CS 7643: Deep Learning

Topics:

- Computational Graphs
  - Notation + example
- Computing Gradients
  - Forward mode vs Reverse mode AD

Dhruv Batra

Georgia Tech

Administrativia

- HW1 Released
  - Due: 09/22

- PS1 Solutions
  - Coming soon

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
  - Encouraged to apply to your research (computer vision, NLP, robotics,…)
  - Must be done this semester.

- 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

Administrativia

- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1AaXY0JE4lAbHvoDaWlc9zsmfKMyuGS39JAn9dpeXhhQ/edit#gid=0
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names + GT IDs

Recap of last time

How do we compute gradients?

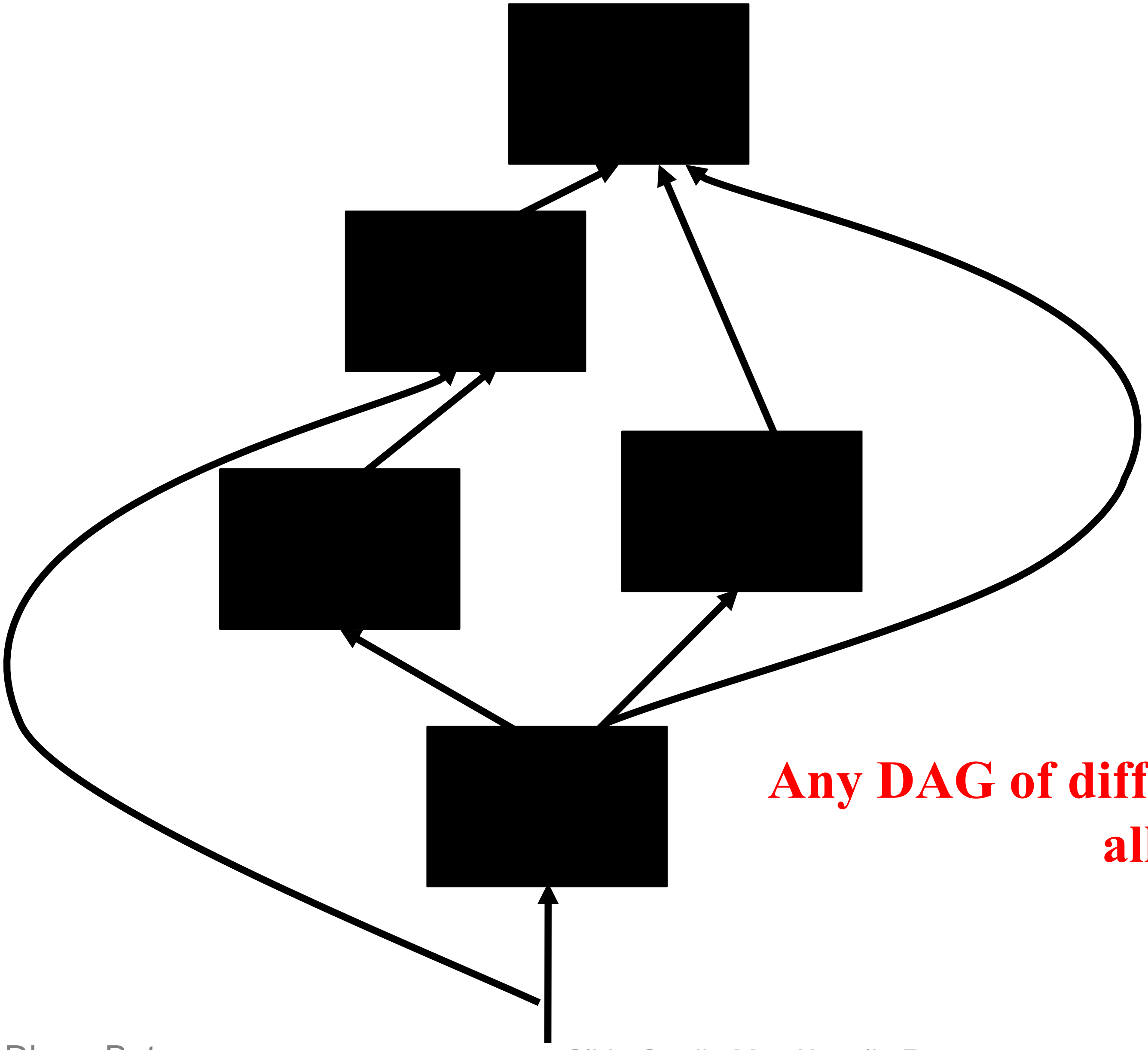
- Manual Differentiation

- Symbolic Differentiation

- Numerical Differentiation

- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

Computational Graph



Any DAG of differentiable modules is allowed!

Slide Credit: Marc'Aurelio Ranzato

Directed Acyclic Graphs (DAGs)

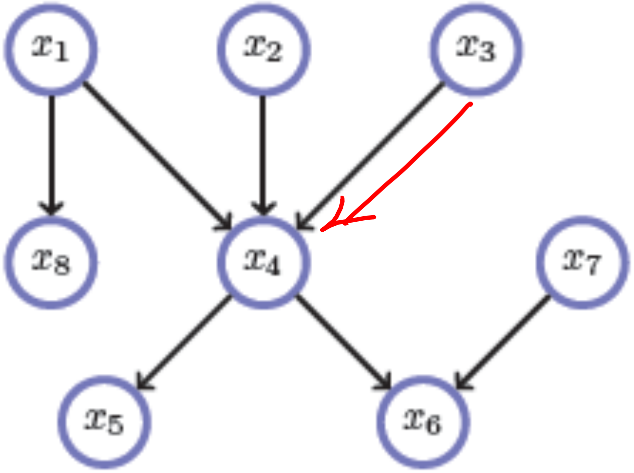
- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

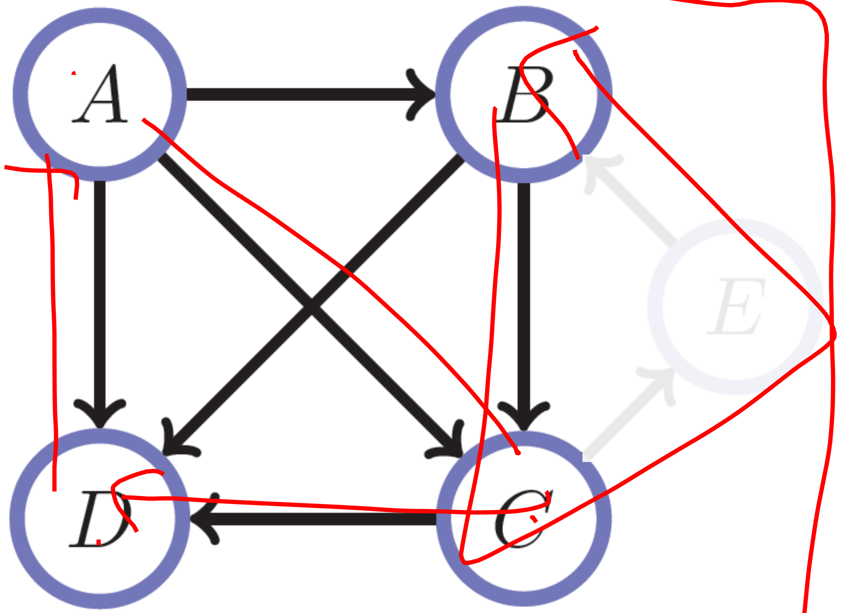
$E = \{(v_i, v_j)\}$

$\{\{v_i, v_j\}\}$

$(v_i, v_k \ldots v_i)$

$(D, A) \notin E$



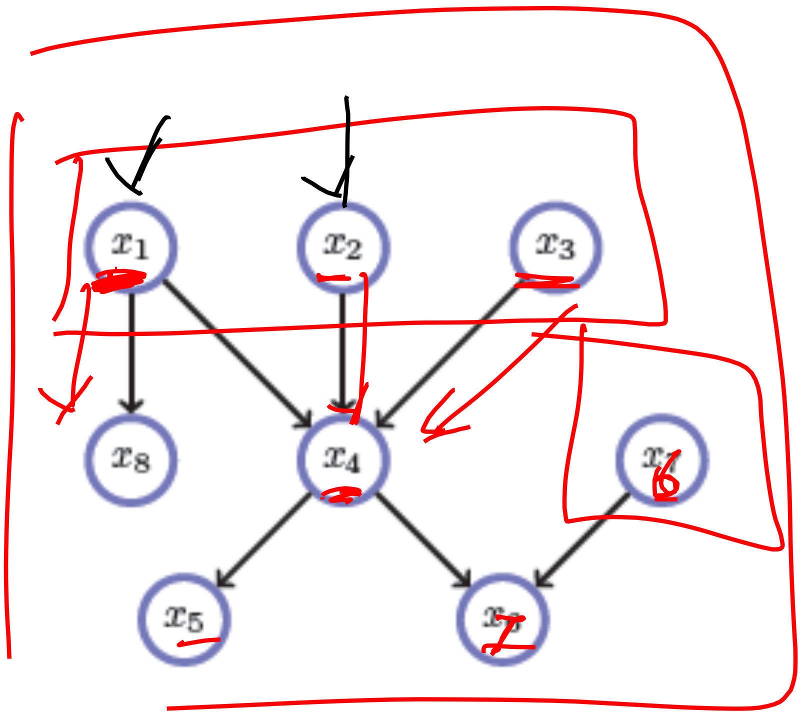


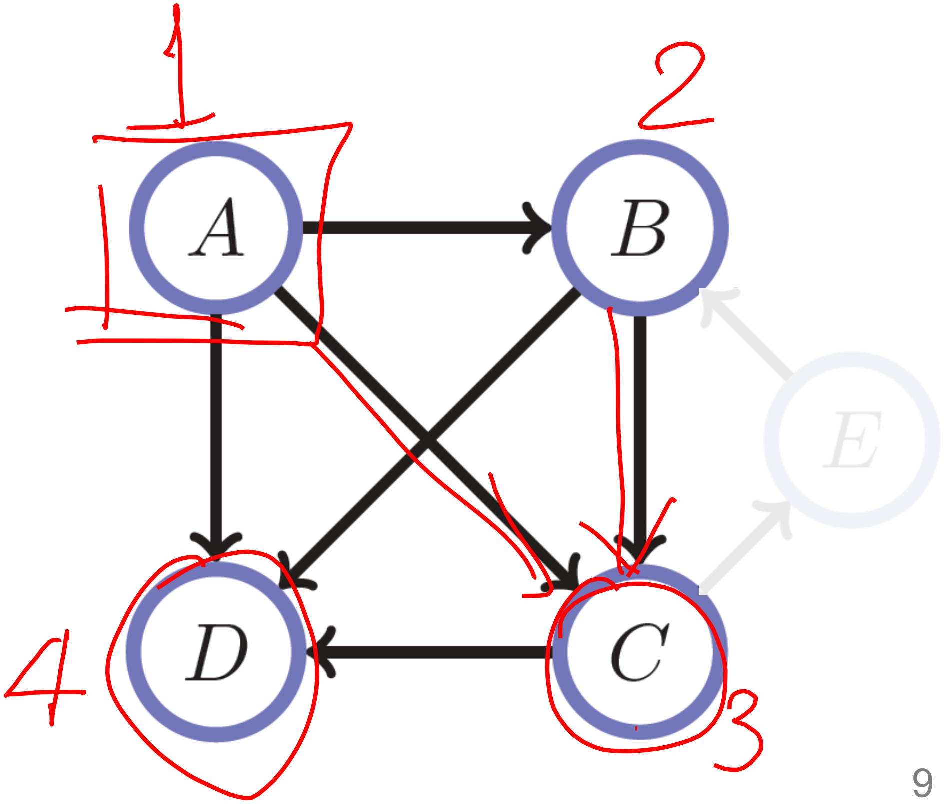
Directed Acyclic Graphs (DAGs)

- Concept
  - Topological Ordering

$\exists\, \sigma : V \rightarrow [n] = \{1, \ldots, n\}$

s.t. $(v_i, v_j) \in E \Rightarrow \boxed{\sigma(v_i) < \sigma(v_j)}$

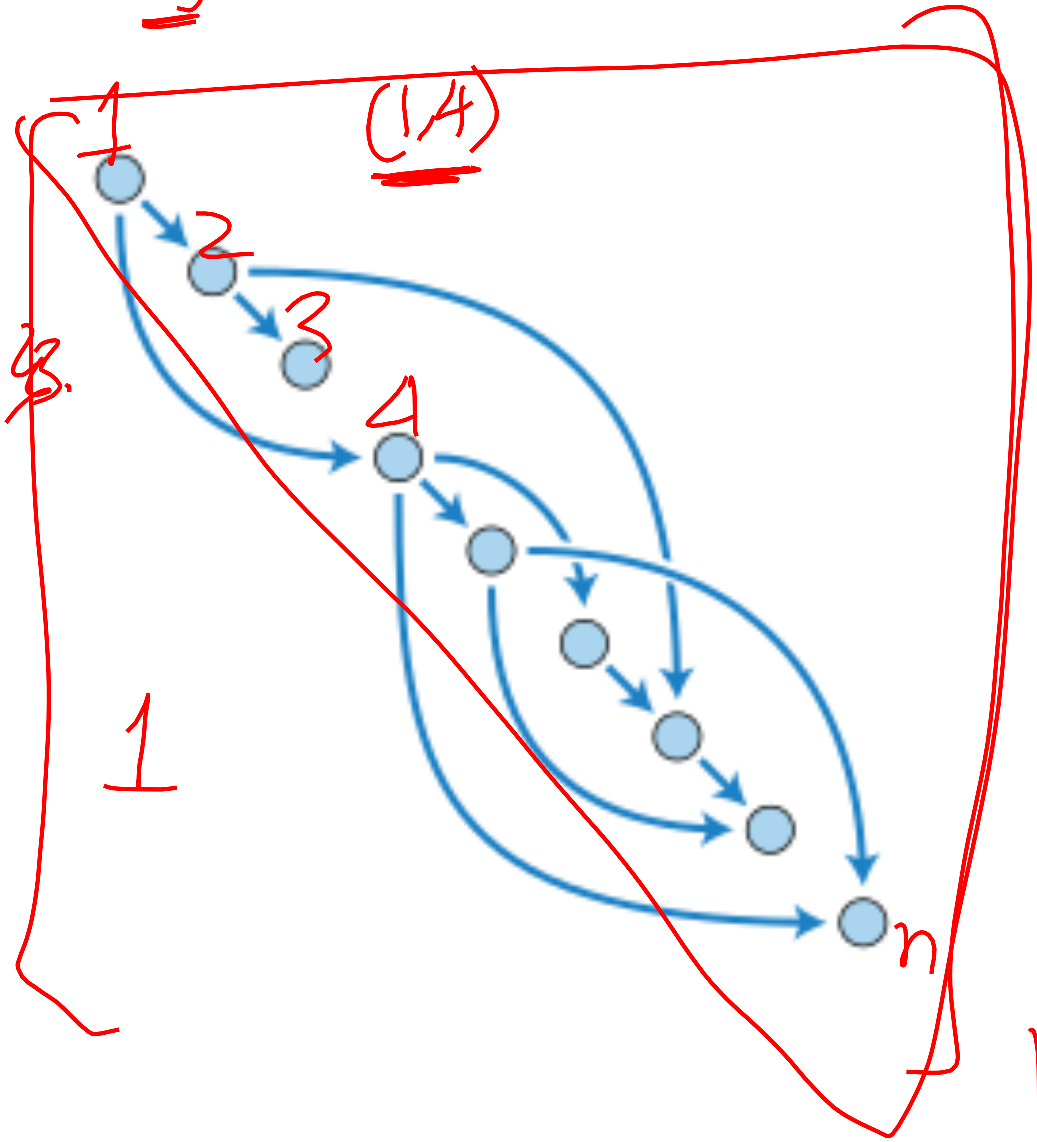




Directed Acyclic Graphs (DAGs)

