Artificial Neural Networks and Deep Learning

Week 2

Gradient descent and backpropagation

Recap

Logistic regression classifier

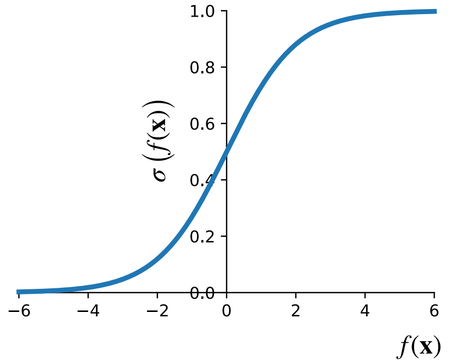
(Input)

Variables **Weights**

$$w_0 + x_0 w_1 + x_1 w_2 = f(\mathbf{x})$$

Output $\sigma\left(f(\mathbf{x})\right) = \dfrac{1}{1 + exp\left(-f(x)\right)}$

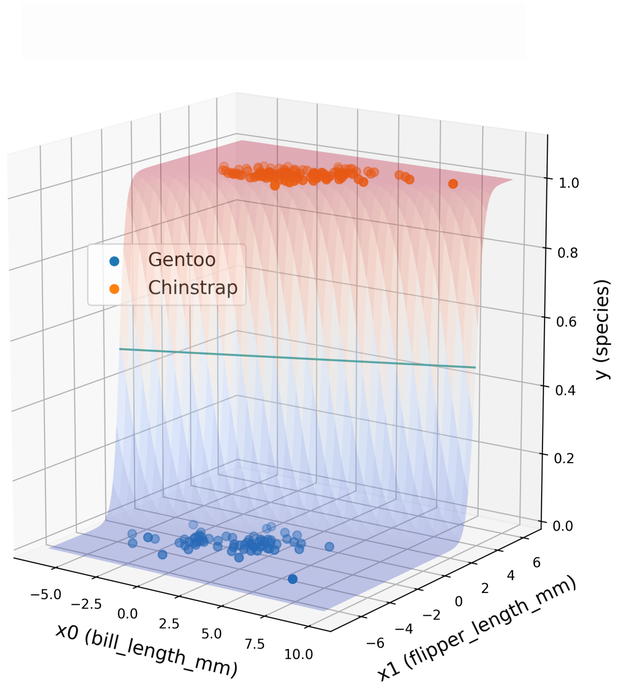
Activation function

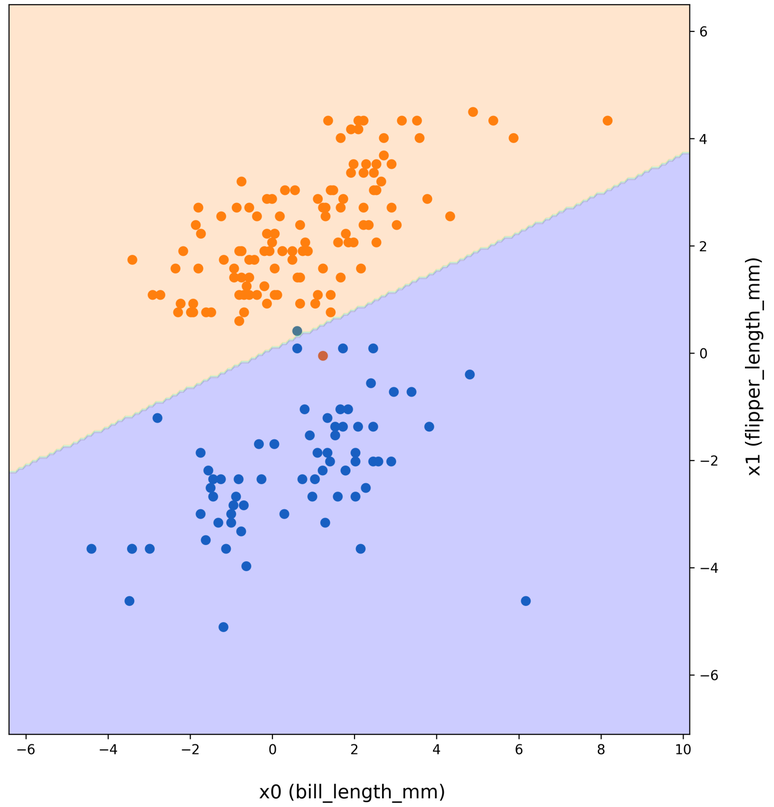


Logistic regression classifier

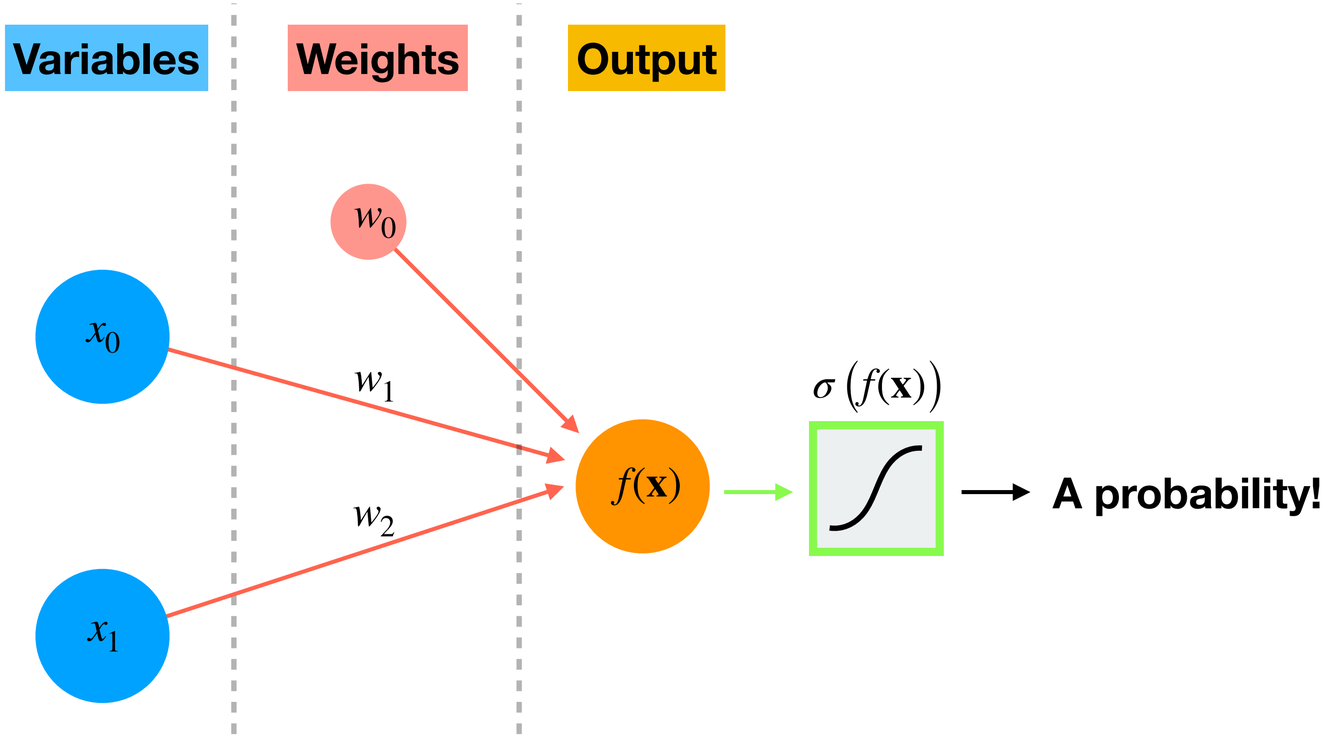
- We want to find the weights that best separates the data

$$\hat{y} = \sigma\left(w_0 + x_0 w_1 + x_1 w_2\right)$$



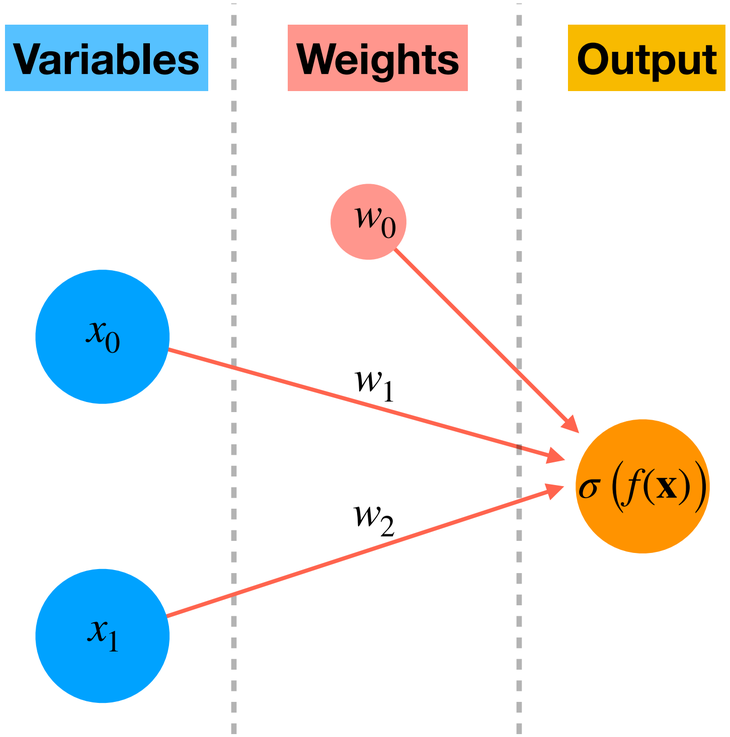


Logistic regression schematic illustration



$$\sigma(w_0 + x_0 w_1 + x_1 w_2)$$

Logistic regression schematic illustration

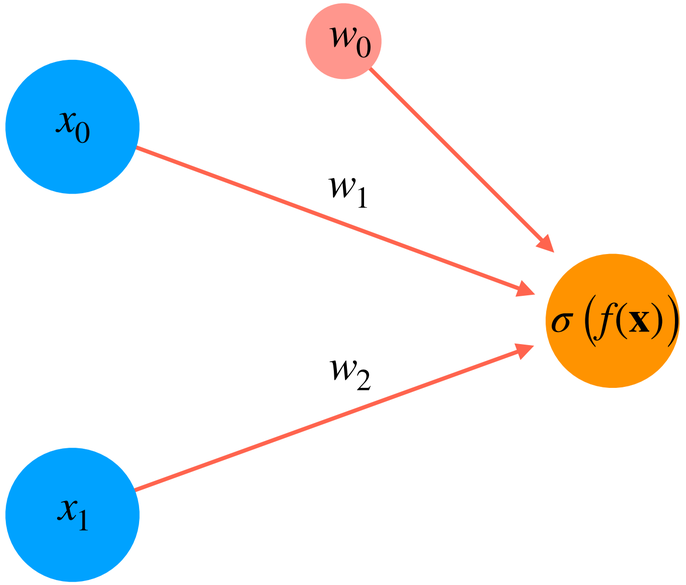


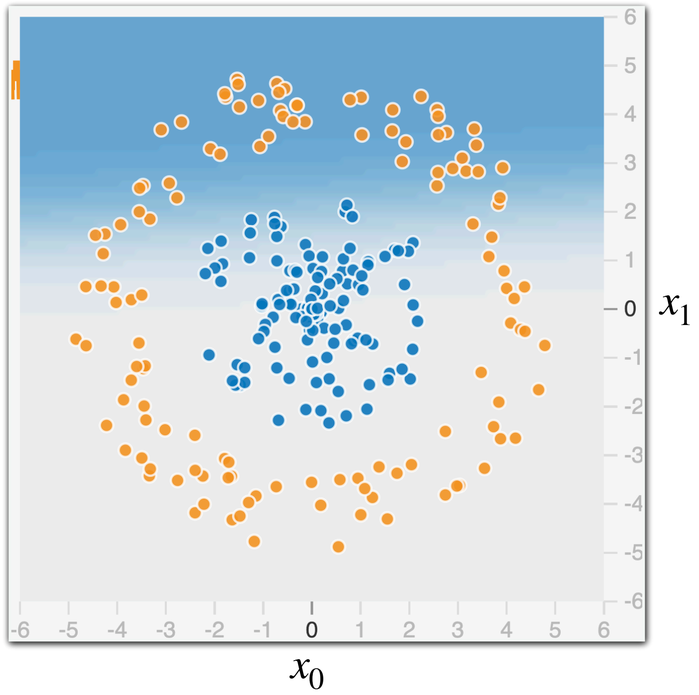
$$\sigma(w_0 + x_0 w_1 + x_1 w_2)$$

Logistic regression not so simple problem

> Find values of $\{w_0, w_1, w_2\}$ that minimizes $\sum_n (\tilde{y}_n - y_n)^2$

Model too simple

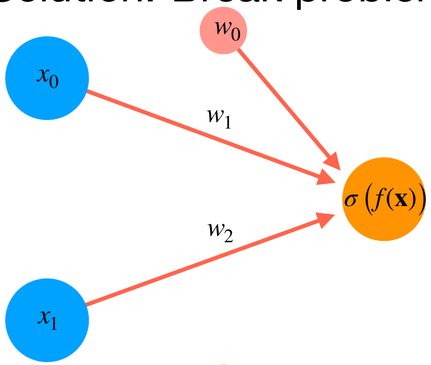




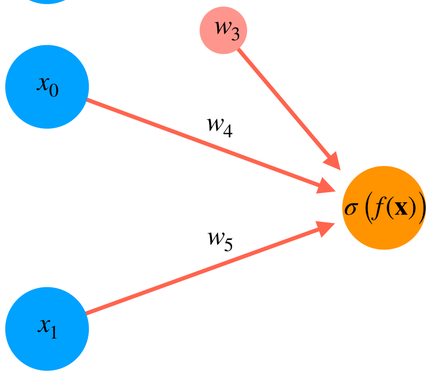
Reminder: $w_0 + x_0 w_1 + x_1 w_2 = f(\mathbf{x})$

Logistic regression not so simple problem

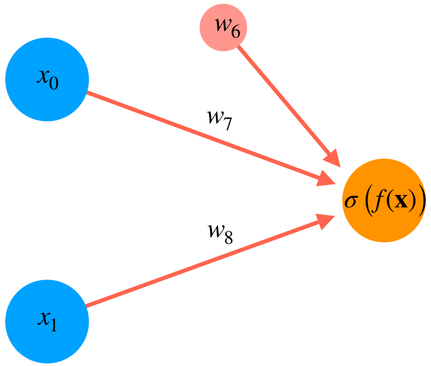
> *Solution:* Break problem into subproblems



$$\sigma(w_0 + x_0 w_1 + x_1 w_2) = a_0(\mathbf{x})$$



$$\sigma(w_3 + x_0 w_4 + x_1 w_5) = a_1(\mathbf{x})$$

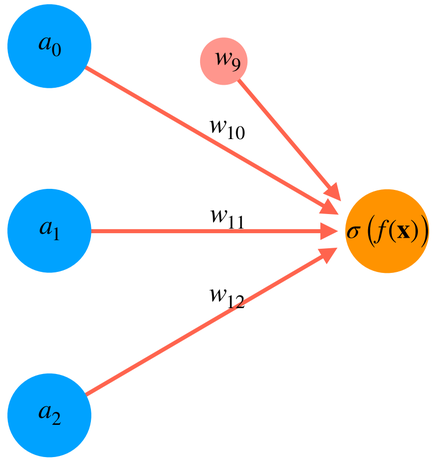


$$\sigma(w_6 + x_0 w_7 + x_1 w_8) = a_2(\mathbf{x})$$

New problem: Given **Z**, predict **y**

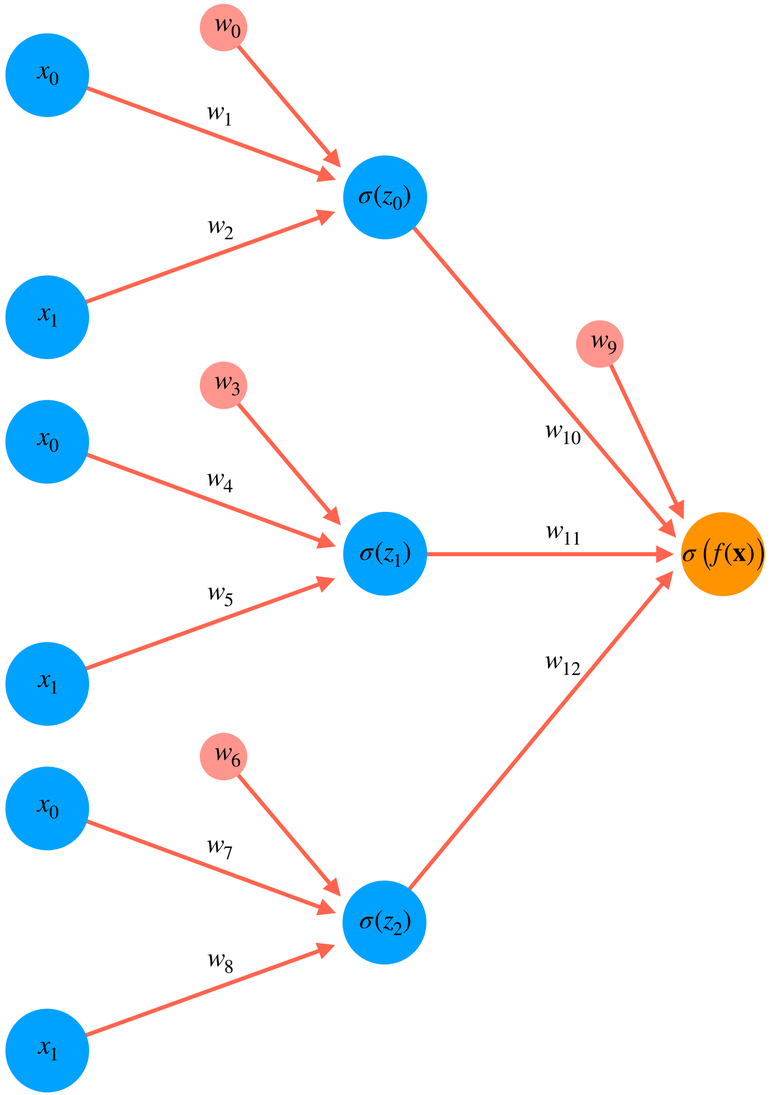
$$\mathbf{Z} = \begin{array}{|c|c|c|} \hline z_0 & z_1 & z_2 \\ \hline 0{,}3 & 0{,}75 & 0{,}78 \\ \hline 0{,}25 & 0{,}1 & 0{,}95 \\ \hline \ldots & \ldots & \\ \hline 0{,}79 & 0{,}99 & 0{,}3 \\ \hline 0{,}34 & 0{,}6 & 0{,}1 \\ \hline \end{array} \qquad \mathbf{y} = \begin{array}{|c|} \hline y \\ \hline 0 \\ \hline 1 \\ \hline \ldots \\ \hline 1 \\ \hline 0 \\ \hline \end{array}$$

Solution: Why not use a logistic regression?



