

Topics in Social Data Science

Week 3

Artificial Neural Networks 2

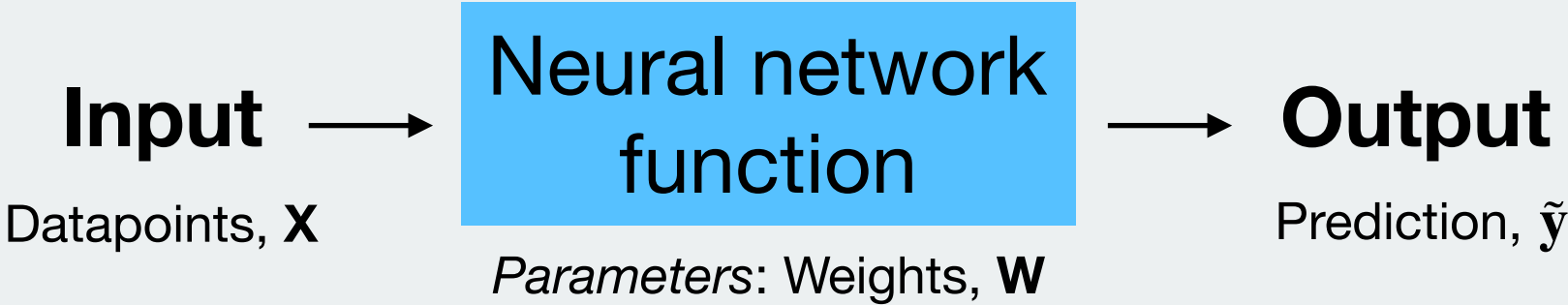
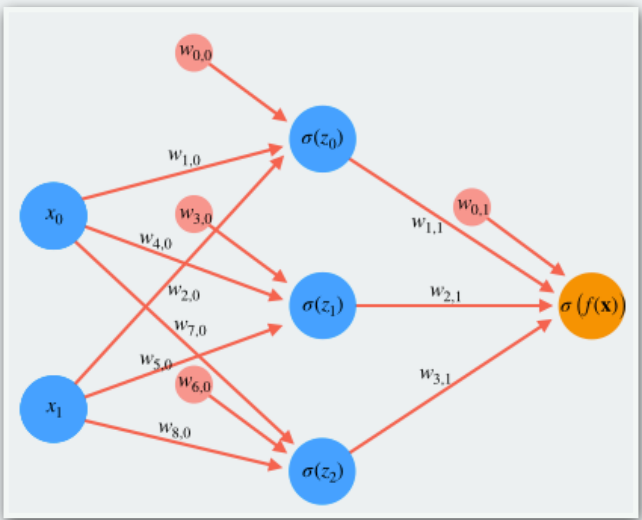
Backpropagation, regularization, vanishing gradients

Overview of today + tomorrow

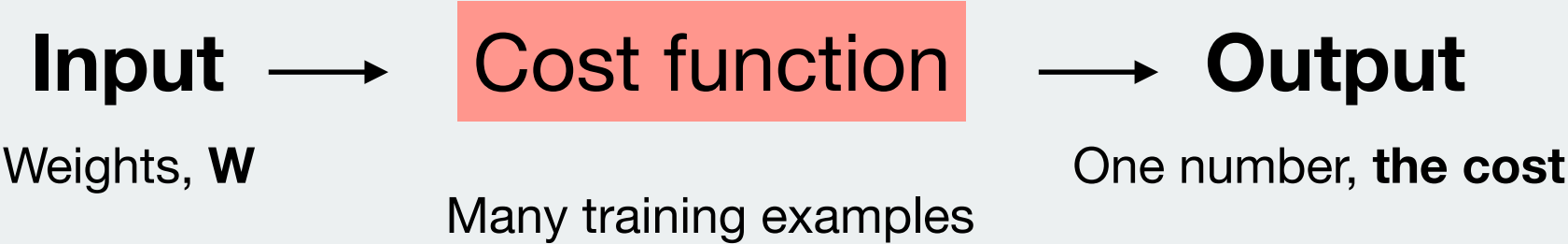
- Watch 3BLUE1BROWN's chapter 3+4 on Neural Networks
- Read (or at least familiarize yourself with) Michael Nielsen's book up to and including Chapter 4
- My lecture (backprop, regul., vanishing grad.)
- Exercises in Python

Quick recap...

(1) The model



(2) Its performance



$$\begin{aligned} C(\mathbf{W}) &= \frac{1}{N} \sum_i (\tilde{y}_i - y_i)^2 \\ &= (0.96 - 1)^2 \\ &\quad + (0.10 - 0)^2 \\ &\quad + (0.04 - 0)^2 \\ &\quad + \dots \\ &\quad + (0.70 - 1)^2 \\ &\quad + (0.02 - 0)^2 \\ &\quad + (0.99 - 1)^2 \end{aligned}$$

(3) The cost function gradient in \mathbf{W}

$$-\nabla C(\mathbf{W}) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ 0.54 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}$$

Find the gradients with
Backpropagation
... this week

(4) Updating \mathbf{W}

$$\mathbf{W} = \mathbf{W}^{\text{old}} + r (-\nabla C(\mathbf{W}^{\text{old}}))$$

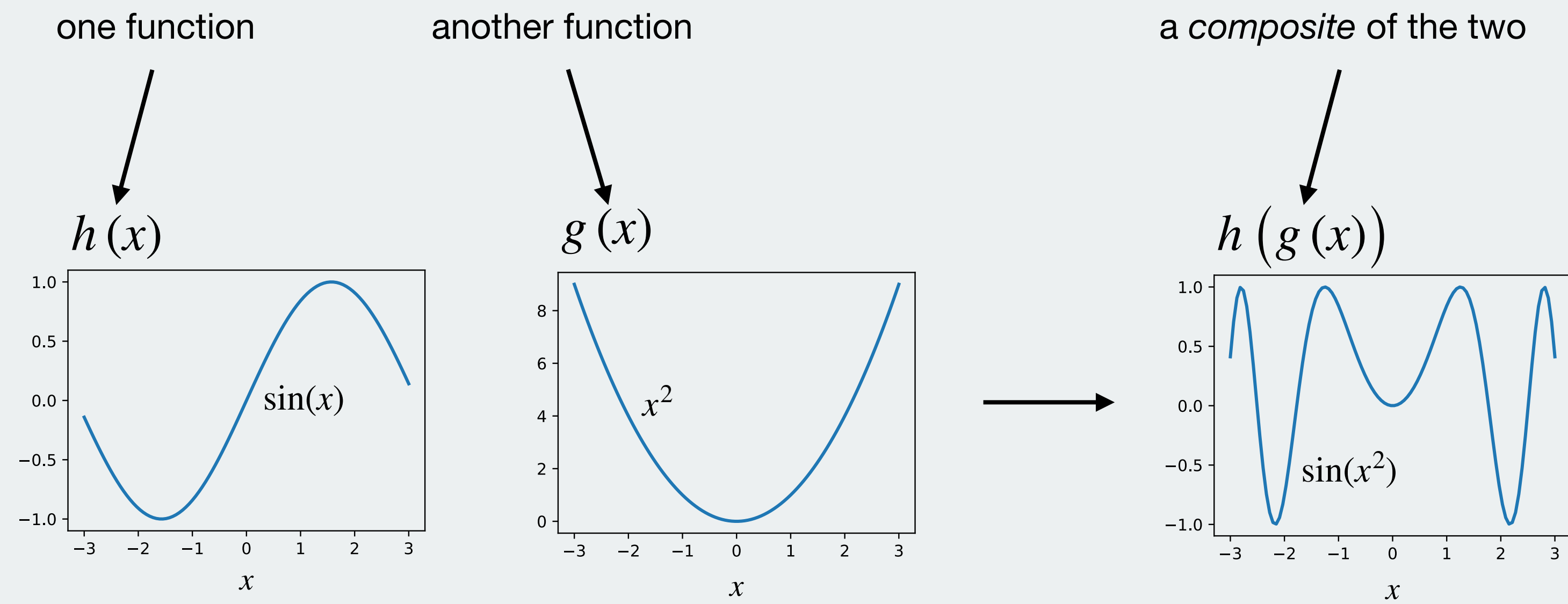
(5) Repeat 3 and 4

r is usually called the *learning rate*

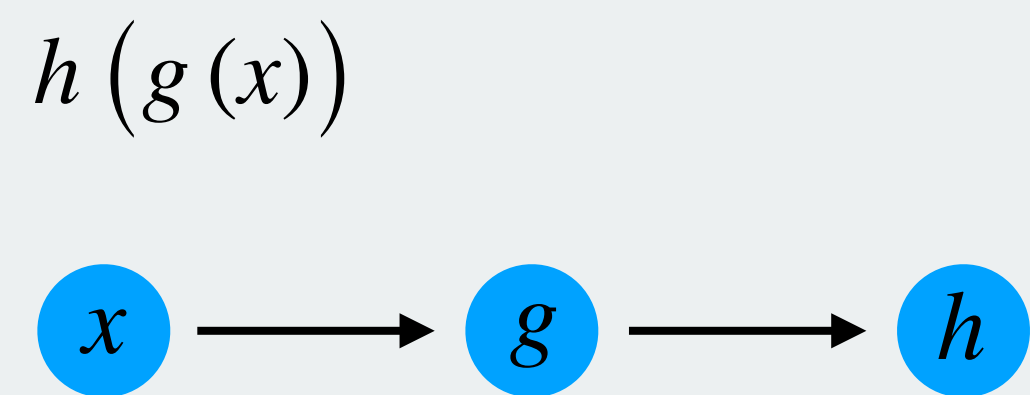
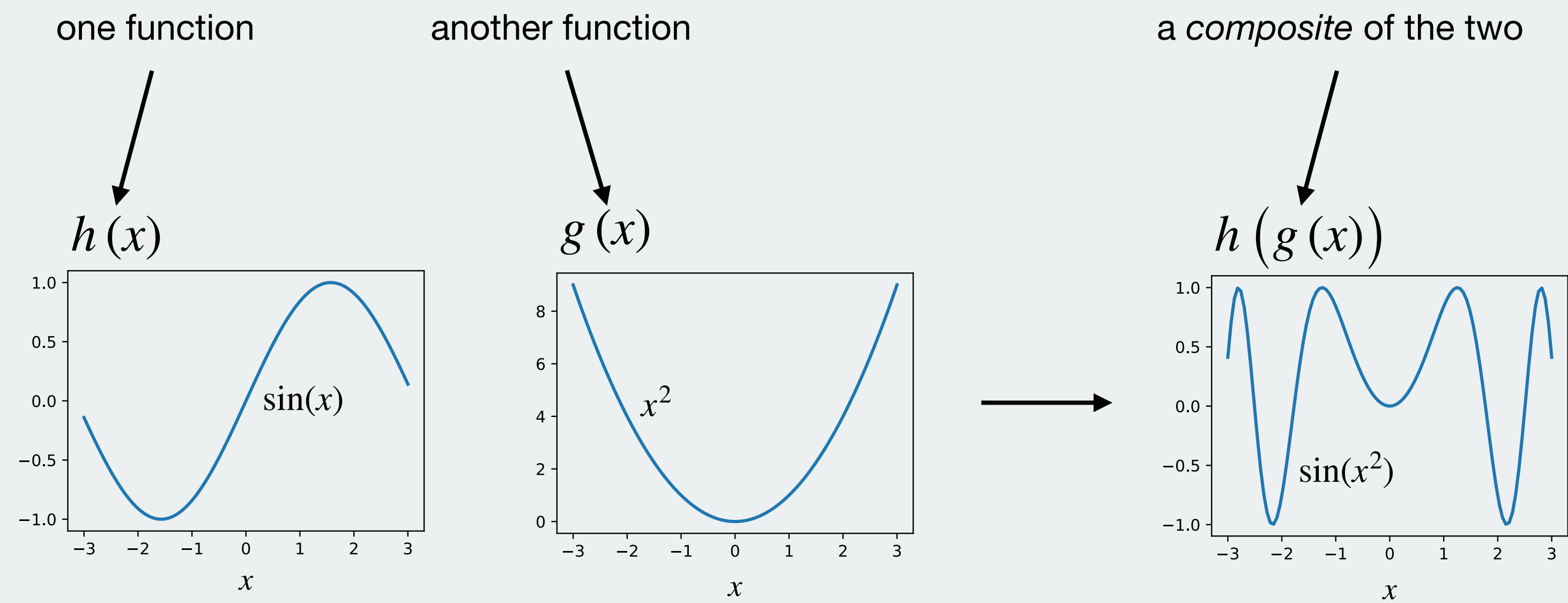
Backpropagation

THE algorithm for computing the analytical **gradient** of the cost function

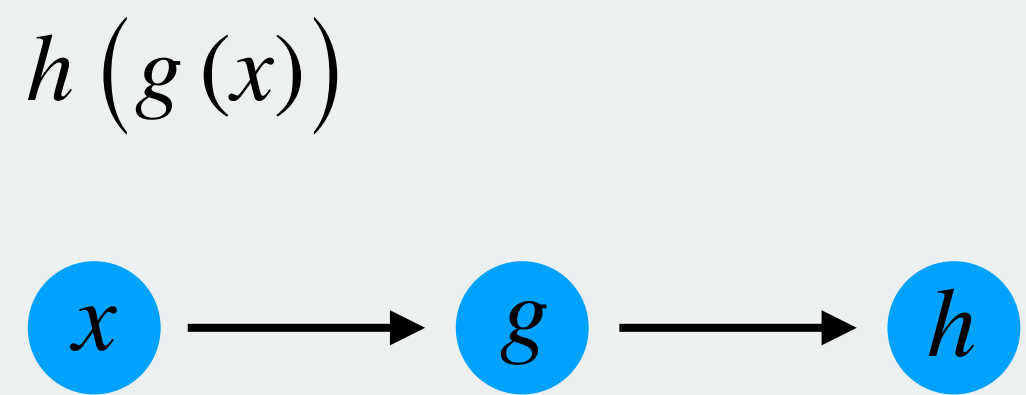
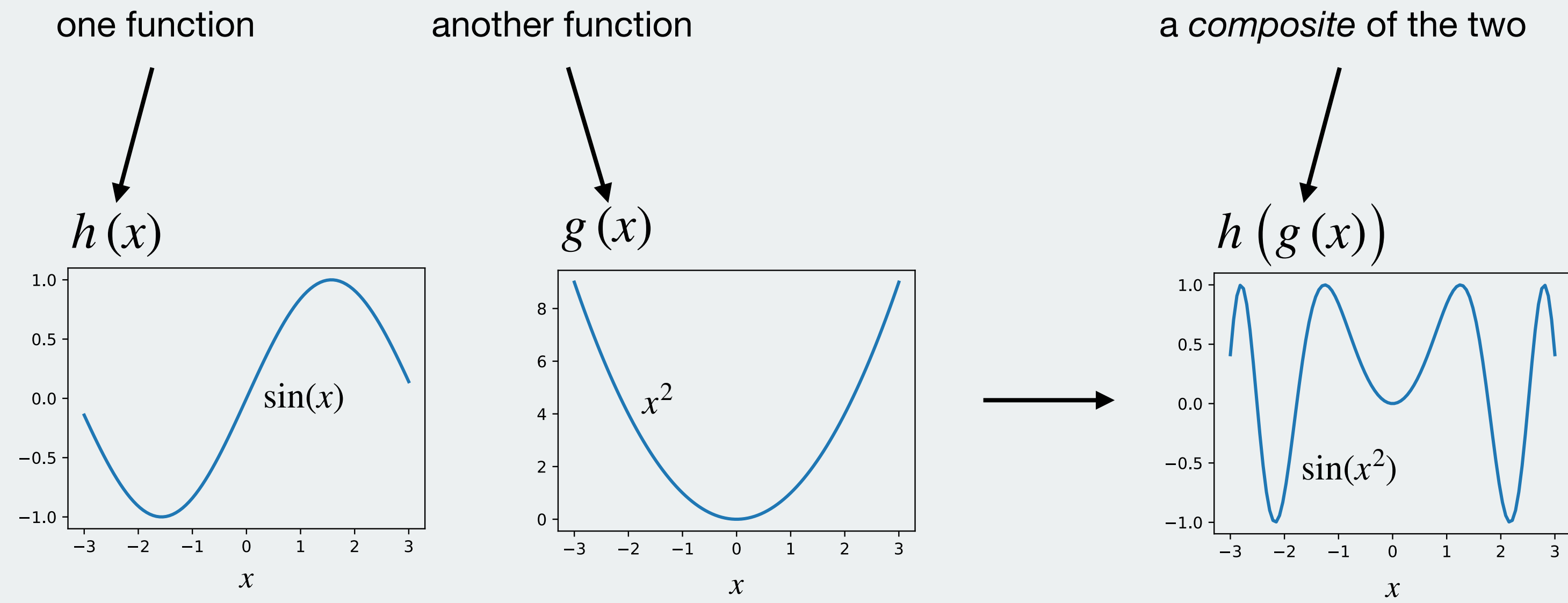
Chain rule – Taking derivatives of *composite functions*



Chain rule – Taking derivatives of *composite functions*



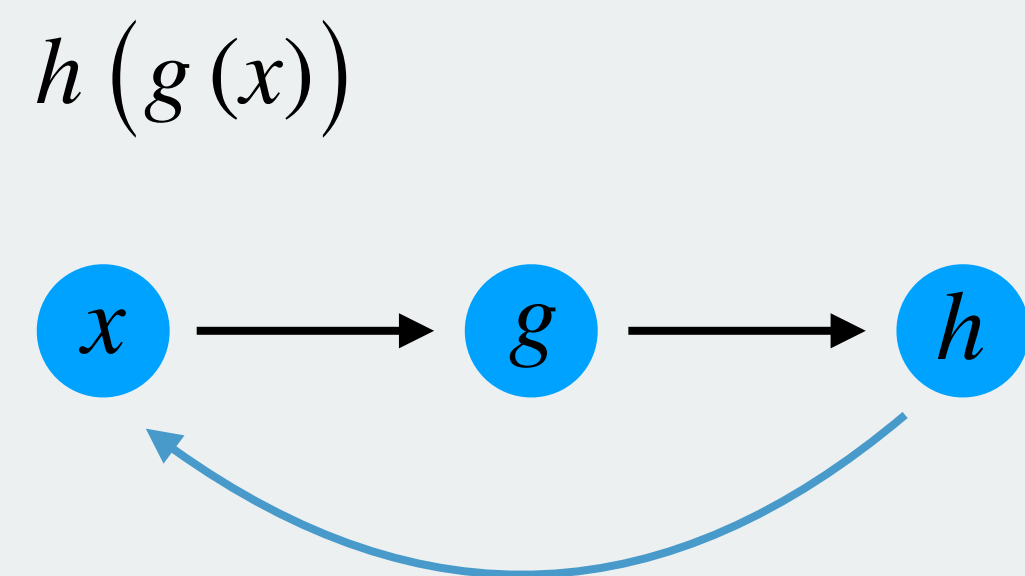
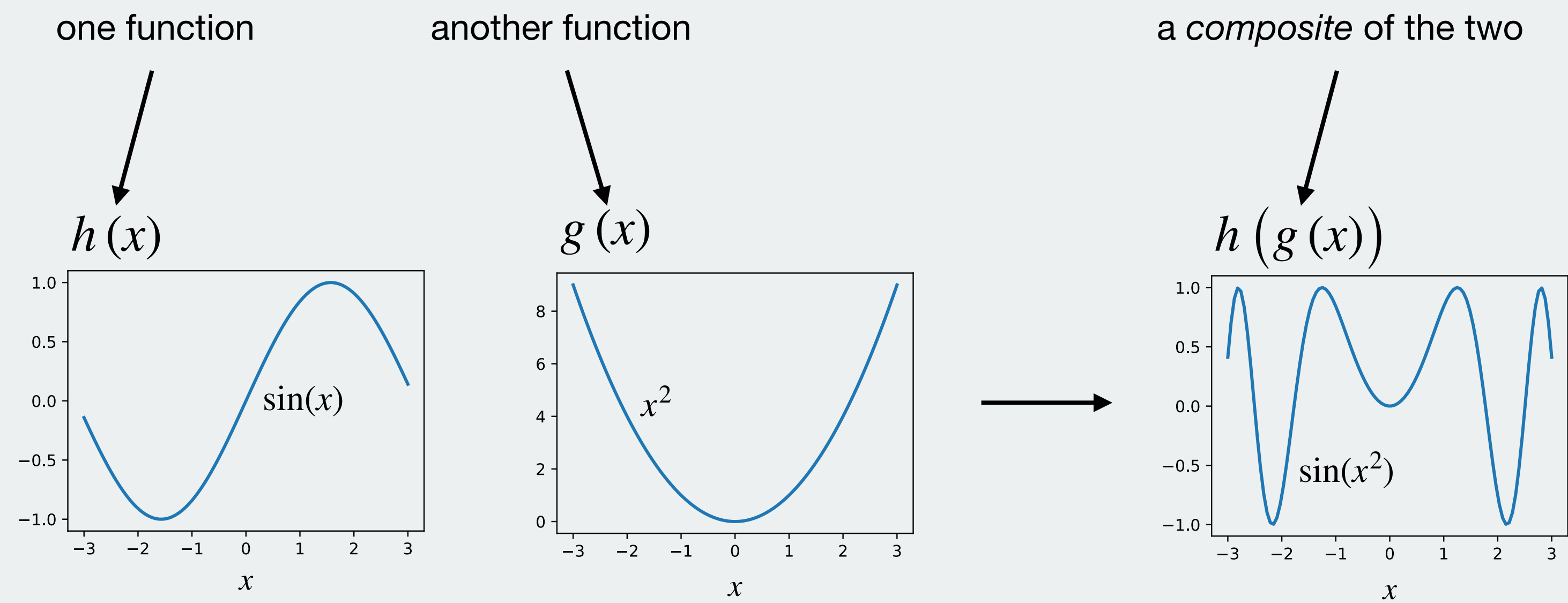
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg} \frac{dg}{dx}$$

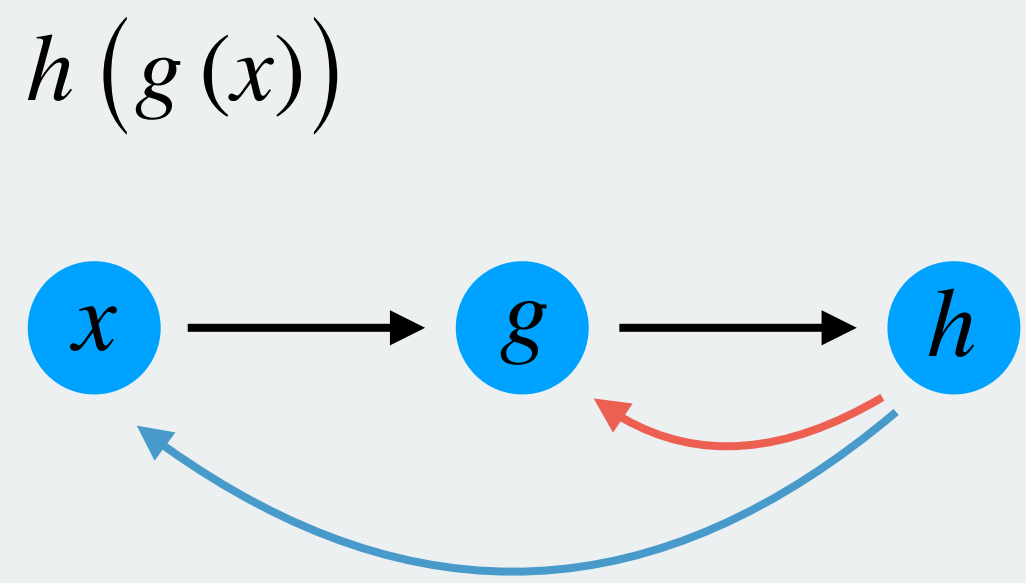
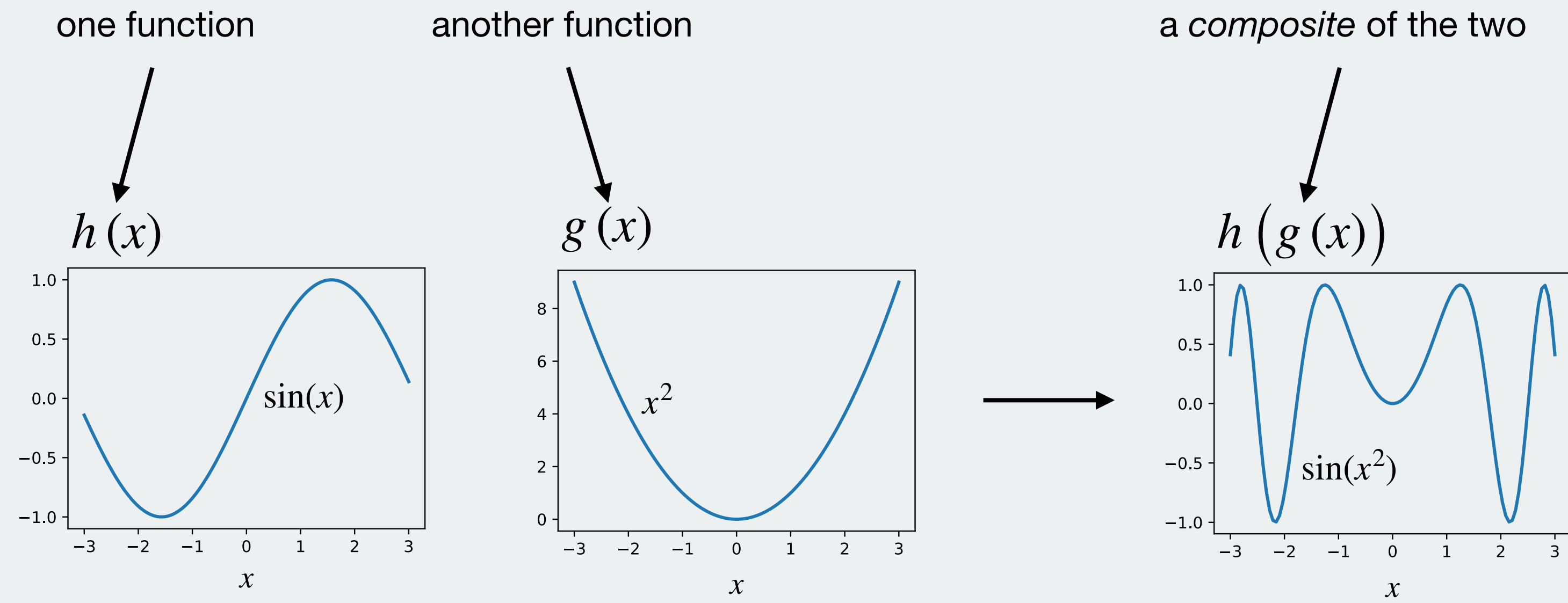
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg} \frac{dg}{dx}$$

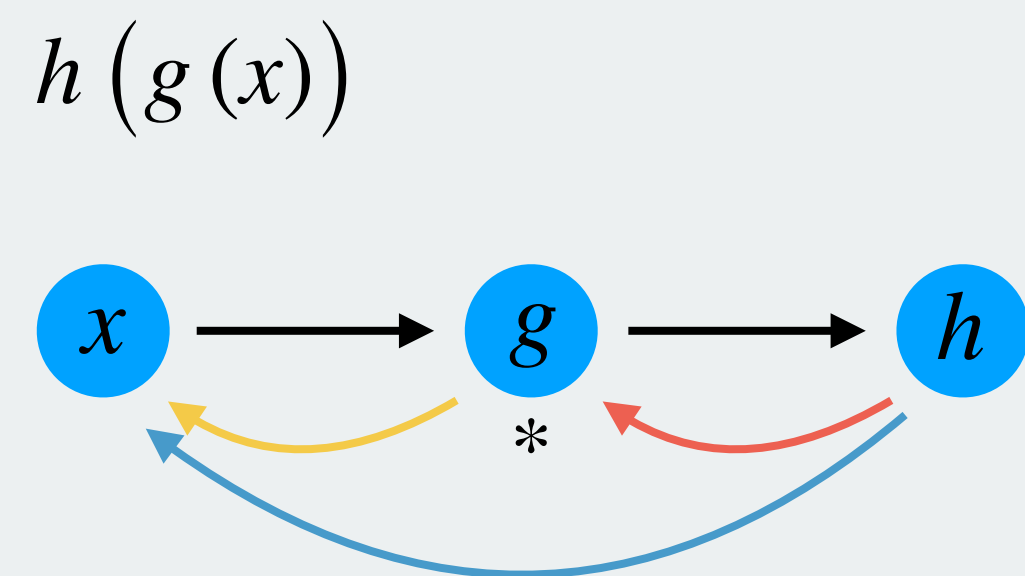
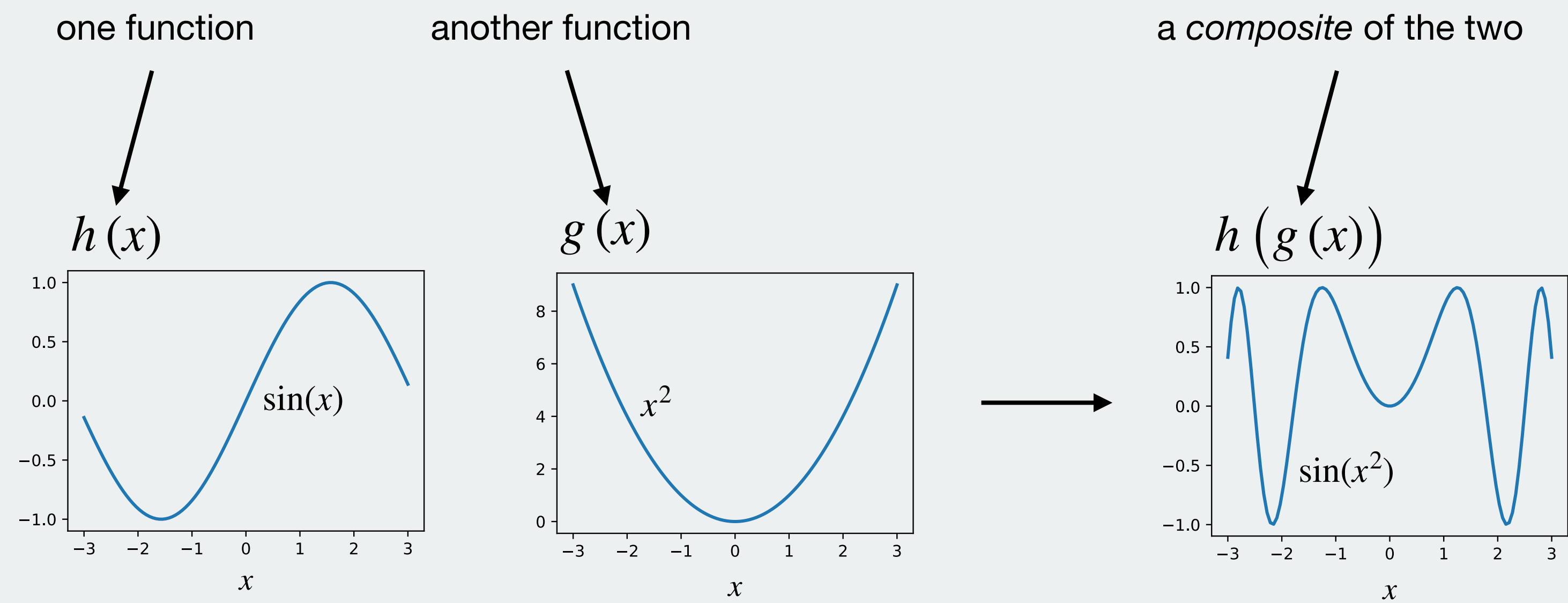
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg} \frac{dg}{dx}$$

