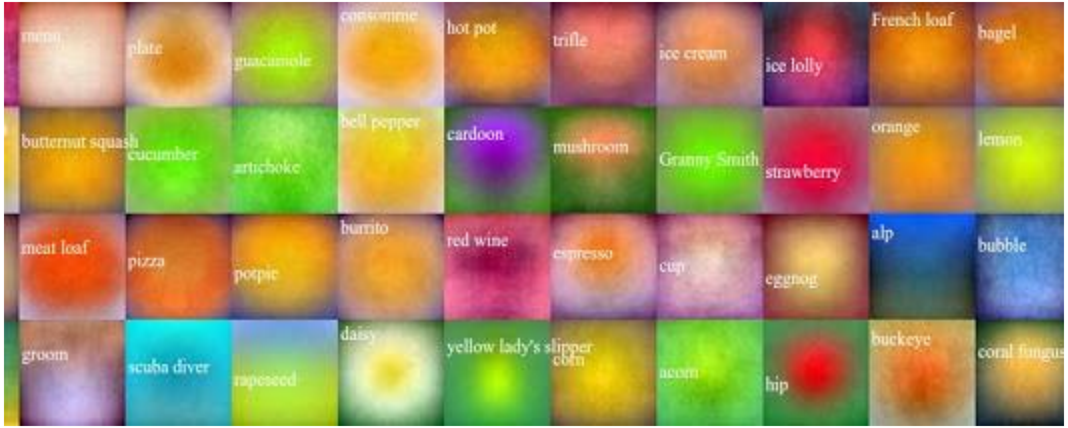
# Optimization and Backpropagation

# Lecture 3 Recap

# Neural Network

- Linear score function $\boldsymbol{f} = \boldsymbol{W}\boldsymbol{x}$


On CIFAR-10




On ImageNet

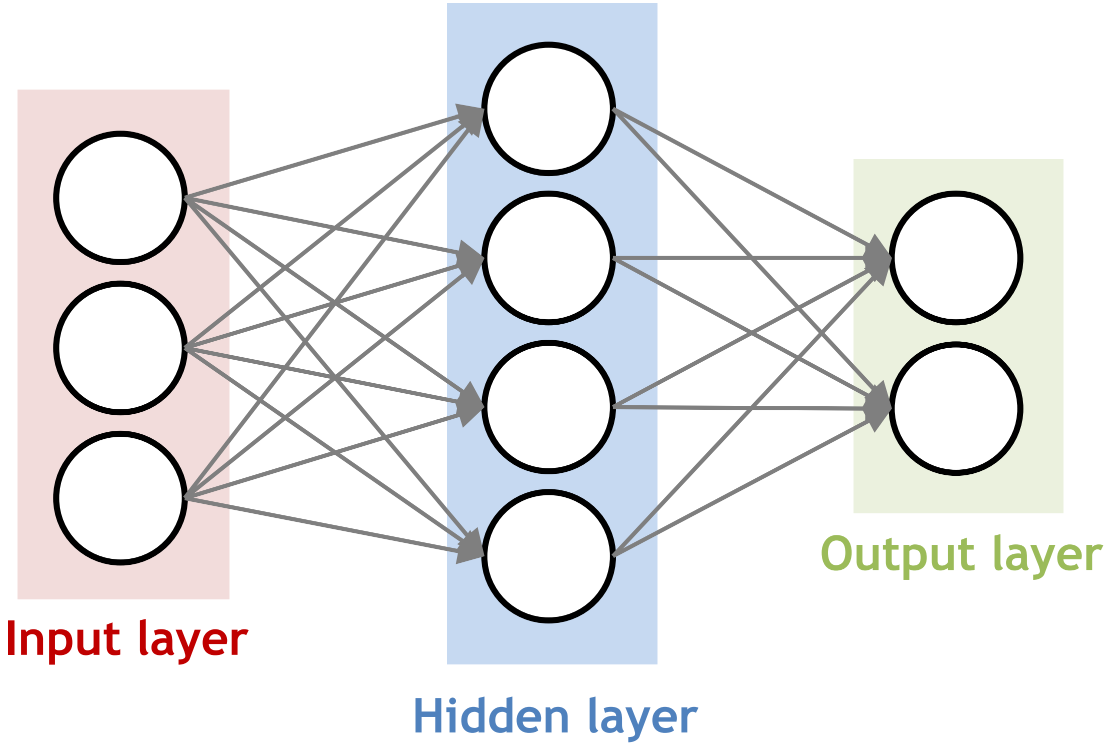
Credit: Li/Karpathy/Johnson

# Neural Network

- Linear score function $\boldsymbol{f} = \boldsymbol{W}\boldsymbol{x}$

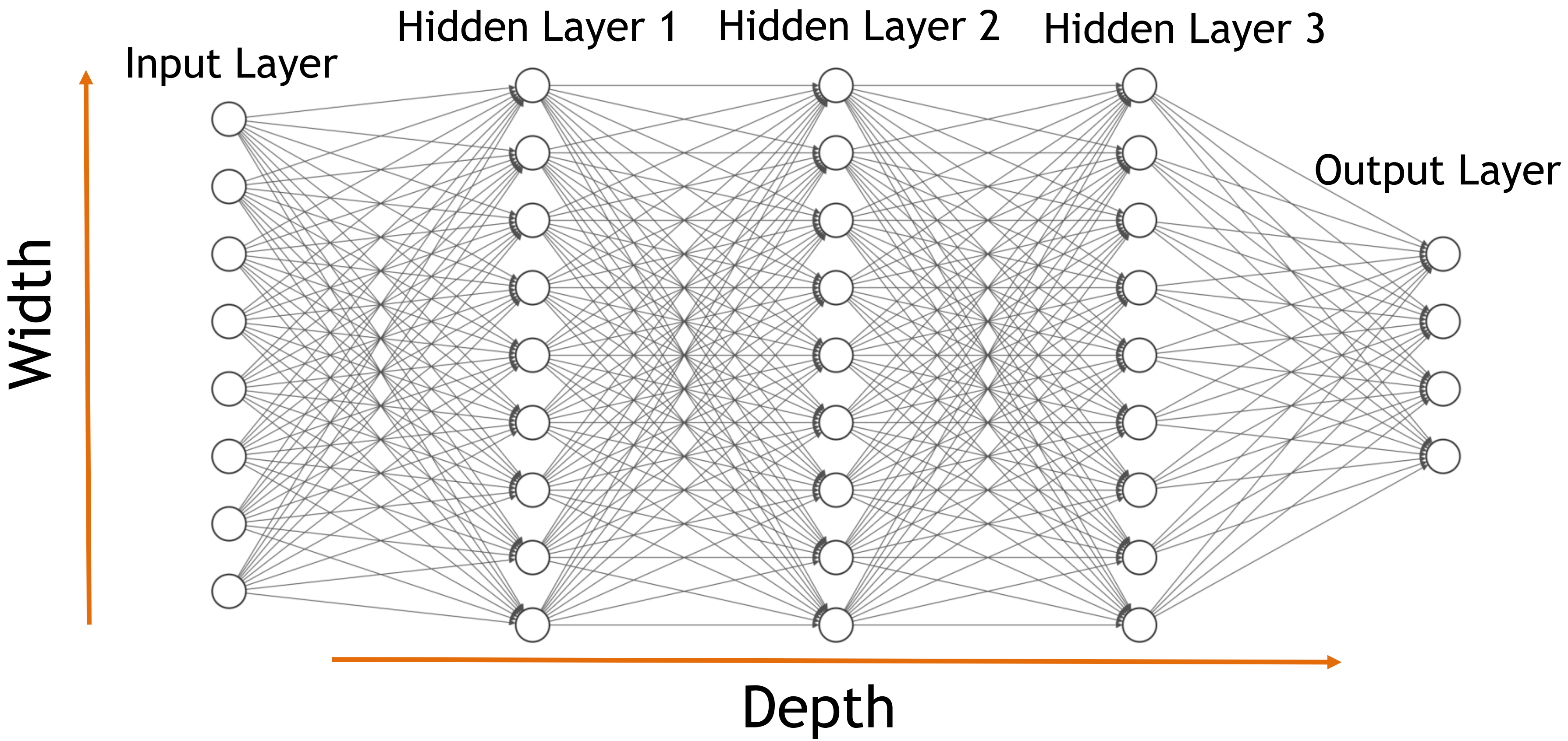
- Neural network is a nesting of 'functions'
  - 2-layers: $\boldsymbol{f} = \boldsymbol{W_2} \max(\boldsymbol{0}, \boldsymbol{W_1}\boldsymbol{x})$
  - 3-layers: $\boldsymbol{f} = \boldsymbol{W_3} \max(\boldsymbol{0}, \boldsymbol{W_2} \max(\boldsymbol{0}, \boldsymbol{W_1}\boldsymbol{x}))$
  - 4-layers: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

# Neural Network



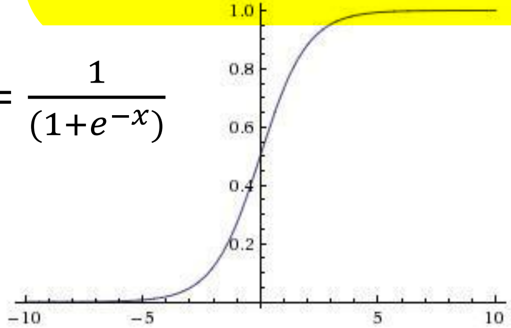

Credit: Li/Karpathy/Johnson

# Neural Network

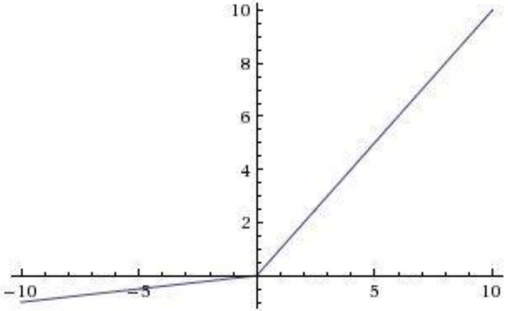


# Activation Functions

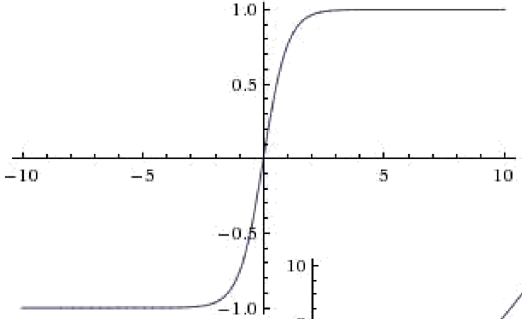
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$




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




tanh: $\tanh(x)$

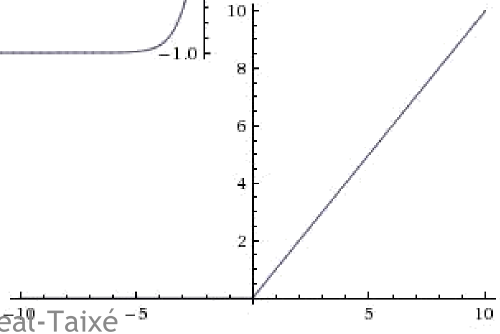



learnable weights

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

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

ReLU: $\max(0, x)$




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

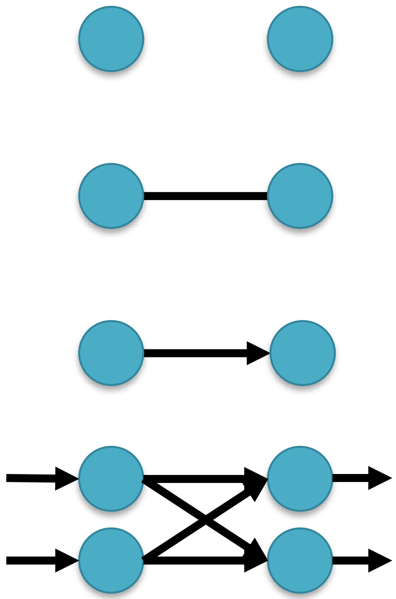
# Loss Functions

- Measure the goodness of the predictions (or equivalently, the network's performance)

- Regression loss
  - L1 loss $L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n} \sum_i^n ||y_i - \hat{y}_i||_1$
  - MSE loss $L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n} \sum_i^n ||y_i - \hat{y}_i||_2^2$
- Classification loss (for multi-class classification)
  - Cross Entropy loss $\mathrm{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = -\sum_{i=1}^n \sum_{k=1}^k (y_{ik} \cdot \log \hat{y}_{ik})$

# 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

# The Importance of Gradients

- Our optimization schemes are based on computing gradients

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

- One can compute gradients analytically but what if our function is too complex?