Logistic regression not so simple problem

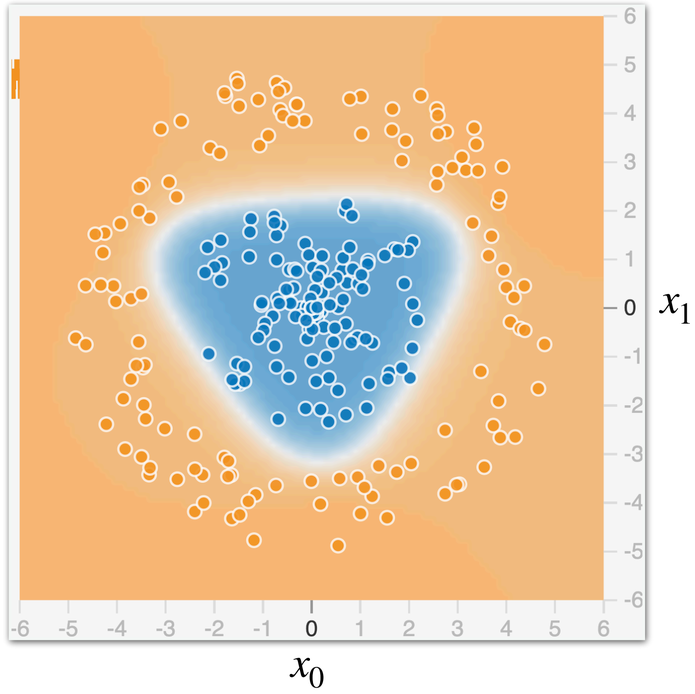
> *Solution:* Connect subproblems



Mathematical form

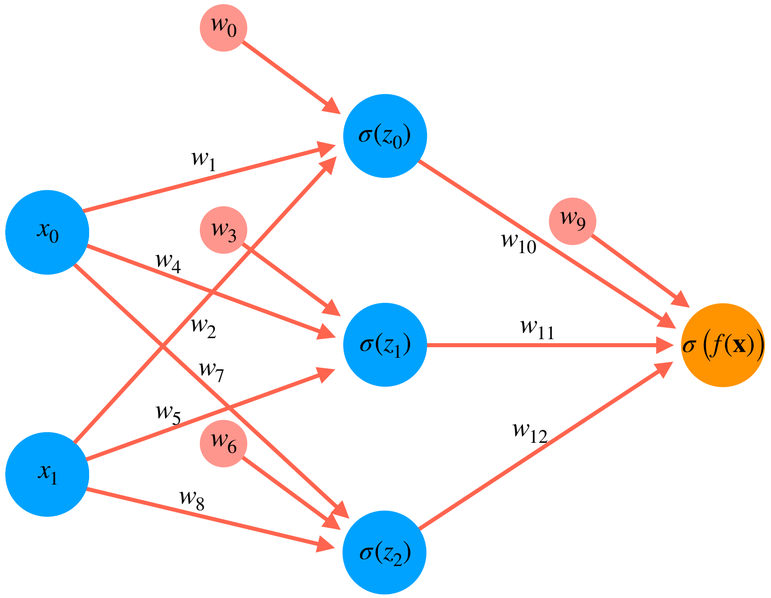
$$\sigma\Big(w_9 + \sigma(w_0 + x_0 w_1 + x_1 w_2) w_{10} \quad + \\ \sigma(w_3 + x_0 w_4 + x_1 w_5) w_{11} \quad + \\ \sigma(w_6 + x_0 w_7 + x_1 w_8) w_{12}\Big) = \sigma\Big(f(x)\Big)$$

Solution



Logistic regression not so simple problem

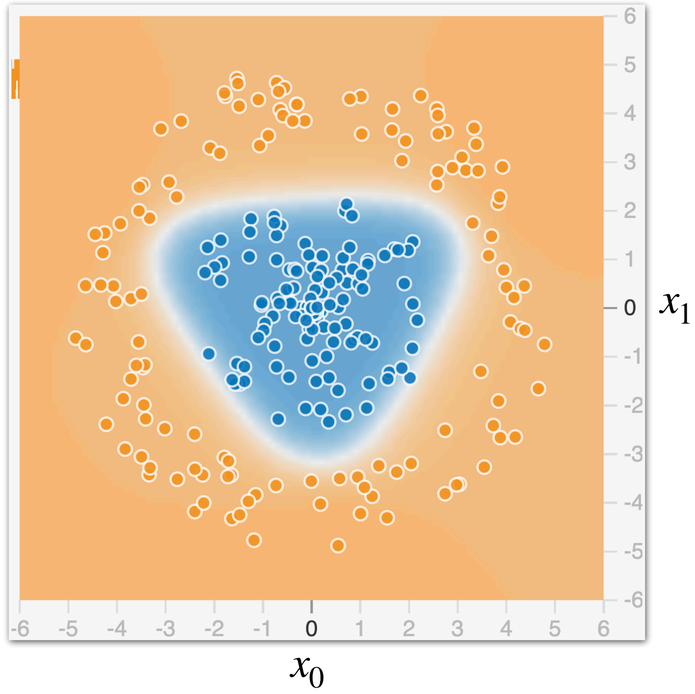
> *Solution:* Connect subproblems



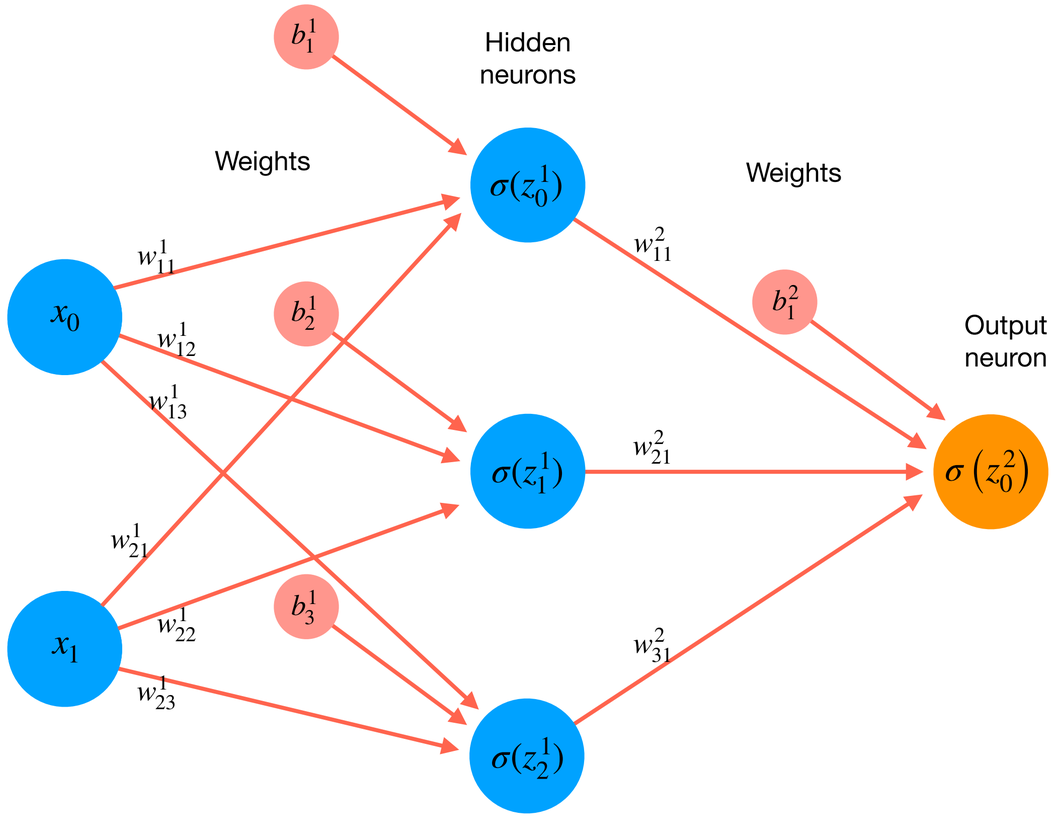
Mathematical form

$$\sigma\Big(w_9 + \sigma(w_0 + x_0 w_1 + x_1 w_2) w_{10} \quad + \\ \sigma(w_3 + x_0 w_4 + x_1 w_5) w_{11} \quad + \\ \sigma(w_6 + x_0 w_7 + x_1 w_8) w_{12}\Big) = \sigma\Big(f(x)\Big)$$

Solution



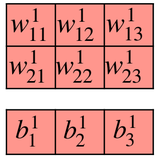
Multilayer perceptron – the structure

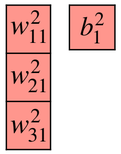


Input layer

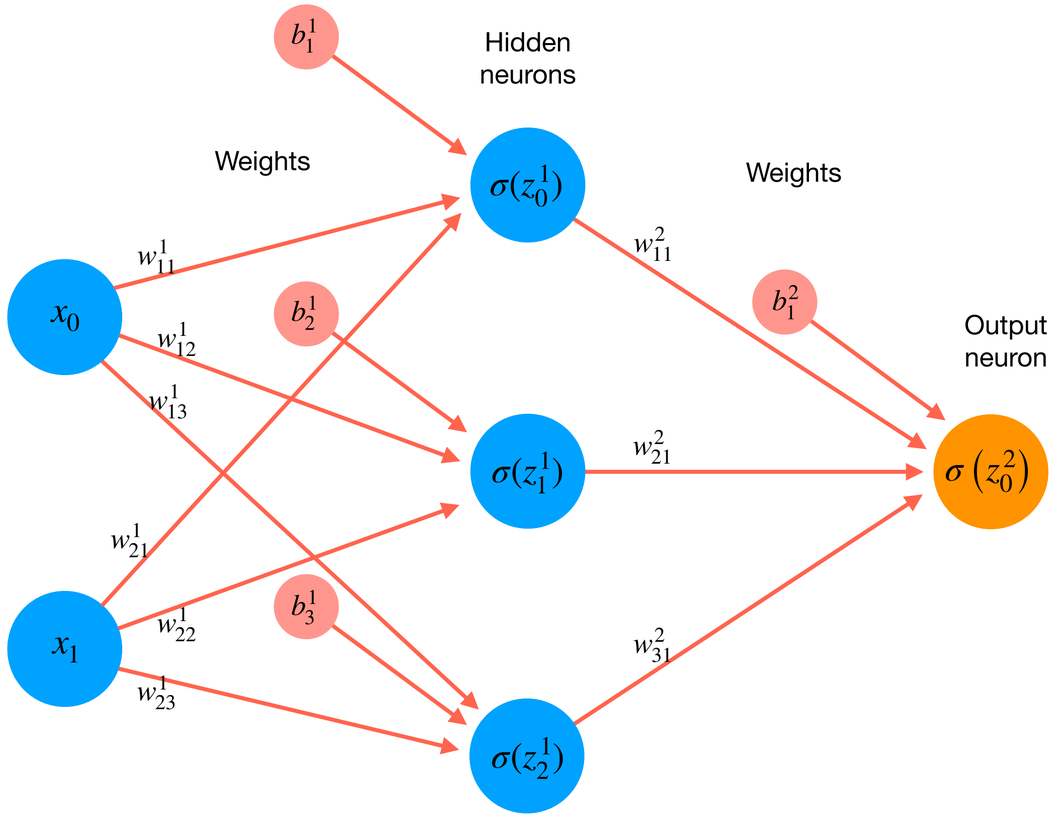
Hidden layer

Output layer

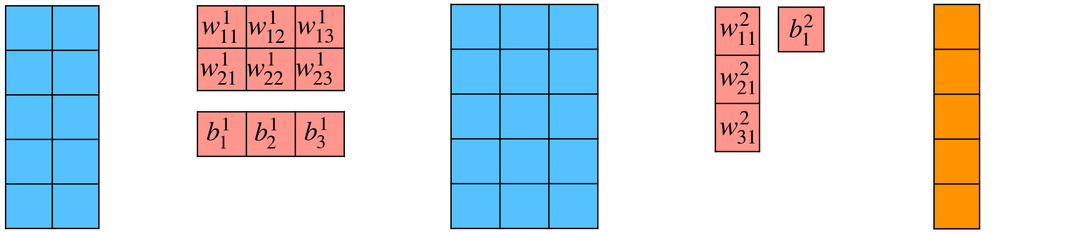




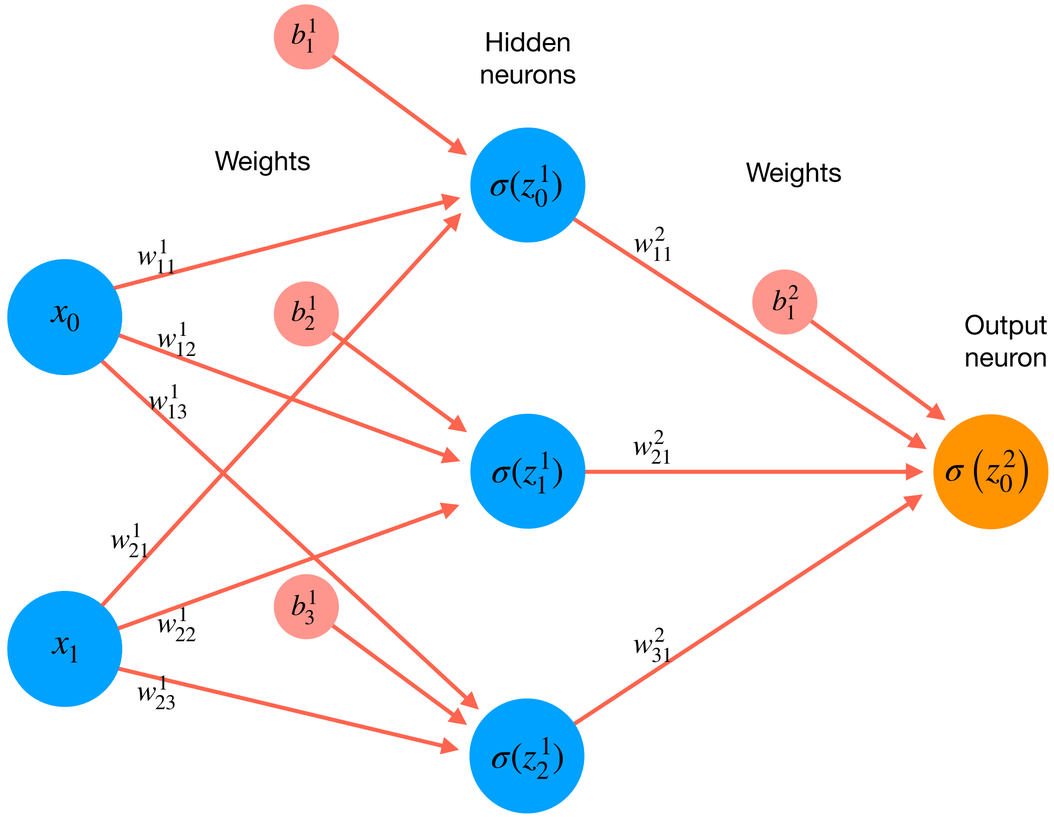
Multilayer perceptron – the structure



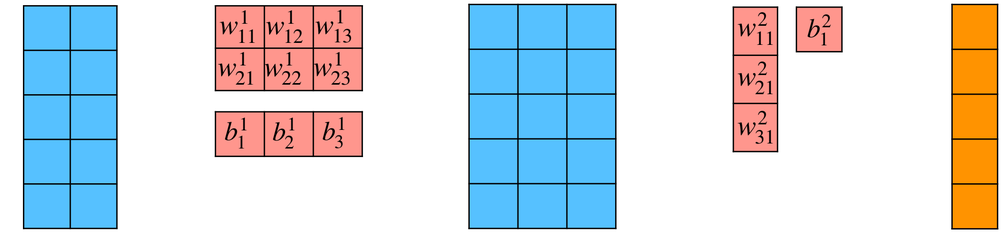
Input layer **Hidden layer** **Output layer**



Multilayer perceptron – the structure



Input layer **Hidden layer** **Output layer**



Activation: The output of a neuron

$$\sigma\left(\mathbf{z}^l\right) = \mathbf{a}^l$$

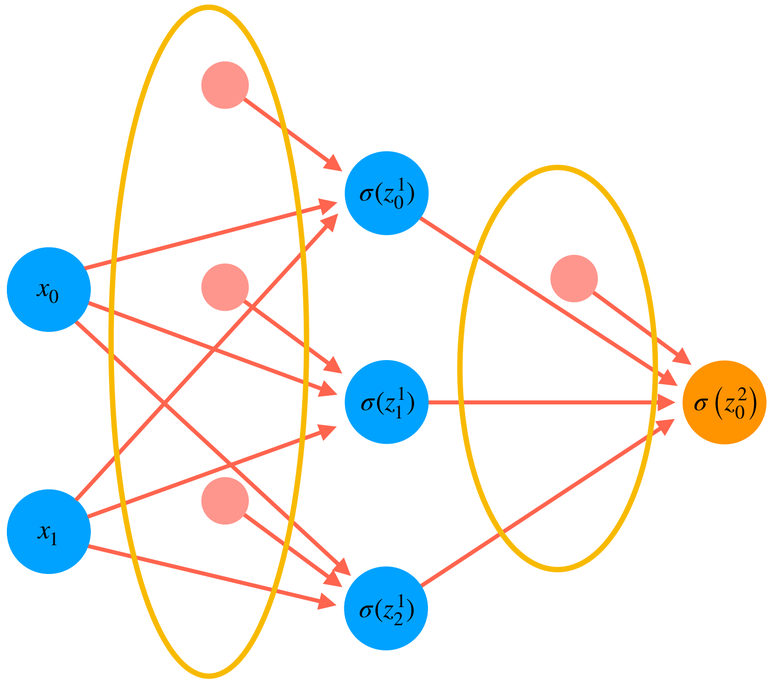
$$\sigma\left(\mathbf{b}^l + \mathbf{W}^l\mathbf{a}^{l-1}\right) = \mathbf{a}^l$$

```python
def feedforward(self, a):
    """Return the output of the network if ``a`` is input."""

    # Q2: What is `a`? How many iterations will this loop run?
    # For a `sizes=[2, 3, 1]` network, what is the shape of `a`
    # at each iteration?
    for b, w in zip(self.biases, self.weights):
        a = sigmoid(np.dot(w, a) + b)
    return a
```

Today: How neural networks learn

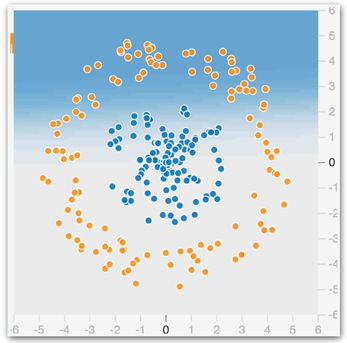
Meaning: how to **set weights** so a network makes **good predictions**

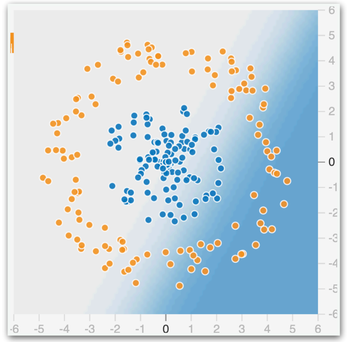


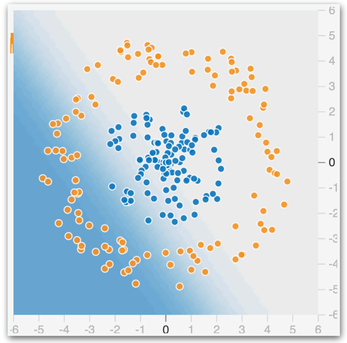
STEPS:

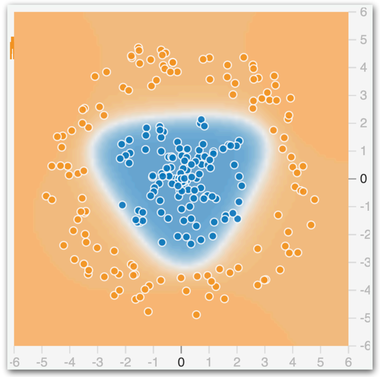
1. Gradient descent
2. Backpropagation

How do we find the weights that give the best classification?



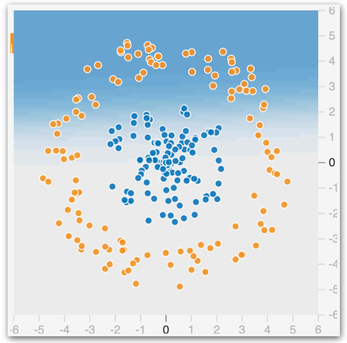


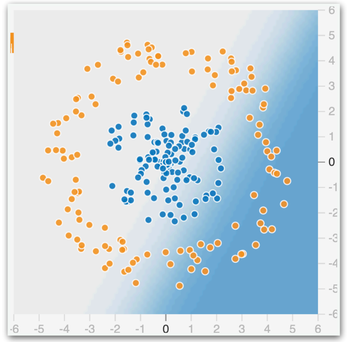


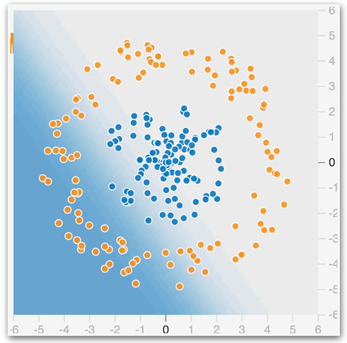


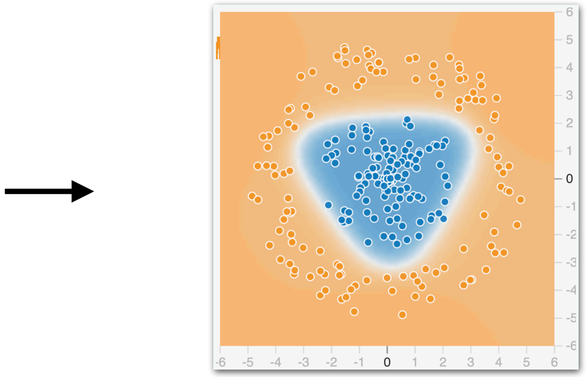
example

How do we find the weights that give the best classification?







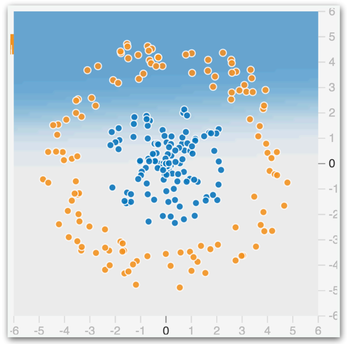


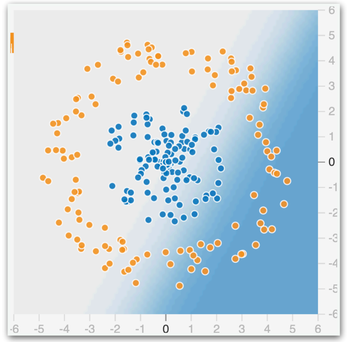
predicted from **X** given **W** | true

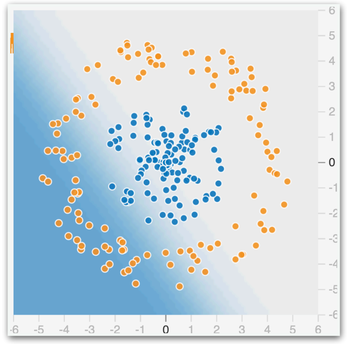
$$\hat{\mathbf{y}} = \begin{bmatrix} 0.96 \\ 0.10 \\ 0.04 \\ \cdots \\ 0.70 \\ 0.02 \\ 0.99 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

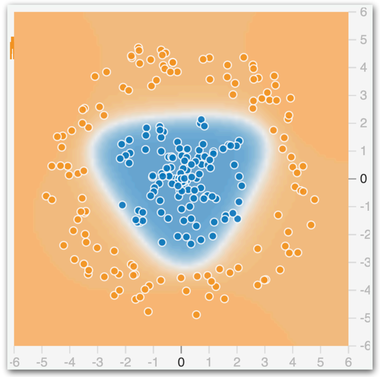
$$\begin{aligned} C(\mathbf{W}) &= \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2 \\ &= (0.96 - 1)^2 \\ &+ (0.10 - 0)^2 \\ &+ (0.04 - 0)^2 \\ &+ \ldots \\ &+ (0.70 - 1)^2 \\ &+ (0.02 - 0)^2 \\ &+ (0.99 - 1)^2 \end{aligned}$$

How do we find the weights that give the best classification?









