

# Lecture 4: Neural Networks and Backpropagation

# Where we are...

$$s = f(x; W) = Wx \quad \text{Linear score function}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM loss (or softmax)}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda \sum_k W_k^2 \quad \text{data loss + regularization}$$

How to find the best  $W$ ?

$$\boxed{\nabla_W L}$$

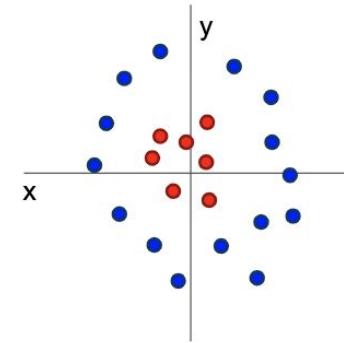
# Problem: Linear Classifiers are not very powerful

## Visual Viewpoint



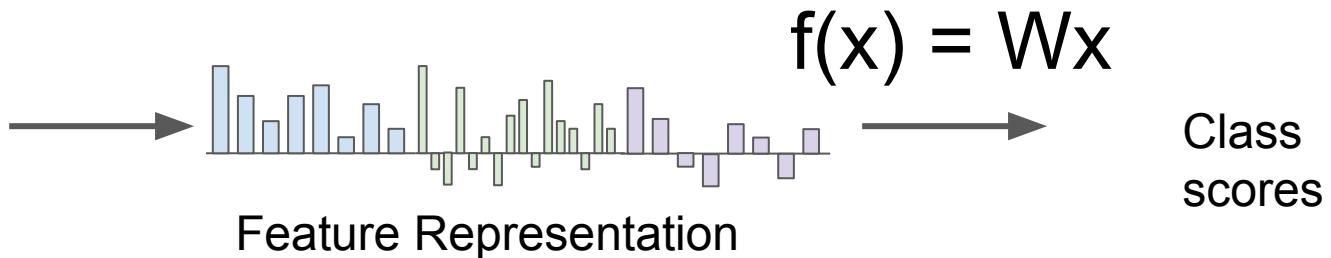
Linear classifiers learn  
one template per class

## Geometric Viewpoint



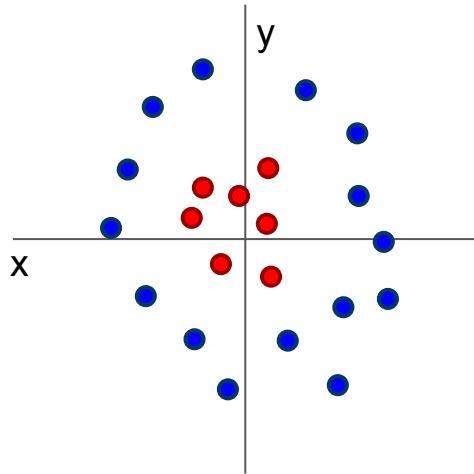
Linear classifiers  
can only draw linear  
decision boundaries

# Image Features



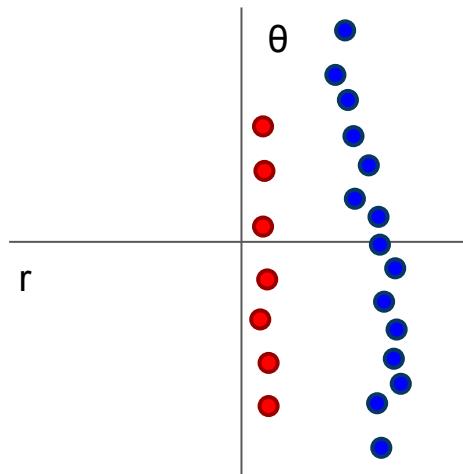
feed feature representation instead of raw pixels

# Image Features: Motivation



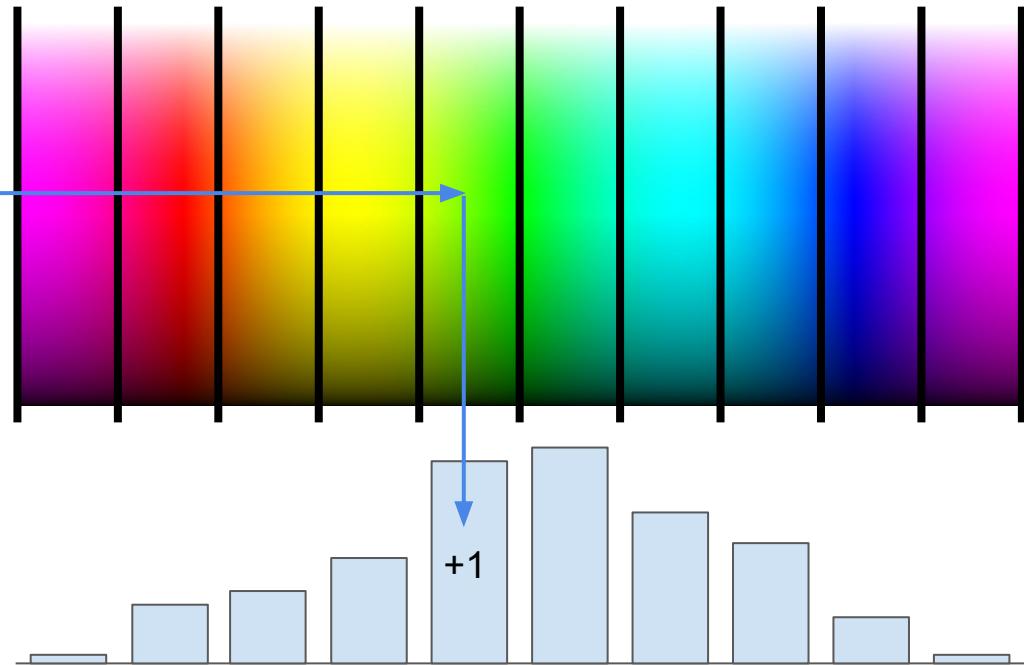
Cannot separate red  
and blue points with  
linear classifier

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



After applying feature  
transform, points can  
be separated by linear  
classifier

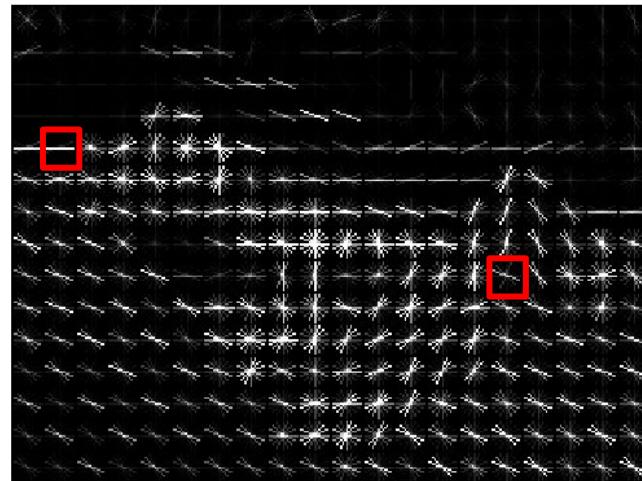
# Example: Color Histogram



# Example: Histogram of Oriented Gradients (HoG)



Divide image into 8x8 pixel regions  
Within each region quantize edge  
direction into 9 bins



Example: 320x240 image gets divided  
into 40x30 bins; in each bin there are  
9 numbers so feature vector has  
 $30*40*9 = 10,800$  numbers

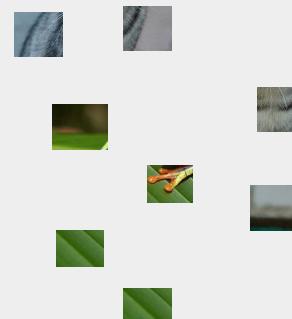
Lowe, "Object recognition from local scale-invariant features", ICCV 1999  
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

# Example: Bag of Words

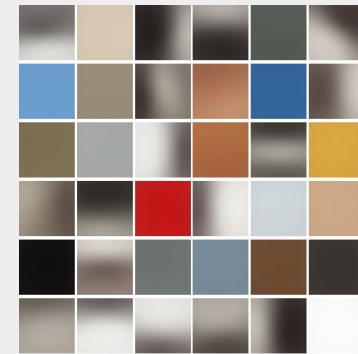
## Step 1: Build codebook



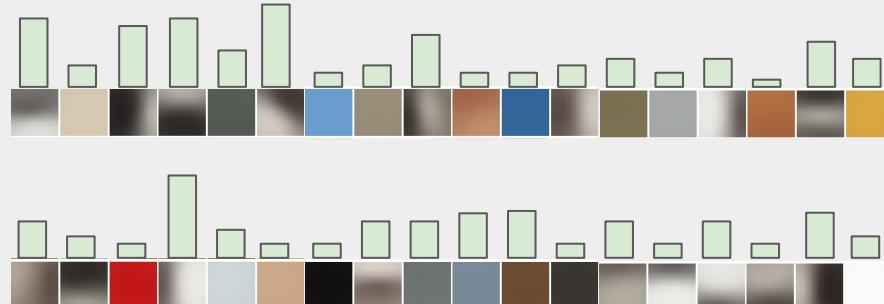
Extract random patches



Cluster patches to form “codebook” of “visual words”

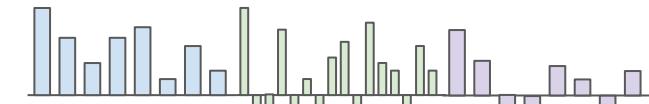
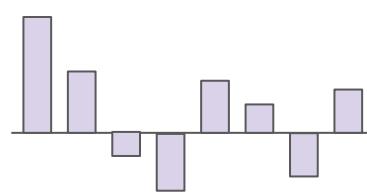
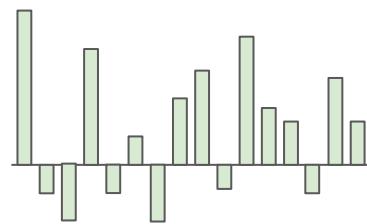
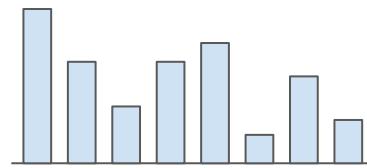


## Step 2: Encode images

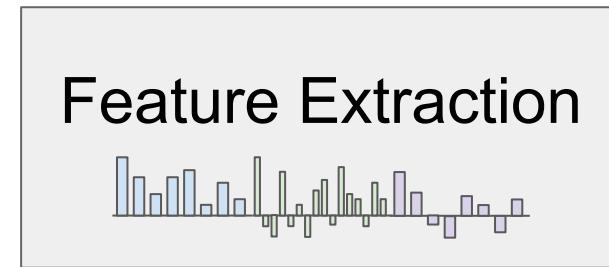


Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005

# Image Features



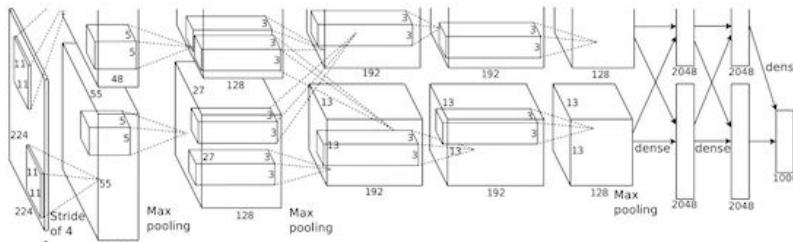
# Image features vs ConvNets



$f$

training

10 numbers giving scores for classes

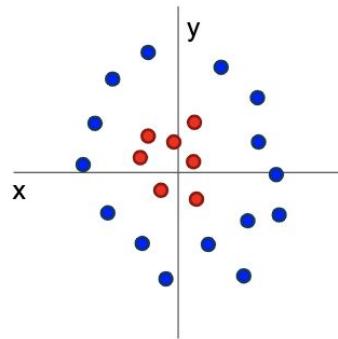


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.  
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012.  
Reproduced with permission.

training

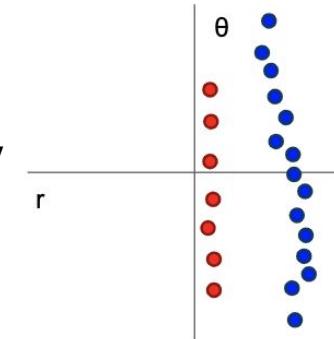
10 numbers giving scores for classes

# One Solution: Feature Transformation

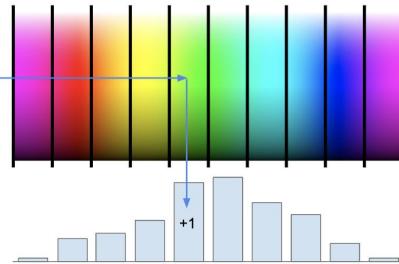


$$f(x, y) = (r(x, y), \theta(x, y))$$

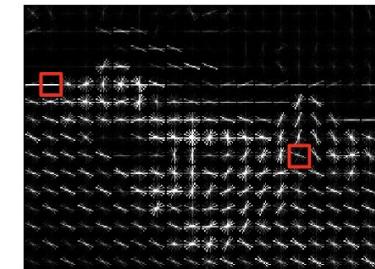
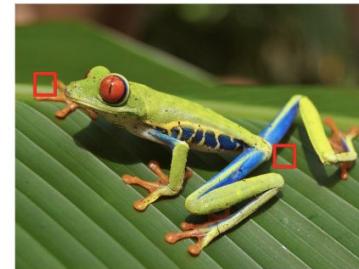
Transform data with a cleverly chosen **feature transform**  $f$ , then apply linear classifier



Color Histogram



Histogram of Oriented Gradients (HoG)



# Today: Neural Networks

# 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 Network” is a very broad term; these are more accurately called “fully-connected networks” or sometimes “multi-layer perceptrons” (MLP)

(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

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

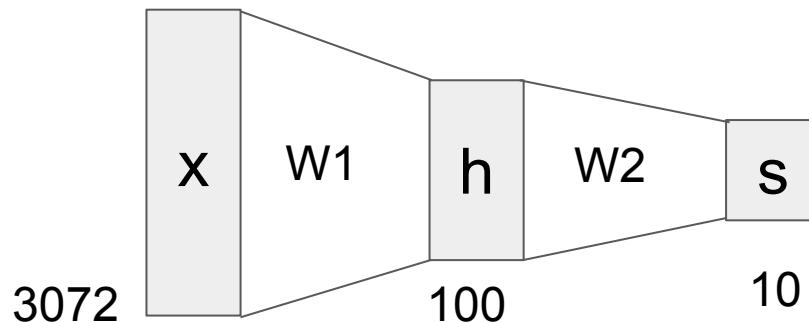
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

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

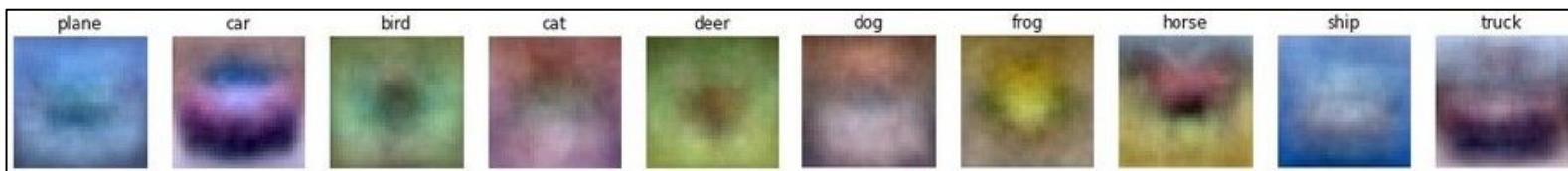
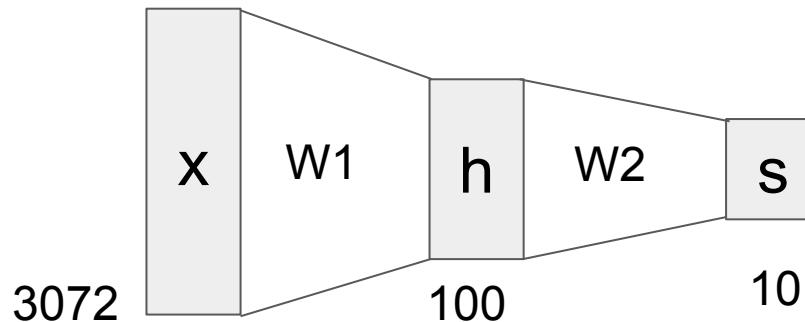


$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural networks: without the brain stuff

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

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



Learn 100 templates instead of 10.

Share templates between classes

# 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  $\max(0, z)$  is called the **activation function**.  
do some transform on `Wx+b`

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

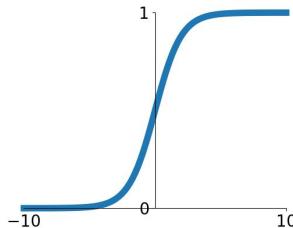
$$f = W_2 W_1 x \quad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

**A:** We end up with a linear classifier again!

# Activation functions

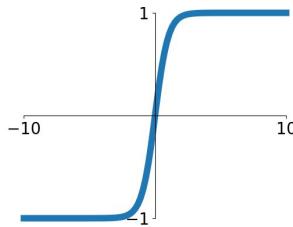
## Sigmoid

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



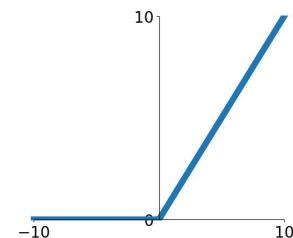
## tanh

$$\tanh(x)$$



## ReLU

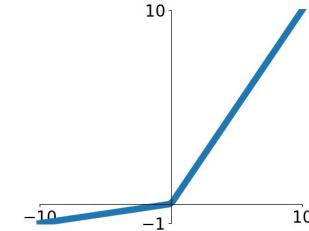
$$\max(0, x)$$



ReLU is a good default choice for most problems

## Leaky ReLU

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

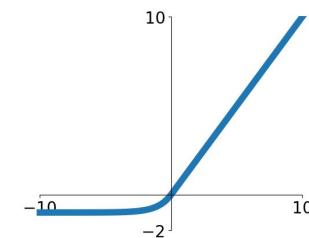


## Maxout

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

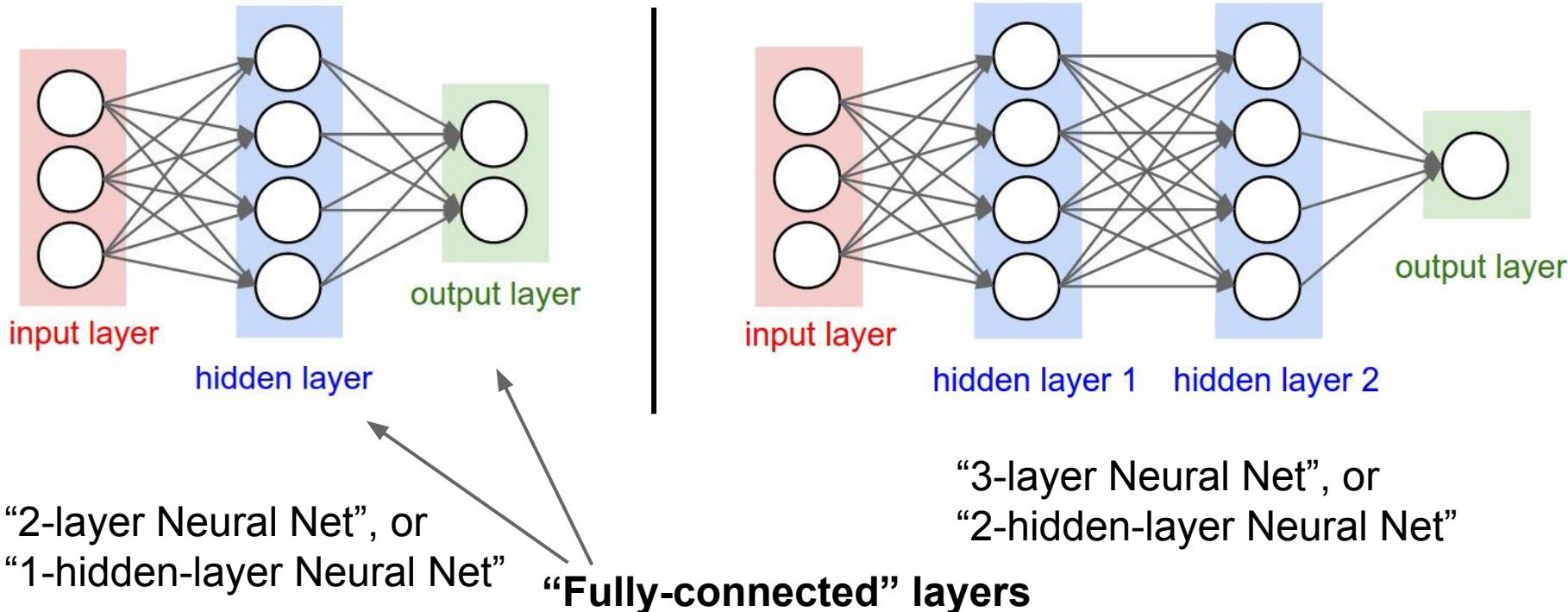
## ELU

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

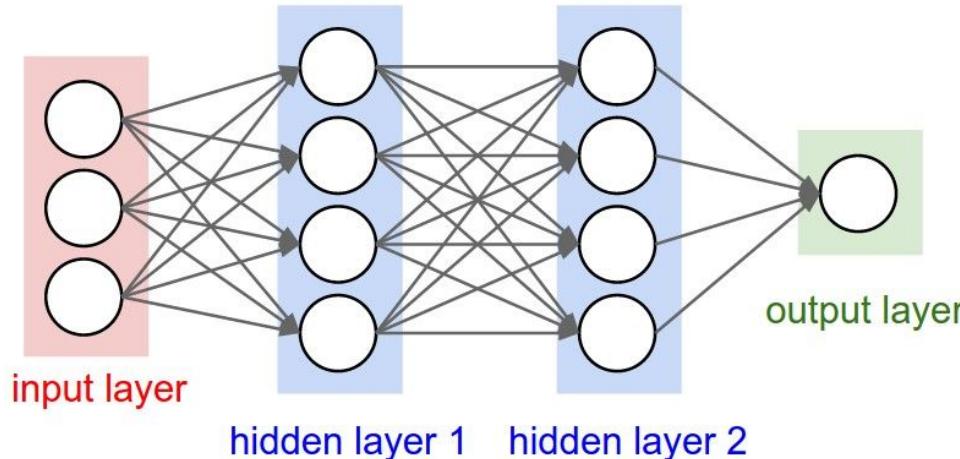


Softmax is an activation function as well

# Neural networks: Architectures



# Example 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)
```

# Full implementation of training a 2-layer Neural Network needs ~20 lines:

