Artificial Neural Networks

Linear Score Functions?

- Can we make linear regression better?
  - Multiply with another weight matrix $\boldsymbol{W_2}$

$$\hat{\boldsymbol{f}} = \boldsymbol{W_2} \cdot \boldsymbol{f}$$
$$\hat{\boldsymbol{f}} = \boldsymbol{W_2} \cdot \boldsymbol{W} \cdot \boldsymbol{x}$$

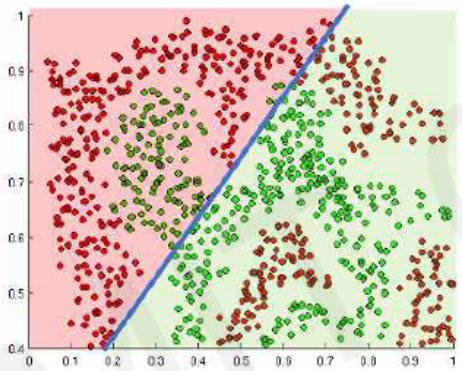
- Operation is still linear.

$$\widehat{\boldsymbol{W}} = \boldsymbol{W_2} \cdot \boldsymbol{W}$$
$$\hat{\boldsymbol{f}} = \widehat{\boldsymbol{W}}\, \boldsymbol{x}$$

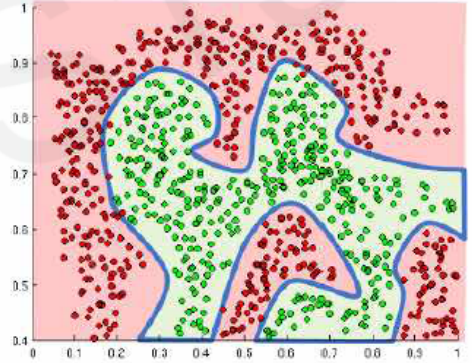
- Solution → add non-linearity!!

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

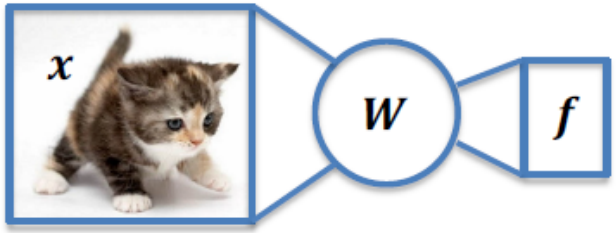
Neural Network

- Linear score function $\boldsymbol{f = Wx}$

- Neural network is a nesting of 'functions'
  - 2-layers: $\boldsymbol{f = W_2 \max(0, W_1 x)}$
  - 3-layers: $\boldsymbol{f = W_3 \max(0, W_2 \max(0, W_1 x))}$
  - 4-layers: $\boldsymbol{f = W_4 \tanh\,(W_3, \max(0, W_2 \max(0, W_1 x)))}$
  - 5-layers: $\boldsymbol{f = W_5 \sigma(W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1 x))))}$
  - … up to hundreds of layers

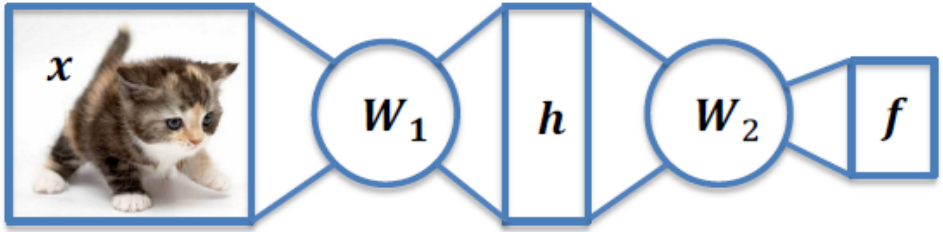
Neural Network

1-layer network: $f = Wx$



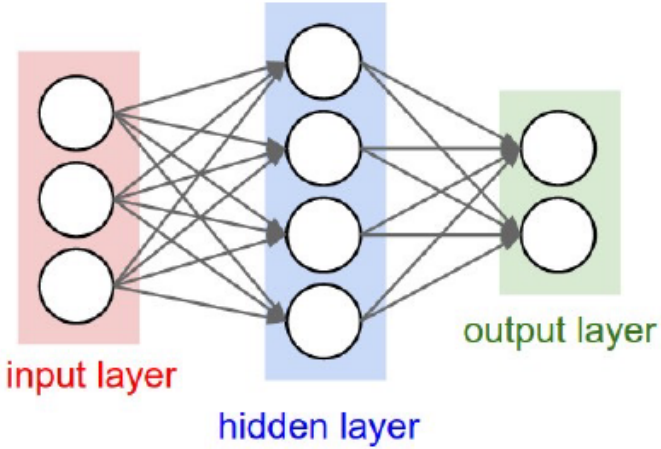
128×128 = 16384 10

2-layer network: $f = W_2 \max(0, W_1 x)$

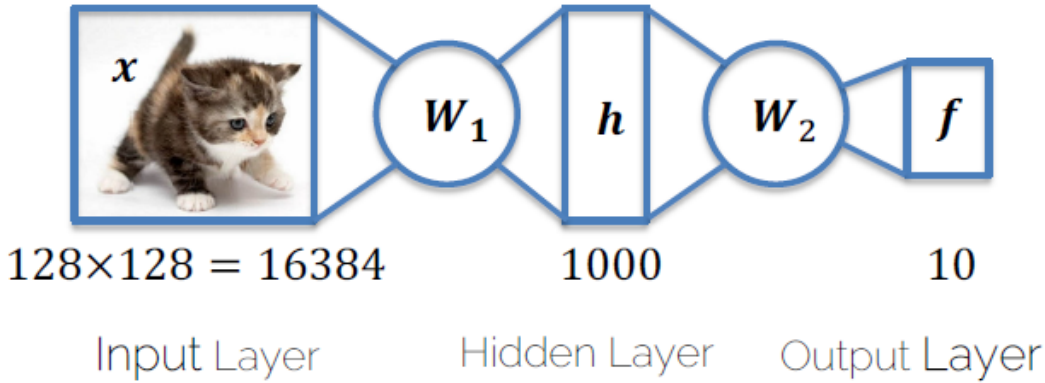


128×128 = 16384 1000 10

Neural Network



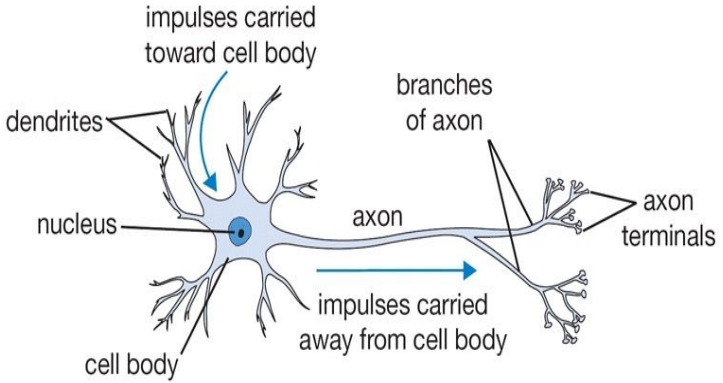
2-layer network: $f = W_2 \max(0, W_1 x)$



Biological Neurons

- The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14} - 10^{15} synapses.

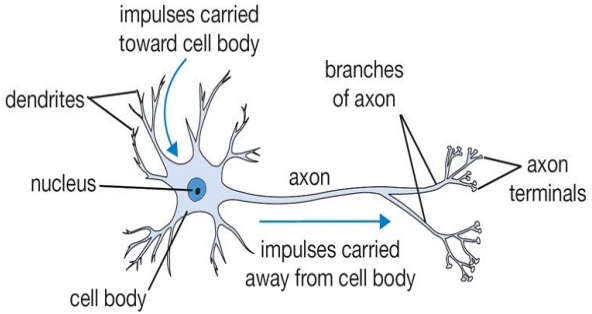
- Each neuron receives input signals from its dendrites and produces output signals along its (single) axon.



- The axon eventually branches out and connects via synapses to dendrites of other neurons.
- In the computational model of a neuron, the signals that travel along the axons (e.g x_0)interact multiplicatively(e.g w_0x_0)with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g w_0).

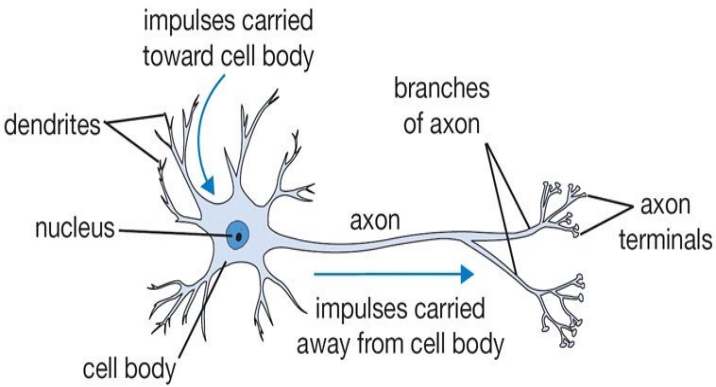
Biological Neurons

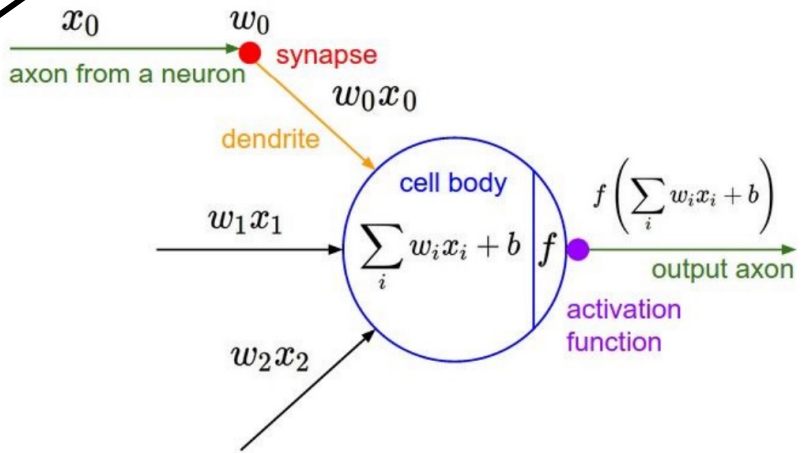
- The idea is that the synaptic strengths are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another.



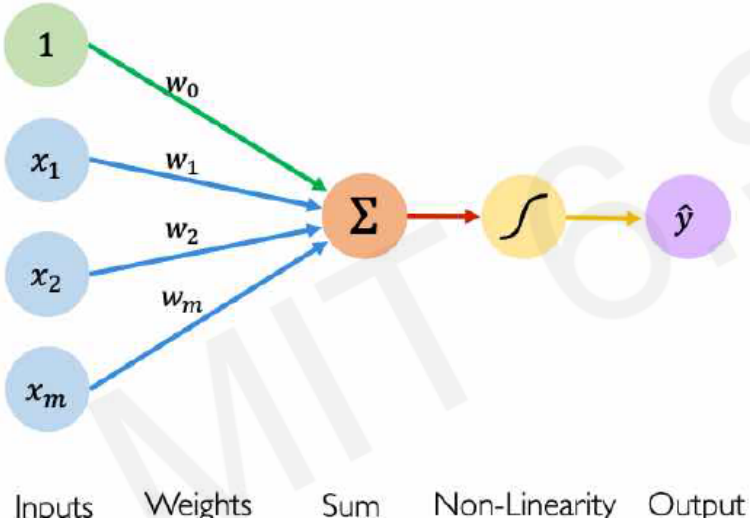
- Then dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon.

