

Backpropagation and Neural Nets

EECS 442 – David Fouhey

Winter 2022, University of Michigan

http://web.eecs.umich.edu/~fouhey/teaching/EECS442_W22/

So Far: Linear Models

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- Example: find \mathbf{w} minimizing squared error over data
 - Each datapoint represented by some vector \mathbf{x}
 - Can find optimal \mathbf{w} with ~10 line derivation

Last Class

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n L(y_i, f(\mathbf{x}; \mathbf{w}))$$

- What about an arbitrary loss function L ?
- What about an arbitrary parametric function f ?
- Solution: take the gradient, do gradient descent

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \underbrace{\alpha \nabla_{\mathbf{w}} L(f(\mathbf{w}_i))}_{\text{gradient}}$$

What if $L(f(\mathbf{w}))$ is complicated?

Today!

Taking the Gradient – Review

$$f(x) = (-x + 3)^2$$

$$f = q^2 \quad q = r + 3 \quad r = -x$$

$$\frac{\partial f}{\partial q} = 2q \quad \frac{\partial q}{\partial r} = 1 \quad \frac{\partial r}{\partial x} = -1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial r} \frac{\partial r}{\partial x} = 2q * 1 * -1$$

Chain rule

$$= -2(-x + 3)$$

$$= 2x - 6$$

Supplemental Reading

- Lectures can only introduce you to a topic
- You will solidify your knowledge by **doing**
- I highly recommend working through everything in the Stanford CS213N resources
 - <http://cs231n.github.io/optimization-2/>
- These slides follow the general examples with a few modifications. The primary difference is that I define local variables n , m per-block.

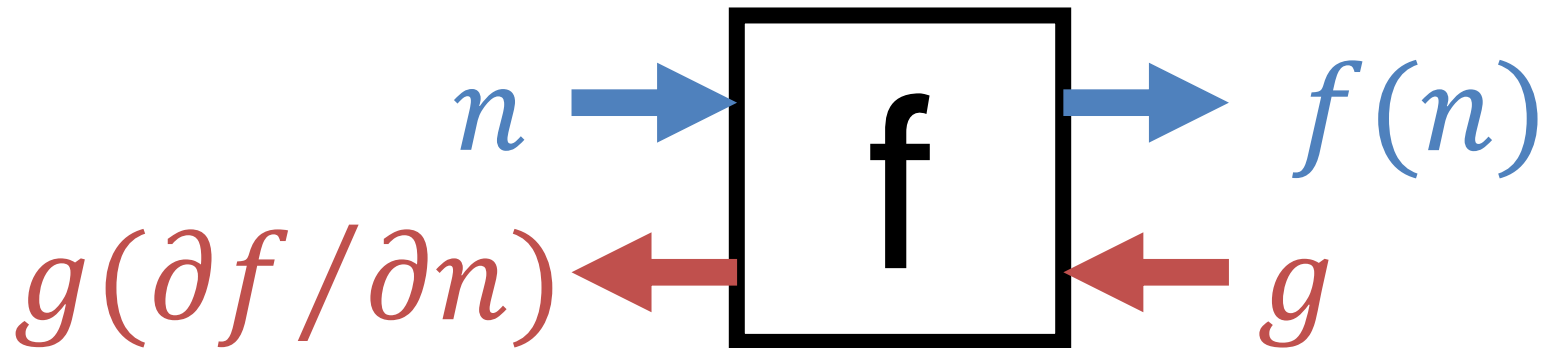
Let's Do This Another Way

Suppose we have a box representing a function f .

This box does two things:

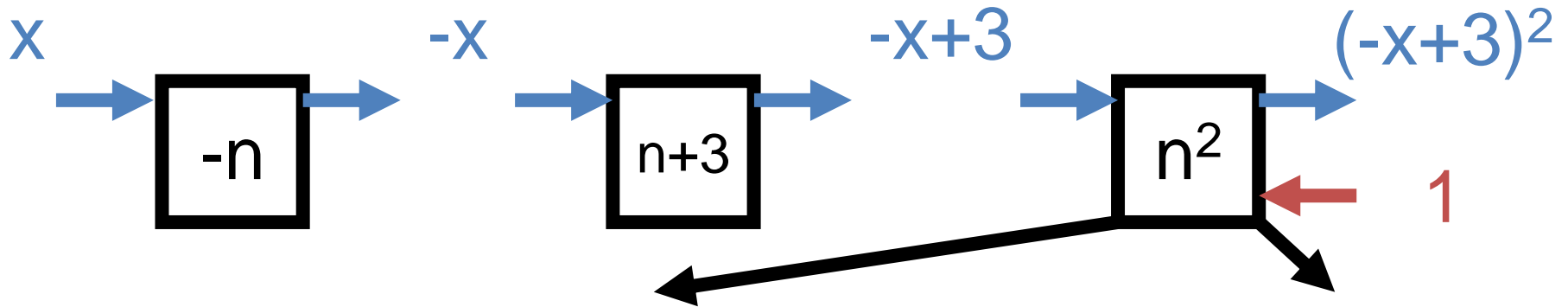
Forward: Given forward input n , compute $f(n)$

Backwards: Given backwards input g , return $g \cdot df/dn$



Let's Do This Another Way

$$f(x) = (-x + 3)^2$$

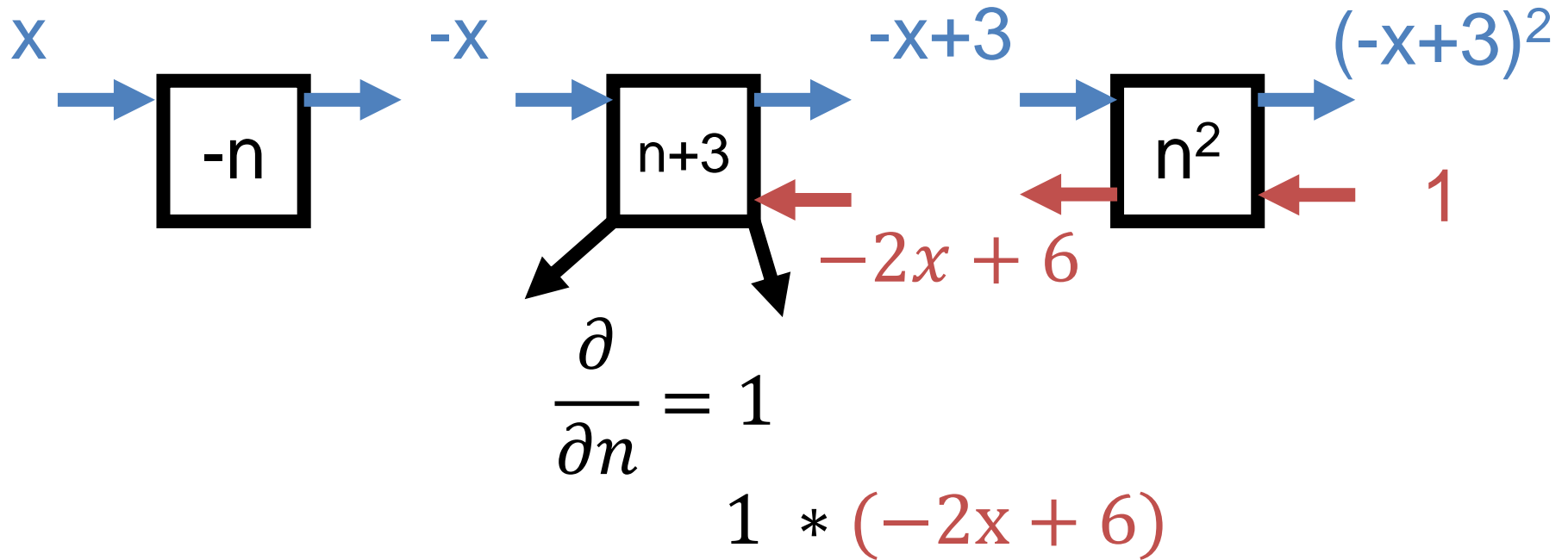


$$\begin{aligned}\frac{\partial}{\partial n} n^2 &= 2n = 2(-x + 3) \\ &= -2x + 6\end{aligned}$$

$$\frac{\partial}{\partial n} * 1 = (-2x + 6) * 1$$

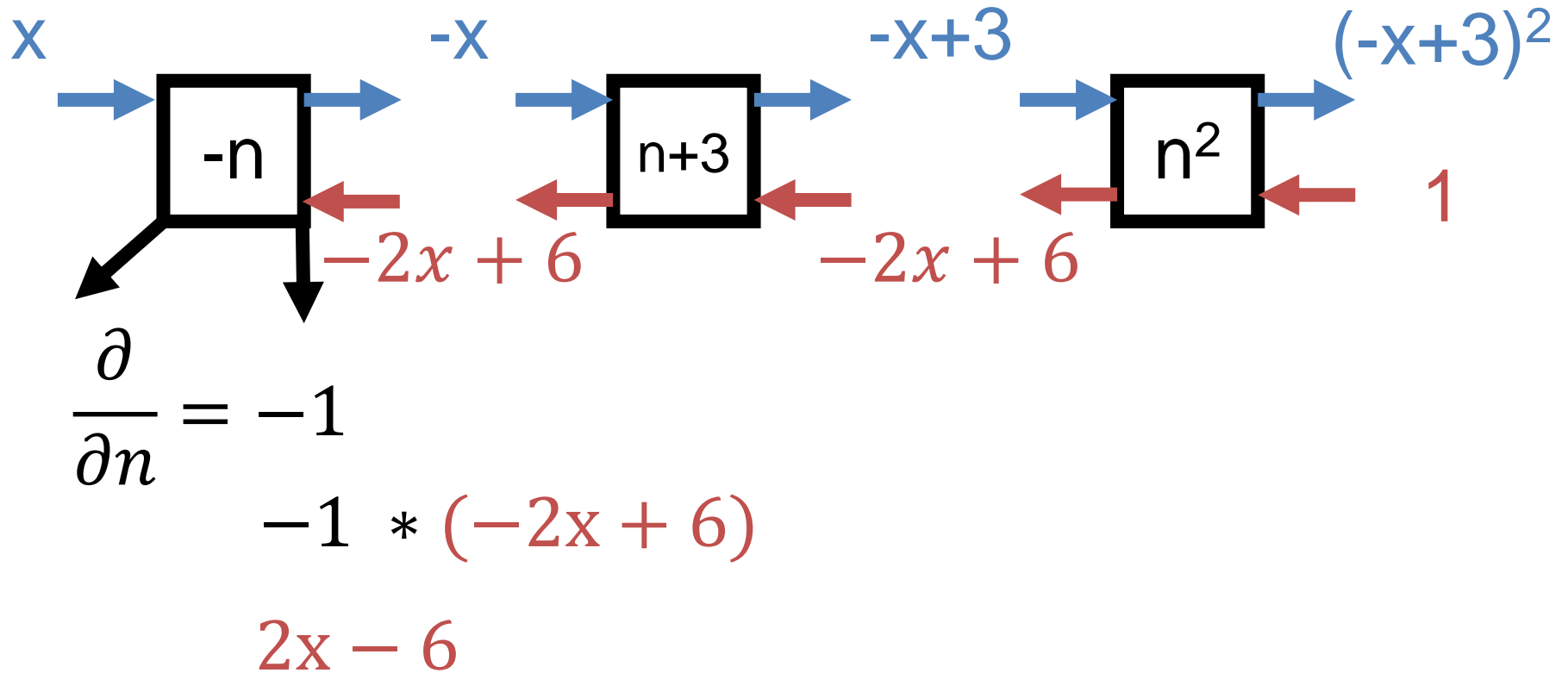
Let's Do This Another Way

$$f(x) = (-x + 3)^2$$



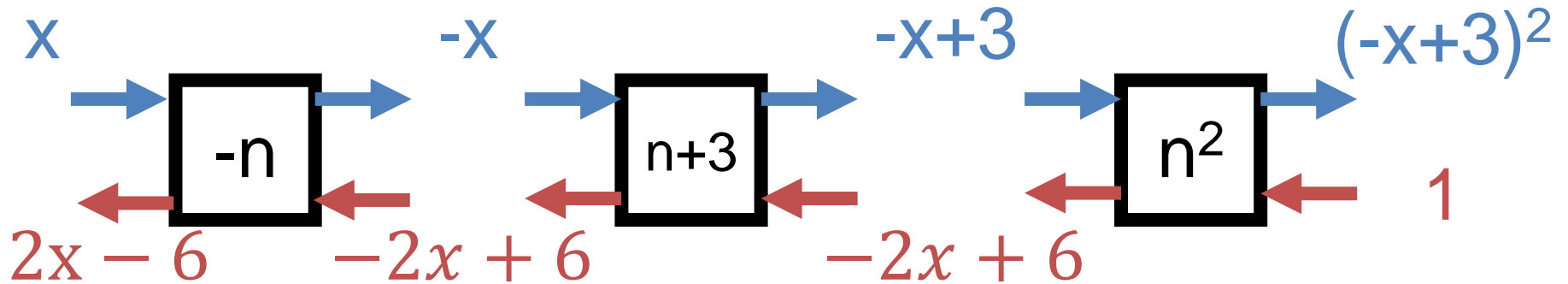
Let's Do This Another Way

$$f(x) = (-x + 3)^2$$



Let's Do This Another Way

$$f(x) = (-x + 3)^2$$

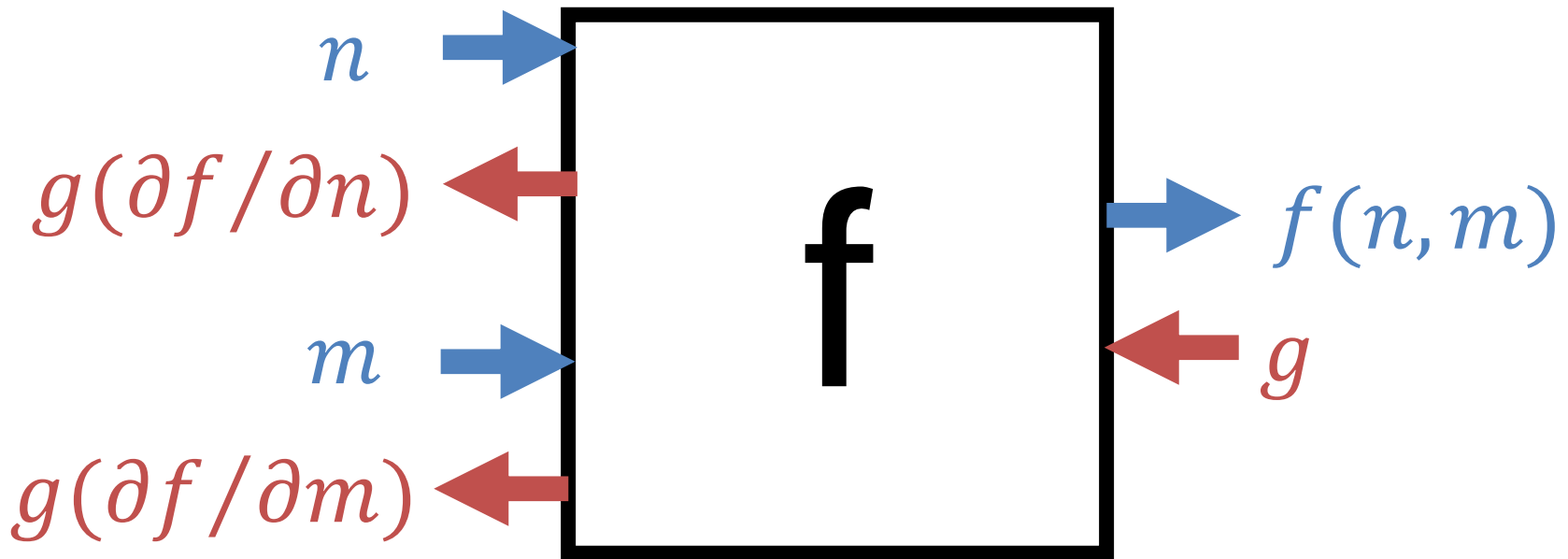


Two Inputs

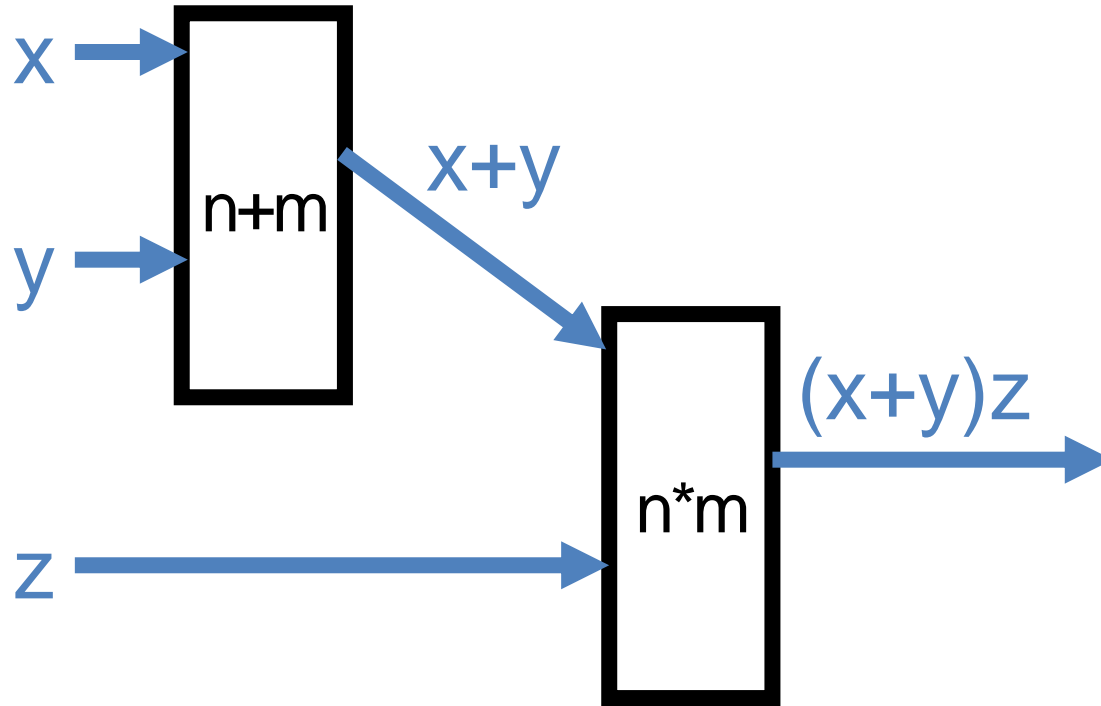
Given two inputs, just have two input/output wires