$a_{ij} = 1 \quad (i,j) \in E$

$A =$

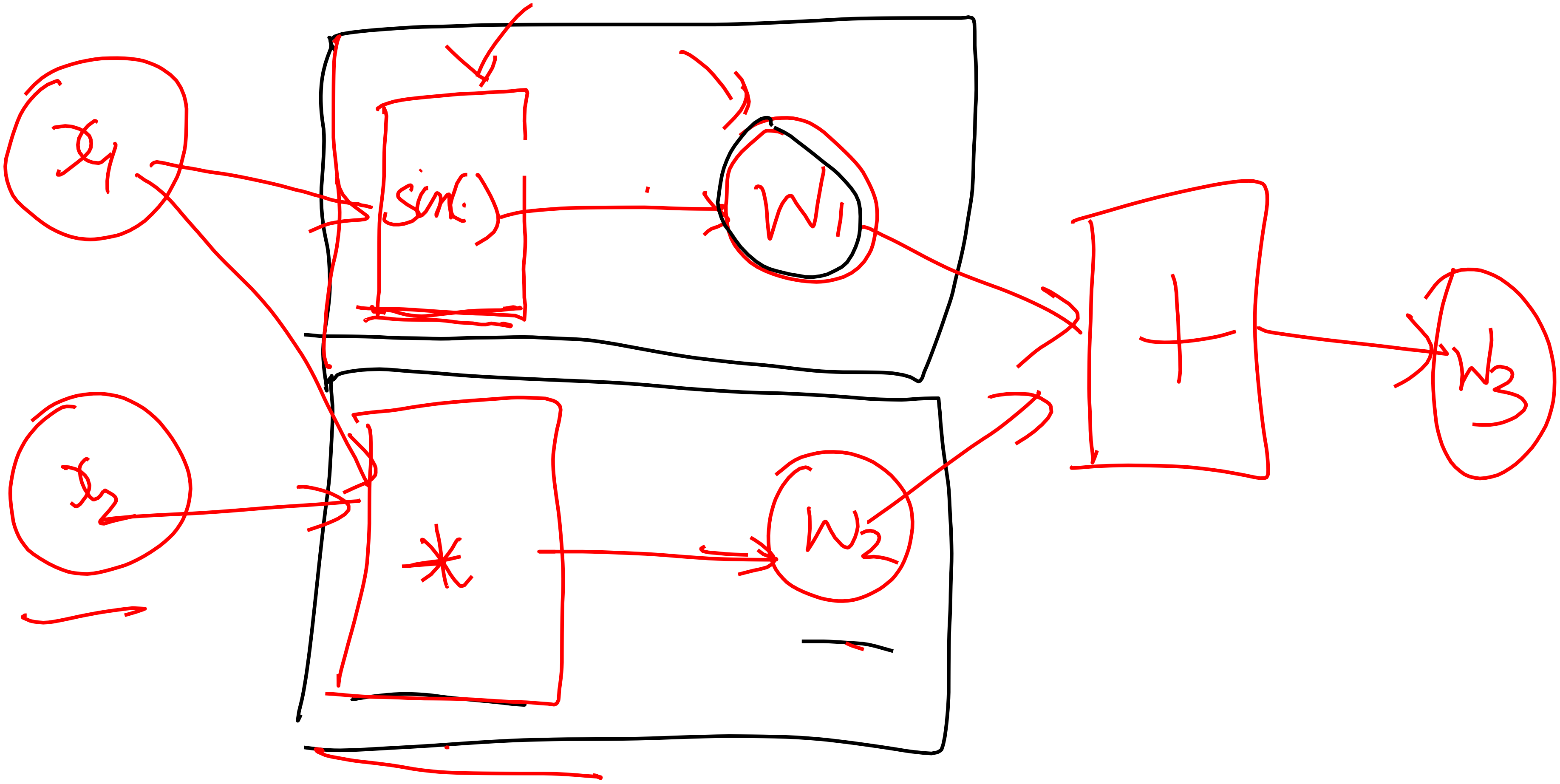


$n \times n$

Computational Graphs

- Notation #1

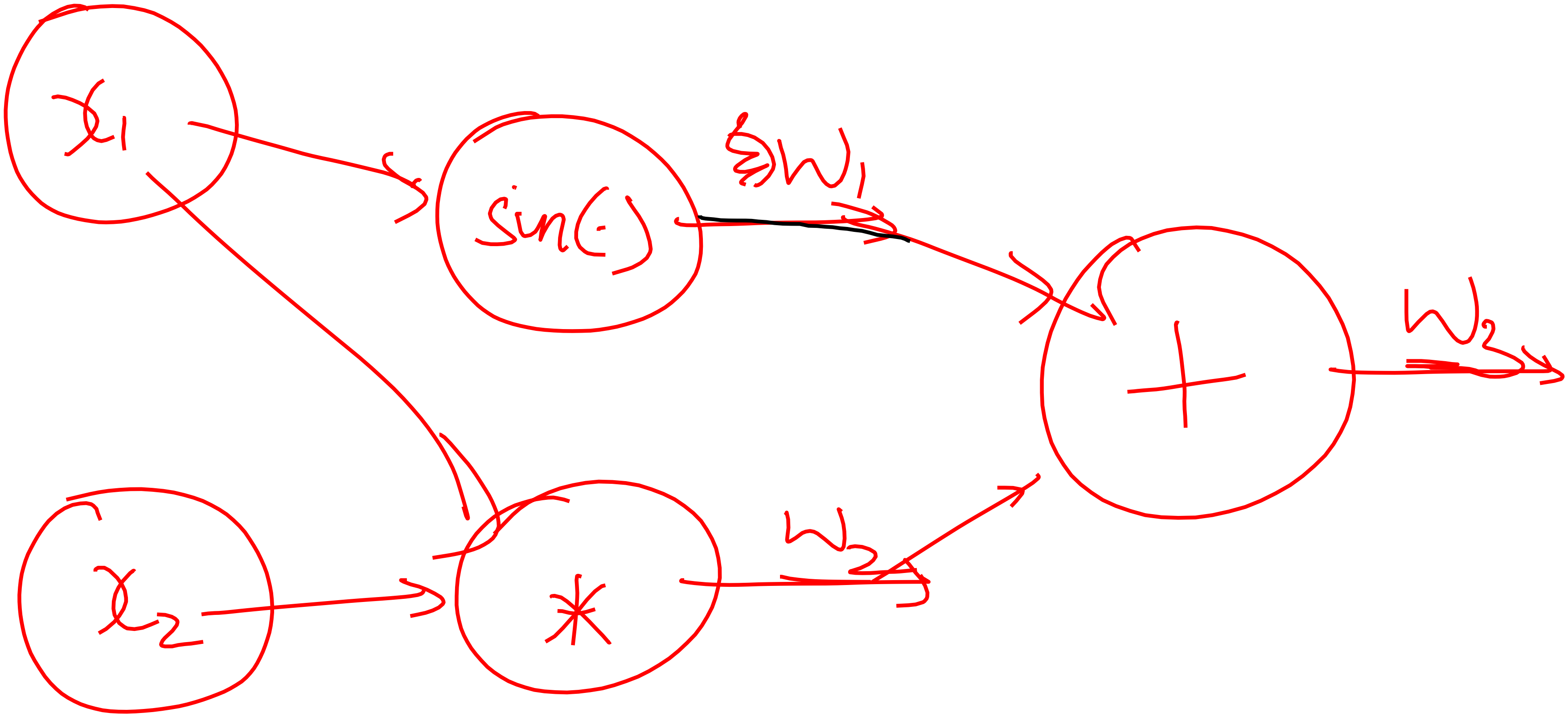
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Computational Graphs

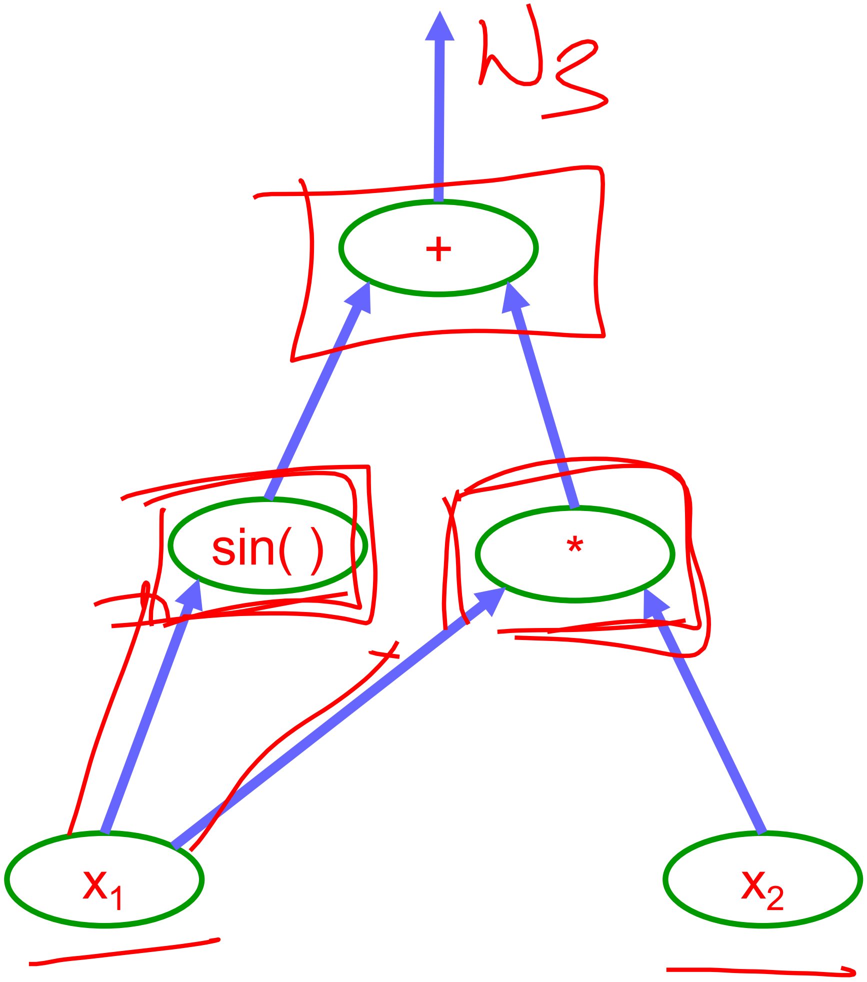
- Notation #2

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



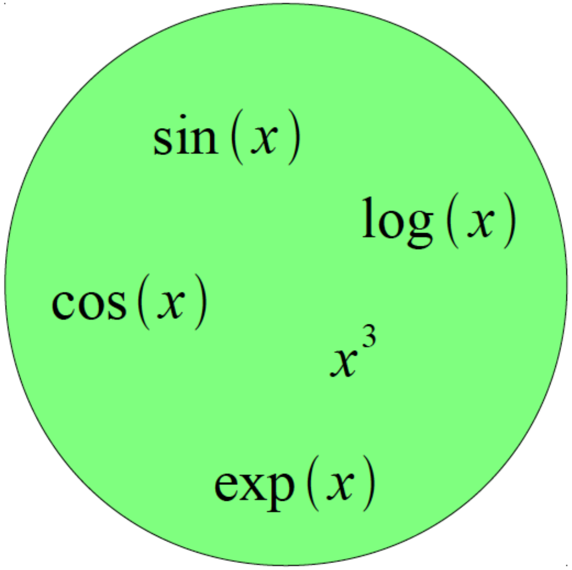
Example

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



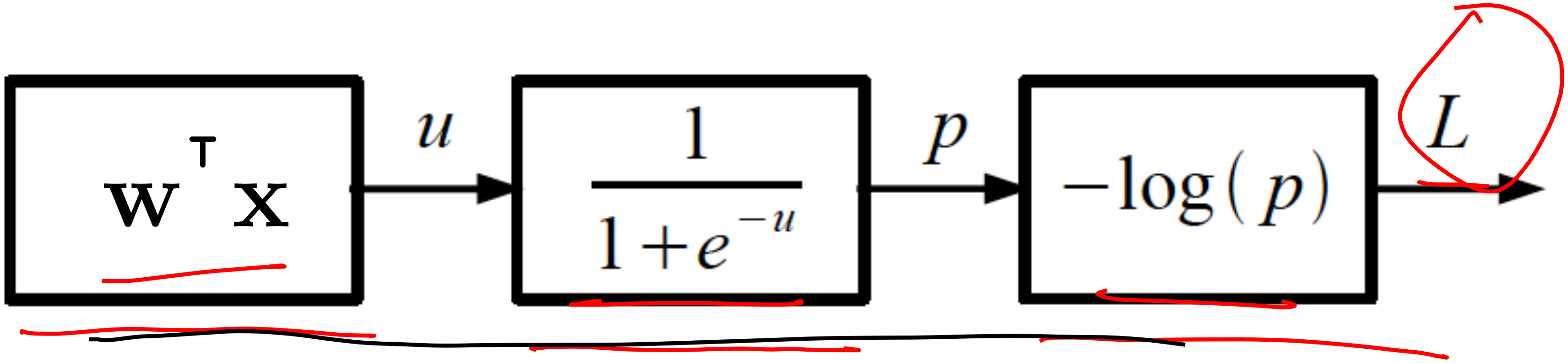
Logistic Regression as a Cascade

Given a library of simple functions



Compose into a complicate function

$$-\log\left(\frac{1}{1+e^{-\mathbf{w}^\intercal \mathbf{x}}}\right)$$



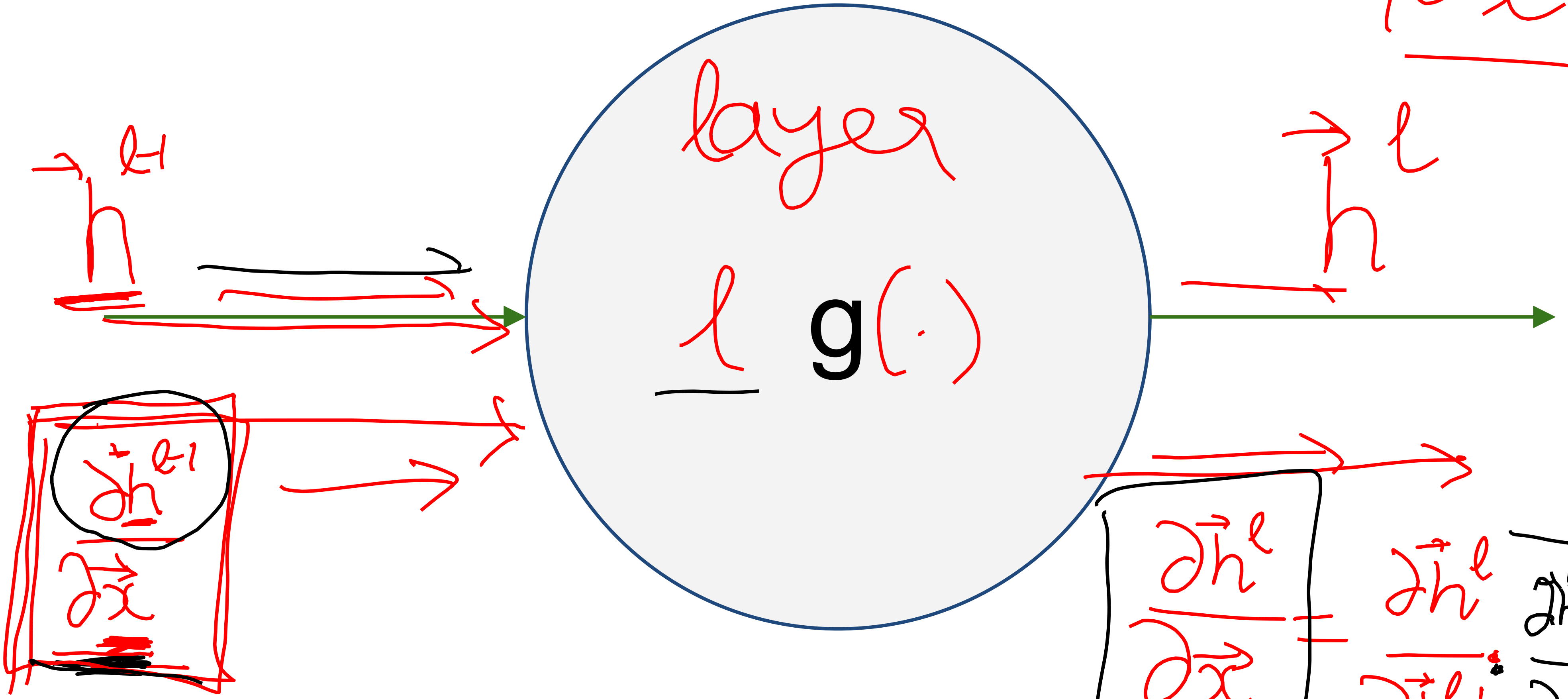
Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Forward mode vs Reverse Mode

- Key Computations

Forward mode AD

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

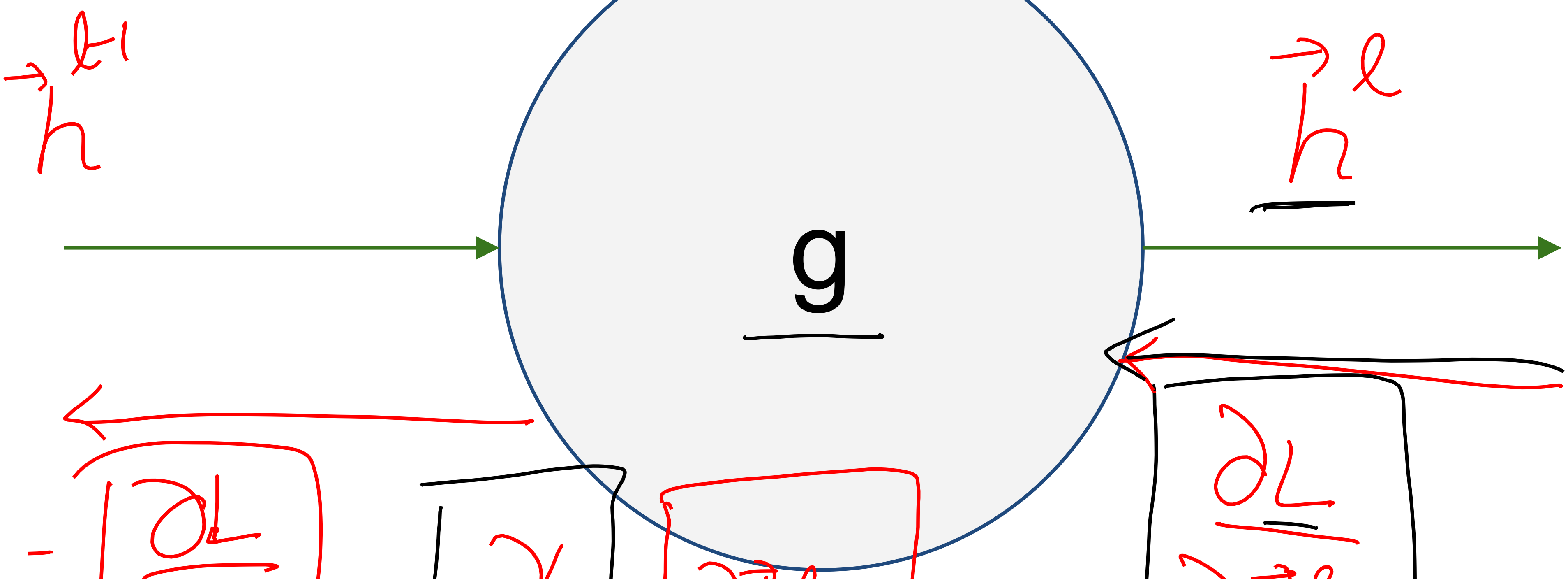


$$\boxed{\frac{\partial \vec{h}^{l}}{\partial \vec{x}}} = \underbrace{\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}} \cdot \frac{\partial \vec{h}^{l-1}}{\partial \vec{x}}}_{\text{Jacobian}}$$

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

Reverse mode AD

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



$$\boxed{\frac{\partial L}{\partial \vec{h}^{l-1}}} = \underbrace{\boxed{\frac{\partial L}{\partial \vec{h}^{l}}}}_{\text{input}} \underbrace{\boxed{\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}}}_{\text{Jacobian}} \qquad \boxed{\frac{\partial L}{\partial \vec{h}^{l}}}$$

Example: Forward mode AD

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

$$\begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

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

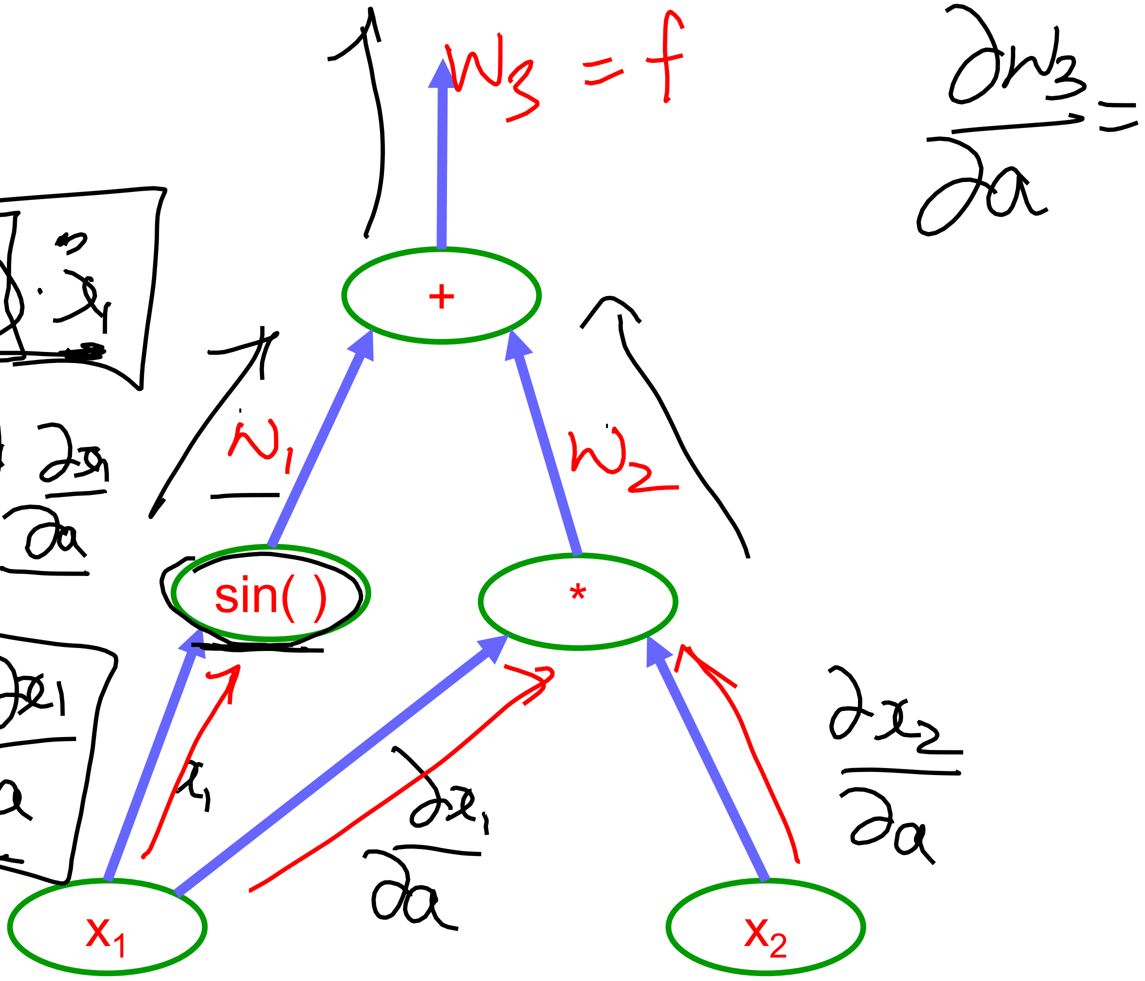
$a = x_1$

$a = x_2$

$$\boxed{\cos(x_1) \cdot \dot{x}_1}$$

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

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



$w_3 = f$
+
$\dot{w}_1$
$\dot{w}_2$
sin( )
*
x_1
$\frac{\partial x_1}{\partial a}$
$\frac{\partial x_2}{\partial a}$
x1
x2

