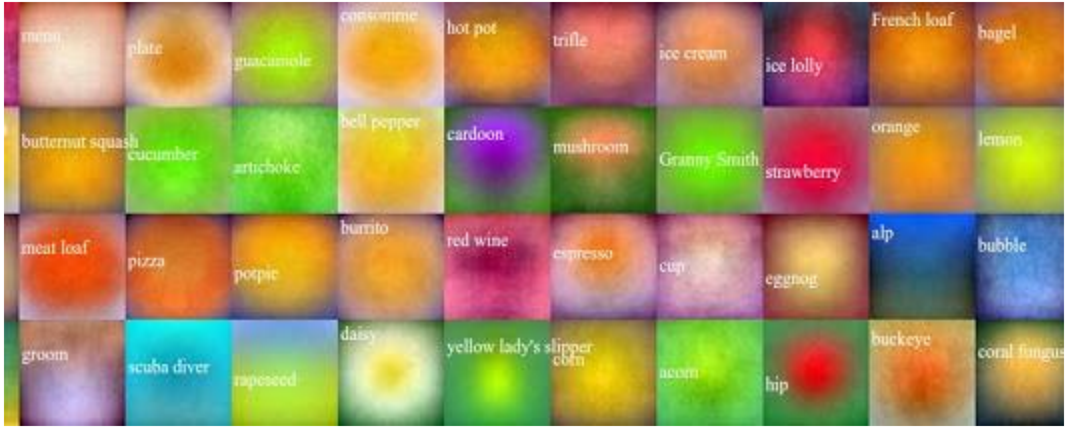
Lecture 3 recap

Beyond linear

- Linear score function $f = Wx$

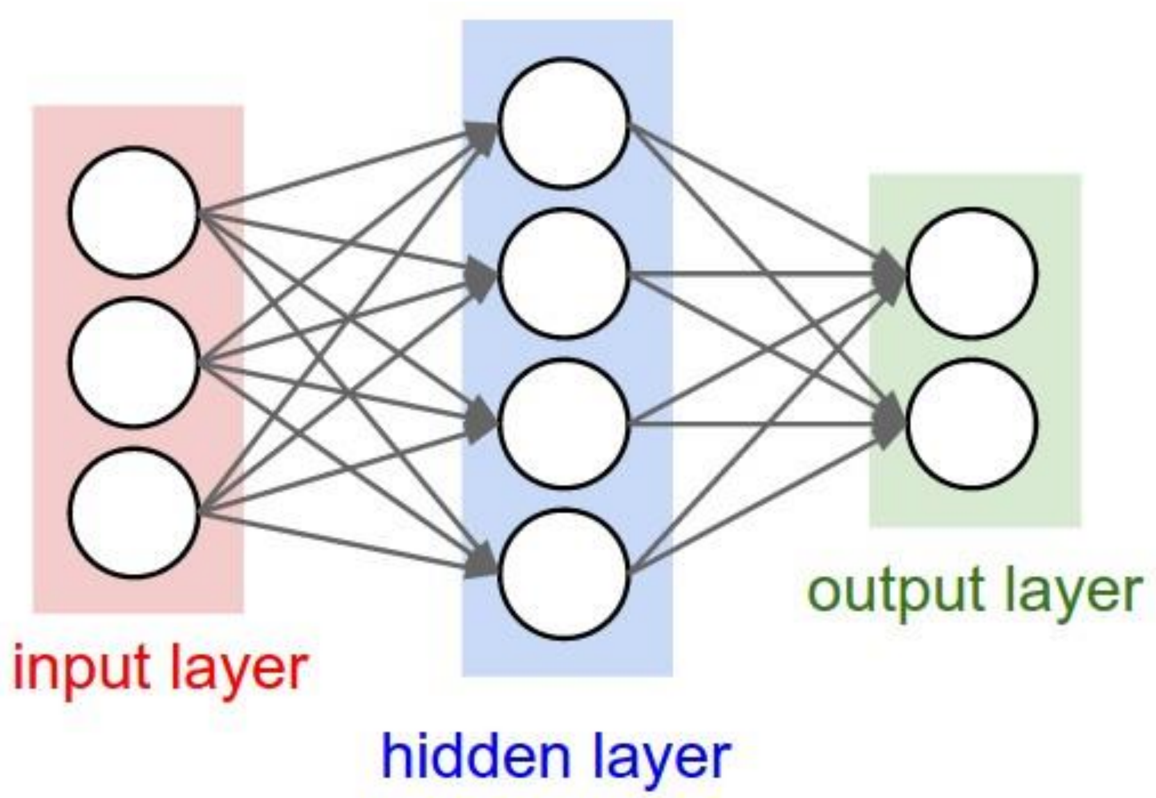
On CIFAR-10



On ImageNet

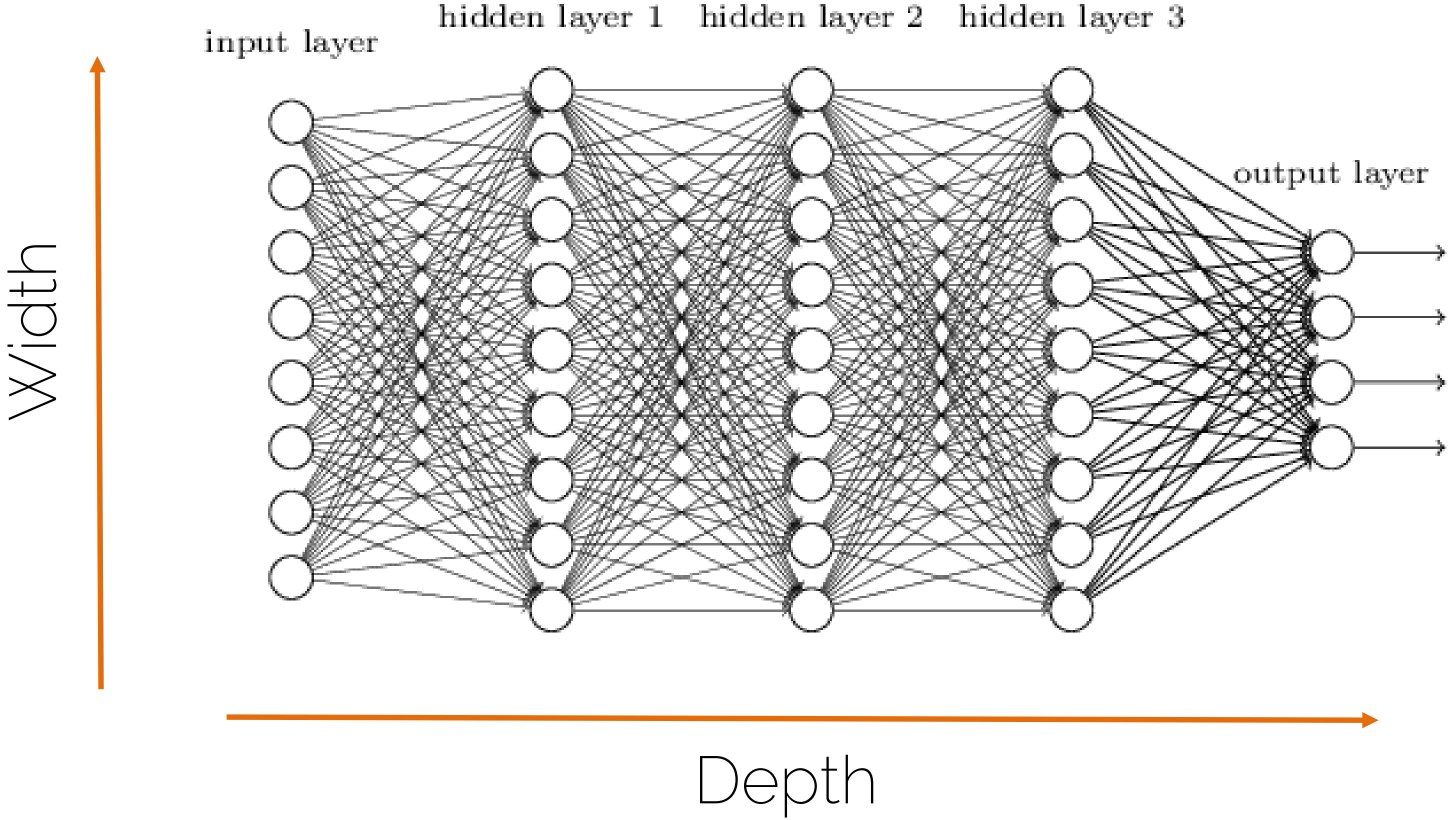
Credit: Li/Karpathy/Johnson

Neural Network



Credit: Li/Karpathy/Johnson

Neural Network



Neural Network

- Linear score function $f = Wx$

- Neural network is a nesting of 'functions'
  - 2-layers: $f = W_2 \max(0, W_1 x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 4-layers: $f = W_4 \tanh\,(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x))))$
  - … up to hundreds of layers

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 con't

The importance of gradients

- All 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?

