DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

LECTURE #04: INTRODUCTION TO NEURAL NETWORKS

CREATING THE NEXT®

ADMINISTRIVIA

- Assignment 0
  - Due today
- Assignment 1
  - Will be released today
  - Focuses on topics covered in first four lectures

LAST CLASS: LOSS FUNCTIONS

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

Score function

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

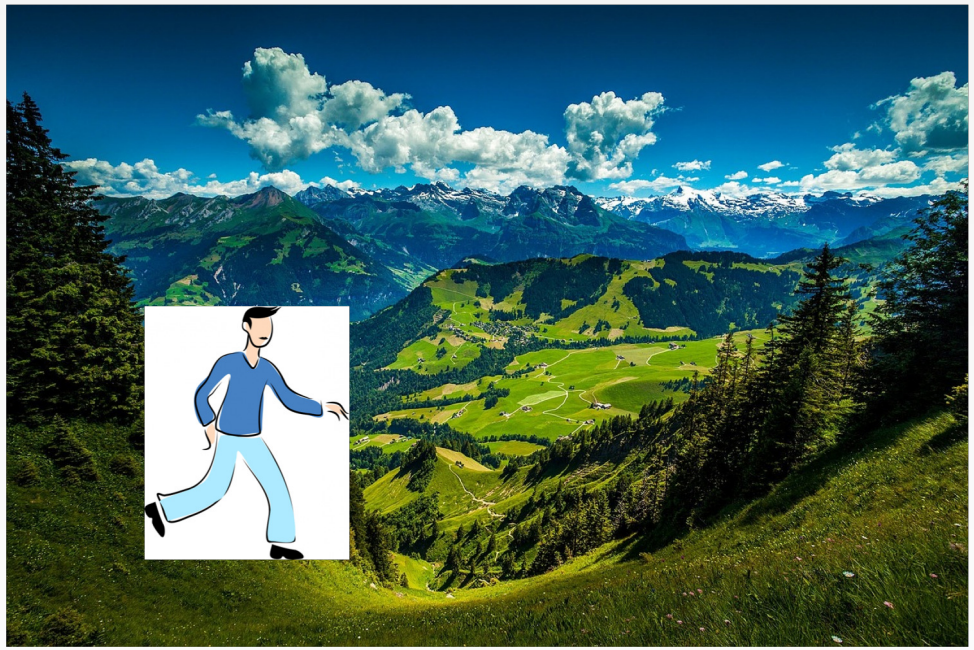
SVM data loss (or softmax)

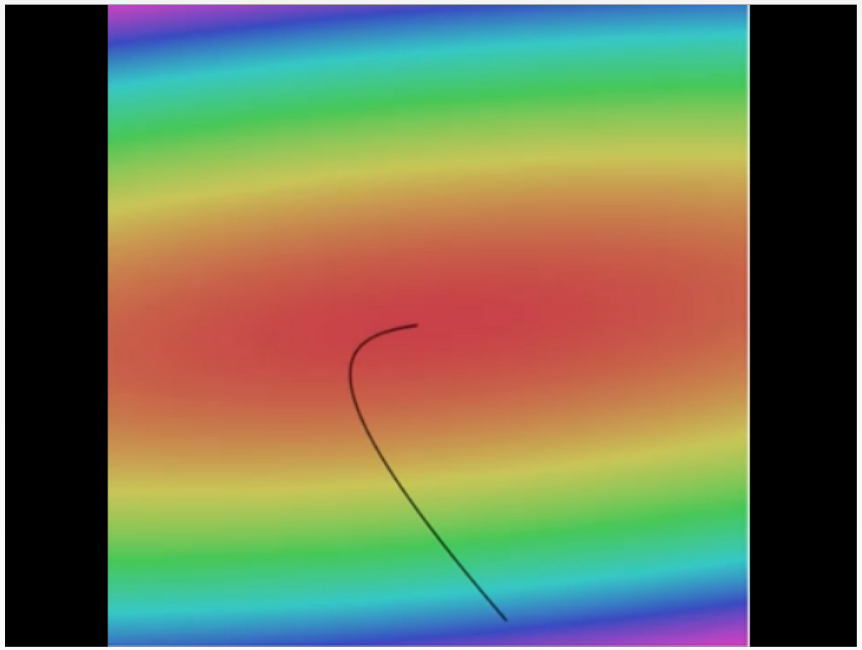
$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

Data loss +
Regularization loss

How to find the best W?

LAST CLASS: FINDING THE BEST W





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

LAST CLASS: GRADIENT DESCENT

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

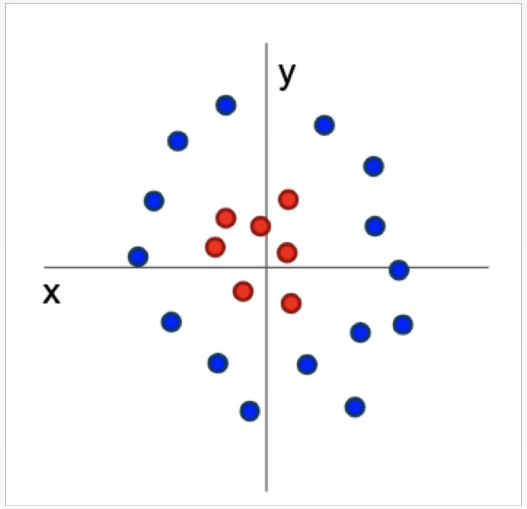
In practice: Derive analytic gradient, check your implementation with numerical gradient

LAST CLASS: LINEAR CLASSIFIERS NOT POWERFUL

Visual Viewpoint

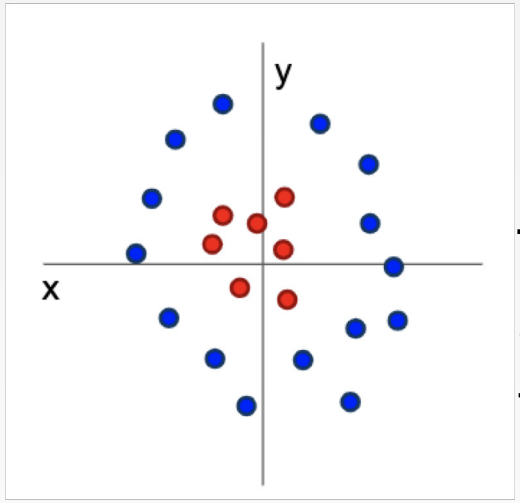
Linear classifiers learn one template per class

Geometric Viewpoint



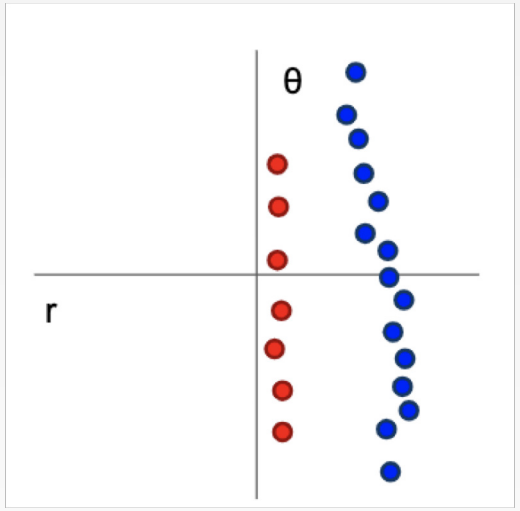
Linear classifiers can only draw linear decision boundaries

LAST CLASS: FEATURE TRANSFORMATION

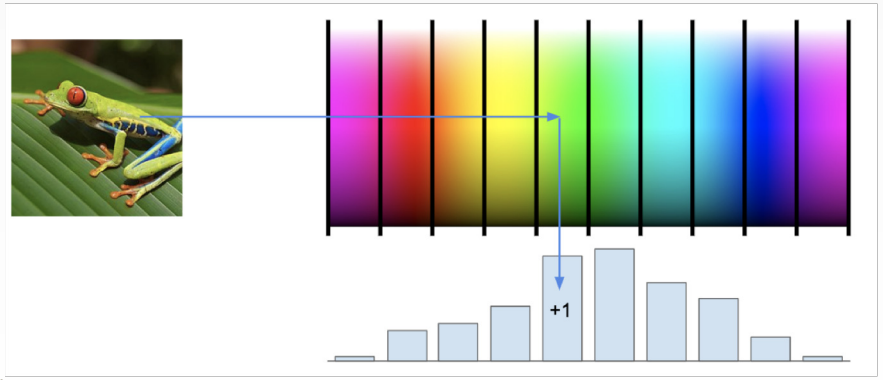


$$f(x, y) = (r(x, y), \theta(x, y))$$

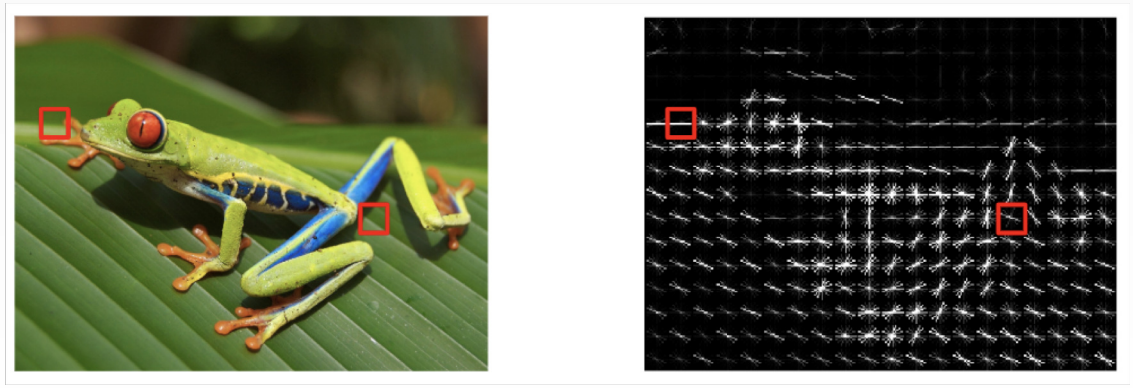
Transform data with a cleverly chosen **feature transform** f, then apply linear classifier



Color Histogram

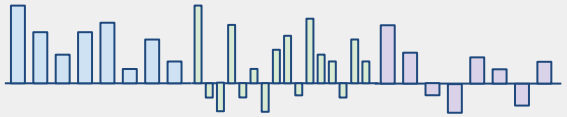


Histogram of Oriented Gradients (HoG)



LAST CLASS: IMAGE FEATURES VS CONVNETS

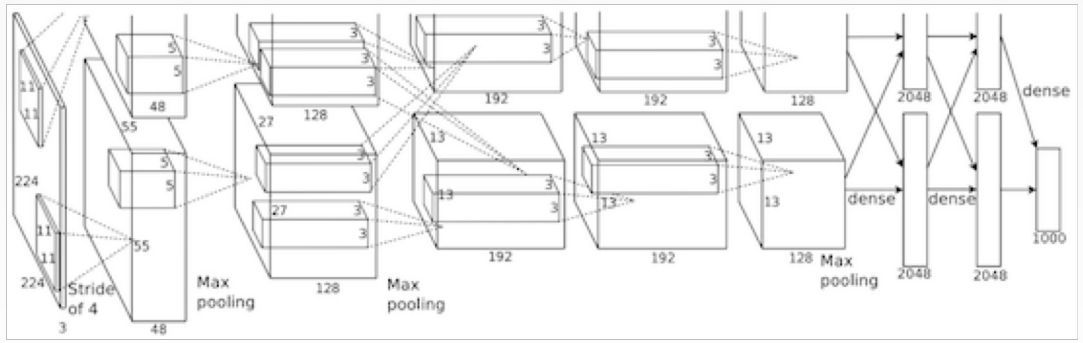
Feature Extraction



f

training

10 numbers giving scores for classes



training

10 numbers giving scores for classes

TODAY'S AGENDA

- Neural Networks
  - Without the brain stuff (as a function)
  - Biological inspiration
- Backpropagation
  - Computational Graphs
  - With vectors and matrices

NEURAL NETWORKS

NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function:

$$f = Wx$$

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function:

$$f = Wx$$

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

(**Now**) 2-layer Neural Network

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

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

(In practice we will usually add a learnable bias at each layer as well)

NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function:

$$f = Wx$$

(**Now**) 2-layer Neural Network

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

or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

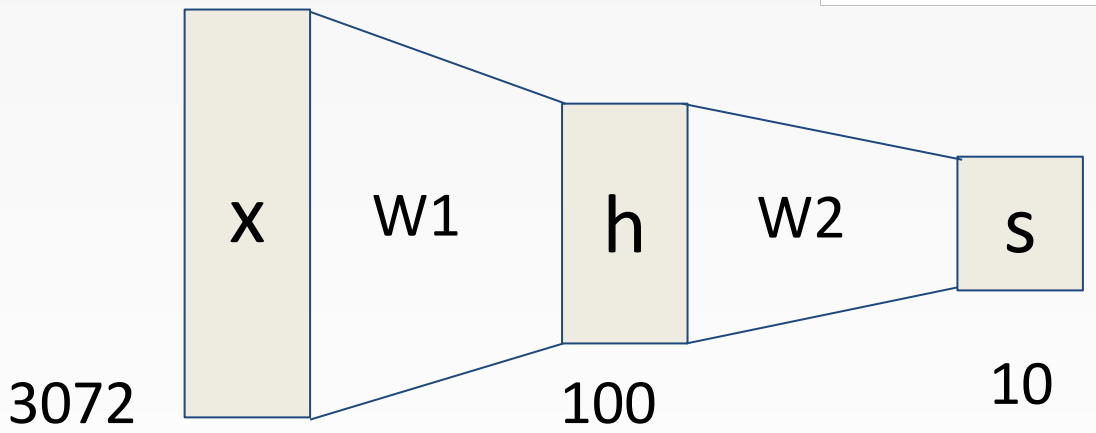
NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function:

$$f = Wx$$

(**Now**) 2-layer Neural Network

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

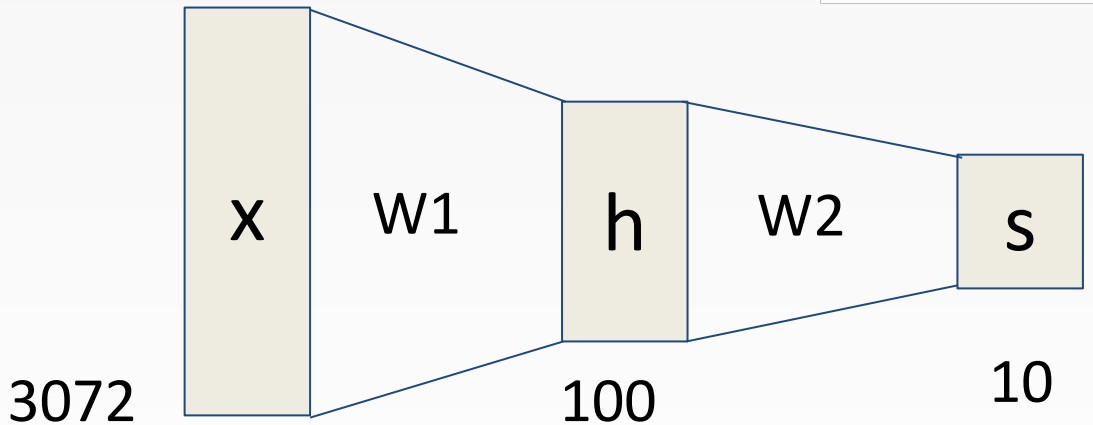


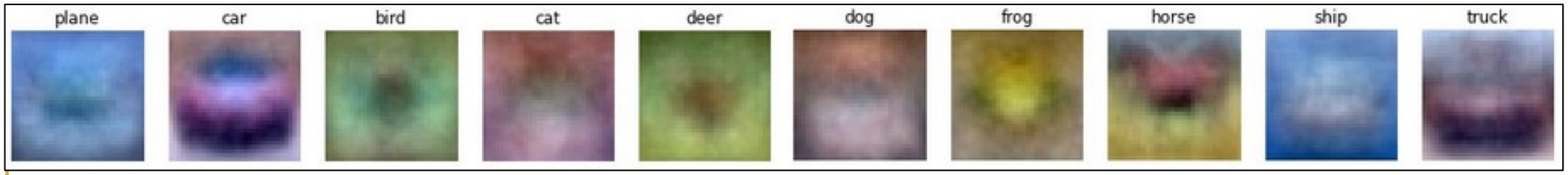
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$





NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

The function $\max(0, z)$ called the **activation function.**

Q: What if we try to build a neural network without one?

$f = W_2 W_1 x$ $W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$

NEURAL NETWORKS: WITHOUT THE BRAIN STUFF

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

The function $\max(0, z)$ called the **activation function.**

Q: What if we try to build a neural network without one?

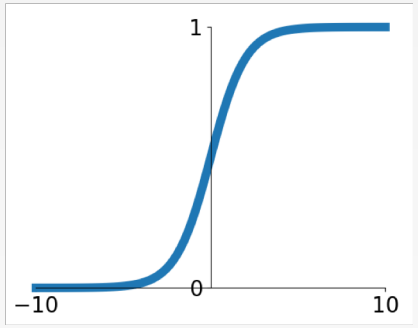
$f = W_2 W_1 x$ $W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$

A: We end up with a linear classifier again!

ACTIVATION FUNCTIONS

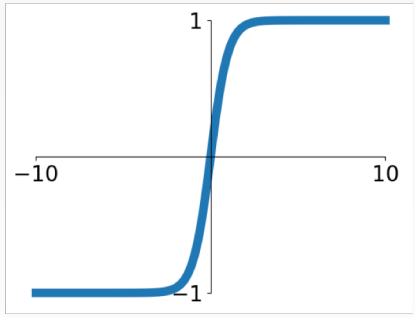
Sigmoid

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



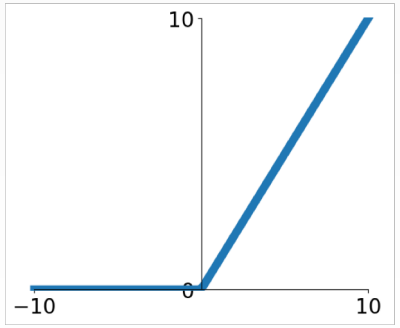
tanh

$$\tanh(x)$$



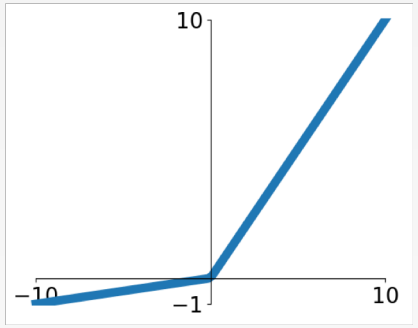
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

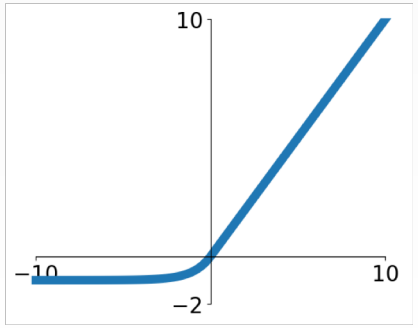


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

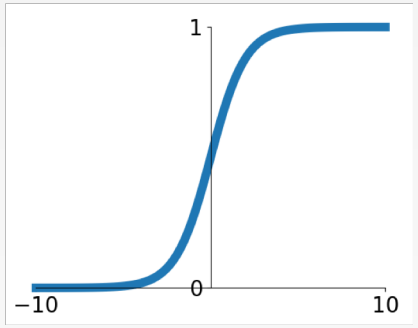
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ACTIVATION FUNCTIONS

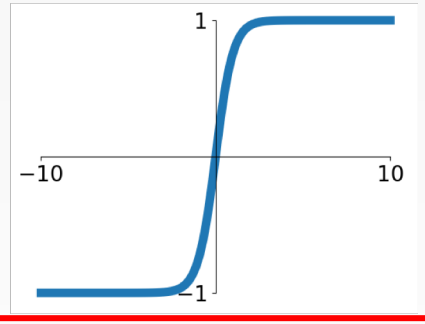
Sigmoid

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



tanh

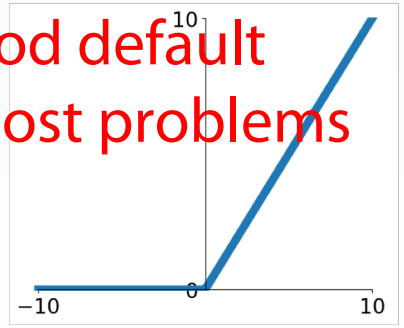
$$\tanh(x)$$



ReLU

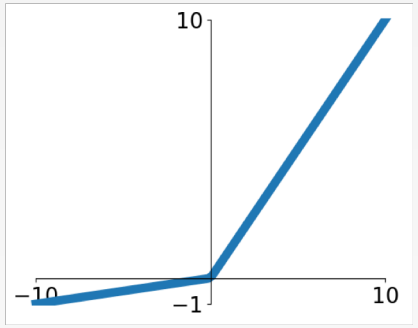
$$\max(0, x)$$

ReLU is a good default choice for most problems



Leaky ReLU

$$\max(0.1x, x)$$

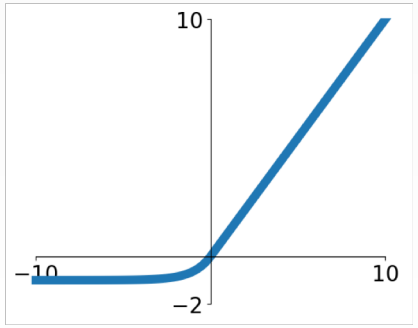


Maxout

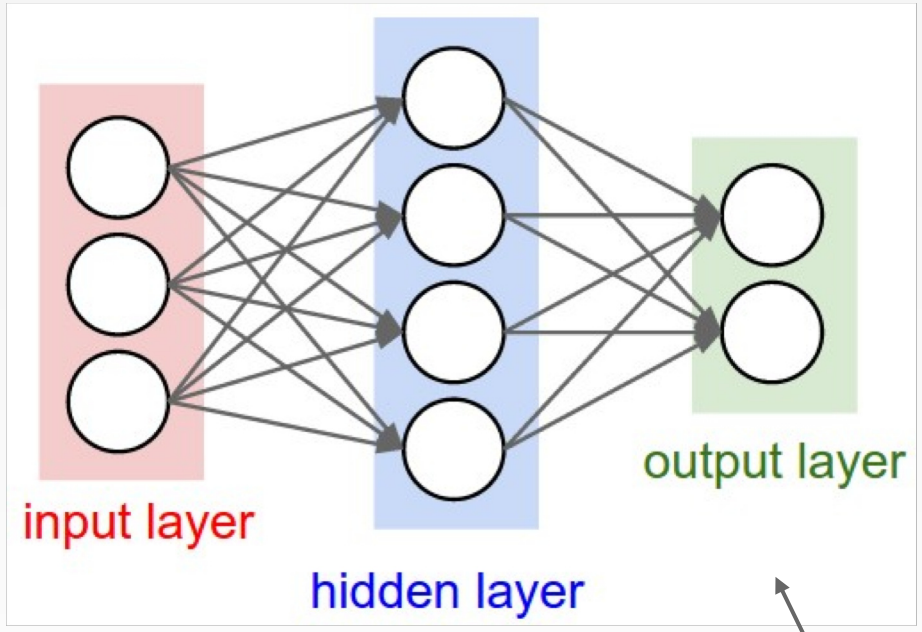
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

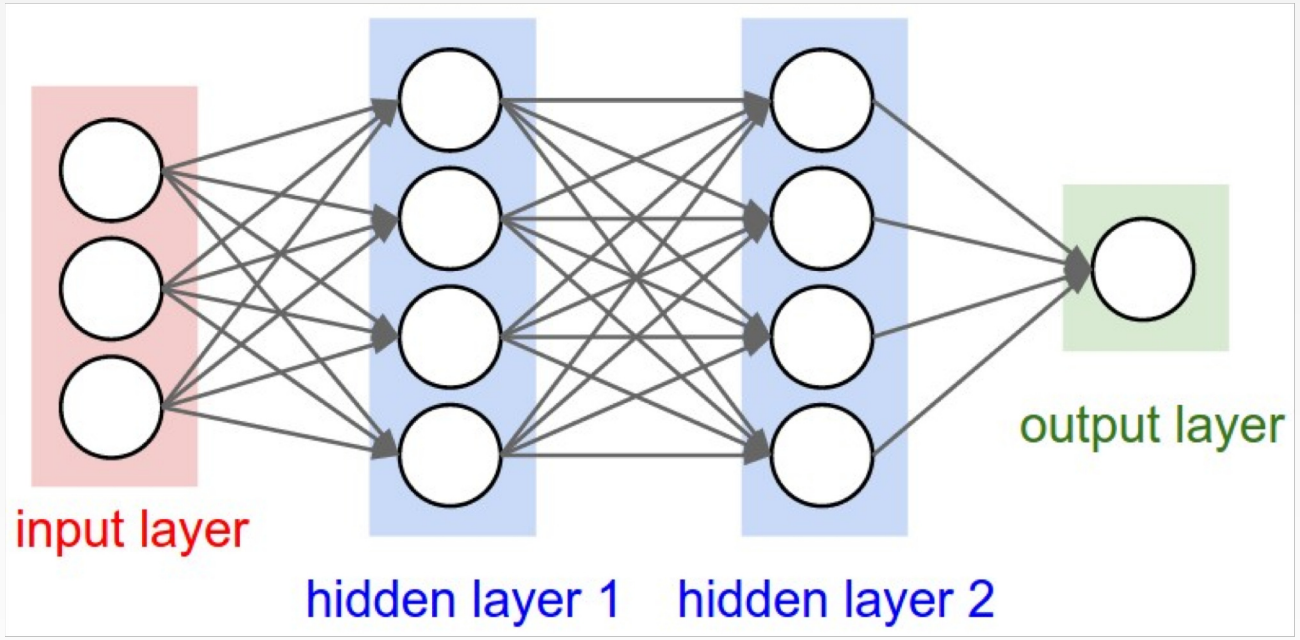
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