Biological Neurons





The Perceptron: Forward Propagation



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

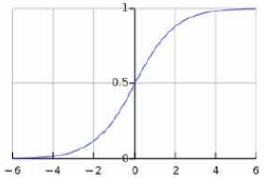
where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Activation Functions

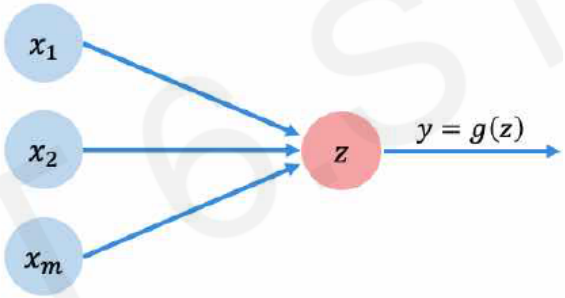
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

- Example: sigmoid function

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



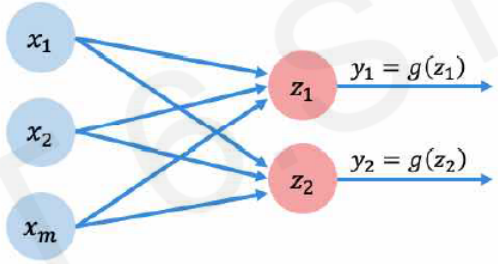
The Perceptron: Simplified



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

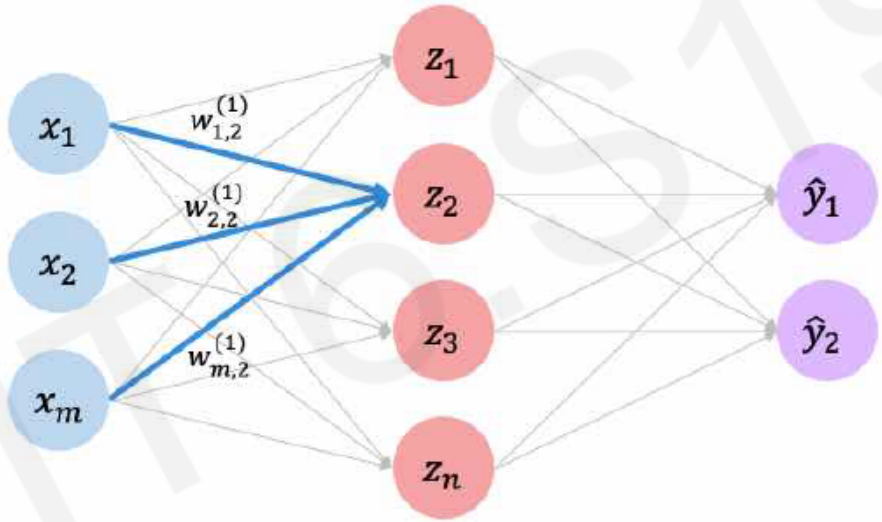
Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

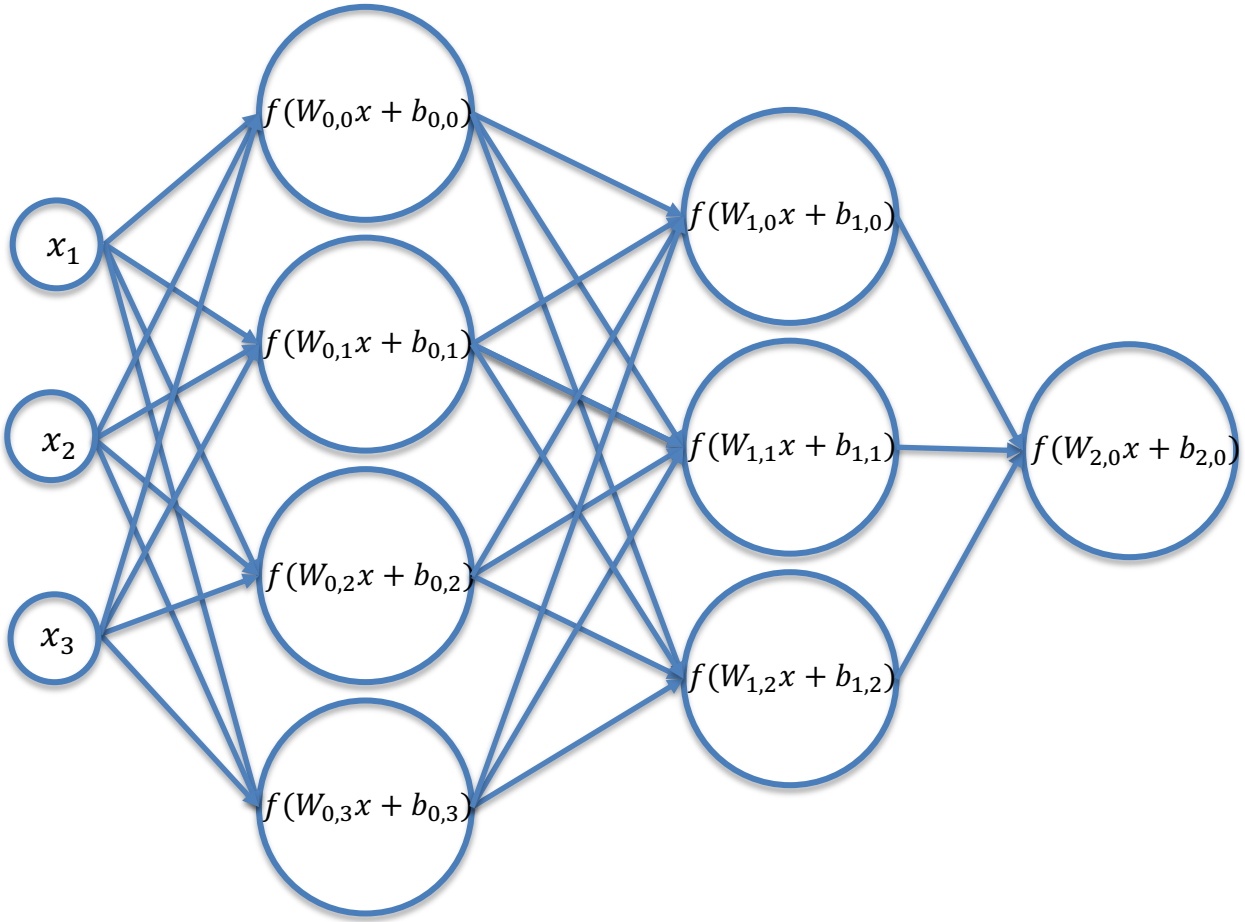
Single Layer Neural Network



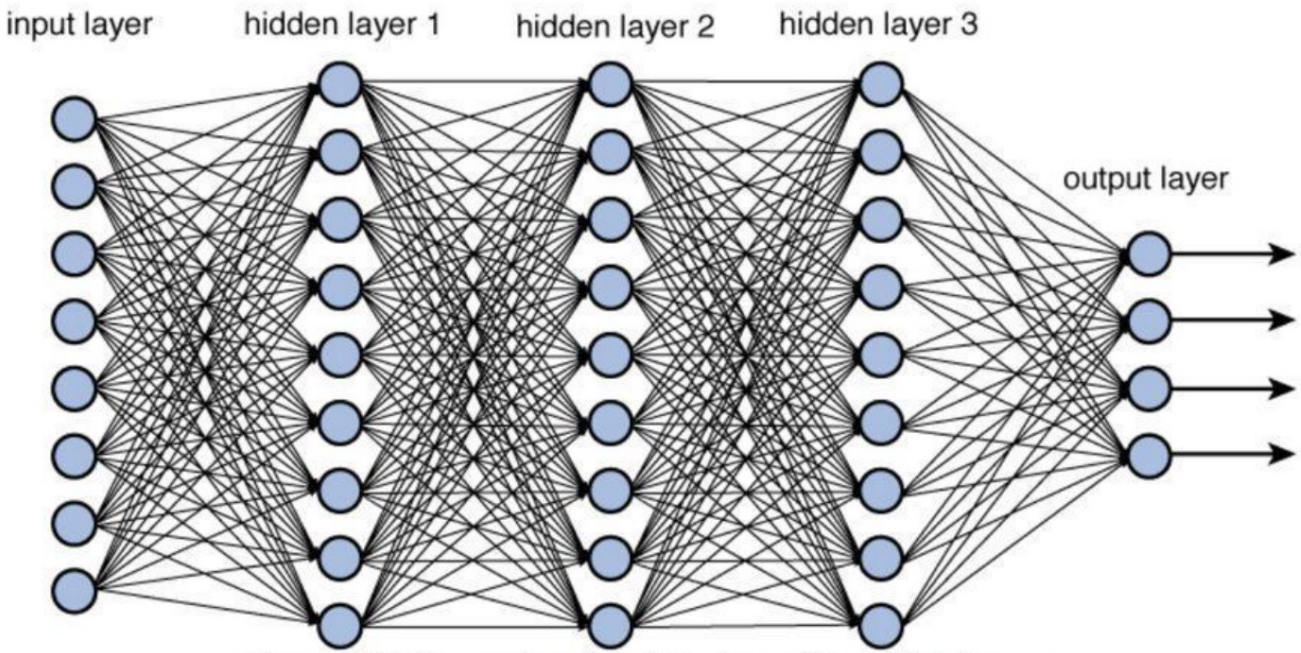
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Net of Artificial Neurons



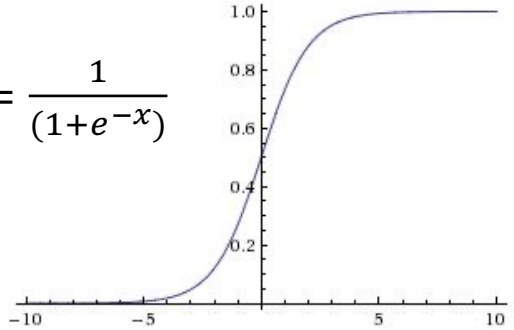
Neural Network



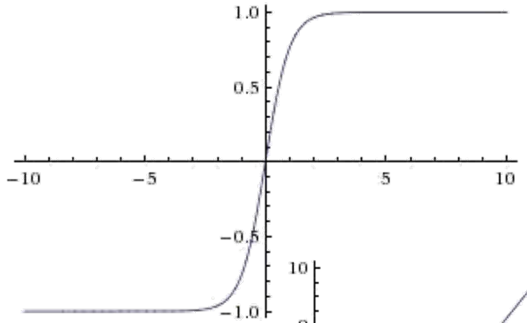
Source: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Activation Functions

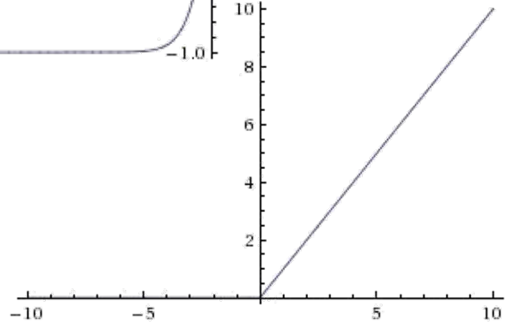
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



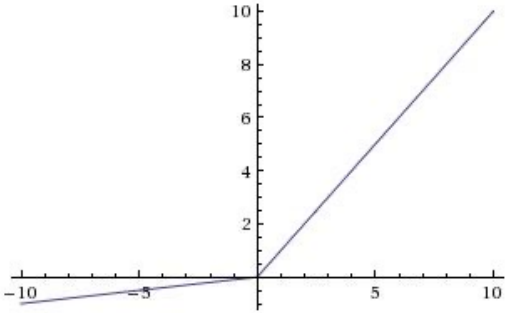
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