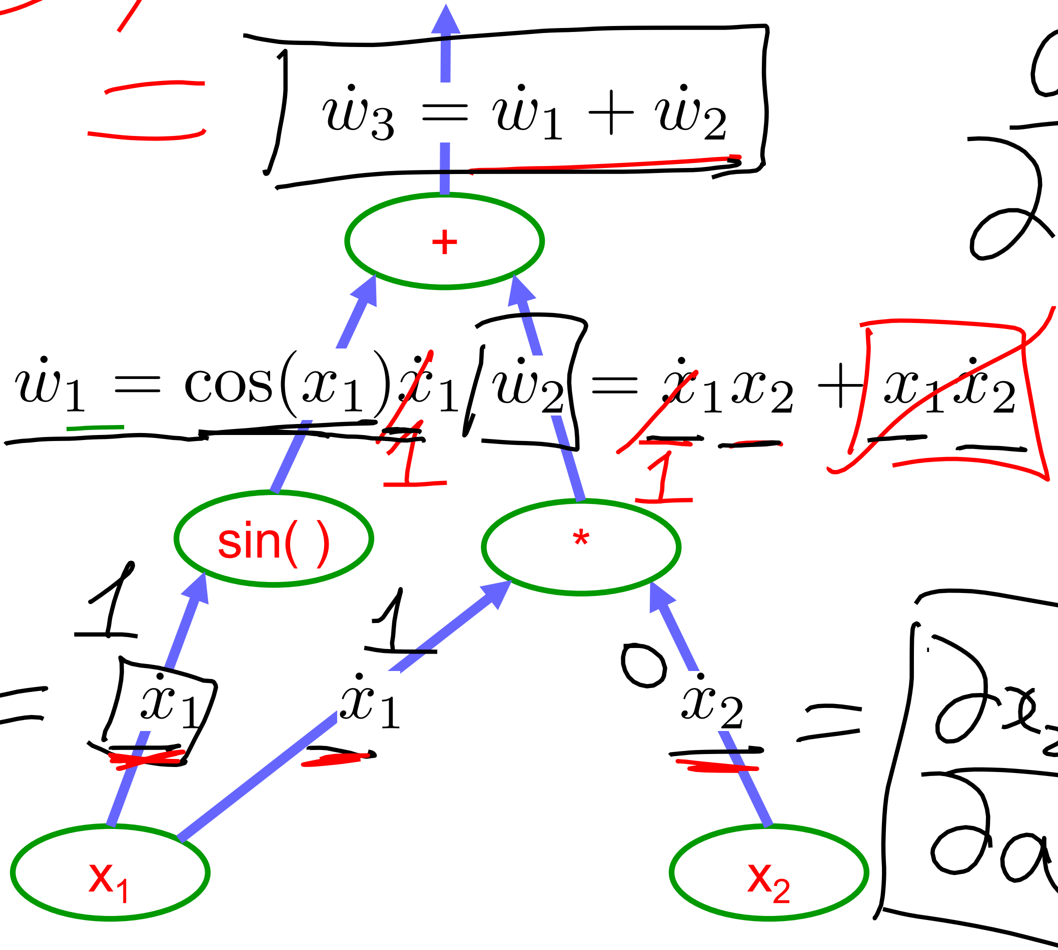
$$\frac{\partial w_3}{\partial a} =$$

Example: Forward mode AD

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

$\cos(x_1) + x_2 =$

$$\frac{\partial f}{\partial a} = x_1$$



$$\boxed{\dot{w}_3 = \dot{w}_1 + \dot{w}_2}$$

$\dot{w}_1 = \cos(x_1)\overset{1}{\cancel{\dot{x}_1}}$ / $\dot{w}_2 = \overset{1}{\cancel{\dot{x}_1}} x_2 + \cancel{\boxed{x_1 \dot{x}_2}}$

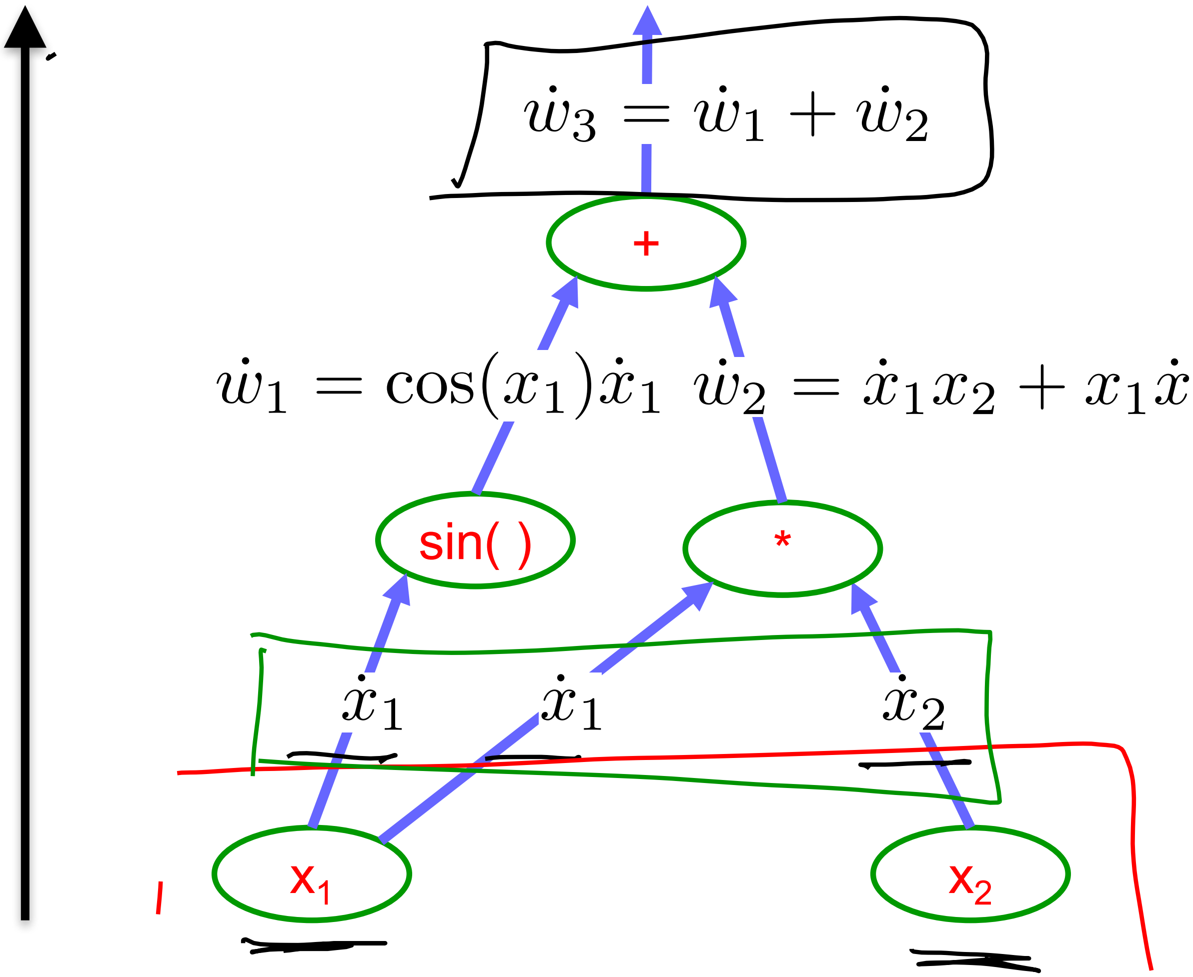
$1 \; \boxed{\frac{\partial x_1}{\partial a}} = \dot{x}_1$

$\dot{x}_2 = \boxed{\frac{\partial x_2}{\partial a}} \; 0$

$a = x_1$

Example: Forward mode AD

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

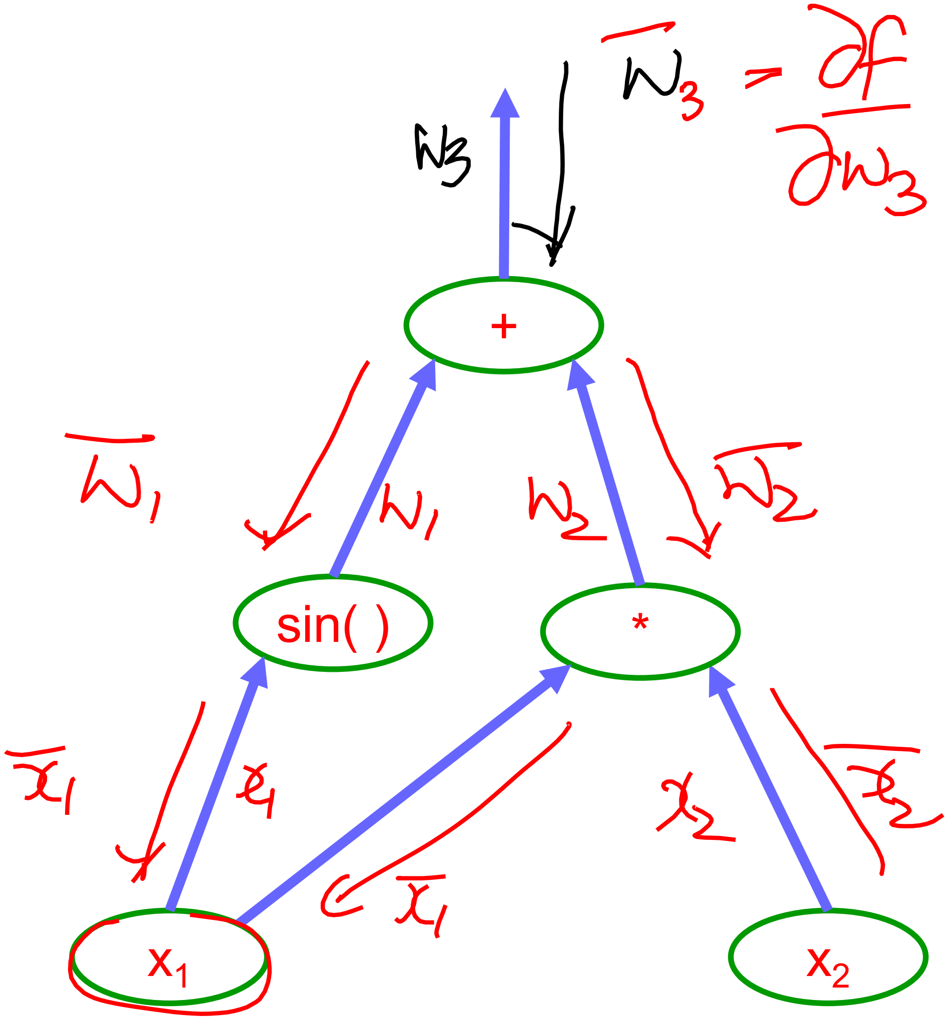


$\dot{w}_3 = \dot{w}_1 + \dot{w}_2$
+
$\dot{w}_1 = \cos(x_1)\dot{x}_1$
$\dot{w}_2 = \dot{x}_1 x_2 + x_1 \dot{x}_2$
sin( )
*
$\dot{x}_1$
$\dot{x}_1$
$\dot{x}_2$
x_1
x_2

Example: Reverse mode AD

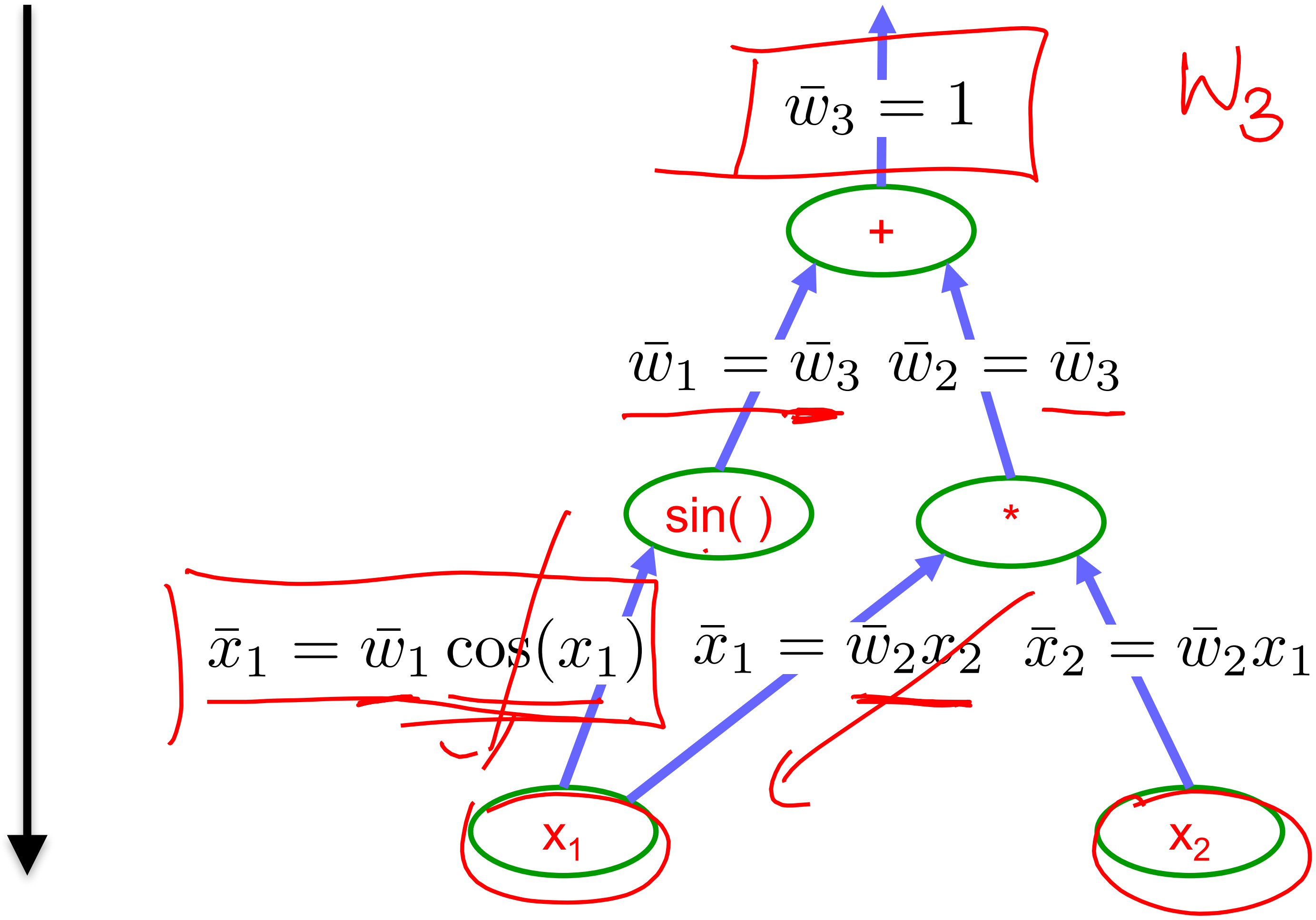
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

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



Example: Reverse mode AD

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



$$\boxed{\bar{w}_3 = 1}$$

$$\underline{\bar{w}_1 = \bar{w}_3} \quad \bar{w}_2 = \underline{\bar{w}_3}$$

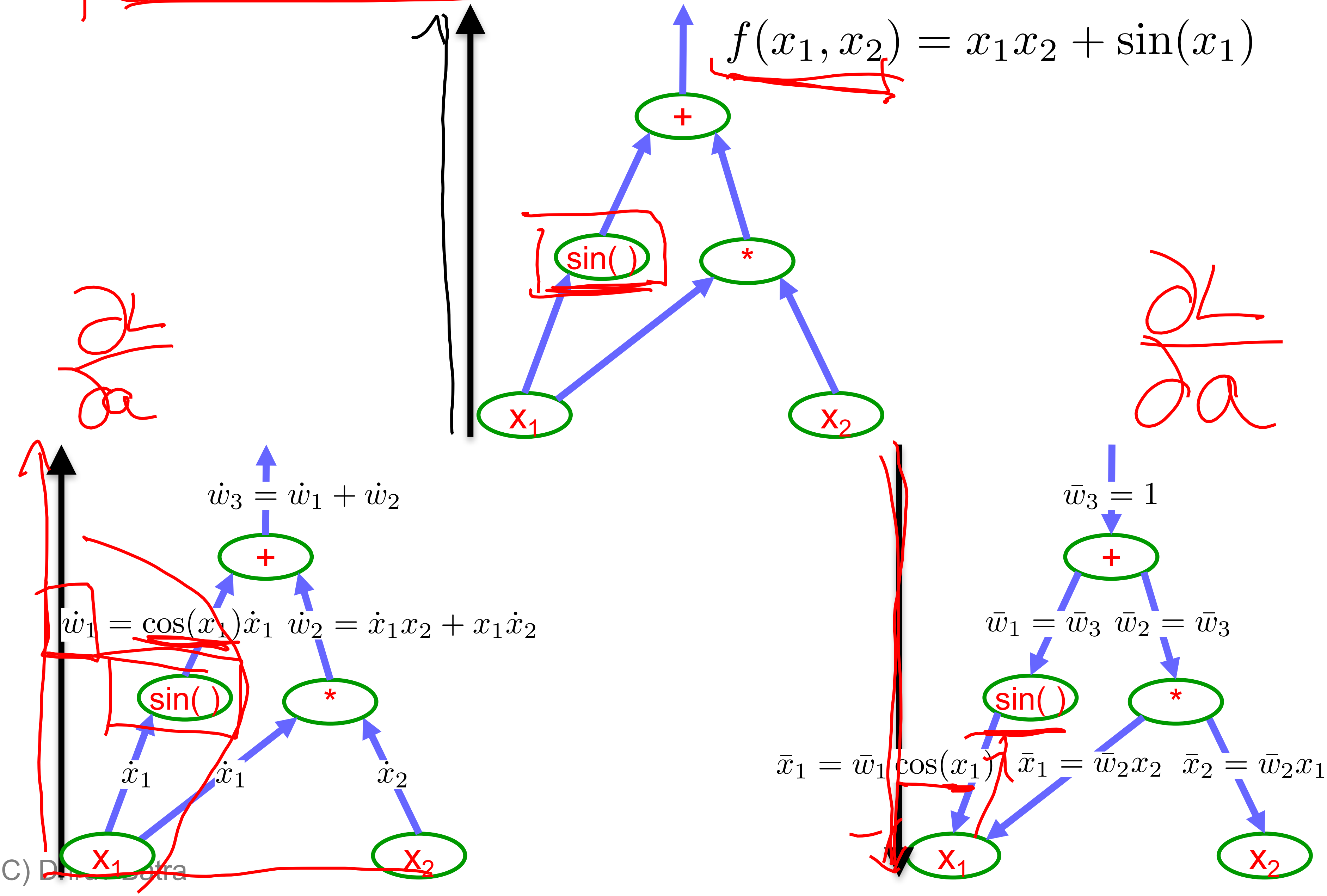
$$\boxed{\underline{\bar{x}_1 = \bar{w}_1 \cos(x_1)}} \quad \bar{x}_1 = \underline{\bar{w}_2 x_2} \quad \bar{x}_2 = \bar{w}_2 x_1$$

$$w_3 = w_1 + w_2$$

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

Forward Pass vs
Forward mode AD vs Reverse Mode AD

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



+
sin( )
*
x1
x2
$\dot{w}_3 = \dot{w}_1 + \dot{w}_2$
+
$\dot{w}_1 = \cos(x_1)\dot{x}_1$
$\dot{w}_2 = \dot{x}_1 x_2 + x_1 \dot{x}_2$
sin( )
*
$\dot{x}_1$
$\dot{x}_1$
$\dot{x}_2$
x1
x2
$\bar{w}_3 = 1$
+
$\bar{w}_1 = \bar{w}_3$
$\bar{w}_2 = \bar{w}_3$
sin( )
*
$\bar{x}_1 = \bar{w}_1 \cos(x_1)$
$\bar{x}_1 = \bar{w}_2 x_2$
$\bar{x}_2 = \bar{w}_2 x_1$
x1
x2

$\frac{\partial L}{\partial a}$

$\frac{\partial L}{\partial a}$

Forward mode vs Reverse Mode

- What are the differences?