Forward: the same

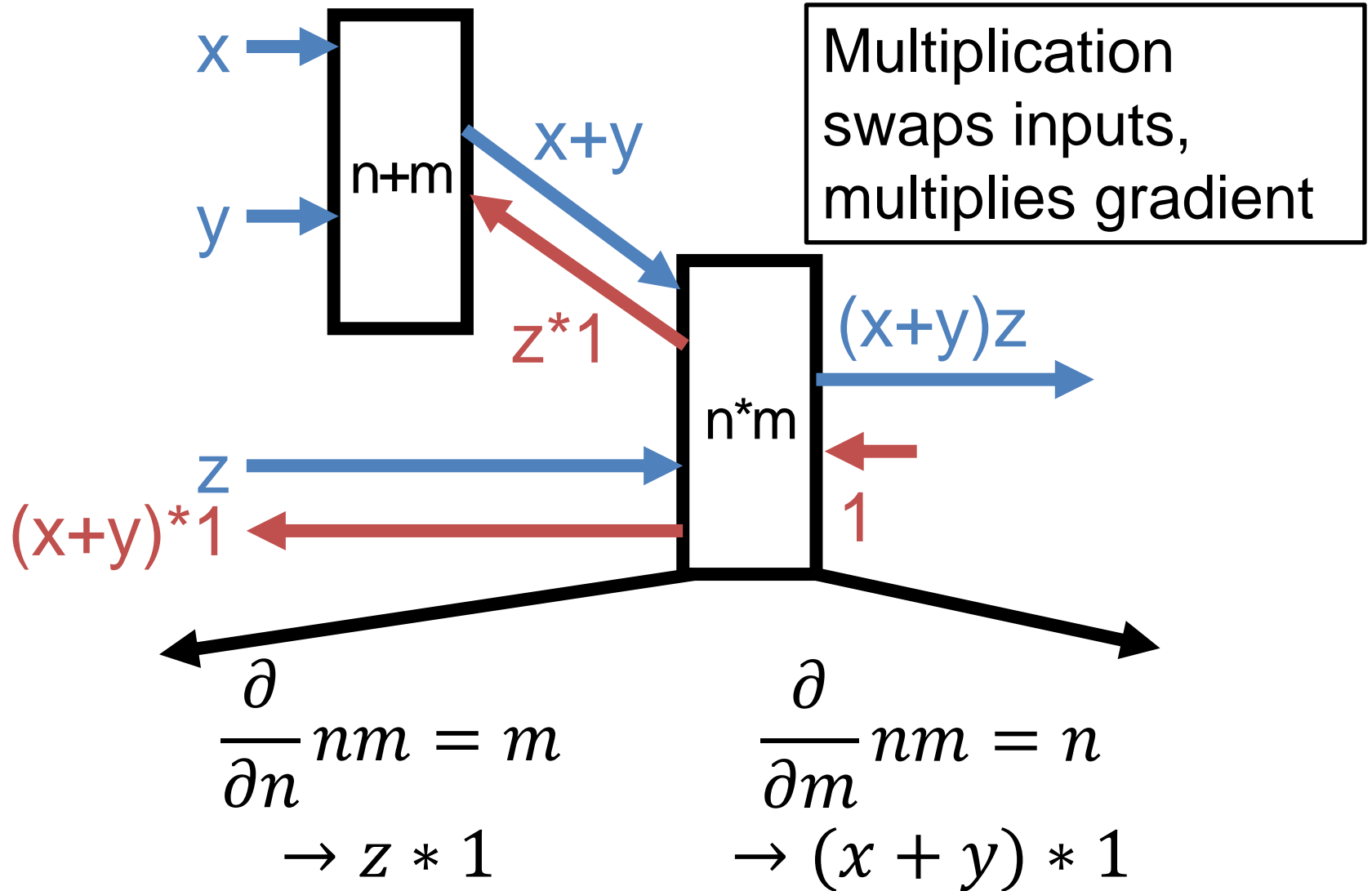
Backward: the same – send gradients with respect to each variable



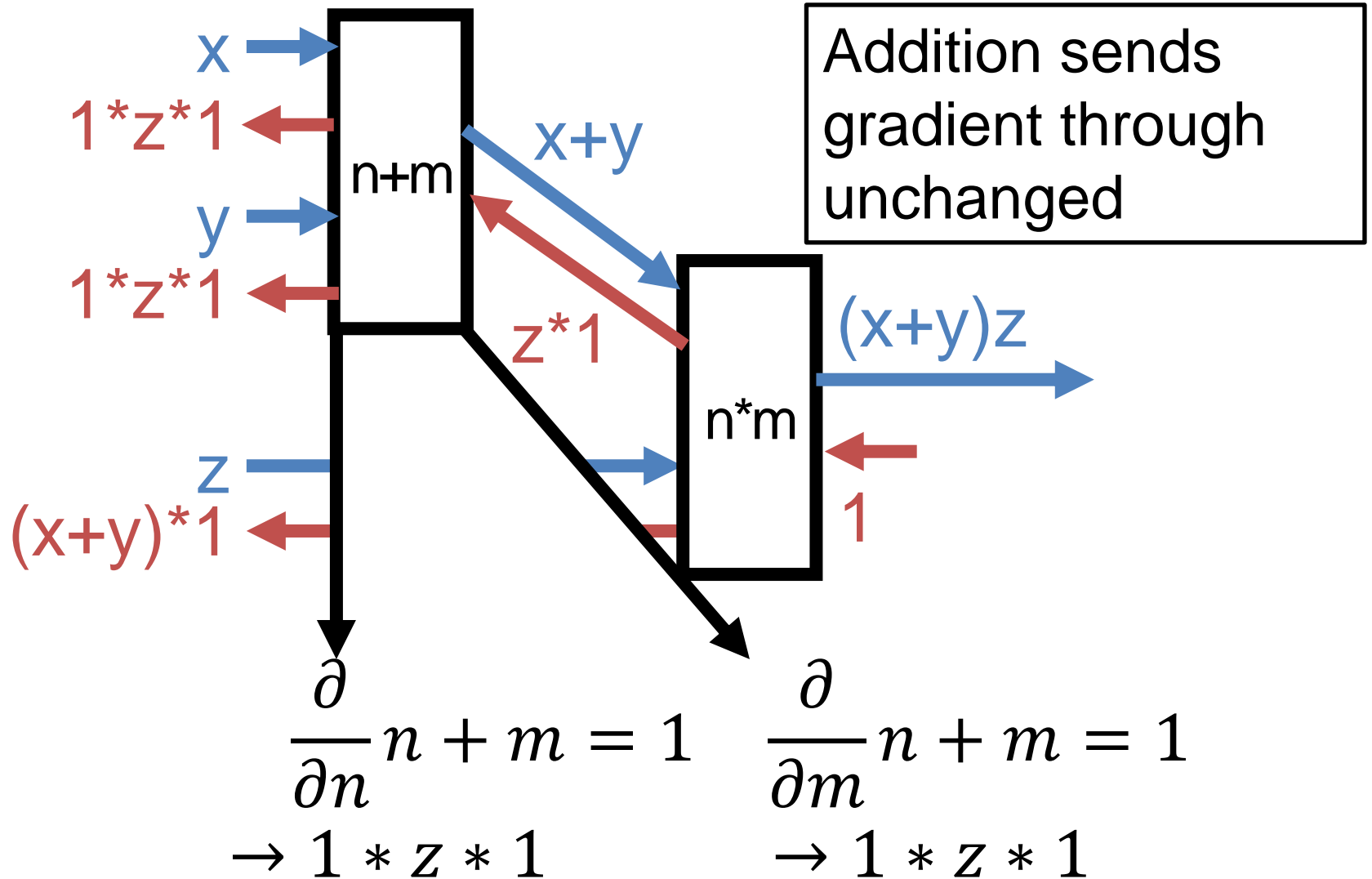
$$f(x,y,z) = (x+y)z$$



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

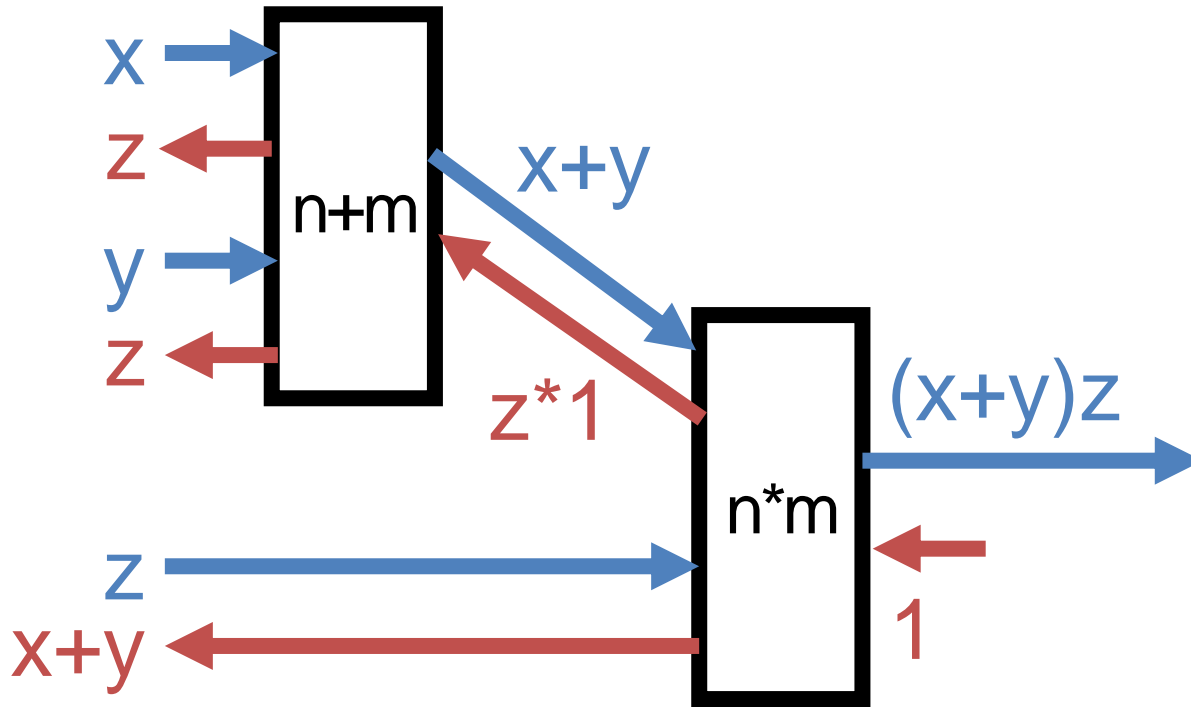


$$f(x,y,z) = (x+y)z$$



Addition sends gradient through unchanged

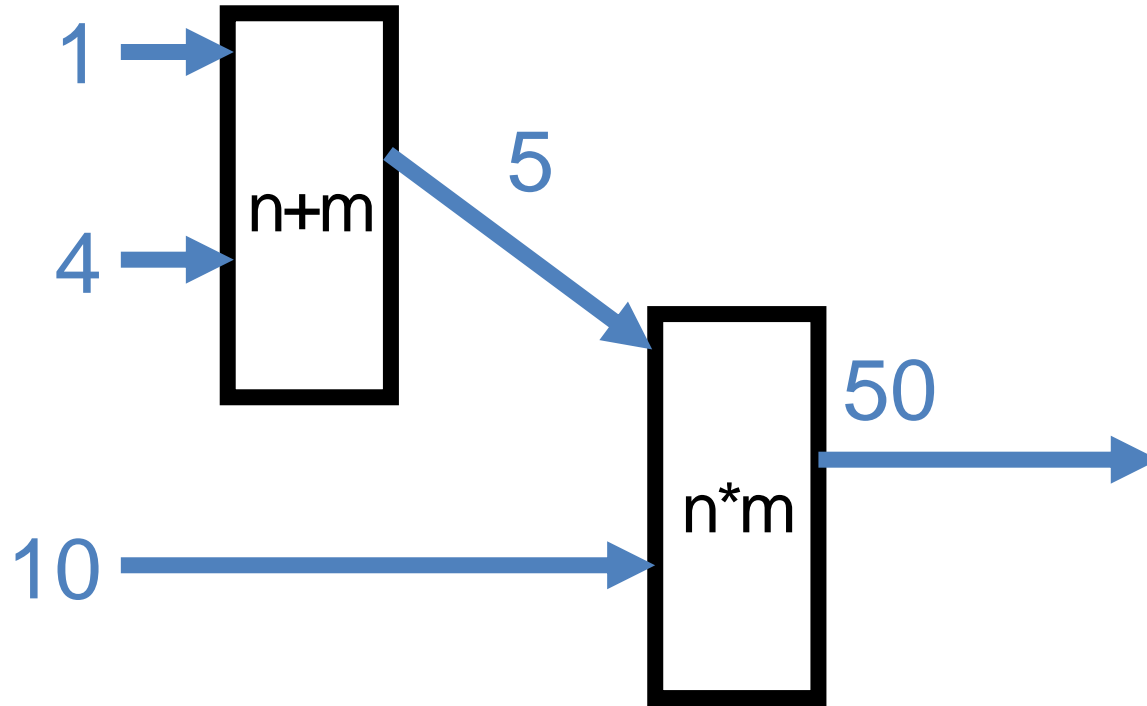
$$f(x, y, z) = (x+y)z$$



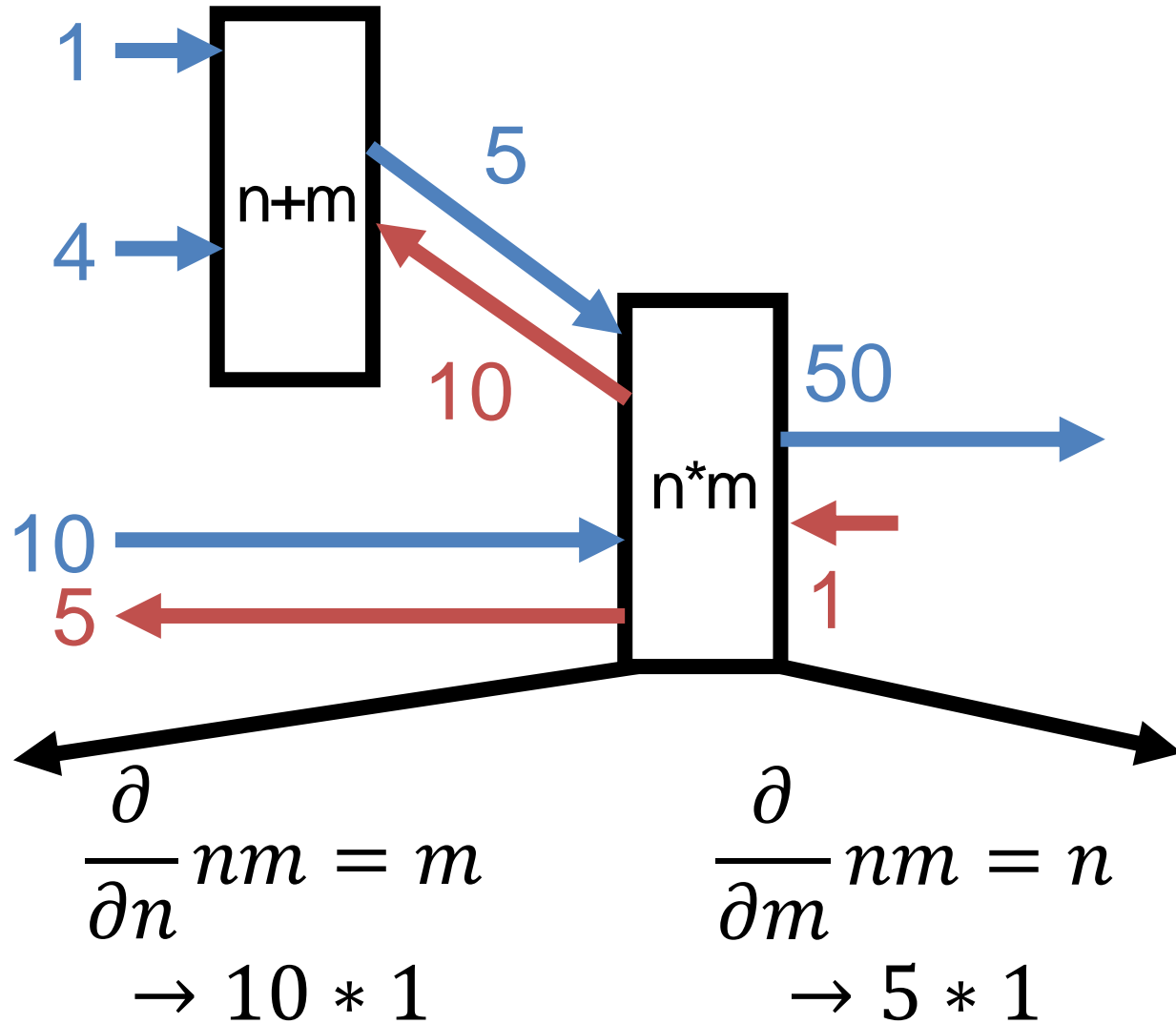
$$\frac{\partial (x+y)z}{\partial x} = z \quad \frac{\partial (x+y)z}{\partial y} = z \quad \frac{\partial (x+y)z}{\partial z} = (x+y)$$

Once More, With Numbers!

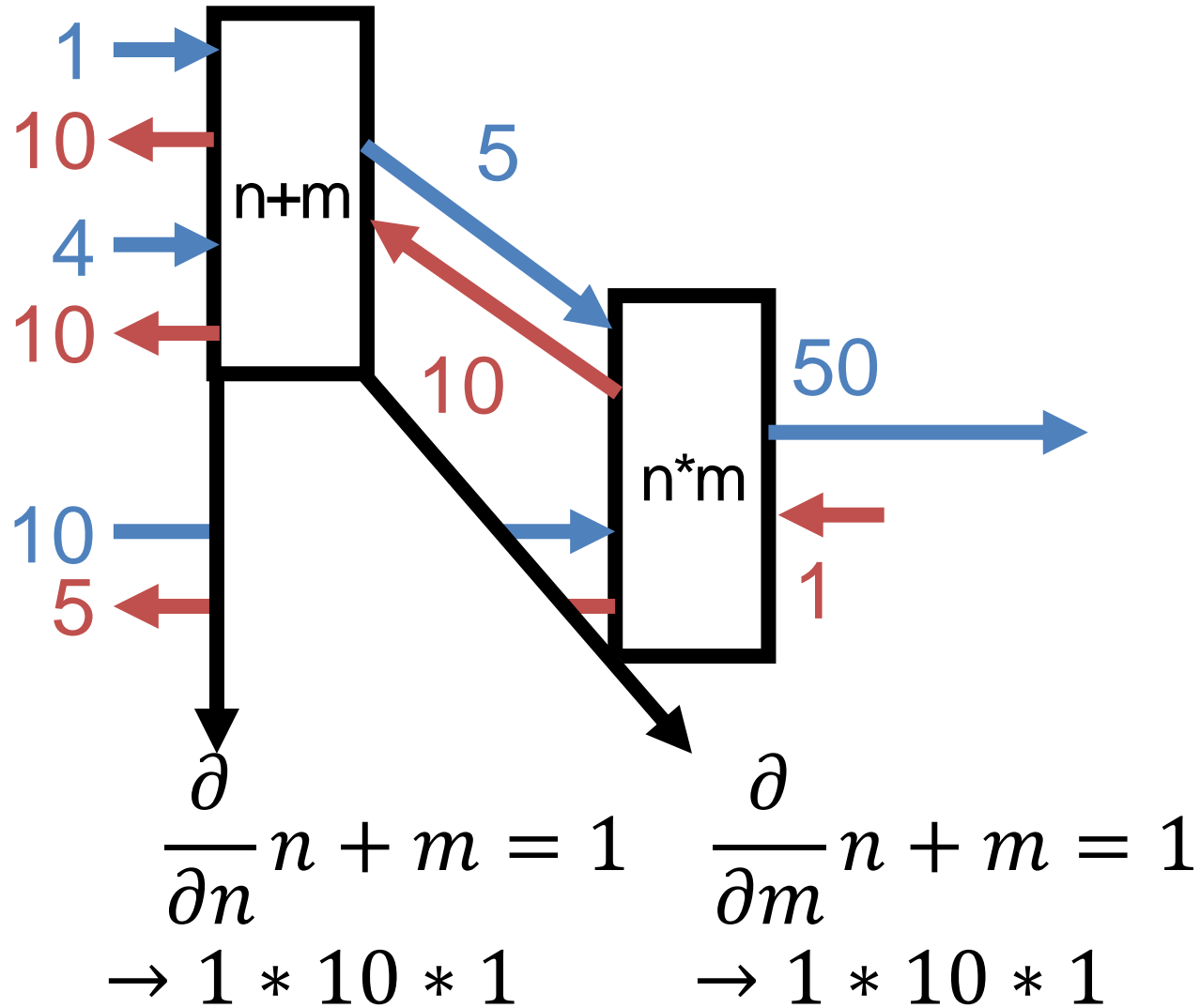
$$f(x,y,z) = (x+y)z$$



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



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



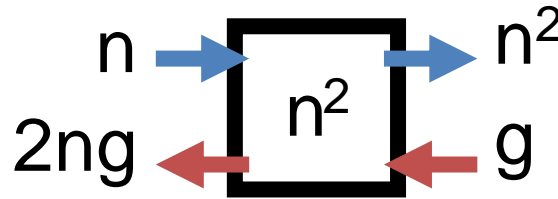
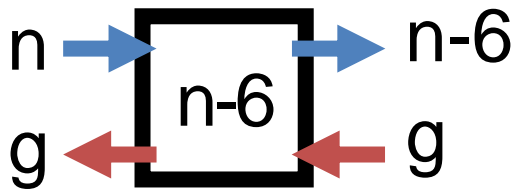
Think You've Got It?

$$L(x) = (w - 6)^2$$

- We want to fit a model w that just will equal 6.
- World's most basic linear model / neural net: no inputs, just constant output.

