Neural Networks

ECE 4200

General steps we do

- $S = \left\{\left(\bar{X}_1, y_1\right), \left(\bar{X}_2, y_2\right), \ldots, \left(\bar{X}_n, y_n\right)\right\}$
- Decide the algorithm/architecture to use (SVM/NN/LR).

Let $\boldsymbol{W}$ be the algorithm's parameters (weights)

$$A(\boldsymbol{W}, \bar{X}): prediction\ of\ \boldsymbol{W}\ on\ \bar{X}$$

- Decide the loss function (L2, hinge loss,L2+Reg, binary cross entropy)

$$L(\boldsymbol{W}, (\bar{X}_i, y_i)) = L(A(\boldsymbol{W}, \bar{X}_i,), y_i)$$

$$L(\boldsymbol{W}, S) = \sum_i L(\boldsymbol{W}, (\bar{X}_i, y_i))$$

- Find parameters that solve:

$$\min_{\boldsymbol{W}} L(\boldsymbol{W}, S)$$

Gradient descent

$$\min_{\boldsymbol{W}} L(\boldsymbol{W}, S)$$

This is hard to solve in most cases (no **closed form**).

$L(\boldsymbol{W}, S)$ is **just** a function of $\boldsymbol{W}$

In practice, use gradient descent (recall logistic regression).

- Start with some $\boldsymbol{W}(\mathbf{0})$
- Repeat until convergence:
  - Compute $\nabla L(\boldsymbol{W}, S)|_{\boldsymbol{w}(\boldsymbol{j})}$
  - $\boldsymbol{W}(\boldsymbol{j}+\mathbf{1}) = \boldsymbol{W}(\boldsymbol{j}) + \eta \cdot \nabla L(\boldsymbol{W}, S)|_{\boldsymbol{W}(\boldsymbol{j})}$

Pros, and cons.

Full gradient descent (GD)

Compute $\nabla L(\boldsymbol{W}, S)|_{\boldsymbol{W(j)}}$

$$\nabla L(\boldsymbol{W}, S)\Big|_{\boldsymbol{W(j)}} = \sum_i \nabla_{\boldsymbol{W}} L(\boldsymbol{W}, (\bar{X}_i, y_i))\Big|_{\boldsymbol{W(j)}}$$

Each step evaluates the gradients for n points and **adds them** up

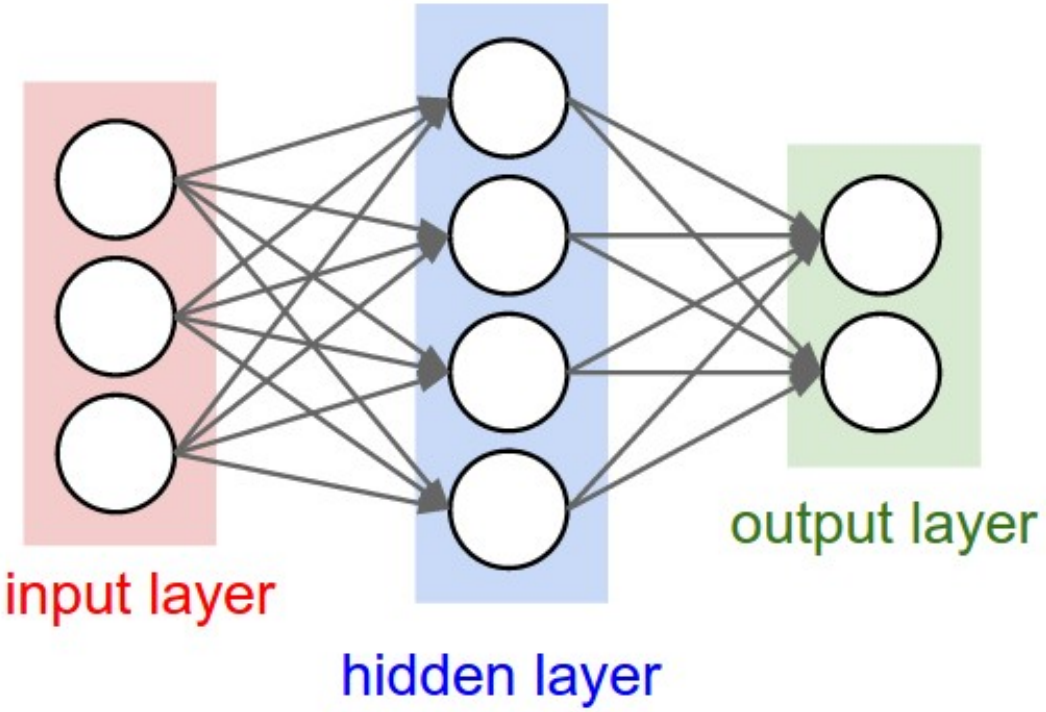
n can be very very large (Millions /Billions)

Full GD Prohibitive

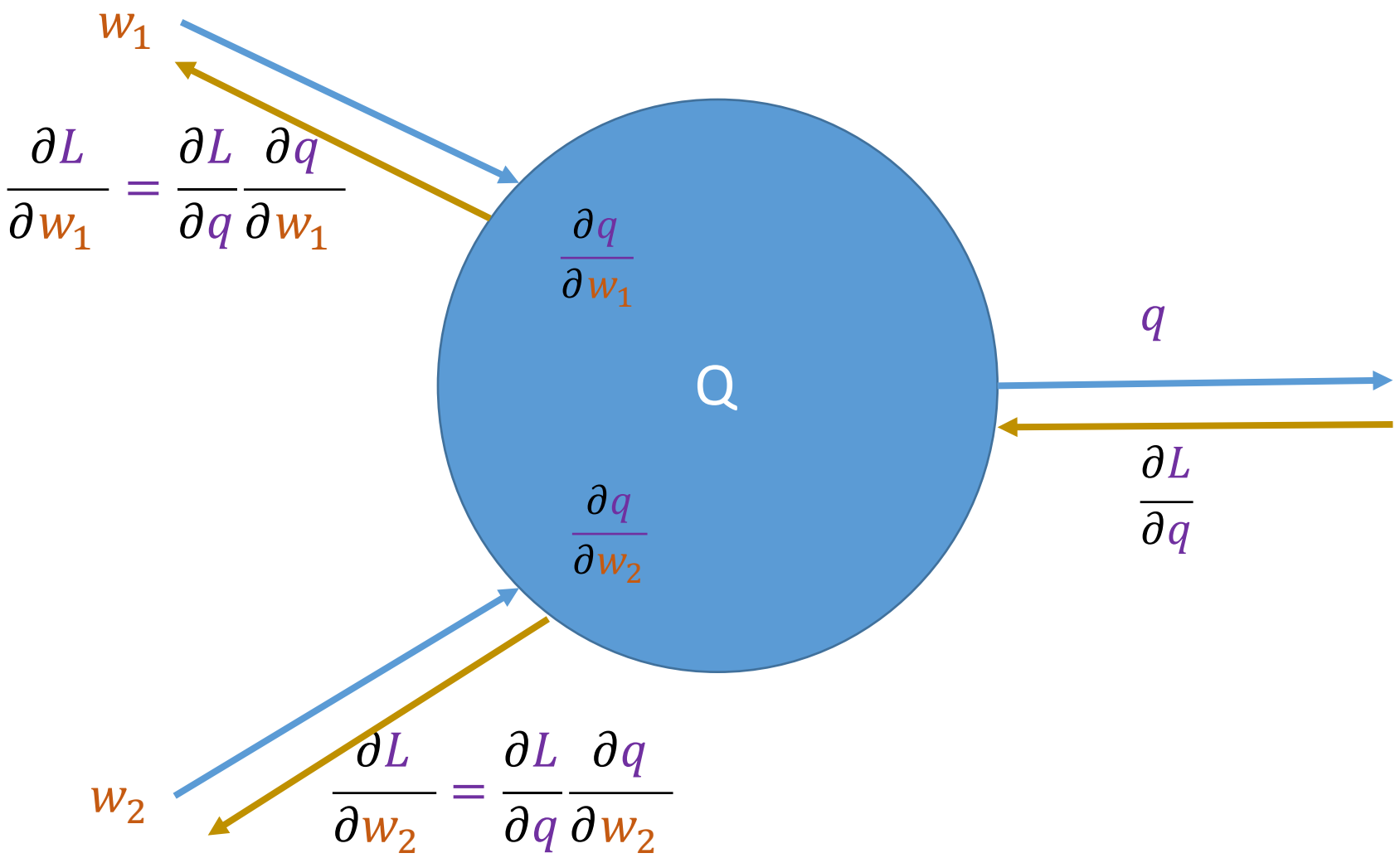
Neural Networks BP

Neural Networks

Large interconnected simple units



One Node


w_1
$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial q}\frac{\partial q}{\partial w_1}$
$\frac{\partial q}{\partial w_1}$
Q
$\frac{\partial q}{\partial w_2}$
q
$\frac{\partial L}{\partial q}$
w_2
$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial q}\frac{\partial q}{\partial w_2}$

Sigmoid activation

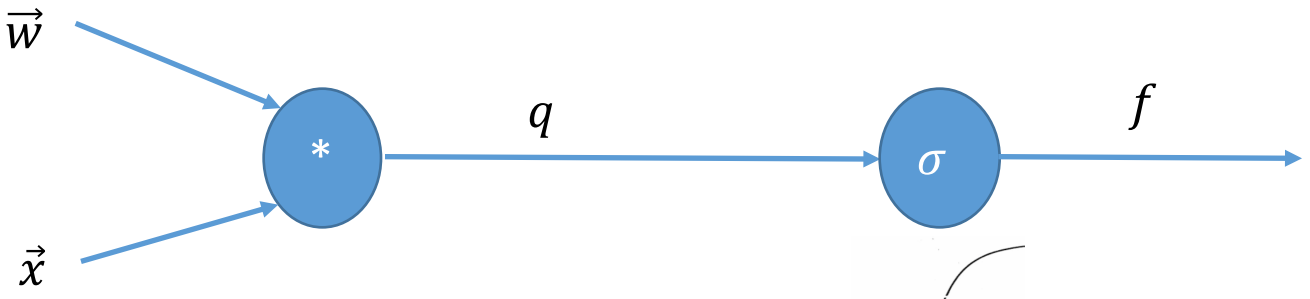
$$f(\bar{X}, \bar{w}) = \sigma(\bar{X} \cdot \bar{w}) = \frac{1}{1 + \exp(-(\bar{X} \cdot \bar{w}))}$$

One of the first and most popular activation functions.