Leaky ReLU: $\max(0.1x, x)$



Parametric ReLU: $\max(\alpha x, x)$

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

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$

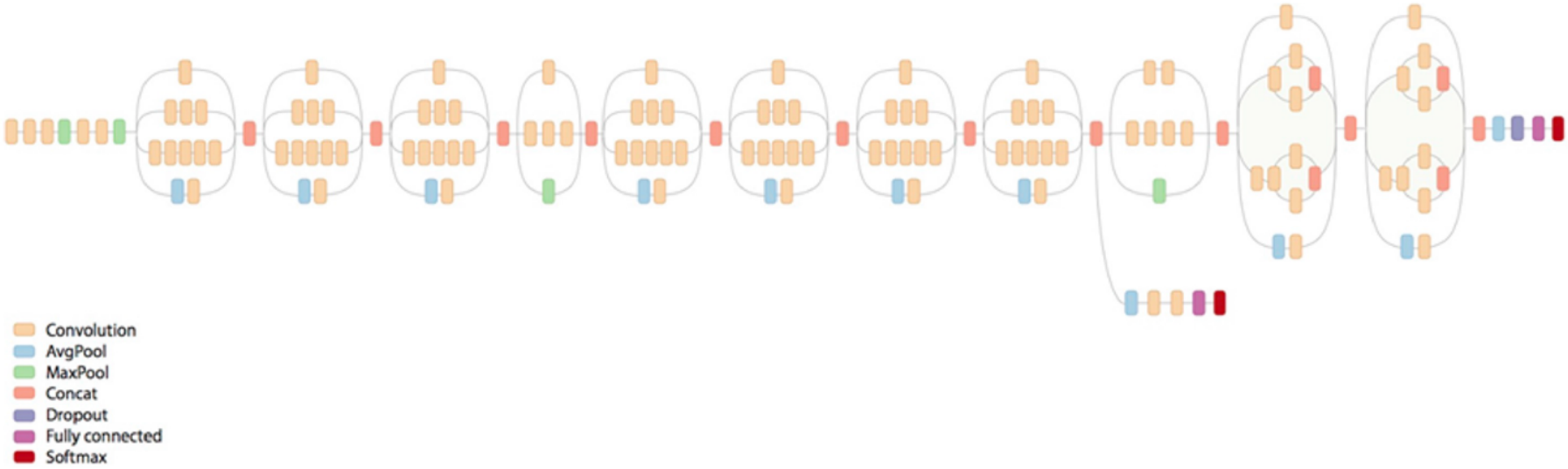
Neural Network

- Summary
  - Given a dataset with ground truth training pairs $[x_i; y_i]$,
  - Find optimal weights $\boldsymbol{W}$ using stochastic gradient descent, such that the loss function is minimized
    - Compute gradients with **back-propagation**
    - Iterate many times over training set (SGD; more later)

Backpropagation

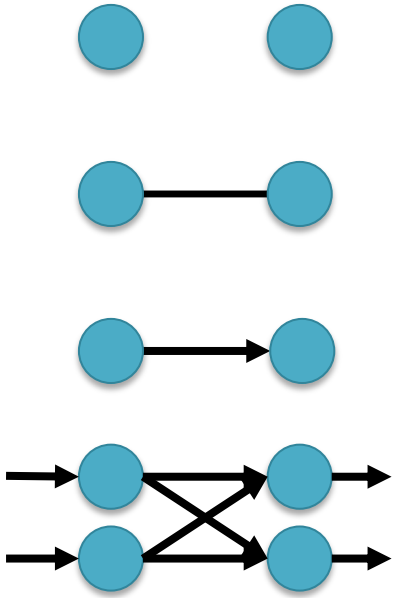
NNs can Become Quite Complex…

- These graphs can be huge!



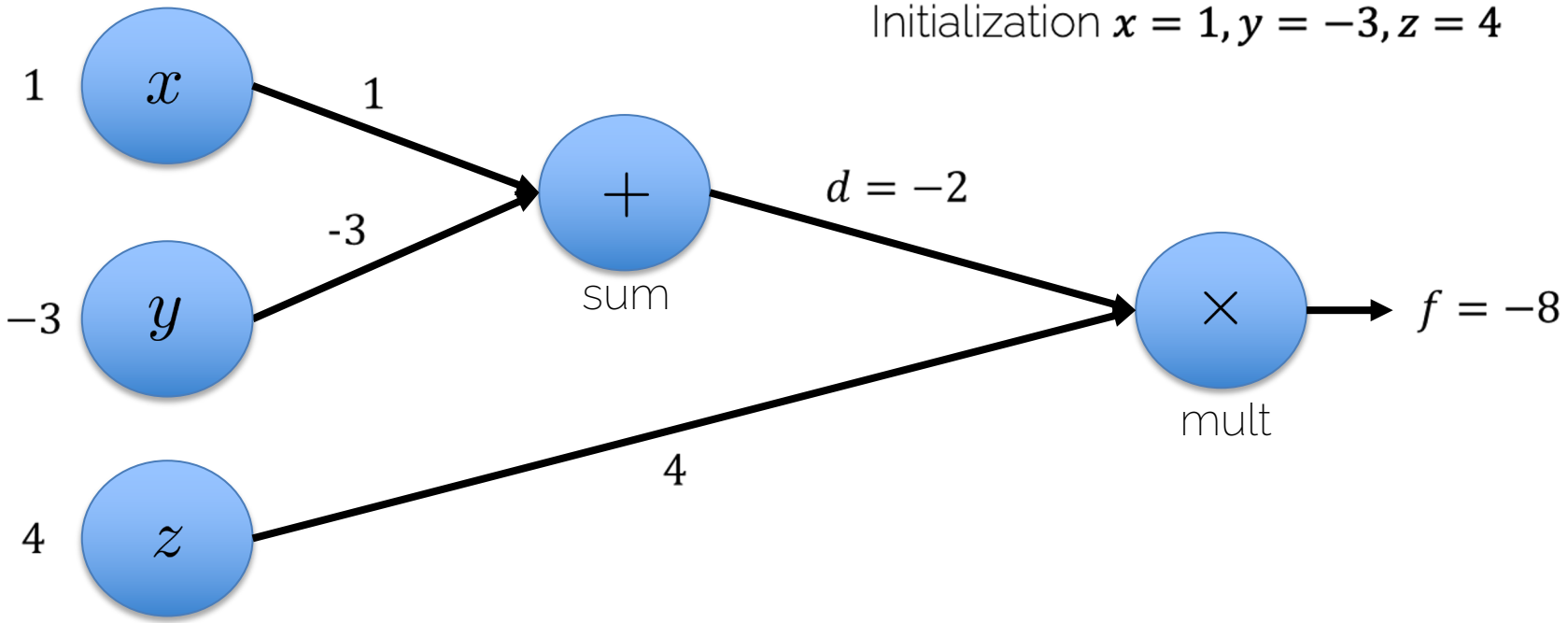
Computational Graphs

- Neural network is a computational graph
  - It has compute nodes
  - It has edges that connect nodes
  - It is directional
  - It is organized in ‘layers’



Backprop: Forward Pass

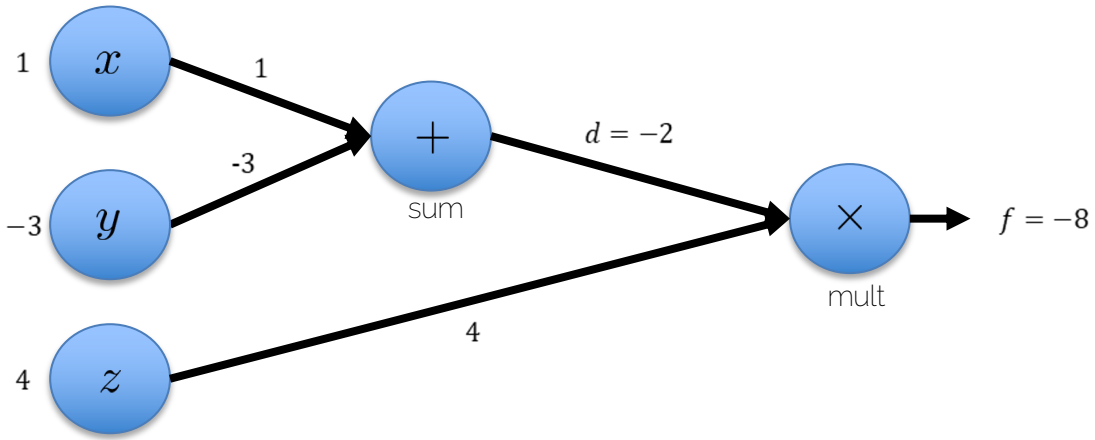
- $f(x, y, z) = (x + y) \cdot z$



Backprop: Backward Pass

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

with $x = 1, y = -3, z = 4$



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

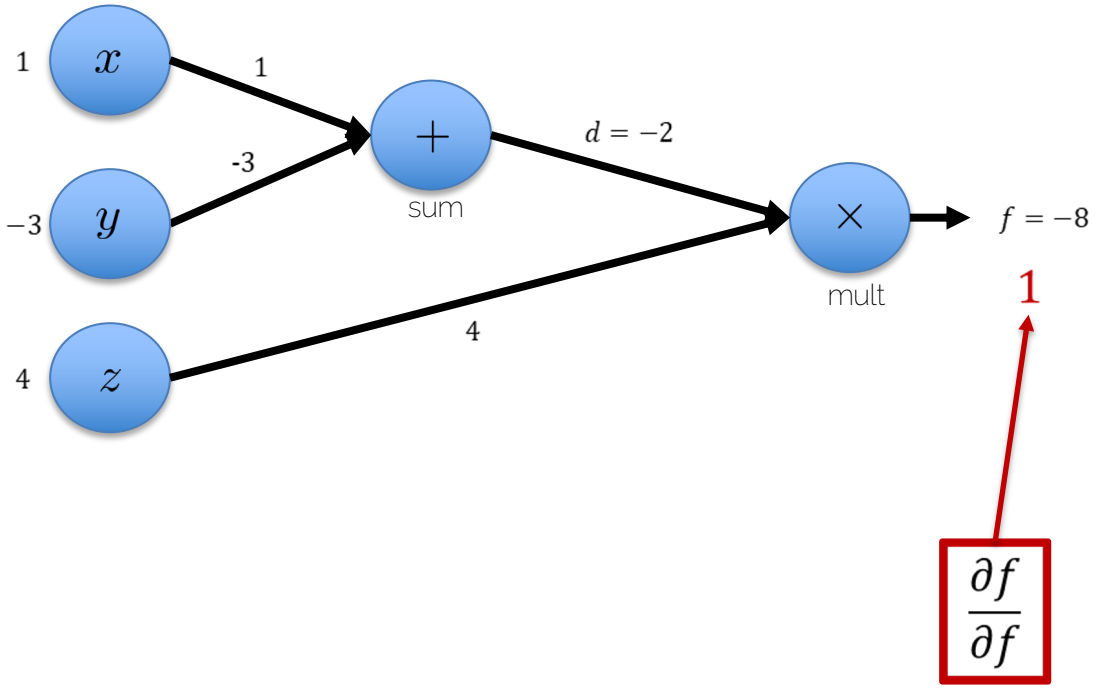
$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Backprop: Backward Pass

