

# Lecture 5: Backpropagation and Neural Networks I

# Where we are ...

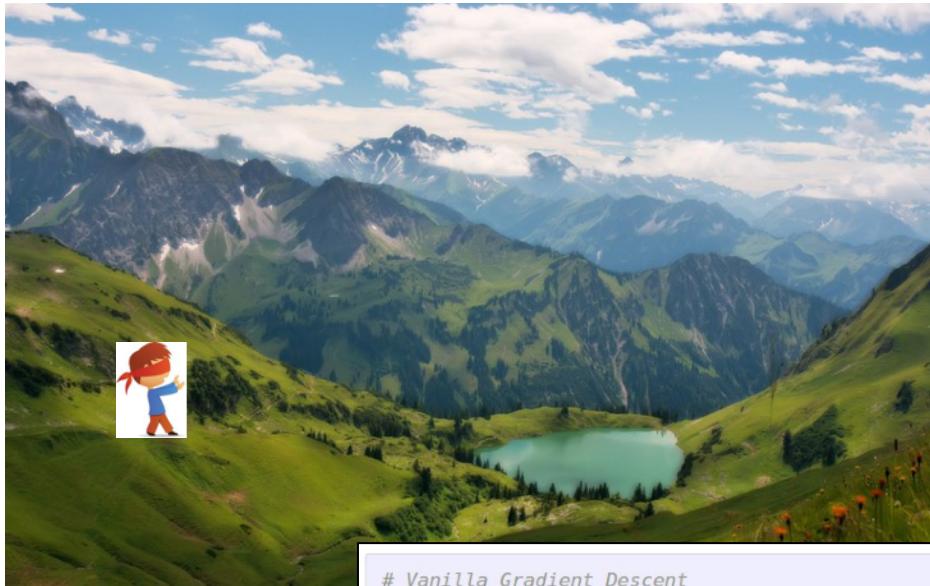
$$s = f(x; W) = Wx \quad \text{scores function}$$

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

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

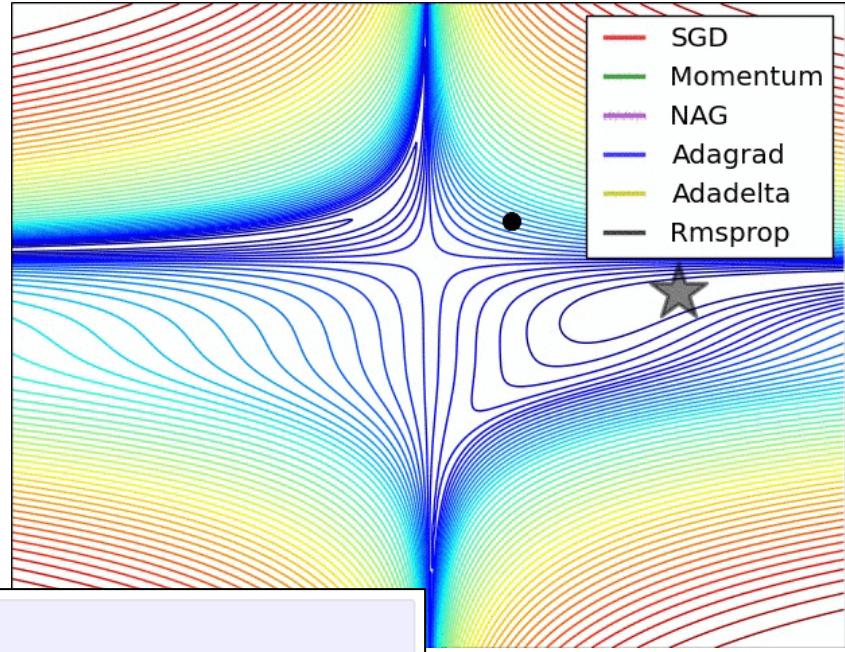
want  $\boxed{\nabla_W L}$

# Optimization



# Vanilla Gradient Descent

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



(image credits  
to Alec Radford)

# Gradient Descent

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

**Numerical gradient:** slow :, approximate :, easy to write :)

**Analytic gradient:** fast :, exact :, error-prone :(

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

# Overview of where we're going

- We want to **evaluate** the gradient of a Loss function  $L(x, W, \dots)$ , with respect to the parameters (weights) of a neural network, at the “point” represented by the arguments to the function  $(x, W, \dots)$ .
  - We are **not interested in an algebraic expression for the gradient**, but rather only in the **evaluation of that gradient at the current value of the function arguments**.

Consider the function

$$z(x, y) = x^2 + y^2,$$

and suppose we are interested in evaluating the gradient of this function at the point

$$(x, y) = (5, 3).$$

Evaluate the gradient:

$$\frac{\partial z}{\partial x} = 2x.$$

$$\frac{\partial z}{\partial y} = 2y.$$

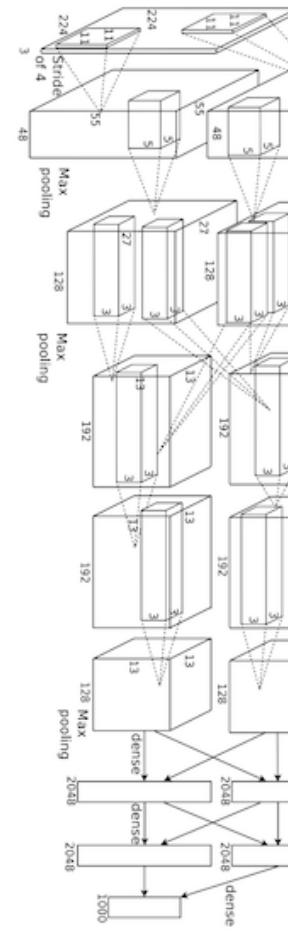
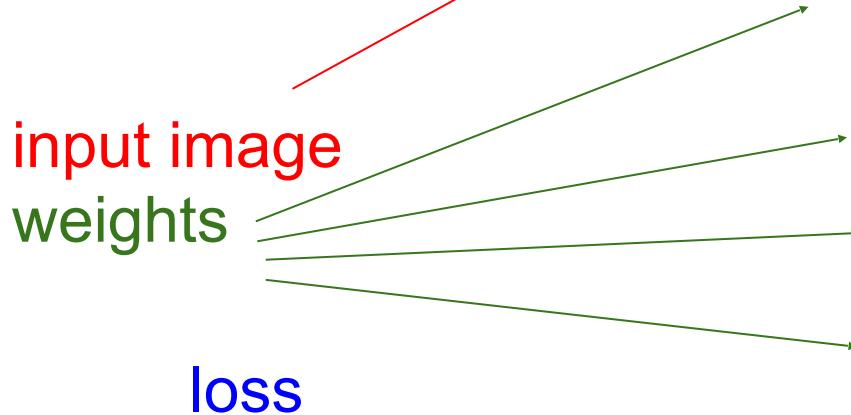
The algebraic expression of the gradient is just the collection of these partials into a “vector”:

$$\nabla z = \begin{bmatrix} 2x \\ 2y \end{bmatrix}. \quad \text{← } \text{Don't care about this}$$

The evaluation of this gradient at the point  $(x, y) = (5, 3)$  is simply

$$\nabla z(5, 3) = \begin{bmatrix} 2 \times 5 \\ 2 \times 3 \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \end{bmatrix}. \quad \text{← } \text{Do care about this}$$

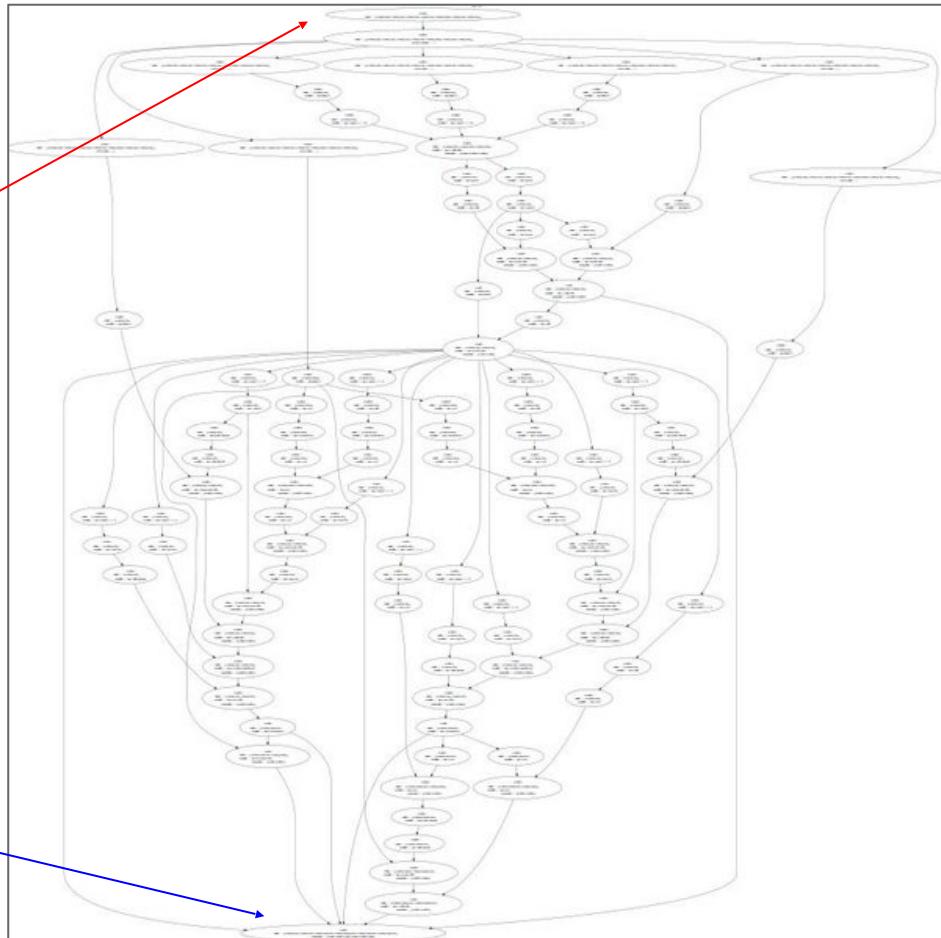
# Convolutional Network (AlexNet)



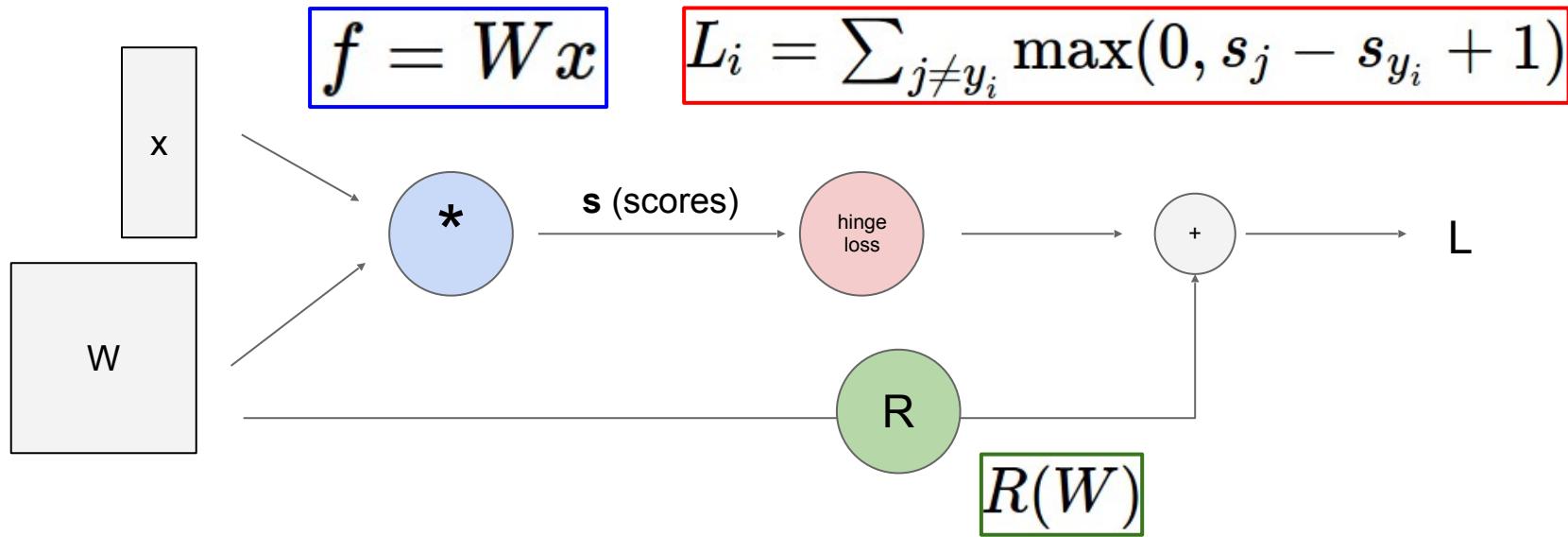
# Neural Turing Machine

input tape

loss



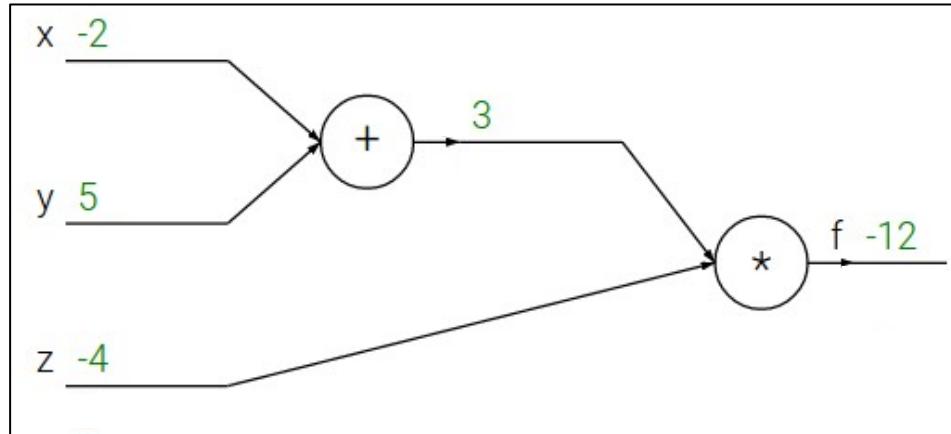
# Computational Graph



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

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

**Forward pass:** evaluating each expression in the computational graph from the inputs to the final output (or outputs). The results of each forward step are shown in green.



```
# set some inputs
x = -2; y = 5; z = -4

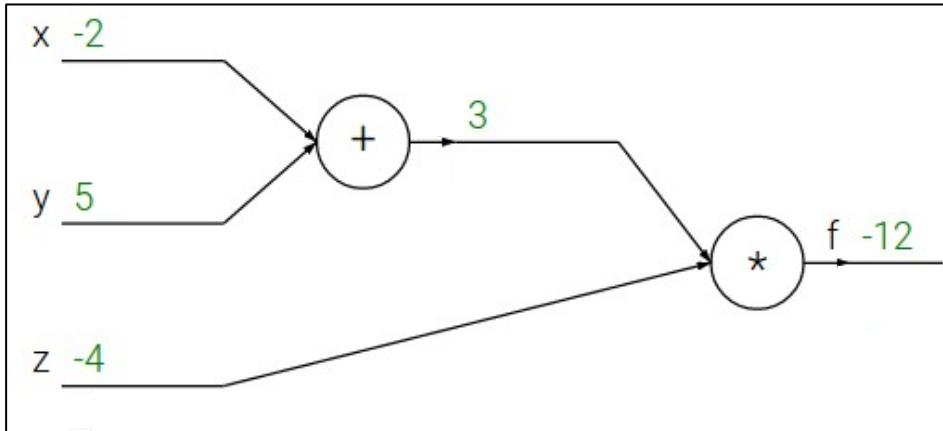
# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdxdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdxdy = 1.0 * dfdq # dq/dy = 1
```

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

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

**Backward pass:** evaluating the partial derivative of each **parameter** or **intermediate result** in the computational graph from the outputs back to the inputs. The results of each backward step are shown in red.