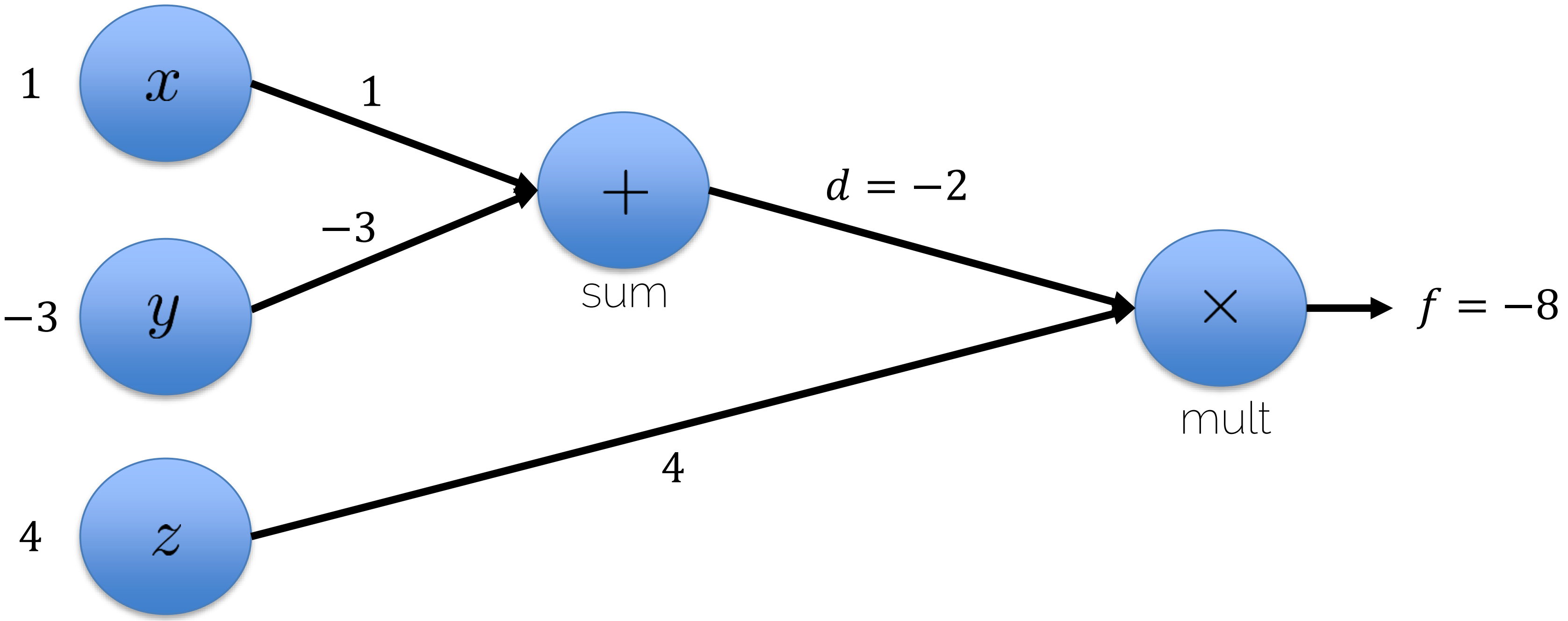
- Break down gradient computation **Backpropagation**