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

with $x = 1, y = -3, z = 4$



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

$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Backprop: Backward Pass

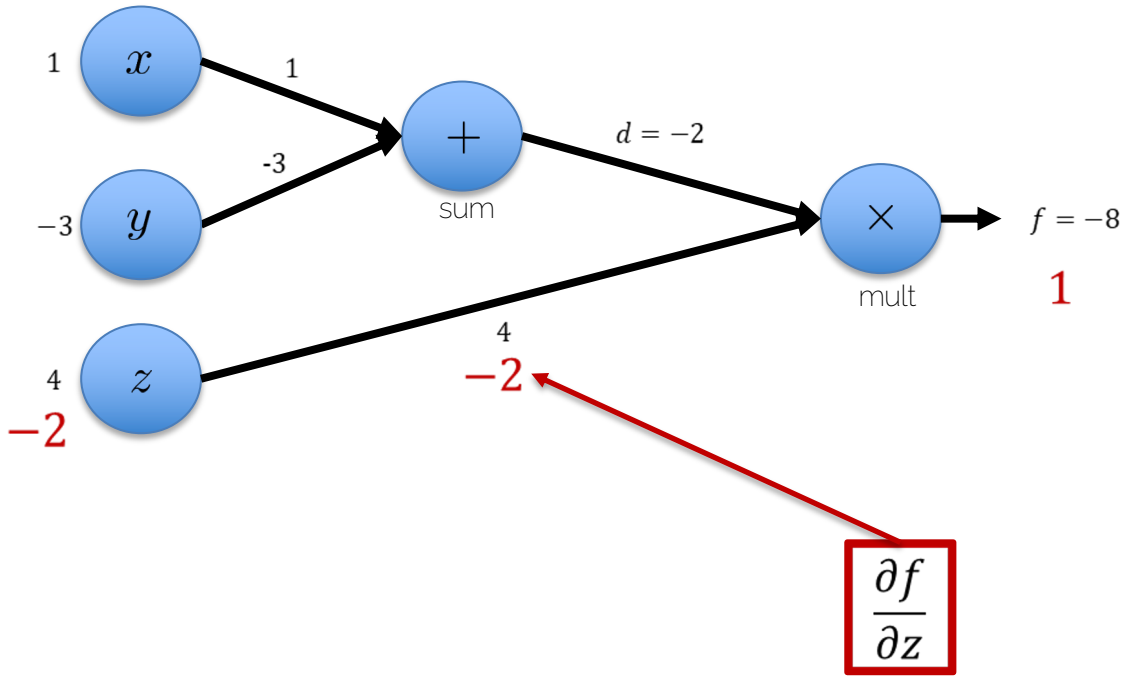
$f(x, y, z) = (x + y) \cdot z$

with $x = 1, y = -3, z = 4$

$d = x + y$ $\quad \frac{\partial d}{\partial x} = 1, \frac{\partial d}{\partial y} = 1$

$f = d \cdot z$ $\quad \frac{\partial f}{\partial d} = z, \boxed{\frac{\partial f}{\partial z} = d}$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?



Backprop: Backward Pass

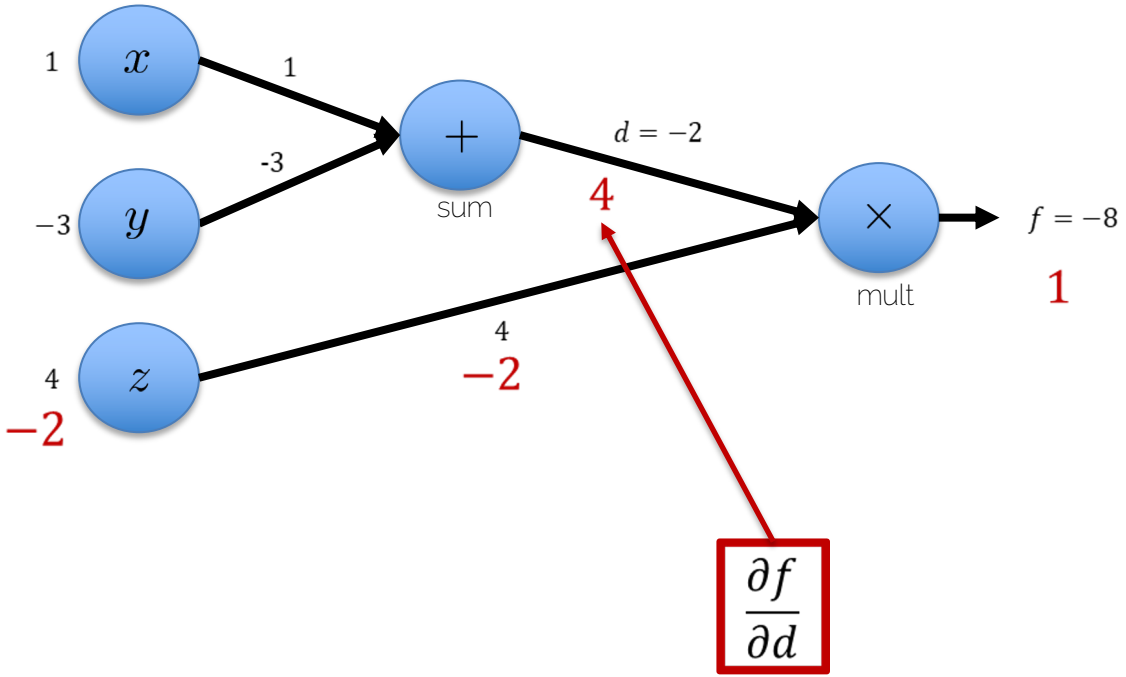
$$f(x, y, z) = (x + y) \cdot z$$

with $x = 1, y = -3, z = 4$

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

$$f = d \cdot z \qquad \boxed{\frac{\partial f}{\partial d} = z}, \frac{\partial f}{\partial z} = d$$

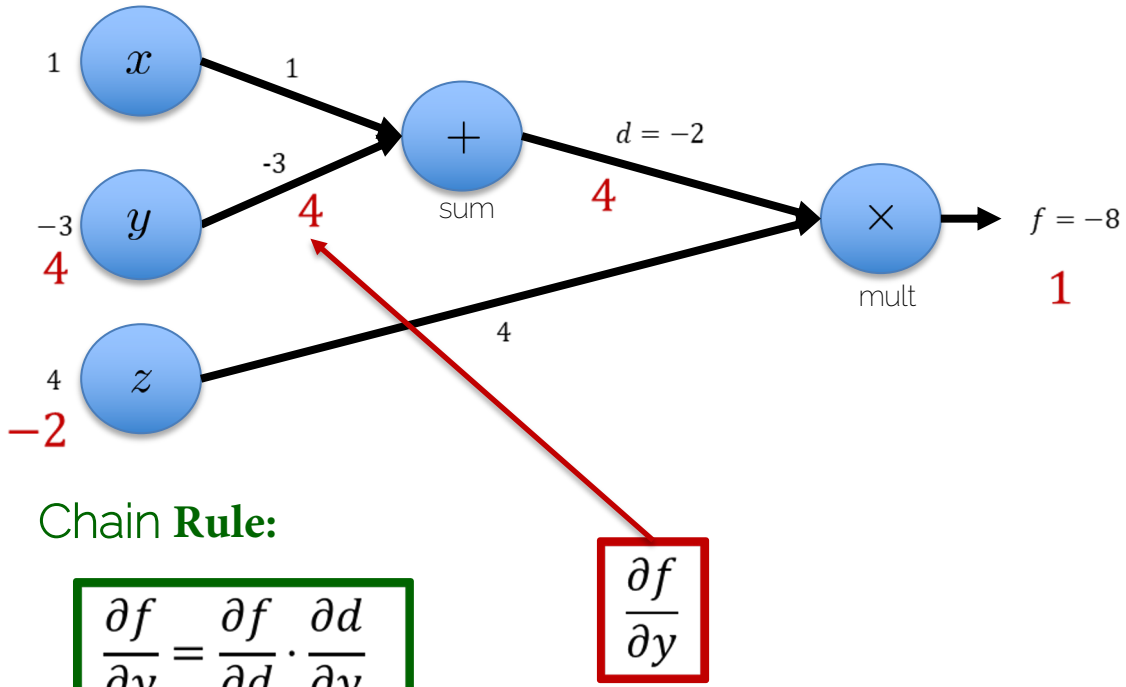
What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?



Backprop: Backward Pass

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

with $x = 1, y = -3, z = 4$



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

$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Chain **Rule:**

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial y}$$

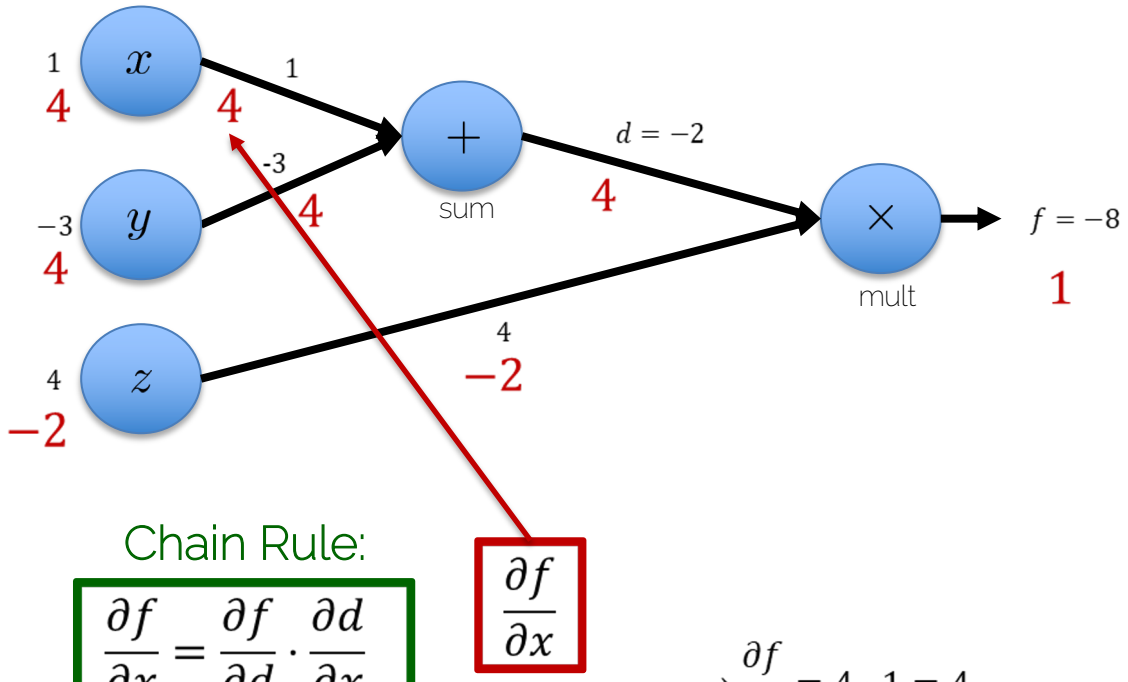
$$\rightarrow \frac{\partial f}{\partial y} = 4 \cdot 1 = 4$$

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

Backprop: Backward Pass

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

with $x = 1, y = -3, z = 4$



$d = x + y$ $\quad \frac{\partial d}{\partial x} = 1, \frac{\partial d}{\partial y} = 1$

$f = d \cdot z$ $\quad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Chain Rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial x}$$