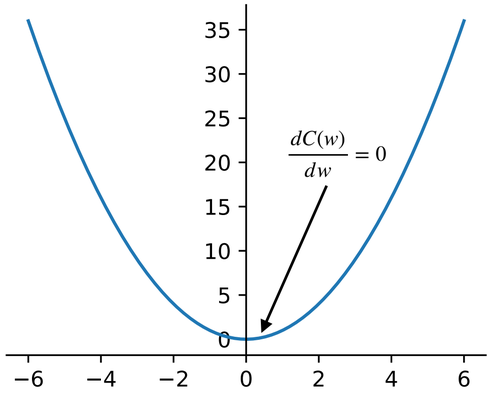
predicted from **X** given **W**

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.96 \\ 0.10 \\ 0.04 \\ \cdots \\ 0.70 \\ 0.02 \\ 0.99 \end{bmatrix}$$

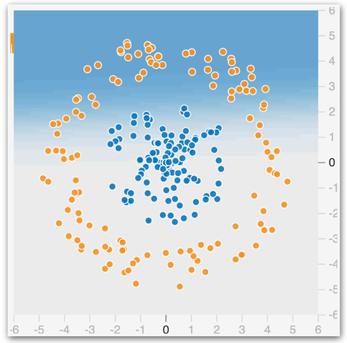
true

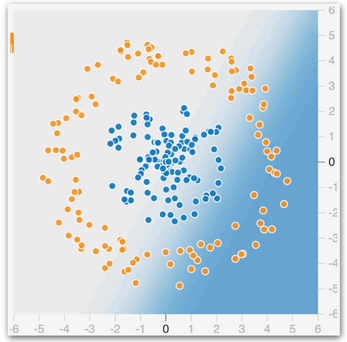
$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

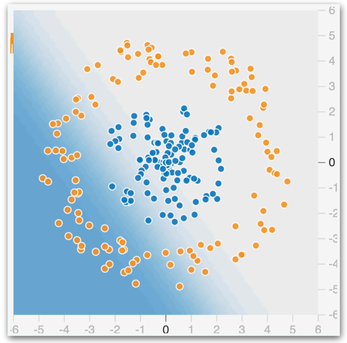
$$\begin{aligned} C(\mathbf{W}) &= \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2 \\ &= (0.96-1)^2 \\ &+(0.10-0)^2 \\ &+(0.04-0)^2 \\ &+\ldots \\ &+(0.70-1)^2 \\ &+(0.02-0)^2 \\ &+(0.99-1)^2 \end{aligned}$$

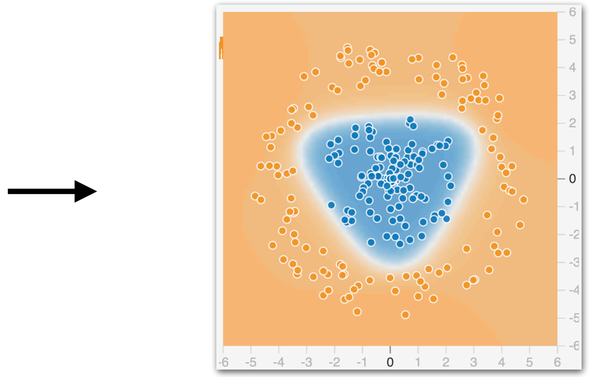


How do we find the weights that give the best classification?









example

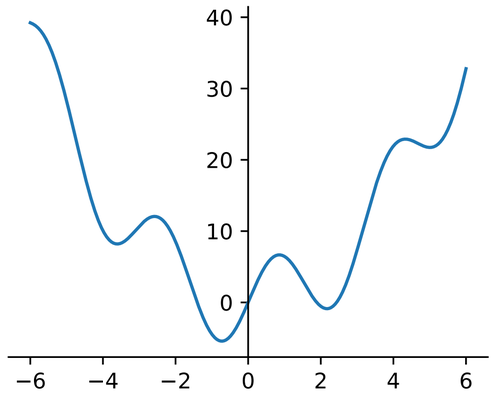
predicted from **X** given **W**

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.96 \\ 0.10 \\ 0.04 \\ \cdots \\ 0.70 \\ 0.02 \\ 0.99 \end{bmatrix}$$

true

$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

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

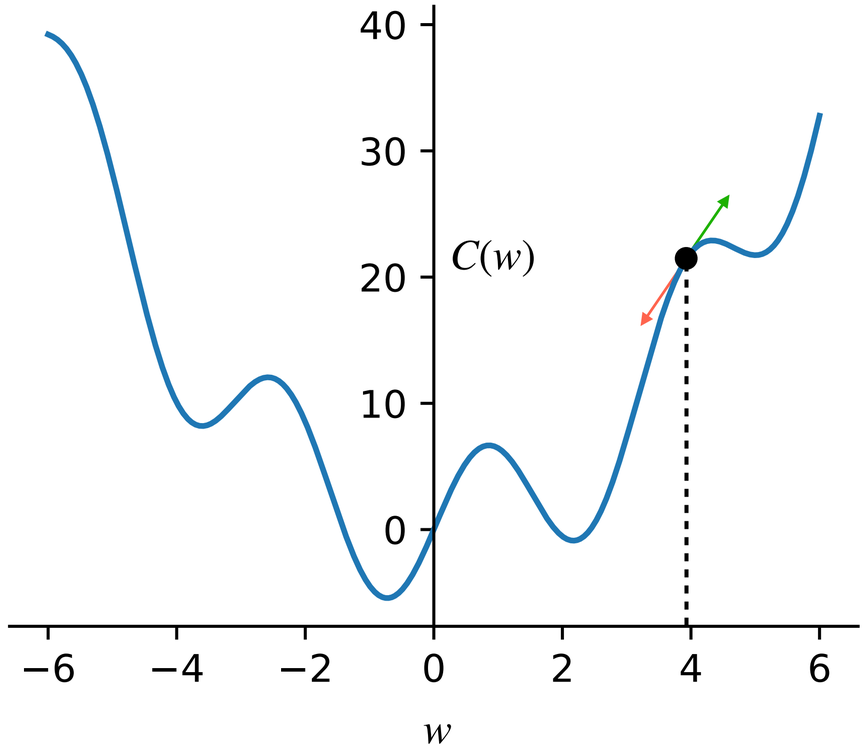


How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.82 \\ 0.47 \\ 0.21 \\ \cdots \\ 0.41 \\ 0.74 \\ 0.23 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

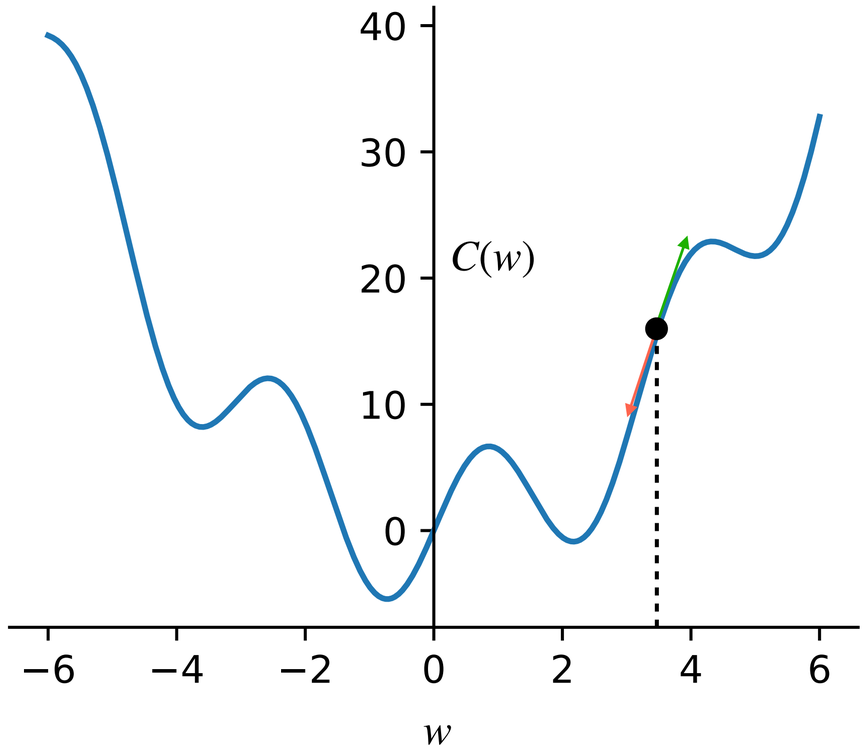
$$\frac{\partial C}{\partial w} = 8.2$$

How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.82 \\ 0.32 \\ 0.12 \\ \cdots \\ 0.25 \\ 0.85 \\ 0.11 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

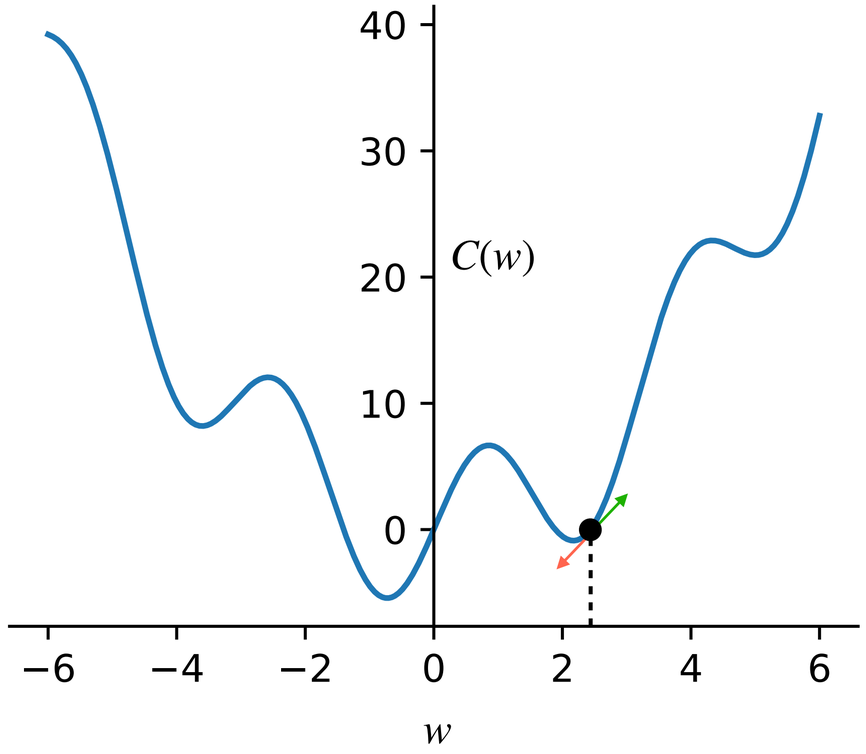
$$\frac{\partial C}{\partial w} = 18.2$$

How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.96 \\ 0.08 \\ 0.02 \\ \cdots \\ 0.12 \\ 0.96 \\ 0.01 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

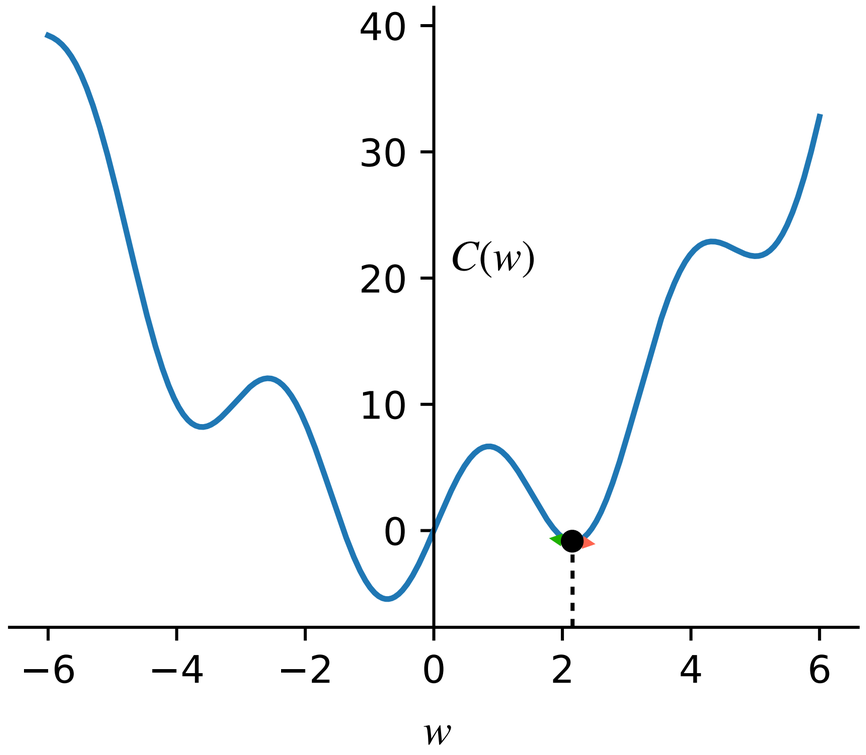
$$\frac{\partial C}{\partial w} = 4.5$$

How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.99 \\ 0.02 \\ 0.00 \\ \cdots \\ 0.01 \\ 1.00 \\ 0.00 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

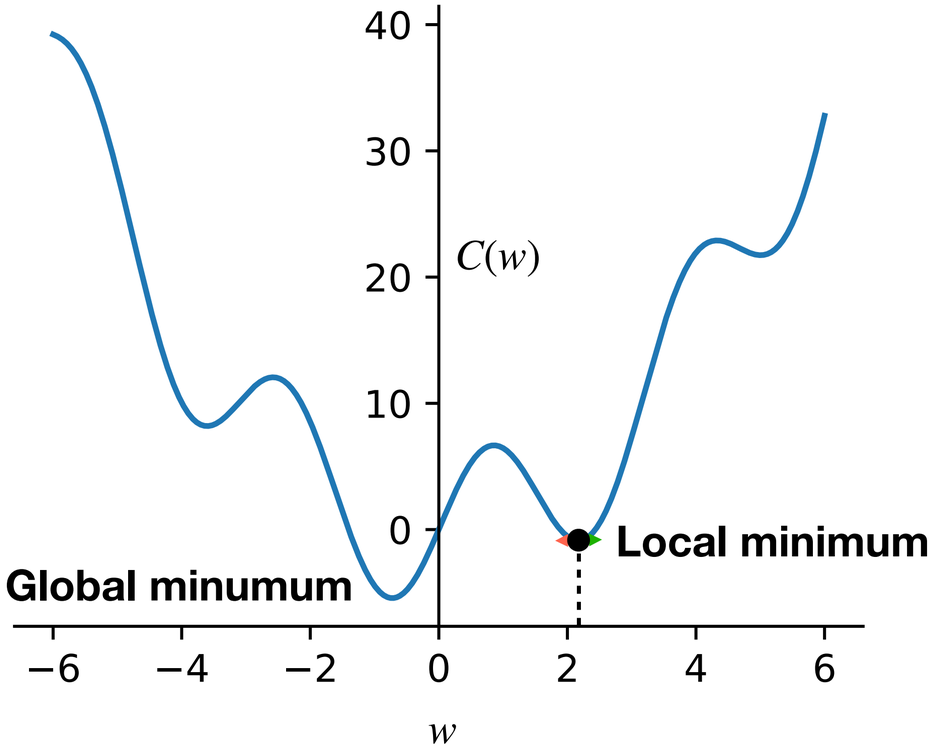
$$\frac{\partial C}{\partial w} = -0.2$$

How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.99 \\ 0.02 \\ 0.00 \\ \cdots \\ 0.01 \\ 1.00 \\ 0.00 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

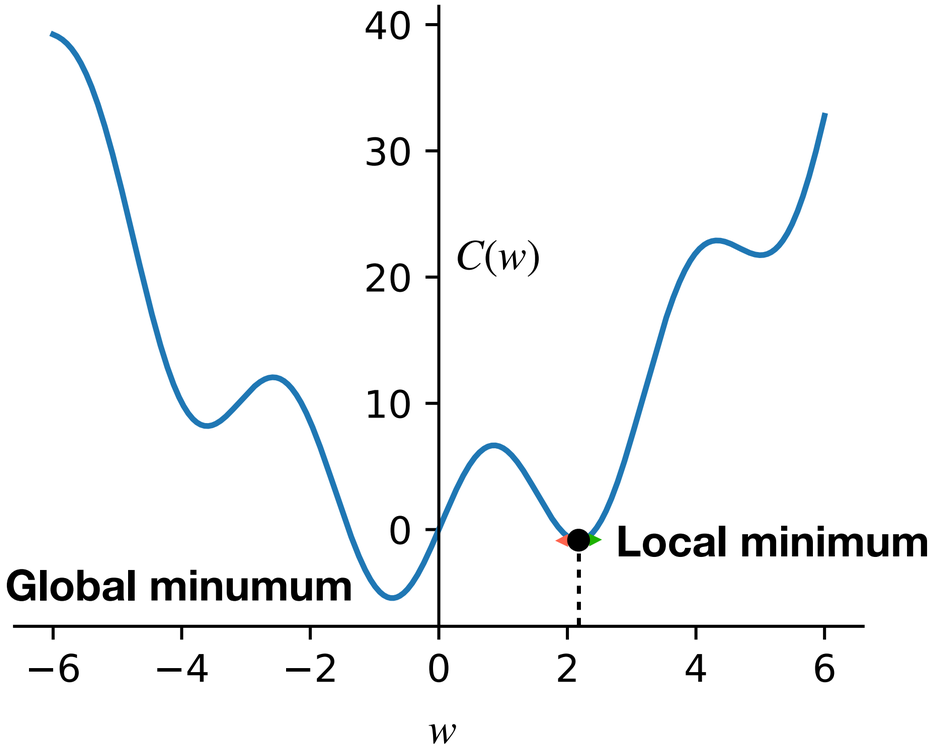
$$\frac{\partial C}{\partial w} = 0.001$$

How do we find the weights that give the best classification?

> Gradient Descent

Prediction $\hat{\mathbf{y}}$ versus ground truth $\mathbf{y}$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.99 \\ 0.02 \\ 0.00 \\ \cdots \\ 0.01 \\ 1.00 \\ 0.00 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdots \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Gradient

$$\frac{\partial C}{\partial w} = 0.001$$

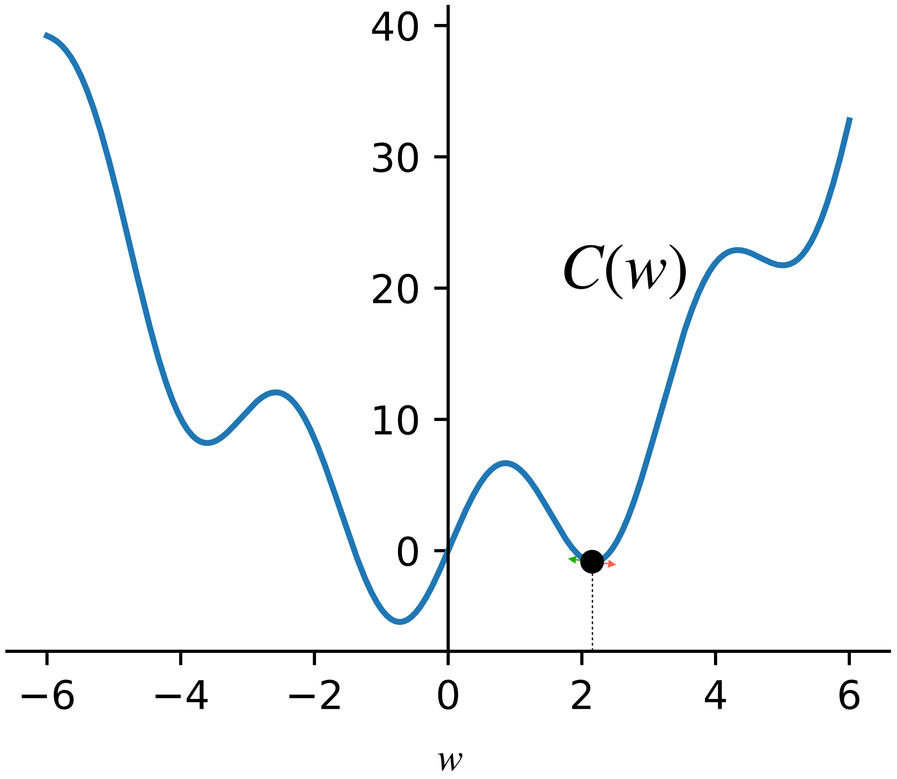
Updating rule
gradient descent**:**

$$w_{t+1} = w_t - \eta \frac{\partial C_{w_t}}{\partial w}$$

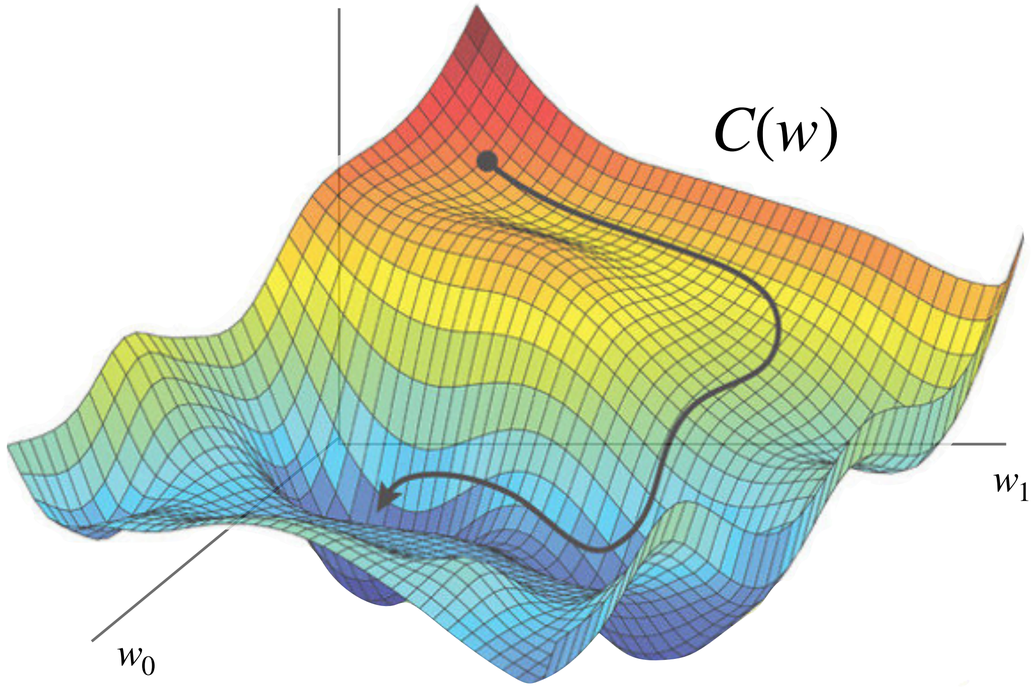
How do we find the weights that give the best classification?