Think of neurons:

Inputs are $\vec{x}$,

Weights are $\vec{w}$



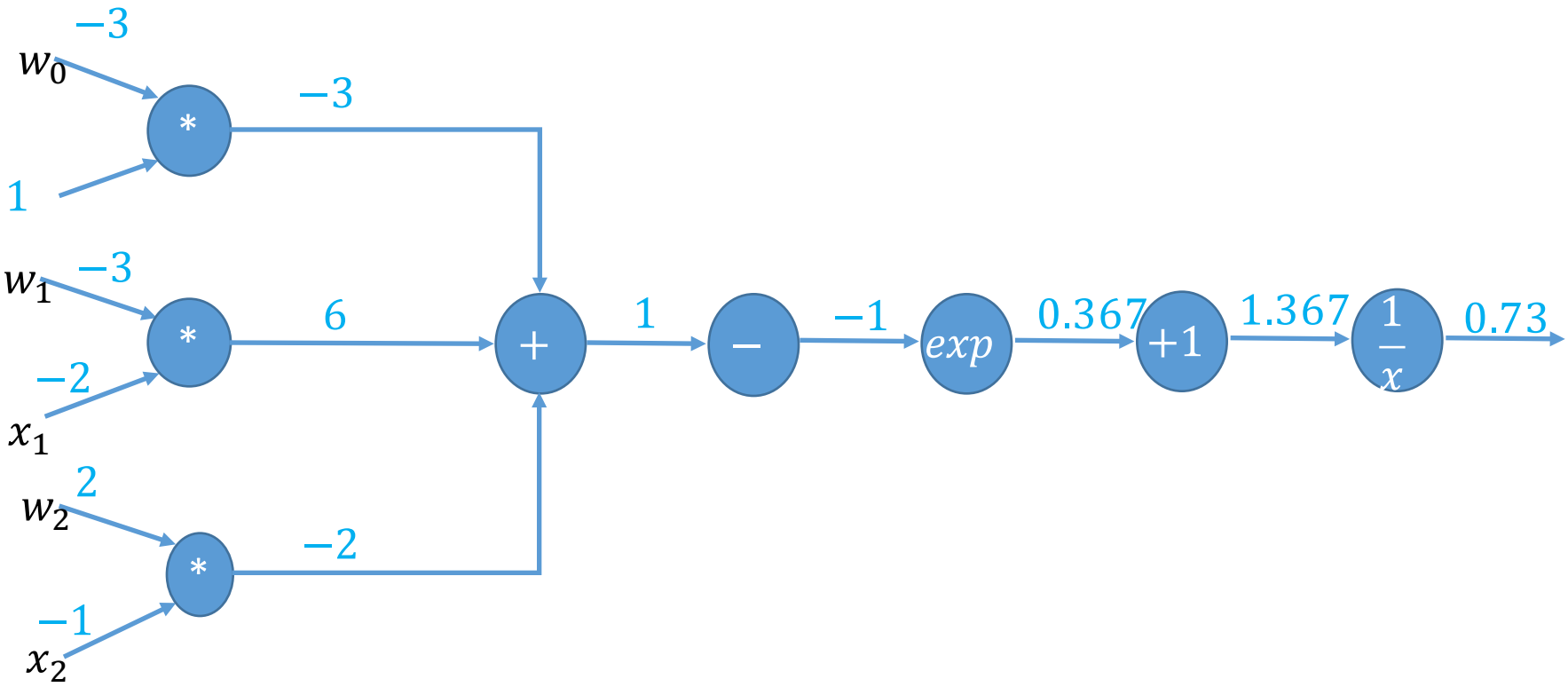
Linear combination of weights and inputs is activation fn

$f(\vec{x}, \vec{w})$: the **probability** that the **neuron fires**

Sigmoid activation

$d=3, \bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$

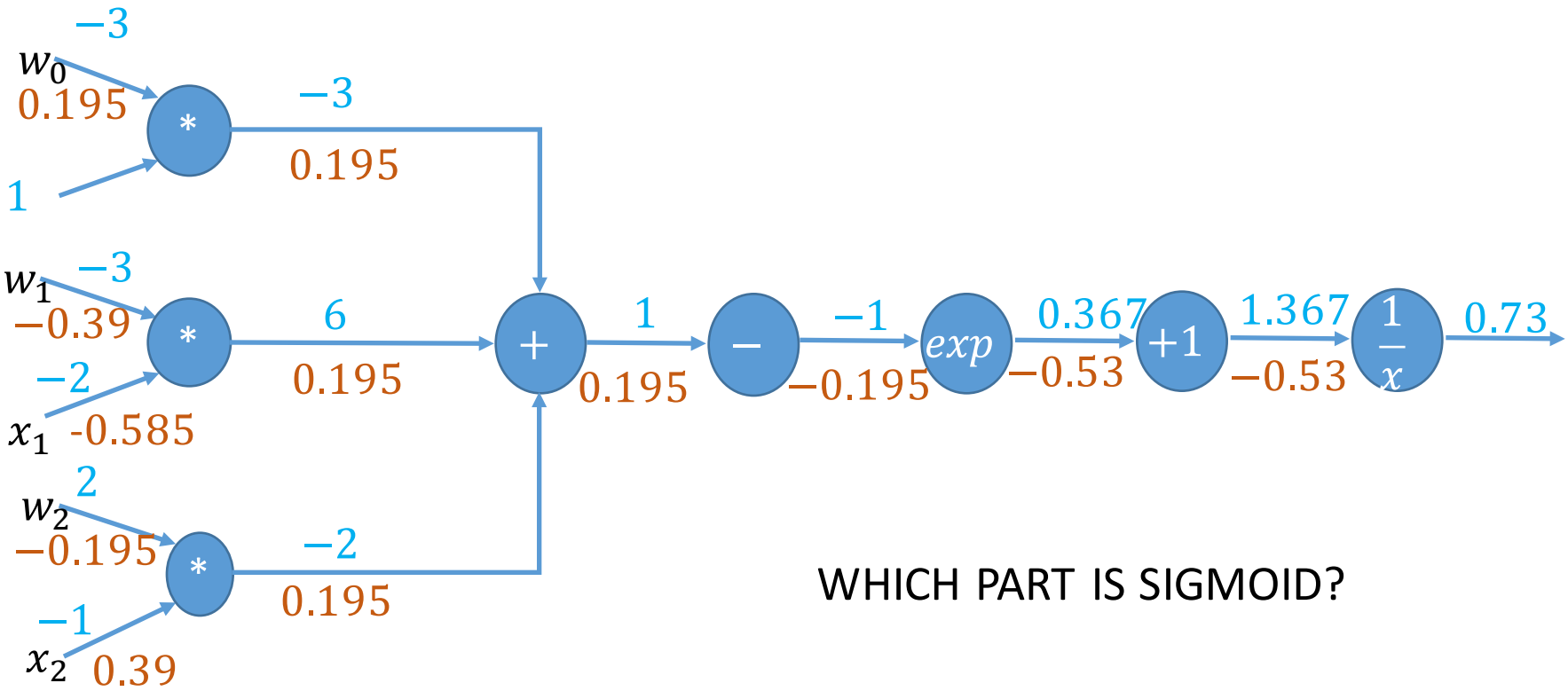
$$f(\vec{x}, \vec{w}) = \frac{1}{1 + \exp(-(w_0 + w_1 x_1 + w_2 x_2))}$$



Sigmoid activation

$d=3, \bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$

$$f\left(\bar{X}, \bar{w}\right) = \frac{1}{1 + \exp(-(w_0 + w_1 x_1 + w_2 x_2))}$$

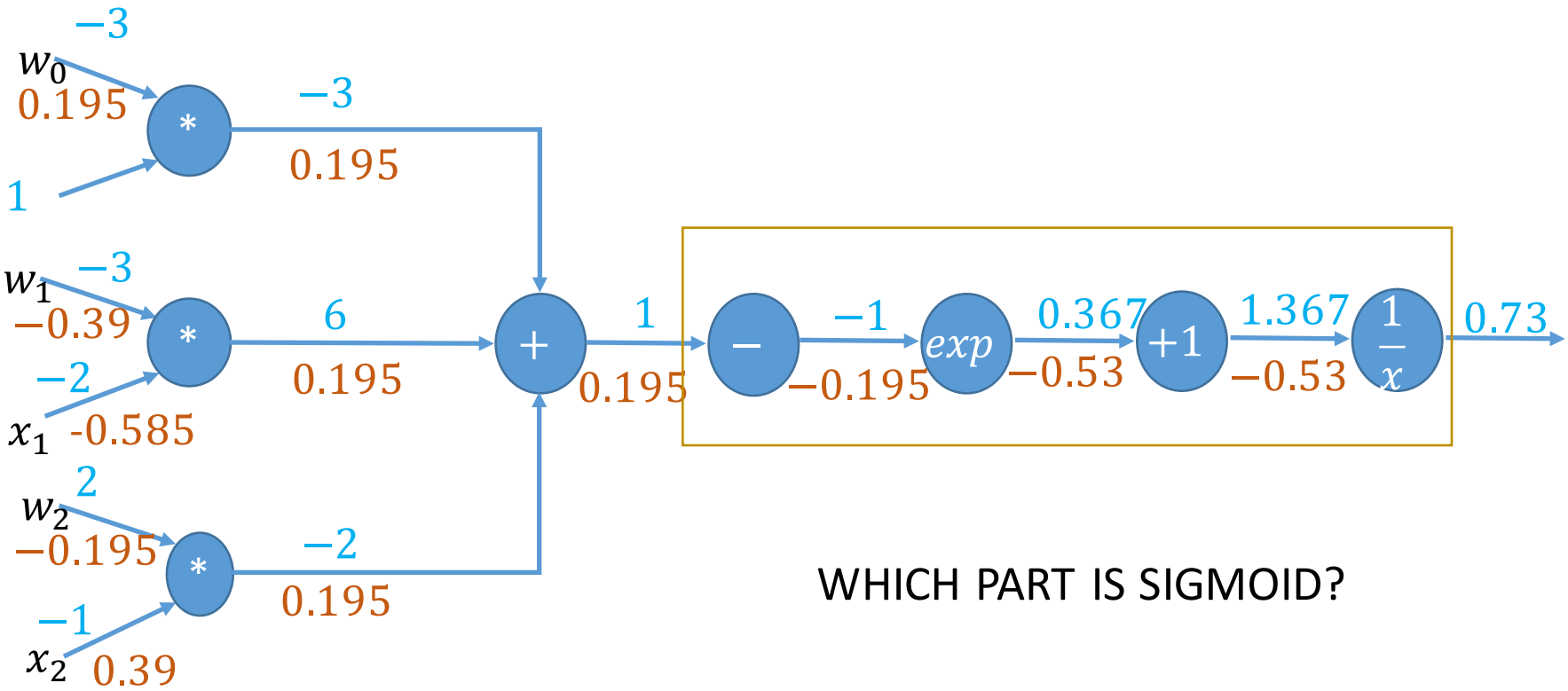


WHICH PART IS SIGMOID?