Goal is to calculate

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

**evaluated at** the point

$$[x = -2, y = 5, z = -4].$$

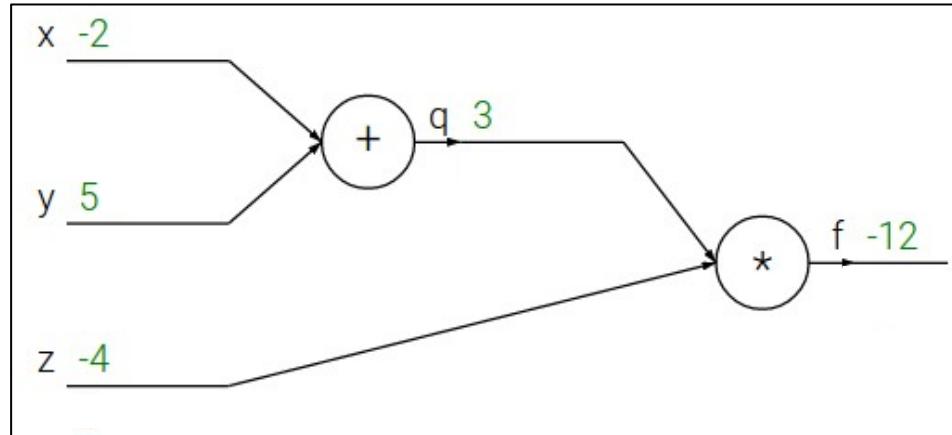
$$f(x, y, z) = (x + y)z$$

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



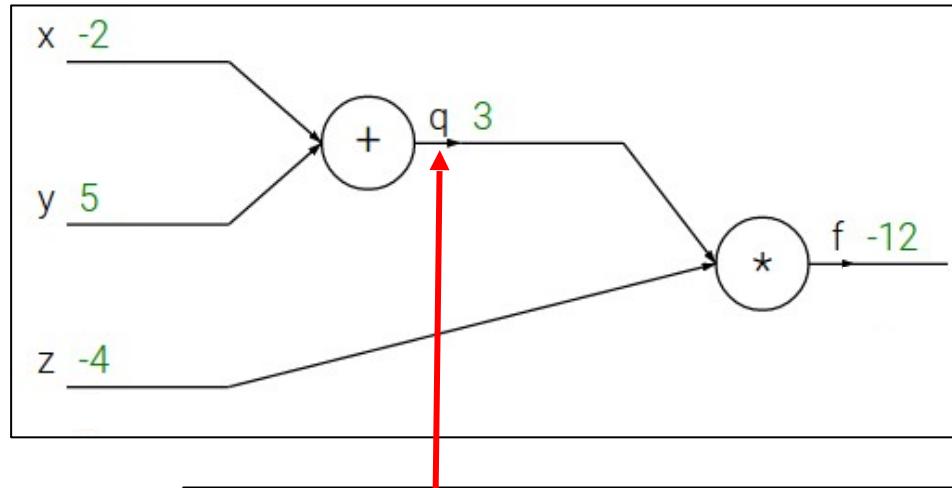
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Important: name the intermediate quantities

Compute some **local partial derivatives**.  
These are derivatives of the outputs of a node with respect to the inputs....

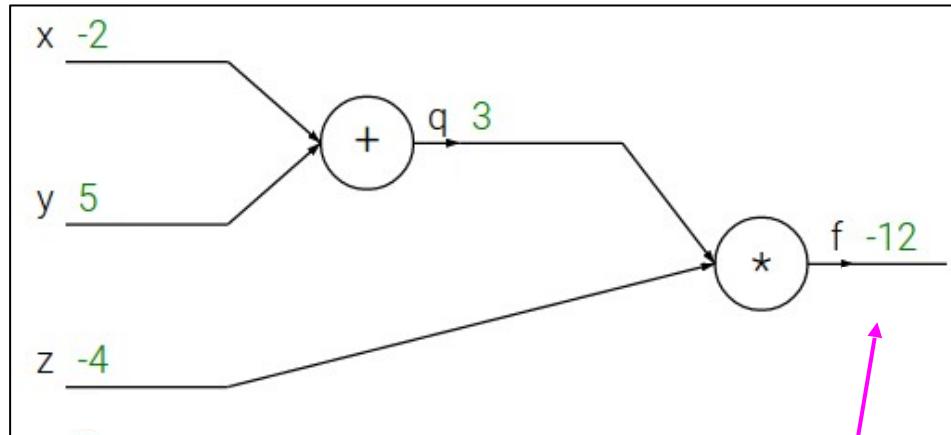
$$f(x, y, z) = (x + y)z$$

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



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

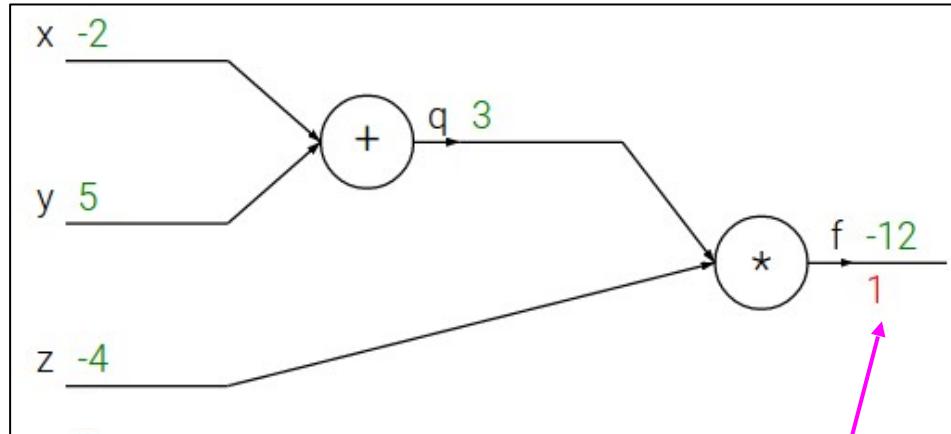
$$f(x, y, z) = (x + y)z$$

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



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

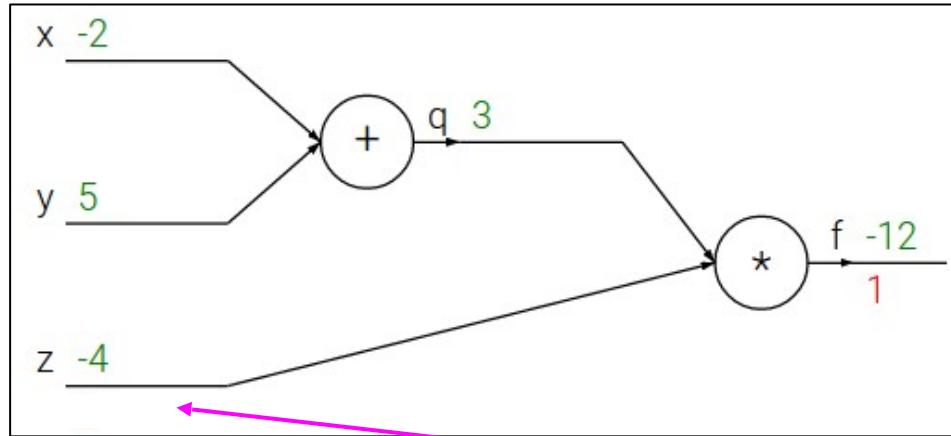
$$f(x, y, z) = (x + y)z$$

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



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

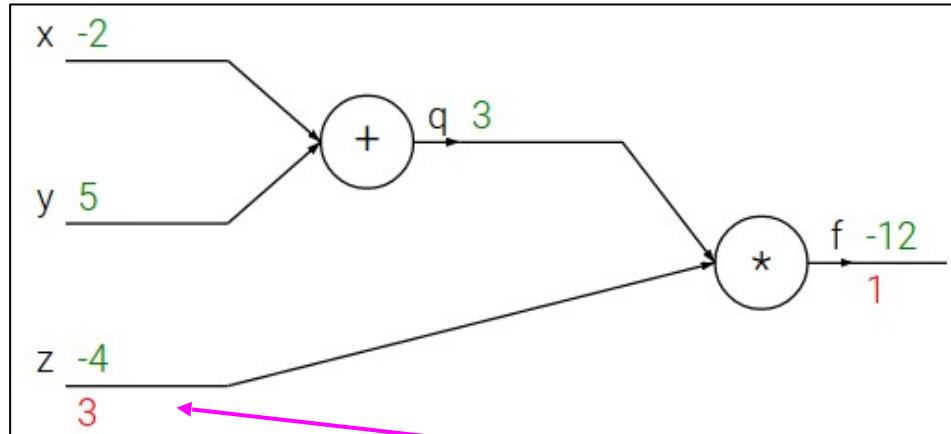
$$f(x, y, z) = (x + y)z$$

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



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

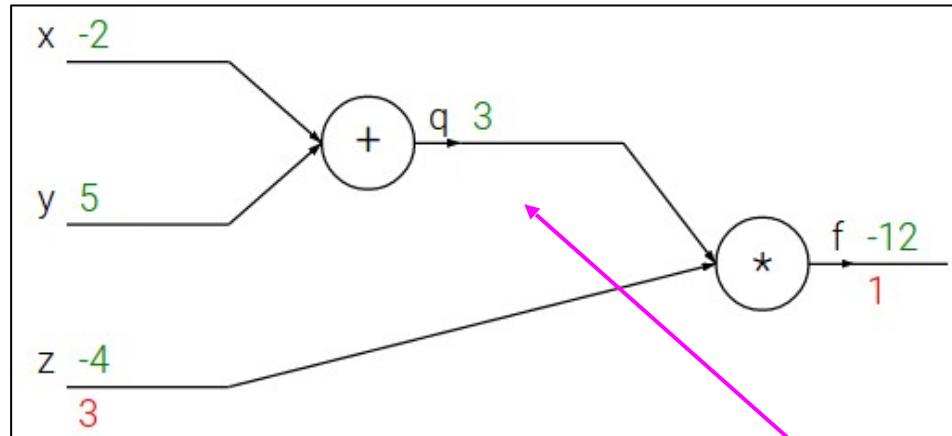
$$f(x, y, z) = (x + y)z$$

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



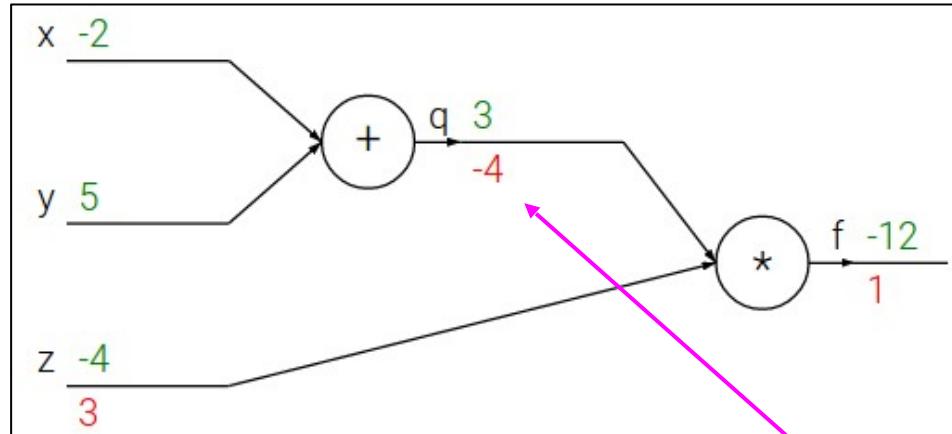
$$f(x, y, z) = (x + y)z$$

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



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

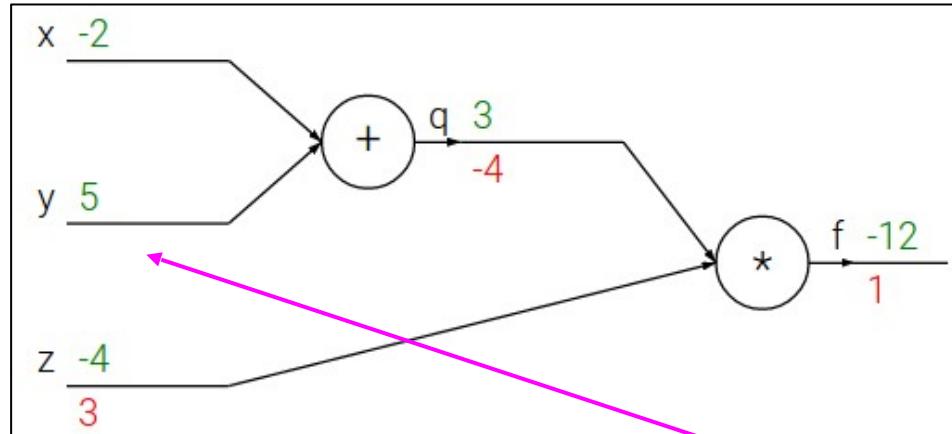
$$f(x, y, z) = (x + y)z$$

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



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

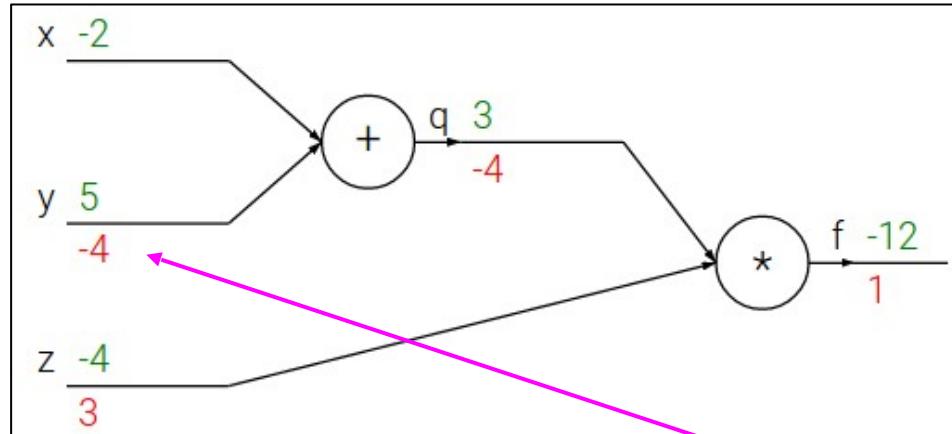
$$f(x, y, z) = (x + y)z$$

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:

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

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

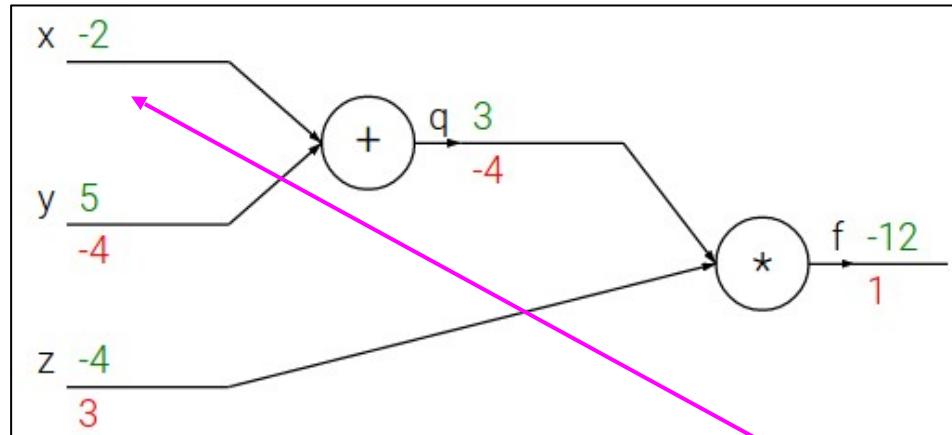
$$f(x, y, z) = (x + y)z$$

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



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

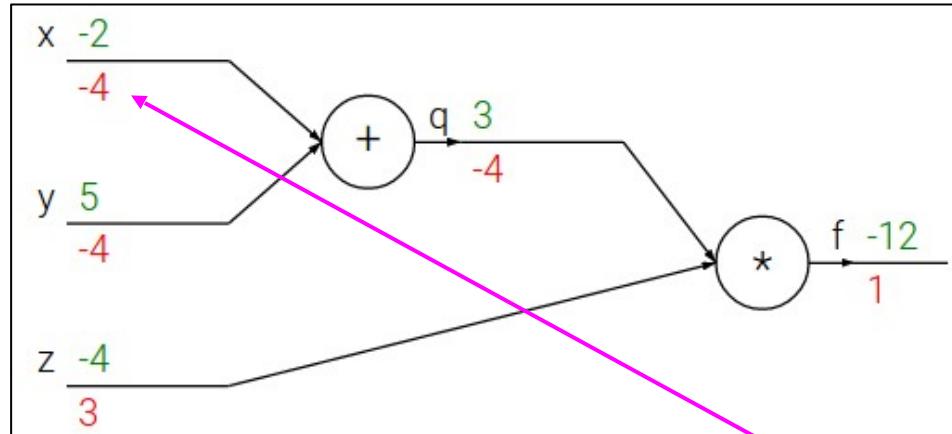
$$f(x, y, z) = (x + y)z$$

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:

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

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

```

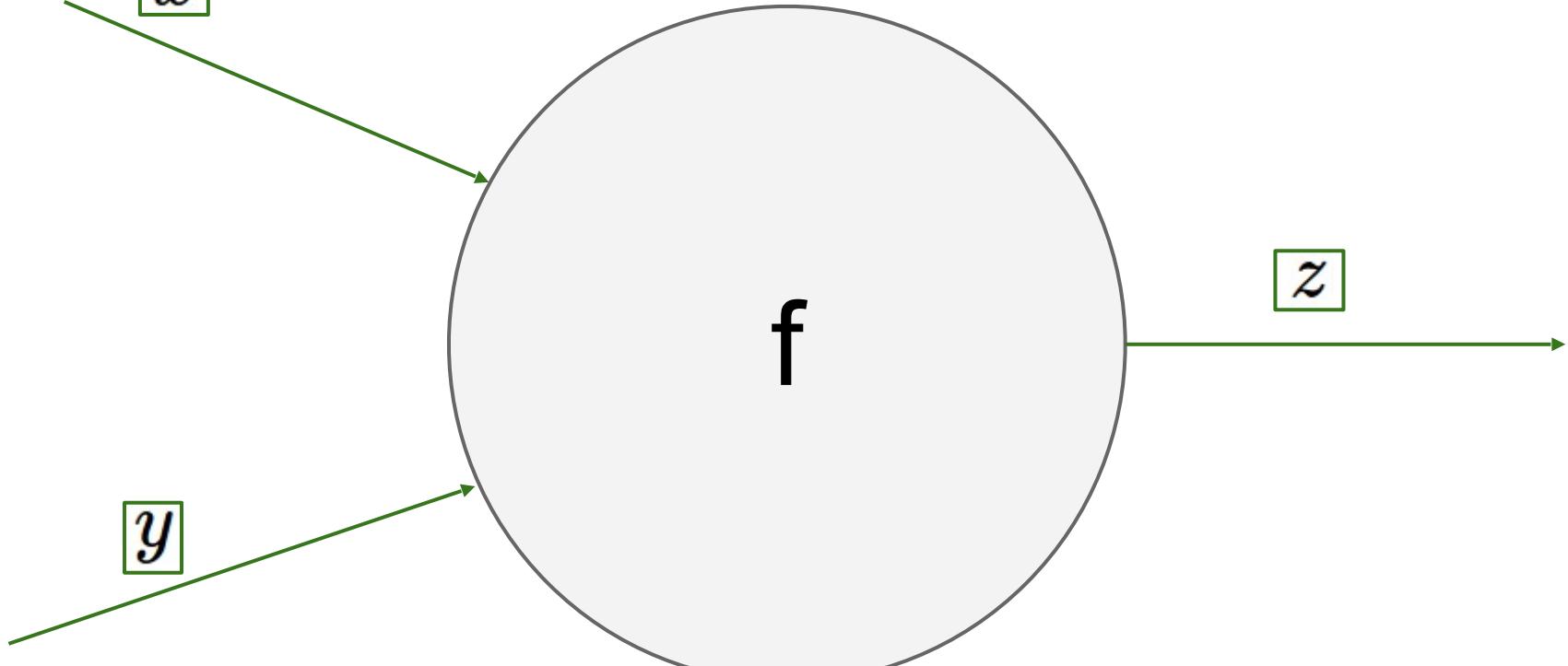
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

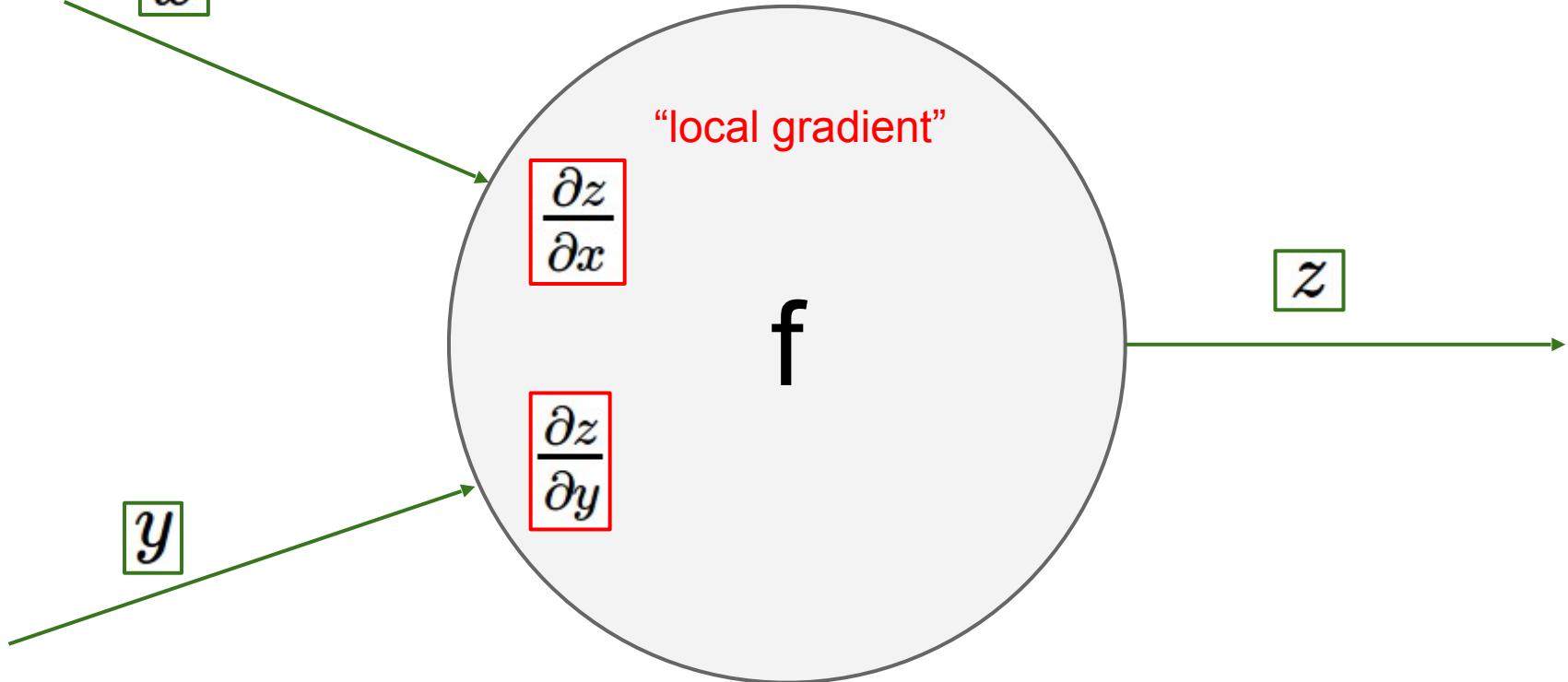
# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdxdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdxdy = 1.0 * dfdq # dq/dy = 1

