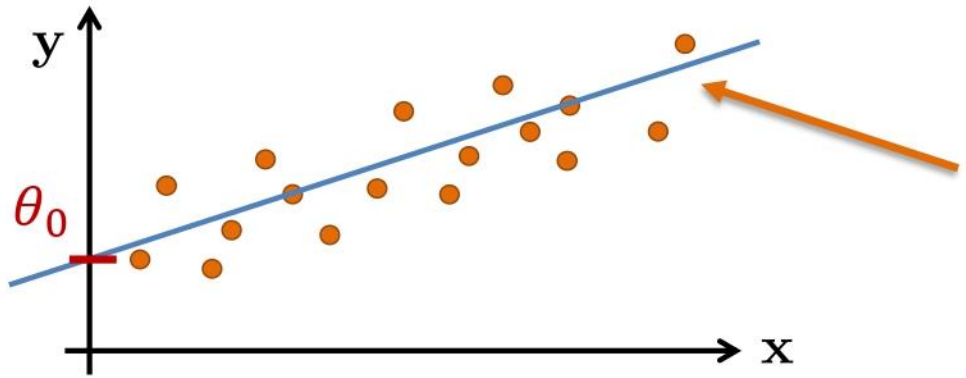
Lecture 5
Introduction to Neural Networks

Linear Regression

= a **supervised learning** method to find a **linear model** of the form

$$\hat{y}_i = \theta_0 + \sum_{j=1}^{d} x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$$



Goal: find a model that explains a target y given the input x

Logistic Regression

- Loss function

$$\mathcal{L}(y_i, \hat{y}_i) = -y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

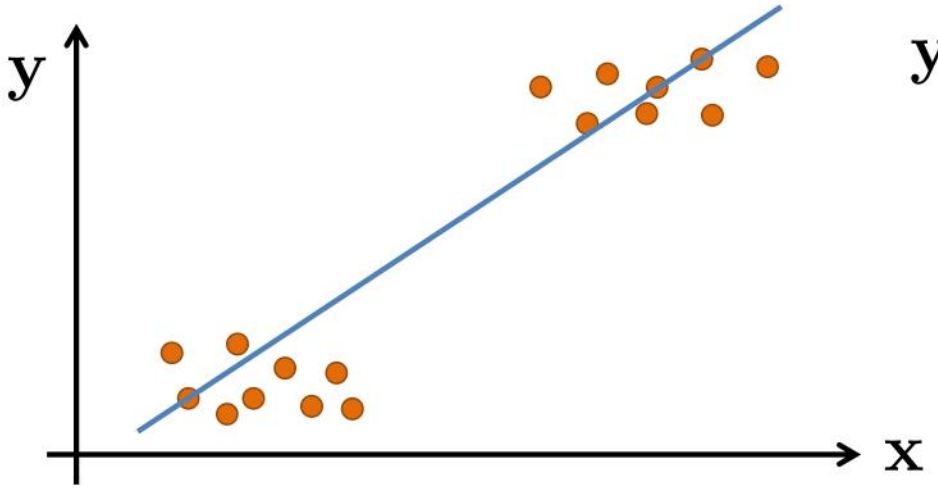
- Cost function

$$\mathcal{C}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

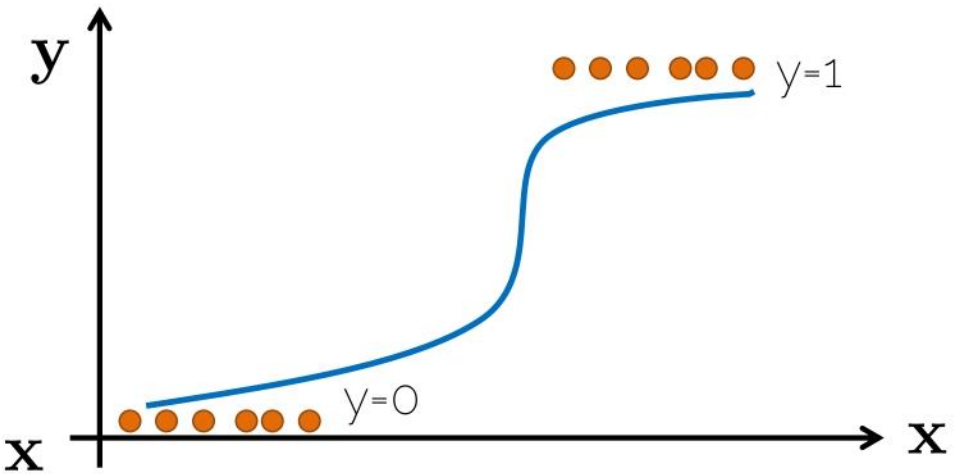
Minimization

$\hat{y}_i = \sigma(x_i \boldsymbol{\theta})$

Linear vs Logistic Regression

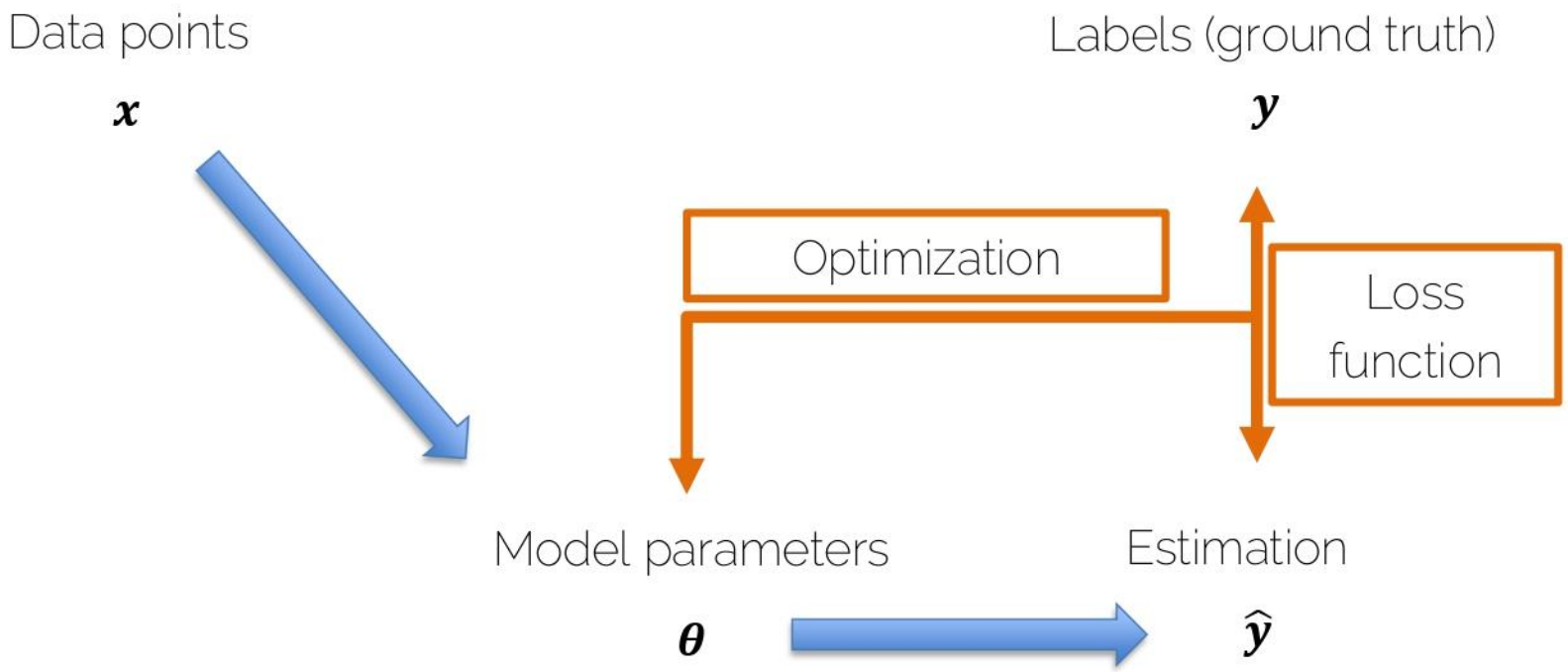


Predictions can exceed the range of the training samples
→ in the case of classification [0;1] this becomes a real issue



Predictions are guaranteed to be within [0;1]

How to obtain the Model?



Linear Score Functions

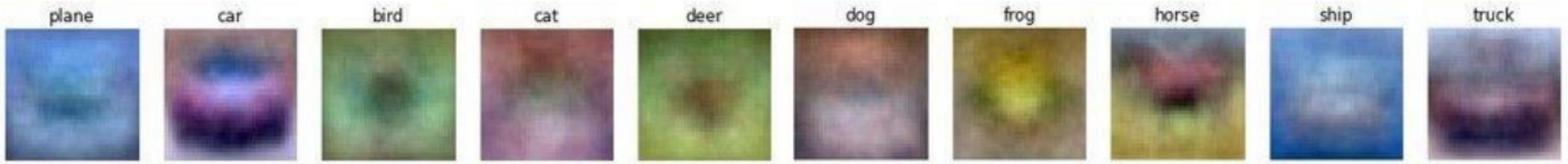
- Linear score function as seen in linear regression

$$\boldsymbol{f_i} = \sum_{\boldsymbol{j}} \boldsymbol{W_{k,j}\, x_{j,i}}$$

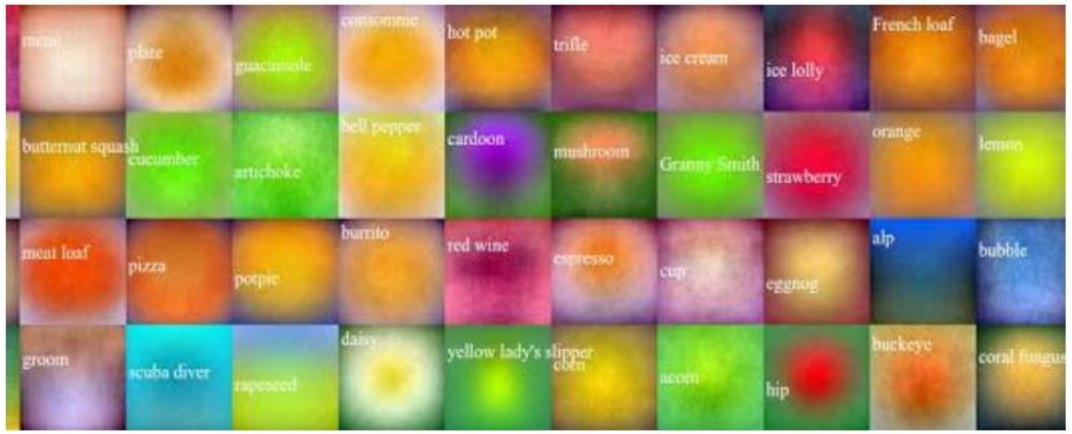
$$\boldsymbol{f = W\, x} \qquad \text{(Matrix Notation)}$$

Linear Score Functions on Images

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



On CIFAR-10

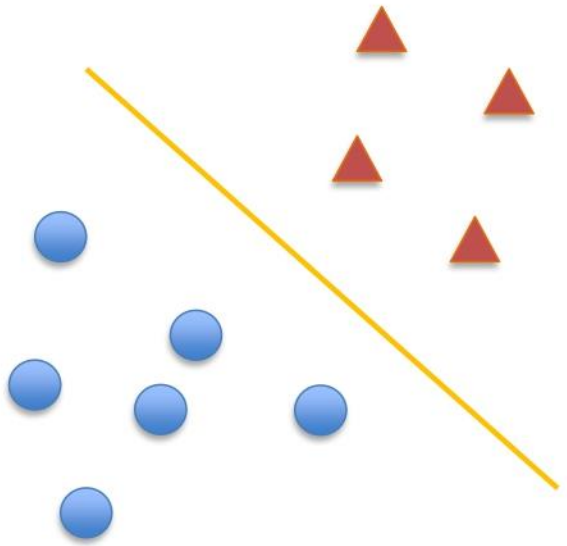


On ImageNet

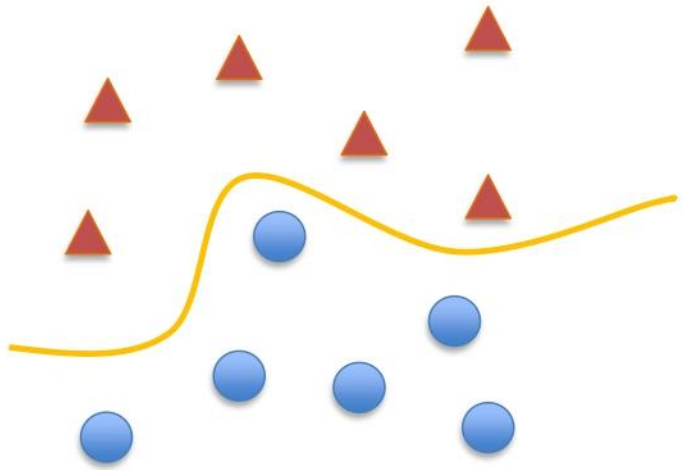
Source:: Li/Karpathy/Johnson

Linear Score Functions?

Logistic Regression



Linear Separation Impossible!



Linear Score Functions?

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

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

- Operation is still linear.

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

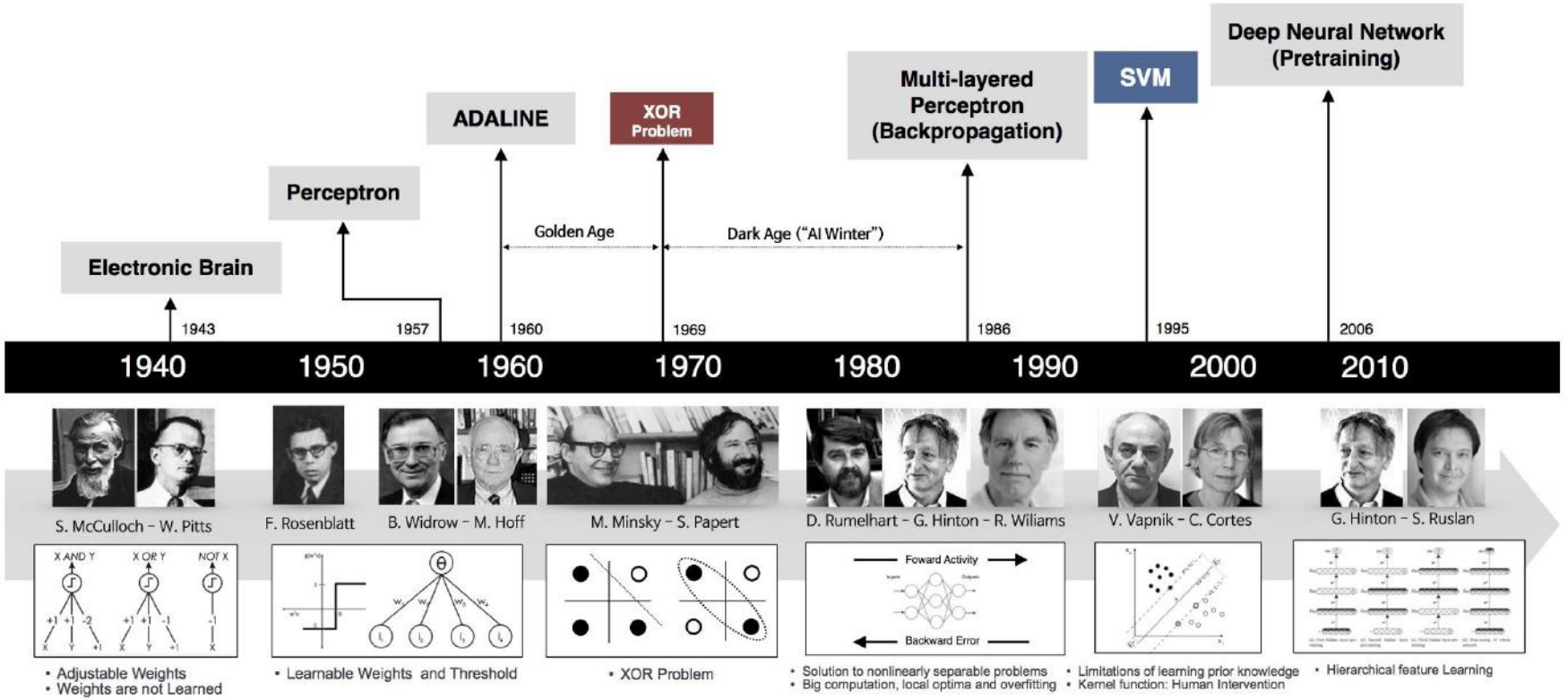
- Solution → add non-linearity!!

Neural Network

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

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

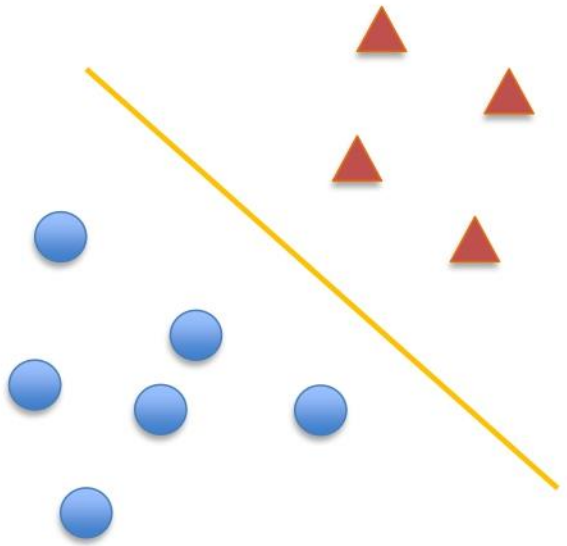
History of Neural Networks



Source: http://beamlab.org/deeplearning/2017/02/23/deep_learning_101_part1.html

Neural Network

Logistic Regression

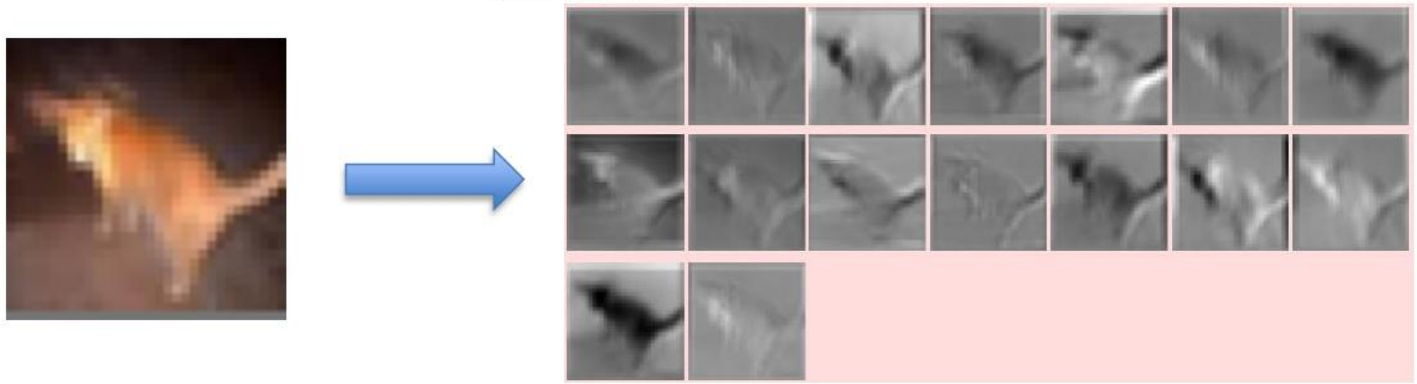


Neural Networks

Neural Network

- Non-linear score function $f = \ldots(\max(0, W_1 x))$

On CIFAR-10

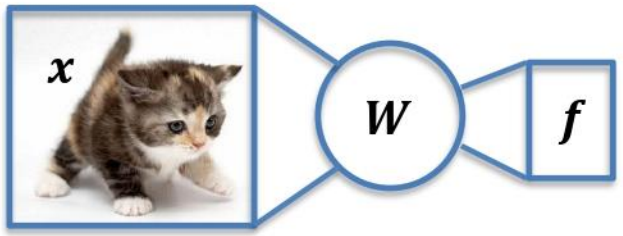


Visualizing activations of first layer.