- Break down gradient computation

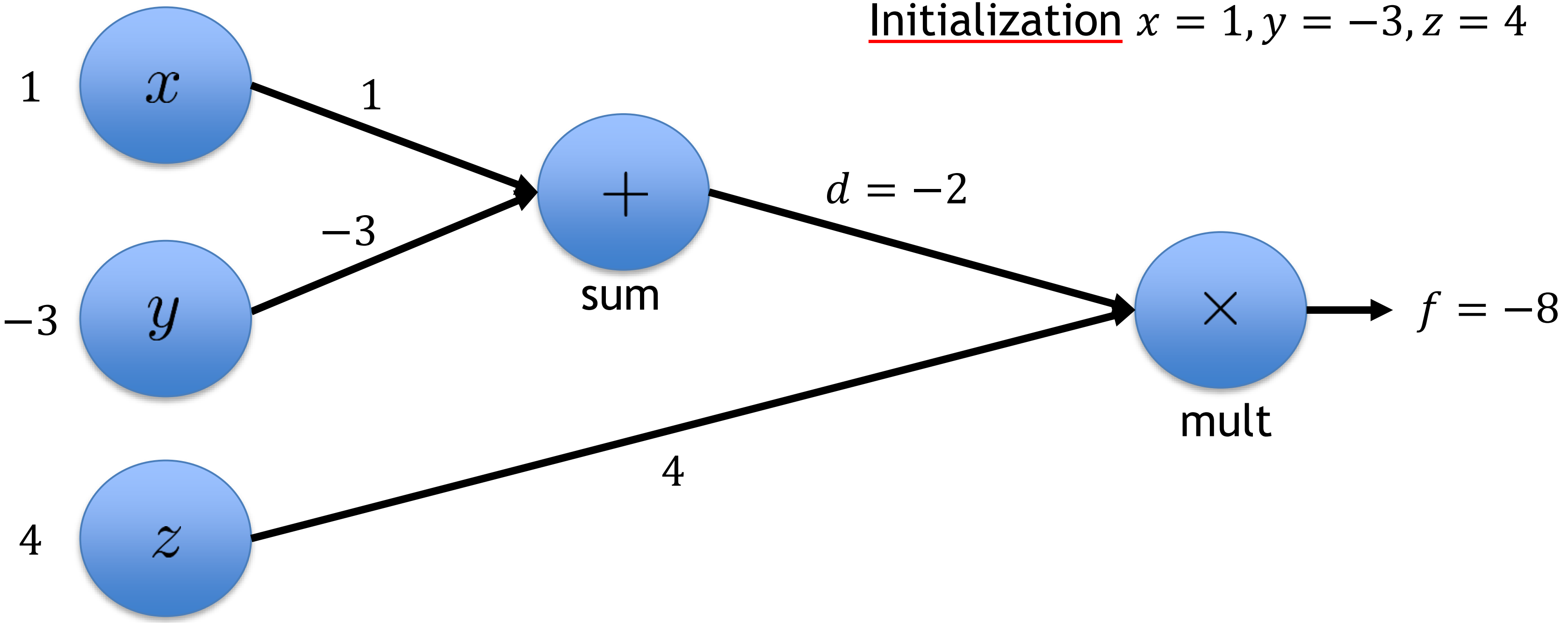
$\boxed{\text{Backpropagation}}$

Rumelhart 1986

# Backprop: Forward Pass

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

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

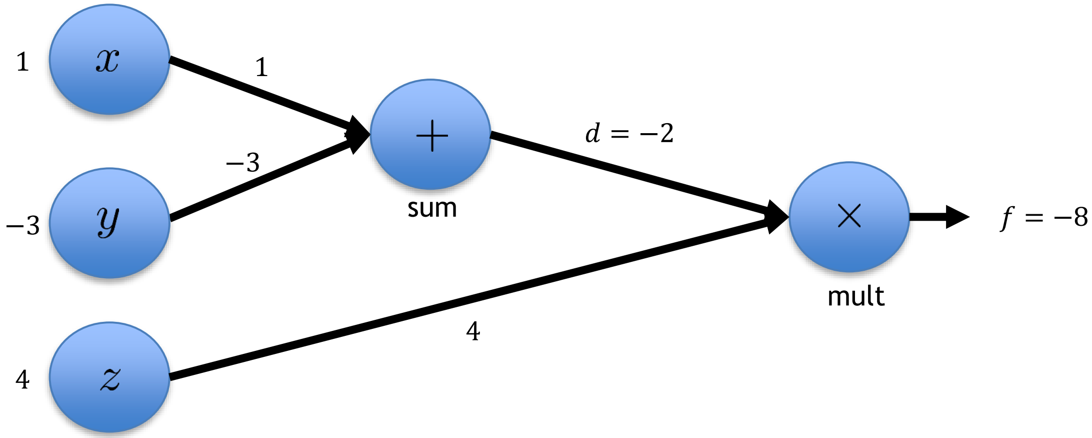


# Backprop: Backward Pass

Most important idea

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

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




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

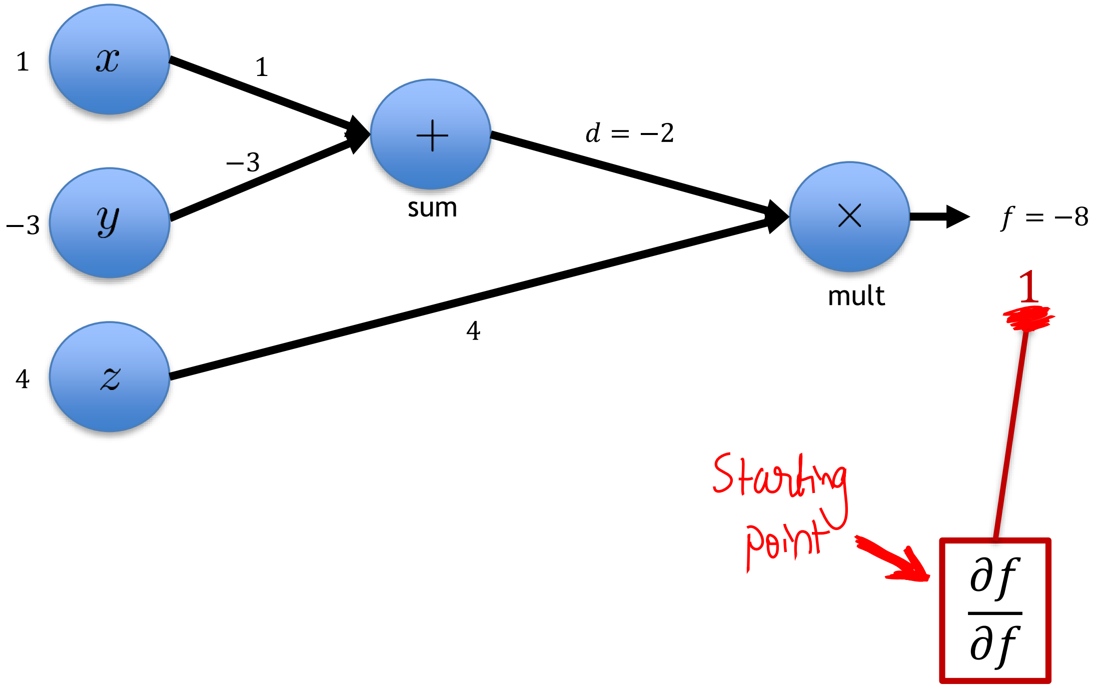
$$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$$

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

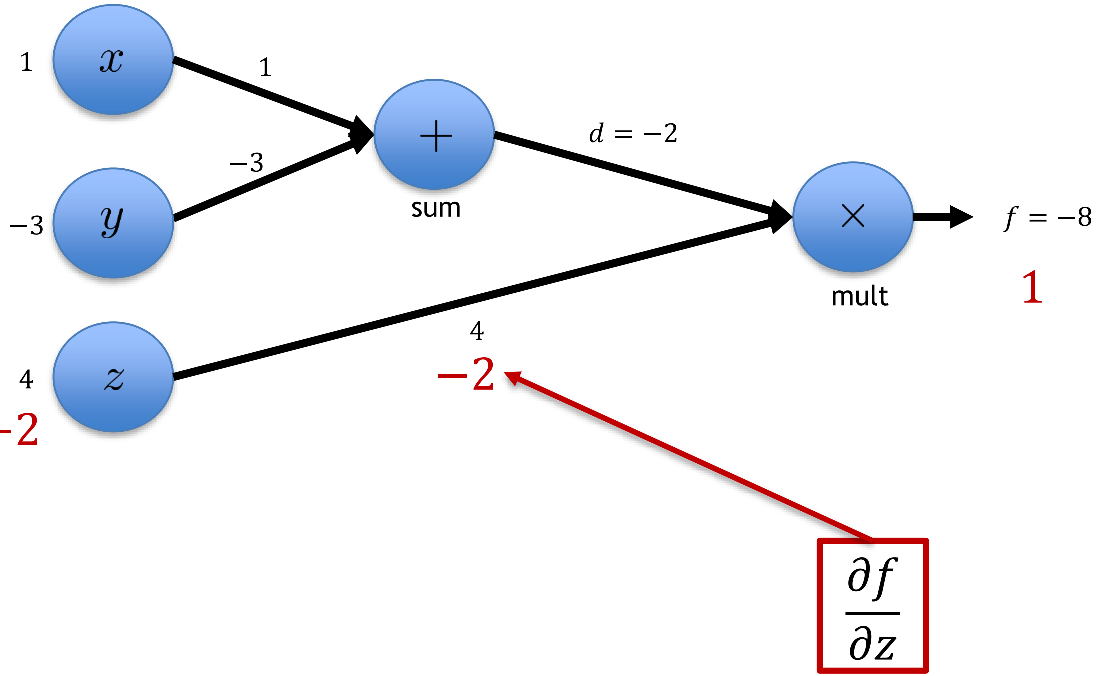
$$f(x, y, z) = (x + y) \cdot z$$

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

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

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

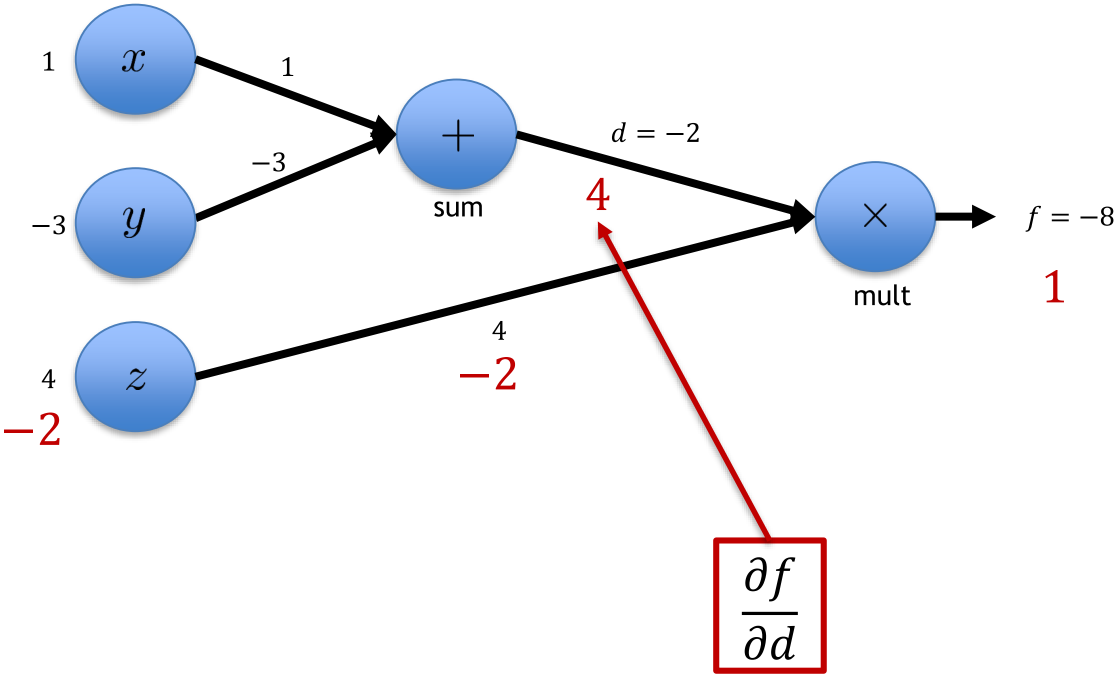
What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?



# Backprop: Backward Pass

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

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




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

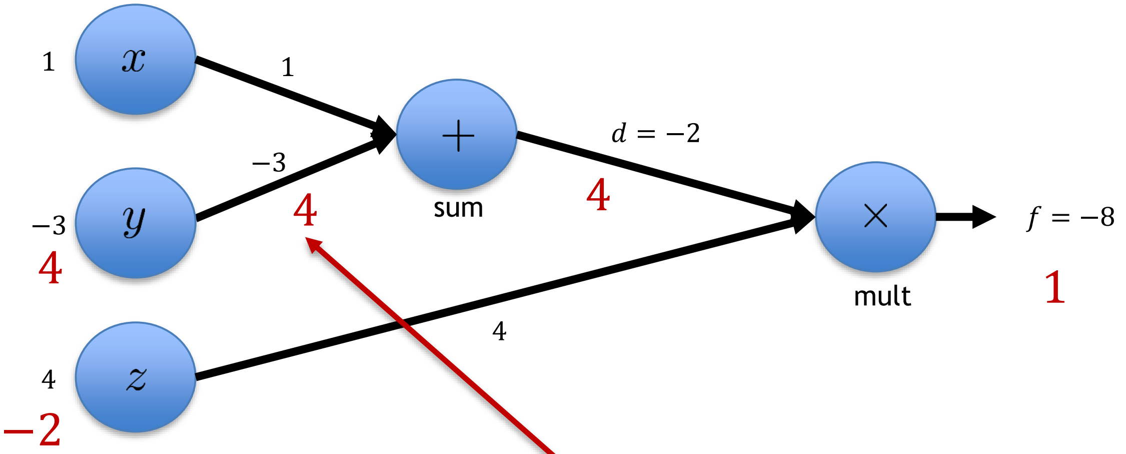
$$f = d \cdot z \qquad \boxed{\frac{\partial f}{\partial d} = z}, \frac{\partial f}{\partial z} = d$$

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




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

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

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

**Chain Rule:**

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

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

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

# Backprop: Backward Pass

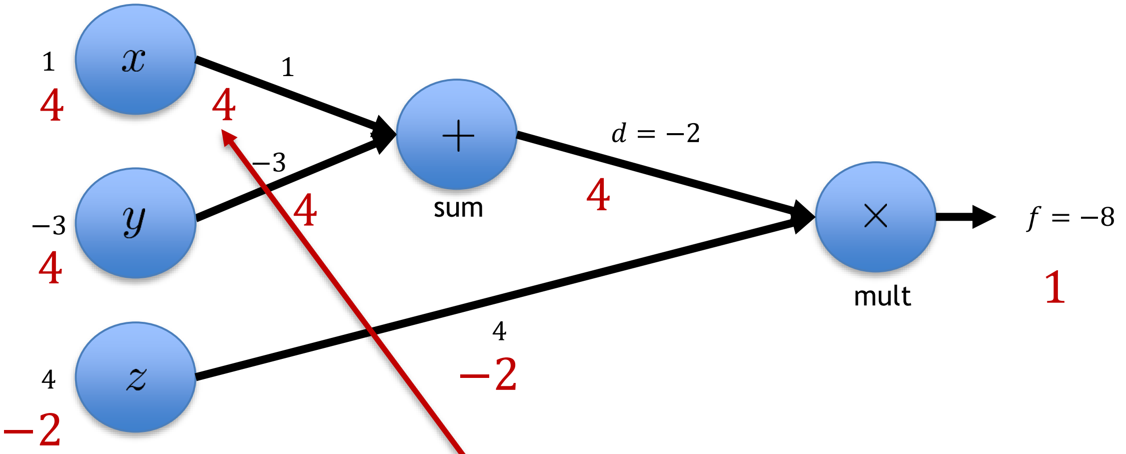
$$f(x,y,z) = (x+y)\cdot z$$

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

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

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

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




Chain Rule:

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

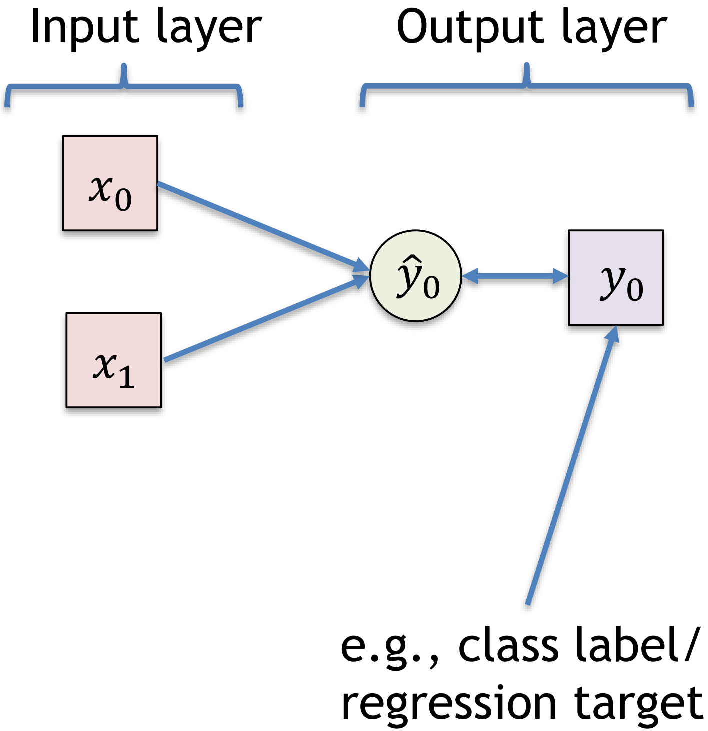
$$\boxed{\frac{\partial f}{\partial x}}$$

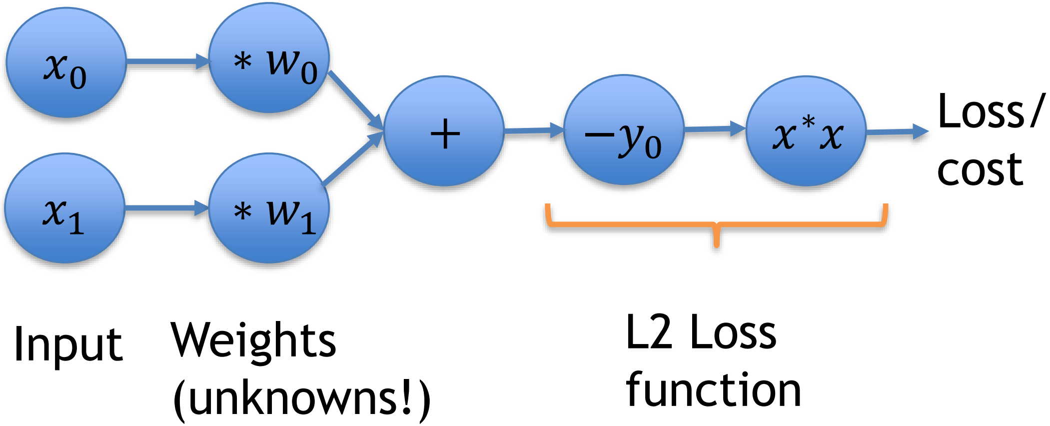
$$\rightarrow \frac{\partial f}{\partial x} = 4 \cdot 1 = 4$$

