

# Optimization and Backpropagation

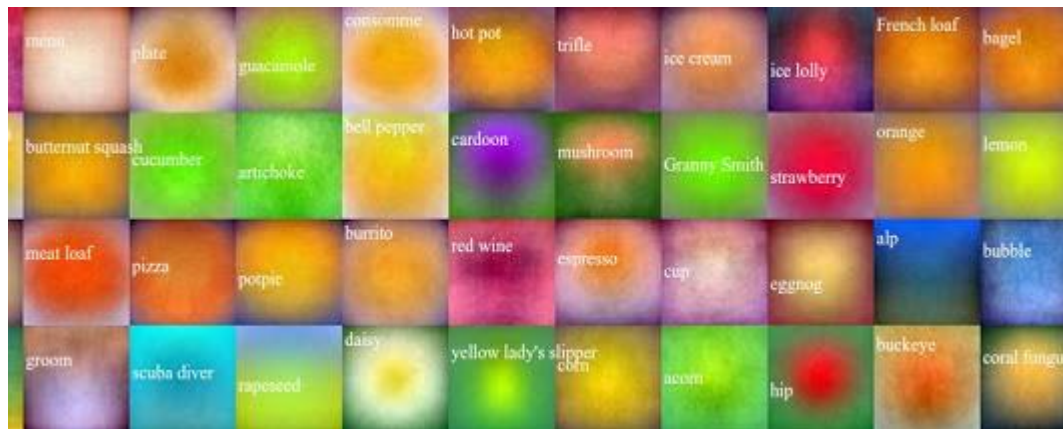
# Lecture 3 Recap

# Neural Network

- Linear score function  $f = Wx$



On CIFAR-10



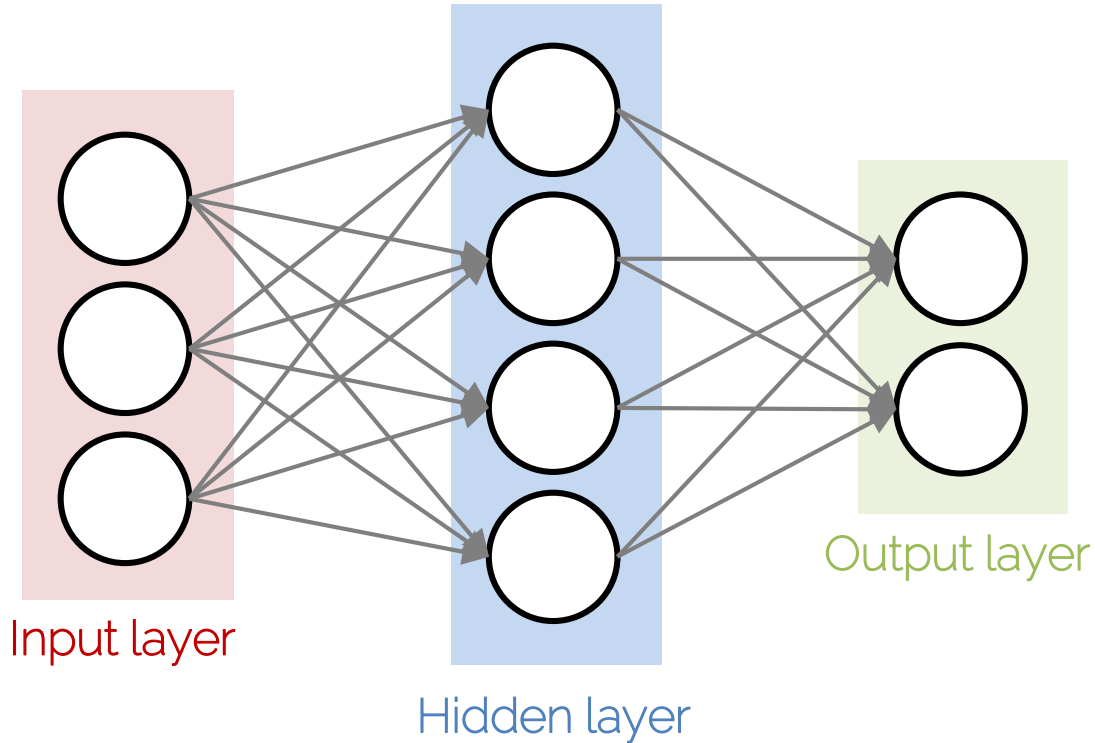
On ImageNet

Credit:  
Li/Karpathy/Johnson

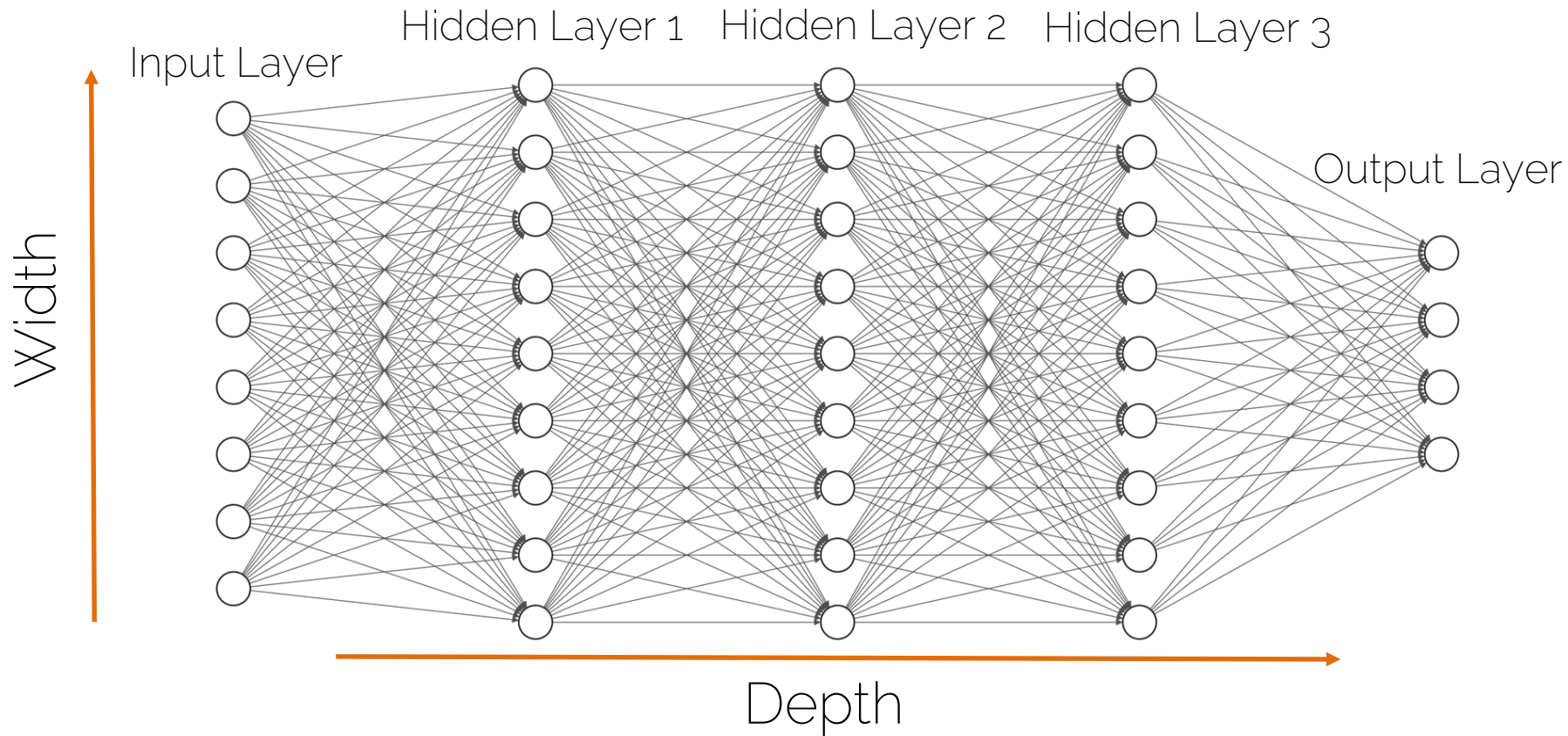
# Neural Network

- Linear score function  $\mathbf{f} = \mathbf{W}\mathbf{x}$
- Neural network is a nesting of “functions” (linear functions with nonlinearities)
  - 2-layers:  $\mathbf{f} = \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x})$
  - 3-layers:  $\mathbf{f} = \mathbf{W}_3 \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))$
  - 4-layers:  $\mathbf{f} = \mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x})))$
  - 5-layers:  $\mathbf{f} = \mathbf{W}_5 \sigma(\mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))))$
  - ... up to hundreds of layers

# Neural Network

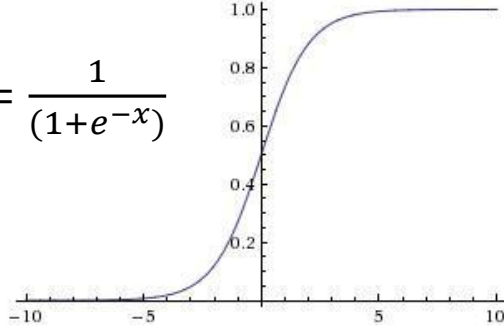


# Neural Network

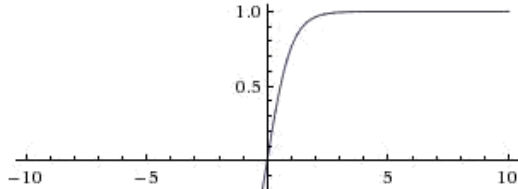


# Activation Functions

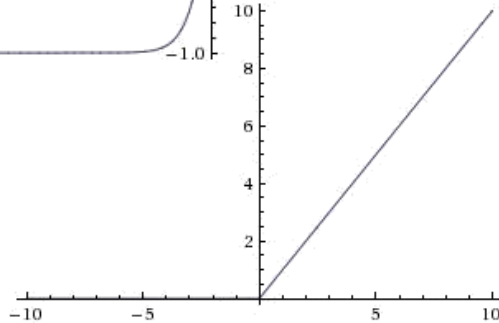
Sigmoid:  $\sigma(x) = \frac{1}{(1+e^{-x})}$



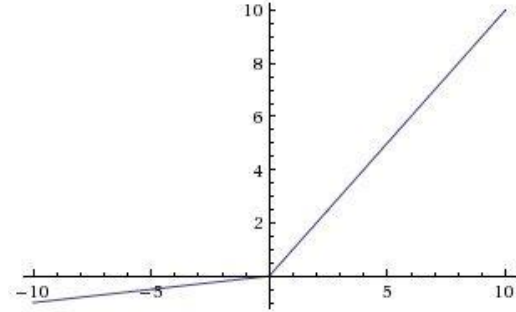
tanh:  $\tanh(x)$



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

# Loss Functions

- Measure the goodness of the predictions (or equivalently, the network's performance)
- Regression loss
  - L1 loss  $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}; \boldsymbol{\theta}) = \frac{1}{n} \sum_i^n \|y_i - \hat{y}_i\|_1$
  - MSE loss  $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}; \boldsymbol{\theta}) = \frac{1}{n} \sum_i^n \|y_i - \hat{y}_i\|_2^2$
- Classification loss (for multi-class classification)
  - Cross Entropy loss  $E(\mathbf{y}, \hat{\mathbf{y}}; \boldsymbol{\theta}) = - \sum_{i=1}^n \sum_{k=1}^K (y_{ik} \cdot \log \hat{y}_{ik})$

# Computational Graphs

- Neural network is a computational graph

- It has compute nodes



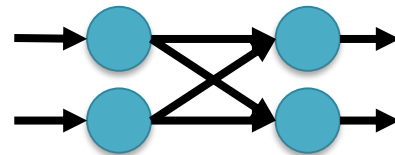
- It has edges that connect nodes



- It is directional



- It is organized in 'layers'



# Backprop

# The Importance of Gradients

- Our optimization schemes are based on computing gradients

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

- One can compute gradients analytically but what if our function is too complex?
- Break down gradient computation

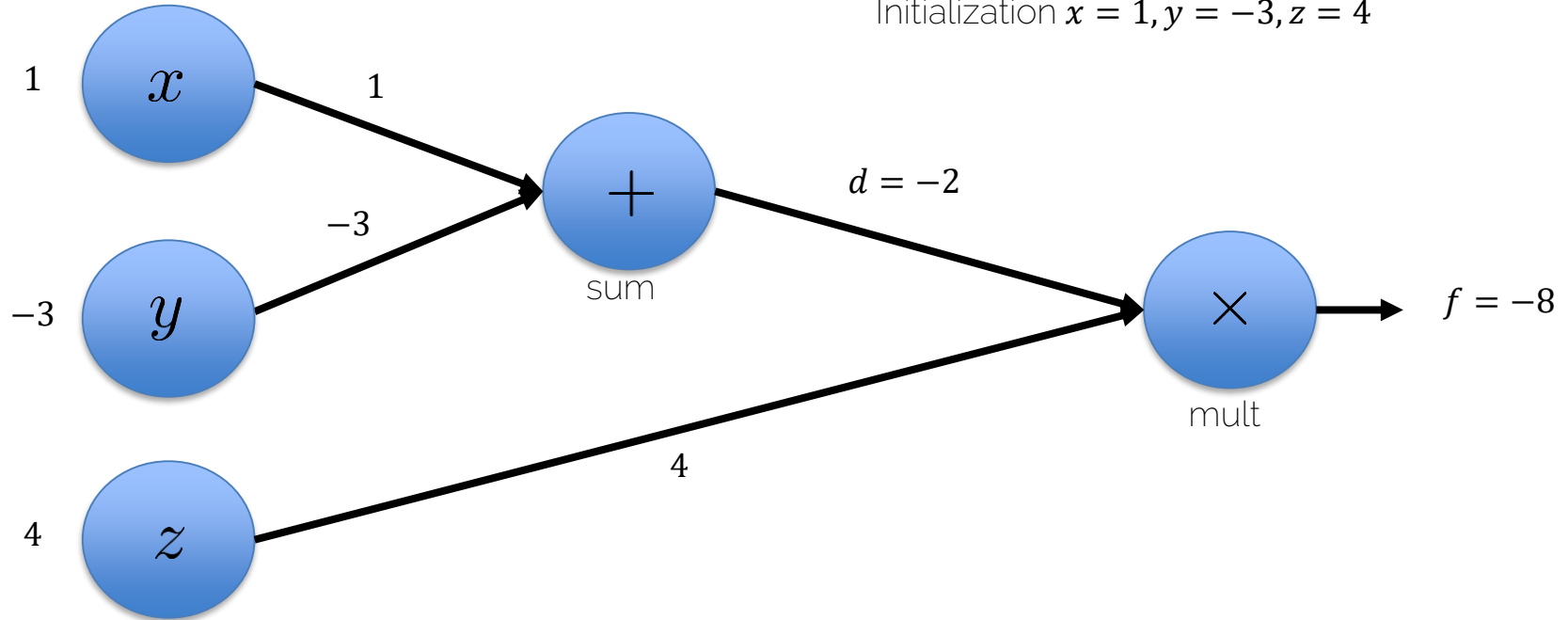
Backpropagation

Done by many people before, but often credited to Rumelhart 1986

# Backprop: Forward Pass

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

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



# Backprop: Backward Pass

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

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

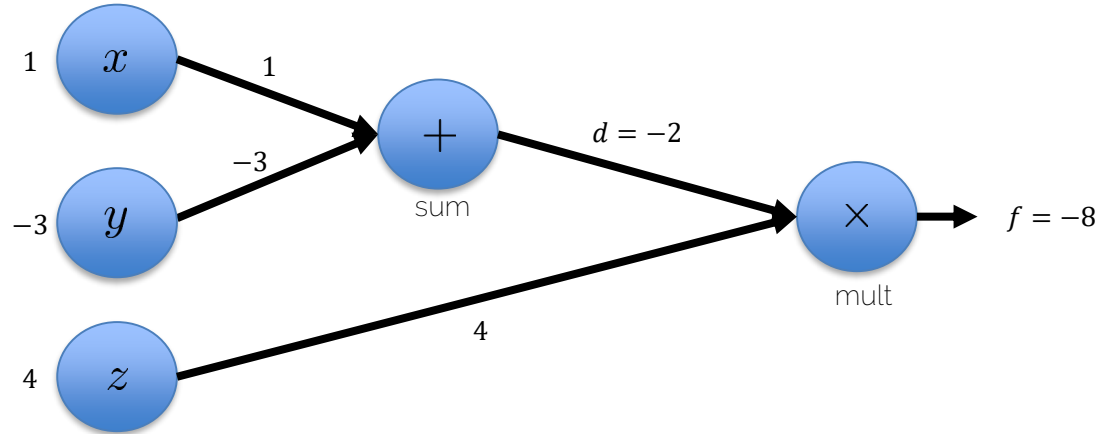
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