> Gradient Descent

1D



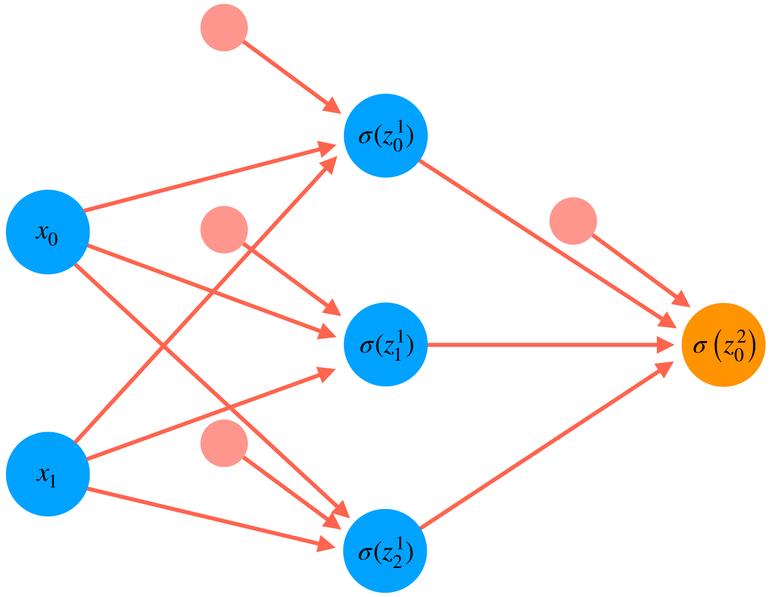
2D



How do we find the weights that give the best classification?

> Gradient Descent

13D? Or higher?

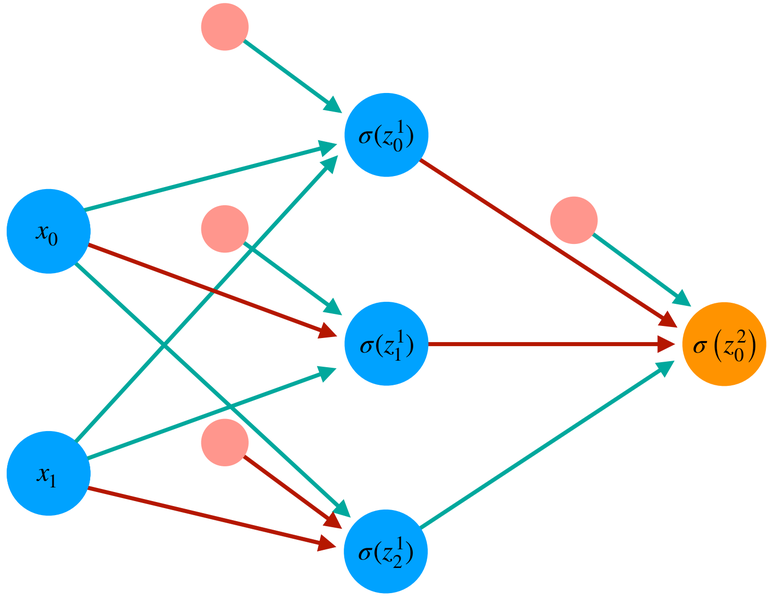


$$\mathbf{W} = \begin{bmatrix} w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \\ w_{21}^2 \\ w_{22}^2 \\ w_{23}^2 \\ b_1^1 \\ b_2^1 \\ b_3^1 \\ w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \\ b_1^2 \end{bmatrix}$$

How do we find the weights that give the best classification?

> Gradient Descent

13D? Or higher?

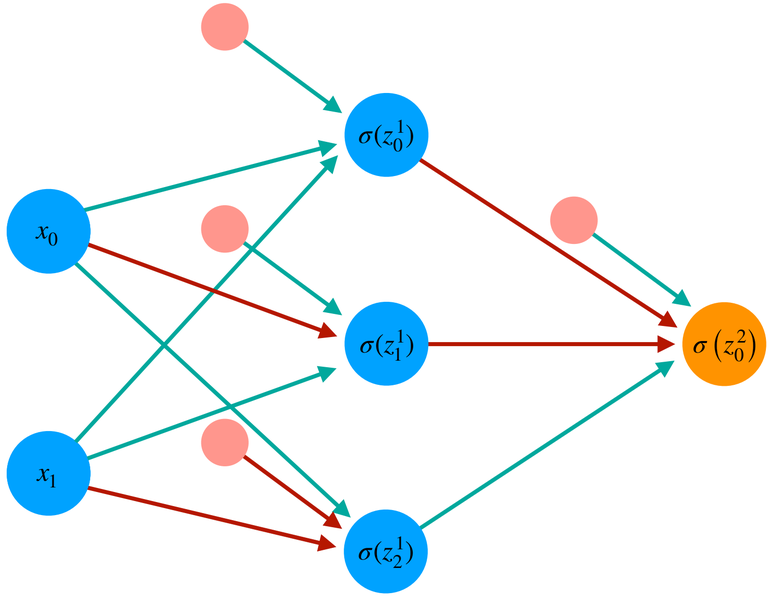


$$\mathbf{W} = \begin{bmatrix} w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \\ w_{21}^2 \\ w_{22}^2 \\ w_{23}^2 \\ b_1^1 \\ b_2^1 \\ b_3^1 \\ w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \\ b_1^2 \end{bmatrix} \qquad \nabla C(\mathbf{W}) = \begin{bmatrix} \partial C/\partial w_{11}^1 \\ \partial C/\partial w_{12}^1 \\ \partial C/\partial w_{13}^1 \\ \partial C/\partial w_{21}^2 \\ \partial C/\partial w_{22}^2 \\ \partial C/\partial w_{23}^2 \\ \partial C/\partial b_1^1 \\ \partial C/\partial b_2^1 \\ \partial C/\partial b_3^1 \\ \partial C/\partial w_{11}^2 \\ \partial C/\partial w_{21}^2 \\ \partial C/\partial w_{31}^2 \\ \partial C/\partial b_1^2 \end{bmatrix}$$

How do we find the weights that give the best classification?

> Gradient Descent

13D? Or higher?

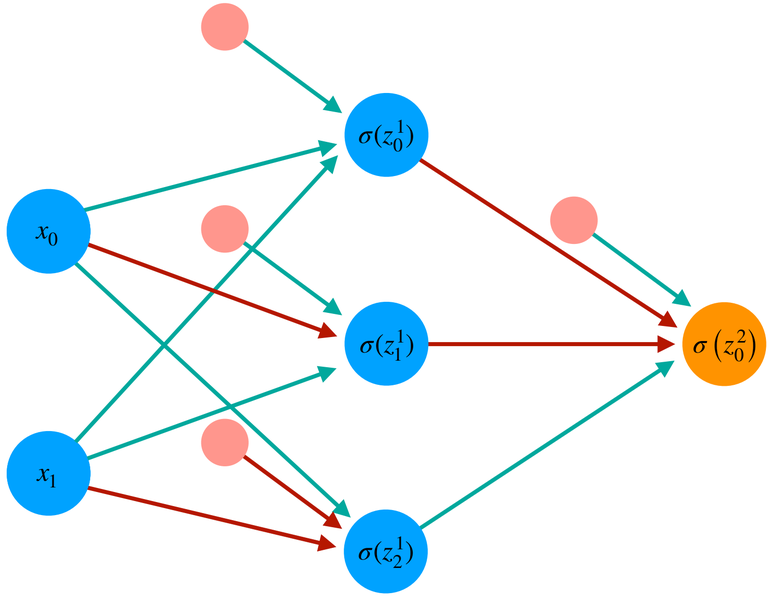


$$\mathbf{W} = \begin{bmatrix} w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \\ w_{21}^2 \\ w_{22}^2 \\ w_{23}^2 \\ b_1^1 \\ b_2^1 \\ b_3^1 \\ w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \\ b_1^2 \end{bmatrix} \qquad \nabla C(\mathbf{W}) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}$$

How do we find the weights that give the best classification?

> Gradient Descent

13D? Or higher?



$$\mathbf{W} = \begin{bmatrix} w_{11}^1 \\ w_{12}^1 \\ w_{13}^1 \\ w_{21}^2 \\ w_{22}^2 \\ w_{23}^2 \\ b_1^1 \\ b_2^1 \\ b_3^1 \\ w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \\ b_1^2 \end{bmatrix} \qquad \nabla C(\mathbf{W}) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta \left(-\nabla C(\mathbf{W}_t) \right)$$

How it all hangs together

(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**

One number, **the cost**

(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**

$$\mathbf{W}_{t+1} = \mathbf{W}_t + -\eta \nabla C(\mathbf{W}_t)$$

(5) Repeat 3 and 4

r is usually called the *learning rate*

How it all hangs together

(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**

$C(\mathbf{W};\ \mathbf{X}, \mathbf{y})$

One number, **the cost**

(3) The cost function gradient at current weights: $-\nabla C(\mathbf{W}_t) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ 0.54 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}$

(4) Updating weights

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta \left(-\nabla C(\mathbf{W}_t)\right)$$

r is usually called the *learning rate*

(5) Repeat 3 and 4

How it all hangs together

(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**

$C(\mathbf{W}\,;\,\mathbf{X},\mathbf{y})$

One number, **the cost**

(3) The cost function gradient at current weights: $-\nabla C(\mathbf{W}_t) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ 0.54 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}$

(4) Updating weights

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta\left(-\nabla C(\mathbf{W}_t)\right)$$

r is usually called the *learning rate*

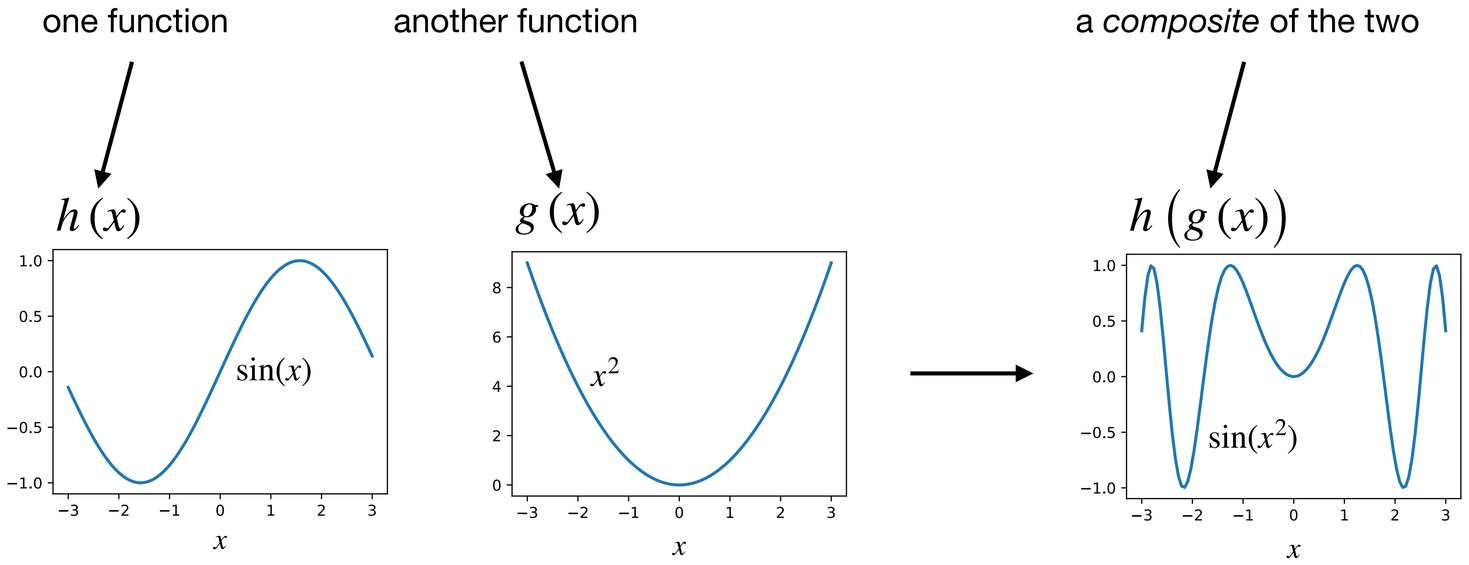
(5) Repeat 3 and 4

Find the gradients with **Backpropagation**

Backpropagation

The algorithm for computing the analytical **gradient** of the cost function
(It's just the chain rule)

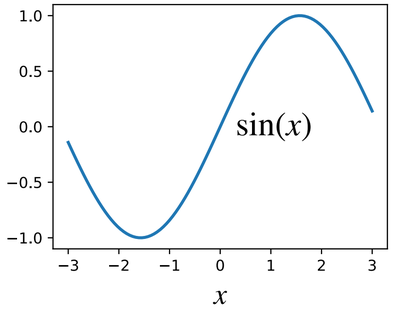
Chain rule – Taking derivatives of *composite functions*



Chain rule – Taking derivatives of *composite functions*

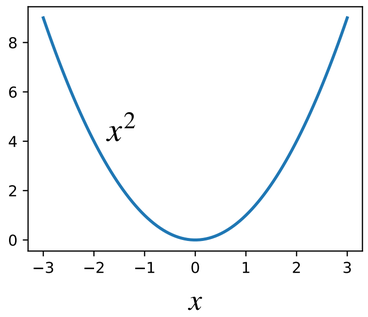
one function

$h(x)$



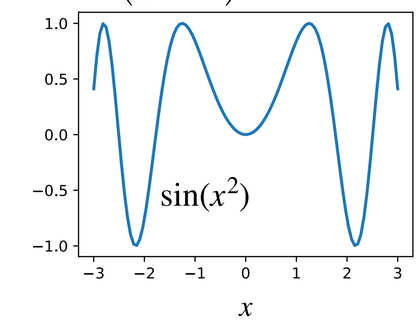
another function

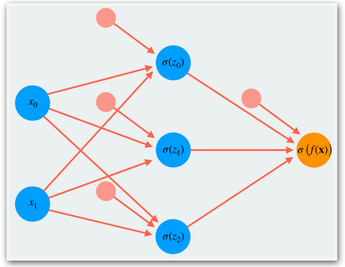
$g(x)$



a *composite* of the two

$h(g(x))$





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

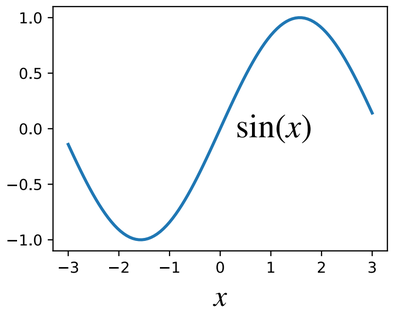
Chain rule – Taking derivatives of *composite functions*

one function

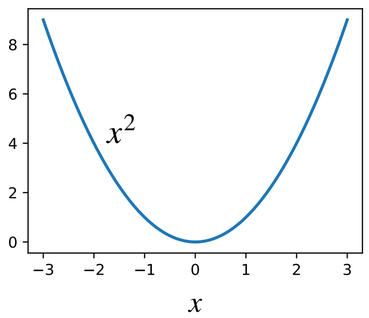
another function

a *composite* of the two

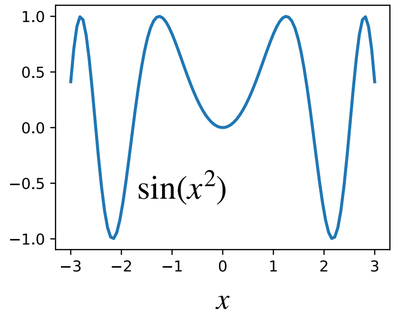
$h(x)$



$g(x)$



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

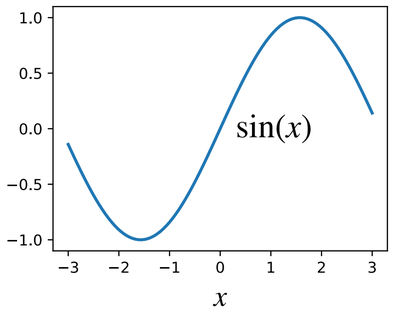


$h\left(g(x)\right)$

Chain rule – Taking derivatives of *composite functions*

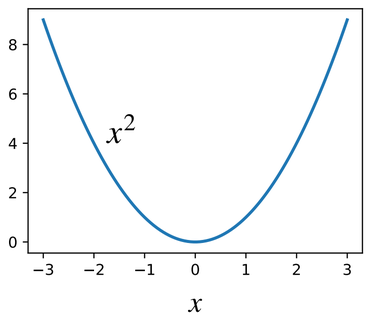
one function

$h(x)$



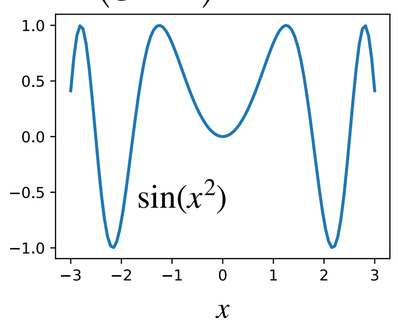
another function

$g(x)$



a *composite* of the two

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



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

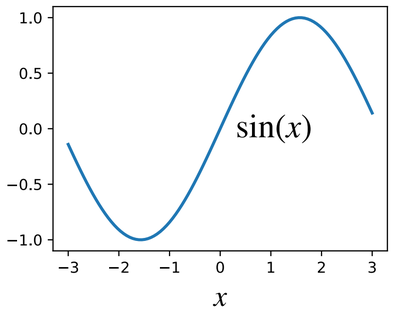
Chain rule says:

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

Chain rule – Taking derivatives of *composite functions*

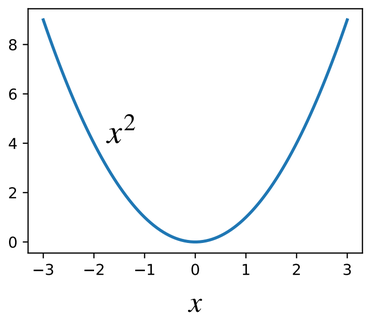
one function

$h(x)$



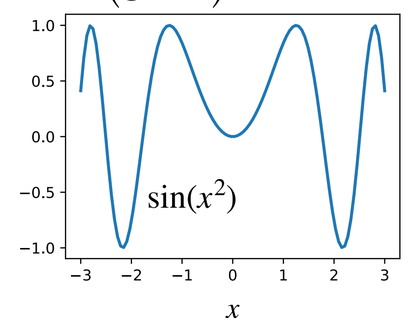
another function

$g(x)$

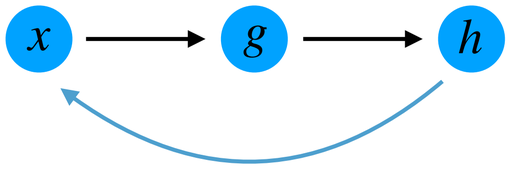


a *composite* of the two

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



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



Chain rule says:

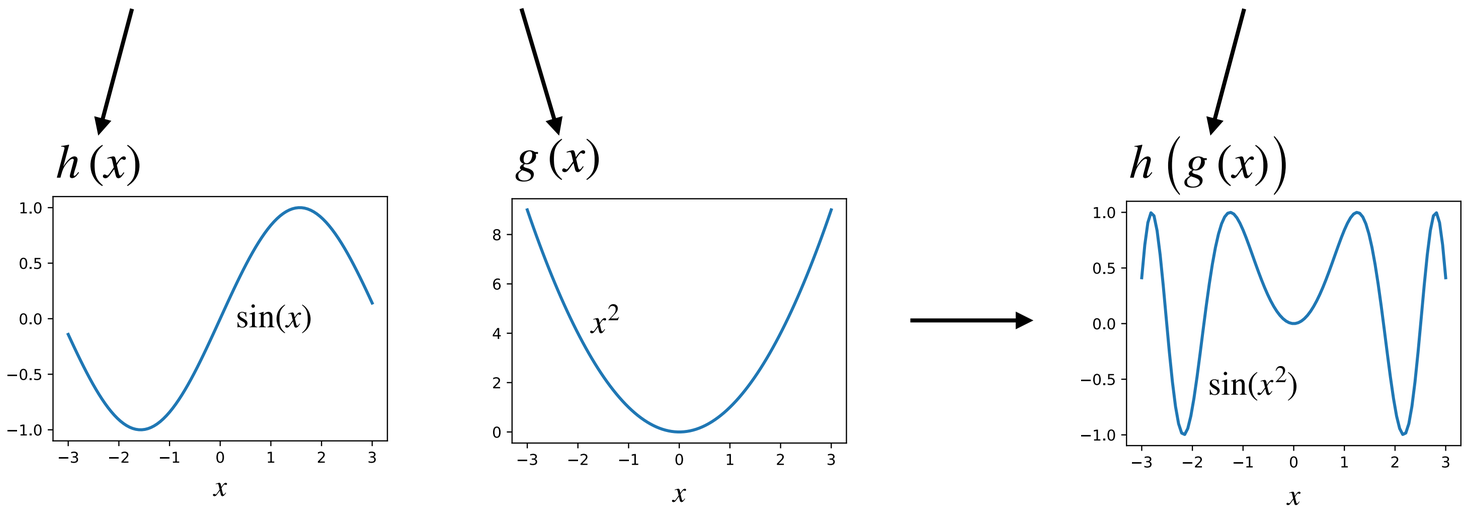
$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

Chain rule – Taking derivatives of *composite functions*

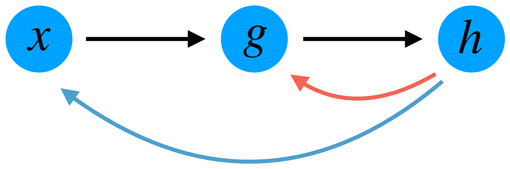
one function

another function

a *composite* of the two



$h(g(x))$



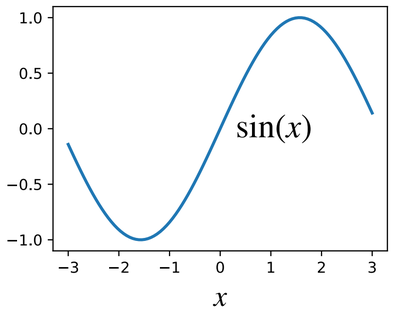
Chain rule says:

