

Lecture 4:

Backpropagation and Neural Networks part 1

Administrative

A1 is due Jan 20 (Wednesday). ~150 hours left

Warning: Jan 18 (Monday) is Holiday (no class/office hours)

Also note:

Lectures are non-exhaustive.

Read course notes for completeness.

I'll hold make up office hours on Wed Jan20, 5pm @ Gates 259

Where we are...

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

scores function

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

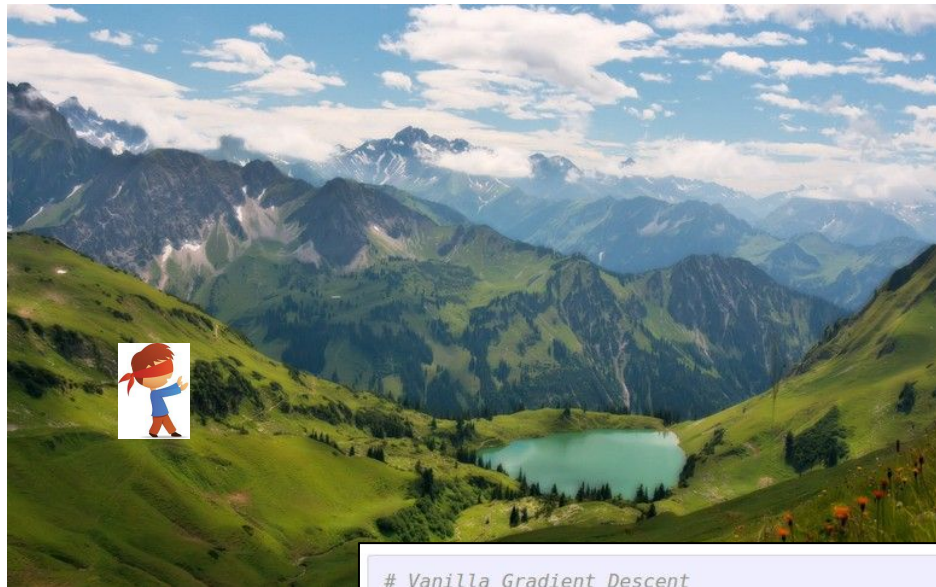
SVM loss

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

data loss + regularization

want $\nabla_W L$

Optimization

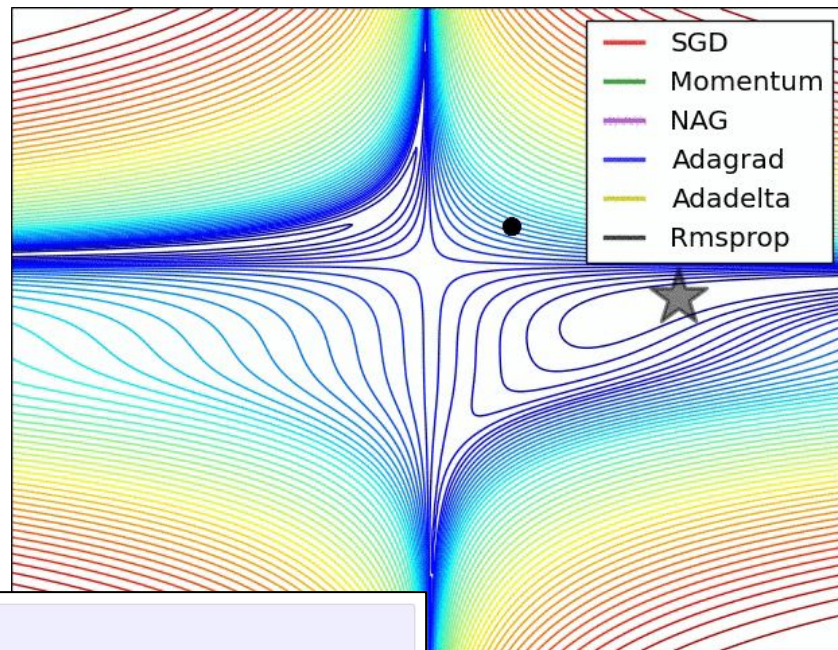


```
# Vanilla Gradient Descent
```

```
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

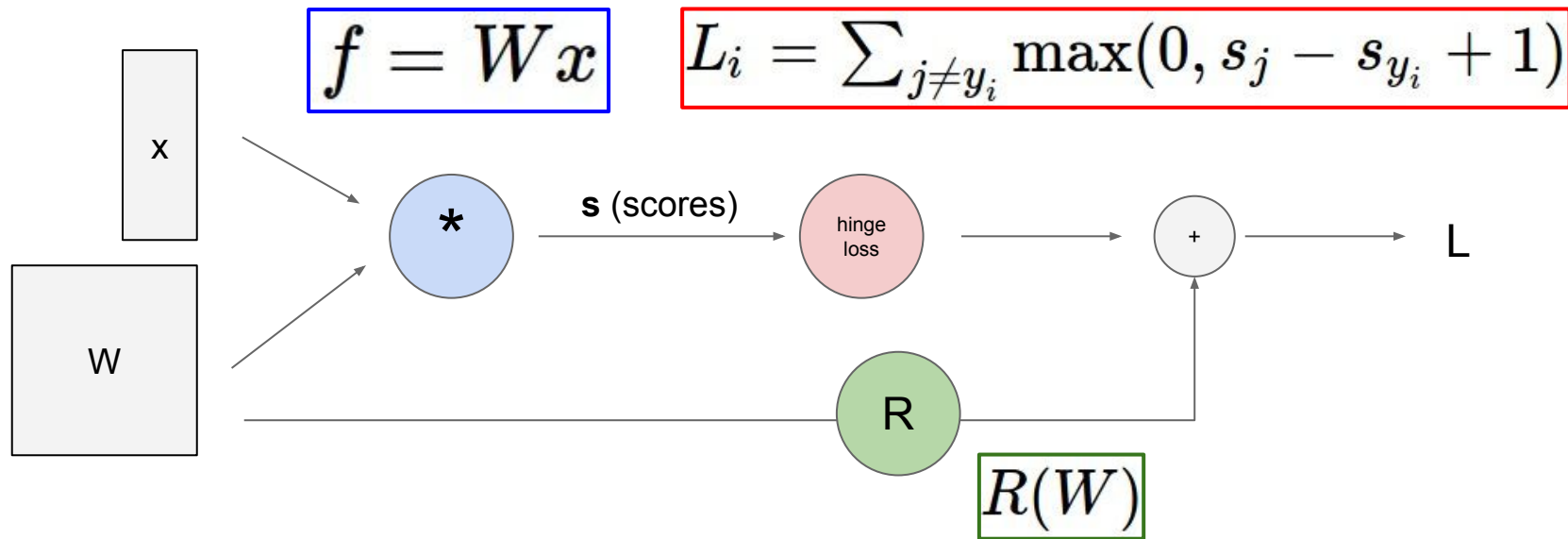
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

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

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

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

Computational Graph

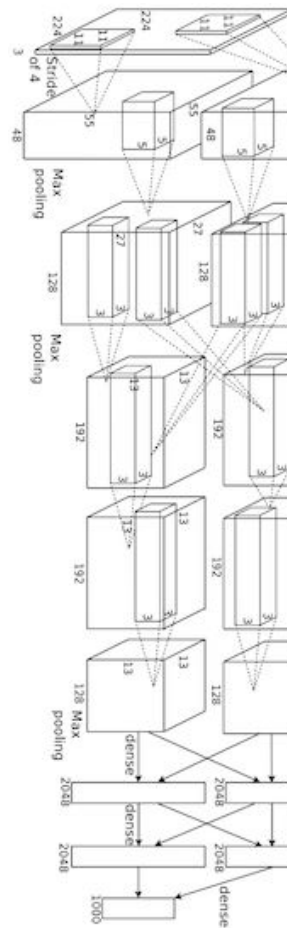


Convolutional Network (AlexNet)

input image

weights

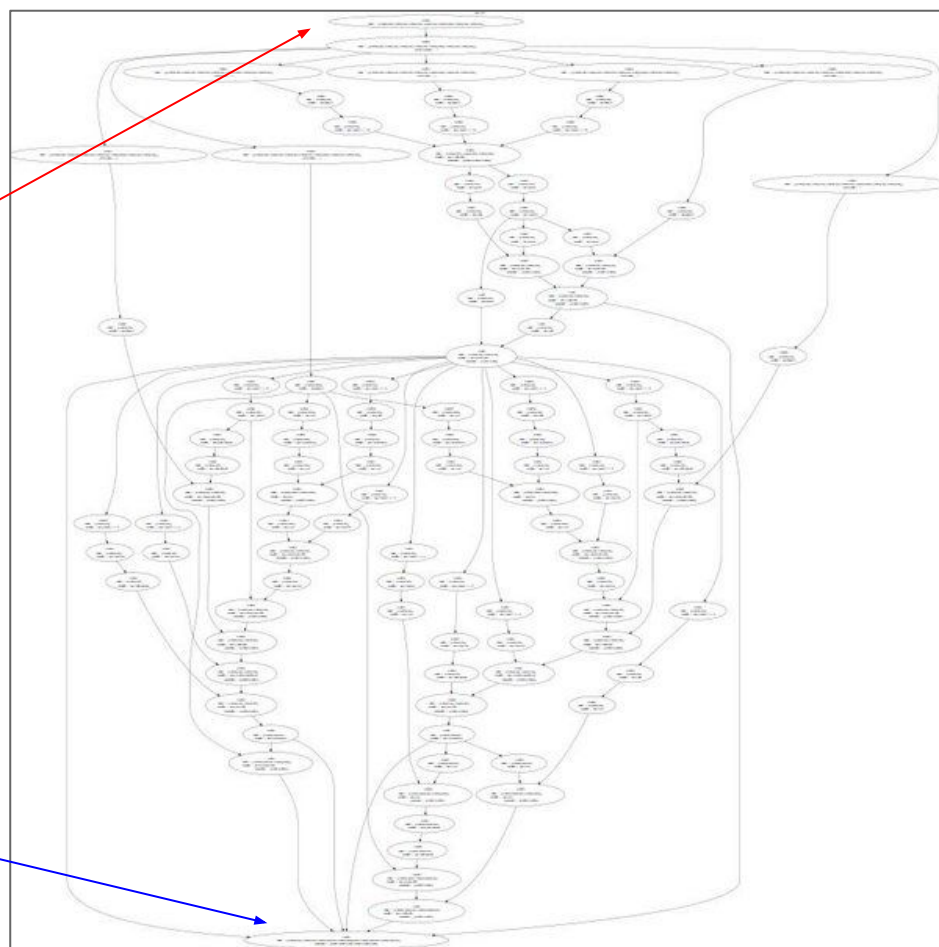
loss



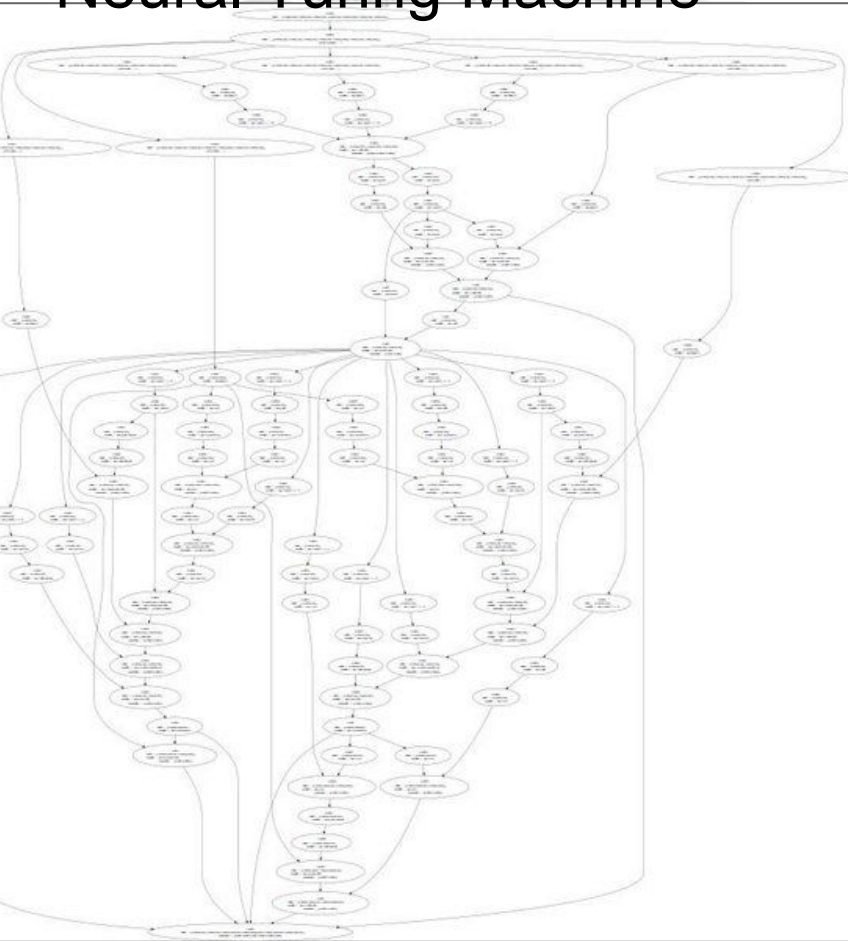
Neural Turing Machine

input tape

loss

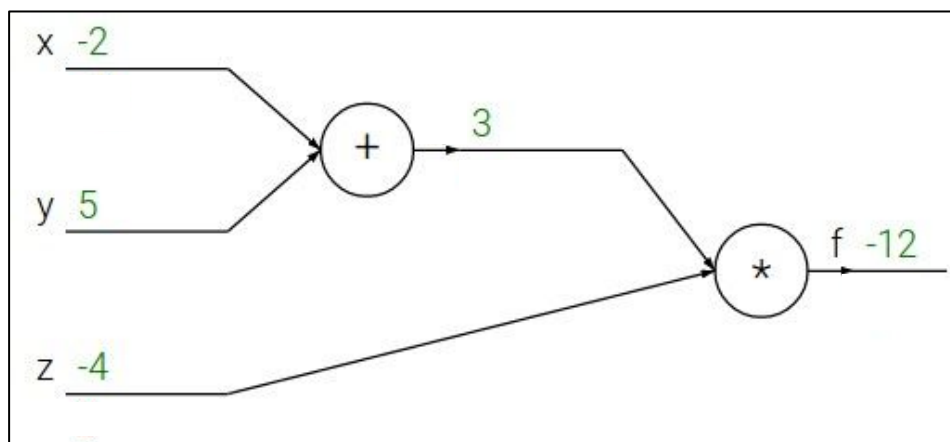


Neural Turing Machine



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

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



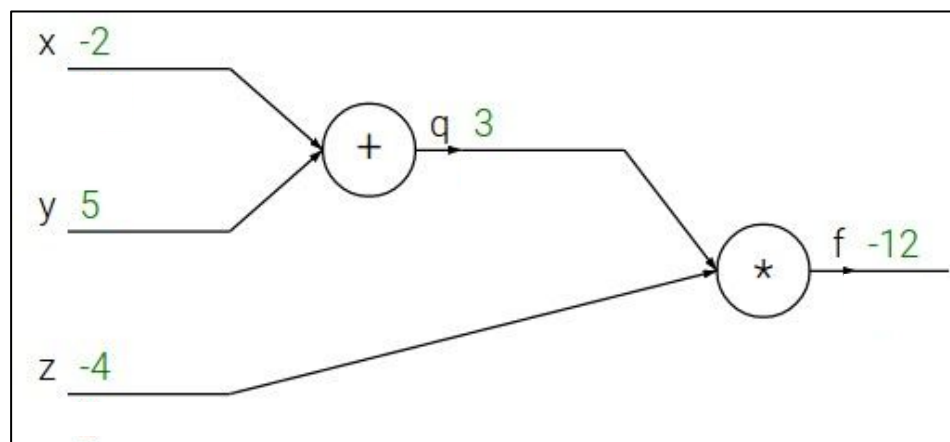
$$f(x, y, z) = (x + y)z$$

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



