CS M146: Introduction to Machine Learning
Neural Networks - Backpropagation

Aditya Grover

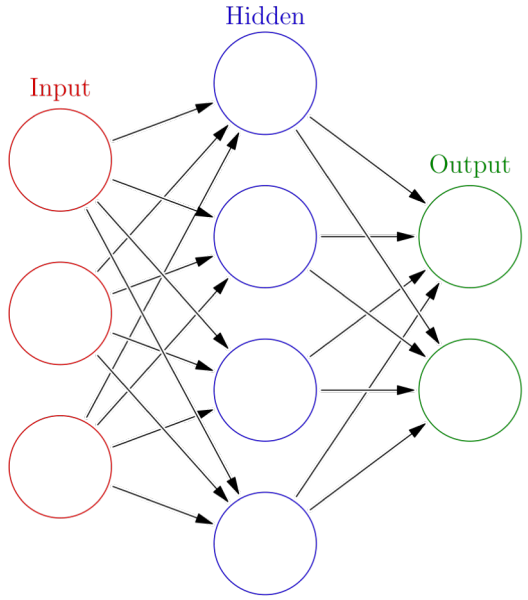
The instructor gratefully acknowledges Justin Johnson for some of the materials and organization used in these slides.

https://aditya-grover.github.io/ @adityagrover_

Artificial Neural Networks

An **artificial neural network** (ANN) is a representation of hypothesis class with 3 special features:

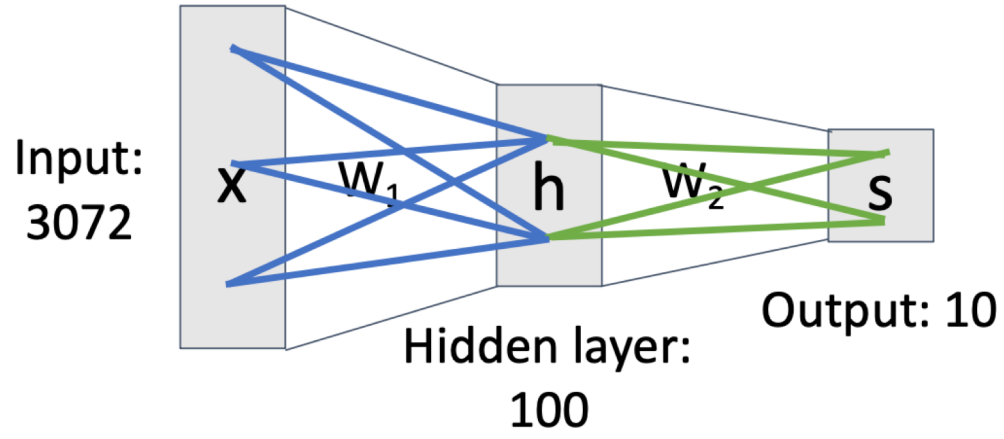
- **Neurons:** Artificial neurons (nodes) that process and transmit information.
- **Layers:** Neurons are organized into layers, including an input layer, one or more hidden layers, and an output layer.
- **Connections:** Neurons are interconnected through weighted connections that transmit signals.



Neural Net → Forward + Backward Pass

$$\boldsymbol{h} = \max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{s} = W_2 \boldsymbol{h} + \boldsymbol{b}_2$$
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{s})$$

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_c^{(i)} \log f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)$$



Forward pass computes outputs (including h, s, f, J)

Backward pass computes gradients to minimize loss

Representing Functions via Graphs

- Consider a function:

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

- Can decompose f via 2 simpler operations:

$$q(x, y) = x + y \text{ [Add]}$$

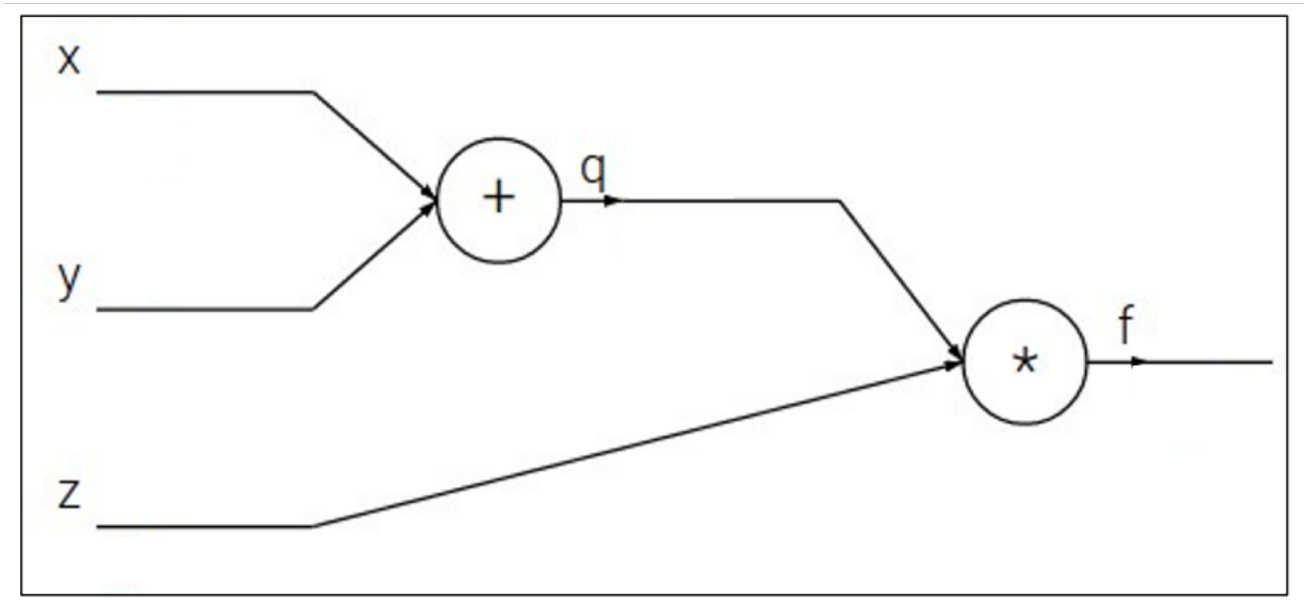
$$f(x, y, z) = q(x, y) * z \text{ [Multiply]}$$

Representing Functions via Graphs

- Consider a function:

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

- Can draw the computation of f as a **directed acyclic graph**



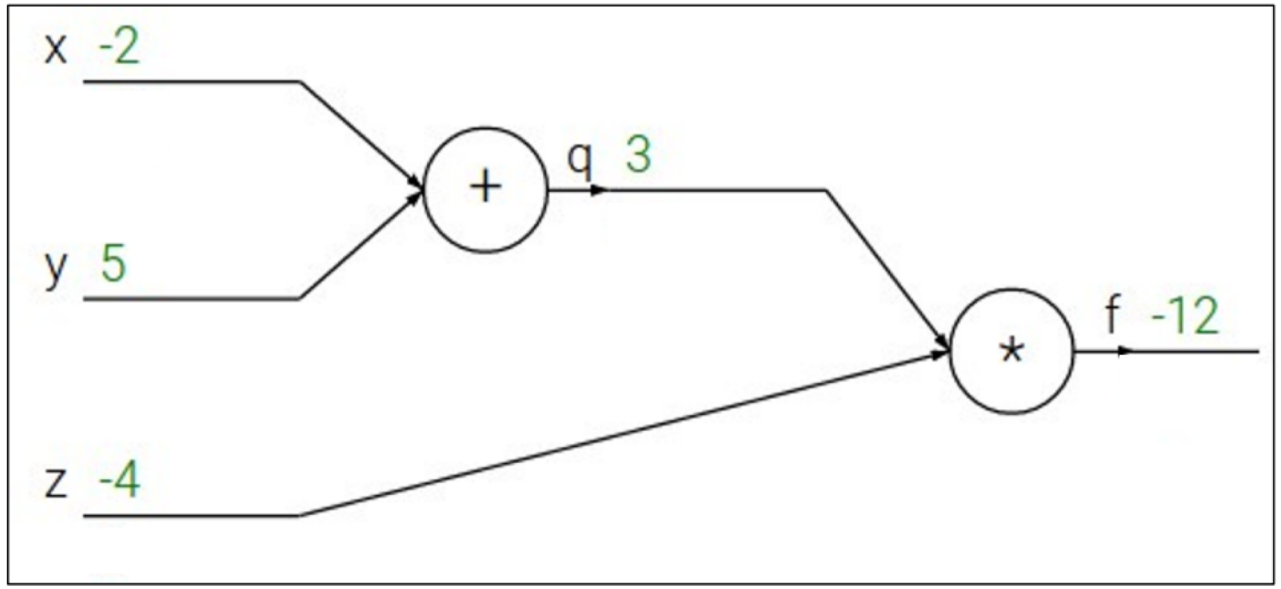
Evaluating Functions – Forward Pass

Function evaluation performs a **forward pass** from **left to right**

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

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

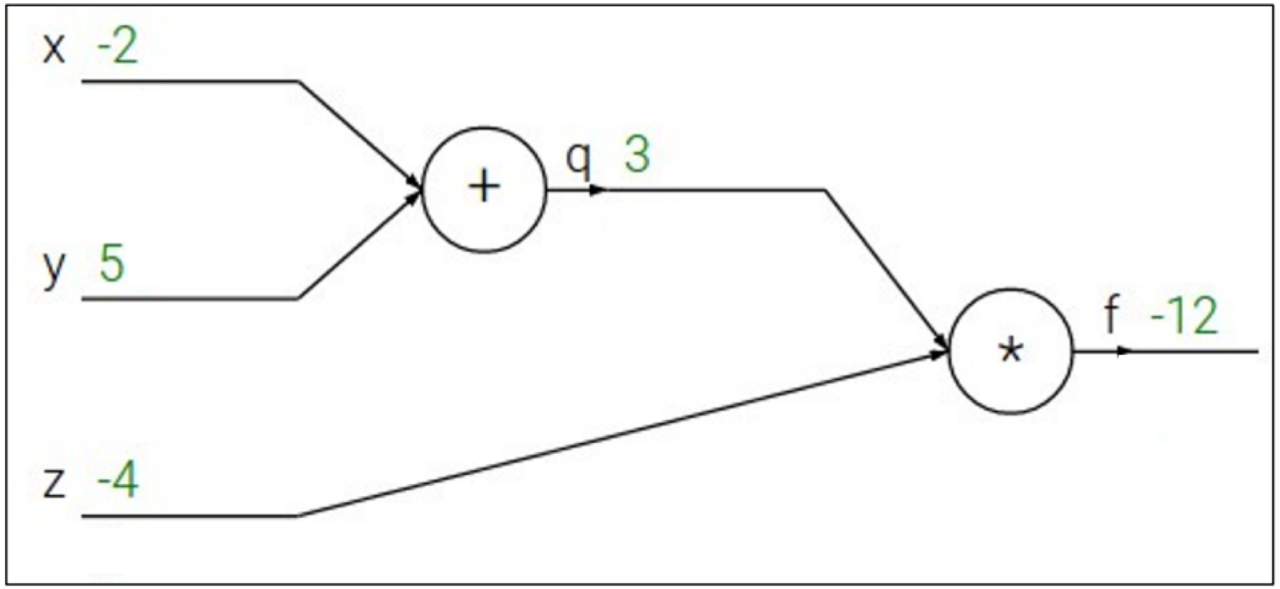
$$q(x, y) = x + y$$
$$f(x, y, z) = q(x, y) * z$$



Optimize Functions – Backward Pass

To optimize functions, we compute derivatives in **backward pass** from **right to left** a.k.a. backpropagation

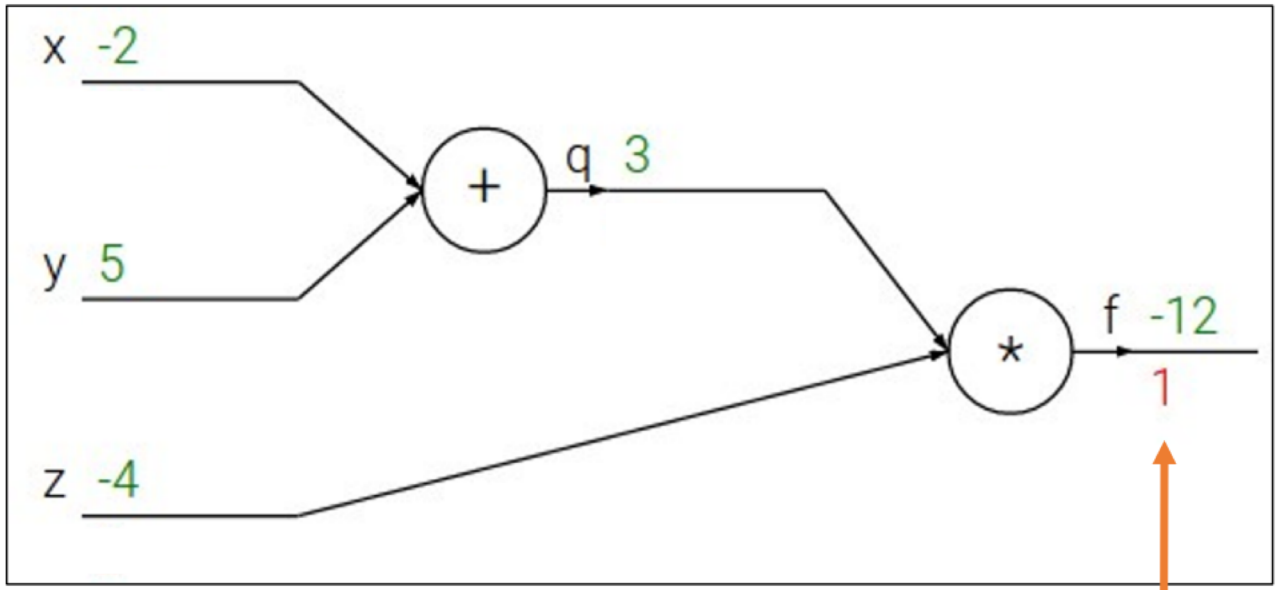
$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