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

Chain rule – Taking derivatives of *composite functions*

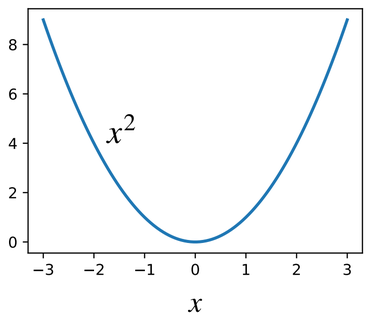
one function

$h(x)$



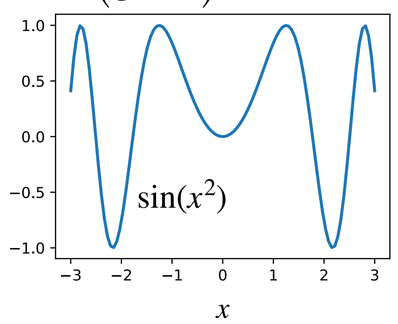
another function

$g(x)$

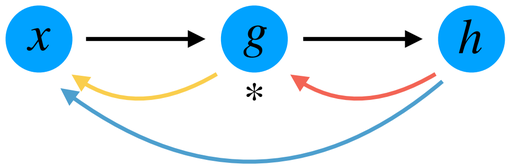


a *composite* of the two

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



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



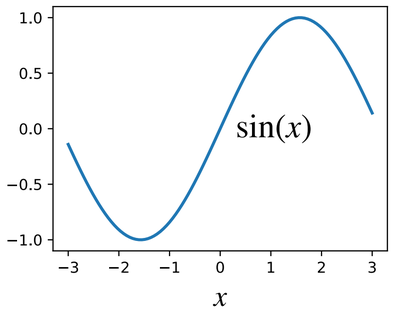
Chain rule says:

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

Chain rule – Taking derivatives of *composite functions*

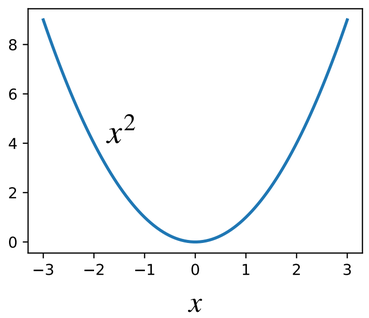
one function

$h(x)$



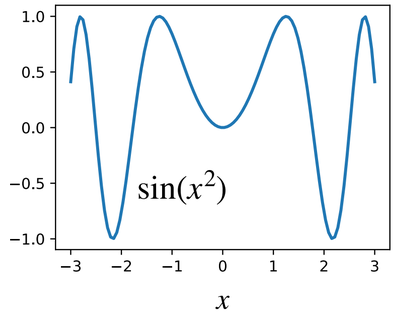
another function

$g(x)$

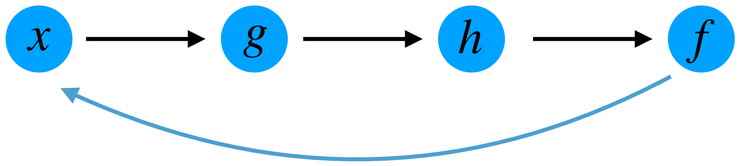


a *composite* of the two

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



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



Chain rule says:

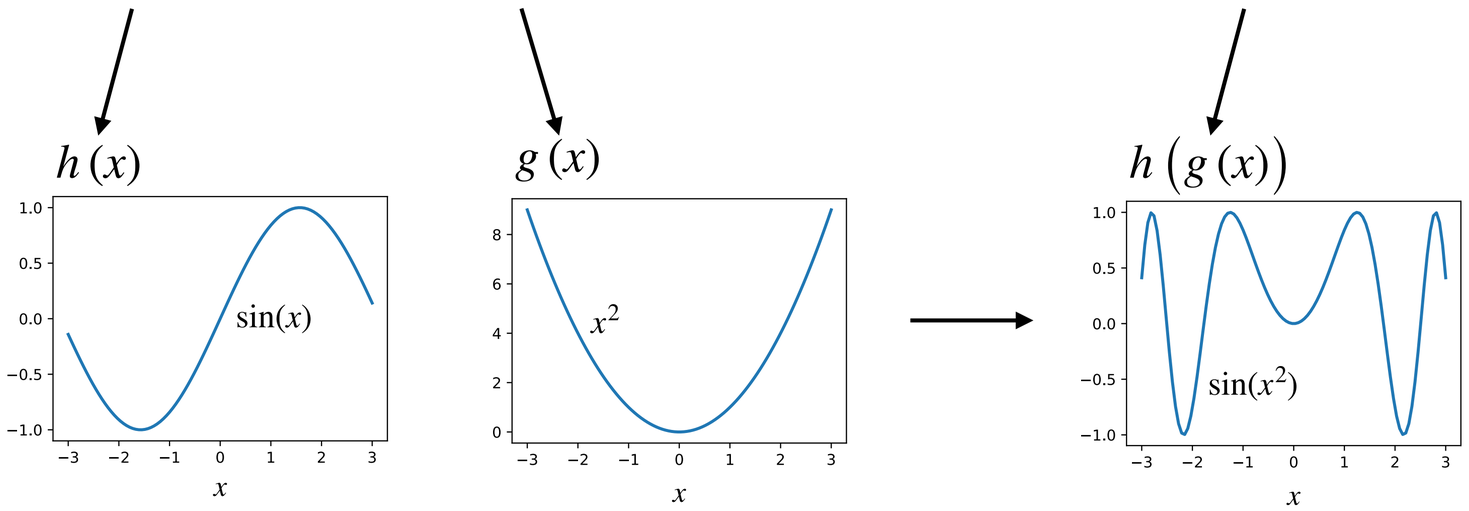
$$\frac{df}{dx} = ?$$

Chain rule – Taking derivatives of *composite functions*

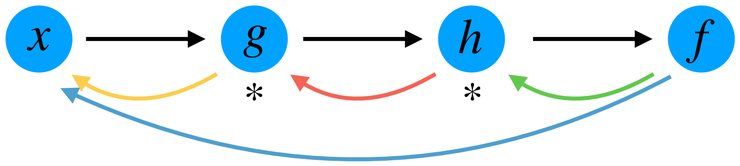
one function

another function

a *composite* of the two



$$f\Big(h\big(g(x)\big)\Big)$$



Chain rule says:

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

3Blue1Brown Chain Rule video

The derivative of the outermost function with respect it's input, multiplied by the derivative of the second-outermost function, with respect to *it's* input, multiplied by the derivate of the innermost function with respect to *it's* input

Composite function:

$$f\Big(h\big(g(x)\big)\Big)$$

Derivative of composite function:

$$\frac{df(h(g(x)))}{dx}\frac{df}{dx} = \frac{df}{dh}\frac{dh}{dg}\frac{dg}{dx}$$

Backpropagation – simple example

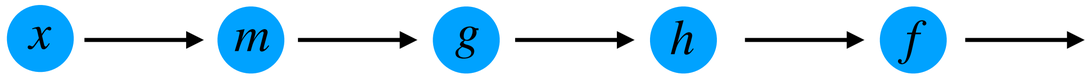
Function:

$$m(z) = -z$$

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

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

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



Backpropagation – simple example

Function:

$$m(z) = -z$$

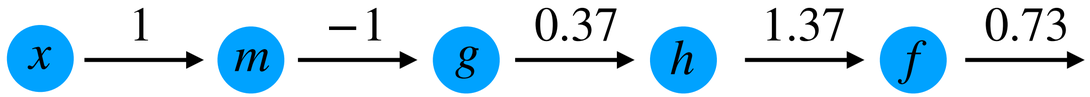
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



Backpropagation – simple example

Function:

$$m(z) = -z$$

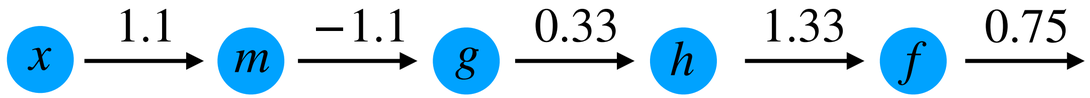
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1.1$$



Backpropagation – simple example

Function:

$$m(z) = -z$$

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

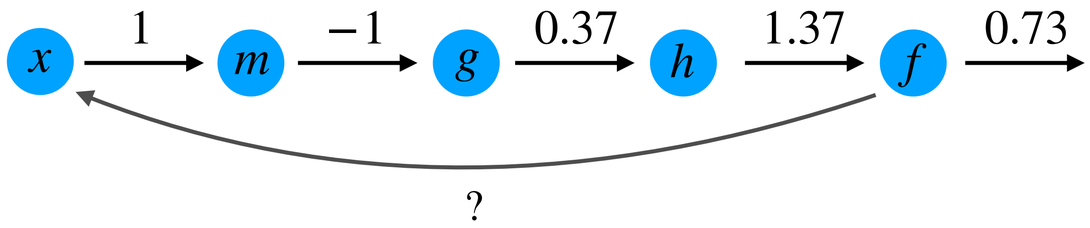
$$h(z) = z + 1$$

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

Input:

$$x' = 1$$

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



Backpropagation – simple example

Function:

$$m(z) = -z$$

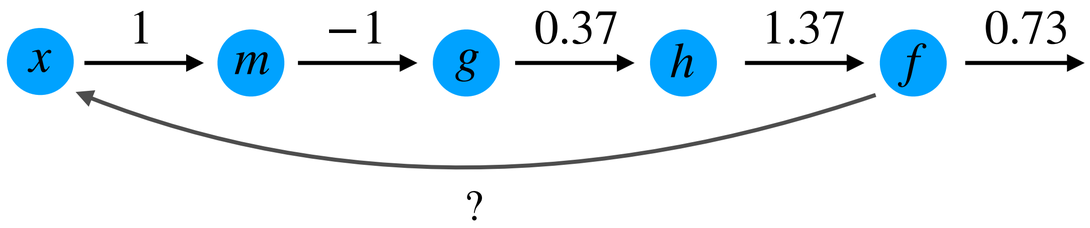
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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

Function:

$$m(z) = -z$$

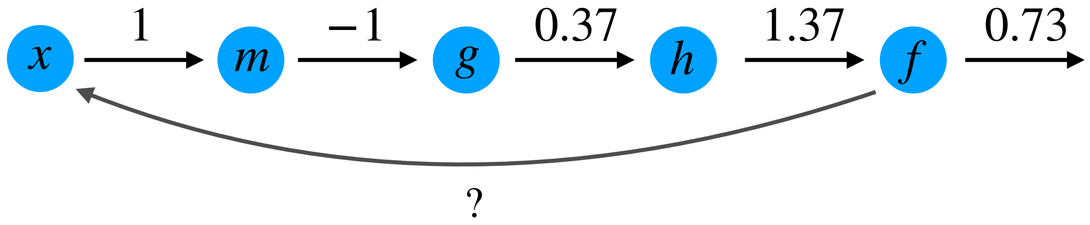
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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:

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

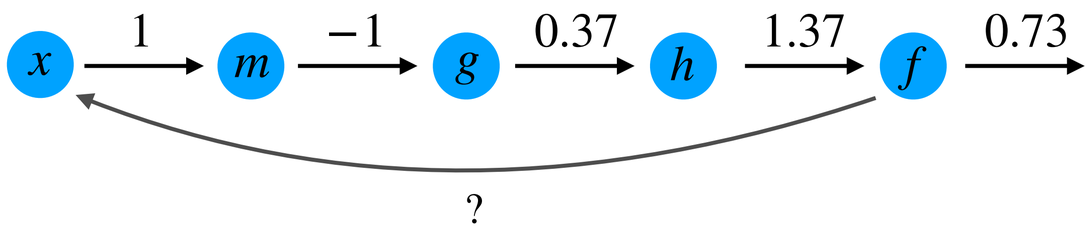
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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

Function:

$$m(z) = -z$$

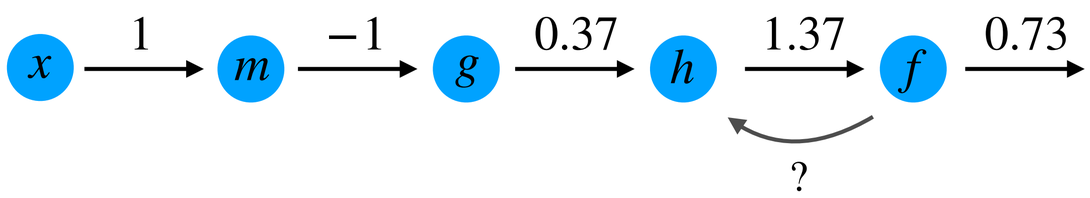
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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*:

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

Model derivatives:

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

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

$$h'(z) = 1$$

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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*:

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

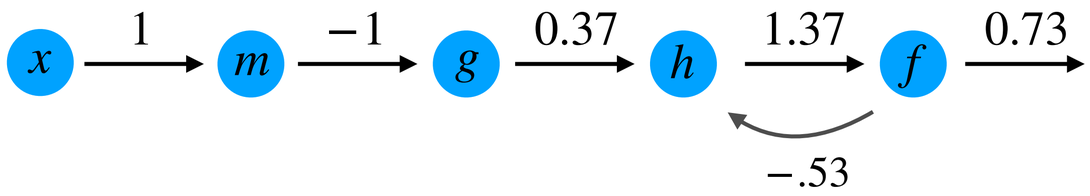
Model derivatives:

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

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

$$h'(z) = 1$$

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



$$\frac{df}{dh} = -\frac{1}{1.37^2} = -.53$$

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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*:

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

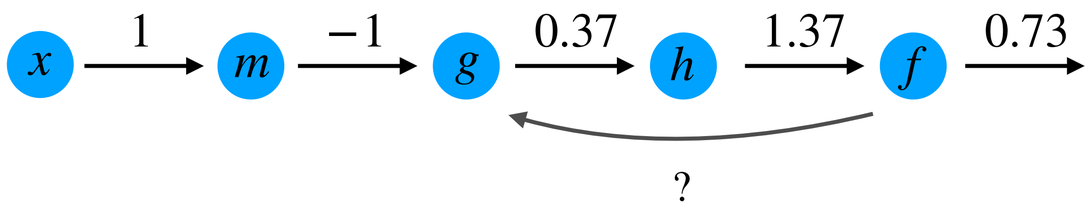
Model derivatives:

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

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

$$h'(z) = 1$$

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



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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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*:

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

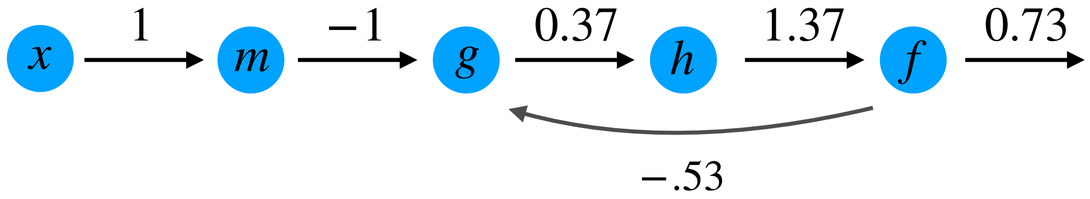
Model derivatives:

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

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

$$h'(z) = 1$$

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



$$\frac{df}{dg} = \frac{df}{dh}\frac{dh}{dg} = -\frac{1}{1.37^2} \cdot 1 = -.53$$

Backpropagation – simple example

Function:

$$m(z) = -z$$

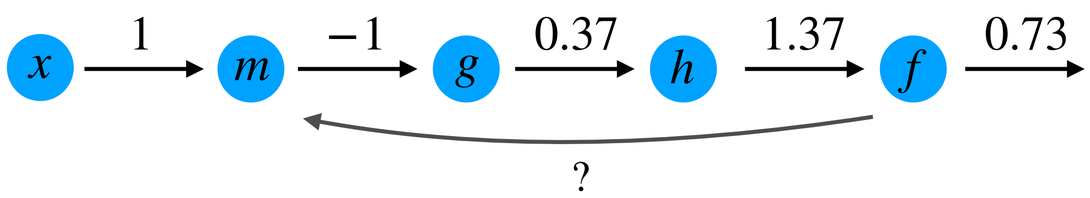
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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

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*:

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

Model derivatives:

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

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

$$h'(z) = 1$$

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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*:

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

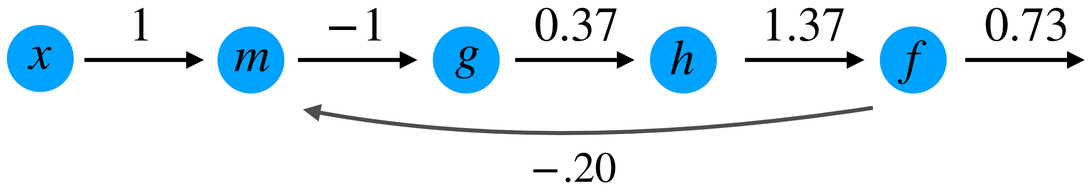
Model derivatives:

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

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

$$h'(z) = 1$$

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



$$\frac{df}{dm} = \frac{df}{dh}\frac{dh}{dg}\frac{dg}{dm} = -\frac{1}{1.37^2} \cdot 1 \cdot e^{-1} = -.20$$

Backpropagation – simple example

Function:

$$m(z) = -z$$

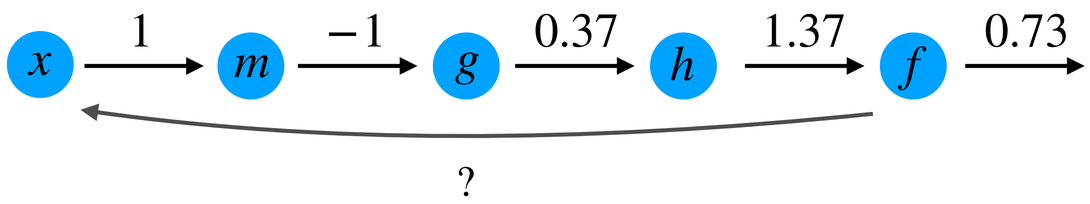
$$g(z) = \exp(z)$$

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

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

Input:

$$x' = 1$$



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

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

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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*:

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

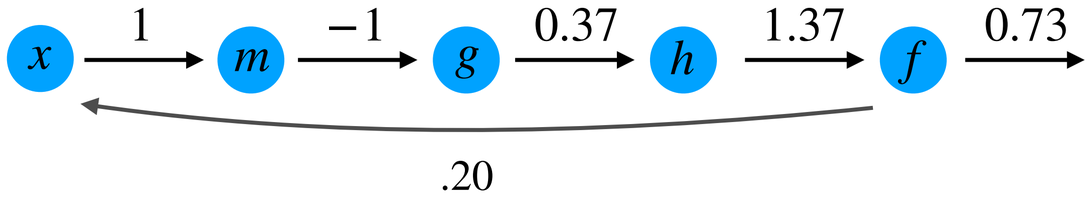
Model derivatives:

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

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

$$h'(z) = 1$$

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



$$\frac{df}{dx} = \frac{df}{dh}\frac{dh}{dg}\frac{dg}{dm}\frac{dm}{dx} = -\frac{1}{1.37^2} \cdot 1 \cdot e^{-1} \cdot (-1) = .20$$

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

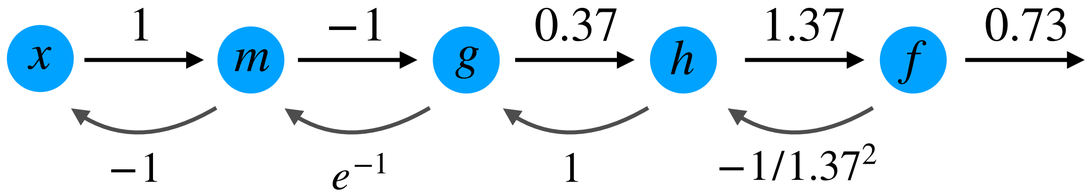
Model derivatives:

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

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

$$h'(z) = 1$$

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



$$\frac{df}{dx} = -\frac{1}{1.37^2} \cdot 1 \cdot e^{-1} \cdot (-1) = .20$$

Sigmoid function:

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

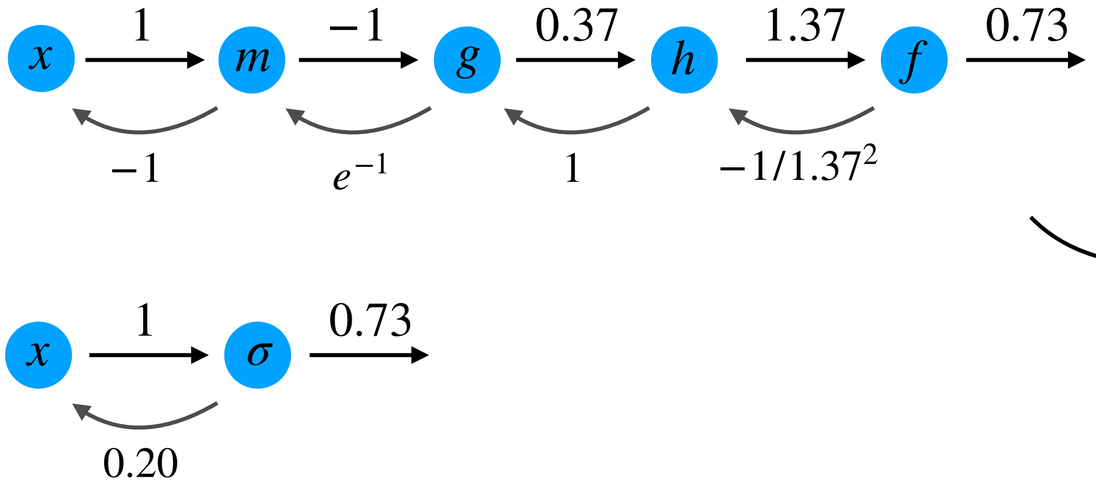
Model derivatives:

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

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

$$h'(z) = 1$$

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



Sigmoid function:

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

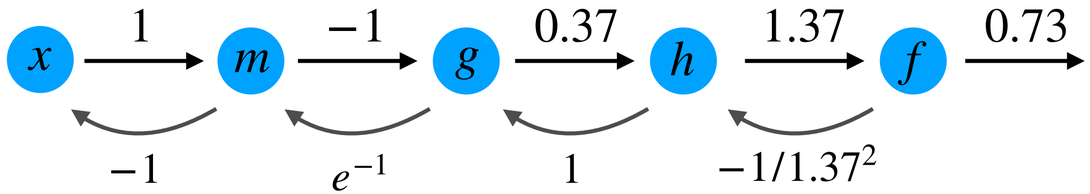
Model derivatives:

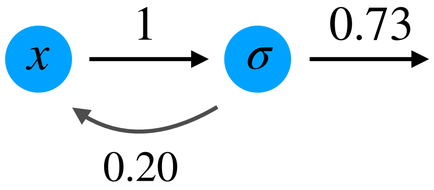
$$m'(z) = -1$$

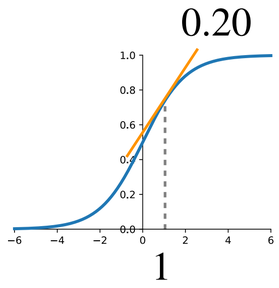
$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

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







Sigmoid function:

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

Backpropagation – simple example

Function:

$$m(z) = -z$$

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

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

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

Input:

$$x' = 1$$

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

A: Propagate gradients backwards using the chain rule!