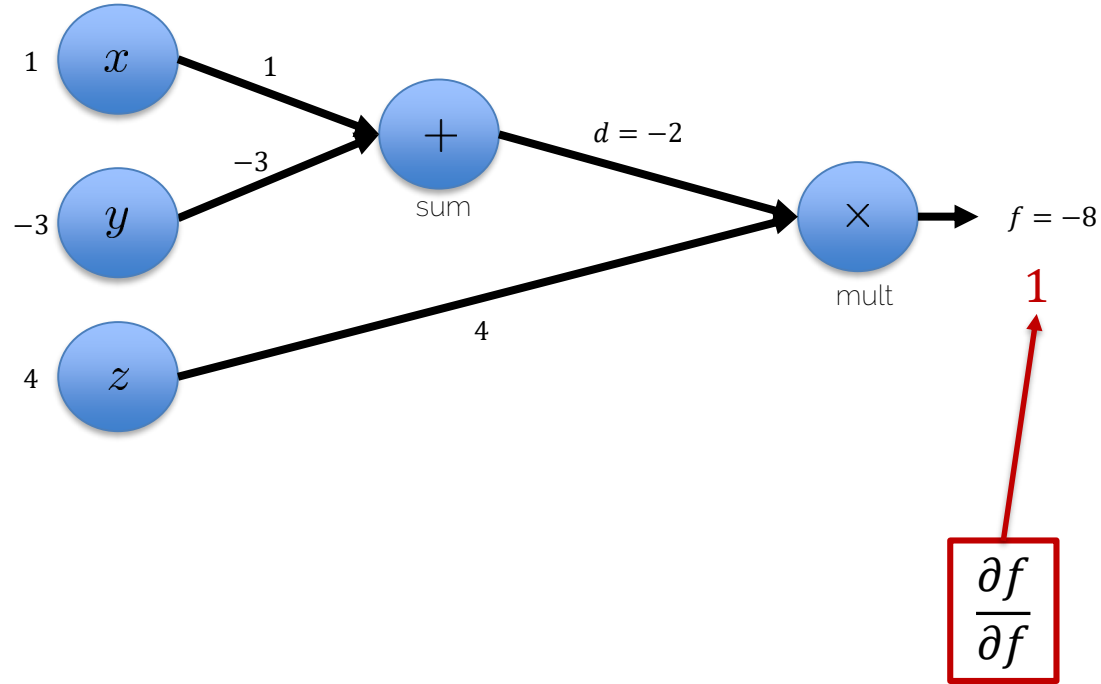
$$d = x + y$$

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

$$f = d \cdot z$$

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

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



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

# Backprop: Backward Pass

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

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

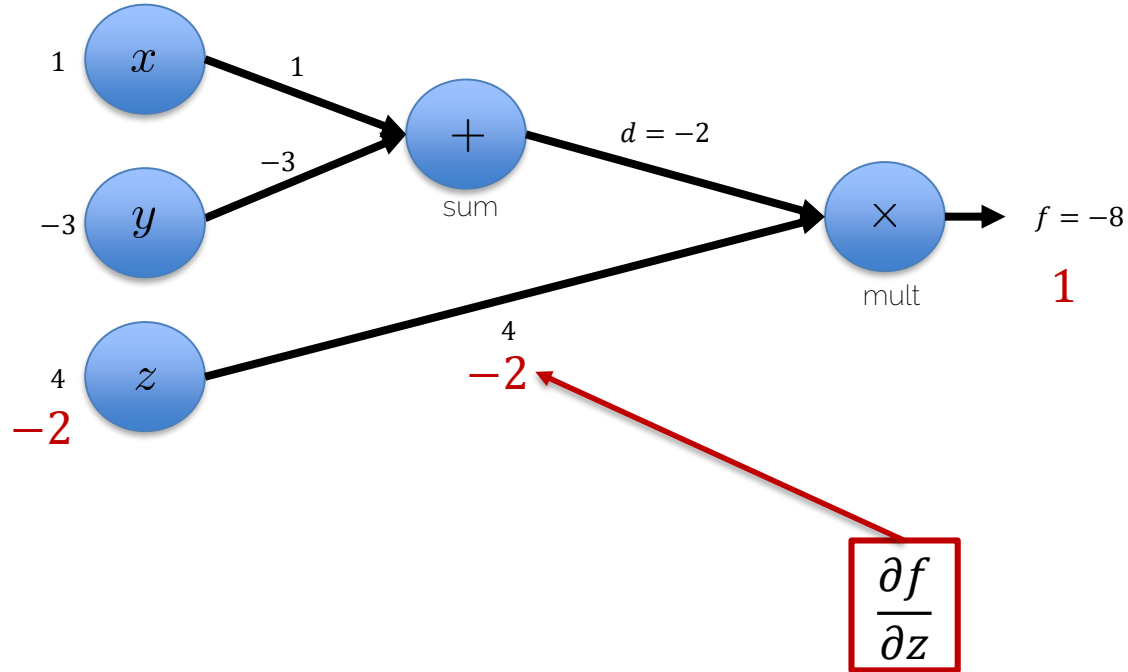
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

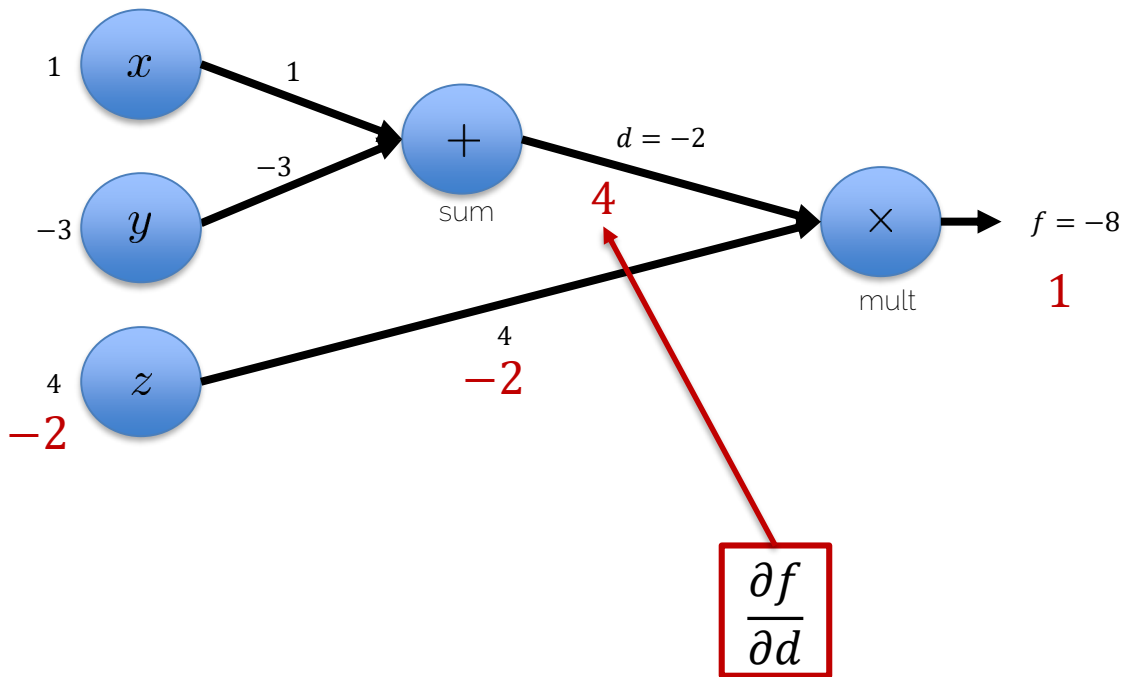
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

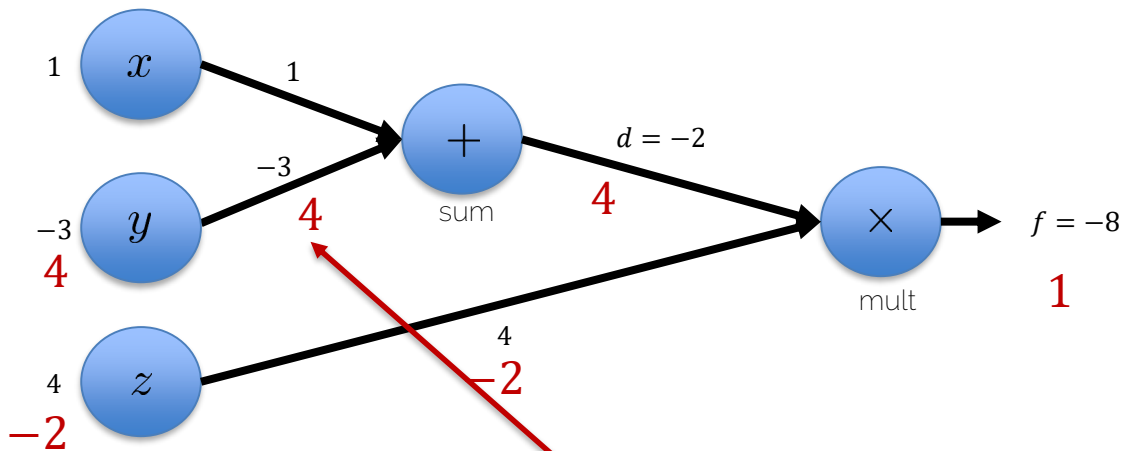
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

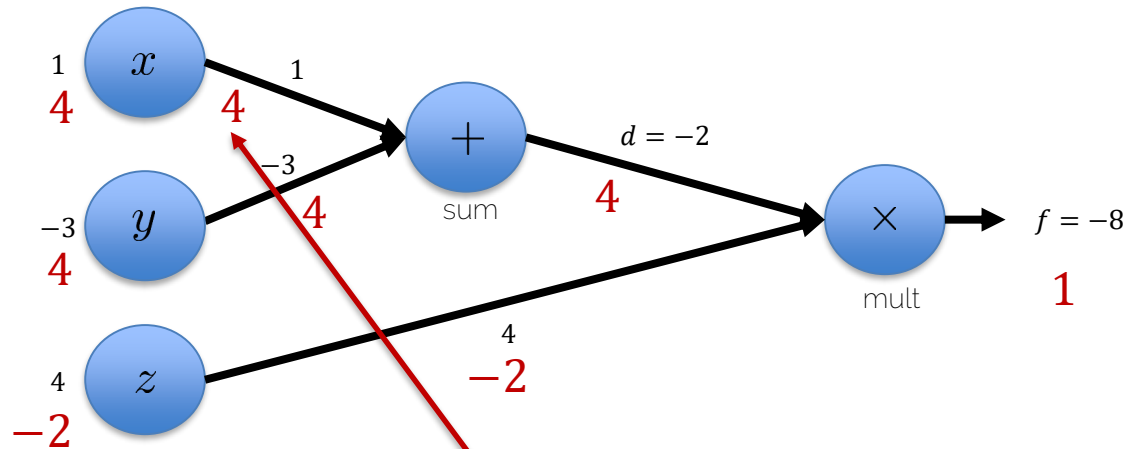
$$d = x + y$$

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

$$f = d \cdot z$$

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

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



Chain Rule:

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

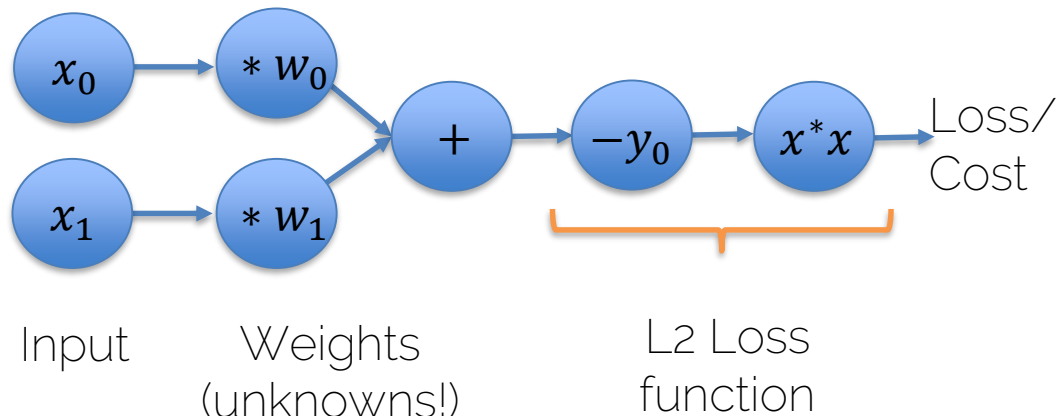
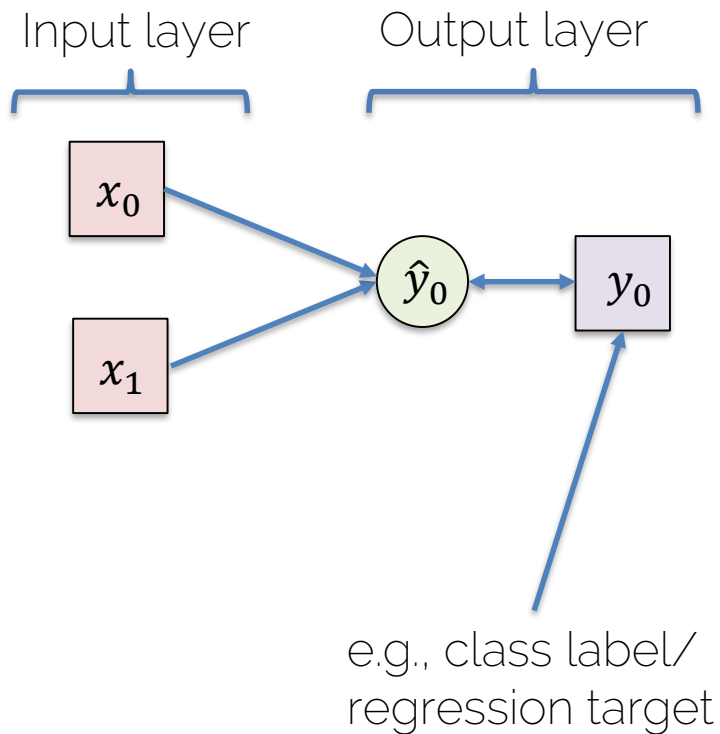
$$\frac{\partial f}{\partial x}$$

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

# Compute Graphs -> Neural Networks

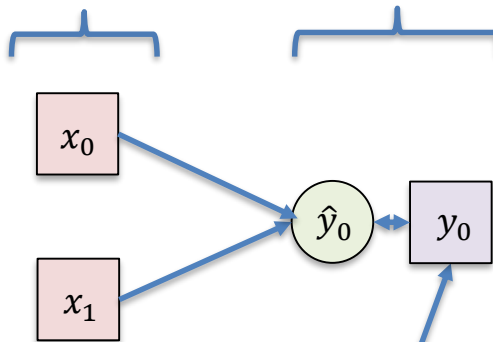
- $\mathbf{x}_k$  input variables
- $\mathbf{w}_{l,m,n}$  network weights (note 3 indices)
  - $l$  which layer
  - $m$  which neuron in layer
  - $n$  which weight in neuron
- $\hat{\mathbf{y}}_i$  computed output ( $i$  output dim;  $\mathbf{n}_{out}$ )
- $\mathbf{y}_i$  ground truth targets
- $L$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