```
1 import numpy as np
2 from numpy.random import randn
3
4 N, D_in, H, D_out = 64, 1000, 100, 10
5 x, y = randn(N, D_in), randn(N, D_out)
6 w1, w2 = randn(D_in, H), randn(H, D_out)
7
8 for t in range(2000):
9     h = 1 / (1 + np.exp(-x.dot(w1)))
10    y_pred = h.dot(w2)
11    loss = np.square(y_pred - y).sum()
12    print(t, loss)
13
14 grad_y_pred = 2.0 * (y_pred - y)
15 grad_w2 = h.T.dot(grad_y_pred)
16 grad_h = grad_y_pred.dot(w2.T)
17 grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19 w1 -= 1e-4 * grad_w1
20 w2 -= 1e-4 * grad_w2
```

Define the network

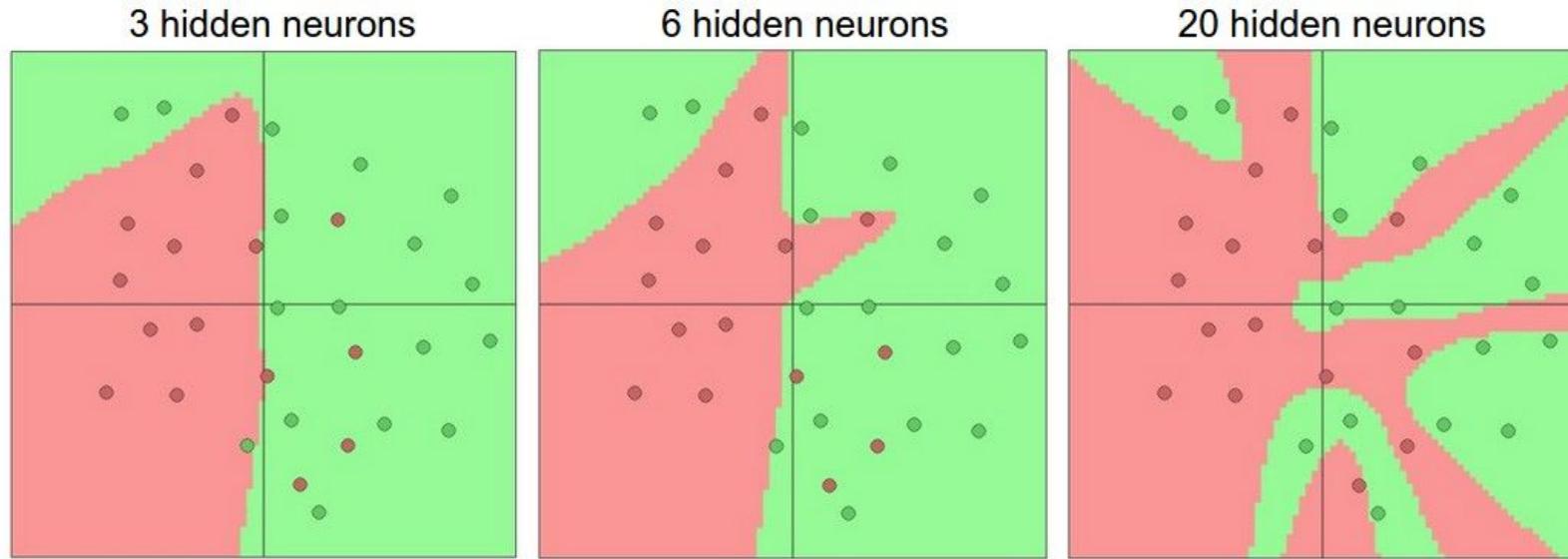
Forward pass

for 2+D array, `dot` is same as `np.matmul`

Calculate the analytical gradients

Gradient descent

# Setting the number of layers and their sizes



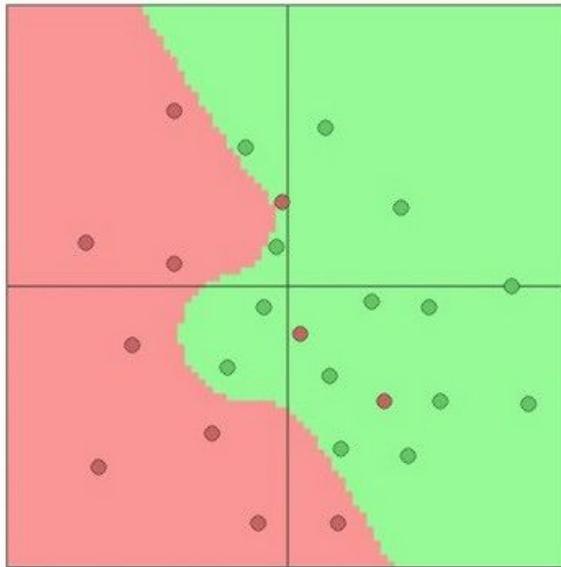
more neurons = more capacity

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

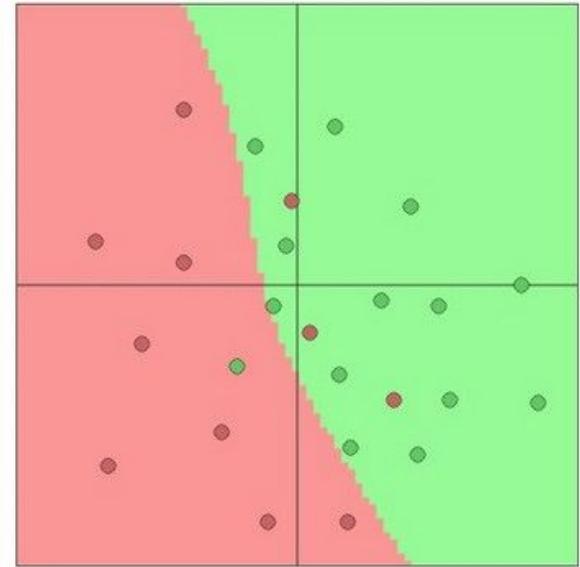
$\lambda = 0.001$



$\lambda = 0.01$



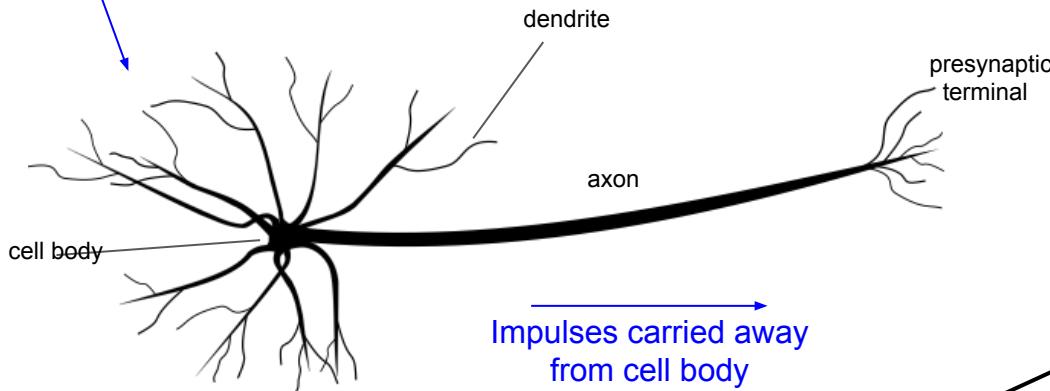
$\lambda = 0.1$



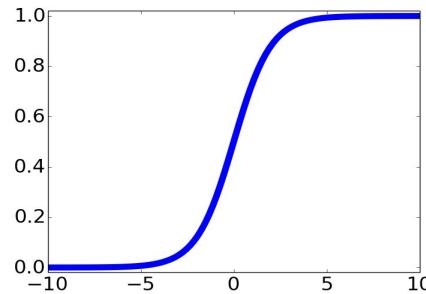
(Web demo with ConvNetJS:  
[http://cs.stanford.edu/people/karpathy/convnetjs/demo  
/classify2d.html](http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html))

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Impulses carried toward cell body



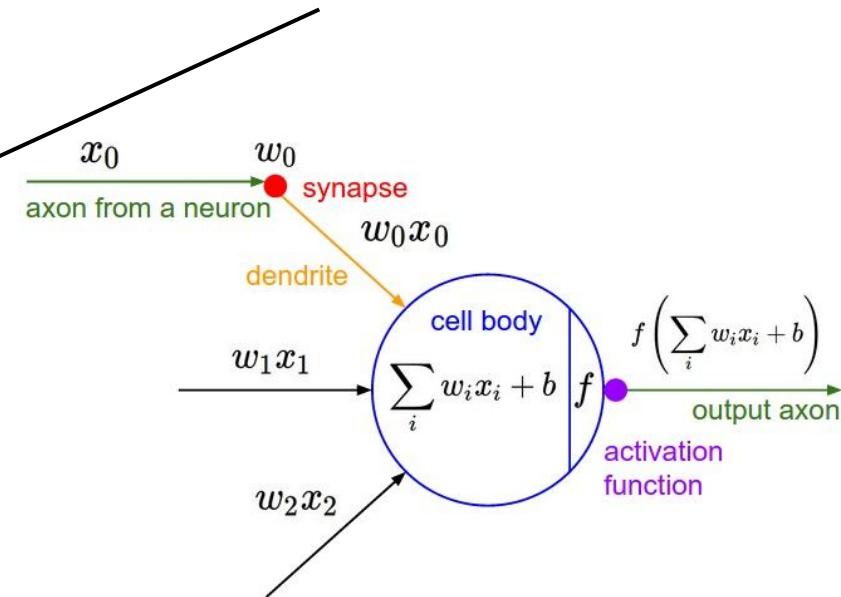
This image by Felipe Perucho  
is licensed under CC-BY 3.0



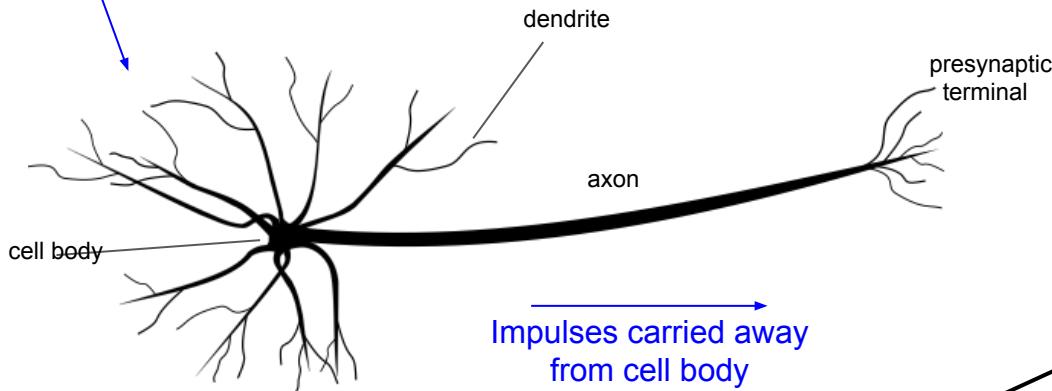
sigmoid activation function

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

Impulses carried away from cell body

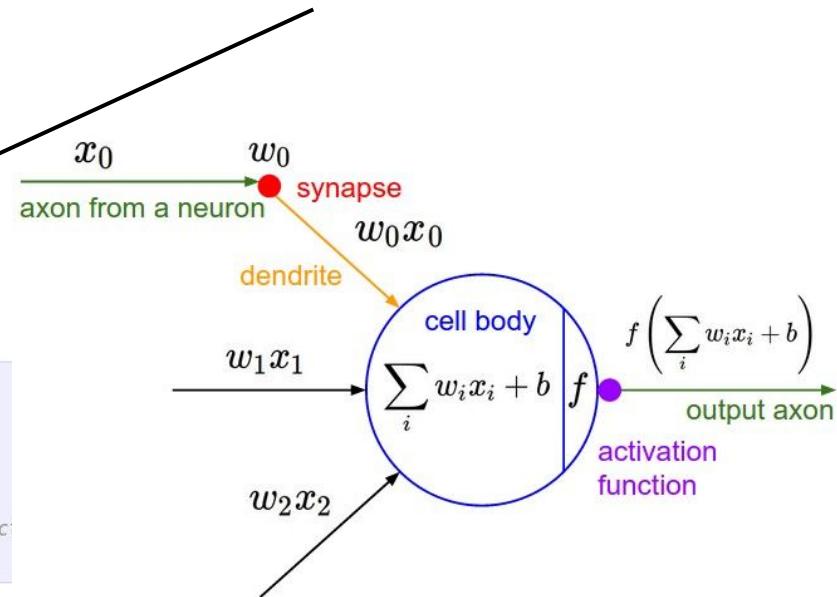


Impulses carried toward cell body

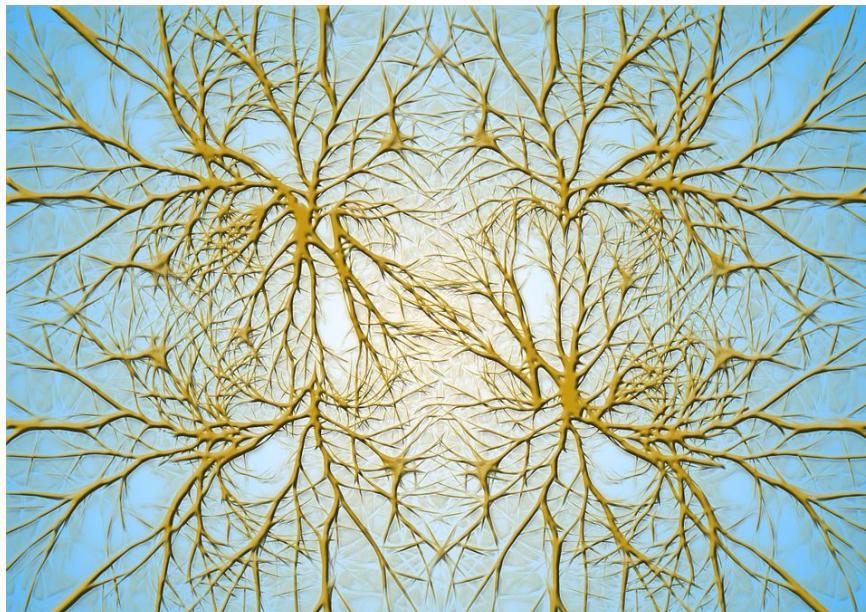


This image by Felipe Perucho  
is licensed under CC-BY 3.0

