Lecture 4: Backpropagation and Neural Networks part 1

박사과정 김성빈 chengbinjin@inha.edu,

지도교수 김학일 교수 hikim@inha.ac.kr

인하대학교 컴퓨터비전 연구실

Recall From Last Time…

Where we are…

$$s = f(x, \boldsymbol{W}) = \boldsymbol{W}x$$

Score function

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

SVM loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Softmax loss

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

Data loss + Regularization loss

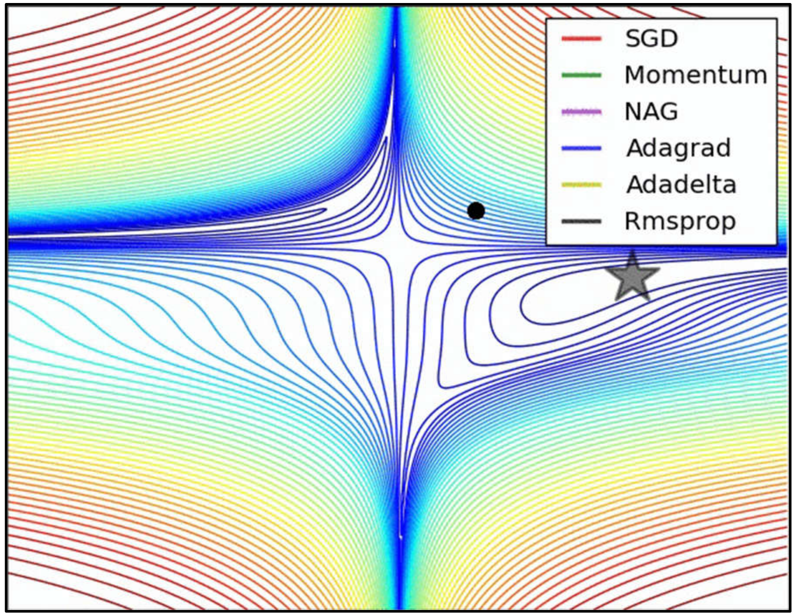
Want $\nabla_{\mathrm{W}} L$ = …

gradient of Loss respect to W

Slide from CS231n

Recall From Last Time…

Optimization




```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

(image credits to Alec Radford)

Recall From Last Time…

Gradient Descent

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

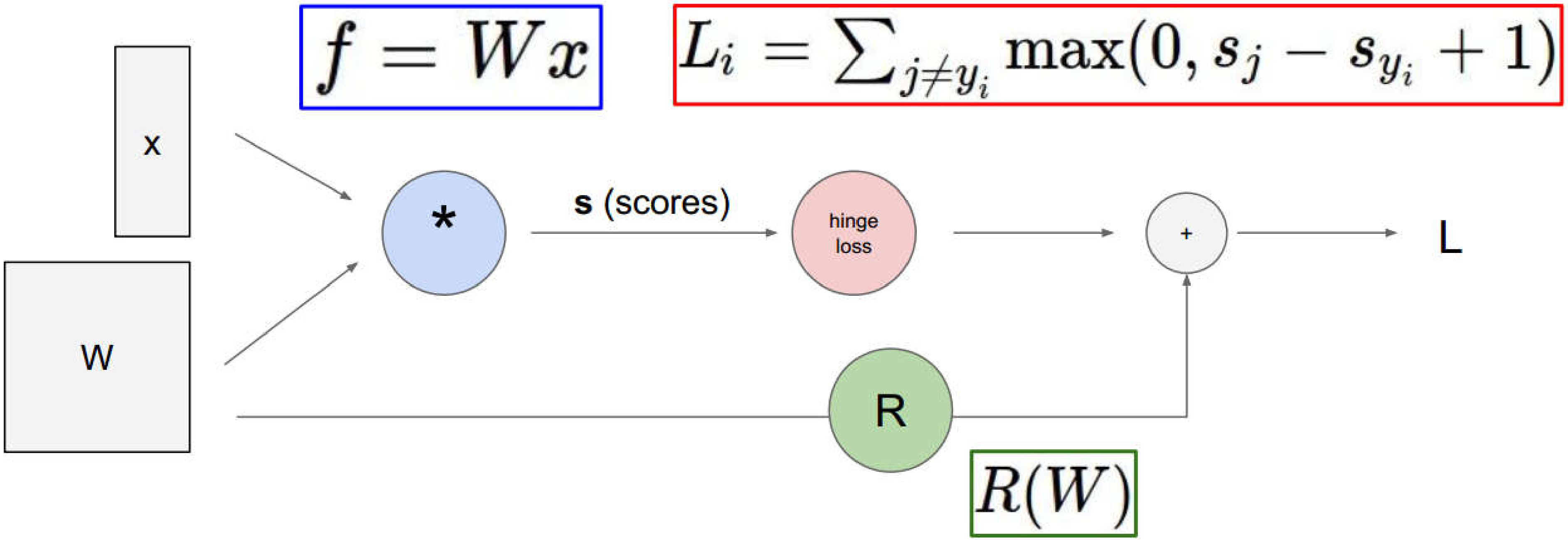
- **Numerical gradient**: approximate, slow, easy to write
- **Analytic gradient** (calculus): exact, fast, error-prone

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

Slide from CS231n

Backpropagation

Computational Graph



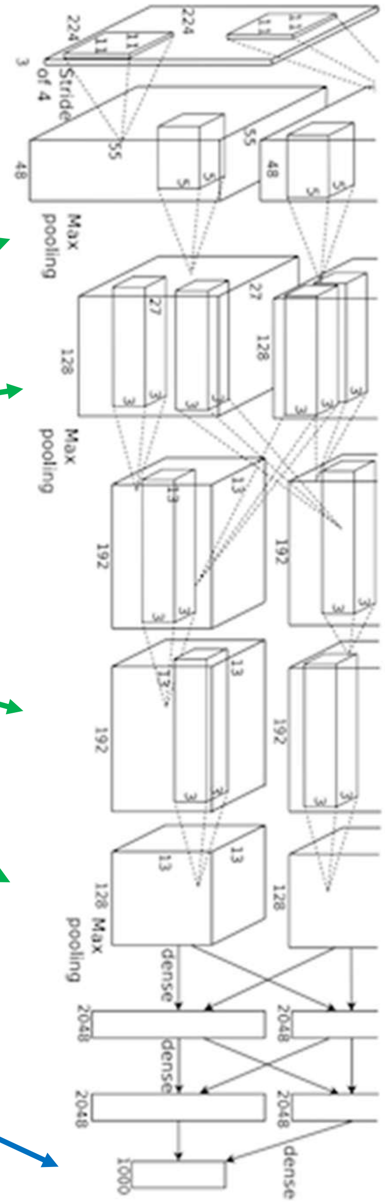
Computation graph of the SVM loss

Convolutional Network (AlexNet)

Input image

weights

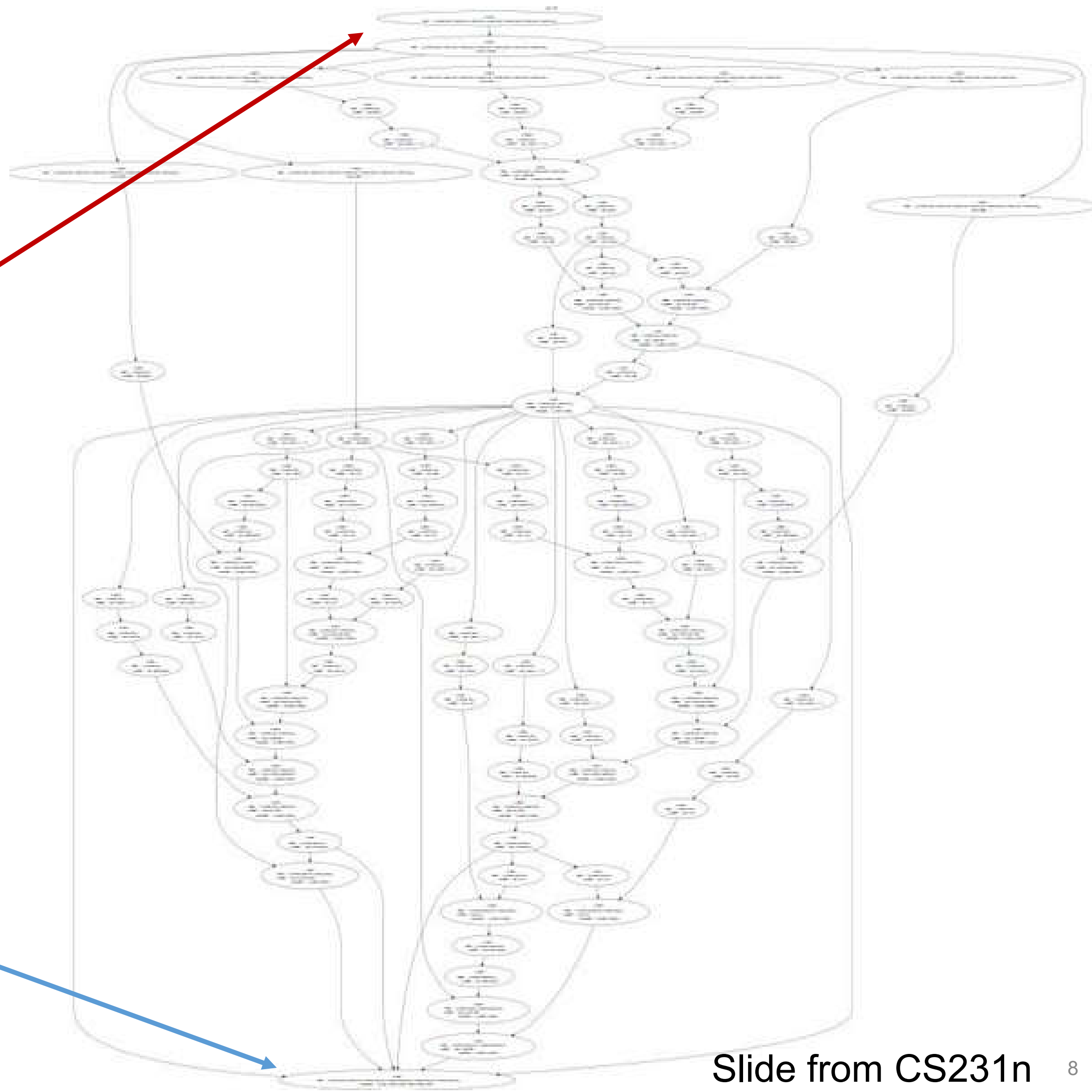
loss



Slide from CS231n

Neural Turing Machine

Input tape

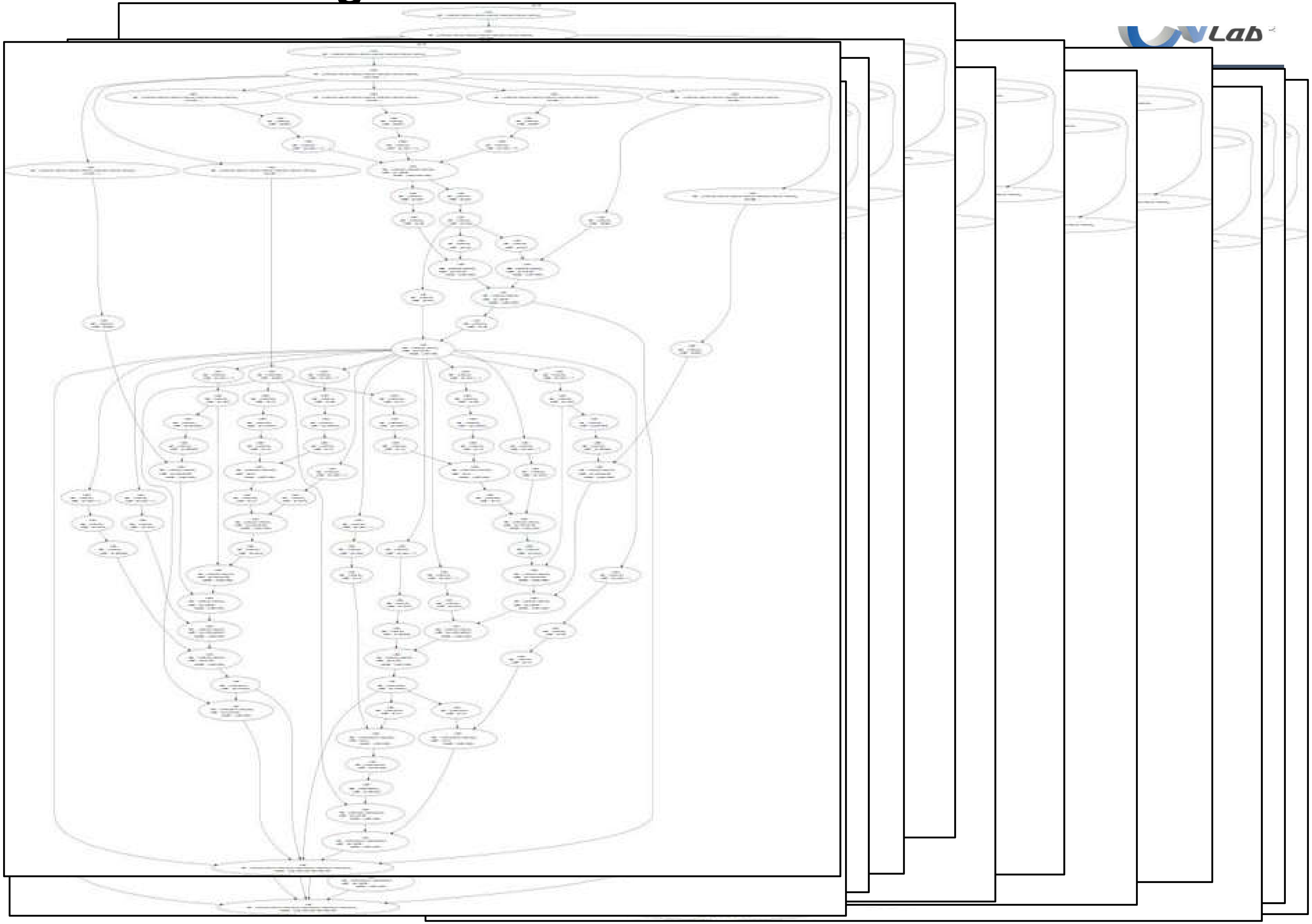


loss

A paper from deepmind

Slide from CS231n

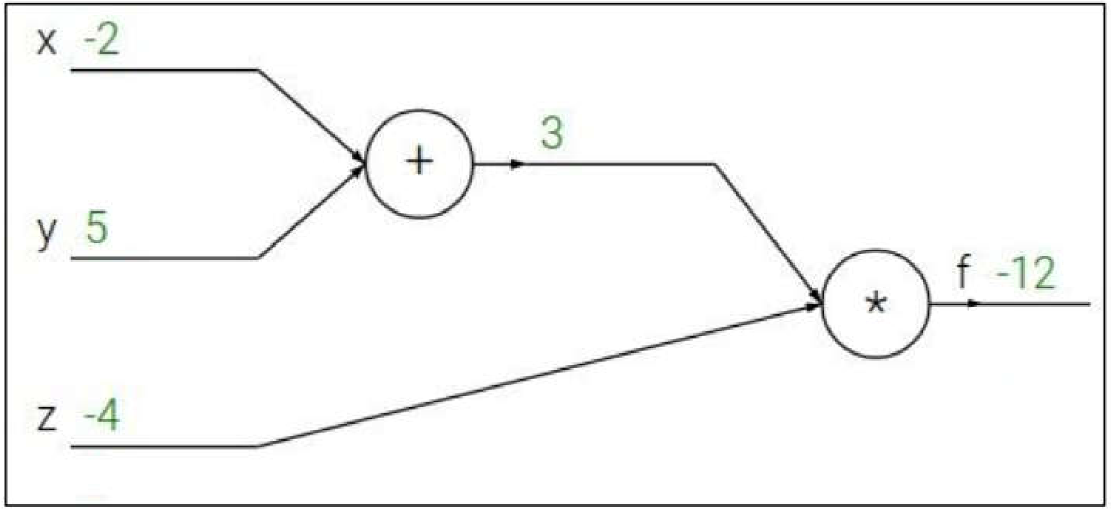
Neural Turing Machine



Start from Simple

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

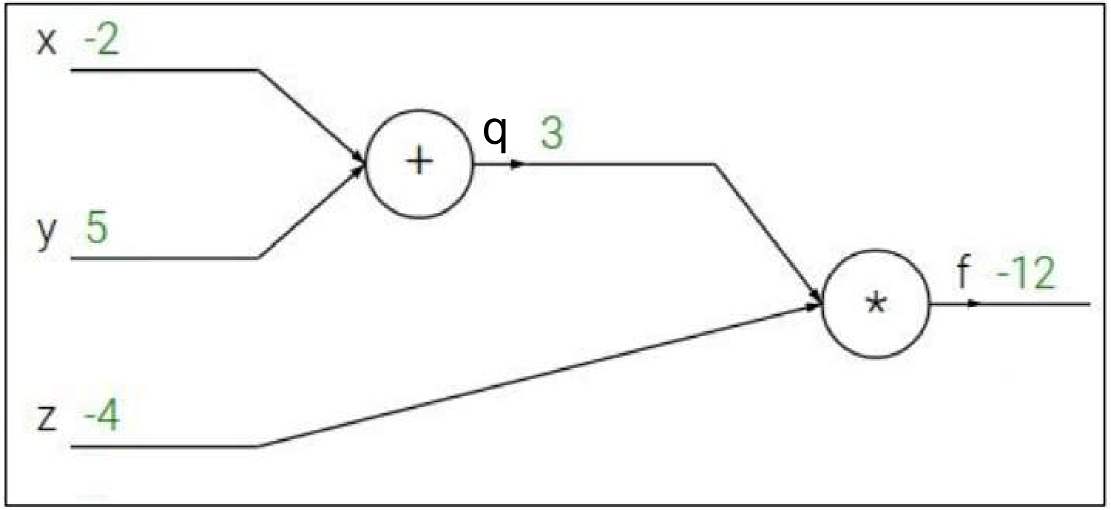
e.g. x = -2, y=5, z = -4



Start from Simple

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

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



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

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

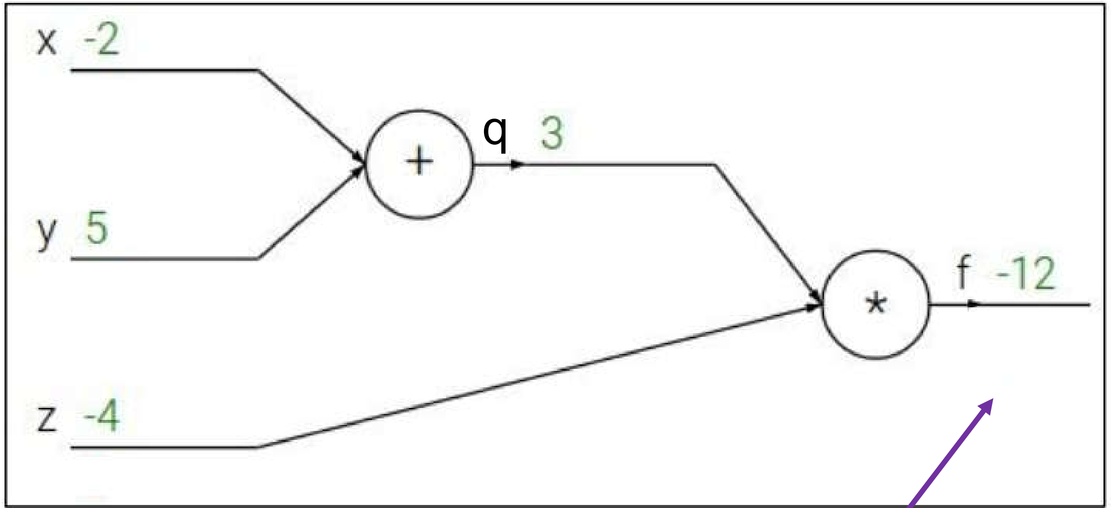
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Slide from CS231n

Start from Simple

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

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



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

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

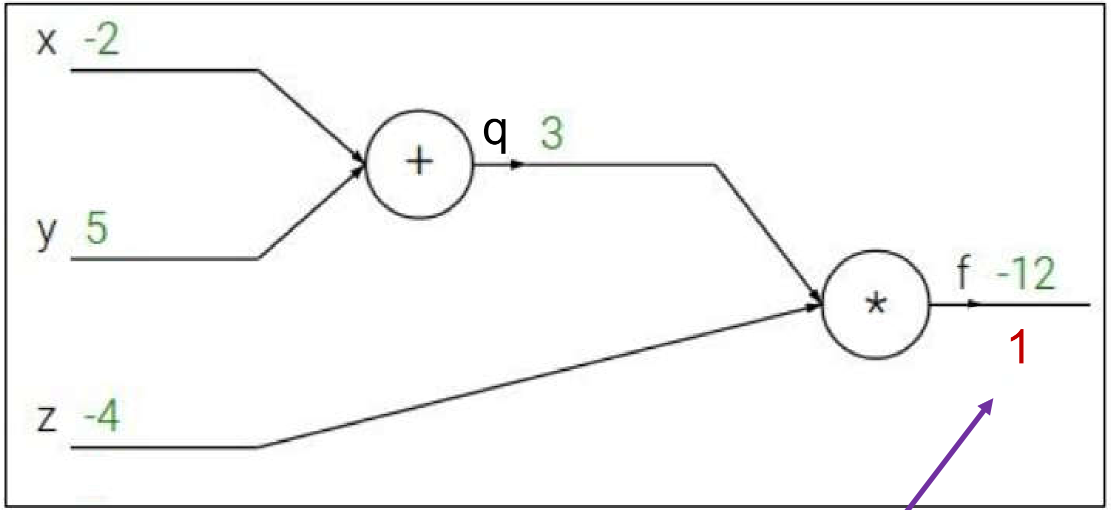
$$\boxed{\frac{\partial f}{\partial f}}$$

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

Start from Simple

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

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



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

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

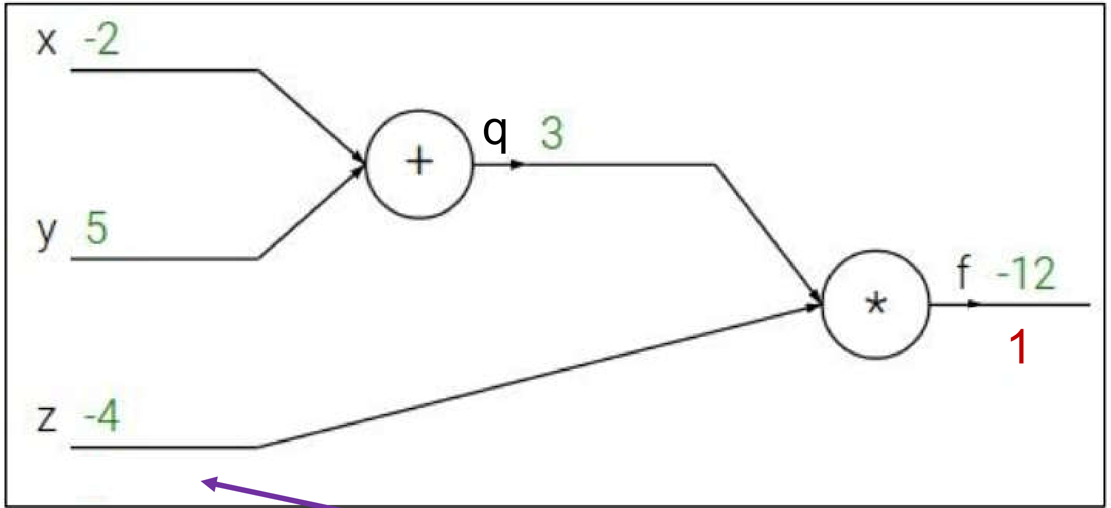
$$\frac{\partial f}{\partial f}$$

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

Start from Simple

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

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



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

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

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

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

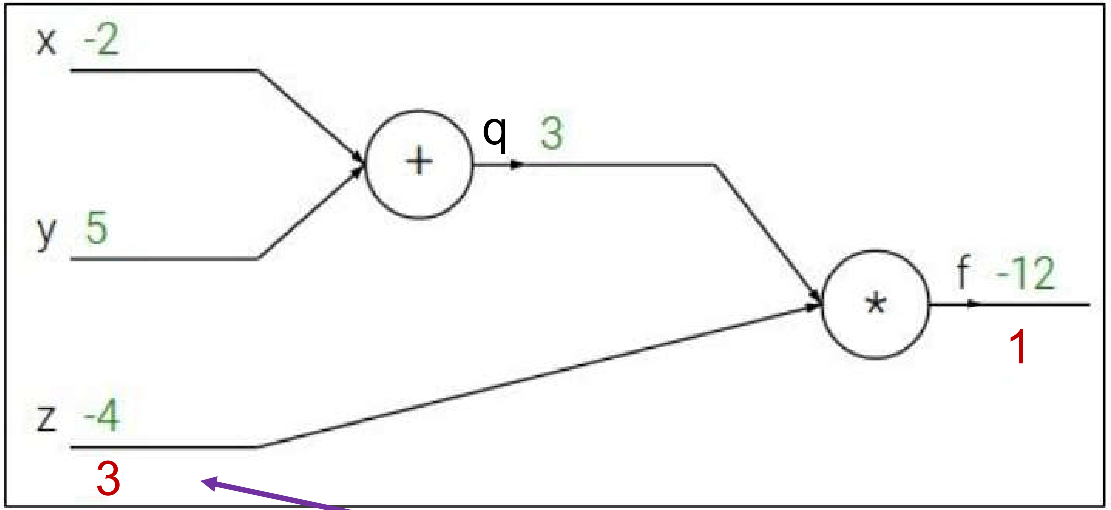
Start from Simple

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

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

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

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



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