Source: ConvNetJS

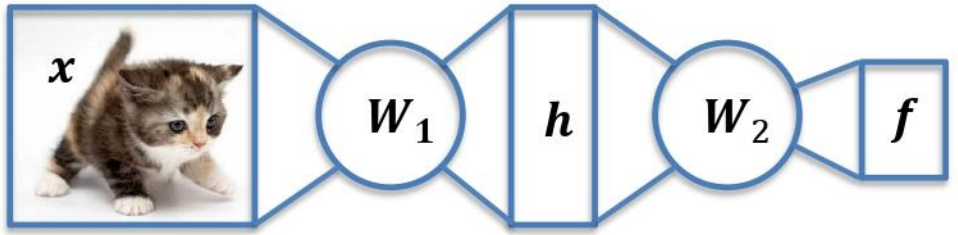
Neural Network

1-layer network: $\boldsymbol{f = Wx}$



$128 \times 128 = 16384$ 10

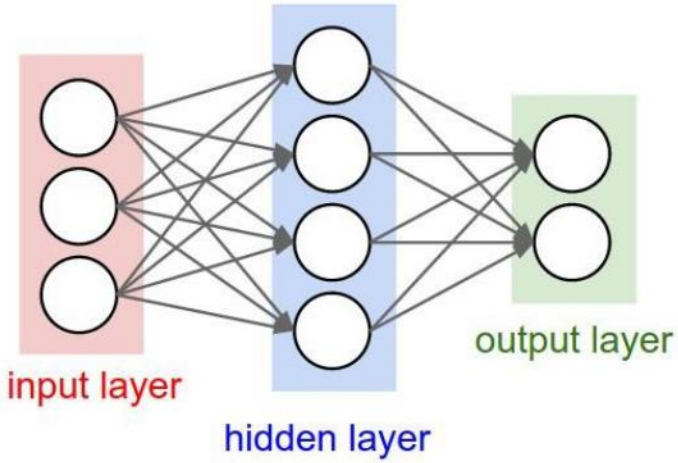
2-layer network: $\boldsymbol{f = W_2 \max(0, W_1 x)}$



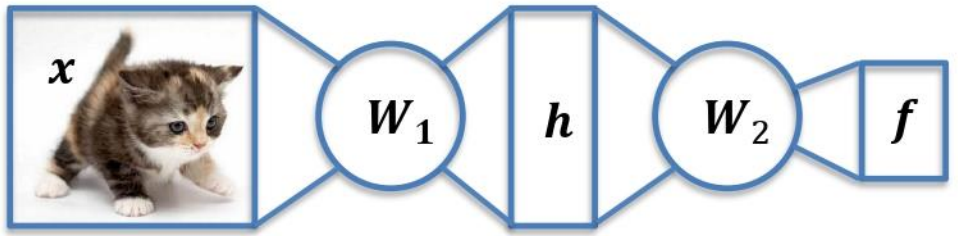
$128 \times 128 = 16384$ 1000 10

Why is this structure useful?

Neural Network



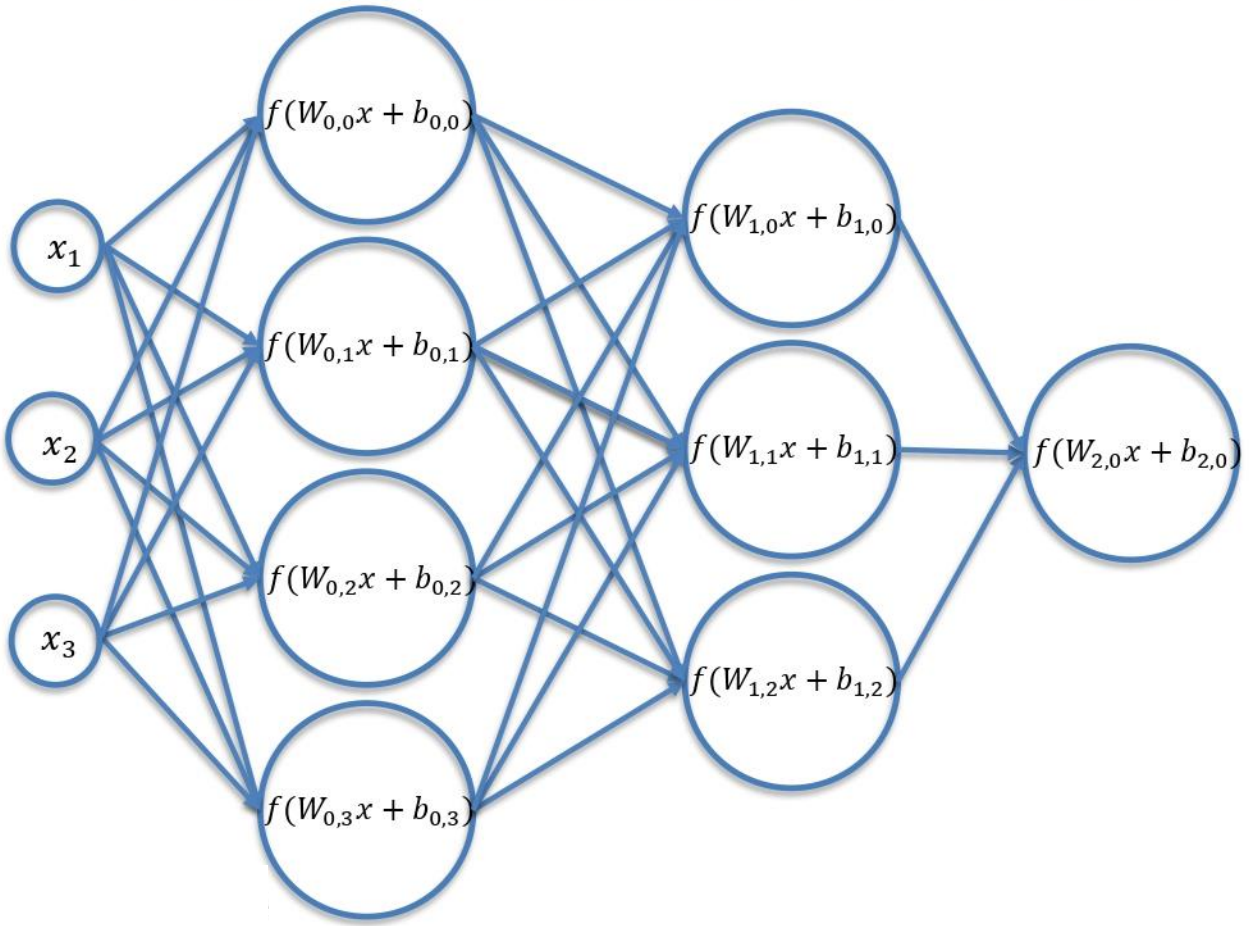
2-layer network: $\boldsymbol{f} = \boldsymbol{W_2}\max(\boldsymbol{0}, \boldsymbol{W_1 x})$



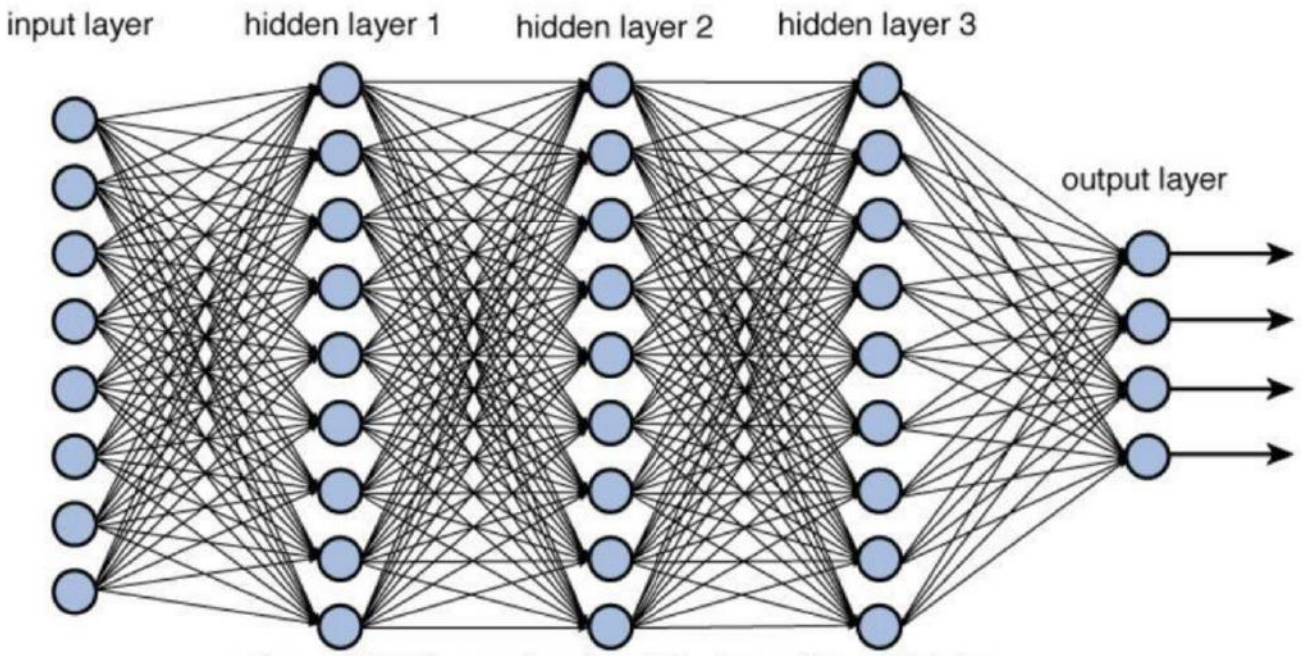
$128 \times 128 = 16384$ $\quad$ 1000 $\quad$ 10

Input Layer $\quad$ Hidden Layer $\quad$ Output Layer

Net of Artificial Neurons



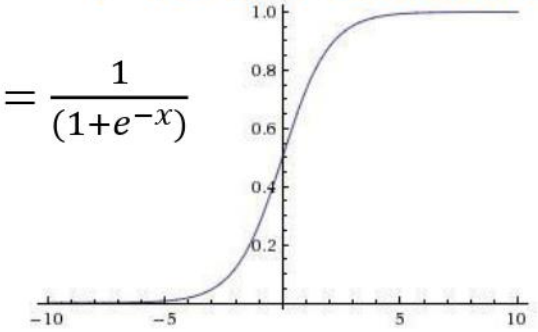
Neural Network



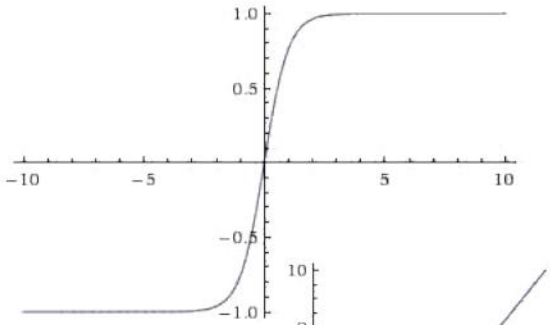
Source: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Activation Functions

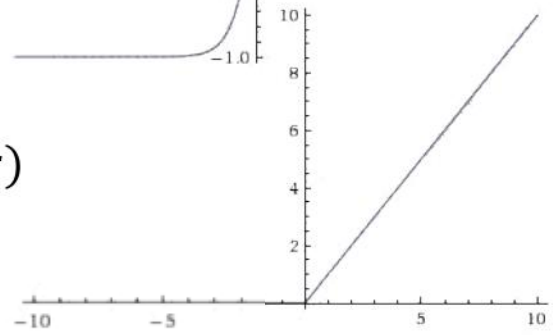
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



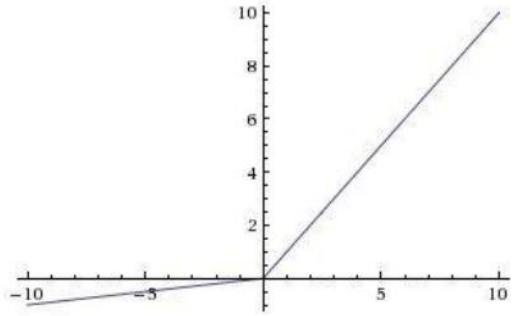
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



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



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

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

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

Neural Network

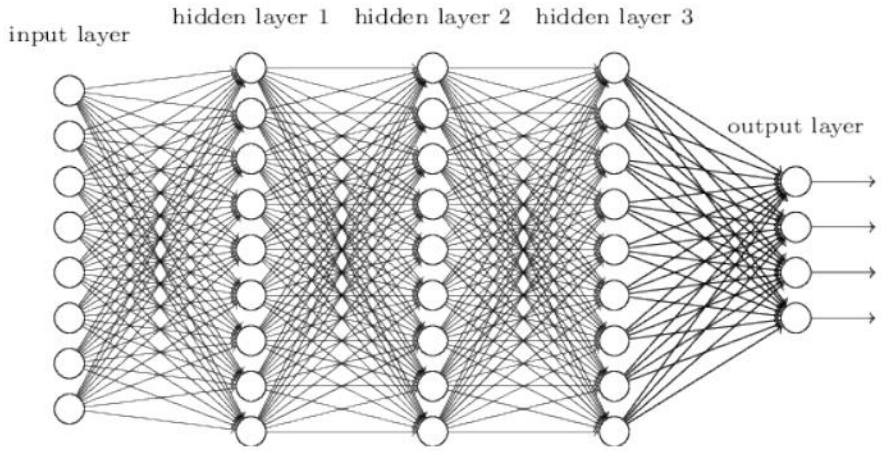
$$f = W_3 \cdot (W_2 \cdot (W_1 \cdot x)))$$

Why activation functions?

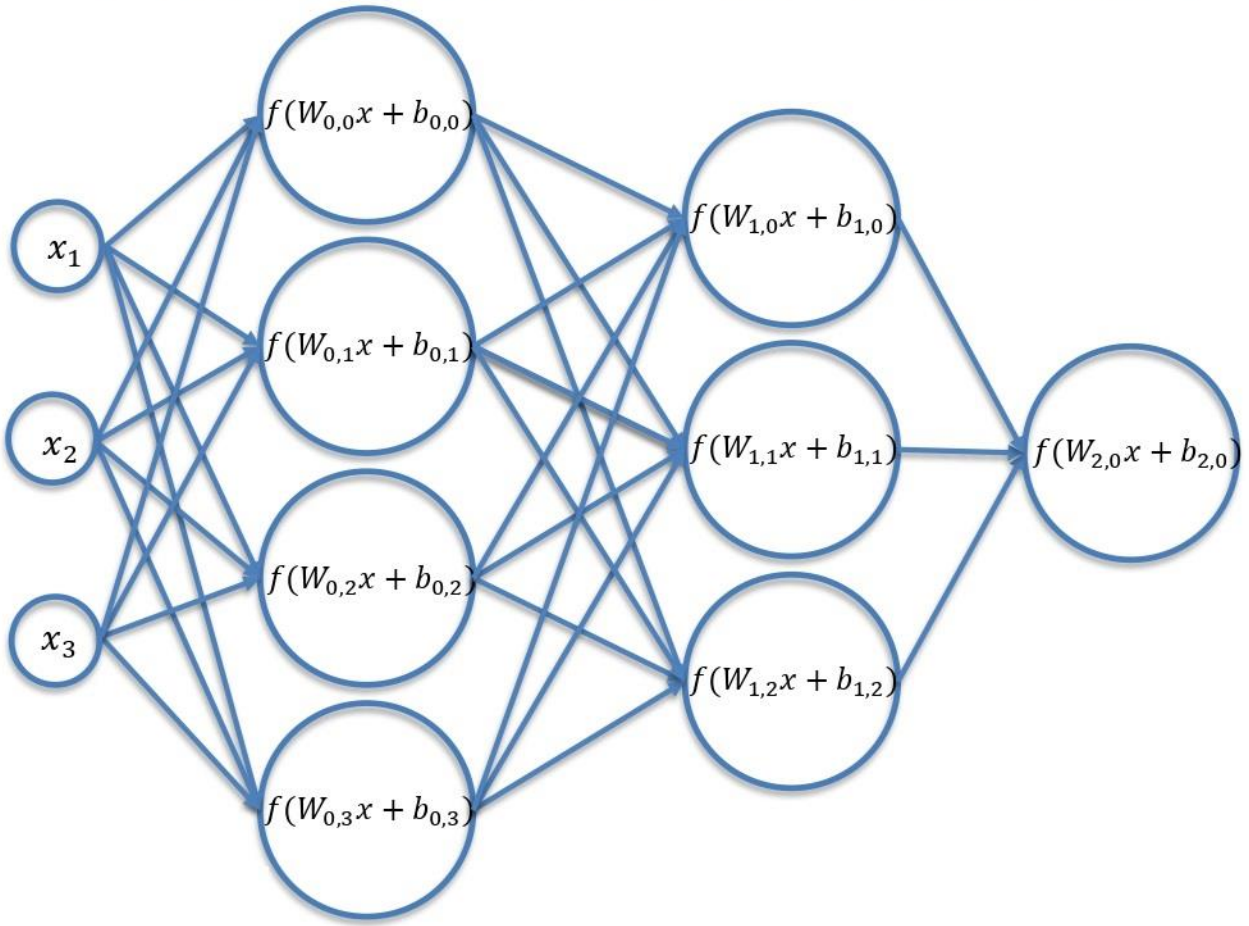
Simply concatenating linear layers would be so much cheaper...

Neural Network

Why organize a neural network into layers?