```
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
```

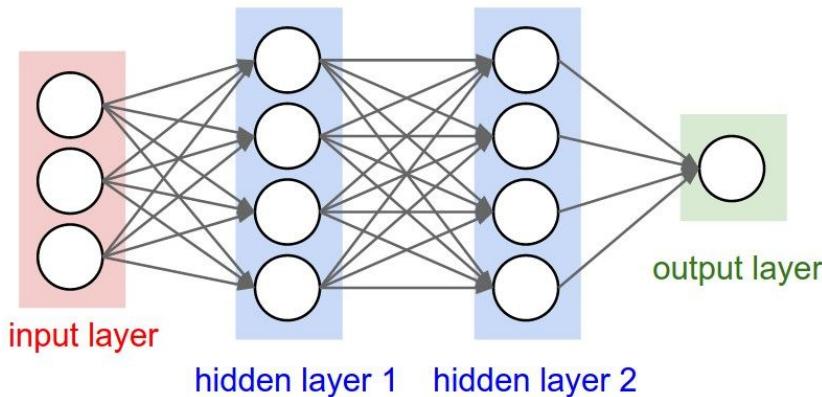


## Biological Neurons: Complex connectivity patterns

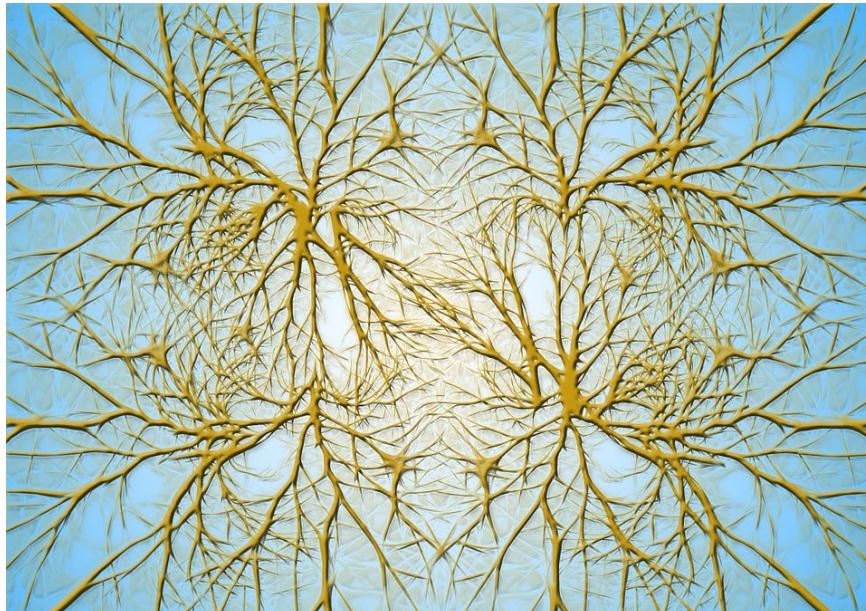


[This image is CC0 Public Domain](#)

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

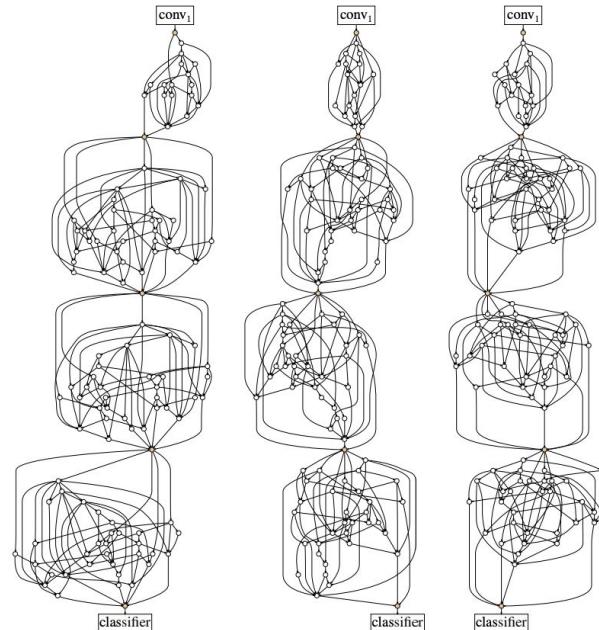


# Biological Neurons: Complex connectivity patterns



[This image is CC0 Public Domain](#)

But neural networks with random connections can work too!



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

# 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

[Dendritic Computation. London and Häusser]

how to compute the analytic gradient for arbitrarily complex functions, using a framework that I'm going to call computational graphs.

# Problem: How to compute gradients?

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

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

$$R(W) = \sum_k W_k^2 \quad \text{Regularization}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(W_1) + \lambda R(W_2) \quad \text{Total loss: data loss + regularization}$$

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

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

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

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

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

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

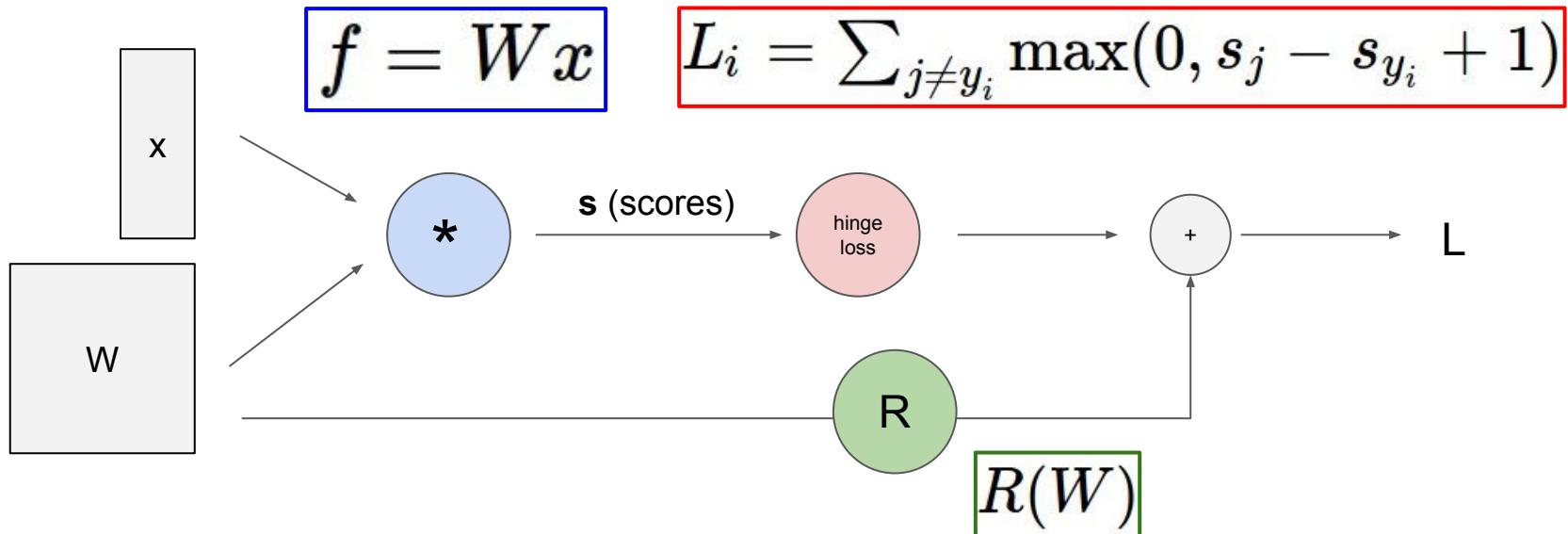
$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

**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



The advantage is that once we can express a function using a computational graph, then we can use a technique called 'backpropagation' which is going to recursively use the chain rule in order to compute the gradient with respect to every variable in the computational graph.

# Solution: Backpropagation

## Backpropagation: a simple example

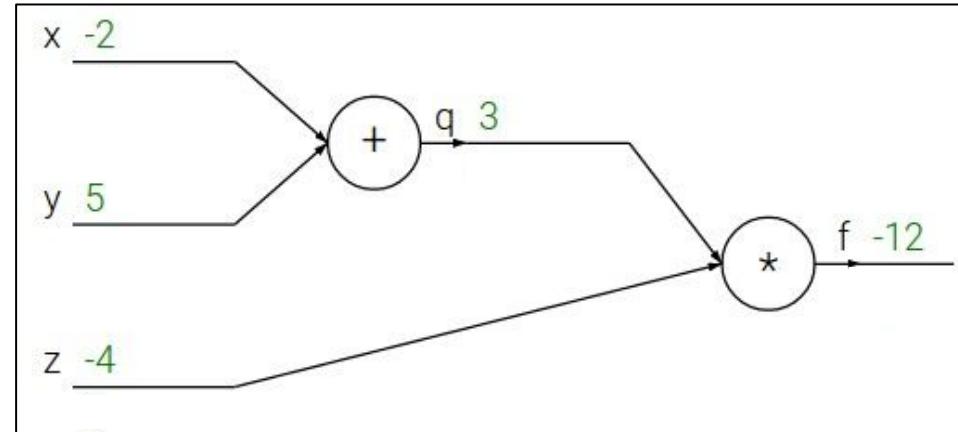
$$f(x, y, z) = (x + y)z$$

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

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

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

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



So what we want to find, is the gradients of  $f$  with respect to  $x, y$ , and  $z$ .

## Backpropagation: a simple example

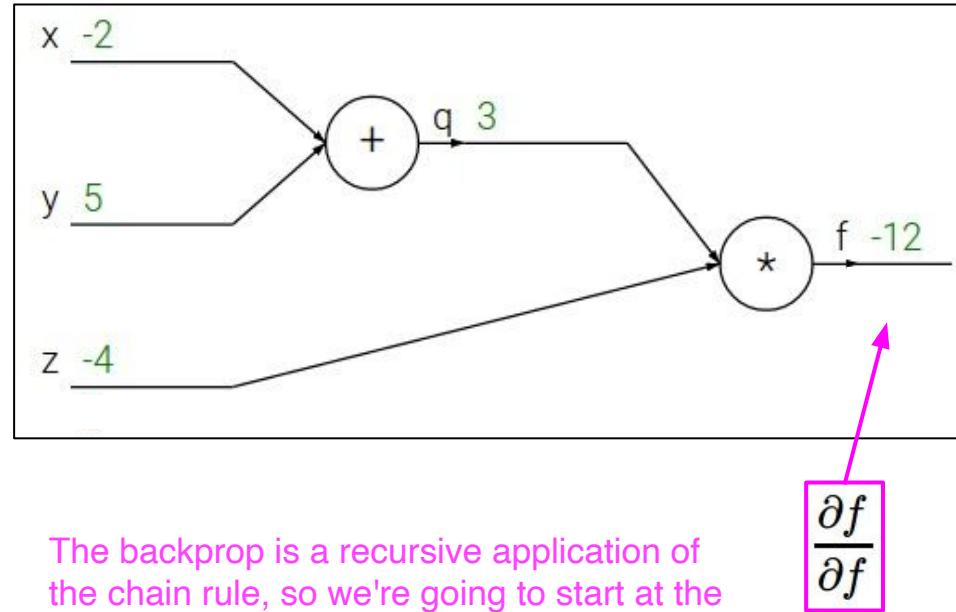
$$f(x, y, z) = (x + y)z$$

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

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

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

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



The backprop is a recursive application of the chain rule, so we're going to start at the back, the very end of the computational graph, and then we're going to work our way backwards. and compute all the gradients along the way.

## Backpropagation: a simple example

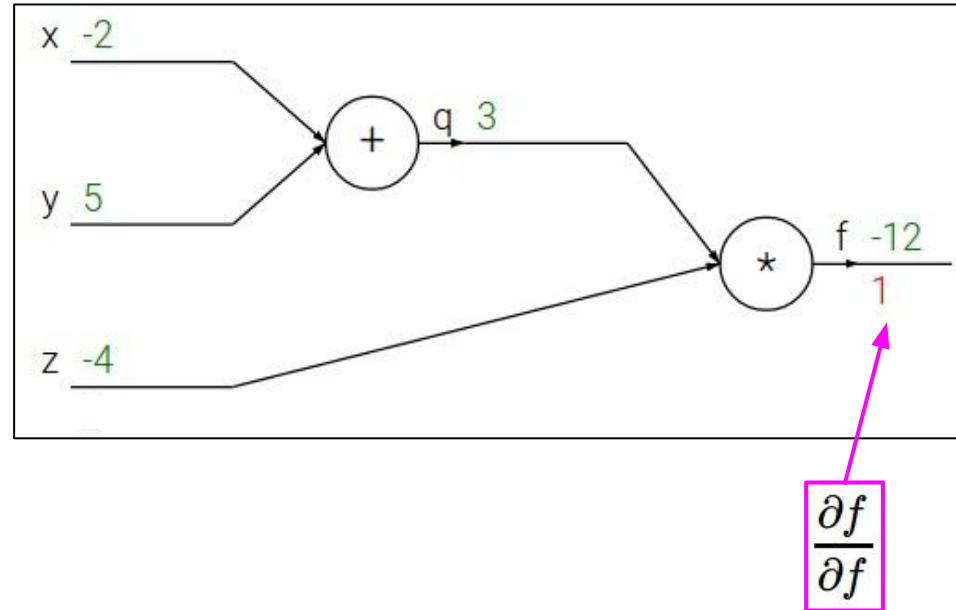
$$f(x, y, z) = (x + y)z$$

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

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

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

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



So here we want to compute the gradient of the output with respect to the last variable,  $f$ . And this gradient is just 1.

## Backpropagation: a simple example

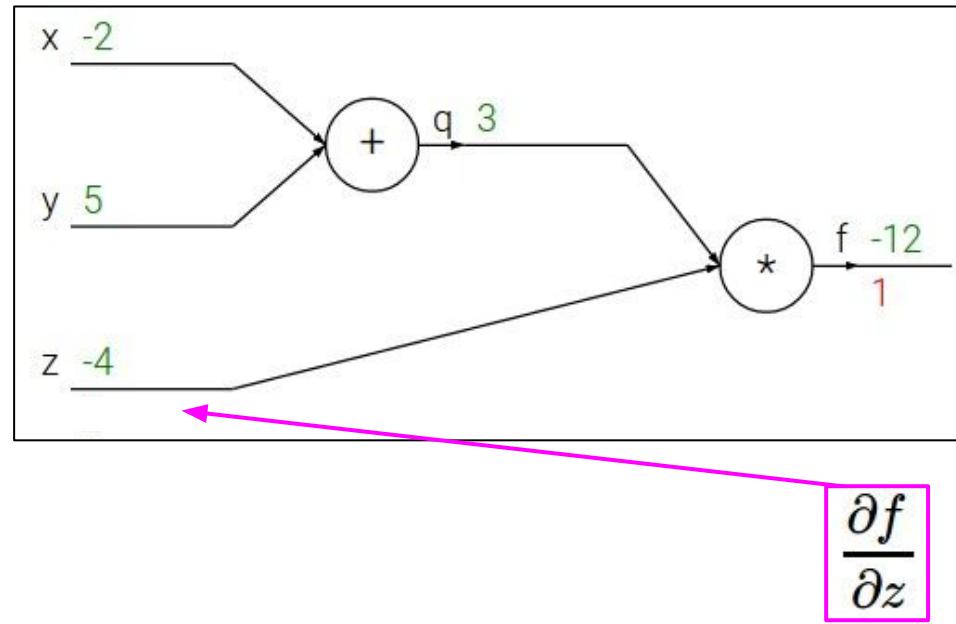
$$f(x, y, z) = (x + y)z$$

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

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