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

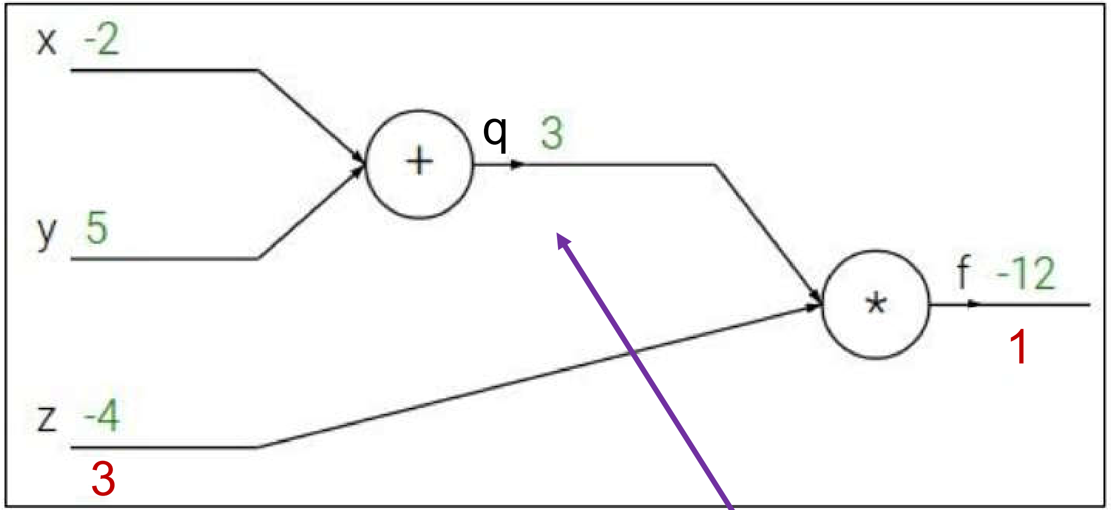
Start from Simple

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

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

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

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



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

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

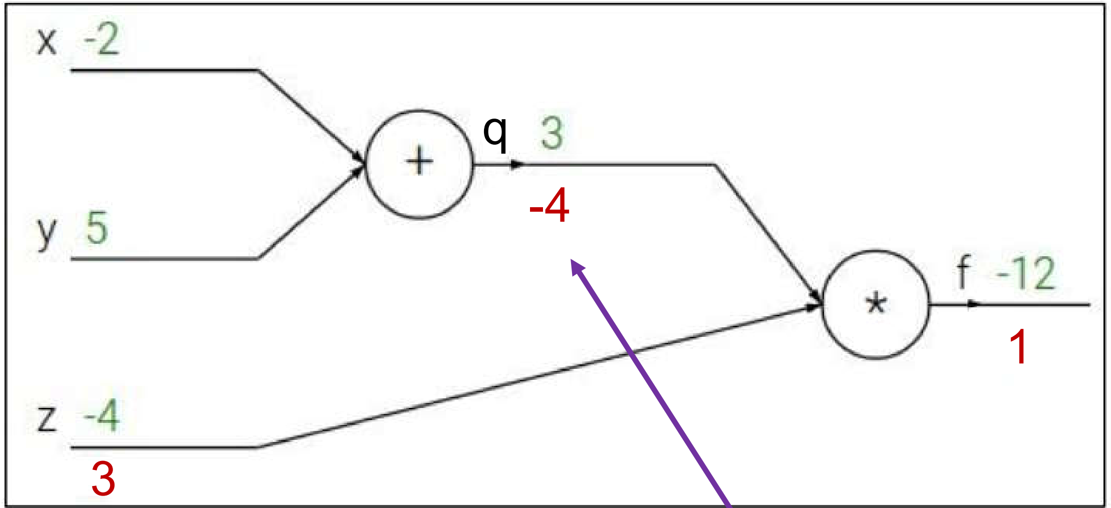
Start from Simple

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

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

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

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



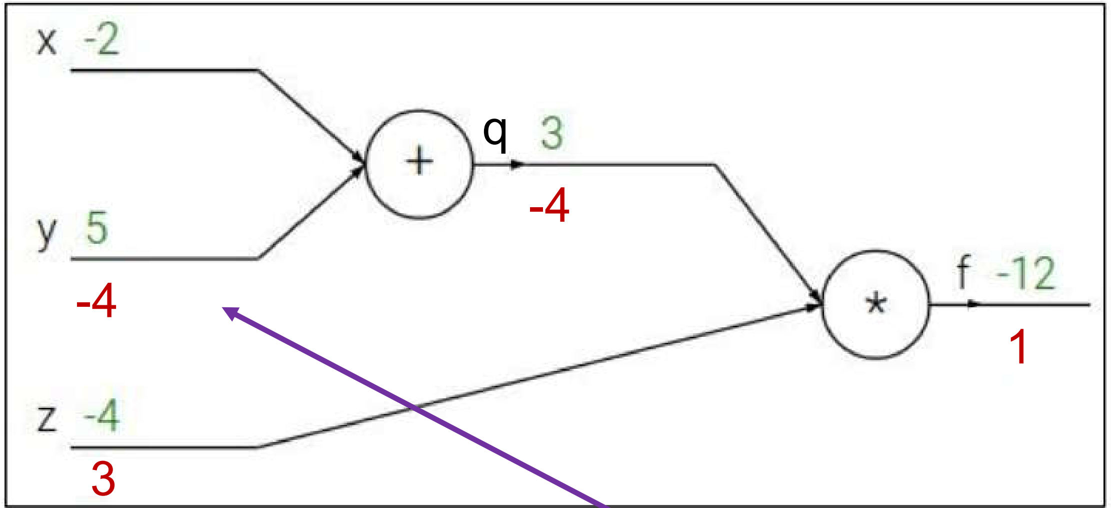
$$\frac{\partial f}{\partial q}$$

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

Start from Simple

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

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



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

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

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

Chain rule:

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

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

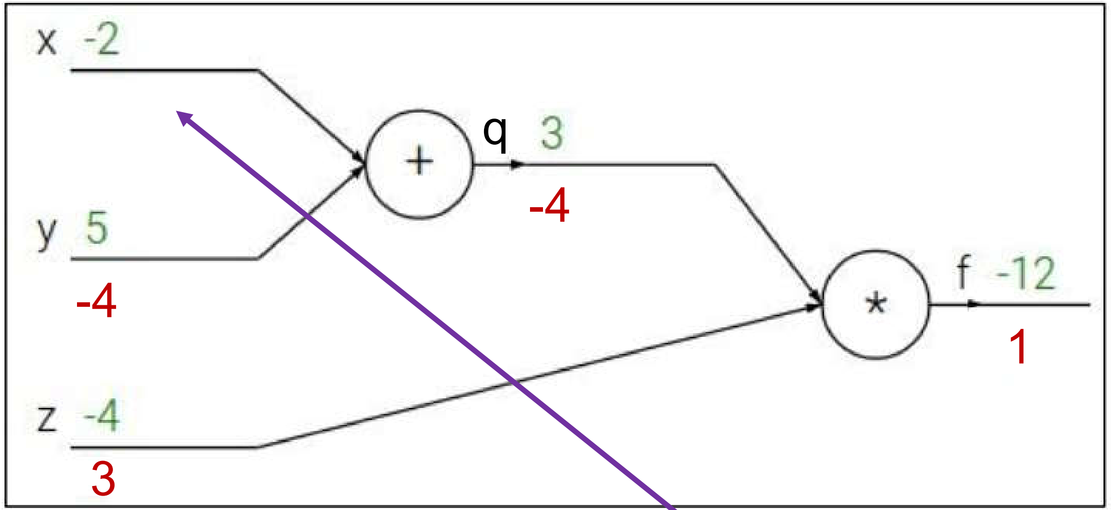
Start from Simple

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

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

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

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



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

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

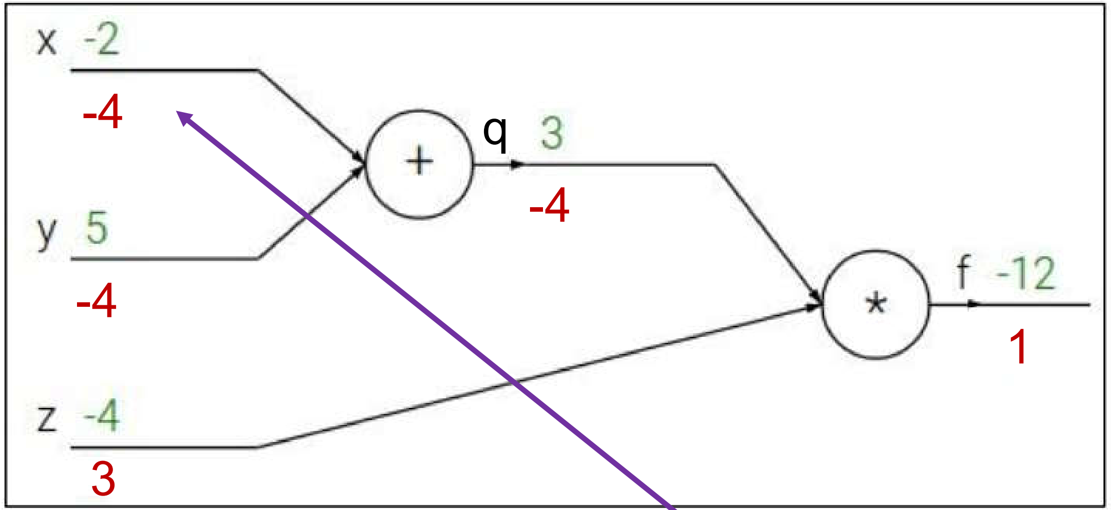
Start from Simple

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

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

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

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



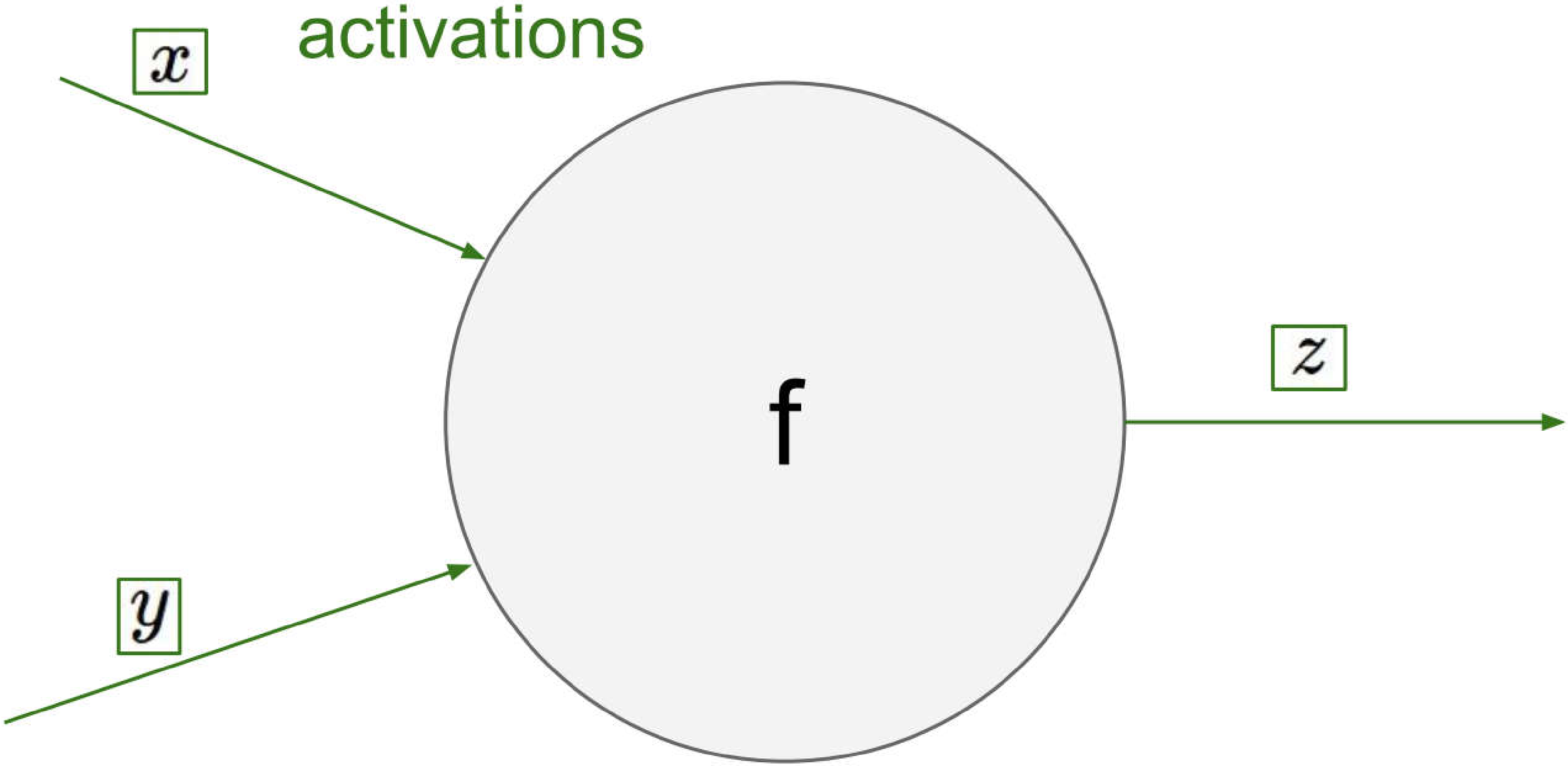
$$\frac{\partial f}{\partial x}$$

Chain rule:

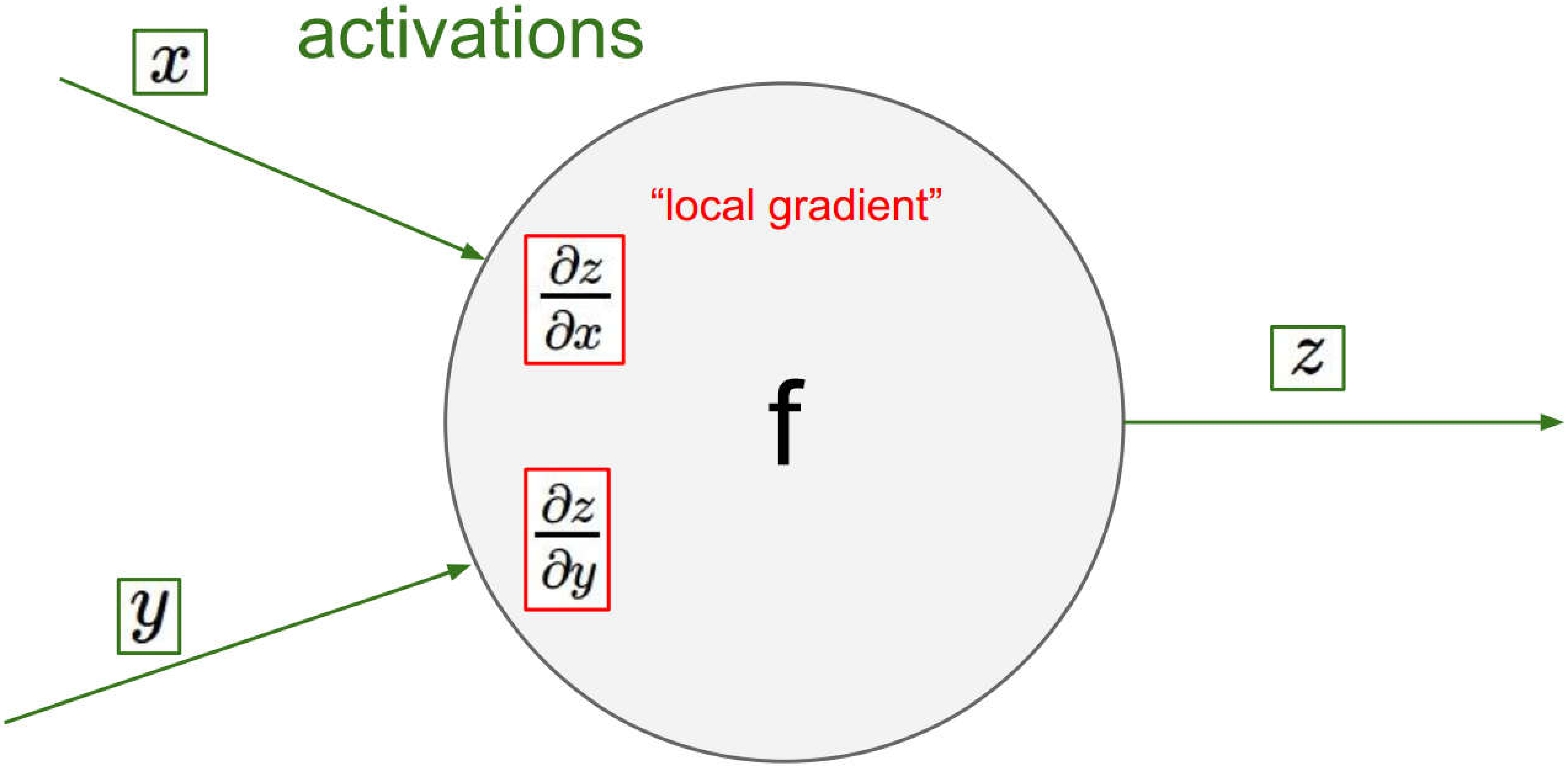
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

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

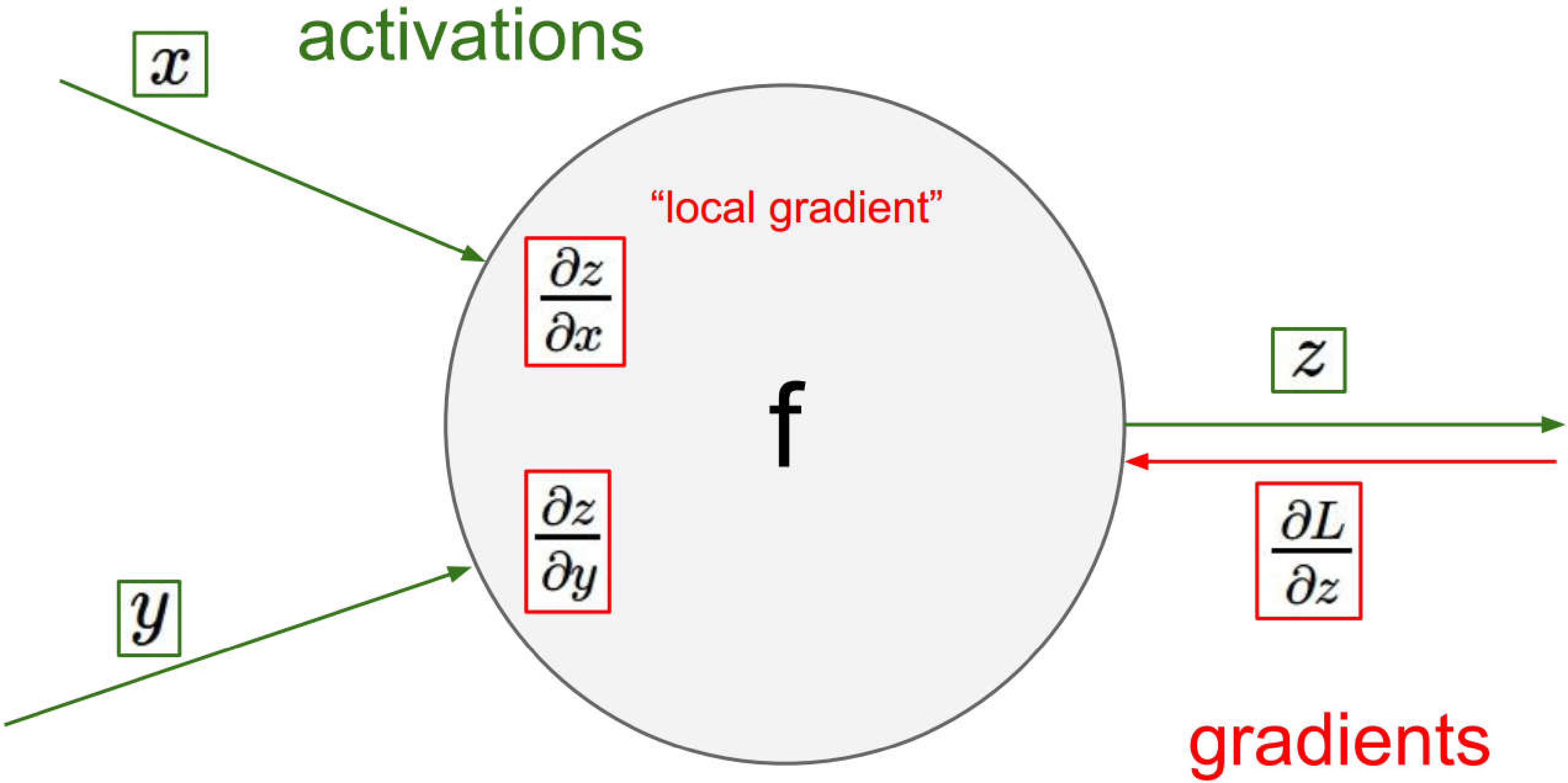
Backpropagation



Backpropagation

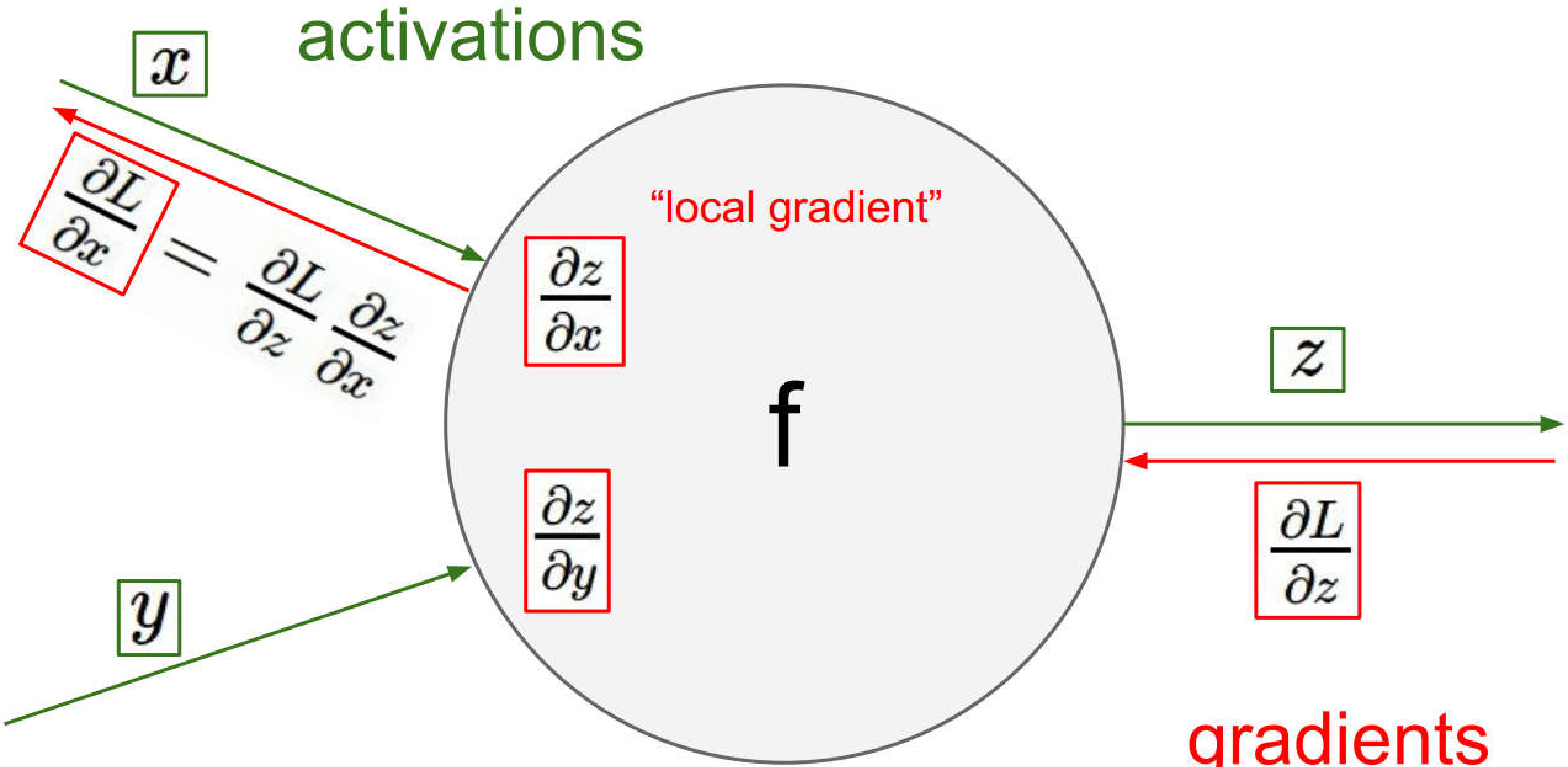


Backpropagation



Slide from CS231n

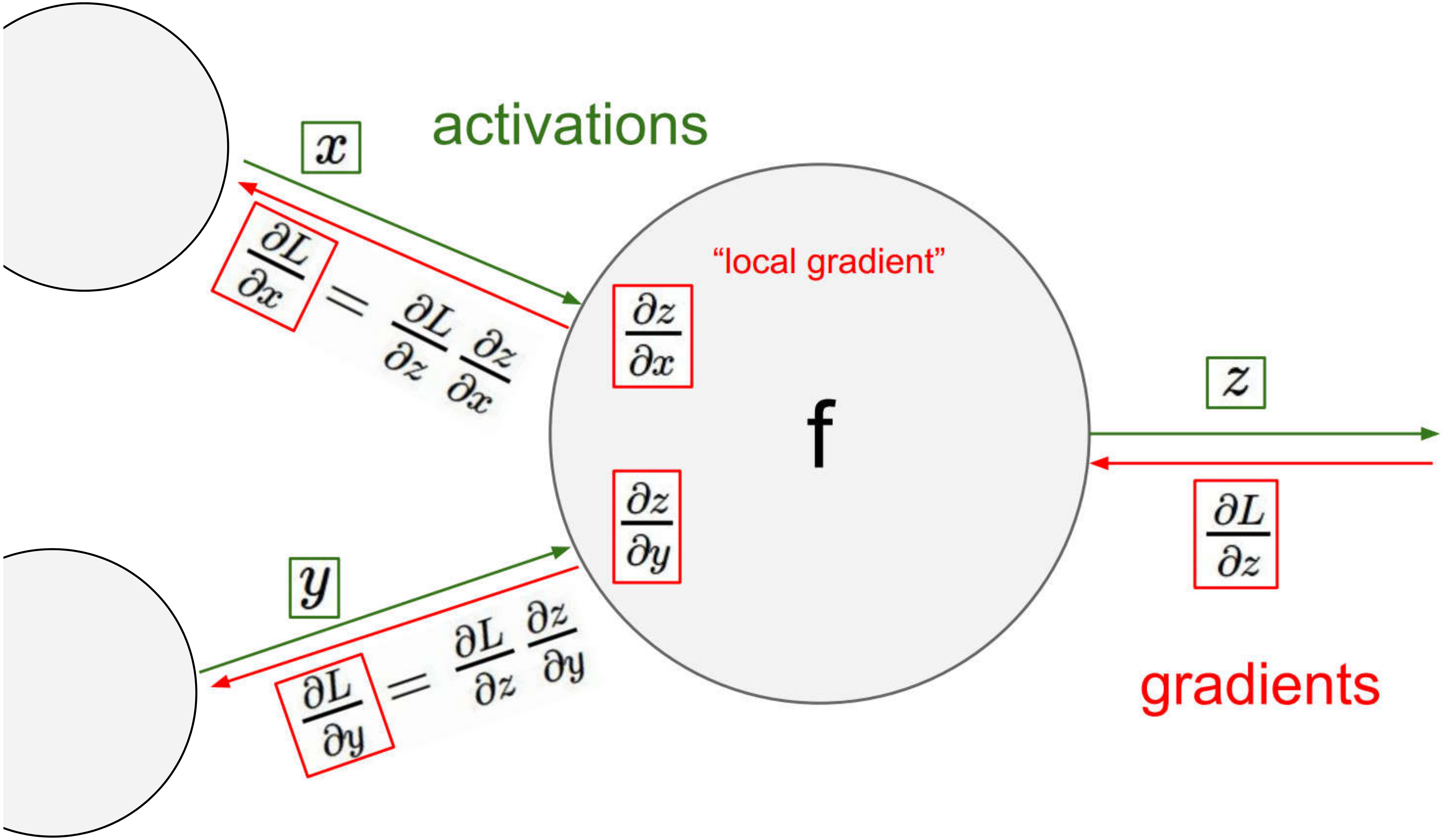
Backpropagation



Chain Rule

Slide from CS231n

Backpropagation

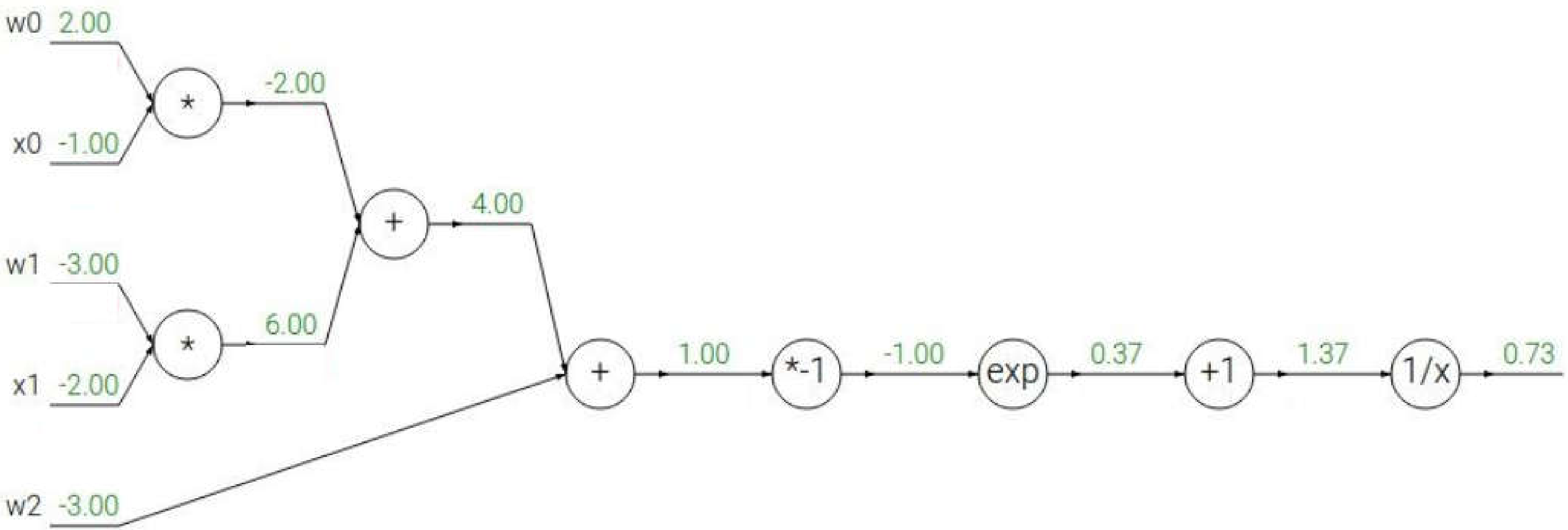


Chain rule

Slide from CS231n

Another Example

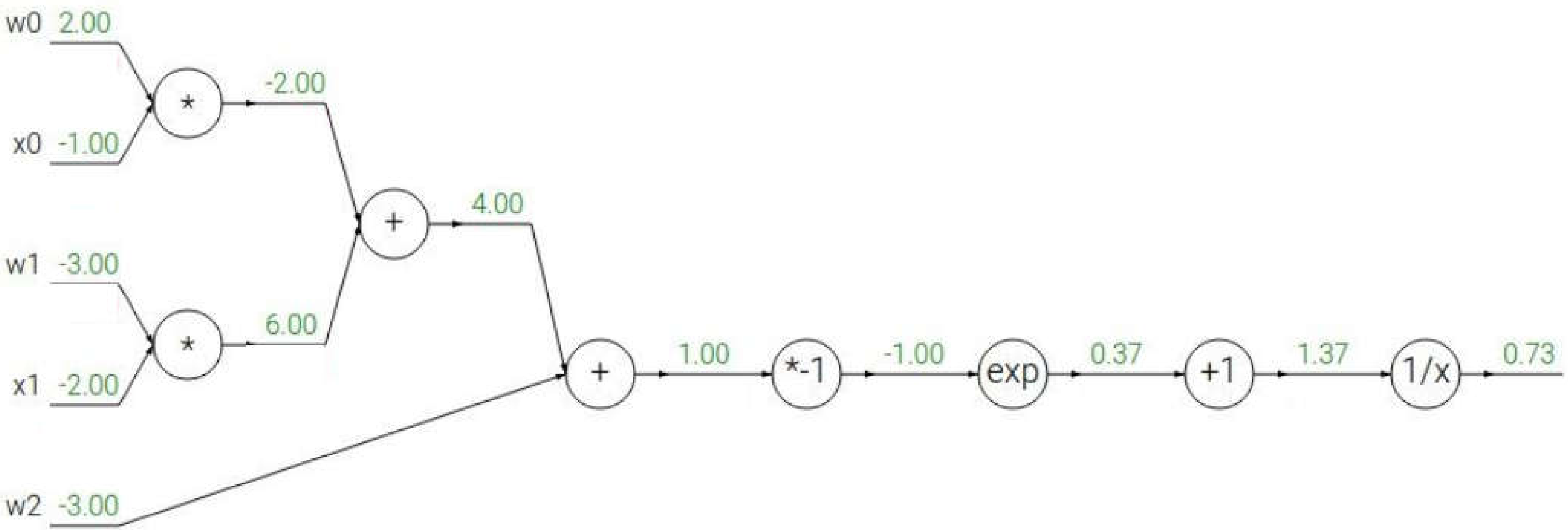
Sigmoid Neuron: $f(\boldsymbol{w}, \boldsymbol{x}) = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$



Computation graph of sigmoid neuron

Another Example

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

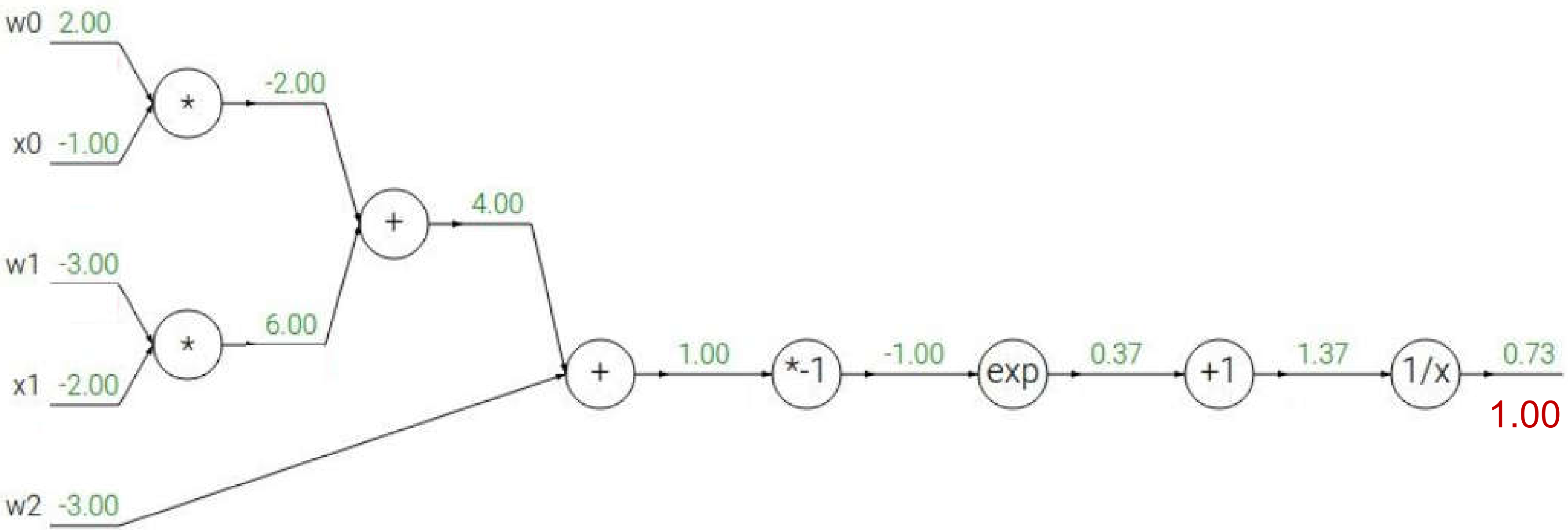


$f(x)=e^x$	$\rightarrow$	$\frac{df}{dx}=e^x$	$f(x)=\frac{1}{x}$	$\rightarrow$	$\frac{df}{dx}=-1/x^2$
$f_a(x)=ax$	$\rightarrow$	$\frac{df}{dx}=a$	$f_c(x)=c+x$	$\rightarrow$	$\frac{df}{dx}=1$

Slide from CS231n

Another Example

Sigmoid Neuron: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



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

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

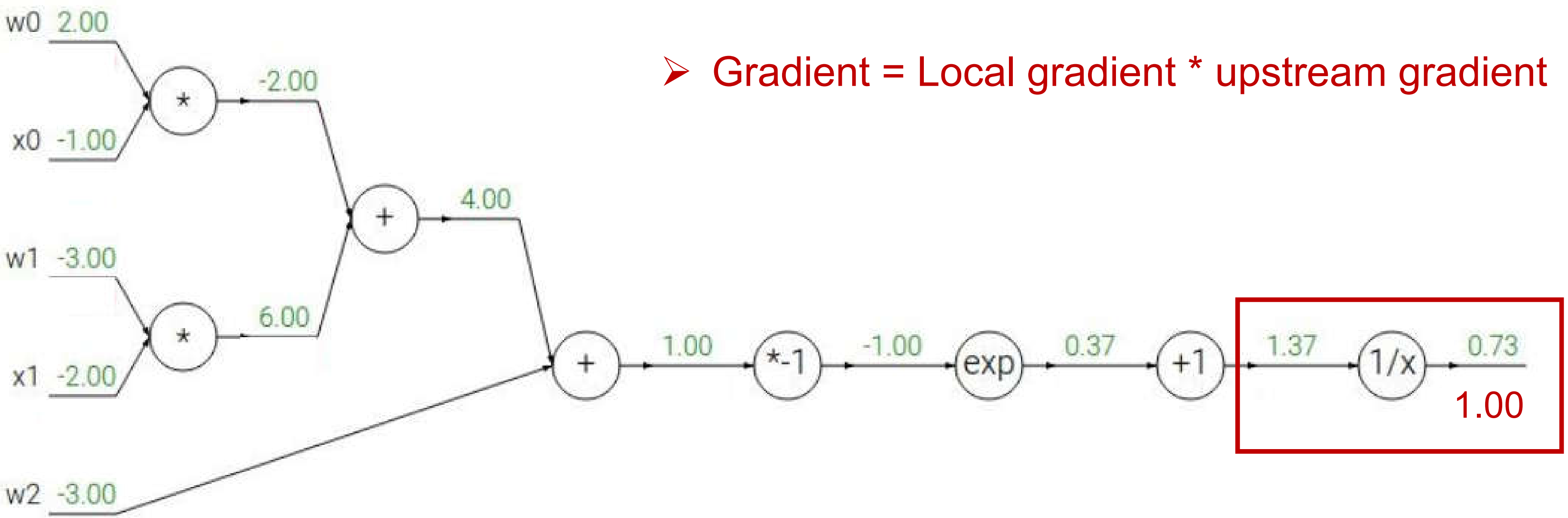
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Another Example