Artificial Neural Network



Neural Network

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

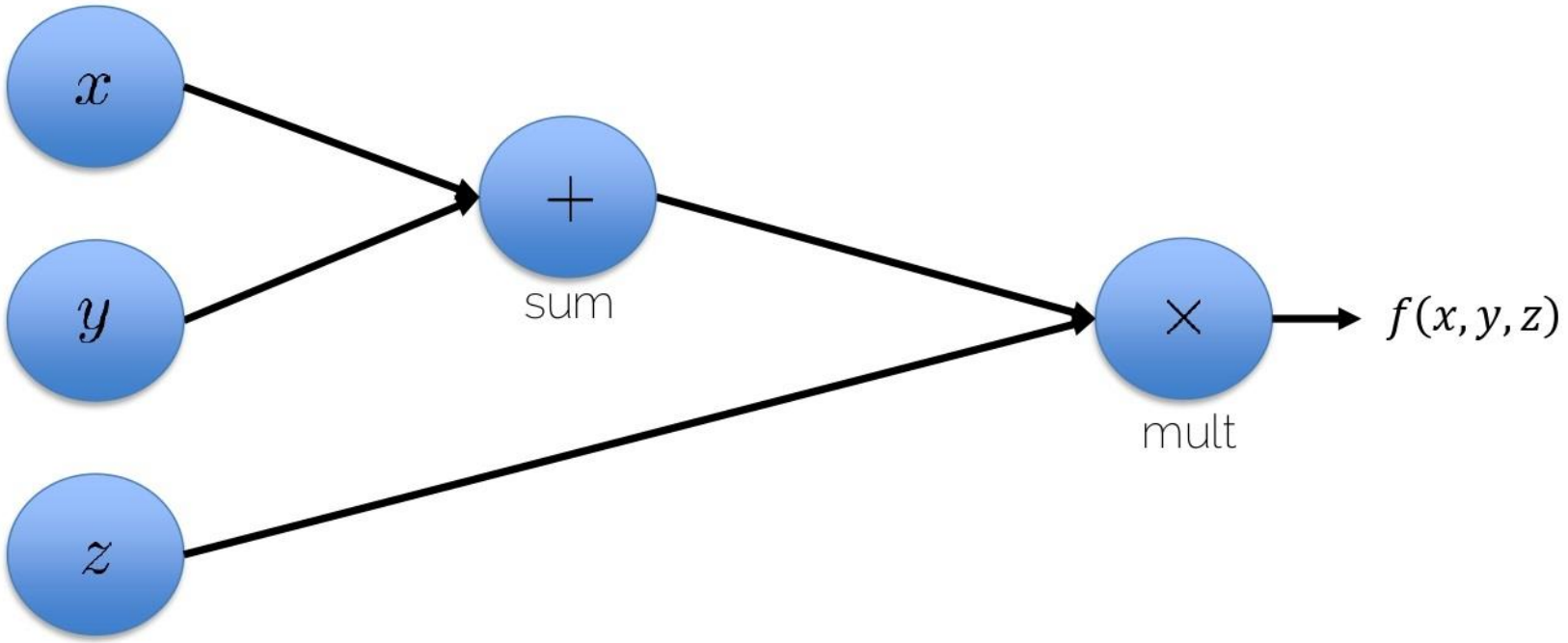
Computational Graphs

Computational Graphs

- Directional graph
- Matrix operations are represented as compute nodes.
- Vertex nodes are variables or operators like +, -, *, /, log(), exp() …
- Directional edges show flow of inputs to vertices

Computational Graphs

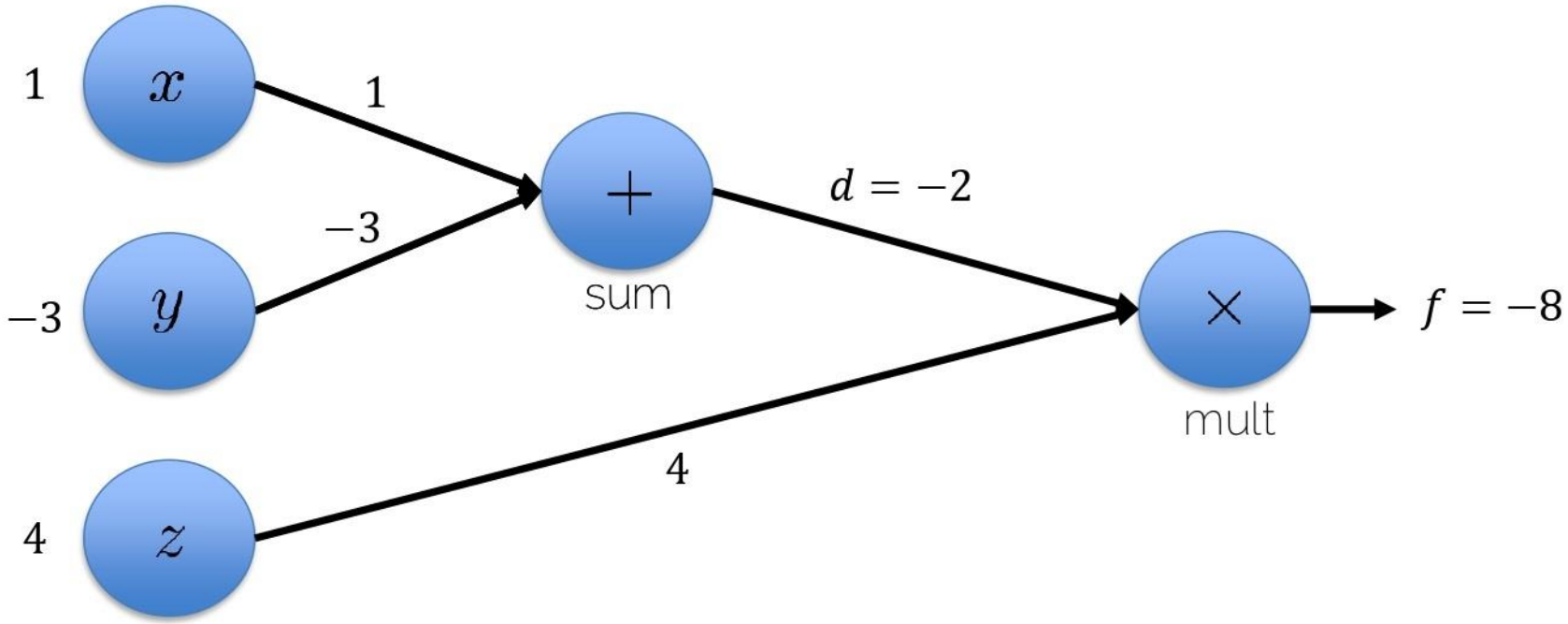
- $f(x, y, z) = (x + y) \cdot z$



Evaluation: Forward Pass

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

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



Computational Graphs

- Why discuss compute graphs?
- Neural networks have complicated architectures

$$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3},\max(\boldsymbol{0},\boldsymbol{W_2}\max(\boldsymbol{0},\boldsymbol{W_1}\boldsymbol{x}))))$$

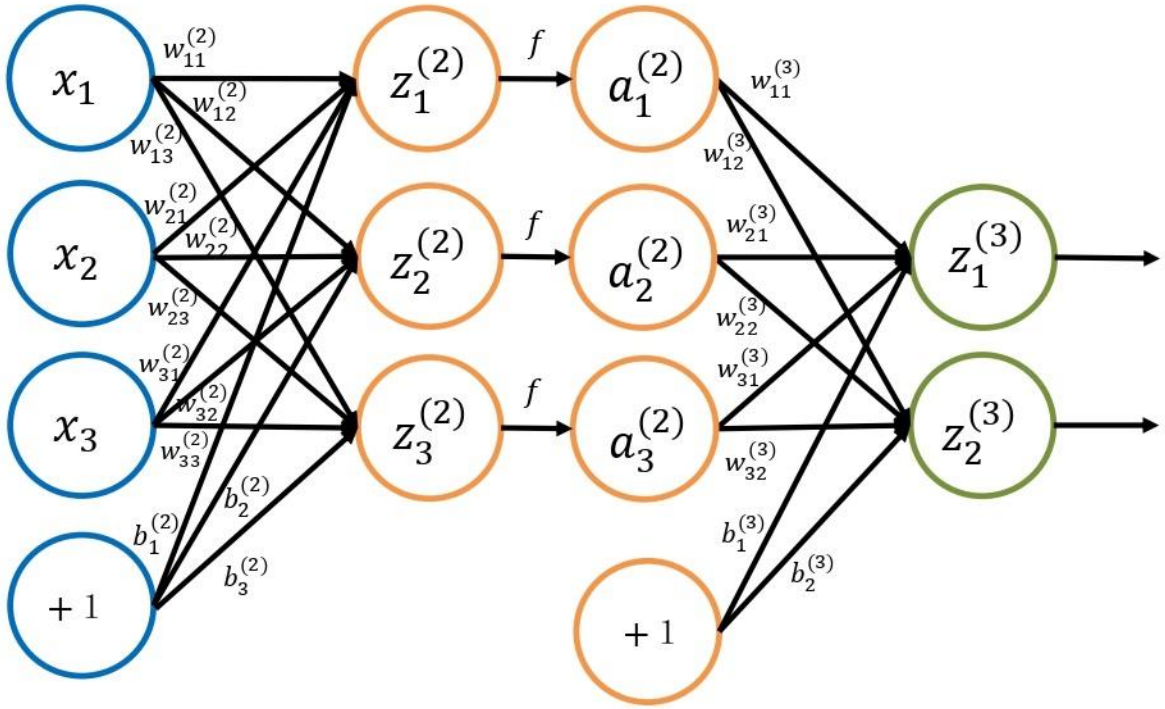
- Lot of matrix operations!
- Represent NN as computational graphs!

Computational Graphs

A neural network can be represented as a computational graph...

- it has compute nodes (operations)
- it has edges that connect nodes (data flow)
- it is directional
- it can be organized into 'layers'

Computational Graphs



$$z_k^{(2)} = \sum_i x_i w_{ik}^{(2)} + b_i^{(2)}$$

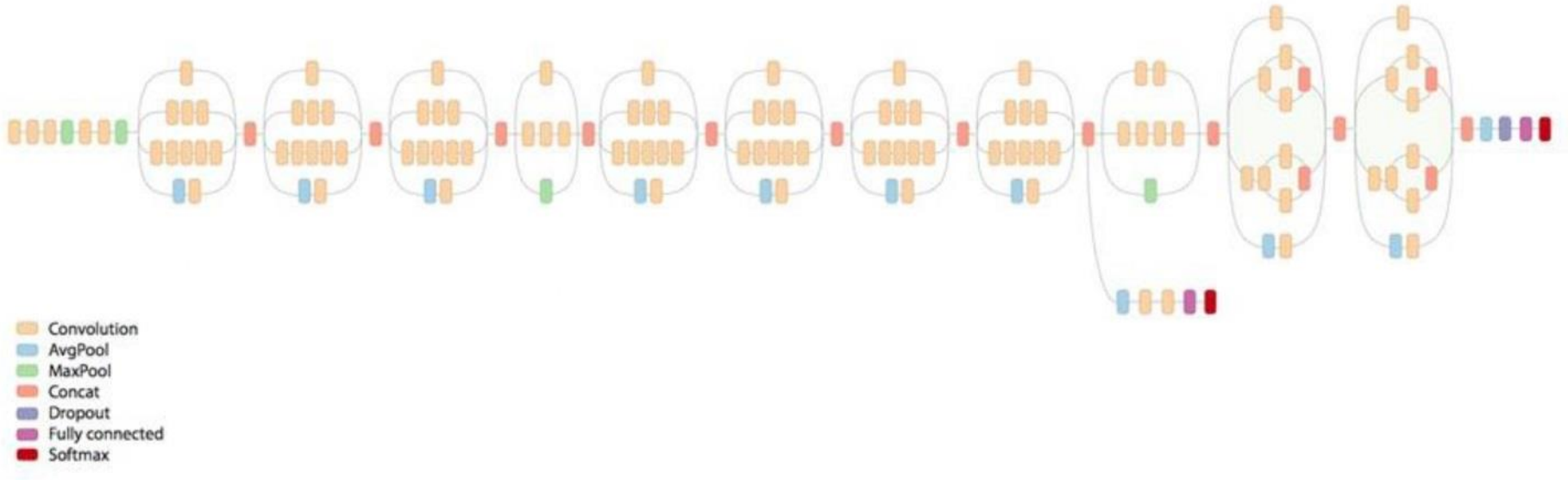
$$a_k^{(2)} = f(z_k^{(2)})$$

$$z_k^{(3)} = \sum_i a_i^{(2)} w_{ik}^{(3)} + b_i^{(3)}$$

...

Computational Graphs

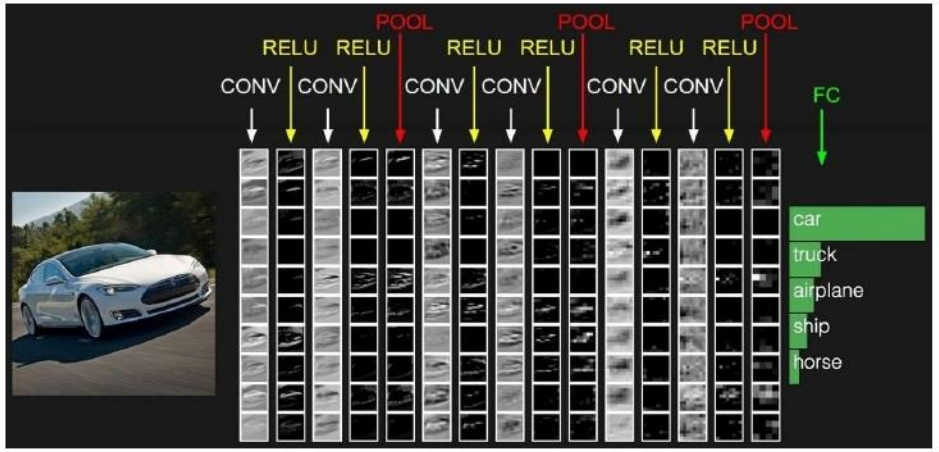
- From a set of neurons to a Structured Compute Pipeline



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

Computational Graphs

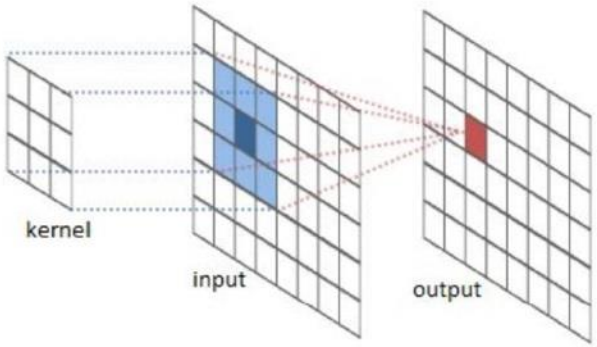
- The computation of Neural Network has further meanings:
  - The multiplication of $\boldsymbol{W_i}$ and $\boldsymbol{x}$: encode input information
  - The activation function: select the key features



Source: https://www.zybuluo.com/liuhui0803/note/981434

Computational Graphs

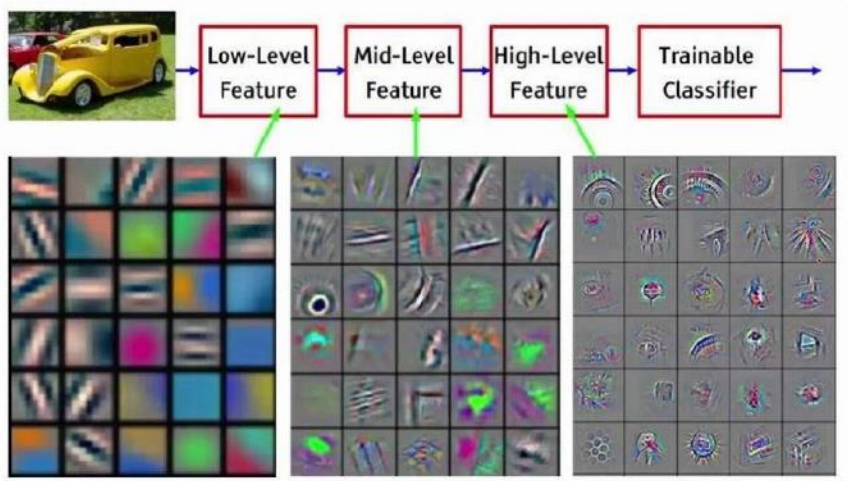
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



Source: https://www.zcfy.cc/original/understanding-convolutions-colah-s-blog

Computational Graphs

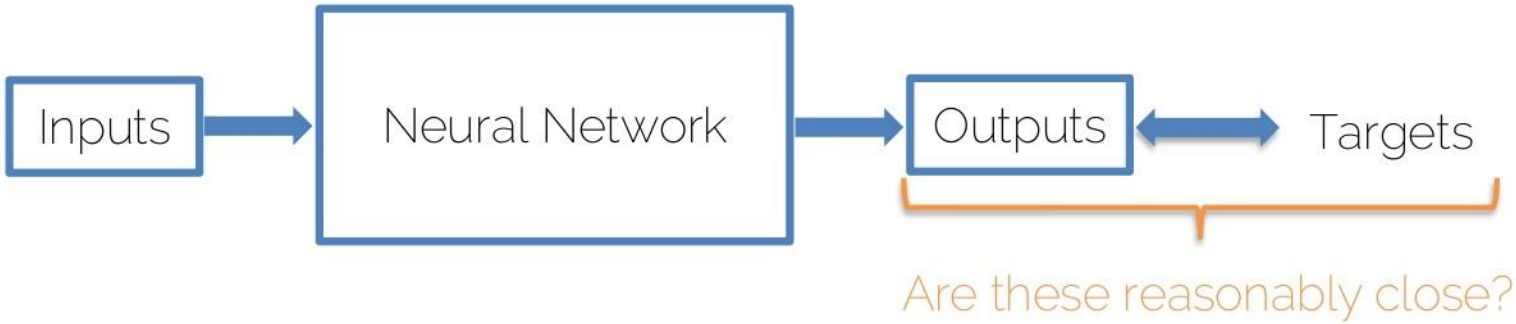
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



Source: https://www.zybuluo.com/liuhui0803/note/981434

Loss Functions

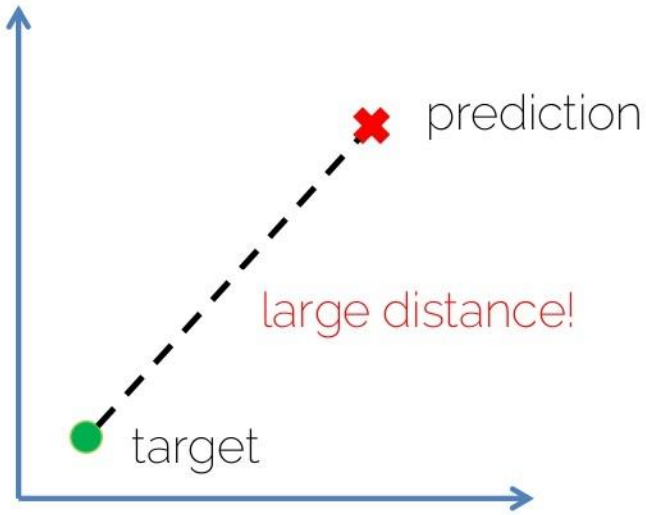
What's Next?



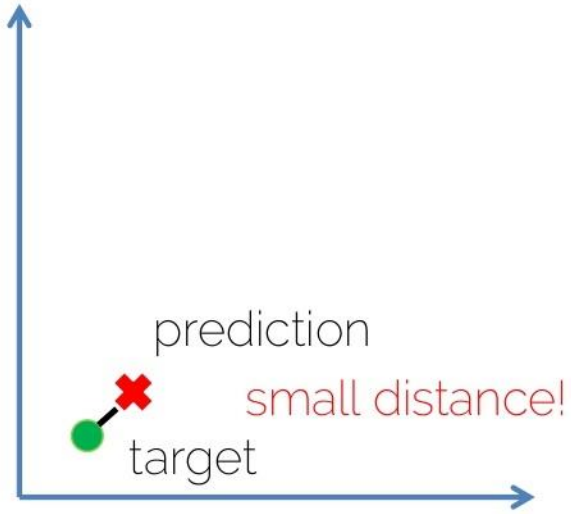
We need a way to describe how close the network's outputs (= predictions) are to the targets!

What's Next?

Idea: calculate a '**distance**' between prediction and target!



bad prediction



good prediction

Loss Functions

- A function to **measure the goodness of the predictions** (or equivalently, the network's performance)

Intuitively, …

- a **large loss indicates bad predictions**/performance (→ performance needs to be improved by training the model)
- the choice of the loss function depends on the concrete problem or the distribution of the target variable

Regression Loss

- L1 Loss:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = \frac{1}{n}\sum_{i}^{n} ||y_i\ - \hat{y_i}||_1$$

- MSE Loss:

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

Binary Cross Entropy

- Loss function for binary (yes/no) classification

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = -\sum_{i=1}^{n} (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

Yes! (0.8)

No! (0.2)

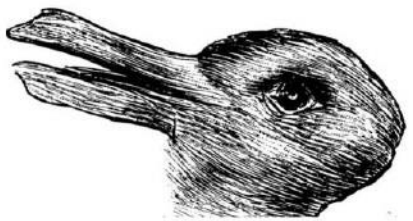
The network predicts the probability of the input belonging to the "yes" class!

