CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Computing Gradients
  - Backprop in FC+ReLU NNs
- Convolutional Neural Networks

Dhruv Batra

Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 10/02, 11:55pm

- HW0 grades
  - Out week of 10/01

Project

- Goal
  - Chance to try Deep Learning
  - **Combine with other classes / research / credits / anything**
    - You have our blanket permission
    - Extra credit for shooting for a publication
    - Get permission from other instructors; delineate different parts
  - Encouraged to apply to your research (computer vision, NLP, robotics,…)
  - Must be done this semester.
  - Groups of 3-4

- Main categories
  - Application/Survey
    - Compare a bunch of existing algorithms on a new application domain of your interest
  - Formulation/Development
    - Formulate a new model or algorithm for a new or old problem
  - Theory
    - Theoretically analyze an existing algorithm

Computing

- Major bottleneck
  - GPUs

- Options
  - Your own / group / advisor's resources
  - Google Cloud Credits
    - $50 credits to every registered student courtesy Google
  - https://colab.research.google.com
  - Minsky cluster in IC

Administrativia

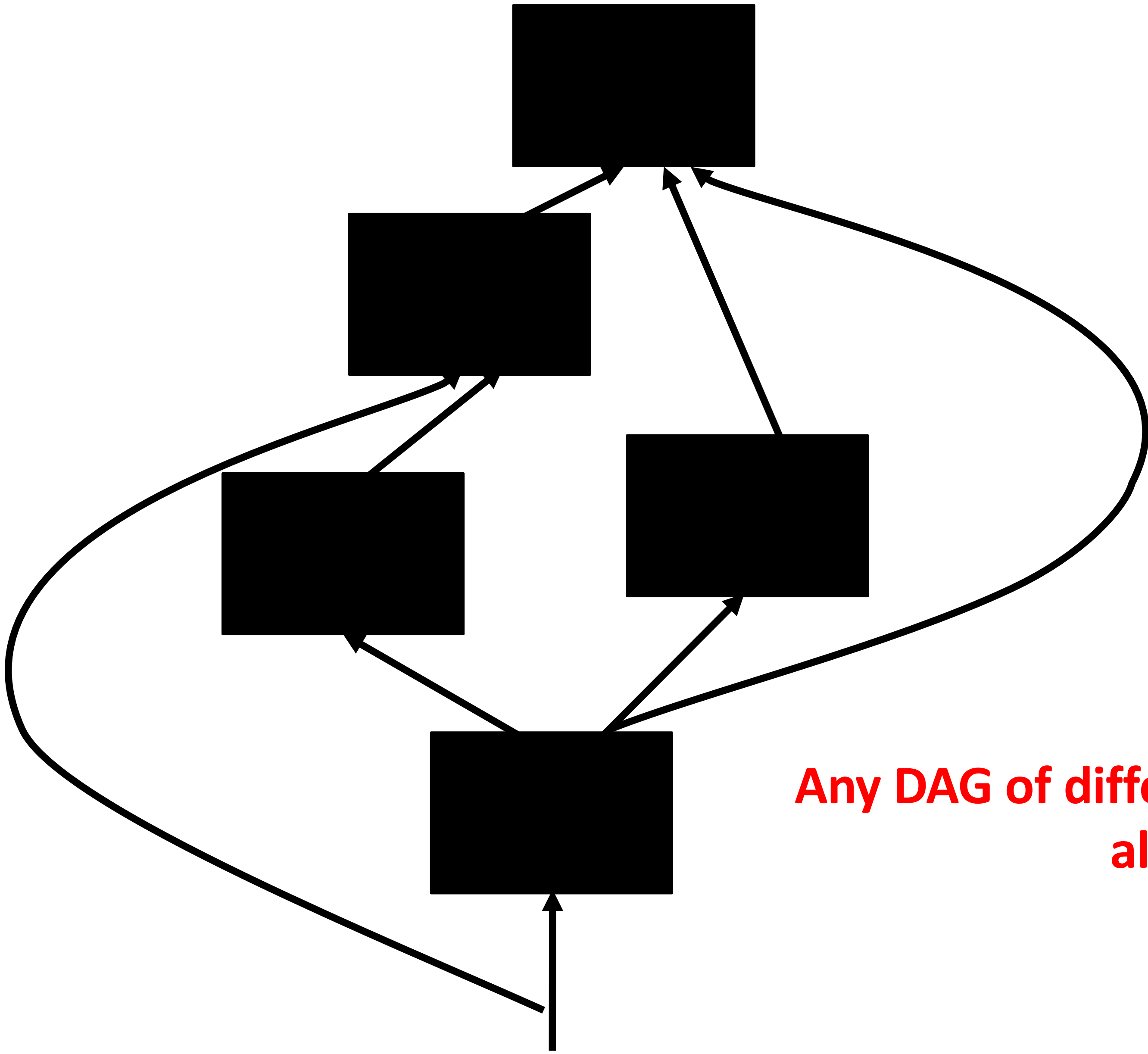
- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1BipWLvvWb7Fu6OSDd-uOCF1Lr_4drKOCRVdhxm_eSHc/edit#gid=0
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names

Recap from last time

How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

Computational Graph



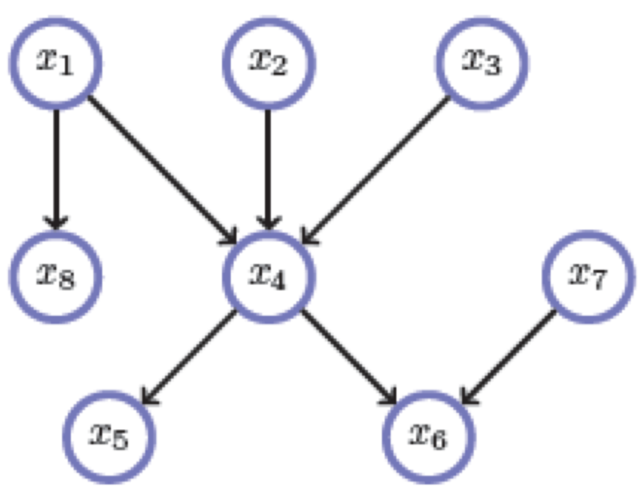
Any DAG of differentiable modules is allowed!

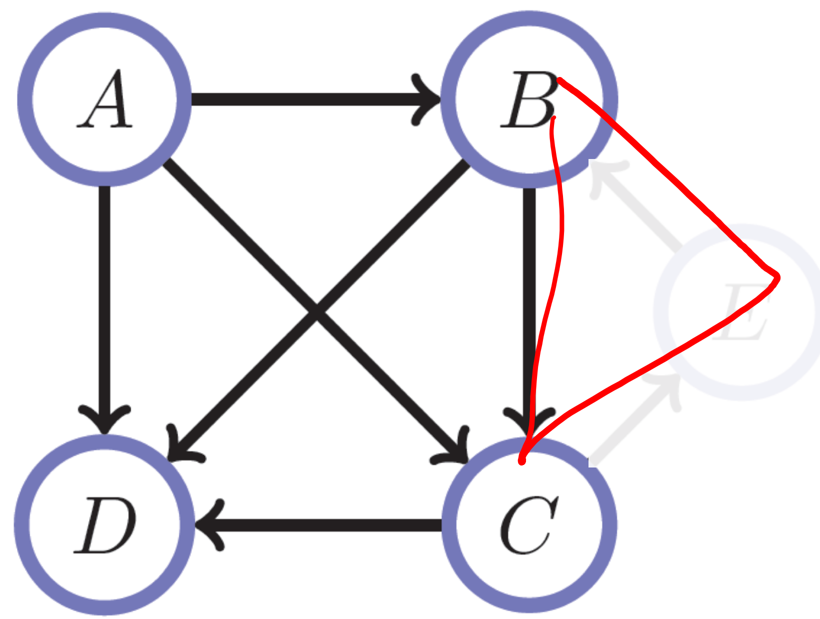
Directed Acyclic Graphs (DAGs)

- Exactly what the name suggests
  - Directed edges?
  - No (directed) cycles
  - Underlying undirected cycles okay

$G = (V, E)$

$E = \{(v_i, v_j) \mid v_i, v_j \in V\}$

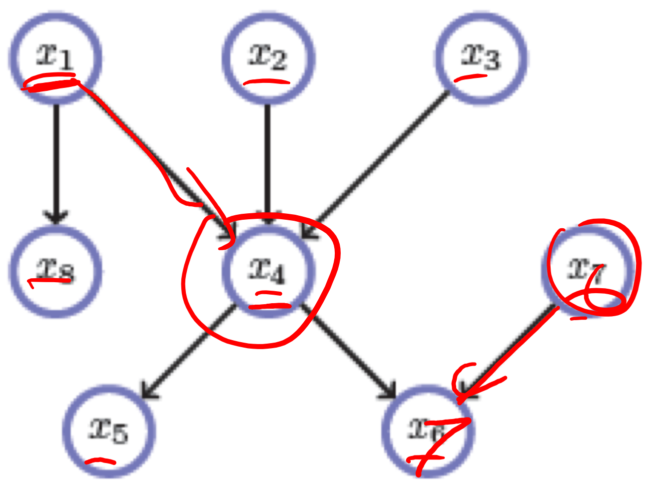


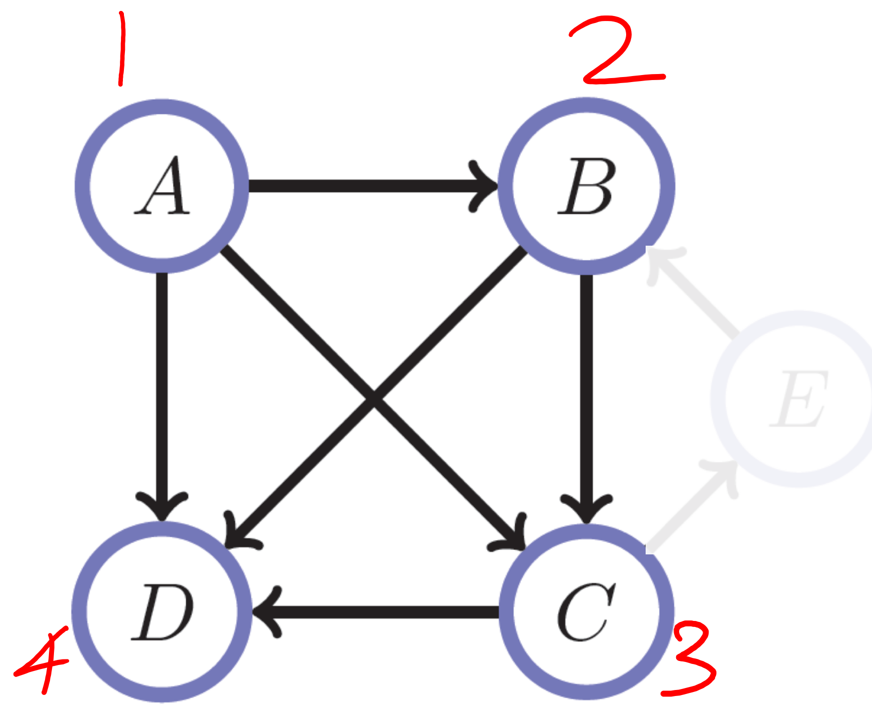


Directed Acyclic Graphs (DAGs)

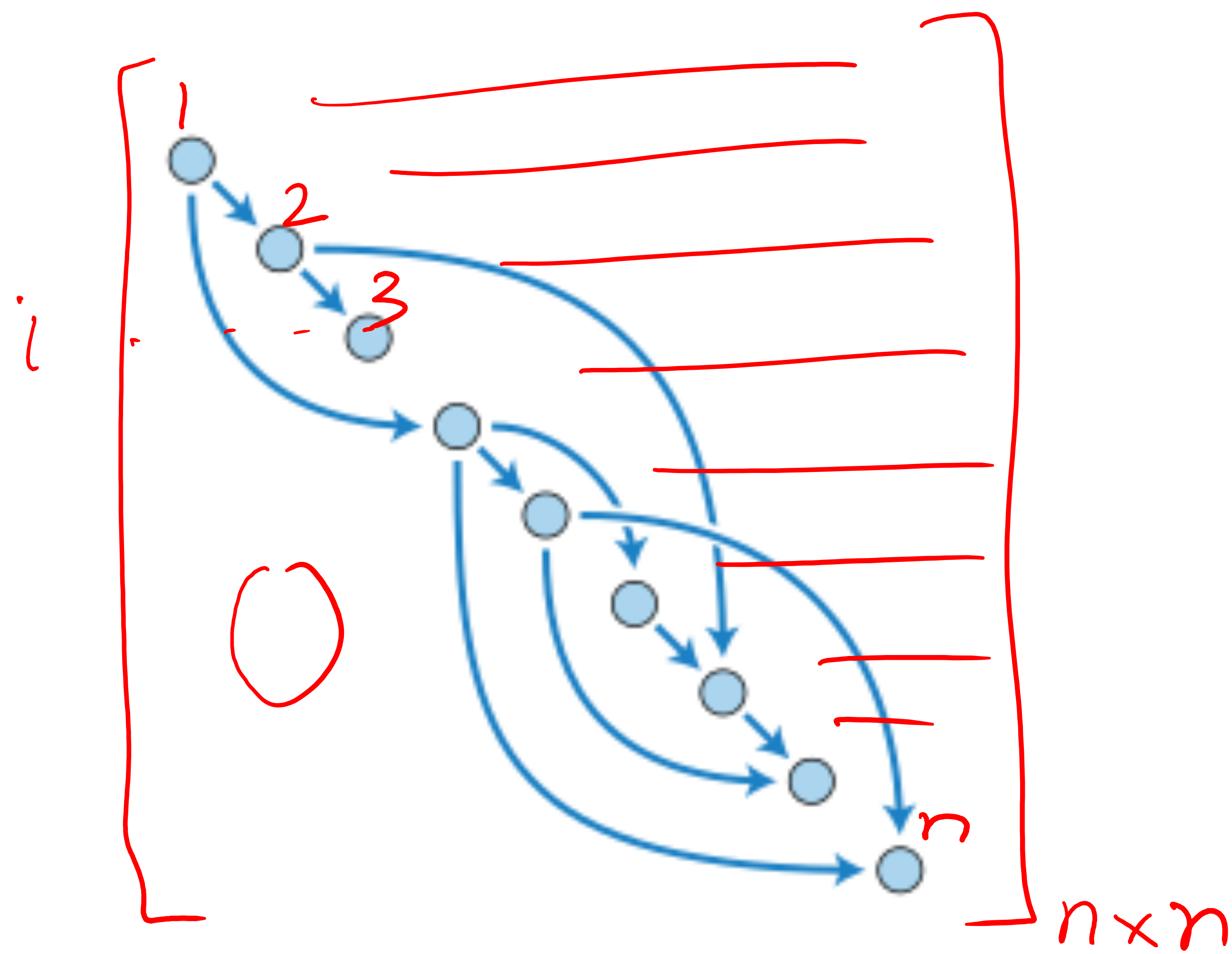
- Concept
  - Topological Ordering

$\exists\, \sigma : V \to \underbrace{[n]} = \{1, ..., n\}$ s.t $\forall (v_i, v_j) \in E \quad \sigma(v_i) < \sigma(v_j)$





Directed Acyclic Graphs (DAGs)

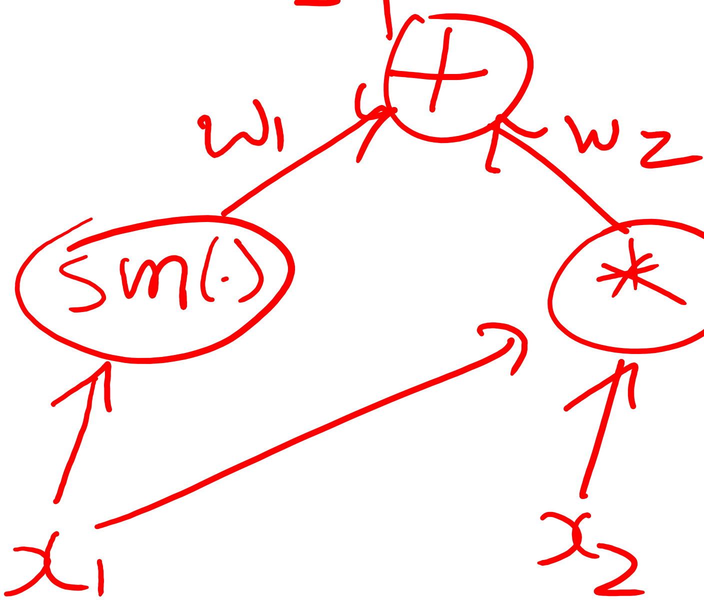


Computational Graphs

- Notation

$$f(x_1, x_2) = \underbrace{x_1 x_2}_{w_2} + \underbrace{\sin(x_1)}_{w_1}$$

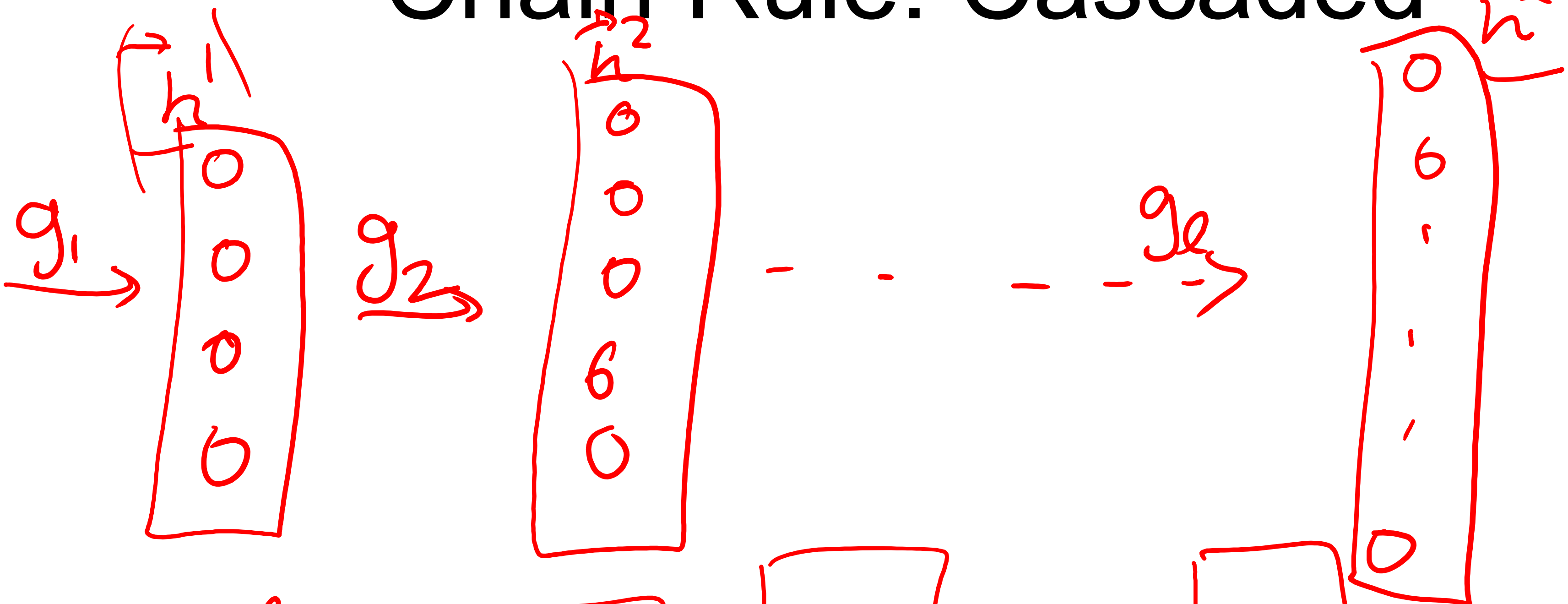
$f(x_1, x_2) = w_3$



How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

Chain Rule: Cascaded



$$\frac{\partial \vec{h}^\ell}{\partial \vec{h}^1} = J_{g_\ell} \, J_{g_{\ell-1}} \cdots J_{g_1}$$

$O(\ell d^2)$

$O(\ell d^3)$

$$= \underbrace{[\quad]_{1\times d} \, [\quad]_{d\times d}}_{O(d^2)} \cdots \underbrace{[\quad][\quad]_{d\times d}}_{O(d^3)}$$

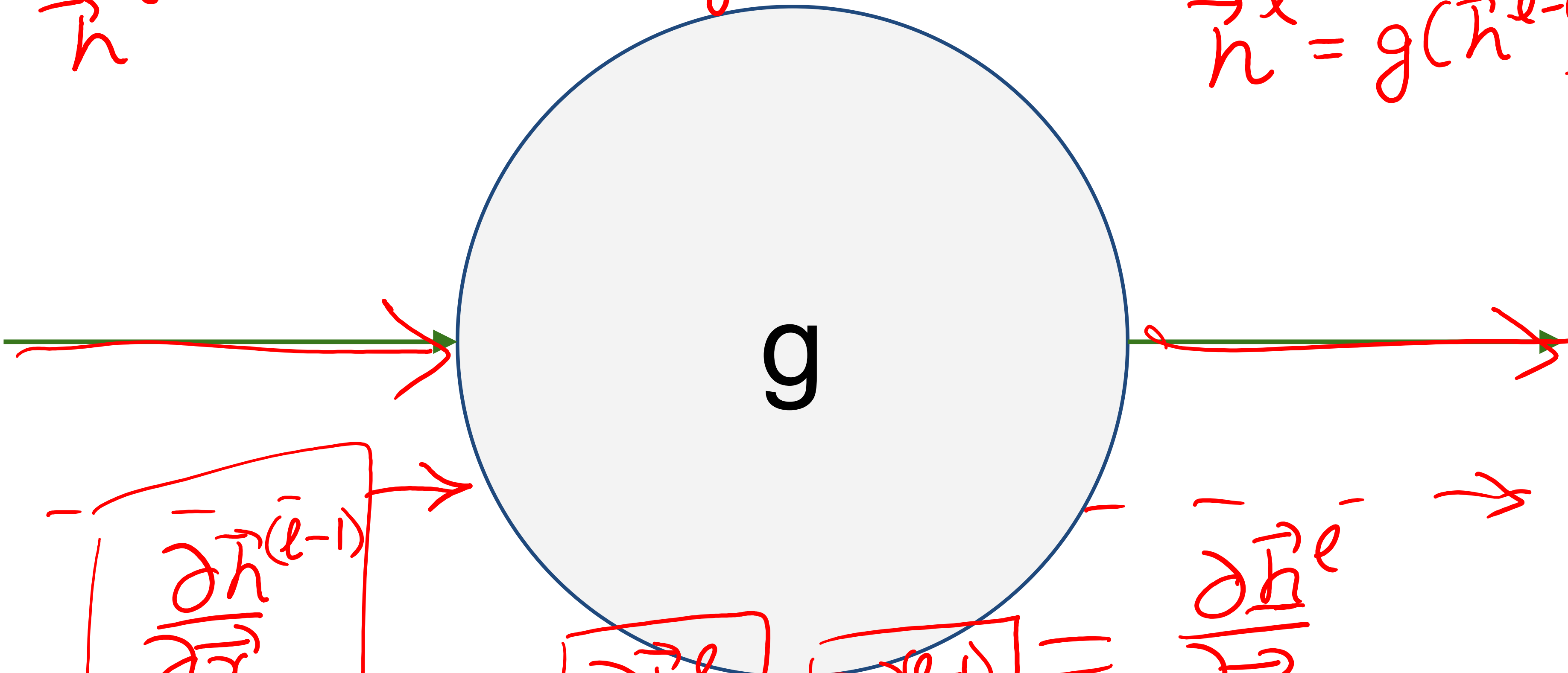
Forward mode AD

$\boxed{\frac{\partial L}{\partial \vec{x}}}$

layer l

$\vec{h}^{l-1}$

$\vec{h}^{l} = g(\vec{h}^{l-1})$

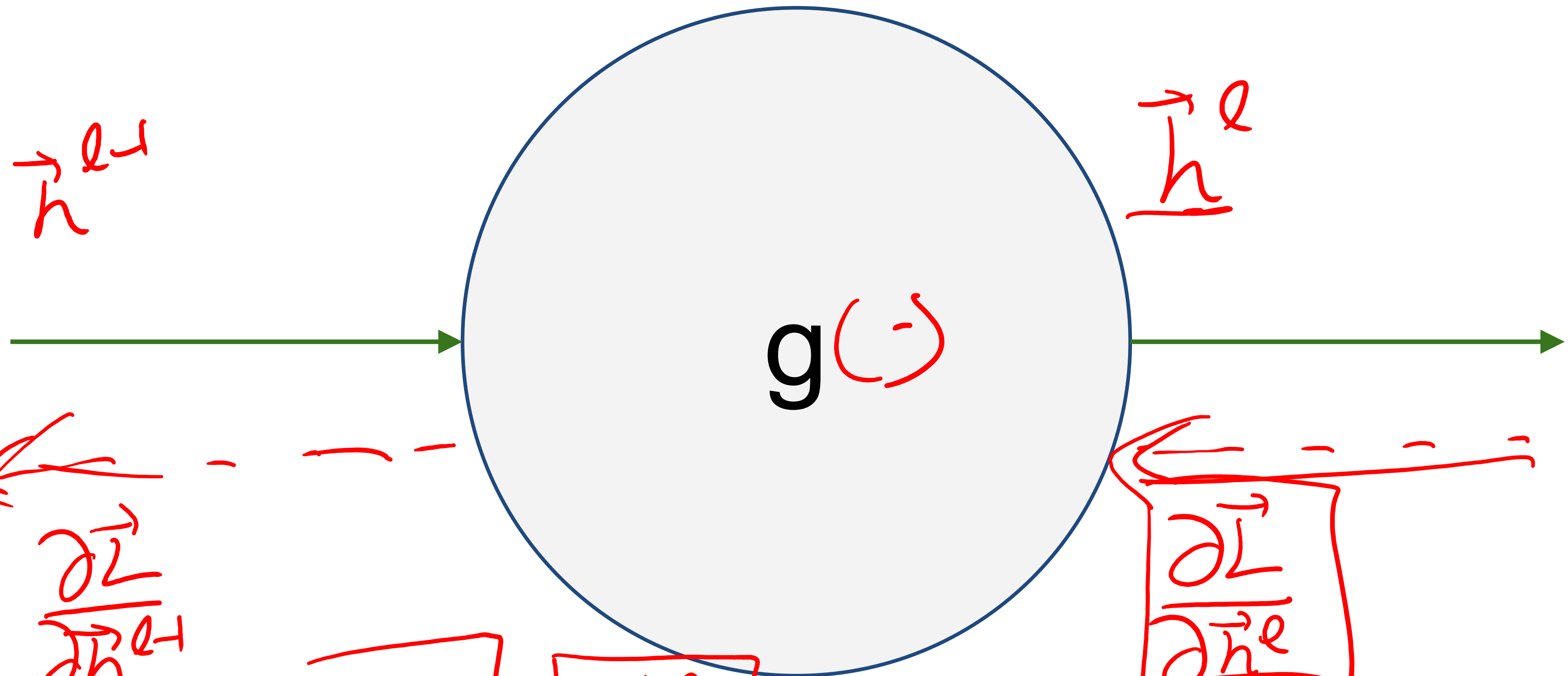


$\underbrace{\frac{\partial \vec{h}^{(l-1)}}{\partial \vec{x}}}_{\text{input}}$

$$\boxed{\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{(l-1)}}} \cdot \boxed{\frac{\partial \vec{h}^{(l-1)}}{\partial \vec{x}}} = \underbrace{\frac{\partial \vec{h}^{l}}{\partial \vec{x}}}_{\text{output}}$$