# Compute Graphs -> Neural Networks

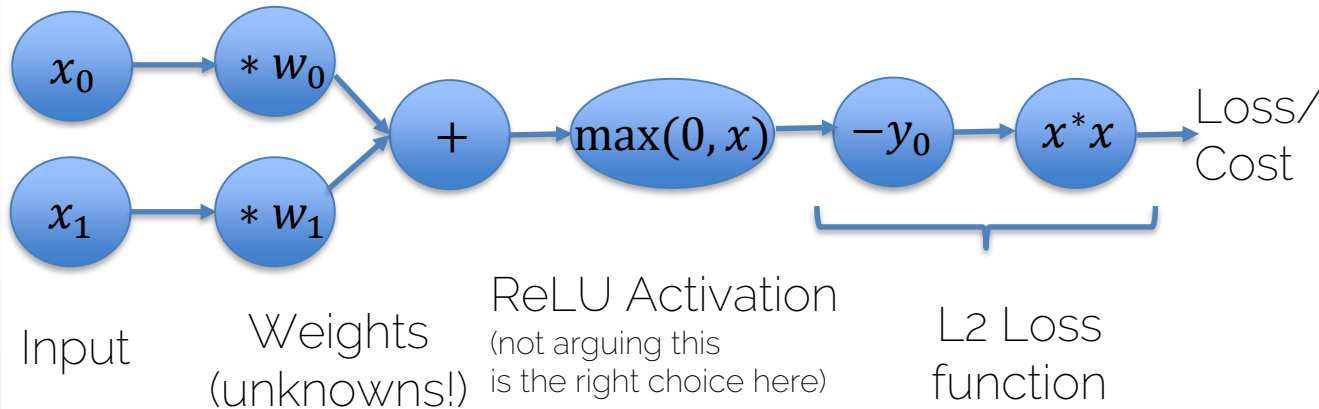


# Compute Graphs -> Neural Networks

Input layer      Output layer

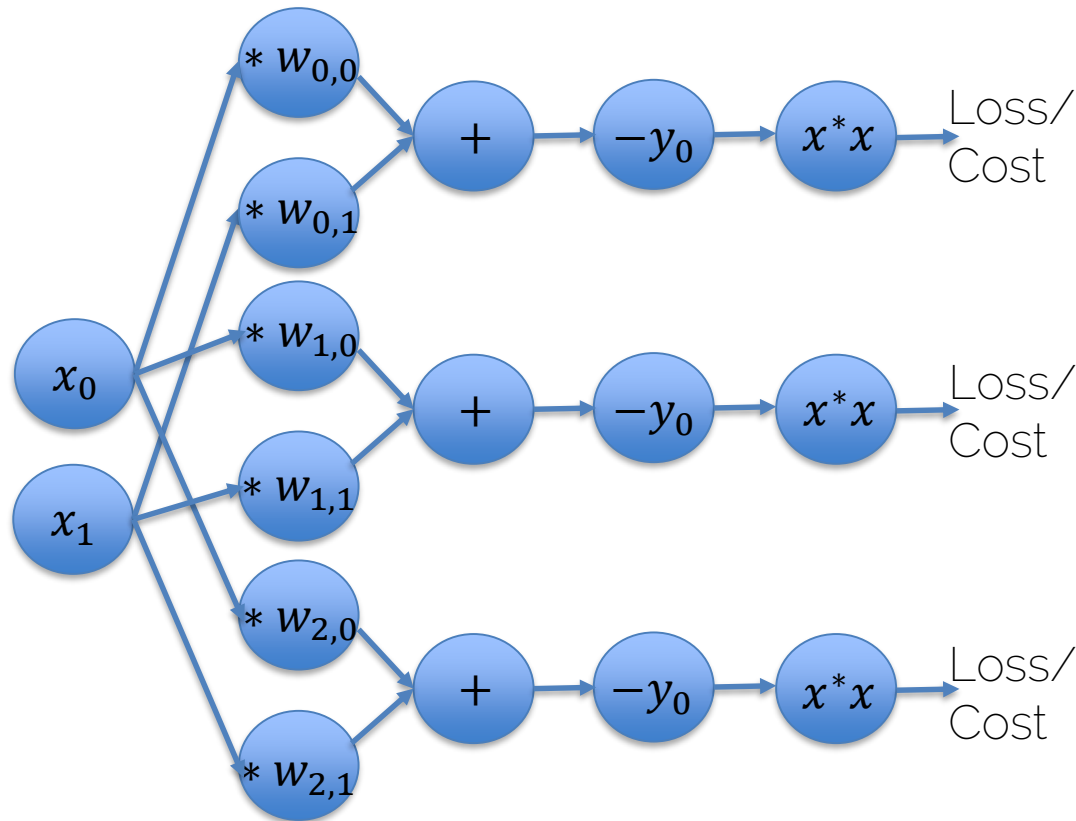
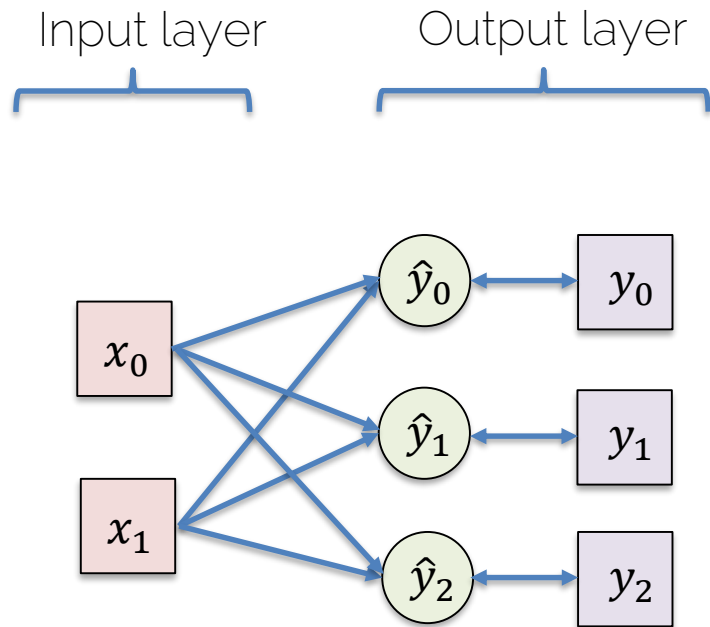


e.g., class label/  
regression target



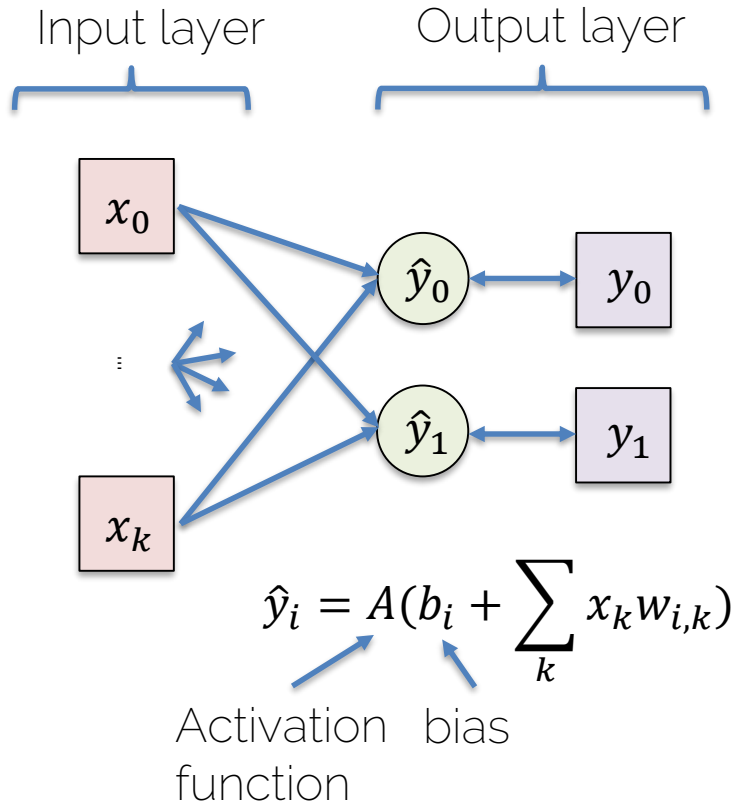
← We want to compute gradients w.r.t. all weights  $\mathbf{W}$

# Compute Graphs $\rightarrow$ Neural Networks



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

# Compute Graphs -> Neural Networks



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

$$L = \sum_i L_i$$

$L$ : sum over loss per sample, e.g.

L2 loss  $\rightarrow$  simply sum up squares:

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

$\rightarrow$  use chain rule to compute partials

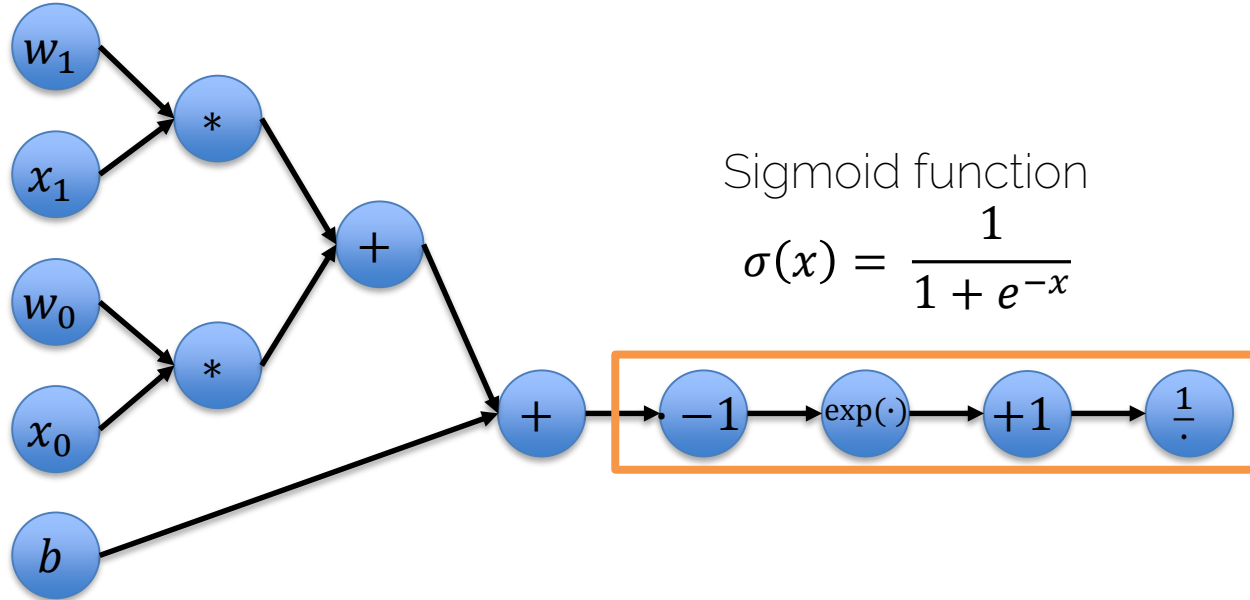
$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{i,k}}$$

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

# NNs as Computational Graphs

- We can express any kind of functions in a

computational graph, e.g.  $f(\mathbf{w}, \mathbf{x}) = \frac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

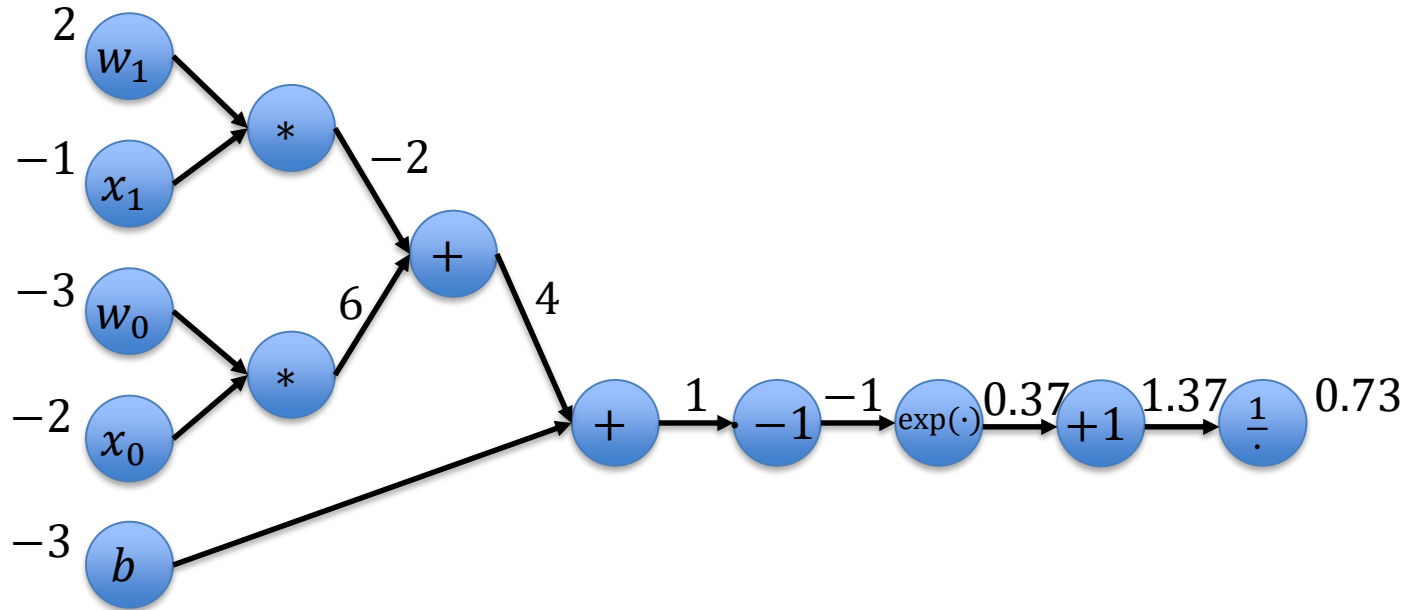


Sigmoid function

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

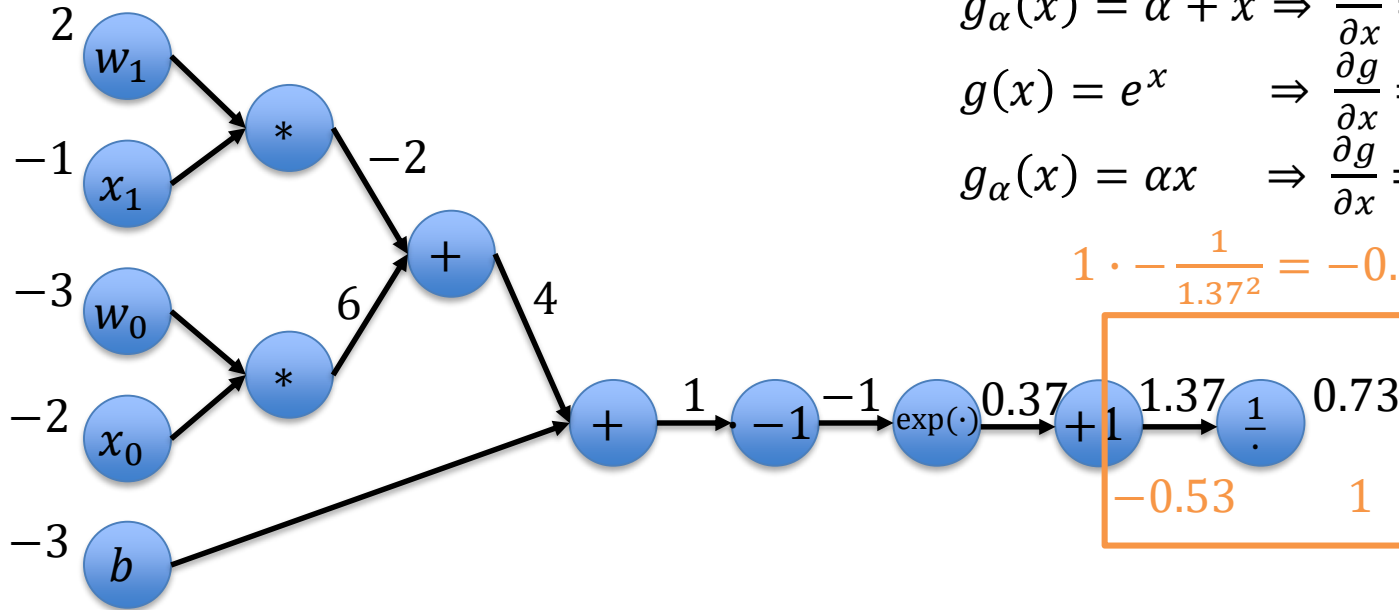
# NNs as Computational Graphs

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



# NNs as Computational Graphs

- $$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}}$$



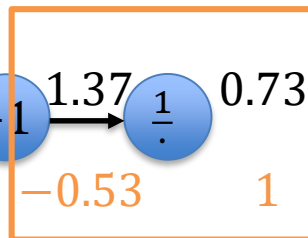
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

