

Optimization and Backpropagation

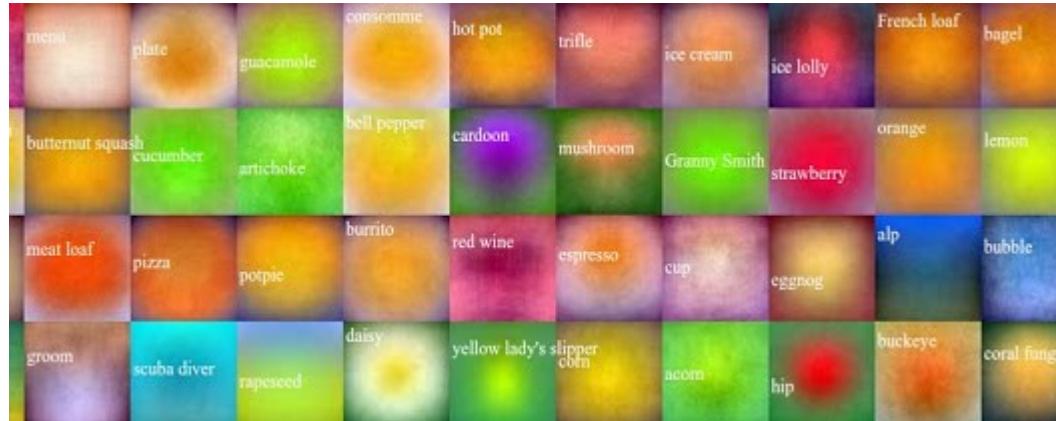
Lecture 3 Recap

Neural Network

- Linear score function $f = Wx$



On CIFAR-10

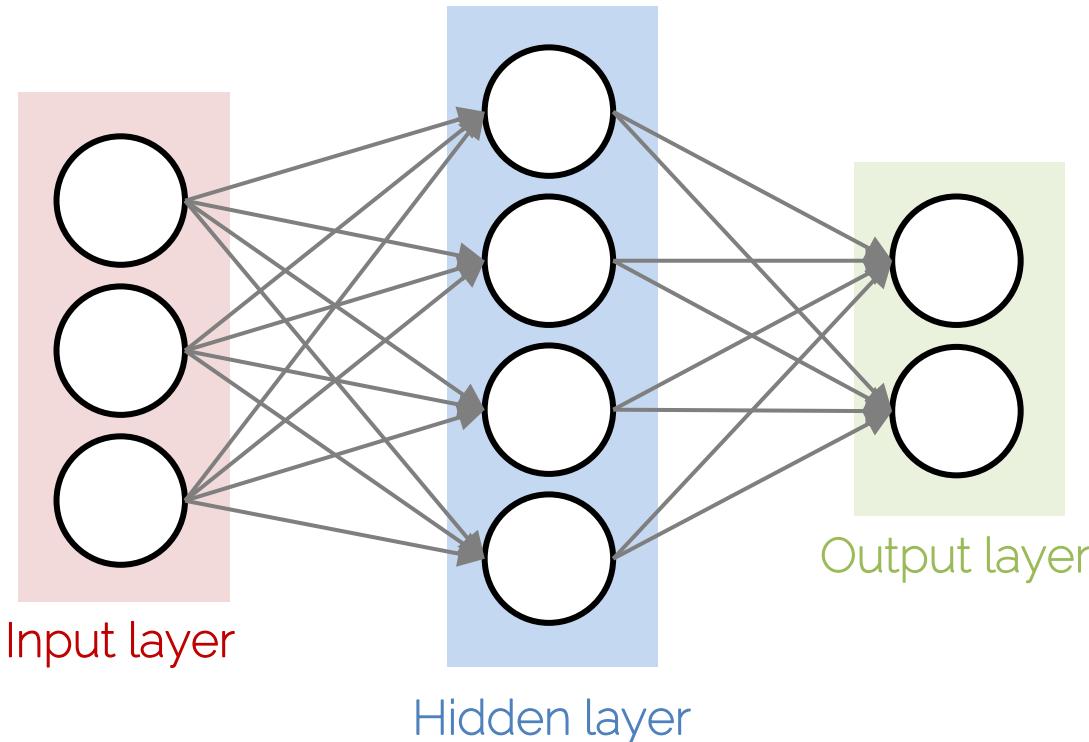


On ImageNet

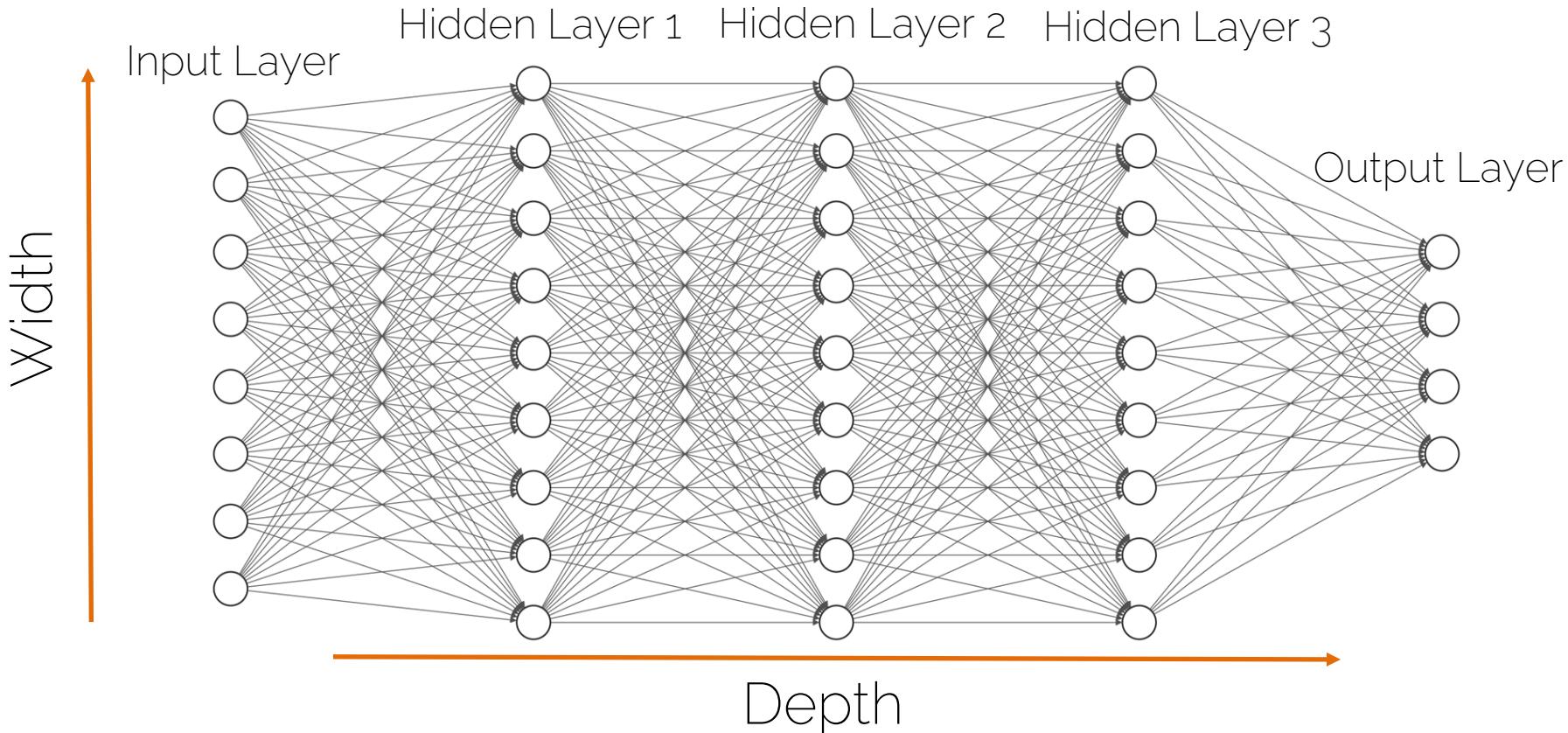
Neural Network

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

Neural Network

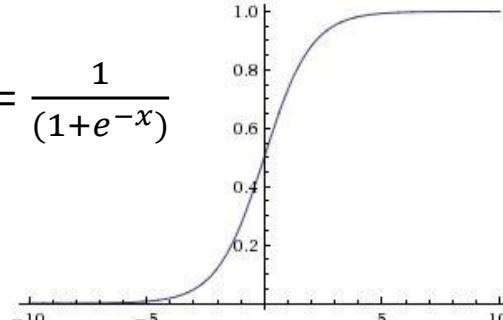


Neural Network

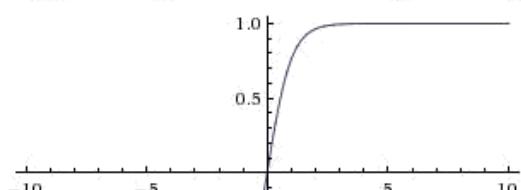


Activation Functions

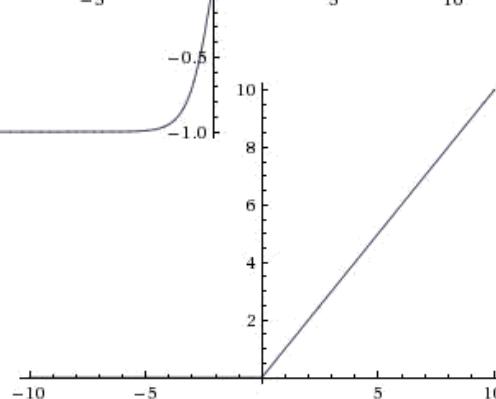
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



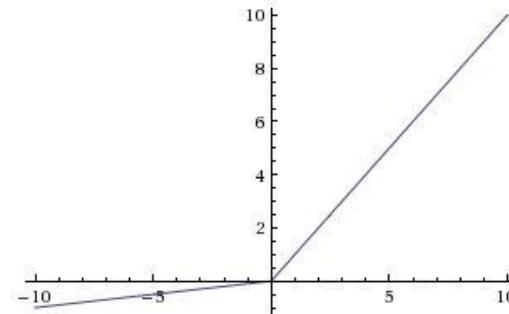
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



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



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

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

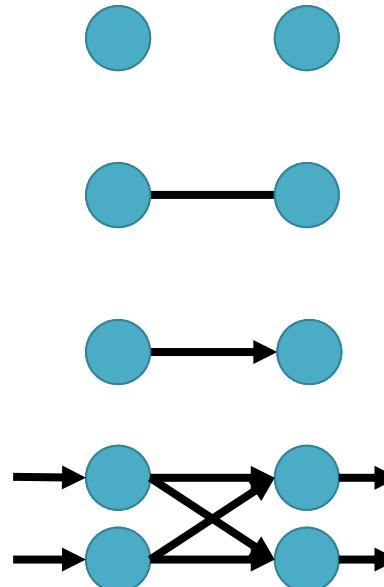
$$\text{ELU } f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

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 $E(y, \hat{y}; \theta) = - \sum_{i=1}^n \sum_{k=1}^K (y_{ik} \cdot \log \hat{y}_{ik})$

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

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

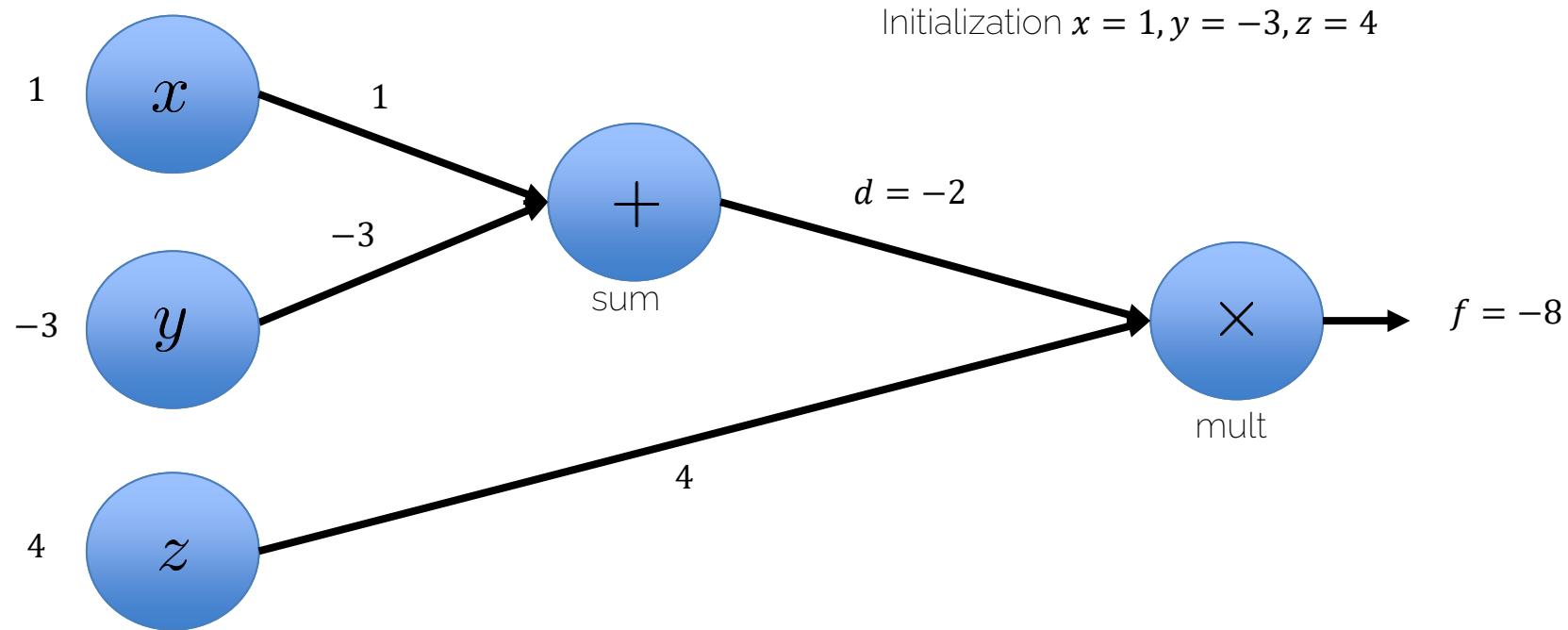
- One can compute gradients analytically but what if our function is too complex?
- Break down gradient computation

Backpropagation

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

Backprop: Forward Pass

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



Backprop: Backward Pass

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

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

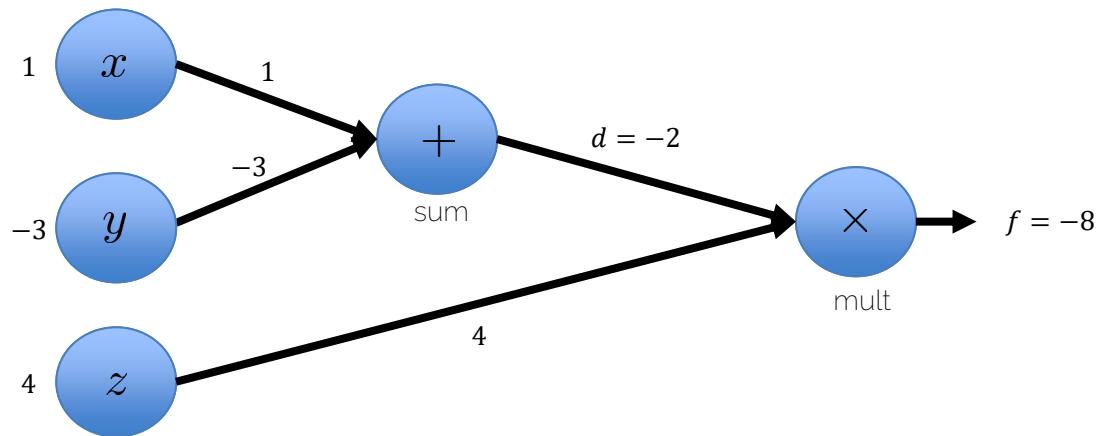
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