$\langle \boxed{J_g} \leftarrow \text{input} \rangle$

Reverse mode AD

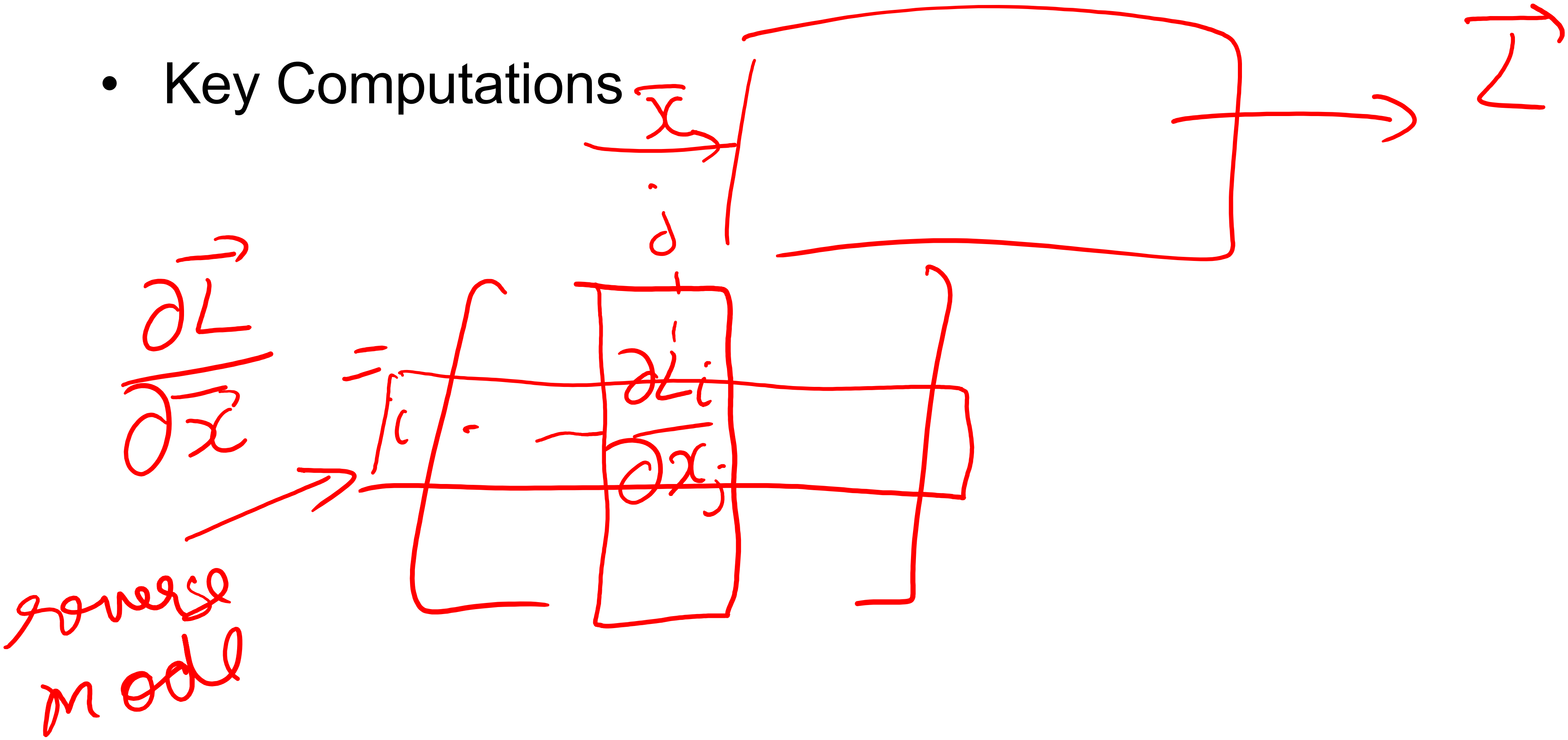


$$\underbrace{\frac{\partial \vec{L}}{\partial \vec{h}^{l}}}_{\text{input}}$$

$$\frac{\partial \vec{L}}{\partial \vec{h}^{l-1}} = \underbrace{\boxed{\frac{\partial \vec{L}}{\partial \vec{h}^{l}}}}_{\text{input}} \cdot \underbrace{\boxed{\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}}}_{\boxed{J_{g_l}}}$$

Forward mode vs Reverse Mode

- Key Computations



Example: Forward mode AD

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

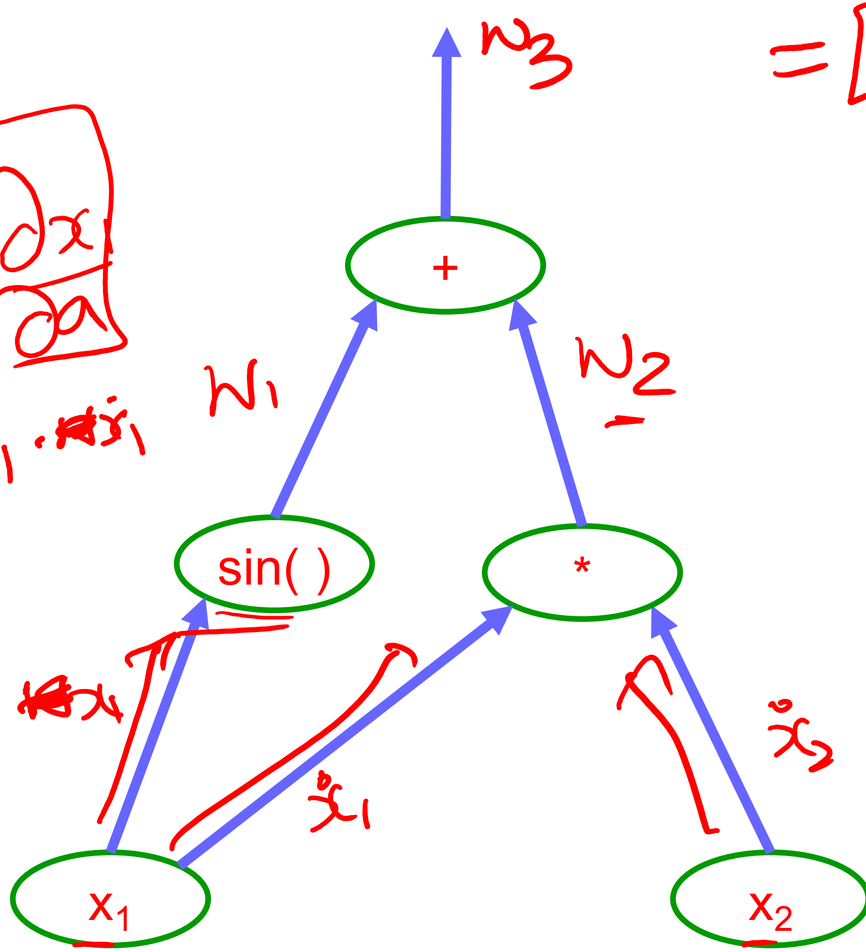
$$\frac{\partial f}{\partial \vec{x}} = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \right]$$

$$\boxed{\frac{\partial f}{\partial a}} = \dot{f}$$

$$\dot{w}_1 = \frac{\partial w_1}{\partial a} = \frac{\partial w_1}{\partial x_1} \cdot \boxed{\frac{\partial x_1}{\partial a}}$$

$$= \cos x_1 \cdot \dot{x}_1$$

$$\dot{x}_1 = \boxed{\frac{\partial x_1}{\partial a}}$$



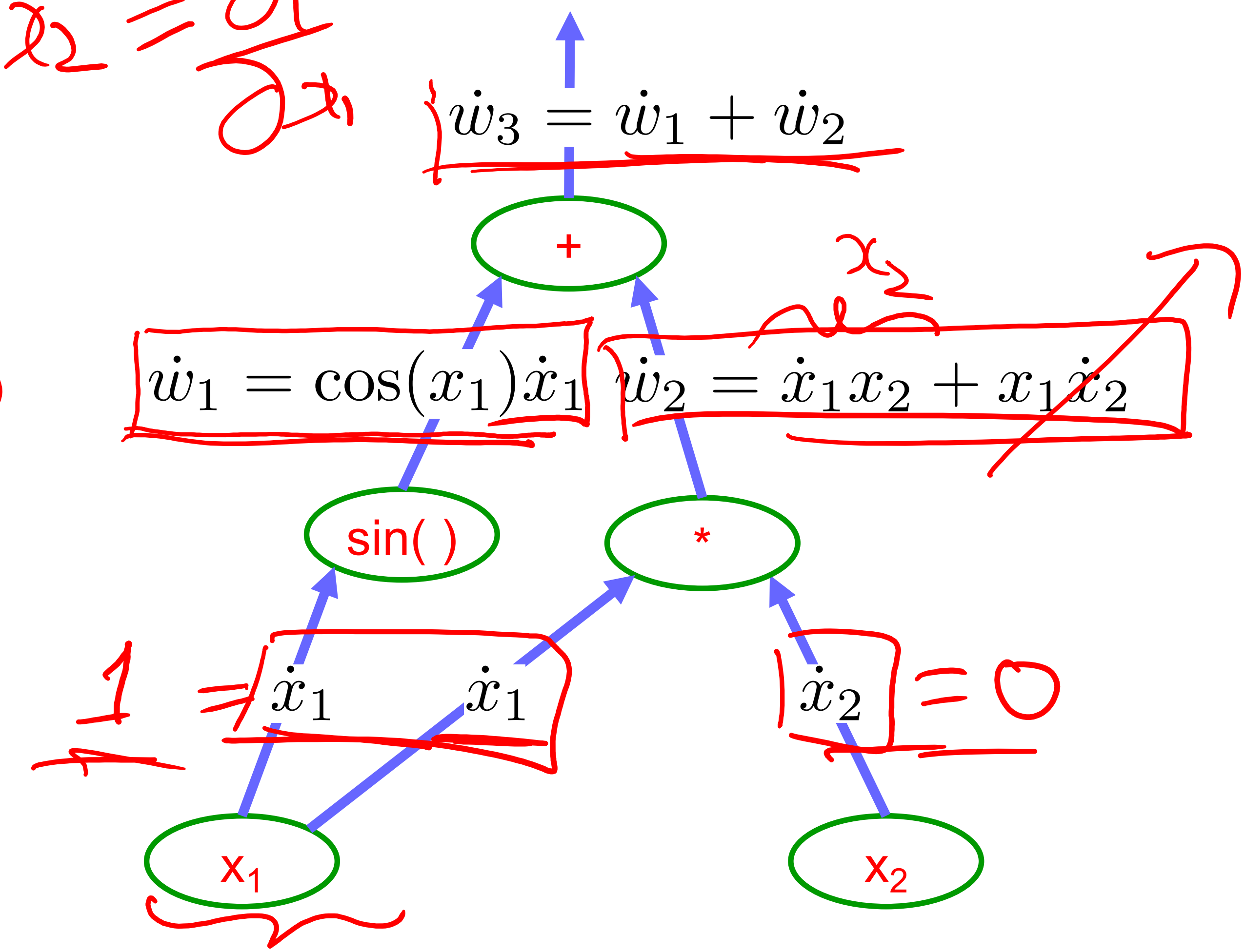
$$\dot{x}_1 = \frac{\partial x_1}{\partial a}$$

Example: Forward mode AD

$$f(x_1, x_2) = \boxed{x_1 x_2 + \sin(x_1)}$$

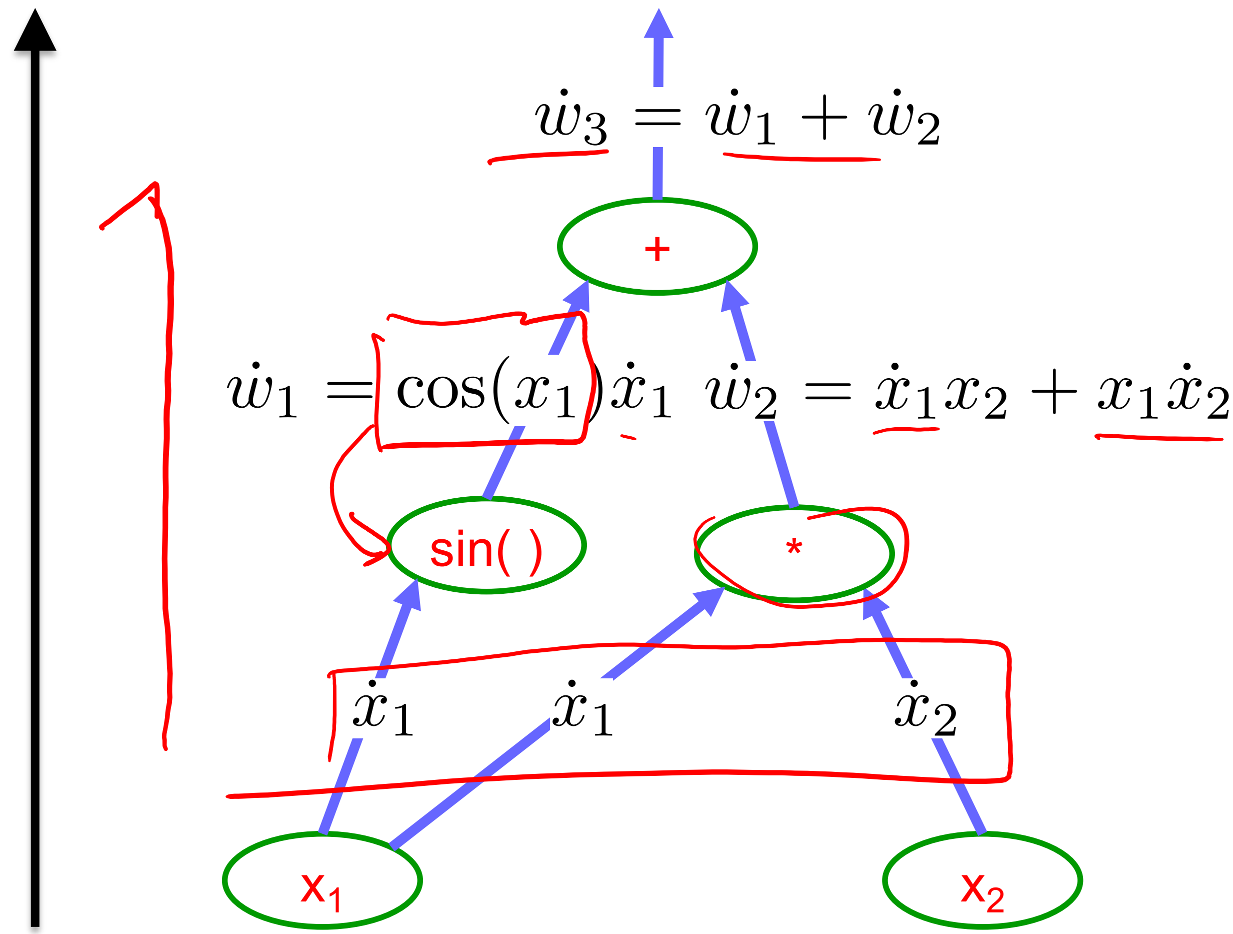
$\frac{\partial f}{\partial a}$

$\cos(x_1) + x_2 = \frac{\partial f}{\partial x_1}$



Example: Forward mode AD

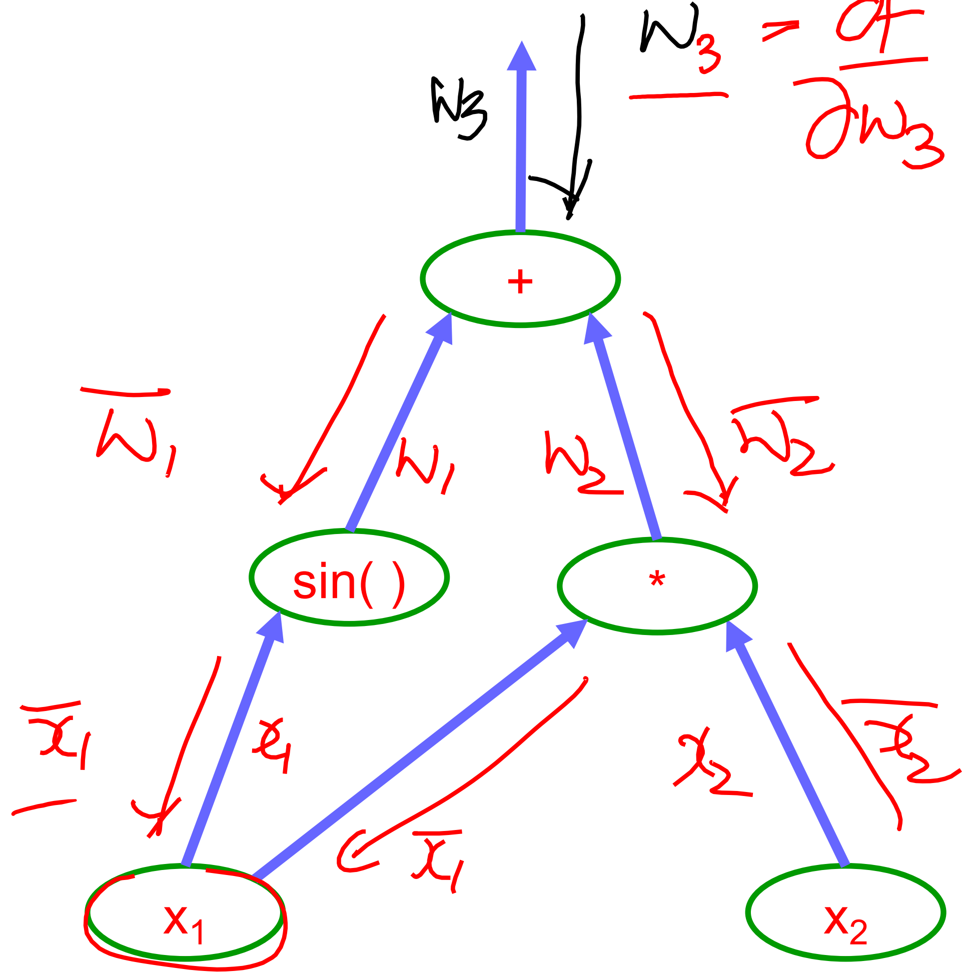
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Example: Reverse mode AD