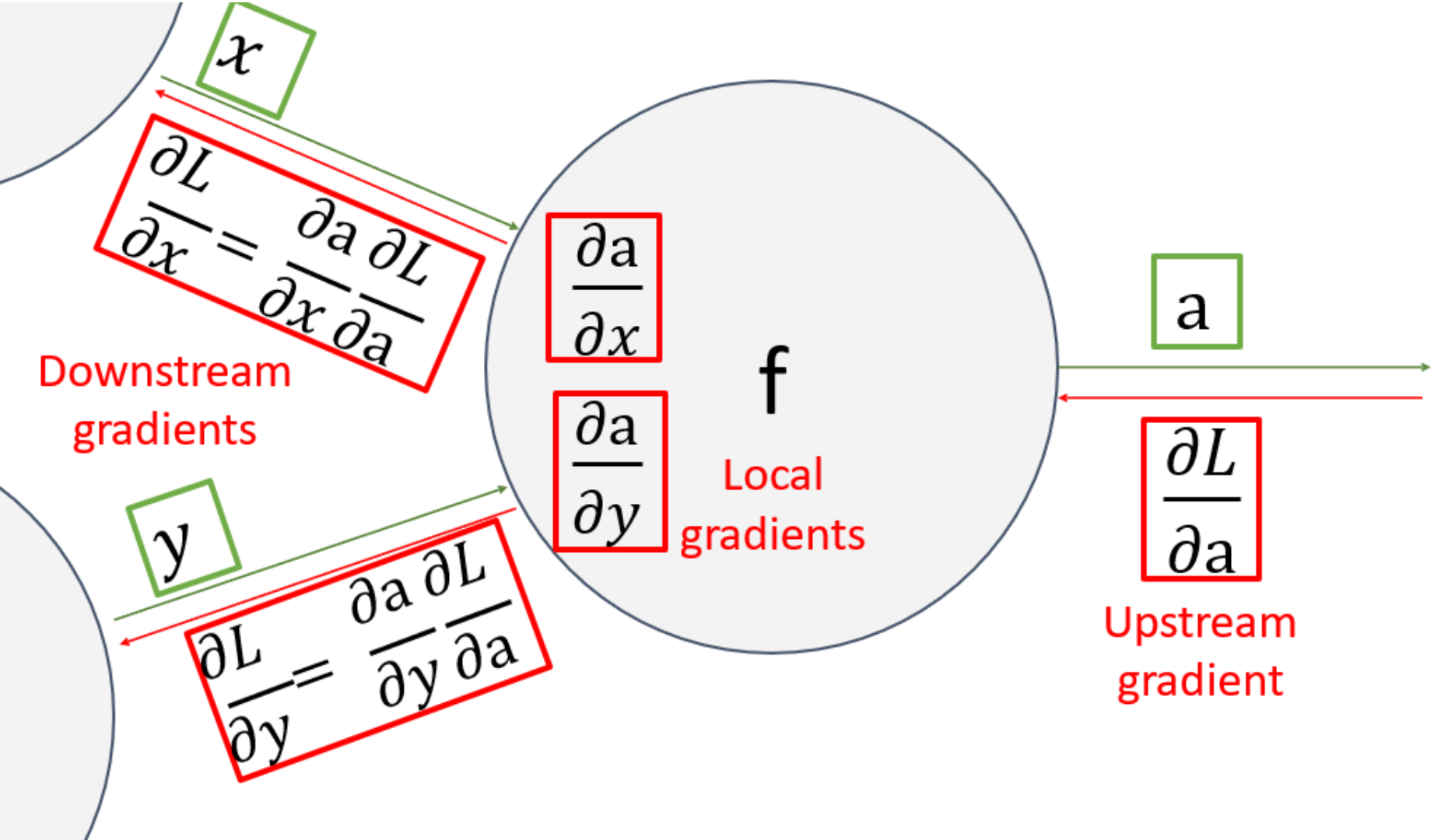
$\frac{\partial f}{\partial x}$

$\rightarrow \frac{\partial f}{\partial x} = 4 \cdot 1 = 4$

Downstream Gradient

Local Gradient

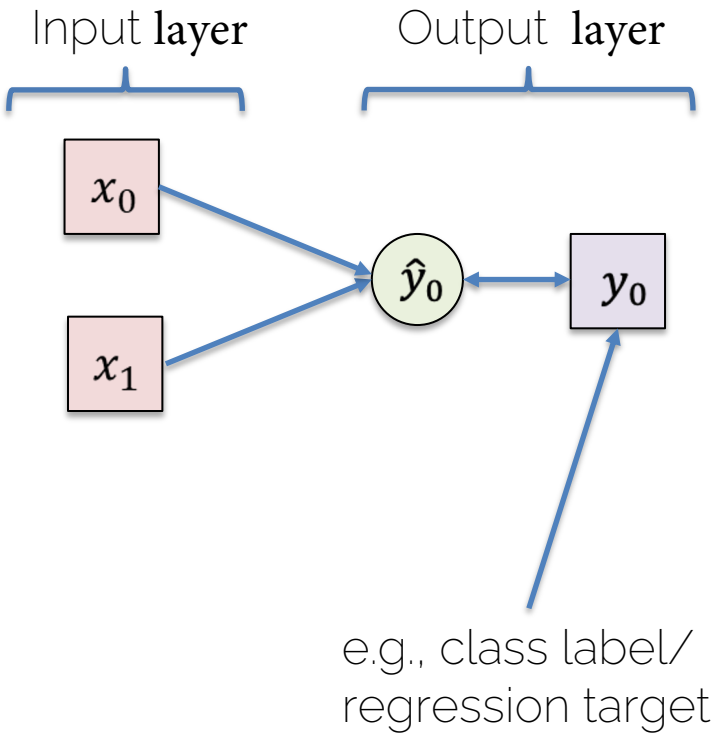
Upstream Gradient

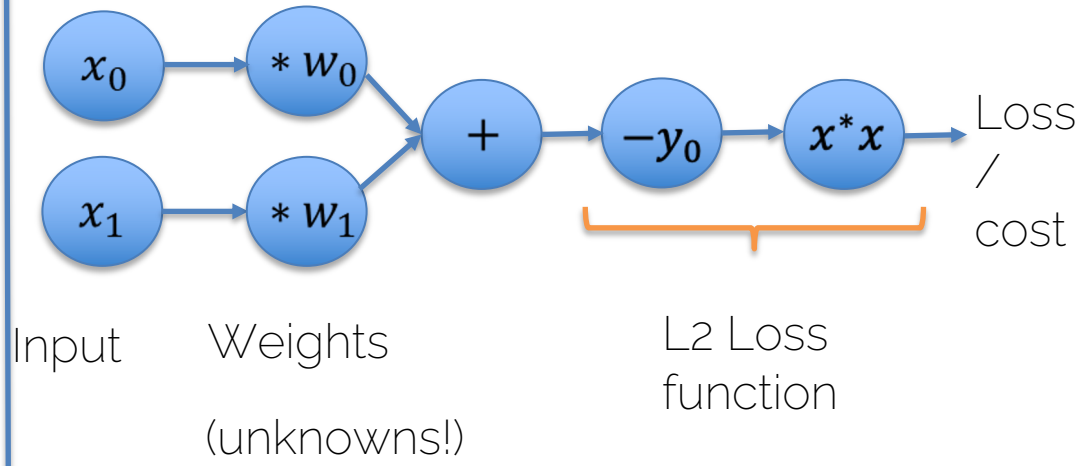

x
$\frac{\partial L}{\partial x} = \frac{\partial a}{\partial x}\frac{\partial L}{\partial a}$
Downstream gradients
$\frac{\partial a}{\partial x}$
$\frac{\partial a}{\partial y}$
f
Local gradients
a
$\frac{\partial L}{\partial a}$
Upstream gradient
y
$\frac{\partial L}{\partial y} = \frac{\partial a}{\partial y}\frac{\partial L}{\partial a}$

Compute Graphs -> Neural Networks

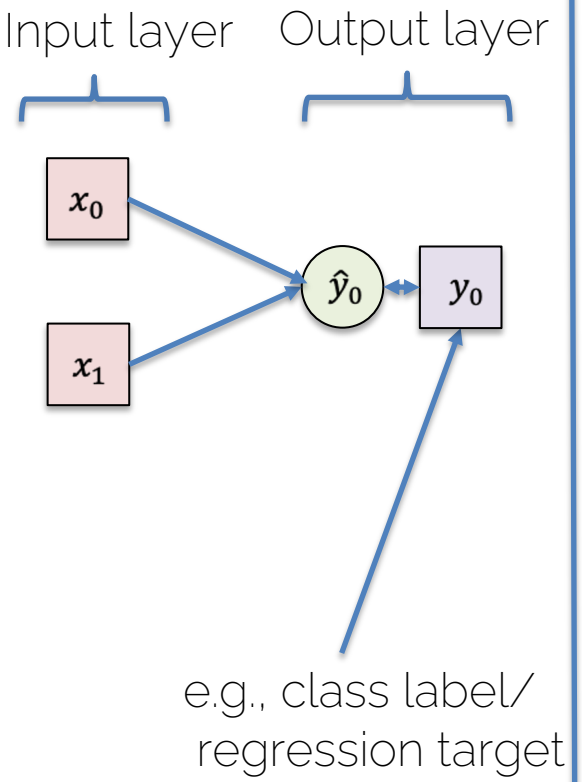
- x_k input variables
- $w_{l,m,n}$ network weights (note 3 indices)
  - l which layer
  - m which neuron in layer
  - n which weight in neuron
- $\hat{y}_i$ computed output (i output dim; n_{out})
- y_i ground truth targets
- L loss function