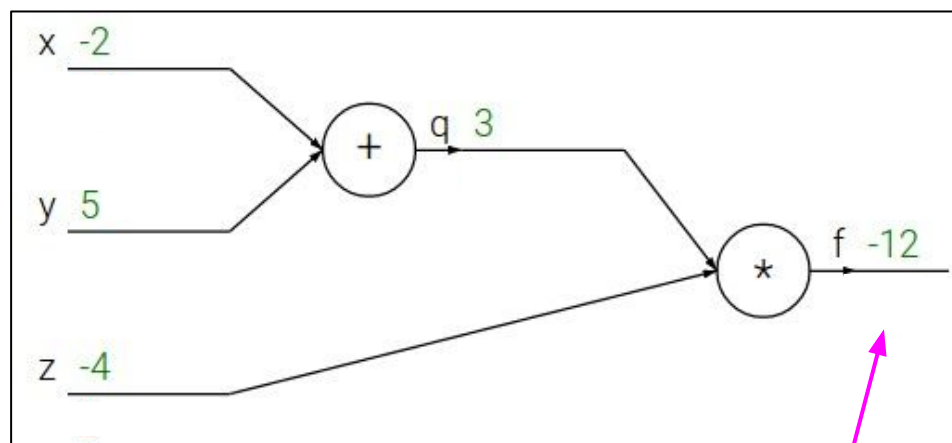
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

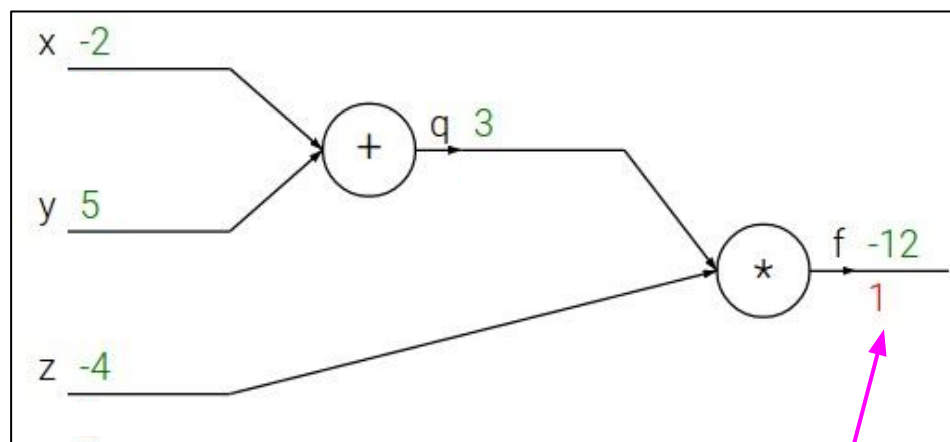
$$f(x, y, z) = (x + y)z$$

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



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

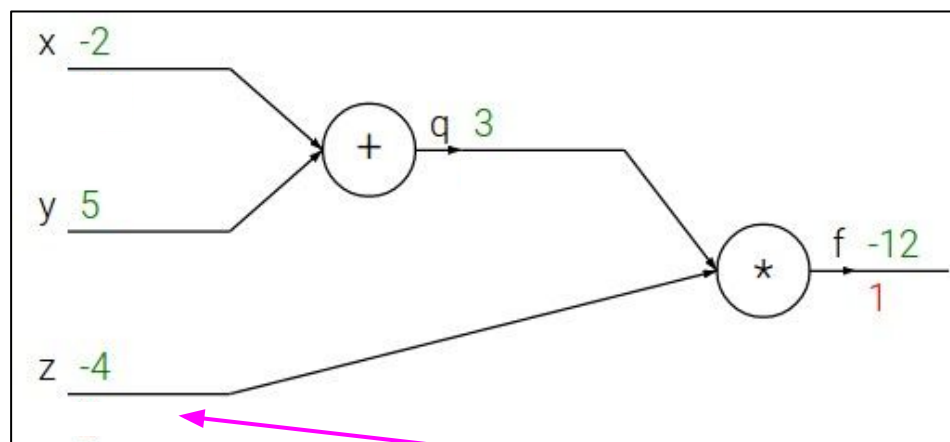
$$f(x, y, z) = (x + y)z$$

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



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

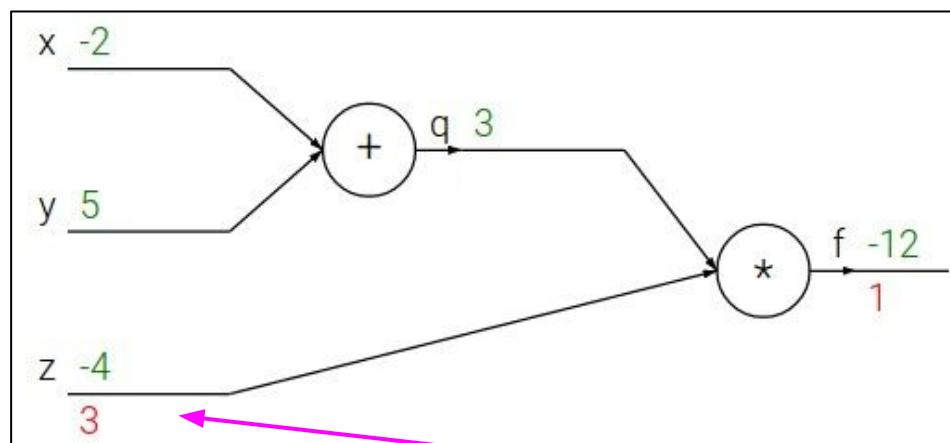
$$f(x, y, z) = (x + y)z$$

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



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

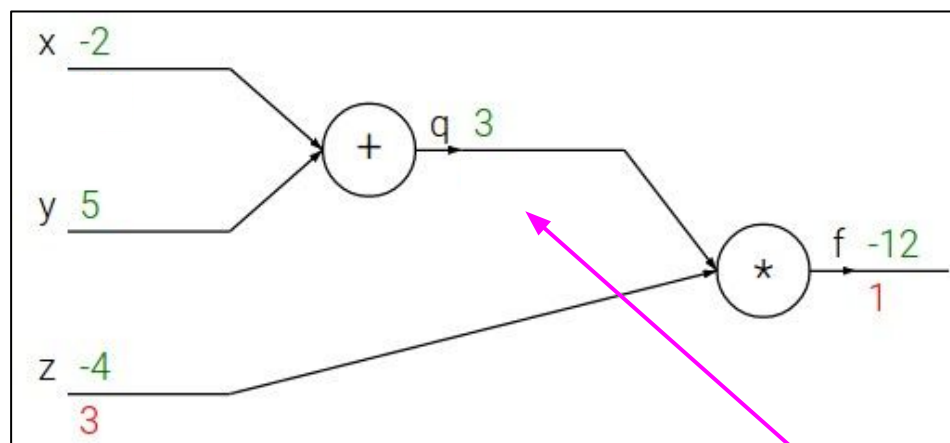
$$f(x, y, z) = (x + y)z$$

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



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

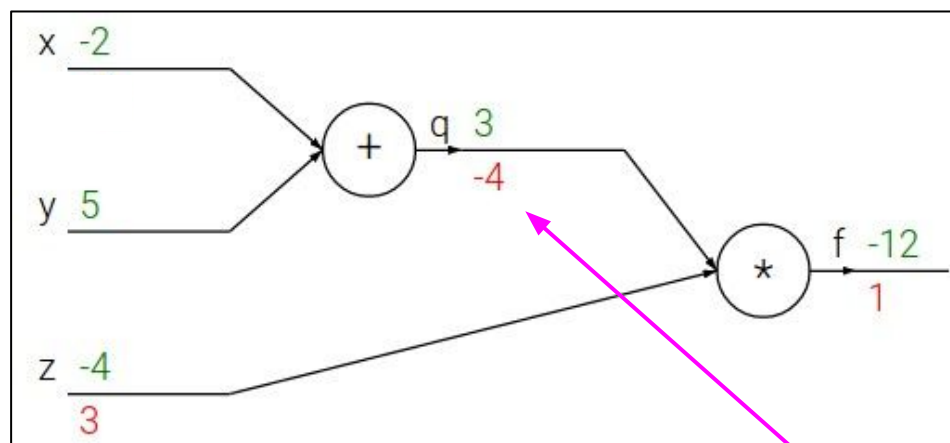
$$f(x, y, z) = (x + y)z$$

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



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

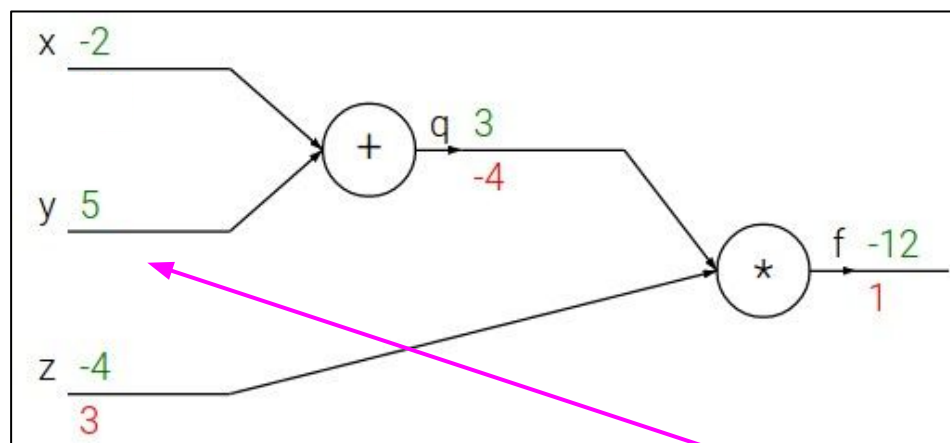
$$f(x, y, z) = (x + y)z$$

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



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

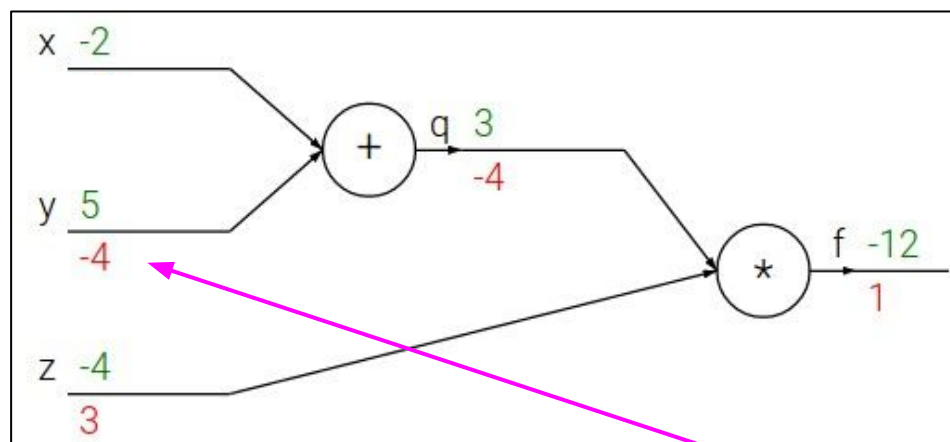
$$f(x, y, z) = (x + y)z$$

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:

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

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

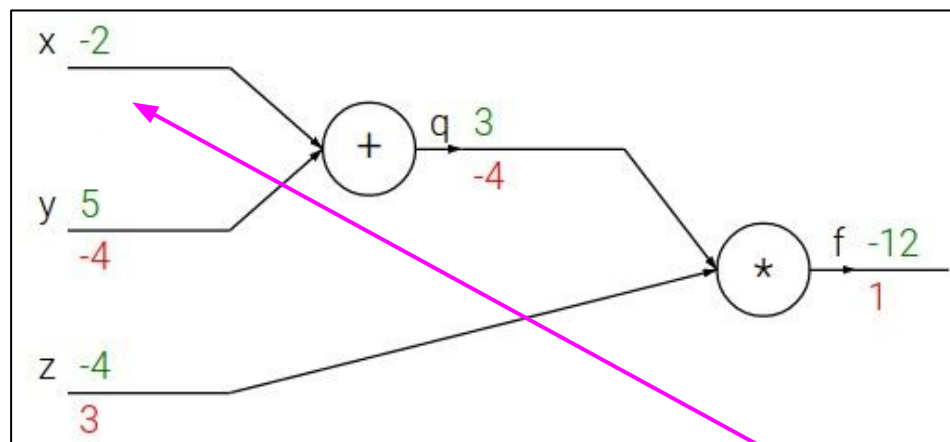
$$f(x, y, z) = (x + y)z$$

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



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

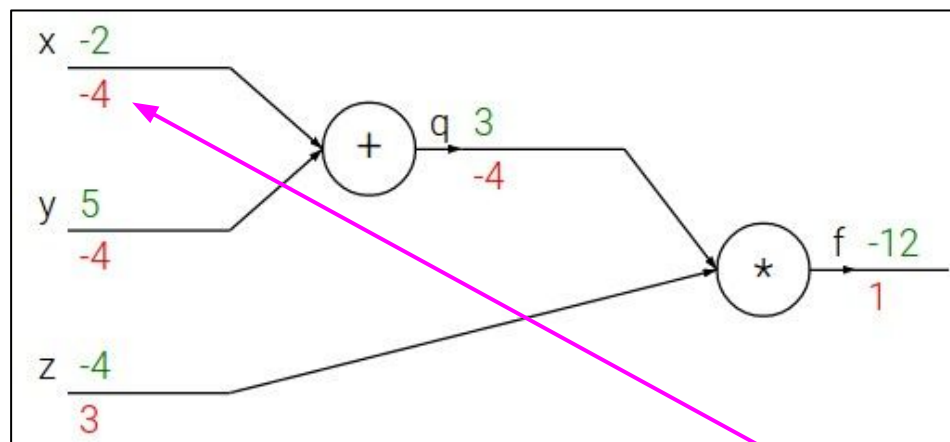