Sigmoid Neuron: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0 + w_1x_1 + w_2)}}$

➢ Gradient = Local gradient * upstream gradient

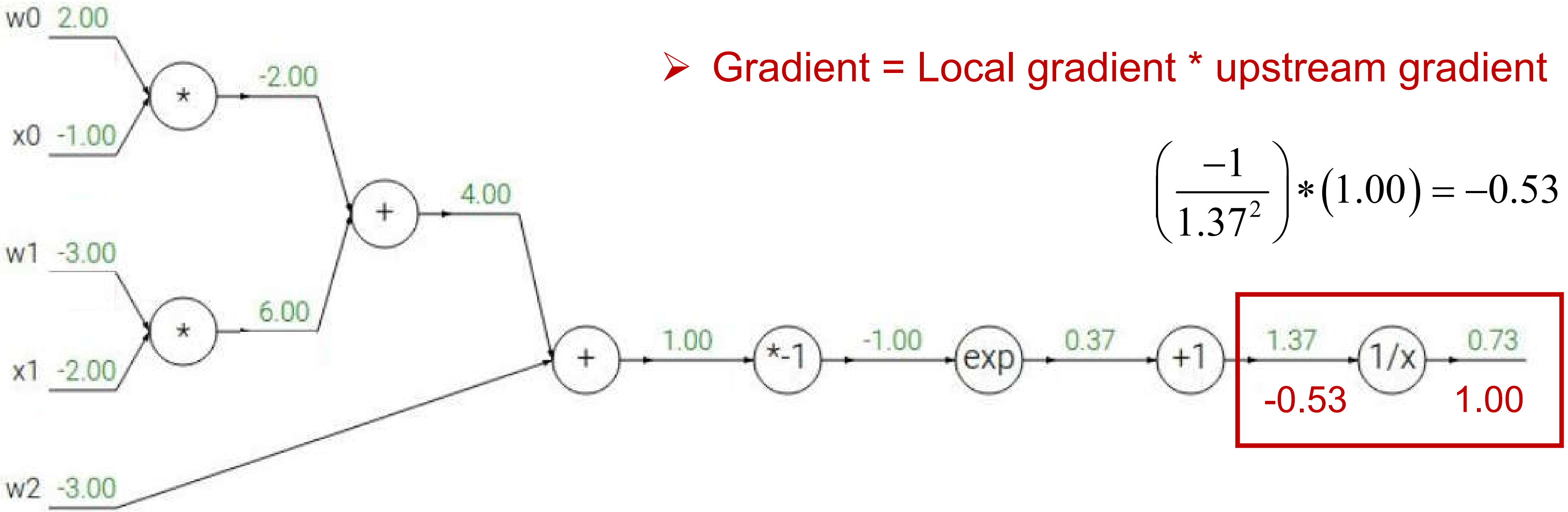


$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Another Example

Sigmoid Neuron: $f(\boldsymbol{w}, \boldsymbol{x}) = \dfrac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$



➢ Gradient = Local gradient * upstream gradient

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

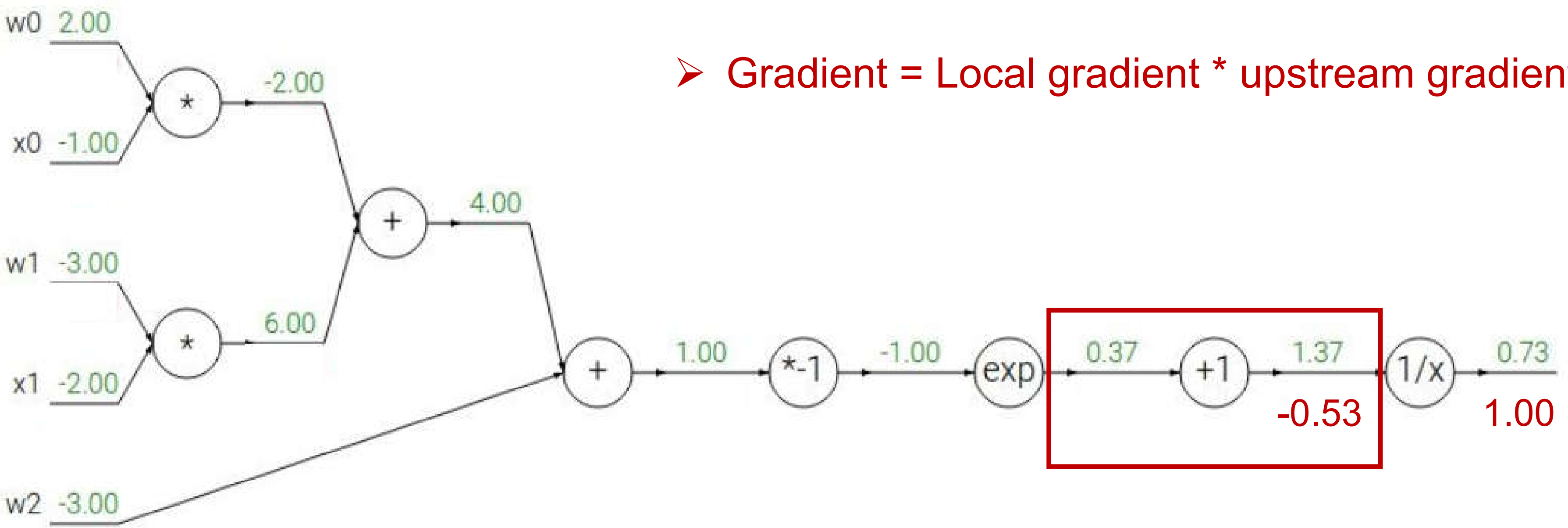
$f(x) = e^x \quad \rightarrow \quad \dfrac{df}{dx} = e^x$		$\boxed{f(x) = \dfrac{1}{x} \quad \rightarrow \quad \dfrac{df}{dx} = -1/x^2}$	
$f_a(x) = ax \quad \rightarrow \quad \dfrac{df}{dx} = a$		$f_c(x) = c + x \quad \rightarrow \quad \dfrac{df}{dx} = 1$	

Slide from CS231n

Another Example

Sigmoid Neuron: $f(w,x)=\dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

➢ Gradient = Local gradient * upstream gradient



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

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

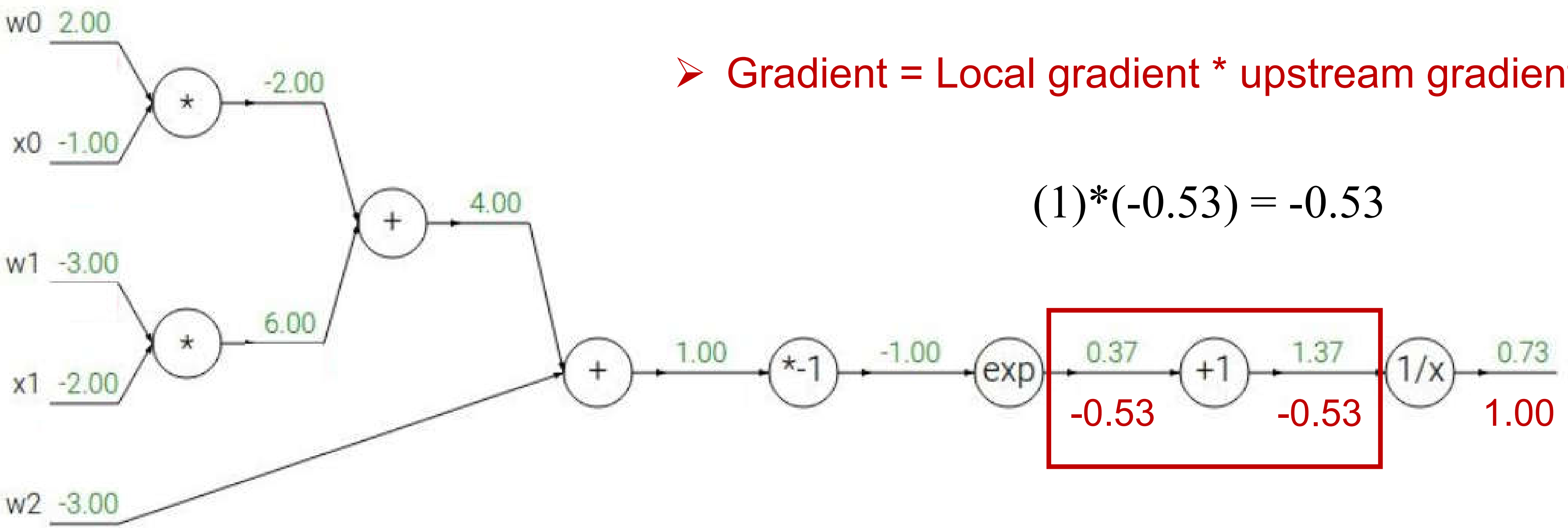
$f(x)=\dfrac{1}{x} \quad \rightarrow \quad \dfrac{df}{dx}=-1/x^2$

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

Slide from CS231n

Another Example

Sigmoid Neuron: $f(\boldsymbol{w}, \boldsymbol{x}) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



➢ Gradient = Local gradient * upstream gradient

(1)*(-0.53) = -0.53

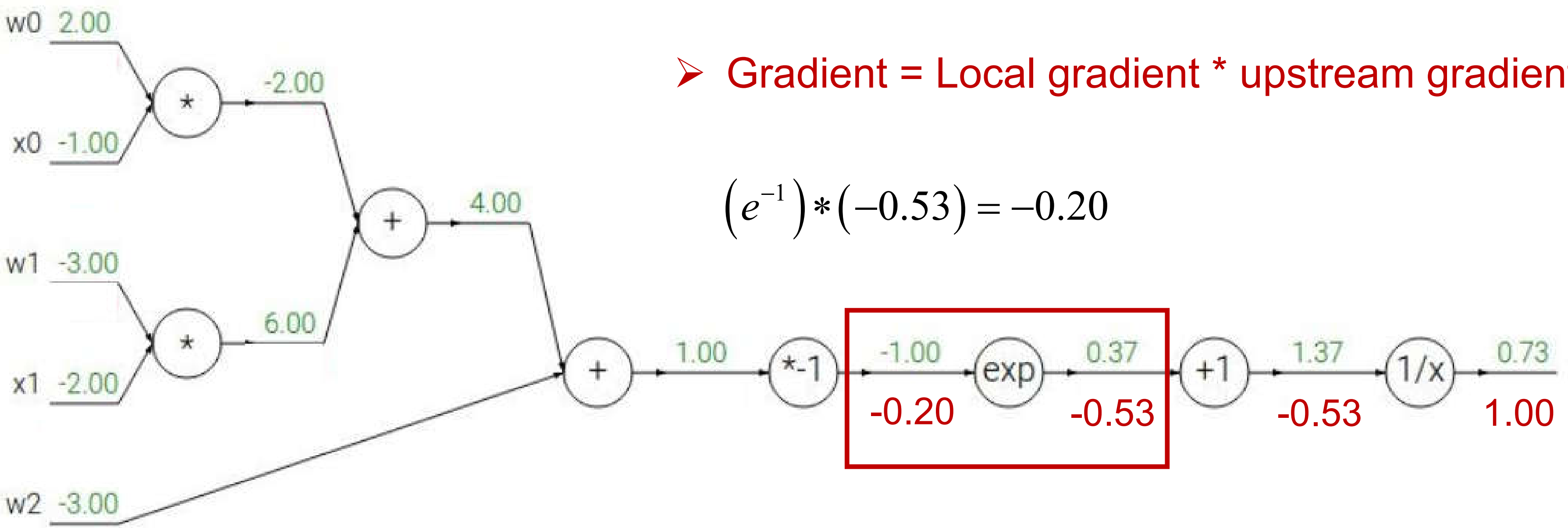
$f(x) = e^x \quad \rightarrow \quad \dfrac{df}{dx} = e^x$		$f(x) = \dfrac{1}{x} \quad \rightarrow \quad \dfrac{df}{dx} = -1/x^2$	
$f_a(x) = ax \quad \rightarrow \quad \dfrac{df}{dx} = a$		$f_c(x) = c + x \quad \rightarrow \quad \dfrac{df}{dx} = 1$	

Another Example

Sigmoid Neuron: $f(w,x) = \dfrac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

➢ Gradient = Local gradient * upstream gradient

$$\left(e^{-1}\right) * (-0.53) = -0.20$$



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

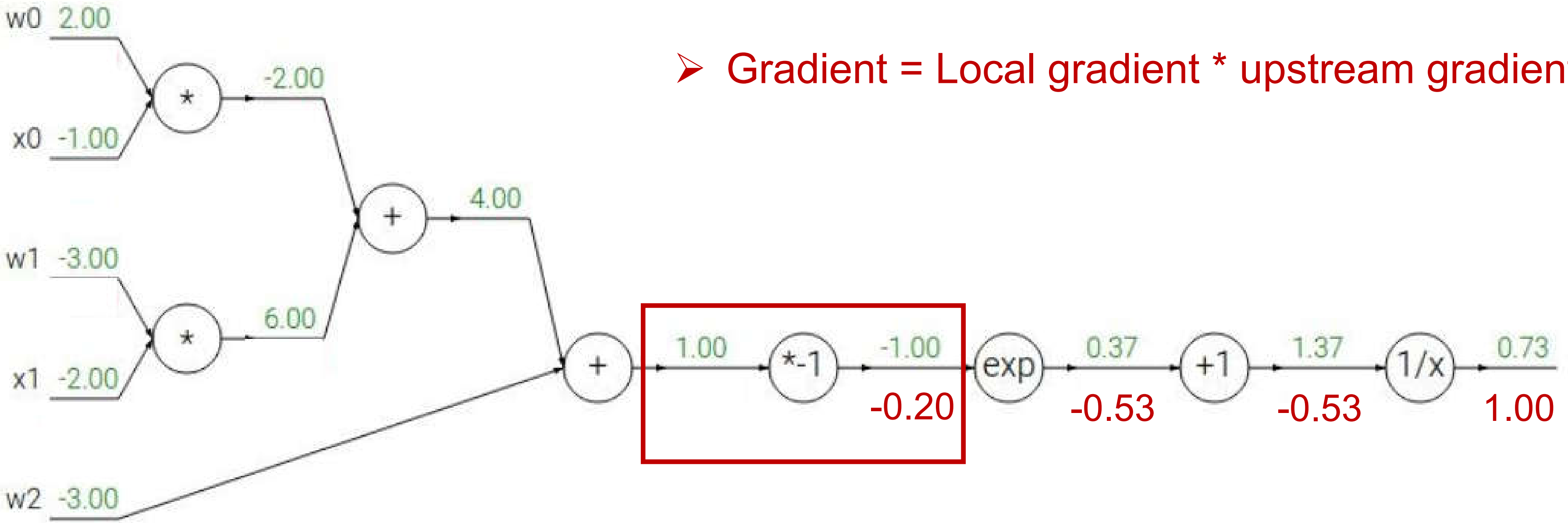
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another Example

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



➢ Gradient = Local gradient * upstream gradient

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

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

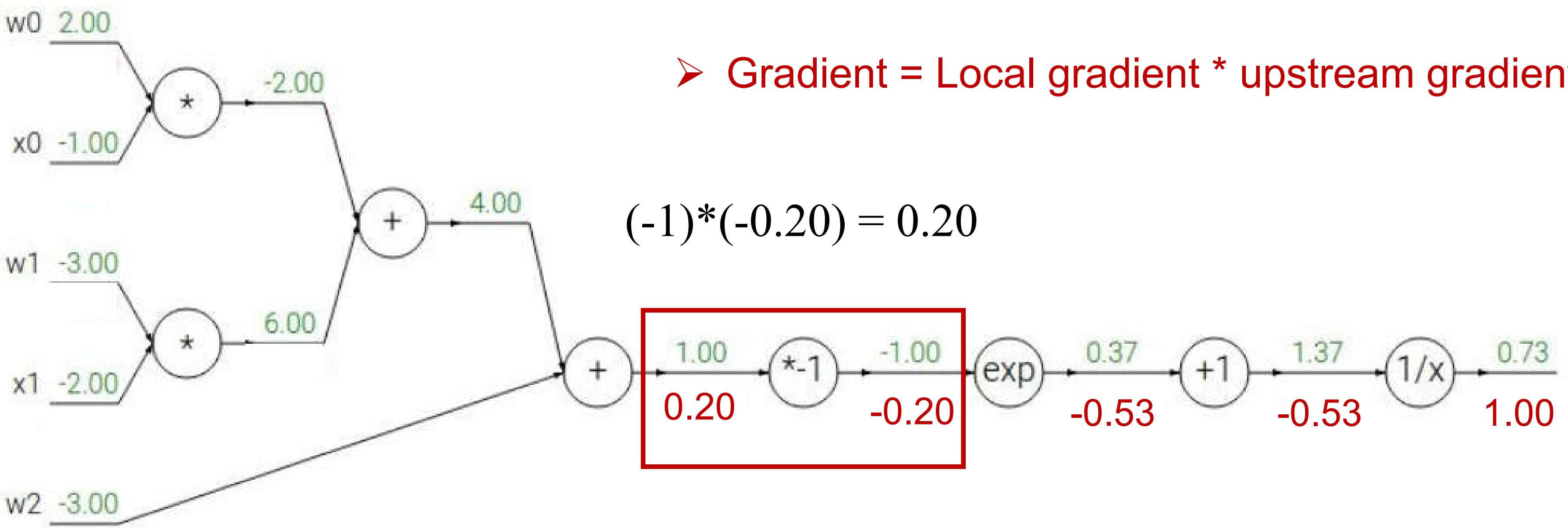
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

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

Another Example

Sigmoid Neuron: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0 + w_1x_1 + w_2)}}$

- Gradient = Local gradient * upstream gradient



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

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

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

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

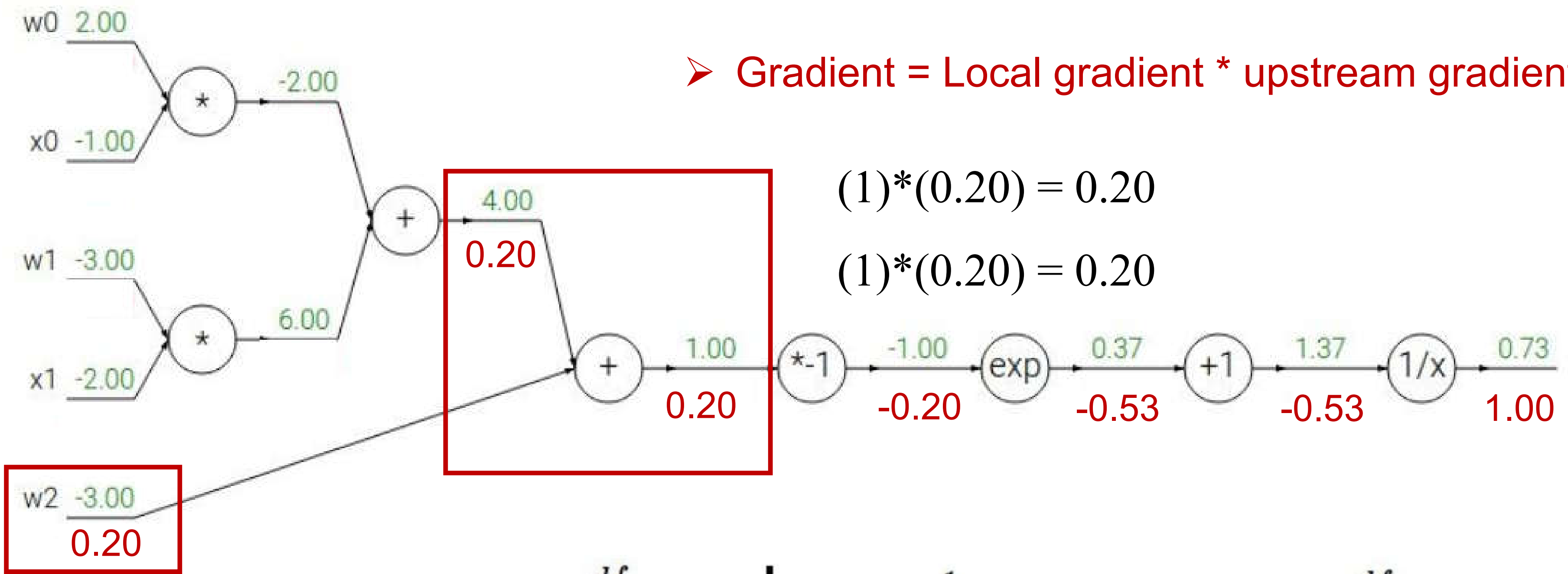
Another Example

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

- Gradient = Local gradient * upstream gradient

$$(1)*(0.20) = 0.20$$

$$(1)*(0.20) = 0.20$$



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

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

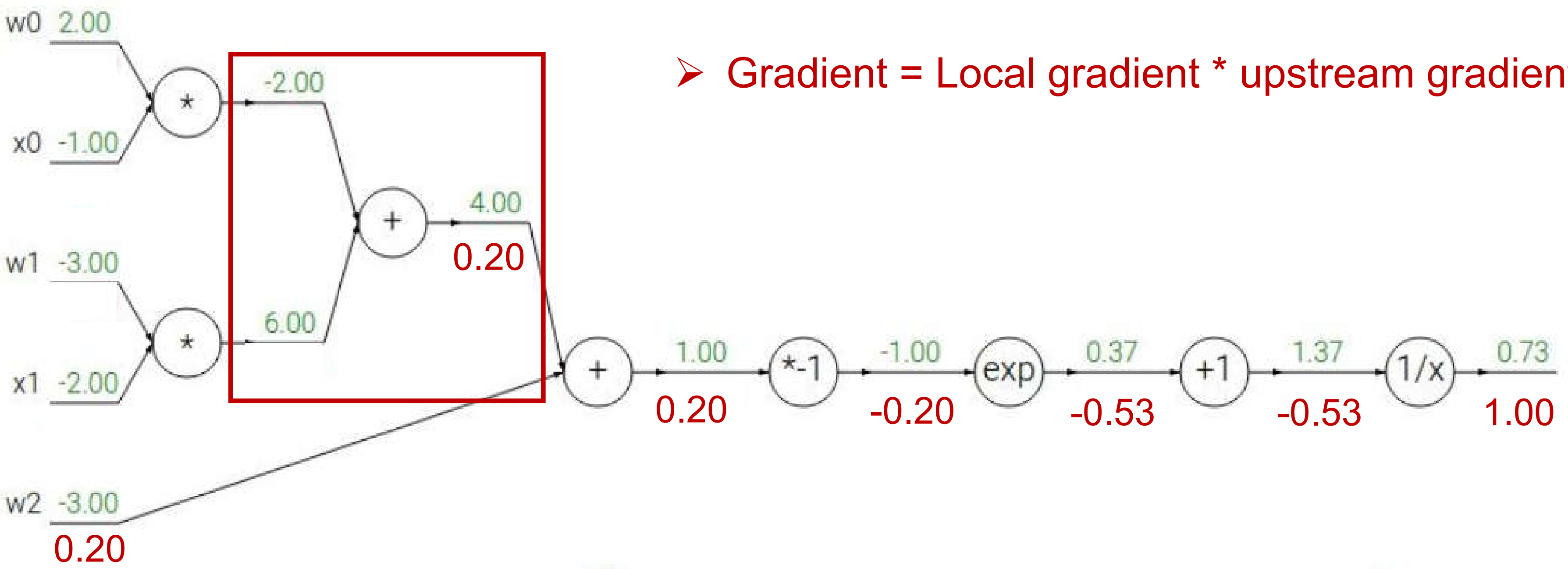
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

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

Another Example

Sigmoid Neuron: $f(\boldsymbol{w},\boldsymbol{x}) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