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

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

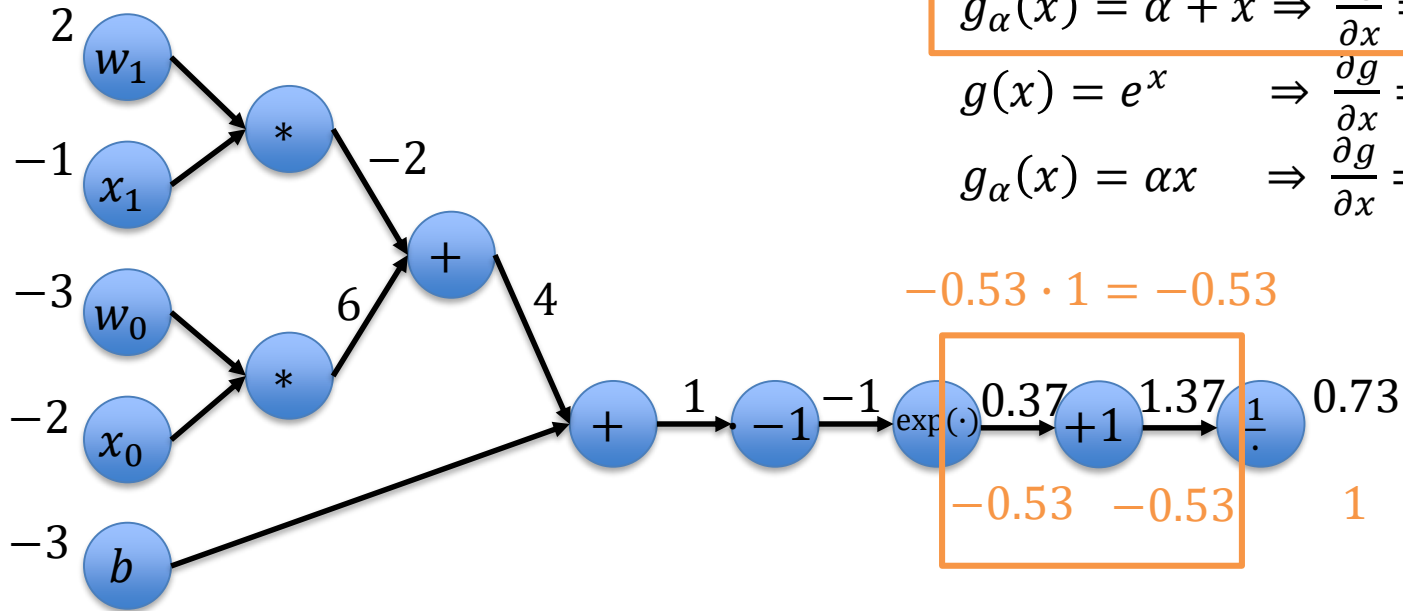
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$

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



# NNs as Computational Graphs

- $$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}}$$



# NNs as Computational Graphs

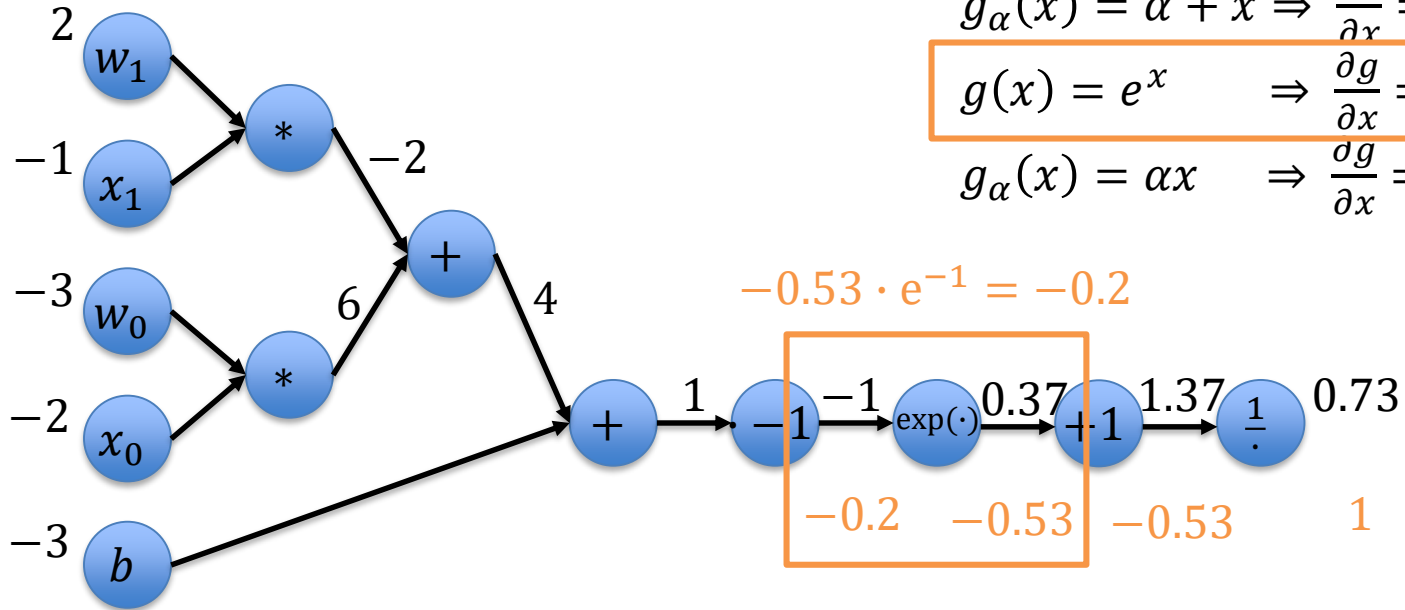
- $$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}}$$

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

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

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

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



# NNs as Computational Graphs

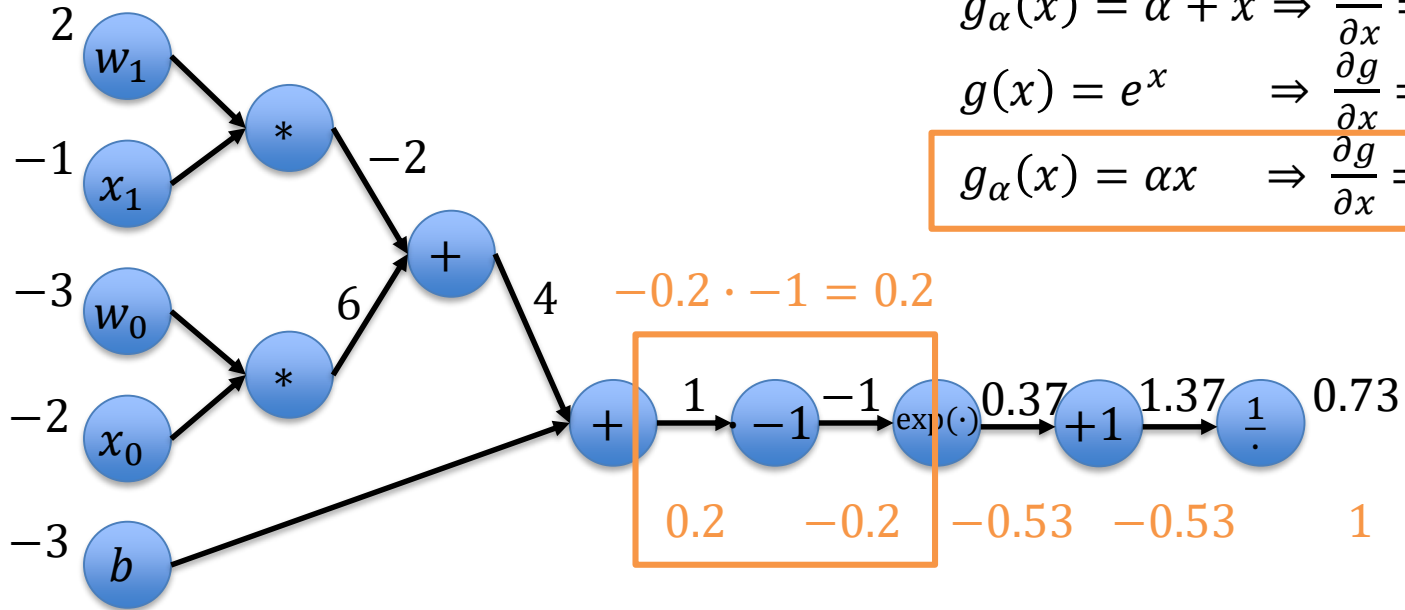
- $$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}}$$

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

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

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

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



# NNs as Computational Graphs

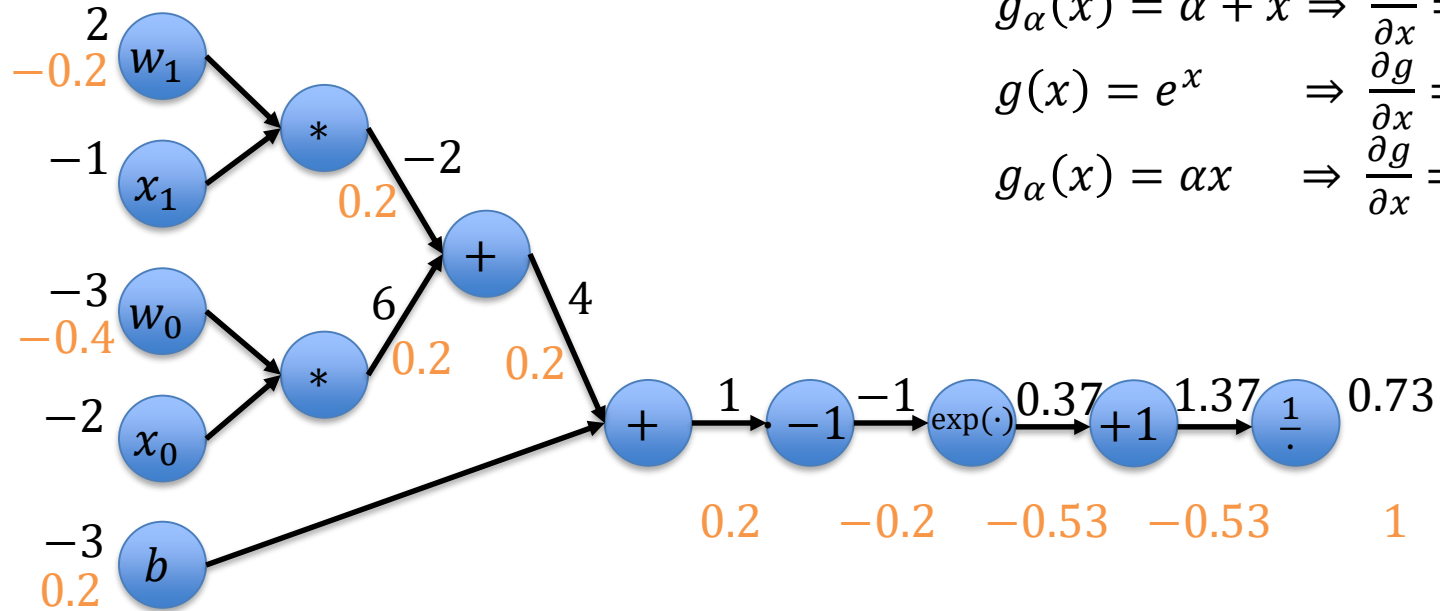
- $$f(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}}$$

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

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

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

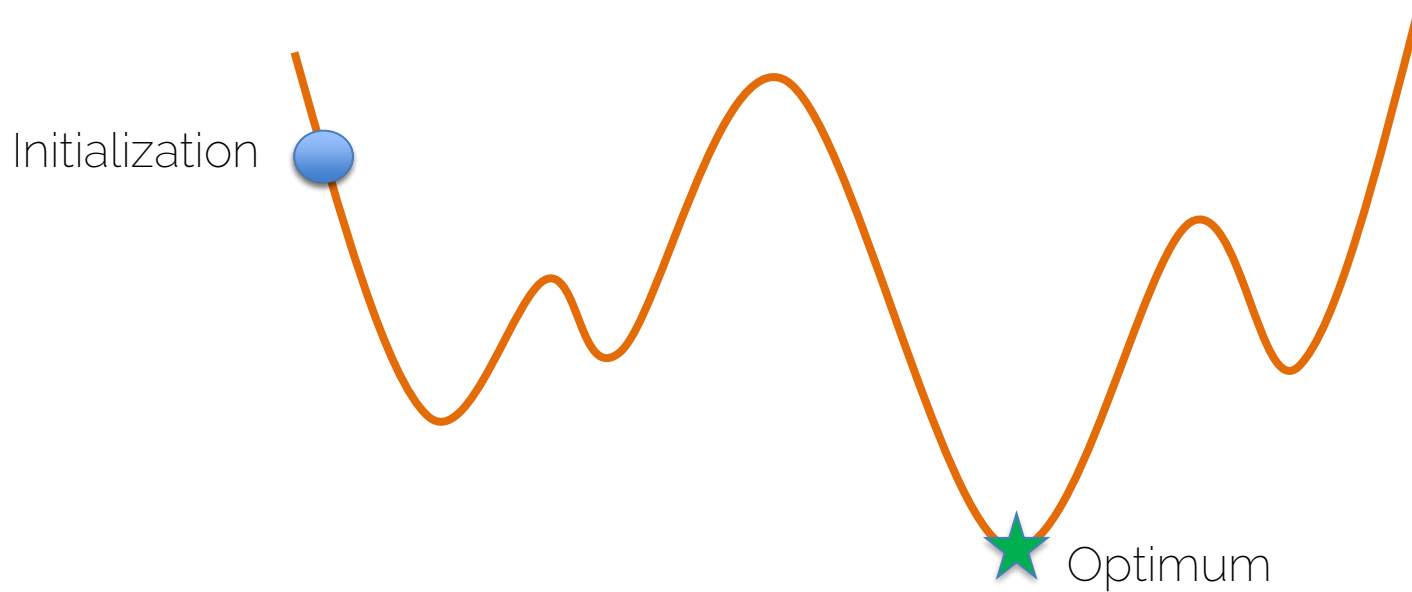
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$