Cross Entropy

= loss function for multi-class classification

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

This generalizes the binary case from the slide before!



dog (0.1)

rabbit (0.2)

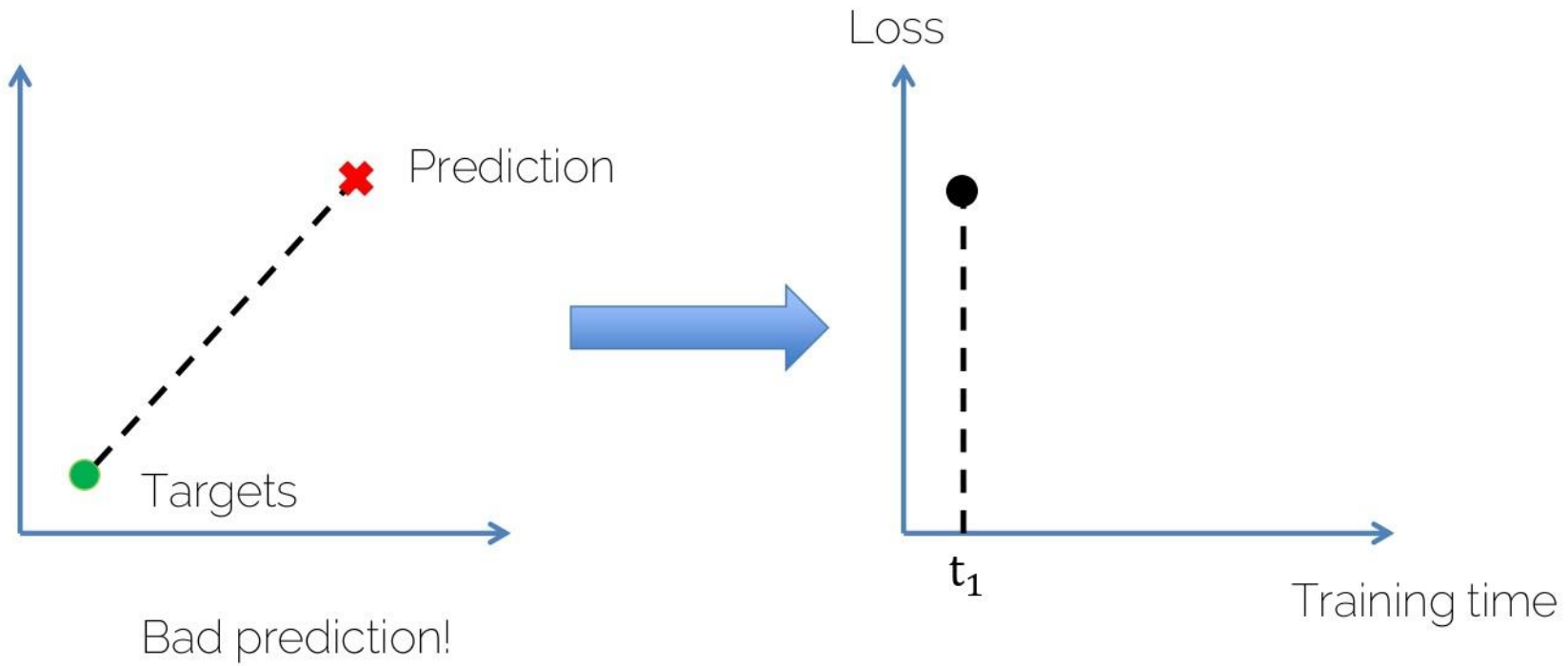
duck (0.7)

…

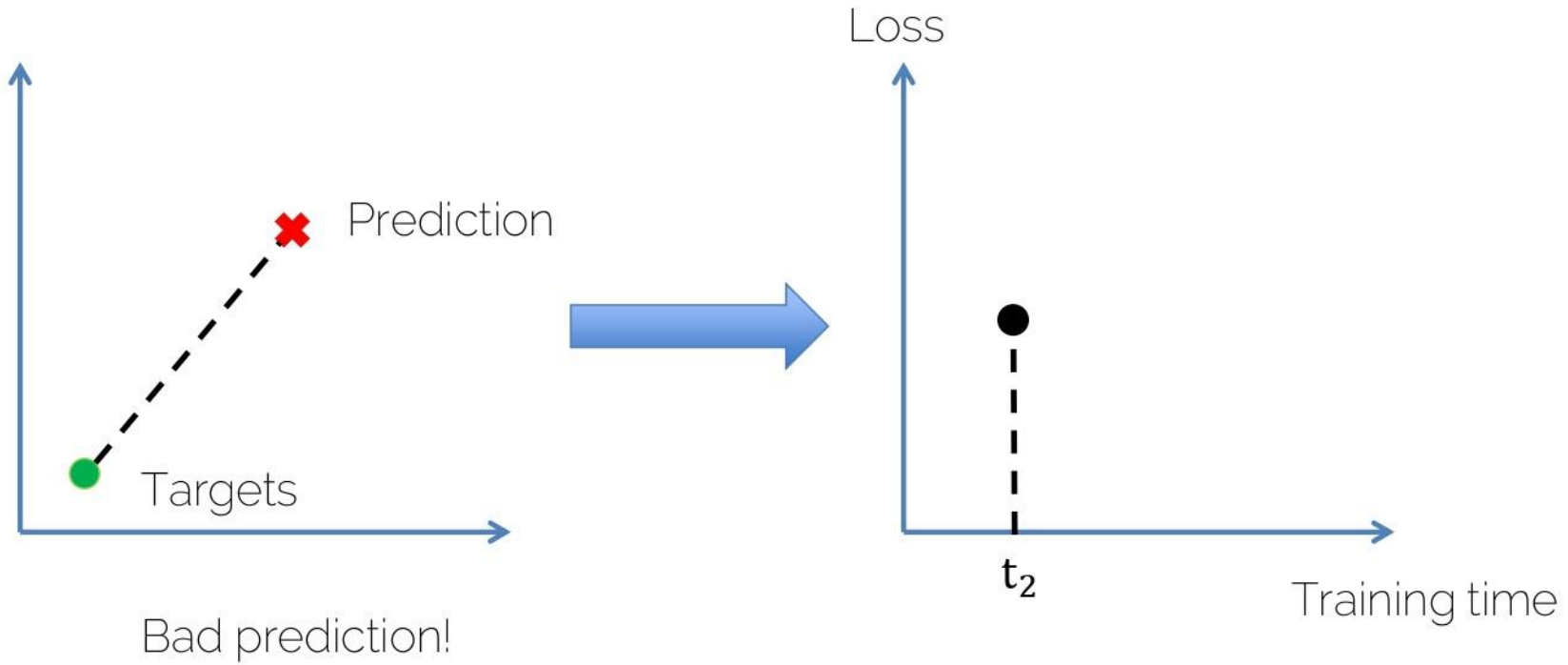
More General Case

- Ground truth: $\boldsymbol{y}$
- Prediction: $\hat{\boldsymbol{y}}$
- Loss function: $\boldsymbol{L(y, \hat{y})}$
- Motivation:
  - minimize the loss <=> find better predictions
  - predictions are generated by the NN
  - find better predictions <=> find better NN

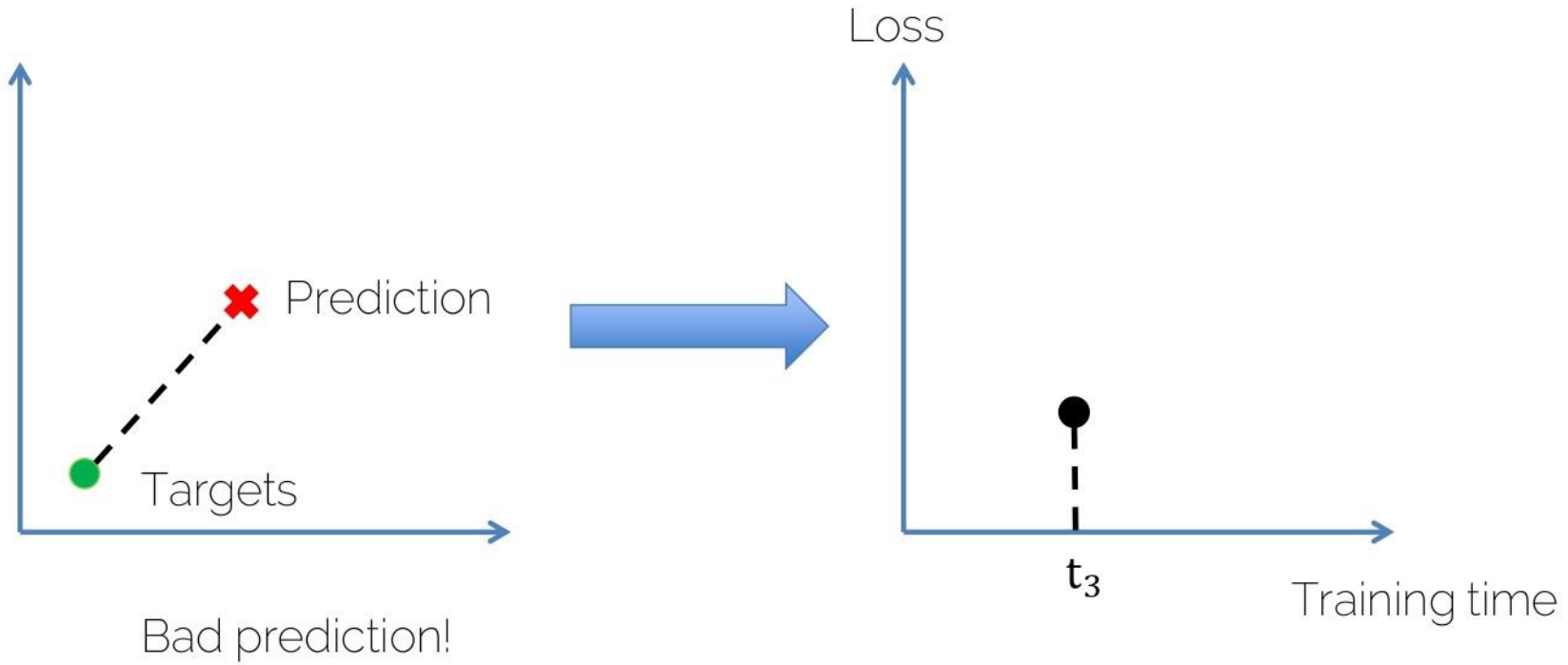
Initially



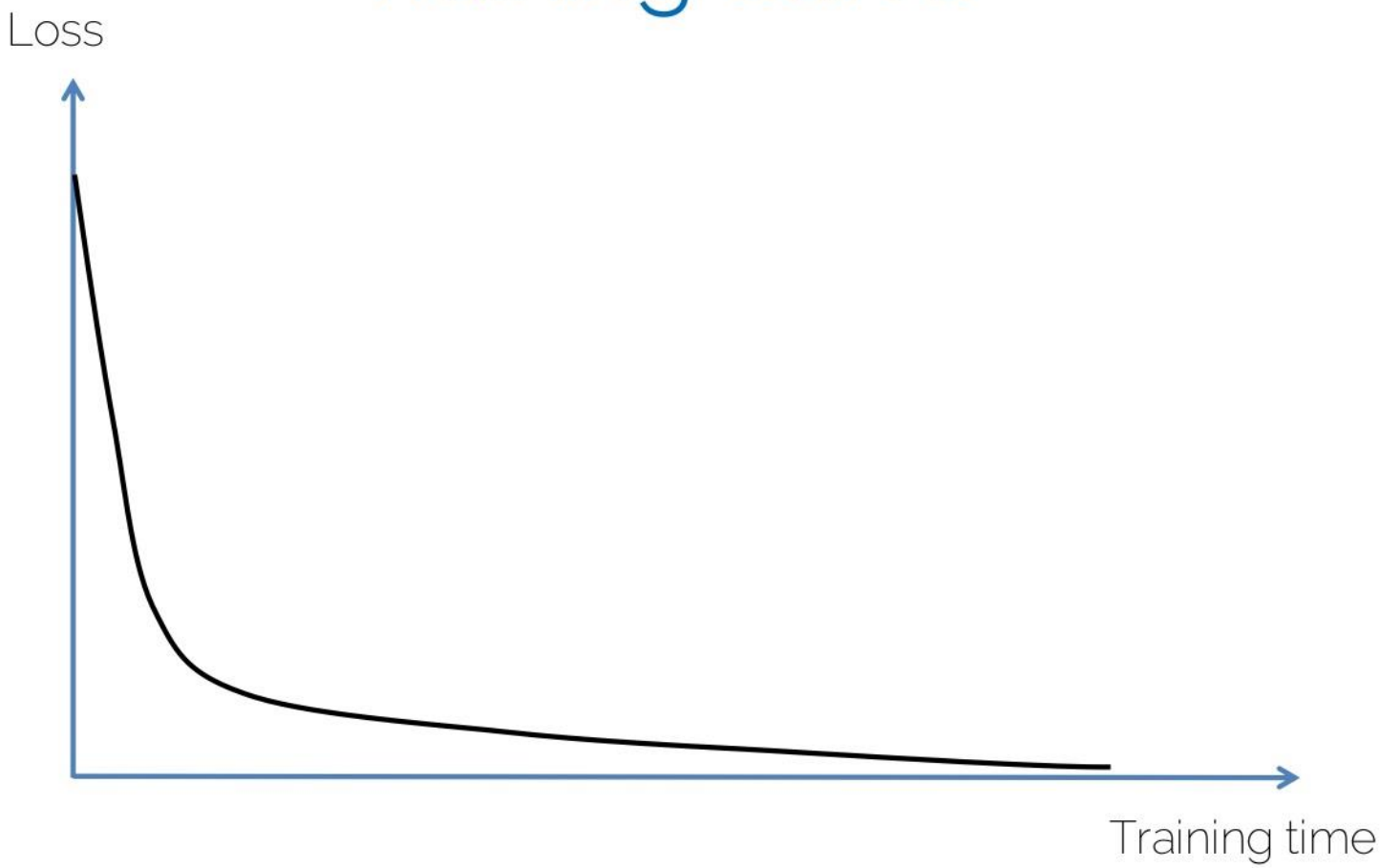
During Training…



During Training…

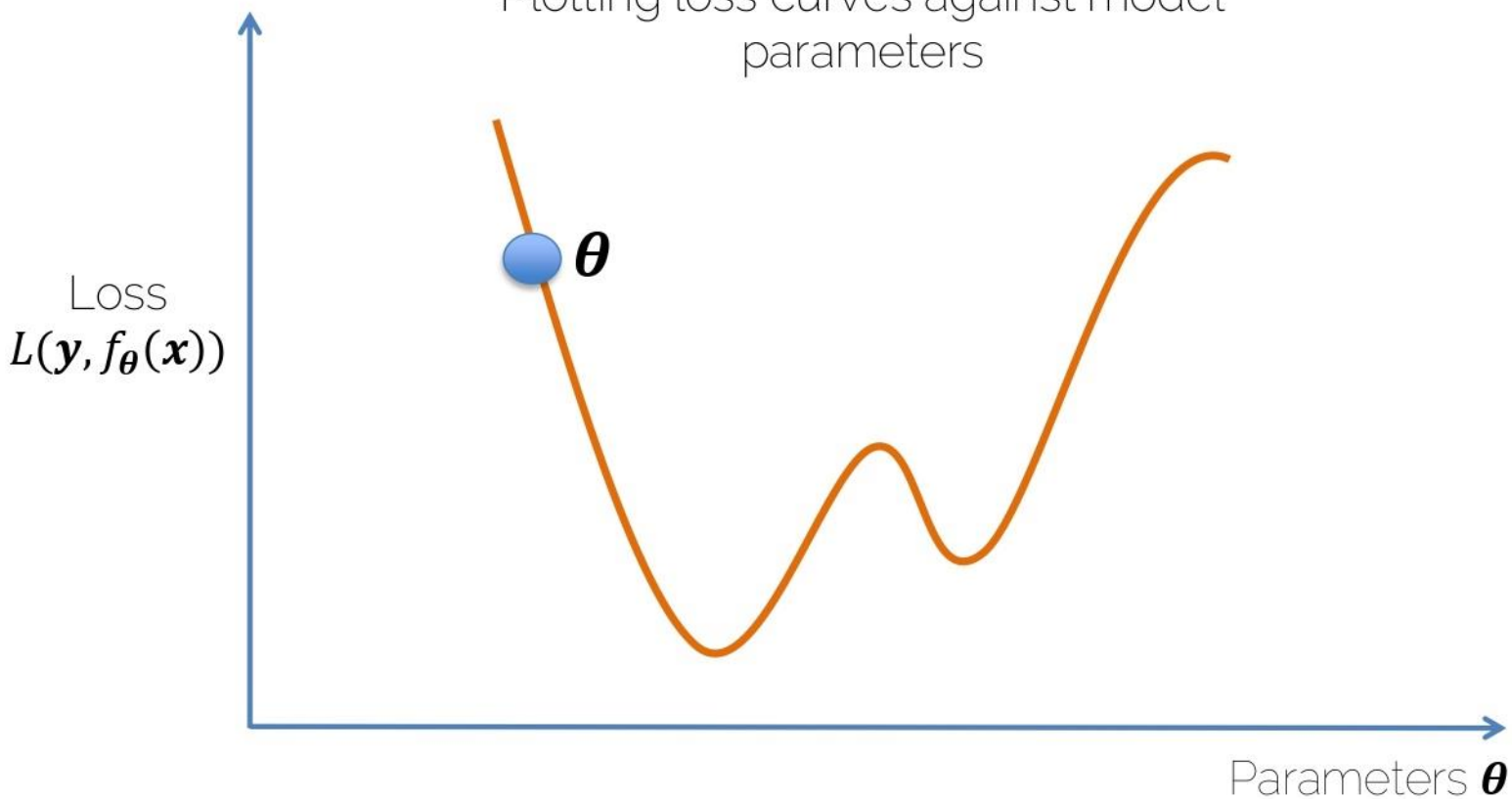


Training Curve



How to Find a Better NN?

Plotting loss curves against model parameters



How to Find a Better NN?

- Why gradient descent?
  - Easy to compute using compute graphs
- Other methods include
  - Newtons method
  - L-BFGS
  - Adaptive moments
  - Conjugate gradient

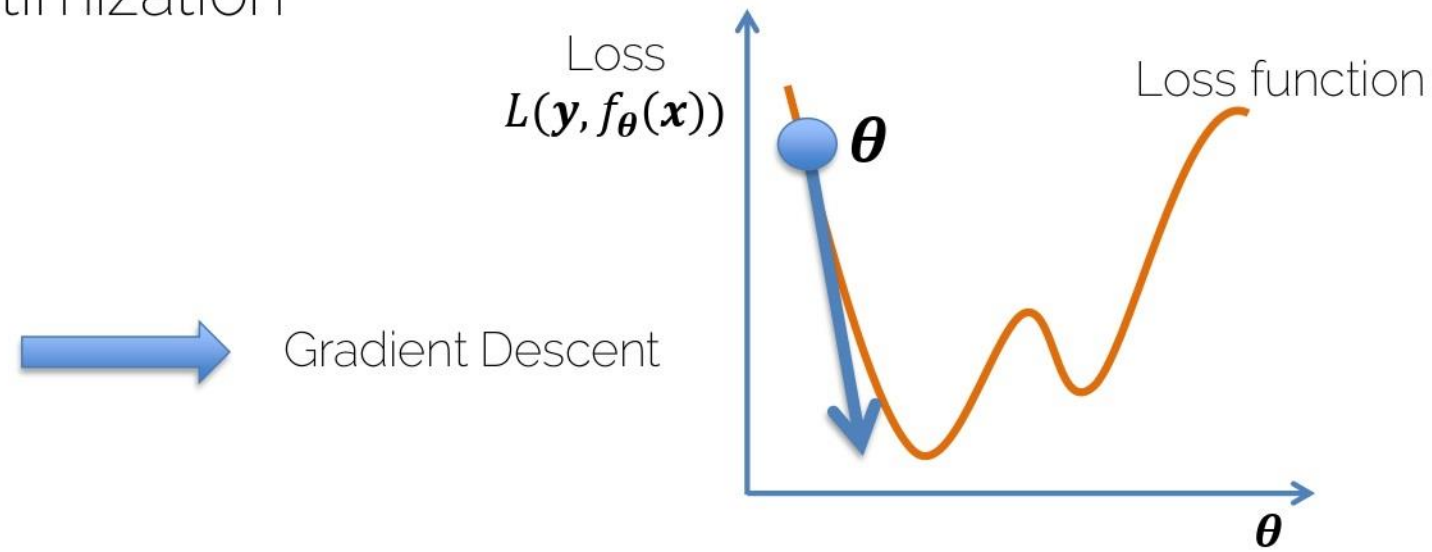
How to Find a Better NN?

- Loss function: $L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$
- Neural Network: $f_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Goal:
  - minimize the loss w. r. t. $\boldsymbol{\theta}$

Optimization! We train compute graphs with some optimization techniques!