I'll Need a Few Volunteers

$$L(x) = (w - 6)^2$$



Job #1 ($n-6$):

Forward:

Compute $n-6$

Backwards:

Multiply by 1

Job #2 (n^2):

Forward:

Compute n^2

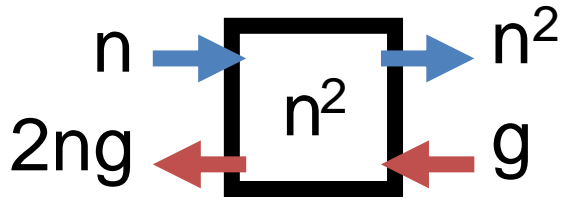
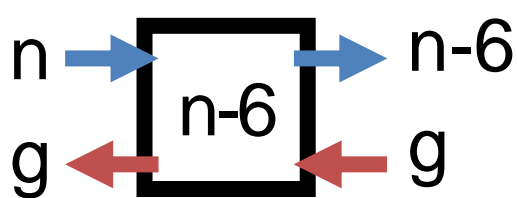
Backwards:

Multiply by $2n$

Job #3:

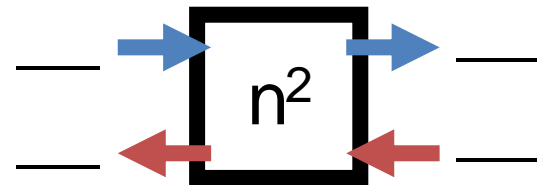
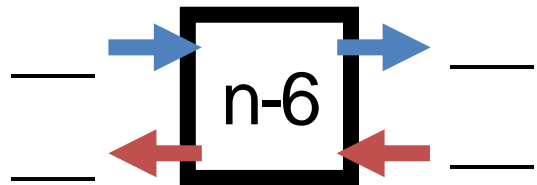
Backwards:

Give me a 1



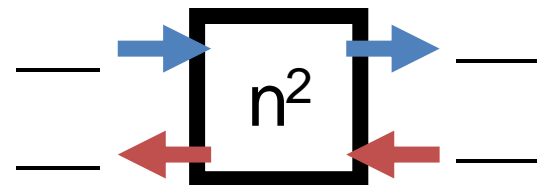
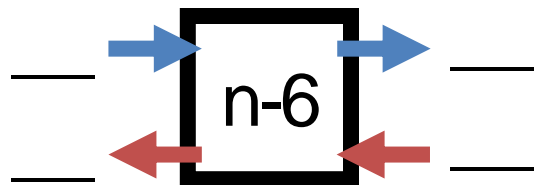
$$L(x) = (w - 6)^2$$

w_0 : _____



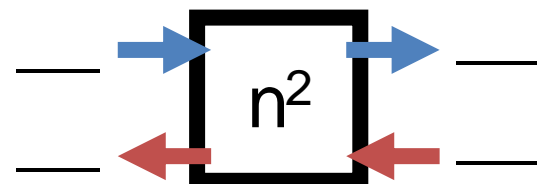
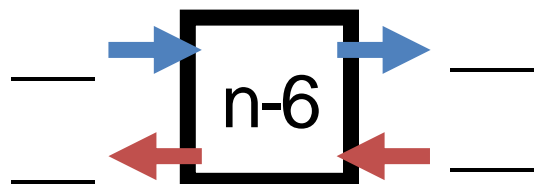
$$w_1 = w_0 - (1/4) \text{ _____} =$$

w_1 : _____



$$w_2 = w_1 - (1/4) \text{ _____} =$$

w_2 : _____



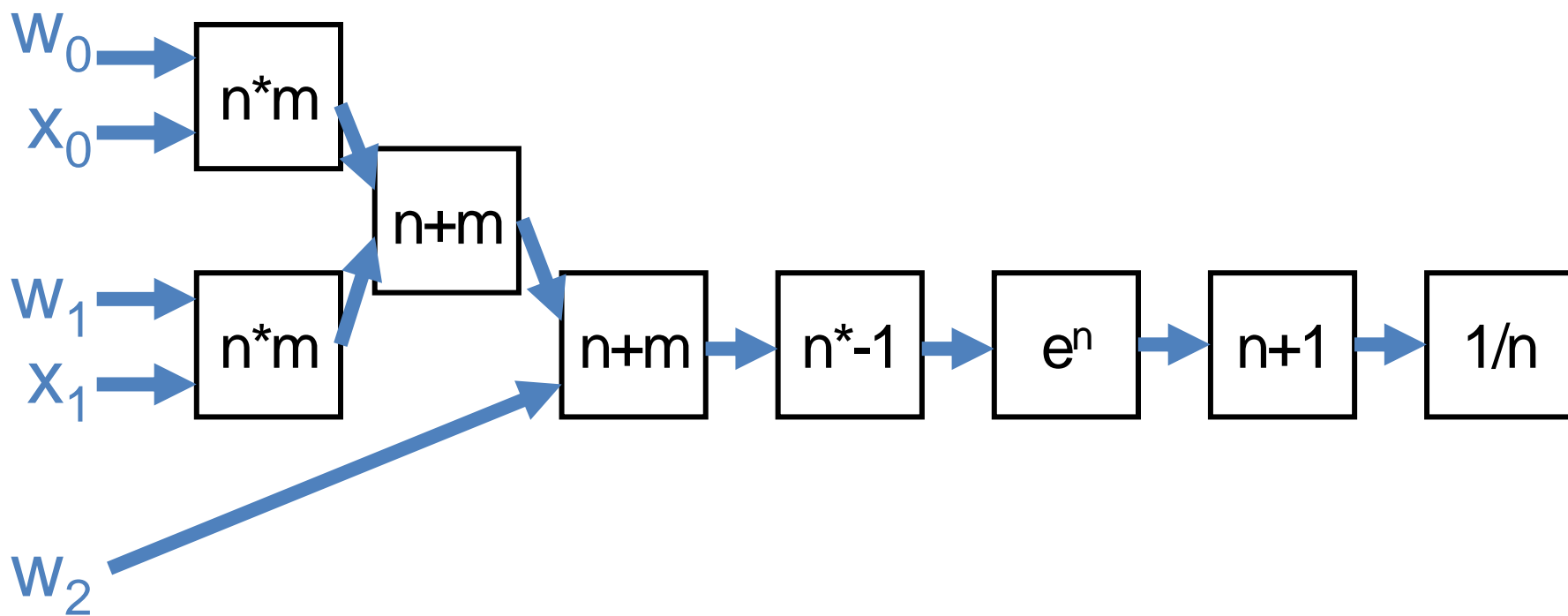
$$w_3 = w_2 - (1/4) \text{ _____} =$$

Preemptively

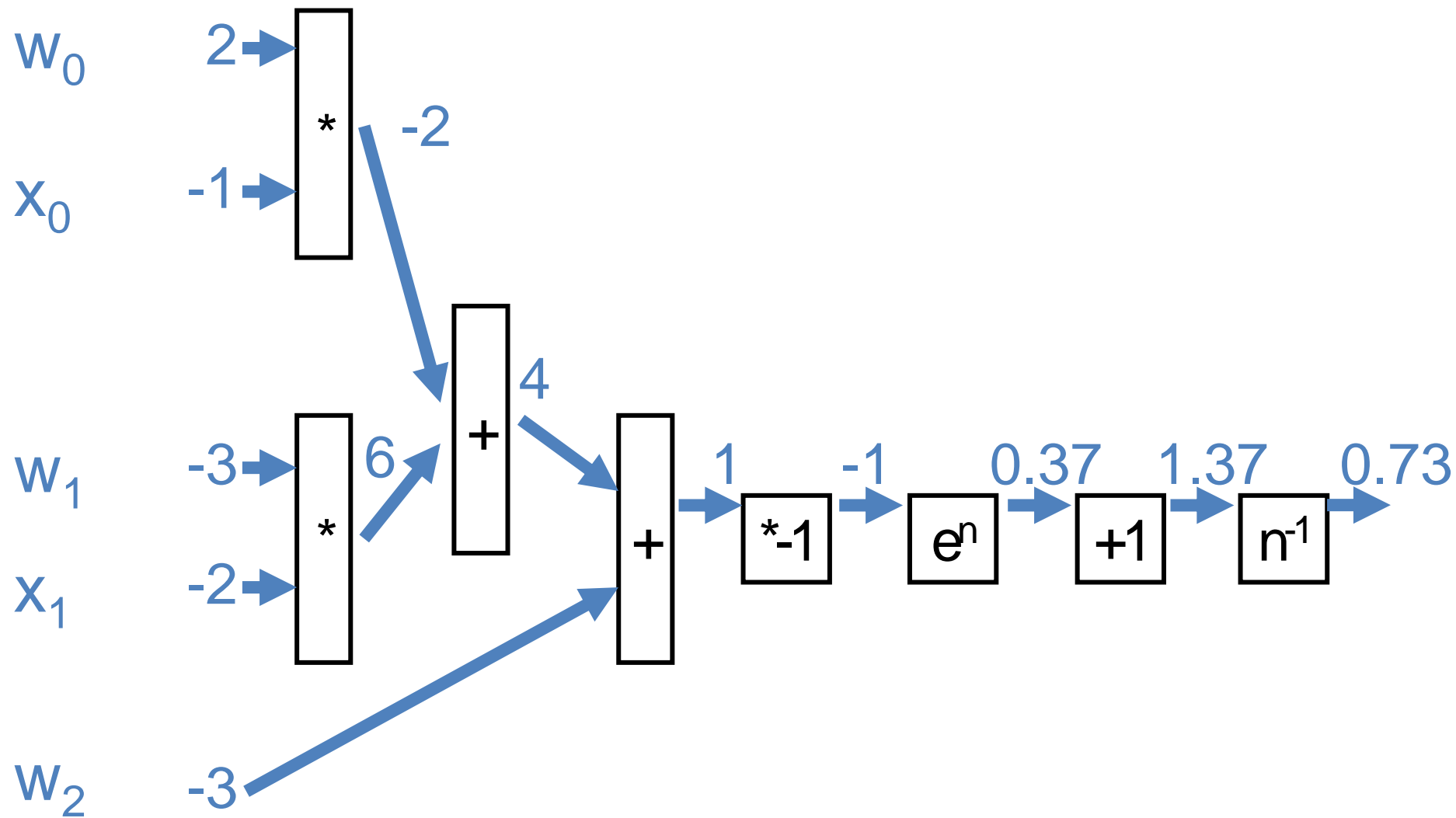
- The diagrams look complex but that's since we're covering the details together

Something More Complex

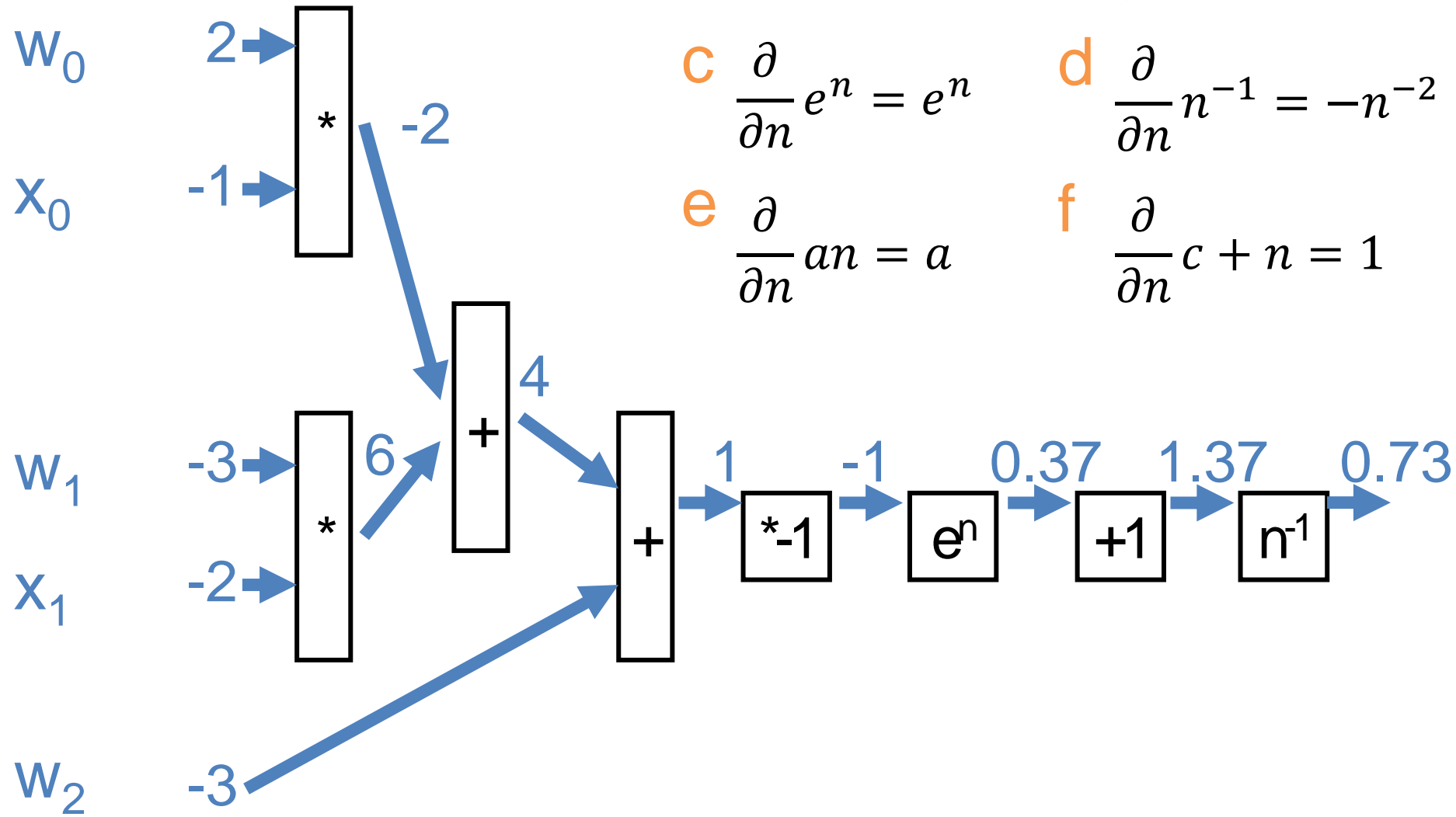
$$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



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



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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

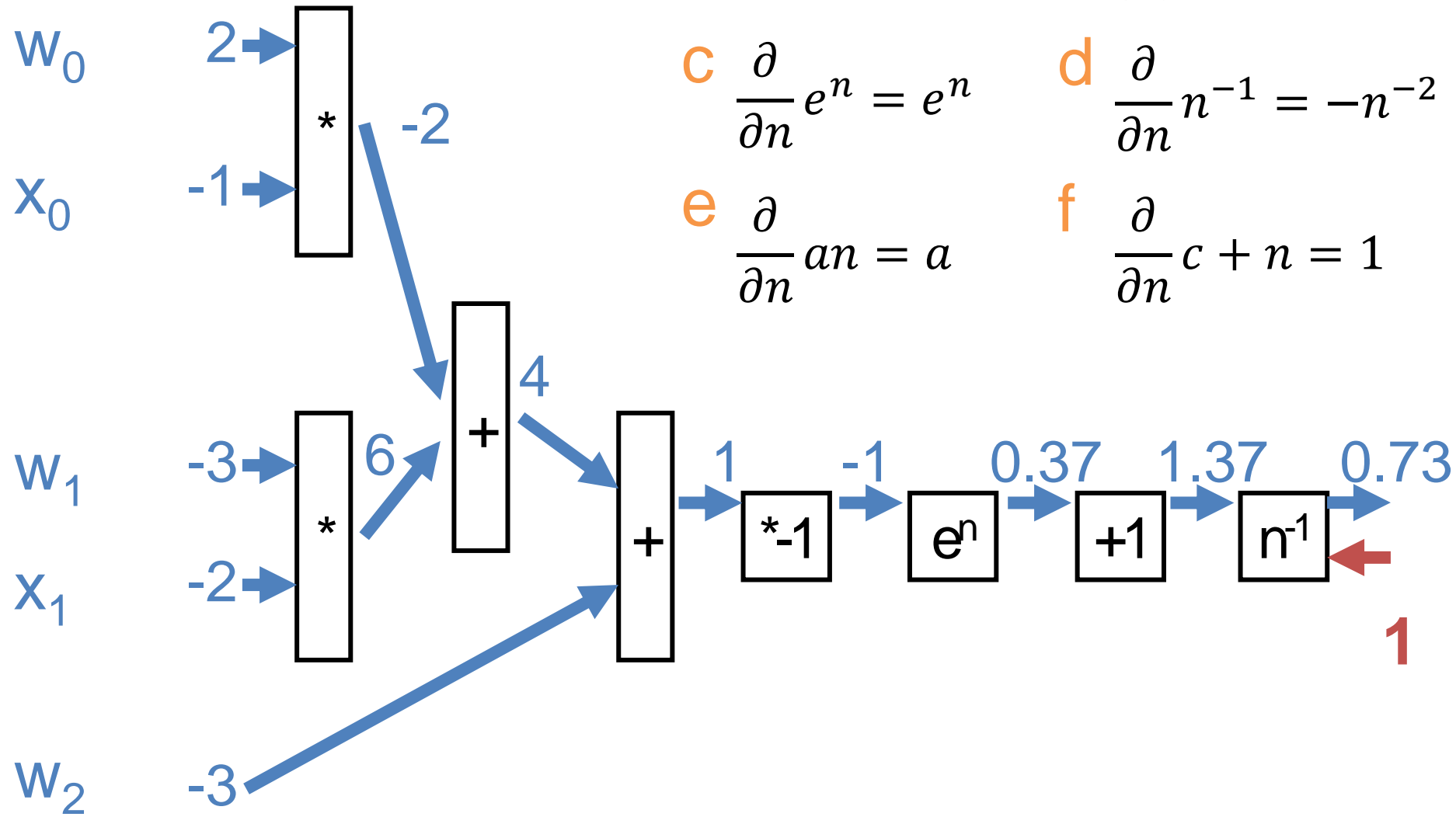
c $\frac{\partial}{\partial n} e^n = e^n$

d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

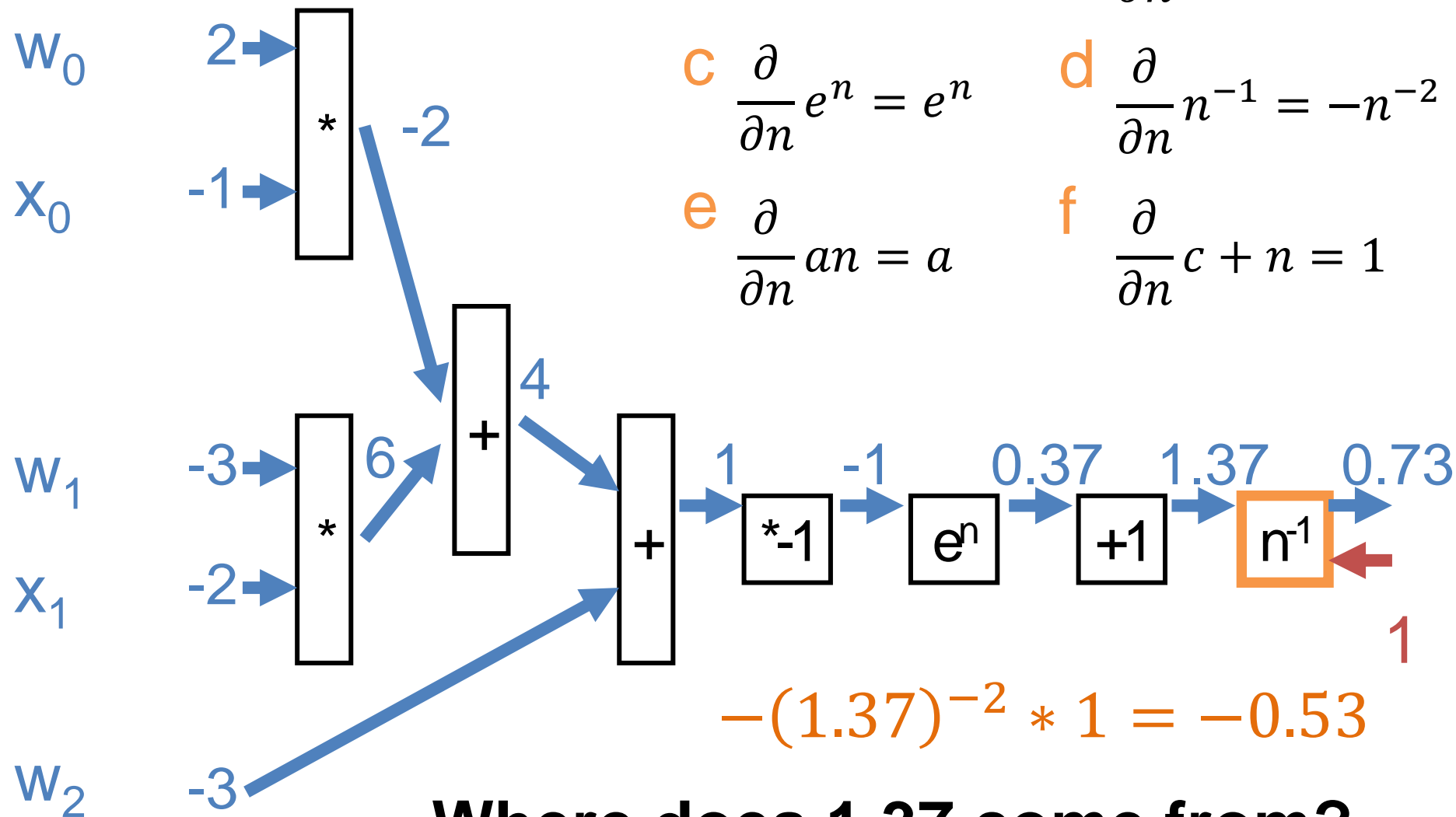
c $\frac{\partial}{\partial n} e^n = e^n$