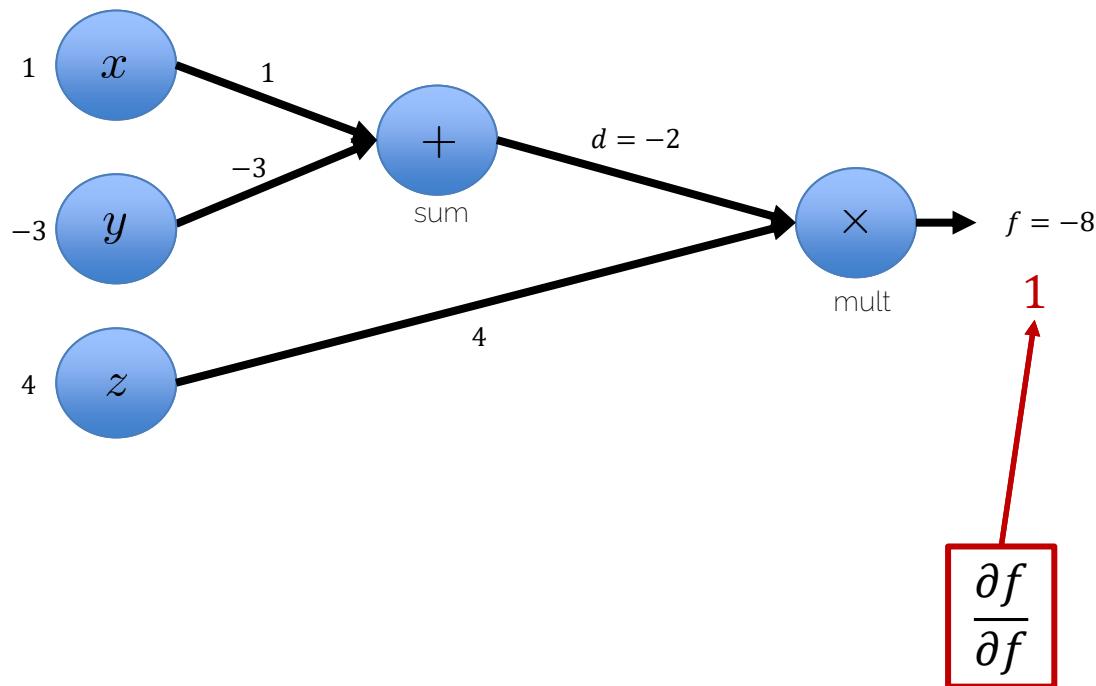
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

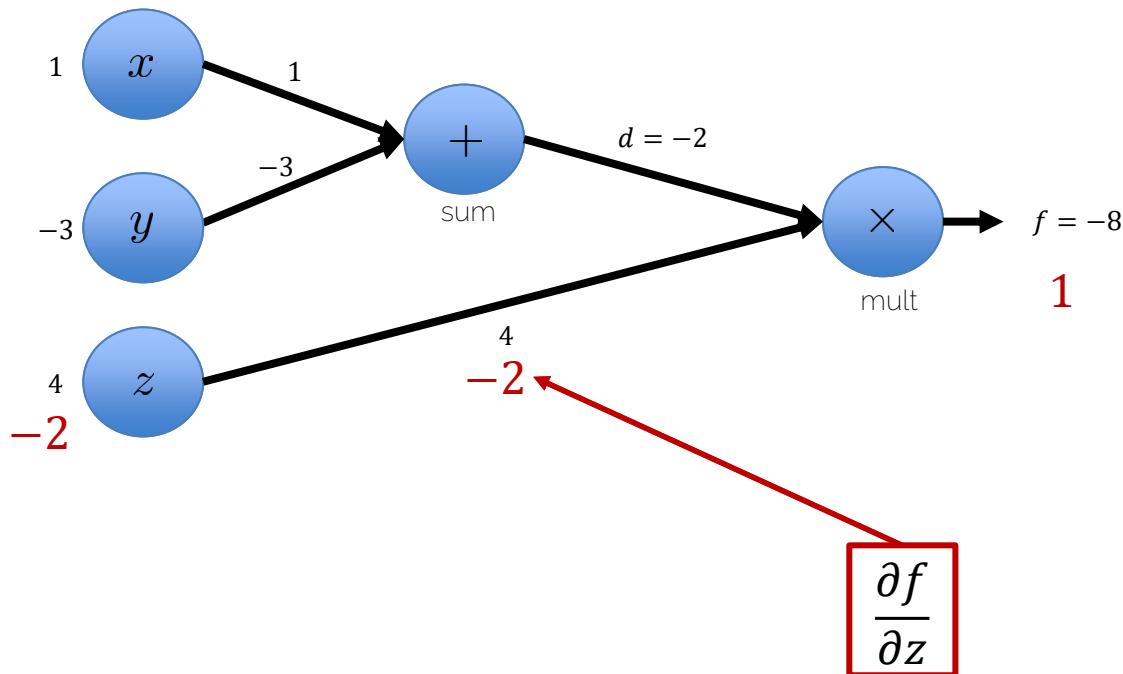
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

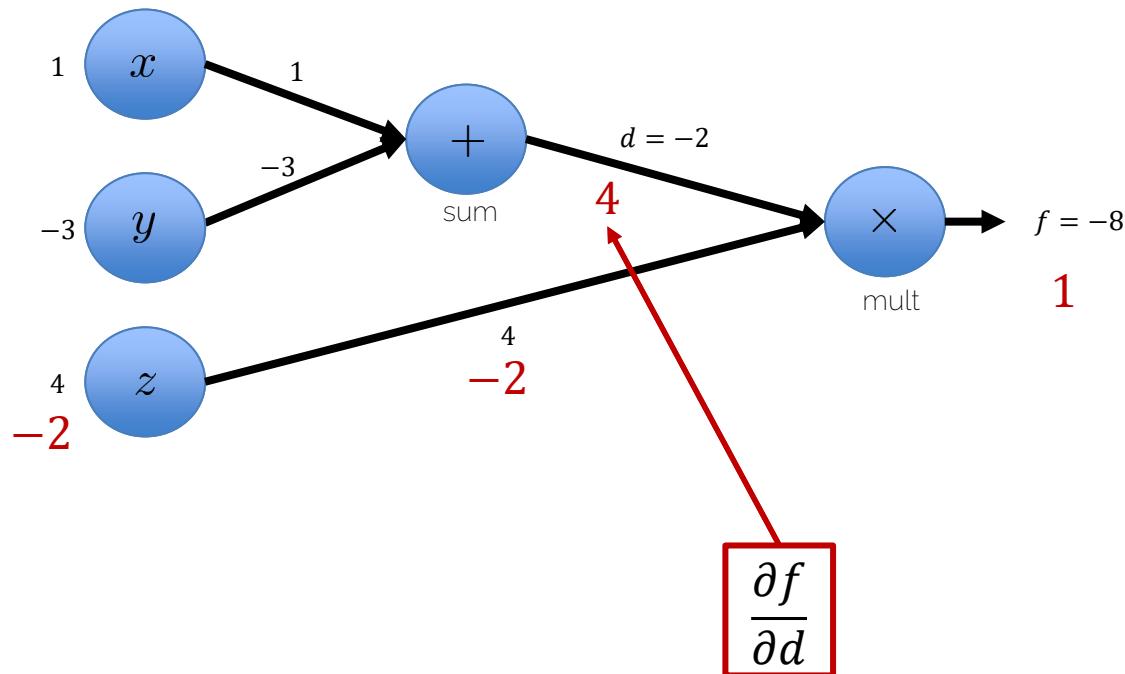
$$d = x + y$$

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

$$f = d \cdot z$$

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

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

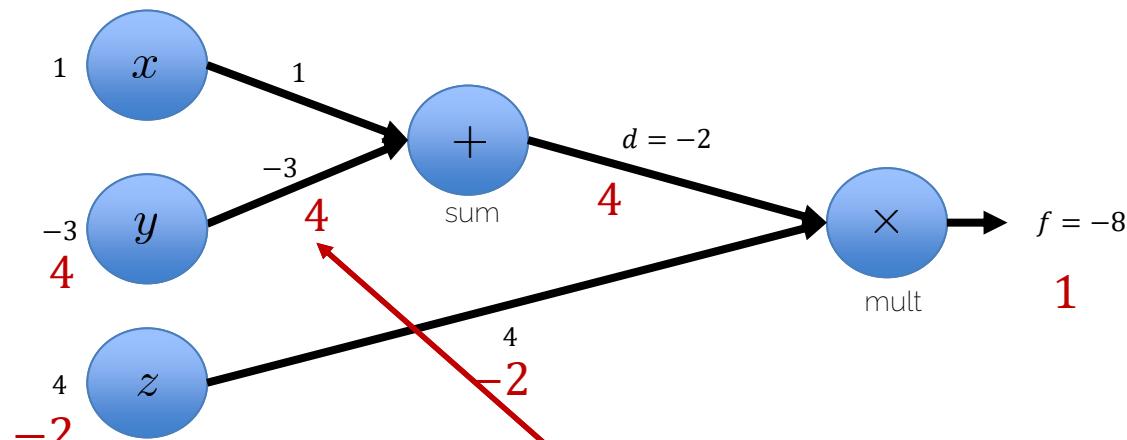
$$d = x + y$$

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

$$f = d \cdot z$$

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

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



Chain Rule:

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

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

Backprop: Backward Pass

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

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

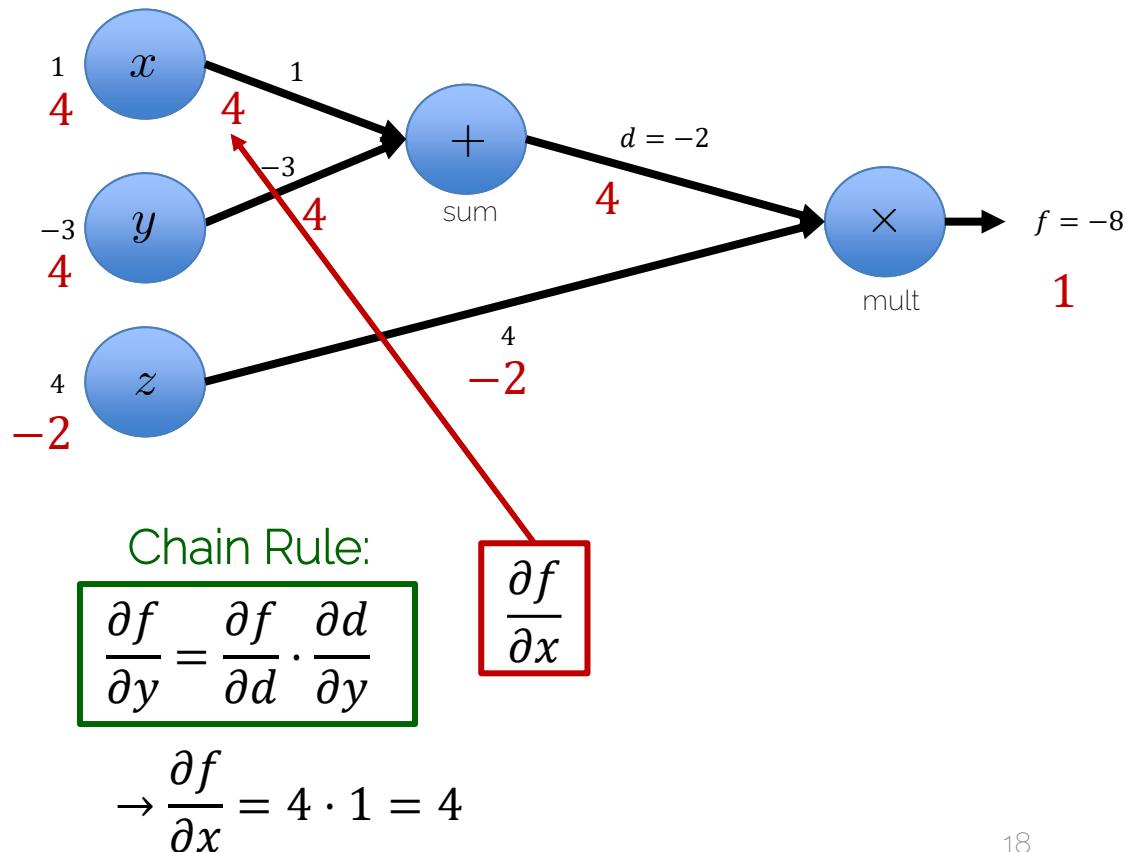
$$d = x + y$$

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

$$f = d \cdot z$$

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

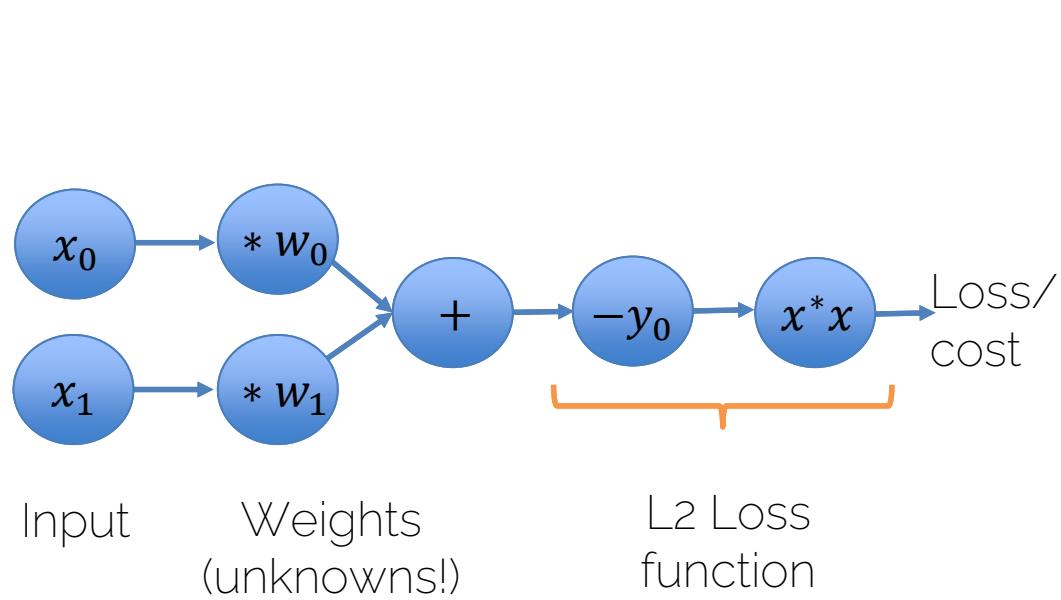
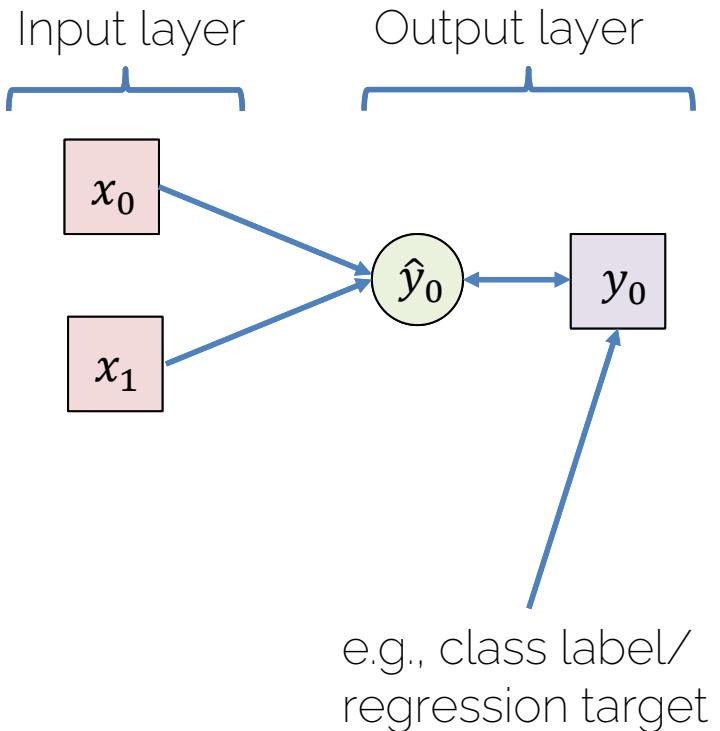
What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?