Chain rule:

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

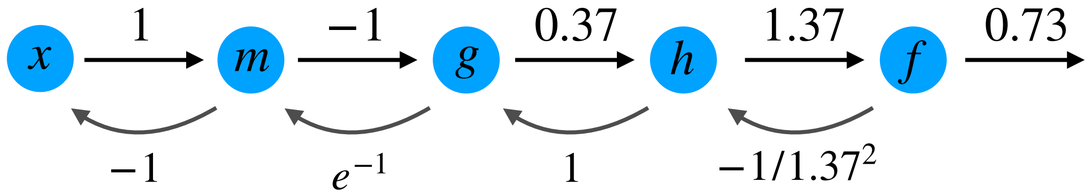
Model derivatives:

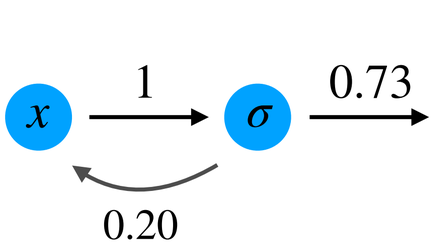
$$m'(z) = -1$$

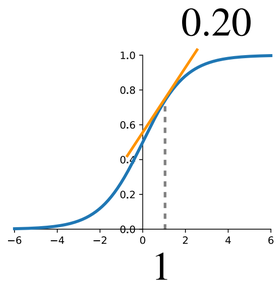
$$g'(z) = \exp(z)$$

$$h'(z) = 1$$

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







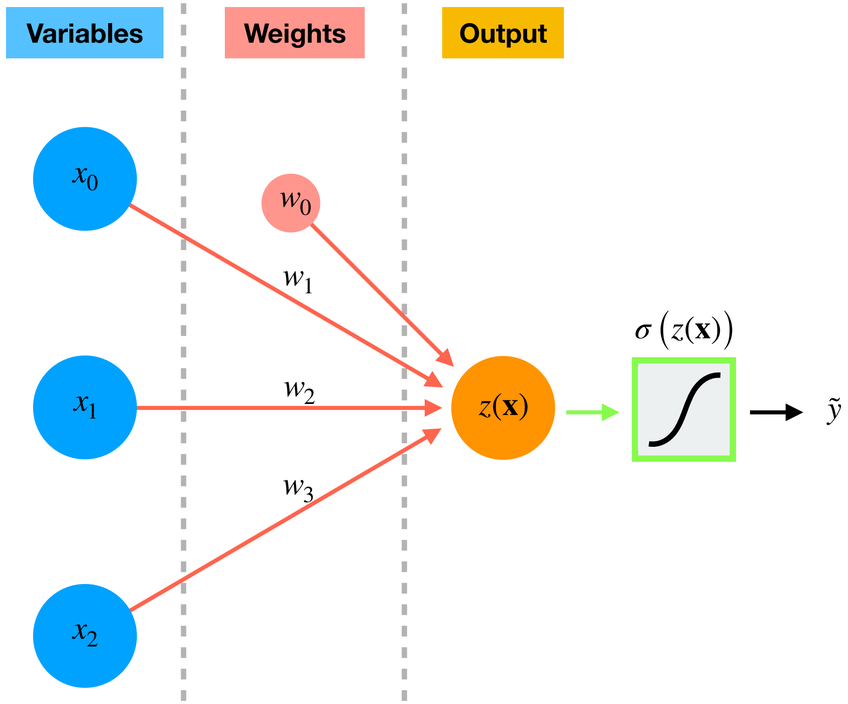
Sigmoid function:

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

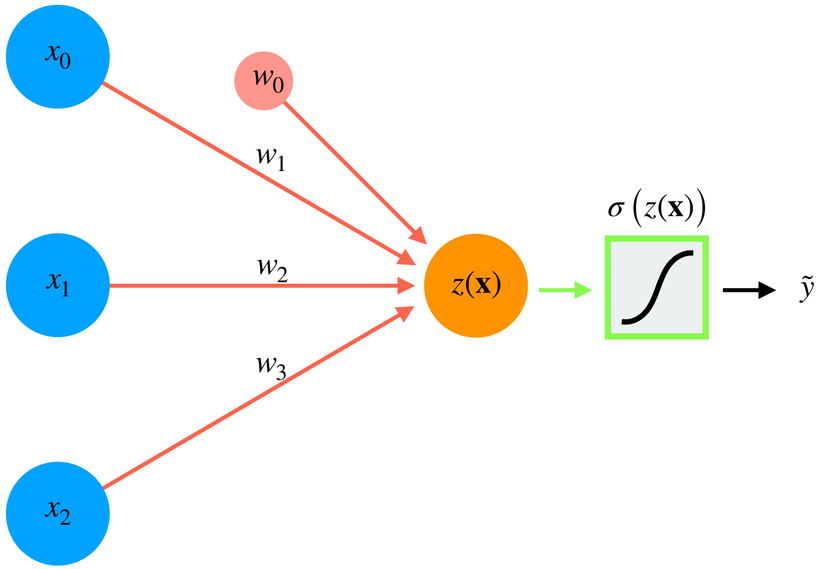
Sigmoid derivative:

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

Backpropagation – on a neural network



Backpropagation – on a neural network



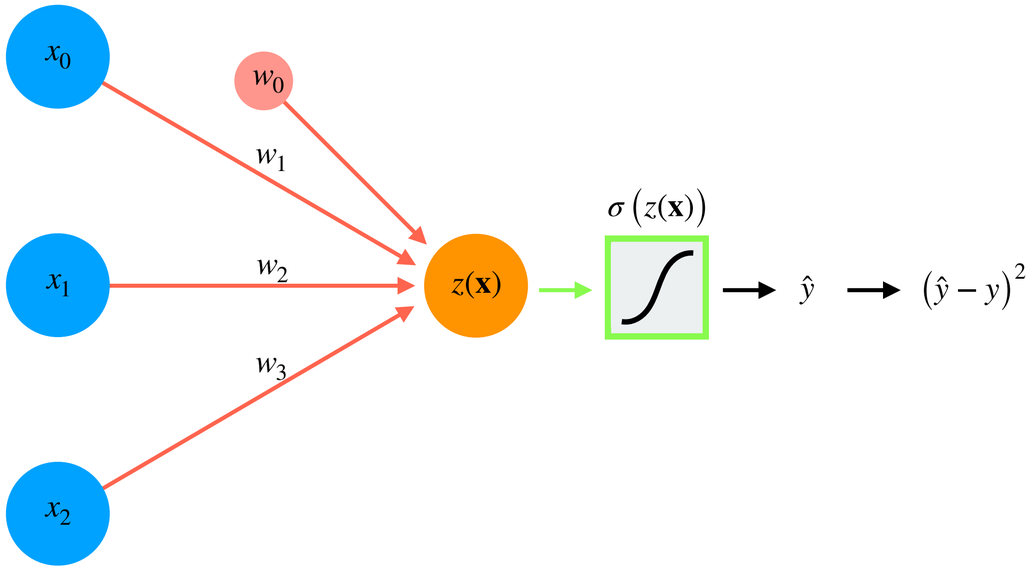
Model:

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

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

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

Backpropagation – on a neural network



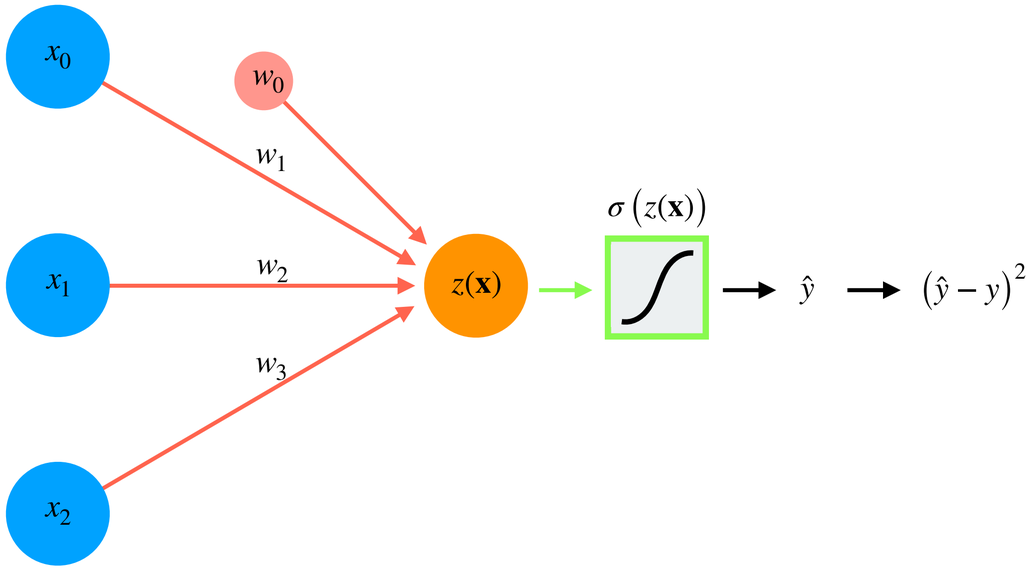
Model:

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

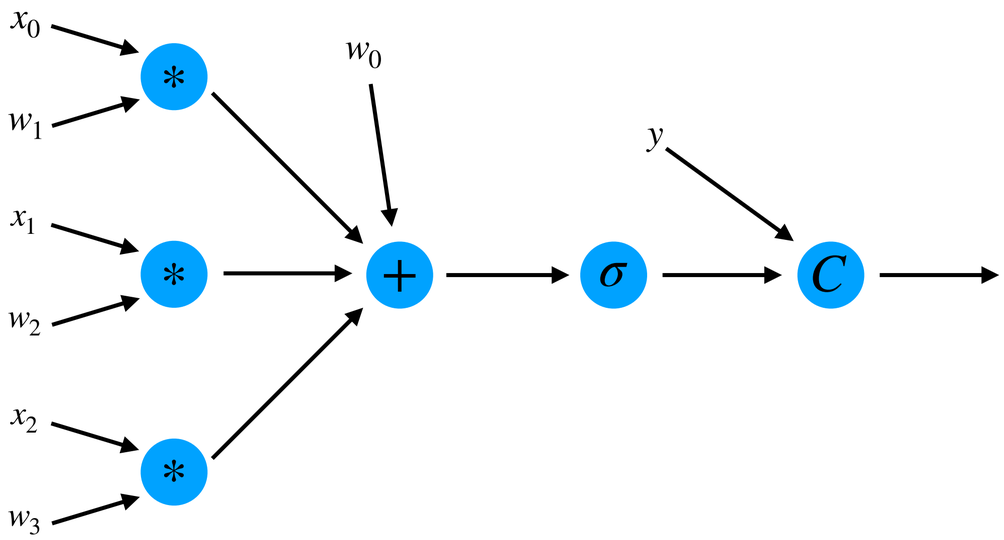
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

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

Backpropagation – on a neural network



As a computational graph:



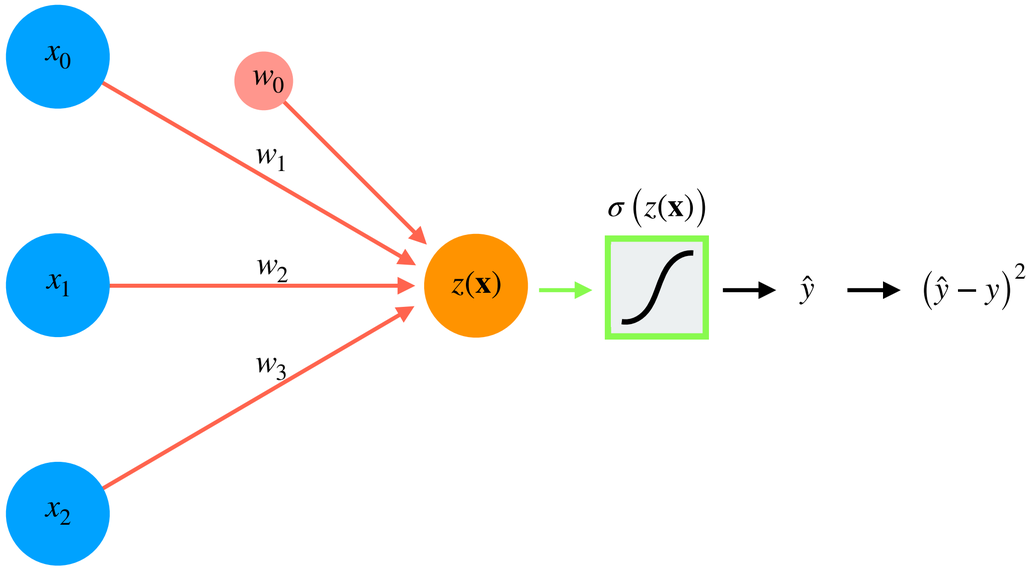
Model:

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

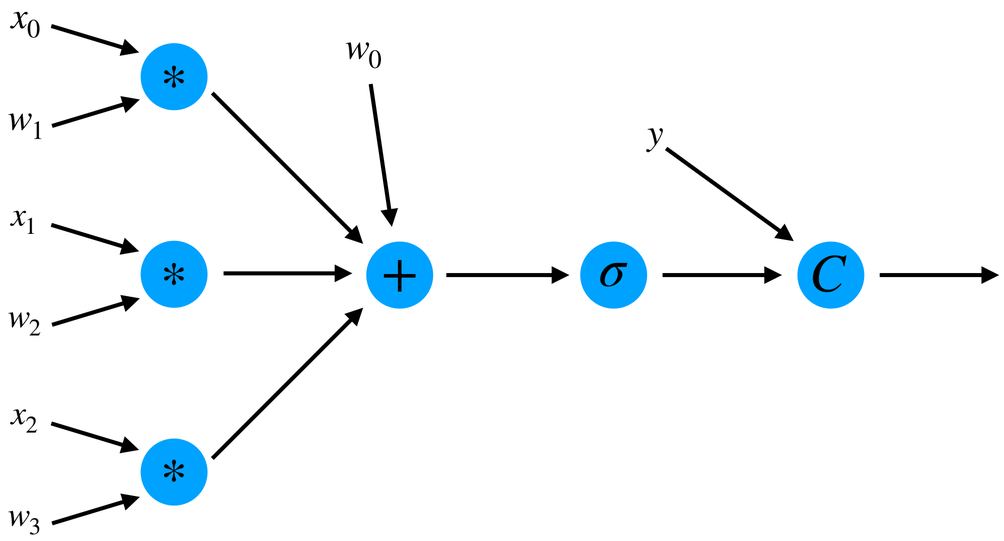
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

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

Backpropagation – on a neural network



As a computational graph:



Model:

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

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

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

Weights:

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

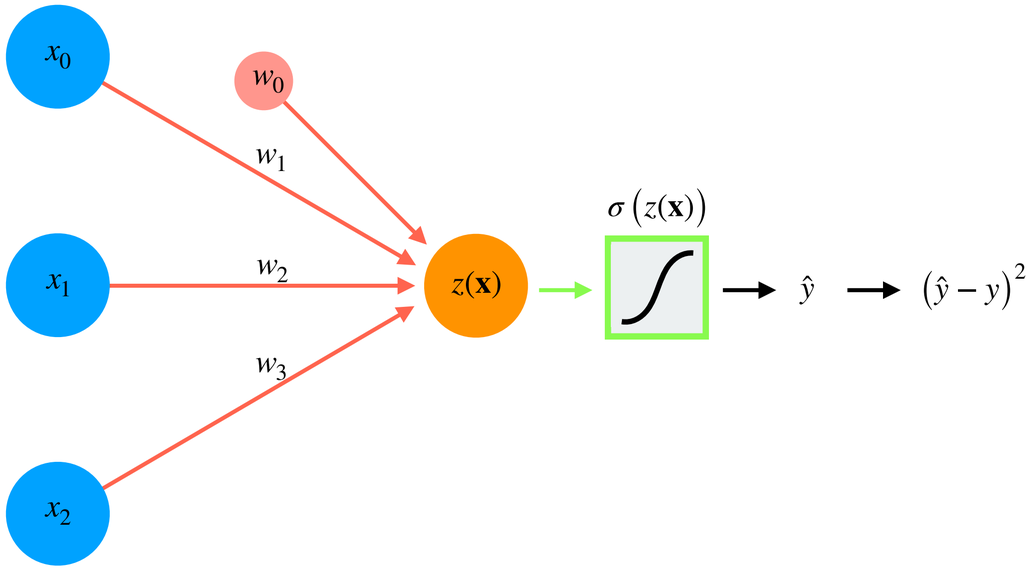
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

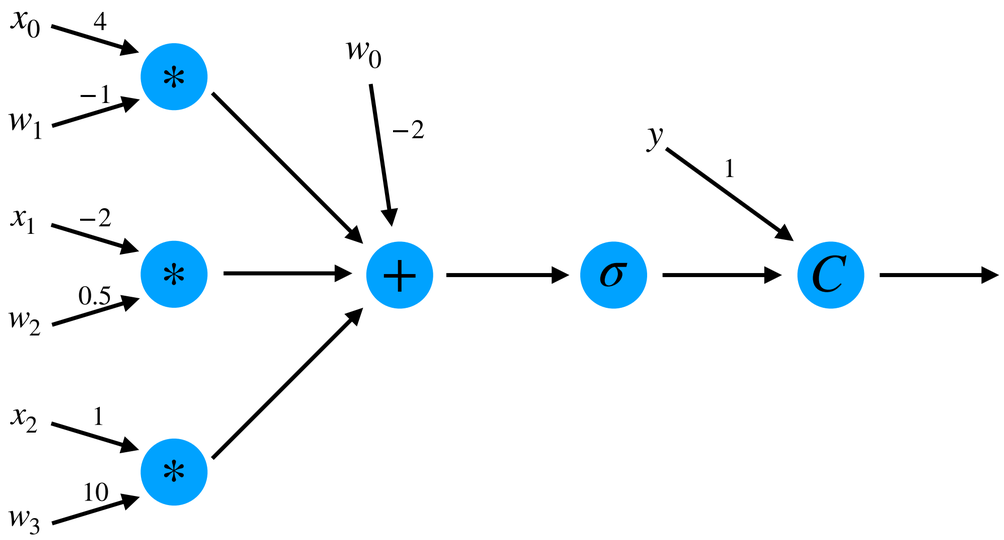
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Forward pass



Model:

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

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

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

Weights:

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

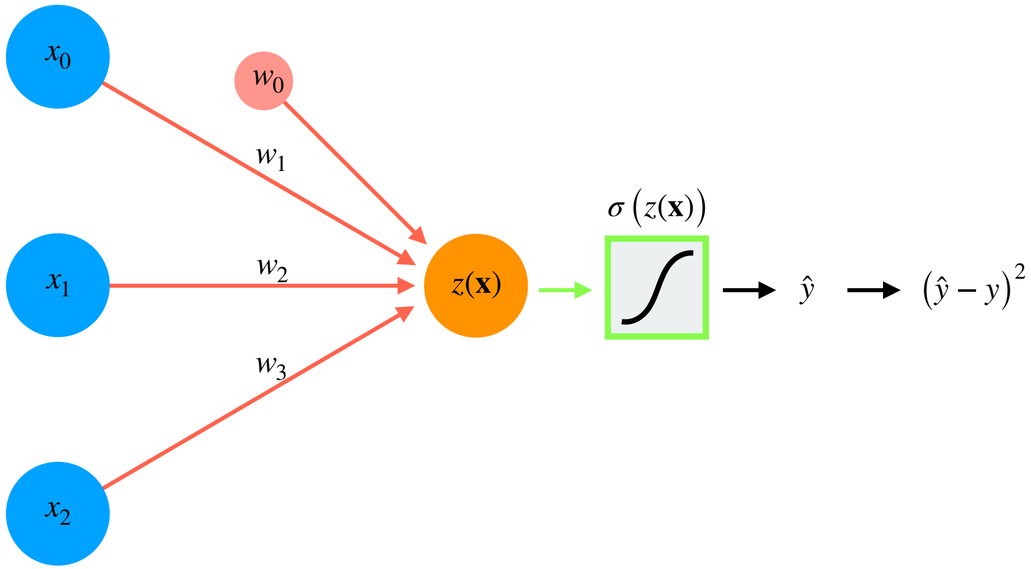
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

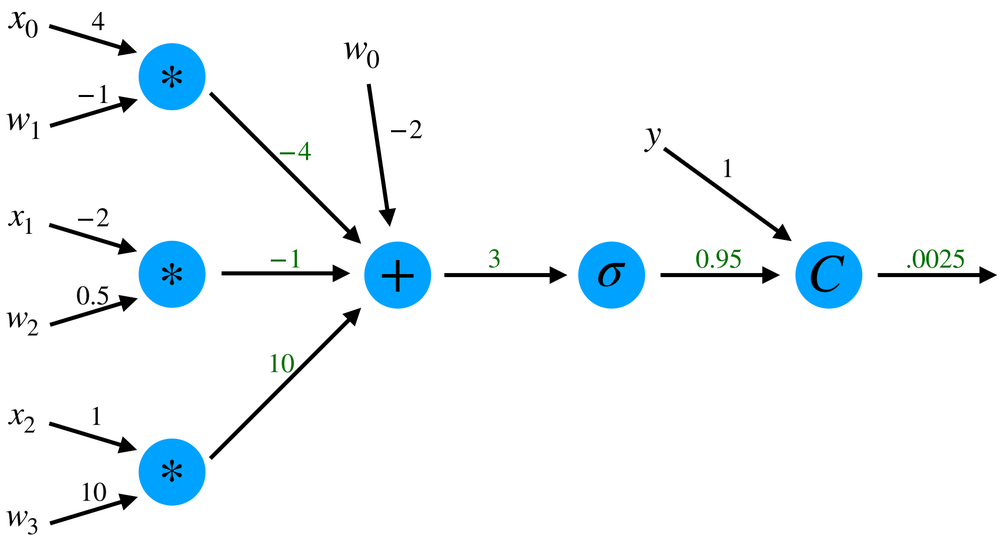
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Forward pass