# Gradient Descent

# Gradient Descent

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



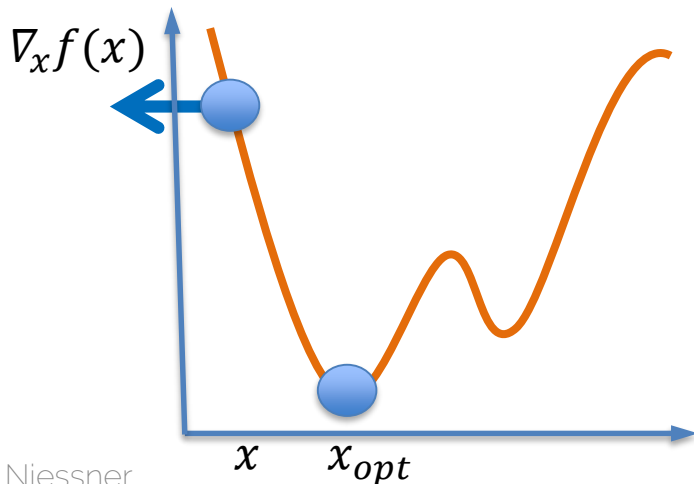
# Gradient Descent

- Gradient = vector of partial derivatives:

$$\nabla_x f(x) = \left( \frac{\partial g}{\partial x_1}, \dots, \frac{\partial g}{\partial x_k} \right)$$

Vector in direction of greatest increase of the function

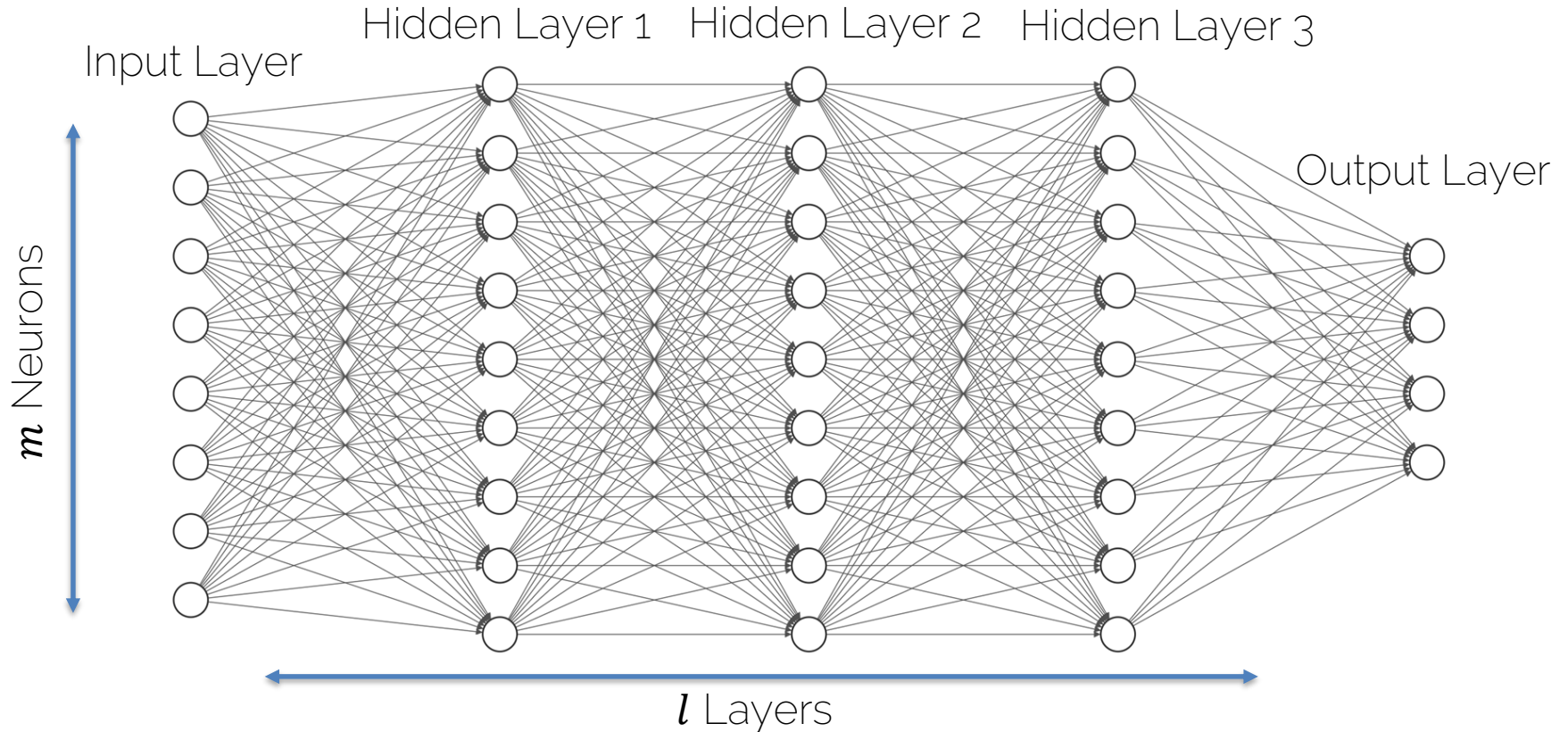
- Take steps in direction of negative gradient:



$$x^{t+1} = x^t - \alpha \nabla_x f(x^t) \rightarrow x_{opt} \text{ for } t \rightarrow \infty$$

Learning rate

# Gradient Descent for Neural Networks



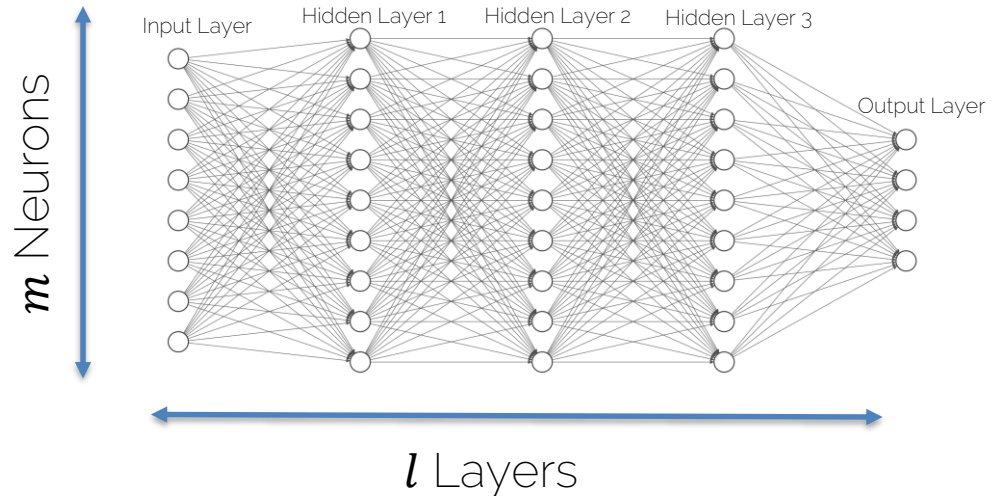
# Gradient Descent for Neural Networks

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

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

Gradient step:

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





# The Flow of the Gradients

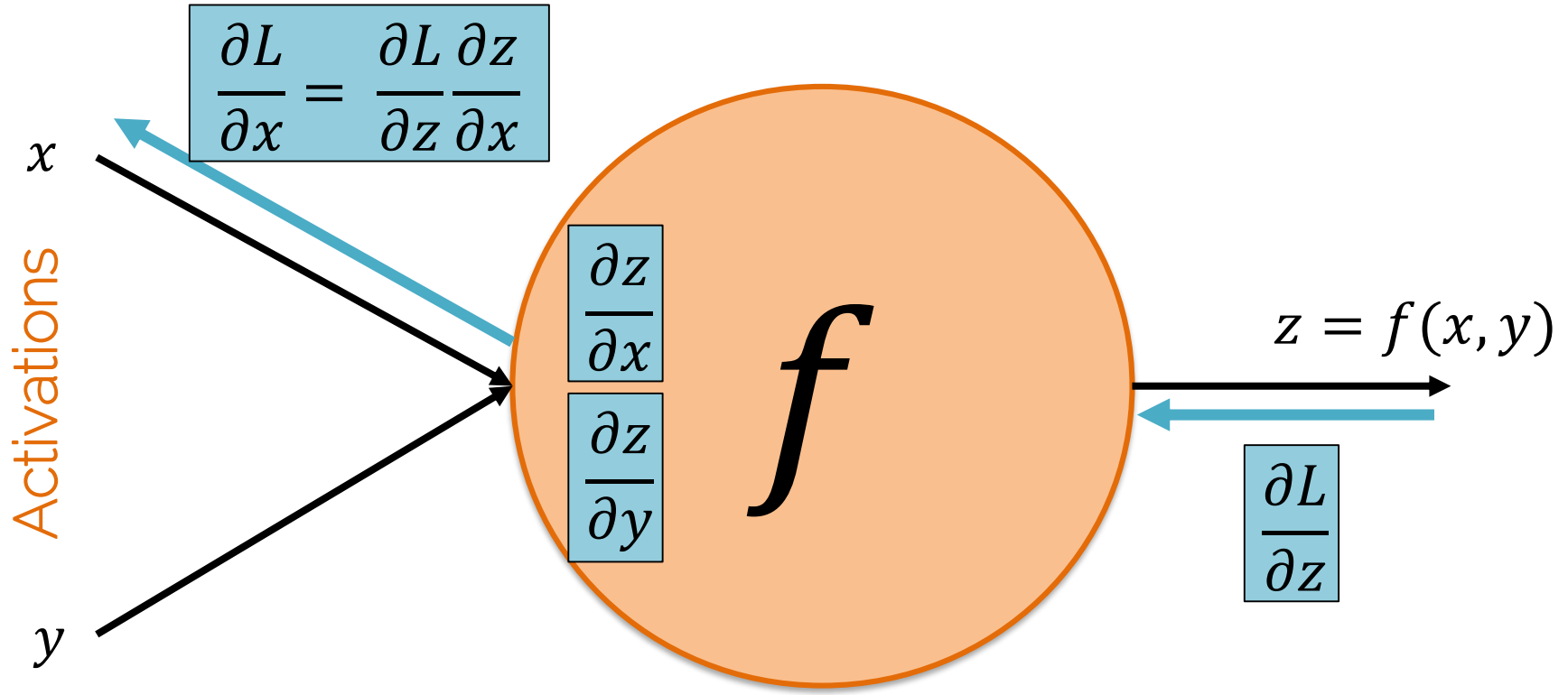
- Many many many many of these nodes form a neural network

## NEURONS

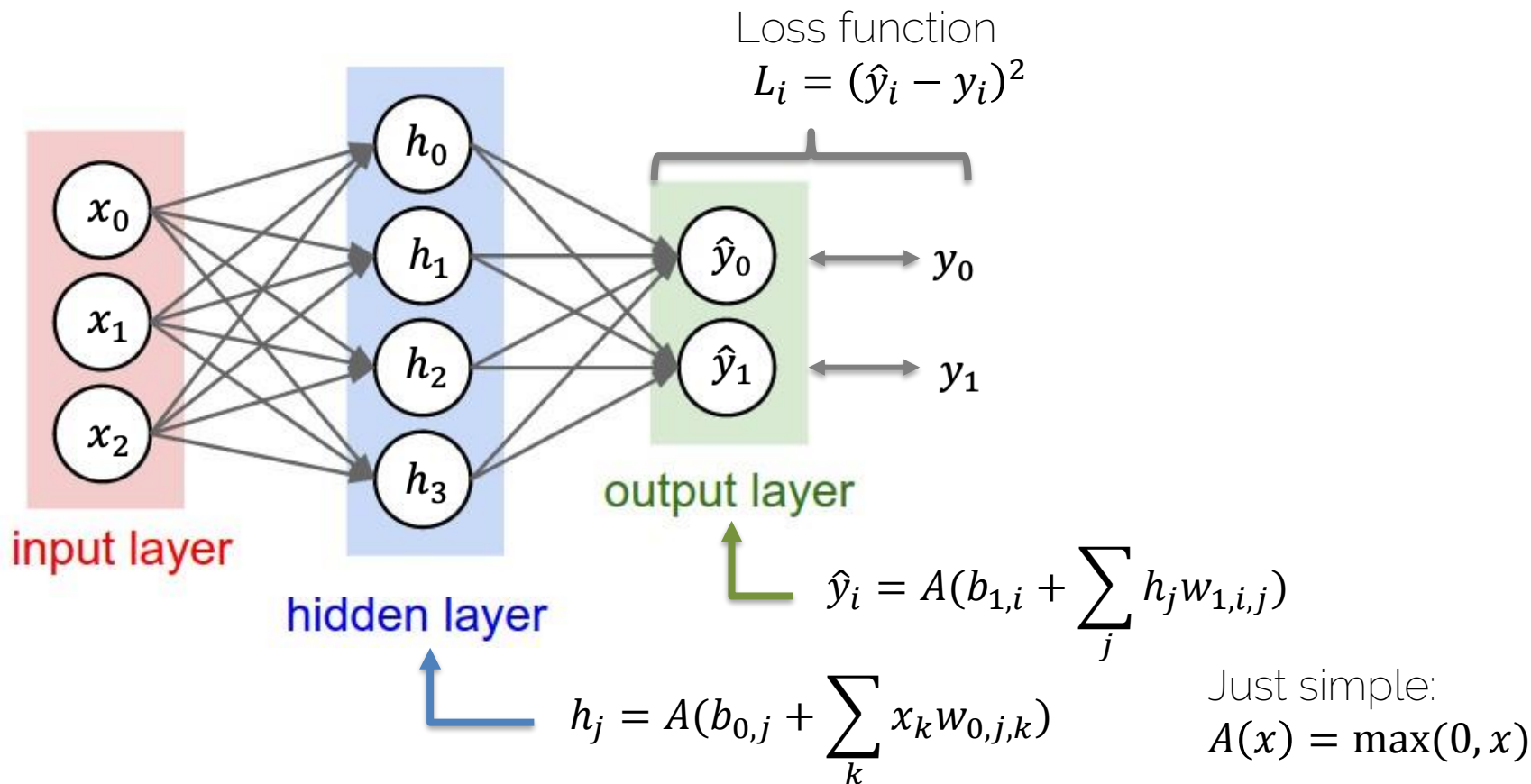
- Each one has its own work to do

## FORWARD AND BACKWARD PASS

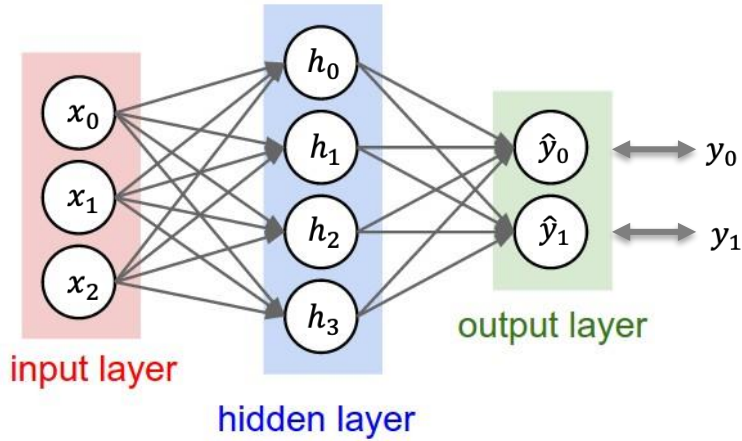
# The Flow of the Gradients



# Gradient Descent for Neural Networks



# Gradient Descent for Neural Networks



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

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

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

## Backpropagation

Just go through layer by layer

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

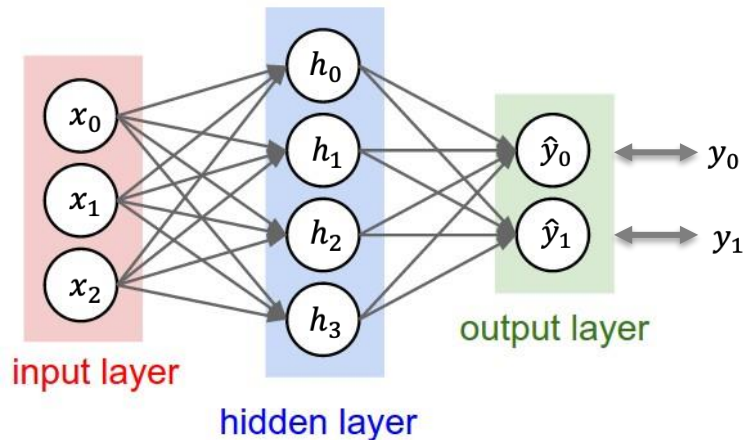
$$\frac{\partial L_i}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)$$

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

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

...

# Gradient Descent for Neural Networks



How many unknown weights?

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

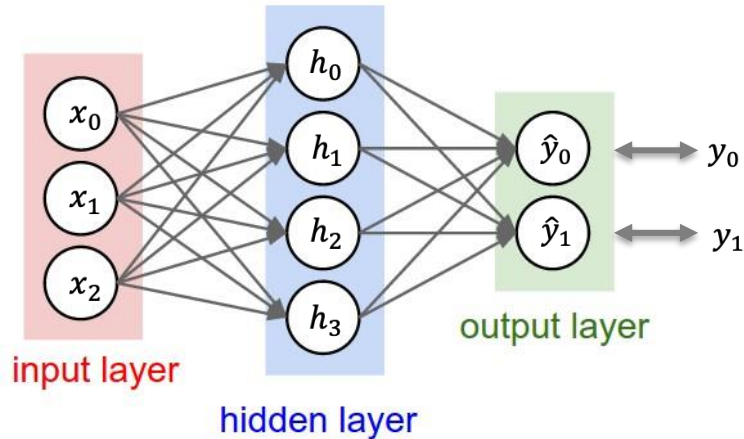
#neurons · #input channels + #biases

$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$
$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

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

Note that some activations have also weights

# Derivatives of Cross Entropy Loss



Gradients of weights of last layer:

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

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

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

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

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

Binary Cross Entropy loss

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

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

output

scores

# Derivatives of Cross Entropy Loss

Gradients of weights of first layer:

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

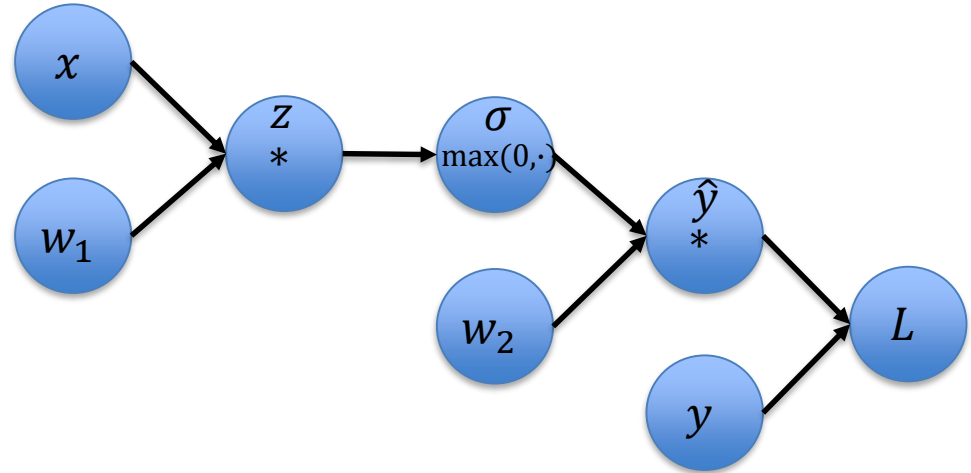
$$\frac{\partial L}{\partial s_j^1} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j^1} = \sum_{i=1}^{n_{out}} (\hat{y}_i - y_i) w_{ji} (h_j (1 - h_j))$$

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

# Back to Compute Graphs & NNs

- Inputs  $\mathbf{x}$  and targets  $\mathbf{y}$
- 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

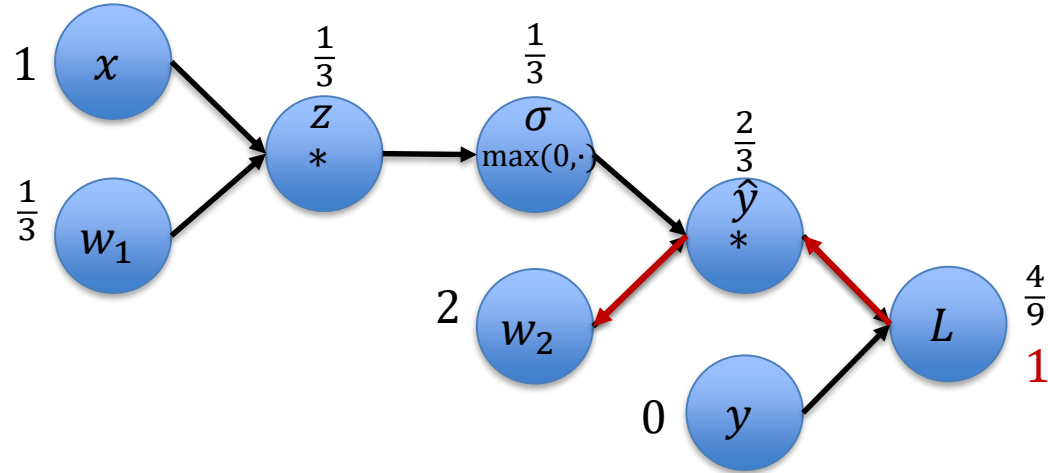
Initialize  $x = 1$ ,  $y = 0$ ,  
 $w_1 = \frac{1}{3}$ ,  $w_2 = 2$

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

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

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

# Gradient Descent for Neural Networks

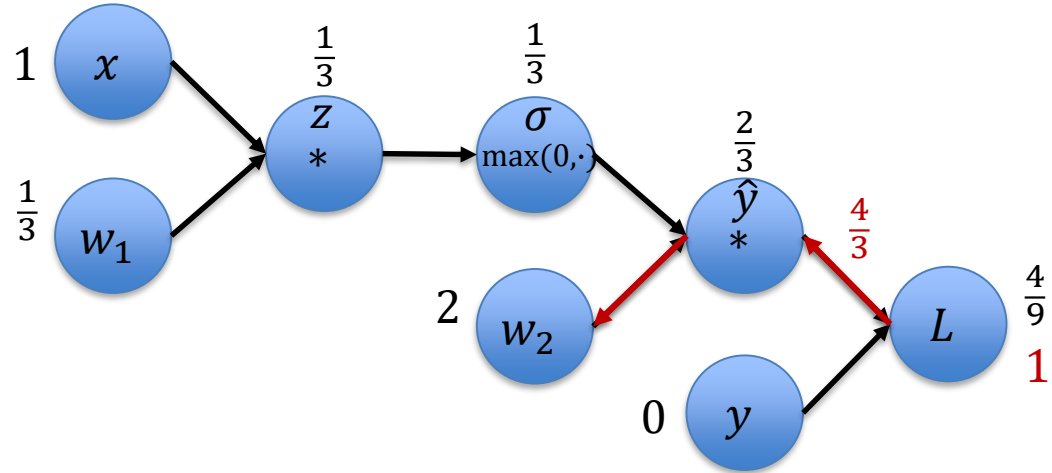
Initialize  $x = 1$ ,  $y = 0$ ,  
 $w_1 = \frac{1}{3}$ ,  $w_2 = 2$

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

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

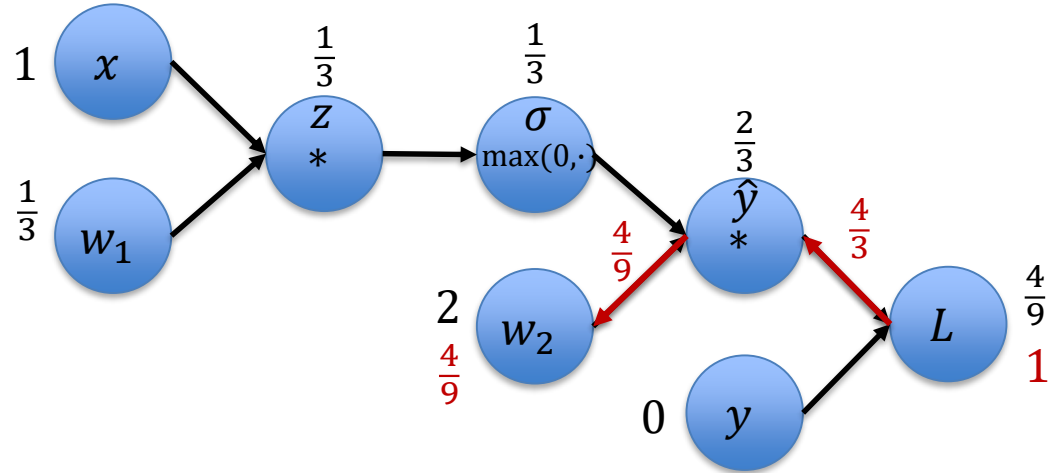
Initialize  $x = 1$ ,  $y = 0$ ,  
 $w_1 = \frac{1}{3}$ ,  $w_2 = 2$

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

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

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

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

# Gradient Descent for Neural Networks

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

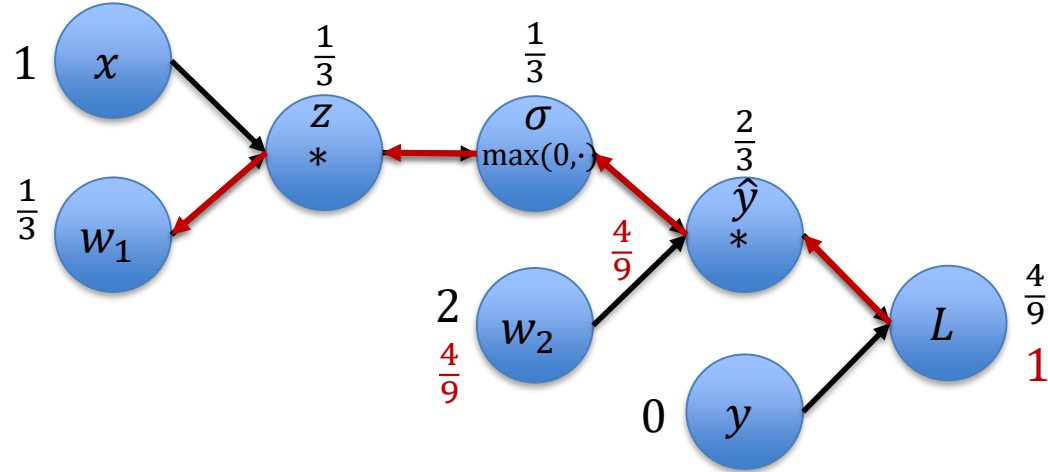
In our case  $n, d = 1$ :

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

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

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

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



Backpropagation

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

# Gradient Descent for Neural Networks

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

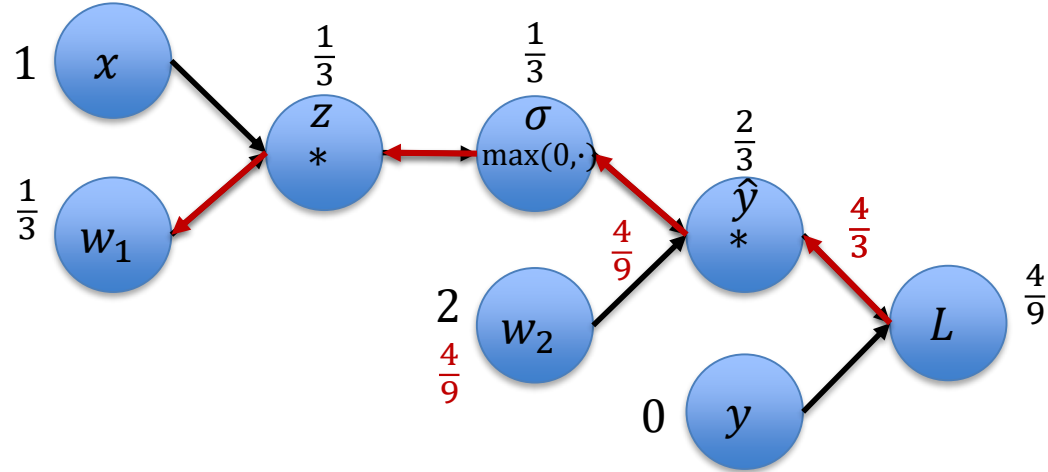
In our case  $n, d = 1$ :

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

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

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

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



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

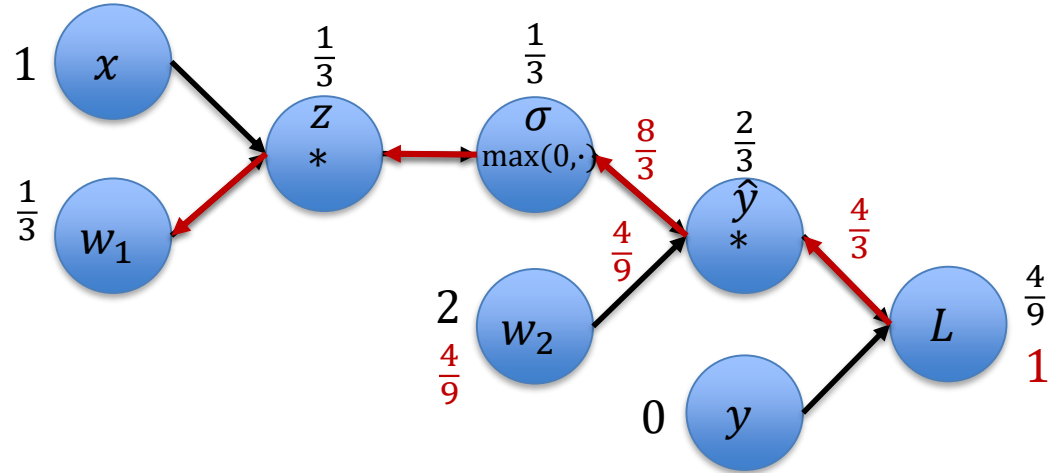
In our case  $n, d = 1$ :

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

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

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

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



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

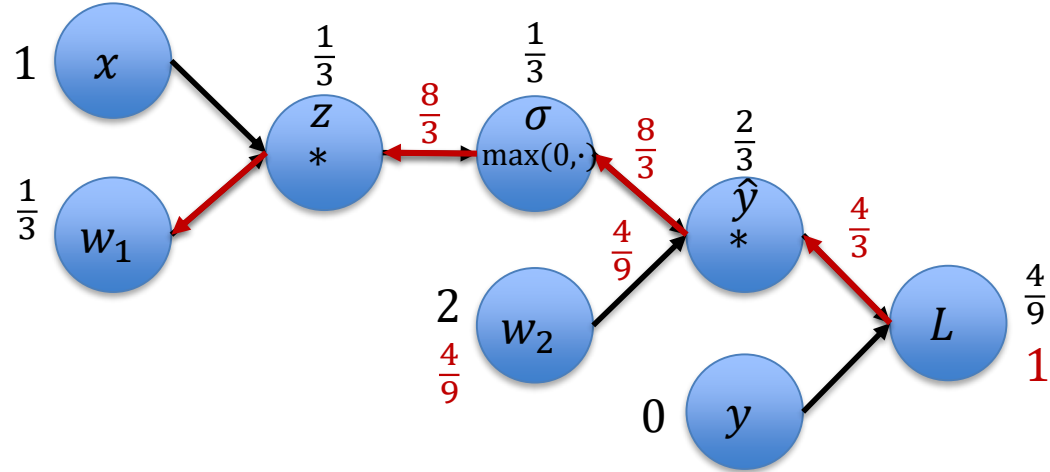
In our case  $n, d = 1$ :

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

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

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

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



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

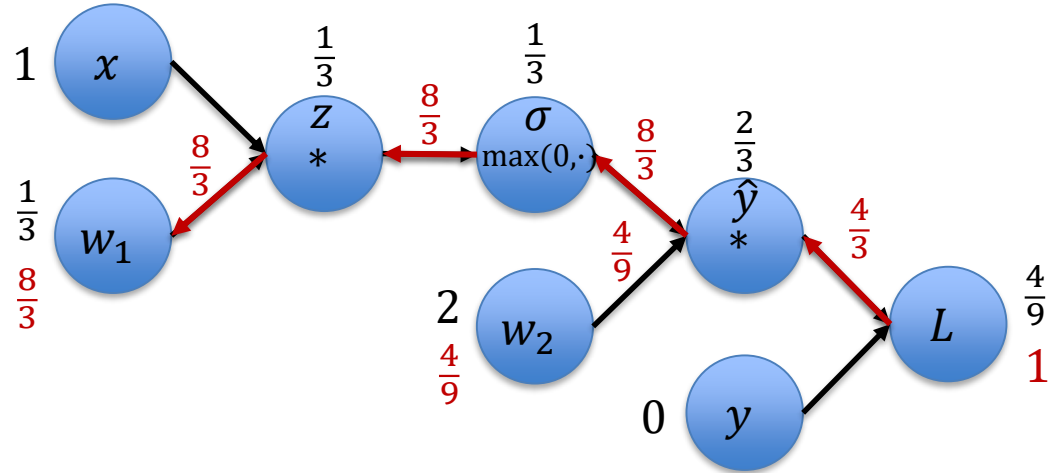
In our case  $n, d = 1$ :

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

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

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

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



Backpropagation

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

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

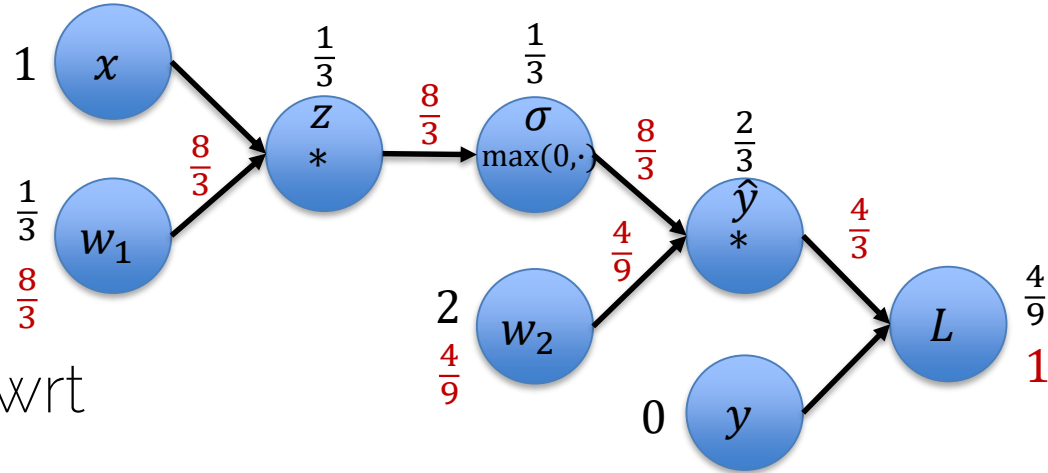
# Gradient Descent for Neural Networks

- Function we want to optimize:

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

- Computed gradients wrt to weights  $\mathbf{w}_1$  and  $\mathbf{w}_2$
- Now: update the weights

$$\begin{aligned} \mathbf{w}' &= \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} f = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \nabla_{w_1} f \\ \nabla_{w_2} f \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{3} \\ 2 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{8}{3} \\ \frac{4}{9} \end{pmatrix} \end{aligned}$$



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

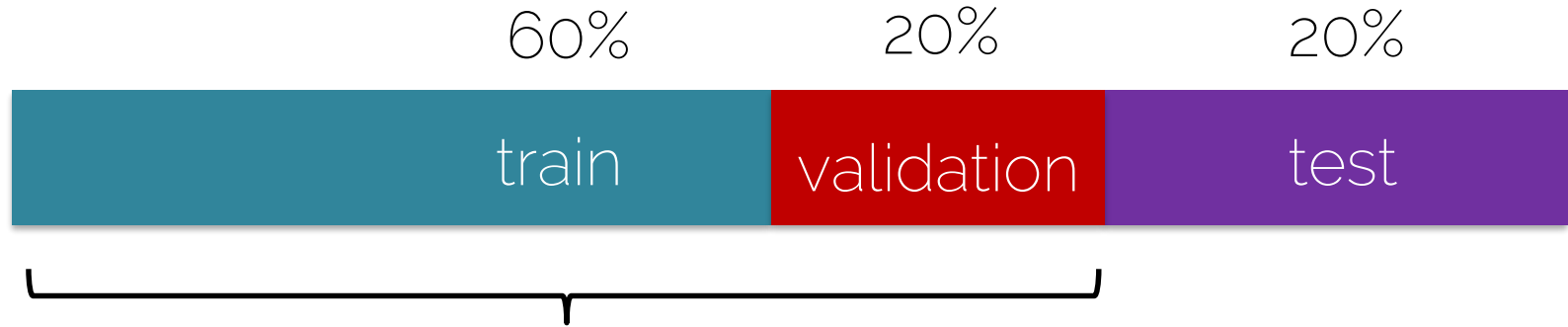
# Gradient Descent

- How to pick good learning rate?
- How to compute gradient for single training pair?
- How to compute gradient for large training set?
- How to speed things up? More to see in next lectures...

# Regularization

# Recap: Basic Recipe for ML

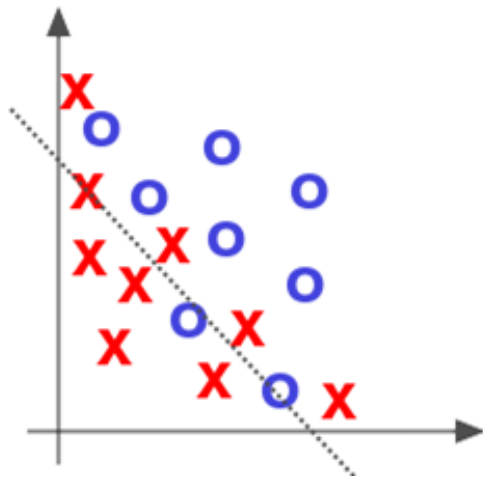
- Split your data



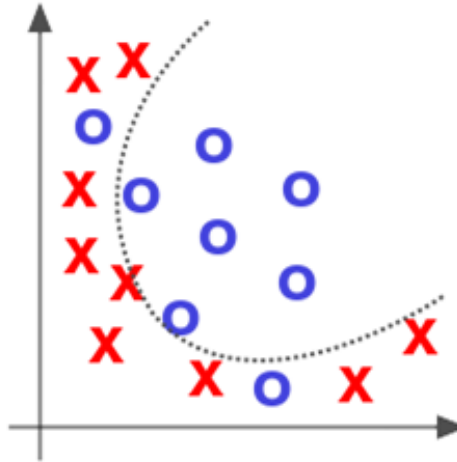
Find your hyperparameters

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

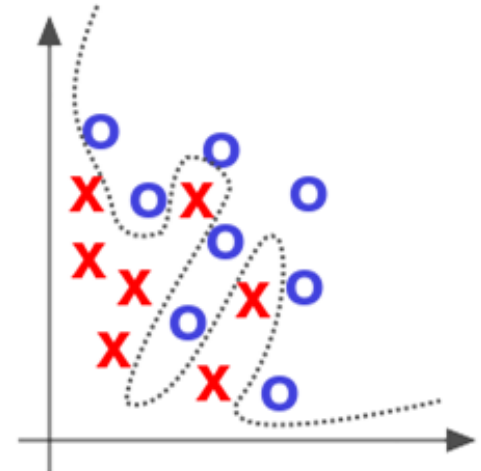
# Over- and Underfitting



Underfitted



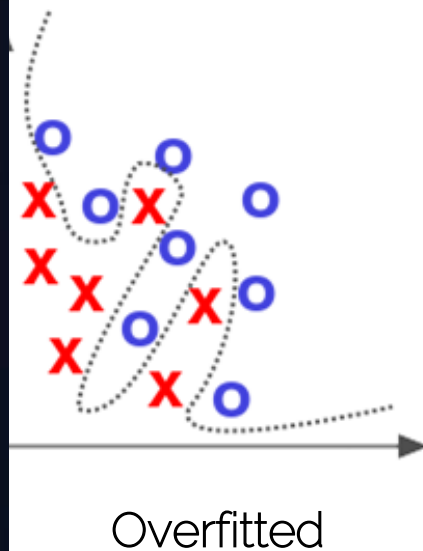
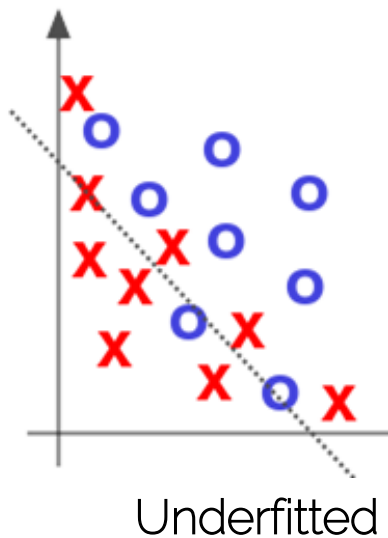
Appropriate



Overfitted

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

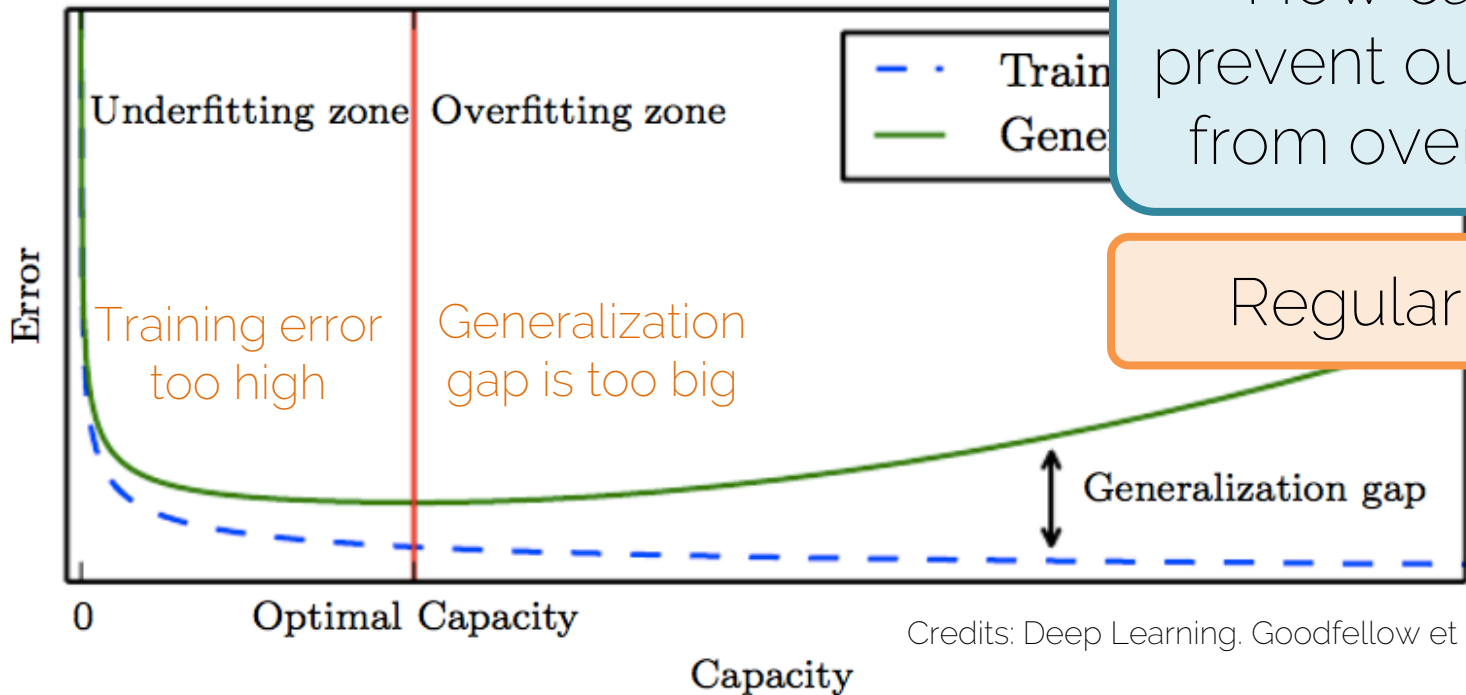
# Over- and Underfitting



© 2017

# Training a Neural Network

- Training/ Validation curve



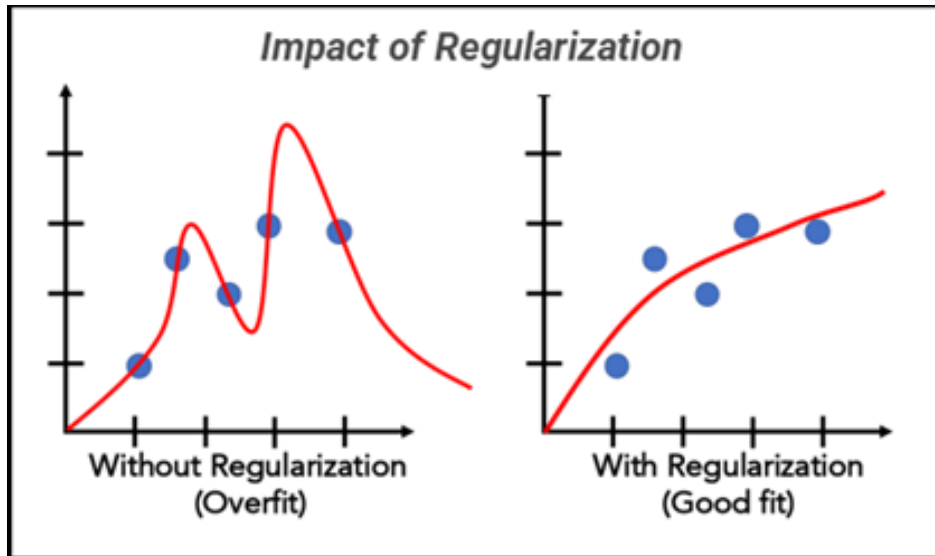
How can we prevent our model from overfitting?

Regularization

Credits: Deep Learning. Goodfellow et al.