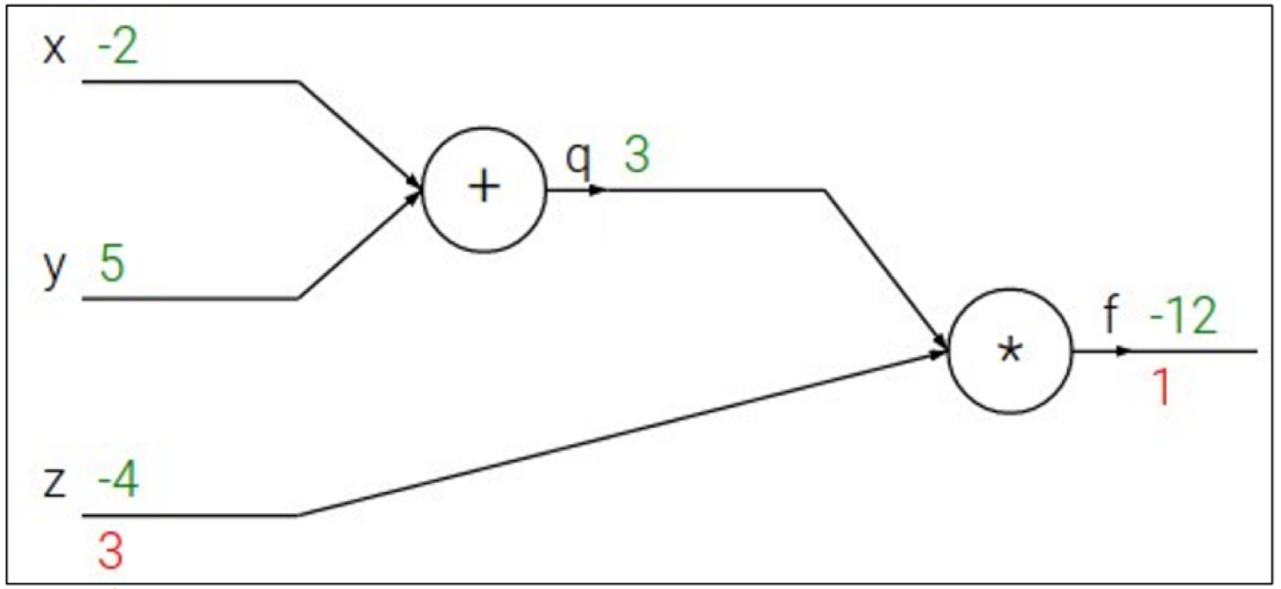
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

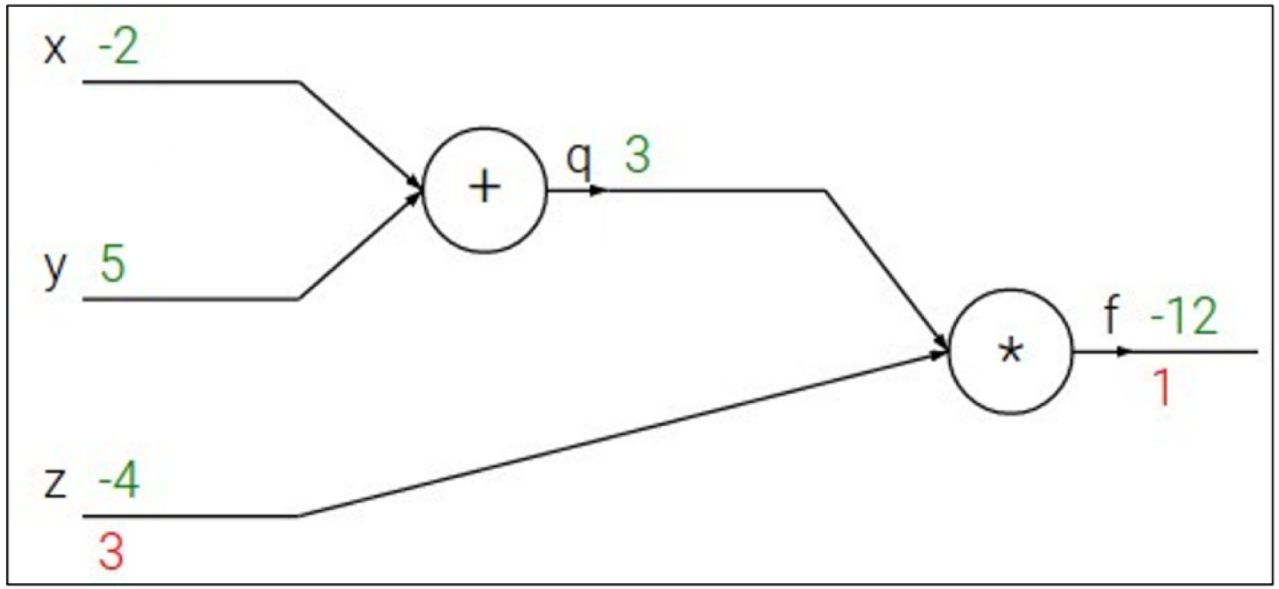
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

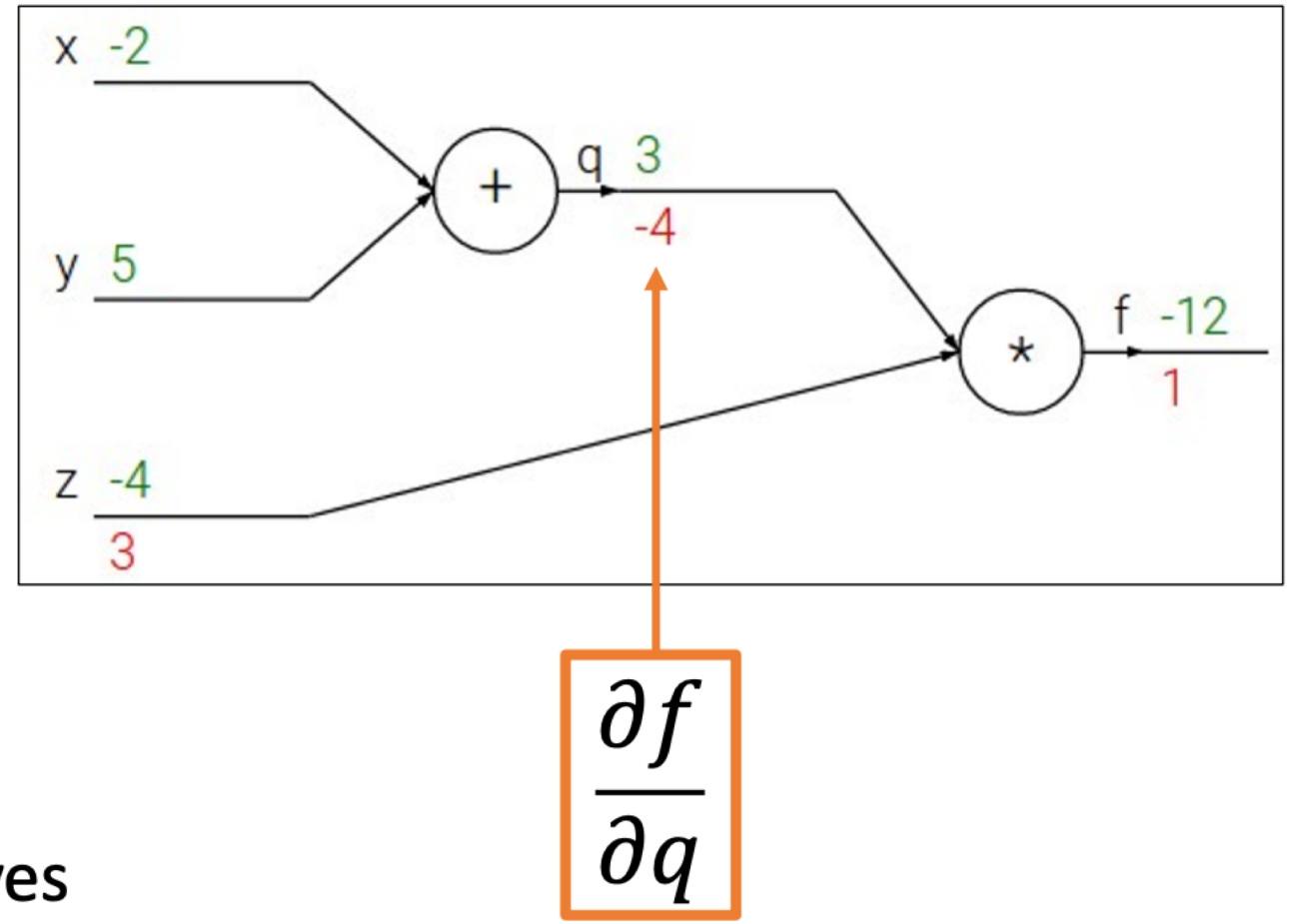
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

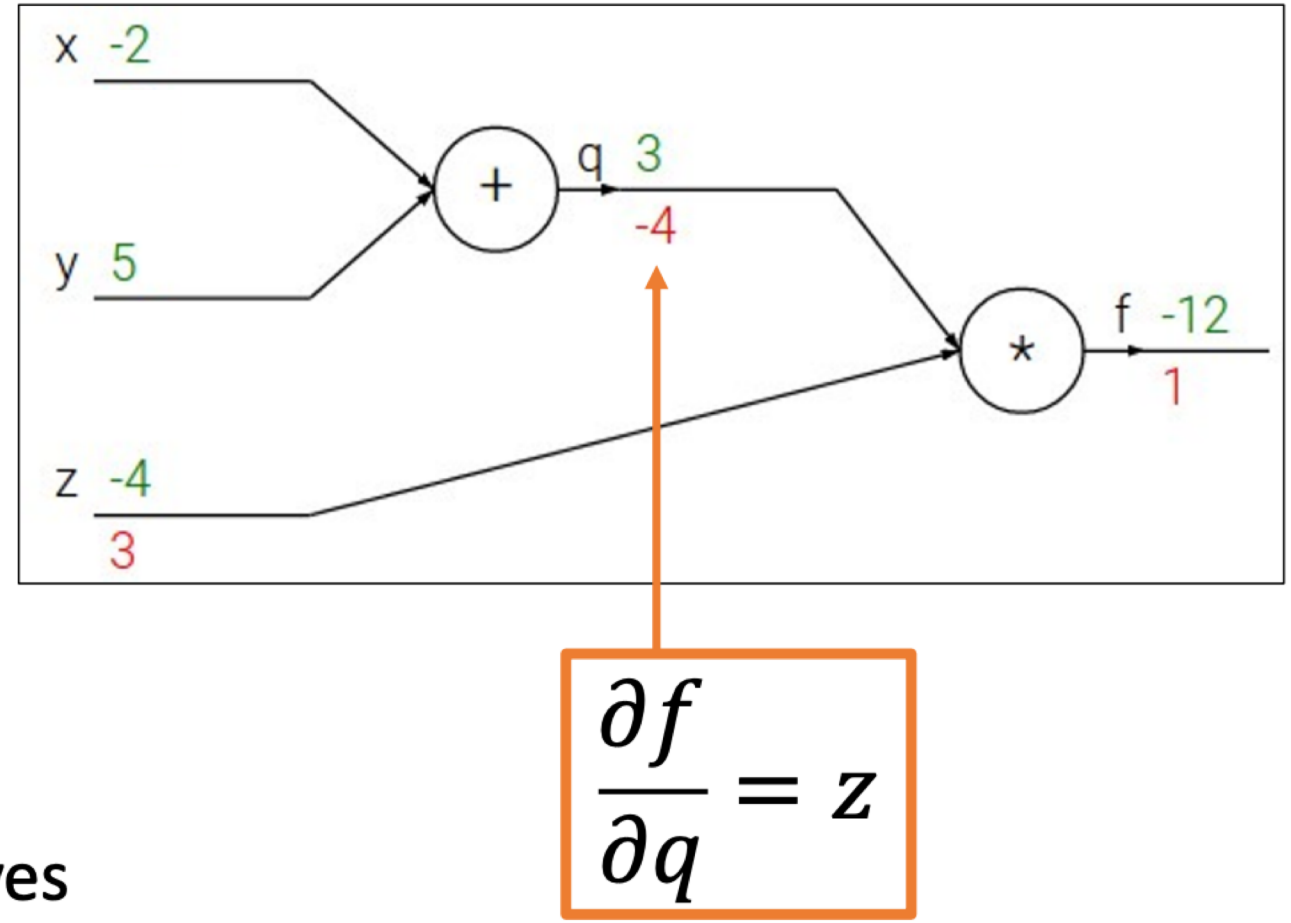
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

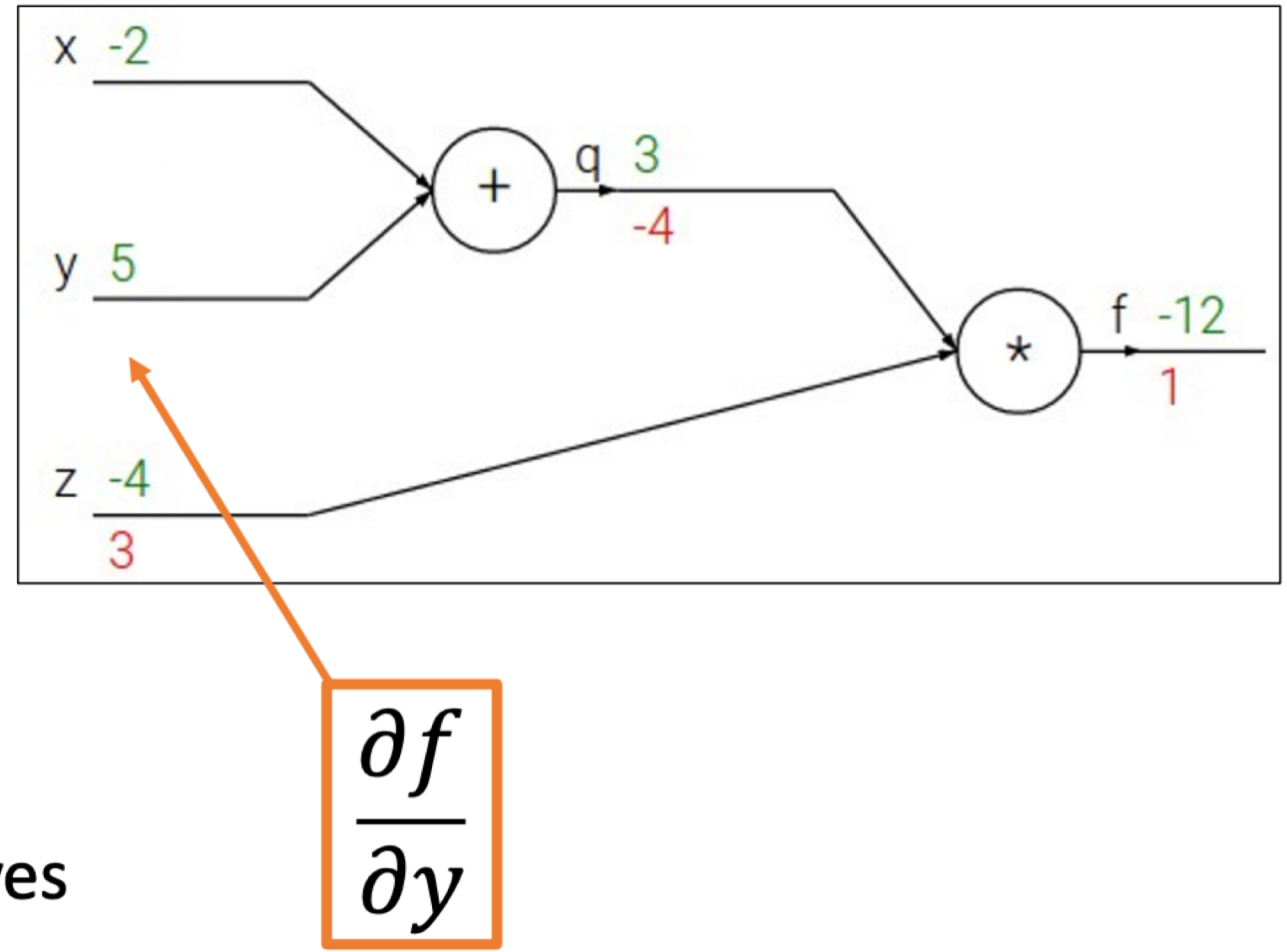
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