Sigmoid activation

$$\bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$$

$$f\left(\bar{X}, \bar{w}\right) = \frac{1}{1 + \exp(-(w_0 + w_1 x_1 + w_2 x_2))}$$

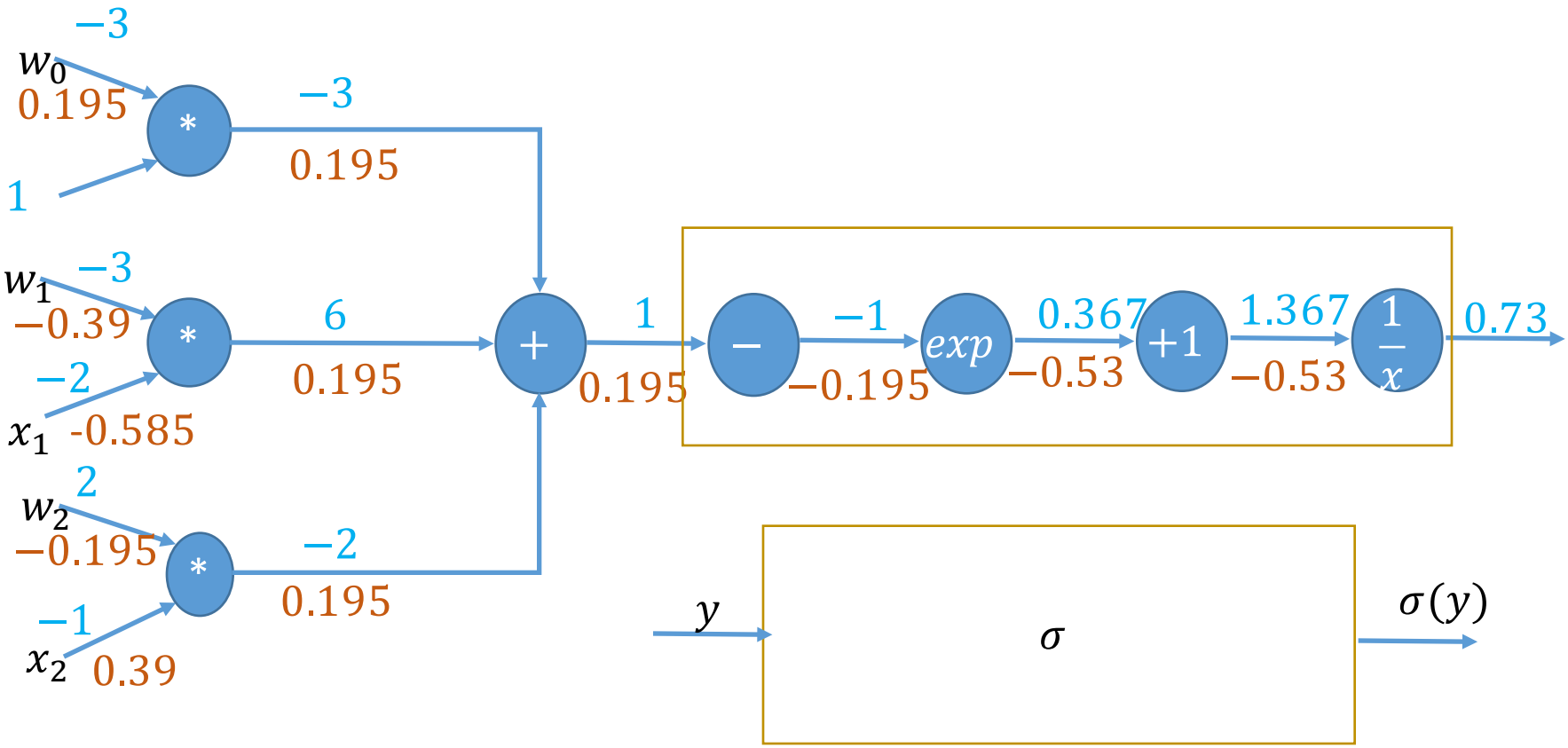


WHICH PART IS SIGMOID?

Sigmoid activation

$$\bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$$

$$f\left(\bar{X}, \bar{w}\right) = \frac{1}{1 + \exp(-(w_0 + w_1 x_1 + w_2 x_2))}$$



Derivative of sigmoid

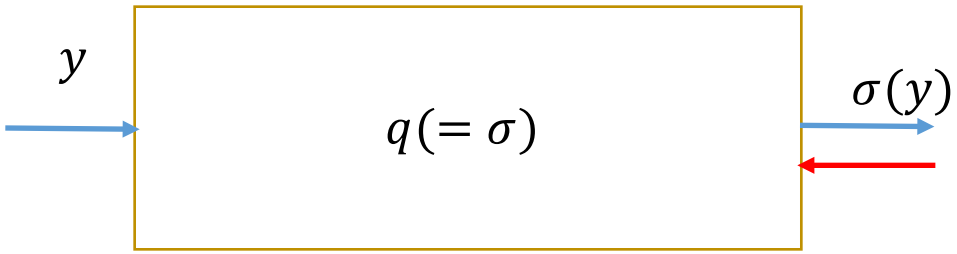
$$\sigma(y) = \frac{1}{(1+e^{-y})}, 1-\sigma(y) = \frac{e^{-y}}{(1+e^{-y})}$$

$$\frac{d\sigma(y)}{dy} = \frac{d(1+e^{-y})^{-1}}{dy} = \frac{-1}{(1+e^{-y})^2}\frac{d(1+e^{-y})}{dy} = \frac{e^{-y}}{(1+e^{-y})^2}$$

$$\frac{d\sigma(y)}{dy} = \sigma(y)\cdot\big(1-\sigma(y)\big).$$

Backpropagation for Sigmoid!

Sigmoid block



$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial q}\frac{\partial q}{\partial y}$$

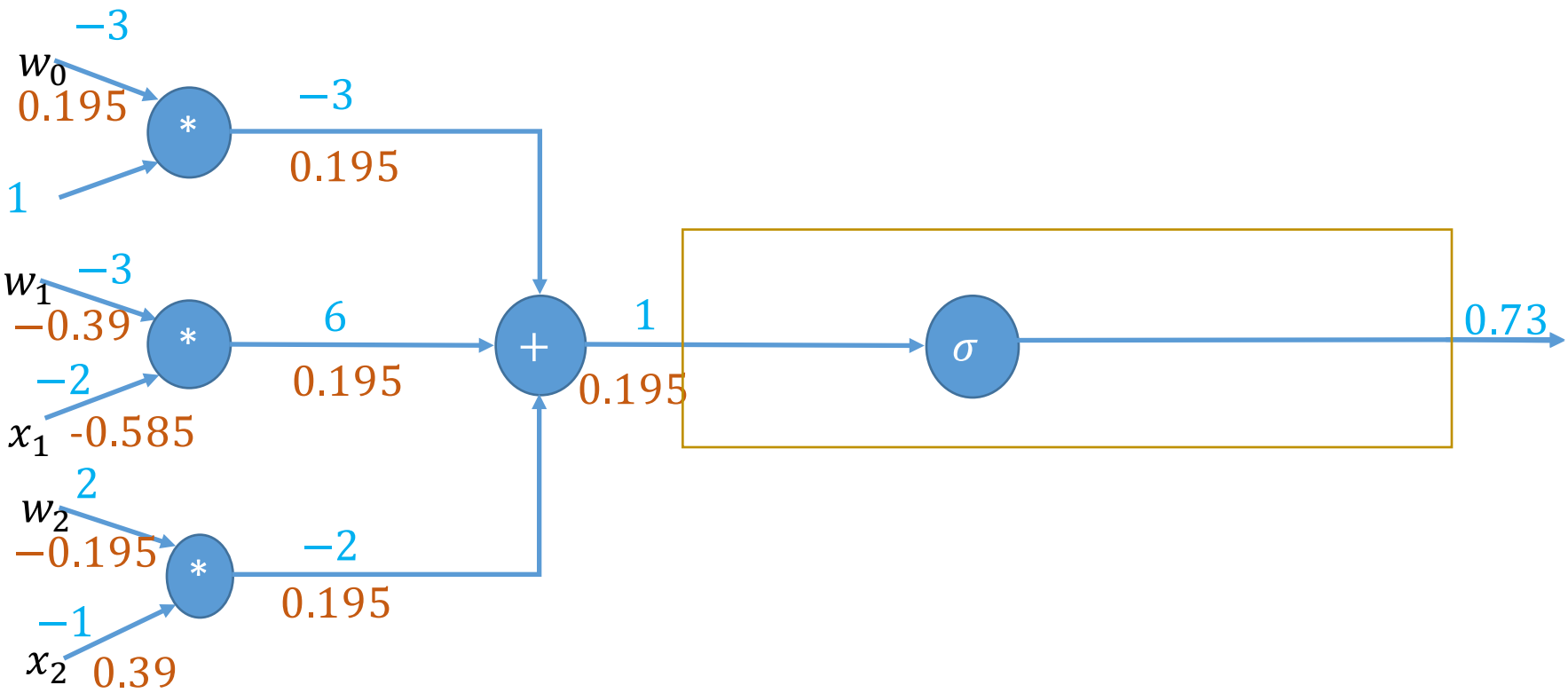
$\frac{\partial L}{\partial q}$: comes from next layer