```

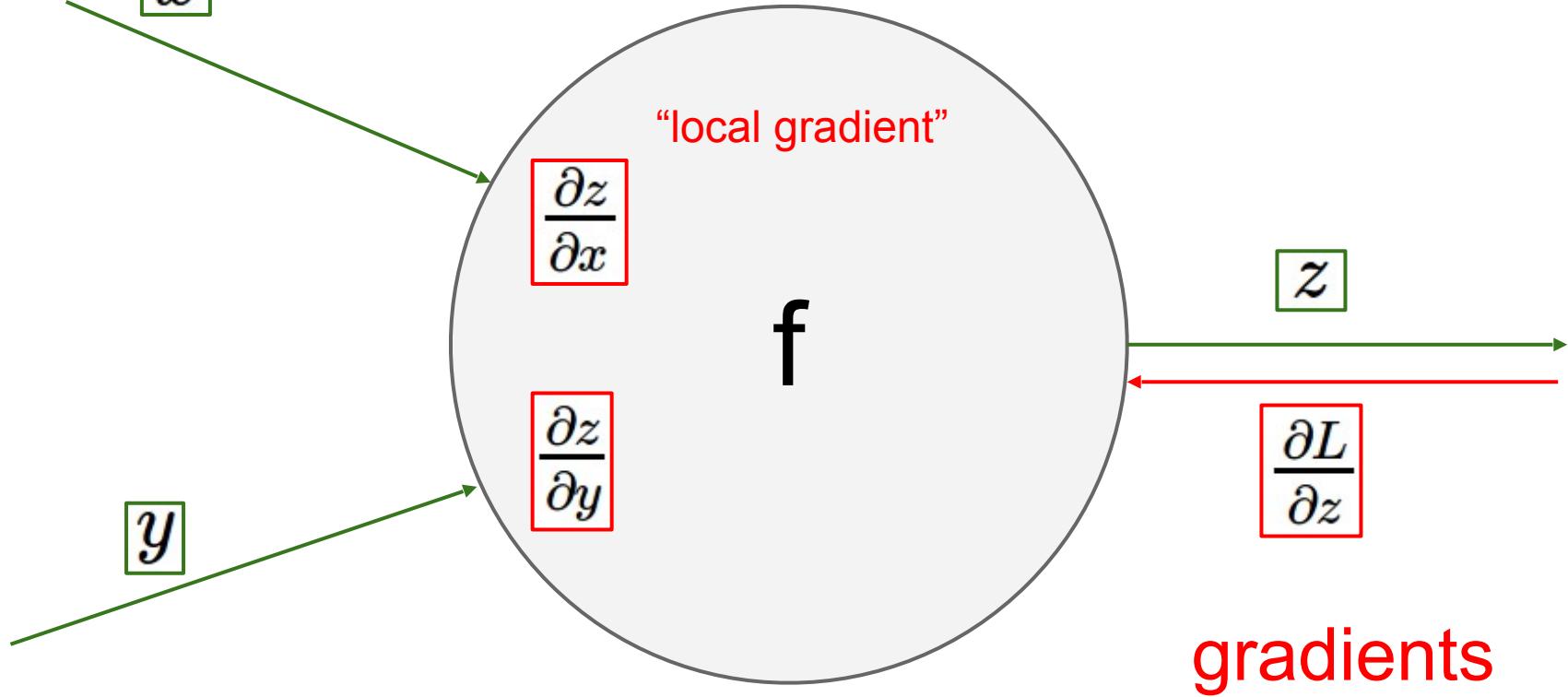
# activations



# activations

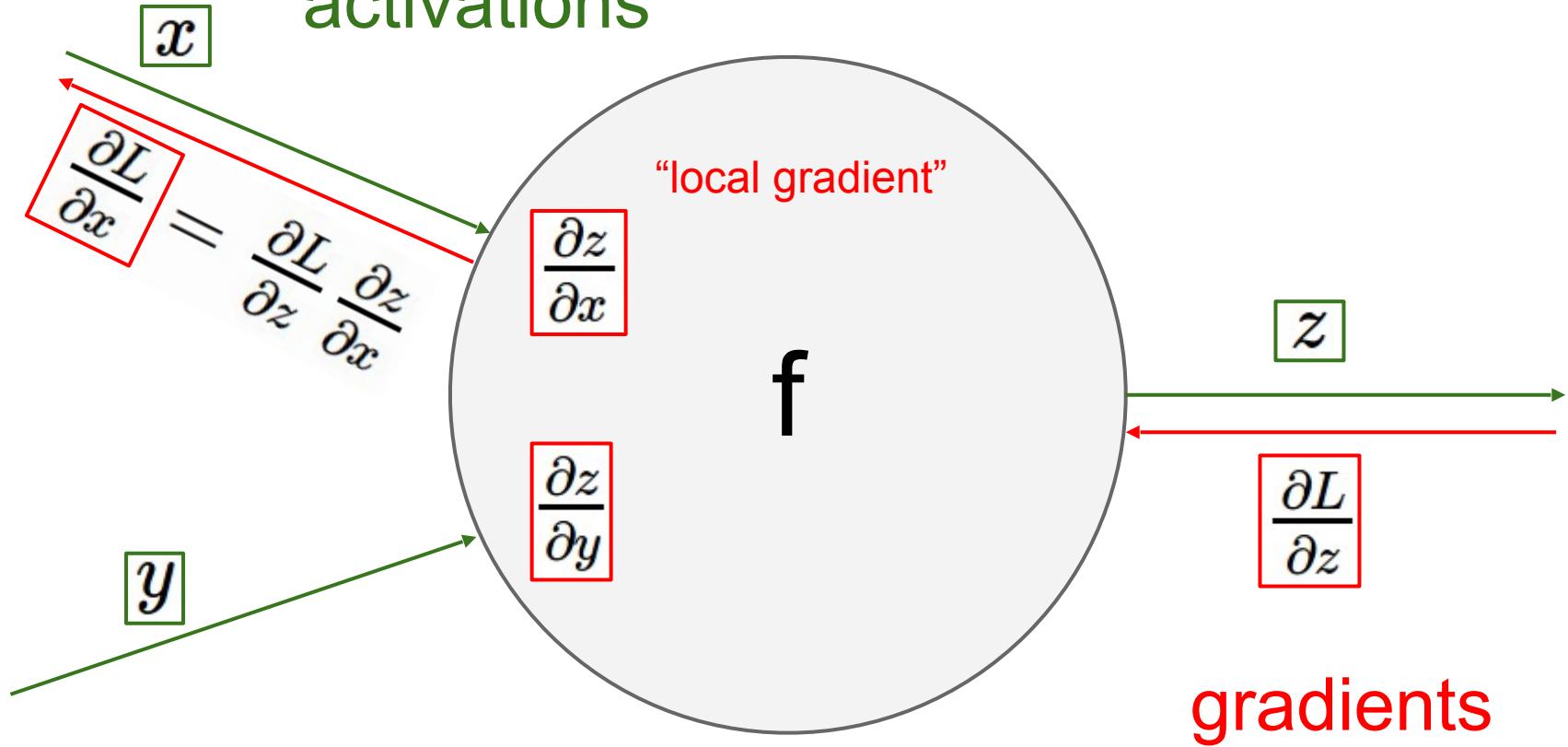


# activations

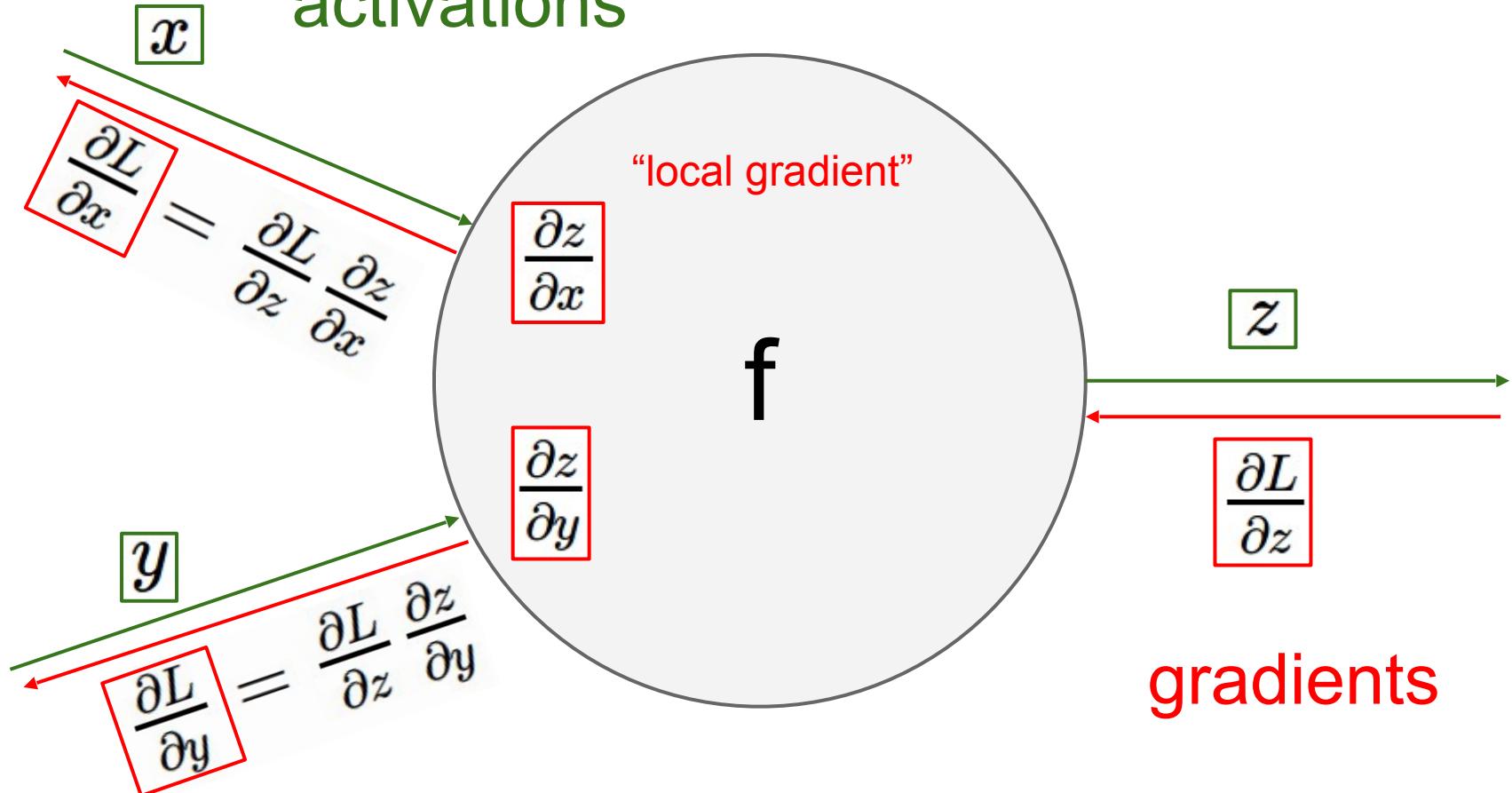


gradients

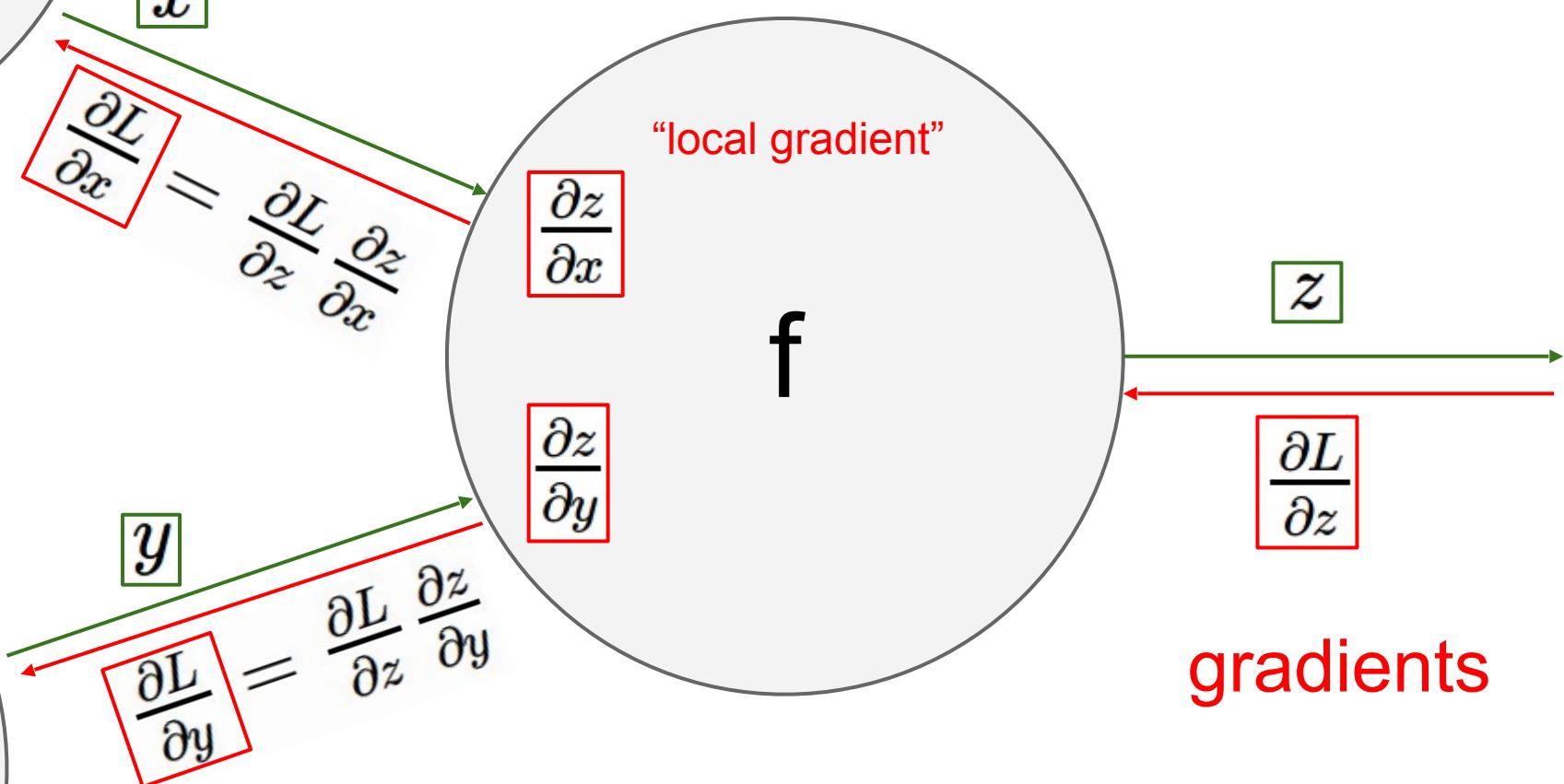
# activations



# activations



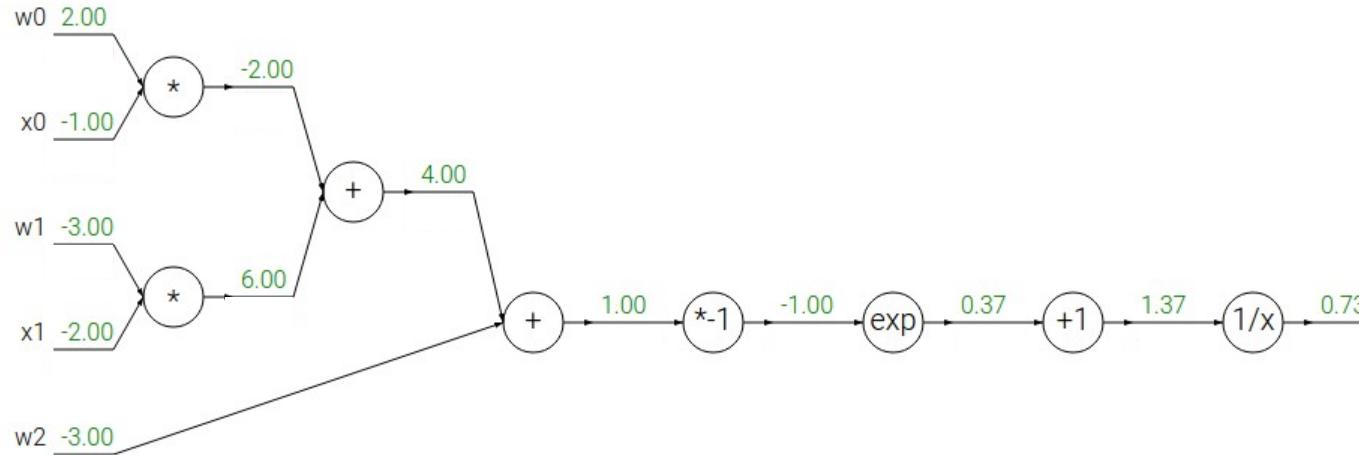
# activations



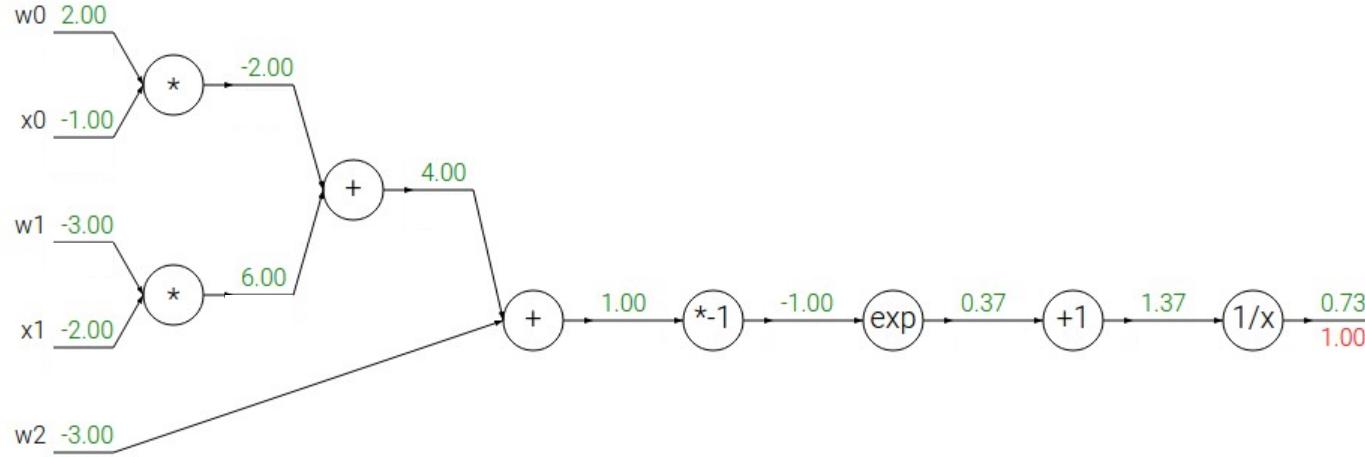
Another example:

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

“sigmoid function”



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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