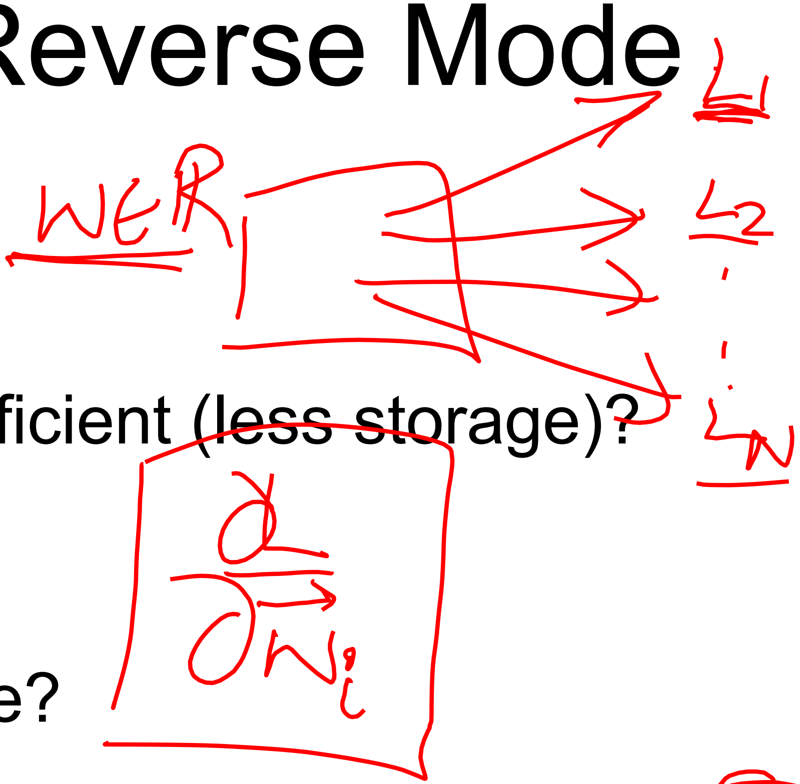
- Which one is more memory efficient (less storage)?
  - Forward or backward?

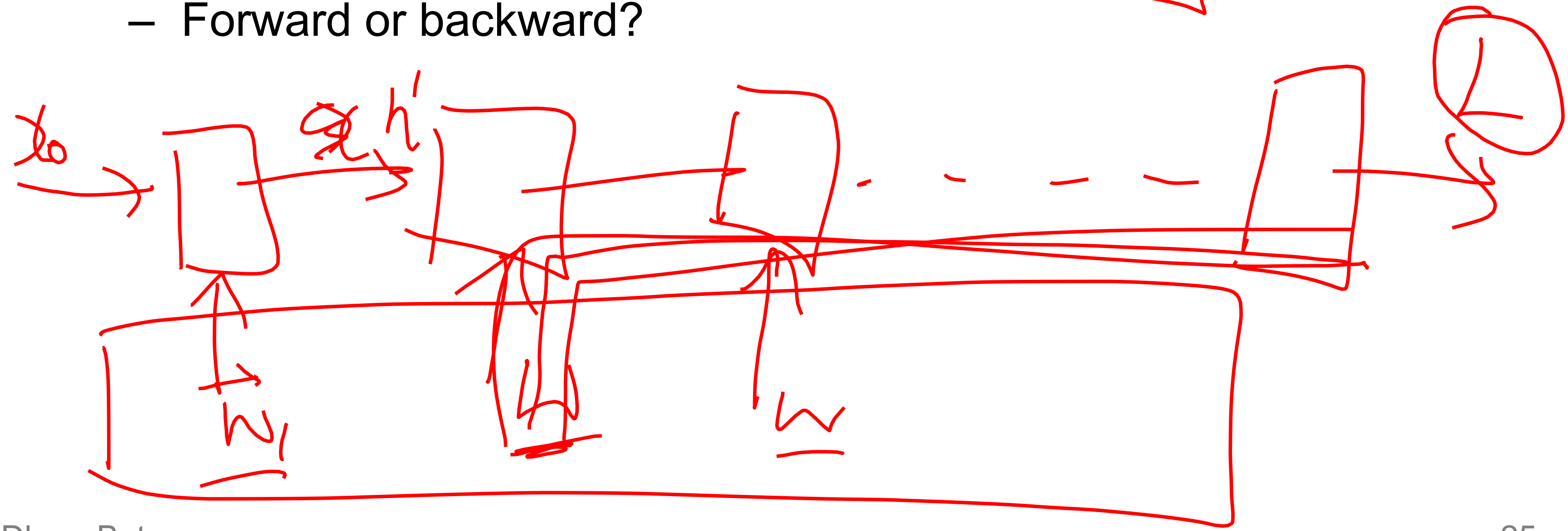
Forward mode vs Reverse Mode

- What are the differences?

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

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



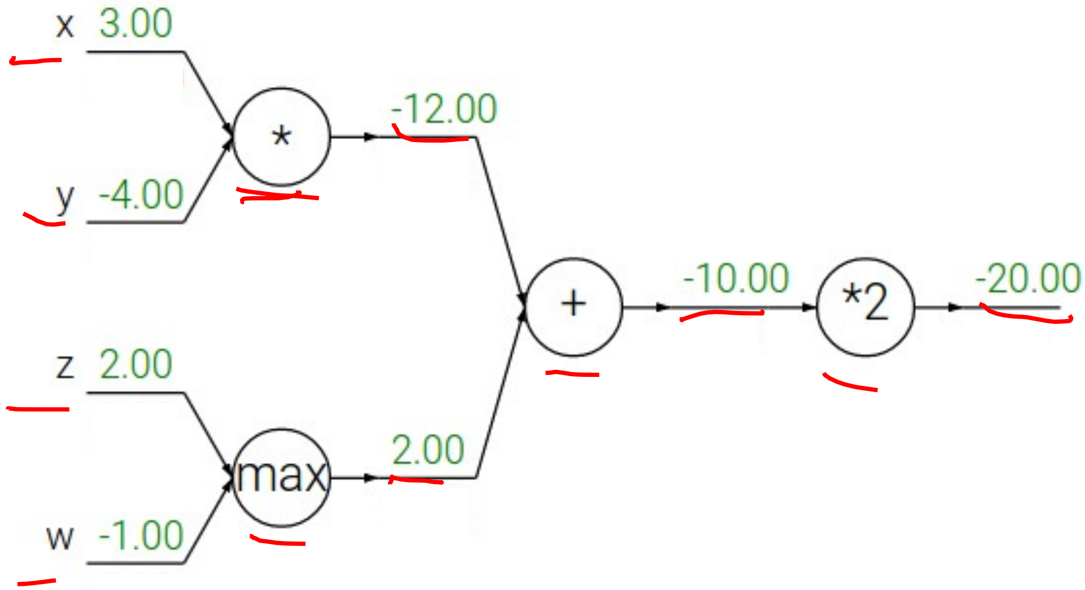


Plan for Today

- (Finish) Computing Gradients
  - Forward mode vs Reverse mode AD
  - Patterns in backprop
  - Backprop in FC+ReLU NNs
- Convolutional Neural Networks

Patterns in backward flow

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



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

Patterns in backward flow

$w_3 = w_1 + w_2$

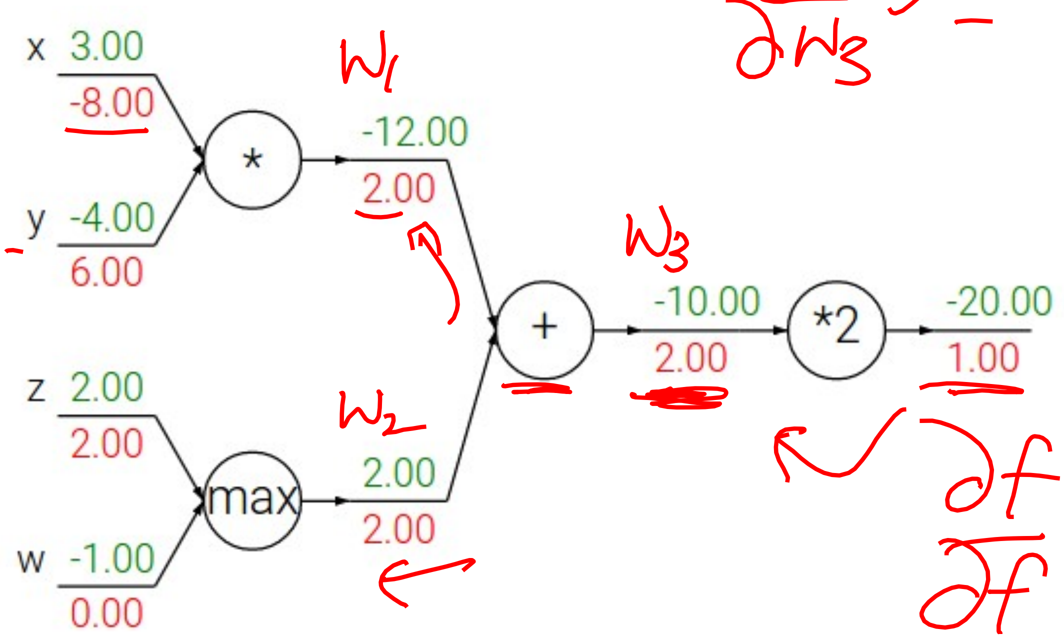
$\frac{\partial w_3}{\partial w_1} = 1$

$w_1 = xy$

$\frac{\partial w_1}{\partial x} = y$

$f = 2 w_3$

$\frac{\partial f}{\partial w_3} = 2$



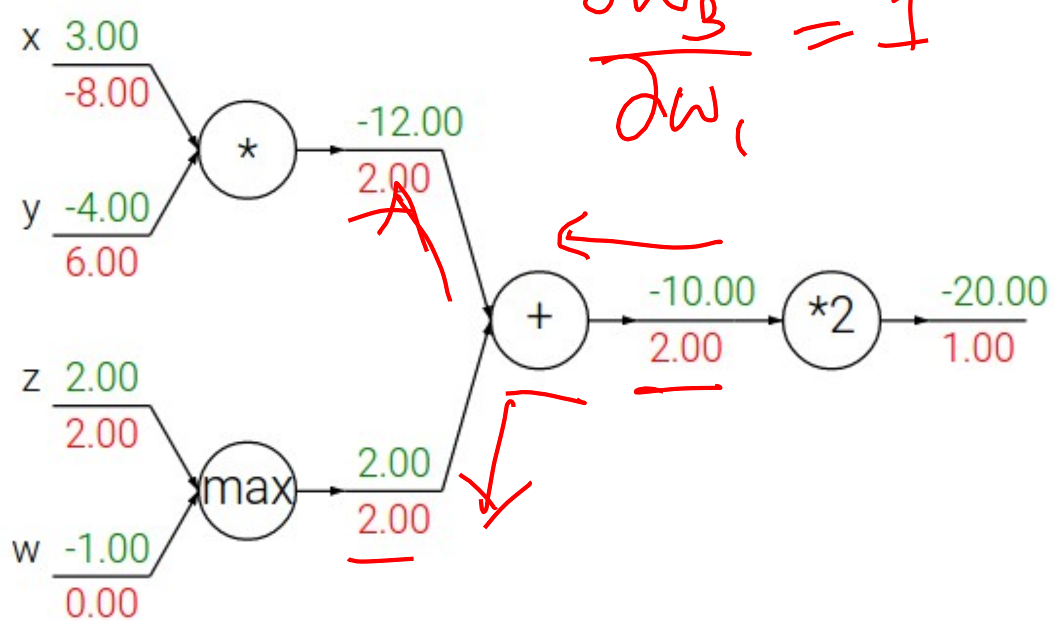
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