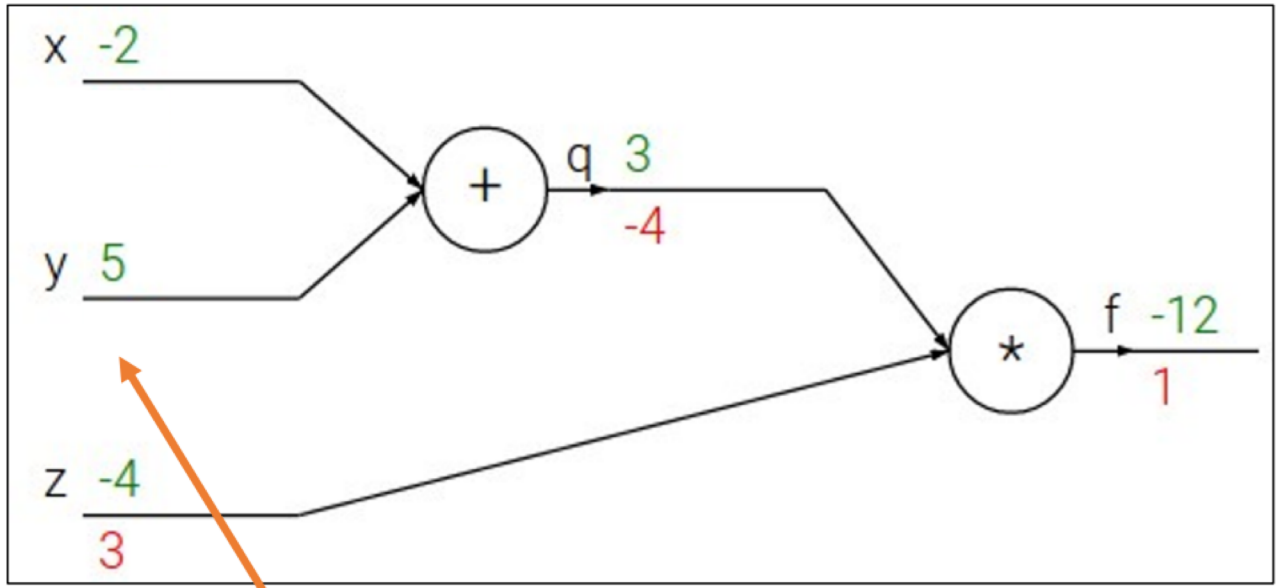
2. Backward pass: Compute derivatives

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

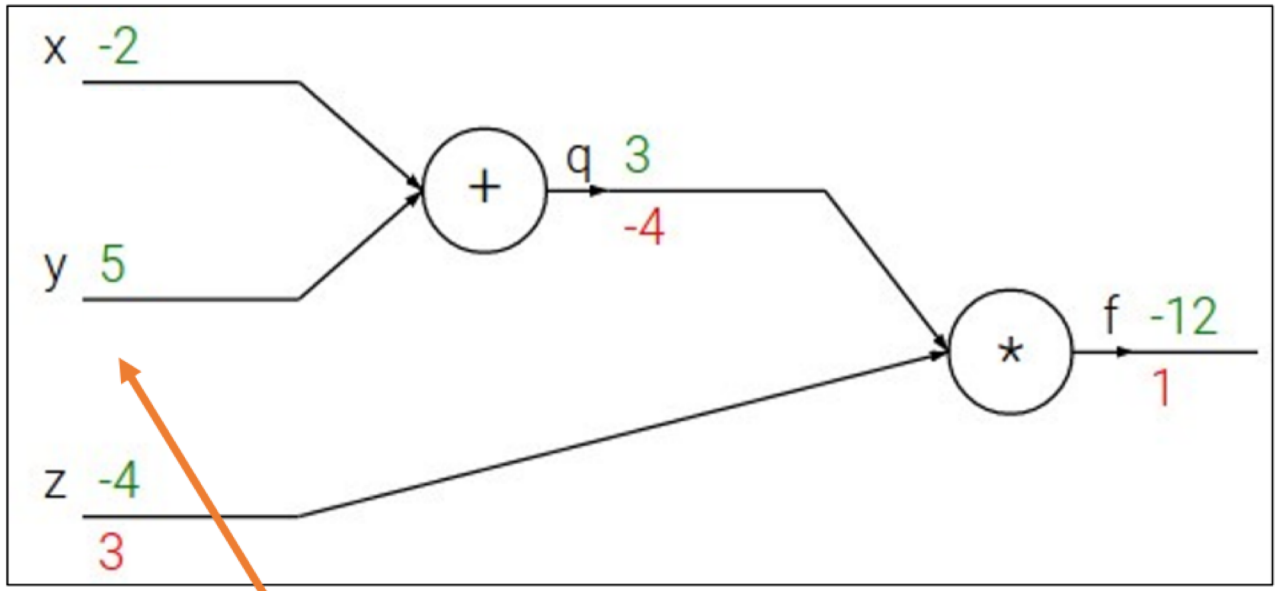
Chain Rule

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

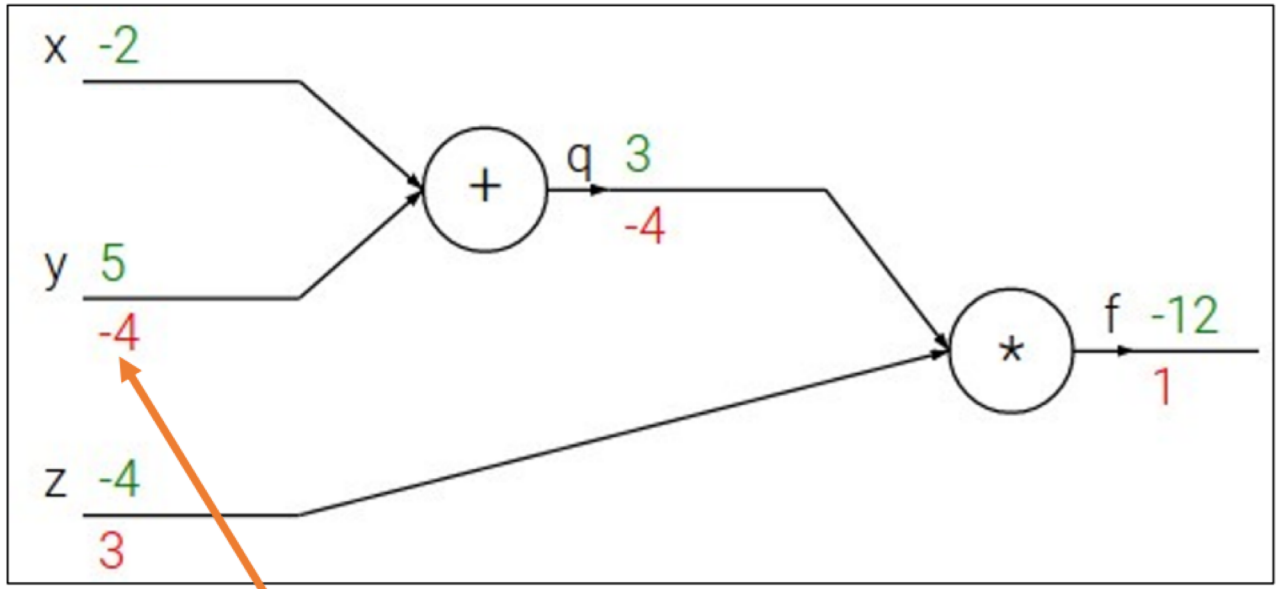
$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