d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

c $\frac{\partial}{\partial n} e^n = e^n$

d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

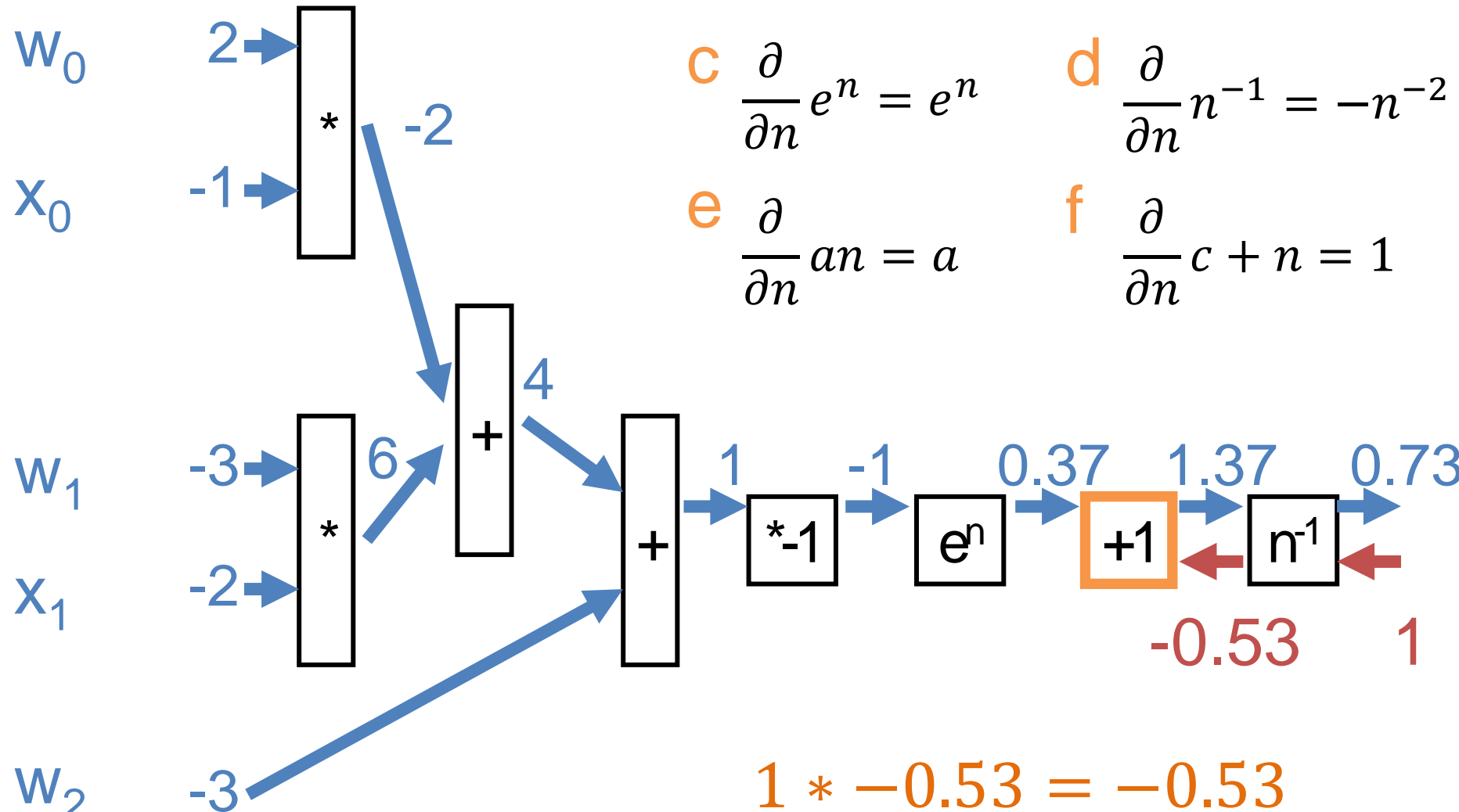
e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

$-(1.37)^{-2} * 1 = -0.53$

Where does 1.37 come from?

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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

c $\frac{\partial}{\partial n} e^n = e^n$

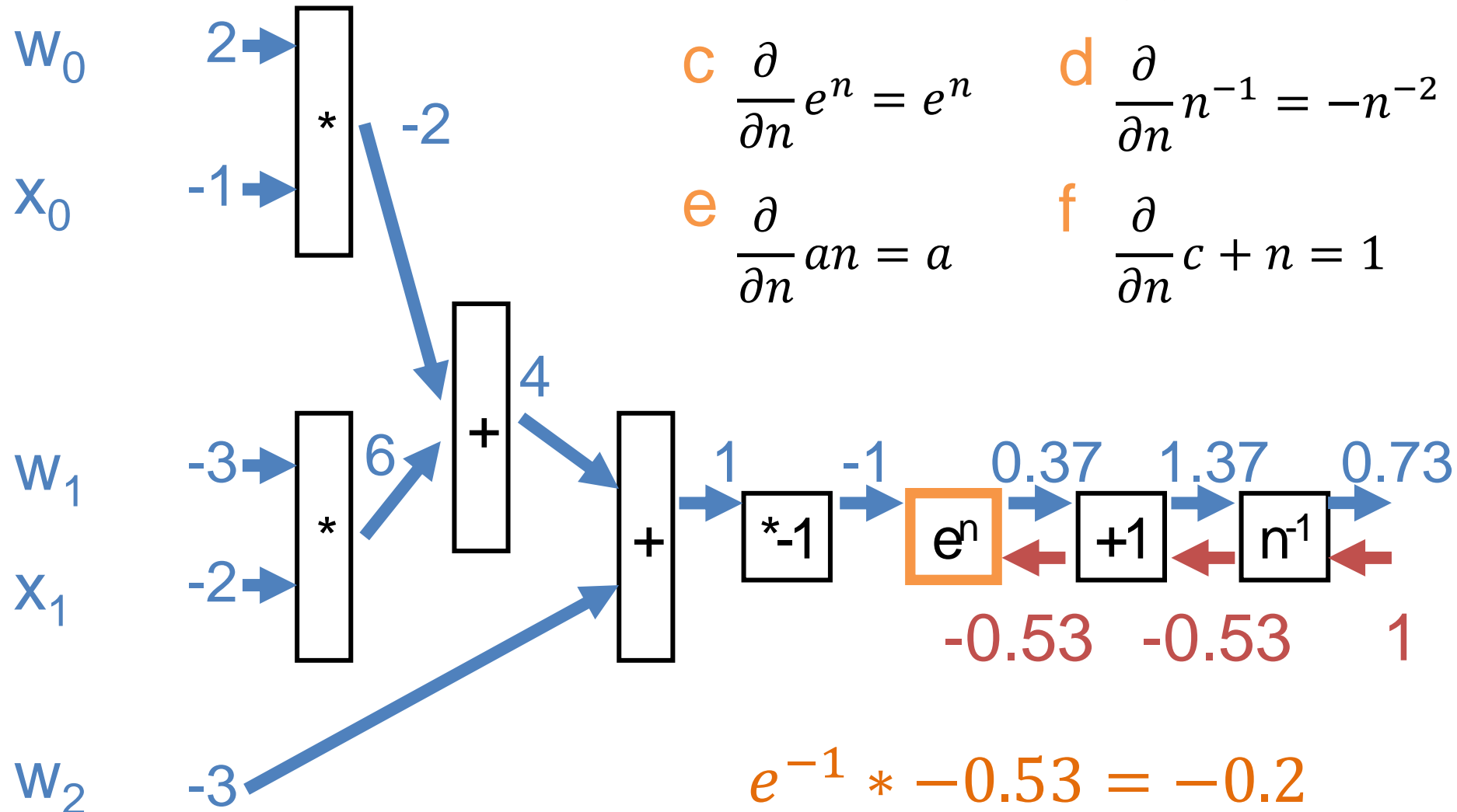
d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

$1 * -0.53 = -0.53$

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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

c $\frac{\partial}{\partial n} e^n = e^n$

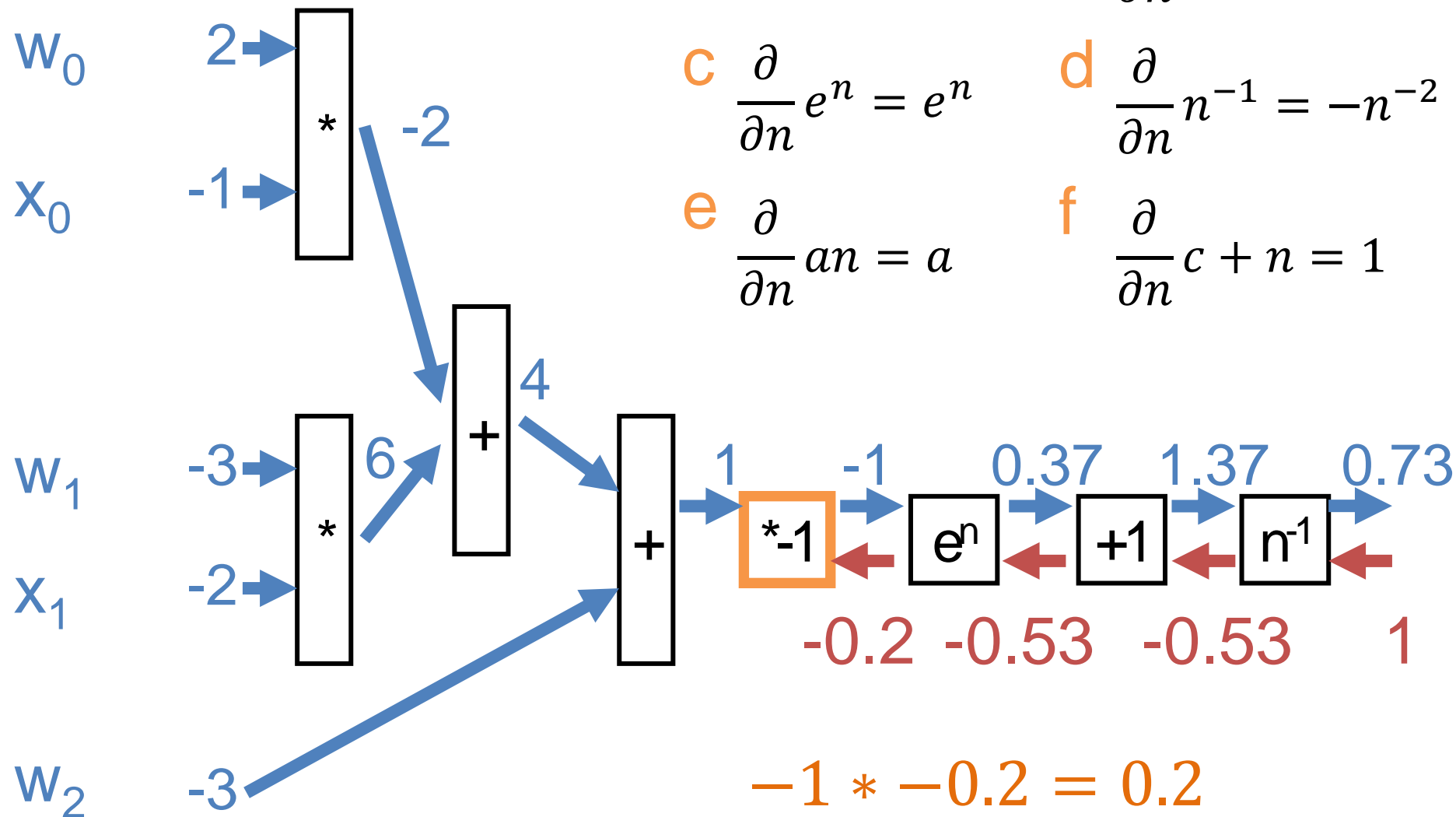
d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

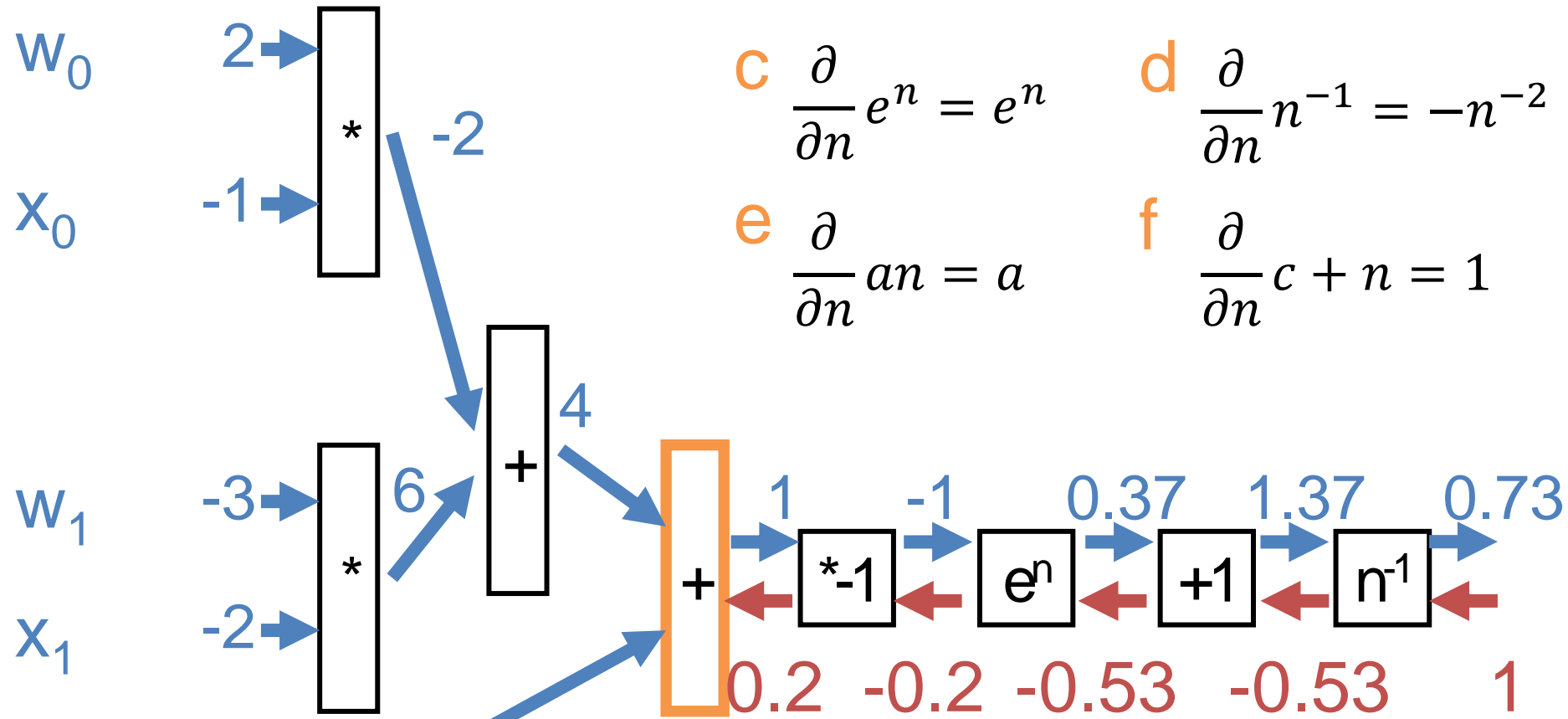
f $\frac{\partial}{\partial n} c + n = 1$

$e^{-1} * -0.53 = -0.2$

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



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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

c $\frac{\partial}{\partial n} e^n = e^n$

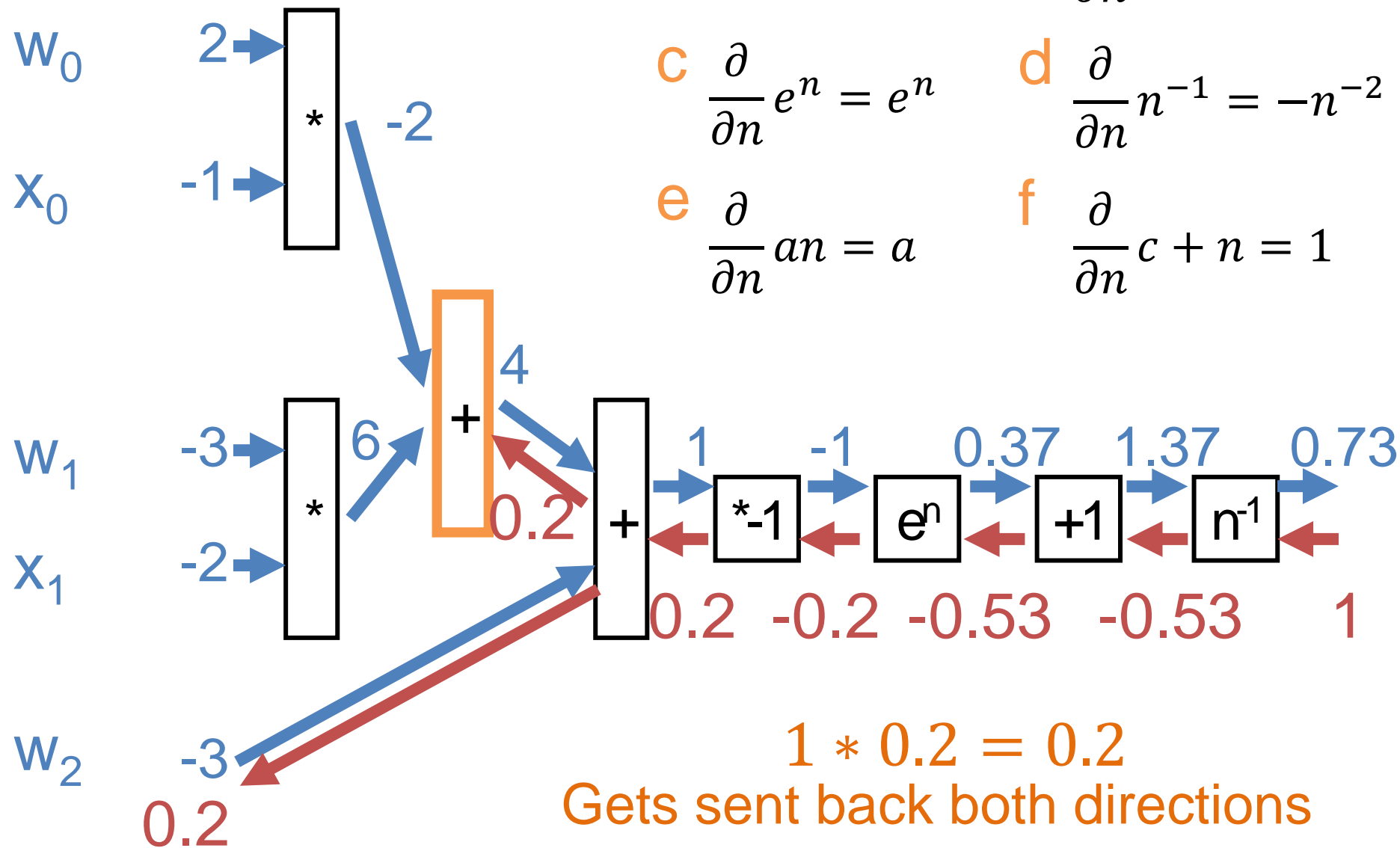
d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