$$\frac{\partial q}{\partial y} = \sigma(y) \cdot \big(1 - \sigma(y)\big)$$

Sigmoid activation

$\bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$

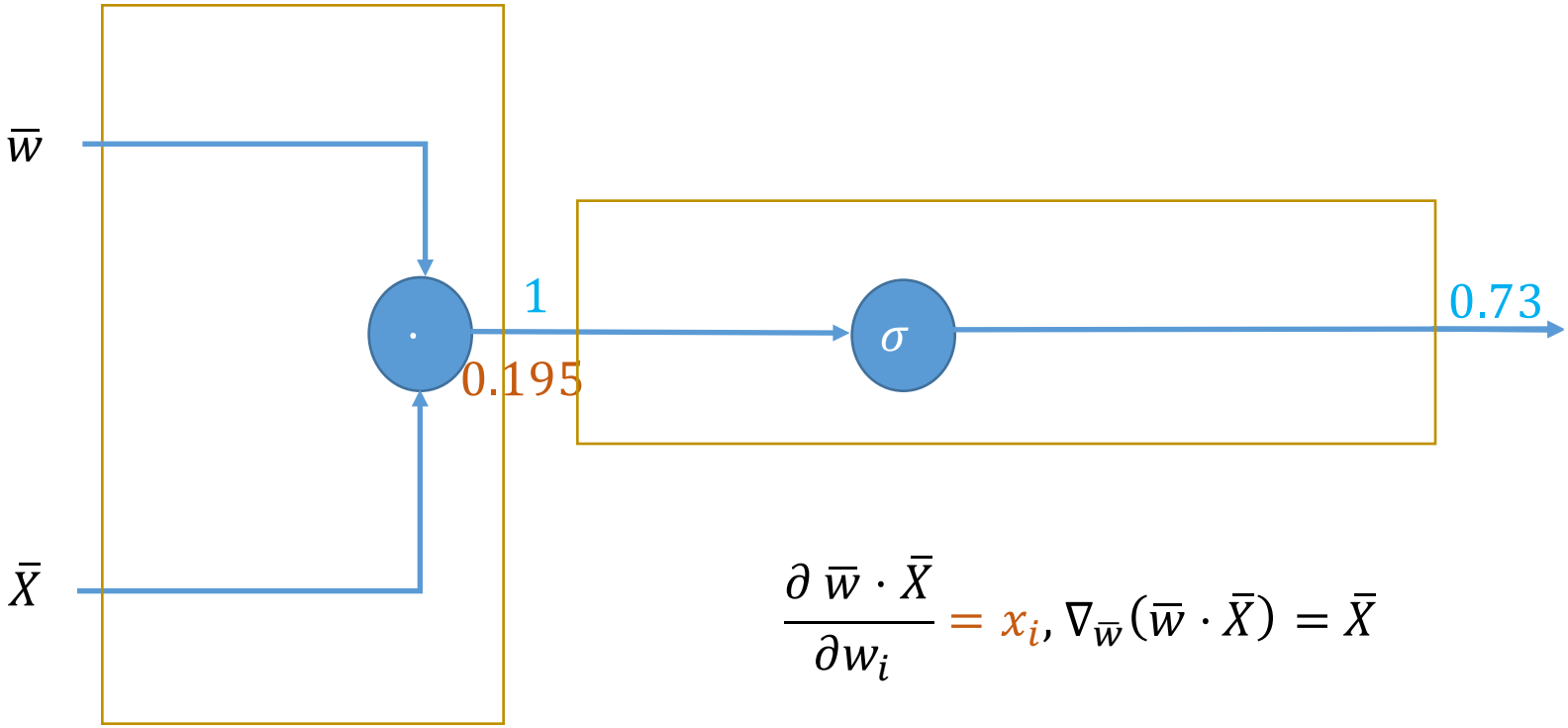
$$f\left(\bar{X}, \bar{w}\right) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$



Sigmoid activation

$\bar{w} = (w_0, w_1, w_2), \bar{X} = (1, x_1, x_2).$

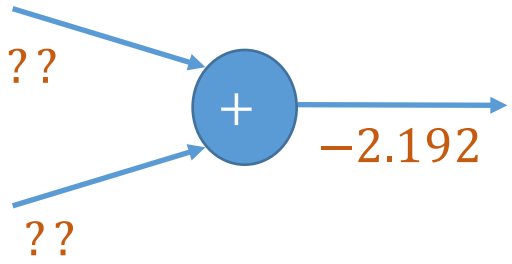
$$f\left(\bar{X}, \bar{w}\right) = \sigma(\bar{w} \cdot \bar{X})$$



$$\frac{\partial\, \bar{w} \cdot \bar{X}}{\partial w_i} = x_i, \nabla_{\bar{w}}(\bar{w} \cdot \bar{X}) = \bar{X}$$

Derivatives for different gates

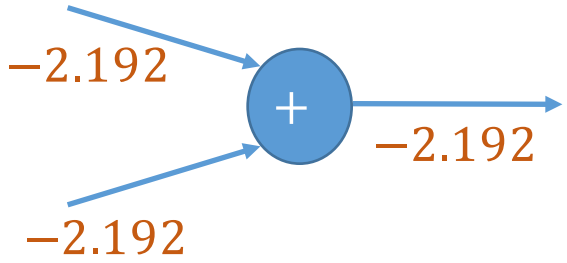
Add gate:



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial q}\frac{\partial q}{\partial w}$$

Derivatives for different gates

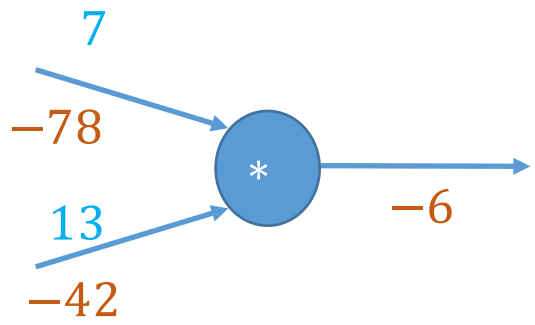
Add gate: gradient distributor



Derivatives for different gates

Add gate: gradient distributor

Mul gate: gradient switcher

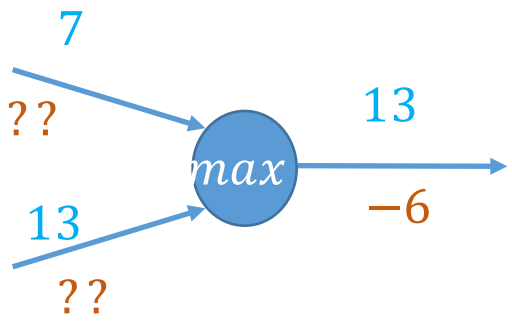


Derivatives for different gates

Add gate: gradient distributor

Mul gate: gradient switcher

Max gate:

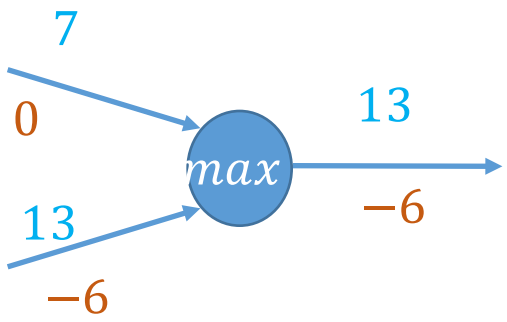


Derivatives for different gates

Add gate: gradient distributor

Mul gate: gradient switcher

Max gate:



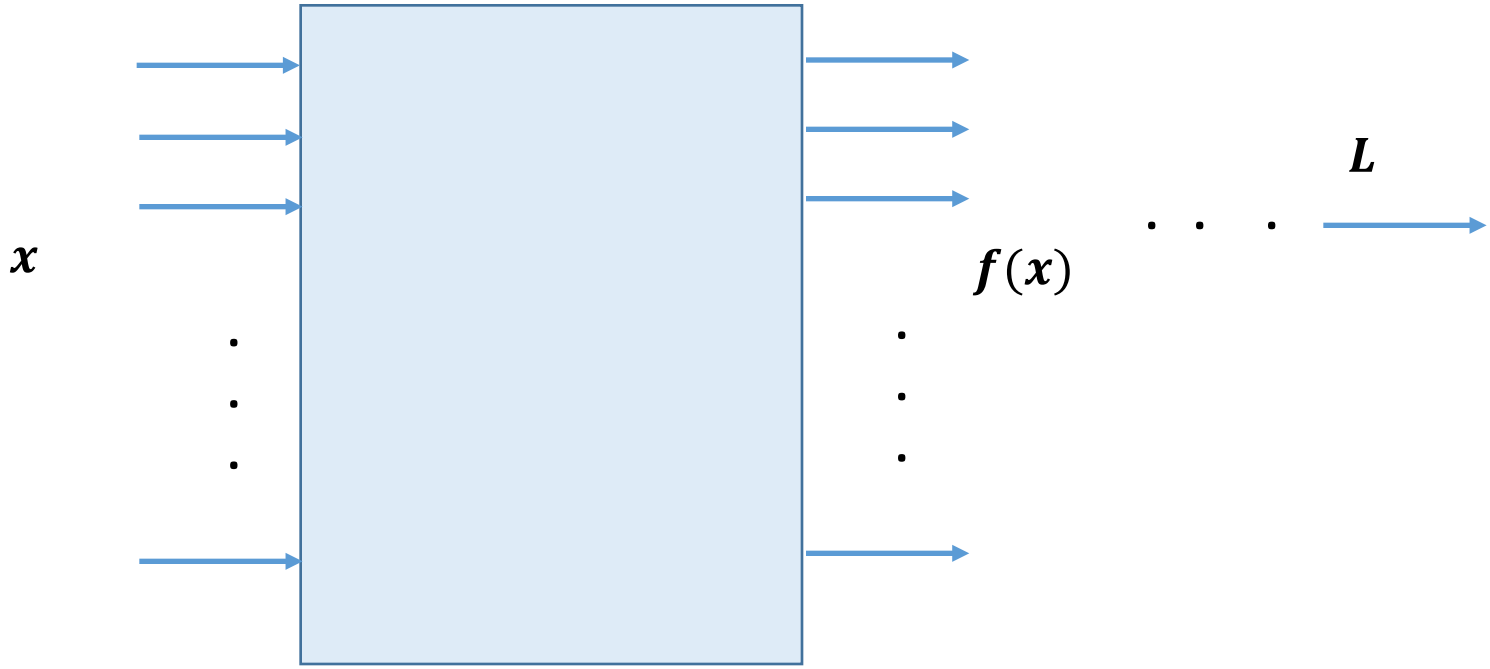
Plan for today

- Vector Outputs (Jacobian)
- Activation Functions
- Training NNs: Stochastic gradient descent, mini batch training
- Pre-processing
- CNN (just a primer)
- Loss Functions

Vector Outputs

Vector Outputs

Consider an operation/function of the form:



Functions $\boldsymbol{f}: \mathrm{R}^{\mathrm{d}} \rightarrow \mathrm{R}^{m}$, and L is real valued loss

Vector Outputs

Until now, $m = 1$, namely f was a scalar.