# Regularization

- Add constraints or penalties that push network to learn simpler, more robust patterns
  - Try to avoid overfitting to noise

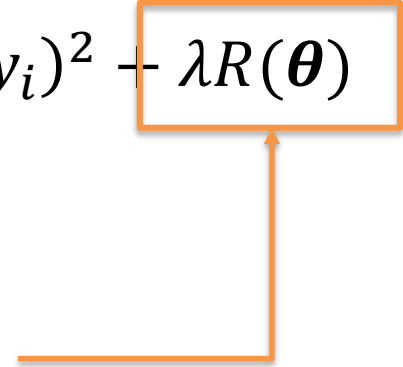


# Regularization

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

- Regularization techniques

- L2 regularization
  - L1 regularization
  - Max norm regularization
  - Dropout
  - Early stopping
  - ...
- } Add regularization term to loss function



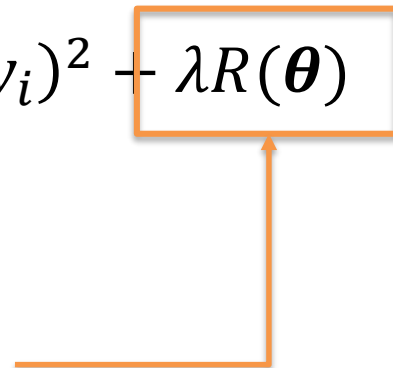
# Regularization

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



- Regularization techniques

- L2 regularization
  - L1 regularization
- } Add regularization term to loss function

- Max norm regularization
  - Dropout
  - Early stopping
  - ...
- } More details later



# Regularization: Example

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

# Regularization: Example

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

- L2 regularization  $R(\boldsymbol{\theta}) = \sum_{i=1}^n \theta_i^2$

$$R(\boldsymbol{\theta}_1) = 0 + 0.75^2 + 0 = 0.5625$$

$$R(\boldsymbol{\theta}_2) = 0.25^2 + 0.5^2 + 0.25^2 = 0.375 \quad \text{Minimization}$$

$$\mathbf{x} = [1, 2, 1], \boldsymbol{\theta}_1 = [0, 0.75, 0], \boldsymbol{\theta}_2 = [0.25, 0.5, 0.25]$$

# Regularization: Example

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



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

$$R(\boldsymbol{\theta}_1) = 0 + 0.75 + 0 = 0.75 \quad \text{Minimization}$$



$$R(\boldsymbol{\theta}_2) = 0.25 + 0.5 + 0.25 = 1$$

$$\mathbf{x} = [1, 2, 1], \boldsymbol{\theta}_1 = [0, 0.75, 0], \boldsymbol{\theta}_2 = [0.25, 0.5, 0.25]$$

# Regularization: Example

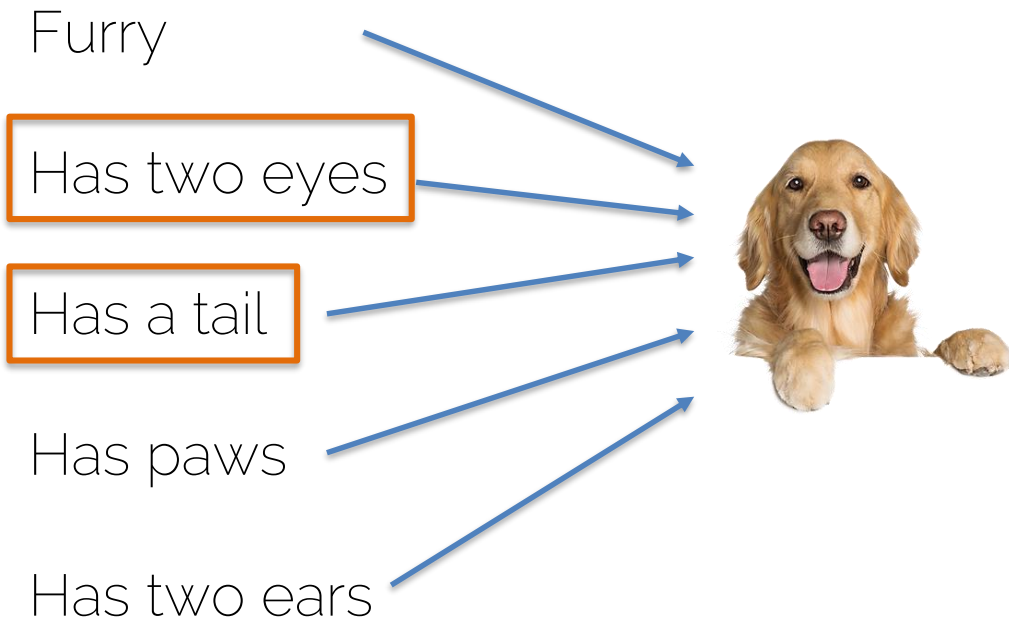
- Input: 3 features  $\mathbf{x} = [1, 2, 1]$
- Two linear classifiers that give the same result:
- $\theta_1 = [0, 0.75, 0]$   Ignores 2 features
- $\theta_2 = [0.25, 0.5, 0.25]$   Takes information from all features

# Regularization: Example

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

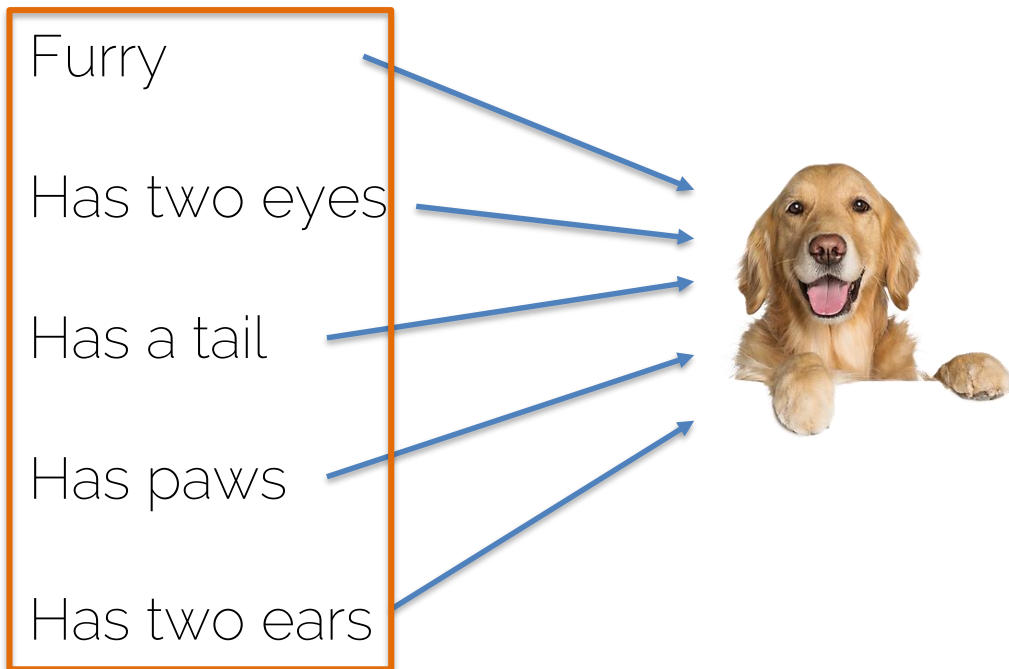
- Dog classifier takes different inputs



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

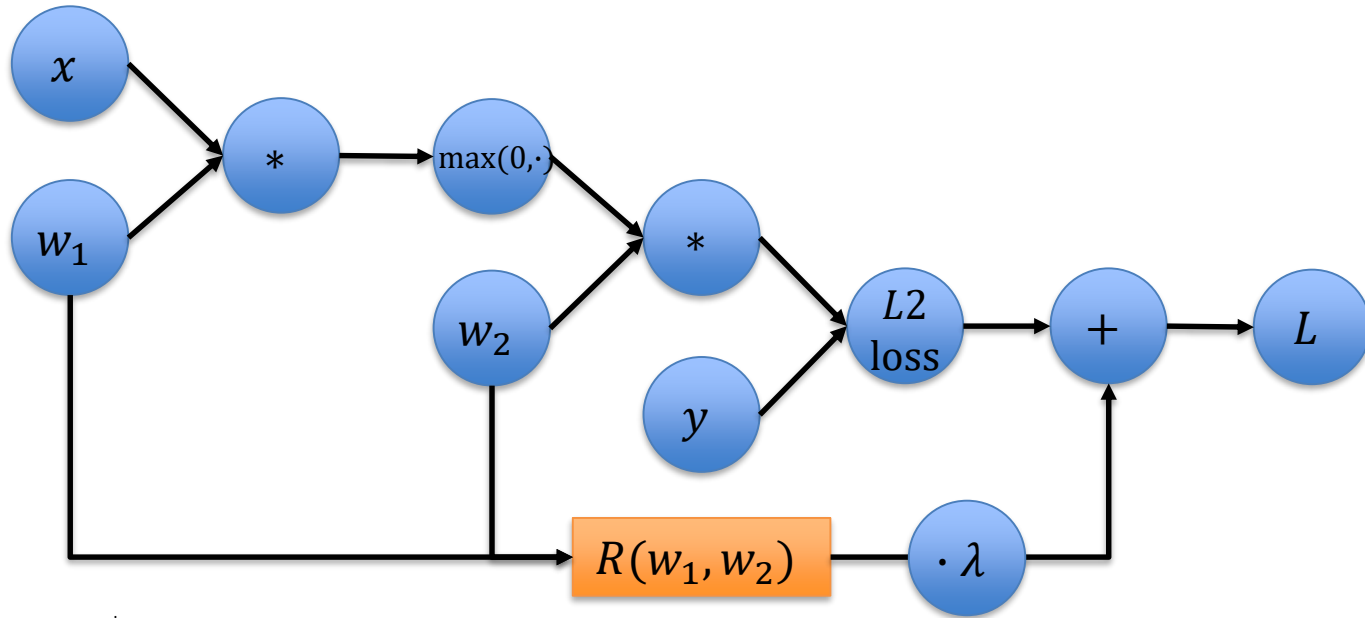
# Regularization: Effect

- Dog classifier takes different inputs



L2 regularization will take all information into account to make decisions

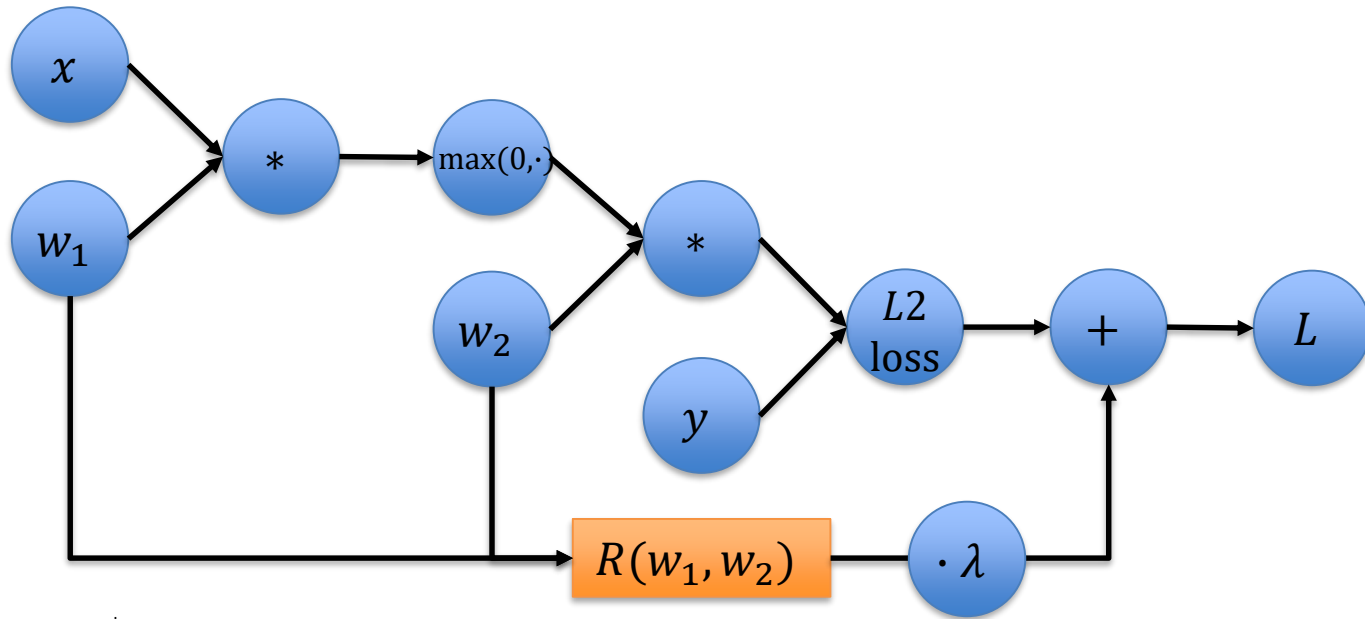
# Regularization for Neural Networks



Combining nodes:  
Network output + L2-loss +  
regularization

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

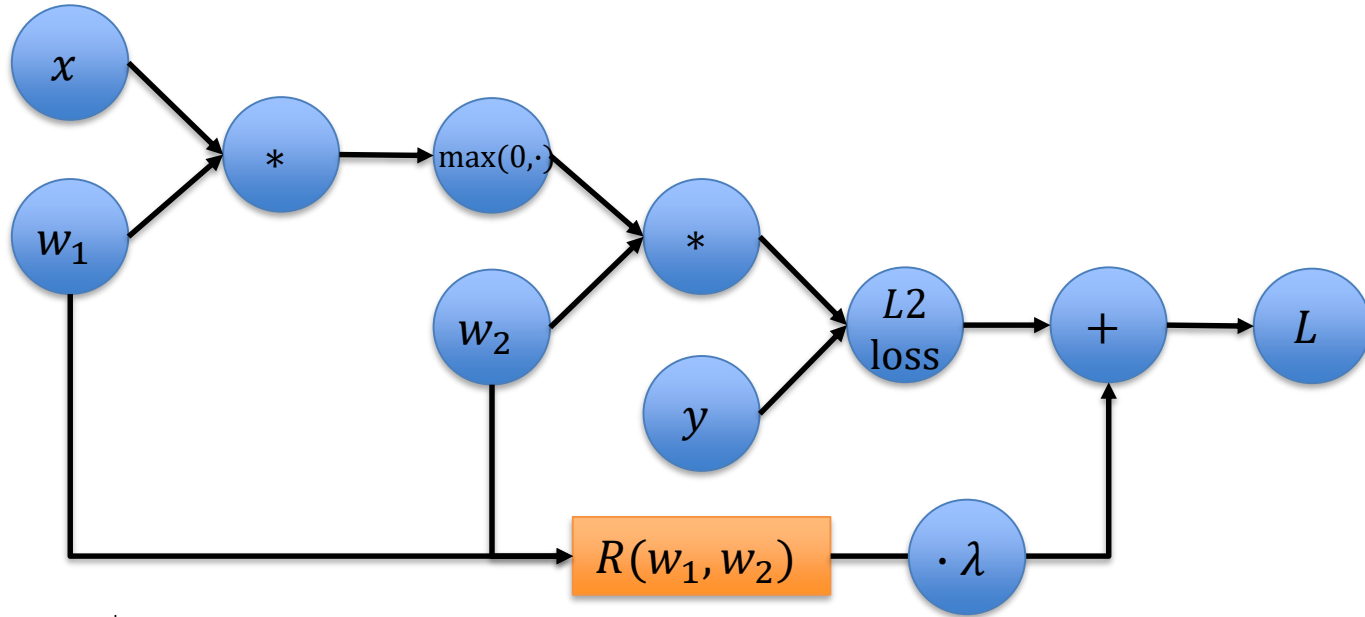
# Regularization for Neural Networks



Combining nodes:  
Network output + L2-loss +  
regularization

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

# Regularization for Neural Networks



Combining nodes:  
Network output + L2-loss +  
regularization

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

# Regularization

Regularization

$\lambda = 0$

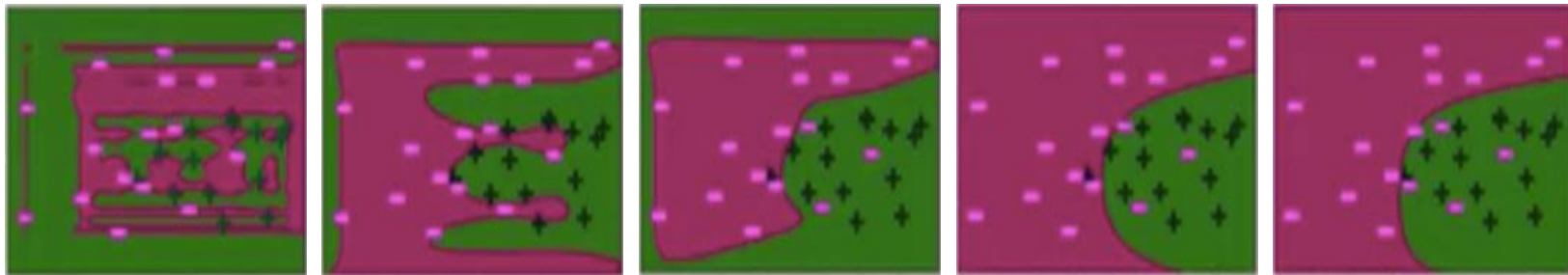
$\lambda = 0.00001$

$\lambda = 0.001$

$\lambda = 1$

$\lambda = 10$

Decision  
Boundary



Credit: University of Washington

What happens to the training error?

What is the goal of regularization?

# Regularization

- Any strategy that aims to



Lower  
validation error



Increasing  
training error

# Next Lecture

- This week:
  - Check exercises!
  - Check piazza / post questions 😊
- Next lecture
  - Optimization of Neural Networks
  - In particular, introduction to SGD (our main method!)

# 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

See you next week 😊