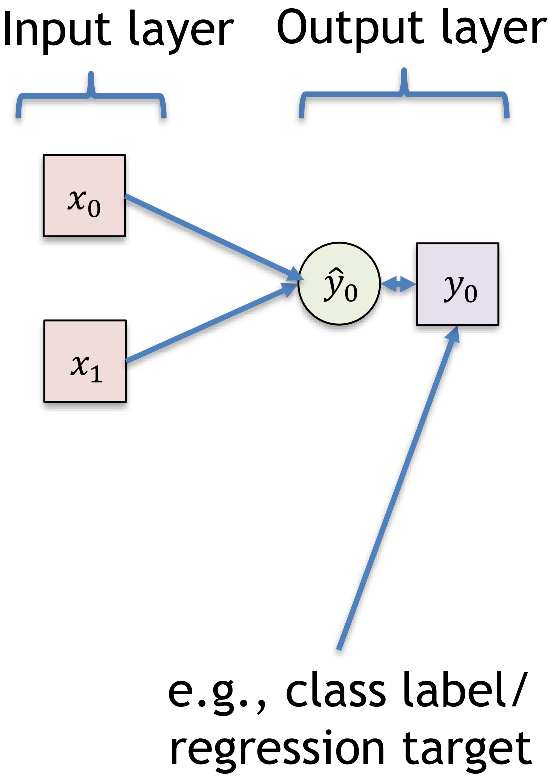
# Compute Graphs -> Neural Networks

- $x_k$ 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

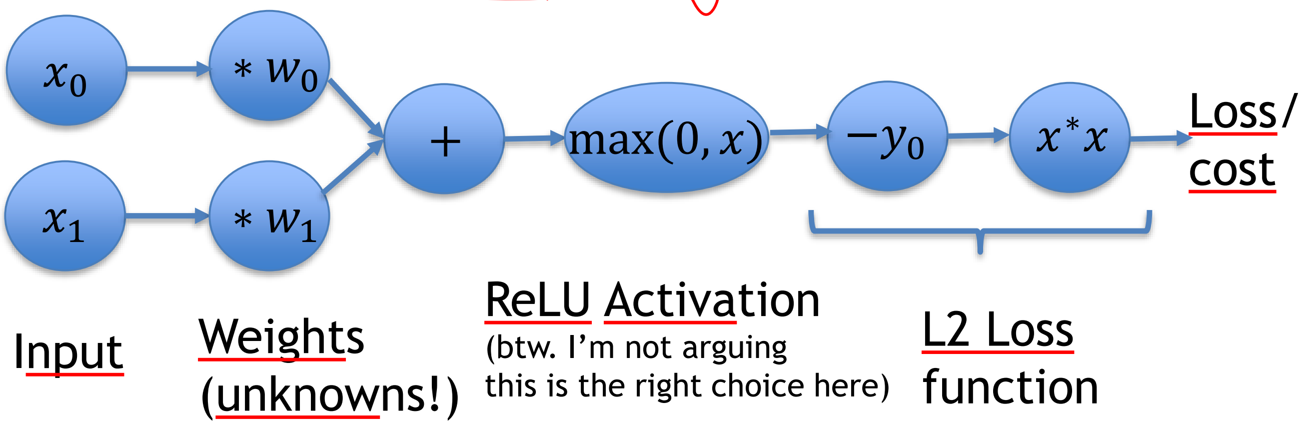
# Compute Graphs -> Neural Networks



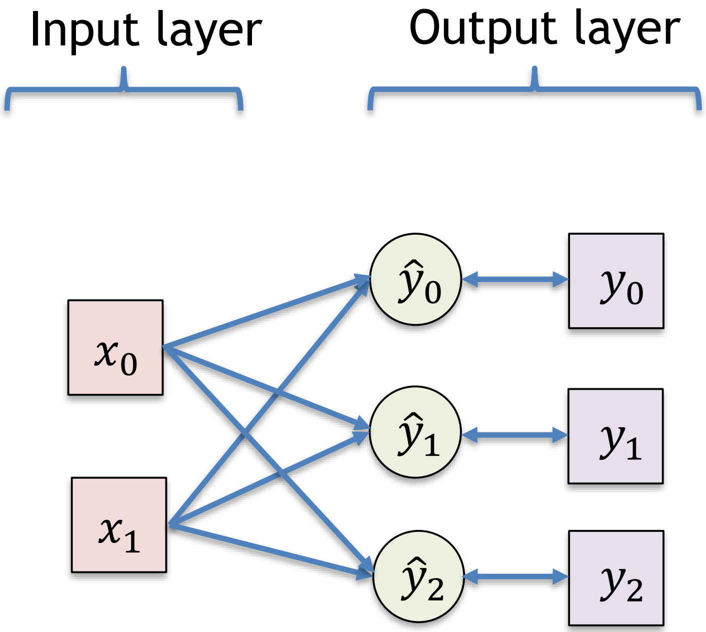



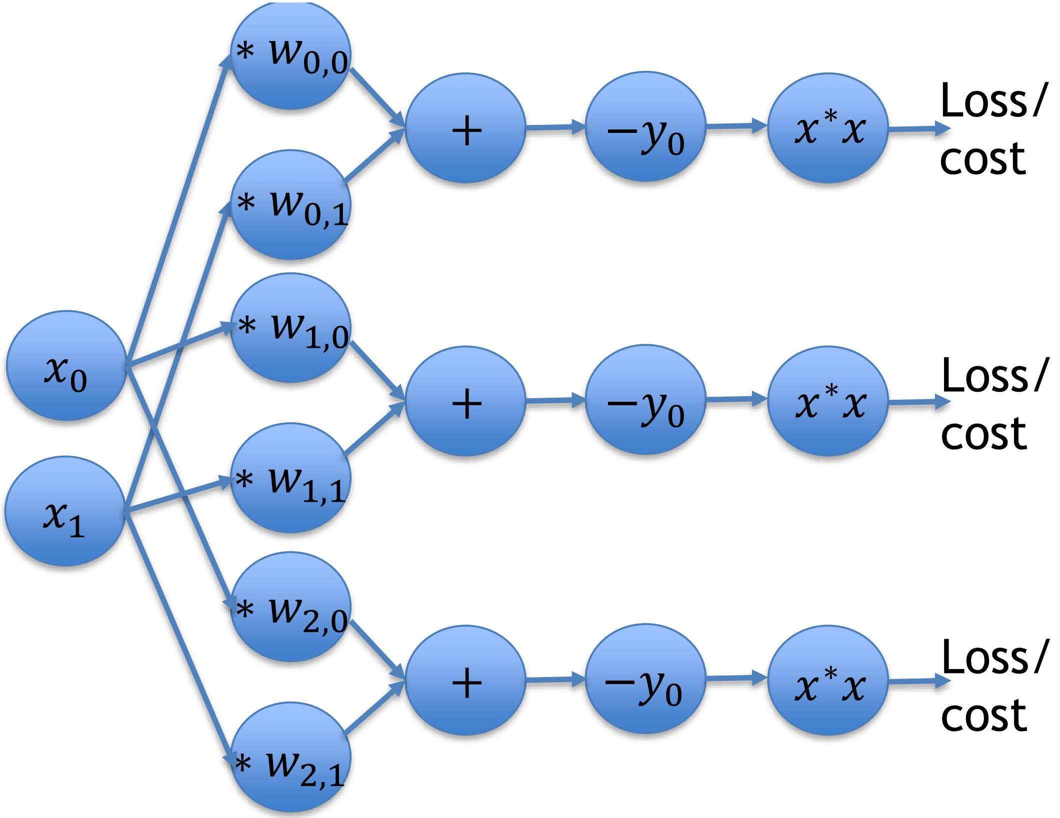



Graphically

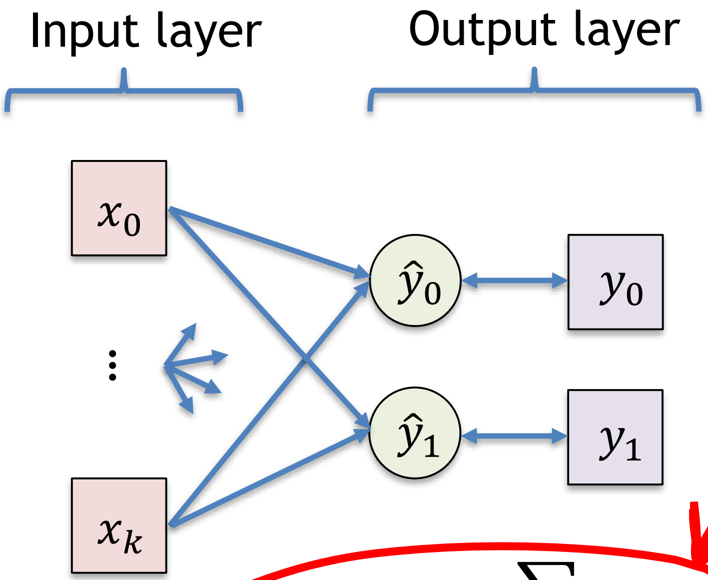



We want to compute gradients w.r.t. all weights $\boldsymbol{W}$




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




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

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 $\longrightarrow$ simply sum up squares:

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

$\longrightarrow$ use chain rule to compute partials

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

NB

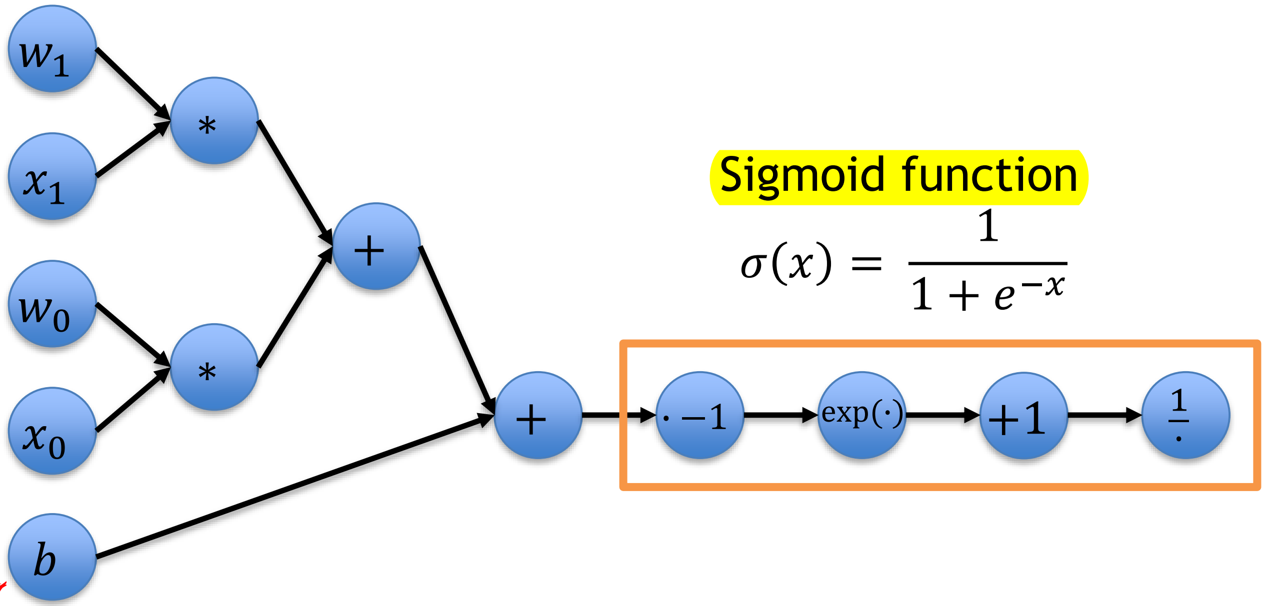
We want to compute gradients w.r.t. all weights $\boldsymbol{W}$ **AND** all biases $\boldsymbol{b}$

Really Important

# NNs as Computational Graphs

- We can express any kind of functions in a computational graph, e.g. $f(\boldsymbol{w}, \boldsymbol{x}) = \dfrac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

Understand the flow

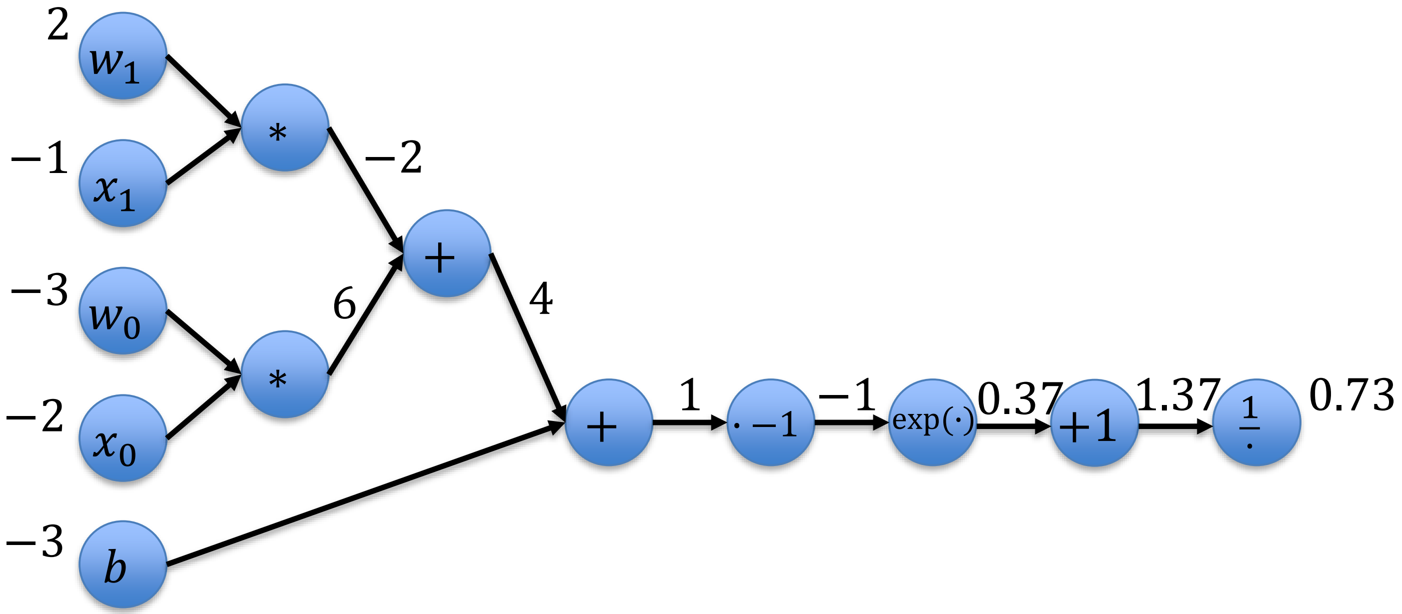



Sigmoid function

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

BIAS

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



* The node operation rep the operation in the forward pass.
We are interested in its derivative in the backward pass.