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

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

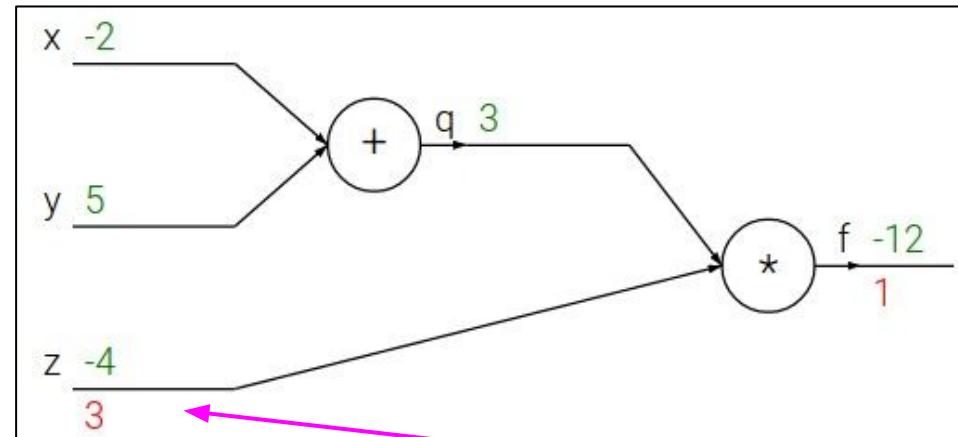
$$f(x, y, z) = (x + y)z$$

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

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

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

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



So now, moving backwards, we want to gradient with respect to  $z$ . We know  $\frac{\partial f}{\partial z} = q$ . So  $\frac{\partial f}{\partial z} = q = 3$ .

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

## Backpropagation: a simple example

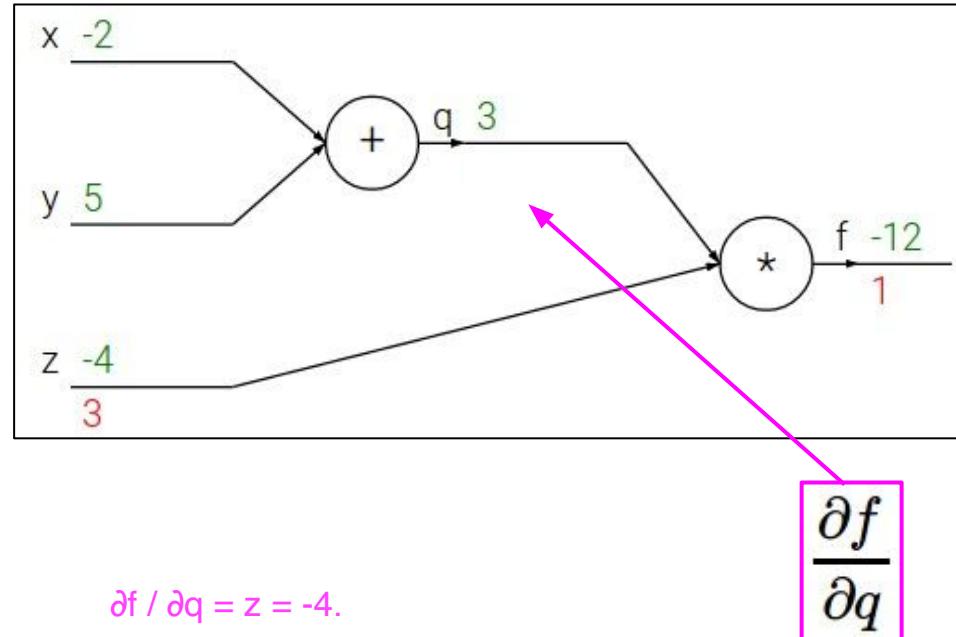
$$f(x, y, z) = (x + y)z$$

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

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

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

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



$$\frac{\partial f}{\partial q} / \frac{\partial q}{\partial z} = z = -4.$$

## Backpropagation: a simple example

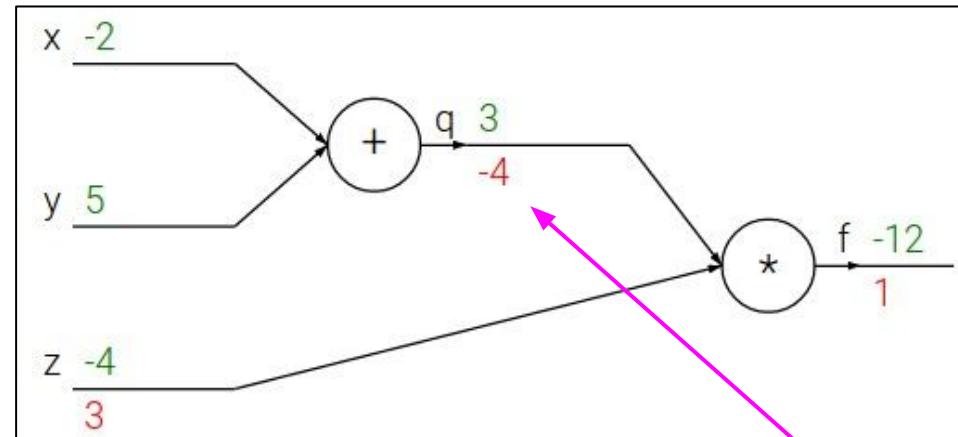
$$f(x, y, z) = (x + y)z$$

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

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

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

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



What is the value of  $y$ ?

$y$  is not connected directly to  $f$ , it's connected through an intermediate node of  $z$ .

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

## Backpropagation: a simple example

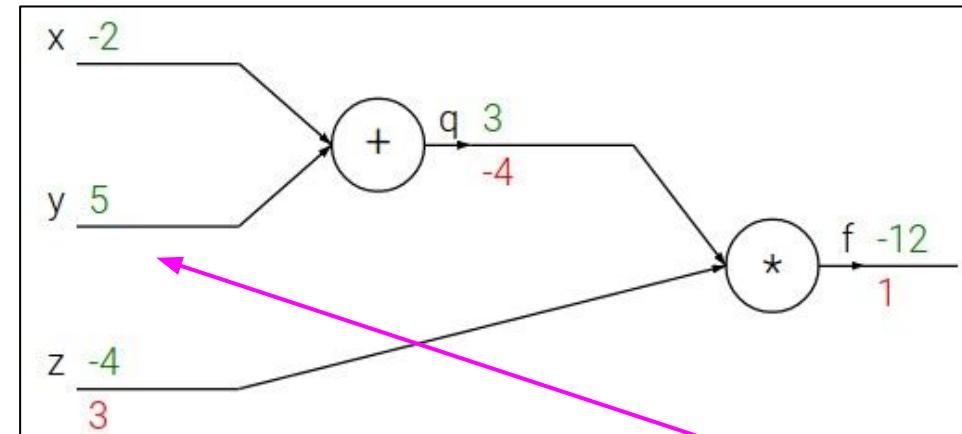
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

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

Upstream  
gradient

Local  
gradient

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

## Backpropagation: a simple example

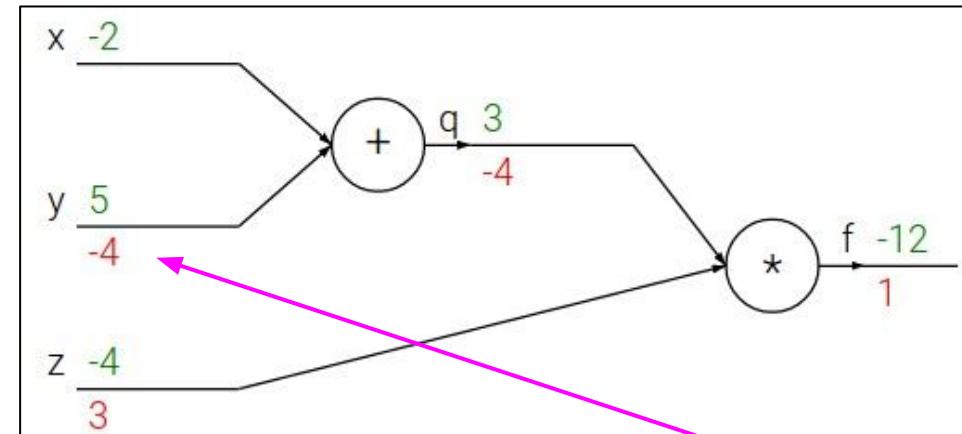
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

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

Upstream  
gradient

Local  
gradient

## Backpropagation: a simple example

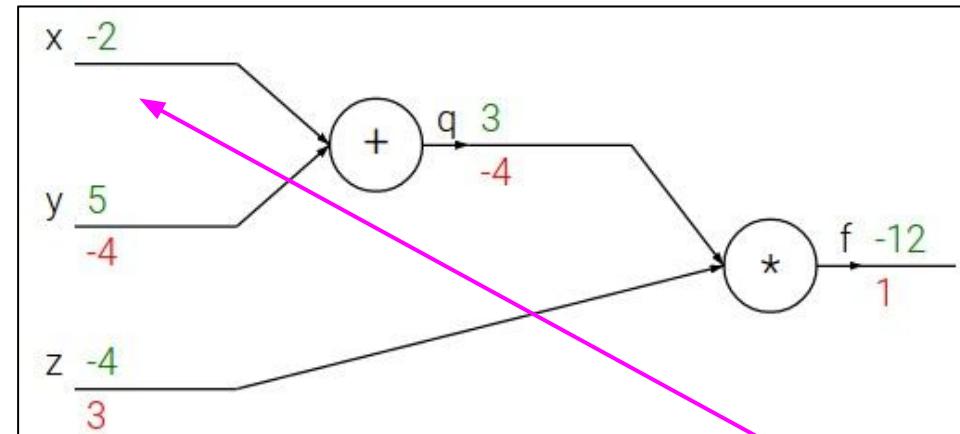
$$f(x, y, z) = (x + y)z$$

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

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

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

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

## Backpropagation: a simple example

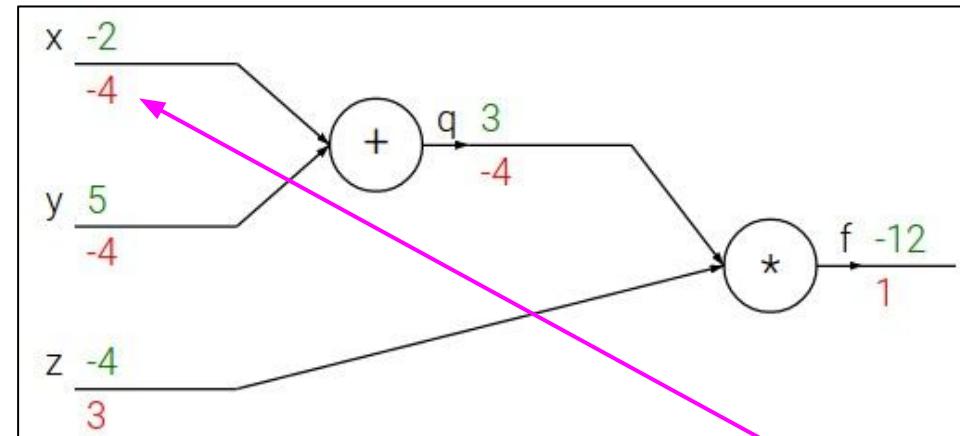
$$f(x, y, z) = (x + y)z$$

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

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

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

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

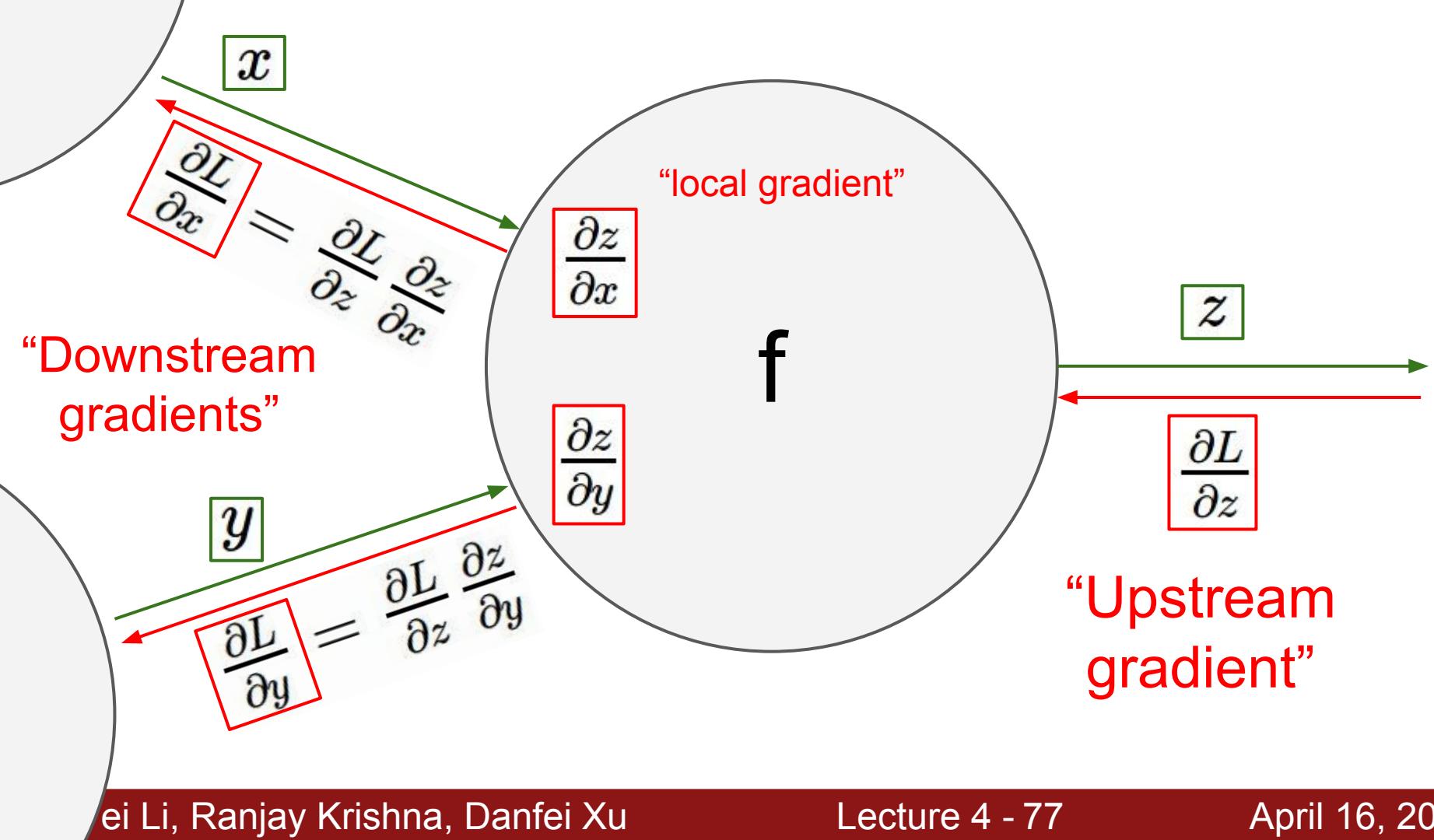


Chain rule:

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

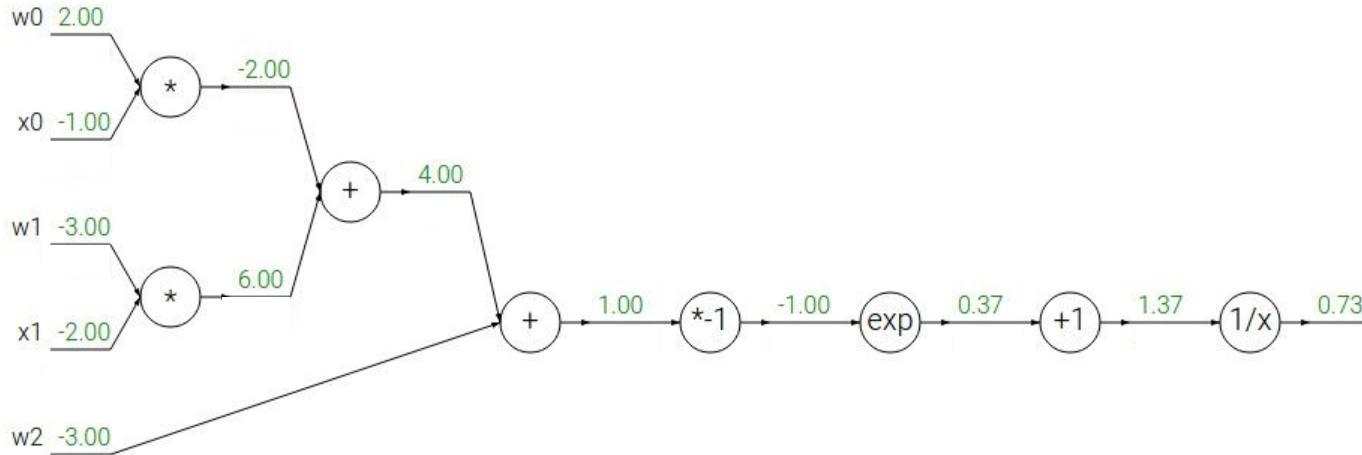
Upstream  
gradient

Local  
gradient



Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

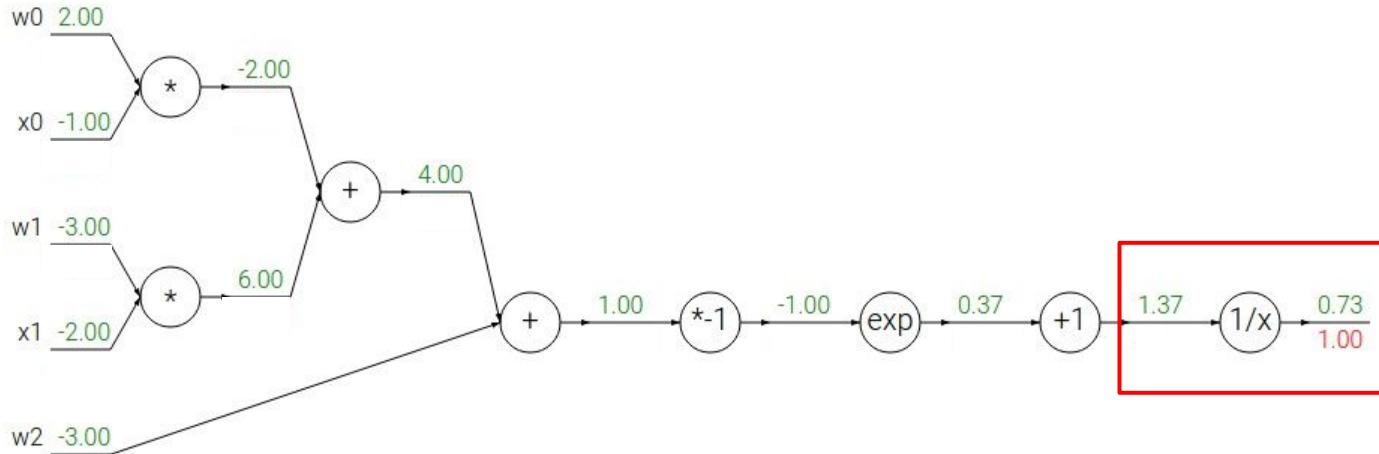
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

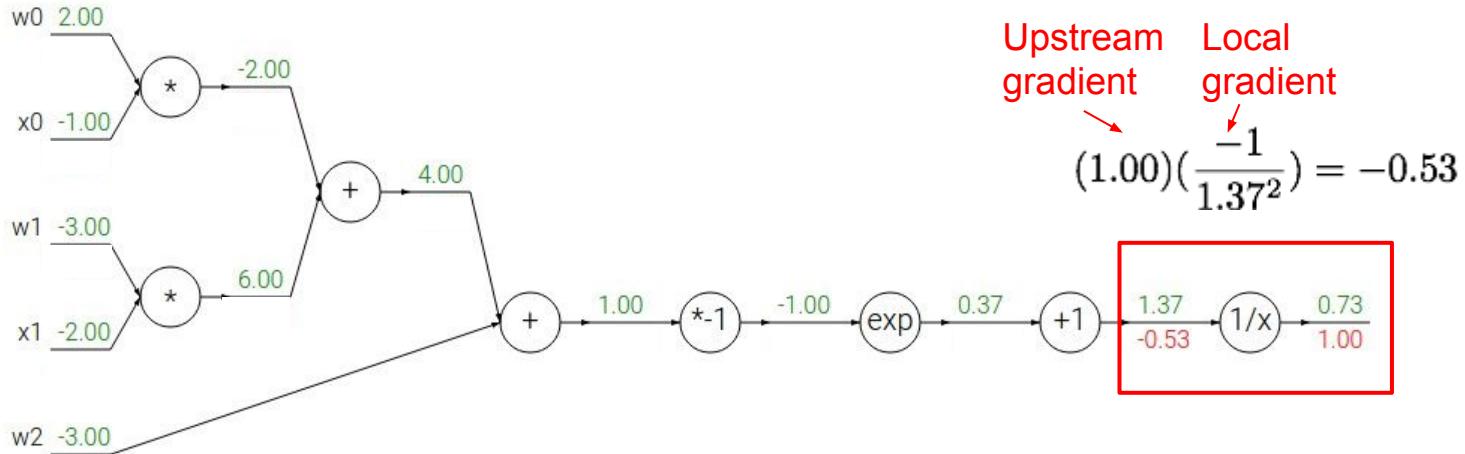
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

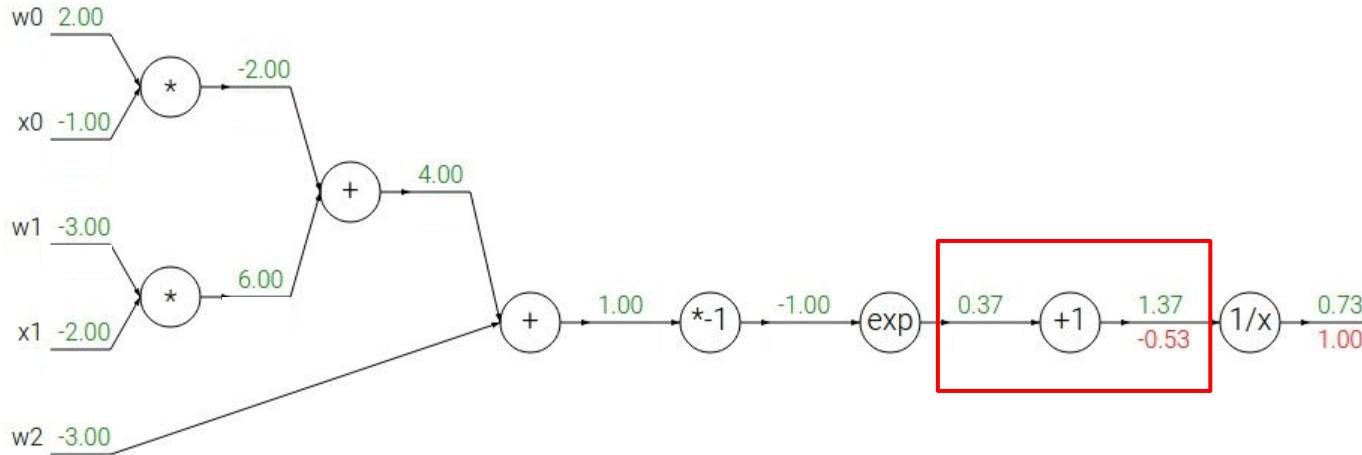
$$f_c(x) = c + x$$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

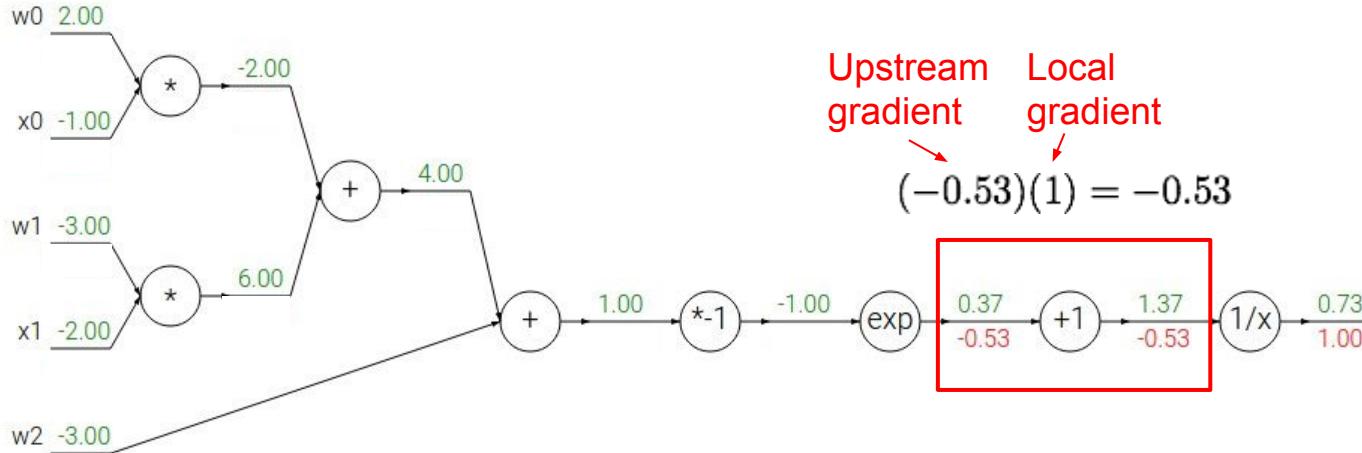
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

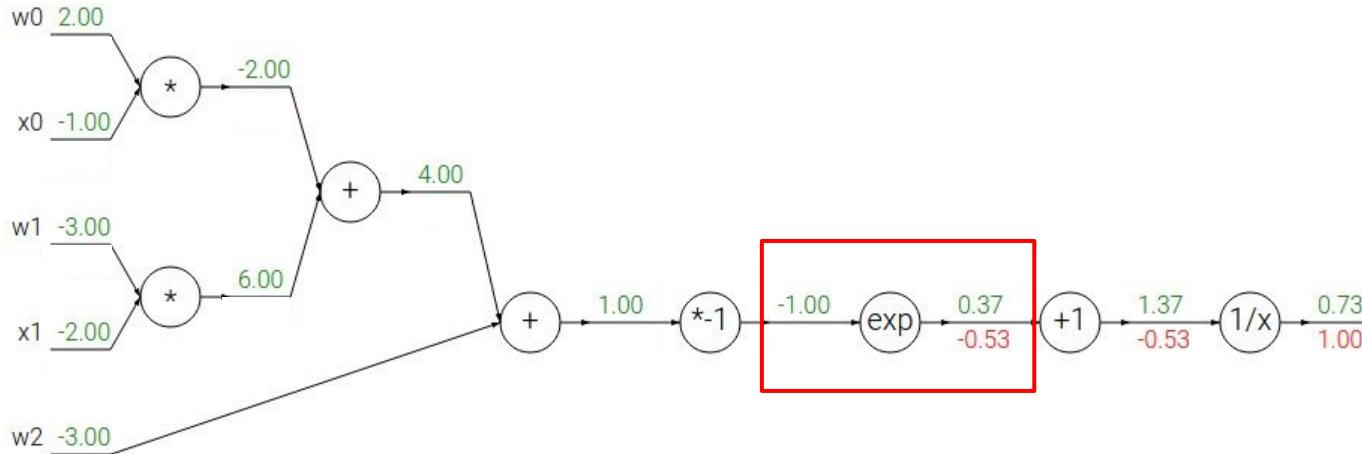
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

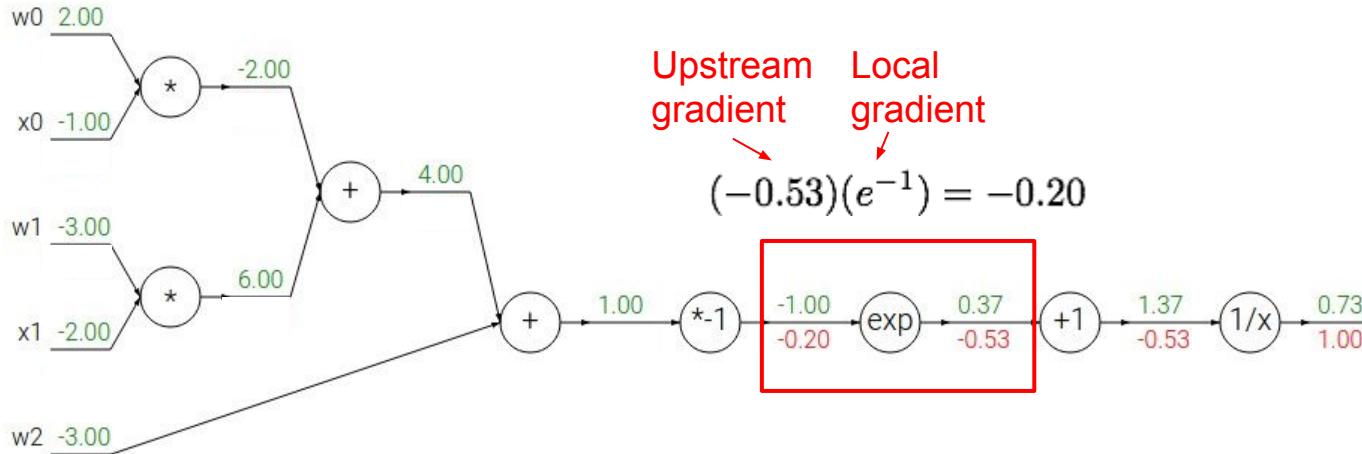
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

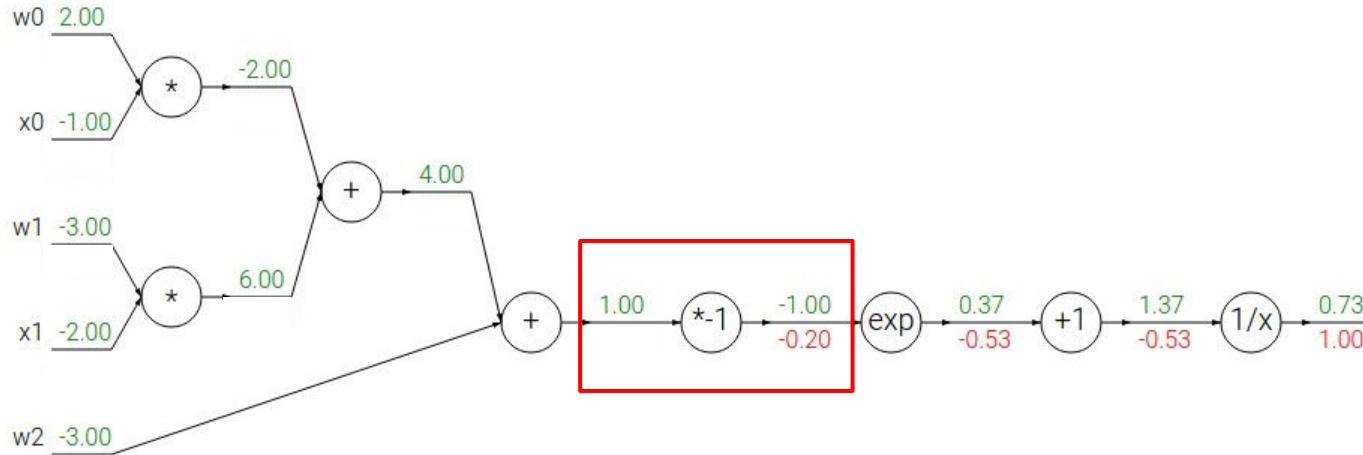