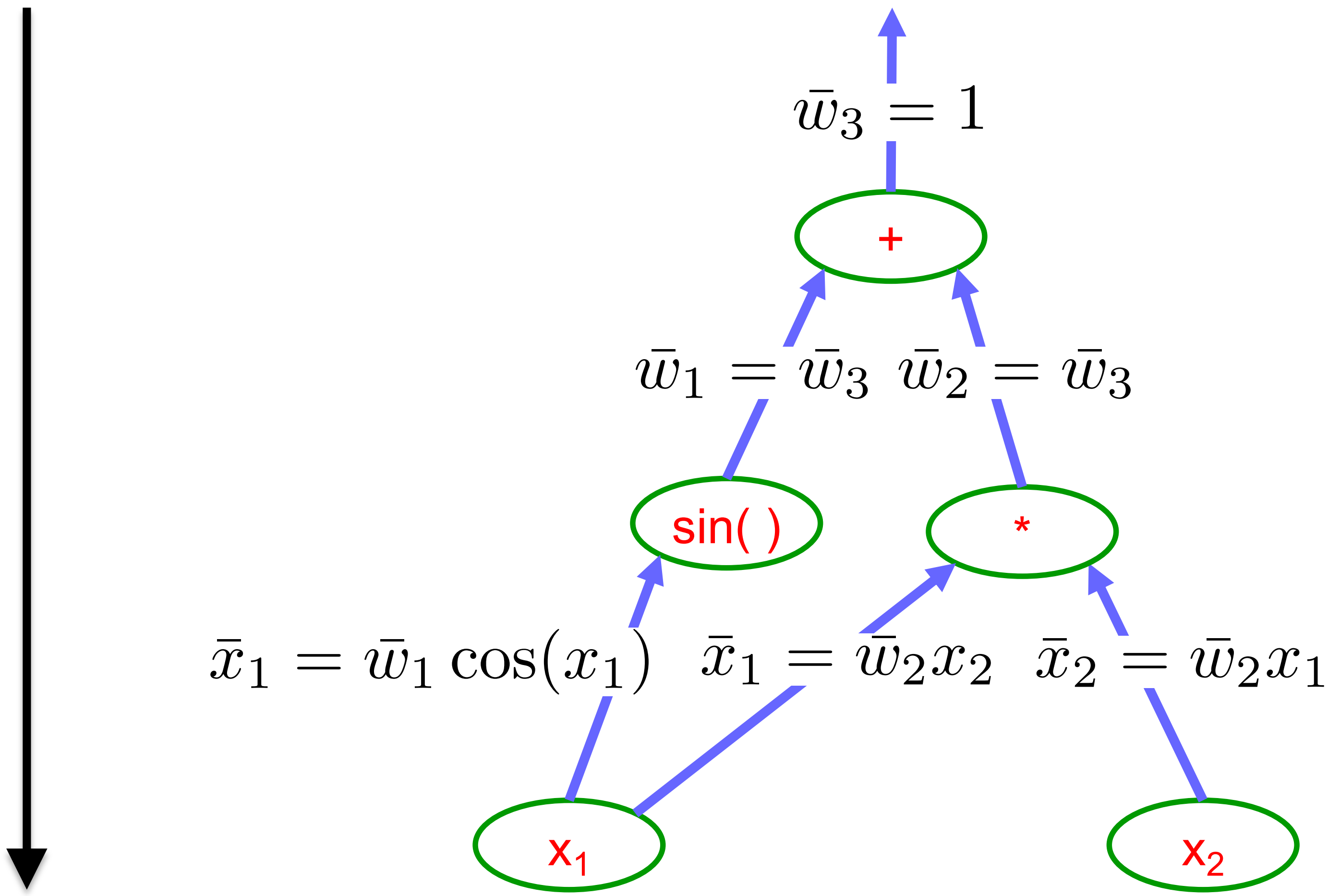
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

$$\bar{w}_3 = \frac{\partial f}{\partial w_3} = 1$$

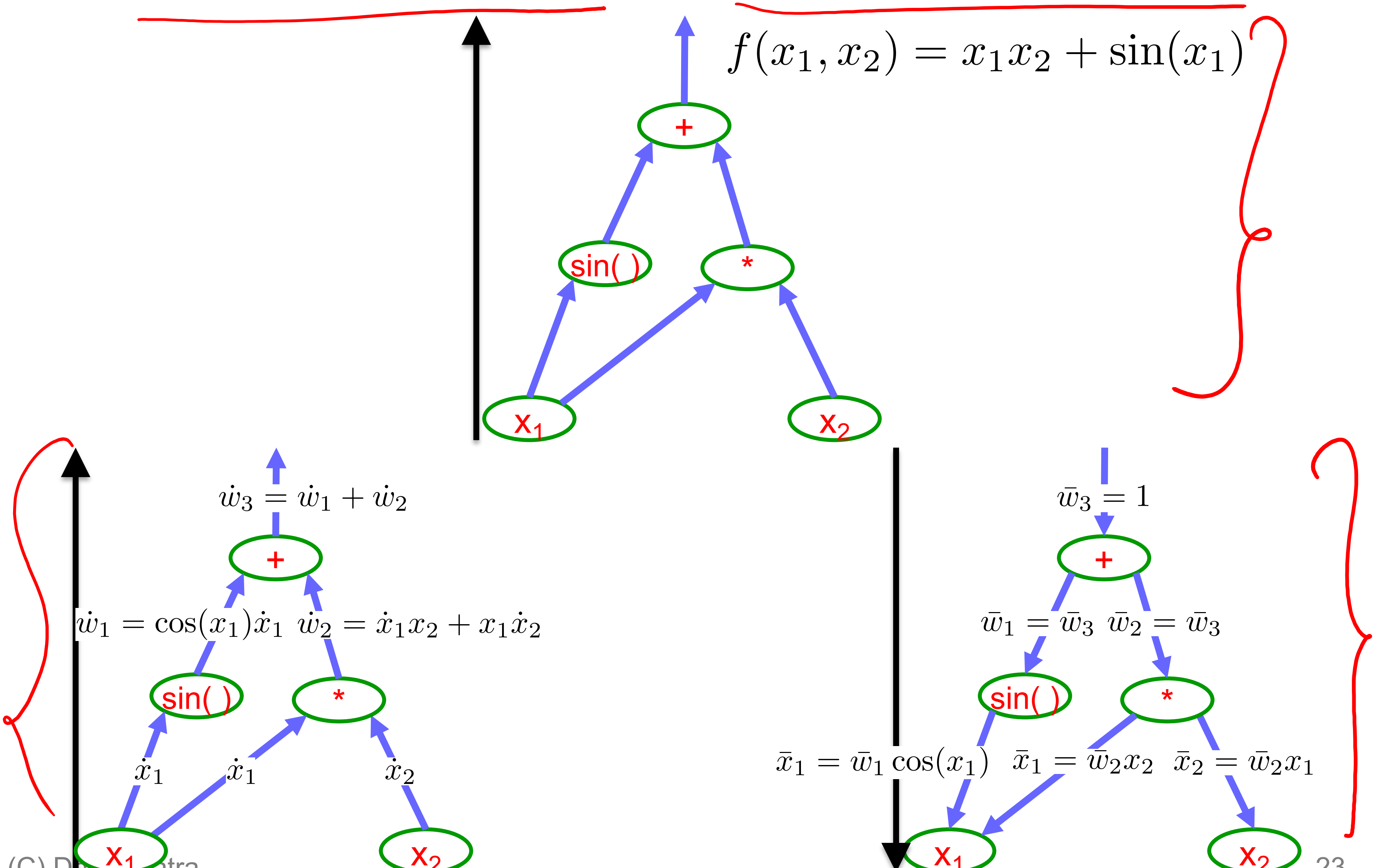


Example: Reverse mode AD

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Forward Pass vs
Forward mode AD vs Reverse Mode AD



Forward mode vs Reverse Mode

- What are the differences?
- Which one is faster to compute?
  - Forward or backward?

Forward mode vs Reverse Mode

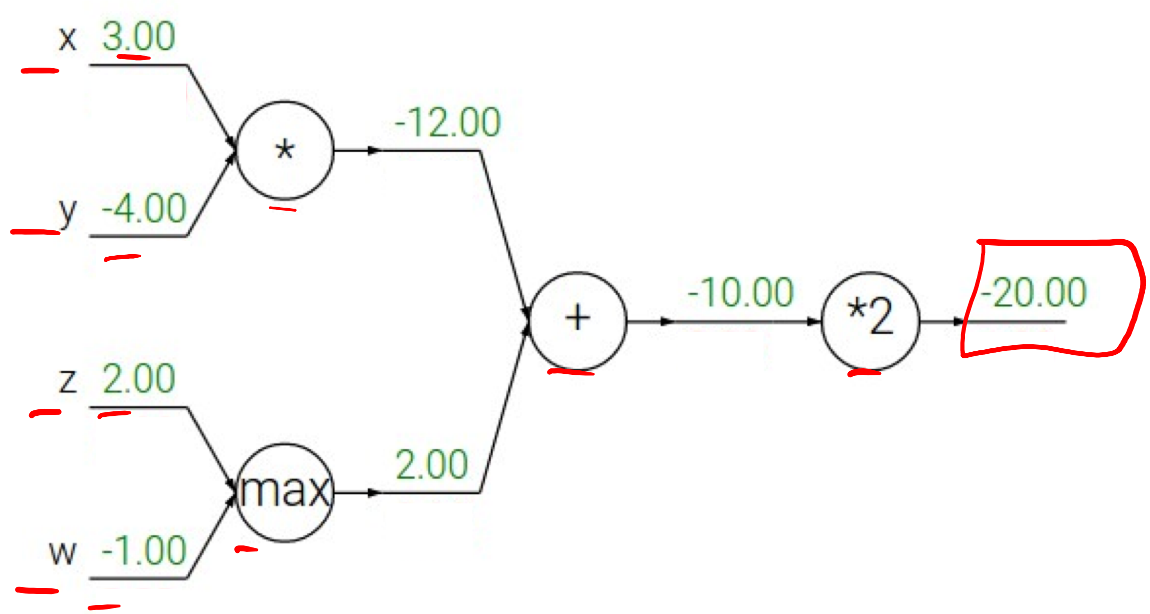
- What are the differences?

- Which one is faster to compute?
  - Forward or backward?

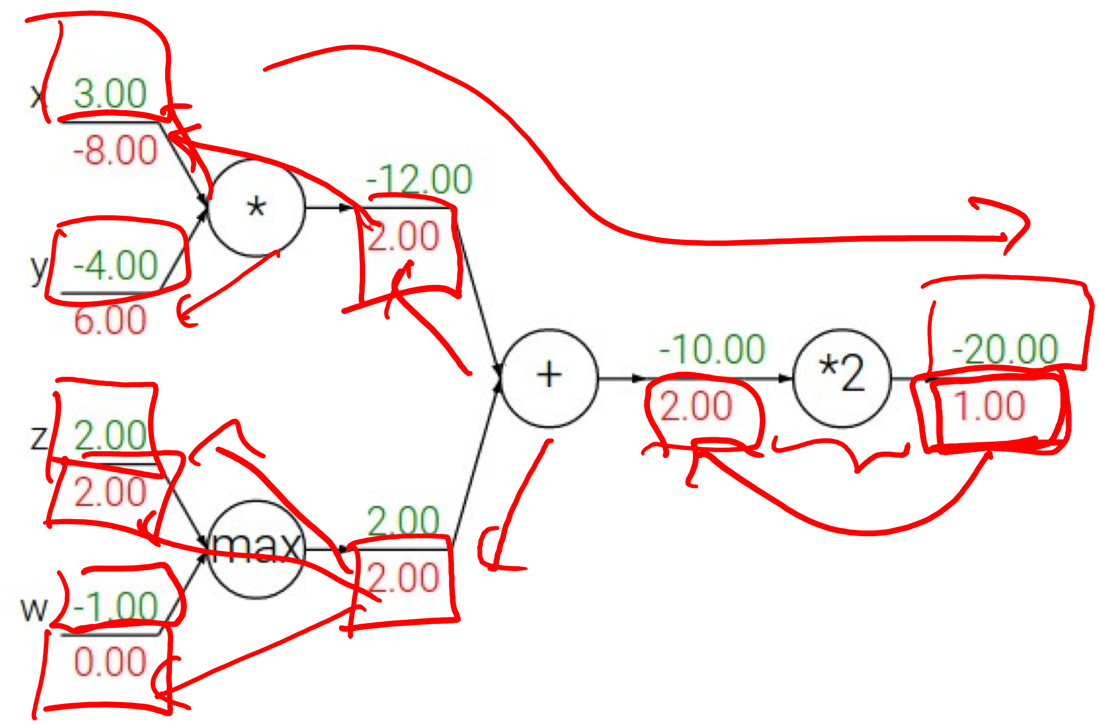
- Which one is more memory efficient (less storage)?
  - Forward or backward?

Patterns in backward flow

$f(\cdot\ \cdot\cdot)$
$= 2(xy + \max\{z, w\})$

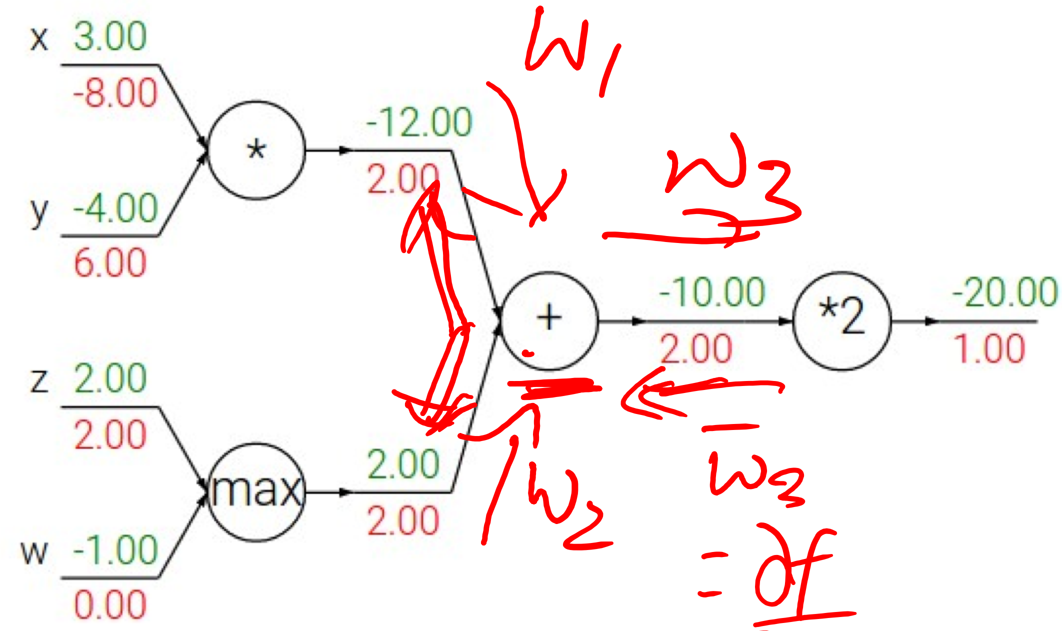


Patterns in backward flow



Patterns in backward flow

add gate: gradient distributor



$\bar{w}_3 = \frac{\partial f}{\partial w_3}$

$w_3 = w_1 + w_2$

$\frac{\partial w_3}{\partial w_1} = 1$

$\boxed{\frac{\partial f}{\partial w_1}} = \boxed{\frac{\partial f}{\partial w_3}} \cdot \boxed{\cancel{\frac{\partial w_3}{\partial w_1}}}$

Patterns in backward flow

$w_2 = \max\{z, w\}$

$\max(0, x)$

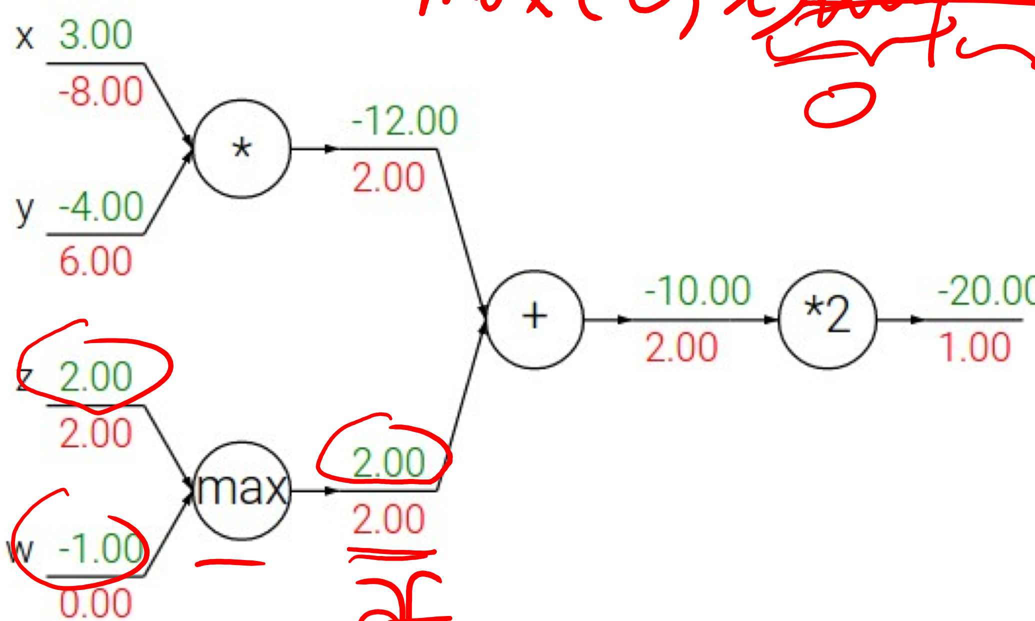
0

add gate: gradient distributor

Q: What is a **max** gate?

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

$\frac{\partial f}{\partial w}$

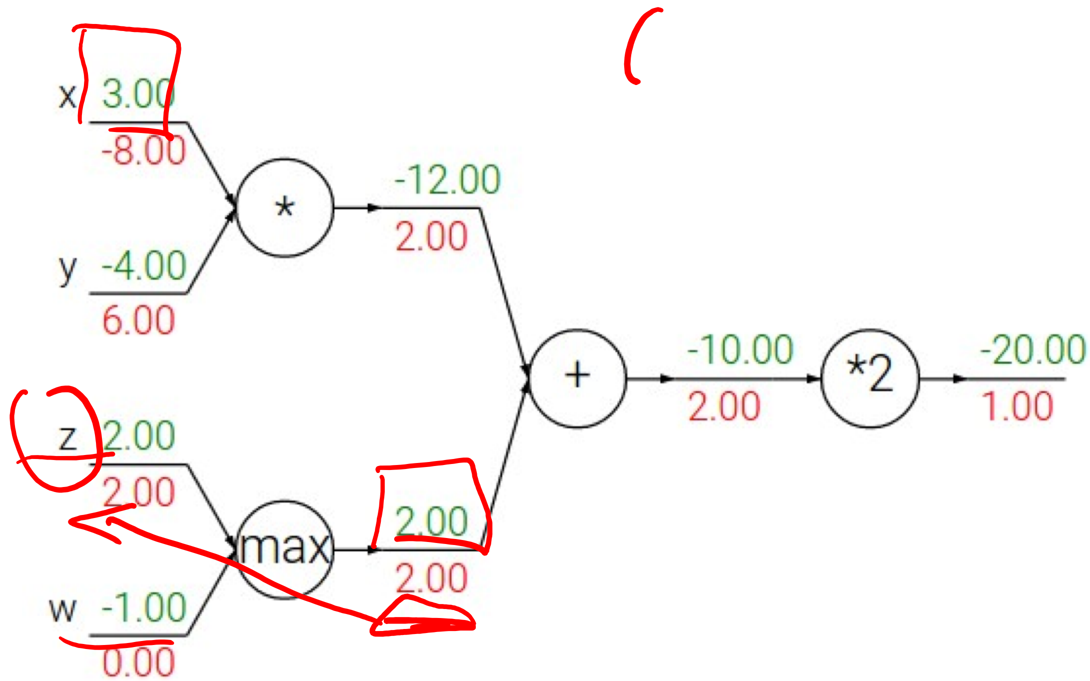


$\frac{\partial f}{\partial w_2}$

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

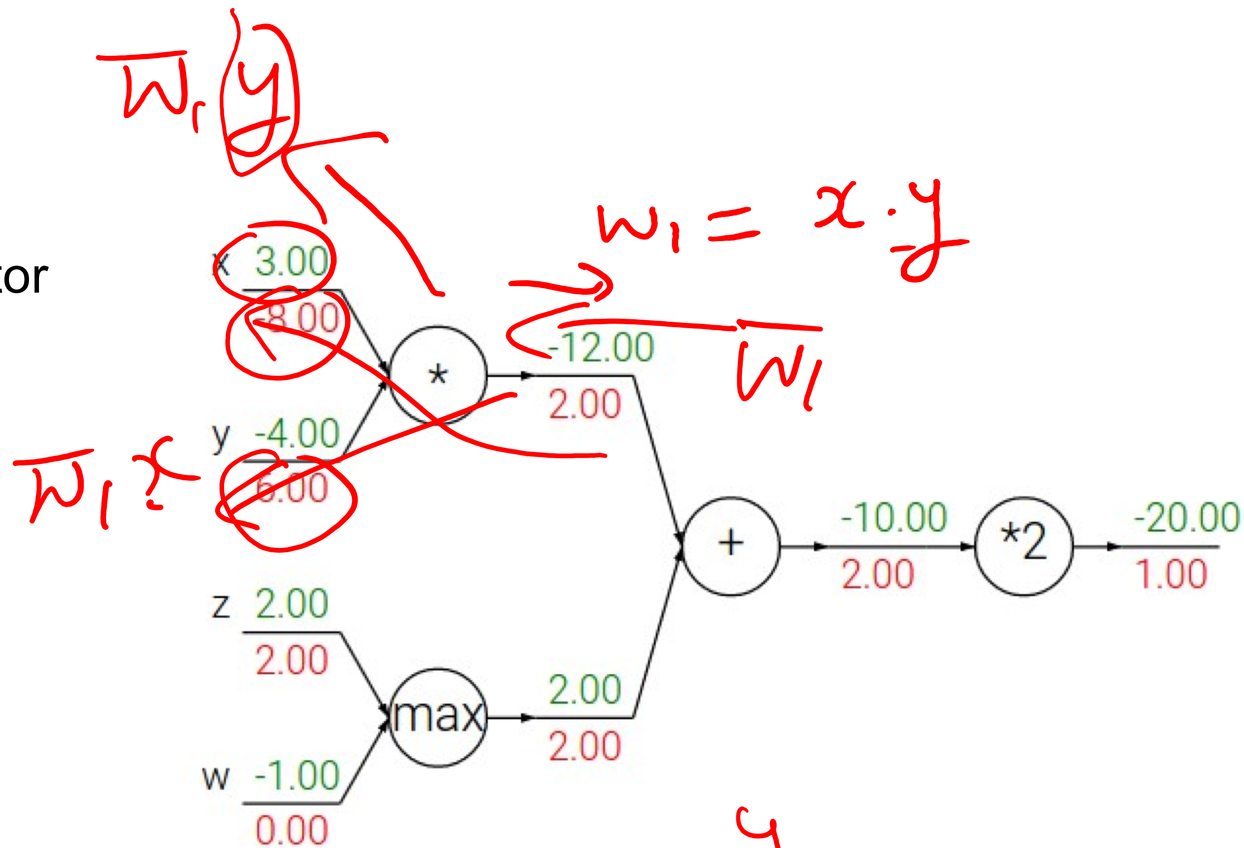


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

Q: What is a **mul** gate?