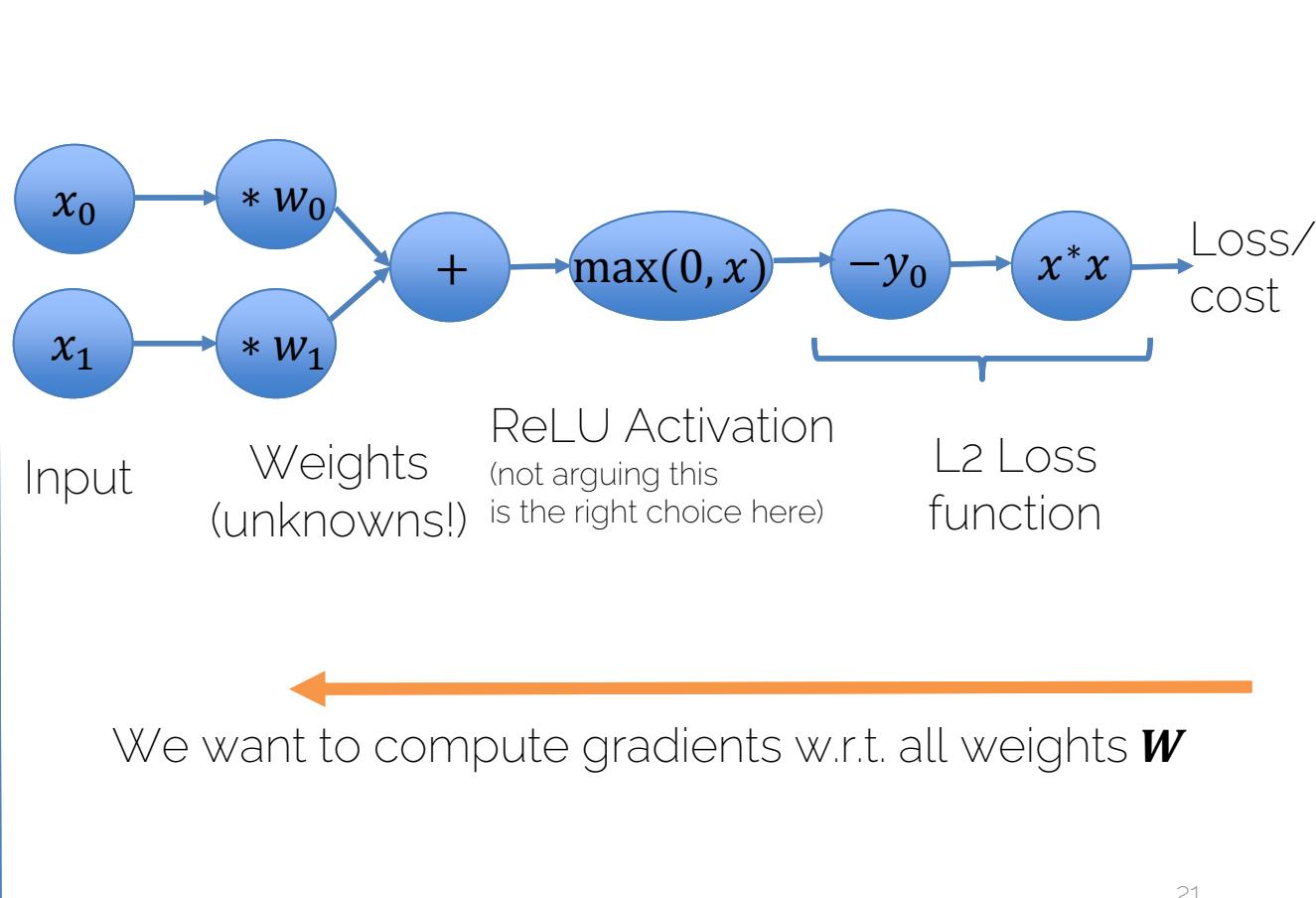
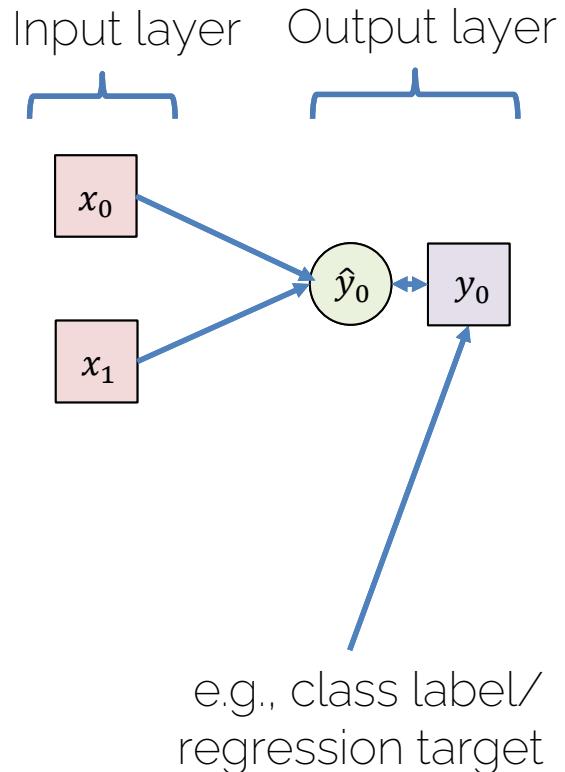
Compute Graphs -> Neural Networks

- 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

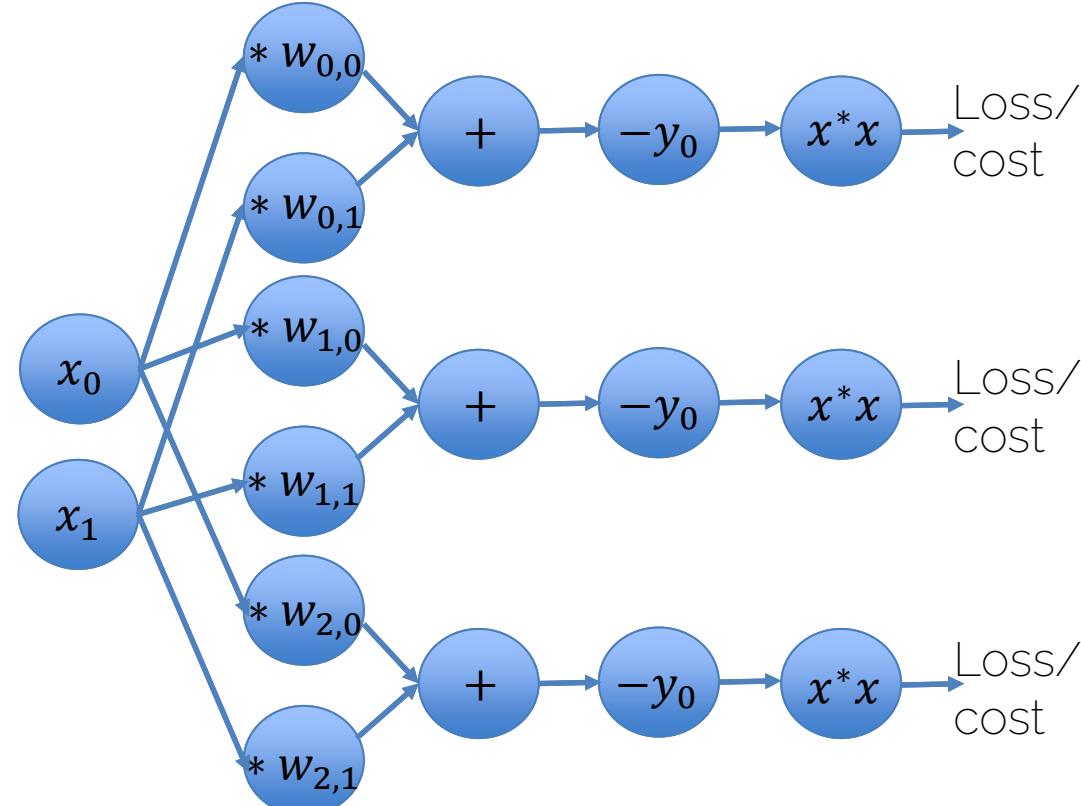
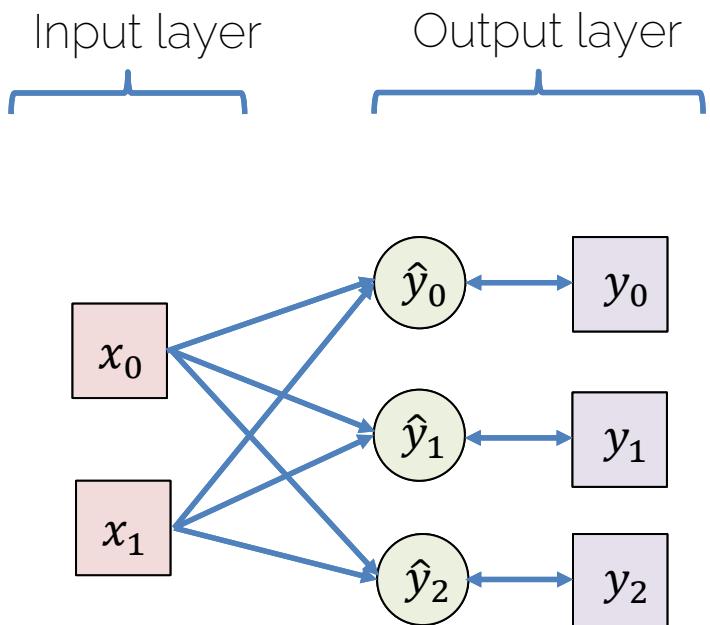
Compute Graphs -> Neural Networks



Compute Graphs -> Neural Networks

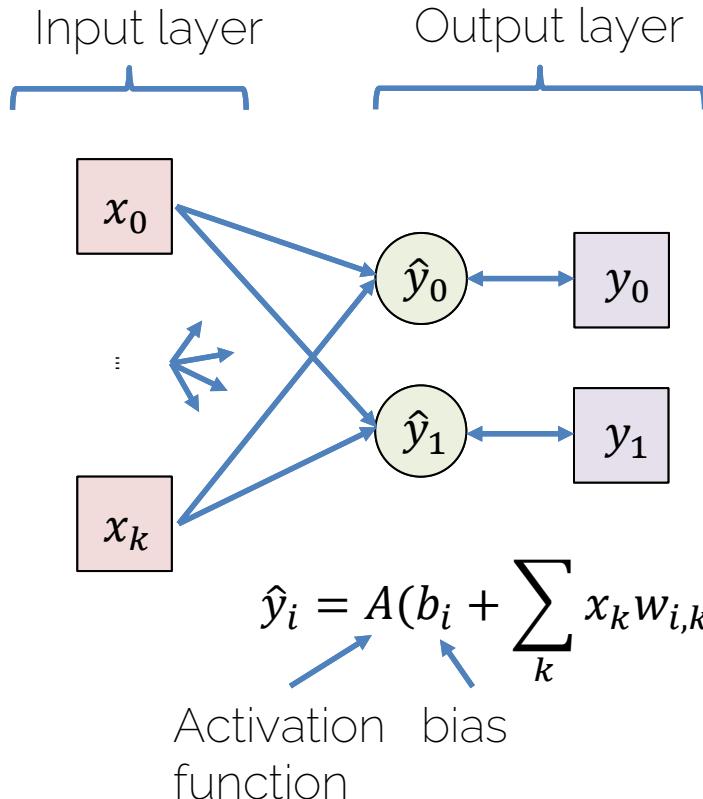


Compute Graphs -> Neural Networks



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 → simply sum up squares:

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

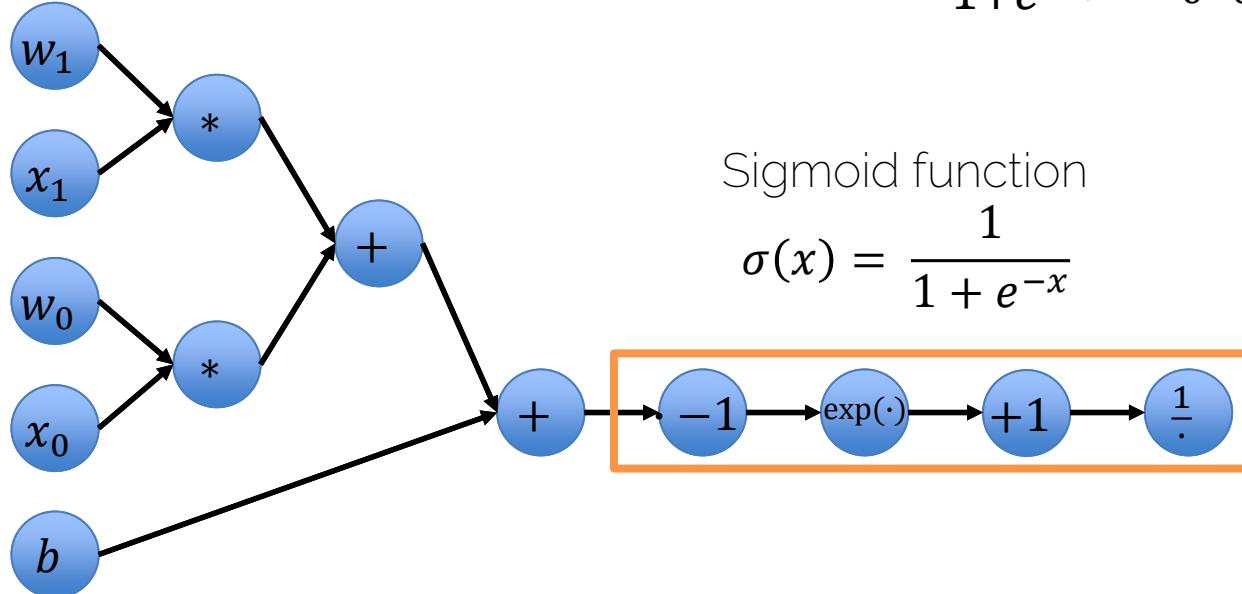
→ use chain rule to compute partials

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

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

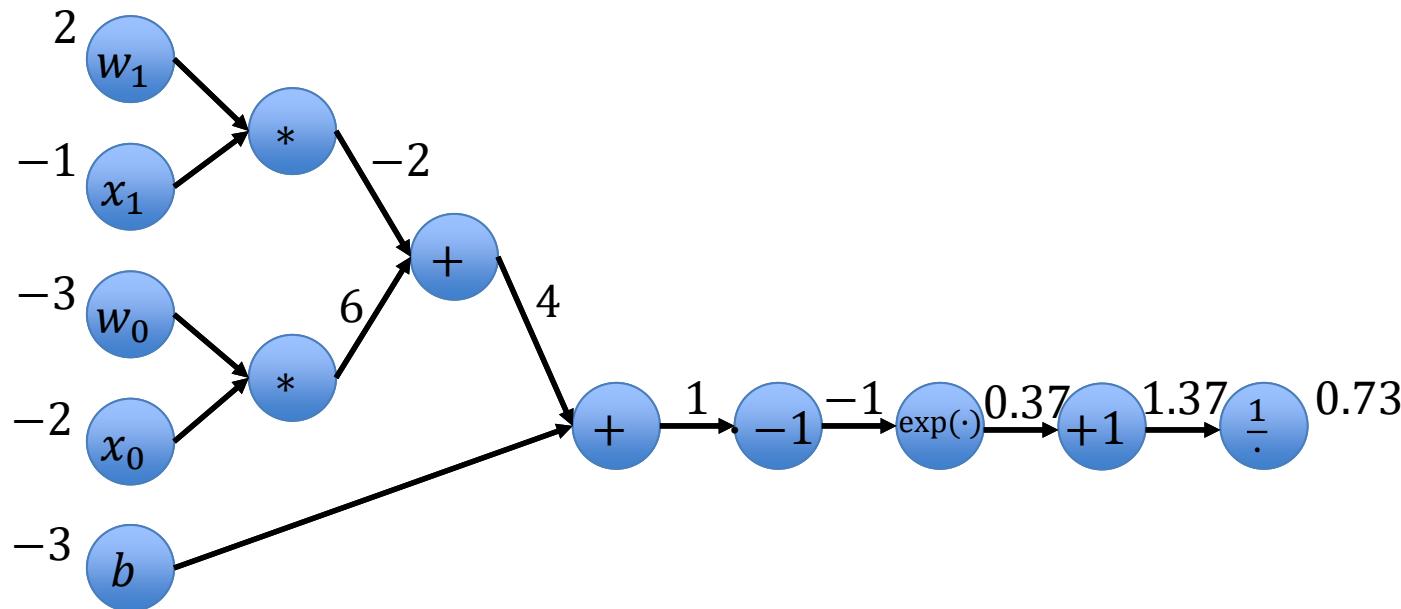
NNs as Computational Graphs

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



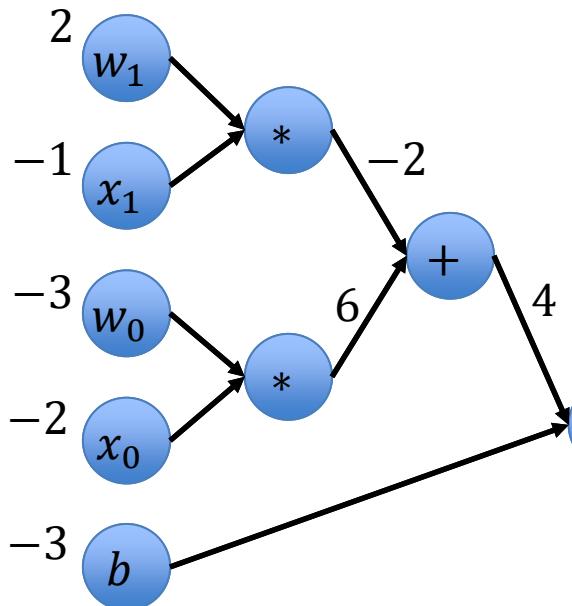
NNs as Computational Graphs

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



NNs as Computational Graphs

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



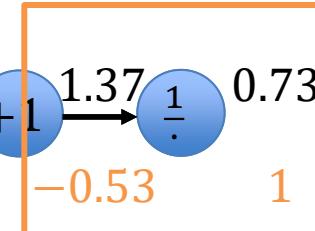
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

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

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

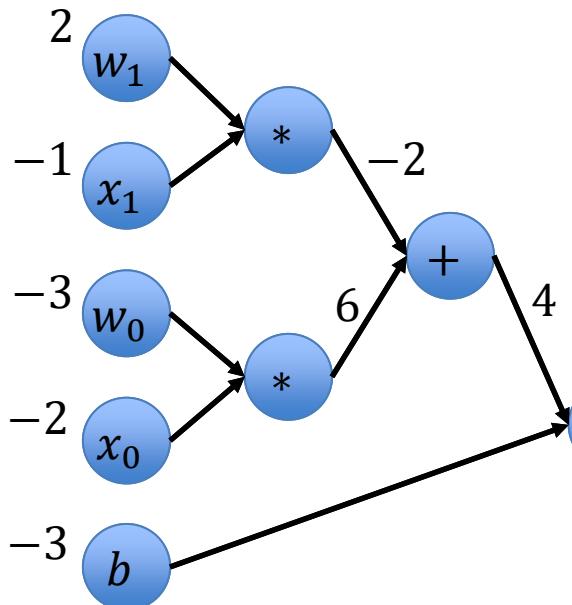
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$

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



NNs as Computational Graphs

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



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

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

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

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

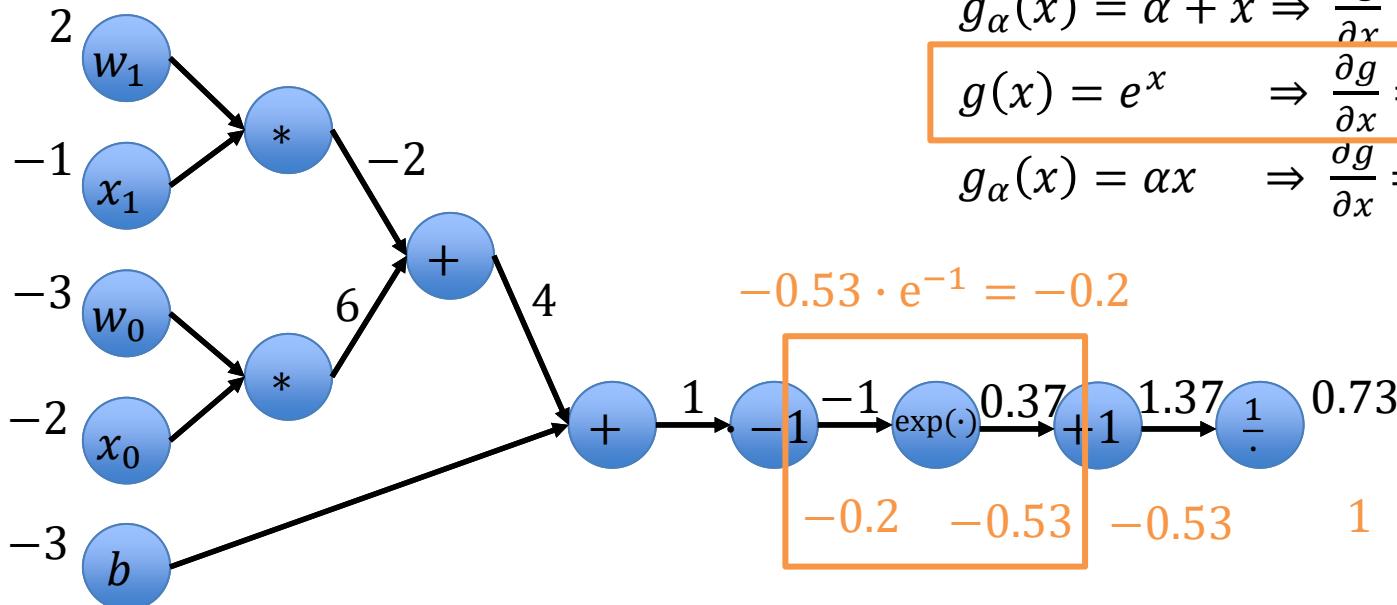
$-0.53 \cdot 1 = -0.53$

A detailed computational graph showing the intermediate values for each node in the main graph. The values are as follows:

- Top multiplication node (*): 0.37 (calculated as $2 \cdot (-1)$)
- Bottom multiplication node (*): 1.37 (calculated as $-3 \cdot (-2)$)
- Bottom addition node (+): -0.53 (calculated as $-0.53 + (-3)$)
- Top addition node (+): 1.37 (calculated as $0.37 + 1.37$)
- Final output: 0.73 (calculated as $\frac{1}{1 + \exp(-1.37)}$)

NNs as Computational Graphs

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



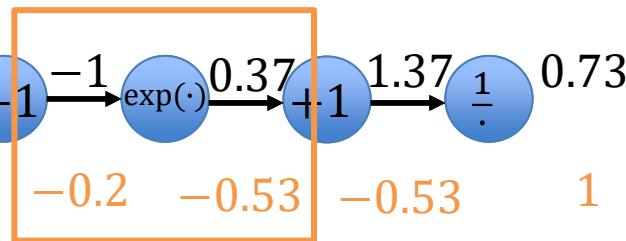
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$

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

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

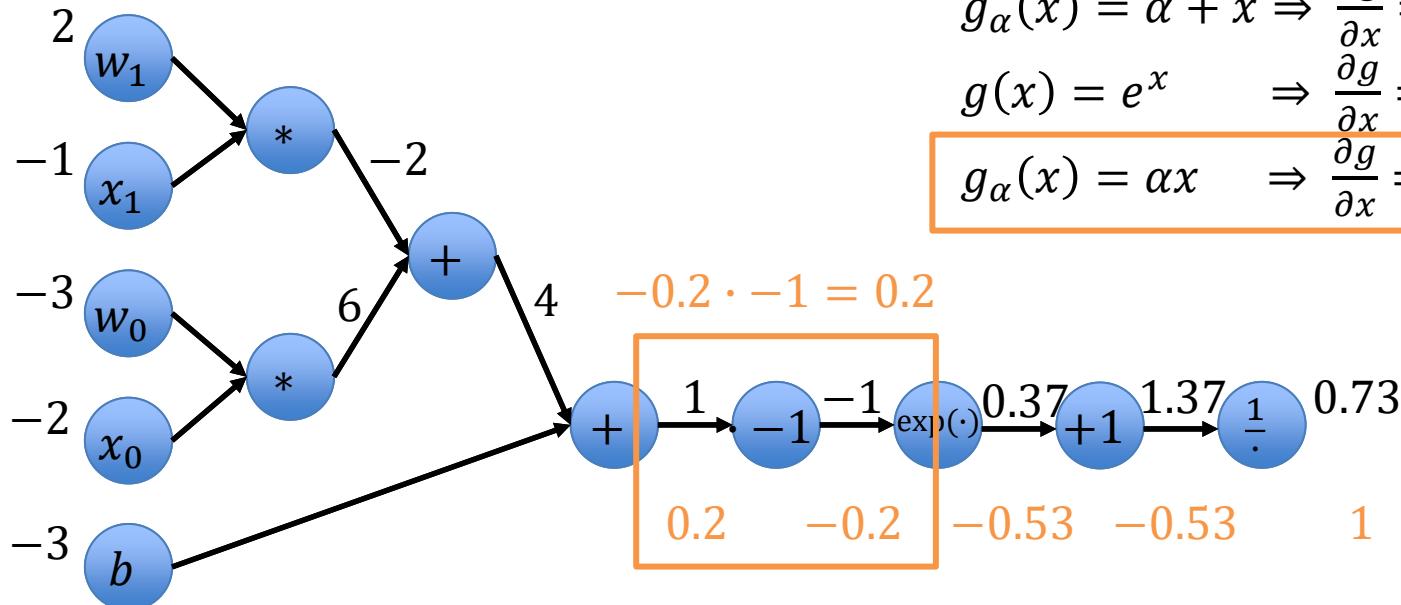
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$

$$-0.53 \cdot e^{-1} = -0.2$$



NNs as Computational Graphs

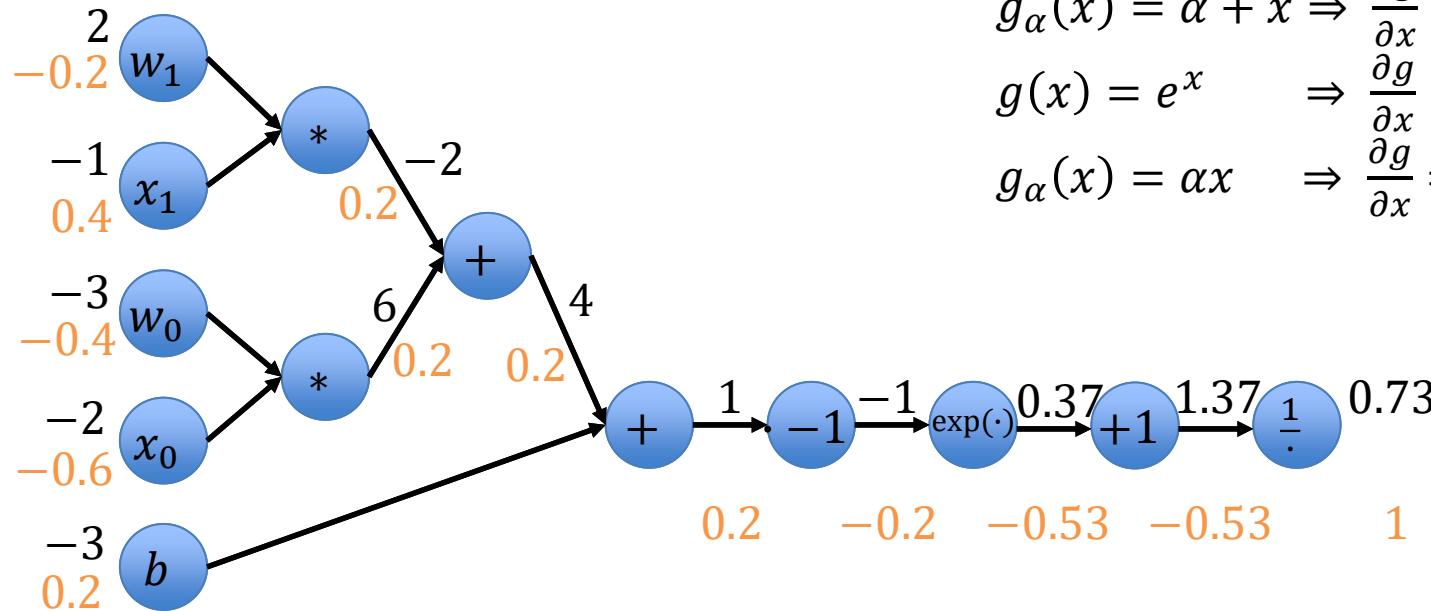
- $f(\mathbf{w}, \mathbf{x}) = \frac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$



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

NNs as Computational Graphs

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

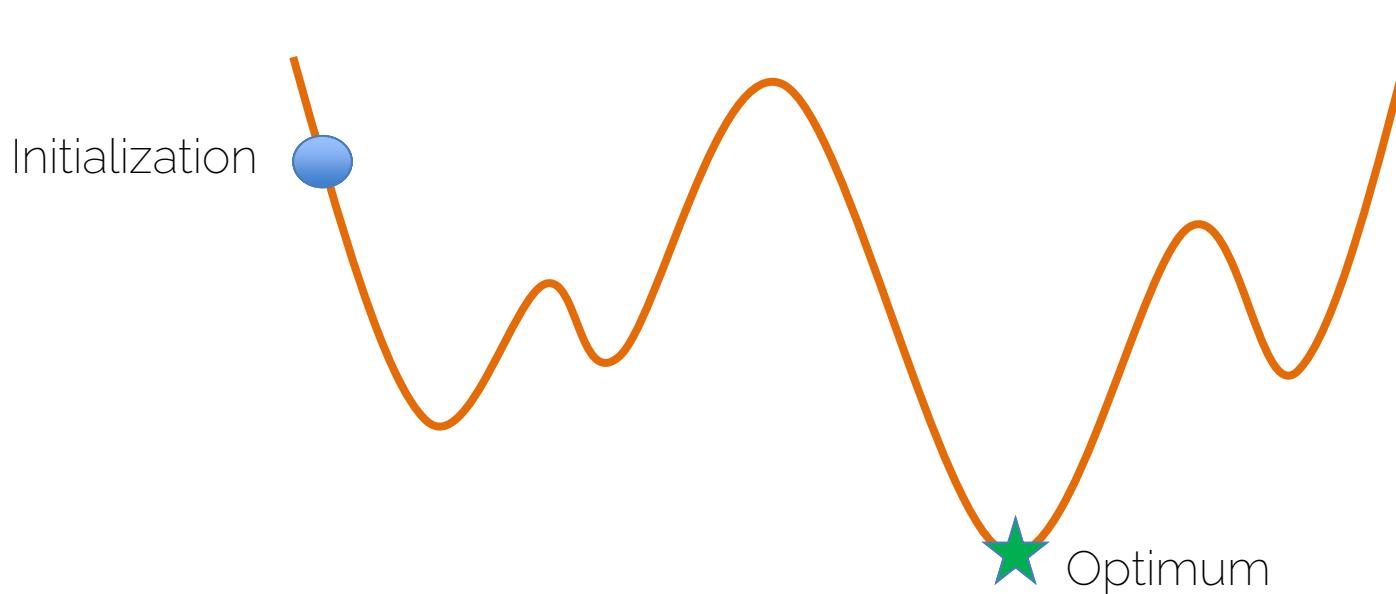


$$\begin{aligned} g(x) &= \frac{1}{x} & \Rightarrow \frac{\partial g}{\partial x} &= -\frac{1}{x^2} \\ g_\alpha(x) &= \alpha + x & \Rightarrow \frac{\partial g}{\partial x} &= 1 \\ g(x) &= e^x & \Rightarrow \frac{\partial g}{\partial x} &= e^x \\ g_\alpha(x) &= \alpha x & \Rightarrow \frac{\partial g}{\partial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