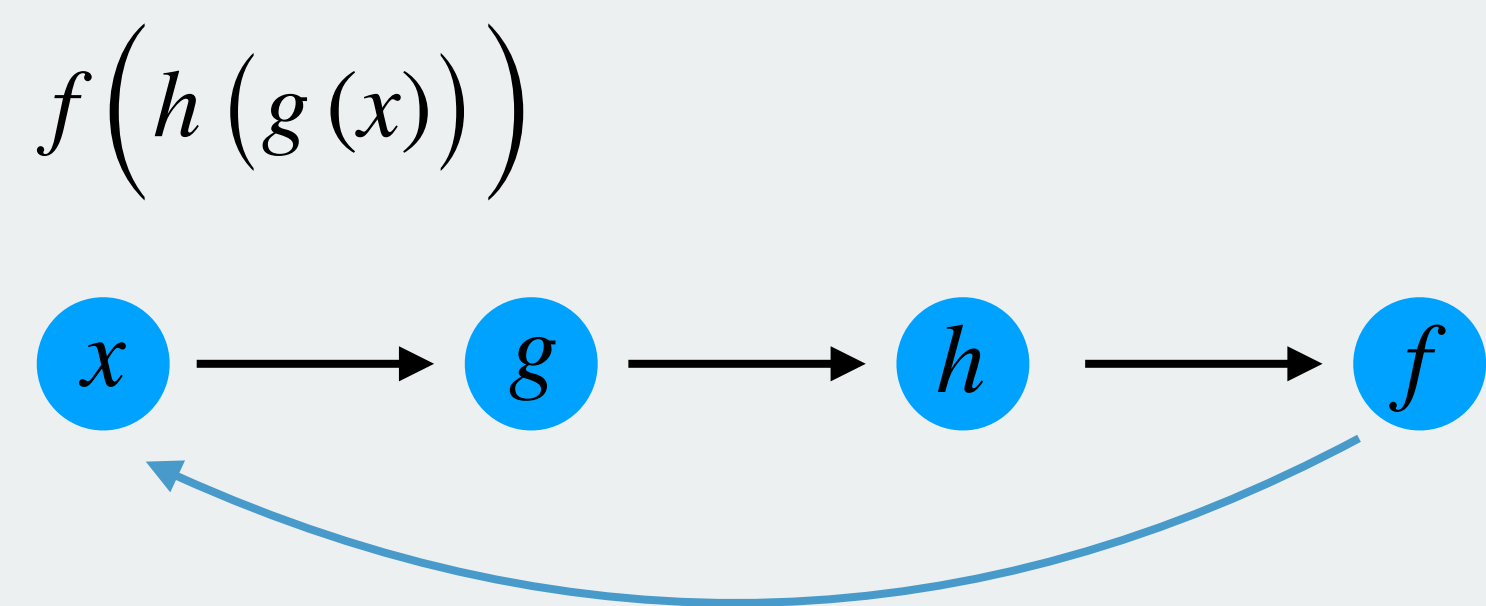
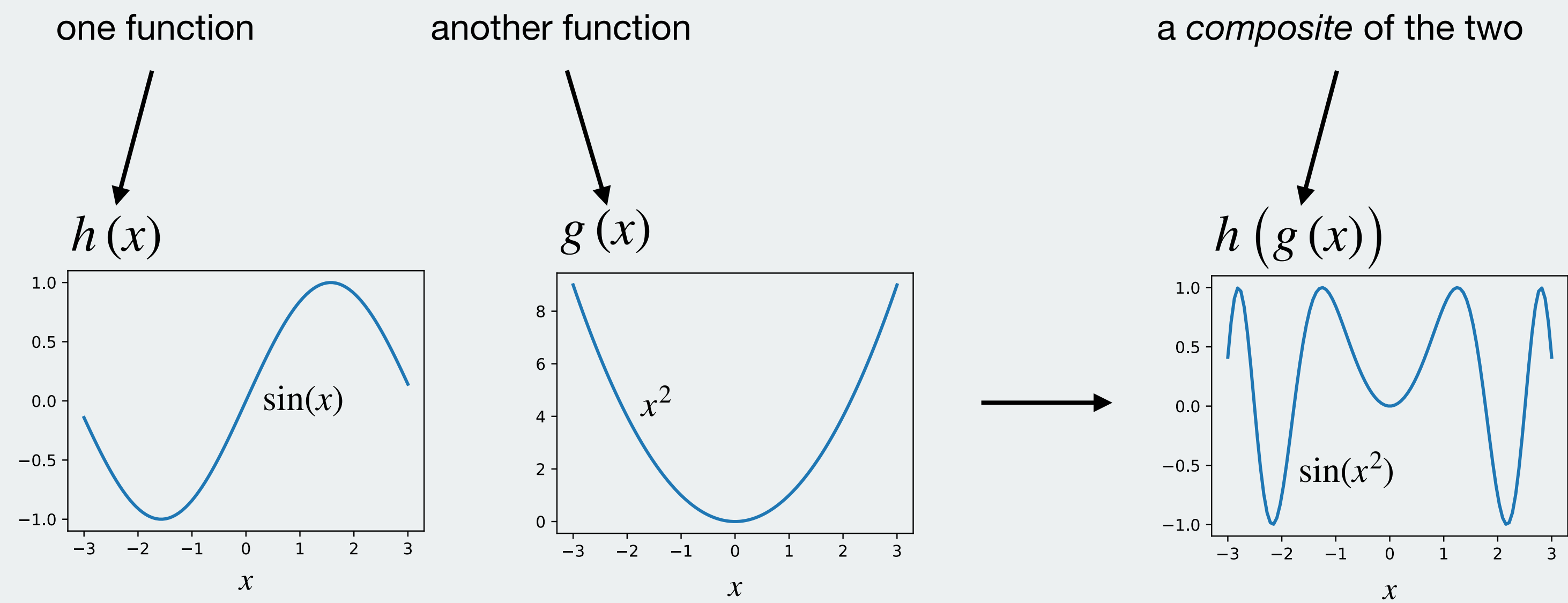
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg} \frac{dg}{dx}$$

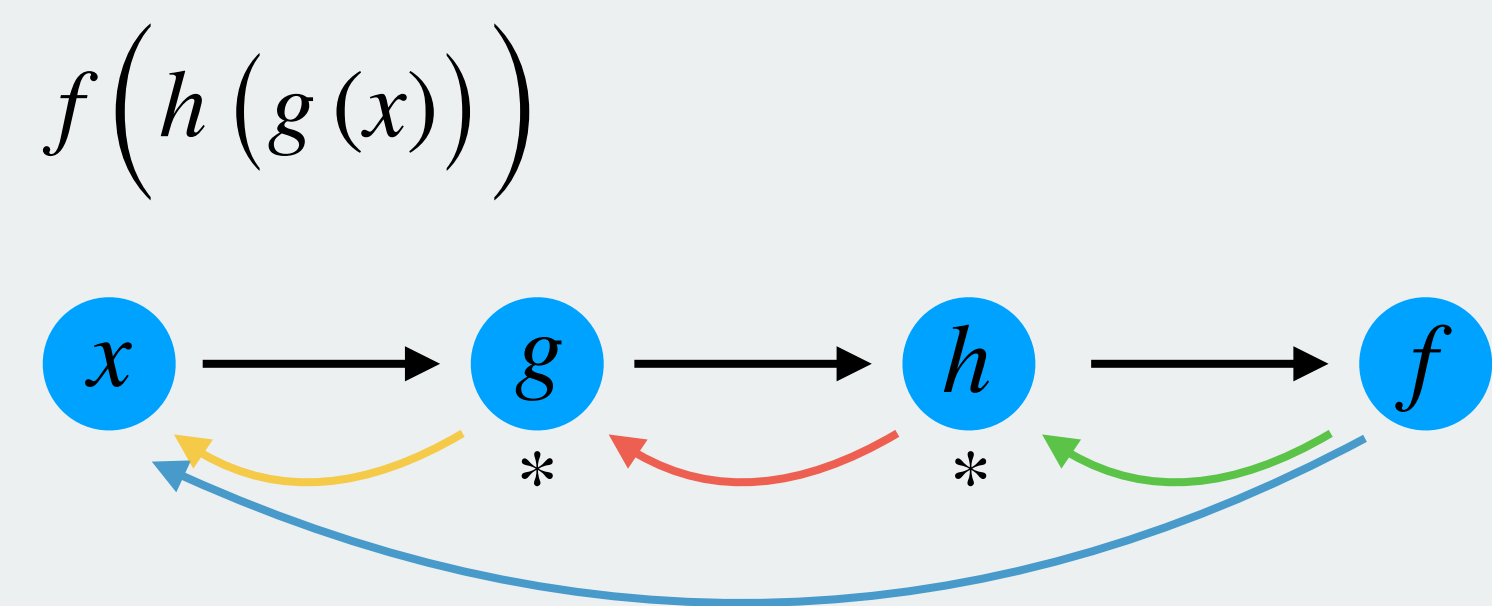
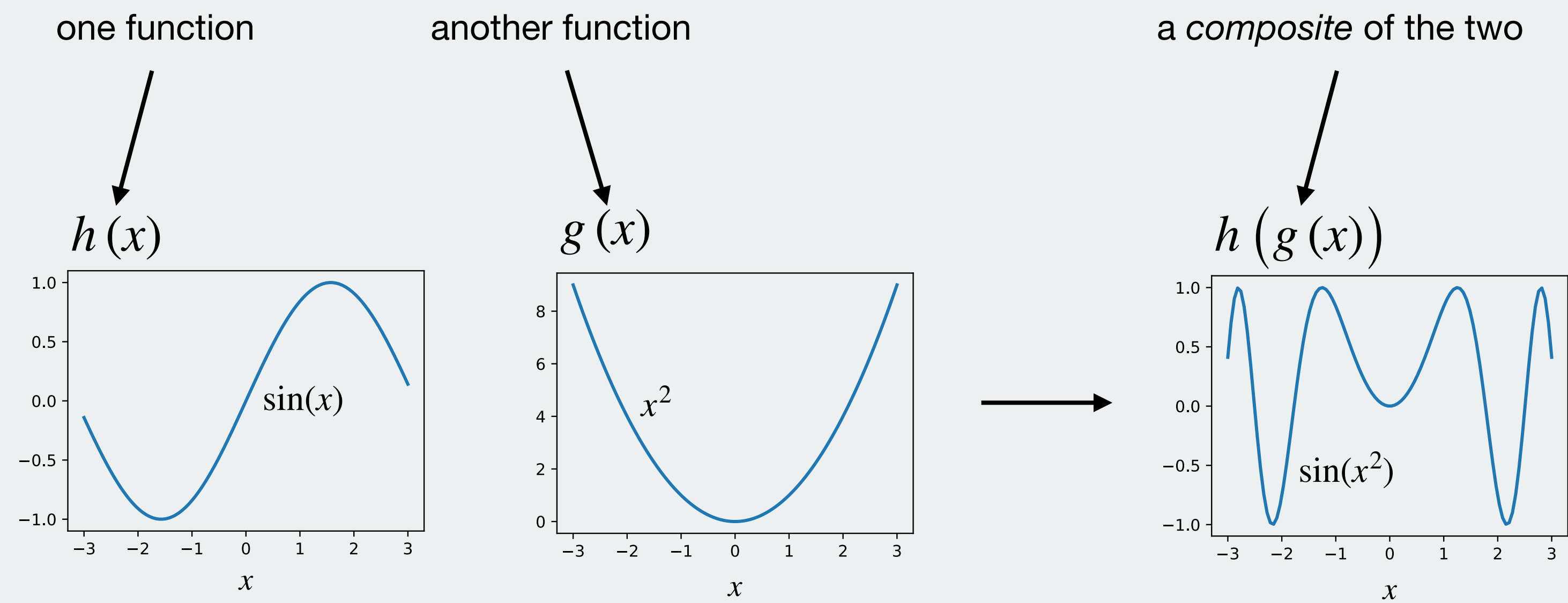
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

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

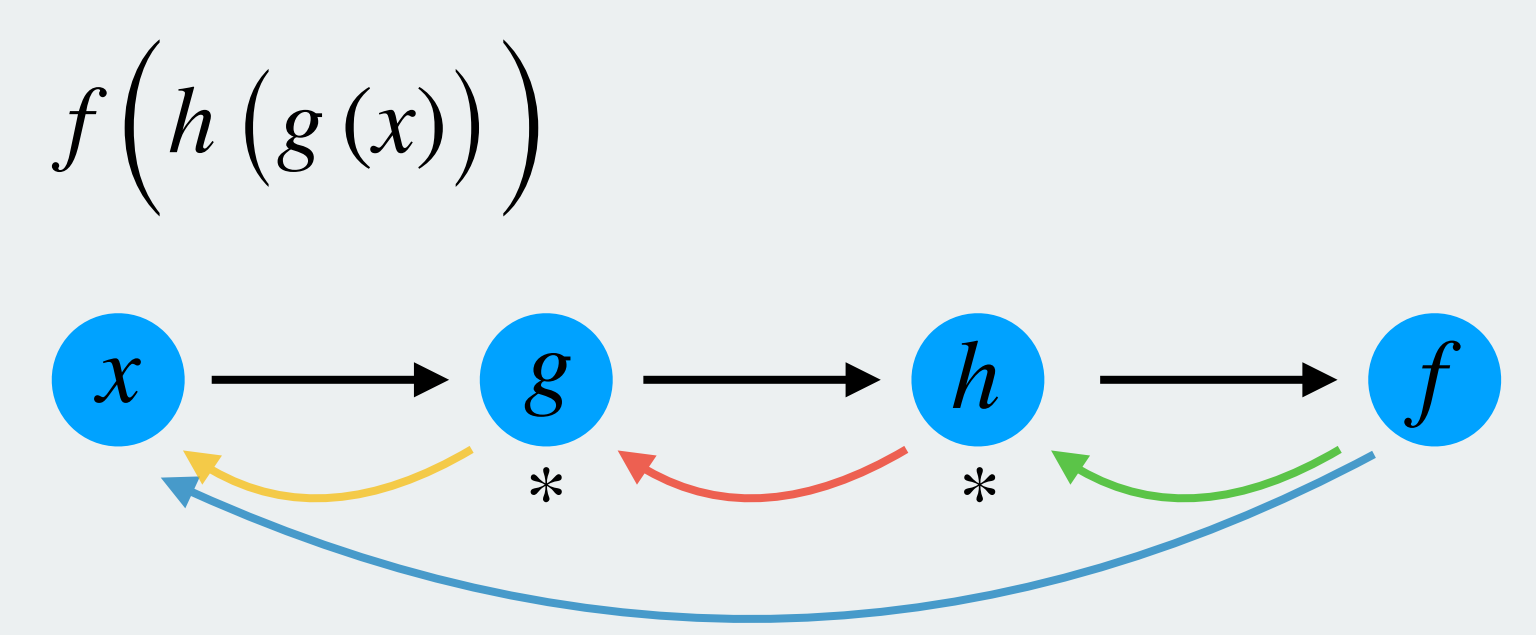
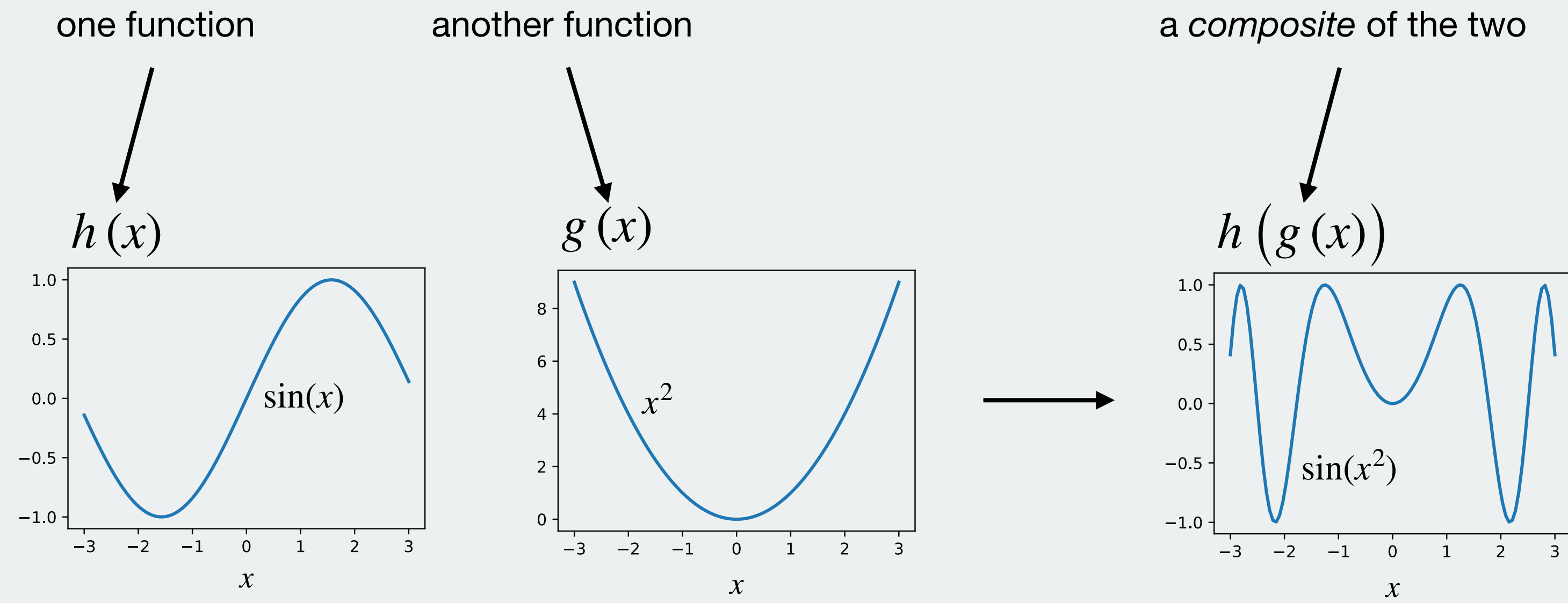
Chain rule – Taking derivatives of *composite functions*



Chain rule says:

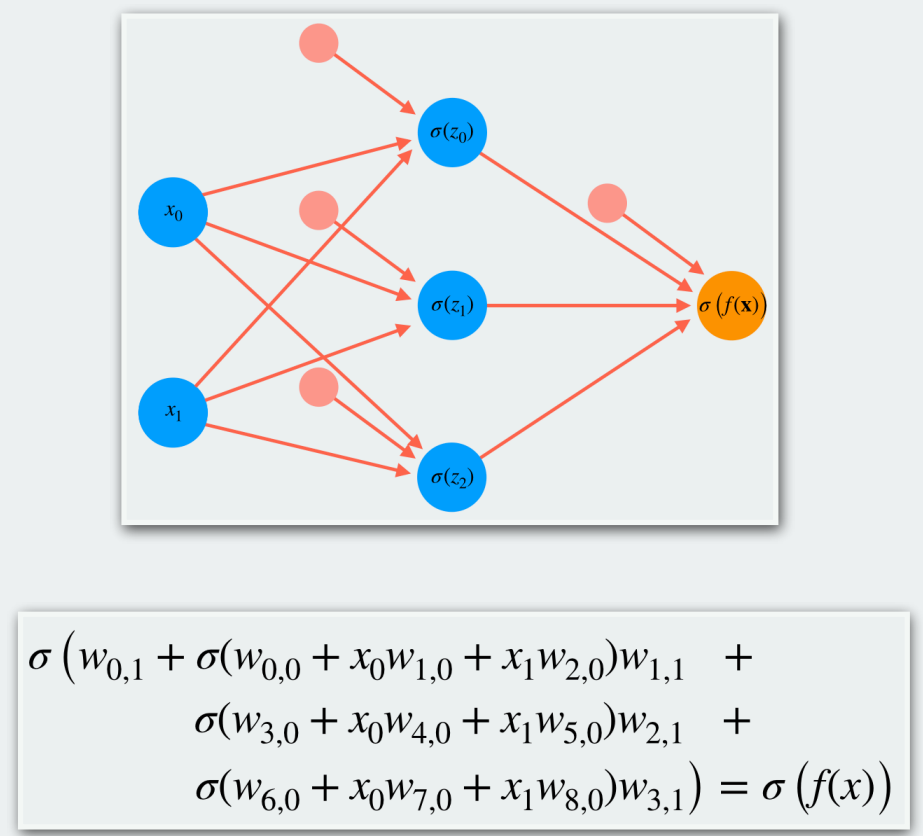
$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dx}$$

Chain rule – Taking derivatives of *composite functions*



Chain rule says:

$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dx}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$
$$g(z) = \exp(z)$$
$$h(z) = z + 1$$
$$f(z) = \frac{1}{z}$$

Data:

$$x_0 = 1$$
$$x_1 = 1.1$$



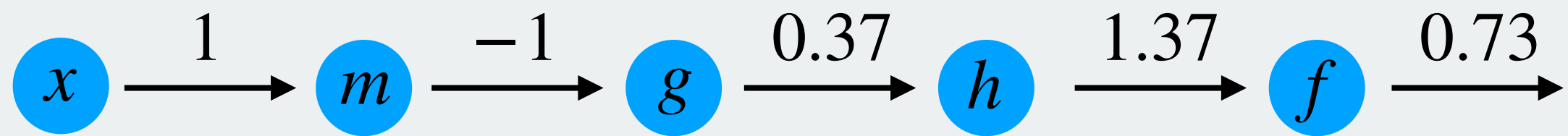
Backpropagation – simple example

Model:

$$m(z) = -z$$
$$g(z) = \exp(z)$$
$$h(z) = z + 1$$
$$f(z) = \frac{1}{z}$$

Data:

$$x_0 = 1$$
$$x_1 = 1.1$$



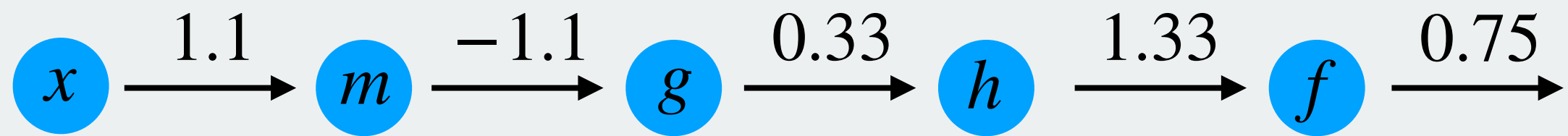
Backpropagation – simple example

Model:

$$m(z) = -z$$
$$g(z) = \exp(z)$$
$$h(z) = z + 1$$
$$f(z) = \frac{1}{z}$$

Data:

$$x_0 = 1$$
$$x_1 = 1.1$$



Backpropagation – simple example

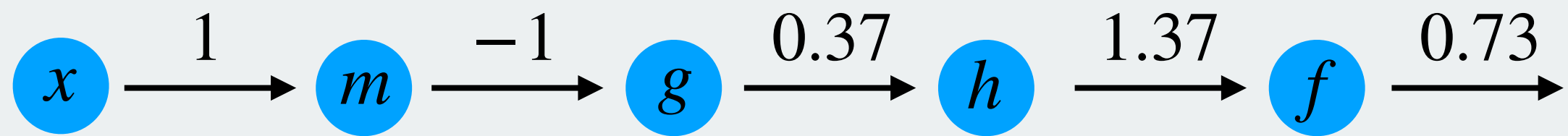
Model:

$$m(z) = -z$$
$$g(z) = \exp(z)$$
$$h(z) = z + 1$$
$$f(z) = \frac{1}{z}$$

Data:

$$x_0 = 1$$
$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?



Backpropagation – simple example

Model:

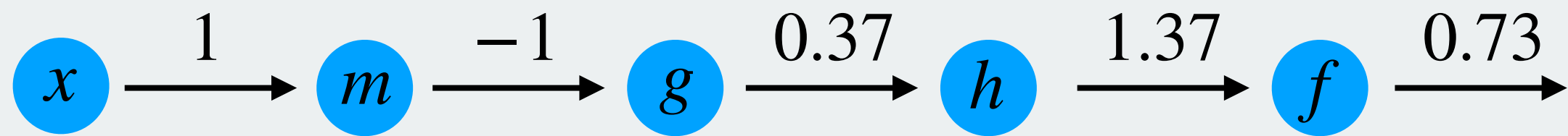
$$\begin{aligned} m(z) &= -z \\ g(z) &= \exp(z) \\ h(z) &= z + 1 \\ f(z) &= \frac{1}{z} \end{aligned}$$

Data:

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 1.1 \end{aligned}$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule from f to x :

$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm} \frac{dm}{dx}$$

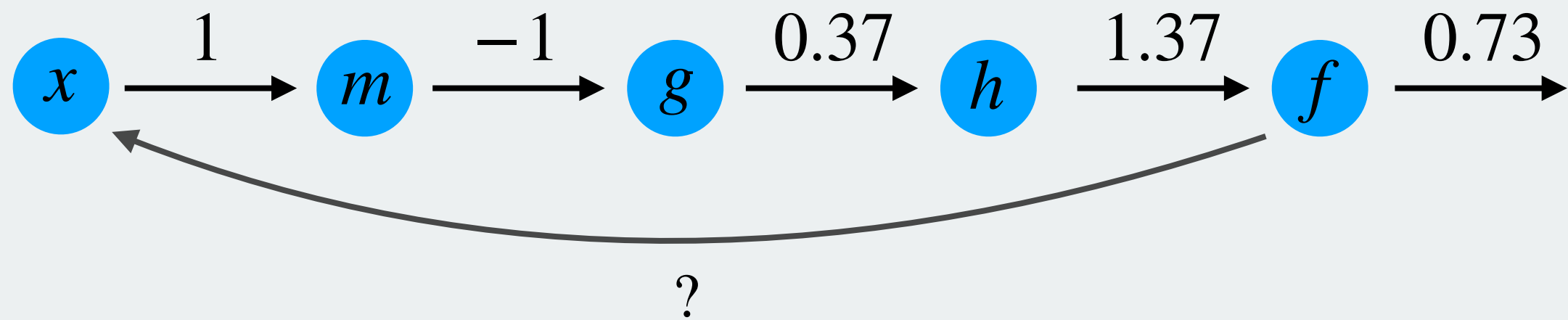
Model derivatives:

$$m'(z) = -1$$

$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

$$f'(z) = -\frac{1}{z^2}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule from f to h :

$$\frac{df}{dh}$$

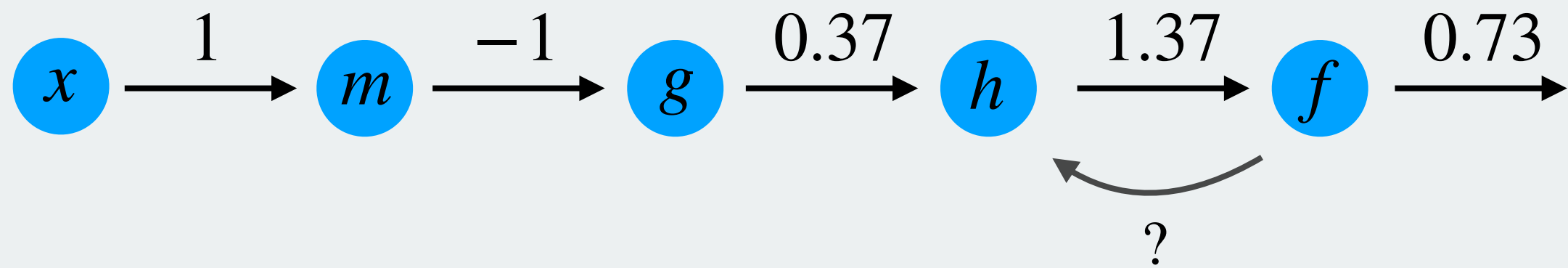
Model derivatives:

$$m'(z) = -1$$

$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

$$f'(z) = -\frac{1}{z^2}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule from f to h :

$$\frac{df}{dh}$$

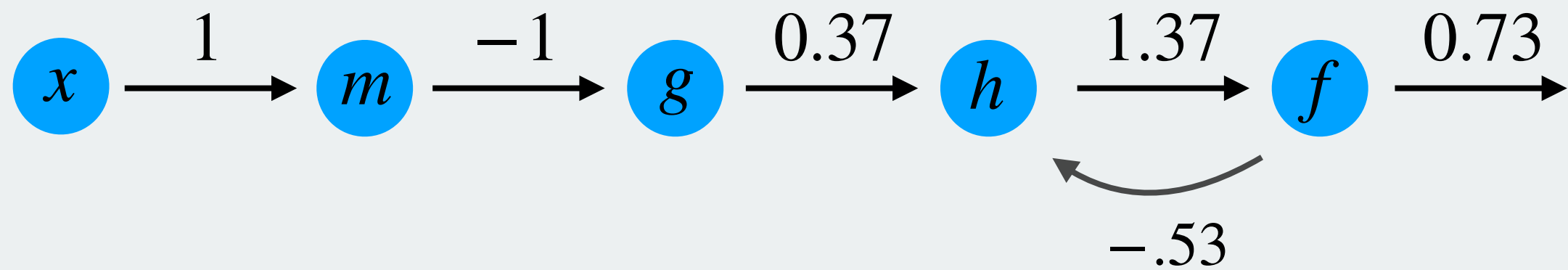
Model derivatives:

$$m'(z) = -1$$

$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

$$f'(z) = -\frac{1}{z^2}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule from f to g :

$$\frac{df}{dg} = \frac{df}{dh} \frac{dh}{dg}$$

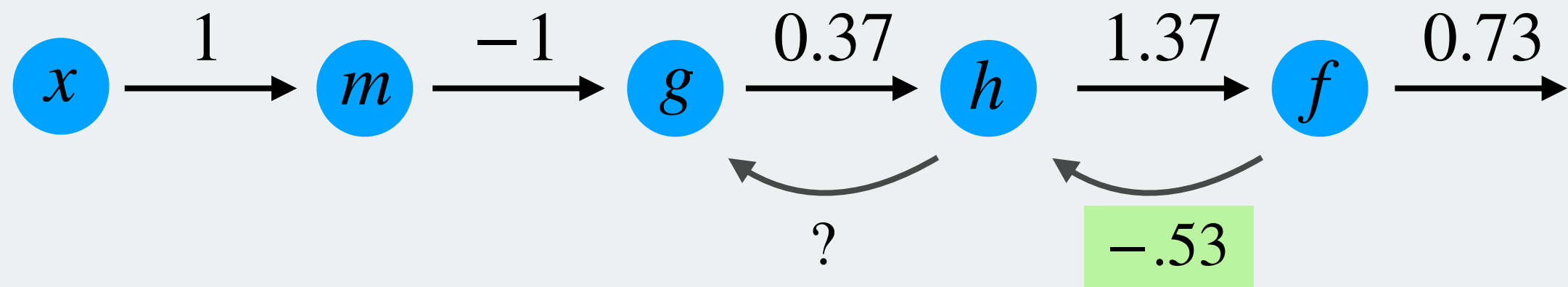
Model derivatives:

$$m'(z) = -1$$

$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

$$f'(z) = -\frac{1}{z^2}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

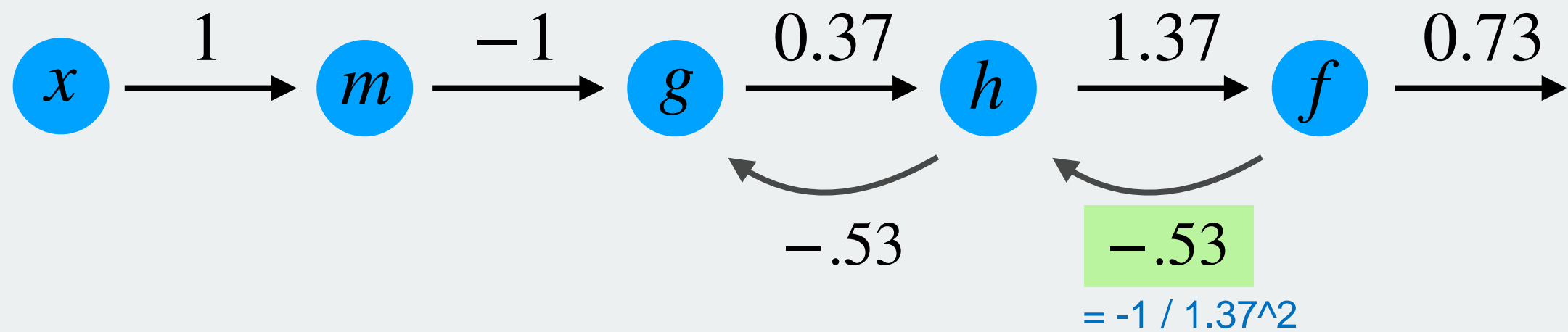
A: Propagate gradients backwards using the chain rule!

