

# Deep Learning

Week - 5

# Today's Lecture

- Review Machine Learning Basics
- Back-propagation
- Convolutional Neural Networks
- Short tutorial on building your own network.

# Recipe for design a Machine Learning system

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define Goal state:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with BP in SGD:  
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$



# Learning Algorithm

- Initialize  $\theta$  ( $\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$ )
  - For N iterations:
    - For each training example  $(\mathbf{x}^{(t)}, y^{(t)})$
    - $\Delta = -\nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$
    - $\theta \leftarrow \theta + \alpha \Delta$
  - To apply this algorithm to neural network training, we need
    - The loss function  $l(f(\mathbf{x}^{(t)}; \theta), y^{(t)})$
    - Procedure to compute parameter gradients  $\nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)})$
    - The regularizer  $\Omega(\theta)$  (and the gradient  $\nabla_{\theta} \Omega(\theta)$ )
    - Initialization method
- Training epoch  
=  
Iteration over all examples

# Back-propagation learning

# Training deep neural networks

- Outlook:
  - Computing gradients is hard (many parameters)  
→ Back-propagation!
- Network represents a chain of function calls (one per layer):

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \mathbf{f}_{L+1} \circ \mathbf{f}_L \circ \cdots \circ \mathbf{f}_1(\mathbf{x}) \\ &= \mathbf{f}_{L+1}(\mathbf{f}_L(\dots \mathbf{f}_1(\mathbf{x}))) \end{aligned}$$

- Gradients: Chain rule can be applied!

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{f}_{L+1}}{\partial \mathbf{f}_L} \cdot \frac{\partial \mathbf{f}_L}{\partial \mathbf{f}_{L-1}} \cdot \dots \cdot \frac{\partial \mathbf{f}_1}{\partial \boldsymbol{\theta}}$$

# How to compute gradient?

## Symbolic Differentiation

- Input formulae is a symbolic expression tree (computation graph).
- Implement differentiation rules, e.g., sum rule, product rule, chain rule

$$\frac{d(f + g)}{dx} = \frac{df}{dx} + \frac{dg}{dx} \quad \frac{d(fg)}{dx} = \frac{df}{dx}g + f\frac{dg}{dx} \quad \frac{d(h(x))}{dx} = \frac{df(g(x))}{dx} \cdot \frac{dg(x)}{x}$$

- ✗ For complicated functions, the resultant expression can be exponentially large.
- ✗ Wasteful to keep around intermediate symbolic expressions if we only need a numeric value of the gradient in the end
- ✗ Prone to error

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x$$
$$[-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x$$

$$[-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

$$f(W, x) = W \cdot x$$

$$[-0.8 + \varepsilon \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

# How to compute gradient?

## Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

- Reduce the truncation error by using center difference

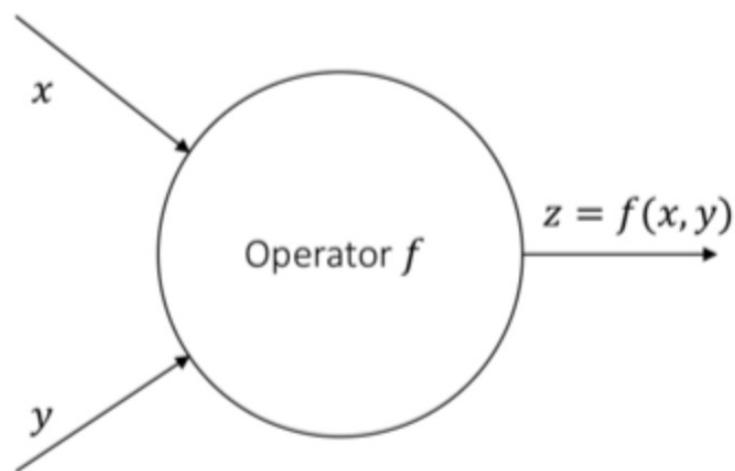
$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

✗ Bad: rounding error, and slow to compute

✓ A powerful tool to check the correctness of implementation, usually use  $h = 1e-6$ .

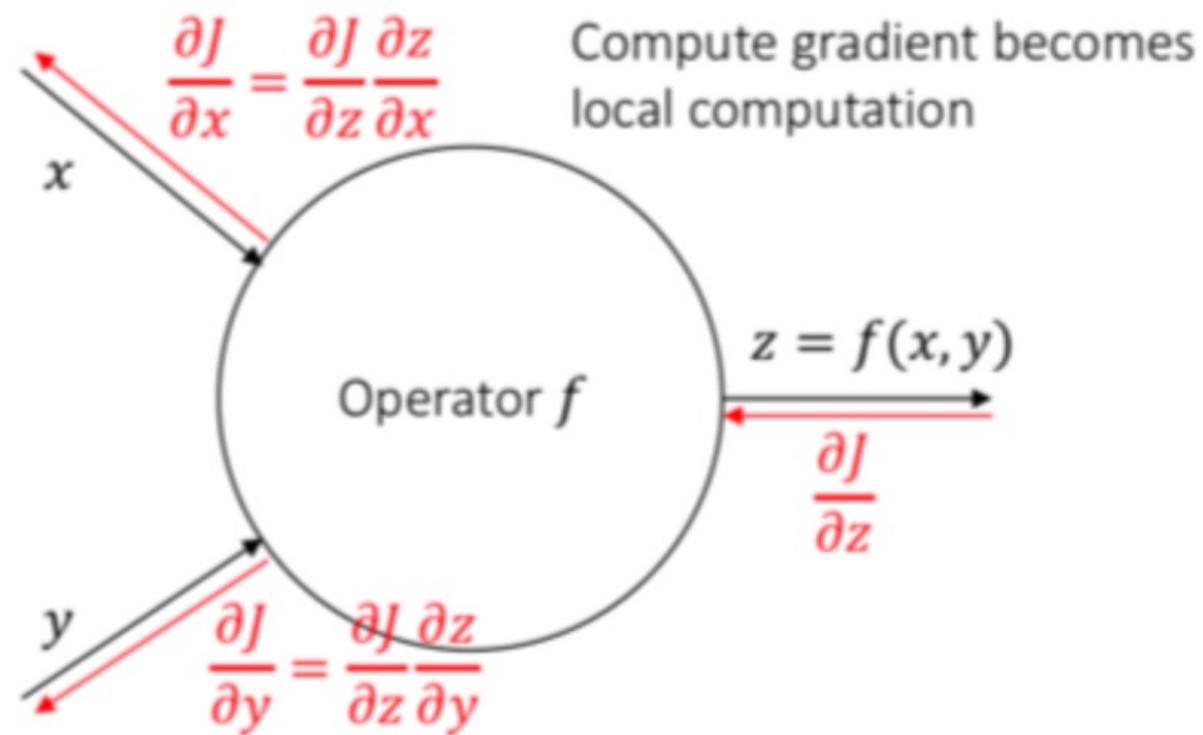
# Back Propagation for computing differentiation

Backpropagation



# Back Propagation for computing differentiation

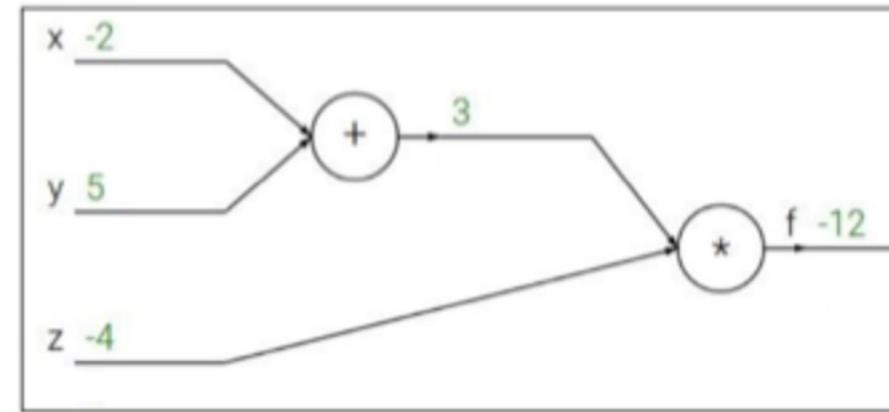
## Backpropagation



# Example: Back-Prop

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

e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$



Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

- slide source: Stanford CS231n lecture note.

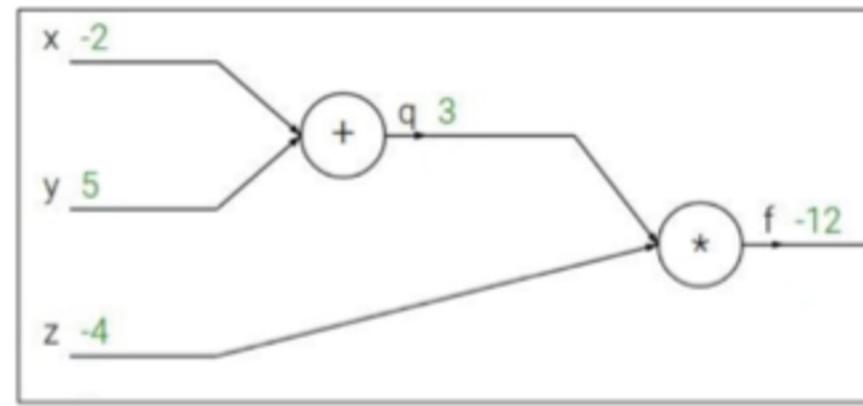
# Example: Back-Prop

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

e.g.  $x = -2, y = 5, z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



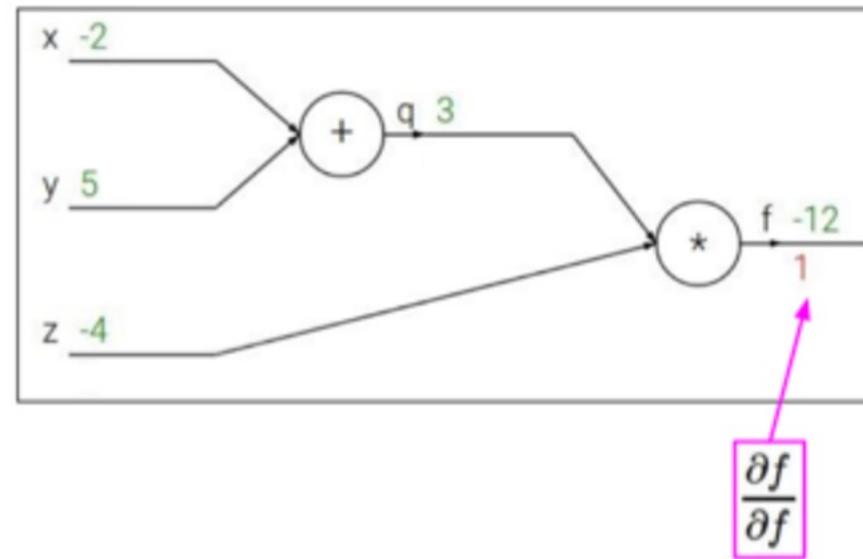
# Example: Back-Prop

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

e.g.  $x = -2, y = 5, z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



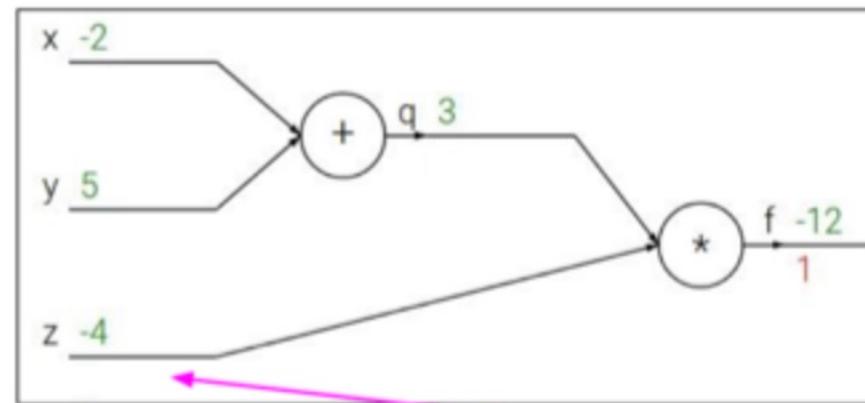
# Example: Back-Prop

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

e.g.  $x = -2, y = 5, z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



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

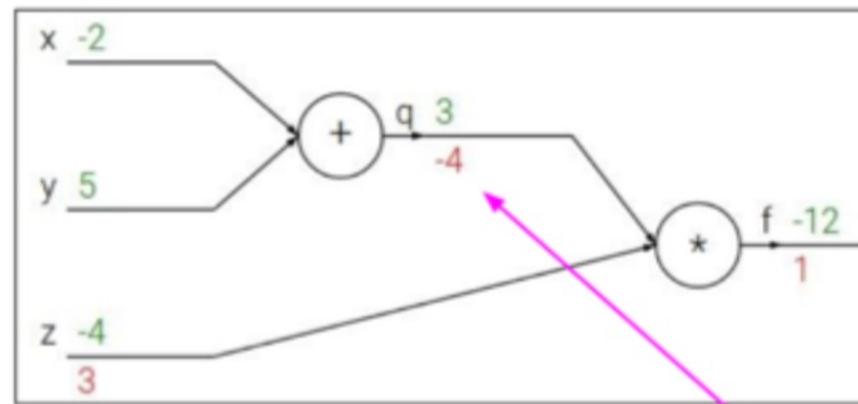
# Example: Back-Prop

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

e.g.  $x = -2, y = 5, z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



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

# Example: Back-Prop

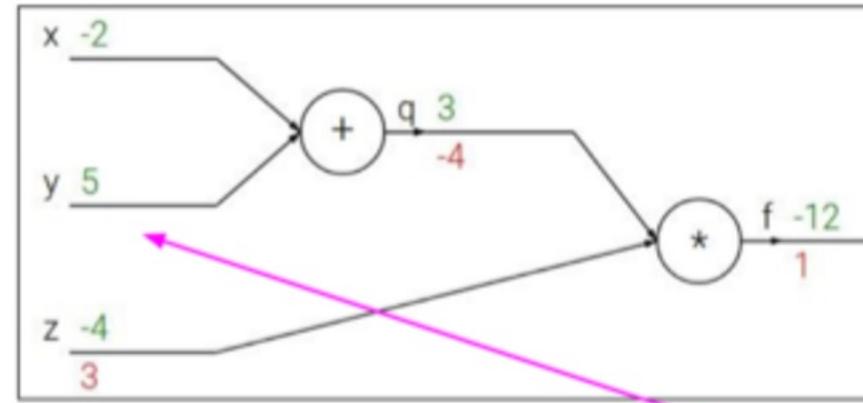
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Example: Back-Prop

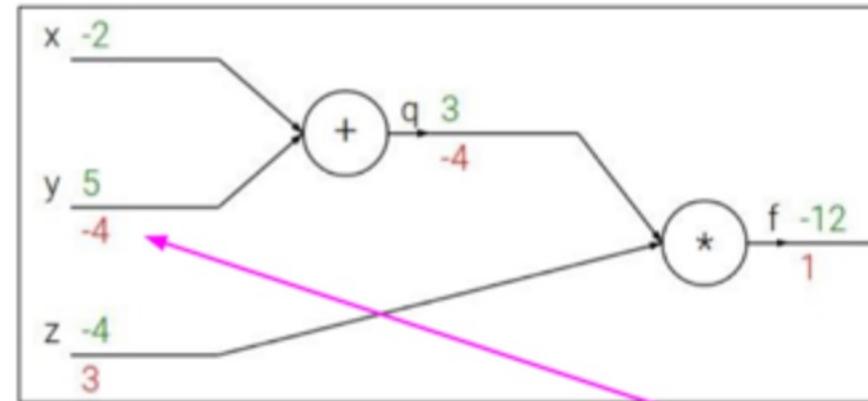
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

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

Upstream  
Gradient

Local  
Gradient

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

# Example: Back-Prop

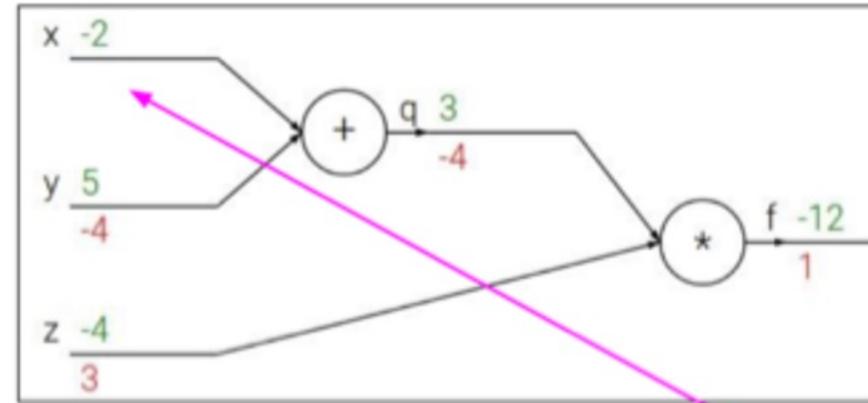
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

# Example: Back-Prop

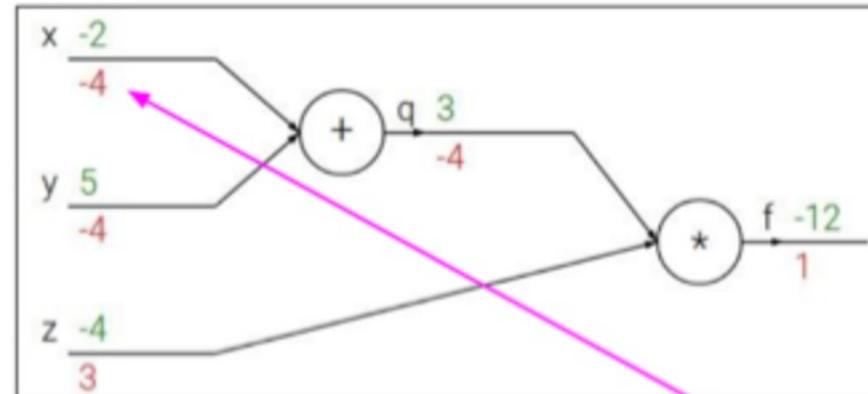
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

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

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



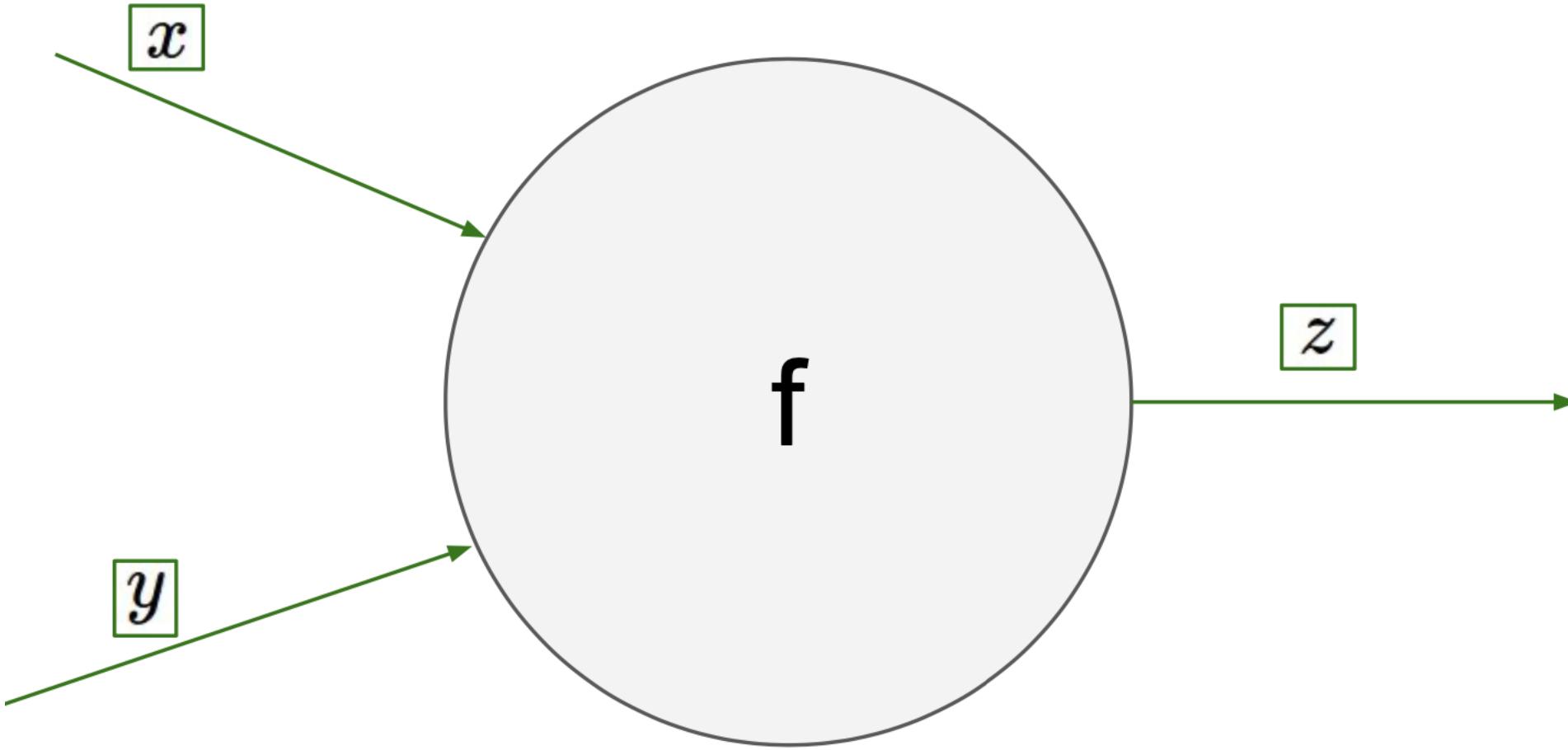
$$\frac{\partial f}{\partial x}$$

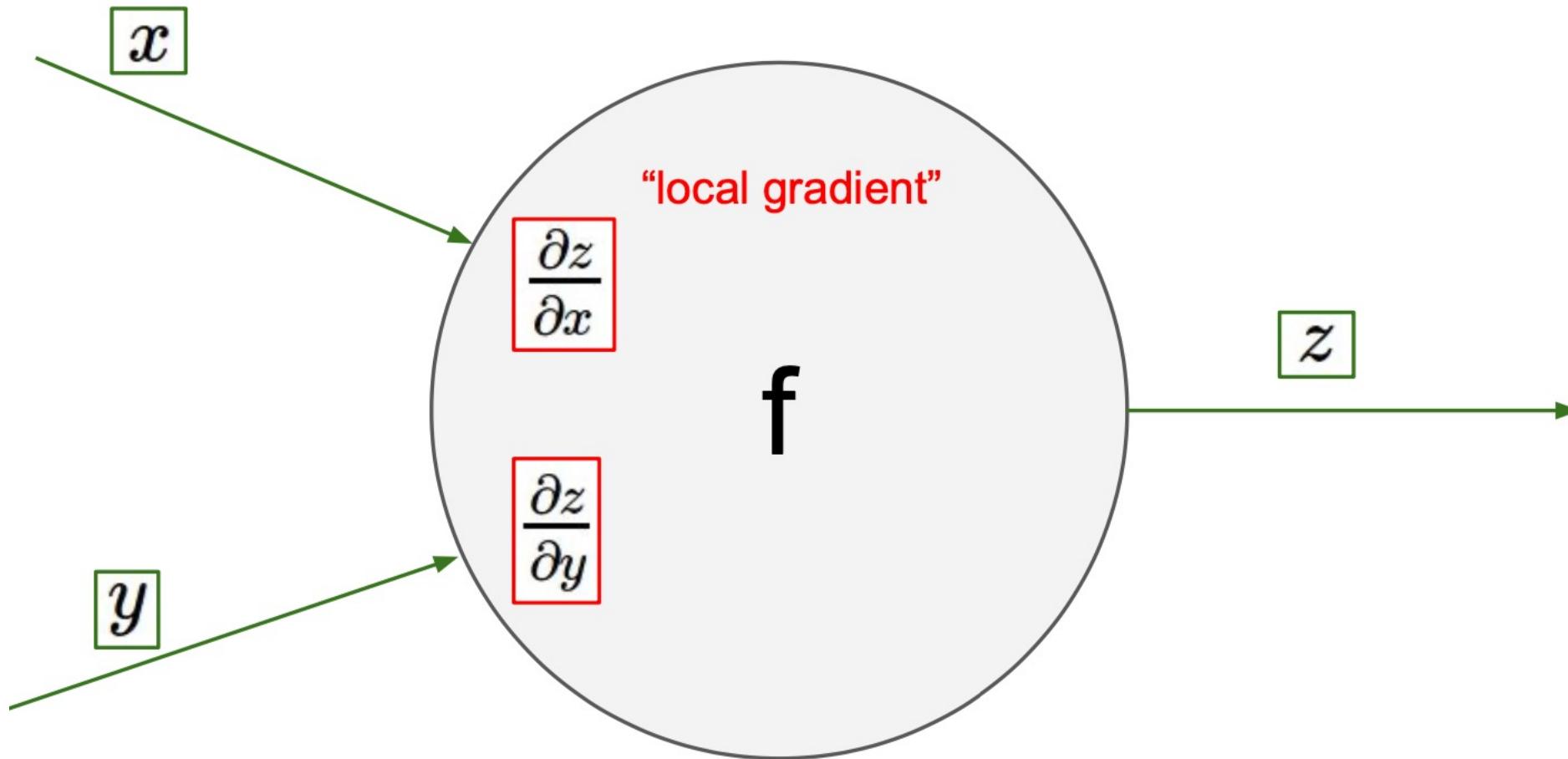
Chain rule:

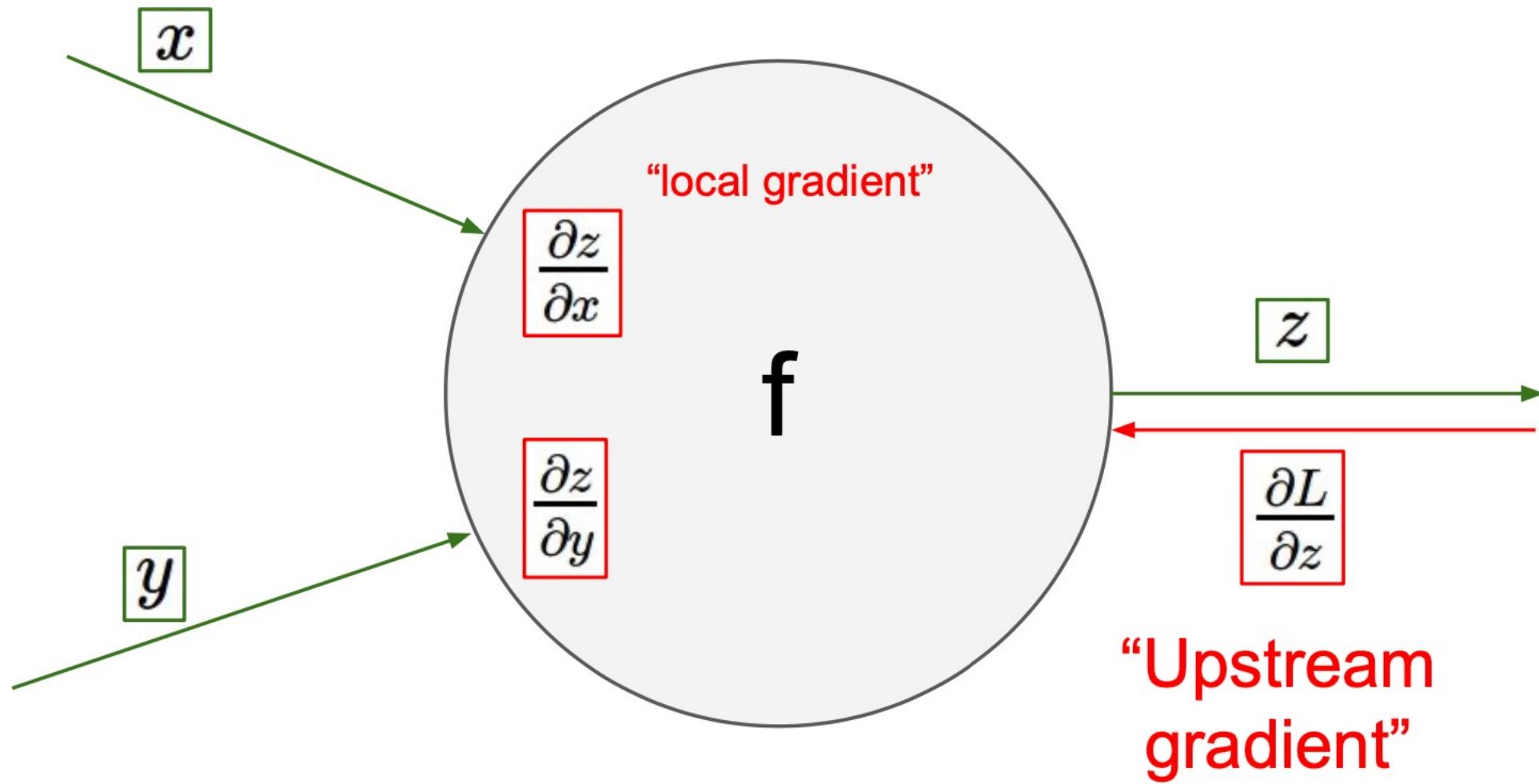
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