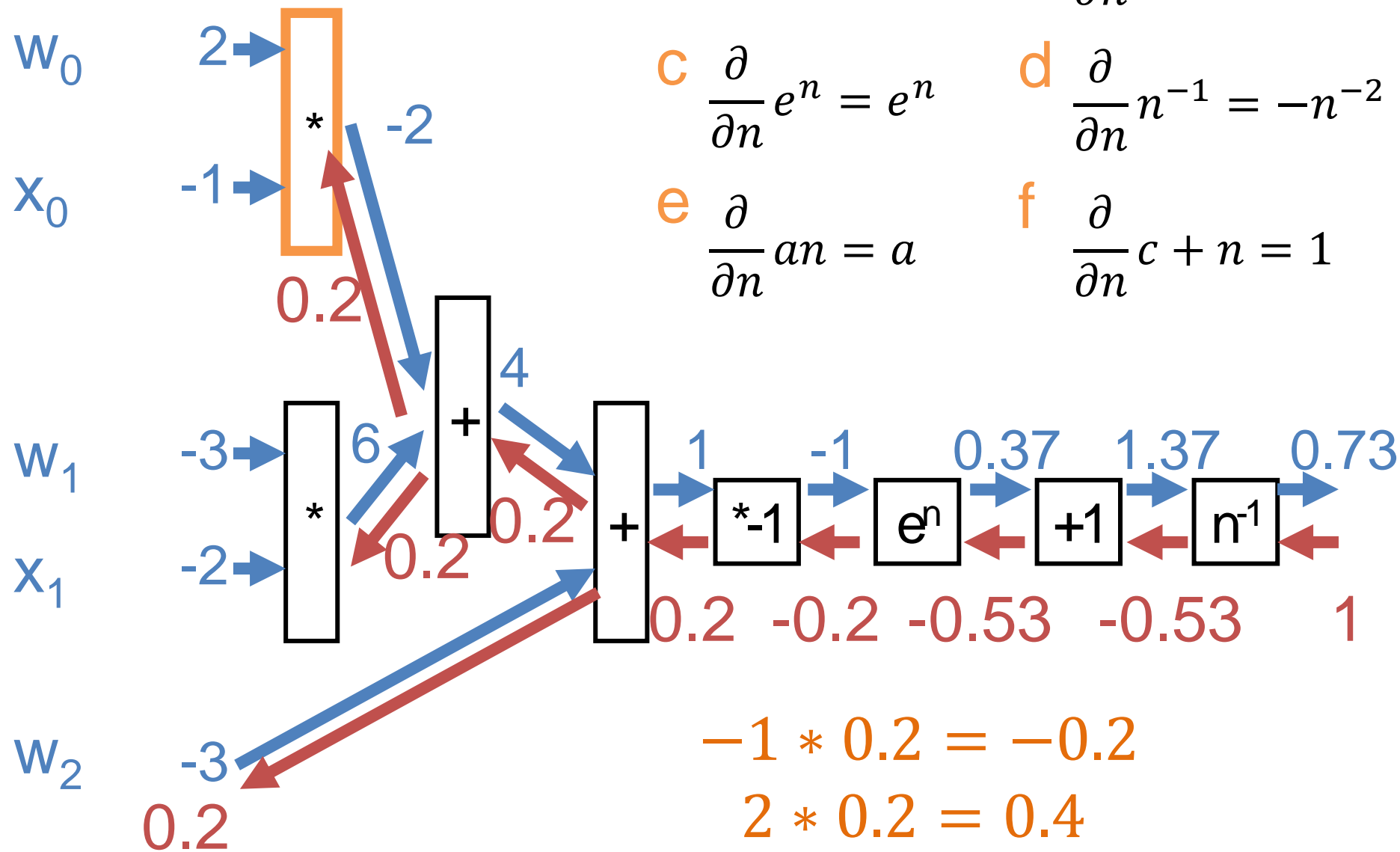
$1 * 0.2 = 0.2$
Gets sent back both directions

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

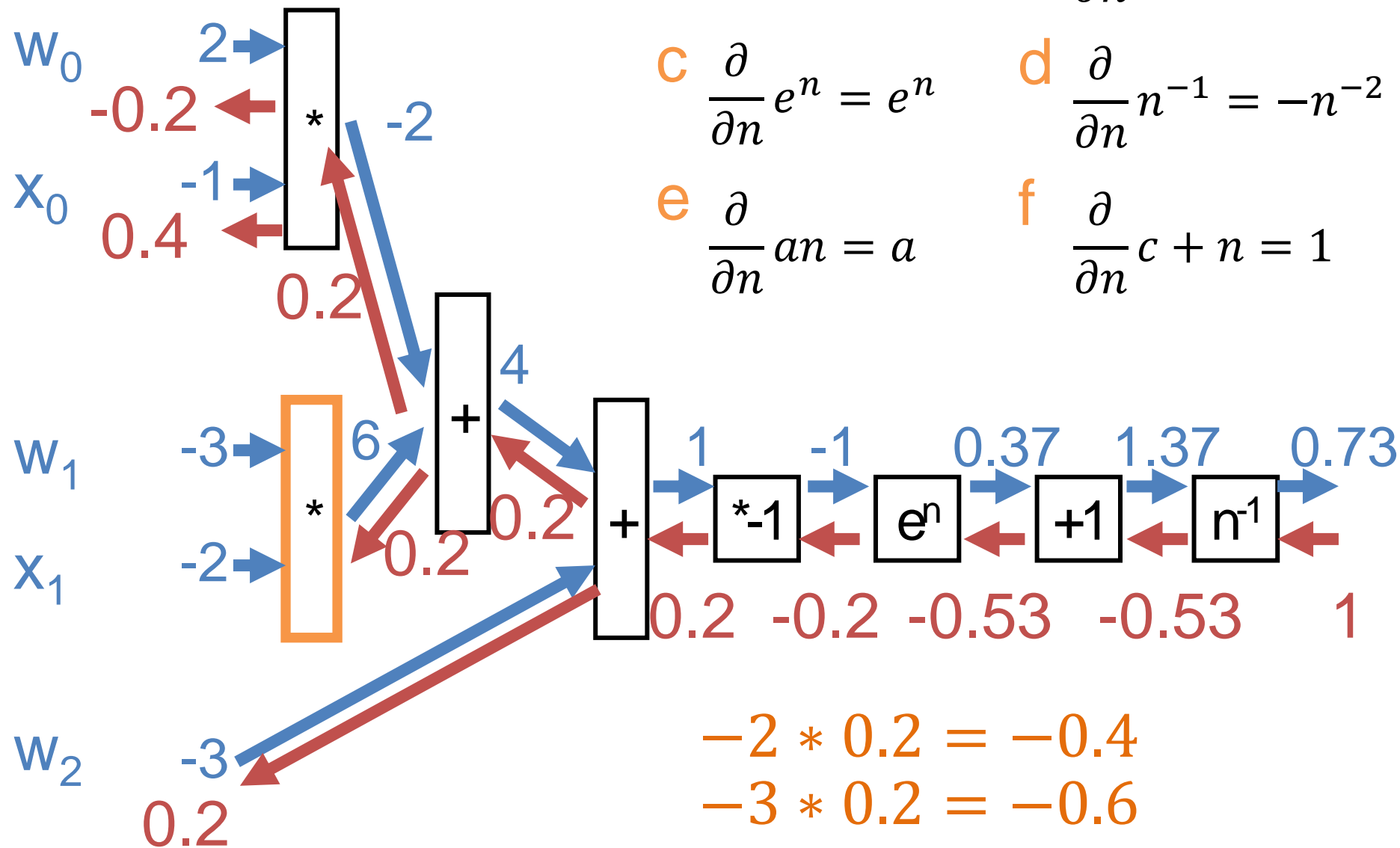


Example Credit: Karpathy and Fei-Fei

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

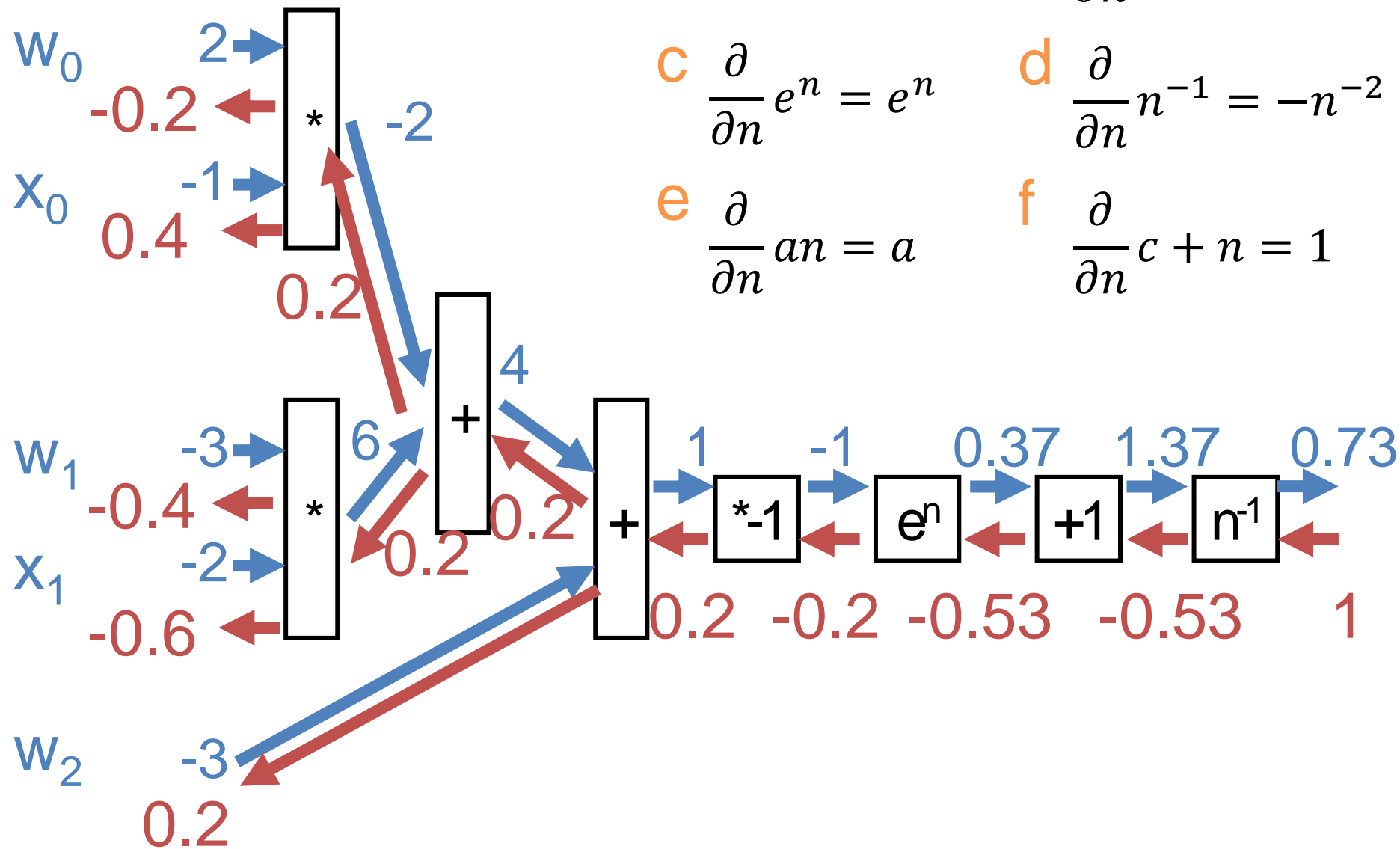


$$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Example Credit: Karpathy and Fei-Fei

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



a $\frac{\partial}{\partial n} m + n = 1$

b $\frac{\partial}{\partial n} mn = m$

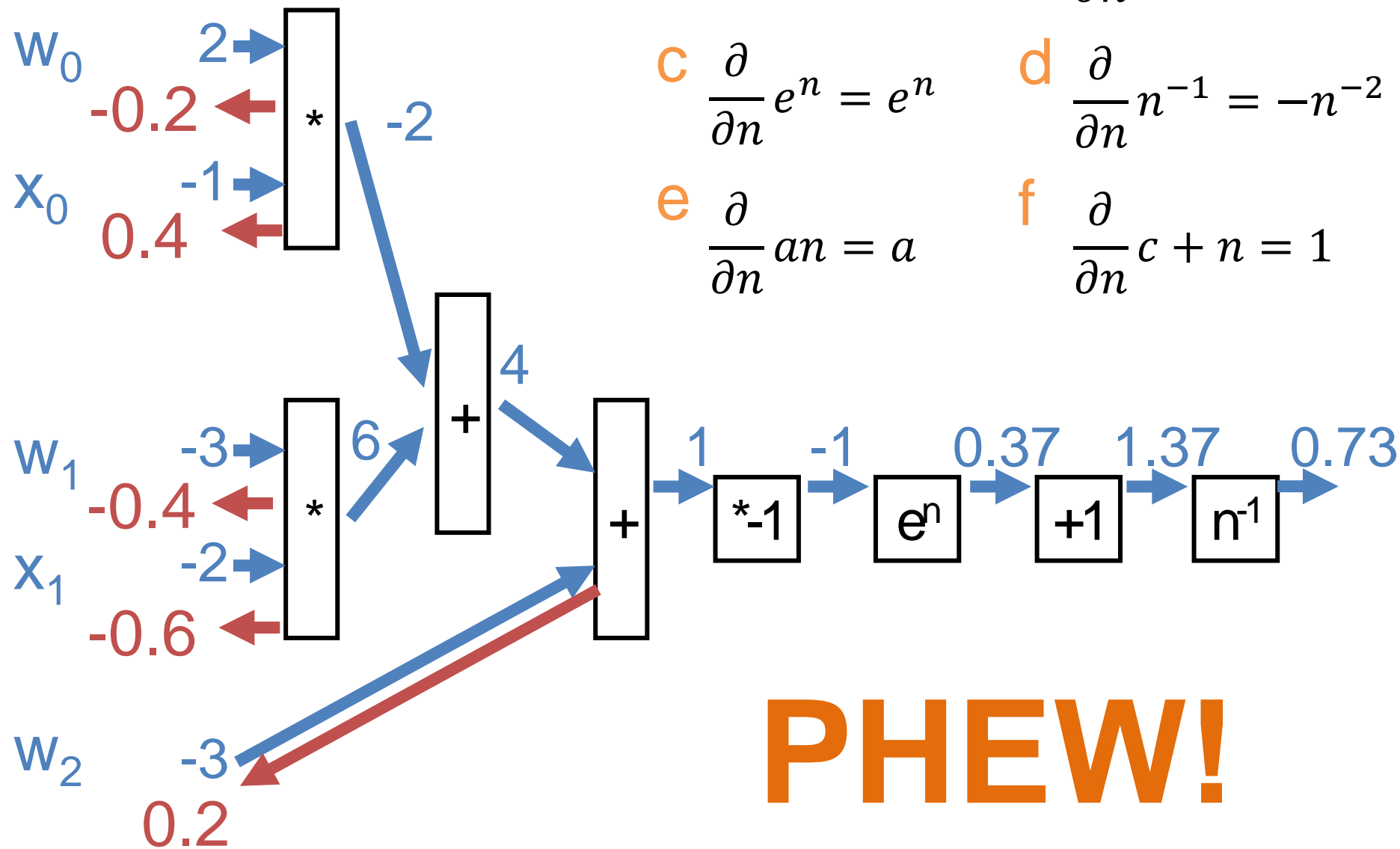
c $\frac{\partial}{\partial n} e^n = e^n$

d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$

f $\frac{\partial}{\partial n} c + n = 1$

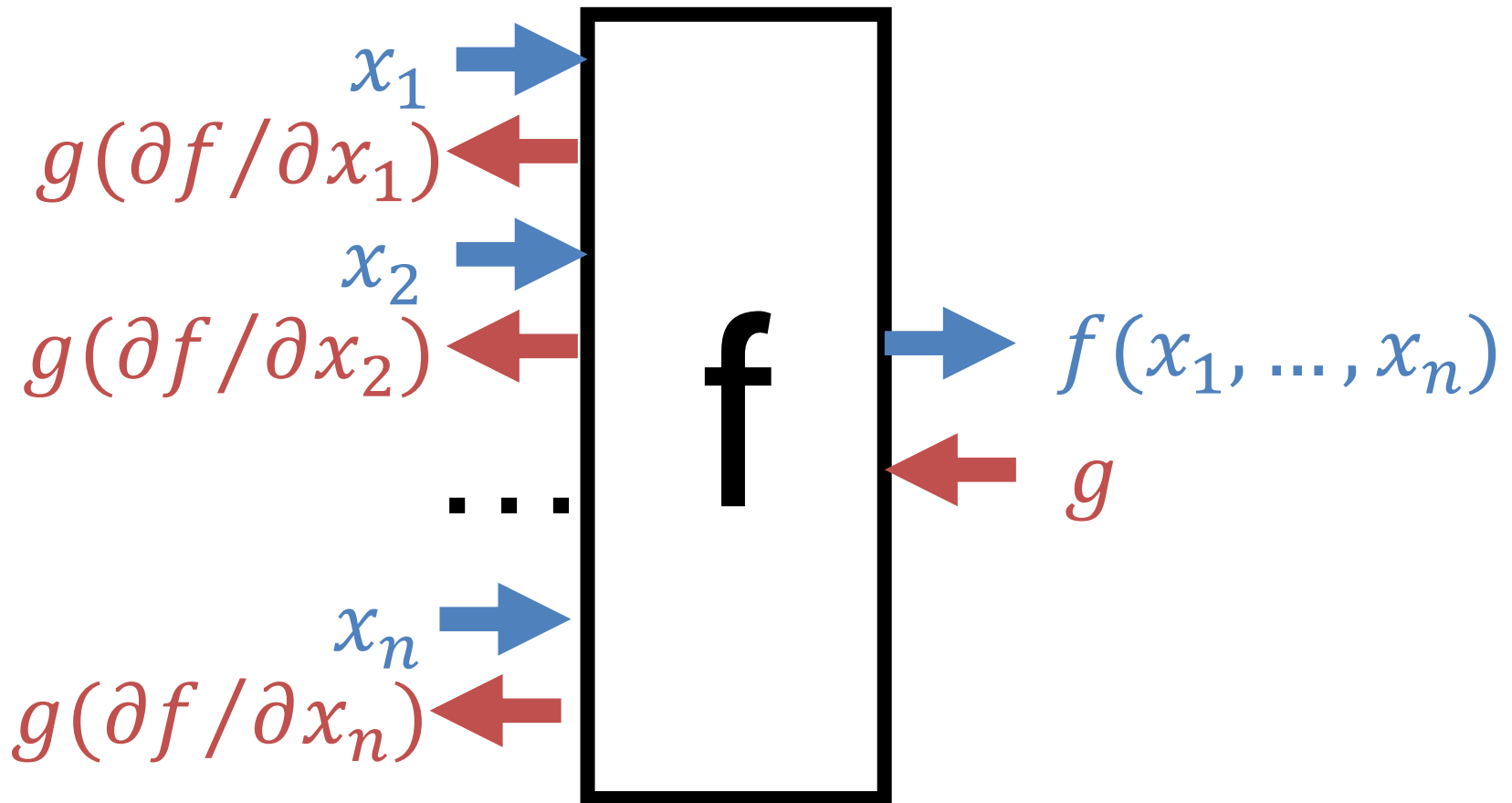
$$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Example Credit: Karpathy and Fei-Fei