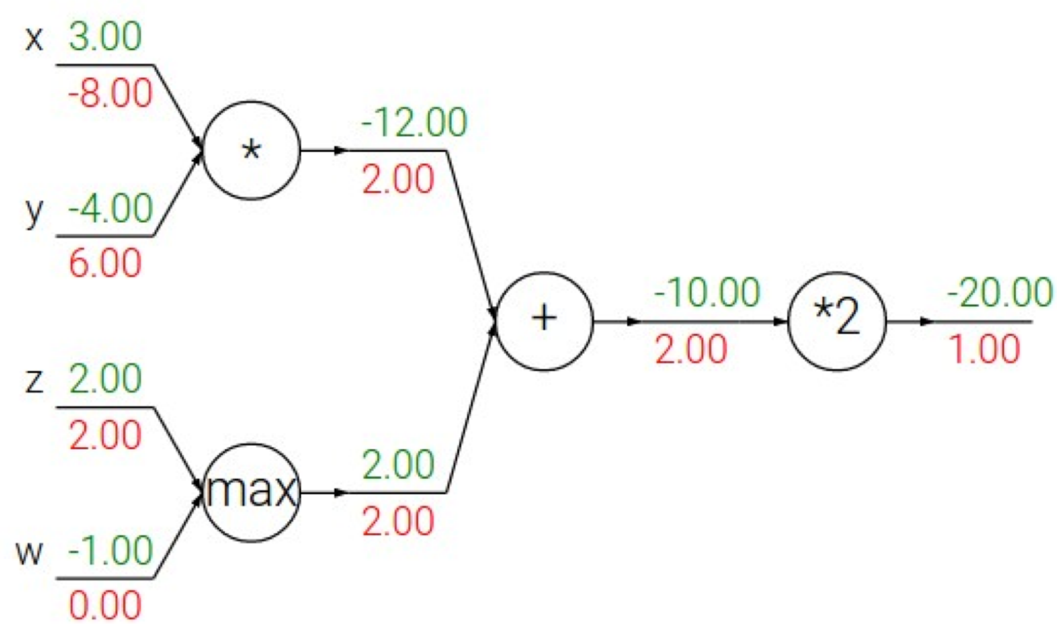
$$\frac{\partial f}{\partial x} = \boxed{\frac{\partial f}{\partial w_1}}\ \overbrace{\boxed{\frac{\partial w_1}{\partial x}}}^{y}$$

Patterns in backward flow

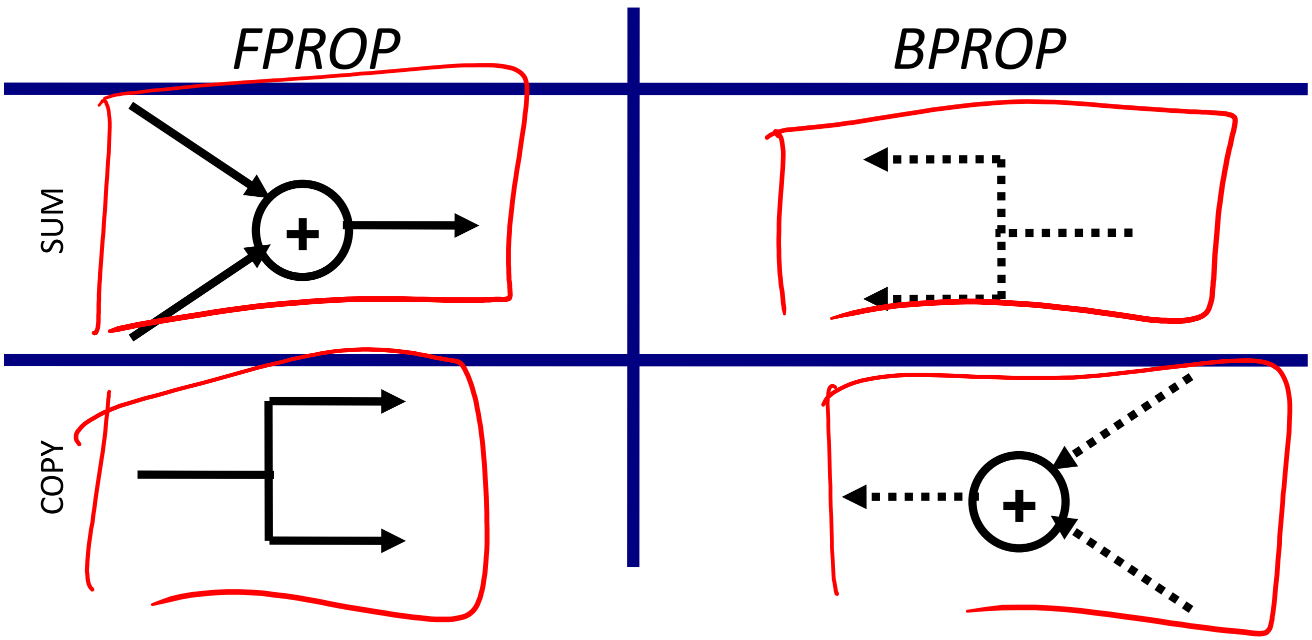
add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher



Duality in Fprop and Bprop



Plan for Today

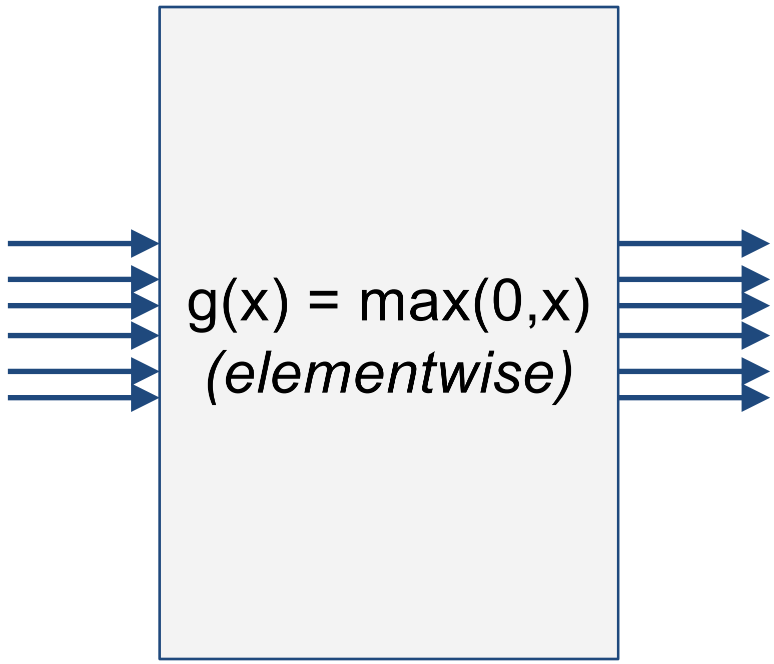
- (Finish) Computing Gradients
  - Backprop in FC+ReLU NNs
- Convolutional Neural Networks

Jacobian of ReLU

$\vec{h}^{\ell-1} \in \mathbb{R}^{4096}$

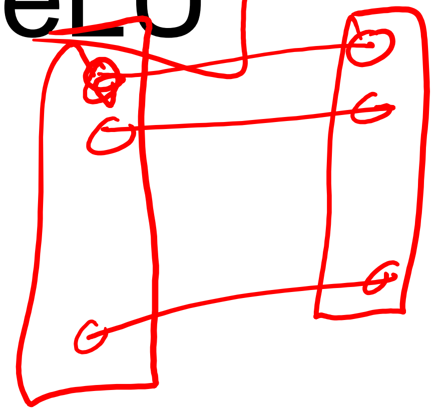
$\vec{h}^{\ell} \in \mathbb{R}^{4096}$

4096-d
input vector



4096-d
output vector

Jacobian of ReLU



4096-d
input vector

$g(x) = \max(0,x)$
(elementwise)

4096-d
output vector

$$\frac{\partial h^l_i}{\partial h^{l-1}_j} = 0 \quad \forall\, i \neq j$$

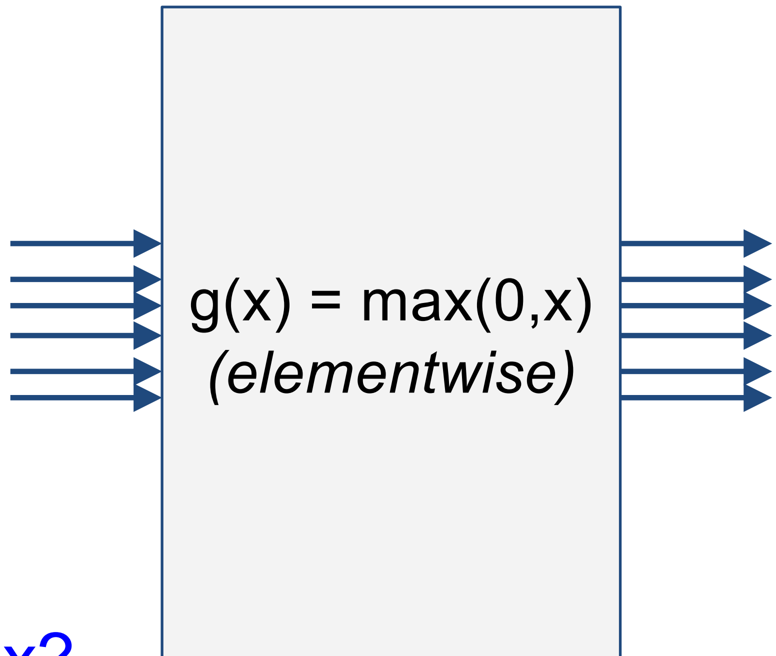
Q: what is the size of the Jacobian matrix?

$$\frac{\partial \vec{h}^l}{\partial \vec{h}^{l-1}} = i \begin{bmatrix} \cdots \frac{\partial h^l_i}{\partial h^{l-1}_j} & & \\ & & \\ & & \end{bmatrix} \rightarrow \begin{bmatrix} \ddots & & 0 \\ & \ddots & \\ 0 & & \ddots \end{bmatrix}$$

4096x4096

Jacobian of ReLU

4096-d
input vector

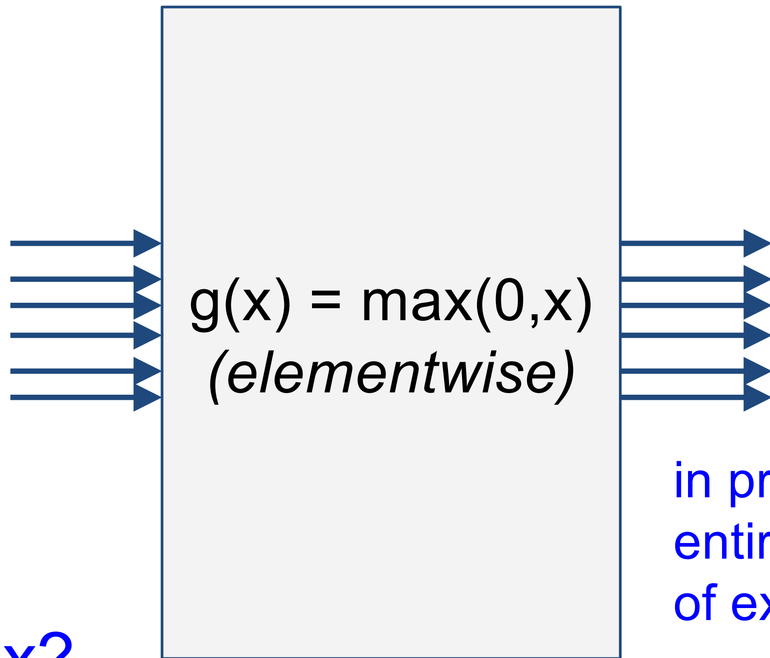


4096-d
output vector

Q: what is the size of the Jacobian matrix?
[4096 x 4096!]

Jacobian of ReLU

4096-d
input vector



4096-d
output vector

Q: what is the size of the Jacobian matrix?
[4096 x 4096!]

in practice we process an entire minibatch (e.g. 100) of examples at one time:

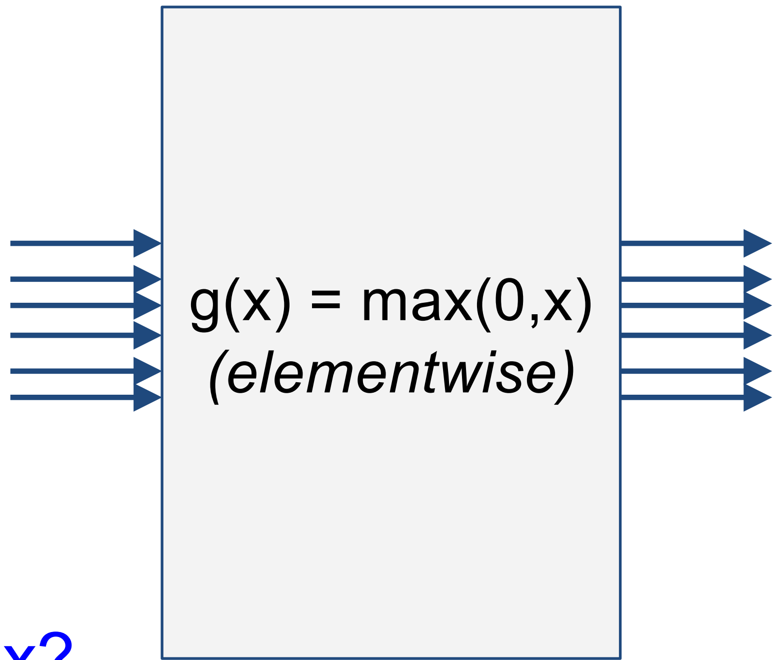
i.e. Jacobian would technically be a

$$\boxed{[409{,}600 \times 409{,}600]}$$

matrix :\

Jacobian of ReLU

4096-d
input vector

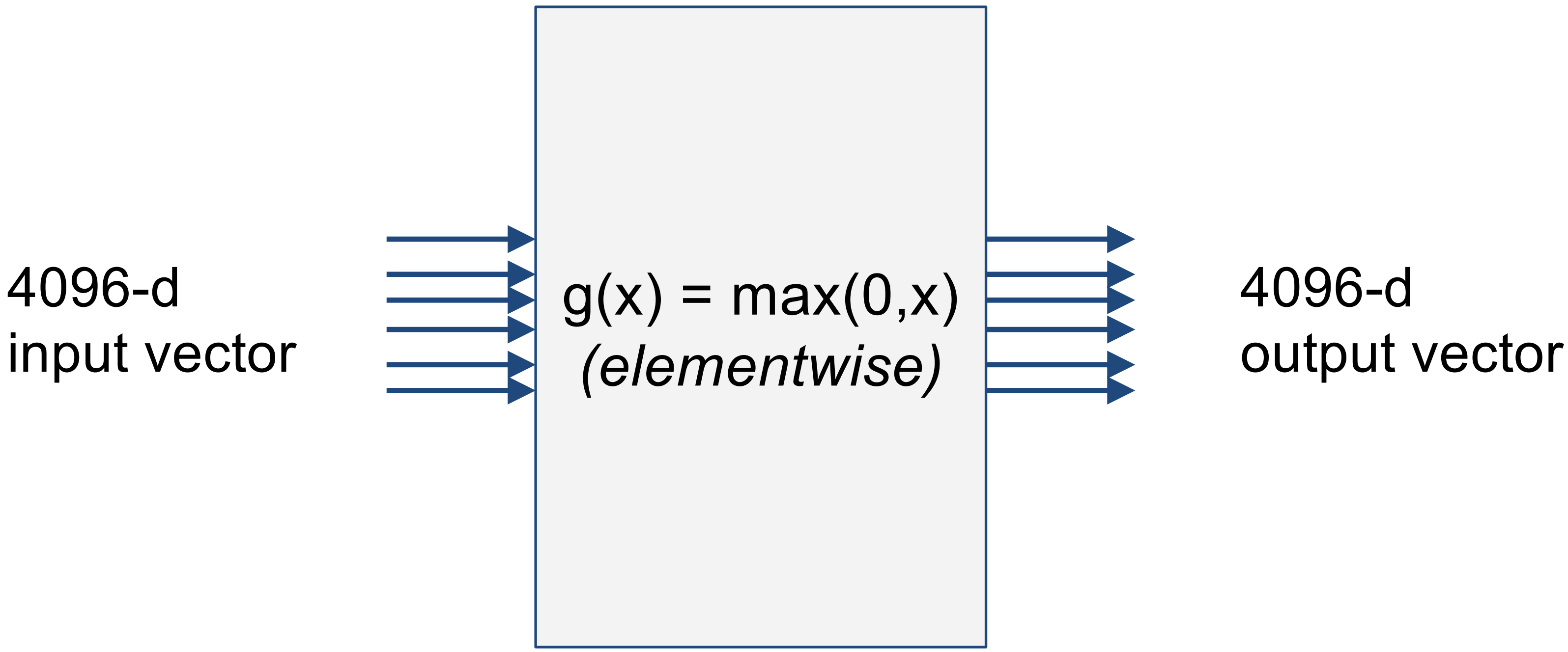


4096-d
output vector

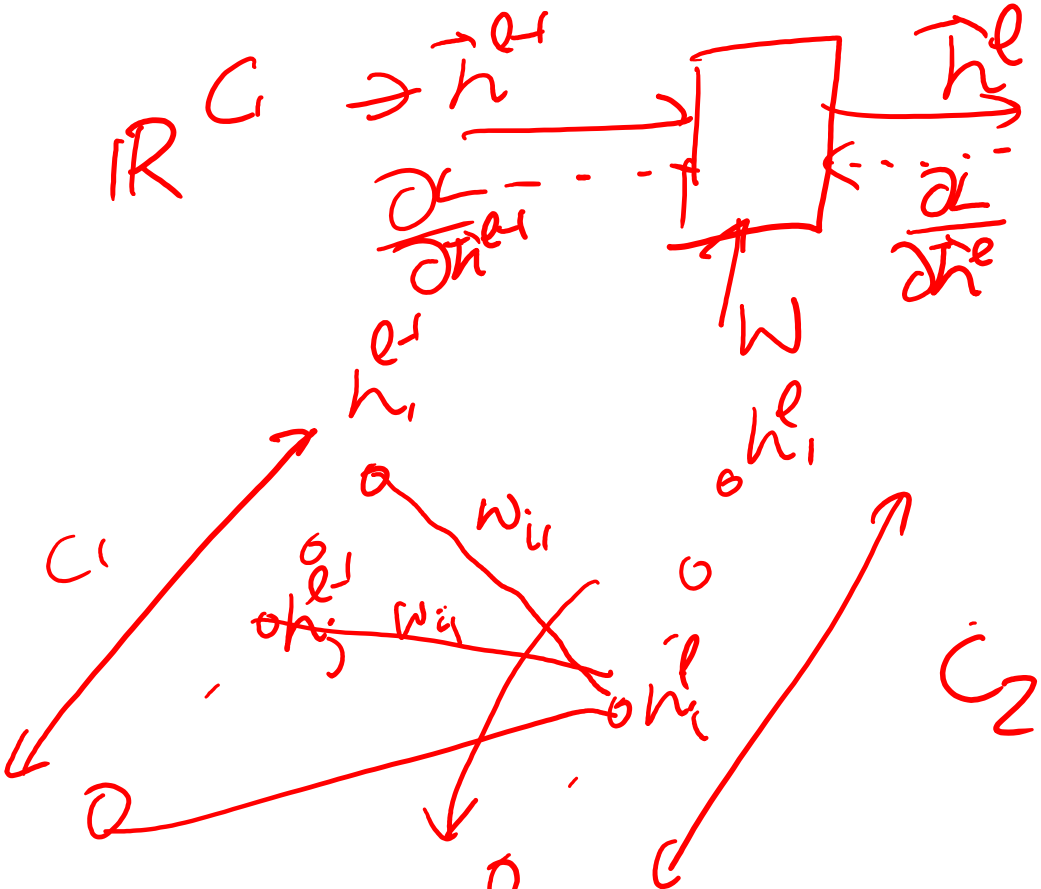
Q: what is the size of the Jacobian matrix?
[4096 x 4096!]

Q2: what does it look like?

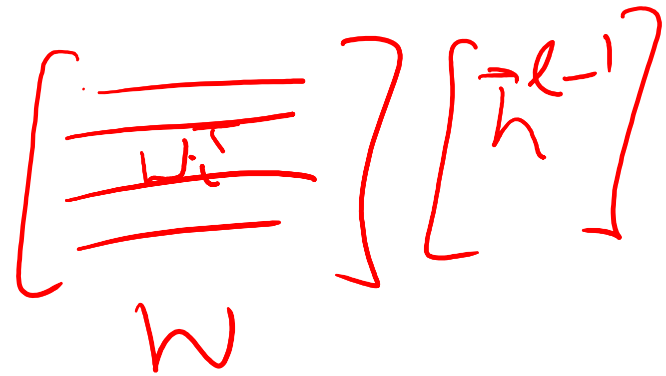
Jacobian of ReLU



Jacobians of FC-Layer



$$h_i^{\ell} = \vec{w}_i^{T} \vec{h}^{\ell-1}$$

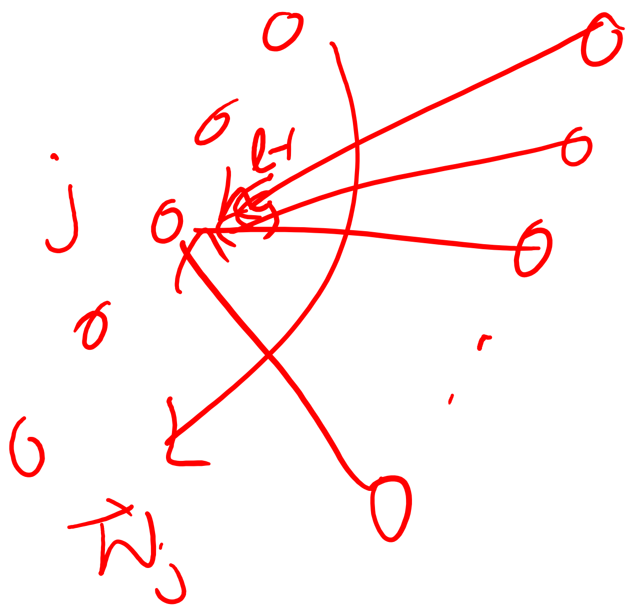


$$\vec{h}^{\ell} = \boxed{W} \vec{h}^{\ell-1} \quad \text{F-Pass}$$

$$\vec{w}_i$$

$$\boxed{\frac{\partial L}{\partial \vec{h}^{\ell-1}}} = \boxed{\frac{\partial L}{\partial \vec{h}^{\ell}}} \cdot \underbrace{\boxed{\frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}}}}_{W} \quad \text{B-Pass}$$

Jacobians of FC-Layer



Jacobians of FC-Layer

Jacobians of FC-Layer

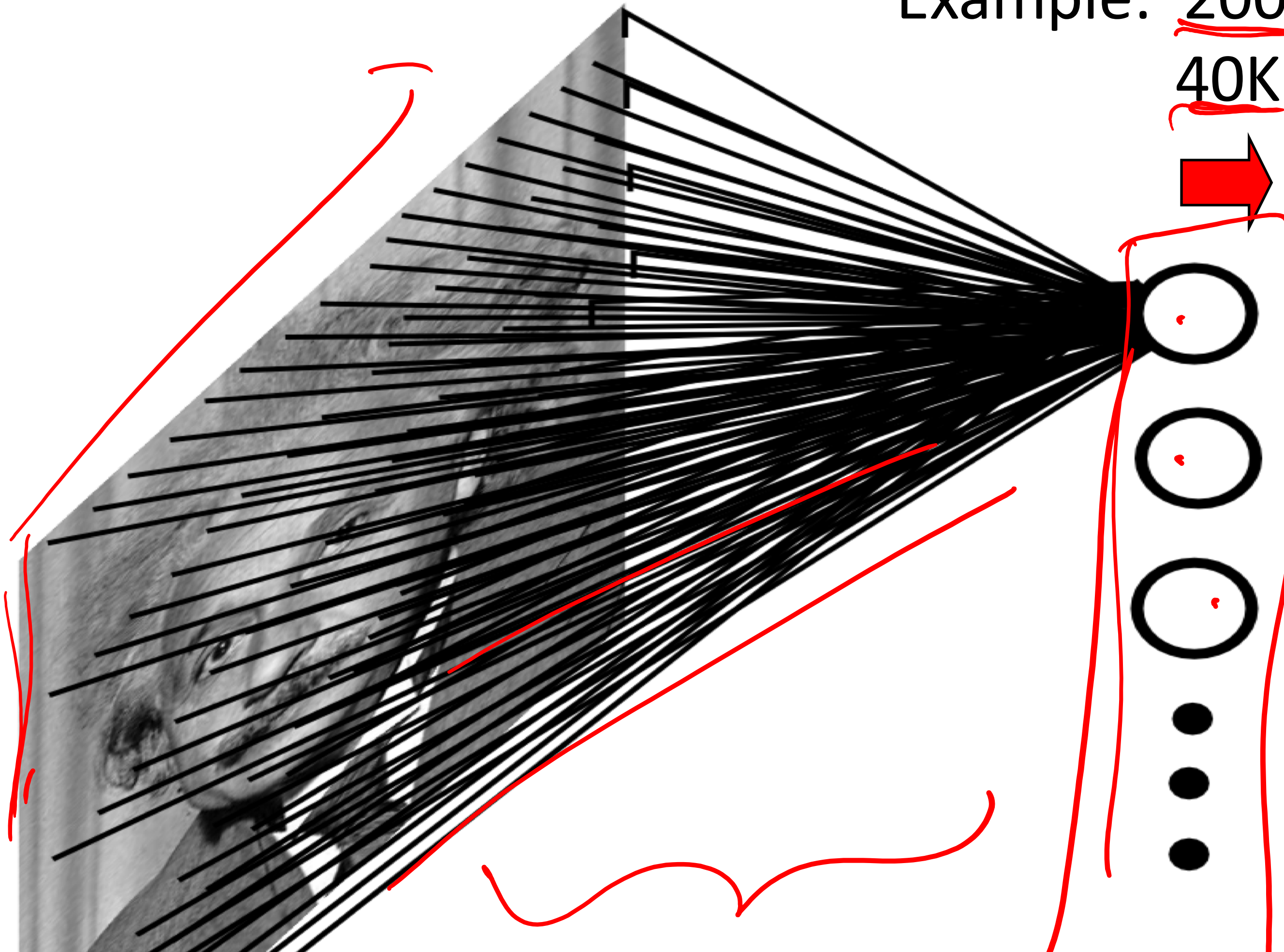
Convolutional Neural Networks

(without the brain stuff)

Fully Connected Layer

Example: 200x200 image 4×10^4
40K hidden units

~2B parameters!!!