$w_3 = w_1 + w_2$

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

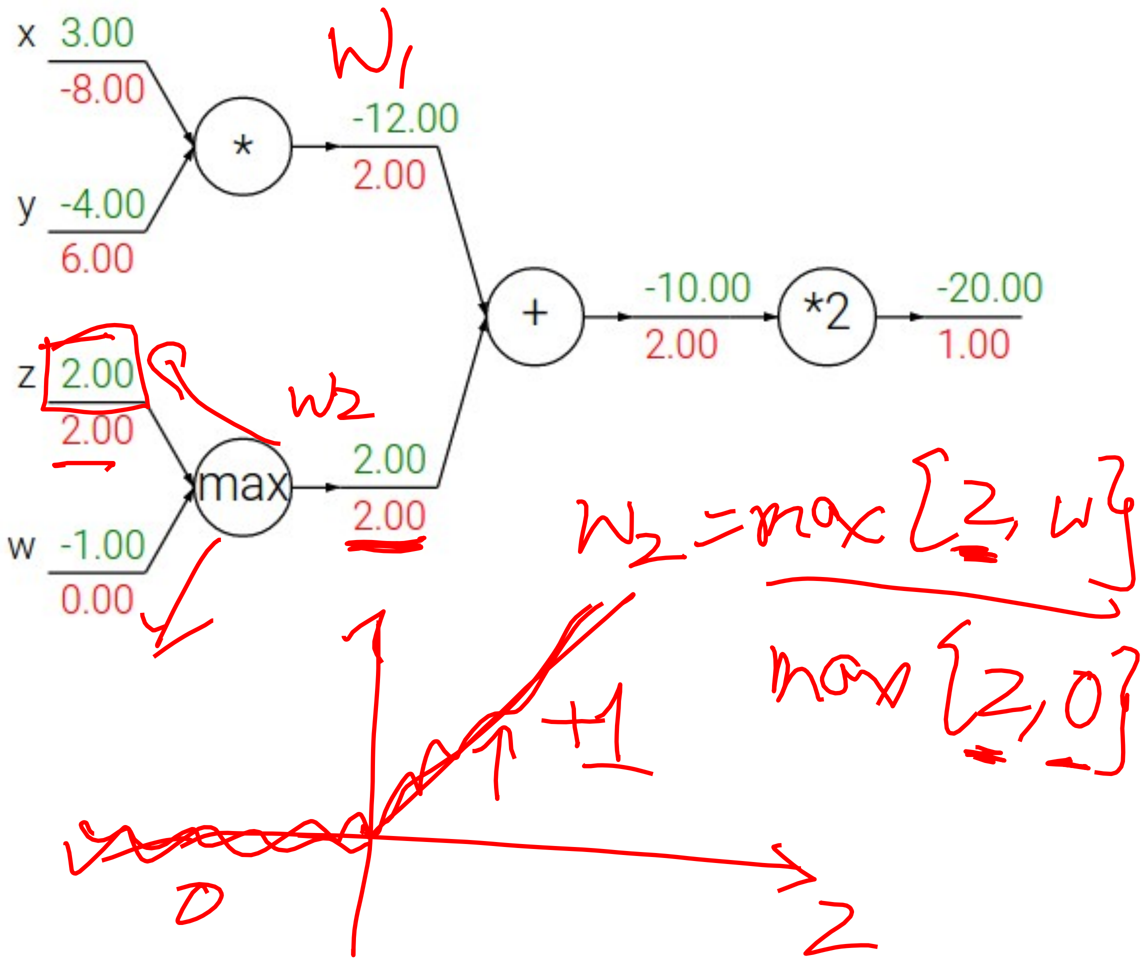


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

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

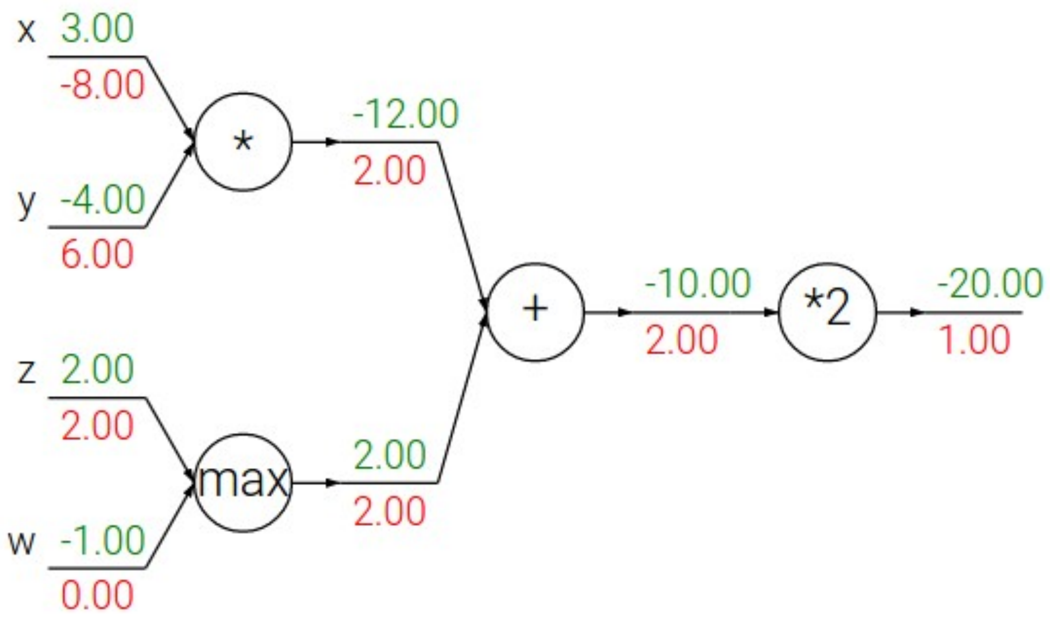


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router



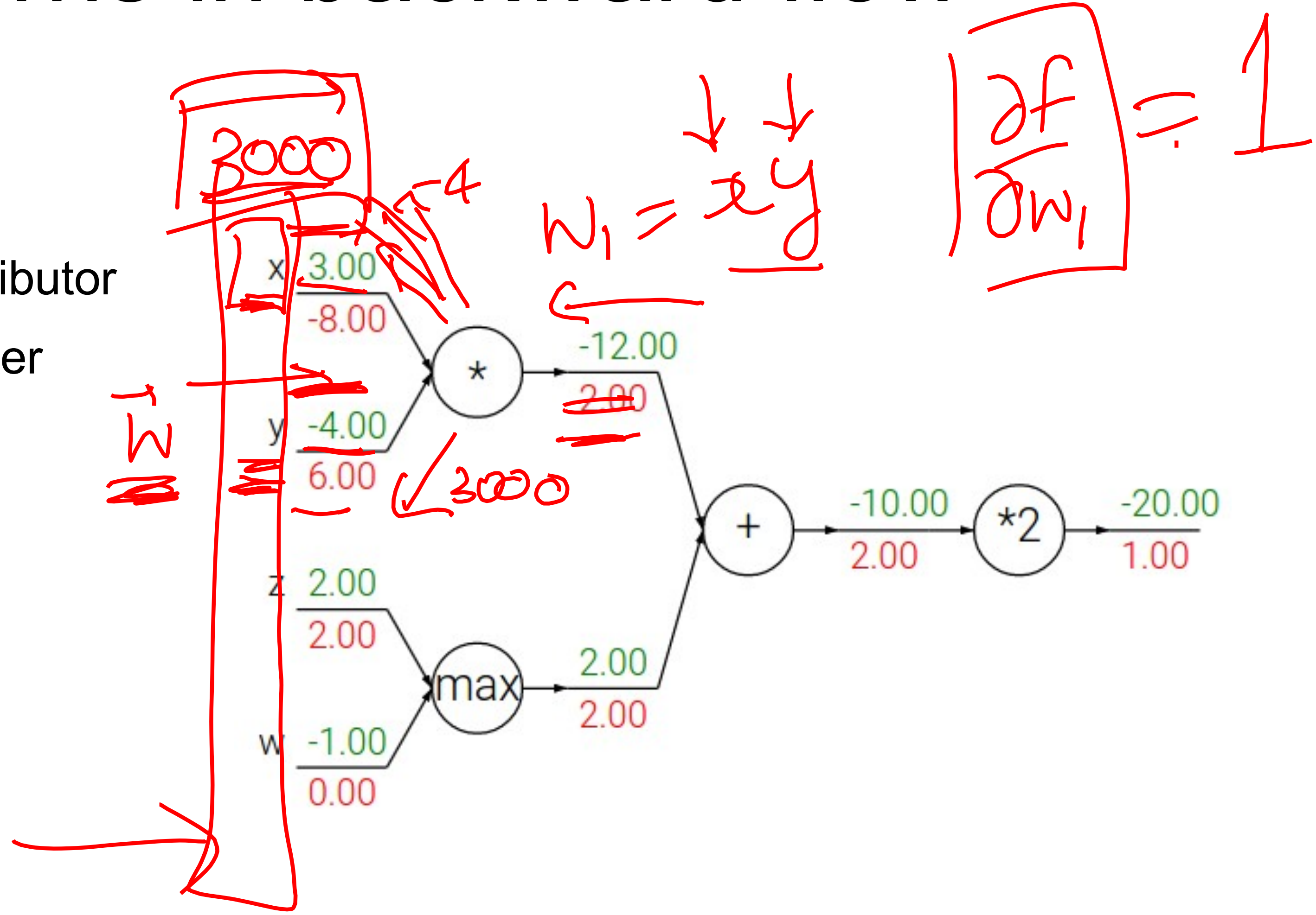
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

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



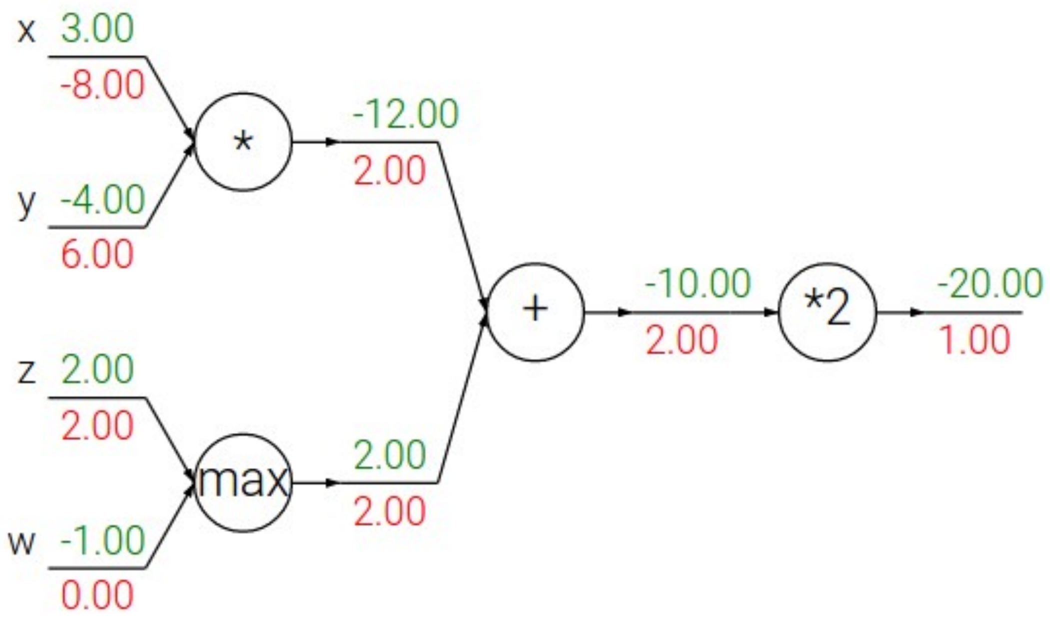
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

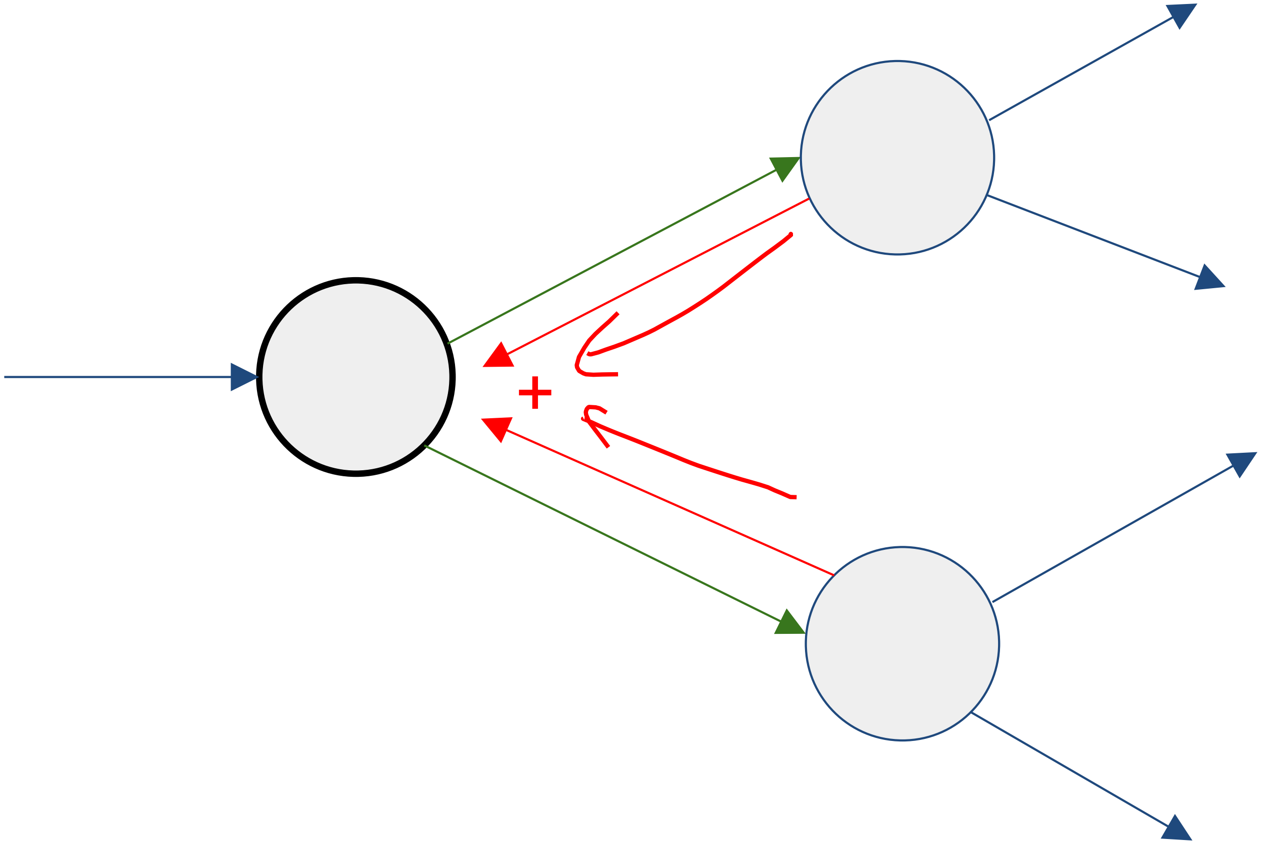
max gate: gradient router

mul gate: gradient switcher



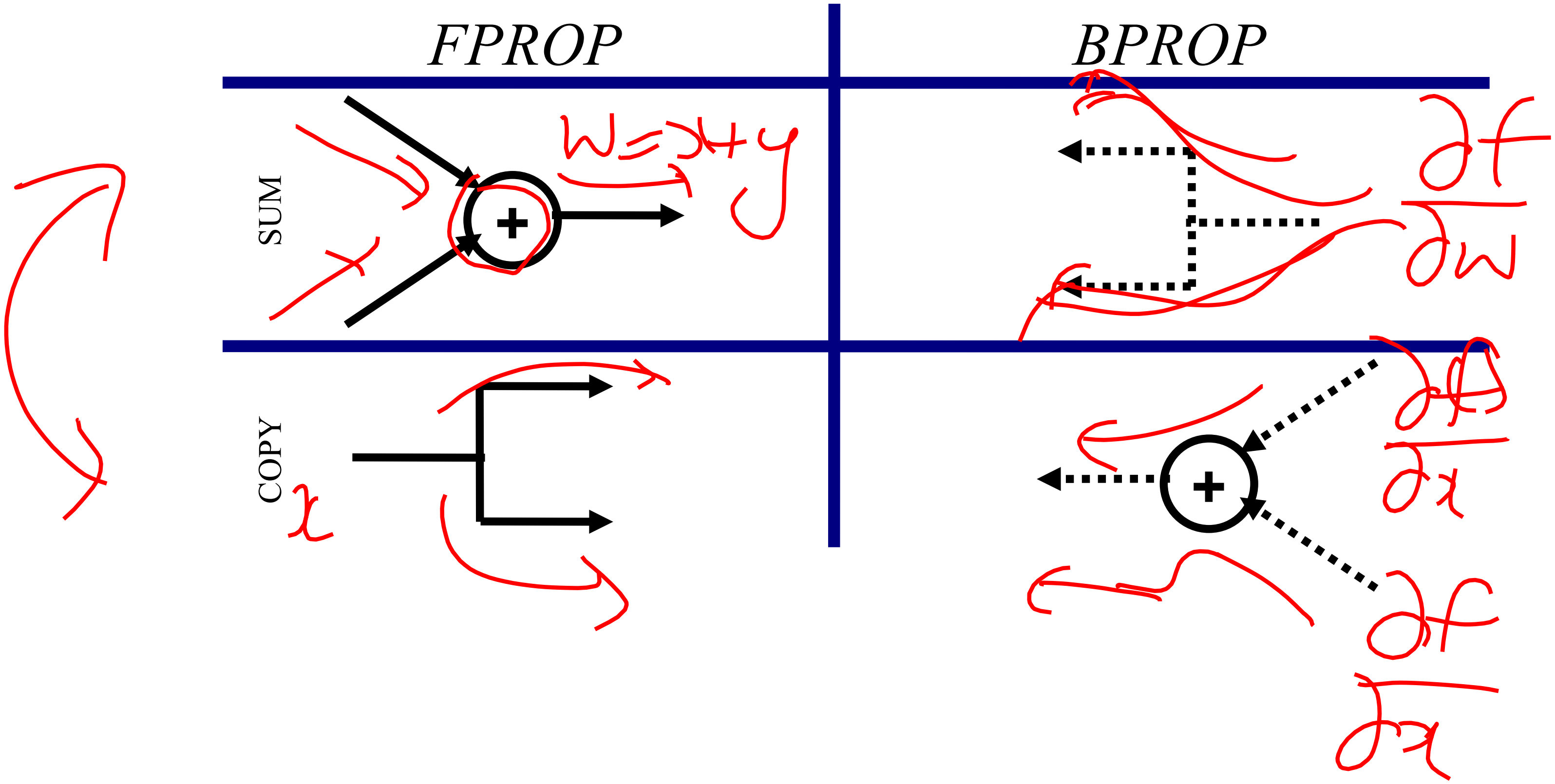
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Gradients add at branches

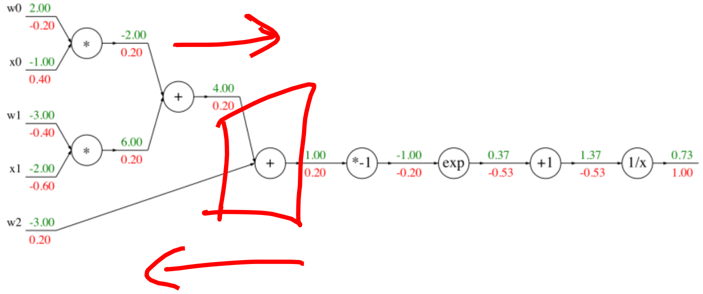


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

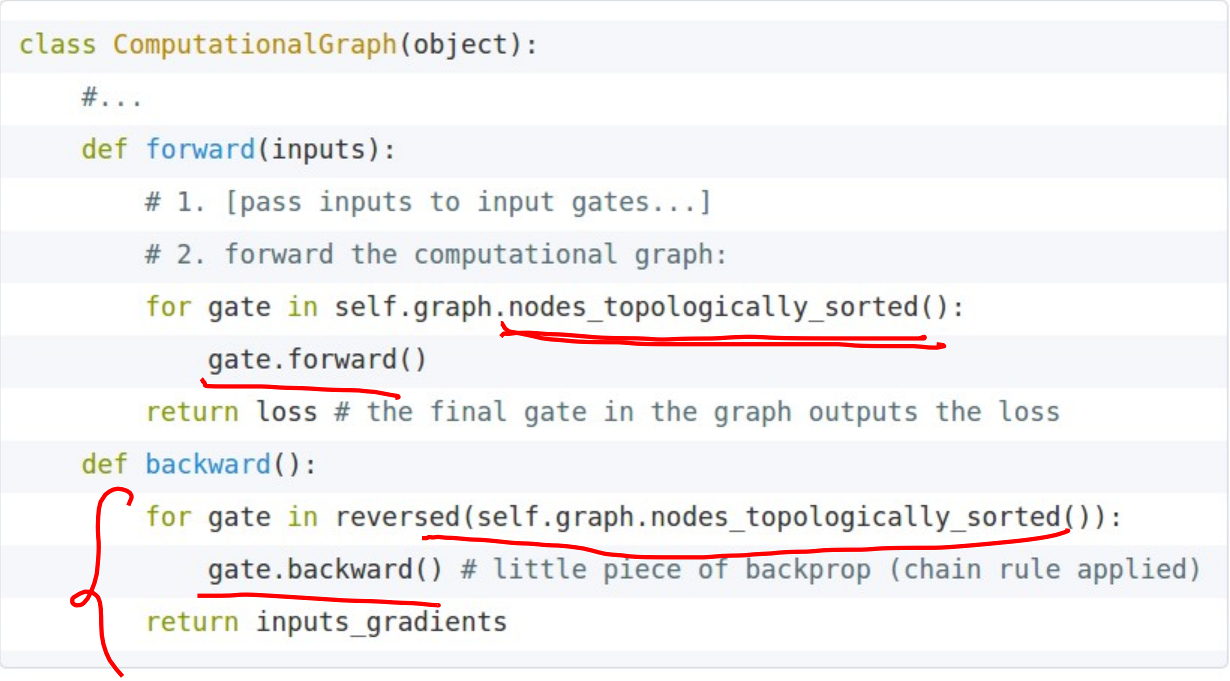
Duality in Fprop and Bprop


FPROP
BPROP
SUM
COPY
+
+
w=x+y
y
x
x
$\frac{\partial f}{\partial w}$
$\frac{\partial f}{\partial x}$
$\frac{\partial f}{\partial x}$

Modularized implementation: forward / backward API



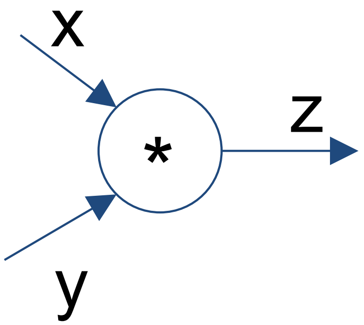
Graph (or Net) object *(rough psuedo code)*