NEURAL NETWORKS: ARCHITECTURES



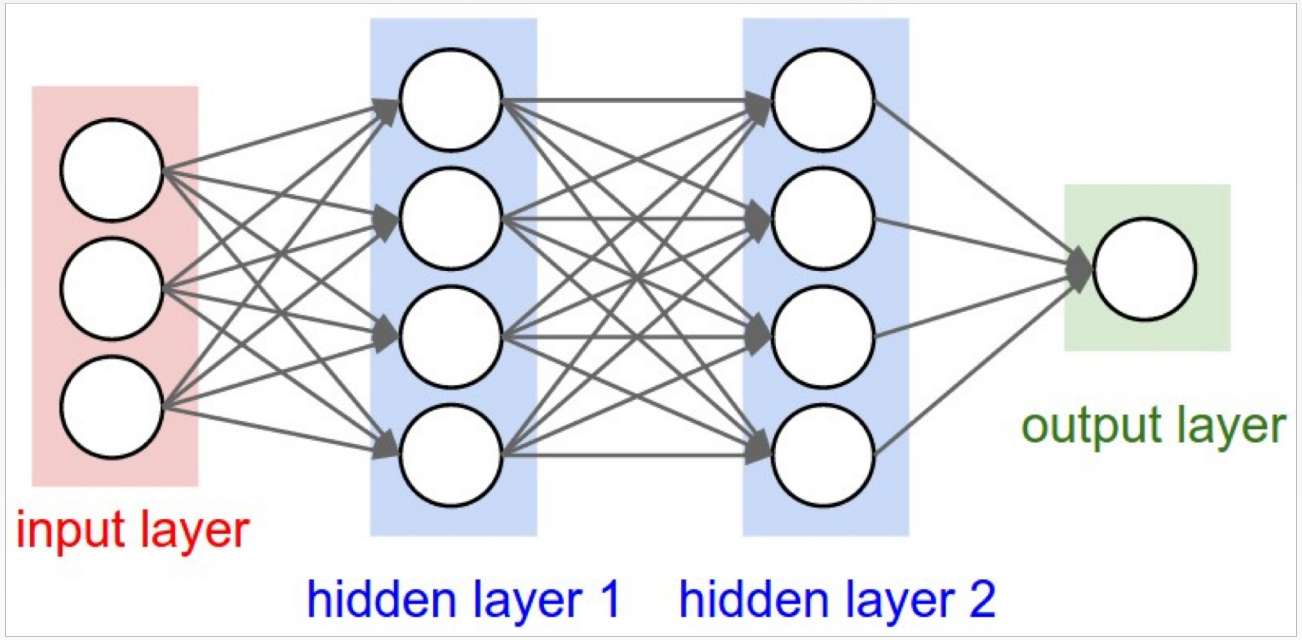


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

FEED-FORWARD COMPUTATION OF A NEURAL NETWORK

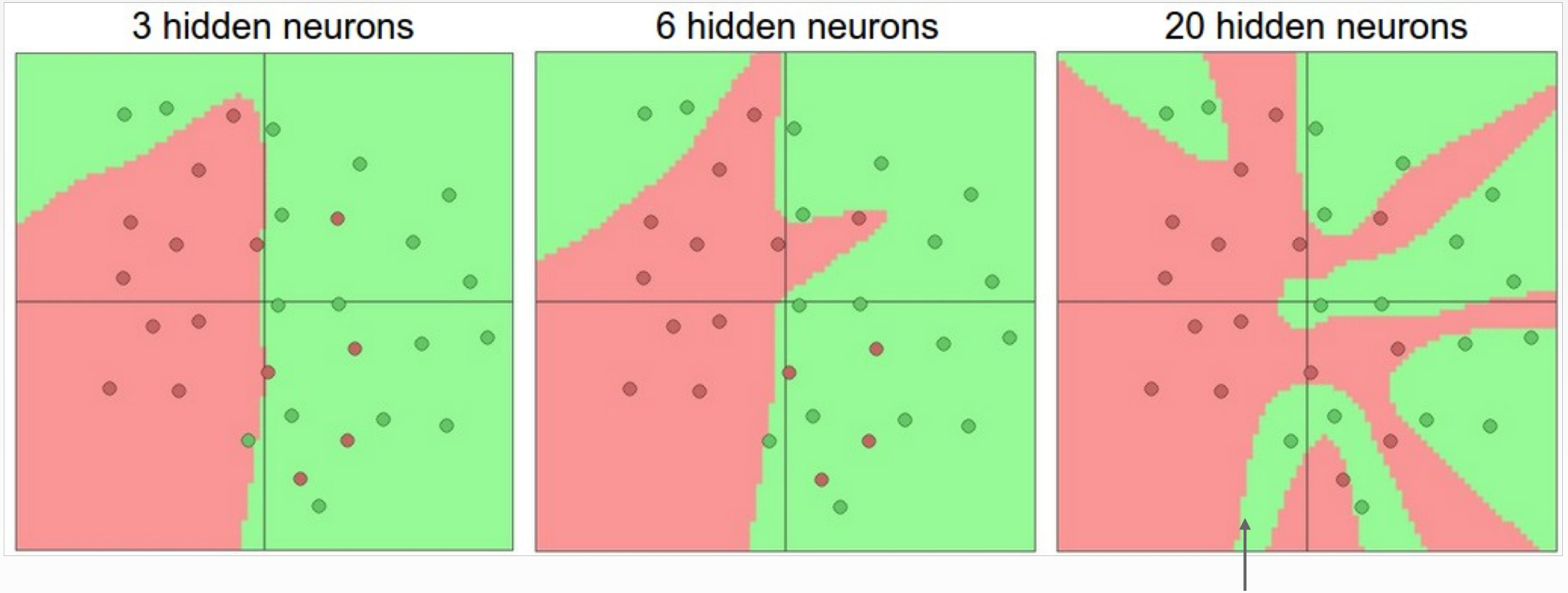



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

FEED-FORWARD COMPUTATION OF A NEURAL NETWORK

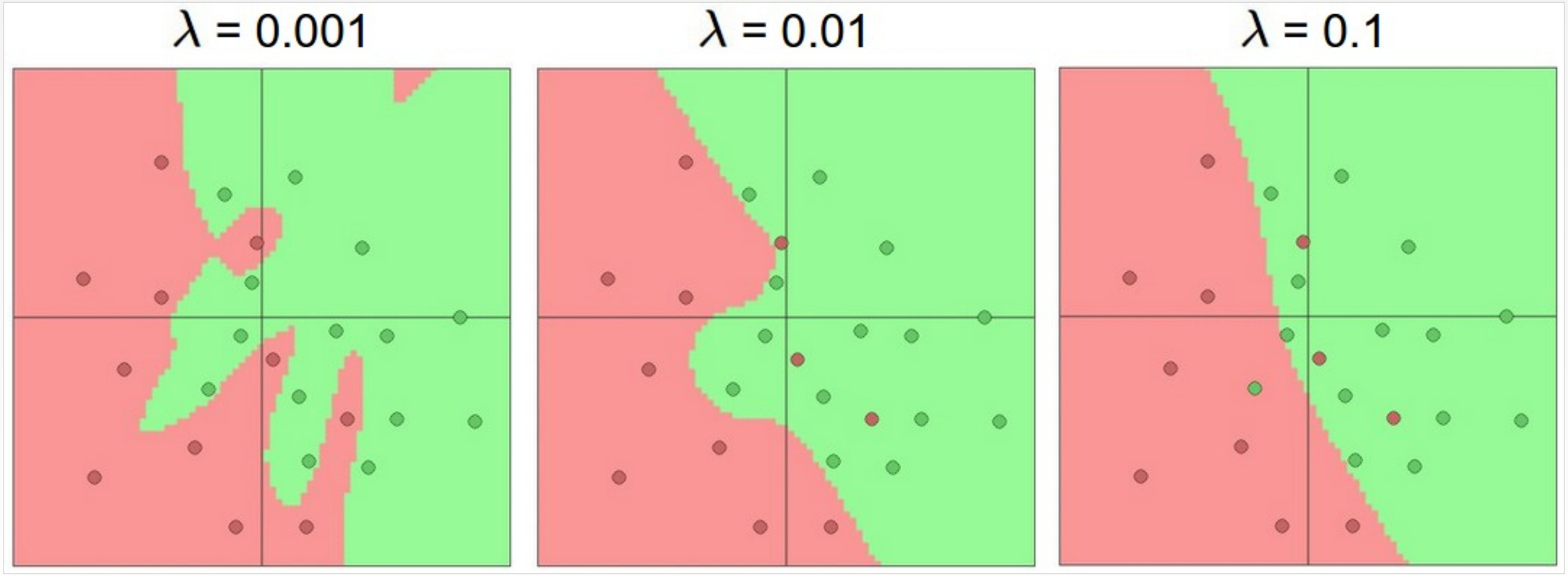
```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

SETTING NUMBER OF LAYERS & THEIR SIZES



more neurons = more capacity

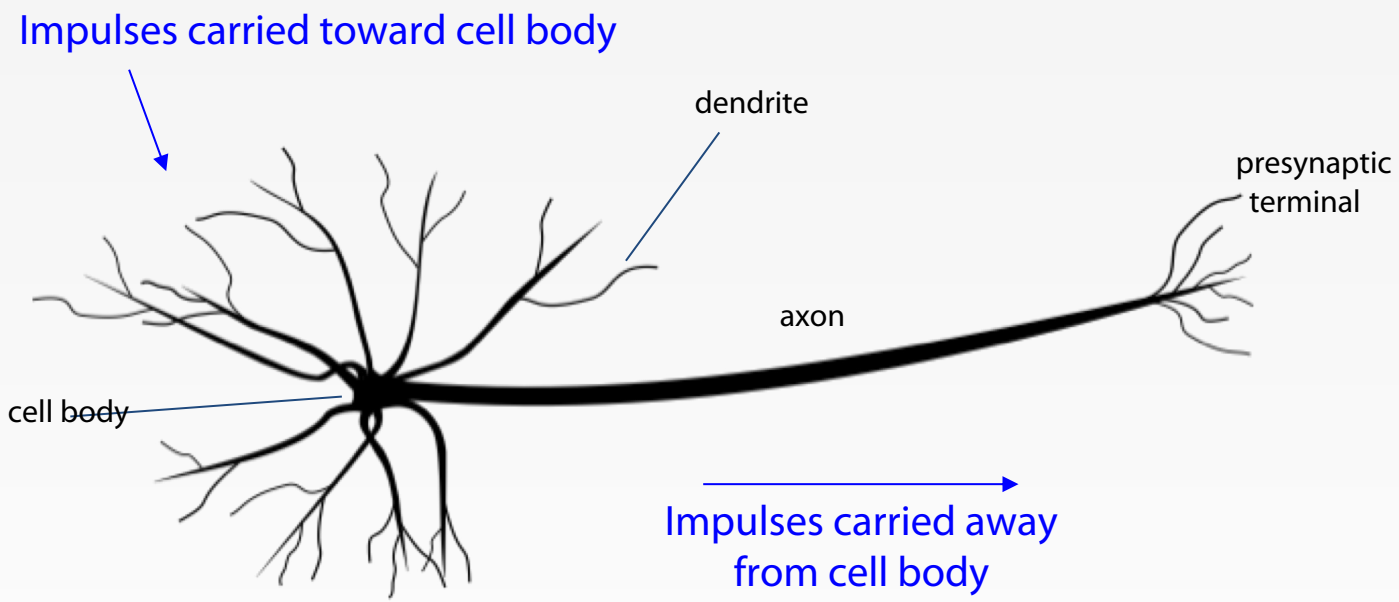
SETTING NUMBER OF LAYERS & THEIR SIZES



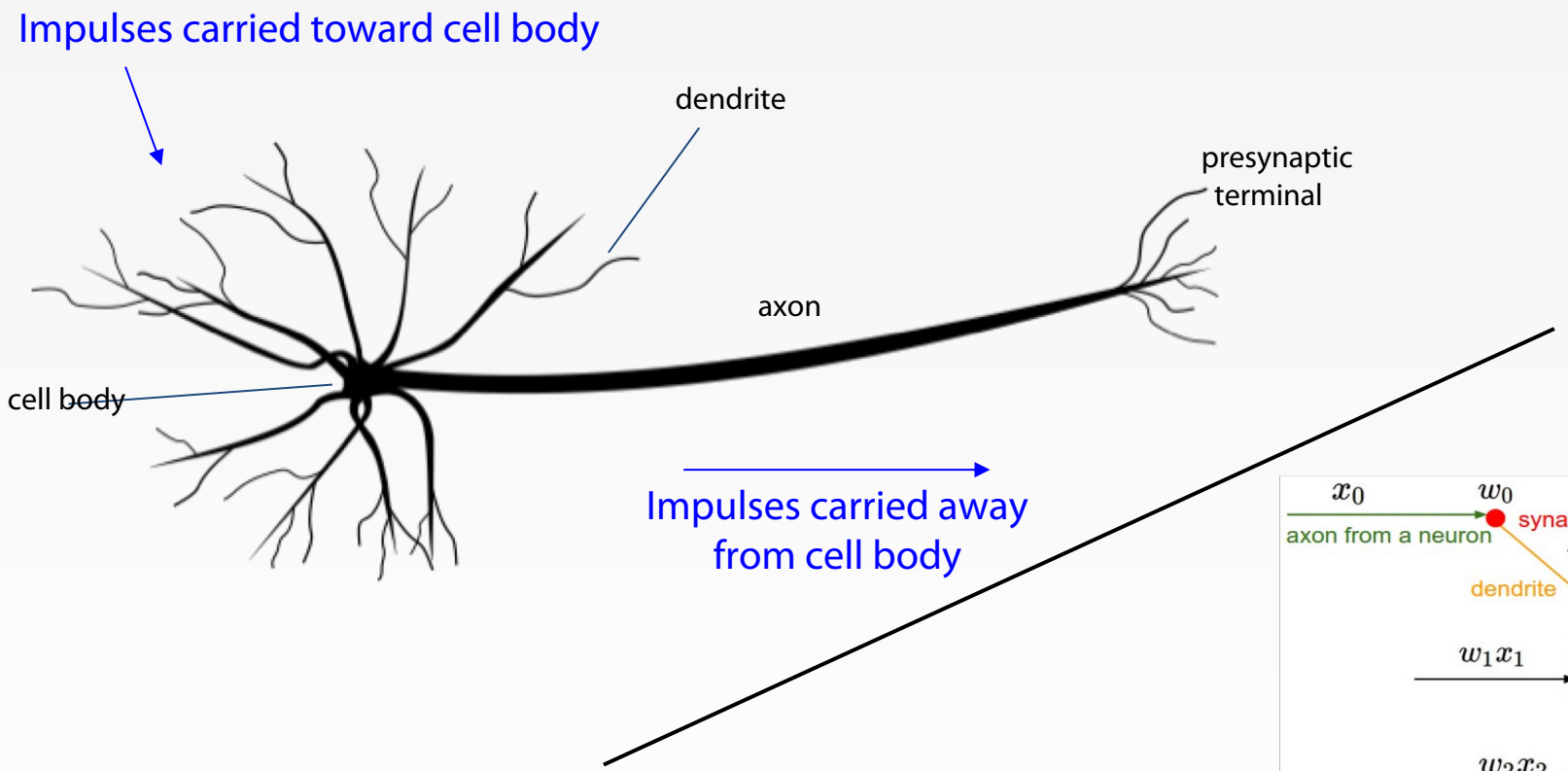
Do not use size of neural network as a regularizer. Use stronger regularization instead.

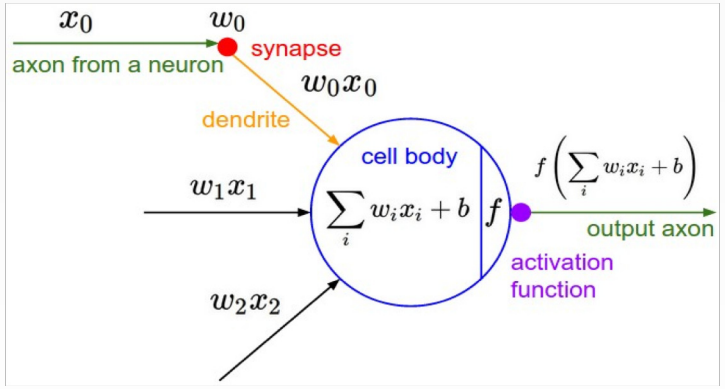
BIOLOGICAL INSPIRATION

BIOLOGICAL INSPIRATION

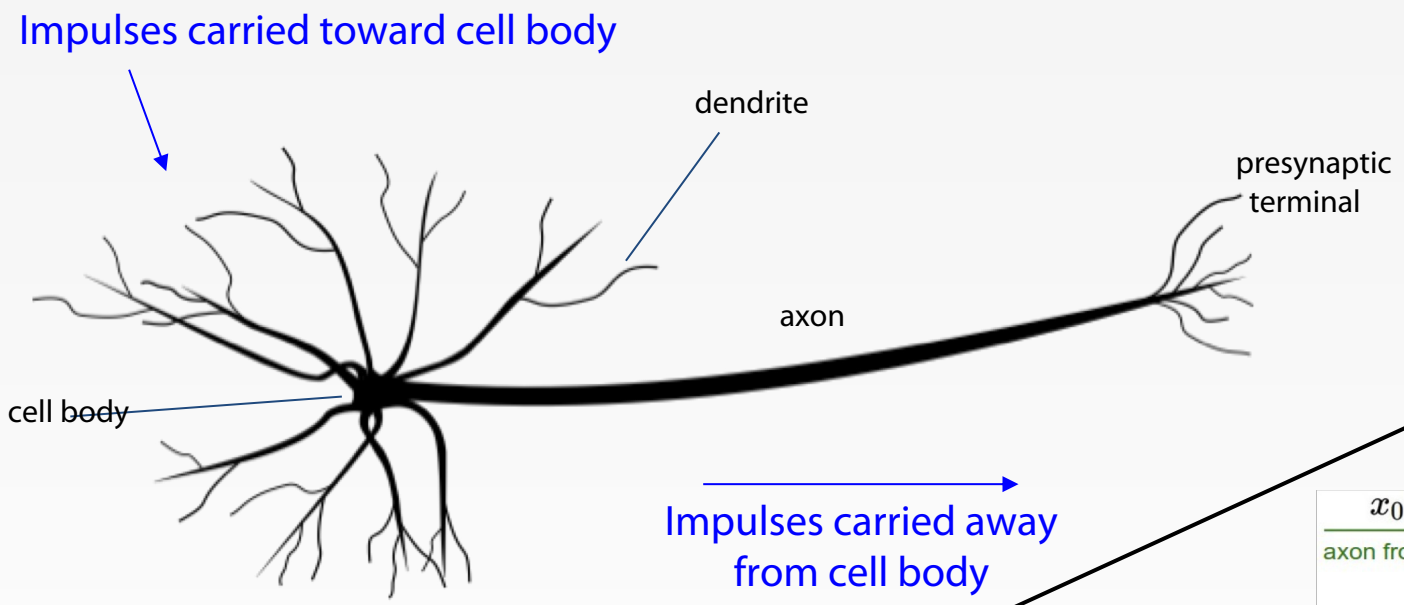


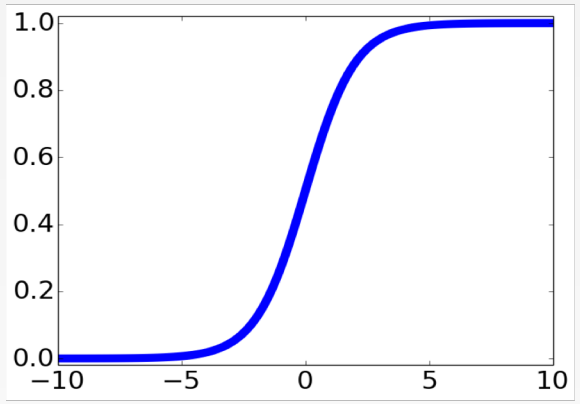
BIOLOGICAL INSPIRATION





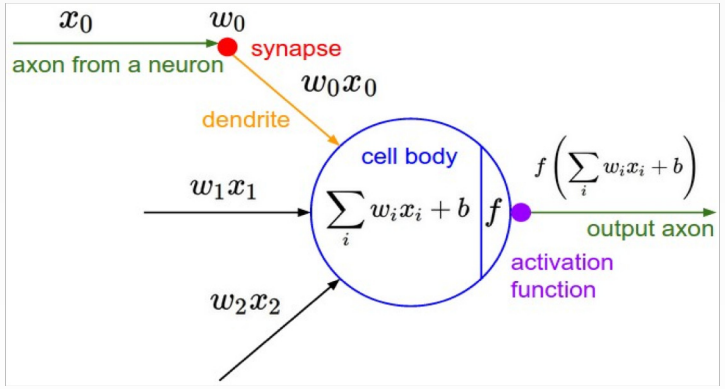
BIOLOGICAL INSPIRATION



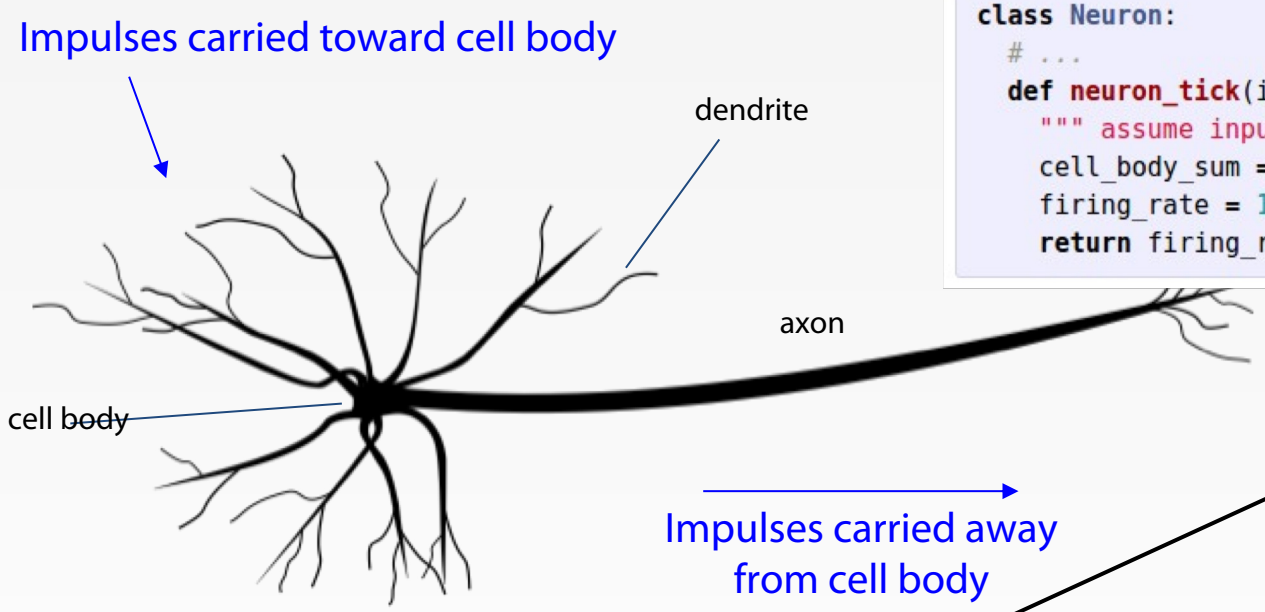


$$\frac{1}{1+e^{-x}}$$

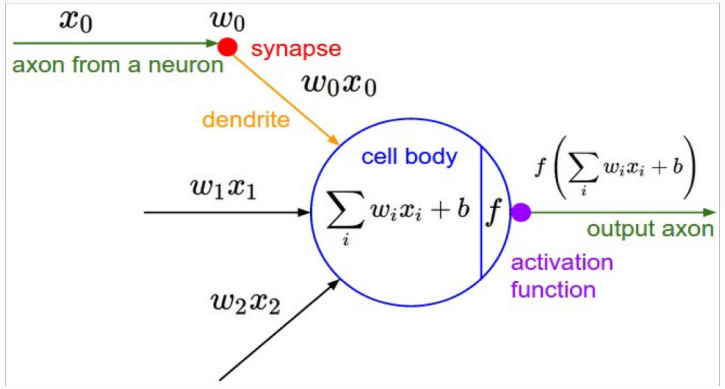
sigmoid activation function



BIOLOGICAL INSPIRATION

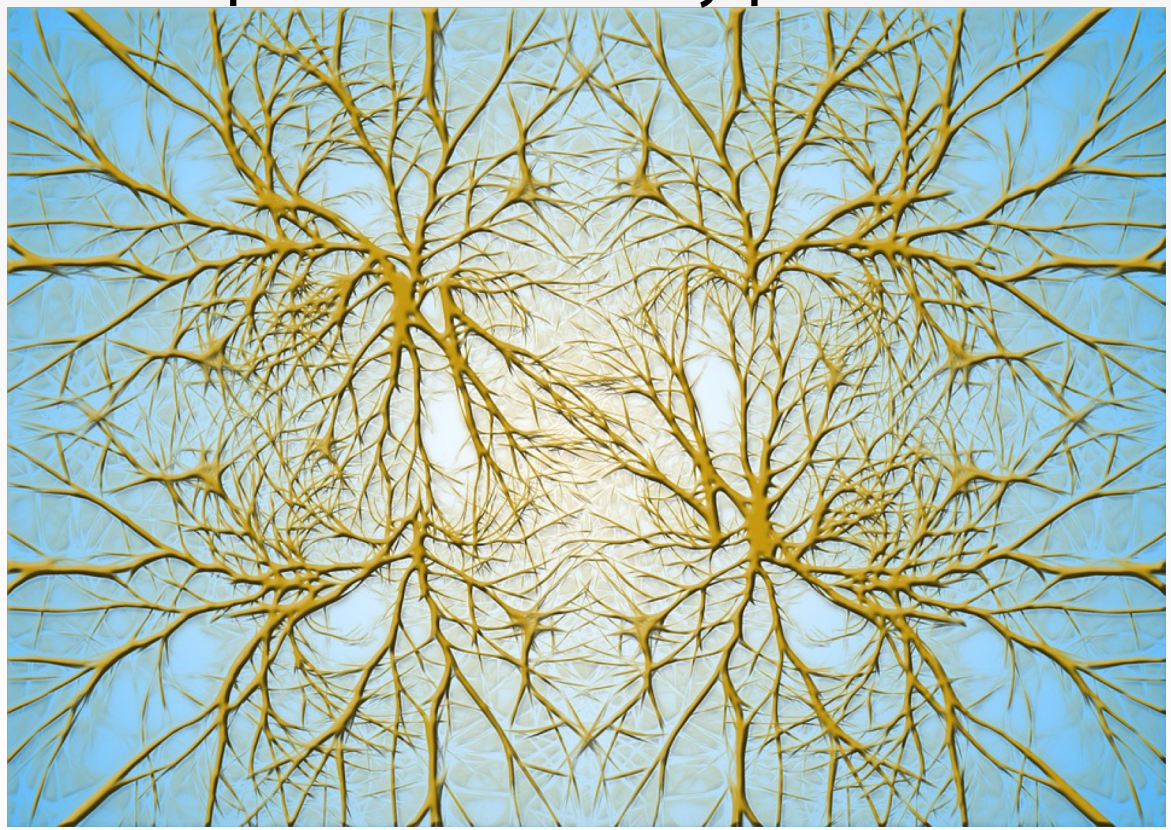



```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

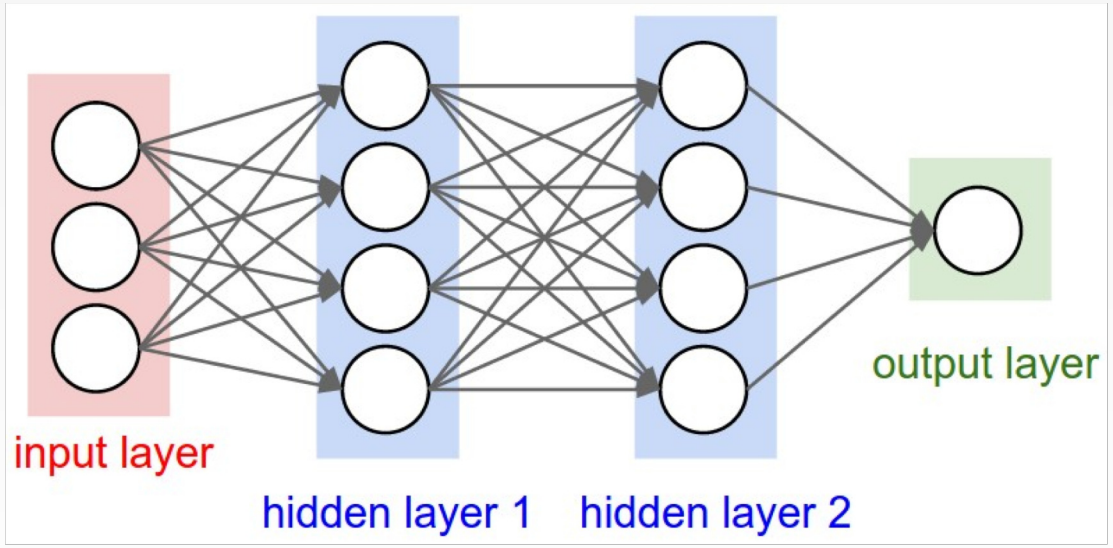


BIOLOGICAL INSPIRATION

Biological Neurons:
Complex connectivity patterns

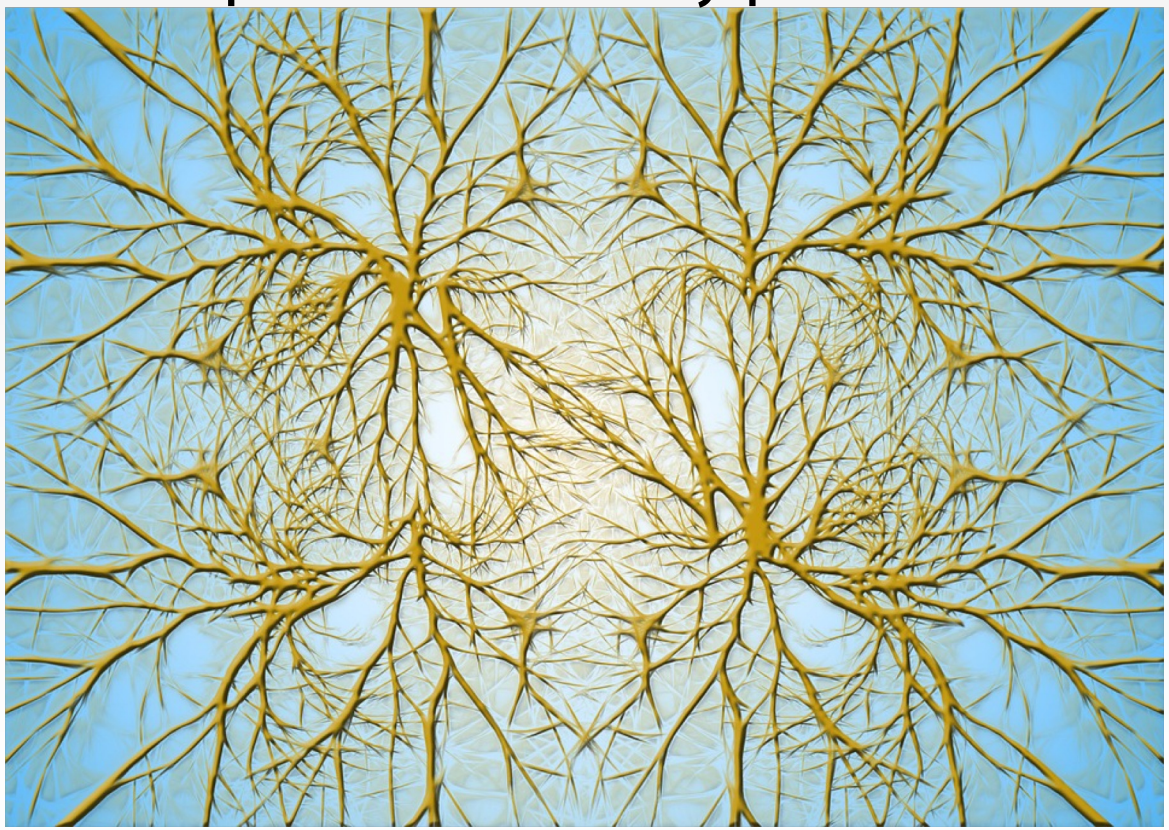


Neurons in a neural network:
Organized into regular layers for computational efficiency

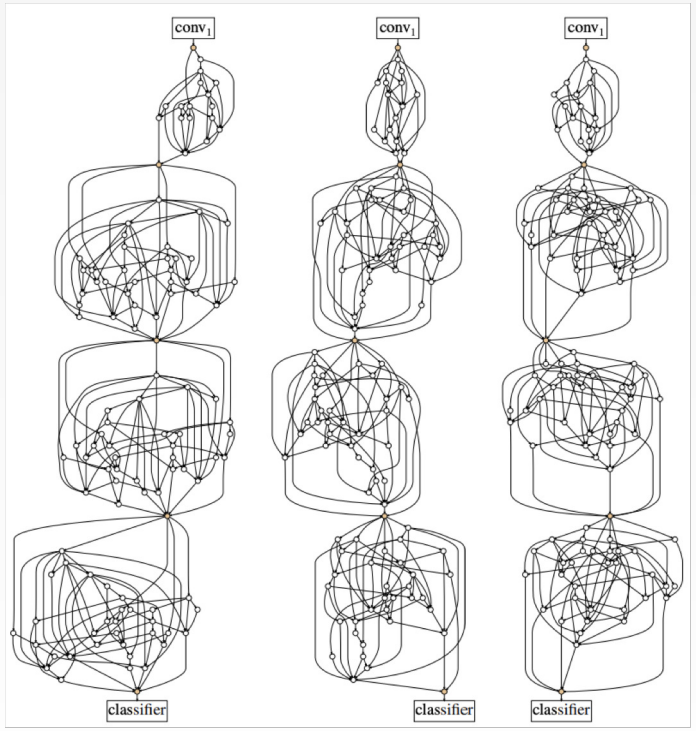


BIOLOGICAL INSPIRATION

Biological Neurons:
Complex connectivity patterns



But neural networks with random connections can work too!



Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”, arXiv 2019

BIOLOGICAL INSPIRATION

- Be very careful with your brain analogies!
- Biological Neurons
  - Many different types
  - Dendrites can perform complex non-linear computations
  - Synapses are not a single weight but a complex non-linear dynamical system
  - Interpreting activation function as a firing rate may not be adequate

BACKPROP

PROBLEM: HOW TO COMPUTE GRADIENTS?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

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

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: Data loss + Regularization loss

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

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

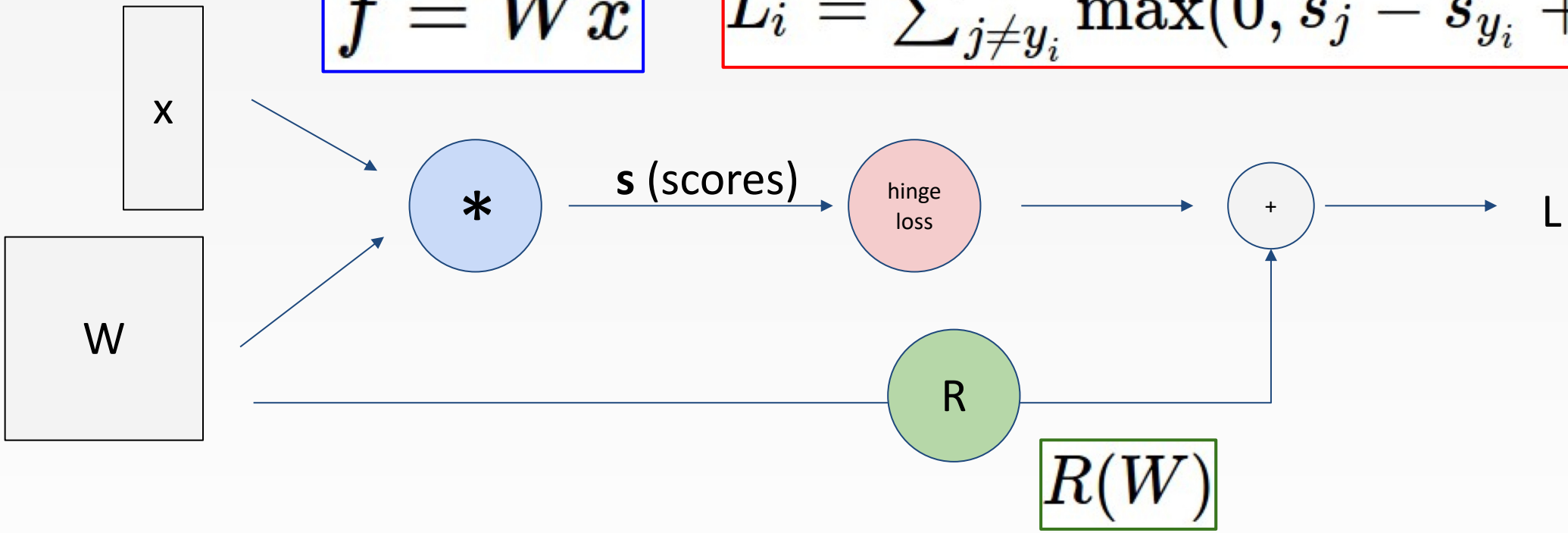
Problem: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

Problem: Not feasible for very complex models!