Model:

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

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

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

Weights:

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

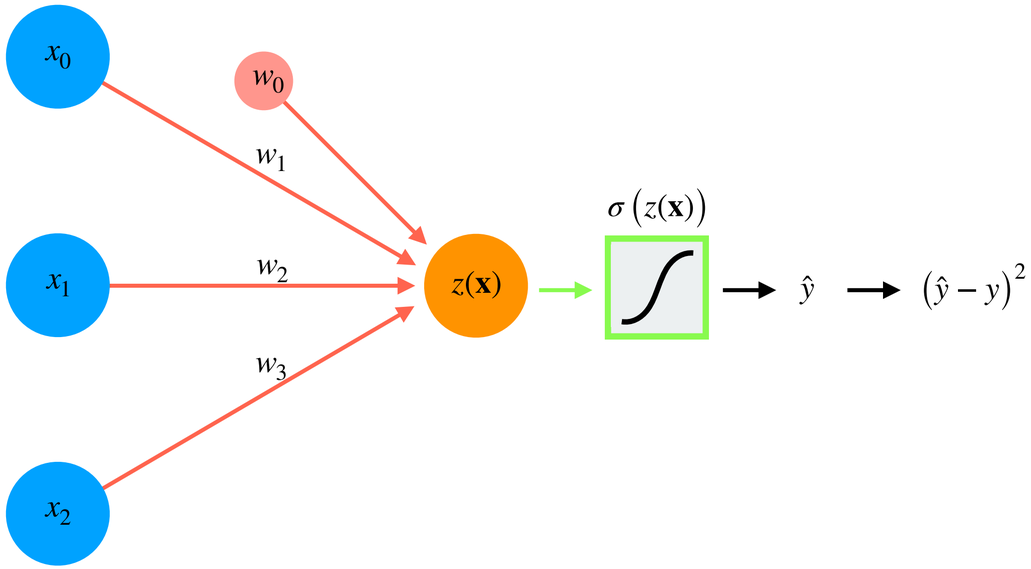
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

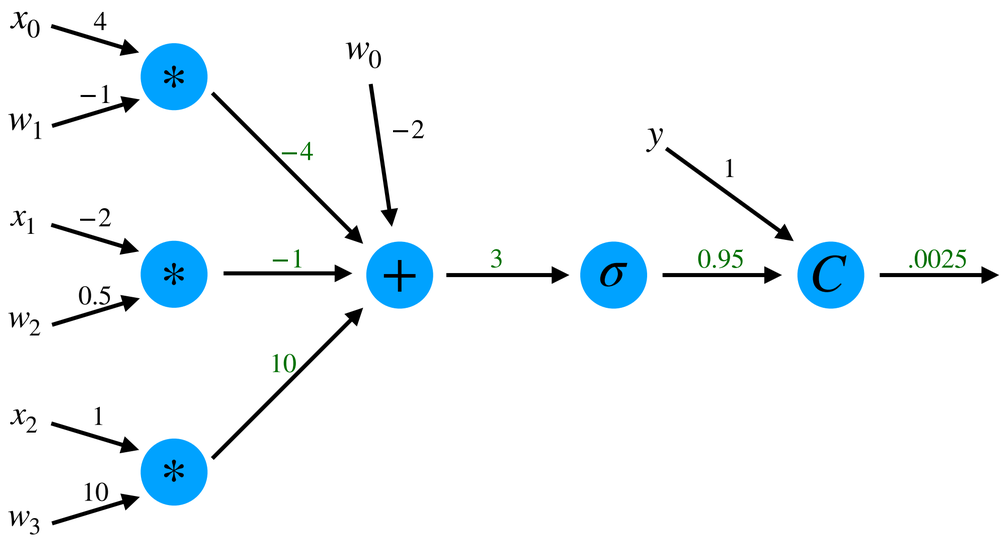
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

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

Weights:

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

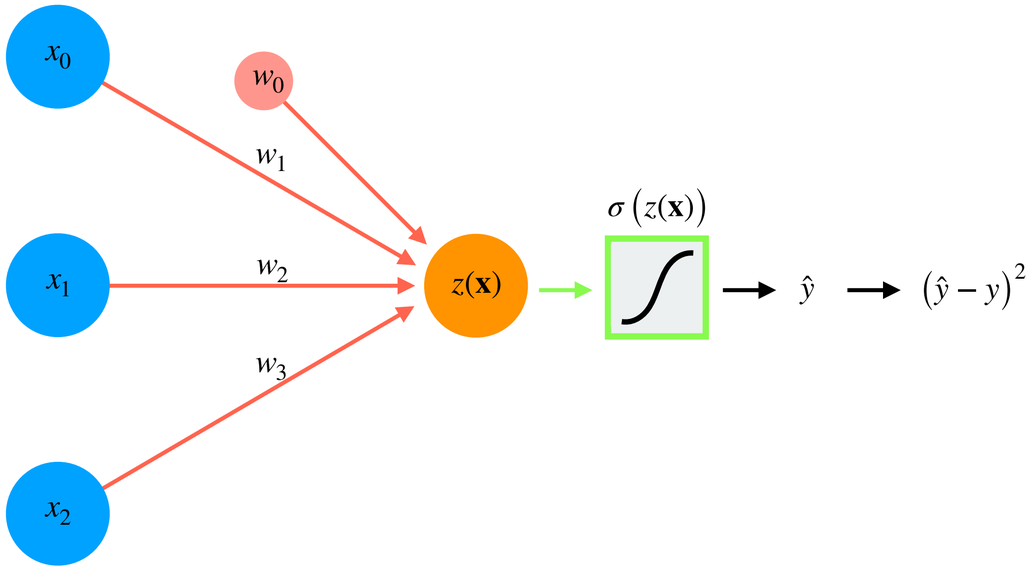
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

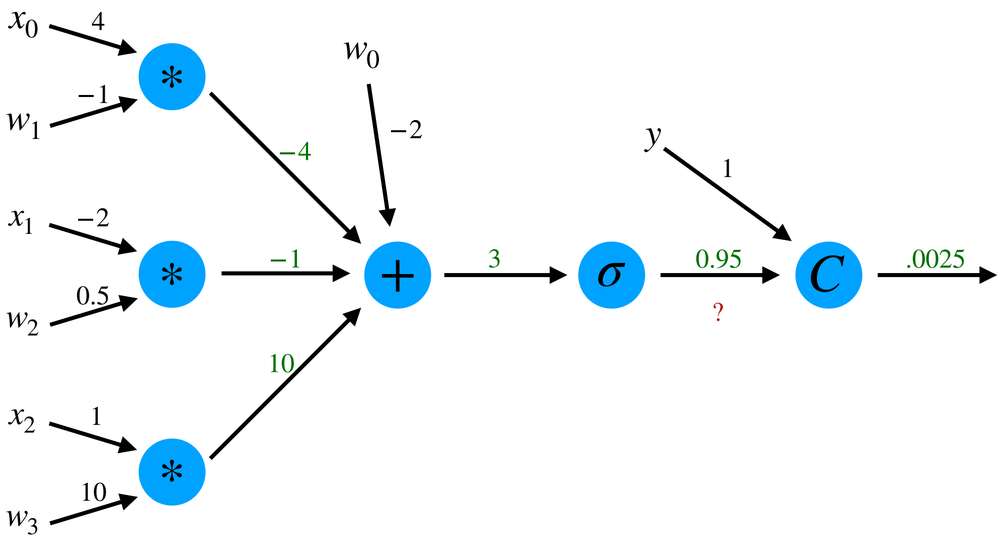
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

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

Weights:

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

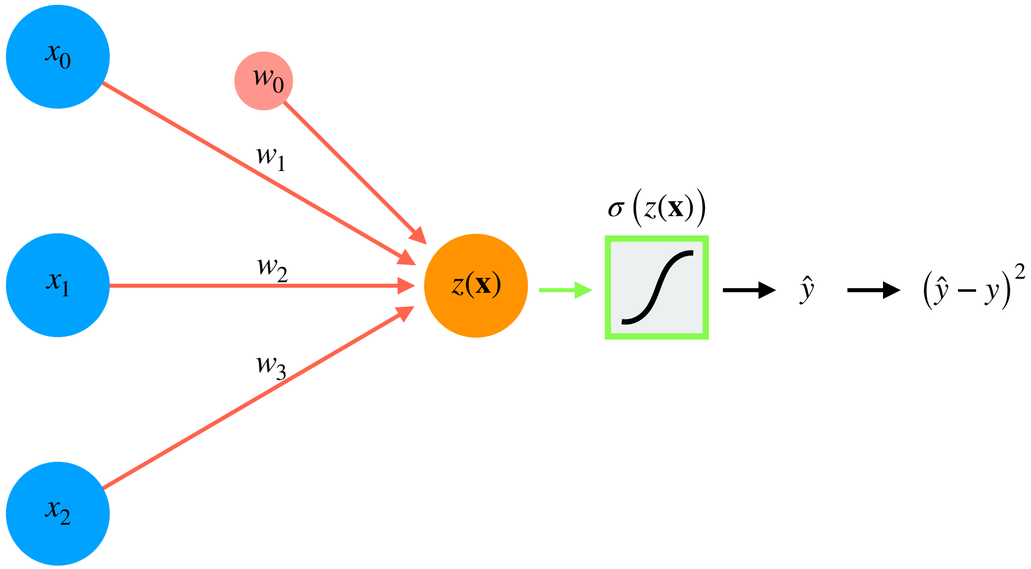
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

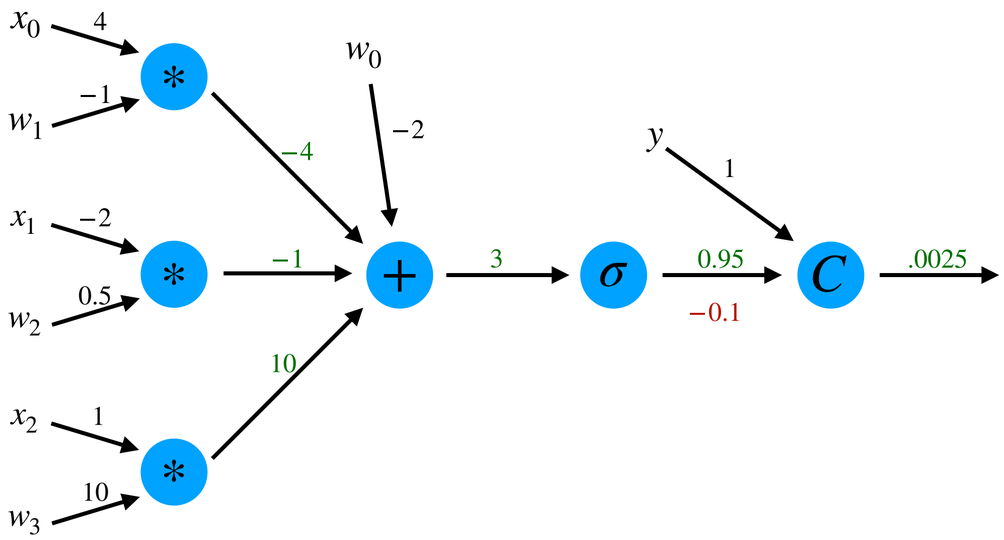
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

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

Weights:

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

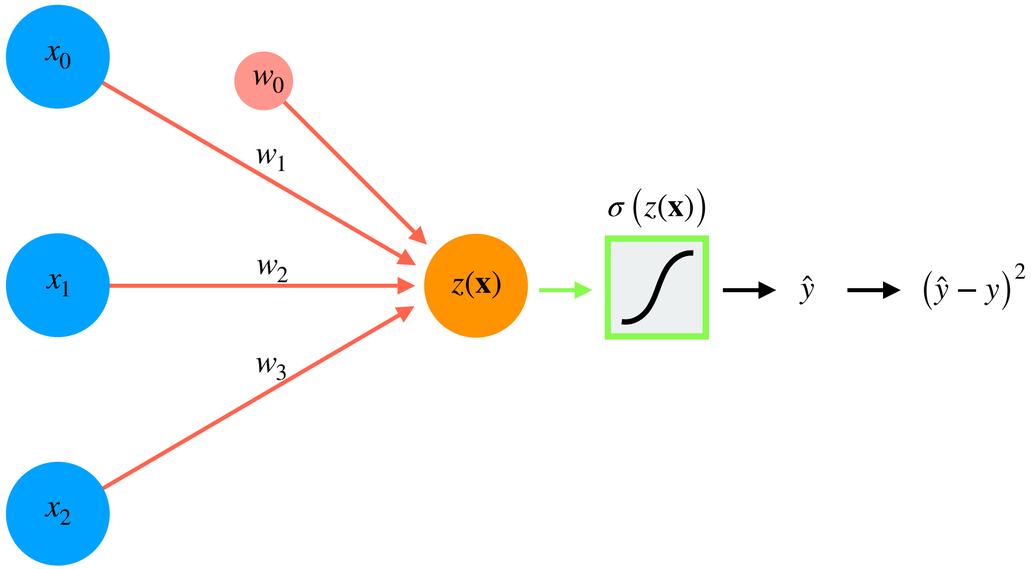
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

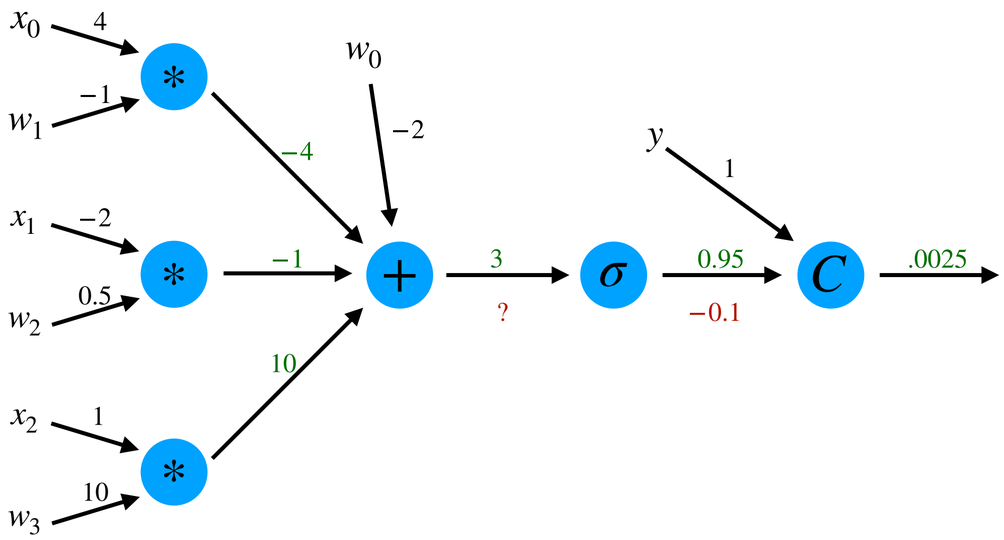
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

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

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

Weights:

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

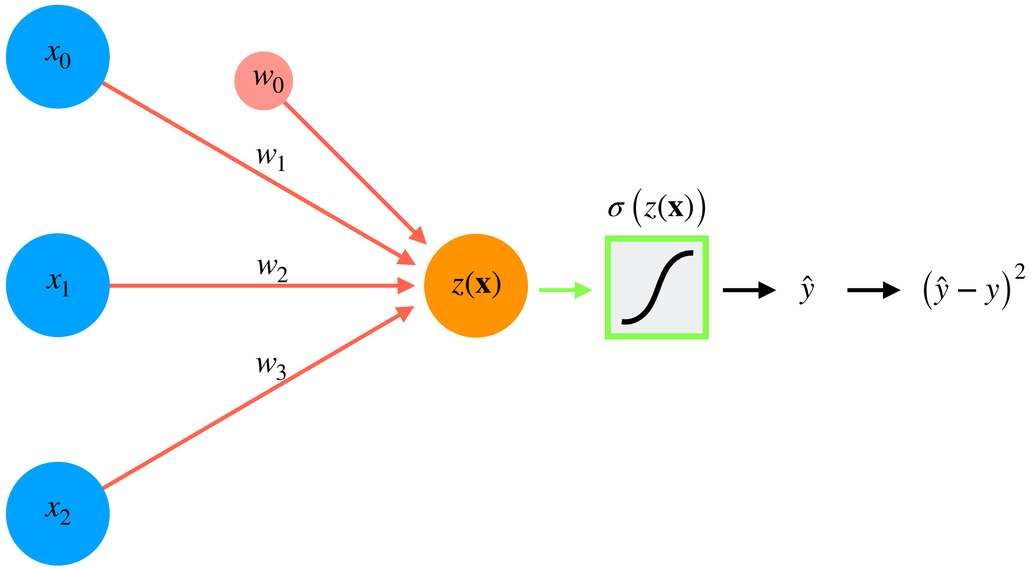
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

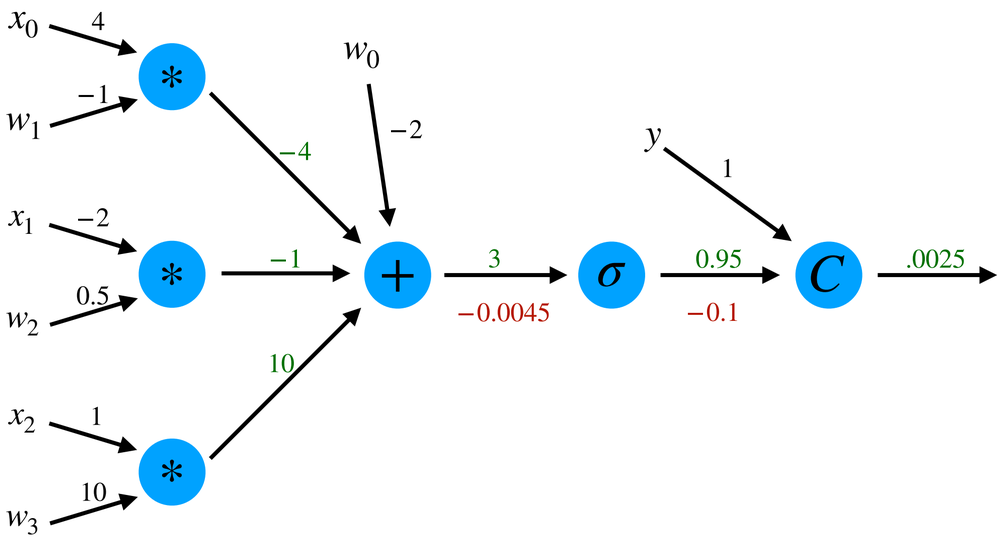
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is a function:

$$x' = 1, \quad (xw)' = w$$

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

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

Weights:

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

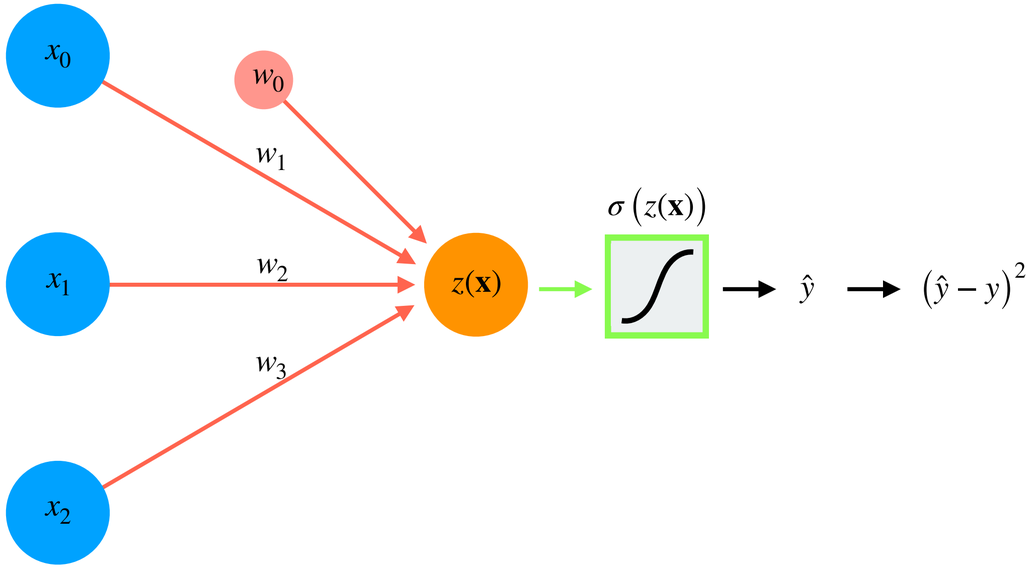
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

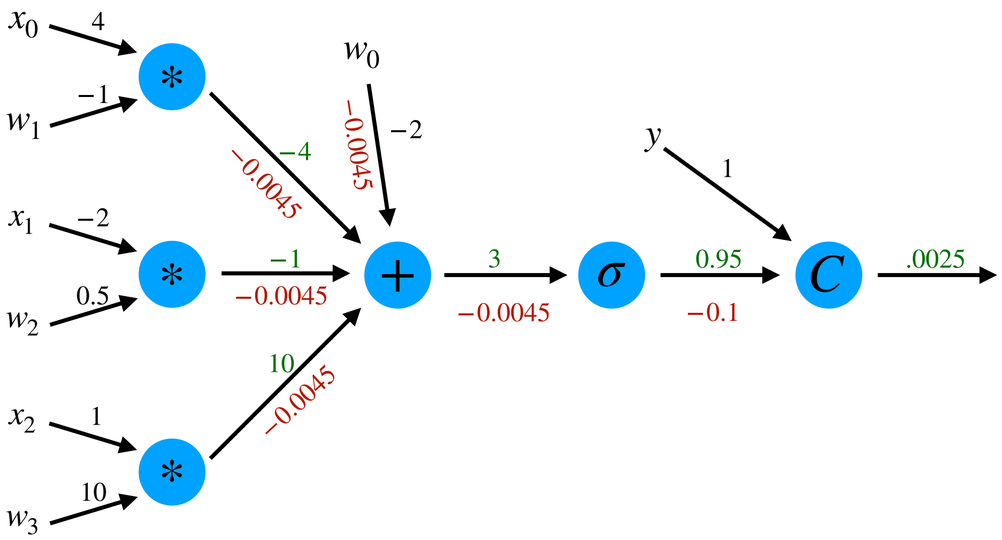
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is a function:

$$x' = 1, \quad (xw)' = w$$

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

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

Weights:

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

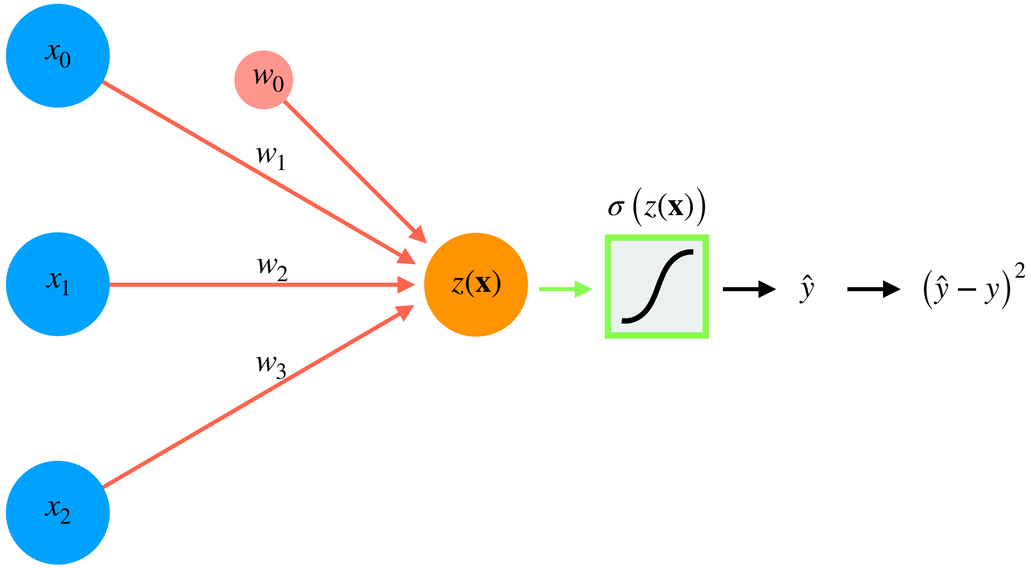
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

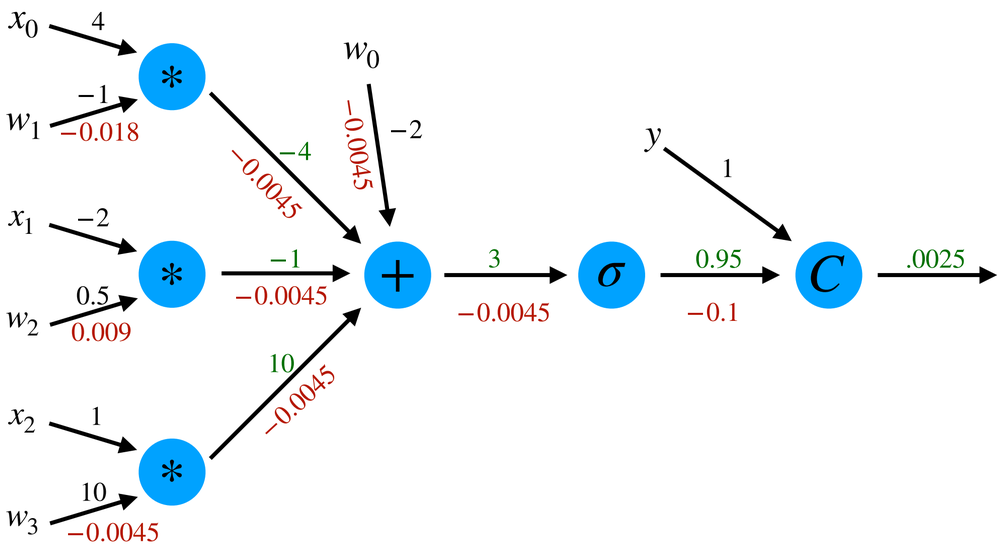
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is a function:

$$x' = 1, \quad (xw)' = w$$

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

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

Weights:

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

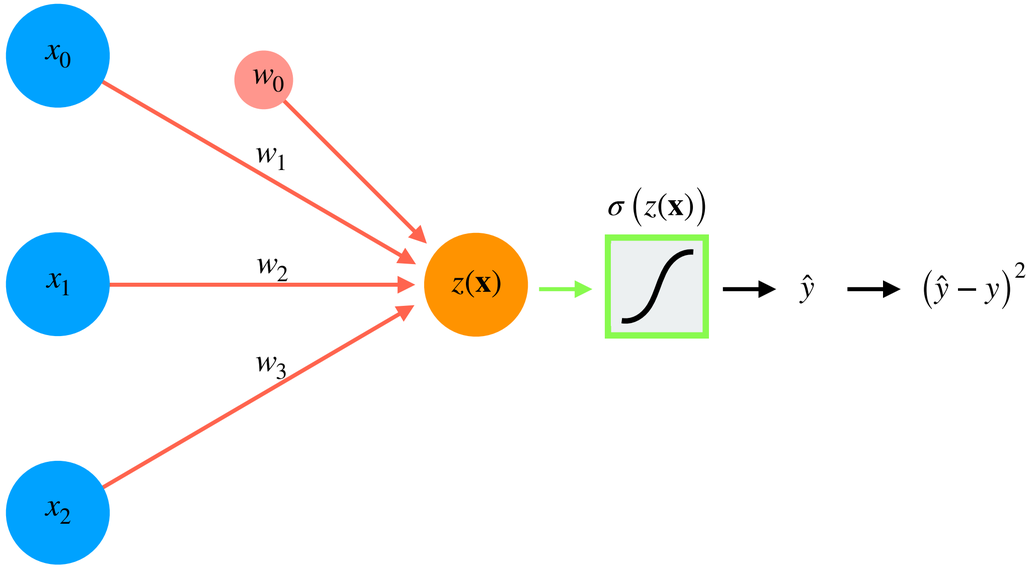
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

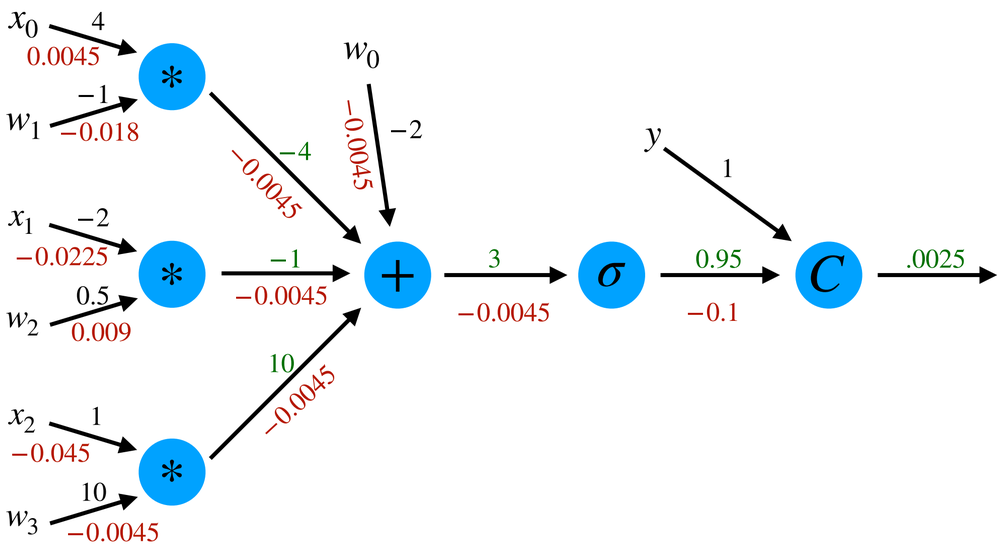
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is a function:

$$x' = 1, \quad (xw)' = w$$

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

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

Weights:

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

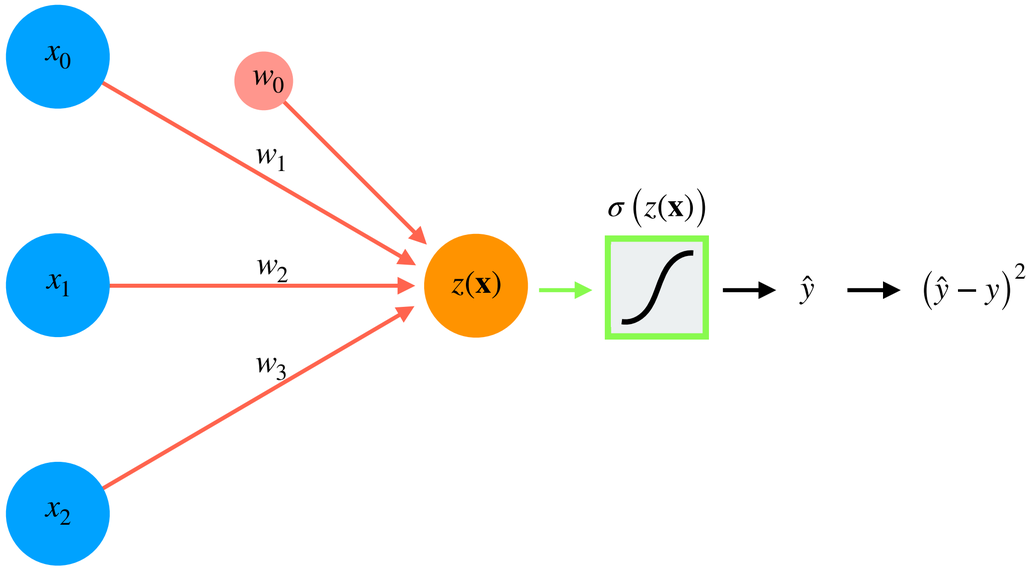
$$\mathbf{W} = \begin{bmatrix} -1 & 0.5 & 10 \end{bmatrix}$$

Data:

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

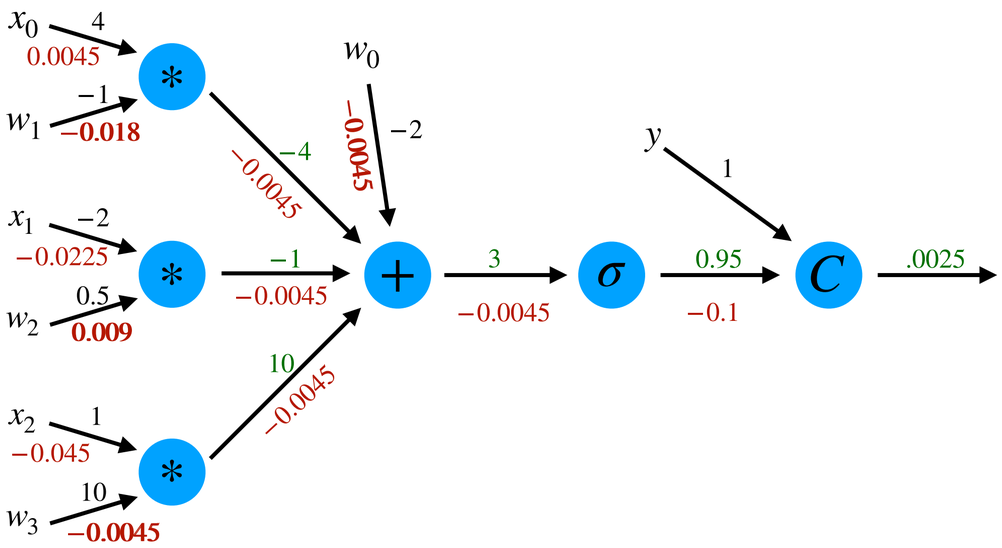
$$\mathbf{y} = \begin{bmatrix} 1 \end{bmatrix}$$

Backpropagation – on a neural network



As a computational graph:

Backward pass



Model:

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

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

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

Model derivatives:

The + gate is a function:

$$x' = 1, \ (xw)' = w$$

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

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

Weights:

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

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

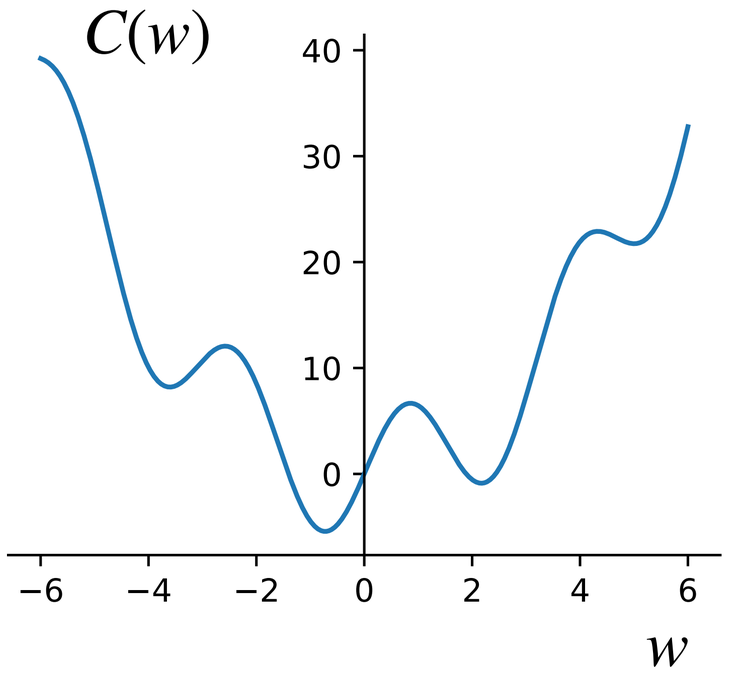
Data:

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

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

Stochastic Gradient Descent

- Gradient descent: Use the gradient of $C(\mathbf{W})$ evaluated on the entire dataset to find optimum $\mathbf{W}$:
  - Use entire dataset $\mathscr{D} = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{N}$
  - Compute predictions and compare to ground truth $C(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}}(\hat{y}_i - y_i)^2$
  - Find gradient $\nabla C(\mathbf{W}_t)$
  - Update weights $\mathbf{W}_{t+1} = \mathbf{W}_t + \eta\left(-\nabla C(\mathbf{W}_t)\right)$

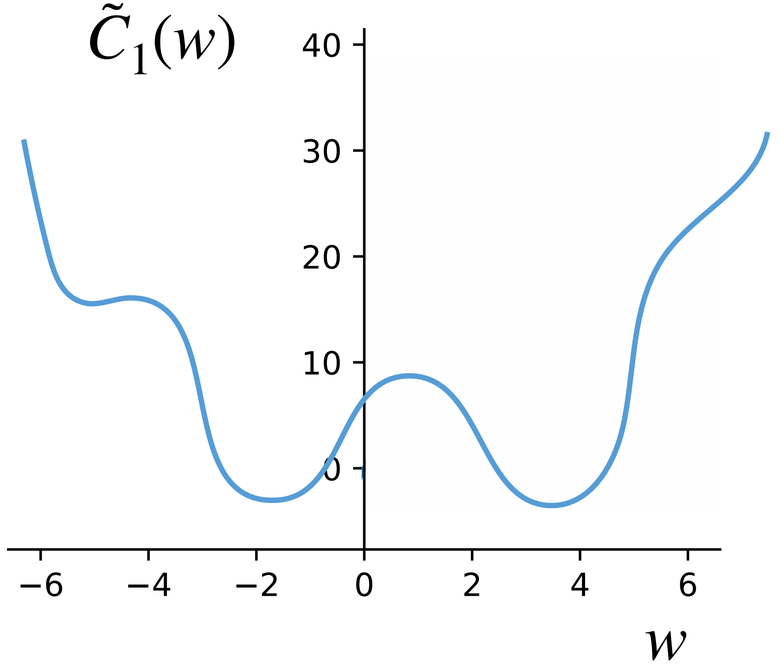


Stochastic Gradient Descent

- Gradient descent: Use the gradient of $C(\mathbf{W})$ evaluated on the entire dataset to find optimum $\mathbf{W}$:
  - Use entire dataset $\mathscr{D} = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{N}$
  - Compute predictions and compare to ground truth $C(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$
  - Find gradient $\nabla C(\mathbf{W}_t)$
  - Update weights $\mathbf{W}_{t+1} = \mathbf{W}_t + \eta\left(-\nabla C(\mathbf{W}_t)\right)$

- Stochastic gradient descent: Same as above, evaluating the cost on **random subsets of the data**
  - Use subset of data $\mathscr{S}_1 = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{M}\ ,\ \mathscr{S}_1 \subset \mathscr{D}$

$$\longrightarrow \tilde{C}_1(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$$

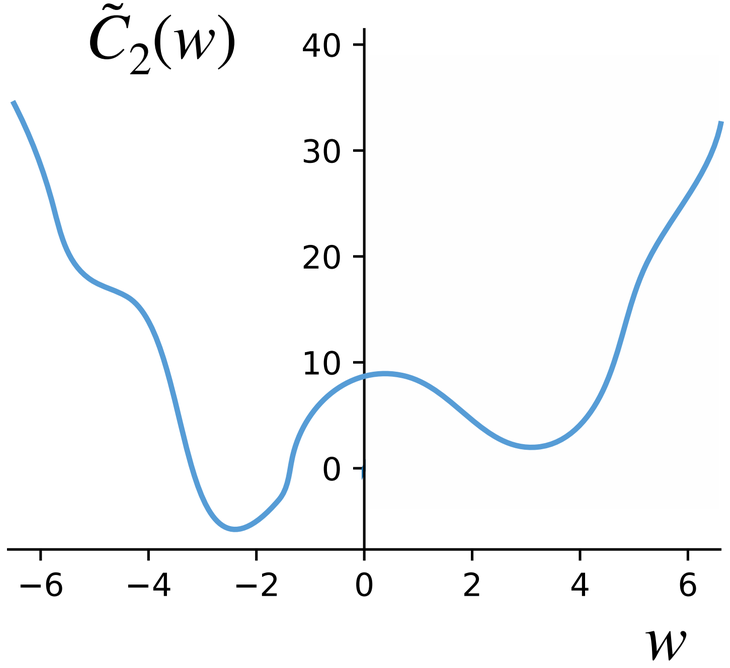


Stochastic Gradient Descent

- Gradient descent: Use the gradient of $C(\mathbf{W})$ evaluated on the entire dataset to find optimum $\mathbf{W}$:
  - Use entire dataset $\mathscr{D} = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{N}$
  - Compute predictions and compare to ground truth $C(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$
  - Find gradient $\nabla C(\mathbf{W}_t)$
  - Update weights $\mathbf{W}_{t+1} = \mathbf{W}_t + \eta\left(-\nabla C(\mathbf{W}_t)\right)$

- Stochastic gradient descent: Same as above, evaluating the cost on **random subsets of the data**
  - Use subset of data $\mathscr{S}_2 = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{M}$, $\mathscr{S}_2 \subset \mathscr{D}$

$$\longrightarrow \tilde{C}_2(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$$

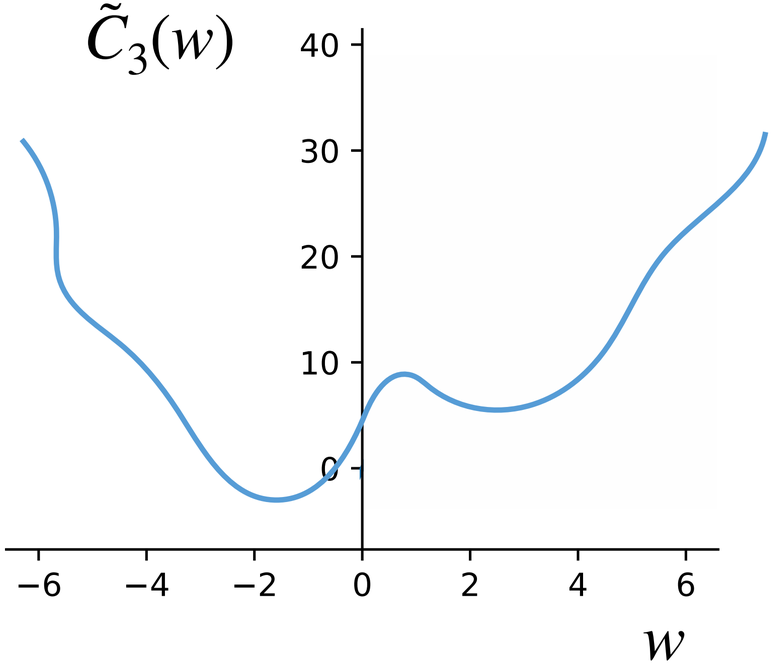


Stochastic Gradient Descent

- Gradient descent: Use the gradient of $C(\mathbf{W})$ evaluated on the entire dataset to find optimum $\mathbf{W}$:
  - Use entire dataset $\mathscr{D} = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{N}$
  - Compute predictions and compare to ground truth $C(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$
  - Find gradient $\nabla C(\mathbf{W}_t)$
  - Update weights $\mathbf{W}_{t+1} = \mathbf{W}_t + \eta\left(-\nabla C(\mathbf{W}_t)\right)$

- Stochastic gradient descent: Same as above, evaluating the cost on **random subsets of the data**
  - Use subset of data $\mathscr{S}_3 = \left\{\mathbf{X_i}, y_i\right\}_{i=1}^{M}\ ,\ \mathscr{S}_3 \subset \mathscr{D}$

$$\longrightarrow \tilde{C}_3(\mathbf{W}_t) = \frac{1}{N}\sum_{i\in\mathscr{D}} (\hat{y}_i - y_i)^2$$



- Converges to the result of regular gradient descent
- Very parallelizable (modern networks don't train without some form of SGD)

Gradient Descent vs. Stochastic Gradient Descent