$$f(x, y, z) = (x + y)z$$

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

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

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

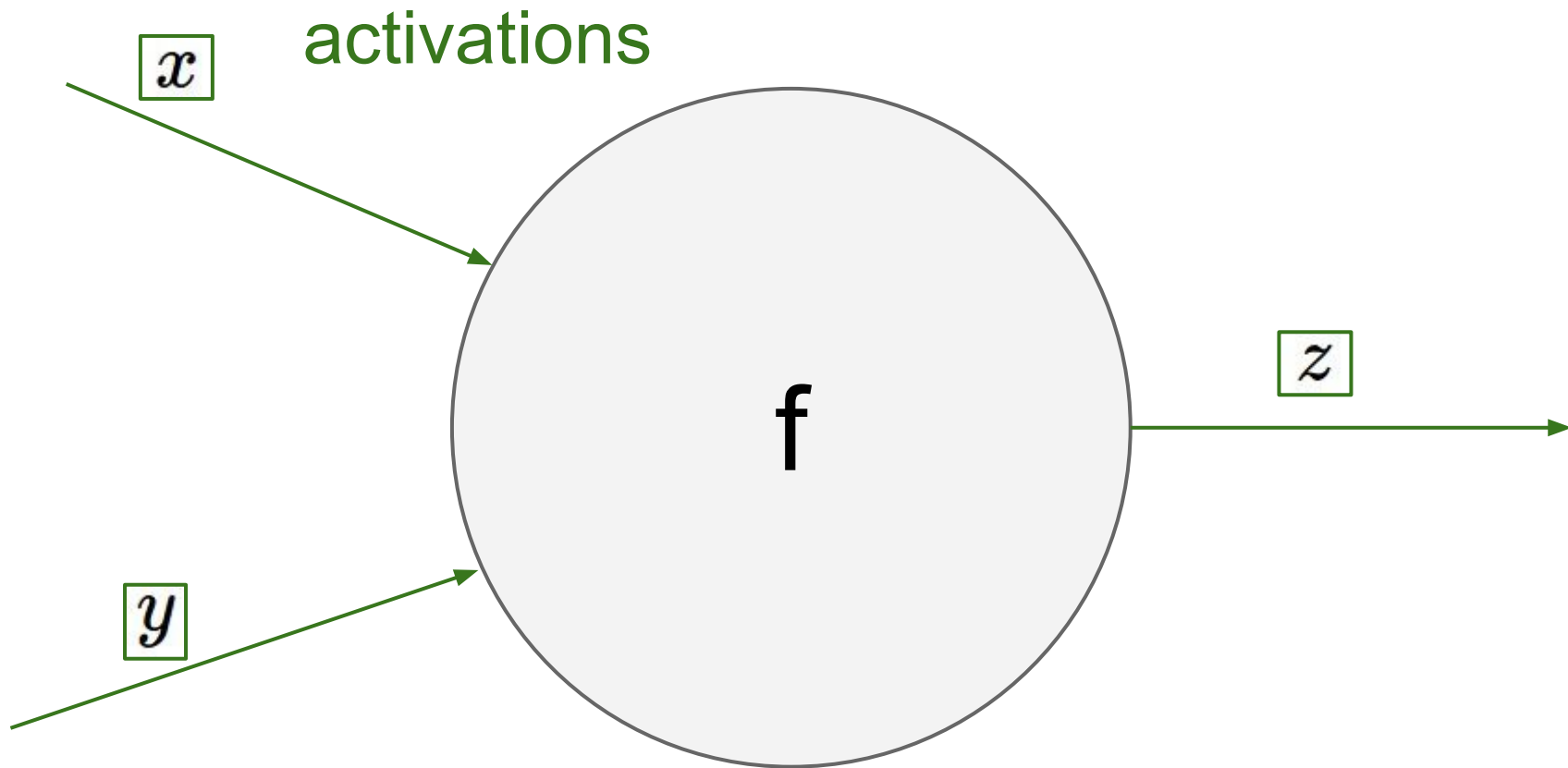
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

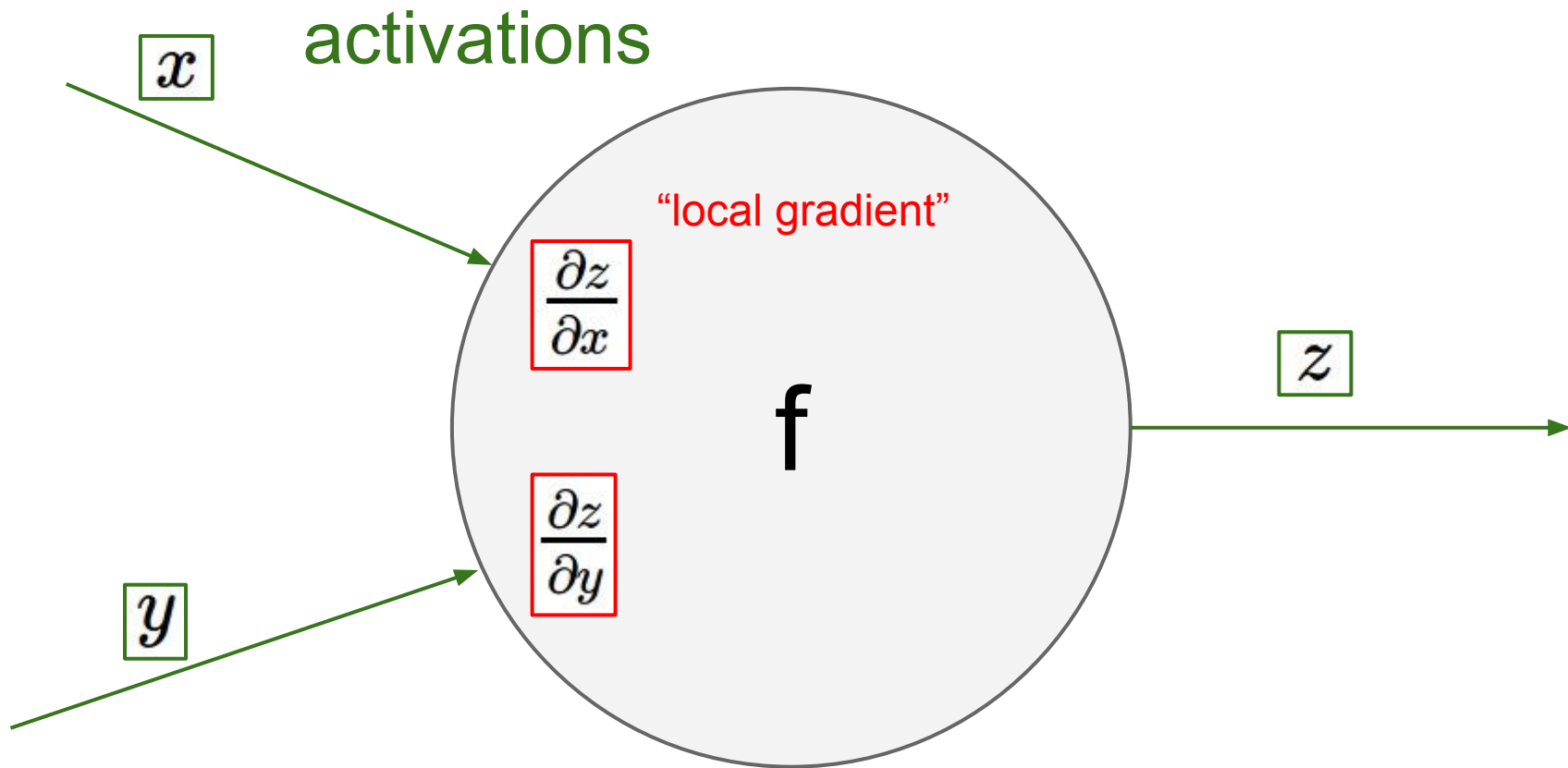


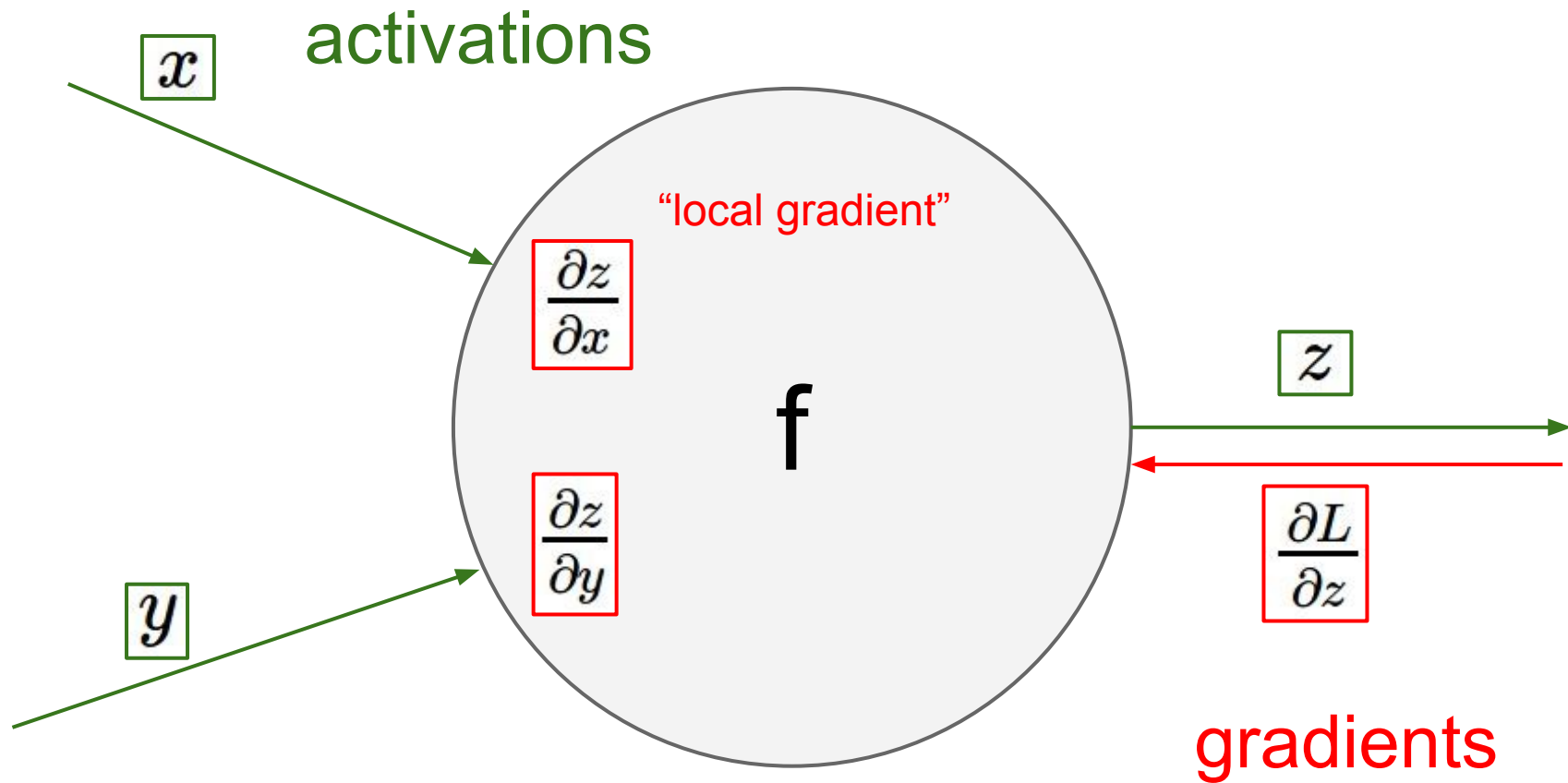
$$\frac{\partial f}{\partial x}$$

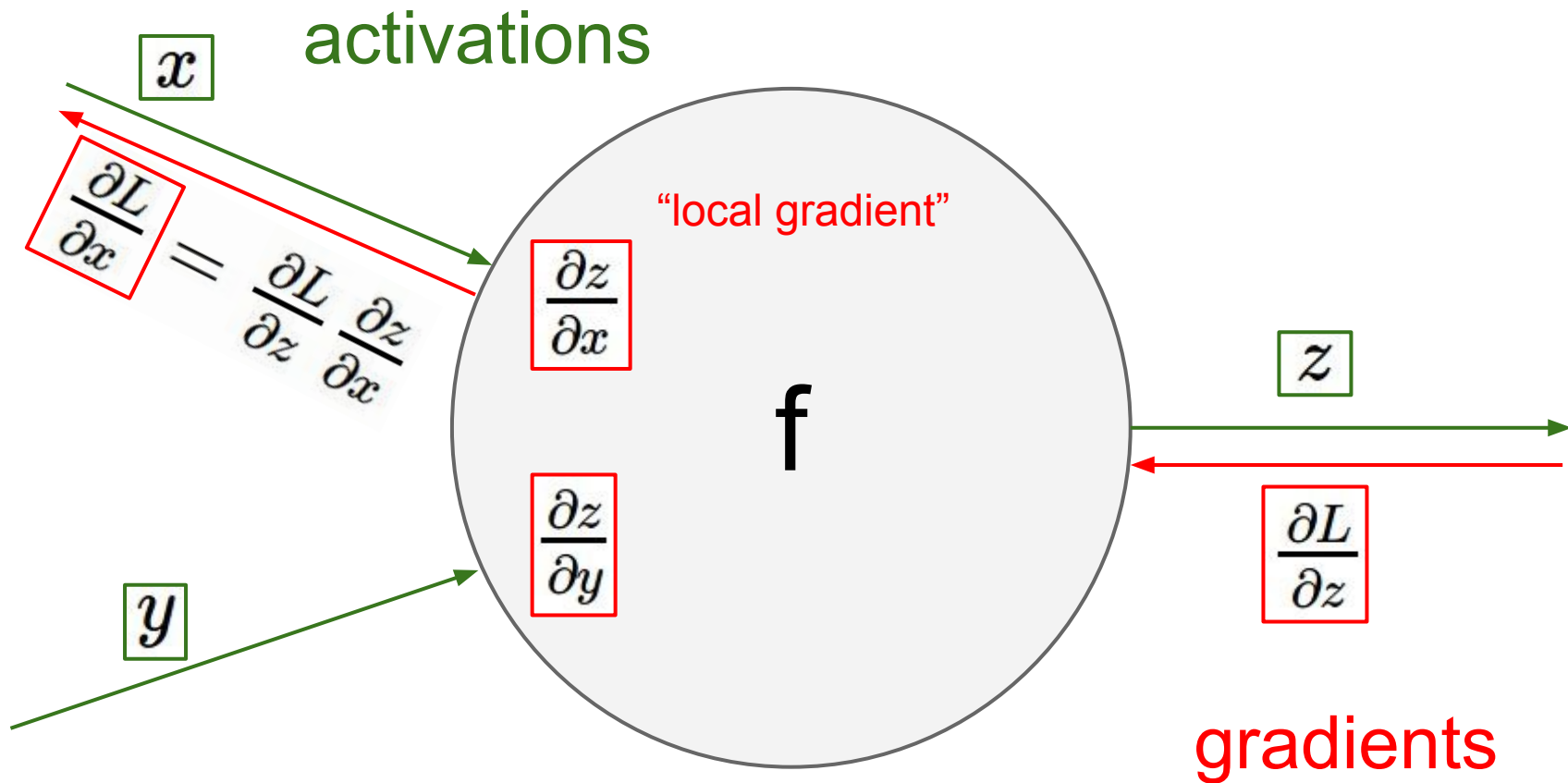
Chain rule:

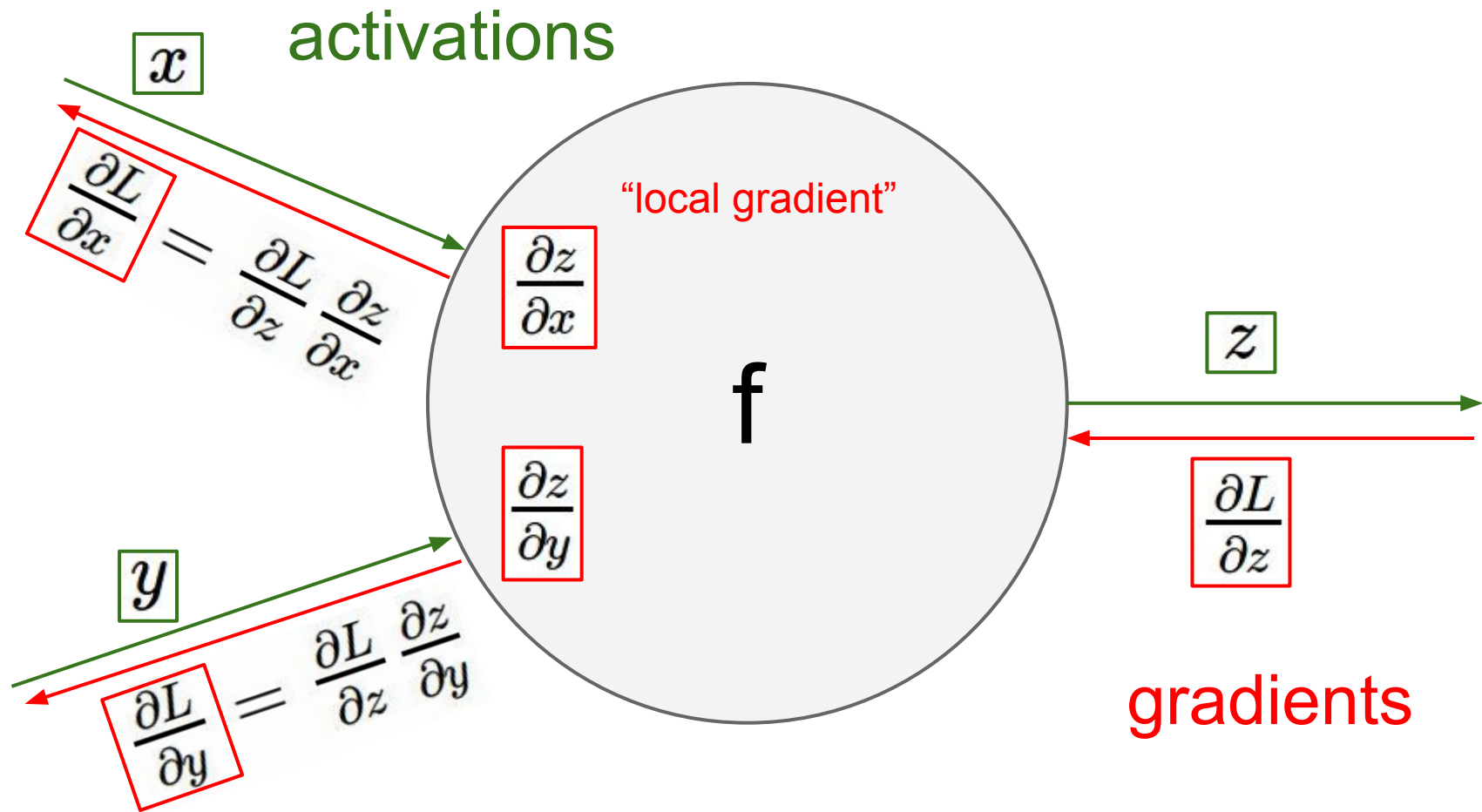
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$











activations

x

分别再返回给x和y

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

“local gradient”

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

$$\frac{\partial z}{\partial y}$$

f

z

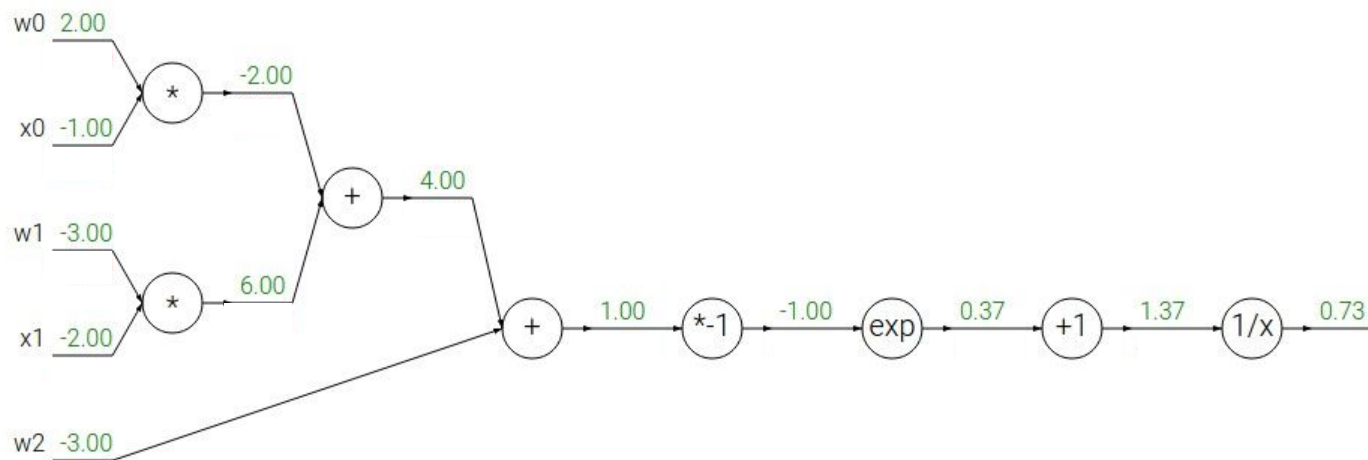
$$\frac{\partial L}{\partial z}$$

从上一个点返回的对这个点的gradient

gradients

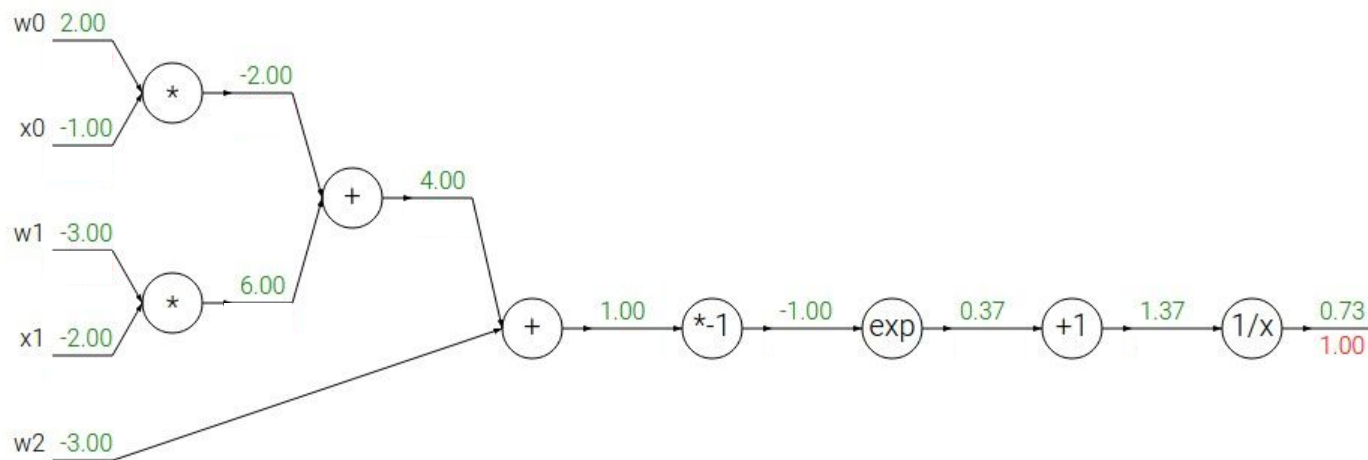
Another example:

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



Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

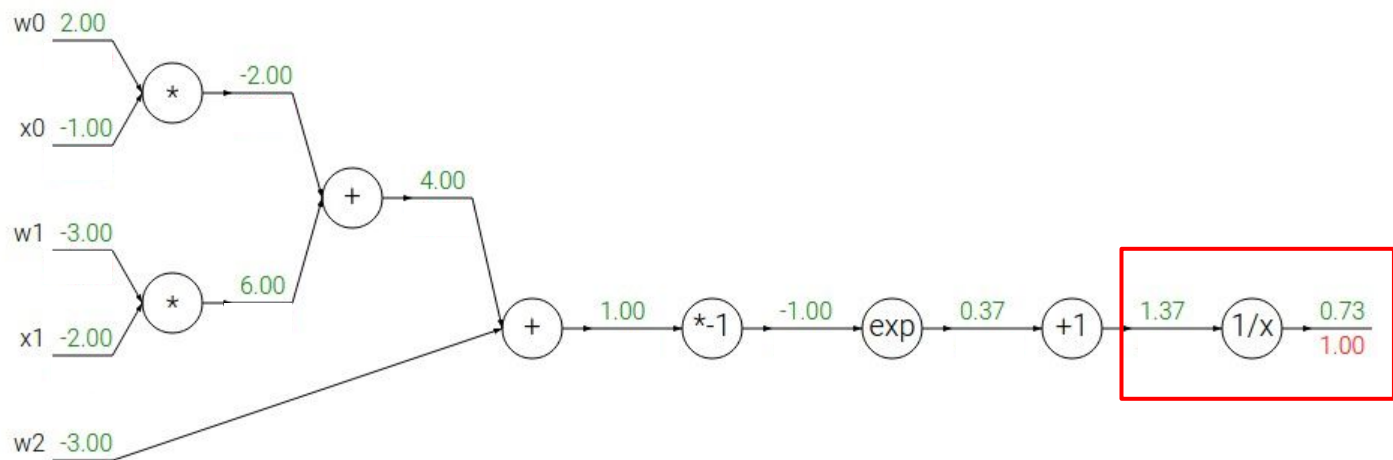
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

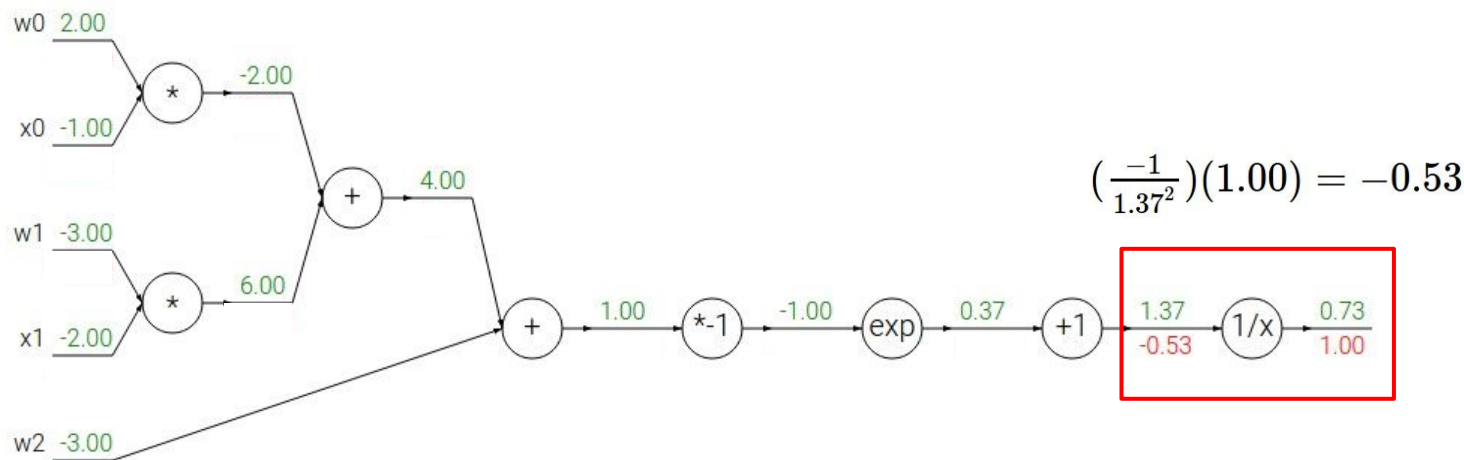
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

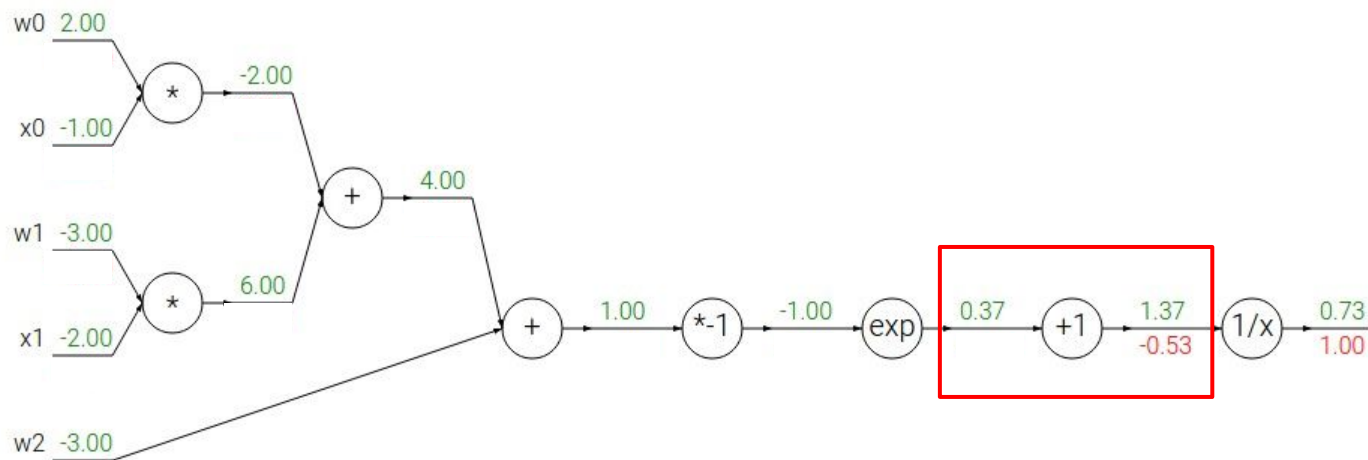
$$f_c(x) = c + x$$

→

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

Another example:

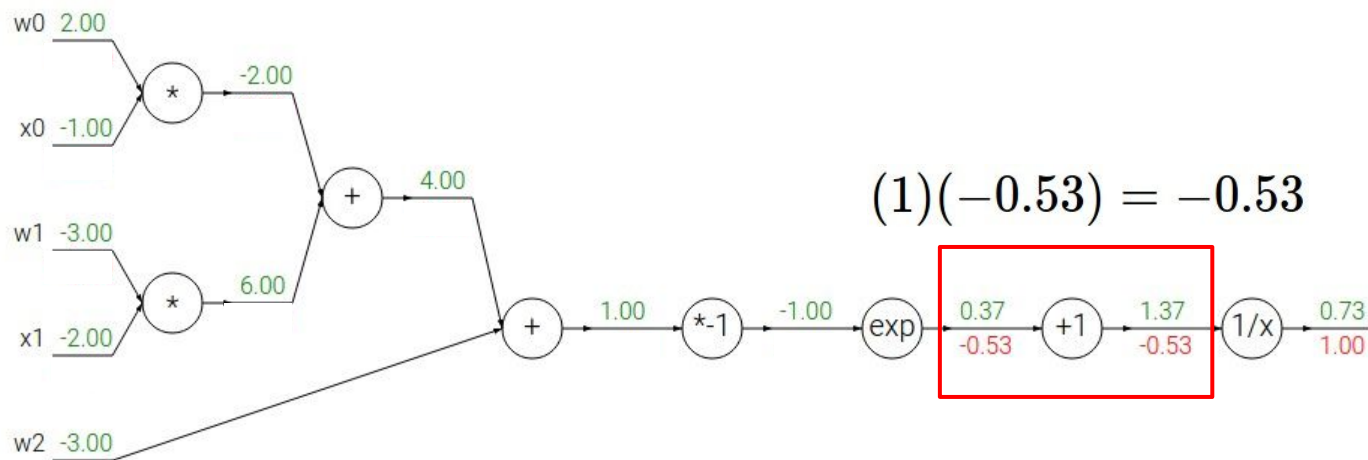
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$		$f_c(x) = c + x$	\rightarrow	$\frac{df}{dx} = 1$

Another example:

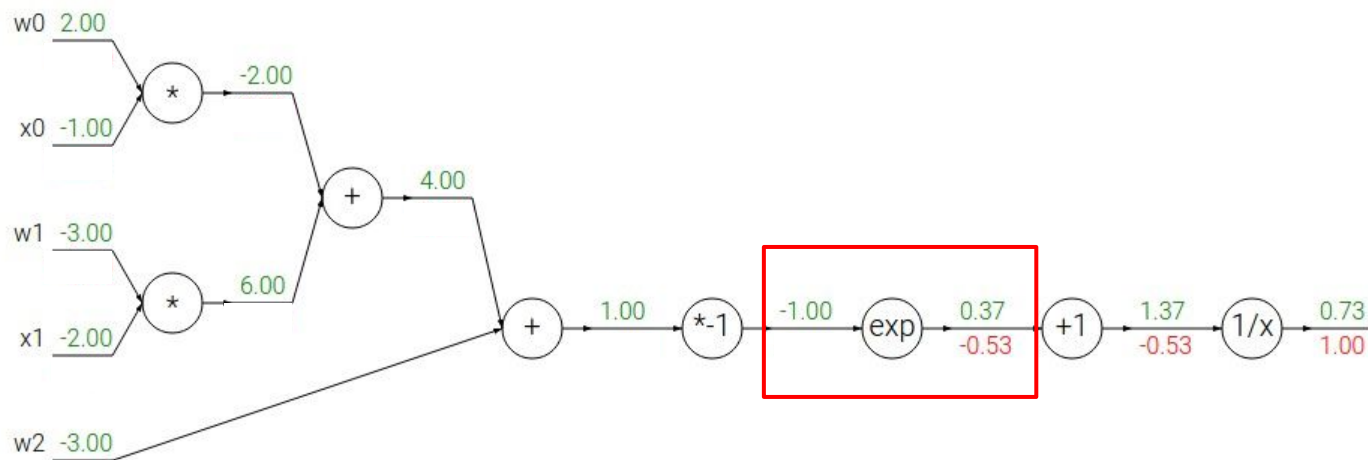