BETTER IDEA: COMPUTATIONAL GRAPHS + BACKPROPAGATION

$$f = Wx$$

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

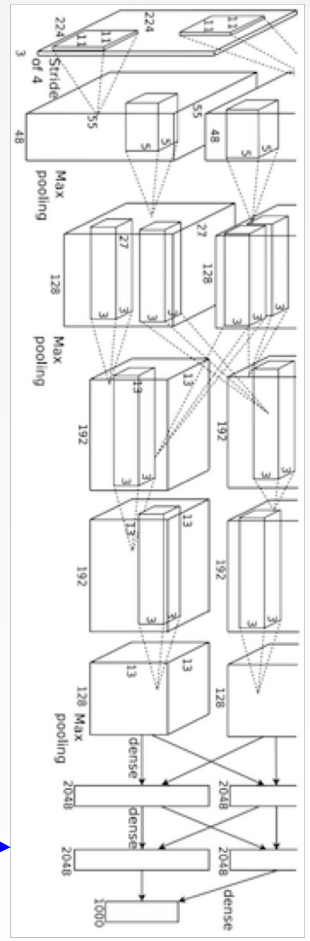


CONVOLUTIONAL NETWORK (ALEXNET)

input image

weights

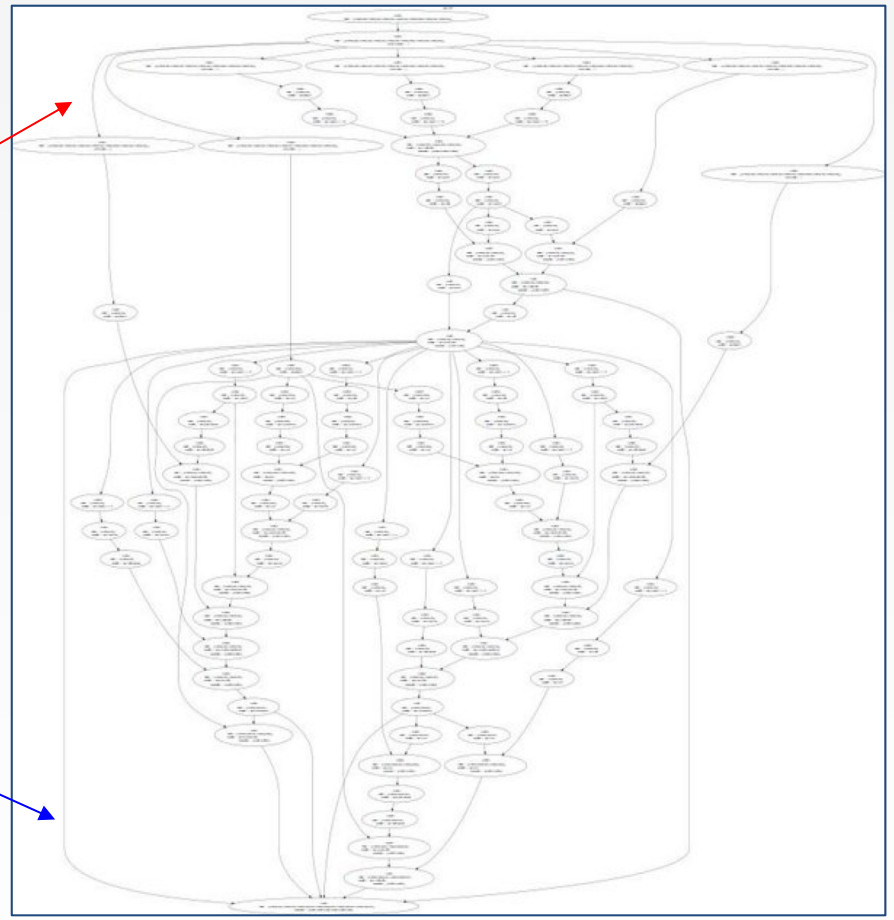
loss



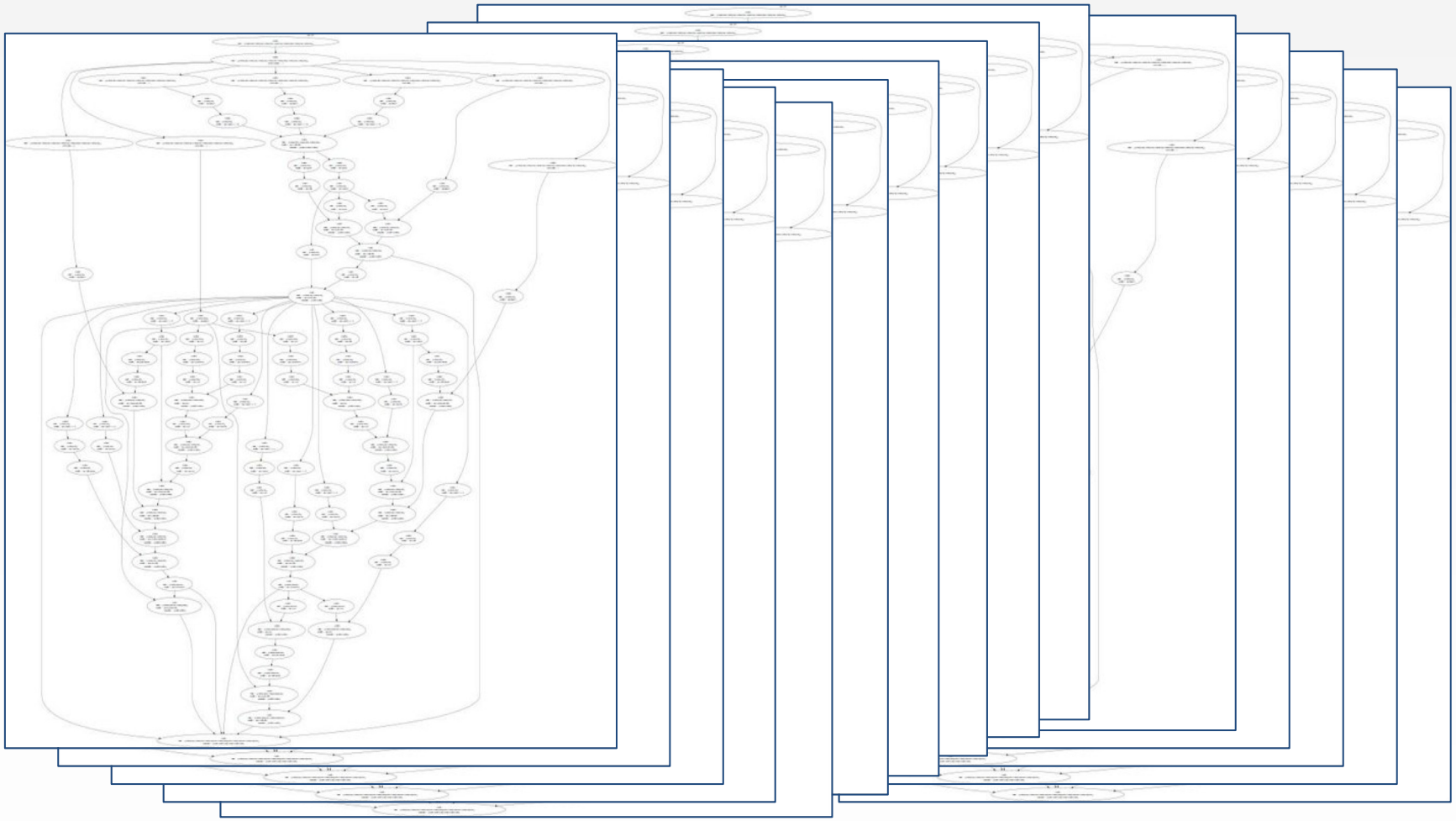
NEURAL TURING MACHINE

input image

loss



NEURAL TURING MACHINE

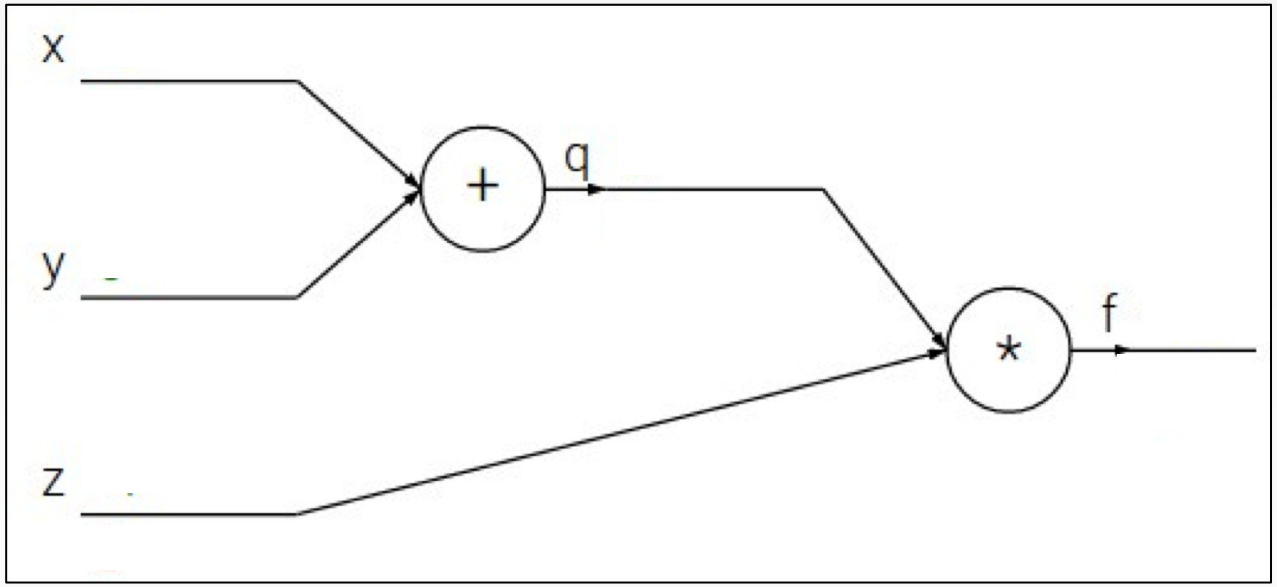


BACKPROPAGATION: A SIMPLE EXAMPLE

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

BACKPROPAGATION: A SIMPLE EXAMPLE

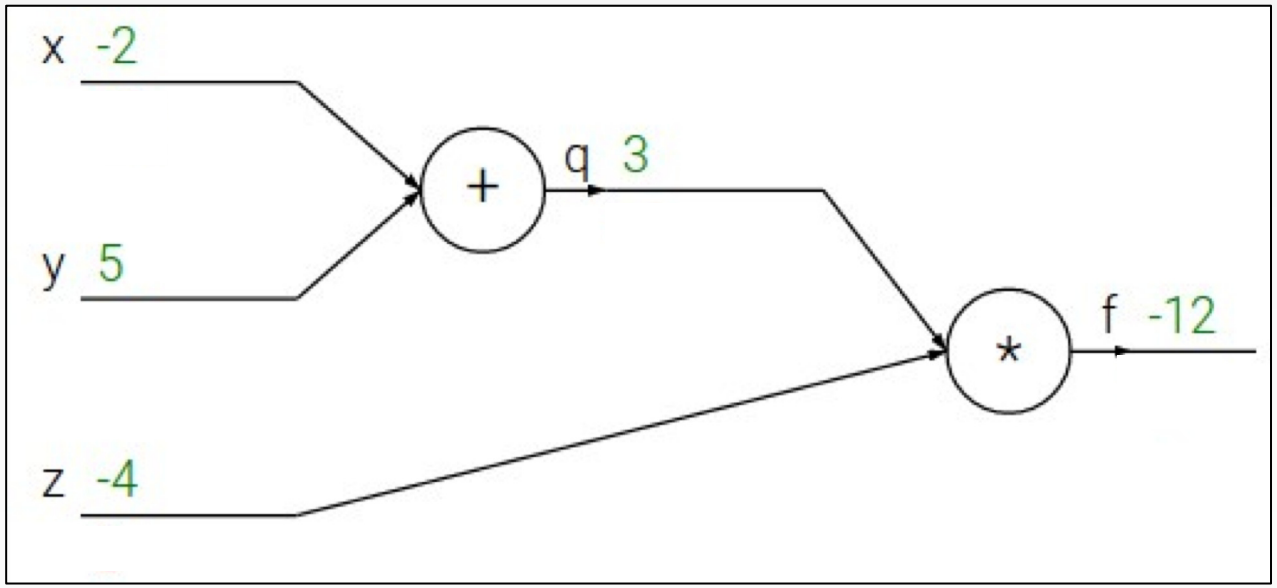
$$f(x, y, z) = (x + y)z$$



BACKPROPAGATION: A SIMPLE EXAMPLE

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

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



BACKPROPAGATION: A SIMPLE EXAMPLE

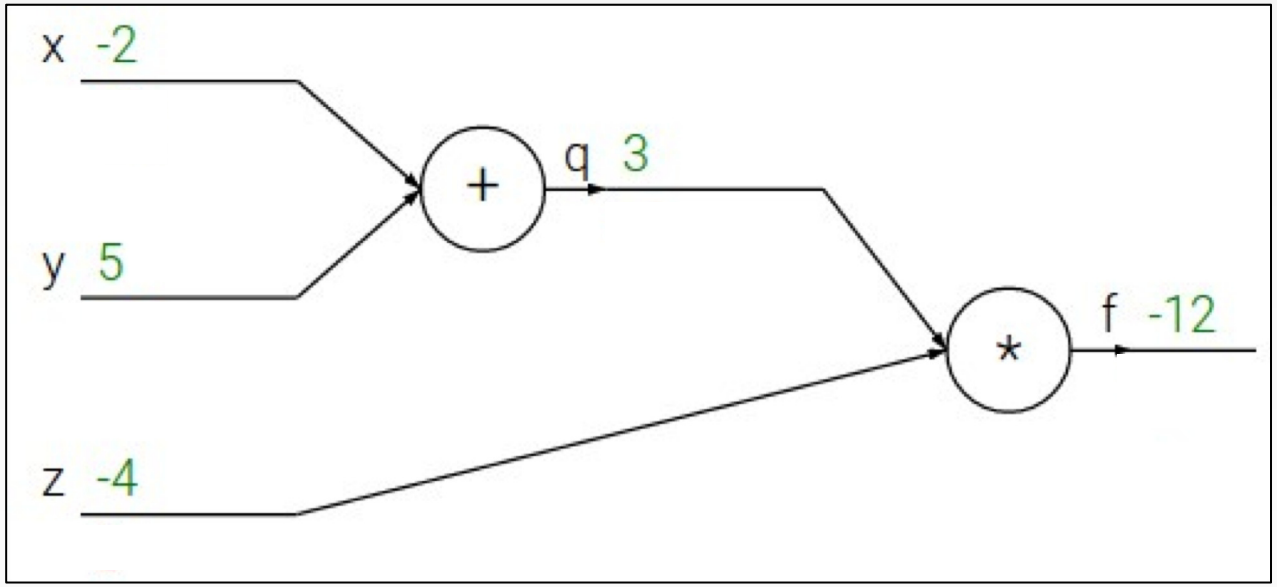
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



BACKPROPAGATION: A SIMPLE EXAMPLE

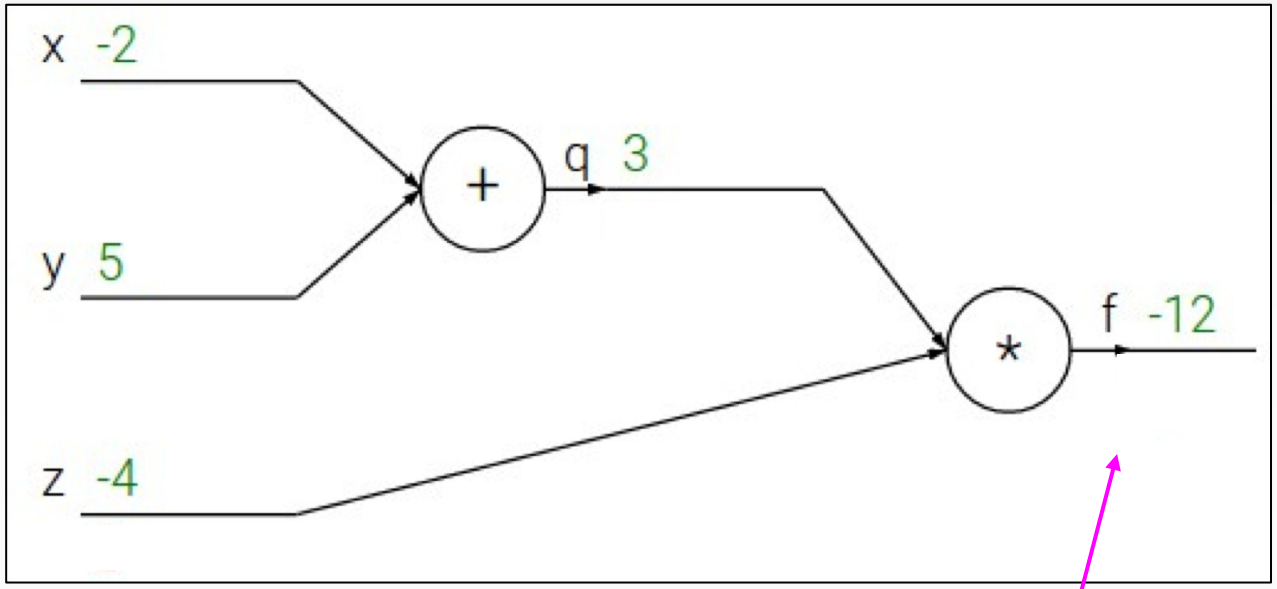
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

BACKPROPAGATION: A SIMPLE EXAMPLE

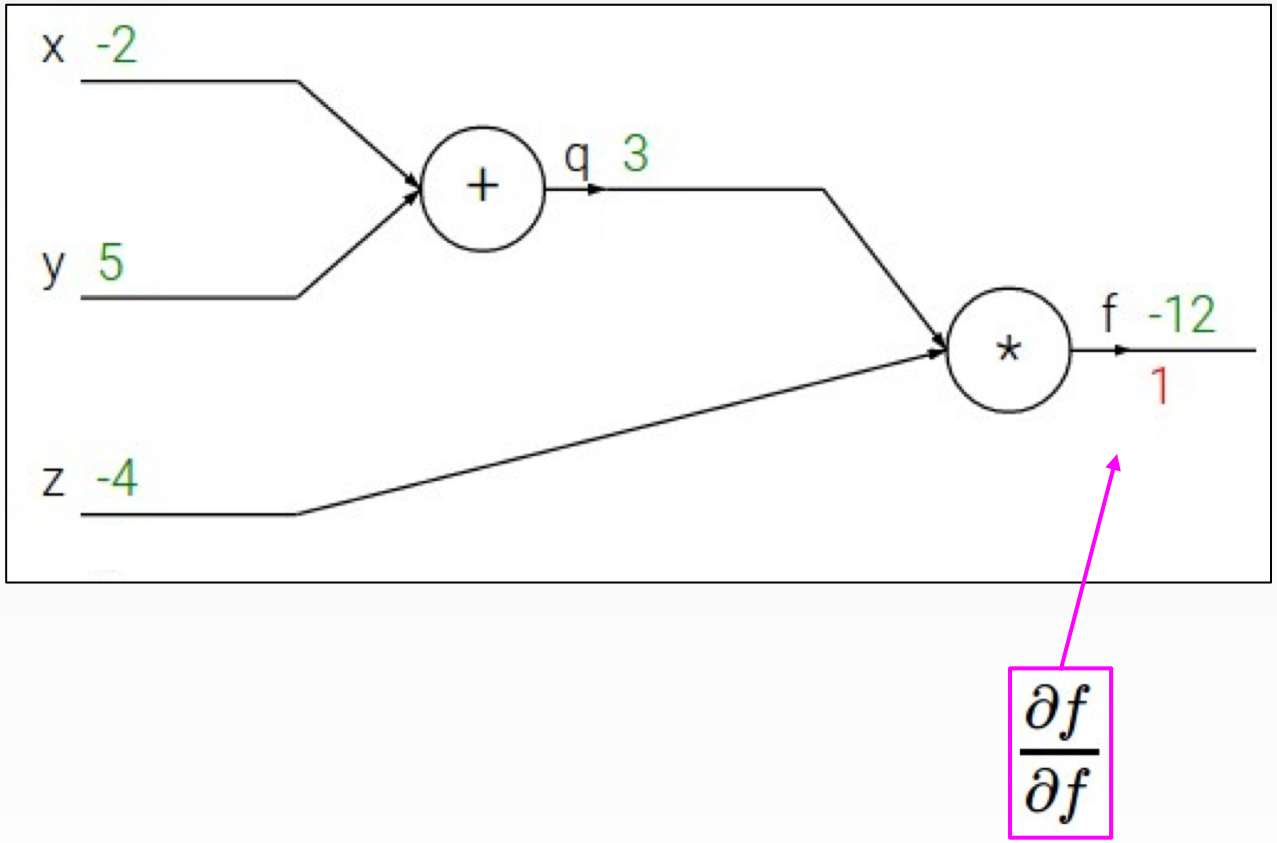
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



BACKPROPAGATION: A SIMPLE EXAMPLE

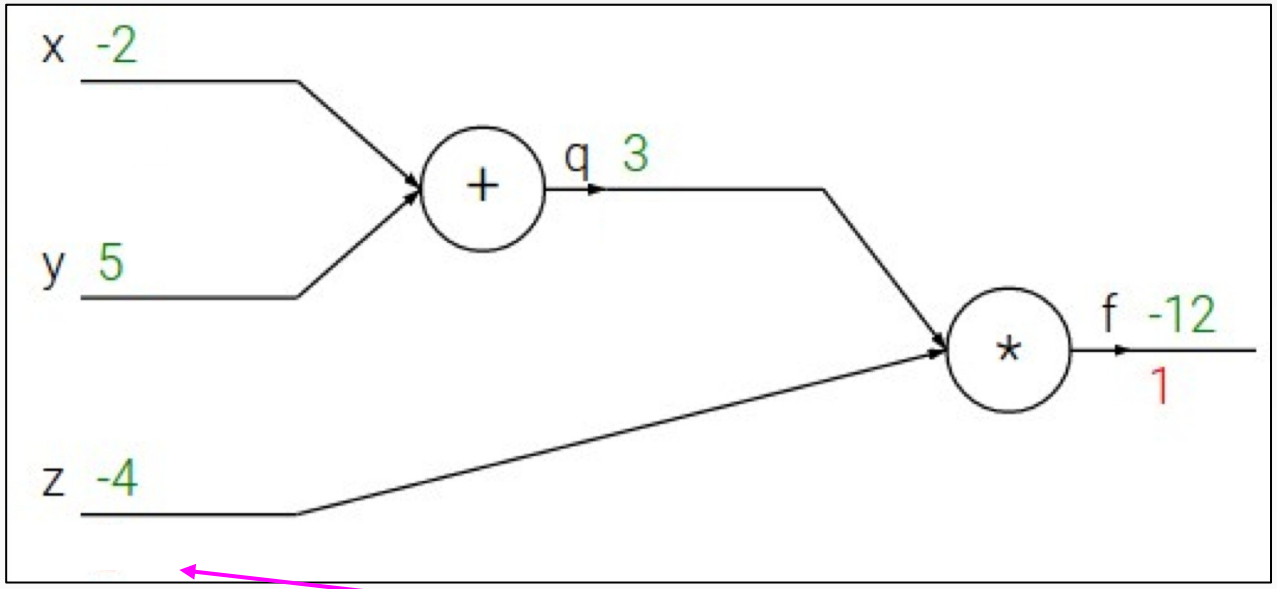
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

BACKPROPAGATION: A SIMPLE EXAMPLE

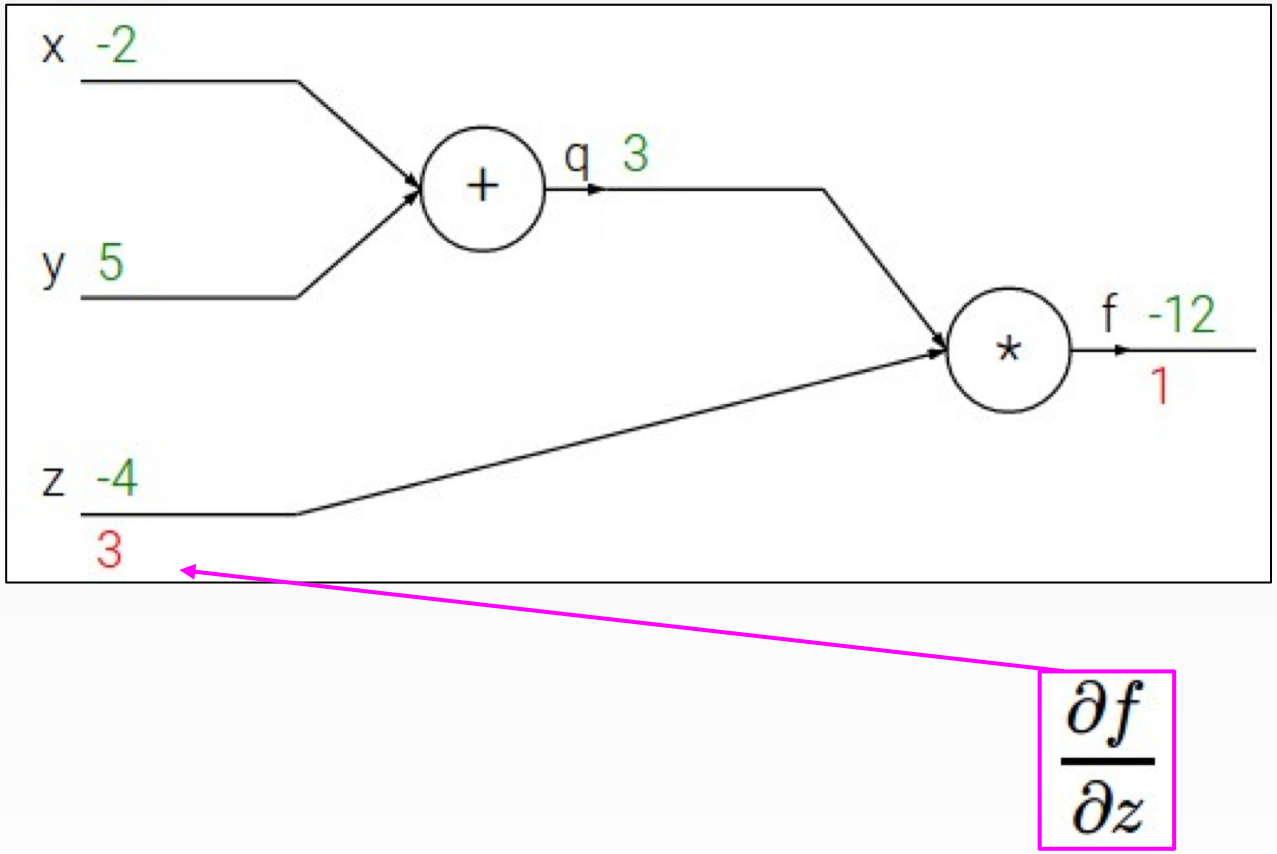
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



BACKPROPAGATION: A SIMPLE EXAMPLE

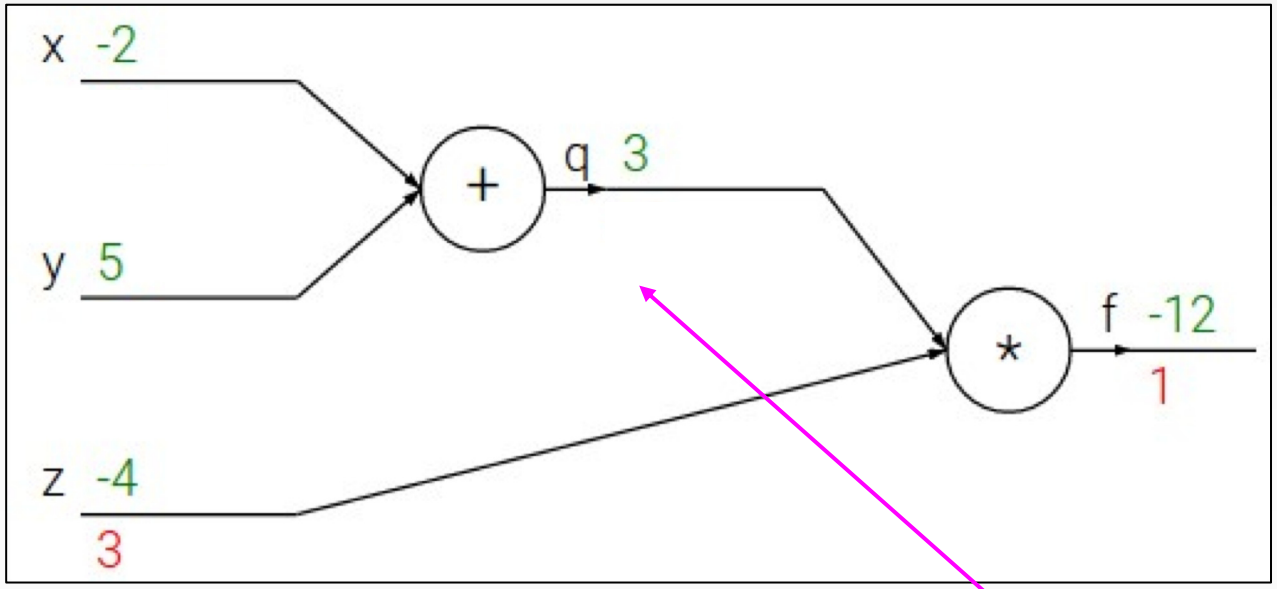
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

BACKPROPAGATION: A SIMPLE EXAMPLE

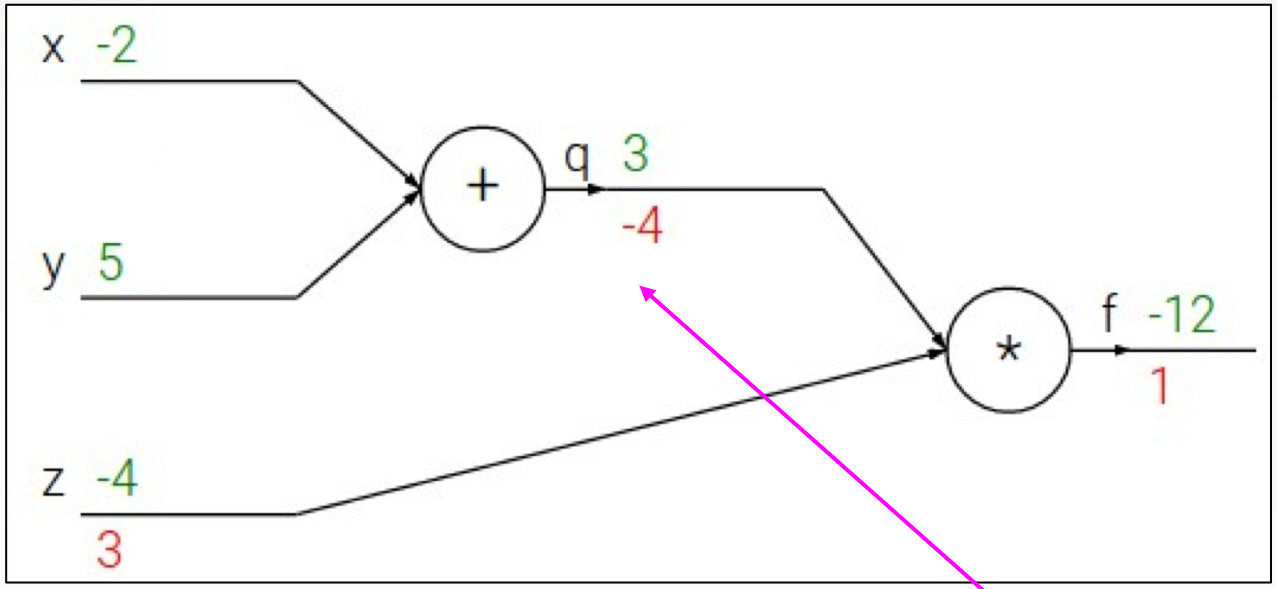
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

BACKPROPAGATION: A SIMPLE EXAMPLE

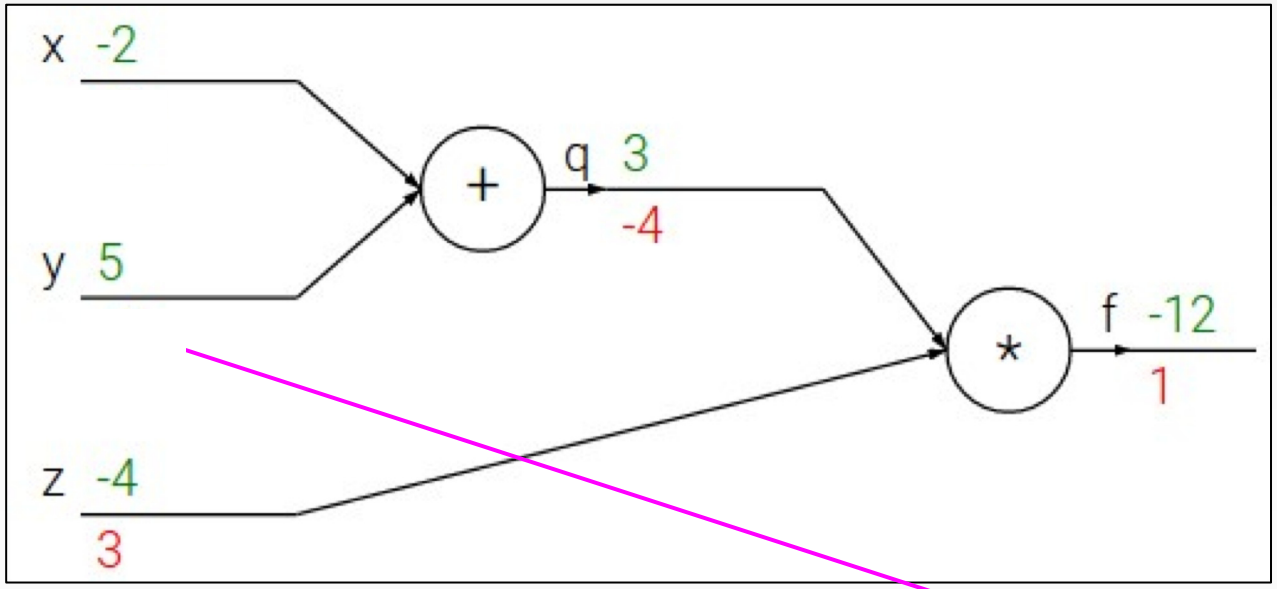
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