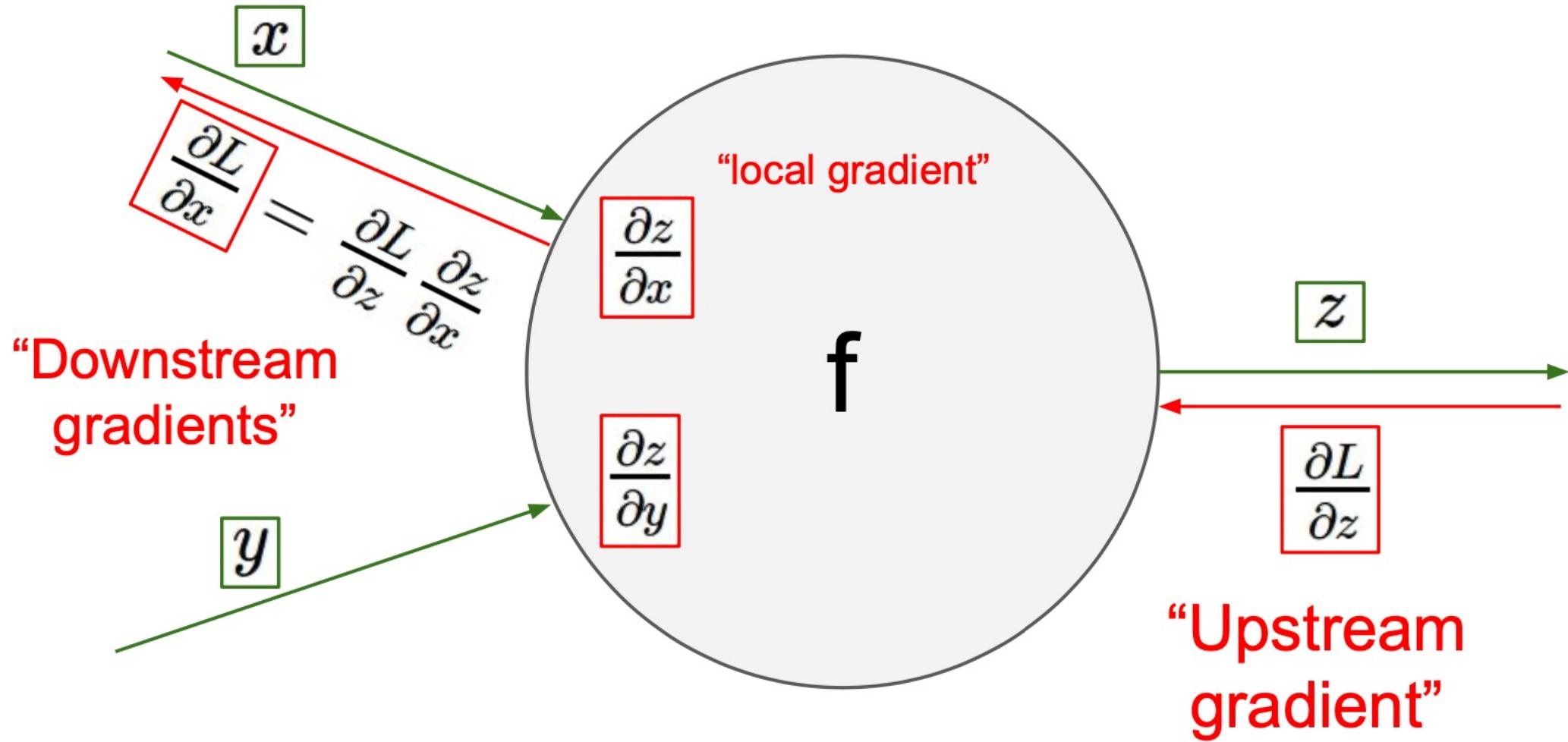
Upstream  
Gradient

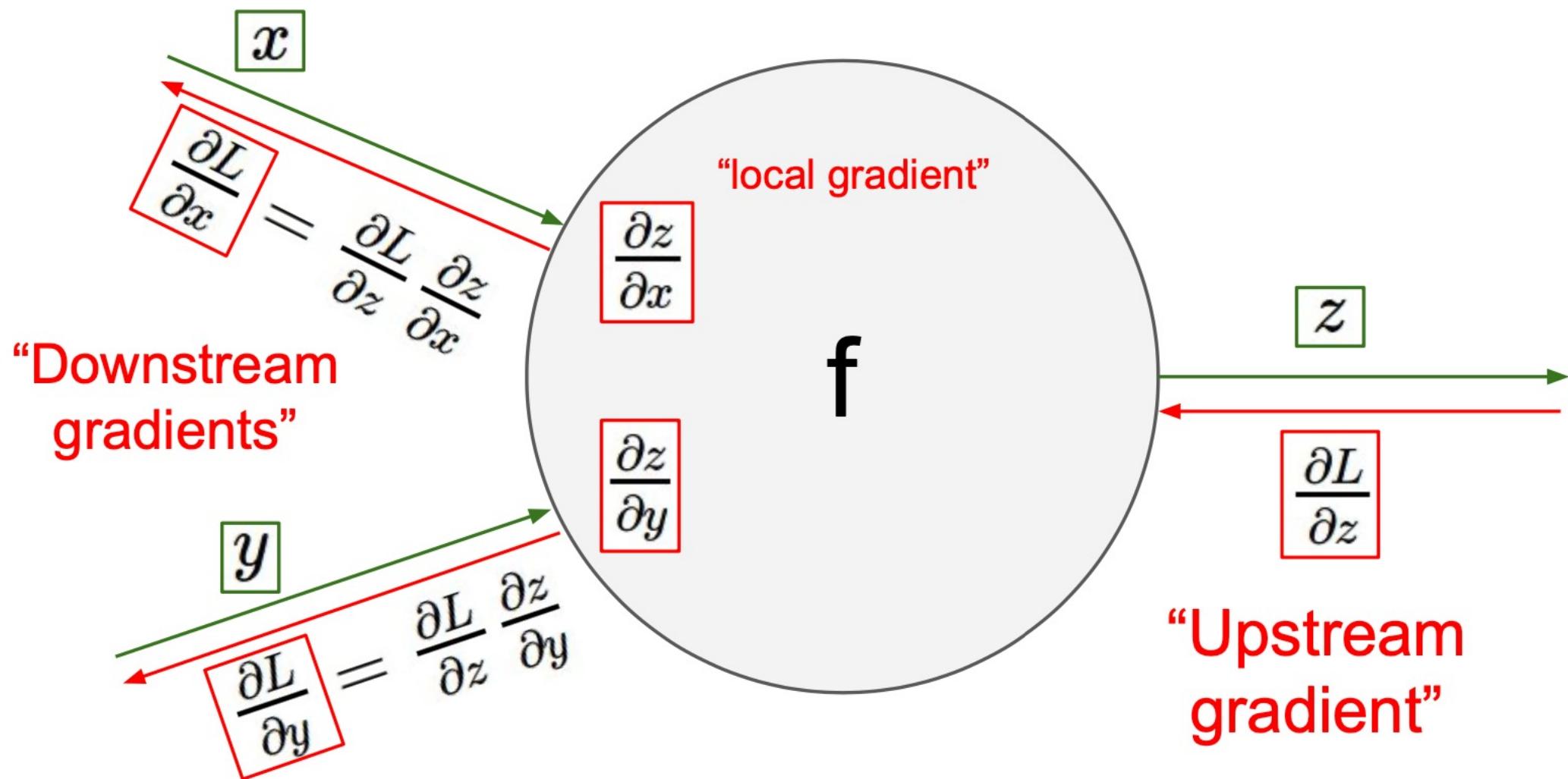
Local  
Gradient

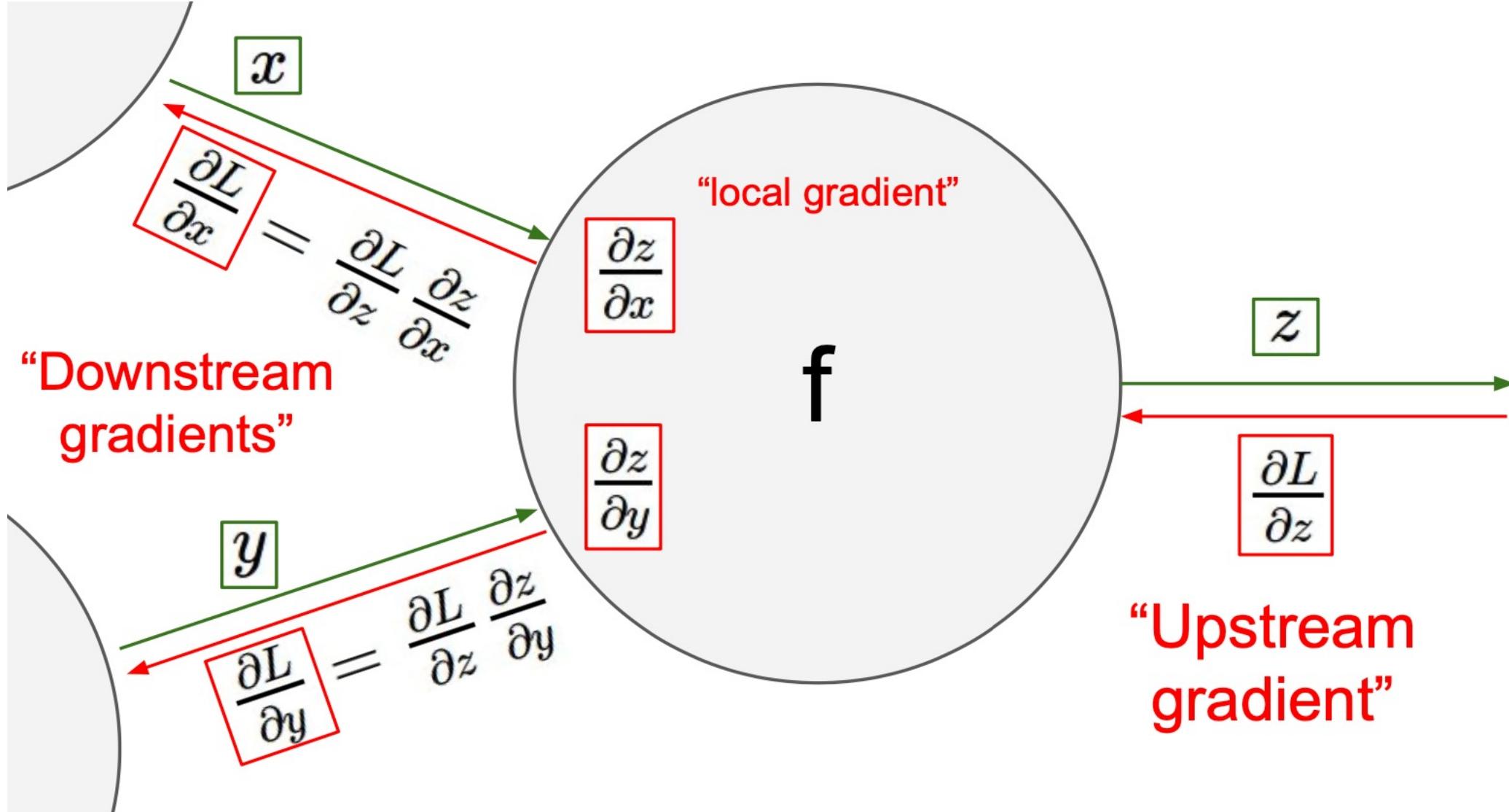






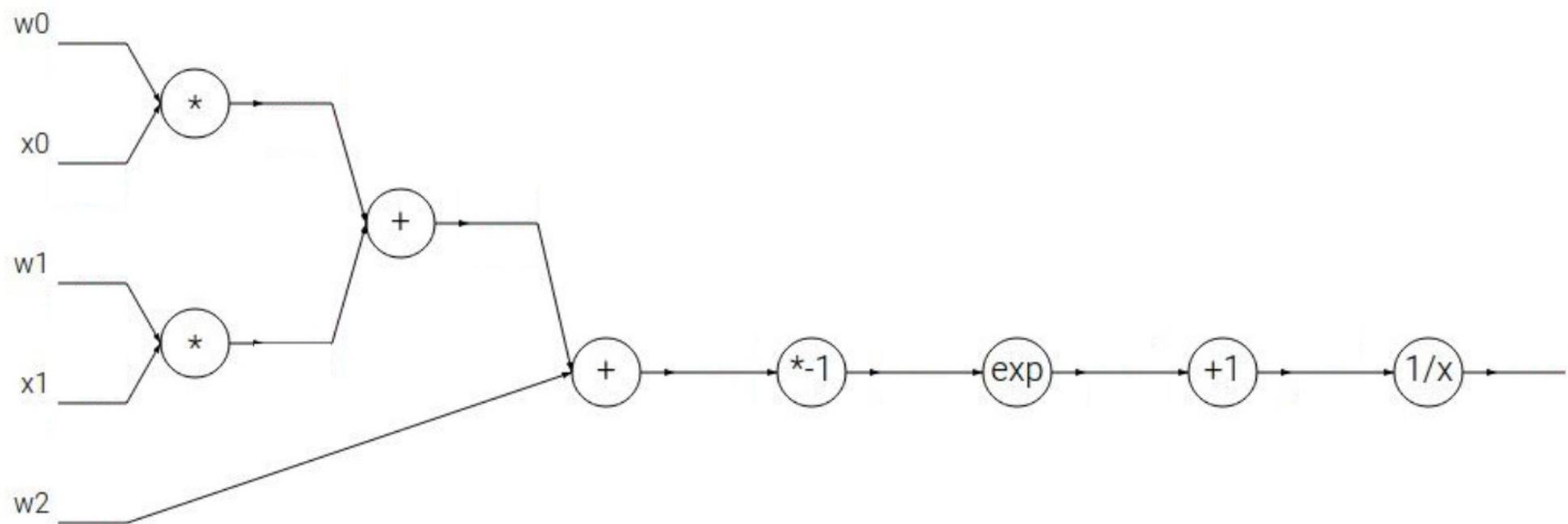






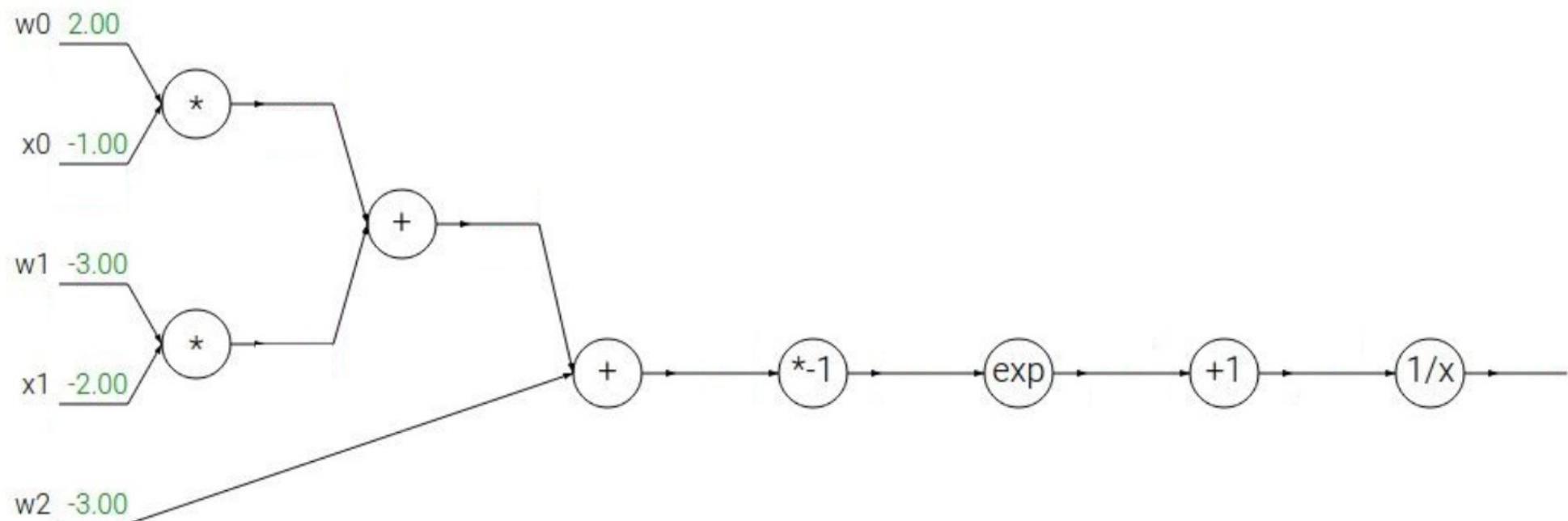
**Another example:**

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



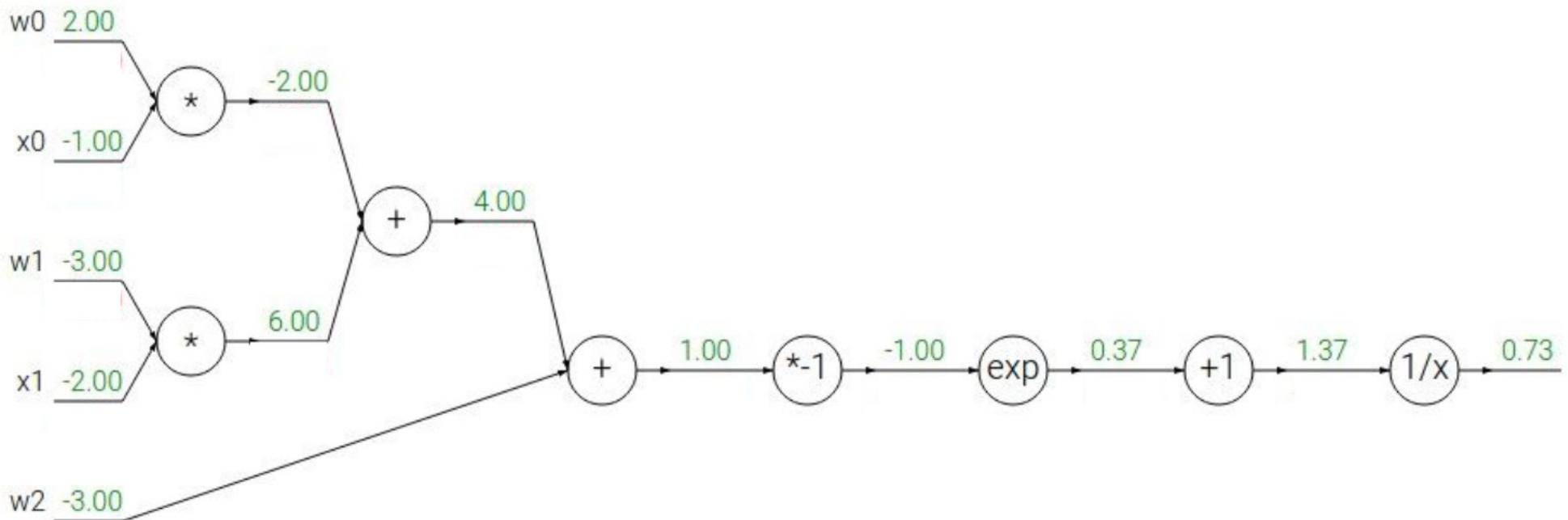
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



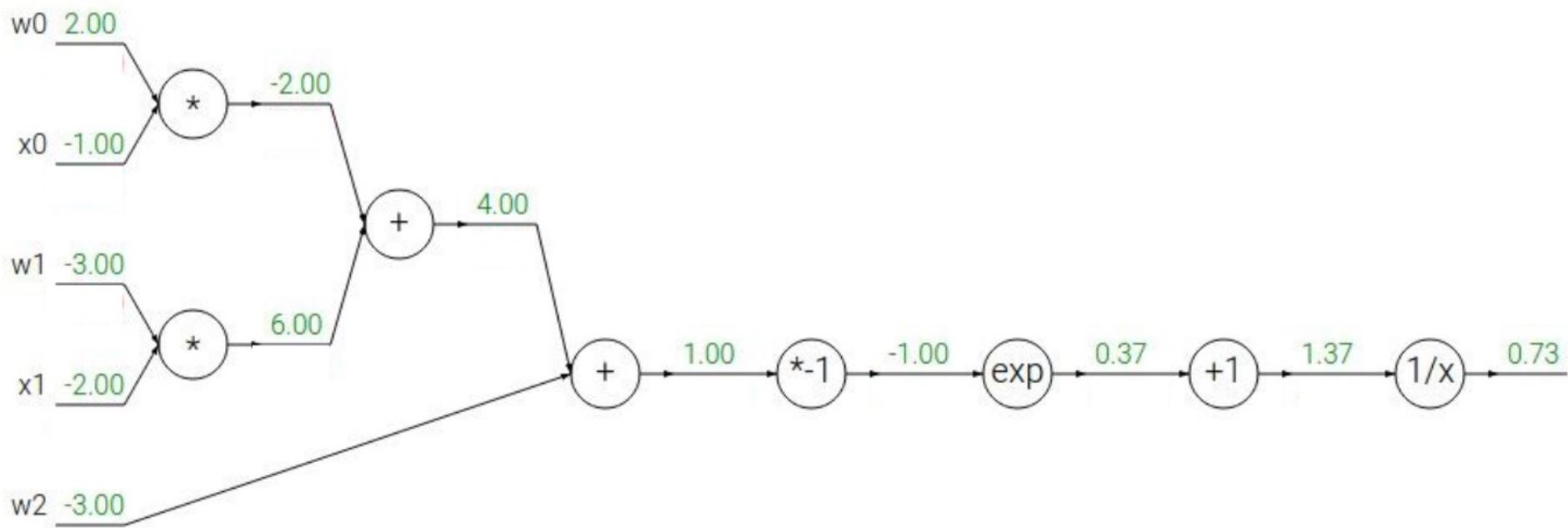
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

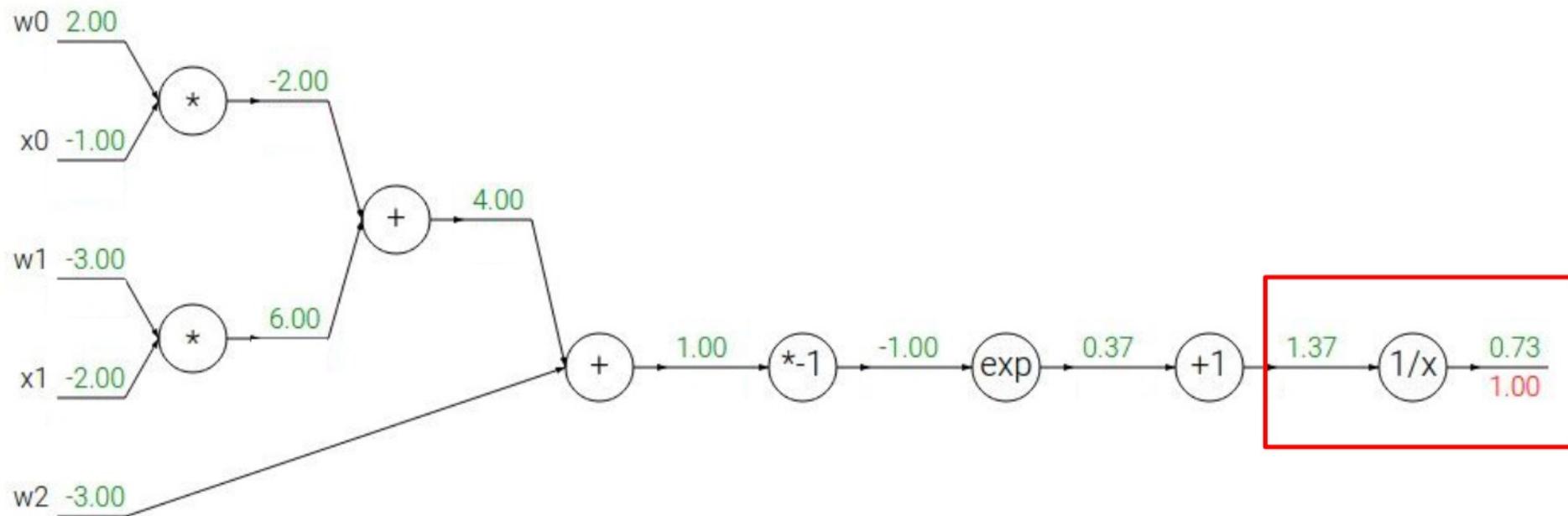
→

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

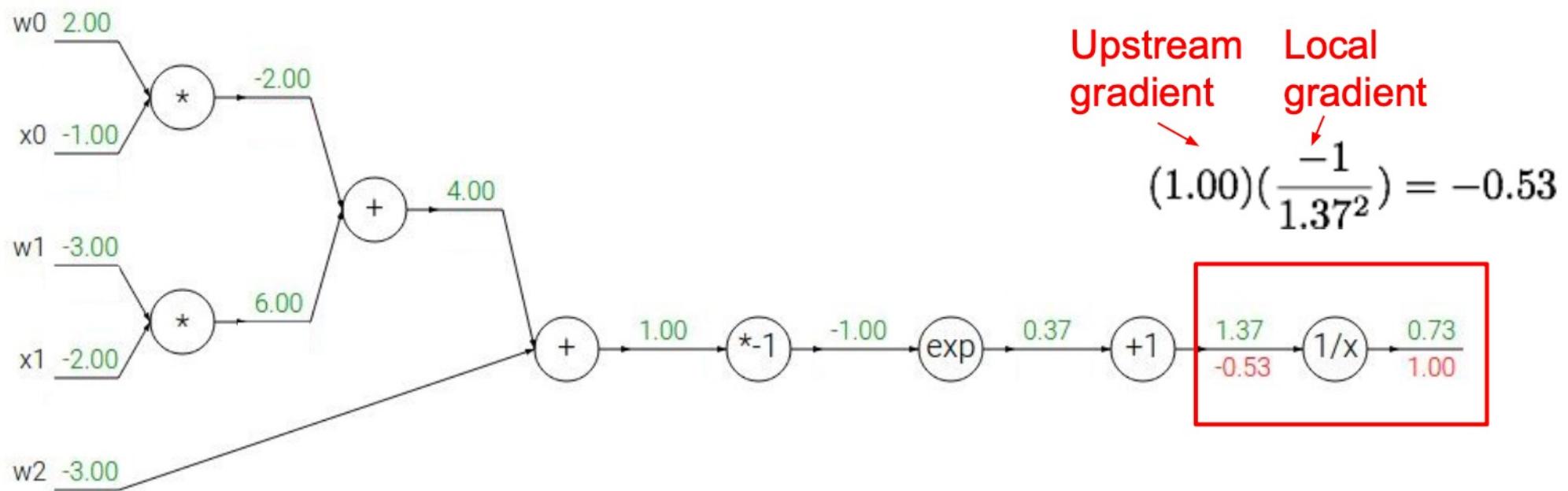
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

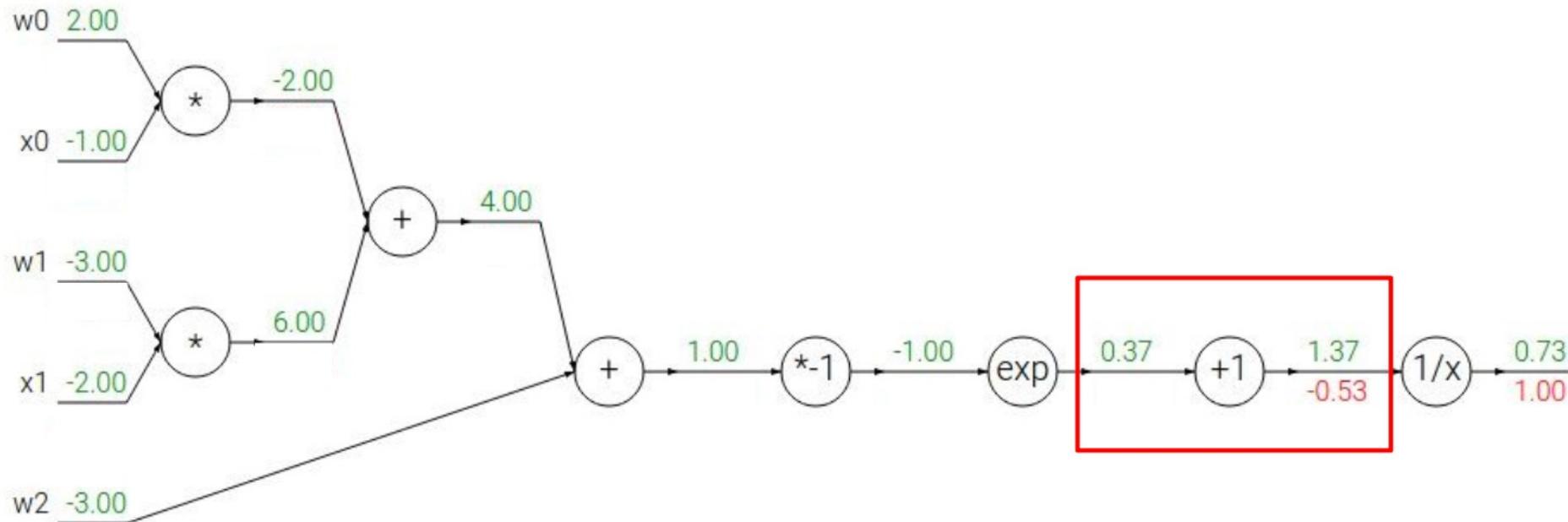
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

→

$$\frac{df}{dx} = -1/x^2$$

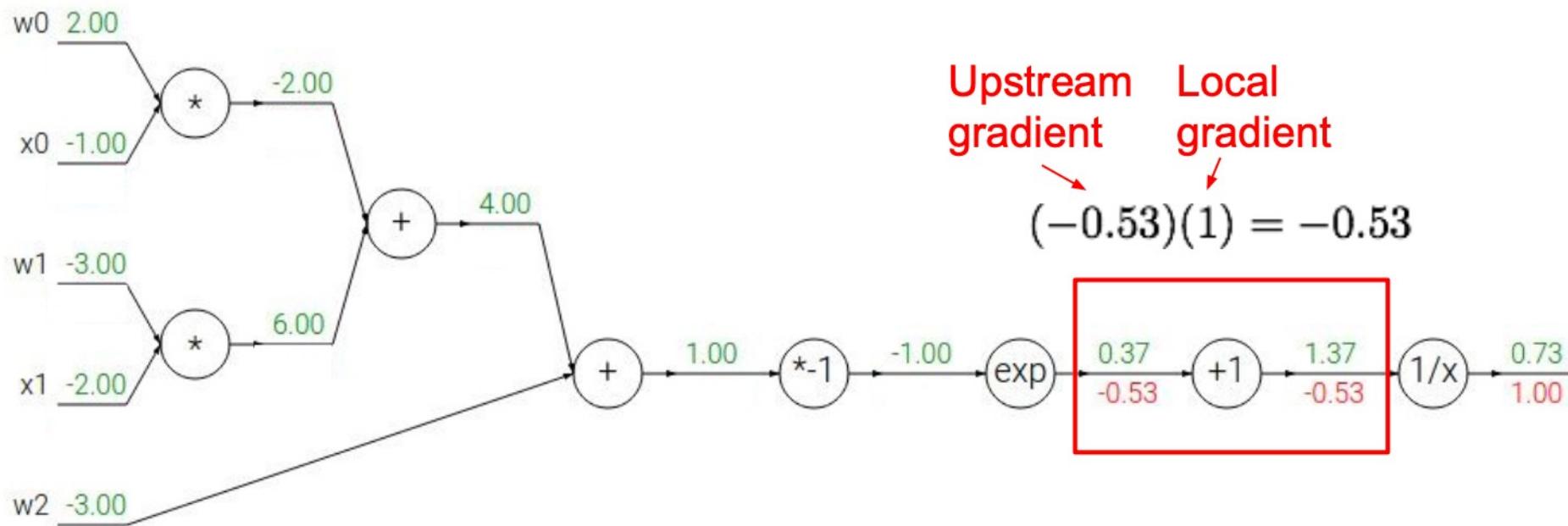
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