4×10^4

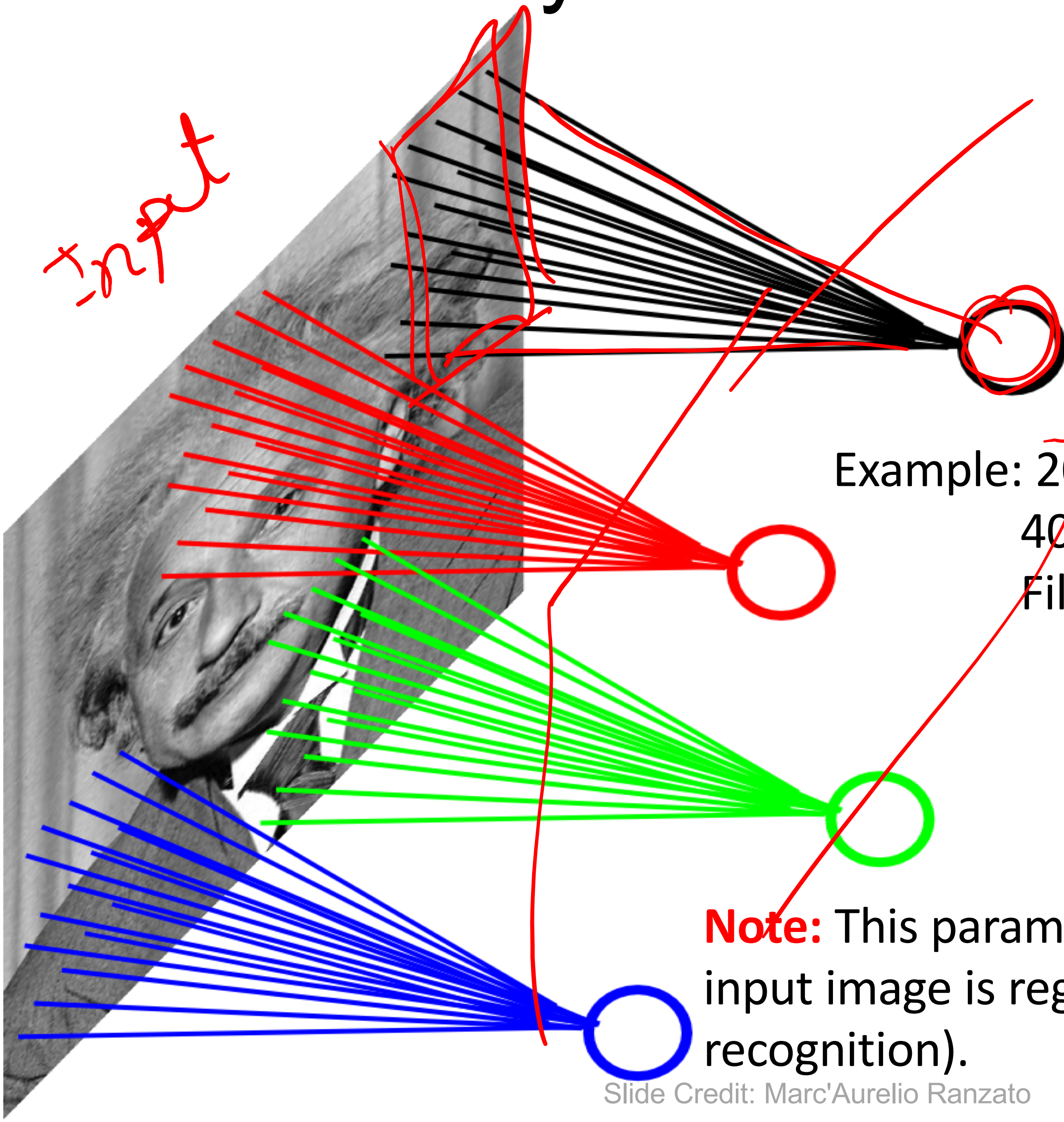
4×10^4 4×10^4

W $\boxed{1.6\times10^9}$

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

Slide Credit: Marc'Aurelio Ranzato

Locally Connected Layer



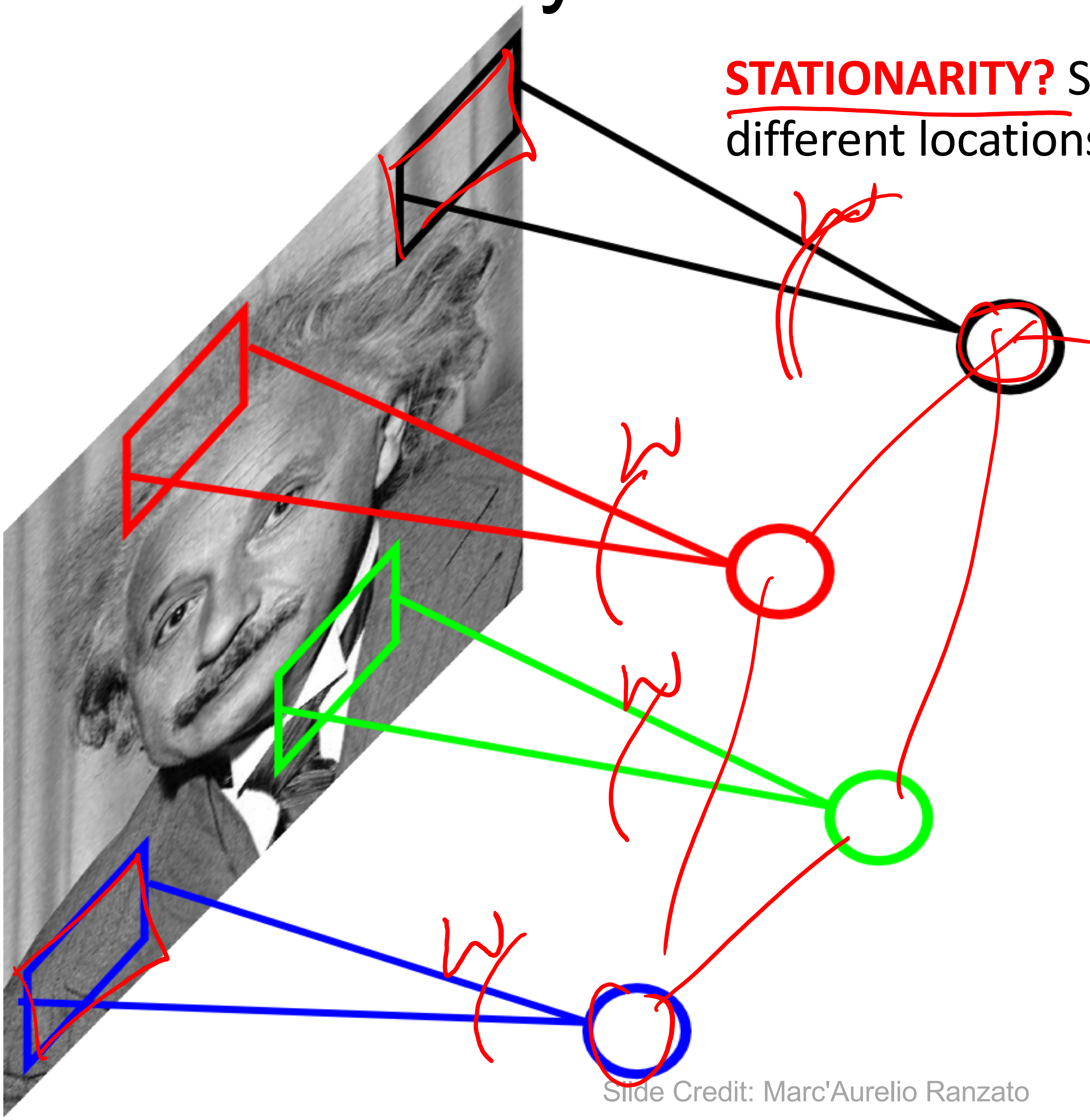
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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

Slide Credit: Marc'Aurelio Ranzato

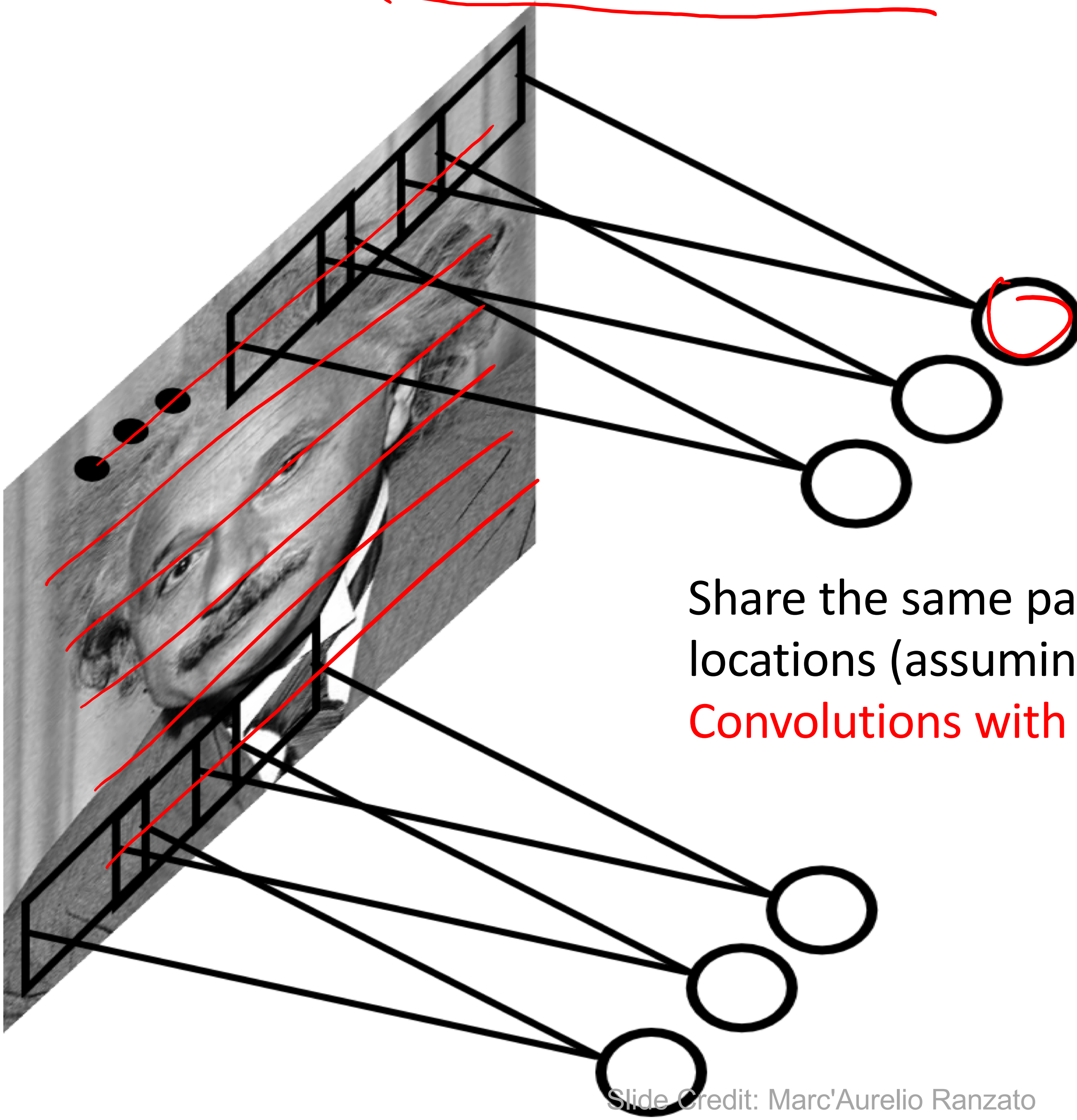
Locally Connected Layer

STATIONARITY? Statistics is similar at different locations



Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Convolutions!

math → CS → programming

Convolutions for mathematicians

$x(t) \quad y(t) \quad w(-t)$

$$y(t) = (x * w)(t)$$

$$= (w * x)(t)$$

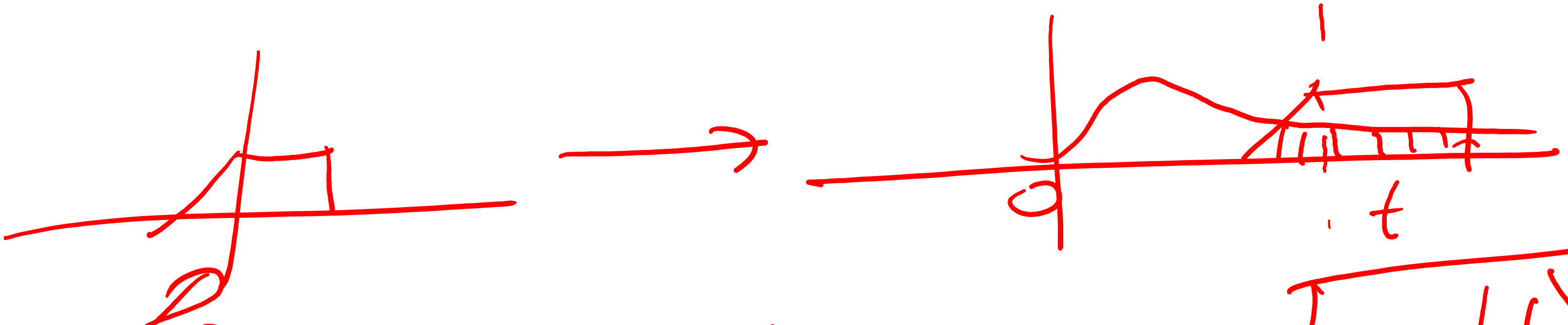
$$= \int_{a=-\infty}^{\infty} \underline{x(t-a)\, w(a)}\, da$$

$$= \int_{a=\infty}^{\infty} x(a)\, \underline{w(t-a)}\, da$$

Convolutions for mathematicians

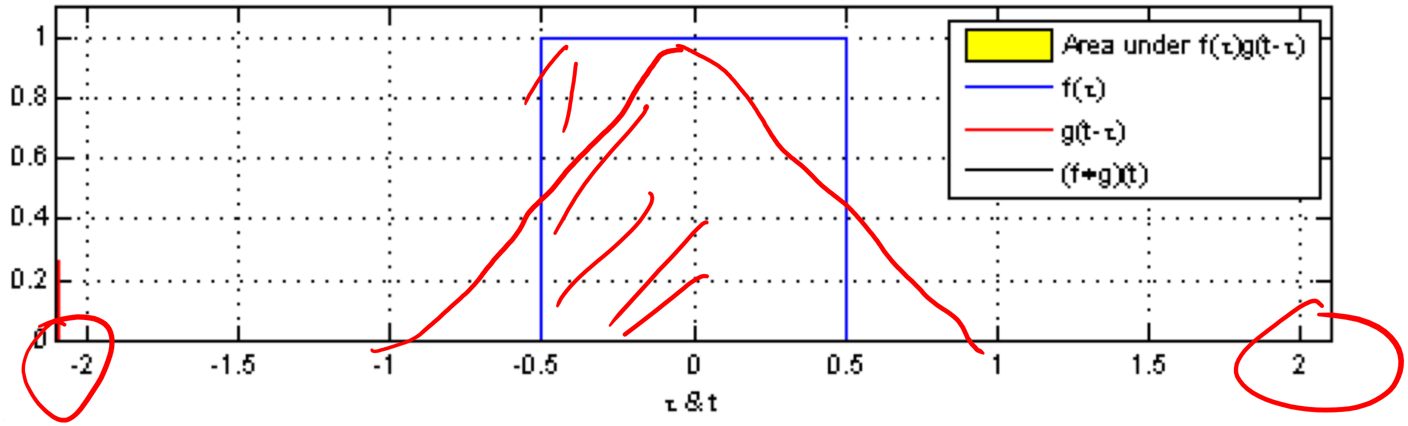
$w(a) \longrightarrow w(-a)$

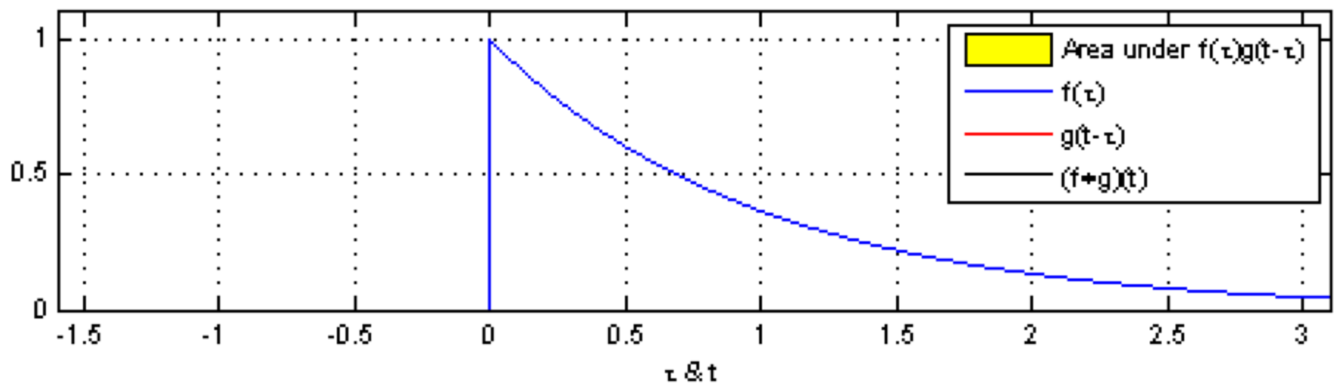
$w(-a) \xrightarrow{\text{translate}} w(-(a-t))$



$$\int_{a=-\infty}^{\infty} x(t)\, w(t-a) = \boxed{y(t)}$$

$$y(t_1, t_2) = \int_{a=-\infty}^{\infty} \int_{b=-\infty}^{\infty} x(t_1 - a, t_2 - b)\, w(a, b)\, da\, db$$

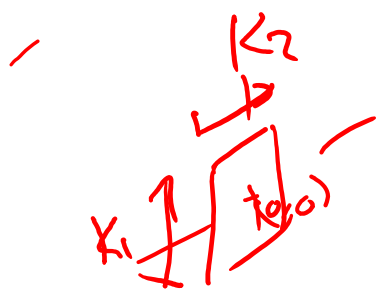




"Convolution of box signal with itself2" by Convolution_of_box_signal_with_itself.gif: Brian Ambergderivative work: Tinos (talk) - Convolution_of_box_signal_with_itself.gif. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself2.gif#/media/File:Convolution_of_box_signal_with_itself2.gif

Convolutions for computer scientists

$$y[r,c] = \sum_{\boxed{a=-\infty}}^{\infty} \sum_{b=-\infty}^{\infty} x[r-a, c-b]\, w[a,b]$$

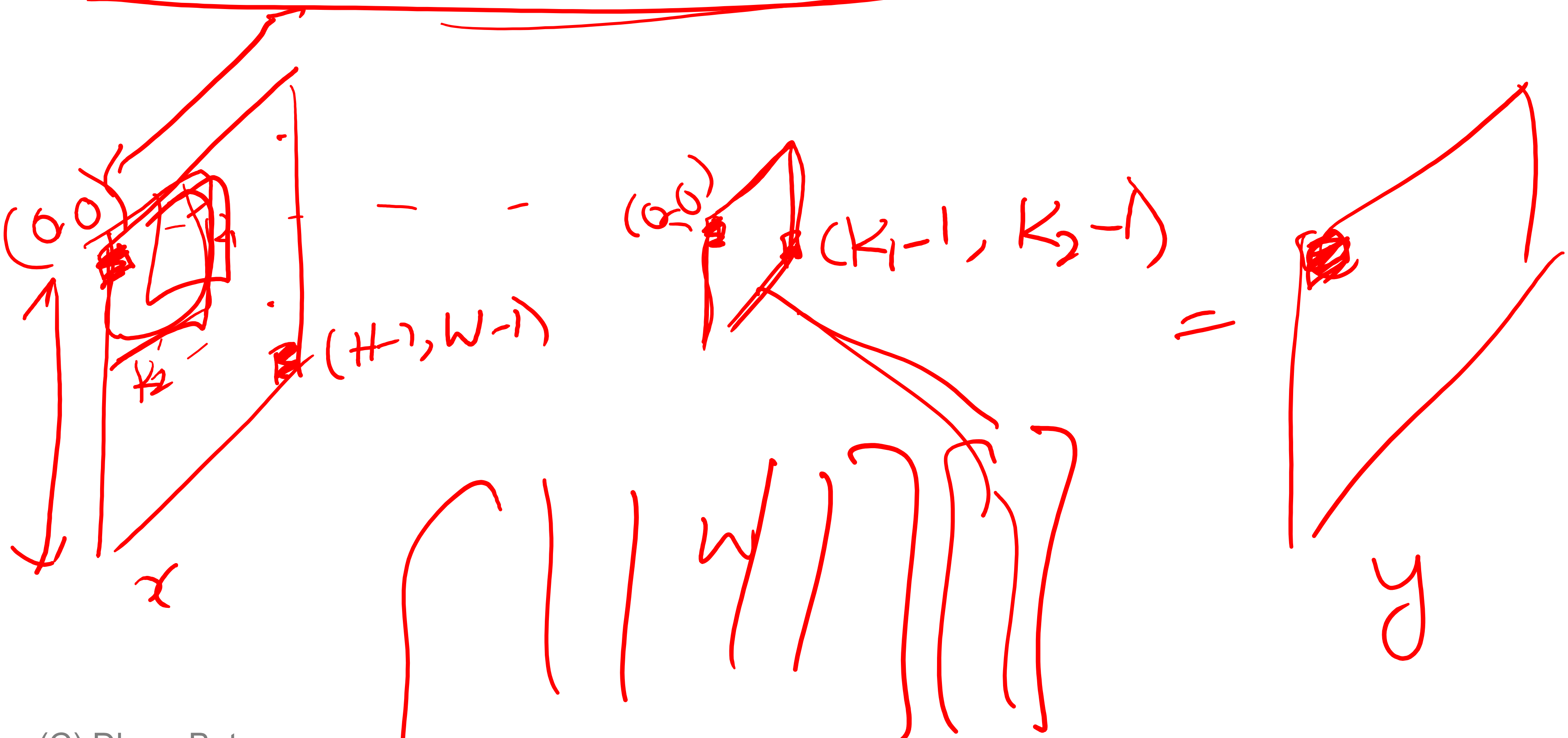


$$y[r,c] = \sum_{a=-\frac{K_1-1}{2}}^{\frac{K_1-1}{2}} \sum_{b=-\frac{K_2-1}{2}}^{\frac{K_2-1}{2}} [\quad]$$

Convolutions for computer scientists

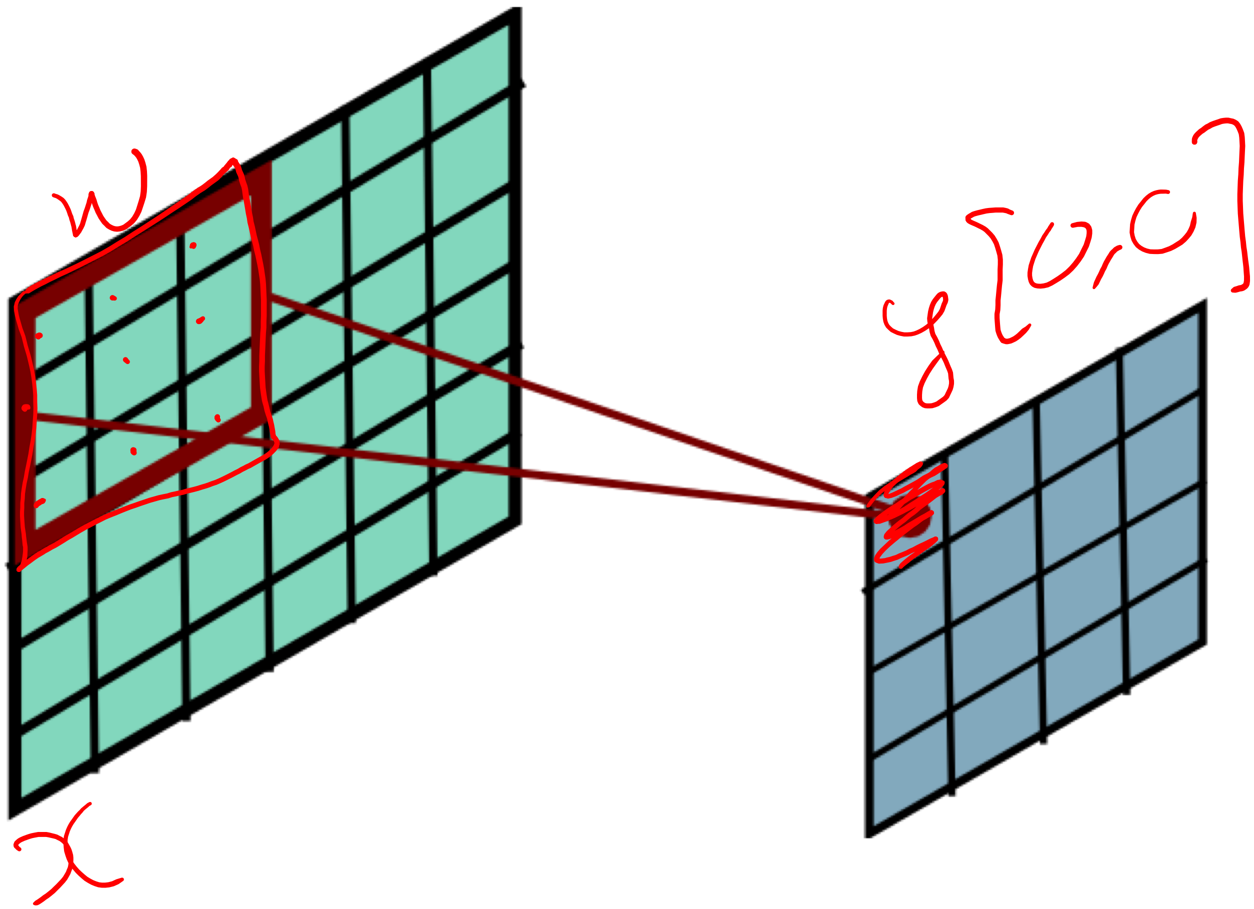
Convolutions for programmers

$$y[r,c] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} x[r+a, c+b]\, w[a,b]$$