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

BACKPROPAGATION: A SIMPLE EXAMPLE

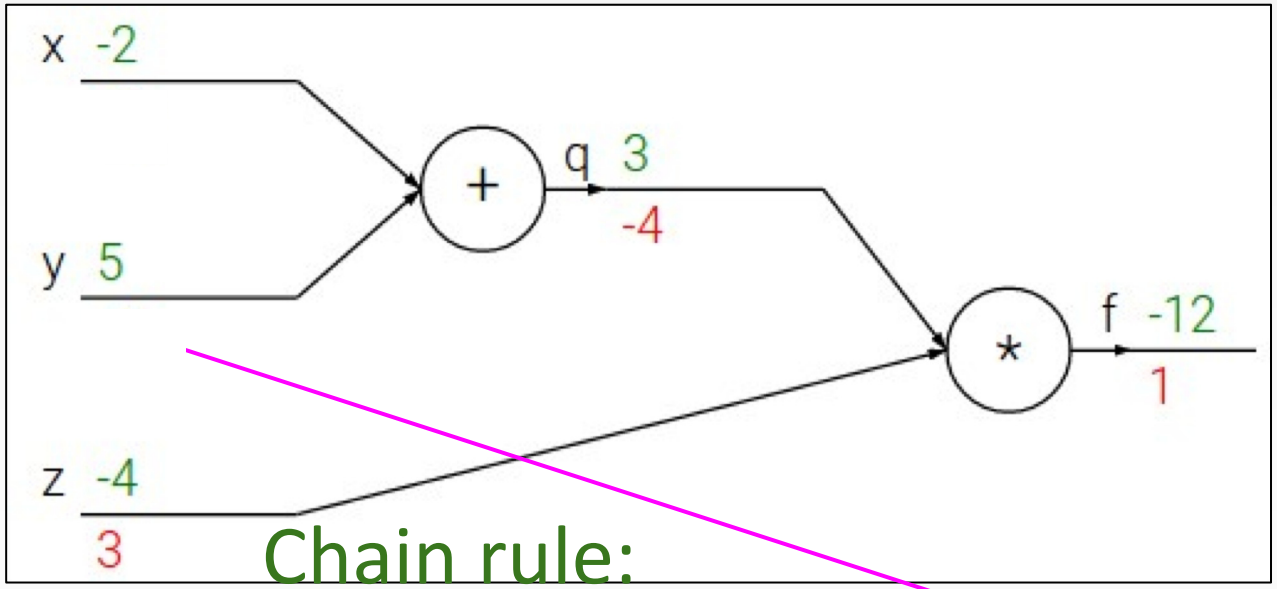
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Chain rule:

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

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

Upstream gradient

Local gradient

BACKPROPAGATION: A SIMPLE EXAMPLE

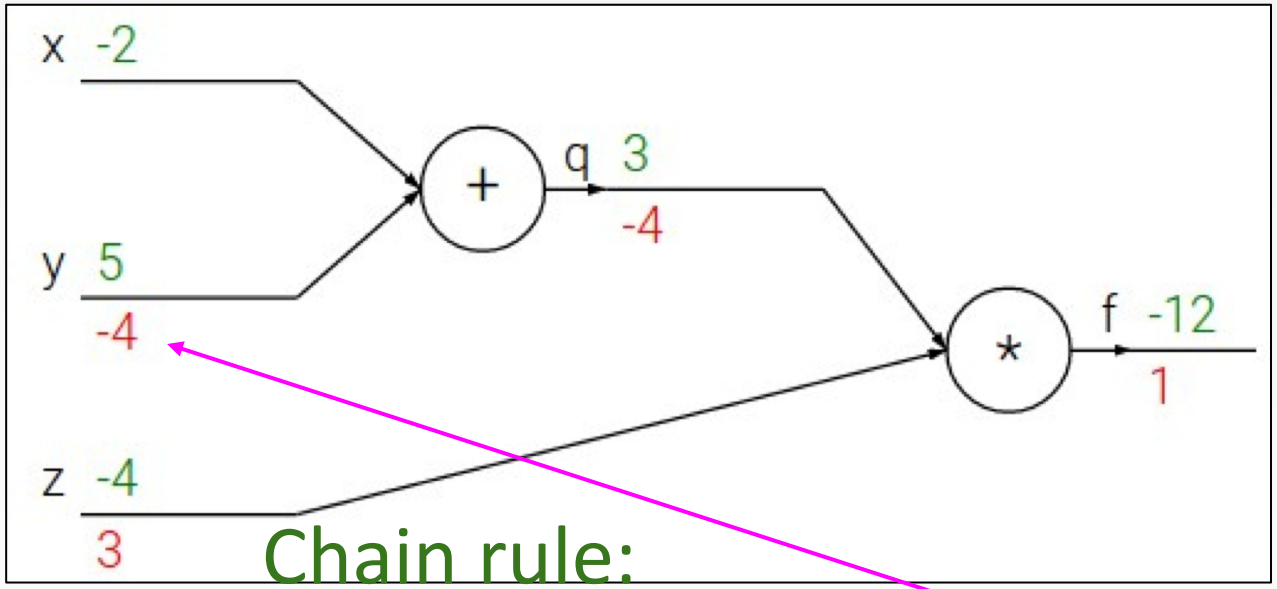
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Chain rule:

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

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

Upstream gradient

Local gradient

BACKPROPAGATION: A SIMPLE EXAMPLE

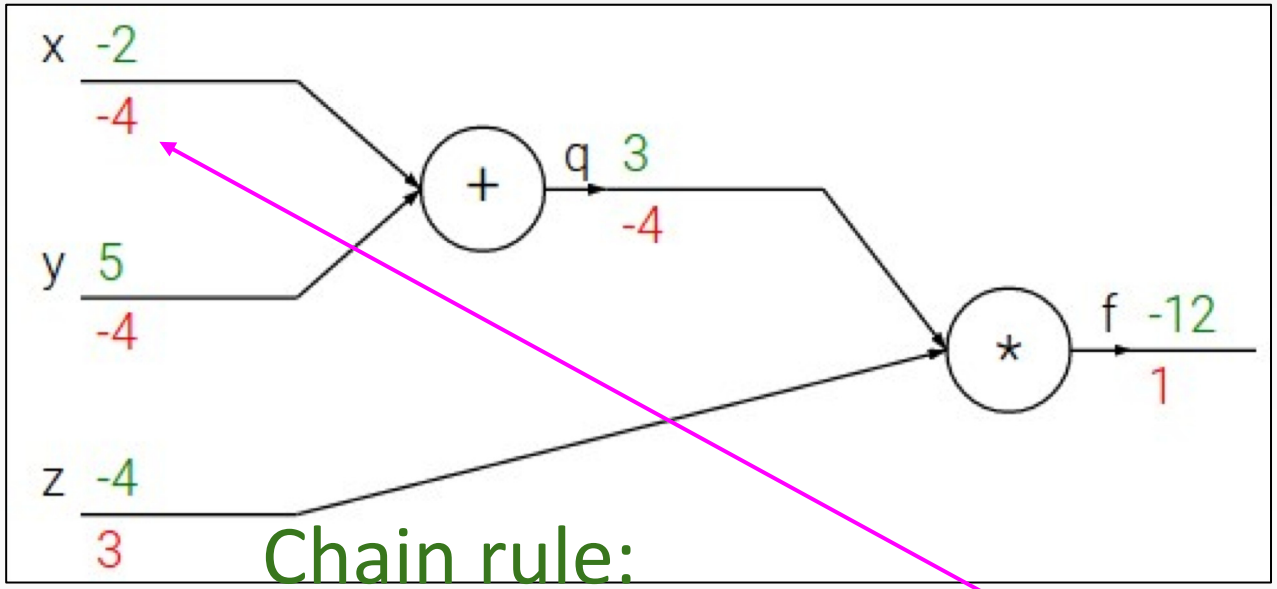
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



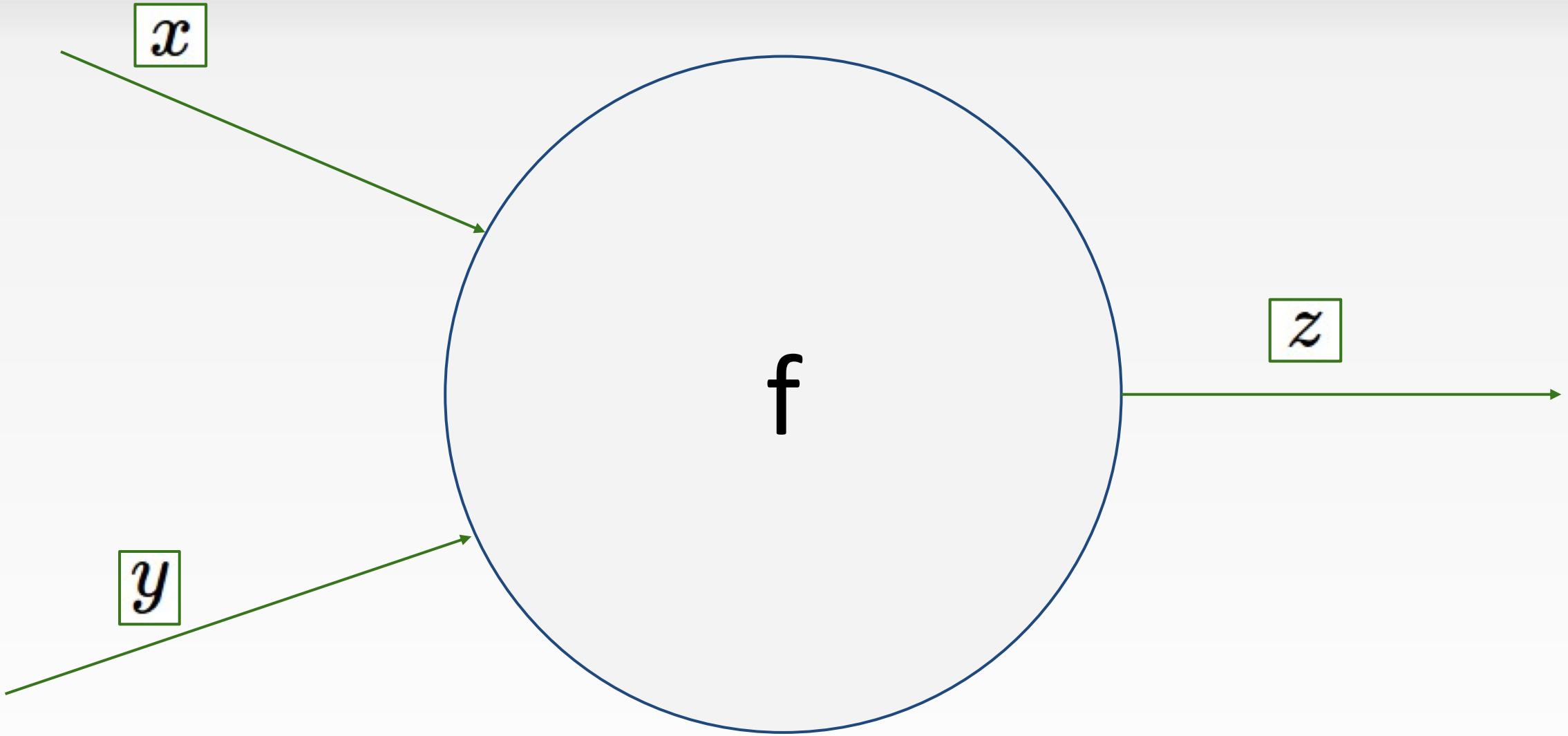
Chain rule:

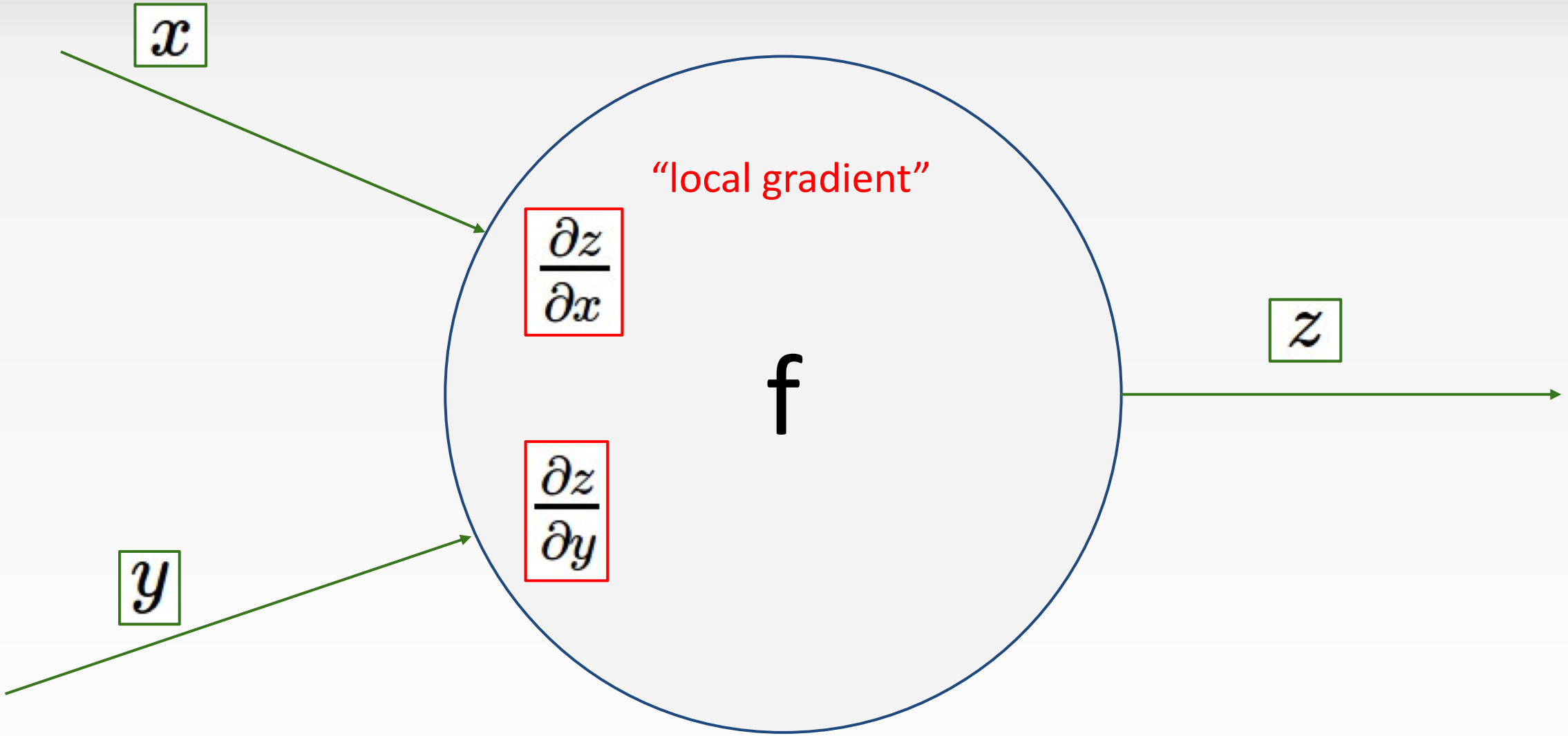
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

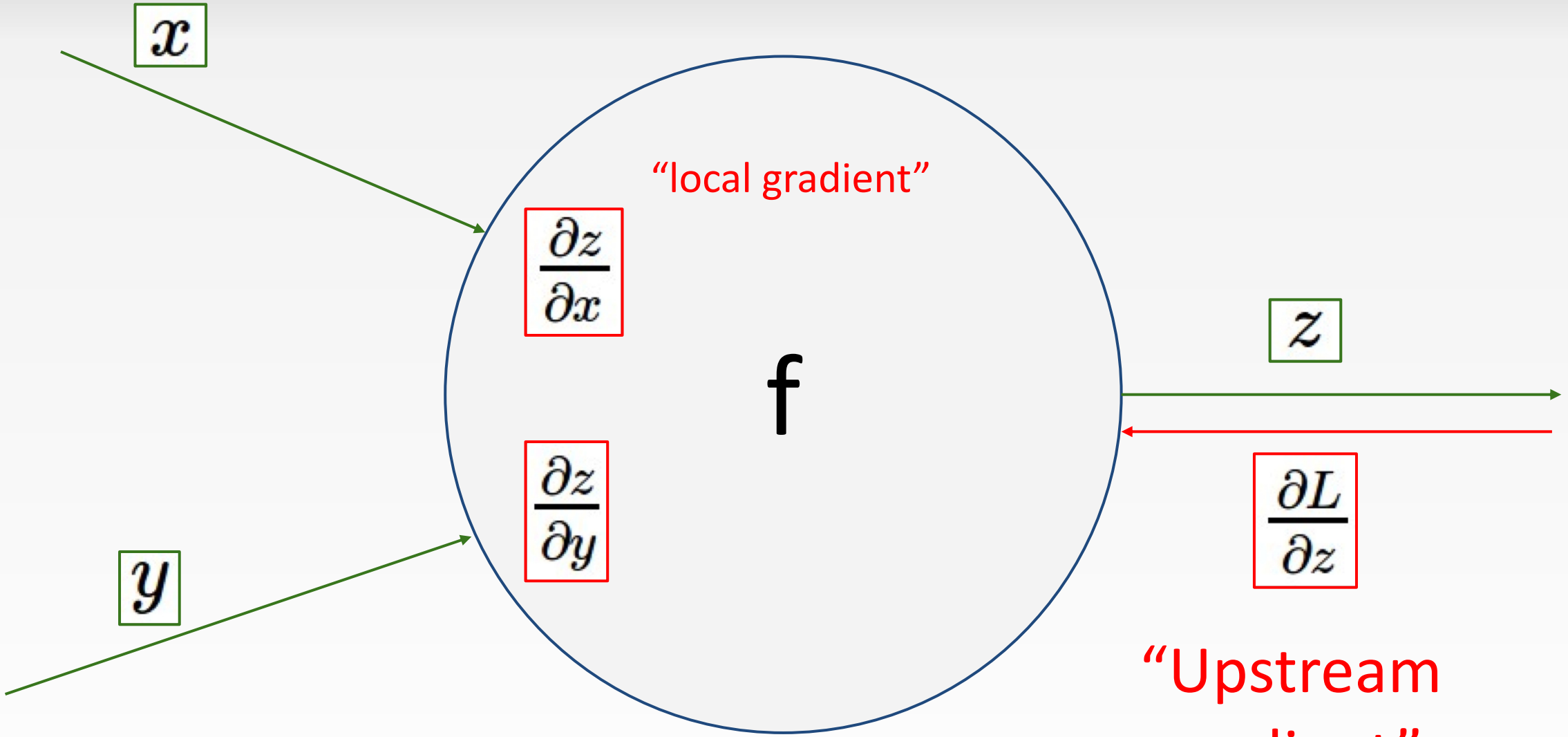
Upstream gradient

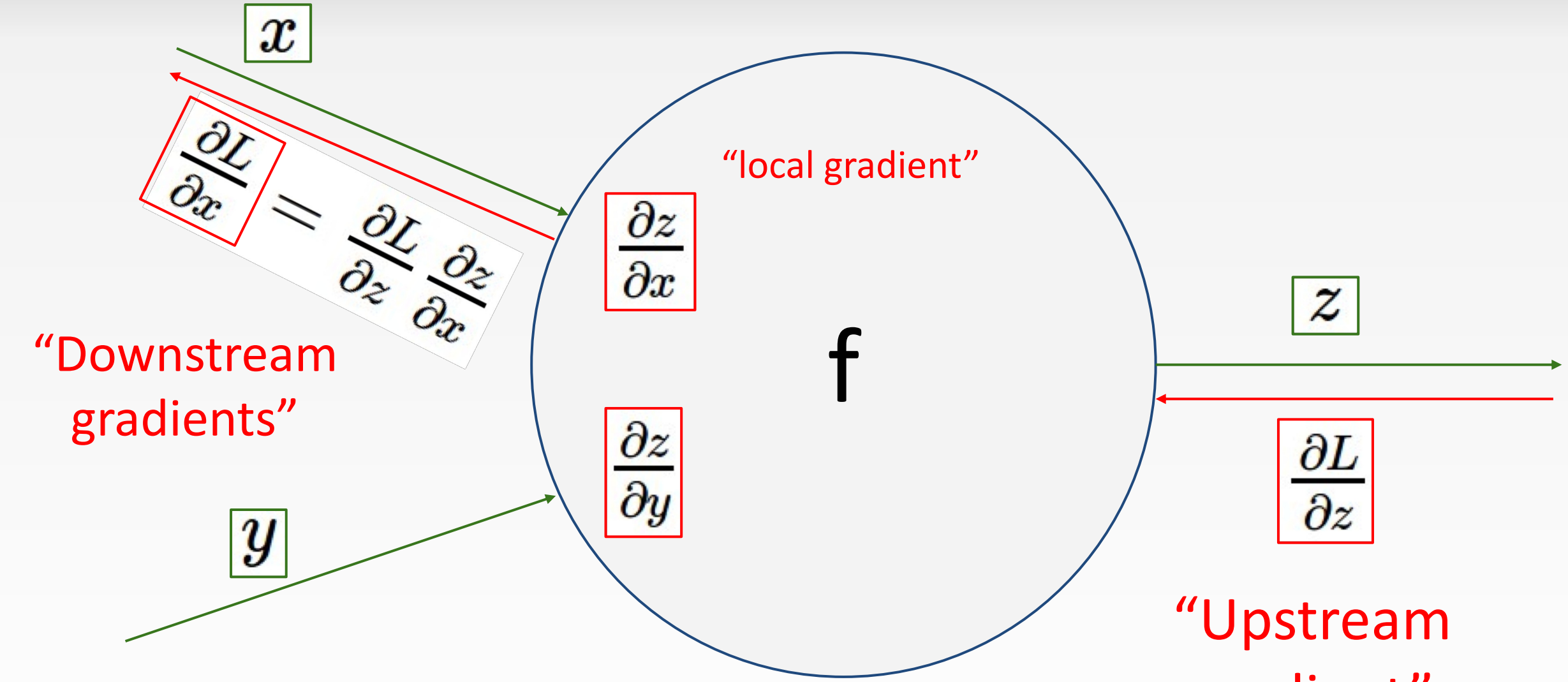
Local gradient

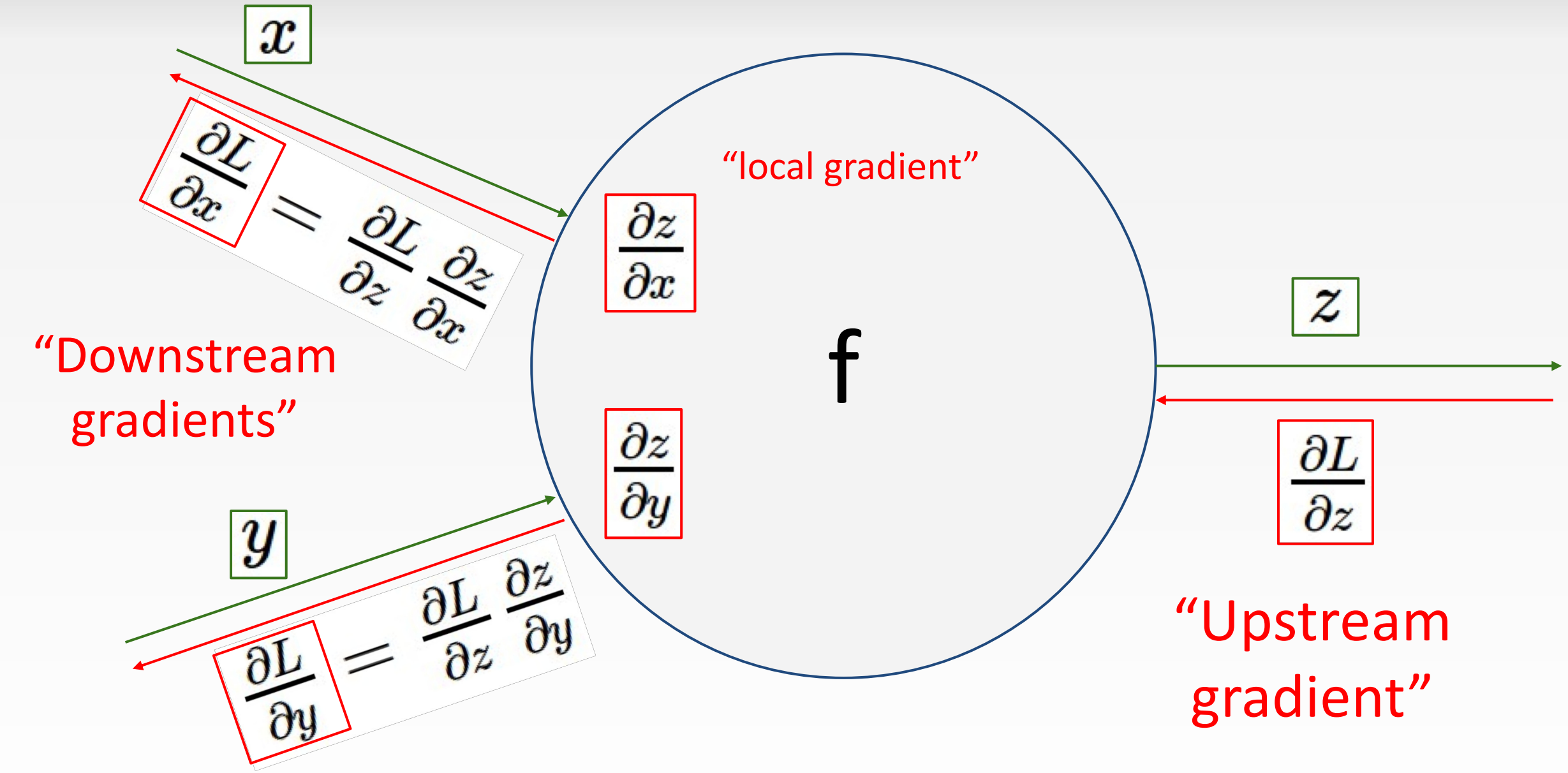
$\frac{\partial f}{\partial x}$

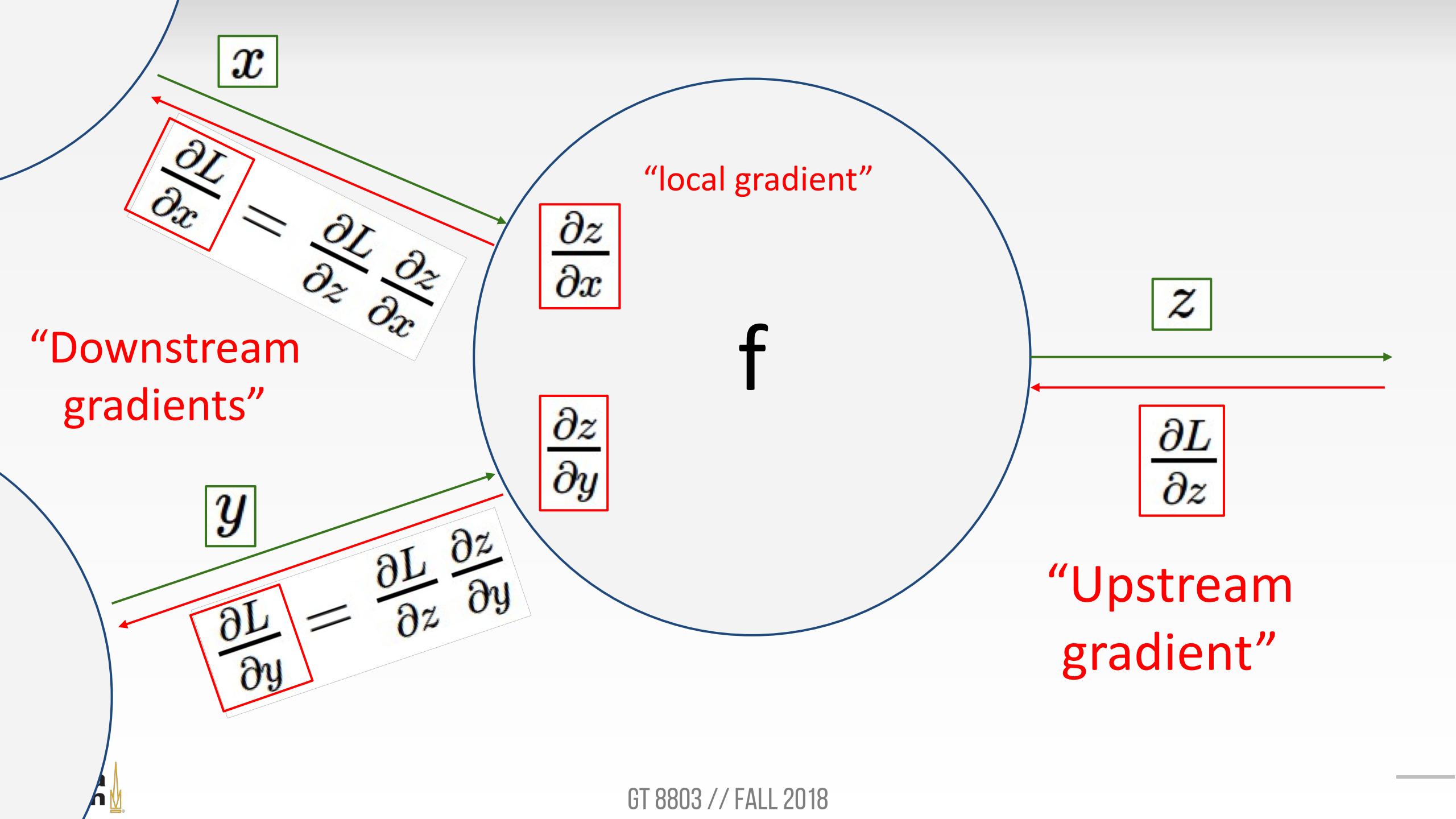

x
f
z
y


x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y


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

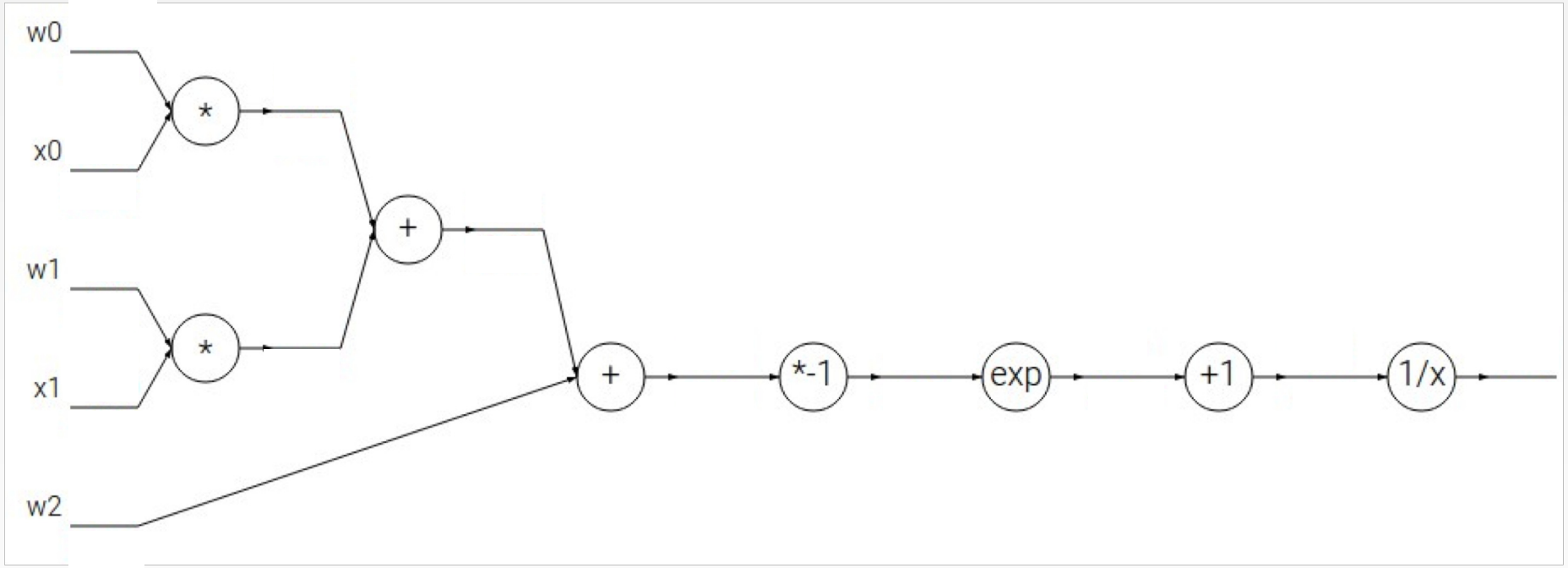

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
y
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"