- Gradient = Local gradient * upstream gradient



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

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

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

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

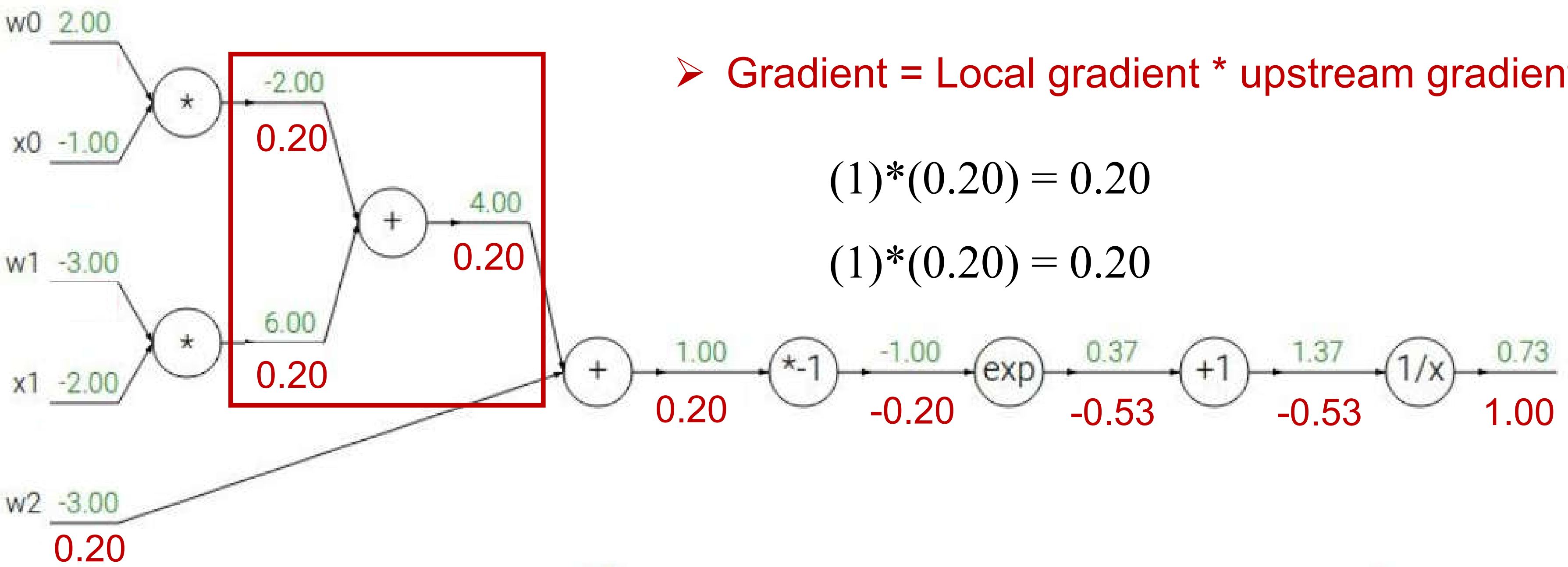
Another Example

Sigmoid Neuron: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

- Gradient = Local gradient * upstream gradient

$(1)*(0.20) = 0.20$

$(1)*(0.20) = 0.20$



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

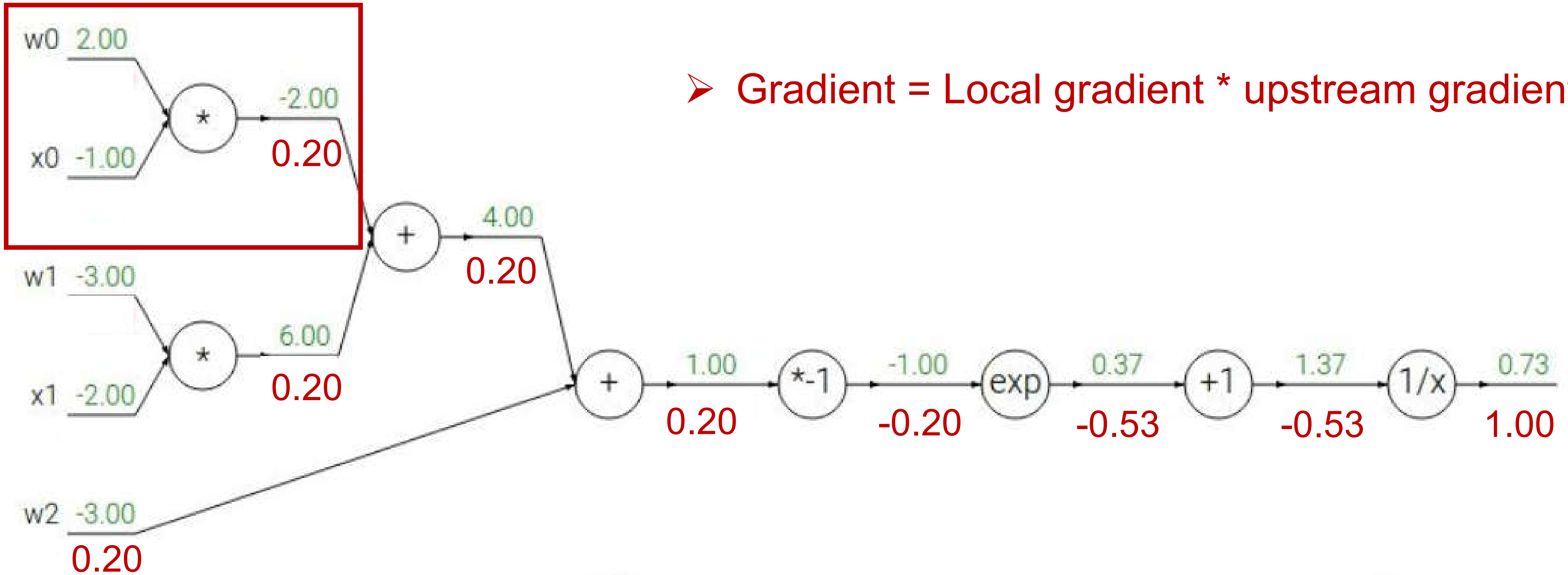
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another Example

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



- Gradient = Local gradient * upstream gradient

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

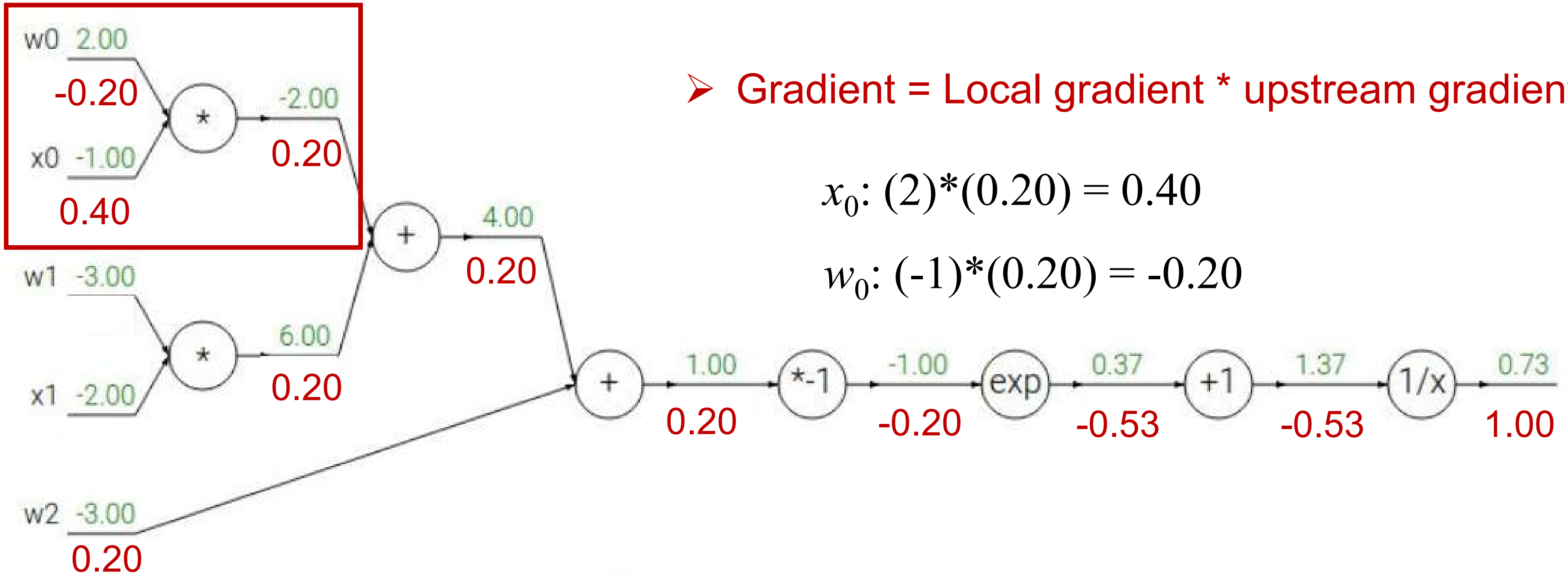
$$\boxed{f_a(x)=ax \quad \rightarrow \quad \frac{df}{dx}=a}$$

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

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

Another Example

Sigmoid Neuron: $f(w,x)=\dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



- Gradient = Local gradient * upstream gradient

x_0: (2)*(0.20) = 0.40

w_0: (-1)*(0.20) = -0.20

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

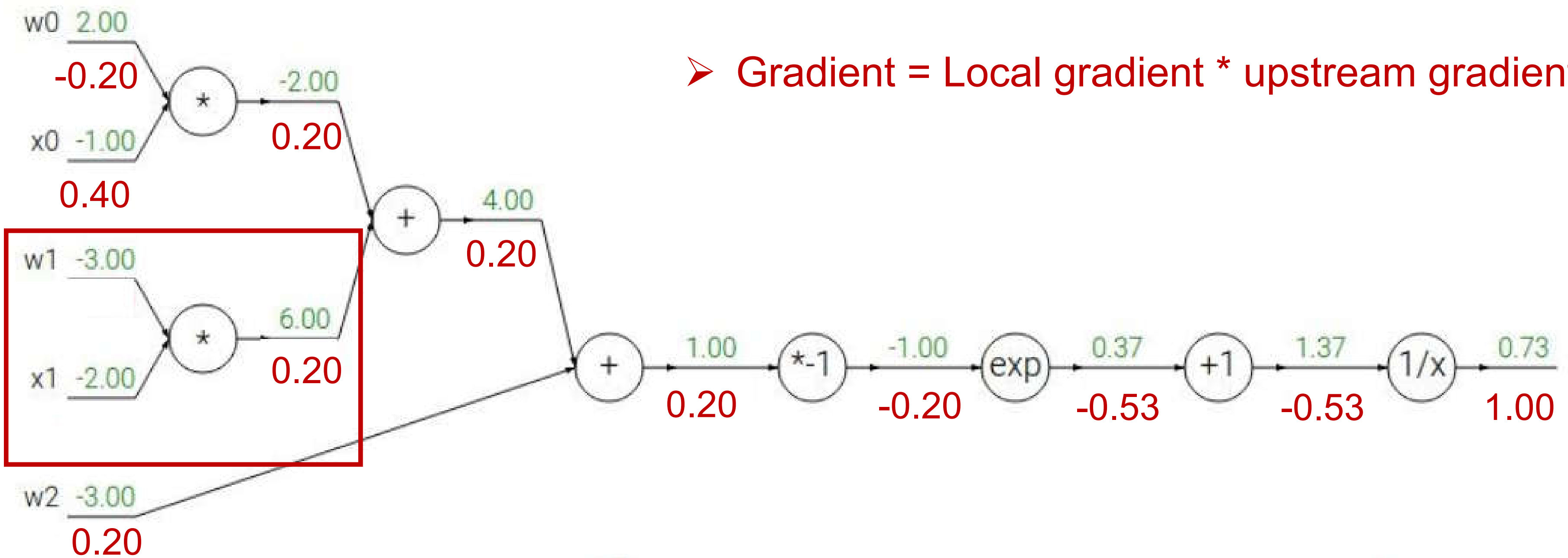
$\boxed{f_a(x)=ax \rightarrow \dfrac{df}{dx}=a}$

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

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

Another Example

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



- Gradient = Local gradient * upstream gradient

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

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

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

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

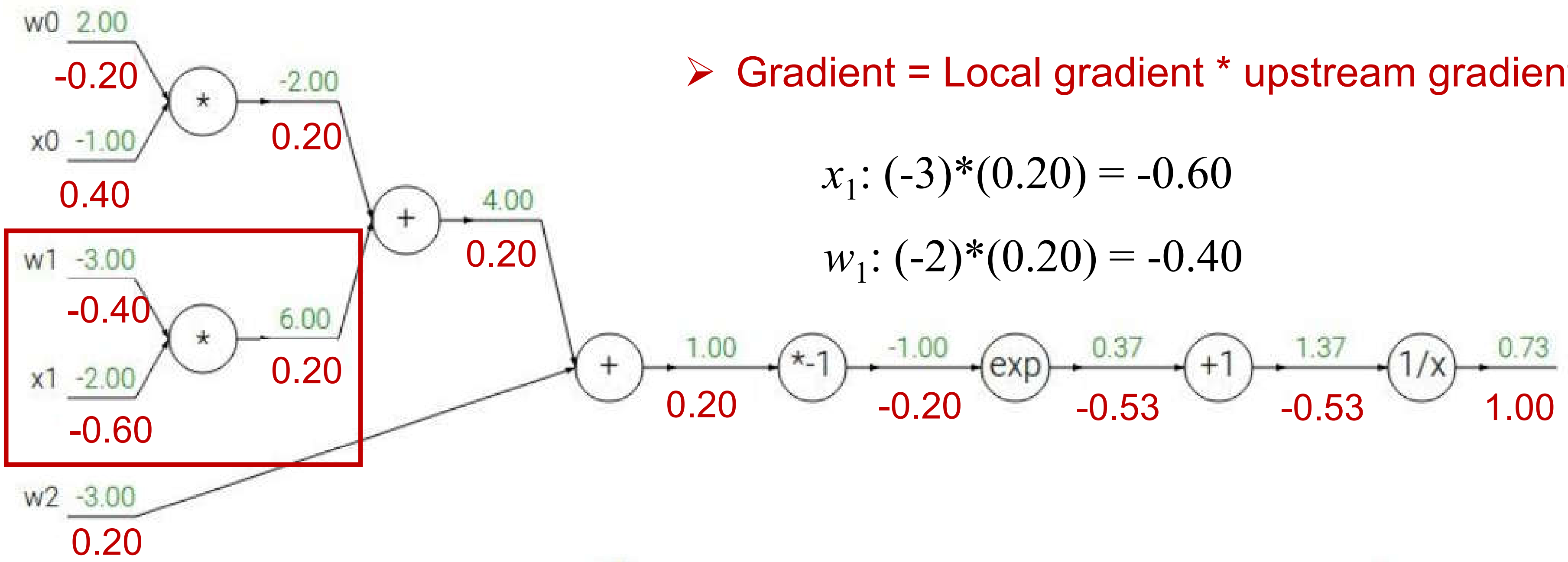
Another Example

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

- Gradient = Local gradient * upstream gradient

x_1: (-3)*(0.20) = -0.60

w_1: (-2)*(0.20) = -0.40



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

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

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

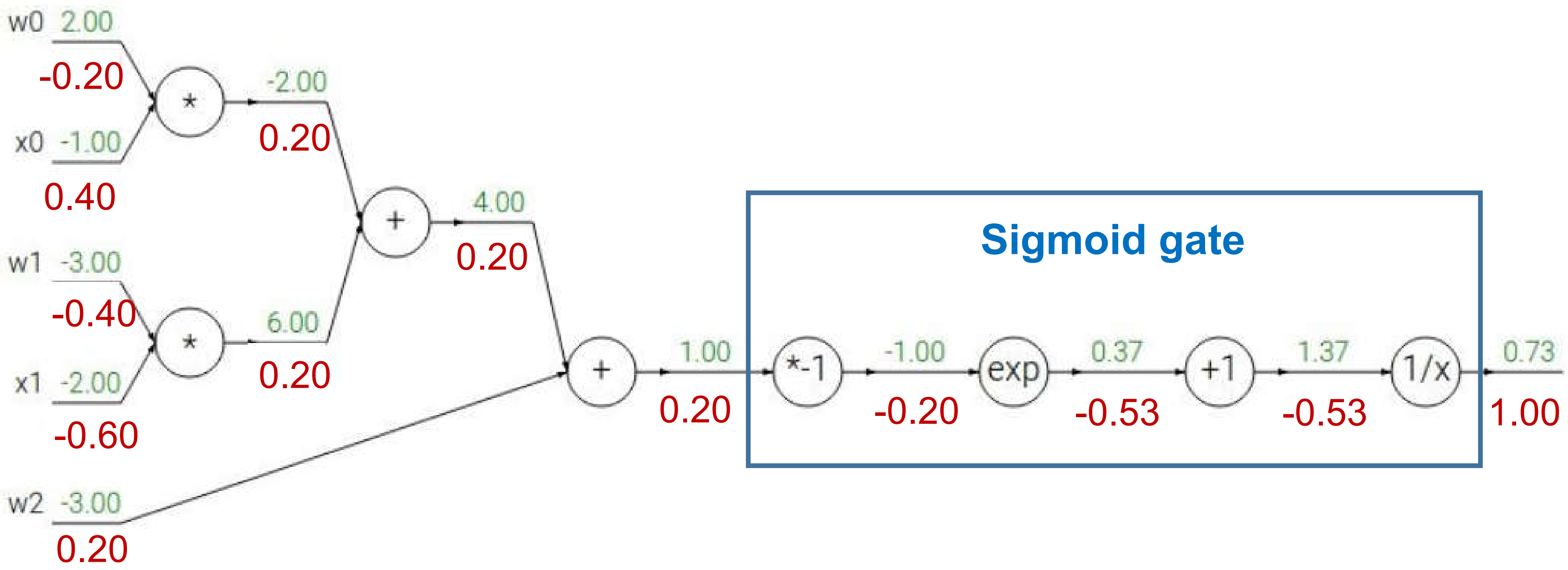
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Sigmoid Function

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

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

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



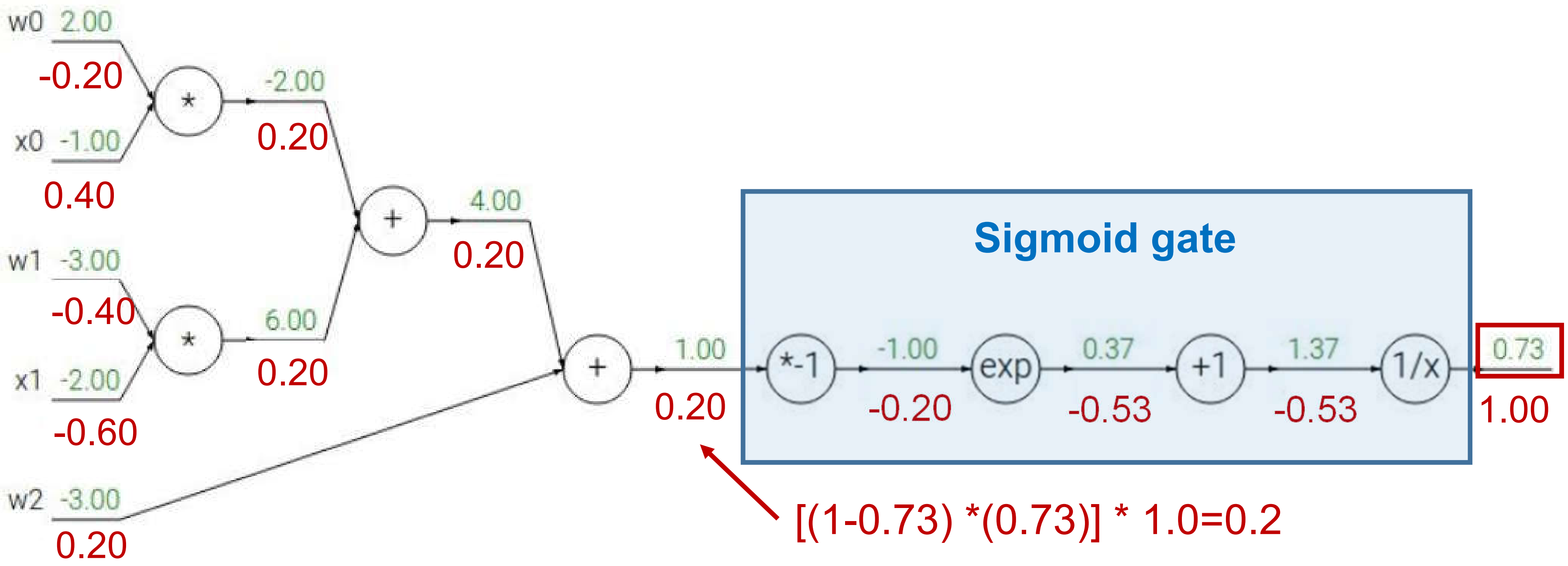
Slide from CS231n

Sigmoid Function

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

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

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



Slide from CS231n

Patterns in Backward Flow

add gate: gradient distributor

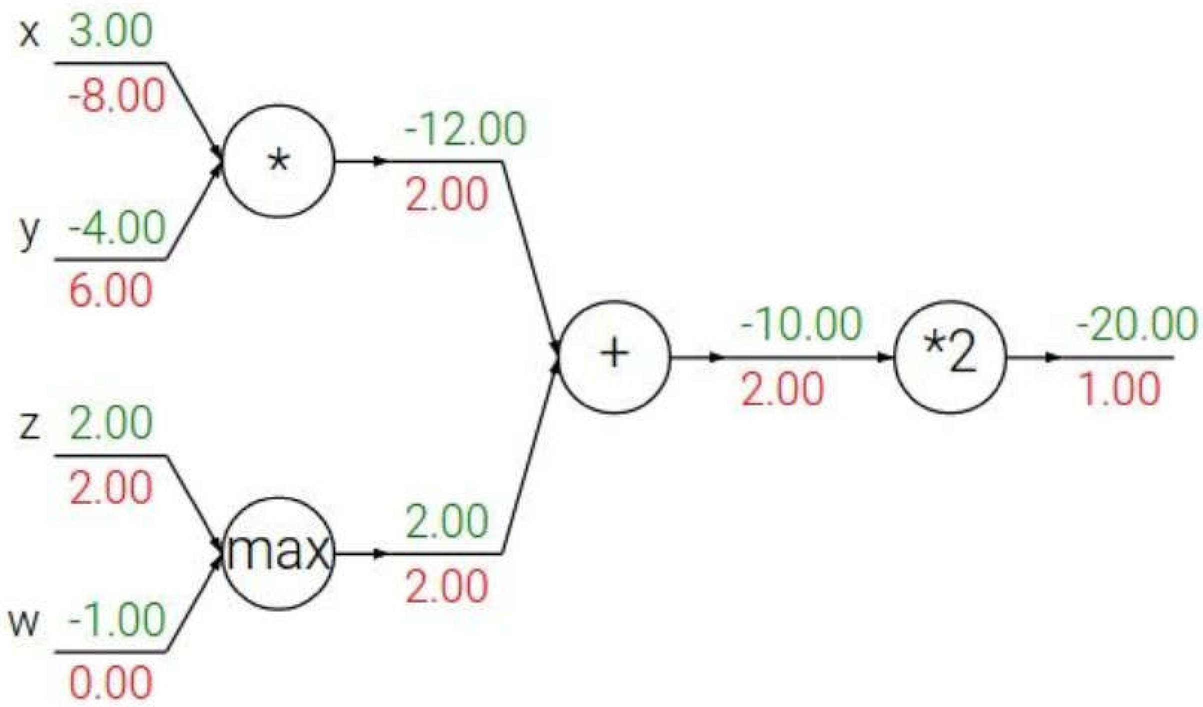
max gate: gradient router

mul gate: gradient “switcher”

max(-1, 2) = 2

max(-1+δ, 2) = 2

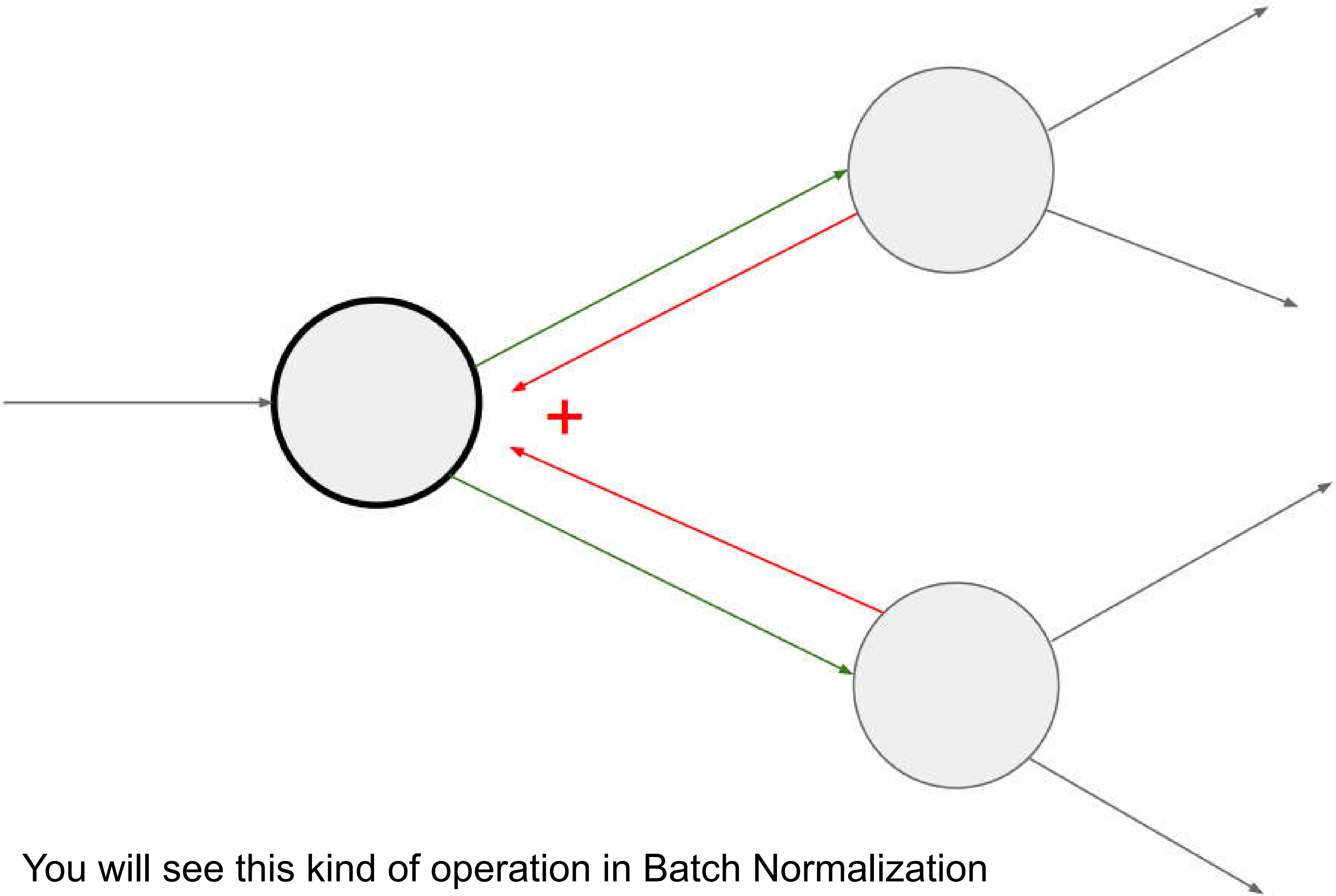
max(-1, 2+δ) = 2+δ



Question #1: what will happen in max gate when two inputs are the same?

Slide from CS231n

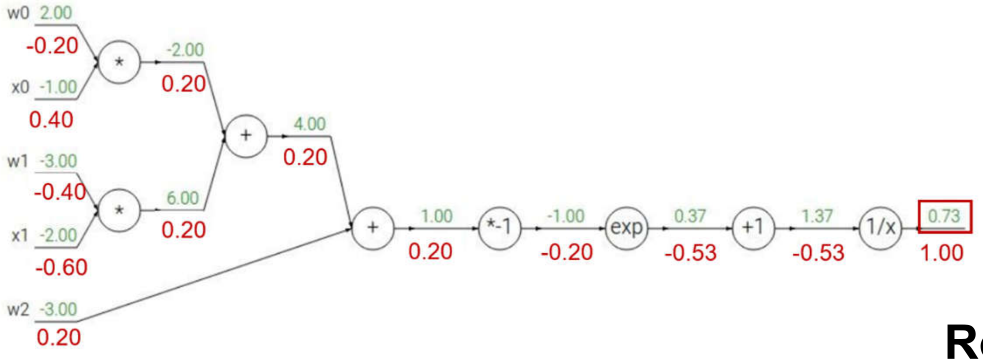
Gradients Add at Branches



- You will see this kind of operation in Batch Normalization

Forward/Backward API

Implementation: Forward/Backward API

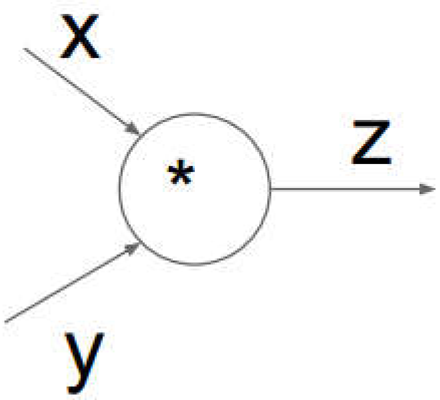


Rough pseudo code