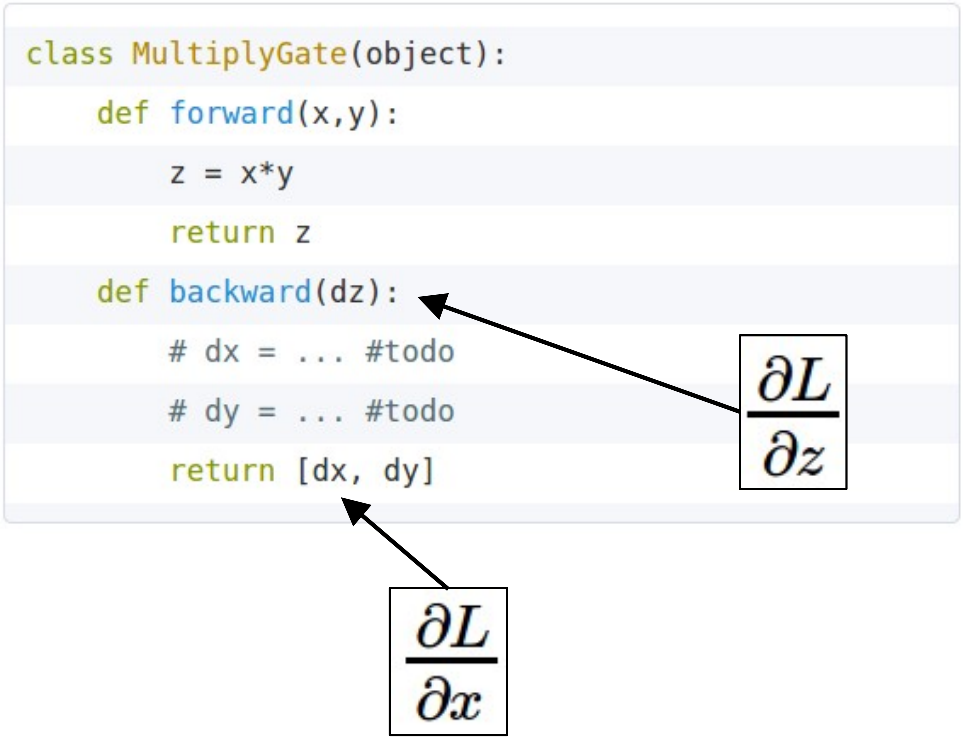
```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

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

Modularized implementation: forward / backward API



(x,y,z are scalars)



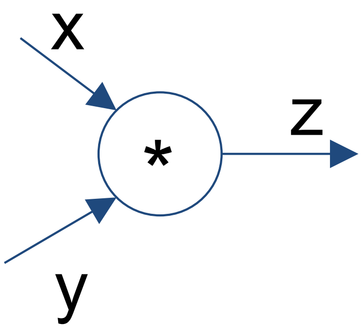
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$$\boxed{\frac{\partial L}{\partial z}}$$

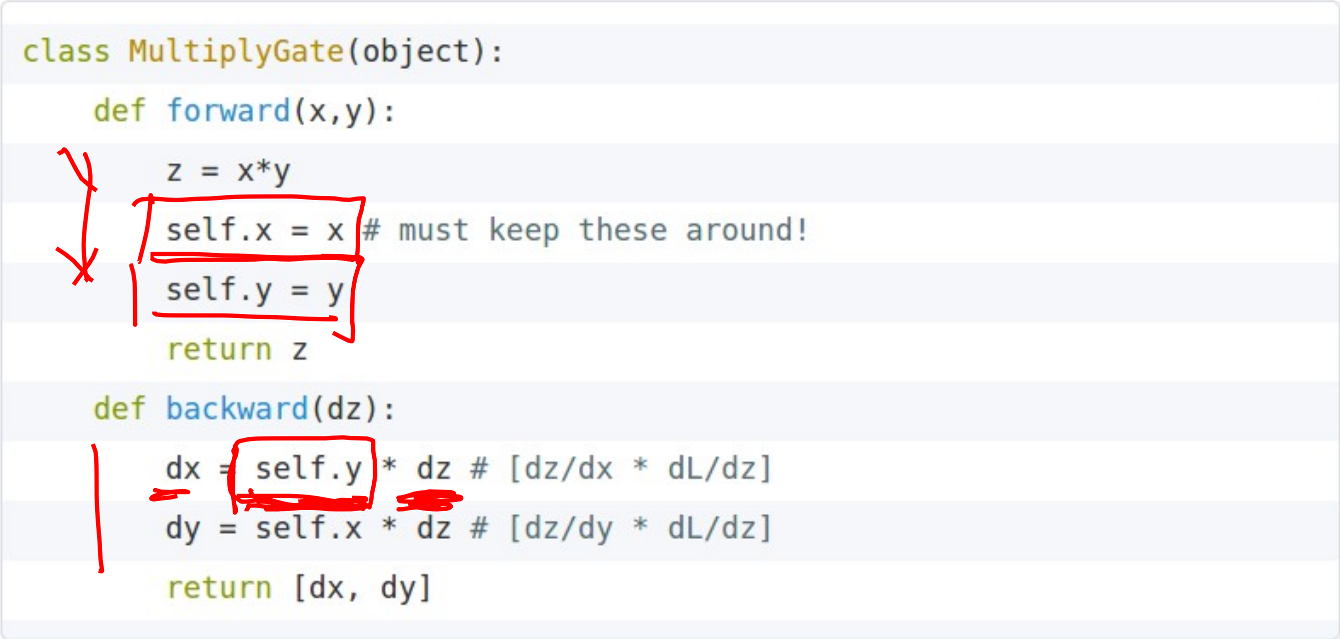
$$\boxed{\frac{\partial L}{\partial x}}$$

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

Modularized implementation: forward / backward API



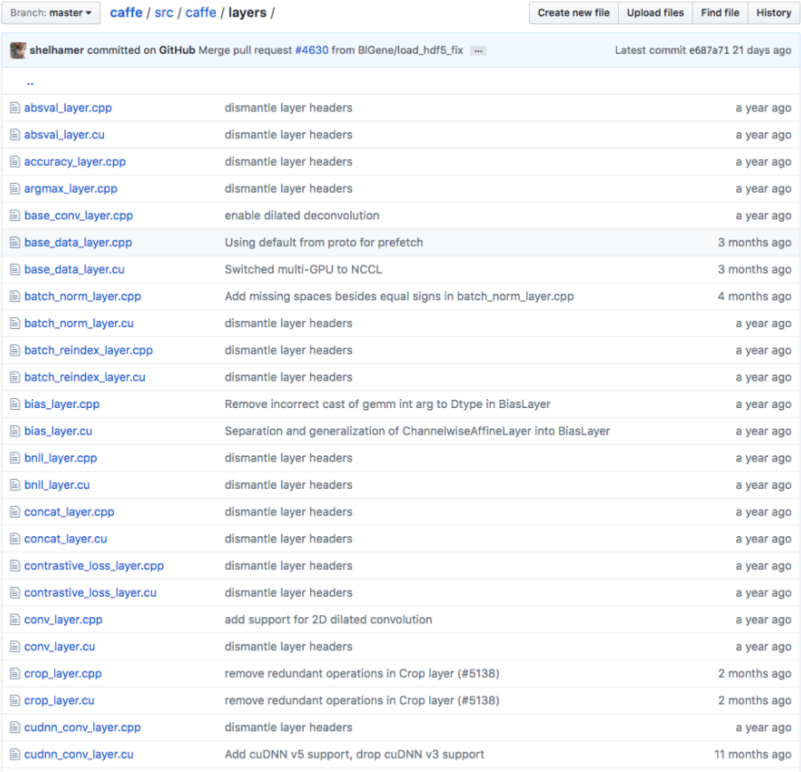
(x,y,z are scalars)



```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

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

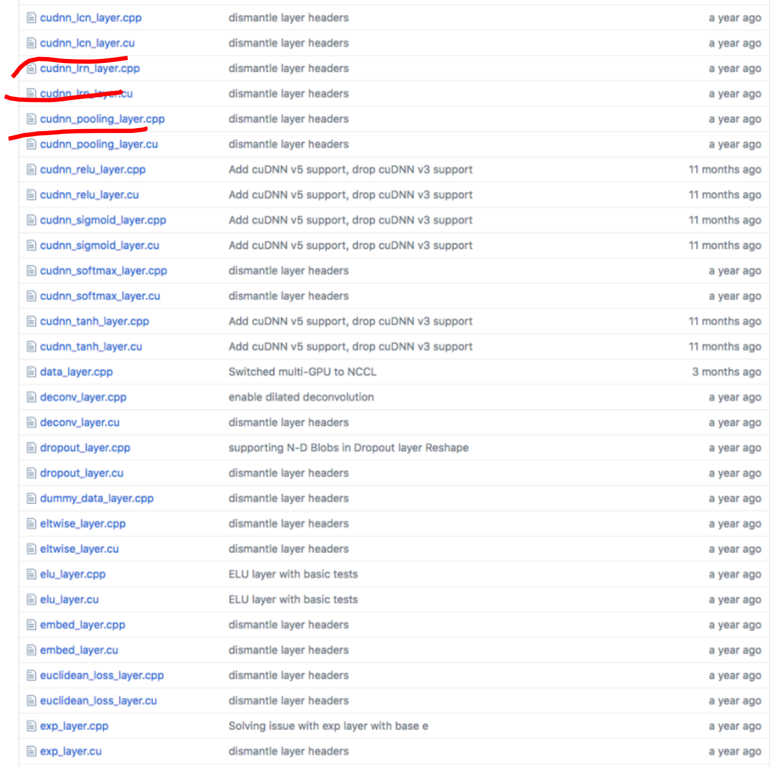
Example: Caffe layers



shelhamer committed on GitHub Merge pull request #4630 from BlGene/load_hdf5_fix … Latest commit e687a71 21 days ago

..

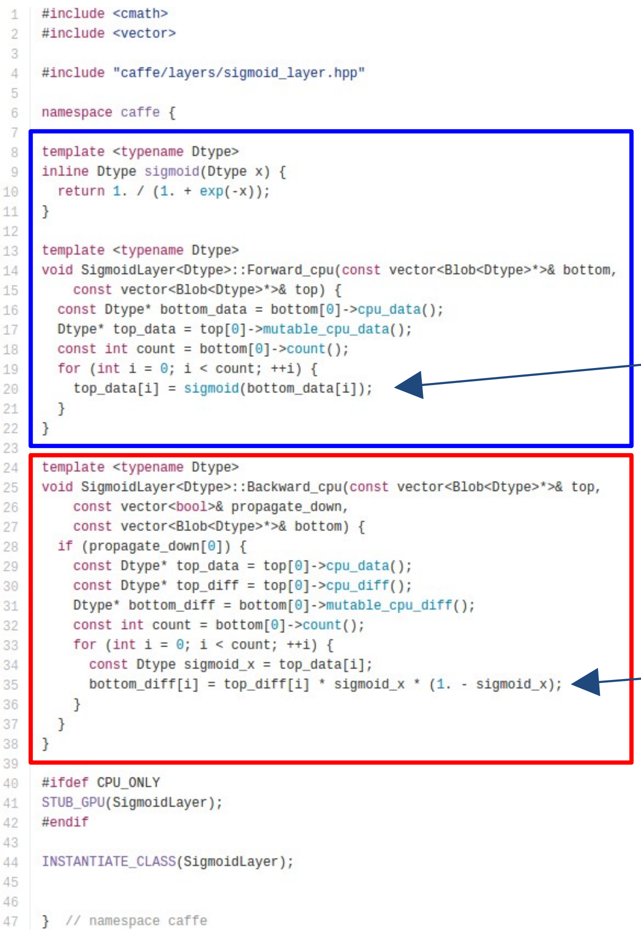
absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default from proto for prefetch	3 months ago
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago
batch_norm_layer.cu	dismantle layer headers	a year ago
batch_reindex_layer.cpp	dismantle layer headers	a year ago
batch_reindex_layer.cu	dismantle layer headers	a year ago
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago
bnll_layer.cpp	dismantle layer headers	a year ago
bnll_layer.cu	dismantle layer headers	a year ago
concat_layer.cpp	dismantle layer headers	a year ago
concat_layer.cu	dismantle layer headers	a year ago
contrastive_loss_layer.cpp	dismantle layer headers	a year ago
contrastive_loss_layer.cu	dismantle layer headers	a year ago
conv_layer.cpp	add support for 2D dilated convolution	a year ago
conv_layer.cu	dismantle layer headers	a year ago
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago
cudnn_conv_layer.cpp	dismantle layer headers	a year ago
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago



cudnn_lcn_layer.cpp	dismantle layer headers	a year ago
cudnn_lcn_layer.cu	dismantle layer headers	a year ago
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago
cudnn_lrn_layer.cu	dismantle layer headers	a year ago
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago
cudnn_pooling_layer.cu	dismantle layer headers	a year ago
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago
cudnn_softmax_layer.cu	dismantle layer headers	a year ago
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
data_layer.cpp	Switched multi-GPU to NCCL	3 months ago
deconv_layer.cpp	enable dilated deconvolution	a year ago
deconv_layer.cu	dismantle layer headers	a year ago
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago
dropout_layer.cu	dismantle layer headers	a year ago
dummy_data_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cu	dismantle layer headers	a year ago
elu_layer.cpp	ELU layer with basic tests	a year ago
elu_layer.cu	ELU layer with basic tests	a year ago
embed_layer.cpp	dismantle layer headers	a year ago
embed_layer.cu	dismantle layer headers	a year ago
euclidean_loss_layer.cpp	dismantle layer headers	a year ago
euclidean_loss_layer.cu	dismantle layer headers	a year ago
exp_layer.cpp	Solving issue with exp layer with base e	a year ago
exp_layer.cu	dismantle layer headers	a year ago