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


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

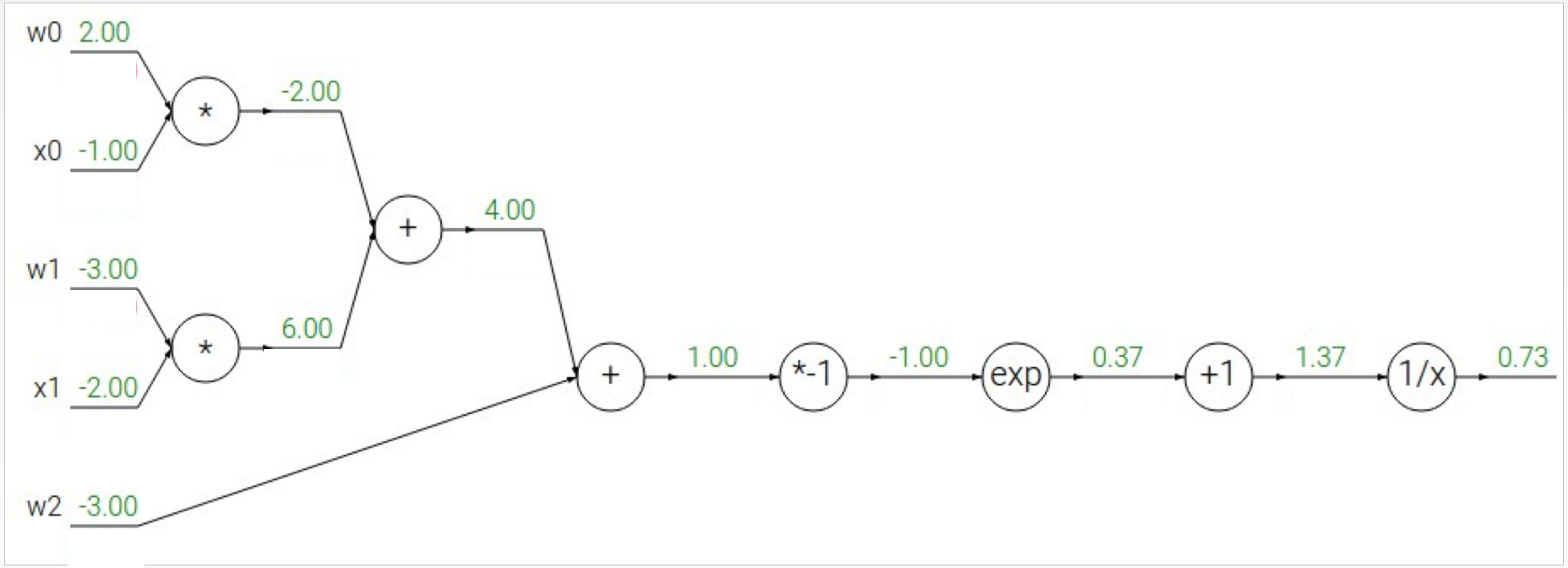
ANOTHER EXAMPLE:

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



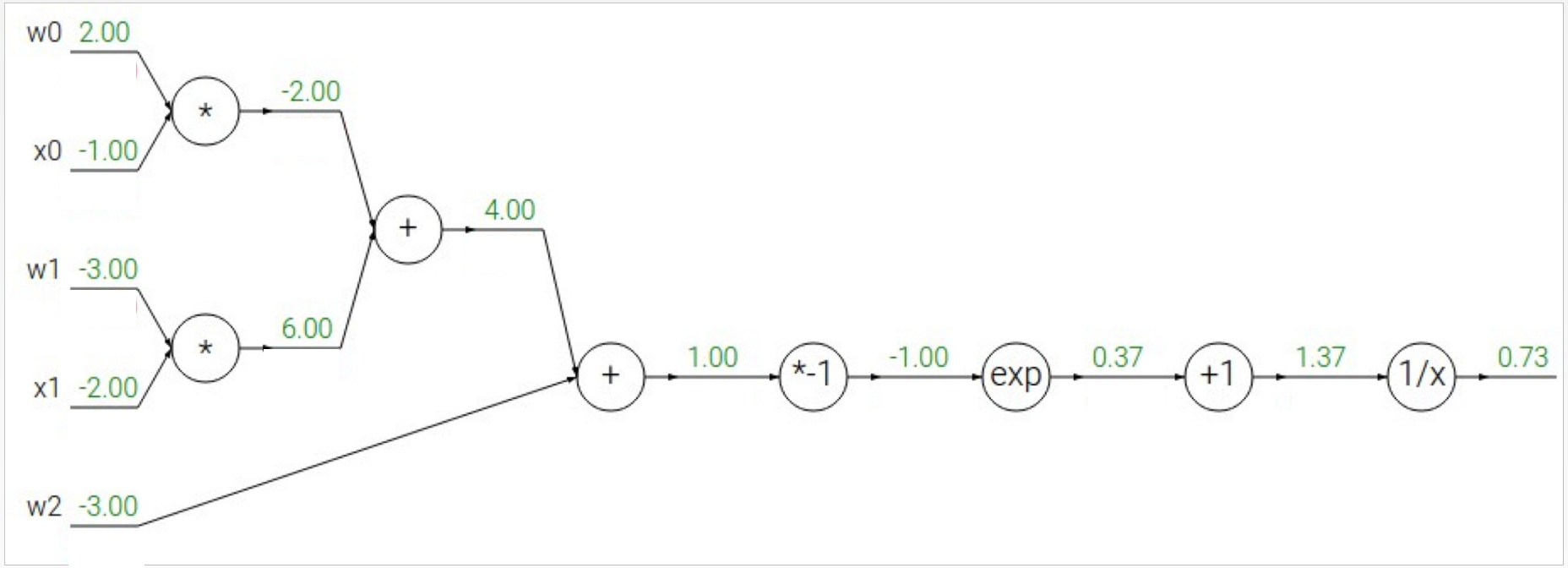
ANOTHER EXAMPLE:

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



ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

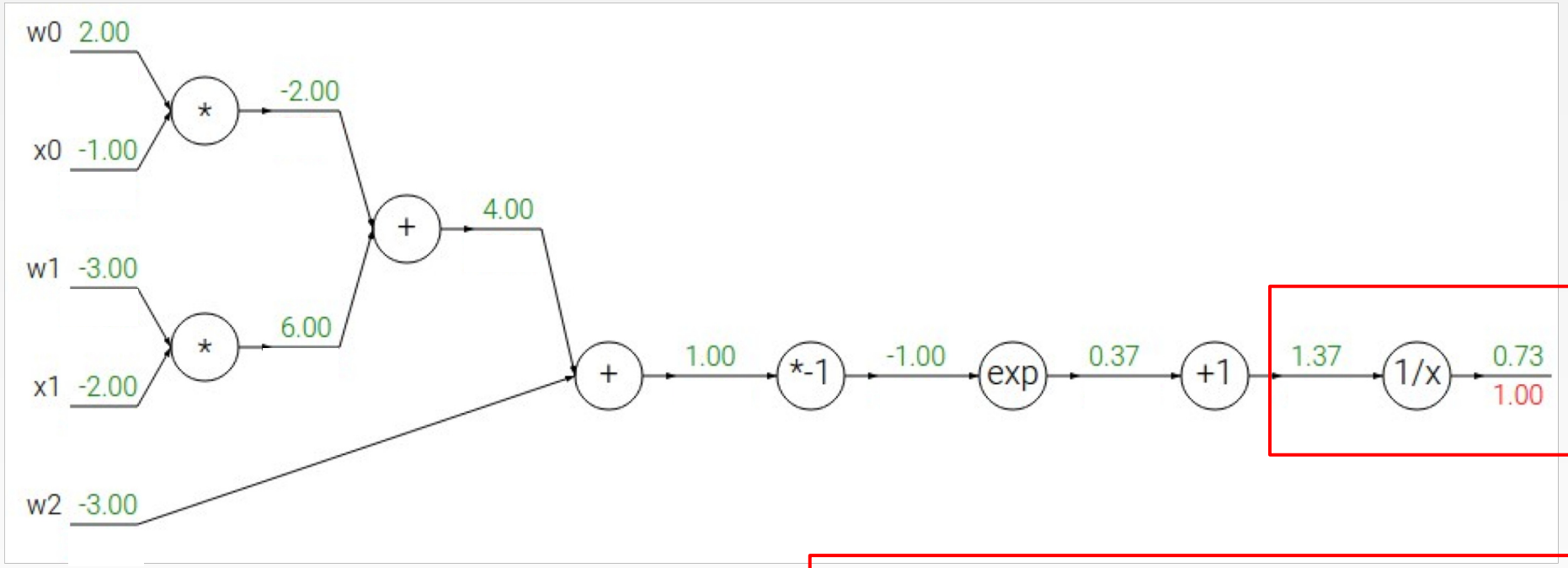
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

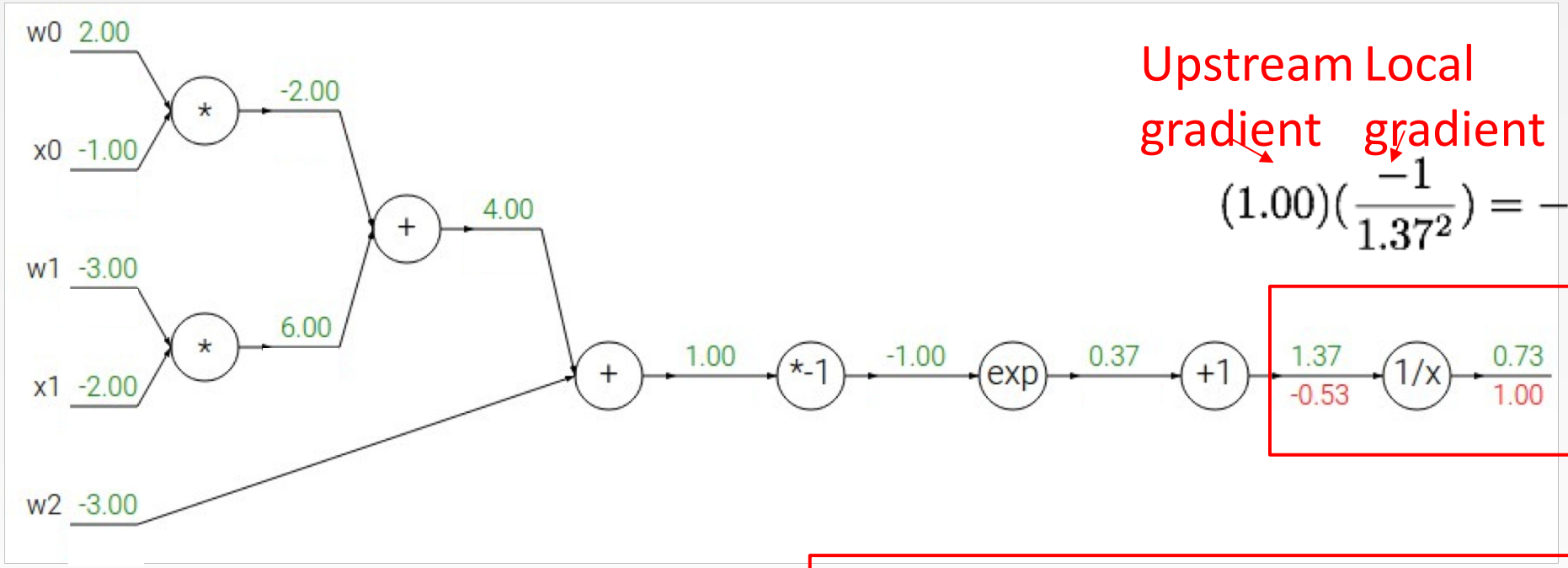
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



Upstream gradient × Local gradient

$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

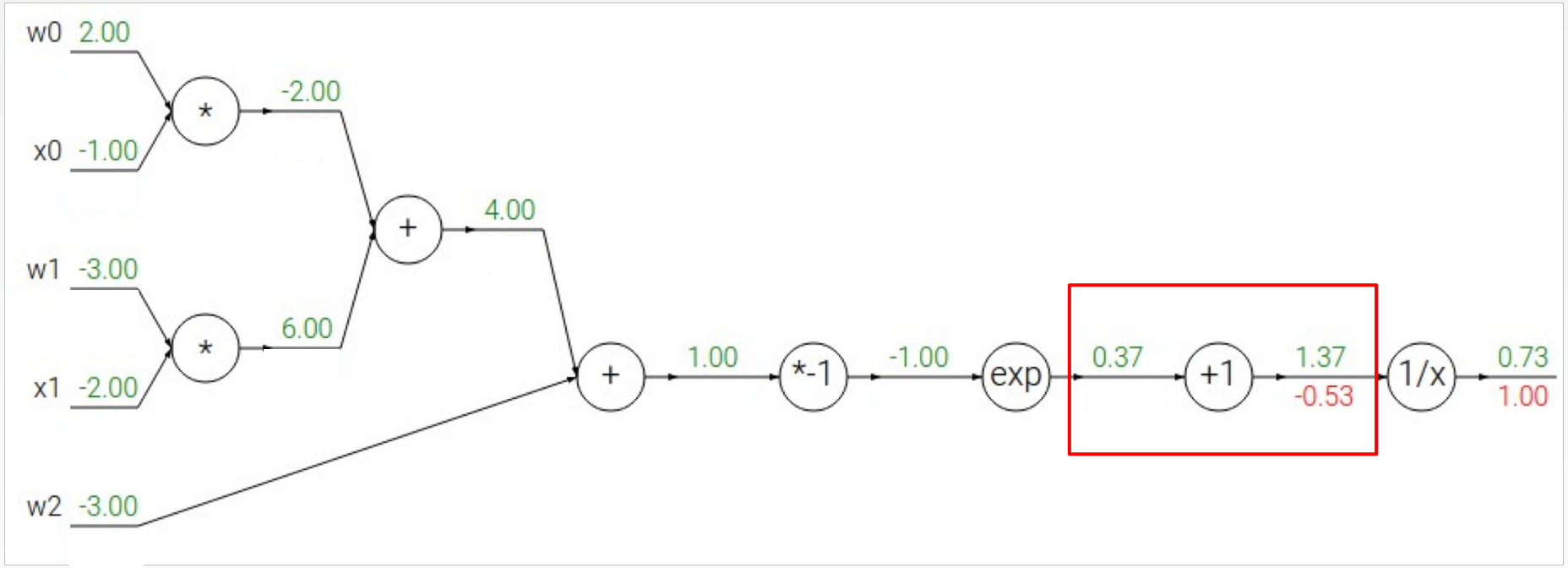
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

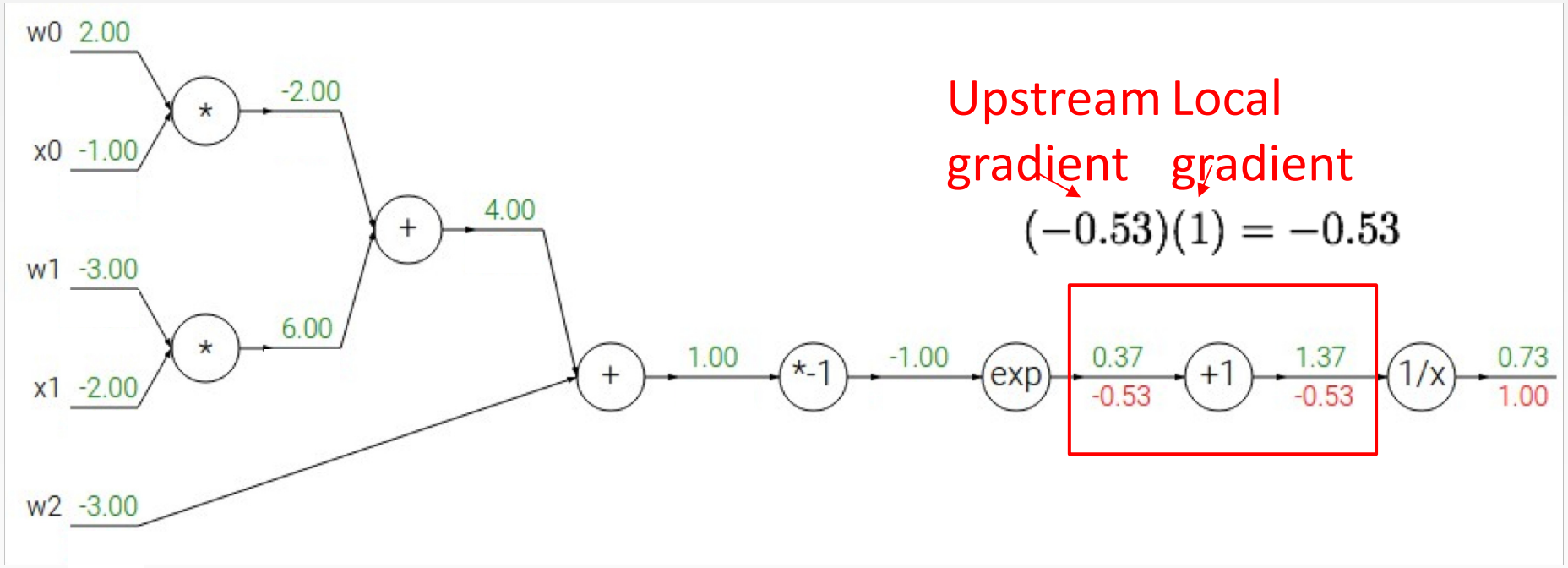
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

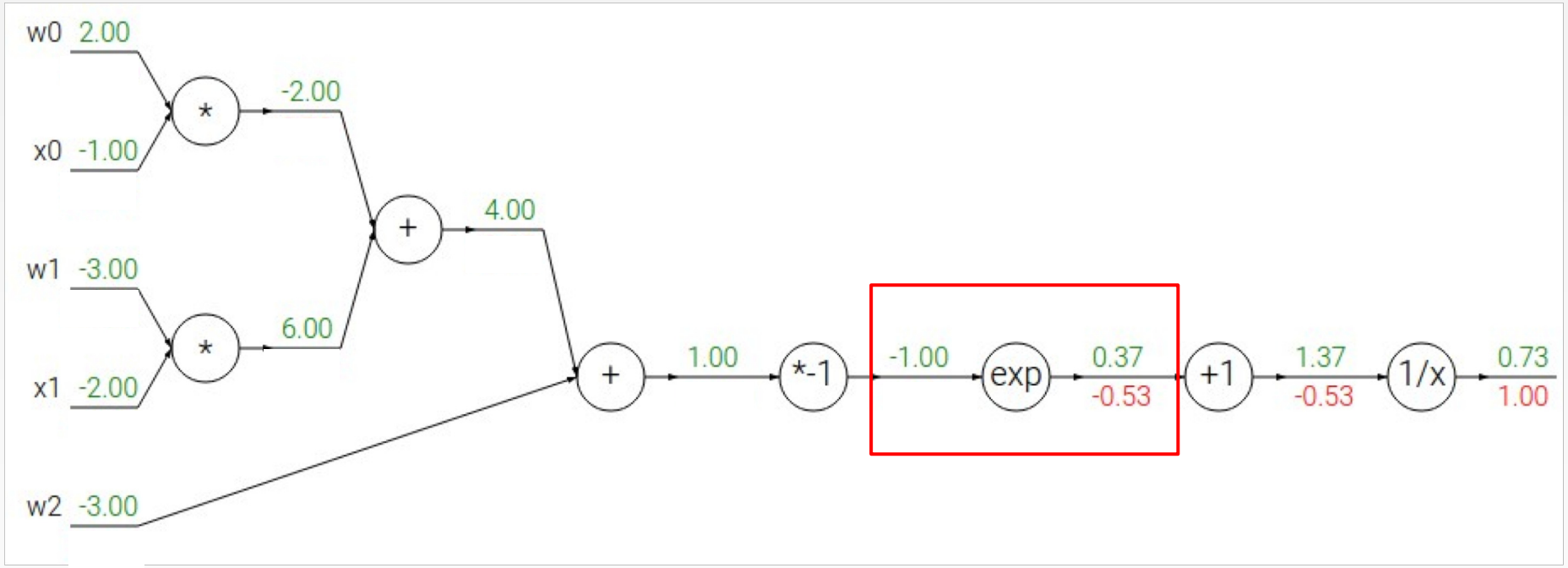
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

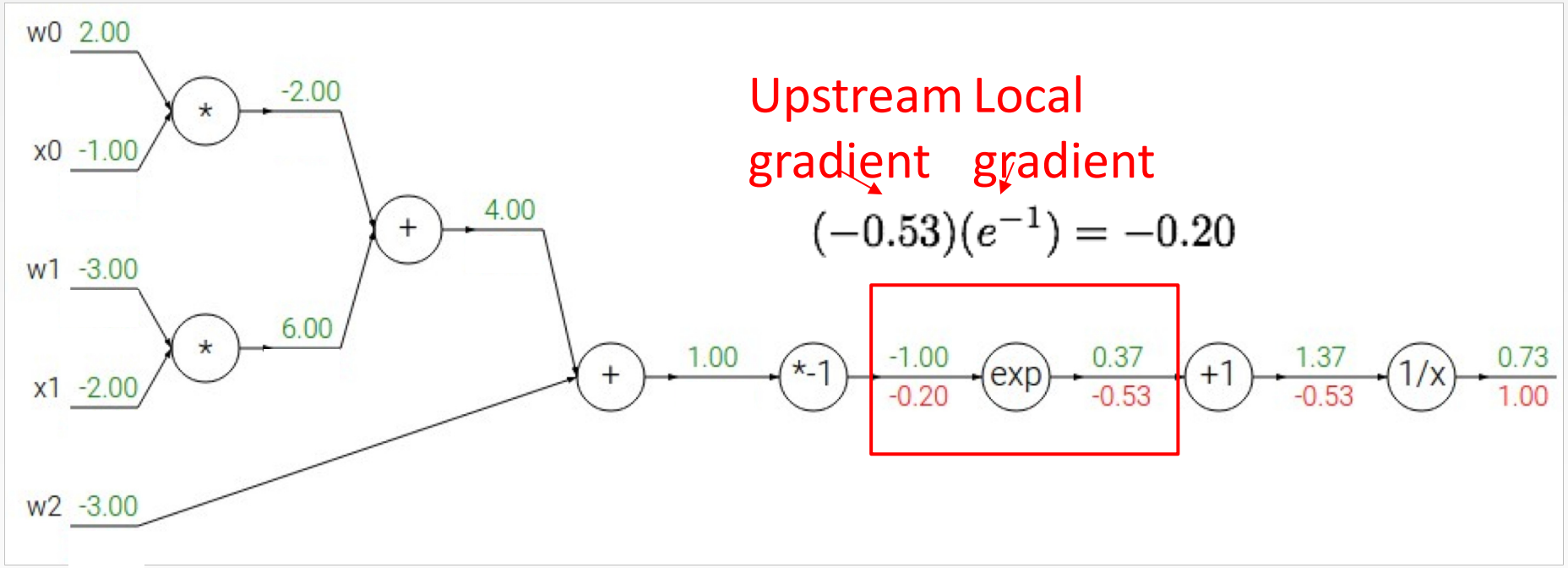
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

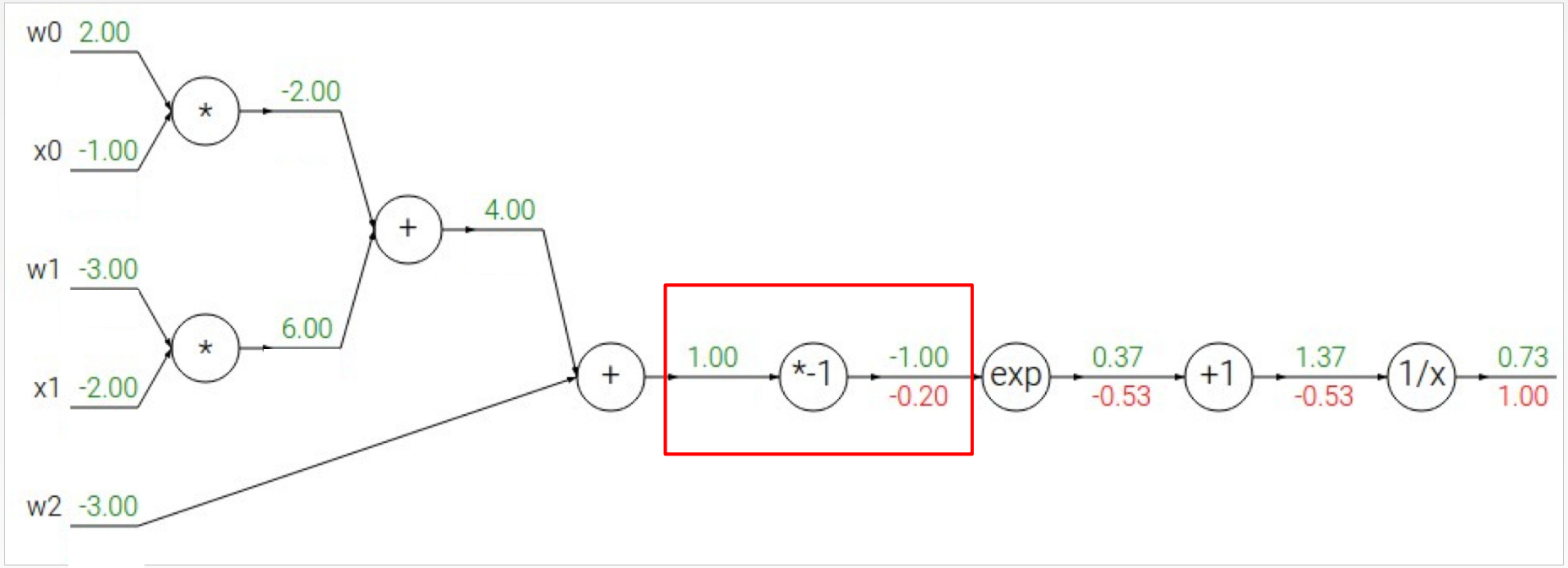
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

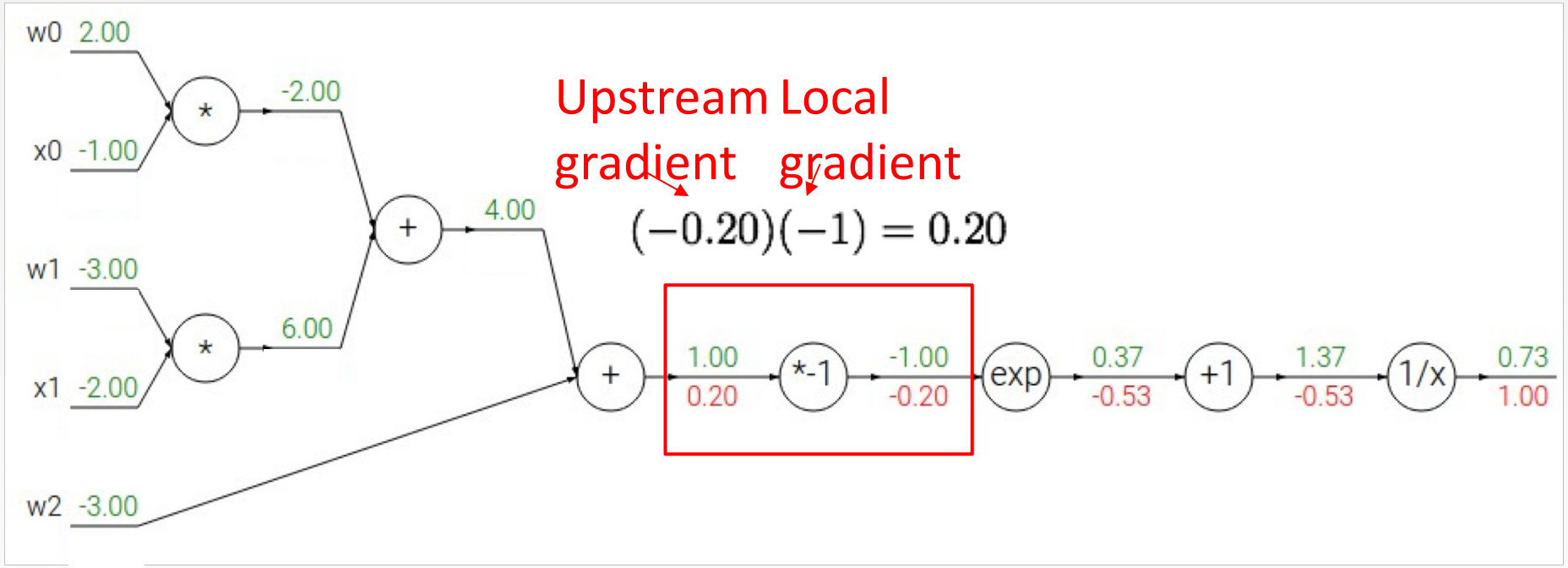
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

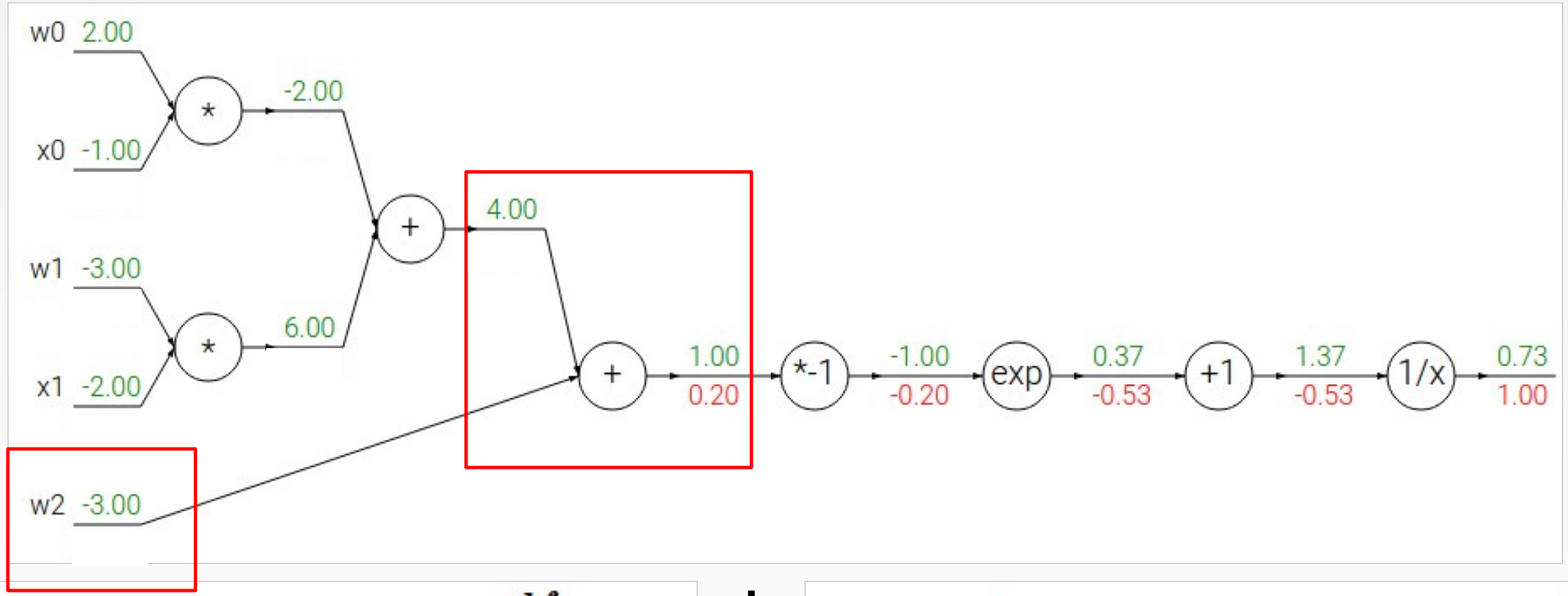
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

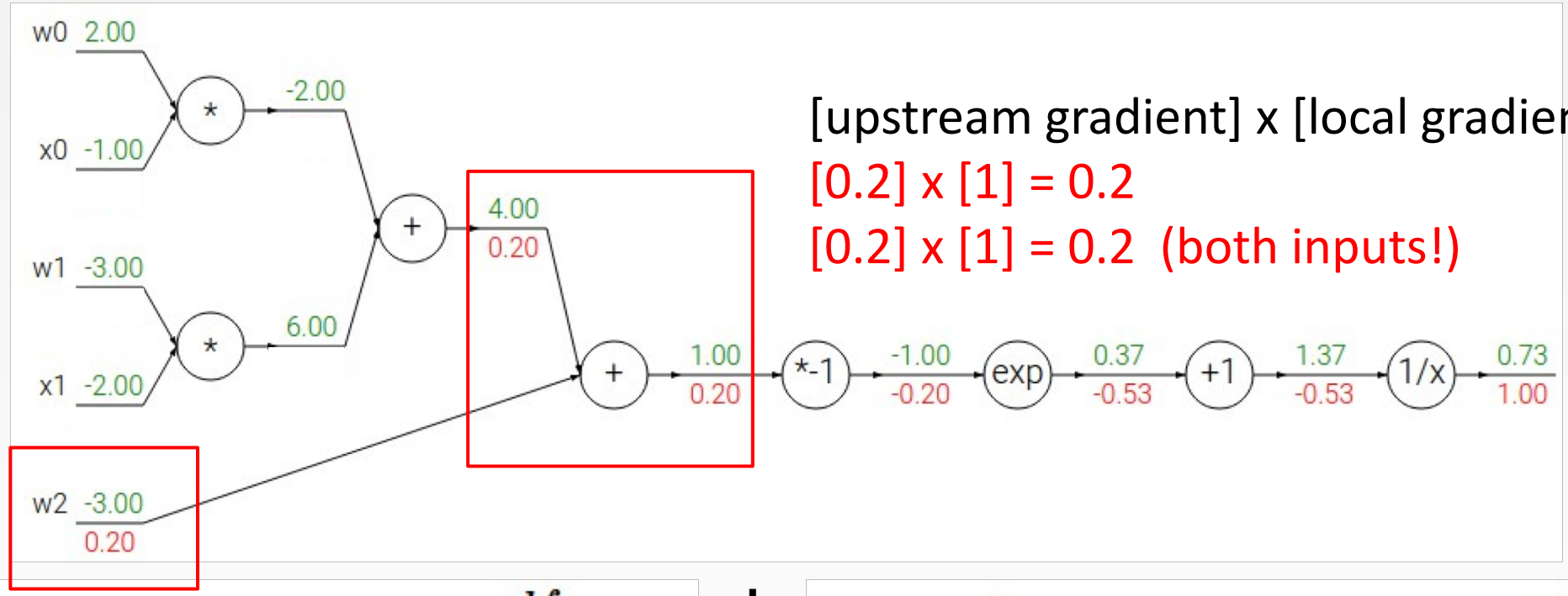
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