```
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
```

Slide from CS231n

Implementation: Forward/Backward API




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

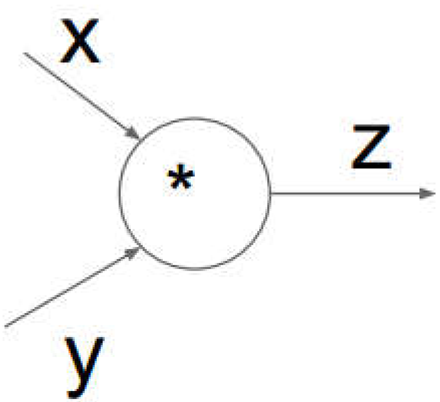
Chain rule

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

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

(x, y, z are scalars)

Implementation: Forward/Backward API



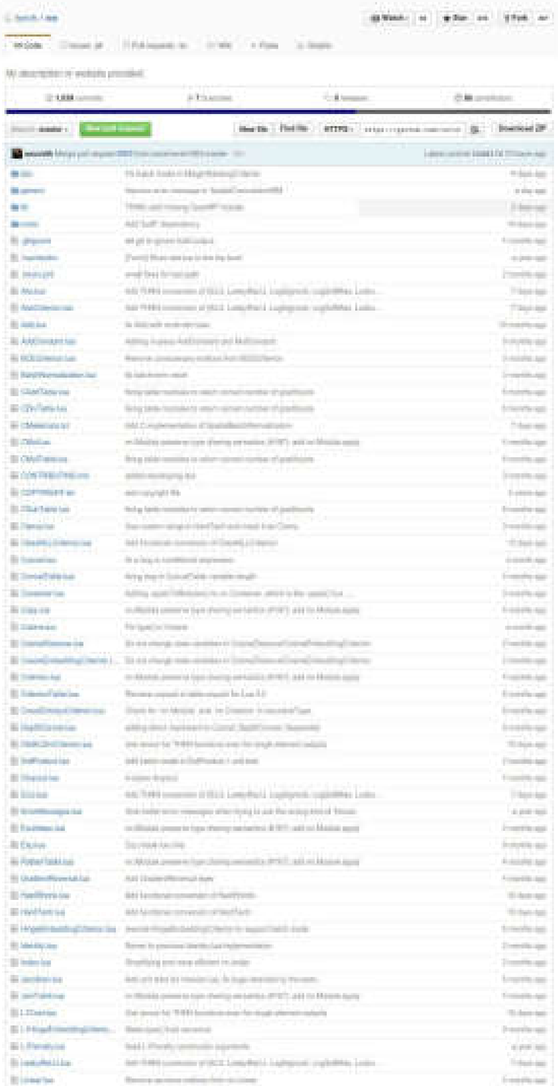

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

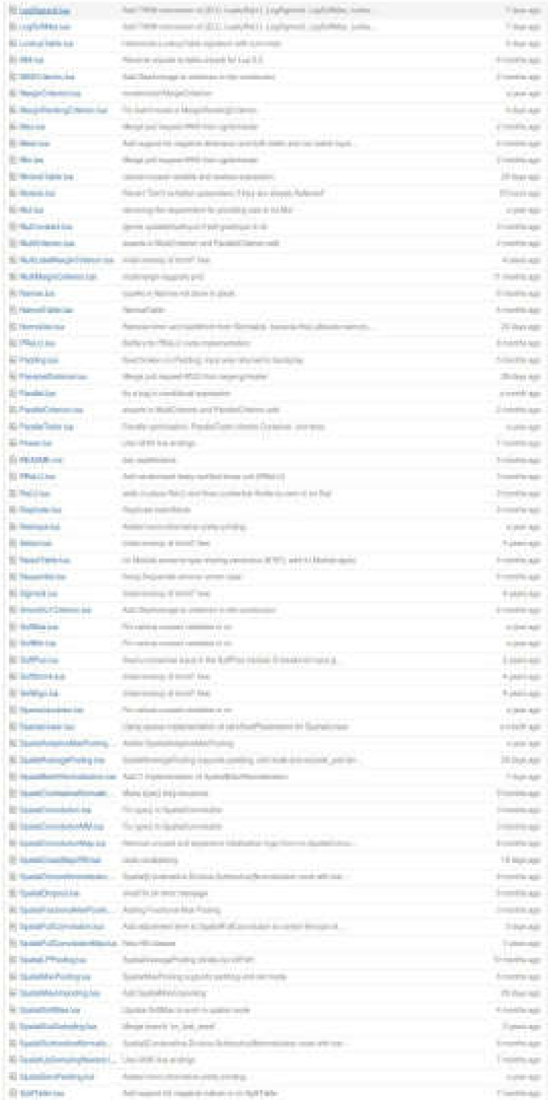
cache

(x, y, z are scalars)

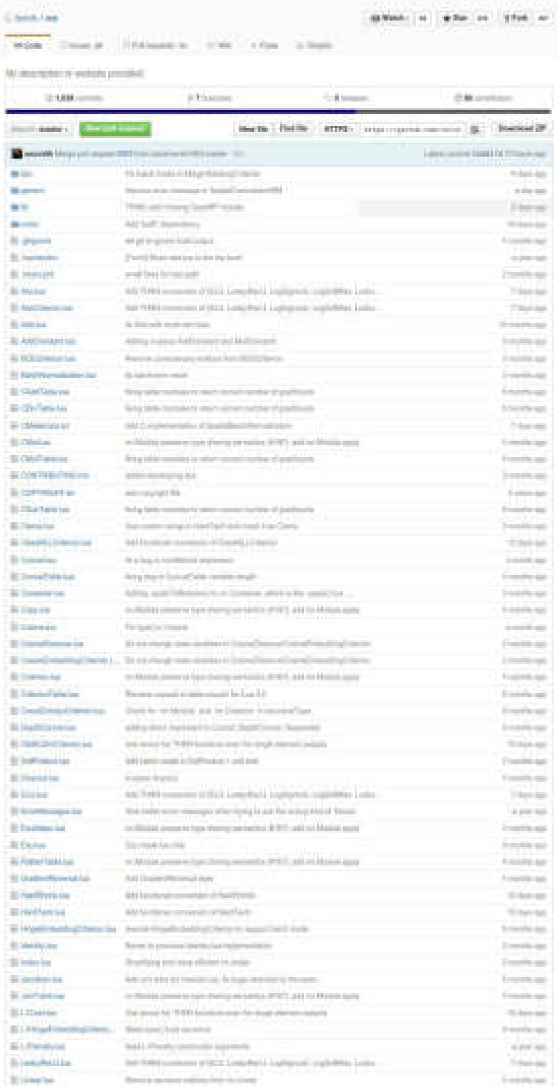
Slide from CS231n

Example: Torch Layers





Example: Torch Layers



=

Slide from CS231n

Example: Torch MulConstant