Rumelhart 1986

Backprop: Forward Pass

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

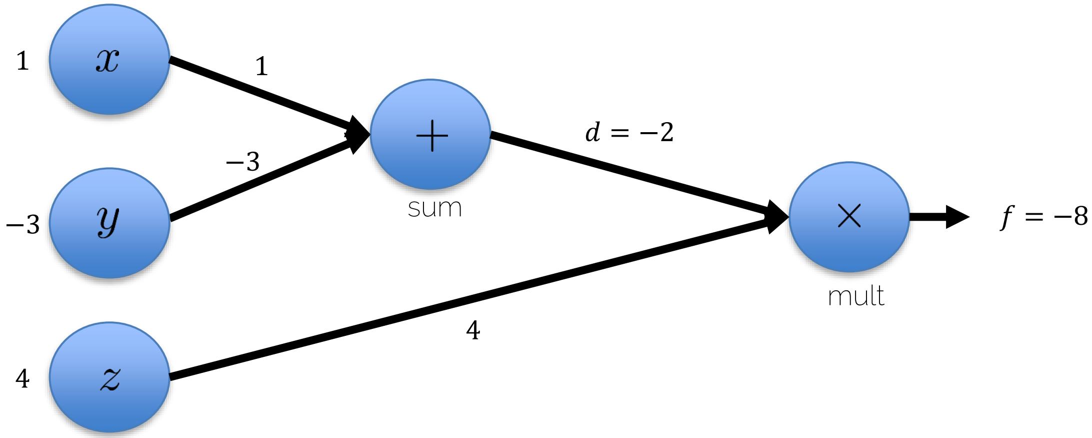
Initialization $x = 1, y = -3, z = 4$



Backprop: Backward Pass

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

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



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

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

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

Backprop: Backward Pass

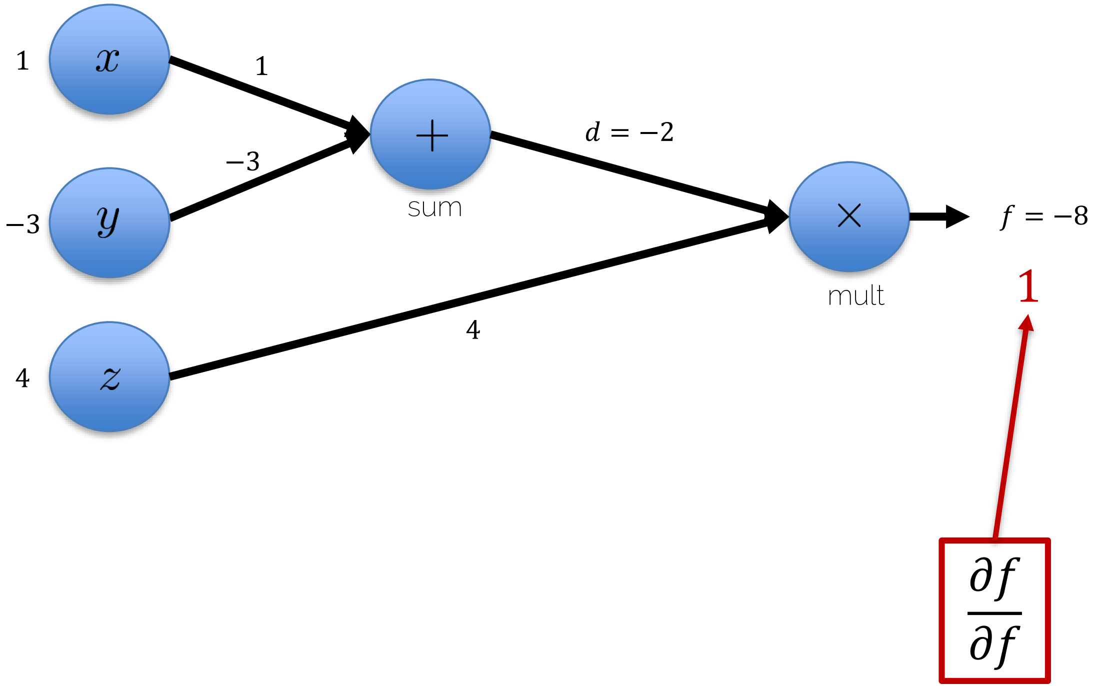
$$f(x, y, z) = (x + y) \cdot z$$

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

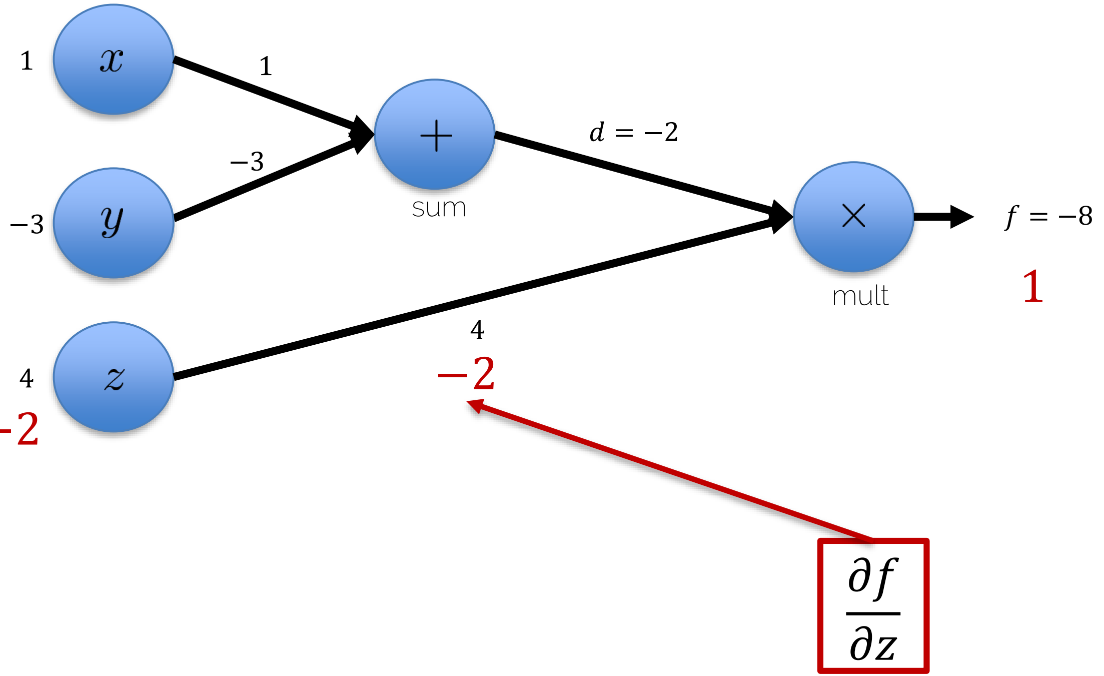
$$f(x, y, z) = (x + y) \cdot z$$

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

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

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

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



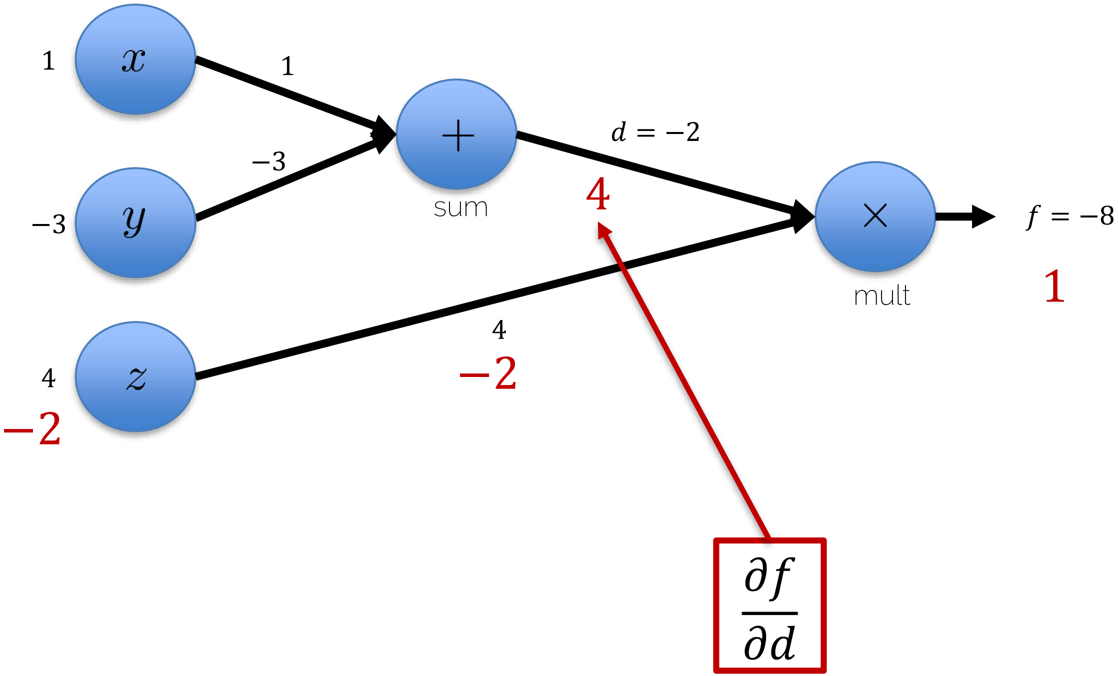
Backprop: Backward Pass

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

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

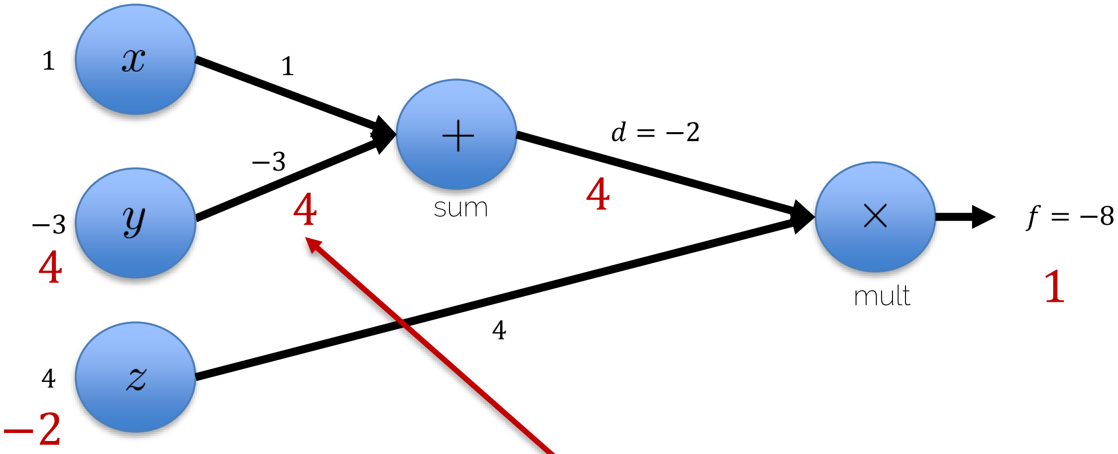
$$f(x, y, z) = (x + y) \cdot z$$

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

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

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

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

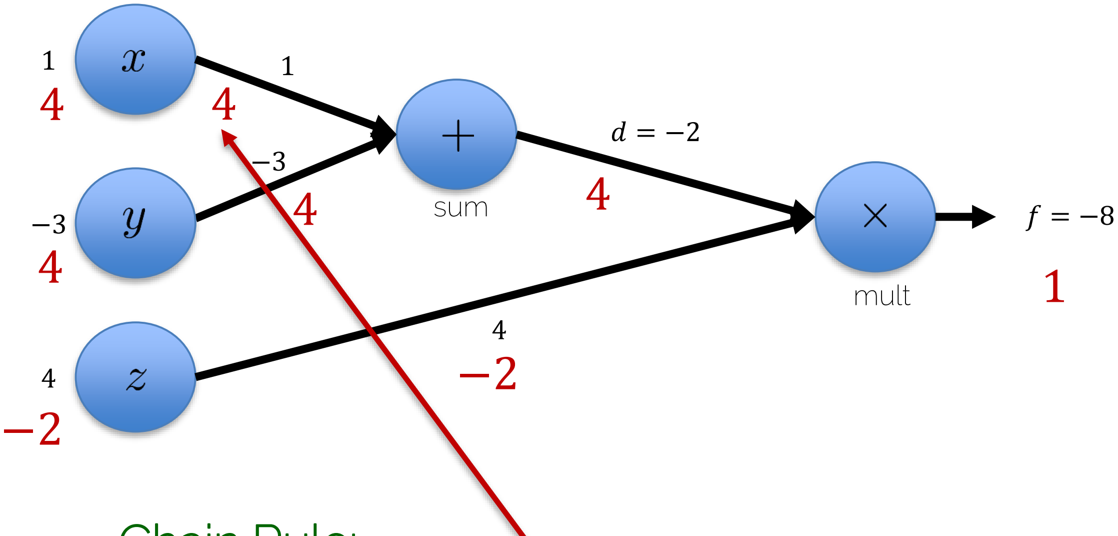
$$f(x, y, z) = (x + y) \cdot z$$

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

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

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

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



Chain Rule:

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

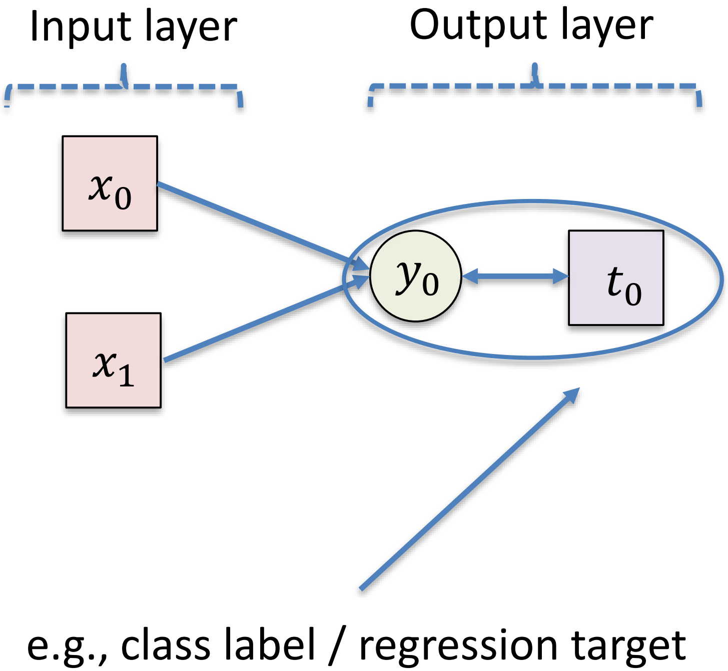
$$\boxed{\frac{\partial f}{\partial x}}$$

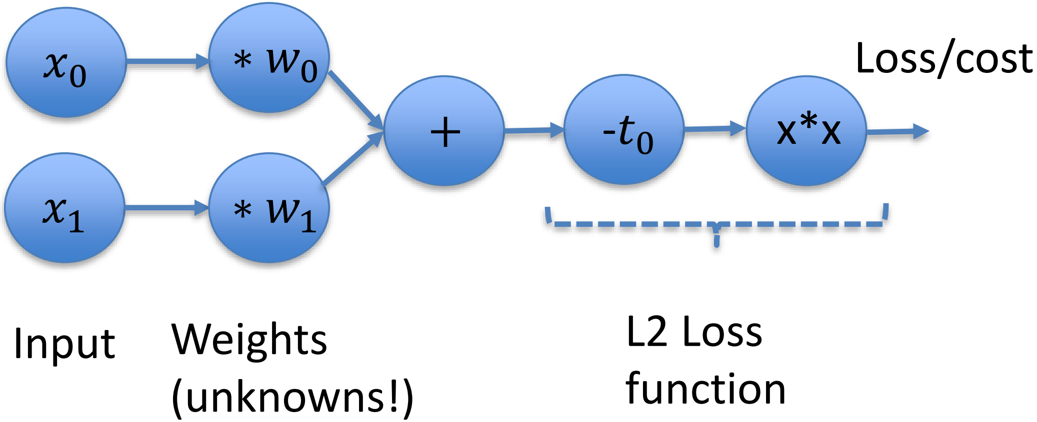
$$\rightarrow \frac{\partial f}{\partial x} = 4 \cdot 1 = 4$$