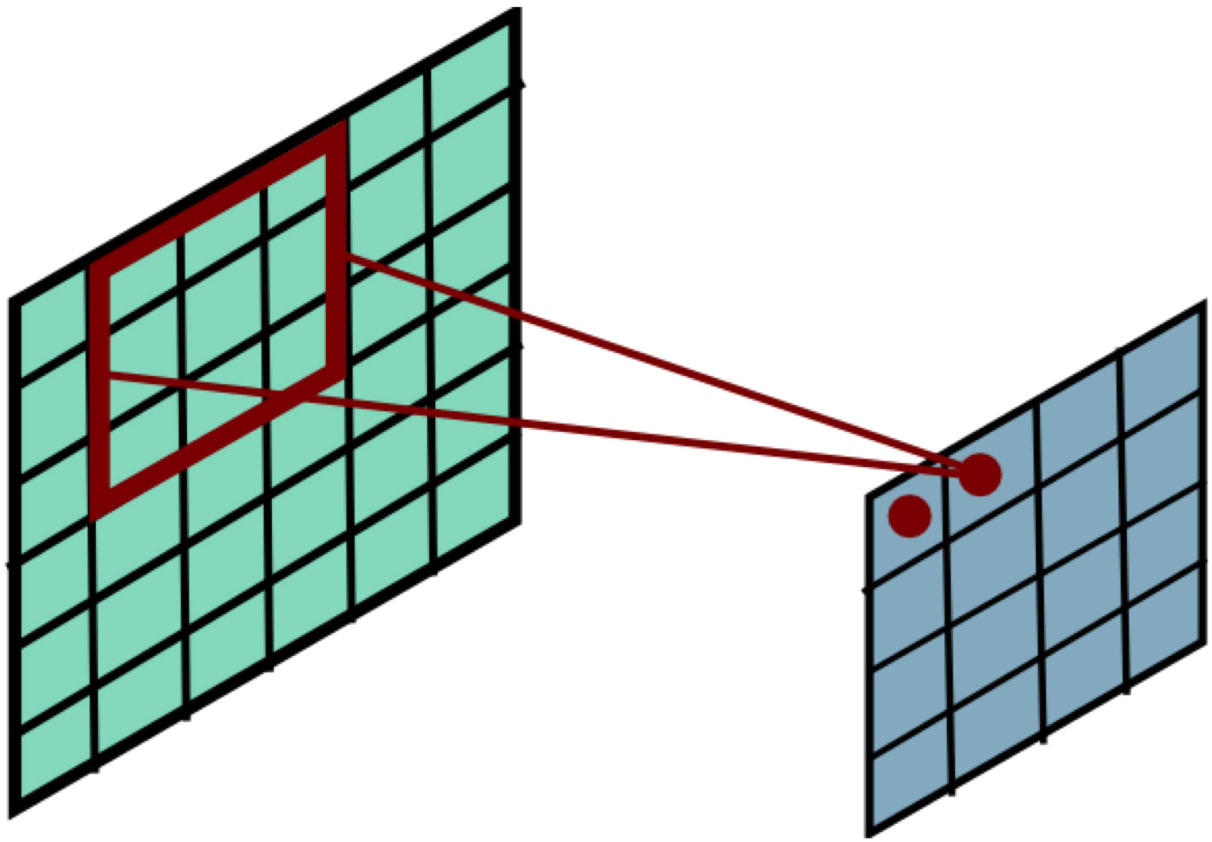


Convolutions for programmers

Convolutional Layer

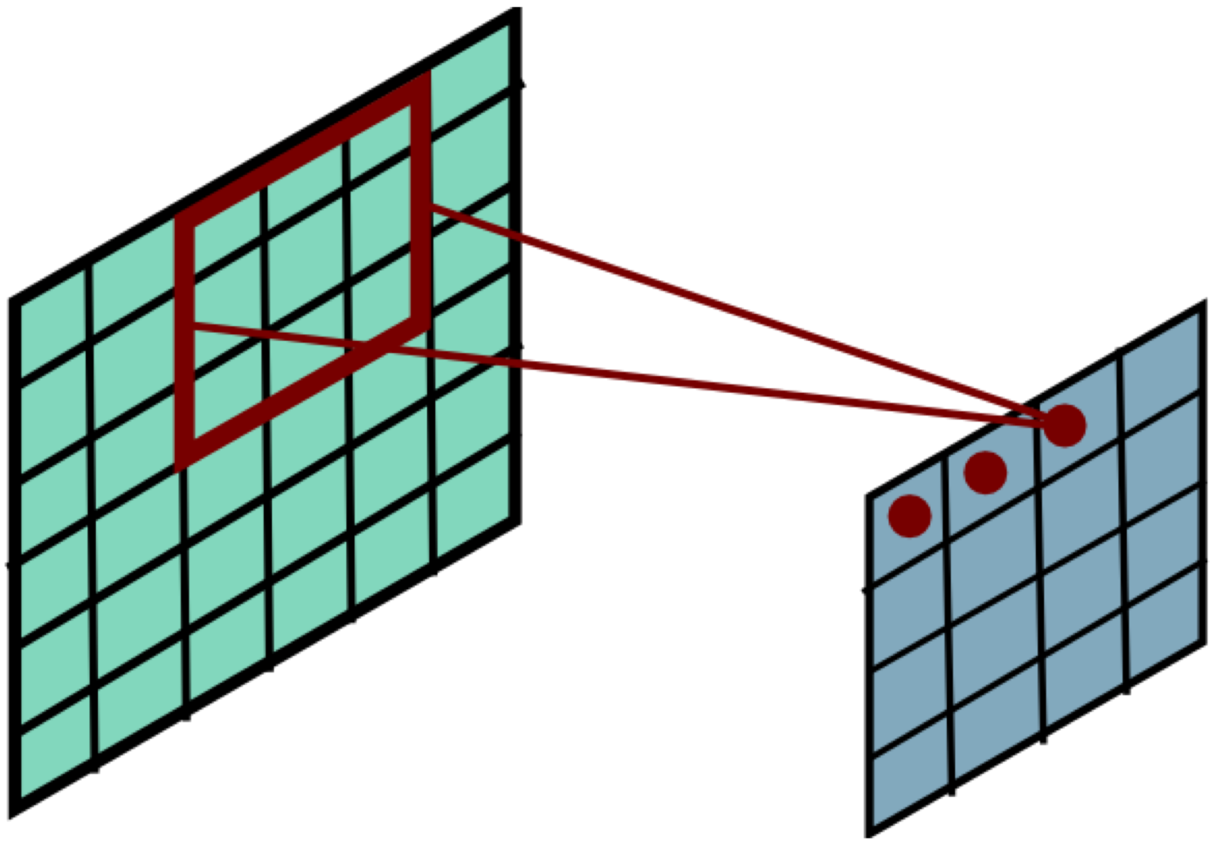


Convolutional Layer

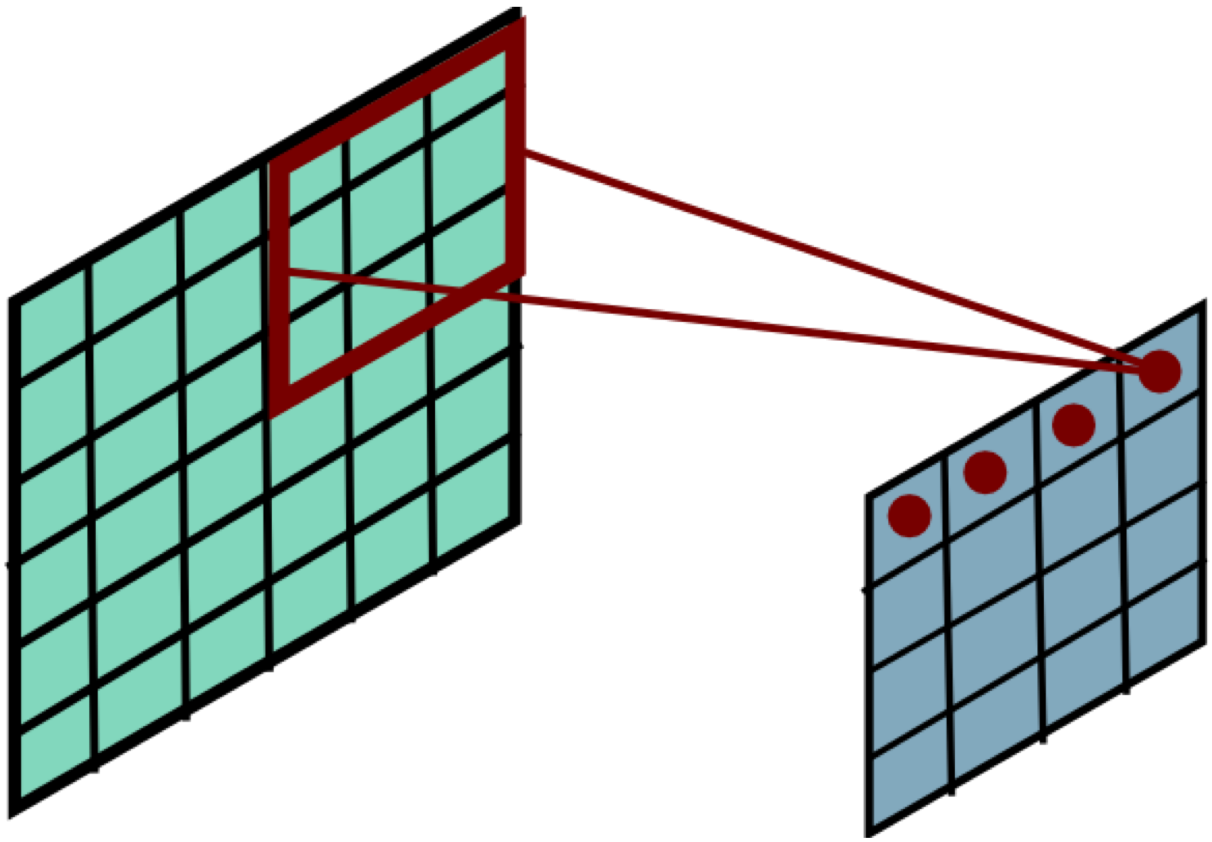


Slide Credit: Marc'Aurelio Ranzato

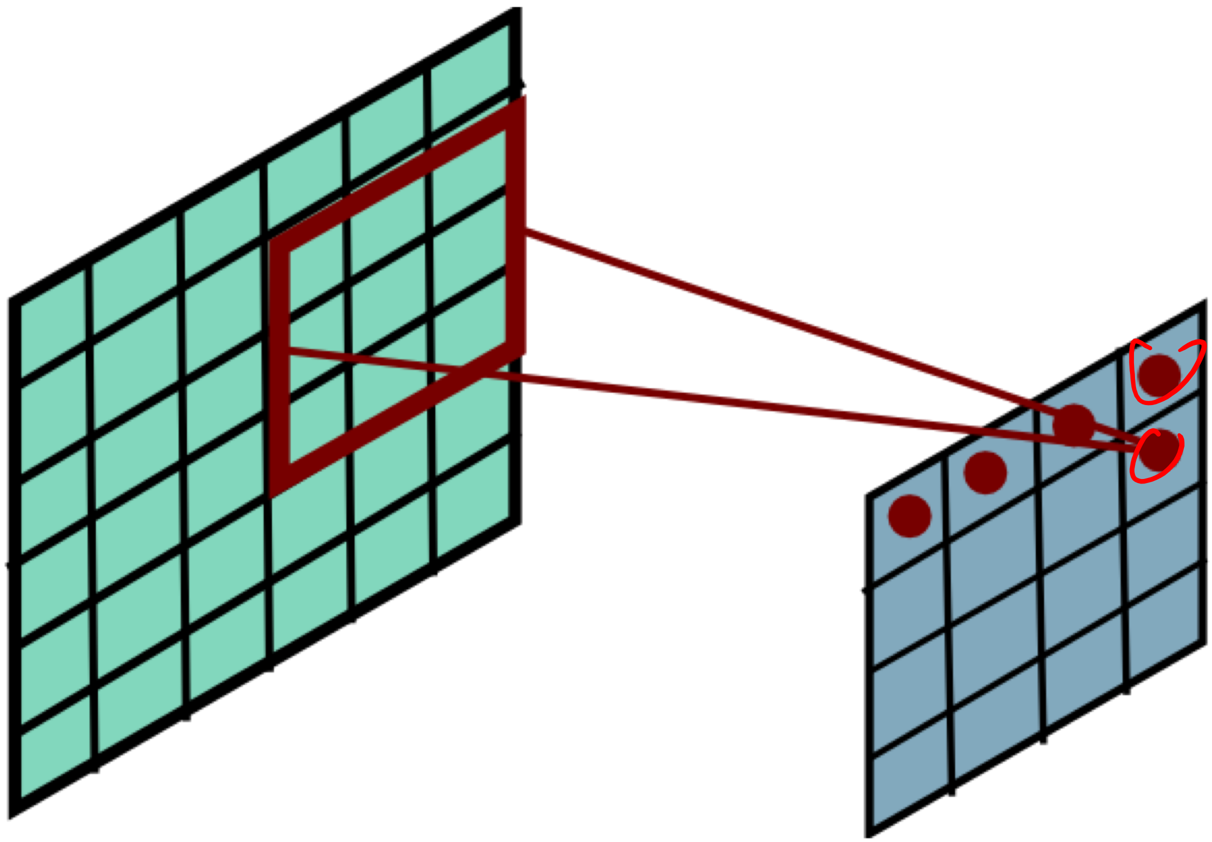
Convolutional Layer



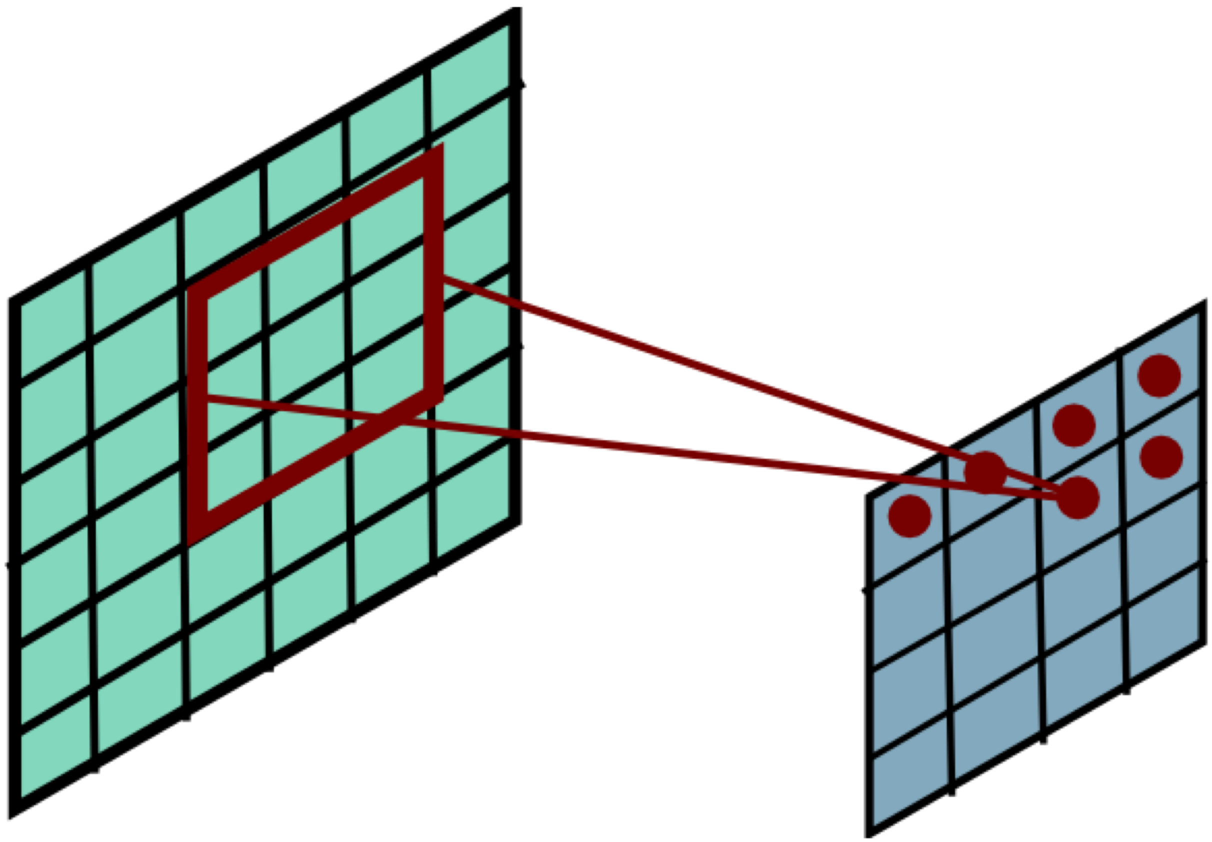
Convolutional Layer



Convolutional Layer

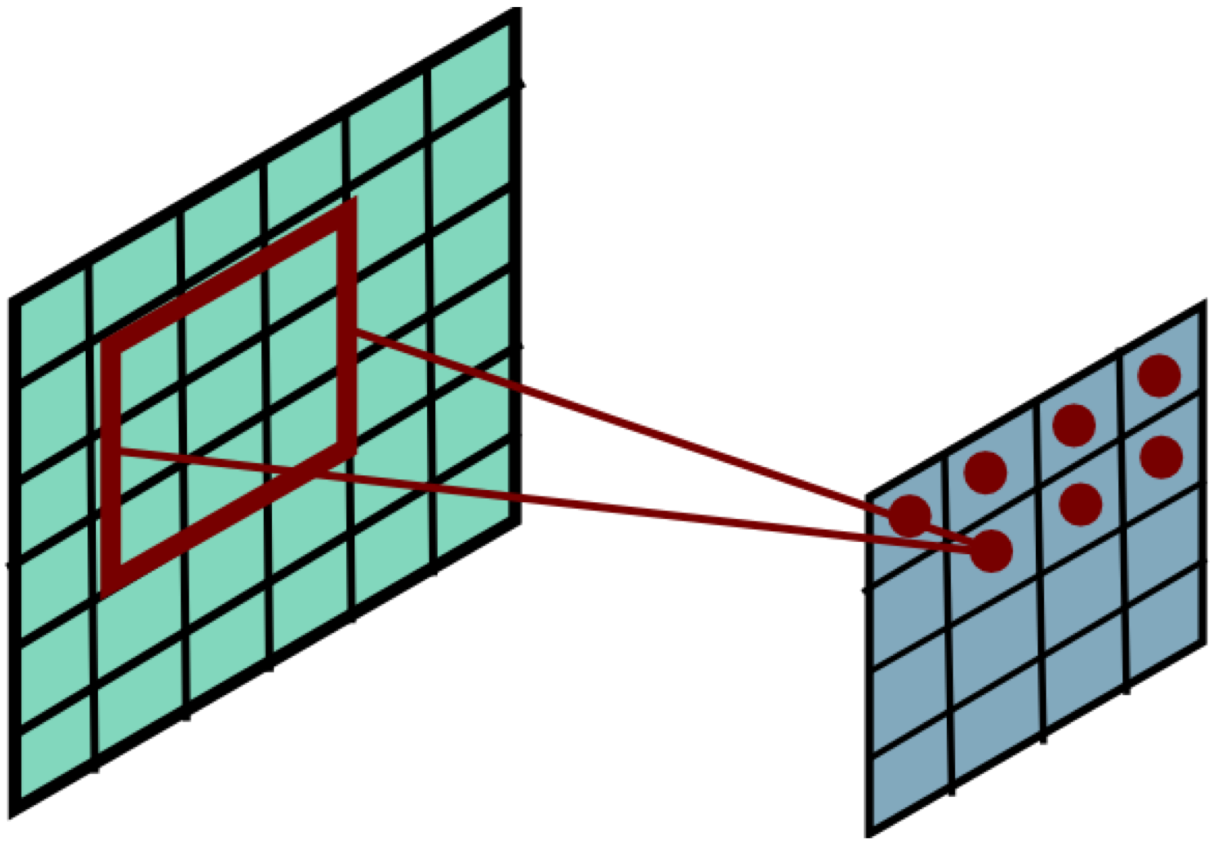


Convolutional Layer

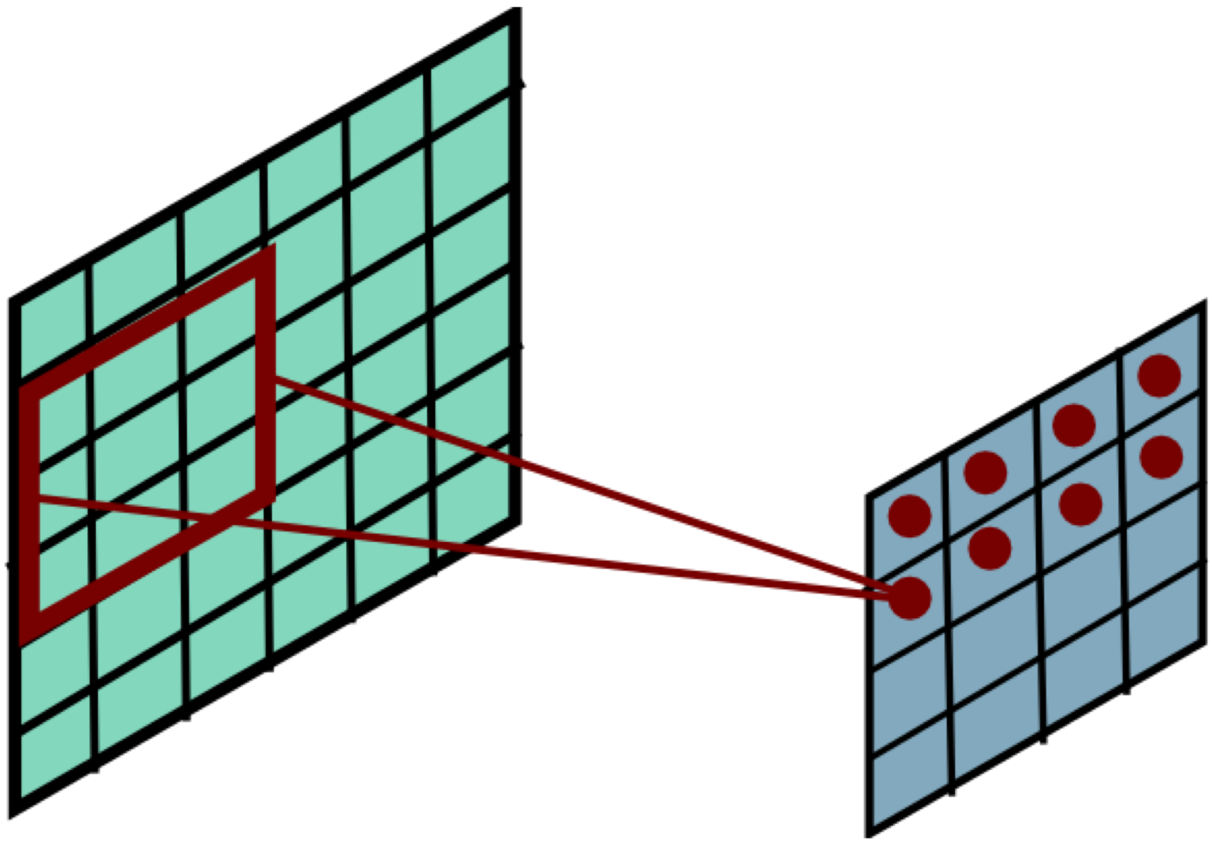


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

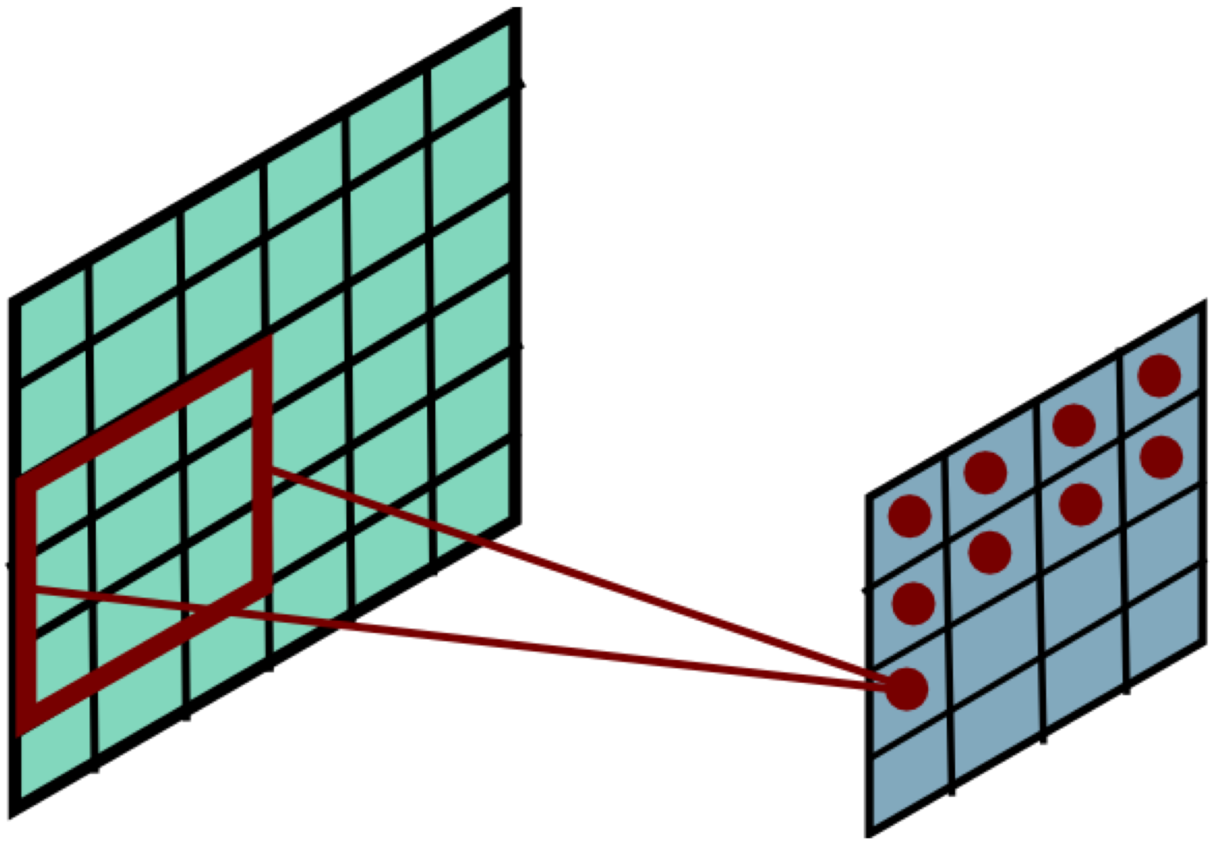


Convolutional Layer



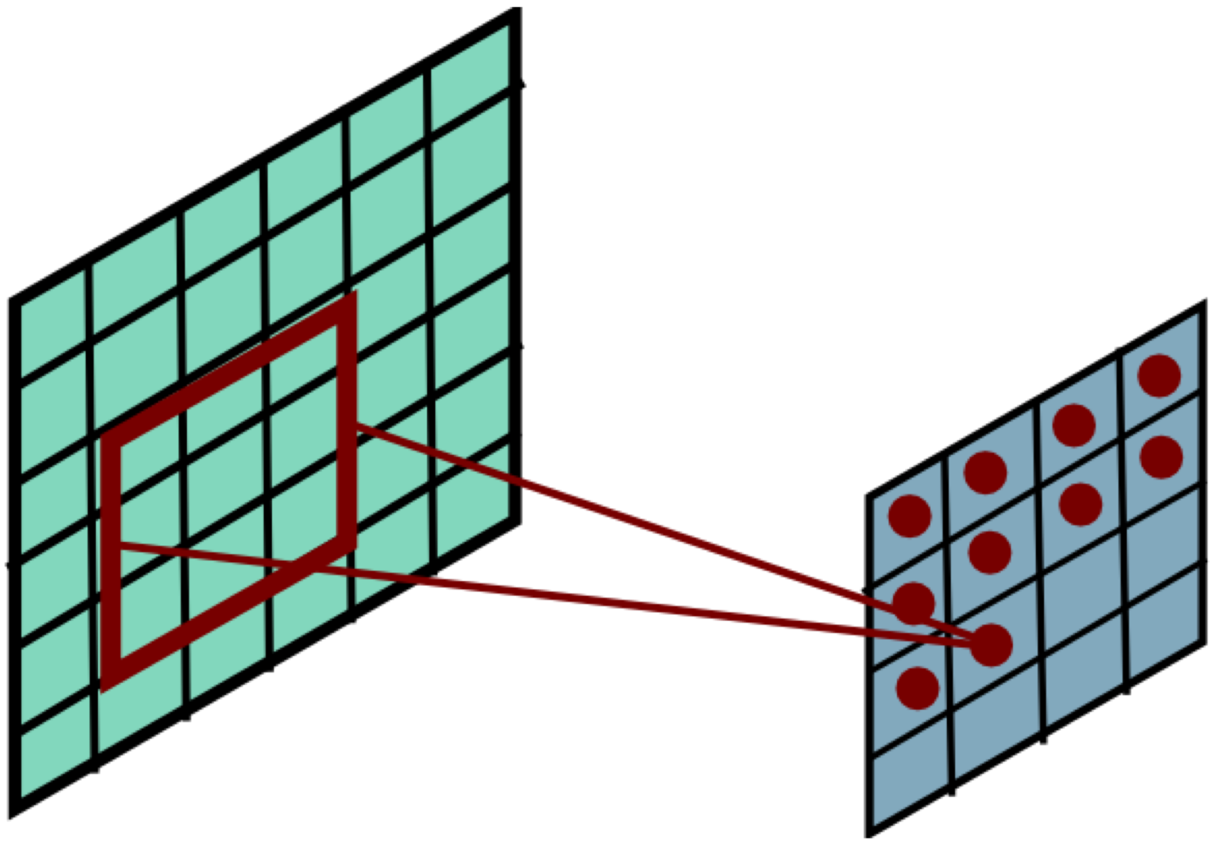
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