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

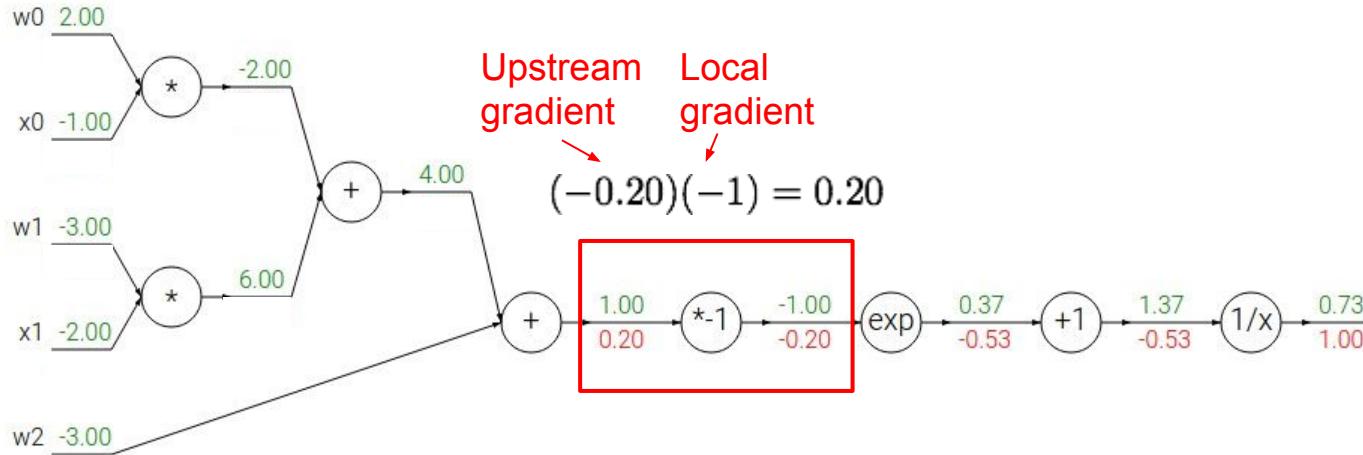
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

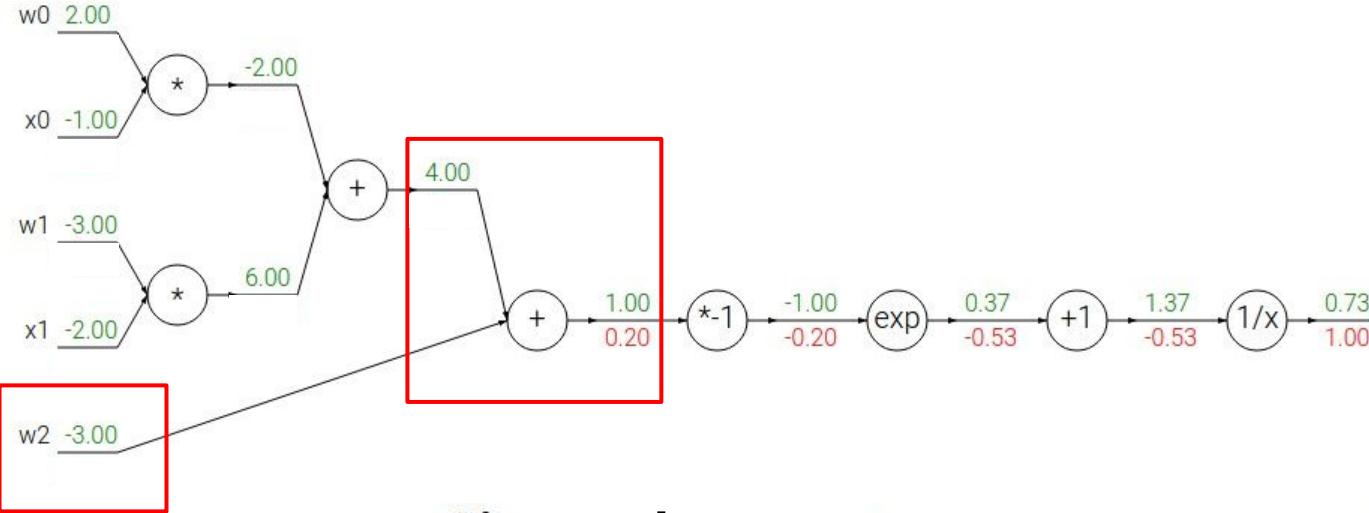
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

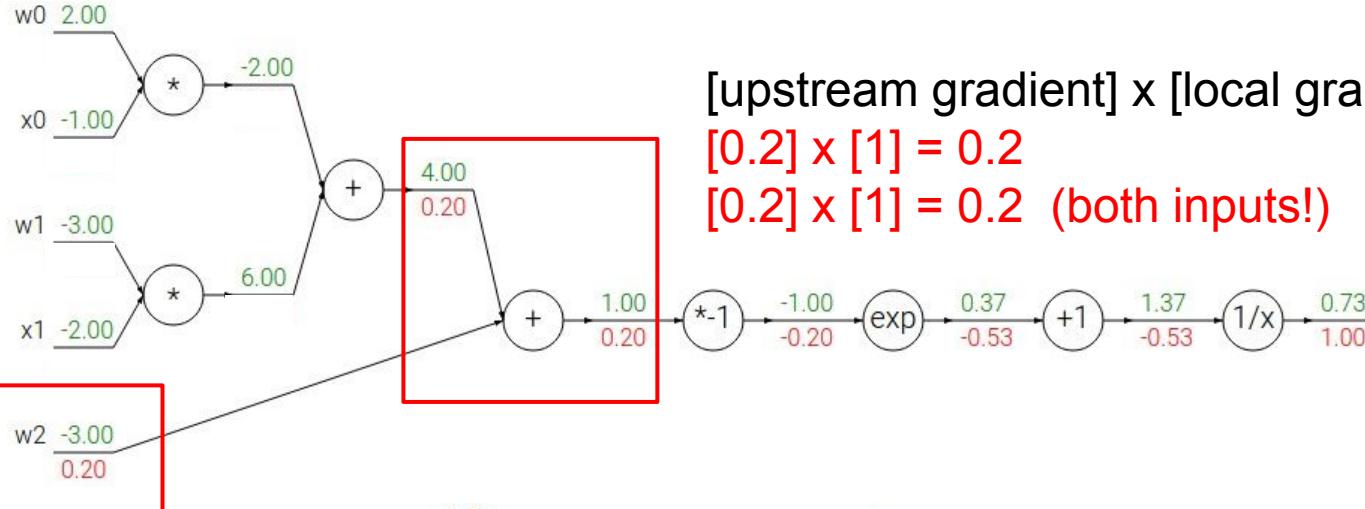
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

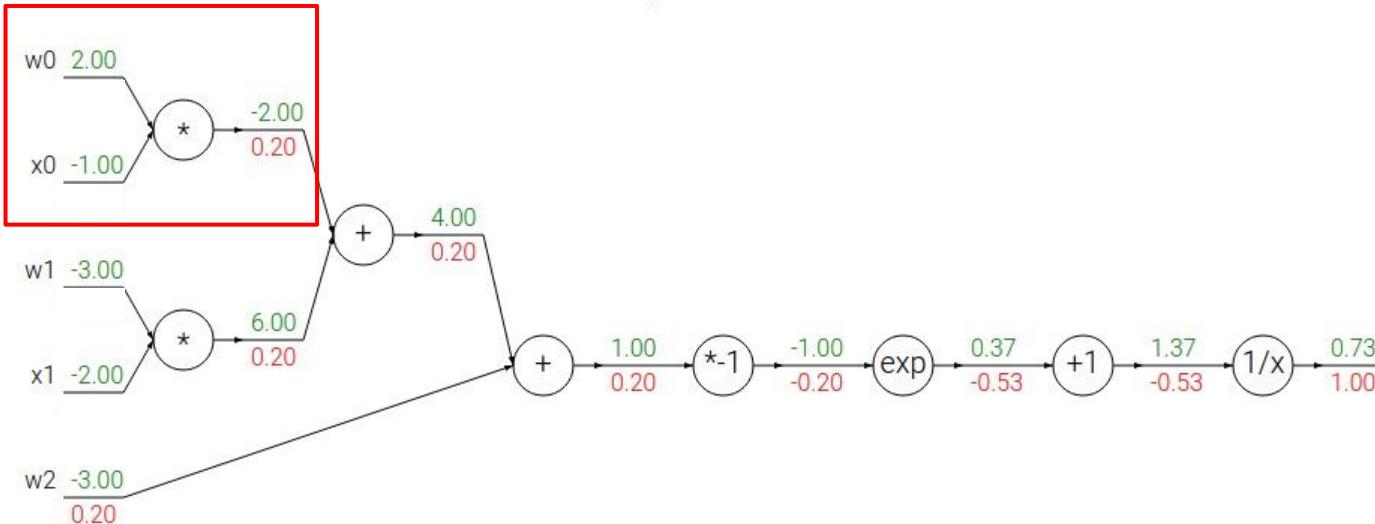
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

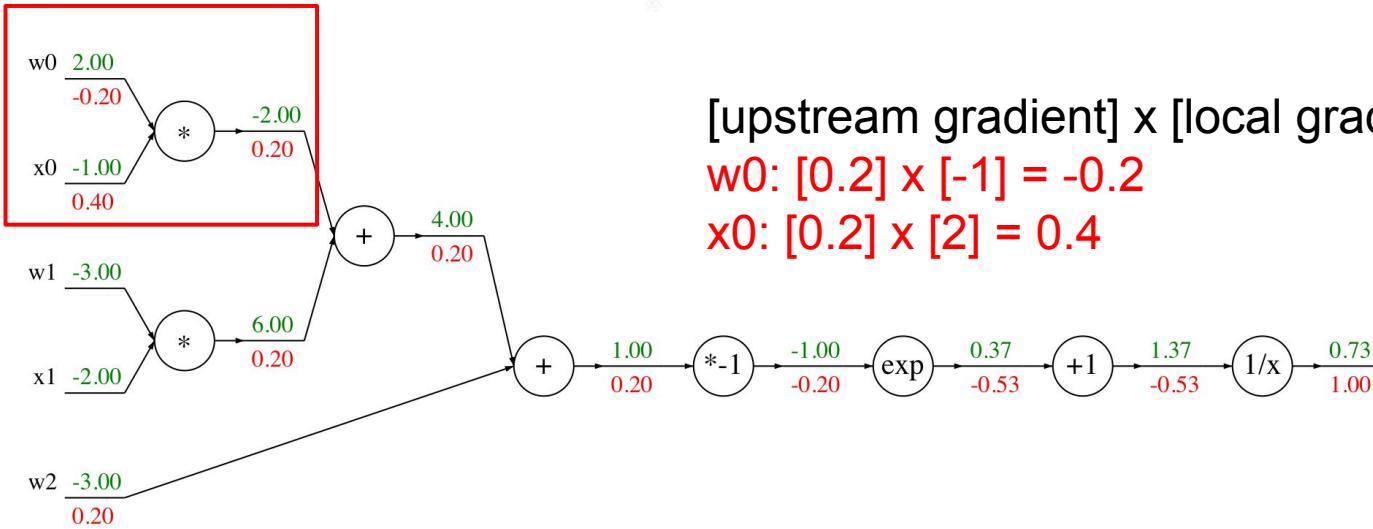
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

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



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

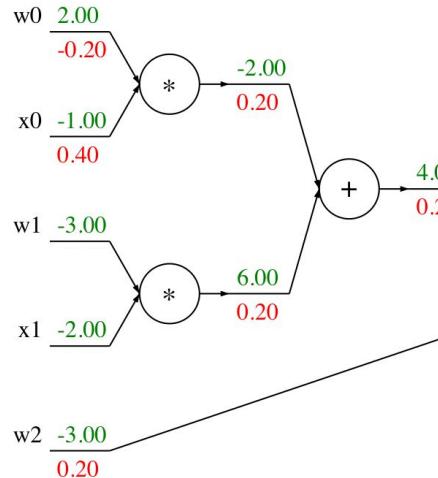
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

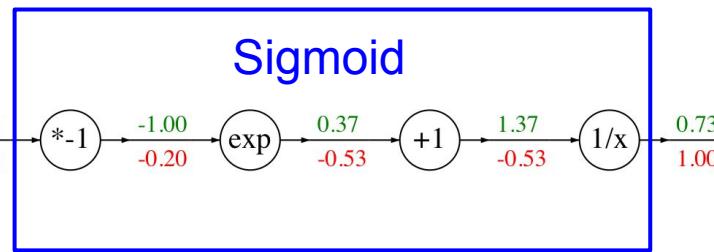
# Another example:

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



Sigmoid  
function

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

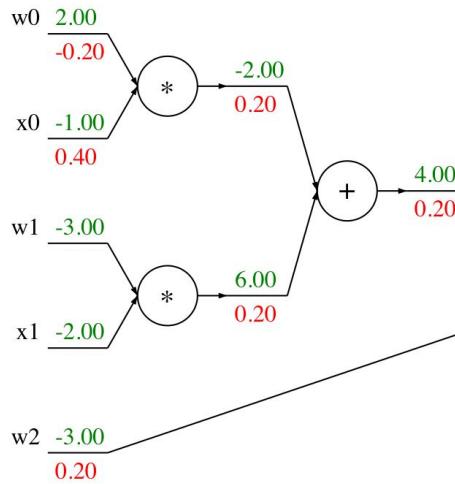


Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

When we're creating these computational graphs, we can define the computational nodes at any granularity that we want to. So in this case, we broke it down into the absolute simplest that we could. But in practice, we can group some of these nodes together into more complex nodes if we want -- as long as we're able to write down the 'local' gradient to that node.

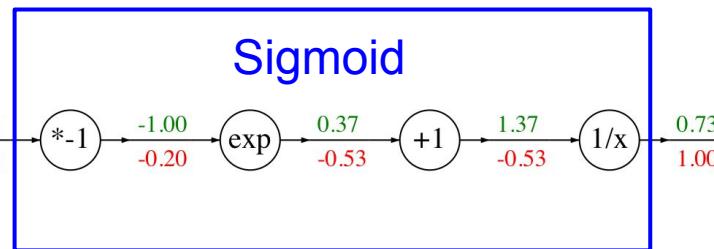
Another example:

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



Sigmoid  
function

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



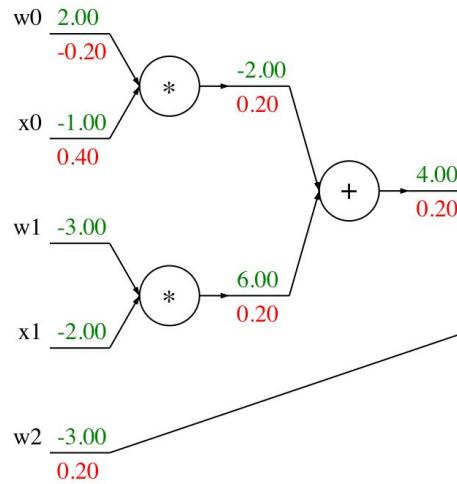
Sigmoid local  
gradient:

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

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

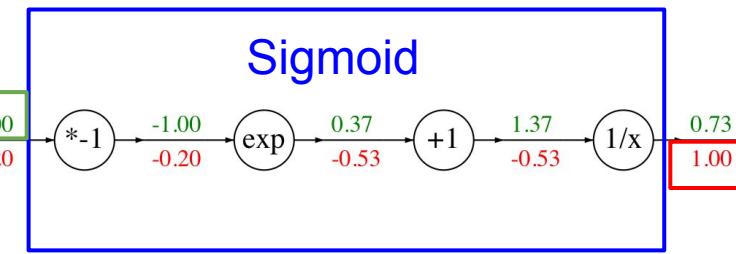
Another example:

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



Sigmoid  
function

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



Sigmoid

$$\begin{aligned} &[\text{upstream gradient}] \times [\text{local gradient}] \\ &[1.00] \times [(1 - 1/(1+e^1)) (1/(1+e^1))] = 0.2 \end{aligned}$$

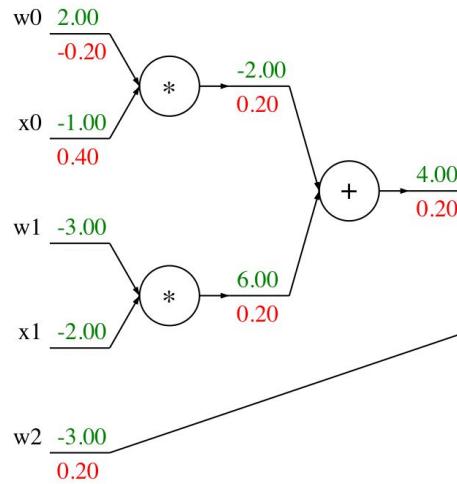
Sigmoid local  
gradient:

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

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

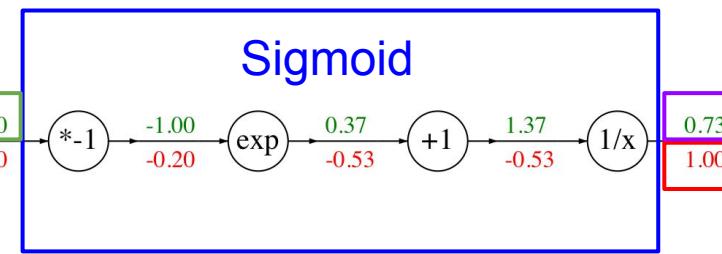
Another example:

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



Sigmoid  
function

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



[upstream gradient]  $\times$  [local gradient]  
 $[1.00] \times [(1 - 0.73)(0.73)] = 0.2$

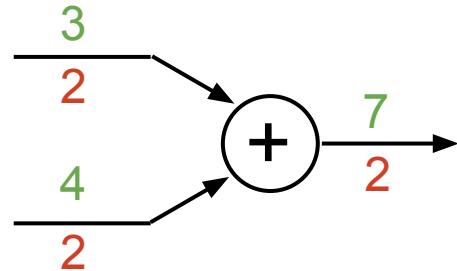
Sigmoid local  
gradient:

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

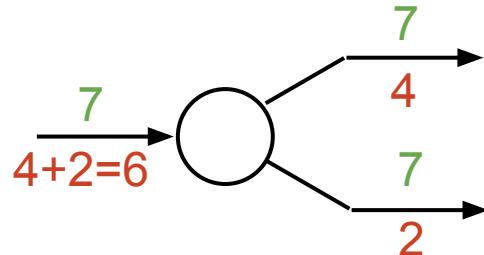
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

# Patterns in gradient flow

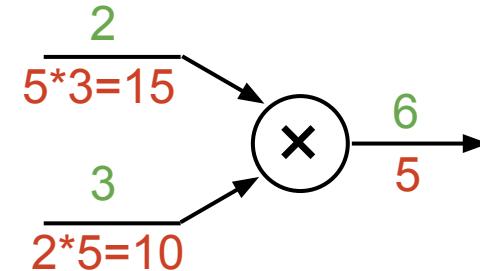
**add** gate: gradient distributor



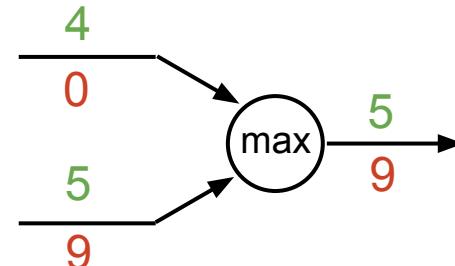
**copy** gate: gradient adder



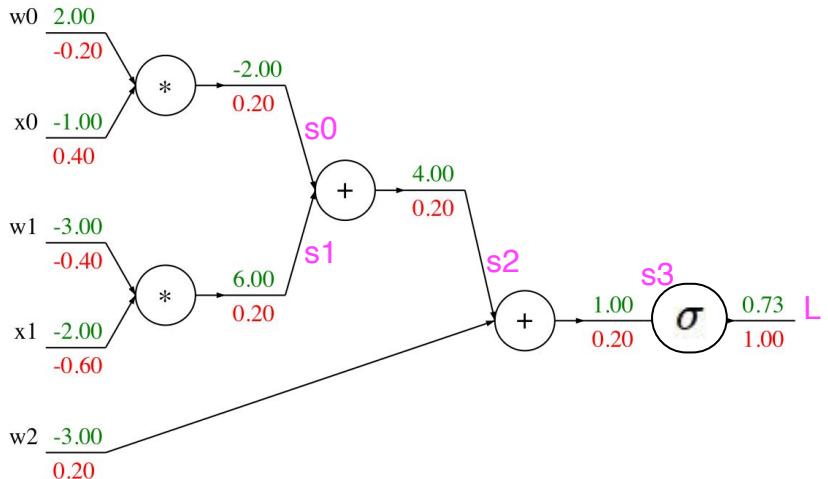
**mul** gate: “swap multiplier”



**max** gate: gradient router



# Backprop Implementation: “Flat” code



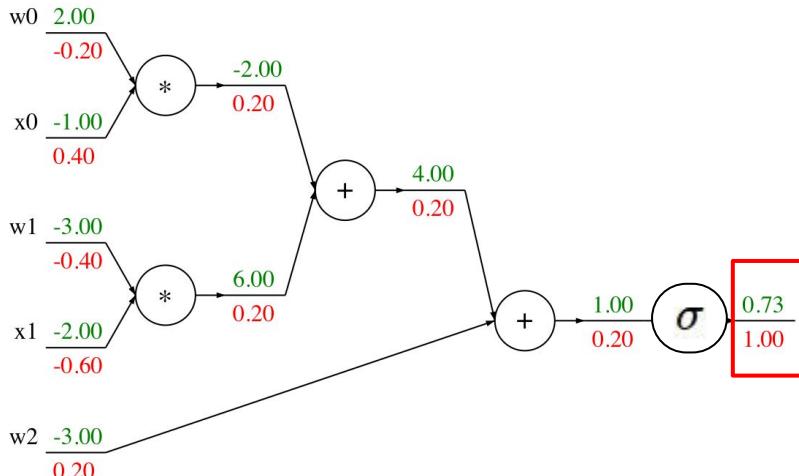
Forward pass:  
Compute output

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

Backward pass:  
Compute grads

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

# Backprop Implementation: “Flat” code



Forward pass:  
Compute output

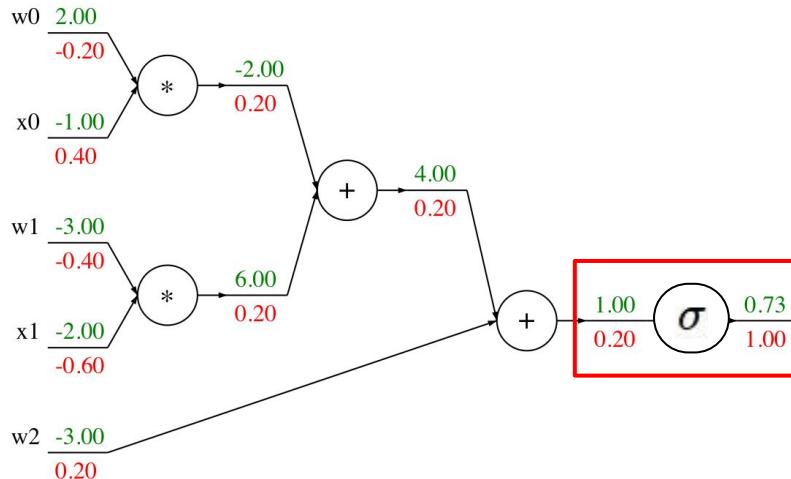
```
def f(w0, x0, w1, x1, w2):  
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