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

Caffe Sigmoid Layer



```
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

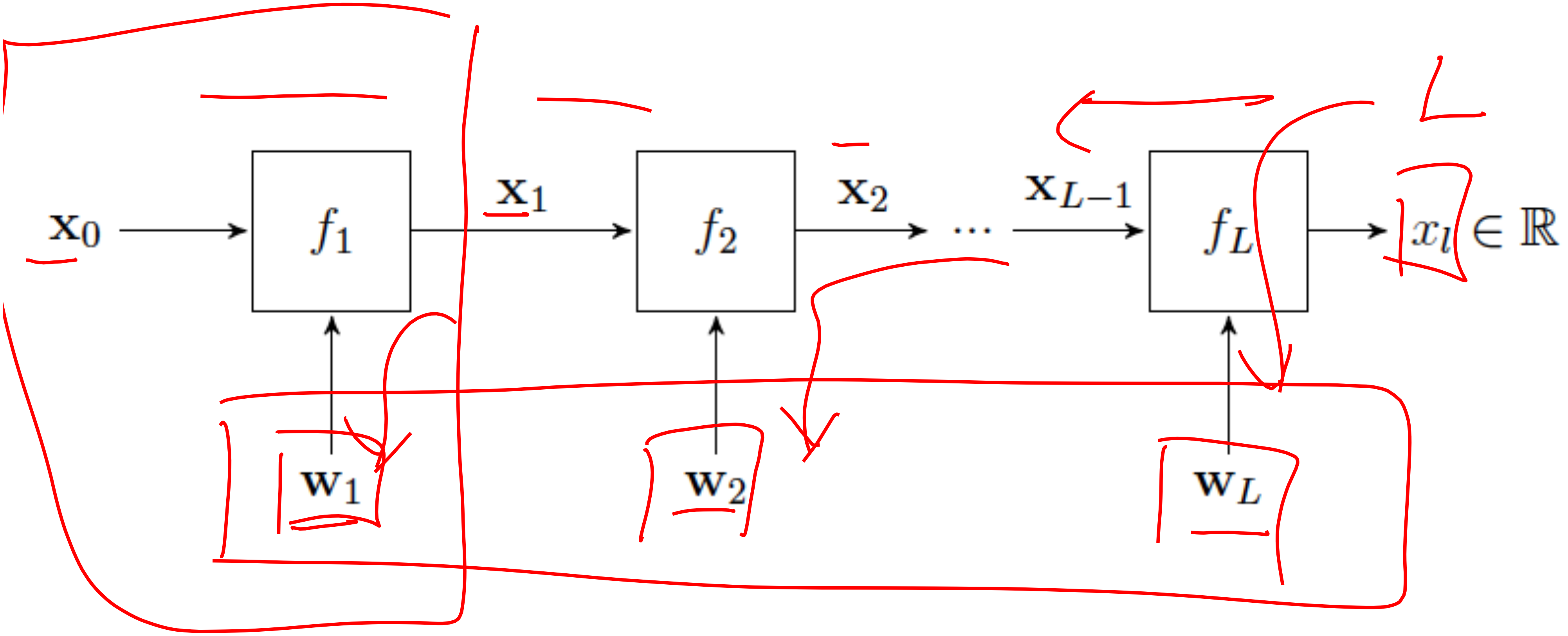
Caffe is licensed under BSD 2-Clause

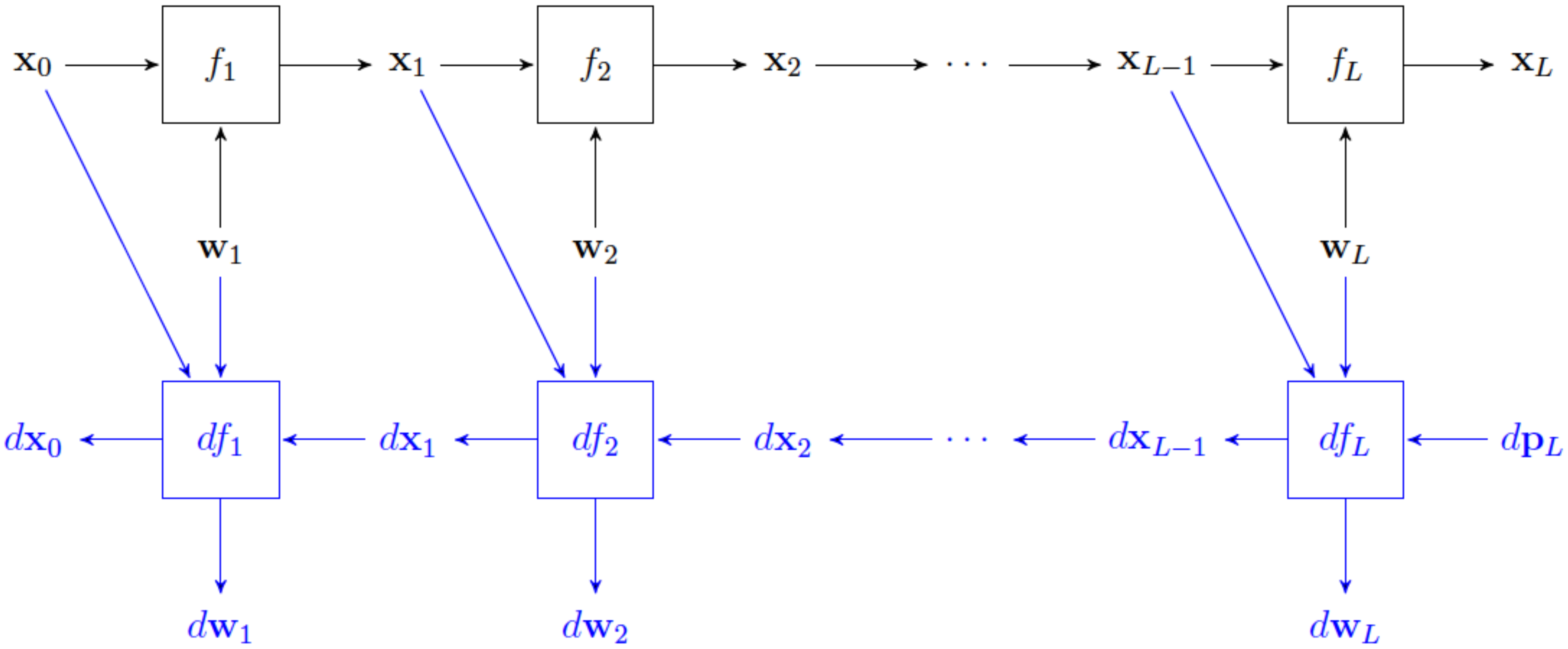
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\left(1-\sigma(x)\right)\sigma(x)$$ * top_diff (chain rule)

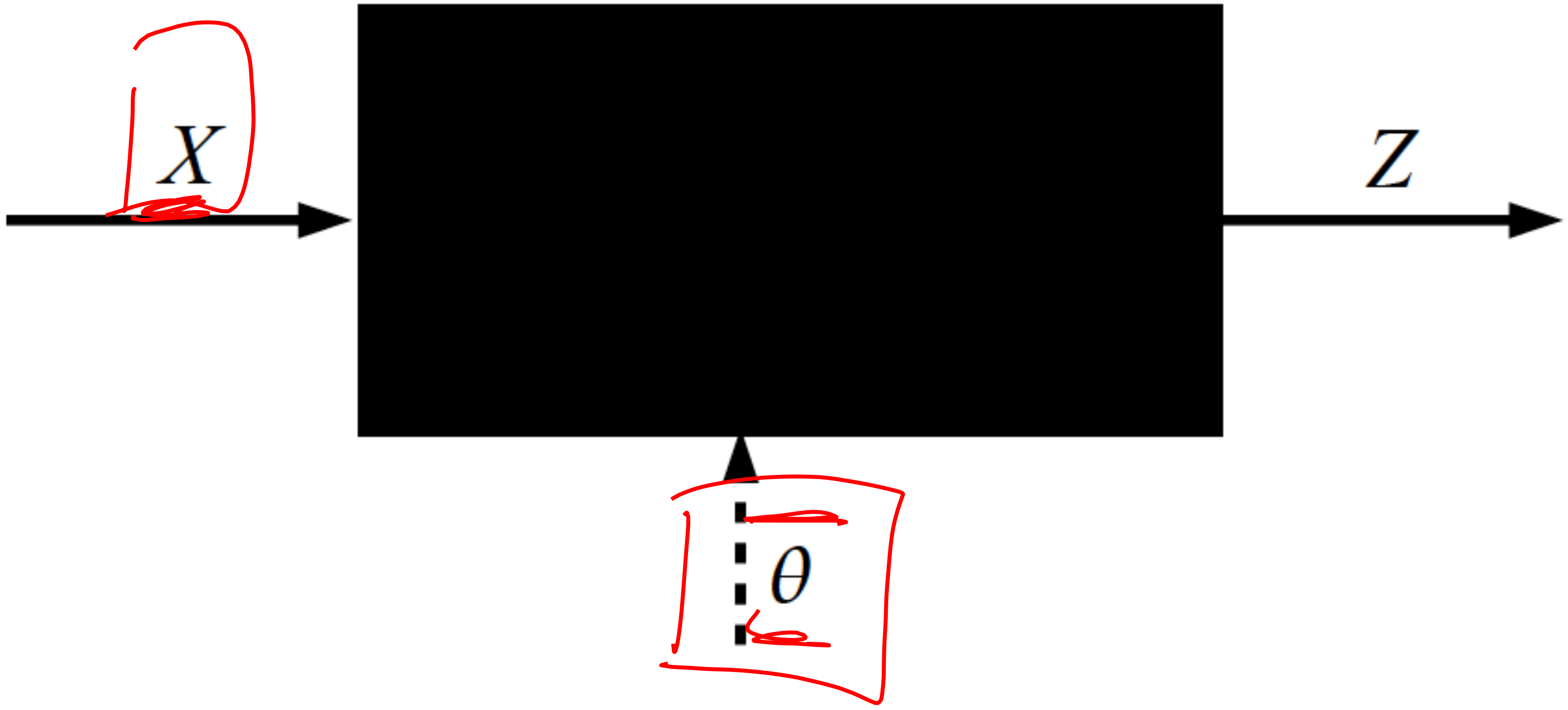
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

$\frac{\partial L}{\partial w_i}$




$\mathbf{x}_0$
f_1
$\mathbf{x}_1$
f_2
$\mathbf{x}_2$
$\cdots$
$\mathbf{x}_{L-1}$
f_L
$\mathbf{x}_L$
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_L$
$d\mathbf{x}_0$
df_1
$d\mathbf{x}_1$
df_2
$d\mathbf{x}_2$
$\cdots$
$d\mathbf{x}_{L-1}$
df_L
$d\mathbf{p}_L$
$d\mathbf{w}_1$
$d\mathbf{w}_2$
$d\mathbf{w}_L$

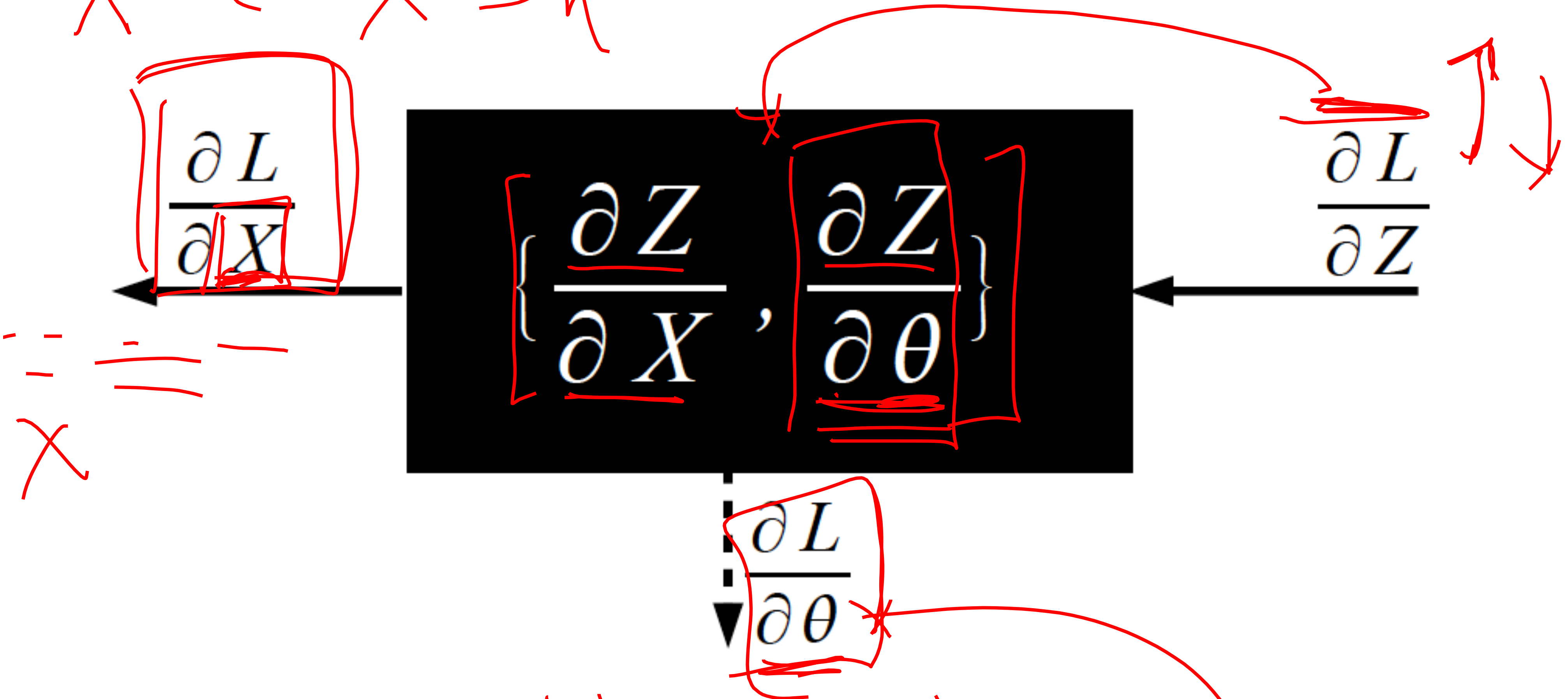
Key Computation in DL: Forward-Prop



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Key Computation in DL: Back-Prop

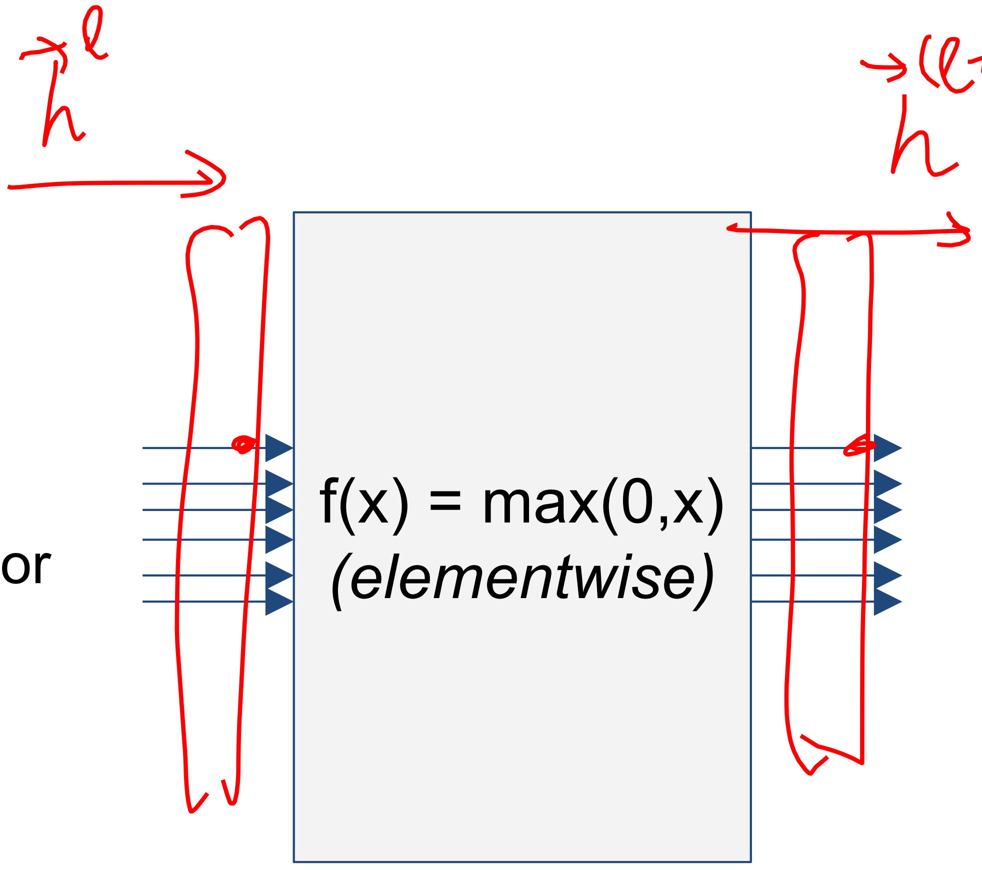
$x^{(t+1)} \leftarrow x^{(t)} - \eta$


$\frac{\partial L}{\partial X}$
$\left\{ \frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta} \right\}$
$\frac{\partial L}{\partial Z}$
$\frac{\partial L}{\partial \theta}$
x

$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta(\ )$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Jacobian of ReLU



$\vec{h}^{\ell}$ $\vec{h}^{(\ell+1)}$

4096-d
input vector

f(x) = max(0,x)
(elementwise)

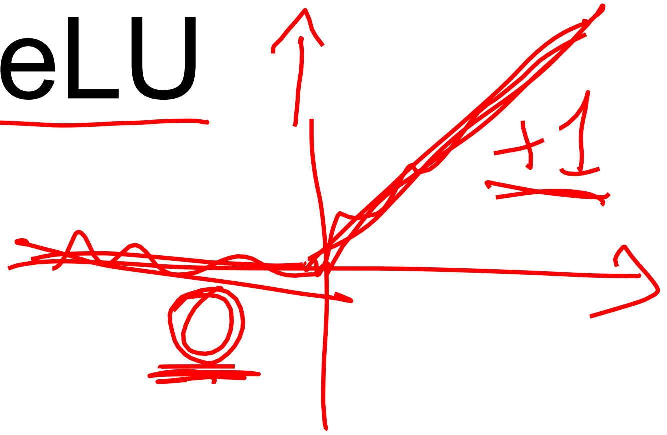
4096-d
output vector

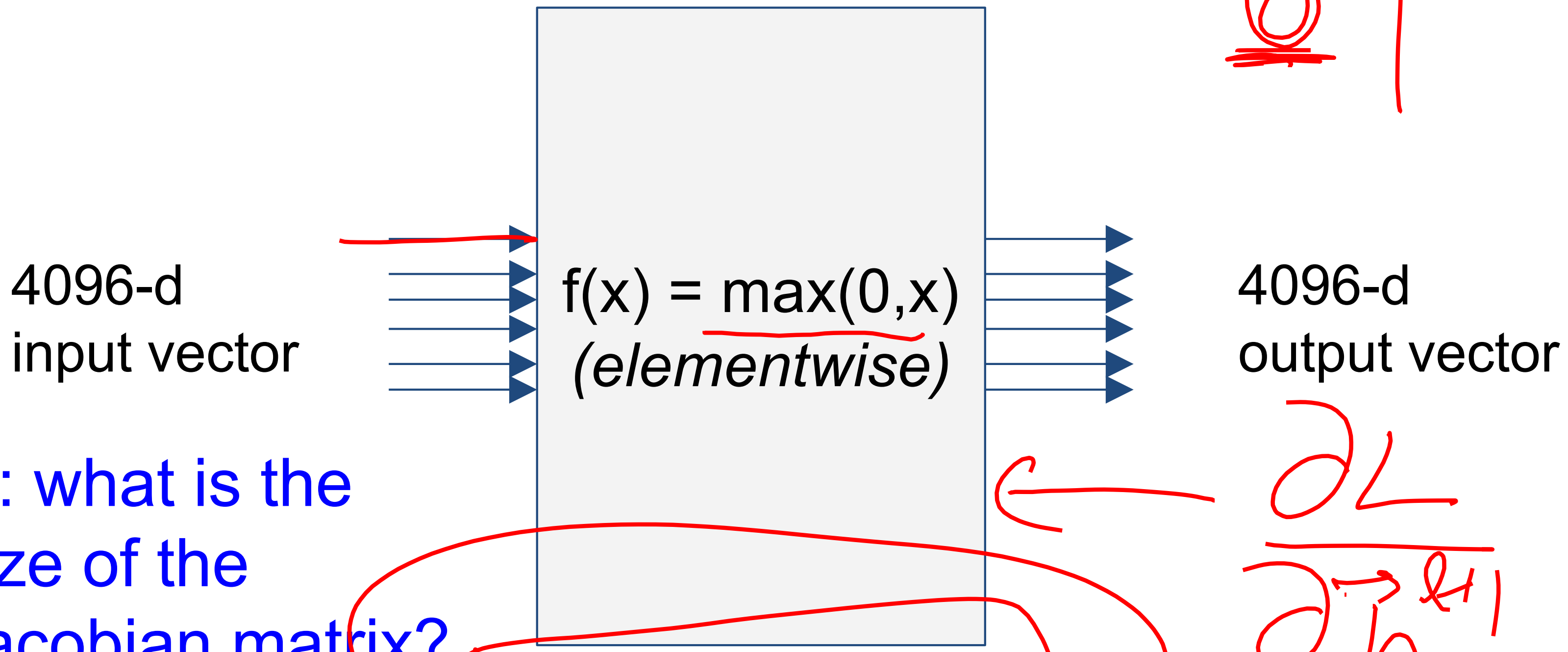
$$h_i^{\ell+1} = \max\{0, h_i^{\ell}\}$$

$$\frac{\partial h_i^{\ell+1}}{\partial h_j^{\ell}} = 0 \quad i \neq j$$

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

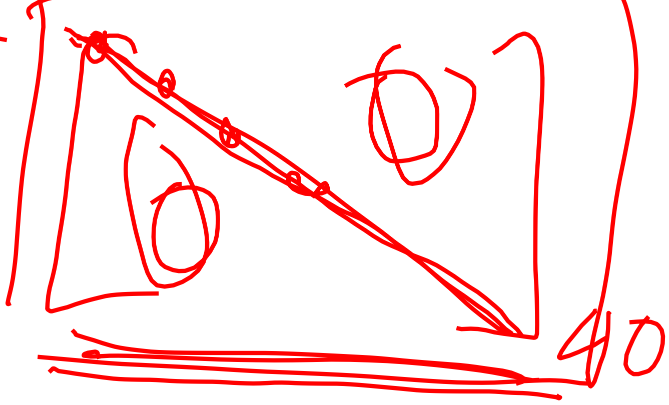
Jacobian of ReLU



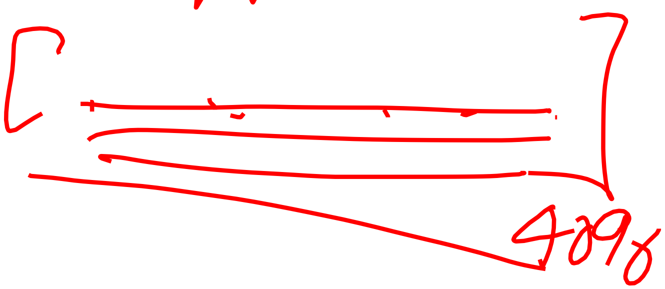


$$\frac{\partial L}{\partial h^{l+1}}$$

Q: what is the size of the Jacobian matrix?



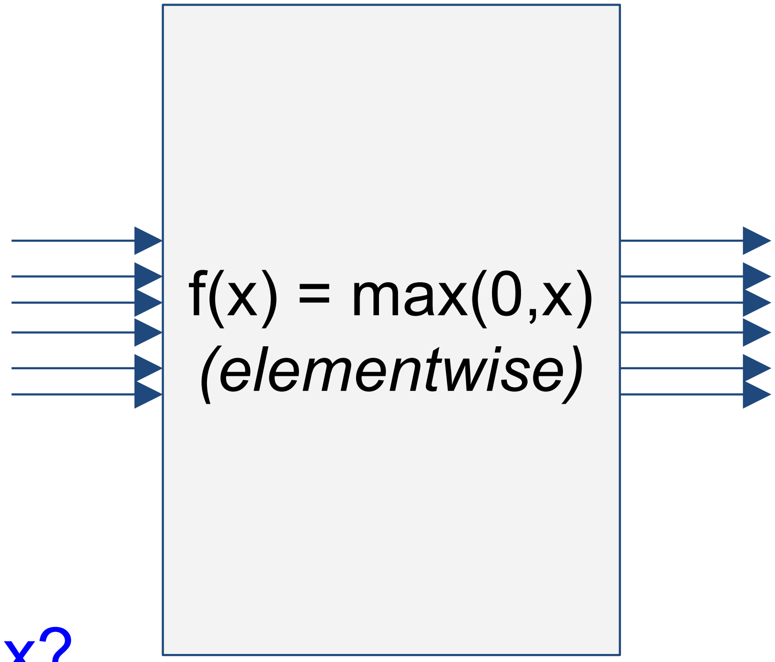
4096 x 4096



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

Jacobian of ReLU

4096-d
input vector



f(x) = max(0,x)
(elementwise)

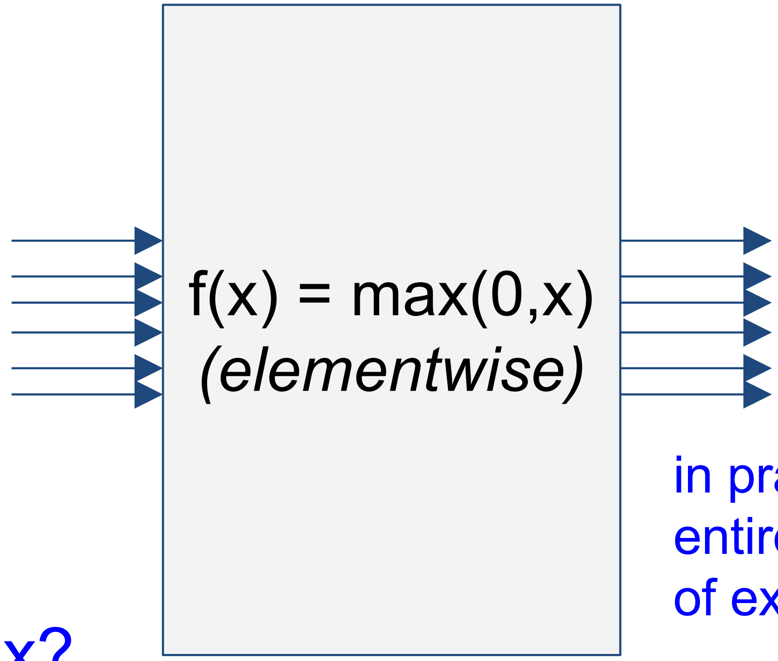
4096-d
output vector

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