```lua
local MulConstant, parent = torch.class('nn.MulConstant', 'nn.Module')

function MulConstant:__init(constant_scalar,ip)
  parent.__init(self)
  assert(type(constant_scalar) == 'number', 'input is not scalar!')
  self.constant_scalar = constant_scalar

  -- default for inplace is false
   self.inplace = ip or false
   if (ip and type(ip) ~= 'boolean') then
      error('in-place flag must be boolean')
   end
end

function MulConstant:updateOutput(input)
  if self.inplace then
    input:mul(self.constant_scalar)
    self.output = input
  else
    self.output:resizeAs(input)
    self.output:copy(input)
    self.output:mul(self.constant_scalar)
  end
  return self.output
end

function MulConstant:updateGradInput(input, gradOutput)
  if self.gradInput then
    if self.inplace then
      gradOutput:mul(self.constant_scalar)
      self.gradInput = gradOutput
      -- restore previous input value
      input:div(self.constant_scalar)
    else
      self.gradInput:resizeAs(gradOutput)
      self.gradInput:copy(gradOutput)
      self.gradInput:mul(self.constant_scalar)
    end
    return self.gradInput
  end
end
```

$$f(X) = aX$$

Initialization()

Forward()

Backward()

Slide from CS231n

Example: Caffe Layers

Caffe Sigmoid Layer