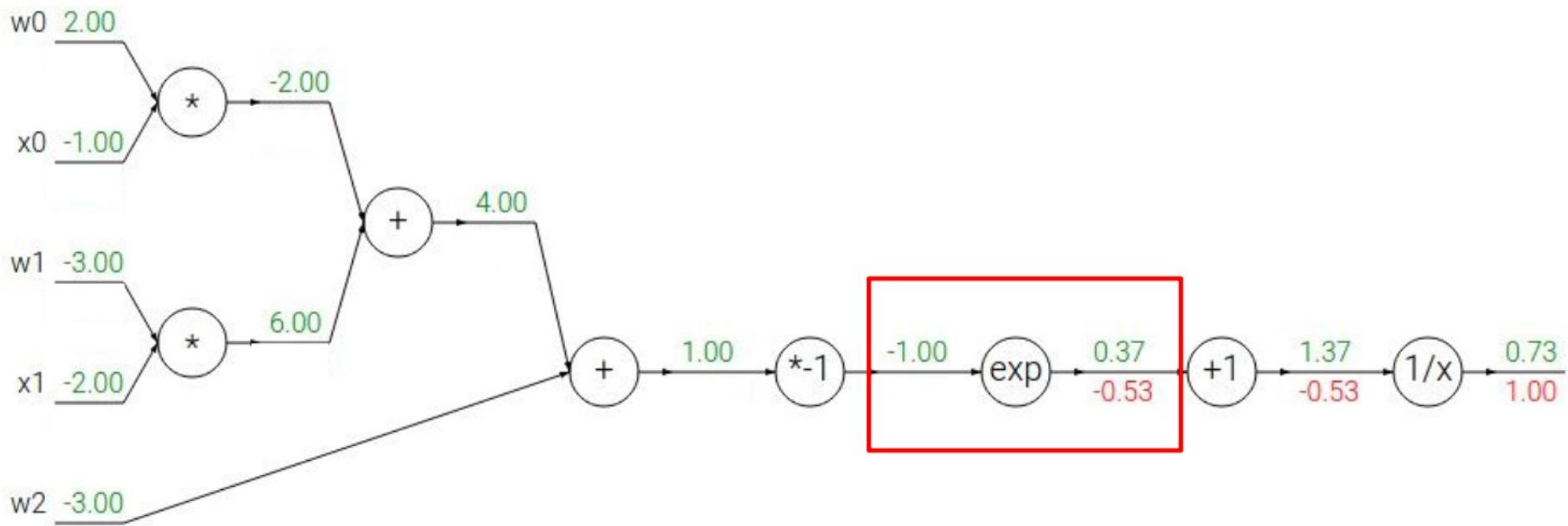
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

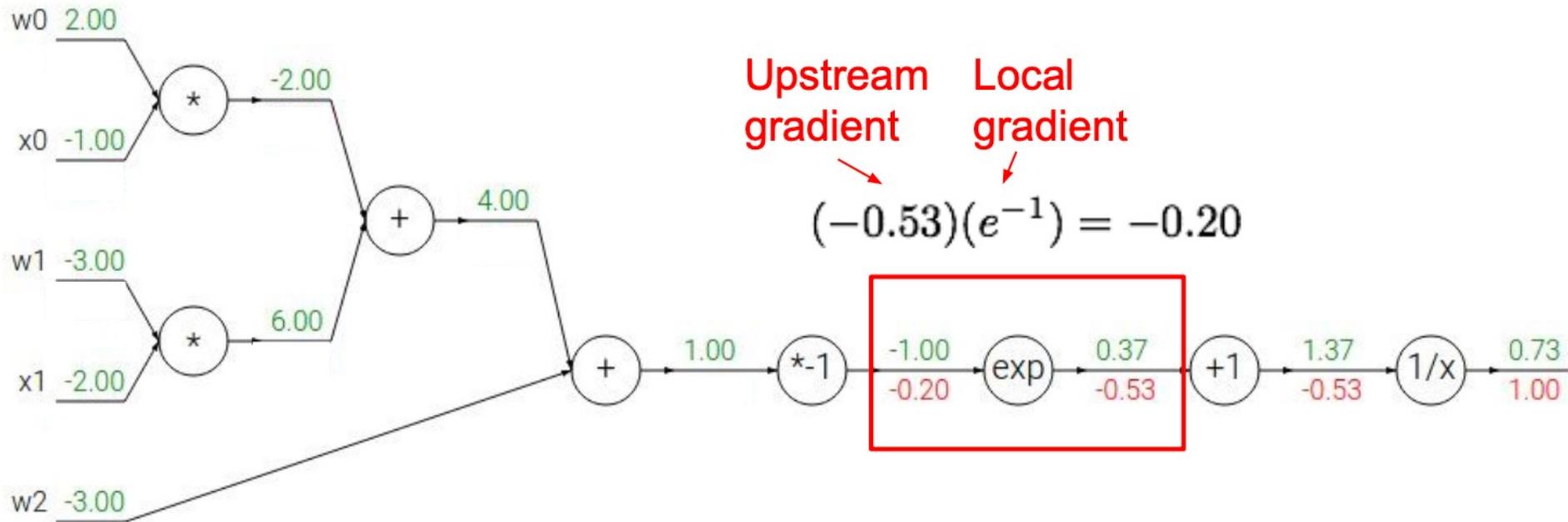
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

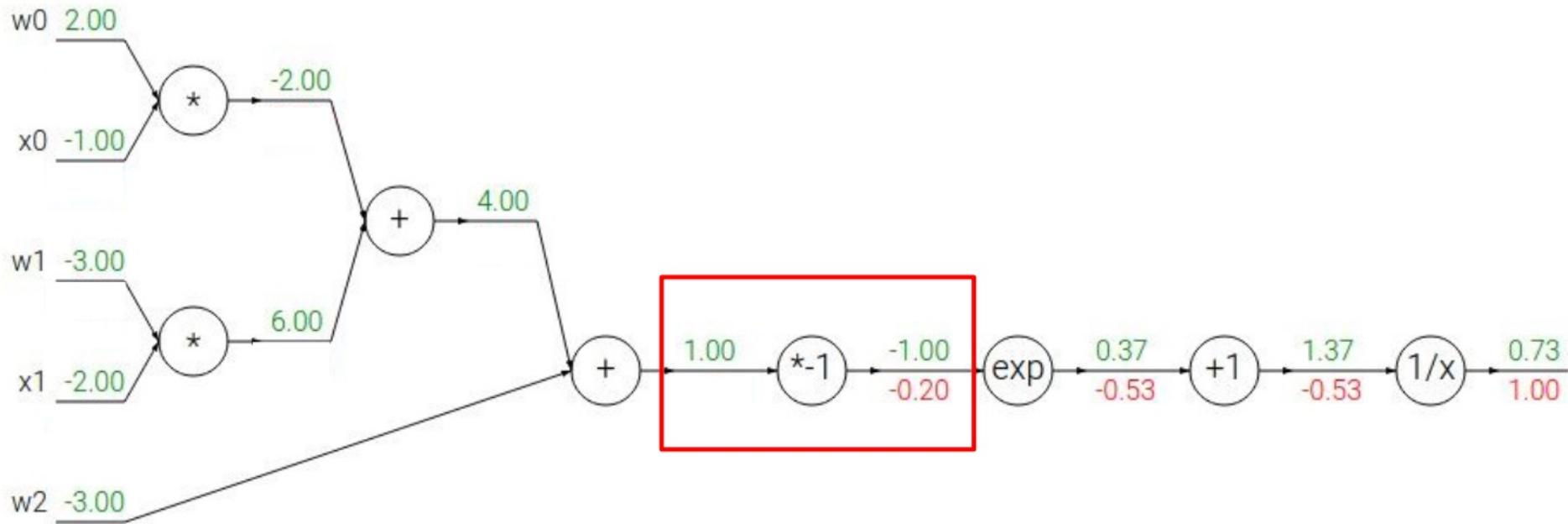
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

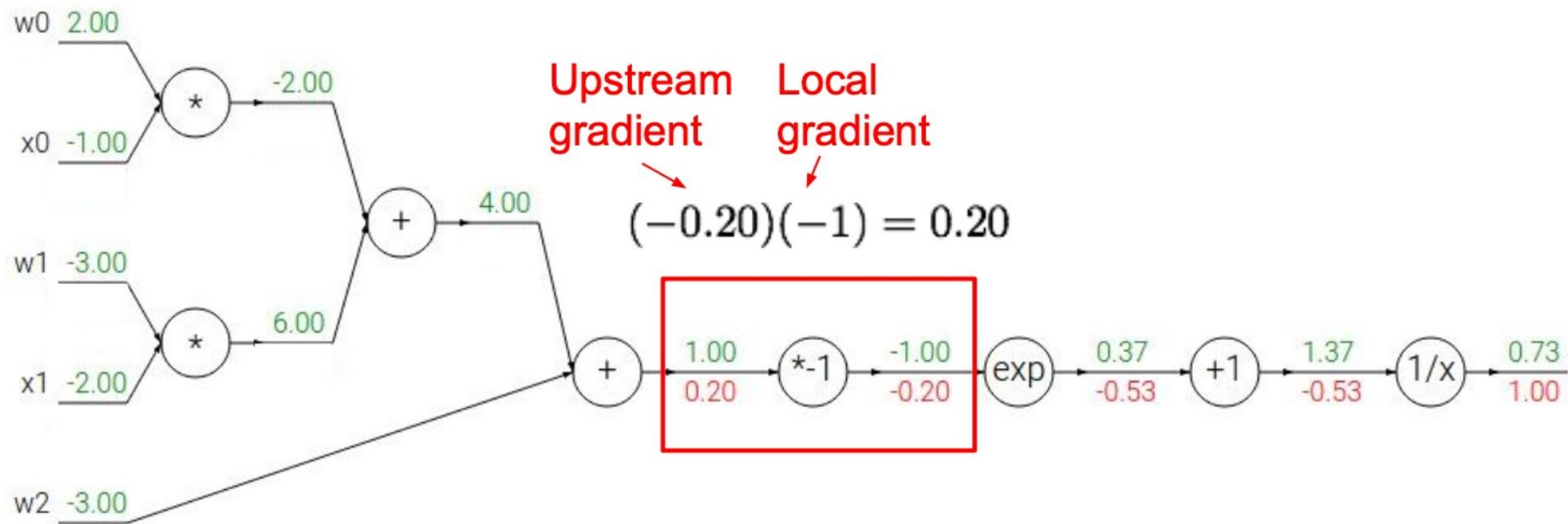
$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

$\rightarrow$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

$\rightarrow$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$\rightarrow$

$$\frac{df}{dx} = -1/x^2$$

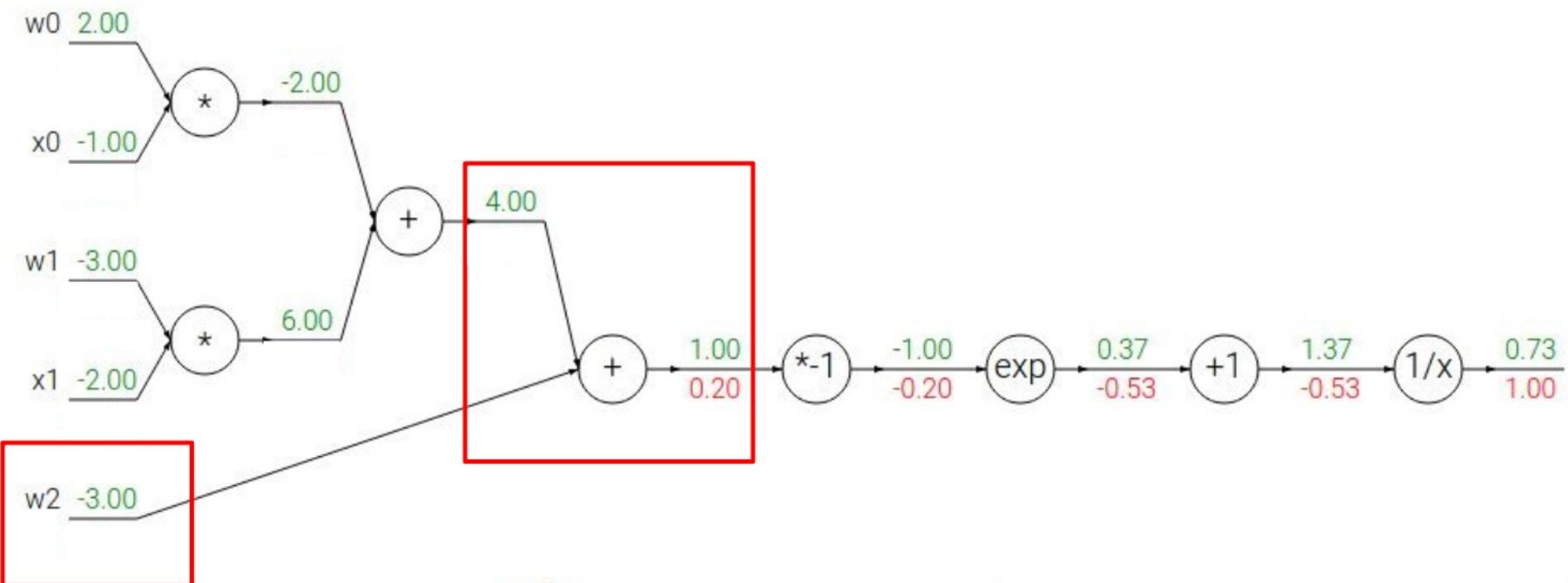
$$f_c(x) = c + x$$

$\rightarrow$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x \rightarrow$$

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow$$

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow$$

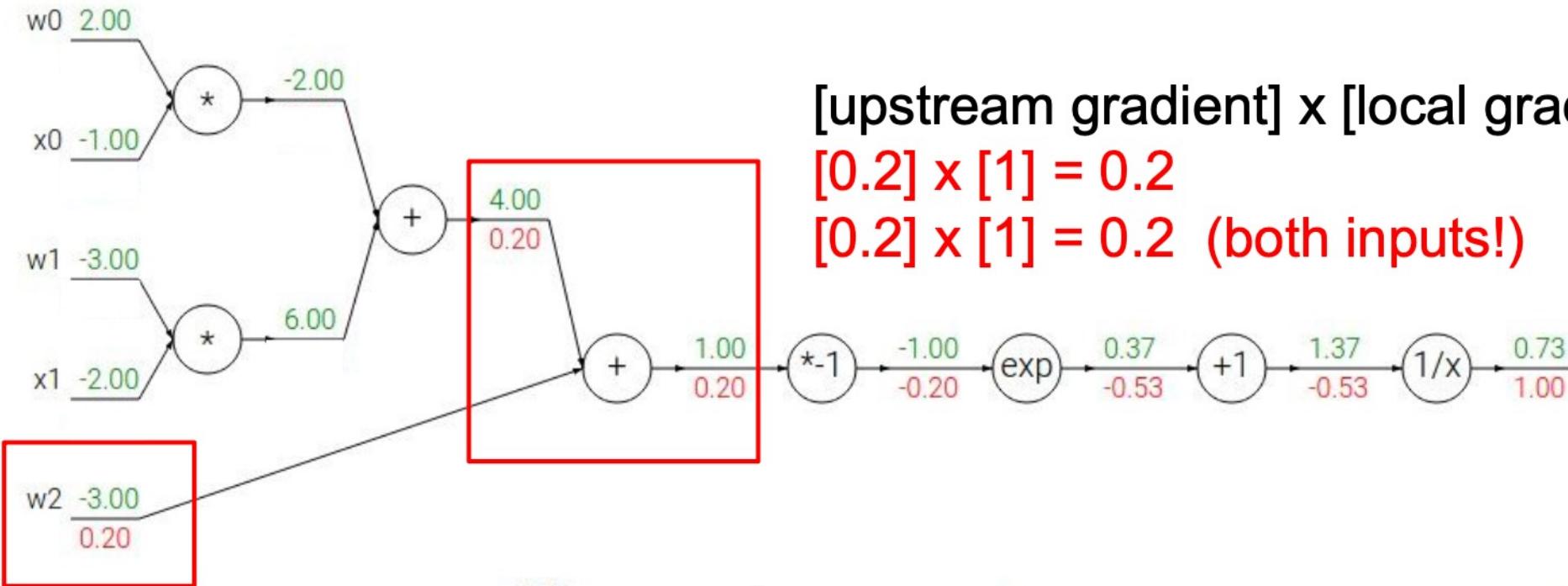
$$f_c(x) = c + x$$

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f(x) = \frac{1}{x}$$

$$f_c(x) = c + x$$

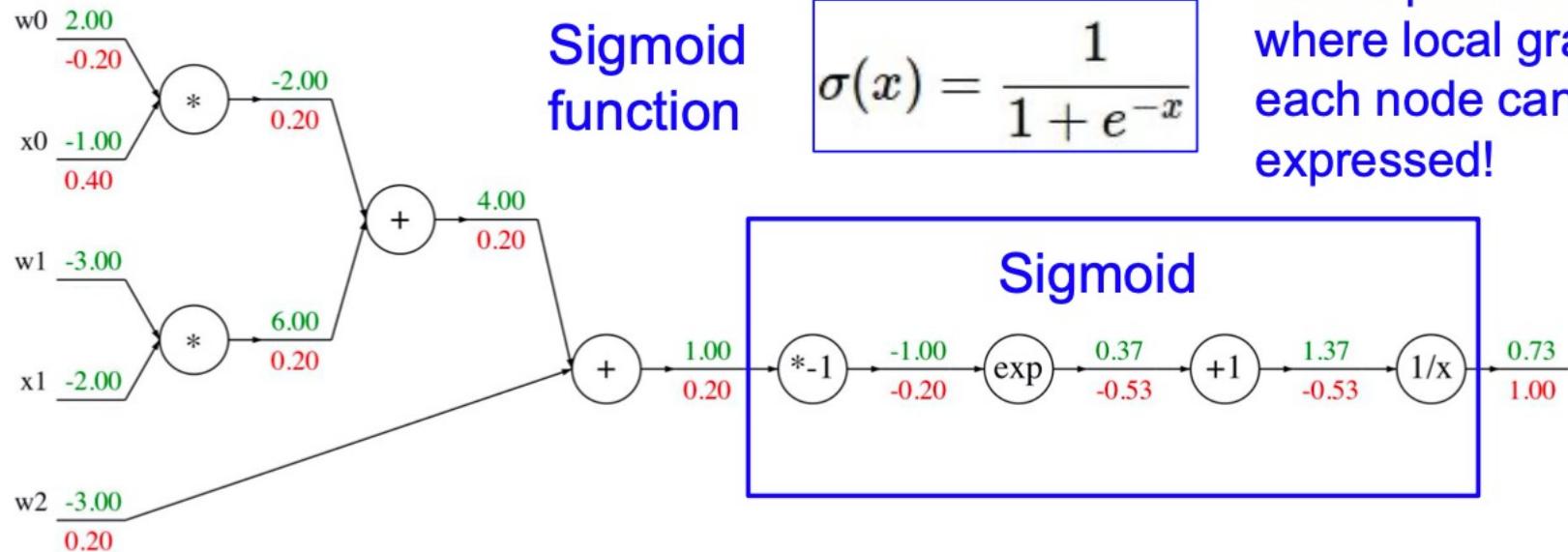
→

$$\frac{df}{dx} = -1/x^2$$

$$\frac{df}{dx} = 1$$

## Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



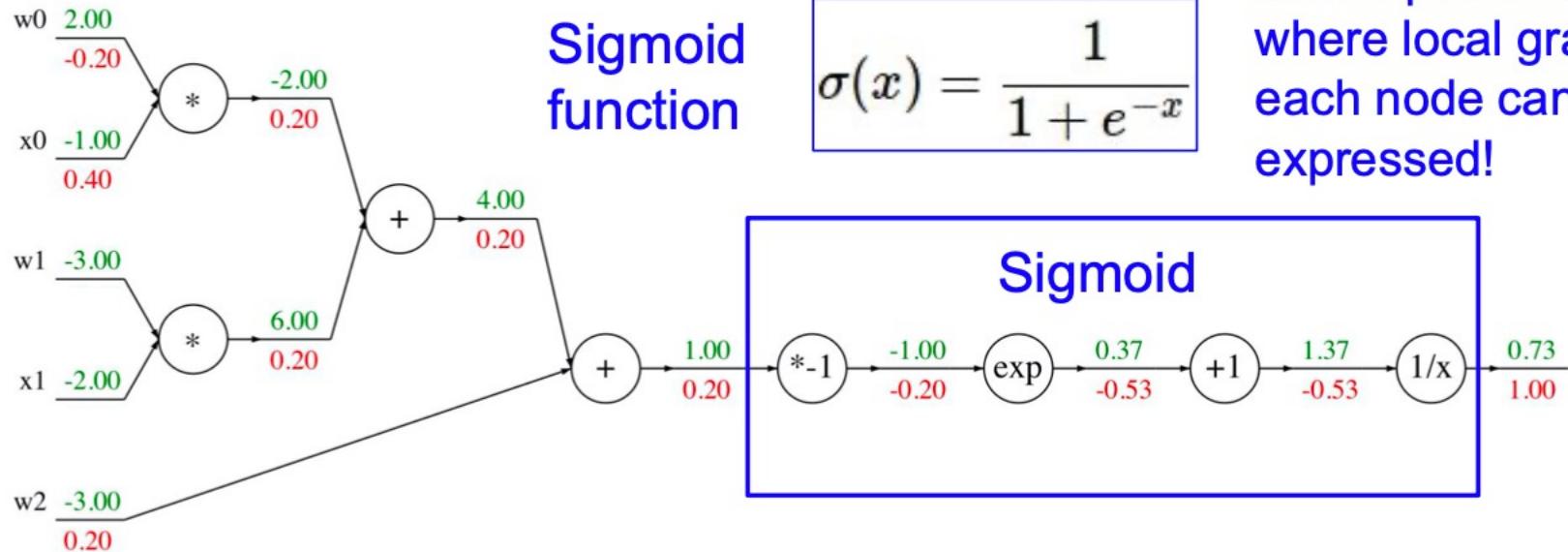
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

## Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

Sigmoid  
function

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



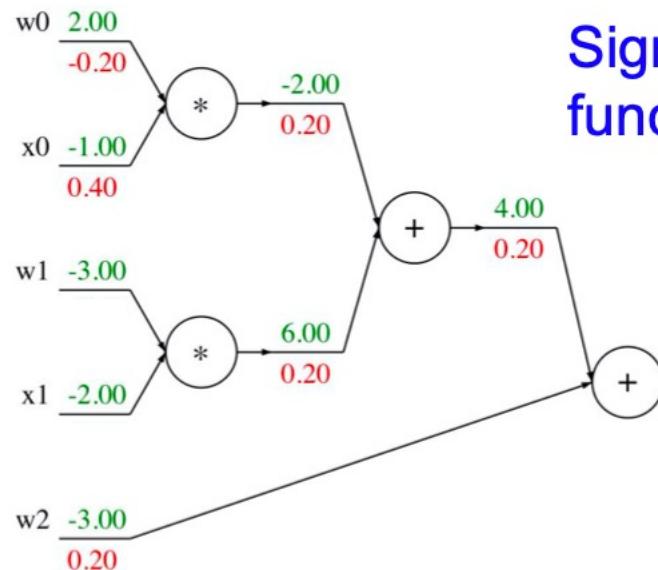
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

Another example:

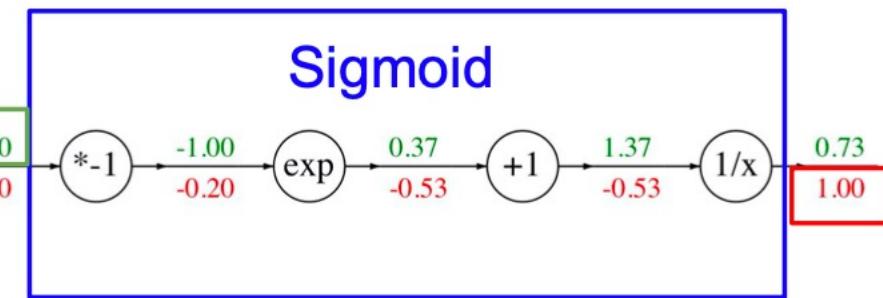
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Sigmoid function

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



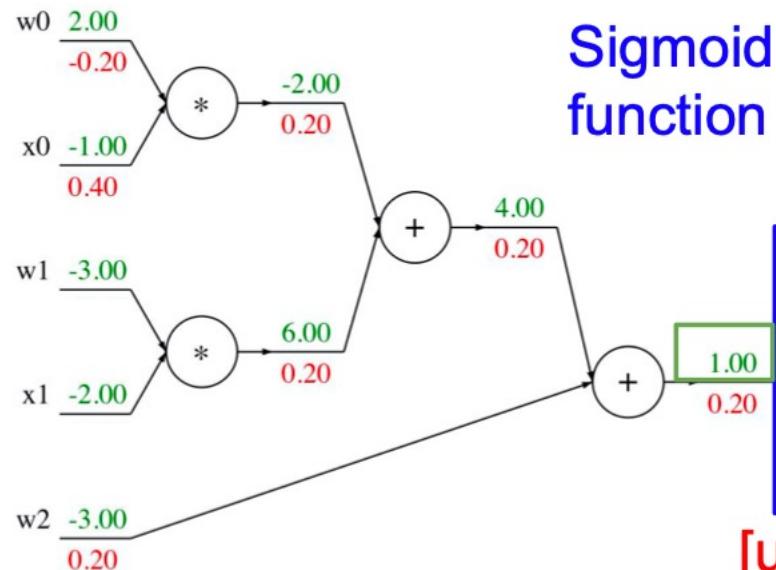
[upstream gradient]  $\times$  [local gradient]  
 $[1.00] \times [(1 - 1/(1+e^1)) (1/(1+e^1))] = 0.2$

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x)$$

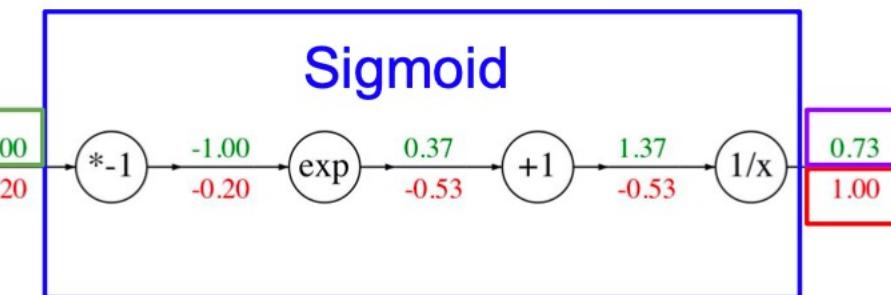
## Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Sigmoid function

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



[upstream gradient]  $\times$  [local gradient]  
 $[1.00] \times [(1 - 0.73)(0.73)] = 0.2$

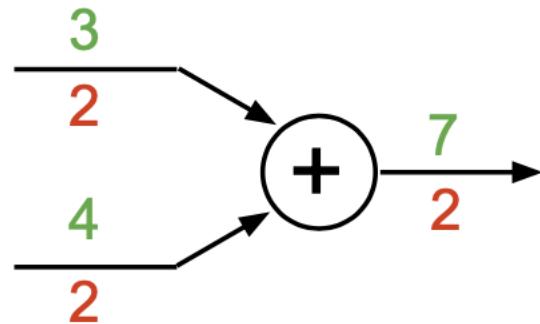
Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

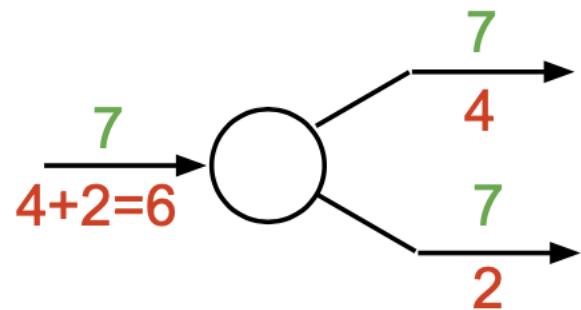
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

# Patterns in gradient flow

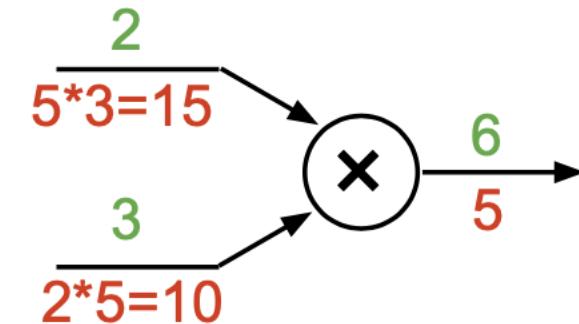
**add gate:** gradient distributor



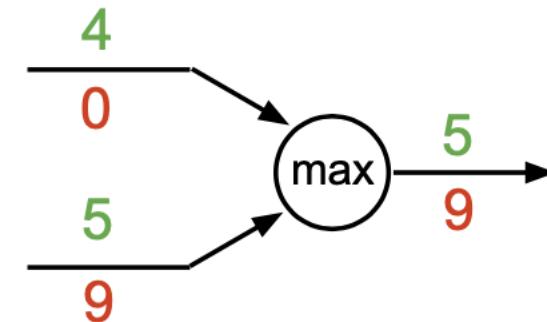
**copy gate:** gradient adder



**mul gate:** “swap multiplier”



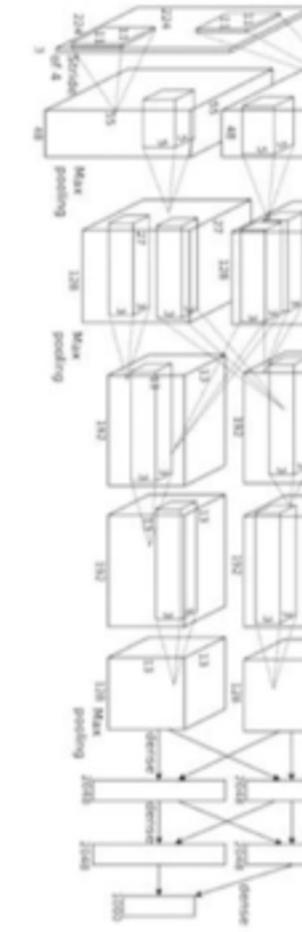
**max gate:** gradient router



Can be applied to arbitrarily complex deep networks

Convolutional Network  
(AlexNet)

input image  
weights  
loss



# Overall Steps of BP

After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

- i) Feed-forward computation
- ii) Backpropagation to the output layer
- iii) Backpropagation to the hidden layer
- iv) Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small.