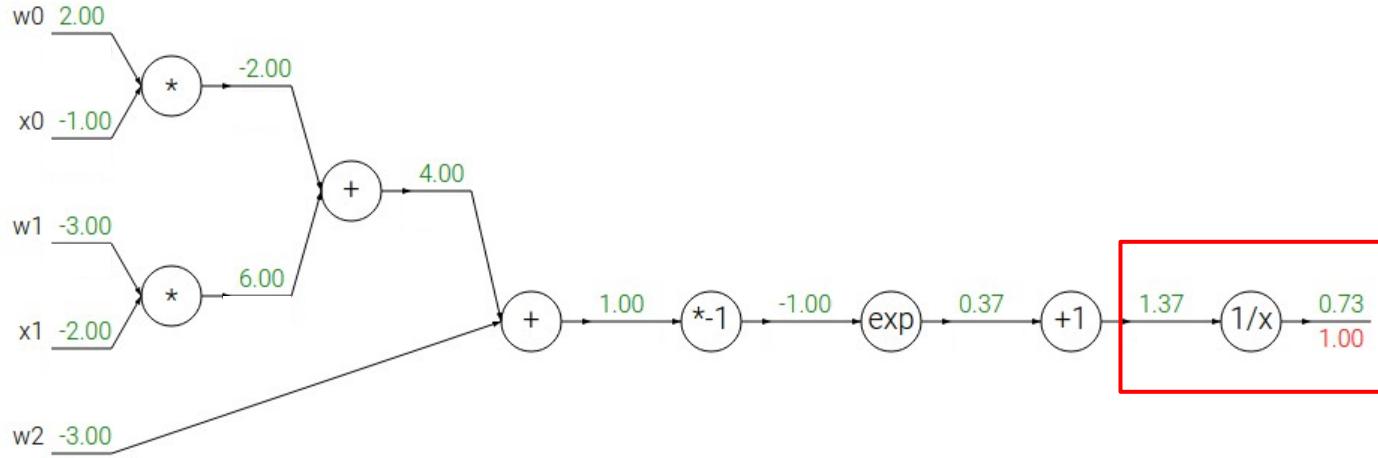
$$f_c(x) = c + x$$

→

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

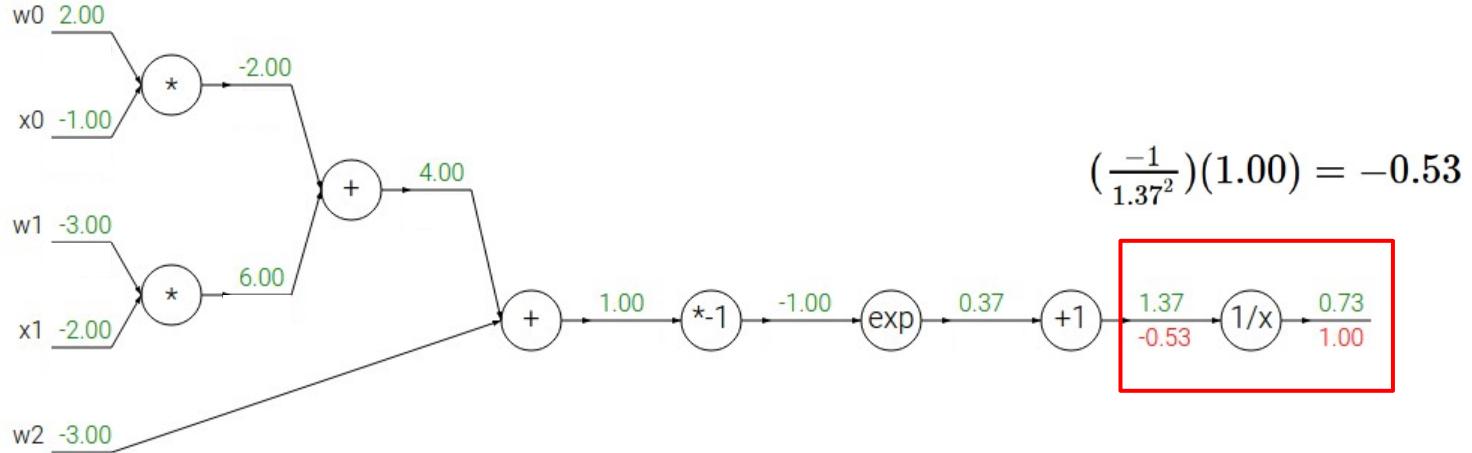
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

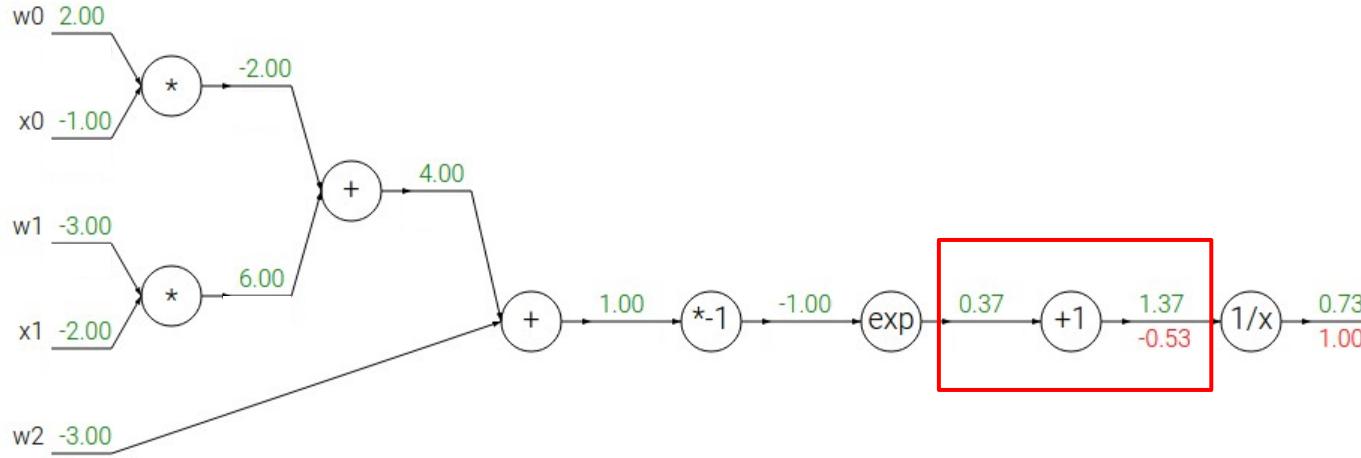
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

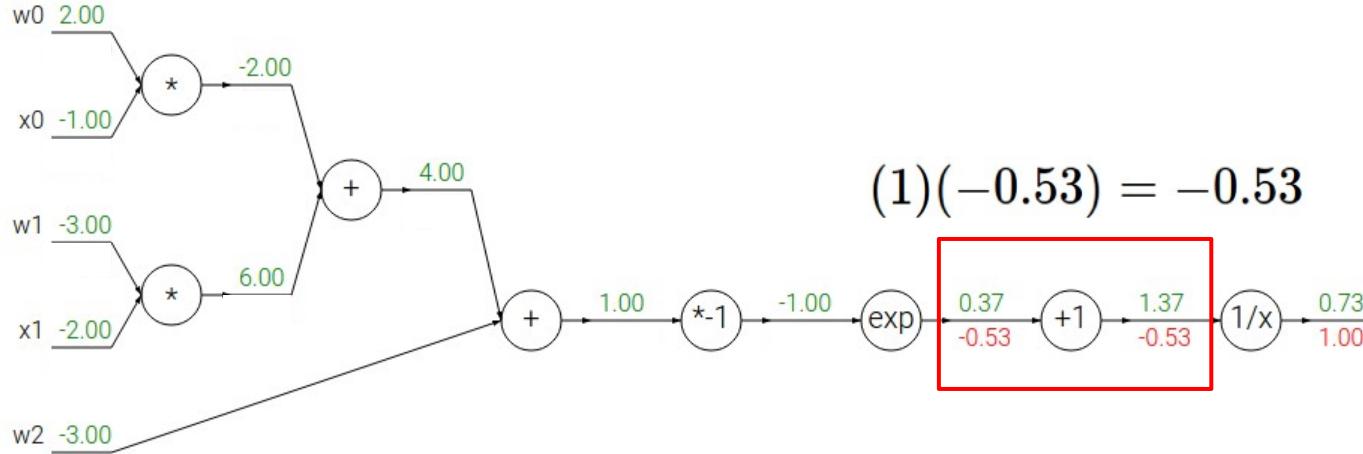
$$f_c(x) = c + x$$

→

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

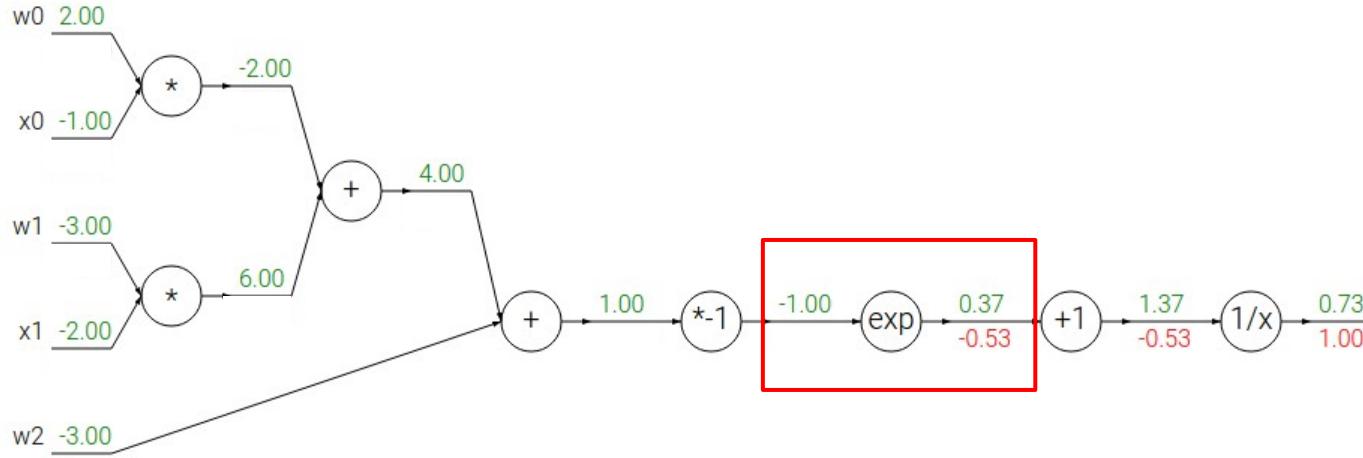
$$\frac{df}{dx} = -1/x^2$$

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

$\rightarrow$

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

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



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

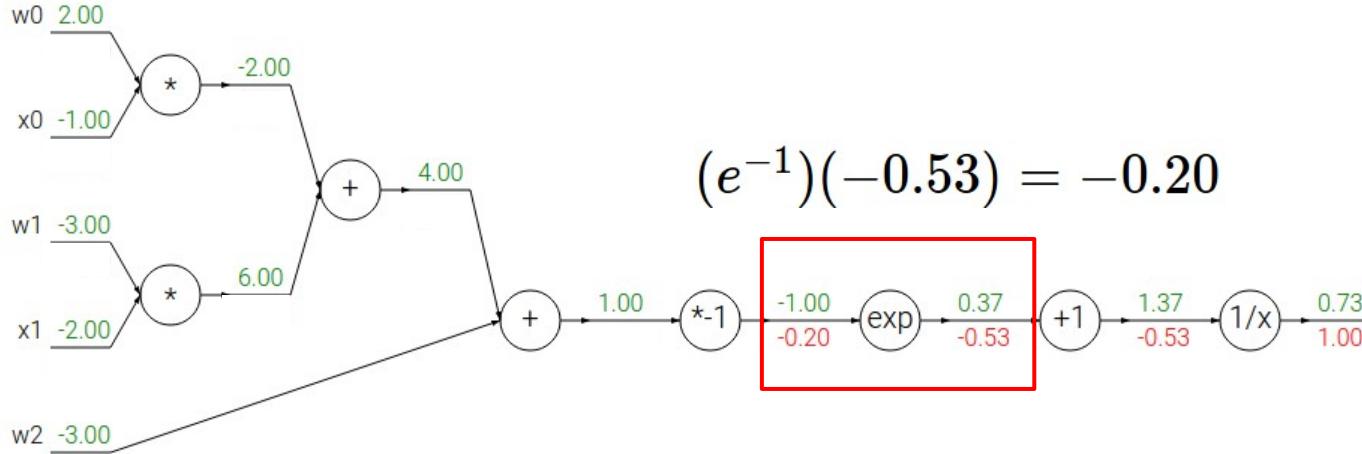
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

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

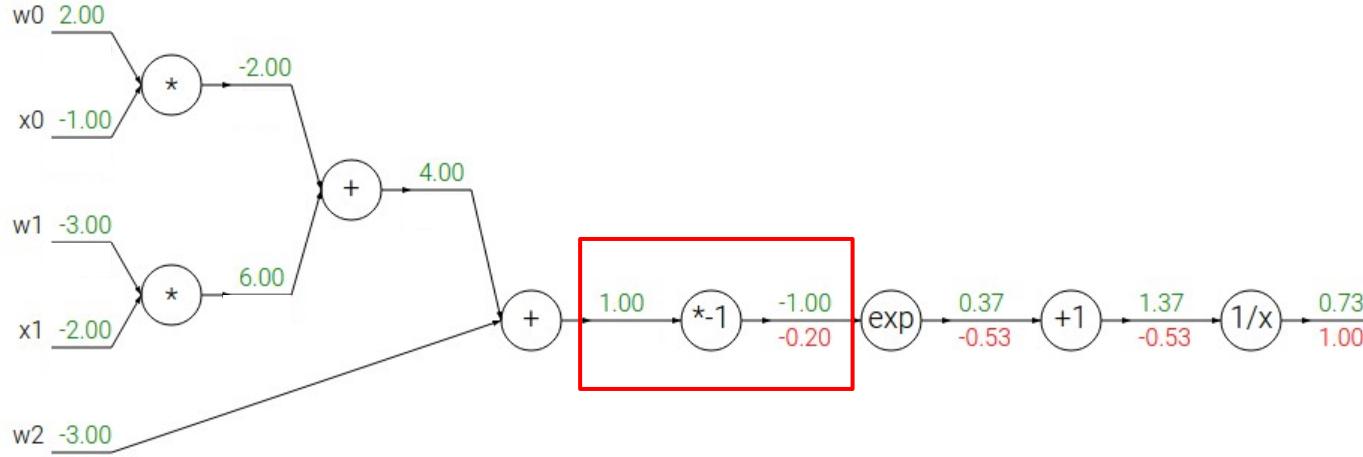
$\rightarrow$

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

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

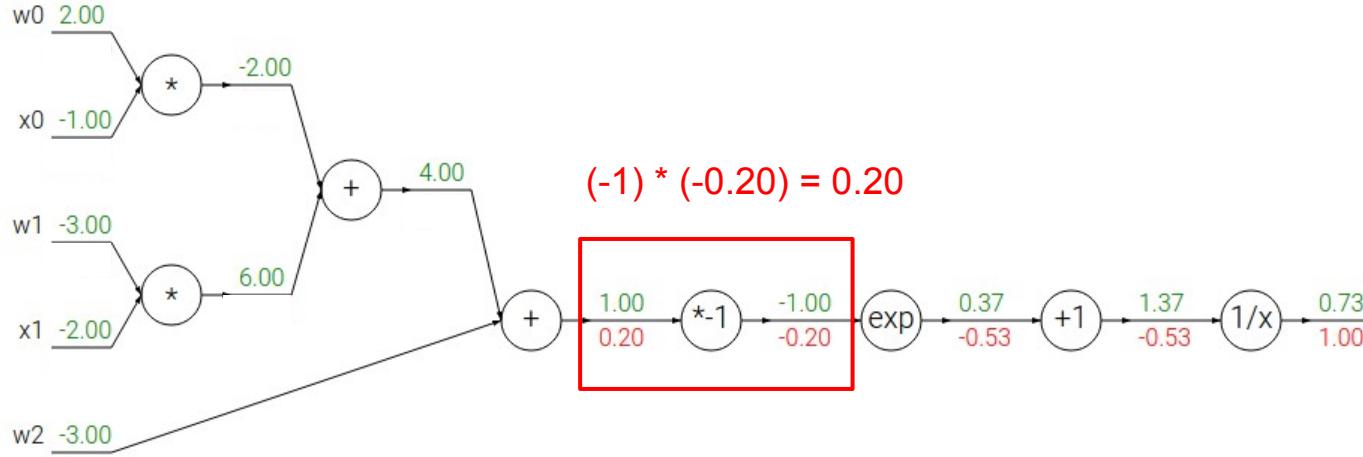
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