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

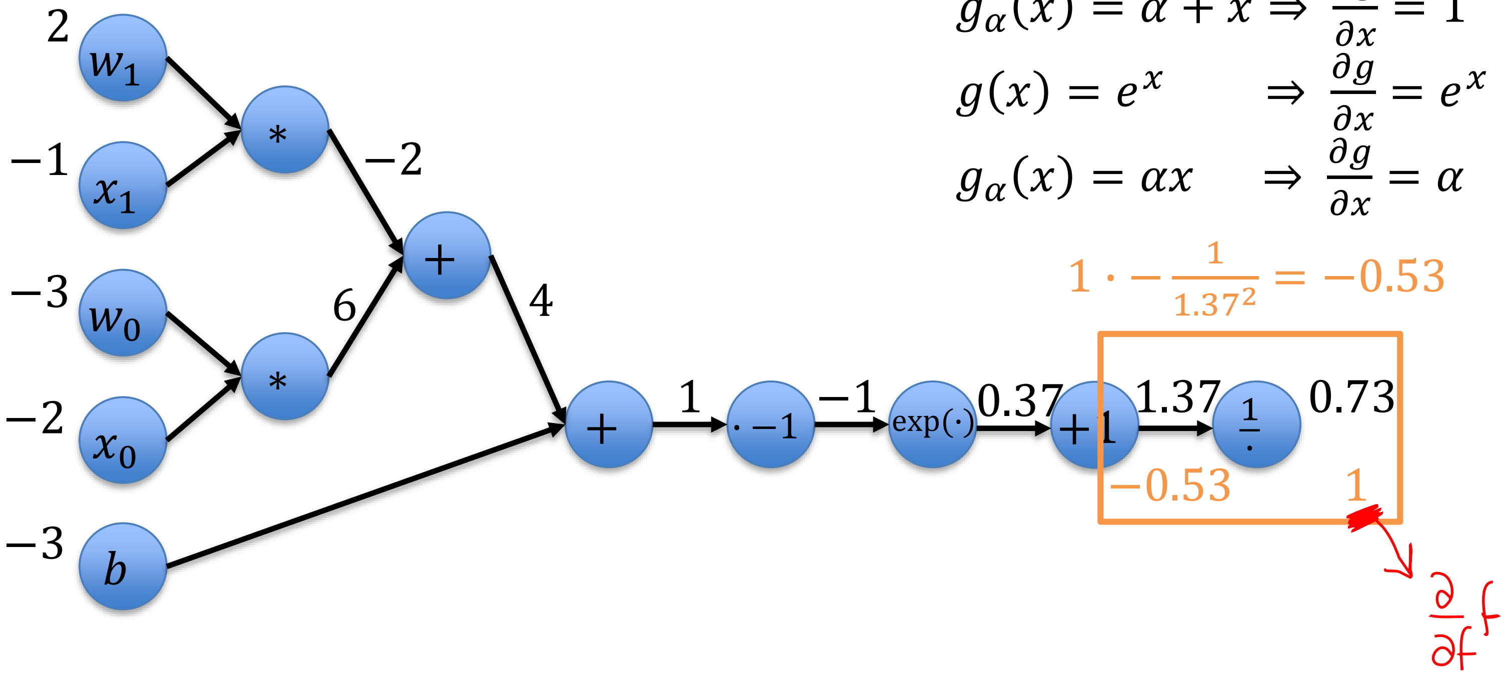
$$\boxed{g(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial g}{\partial x} = -\frac{1}{x^2}}$$

$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$

$$g(x) = e^x \quad \Rightarrow \frac{\partial g}{\partial x} = e^x$$

$$g_\alpha(x) = \alpha x \quad \Rightarrow \frac{\partial g}{\partial x} = \alpha$$

$$1 \cdot -\frac{1}{1.37^2} = -0.53$$



$x$ in the derivative is the output from the forward pass at each node

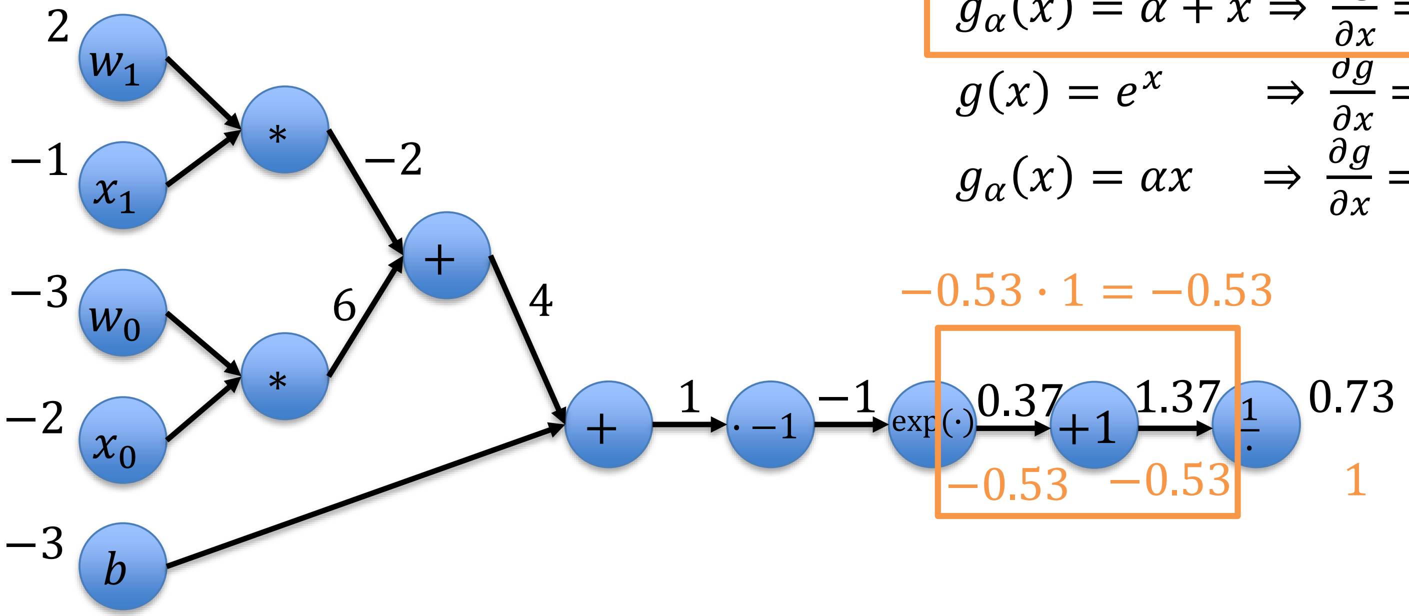
- $f(\mathbf{w}, \mathbf{x}) = \dfrac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

$$\boxed{g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1}$$

$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$

$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$



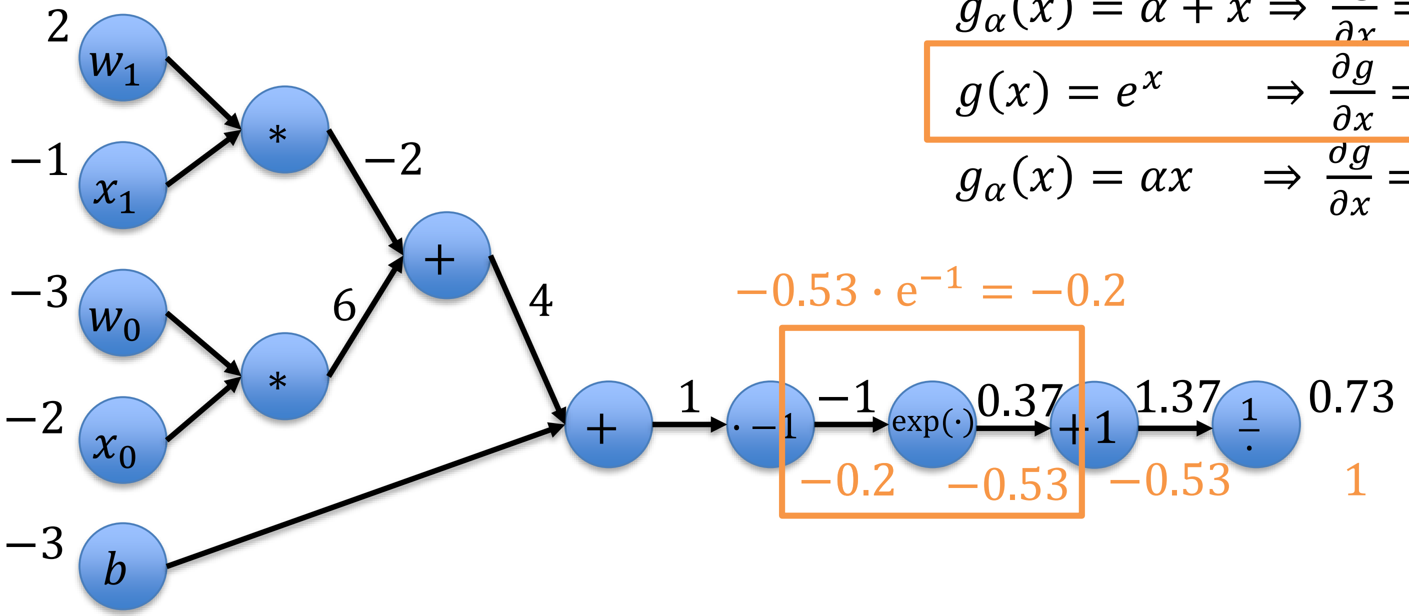
- $f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

$$g(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

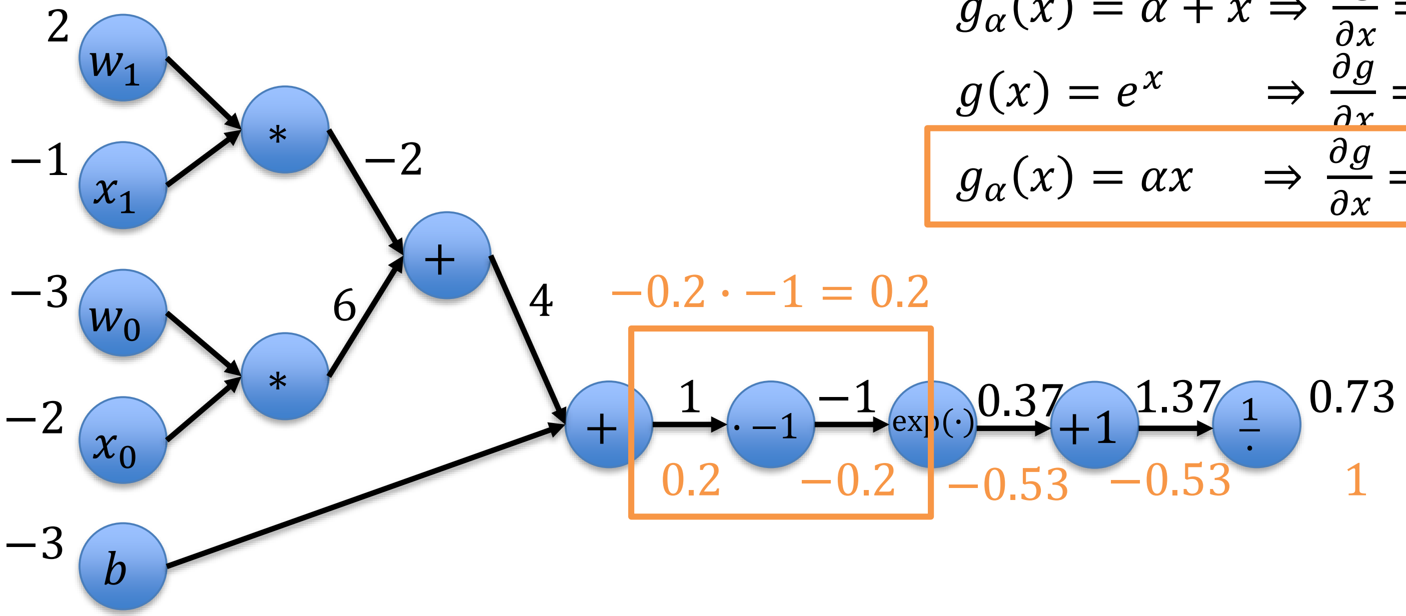
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$

$$\boxed{g(x) = e^x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = e^x}$$

$$g_\alpha(x) = \alpha x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = \alpha$$



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




$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

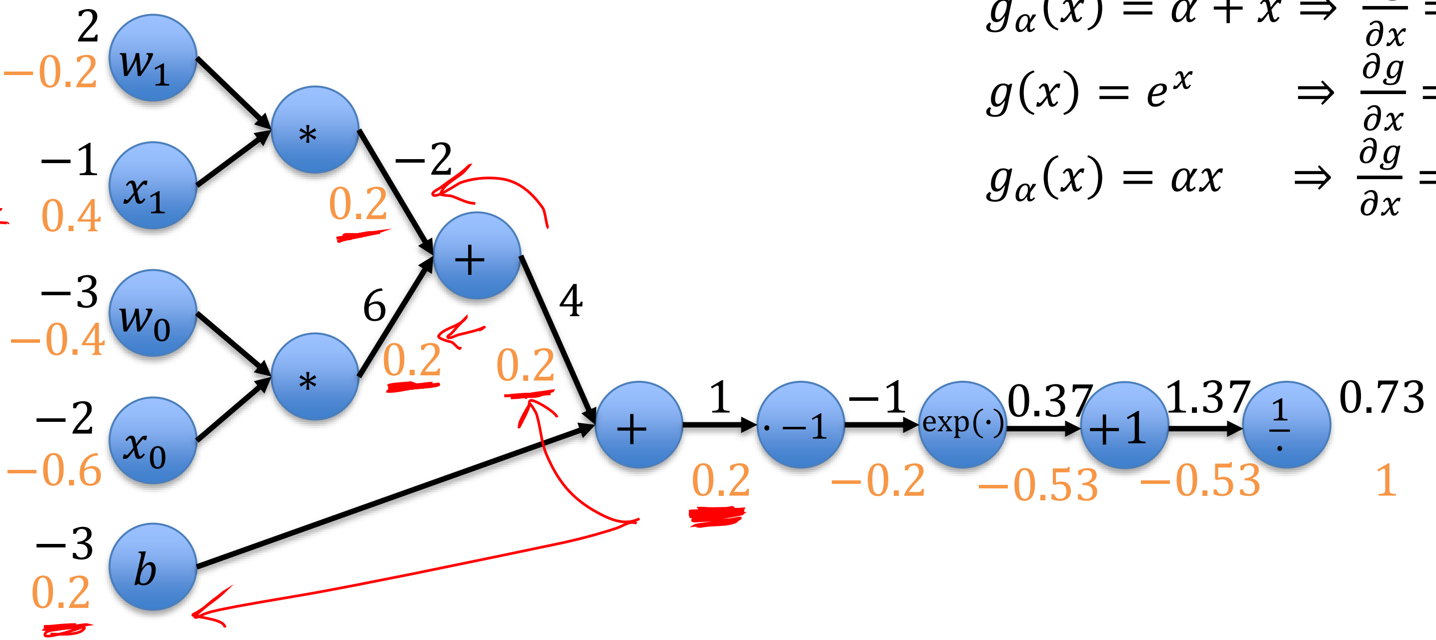
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$

$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$

$$\boxed{g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha}$$

- $f(\boldsymbol{w}, \boldsymbol{x}) = \dfrac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$
$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$