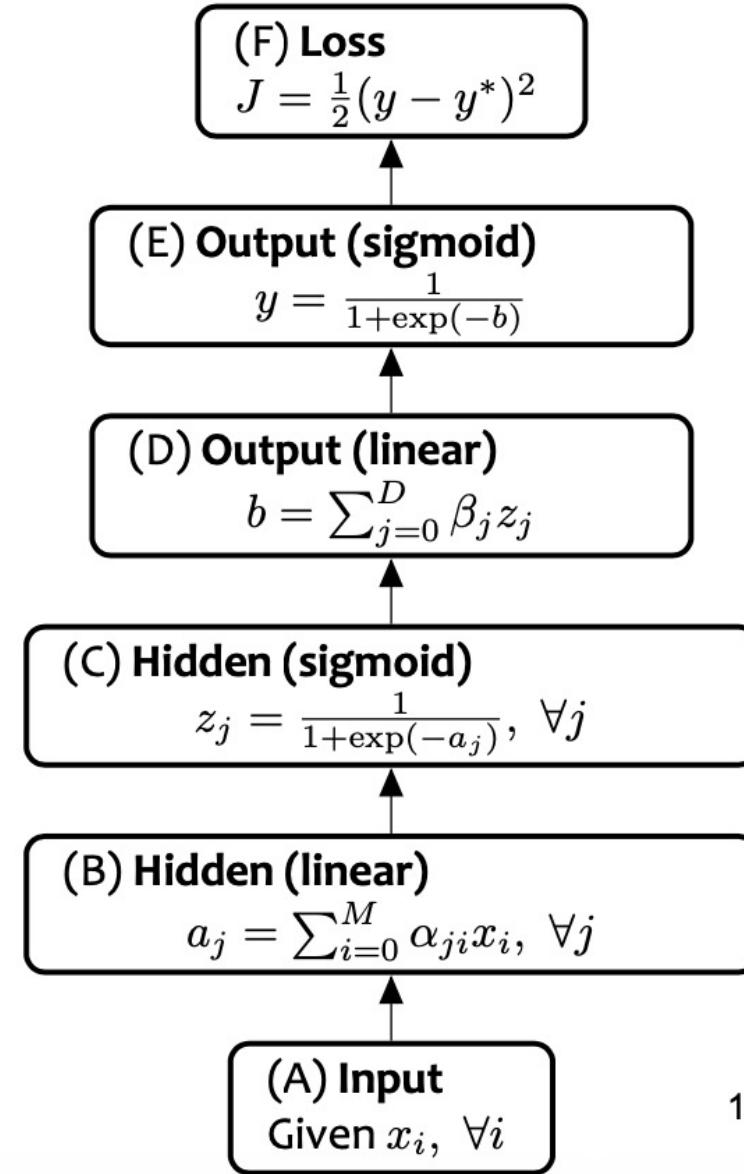
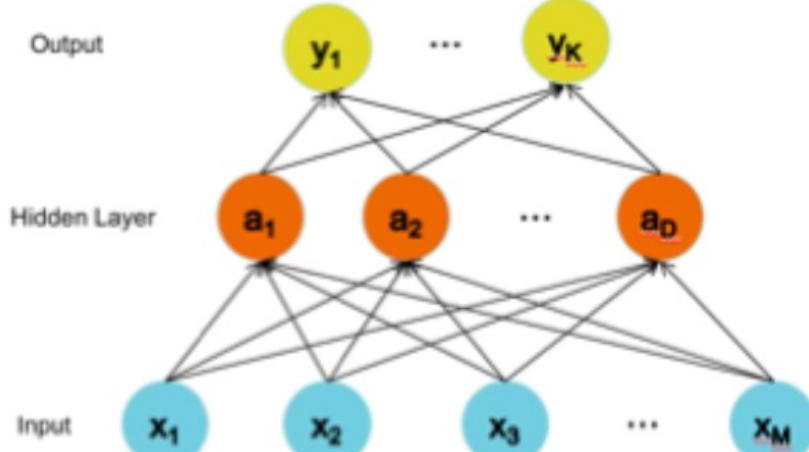
# Multilayer Neural Networks

## Network Design

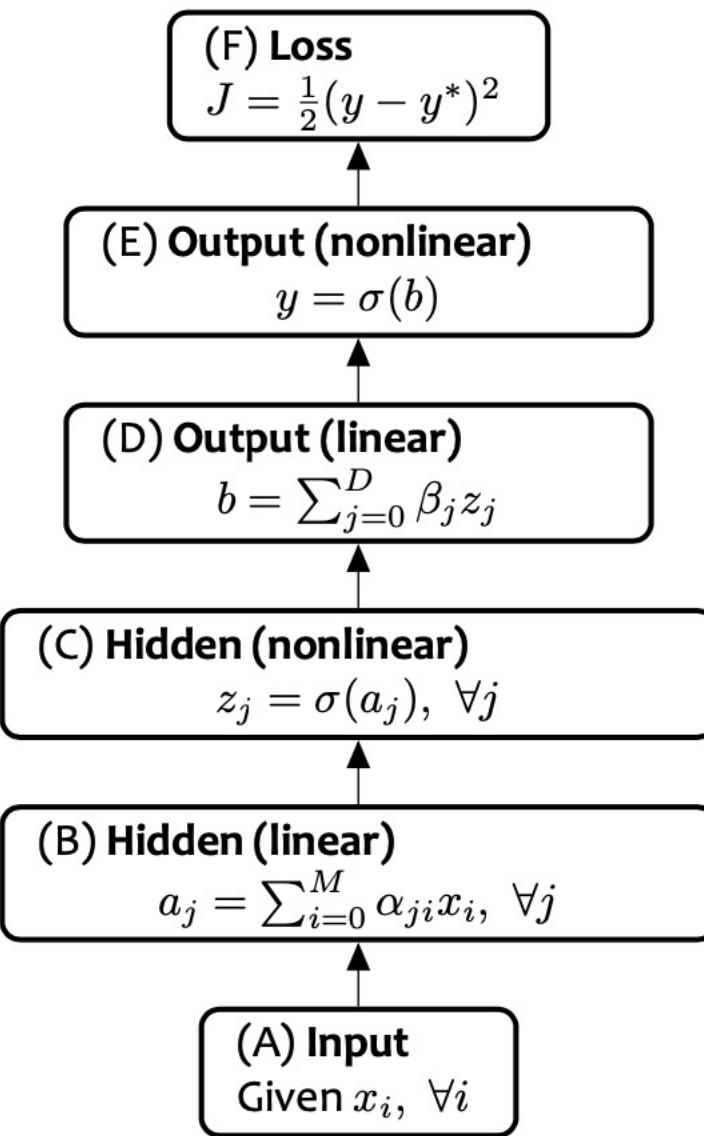
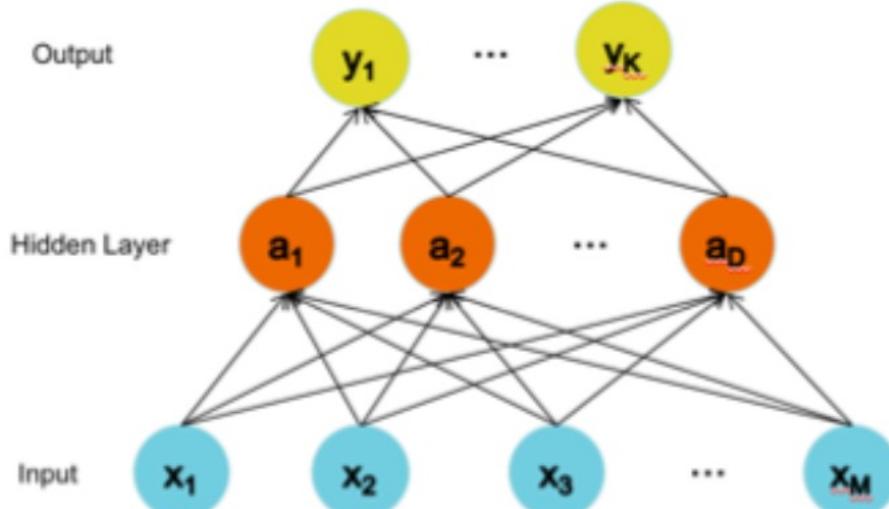
# Neural Network Architectures

- Even for a basic Multilayer Neural Network, there are many design decisions to make:
  - 1) Number of hidden layers (depth of the network)
  - 2) Number of neurons per hidden layer (width)
  - 3) Type of activation functions
  - 4) Form of objective functions

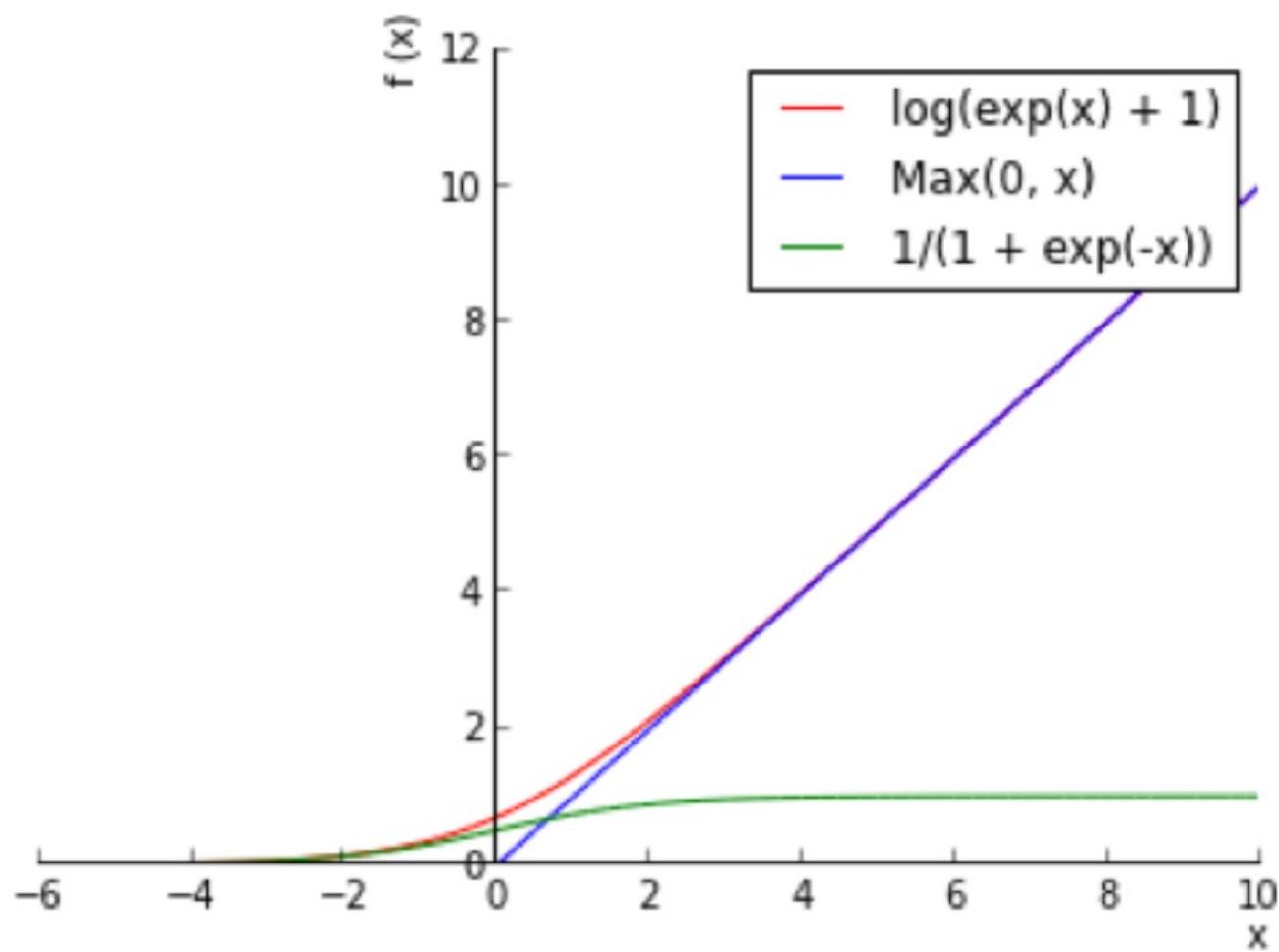
## Example: Neural Network with sigmoid activation functions



## Neural Network with arbitrary nonlinear activation functions



# ReLU often used in computer vision tasks

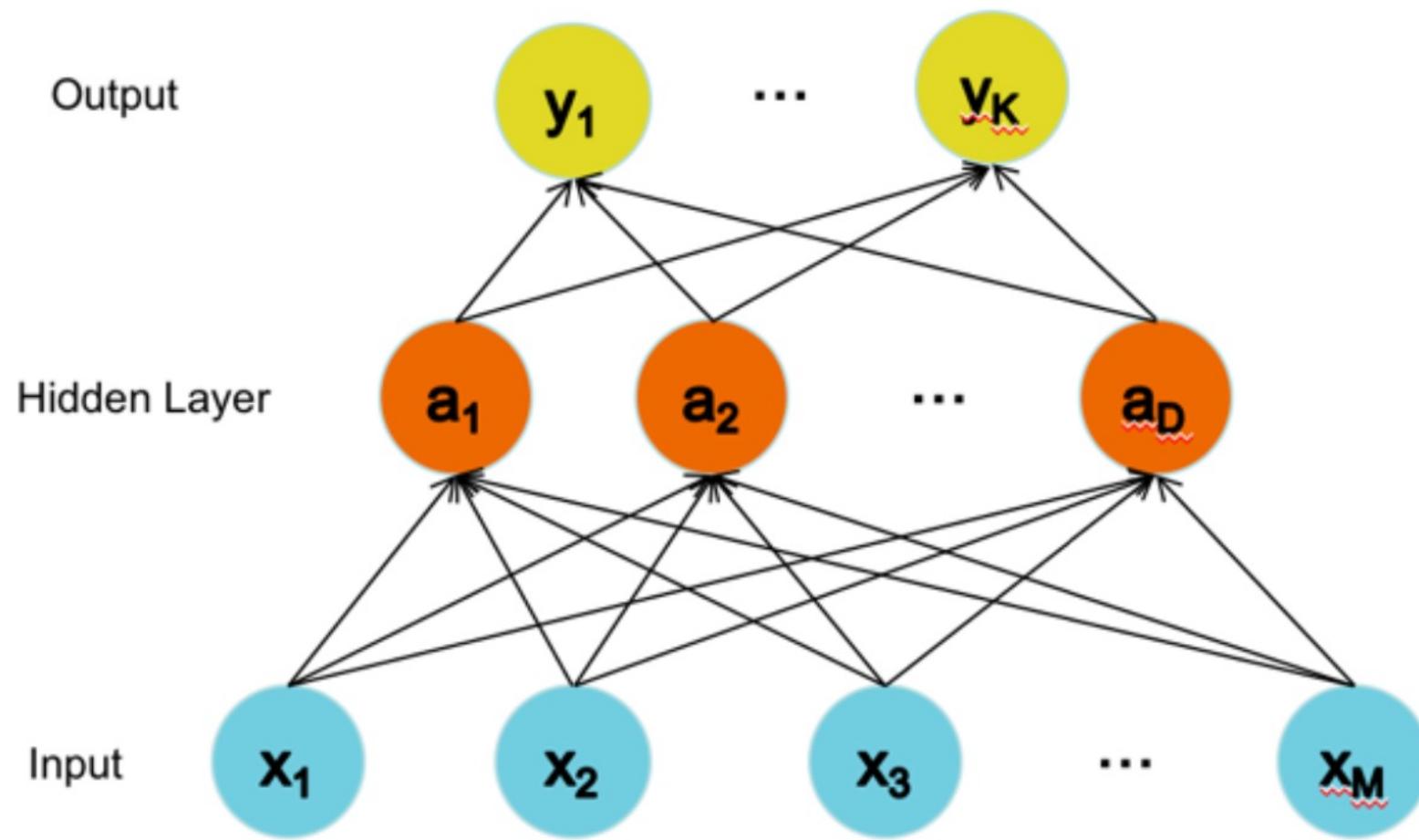


# Loss Functions for NN

- **Regression:**
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- **Classification:**
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional “softmax” layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{1-y}$

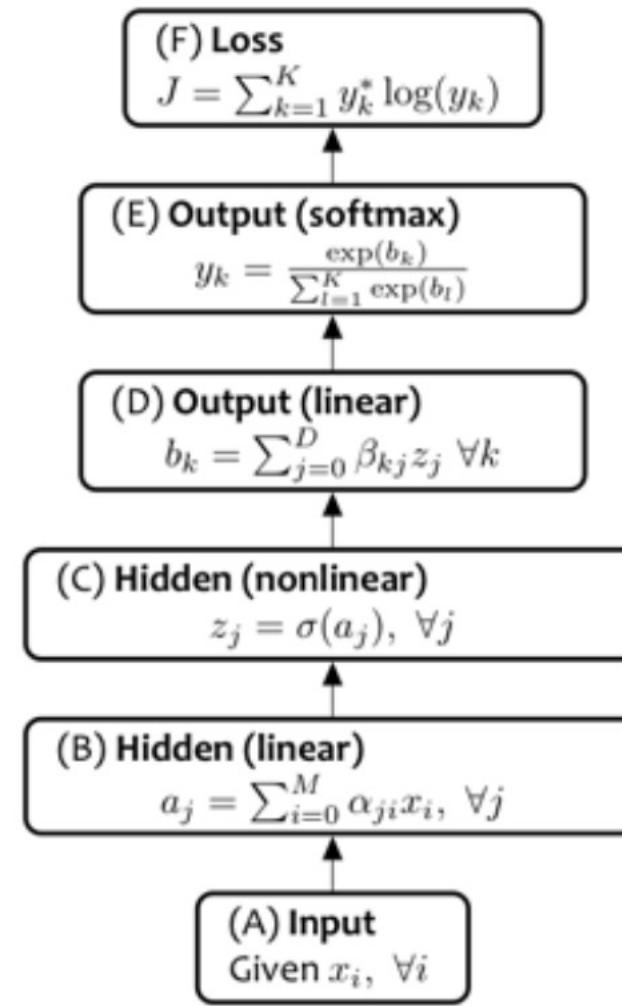
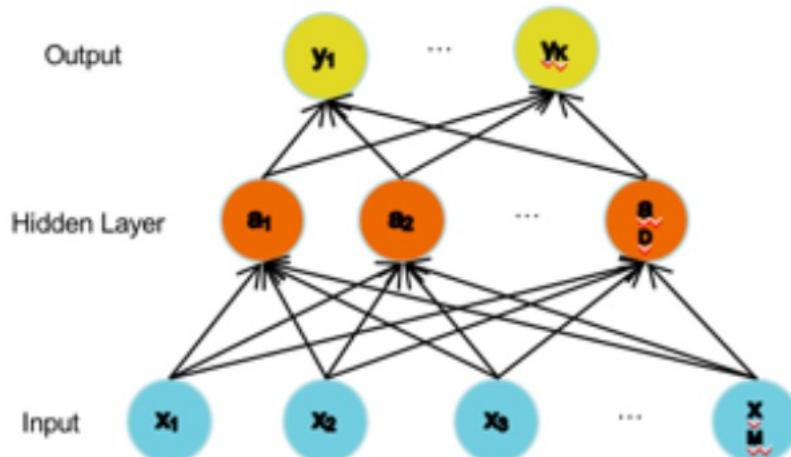
# Multi-class Output



# Multi-class Output

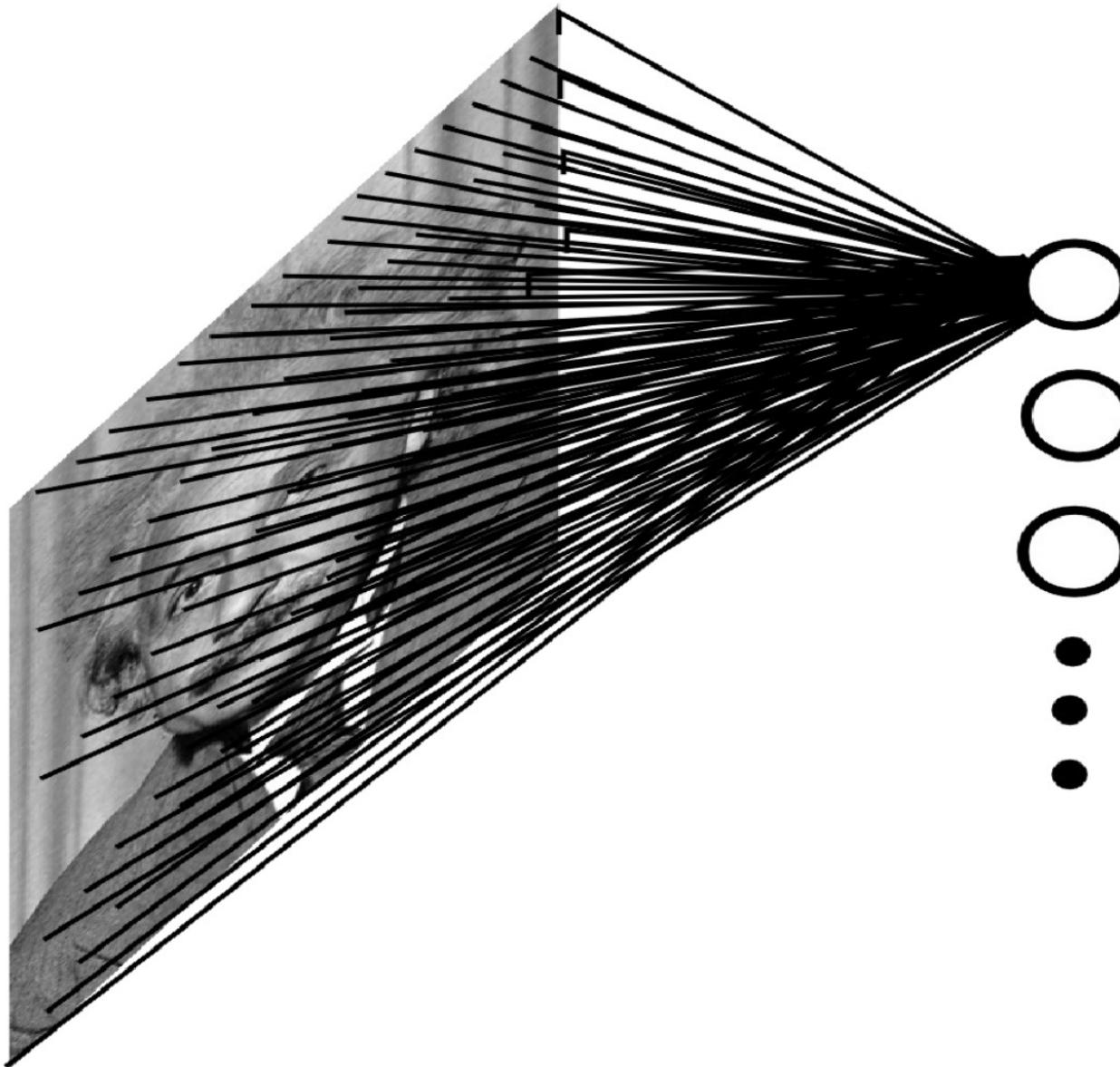
Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$