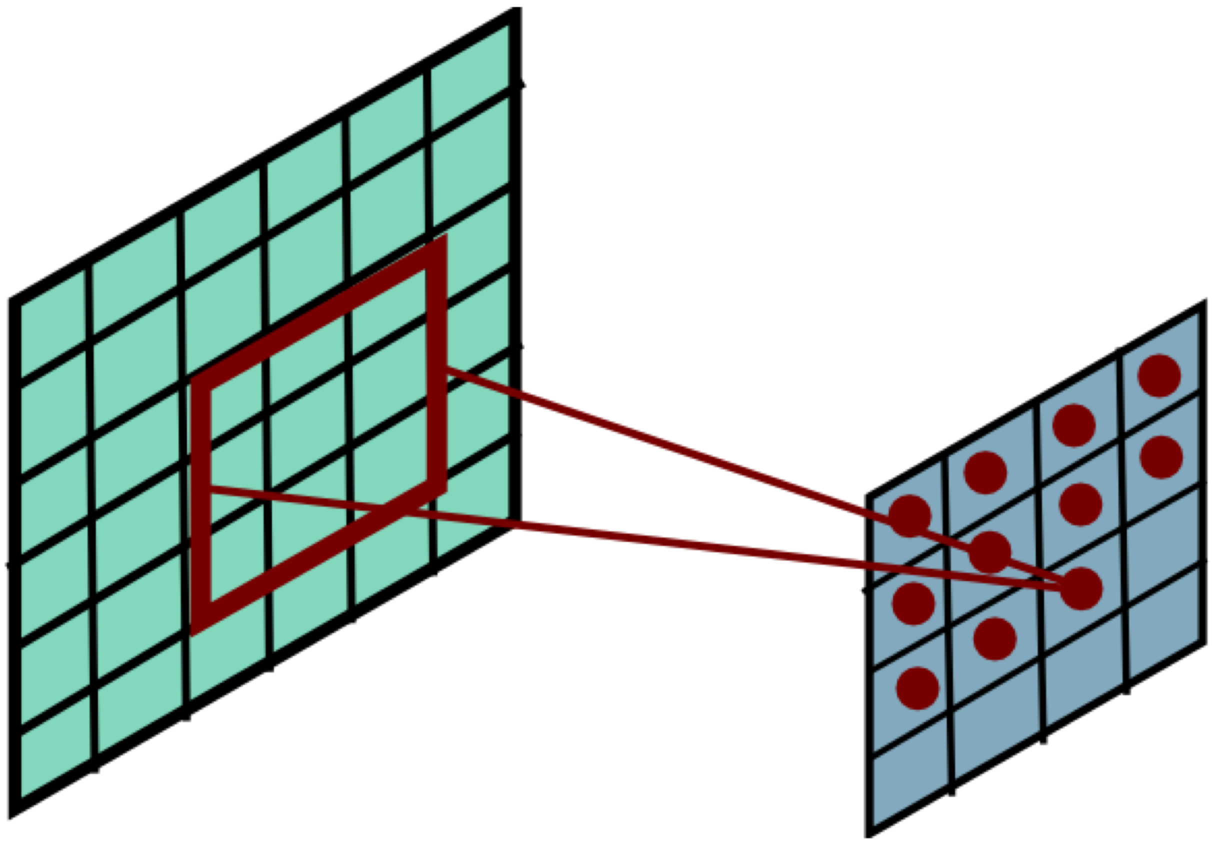


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

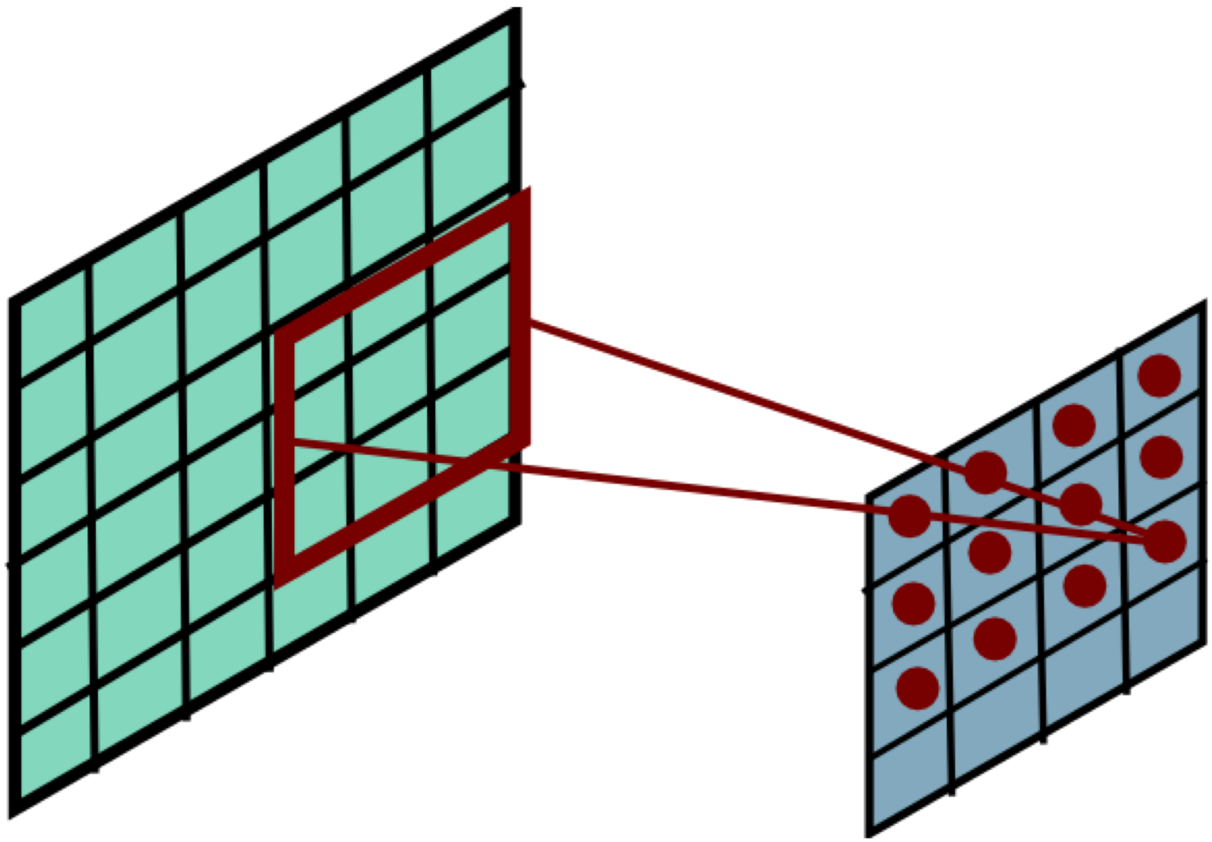


Convolutional Layer

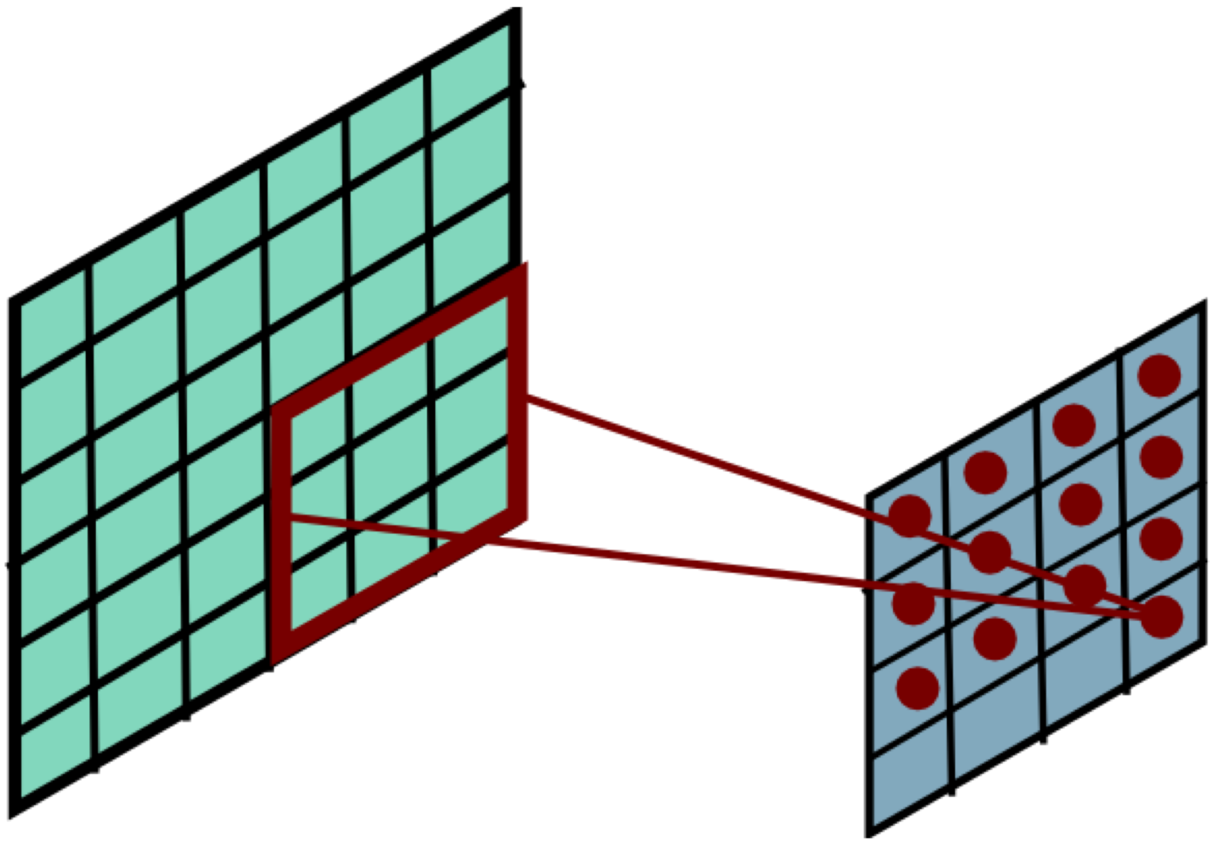


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

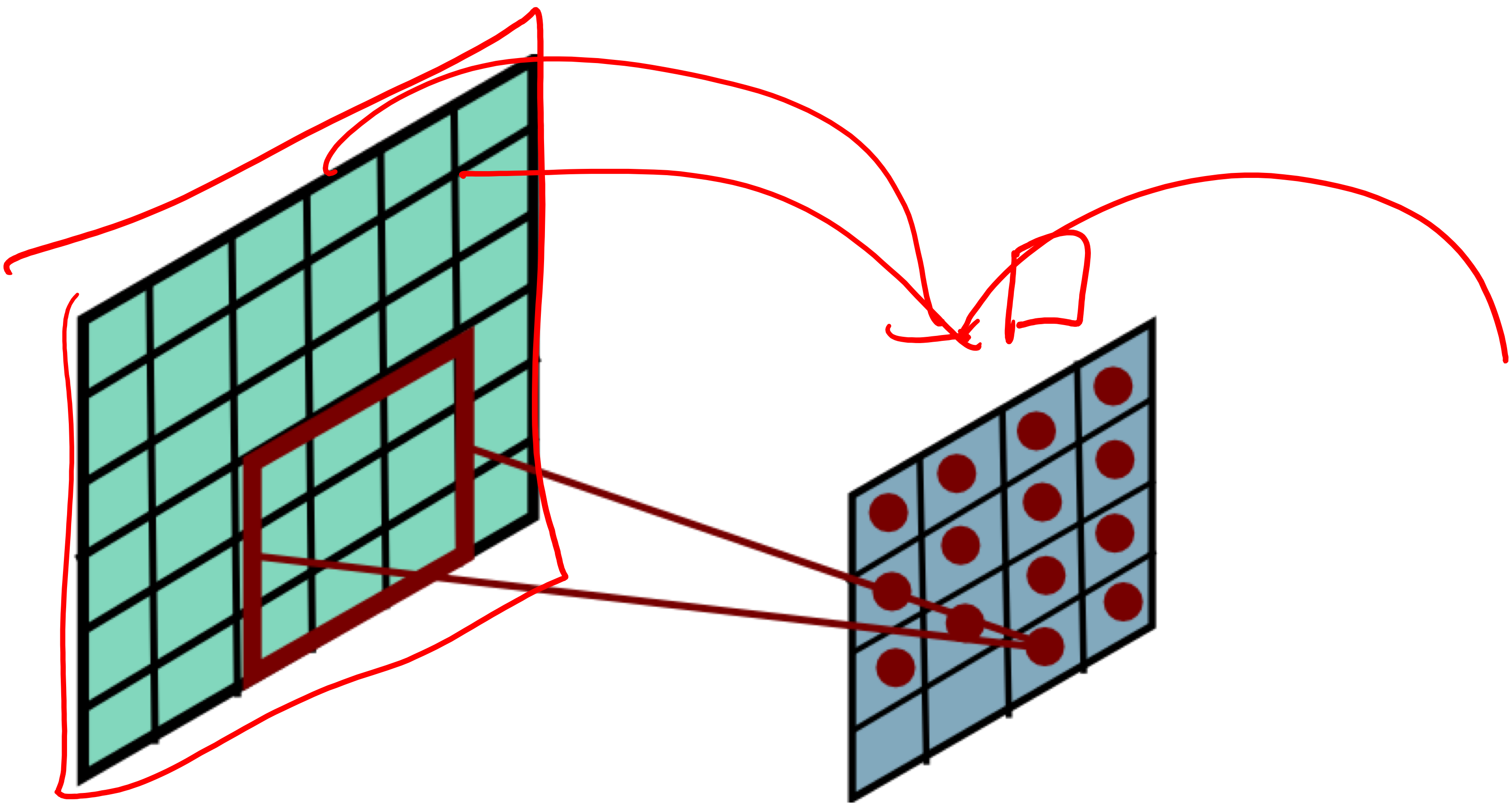


Convolutional Layer

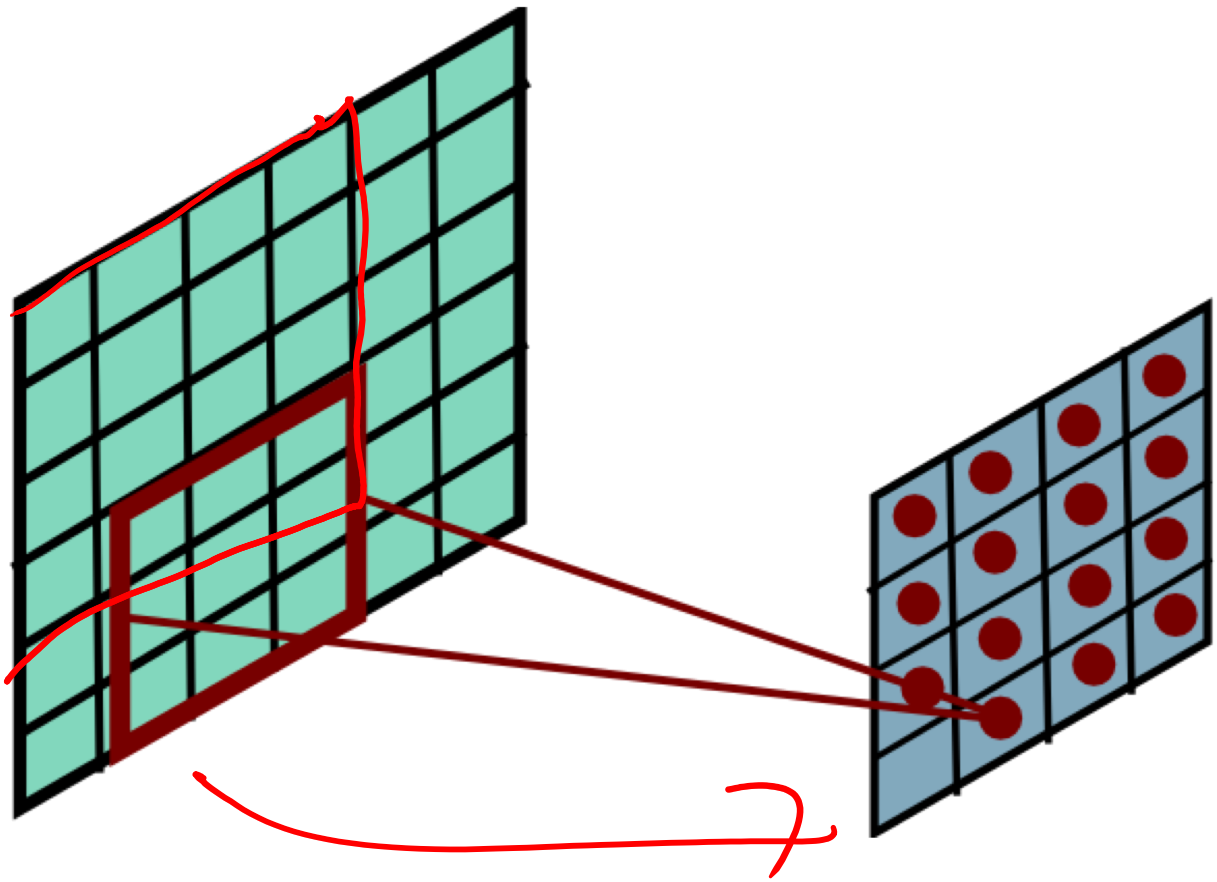


Slide Credit: Marc'Aurelio Ranzato

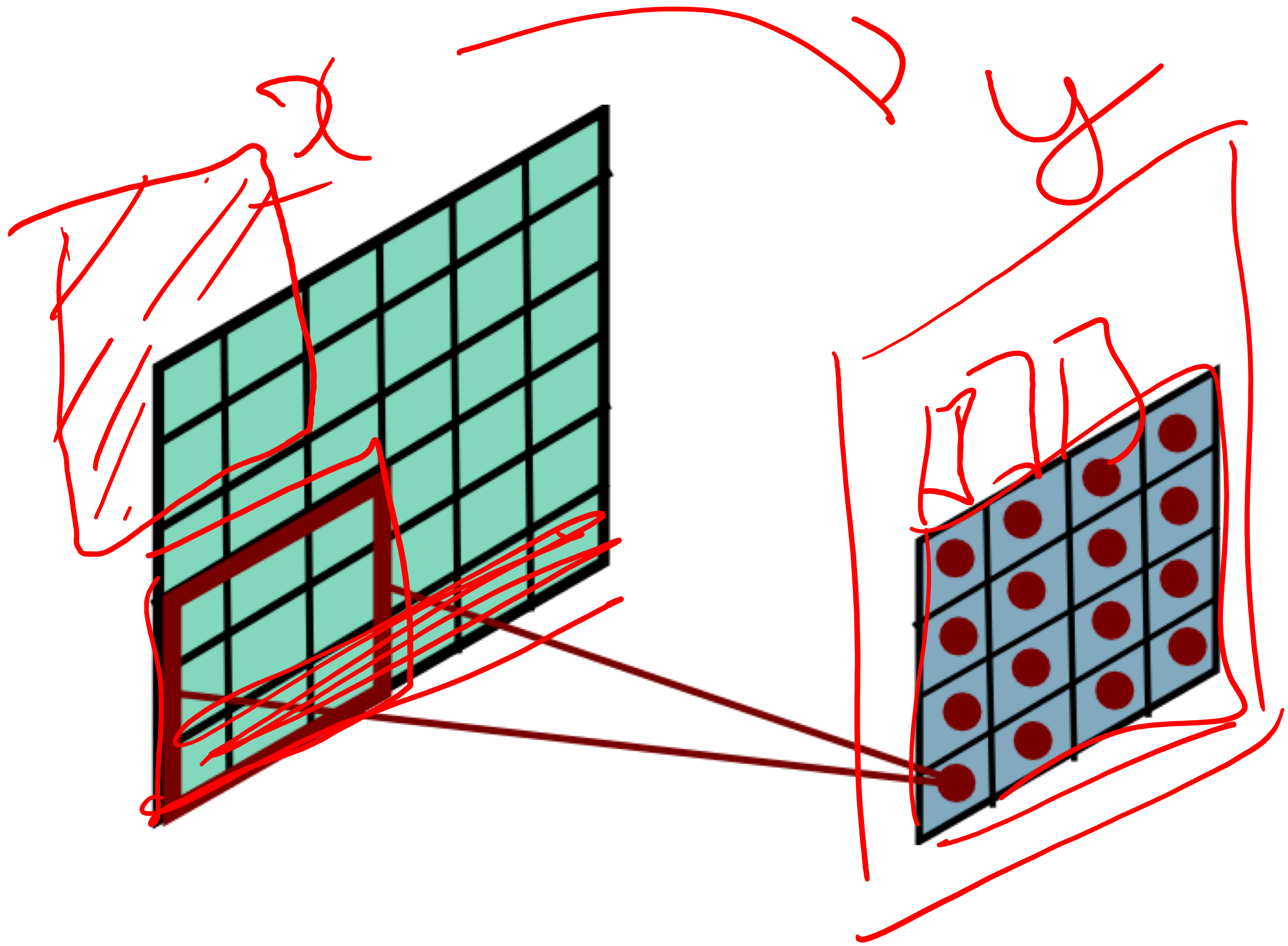
Convolutional Layer



Convolutional Layer



Convolutional Layer



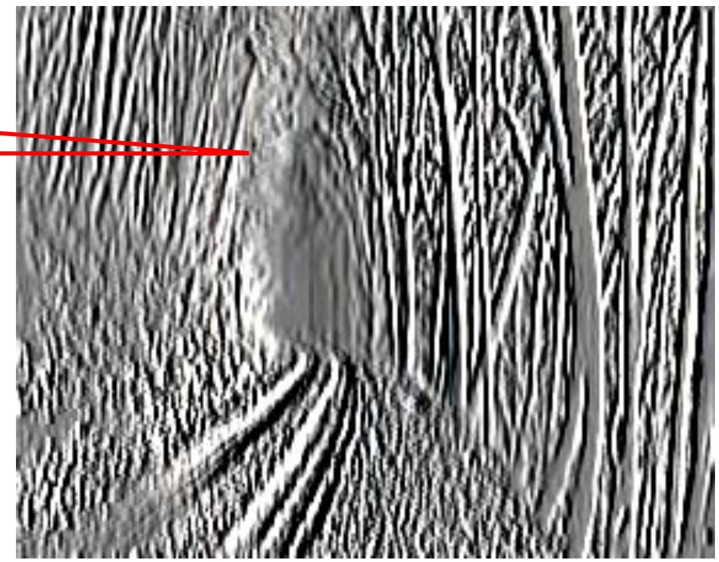
Mathieu et al. "Fast training of CNNs through FFTs" ICLR 2014

Convolution Explained

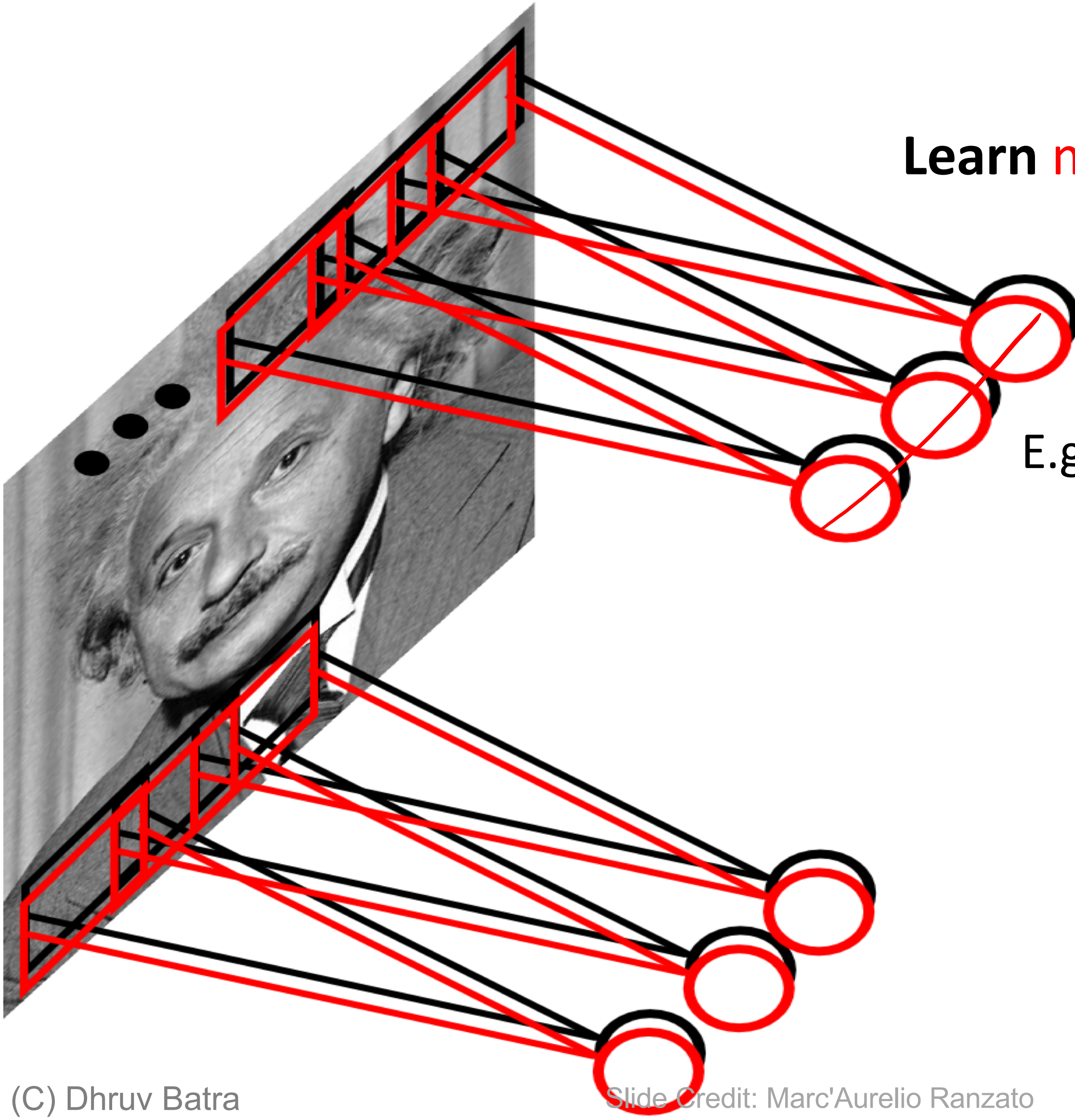
- http://setosa.io/ev/image-kernels/
- https://github.com/bruckner/deepViz

Convolutional Layer

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



Convolutional Layer



Learn multiple filters.

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

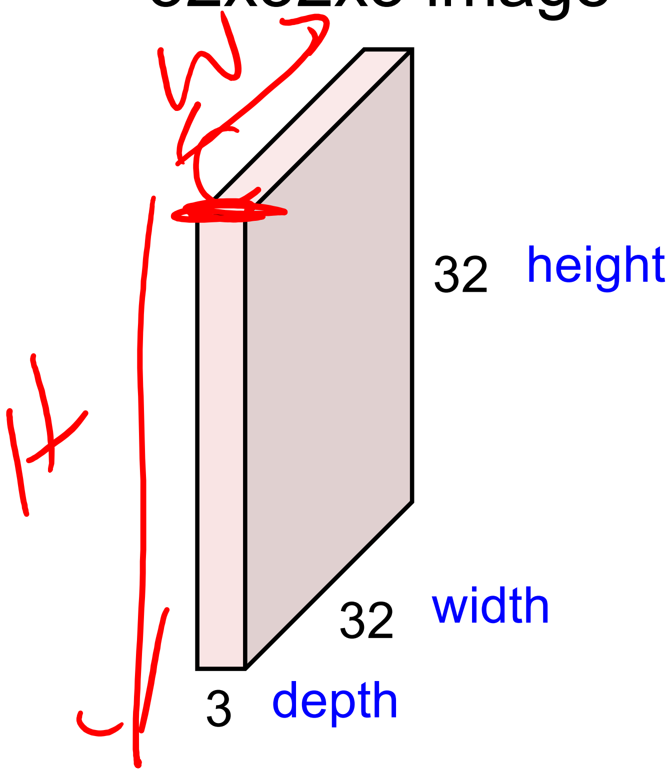