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$		$f_c(x) = c + x$	\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

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

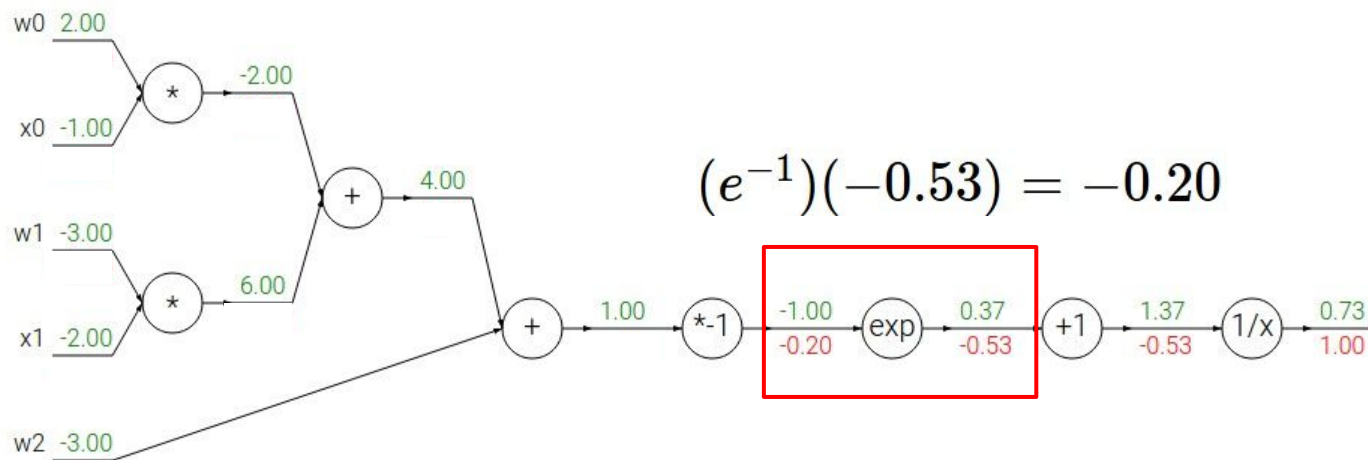
$$f_c(x) = c + x$$

\rightarrow

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

Another example:

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



$$(e^{-1})(-0.53) = -0.20$$

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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

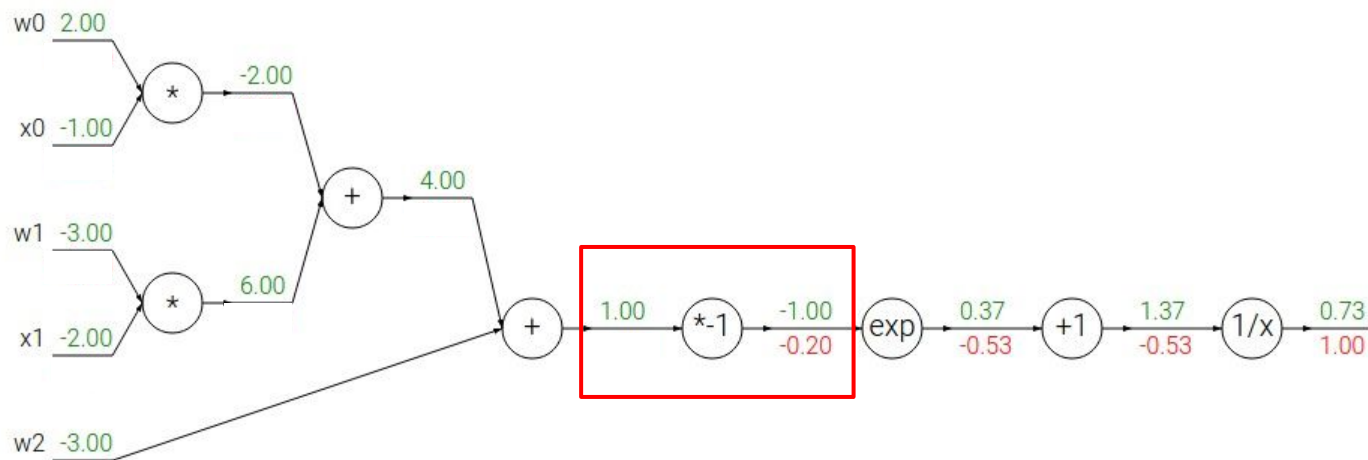
$$f_c(x) = c + x$$

→

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

Another example:

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



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

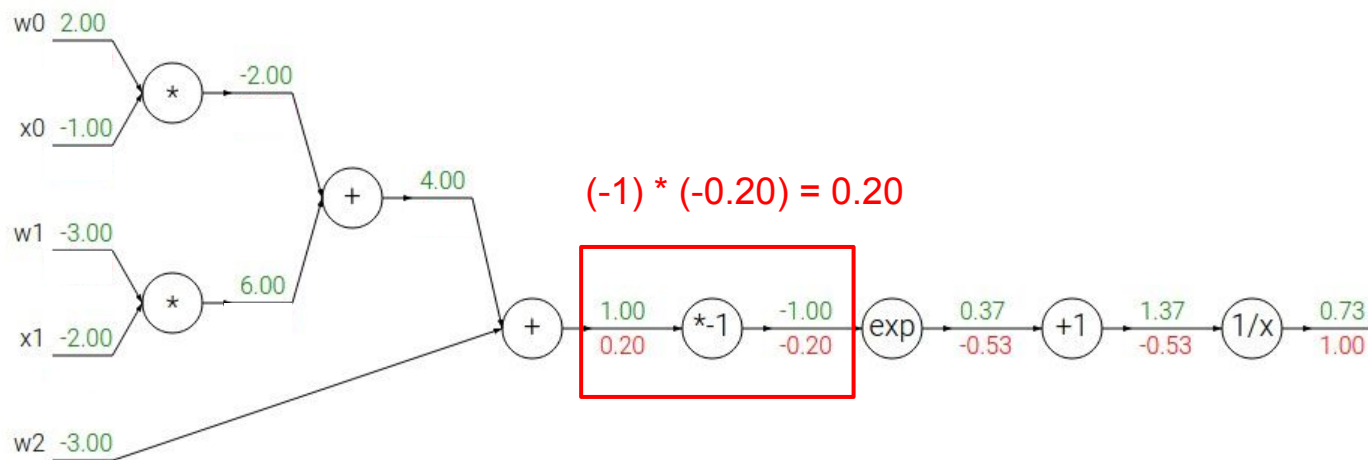
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

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

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

Another example:

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



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

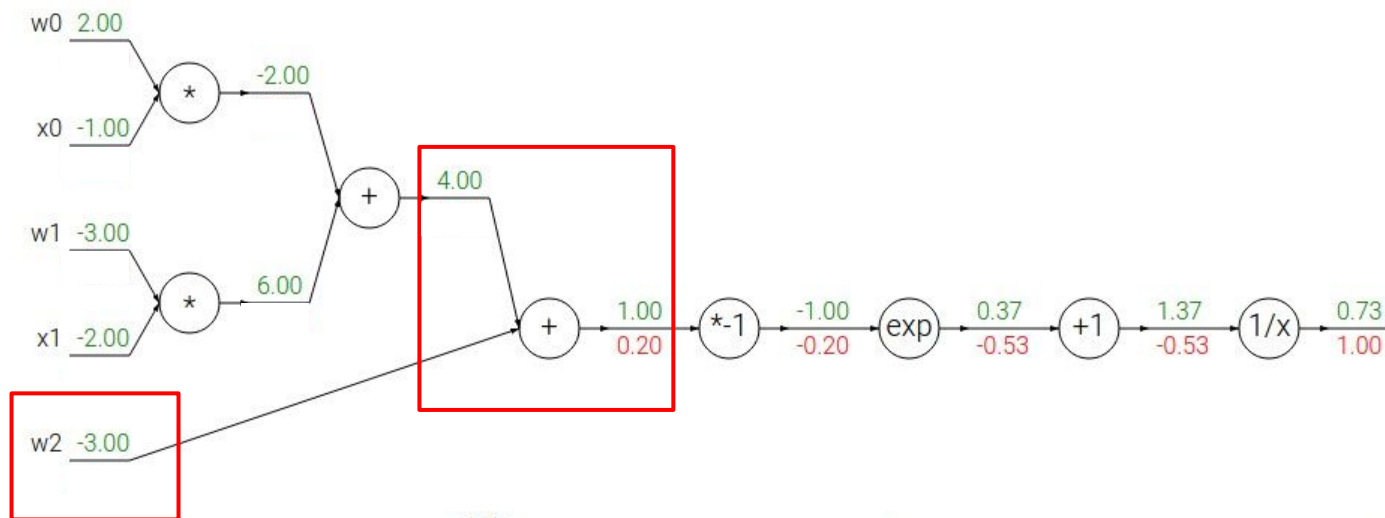
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

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

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

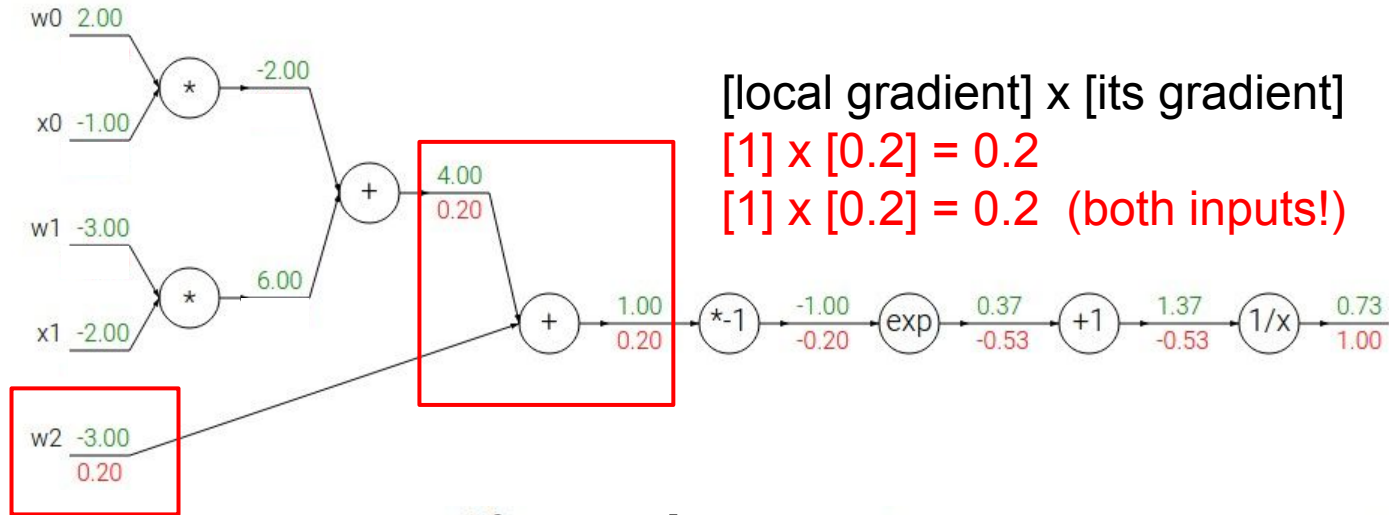
$$f_c(x) = c + x$$

→

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

Another example:

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



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

→

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

$$f_a(x) = ax$$