Gradient Descent

Gradient Descent

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



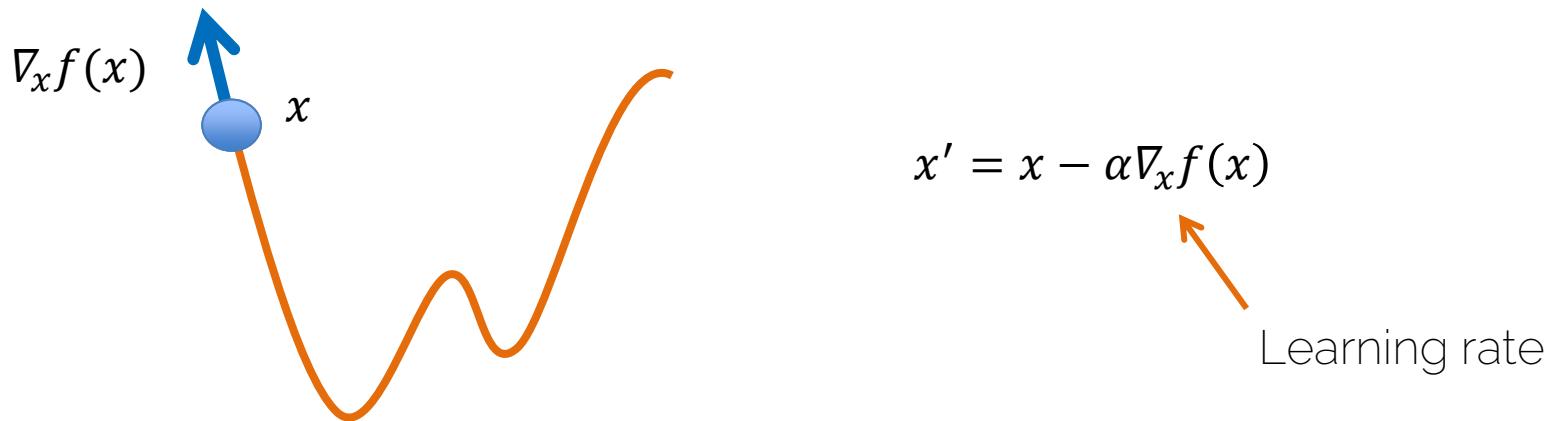
Gradient Descent

- From derivative to gradient

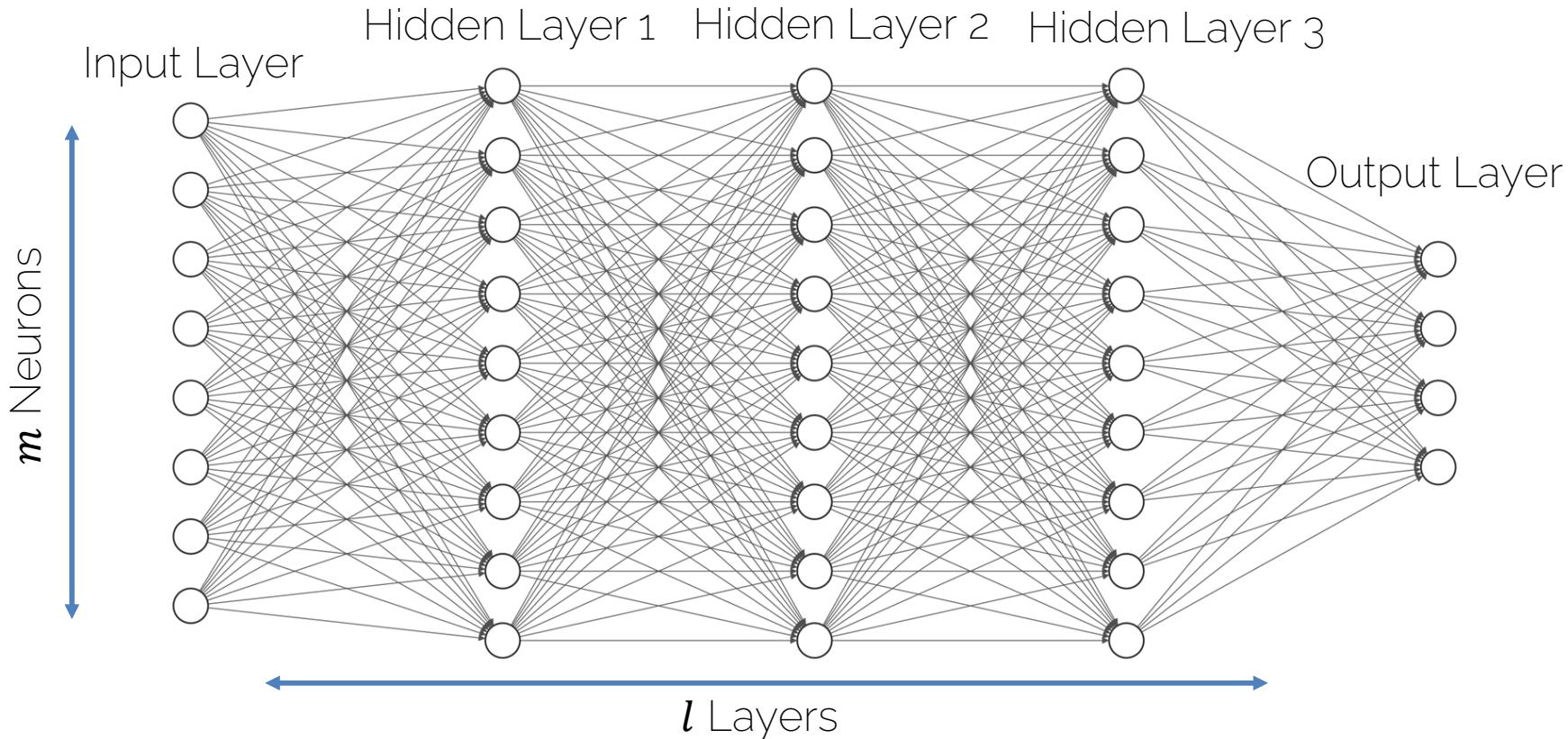
$$\frac{df(x)}{dx} \longrightarrow \nabla_x f(x)$$

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



Gradient Descent for Neural Networks



Gradient Descent for Neural Networks

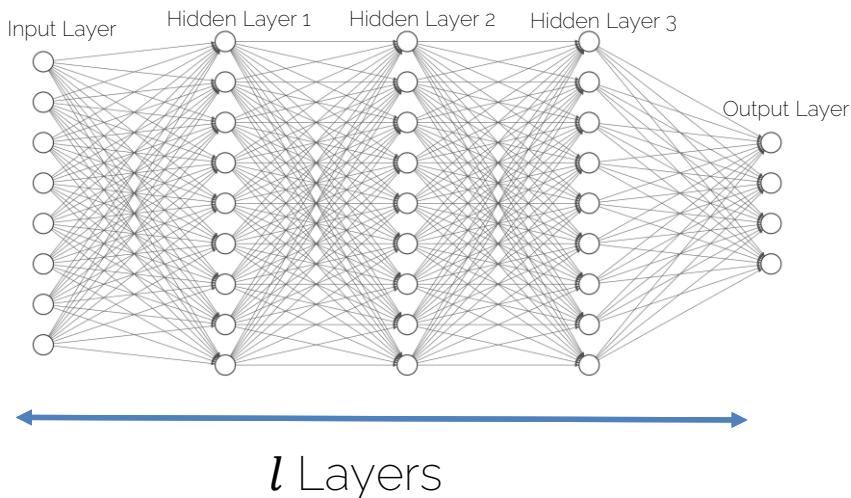
For a given training pair $\{\mathbf{x}, \mathbf{y}\}$, we want to update all weights, i.e., we need to compute the derivatives w.r.t. to all weights:

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

m Neurons

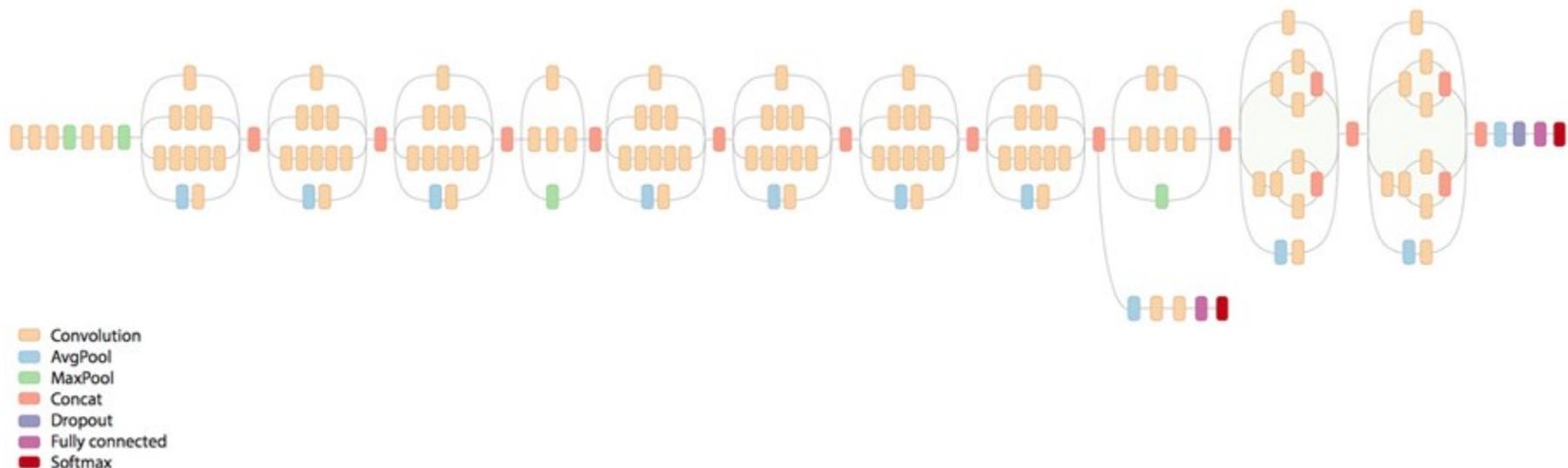
Gradient step:

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



NNs can Become Quite Complex...

- These graphs can be huge!



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

The Flow of the Gradients

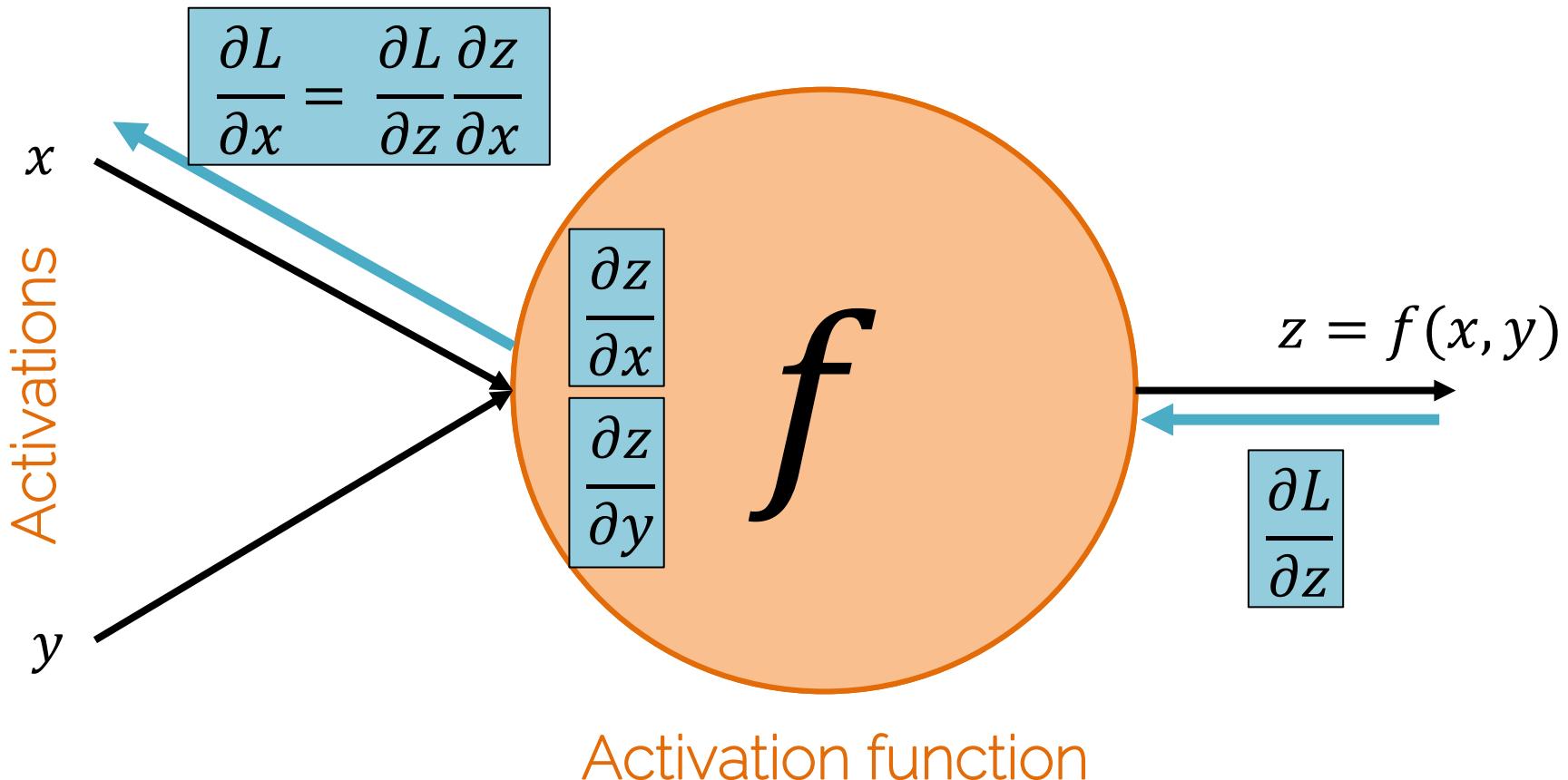
- Many many many many of these nodes form a neural network