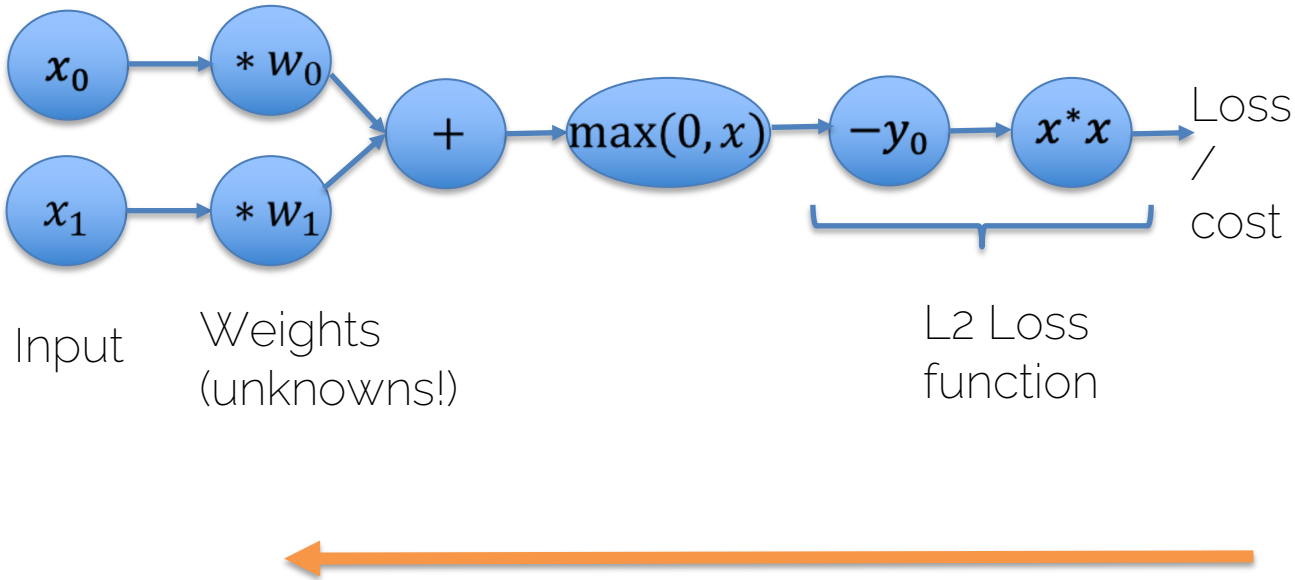
Compute Graphs -> Neural Networks





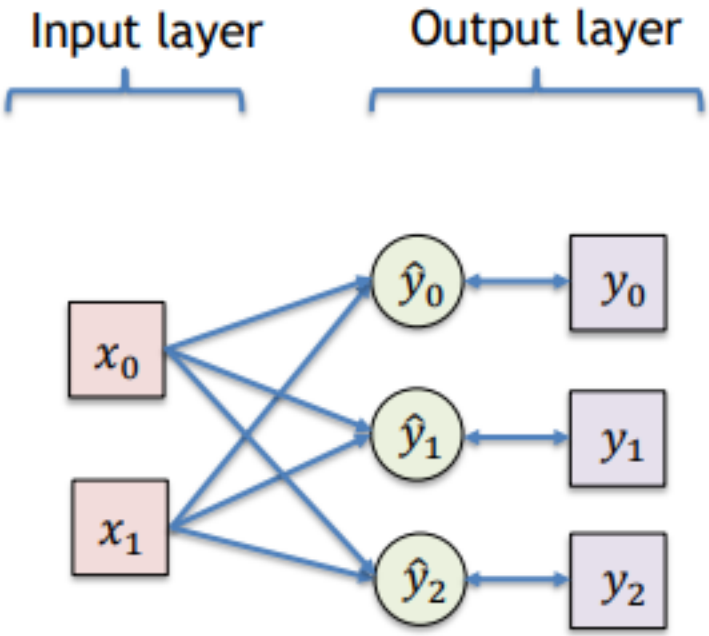
Compute Graphs -> Neural Networks

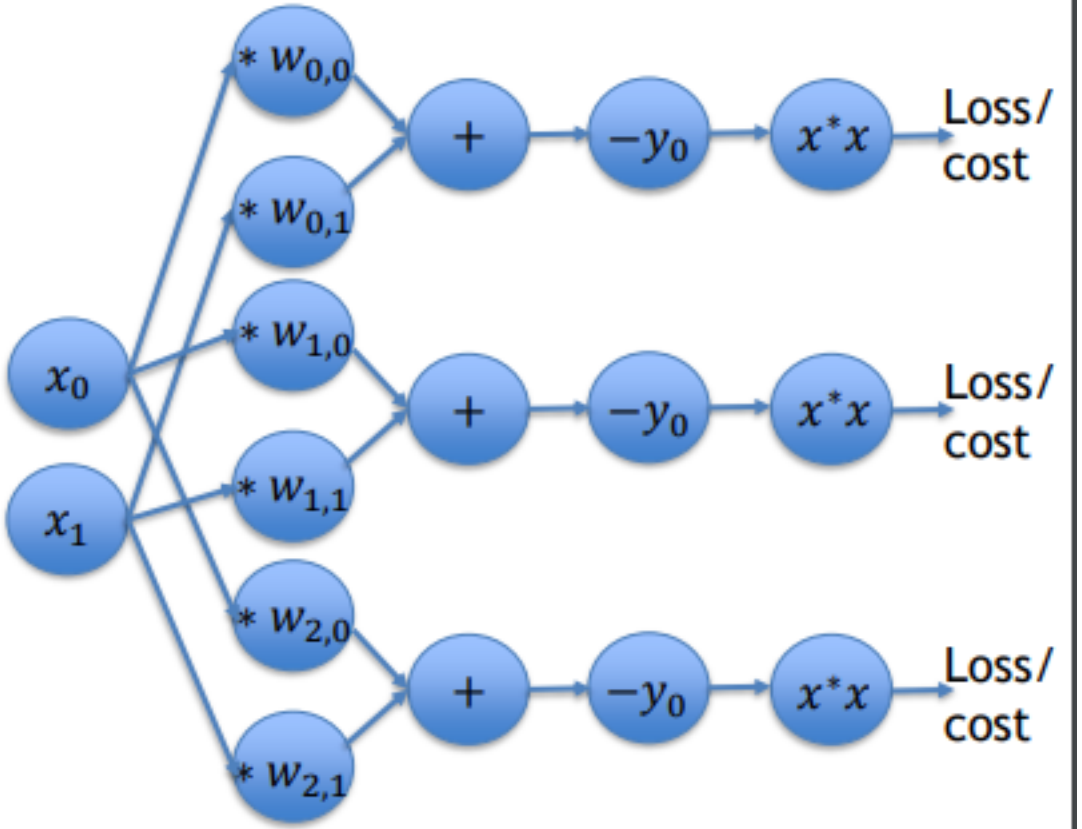




We want to compute gradients w.r.t. all weights $\boldsymbol{W}$

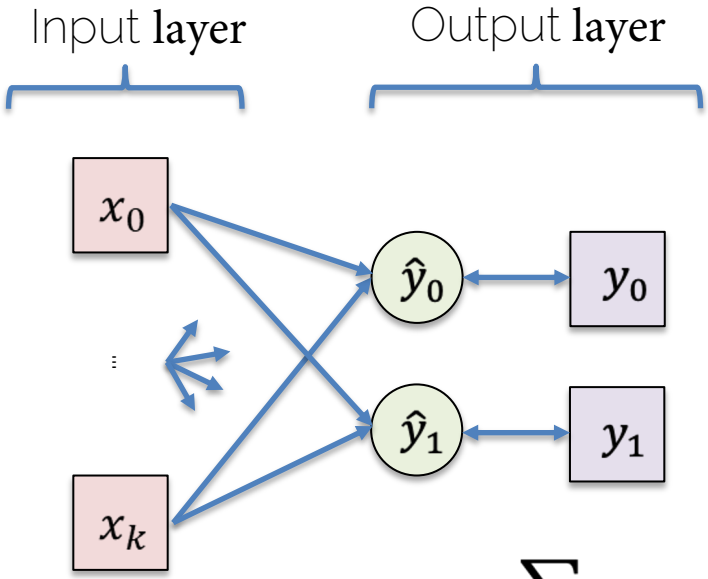
Compute Graphs -> Neural Networks





We want to compute gradients w.r.t. all weights W

Compute Graphs -> Neural Networks



$$\hat{y}_i = A(b_i + \sum_k x_k w_{i,k})$$

Activation function

bias

Goal: We want to compute gradients of the loss function L w.r.t. all weights $\boldsymbol{W}$

$$L = \sum_i L_i$$

L: sum over loss per sample, e.g. L2 loss ⟶ simply sum up squares:

$$L_i = (\hat{y}_i - y_i)^2$$

⟶ use chain rule to compute partials

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{i,k}}$$

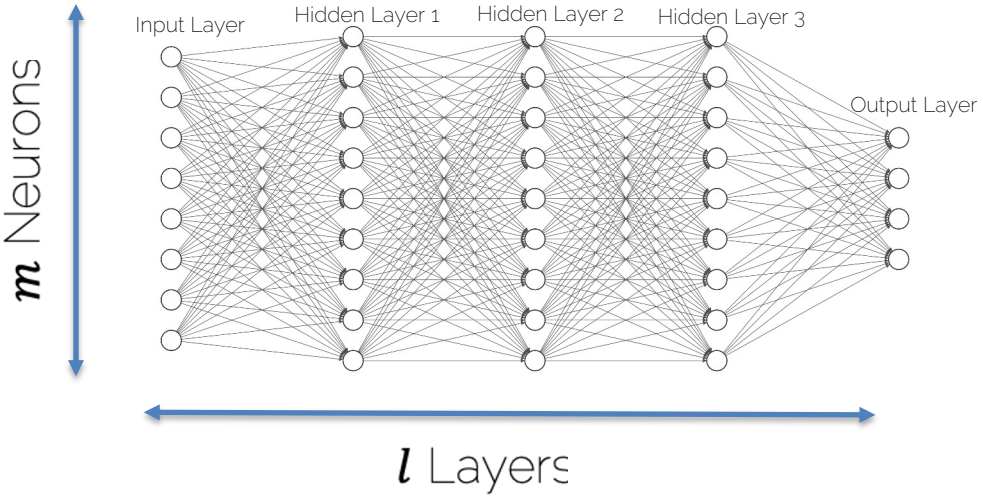
We want to compute gradients w.r.t. all weights $\boldsymbol{W}$ AND all biases $\boldsymbol{b}$

Gradient Descent for Neural Networks

For a given training pair $\{\boldsymbol{x}, \boldsymbol{y}\}$, we want to update all weights, i.e., we need to compute the derivatives w.r.t. to all weights:

$$\nabla_{\boldsymbol{W}} f_{\{\boldsymbol{x},\boldsymbol{y}\}}(\boldsymbol{W}) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial w_{l,m,n}} \end{bmatrix}$$

Gradient step $\boldsymbol{W'} = \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} f_{\{\boldsymbol{x},\boldsymbol{y}\}}(\boldsymbol{W})$



The Flow of the Gradients

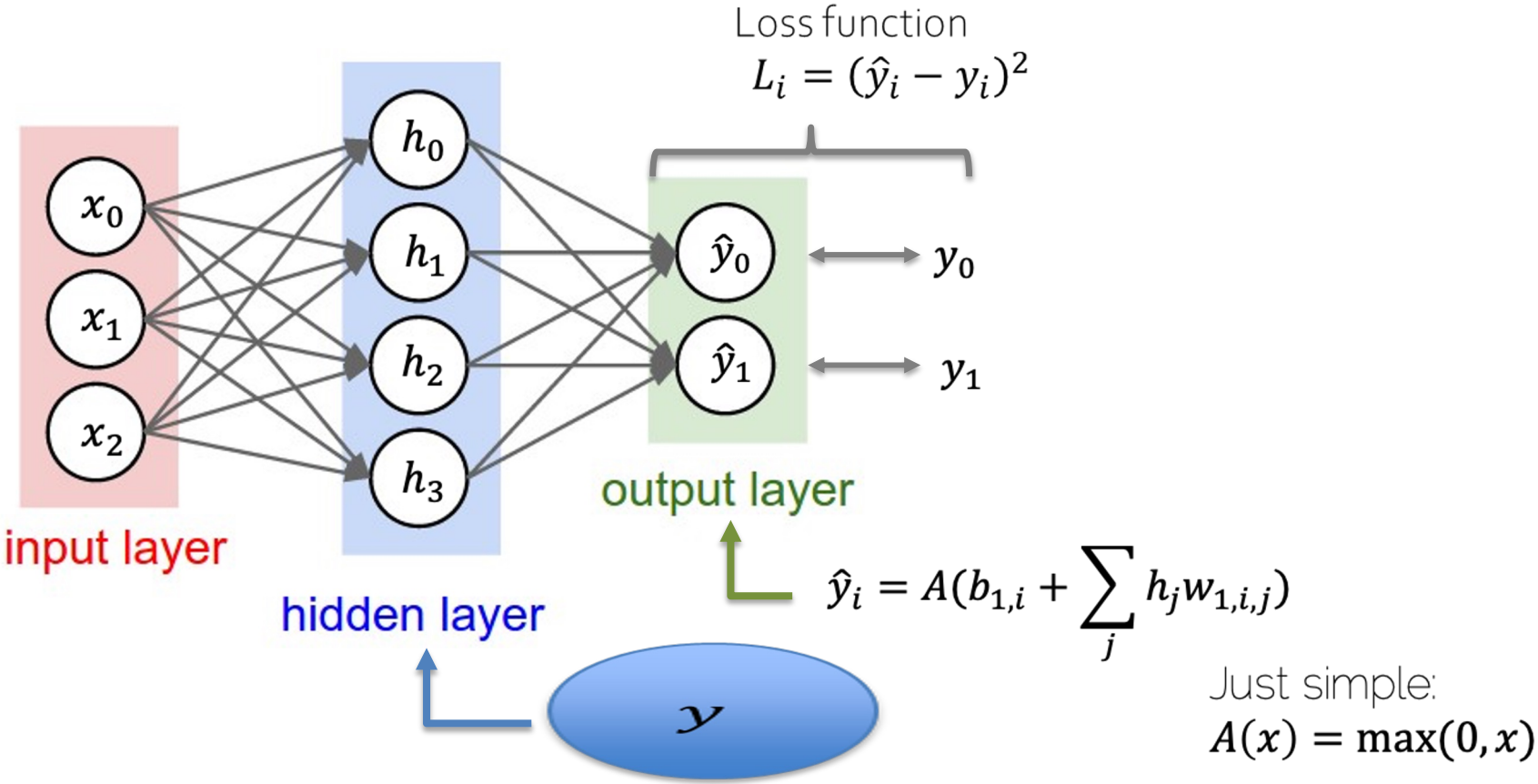
- Many many many many of these nodes form a neural network

