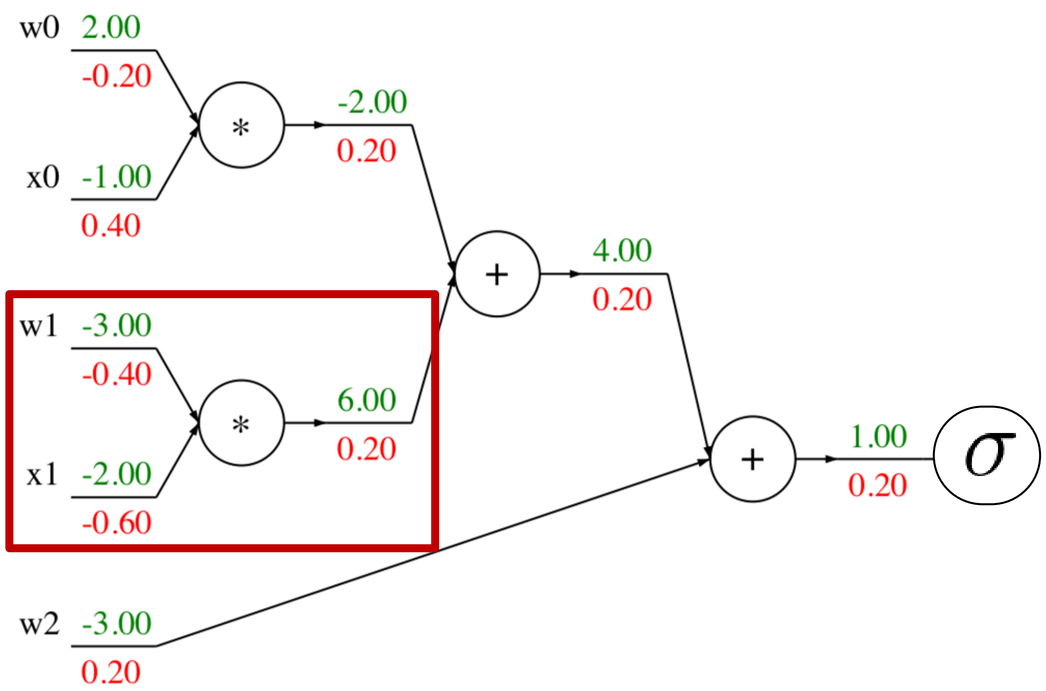
Forward pass:
Compute output

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

    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:



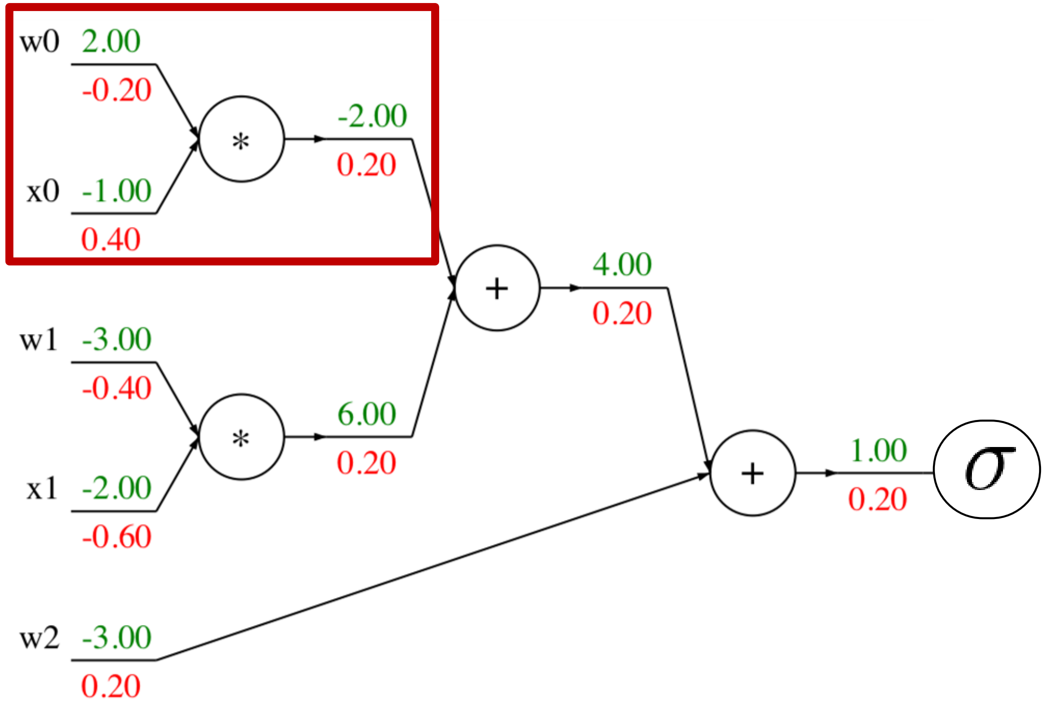
Forward pass: Compute output

```
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
```