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 weights in neuron
- y_i computed output (i output dim; nout)
- t_i ground truth targets
- L is loss function

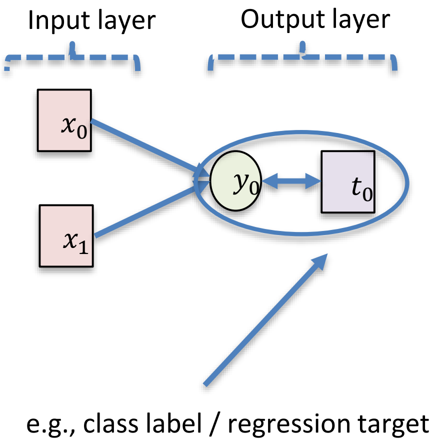
Compute Graphs -> Neural Networks

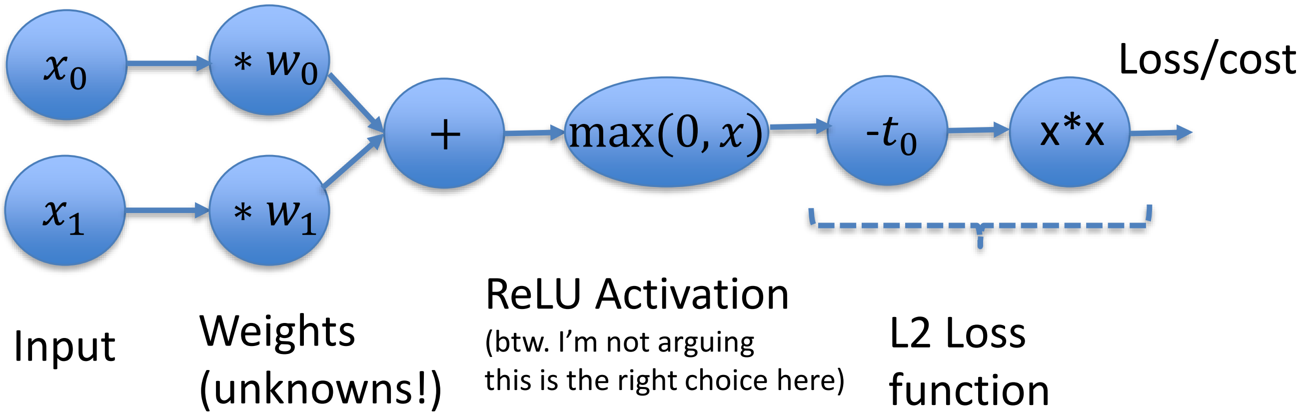




We want to compute gradients w.r.t. all weights w

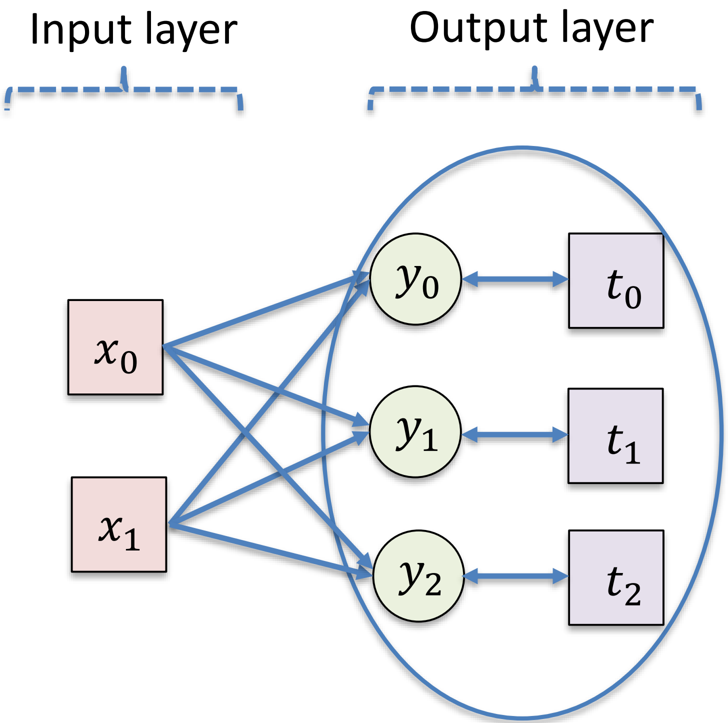
Compute Graphs -> Neural Networks

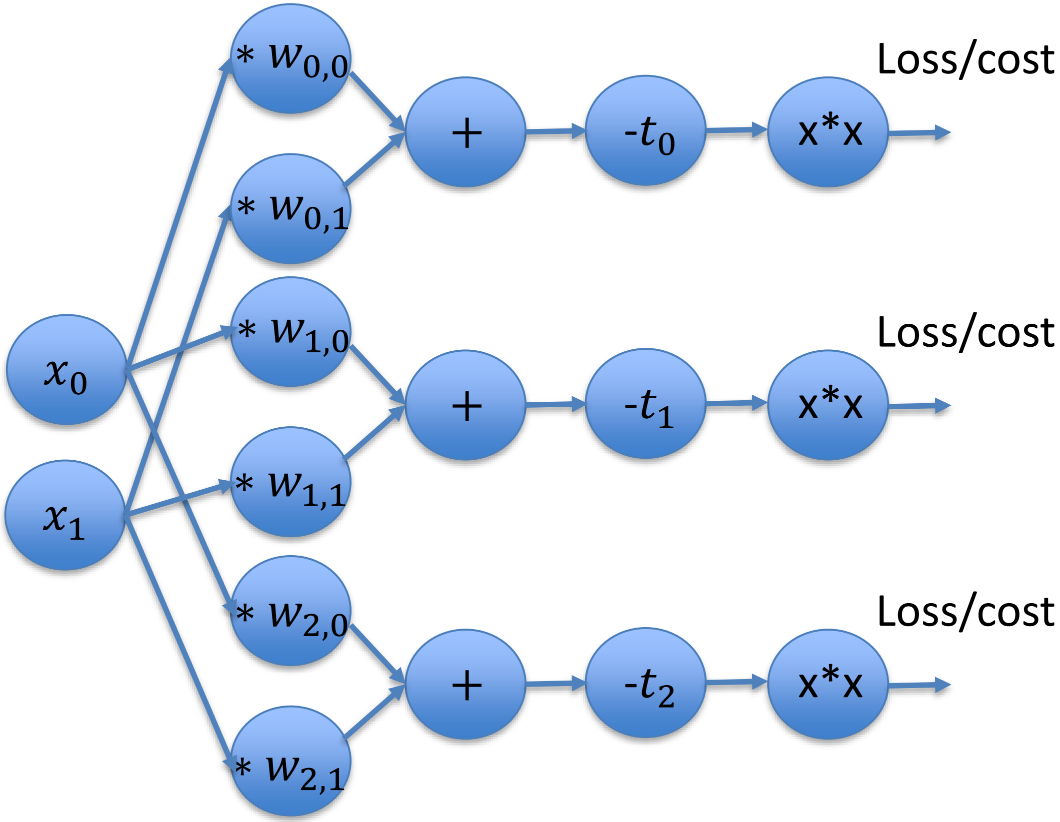




We want to compute gradients w.r.t. all weights w

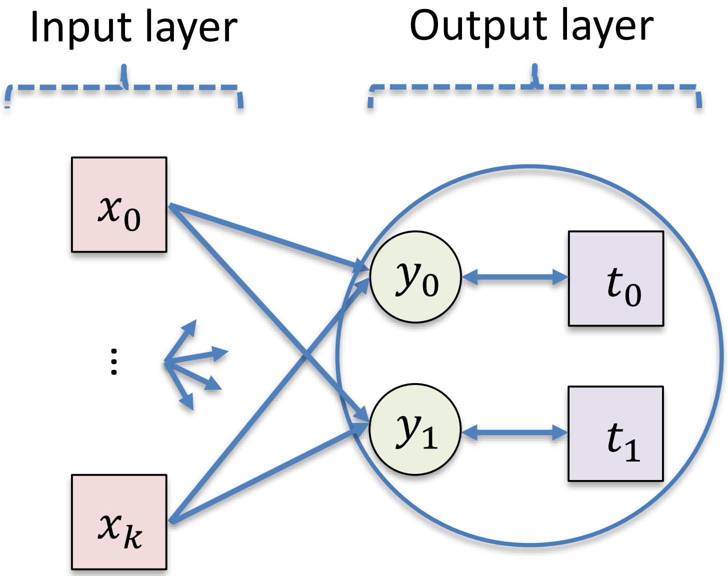
Compute Graphs -> Neural Networks





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

Compute Graphs -> Neural Networks



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

Activation function

bias

$$L_i = (y_i - t_i)^2$$

$$L = \sum_i L_i$$

L2 loss -> simply sum up squares

Energy to minimize is E=L

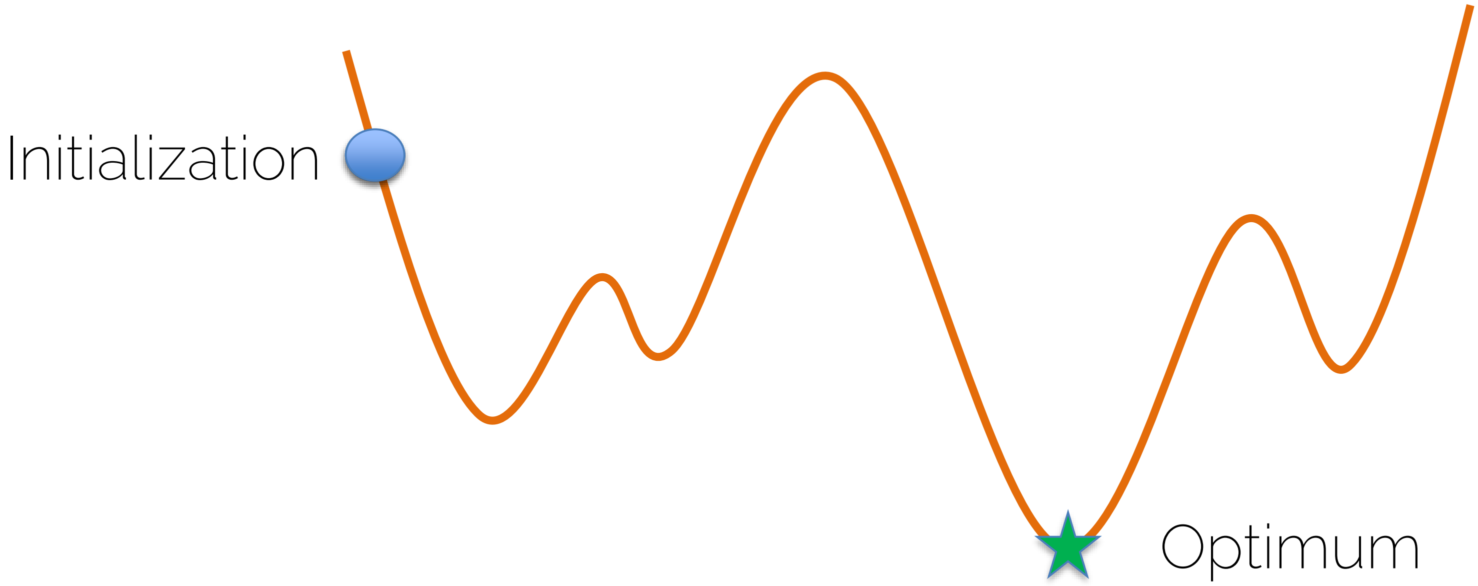
$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_{i,k}}$$

-> use chain rule to compute partials

We want to compute gradients w.r.t. all weights w *AND* biases b