→

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

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

→

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

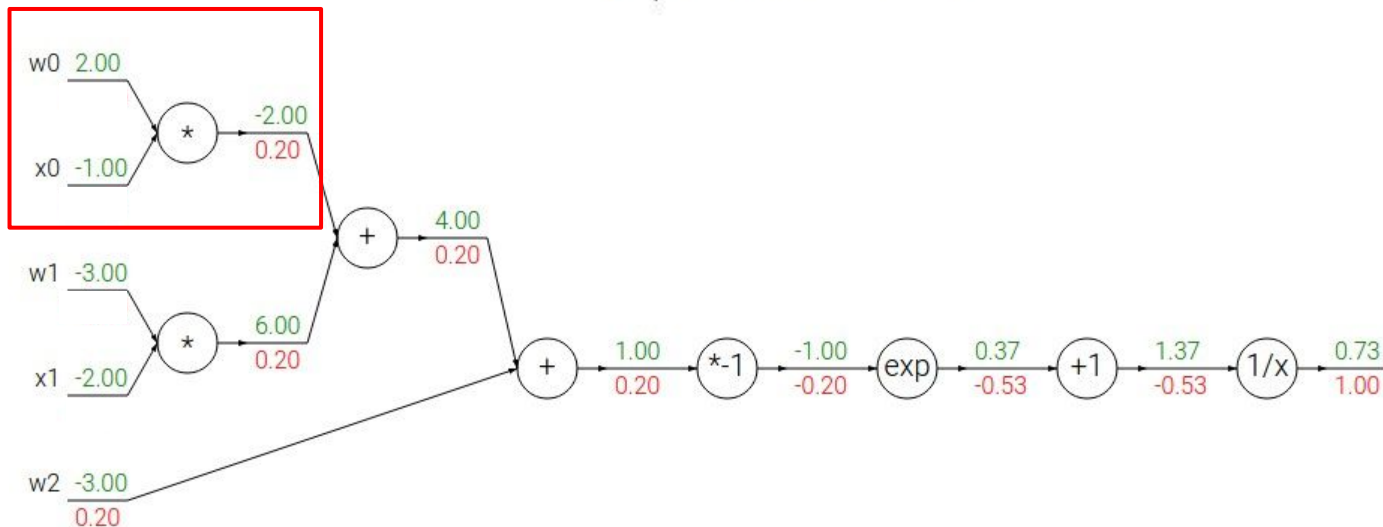
$$f_c(x) = c + x$$

→

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

Another example:

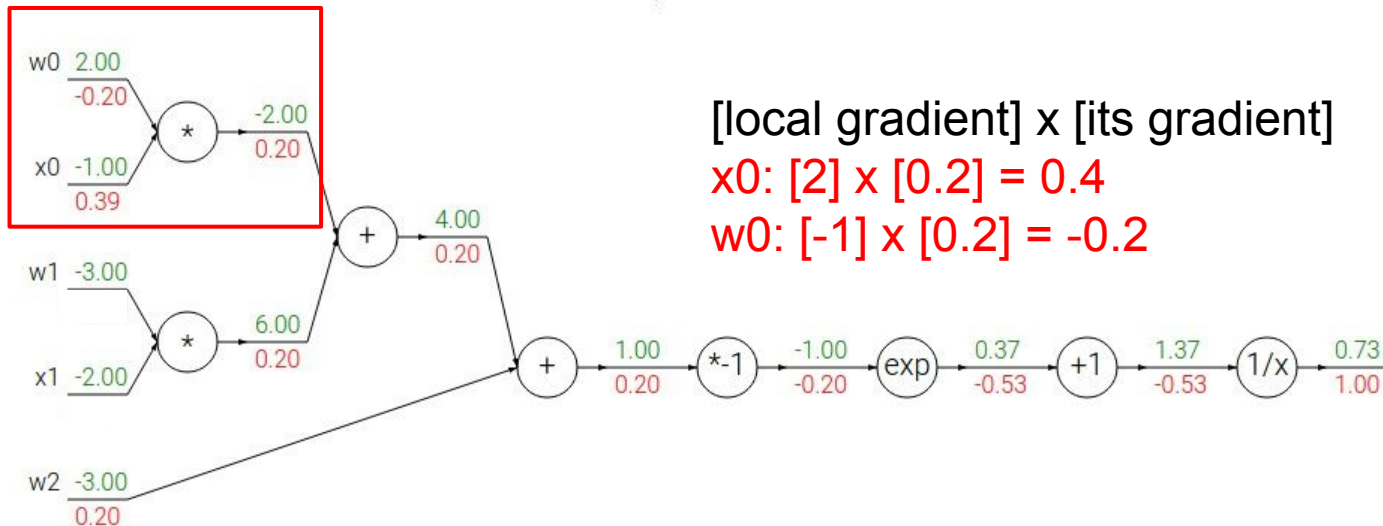
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$		$f_c(x) = c + x$	\rightarrow	$\frac{df}{dx} = 1$

Another example:

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



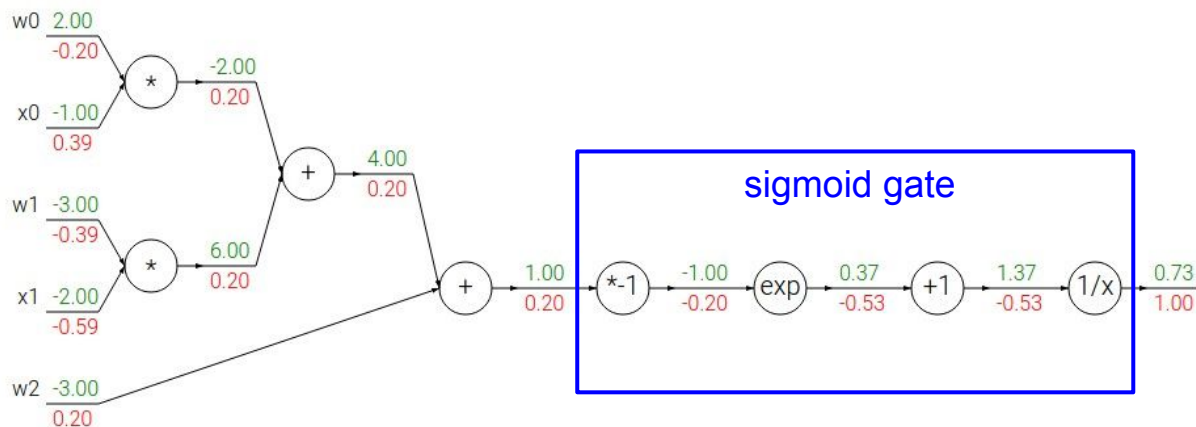
$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$		$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$		$f_c(x) = c + x$	\rightarrow	$\frac{df}{dx} = 1$

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$

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

sigmoid function

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

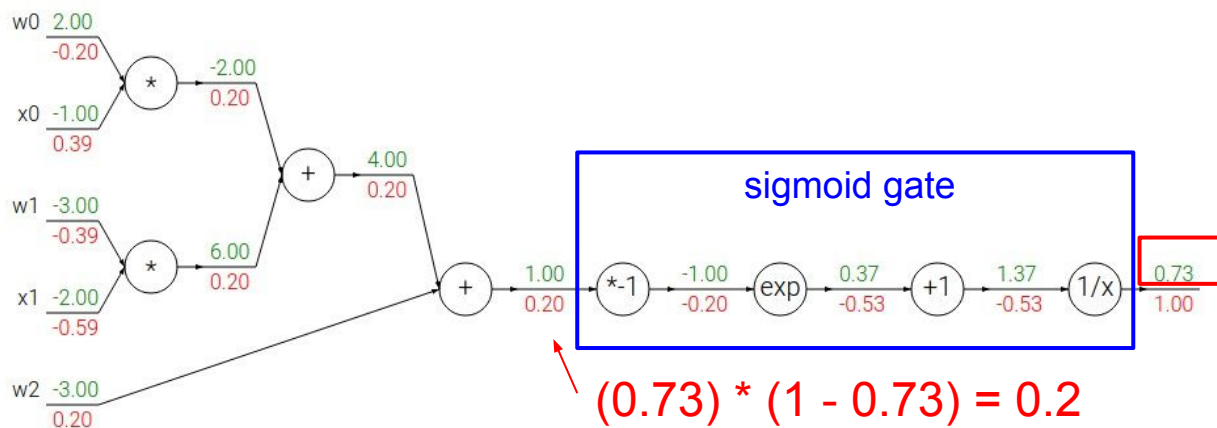


$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$

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

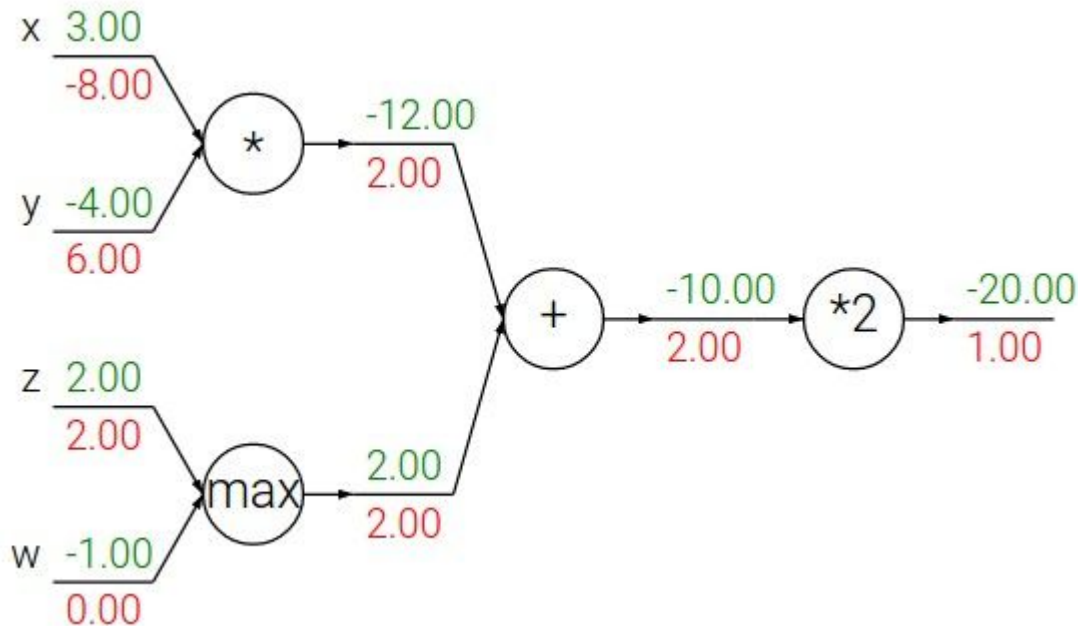
sigmoid function

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

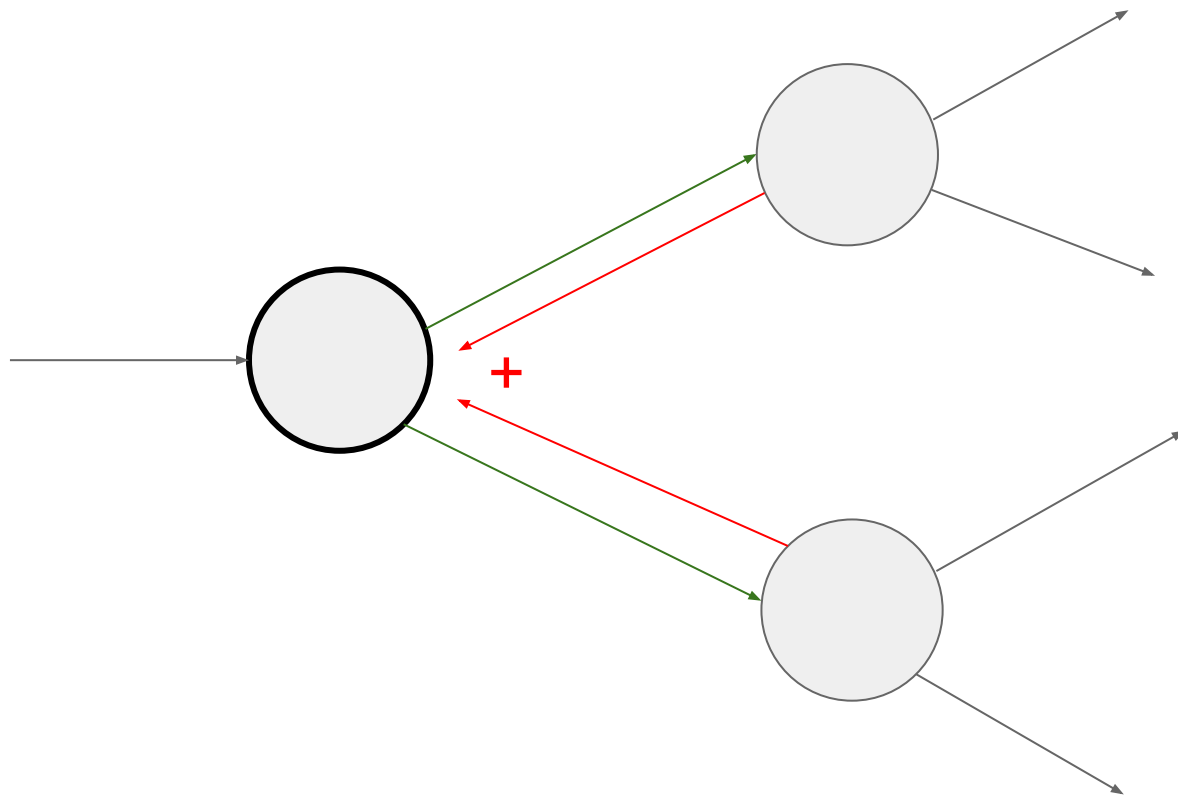


Patterns in backward flow

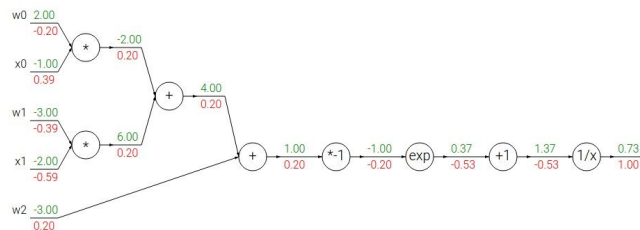
add gate: gradient distributor
max gate: gradient router
mul gate: gradient... “switcher”?



Gradients add at branches



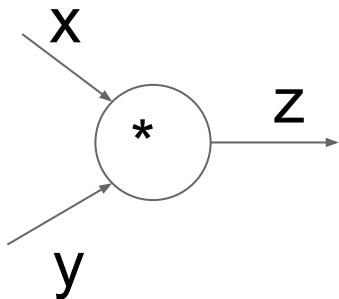
Implementation: forward/backward API



Graph (or Net) object. (*Rough psuedo 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]
```