Chain Rule

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

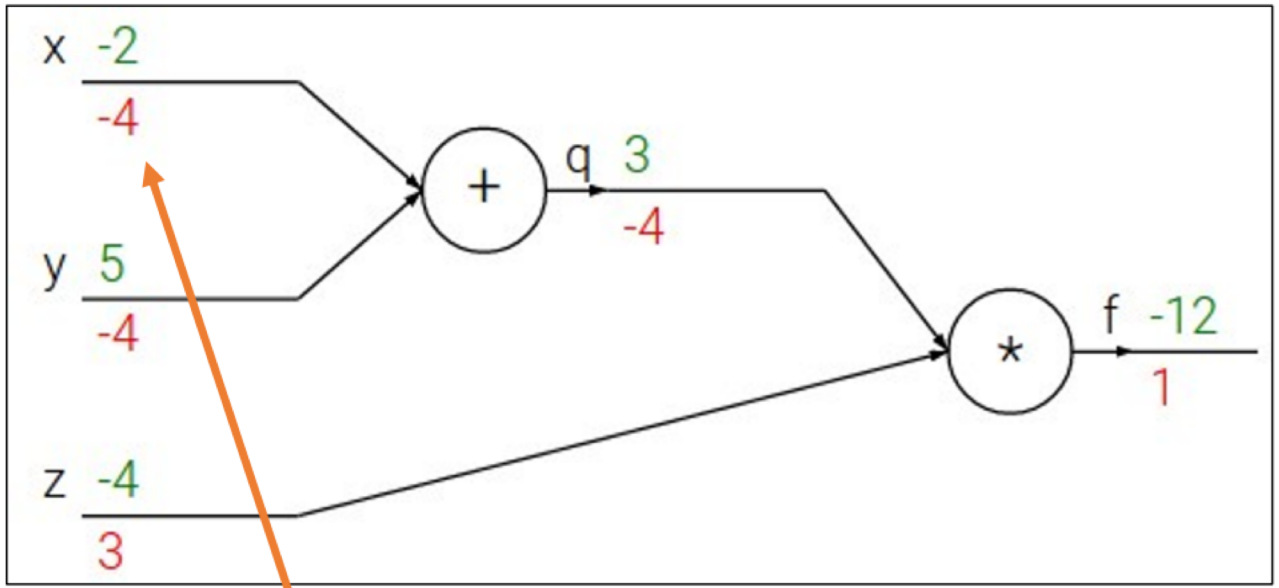
$$\frac{\partial q}{\partial y} = 1$$

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

Backpropagation: Simple Example

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

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



1. Forward pass: Compute outputs

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

2. Backward pass: Compute derivatives

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

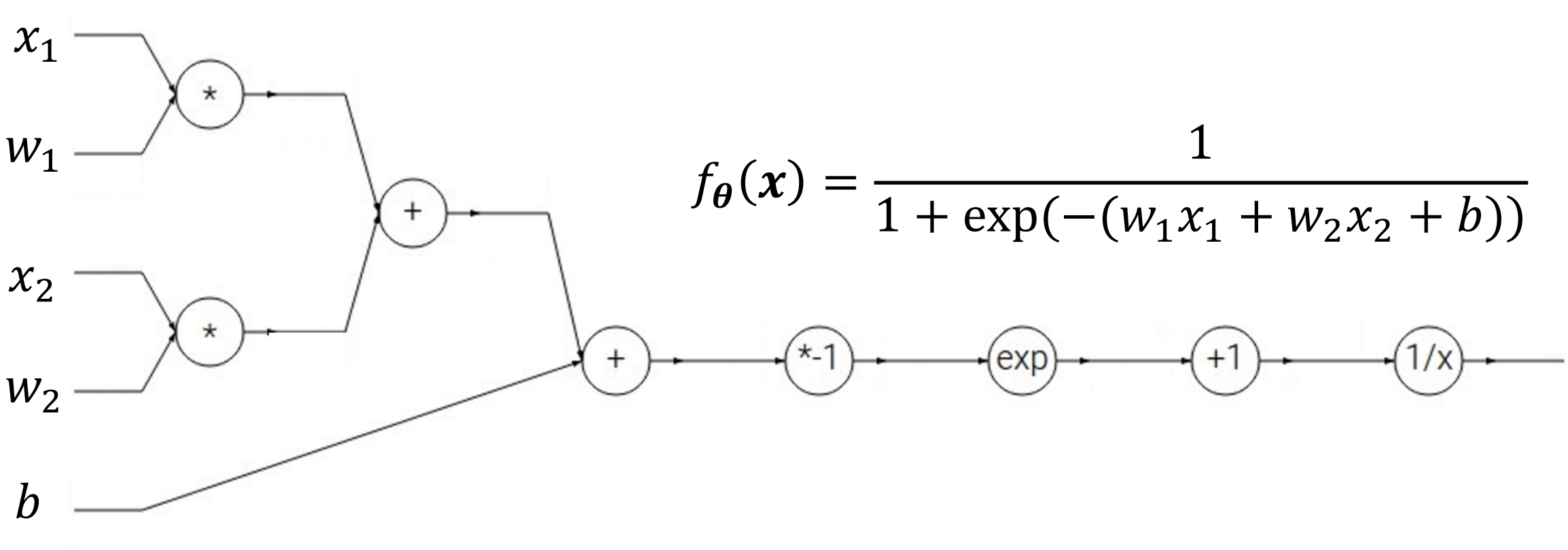
Chain Rule

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

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

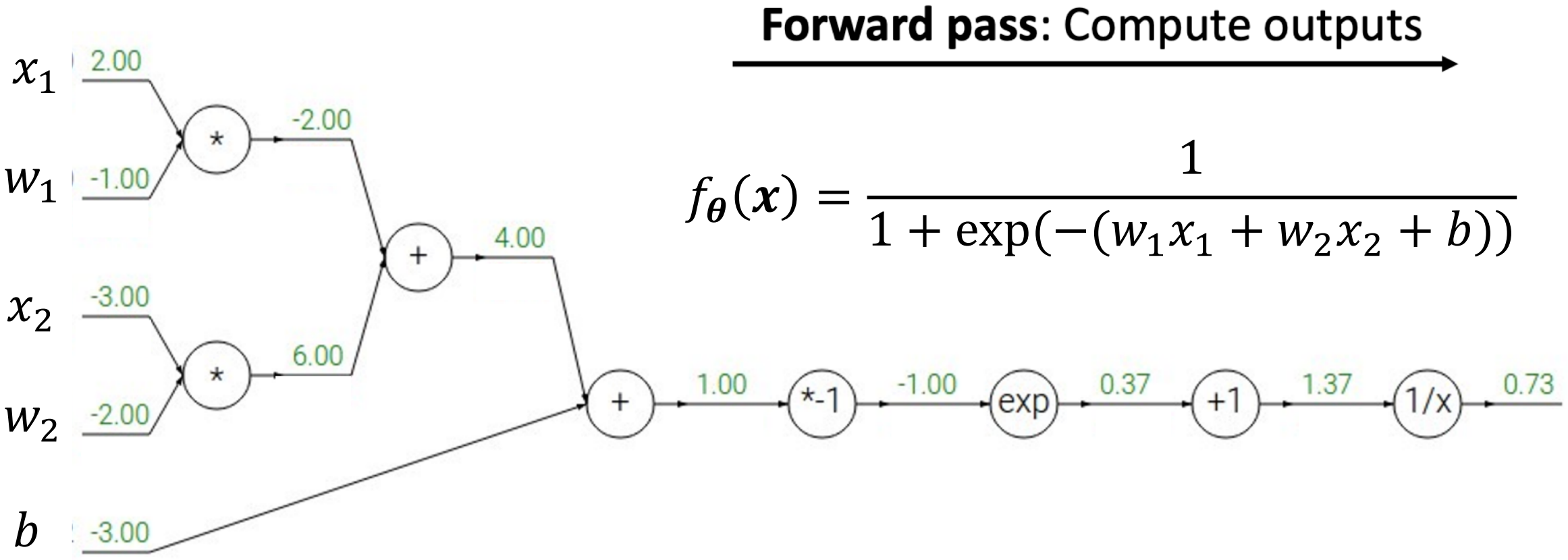
Downstream Gradient **Local Gradient** **Upstream Gradient**

Backpropagation



$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation

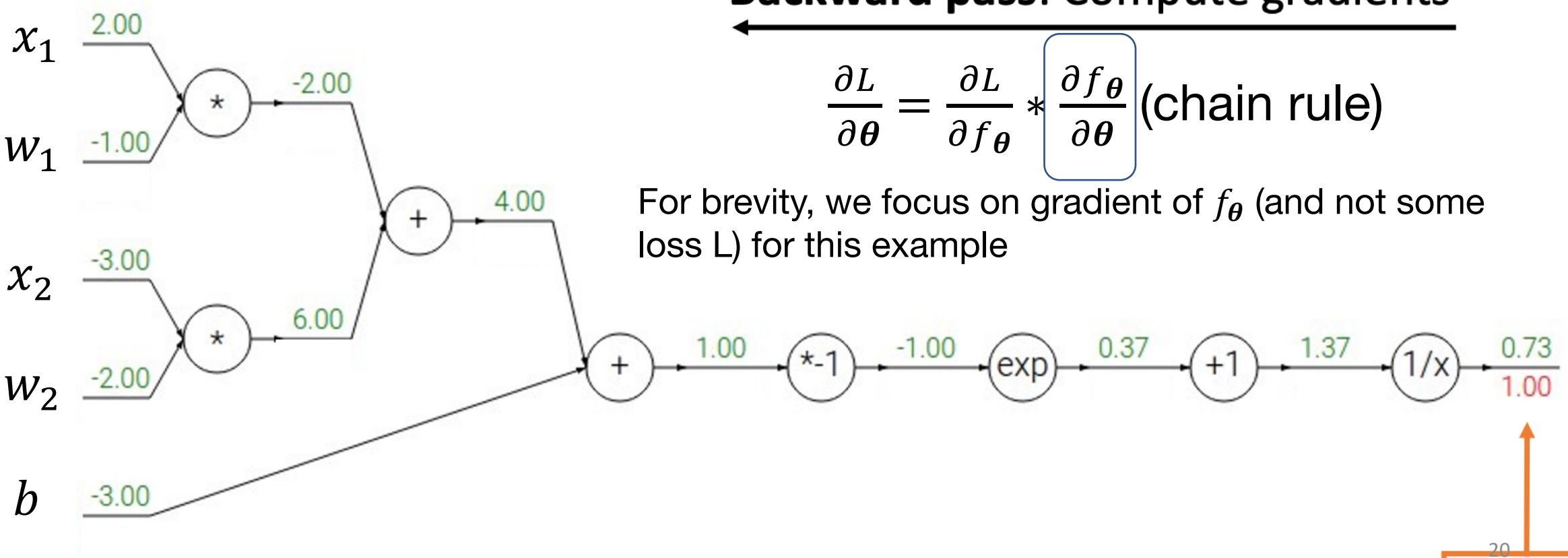


Backpropagation

Backward pass: Compute gradients

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial L}{\partial f_{\boldsymbol{\theta}}} * \frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \text{ (chain rule)}$$

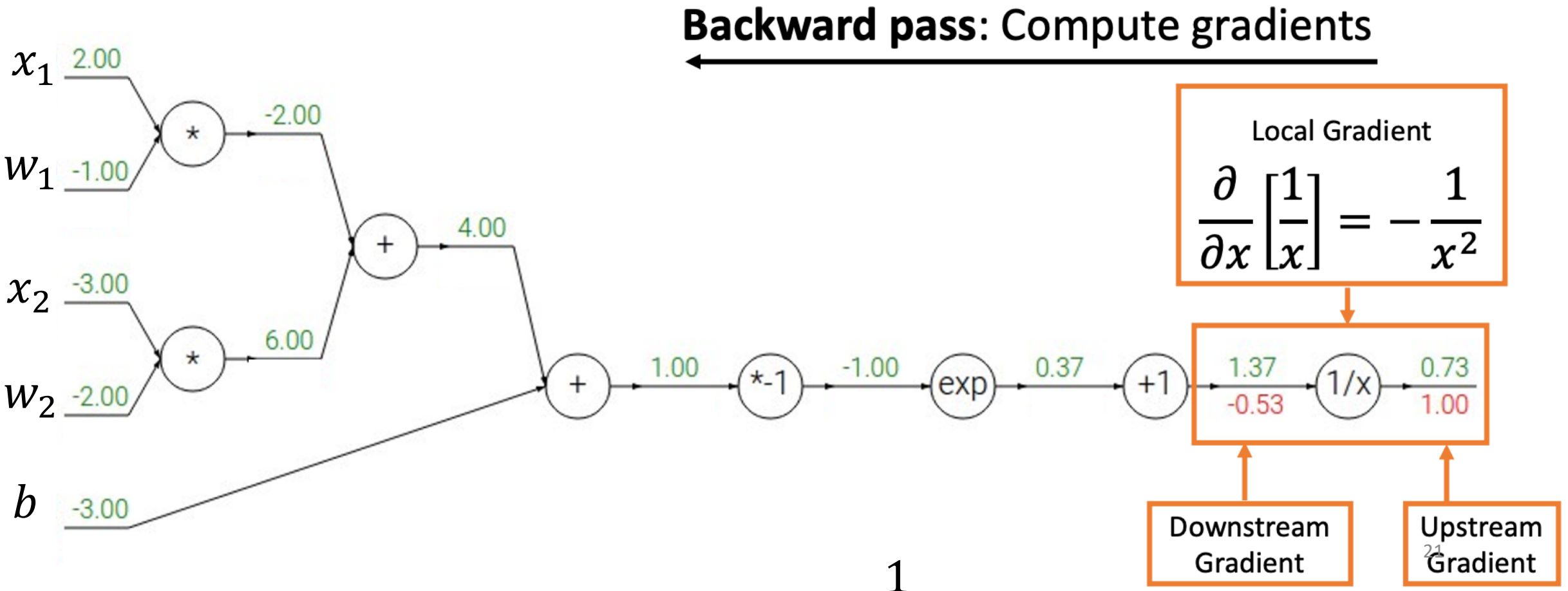
For brevity, we focus on gradient of $f_{\boldsymbol{\theta}}$ (and not some loss L) for this example



Base Case

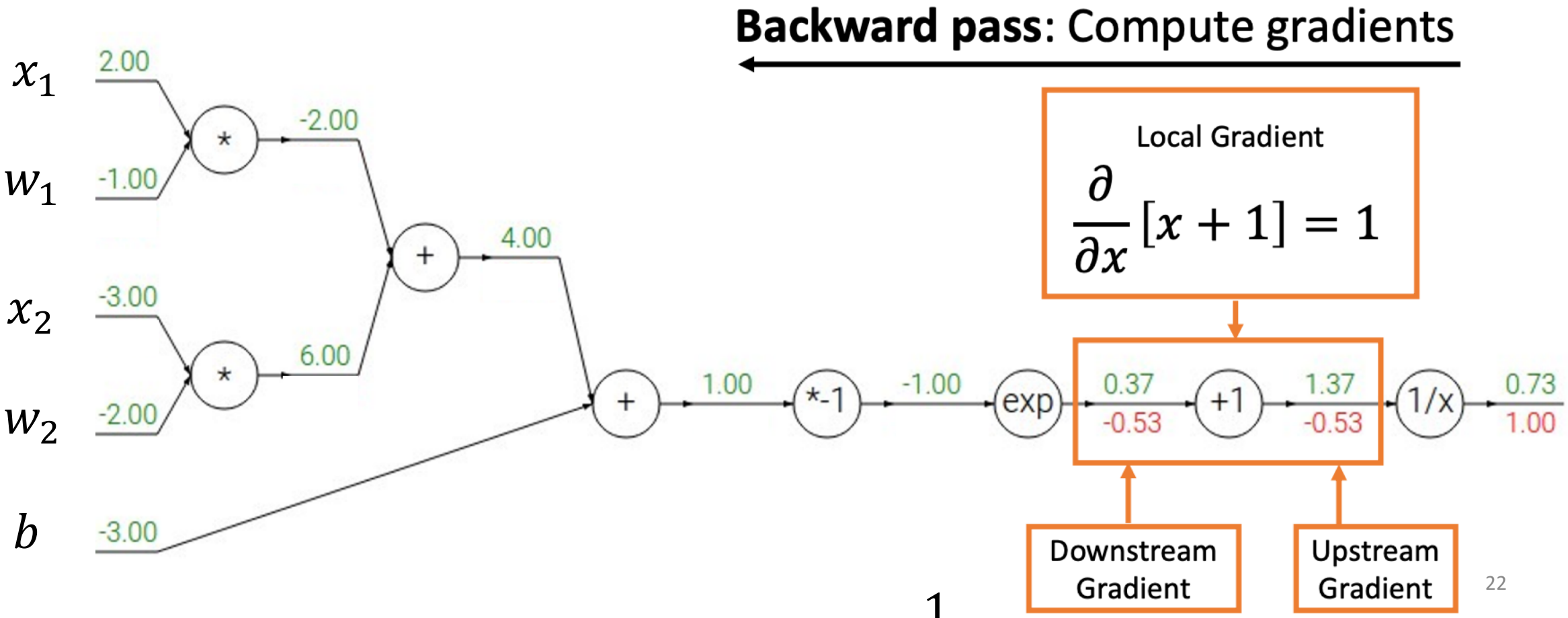
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



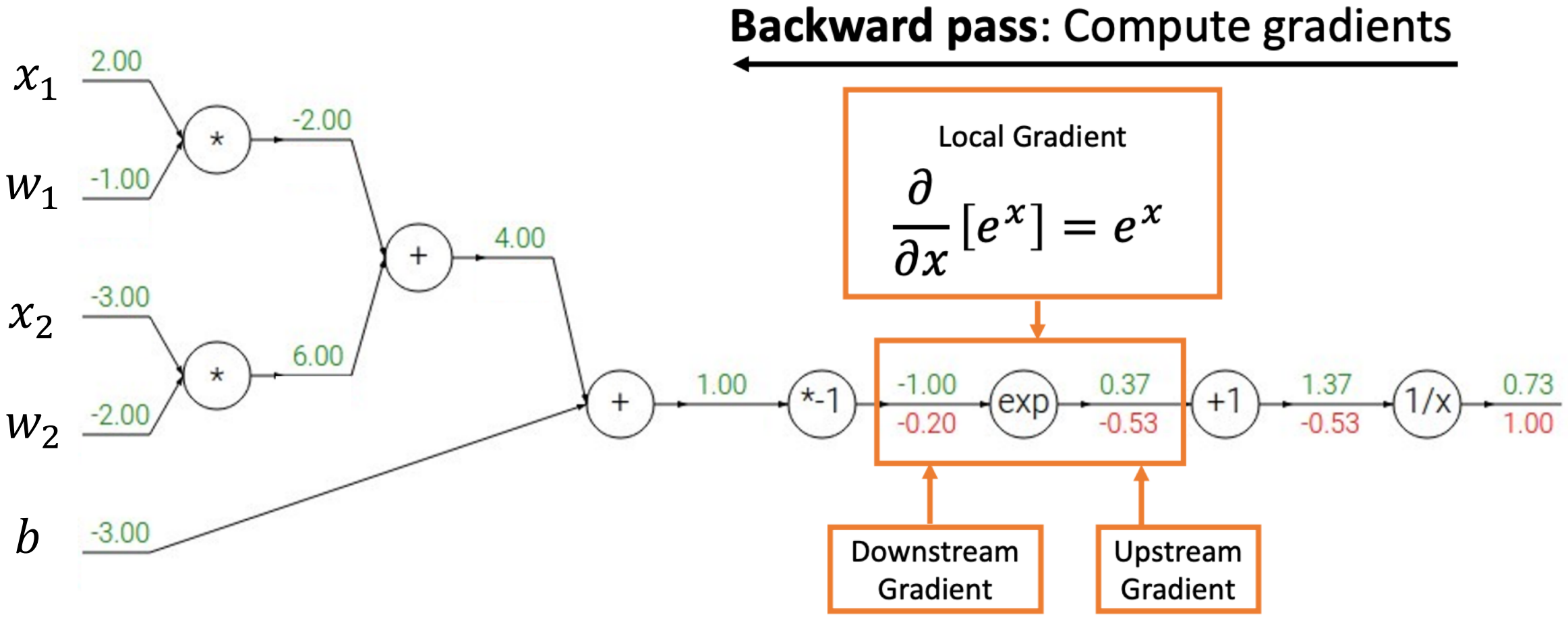
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