Gradient Descent

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



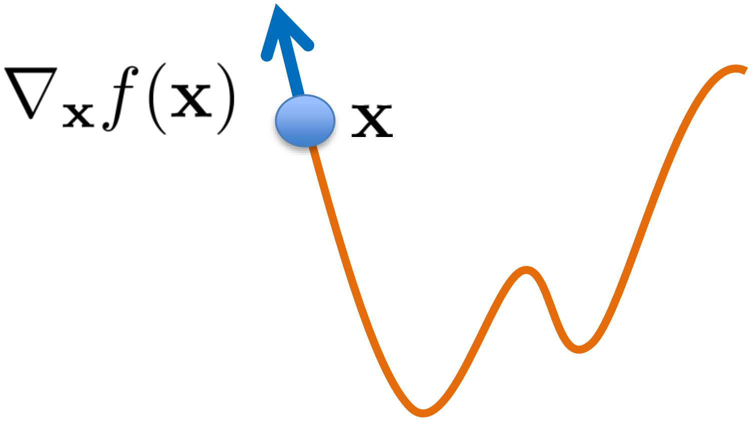
Gradient Descent

- From derivative to gradient

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

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



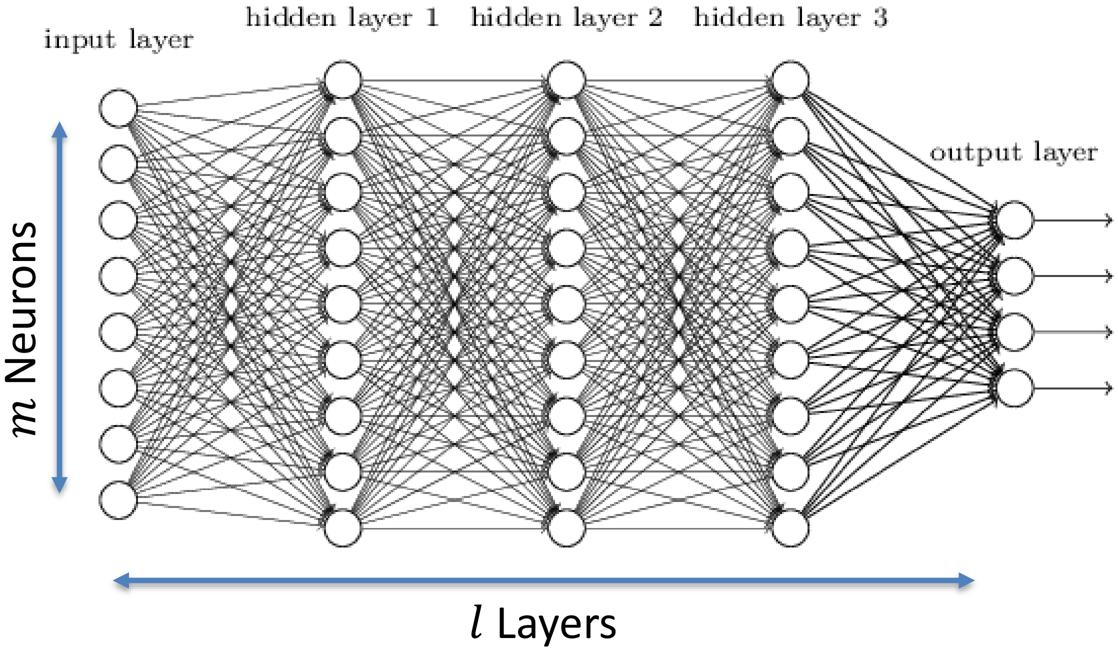
$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

Learning rate

Gradient Descent for Neural Networks

For a given training pair {x,t},
we want to update all weights;
i.e., we need to compute derivatives
w.r.t. to all weights

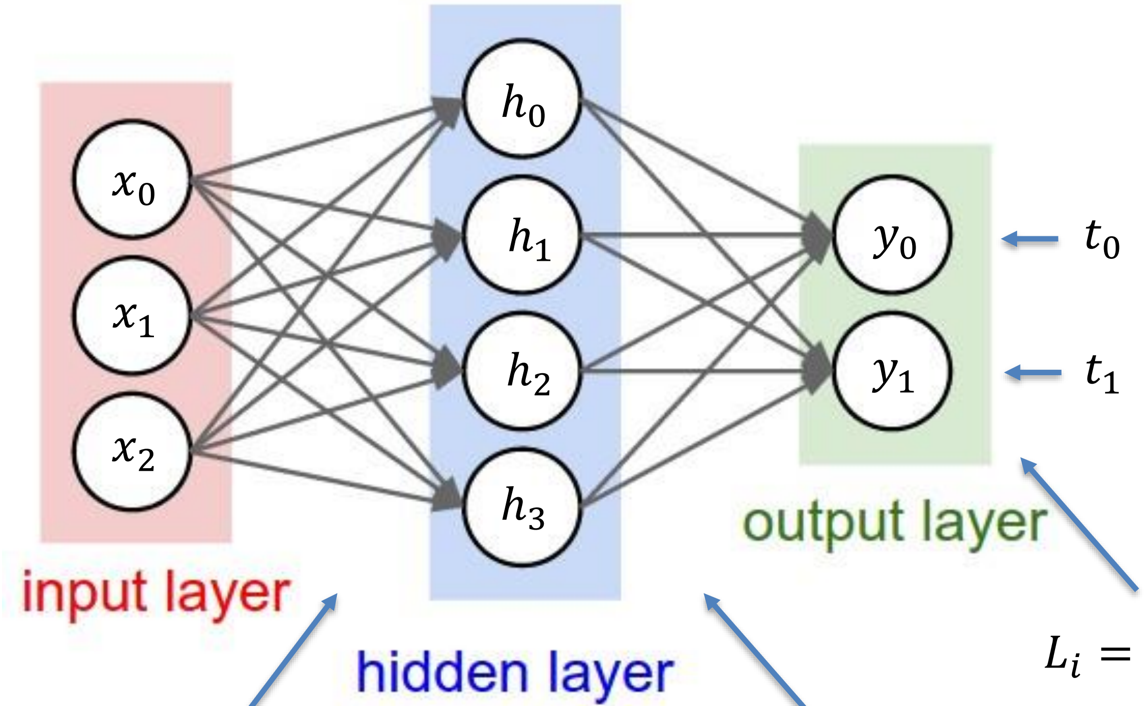
$$\nabla_w f_{\{x,t\}}(w) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \dots \\ \dots \\ \frac{\partial f}{\partial w_{l,m,n}} \end{bmatrix}$$



Gradient step:

$$w' = w - \epsilon \nabla_w f_{\{x,t\}}(w)$$

Gradient Descent for Neural Networks



$$\nabla_w f_{\{x,t\}}(w) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \dots \\ \dots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \dots \\ \dots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

