Topics in Social Data Science

Week 3

Artificial Neural Networks 2

Backpropagation, regularization, vanishing gradients

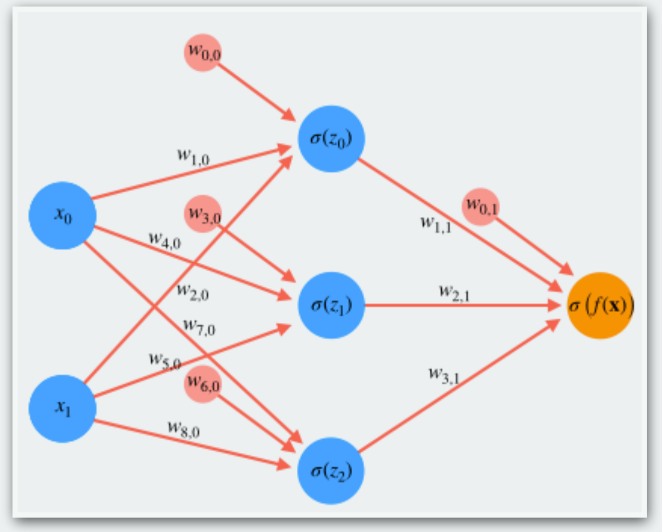
Overview of today + tomorrow

- 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…

Find the gradients with **Backpropagation** … this week

(1) The model



Input → Neural network function → **Output**

Datapoints, **X**

Parameters: Weights, **W**

Prediction, $\tilde{\mathbf{y}}$

(2) Its performance

Input → Cost function → **Output**

Weights, **W**

Many training examples

One number, **the cost**

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

(3) The cost function gradient in **W**

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

(4) Updating **W**

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

(5) Repeat 3 and 4

r is usually called the *learning rate*

Backpropagation

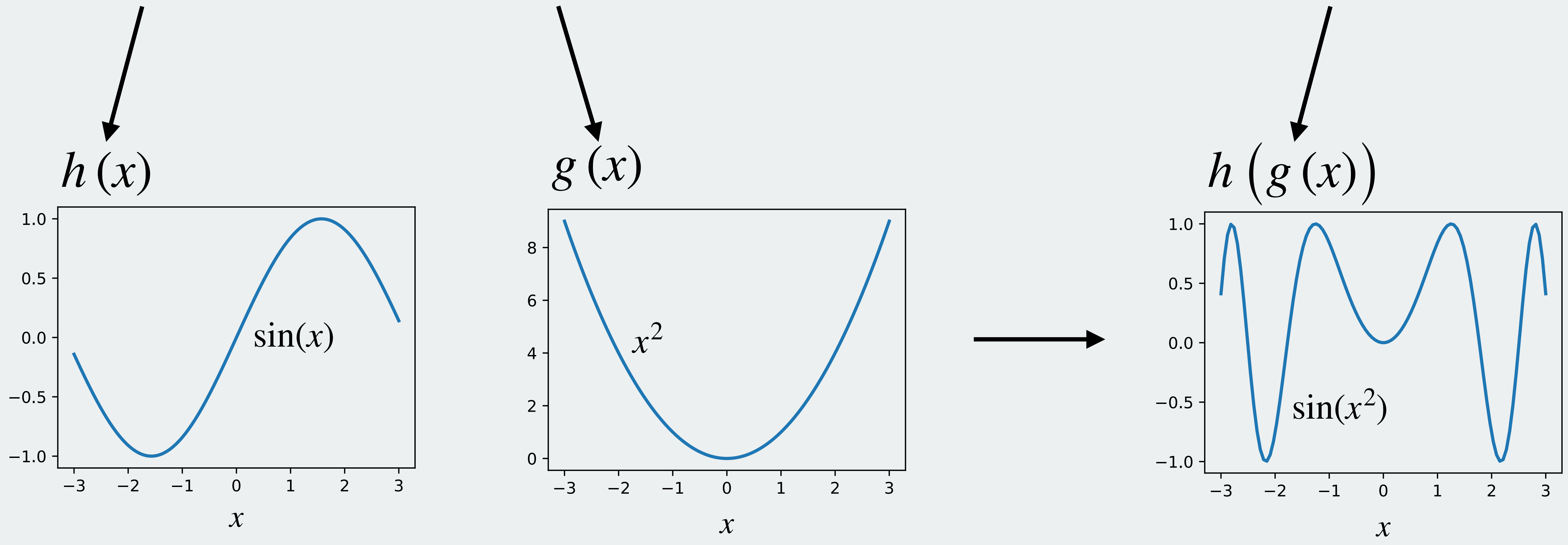
THE algorithm for computing the analytical **gradient** of the cost function

Chain rule – Taking derivatives of *composite functions*

one function

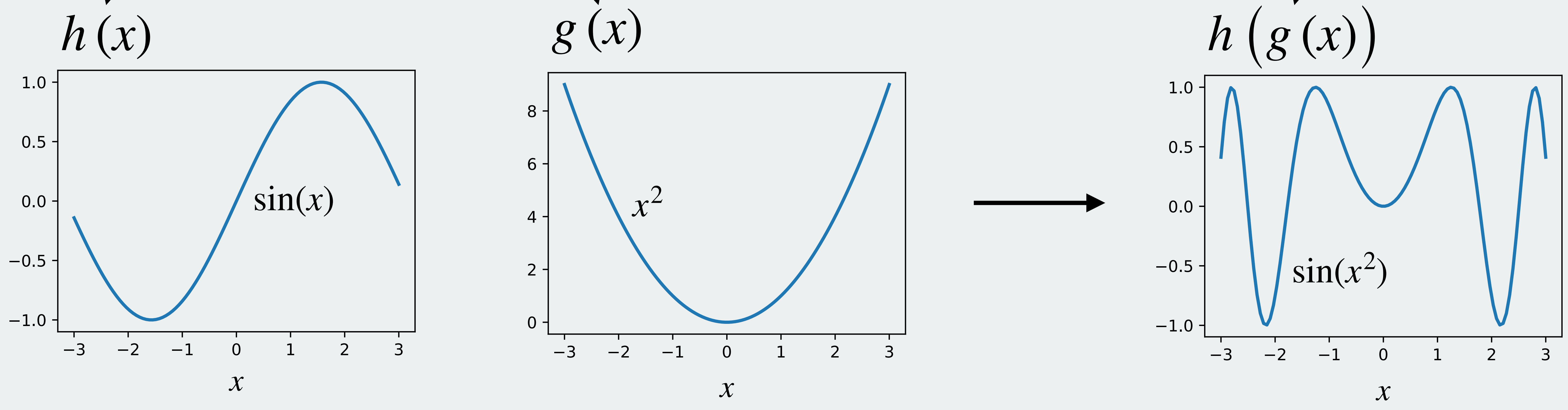
another function

a *composite* of the two



Chain rule – Taking derivatives of *composite functions*

one function — another function — a *composite* of the two



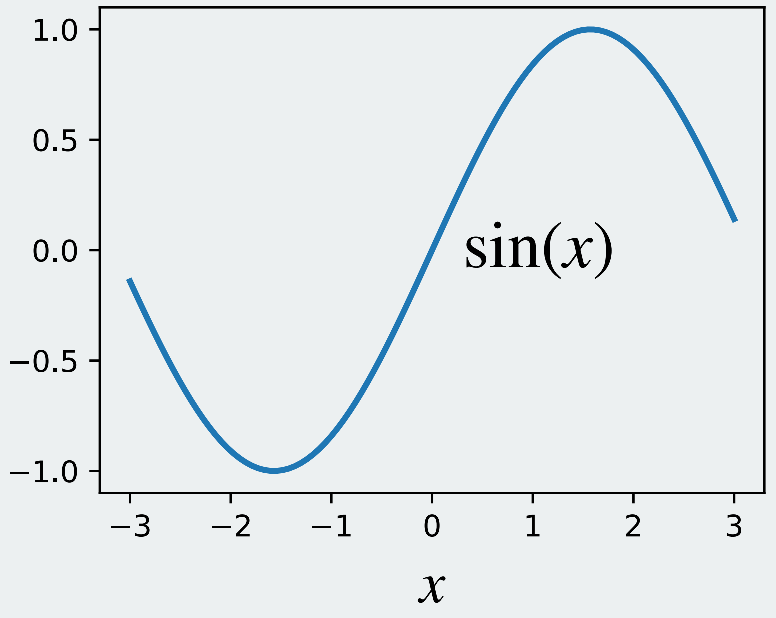
$$h\left(g\left(x\right)\right)$$

$$x \rightarrow g \rightarrow h$$

Chain rule – Taking derivatives of *composite functions*

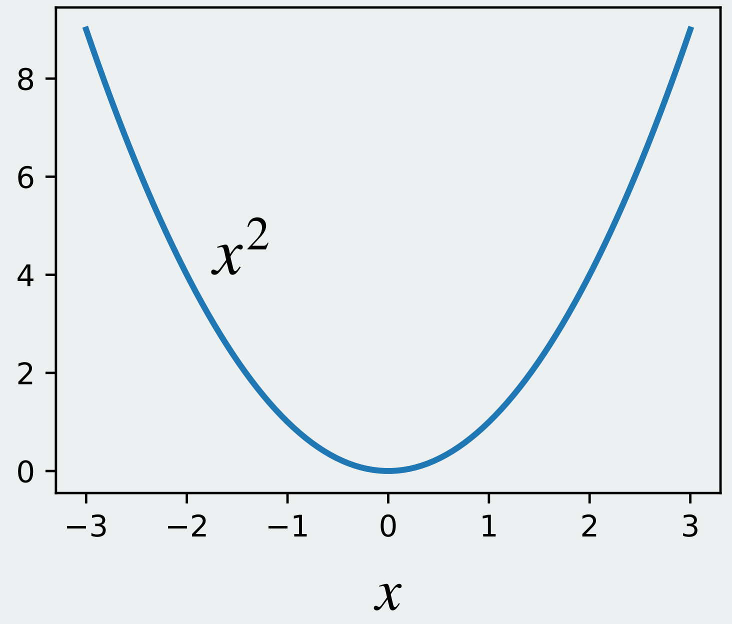
one function

$h(x)$



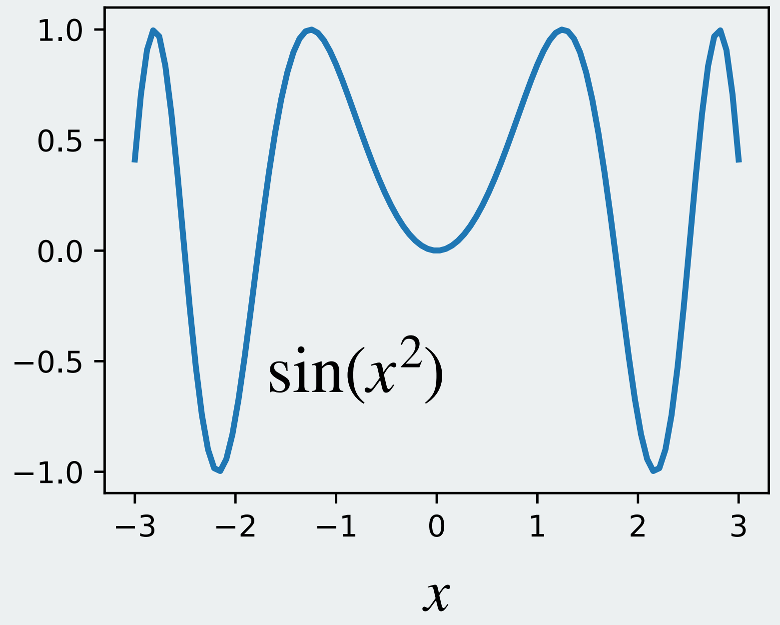
another function

$g(x)$



a *composite* of the two

$h\left(g(x)\right)$



$h\left(g(x)\right)$

$$x \to g \to h$$

Chain rule says:

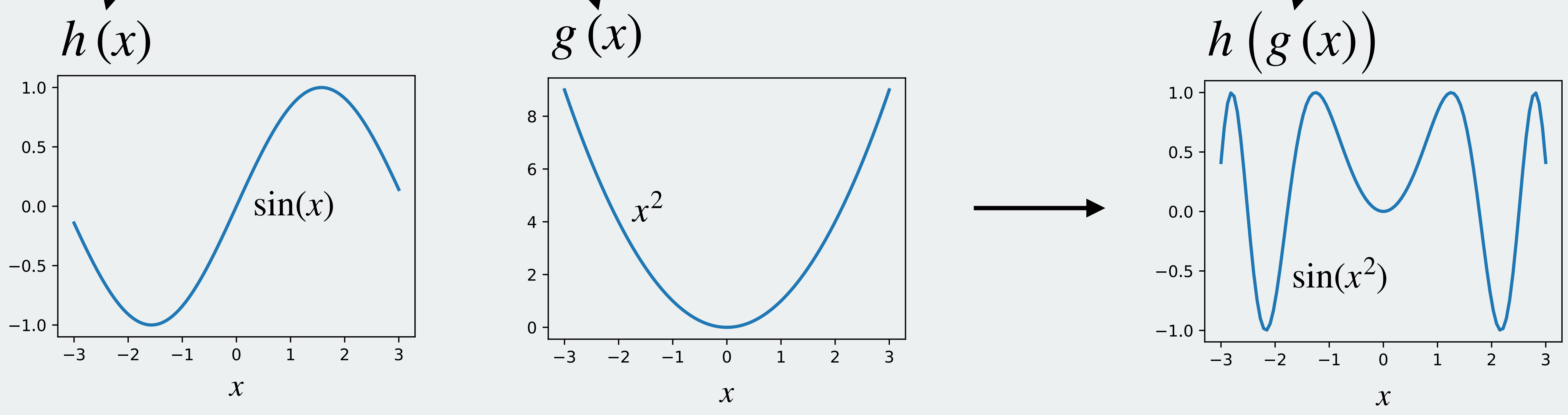
$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

Chain rule – Taking derivatives of *composite functions*

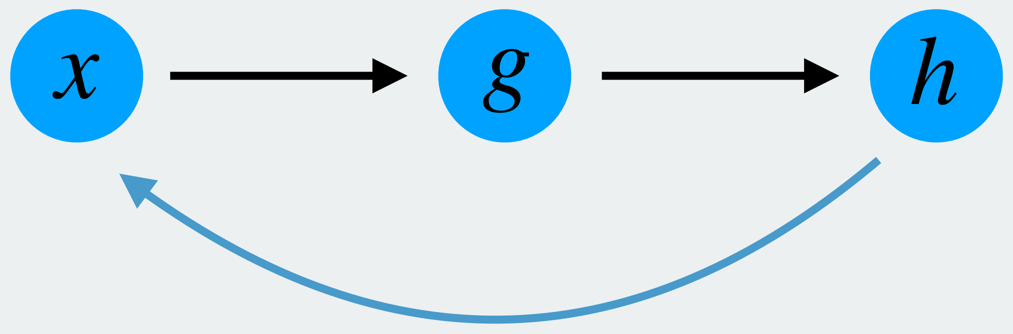
one function

another function

a *composite* of the two



$$h\left(g\left(x\right)\right)$$

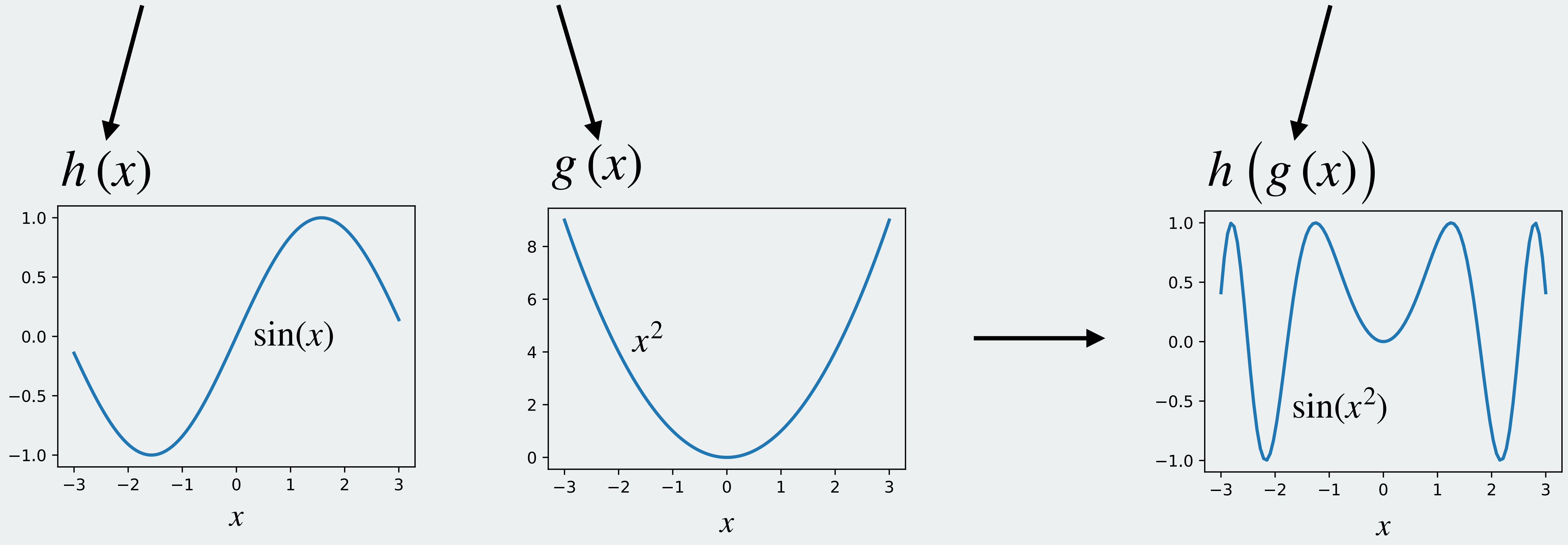


Chain rule says:

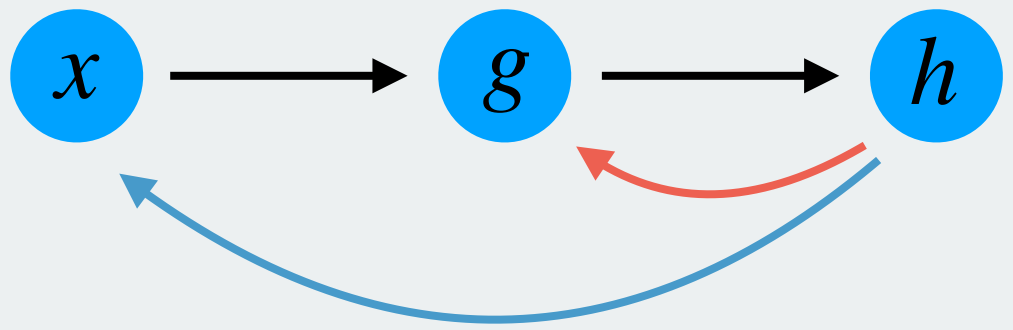
$$\boxed{\frac{dh}{dx}} = \frac{dh}{dg}\frac{dg}{dx}$$

Chain rule – Taking derivatives of *composite functions*

one function   another function   a *composite* of the two



$h\left(g(x)\right)$

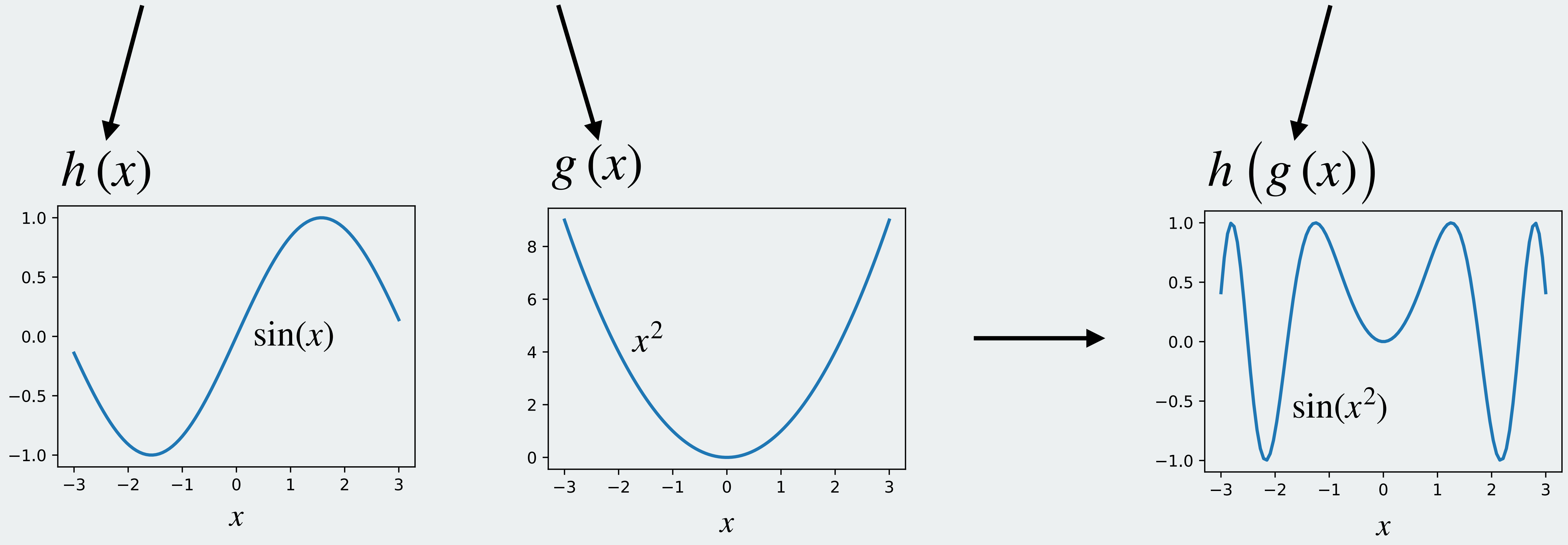


Chain rule says:

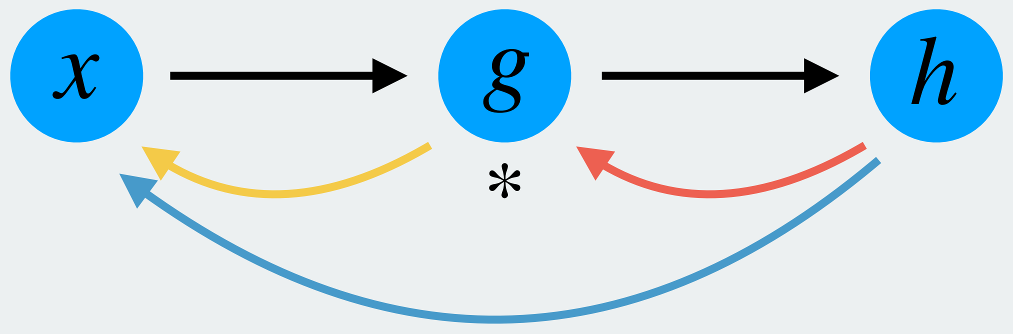
$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

Chain rule – Taking derivatives of *composite functions*

one function | another function | a *composite* of the two



$h\left(g(x)\right)$



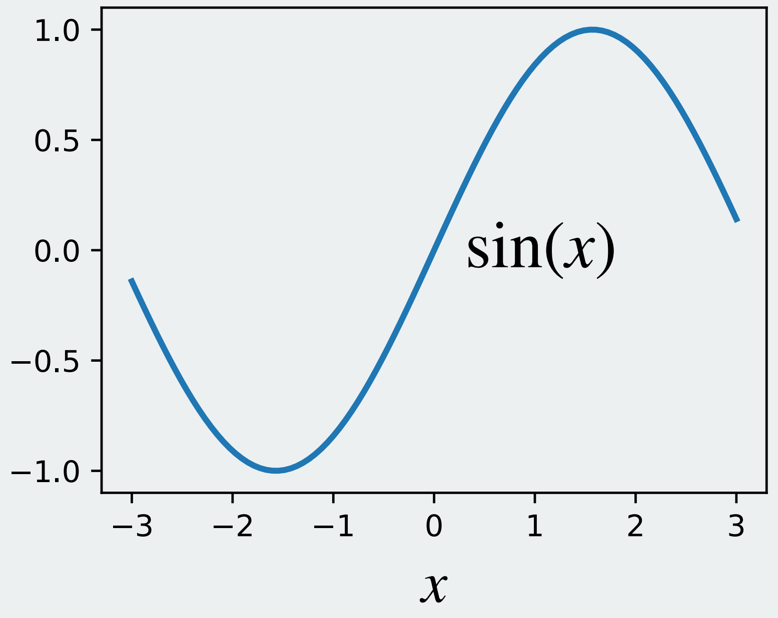
Chain rule says:

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

Chain rule – Taking derivatives of *composite functions*

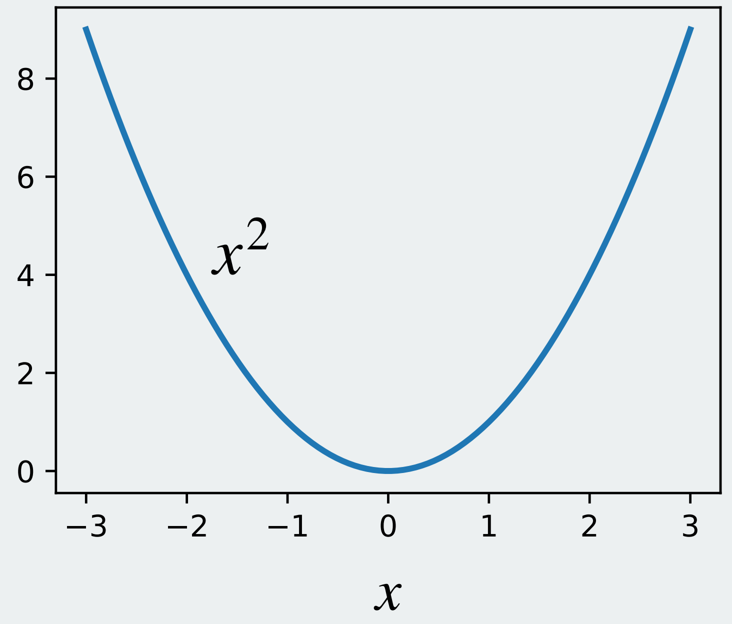
one function

$h(x)$



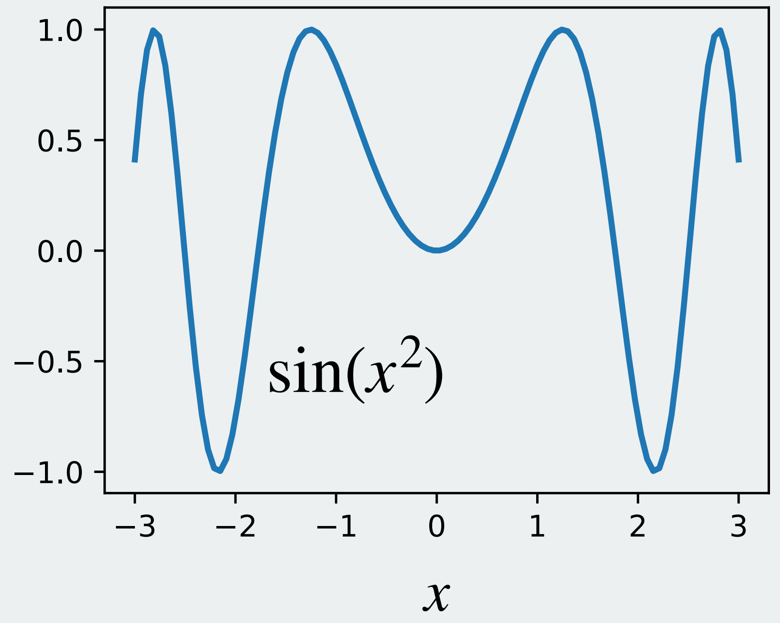
another function

$g(x)$



a *composite* of the two

$h\left(g(x)\right)$



$$f\left(h\left(g(x)\right)\right)$$

$$\begin{tikzcd} x \arrow[r] & g \arrow[r] & h \arrow[r] & f \arrow[lll, bend left] \end{tikzcd}$$

Chain rule says:

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

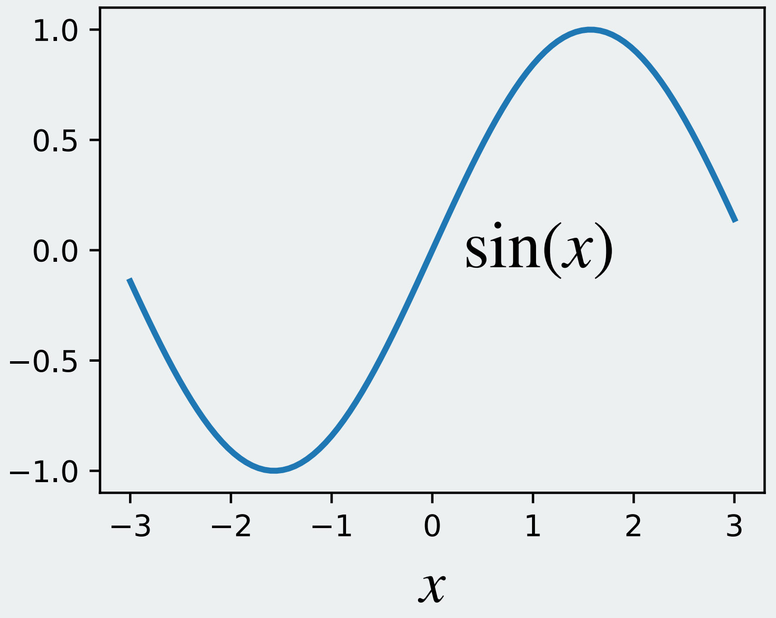
Chain rule – Taking derivatives of *composite functions*

one function

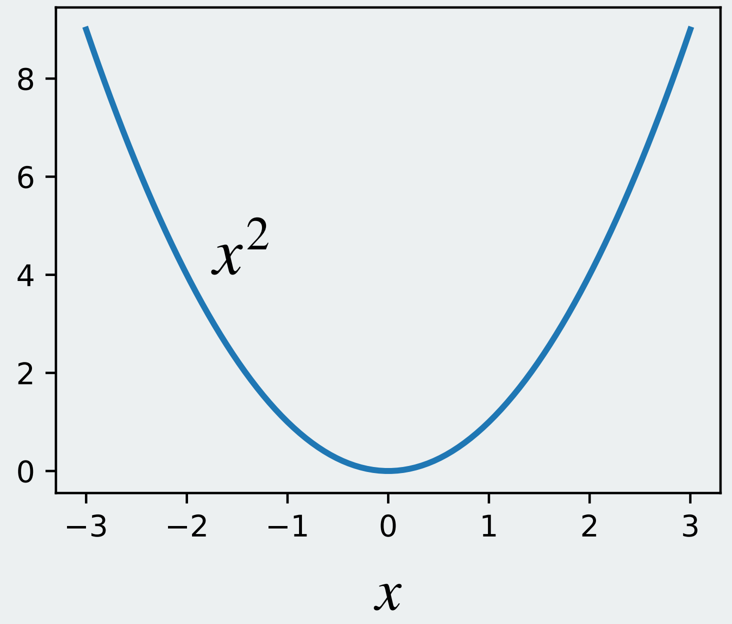
another function

a *composite* of the two

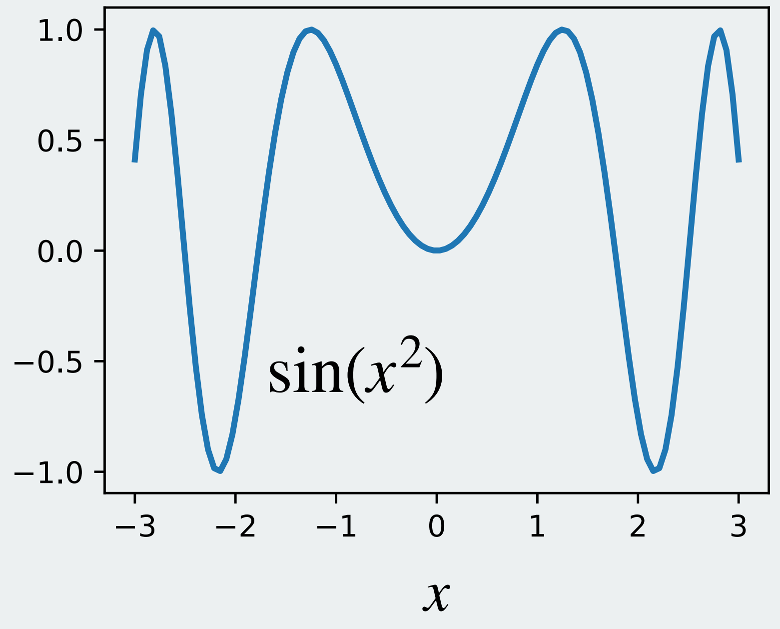
$h(x)$



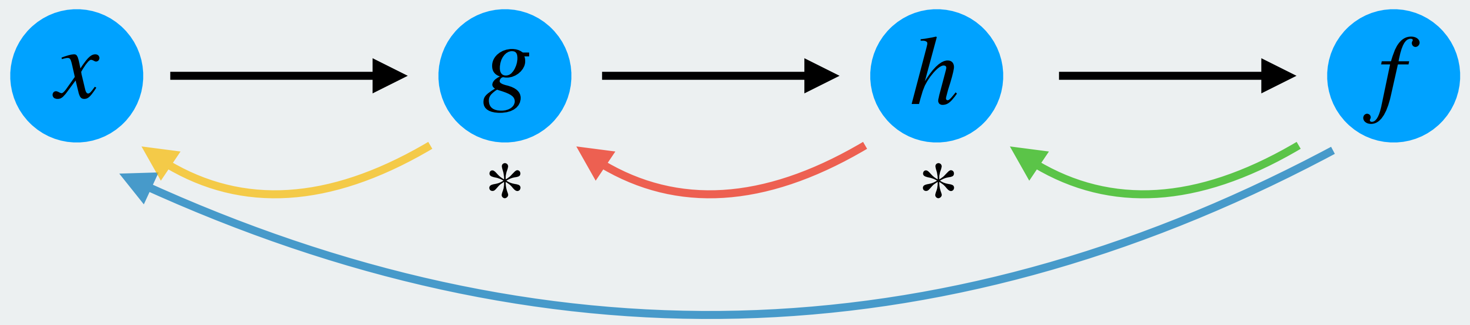
$g(x)$



$h\left(g(x)\right)$



$$f\left(h\left(g(x)\right)\right)$$



Chain rule says:

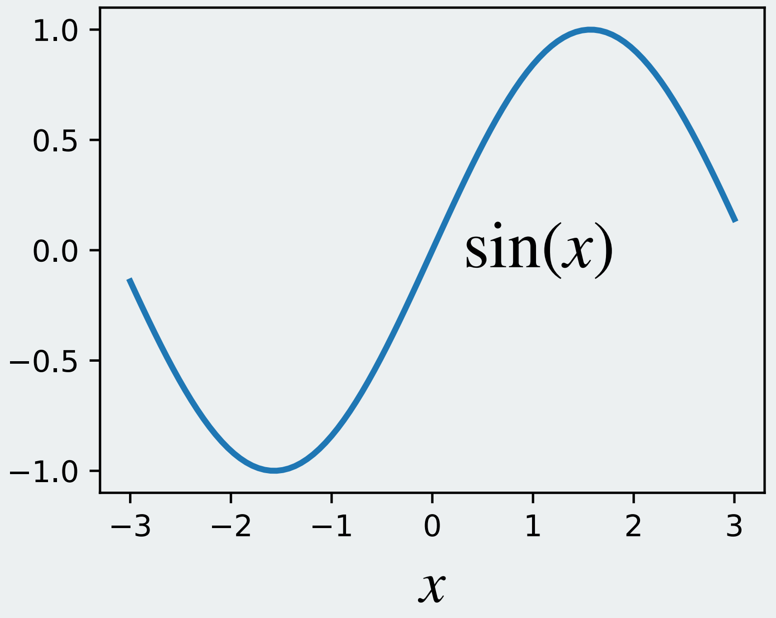
$$\frac{df}{dx} = \boxed{\frac{df}{dh}}\frac{dh}{dg}\frac{dg}{dx}$$

3Blue1Brown Chain Rule video

Chain rule – Taking derivatives of *composite functions*

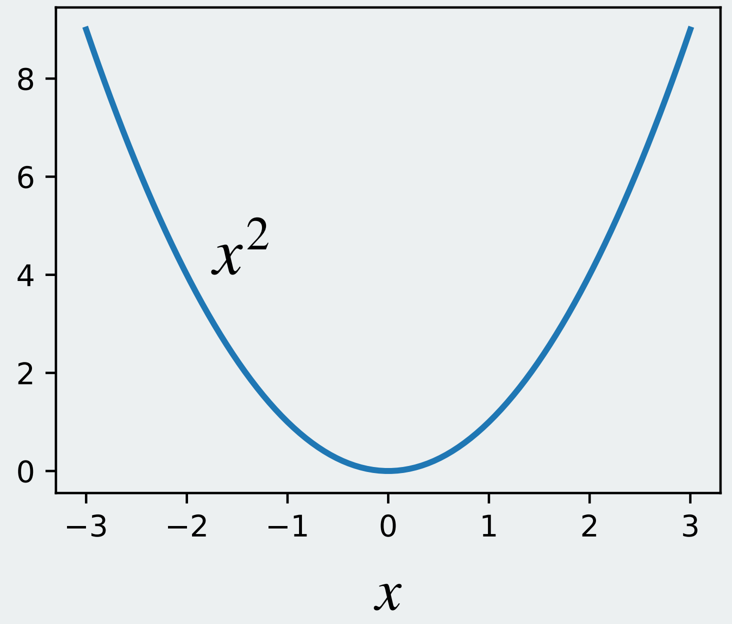
one function

$h(x)$



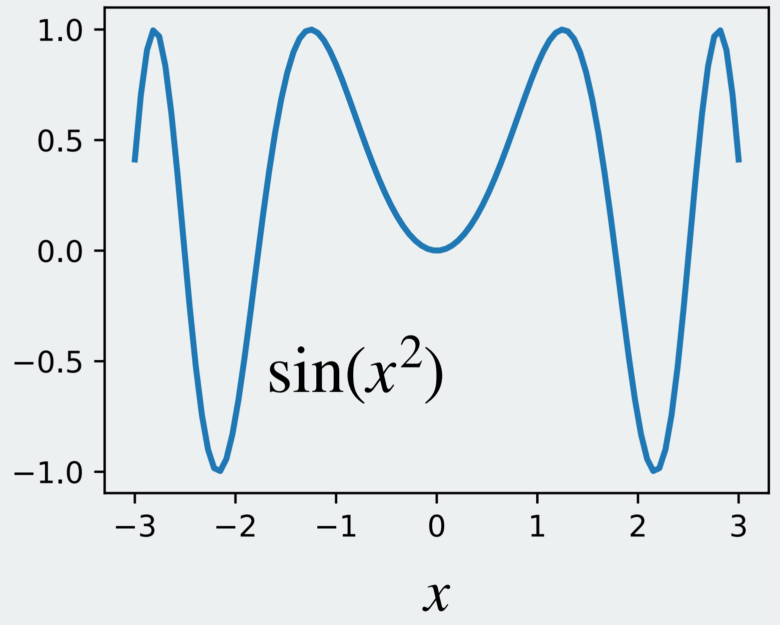
another function

$g(x)$

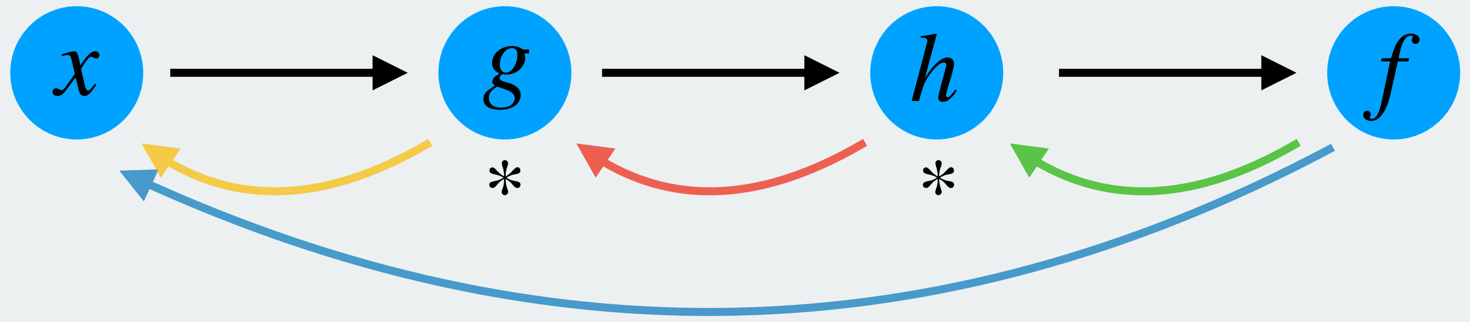


a *composite* of the two

$h\left(g(x)\right)$

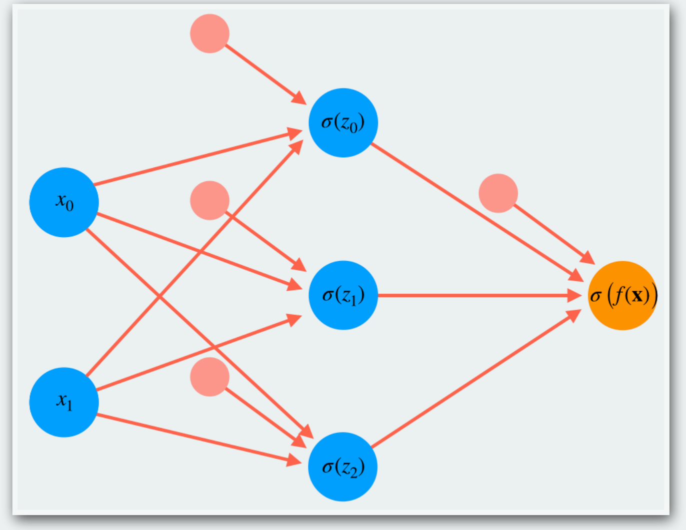


$$f\left(h\left(g(x)\right)\right)$$



Chain rule says:

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



$$\sigma\left(w_{0,1} + \sigma(w_{0,0} + x_0 w_{1,0} + x_1 w_{2,0})w_{1,1} + \sigma(w_{3,0} + x_0 w_{4,0} + x_1 w_{5,0})w_{2,1} + \sigma(w_{6,0} + x_0 w_{7,0} + x_1 w_{8,0})w_{3,1}\right) = \sigma\left(f(x)\right)$$

3Blue1Brown Chain Rule video

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

$$m \longrightarrow g \longrightarrow h \longrightarrow f \longrightarrow$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$x_0 = 1$$

$$x_1 = 1.1$$

$$x \xrightarrow{1} m \xrightarrow{-1} g \xrightarrow{0.37} h \xrightarrow{1.37} f \xrightarrow{0.73}$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$x_0 = 1$$

$$\boxed{x_1 = 1.1}$$

$$x \xrightarrow{1.1} m \xrightarrow{-1.1} g \xrightarrow{0.33} h \xrightarrow{1.33} f \xrightarrow{0.75}$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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

$$x \xrightarrow{1} m \xrightarrow{-1} g \xrightarrow{0.37} h \xrightarrow{1.37} f \xrightarrow{0.73}$$

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!

$$x \xrightarrow{1} m \xrightarrow{-1} g \xrightarrow{0.37} h \xrightarrow{1.37} f \xrightarrow{0.73}$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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:

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

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

$$h'(z) = 1$$

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

$$\begin{tikzcd} x \arrow[r, "1"] & m \arrow[r, "-1"] & g \arrow[r, "0.37"] & h \arrow[r, "1.37"] & f \arrow[r, "0.73"] \arrow[llll, bend left, "?"] & {} \end{tikzcd}$$

Backpropagation – simple example

Model:

$m(z) = -z$

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

$h(z) = z + 1$

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

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!

$$
\begin{tikzcd}
x \arrow[r, "1"] & m \arrow[r, "-1"] & g \arrow[r, "0.37"] & h \arrow[r, "1.37"] & f \arrow[r, "0.73"] \arrow[l, bend left, "?"] & {}
\end{tikzcd}
$$

Chain rule from *f* to *h*:

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

Model derivatives:

$m'(z) = -1$

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

$h'(z) = 1$

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

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:

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

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

$$h'(z) = 1$$

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

$$\begin{tikzcd} x \arrow[r, "1"] & m \arrow[r, "-1"] & g \arrow[r, "0.37"] & h \arrow[r, "1.37"] & f \arrow[l, bend left, "-.53"] \arrow[r, "0.73"] & {} \end{tikzcd}$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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:

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

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

$$h'(z) = 1$$

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

$$x \xrightarrow{1} m \xrightarrow{-1} g \xrightarrow{0.37} h \xrightarrow{1.37} f \xrightarrow{0.73}$$

$$h \xrightarrow{?} g \qquad f \xrightarrow{-.53} h$$

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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*:

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

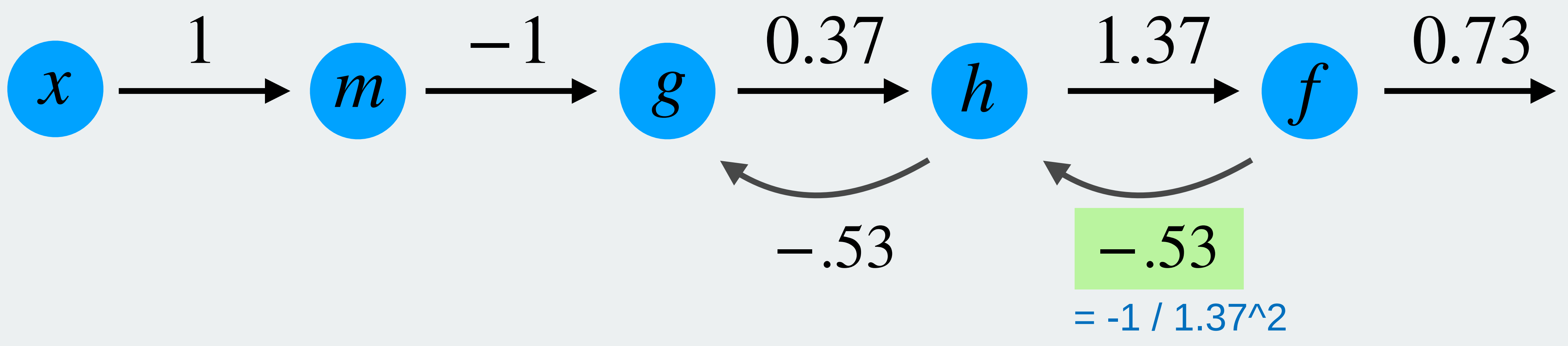
Model derivatives:

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

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

$$h'(z) = 1$$

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



Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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*:

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

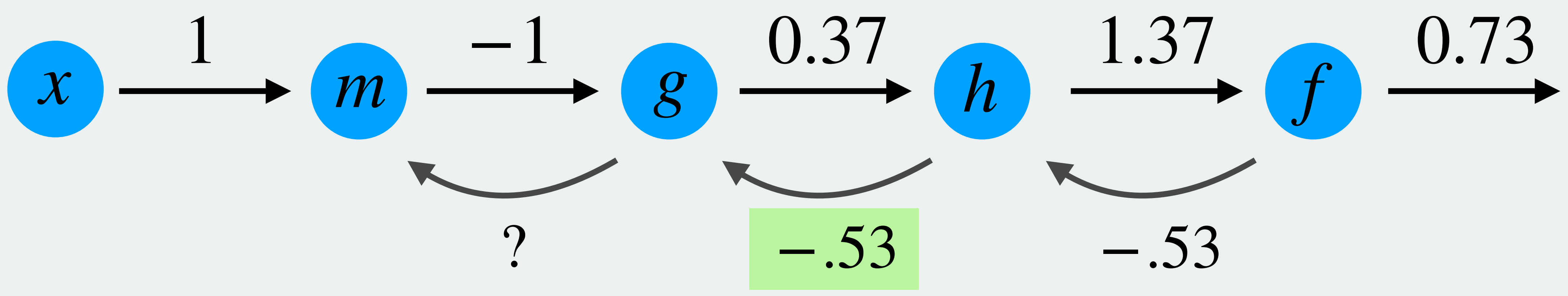
Model derivatives:

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

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

$$h'(z) = 1$$

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



Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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*:

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

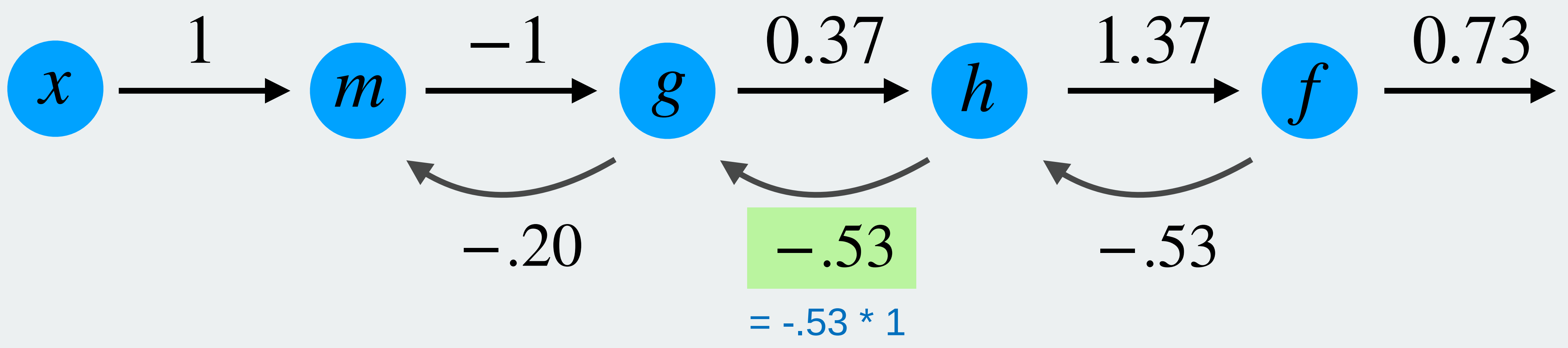
Model derivatives:

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

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

$$h'(z) = 1$$

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



Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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:

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

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

$$h'(z) = 1$$

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

$$x \xrightarrow{1} m \xrightarrow{-1} g \xrightarrow{0.37} h \xrightarrow{1.37} f \xrightarrow{0.73}$$

$$.20 \qquad -.20 \qquad -.53 \qquad -.53$$

= -.53 * 1 * exp(-1)

Backpropagation – simple example

Model:

$$m(z) = -z$$

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

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

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

Data:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

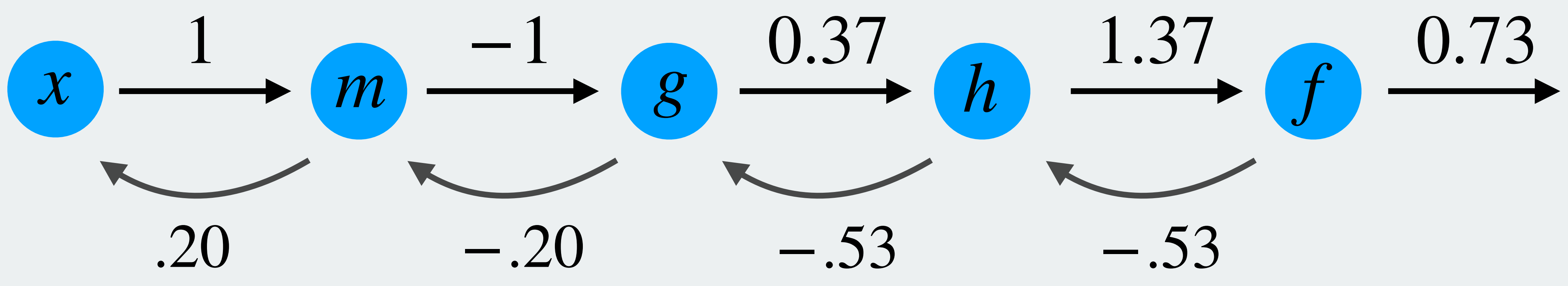
Model derivatives:

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

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

$$h'(z) = 1$$

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



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:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

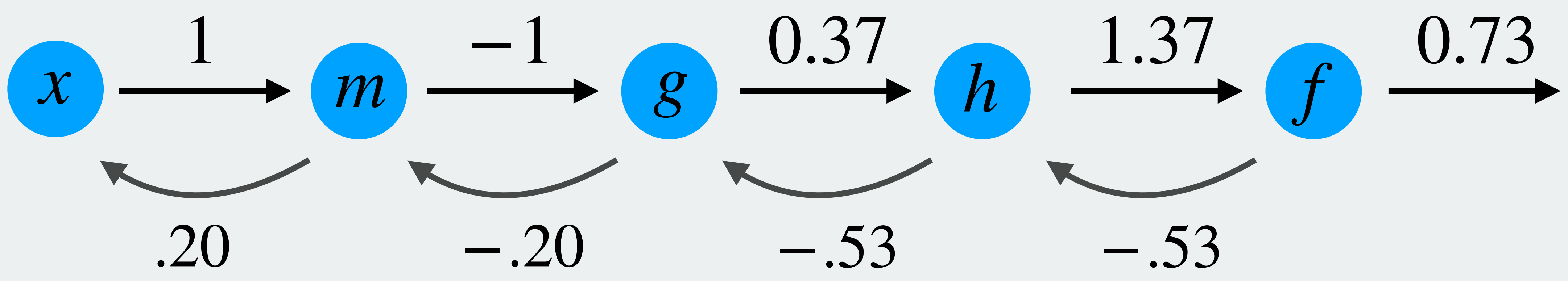
Model derivatives:

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

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

$$h'(z) = 1$$

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

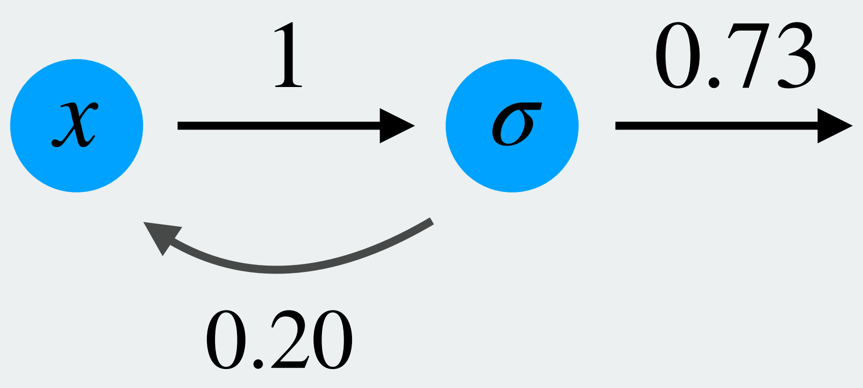


Sigmoid function:

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

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:

$$\boxed{x_0 = 1}$$

$$x_1 = 1.1$$

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:

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

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

$$h'(z) = 1$$

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

$$\begin{tikzcd} x \arrow[r, "1"] & m \arrow[r, "-1"] \arrow[l, bend left, ".20"] & g \arrow[r, "0.37"] \arrow[l, bend left, "-.20"] & h \arrow[r, "1.37"] \arrow[l, bend left, "-.53"] & f \arrow[r, "0.73"] \arrow[l, bend left, "-.53"] & {} \end{tikzcd}$$

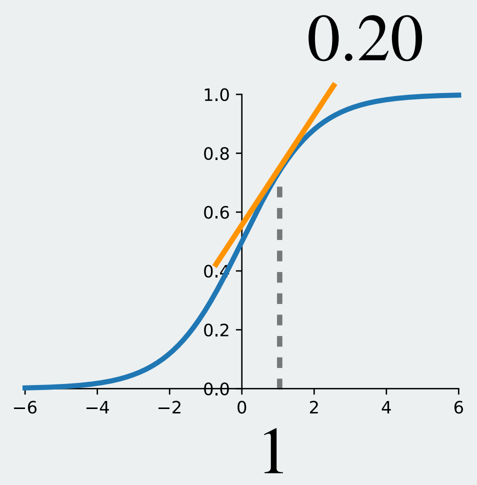
Sigmoid function:

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

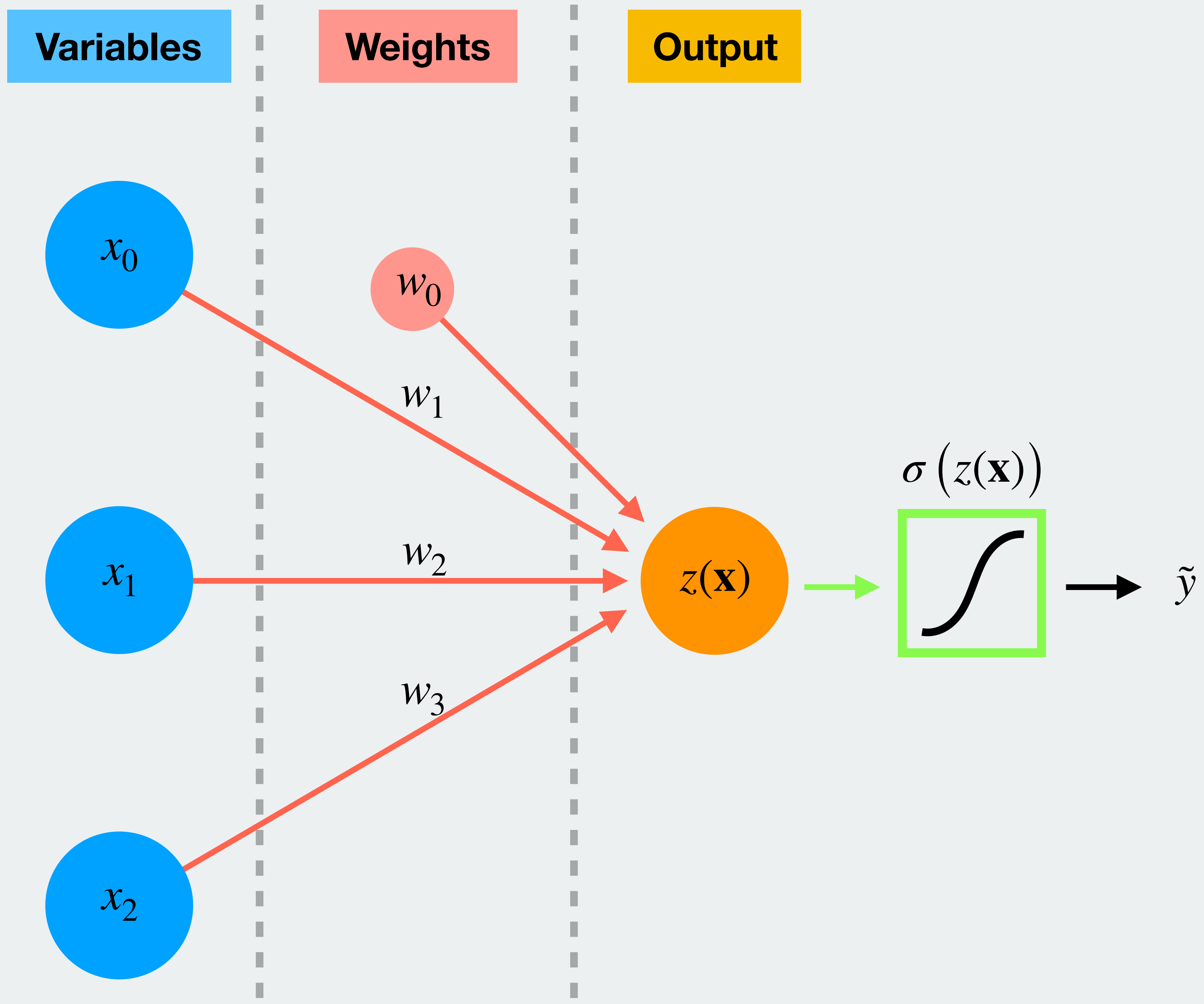
Sigmoid derivative:

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

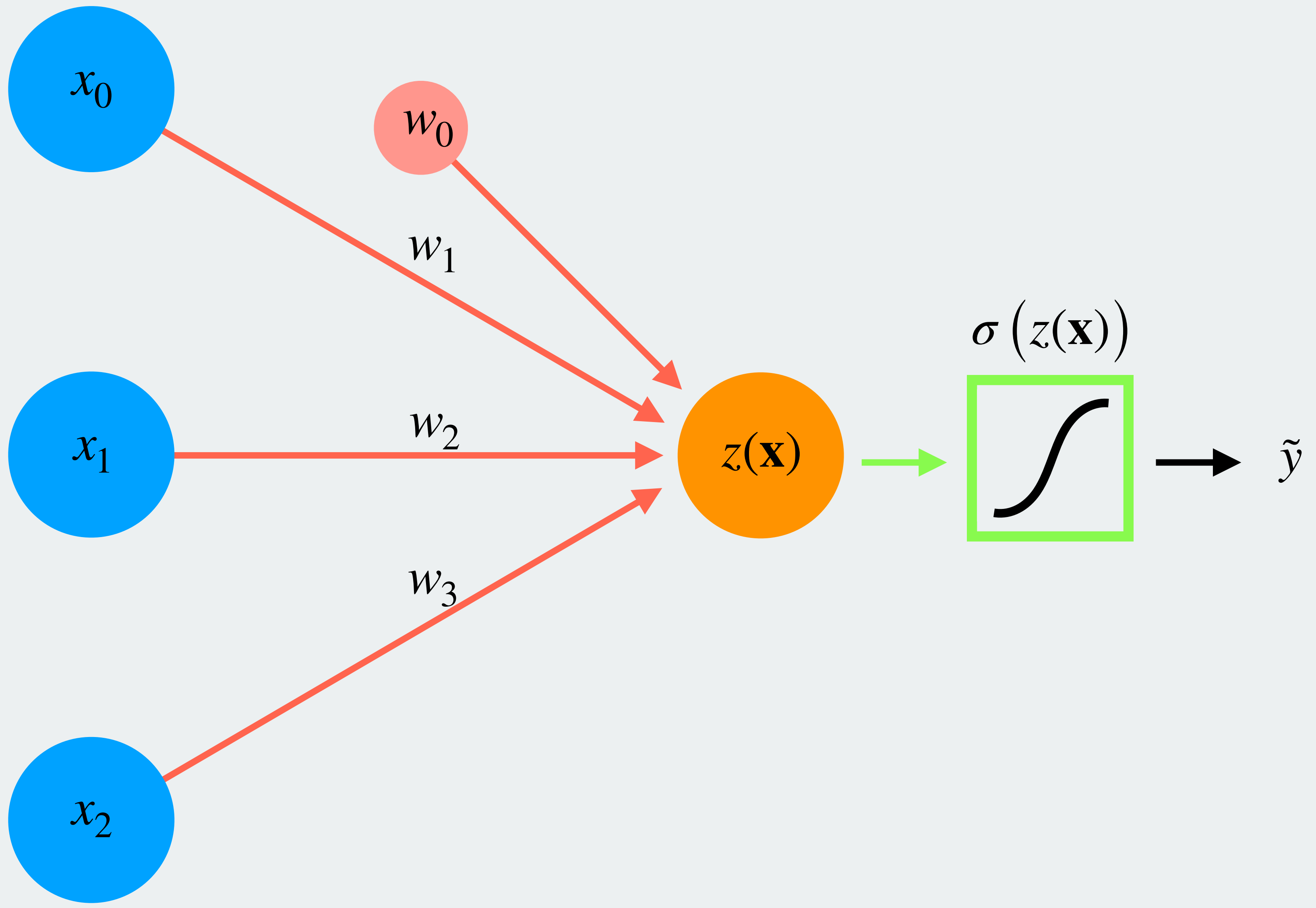
$$\begin{tikzcd} x \arrow[r, "1"] & \sigma \arrow[r, "0.73"] \arrow[l, bend left, "0.20"] & {} \end{tikzcd}$$



Backpropagation – on a neural network



Backpropagation – on a neural network



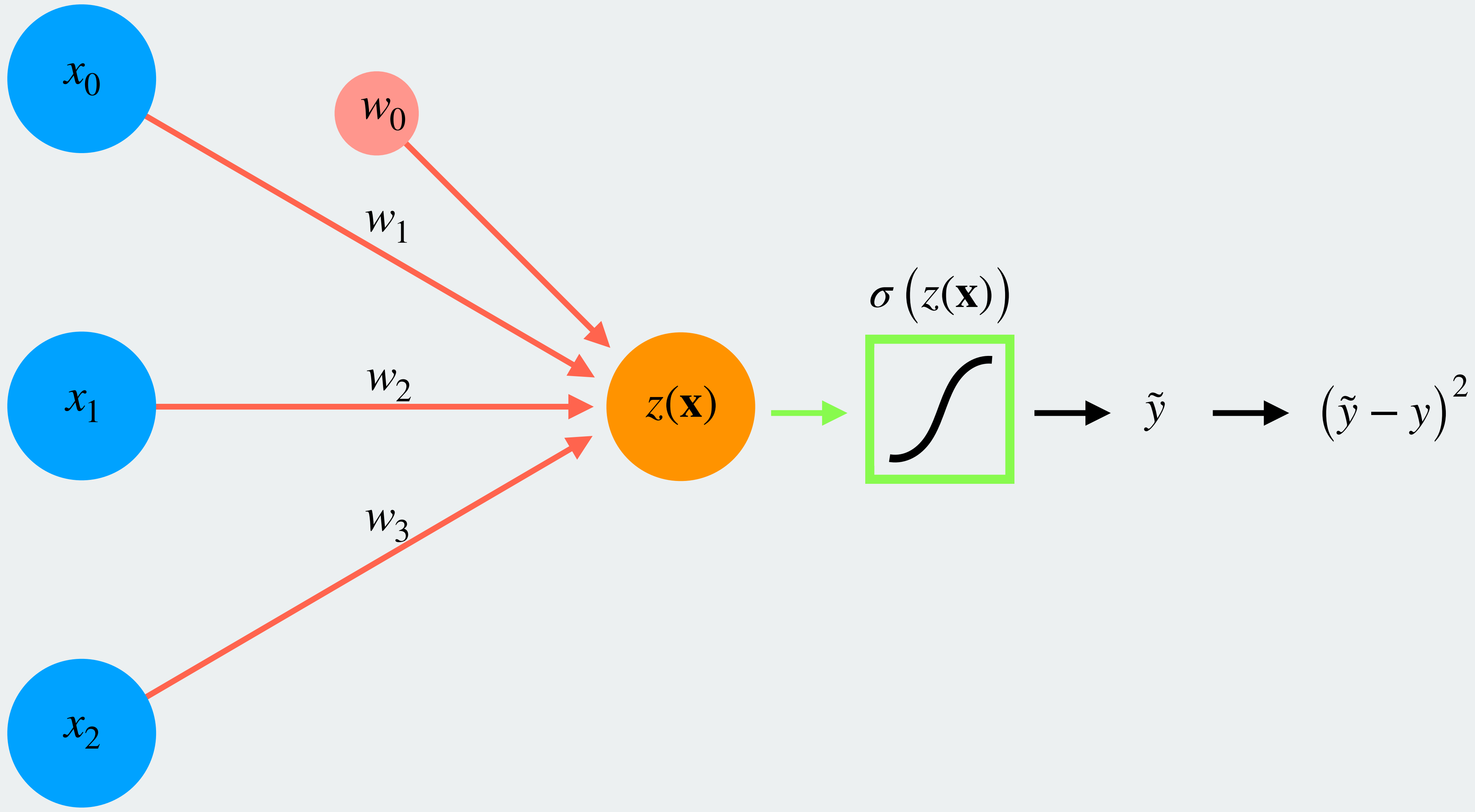
Model:

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

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

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

Backpropagation – on a neural network



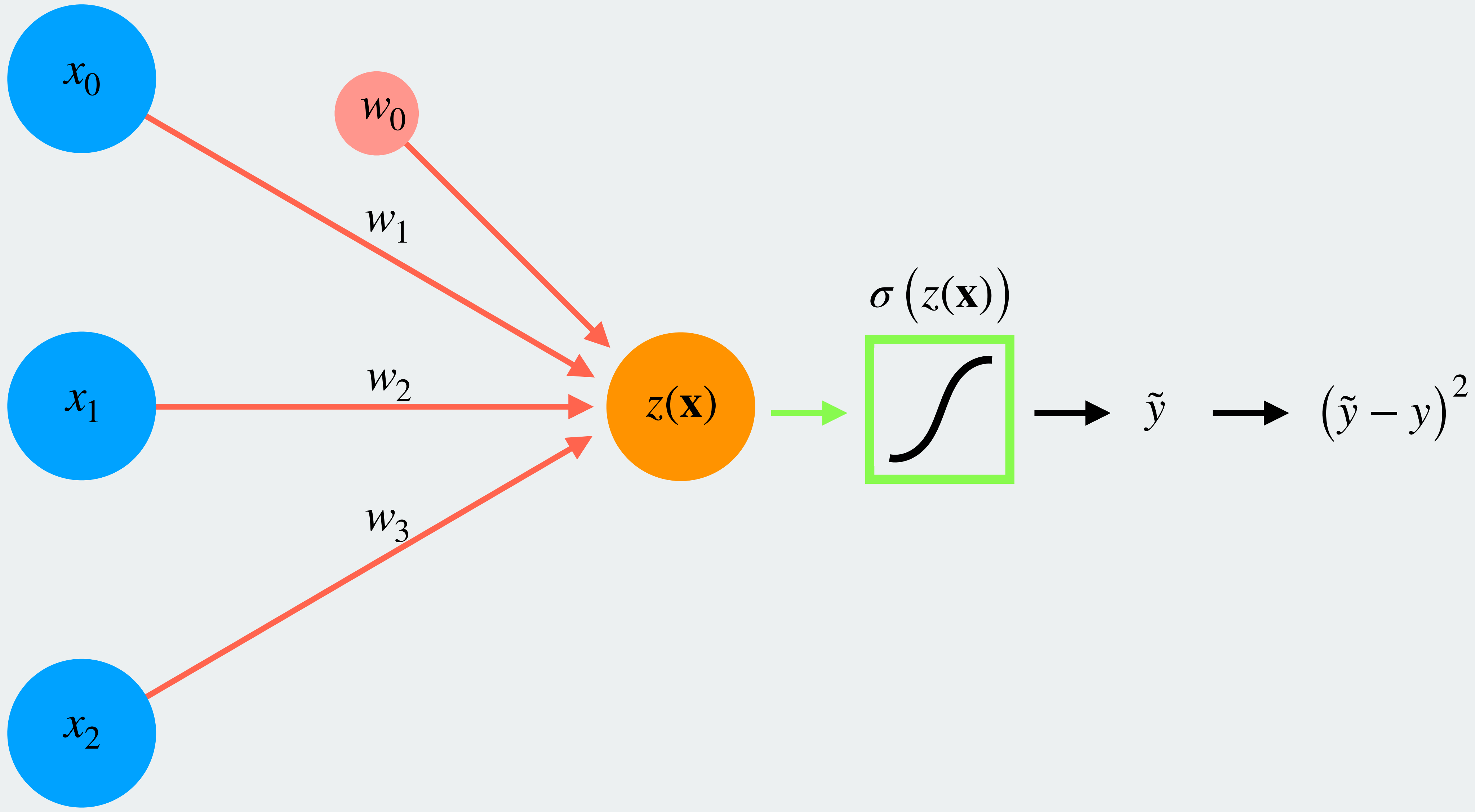
Model:

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

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

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

Backpropagation – on a neural network



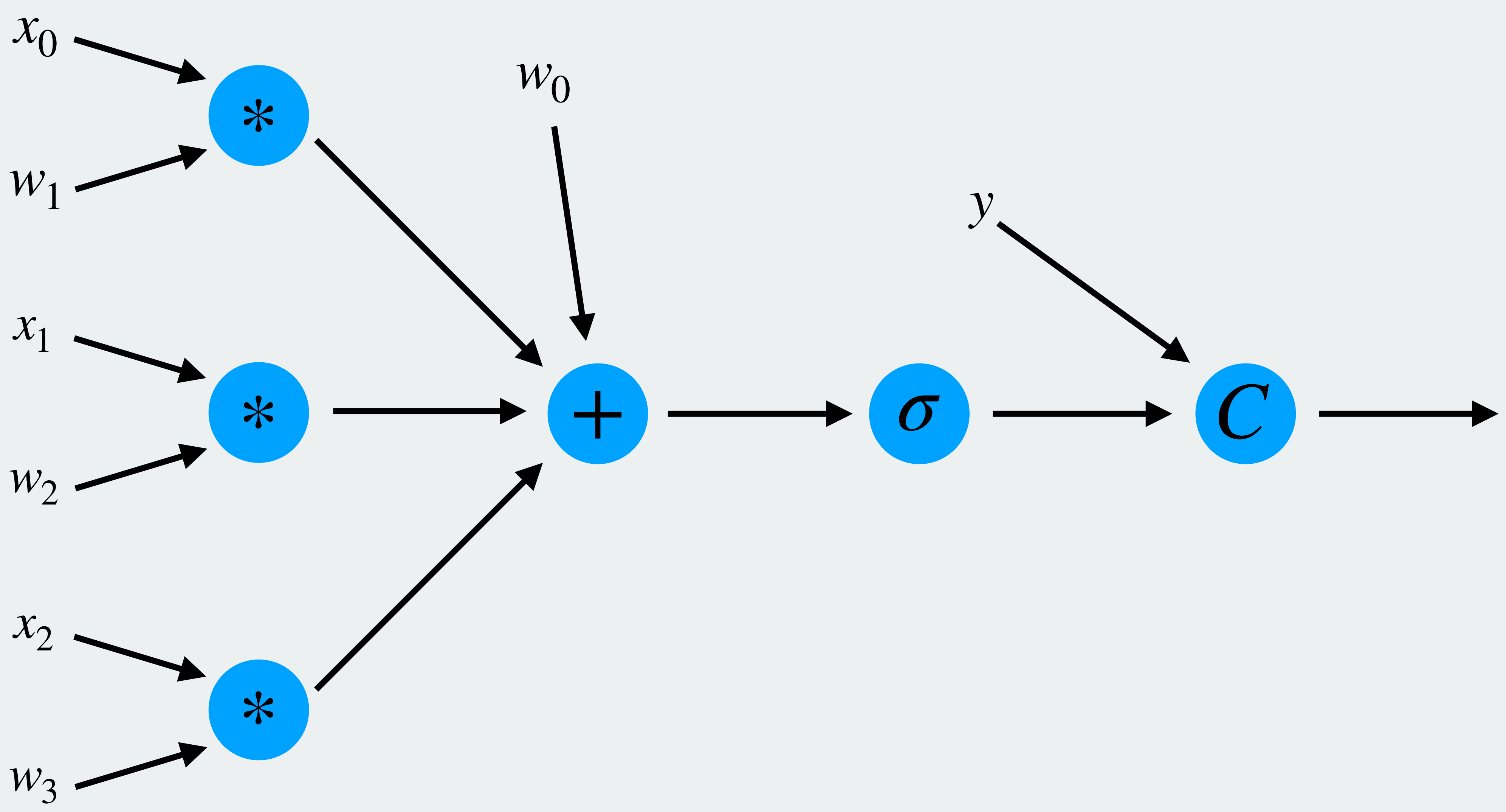
Model:

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

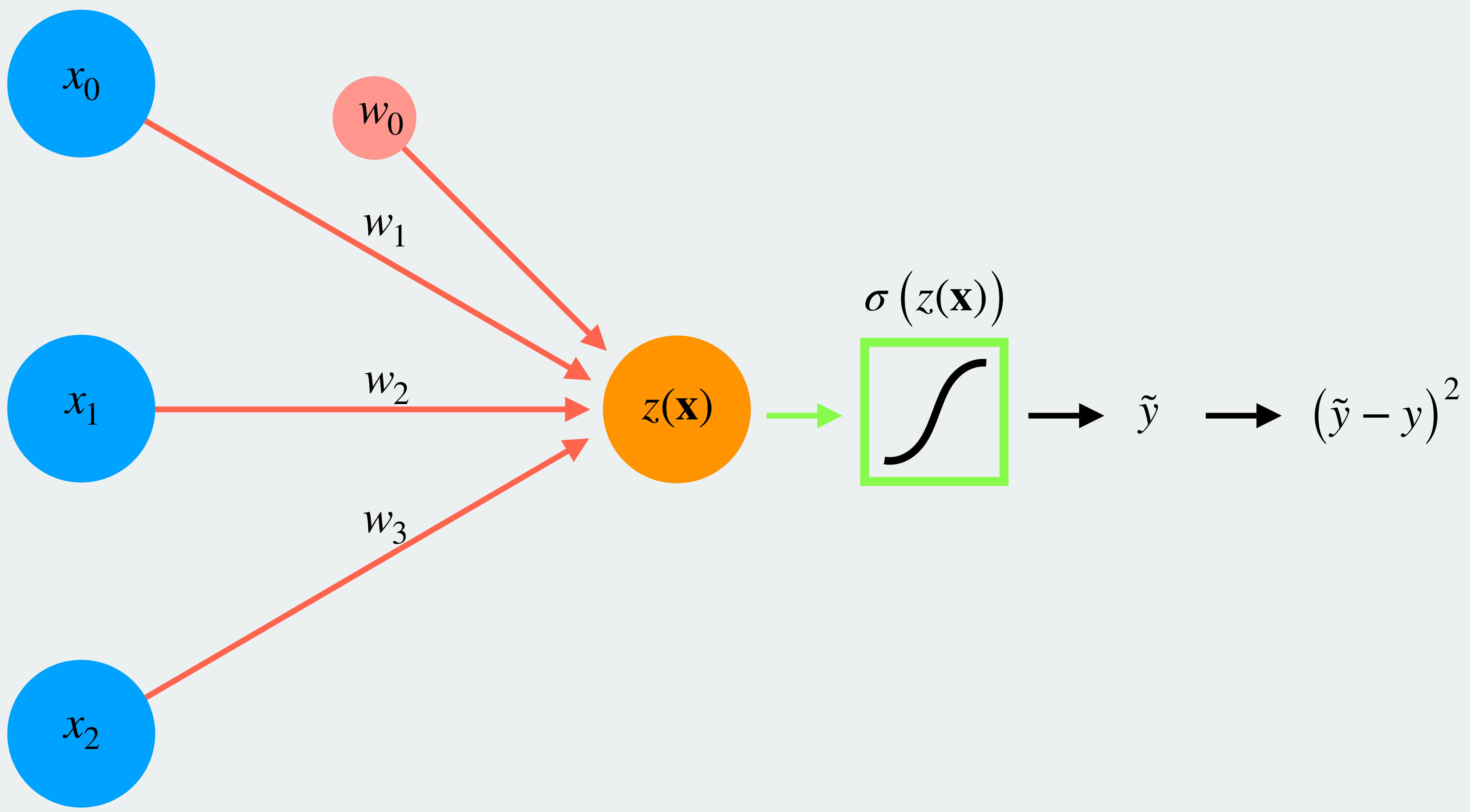
$$\sigma(x) = \frac{1}{1 + \exp\left(-x\right)}$$

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

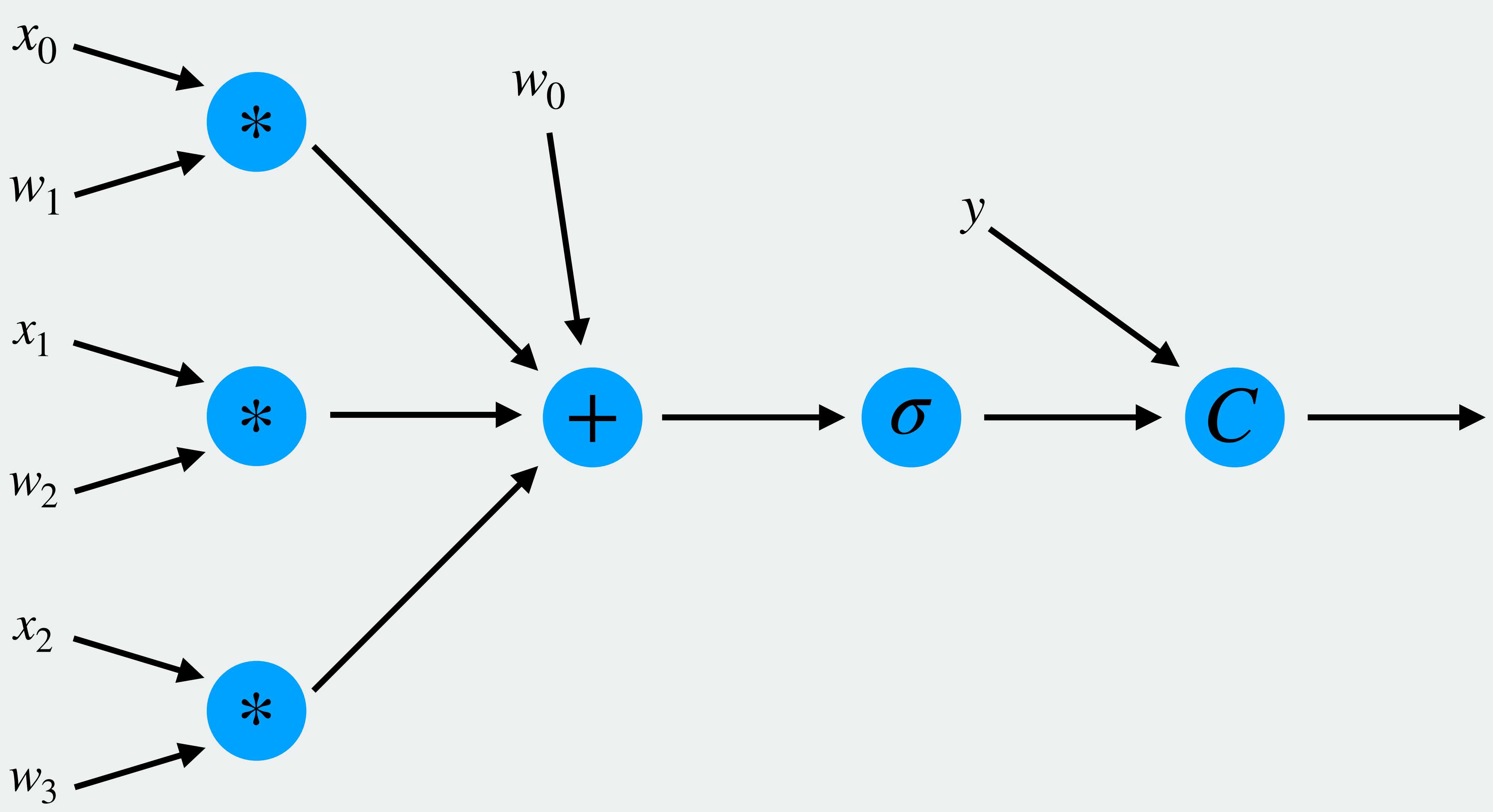
As a computational graph:



Backpropagation – on a neural network



As a computational graph:



Model:

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

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

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

Weights:

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

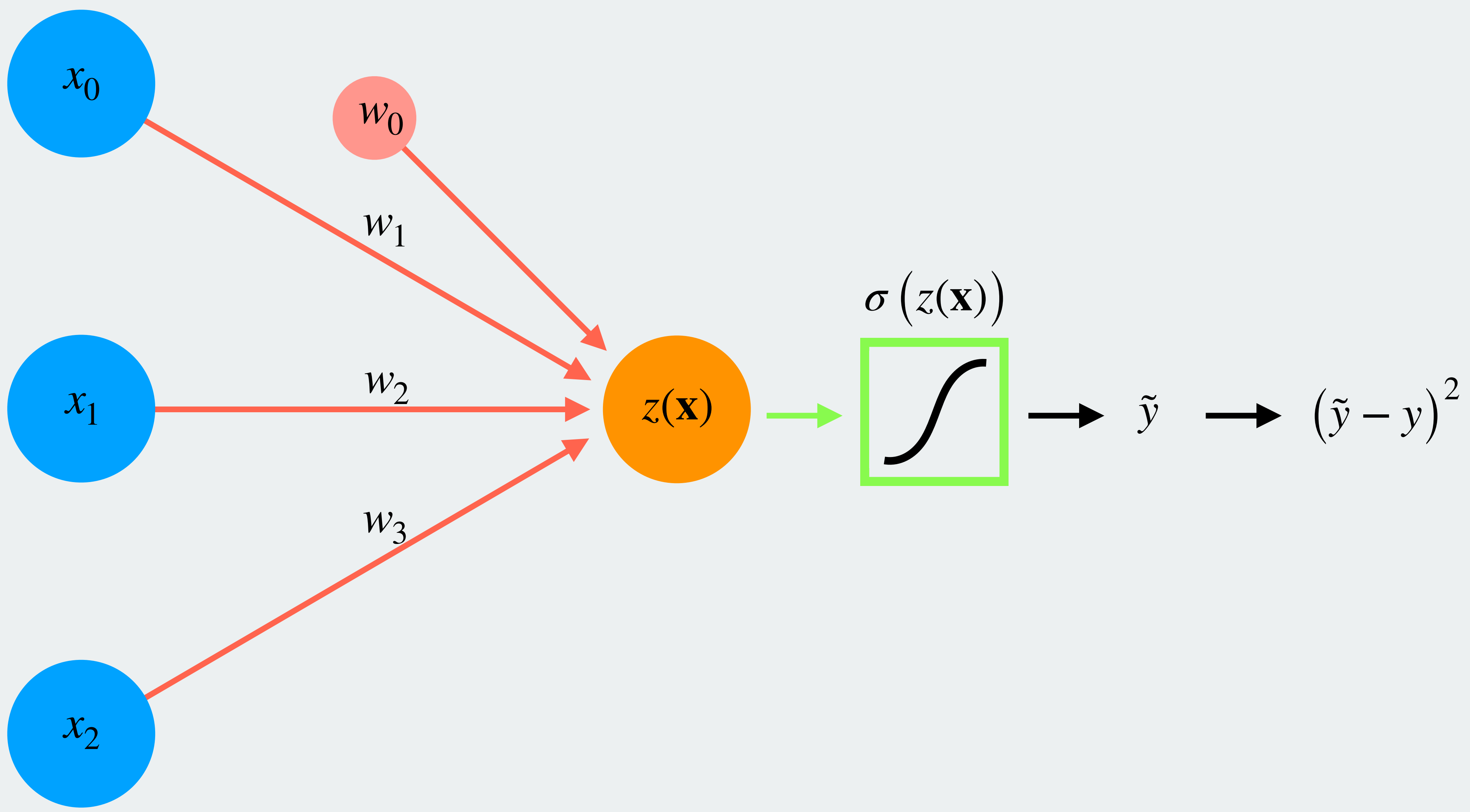
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

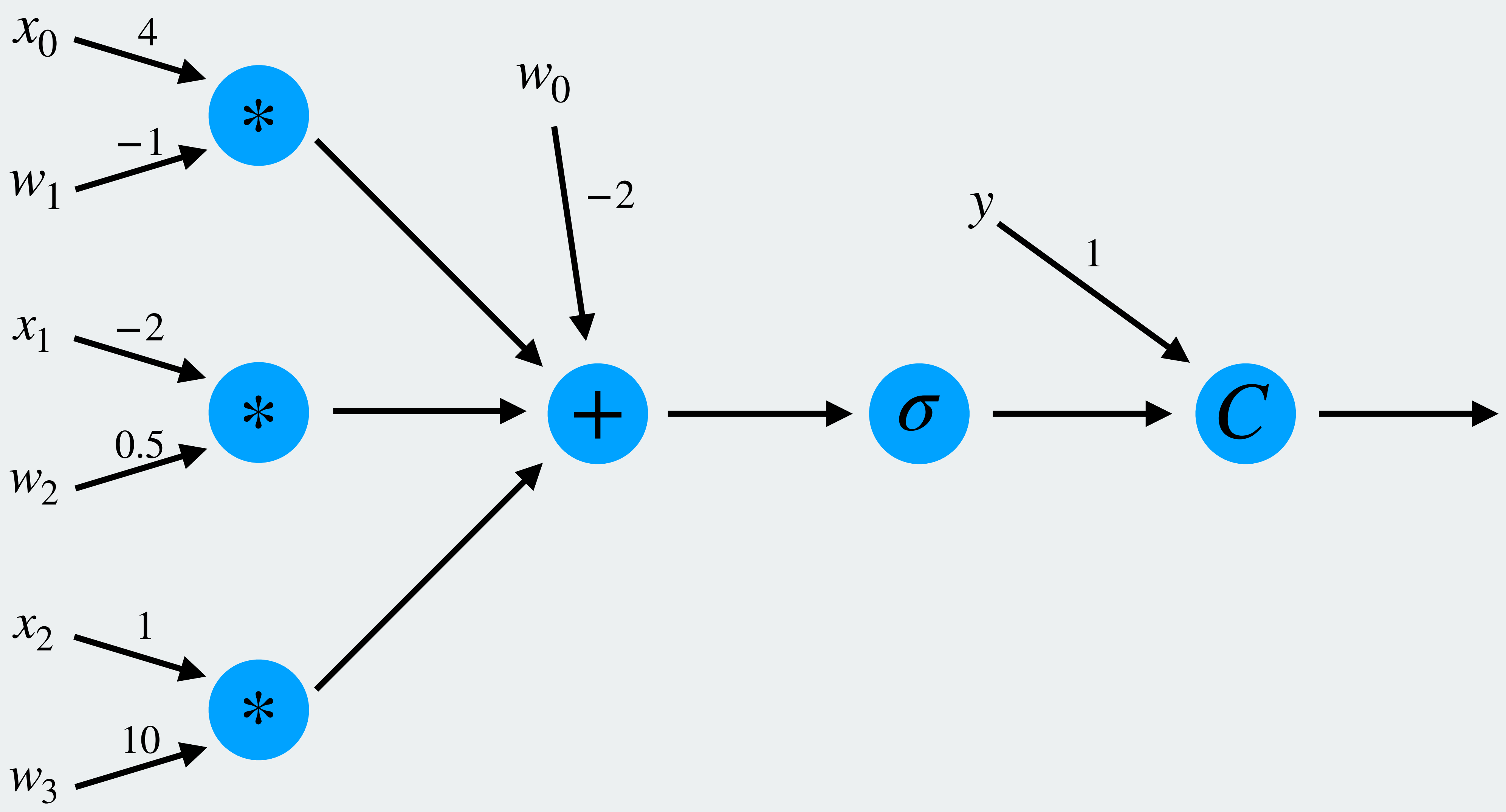
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Forward pass



Model:

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

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

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

Weights:

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

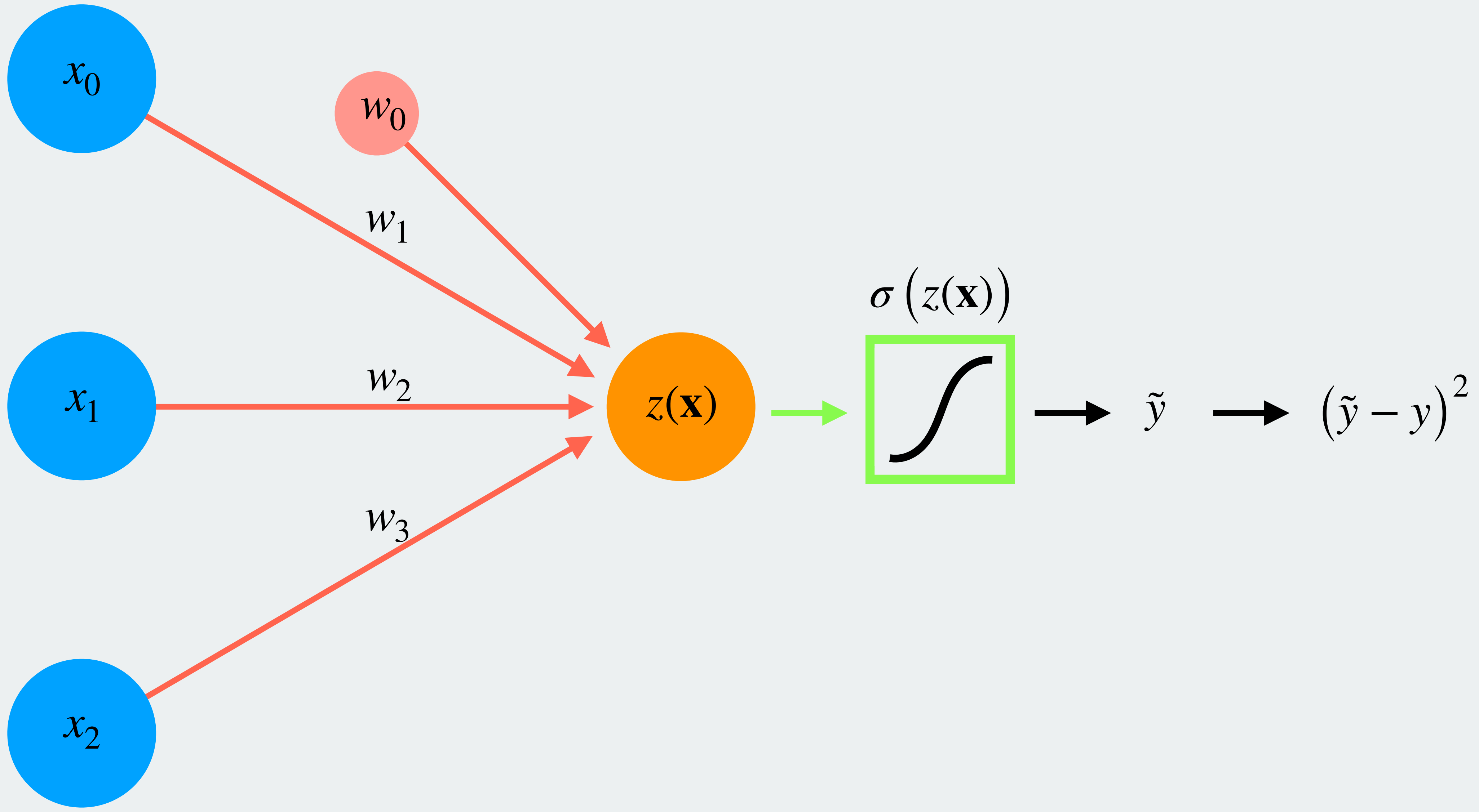
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

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

Backpropagation – on a neural network



Model:

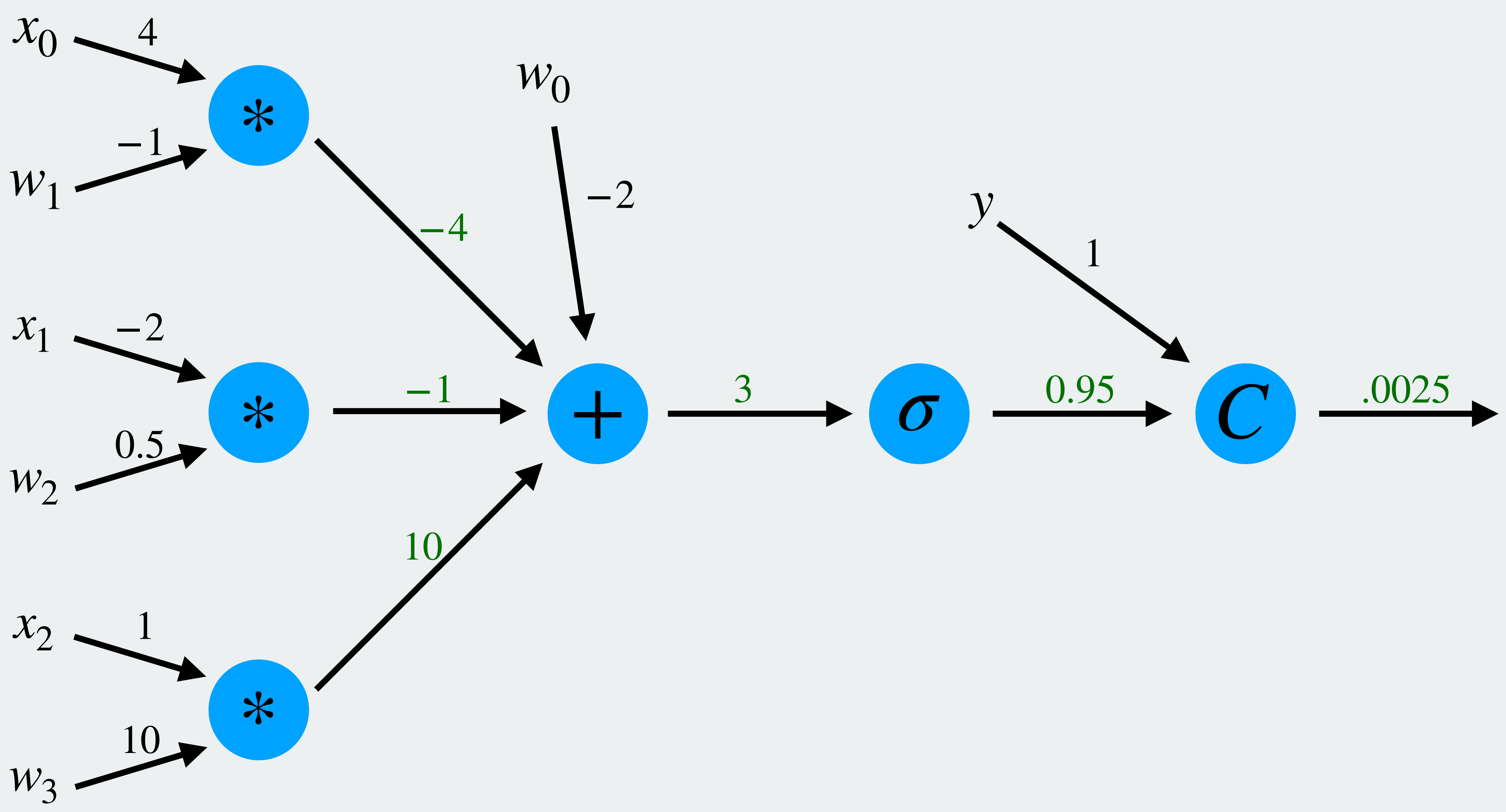
$$w_0 + x_0 w_1 + x_1 w_2 + x_2 w_3 = z(\mathbf{x})$$

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

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

As a computational graph:

Forward pass



Weights:

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

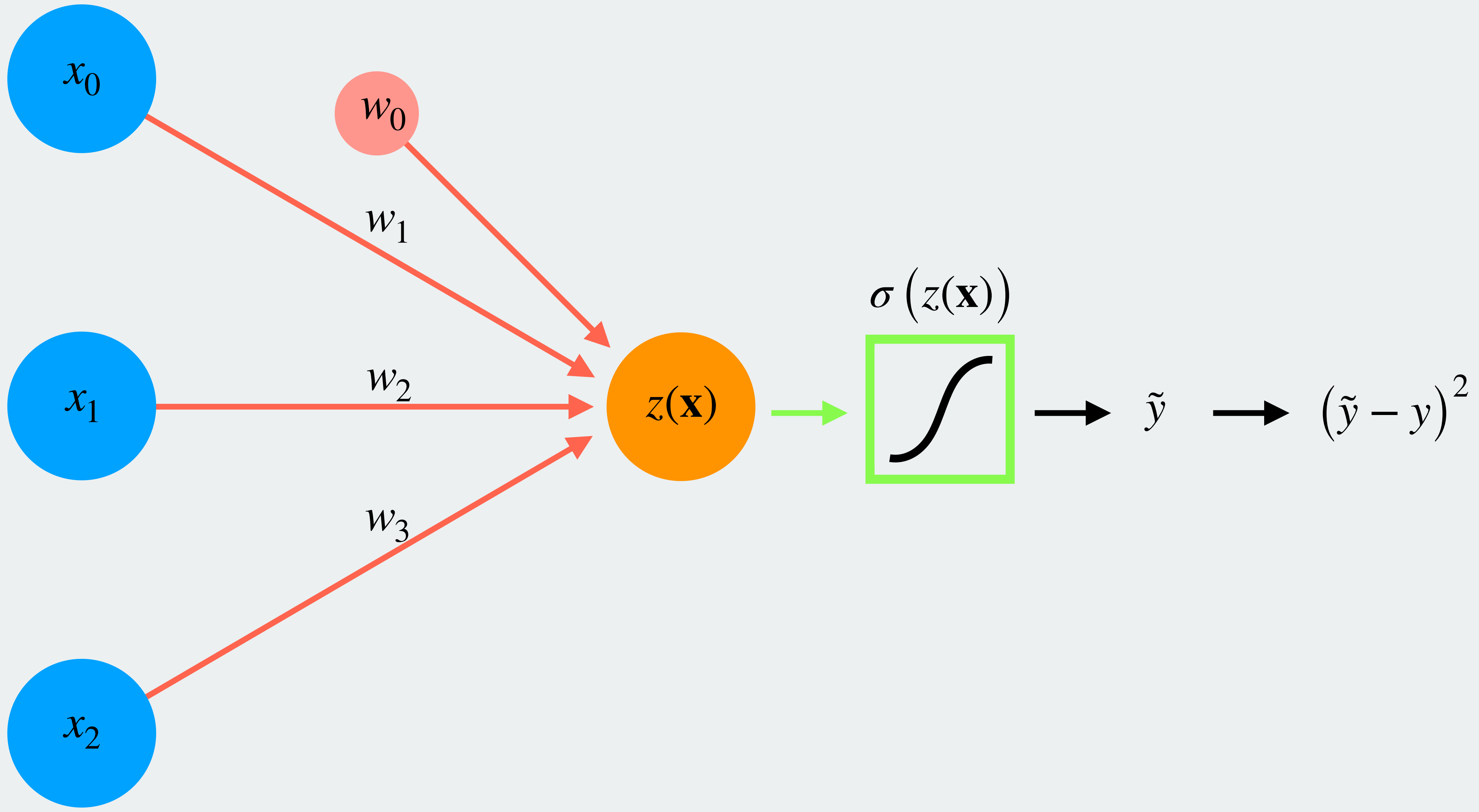
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

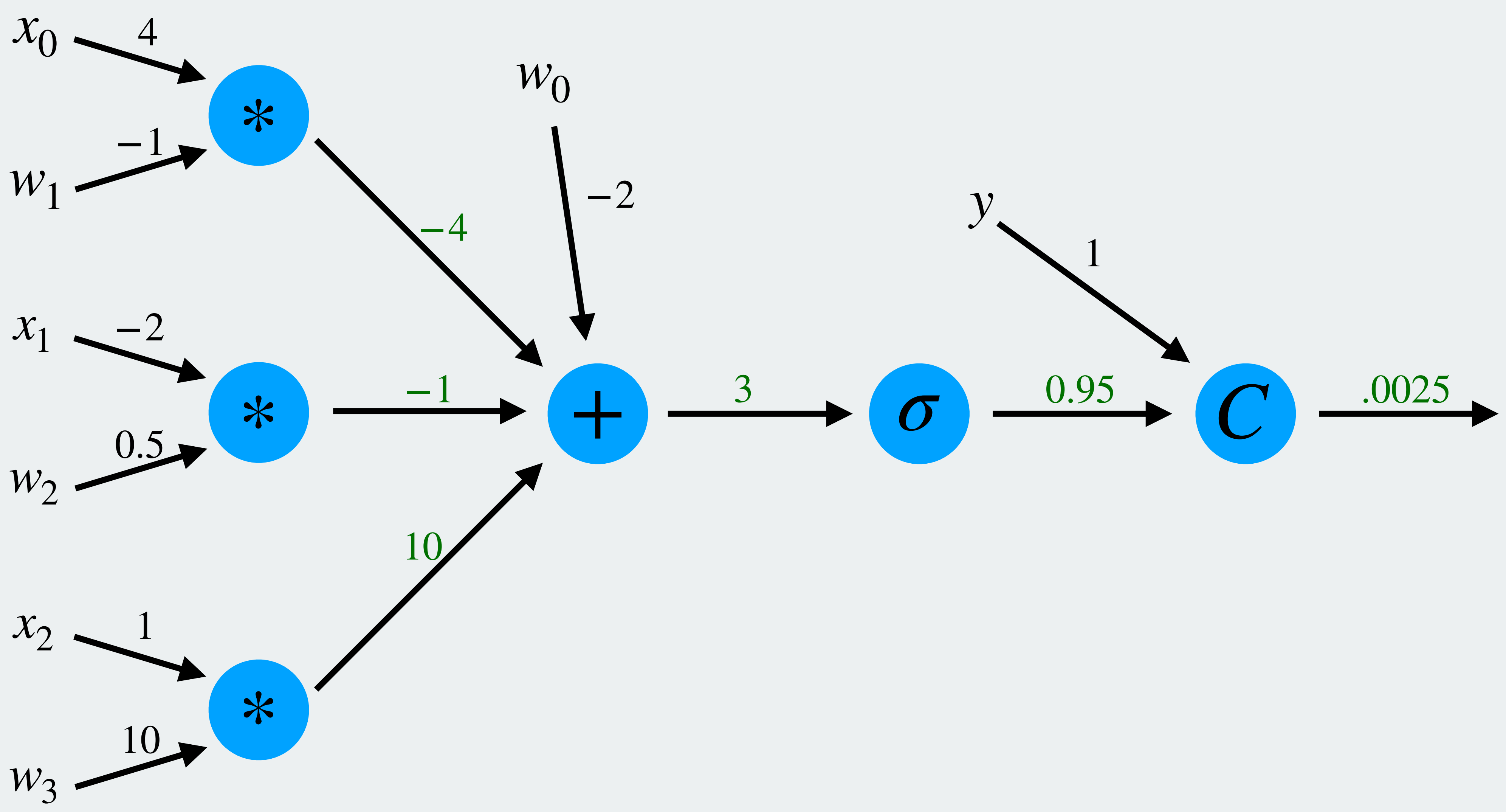
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



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:

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

Weights:

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

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

Data:

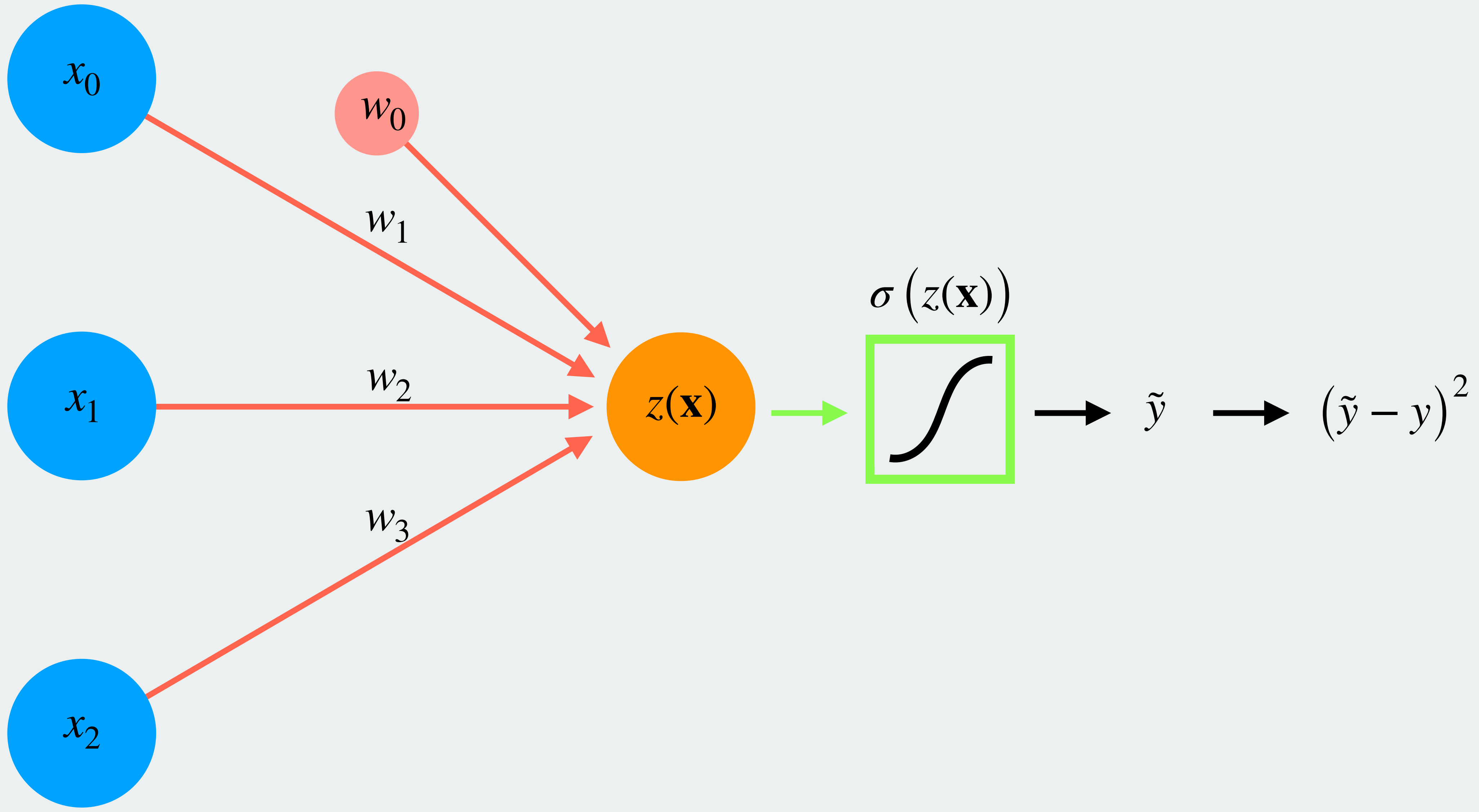
$$\mathbf{x} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}$$

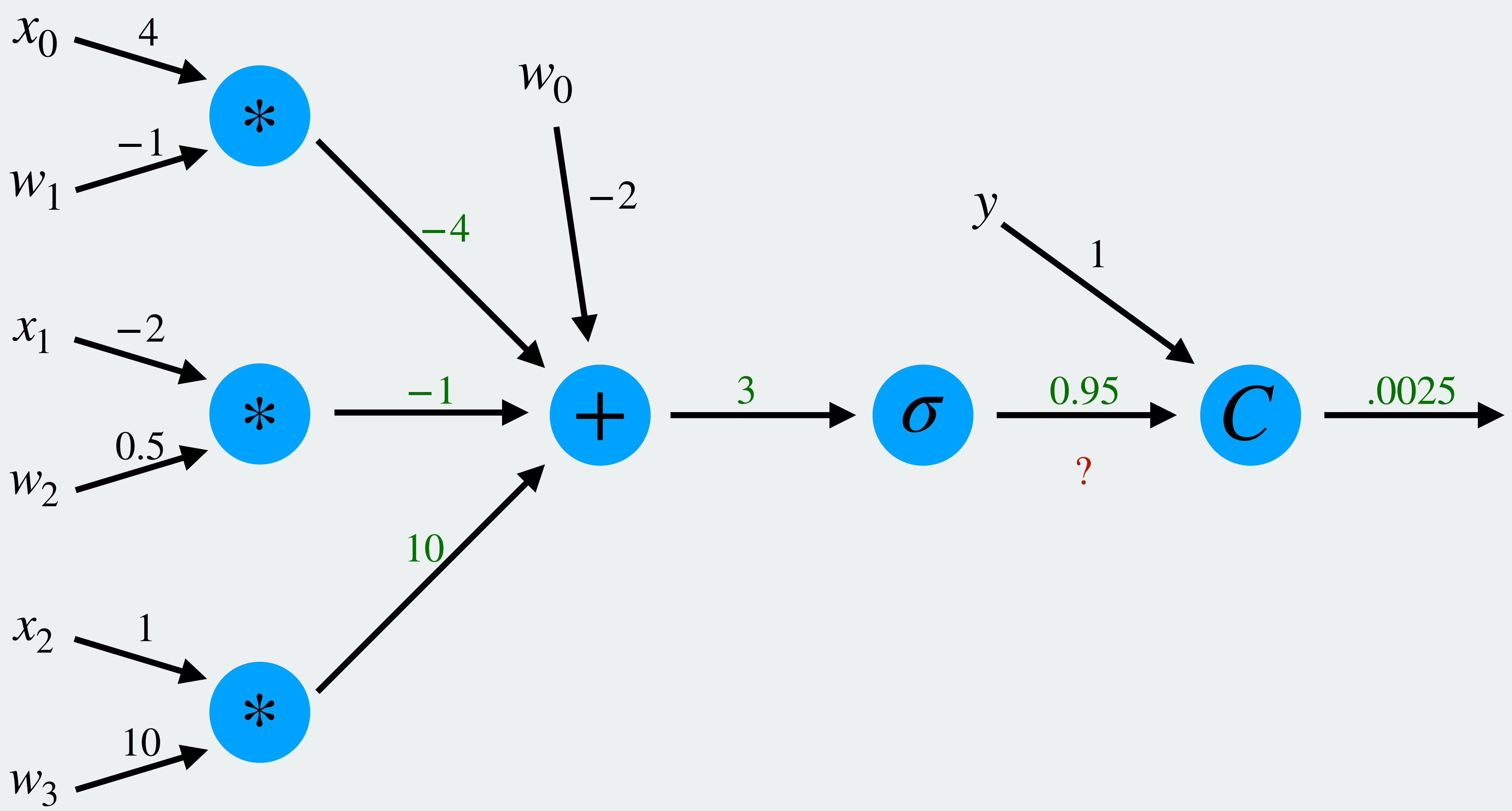
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network

As a computational graph:

Backward pass





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:

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

Weights:

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

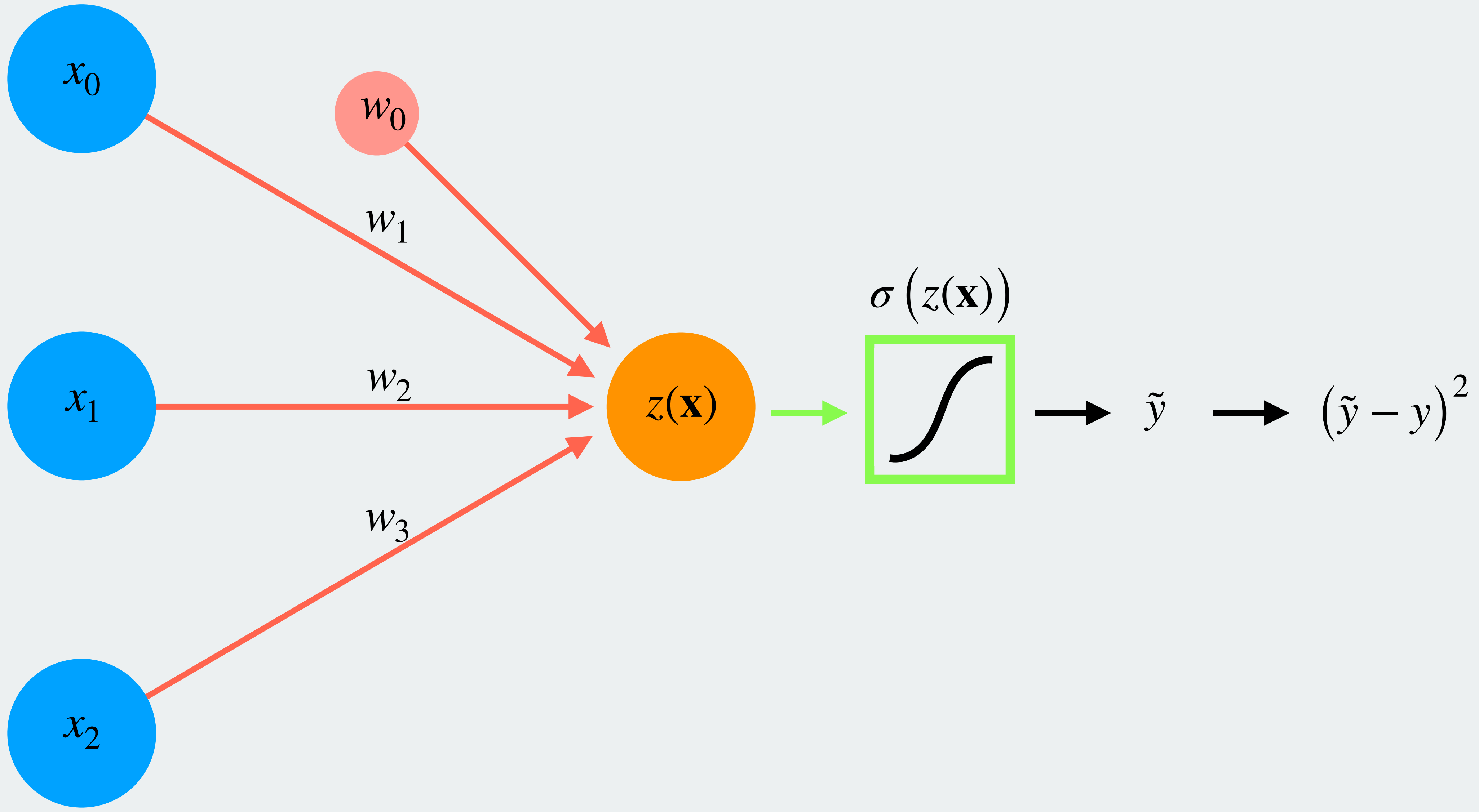
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

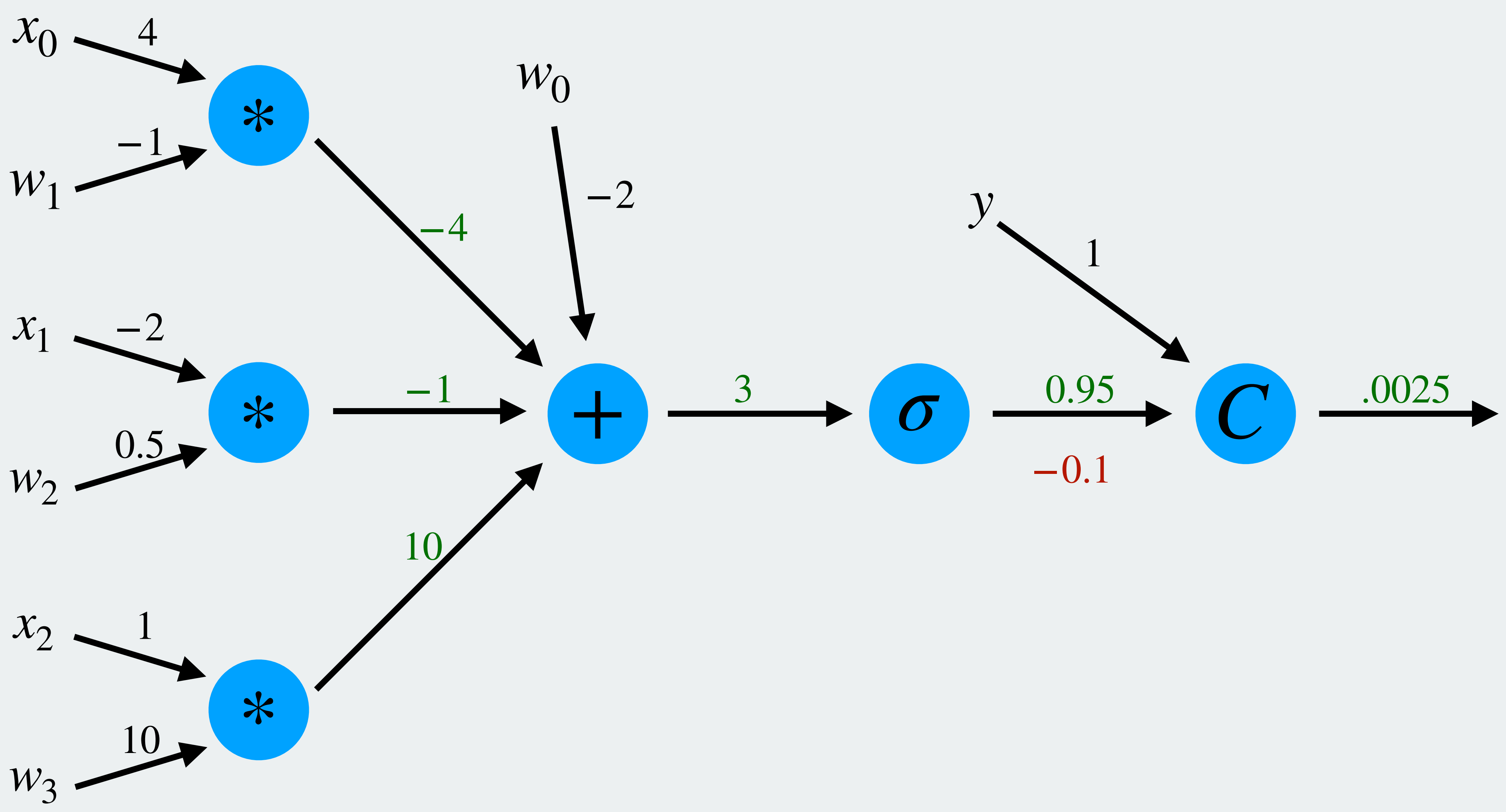
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



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:

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

Weights:

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

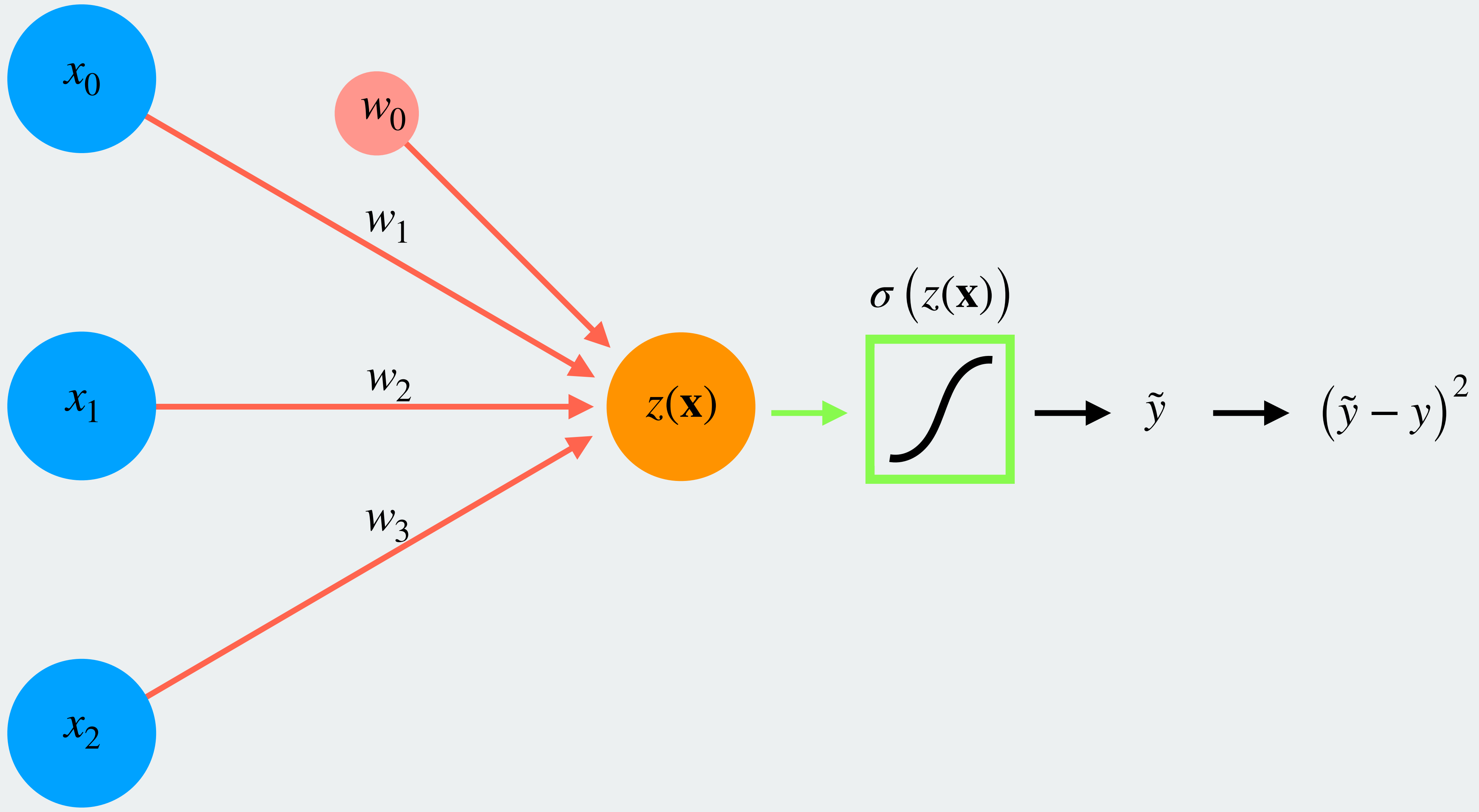
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

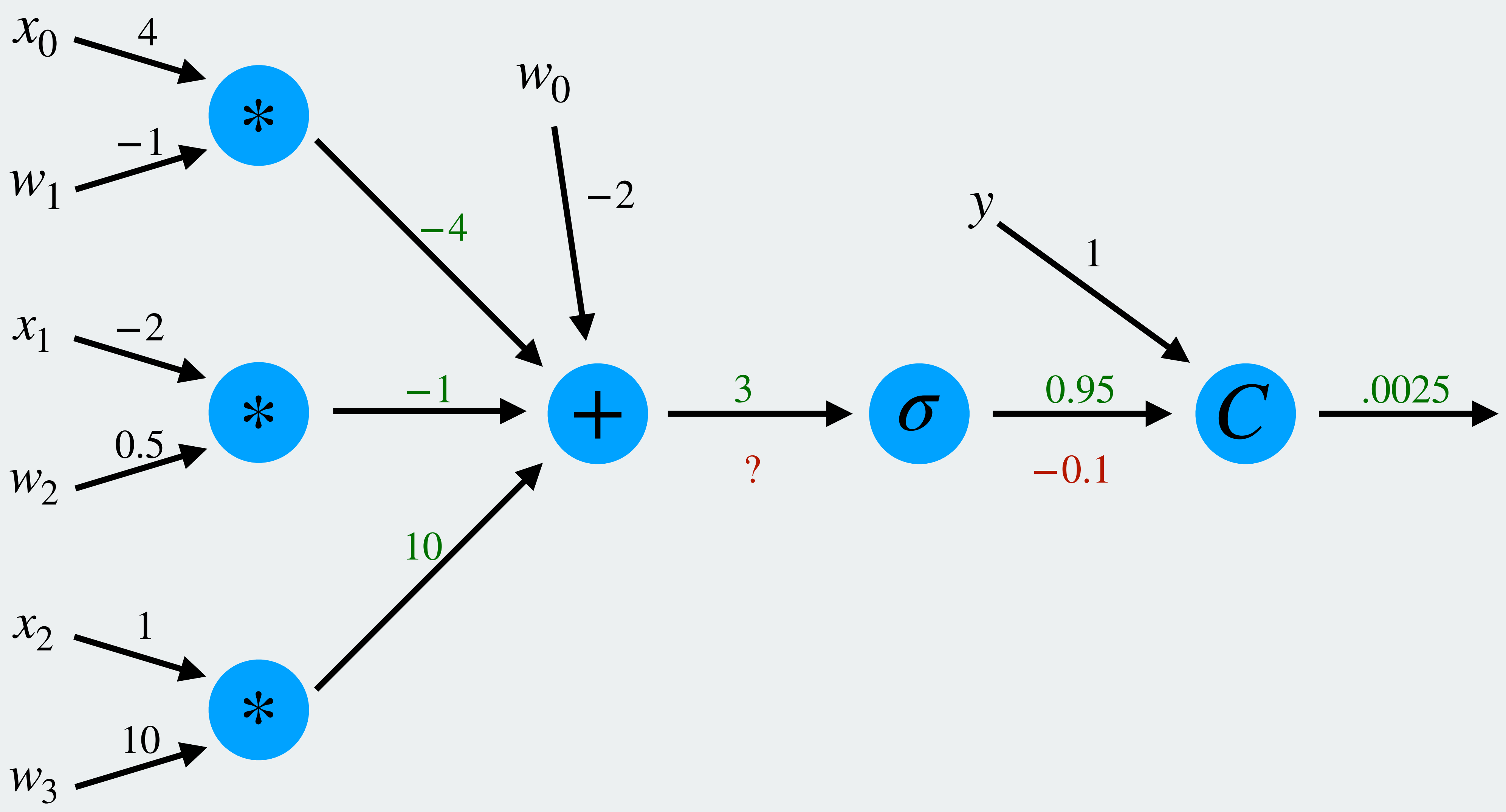
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



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:

$$\sigma'(x) = \sigma(x)\left(1-\sigma(x)\right)$$

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

Weights:

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

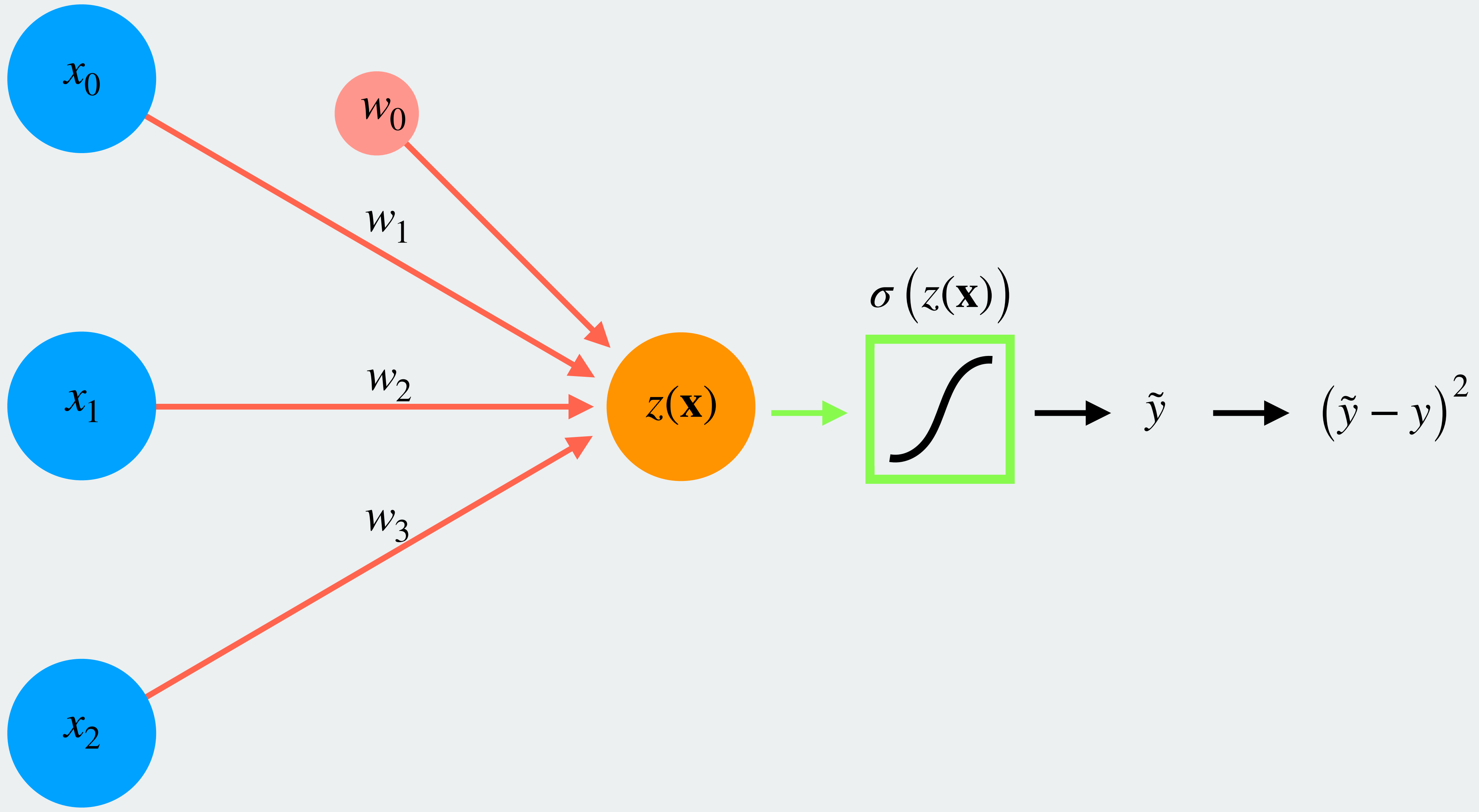
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

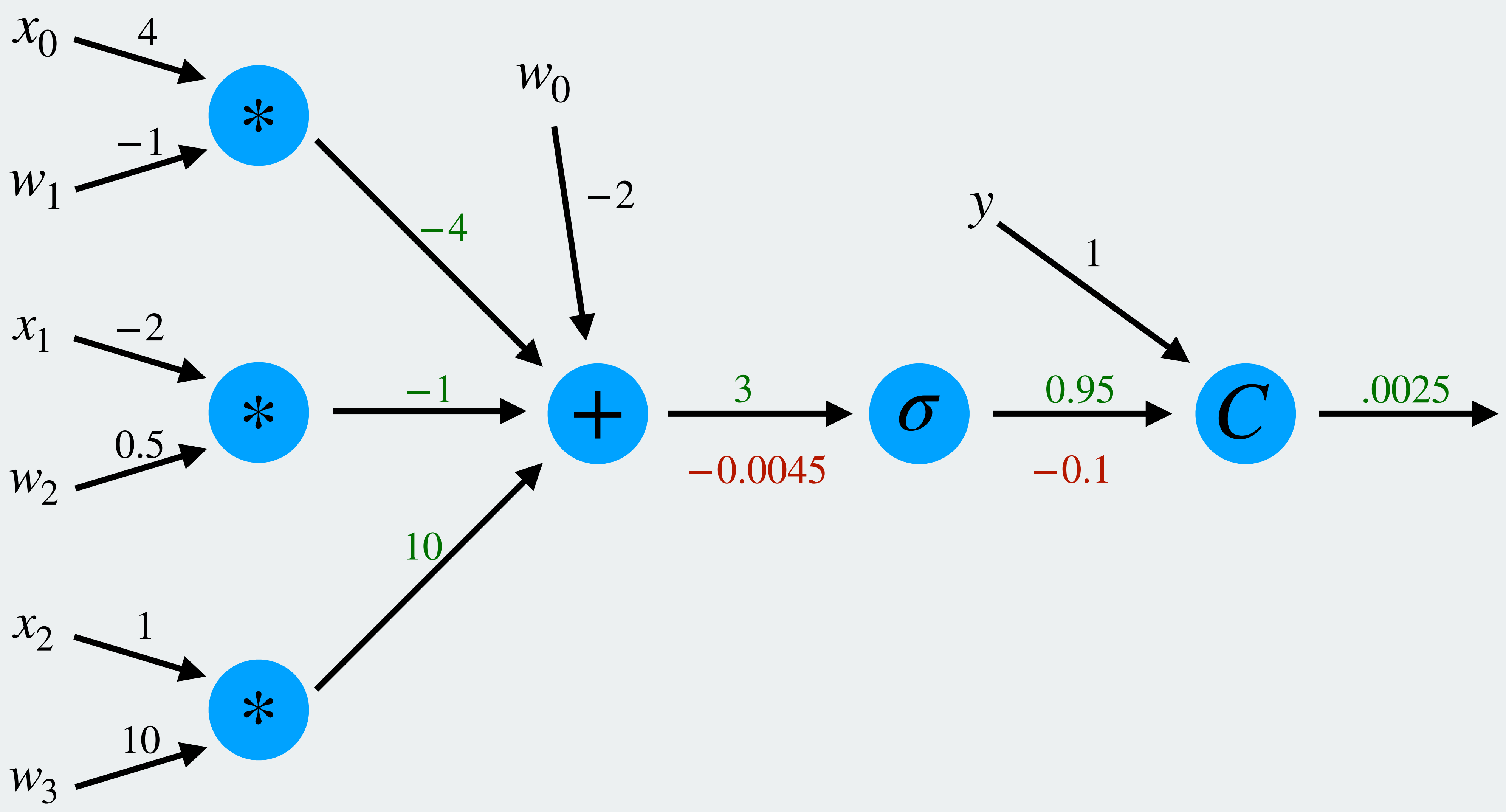
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is like a function: $y = \sum_{i=0}^{N} z_n$

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

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

Weights:

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

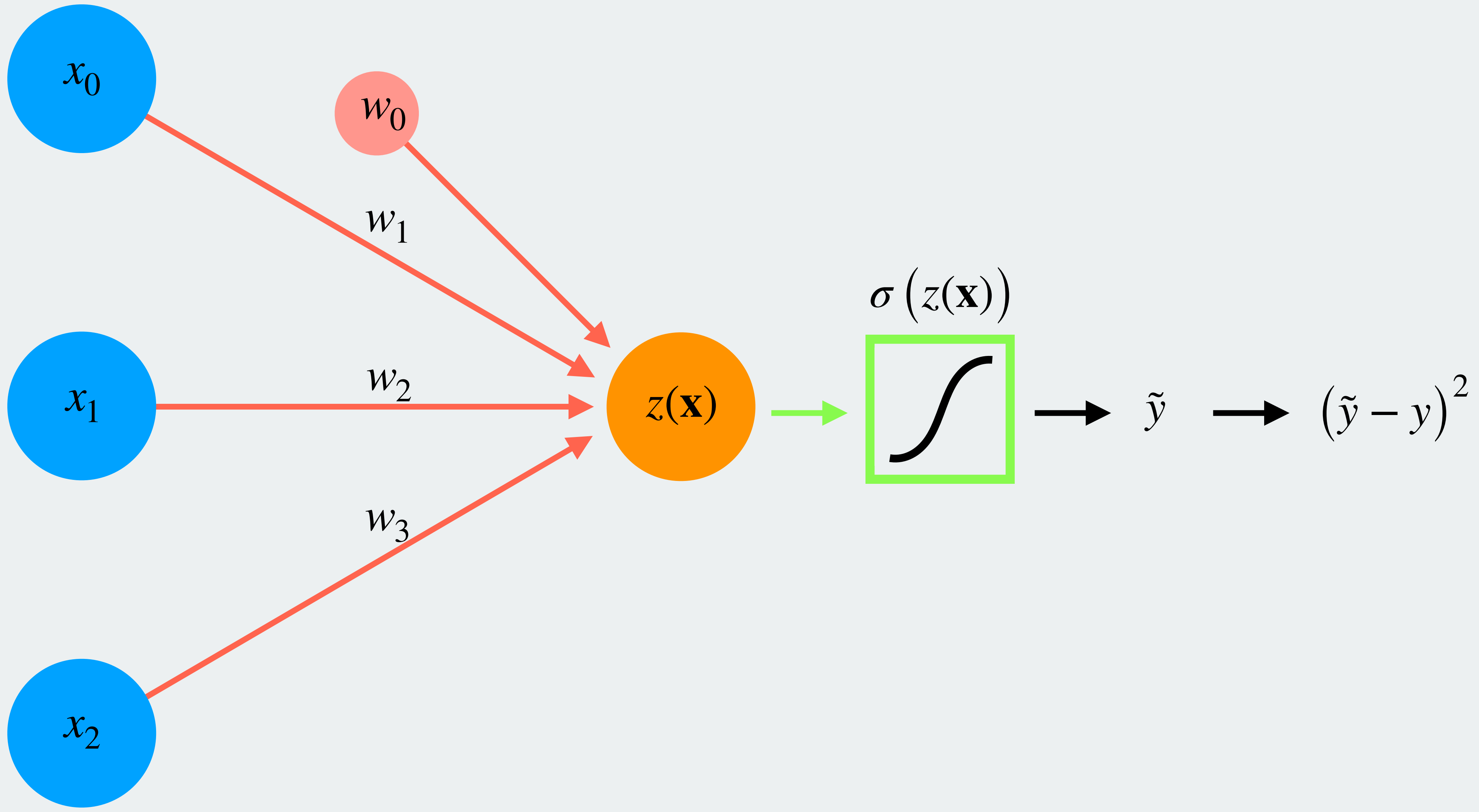
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

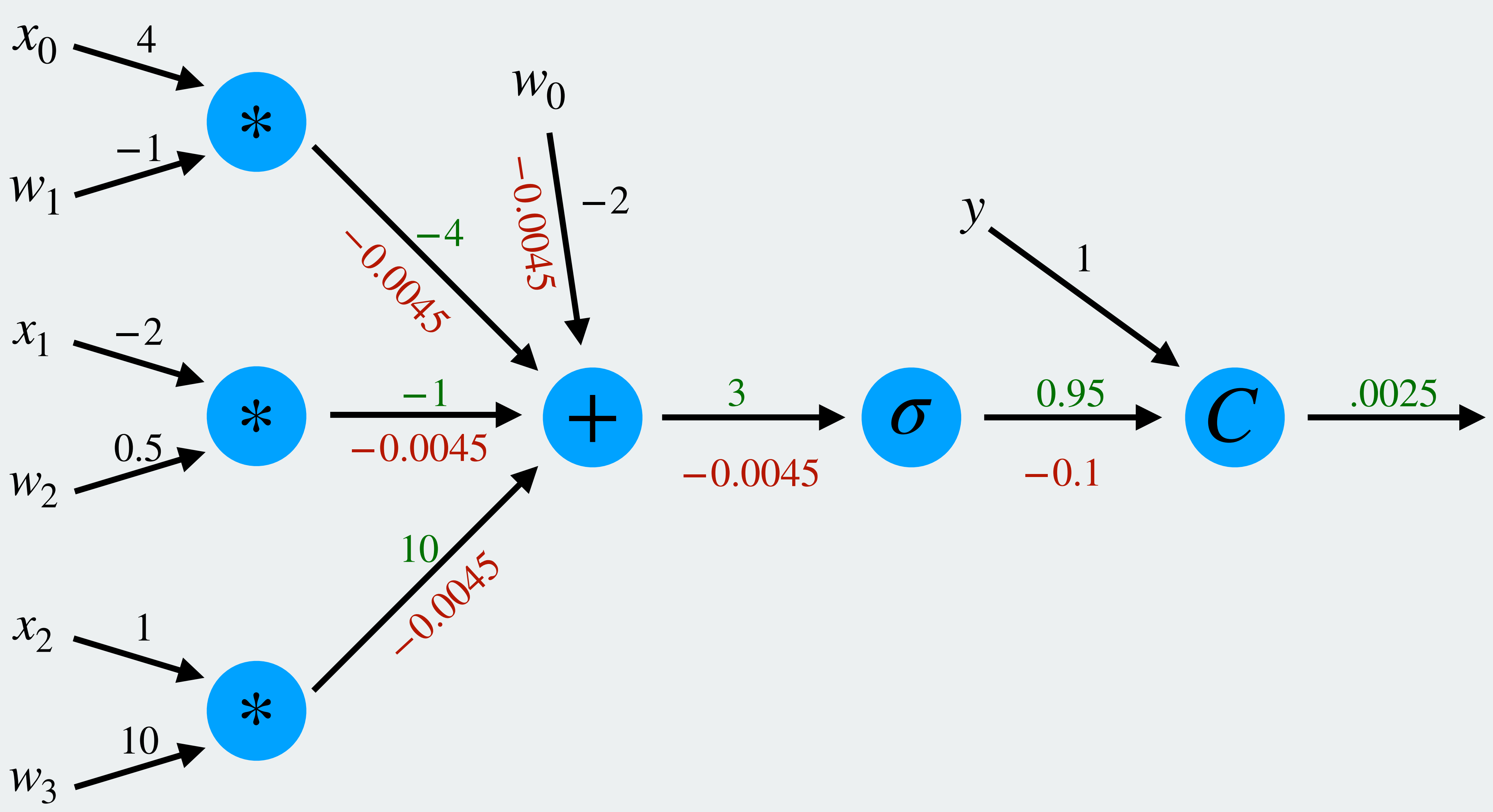
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is like a function: $y = \sum_{i=0}^{N} z_n$

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

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

Weights:

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

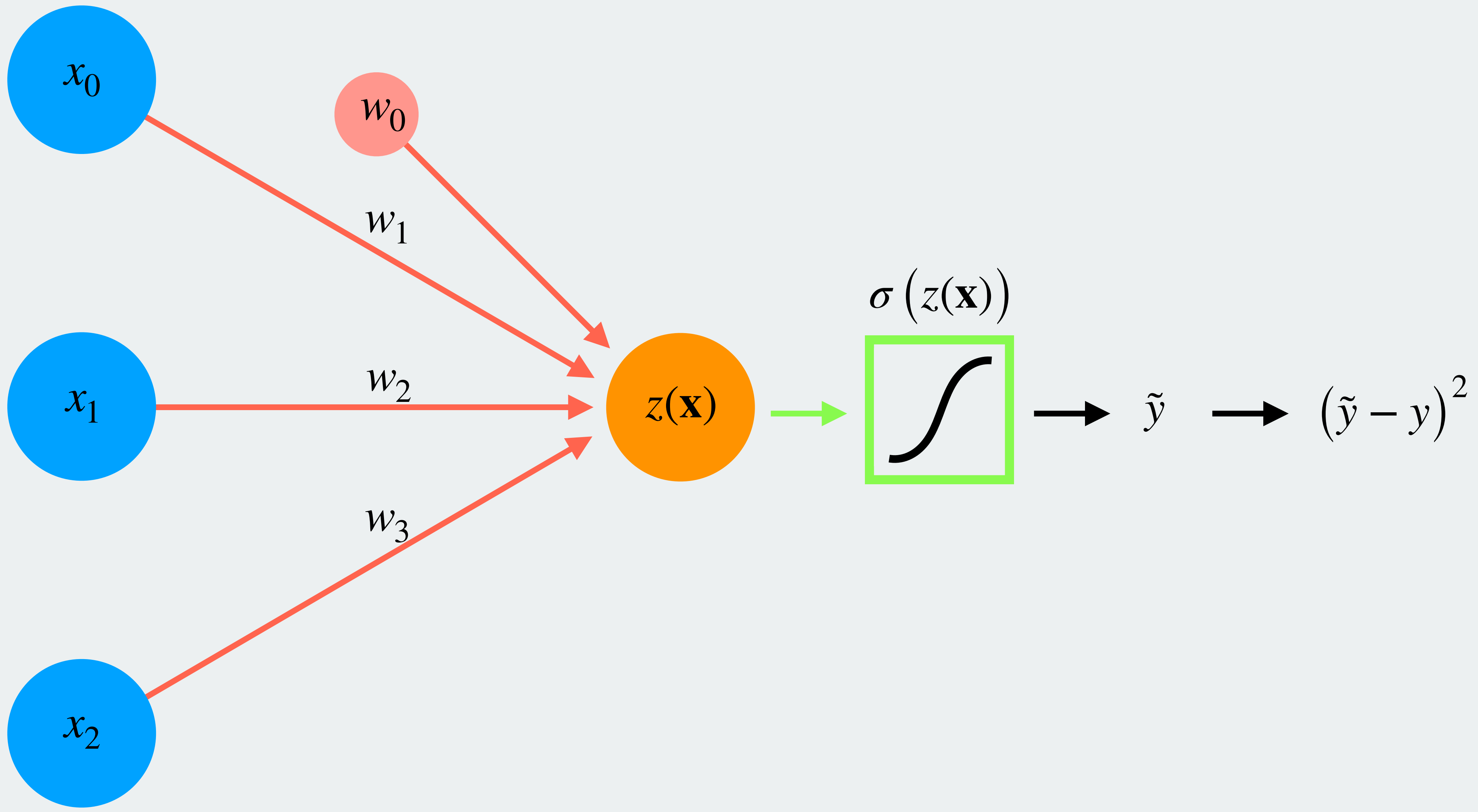
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

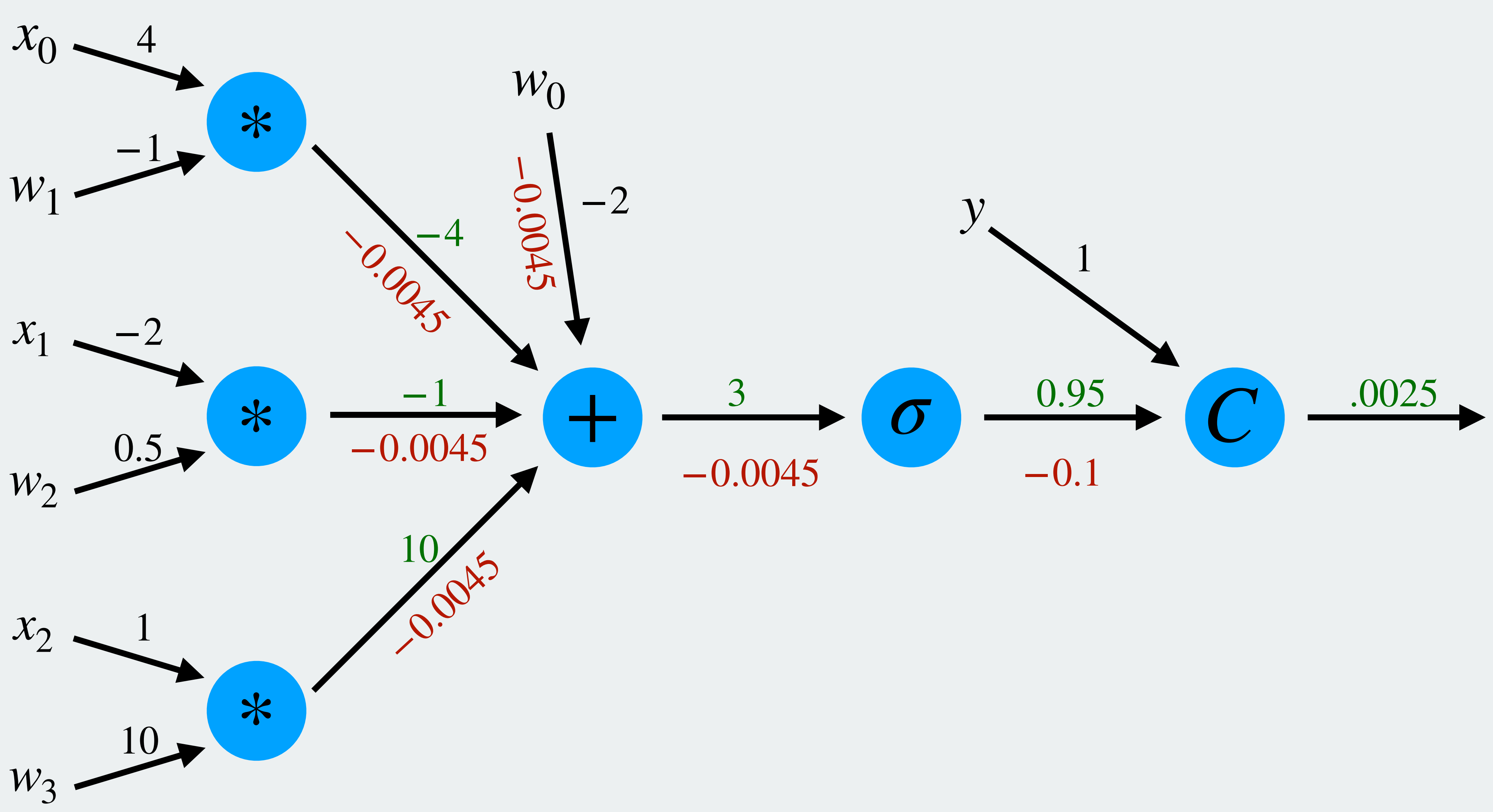
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

Each branch is a function: $y = wx$

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

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

Weights:

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

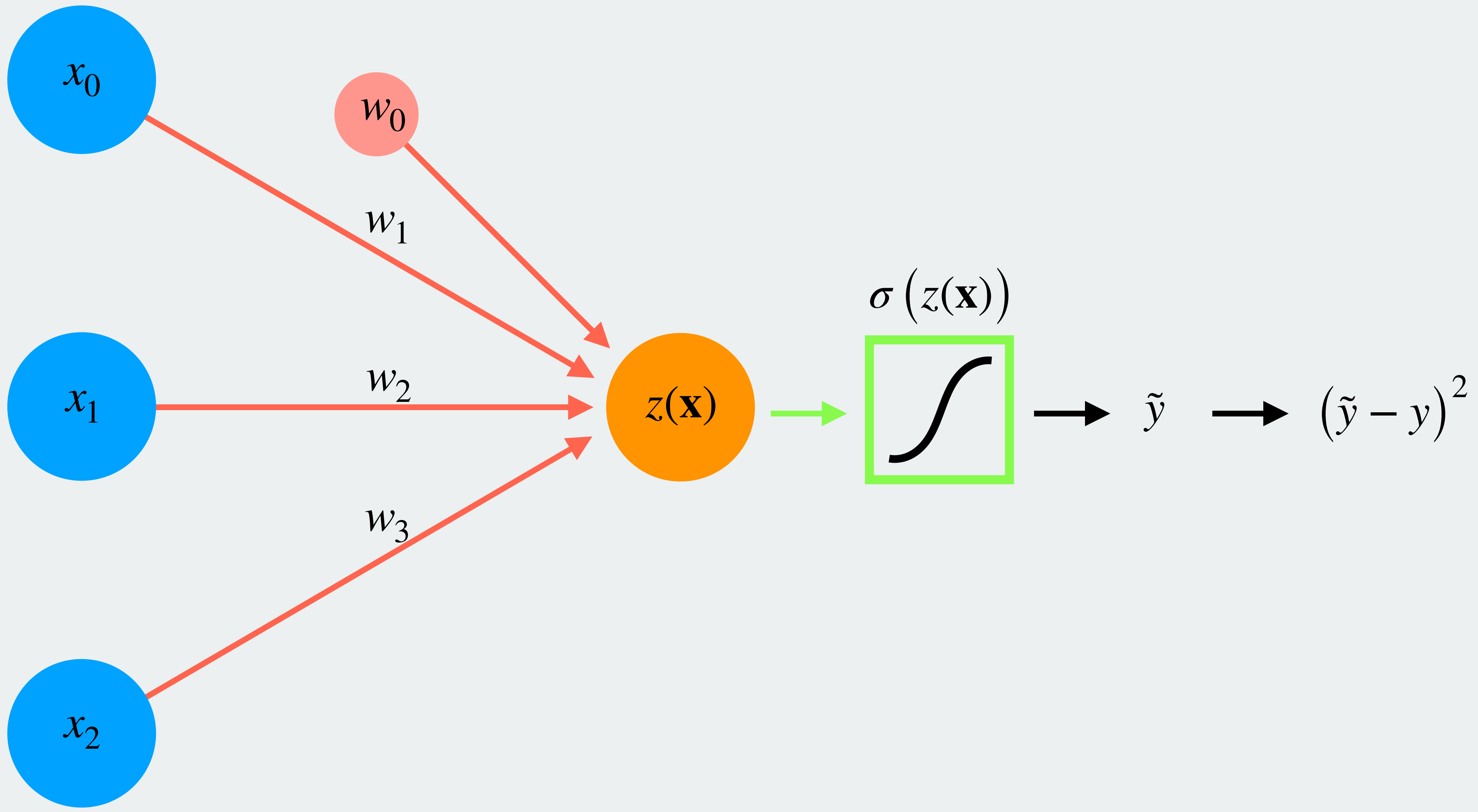
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

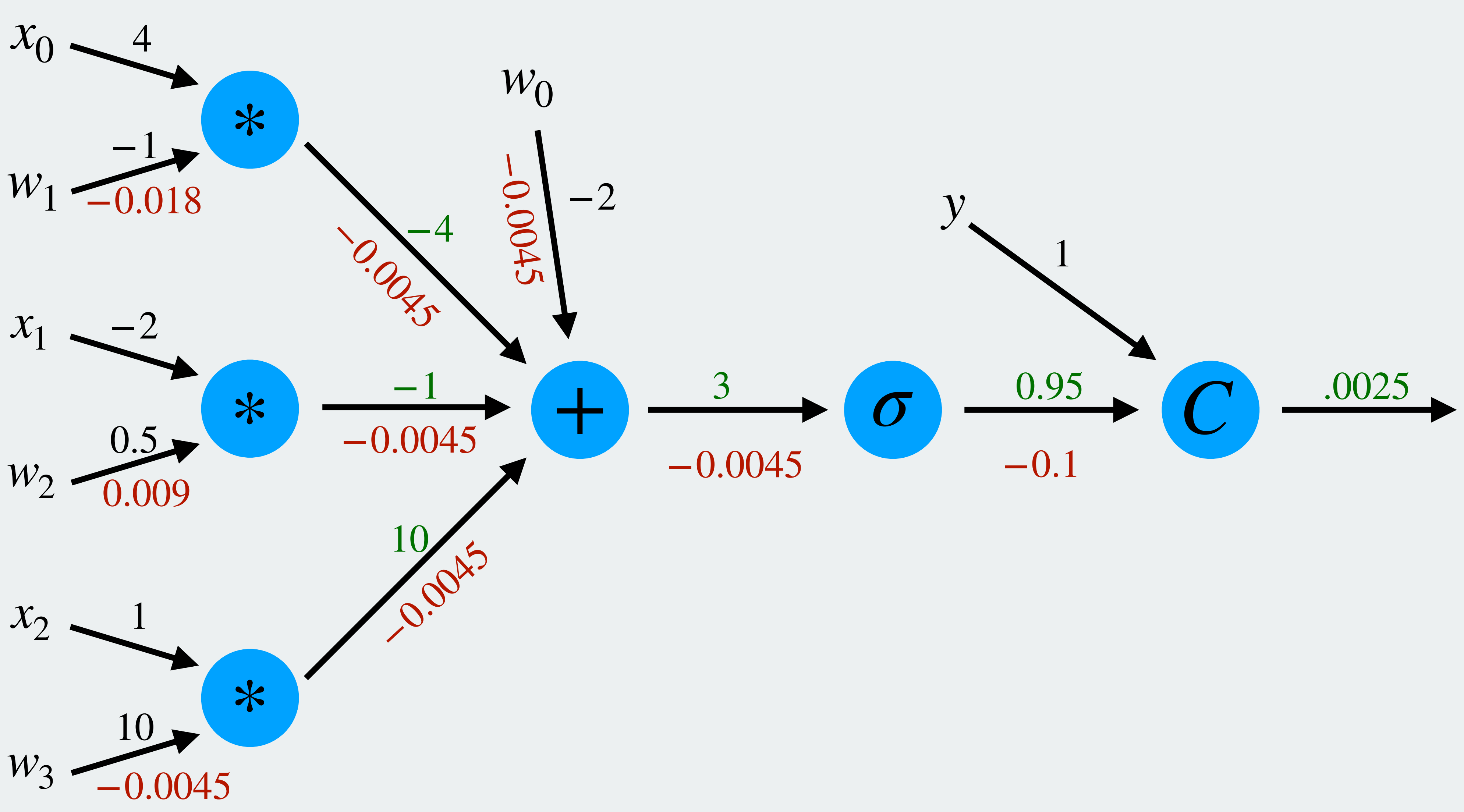
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



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:

Each branch is a function: $y = wx$

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

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

Weights:

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

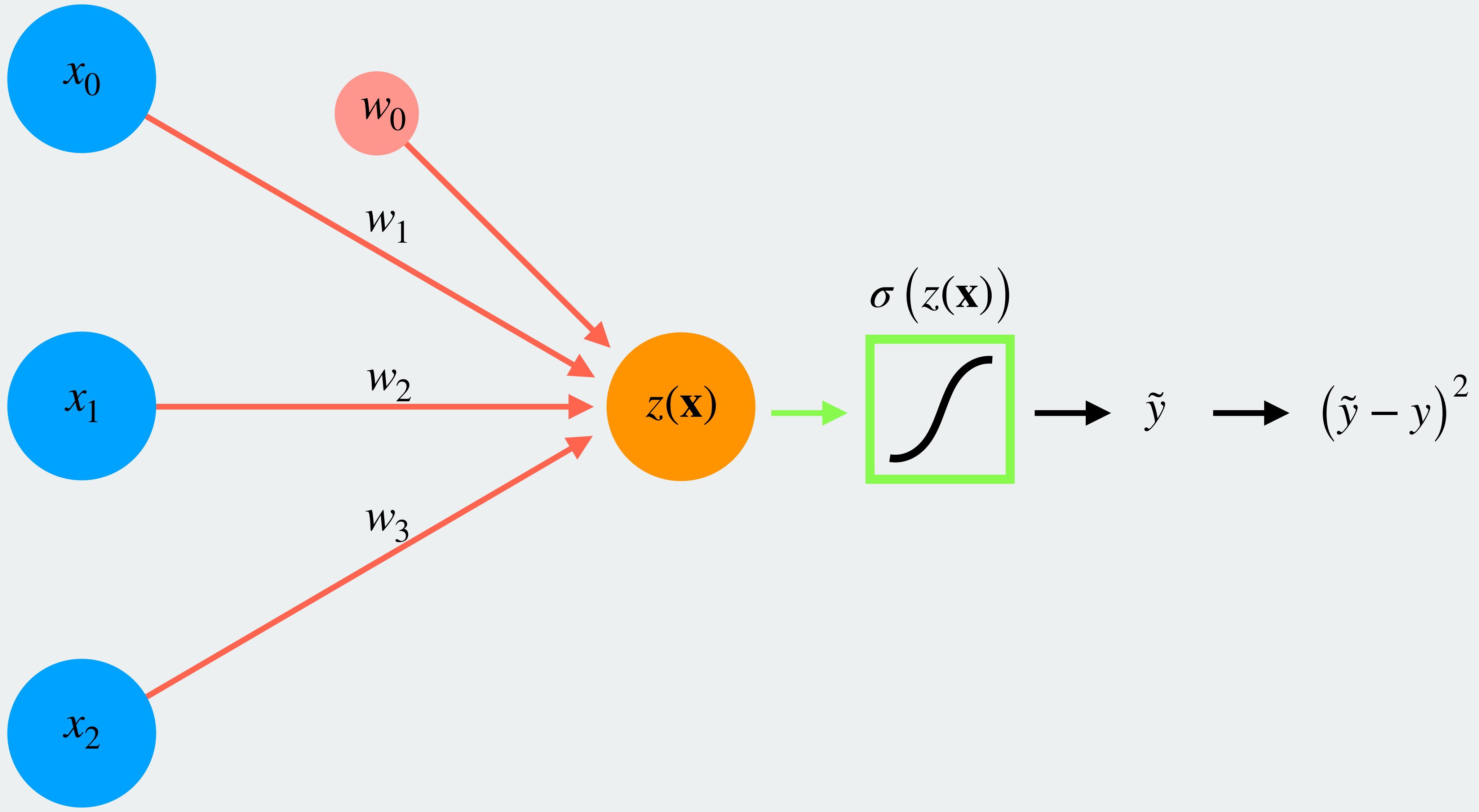
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

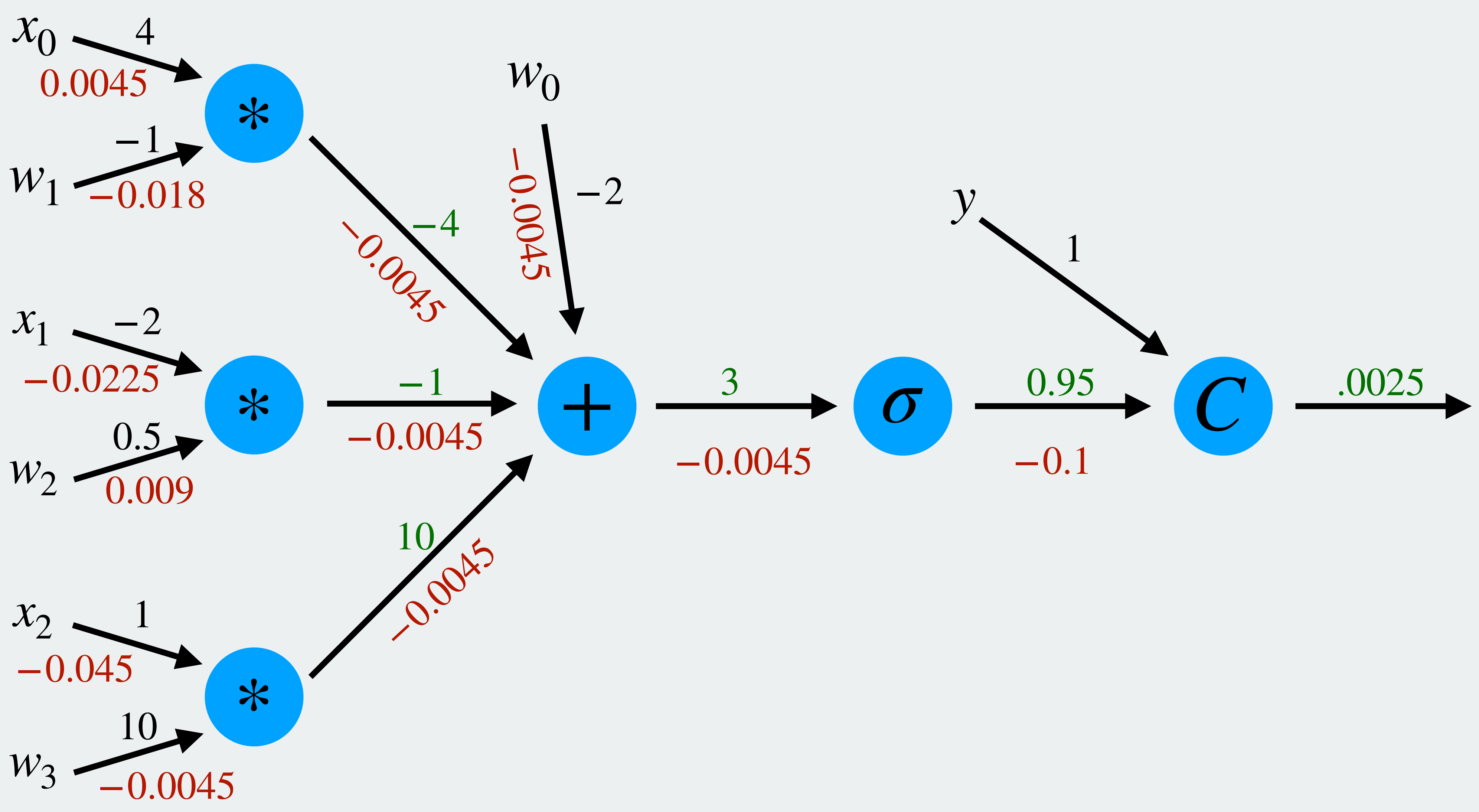
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



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:

Each branch is a function: $y = wx$

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

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

Weights:

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

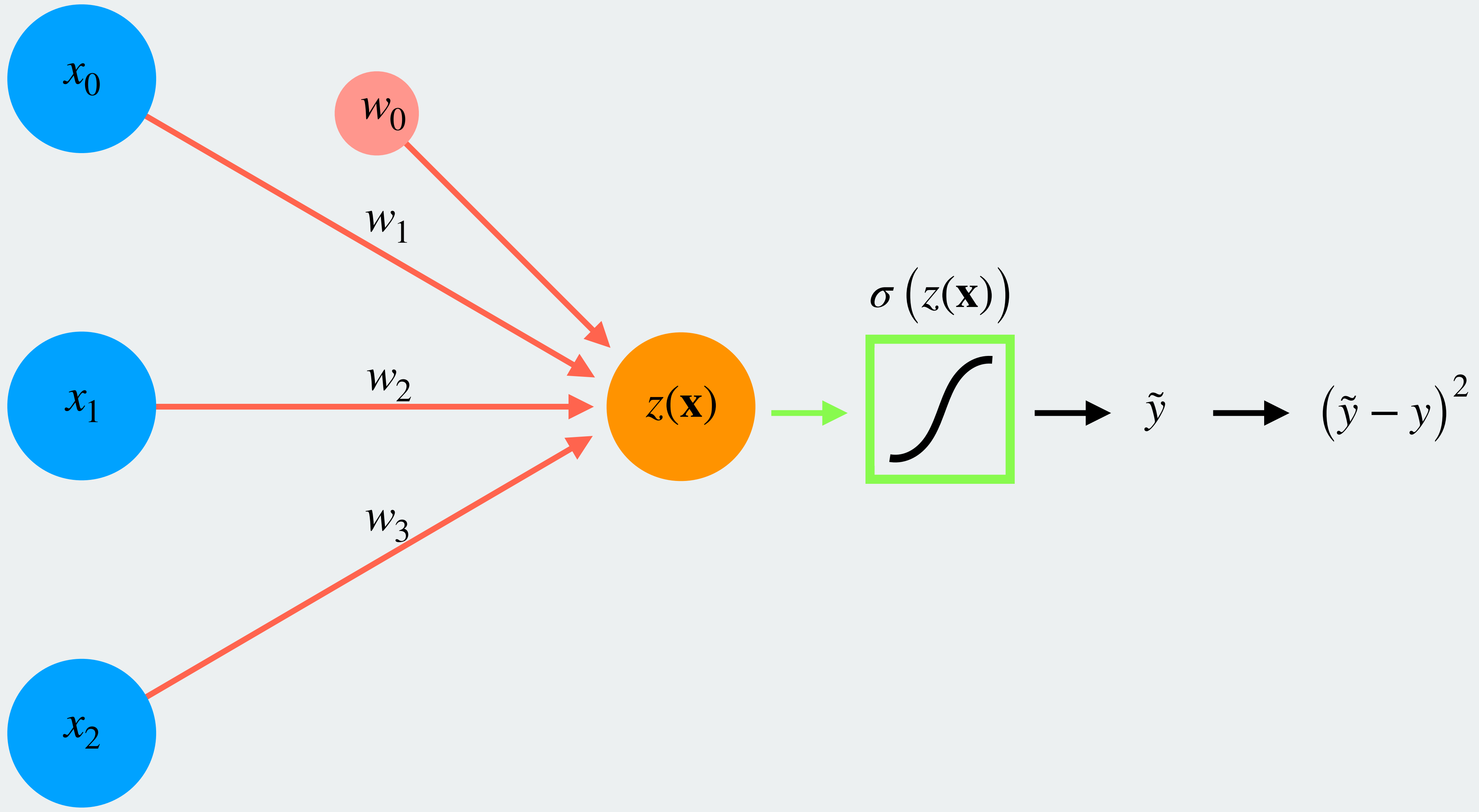
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

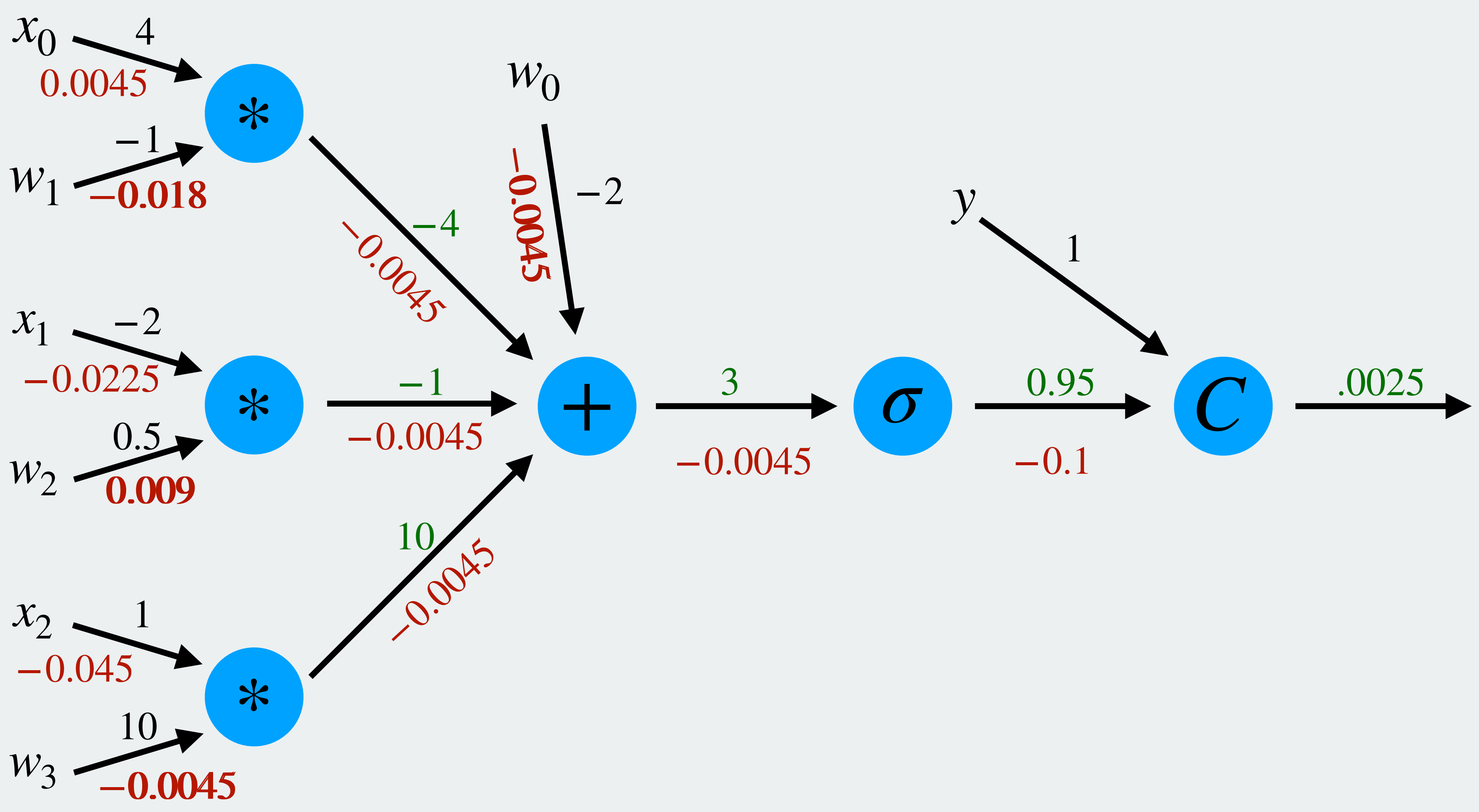
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

Each branch is a function: $y = wx$

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

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

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

	2	5	0	4	1	9	$\cdots$	Average over all training data
w_0	-0.08	$+0.02$	-0.02	$+0.11$	-0.05	-0.14	$\cdots$	$\rightarrow -0.08$
w_1	-0.11	$+0.11$	$+0.07$	$+0.02$	$+0.09$	$+0.05$	$\cdots$	
w_2	-0.07	-0.04	-0.01	$+0.02$	$+0.13$	-0.15	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
$w_{13,001}$	$+0.13$	$+0.08$	-0.06	-0.09	-0.02	$+0.04$	$\cdots$	

https://www.youtube.com/watch?v=Ilg3gGewQ5U

Regularization

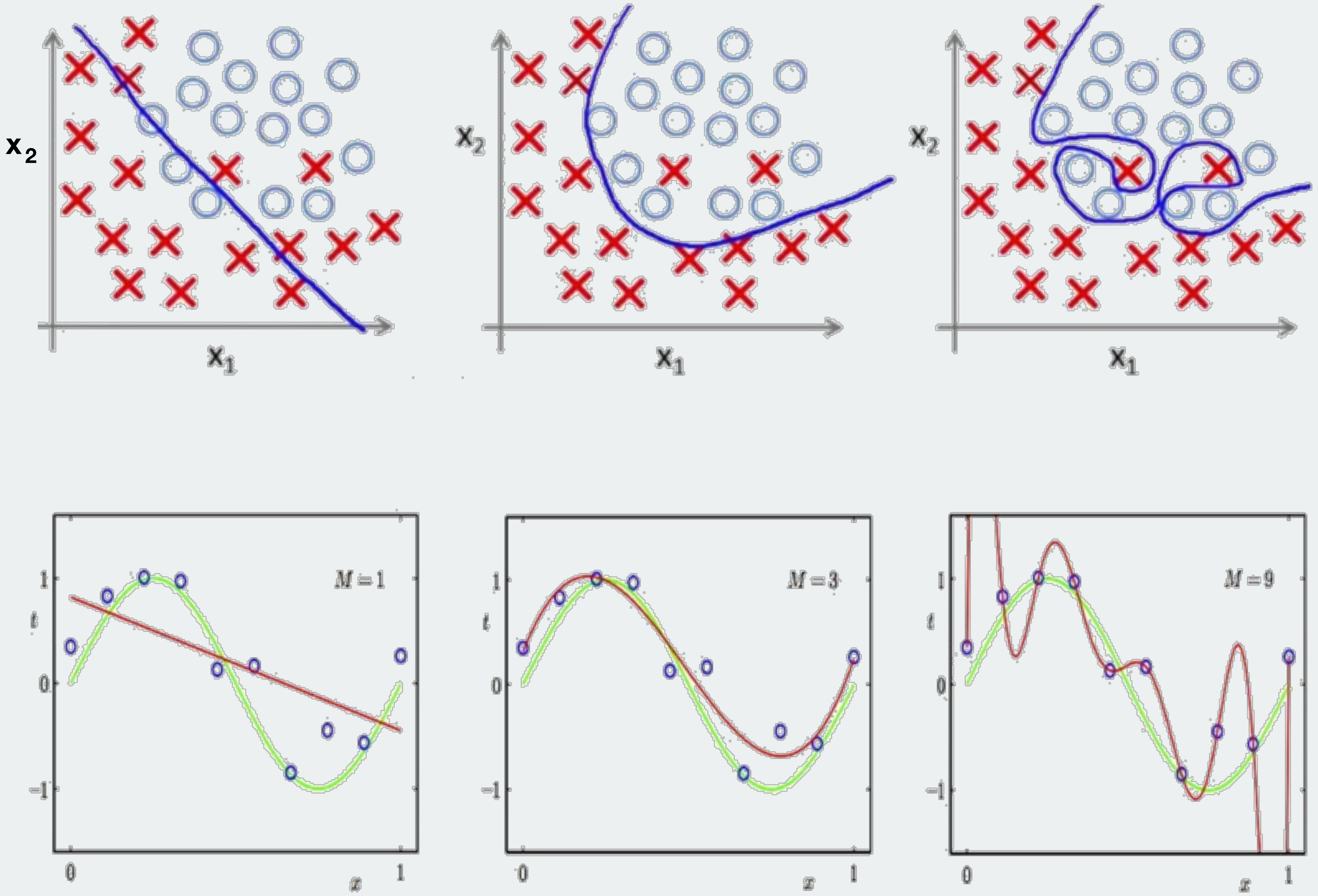
Tricks to avoid overfitting

Regularization – underfitting and overfitting

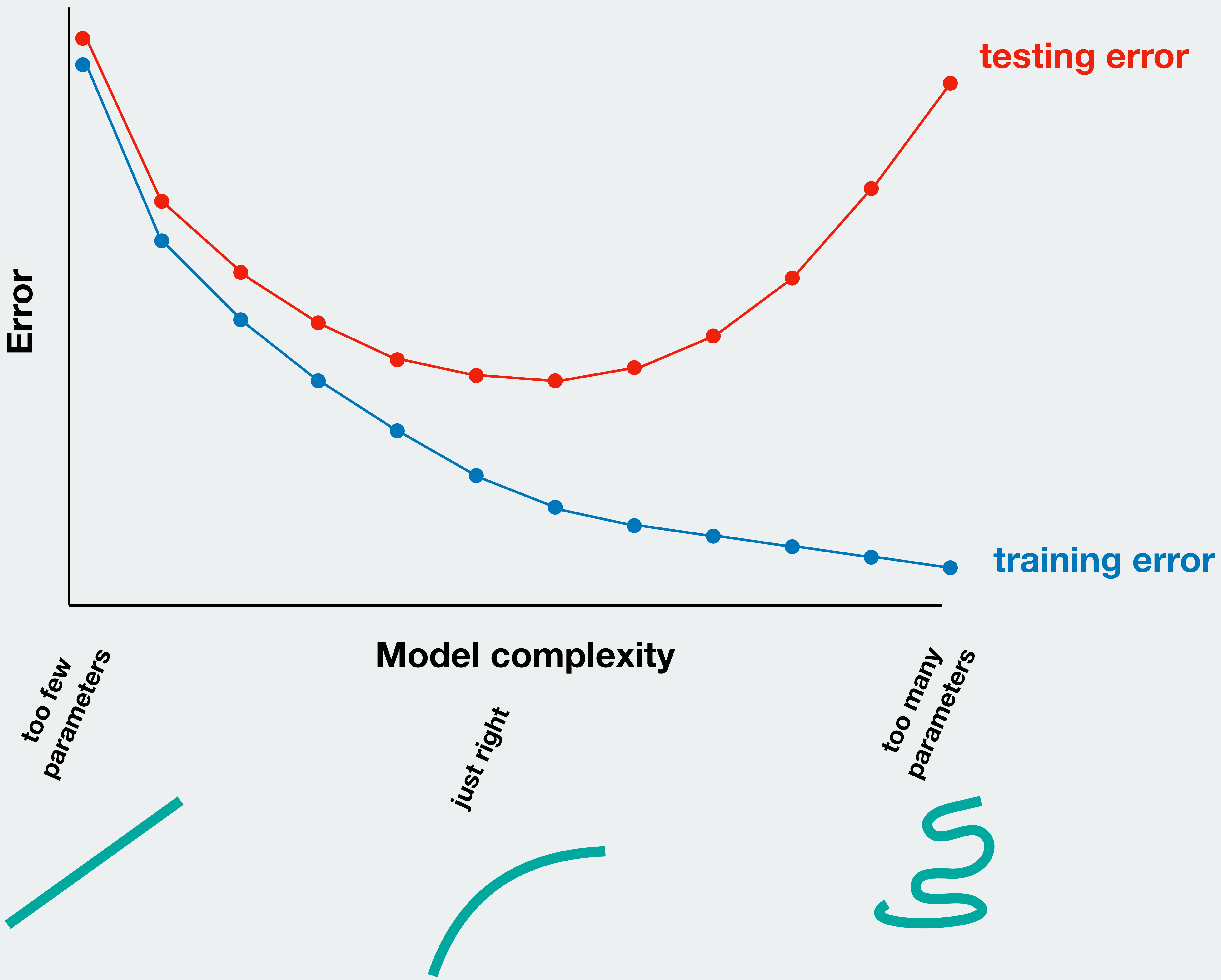
Underfitting **Overfitting**

Classification

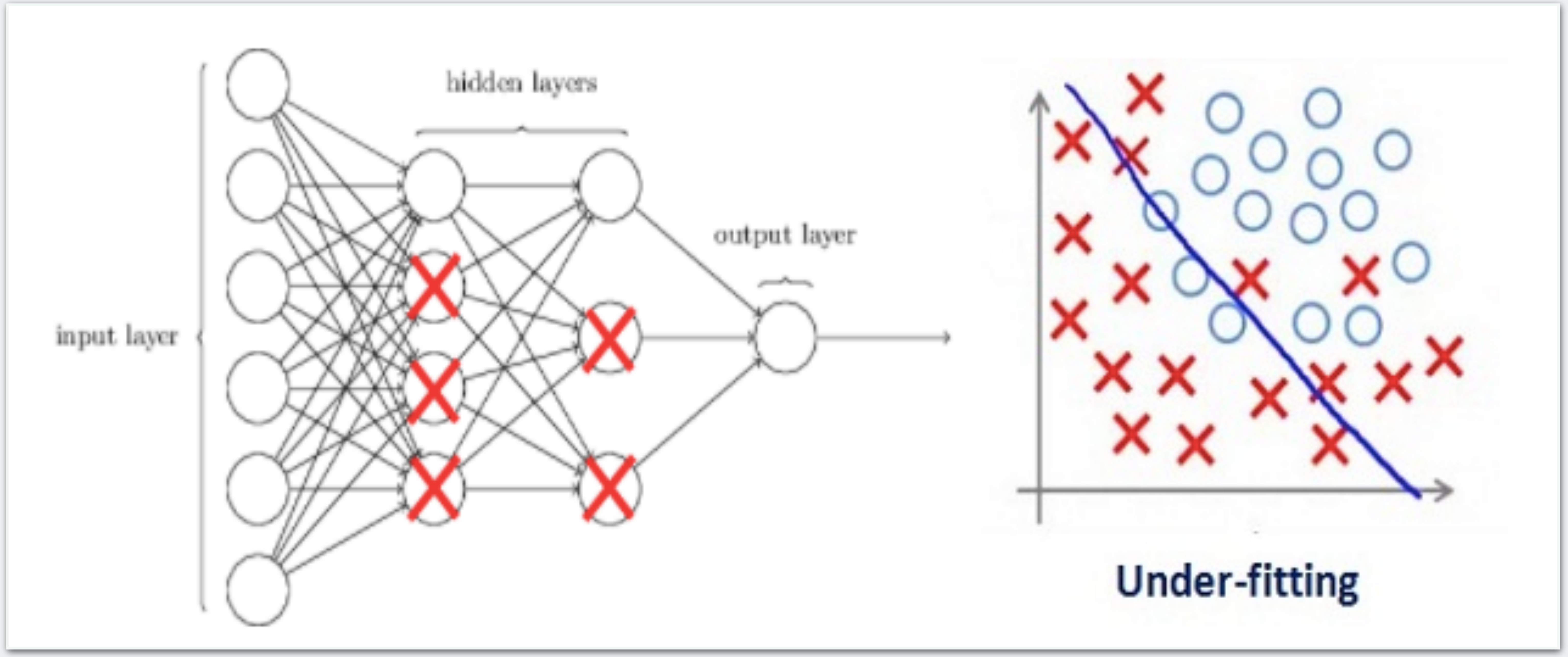
Regression



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

<u>L-norm regularization:</u> "Introduce a cost for large weights"

$$C = Loss + Regularization\ term$$

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

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

Regularization – Different techniques

L-norm regularization: "Introduce a cost for large weights"

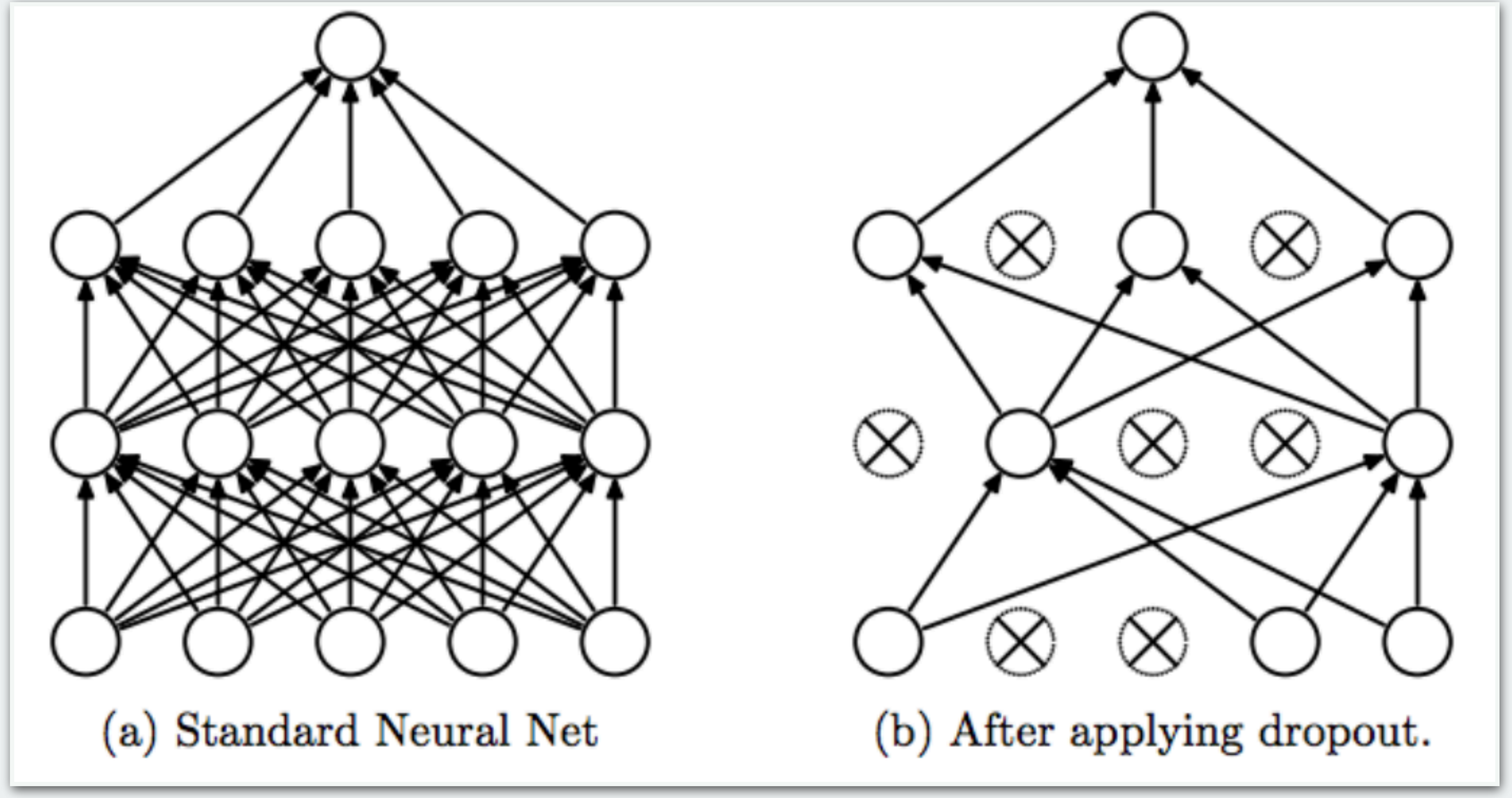
$$C = Loss + Regularization\ term$$

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

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

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"

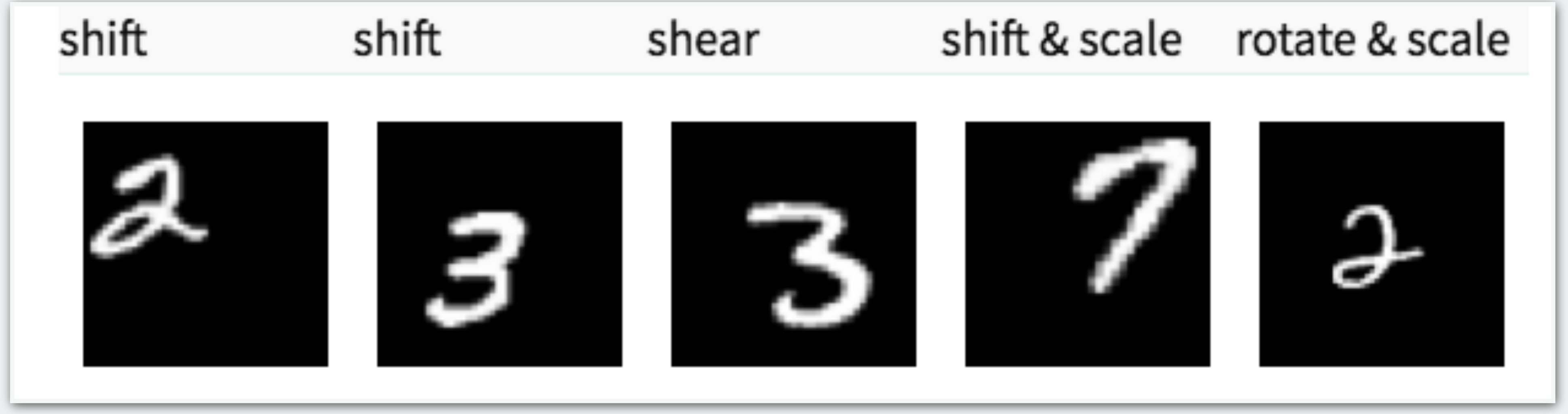
$$C = Loss + Regularization\ term$$

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

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

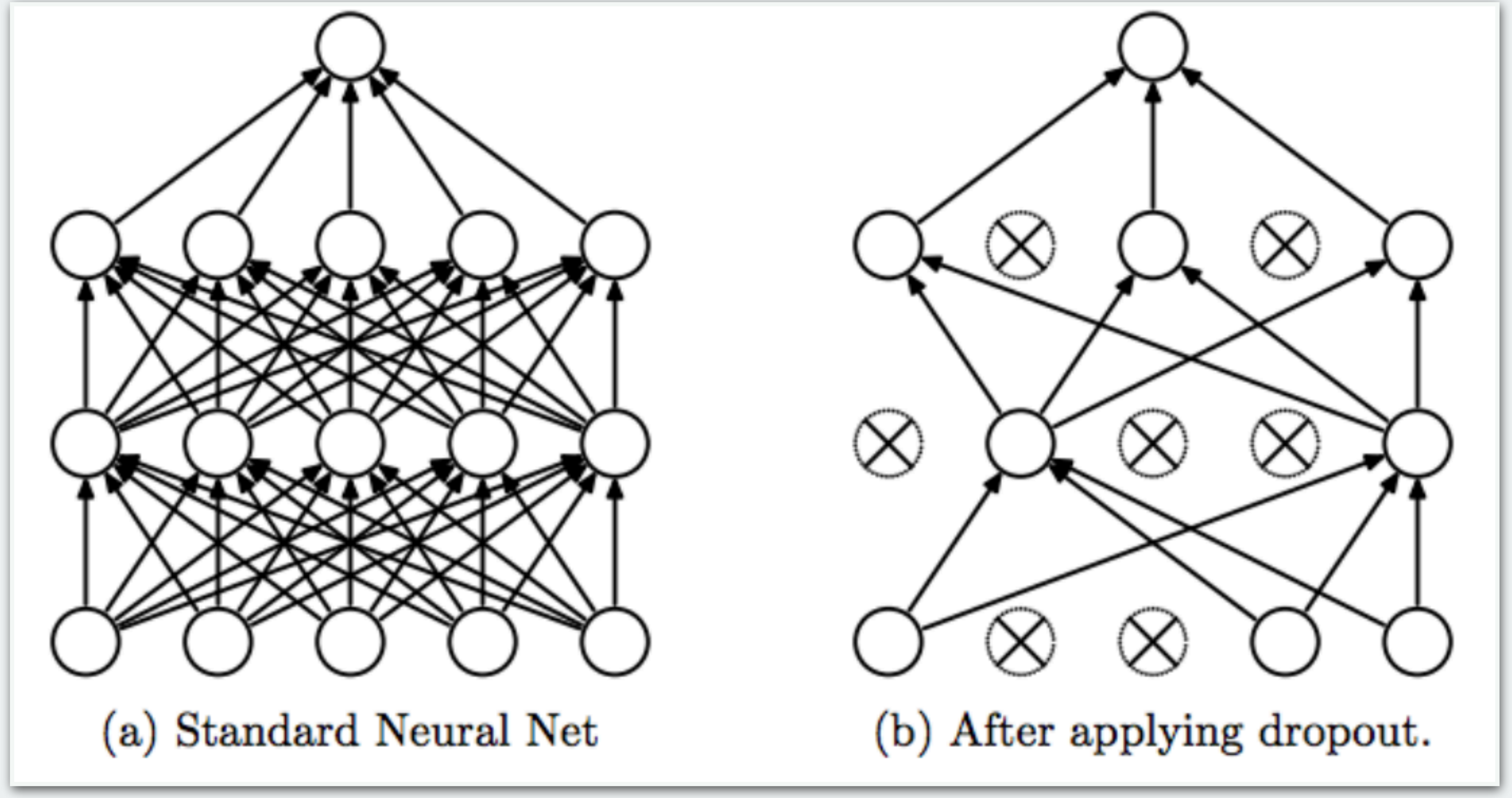
Data augmentation

"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"

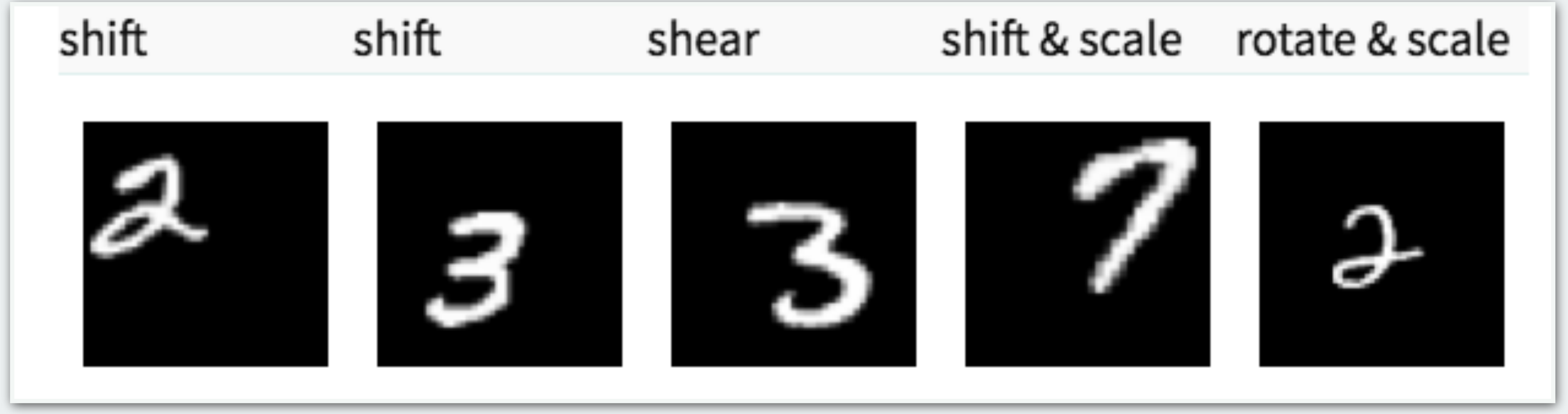
$$C = Loss + Regularization\ term$$

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

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

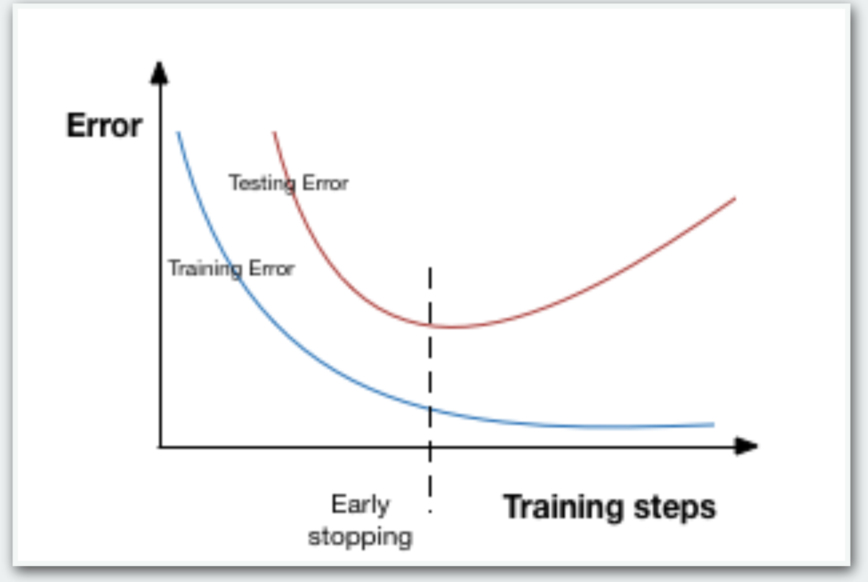
Data augmentation

"Shear, shift, scale and/or rotate input data"



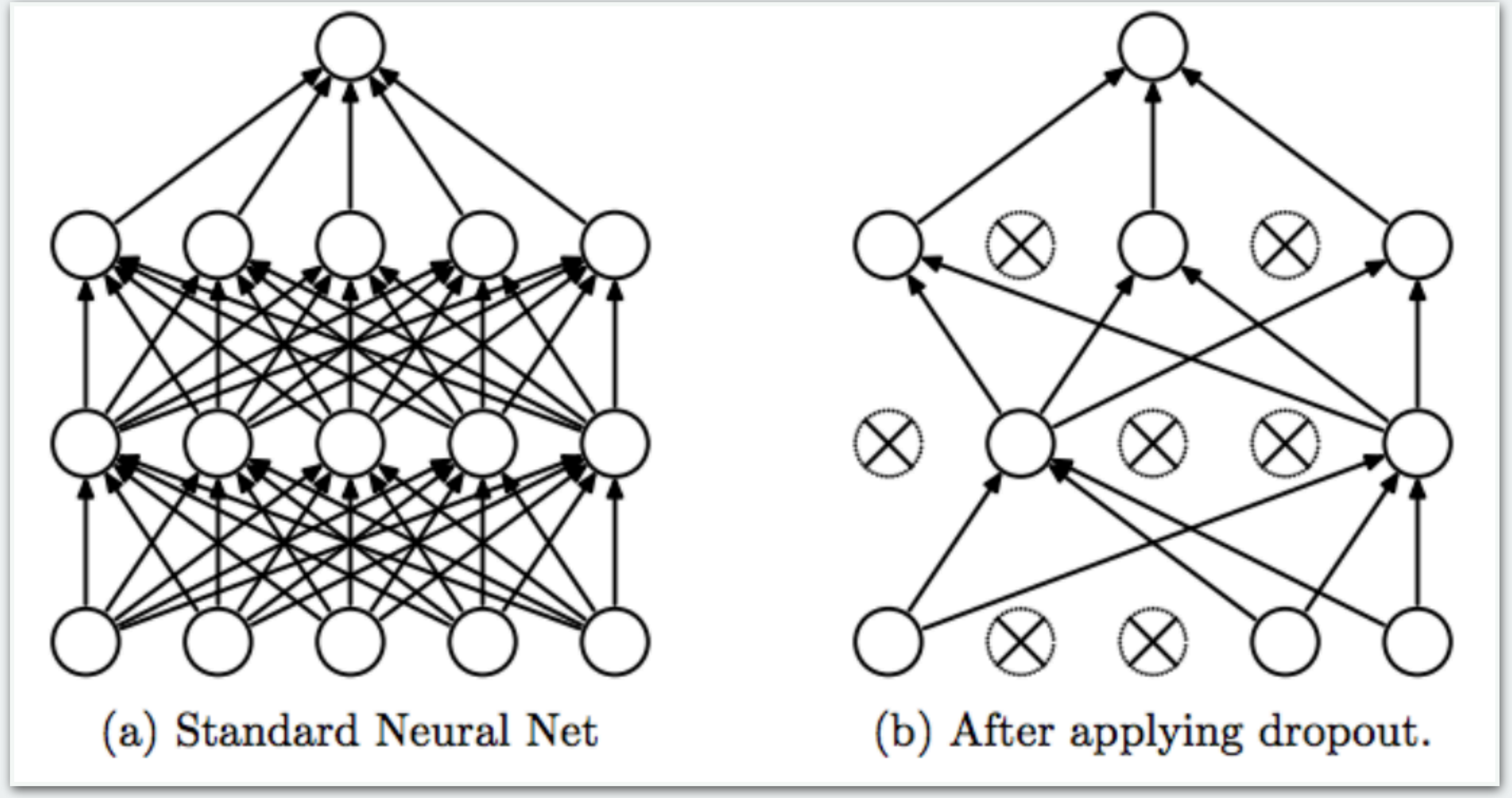
Early stopping

"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 notes.
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

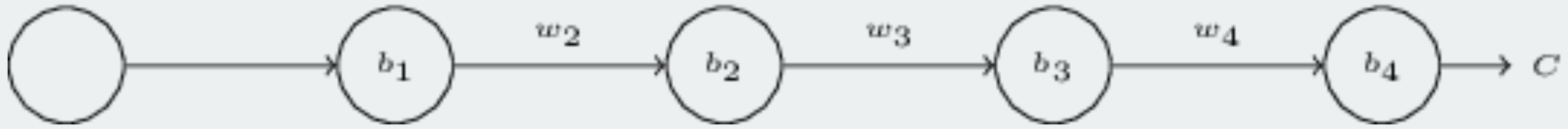
$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

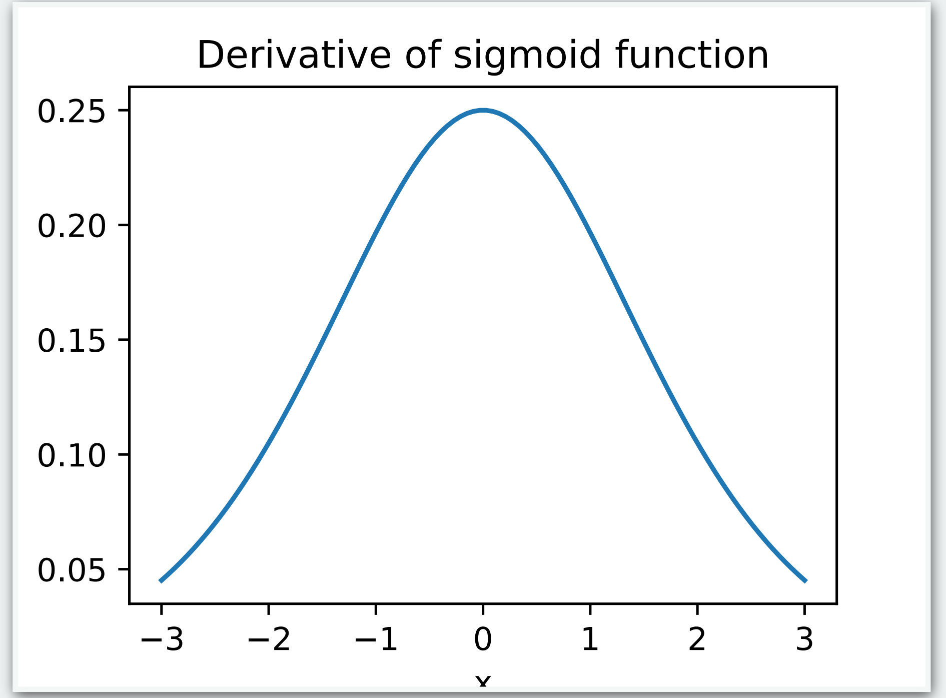
Vanishing gradients – A problem in *deep* neural nets

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

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



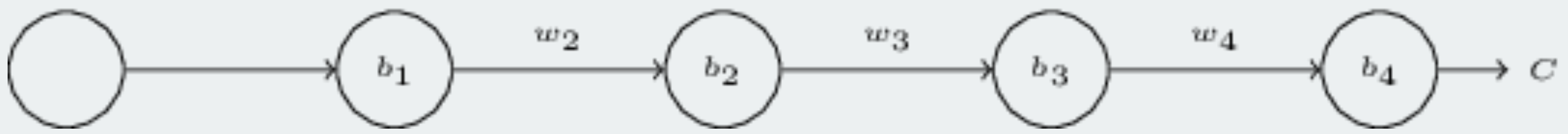


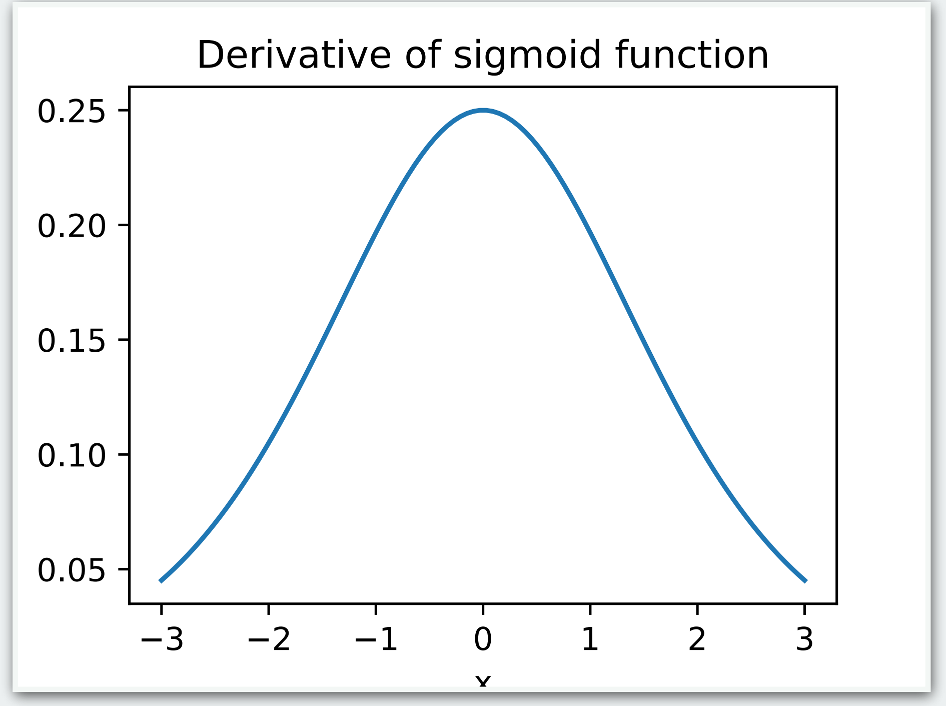
Vanishing gradients – A problem in *deep* neural nets

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

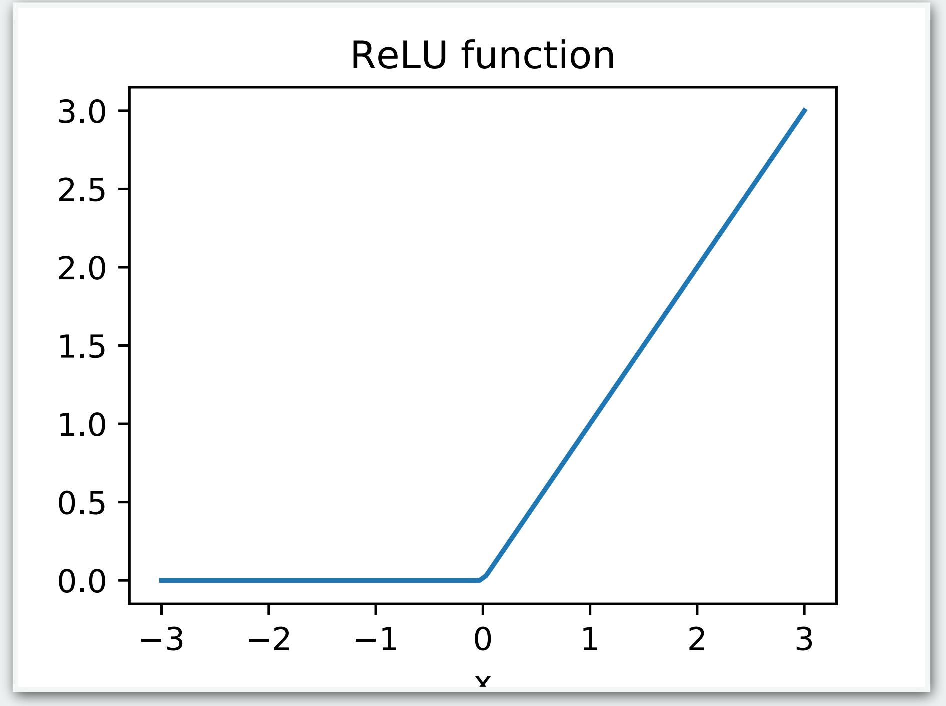
$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

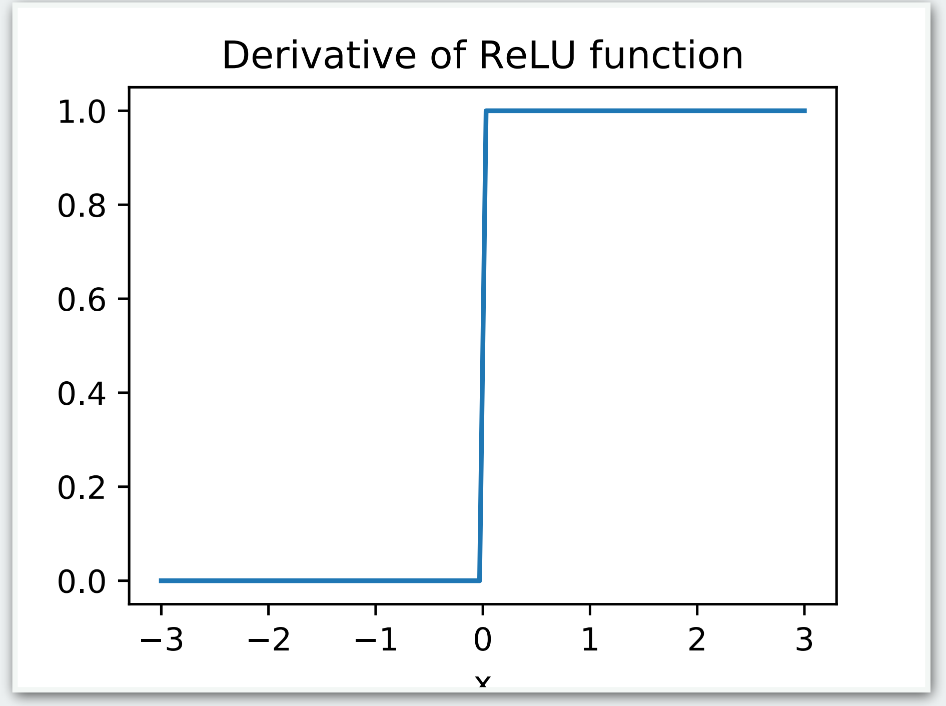




Solution:

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



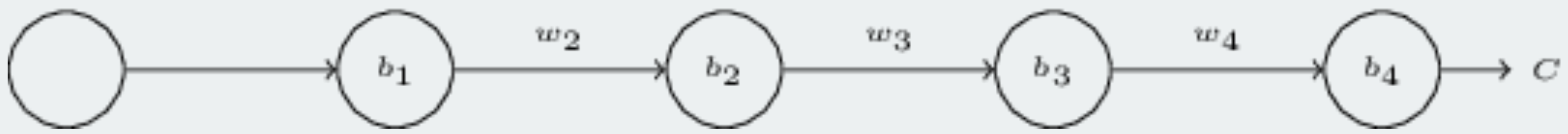


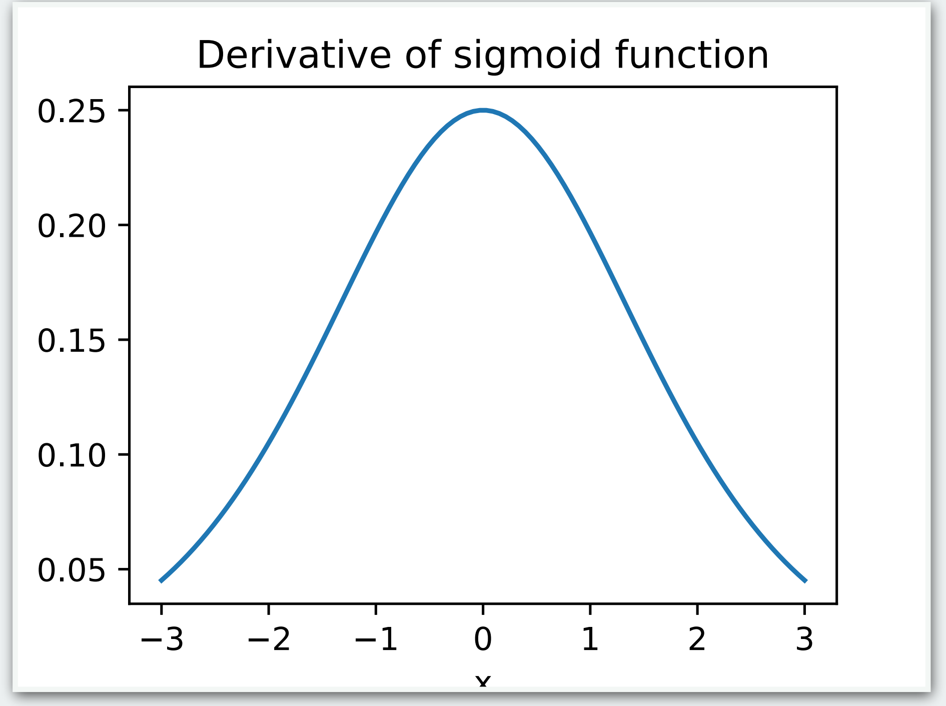
Vanishing gradients – A problem in *deep* neural nets

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

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





Solution:

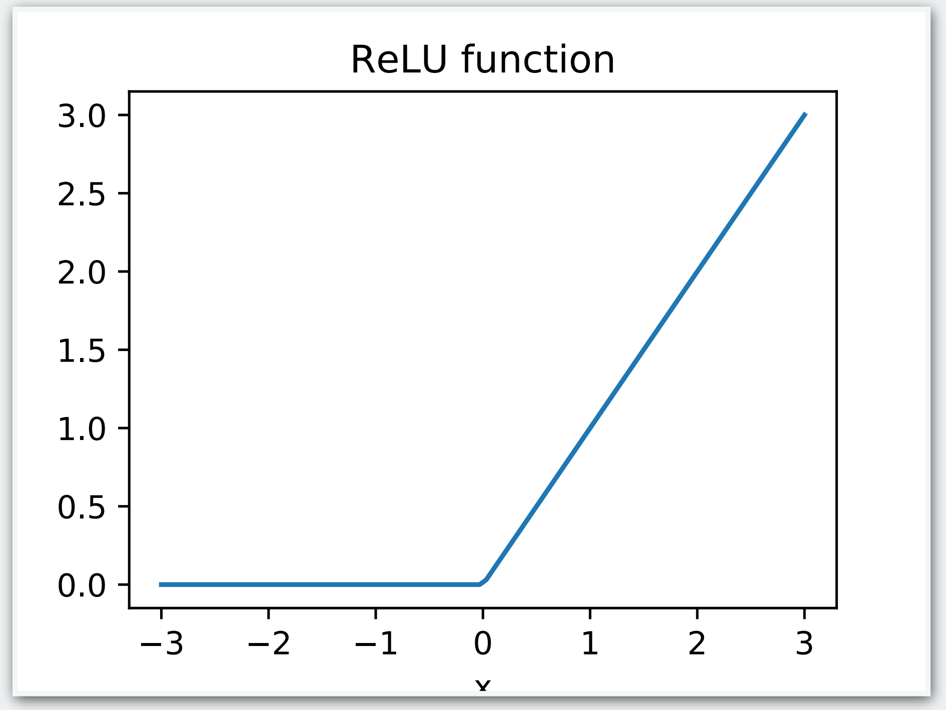
- Use an activation function without small gradient for high values
- Candidate activation function: ReLU

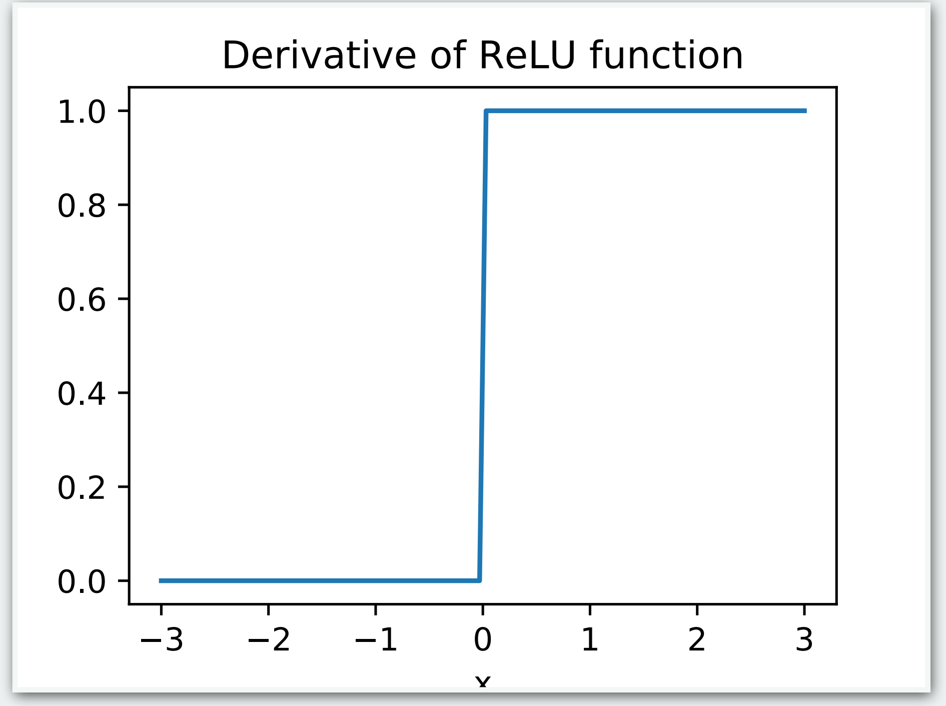
Problems with ReLU:

- Exploding gradients!

Solution:

- Batch normalization, gradient clipping, weight regularization



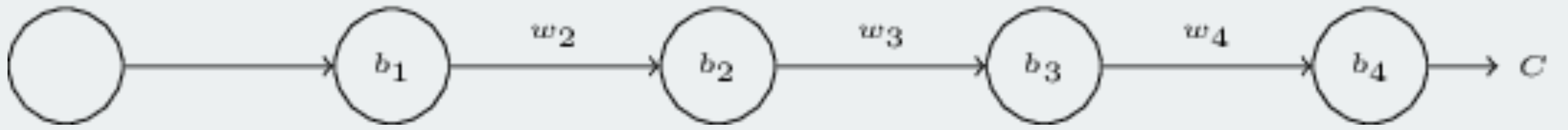


Vanishing gradients – A problem in *deep* neural nets

Problem:

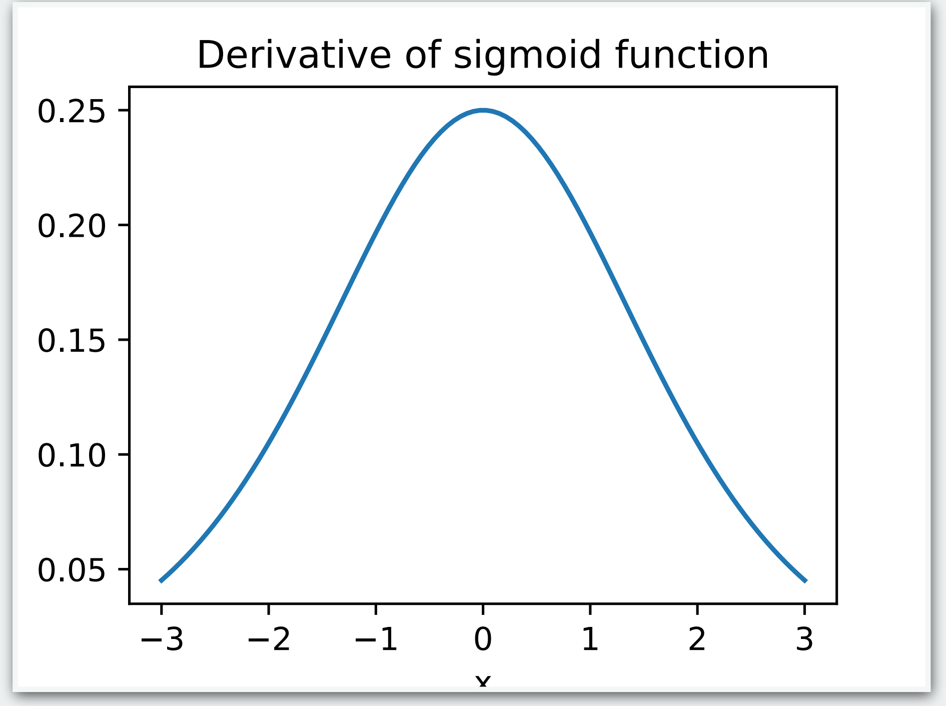
- Gradients closer and closer to the input tend to get smaller and smaller
- Leads to smaller weight updates near input and larger weight updates near output
- Bad because layers near input take part in recognizing “simple” patterns, which are important to learning

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



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