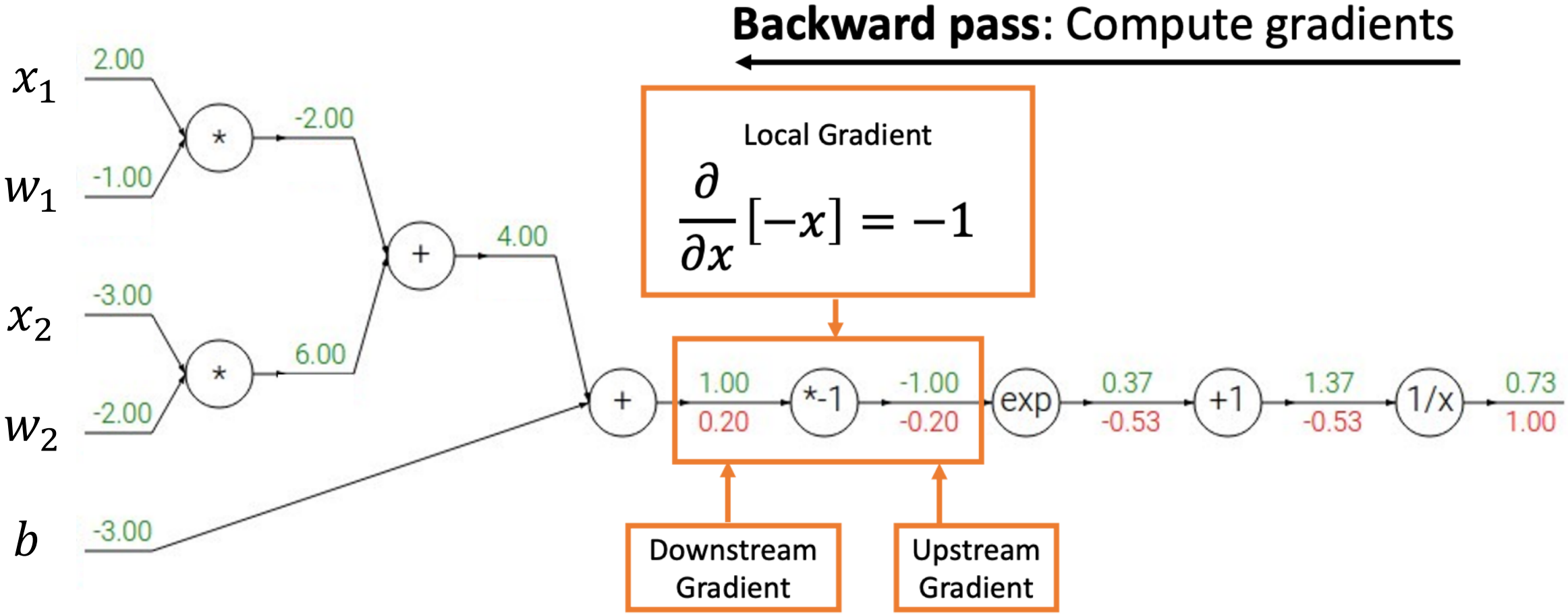
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



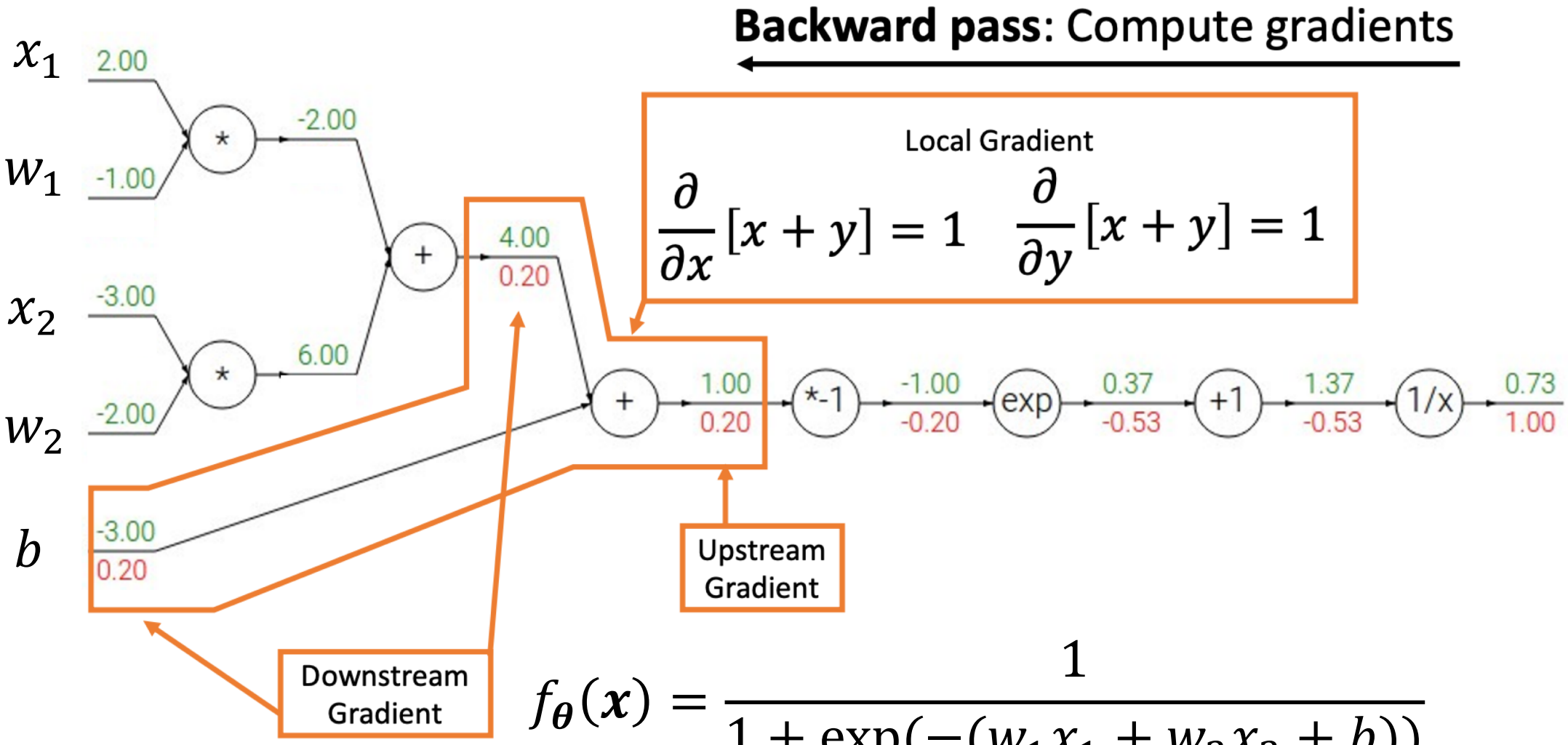
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



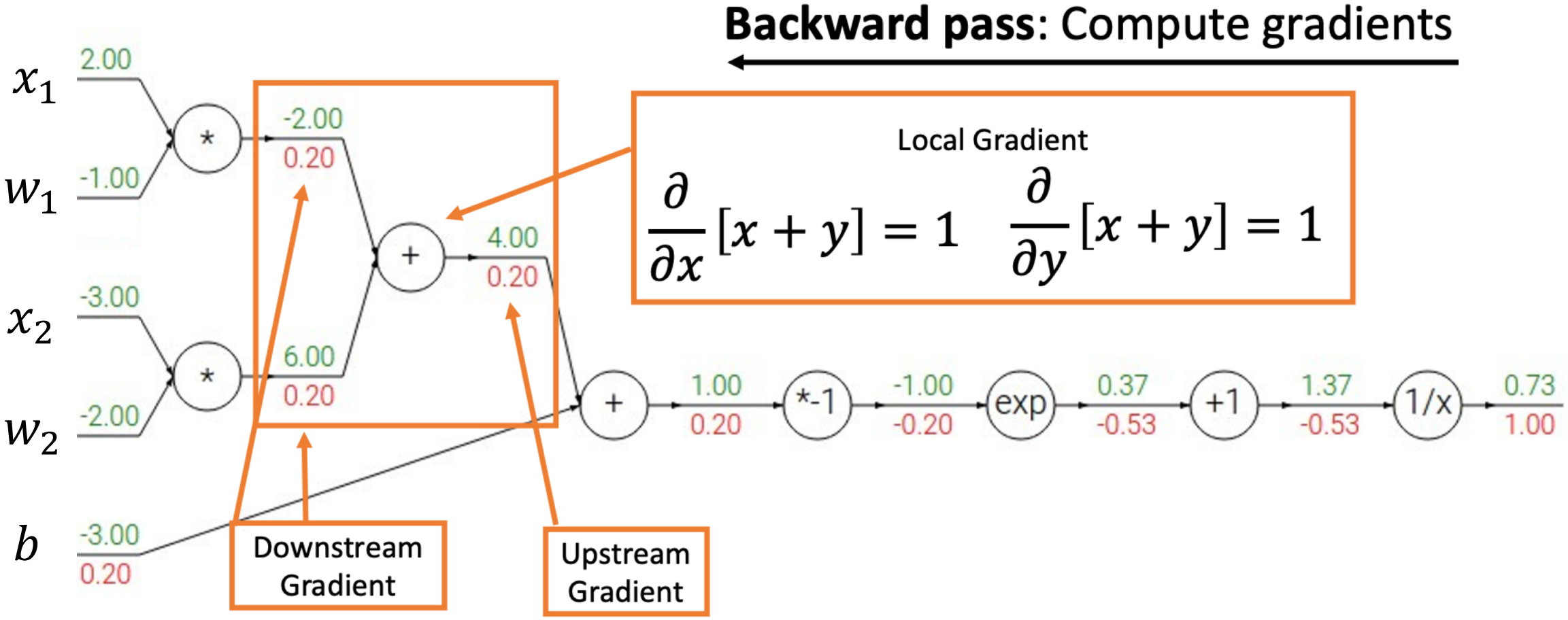
$$f_{\theta}(x) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



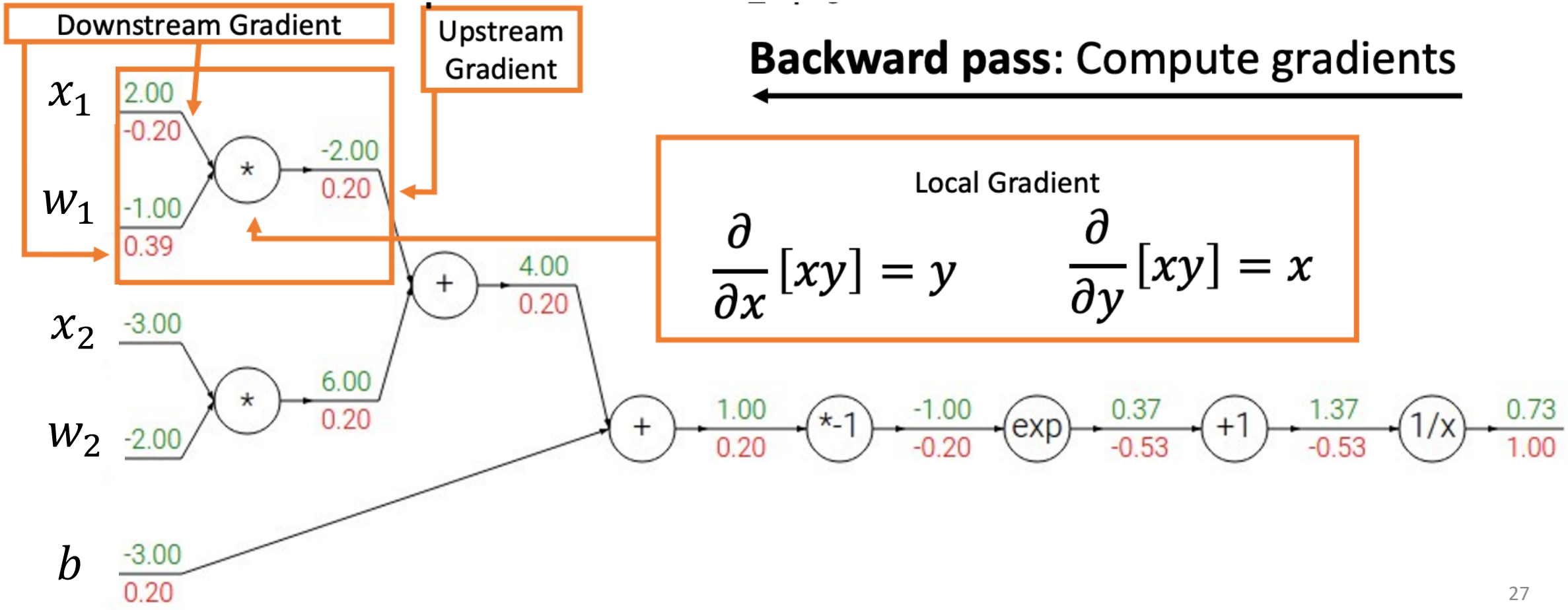
$$f_{\theta}(x) = \frac{1}{1+\exp(-(w_1x_1 + w_2x_2 + b))}$$