Chain rule from f to g :

$$\frac{df}{dg} = \frac{df}{dh} \frac{dh}{dg}$$

Model derivatives:

$$\begin{aligned} m'(z) &= -1 \\ g'(z) &= \exp(z) \\ h'(z) &= 1 \\ f'(z) &= -\frac{1}{z^2} \end{aligned}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

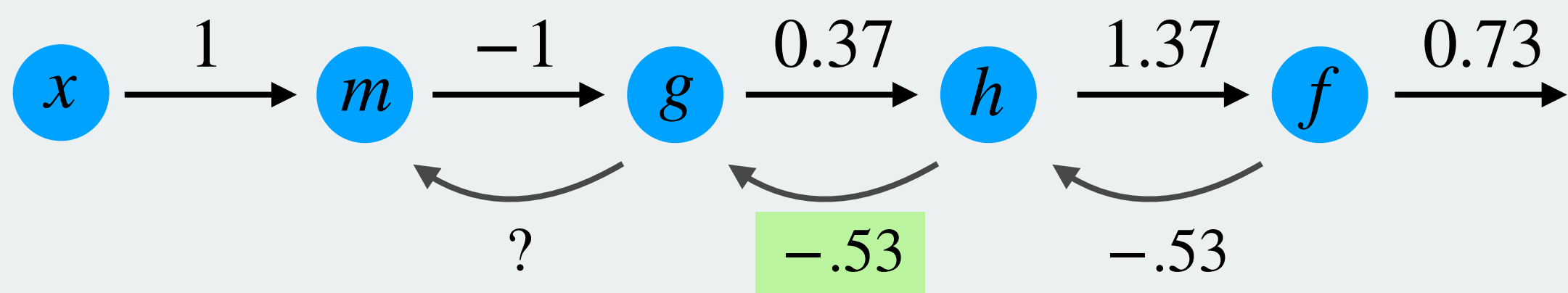
A: Propagate gradients backwards using the chain rule!

Chain rule from f to m :

$$\frac{df}{dm} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm}$$

Model derivatives:

$$\begin{aligned} m'(z) &= -1 \\ g'(z) &= \exp(z) \\ h'(z) &= 1 \\ f'(z) &= -\frac{1}{z^2} \end{aligned}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule from f to m :

$$\frac{df}{dm} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm}$$

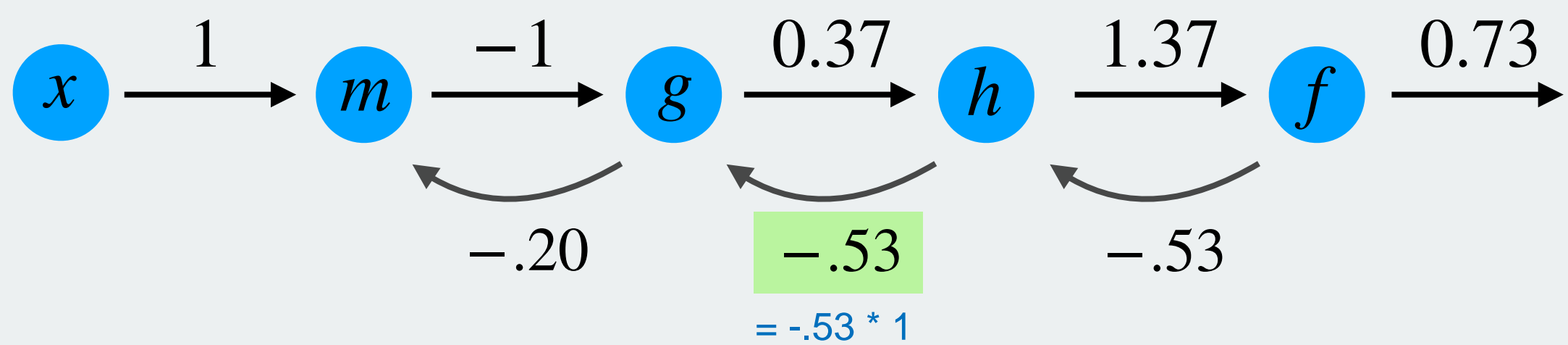
Model derivatives:

$$m'(z) = -1$$

$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

$$f'(z) = -\frac{1}{z^2}$$



Backpropagation – simple example

Model:

$$m(z) = -z$$

$$g(z) = \exp(z)$$

$$h(z) = z + 1$$

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

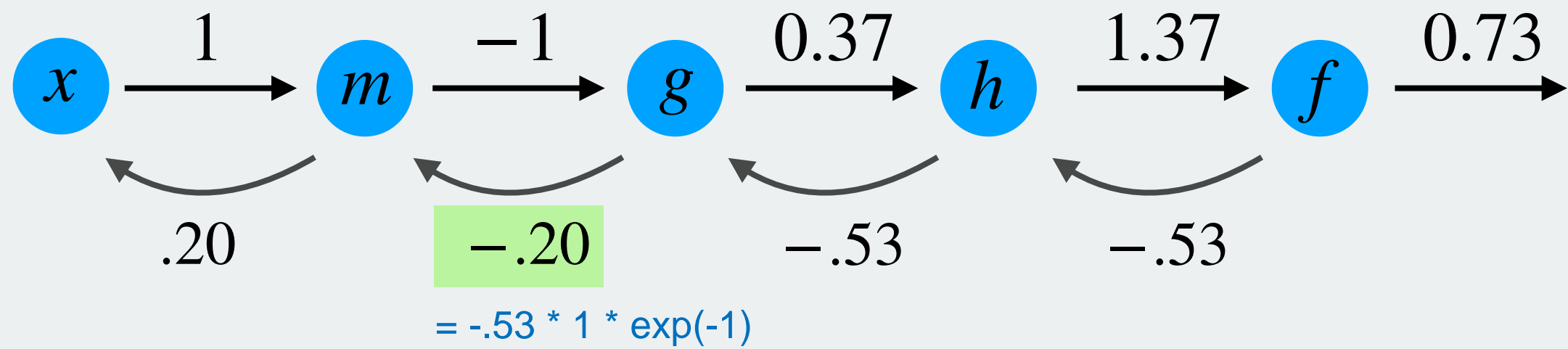
A: Propagate gradients backwards using the chain rule!

Chain rule from f to x :

$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm} \frac{dm}{dx}$$

Model derivatives:

$$\begin{aligned} m'(z) &= -1 \\ g'(z) &= \exp(z) \\ h'(z) &= 1 \\ f'(z) &= -\frac{1}{z^2} \end{aligned}$$



Backpropagation – simple example

Model:
 $m(z) = -z$
 $g(z) = \exp(z)$
 $h(z) = z + 1$
 $f(z) = \frac{1}{z}$

Data:
 $x_0 = 1$
 $x_1 = 1.1$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

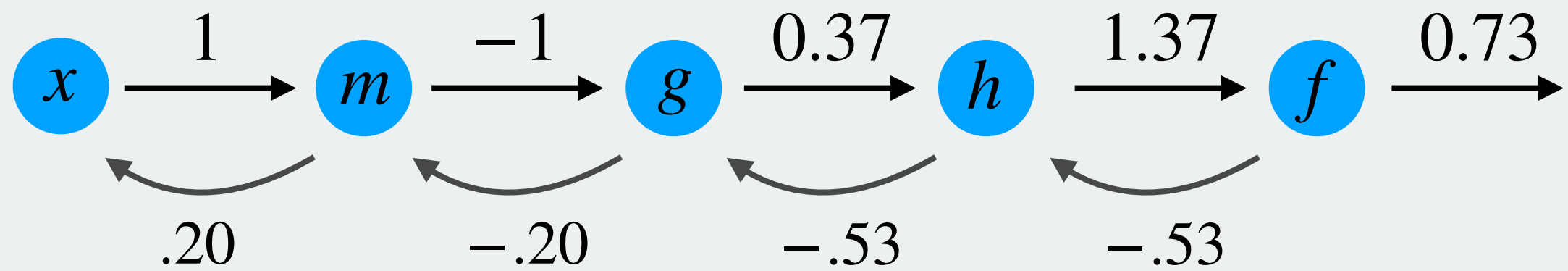
Chain rule:
$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm} \frac{dm}{dx}$$

Model derivatives:
 $m'(z) = -1$
 $g'(z) = \exp(z)$
 $h'(z) = 1$
 $f'(z) = -\frac{1}{z^2}$

Sigmoid function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Sigmoid derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$


Backpropagation – simple example

Model:
 $m(z) = -z$
 $g(z) = \exp(z)$
 $h(z) = z + 1$
 $f(z) = \frac{1}{z}$

Data:
 $x_0 = 1$
 $x_1 = 1.1$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

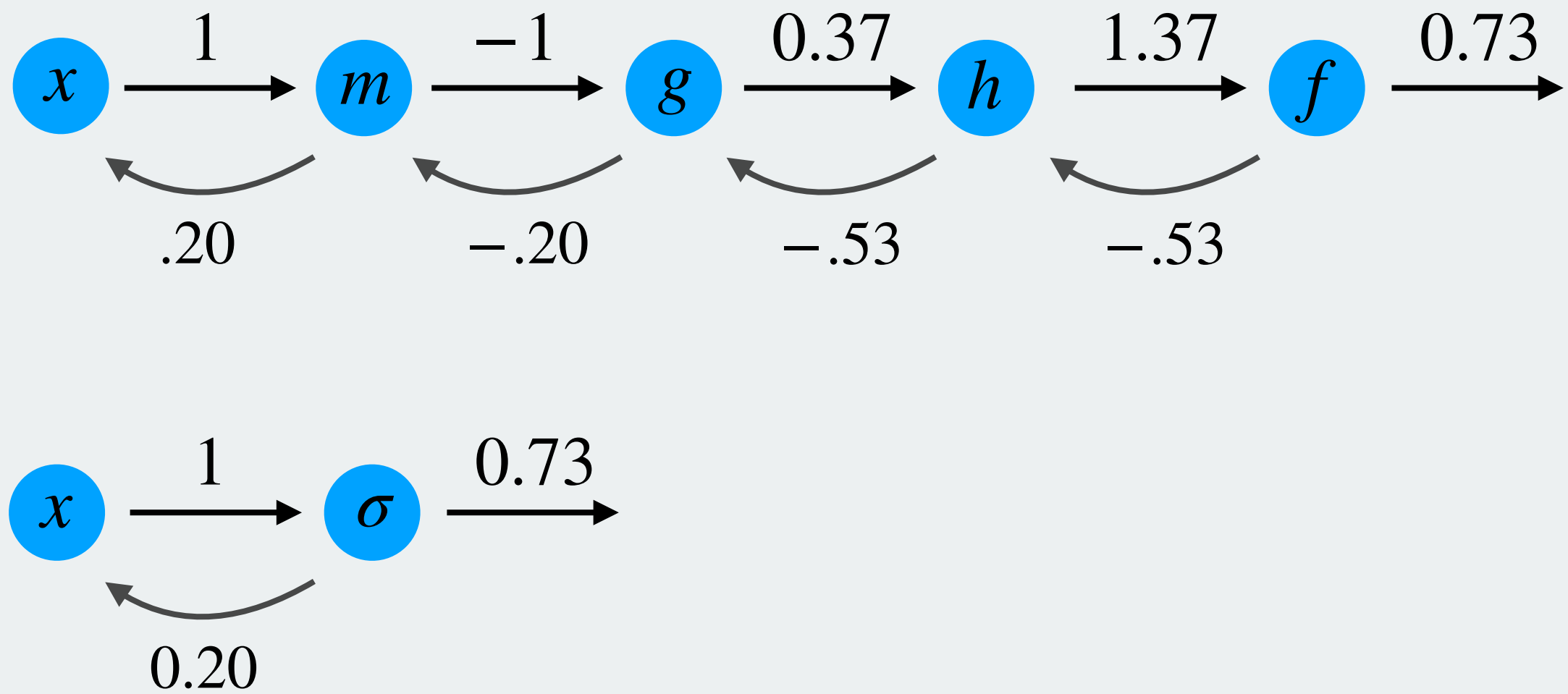
Chain rule:
$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm} \frac{dm}{dx}$$

Model derivatives:
 $m'(z) = -1$
 $g'(z) = \exp(z)$
 $h'(z) = 1$
 $f'(z) = -\frac{1}{z^2}$

Sigmoid function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Sigmoid derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$


Backpropagation – simple example

Model:
 $m(z) = -z$
 $g(z) = \exp(z)$
 $h(z) = z + 1$
 $f(z) = \frac{1}{z}$

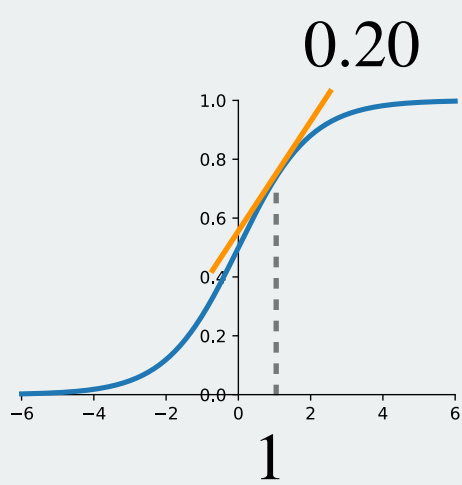
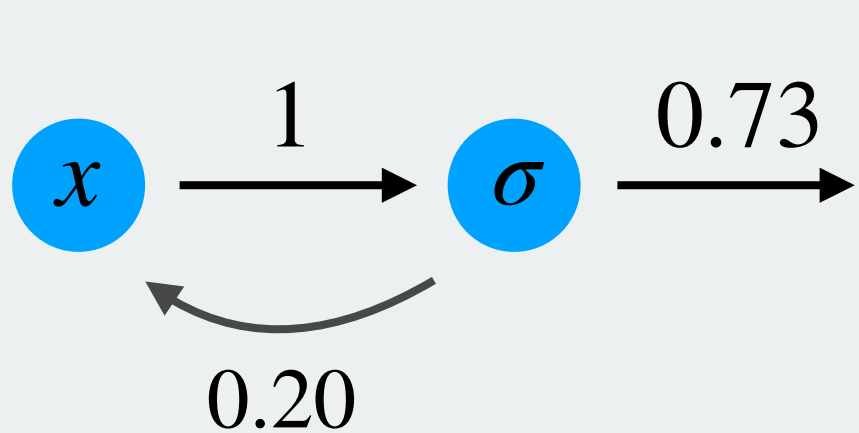
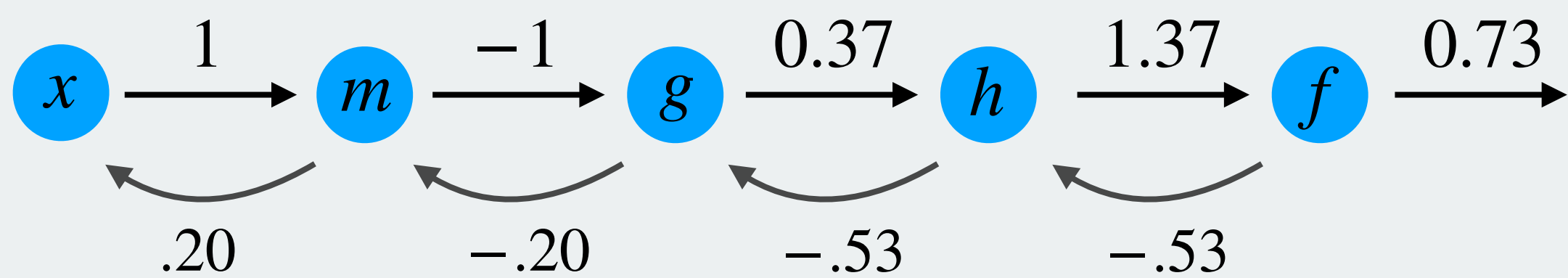
Data:
 $x_0 = 1$
 $x_1 = 1.1$

Q: How does a small nudge in x influence $f(h(g(m(x))))$?

A: Propagate gradients backwards using the chain rule!

Chain rule:
$$\frac{df}{dx} = \frac{df}{dh} \frac{dh}{dg} \frac{dg}{dm} \frac{dm}{dx}$$

Model derivatives:
 $m'(z) = -1$
 $g'(z) = \exp(z)$
 $h'(z) = 1$
 $f'(z) = -\frac{1}{z^2}$



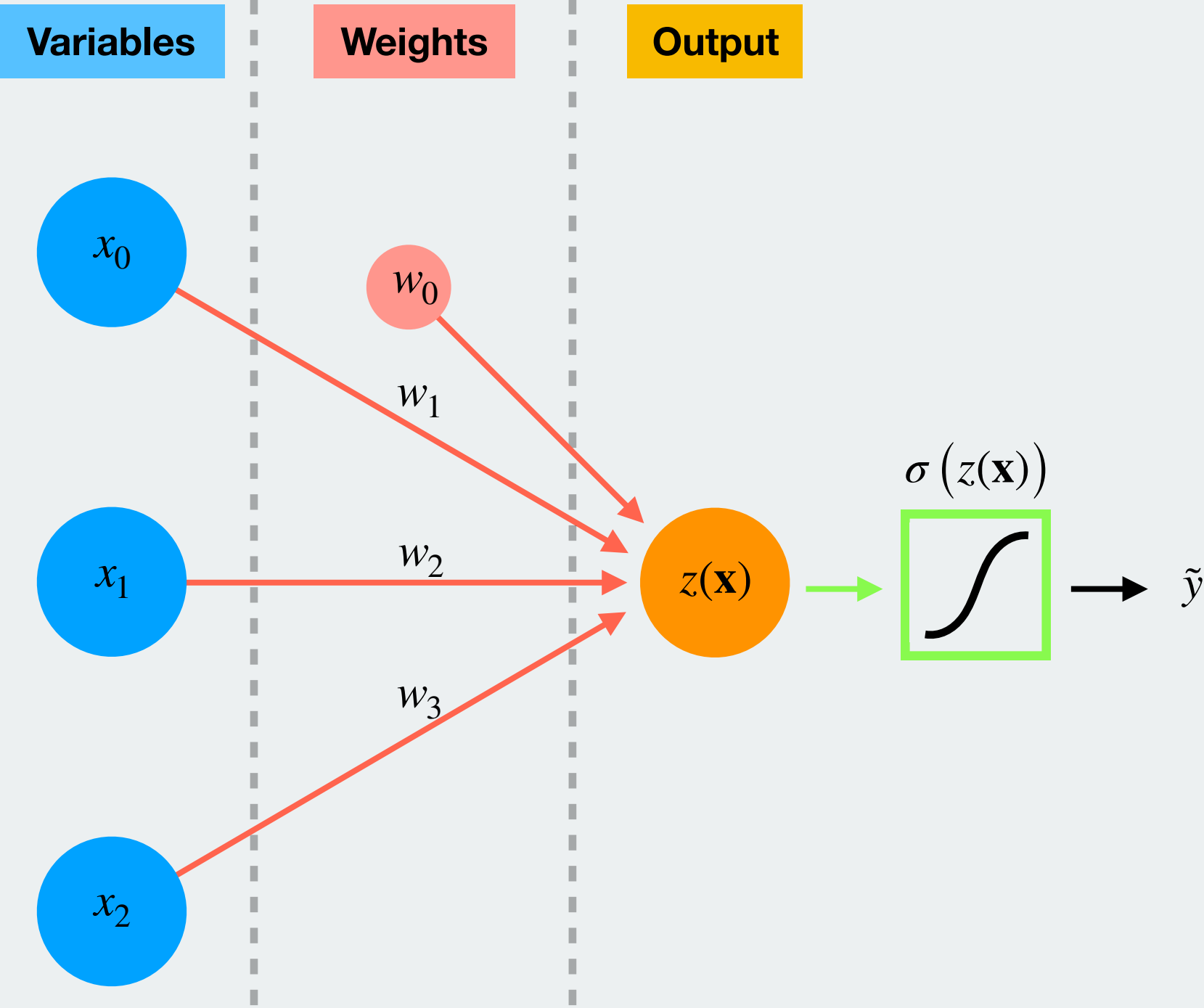
Sigmoid function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

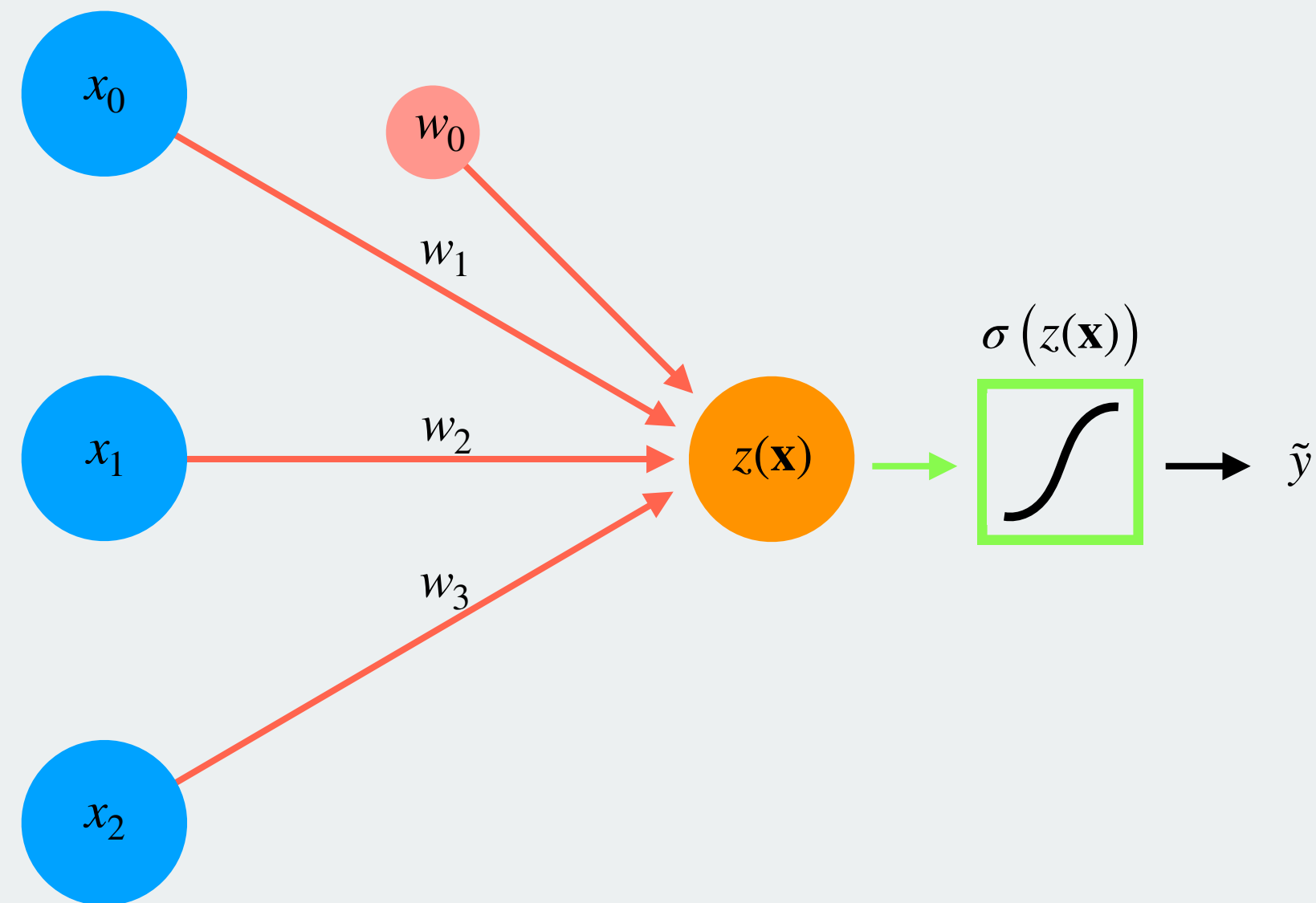
Sigmoid derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Backpropagation – on a neural network



Backpropagation – on a neural network



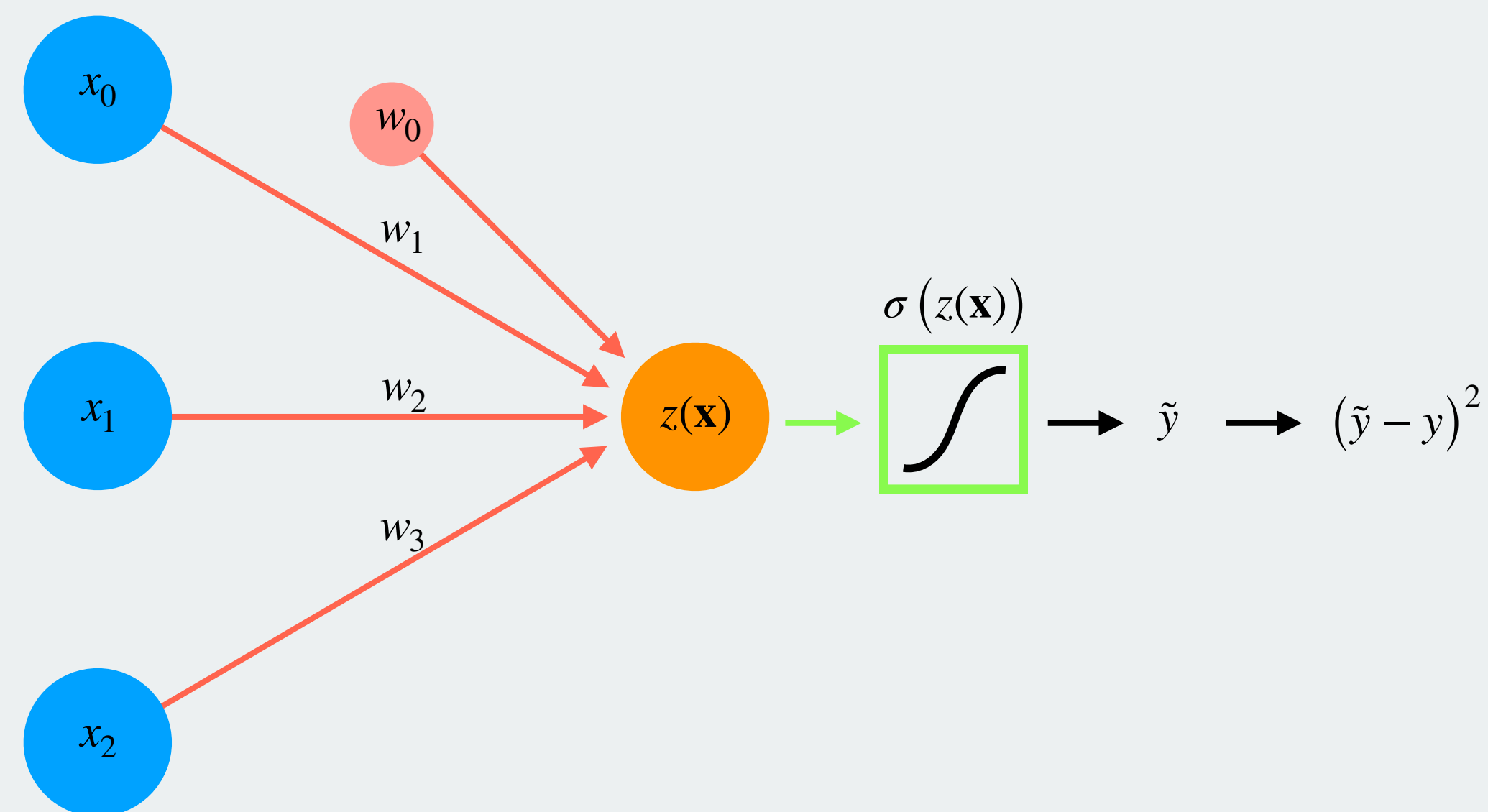
Model:

$$w_0 + x_0 w_1 + x_1 w_2 + x_2 w_3 = z(\mathbf{x})$$

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

$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

Backpropagation – on a neural network



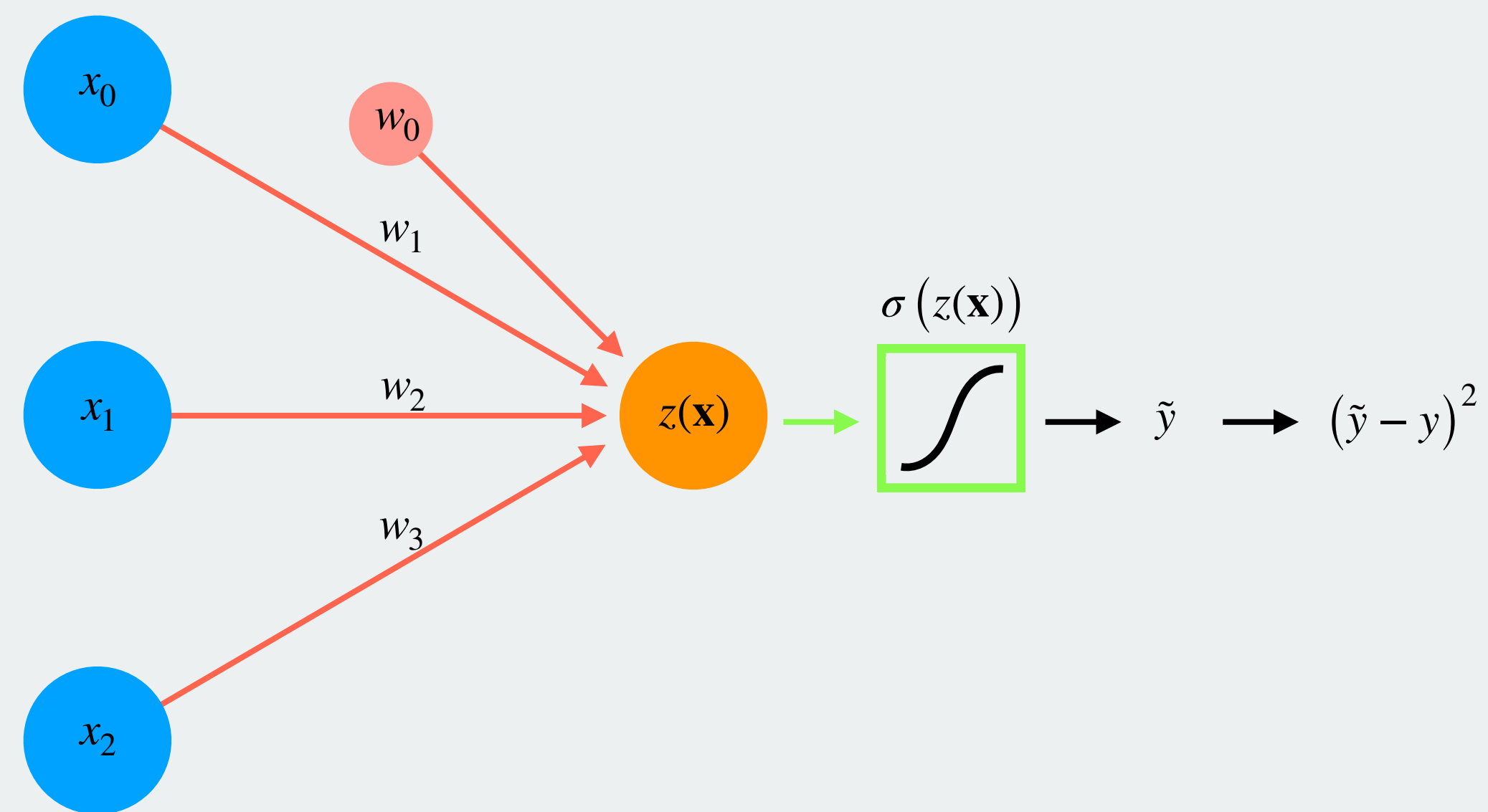
Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$