Summary

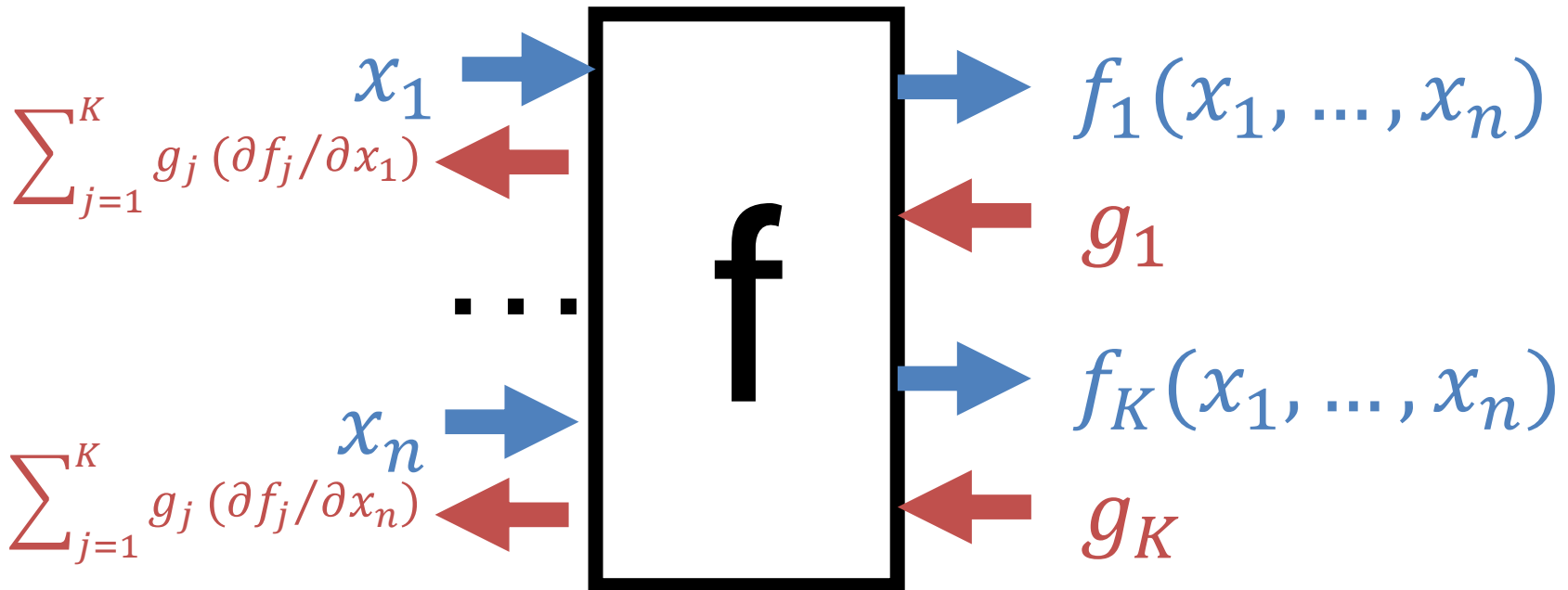
Each block computes backwards $(g) * \text{local gradient } (\partial f / \partial x_i)$ at the evaluation point



Multiple Outputs Flowing Back

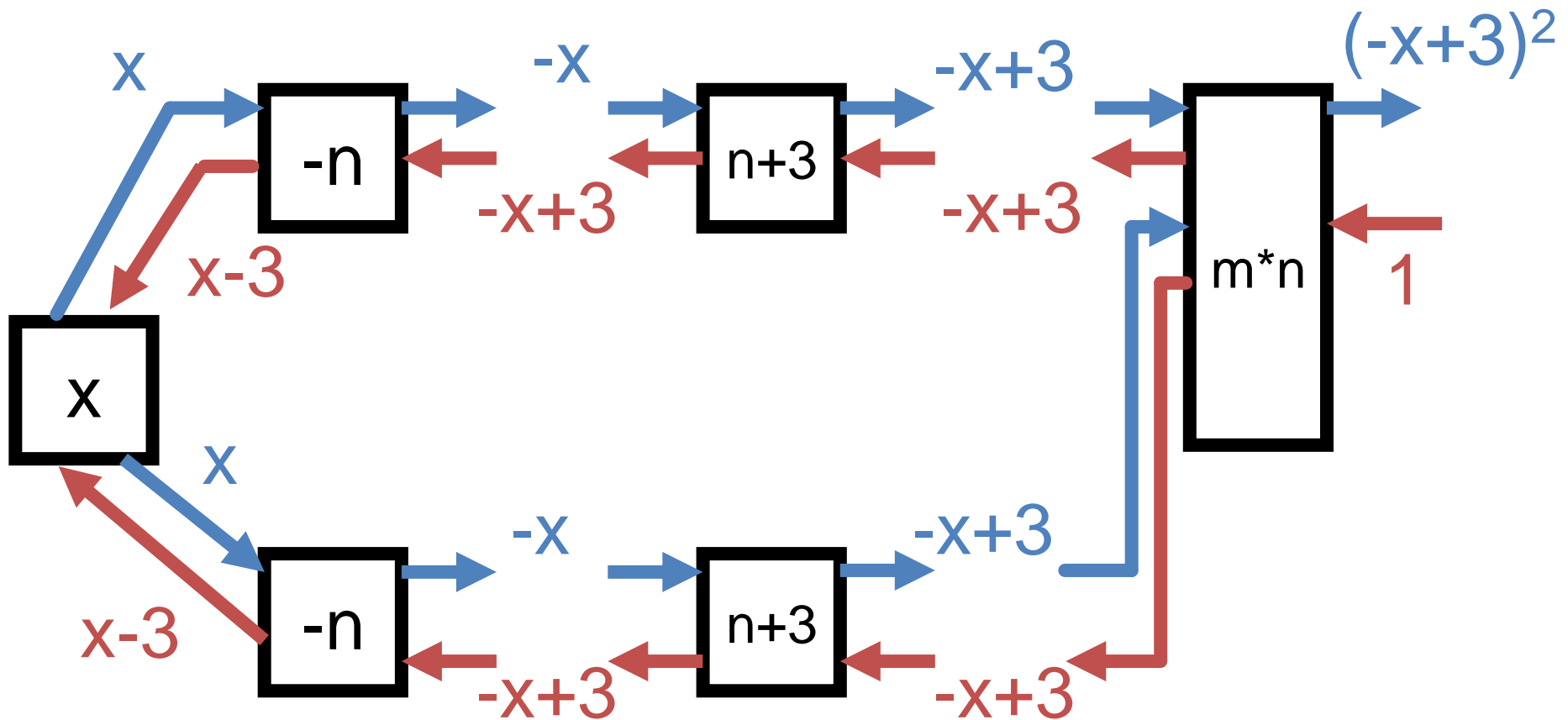
Gradients from different backwards sum up

$$\sum_{j=1}^K g_j (\partial f_j / \partial x_i)$$



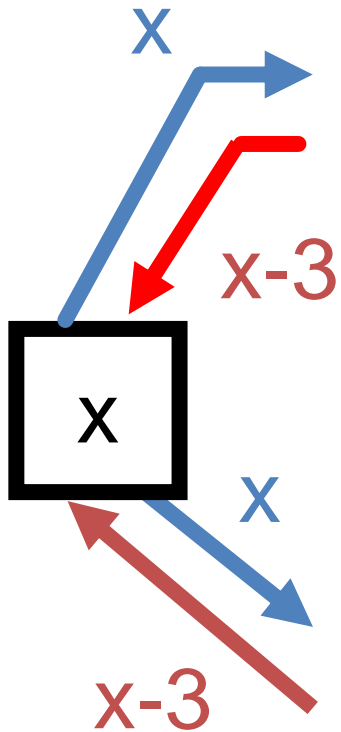
Multiple Outputs Flowing Back

$$f(x) = (-x + 3)^2$$



Multiple Outputs Flowing Back

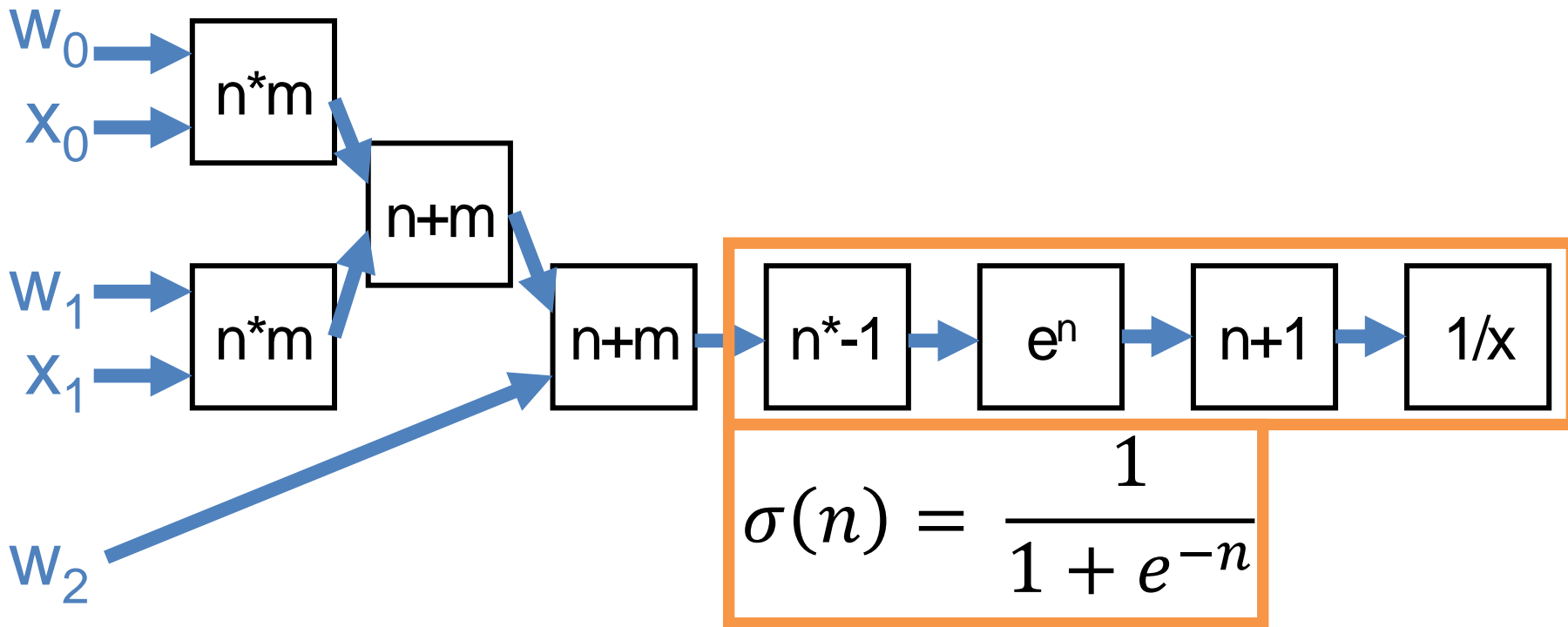
$$f(x) = (-x + 3)^2$$



$$\begin{aligned}\frac{\partial f}{\partial x} &= (x - 3) + (x - 3) \\ &= 2x - 6\end{aligned}$$

Does It Have To Be So Painful?

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



Does It Have To Be So Painful?

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

$$\frac{\partial}{\partial n} \sigma(n) = \frac{e^{-n}}{(1 + e^{-n})^2} = \left(\frac{1 + e^{-n} - 1}{1 + e^{-n}} \right) \left(\frac{1}{1 + e^{-n}} \right)$$

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

$$= (1 - \sigma(n))\sigma(n)$$

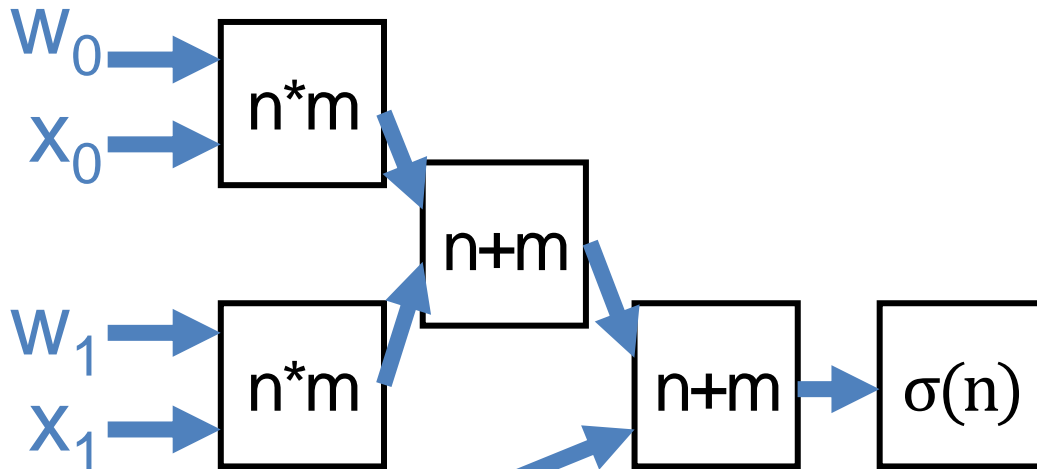
For the curious

Line 1 to 2: $\frac{\partial}{\partial n} \sigma(n) = \left(\frac{-1}{(1 + e^{-n})^2} \right) * 1 * e^{-n} * -1$

Chain rule: $\frac{d}{dx} (1/x) * \frac{d}{dx} (1+x) * \frac{d}{dx} (e^*x) * \frac{d}{dx} (-x)$

Does It Have To Be So Painful?

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



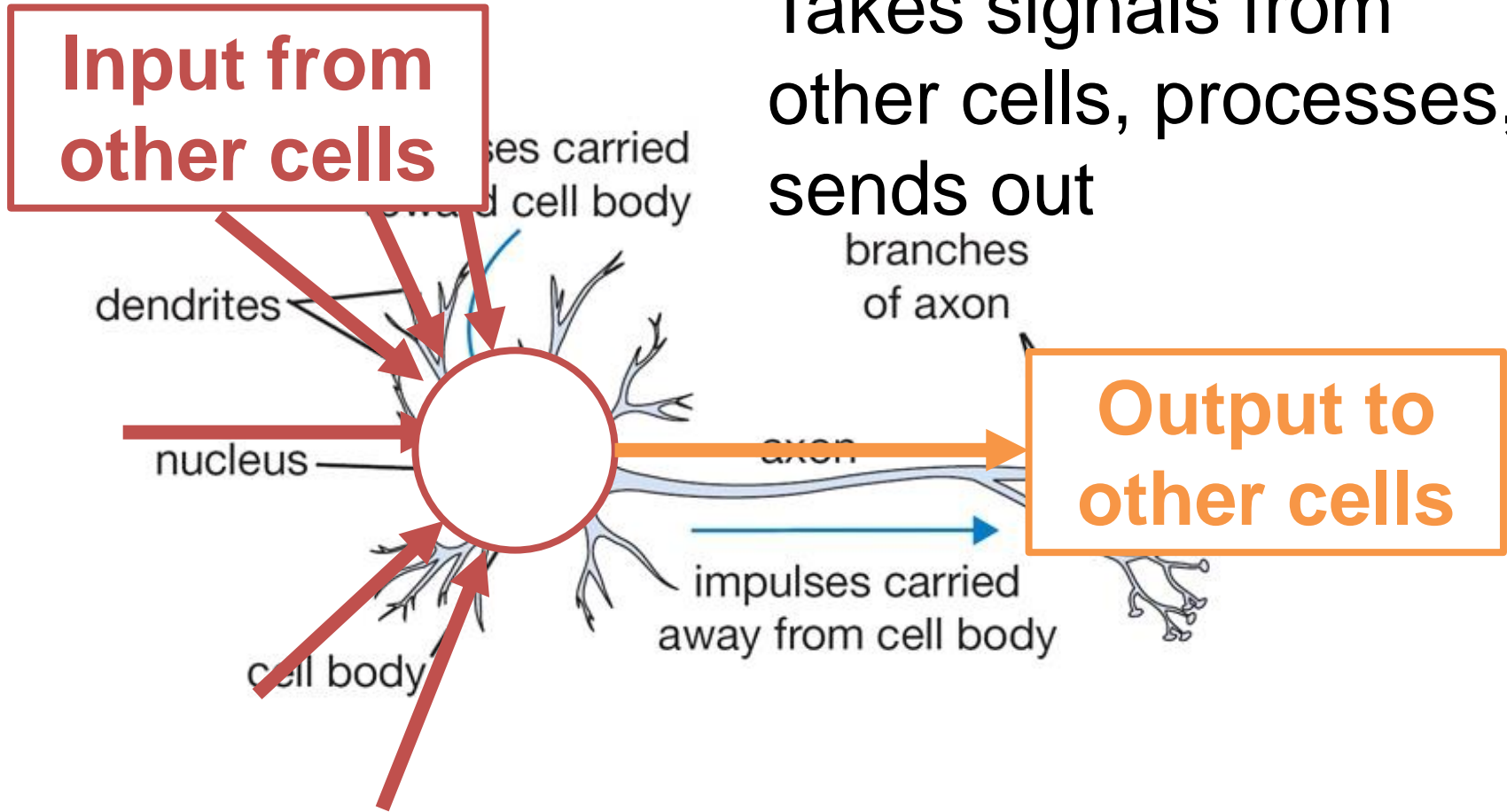
$$\sigma(n) = \frac{1}{1 + e^{-n}} \quad \frac{\partial \sigma(n)}{\partial n} = (1 - \sigma(n))\sigma(n)$$

Does It Have To Be So Painful?