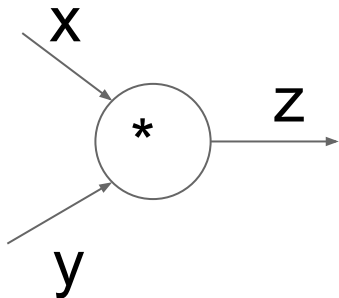
$$\frac{\partial L}{\partial z}$$

An arrow points from this box to the 'dz' parameter in the backward method signature of the code block above.

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

An arrow points from this box to the 'dx' element in the return list of the backward method in the code block above.

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

[illegible][illegible]

Example: Torch Layers

[illegible][illegible]

Example: Torch MulConstant

$$f(X) = aX$$

initialization

forward()

























































































backward()

```
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
```

Example: Caffe Layers

mate / cafe / src / cafe / layers /		New file	Find file	History				
🔗 login enable dated deconvolution (10)				Latest commit 86c4657 · 10 days ago				
	absorb_layer.cpp	absorb_layer.hpp		a month ago		memory_data_layer.cpp	memory_data_layer.hpp	a month ago
	absorb_layer.cu	absorb_layer.hpp		a month ago		multiscale_sagittal_layer.hpp	multiscale_sagittal_layer.hpp	a month ago
	accuracy_layer.cpp	absorb_layer.hpp		a month ago		nrmn_layer.cpp	memory_data_layer.hpp	a month ago
	argmax_layer.cpp	absorb_layer.hpp		a month ago		nrmn_layer.cpp	memory_data_layer.hpp	a month ago
	base_conv_layer.cpp	enable dated deconvolution	15 days ago			pooling_layer.cpp	memory_data_layer.hpp	a month ago
	base_data_layer.cpp	absorb_layer.hpp		a month ago		pooling_layer.cu	memory_data_layer.hpp	a month ago
	base_data_layer.cu	absorb_layer.hpp		a month ago		power_layer.cpp	memory_data_layer.hpp	a month ago
	batch_norm_layer.hpp	absorb_layer.hpp		a month ago		power_layer.cu	memory_data_layer.hpp	a month ago
	batch_norm_layer.cu	absorb_layer.hpp		a month ago		prelu_layer.cpp	memory_data_layer.hpp	a month ago
	batch_redund_layer.cpp	absorb_layer.hpp		a month ago		prelu_layer.cu	memory_data_layer.hpp	a month ago
	batch_redund_layer.cu	absorb_layer.hpp		a month ago		reduction_layer.cpp	memory_data_layer.hpp	a month ago
	brn_layer.cpp	absorb_layer.hpp		a month ago		reduction_layer.cu	memory_data_layer.hpp	a month ago
	brn_layer.cu	absorb_layer.hpp		a month ago		rnn_layer.cpp	memory_data_layer.hpp	a month ago
	convst_layer.cpp	absorb_layer.hpp		a month ago		rnn_layer.cu	memory_data_layer.hpp	a month ago
	convst_layer.cu	absorb_layer.hpp		a month ago		reshaper_layer.cpp	memory_data_layer.hpp	a month ago
	convstative_data_layer.cpp	absorb_layer.hpp		a month ago		sigmoid_cross_entropy_loss...	memory_data_layer.hpp	a month ago
	convstative_rnn_layer.cu	absorb_layer.hpp		a month ago		sigmoid_cross_entropy_loss...	memory_data_layer.hpp	a month ago
	convst_layer.cu	add support for 3D dated convolution	15 days ago			sigmoid_layer.cpp	memory_data_layer.hpp	a month ago
	convst_layer.cu	absorb_layer.hpp		a month ago		sigmoid_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_data_layer.cpp	Fix CUDNNConvolutionLayer for cuDNN 14	a month ago			silence_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_data_layer.cu	absorb_layer.hpp		a month ago		silence_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_rnn_layer.cu	absorb_layer.hpp		a month ago		slice_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_rnn_layer.cu	absorb_layer.hpp		a month ago		slice_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_rnn_layer.cu	absorb_layer.hpp		a month ago		softmax_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_pooling_layer.cpp	absorb_layer.hpp		a month ago		softmax_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_pooling_layer.cu	absorb_layer.hpp		a month ago		softmax_loss_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_rnn_layer.cpp	absorb_layer.hpp		a month ago		softmax_loss_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_rnn_layer.cu	absorb_layer.hpp		a month ago		split_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_sigmoid_layer.cpp	absorb_layer.hpp		a month ago		split_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_sigmoid_layer.cu	absorb_layer.hpp		a month ago		ssg_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_softmax_layer.cu	absorb_layer.hpp		a month ago		ssg_layer.cu	memory_data_layer.hpp	a month ago
	cudnn_soft_layer.cpp	absorb_layer.hpp		a month ago		tanh_layer.cpp	memory_data_layer.hpp	a month ago
	cudnn_soft_layer.cu	absorb_layer.hpp		a month ago		tanh_layer.cu	memory_data_layer.hpp	a month ago
	data_layer.cpp	absorb_layer.hpp		a month ago		threshold_layer.cpp	memory_data_layer.hpp	a month ago
	decnn_layer.cpp	enable dated deconvolution	15 days ago			threshold_layer.cu	memory_data_layer.hpp	a month ago
	decnn_layer.cu	absorb_layer.hpp		a month ago		tss_layer.cpp	memory_data_layer.hpp	a month ago
	dropout_layer.cpp	absorb_layer.hpp		a month ago		tss_layer.cu	memory_data_layer.hpp	a month ago
	dropout_layer.cu	absorb_layer.hpp		a month ago		window_data_layer.cpp	memory_data_layer.hpp	a month ago
	dummys_data_layer.cpp	absorb_layer.hpp		a month ago				
	ethnet_layer.cpp	absorb_layer.hpp		a month ago				
	ethnet_layer.cu	absorb_layer.hpp		a month ago				
	embnet_layer.cpp	absorb_layer.hpp		a month ago				
	embnet_layer.cu	absorb_layer.hpp		a month ago				
	envidnet_layer.cpp	absorb_layer.hpp		a month ago				
	envidnet_layer.cu	absorb_layer.hpp		a month ago				
	exu_layer.cpp	absorb_layer.hpp		a month ago				
	exu_layer.cu	absorb_layer.hpp		a month ago				
	fft_layer.cpp	absorb_layer.hpp		a month ago				
	fft_layer.cu	absorb_layer.hpp		a month ago				
	fatten_layer.cpp	absorb_layer.hpp		a month ago				

Caffe Sigmoid Layer

```
1 #include <cmath>
2 #include <vector>
3
4 #include "caffe/layers/sigmoid_layer.hpp"
5
6 namespace caffe {
7
8 template <typename Dtype>
9 inline Dtype sigmoid(Dtype x) {
10     return 1. / (1. + exp(-x));
11 }
12
13 template <typename Dtype>
14 void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
15     const vector<Blob<Dtype>*>& top) {
16     const Dtype* bottom_data = bottom[0]->cpu_data();
17     Dtype* top_data = top[0]->mutable_cpu_data();
18     const int count = bottom[0]->count();
19     for (int i = 0; i < count; ++i) {
20         top_data[i] = sigmoid(bottom_data[i]);
21     }
22 }
```

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

```
24 template <typename Dtype>
25 void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
26     const vector<bool>& propagate_down,
27     const vector<Blob<Dtype>*>& bottom) {
28     if (propagate_down[0]) {
29         const Dtype* top_data = top[0]->cpu_data();
30         const Dtype* top_diff = top[0]->cpu_diff();
31         Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
32         const int count = bottom[0]->count();
33         for (int i = 0; i < count; ++i) {
34             const Dtype sigmoid_x = top_data[i];
35             bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
36         }
37     }
38 }
```

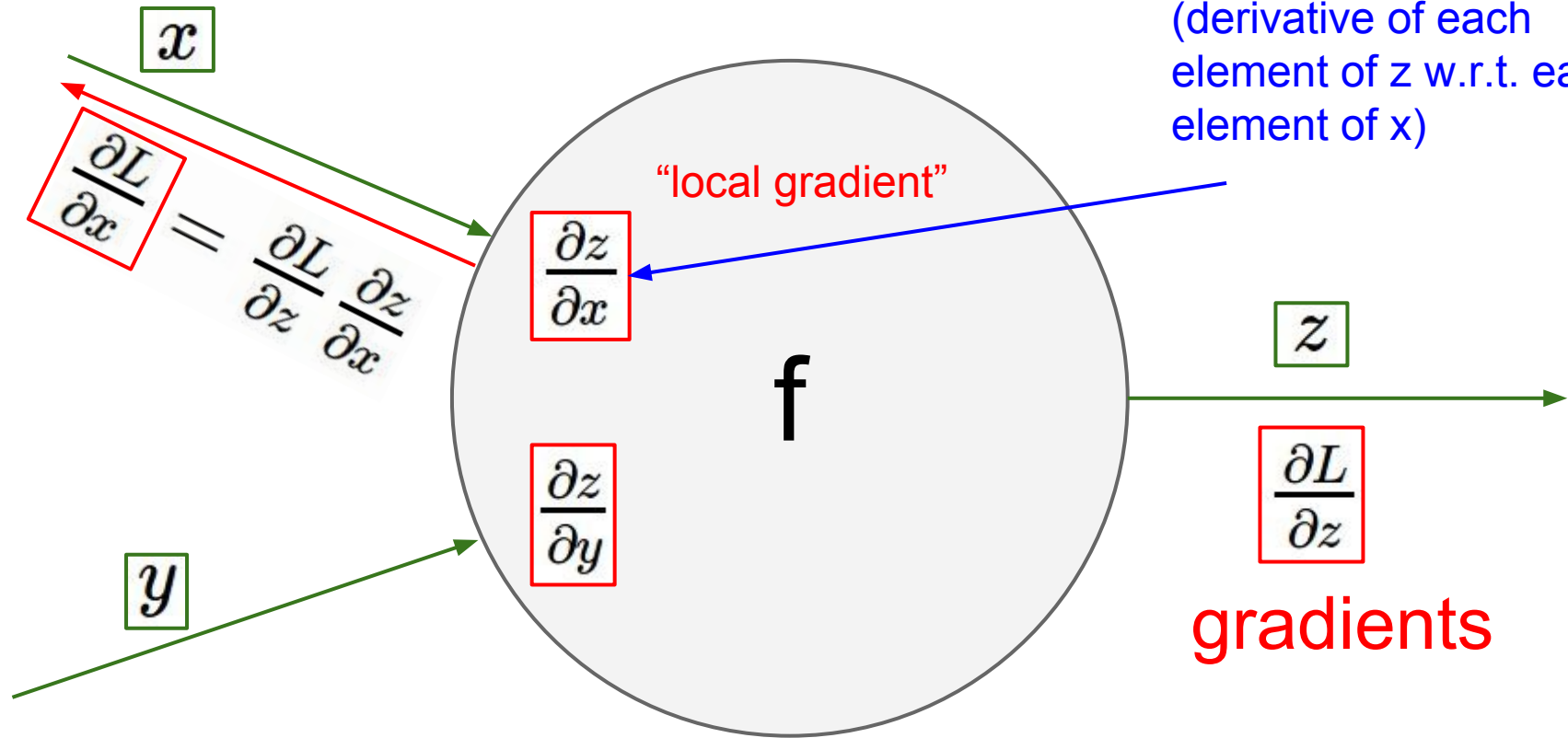
$$(1 - \sigma(x)) \sigma(x) * \text{top_diff} \quad (\text{chain rule})$$

```
40 #ifdef CPU_ONLY
41 STUB_GPU(SigmoidLayer);
42 #endif
43
44 INSTANTIATE_CLASS(SigmoidLayer);
45
46
47 } // namespace caffe
```

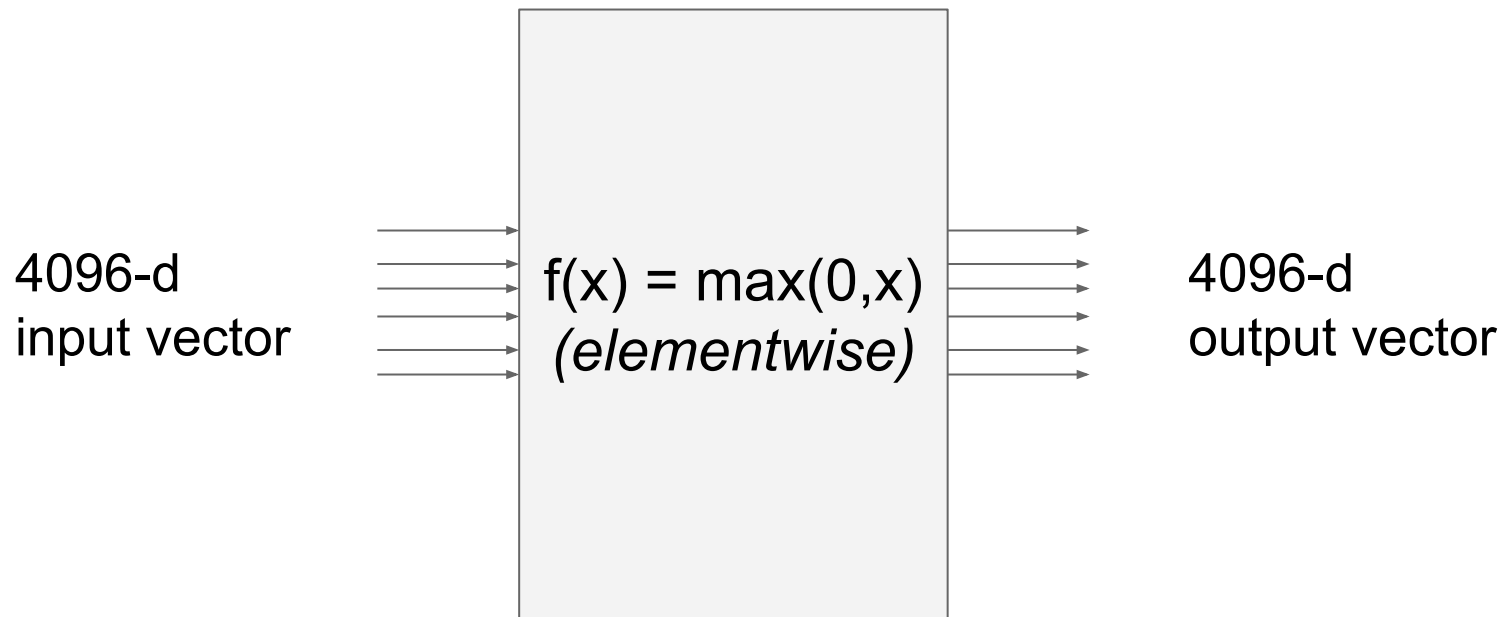
Gradients for vectorized code

(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)



Vectorized operations

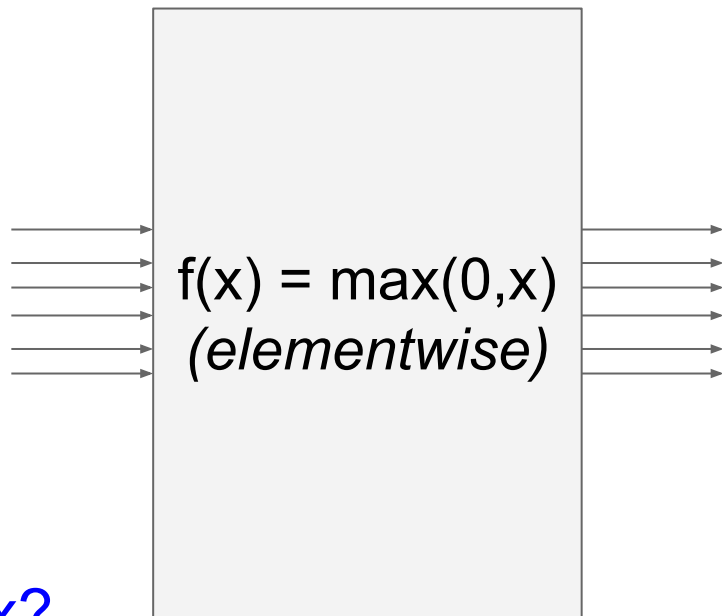


Vectorized operations

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

Jacobian matrix

4096-d
input vector



4096-d
output vector

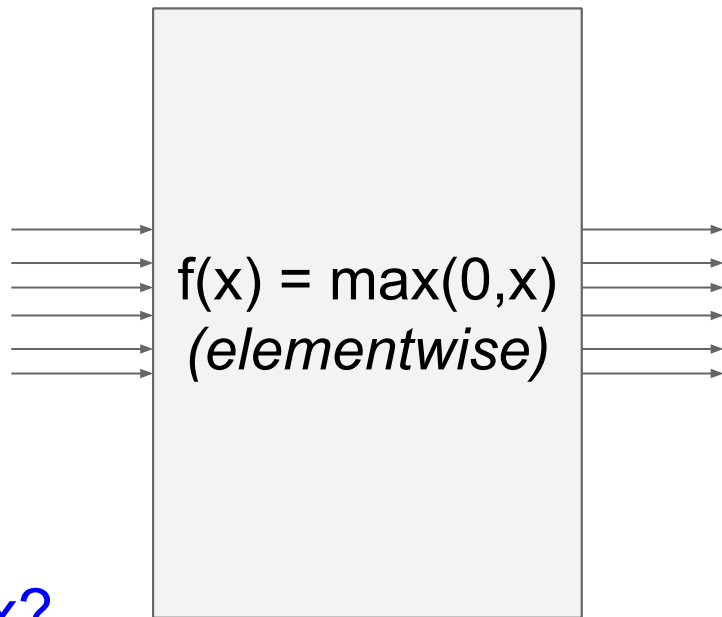
Q: what is the
size of the
Jacobian matrix?

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

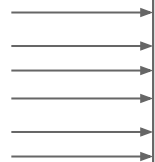
Q: what is the
size of the
Jacobian matrix?
[4096 x 4096!]

Q2: what does it
look like?