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

$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

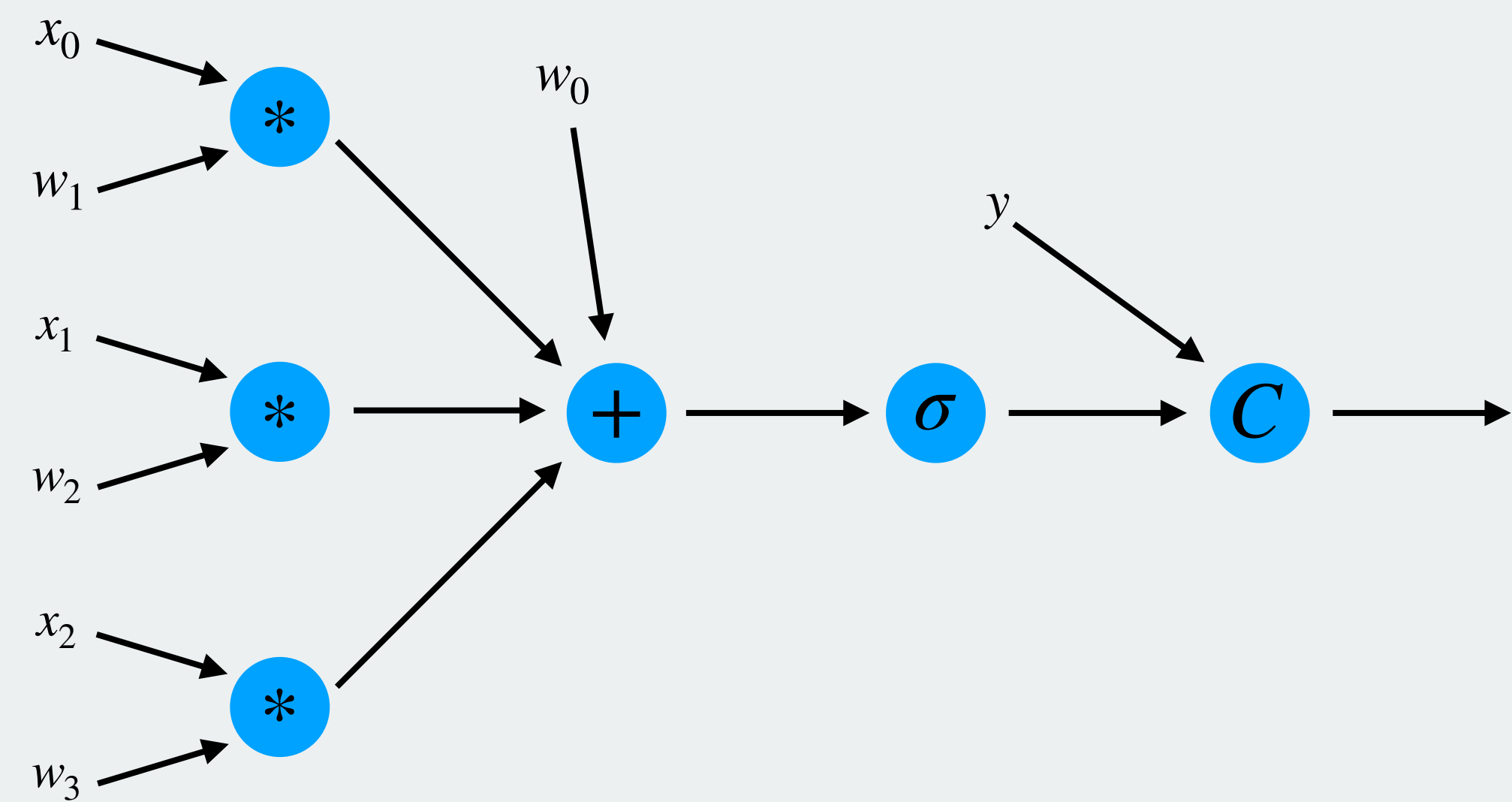
Backpropagation – on a neural network



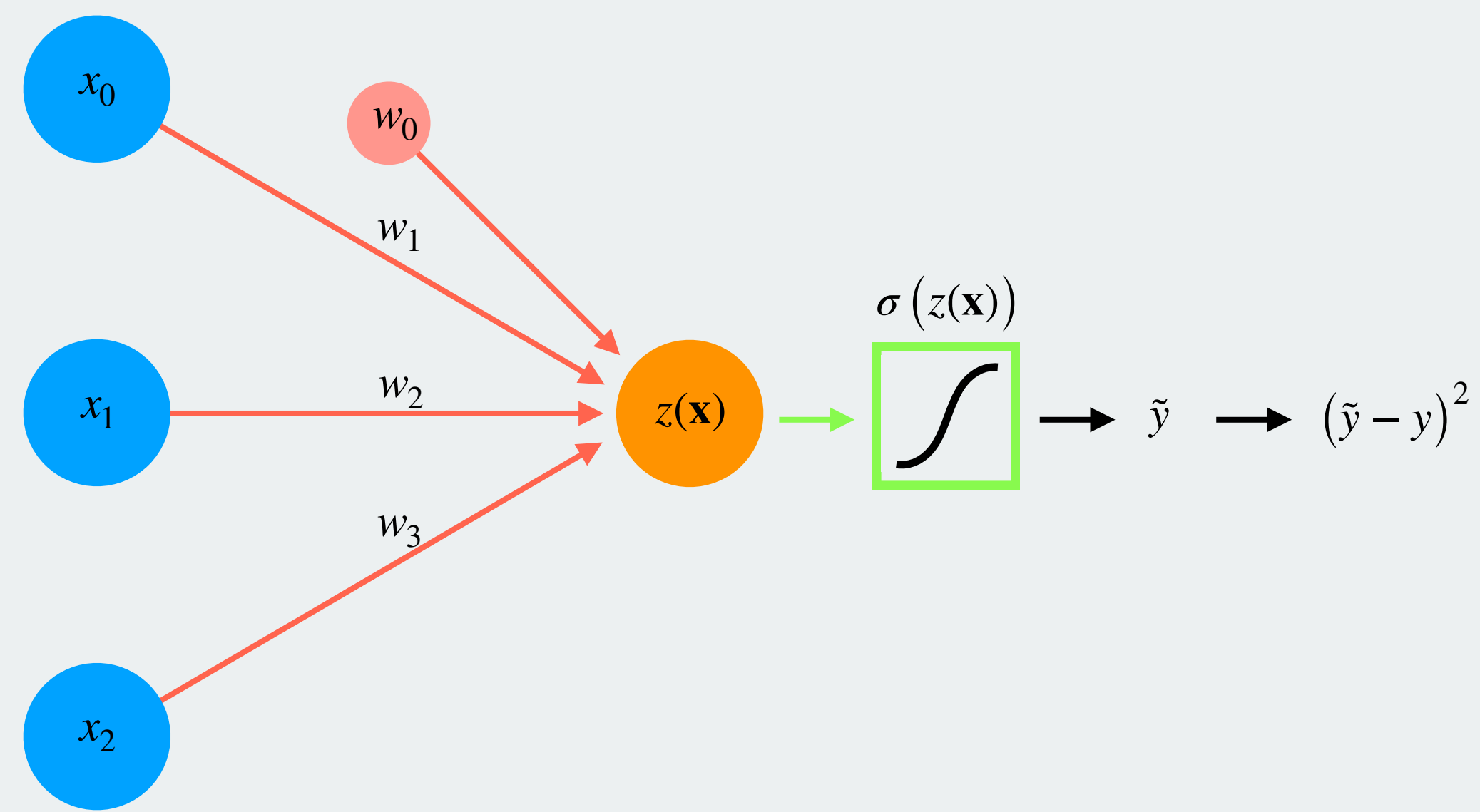
Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

As a computational graph:



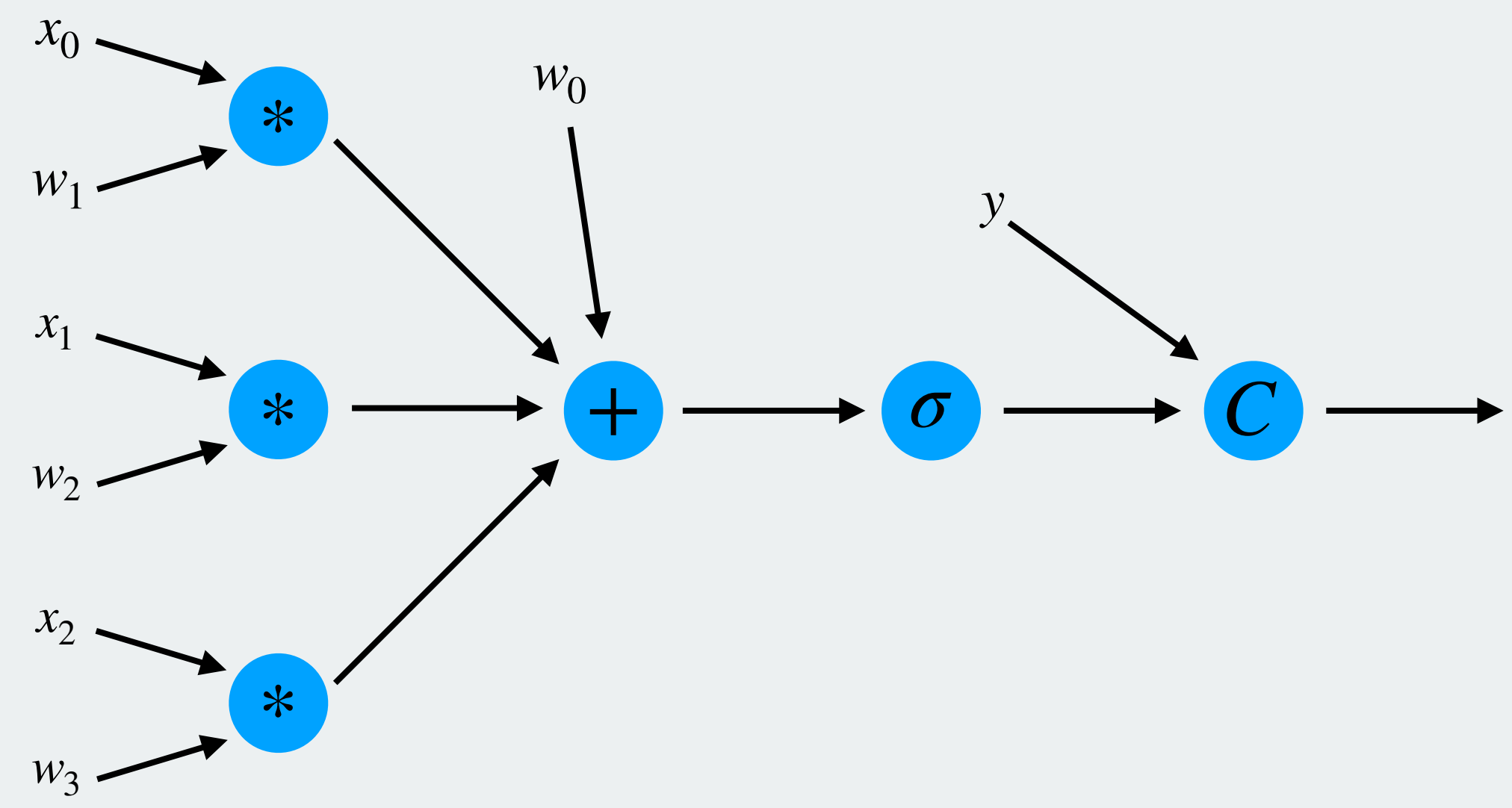
Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

As a computational graph:



Weights:

$\mathbf{b} = [-2]$

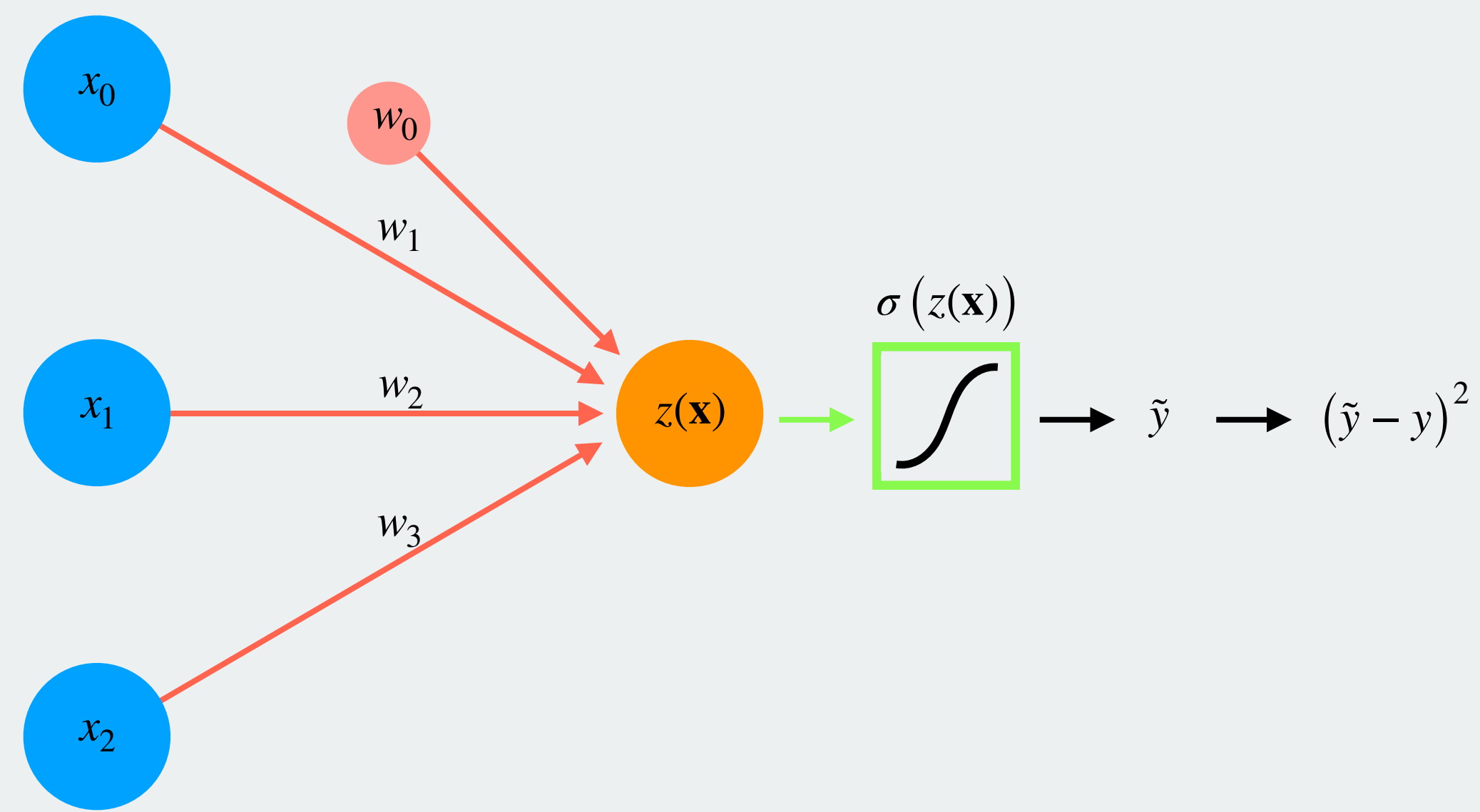
$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$

Data:

$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$

$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$

Backpropagation – on a neural network

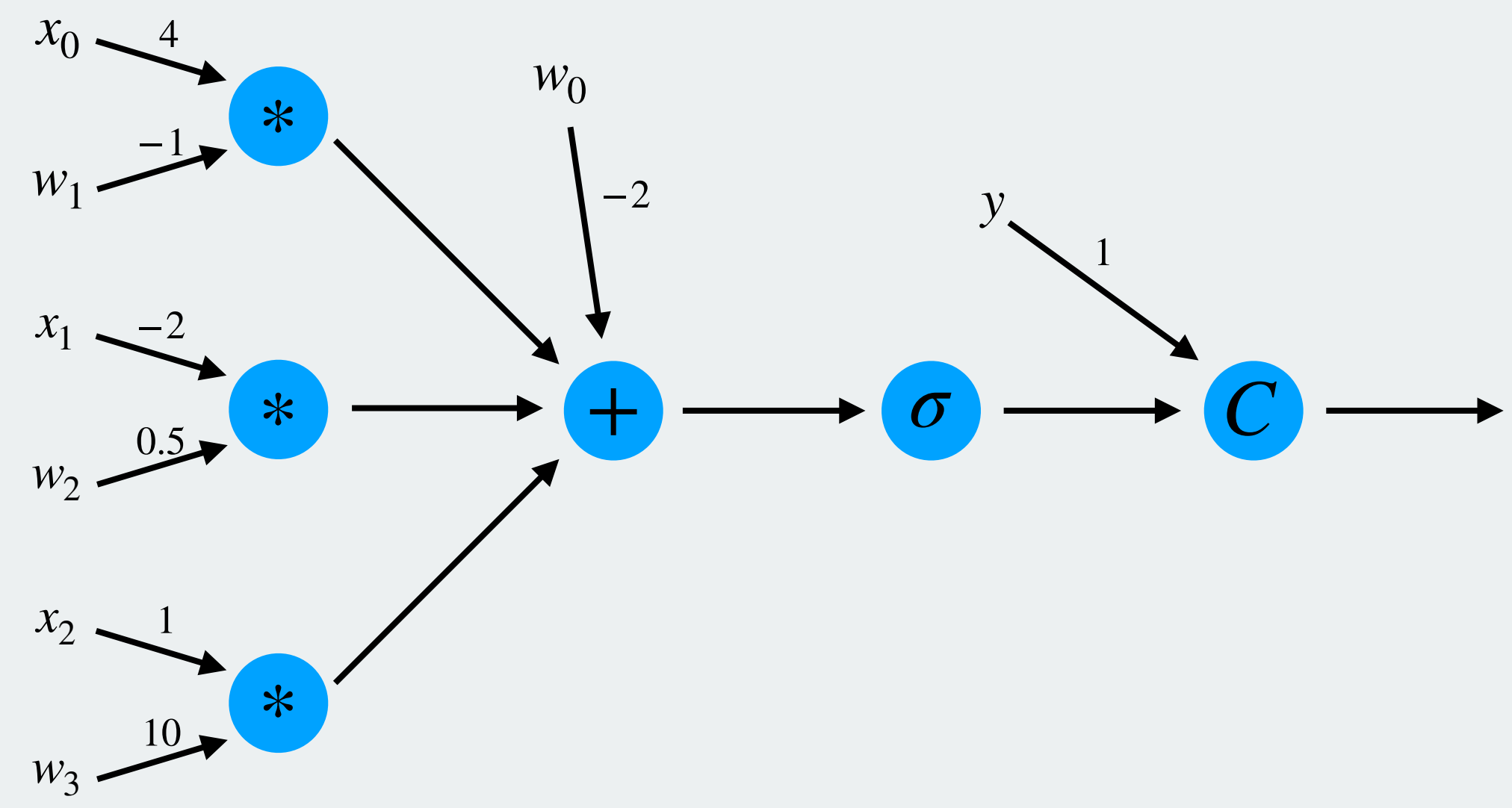


Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

As a computational graph:

Forward pass



Weights:

$$\mathbf{b} = [-2]$$

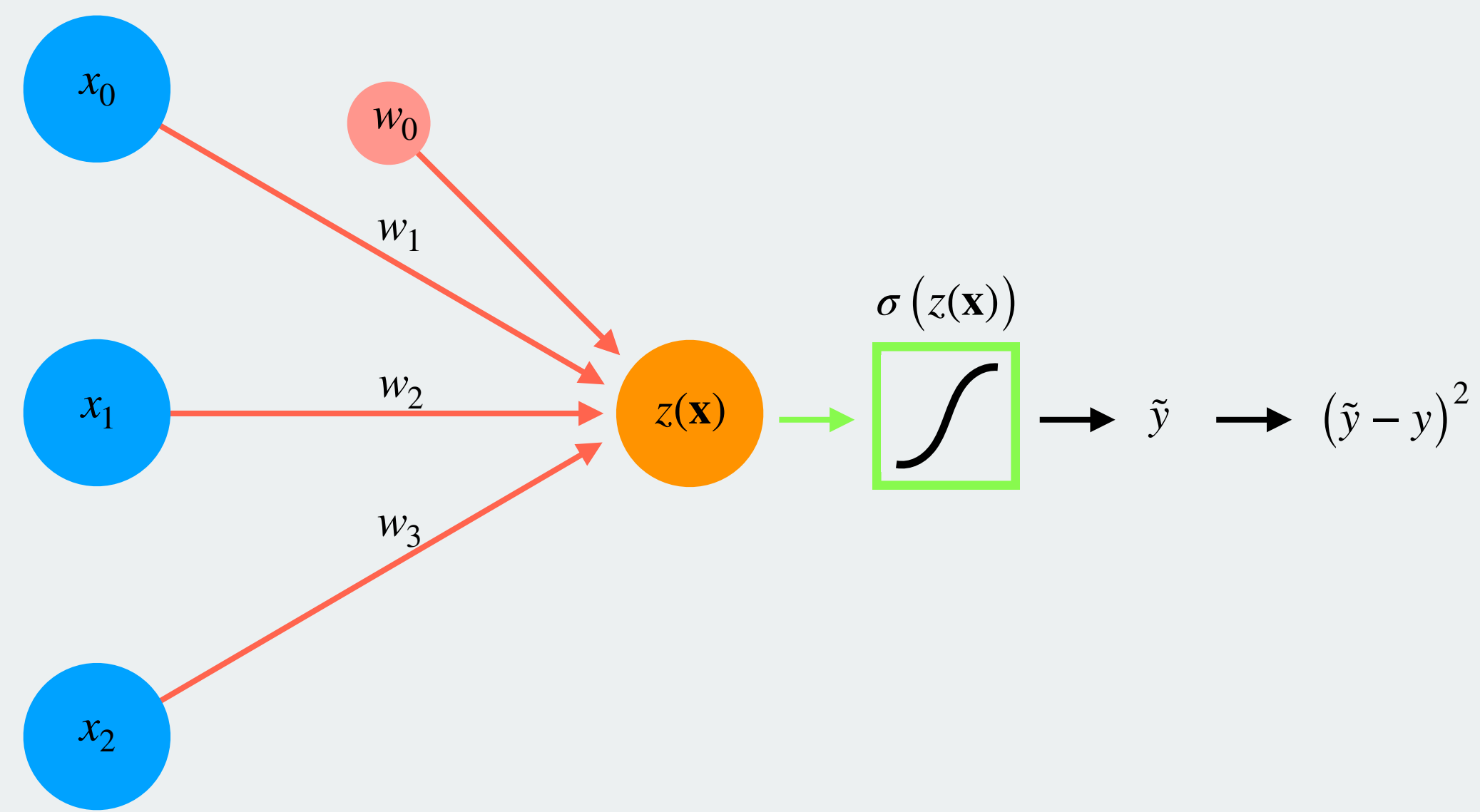
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network

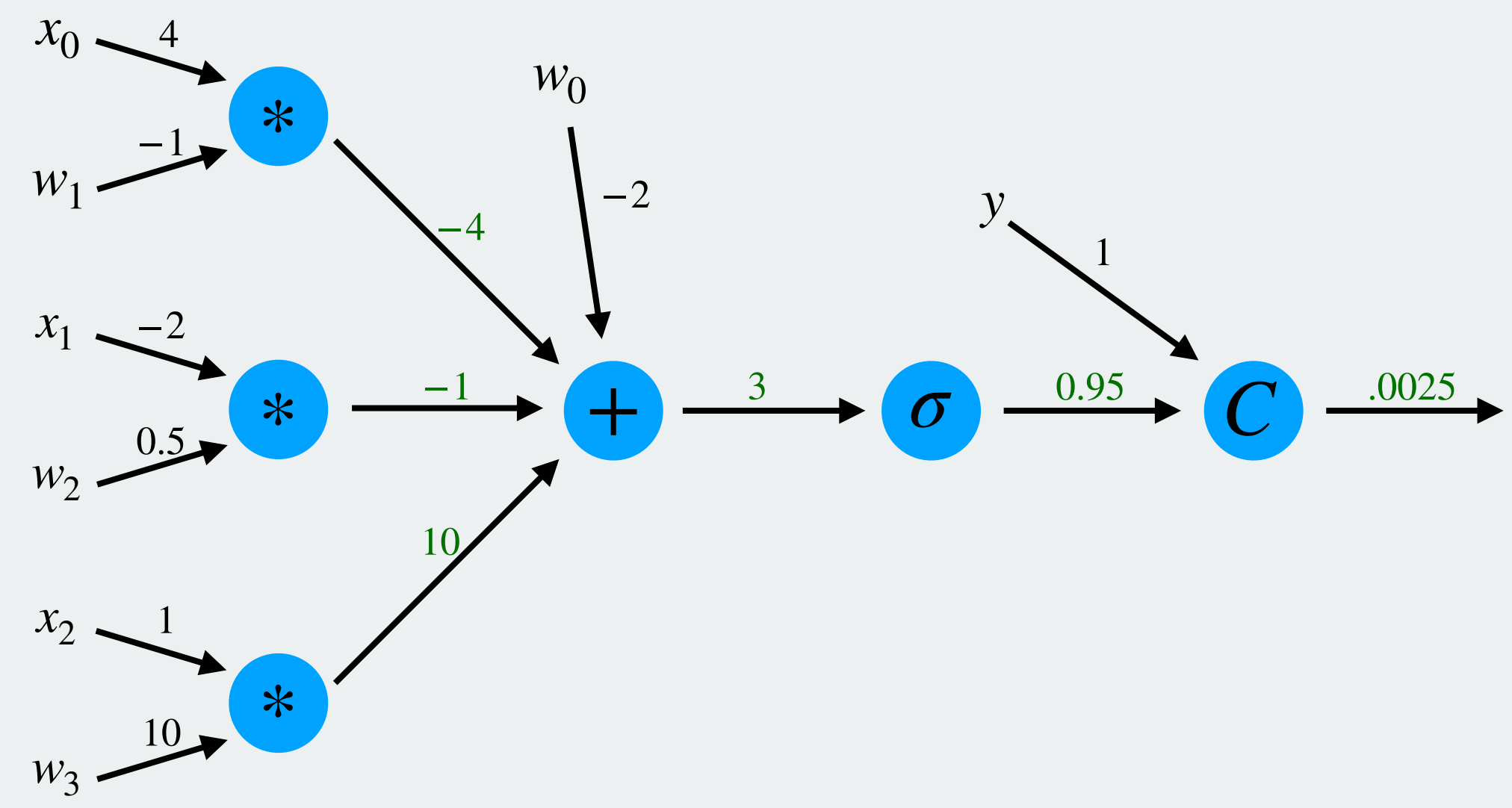


Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

As a computational graph:

Forward pass



Weights:

$\mathbf{b} = [-2]$

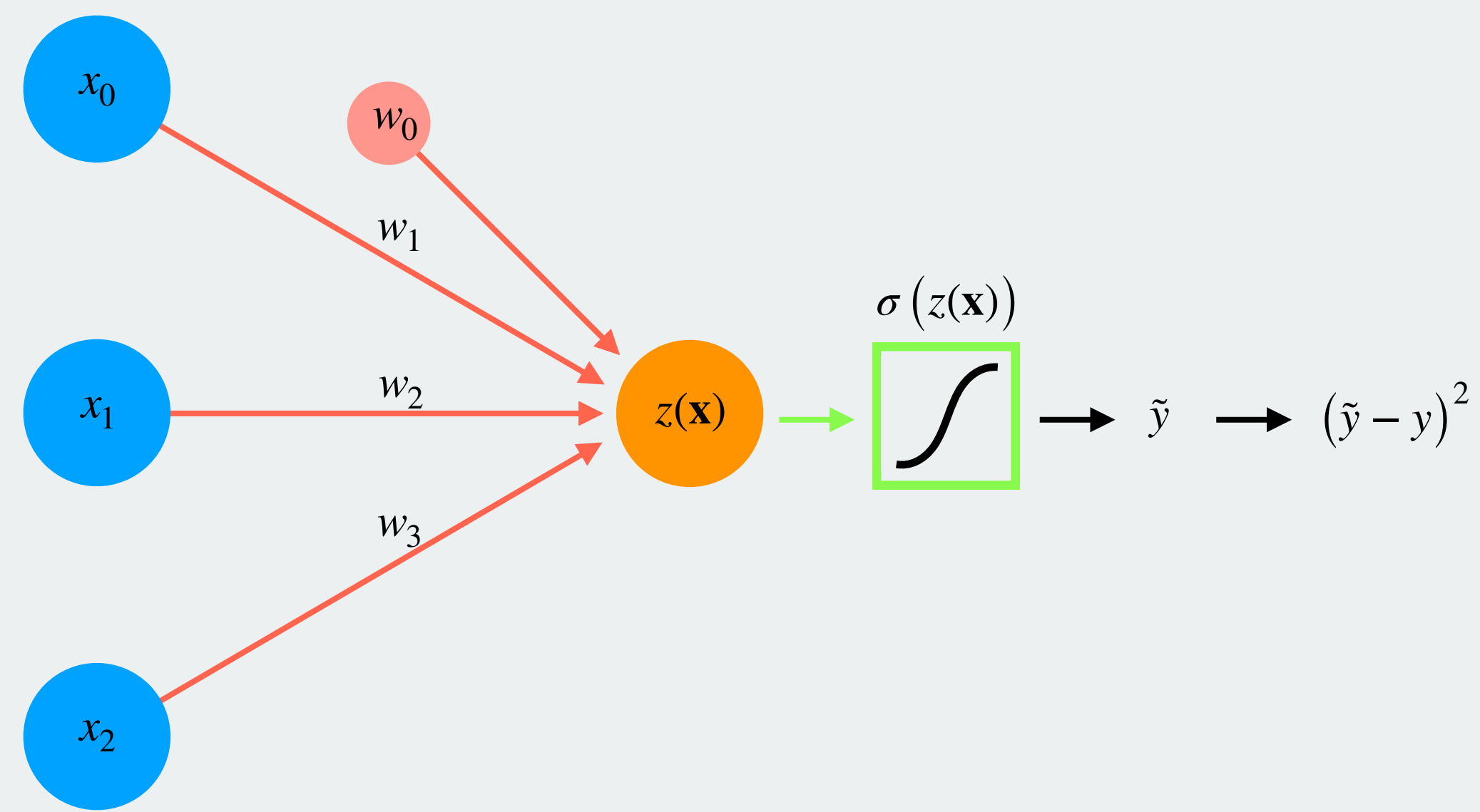
$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$