NEURONS

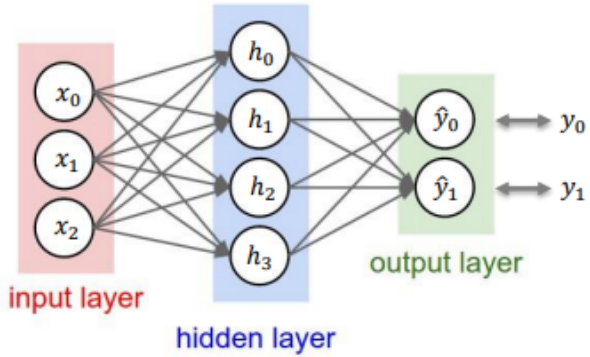
- Each one has its own work to do

FORWARD AND BACKWARD PASS

Gradient Descent for Neural Networks



Gradient Descent for Neural Networks



$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

$$L_i = (\hat{y}_i - y_i)^2$$

Just go through layer by layer

Backpropagation

$$\frac{\partial L}{\partial w_{1,i,j}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{1,i,j}}$$

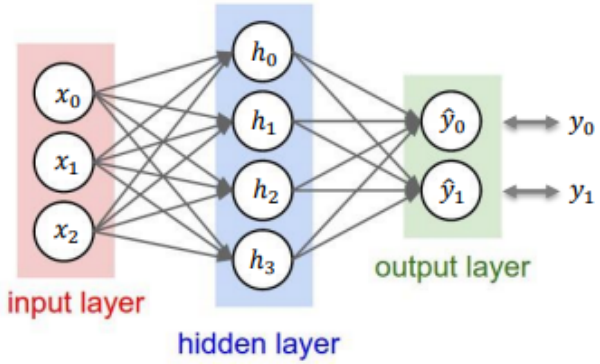
$$\frac{\partial L_i}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)$$

$$\frac{\partial \hat{y}_i}{\partial w_{1,i,j}} = h_j \quad \text{if} > 0, \text{else } 0$$

$$\frac{\partial L}{\partial w_{0,j,k}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{0,j,k}}$$

...

Gradient Descent for Neural Networks



$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

$$L_i = (\hat{y}_i - y_i)^2$$

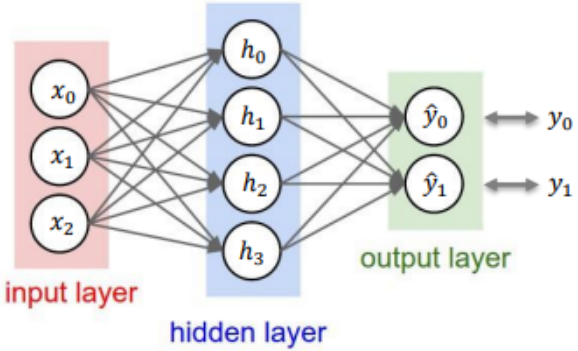
How many unknown weights?

- Output layer: $2 \cdot 4 + 2$
- Hidden Layer: $4 \cdot 3 + 4$

#neurons · #input channels + #biases

Note that some activations have also weights

Derivatives of Cross Entropy Loss



Binary Cross Entropy loss

$$L = -\sum_{i=1}^{n_{out}} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

$$\hat{y}_i = \frac{1}{1 + e^{-s_i}} \qquad s_i = \sum_{j} h_j w_{ji}$$

output scores

Gradients of weights of last layer:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial s_i} \cdot \frac{\partial s_i}{\partial w_{ji}}$$

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{-y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} = \frac{\hat{y}_i - y_i}{\hat{y}_i (1 - \hat{y}_i)},$$

$$\frac{\partial \hat{y}_i}{\partial s_i} = \hat{y}_i (1 - \hat{y}_i),$$

$$\frac{\partial s_i}{\partial w_{ji}} = h_j$$

$$\Rightarrow \frac{\partial L}{\partial w_{ji}} = (\hat{y}_i - y_i) h_j, \qquad \frac{\partial L}{\partial s_i} = \hat{y}_i - y_i$$

Neural Network

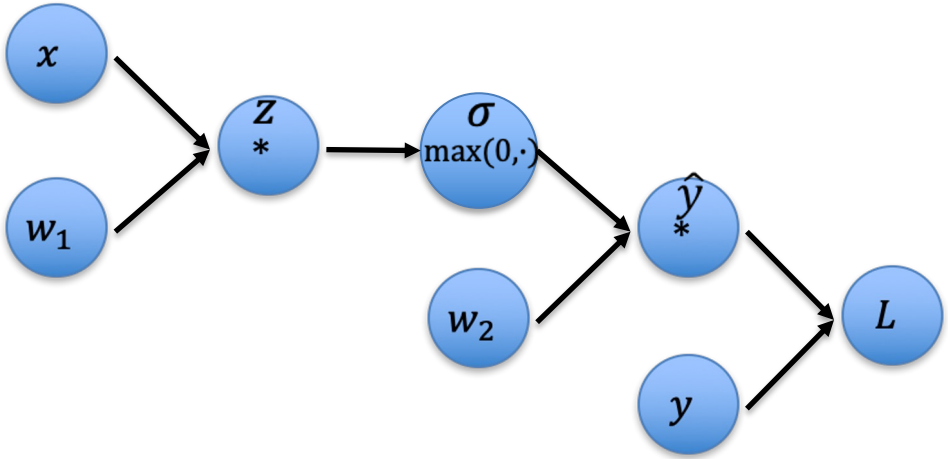
- Linear score function $\boldsymbol{f = Wx}$

- Neural network is a nesting of 'functions'
  - 2-layers: $\boldsymbol{f = W_2 \max(0, W_1 x)}$
  - 3-layers: $\boldsymbol{f = W_3 \max(0, W_2 \max(0, W_1 x))}$
  - 4-layers: $\boldsymbol{f = W_4 \tanh\,(W_3, \max(0, W_2 \max(0, W_1 x)))}$
  - 5-layers: $\boldsymbol{f = W_5 \sigma(W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1 x))))}$
  - … up to hundreds of layers

Back to Compute Graphs & NNs

- Inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Two-layer NN for regression with ReLU activation
- Function we want to optimize:

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2$$



Gradient Descent for Neural Networks

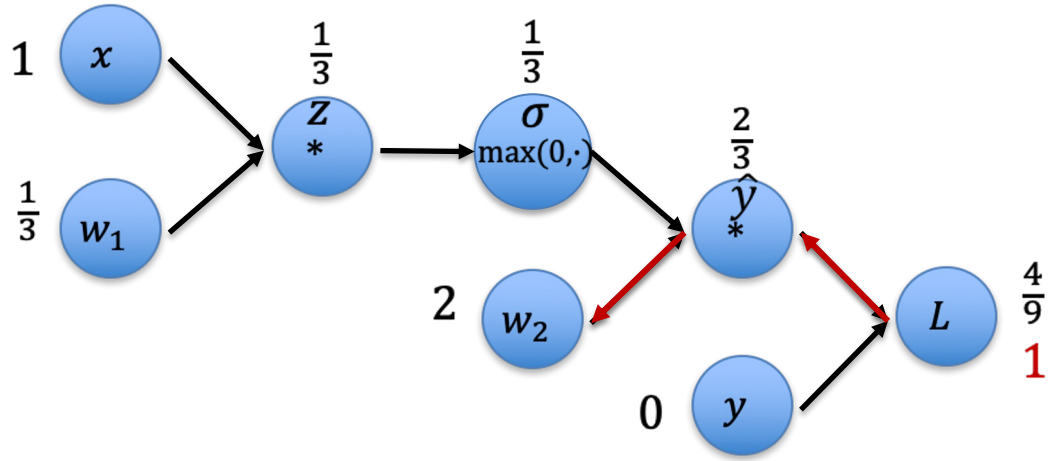
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \quad \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

Gradient Descent for Neural Networks

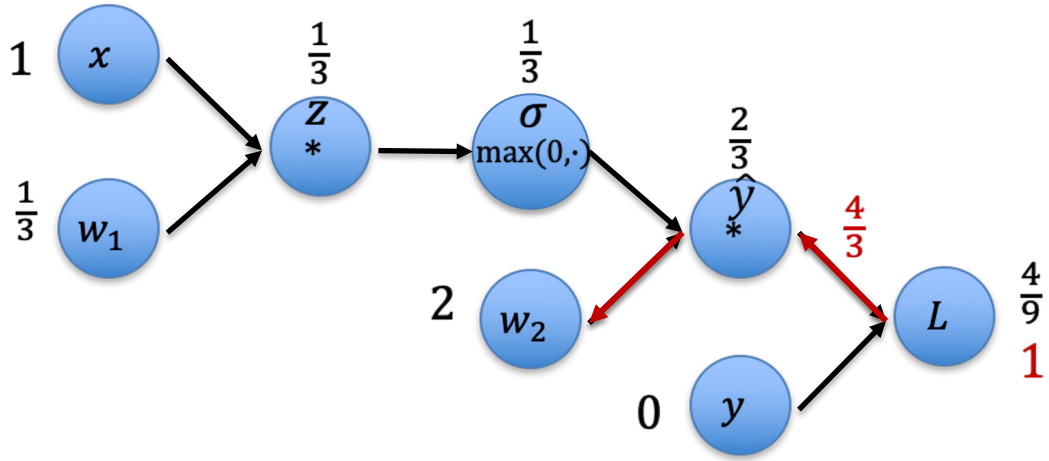
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$2 \cdot \frac{2}{3}$

Gradient Descent for Neural Networks

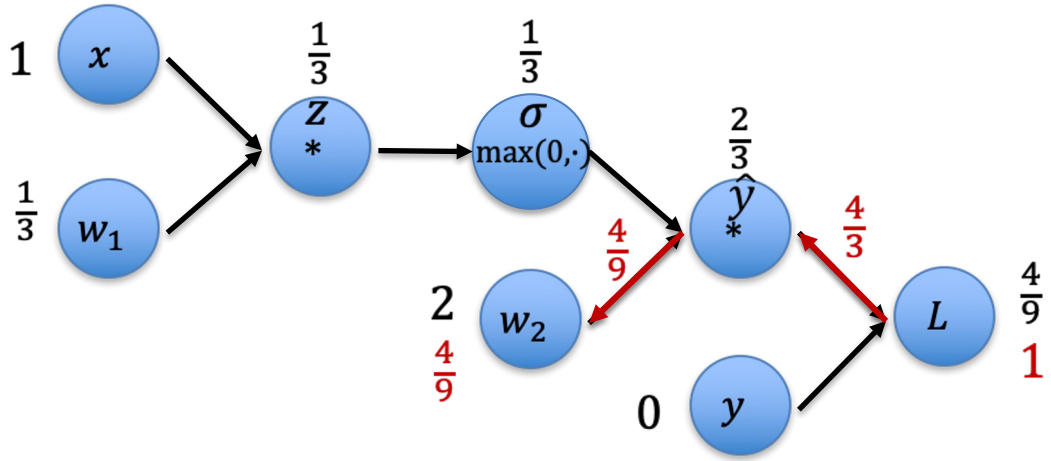
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\hat{y}_i - y_i||_2^2$$