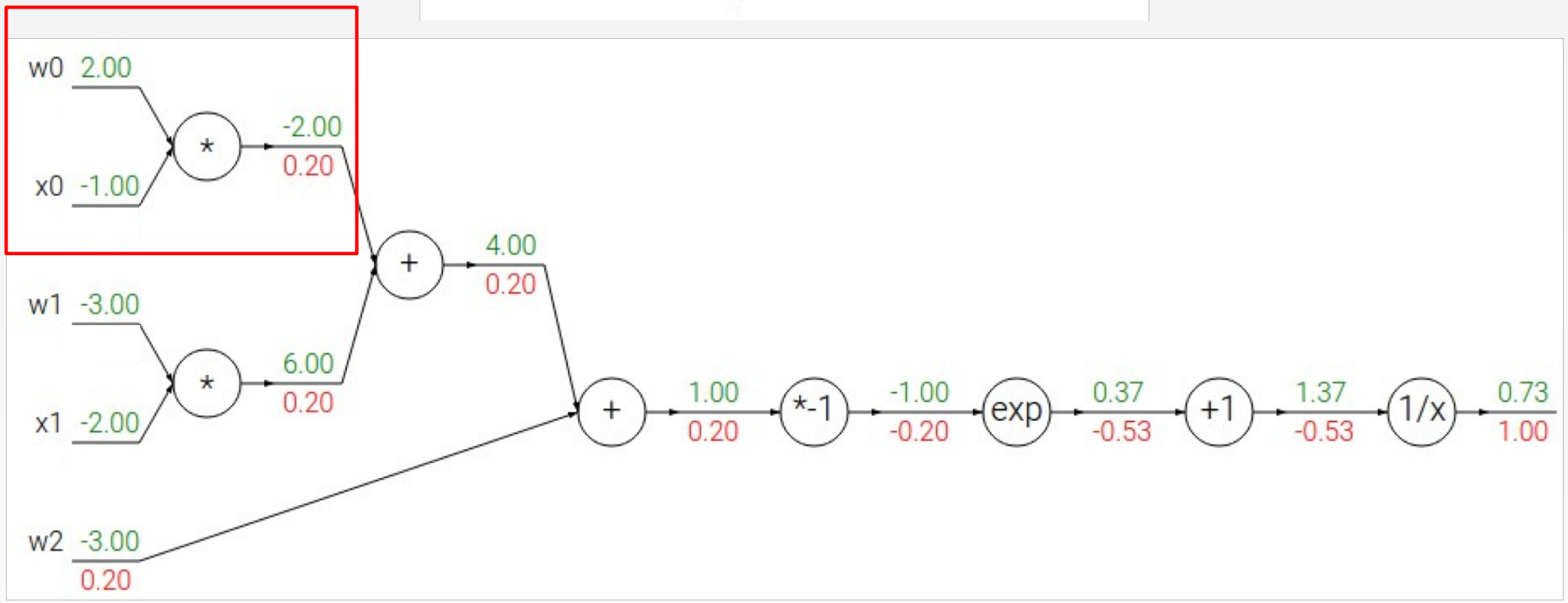
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

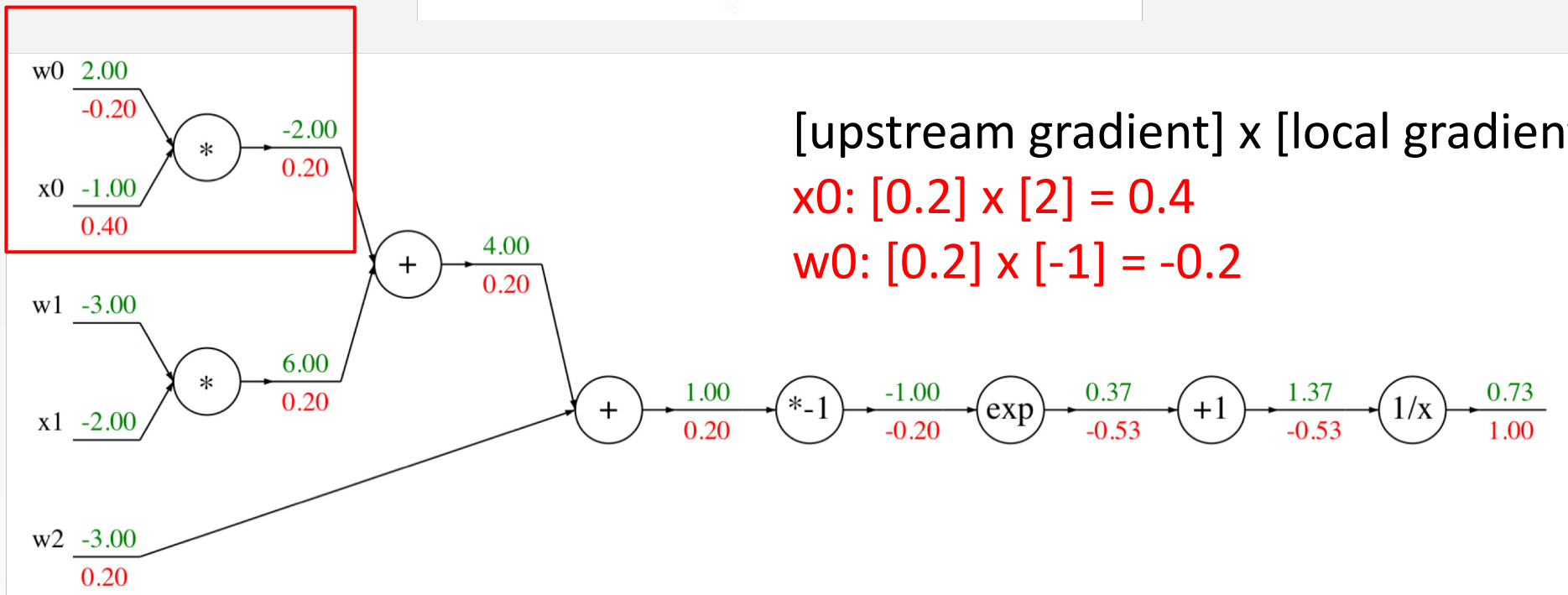
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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



[upstream gradient] x [local gradient]

x0: [0.2] x [2] = 0.4

w0: [0.2] x [-1] = -0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

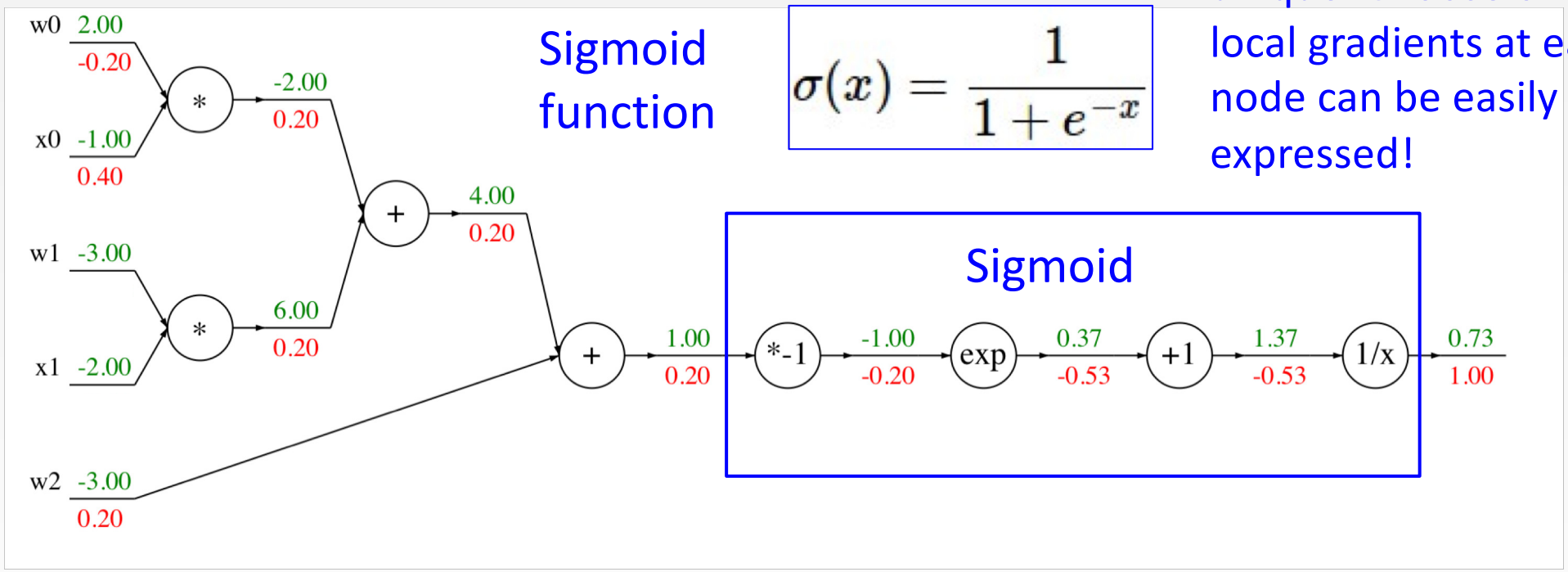
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

ANOTHER EXAMPLE:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



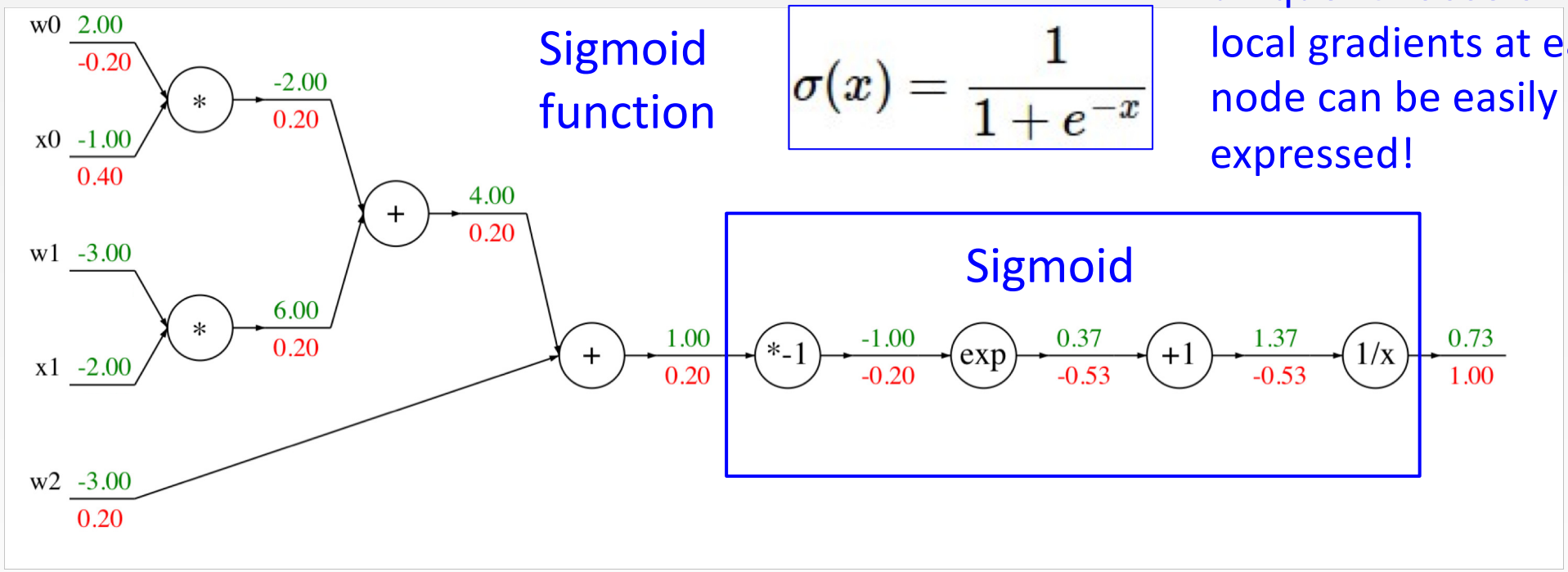
Sigmoid function

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

ANOTHER EXAMPLE:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

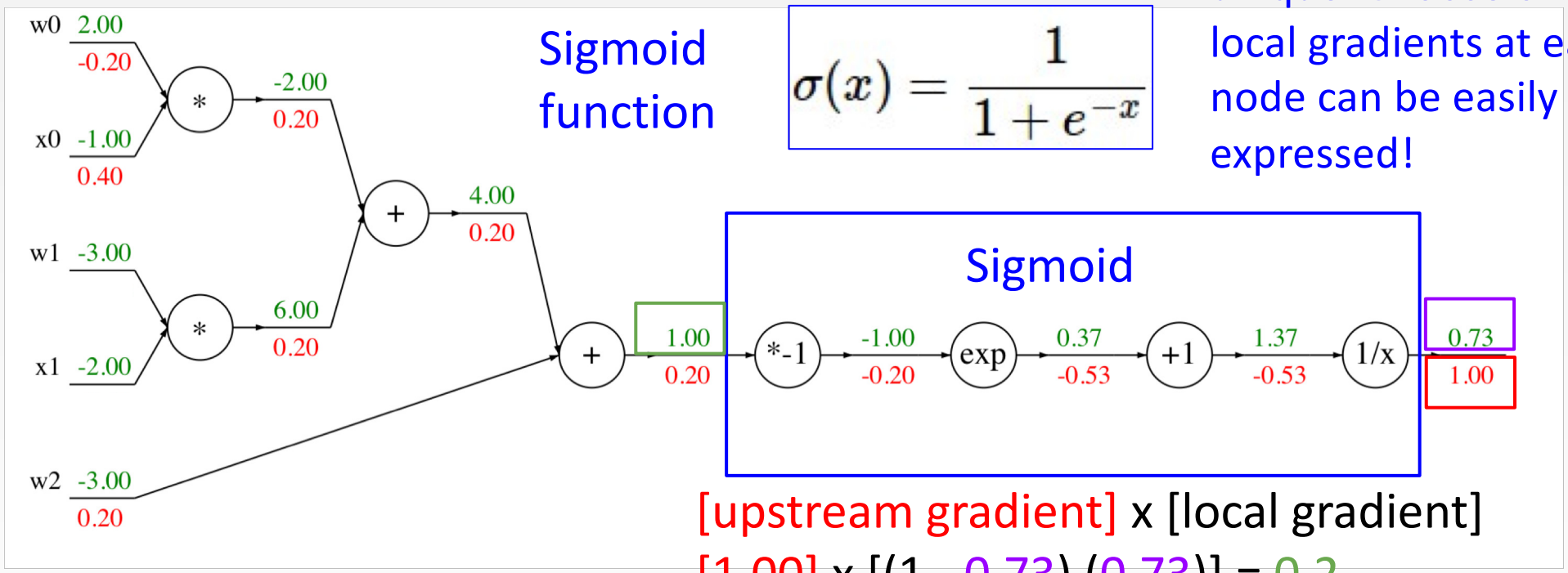
Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

ANOTHER EXAMPLE:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

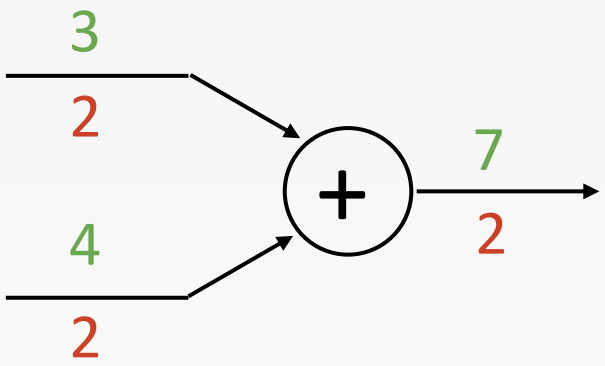
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

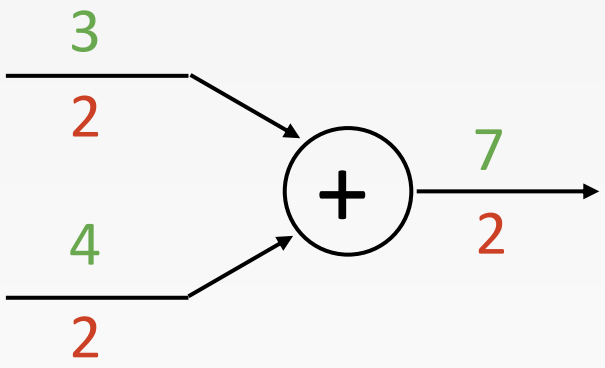
PATTERNS IN GRADIENT FLOW

add gate: gradient distributor

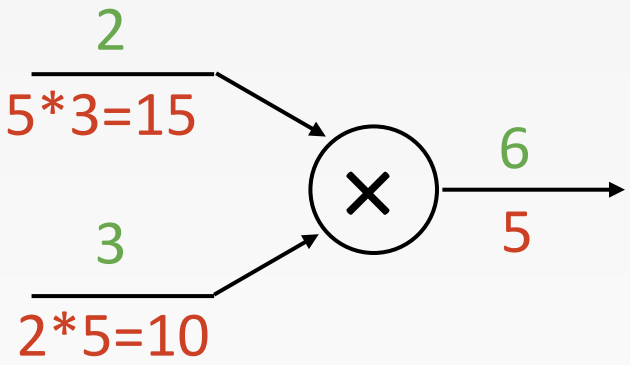


PATTERNS IN GRADIENT FLOW

add gate: gradient distributor

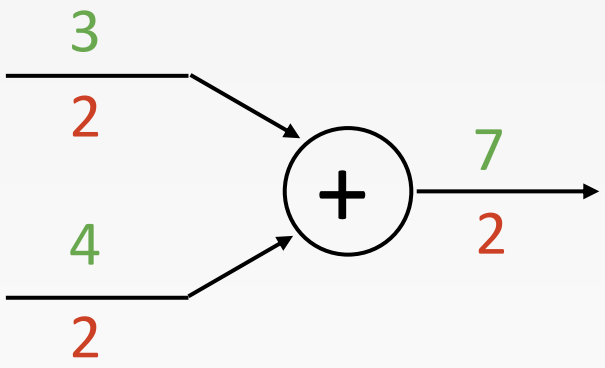


mul gate: “swap multiplier”

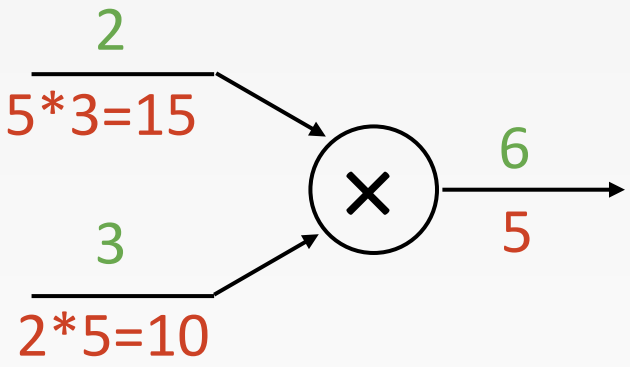


PATTERNS IN GRADIENT FLOW

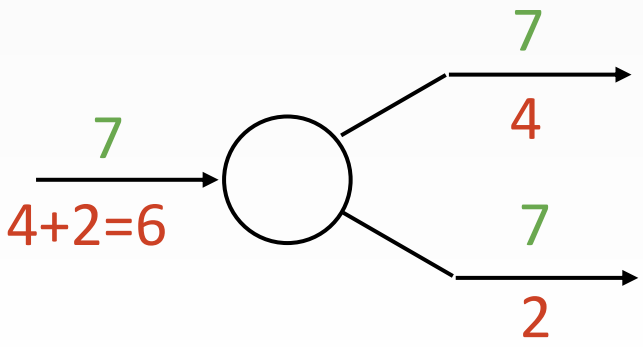
add gate: gradient distributor



mul gate: "swap multiplier"

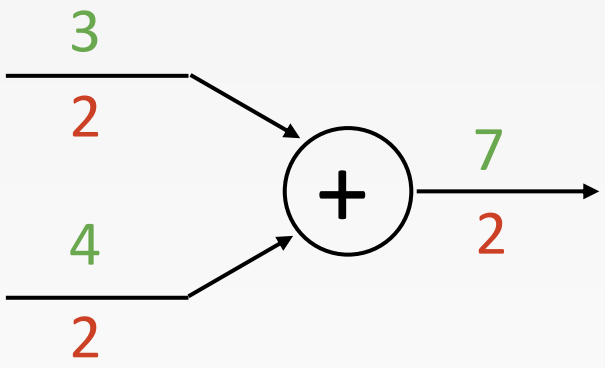


copy gate: gradient adder

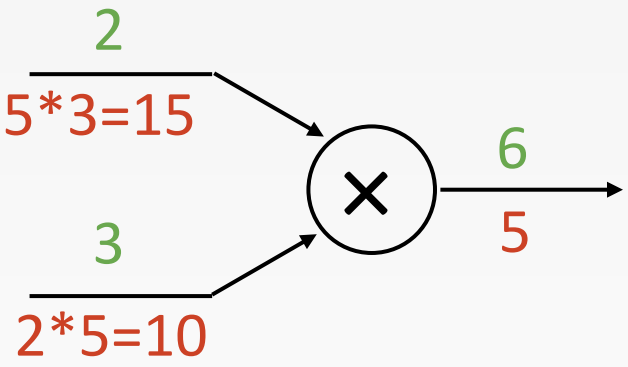


PATTERNS IN GRADIENT FLOW

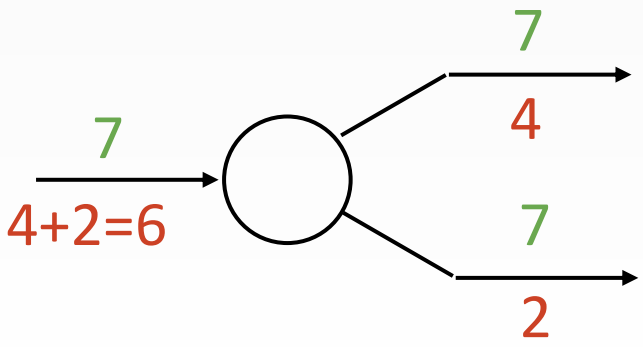
add gate: gradient distributor



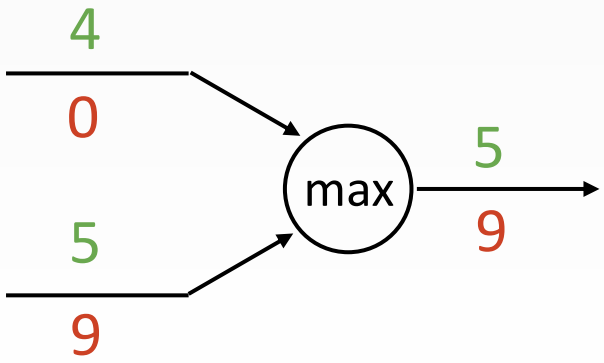
mul gate: "swap multiplier"



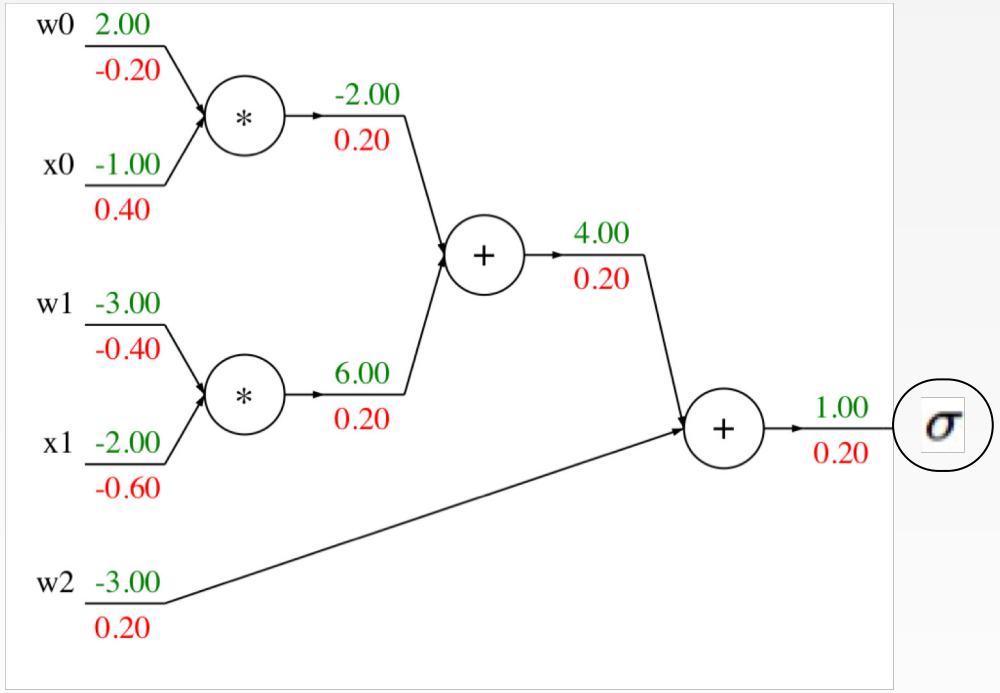
copy gate: gradient adder



max gate: gradient router



BACKPROP IMPLEMENTATION: "FLAT" CODE

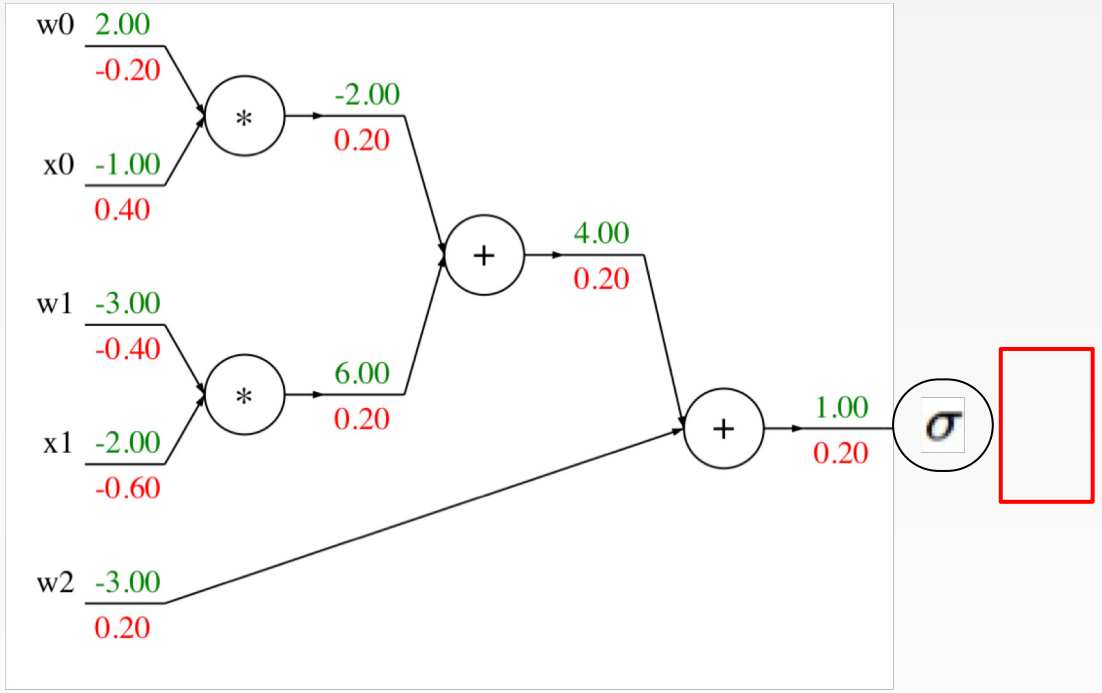


Forward pass:
Compute output

Backward pass:
Compute grads

```
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" CODE

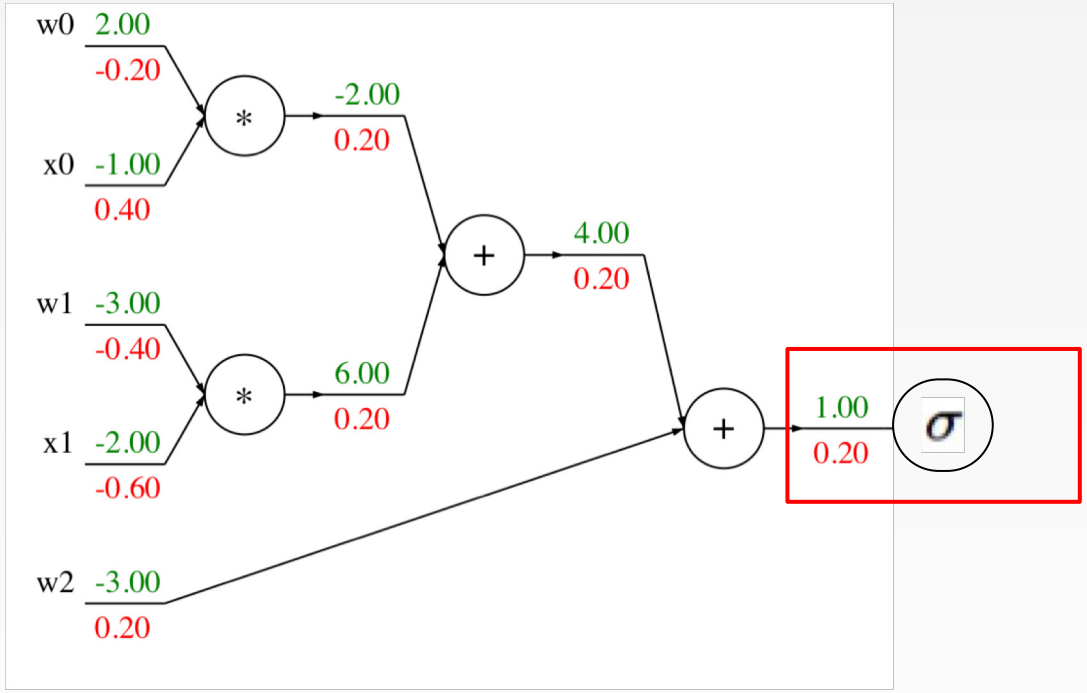


Forward pass:
Compute output

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" CODE

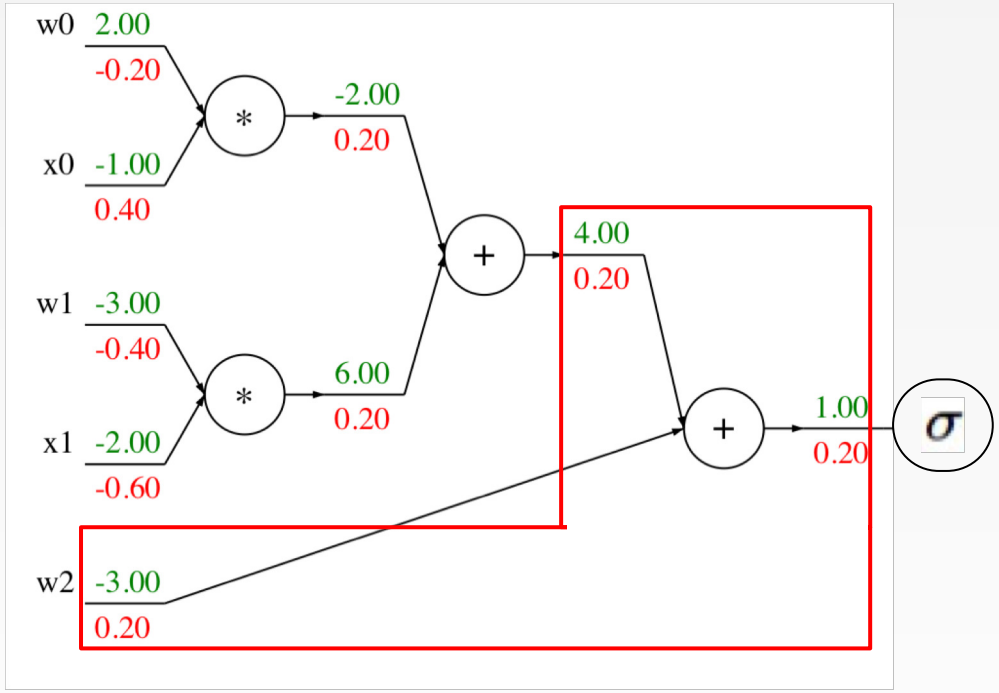


Forward pass:
Compute output

Sigmoid