- Can compute for any function
- Pick your functions carefully: existing code is usually structured into sensible blocks

Building Blocks

Takes signals from other cells, processes, sends out

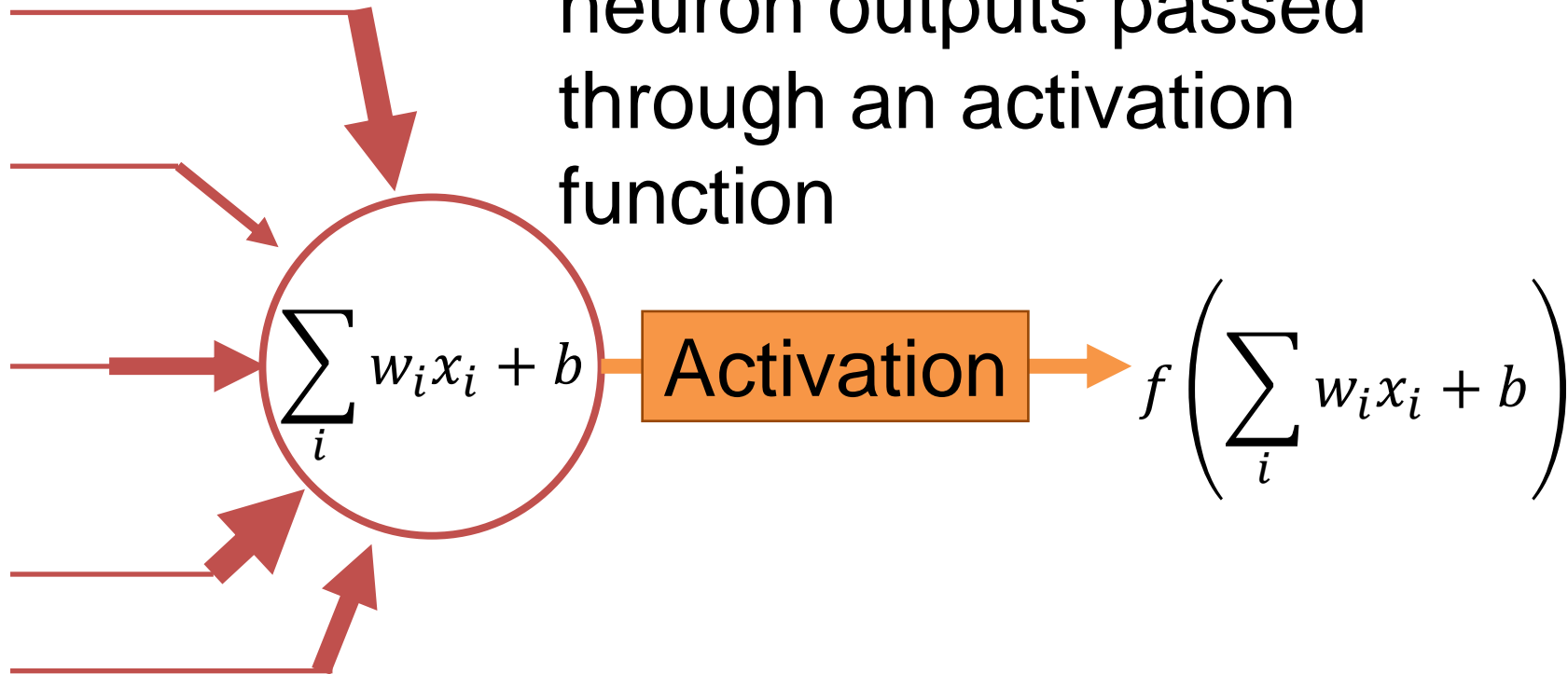


Input from other cells

Output to other cells

Artificial Neuron

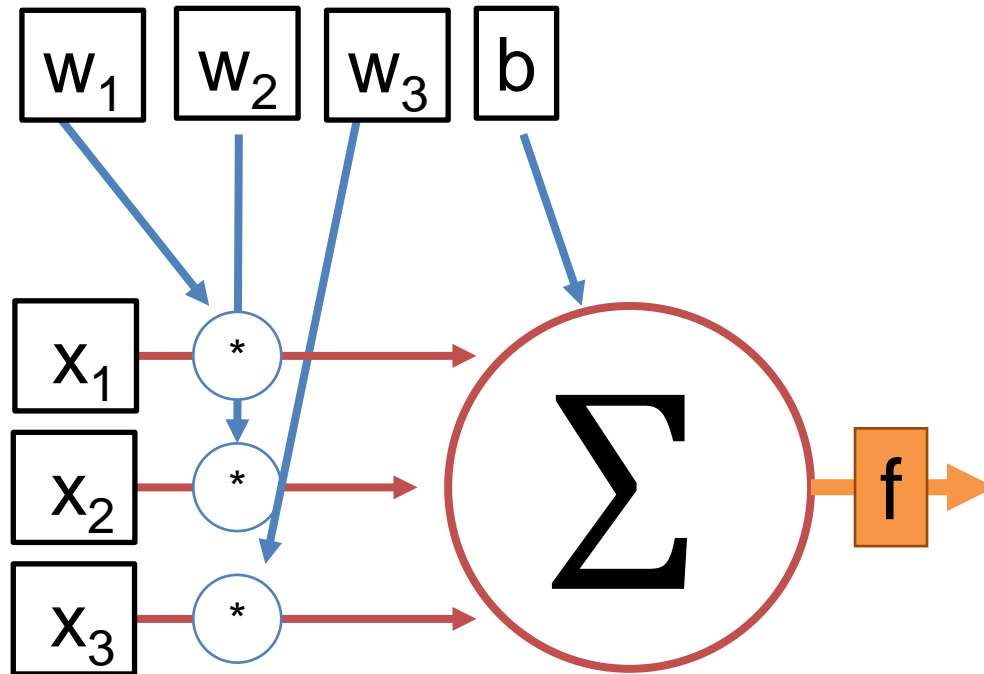
Weighted average of other neuron outputs passed through an activation function



Artificial Neuron

Can differentiate whole thing e.g., $d\text{Neuron}/dx_1$.

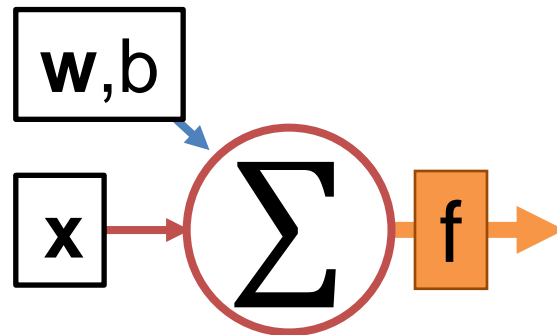
What can we now do?



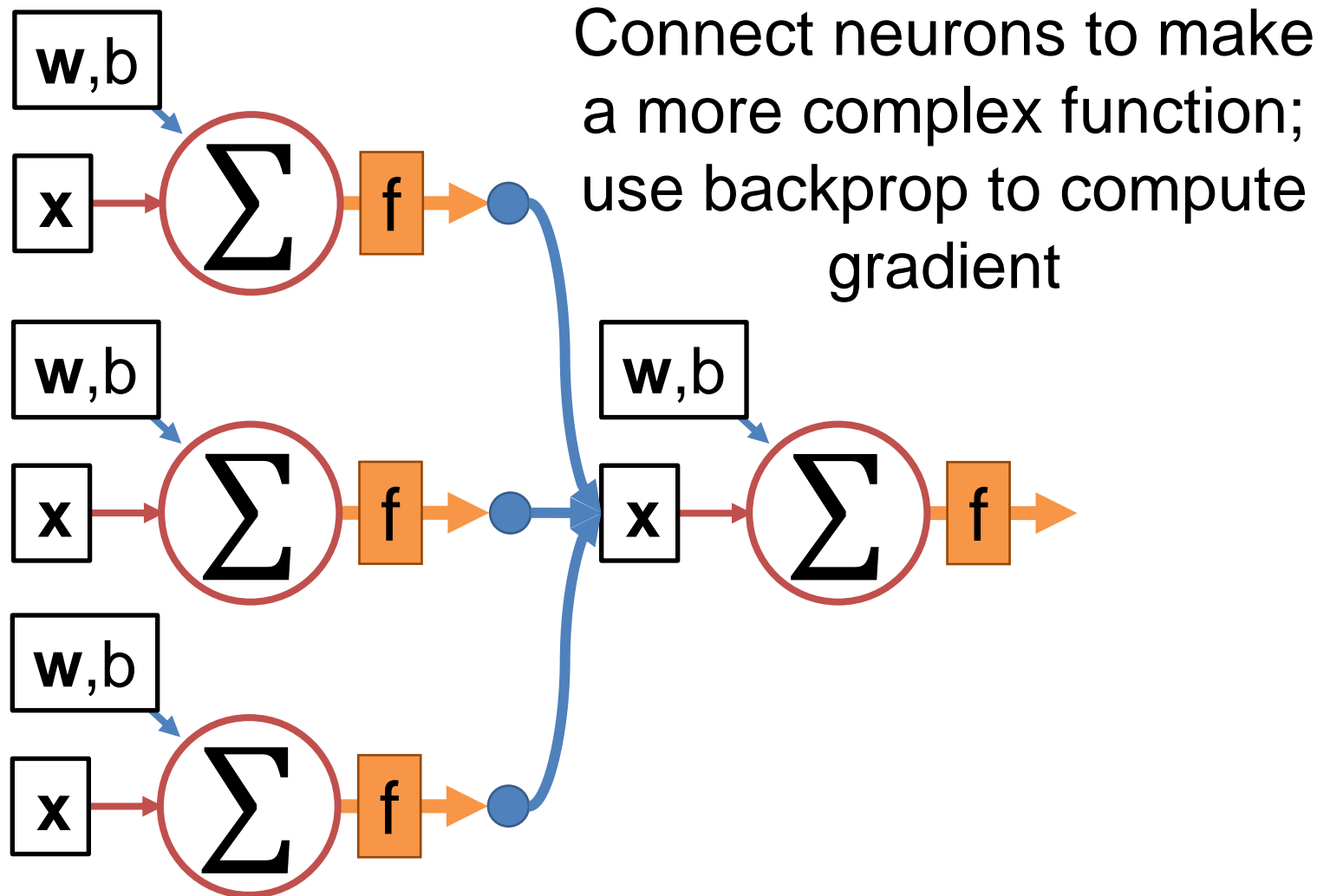
Artificial Neuron

Each artificial neuron is a linear model +
an **activation function** f

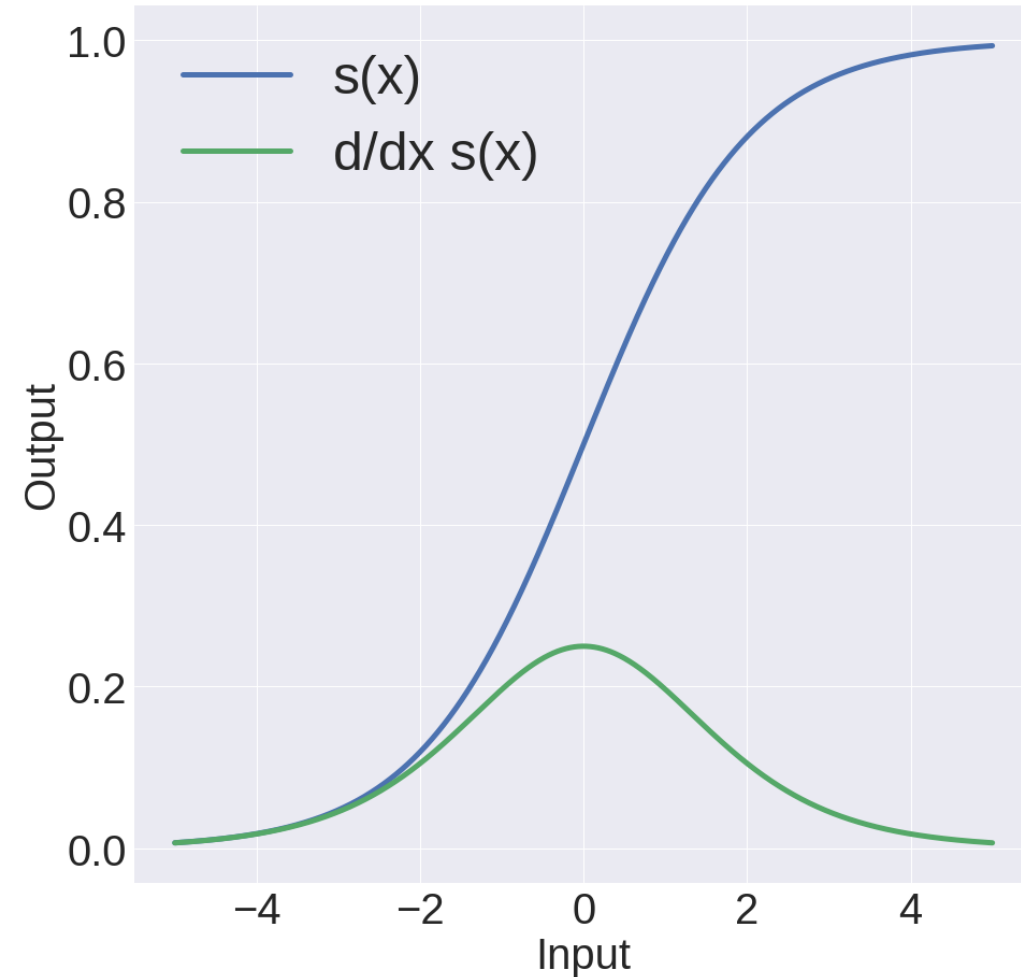
Can find \mathbf{w} , b that minimizes a loss
function with gradient descent



Artificial Neurons



What's The Activation Function

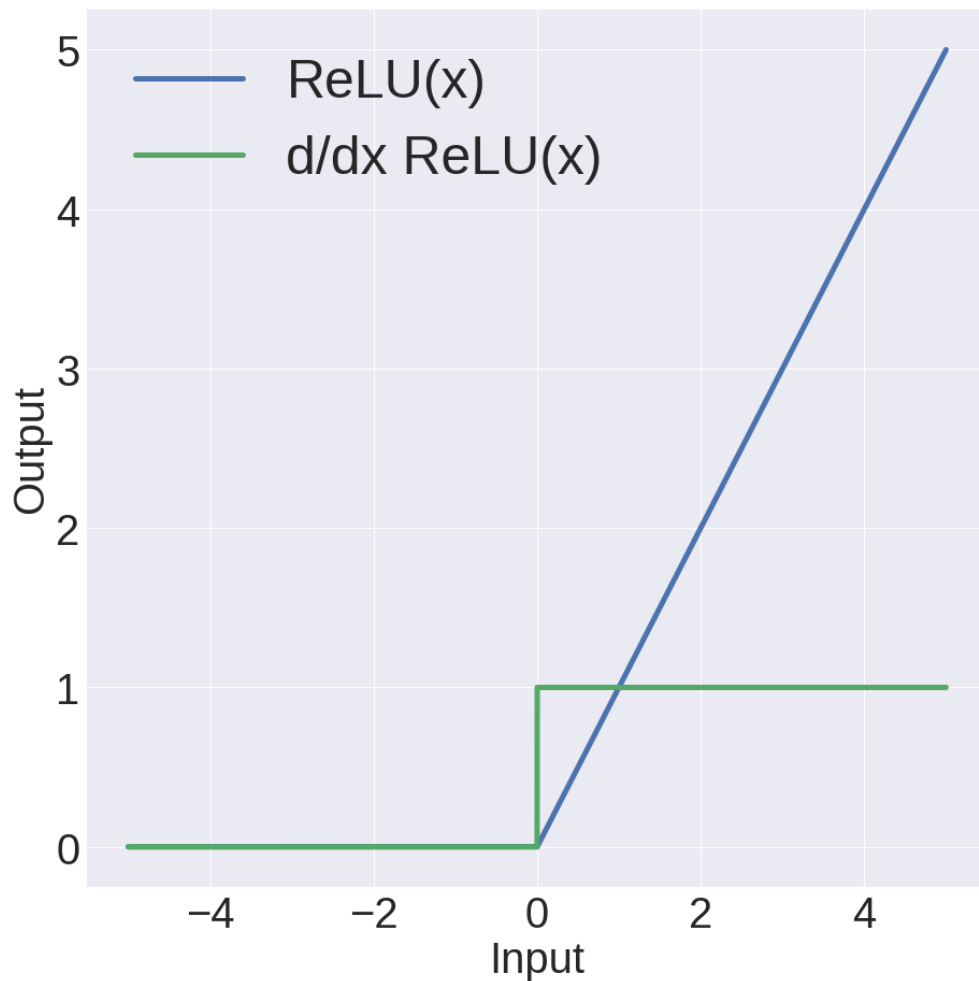


Sigmoid

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

- Nice interpretation
- Squashes things to (0,1)
- Gradients are near zero if neuron is high/low

What's The Activation Function



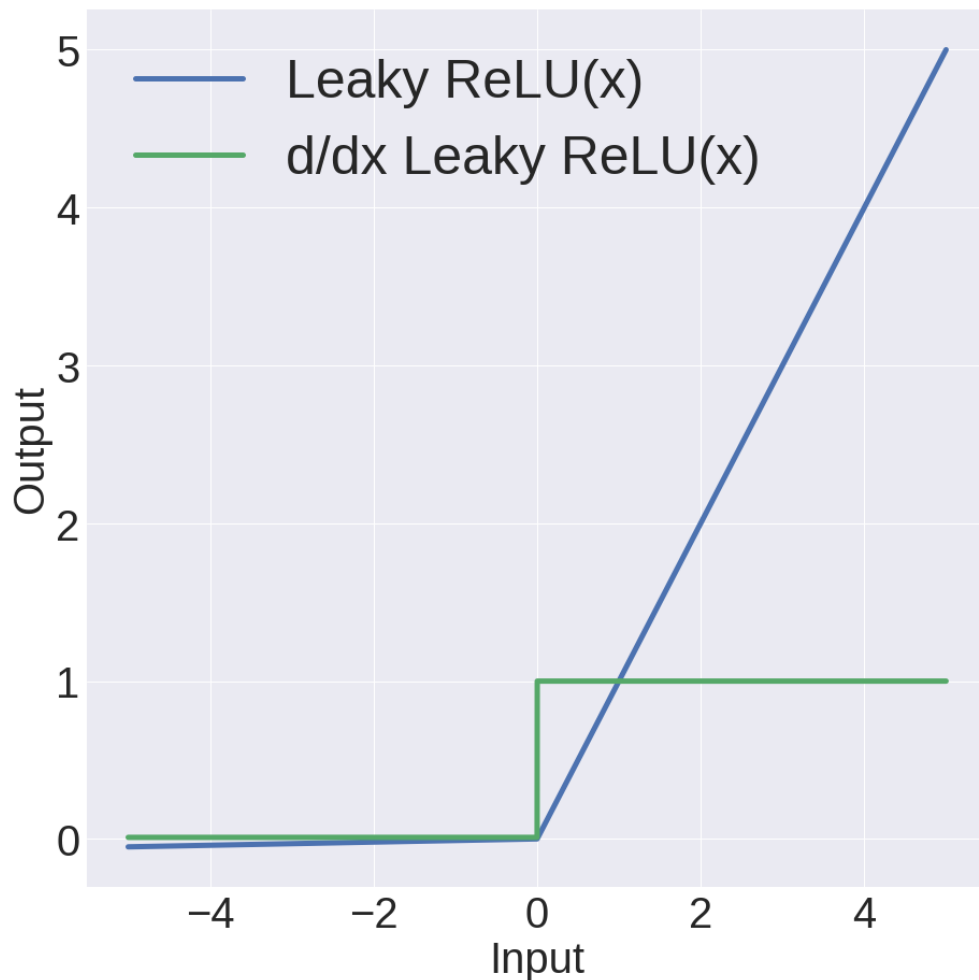
ReLU

(Rectifying Linear Unit)

$$\max(0, x)$$

- Constant gradient
- Converges $\sim 6x$ faster
- If neuron negative, zero gradient. Be careful!