Base case

```
grad_L = 1.0
```

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

# Backprop Implementation: “Flat” code



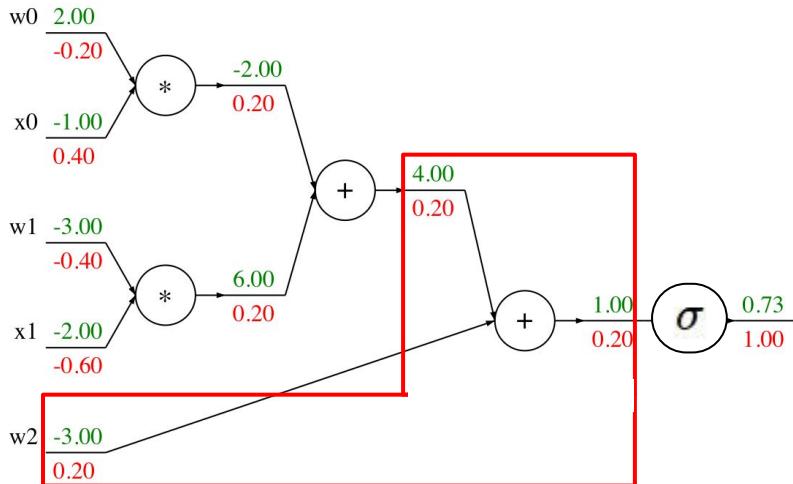
Forward pass:  
Compute output

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



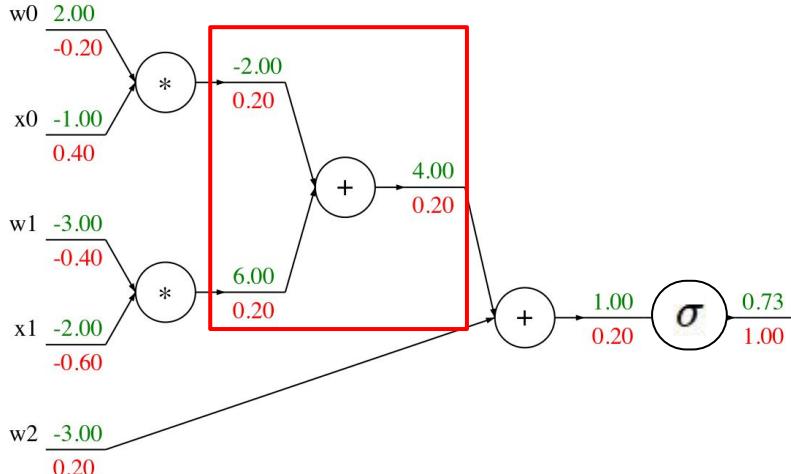
Forward pass:  
Compute output

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



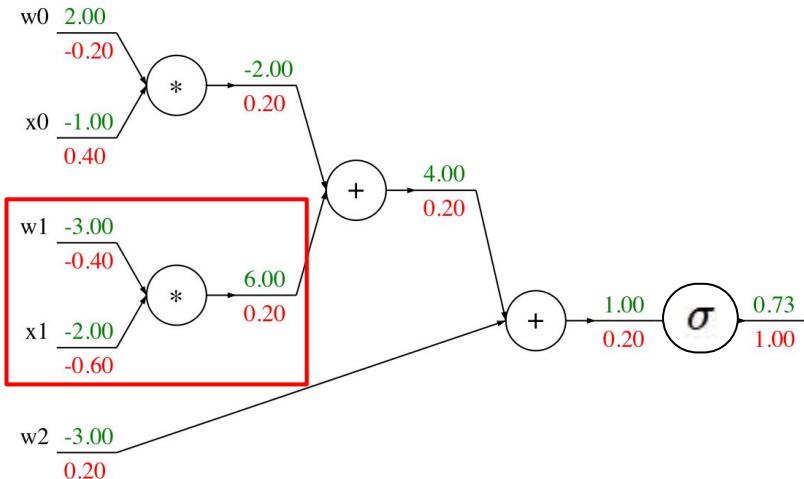
Forward pass:  
Compute output

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



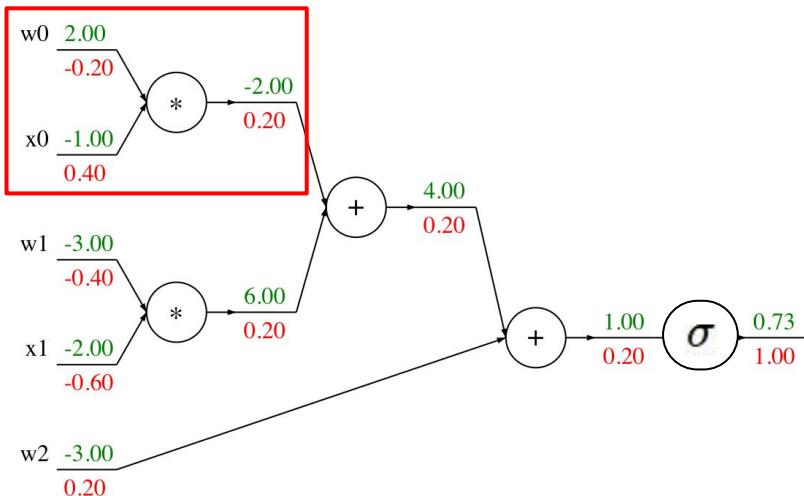
Forward pass:  
Compute output

```
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
```

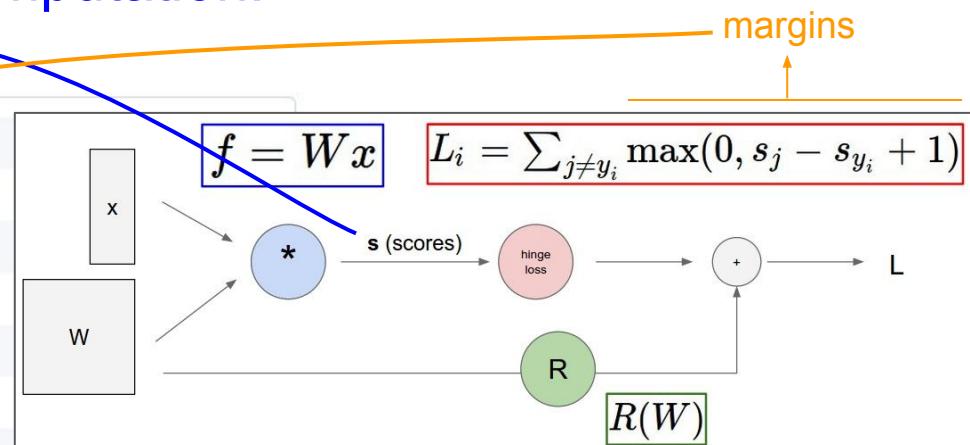
Multiply gate

# “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 6 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 = #...
```



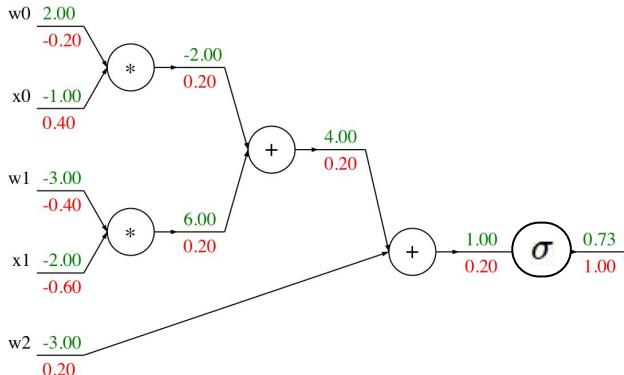
# “Flat” Backprop: Do this for assignment 1!

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*)

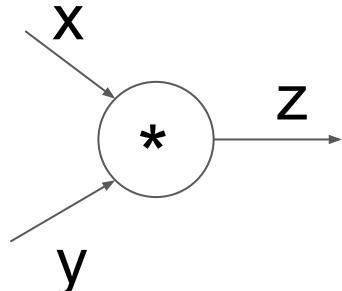


```
class ComputationalGraph(object):  
    ...  
    def forward(inputs):  
        # 1. [pass inputs to input gates...]  
        # 2. forward the computational graph:  
        for gate in self.graph.nodes_topologically_sorted():  
            gate.forward()  
        return loss # the final gate 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
```

Assignment 2 : FullyConnectedNets

# 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

Upstream  
gradient

Multiply upstream  
and local gradients

# Example: PyTorch operators

pytorch / pytorch		
Code	Issues 2,286	Pull requests 561
Tree: 517c7c9861 → <a href="#">pytorch / aten / src / THNN / generic /</a>	Create new file	Upload files
<a href="#">ezyang and facebook-github-bot Canonicalize all includes in PyTorch. (#14849)</a>	Latest commit 517c7c9 on Dec 8, 2018	
..		
<a href="#">AbsCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">BCECriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">ClassNLLCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Col2im.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">ELU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">FeatureLPPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">GatedLinearUnit.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">HardTanh.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Im2Col.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">IndexLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">LeakyReLU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">LogSigmoid.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MSECriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MultiLabelMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">MultiMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">RReLU.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Sigmoid.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SmoothL1Criterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftMarginCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftPlus.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SoftShrink.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SparseLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAdaptiveAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAdaptiveMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialClassNLLCriterion.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialConvolutionMM.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialDilatedMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialFractionalMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialFullDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialMaxUnpooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialReflectionPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialUpSamplingBilinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">SpatialUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">THNN.h</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">Tanh.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalReflectionPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalRowConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalUpSamplingLinear.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">TemporalUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAdaptiveAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAdaptiveMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricAveragePooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricConvolutionMM.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricDilatedMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricFractionalMaxPooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricFullDilatedConvolution.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricMaxUnpooling.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricReplicationPadding.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">VolumetricUpSamplingNearest.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago
<a href="#">linear_upsampling.h</a>	Implement nn.functional.interpolate based on upsample. (#8591)	9 months ago
<a href="#">pooling_shape.h</a>	Use integer math to compute output size of pooling operations (#14405)	4 months ago
<a href="#">unfold.c</a>	Canonicalize all includes in PyTorch. (#14849)	4 months ago

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19     THNN_CHECK_NELEMENT(output, gradOutput);
20     THTensor_(resizeAs)(gradInput, output);
21     TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22         scalar_t z = *output_data;
23         *gradInput_data = *gradOutput_data * (1. - z) * z;
24     );
25 }
26
27 #endif
```

Forward

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

[Source](#)

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19     THNN_CHECK_NELEMENT(output, gradOutput);
20     THTensor_(resizeAs)(gradInput, output);
21     TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22         scalar_t z = *output_data;
23         *gradInput_data = *gradOutput_data * (1. - z) * z;
24     );
25 }
26
27 #endif
```

Forward

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

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

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

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

[Source](#)

# PyTorch sigmoid layer

```
1 #ifndef TH_GENERIC_FILE
2 #define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
3 #else
4
5 void THNN_(Sigmoid_updateOutput)(
6     THNNState *state,
7     THTensor *input,
8     THTensor *output)
9 {
10    THTensor_(sigmoid)(output, input);
11 }
12
13 void THNN_(Sigmoid_updateGradInput)(
14     THNNState *state,
15     THTensor *gradOutput,
16     THTensor *gradInput,
17     THTensor *output)
18 {
19     THNN_CHECK_NELEMENT(output, gradOutput);
20     THTensor_(resizeAs)(gradInput, output);
21     TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
22         scalar_t z = *output_data;
23         *gradInput_data = *gradOutput_data * (1. - z) * z;
24     );
25 }
26
27 #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

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

[Source](#)

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?

## Vector to Scalar

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

Derivative is **Gradient**:

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

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

## Vector to Vector

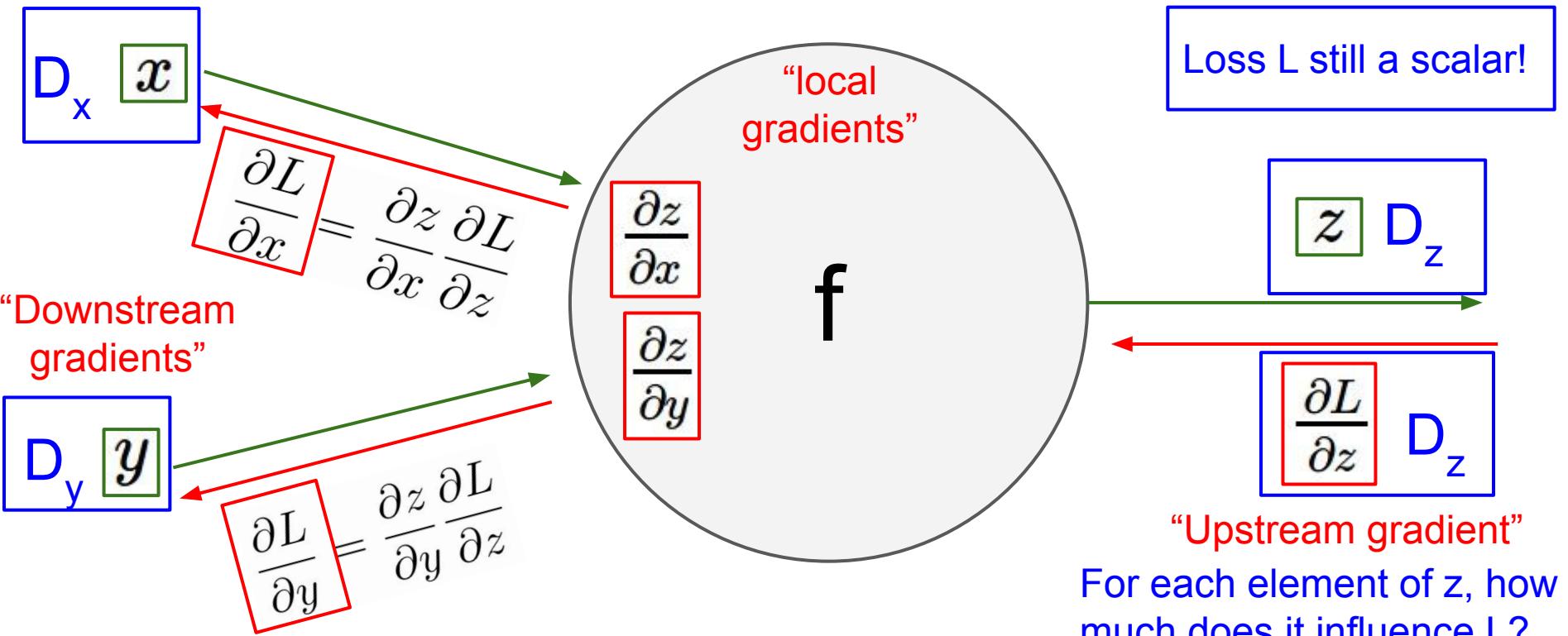
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

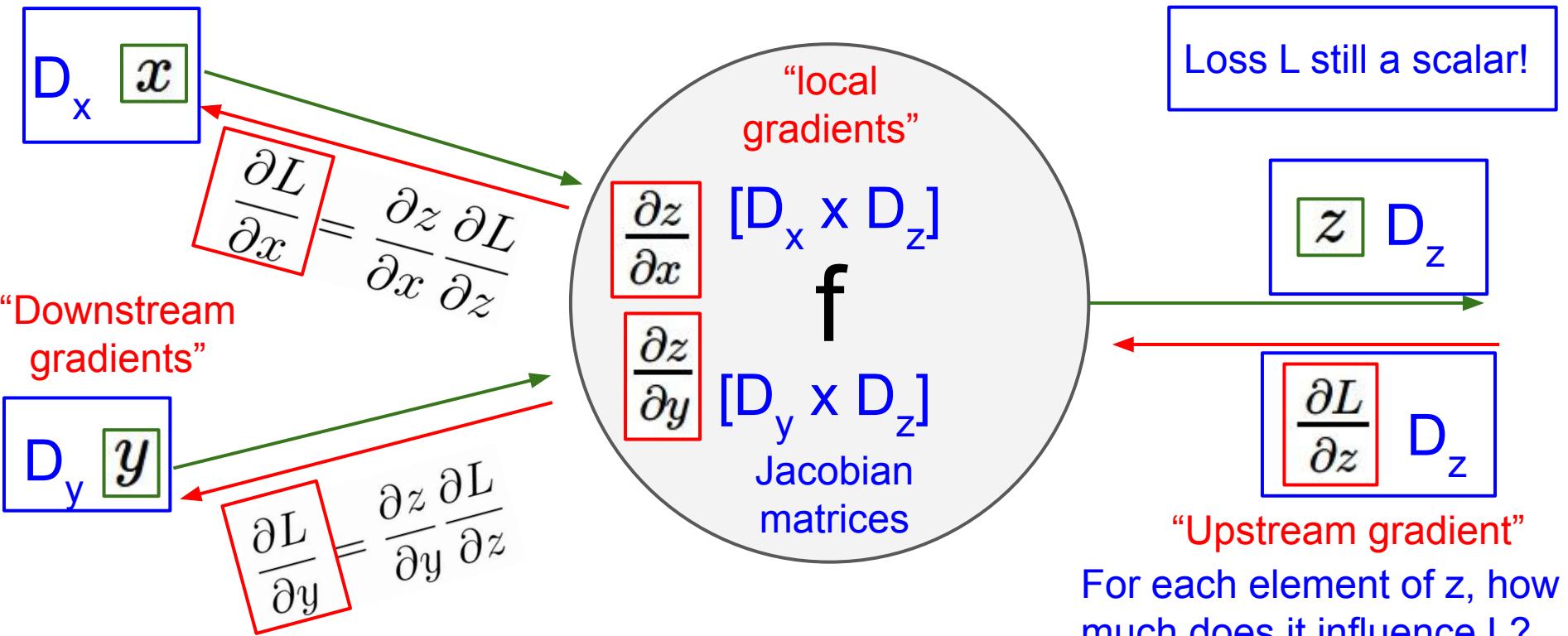
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

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

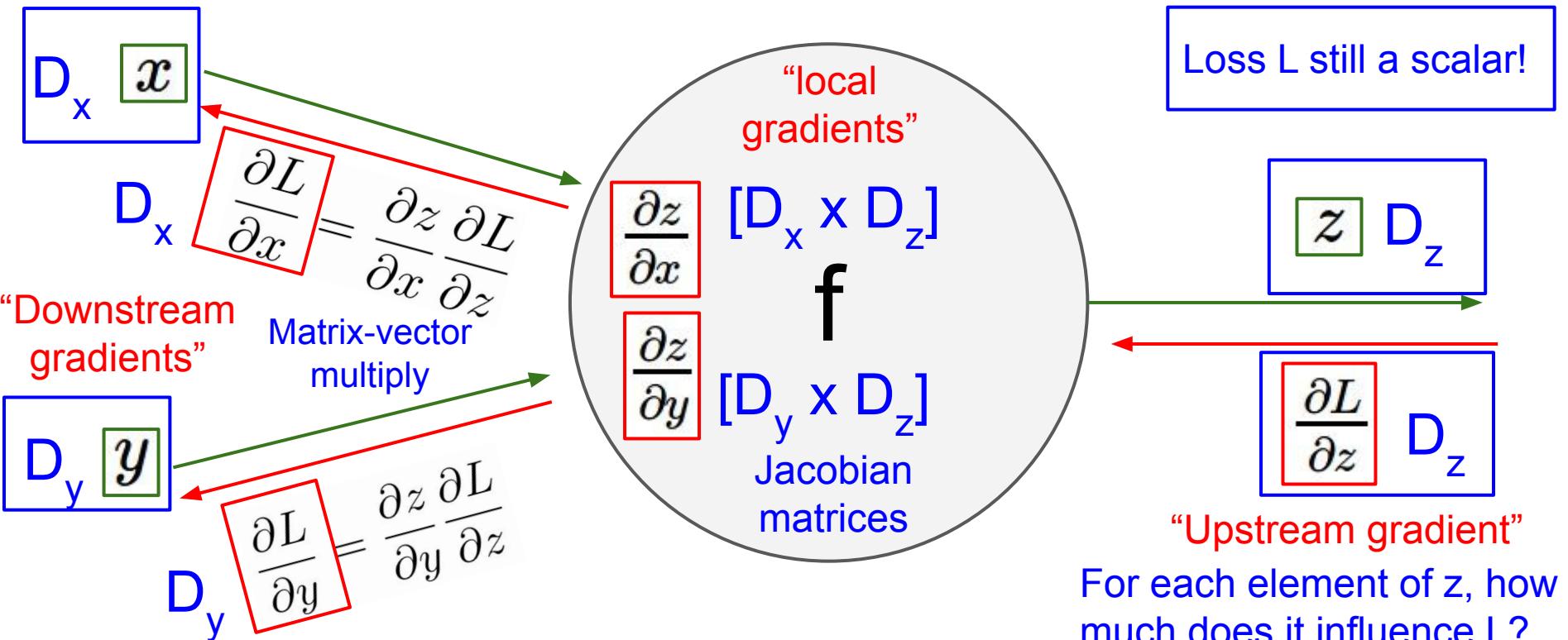
# Backprop with Vectors



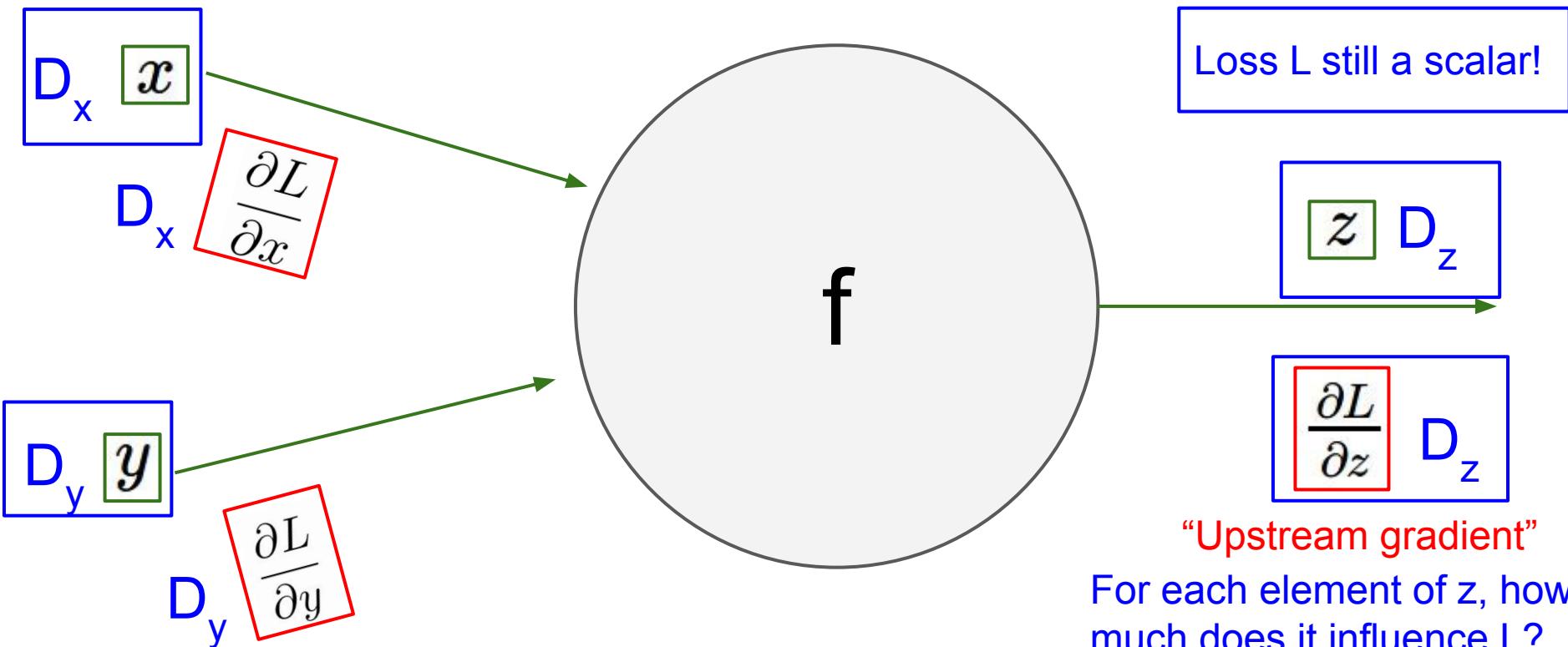
# Backprop with Vectors



# Backprop with Vectors



Gradients of variables wrt loss have same dims as the original variable



Loss L still a scalar!