```
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" CODE

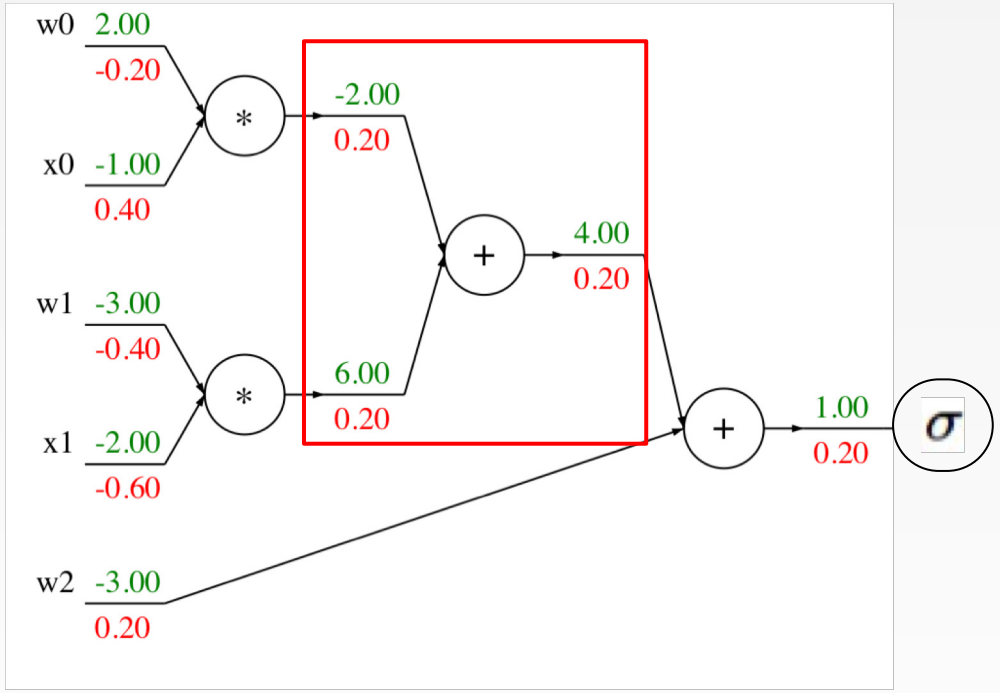


Forward pass:
Compute output

Add gate

```
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" CODE

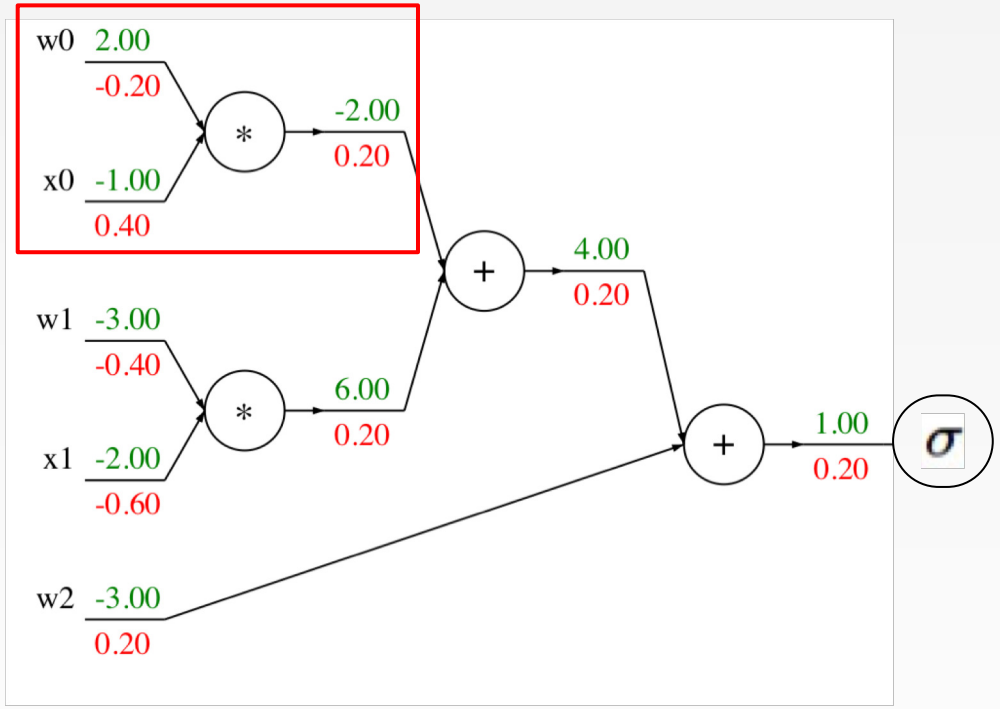


Forward pass:
Compute output

Add gate

```
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" CODE

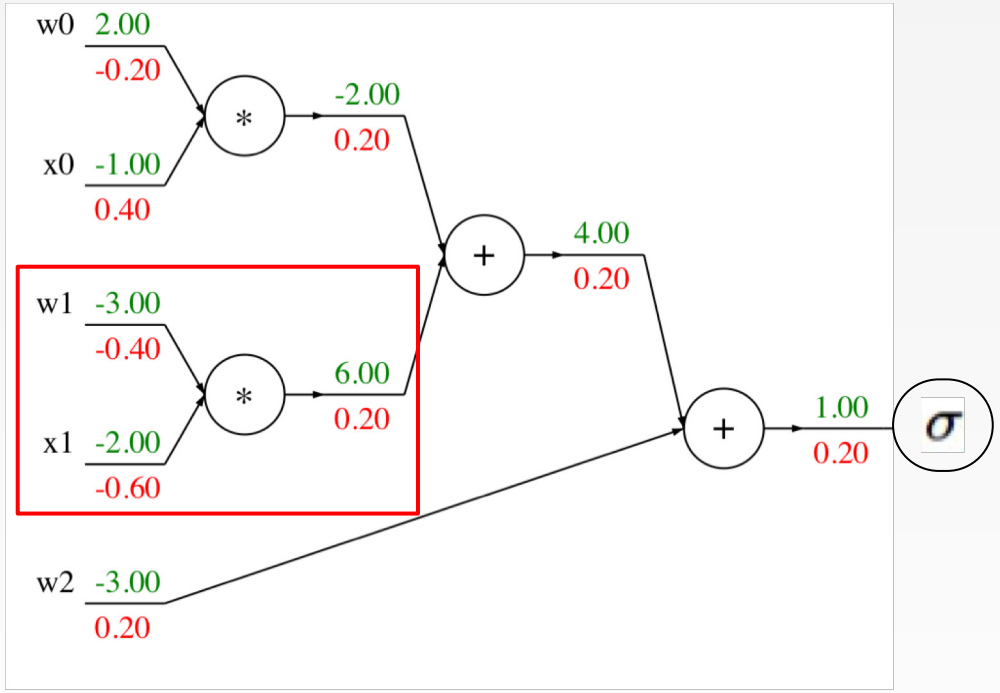


Forward pass:
Compute output

```
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 gate

BACKPROP IMPLEMENTATION: "FLAT" 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
```

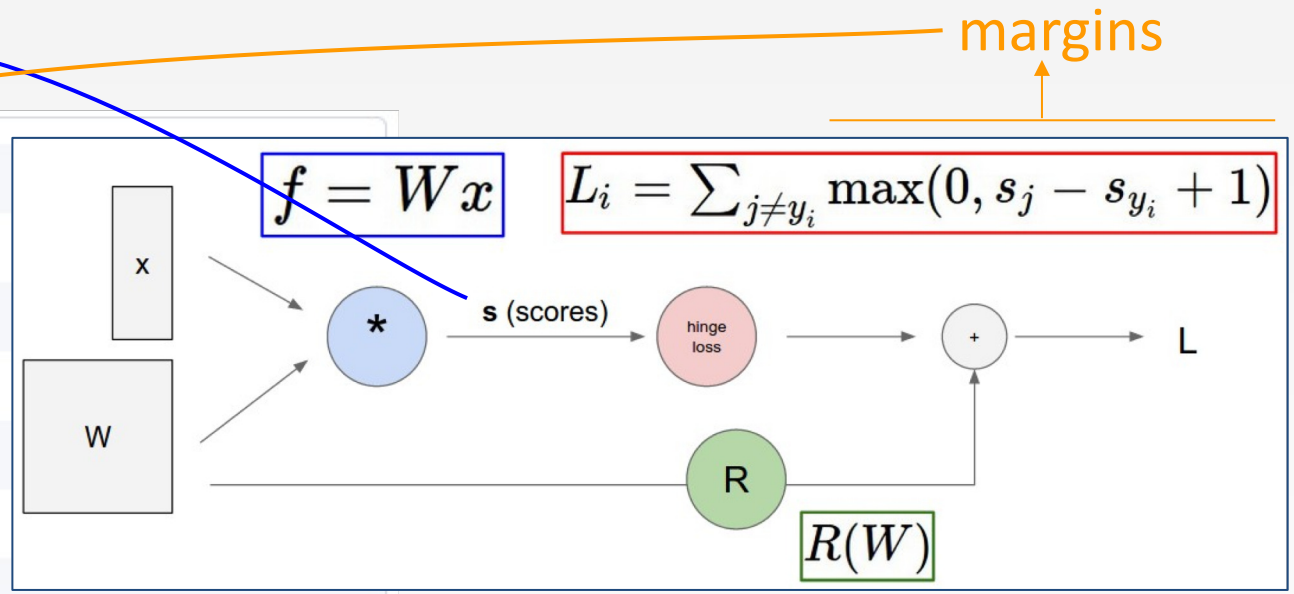
Multiply gate

"FLAT" BACKPROP: DO THIS FOR ASSIGNMENT 1!

Stage your forward/backward computation!

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 = #...
```

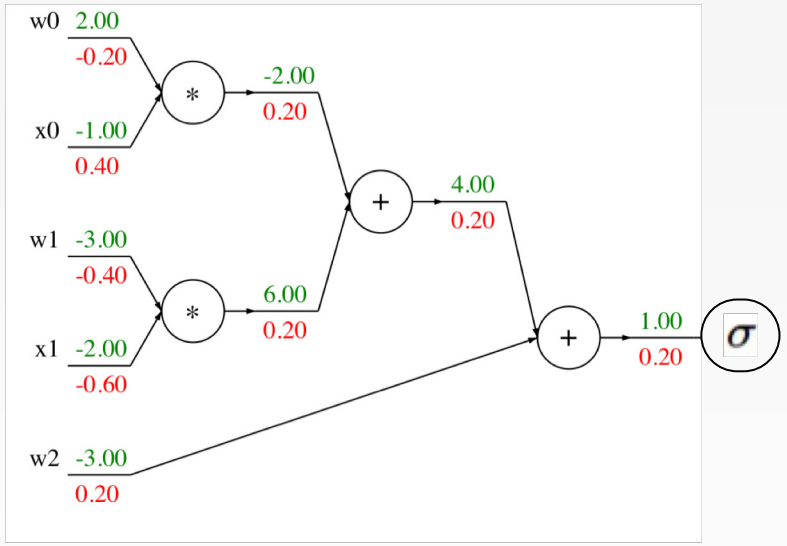


"FLAT" BACKPROP: DO THIS FOR ASSIGNMENT 1!

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: MODULARIZED API

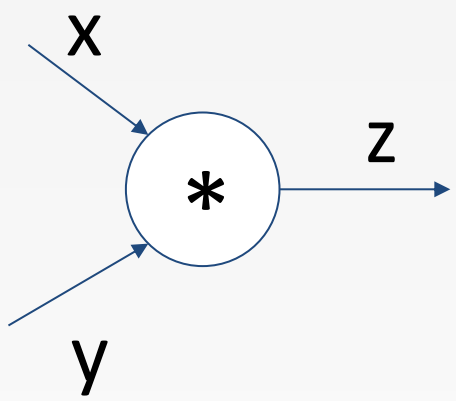


Graph (or Net) object *(rough pseudo code)*

```
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
```

MODULARIZED IMPLEMENTATION: FORWARD / BACKWARD API

Gate / Node / Function object: Actual PyTorch code



(x,y,z are scalars)

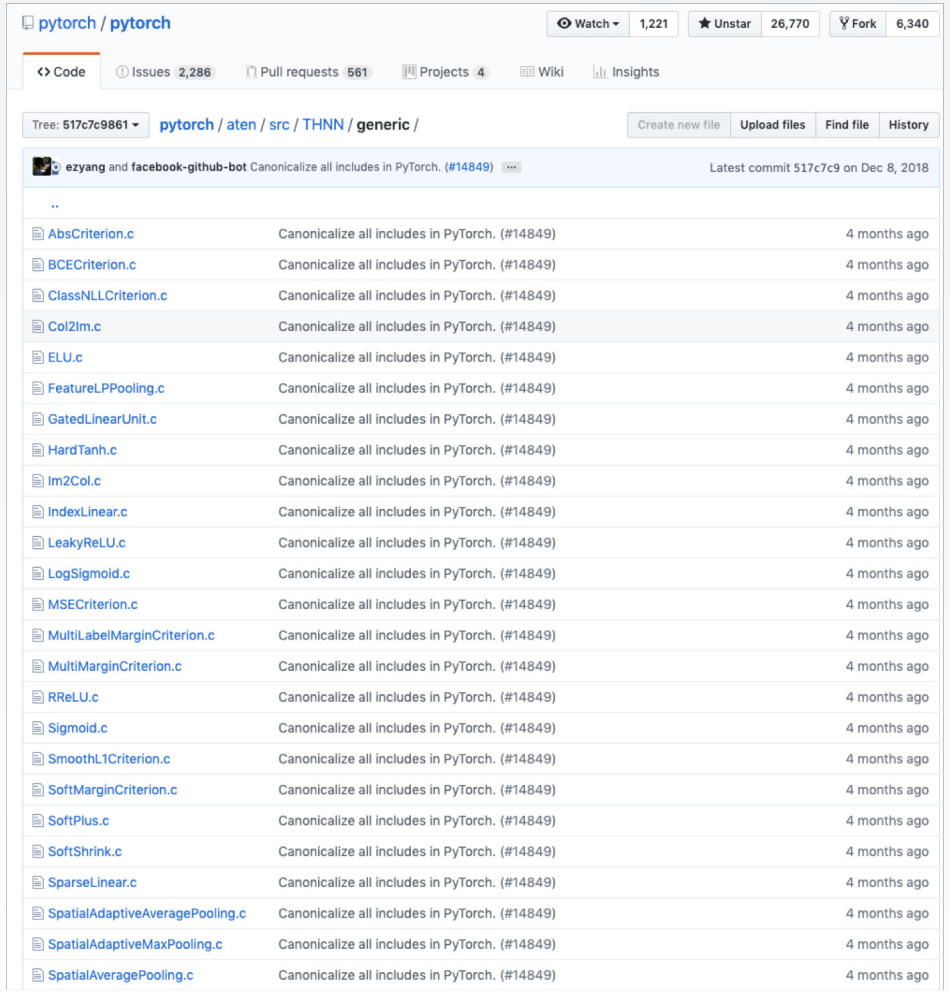
```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z    # dz/dx * dL/dz
    grad_y = x * grad_z    # dz/dy * dL/dz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

Need to stash some values for use in backward

Upstrea m gradient

Multiply upstream and local gradients

EXAMPLE: PYTORCH OPERATORS



pytorch / pytorch

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

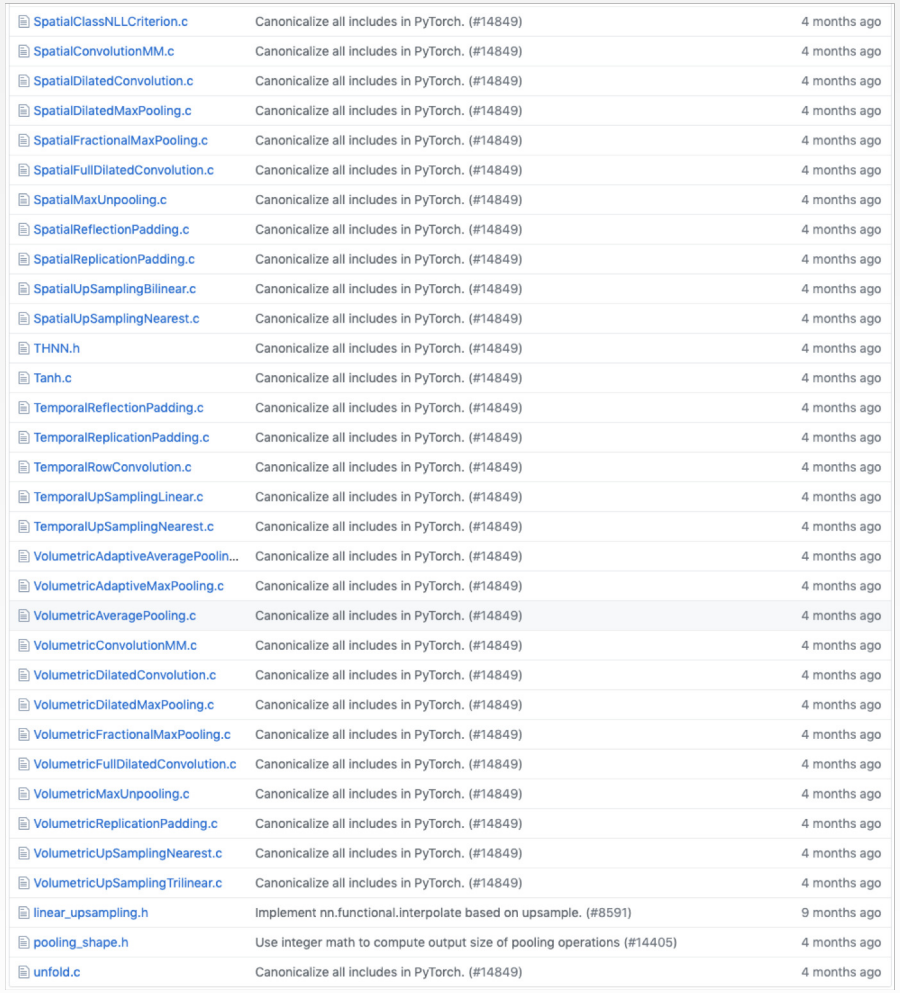
Code | Issues 2,286 | Pull requests 561 | Projects 4 | Wiki | Insights

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
```

Forward

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

PYTORCH SIGMOID LAYER

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

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

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

#endif
```

Forward

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

```
static void sigmoid_kernel(TensorIterator& iter) {
  AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
    unary_kernel_vec(
        iter,
        [=](scalar_t a) -> scalar_t { return (1 / (1 + std::exp((-a)))); },
        [=](Vec256<scalar_t> a) {
          a = Vec256<scalar_t>((scalar_t)(0)) - a;
          a = a.exp();
          a = Vec256<scalar_t>((scalar_t)(1)) + a;
          a = a.reciprocal();
          return a;
        });
  });
}
```

Forward actually defined elsewhere...

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

PYTORCH SIGMOID LAYER

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

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

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

#endif
```

Forward

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

```
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...

Backward

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

SO FAR: BACKPROP WITH SCALARS

WHAT ABOUT VECTOR-VALUED FUNCTIONS?

RECAP: VECTOR DERIVATIVES

Scalar to Scalar

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

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

Scalar to Scalar

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

Regular derivative:

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

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

Vector to Scalar

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

Derivative is **Gradient**:

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

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

RECAP: VECTOR DERIVATIVES

Scalar to Scalar

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

Regular derivative:

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

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

Vector to Scalar

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

Derivative is **Gradient**:

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

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

Vector to Vector

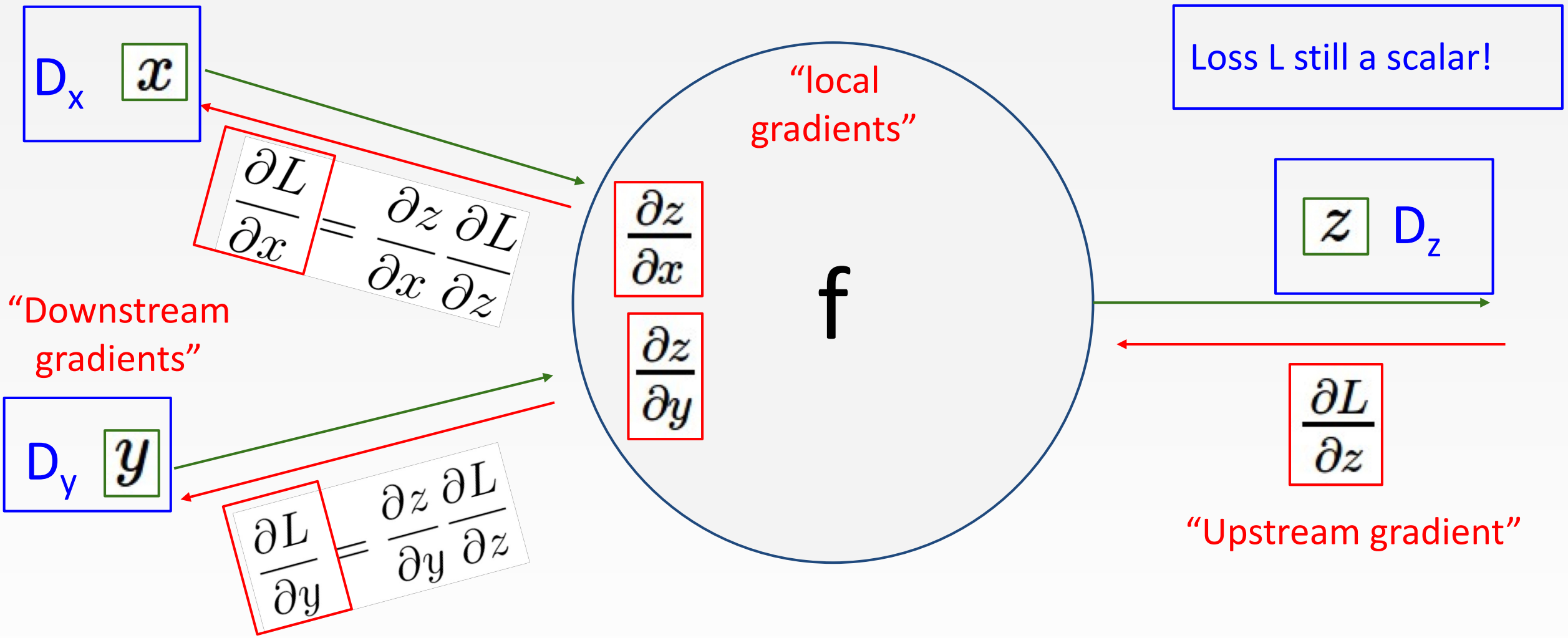
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

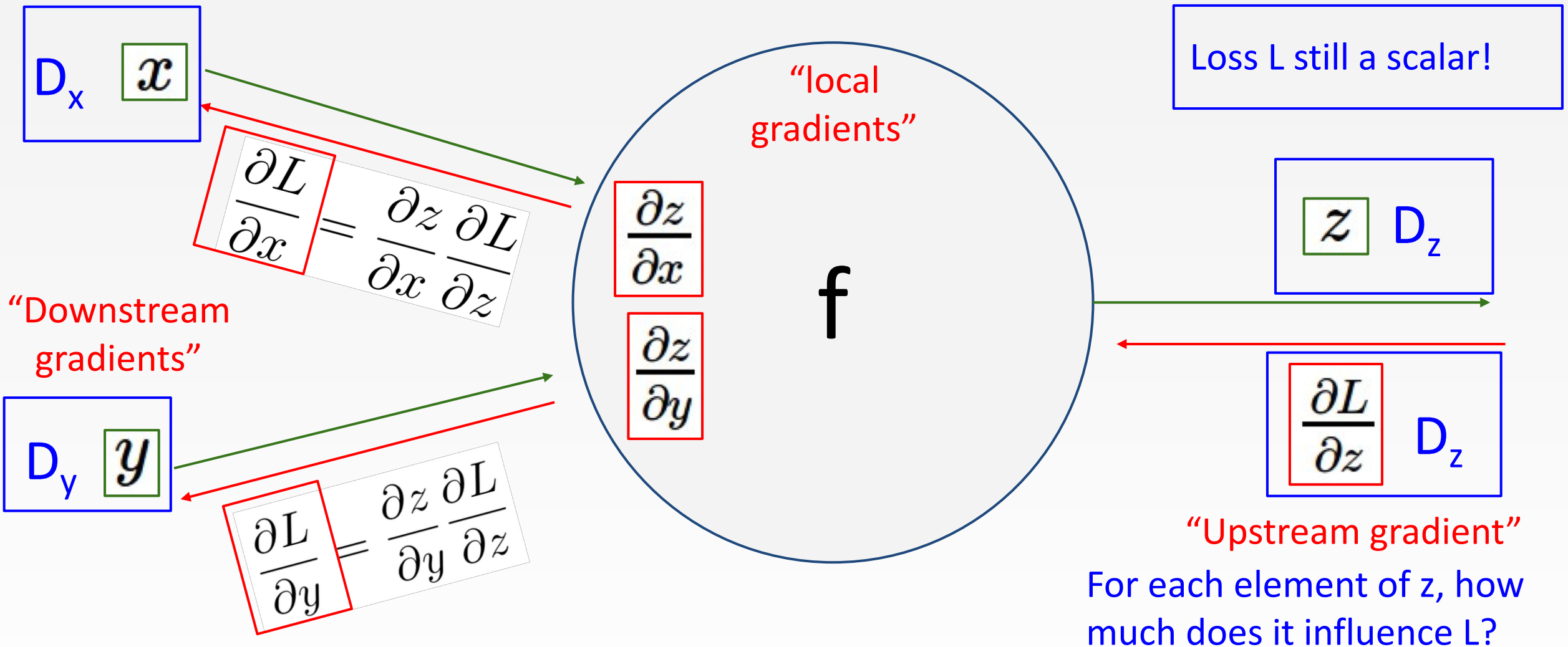
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \qquad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