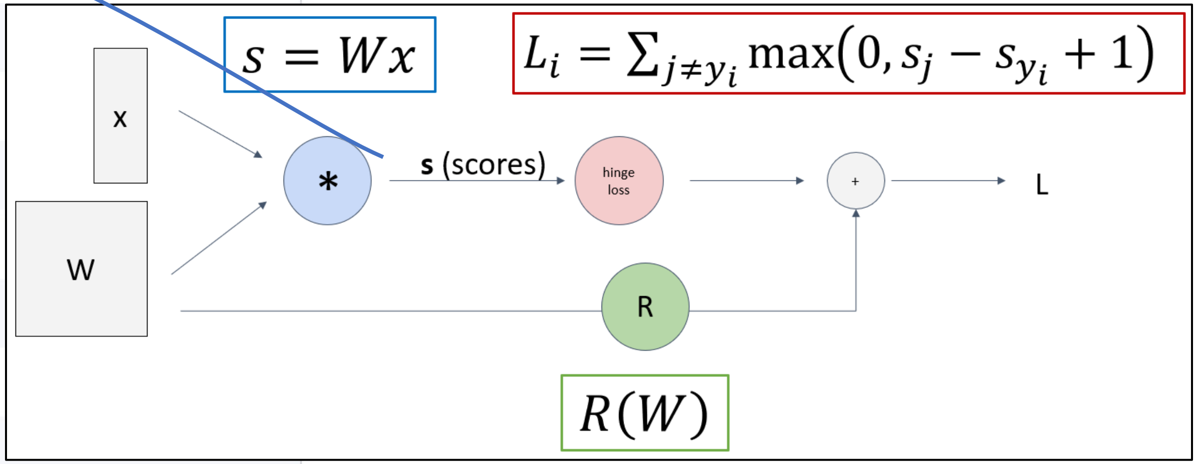
Multiply

“Flat” Backpropagation

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 = #...
```



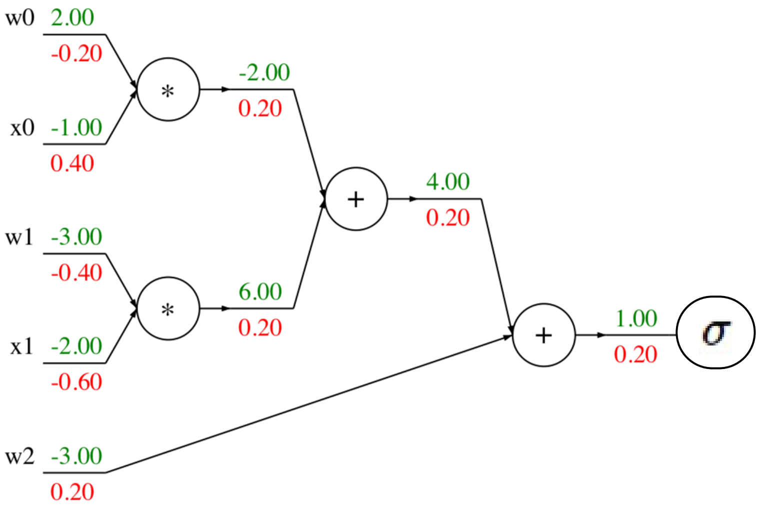
"Flat" Backpropagation

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 = #...
```

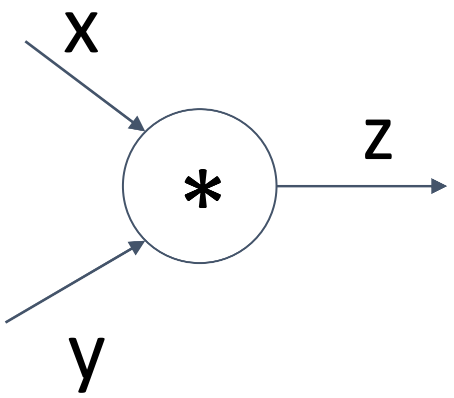
Backpropagation Implementation: Modular API



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

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

Example: PyTorch Autograd Functions



(x,y,z are scalars)

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

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Example: PyTorch operators

pytorch / pytorch — Watch 1,221 — Unstar 26,770 — Fork 6,340

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

Tree: 517c7c9861 — pytorch / aten / src / THNN / generic / — Create new file — Upload files — Find file — History

ezyang and facebook-github-bot Canonicalize all includes in PyTorch. (#14849) … Latest commit 517c7c9 on Dec 8, 2018

..

AbsCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
BCECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ClassNLLCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Col2Im.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
ELU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
FeatureLPPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
GatedLinearUnit.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
HardTanh.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Im2Col.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
IndexLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LeakyReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
LogSigmoid.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MSECriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiLabelMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
MultiMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
RReLU.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Sigmoid.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SmoothL1Criterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftMarginCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftPlus.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SoftShrink.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SparseLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAdaptiveAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAdaptiveMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialClassNLLCriterion.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialConvolutionMM.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialDilatedMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialFractionalMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialFullDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialMaxUnpooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialReflectionPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialUpSamplingBilinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
SpatialUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
THNN.h	Canonicalize all includes in PyTorch. (#14849)	4 months ago
Tanh.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalReflectionPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalRowConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalUpSamplingLinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
TemporalUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAdaptiveAveragePoolin…	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAdaptiveMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricAveragePooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricConvolutionMM.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricDilatedMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricFractionalMaxPooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricFullDilatedConvolution.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricMaxUnpooling.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricReplicationPadding.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricUpSamplingNearest.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
VolumetricUpSamplingTrilinear.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago
linear_upsampling.h	Implement nn.functional.interpolate based on upsample. (#8591)	9 months ago
pooling_shape.h	Use integer math to compute output size of pooling operations (#14405)	4 months ago
unfold.c	Canonicalize all includes in PyTorch. (#14849)	4 months ago

PyTorch sigmoid layer

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

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

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

#endif
```

[Source]

PyTorch sigmoid layer

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

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

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

#endif
```

Forward

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

```
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))));
```

Source

PyTorch sigmoid layer

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

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

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

#endif
```

Forward

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

Backward

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

Source

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?

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

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?

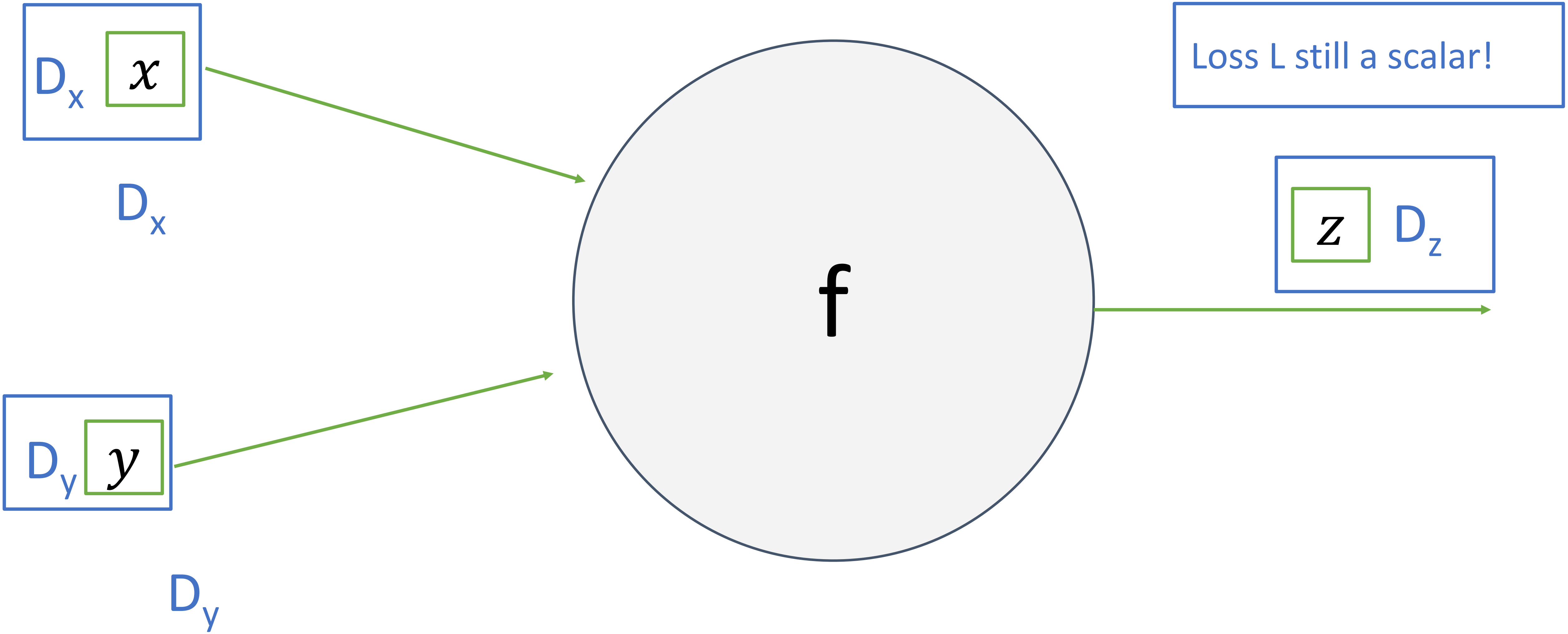
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

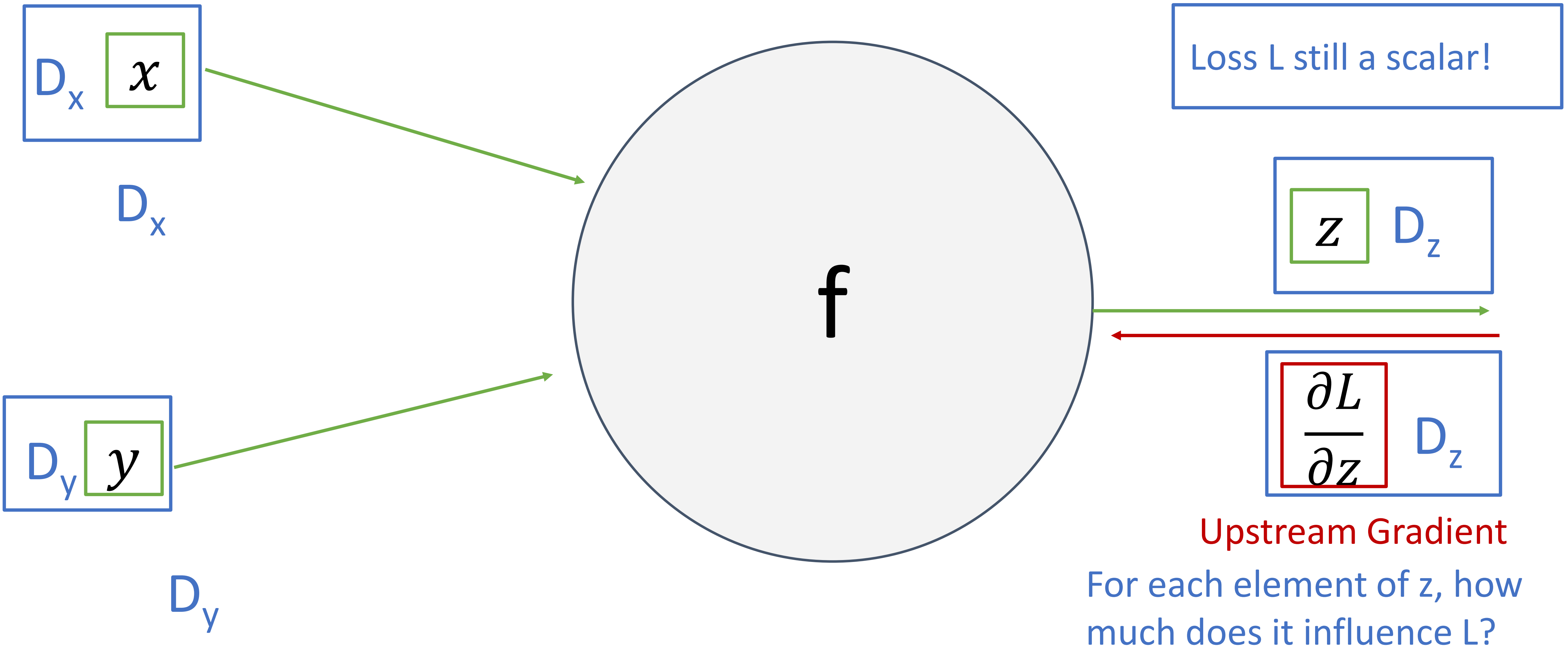
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

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

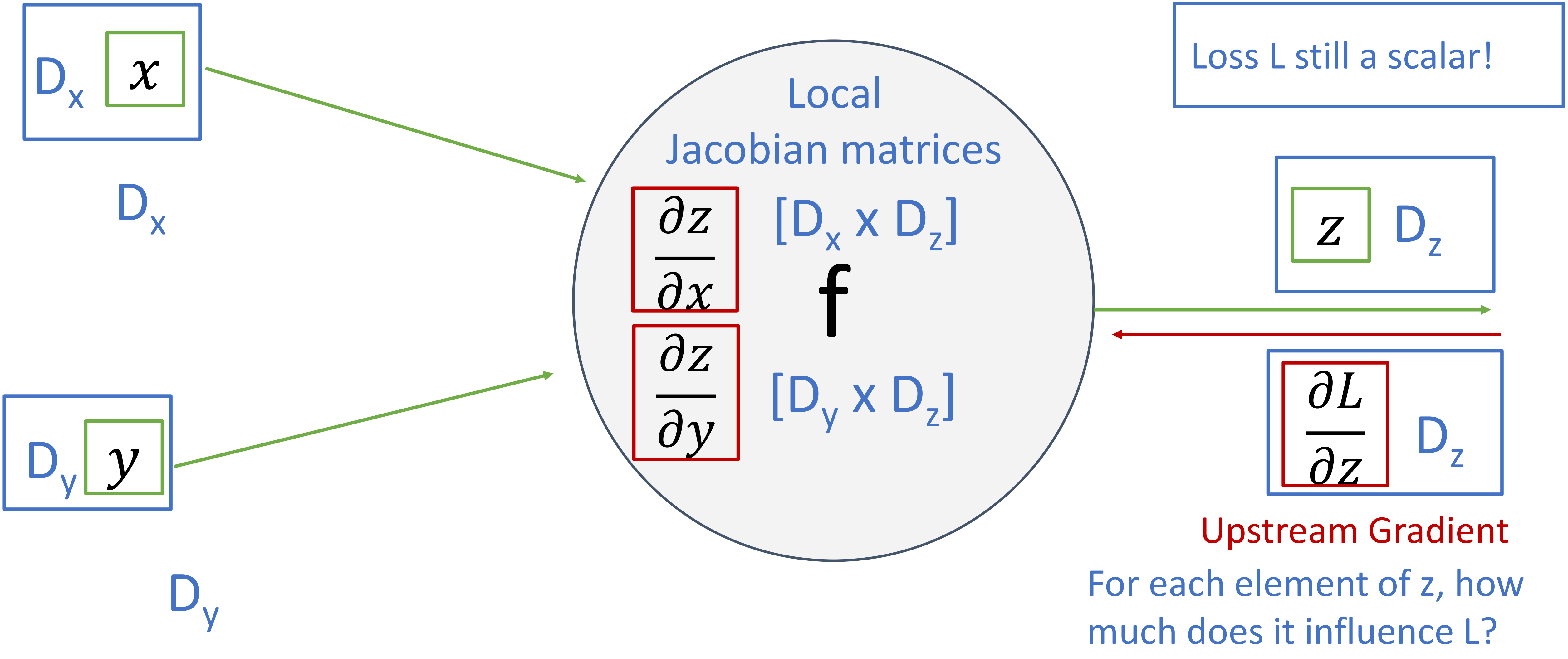
Backprop with Vectors



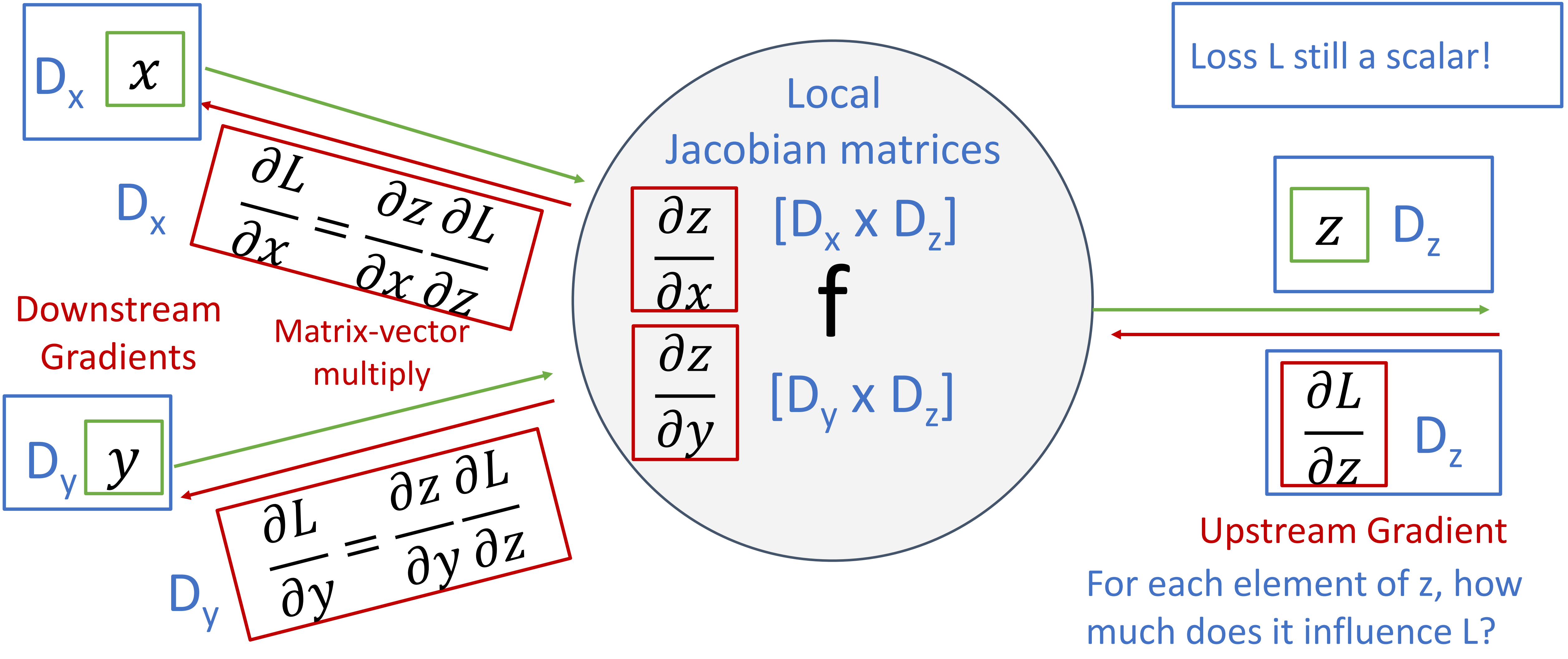
Backprop with Vectors



Backprop with Vectors



Backprop with Vectors



Backprop with Vectors

4D input x:

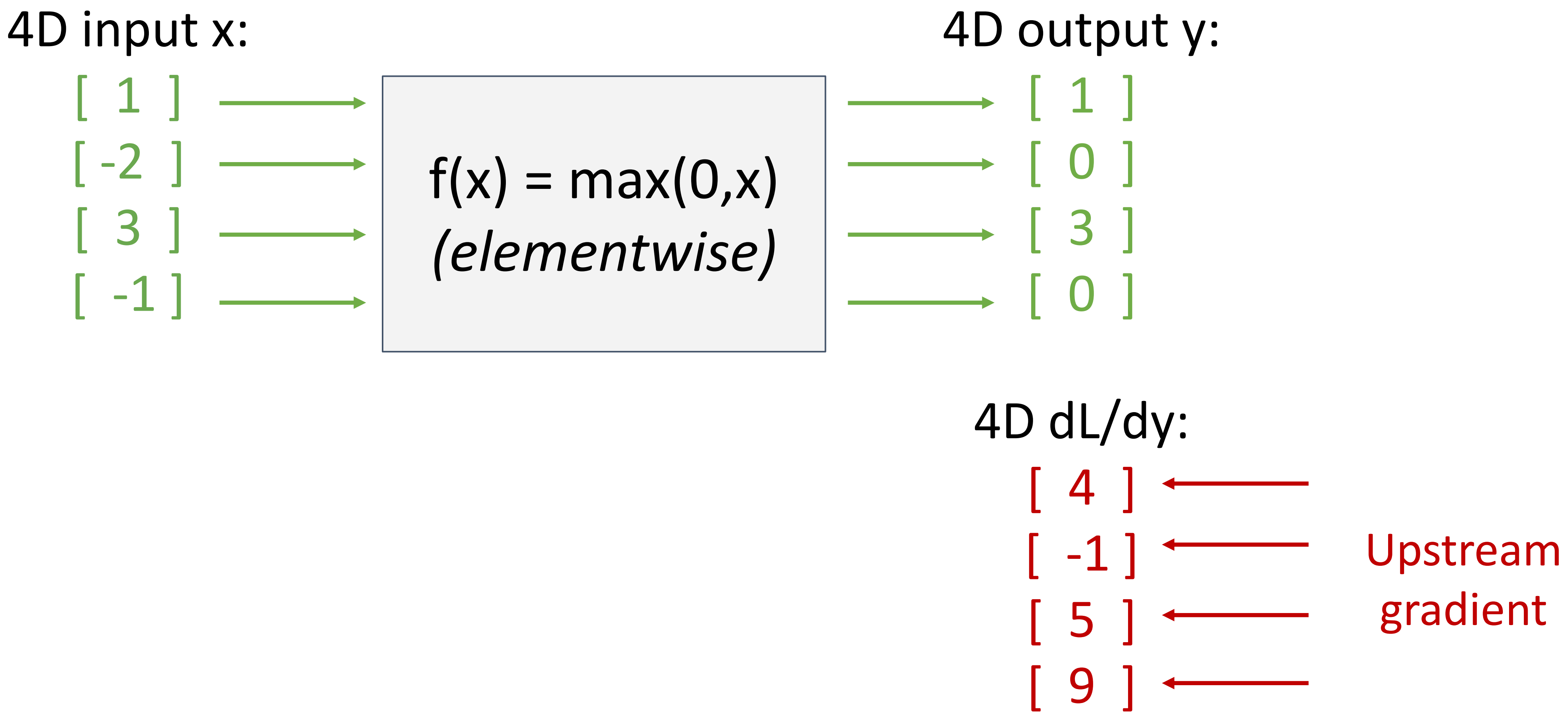
[1]
[-2]
[3]
[-1]

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

4D output y:

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

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]

Jacobian dy/dx

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

4D dL/dy:

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

Upstream gradient

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:

[dy/dx] [dL/dy]

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

4D dL/dy:

Upstream gradient

Backprop with Vectors

4D input x:

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

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

4D output y:

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

4D dL/dx:

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

[dy/dx] [dL/dy]

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

4D dL/dy:

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

Upstream gradient

Backprop with Vectors

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

4D input x:

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

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

4D output y:

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

4D dL/dx:

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

[dy/dx] [dL/dy]

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

4D dL/dy:

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

Upstream gradient

Backprop with Vectors

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:

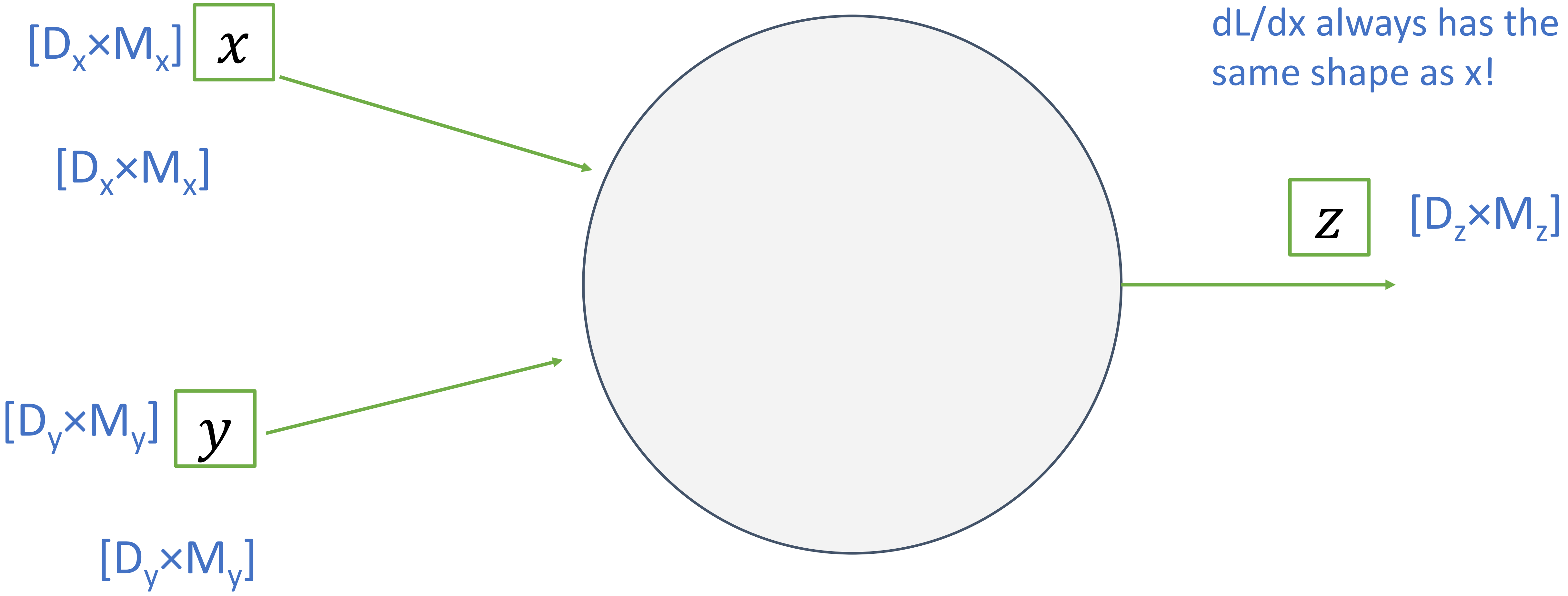
[4]
[-1]
[5]
[9]

Upstream gradient

Backprop with Matrices (or Tensors):

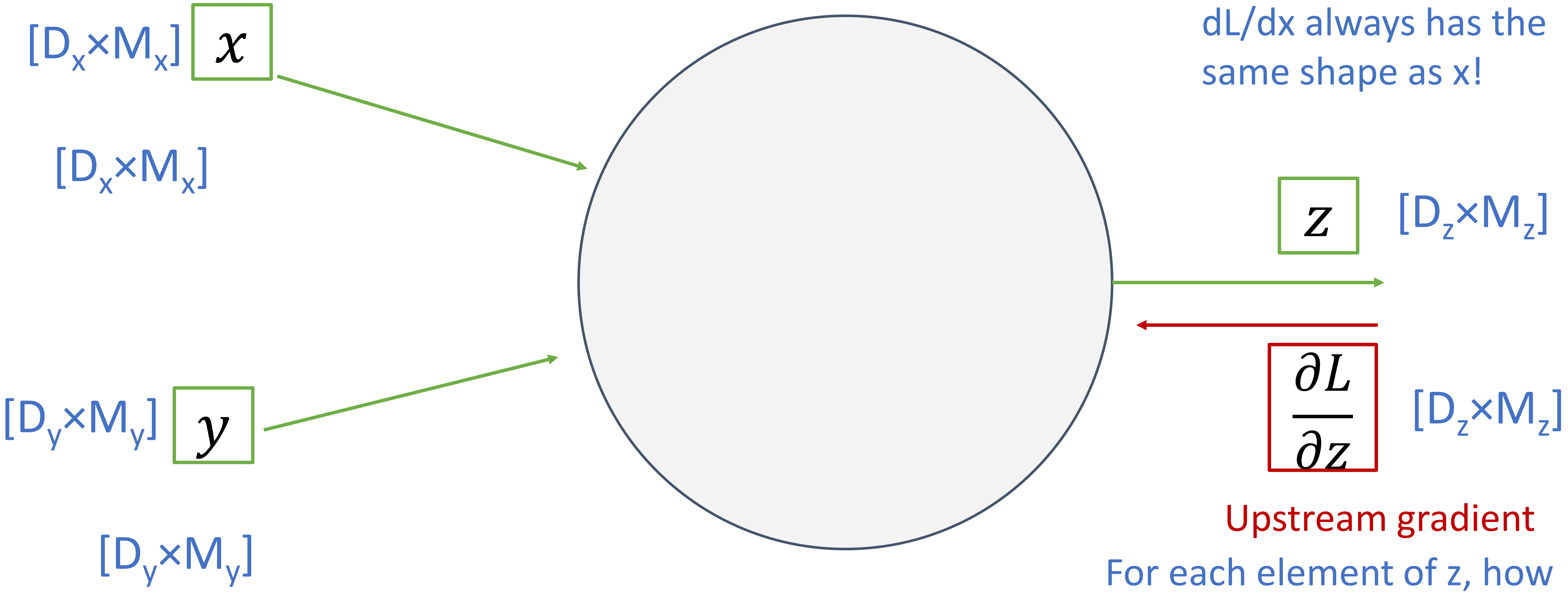
Loss L still a scalar!

dL/dx always has the same shape as x!



Backprop with Matrices (or Tensors):

Loss L still a scalar!



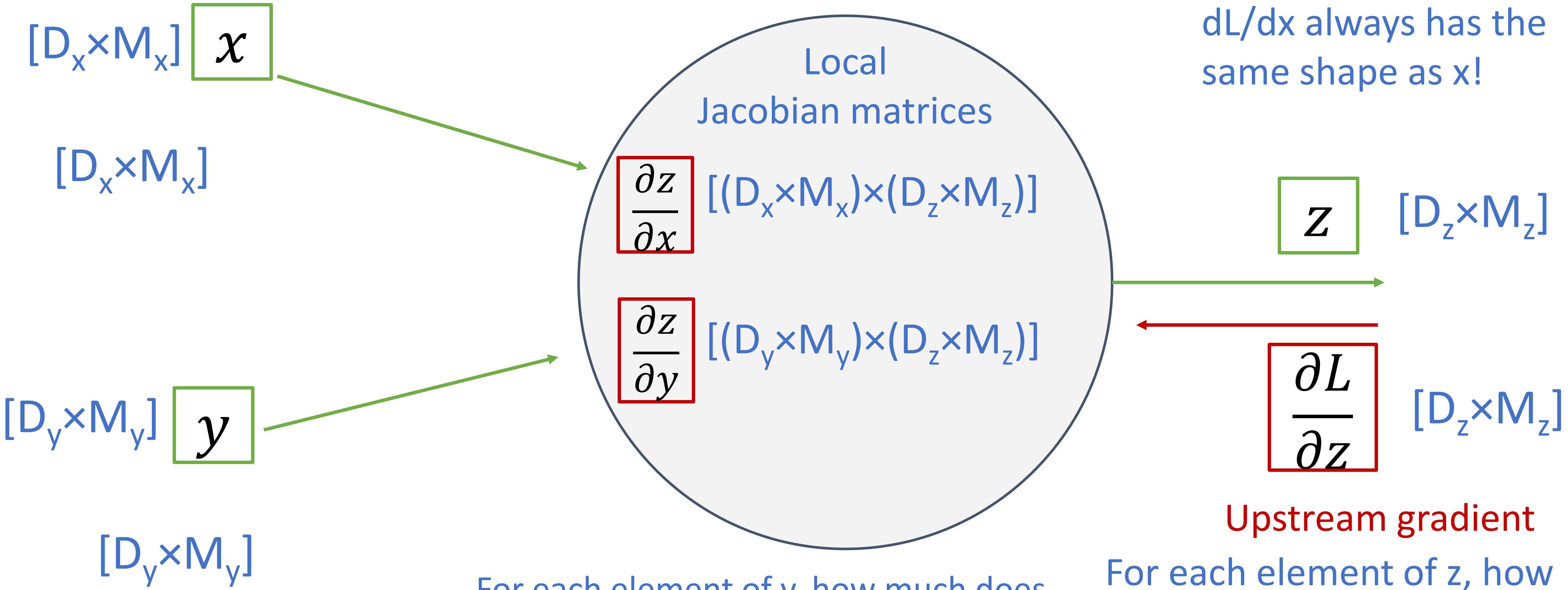
Upstream gradient

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

Backprop with Matrices (or Tensors):

Loss L still a scalar!

dL/dx always has the same shape as x!



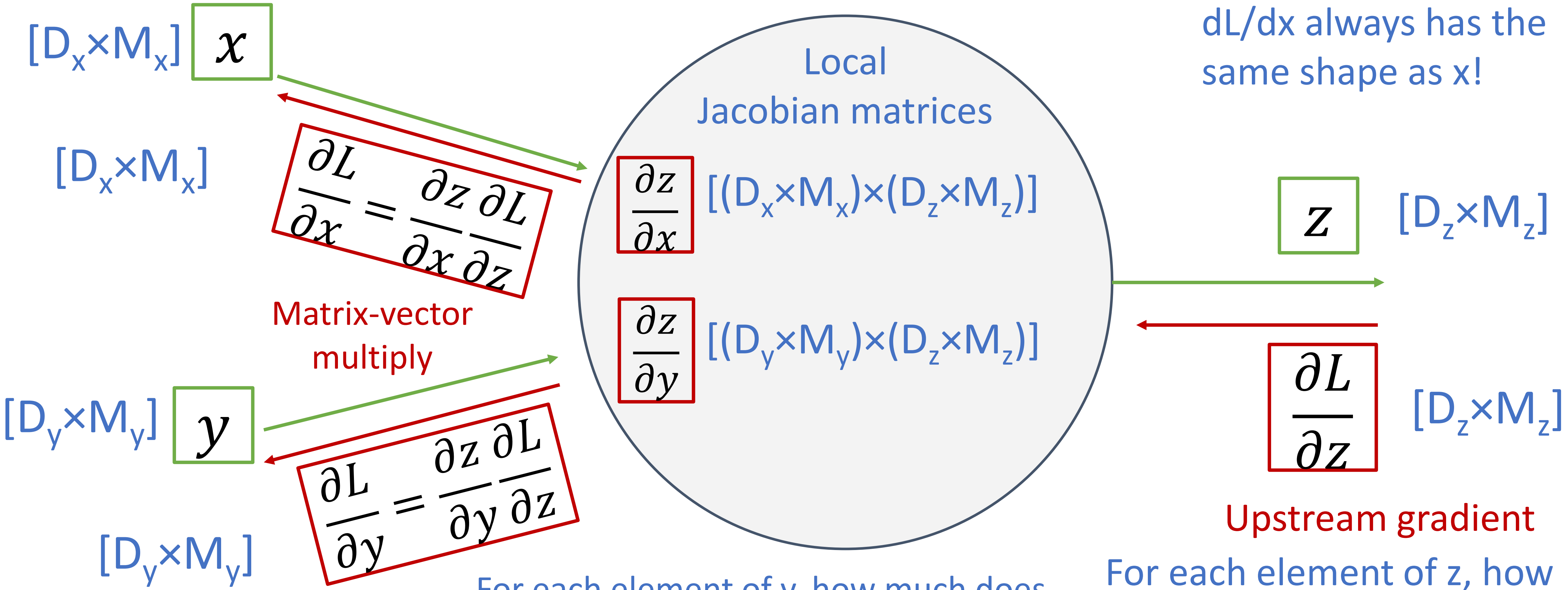
For each element of y, how much does it influence each element of z?

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

Backprop with Matrices (or Tensors):

Loss L still a scalar!

dL/dx always has the same shape as x!



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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[? ? ?]
[? ? ?]

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:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

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:

$dy/dx_{1,1}$

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

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

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

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:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

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]