Chain rule:

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial \boldsymbol{x}}$$

The left hand side is actually the gradient, namely $\nabla_{\boldsymbol{x}} L$.

How to compute **gradient** $\nabla_{\boldsymbol{x}} L$ when f is vector valued?

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial \boldsymbol{f}} \cdot \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$$

Vector Outputs

Until now, $m = 1$, namely f was a scalar.

Chain rule:

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial \boldsymbol{x}}$$

The left hand side is actually the gradient, namely $\nabla_{\boldsymbol{x}} L$.

How to compute **gradient** $\nabla_{\boldsymbol{x}} L$ when $\boldsymbol{f} \in \mathrm{R}^m$ is vector valued?

chain rule:

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{f}}$$

What are dimensions of these objects?

Jacobian

$\frac{\partial L}{\partial \boldsymbol{f}}$:Gradient w.r.t $\boldsymbol{f}$

Dimension of $\frac{\partial L}{\partial \boldsymbol{f}}$?

$m \times 1$

$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$: Jacobian matrix

Dimension of $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$?

$d \times m$

Jacobian Matrix

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} := \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_d} & \cdots & \frac{\partial f_m}{\partial x_d} \end{bmatrix}$$

Dimension of $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$?

$$d \times m$$

Jacobian Matrix

You can also think in terms of the following equation:

$$\frac{\partial L}{\partial x_1} = \sum_{i=1}^{m} \frac{\partial L}{\partial f_i} \cdot \frac{\partial f_i}{\partial x_1}$$

Each entry comes from one entry in the matrix vector product.

Jacobian Matrix

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \\ \vdots \\ \frac{\partial L}{\partial x_d} \end{bmatrix} \coloneqq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_d} & \cdots & \frac{\partial f_m}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial f_1} \\ \frac{\partial L}{\partial f_2} \\ \vdots \\ \frac{\partial L}{\partial f_m} \end{bmatrix}$$

Dimensionality and computation?

Hopefully some structure present

Jacobian for ReLU

Example.

$\boldsymbol{f}: \mathrm{R}^d \rightarrow \mathrm{R}^d$, element wise thresholding.

$$\boldsymbol{f(x) = max\{0, x\}}$$

What is the complexity/dimensionality of: $\frac{\partial f}{\partial x}$

What is $\frac{\partial f}{\partial x}$*?*

This function forms a crucial part of **ReLU networks**

Linear Transformations

For a $m \times d$ matrix W, and $\boldsymbol{x} \in \mathrm{R}^d$

$$\boldsymbol{L}(\boldsymbol{x}, W) = \| W \cdot \boldsymbol{x} \|_2^2$$

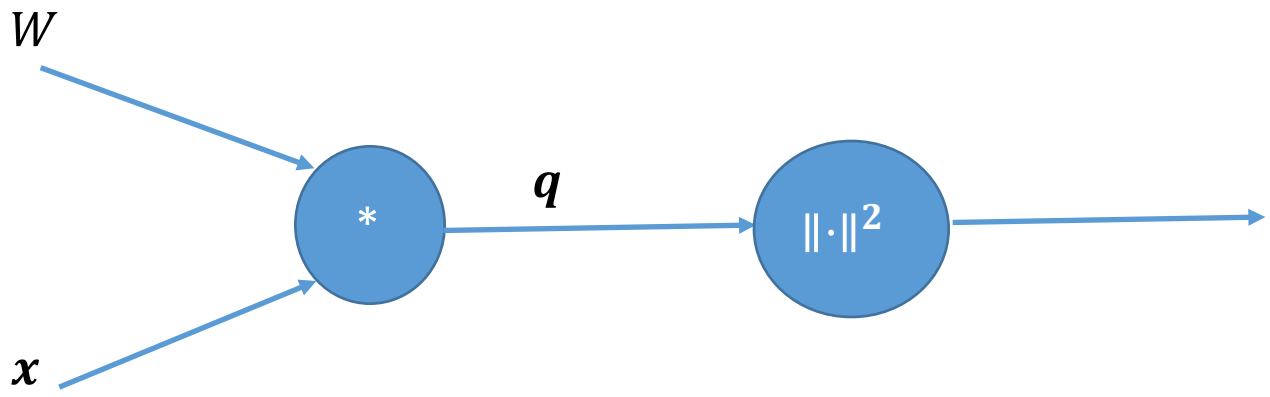
Computational graph for this?

Linear Transformations

For a $m \times d$ matrix W, and $\boldsymbol{x} \in \mathrm{R}^d$

$$\boldsymbol{L}(\boldsymbol{x}, W) = \| W \cdot \boldsymbol{x} \|_2^2$$

Computational graph for this?



Linear Transformations

$$\boldsymbol{L}(\boldsymbol{x}, W) = \| W \cdot \boldsymbol{x} \|_2^2$$

$$W := \begin{bmatrix} W_{11} & \cdots & W_{1d} \\ \vdots & \ddots & \vdots \\ W_{m1} & \cdots & W_{md} \end{bmatrix}, \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_d \end{bmatrix}$$

$$\boldsymbol{q} = W \cdot \boldsymbol{x} = \begin{bmatrix} W_{11}x_1 + \cdots + W_{1d}x_d \\ \cdot \\ \cdot \\ \cdot \\ W_{m1}x_1 + \cdots + W_{md}x_d \end{bmatrix}$$

$$\boldsymbol{L}(\boldsymbol{x}, W) = \sum_i \boldsymbol{q}_i^2 = \boldsymbol{q}^T \boldsymbol{q} = \sum_i (W_{i1}x_1 + \cdots + W_{id}x_d)^2$$

Computing gradients

$$\boldsymbol{L}(\boldsymbol{x}, W) = \sum_i \boldsymbol{q}_i^2 = \boldsymbol{q}^T \boldsymbol{q} = \sum_i (W_{i1} x_1 + \cdots + W_{id} x_d)^2$$

$$\frac{\partial(\boldsymbol{q}_1^2 + \cdots + \boldsymbol{q}_m^2)}{\partial \boldsymbol{q}_i} = 2\boldsymbol{q}_i^2$$

$$\boldsymbol{\nabla_q L}(\boldsymbol{x}, W) = \boldsymbol{\nabla_q}(\boldsymbol{q}^T \boldsymbol{q}) = \boldsymbol{2q}.$$

Gradient dimension always matches what gradient is w.r.t.

Computing gradients

$$\boldsymbol{L}(\boldsymbol{x}, W) = \sum_i \boldsymbol{q}_i^2 = \boldsymbol{q}^T \boldsymbol{q} = \sum_i (W_{i1} x_1 + \cdots + W_{id} x_d)^2$$

$$\frac{\partial \boldsymbol{L}(\boldsymbol{x}, W)}{\partial \boldsymbol{x_j}} = \frac{\partial \sum_i \boldsymbol{q}_i^2}{\partial \boldsymbol{x}_j} = \sum_i 2\boldsymbol{q}_i \frac{\partial \boldsymbol{q}_i}{\partial \boldsymbol{x}_j} = \sum_i 2\boldsymbol{q}_i W_{ij}$$

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \boldsymbol{L}(\boldsymbol{x}, W) = \mathbf{2} W^T \boldsymbol{q}.$$

Do it diffferently: $\boldsymbol{L}(\boldsymbol{x}, W) = \boldsymbol{q}^T \boldsymbol{q} = \boldsymbol{x}^T W^T W \boldsymbol{x}$

Gradient w.r.t. $\boldsymbol{x}$ *will be* $2W^T W \boldsymbol{x} = 2W^T \boldsymbol{q}$

Computing gradients

$$\boldsymbol{L}(\boldsymbol{x}, W) = \sum_i \boldsymbol{q}_i^2 = \boldsymbol{q}^T \boldsymbol{q} = \sum_i (W_{i1} x_1 + \cdots + W_{id} x_d)^2$$

$$\frac{\partial \boldsymbol{L}(\boldsymbol{x}, W)}{\partial W_{jk}} = \frac{\partial \sum_i \boldsymbol{q}_i^2}{\partial W_{jk}} = \sum_i 2\boldsymbol{q}_i \frac{\partial \boldsymbol{q}_i}{\partial W_{jk}} = 2\boldsymbol{q}_j \boldsymbol{x}_k$$

$$\boldsymbol{\nabla}_W \boldsymbol{L}(\boldsymbol{x}, W) = \mathbf{2}\boldsymbol{q}\boldsymbol{x}^T.$$

Gradient dimension always matches what gradient is w.r.t.