```cpp
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

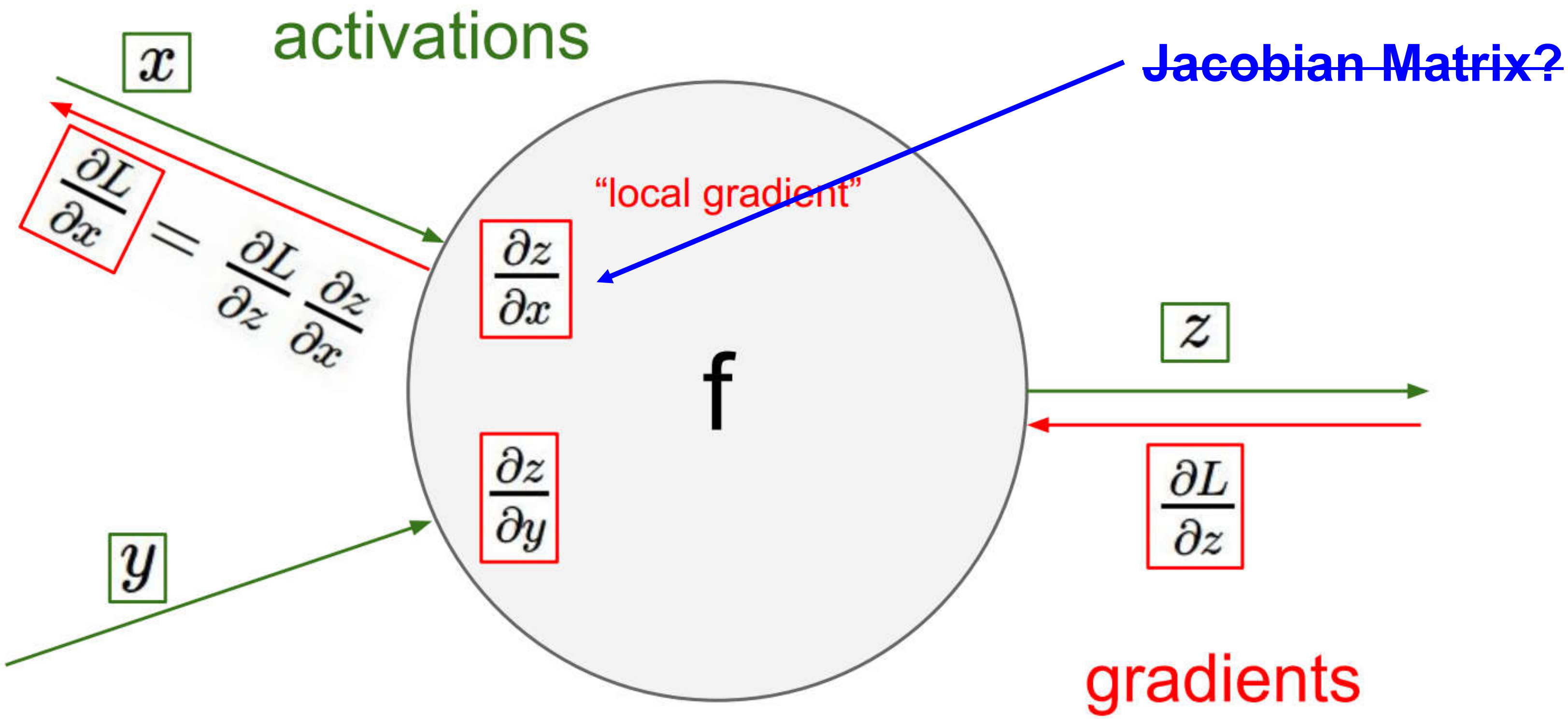
Forward_cpu

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

Backward_cpu

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

Gradients for Vectorized Code



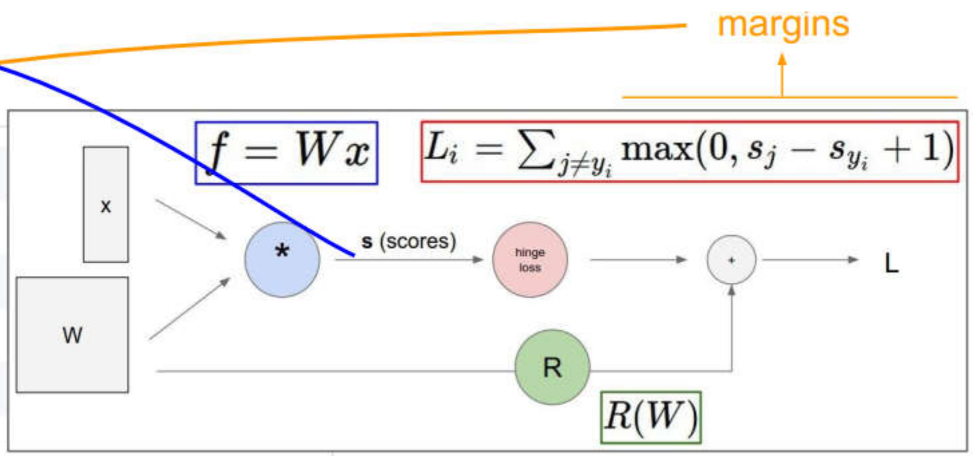
Slide from CS231n

Assignment #1(Q2,Q3): 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 = #...
```



Computation graph of the SVM loss

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.

Slide from CS231n

Neural Network

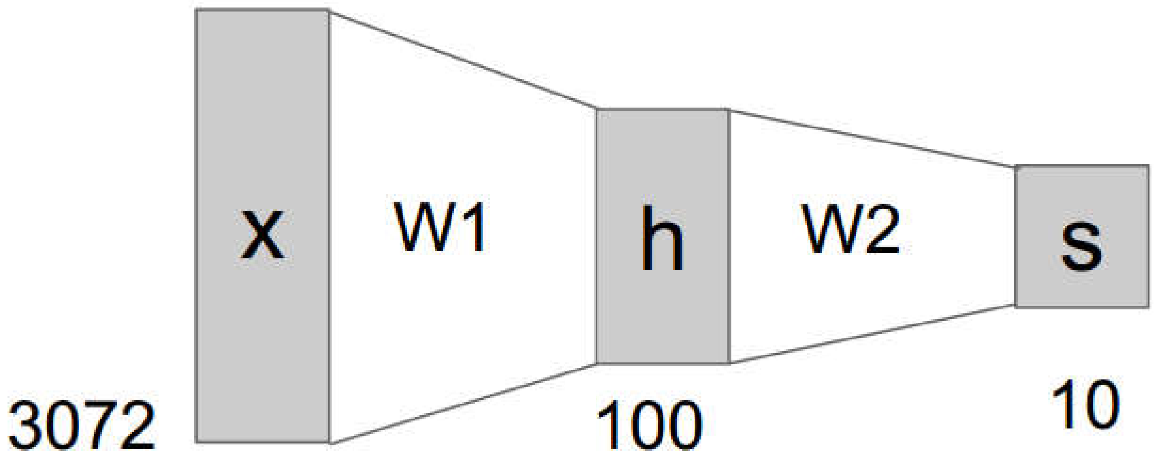
Neural Network: without the Brain Stuff

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

Neural Network: without the Brain Stuff

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

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$ more complex mathematic expression of X

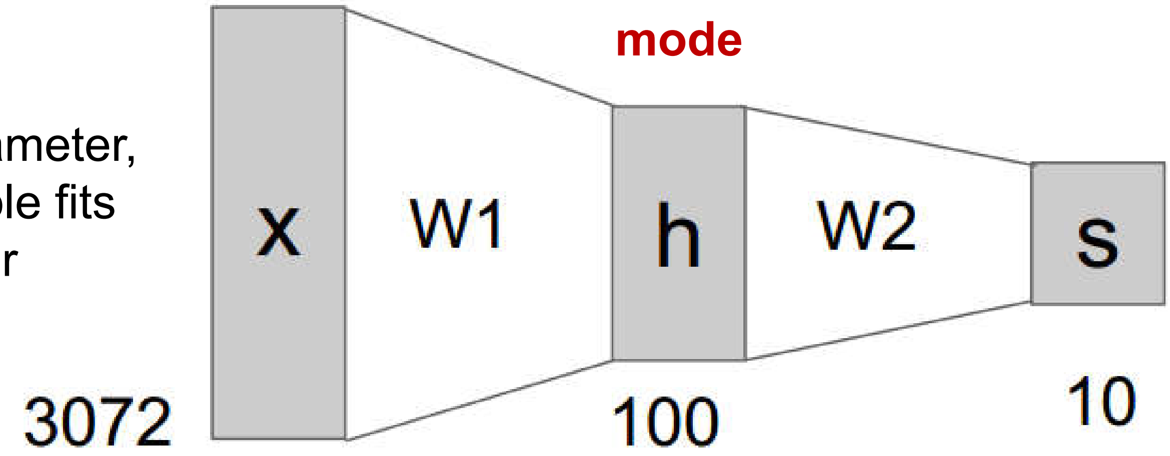


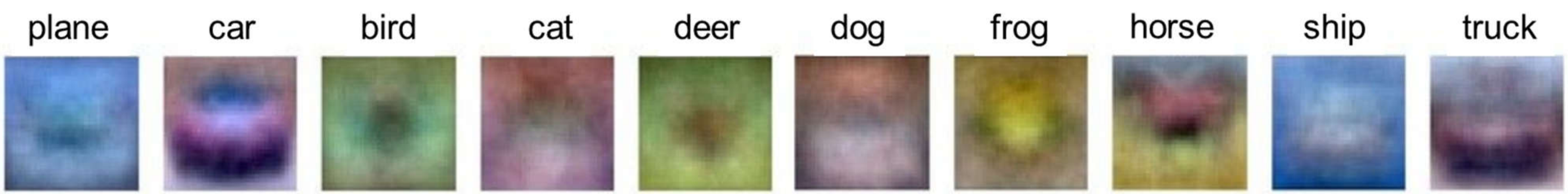
Neural Network: without the Brain Stuff

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

(Now) **2-layer Neural Network** $\boxed{f = W_2 \max(0, W_1 x)}$ more complex mathematic expression of X

- h is a hyperparameter, as big as possible fits in your computer





Slide from CS231n

Neural Network: Without the Brain Stuff

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

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$ more complex mathematic expression of X

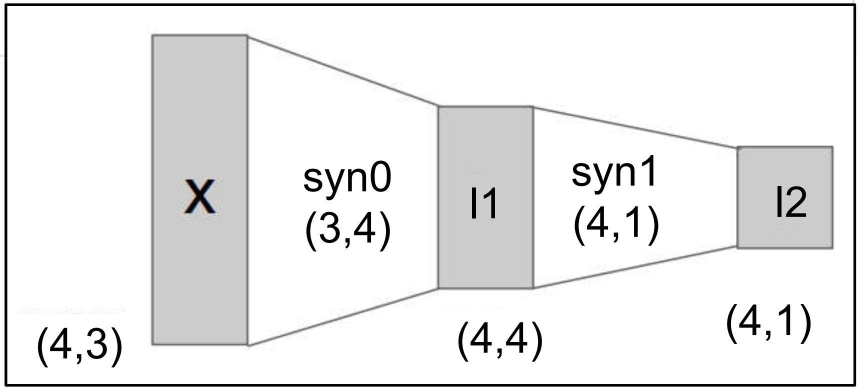
or **3-layer Neural Network** $\boxed{f = W_3 \max(0, W_2 \max(0, W_1 x))}$

Python Code for 2-layer Neural Network

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

```
X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
y = np.array([[0,1,1,0]]).T
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,1)) - 1
for j in xrange(60000):
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
    l2_delta = (y - l2)*(l2*(1-l2))
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
    syn1 += l1.T.dot(l2_delta)
    syn0 += X.T.dot(l1_delta)
```

Loss = (y – y_predict)



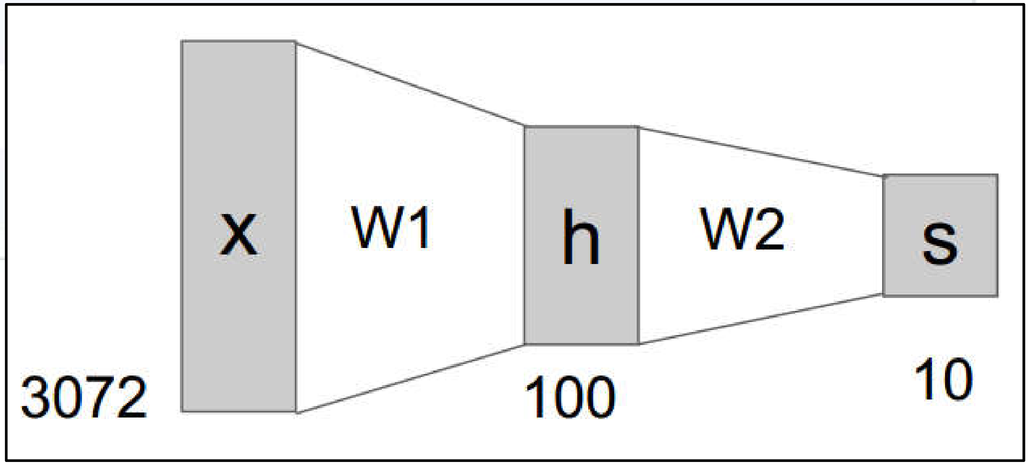
- Sigmoid activation function
- Logistic regression loss

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

Assignment: Writing 2-Layer Net

- Stage your forward/backward computation!

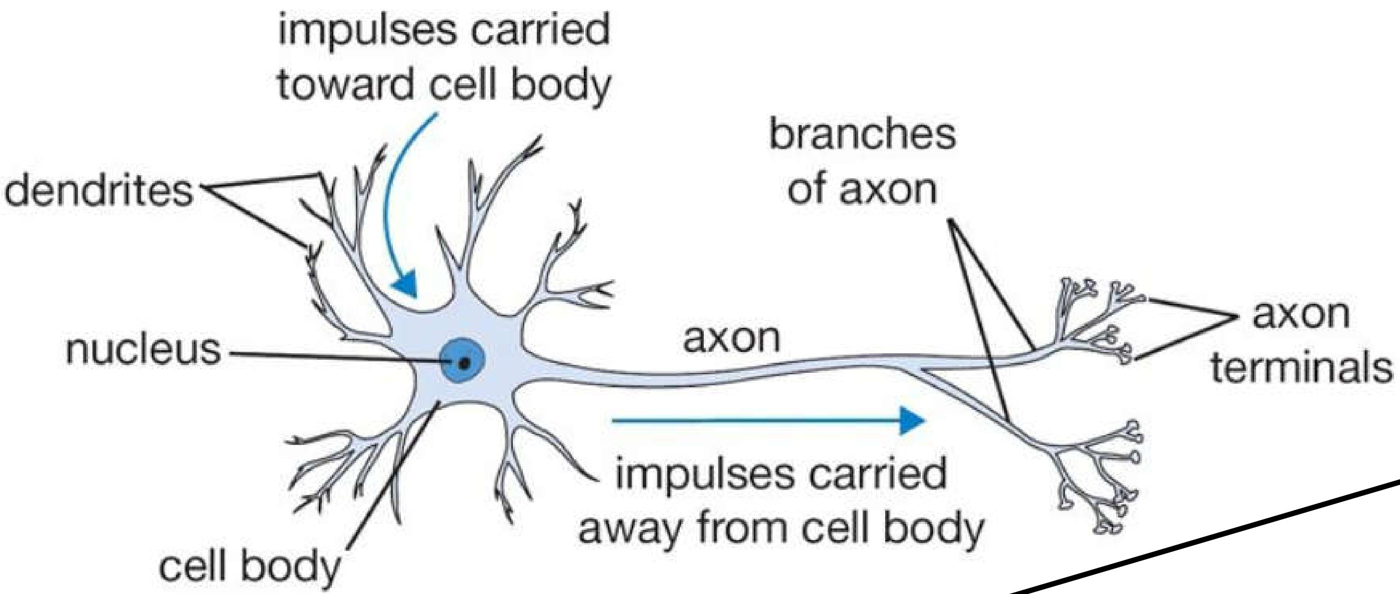
```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```