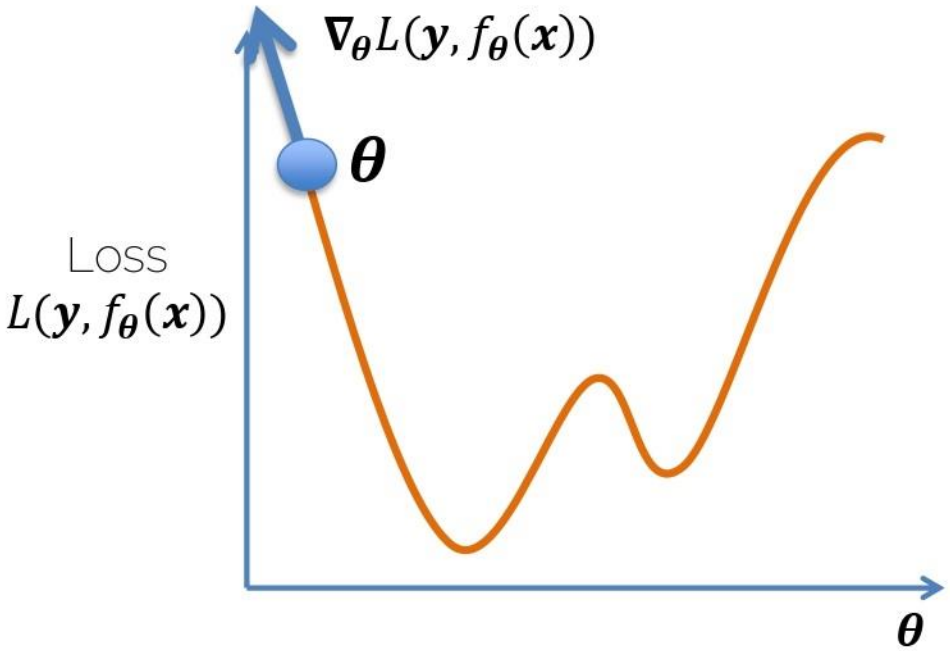
How to Find a Better NN?

- Minimize: $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w.r.t. $\boldsymbol{\theta}$
- In the context of NN, we use gradient-based optimization



How to Find a Better NN?

- Minimize: $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w.r.t. $\boldsymbol{\theta}$



Learning rate

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$$

$$\boldsymbol{\theta}^* = \arg\min L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$$

How to Find a Better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given one layer NN with no activation function

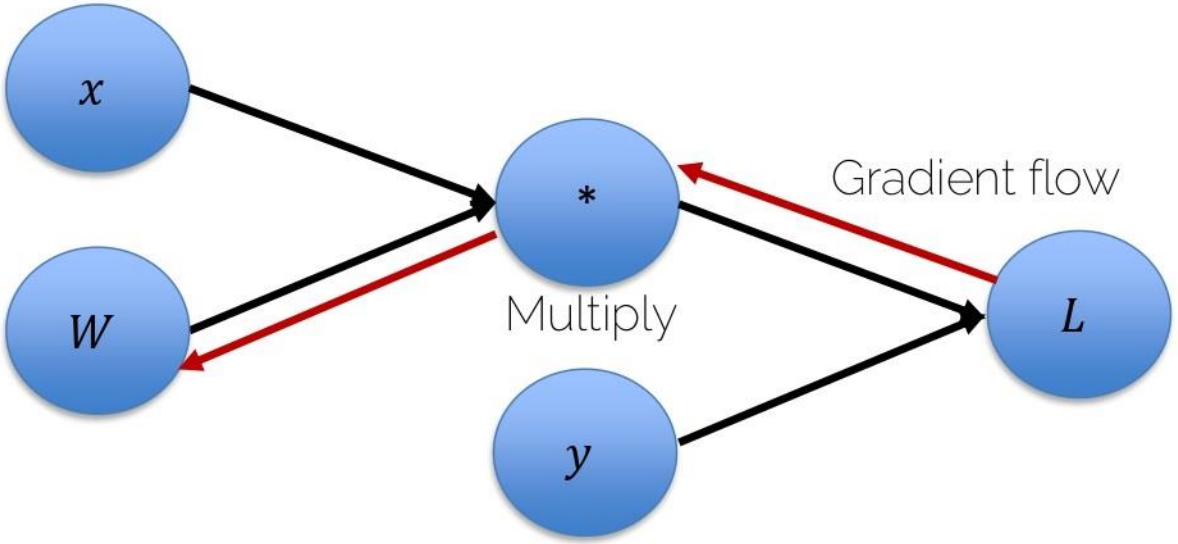
$$\boldsymbol{f_\theta(x) = Wx}\,, \qquad \boldsymbol{\theta = W}$$

Later $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

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

How to Find a Better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given one layer NN with no activation function
- Given MSE Loss: $\boldsymbol{L(y, \hat{y}; \theta)} = \frac{1}{n}\sum_i^n ||y_i - \boldsymbol{W} \cdot x_i||_2^2$



How to Find a Better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given a one layer NN with no activation function

$$\boldsymbol{f_\theta(x) = Wx}\,, \qquad \boldsymbol{\theta = W}$$

- Given MSE Loss: $\boldsymbol{L(y, \hat{y}; \theta) = \frac{1}{n}\sum_i^n ||W \cdot x_i - y_i||_2^2}$
- $\boldsymbol{\nabla_\theta L\big(y, f_\theta(x)\big) = \frac{1}{n}\sum_i^n (W \cdot x_i - y_i) \cdot x_i^T}$

How to Find a Better NN?

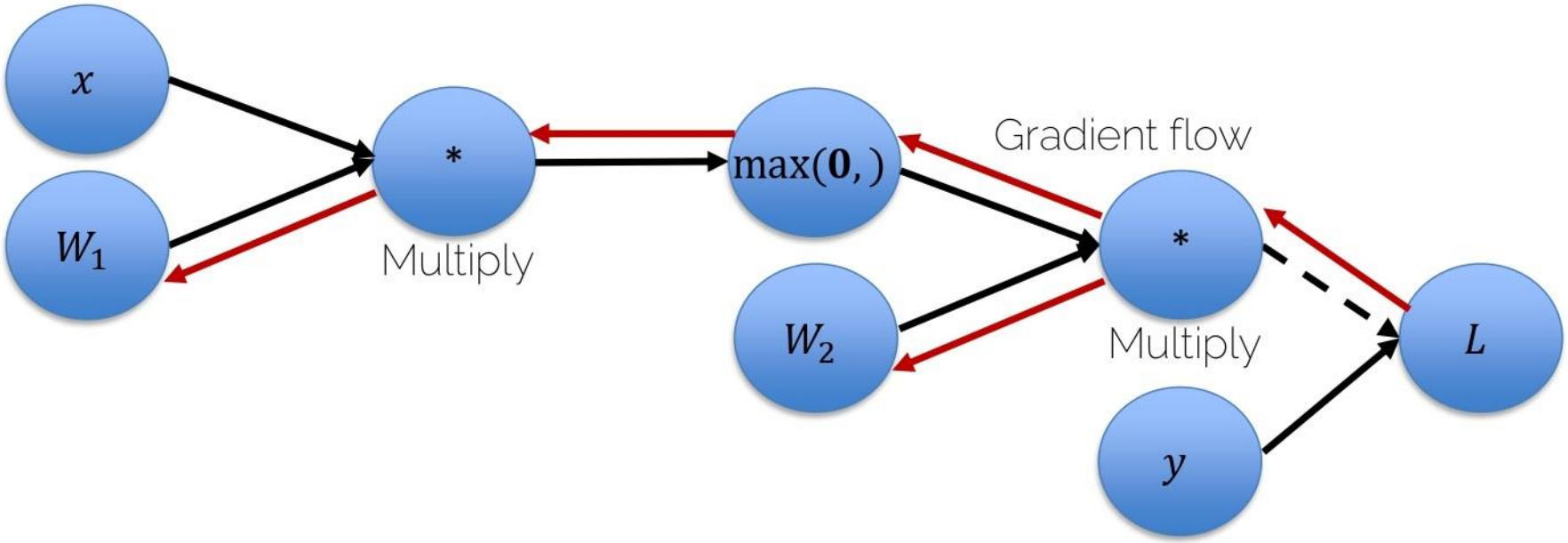
- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given a multi-layer NN with many activations

$$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3},\max(\boldsymbol{0},\boldsymbol{W_2}\max(\boldsymbol{0},\boldsymbol{W_1}\boldsymbol{x}))))$$

- Gradient descent for $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w. r. t. $\boldsymbol{\theta}$
  - Need to propagate gradients from end to first layer ($\boldsymbol{W_1}$).

How to Find a Better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given multi-layer NN with many activations



How to Find a Better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given multilayer layer NN with many activations

  $$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3},\max(\mathbf{0},\boldsymbol{W_2}\max(\mathbf{0},\boldsymbol{W_1}\boldsymbol{x}))))$$

- Gradient descent solution for $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w. r. t. $\boldsymbol{\theta}$
  - Need to propagate gradients from end to first layer ($\boldsymbol{W_1}$)
- Backpropagation: Use chain rule to compute gradients
  - Compute graphs come in handy!

Summary

- Neural Networks are computational graphs
- Goal: for a given train set, find optimal weights

- Optimization is done using gradient-based solvers
  - Many options (more in the next lectures)

- Gradients are computed via backpropagation
  - Nice because can easily modularize complex functions

Backprop

The Importance of Gradients

- Our optimization schemes are based on computing gradients

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

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

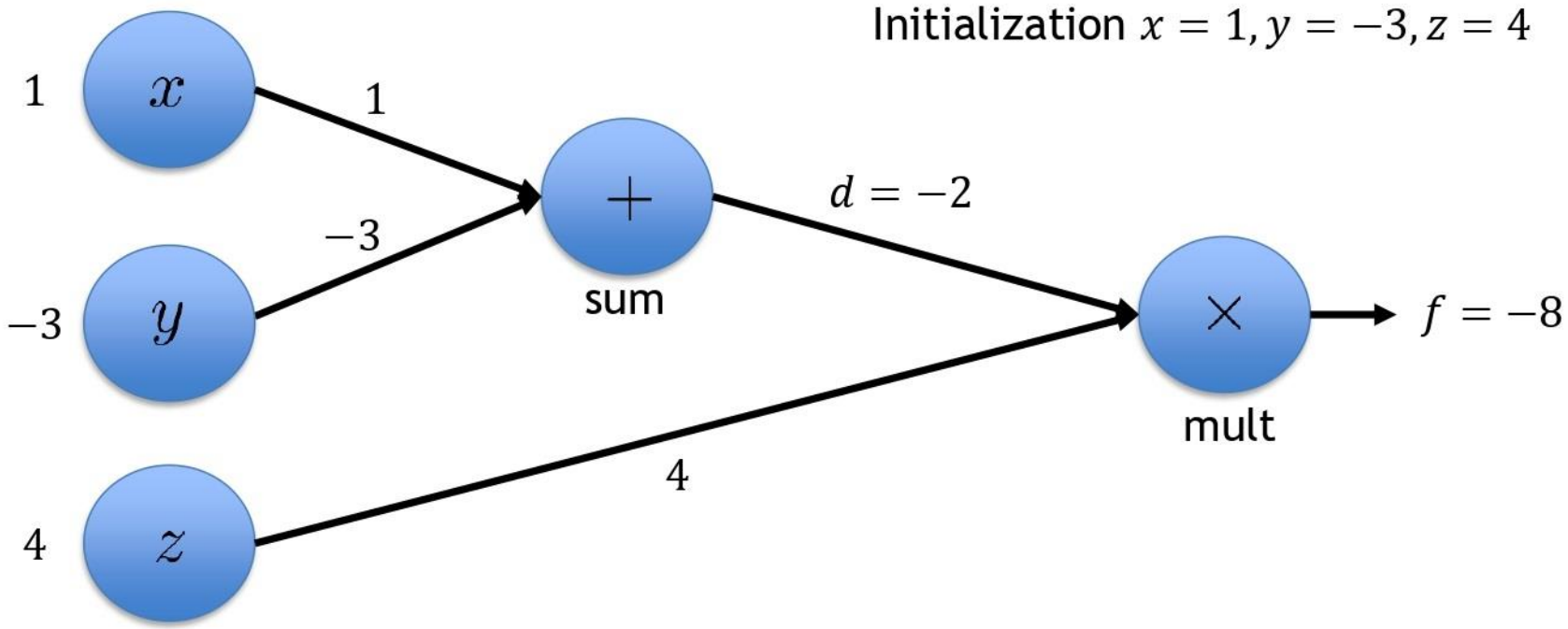
- Break down gradient computation

Backpropagation

Rumelhart 1986

Backprop: Forward Pass

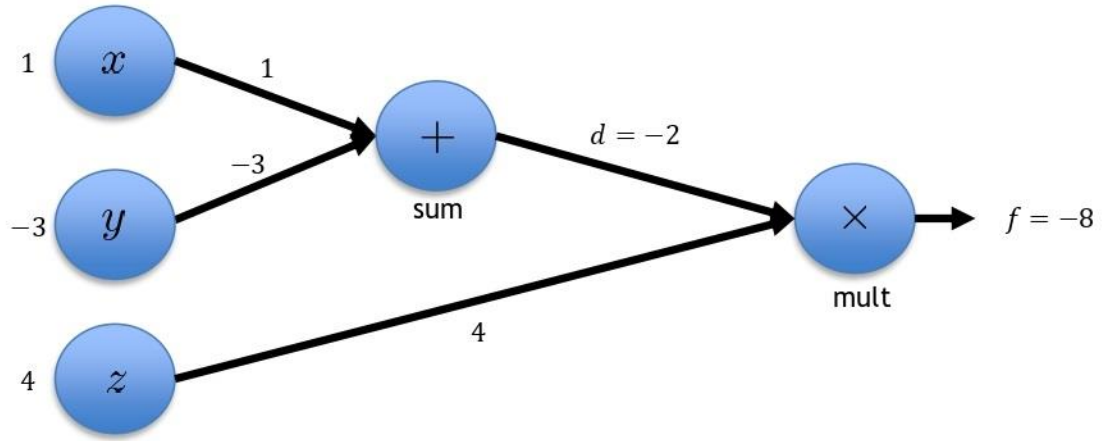
- $f(x, y, z) = (x + y) \cdot z$



Backprop: Backward Pass

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

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



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

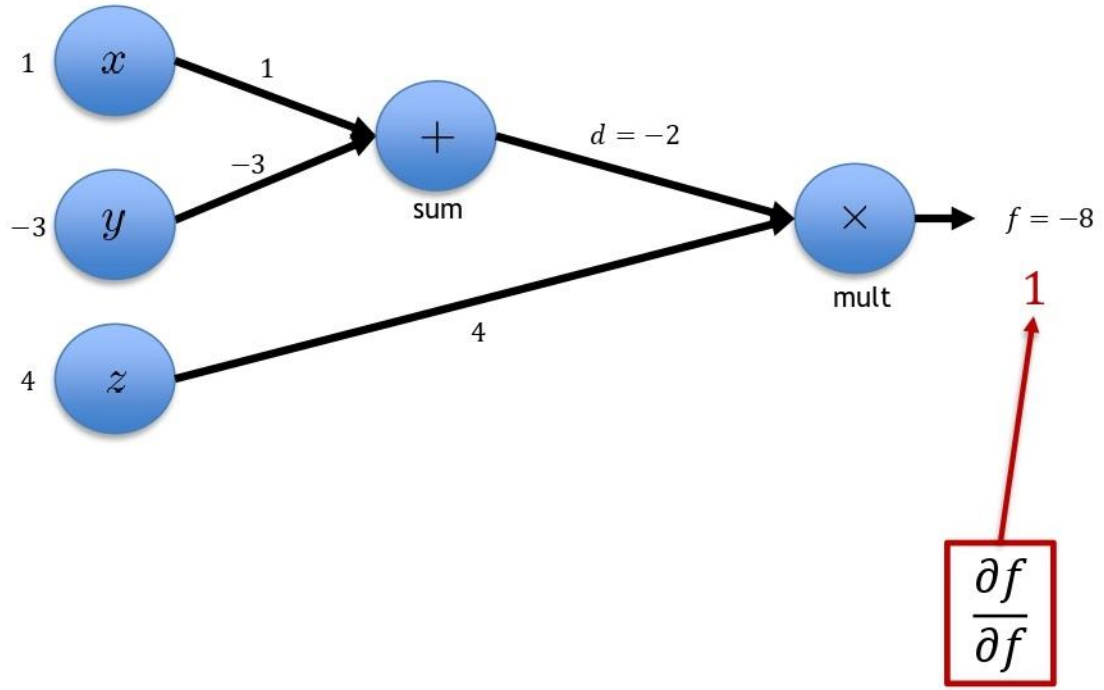
$f = d \cdot z$ $\quad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$

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

Backprop: Backward Pass

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

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



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

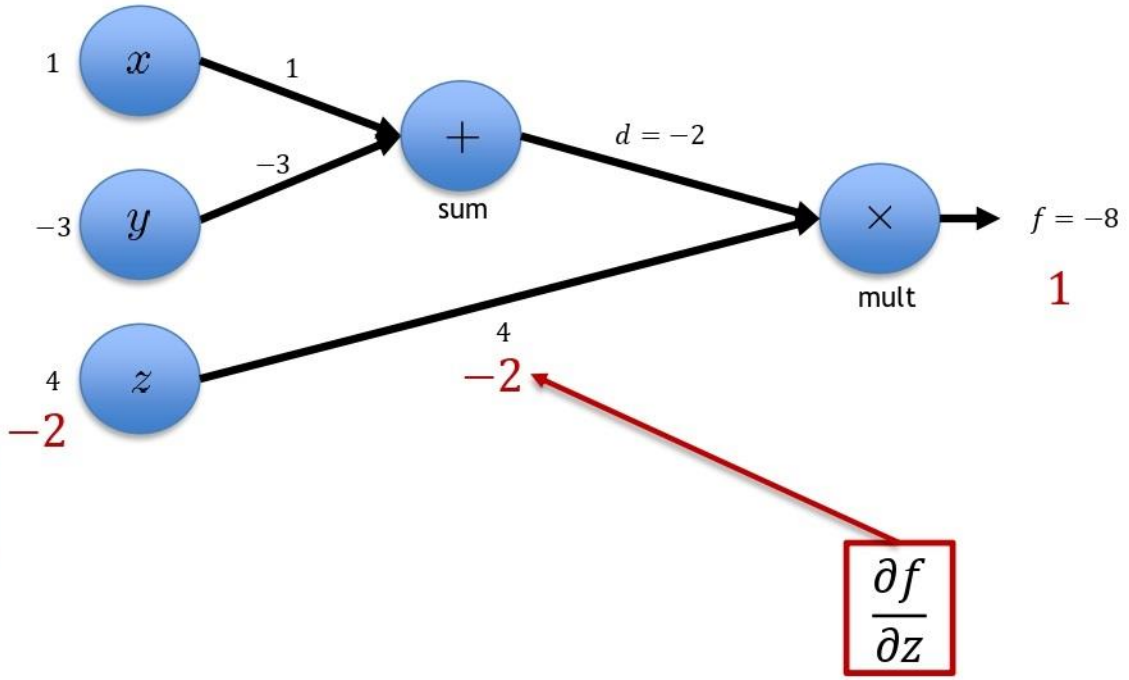
$f = d \cdot z \qquad \frac{\partial f}{\partial d} = z, \frac{\partial f}{\partial z} = d$