Data:

$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$

$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$

Backpropagation – on a neural network



Model:

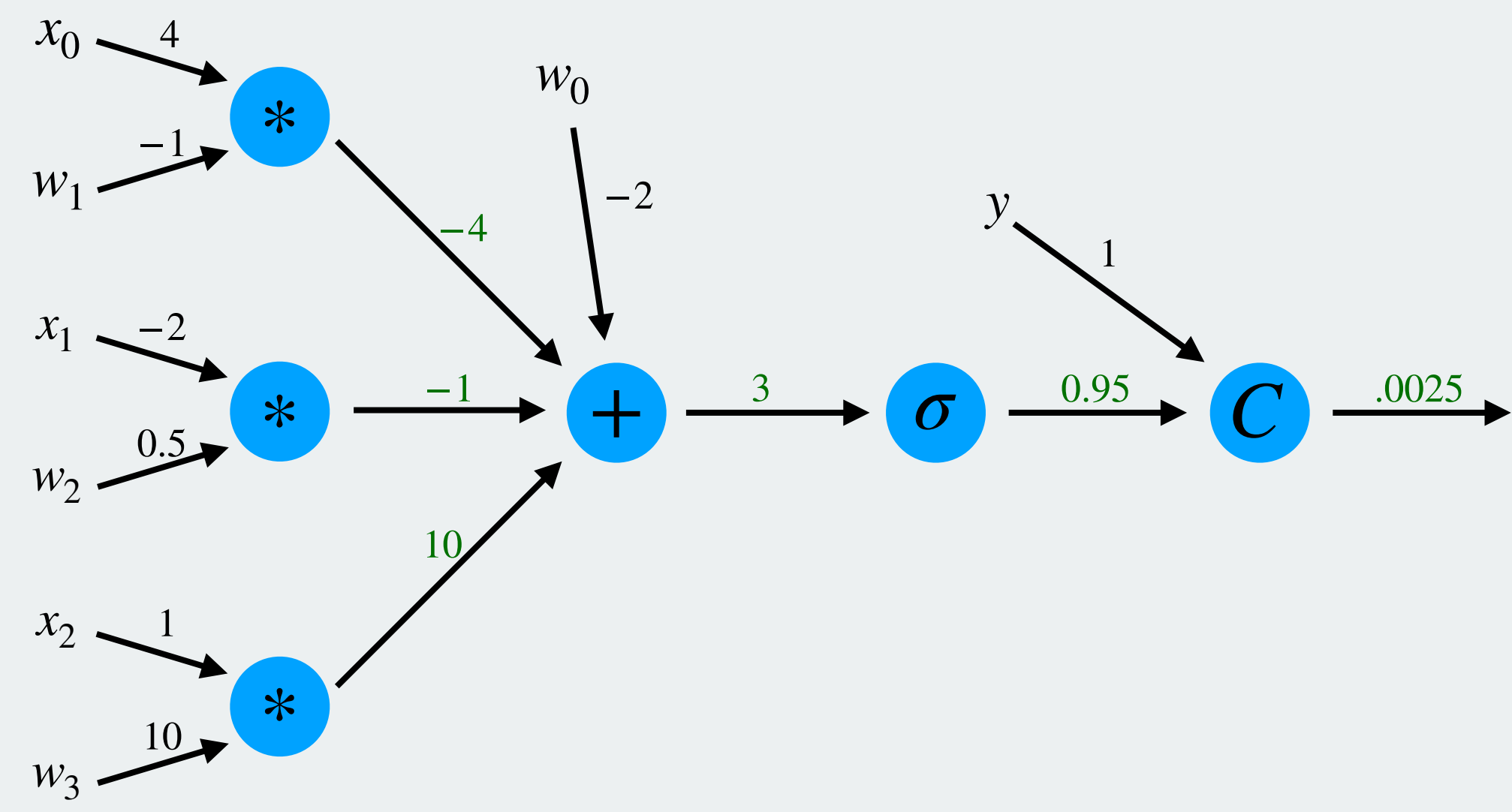
$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

Model derivatives:

$$C'(\tilde{y}, y) = ?$$

As a computational graph:

Backward pass



Weights:

$$\mathbf{b} = [-2]$$

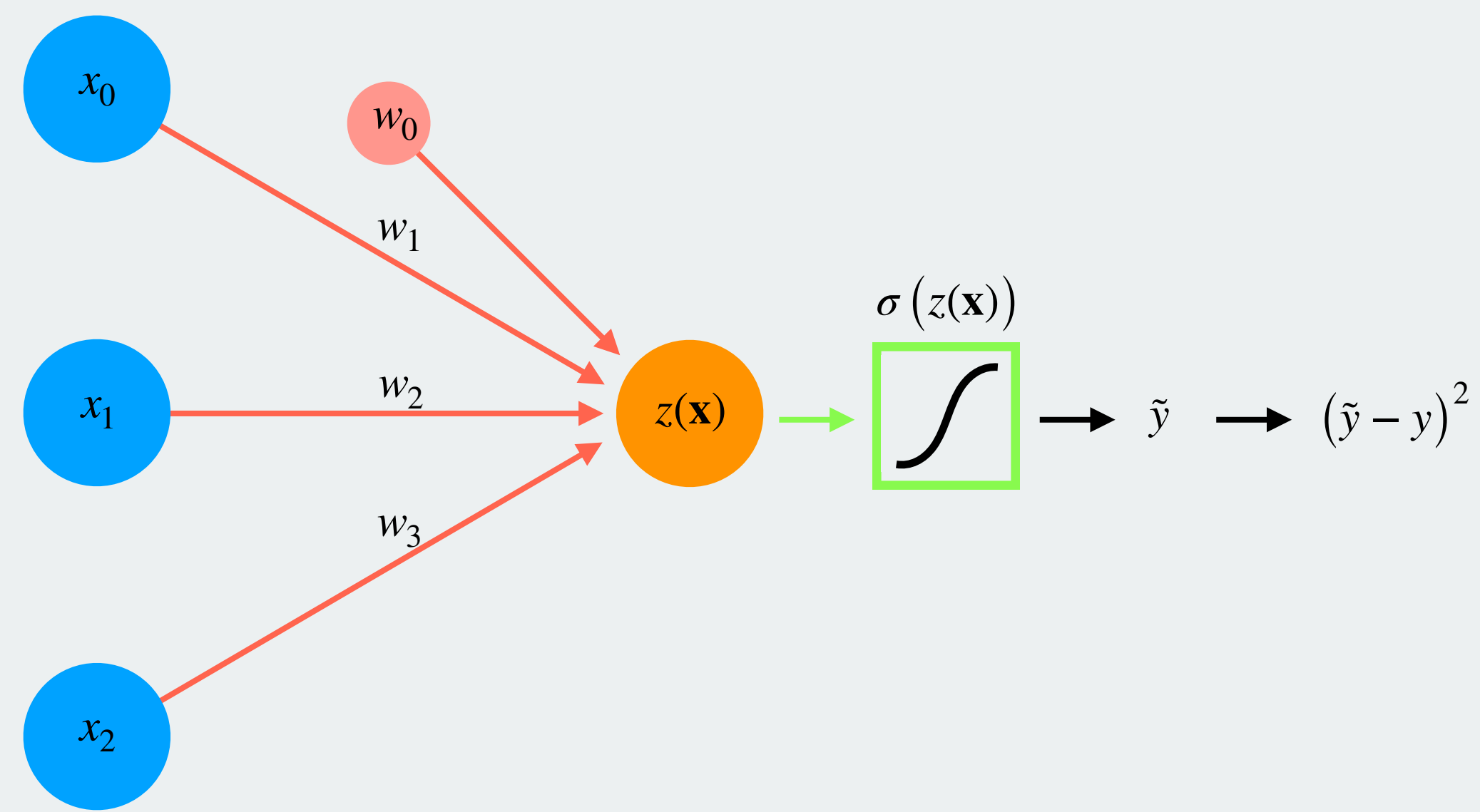
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

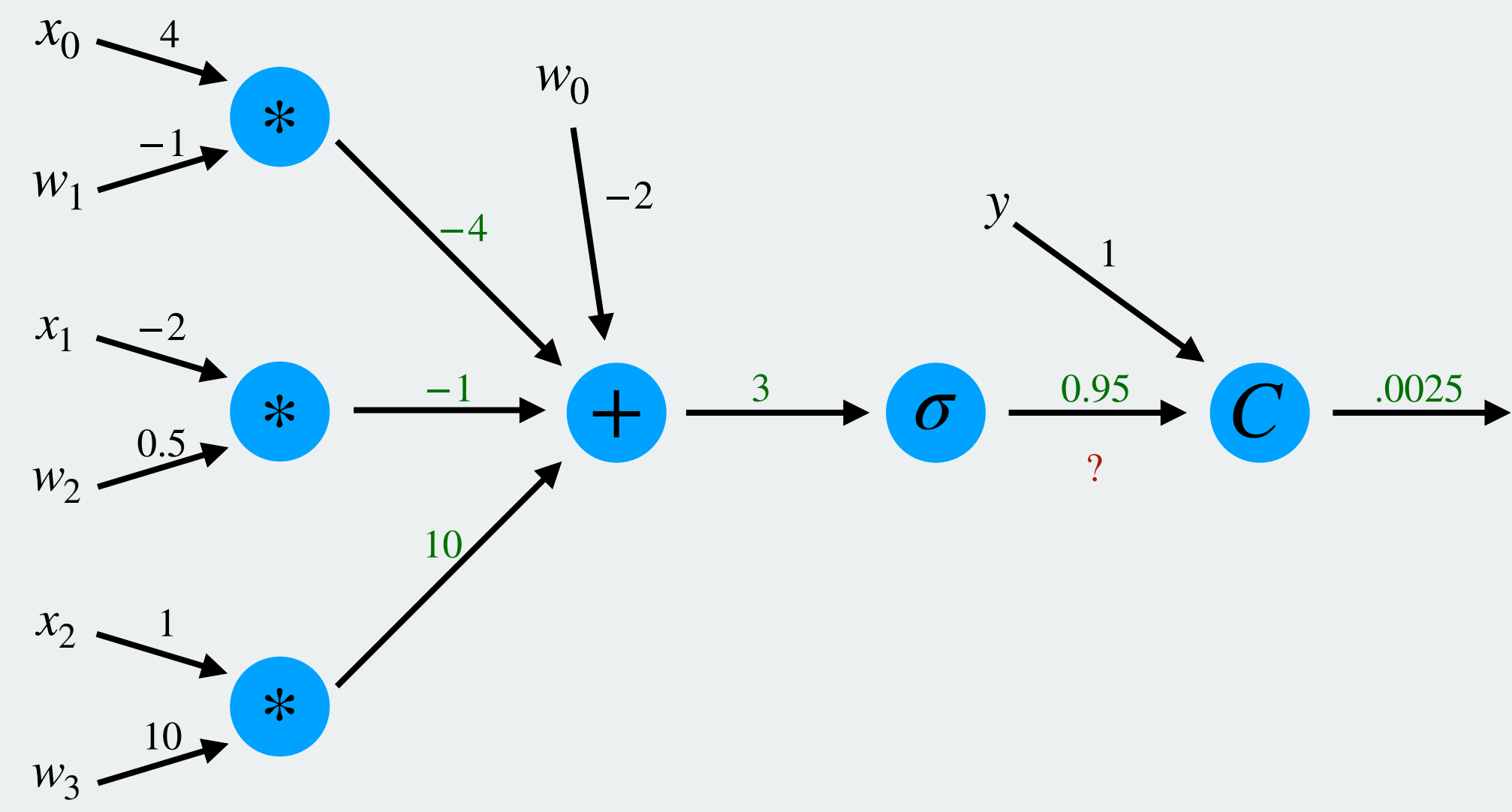
$$w_0 + x_0 w_1 + x_1 w_2 + x_2 w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

Model derivatives:

$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



Weights:

$$\mathbf{b} = [-2]$$

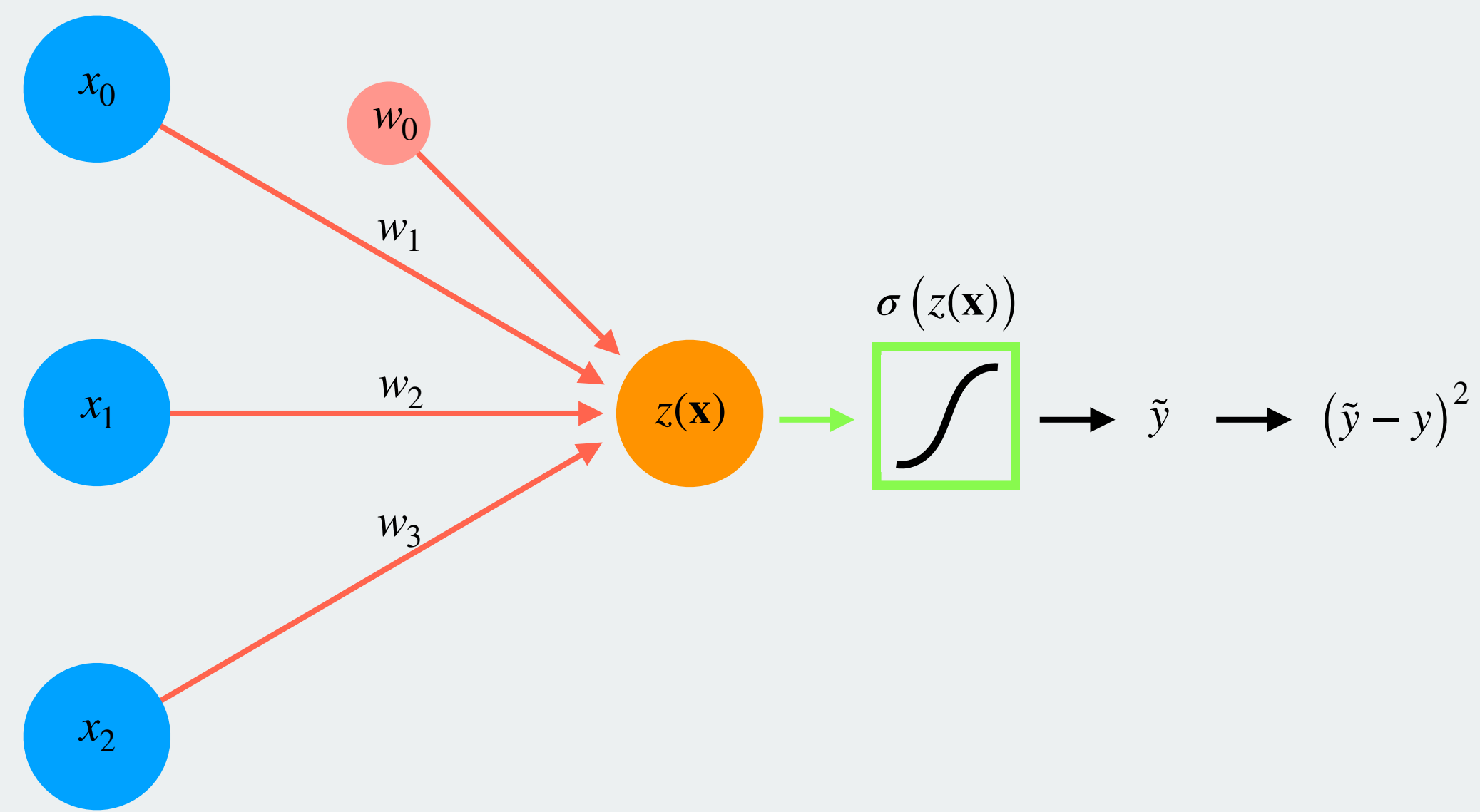
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

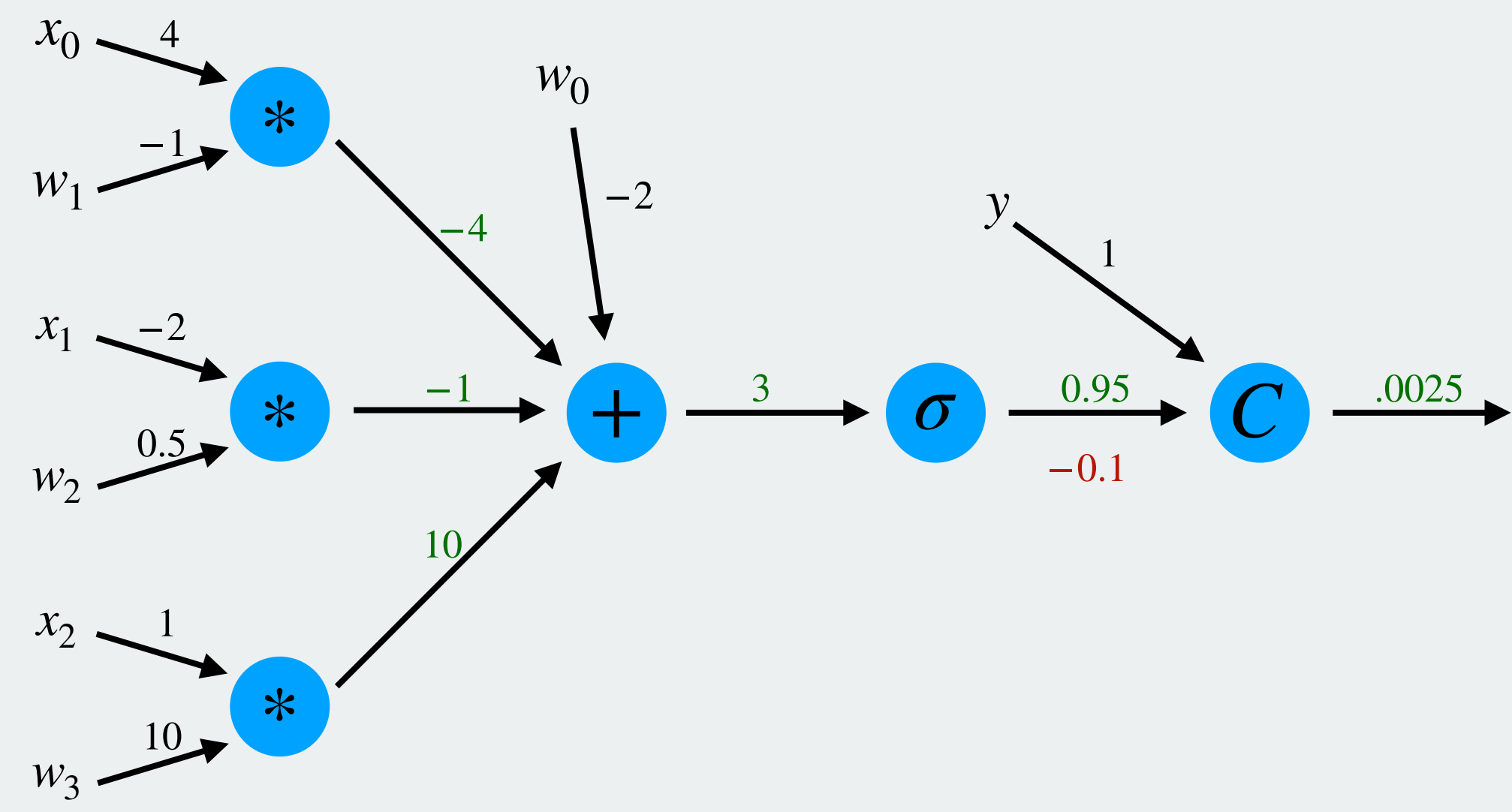
$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

Model derivatives:

$$C'(\tilde{y}, y) = 2 (\tilde{y} - y)$$

As a computational graph:

Backward pass



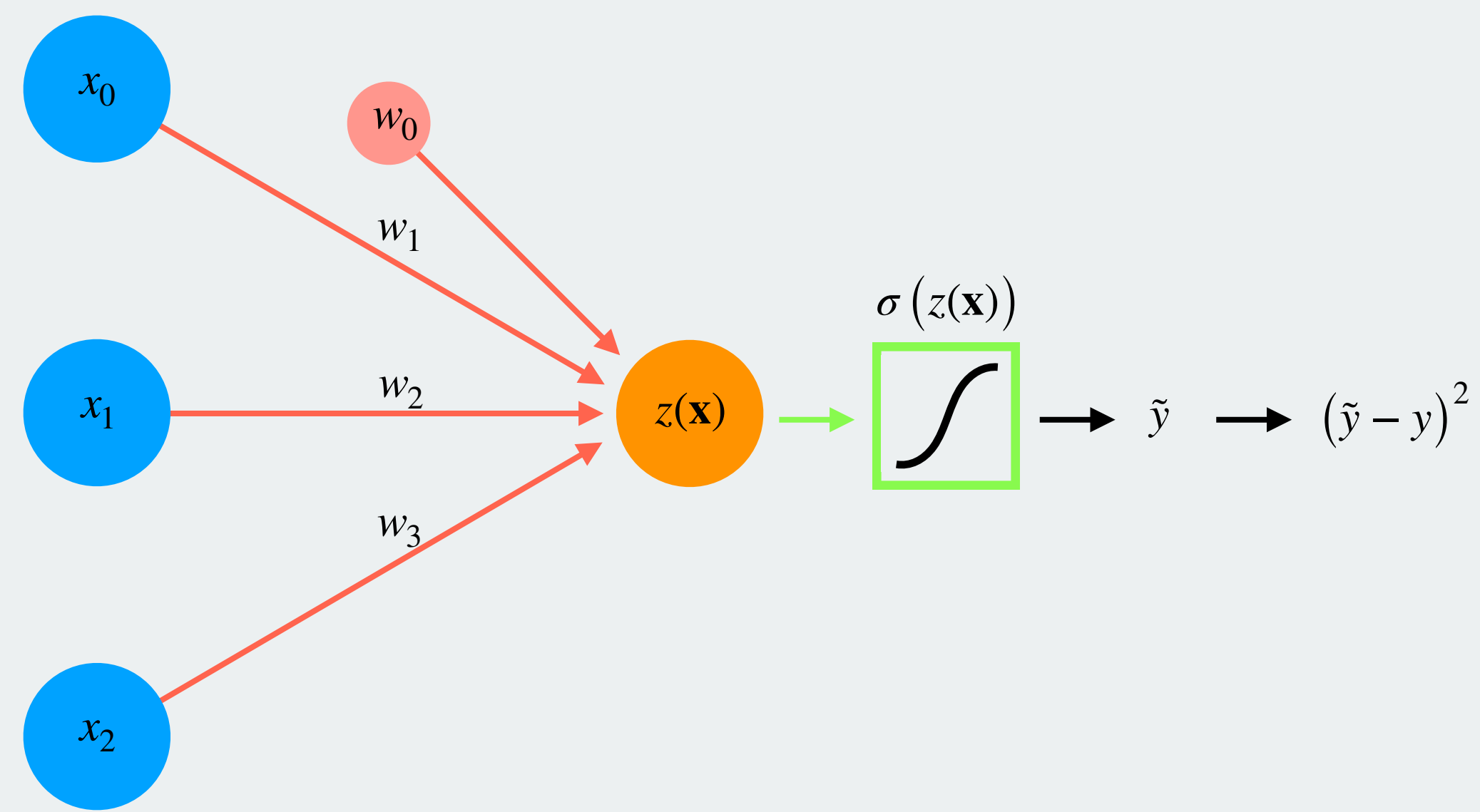
Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

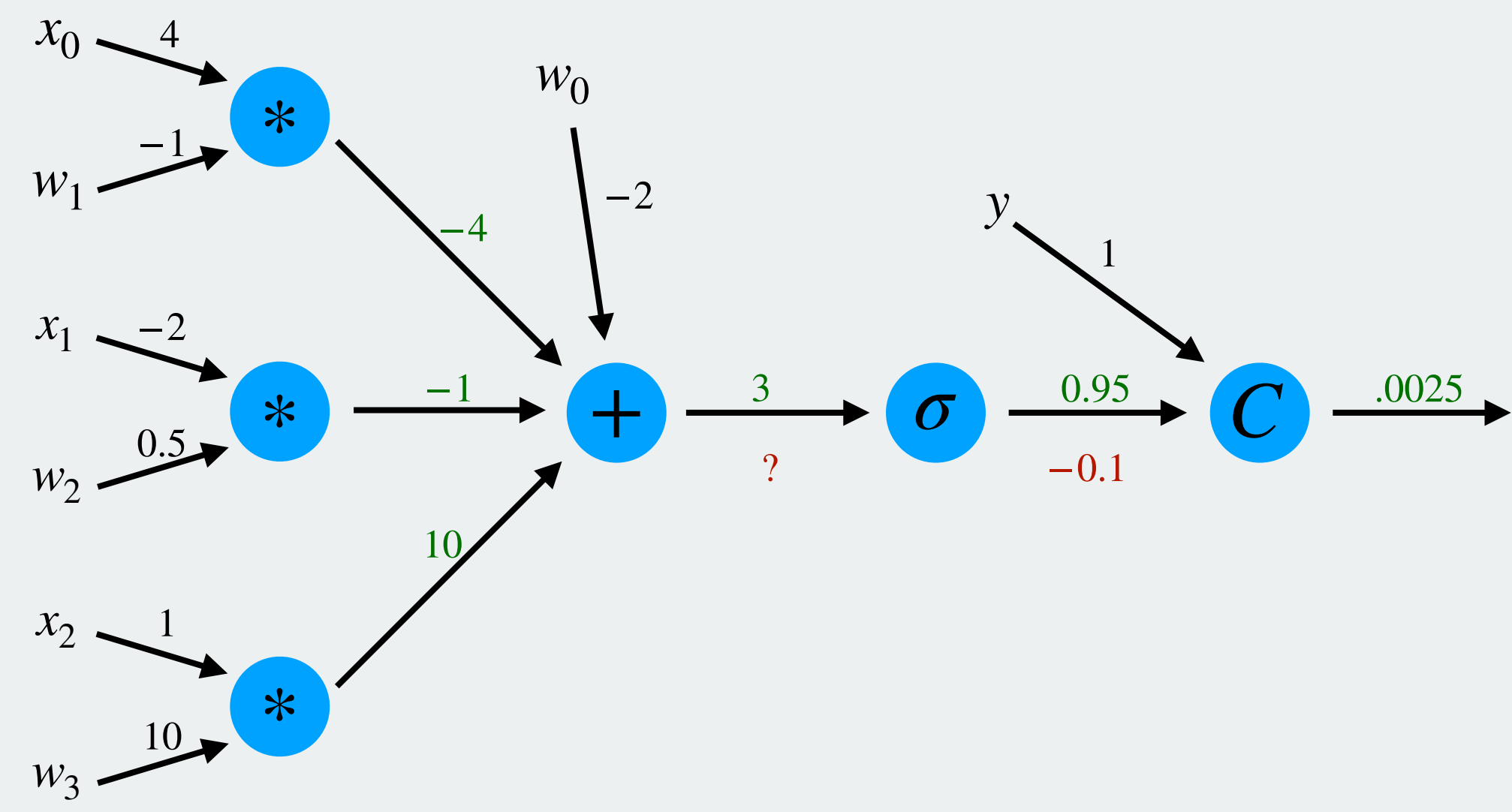
$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

Model derivatives:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



Weights:

$\mathbf{b} = [-2]$

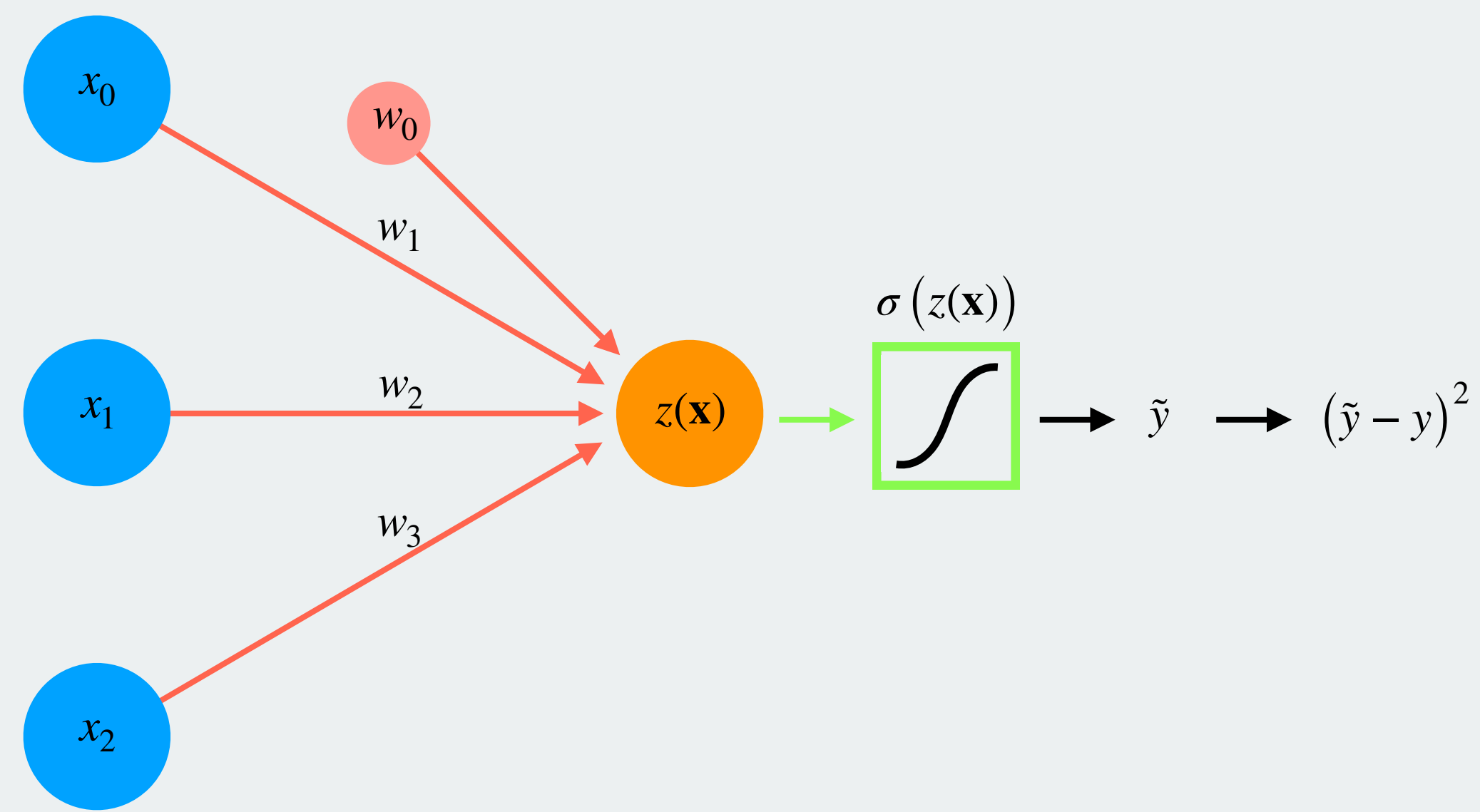
$\mathbf{W} = [-1 \quad 0.5 \quad 10]$

Data:

$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$

$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

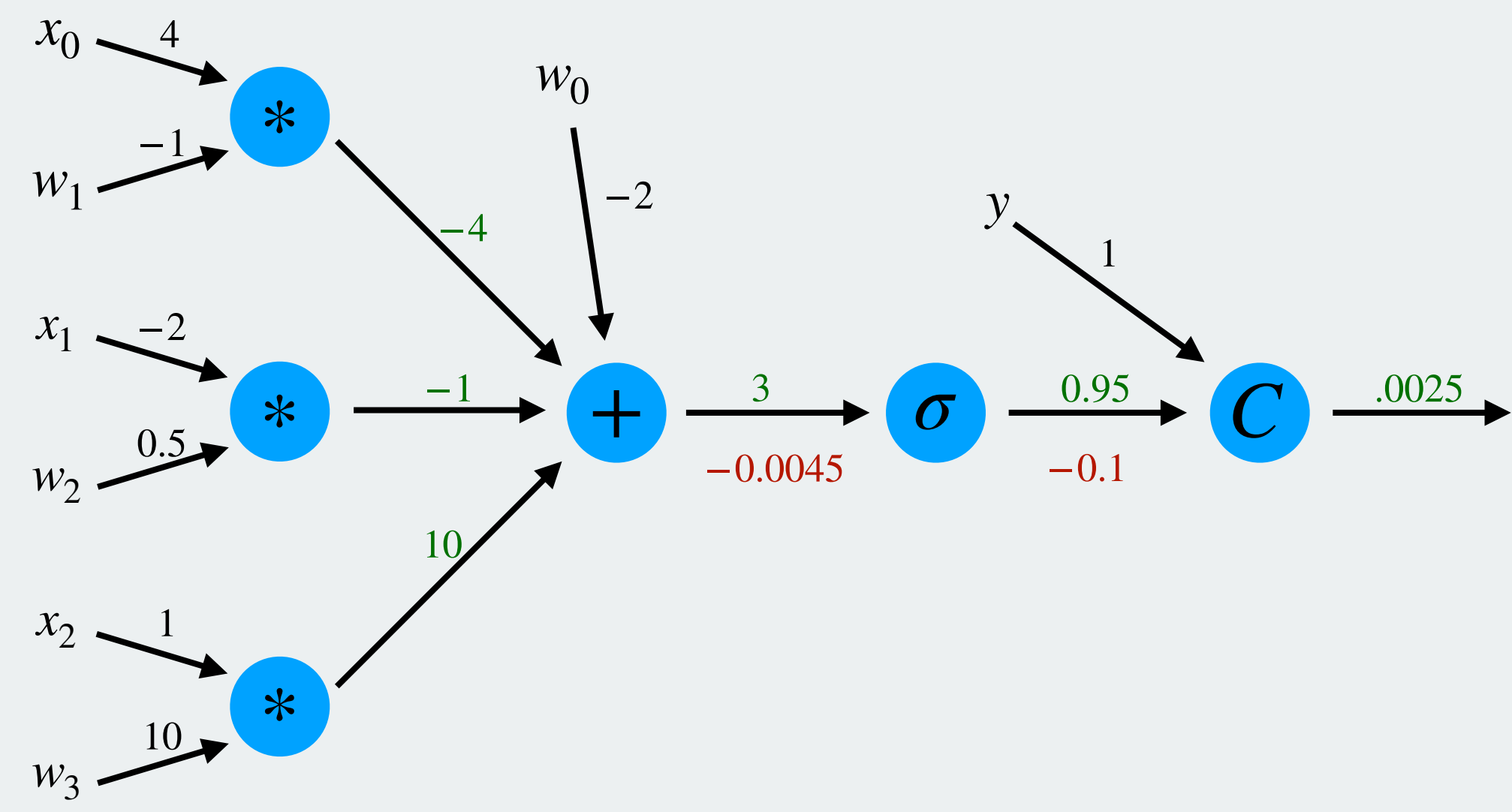
Model derivatives:

The + gate is like a function:

$$y = \sum_{i=0}^N z_n$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



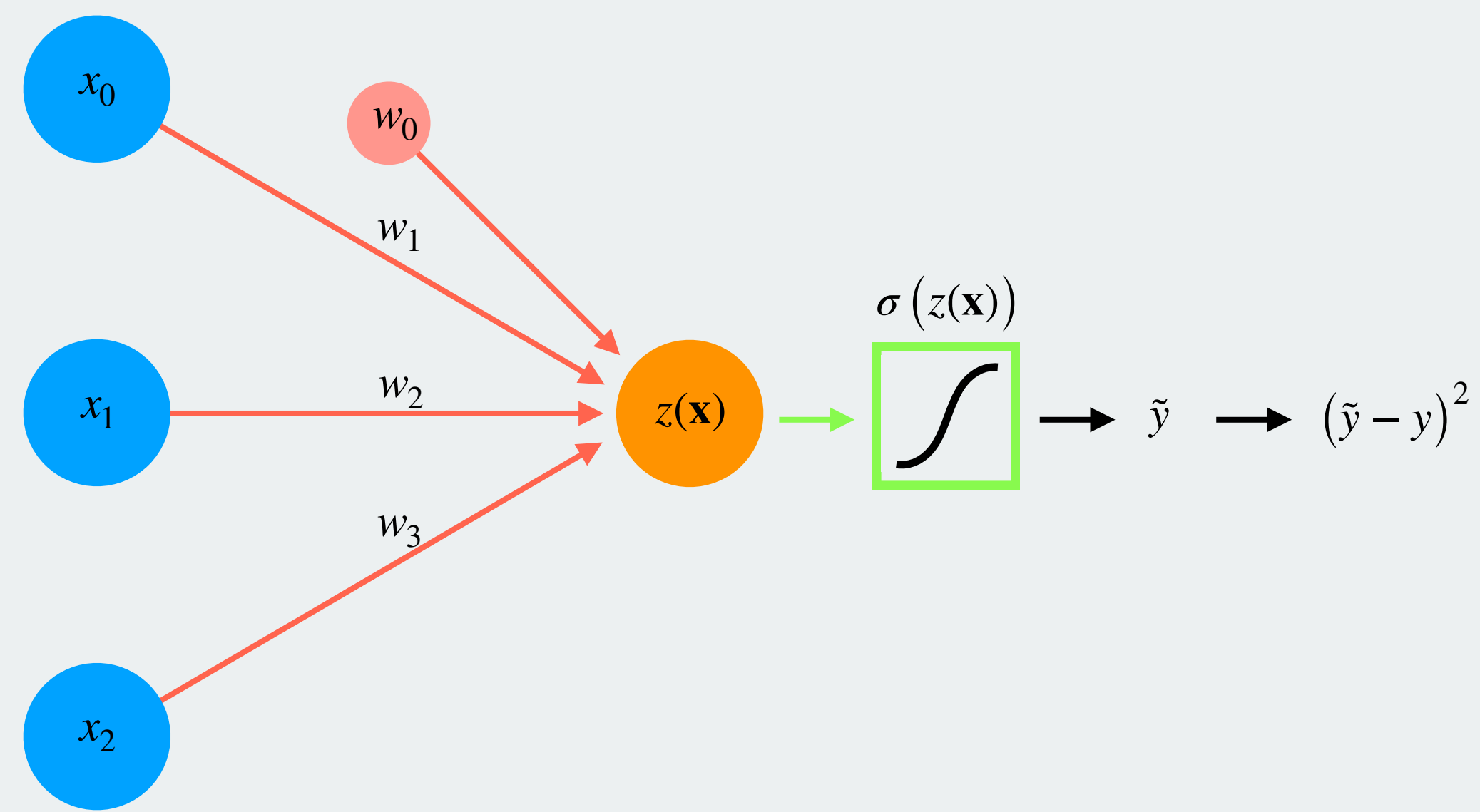
Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

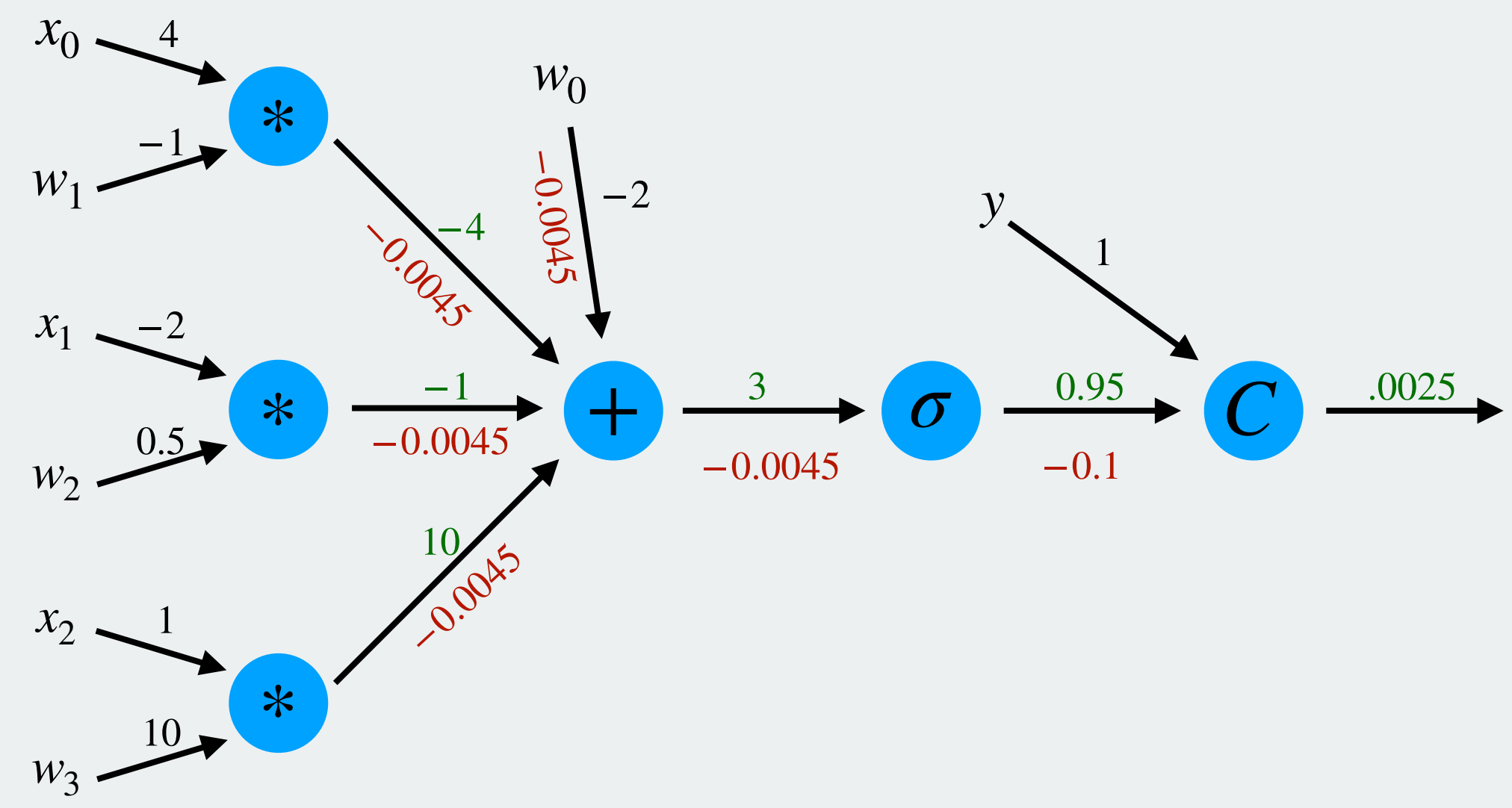
Model derivatives:

The + gate is like a function:

$$y = \sum_{i=0}^N z_n$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



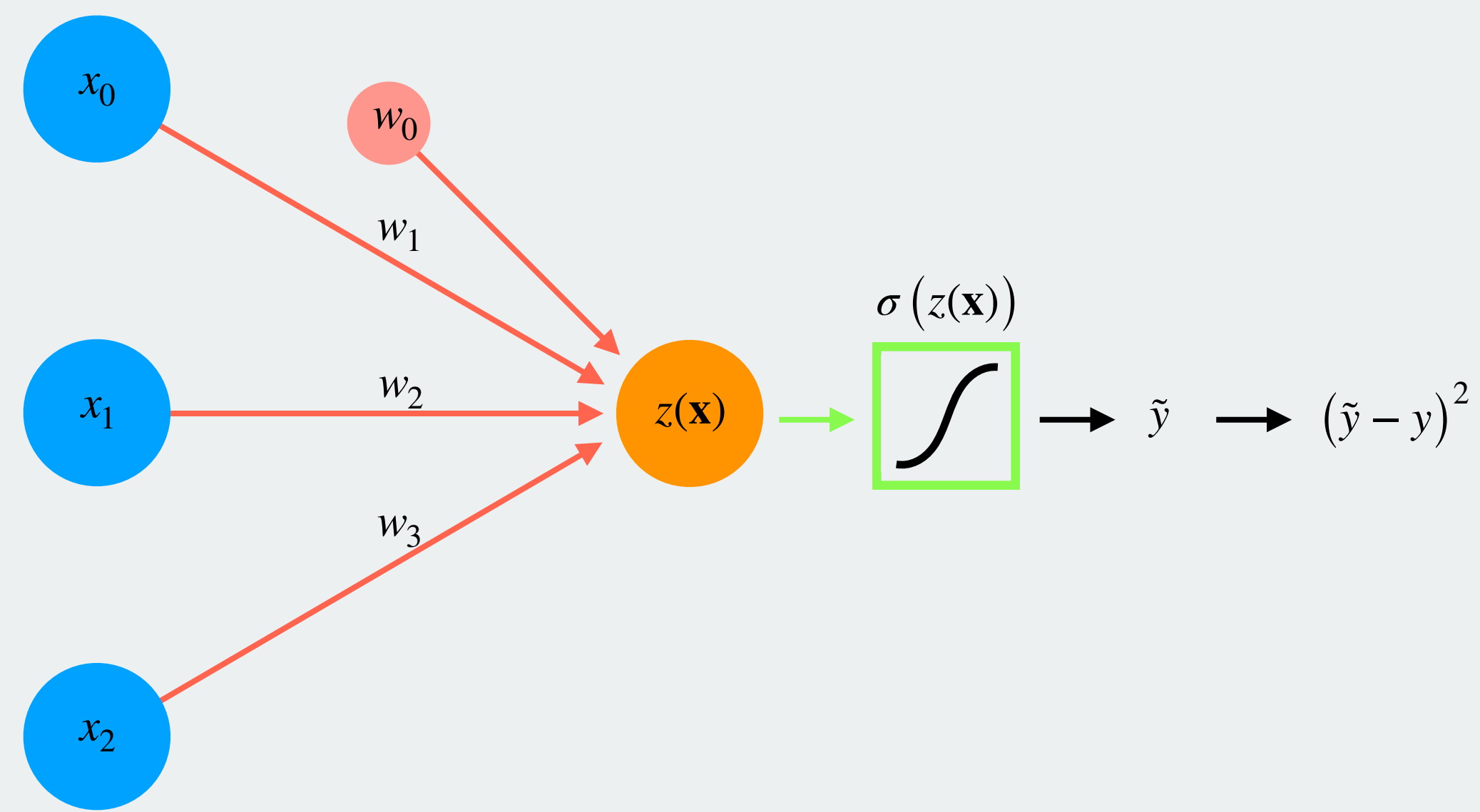
Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

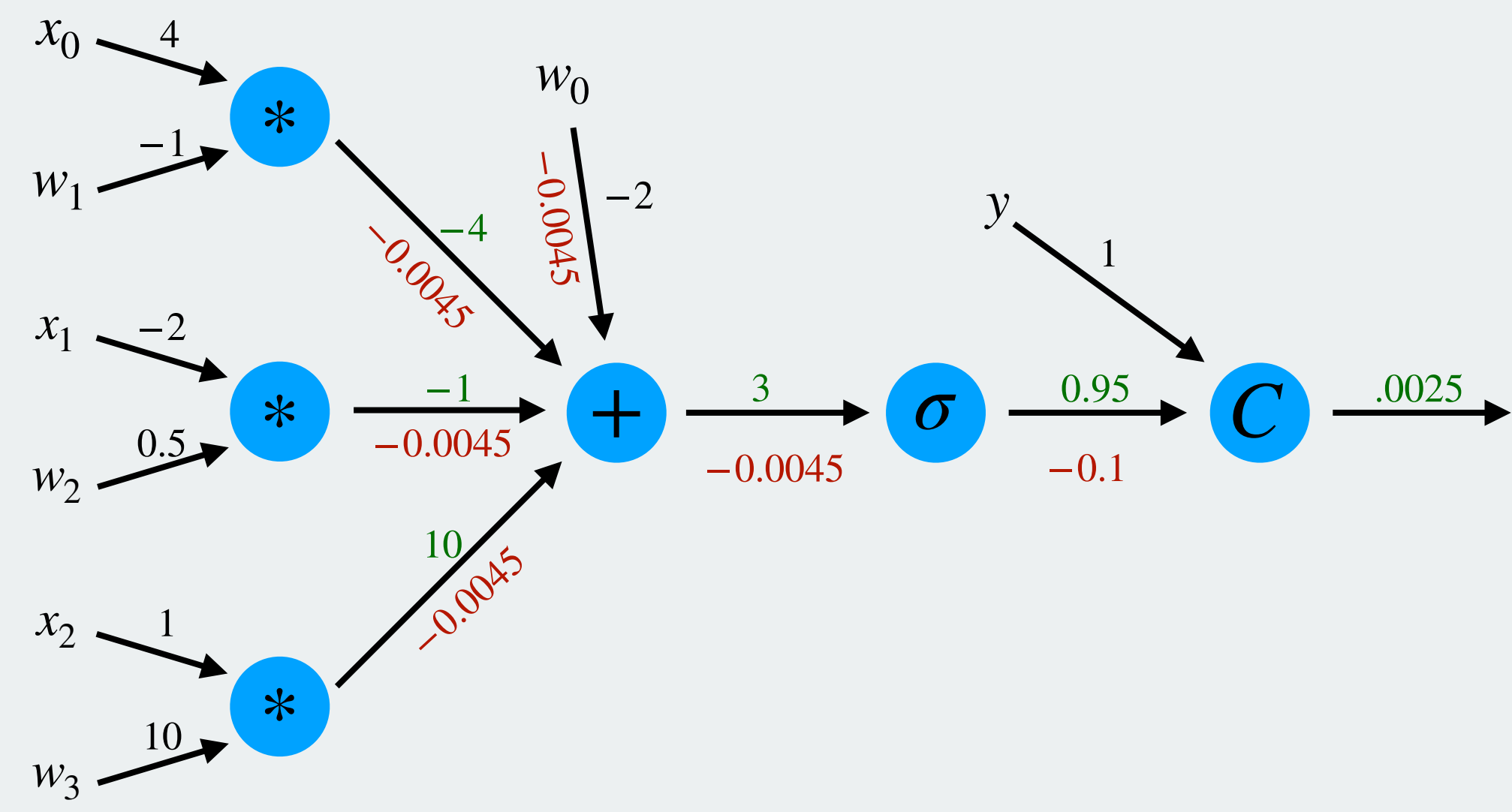
Model derivatives:

Each branch is a function:

$$y = wx$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



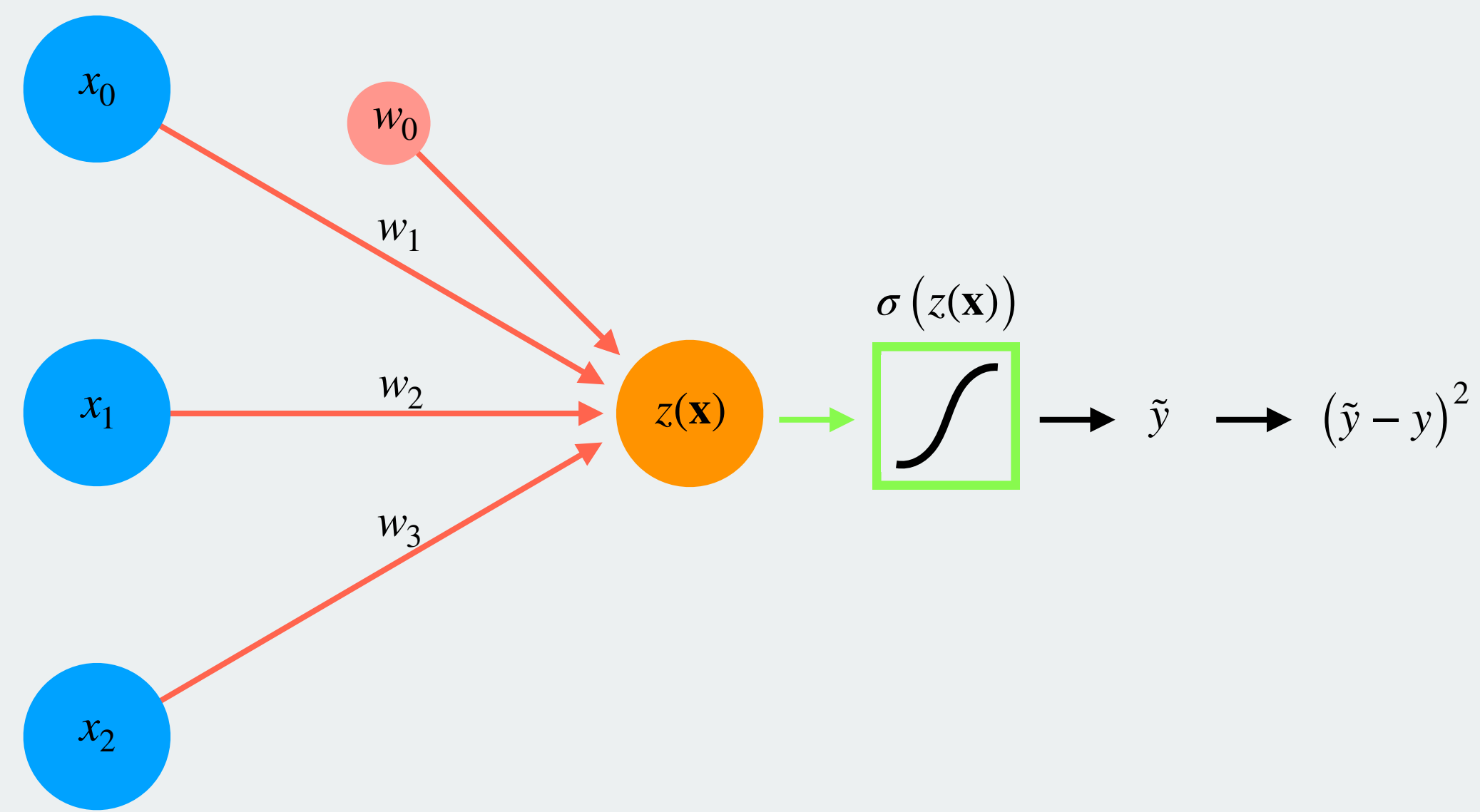
Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

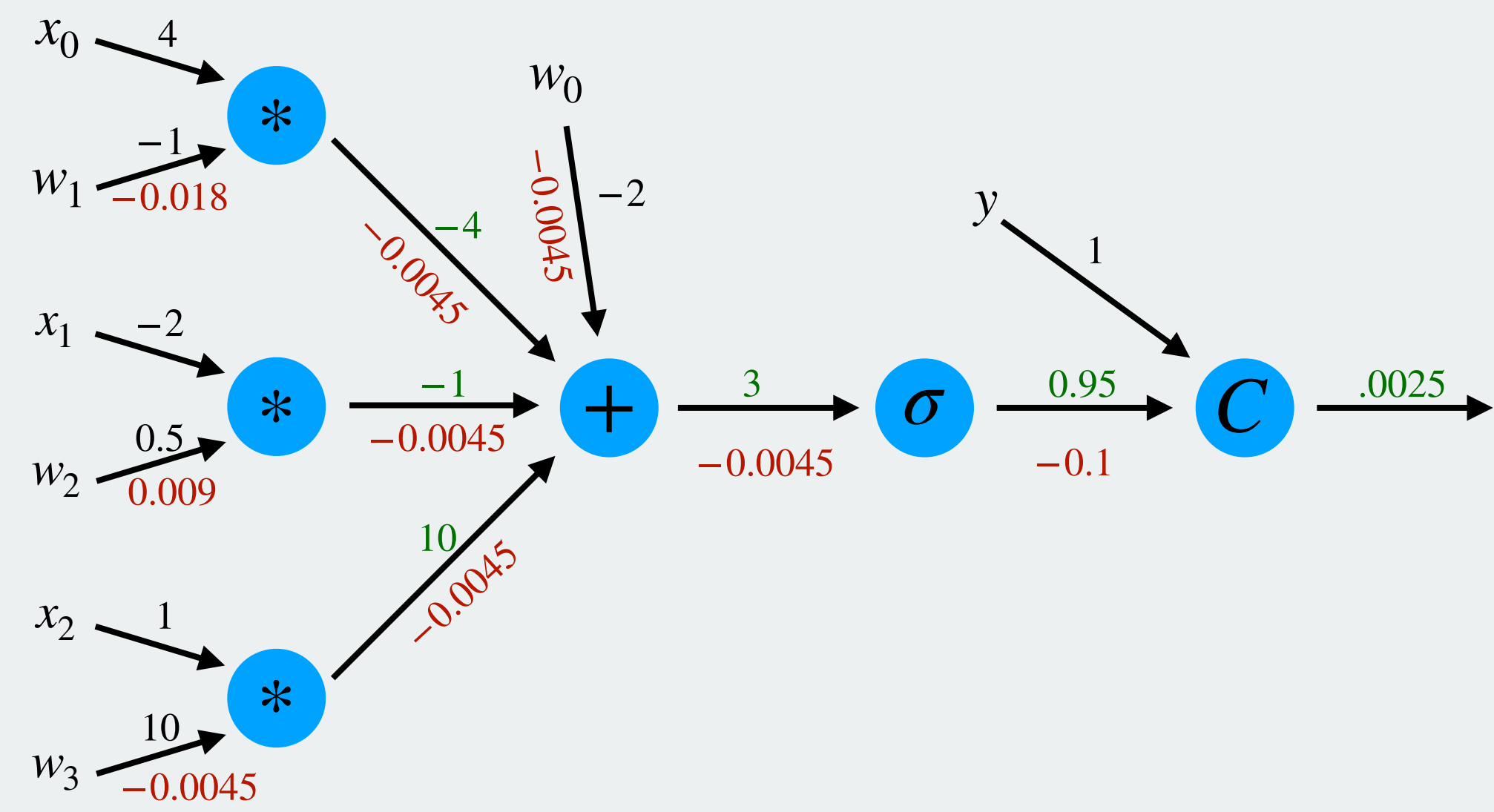
Model derivatives:

Each branch is a function:

$$y = wx$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



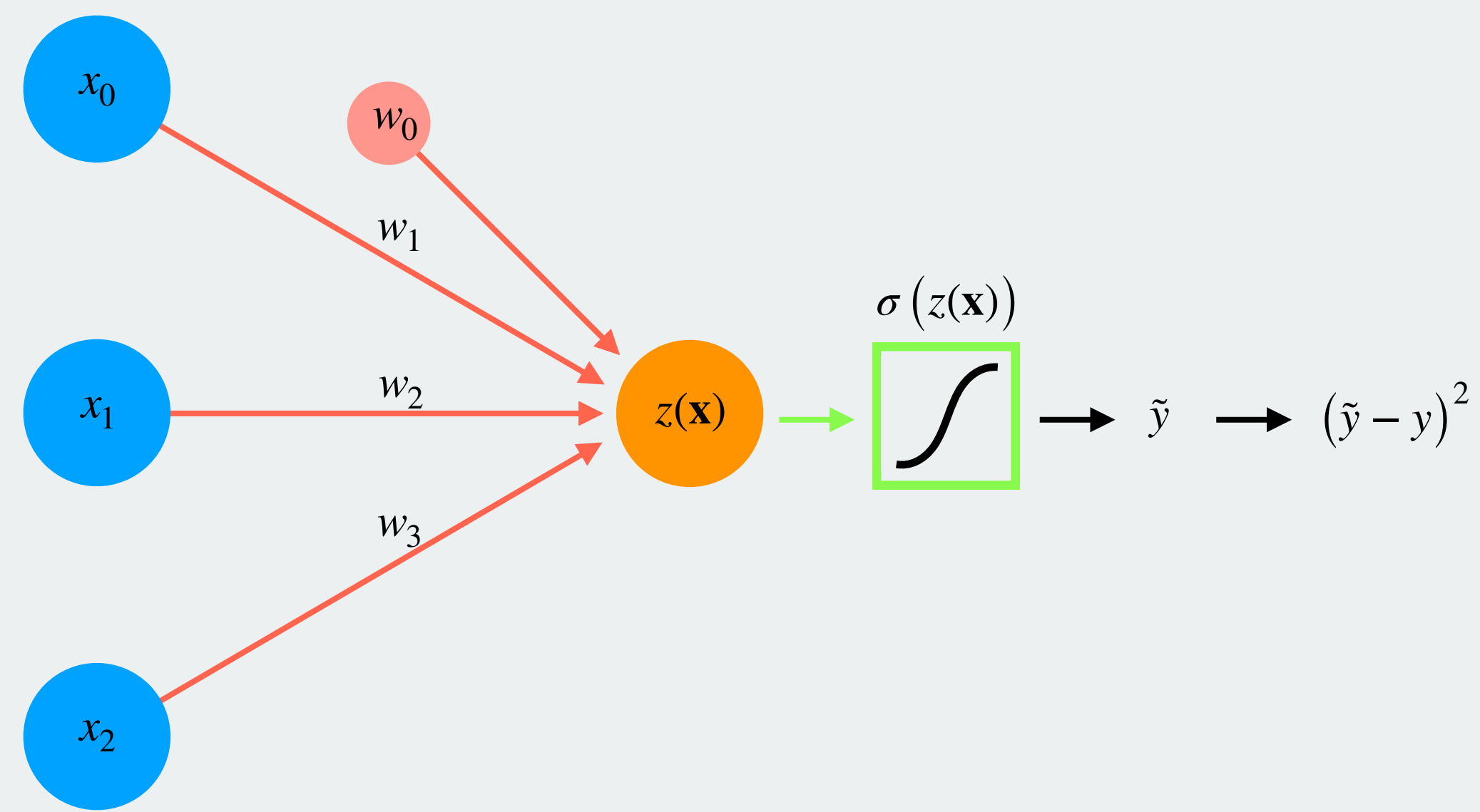
Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