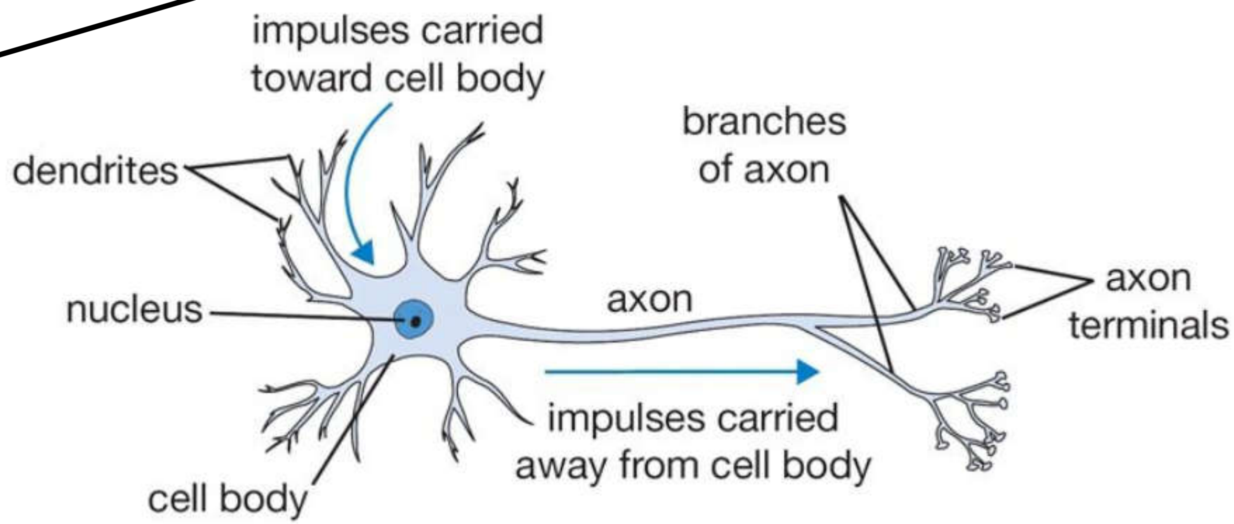


Slide from CS231n

Neural Network Without Brain Stuff

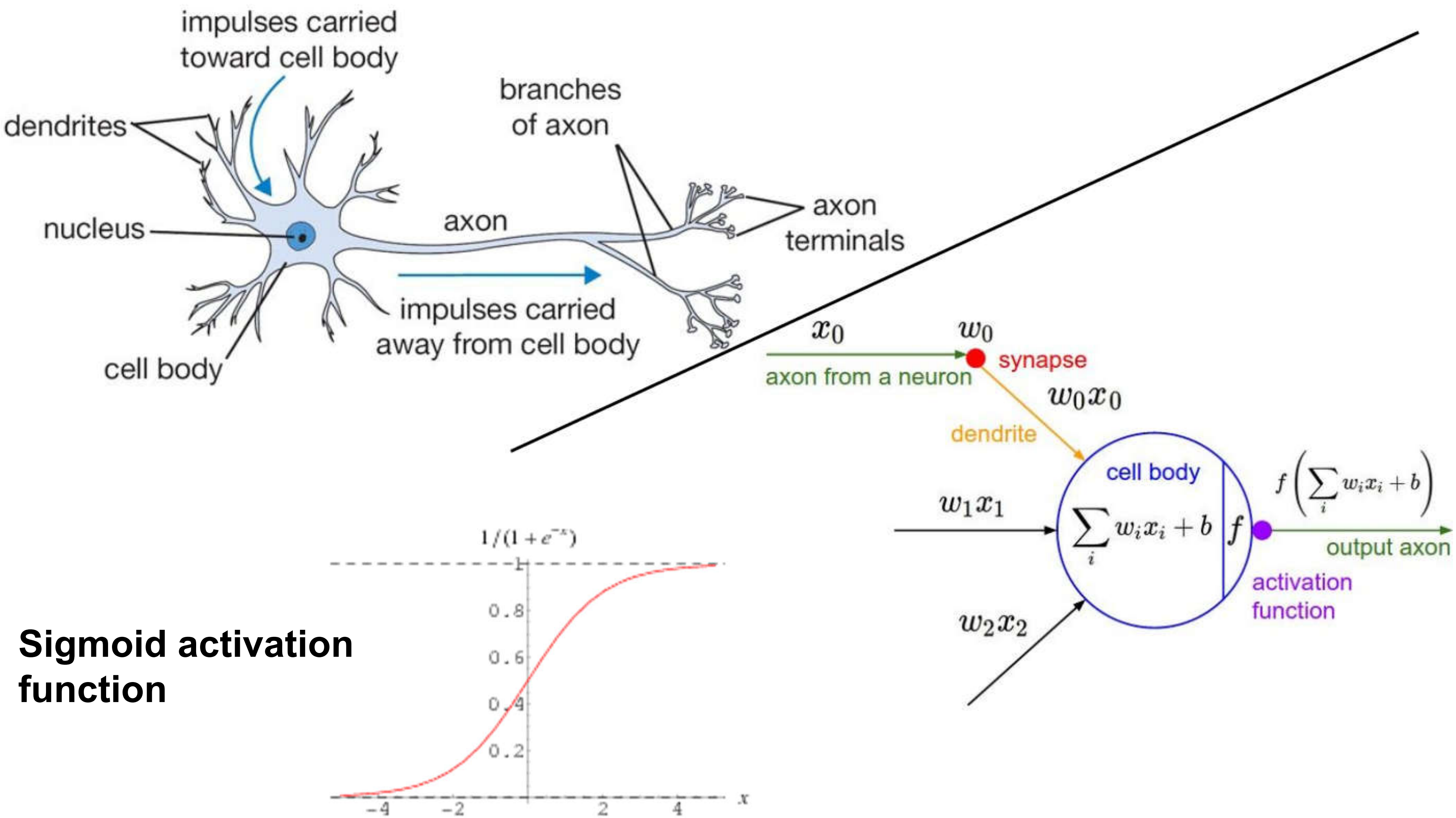
Biological Neuron Without Brain Stuff





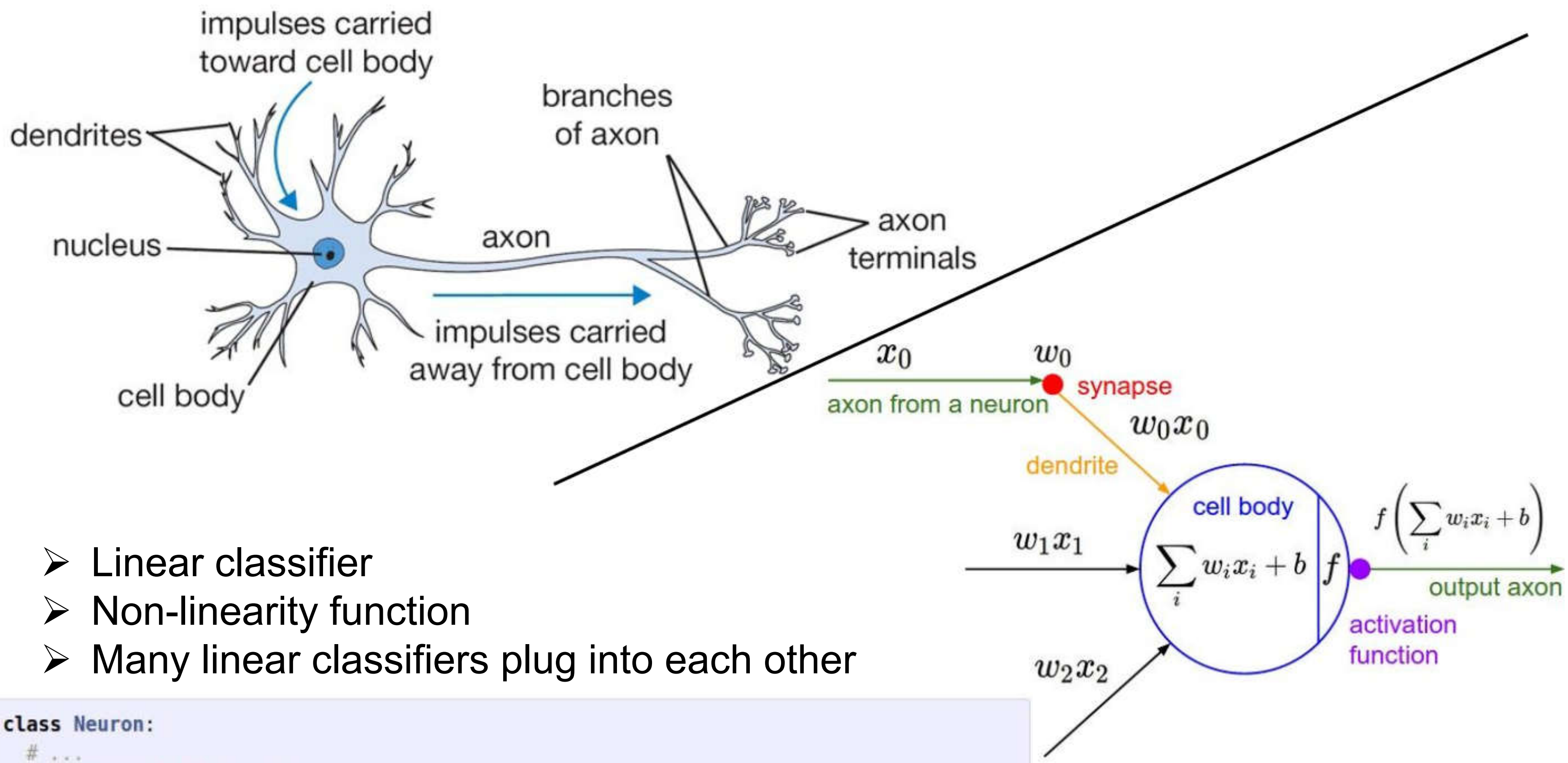
Slide from CS231n

Biological Neuron Without Brain Stuff



Sigmoid activation function

Biological Neuron Without Brain Stuff



- Linear classifier
- Non-linearity function
- Many linear classifiers plug into each other

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

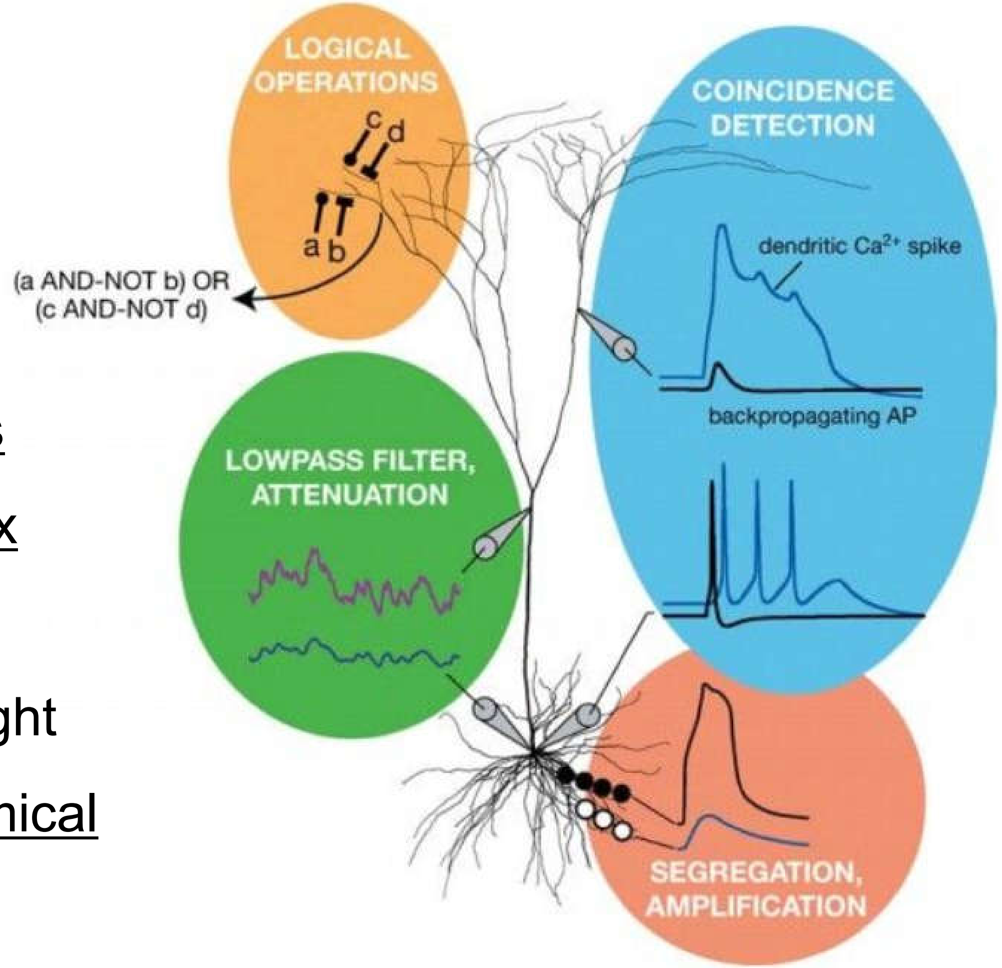
Slide from CS231n

Biological Neuron

Be very careful with your Brain analogies:

Biological Neurons:

- Many different types of neurons
- **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]

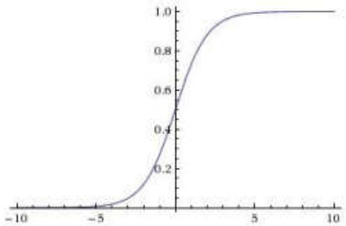
Activation Functions

- Historically people use **Sigmoid and tanh**
- From 2012 **ReLu** became quite popular (default recommendation)
- A few kind of hipster activation functions: **Leaky Relu**, **Maxout,** and **ELU**

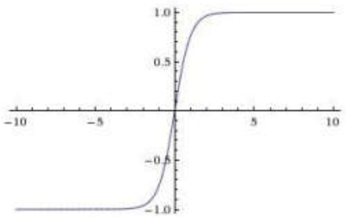
Question #2: why we need activation function?

Sigmoid

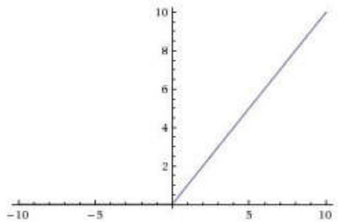
$$\sigma(x) = 1/(1+e^{-x})$$



tanh tanh(x)

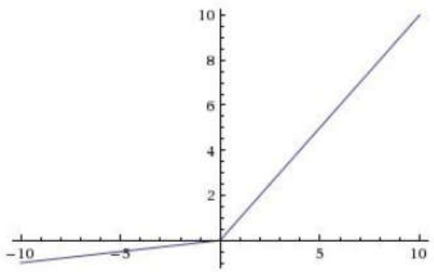


ReLU max(0,x)



Leaky ReLU

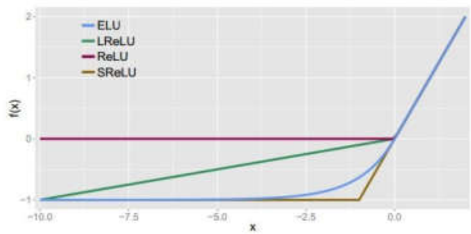
max(0.1x, x)



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

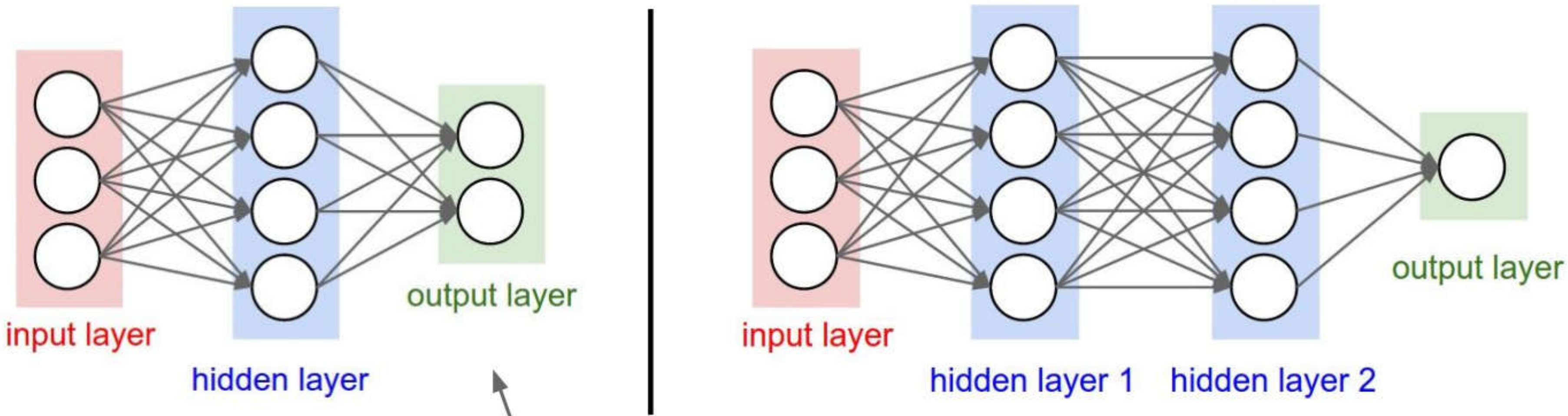
ELU

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



Slide from CS231n

Neural Networks: Architectures



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

“Fully-connected” layers

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

Slide from CS231n

Example Feed-Forward Computation of a Neural Network

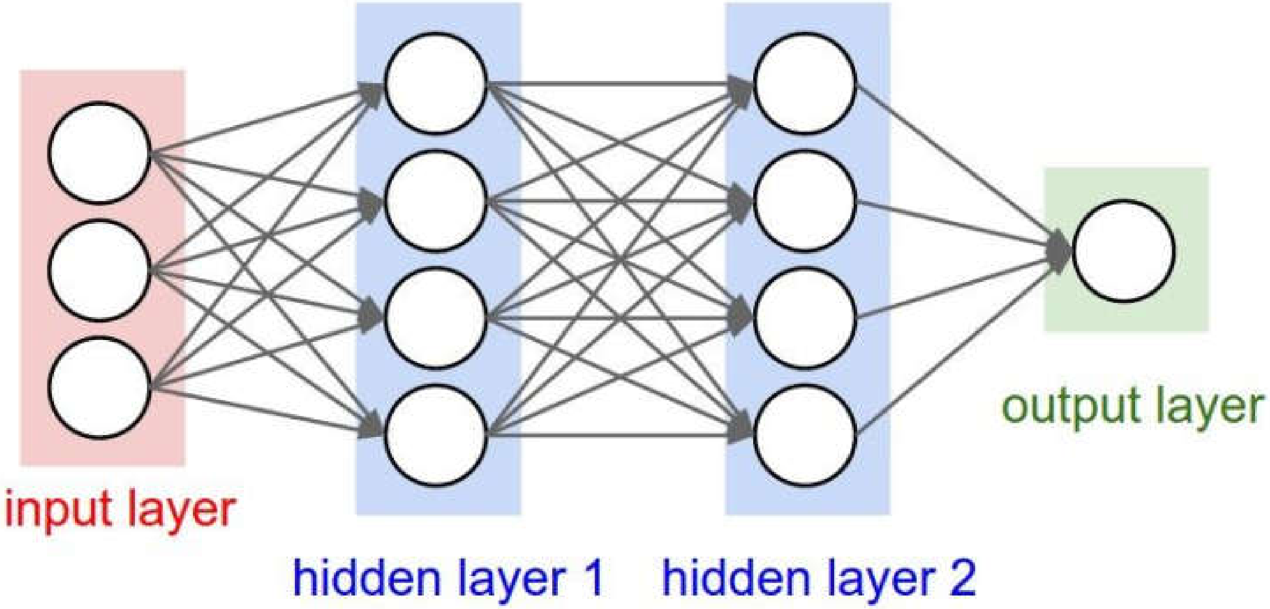
- Assume inputs and weights are 1-D numpy arrays and bias is a number

```
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
```

Slide from CS231n

Example Feed-Forward Computation of a Neural Network

- We can compute an entire set of neurons in a single hidden layer using **a single time matrix multiplication**

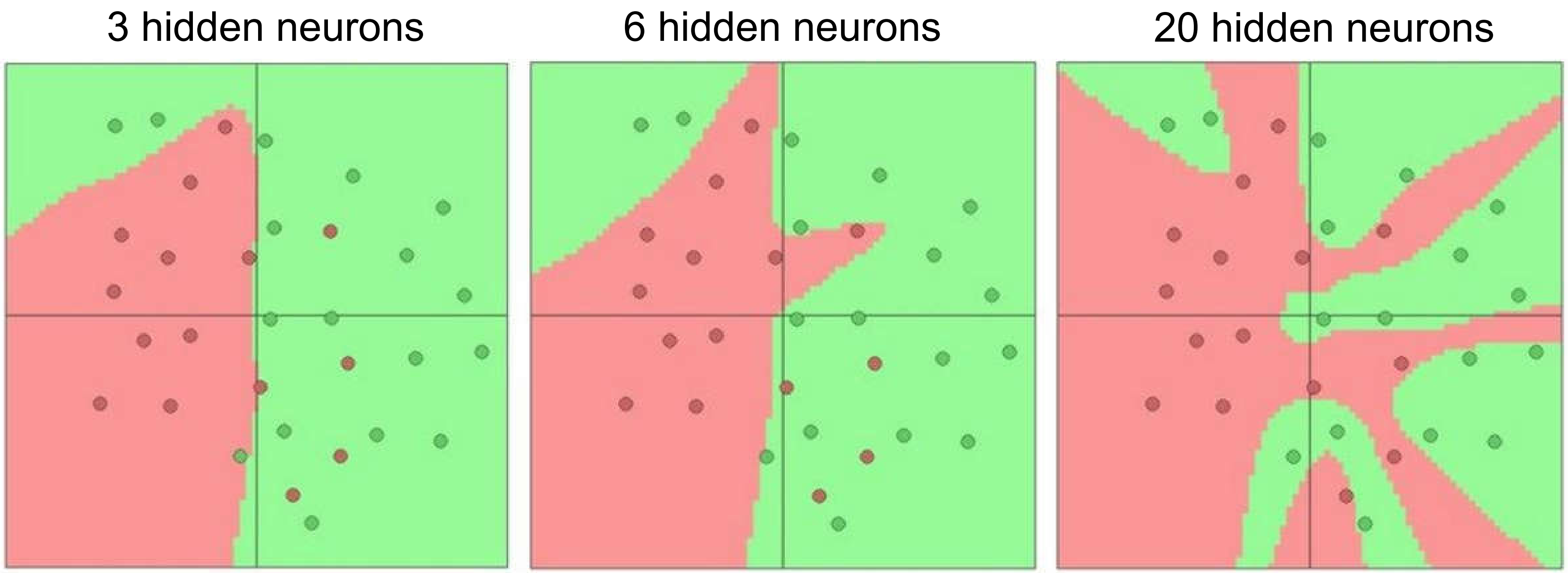


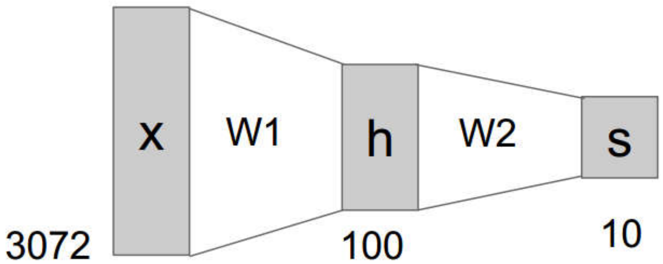

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate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

Slide from CS231n

Example of 2-Layer Neural Network

- Setting the **number of neurons** in the hidden layer
- h is **a hyperparameter**

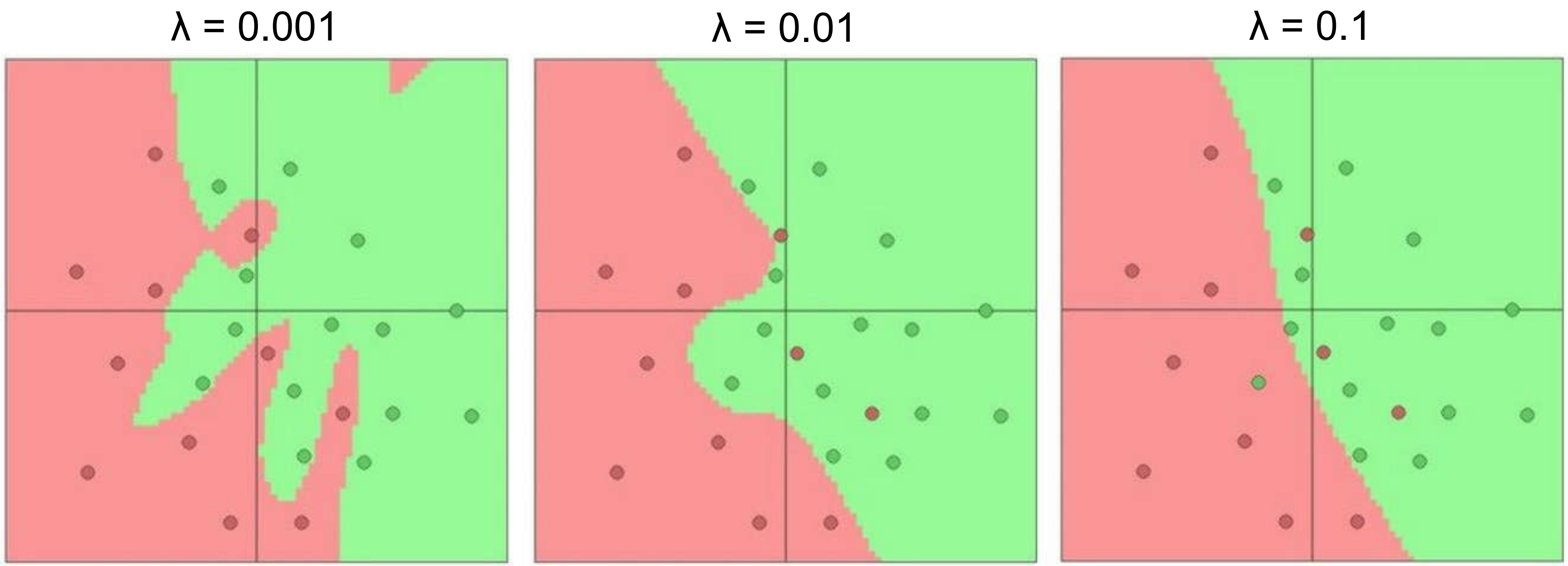




> More neurons = more capacity

Example of 2-Layer Neural Network

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



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

You can play with this demo over at ConvNetJs:
http://cs.Stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Slide from CS231n

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** (<u>but might have to regularize more strongly</u>)

Slide from CS231n

Next Lecture

More than you ever wanted to know about

Neural Networks and **how to train them**.

Slide from CS231n

Thank you for your attention!