NEURONS

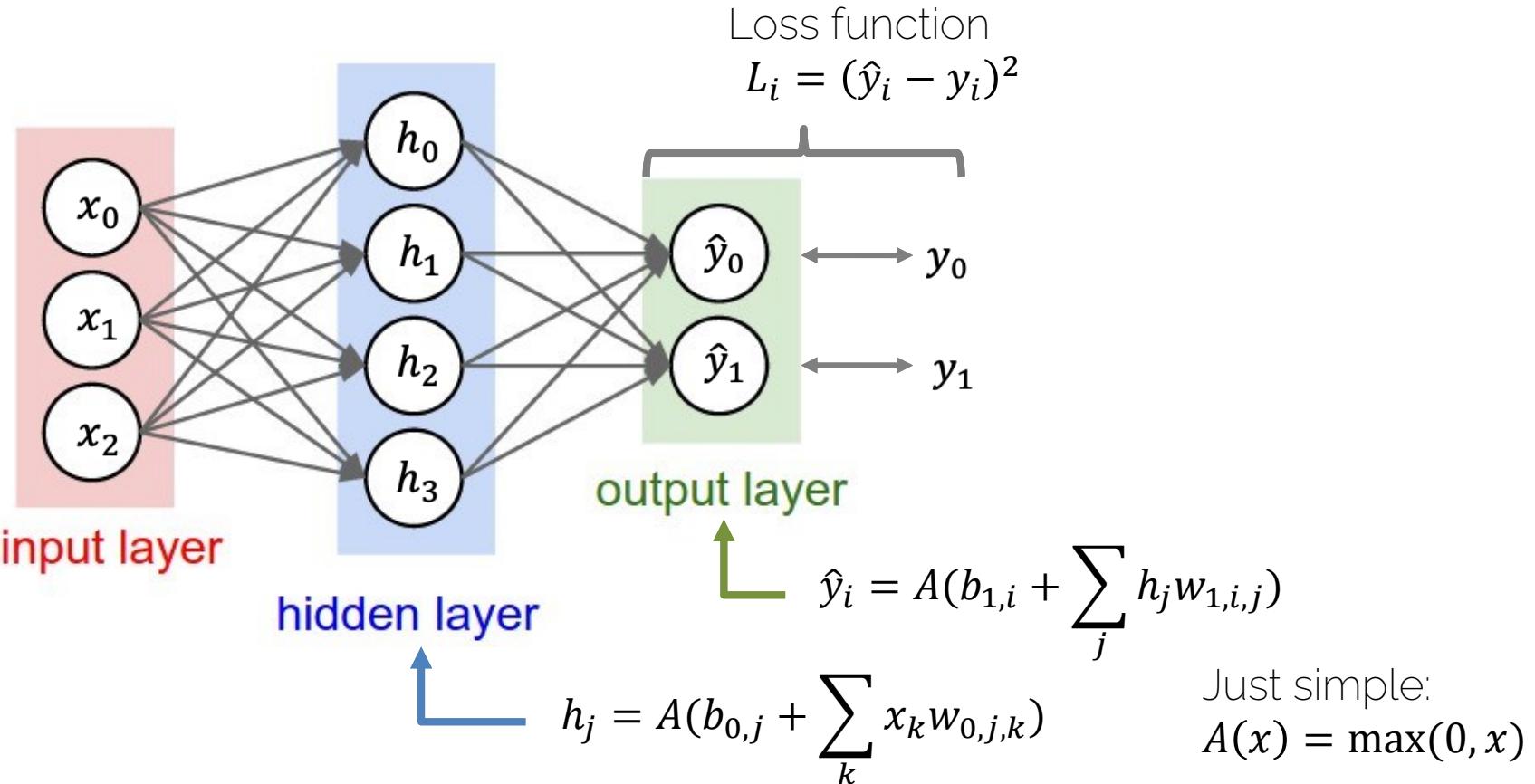
- Each one has its own work to do

FORWARD AND BACKWARD PASS

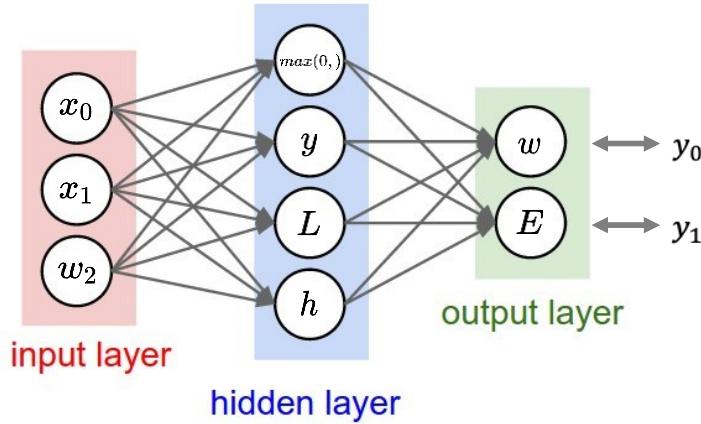
The Flow of the Gradients



Gradient Descent for Neural Networks



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

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

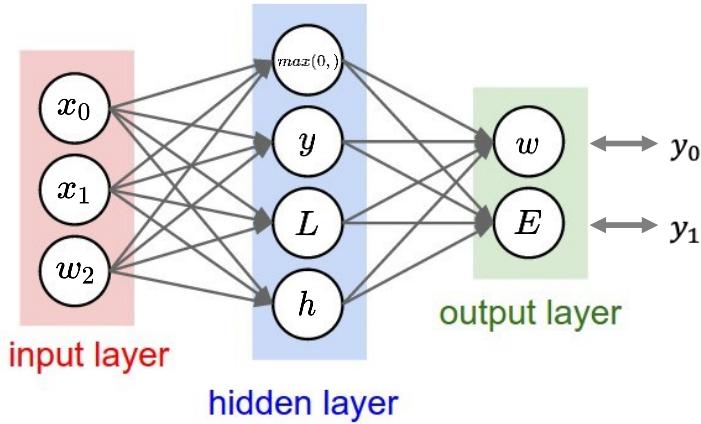
$$\frac{\partial L_i}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)$$

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

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

...

Gradient Descent for Neural Networks



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

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

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

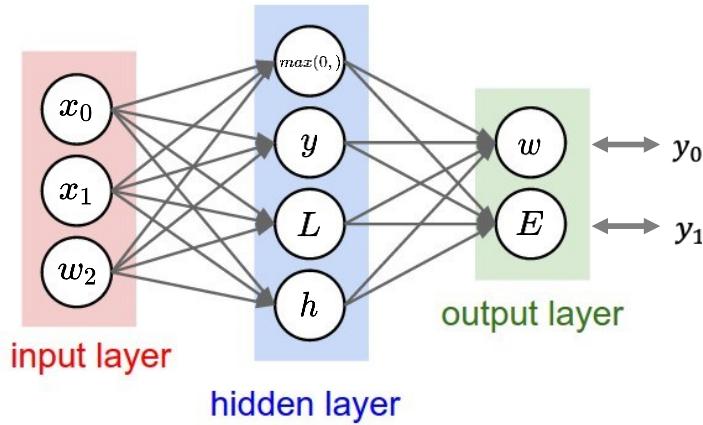
How many unknown weights?

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

#neurons \cdot #input channels + #biases

Note that some activations have also weights

Derivatives of Cross Entropy Loss



Binary Cross Entropy loss

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

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

output scores

Gradients of weights of last layer:

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

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

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

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

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

Derivatives of Cross Entropy Loss

Gradients of weights of first layer:

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

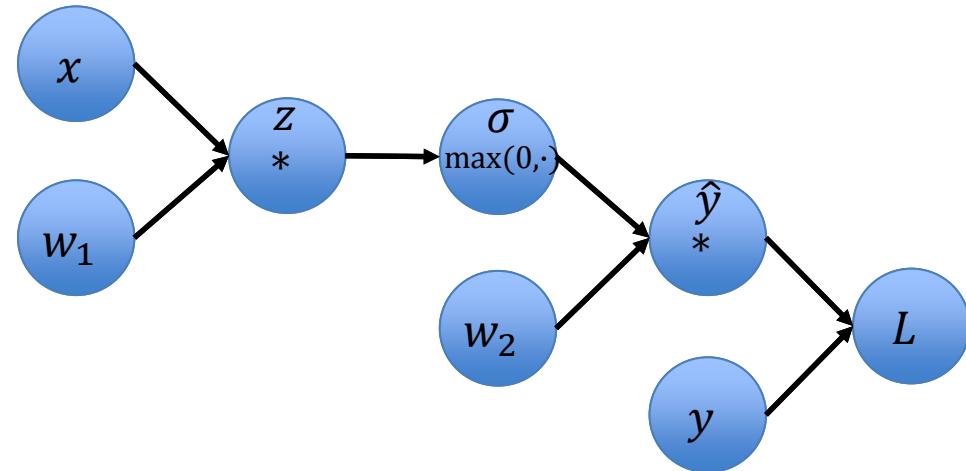
$$\frac{\partial L}{\partial s_j^1} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j^1} = \sum_{i=1}^{n_{out}} (\hat{y}_i - y_i) w_{ji} (h_j (1 - h_j))$$

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

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

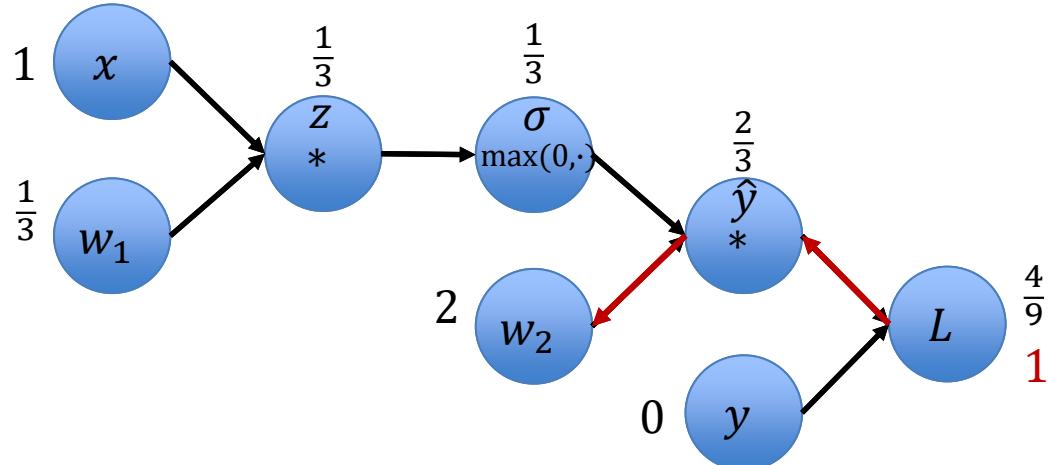
Initialize $x = 1$, $y = 0$,
 $w_1 = \frac{1}{3}$, $w_2 = 2$

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

In our case $n, d = 1$:

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

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



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

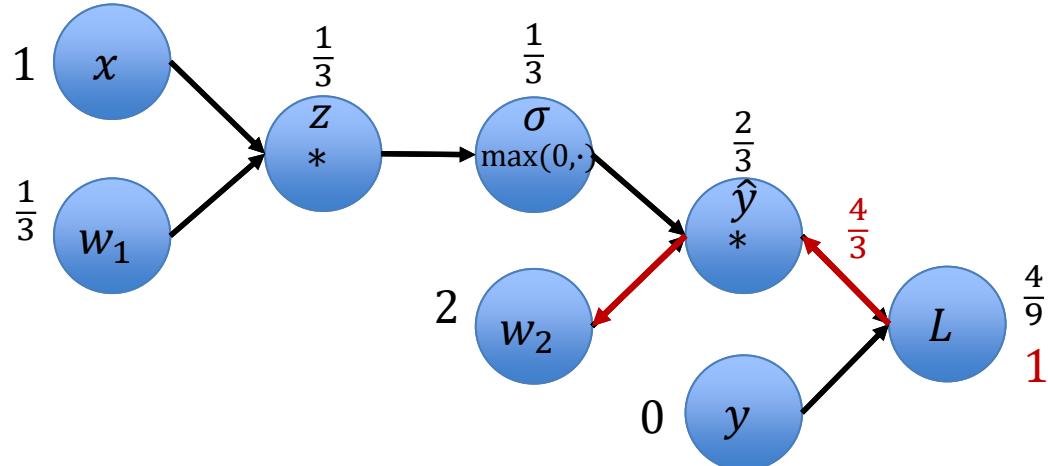
Initialize $x = 1$, $y = 0$,
 $w_1 = \frac{1}{3}$, $w_2 = 2$

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

In our case $n, d = 1$:

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

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



Backpropagation

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

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

Gradient Descent for Neural Networks

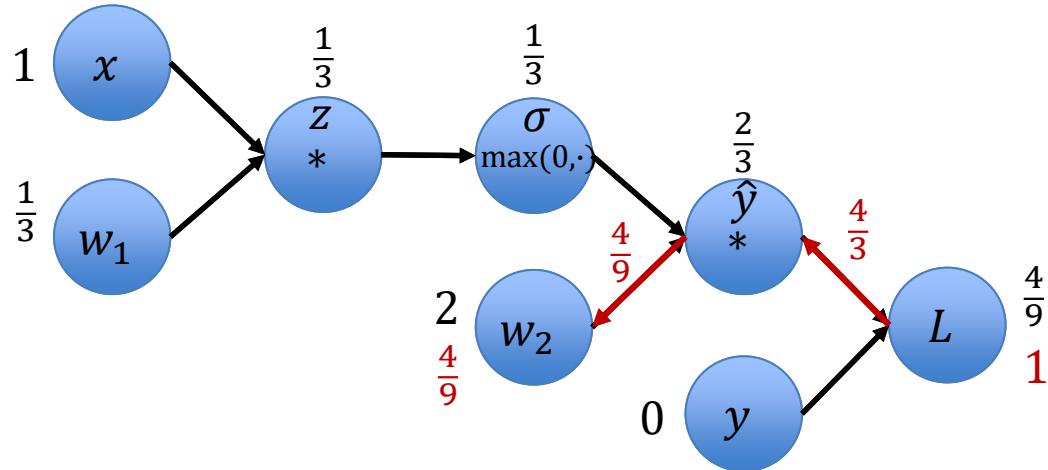
Initialize $x = 1, y = 0,$
 $w_1 = \frac{1}{3}, w_2 = 2$

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

In our case $n, d = 1:$

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

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



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

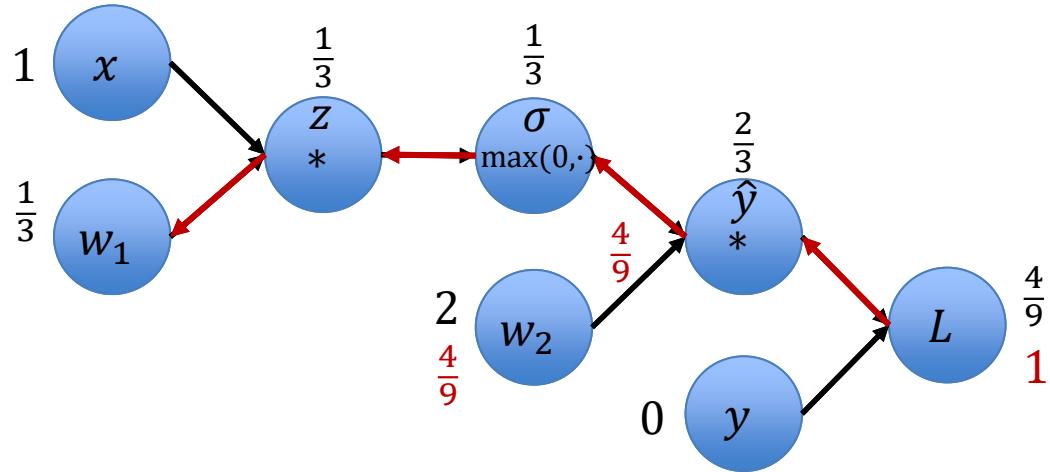
In our case $n, d = 1$:

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

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

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

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



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

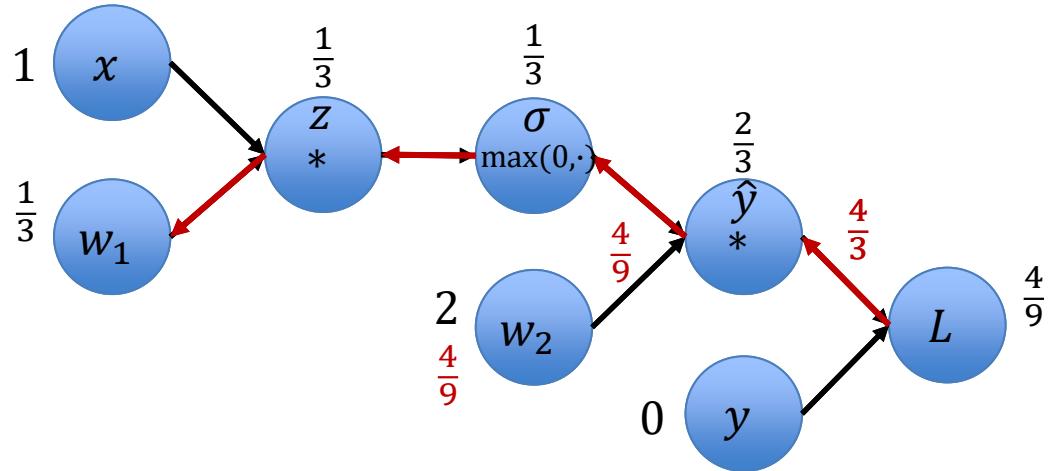
In our case $n, d = 1$:

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

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

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

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



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

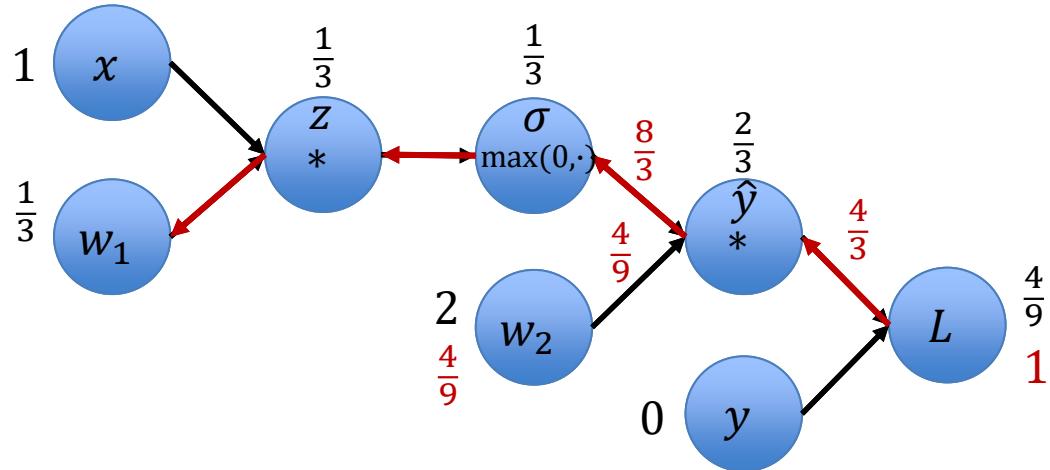
In our case $n, d = 1$:

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

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

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

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



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

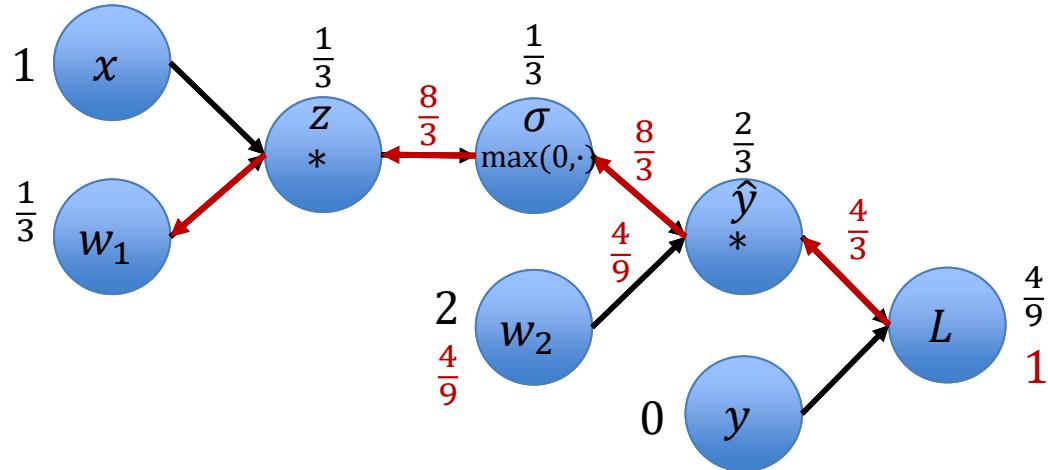
In our case $n, d = 1$:

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

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

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

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



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

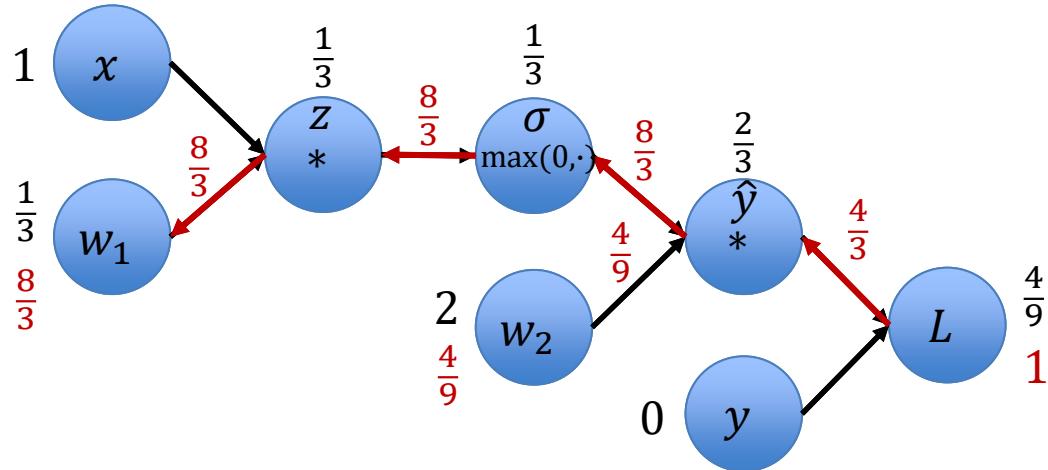
In our case $n, d = 1$:

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

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

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

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



Backpropagation

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

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

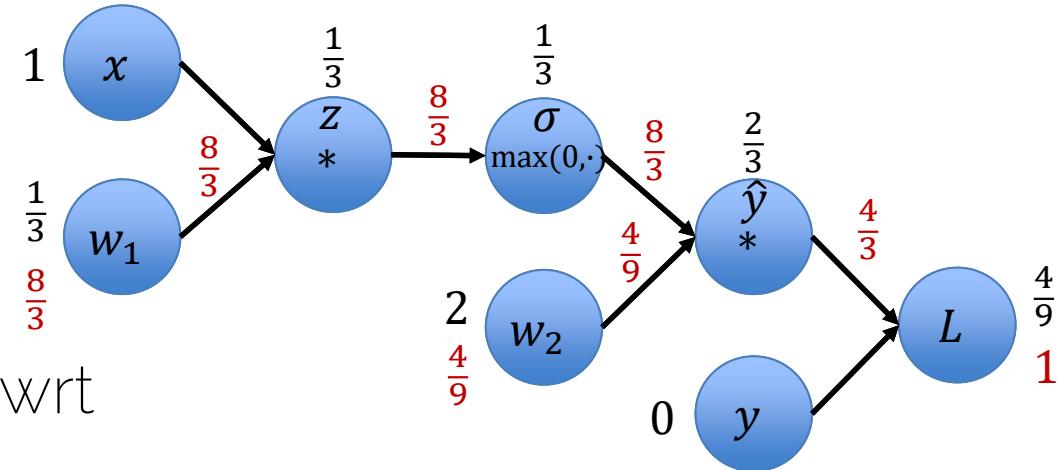
Gradient Descent for Neural Networks

- Function we want to optimize:

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

- 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 α ?

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

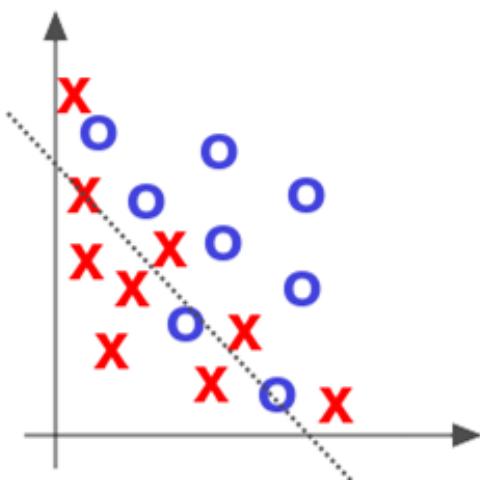
- Split your data



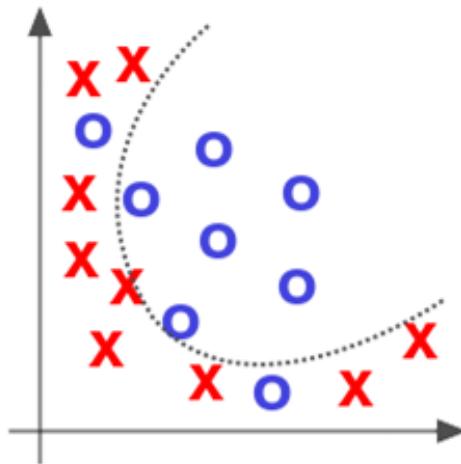
Find your hyperparameters

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

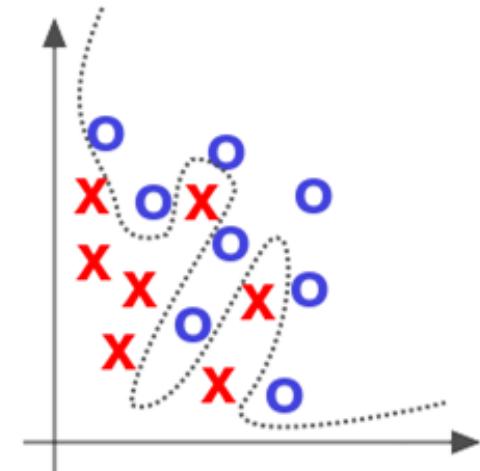
Over- and Underfitting



Underfitted



Appropriate

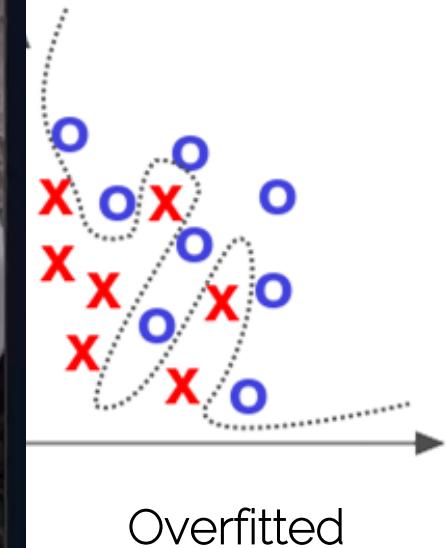
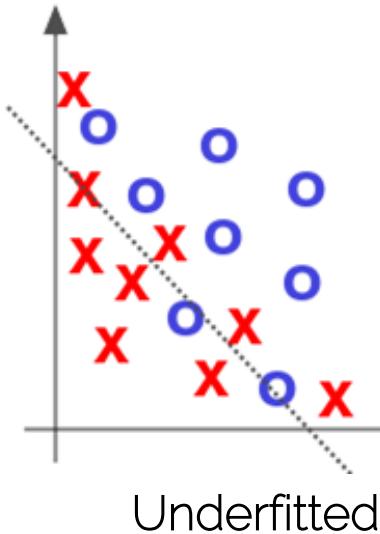


Overfitted

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

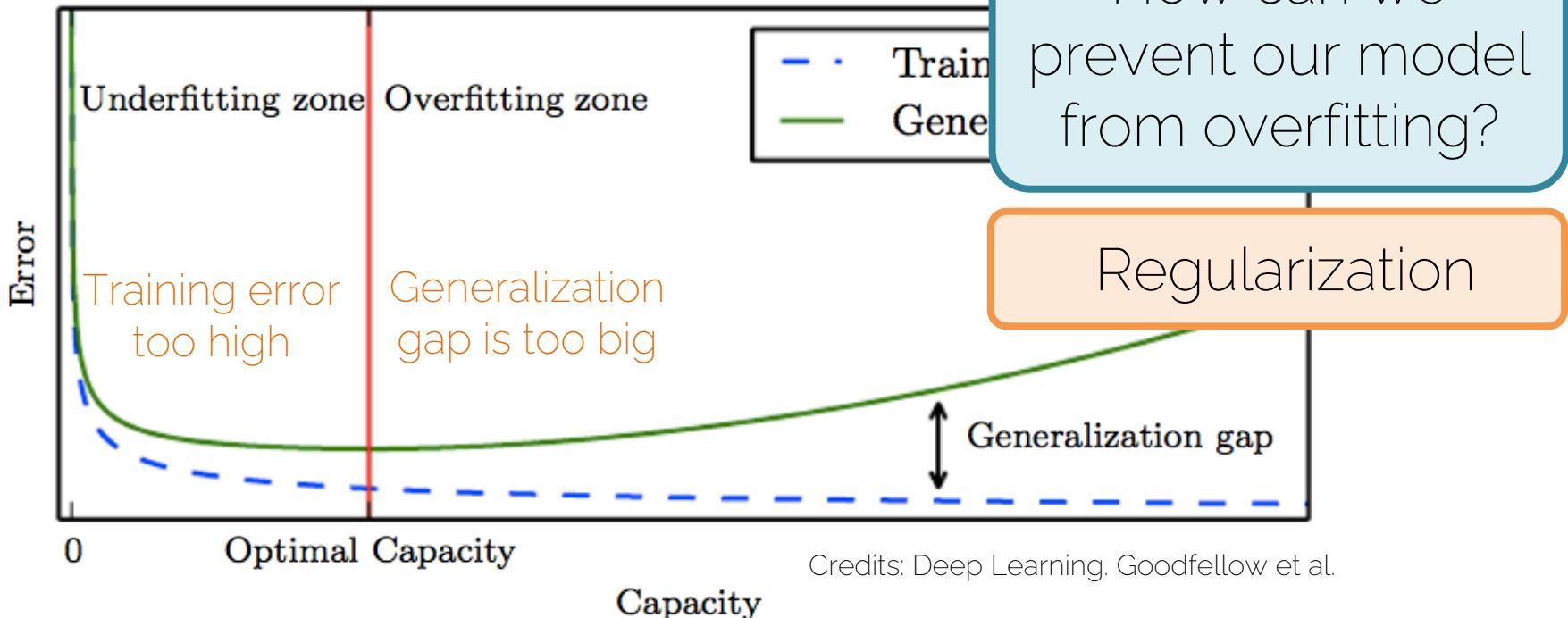
Over- and Underfitting

ML Engineers looking at their classification model running on the test set.



Training a Neural Network

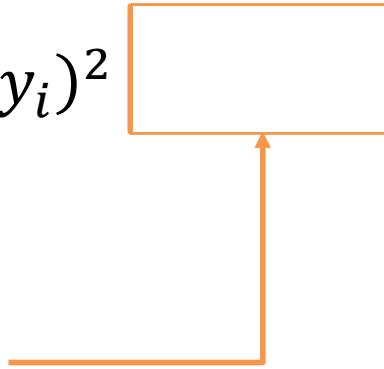
- Training/ Validation curve



Regularization

- Loss function $L(\mathbf{y}, \hat{\mathbf{y}}, \boldsymbol{\theta}) = \sum_{i=1}^n (\hat{y}_i - y_i)^2$
- Regularization techniques
 - L₂ regularization
 - L₁ regularization
 - Max norm regularization
 - Dropout
 - Early stopping
 - ...

Add regularization term to loss function



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
 - ...
-
- The diagram consists of two orange brackets. One bracket groups the regularization techniques (L2, L1, Max norm, Dropout, Early stopping, ...) and points to the term $\lambda R(\boldsymbol{\theta})$ in the loss function formula. Another bracket groups the regularization term and points to the text "Add regularization term to loss function".