Do it diffferently: $\boldsymbol{L}(\boldsymbol{x}, W) = \boldsymbol{q}^T \boldsymbol{q} = \boldsymbol{x}^T W^T W \boldsymbol{x}$

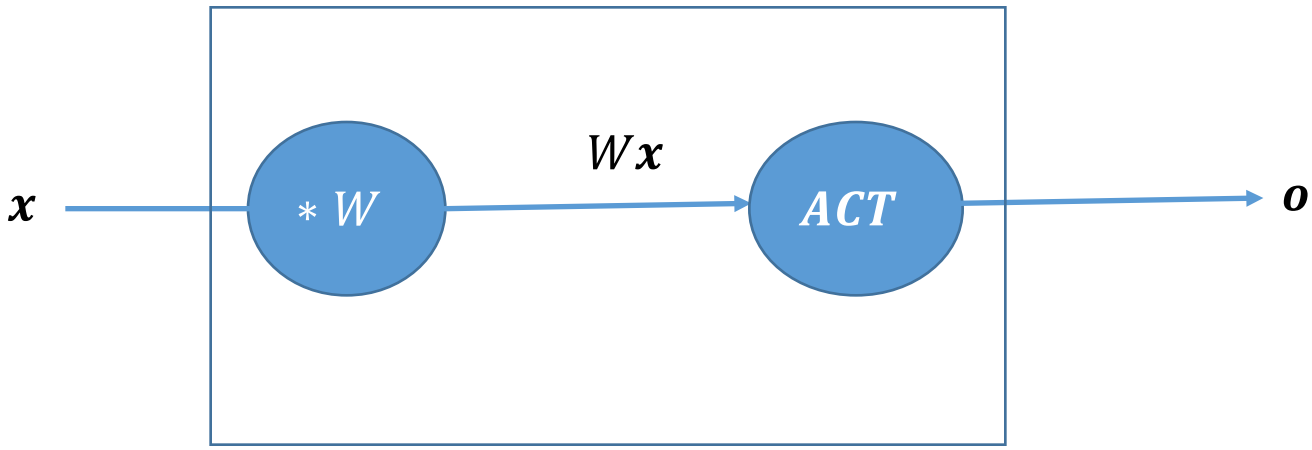
Gradient w.r.t. W will be what ??

Neural Networks

Linear transformations:

At a node, input $\boldsymbol{x}$, and matrix W, output is $W\boldsymbol{x}$.

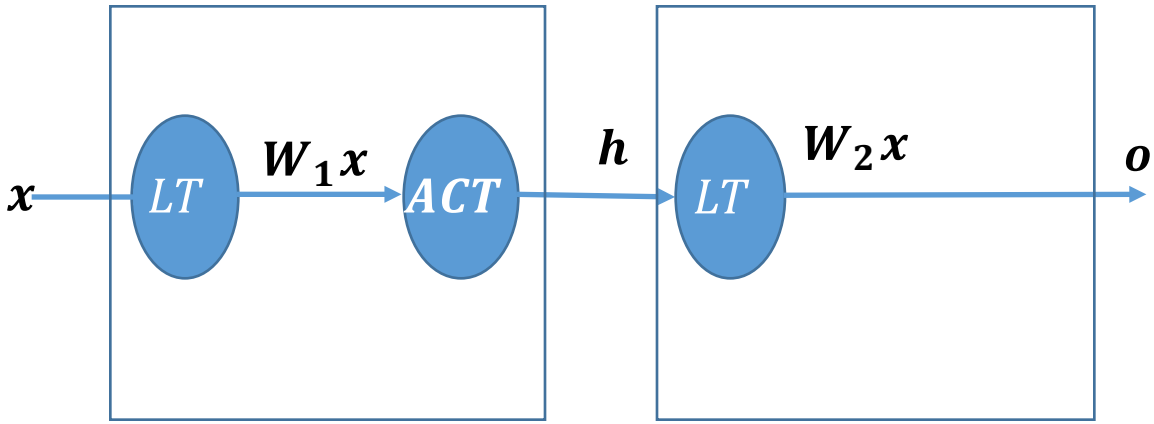
One LAYER:



Neural Networks

One LAYER:

Linear transformation, followed by non-linear transformation

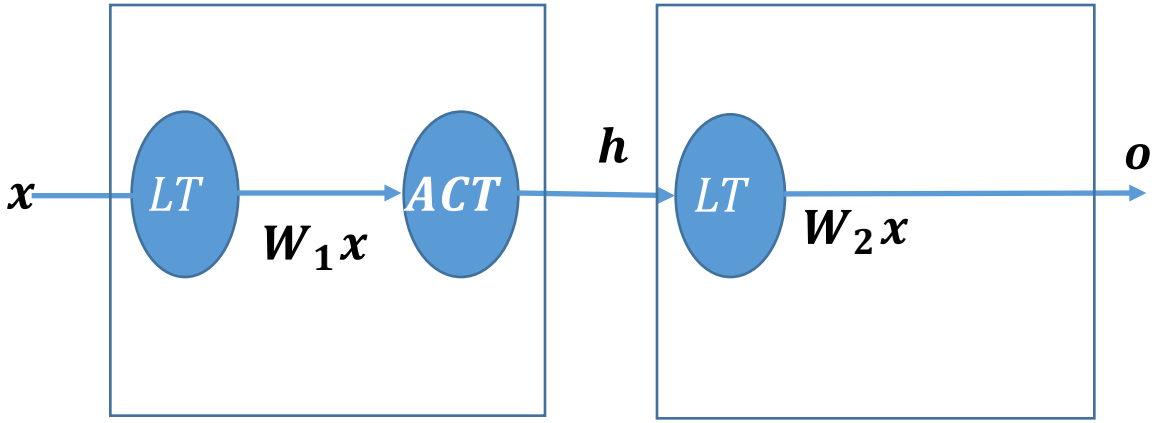


ReLu activation: $\boldsymbol{o} = \max\{0, W\boldsymbol{x}\}$, saves the non-negative terms

Neural Networks

One LAYER:

Linear transformation, followed by non-linear transformation

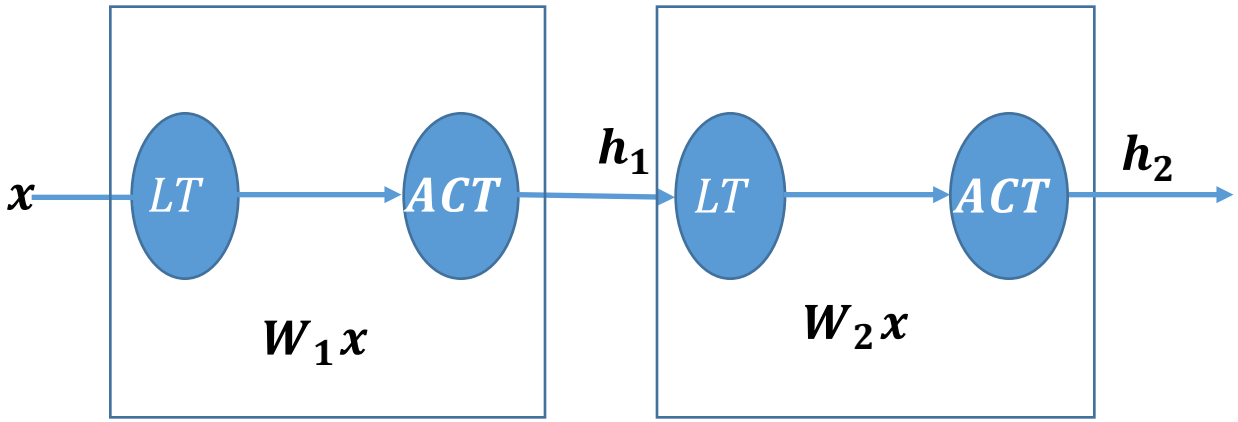


$$h = W_1 \boldsymbol{x}$$

2 layer network: $o = W_2 \cdot max\{0, W_1 \boldsymbol{x}\}$, ReLU networks

Neural Networks

Can now cascade many layers …



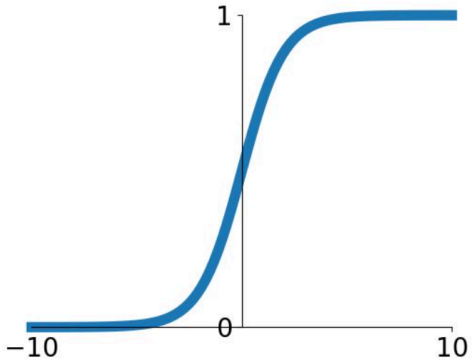
3 layer network: $o = W_3 \cdot \max\{0, W_2 \cdot \max\{0, W_1 \boldsymbol{x}\}\}$, ReLU networks

Activation Functions

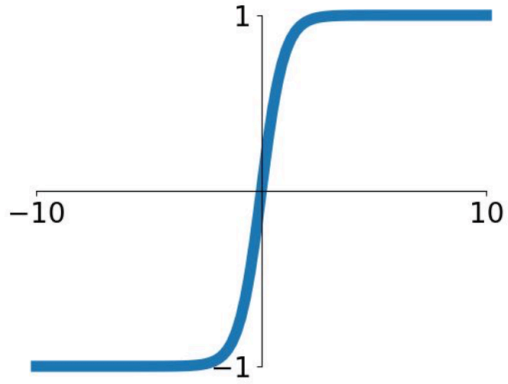
Activation functions/Non-linearity

What are the activations used?

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



Tanh: $\tanh y = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}$



Pros and Cons

Saturated neurons kill gradients

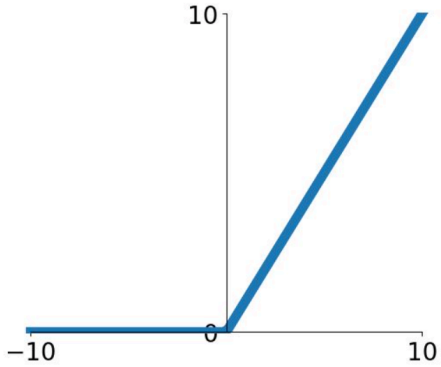
Sigmoid not super easy to compute

Not centered (why do we care?)

ReLU, Rectified Linear Units

What are the activations used?

$$\mathrm{ReLU}\,(y) = \max(0, y)$$



Pros, and cons?