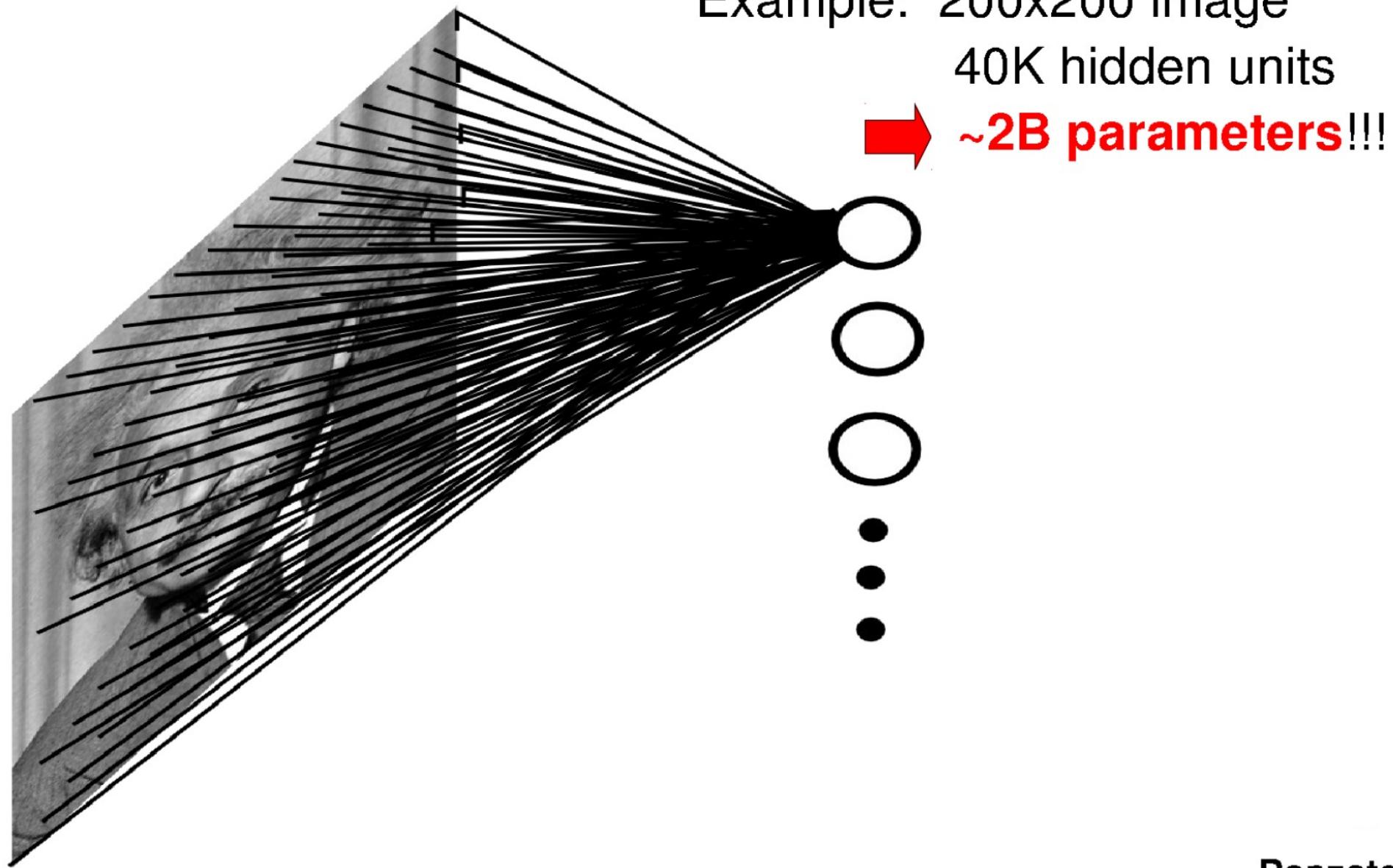


# Convolutional Neural Network

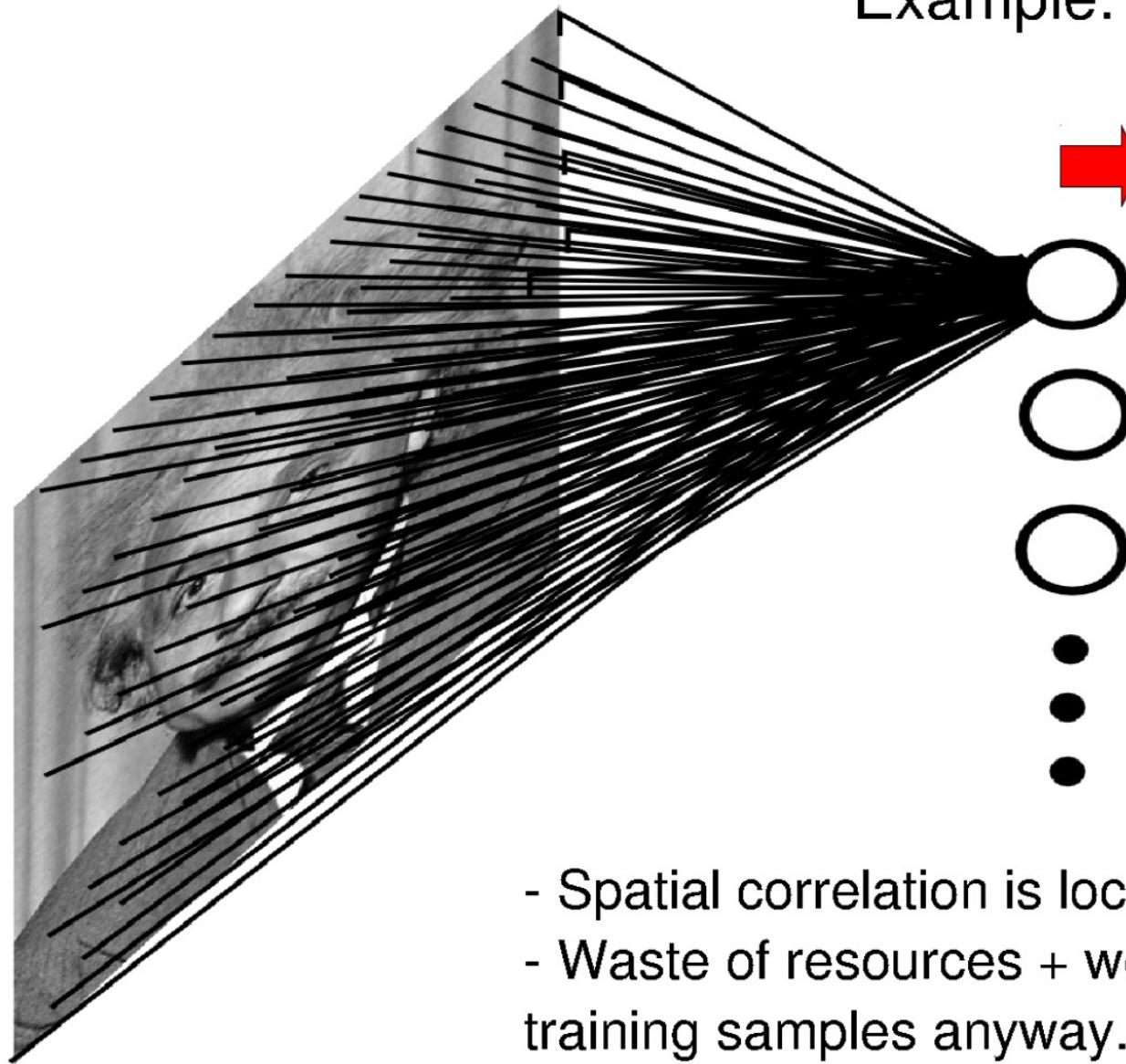
# Images as input to neural networks



# Images as input to neural networks



# Images as input to neural networks

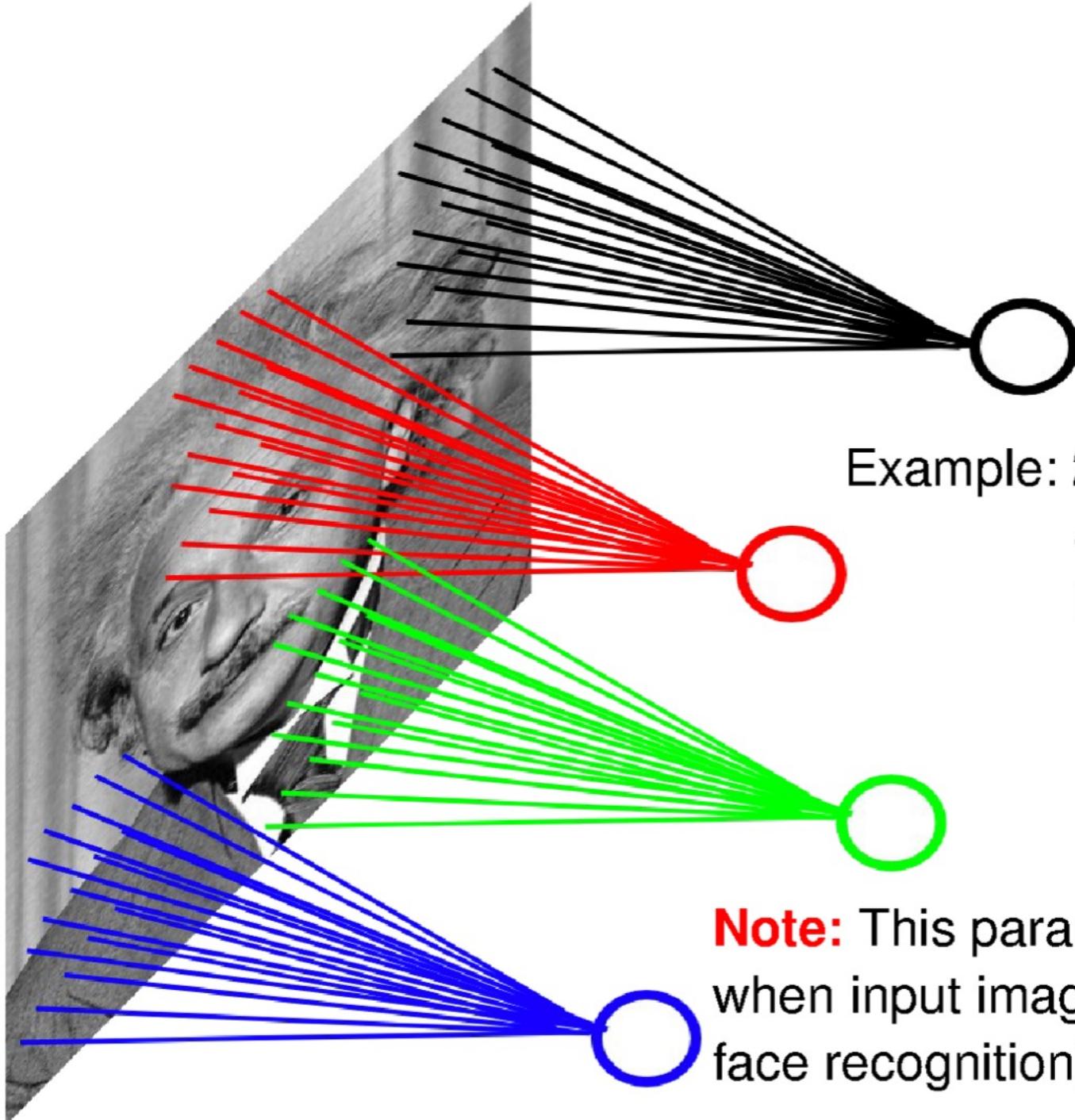


Example: 200x200 image  
40K hidden units  
→ **~2B parameters!!!**

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

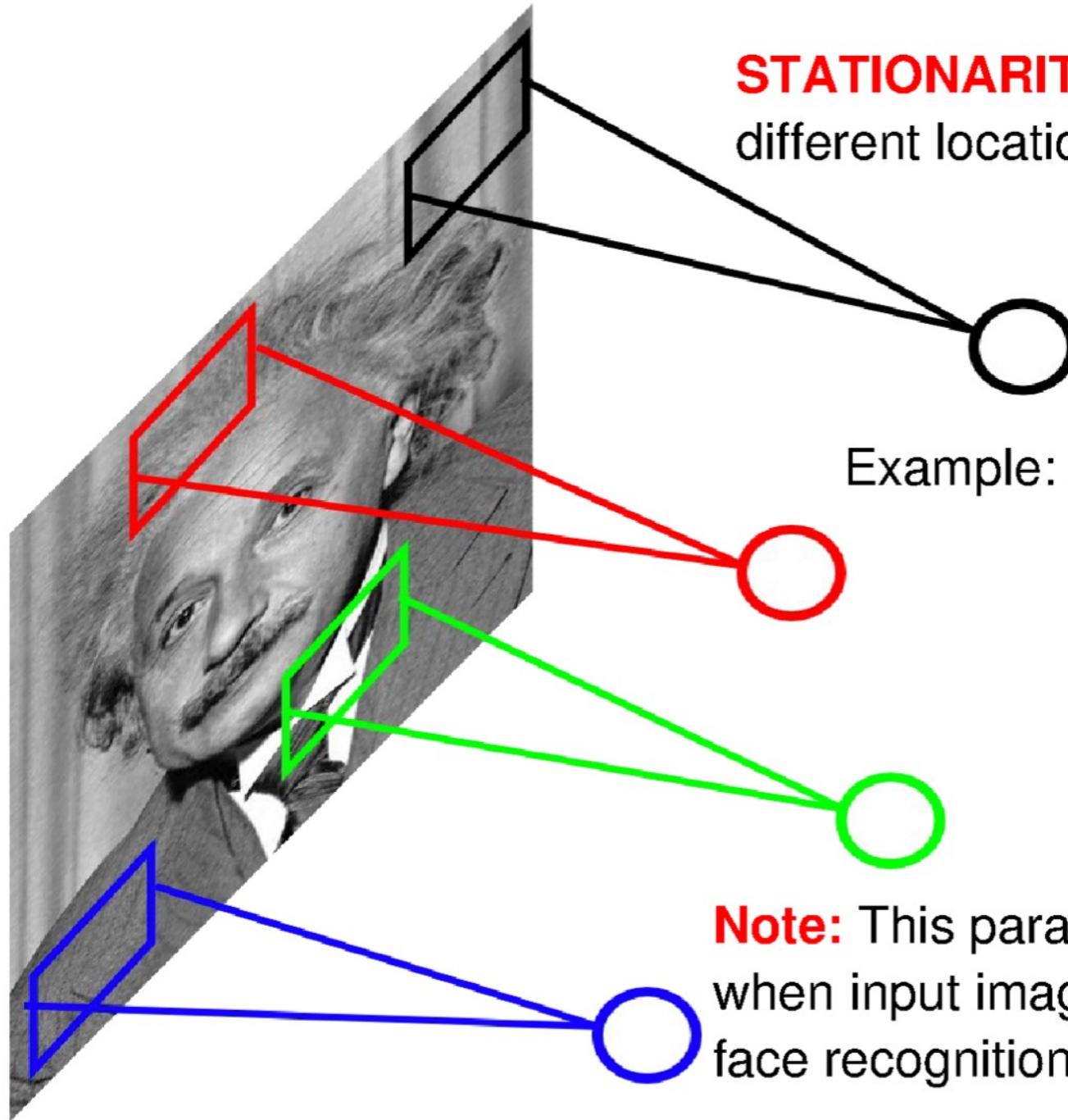
# Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local -> global.



Example: 200x200 image  
40K hidden units  
Filter size: 10x10  
4M parameters

**Note:** This parameterization is good  
when input image is registered (e.g.,  
face recognition).



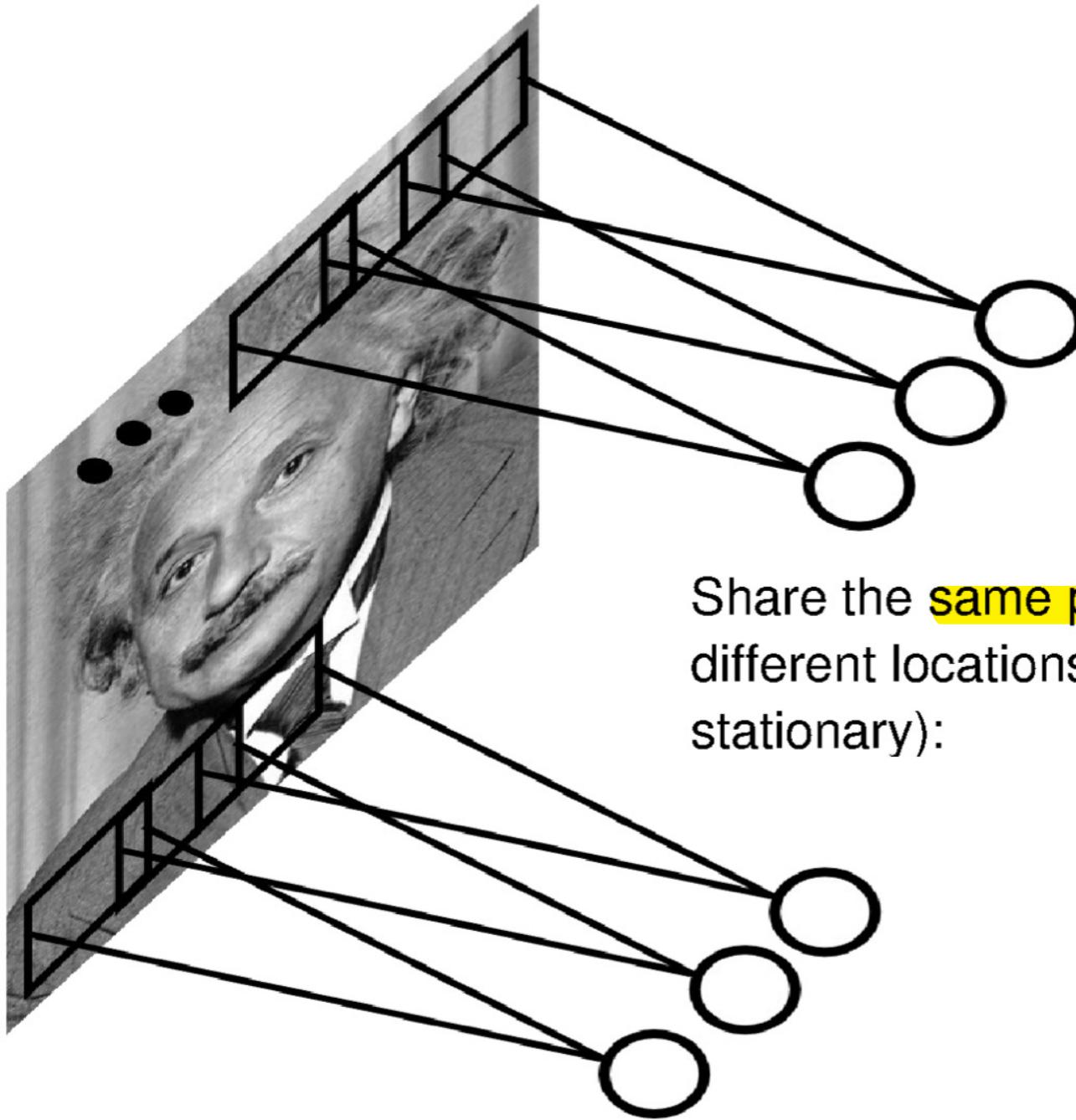
**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image  
40K hidden units  
Filter size: 10x10  
4M parameters

**Note:** This parameterization is good when input image is registered (e.g.,  
face recognition).

# Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local  $\rightarrow$  global.
- **Parameter sharing**
  - ‘Tied weights’ – use **same weights** for more than one perceptron in the neural network.
  - Leads to *equivariant representation*
    - If input changes (e.g., translates), then output changes similarly



Share the **same parameters** across  
different locations (assuming input is  
stationary):

# Filtering remainder: Correlation (rotated convolution)

$$f[\cdot, \cdot] \quad \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$I[., .]$$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

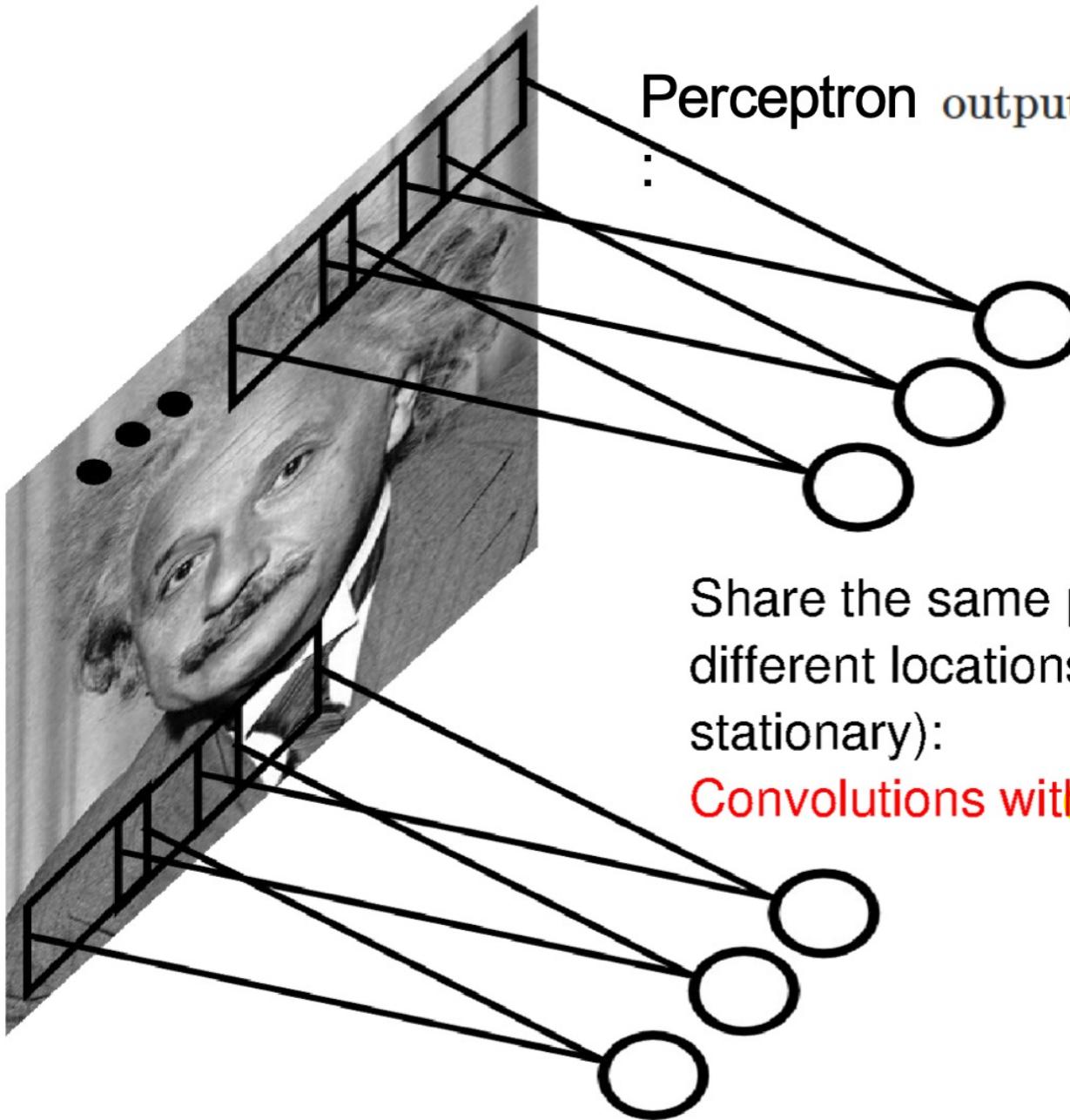
$$h[., .]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

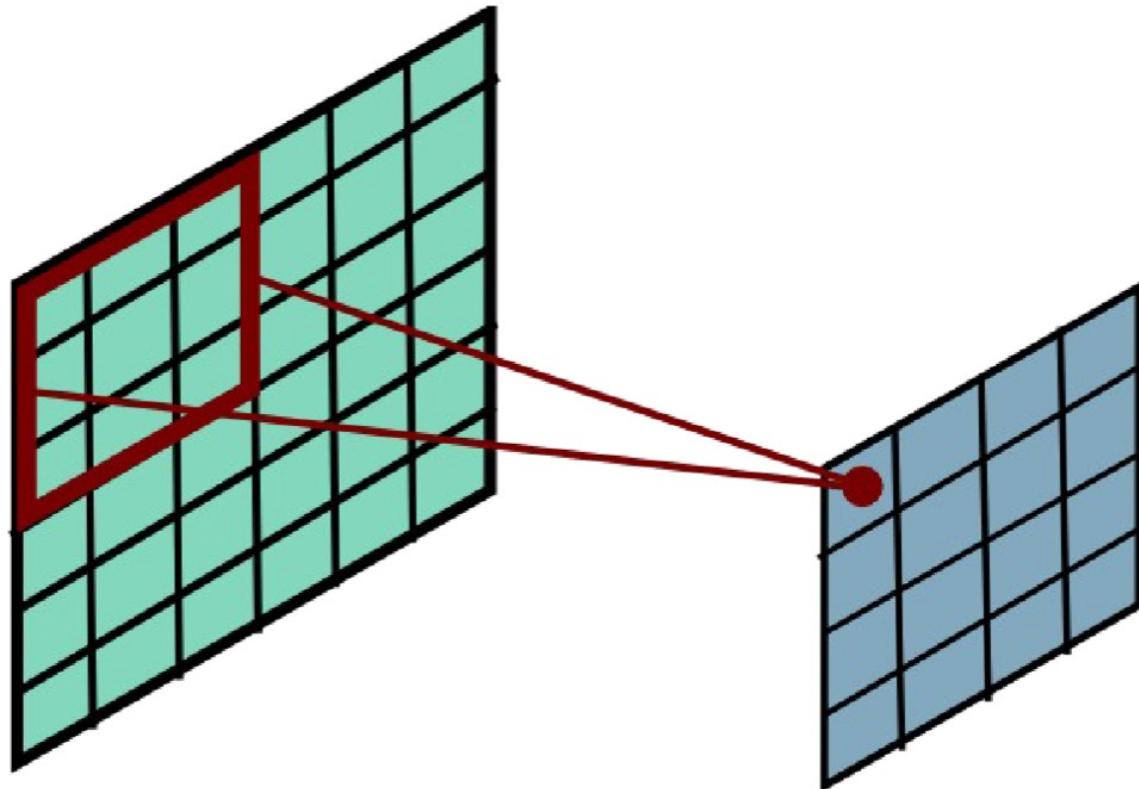
$$h[m, n] = \sum_{k,l} f[k, l] I[m+k, n+l]$$