$0.2\, X_1 =$

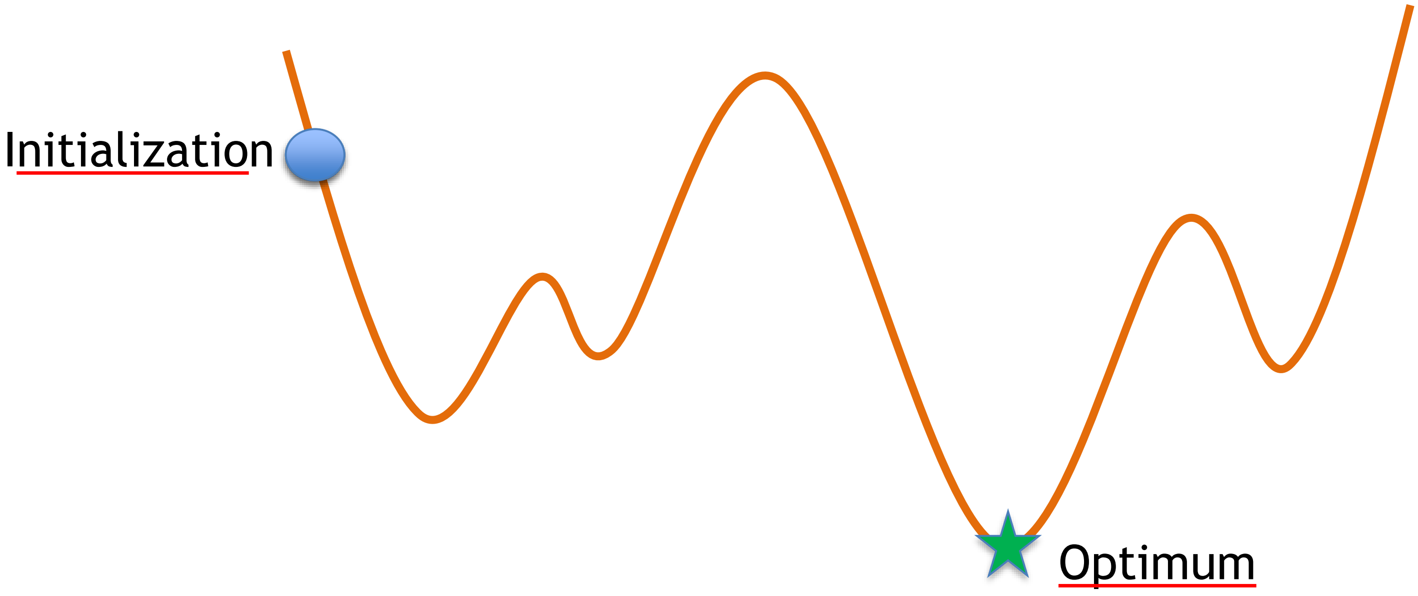
$0.2\, W_1 =$

practically, no $x$ shall be updated

# Gradient Descent

# Gradient Descent

$$\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$$



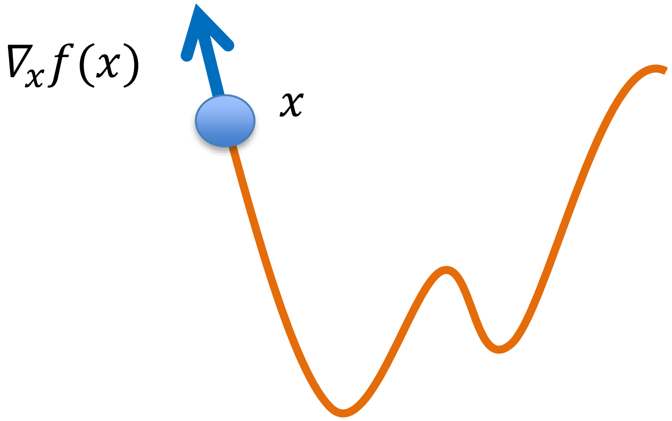
# Gradient Descent

- From derivative to gradient

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \longrightarrow \nabla_x f(x)$$

Direction of greatest increase of the function

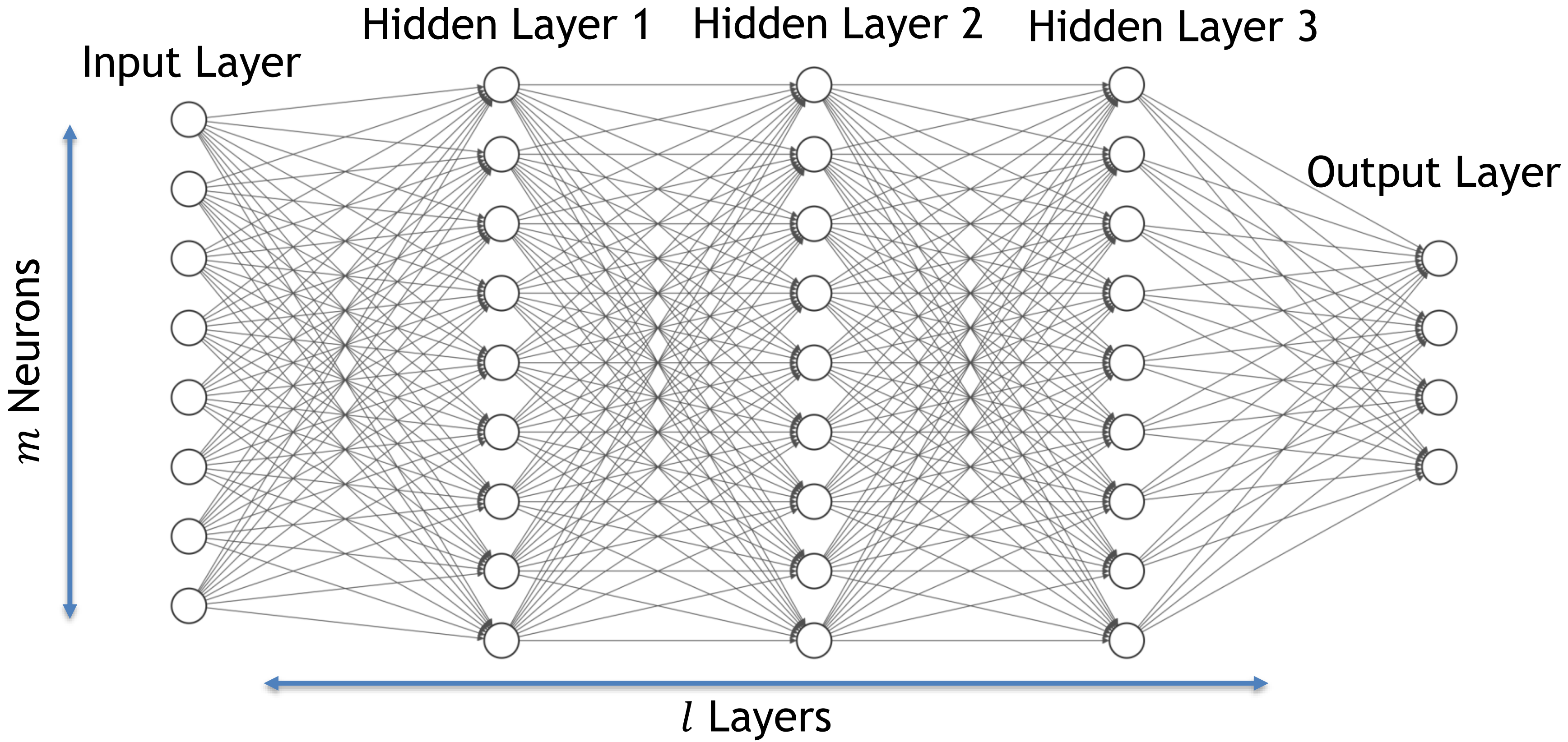
- Gradient steps in direction of negative gradient




$$x' = x - \alpha \nabla_x f(x)$$

Learning rate

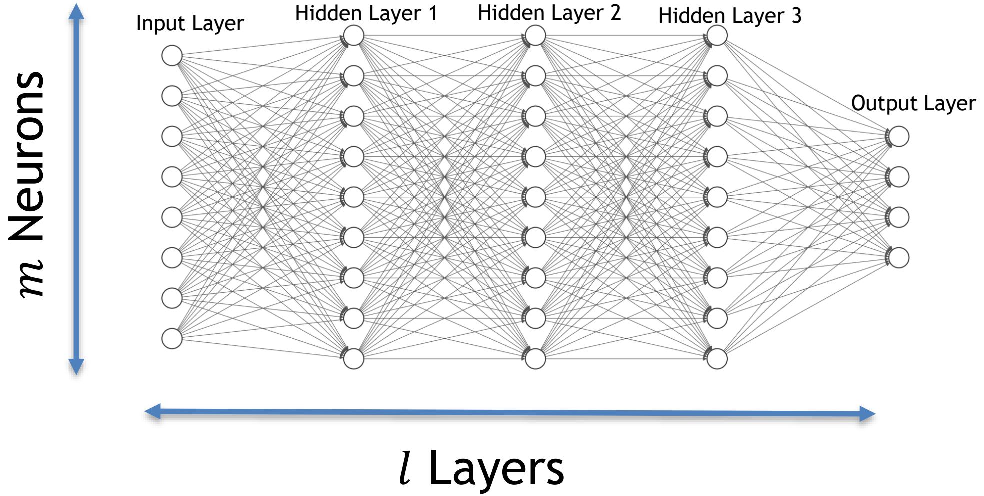
# Gradient Descent for Neural Networks



# 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:

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

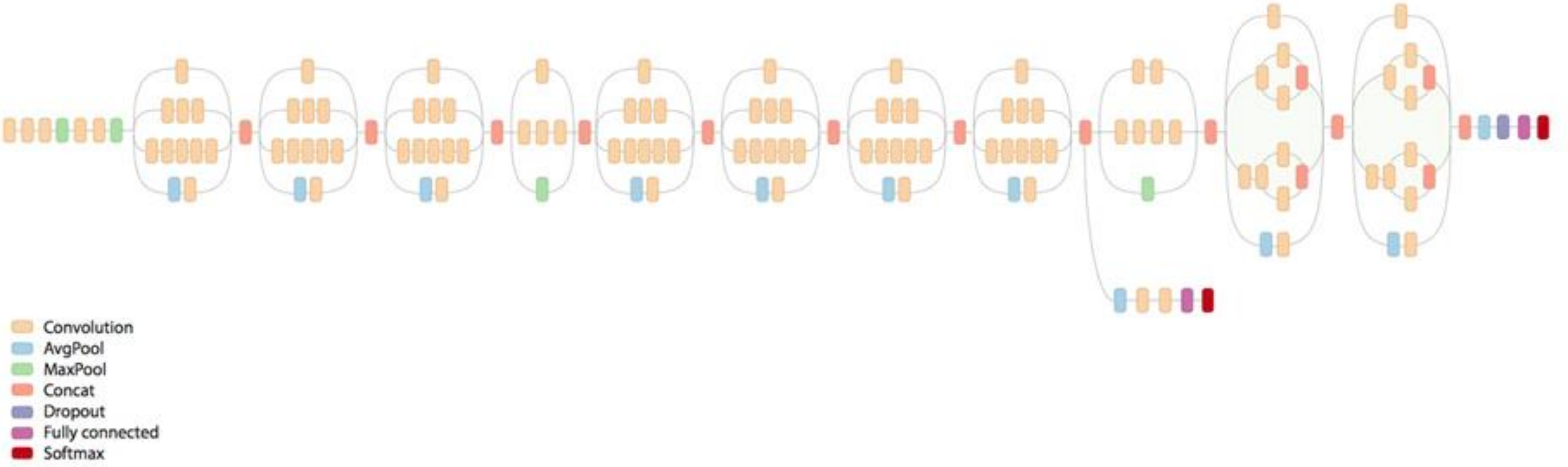



**Gradient step:**

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

# NNs can Become Quite Complex…

- These graphs can be huge!




[Szegedy et al.,CVPR'15] Going Deeper with Convolutions

# The Flow of the Gradients

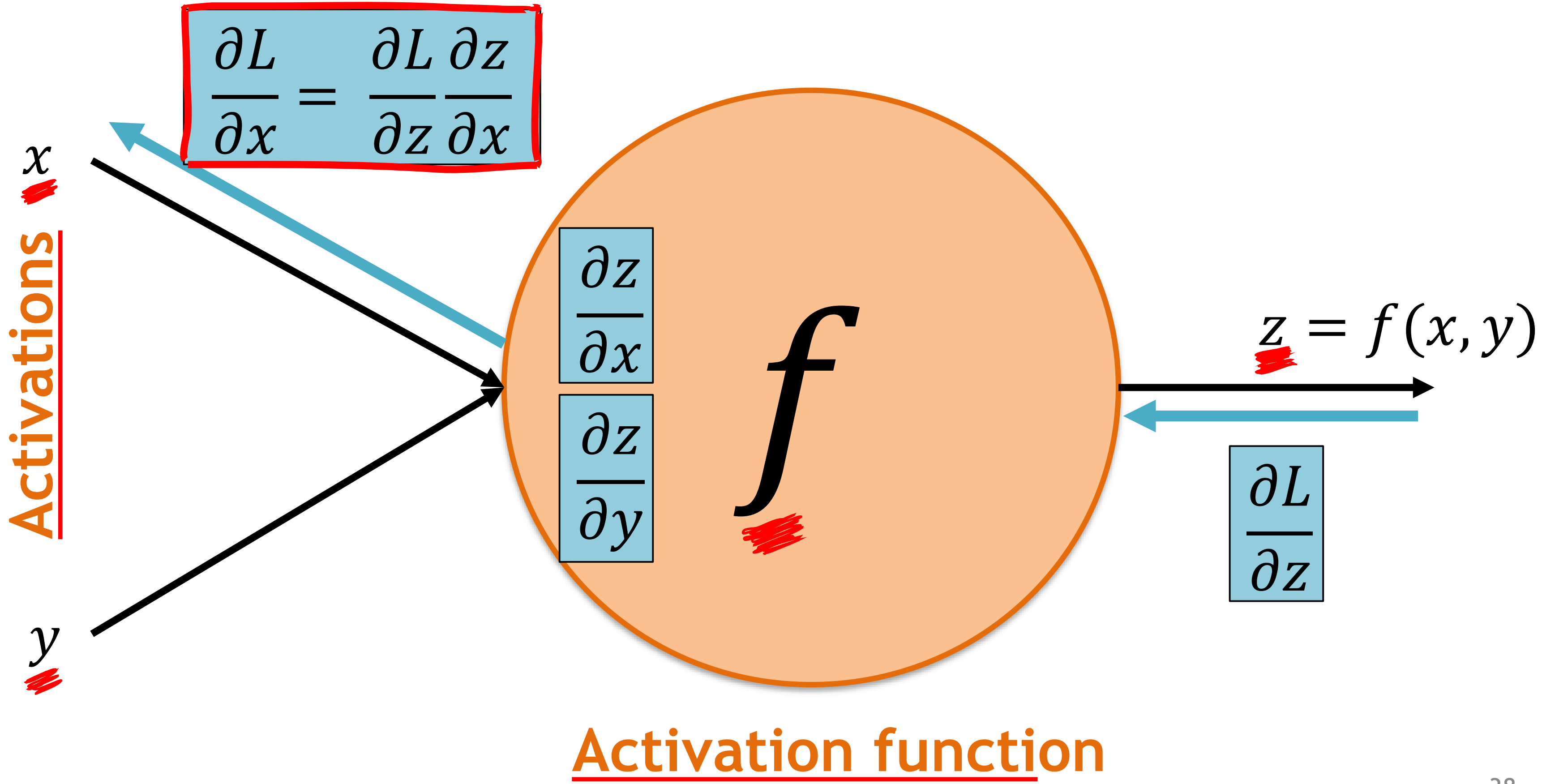
- Many many many many of these nodes form a neural network

  **NEURONS**

- Each one has its own work to do

  **FORWARD AND BACKWARD PASS**

# The Flow of the Gradients



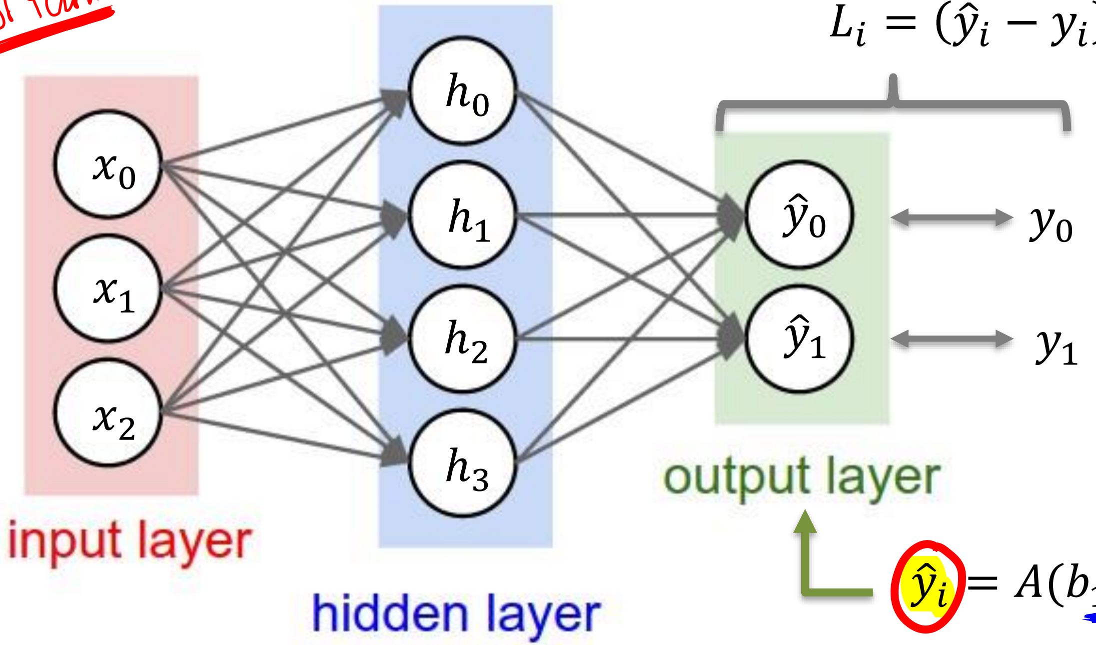
# Gradient Descent for Neural Networks

Important

Loss function

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

Do not forget bias!