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

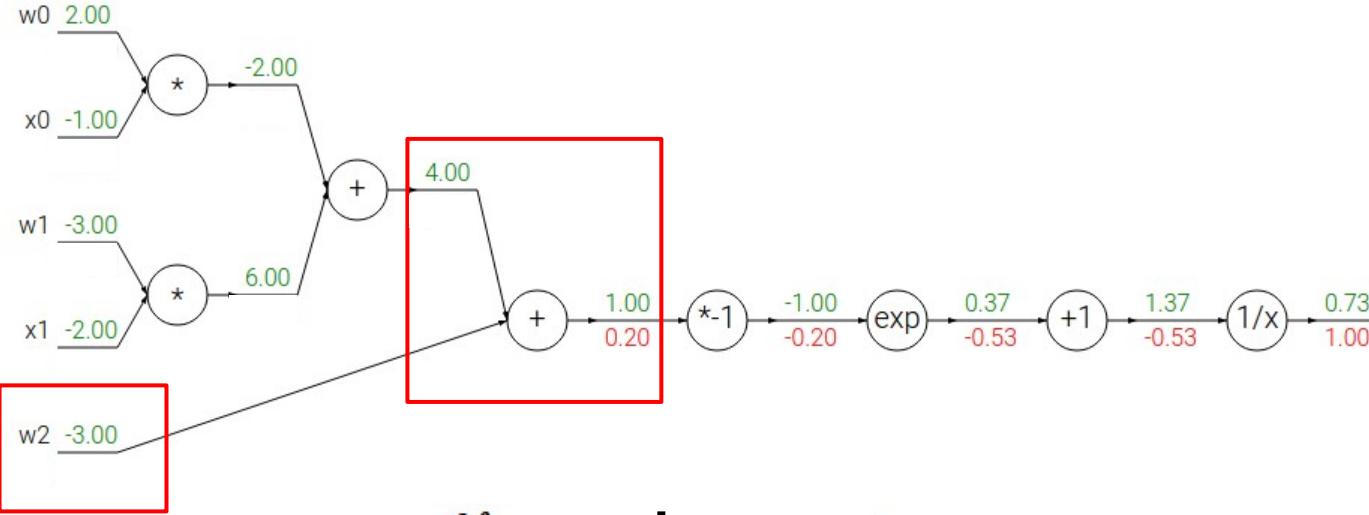
$\rightarrow$

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

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

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

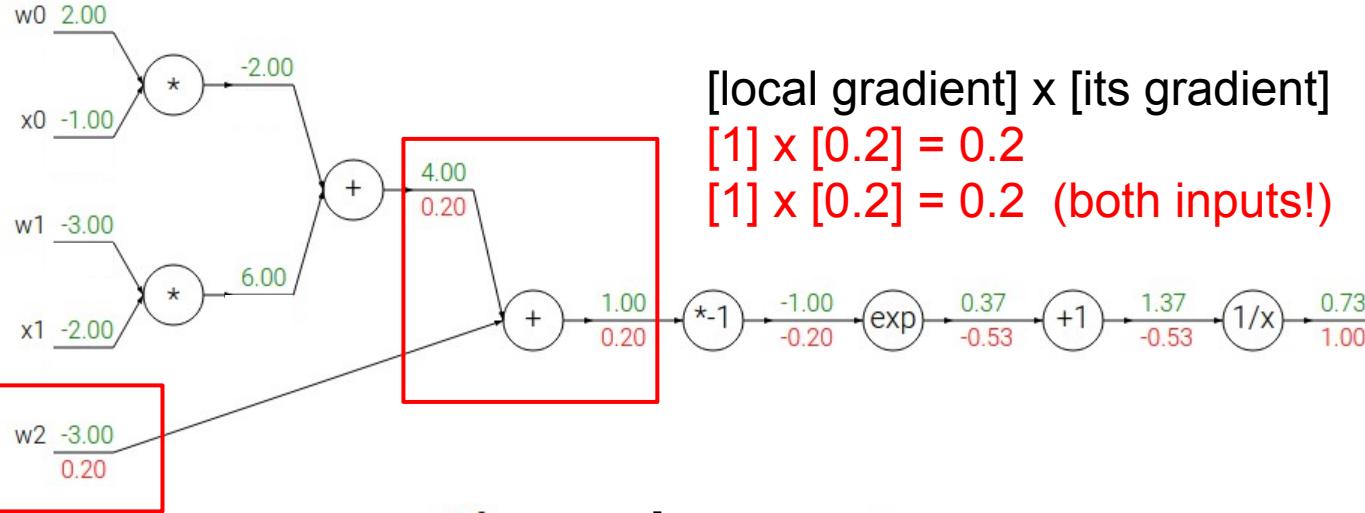
→

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

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

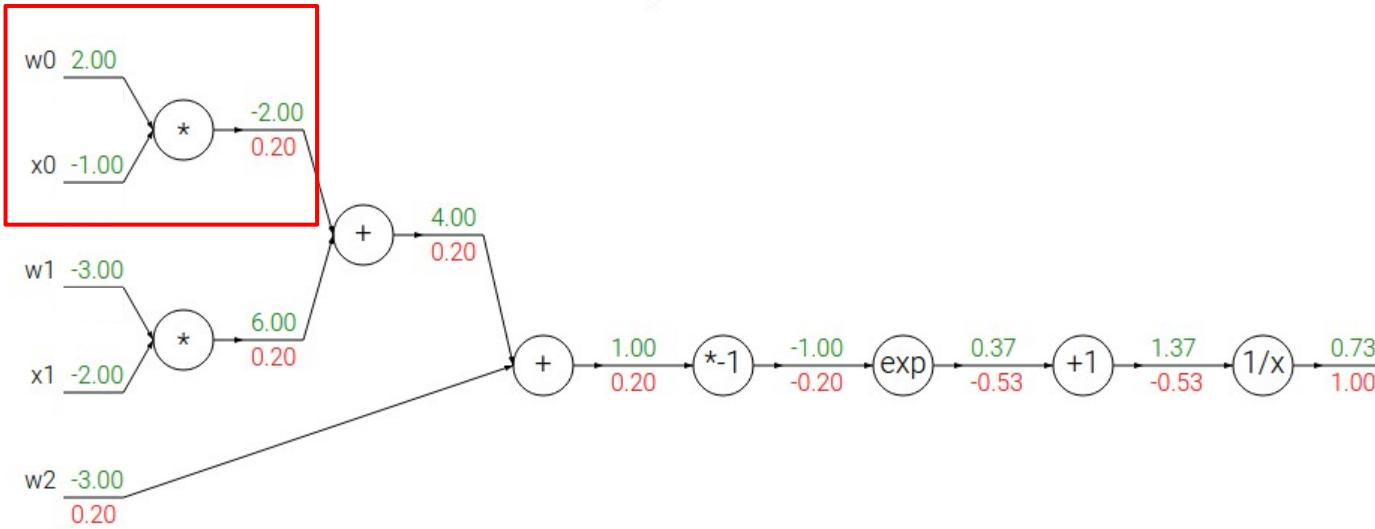
$$f_c(x) = c + x$$

→

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

Another example:

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



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

$\rightarrow$

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

$$f_a(x) = ax$$

$\rightarrow$

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

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

$\rightarrow$

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

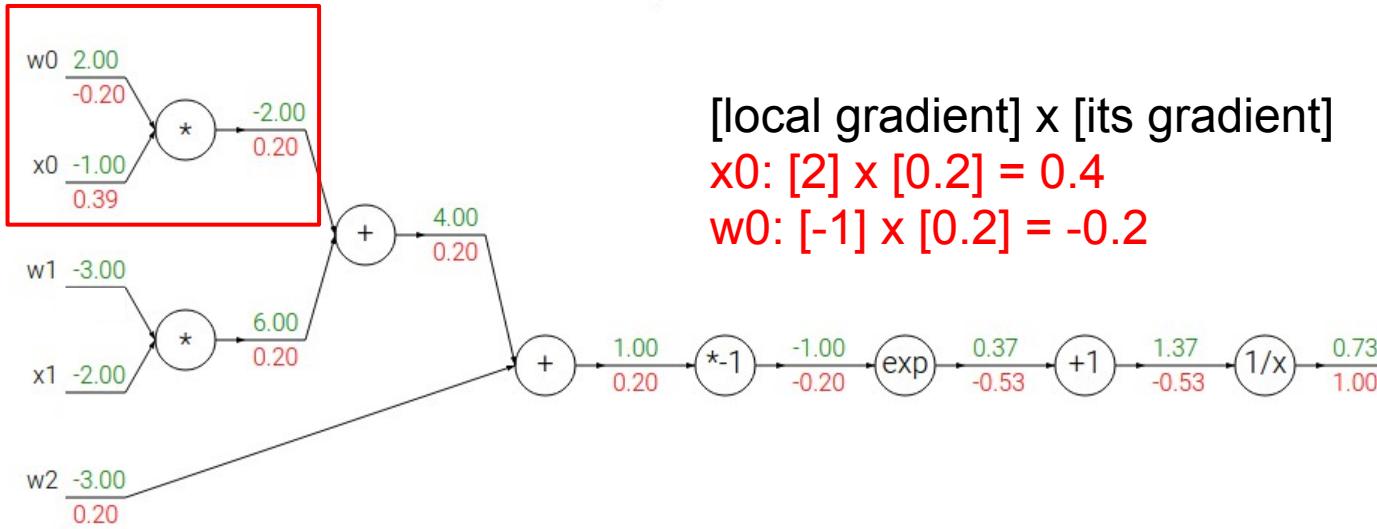
$$f_c(x) = c + x$$

$\rightarrow$

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

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

→

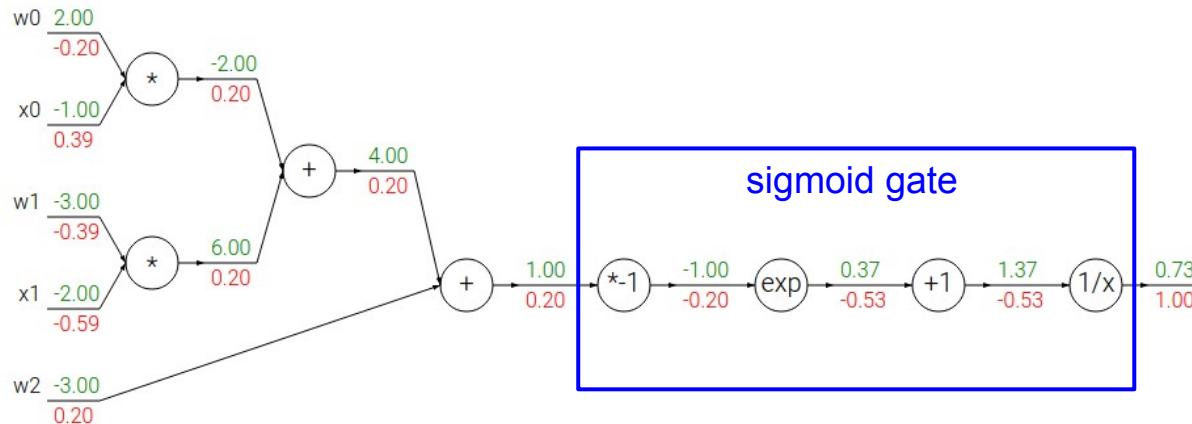
$$\frac{df}{dx} = 1$$

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

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

sigmoid function

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

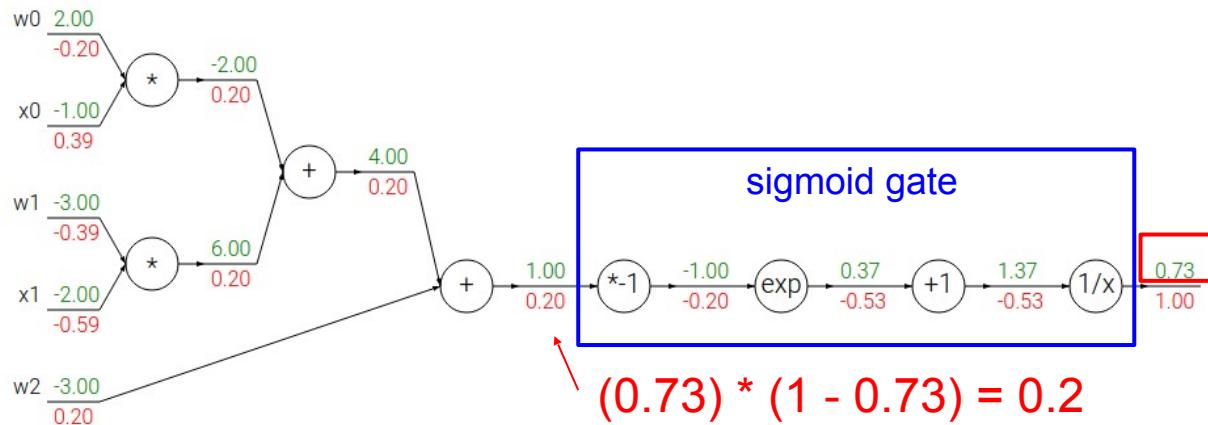


$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

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

sigmoid function

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



```

w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the sigmoid gradient derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the circuit

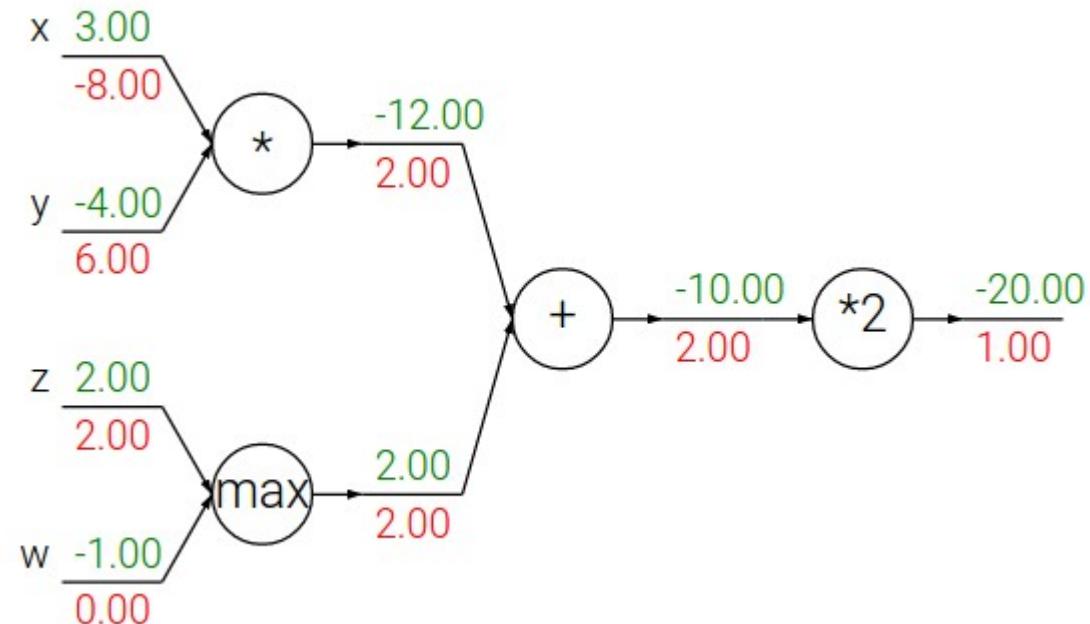
```

# Patterns in backward flow

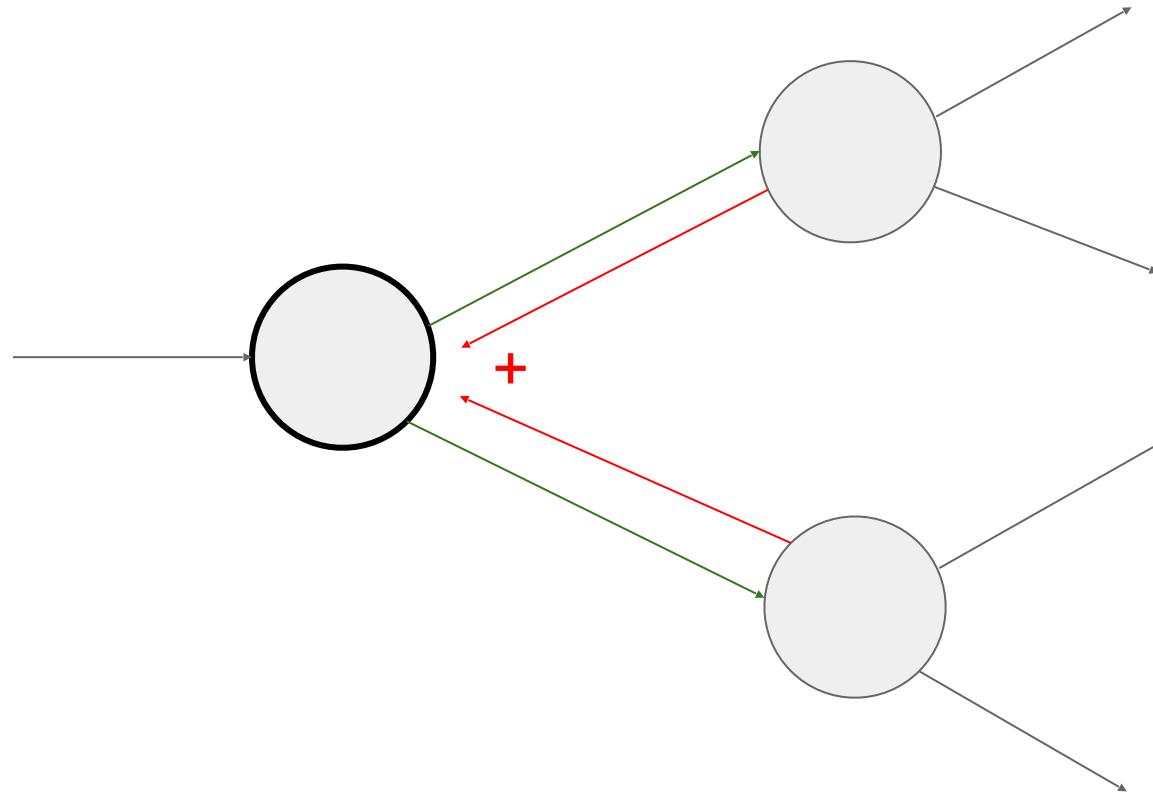
**add** gate: gradient distributor

**max** gate: gradient router

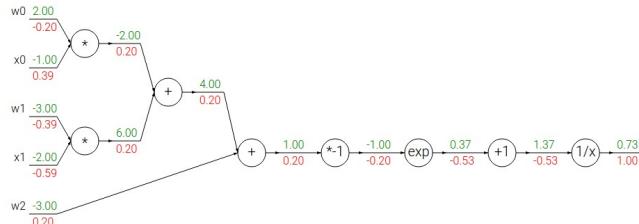
**mul** gate: gradient... “switcher”?



# Gradients add at branches



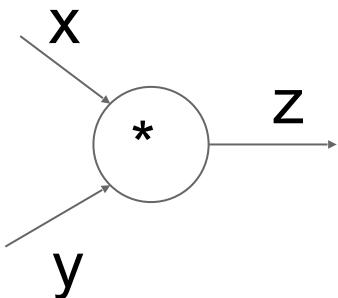
# Implementation: forward/backward API



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
```

# Implementation: forward/backward API



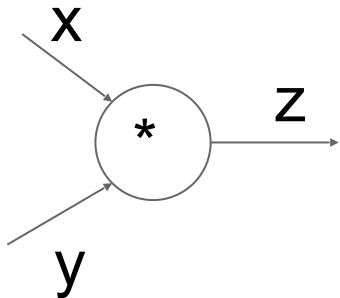
( $x, y, z$  are scalars)

```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        return z  
    def backward(dz):  
        # dx = ... #todo  
        # dy = ... #todo  
        return [dx, dy]
```

$$\frac{\partial L}{\partial x}$$

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

# Implementation: forward/backward API

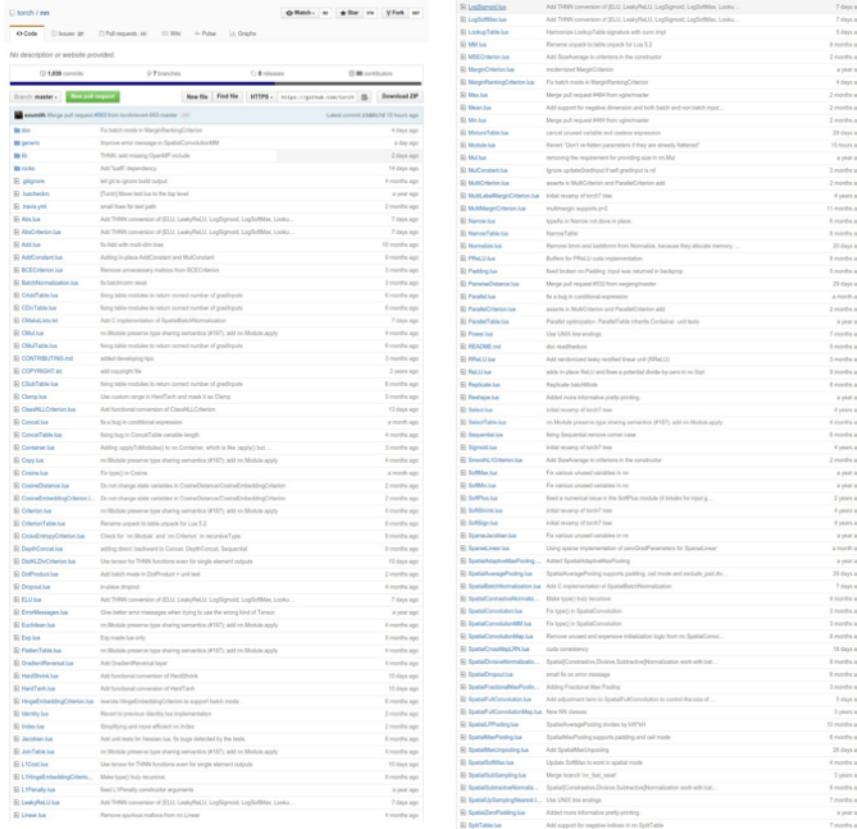


```
class MultiplyGate(object):  
    def forward(x,y):  
        z = x*y  
        self.x = x # must keep these around!  
        self.y = y  
        return z  
    def backward(dz):  
        dx = self.y * dz # [dz/dx * dL/dz]  
        dy = self.x * dz # [dz/dy * dL/dz]  
        return [dx, dy]
```

( $x, y, z$  are scalars)



# Example: Torch Layers



Subhransu Maji, Chuang Gan and TAs  
Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

Lecture 5 - 54

Sept. 19, 2023

# Example: Torch Layers

LogFileTable	Add THINNEN connector of (ELU, LeafReLU, LogSigmoid, LogSoftmax, LogSoftplus, LogSoftmaxTable)	7 days ago
LogSoftplusTable	Hanninen LogSoftplus function with const input	7 days ago
MRI	Remove unprintable characters for Lut 0.2	1 day ago
MSECriterion	Add SoftmaxCriterion to the constructor	8 months ago
MultiCriterion	Remove unused code	2 months ago
MultiLogSoftplusCriterion	Fix batch mode in MultiLogSoftplusCriterion	2 months ago
Max	Merge out printInfo from optimizer	2 months ago
Mean	Add support for negative dimension and both batch and non-batch input.	2 months ago
Min	Merge out printInfo from optimizer	2 months ago
MinValueTable	convert unused variable and useless code	29 days ago
Modulus	Remove 'Does it return parameters? is it already returned?' removing the requirement for printing size as in Mkl	10 hours ago
MultiCriterion	Ignore optimality condition if gradEnrg is n/a	2 months ago
MultiCriterion	use MultiCriterion instead of ParallelCriterion add	2 months ago
MultiLogSoftplusCriterion	Initial warning of initNet	4 years ago
MultiLogSoftplusCriterion	remove unused code	11 months ago
Normalizer	remove unused code in place	4 years ago
Normalizer	None/None	6 months ago
Normalizer	Remove term and condition from Normalizer, because they obscure memory	20 days ago
PrintBuffer	Buffers for PRNG code implementation	8 months ago
Padding	fixed broken Padding, input was returned in backup	5 months ago
ParallelCriterion	Merge out printSize from energemeter	29 days ago
ParallelCriterion	Be a log in conditional expression	1 month ago
ParallelCriterion	assets to MultiCriterion or ParallelCriterion add	2 months ago
ParallelCriterion	Parallel optimization: ParallelUpdate instead of Counter, unit tests	1 year ago
PrintImage	User can run the image	1 year ago
PrintImage	initial warning of initNet	4 years ago
PrintImage	on Module: previous type sharing semantics (#157), add noModule option	4 months ago
Sequential	Using Sequential instead of inner class	6 months ago
SignGrad	Initial warning of initNet	4 years ago
Sigmoid, SigmoidTable	Add SoftmaxCriterion to the constructor	2 months ago
SigmoidTable	Fix for SigmoidTable	1 year ago
Softplus	Add softmax gradient method (softmaxTable)	1 year ago
SoftPlus	add in place PRG and then a product divide by zero in that	8 months ago
SoftplusTable	Replace printTable	8 months ago
SoftplusTable	Add more informative pretty printing	8 months ago
SoftplusTable	initial warning of initNet	4 years ago
SpanningCriterion	On various unused variables in	1 year ago
SpanningCriterion	Using sparse representation of predCandidates for SpanningCriterion	1 month ago
SpanningGraphPrinting	Added SpanningGraphPrinting	29 days ago
SpanningGraphPrinting	SpanningGraphPrinting supports padding, cut mode and exclude, just do the	29 days ago
SpanningGraphImplementation	Add C implementation of SpanningGraphImplementation	6 months ago
SpanningGraphImplementation	Fix for SpanningGraphImplementation	1 year ago
SpanningGraphImplementation	Fix for SpanningGraphImplementation	1 year ago
SpanningGraphImplementation	Fix type in SpanningGraphImplementation	2 months ago
SpanningGraphImplementation	Remove unused and expensive initialization logic from on SpanningGraphImplementation	8 months ago
SpanningGraphImplementation	SpanningGraphImplementation	18 days ago
SpanningGraphImplementation	SpanningGraphImplementation, SubgraphImplementation with bid...	8 months ago
SpanningGraphImplementation	short to an error message	6 months ago
SpanningGraphImplementation	Adding Fractional Max Pooling	3 months ago
SpanningGraphImplementation	Add adjustment term to SpanningGraphImplementation to control the size of...	3 years ago
SpanningGraphImplementation	New NN classes	1 day ago
SpanningGraphImplementation	SpanningGraphImplementation divided by KNN	10 months ago
SpanningGraphImplementation	SpanningGraphImplementation, padding and cut mode	20 days ago
SpanningGraphImplementation	Add SpanningGraphImplementation	20 days ago
SpanningGraphImplementation	Update SpanningGraphImplementation to standard mode	4 months ago
SpanningGraphImplementation	Move branch 'try .. catch ..'	2 years ago
SpanningGraphImplementation	SpanningGraphImplementation, SubgraphImplementation with bid...	8 months ago
SpanningGraphImplementation	Use LINNEN instead	7 months ago
SpanningGraphPrinting	Add more informative pretty printing	1 year ago
SpanTable	Add support for negative indices in a SpanTable	7 months ago



# Example: Torch MulConstant