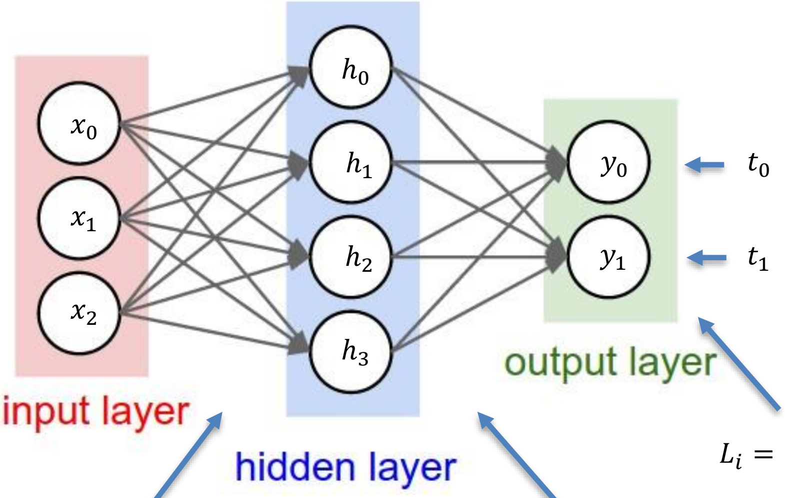
$$L_i = (y_i - t_i)^2$$

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

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

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

Gradient Descent for Neural Networks



$$L_i = (y_i - t_i)^2$$

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

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

Just go through layer by layer

Backpropagation

$$\frac{\partial L}{\partial w_{1,i,j}} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_{1,i,j}}$$

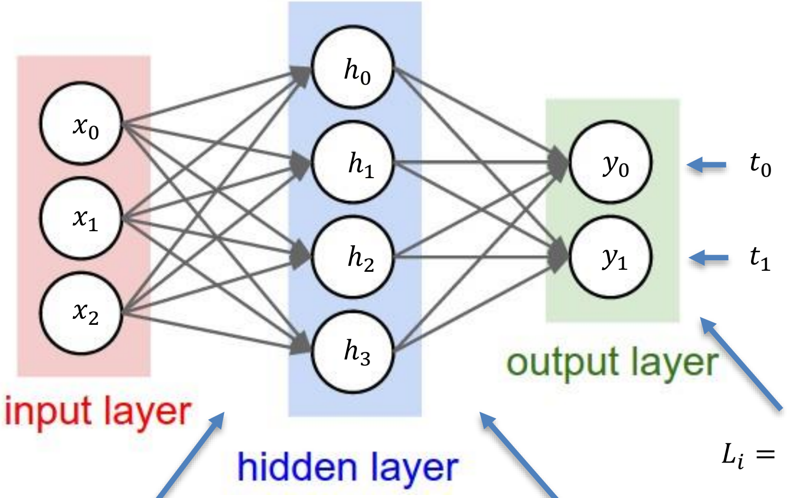
$$\frac{\partial L}{\partial w_{0,j,k}} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{0,j,k}}$$

$$\frac{\partial L_i}{\partial y_i} = 2(\mathrm{y_i} - \mathrm{t_i})$$

…

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

Gradient Descent for Neural Networks



$$L_i = (y_i - t_i)^2$$

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

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

How many unknown weights?

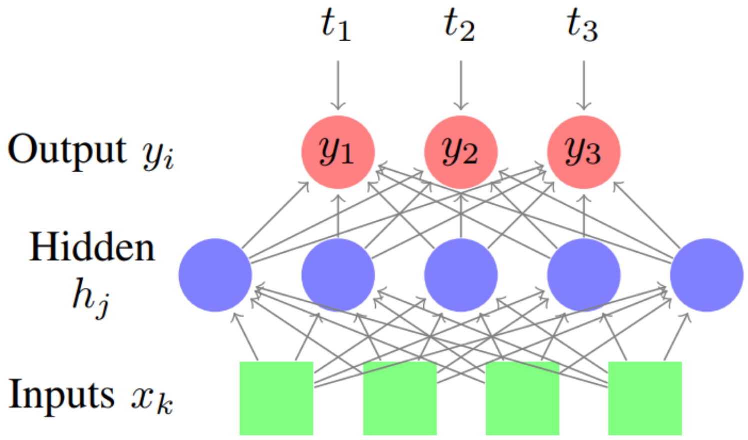
Output layer: $2 \cdot 4 + 2$

#neurons #input channels #biases

Hidden Layer: $4 \cdot 3 + 4$

Note that some activations have also weights

Derivatives of Cross Entropy Loss



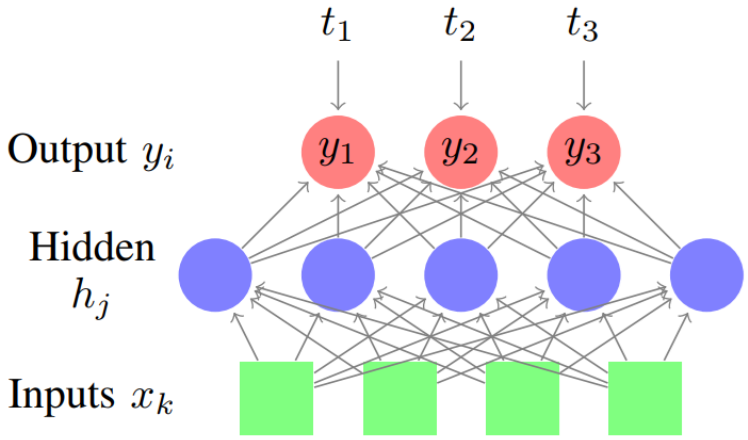
$$E = -\sum_{i=1}^{nout} \left(t_i \log(y_i) + (1 - t_i)\log(1 - y_i)\right)$$

$$y_i = \frac{1}{1 + e^{-s_i}}$$

$$s_i = \sum_{j=1} h_j w_{ji}$$

[Sadowski]

Derivatives of Cross Entropy Loss



$$E = -\sum_{i=1}^{nout} \left(t_i \log(y_i) + (1 - t_i) \log(1 - y_i) \right)$$

$$y_i = \frac{1}{1 + e^{-s_i}}$$

$$s_i = \sum_{j=1} h_j w_{ji}$$

Gradients of weights of last layer:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}}$$

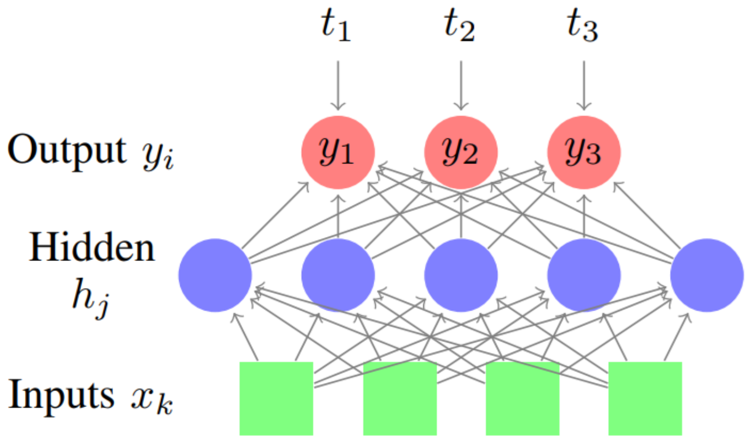
$$\begin{aligned} \frac{\partial E}{\partial y_i} &= \frac{-t_i}{y_i} + \frac{1 - t_i}{1 - y_i}, \\ &= \frac{y_i - t_i}{y_i(1 - y_i)}, \\ \frac{\partial y_i}{\partial s_i} &= y_i(1 - y_i) \\ \frac{\partial s_i}{\partial w_{ji}} &= h_j \end{aligned}$$

$$\frac{\partial E}{\partial s_i} = y_i - t_i$$

$$\frac{\partial E}{\partial w_{ji}} = (y_i - t_i) h_j$$

[Sadowski]

Derivatives of Cross Entropy Loss



$$E = -\sum_{i=1}^{nout} (t_i \log(y_i) + (1 - t_i) \log(1 - y_i))$$

$$y_i = \frac{1}{1 + e^{-s_i}}$$

$$s_i = \sum_{j=1} h_j w_{ji}$$

Gradients of weights of first layer:

$$\frac{\partial E}{\partial s_j^1} = \sum_{i=1}^{nout} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j^1}$$

$$= \sum_{i=1}^{nout} (y_i - t_i)(w_{ji})(h_j(1 - h_j))$$

$$\frac{\partial E}{\partial h_j} = \sum_{i=1} \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{h_j}$$

$$= \sum_{i} \frac{\partial E}{\partial y_i} y_i(1 - y_i) w_{ji}$$

$$\frac{\partial E}{\partial w_{kj}^1} = \frac{\partial E}{\partial s_j^1} \frac{\partial s_j^1}{\partial w_{kj}^1}$$

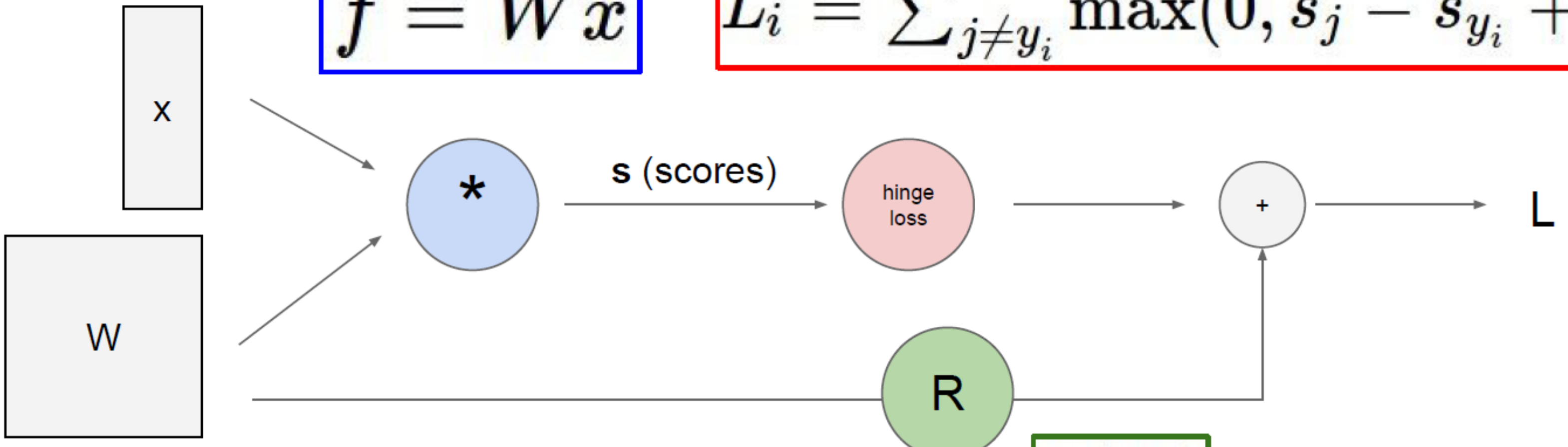
$$= \sum_{i=1}^{nout} (y_i - t_i)(w_{ji})(h_j(1 - h_j))(x_k)$$