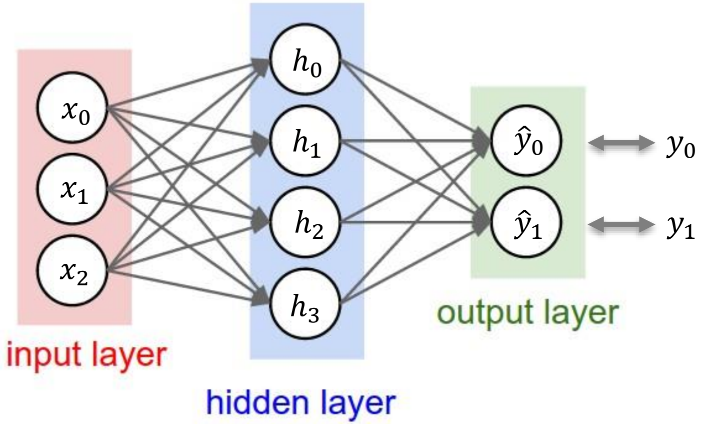



$$\hat{y}_i = A\left(b_{1,i} + \sum_j h_j w_{1,i,j}\right)$$

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

Sample

Just simple:
$A(x) = \max(0, x)$




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

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

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

# Backpropagation

Just go through layer by layer

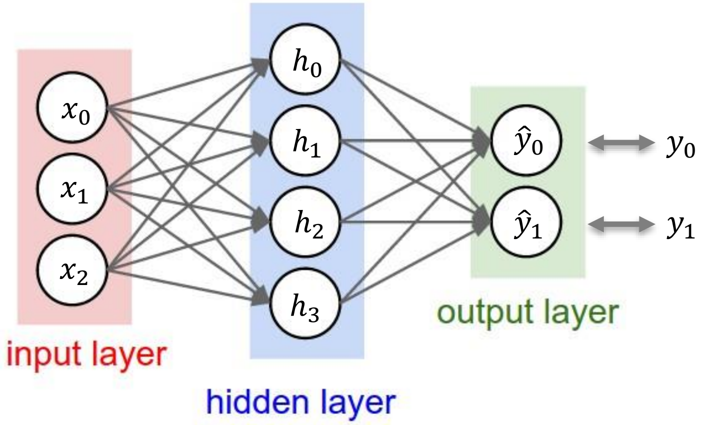
$$\frac{\partial L_i}{\partial w_{1,i,j}} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{1,i,j}}$$

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

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

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

…




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

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

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

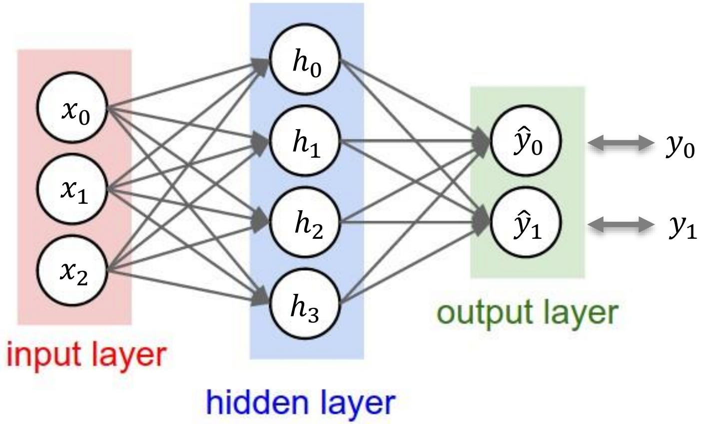
## How many unknown weights?

- 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**

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

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

output scores

**Gradients of weights of last layer:**

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

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

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

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

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

# Gradients of weights of first layer:

$$\boxed{\frac{\partial L}{\partial h_j}} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_j}\frac{\partial s_j}{\partial h_j} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial \hat{y}_i}\hat{y}_i(1-\hat{y}_i)w_{ji} = \sum_{i=1}^{n_{out}}(\hat{y}_i - y_i)w_{ji}$$

$$\boxed{\frac{\partial L}{\partial s_j^1}} = \boxed{\sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_i}\frac{\partial s_i}{\partial h_j}}\frac{\partial h_j}{\partial s_j^1} = \boxed{\sum_{i=1}^{n_{out}}(\hat{y}_i - y_i)w_{ji}}\underline{\left(h_j(1-h_j)\right)}$$

derivative of sigmoid act.
of hidden layer

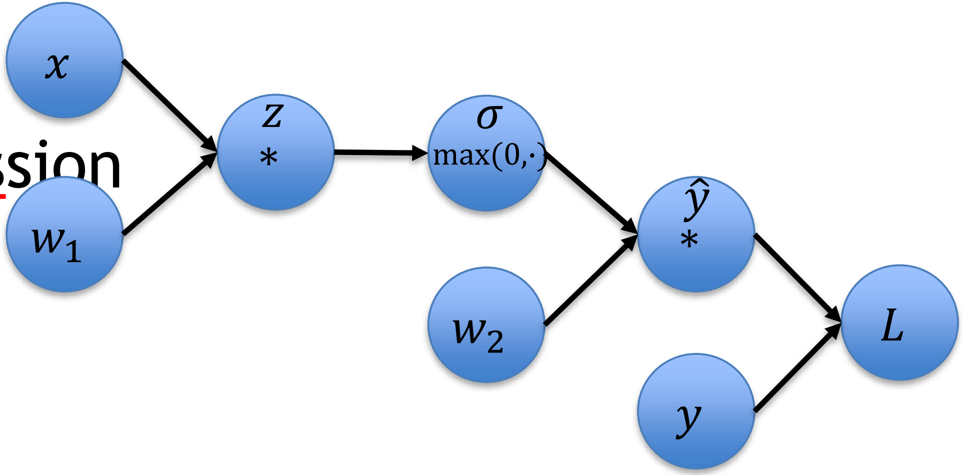
$$\frac{\partial L}{\partial w_{kj}^1} = \boxed{\sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_j^1}}\frac{\partial s_j^1}{\partial w_{kj}^1} = \boxed{\sum_{i=1}^{n_{out}}(\hat{y}_i - y_i)w_{ji}\left(h_j(1-h_j)\right)}x_k$$

# Back to Compute Graphs & NNs

NB

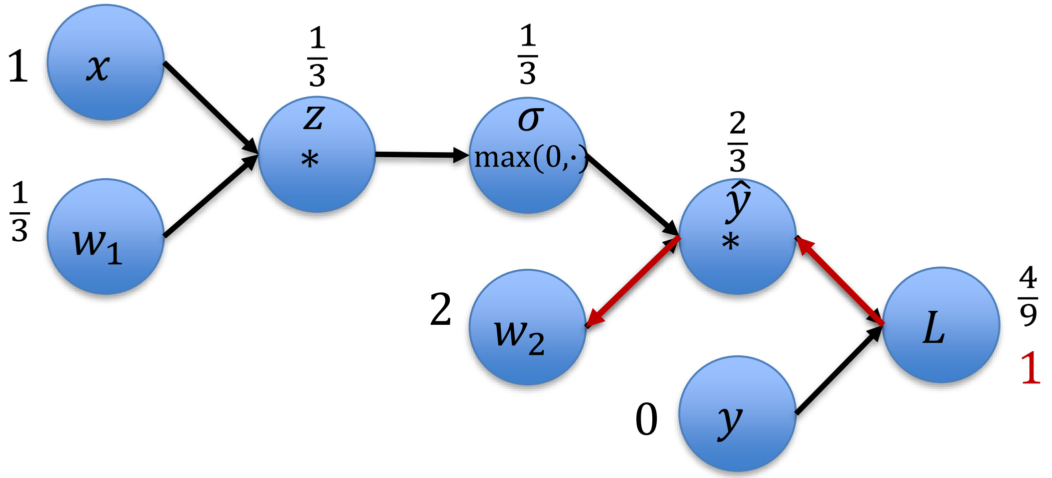
- Inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Two-layer NN for regression with ReLU activation
- Function we want to optimize:

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



# 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 $n, d = 1$:

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

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

Backpropagation

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

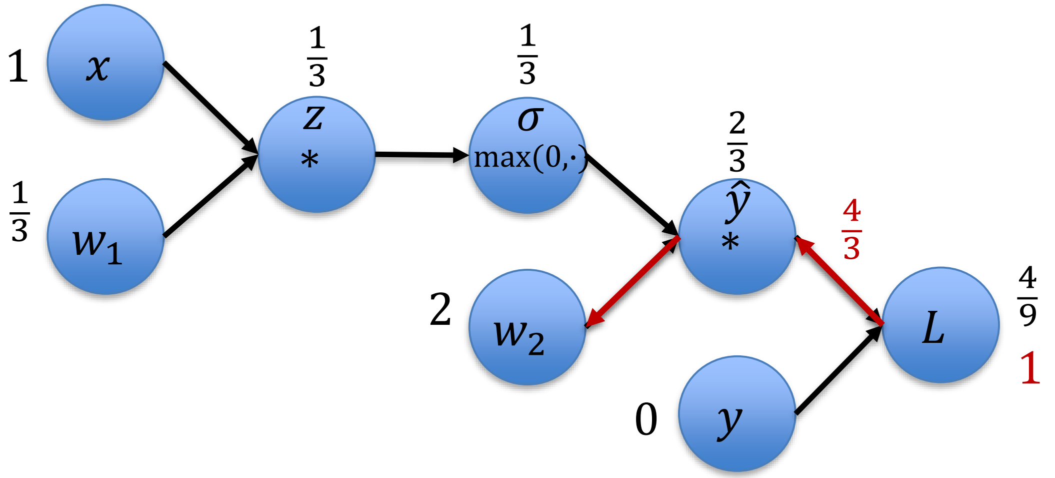
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

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

In our case $n, d = 1$:

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

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




Backpropagation

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

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

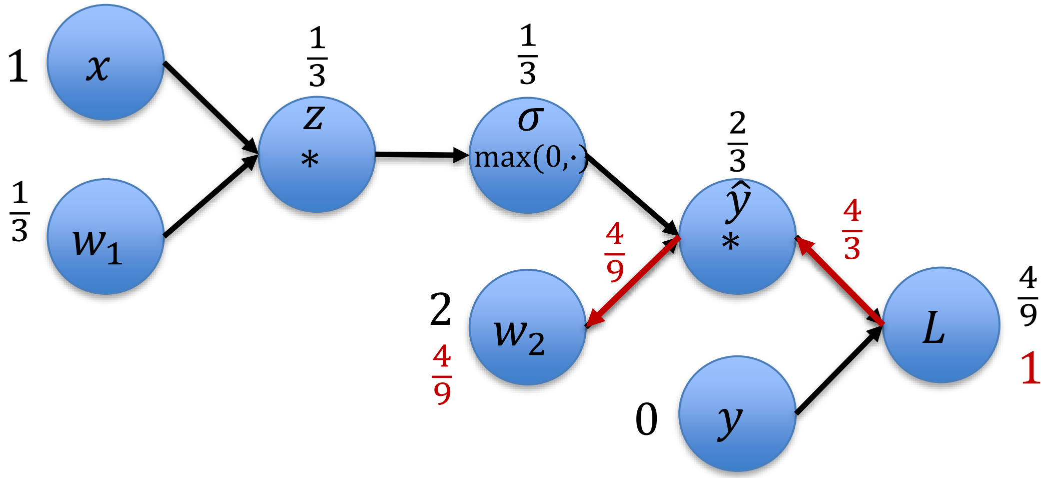
Initialize $x = 1,\ y = 0,$
$w_1 = \frac{1}{3}, w_2 = 2$

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

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

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

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

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

In our case $n, d = 1$:

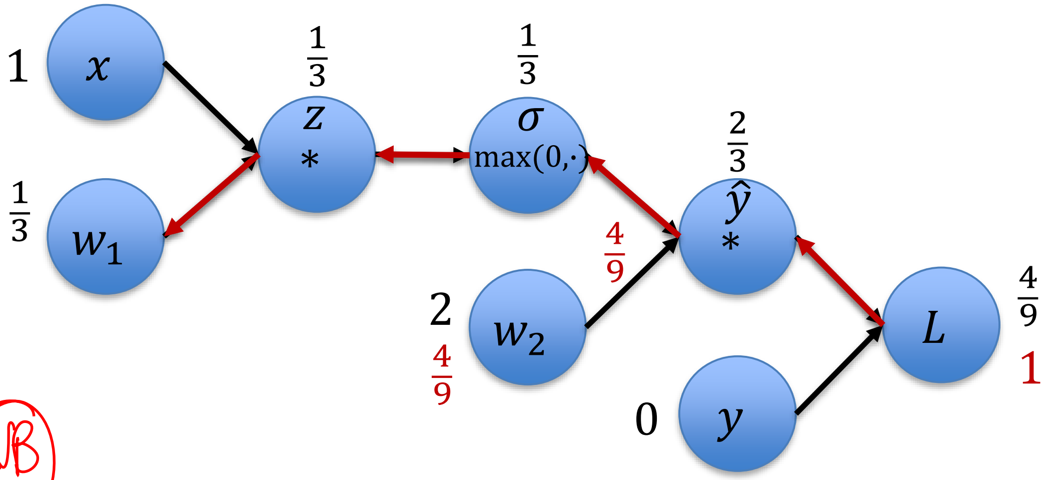
$$L = (\hat{y} - y)^2 \quad \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

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

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

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

NB




Backpropagation

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

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

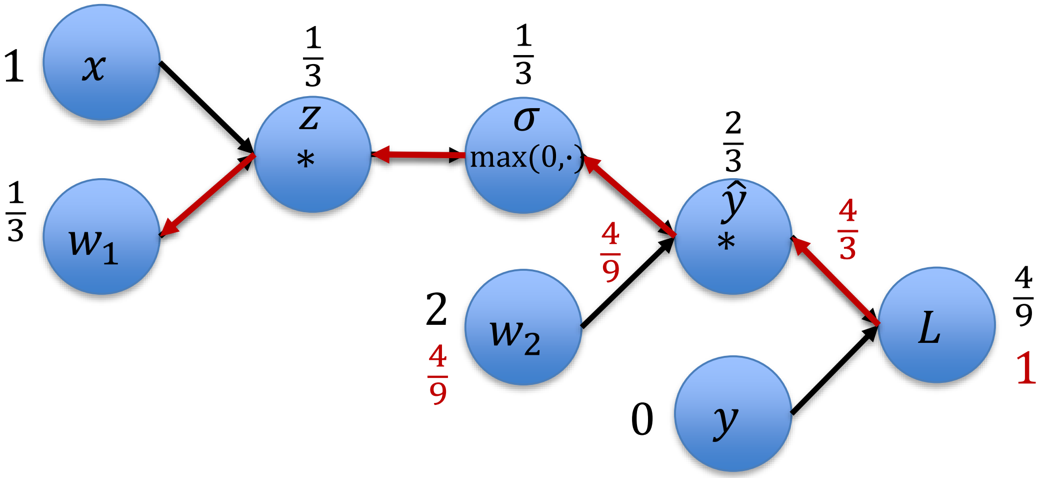
In our case $n, d = 1$:

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

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

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

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




Backpropagation

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

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

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

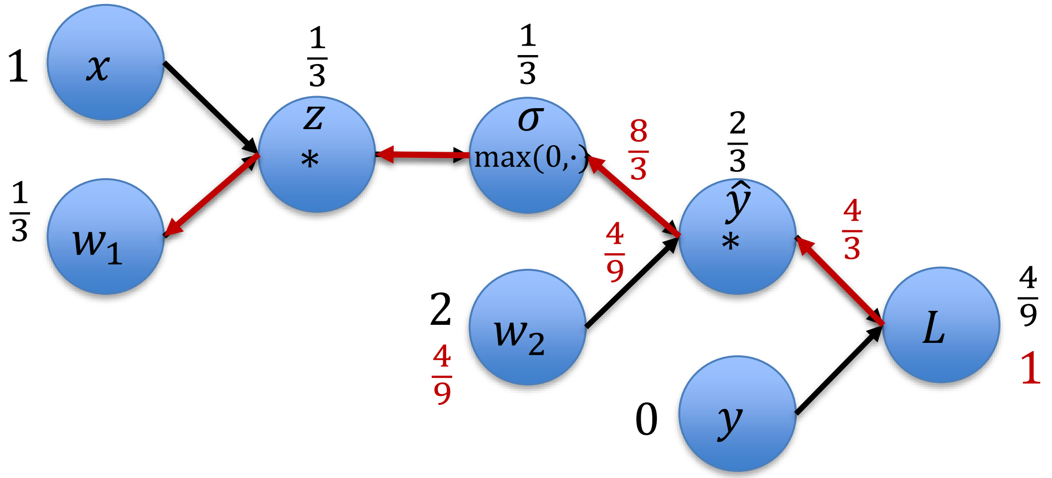
In our case $n, d = 1$:

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

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

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

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




Backpropagation

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

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

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

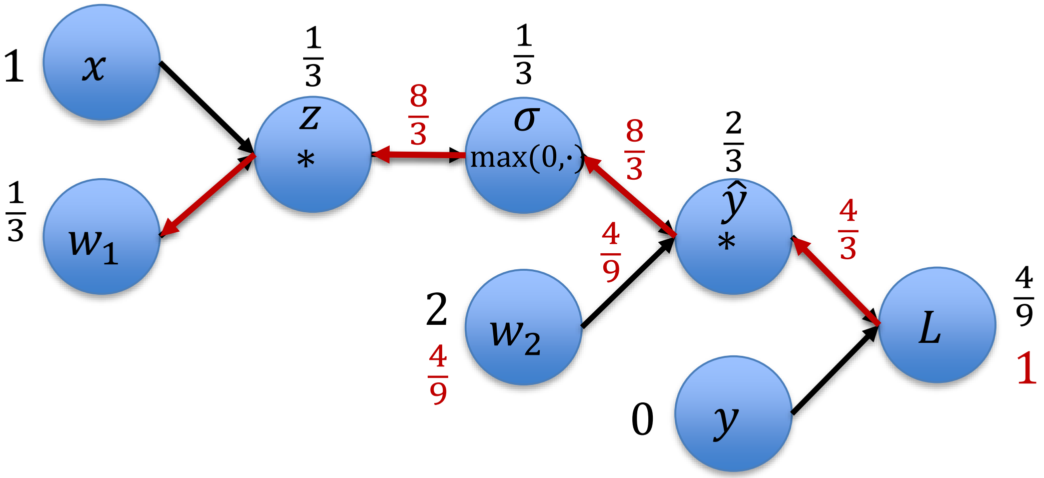
In our case $n, d = 1$:

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

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

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

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




Backpropagation

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

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

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

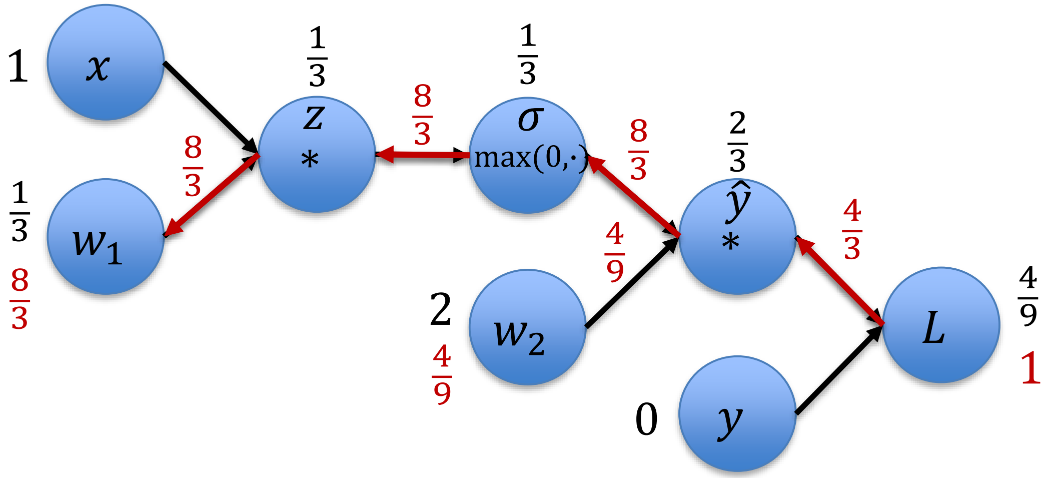
In our case $n, d = 1$:

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

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

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

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




Backpropagation

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

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

Important

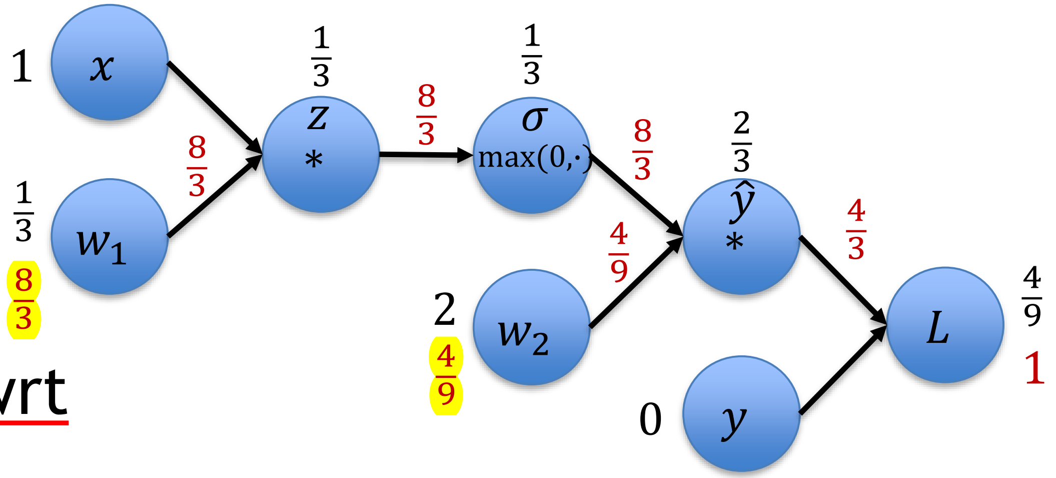
- Function we want to optimize:

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

- Computed gradients wrt to weights $w_1$ and $w_2$
- Now: update the weights

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

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




But: how to choose a good learning rate $\alpha$?

# Gradient Descent

- How to pick good learning rate?

- How to compute gradient for single training pair?

- How to compute gradient for large training set?

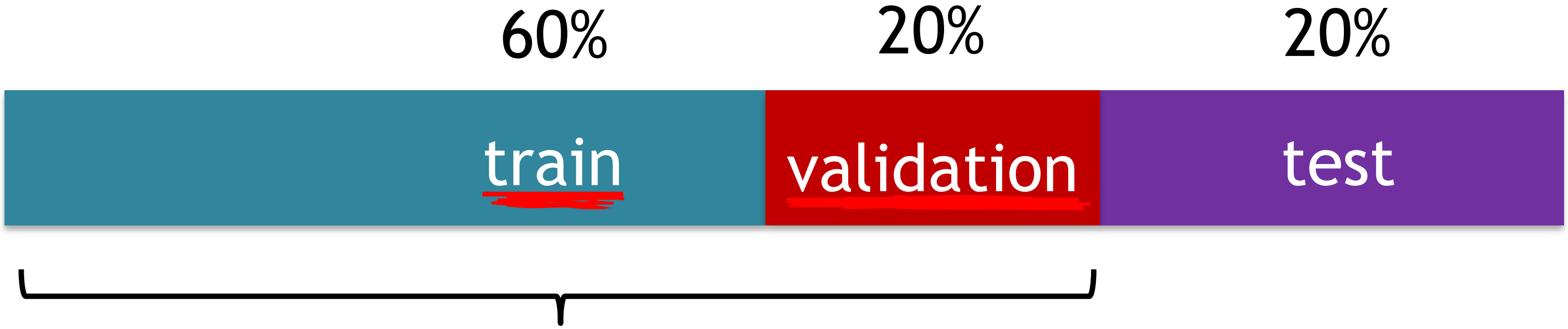
averaging grad over multiple training samples

- How to speed things up? More to see in next lectures…

# Regularization

# Recap: Basic Recipe for ML

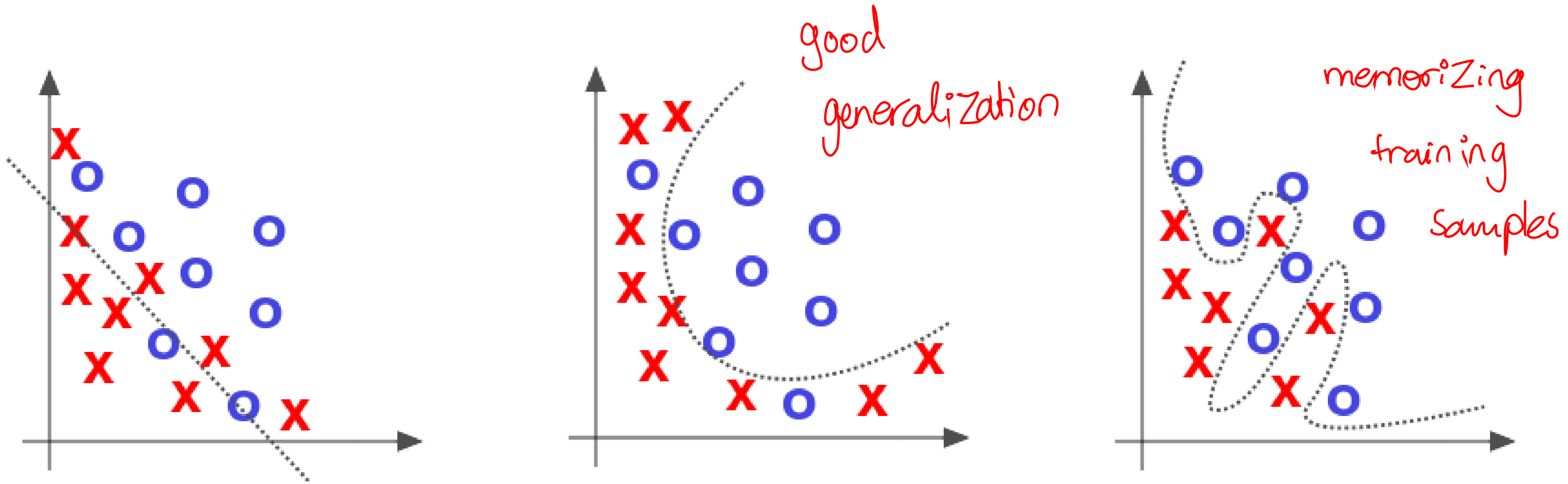
- Split your data




Find your hyperparameters

Other splits are also possible (e.g., 80%/10%/10%)

# Over- and Underfitting

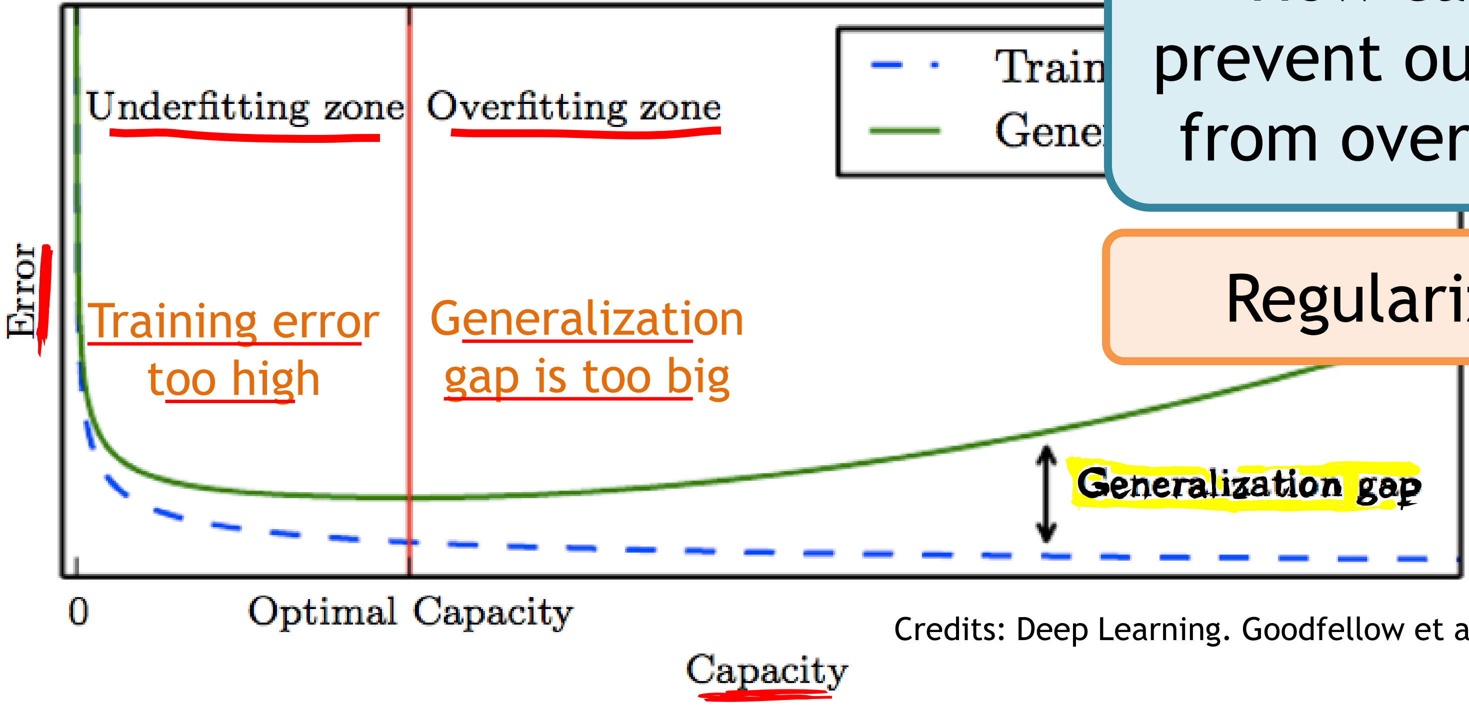



**Underfitted** **Appropriate** **Overfitted**

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

# Training a Neural Network

- Training/ Validation curve




How can we prevent our model from overfitting?

Regularization

Credits: Deep Learning. Goodfellow et al.