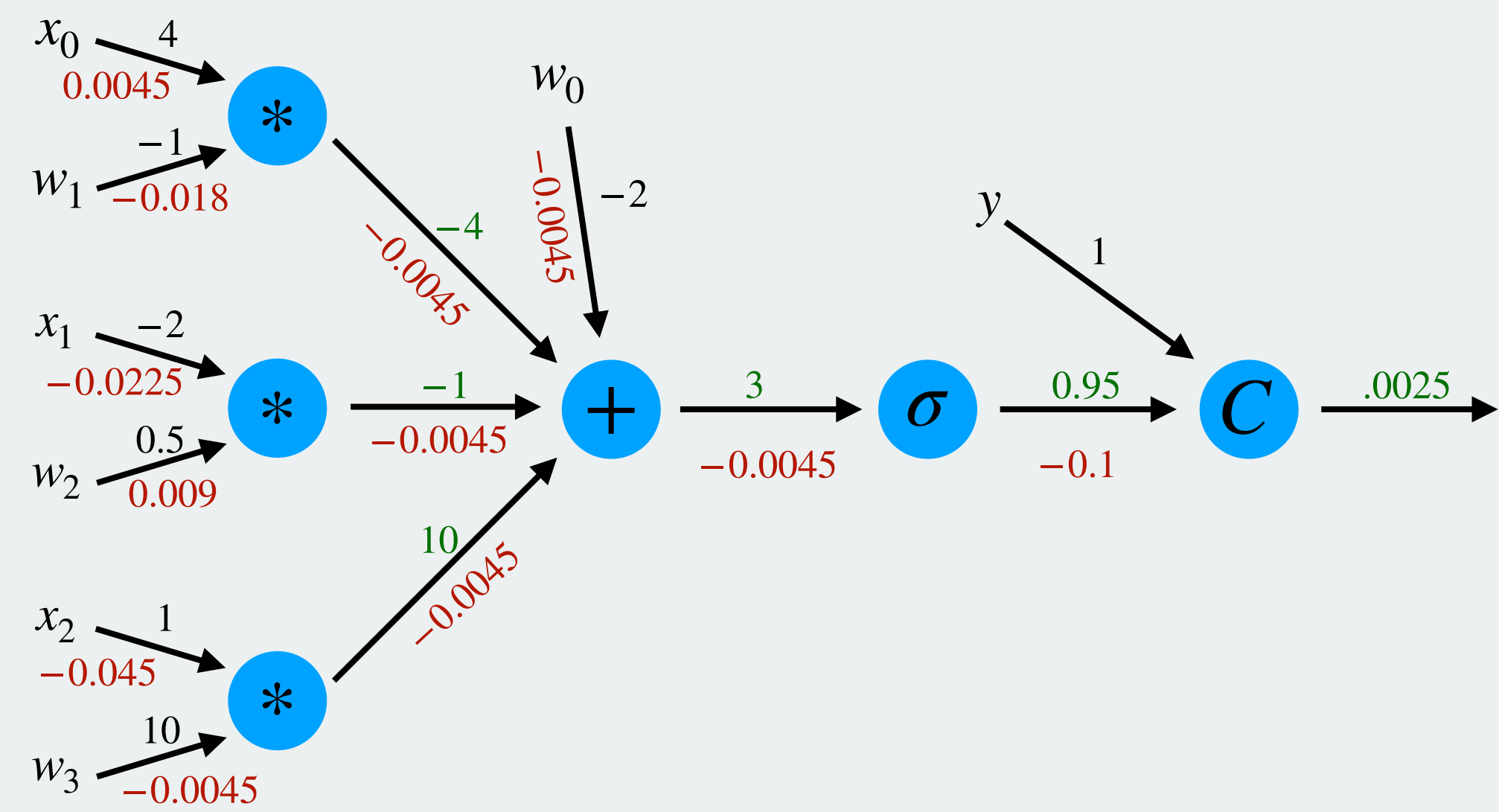
Model derivatives:

Each branch is a function:

$$y = wx$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



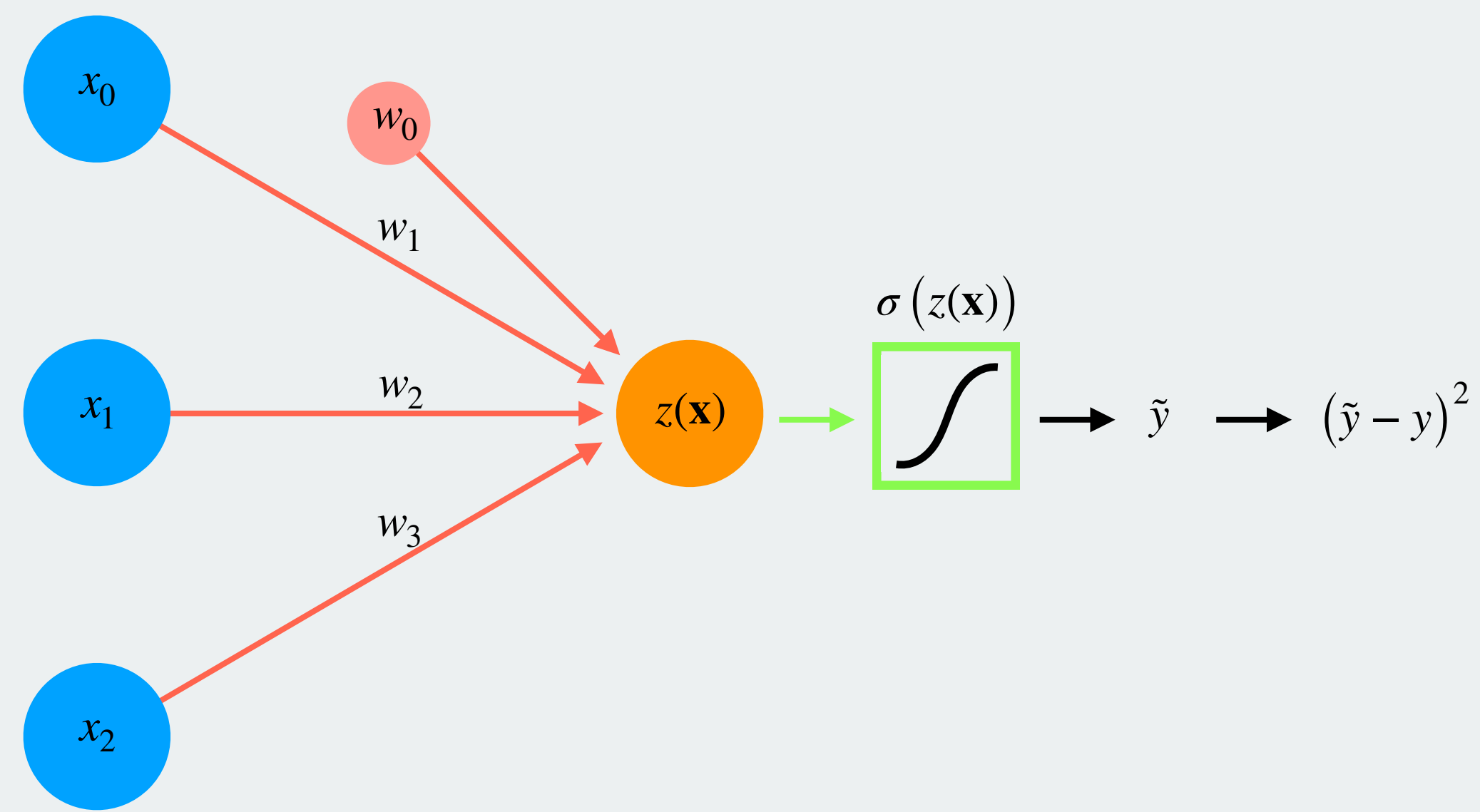
Weights:

$$\mathbf{b} = [-2]$$
$$\mathbf{W} = [-1 \quad 0.5 \quad 10]$$

Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



Model:

$$w_0 + x_0w_1 + x_1w_2 + x_2w_3 = z(\mathbf{x})$$
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$C(\tilde{y}, y) = (\tilde{y} - y)^2$$

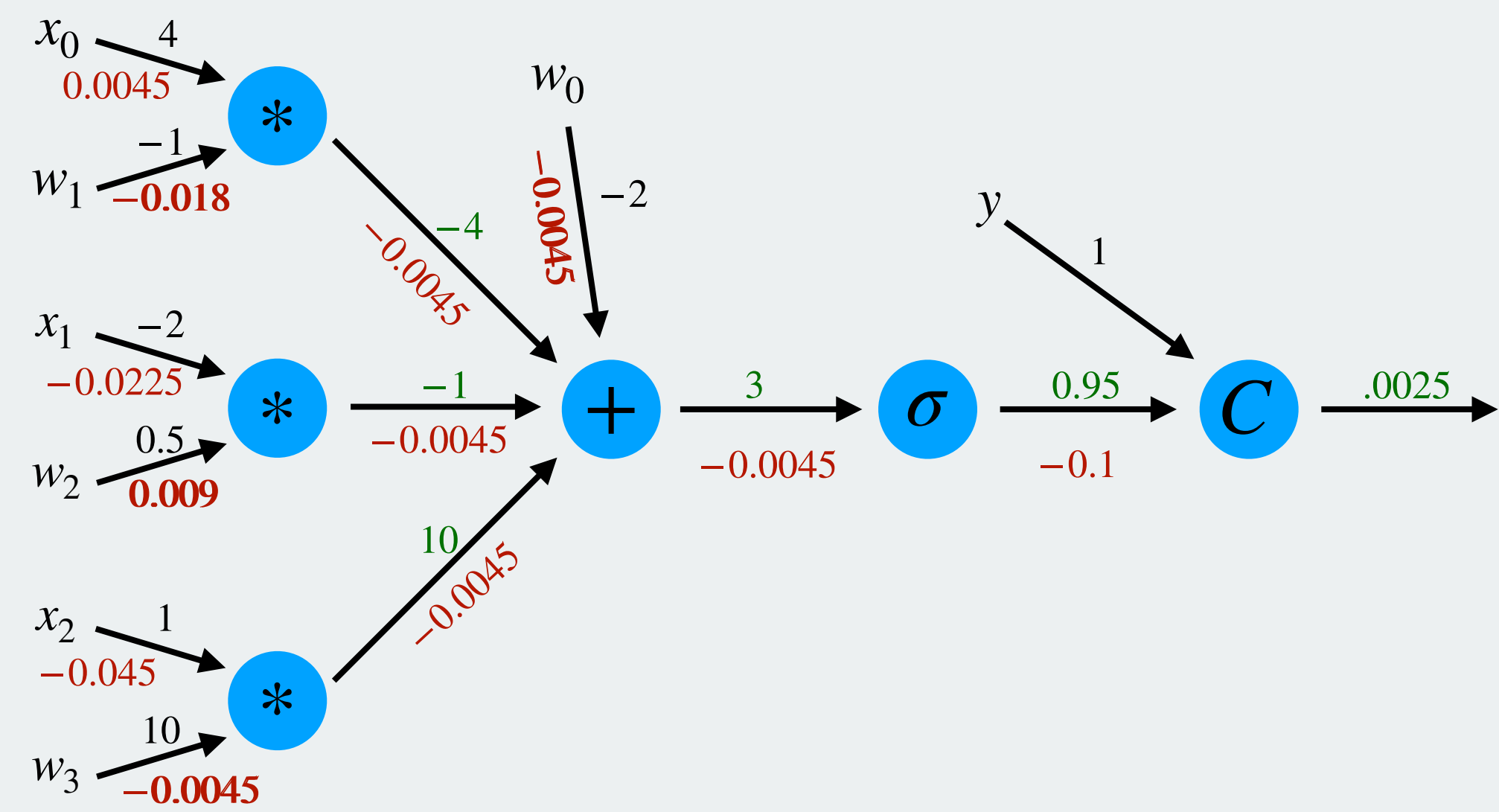
Model derivatives:

Each branch is a function:

$$y = wx$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
$$C'(\tilde{y}, y) = 2(\tilde{y} - y)$$

As a computational graph:

Backward pass



Weights:

$$\mathbf{b} = \begin{bmatrix} -2 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

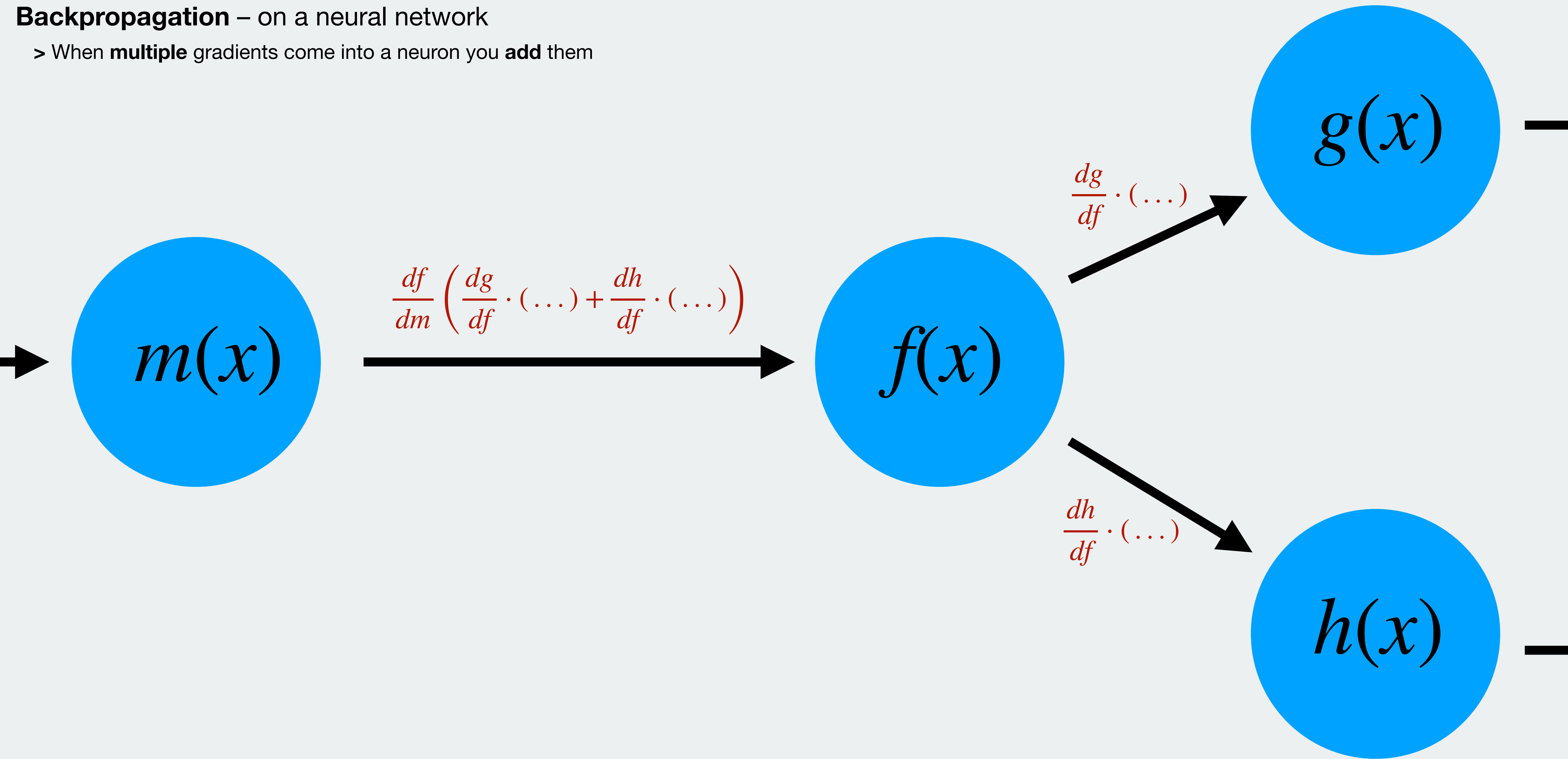
Data:

$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$







$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network

> When **multiple** gradients come into a neuron you **add** them



Backpropagation – in the computer

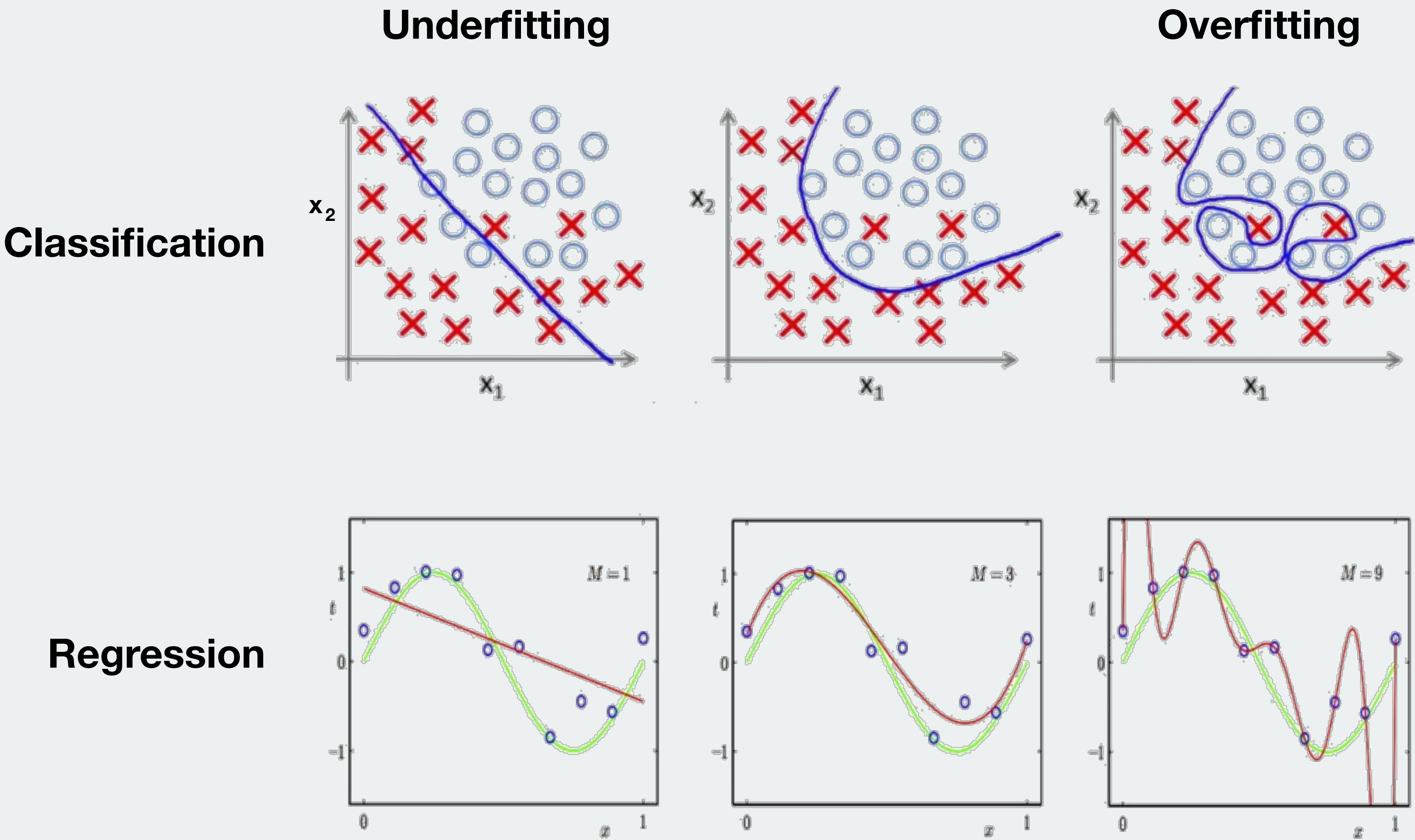
							Average over all training data ...
w_0	-0.08	+0.02	-0.02	+0.11	-0.05	-0.14	... → -0.08
w_1	-0.11	+0.11	+0.07	+0.02	+0.09	+0.05	...
w_2	-0.07	-0.04	-0.01	+0.02	+0.13	-0.15	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$w_{13,001}$	+0.13	+0.08	-0.06	-0.09	-0.02	+0.04	...

<https://www.youtube.com/watch?v=llg3gGewQ5U>

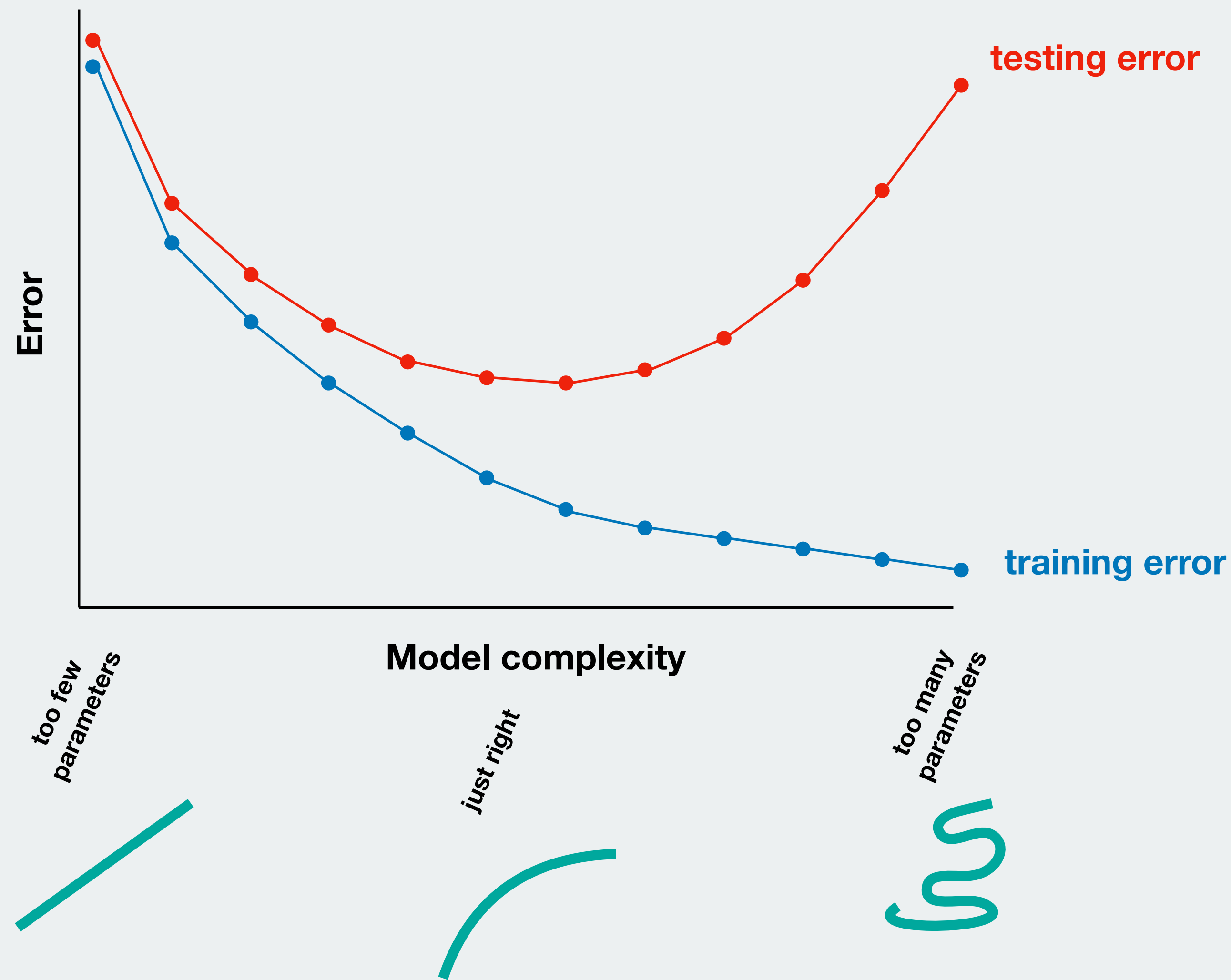
Regularization

Tricks to avoid overfitting

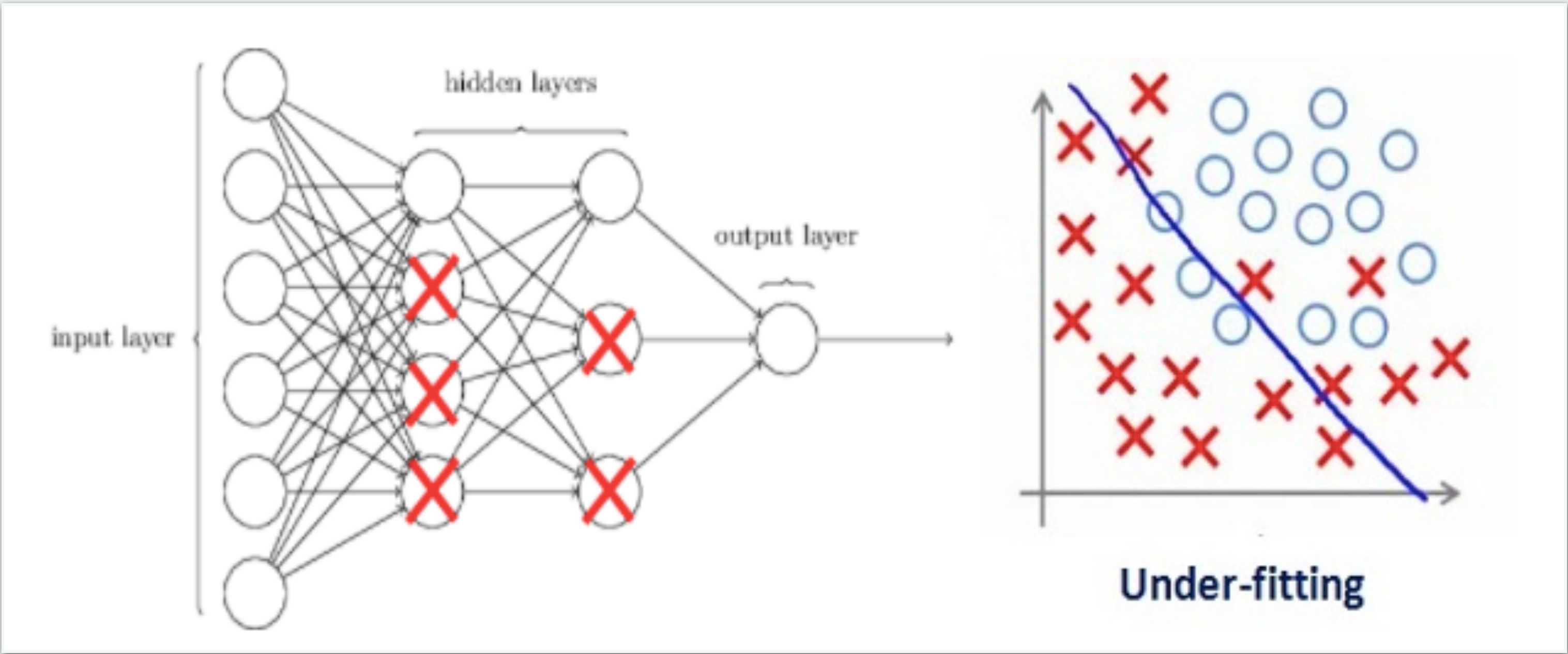
Regularization – underfitting and overfitting



Regularization – underfitting and overfitting



Regularization – *how* does regularization reduce overfitting?



<https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/>

Regularization – Different techniques

L-norm regularization: “Introduce a cost for large weights”

$$C = Loss + Regularization\ term$$

Regularization – Different techniques

L-norm regularization: “Introduce a cost for large weights”

$$C = Loss + Regularization\ term$$

$$\mathbf{L1:} \quad C = Loss + \lambda \sum_{l=1}^L \|\mathbf{W}_l\| \qquad \mathbf{L2:} \quad C = Loss + \lambda \sum_{l=1}^L \|\mathbf{W}_l^2\|$$

Regularization – Different techniques

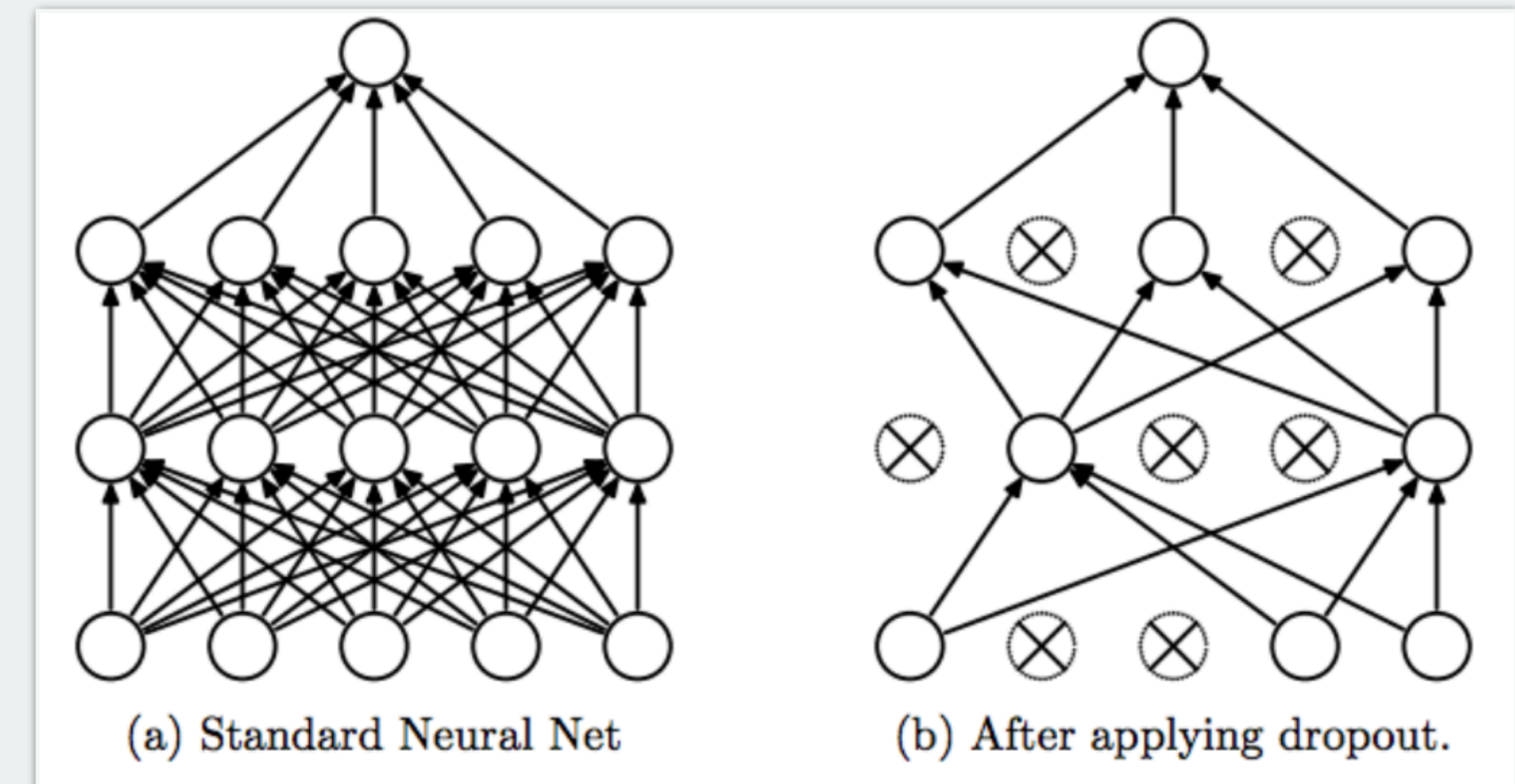
L-norm regularization: “Introduce a cost for large weights”

$$C = Loss + Regularization\ term$$

$$\mathbf{L1:} \quad C = Loss + \lambda \sum_{l=1}^L \|\mathbf{W}_l\| \qquad \mathbf{L2:} \quad C = Loss + \lambda \sum_{l=1}^L \|\mathbf{W}_l^2\|$$

Dropout:

“In each SGD step, randomly ignore a fraction p of neurons”



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

- Can select p in wide range. Typical is 0.2 – 0.8, dependent on size of ANN
- Can apply only in specific layers. It is typical to only do dropout in a designated “dropout layer” somewhere close to output.