Credit: S. Seitz

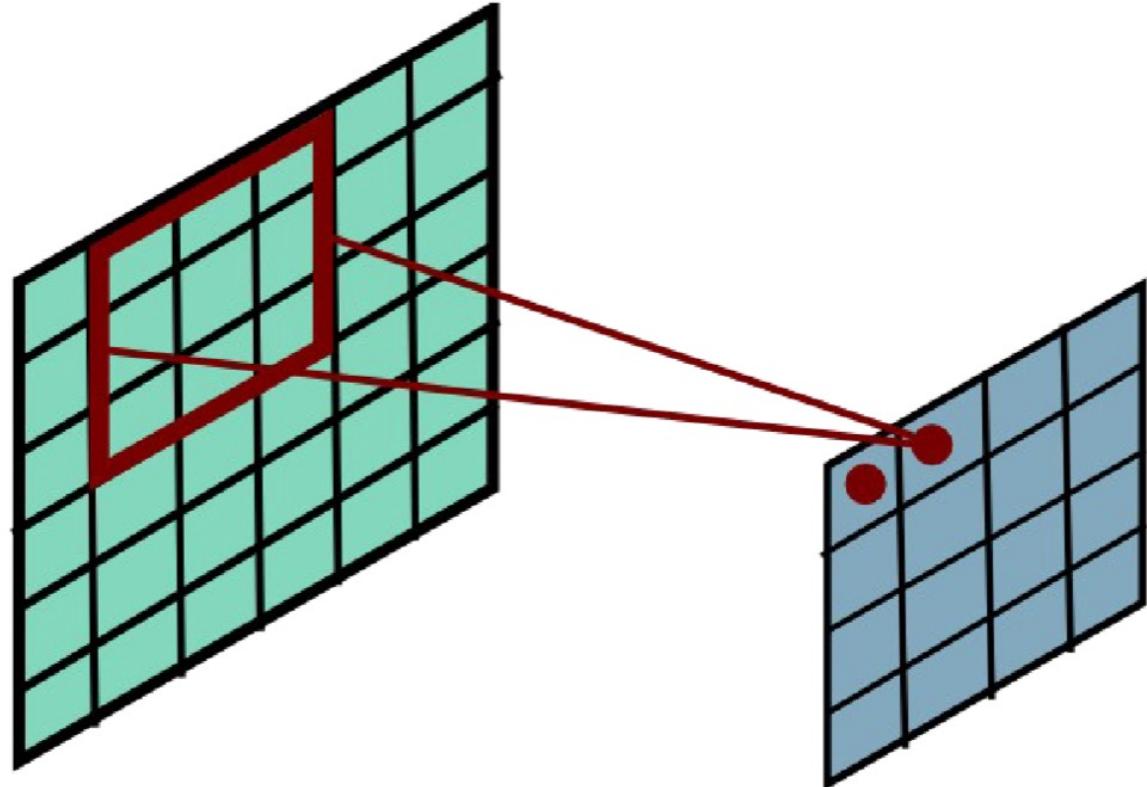
# Convolutional Layer



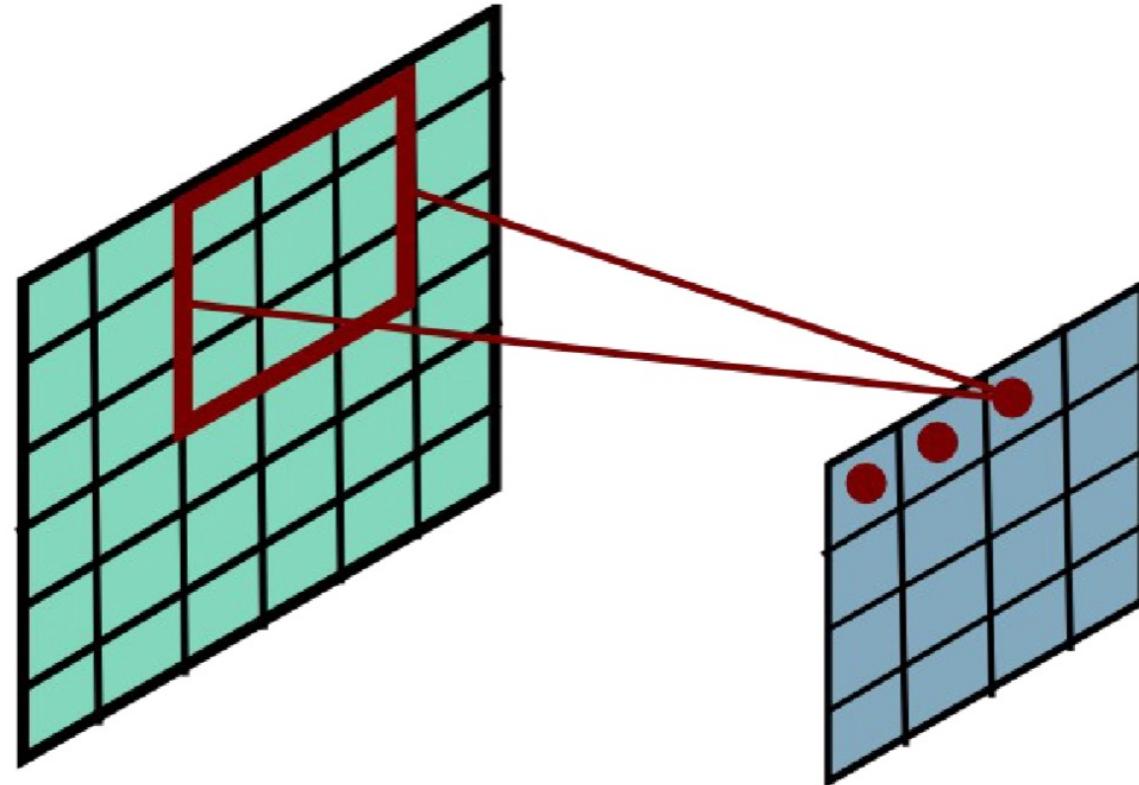
# Convolutional Layer



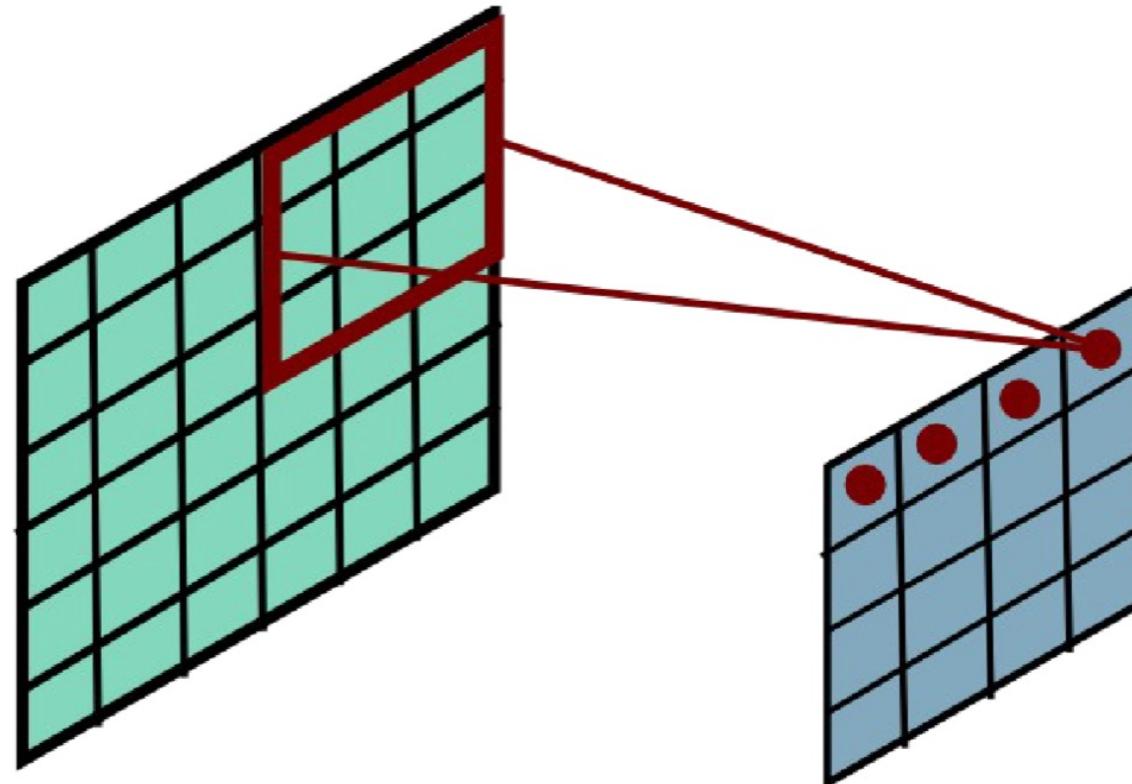
# Convolutional Layer



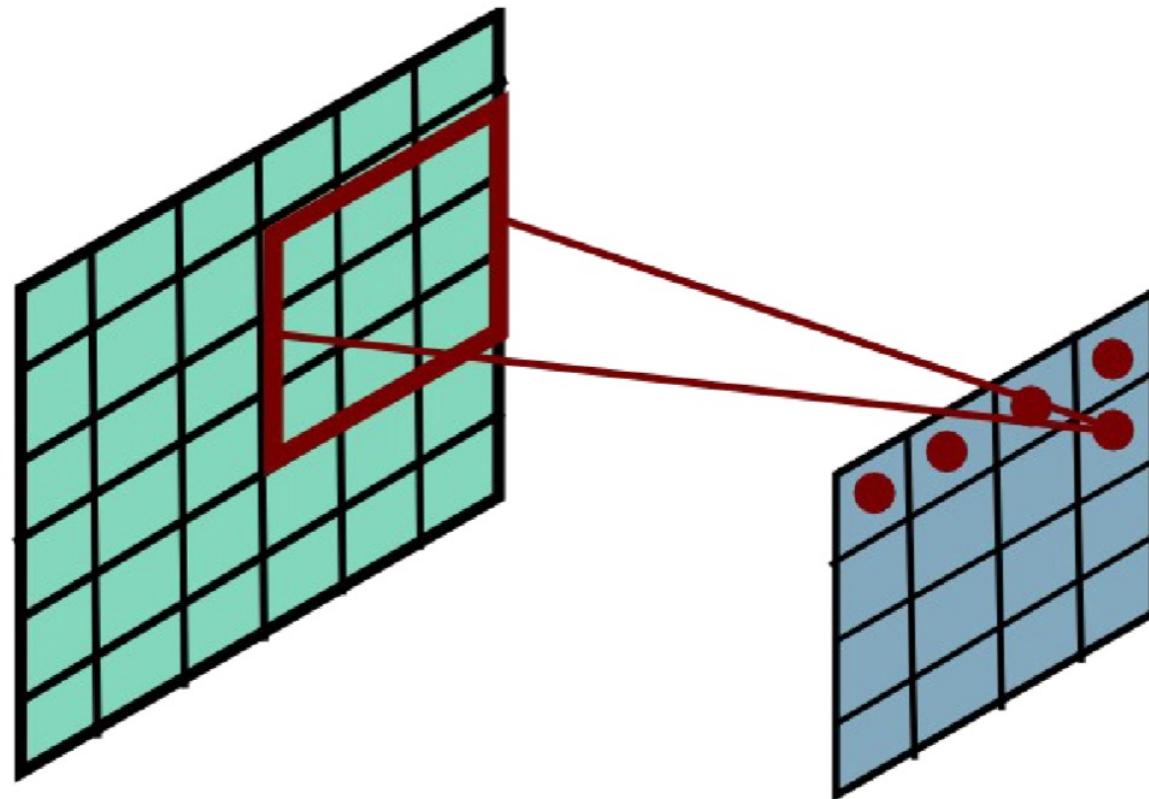
# Convolutional Layer



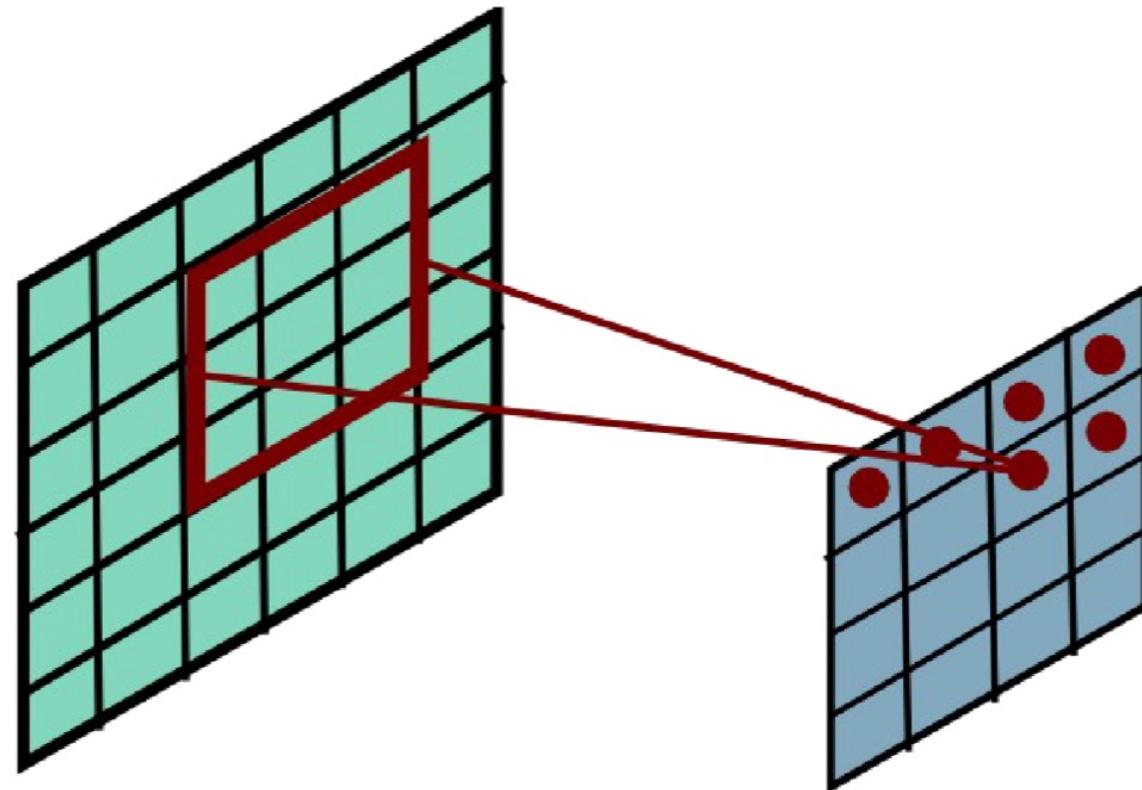
# Convolutional Layer



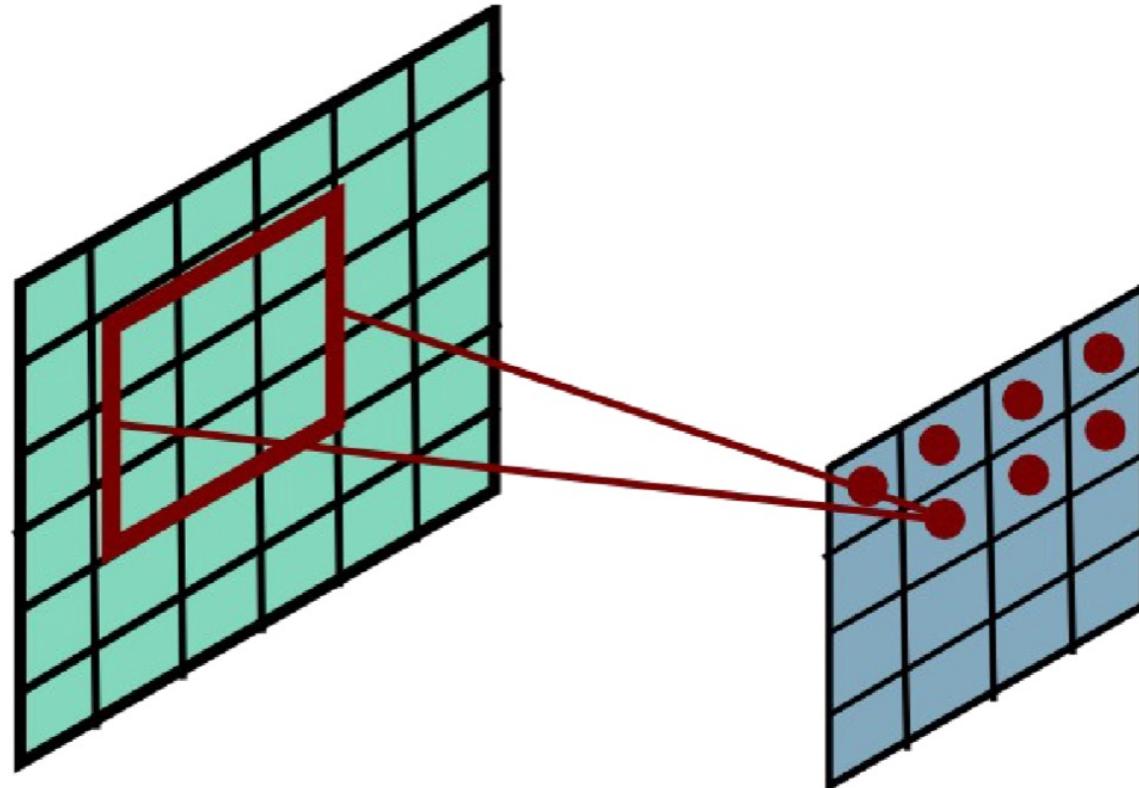
# Convolutional Layer



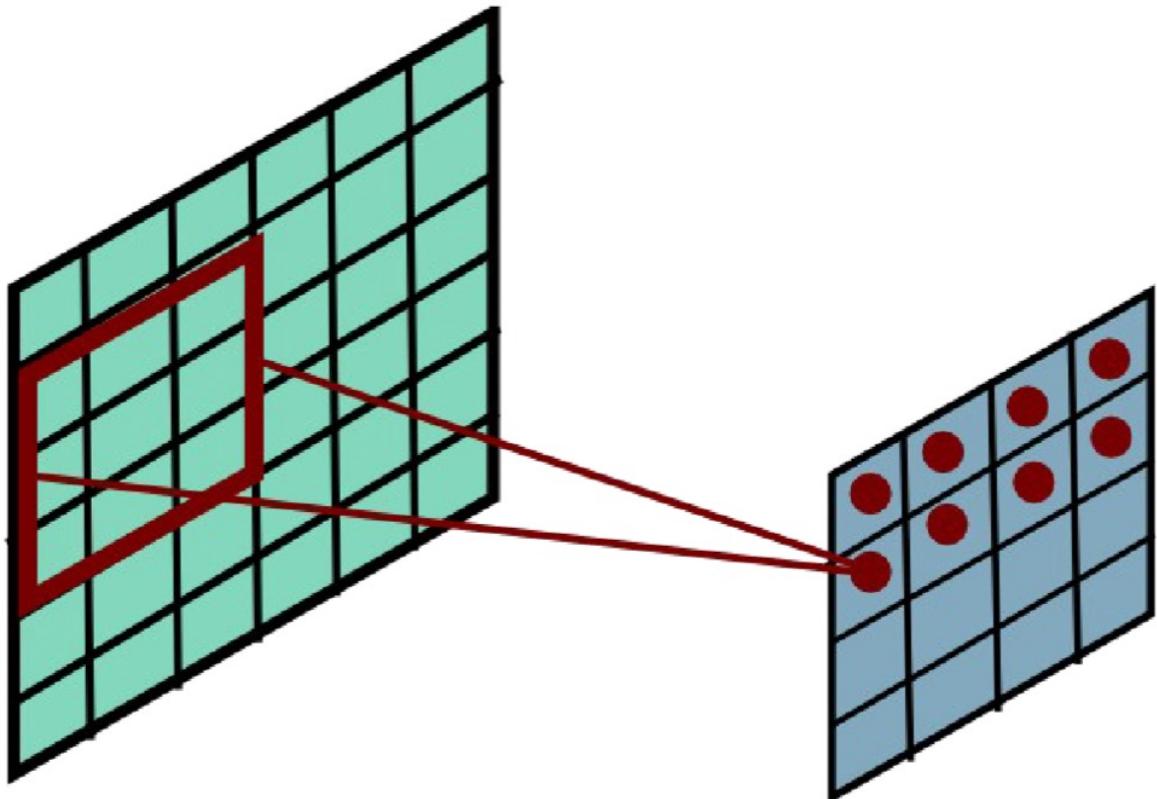
# Convolutional Layer



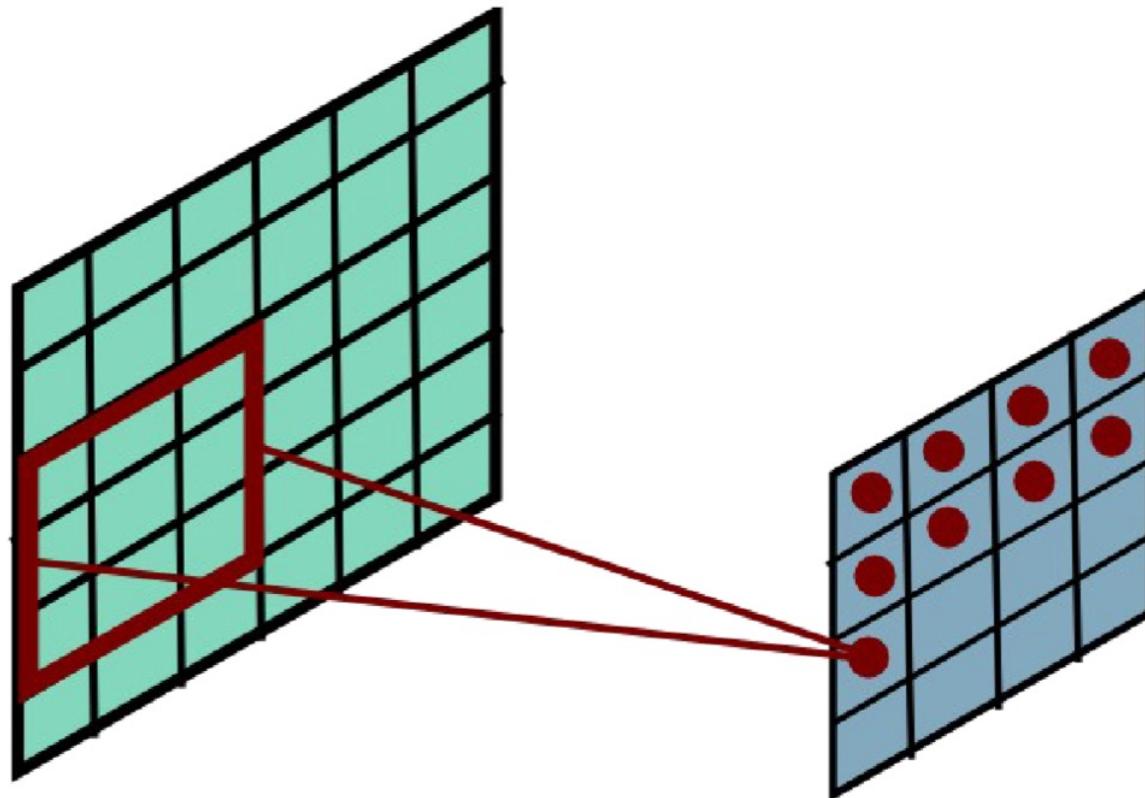
# Convolutional Layer



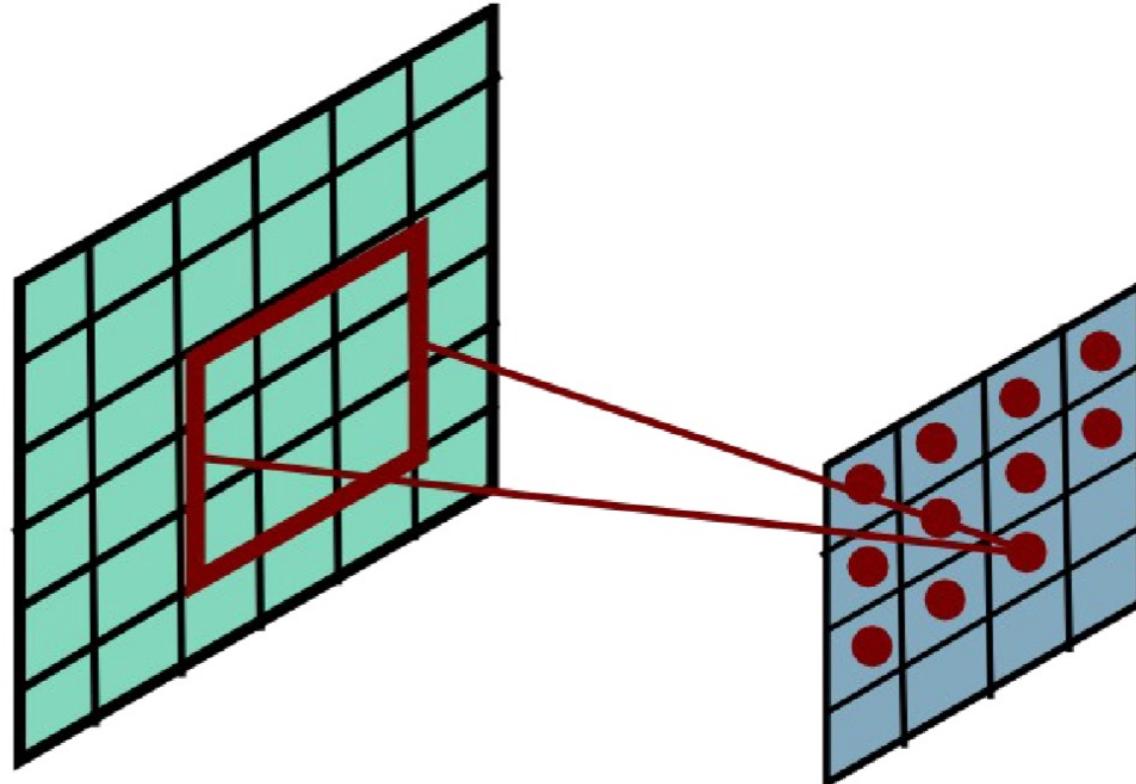
# Convolutional Layer



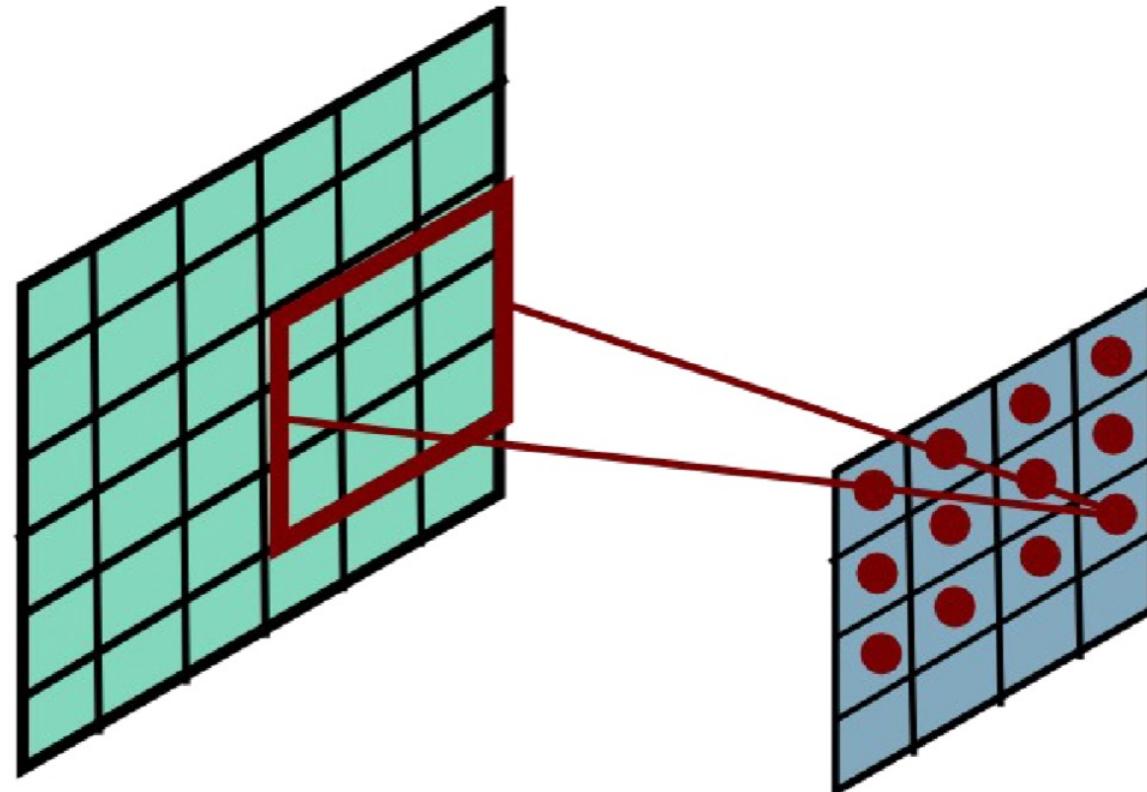
# Convolutional Layer



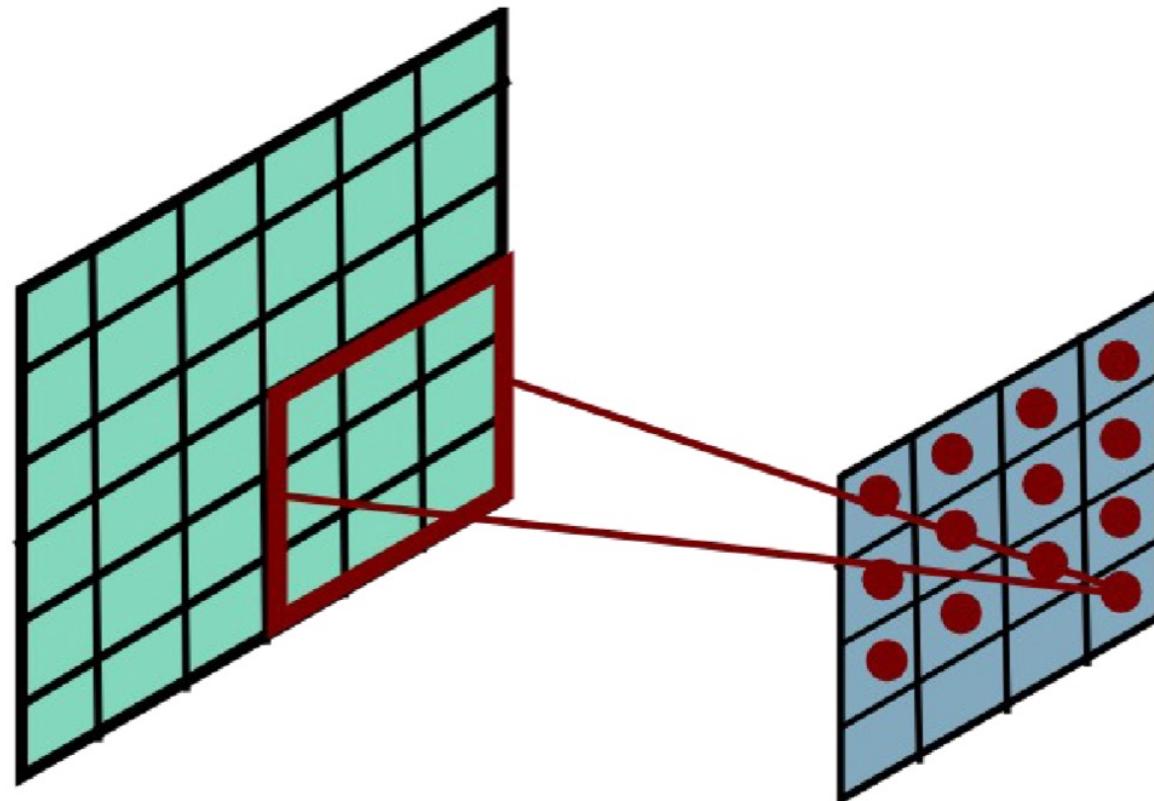
# Convolutional Layer



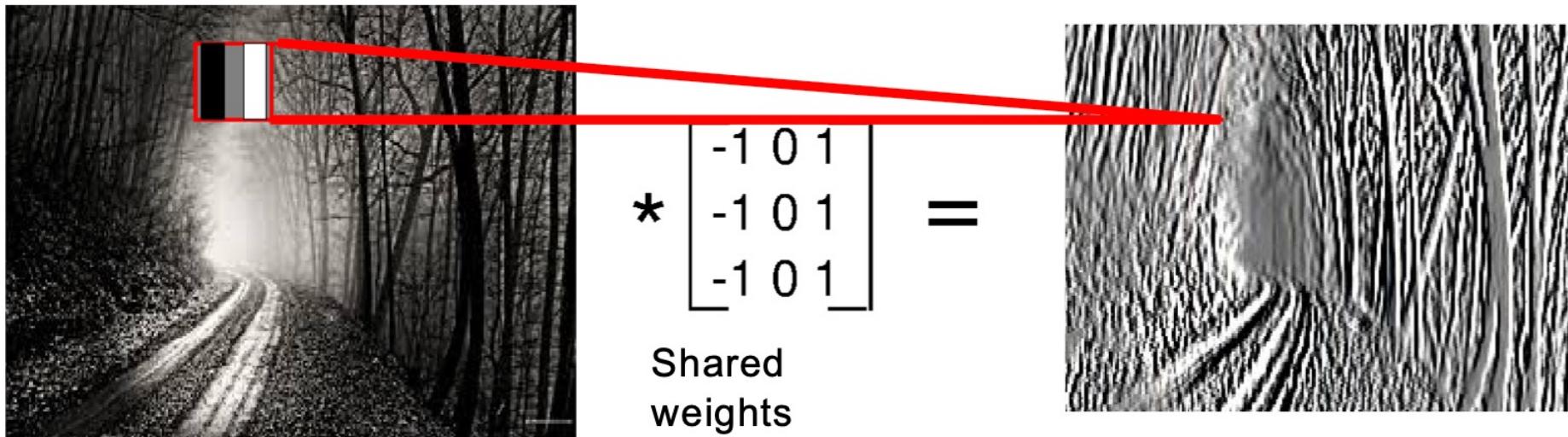
# Convolutional Layer



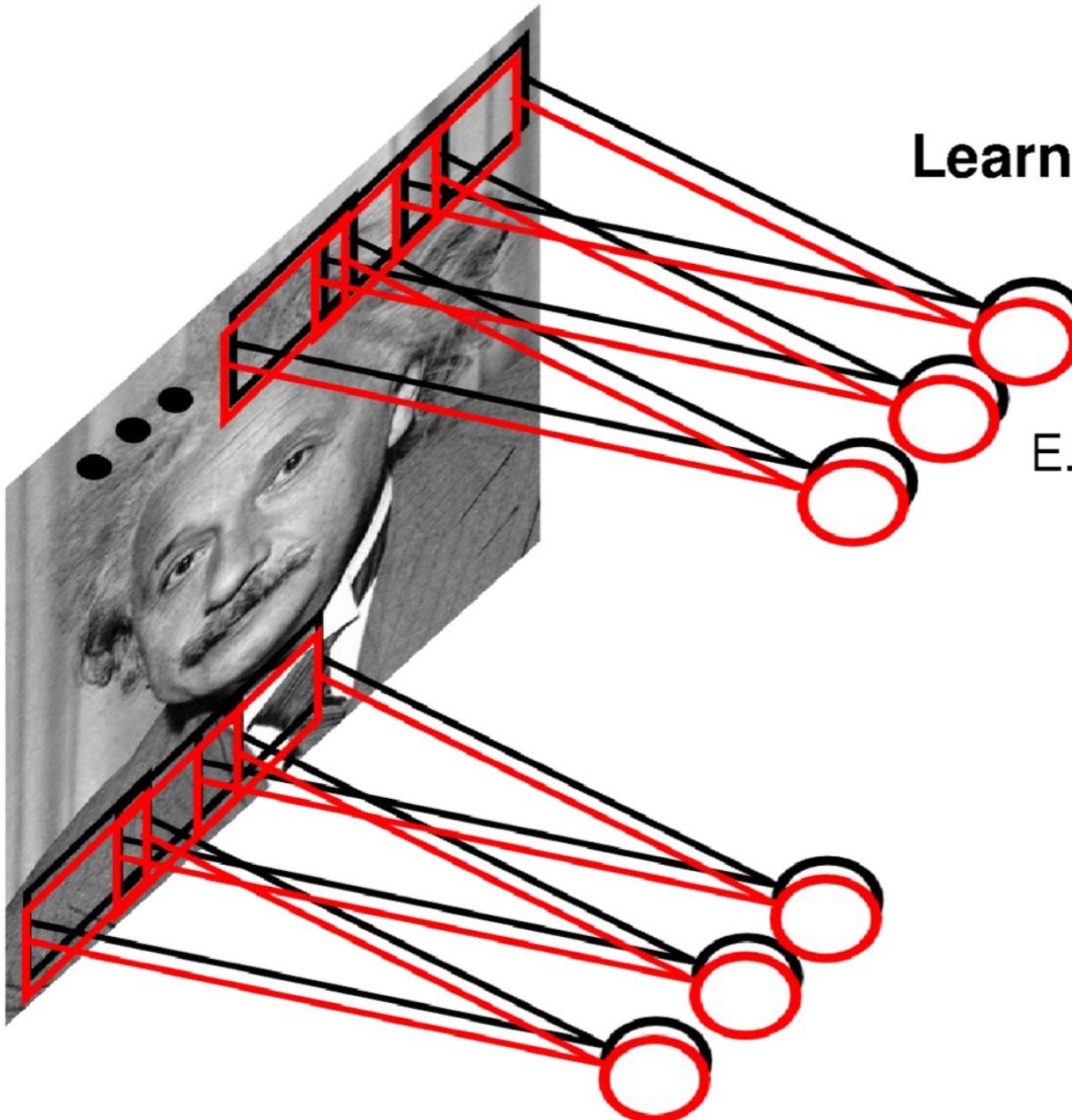
# Convolutional Layer



# Convolutional Layer



# Convolutional Layer

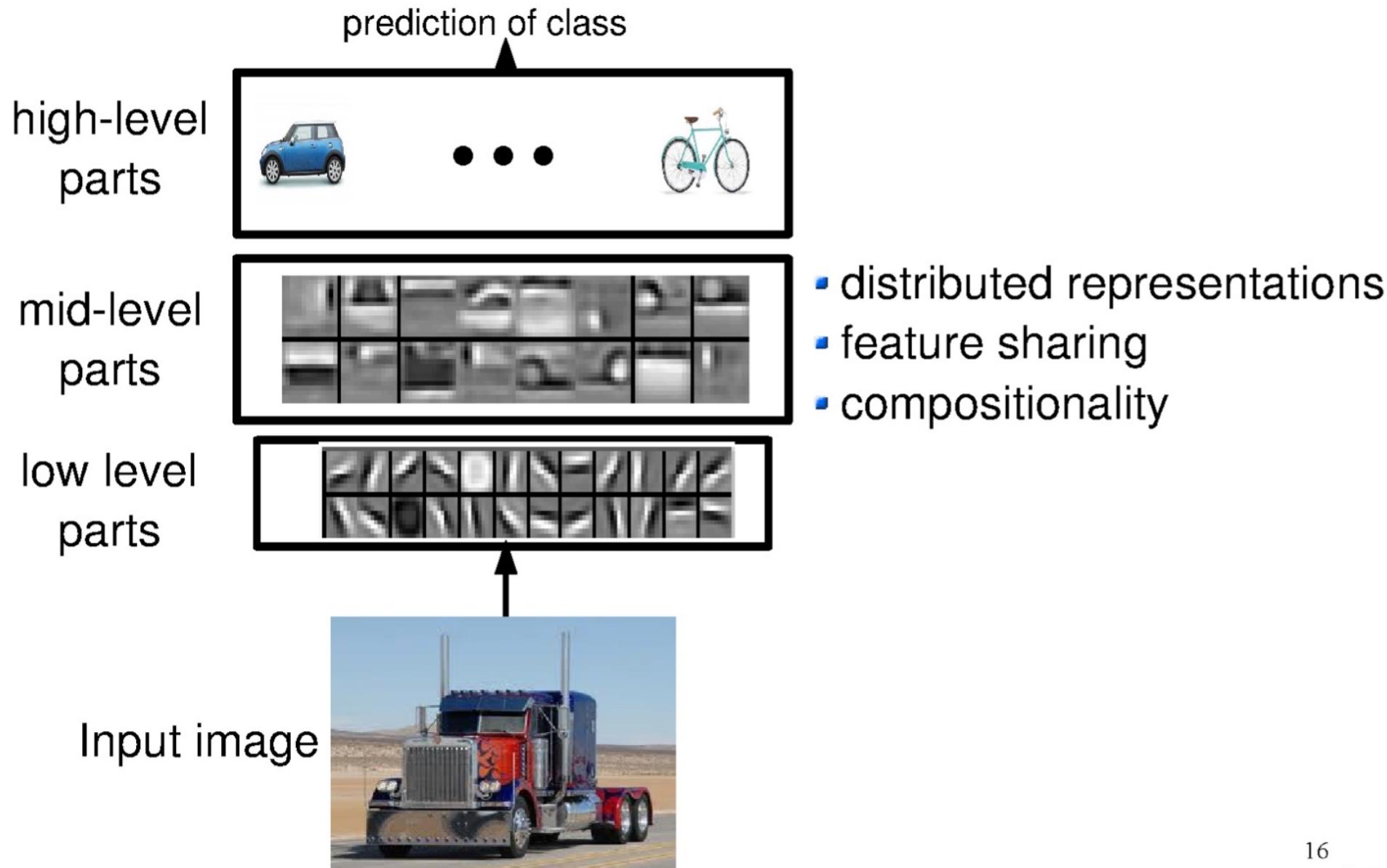


Learn **multiple filters**.