Does not saturate

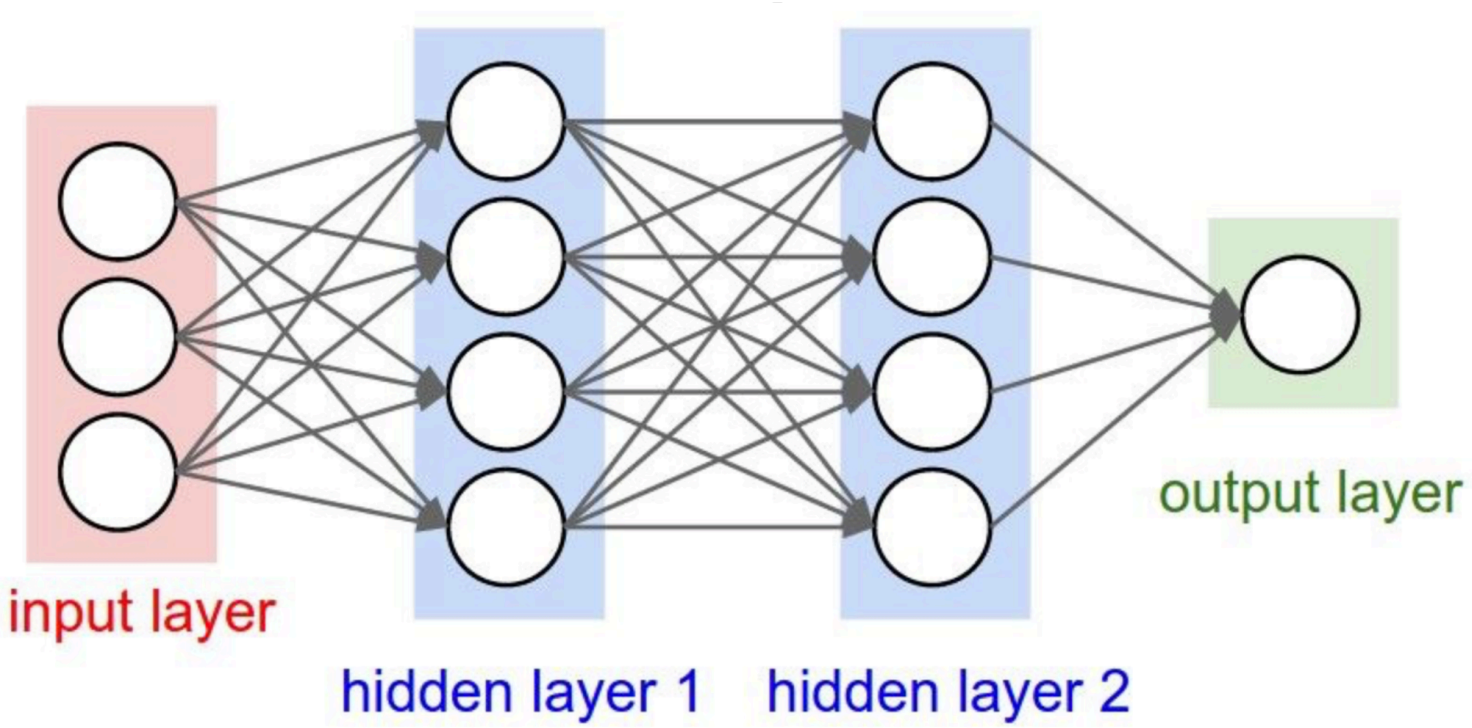
Fast to compute

Works great!

Many work **arounds**, no time to go **around** all of them

Training Neural Networks

3 Layer Neural Networks



Training Neural Networks

Fix the architecture (# Layers, size of each layer, loss functions, …)

This tells you what the parameters are

Training NN using minibatch:

- Take a mini batch T
- Compute the functions in the computational graph, obtain loss
- Back propagate to calculate the gradients
- Update (each layer) based on the gradients

Repeat the process until convergence

Gradient descent

$$\min_{\boldsymbol{W}} L(\boldsymbol{W}, S)$$

This is hard to solve in most cases (no **closed form**).

$L(\boldsymbol{W}, S)$ is **just** a function of $\boldsymbol{W}$

In practice, use gradient descent (recall logistic regression).

- Start with some $\boldsymbol{W}(\mathbf{0})$
- Repeat until convergence:
  - Compute $\nabla L(\boldsymbol{W}, S)|_{\boldsymbol{W}(\boldsymbol{j})}$
  - $\boldsymbol{W}(\boldsymbol{j}+\mathbf{1}) = \boldsymbol{W}(\boldsymbol{j}) + \eta \cdot \nabla L(\boldsymbol{W}, S)|_{\boldsymbol{W}(\boldsymbol{j})}$

SGD

At iteration j:

Pick a random training $(\bar{X}_i, y_i)$, then

$$\boldsymbol{W}(j+1) = \boldsymbol{W}(j) + \eta \cdot \nabla L(\boldsymbol{W}, (\bar{X}_i, y_i))\Big|_{\boldsymbol{W}(j)}$$

SGD + Momentum:

$$\boldsymbol{W}(j+1) = \boldsymbol{W}(j) + \eta \cdot \nabla L(\boldsymbol{W}, (\bar{X}_i, y_i))\Big|_{\boldsymbol{W}(j)} + \gamma \cdot (\boldsymbol{W}(j) - \boldsymbol{W}(j-1))$$

Minibatch SGD

At each round:

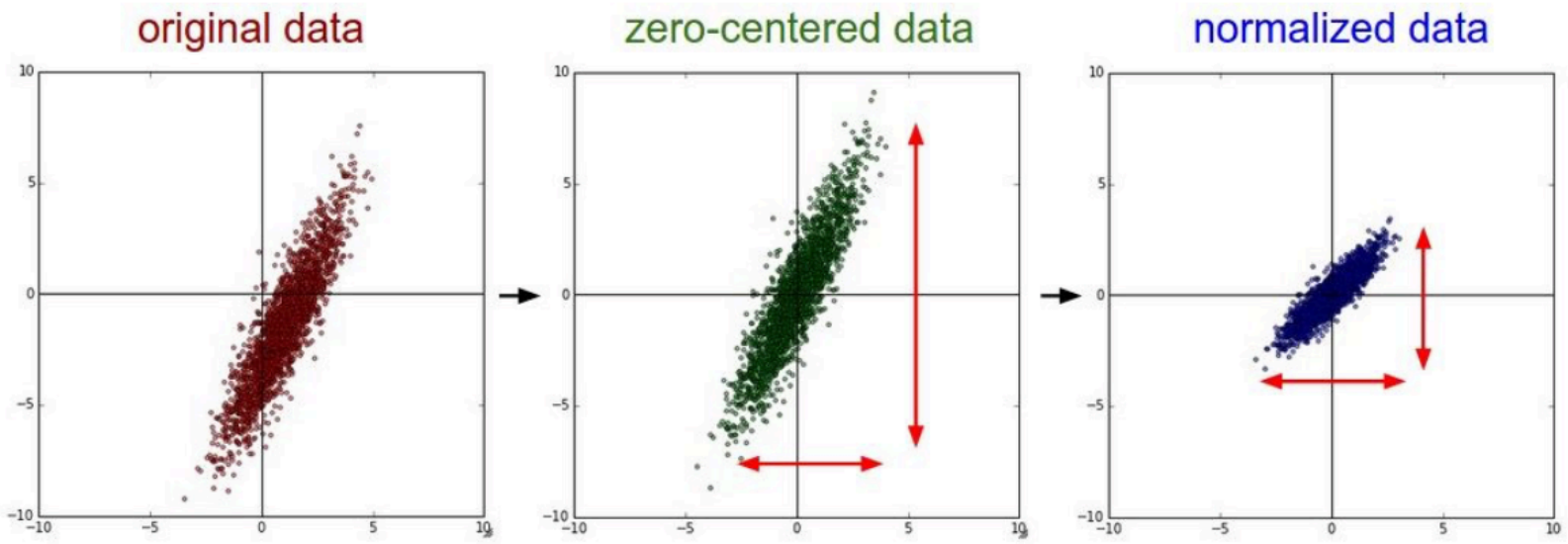
- Pick a few (tens/100's) training examples at random, say $T \subset S$
- Compute the gradient only for this batch:

$$\sum_{i \in T} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}, (\bar{X}_i, y_i)) \Big|_{\boldsymbol{W}(\boldsymbol{j})}$$

- $\boldsymbol{W}(j+1) = \boldsymbol{W}(j) + \eta \cdot \nabla L(\boldsymbol{W}, T)|_{\boldsymbol{W}(j)}$