Backpropagation



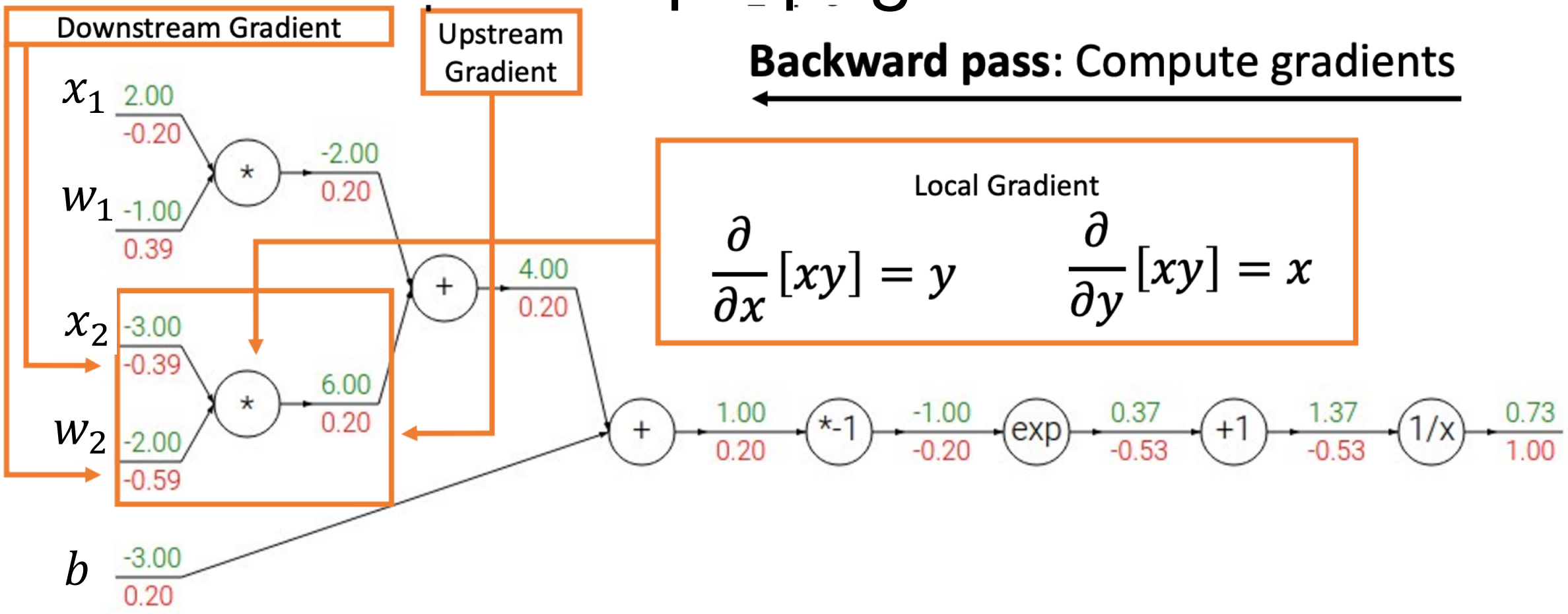
$$f_{\theta}(x) = \frac{1}{1+\exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation



$$f_{\theta}(x) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

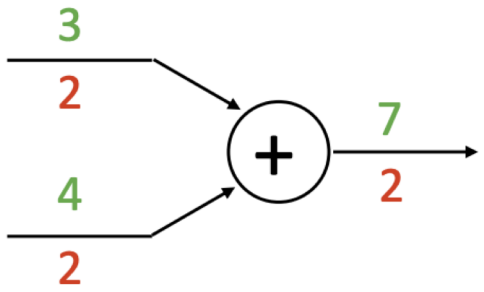
Backpropagation



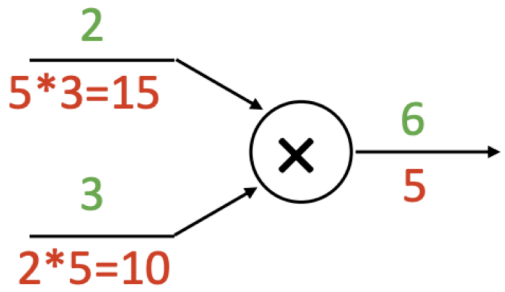
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Patterns in Gradient Flow

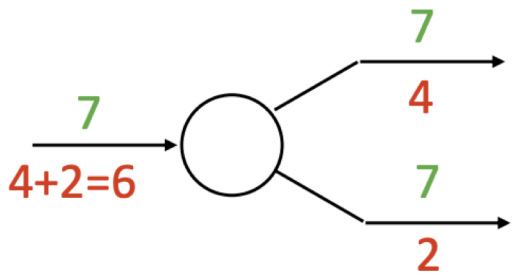
add gate: gradient distributor



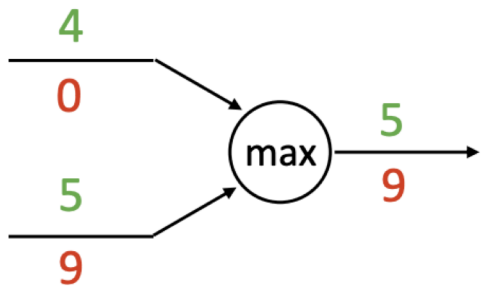
mul gate: "swap multiplier"



copy gate: gradient adder

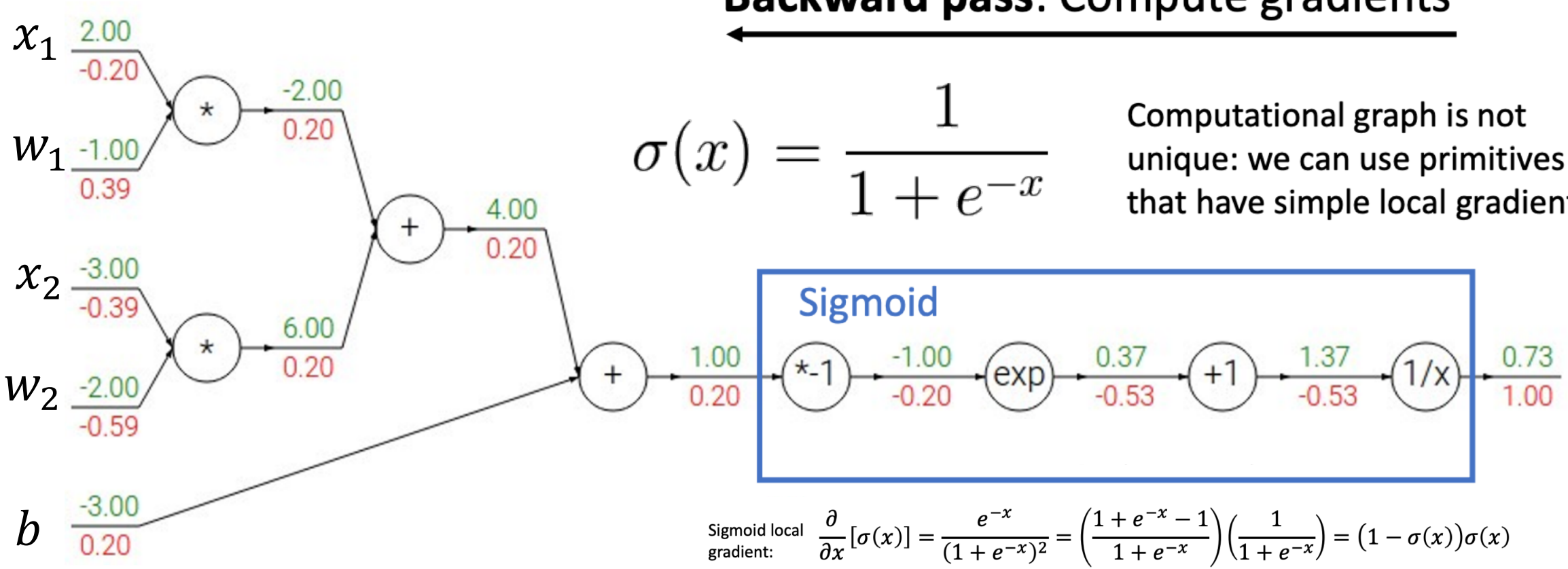


max gate: gradient router



Backpropagation

Backward pass: Compute gradients



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

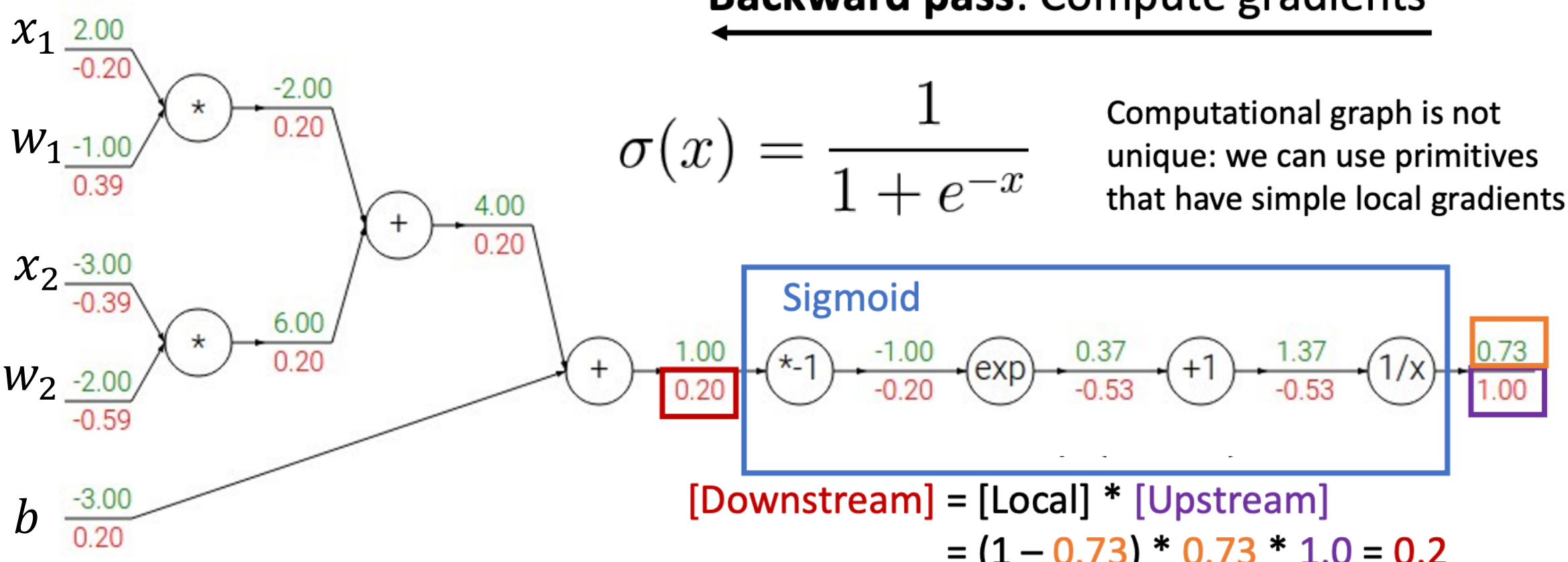
Computational graph is not unique: we can use primitives that have simple local gradients

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

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backpropagation

Backward pass: Compute gradients



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

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

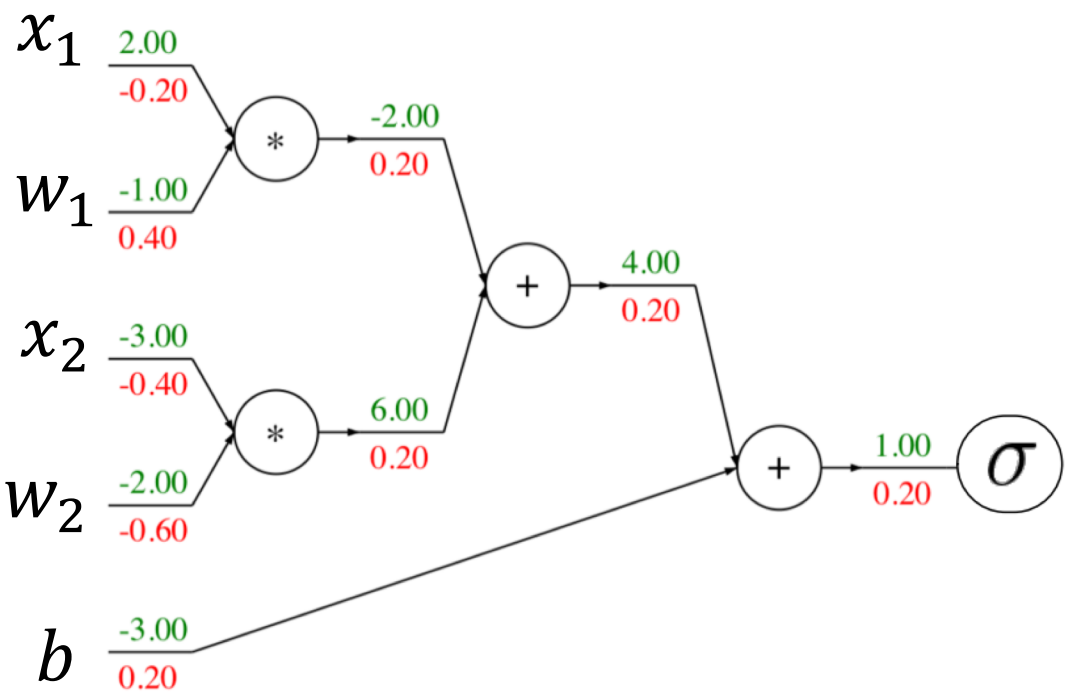
[Downstream] = [Local] * [Upstream]

= (1 – 0.73) * 0.73 * 1.0 = 0.2

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 + b))}$$

Backprop Implementation: "Flat" gradient code:

Forward pass:
Compute output



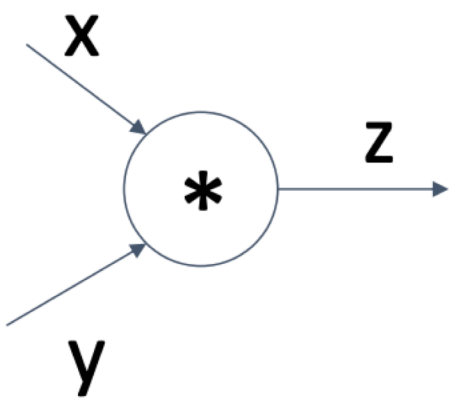

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  f = sigmoid(s3)
```

Backward pass:
Compute grads