**Filter** = 'local' perceptron.  
Also called *kernel*.

E.g.: 200x200 image  
100 Filters  
Filter size: 10x10  
10K parameters

# Interpretation

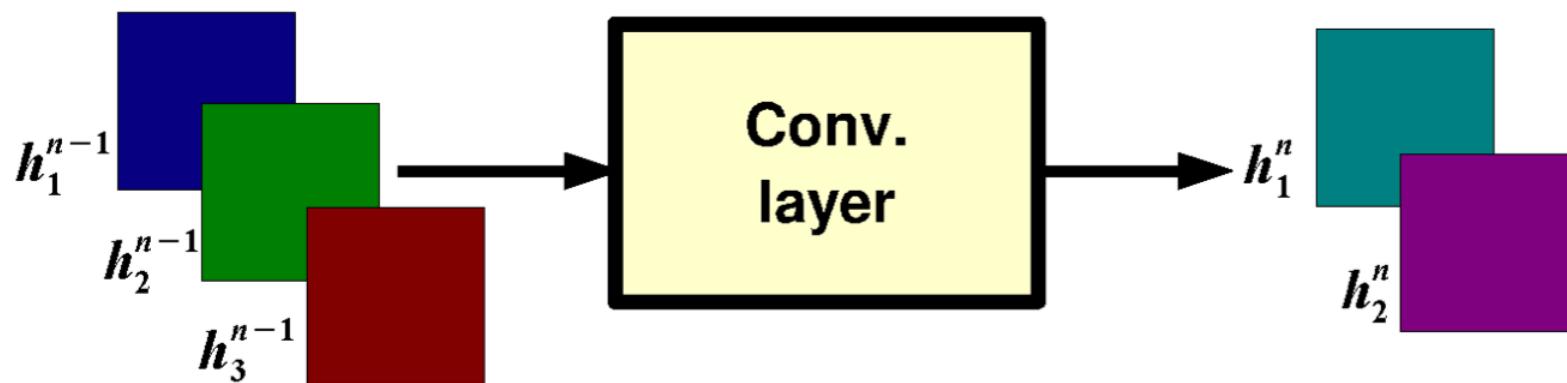


# Convolutional Layer

$$h_j^n = \max \left( 0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n \right)$$

output feature map      input feature map      kernel

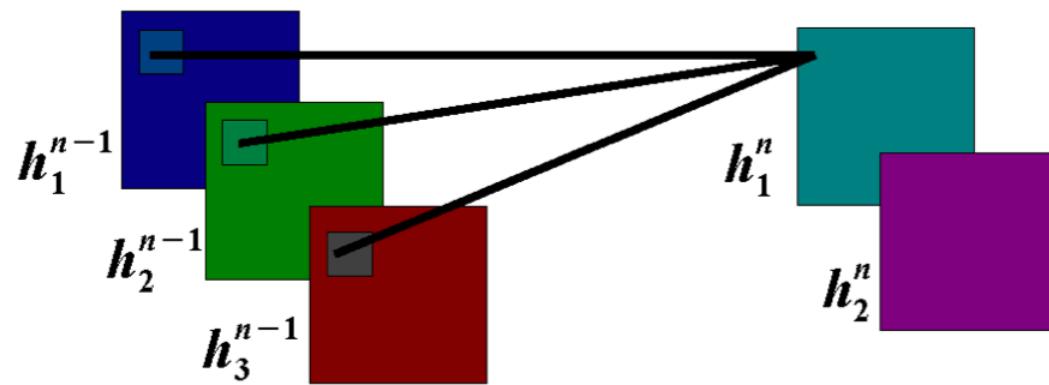
*n = layer number  
K = kernel size  
j = # channels (input)  
or # filters (depth)*



# Convolutional Layer

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n)$$

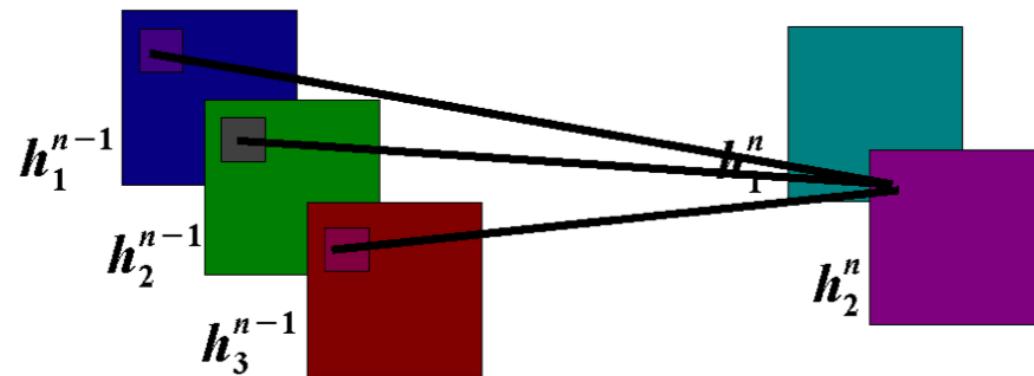
**output feature map**      **input feature map**      **kernel**



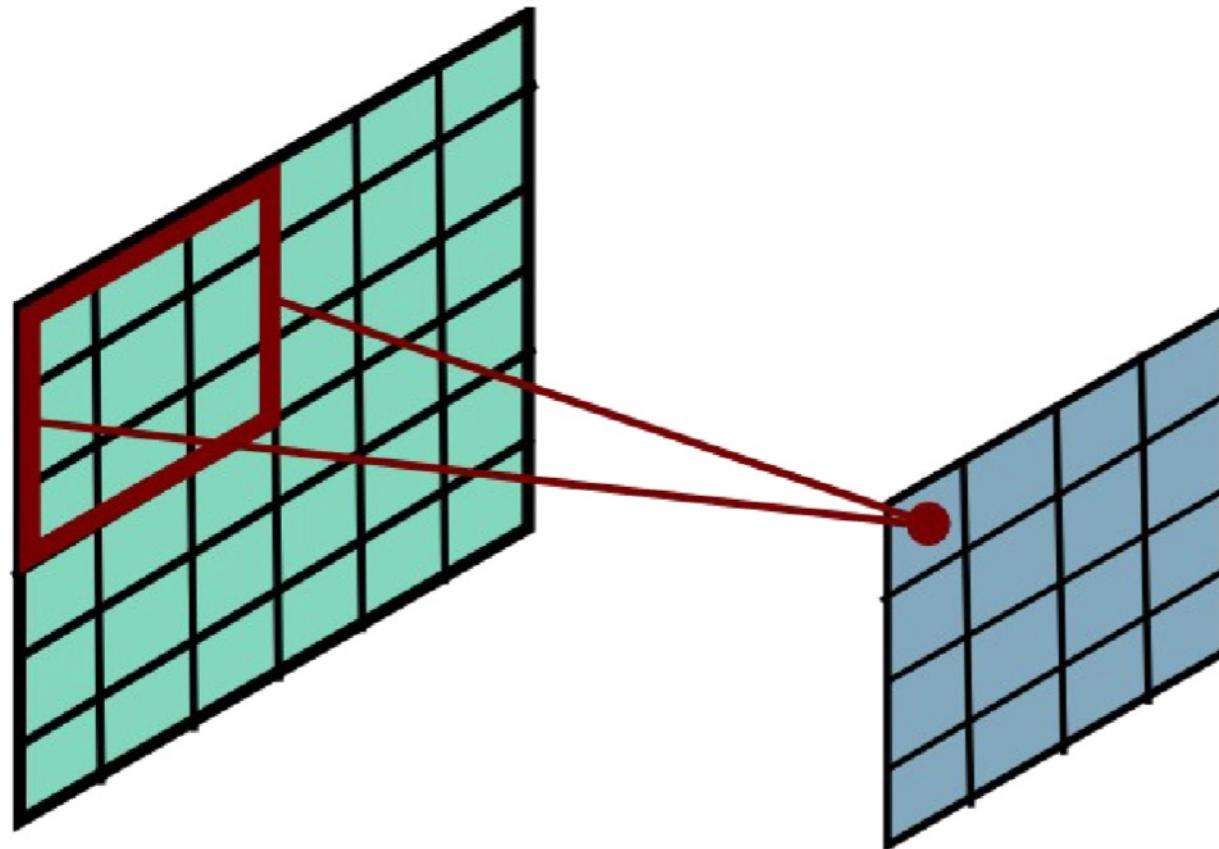
# Convolutional Layer

$$h_j^n = \max \left( 0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n \right)$$

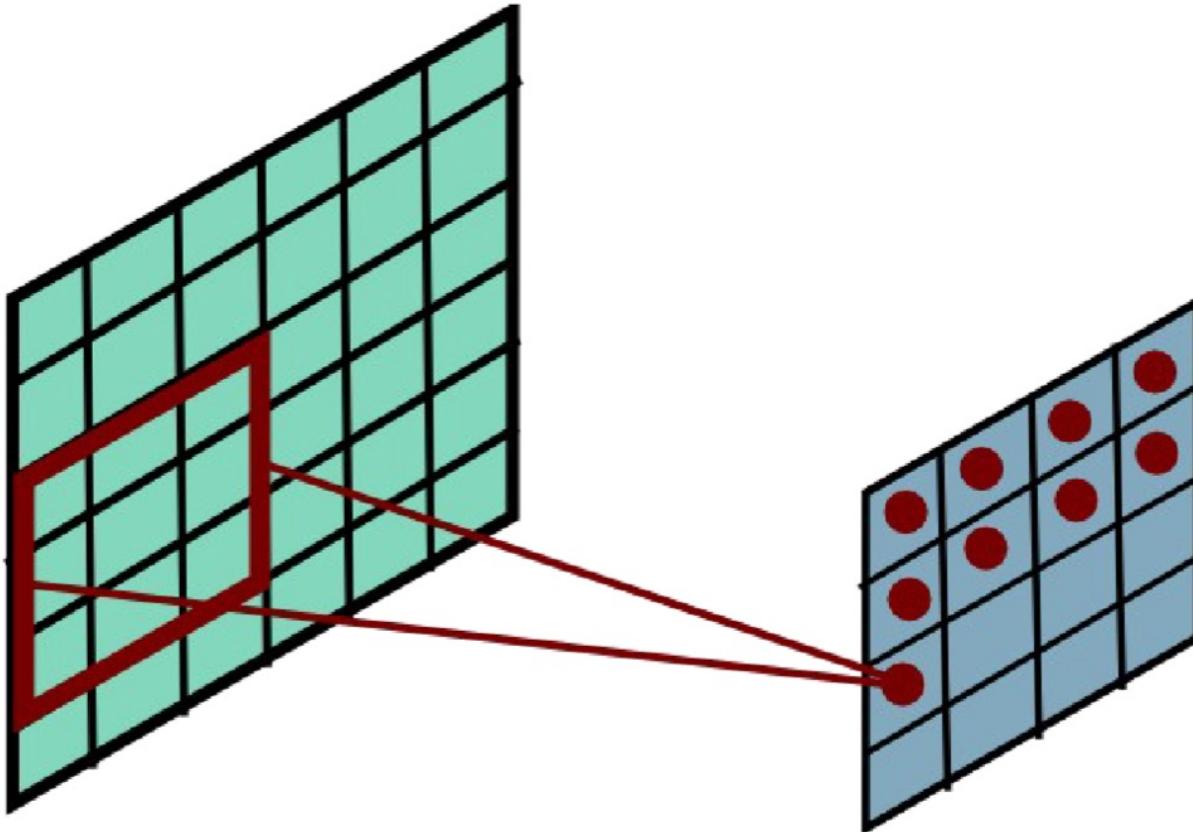
**output feature map**      **input feature map**      **kernel**



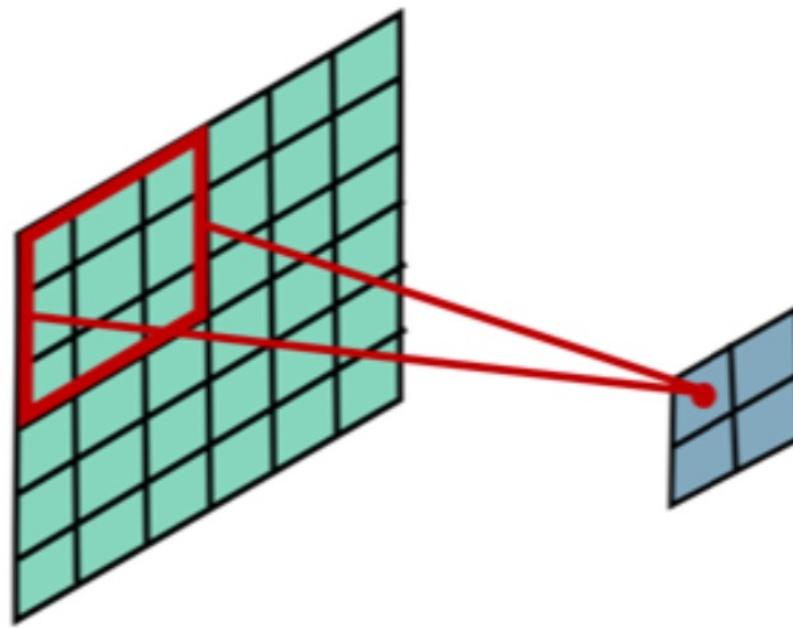
# Stride = 1



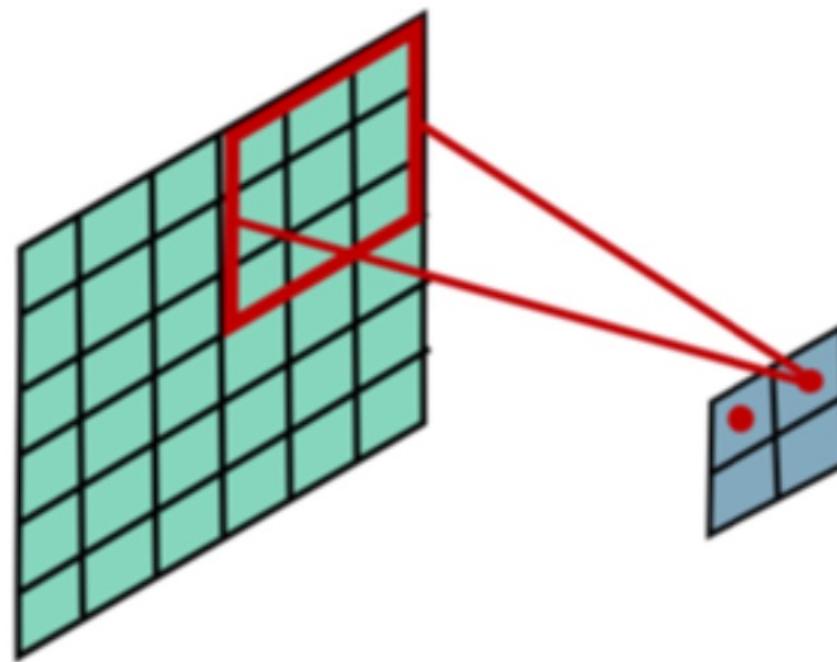
# Stride = 1



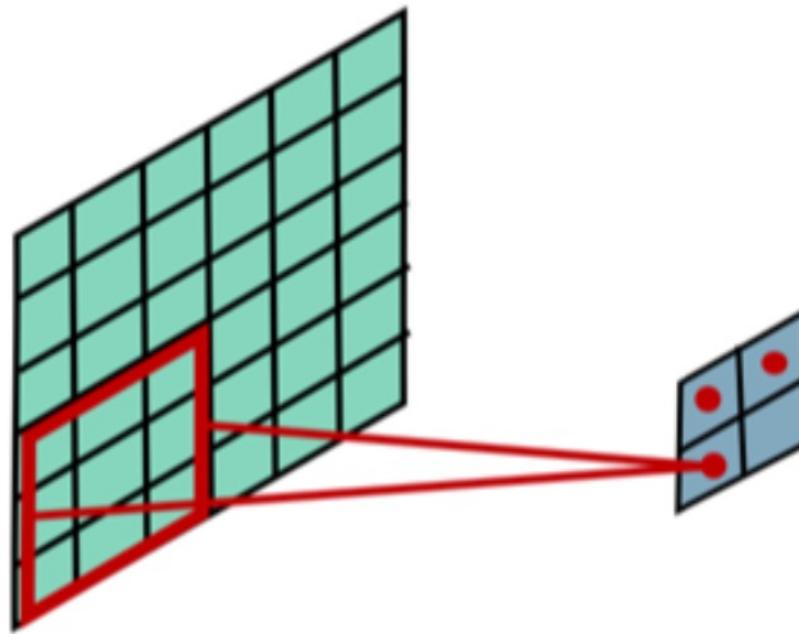
# Stride = 3



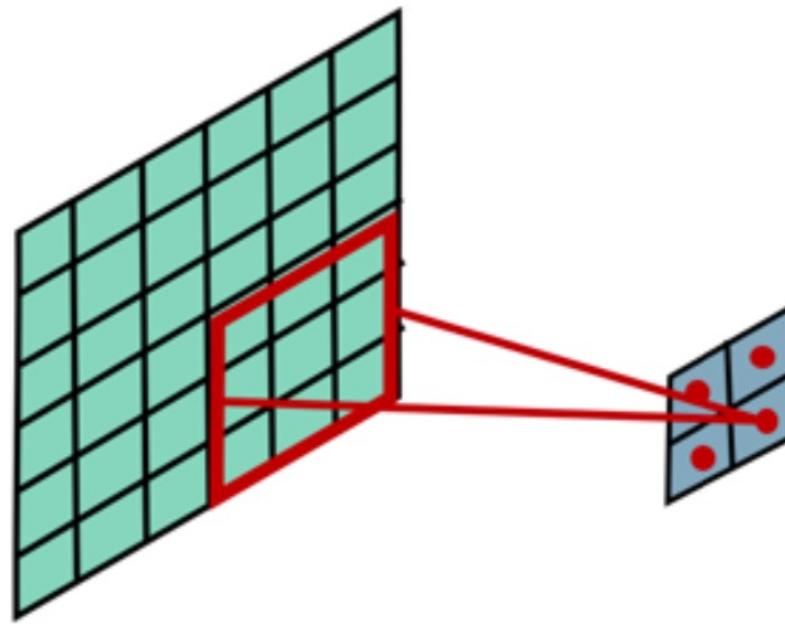
# Stride = 3



# Stride = 3



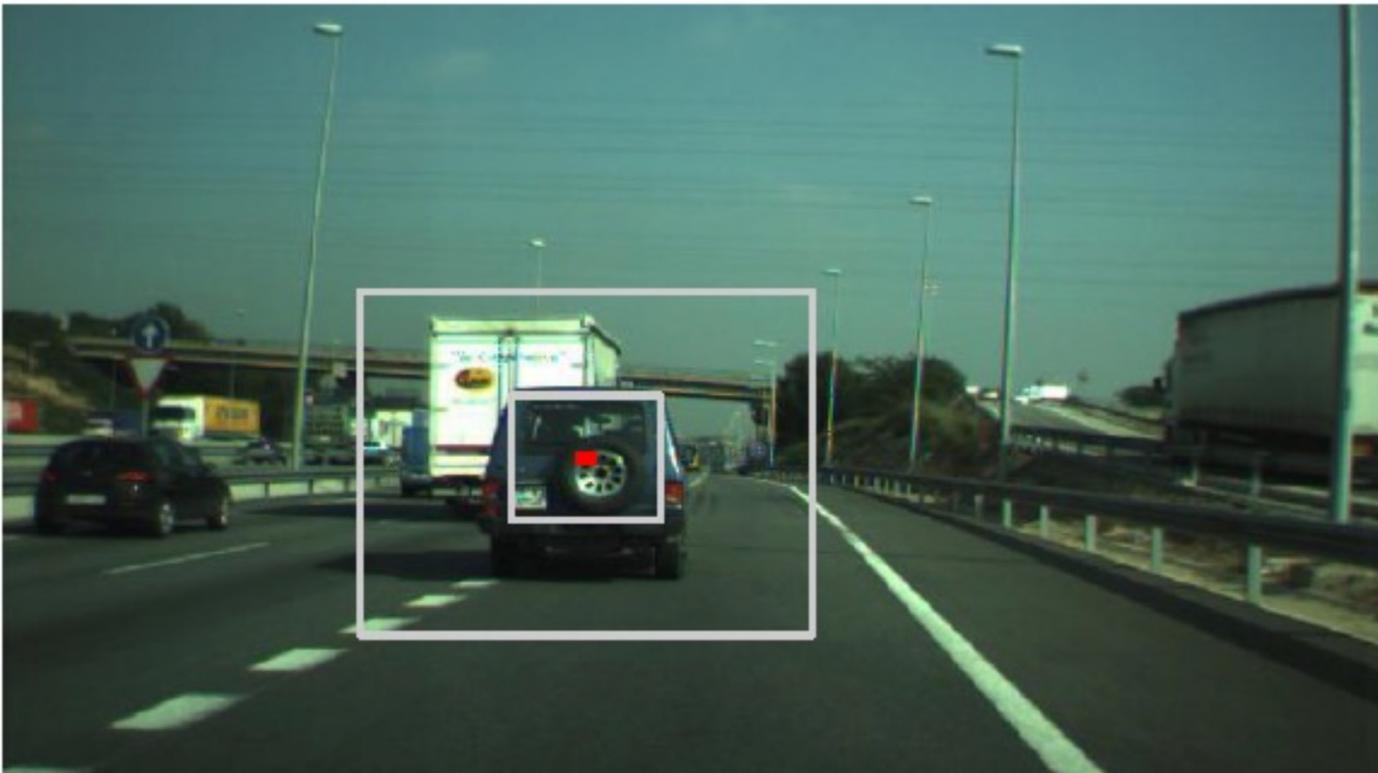
# Stride = 3



Stride is the number of pixels shifted in both dimensions for convolution if Stride is applied to both.

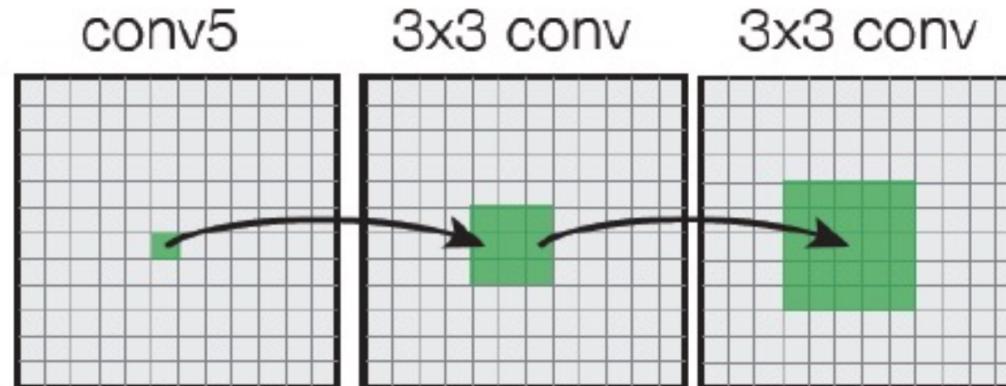
# Receptive Field

- Is the spatial extent of the connectivity (equivalently this is the filter size).

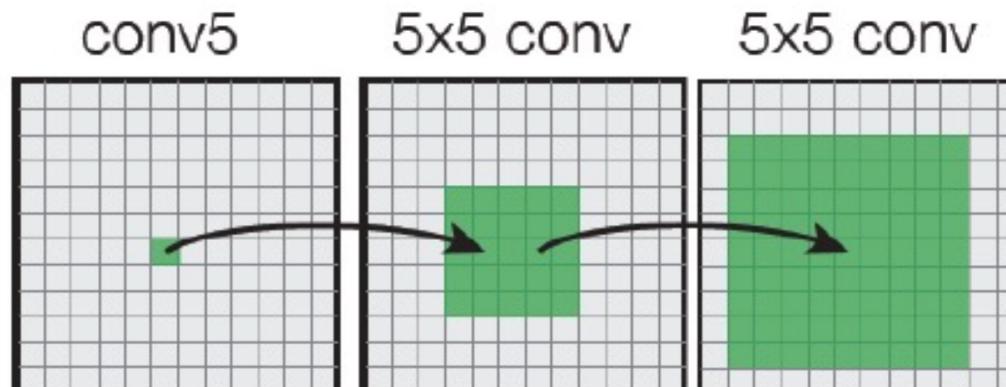


# Receptive Field

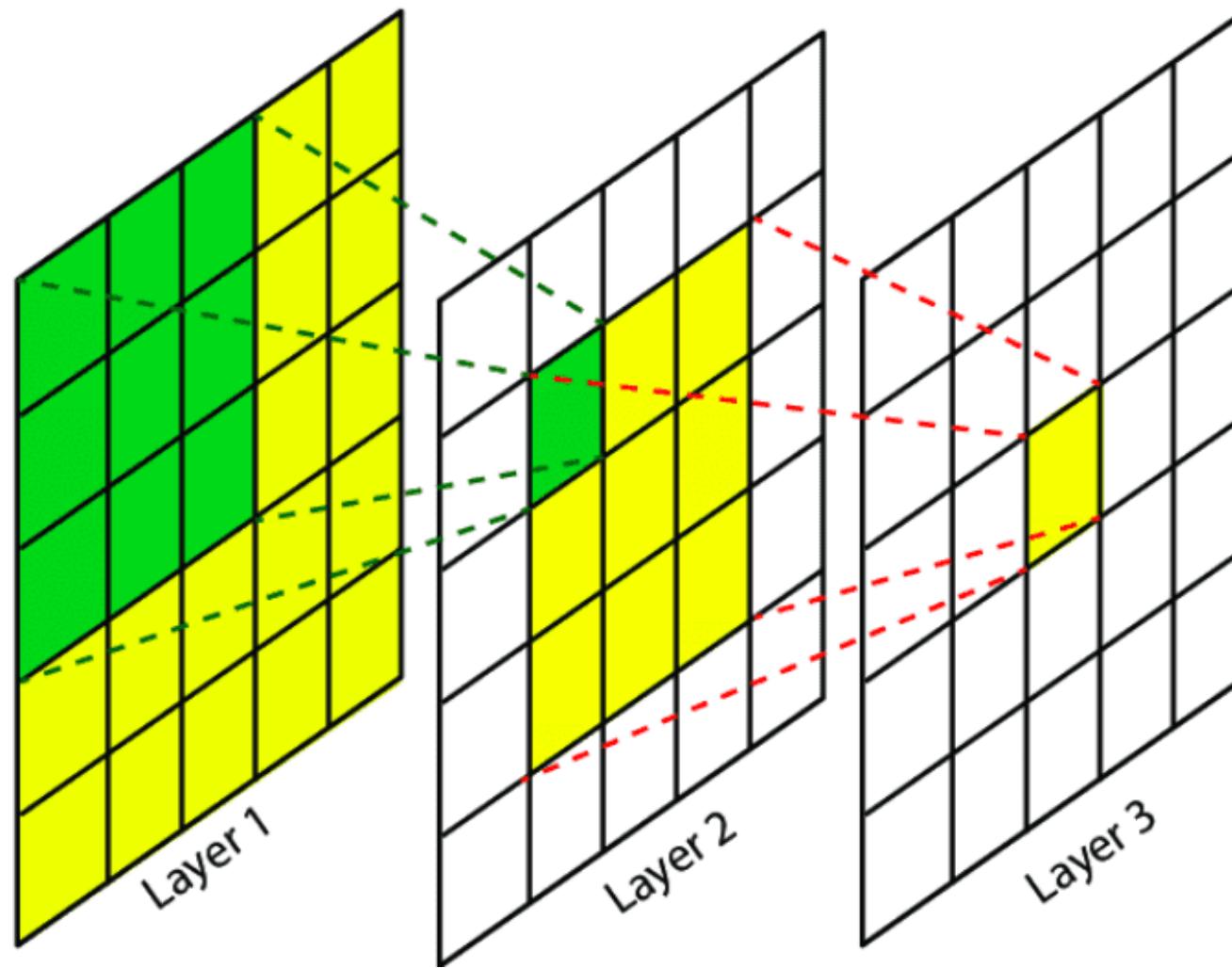
- Receptive Field



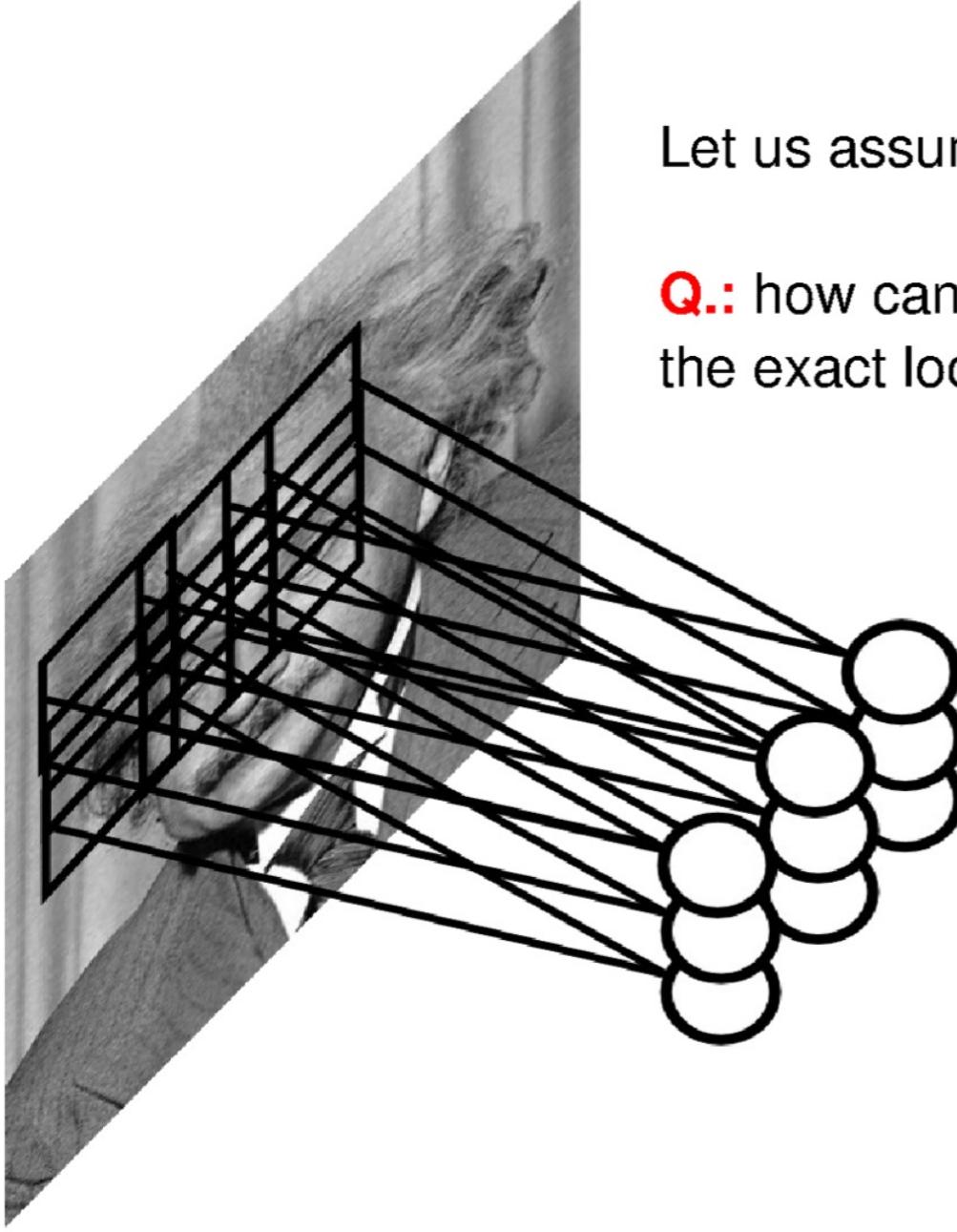
(a) two stacked 3x3 convolution layers



(b) two stacked 5x5 convolution layers



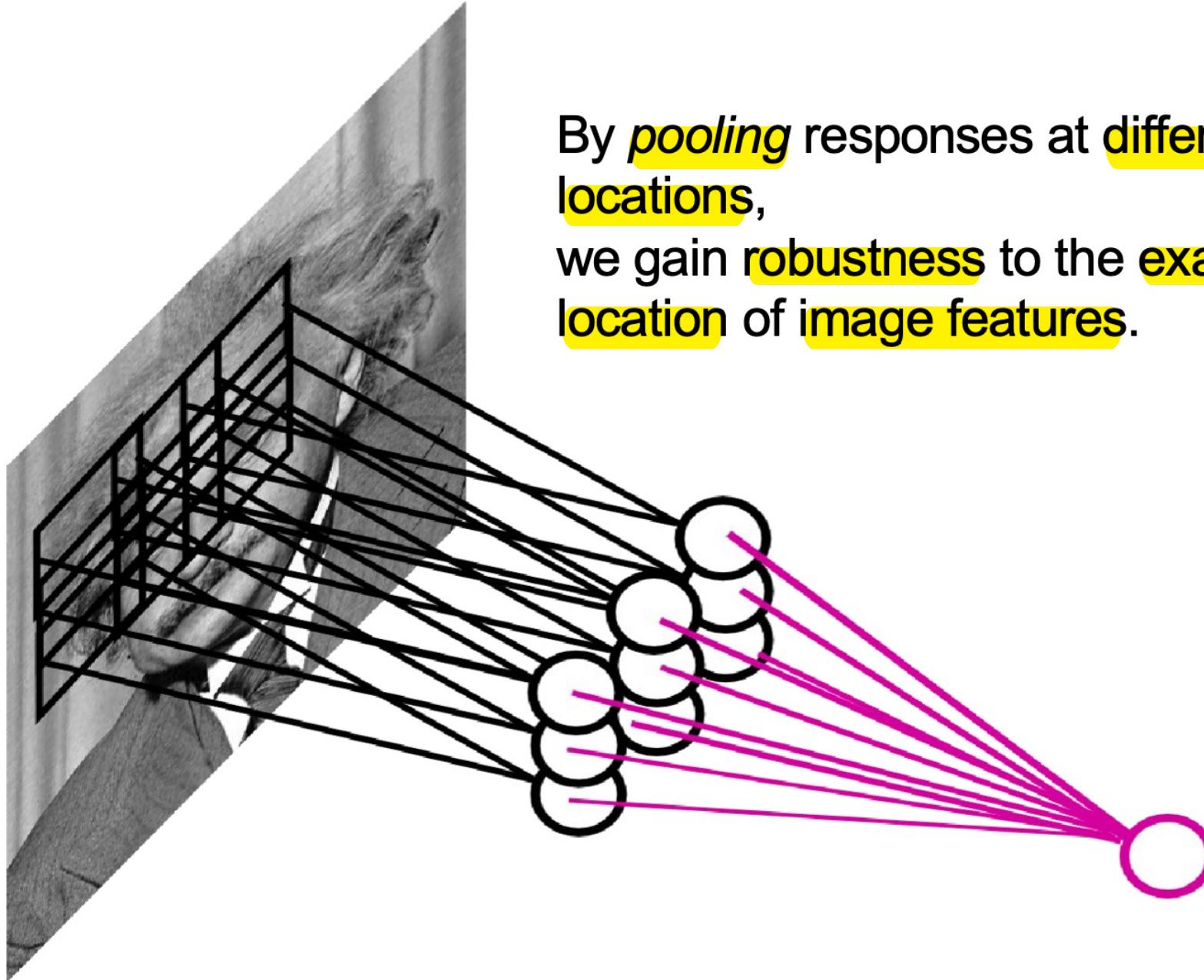
# Pooling Layer



Let us assume filter is an “eye” detector.