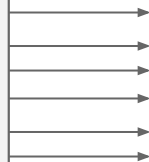
Vectorized operations

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

100 4096-d
input vectors



$f(x) = \max(0, x)$
(*elementwise*)



100 4096-d
output vectors

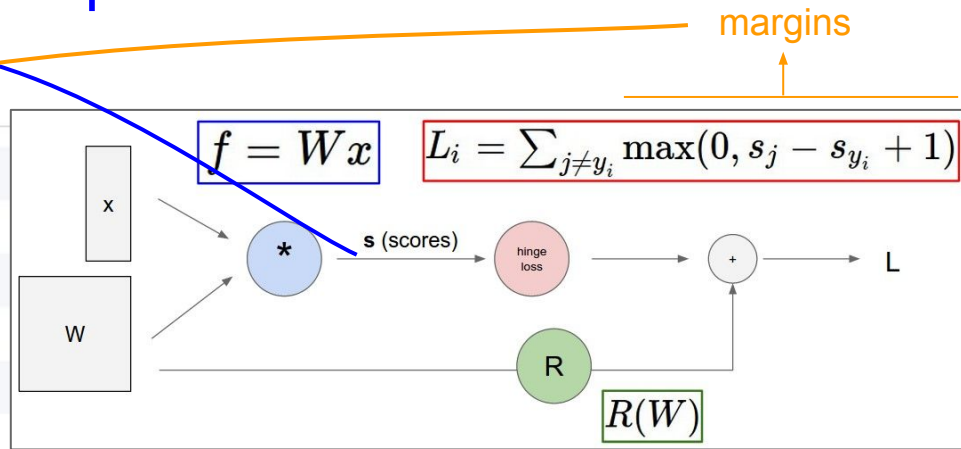
i.e. Jacobian would technically be a [409,600 x 409,600] matrix :\'

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.



Neural Network: without the brain stuff

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

Neural Network: without the brain stuff

(**Before**) Linear score function:

$$f = Wx$$

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

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

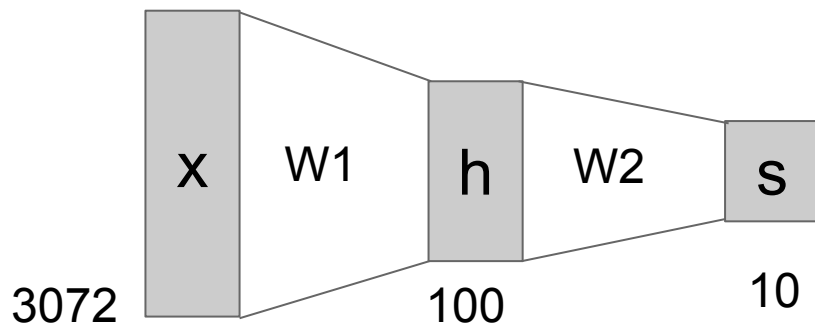
Neural Network: without the brain stuff

(**Before**) Linear score function:

$$f = Wx$$

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

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



Neural Network^{without the brain stuff}

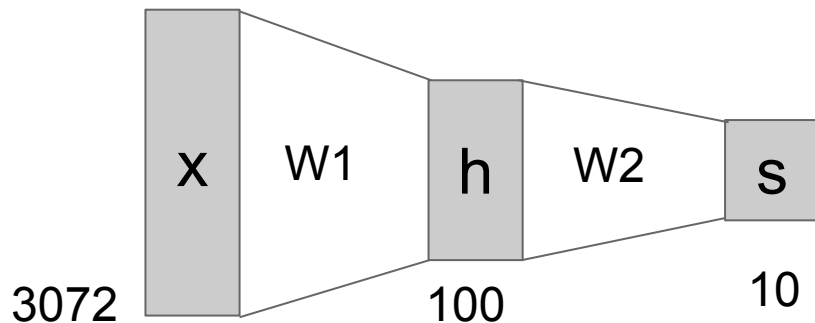


(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network

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



Neural Network: without the brain stuff

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

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

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

Full implementation of training a 2-layer Neural Network needs ~11 lines:

```
01. X = np.array([ [0,0,1], [0,1,1], [1,0,1], [1,1,1] ])
02. y = np.array([[0,1,1,0]]).T
03. syn0 = 2*np.random.random((3,4)) - 1
04. syn1 = 2*np.random.random((4,1)) - 1
05. for j in xrange(60000):
06.     l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
07.     l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
08.     l2_delta = (y - l2)*(l2*(1-l2))
09.     l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
10.     syn1 += l1.T.dot(l2_delta)
11.     syn0 += X.T.dot(l1_delta)
```

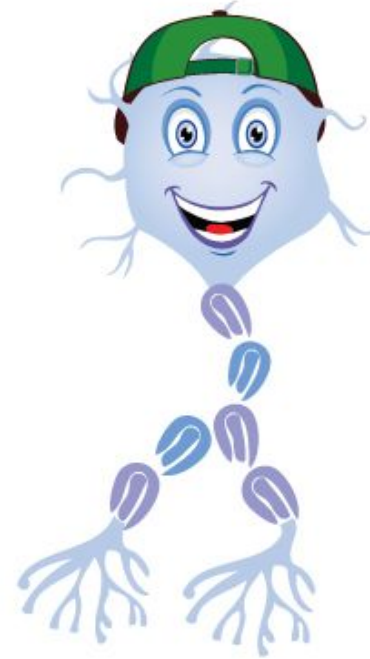
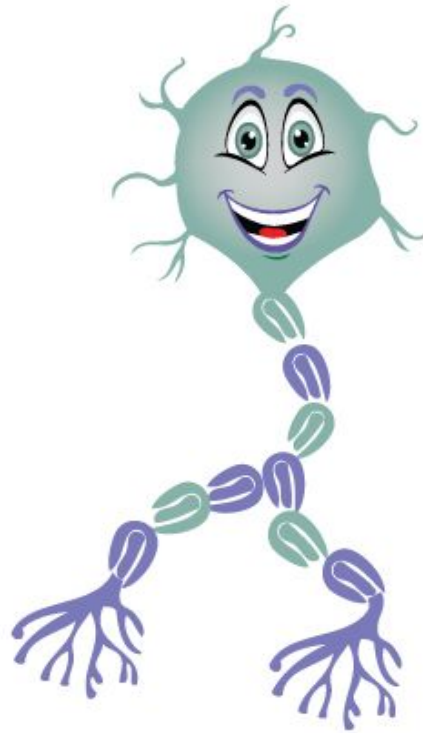
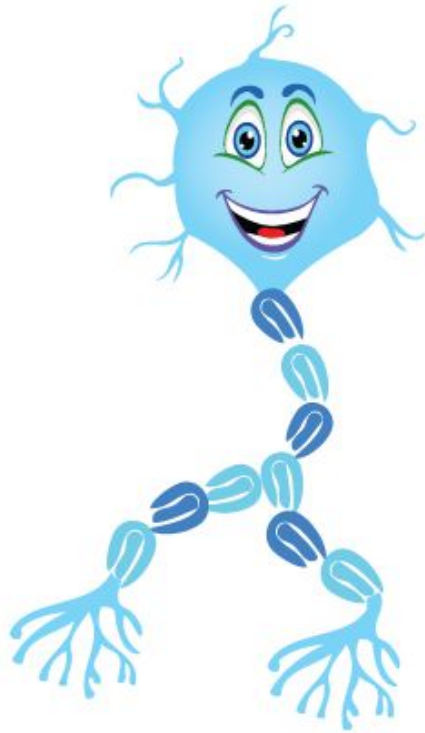
from @iamtrask, <http://iamtrask.github.io/2015/07/12/basic-python-network/>

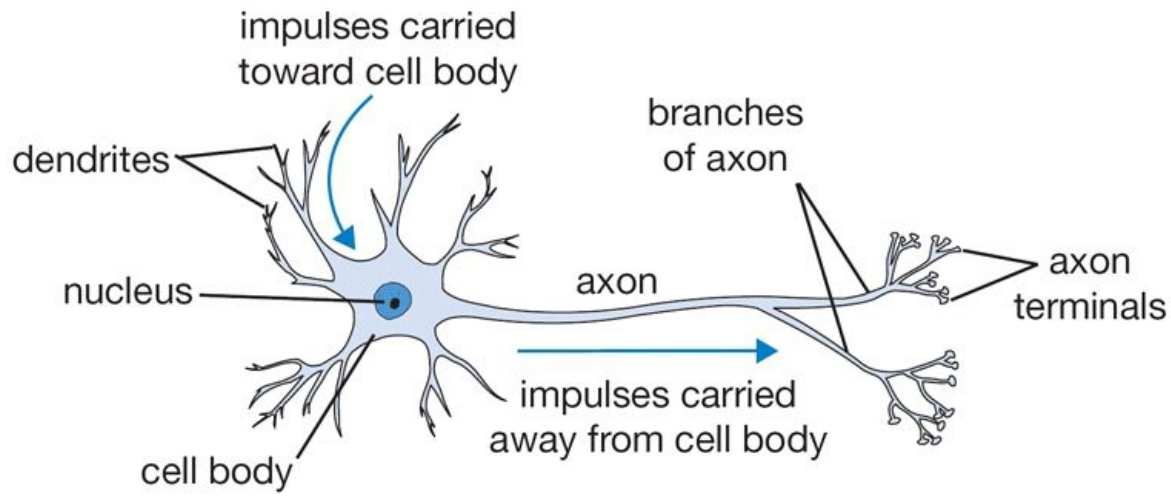
Assignment: Writing 2layer Net

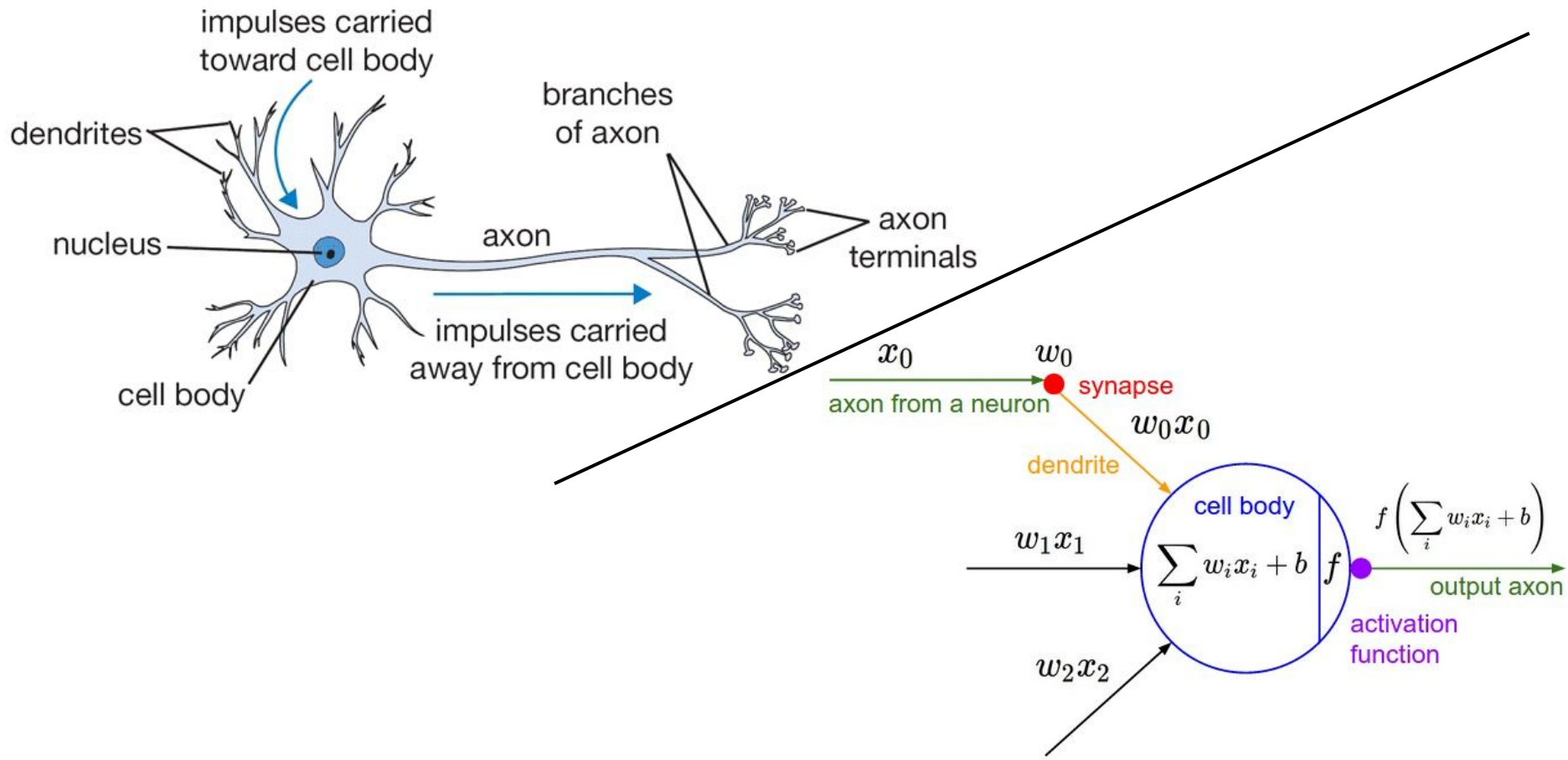
Stage your forward/backward computation!

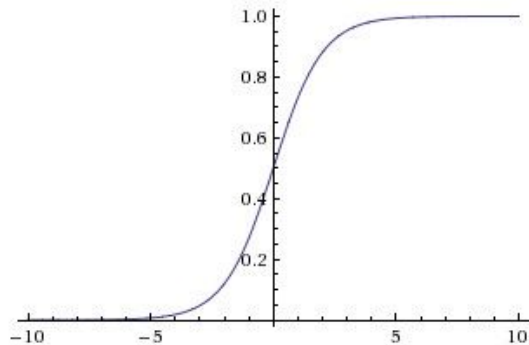
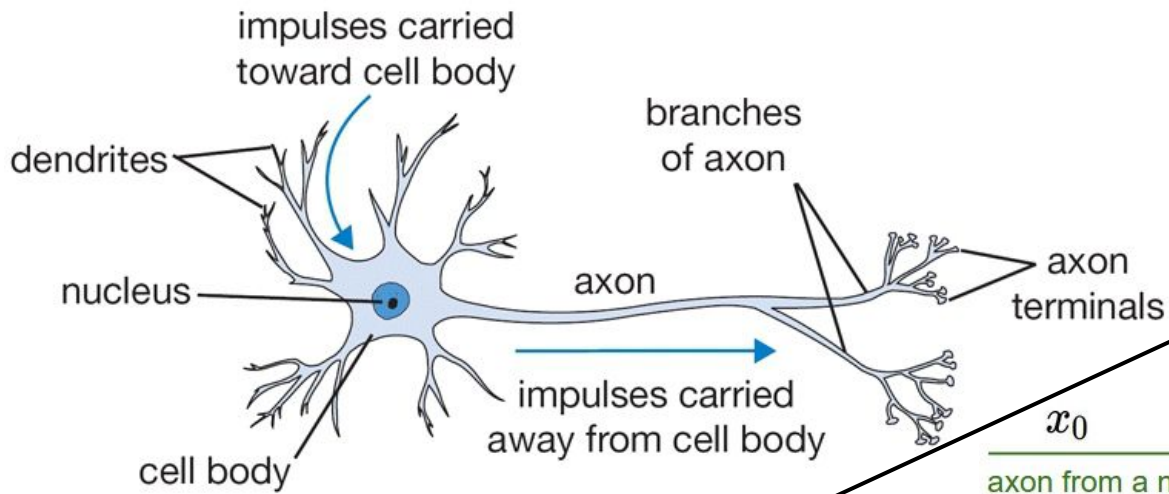
```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```





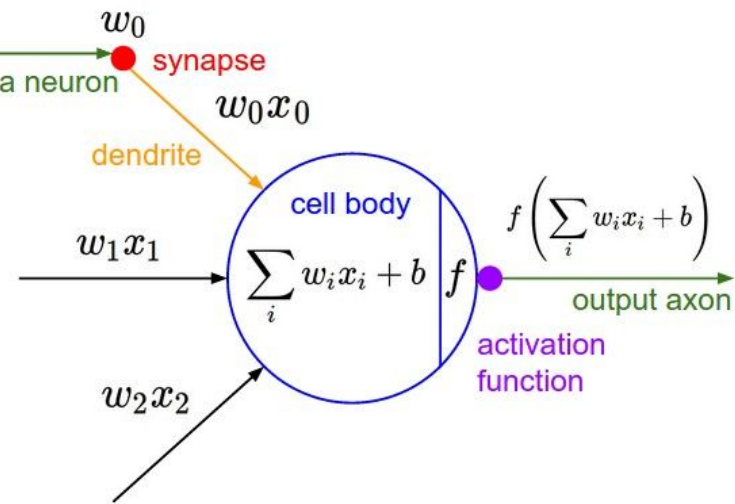


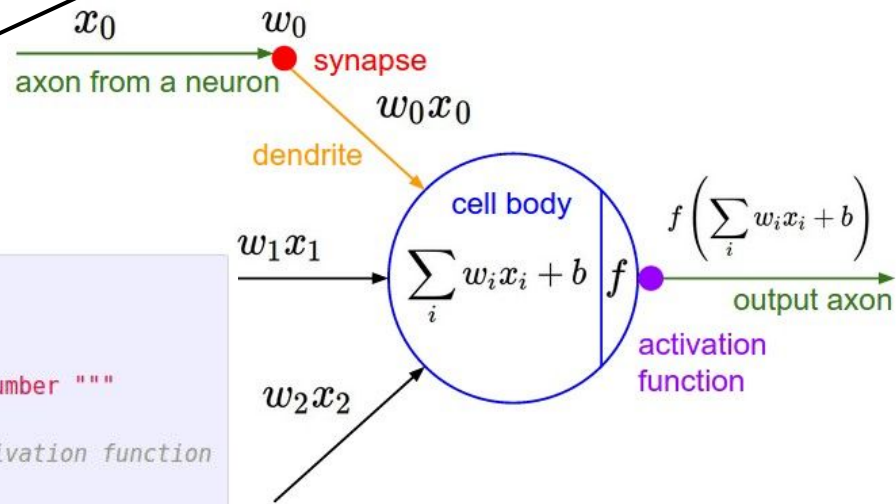
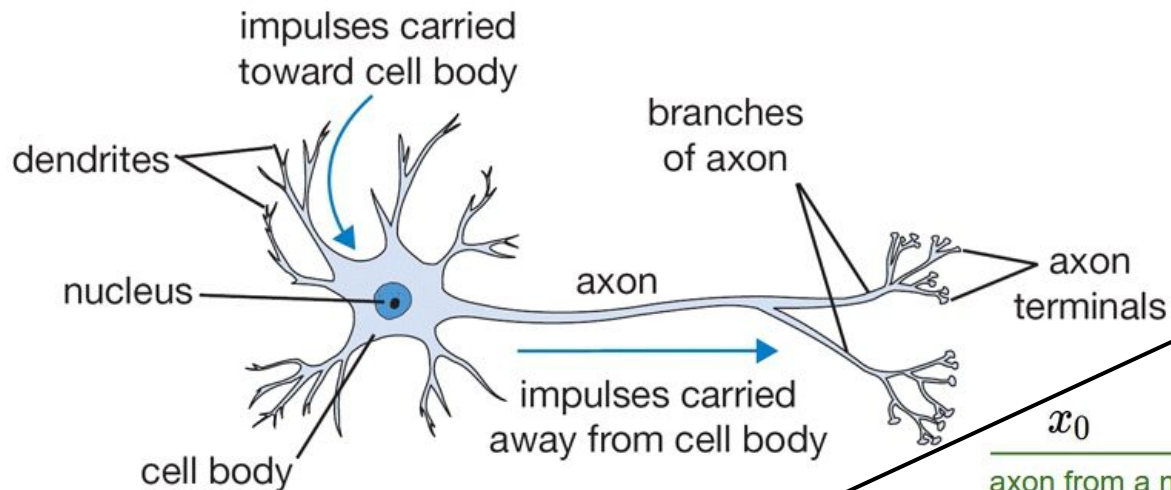




**sigmoid activation
function**

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



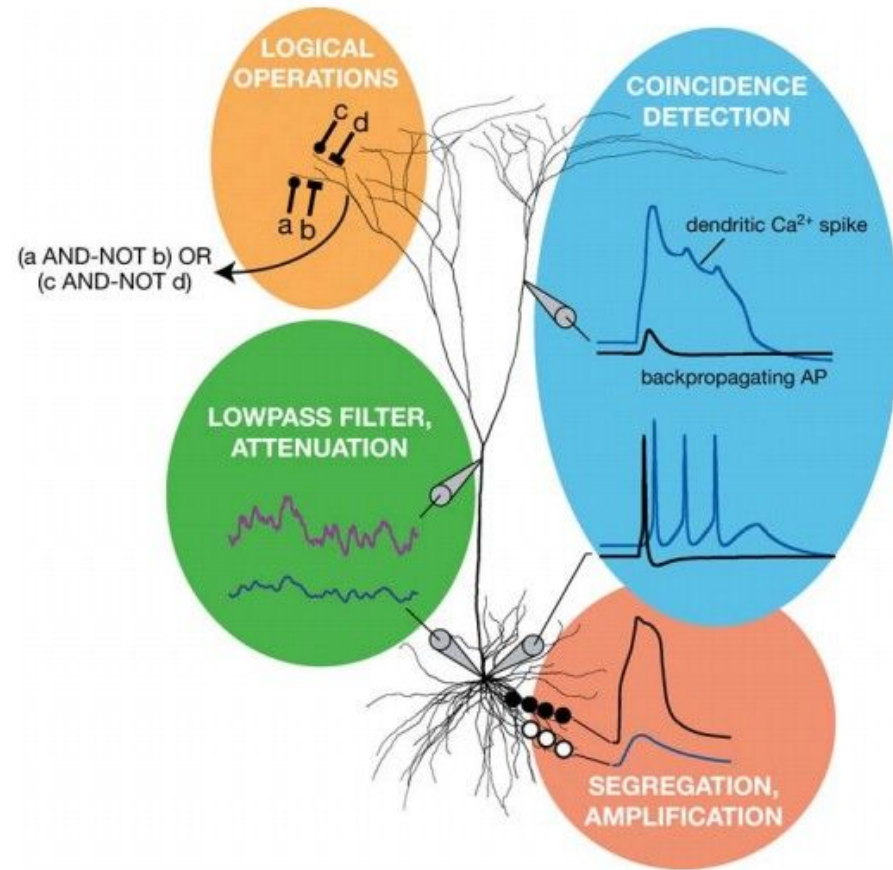