What's The Activation Function



Leaky ReLU
(Rectifying Linear Unit)

$$x : x \geq 0$$

$$0.01x : x < 0$$

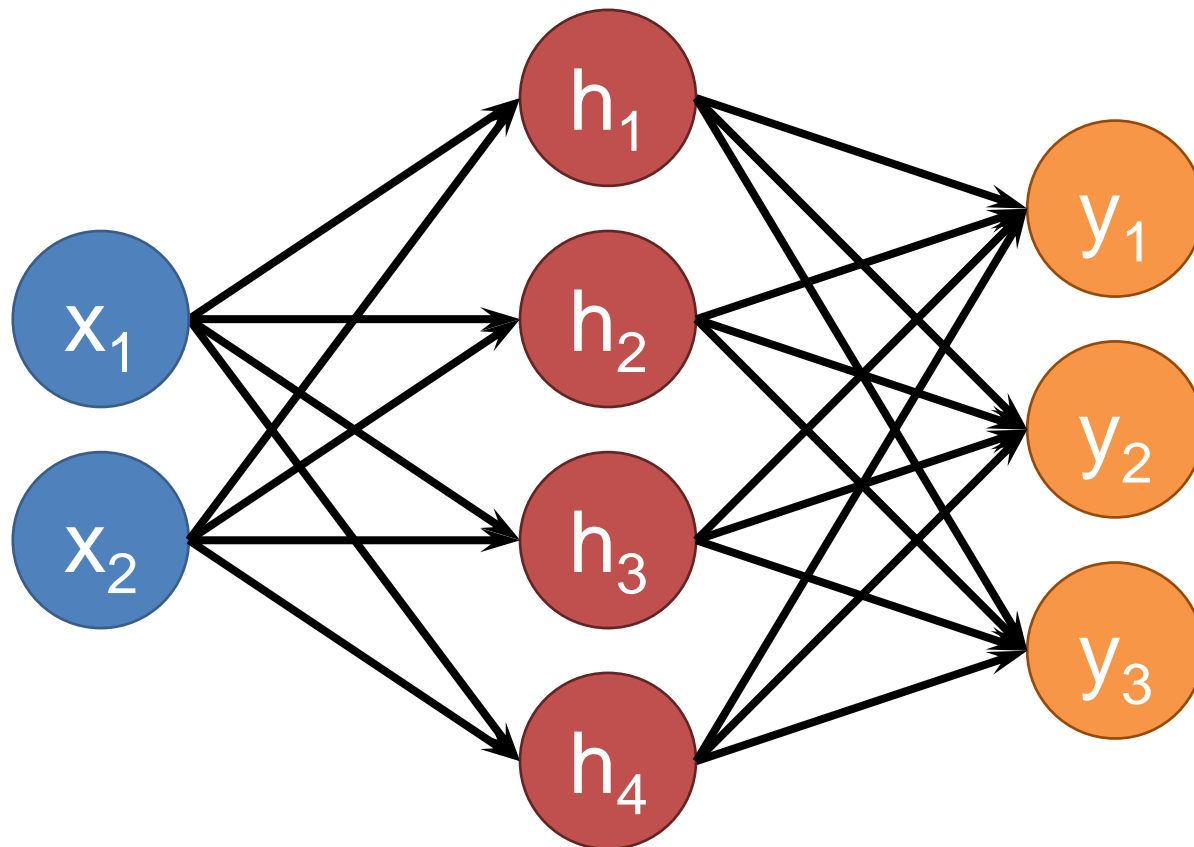
- ReLU, but allows some small gradient for negative values

Setting Up A Neural Net

Input

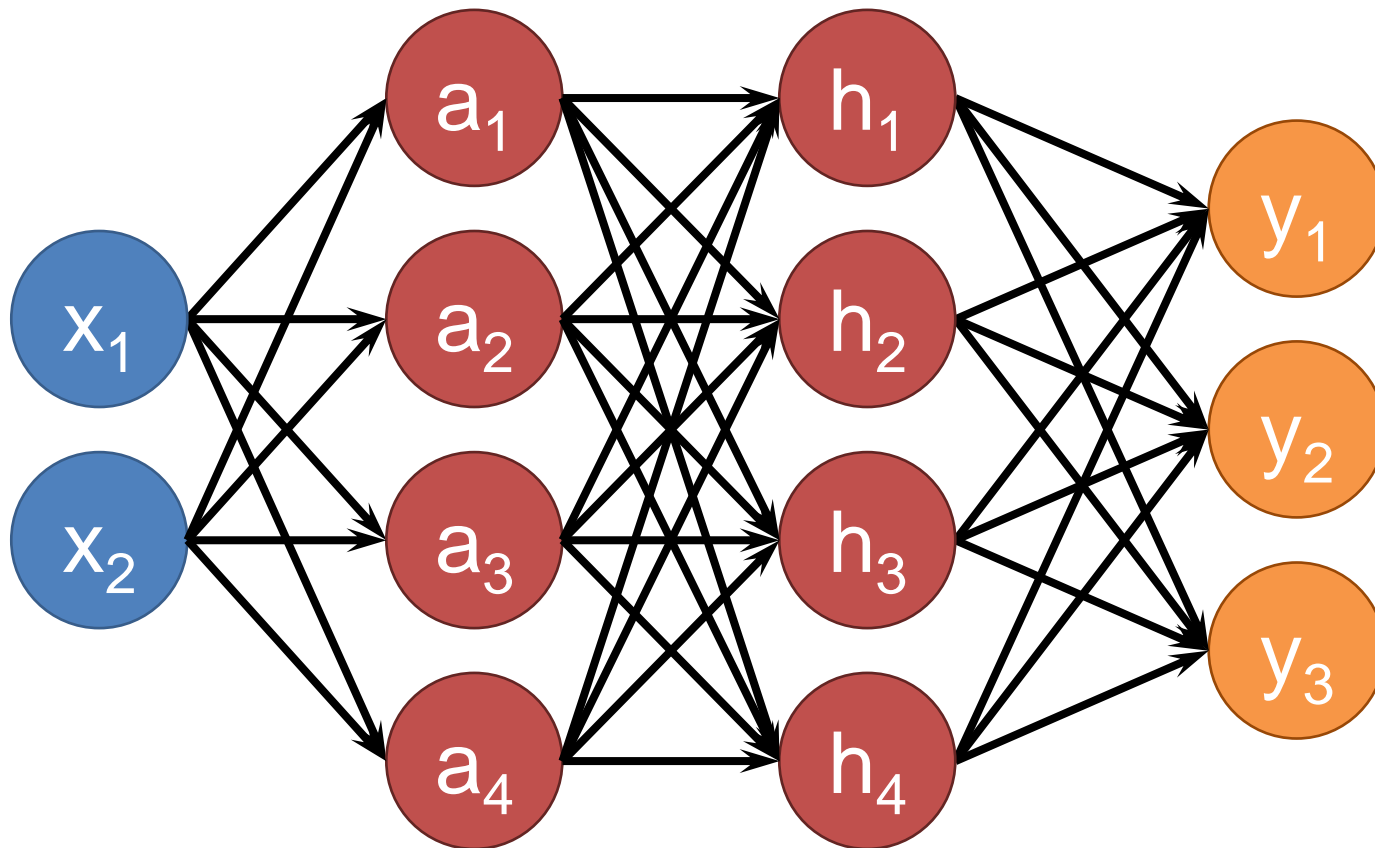
Hidden

Output

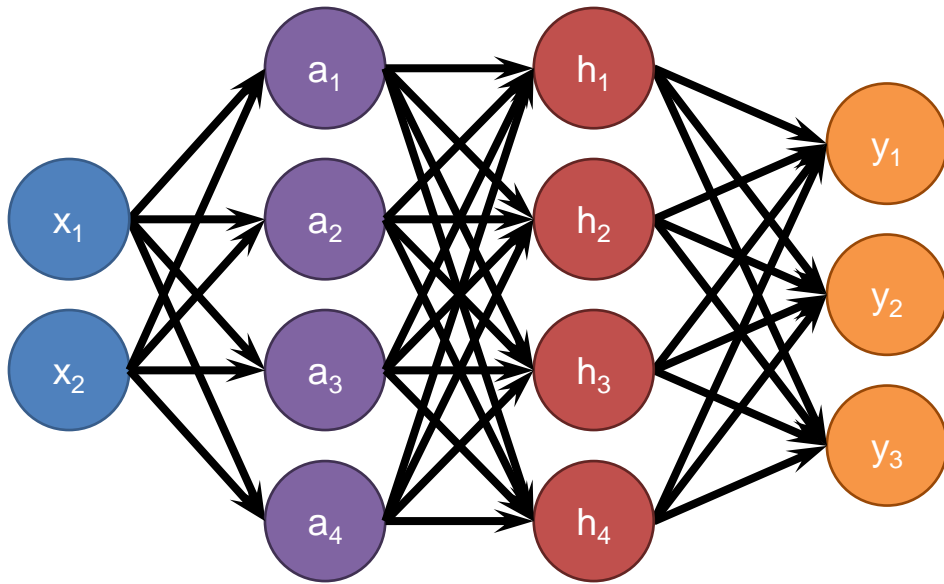


Setting Up A Neural Net

Input Hidden 1 Hidden 2 Output

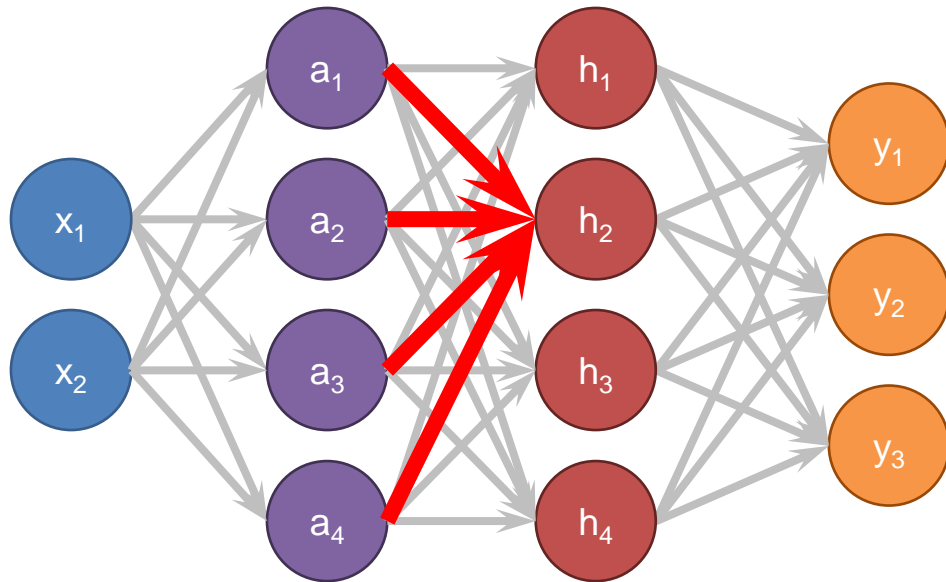


Fully Connected Network



Each neuron connects to each neuron in the previous layer

Fully Connected Network

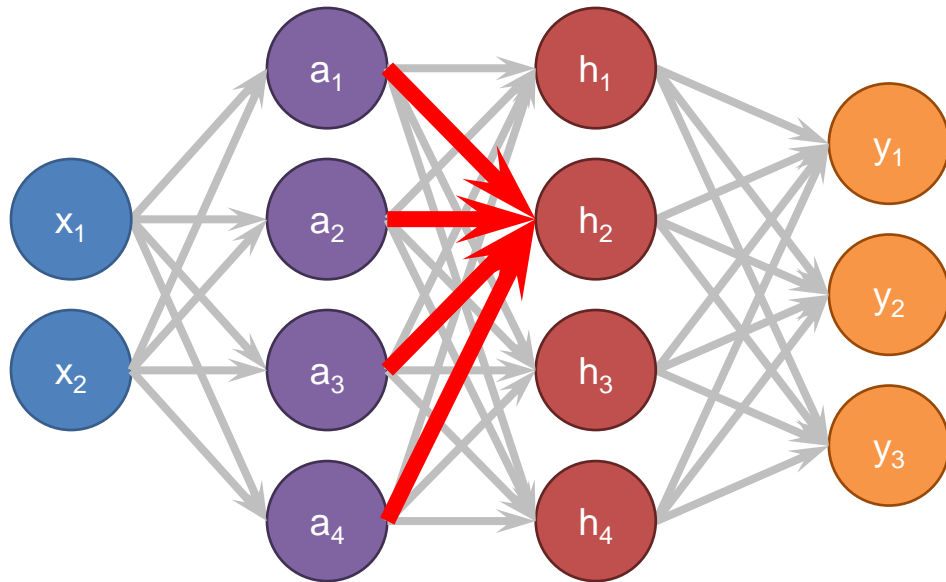


\mathbf{a} All layer a values
 \mathbf{w}_i, b_i Neuron i weights, bias
 f Activation function

$$h_i = f(\mathbf{w}_i^T \mathbf{a} + b_i)$$

How do we do all the neurons all at once?

Fully Connected Network



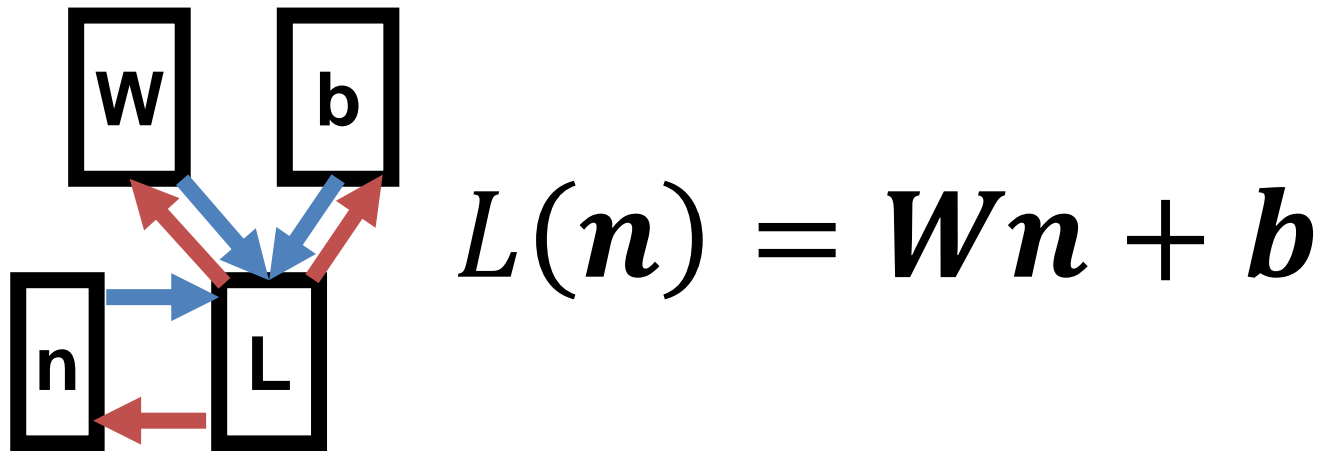
\mathbf{a} All layer a values
 \mathbf{w}_i, b_i Neuron i weights, bias
 f Activation function

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = f \left(\begin{bmatrix} \leftarrow w_1^T \rightarrow \\ \leftarrow w_2^T \rightarrow \\ \leftarrow w_3^T \rightarrow \\ \leftarrow w_4^T \rightarrow \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \right)$$

Fully Connected Network

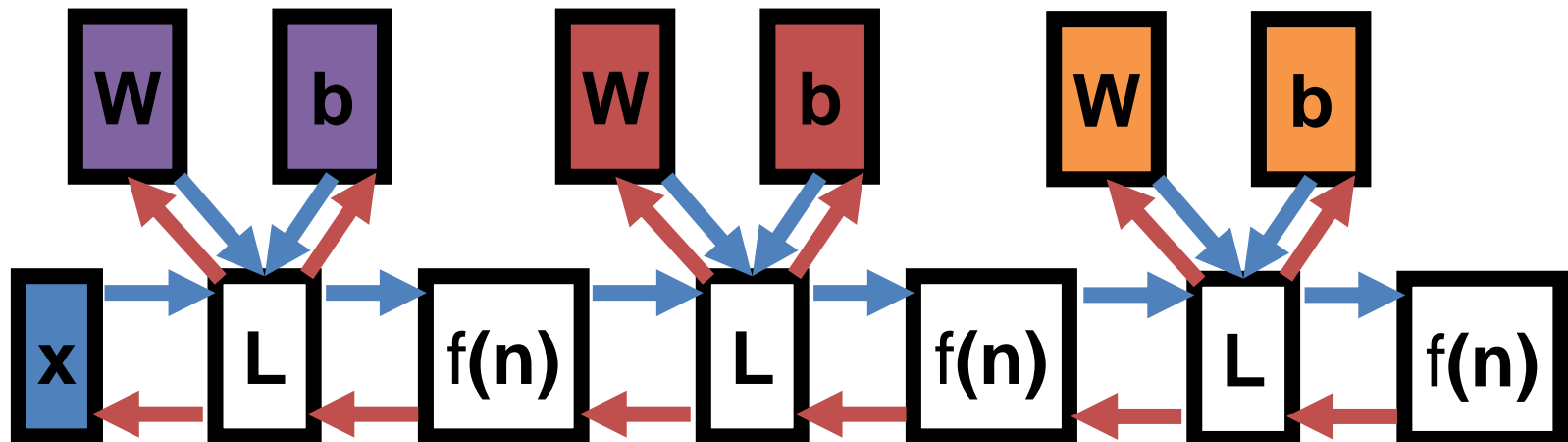
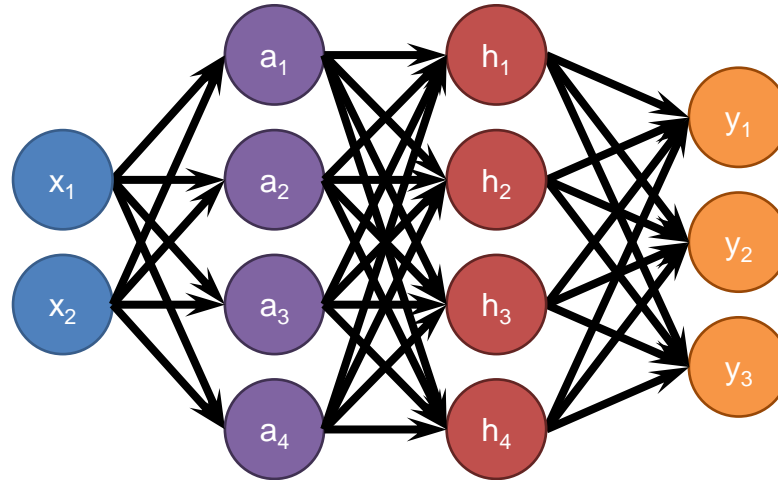
Define New Block: “Linear Layer”

(Ok technically it's Affine)

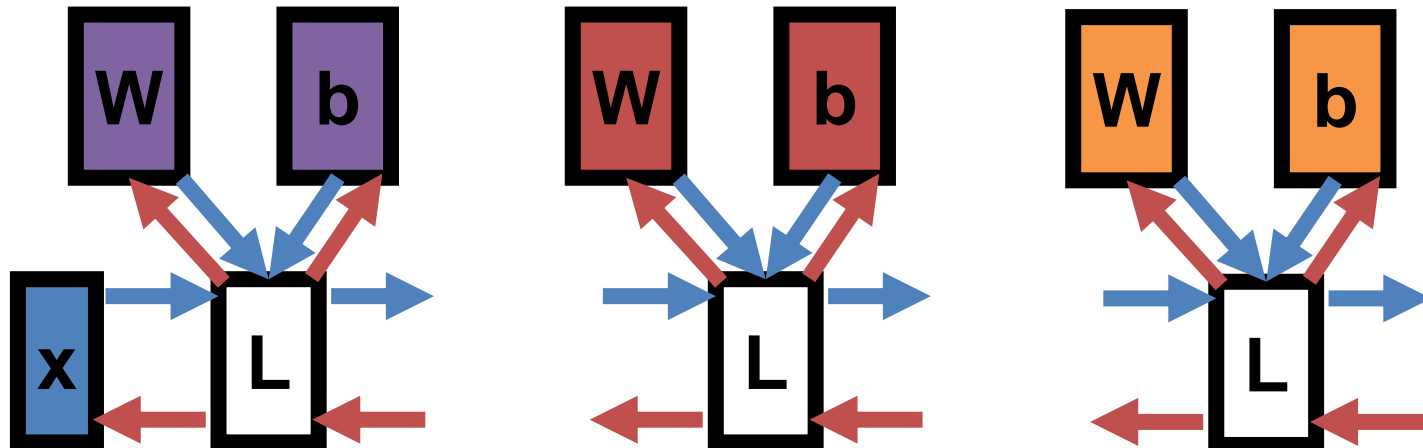
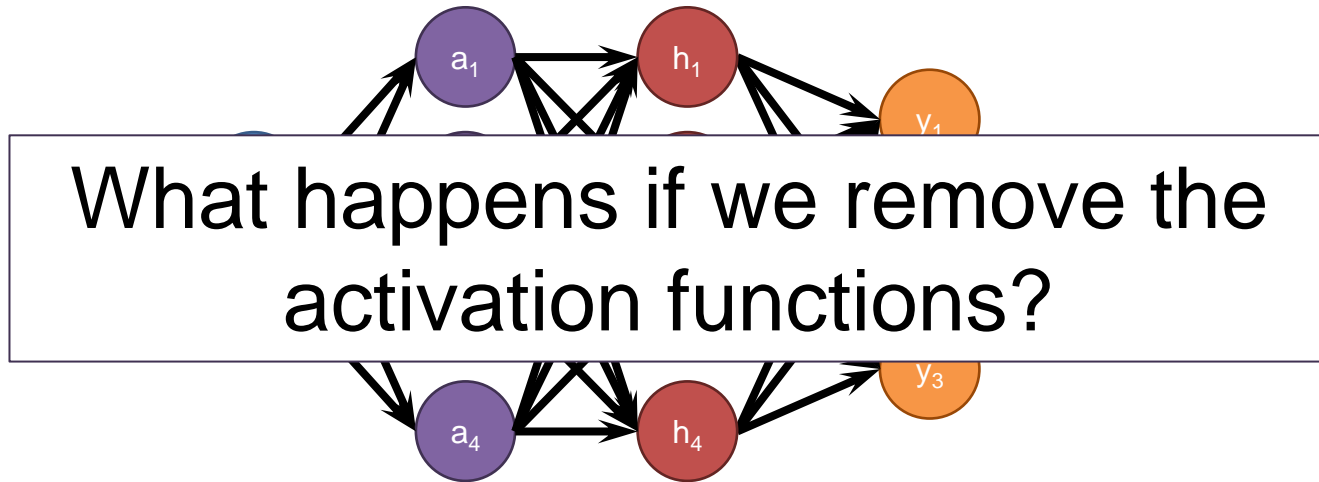


Can get gradient with respect to all the inputs
(do on your own; useful trick: have to be able to do matrix multiply)

Fully Connected Network



Fully Connected Network



Demo Time

<https://cs.stanford.edu/people/karpathy/convnets/demo/classify2d.html>