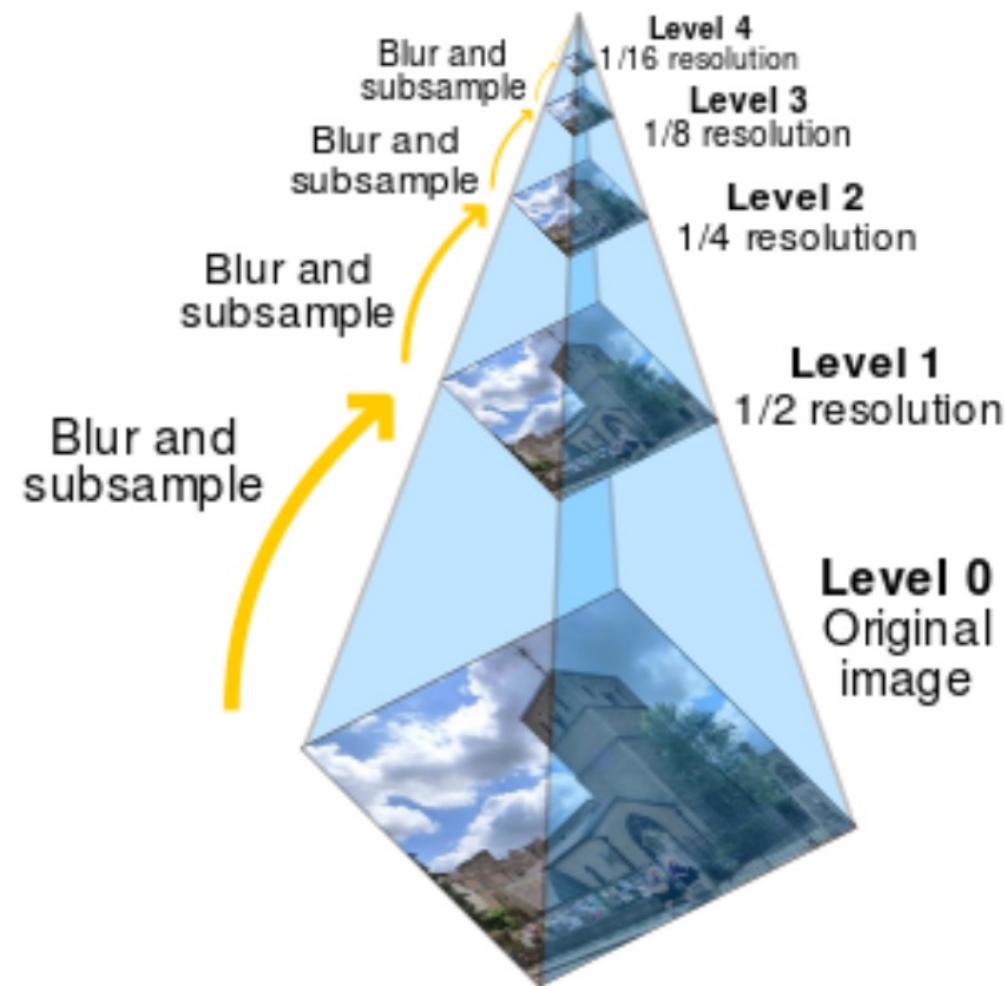
**Q.:** how can we make the detection robust to the exact location of the eye?

# Pooling Layer

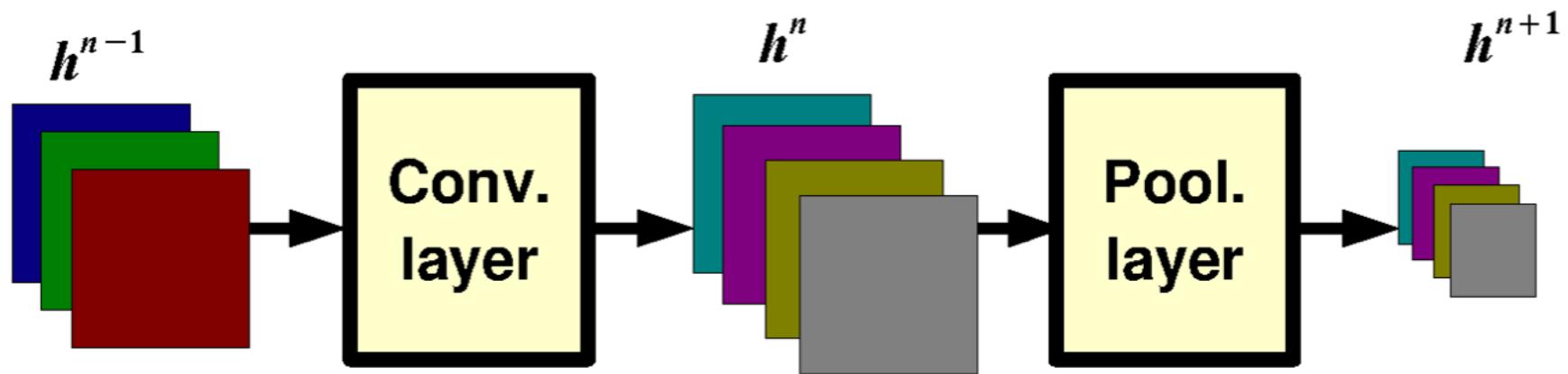


By *pooling* responses at **different locations**, we gain **robustness** to the **exact spatial location** of image features.

# Pooling is similar to pyramid downsampling



# Pooling Layer: Receptive Field Size



# Pooling Layer: Examples

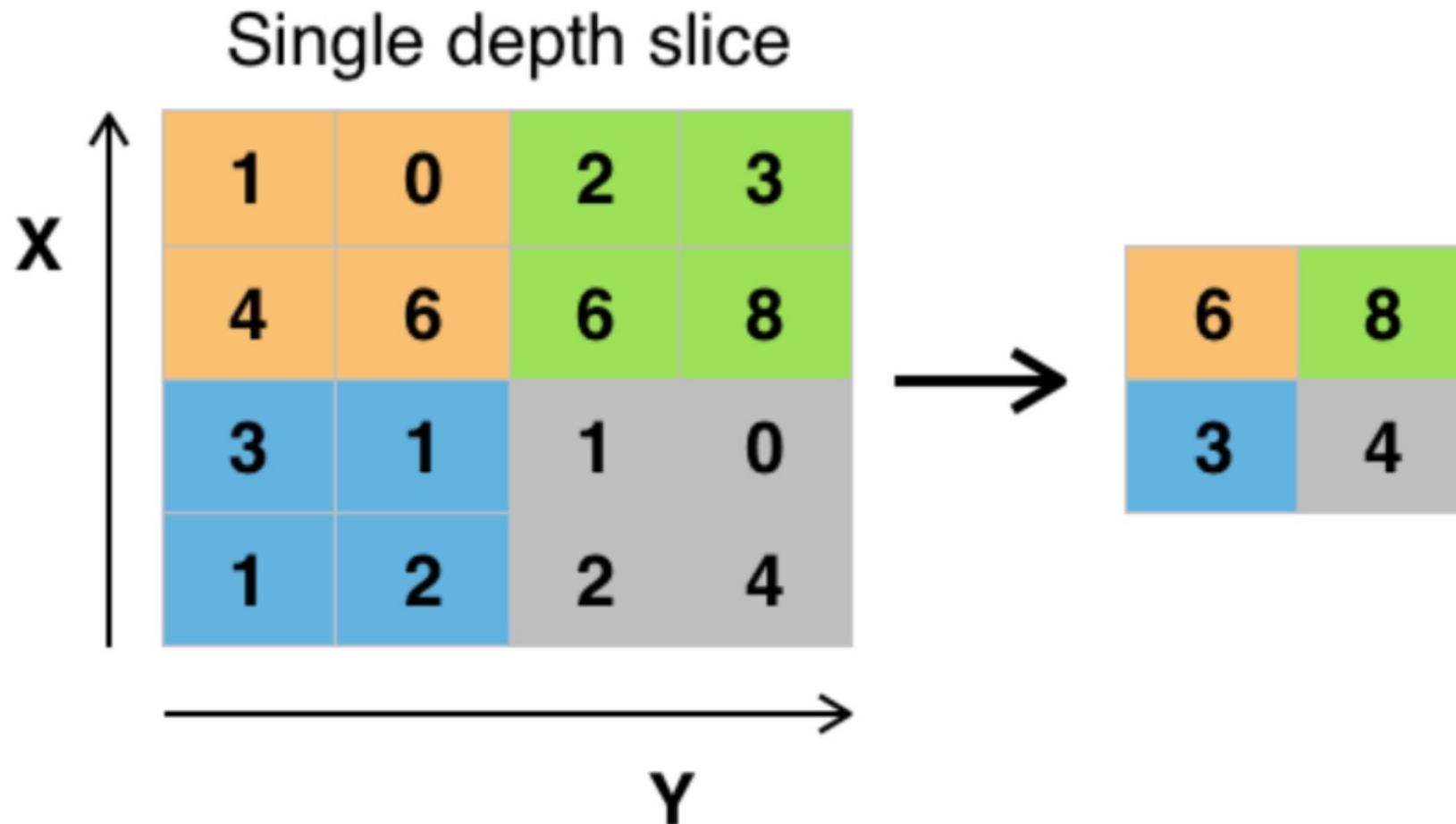
Max-pooling:

$$h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

# Max pooling



# Pooling Layer: Examples

Max-pooling:

$$h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

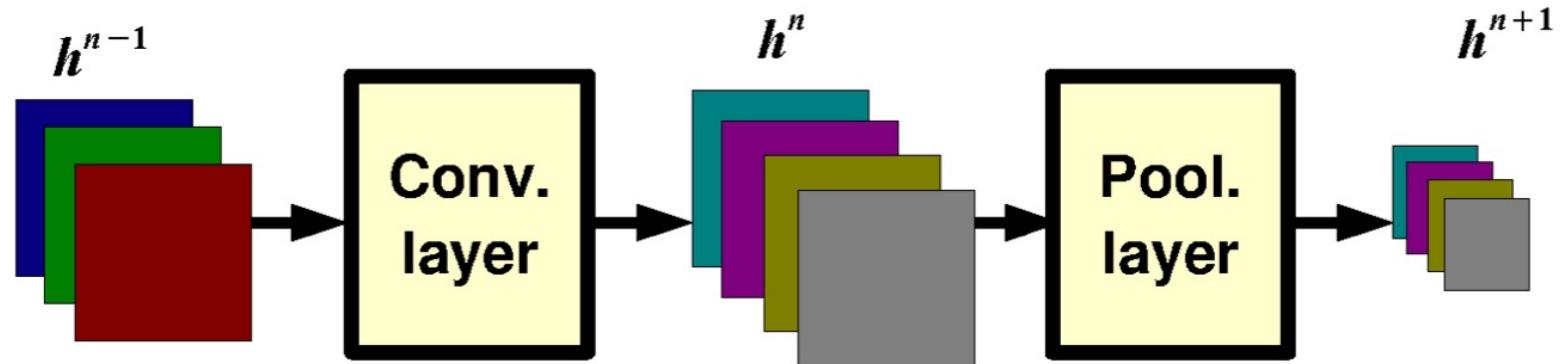
L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

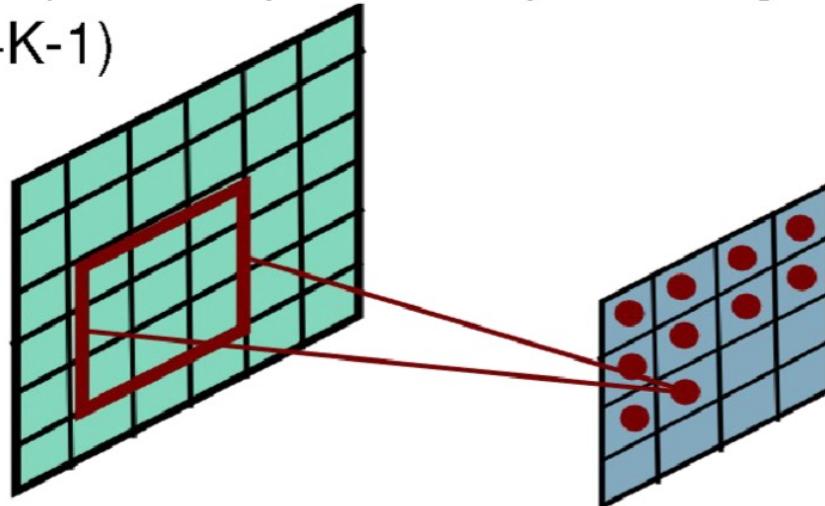
L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

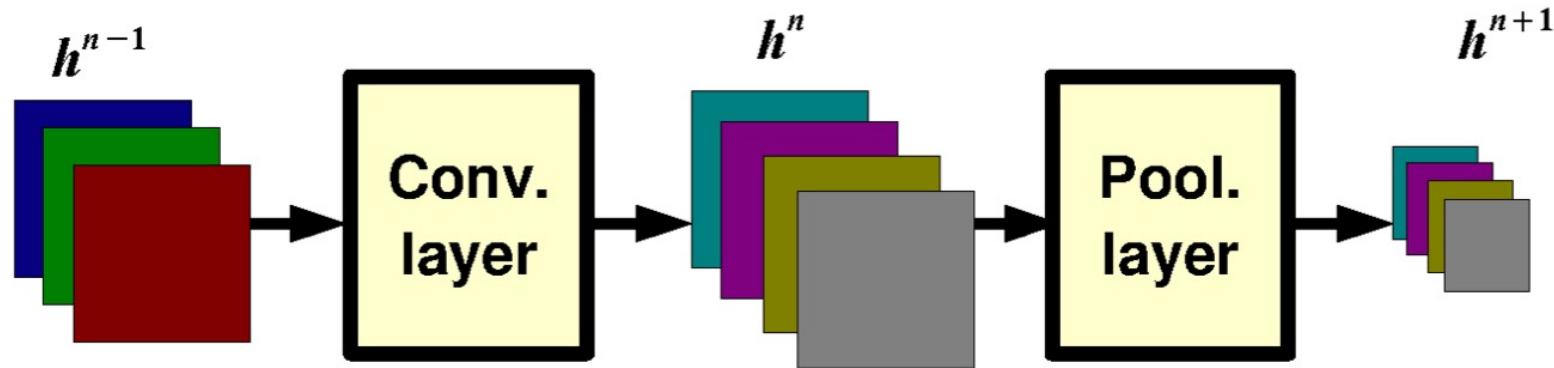
# Pooling Layer: Receptive Field Size



If convolutional filters have size  $K \times K$  and stride 1, and pooling layer has pools of size  $P \times P$ , then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:  
 $(P+K-1) \times (P+K-1)$



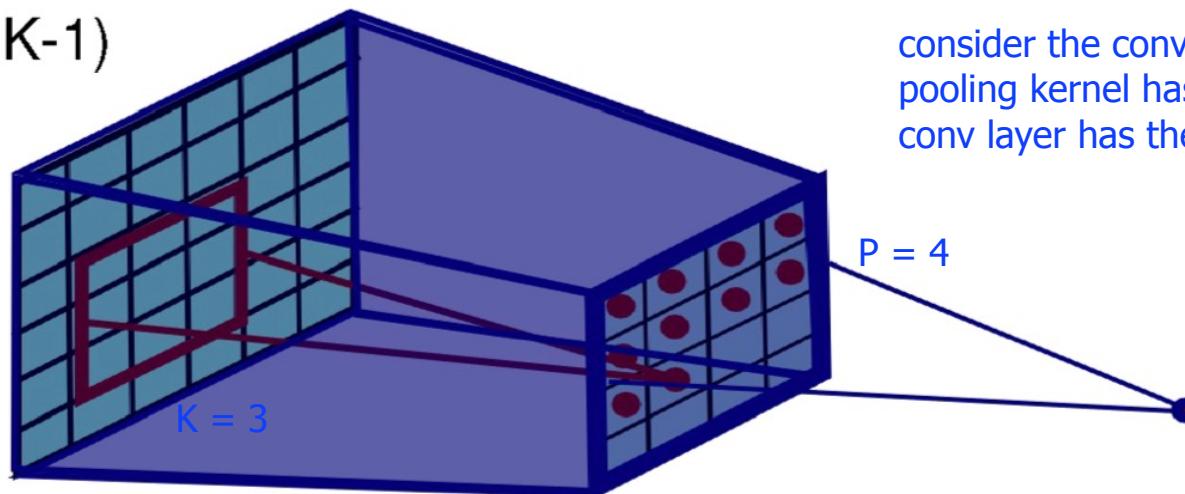
# Pooling Layer: Receptive Field Size



If convolutional filters have size  $K \times K$  and stride 1, and pooling layer has pools of size  $P \times P$ , then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:

$$(P+K-1) \times (P+K-1)$$

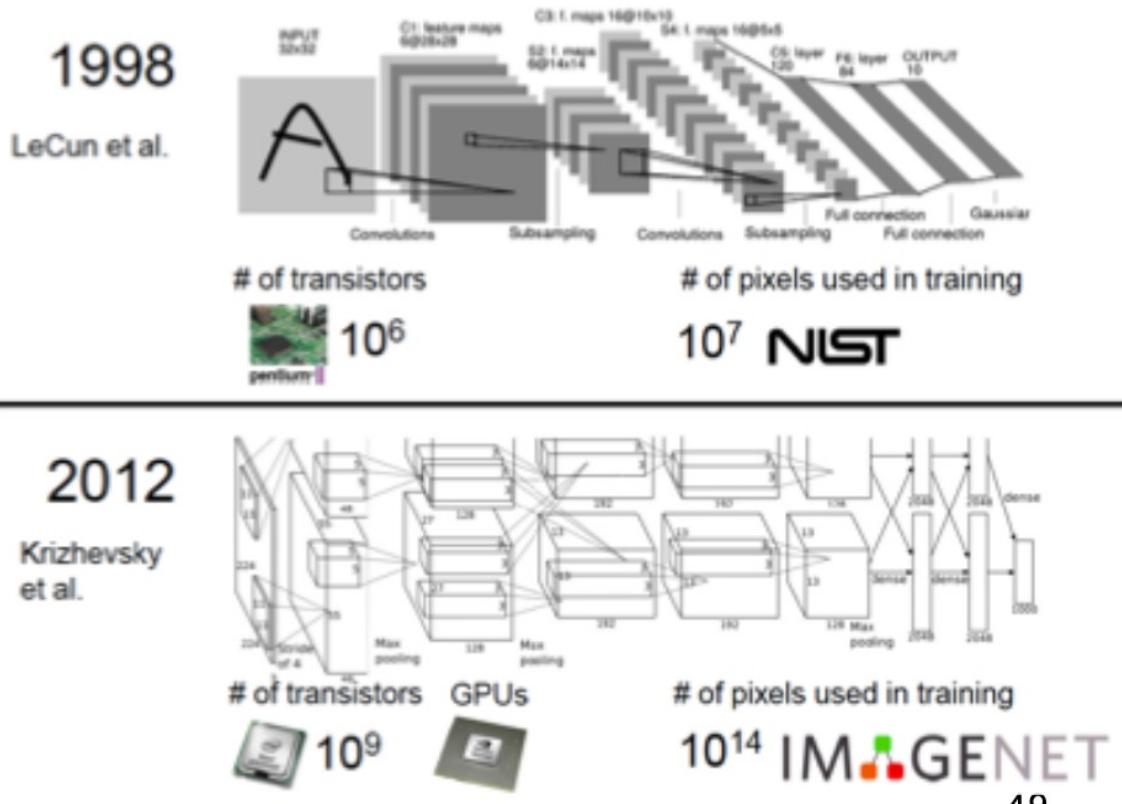
input size:  
 $(K * P + 1) * (K * P + 1)$



consider the conv kernel has the size of  $(K * K)$ , the pooling kernel has the size of  $(P * P)$ , then the input of conv layer has the size of  $(K * P + 1) * (K * P + 1)$ .

# Convolutional Neural Networks

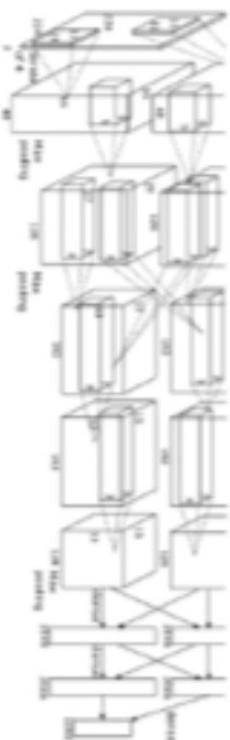
- Not invented over night
  - Back-propagation: ~1975
  - Early convolutional neural networks: ~1988 (Yann LeCun)



# Other variants of CNN

Year 2012

SuperVision

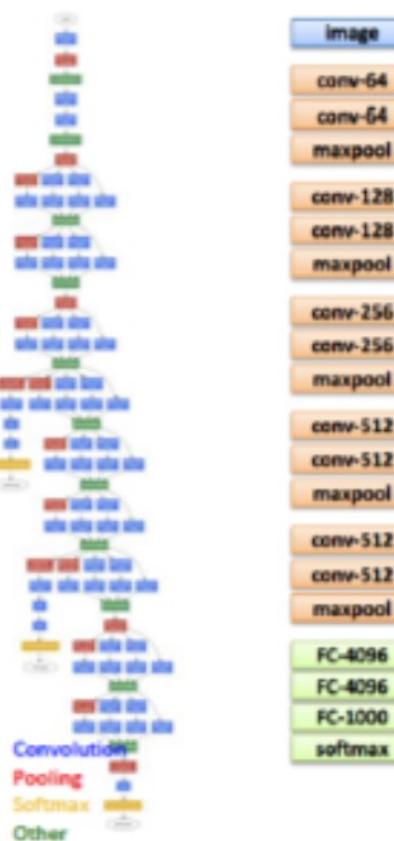


[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet

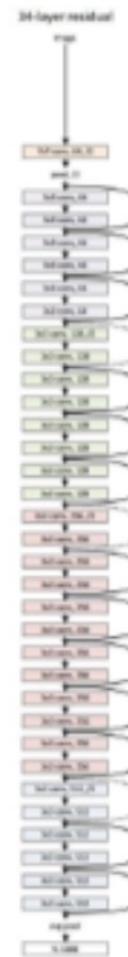
VGG



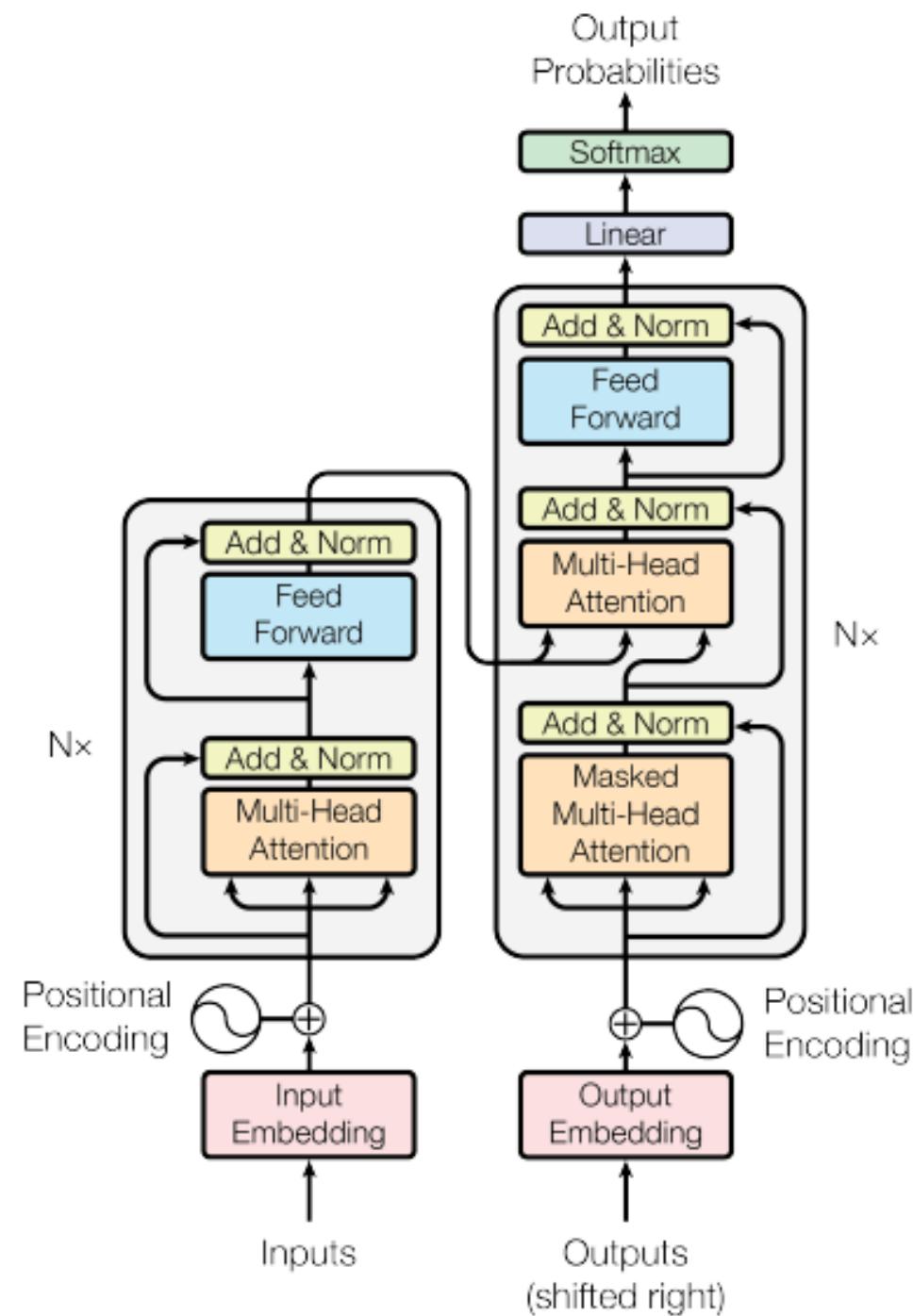
[Szegedy arxiv 2014]

Year 2015

MSRA

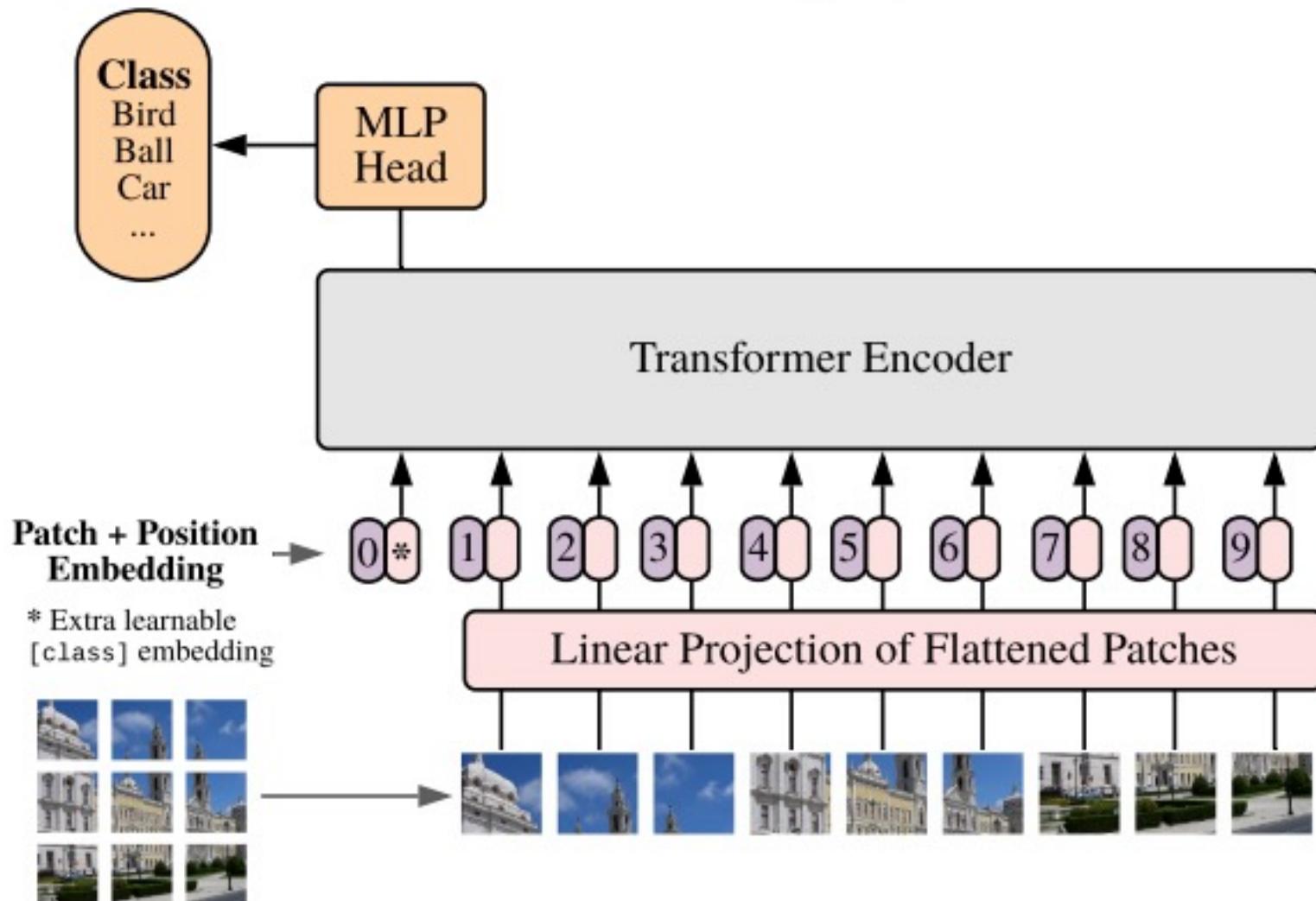


[Simonyan arxiv 2014]



2017: Transformer, Attention is all you need

## Vision Transformer (ViT)



2020: ViT, transformer for computer vision tasks.

# Reference

- Chapter 6 Deep feedforward networks, Deep Learning,  
<https://www.deeplearningbook.org/>