[Sadowski]

Derivatives of Neural Networks

$f = Wx$

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

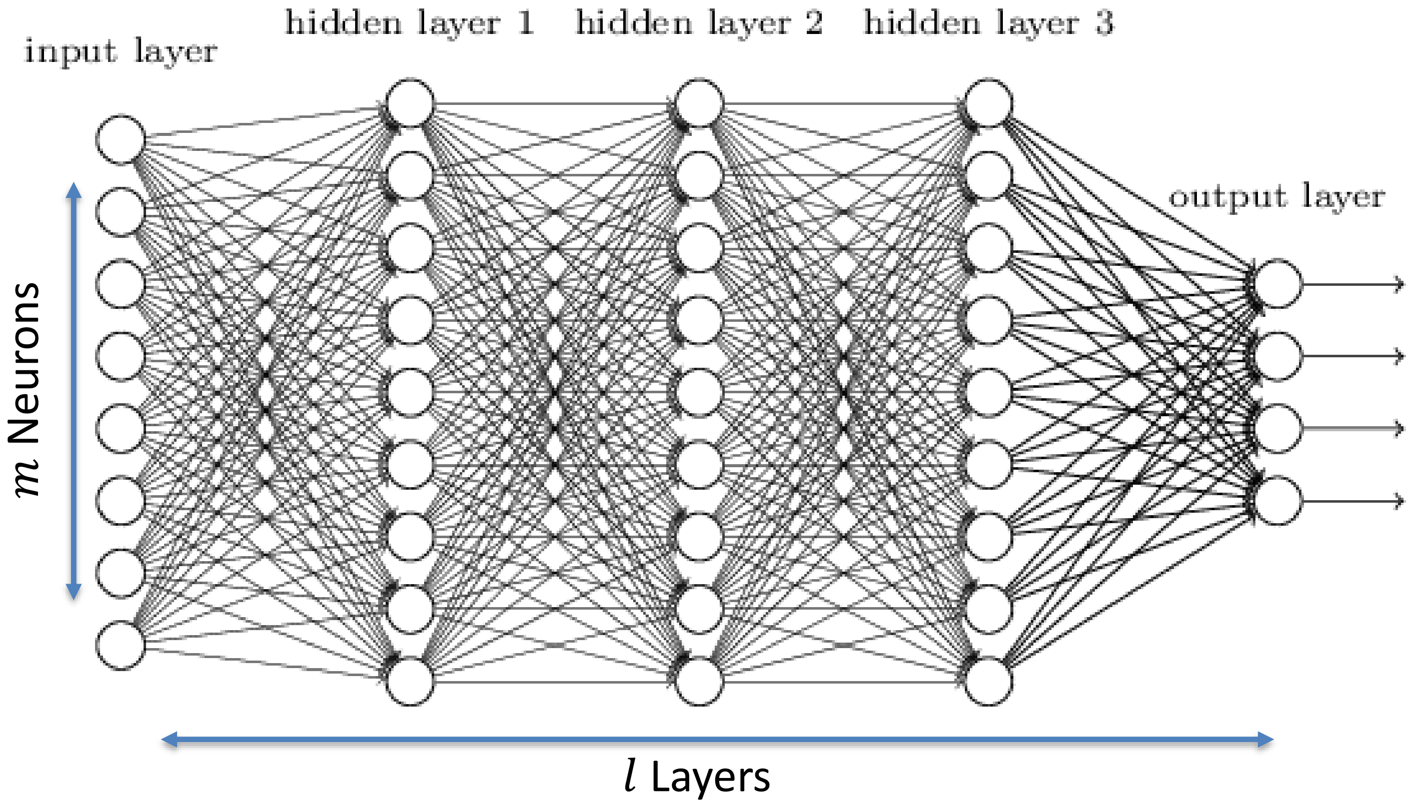


$R(W)$

Combining nodes:
Linear activation node + hinge loss + regularization

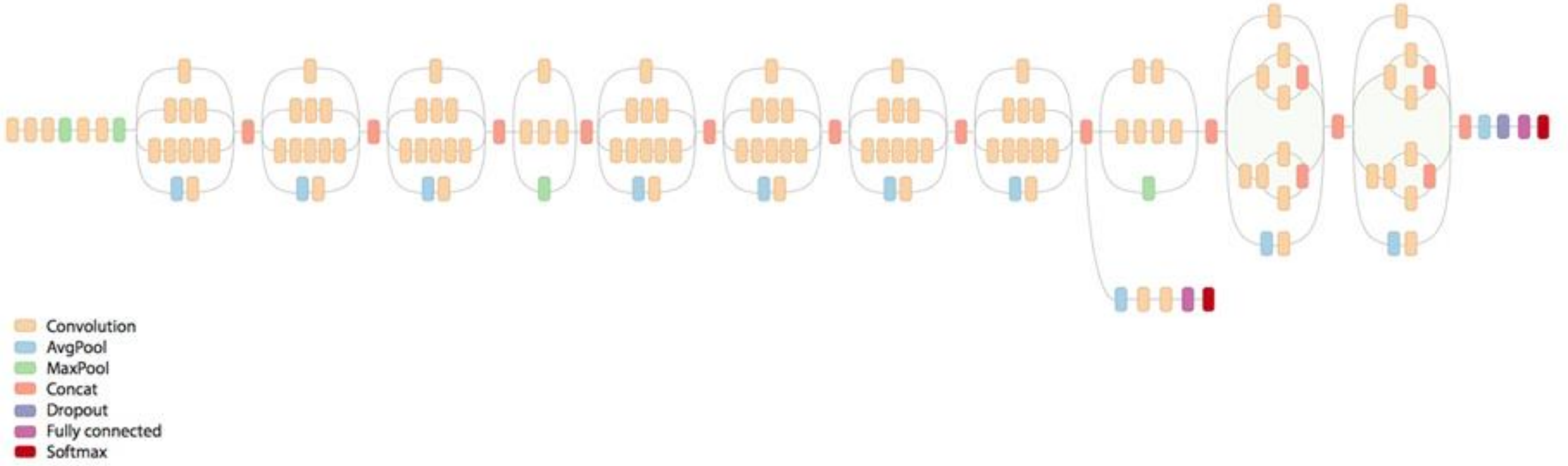
Credit: Li/Karpathy/Johnson

Can become quite complex...



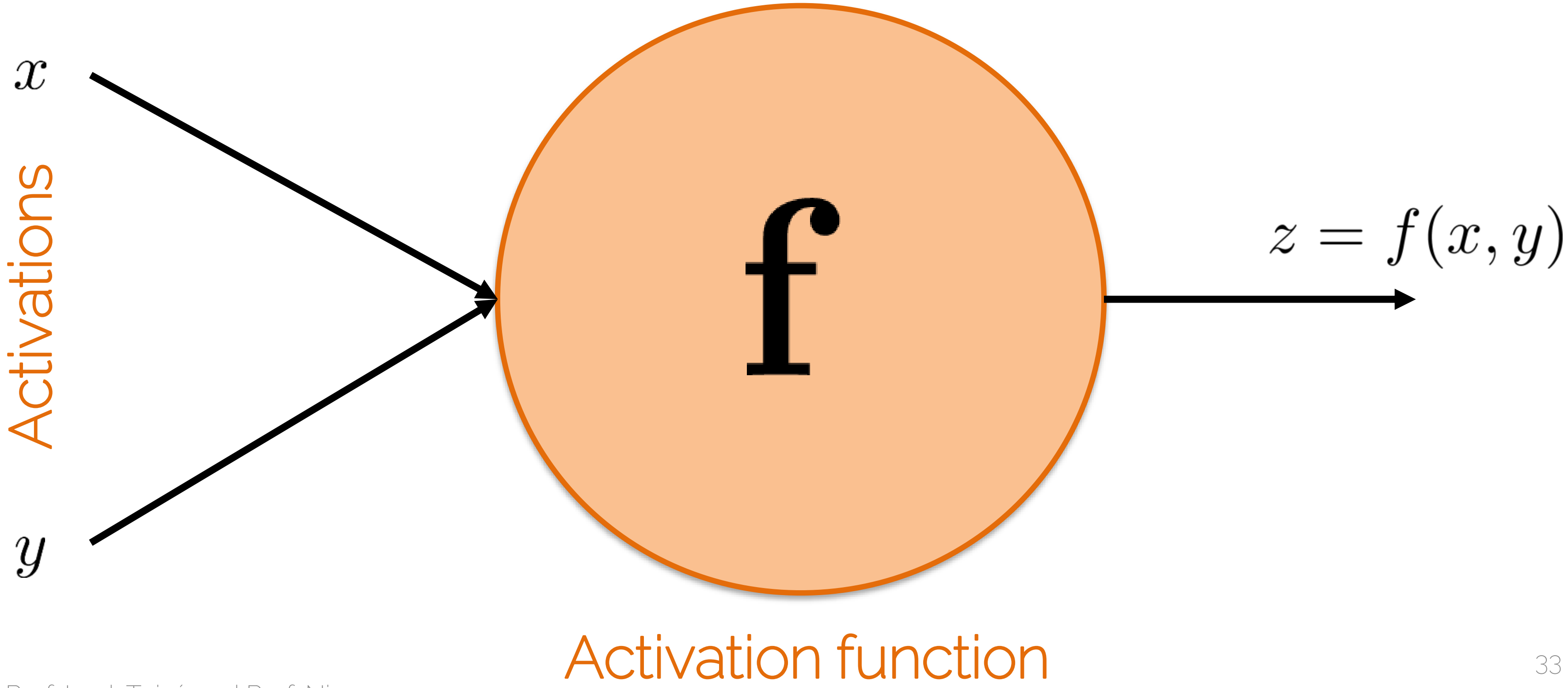
Can become quite complex…

- These graphs can be huge!