Regularization – Different techniques

L-norm regularization: “Introduce a cost for large weights”

$$C = Loss + Regularization\ term$$

L1:

$$C = Loss + \lambda \sum_{l=1}^L \|W_l\|$$

L2:

$$C = Loss + \lambda \sum_{l=1}^L \|W_l^2\|$$

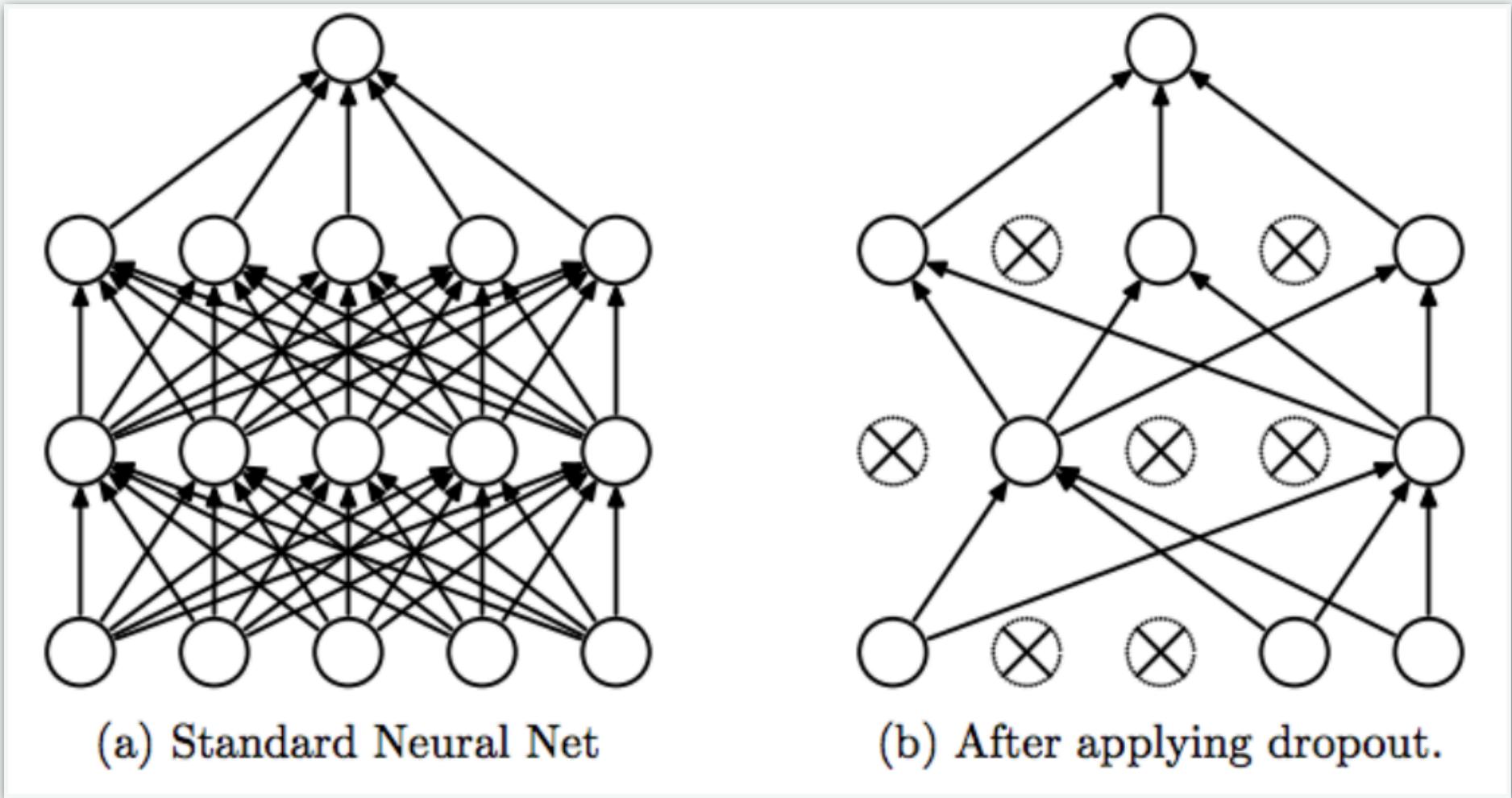
Data augmentation

“Shear, shift, scale and/or rotate input data”



Dropout:

“In each SGD step, randomly ignore a fraction p of neurons”



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

- Can select p in wide range. Typical is 0.2 – 0.8, dependent on size of ANN
- Can apply only in specific layers. It is typical to only do dropout in a designated “dropout layer” somewhere close to output.

Regularization – Different techniques

L-norm regularization: “Introduce a cost for large weights”

$$C = Loss + Regularization\ term$$

L1: $C = Loss + \lambda \sum_{l=1}^L \|W_l\|$ **L2:** $C = Loss + \lambda \sum_{l=1}^L \|W_l^2\|$

Data augmentation

“Shear, shift, scale and/or rotate input data”



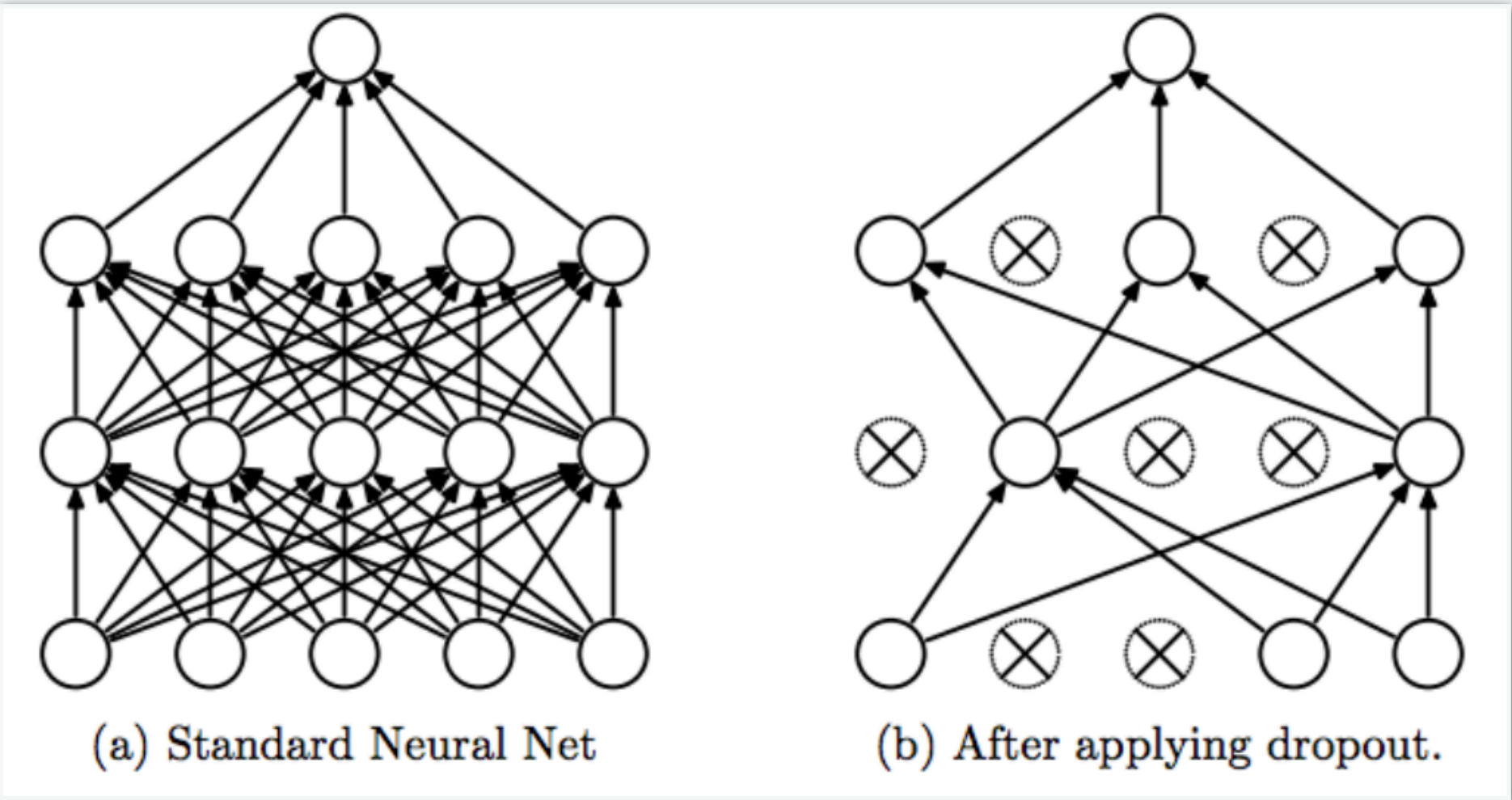
Early stopping

“Stop training when performance on validation dataset starts worsening”



Dropout:

“In each SGD step, randomly ignore a fraction p of neurons”



Srivastava, Nitish, et al. “Dropout: a simple way to prevent neural networks from overfitting”, JMLR 2014

- Can select p in wide range. Typical is 0.2 – 0.8, dependent on size of ANN
- Can apply only in specific layers. It is typical to only do dropout in a designated “dropout layer” somewhere close to output.

Very useful! Zero out random nodes.
Intuitively works like ensemble models - very powerful

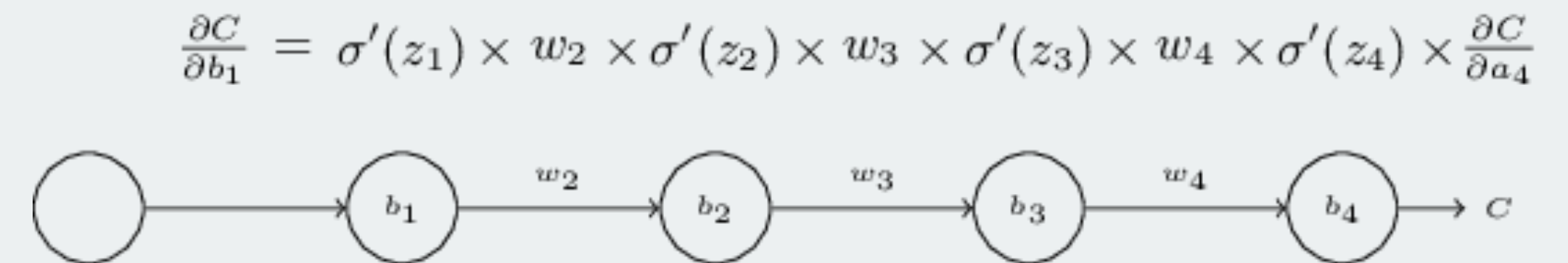
A quick word on:

The Vanishing Gradient Problem

Vanishing gradients – A problem in *deep* neural nets

Problem:

- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning

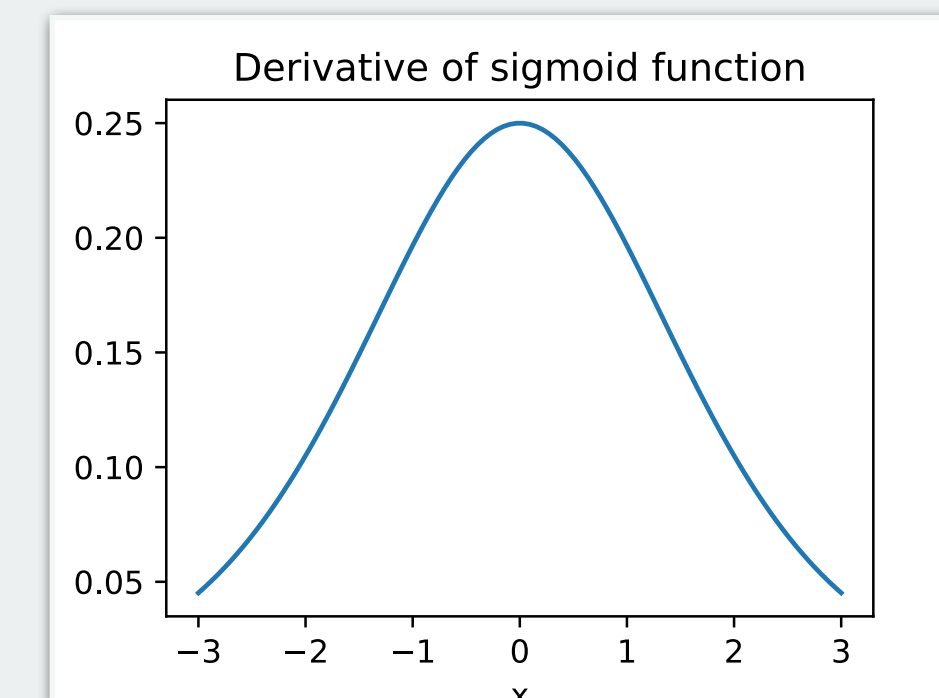


Vanishing gradients – A problem in *deep* neural nets

Problem:

- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Vanishing gradients – A problem in *deep* neural nets

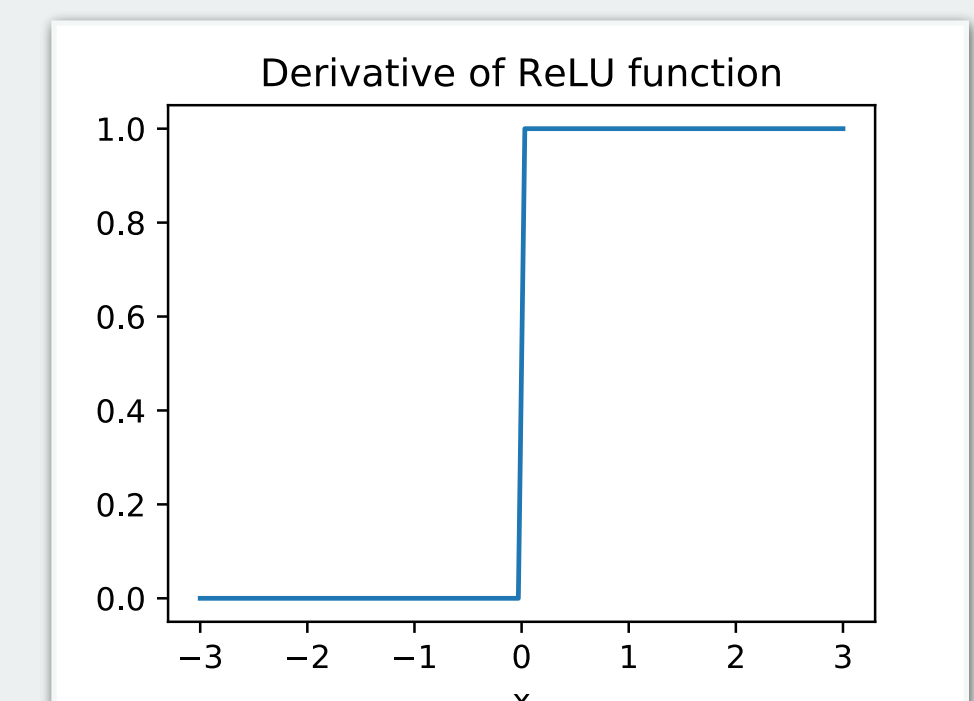
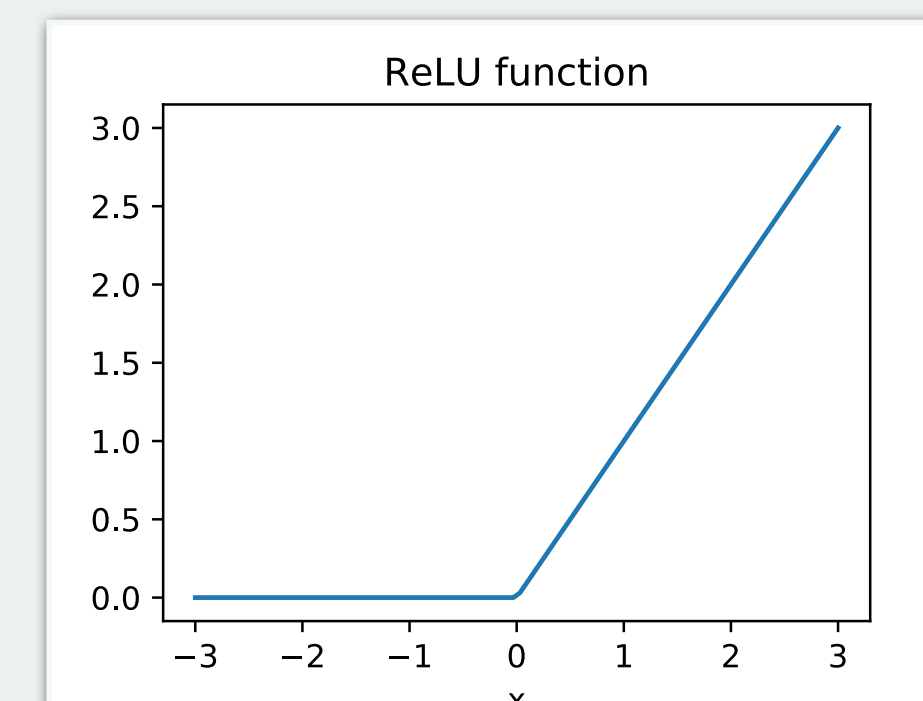
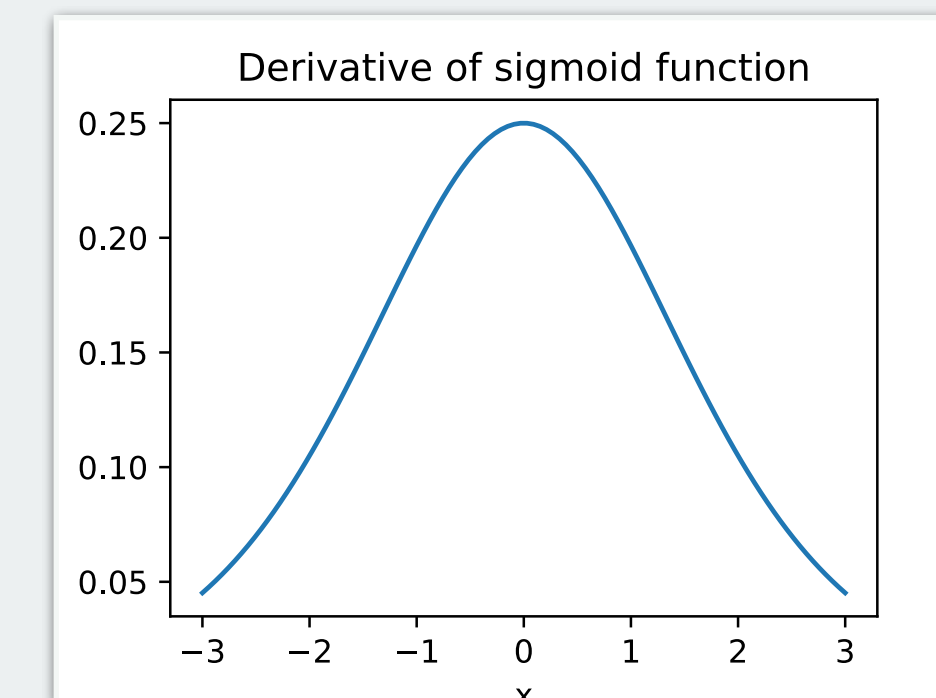
Problem:

- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning

Solution:

- Use an activation function without small gradient for high values
- Candidate activation function: ReLU

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Vanishing gradients – A problem in *deep* neural nets

Problem:

- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning

Solution:

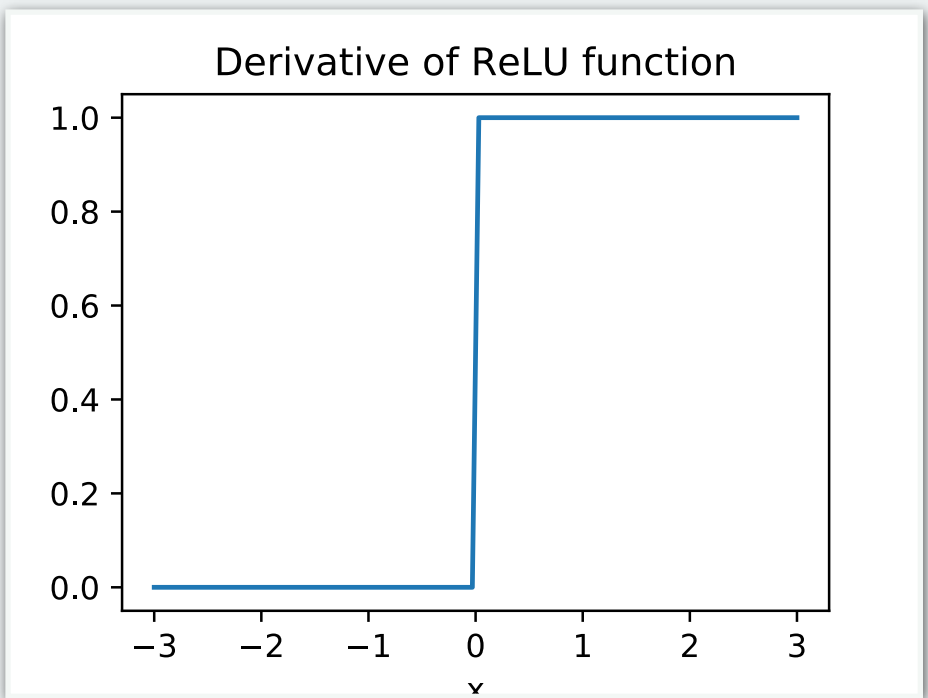
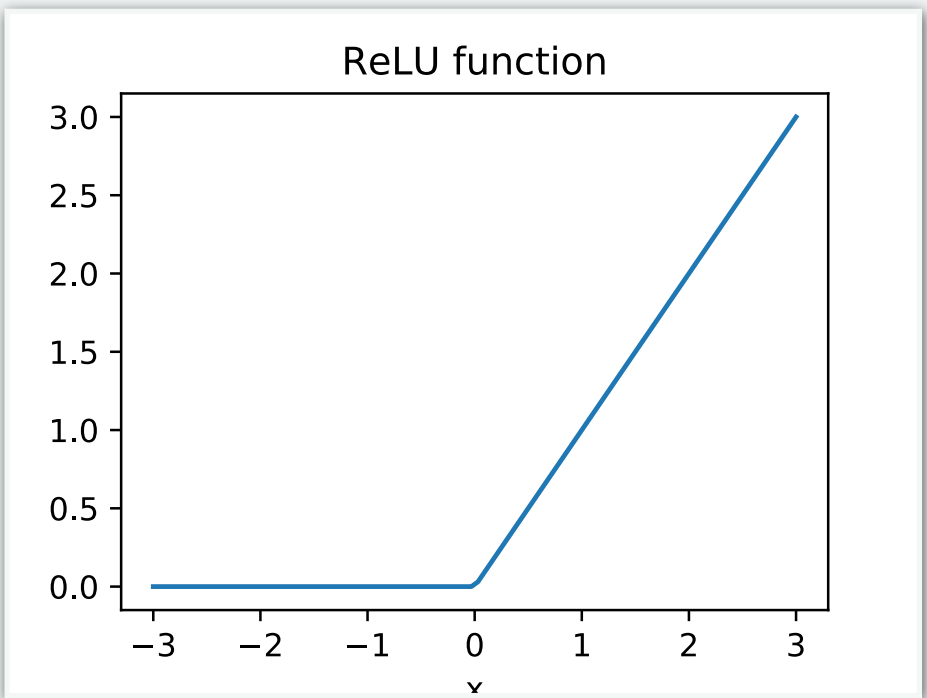
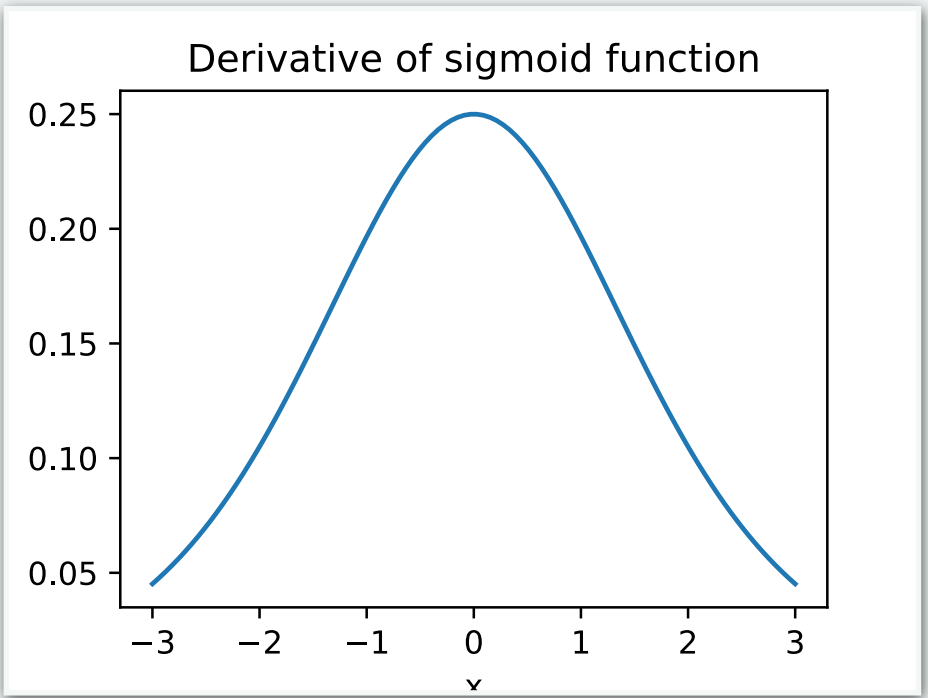
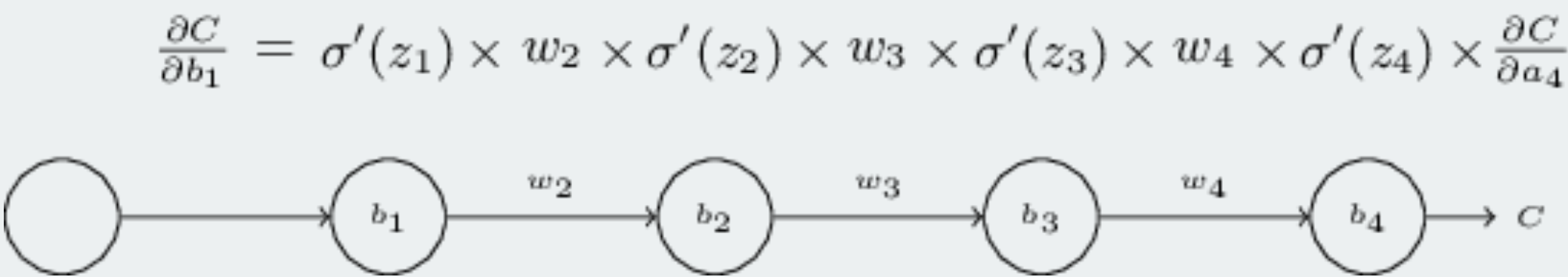
- Use an activation function without small gradient for high values
- Candidate activation function: ReLU

Problems with ReLU:

- Exploding gradients!

Solution:

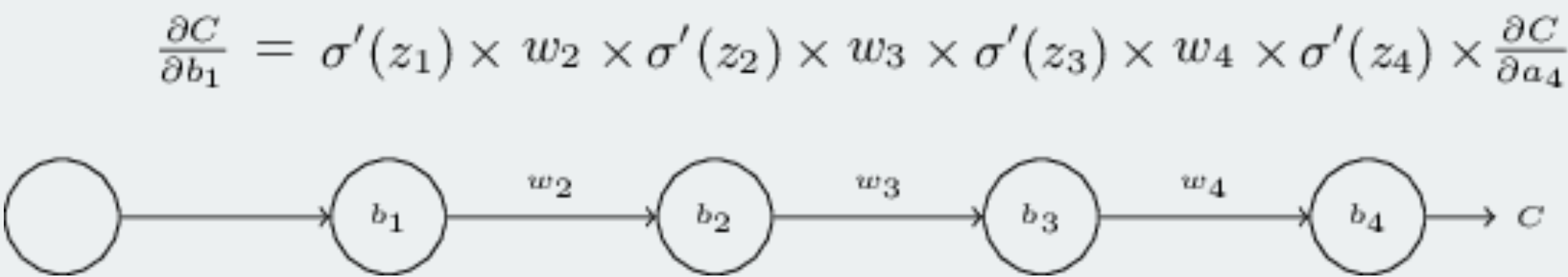
- Batch normalization, gradient clipping, weight regularization



Vanishing gradients – A problem in *deep* neural nets

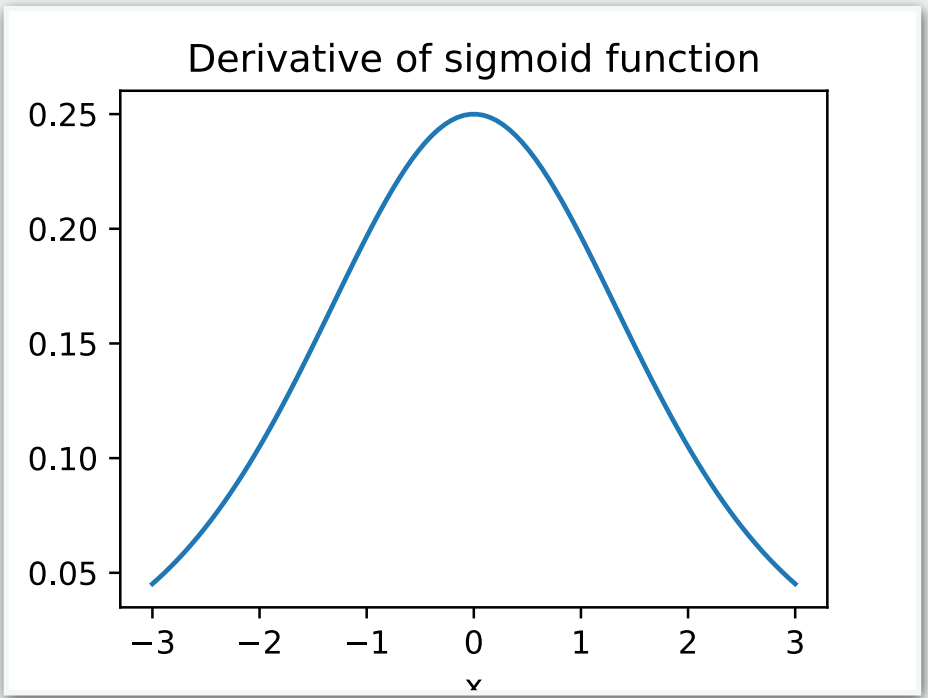
Problem:

- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning



Solution:

- Use an activation function without small gradient for high values
- Candidate activation function: ReLU



Problems with ReLU:

- Exploding gradients!

Solution:

- Batch normalization, gradient clipping, weight regularization