$$z \quad D_z$$

$$\frac{\partial L}{\partial z} \quad D_z$$

“Upstream gradient”

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

# Backprop with Vectors

4D input  $x$ :

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\hspace{1cm}} \begin{array}{c} \text{f}(x) = \max(0, x) \\ (\text{elementwise}) \end{array}$$

4D output  $z$ :

$$\begin{array}{l} \xrightarrow{\hspace{1cm}} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix} \\ \xrightarrow{\hspace{1cm}} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 3 \end{bmatrix} \\ \xrightarrow{\hspace{1cm}} \begin{bmatrix} 3 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ \xrightarrow{\hspace{1cm}} \begin{bmatrix} 0 \\ 3 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

# Backprop with Vectors

4D input  $x$ :

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\quad} \begin{array}{c} f(x) = \max(0, x) \\ (\text{elementwise}) \end{array}$$

4D output  $z$ :

$$\begin{array}{l} \xrightarrow{\quad} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix} \\ \xrightarrow{\quad} \\ \xrightarrow{\quad} \\ \xrightarrow{\quad} \end{array}$$

Jacobian  $\frac{\partial z}{\partial x}$

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

4D  $\frac{\partial L}{\partial z}$ :

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

Upstream  
gradient

# Backprop with Vectors

4D input  $x$ :

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\quad} \begin{array}{c} f(x) = \max(0, x) \\ (\text{elementwise}) \end{array}$$

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

4D output  $z$ :

$$\begin{array}{c} \xrightarrow{\quad} [1] \\ \xrightarrow{\quad} [0] \\ \xrightarrow{\quad} [3] \\ \xrightarrow{\quad} [0] \end{array}$$

4D  $dL/dx$ :

$$\begin{array}{l} [4] \\ [0] \\ [5] \\ [0] \end{array} \xleftarrow{\quad} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

4D  $dL/dz$ :

$$\begin{array}{l} [4] \\ [-1] \\ [5] \\ [9] \end{array} \xleftarrow{\quad}$$

Upstream  
gradient

# Backprop with Vectors

4D input  $x$ :

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\quad} \begin{array}{c} f(x) = \max(0, x) \\ (\text{elementwise}) \end{array}$$

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

4D output  $z$ :

$$\begin{array}{l} \xrightarrow{\quad} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix} \end{array}$$

4D  $dL/dx$ :

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

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

$[dz/dx]$   $[dL/dz]$

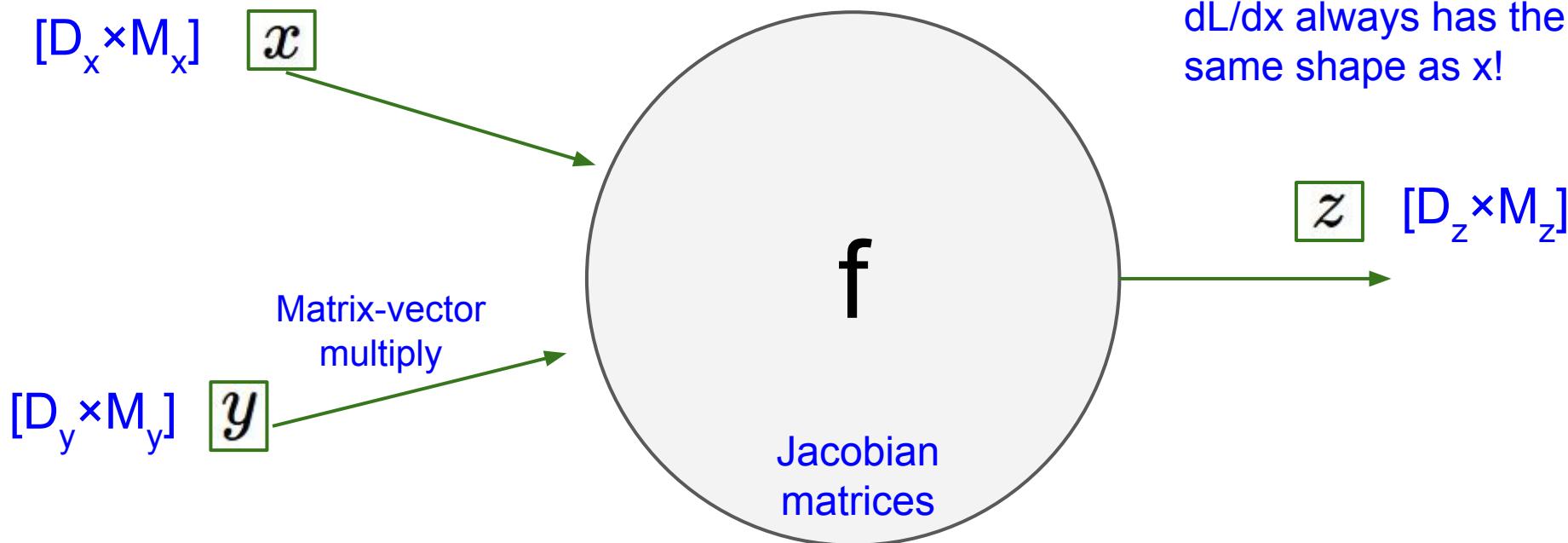
4D  $dL/dz$ :

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

Upstream  
gradient

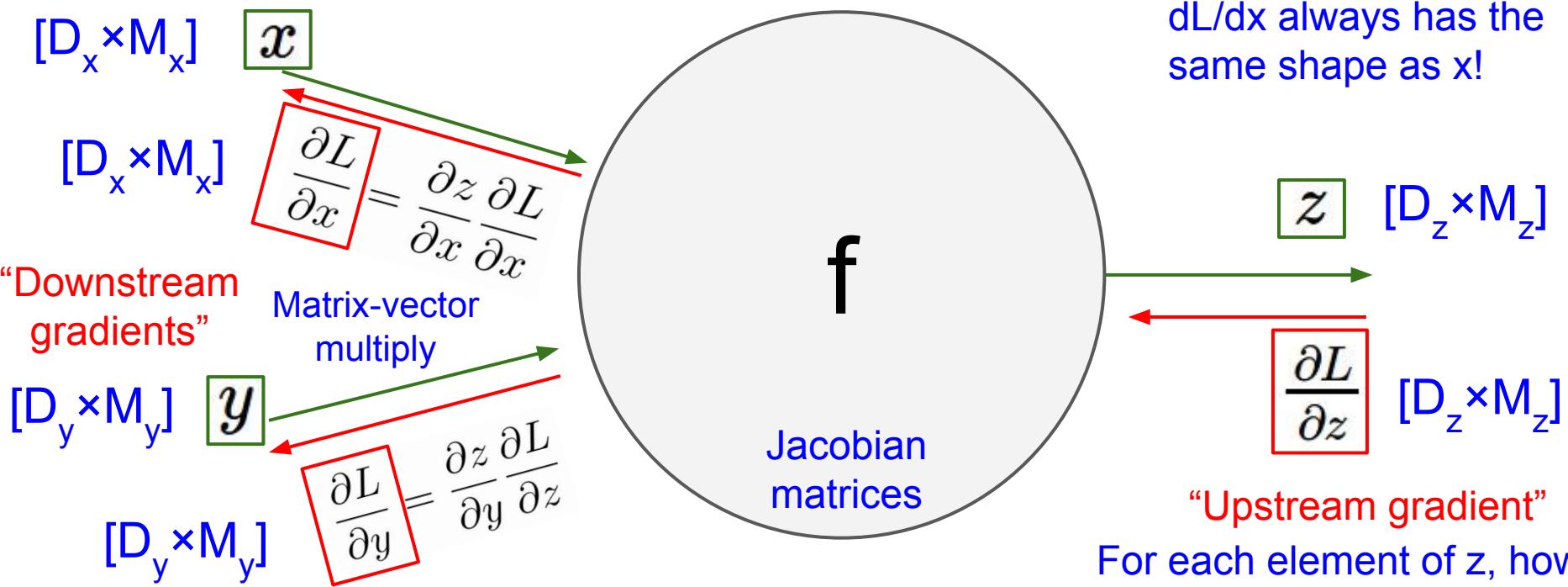
# Backprop with Matrices (or Tensors)

Loss L still a scalar!



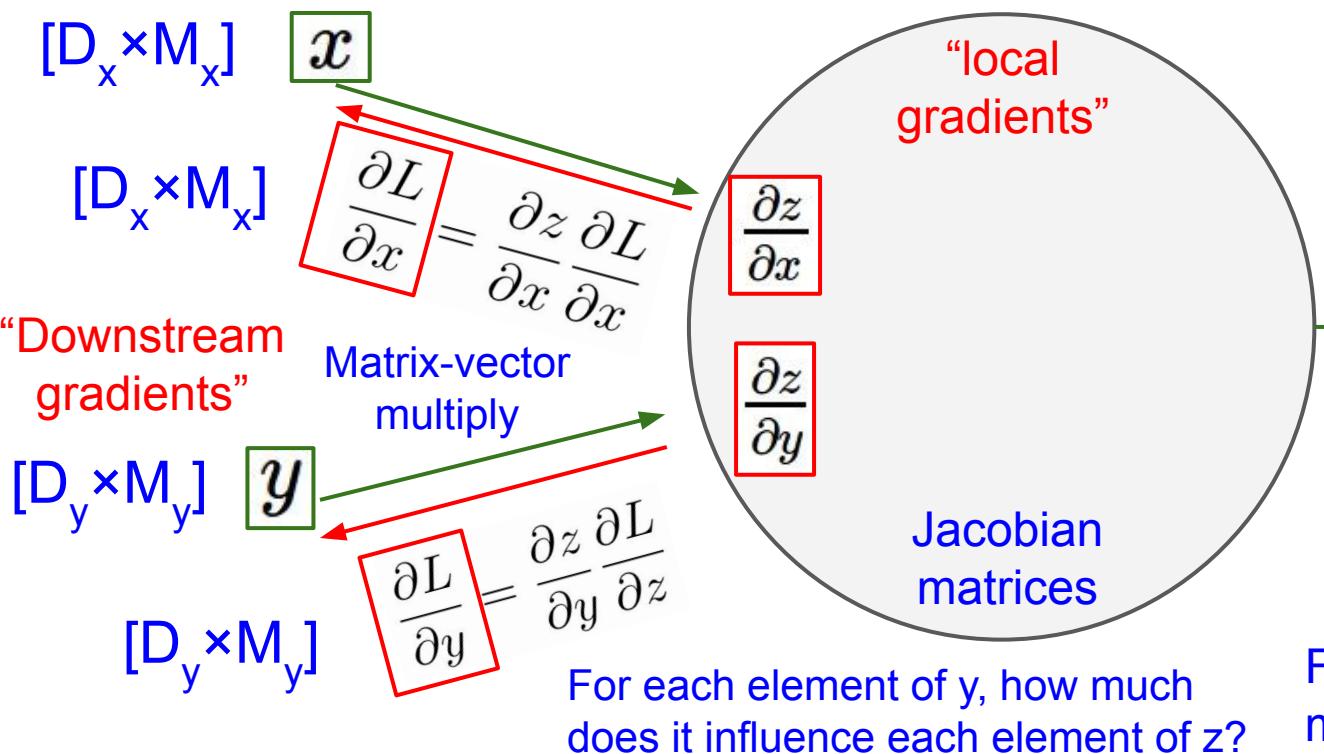
# Backprop with Matrices (or Tensors)

Loss L still a scalar!



# Backprop with Matrices (or Tensors)

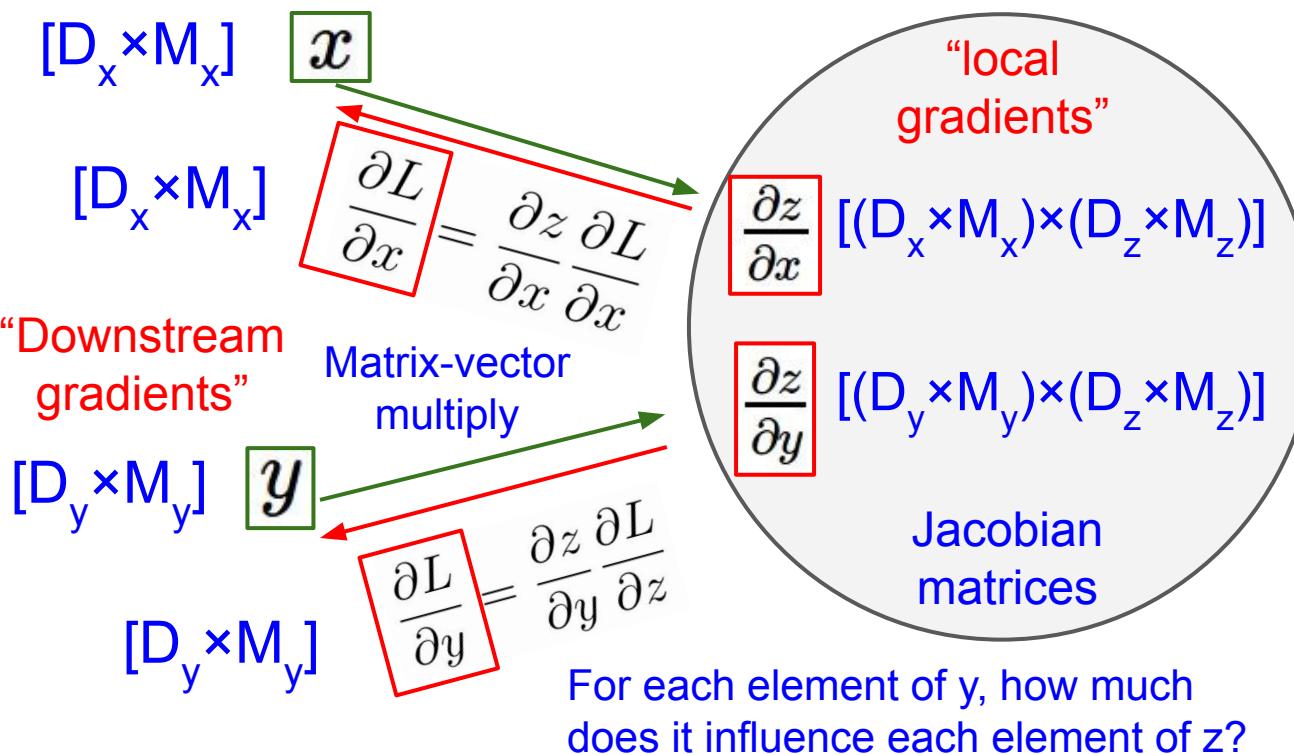
Loss L still a scalar!



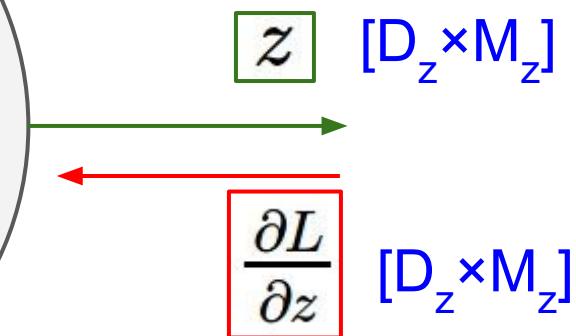
$dL/dx$  always has the same shape as  $x$ !

# Backprop with Matrices (or Tensors)

## Loss L still a scalar!



$dL/dx$  always has the same shape as  $x$ !



## “Upstream gradient”

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

# Backprop with Matrices

x: [N×D]

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

w: [D×M]

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

Matrix Multiply

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

y: [N×M]

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

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

Also see derivation in the course notes:

<http://cs231n.stanford.edu/handouts/linear-backprop.pdf>

# Backprop with Matrices

$x: [N \times D]$

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

$w: [D \times M]$

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

Matrix Multiply

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

Jacobians:

$$\begin{aligned} dy/dx &: [(N \times D) \times (N \times M)] \\ dy/dw &: [(D \times M) \times (N \times M)] \end{aligned}$$

$y: [N \times M]$

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

$dL/dy: [N \times M]$

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

For a neural net we may have

$$N=64, D=M=4096$$

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

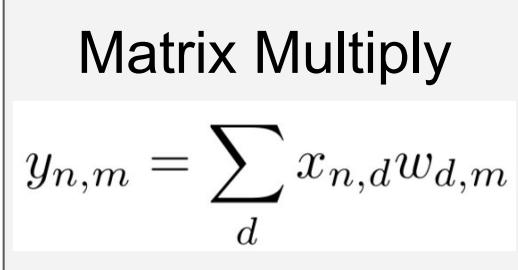
# Backprop with Matrices

$x: [N \times D]$

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

$w: [D \times M]$

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



$y: [N \times M]$

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

$dL/dy: [N \times M]$

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

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

**A:**  $x_{n,d}$  affects 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]

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

w: [D×M]

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

[N×D] [N×M] [M×D]

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

Matrix Multiply

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

**Q:** What parts of y are affected by one element of x?  
**A:**  $x_{n,d}$  affects the whole row  $y_{n,:}$ .

**Q:** How much does  $x_{n,d}$  affect  $y_{n,m}$ ?  
**A:**  $w_{d,m}$

y: [N×M]

$$\begin{bmatrix} 13 & 9 & \boxed{-2} & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

dL/dy: [N×M]

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

# Backprop with Matrices

$x: [N \times D]$

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

$w: [D \times M]$

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

Matrix Multiply

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

$y: [N \times M]$

$$\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$$

$dL/dy: [N \times M]$

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

By similar logic:

$[N \times D] \quad [N \times M] \quad [M \times D]$

$[D \times M] \quad [D \times N] \quad [N \times M]$

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

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

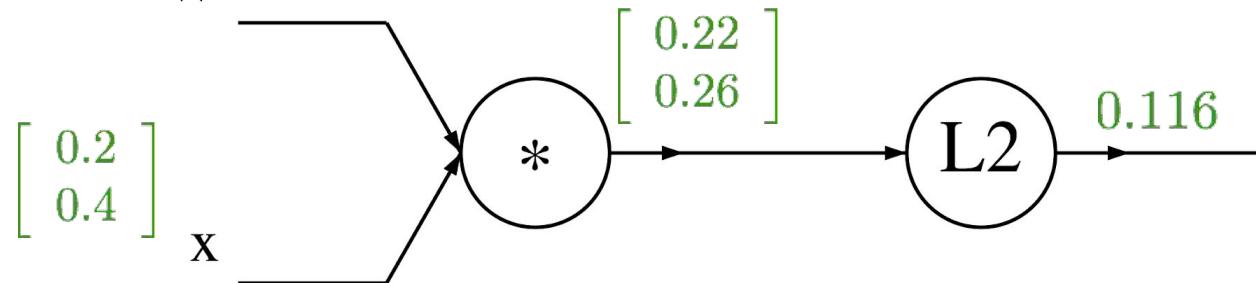
These formulas are easy to remember: they are the only way to make shapes match up!

# Summary for today:

- (**Fully-connected**) **Neural Networks** are 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
- 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

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$

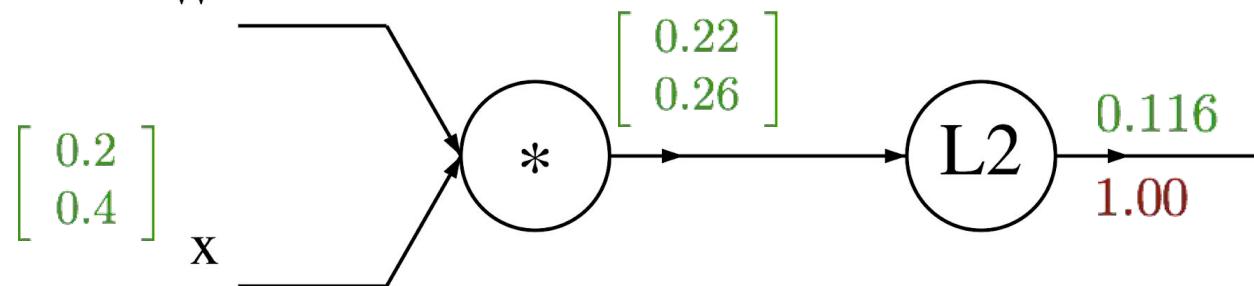


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$

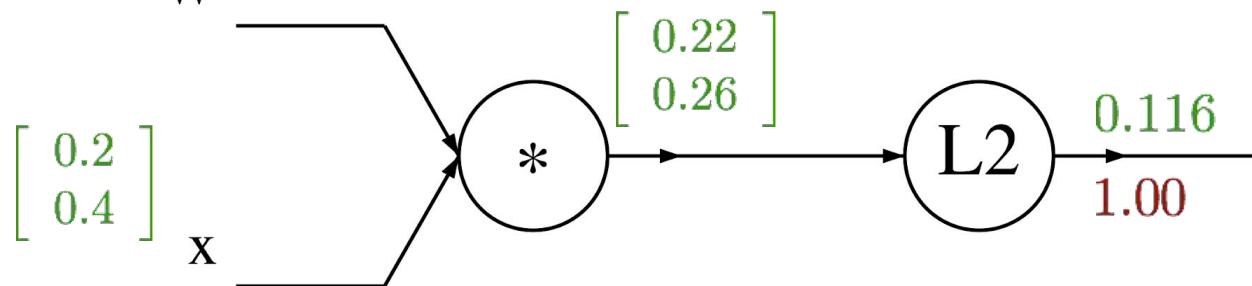


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

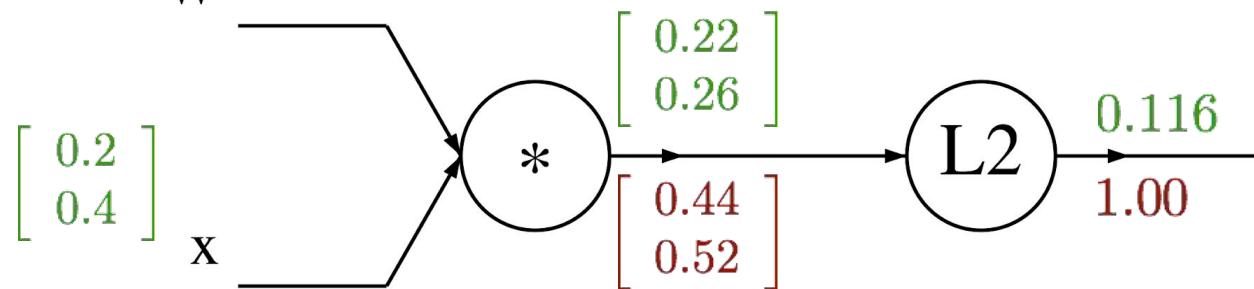
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\nabla_q f = 2q$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

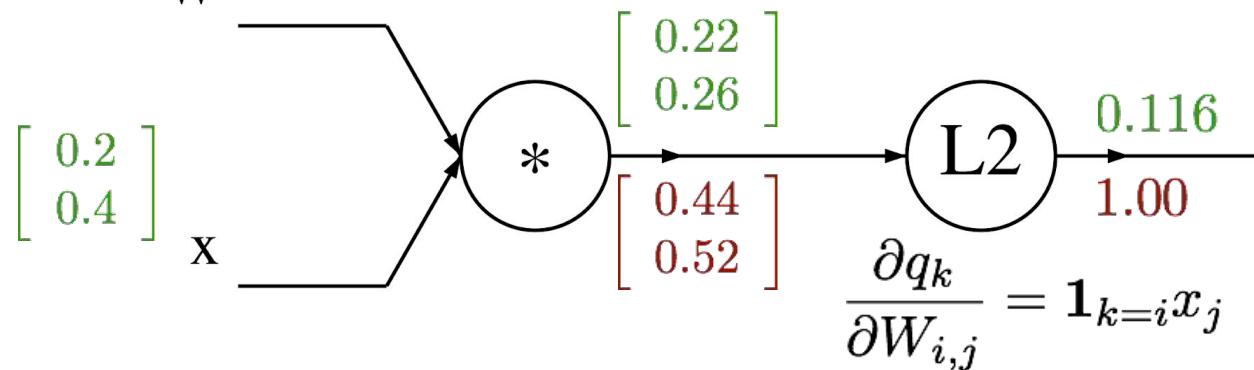
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$

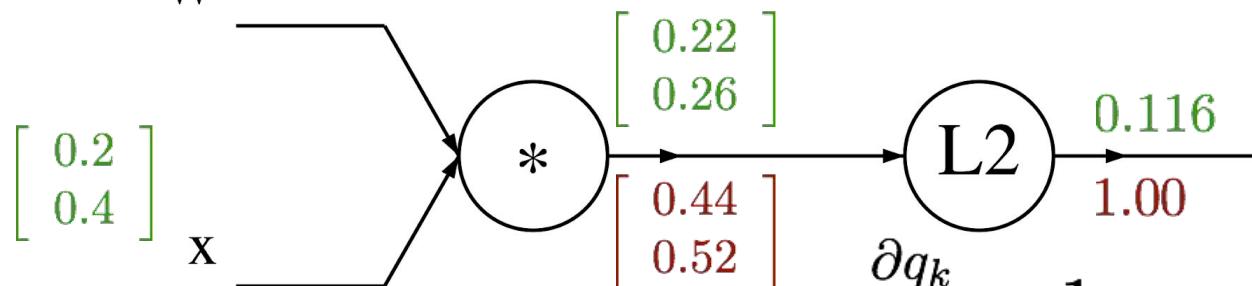


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

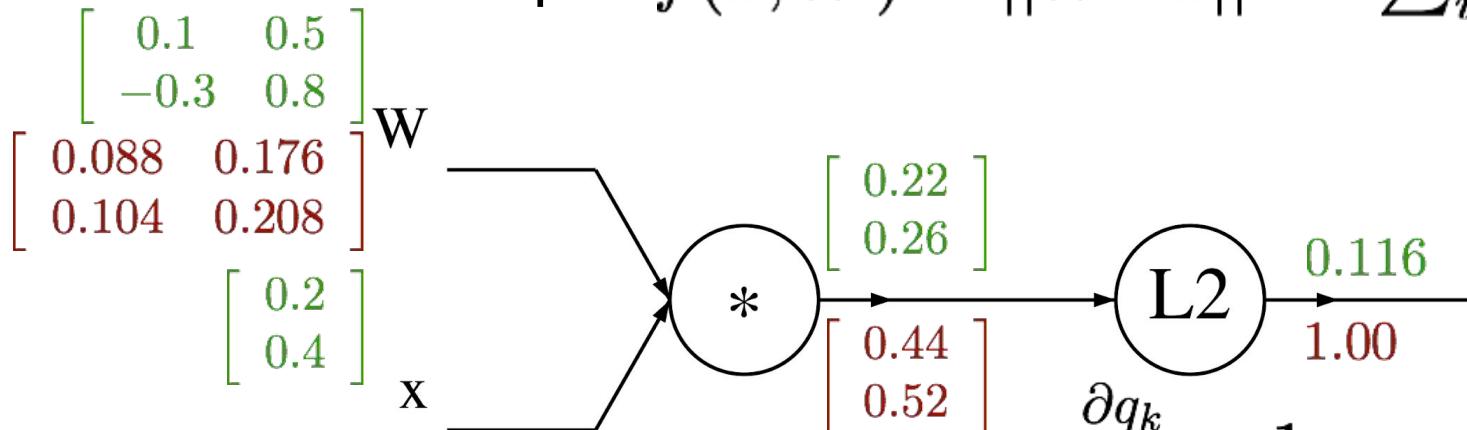
$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$



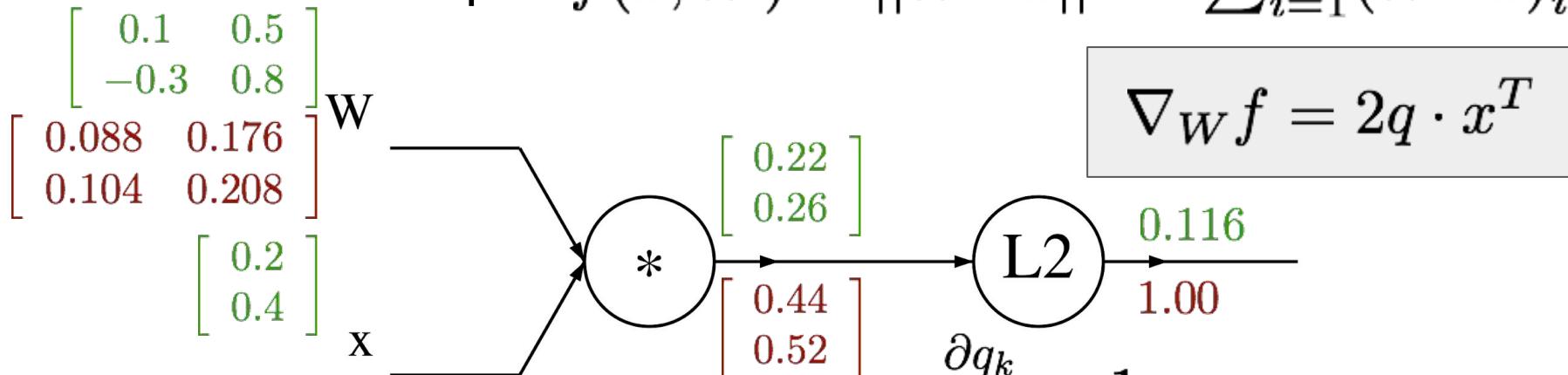
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

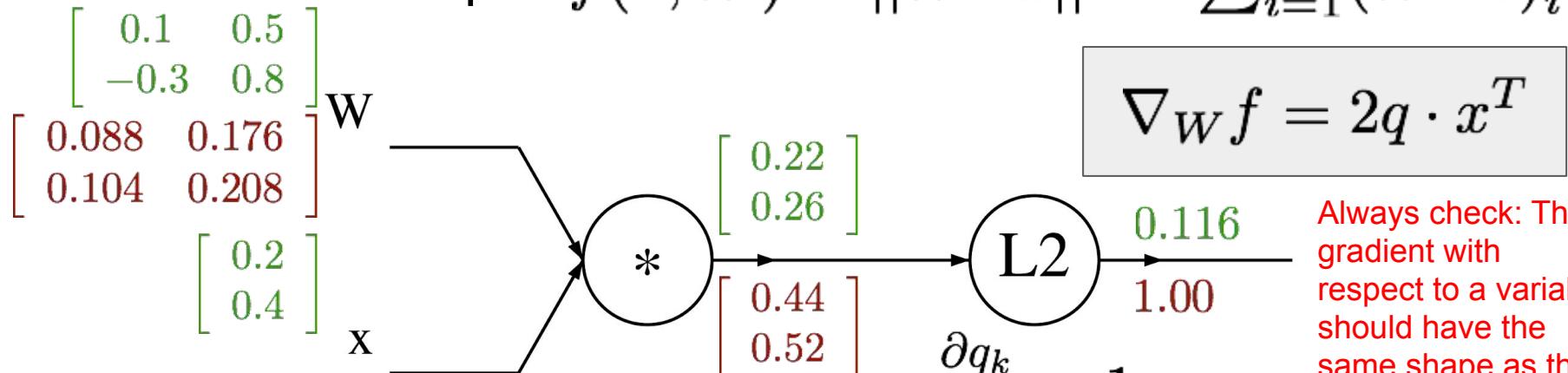
$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k) (\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$



$$\nabla_W f = 2q \cdot x^T$$

Always check: The gradient with respect to a variable should have the same shape as the variable

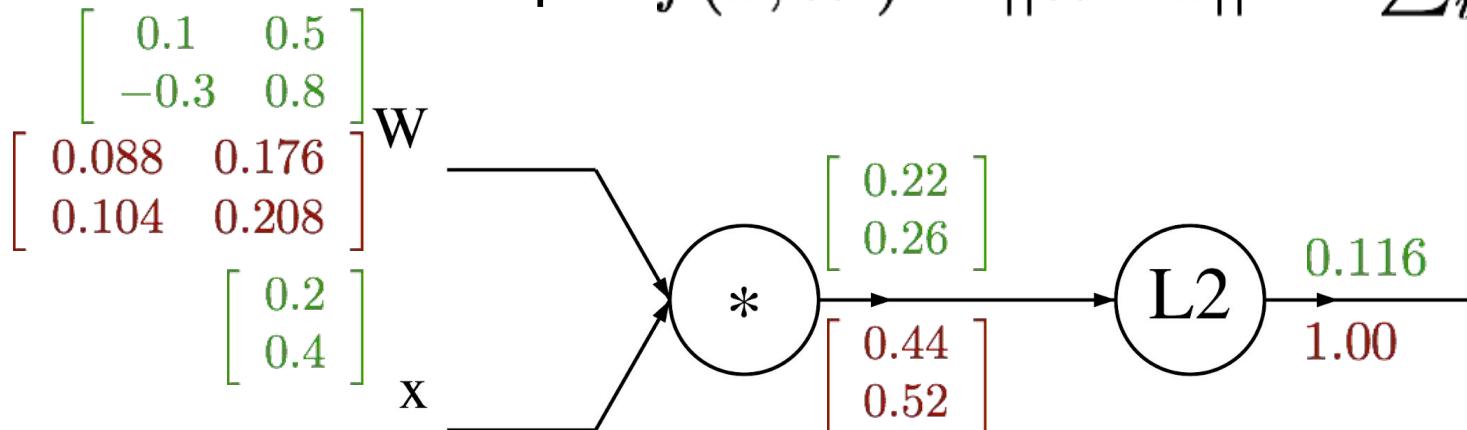
$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned}\frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j\end{aligned}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

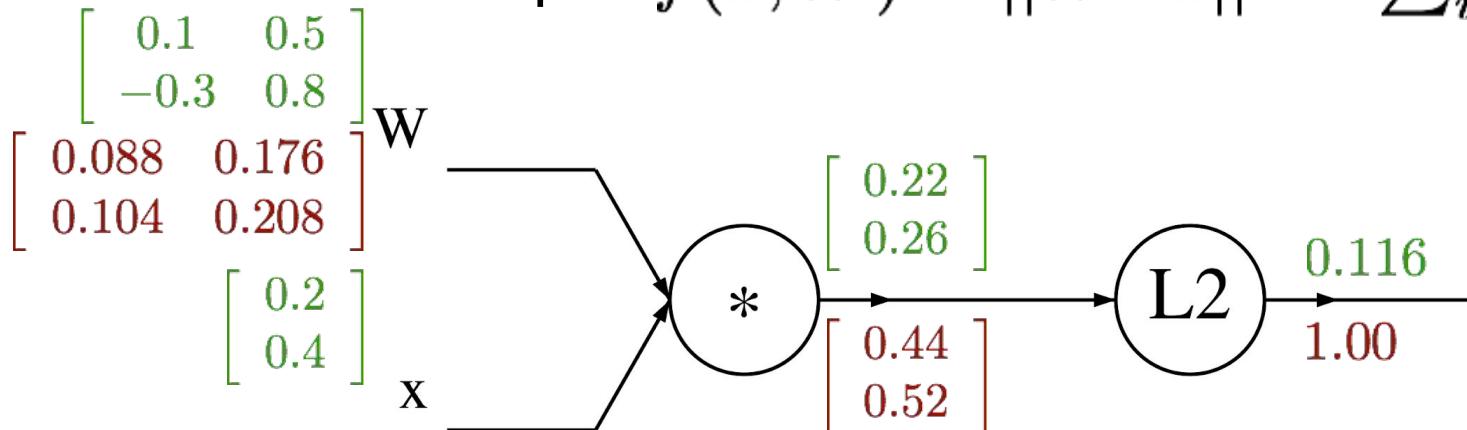


$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$

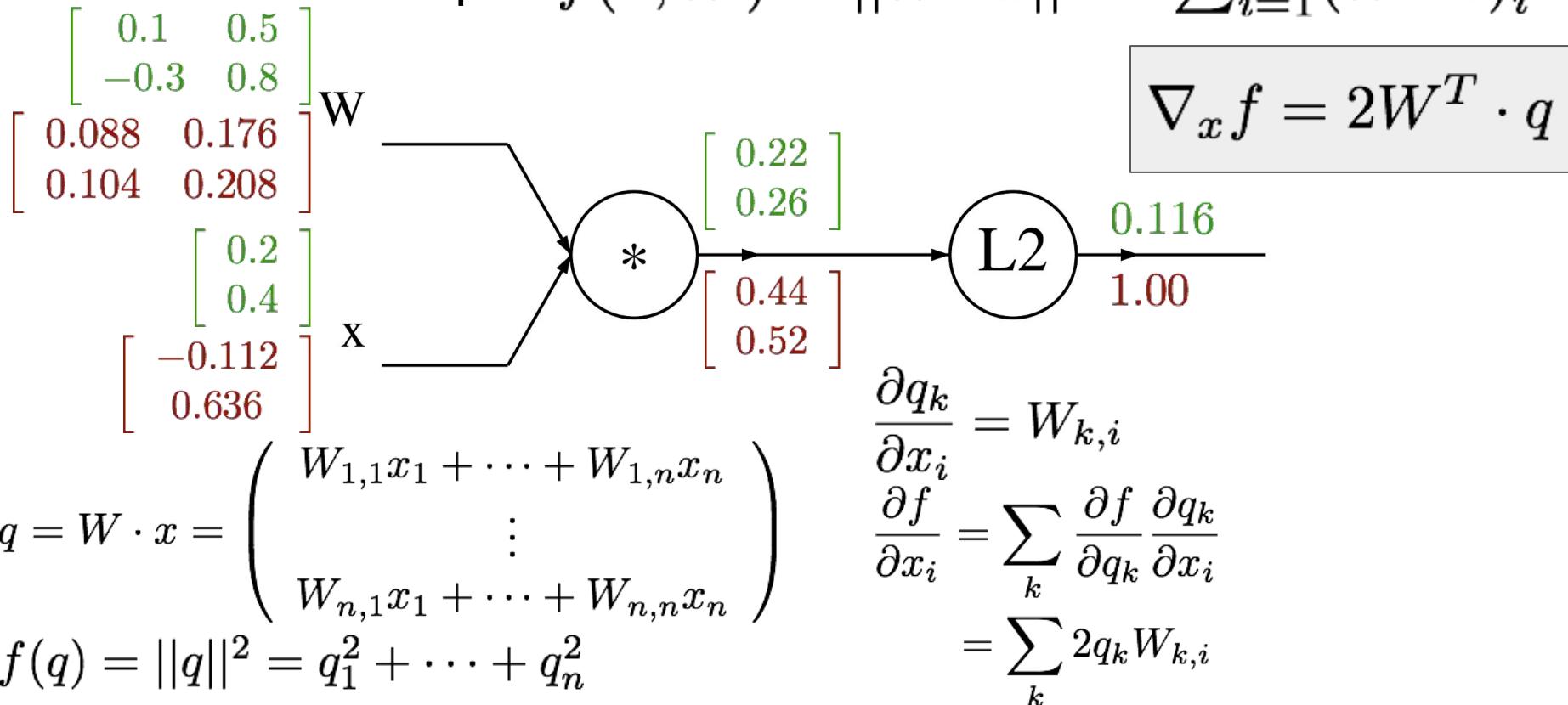


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\begin{aligned}\frac{\partial q_k}{\partial x_i} &= W_{k,i} \\ \frac{\partial f}{\partial x_i} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} \\ &= \sum_k 2q_k W_{k,i}\end{aligned}$$

A vectorized example:  $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^n (W \cdot x)_i^2$



$$\begin{aligned}\frac{\partial q_k}{\partial x_i} &= W_{k,i} \\ \frac{\partial f}{\partial x_i} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} \\ &= \sum_k 2q_k W_{k,i}\end{aligned}$$

# In discussion section: A matrix example...

$$z_1 = XW_1$$

$$h_1 = \text{ReLU}(z_1)$$

$$\hat{y} = h_1 W_2$$

$$L = \|\hat{y}\|_2^2$$

$$\frac{\partial L}{\partial W_2} = ?$$

$$\frac{\partial L}{\partial W_1} = ?$$