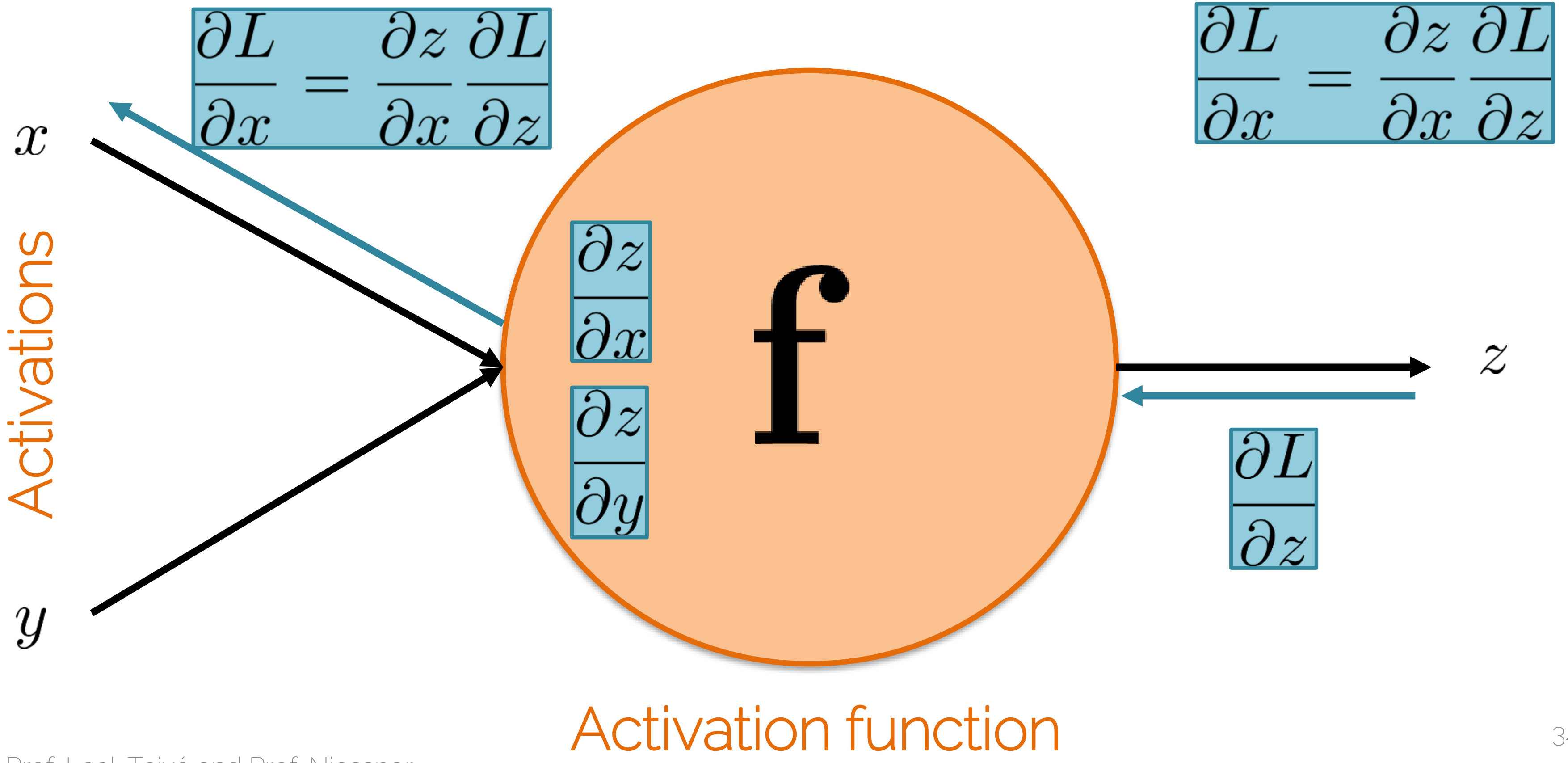


Another view of GoogLeNet's architecture.

The flow of the gradients



The flow of the gradients



The flow of the gradients

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

NEURONS

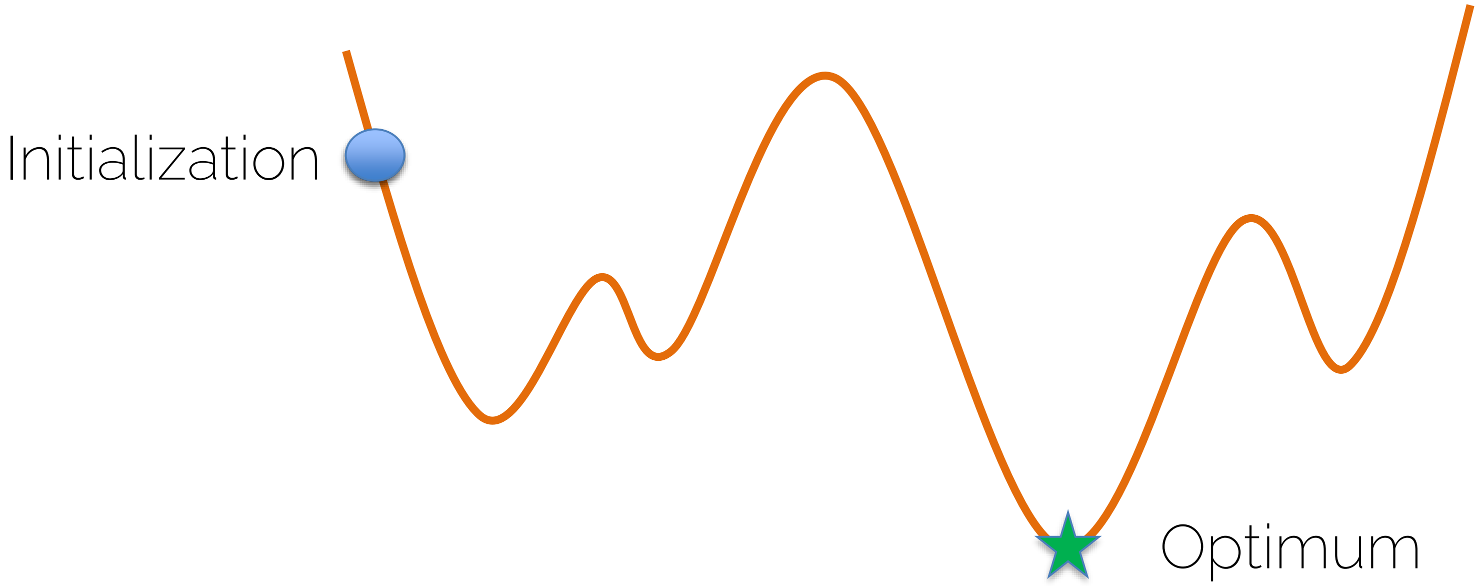
- Each one has its own work to do

FORWARD AND BACKWARD PASS

Gradient descent

Gradient Descent

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



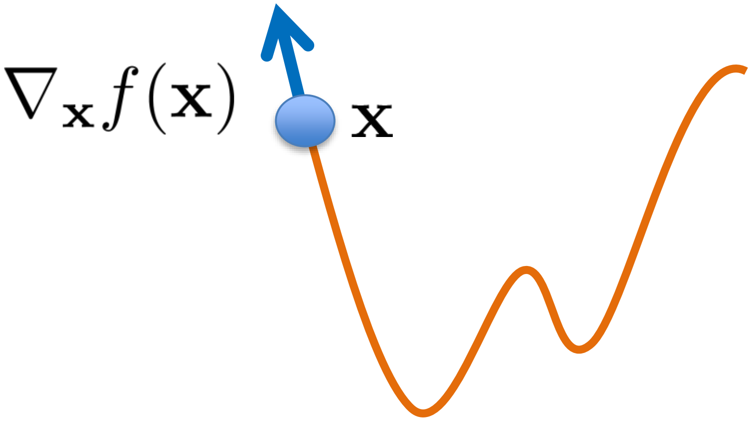
Gradient Descent

- From derivative to gradient

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

Direction of greatest increase of the function

- Gradient steps in direction of negative gradient



$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

Learning rate

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 ☺

Next lecture

- This week:
  - **No tutorial this Thursday** (due to MaiTUM)
  - Exercise 1 will be released on Thursday as planned

- Next lecture on May 7th:
  - Optimization of Neural Networks
  - In particular, introduction to SGD (our main method!)

See you next week!