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

Backprop: Backward Pass

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

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



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

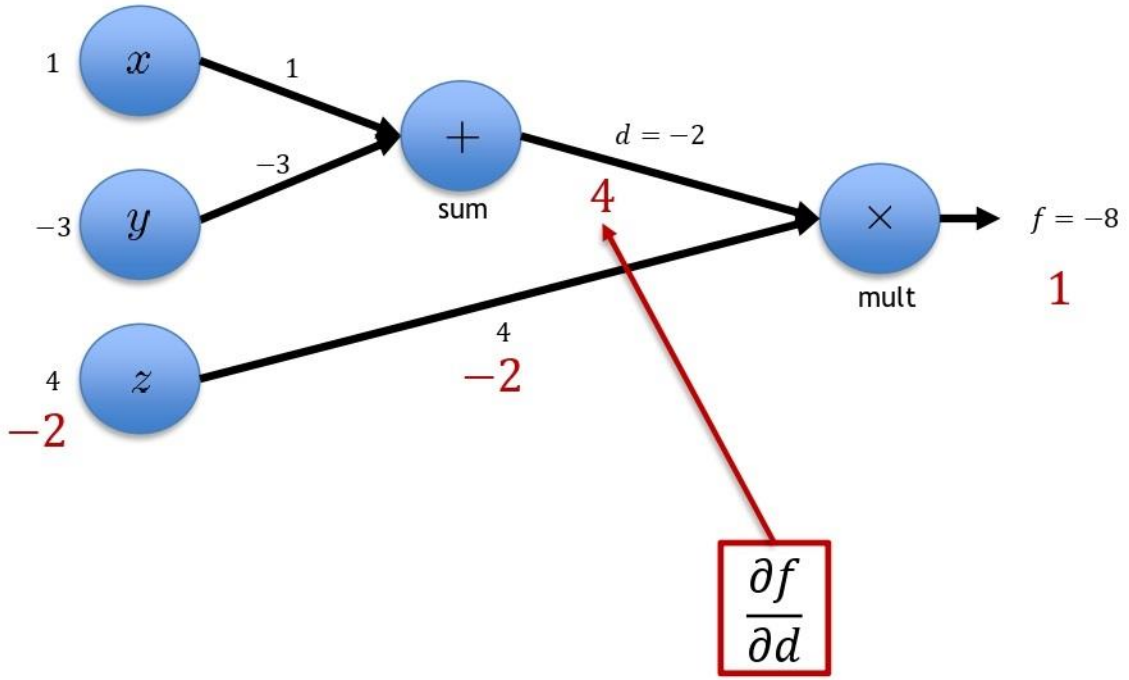
$f = d \cdot z$ $\quad \frac{\partial f}{\partial d} = z, \boxed{\frac{\partial f}{\partial z} = d}$

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

Backprop: Backward Pass

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

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



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

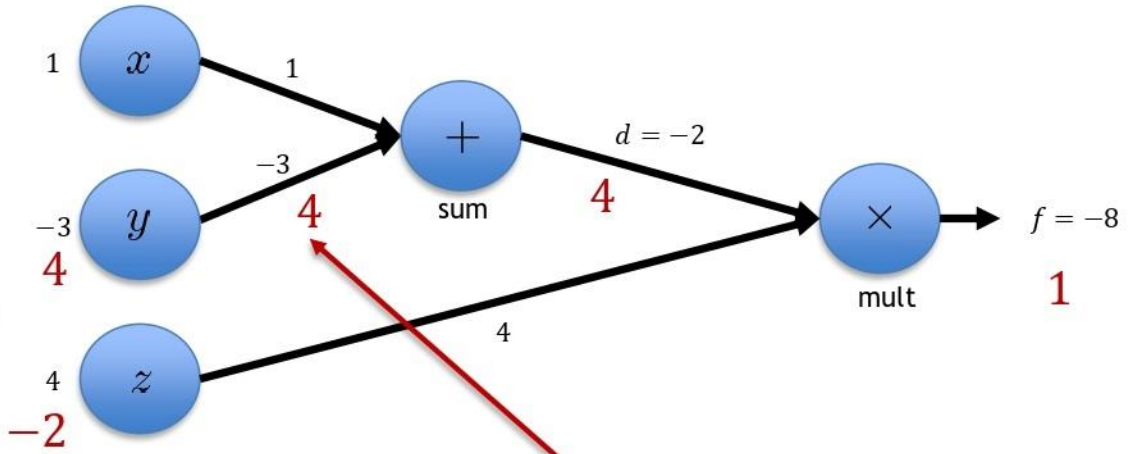
$f = d \cdot z$ $\quad \boxed{\frac{\partial f}{\partial d} = z}, \frac{\partial f}{\partial z} = d$

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

Backprop: Backward Pass

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

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



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

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

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

Chain Rule:

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

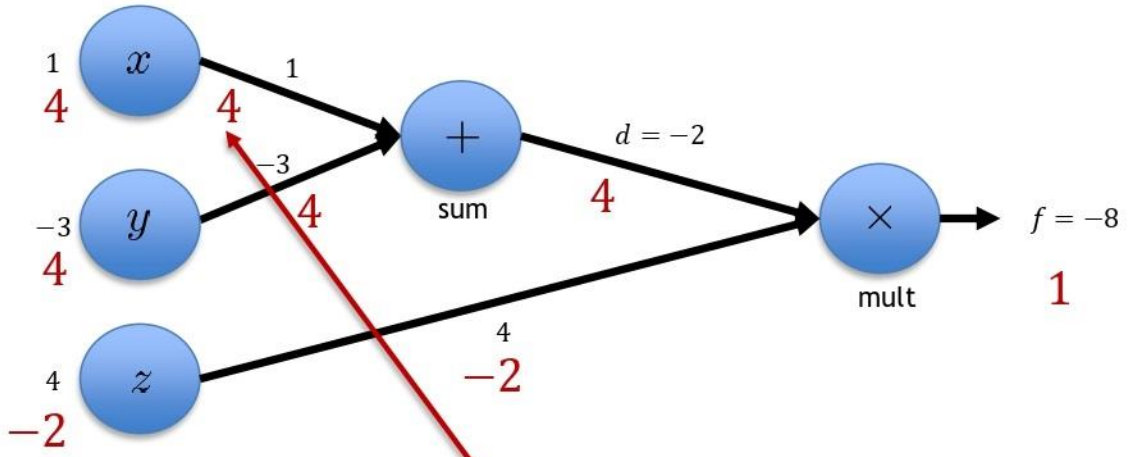
$\boxed{\frac{\partial f}{\partial y}}$

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

Backprop: Backward Pass

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

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



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

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

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

Chain Rule:

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

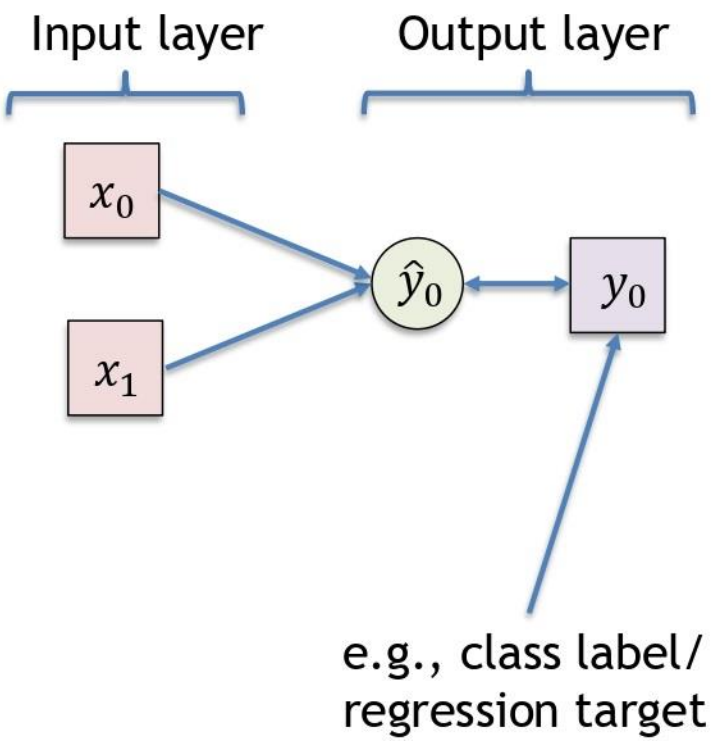
$\frac{\partial f}{\partial x}$

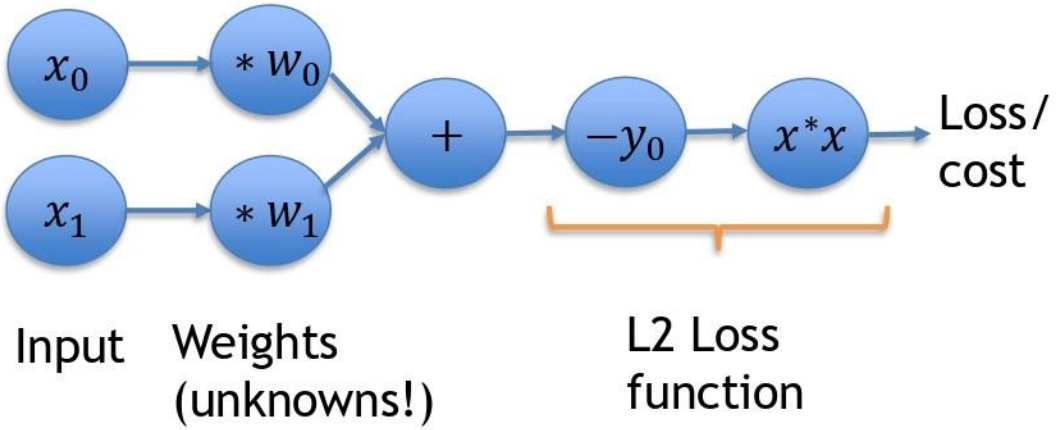
$\rightarrow \frac{\partial f}{\partial x} = 4 \cdot 1 = 4$

Compute Graphs -> Neural Networks

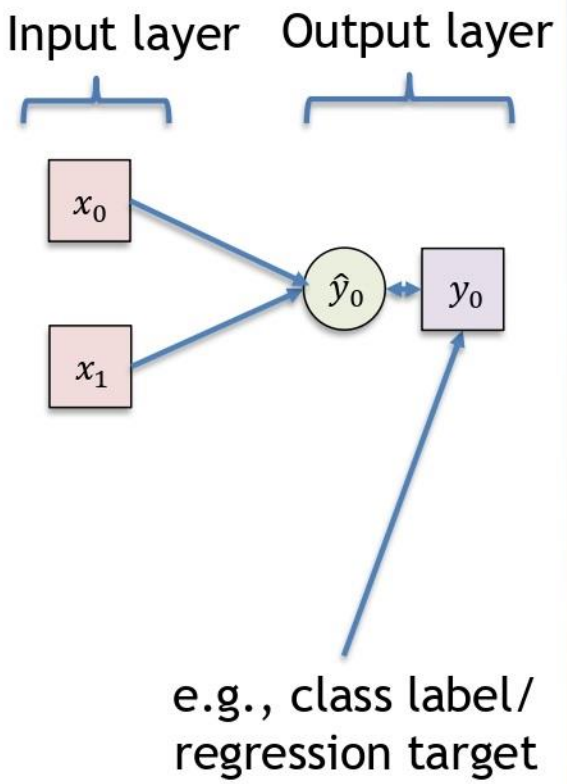
- x_k input variables
- $w_{l,m,n}$ network weights (note 3 indices)
  - l which layer
  - m which neuron in layer
  - n which weight in neuron
- $\hat{y}_i$ computed output (i output dim; n_{out})
- y_i ground truth targets
- L loss function

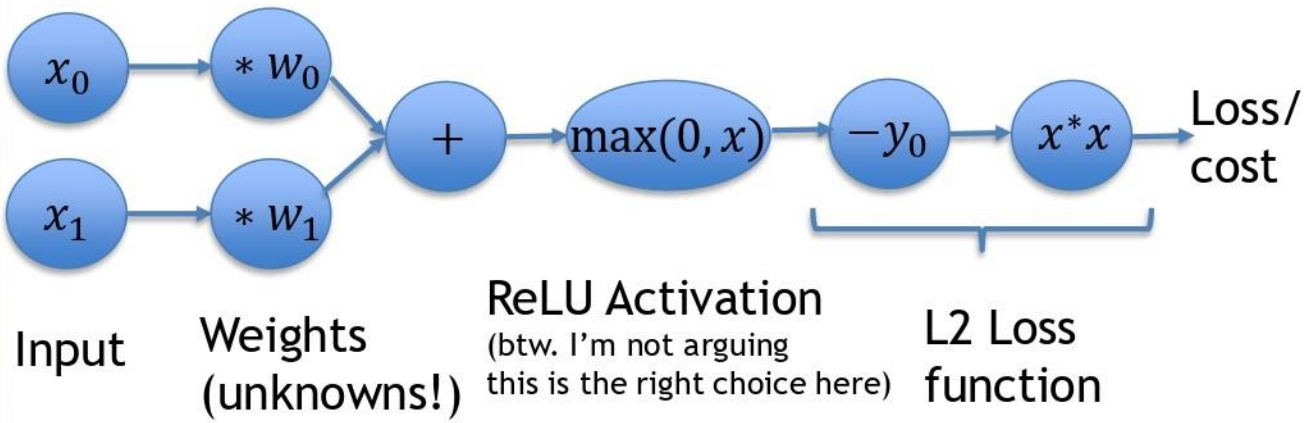
Compute Graphs -> Neural Networks





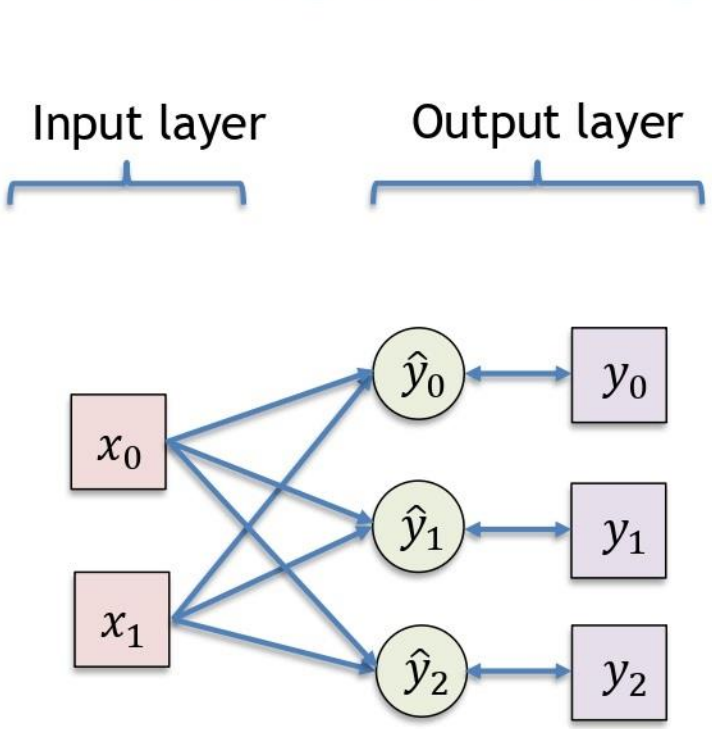
Compute Graphs -> Neural Networks

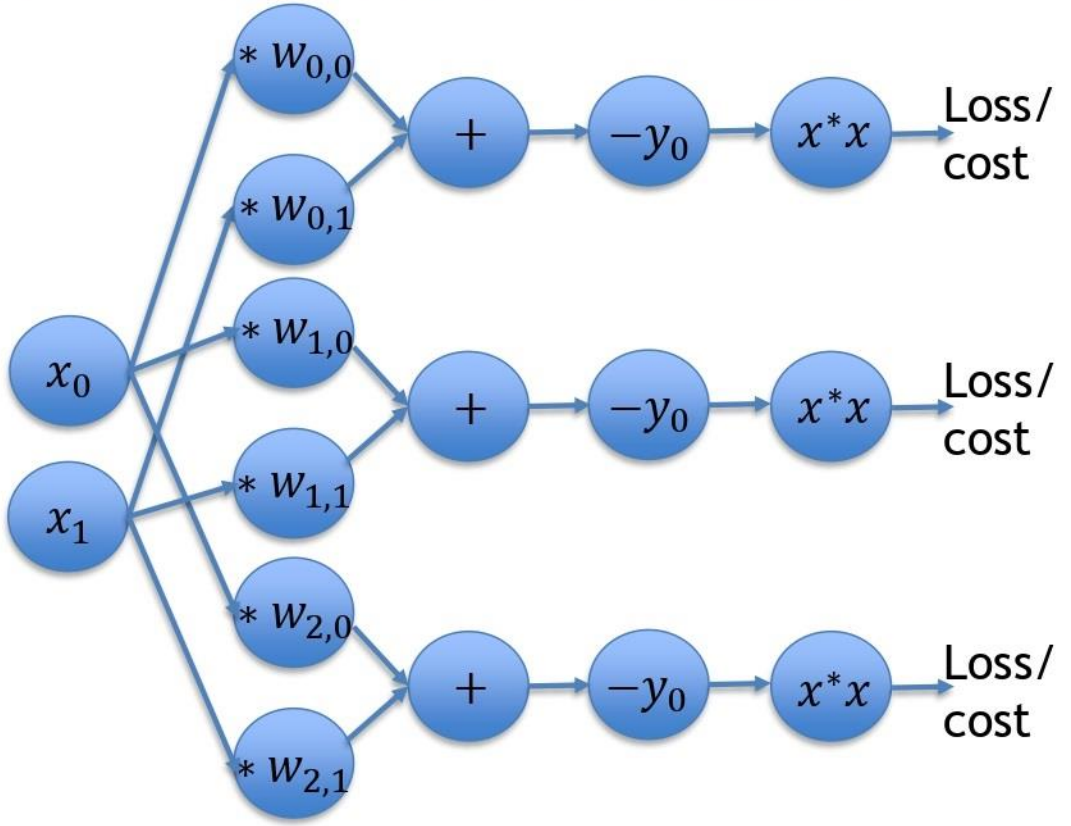




We want to compute gradients w.r.t. all weights W

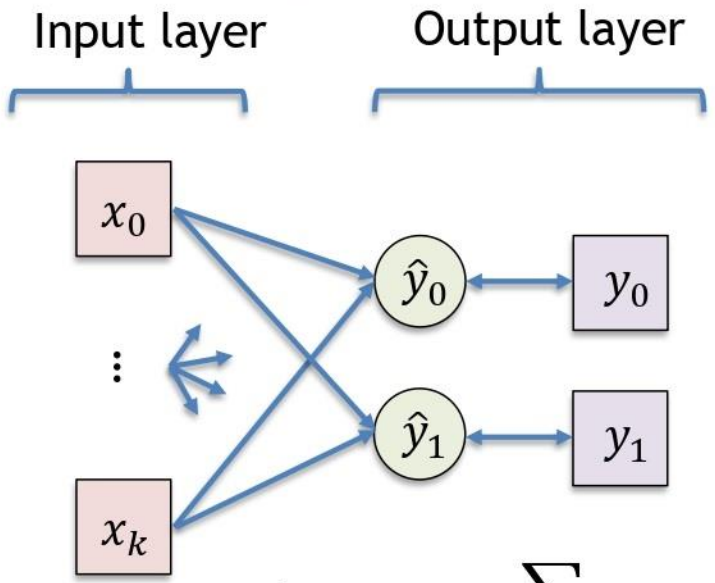
Compute Graphs -> Neural Networks





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

Compute Graphs -> Neural Networks



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

Activation function

bias

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

$$L = \sum_i L_i$$

L: sum over loss per sample, e.g. L2 loss $\longrightarrow$ simply sum up squares:

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

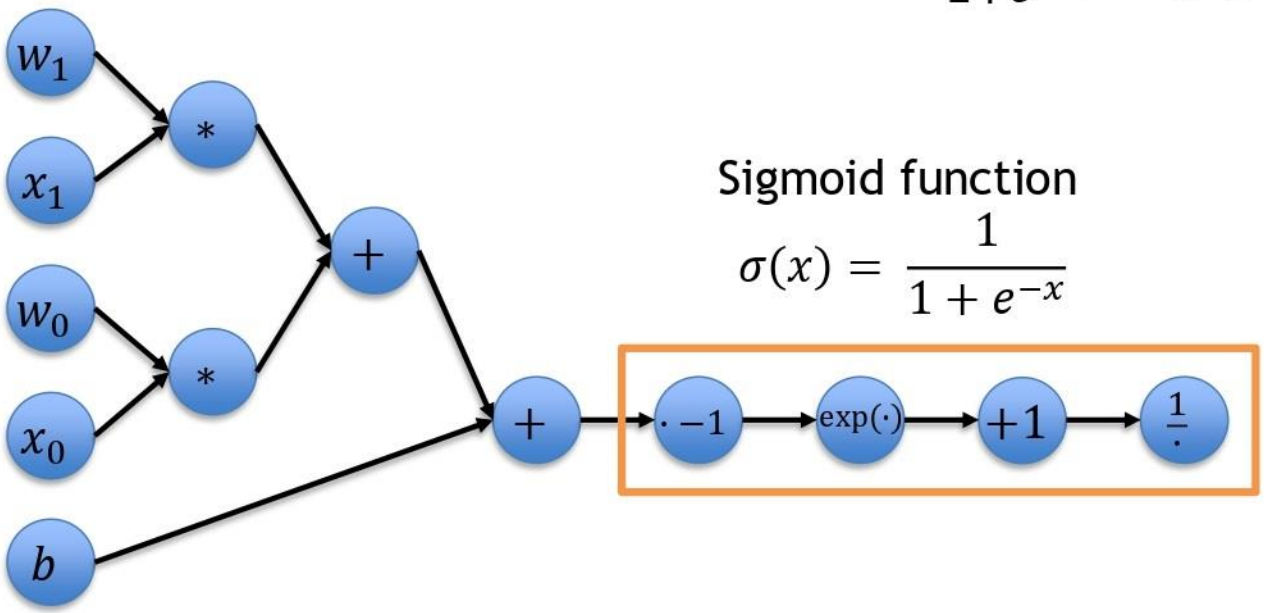
$\longrightarrow$ use chain rule to compute partials

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

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

NNs as Computational Graphs

- We can express any kind of functions in a computational graph, e.g. $f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1+e^{-(b+w_0 x_0 + w_1 x_1)}}$

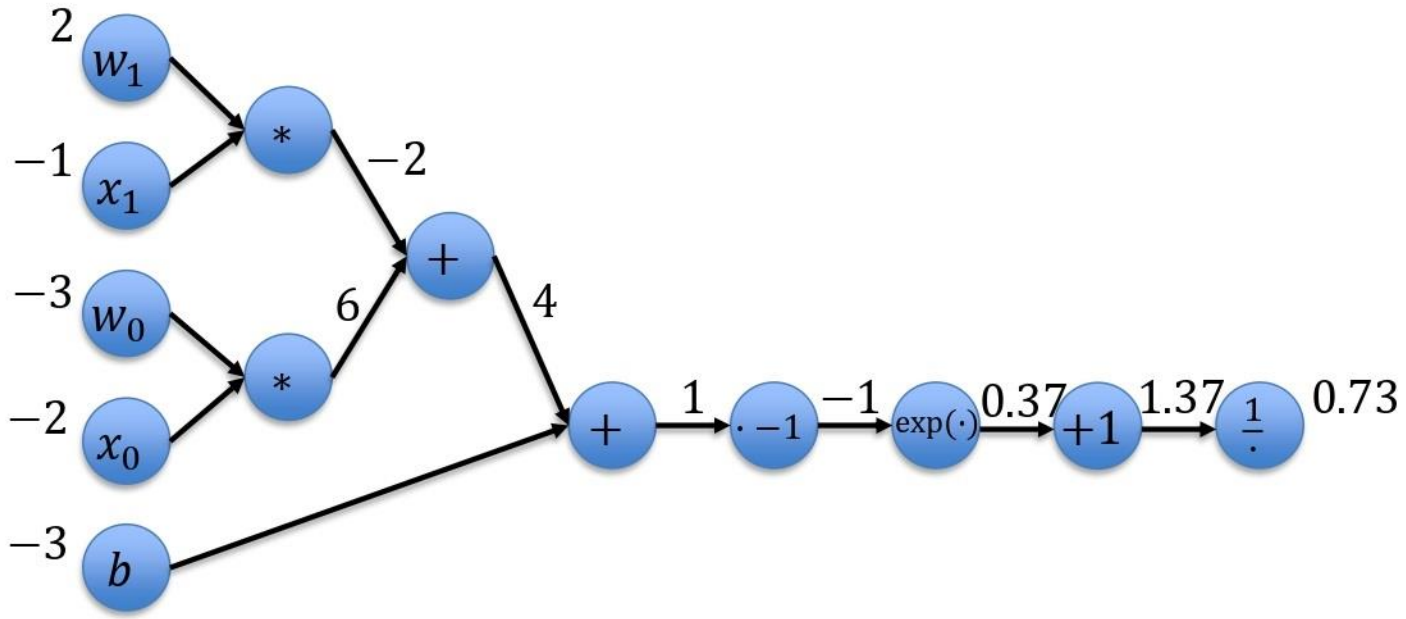


Sigmoid function

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

NNs as Computational Graphs

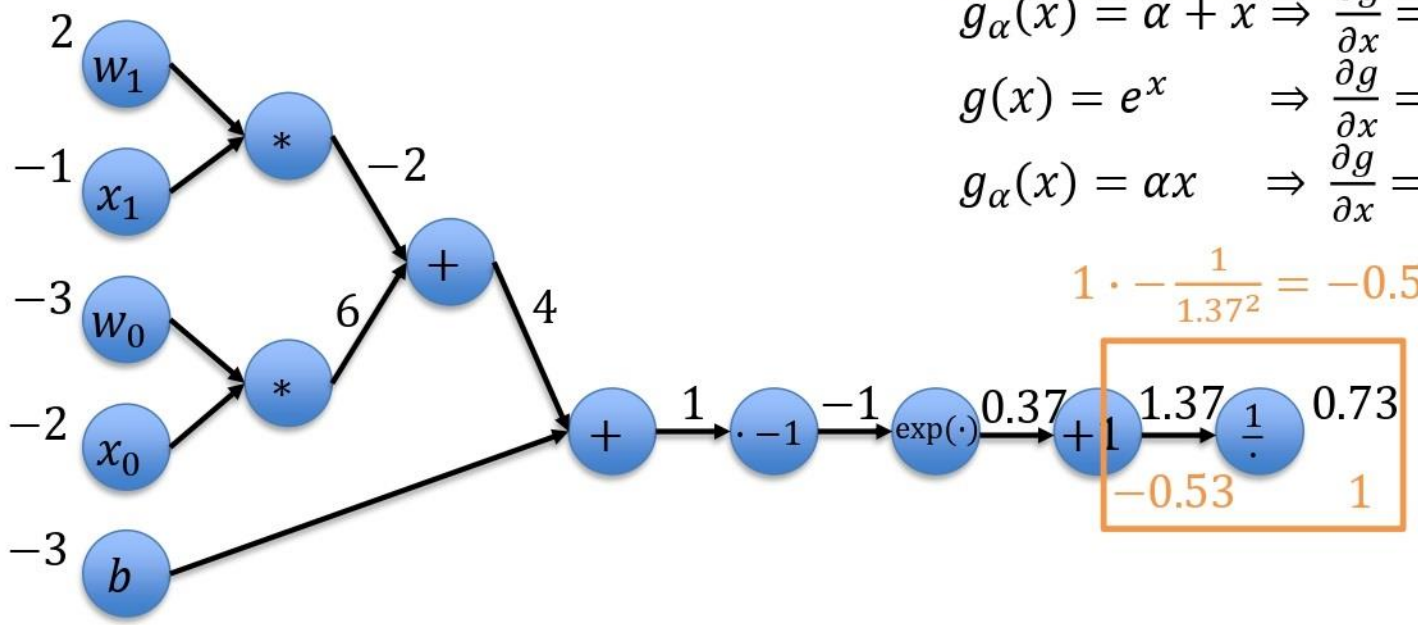
- $f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$



NNs as Computational Graphs

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

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



NNs as Computational Graphs

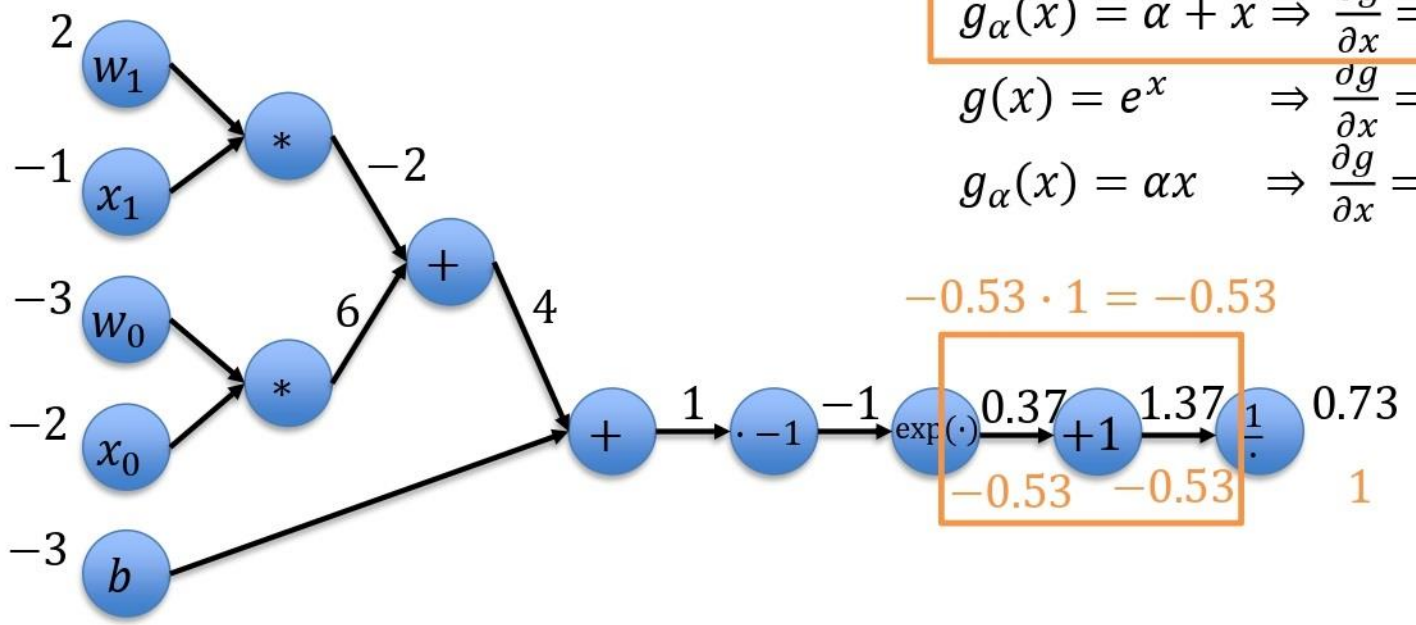
- $f(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1+e^{-(b+w_0x_0+w_1x_1)}}$

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

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

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

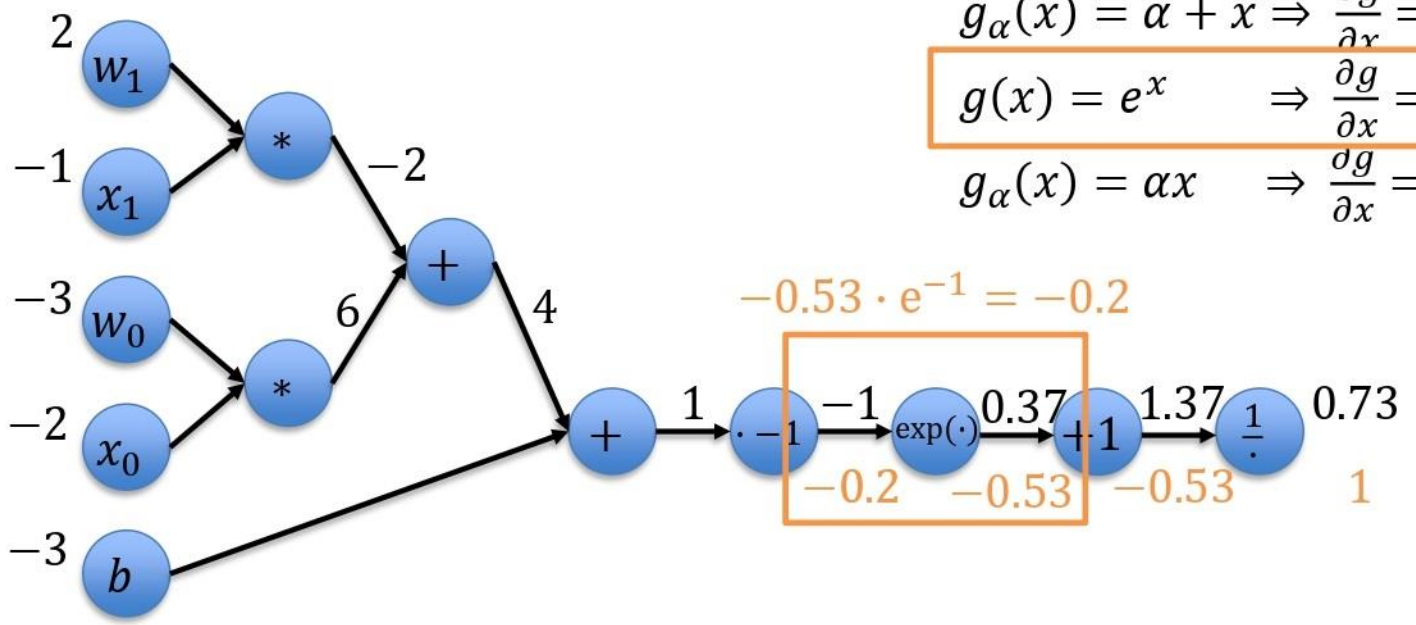
$$g_\alpha(x) = \alpha x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = \alpha$$



NNs as Computational Graphs

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

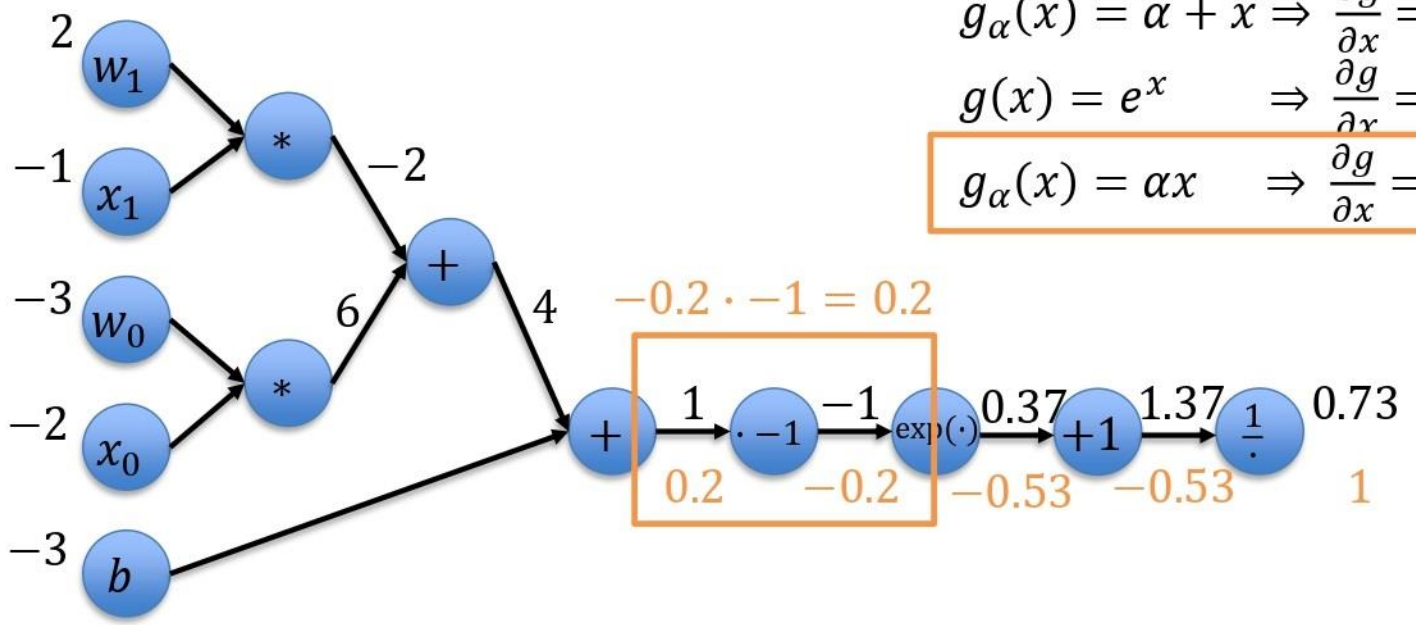
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$
$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$



NNs as Computational Graphs

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

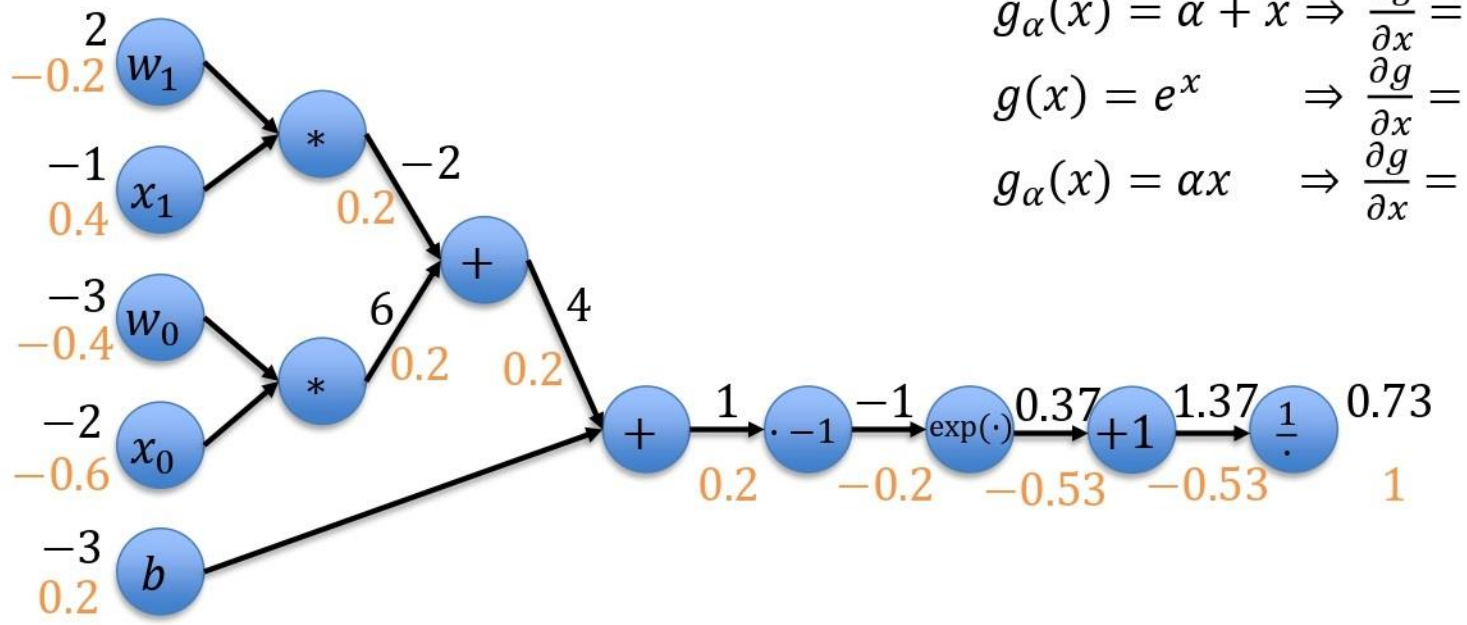
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$
$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$