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

$$\theta_1 \longrightarrow 0 + 0.75^2 + 0 = 0.5625$$

$$\theta_2 \longrightarrow 0.25^2 + 0.5^2 + 0.25^2 = 0.375 \quad \text{Minimization}$$

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

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

$$\theta_1 \longrightarrow 0 + 0.75 + 0 = 0.75 \quad \text{Minimization}$$

$$\theta_2 \longrightarrow 0.25 + 0.5 + 0.25 = 1$$

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

Regularization: Example

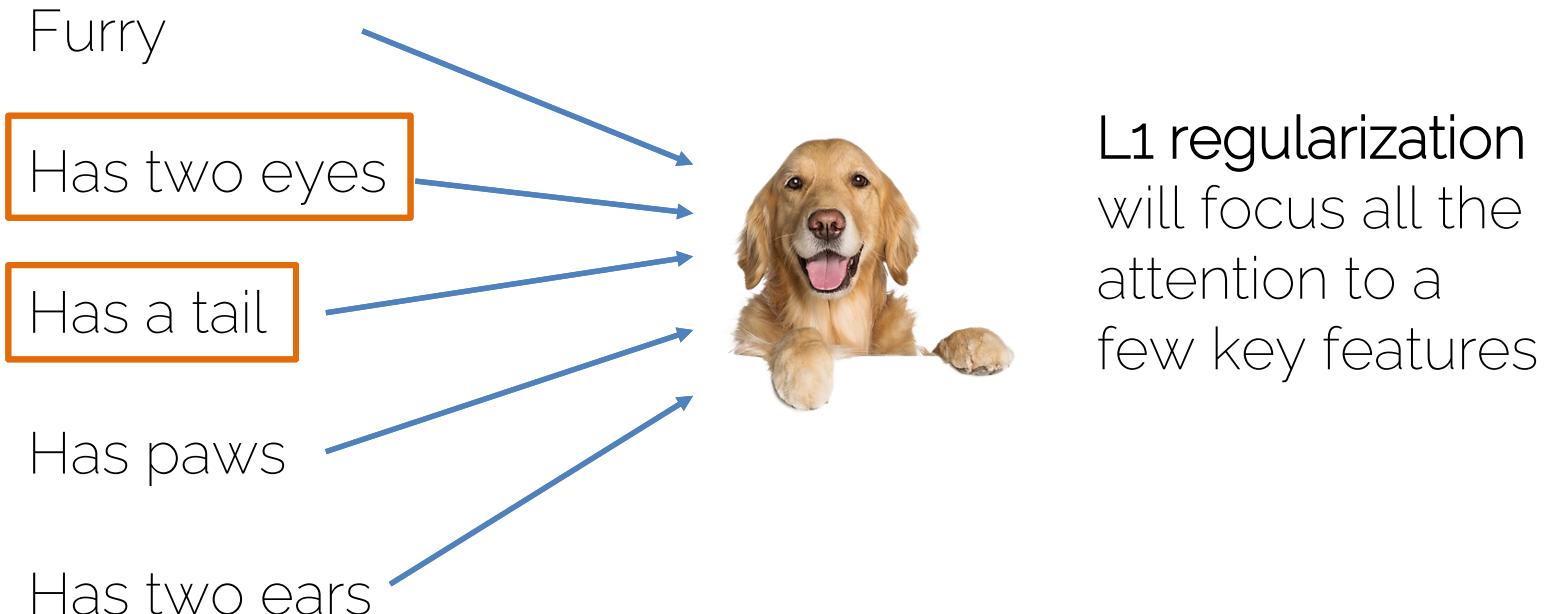
- Input: 3 features $\mathbf{x} = [1, 2, 1]$
- Two linear classifiers that give the same result:
- $\theta_1 = [0, 0.75, 0]$  Ignores 2 features
- $\theta_2 = [0.25, 0.5, 0.25]$  Takes information from all features

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**

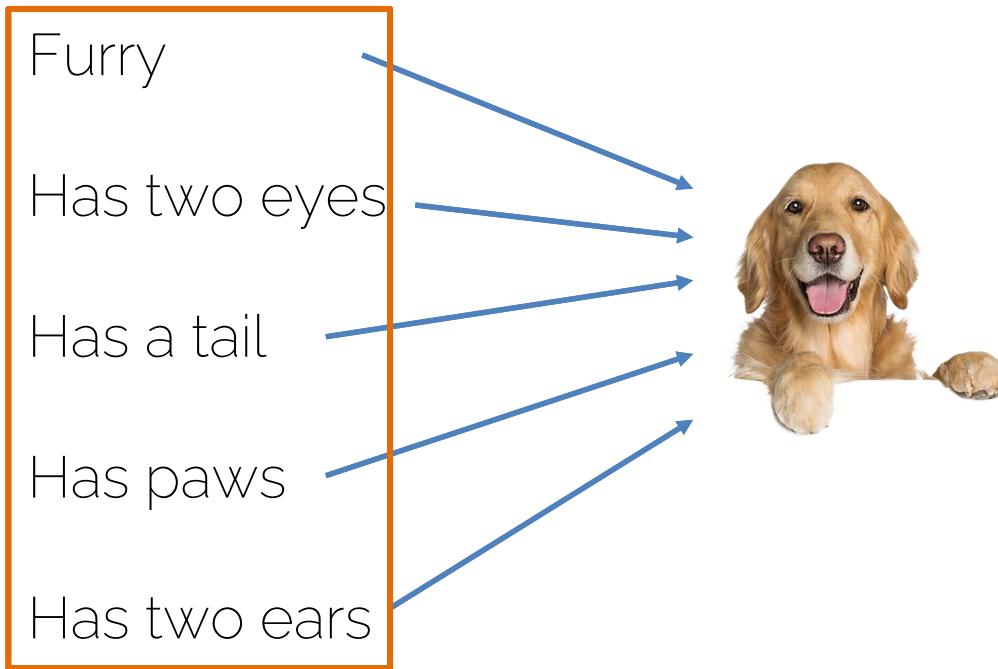
Regularization: Effect

- Dog classifier takes different inputs



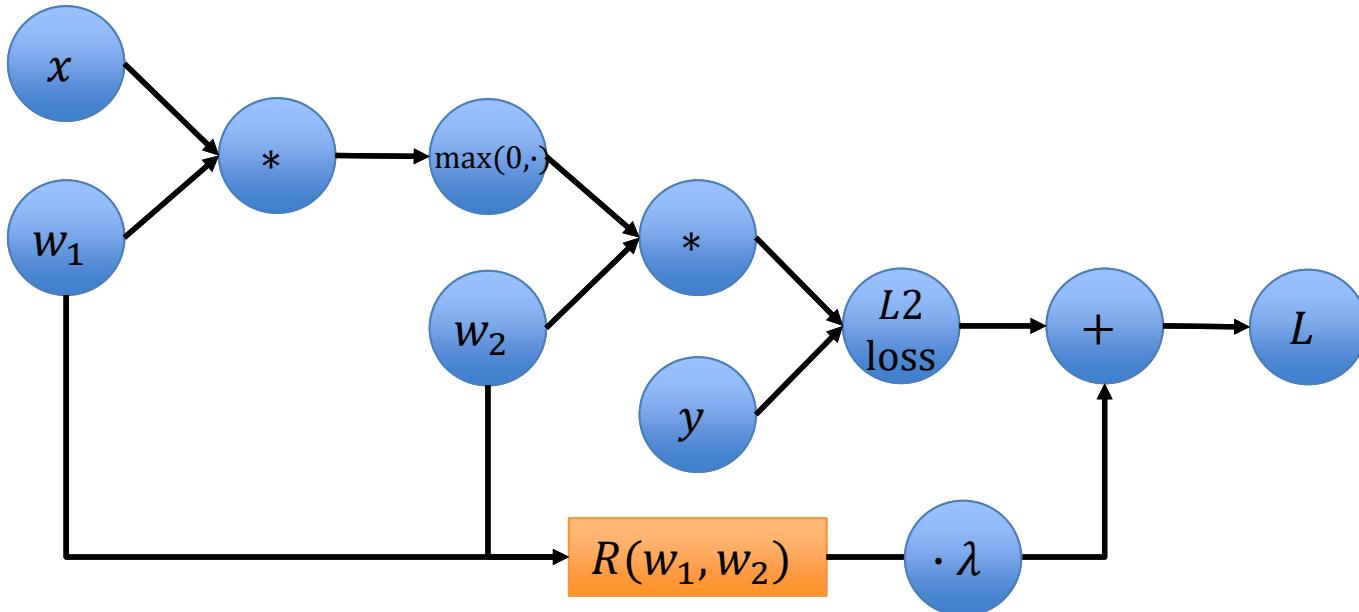
Regularization: Effect

- Dog classifier takes different inputs



L2 regularization will take all information into account to make decisions

Regularization for Neural Networks

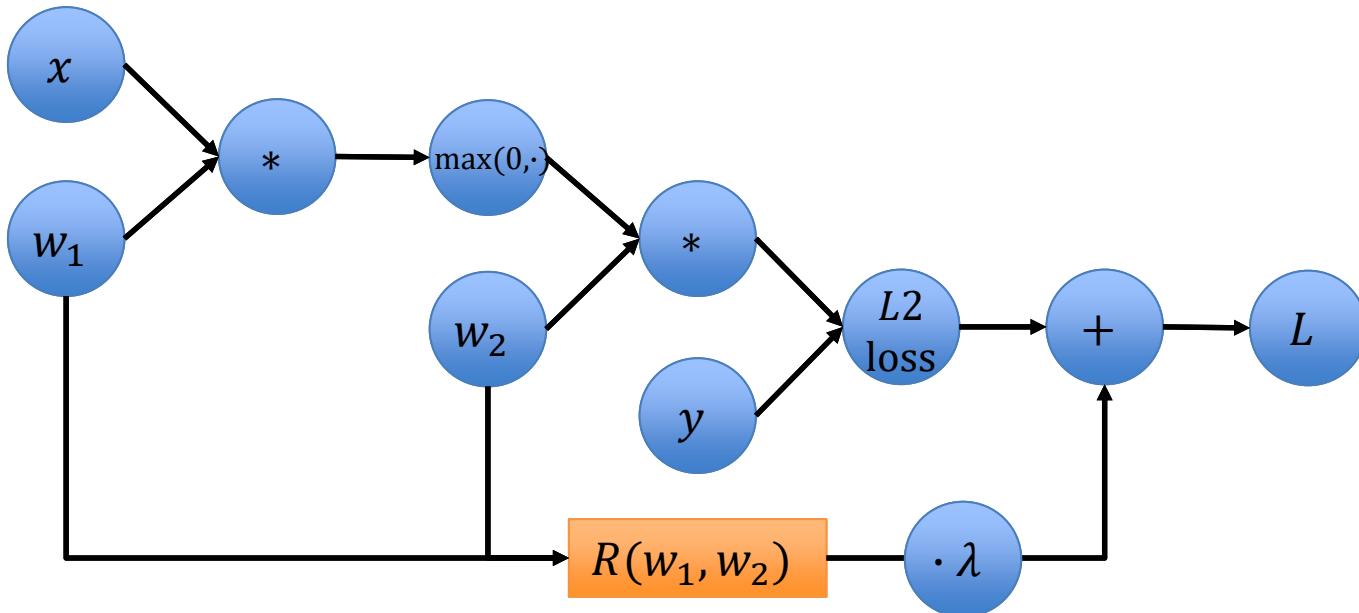


Combining nodes:

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

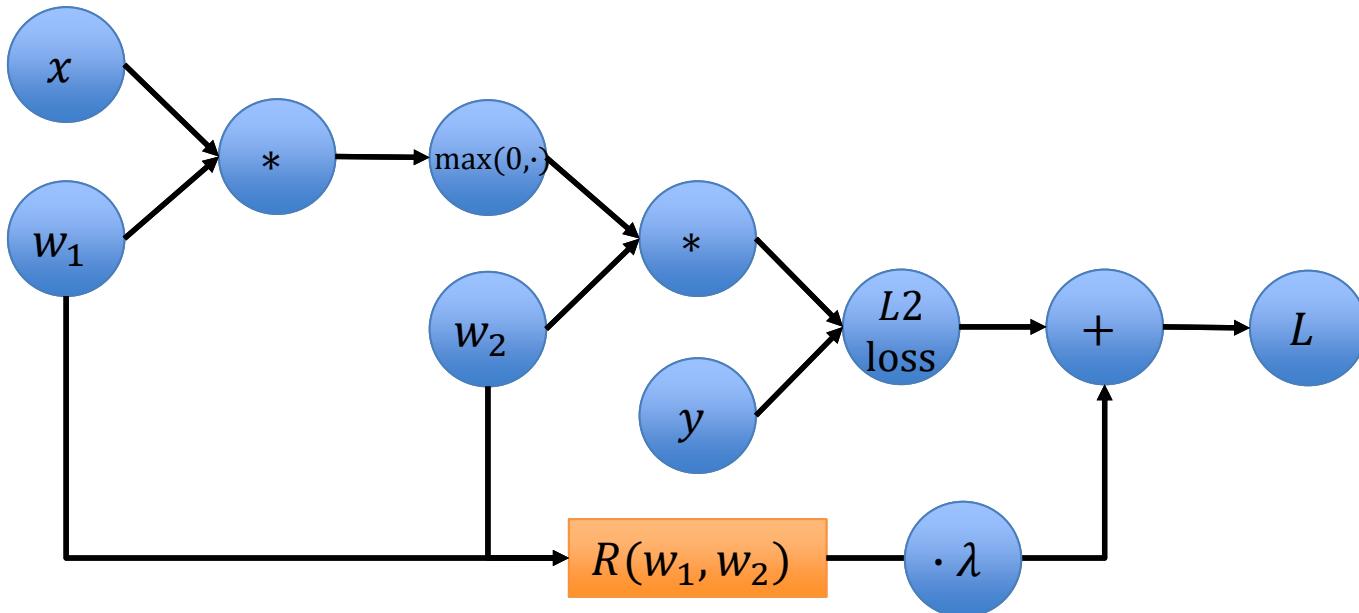


Combining nodes:

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

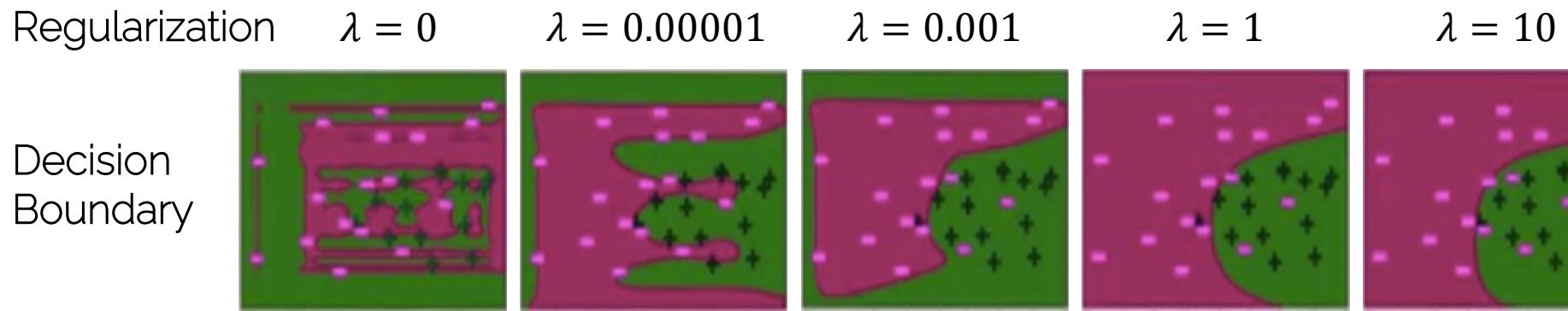


Combining nodes:

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



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!)

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