```
1 local MulConstant, parent = torch.class('nn.MulConstant', 'nn.Module')
2
3 function MulConstant:_init(constant_scalar,ip)
4     parent._init(self)
5     assert(type(constant_scalar) == 'number', 'input is not scalar!')
6     self.constant_scalar = constant_scalar
7
8     -- default for inplace is false
9     self.inplace = ip or false
10    if (ip and type(ip) ~= 'boolean') then
11        error('in-place flag must be boolean')
12    end
13 end
14
15 function MulConstant:updateOutput(input)
16    if self.inplace then
17        input:mul(self.constant_scalar)
18        self.output = input
19    else
20        self.output:resizeAs(input)
21        self.output:copy(input)
22        self.output:mul(self.constant_scalar)
23    end
24    return self.output
25 end
26
27 function MulConstant:updateGradInput(input, gradOutput)
28    if self.gradInput then
29        if self.inplace then
30            gradOutput:mul(self.constant_scalar)
31            self.gradInput = gradOutput
32            -- restore previous input value
33            input:div(self.constant_scalar)
34        else
35            self.gradInput:resizeAs(gradOutput)
36            self.gradInput:copy(gradOutput)
37            self.gradInput:mul(self.constant_scalar)
38        end
39        return self.gradInput
40    end
41 end
```

$$f(X) = aX$$

initialization

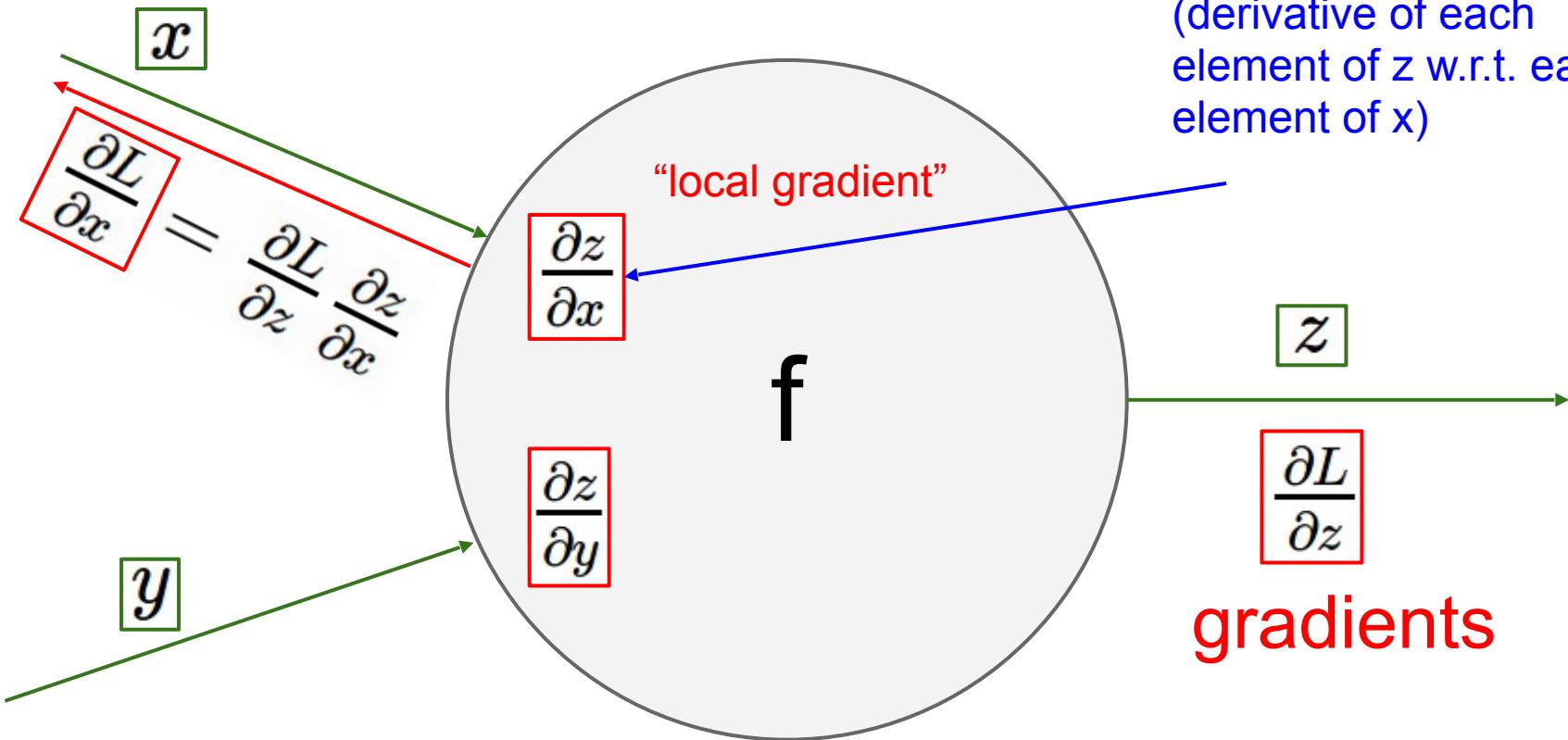
forward()

backward()

# Gradients for vectorized code

( $x, y, z$  are now vectors)

This is now the **Jacobian matrix**  
(derivative of each element of  $z$  w.r.t. each element of  $x$ )



[\[slides\]](#)

[\[backprop notes\]](#)

[\[Efficient BackProp\] \(optional\)](#)

related: [\[1\]](#), [\[2\]](#), [\[3\]](#) (optional)

[\[slides\]](#)

→ handout 1: [Vector, Matrix, and Tensor Derivatives](#)

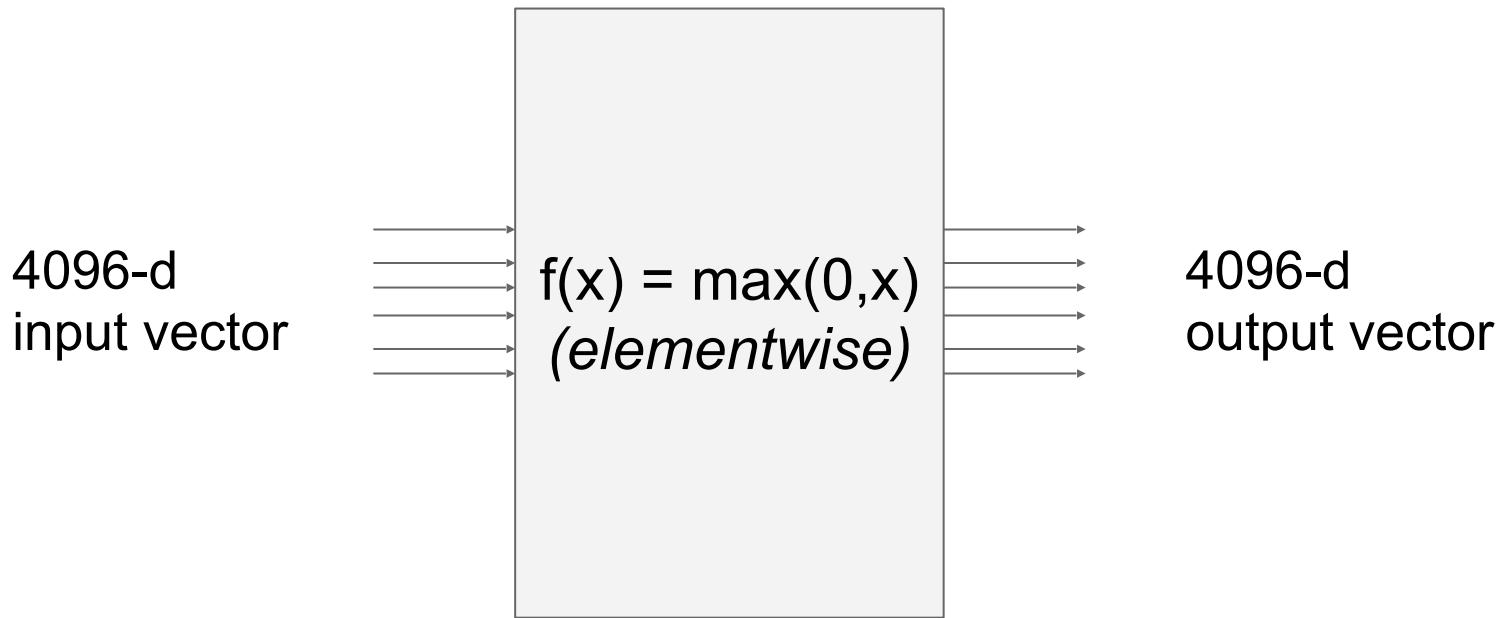
→ handout 2: [Derivatives, Backpropagation, and  
Vectorization](#)

[Deep Learning \[Nature\] \(optional\)](#)

[\[slides\]](#)

tips/tricks: [\[1\]](#), [\[2\]](#) (optional)

# Vectorized operations

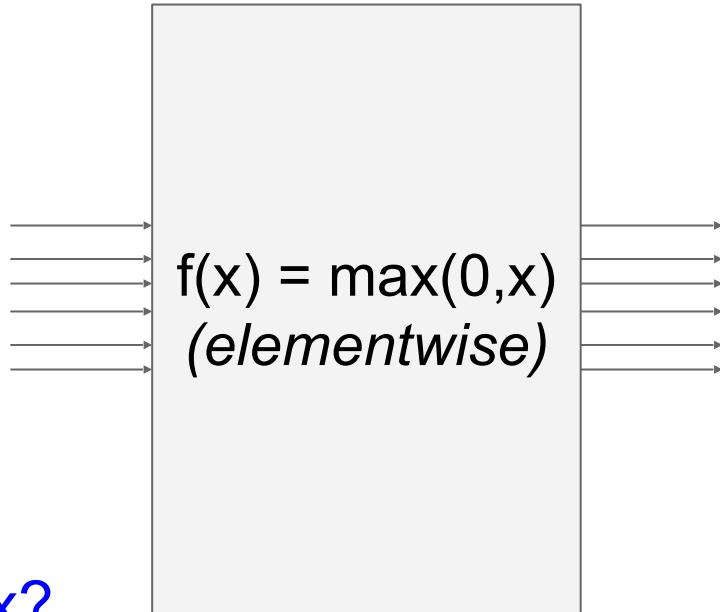


# Vectorized operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix

4096-d  
input vector



4096-d  
output vector

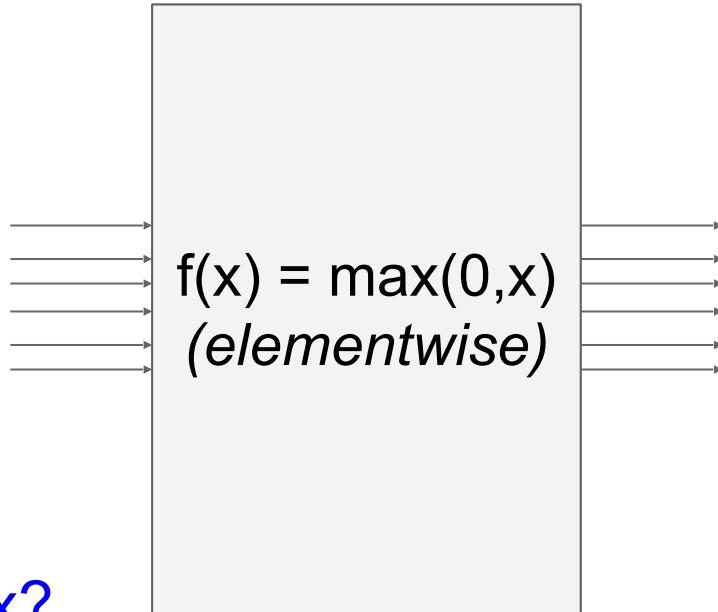
Q: what is the  
size of the  
Jacobian matrix?

# Vectorized operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix

4096-d  
input vector



4096-d  
output vector

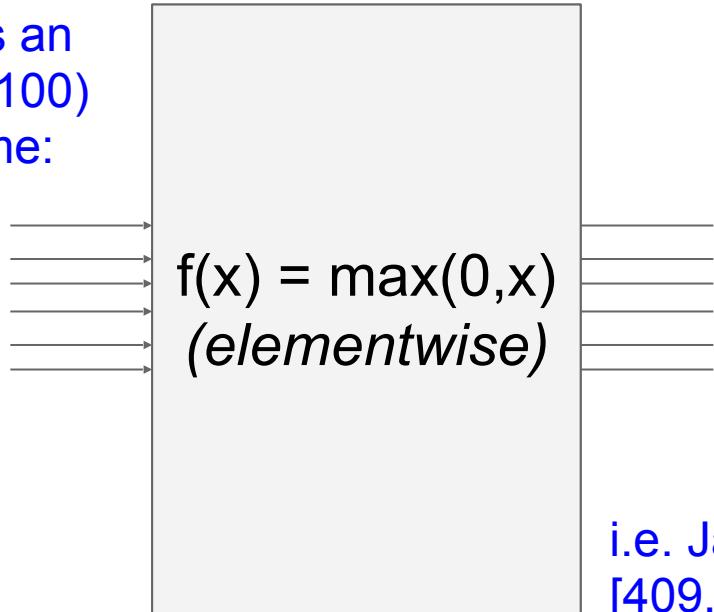
Q: what is the  
size of the  
Jacobian matrix?  
[4096 x 4096!]

Q2: what does it  
look like?

# Vectorized operations

in practice we process an entire minibatch (e.g. 100) of examples at one time:

100 4096-d  
input vectors



100 4096-d  
output vectors

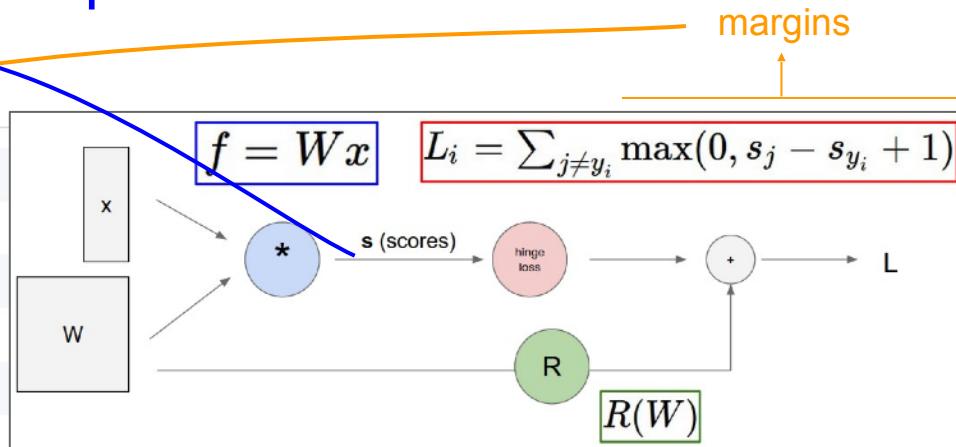
i.e. Jacobian would technically be a [409,600 x 409,600] matrix :\  
\\

# Assignment: Writing SVM/Softmax

## Stage your forward/backward computation!

E.g. for the SVM:

```
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```



# Summary so far

- neural nets will be very large: no hope of writing down gradient formula by hand for all parameters
- **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.