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

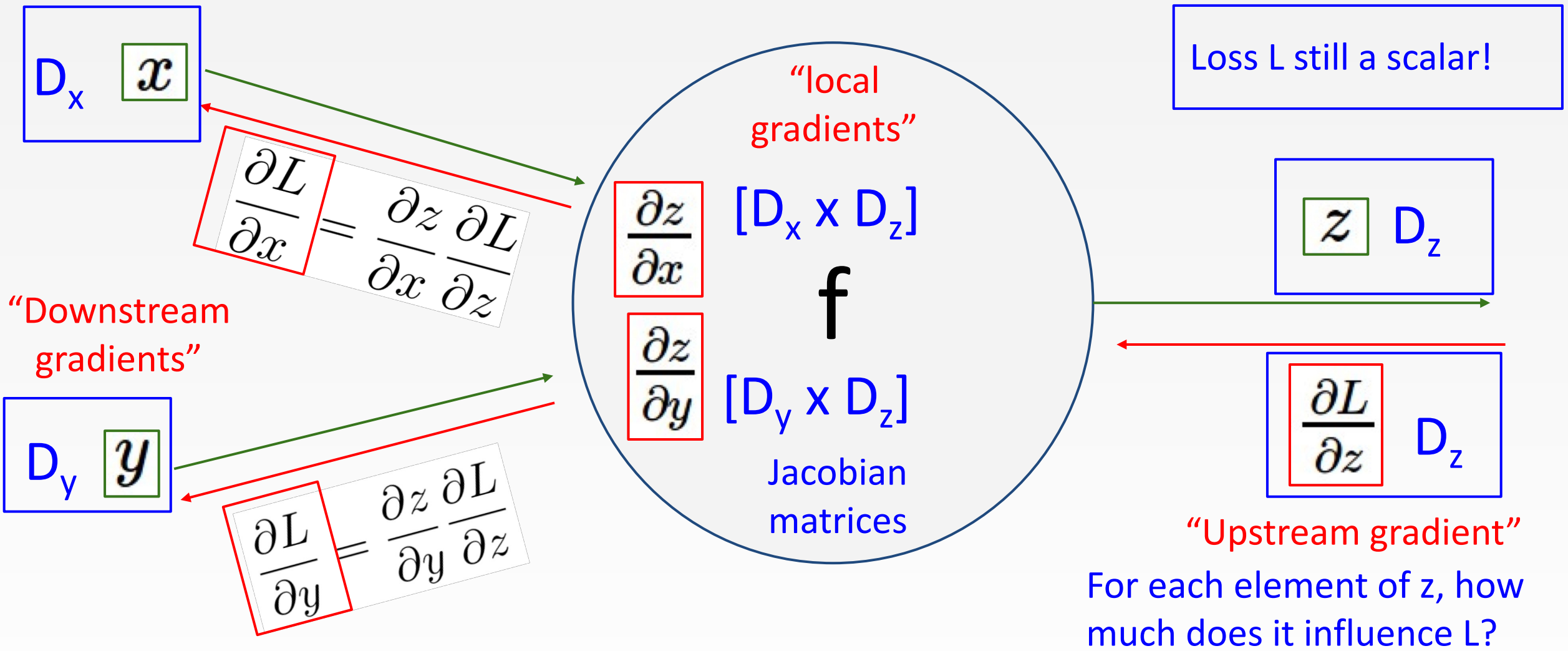
BACKPROP WITH VECTORS



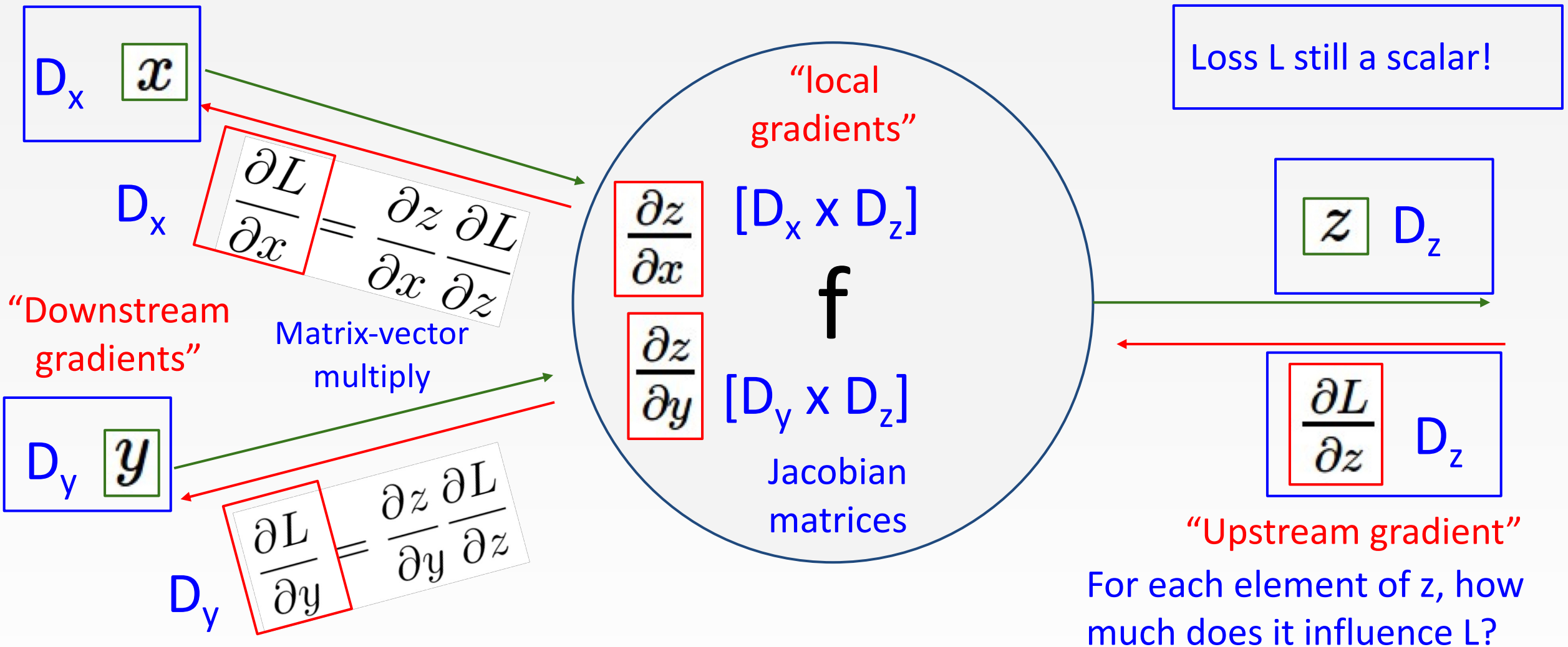
BACKPROP WITH VECTORS



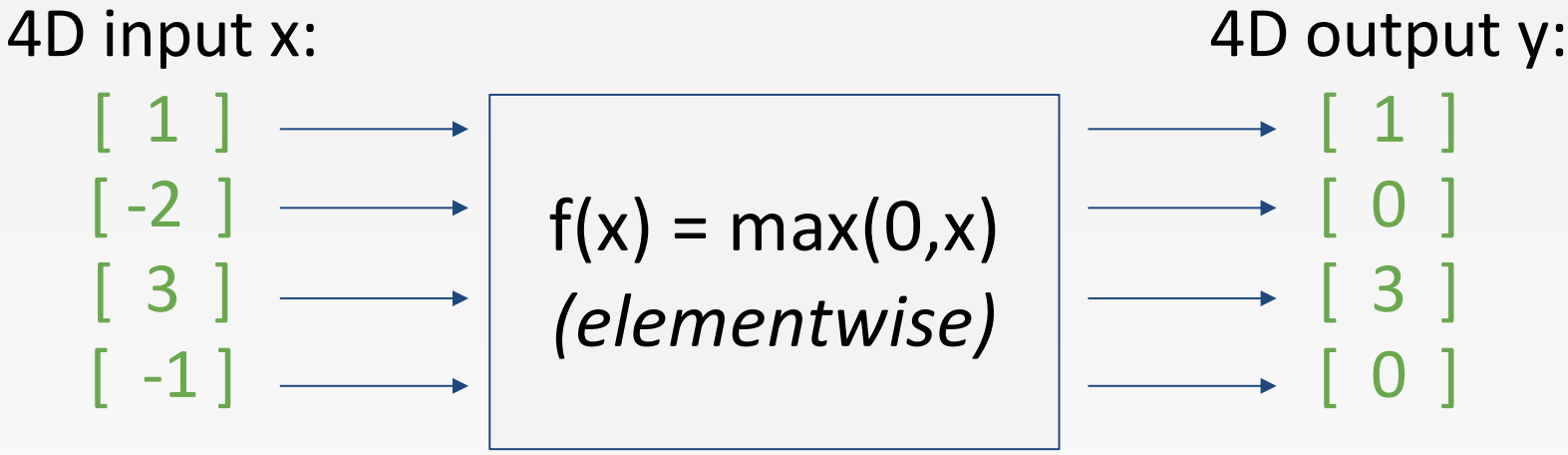
BACKPROP WITH VECTORS



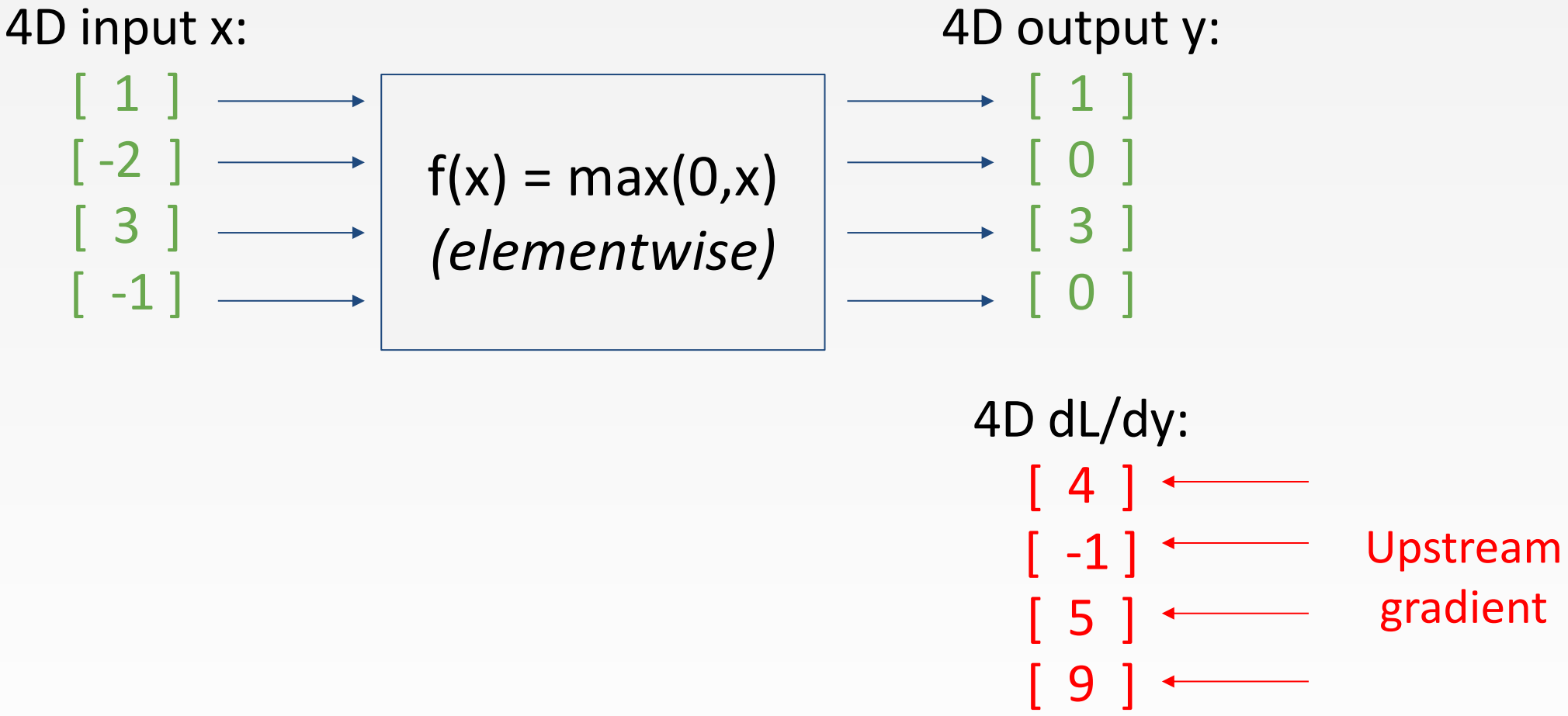
BACKPROP WITH VECTORS



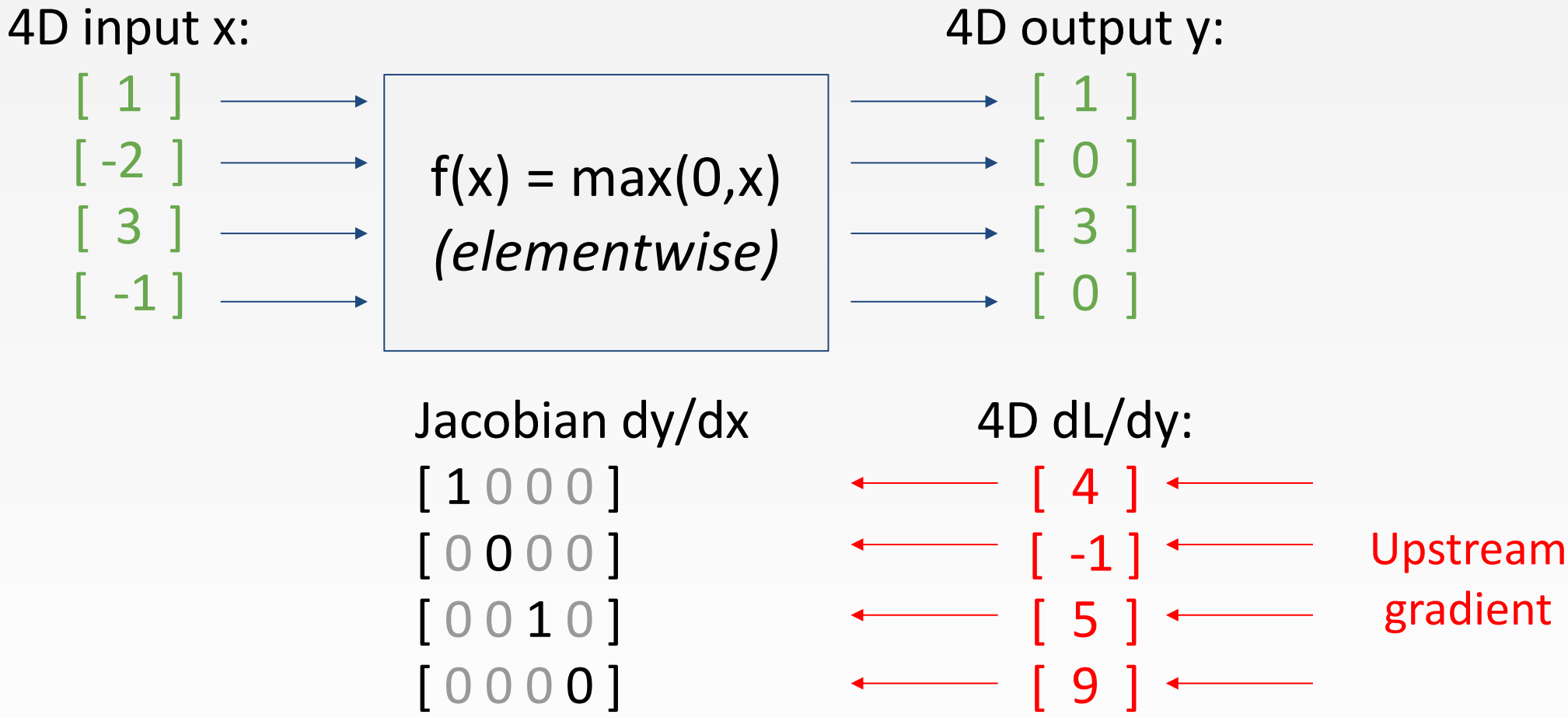
BACKPROP WITH VECTORS



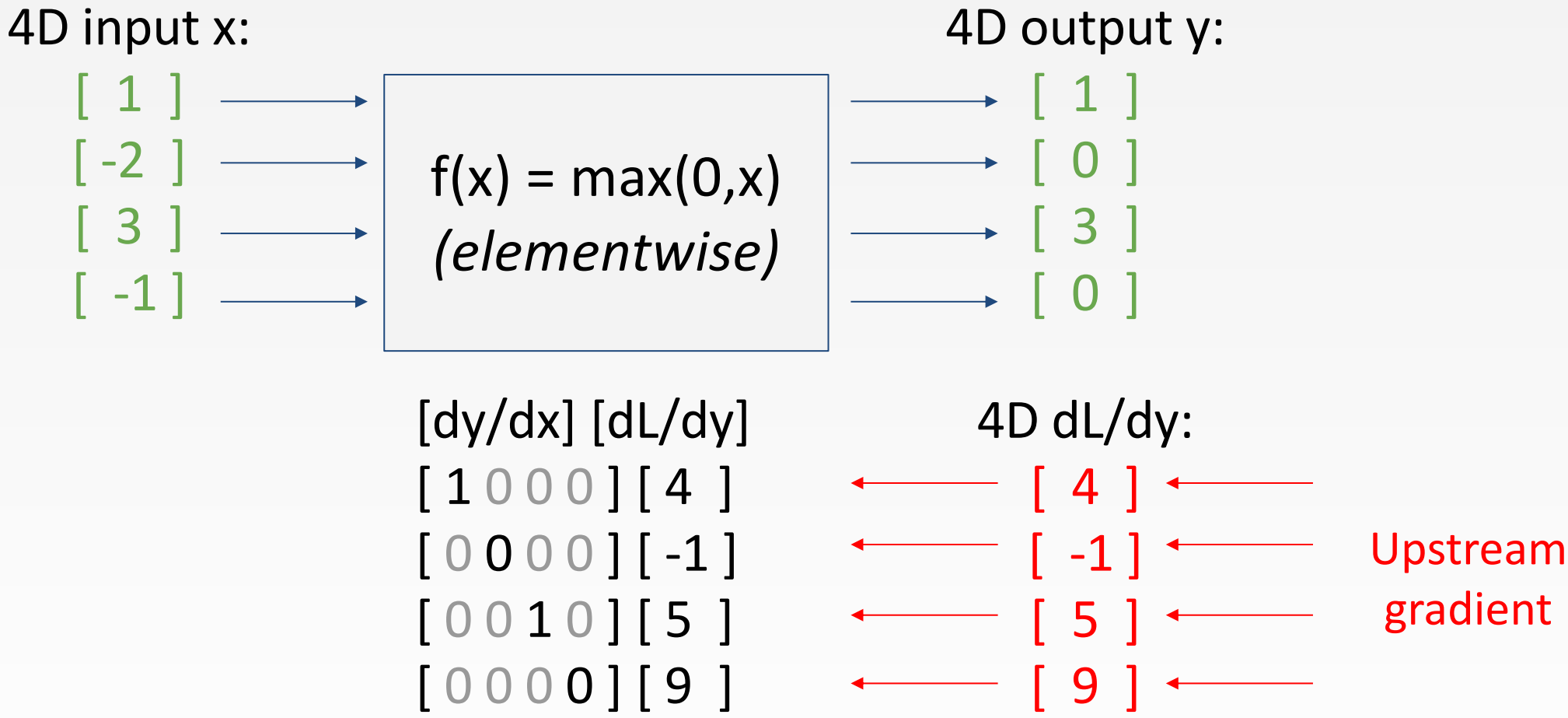
BACKPROP WITH VECTORS



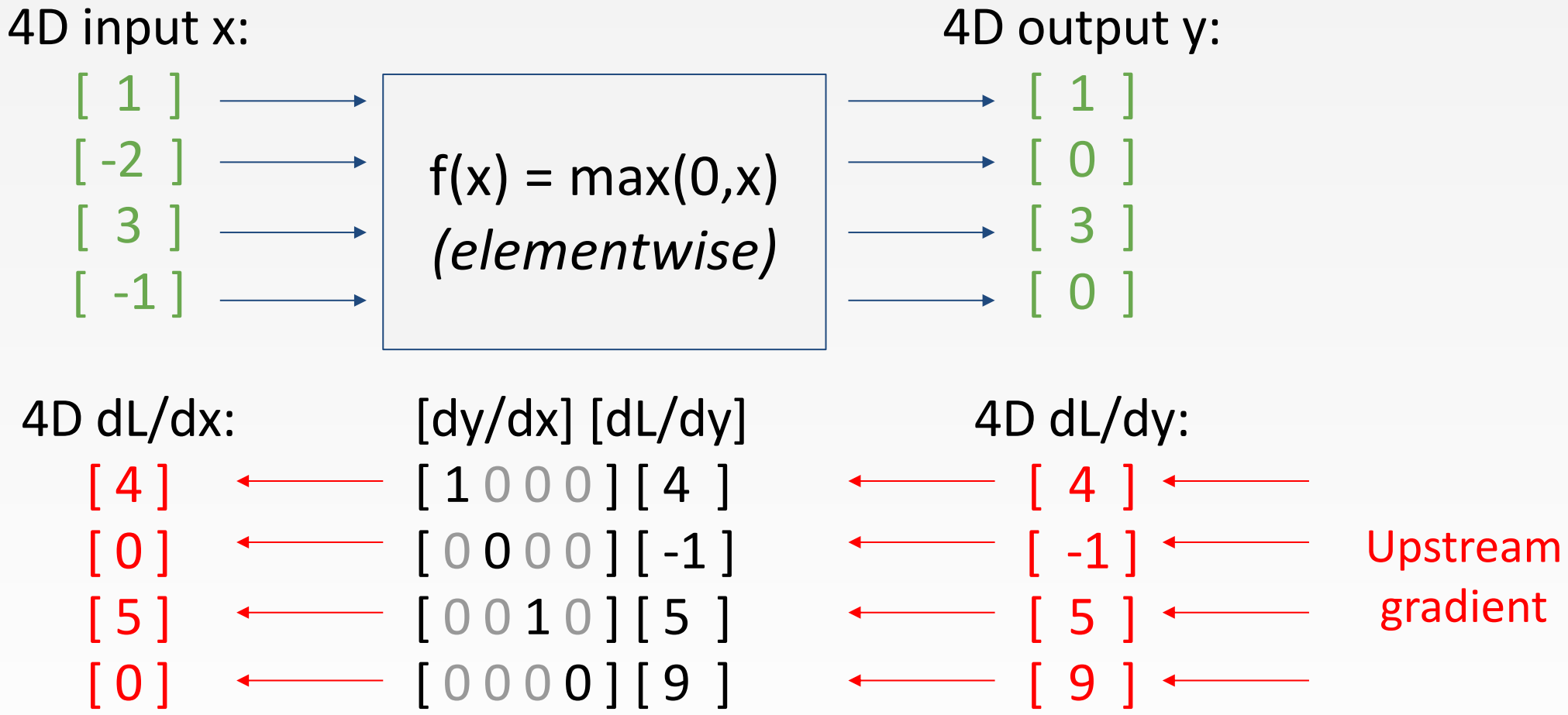
BACKPROP WITH VECTORS



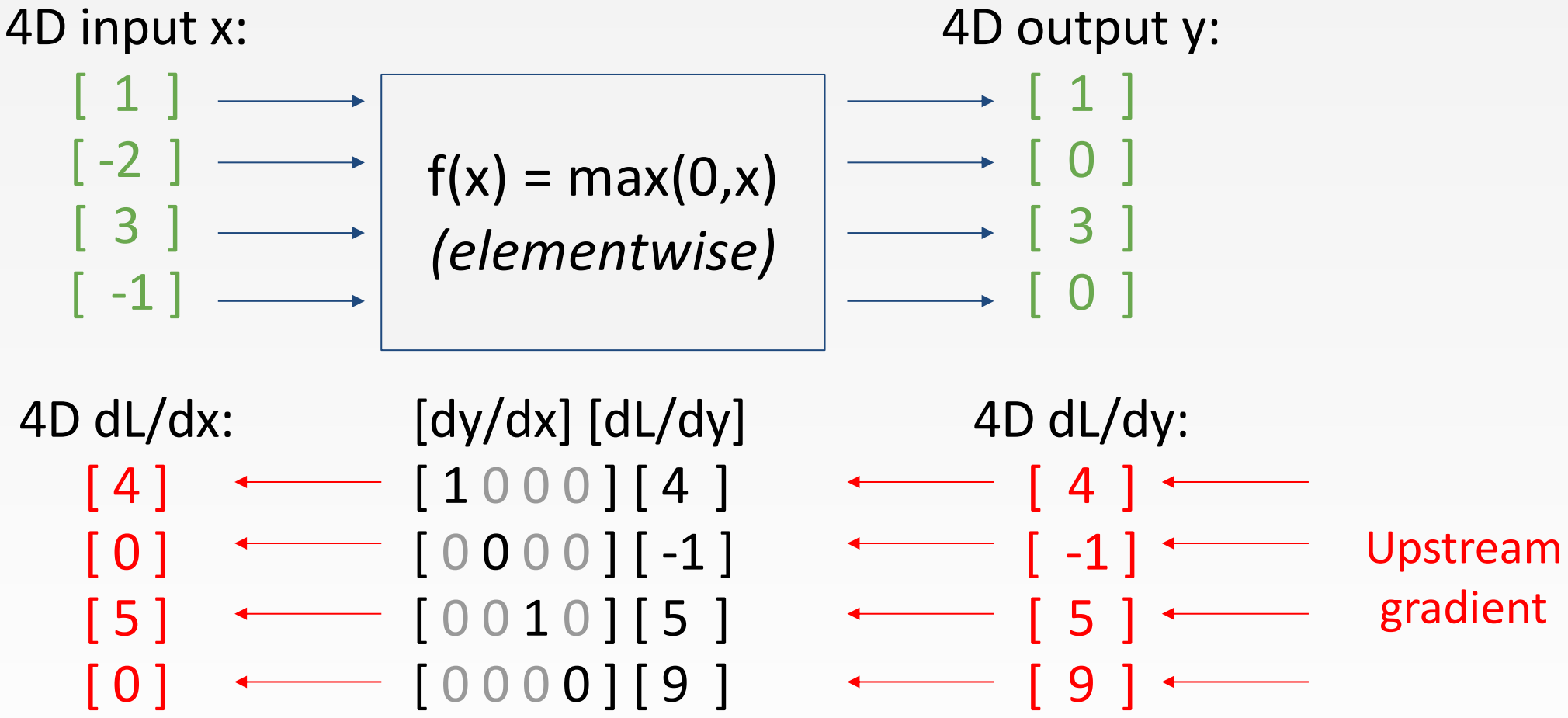
BACKPROP WITH VECTORS



BACKPROP WITH VECTORS

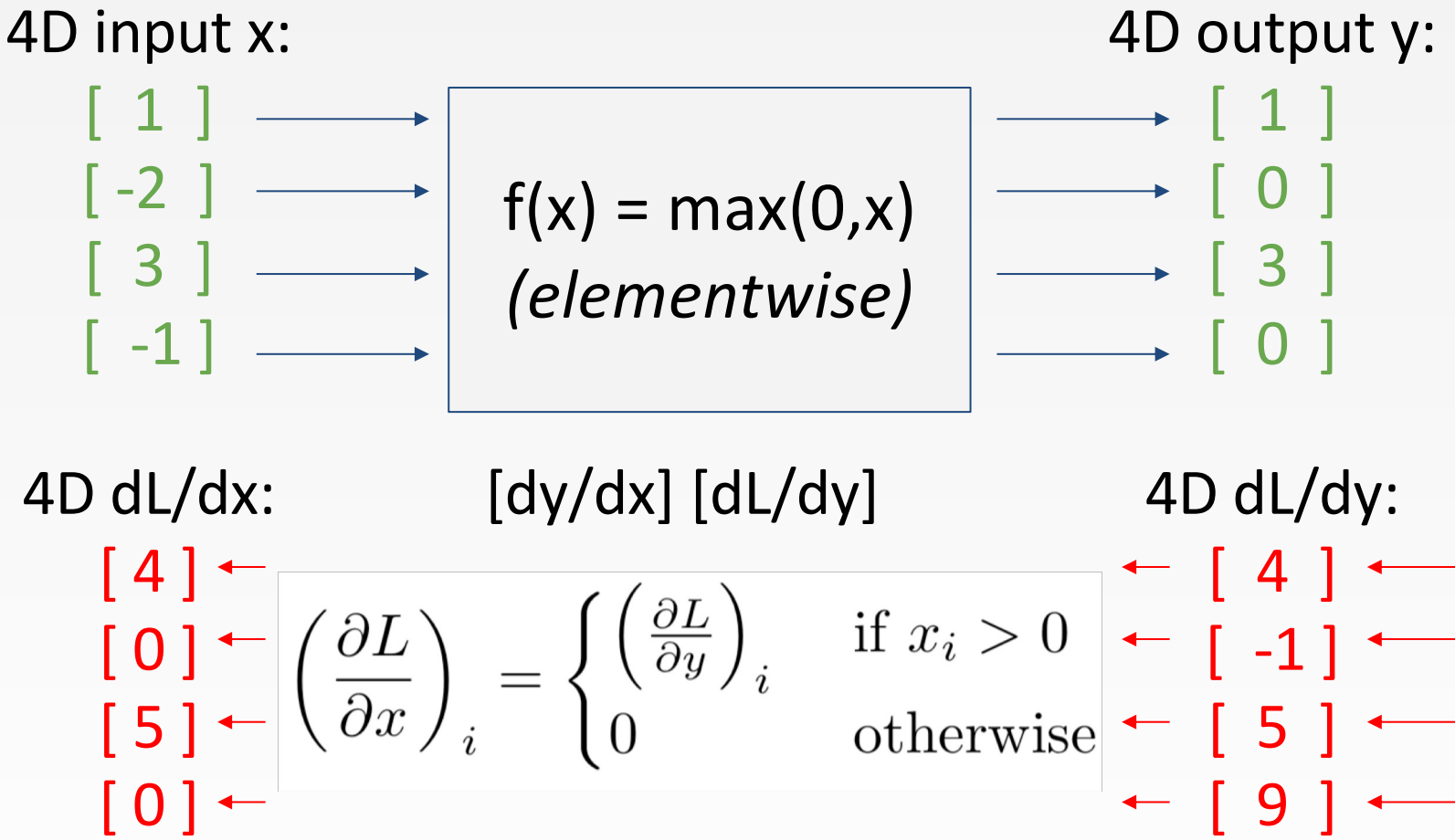


BACKPROP WITH VECTORS



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

BACKPROP WITH VECTORS

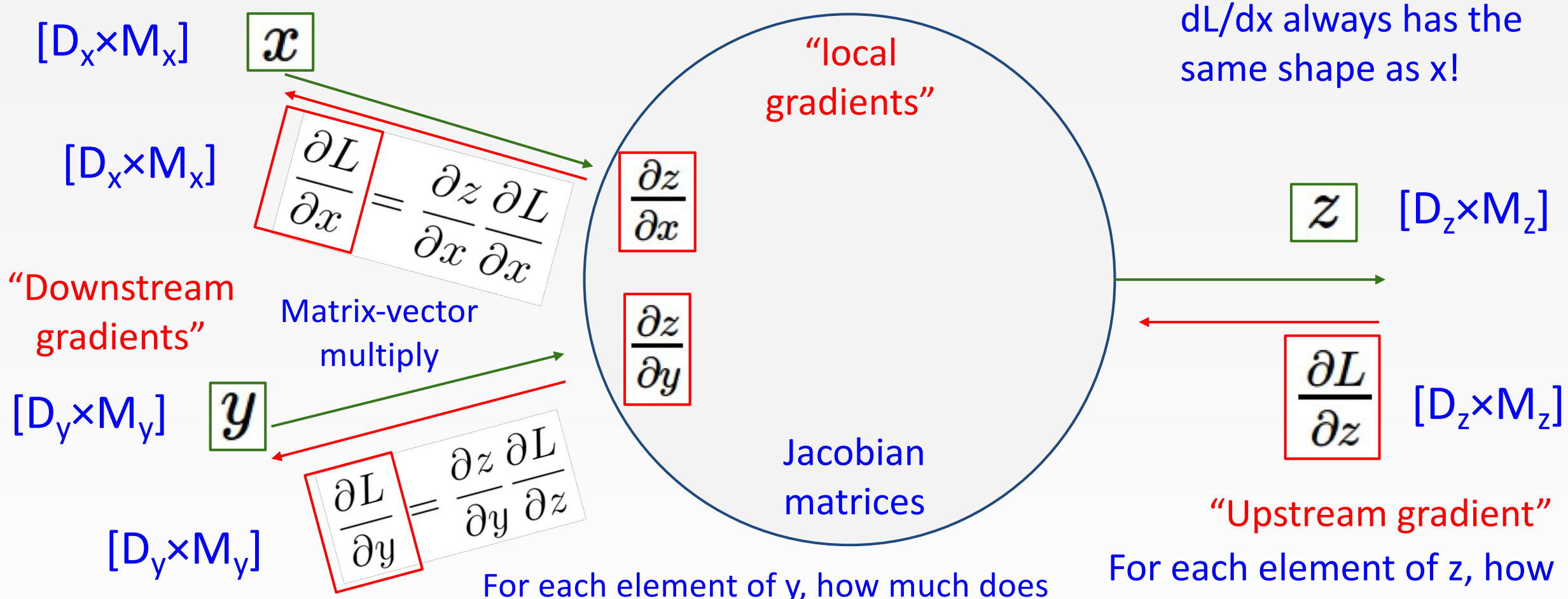


Jacobian is **sparse**: off-diagonal entries always zero! Never **explicitly** form Jacobian -- instead use **implicit** multiplication

BACKPROP WITH MATRICES (OR TENSORS)

Loss L still a scalar!

dL/dx always has the same shape as x!



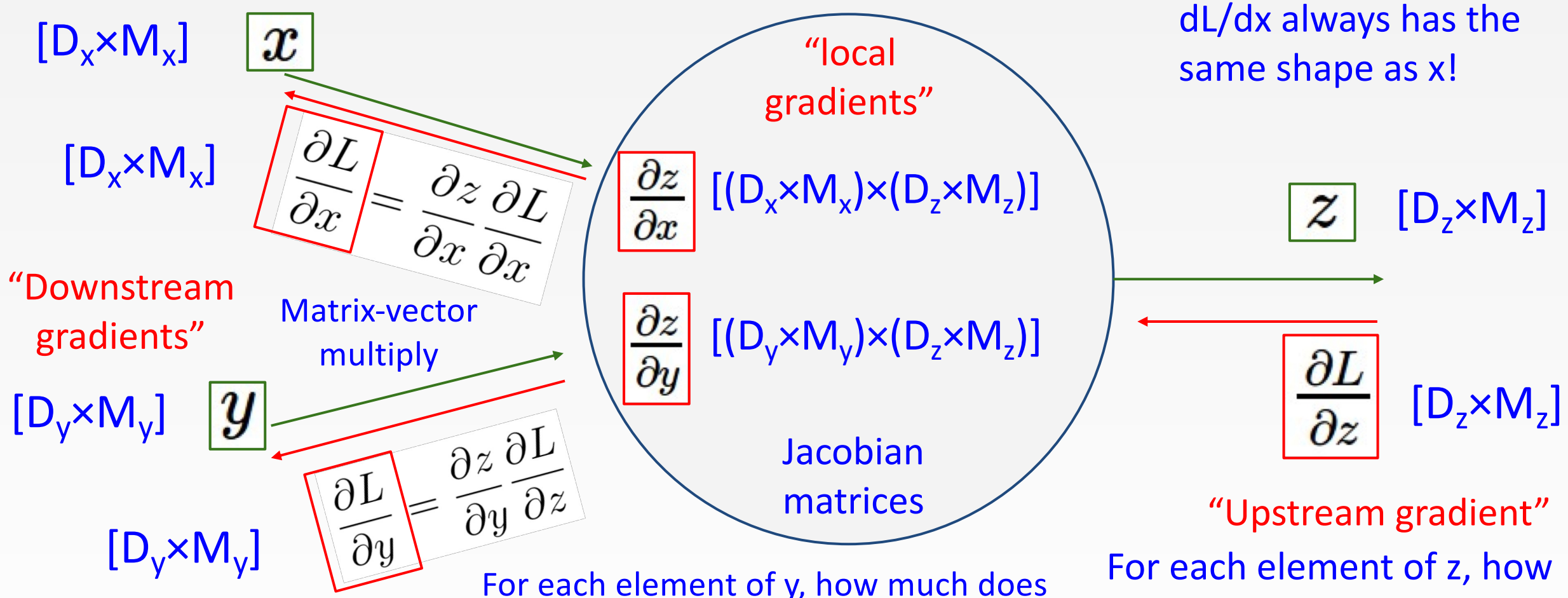
For each element of y, how much does it influence each element of z?

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?

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]
[5 2 17 1]

dL/dy: [N×M]

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

Also see derivation in the course notes:
http://cs231n.stanford.edu/handouts/linear-backprop.pdf

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]
[5 2 17 1]

dL/dy: [N×M]

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

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!

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]

[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]

[-8 1 4 6]

Q: What parts of y are affected by one element of x?

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]

[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]

[-8 1 4 6]

Q: What parts of y are affected by one element of x?

A: $x_{n,d}$ ffects the whole row $y_{n,\cdot}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]

[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]

[-8 1 4 6]

Q: What parts of y are affected by one element of x?

A: $x_{n,d}$ ffects the whole row $y_{n,\cdot}$

Q: How much does $x_{n,d}$ affect $y_{n,m}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}$$

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]

[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]

[-8 1 4 6]

Q: What parts of y are affected by one element of x?

A: $x_{n,d}$ ffects the whole row $y_{n,\cdot}$

Q: How much does $x_{n,d}$ affect $y_{n,m}$

A: $w_{d,m}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

BACKPROP WITH MATRICES

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

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

y: [N×M]

[**13 9 -2 -6**]
[5 2 17 1]

dL/dy: [N×M]

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

Q: What parts of y are affected by one element of x?

A: $x_{n,d}$ ffects the whole row $y_{n,\cdot}$

Q: How much does $x_{n,d}$ affect $y_{n,m}$

A: $w_{d,m}$

[N×D] [N×M] [M×D]

$$\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T$$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

BACKPROP WITH MATRICES

x: [N×D]

[2 **1** -3]

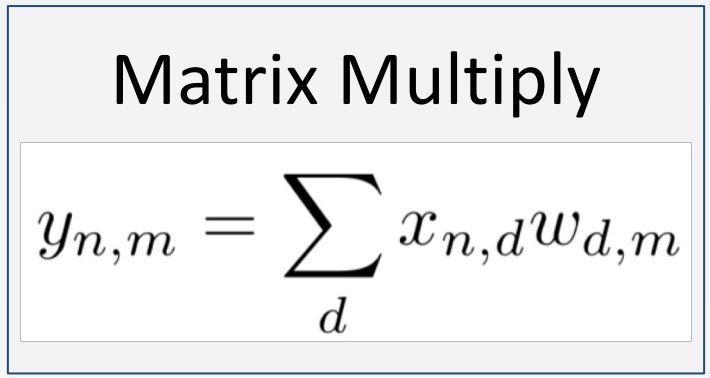
[-3 4 2]

w: [D×M]

[3 2 1 -1]

[2 1 **3** 2]

[3 2 1 -2]



y: [N×M]

[**13 9 -2 -6**]

[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]

[-8 1 4 6]

By similar logic:

[N×D] [N×M] [M×D]

$$\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T$$

[D×M] [D×N] [N×M]

$$\frac{\partial L}{\partial w} = x^T \left(\frac{\partial L}{\partial y}\right)$$

These formulas are easy to remember: they are the only way to make shapes match up!

SUMMARY

- **(Fully-connected) Neural Networks**
  - Stacks of linear functions and nonlinear activation functions; they have much more representational power than linear classifiers
- **Backpropagation**
  - Recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates

SUMMARY

- Implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() API
- **Forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **Backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs

NEXT LECTURE

- Introduction to database systems
- Advanced SQL