```
grad_f = 1.0
grad_s3 = grad_f * (1-f) * f
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
```

Computed during forward pass.
Need to cache it!

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
```

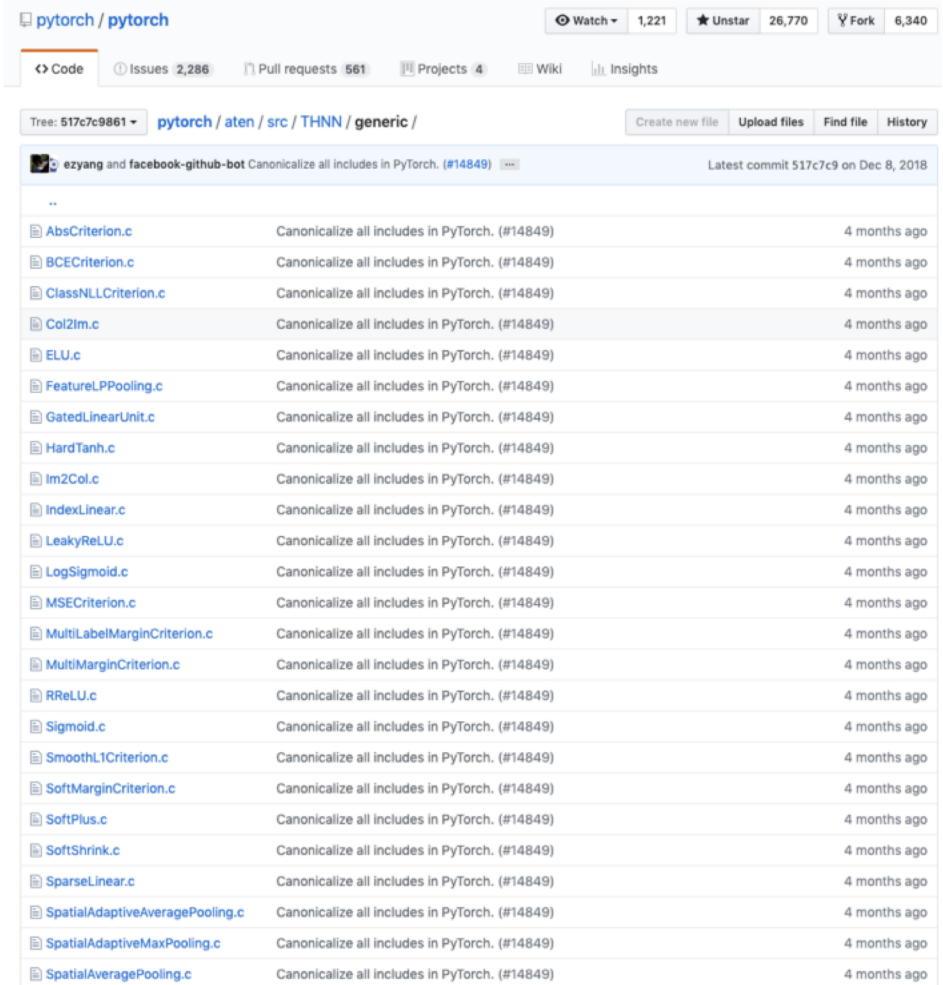
Caching!

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

Backprop for 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**:

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

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

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

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

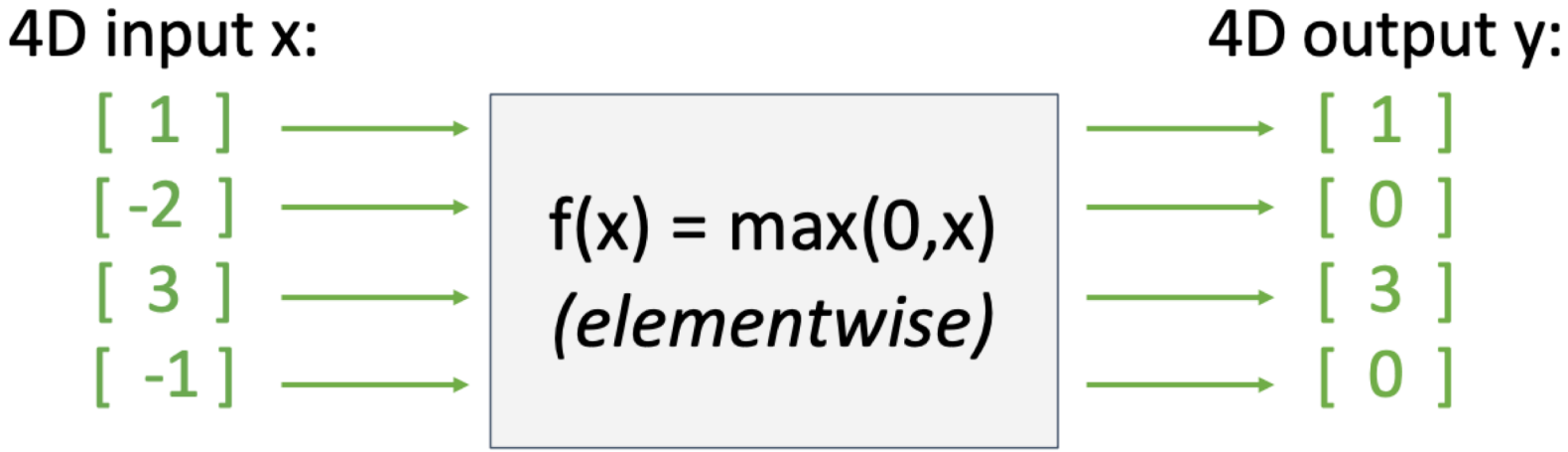
Derivative is **Jacobian**:

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

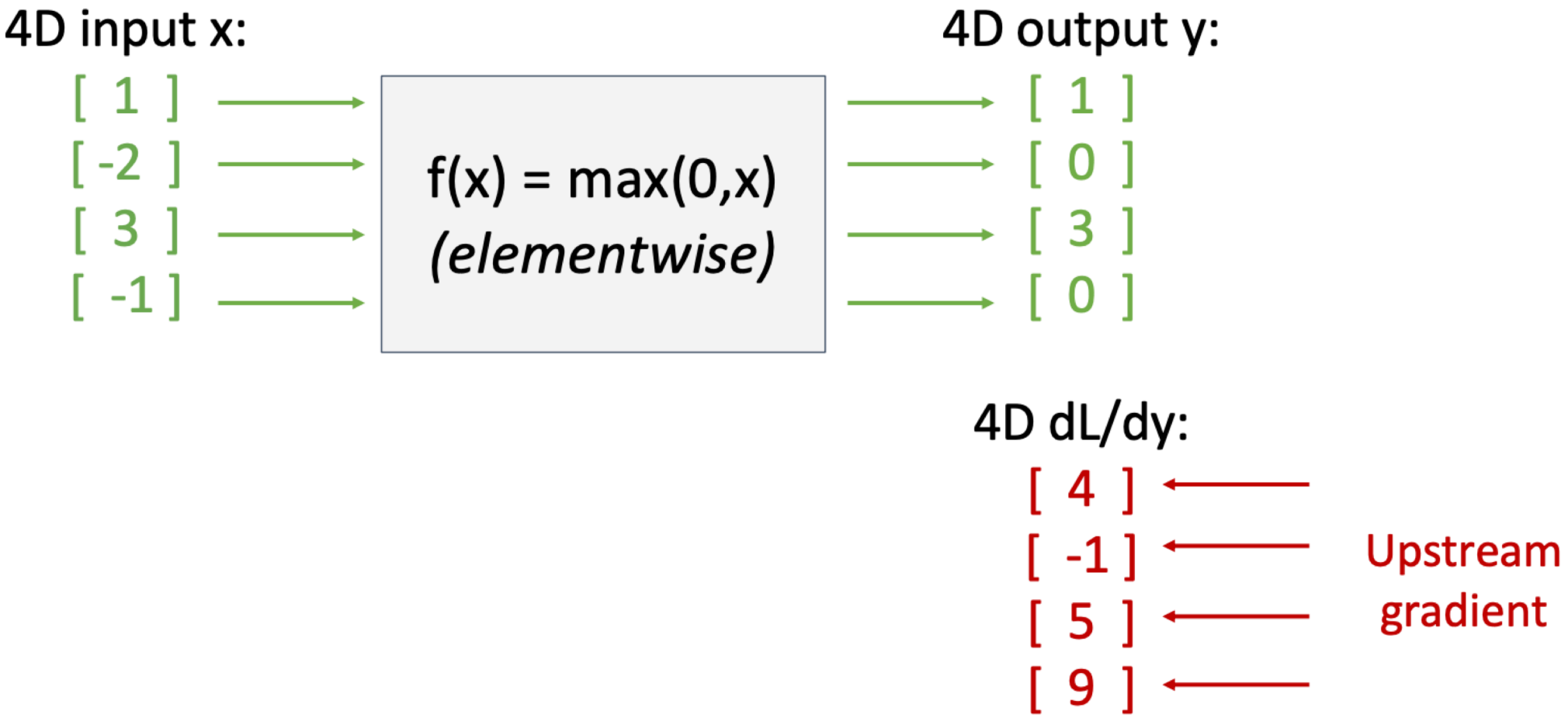
$$\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_j}{\partial x_i}$$

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

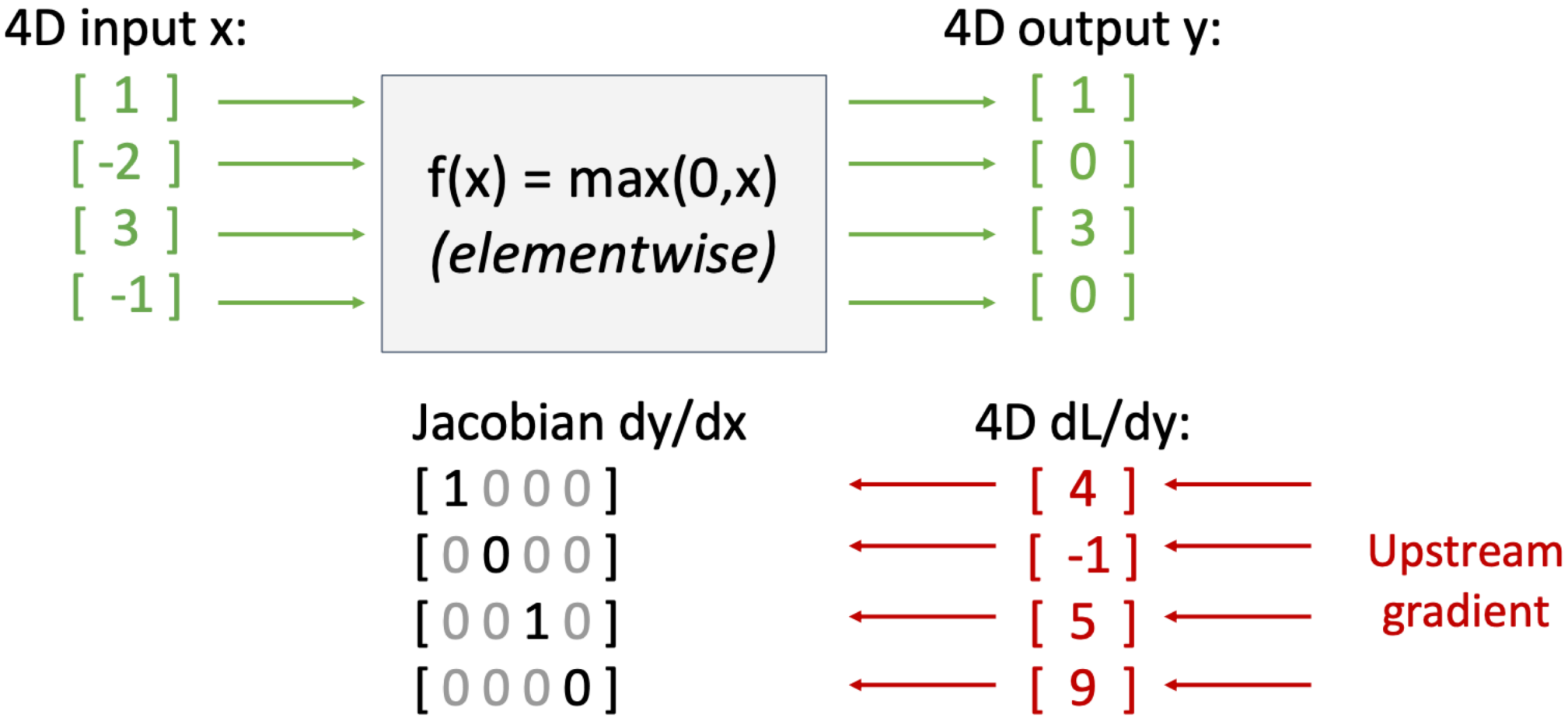
Backpropagation with Vectors



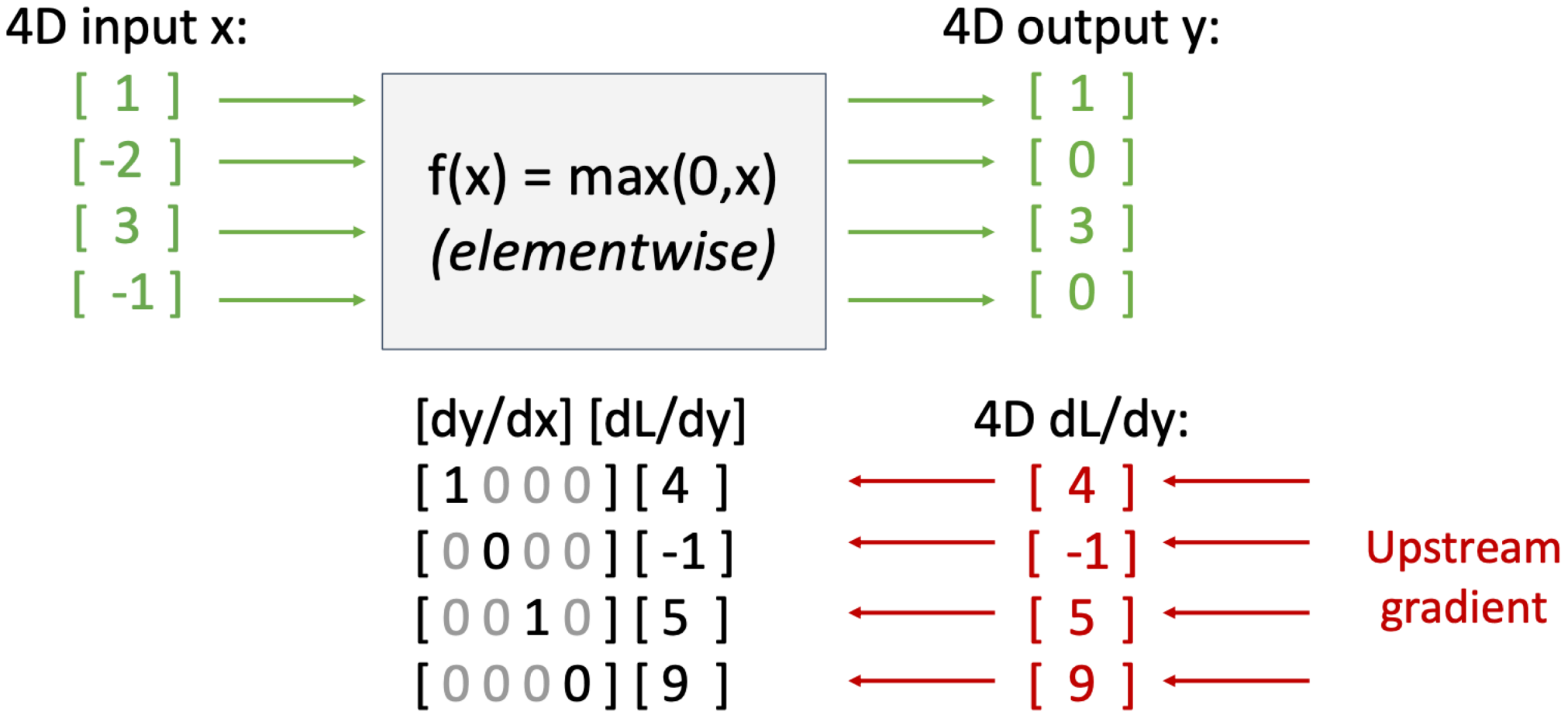
Backpropagation with Vectors



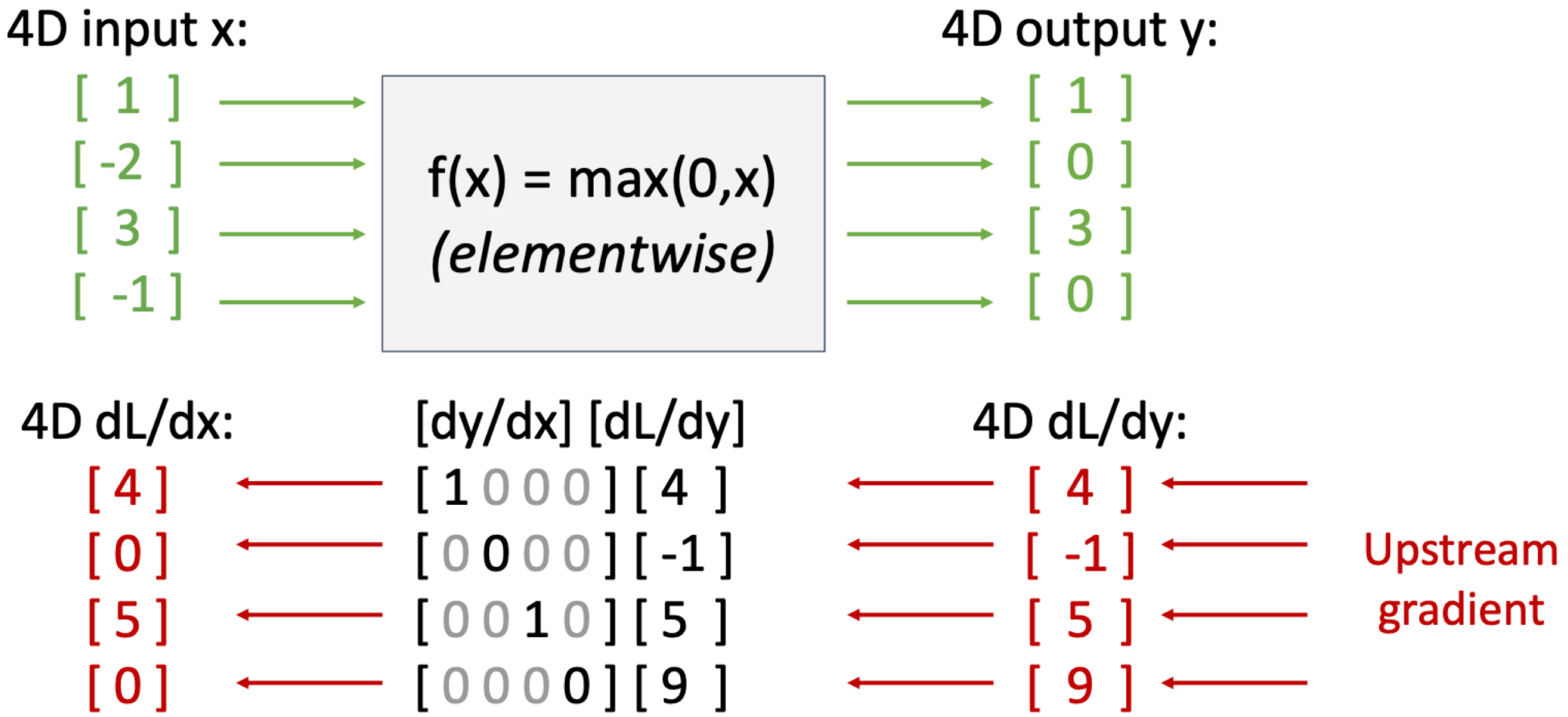
Backpropagation with Vectors



Backpropagation with Vectors

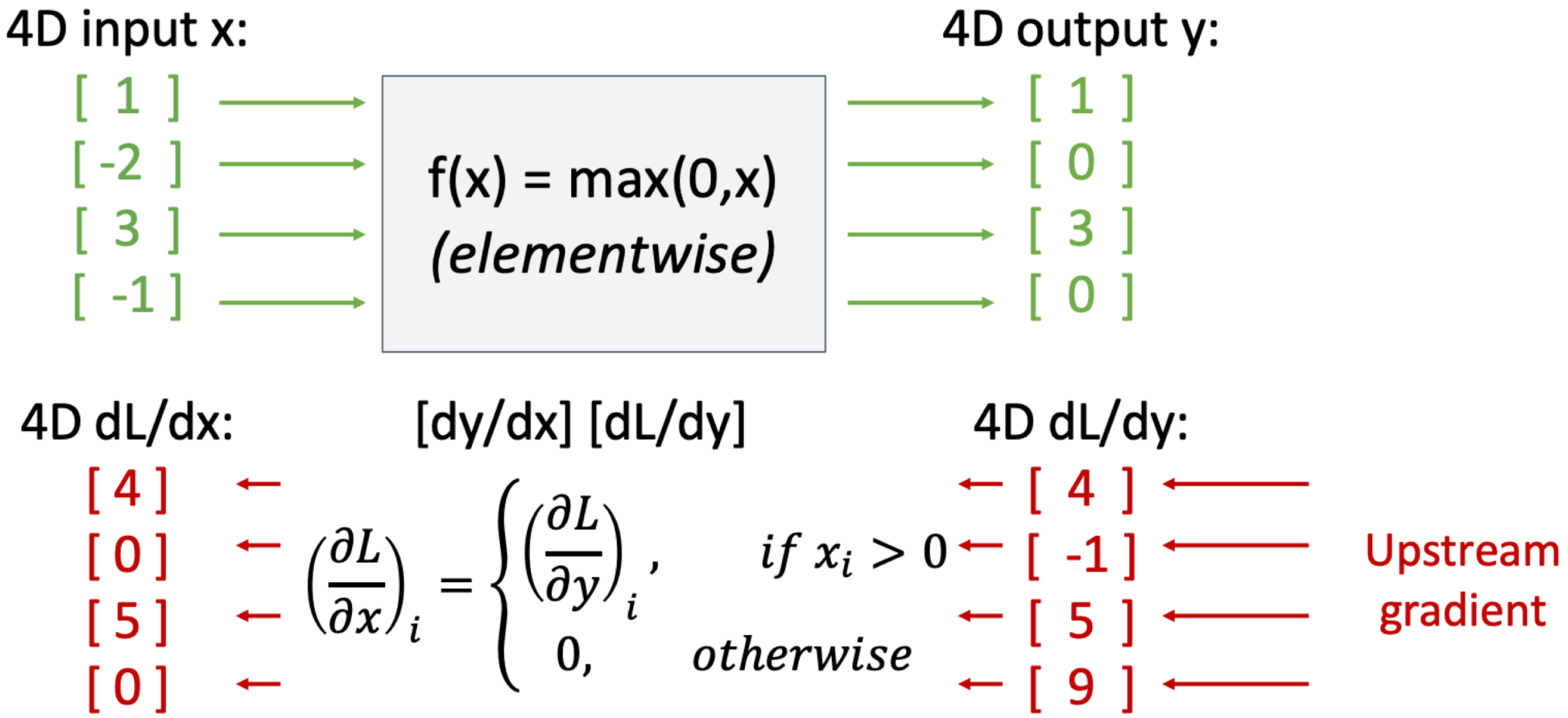


Backprop with Vectors



Jacobian is sparse: off diagonal entries are all zero.

Backpropagation with Vectors



Never explicitly form Jacobian; instead use implicit multiplication

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

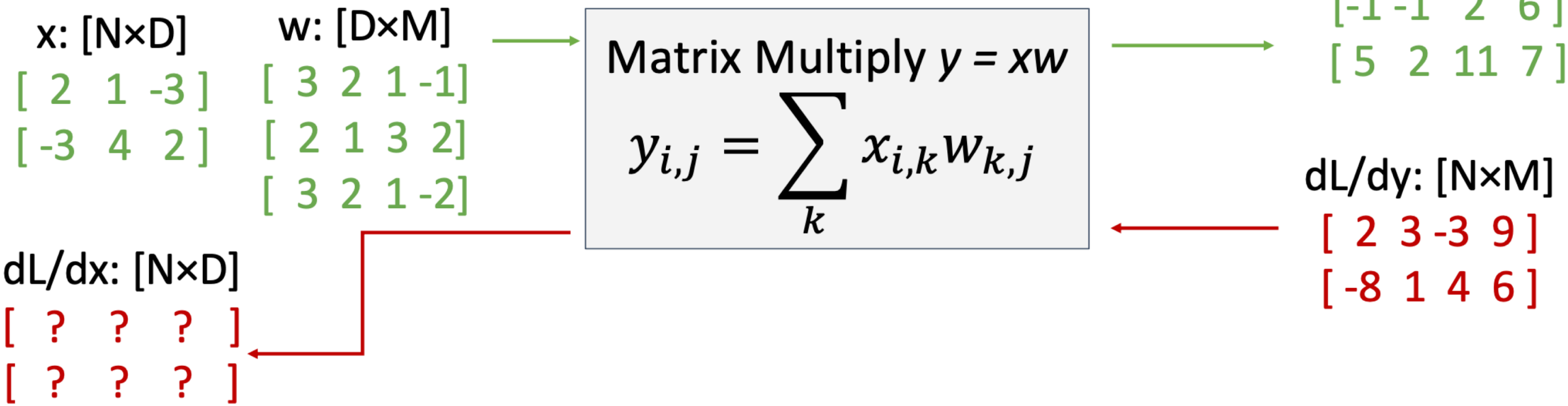
Matrix Multiply $y = xw$

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

y: [N×M]

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

Example: Matrix Multiplication



Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

y: [N×M]
[-1 -1 2 6]
[5 2 11 7]

dL/dy: [N×M]
[2 3 -3 9]
[-8 1 4 6]

dL/dx: [N×D]
[? ? ?]
[? ? ?]

Jacobians:
dy/dx: [(N×D)×(N×M)]
dy/dw: [(D×M)×(N×M)]

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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]

[? ? ?]
[? ? ?]

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

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[? ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

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 ? ?]
[? ? ?]

Local Gradient Slice:

$dy/dx_{1,1}$

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

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

Example: Matrix Multiplication

x: [N×D]

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

w: [D×M]

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

Matrix Multiply $y = xw$

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

y: [N×M]

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

dL/dy: [N×M]

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

dL/dx: [N×D]

[0 ? ?]
[? ? -30]

Local Gradient Slice:

$dy/dx_{2,3}$

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

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

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]

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

Example: Matrix Multiplication

x: [N×D]
[2 1 -3]
[-3 4 2]

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

Matrix Multiply $y = xw$

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

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]

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

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]

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

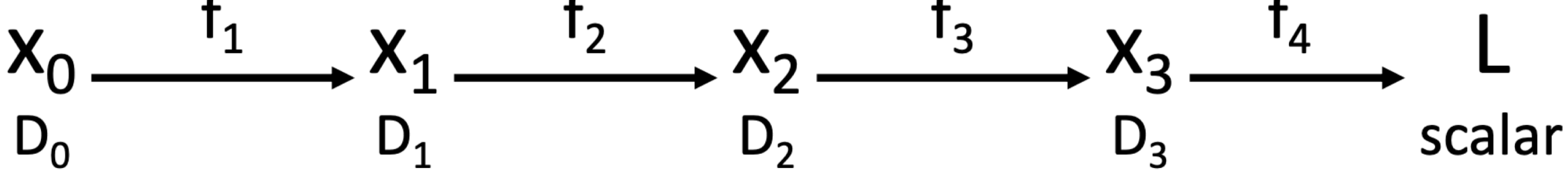
[N x D] [N x M] [M x D]

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

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

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

Backpropagation: Another View



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

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

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

Summary - Backpropagation

- Similar to any learning algorithm:
  - We define a (regularized) loss function in terms of unknowns (weights and biases)
  - We cycle through our examples, compute the loss, and update the weights to minimize it
- **Backpropagation:** The trick is to use chain rule and compute gradients in reverse order of forward computation
  - Can require caching of forward computation
  - Compute and memory-efficient when output is scalar (e.g., loss)