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

$dL/dx_{1,1}$

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

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]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

$dL/dx_{1,1}$

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

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

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:

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

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]

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

dL/dx: [N×D]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

$dL/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 $\boxed{2}$]

w: [D×M]
[3 2 1 -1]
[2 1 3 2]
[$\boxed{3\ 2\ 1\ -2}$]

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

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

dL/dx: [N×D]
[0 ? ?]
[? ? $\boxed{-30}$]

Local Gradient Slice:

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

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

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

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

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

$dL/dx_{i,j}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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 x D]

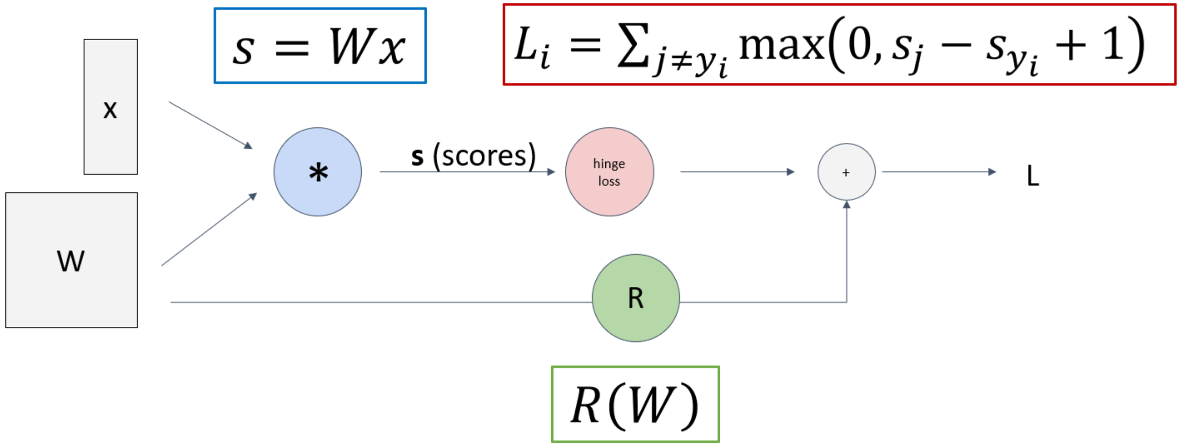
$dL/dw = x^T\, (dL/dy)$

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

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

Summary

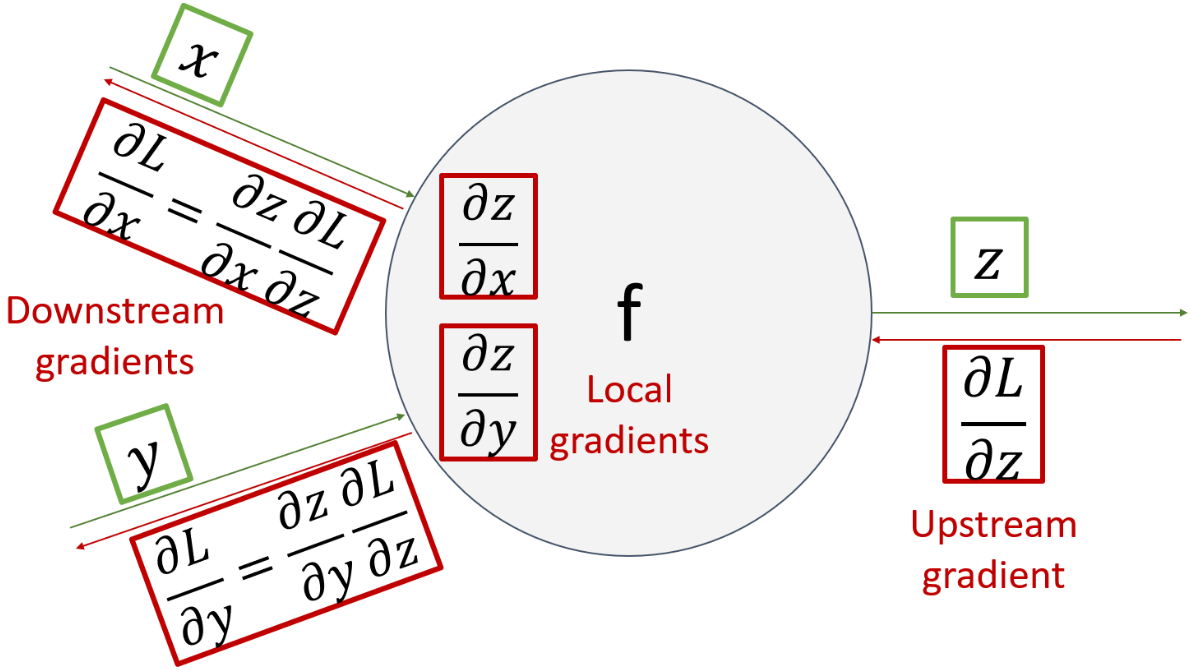
Represent complex expressions as **computational graphs**



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

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

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

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

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

Next:
Convolutional Neural Networks