# Regularization

- Loss function $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

- Regularization techniques
  - L2 regularization
  - L1 regularization
  - Max norm regularization
  - Dropout
  - Early stopping
  - ...

Add **regularization term** to loss function

More details later

* We aim to make training harder; so that the net can learn better features

# Regularization: Example

Important

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$
- Two linear classifiers that give the same result:
- $\theta_1 = [0, 0.75, 0]$ → Ignores 2 features
- $\theta_2 = [0.25, 0.5, 0.25]$ → Takes information from all features

better Potential to generalize

- L2 tends to shrink coeff evenly.
- L2 is useful when dealing with collinear/dependent features.

- Loss $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n}\left(x_i\theta_{ji} - y_i\right)^2 \boxed{+ \lambda R(\boldsymbol{\theta})}$
- L2 regularization $R(\boldsymbol{\theta}) = \sum_{i=1}^{n} \theta_i^2$

$$\theta_1 \longrightarrow 0 + 0.75^2 + 0 = 0.5625$$
$$\theta_2 \longrightarrow 0.25^2 + 0.5^2 + 0.25^2 = \boxed{0.375}$$ Minimization

$$x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25]$$

NB L2 favors $\theta_2$, which is more sparse

- L1 tends to shrink coeff to zero
- L1 is useful for few feature selection; dropping any var associated with coeff that go to zero.

- Loss $L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(x_i \theta_{ji} - y_i\right)^2 \boxed{+ \lambda R(\boldsymbol{\theta})}$

- L1 regularization $R(\boldsymbol{\theta}) = \sum_{i=1}^{n} |\theta_i|$

$$\theta_1 \longrightarrow 0 + 0.75 + 0 = 0.75$$
$$\theta_2 \longrightarrow 0.25 + 0.5 + 0.25 = \boxed{1}$$ Minimization

$$x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25]$$

NB L1 favors $\Theta_1$

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ $\longrightarrow$ Ignores 2 features

$\theta_2 = [0.25, 0.5, 0.25]$ $\longrightarrow$ Takes information from all features

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ → L1 regularization enforces **sparsity**

$\theta_2 = [0.25, 0.5, 0.25]$ → Takes information from all features

Conclusion

- Input: 3 features $\boldsymbol{x} = [1, 2, 1]$

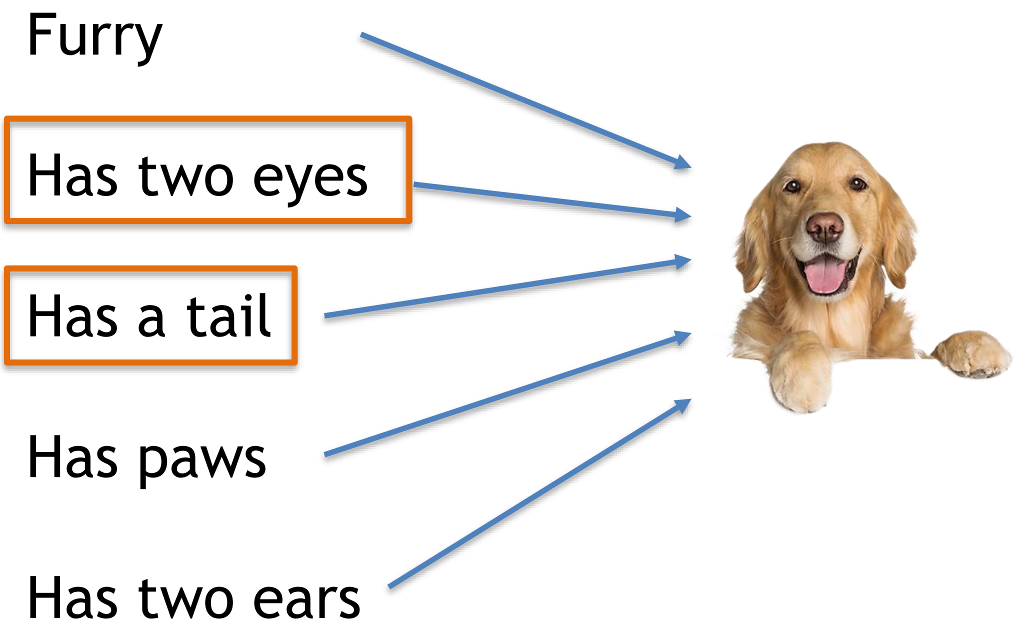
- Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$ → L1 regularization enforces **sparsity**

$\theta_2 = [0.25, 0.5, 0.25]$ → L2 regularization enforces that the weights have **similar values**

# Regularization: Effect

- Dog classifier takes different inputs




**L1 regularization** will focus all the attention to a few key features

- Dog classifier takes different inputs

Furry

Has two eyes

Has a tail

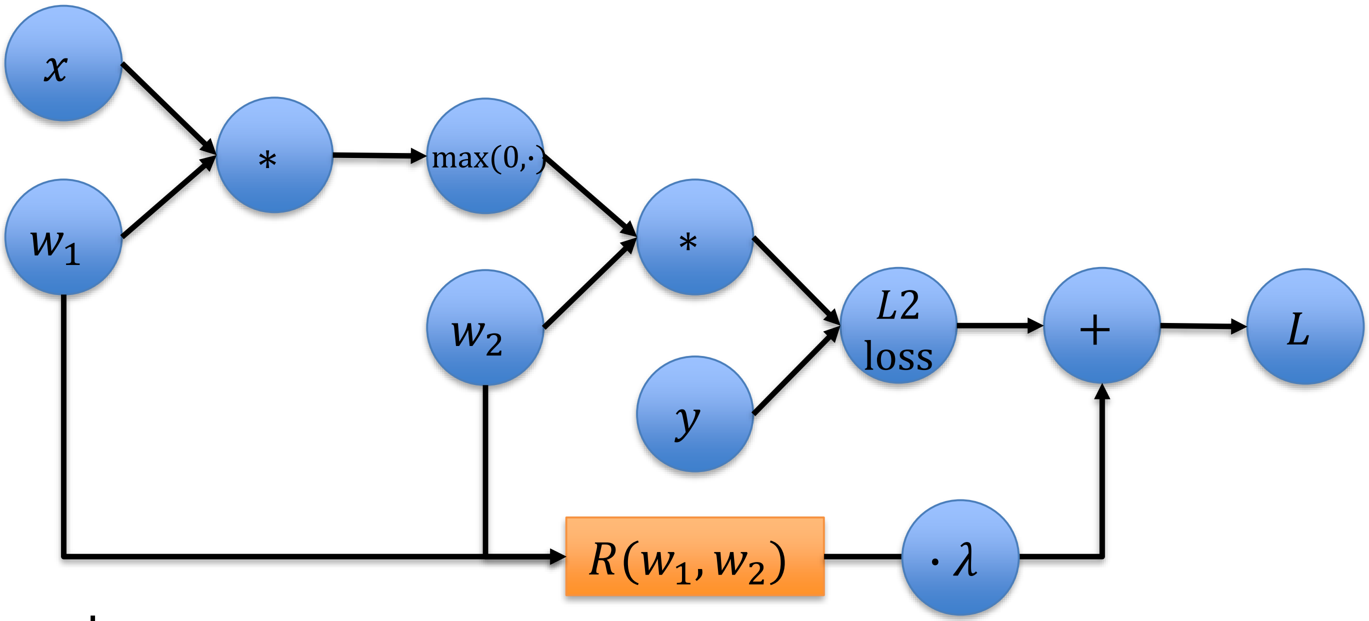
Has paws

Has two ears

**L2 regularization** will take all information into account to make decisions

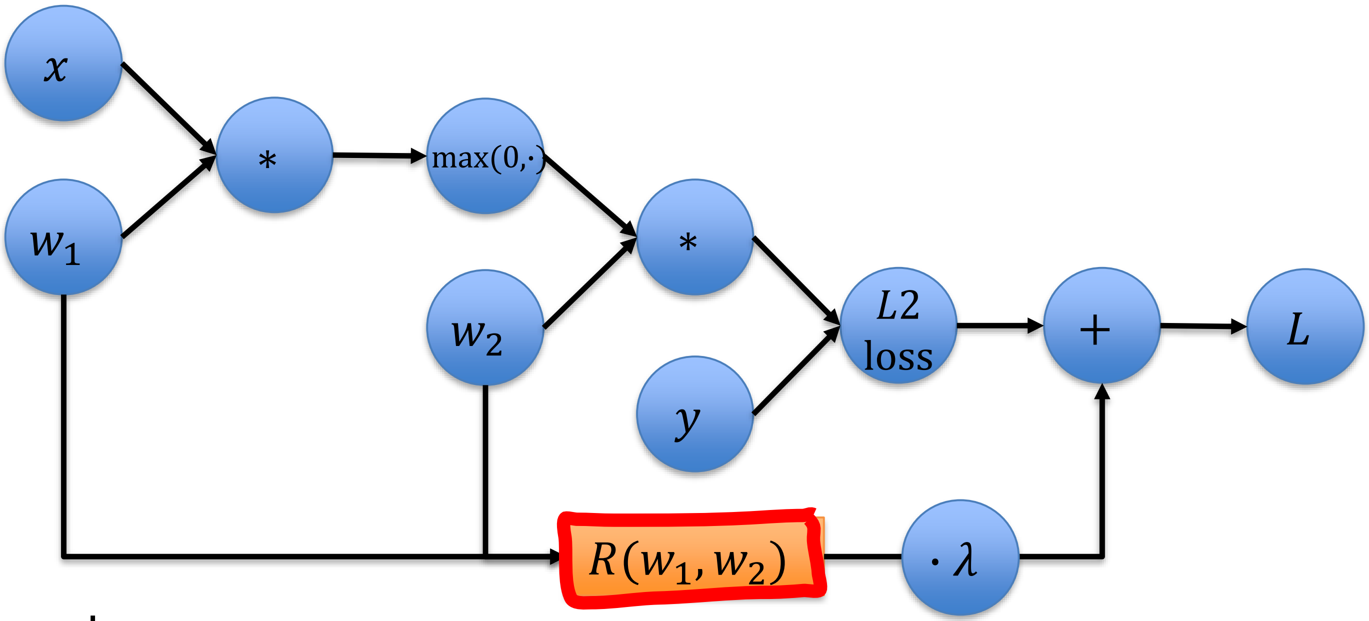
i.e more robust for generalization

# Regularization for Neural Networks



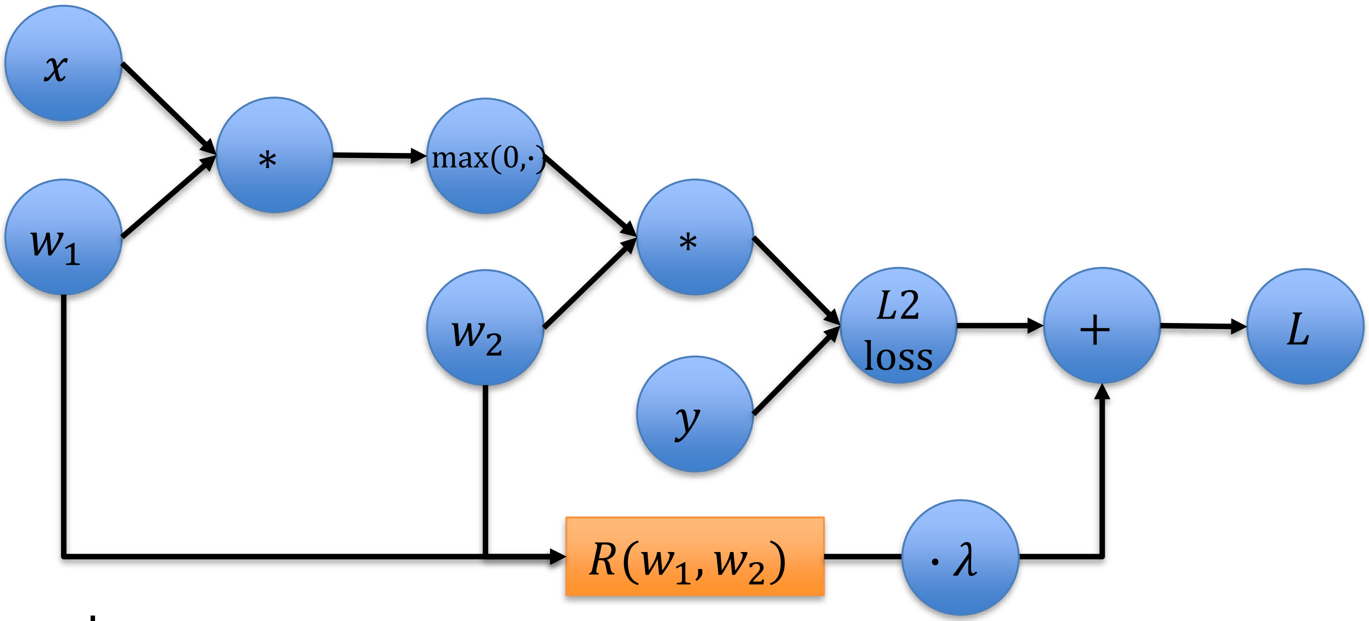

Combining nodes:
Network output + L2-loss + regularization

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




Combining nodes:
Network output + L2-loss + regularization

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2 + \lambda \left\| \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \right\|_2^2$$




Combining nodes:
Network output + L2-loss + regularization

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

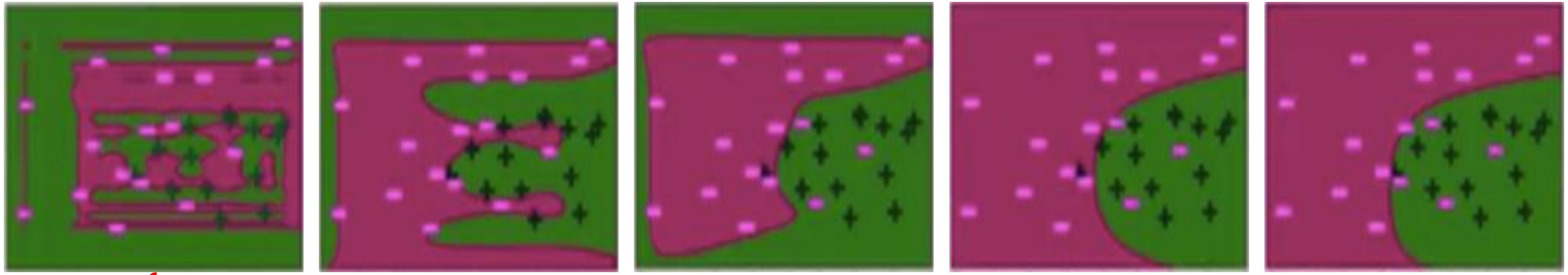
Understand

# Regularization

as Smoothing

**Regularization** $\lambda = 0$ $\lambda = 0.00001$ $\lambda = 0.001$ $\lambda = 1$ $\lambda = 10$

**Decision Boundary**



overfitting

'Oversmoothing cause weights to be similar i.e nothing learnt

What is the goal of regularization?

What happens to the training error?

* A sign of big reg. → train & val curves roughly the same, yet they do not go all the way till end

- Any strategy that aims to

Lower validation error

Increasing training error

* A sign of small reg → train curve goes down quickly, while val does not [overfitting]

# Next Lecture

- This week:
  - Check exercises
  - Check office hours ☺

- Next lecture
  - Optimization of Neural Networks
  - In particular, introduction to SGD (our main method!)

# See you next week ☺

# Further Reading

- Backpropagation
  - Chapter 6.5 (6.5.1 - 6.5.3) in http://www.deeplearningbook.org/contents/mlp.html
  - Chapter 5.3 in Bishop, Pattern Recognition and Machine Learning
  - http://cs231n.github.io/optimization-2/
- Regularization
  - Chapter 7.1 (esp. 7.1.1 & 7.1.2) http://www.deeplearningbook.org/contents/regularization.html
  - Chapter 5.5 in Bishop, Pattern Recognition and Machine Learning