NNs as Computational Graphs

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

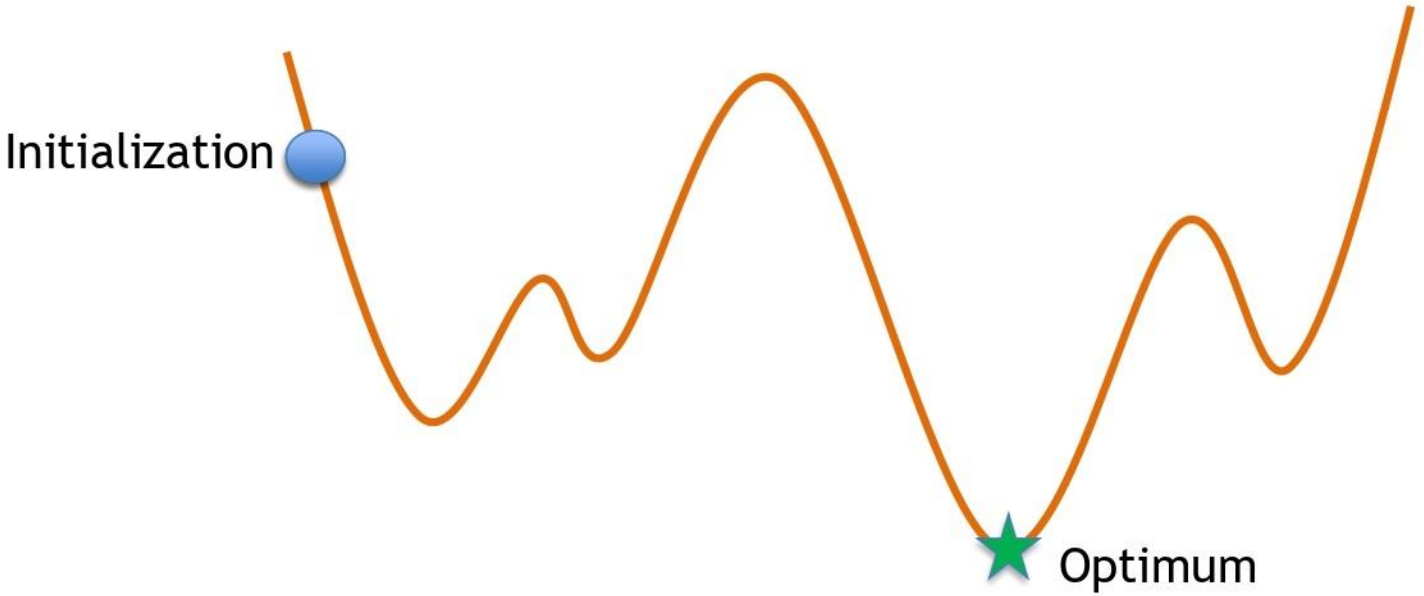
$$g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2}$$
$$g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g}{\partial x} = 1$$
$$g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x$$
$$g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha$$



Gradient Descent

Gradient Descent

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



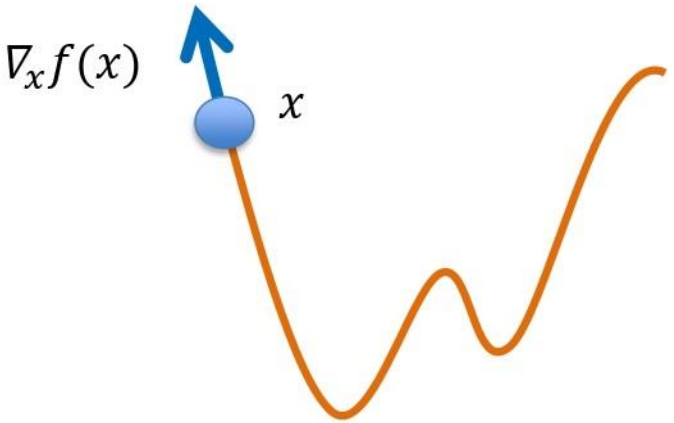
Gradient Descent

- From derivative to gradient

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

Direction of greatest increase of the function

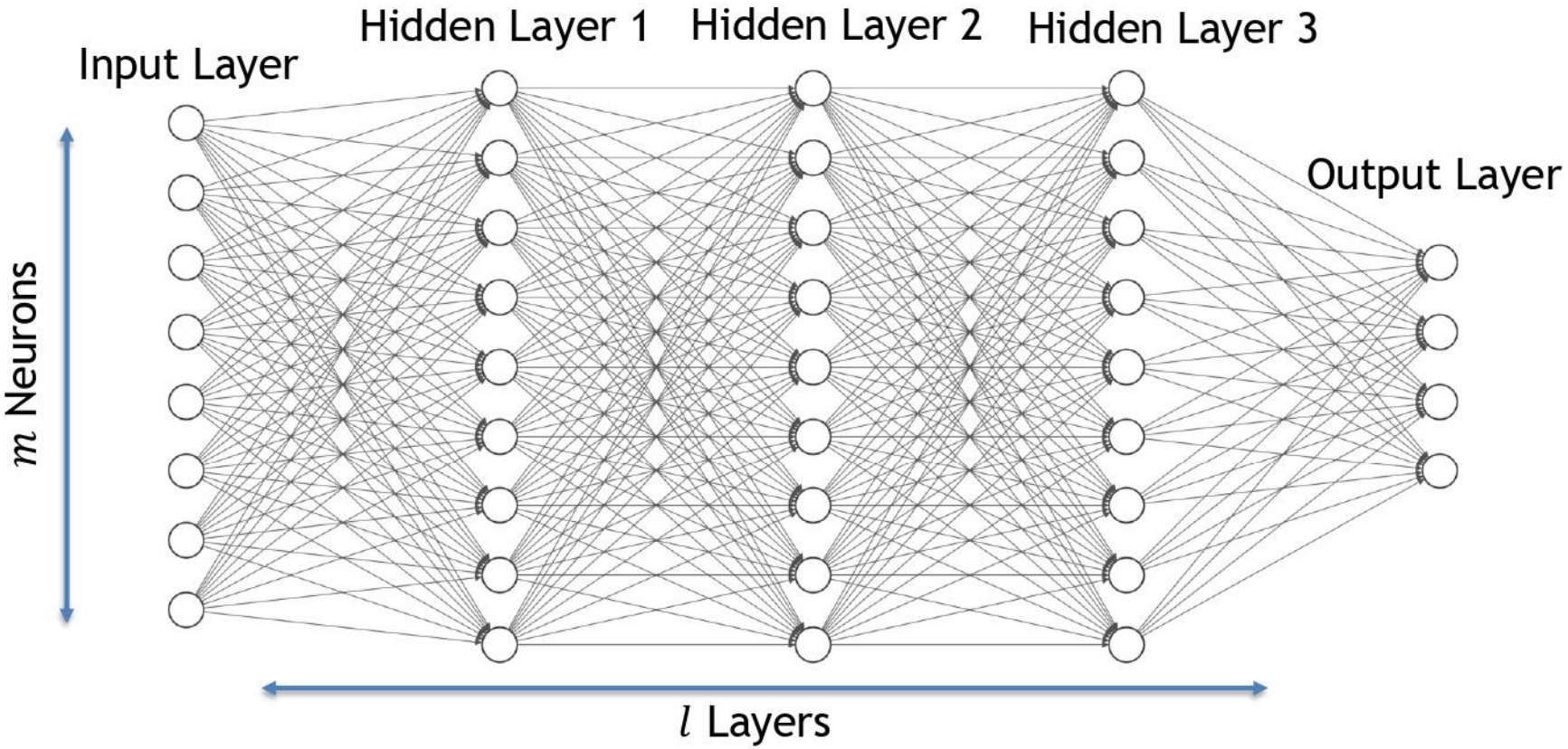
- Gradient steps in direction of negative gradient



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

Learning rate

Gradient Descent for Neural Networks



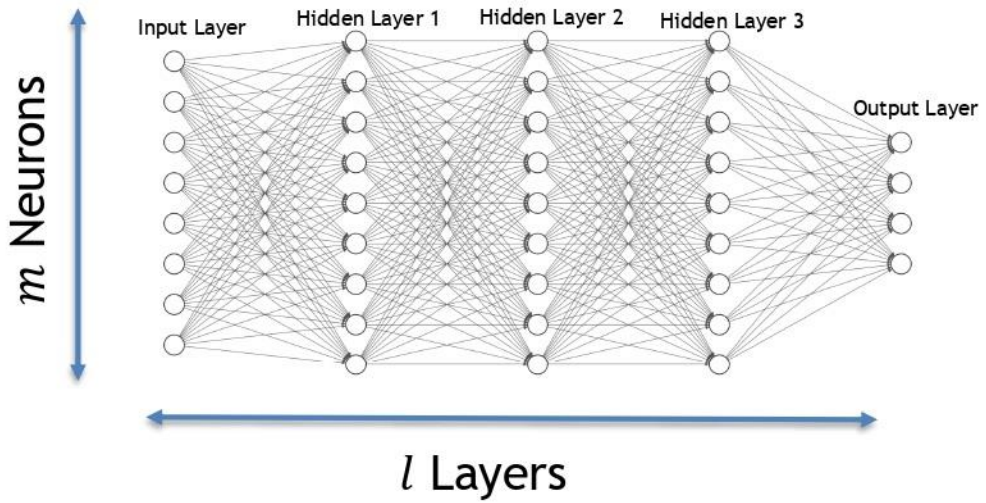
Gradient Descent for Neural Networks

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

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

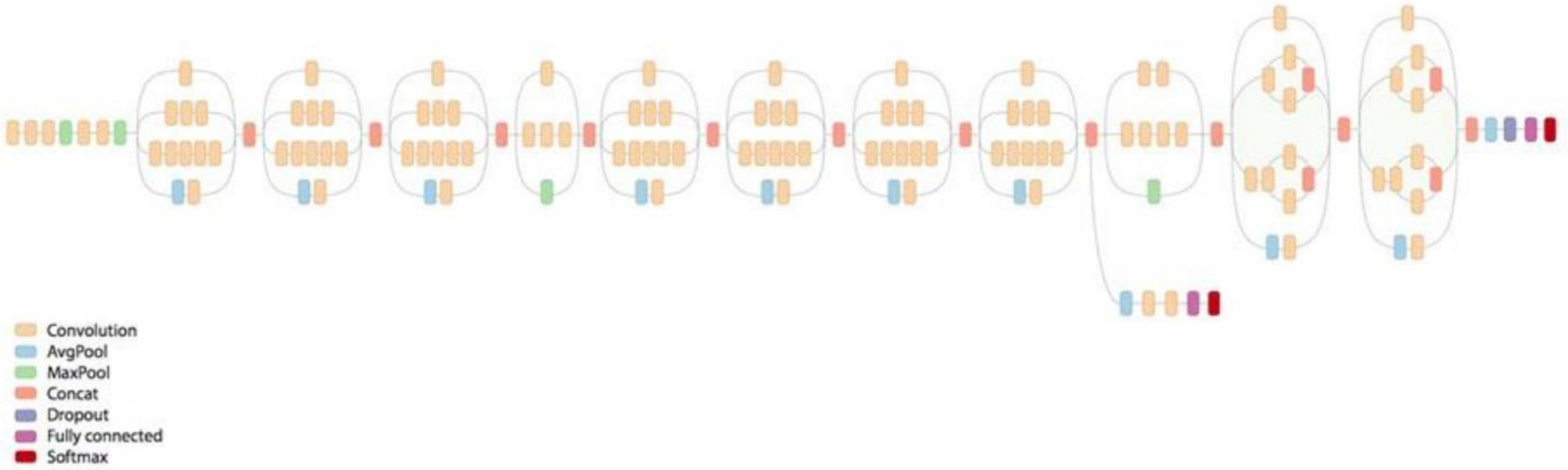
Gradient step:

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



NNs can Become Quite Complex…

- These graphs can be huge!



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

The Flow of the Gradients

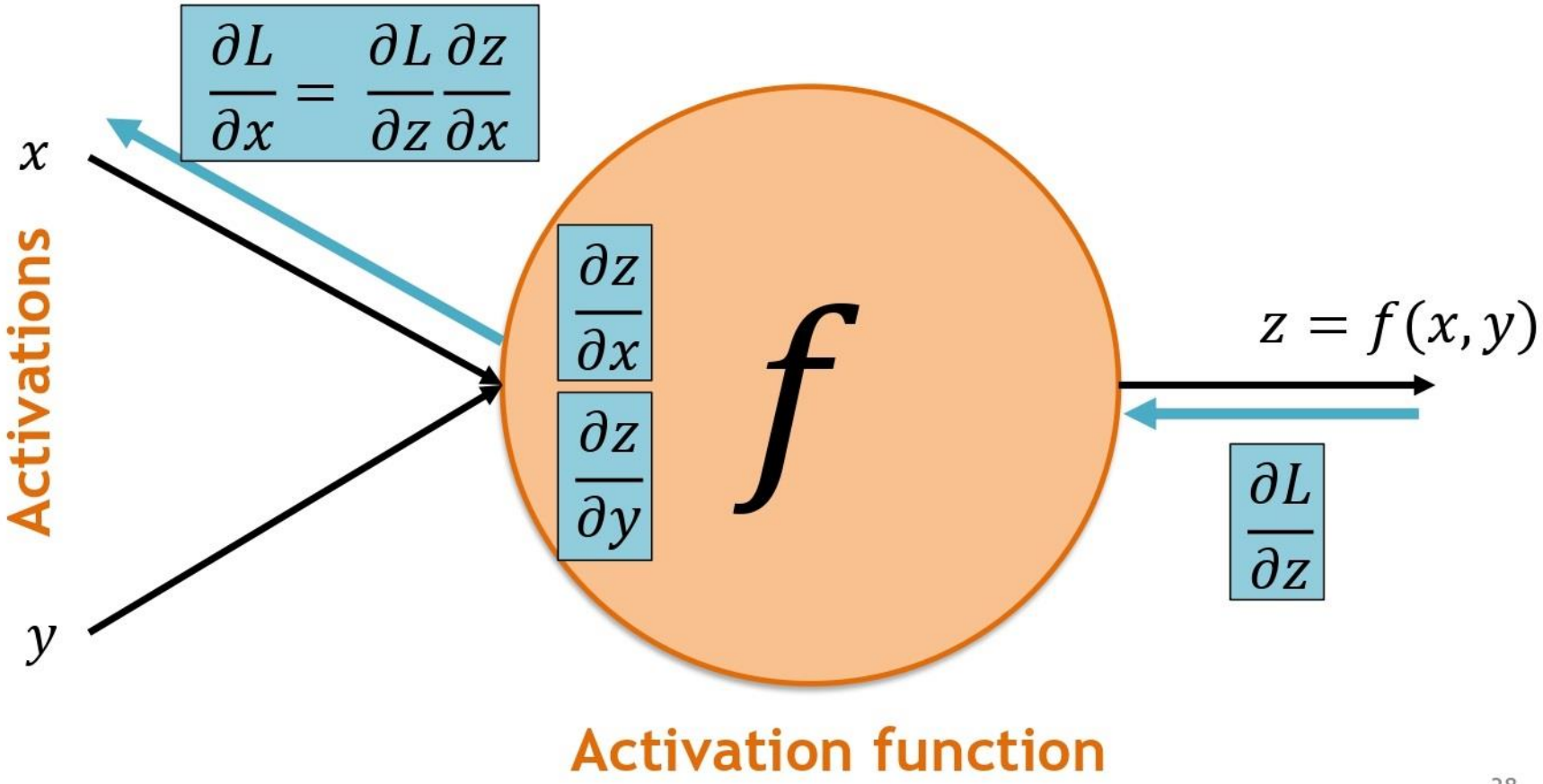
- Many many many many of these nodes form a neural network

NEURONS

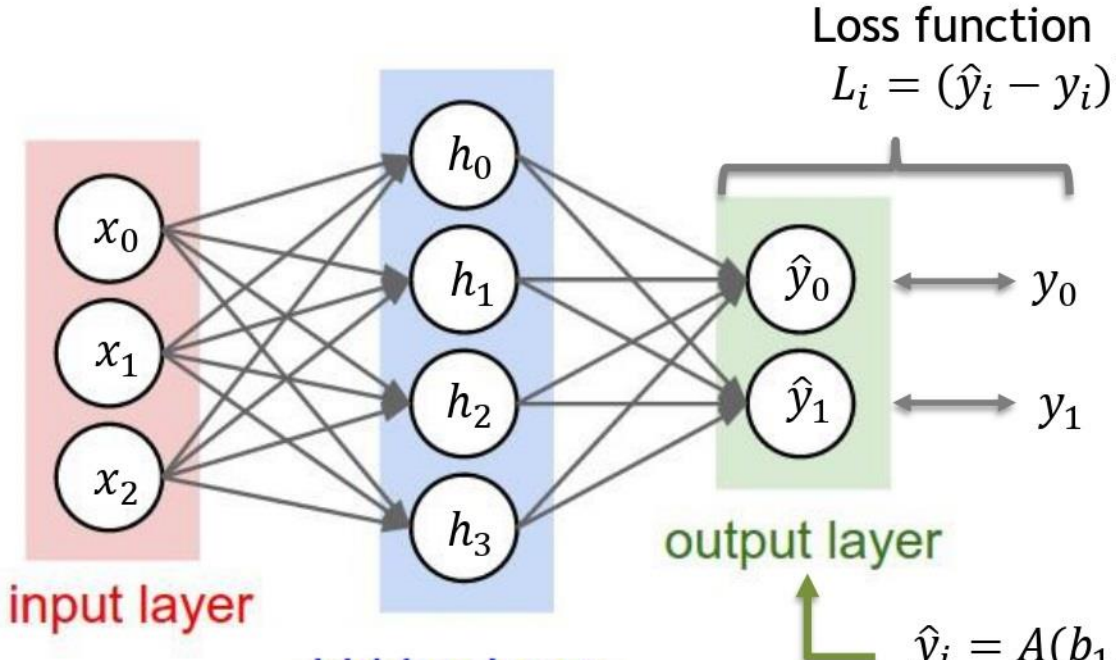
- Each one has its own work to do

FORWARD AND BACKWARD PASS

The Flow of the Gradients



Gradient Descent for Neural Networks

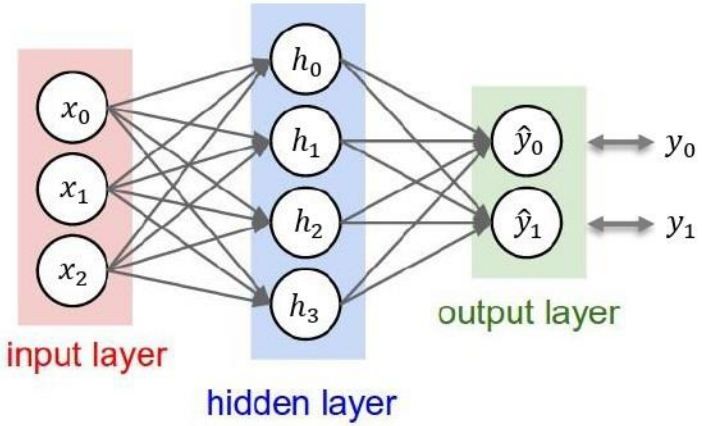


$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

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

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

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

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

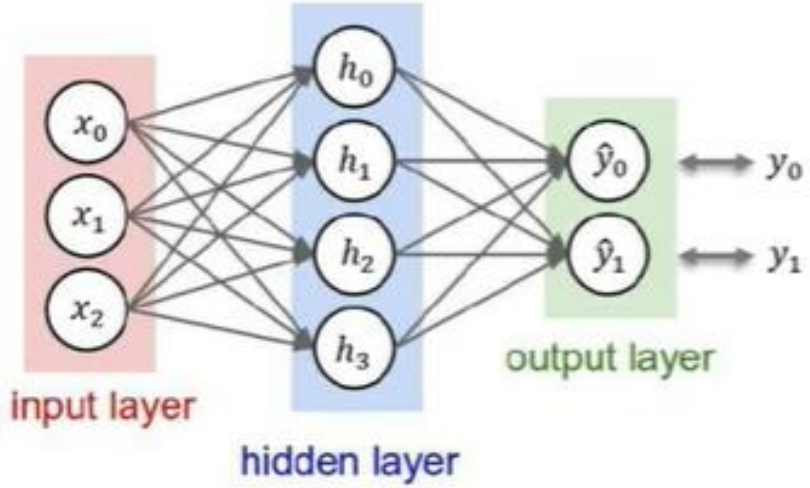
$$\frac{\partial L_i}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)$$

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

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

...

Gradient Descent for Neural Networks



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

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

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

How many unknown weights?

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

#neurons · #input channels + #biases

Note that some activations have also weights