```
class Neuron:
    # ...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
  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

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
- Rate code may not be adequate

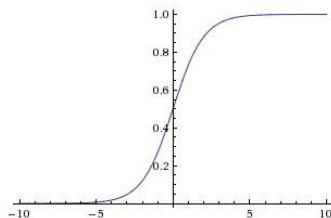


[Dendritic Computation. London and Hausser]

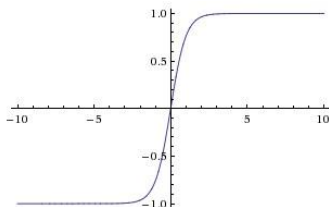
Activation Functions

Sigmoid

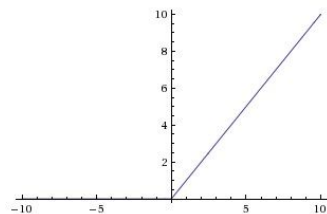
$$\sigma(x) = 1/(1 + e^{-x})$$



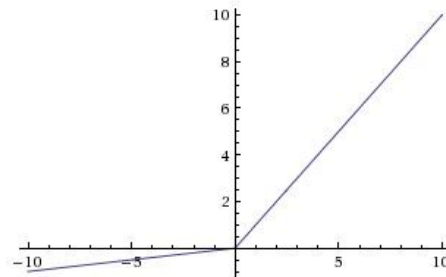
tanh $\tanh(x)$



ReLU $\max(0, x)$

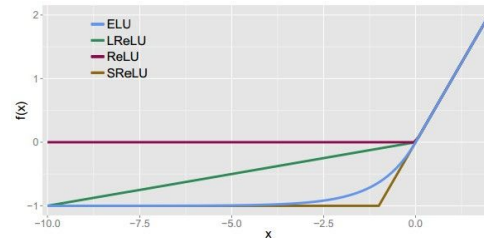


Leaky ReLU
 $\max(0.1x, x)$

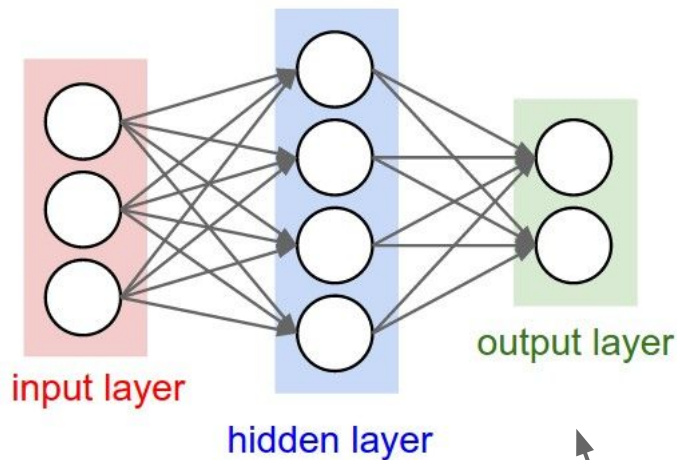


Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

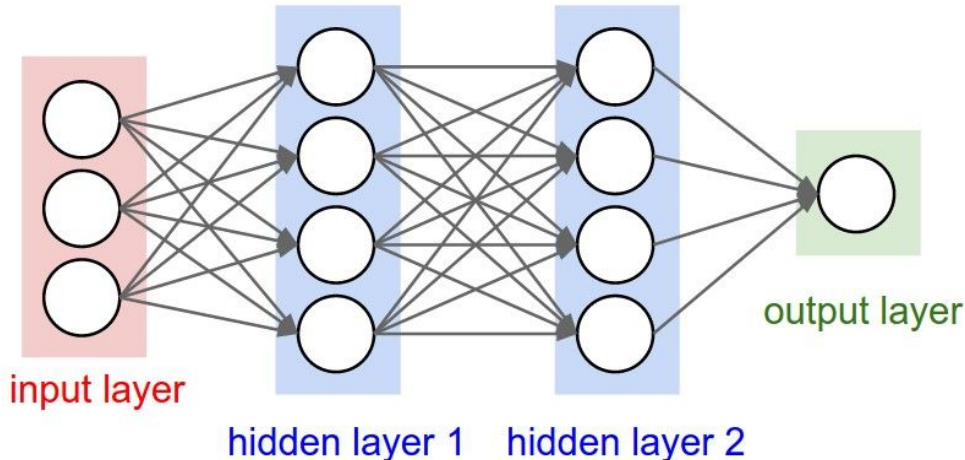
ELU
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



Neural Networks: Architectures



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”



“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

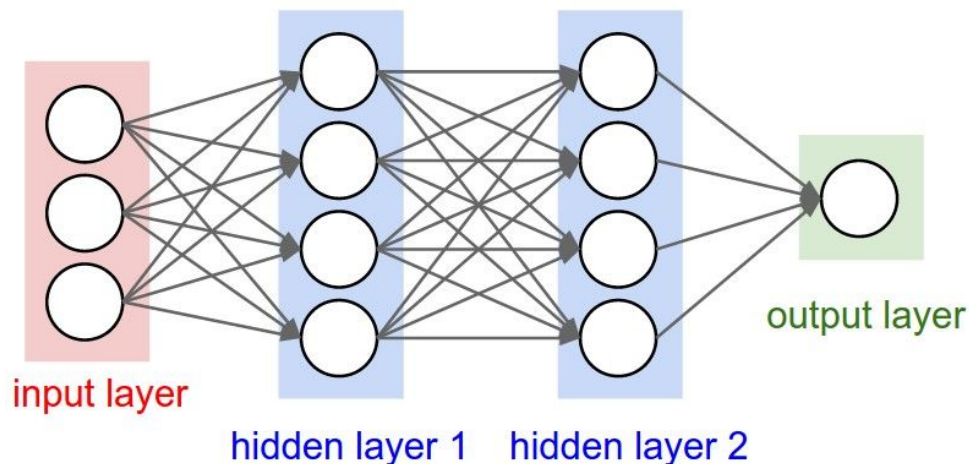
“Fully-connected” layers

Example Feed-forward computation of a Neural Network

```
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
```

We can efficiently evaluate an entire layer of neurons.

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)
```

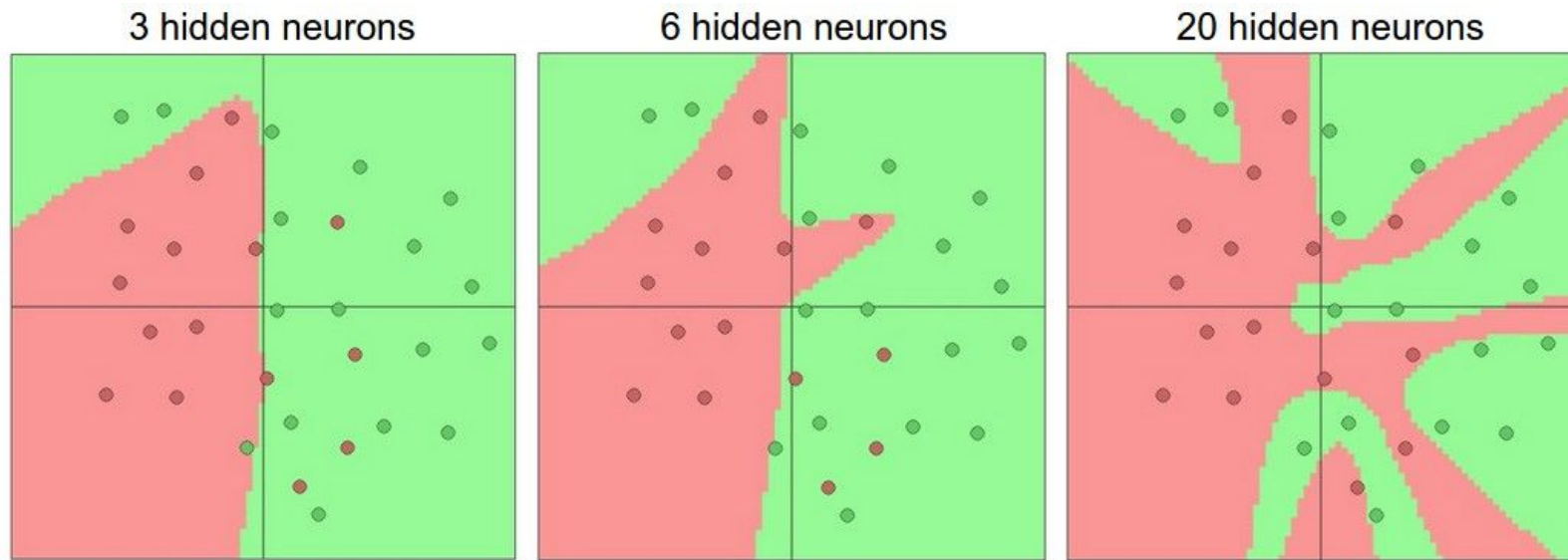
```
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
```

```
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)
```

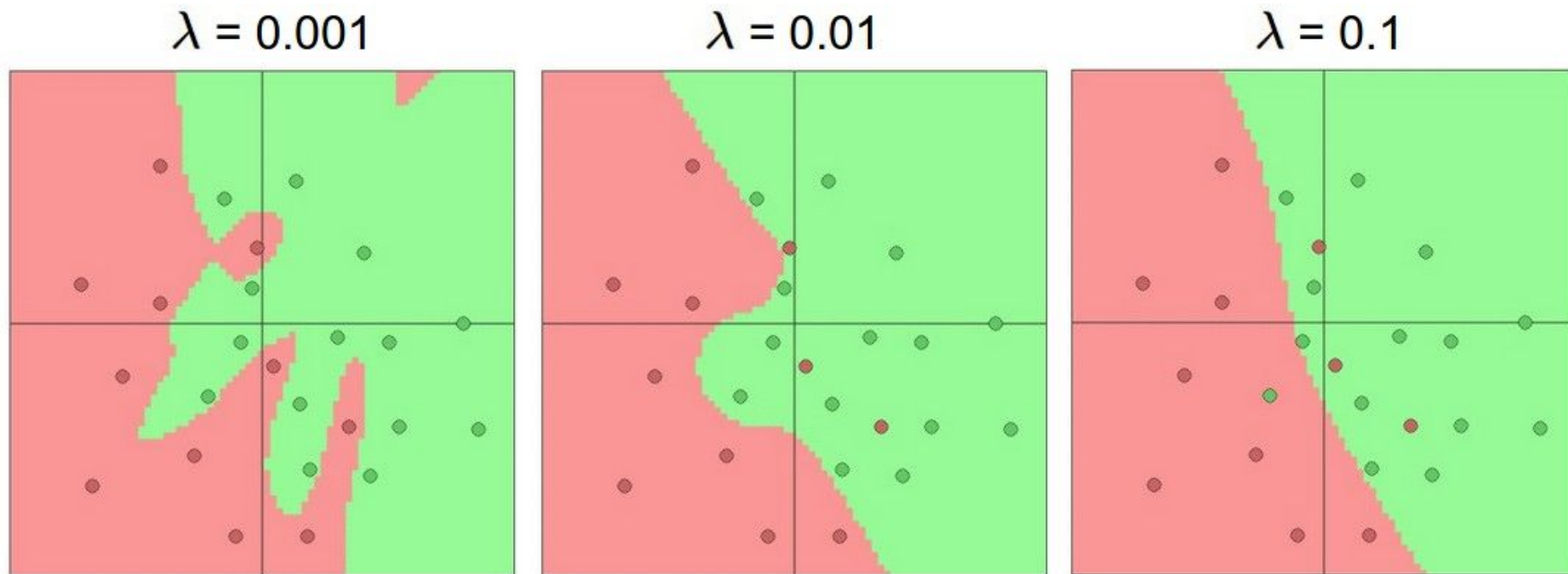
```
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

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:



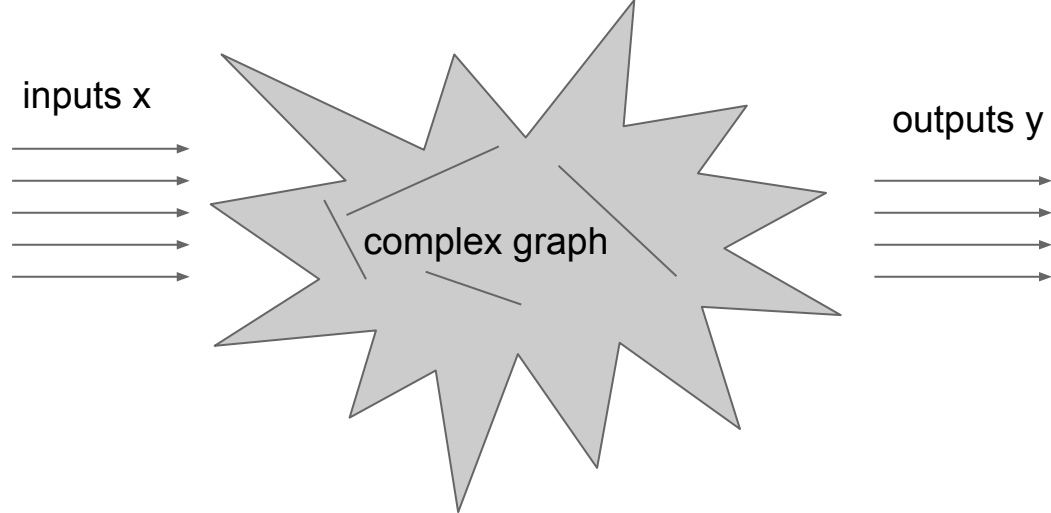
(you can play with this demo over at ConvNetJS: <http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>)

Summary

- we arrange neurons into fully-connected layers
- the abstraction of a **layer** has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- neural networks are not really *neural*
- neural networks: bigger = better (but might have to regularize more strongly)

Next Lecture:

More than you ever wanted to know about Neural Networks and how to train them.



← reverse-mode differentiation (if you want effect of many things on one thing)

$$\frac{\partial y}{\partial x} \text{ for many different } x$$

→ forward-mode differentiation (if you want effect of one thing on many things)

$$\frac{\partial y}{\partial x} \text{ for many different } y$$