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

Convolutional Layer

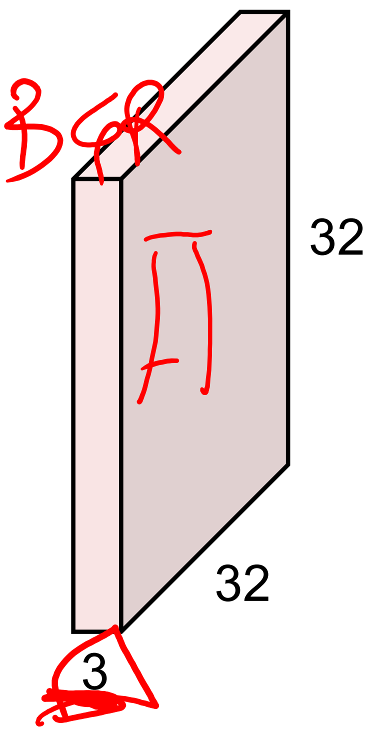
Convolution Layer

32x32x3 image -> preserve spatial structure

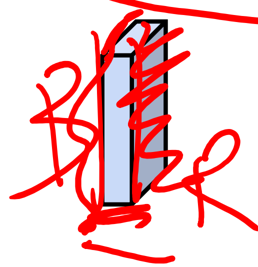


Convolution Layer

32x32x3 image



5x5x3 filter

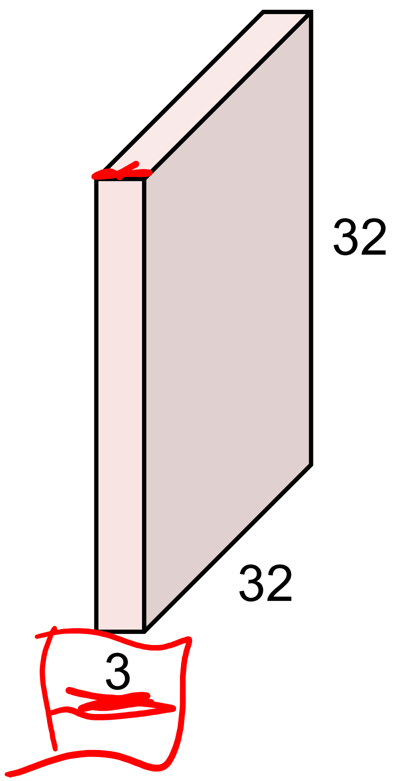


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

$W \in \mathbb{R}^{k_1 \times k_2 \times 3}$

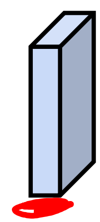
Convolution Layer

32x32x3 image



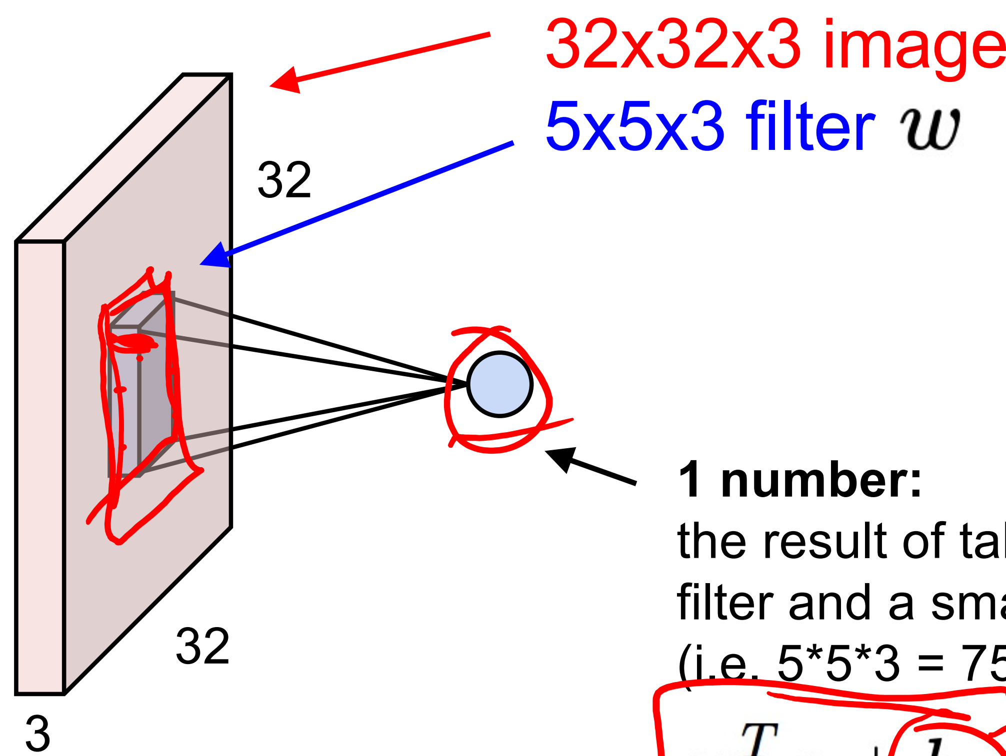
Filters always extend the full depth of the input volume

5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

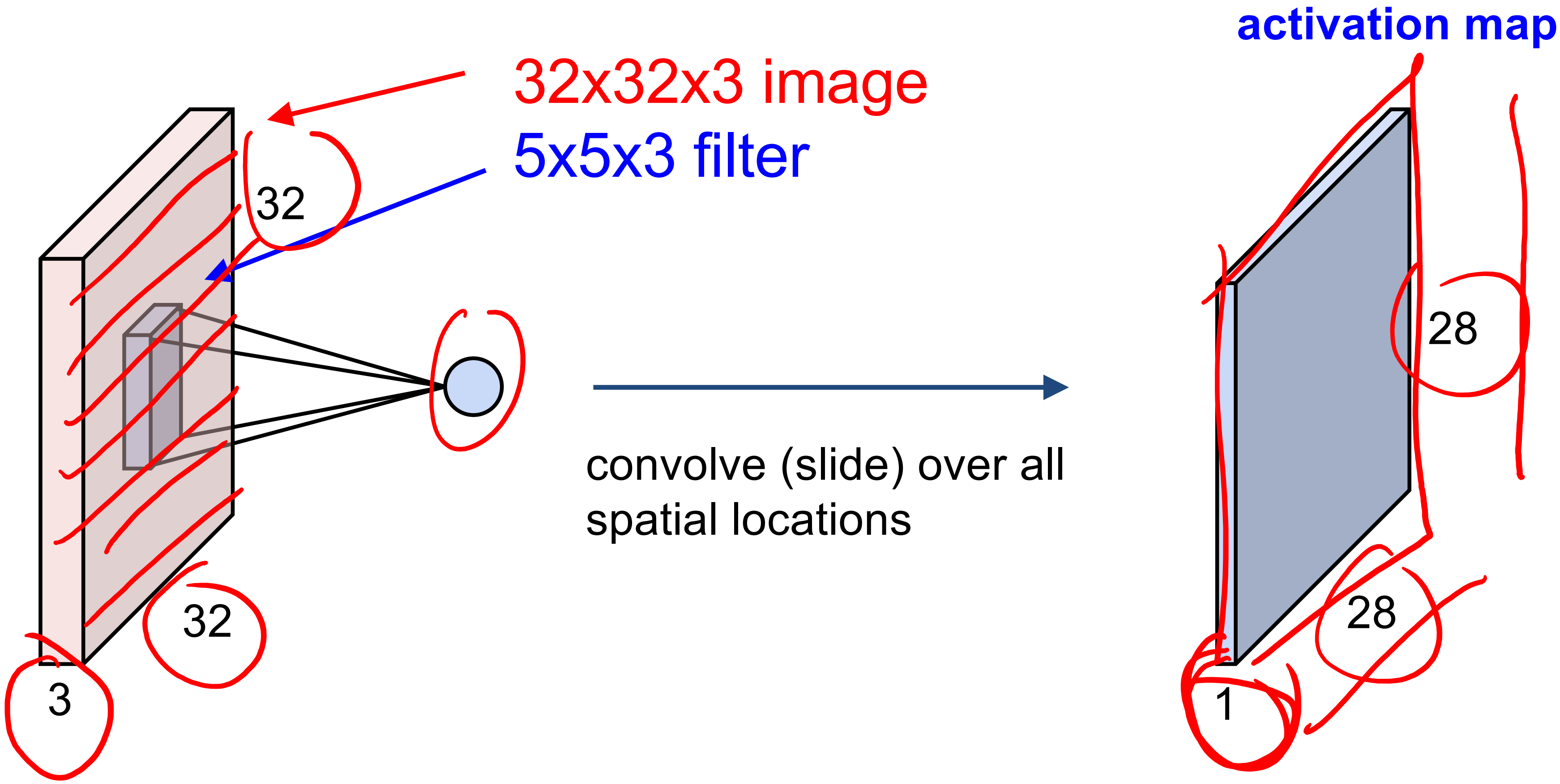
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$\boxed{w^T x + b}$$

Convolution Layer



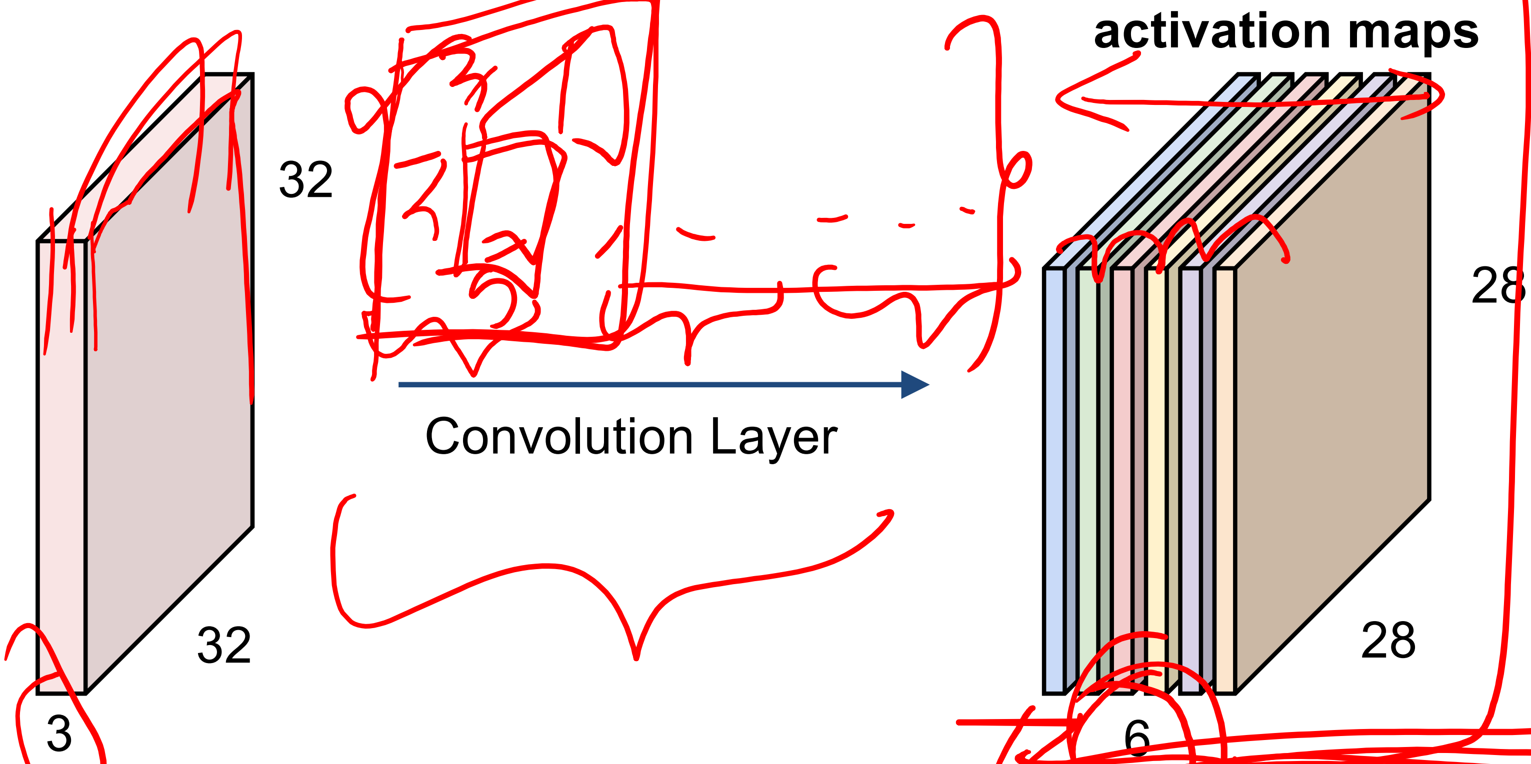
Convolution Layer

consider a second, green filter

10 x {5x5x3}

10

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

x

y

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n