In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \quad \Rightarrow \boxed{\frac{\partial \hat{y}}{\partial w_2} = \sigma}$$



Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

$$2 \cdot \frac{2}{3} \cdot \frac{1}{3}$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

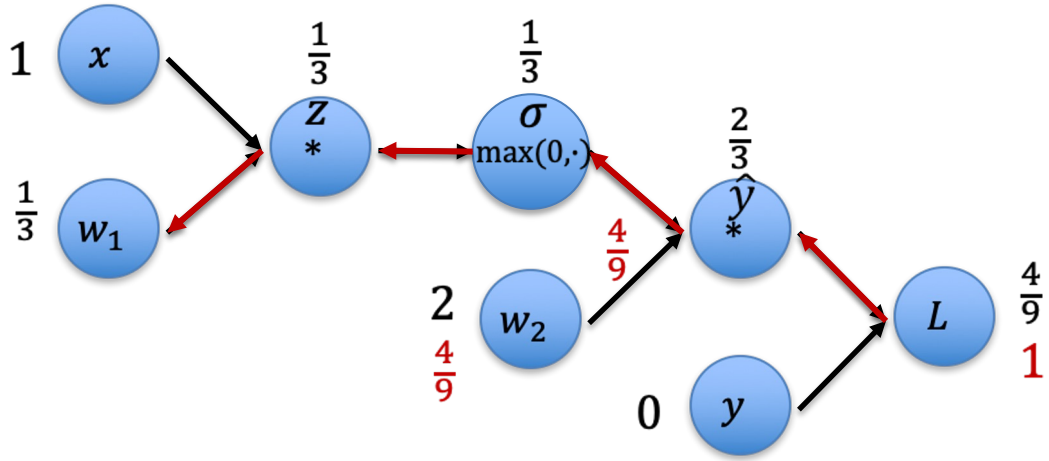
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \quad \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$$

$$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$

$$z = x \cdot w_1 \quad \Rightarrow \frac{\partial z}{\partial w_1} = x$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

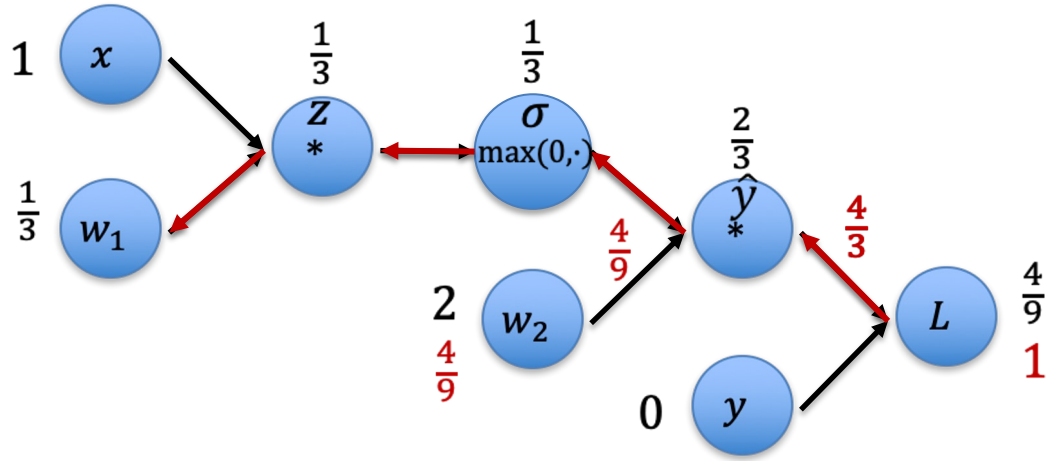
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$$

$$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$

$$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3}$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

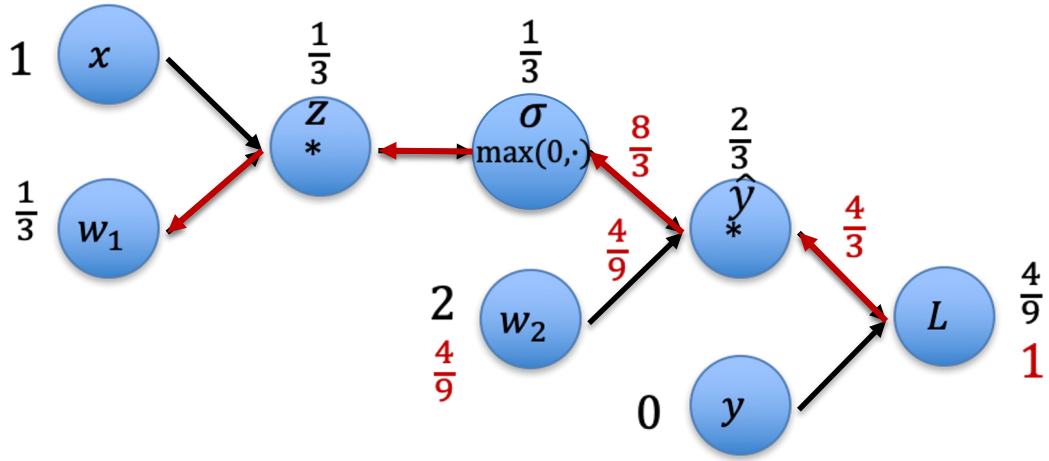
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \Rightarrow \boxed{\frac{\partial \hat{y}}{\partial \sigma} = w_2}$$

$$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$

$$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3} \cdot 2$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

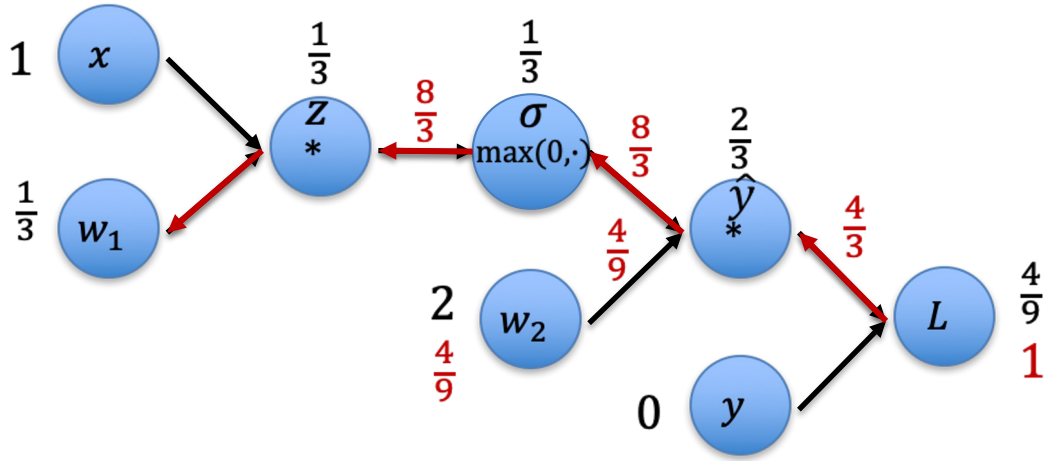
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$$

$$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$

$$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3} \cdot 2 \cdot 1$$

Gradient Descent for Neural Networks

Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

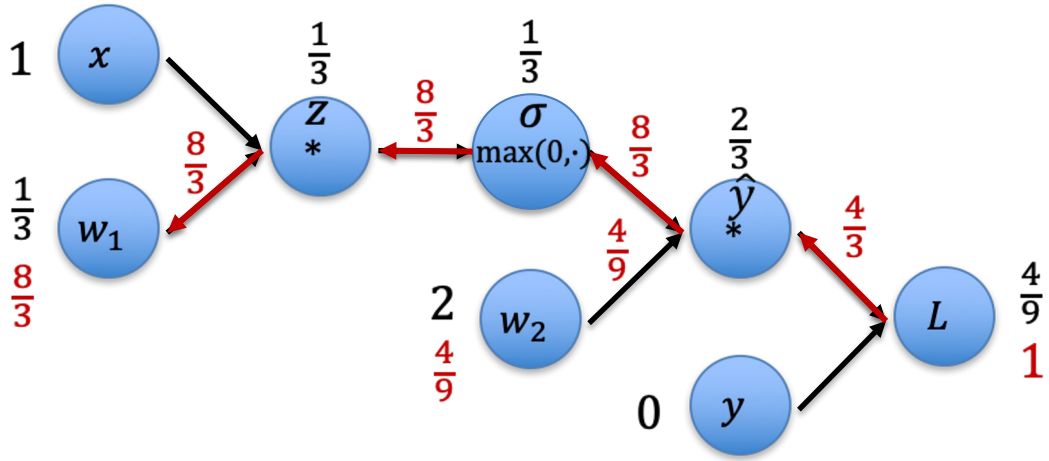
In our case $n, d = 1$:

$$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$$

$$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$

$$z = x \cdot w_1 \Rightarrow \boxed{\frac{\partial z}{\partial w_1} = x}$$



Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3} \cdot 2 \cdot 1 \cdot 1$$

Gradient Descent for Neural Networks

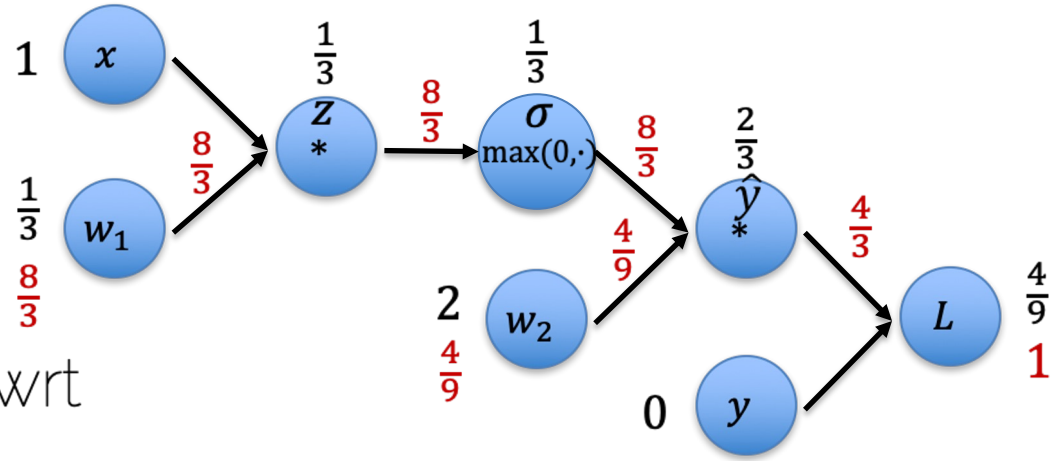
- Function we want to optimize:

$$f(x, \boldsymbol{w}) = \sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2$$

- Computed gradients wrt to weights $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$
- Now: update the weights

$$\boldsymbol{w}' = \boldsymbol{w} - \alpha \cdot \nabla_{\boldsymbol{w}} f = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \nabla_{w_1} f \\ \nabla_{w_2} f \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{3} \\ 2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{8}{3} \\ \frac{4}{9} \end{pmatrix}$$



Some Concepts

Batch gradient descent

Stochastic gradient descent

Mini-batch gradient descent

Batch Normalization

Transfer Learning vs Fine Tuning