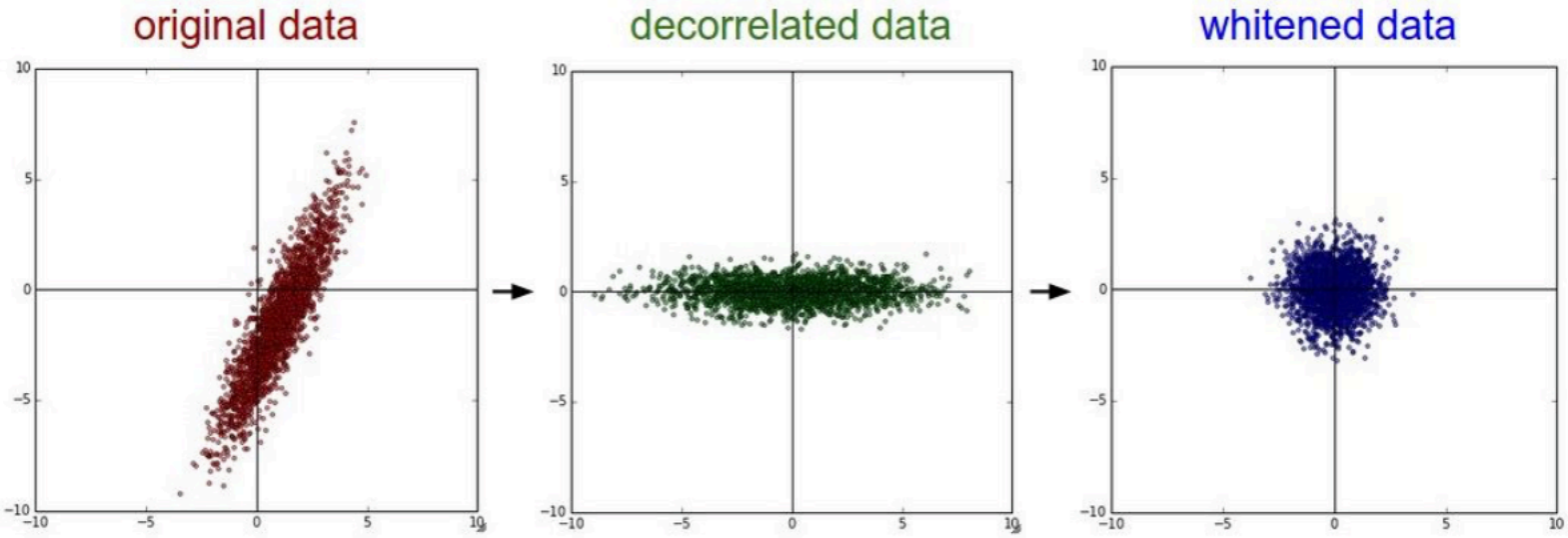
Data Pre-processing

The first steps



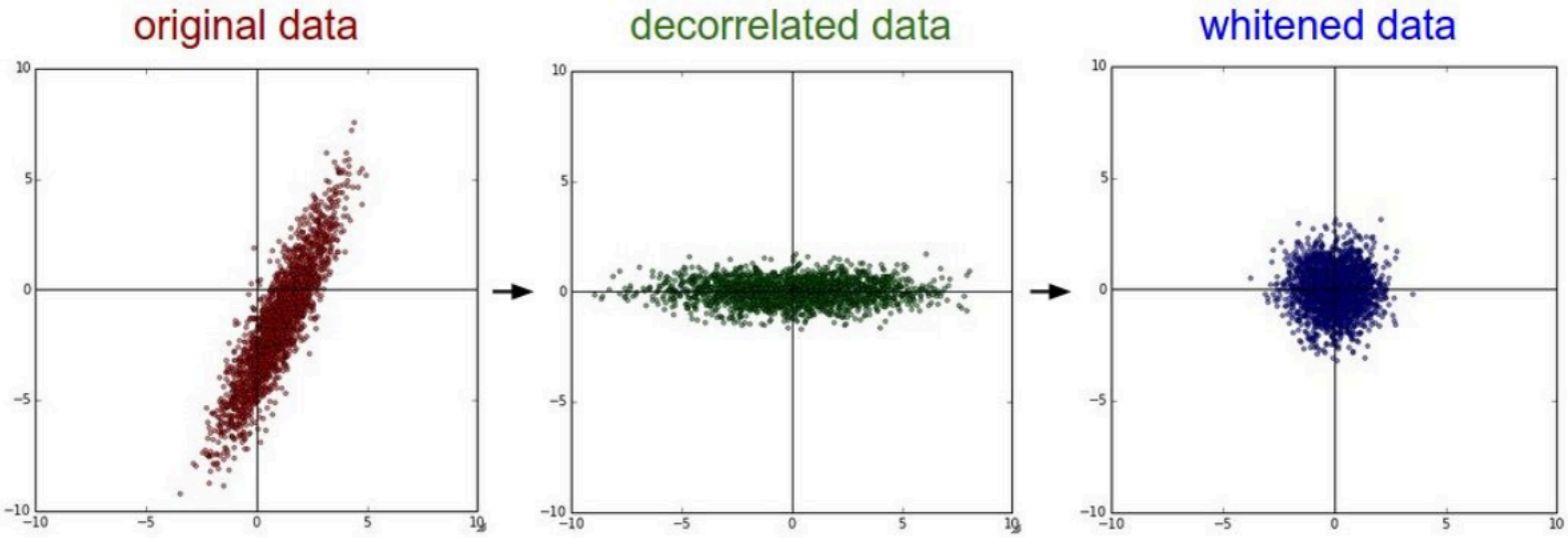
The first steps

In practice, you may also see **PCA** and **Whitening** of the data



The first steps

In practice, you may also see **PCA** and **Whitening** of the data



CNN

Next: Convolutional Neural Networks

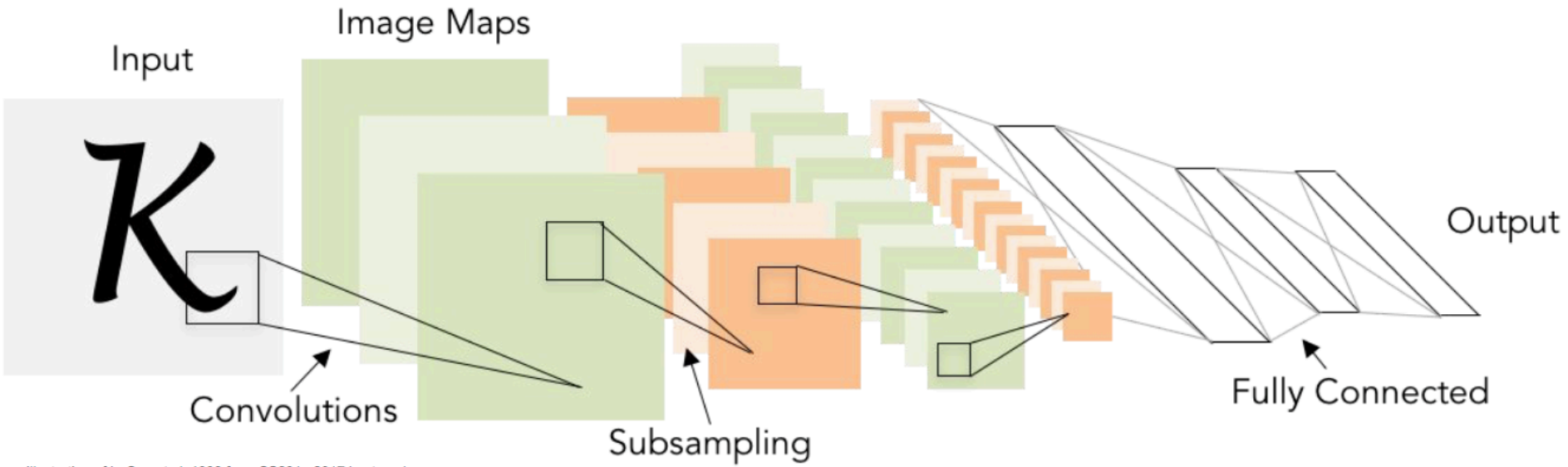


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNN

Not super new … but super powerful now

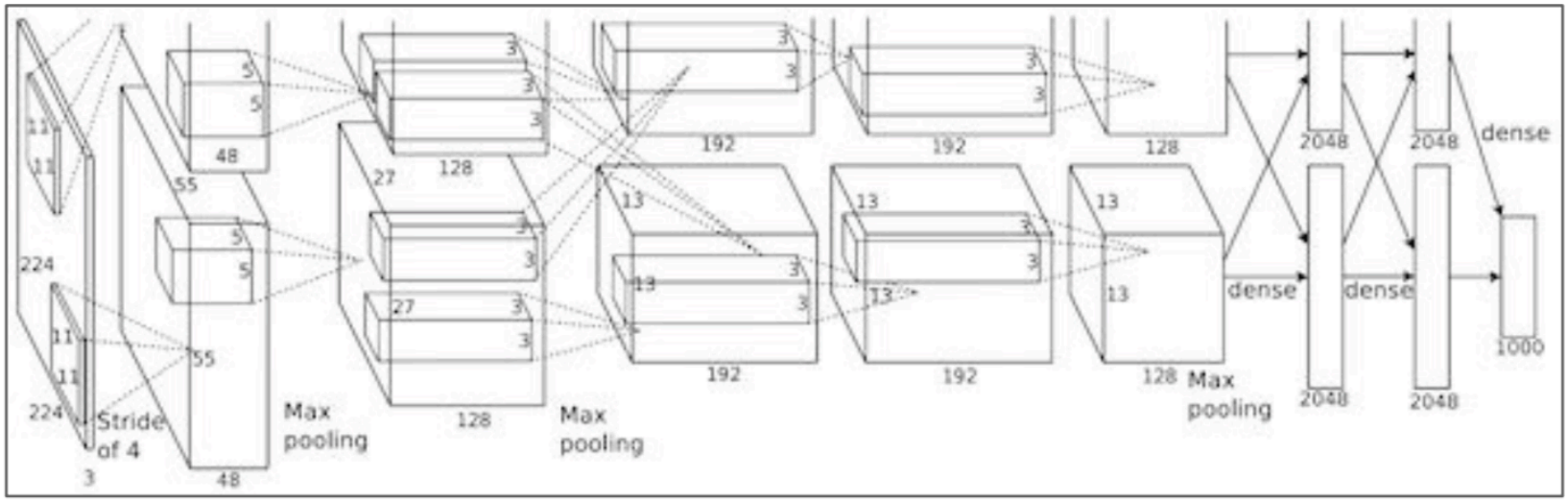
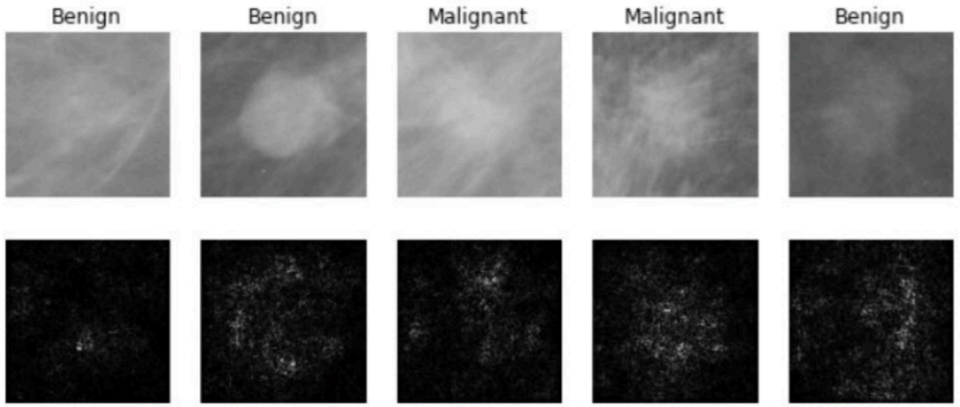


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

CNN is everywhere

Yes …



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

CNN simple

One simple step of convolution: