

# Neural Networks

## Part I: Multilayer Perceptron



SAPIENZA  
UNIVERSITÀ DI ROMA

# Logistic regression as a linear classifier

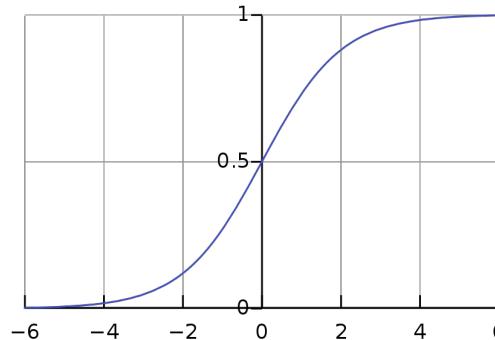


SAPIENZA  
UNIVERSITÀ DI ROMA

# Linear classifier

**Don't be confused by the sigmoid function: the Logistic Regression is a linear classifier!**

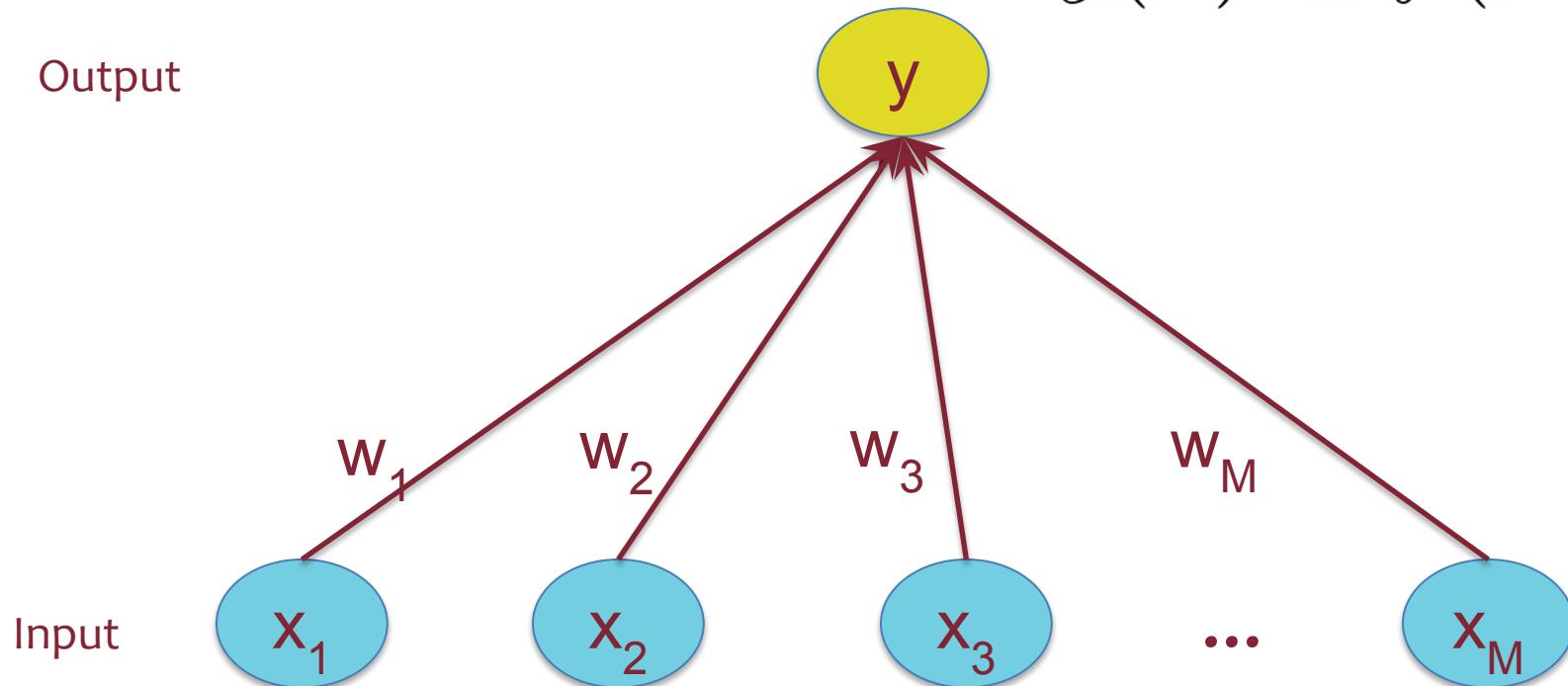
$$P(y|x) = \frac{e^{W^T x}}{1 + e^{W^T x}} > 0.5 \quad \rightarrow \quad W^T x > 0$$



# Linear Regression

$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x})$$

Output

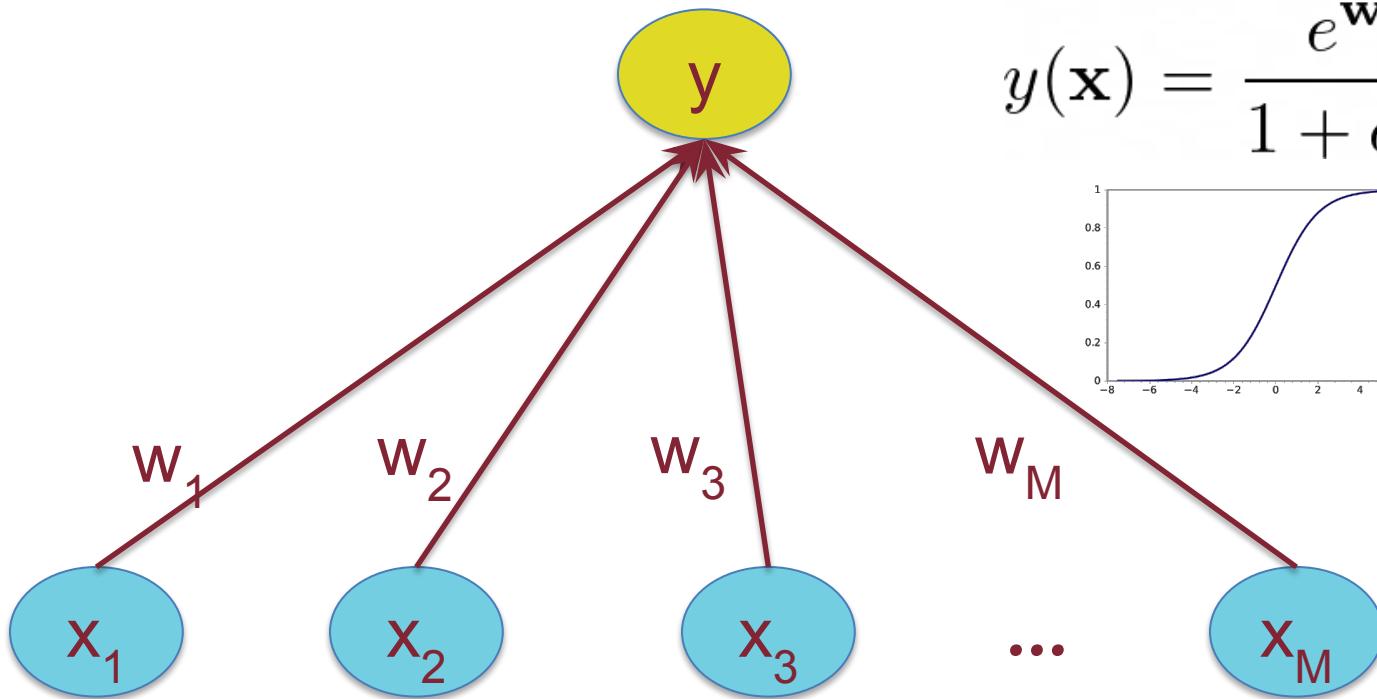


## Decision Functions

# Logistic Regression

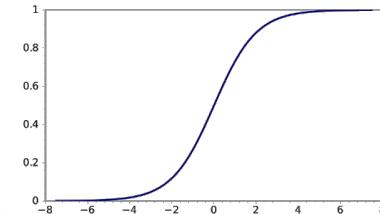
Output

Input



$$y(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

$$y(\mathbf{x}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}}$$



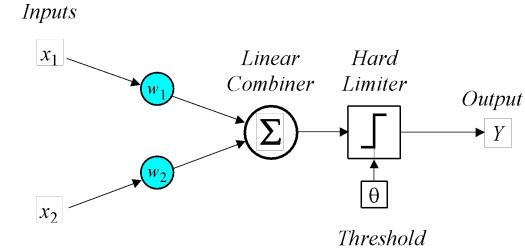
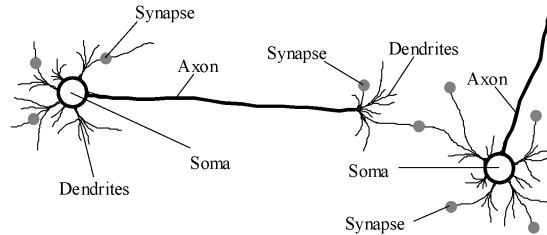
# Perceptron



SAPIENZA  
UNIVERSITÀ DI ROMA

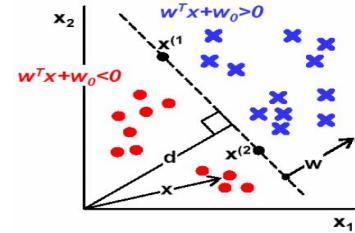
# Perceptron and Neural Nets

- From biological neuron to artificial neuron (perceptron)

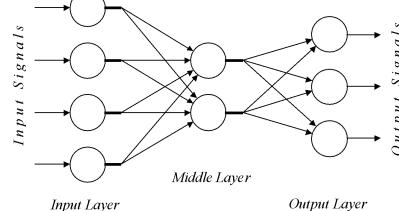


- Activation function

$$X = \sum_{i=1}^n x_i w_i \quad Y = \begin{cases} +1, & \text{if } X \geq \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases}$$



- Artificial neuron networks
  - supervised learning
  - gradient descent



# Perceptron algorithm - Rosenblatt (1962)

To stick with Bishop's notation, we consider a nonlinear transformation of the features (feature expansion):

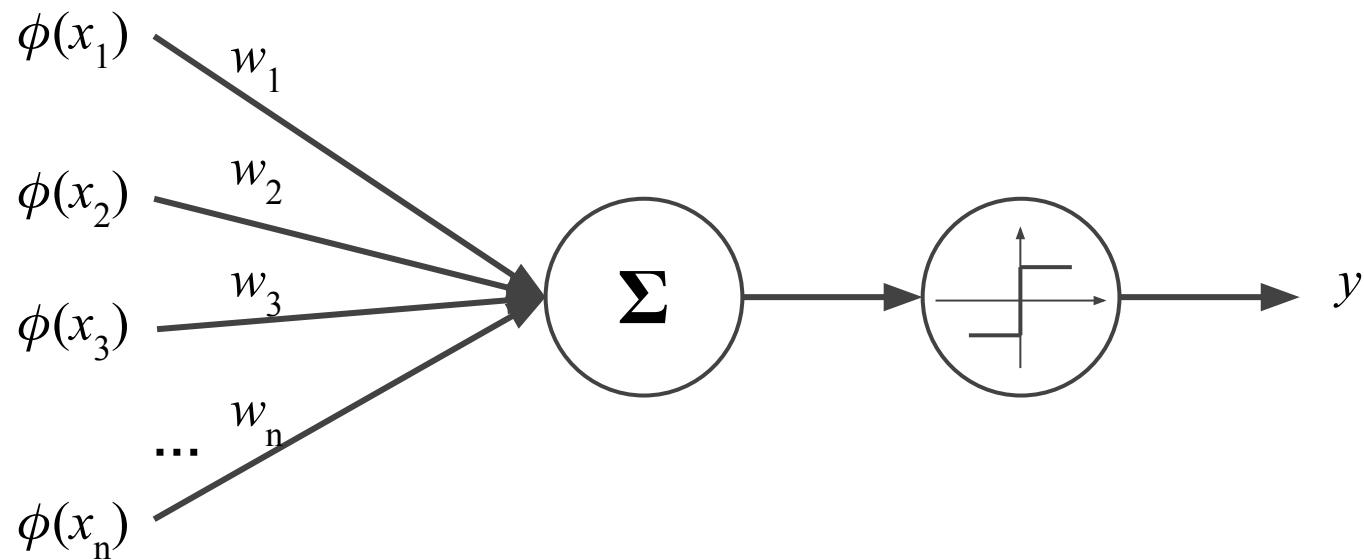
$$\phi(\mathbf{x})$$

These are  
called “basis  
functions”

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$

# Perceptron



# From Bishop's book



## Frank Rosenblatt

1928–1969

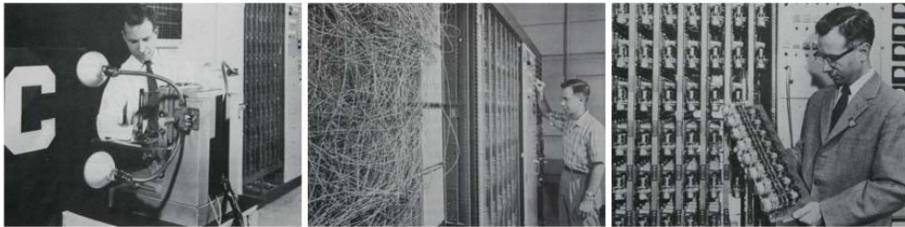
Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minsky, whose objections were published in the book "Perceptrons", co-authored with

Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

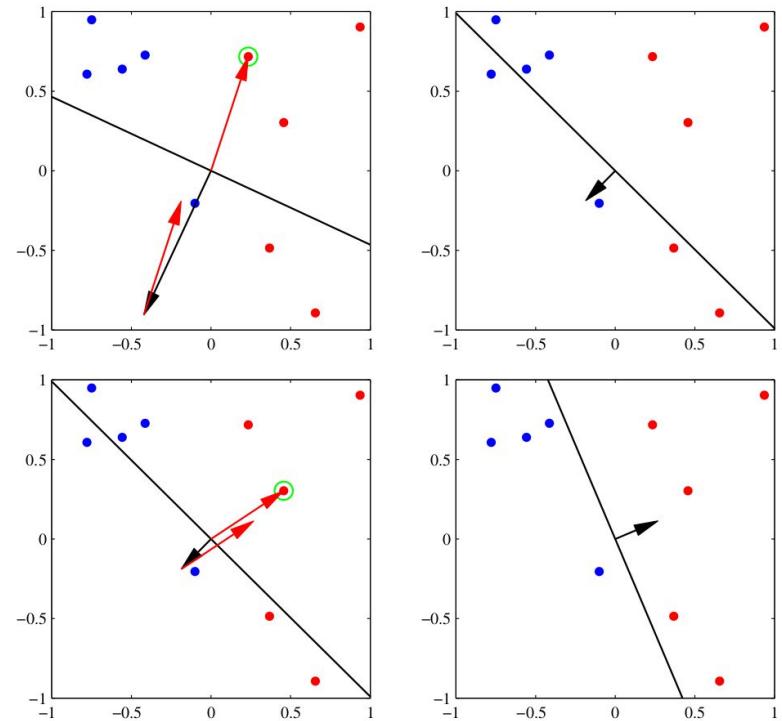
# Perceptron algorithm

$$E_P(\mathbf{w}) = - \sum_{n \in \mathcal{M}} \mathbf{w}^T \phi_n t_n$$

$\mathcal{M}$  is the set of misclassified examples



**Figure 4.8** Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focussed onto a  $20 \times 20$  array of cadmium sulphide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.



# Limitations

- Finds solutions only for linearly separable problems
- It is hard to distinguish between linearly separable with slow convergence and non separable problems
- For linearly separable problems there are multiple valid solutions
- There is no measure of probability or uncertainty in the classification
- Does not generalize to more than two classes

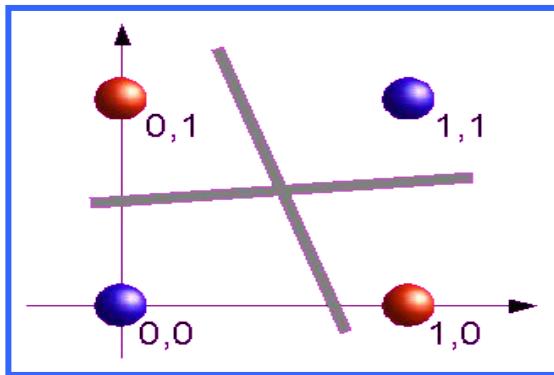
# Multi layer perceptron

FROM CS231n 2022 Course slides  
<http://cs231n.stanford.edu/schedule.html>

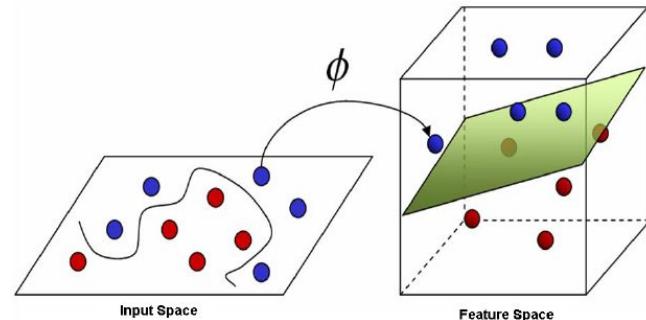


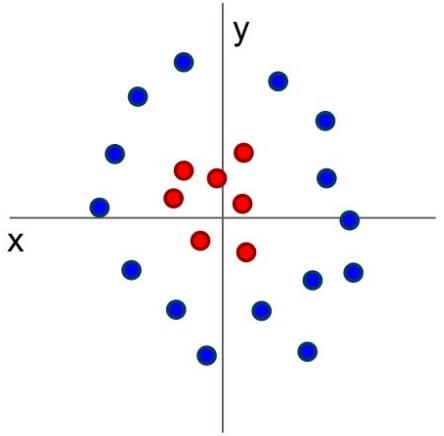
# Learning highly non-linear functions

One example of non linearly separable problem is the XOR function, but many other exist:

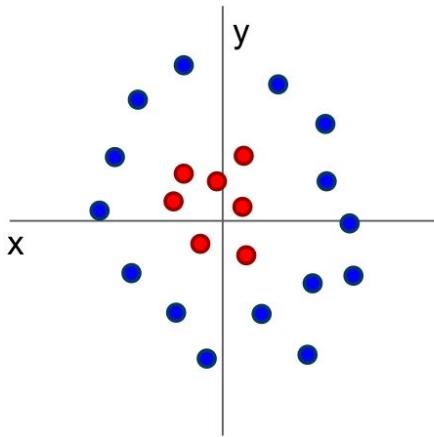


Learning a transformation of the features seems effective. How do we do it automatically?



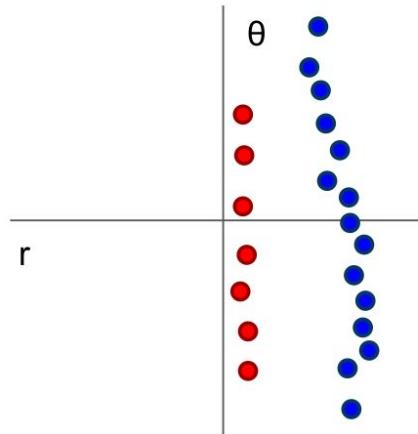


Cannot separate red  
and blue points with  
linear classifier



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

# Addressing the limitations

Clearly, the perceptron algorithm is a son of his time. As the ML knowledge advanced, the classification of non separable classes found different solutions (we saw the logistic regression, but many others exist).

However, the research on an electronic equivalent of a brainlike neuron network stopped for a while, until the '80s, when a brief resurgence happened.

The main problem with the perceptron algorithm is that it has only one layer. It is possible to formally show that using at least two layers the “multilayer perceptron” is a universal function approximator.

# Generalize the feature expansion process

We saw that a linear model is often depicted having a “generalized basis functionS” input, instead of having directly the data features. We are now asking how to make the definition of these functions learnable, i.e. dependant on trainable parameters:

# Let's start over: Linear classifier

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

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

# Neural Networks: 2 layers

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

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

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

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

**(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)

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

**(Now)** 2-layer Neural Network       $f = W_2 \max(0, W_1 x)$   
or 3-layer Neural Network

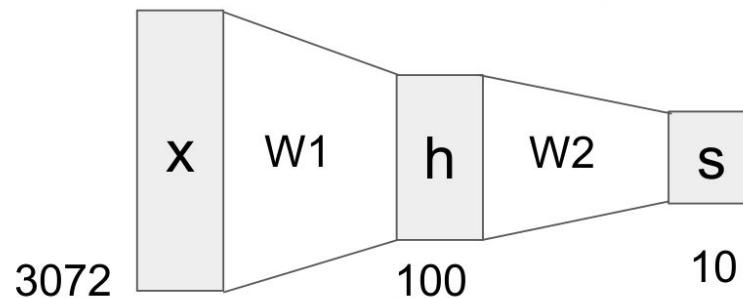
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

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

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

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

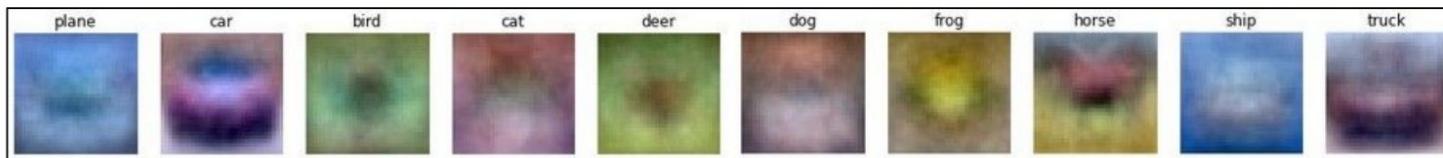
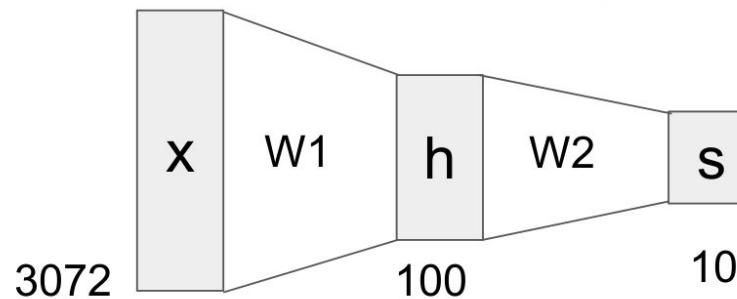
**(Now)** 2-layer Neural Network  $f = W_2 \max(0, W_1 x)$



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

**(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

(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**.

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

$$f = W_2 W_1 x$$

**(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**.

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

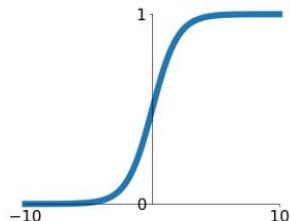
$$f = W_2 W_1 x \quad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

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

# Activation functions

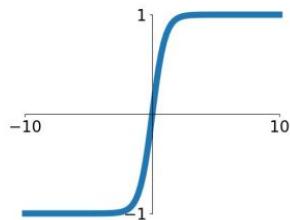
## Sigmoid

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



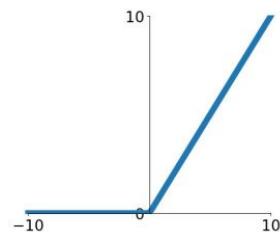
## tanh

$$\tanh(x)$$



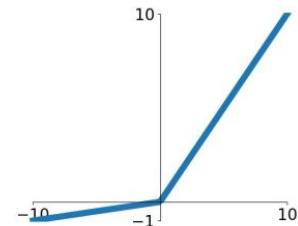
## ReLU

$$\max(0, x)$$



## Leaky ReLU

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

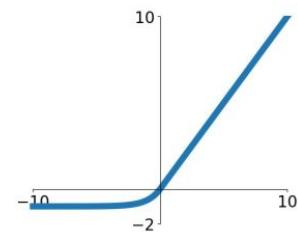


## Maxout

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

## ELU

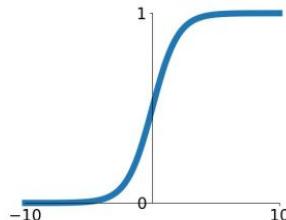
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Activation functions

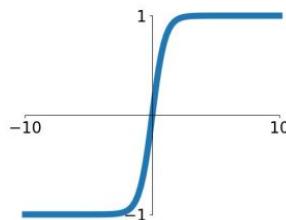
## Sigmoid

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



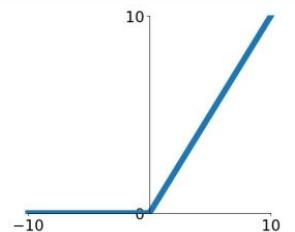
## tanh

$$\tanh(x)$$



## ReLU

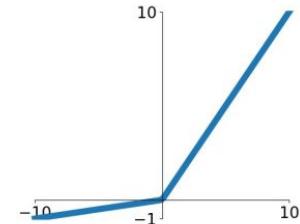
$$\max(0, x)$$



ReLU is a good default choice for most problems

## Leaky ReLU

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

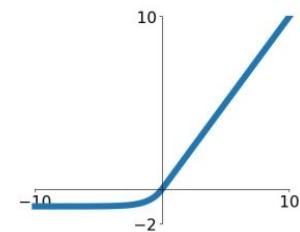


## Maxout

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

## ELU

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

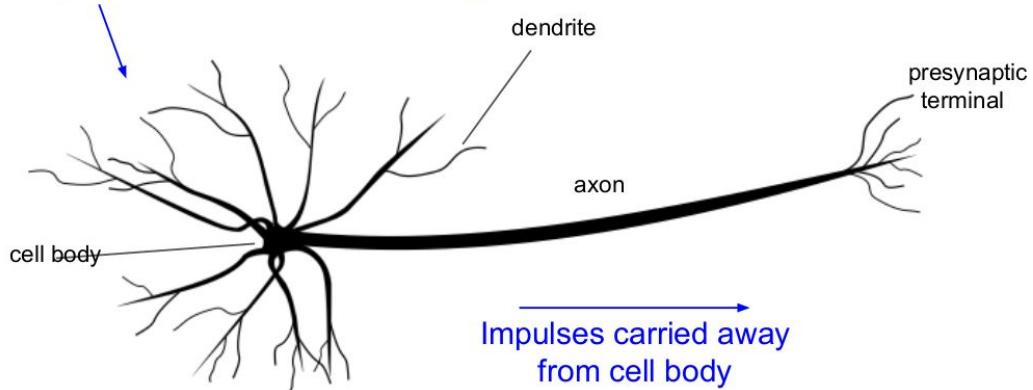


# Neural Network abstraction



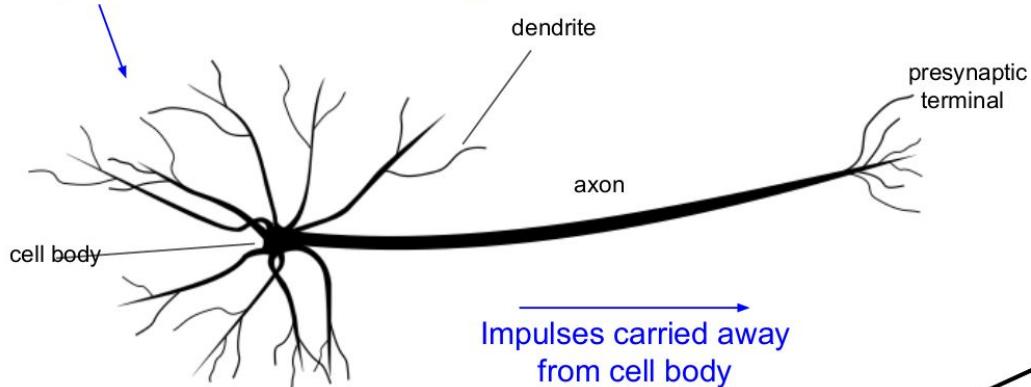
SAPIENZA  
UNIVERSITÀ DI ROMA

Impulses carried toward cell body

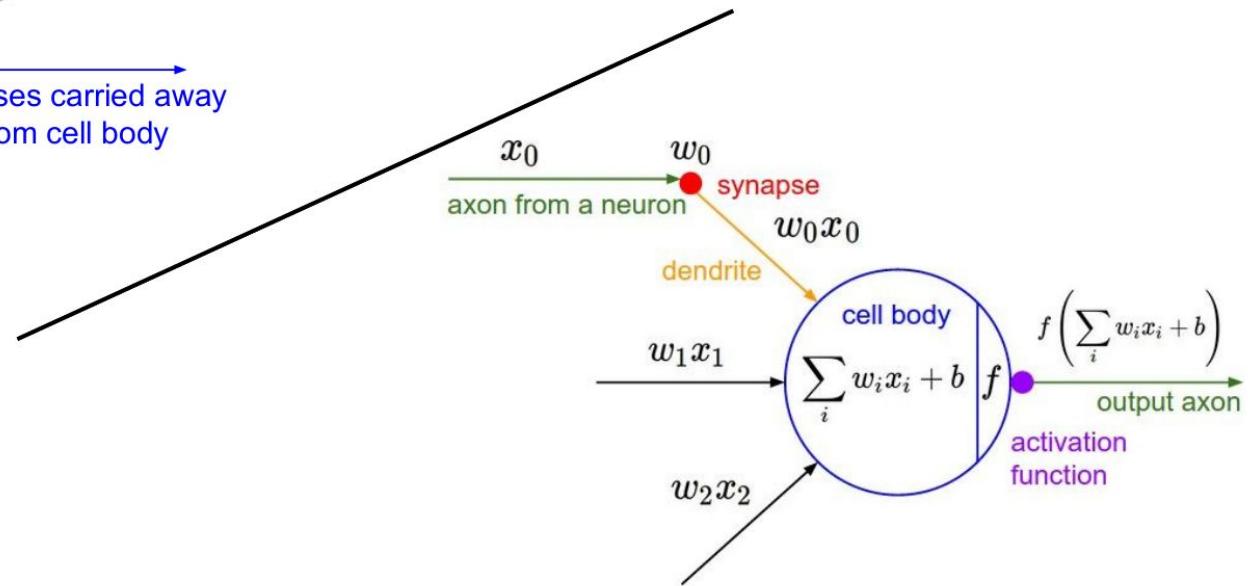


This image by Felipe Perucho  
is licensed under CC-BY 3.0

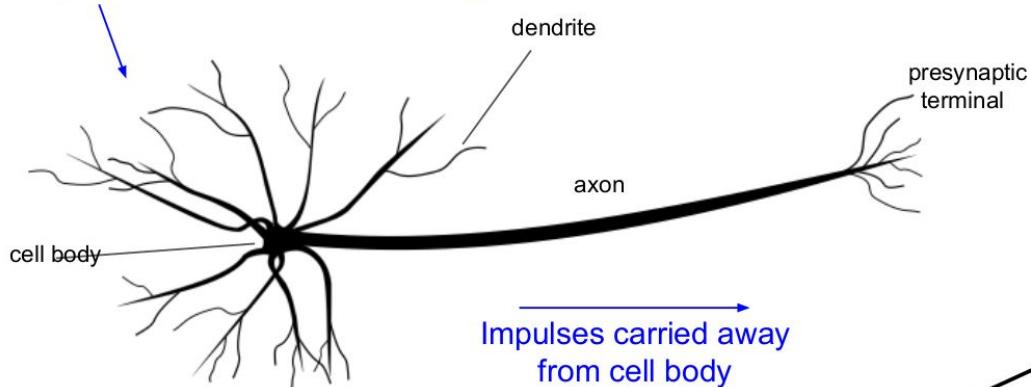
Impulses carried toward cell body



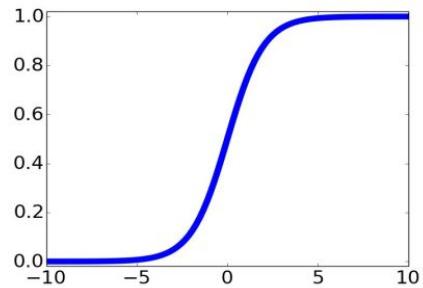
This image by Felipe Perucho  
is licensed under CC-BY 3.0



Impulses carried toward cell body



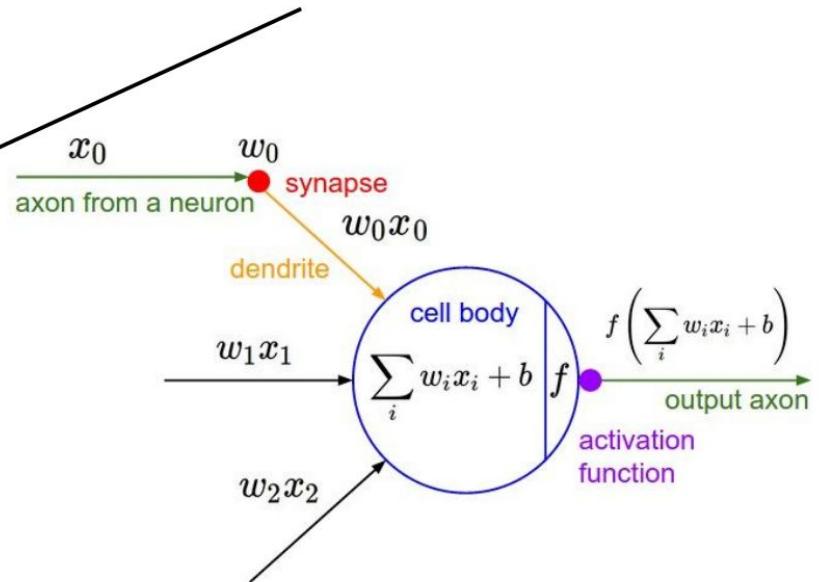
This image by Felipe Perucho  
is licensed under CC-BY 3.0



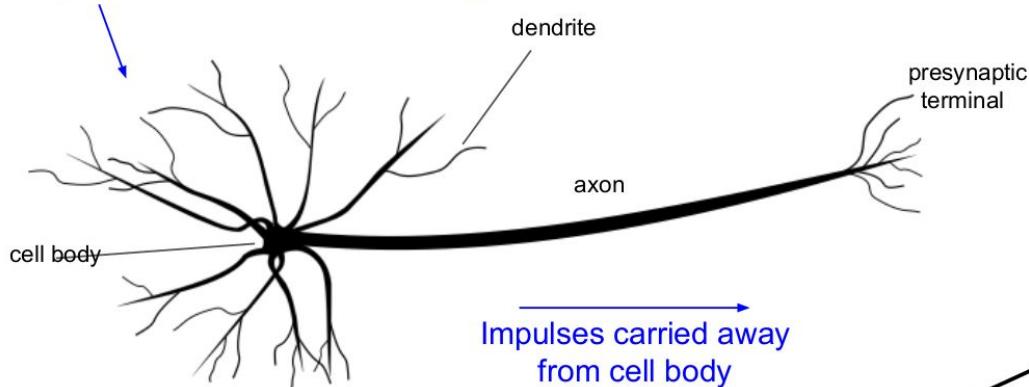
sigmoid activation function

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

Impulses carried away from cell body

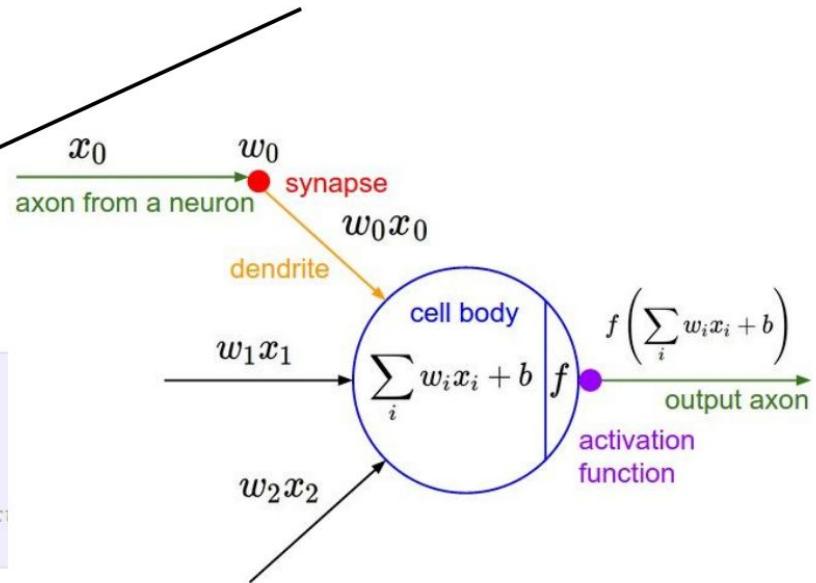


Impulses carried toward cell body

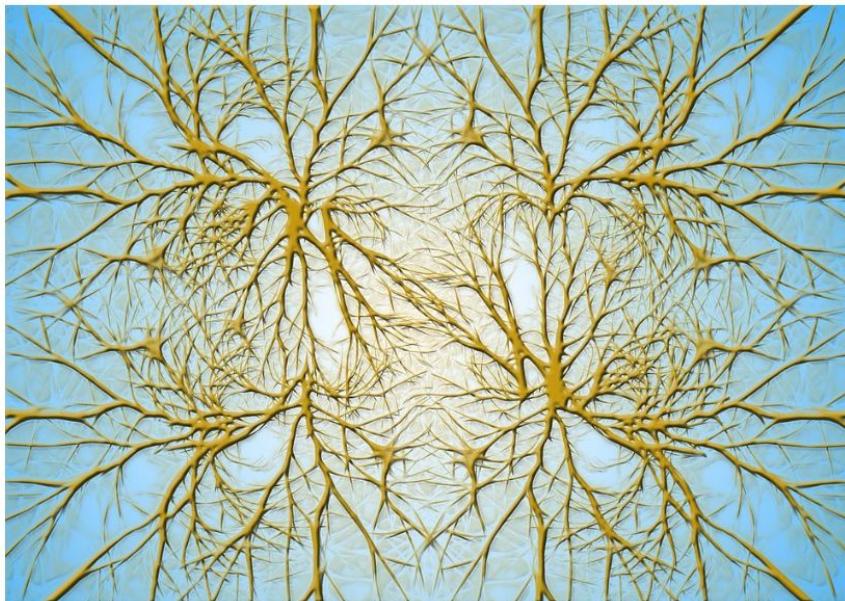


This image by Felipe Perucho  
is licensed under CC-BY 3.0

```
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
```

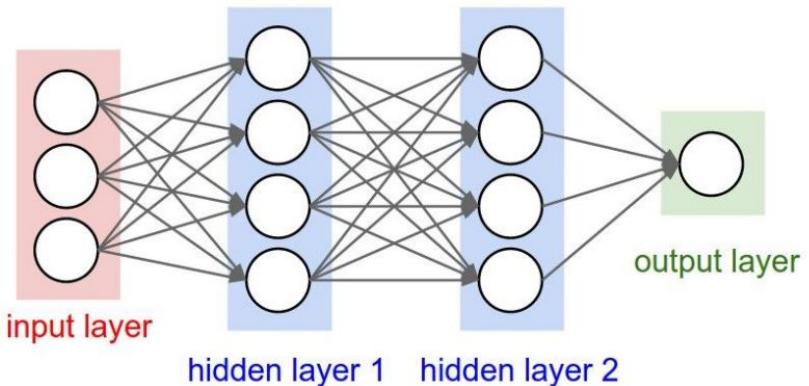


## Biological Neurons: Complex connectivity patterns

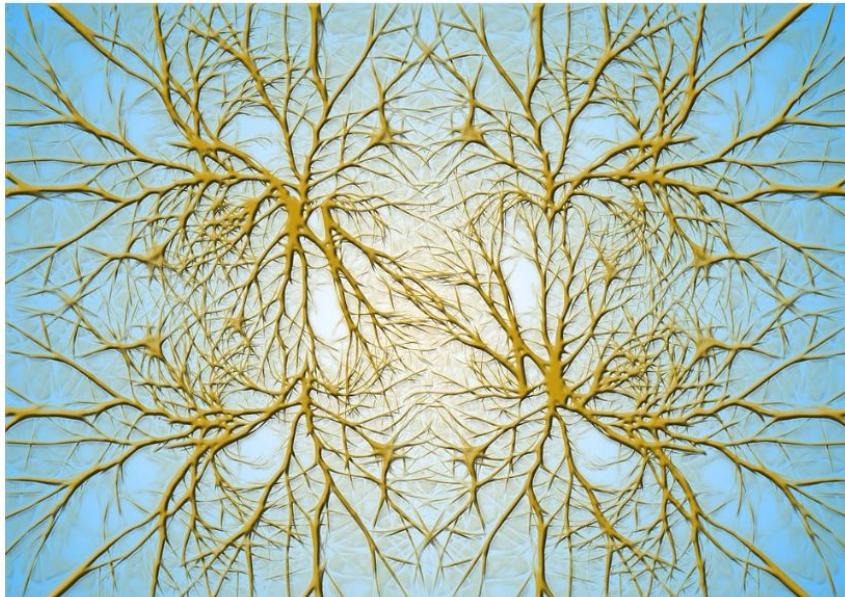


[This image is CC0 Public Domain](#)

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

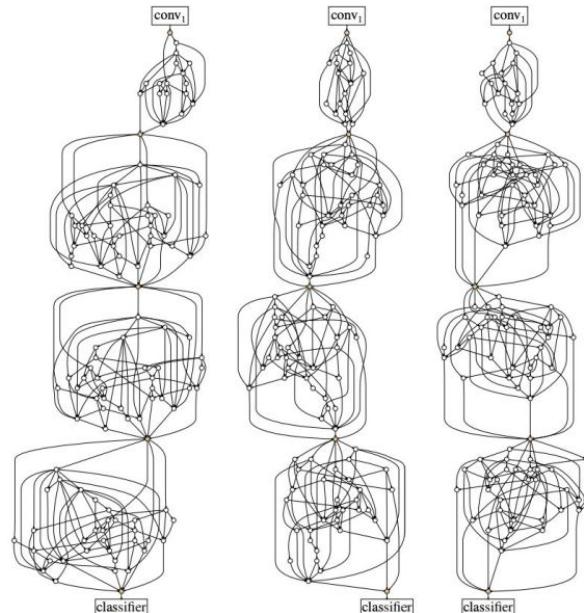


## Biological Neurons: Complex connectivity patterns



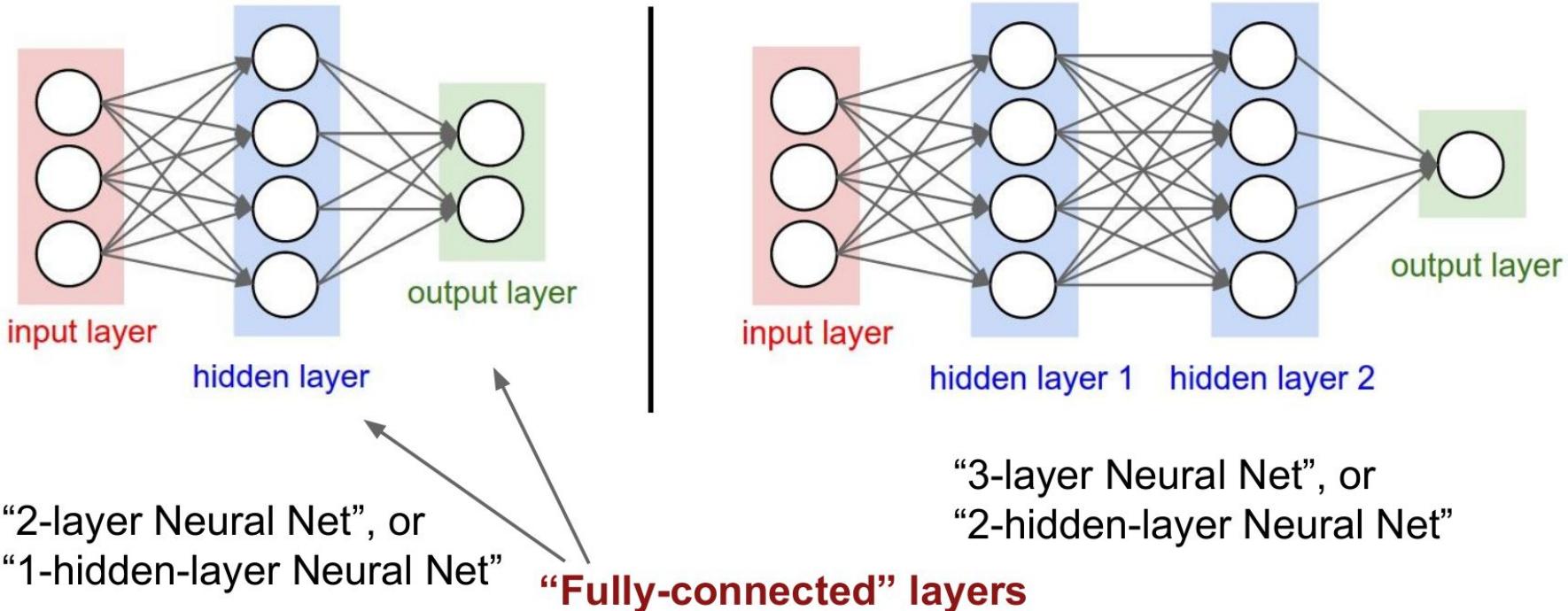
[This image is CC0 Public Domain](#)

## But neural networks with random connections can work too!

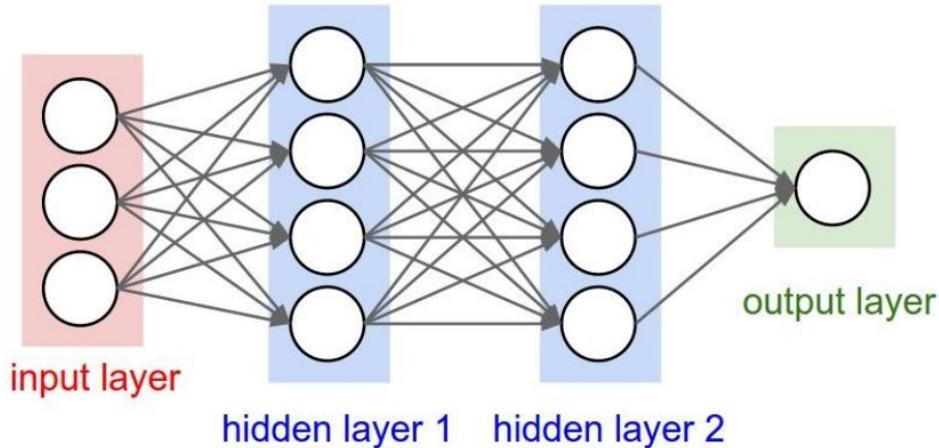


Xie et al, "Exploring Randomly Wired Neural Networks for Image Recognition", arXiv 2019

# 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)
```

# 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

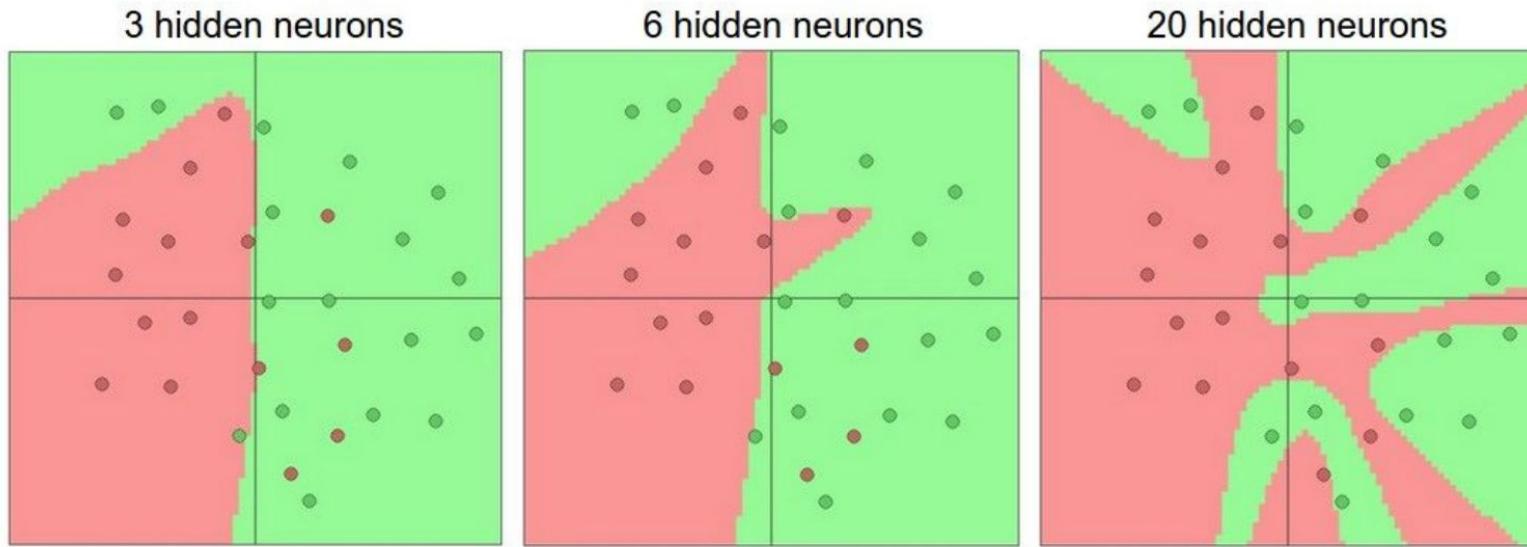
[Dendritic Computation. London and Häusser]

# Backpropagation



SAPIENZA  
UNIVERSITÀ DI ROMA

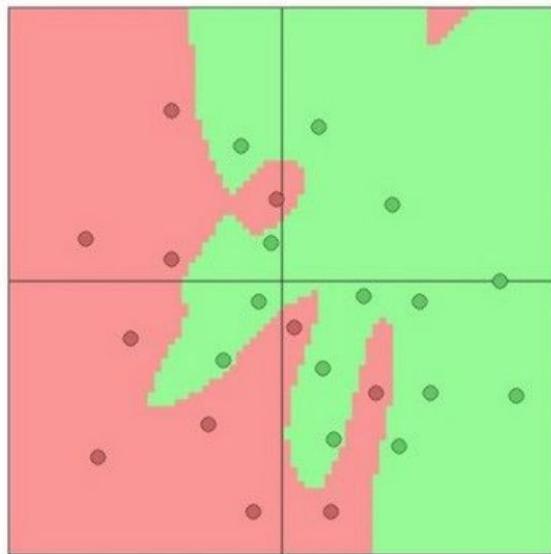
# Setting the number of layers and their sizes



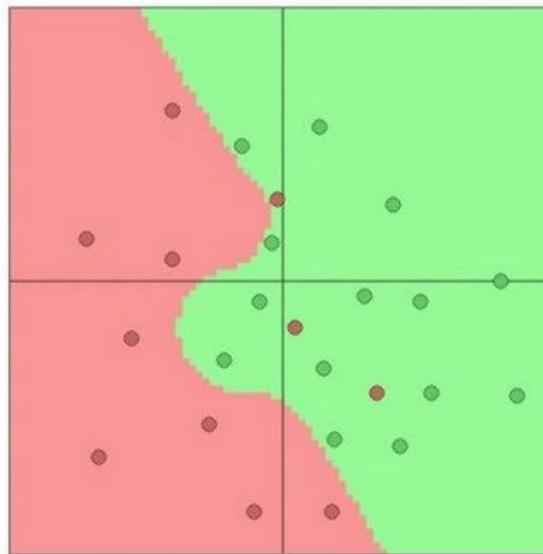
more neurons = more capacity

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>

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

# Plugging in neural networks with loss functions

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

Nonlinear score function

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

SVM Loss on predictions

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

Regularization

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

Total loss: data loss + regularization

# 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

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

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

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

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

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

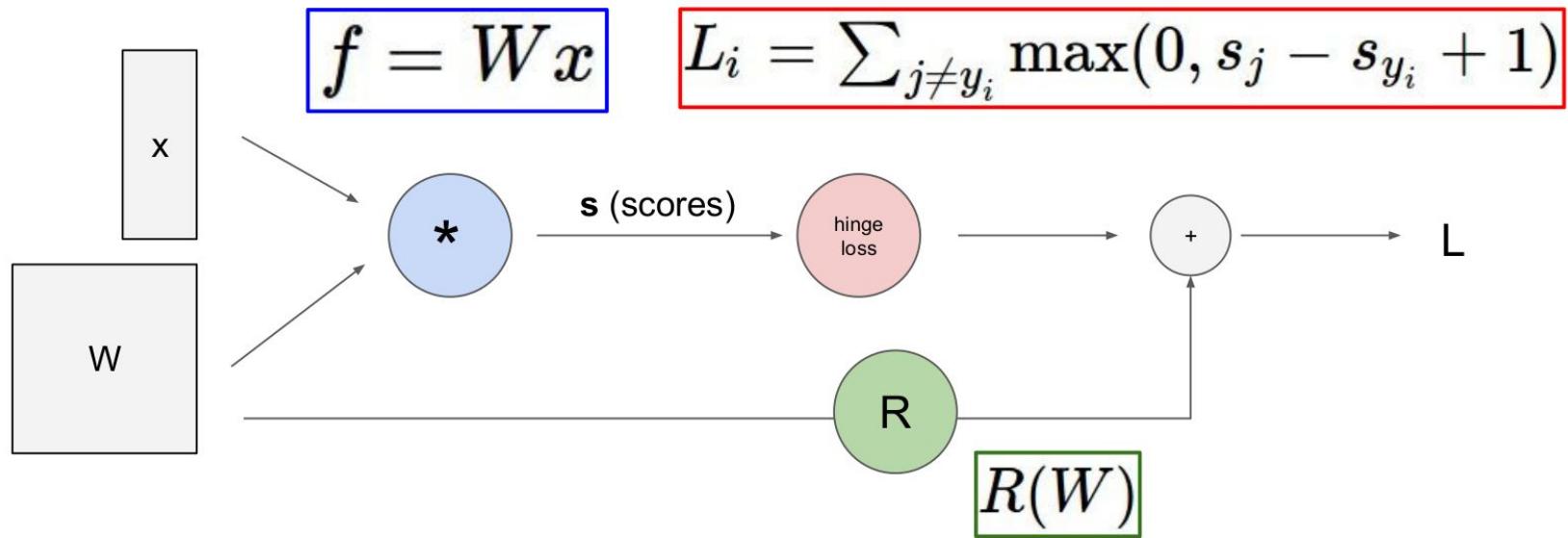
$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

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

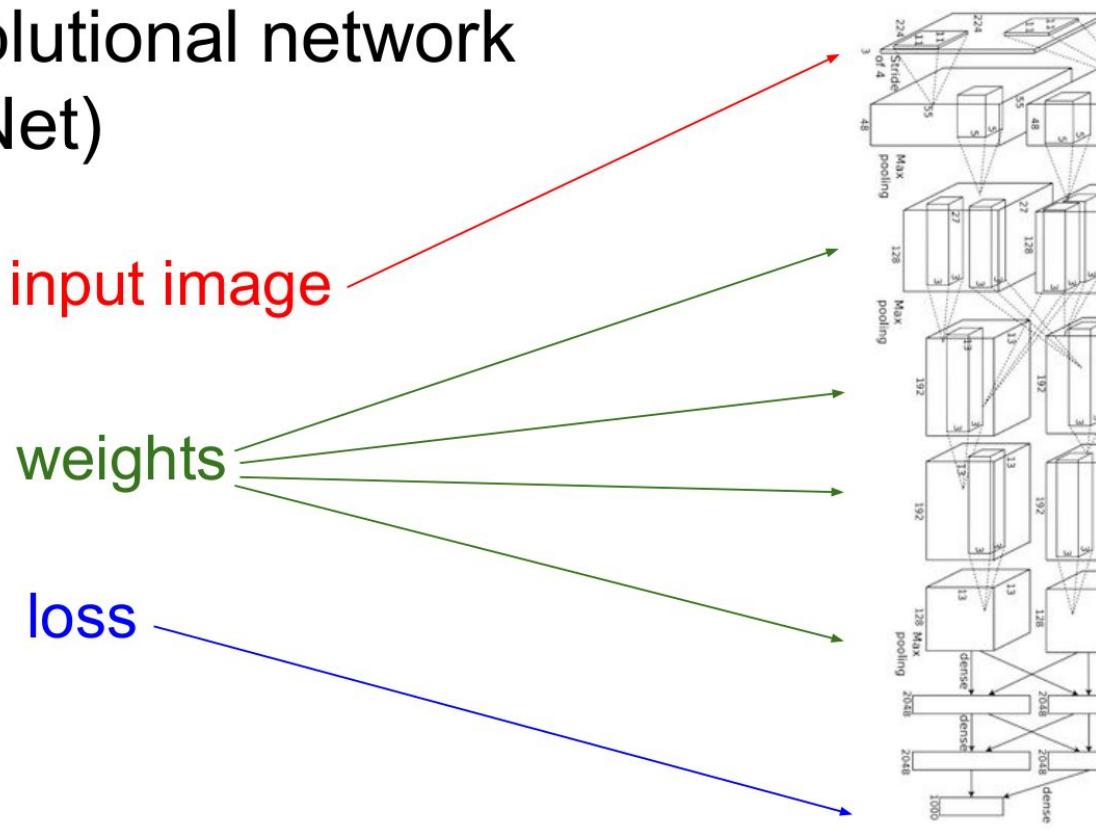
**Problem:** What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

**Problem:** Not feasible for very complex models!

# Better Idea: Computational graphs + Backpropagation



# Convolutional network (AlexNet)



# Really complex neural networks!!

input image

loss

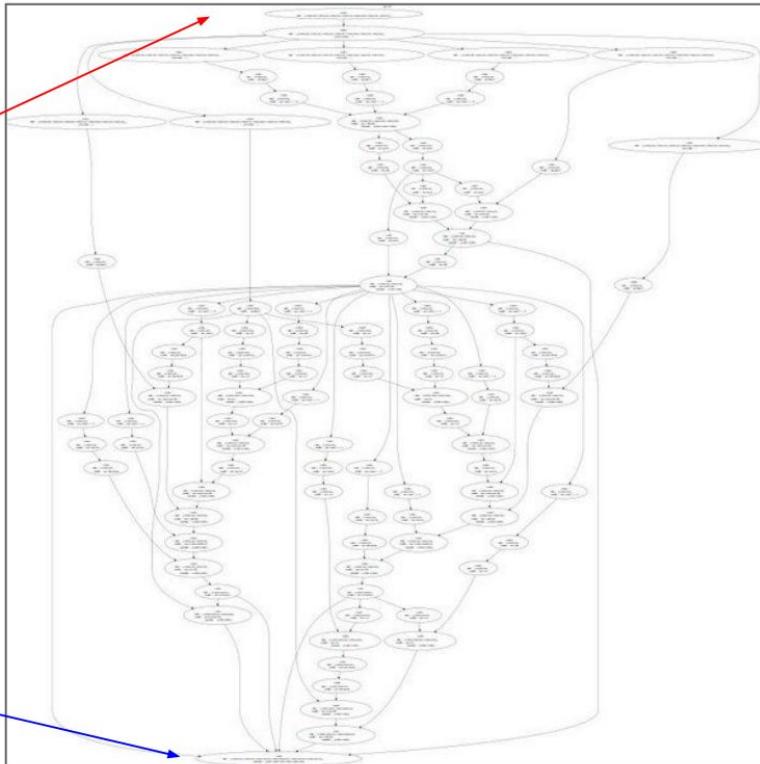


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

# Neural Turing Machine

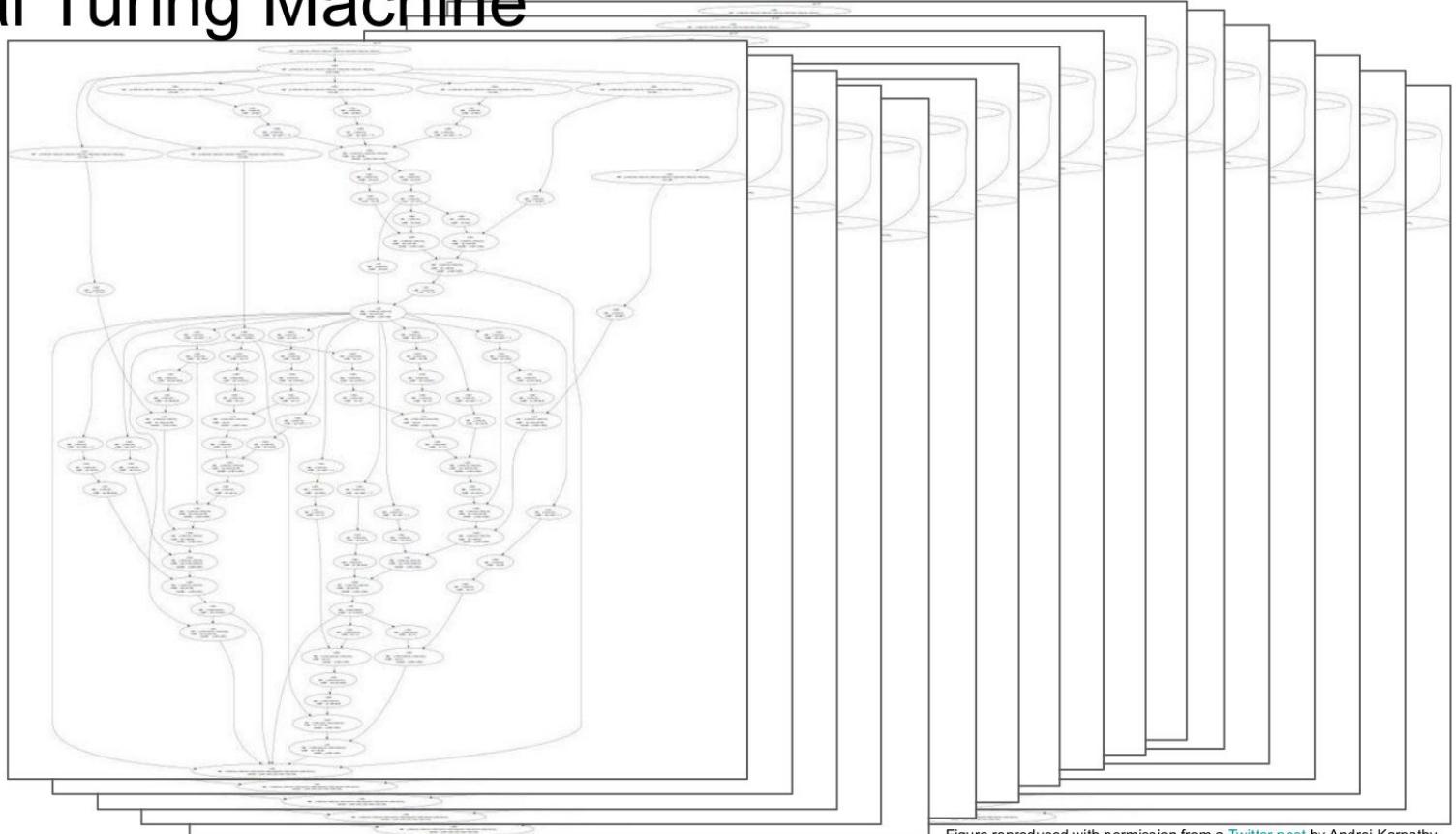
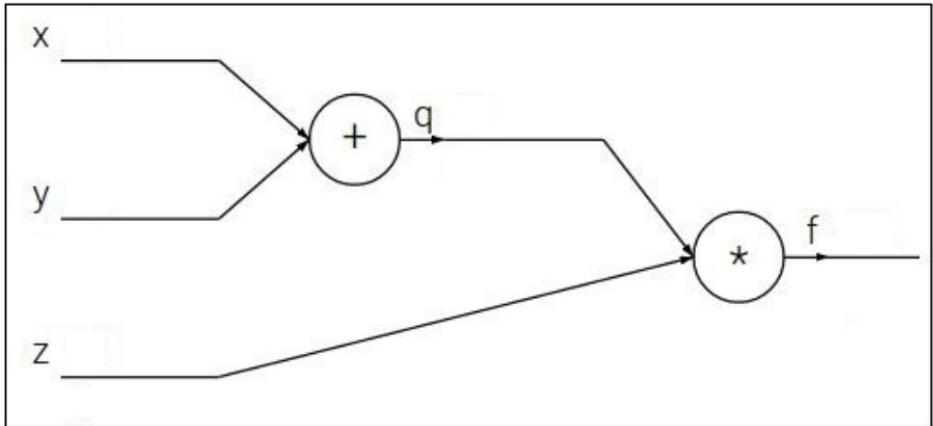


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

## Backpropagation: a simple example

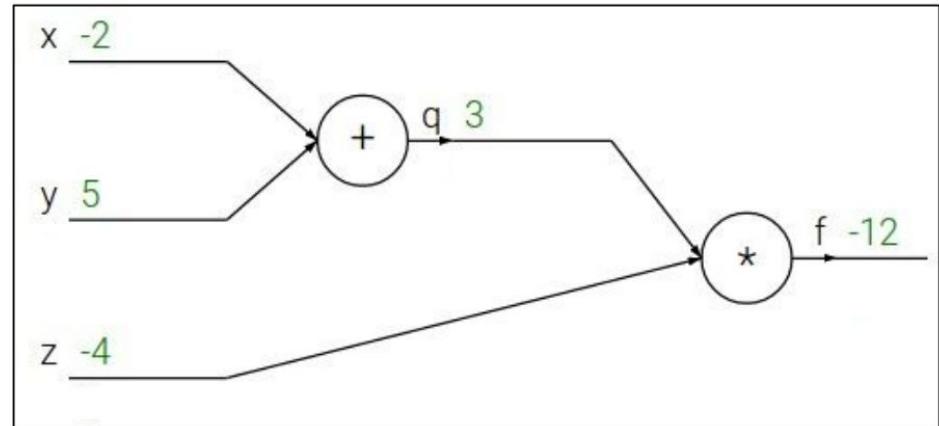
$$f(x, y, z) = (x + y)z$$



## Backpropagation: a simple example

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

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

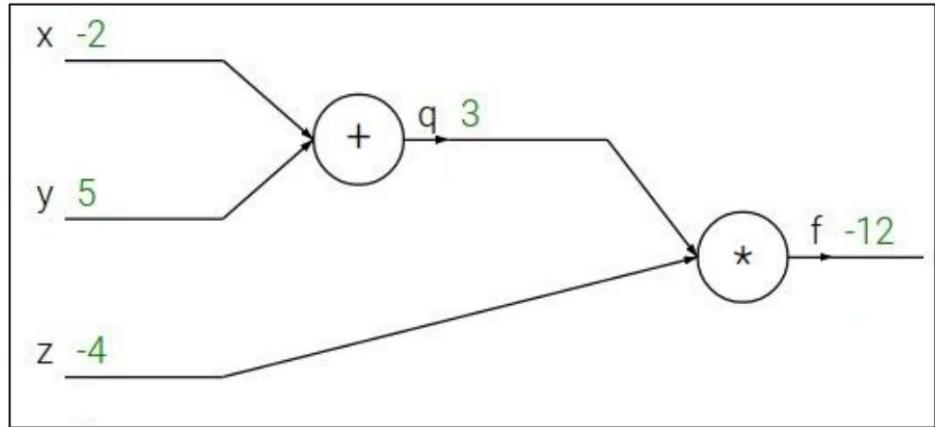


## Backpropagation: a simple example

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

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

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



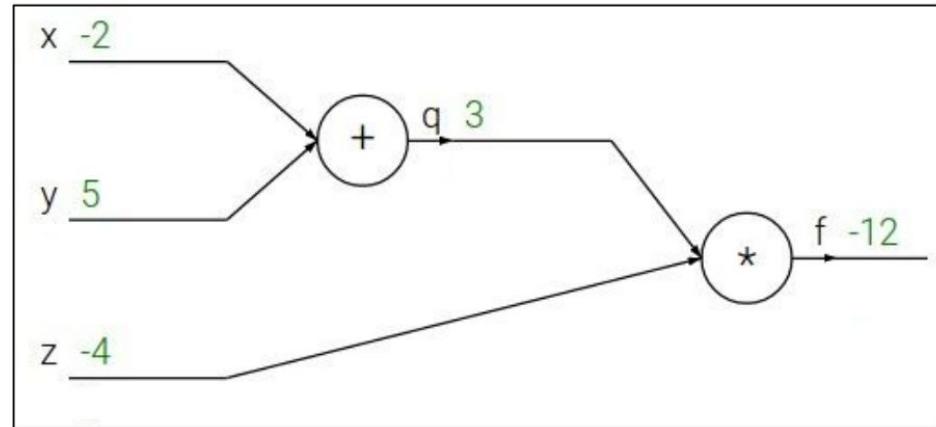
## Backpropagation: a simple example

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

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

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

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

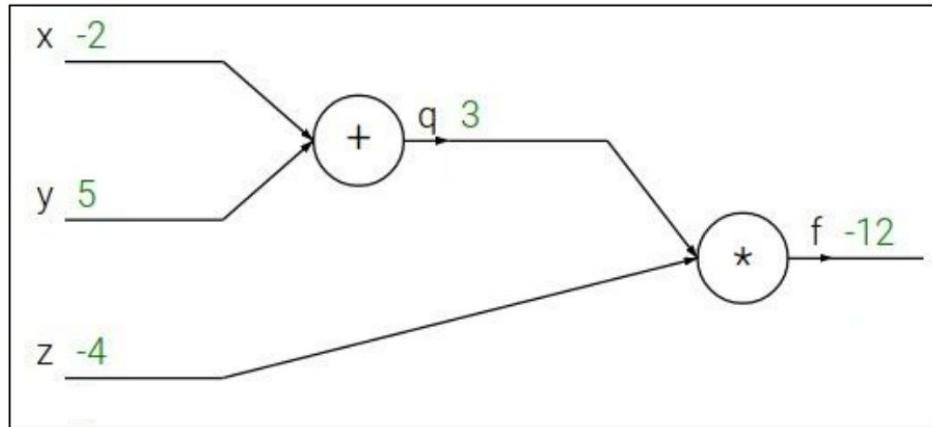


## Backpropagation: a simple example

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

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

## Backpropagation: a simple example

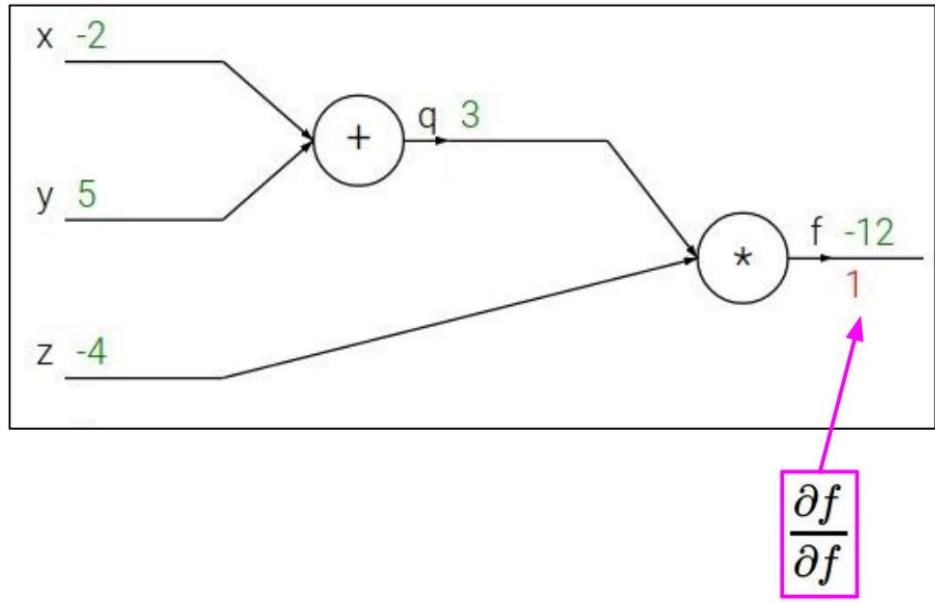
$$f(x, y, z) = (x + y)z$$

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



## Backpropagation: a simple example

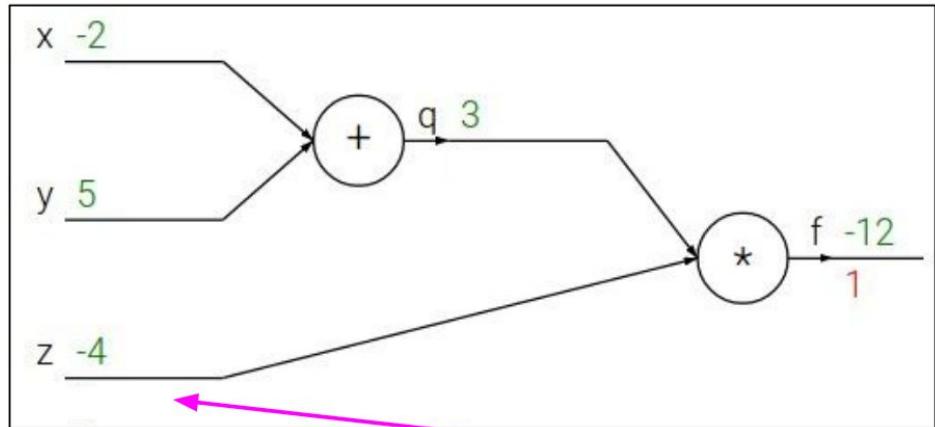
$$f(x, y, z) = (x + y)z$$

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

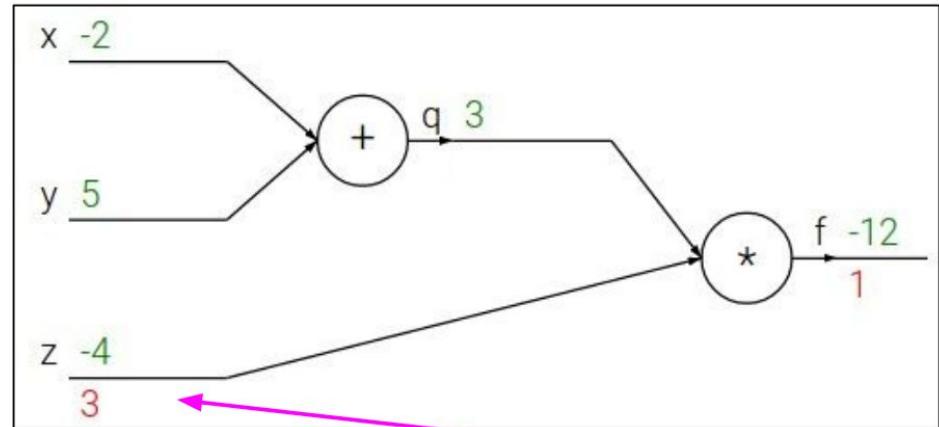
$$f(x, y, z) = (x + y)z$$

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



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

## Backpropagation: a simple example

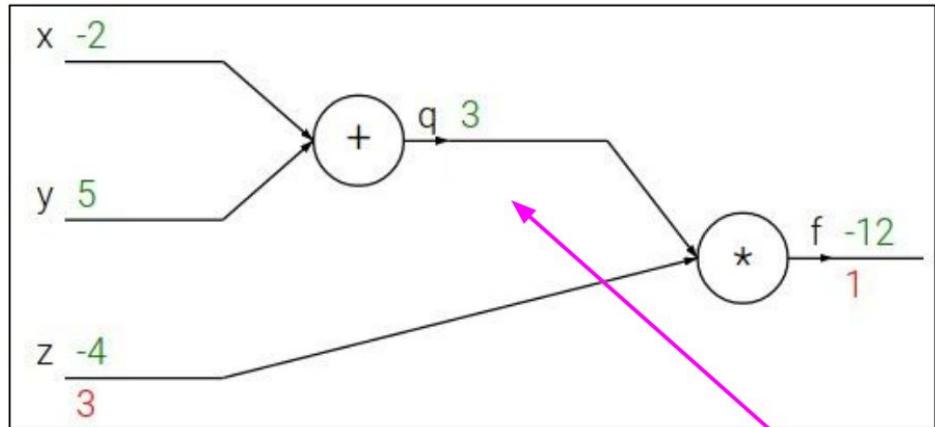
$$f(x, y, z) = (x + y)z$$

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



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

## Backpropagation: a simple example

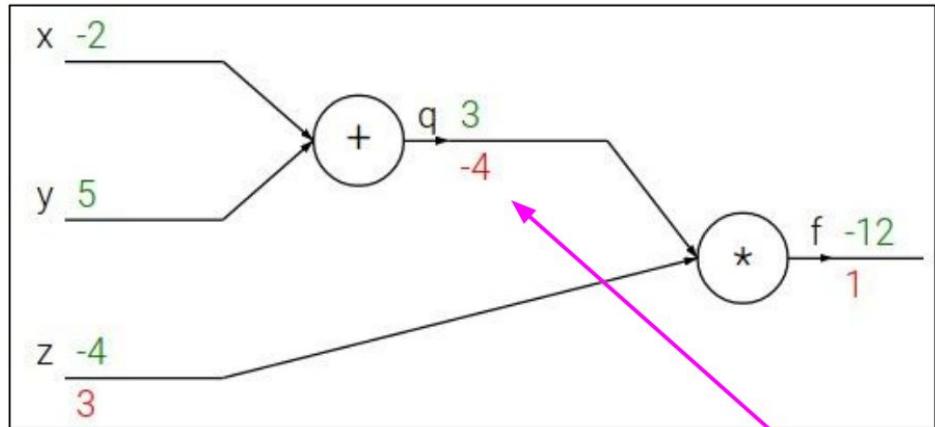
$$f(x, y, z) = (x + y)z$$

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



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

## Backpropagation: a simple example

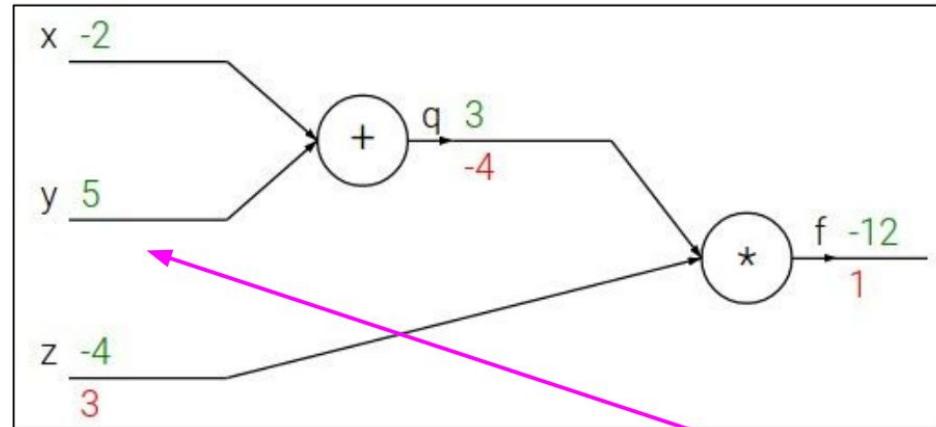
$$f(x, y, z) = (x + y)z$$

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

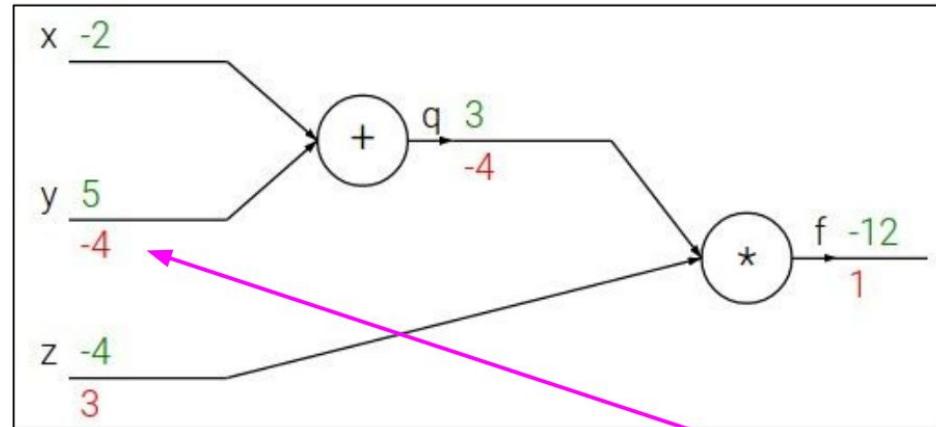
$$f(x, y, z) = (x + y)z$$

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

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

## Backpropagation: a simple example

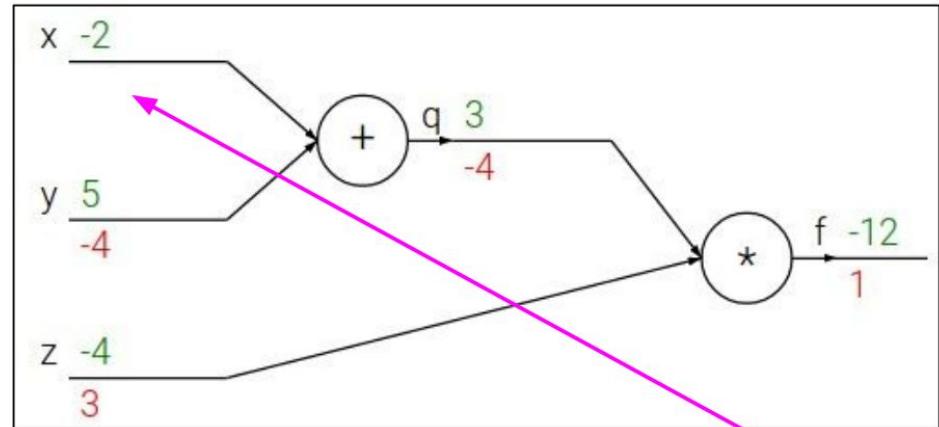
$$f(x, y, z) = (x + y)z$$

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:

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

Upstream  
gradient

Local  
gradient

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

## Backpropagation: a simple example

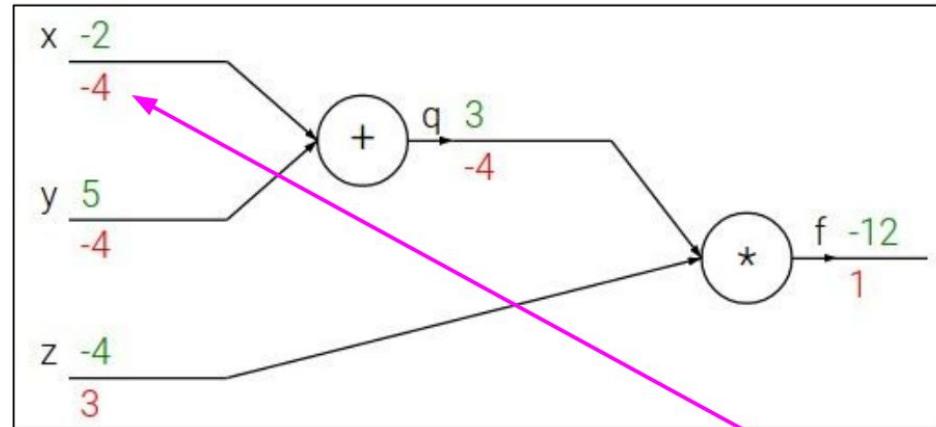
$$f(x, y, z) = (x + y)z$$

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



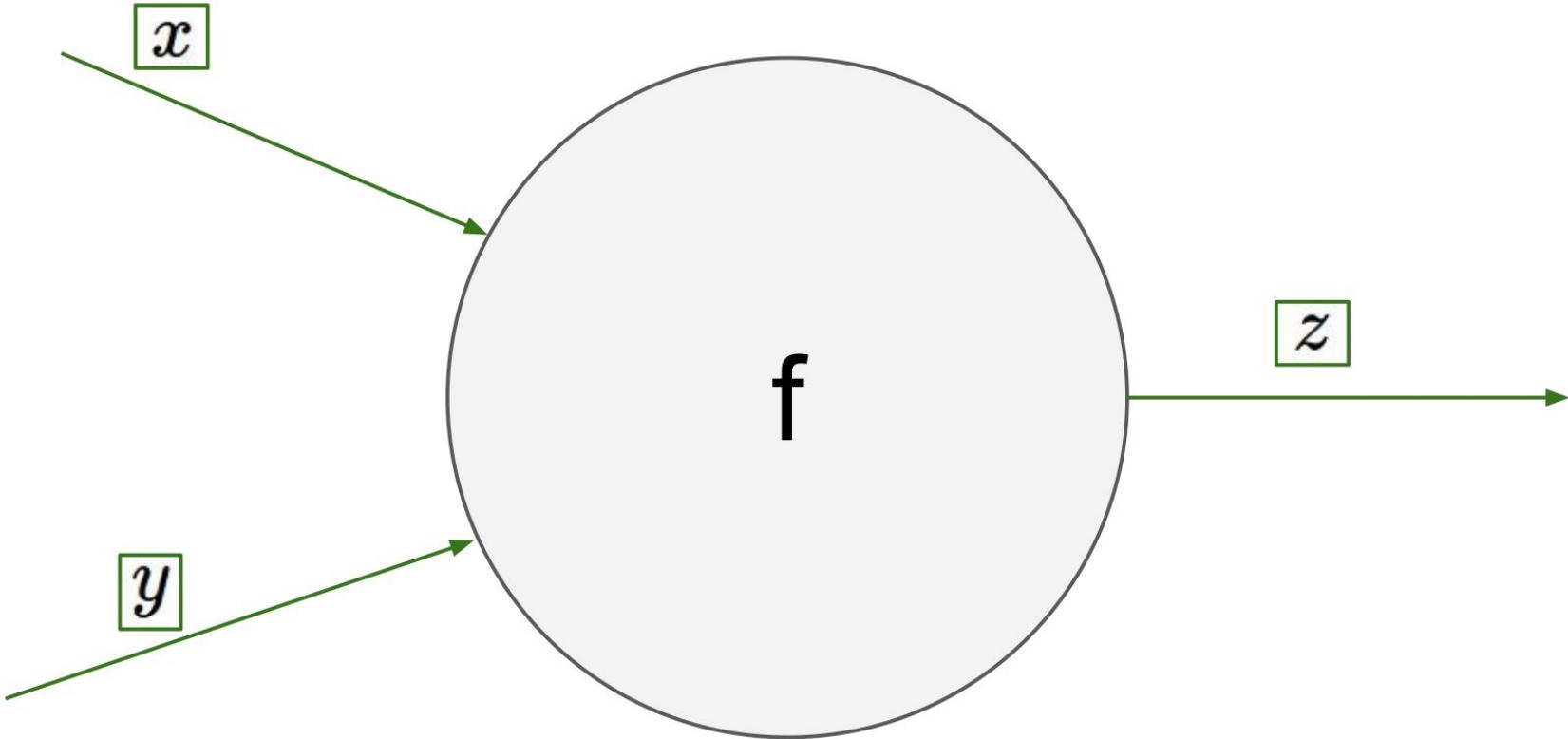
Chain rule:

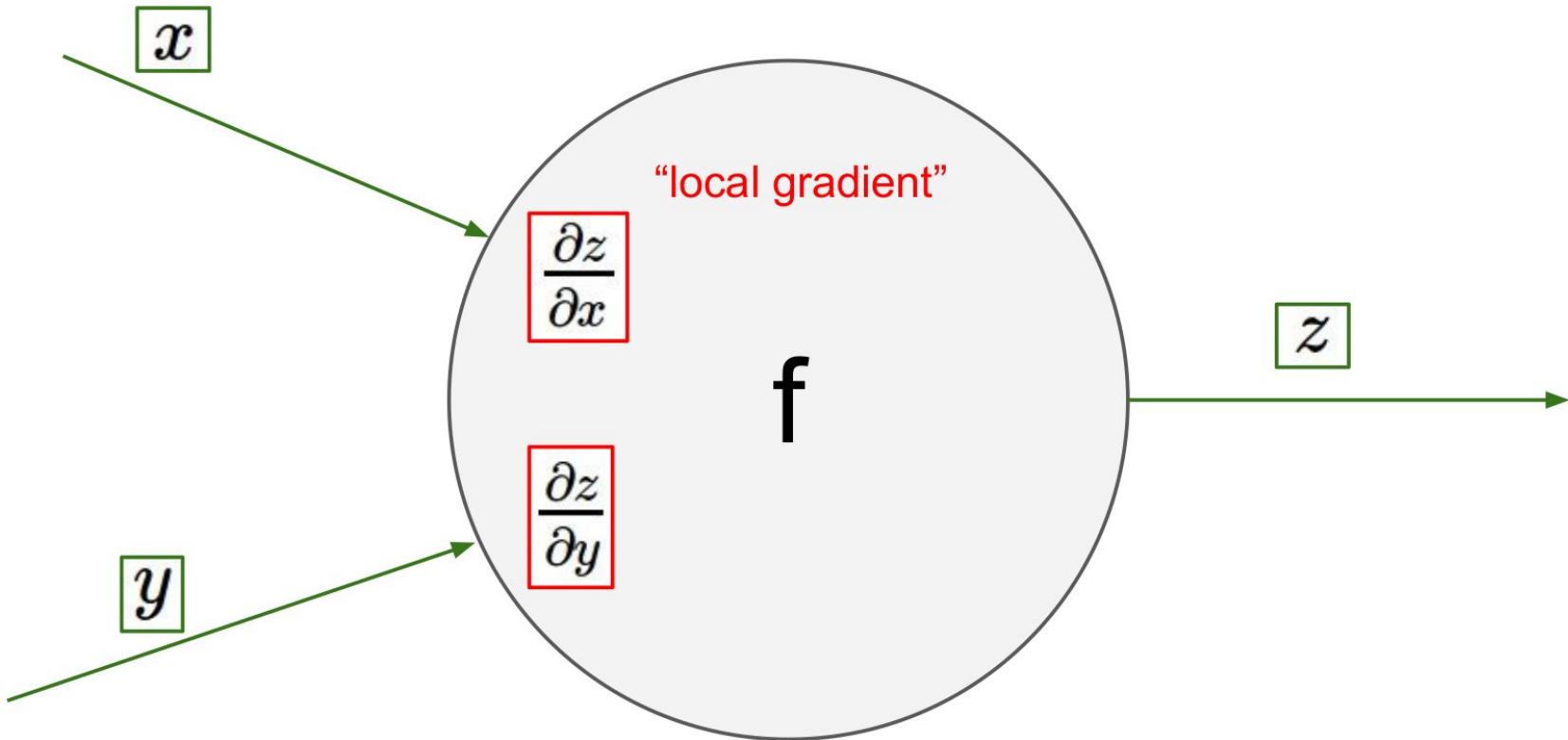
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

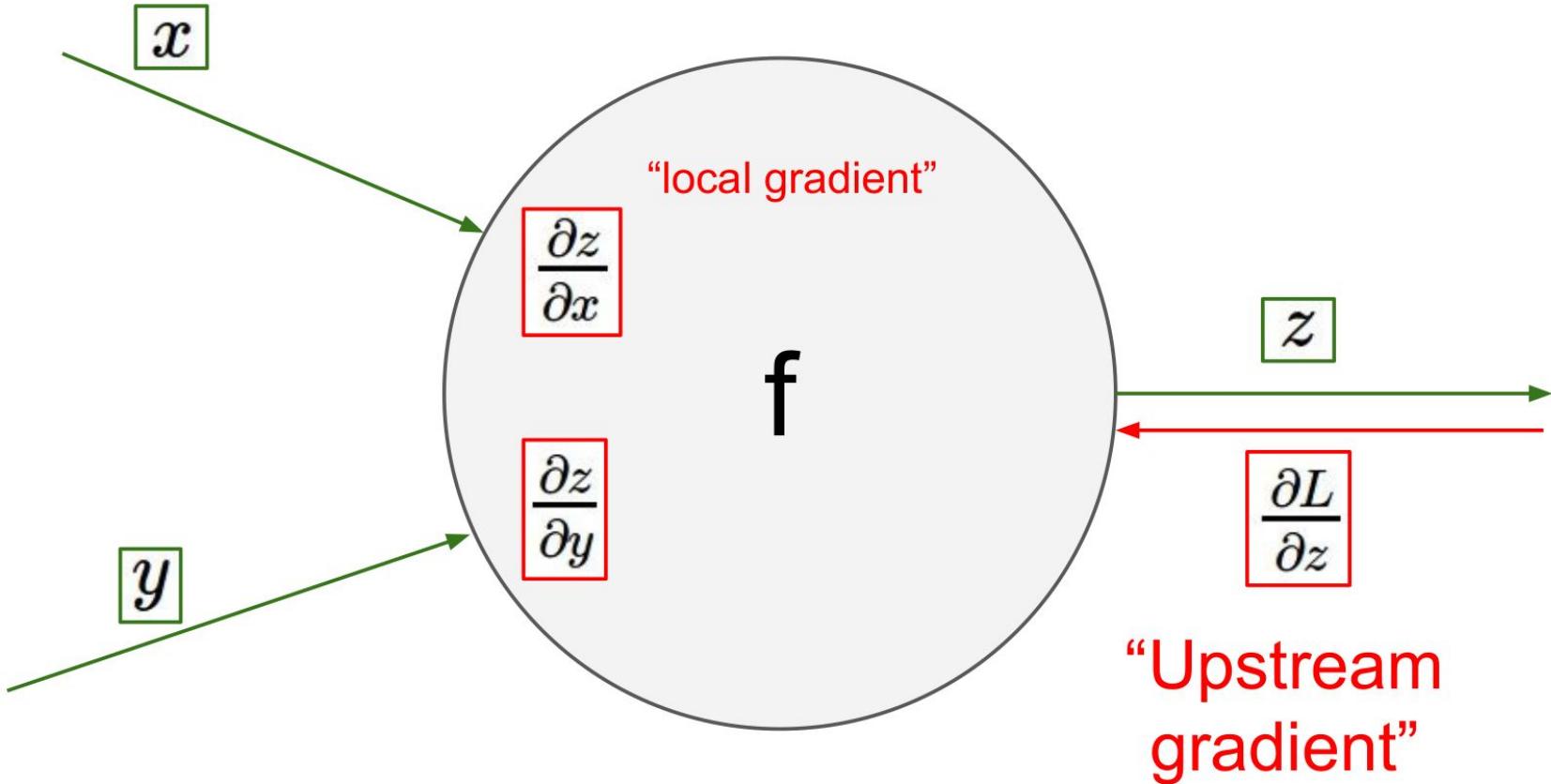
Upstream  
gradient

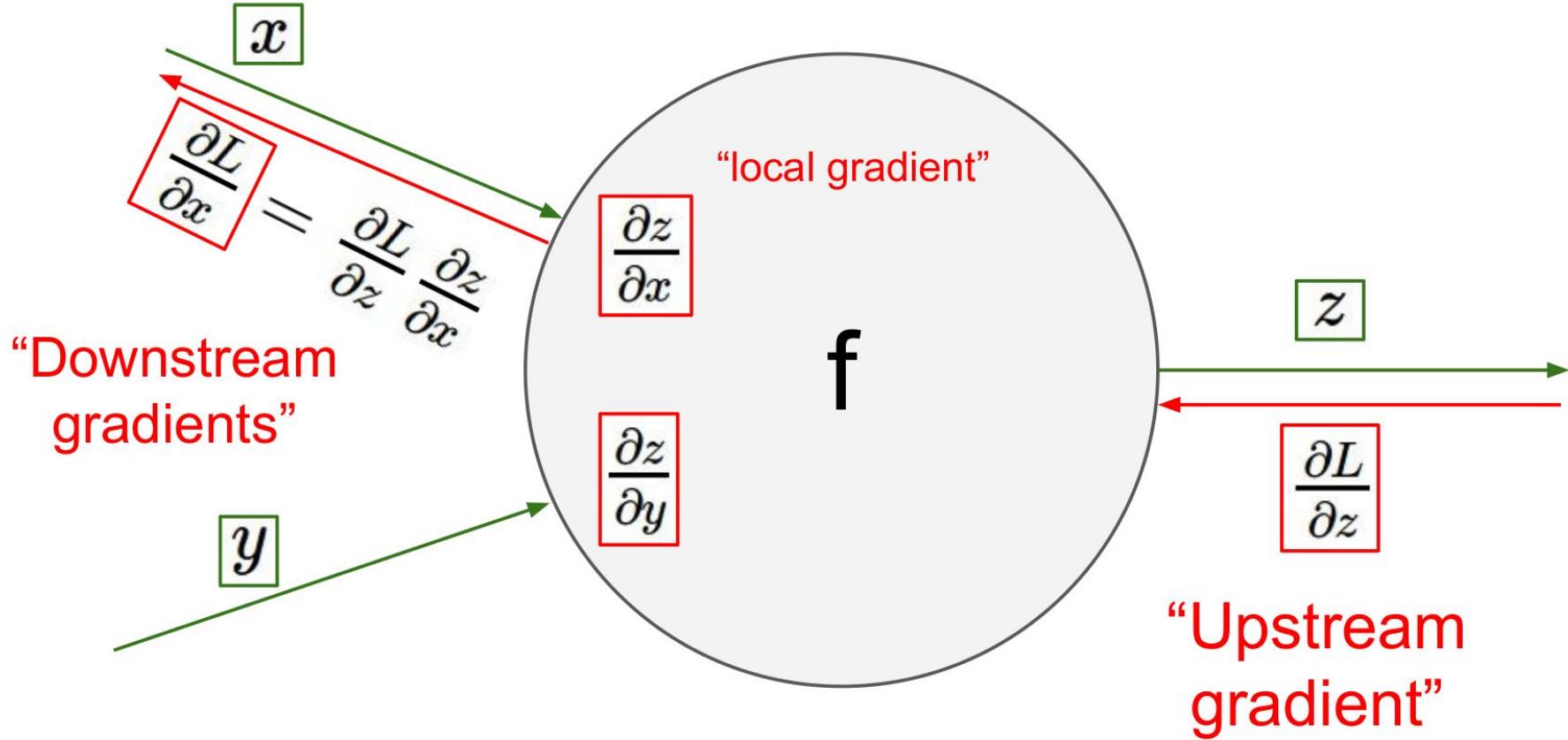
Local  
gradient

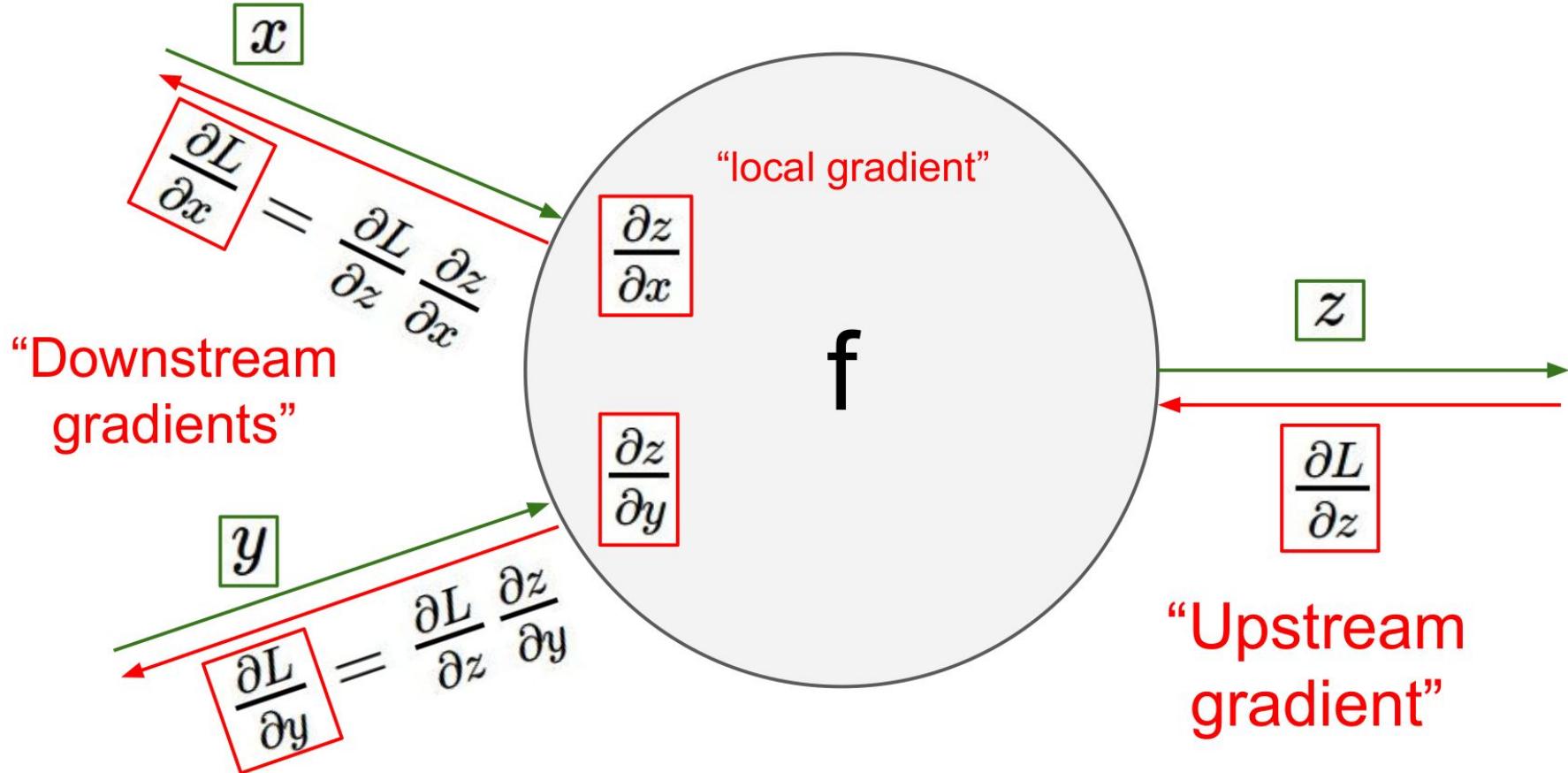
$$\frac{\partial f}{\partial x}$$

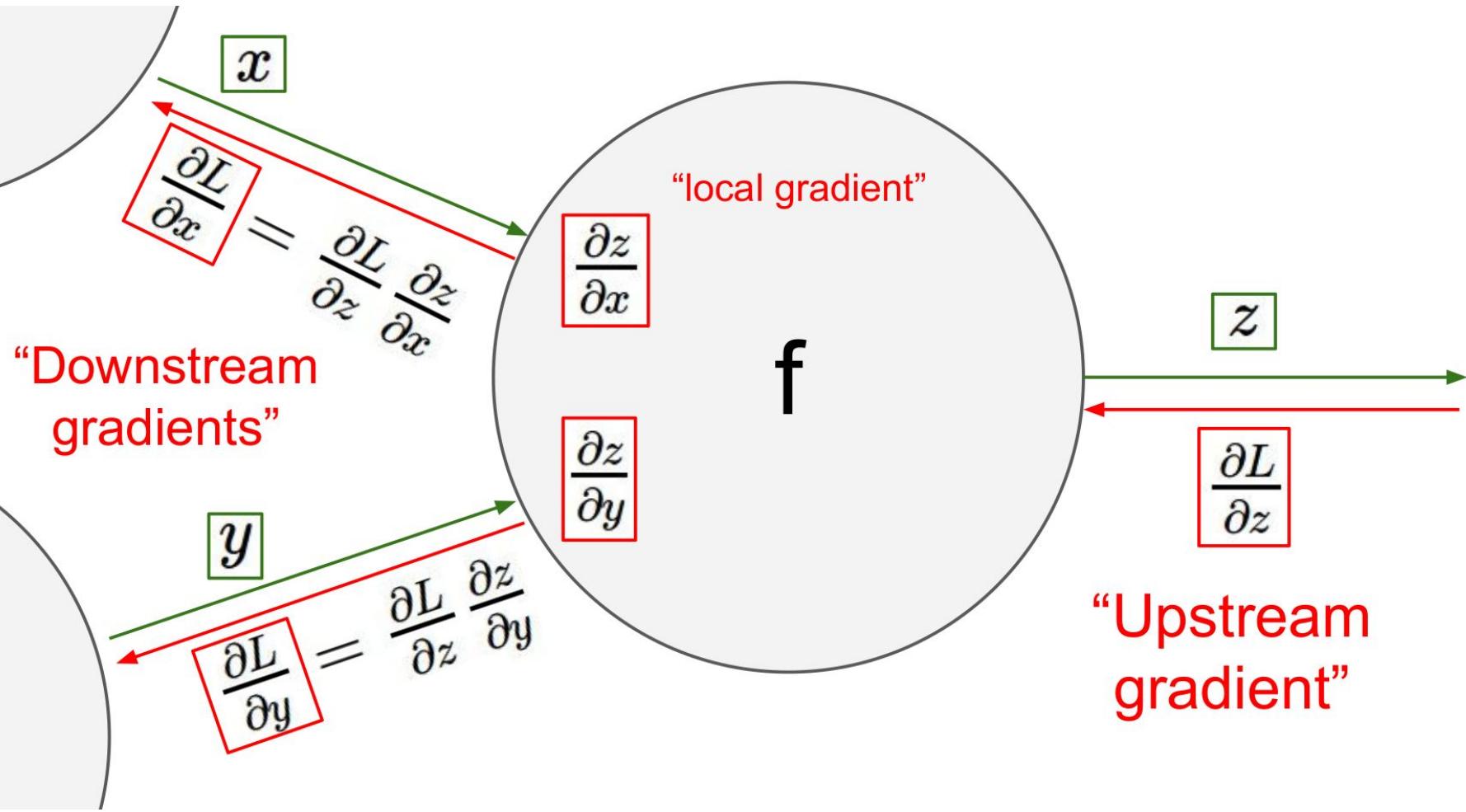






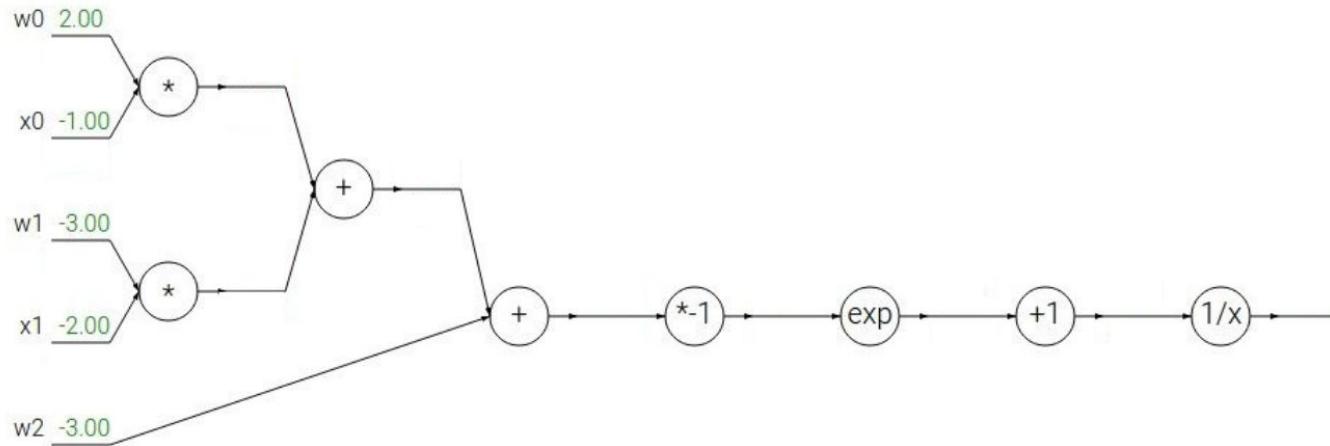






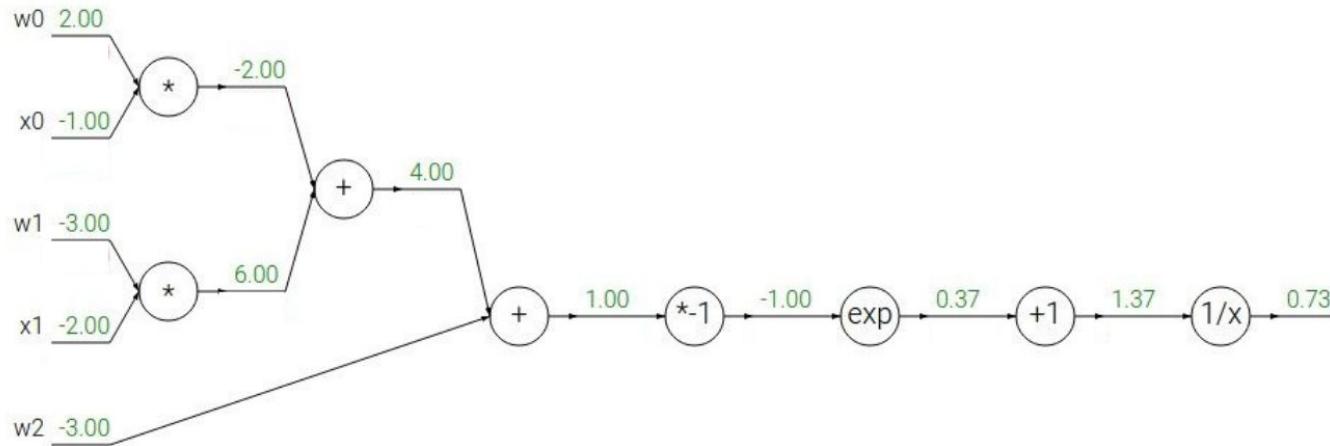
Another example:

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



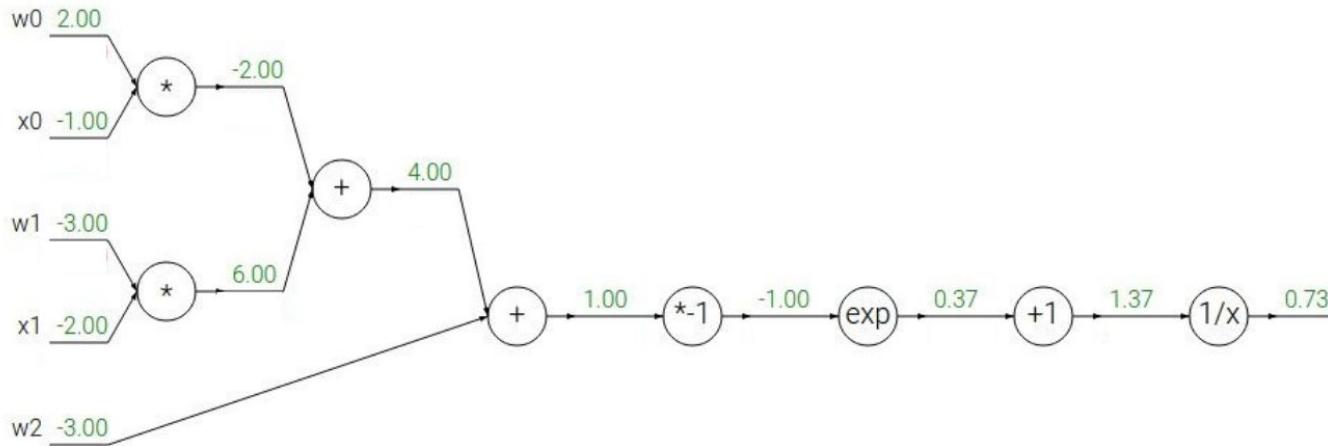
Another example:

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



Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

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

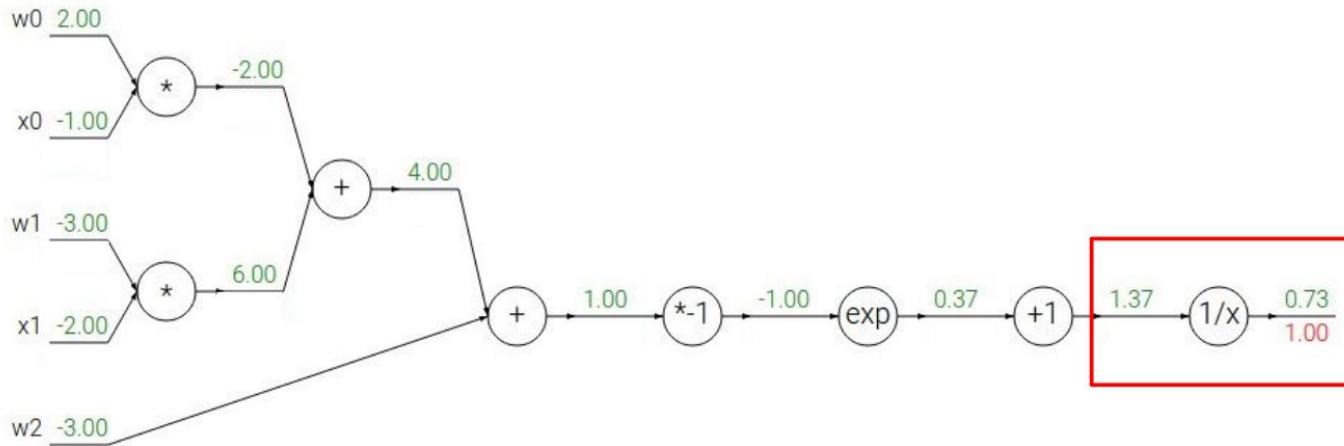
→

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

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

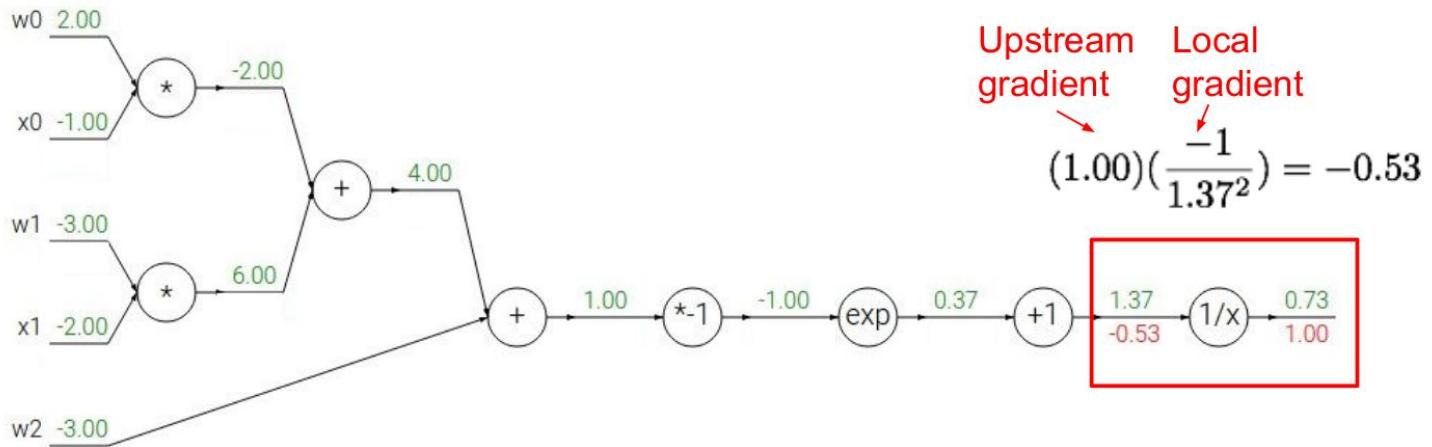
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

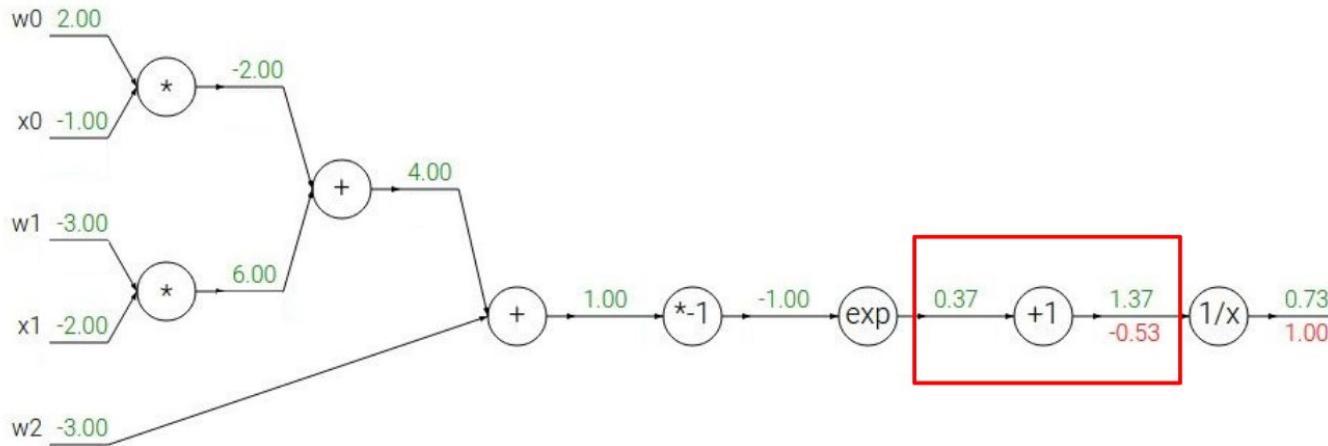
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

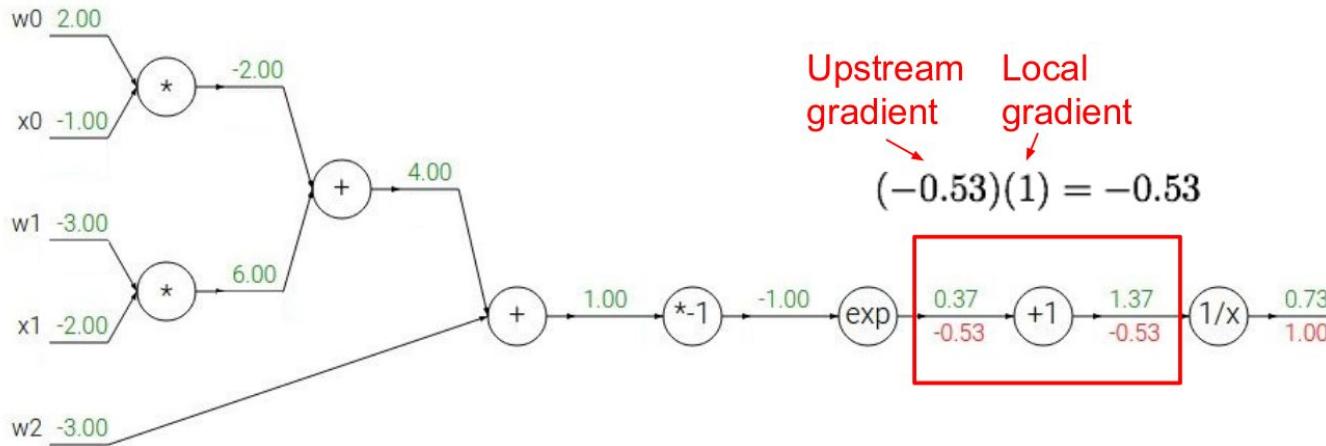
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

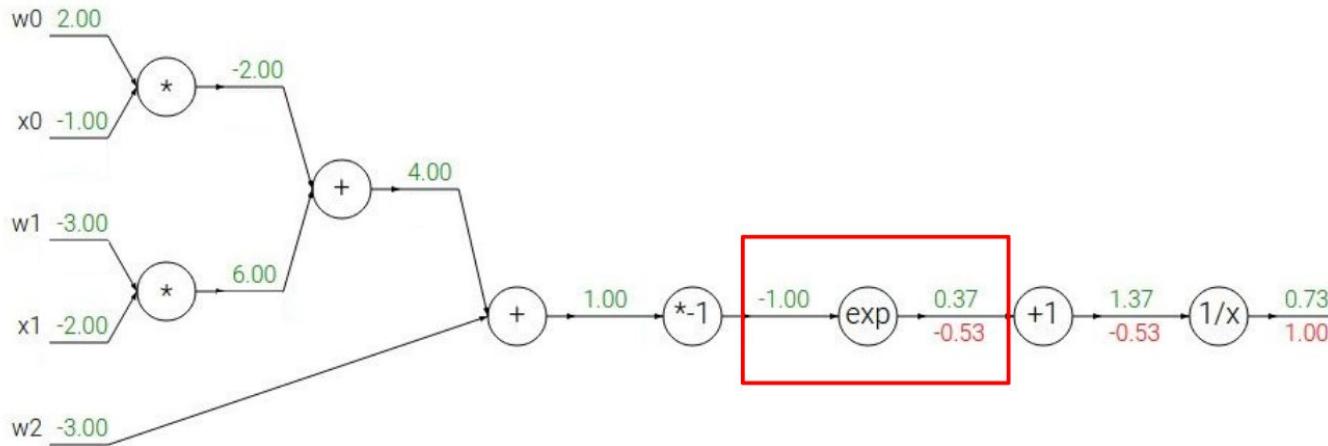
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

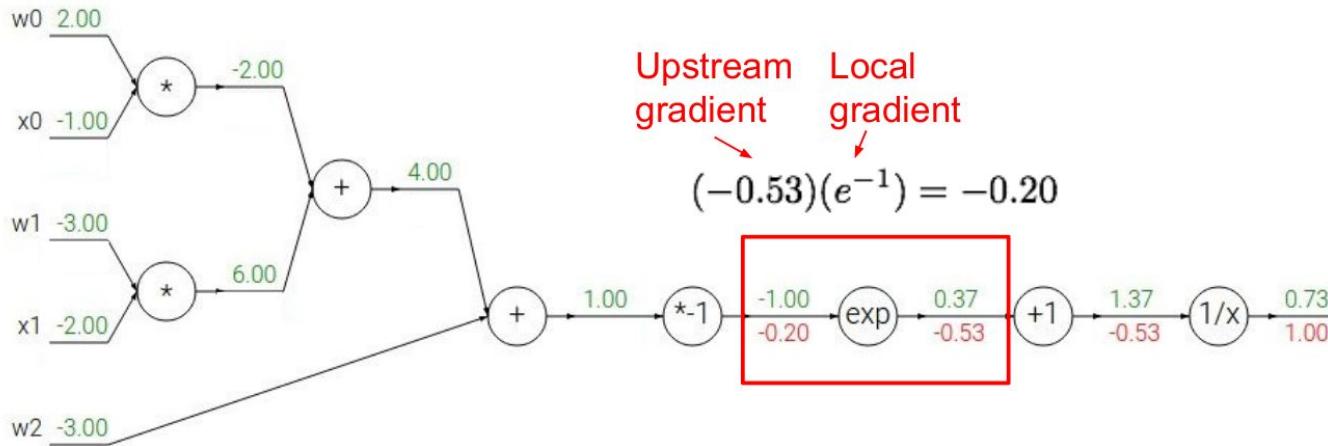
$$f_c(x) = c + x$$

→

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

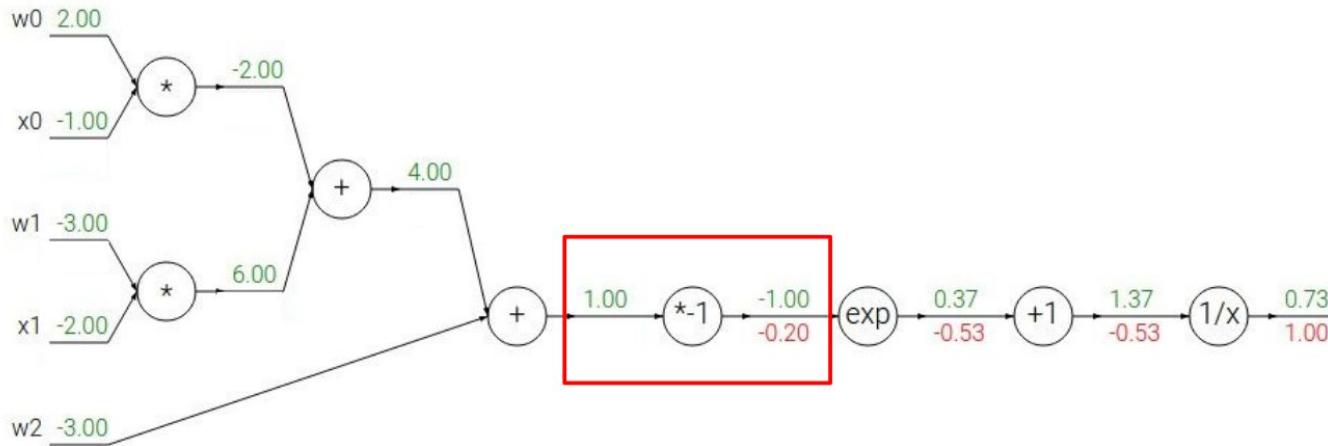
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

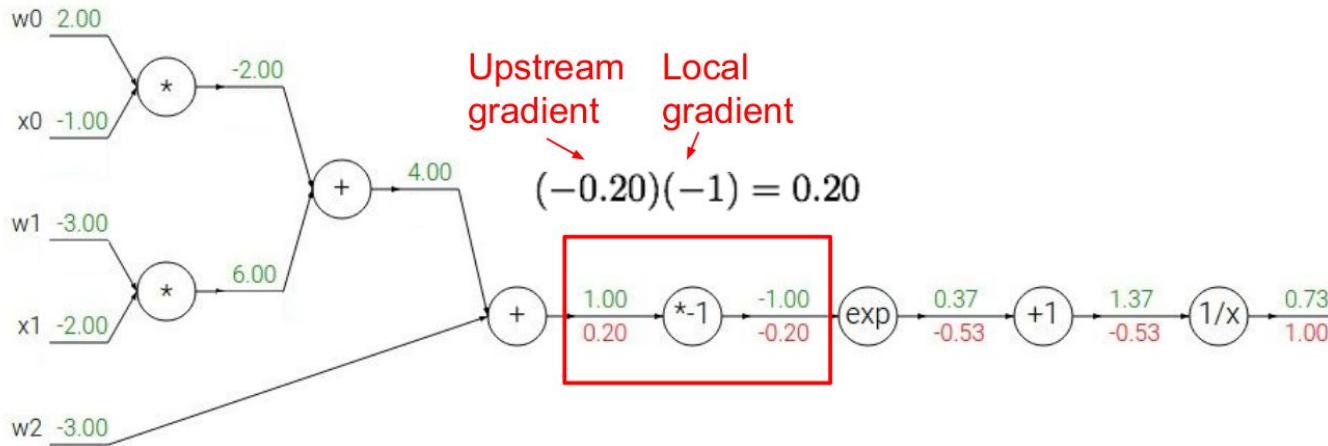
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

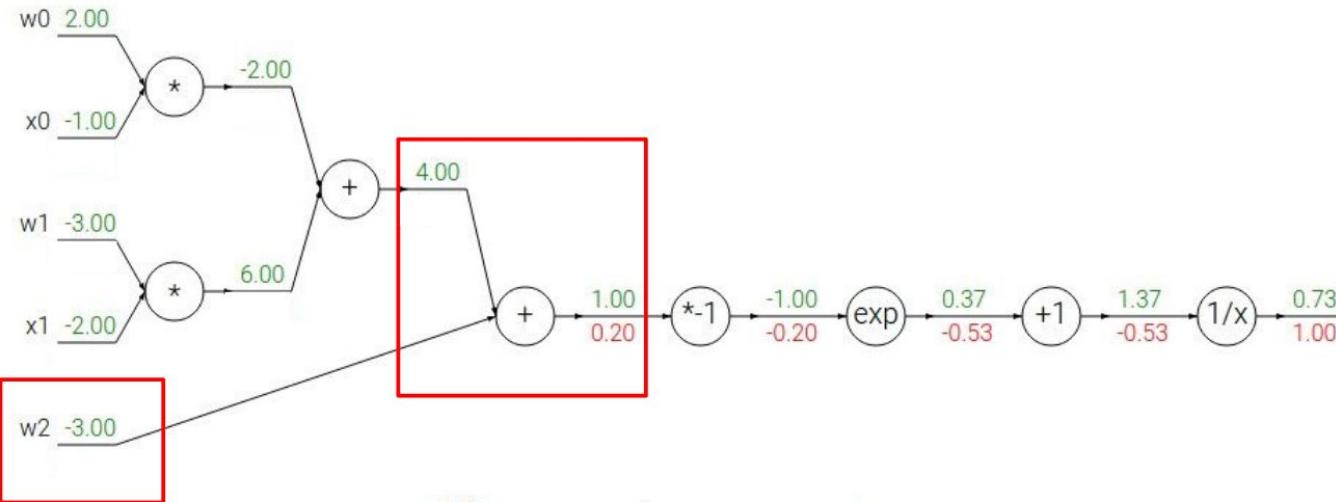
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

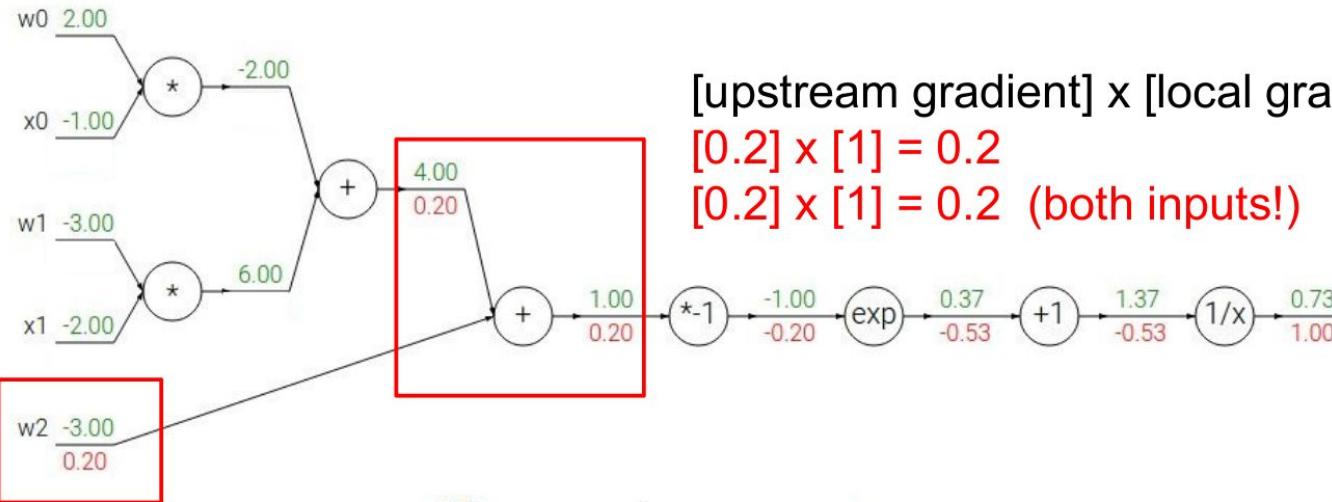
$$f_c(x) = c + x$$

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

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

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

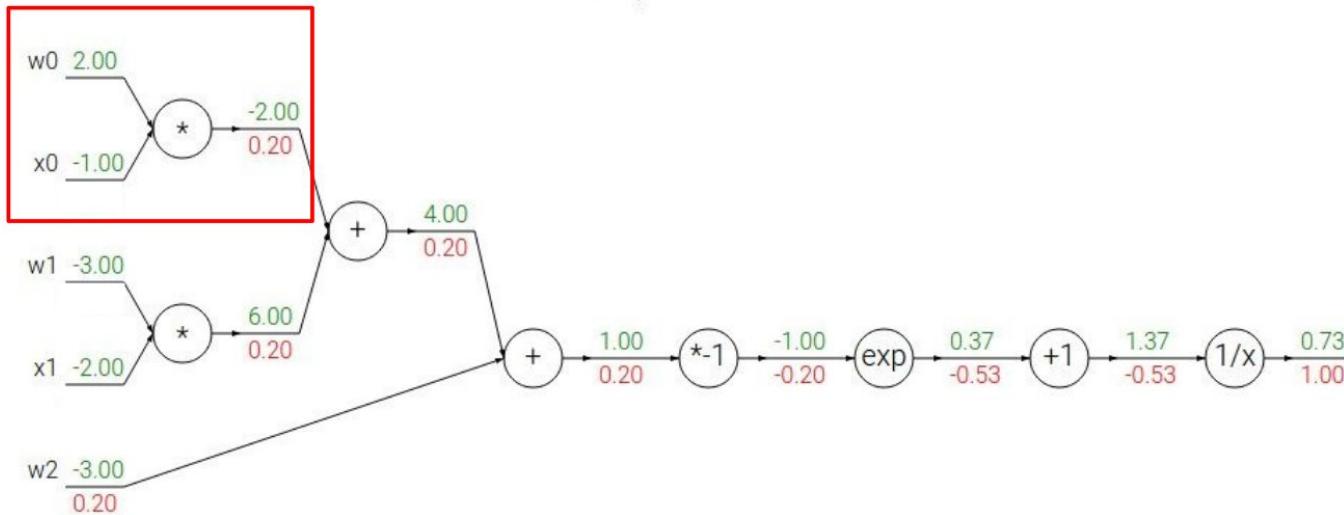
$\rightarrow$

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

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

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

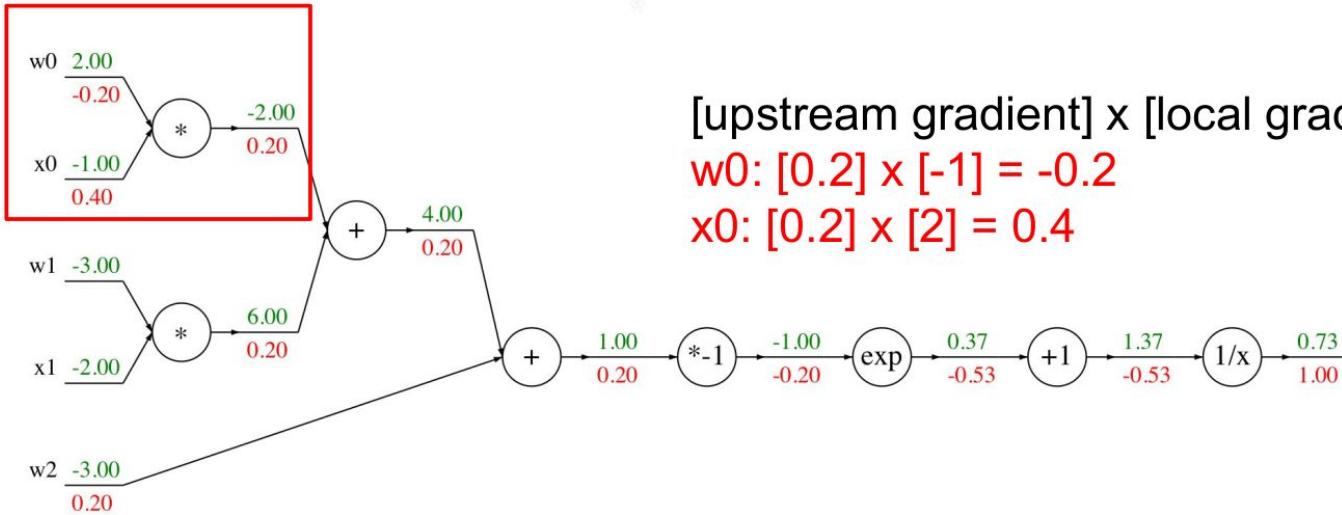
→

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

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

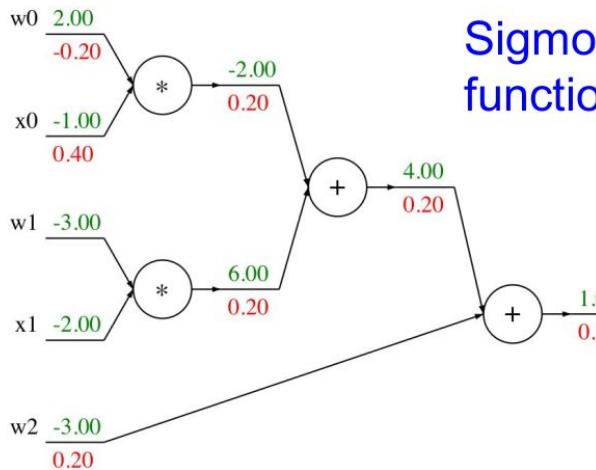
$$f_c(x) = c + x$$

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

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

Another example:

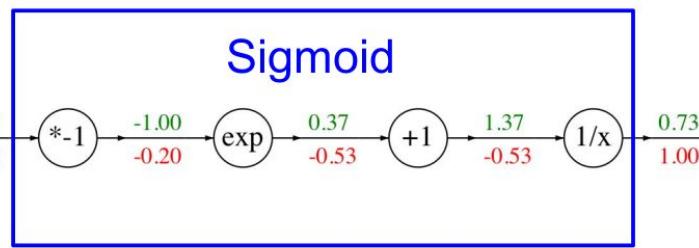
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Sigmoid  
function

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

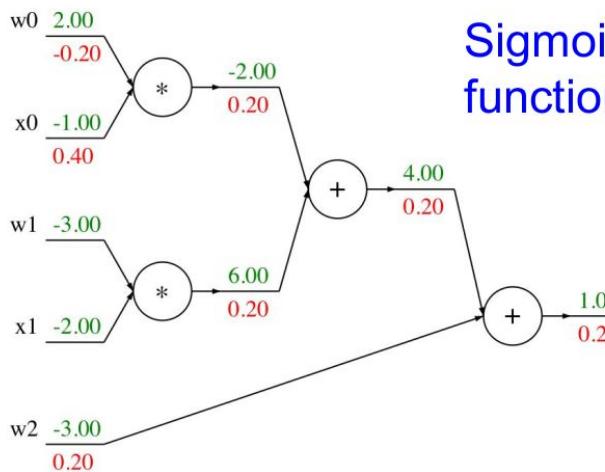
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



## Another example:

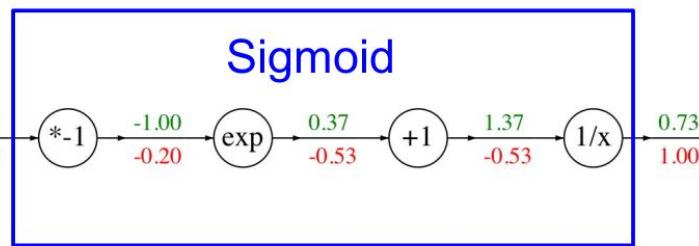
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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



## Sigmoid function

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

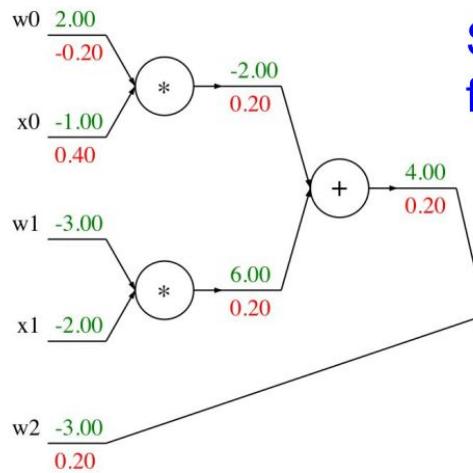


## Sigmoid local gradient:

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

Another example:

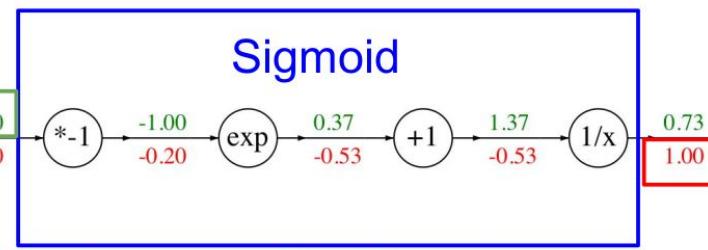
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



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!



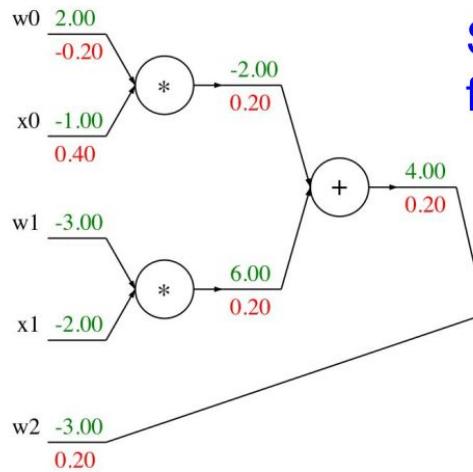
[upstream gradient] x [local gradient]  
 $[1.00] \times [(1 - 1/(1+e^{-1})) (1/(1+e^{-1}))] = 0.2$

Sigmoid local gradient:

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

Another example:

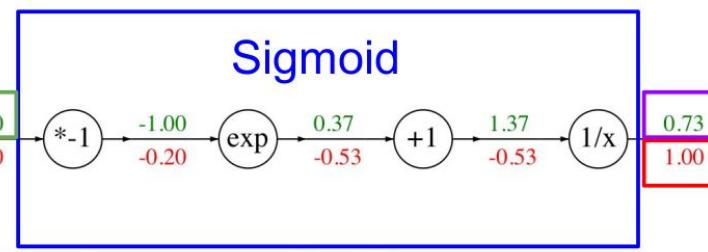
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



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!



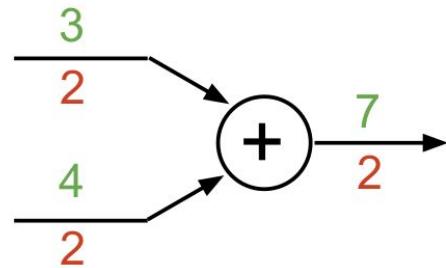
[upstream gradient] x [local gradient]  
 $[1.00] \times [(1 - 0.73)(0.73)] = 0.2$

Sigmoid local gradient:

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

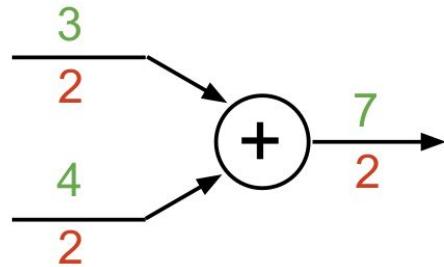
# Patterns in gradient flow

**add gate:** gradient distributor

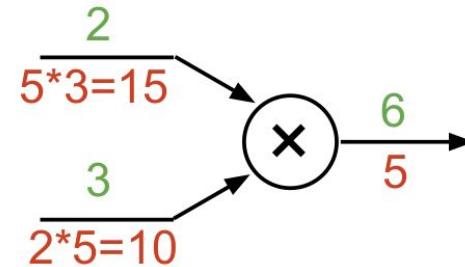


# Patterns in gradient flow

**add gate:** gradient distributor

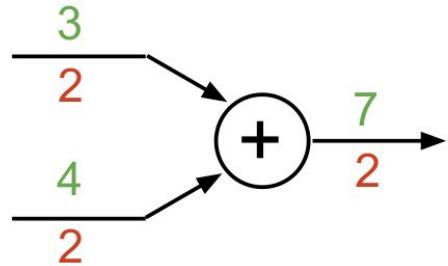


**mul gate:** “swap multiplier”

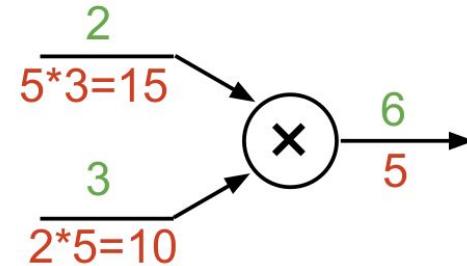


# Patterns in gradient flow

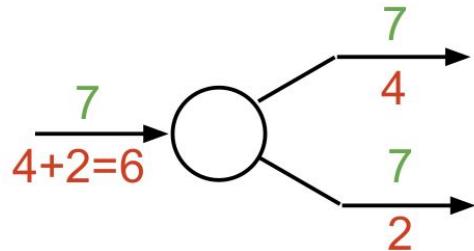
**add gate:** gradient distributor



**mul gate:** “swap multiplier”

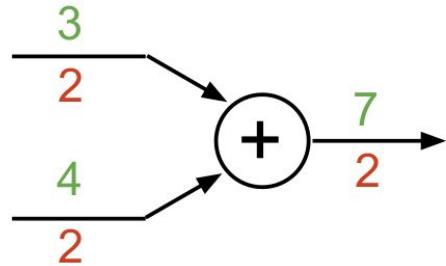


**copy gate:** gradient adder

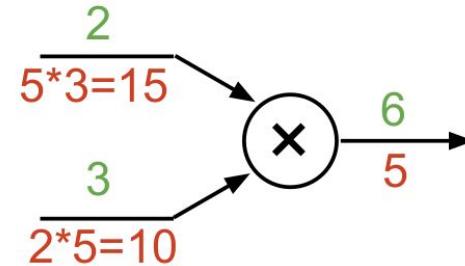


# Patterns in gradient flow

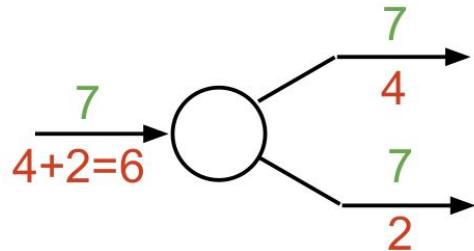
**add gate:** gradient distributor



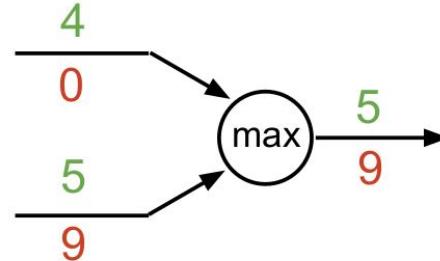
**mul gate:** “swap multiplier”



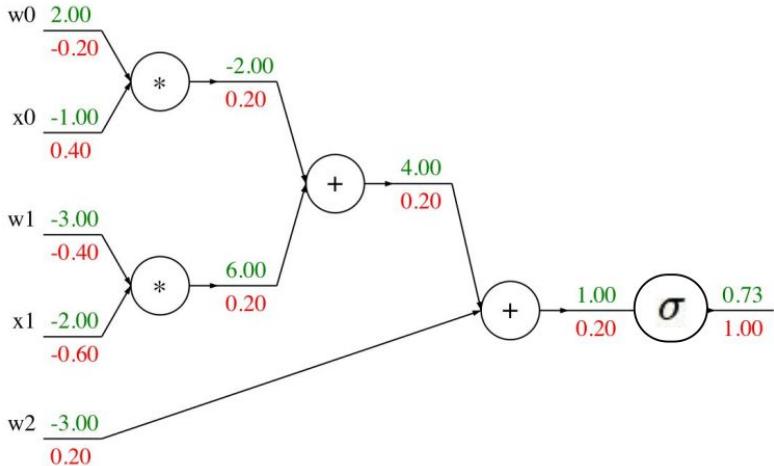
**copy gate:** gradient adder



**max gate:** gradient router



# Backprop Implementation: “Flat” code



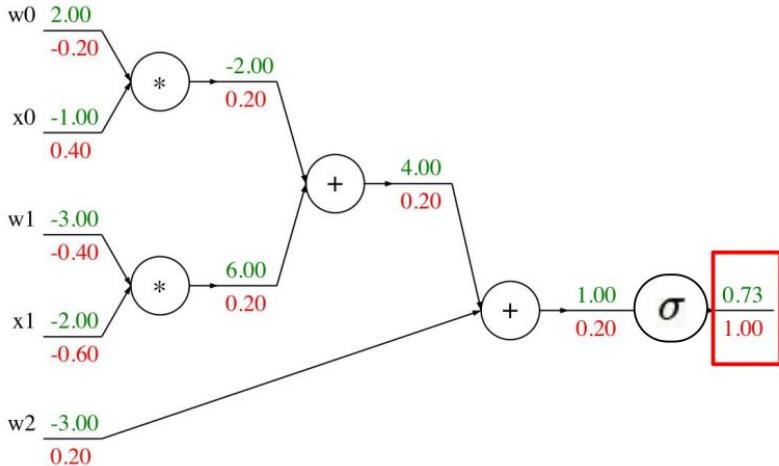
Forward pass:  
Compute output

```
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



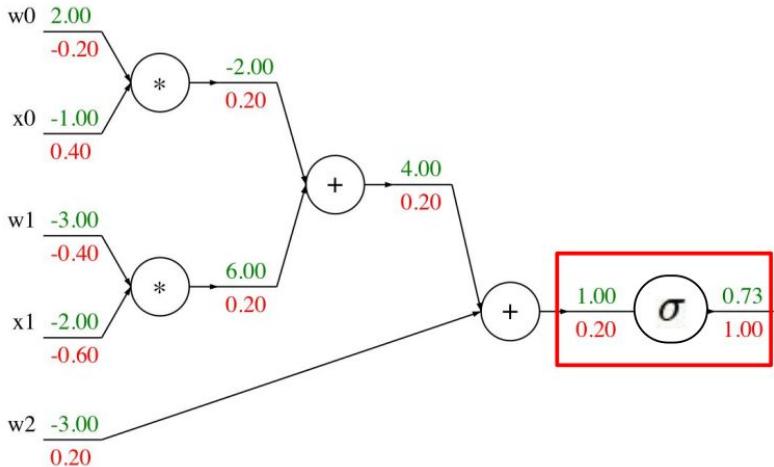
Forward pass:  
Compute output

```
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



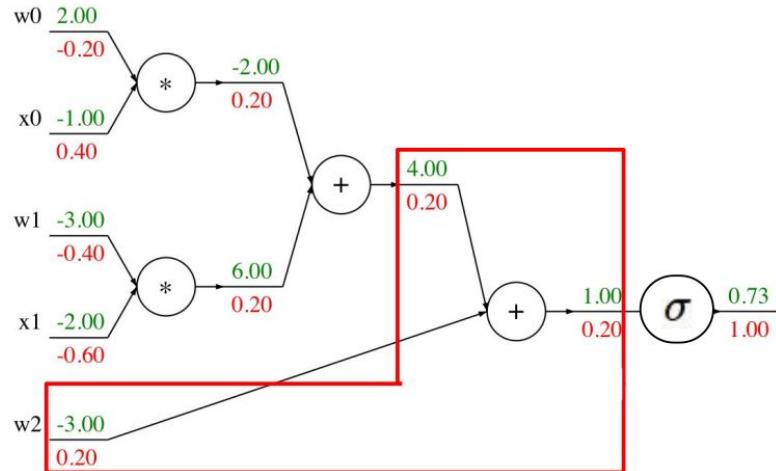
Forward pass:  
Compute output

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



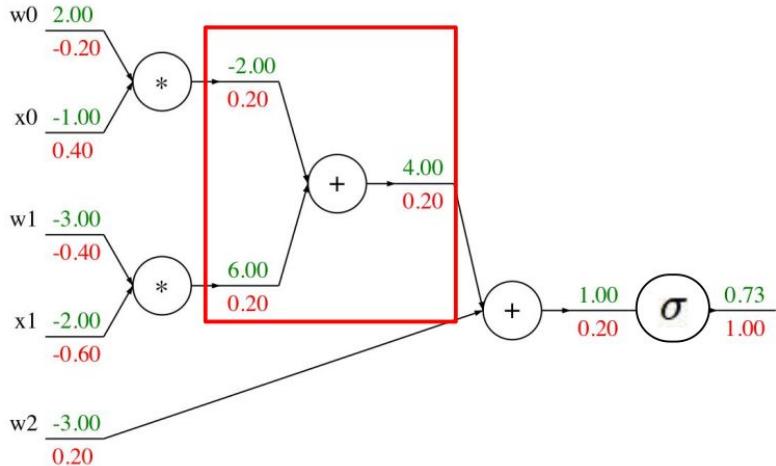
Forward pass:  
Compute output

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



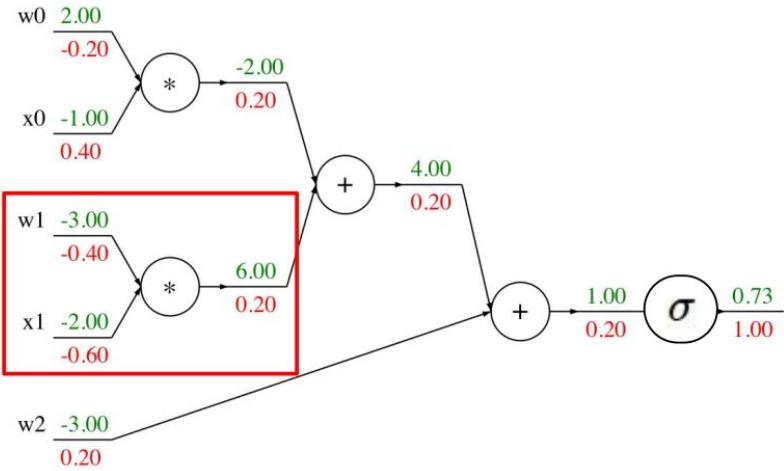
Forward pass:  
Compute output

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



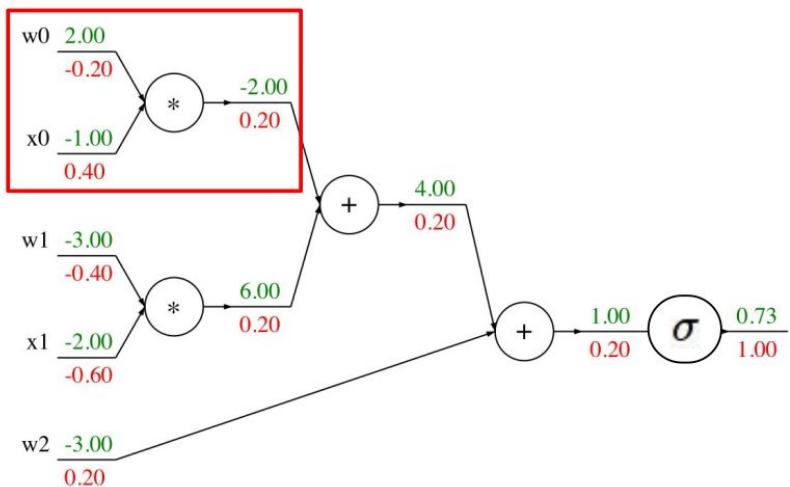
Forward pass:  
Compute output

```
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

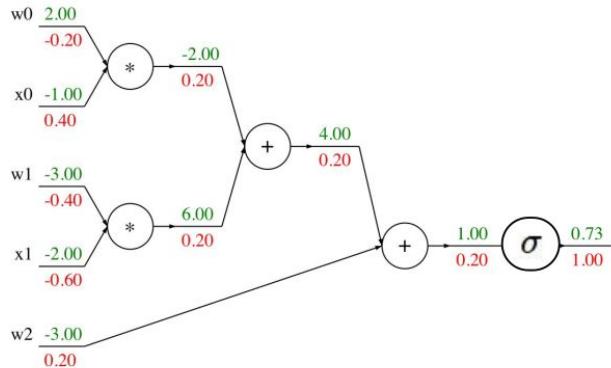
```
def f(w0, x0, w1, x1, w2):  
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

```
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

# Backprop Implementation: Modularized API

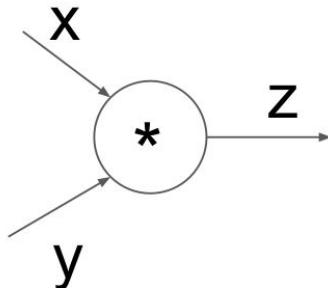
Graph (or Net) object (*rough pseudo code*)



```
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) ← Need to cache some values for use in backward  
        z = x * y  
        return z  
  
    @staticmethod  
    def backward(ctx, grad_z): ← Upstream gradient  
        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 cash 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	Projects 4
<a href="#">Tree: 517c7c98861 -</a>	<a href="#">pytorch / aten / src / THNN / generic /</a>	<a href="#">Create new file</a>	<a href="#">Upload files</a>
		<a href="#">Find file</a>	<a href="#">History</a>
 <a href="#">ezyang</a> and <a href="#">facebook-github-bot</a> Canonicalize all includes in PyTorch. (#14849) 	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="#">FeatureLPooling.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="#">ReLU6.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="#">VolumetricUpSamplingTrilinear.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

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

Forward actually defined elsewhere...

Backward

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

```
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;
            });
    });
}
```

Source

# Backpropagation with 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**:

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

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

# 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

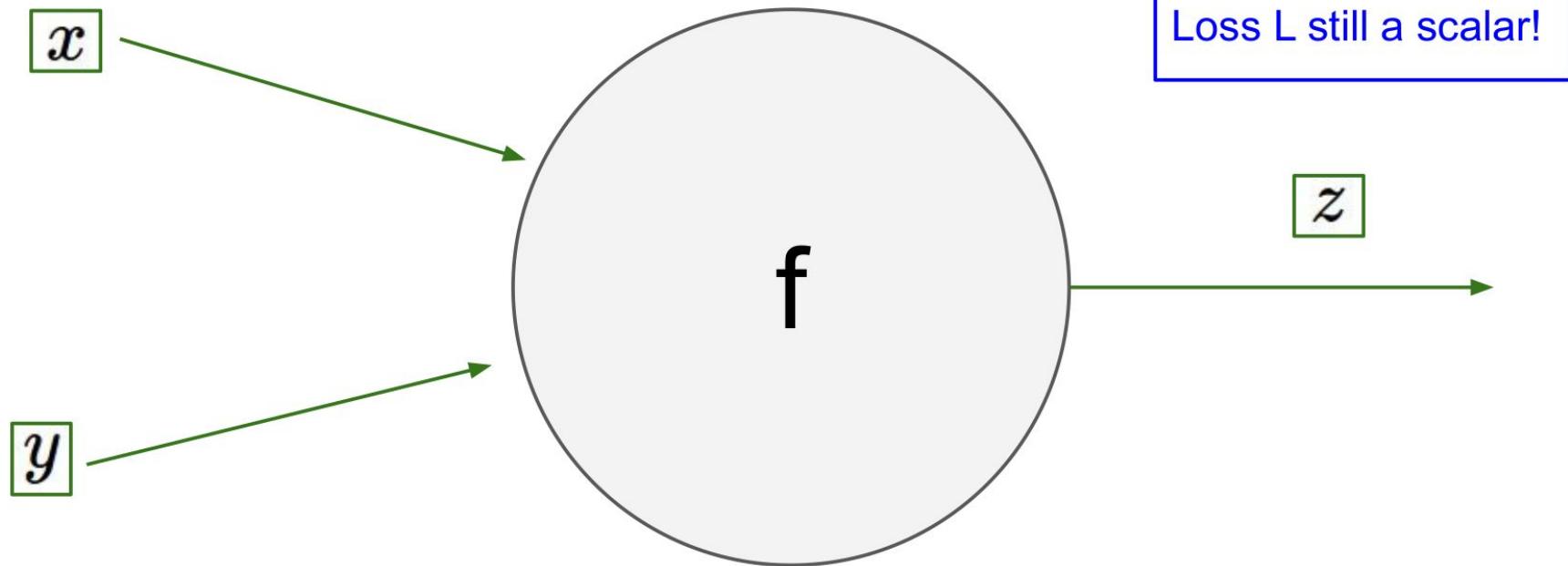
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

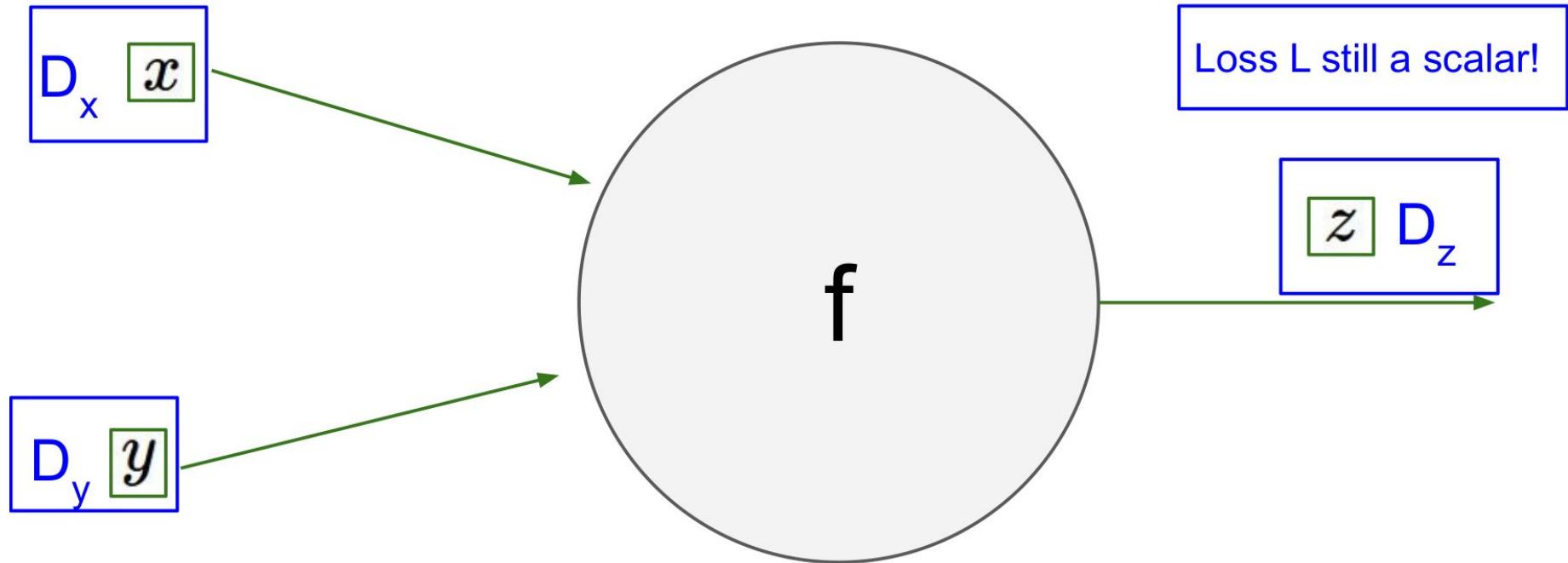
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

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

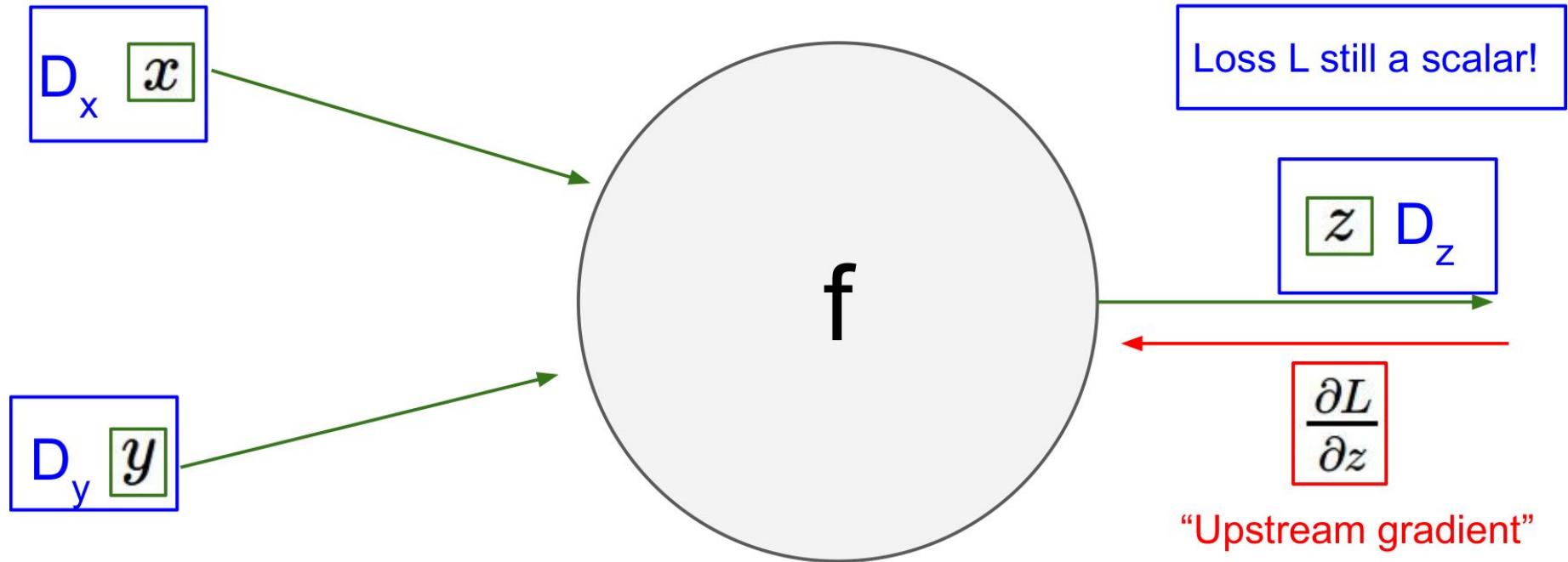
# Backprop with Vectors



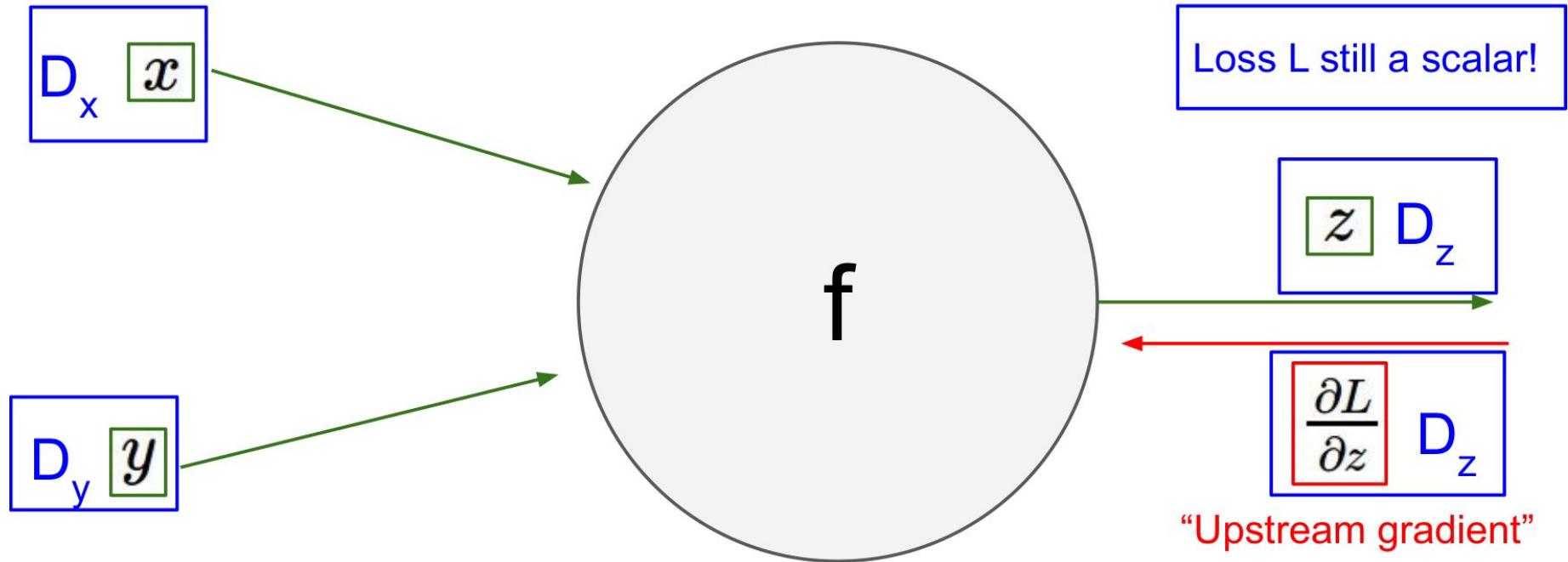
# Backprop with Vectors



# Backprop with Vectors



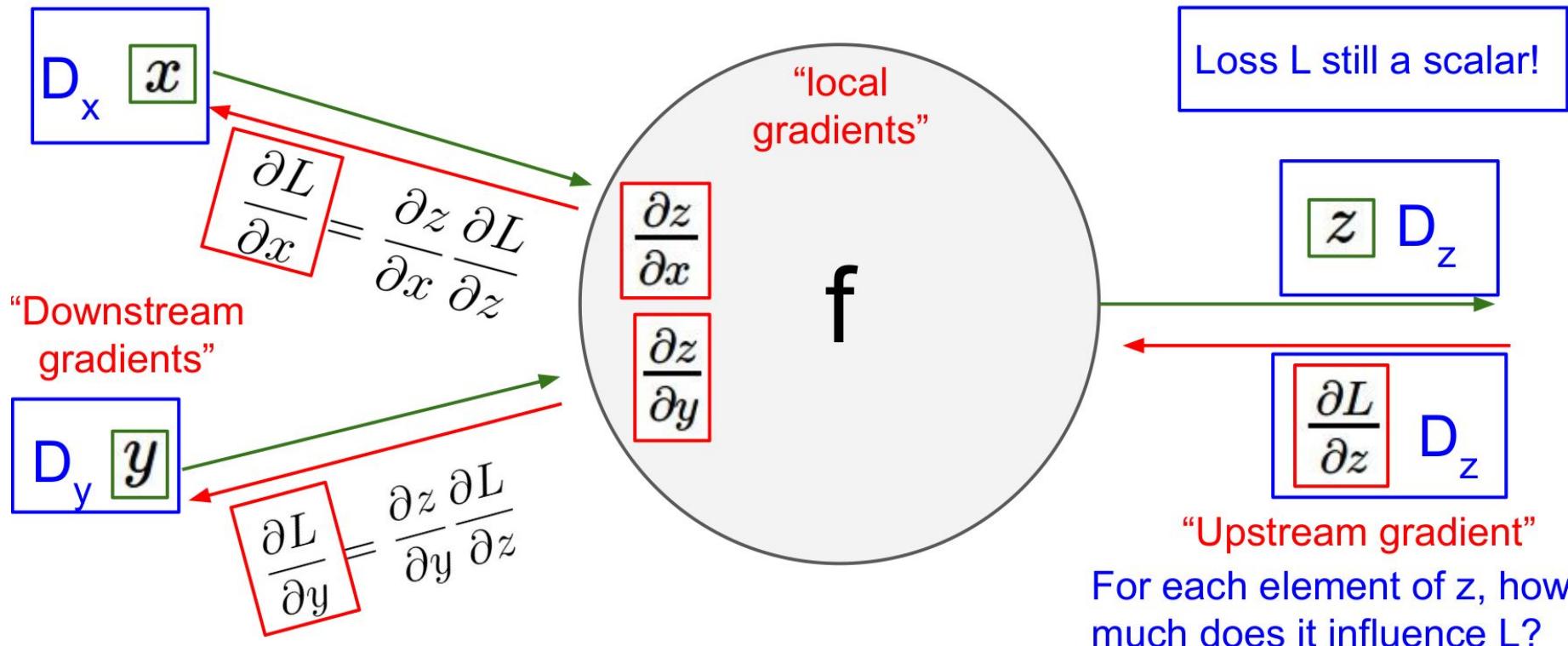
# Backprop with Vectors



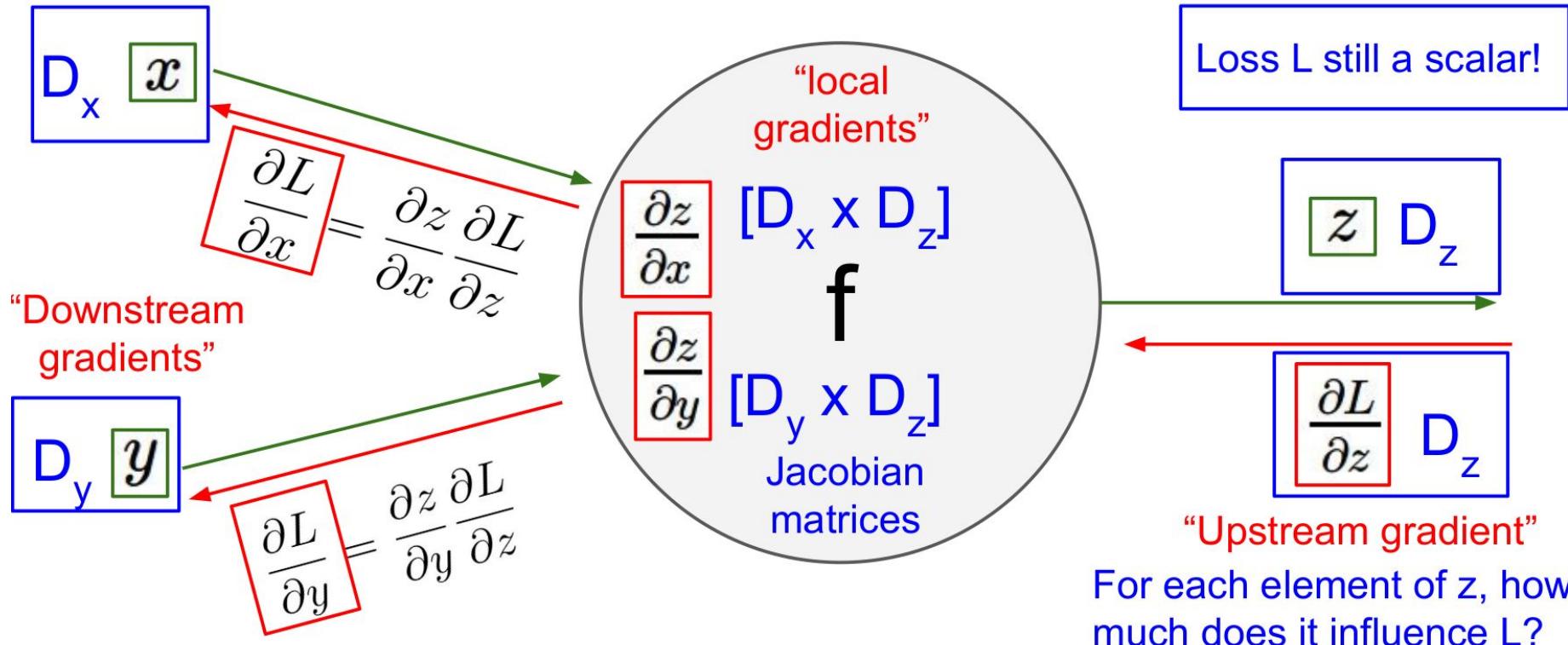
“Upstream gradient”

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

# Backprop with Vectors

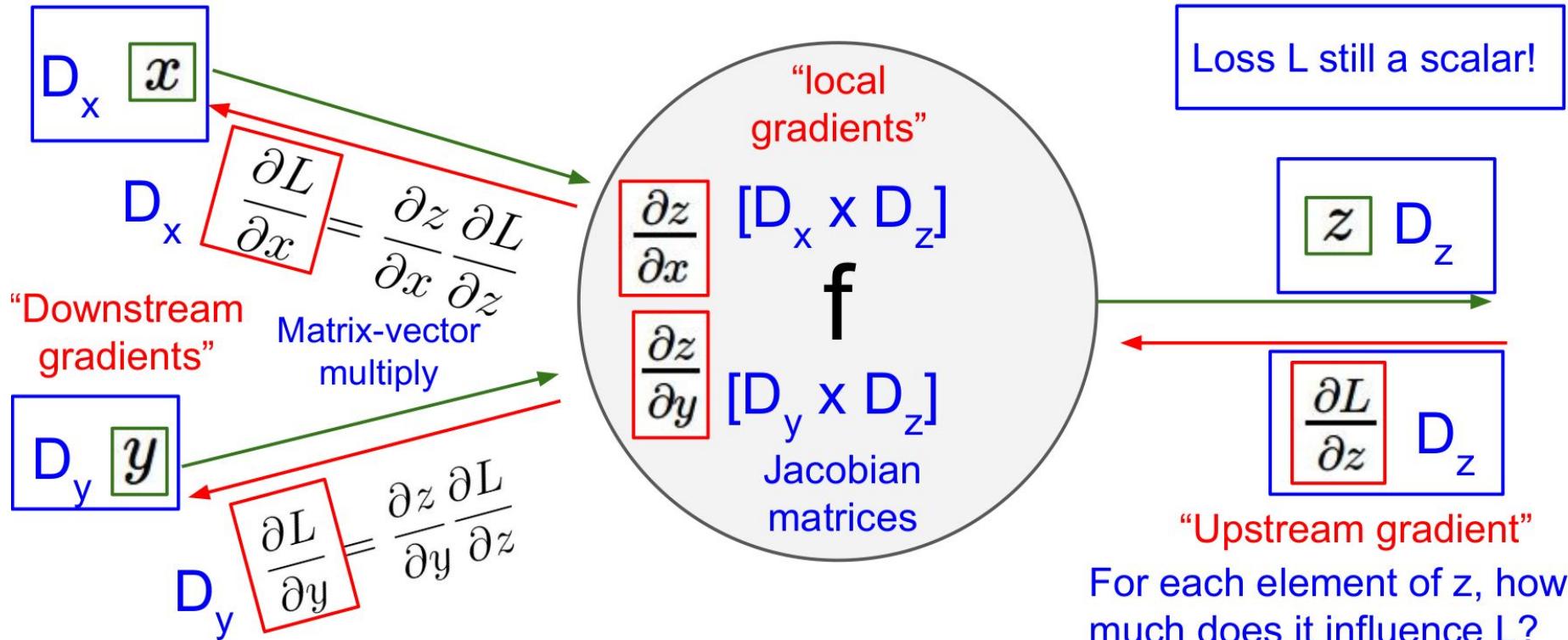


# Backprop with Vectors

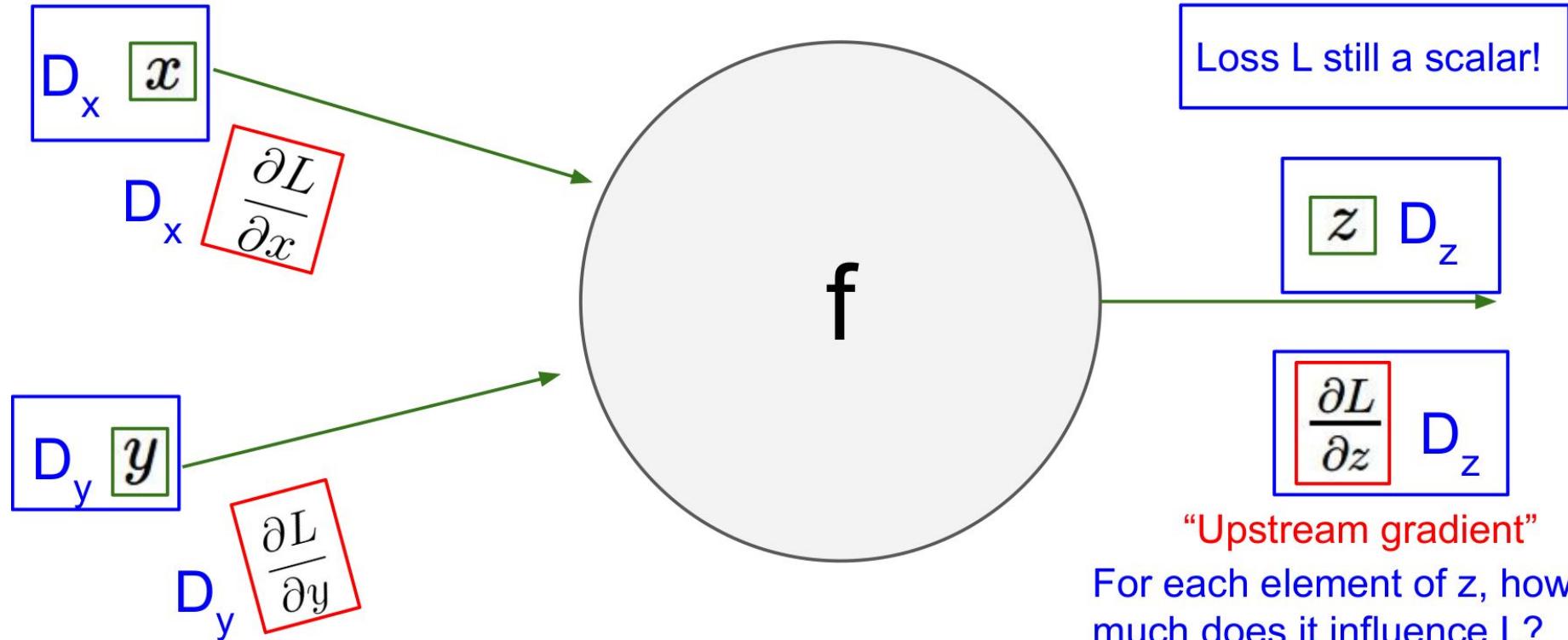


"Upstream gradient"  
For each element of  $z$ , how  
much does it influence  $L$ ?

# Backprop with Vectors



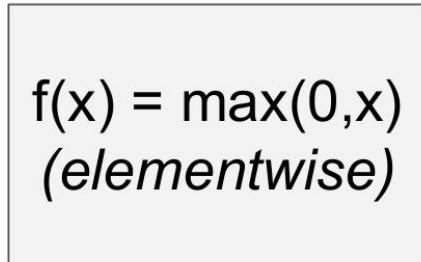
Gradients of variables wrt loss have same dims as the original variable



# Backprop with Vectors

4D input x:

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



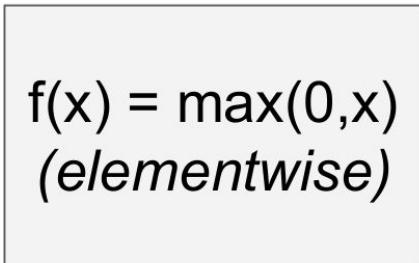
4D output z:

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

# Backprop with Vectors

4D input  $x$ :

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



4D output  $z$ :

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

4D  $dL/dz$ :

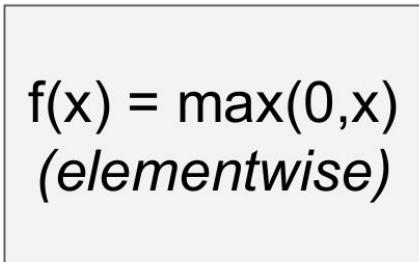
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream  
gradient

# Backprop with Vectors

4D input  $x$ :

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



4D output  $z$ :

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

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

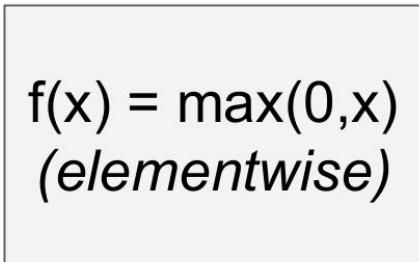
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream  
gradient

# Backprop with Vectors

4D input  $x$ :

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



4D output  $z$ :

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

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

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

4D  $dL/dz$ :

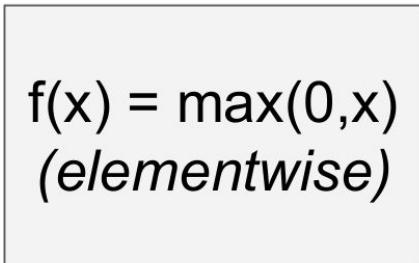
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream  
gradient

# Backprop with Vectors

4D input  $x$ :

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



4D output  $z$ :

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

4D  $dL/dx$ :

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

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

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

4D  $dL/dz$ :

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

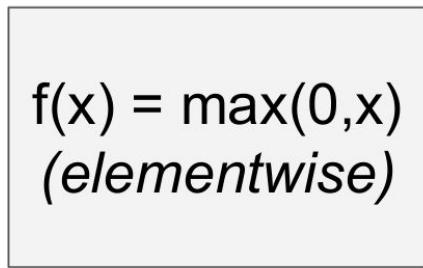
Upstream  
gradient

# Backprop with Vectors

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

4D input  $x$ :

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



4D output  $z$ :

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

4D  $dL/dx$ :

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

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

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

4D  $dL/dz$ :

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

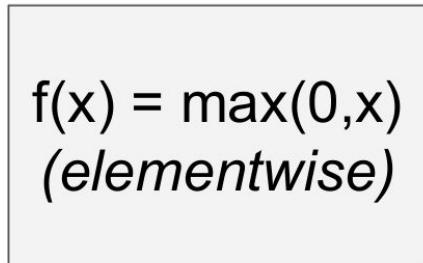
Upstream  
gradient

# Backprop with Vectors

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

4D input  $x$ :

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



4D output  $z$ :

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

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

# 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
```

# 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

# 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

# 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

Calculate the analytical gradients

# 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

Calculate the analytical gradients

Gradient descent