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

Jacobian of ReLU

4096-d
input vector



f(x) = max(0,x)
(elementwise)

4096-d
output vector

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

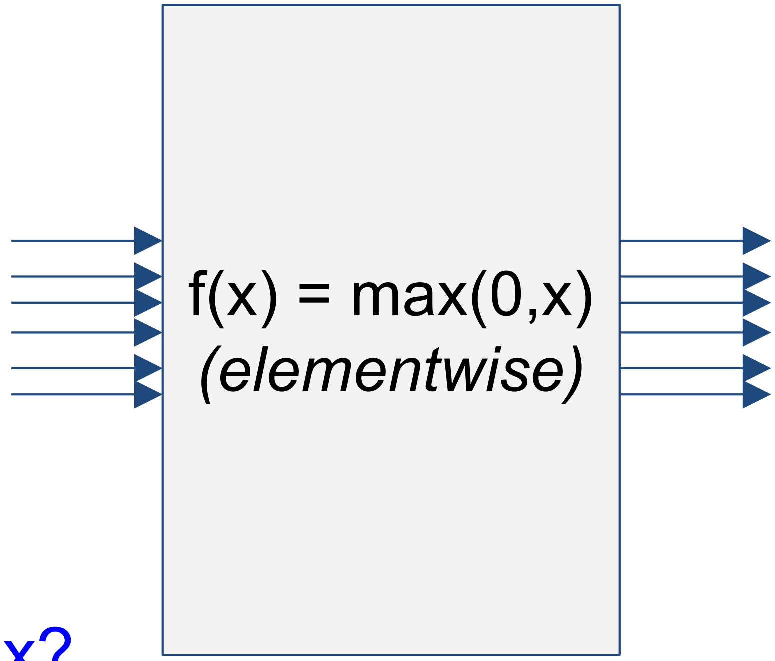
in practice we process an entire minibatch (e.g. 100) of examples at one time:

i.e. Jacobian would technically be a [409,600 x 409,600] matrix :\

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

Jacobian of ReLU

4096-d
input vector



f(x) = max(0,x)
(elementwise)

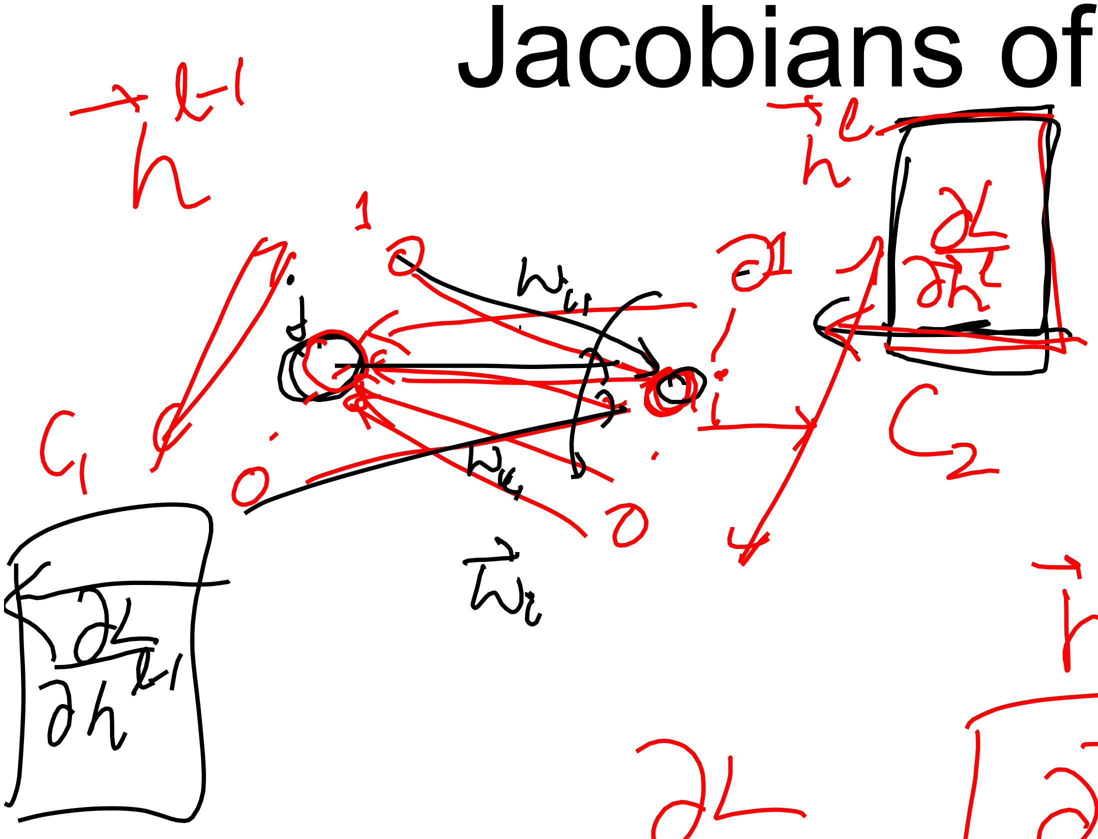
4096-d
output vector

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

Q2: what does it look like?

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

Jacobians of FC-Layer


$\vec{h}^{\ell-1}$
$\vec{h}^{\ell}$
$\frac{\partial L}{\partial \vec{h}^{\ell}}$
$\frac{\partial L}{\partial h^{\ell-1}}$
C_1
C_2
w_{i1}
w_{i4}
$\vec{w}_i$
j
i
1
0
0
0
1
4
4

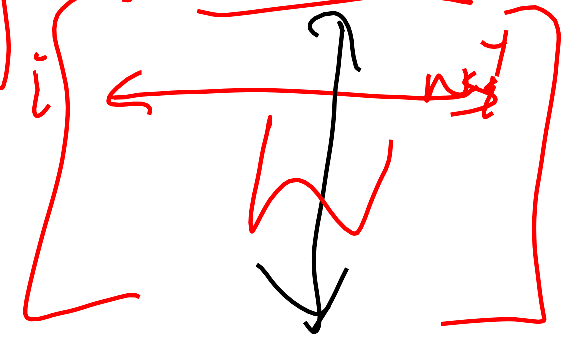
$$\vec{h}^{\ell}_i = (\vec{w}_i) \begin{bmatrix} \vec{h}^{\ell-1} \\ 1 \end{bmatrix} + b_i$$

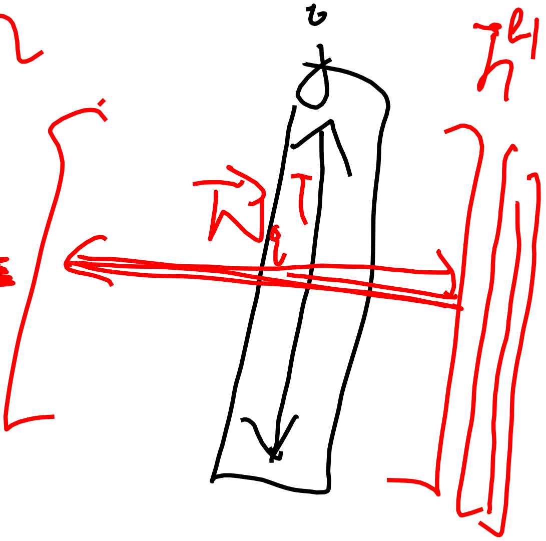
$$\vec{h}^{\ell} = W \vec{h}^{\ell-1}$$

$$\frac{\partial L}{\partial \vec{h}^{\ell}}$$

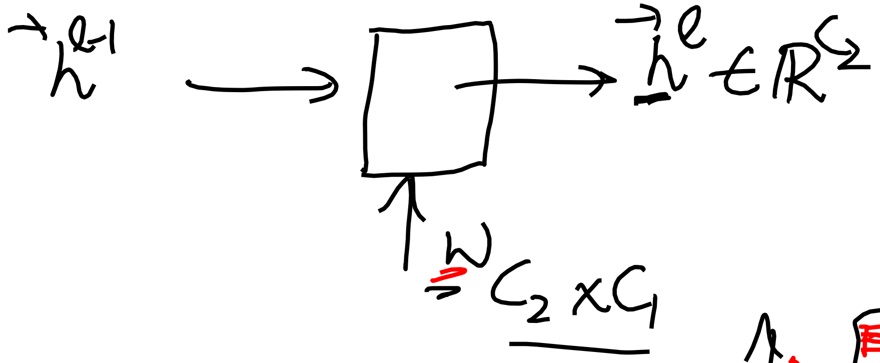
$$\frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}} = [\quad] = W$$

$$\frac{\partial L}{\partial \vec{h}^{\ell-1}} = [\quad] = [\ (\cdot)_i\ ]$$

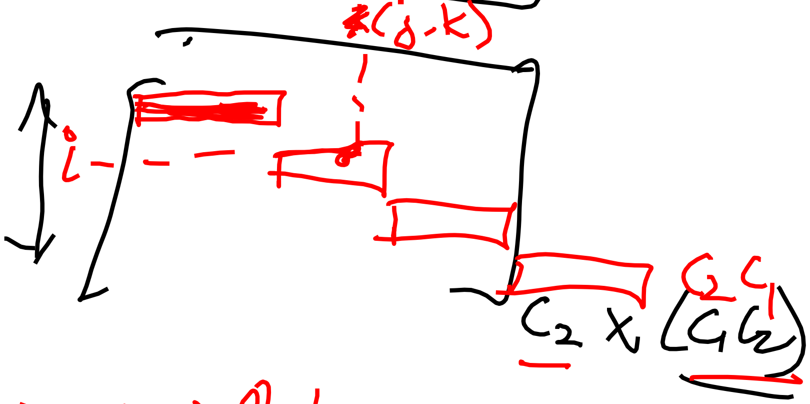




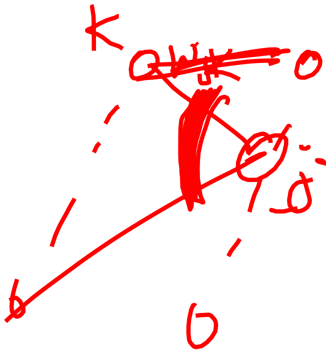
Jacobians of FC-Layer



$$\boxed{\frac{\partial \vec{h}^{\ell}}{\partial W}}$$



$$\frac{\partial \vec{h}^{\ell}_i}{\partial W_{j,k}}$$



$$\frac{\partial h^{\ell}_j}{\partial \vec{w}}$$

$$\begin{bmatrix} — \; w^{T}_{j,k} \; — \end{bmatrix}$$

$$h^{\ell}_j = \vec{w}_j^{\,T} \vec{h}^{\ell-1}$$

$$\frac{\partial h^{\ell}_j}{\partial \vec{w}_j} = \left(\vec{h}^{\ell-1}\right)^{T}$$

Jacobians of FC-Layer

Convolutional Neural Networks

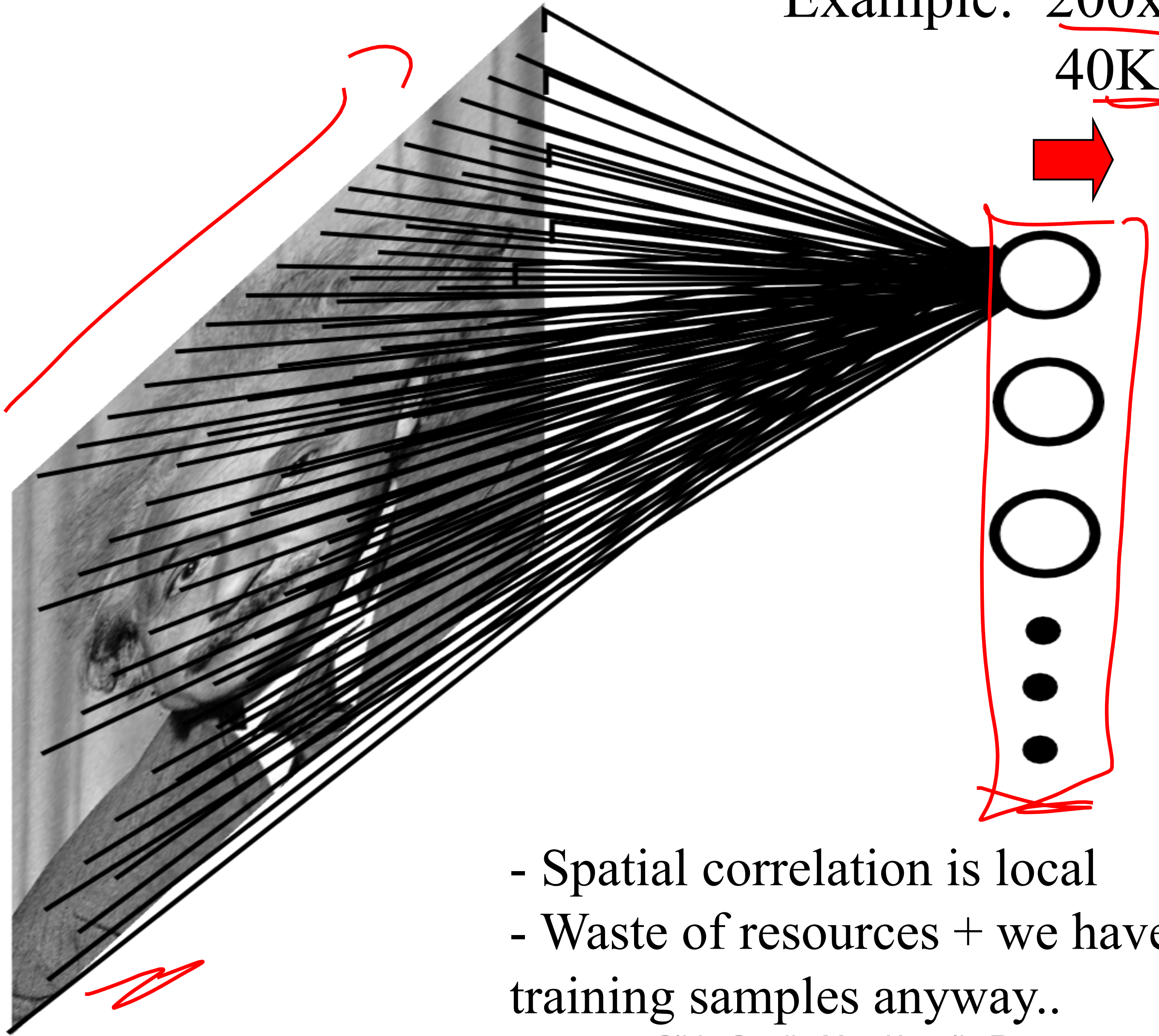
(without the brain stuff)

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

Fully Connected Layer

4×10^4

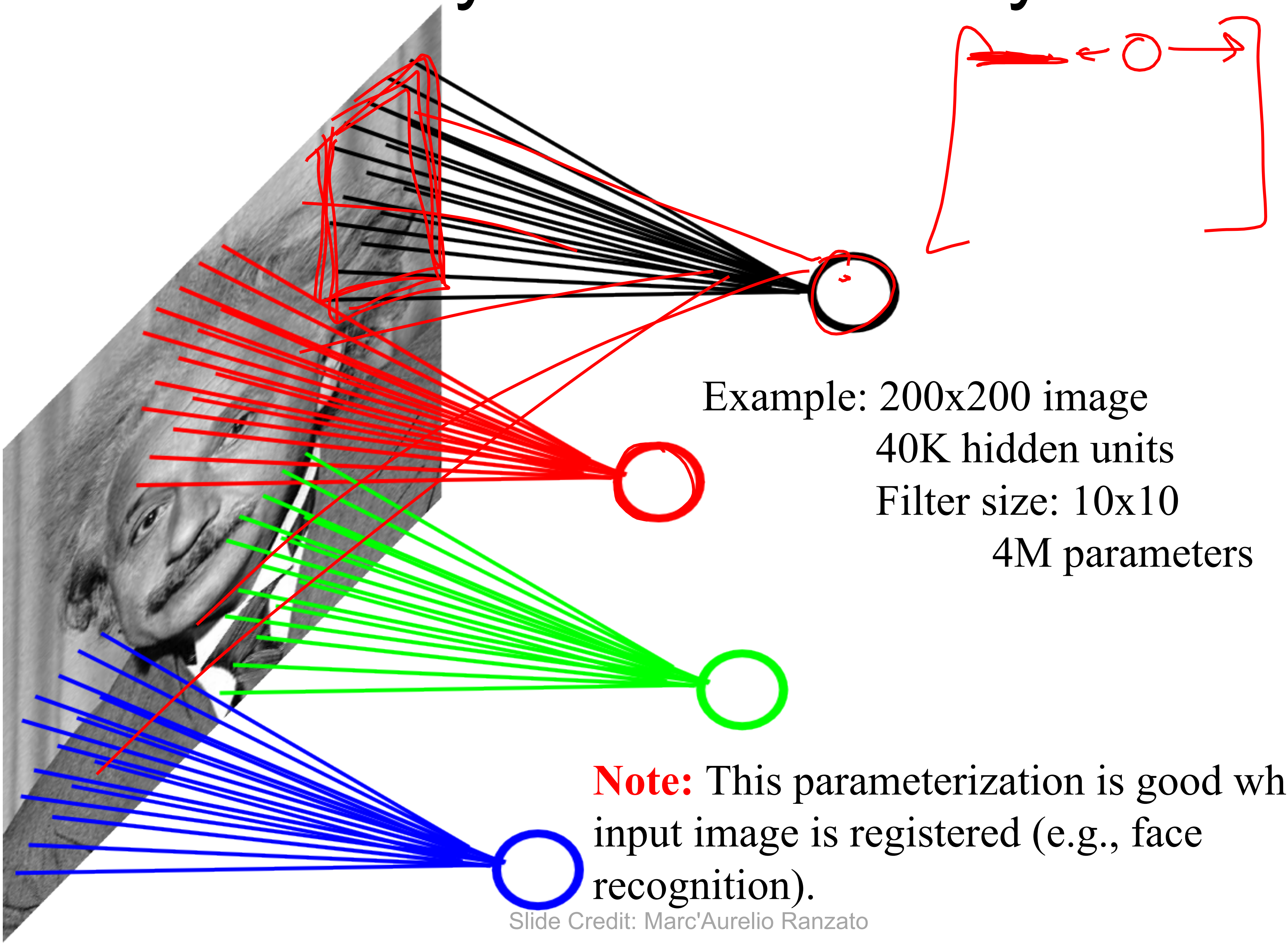
Example: 200x200 image
40K hidden units
→ **~2B parameters**!!!



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

Slide Credit: Marc'Aurelio Ranzato

Locally Connected Layer

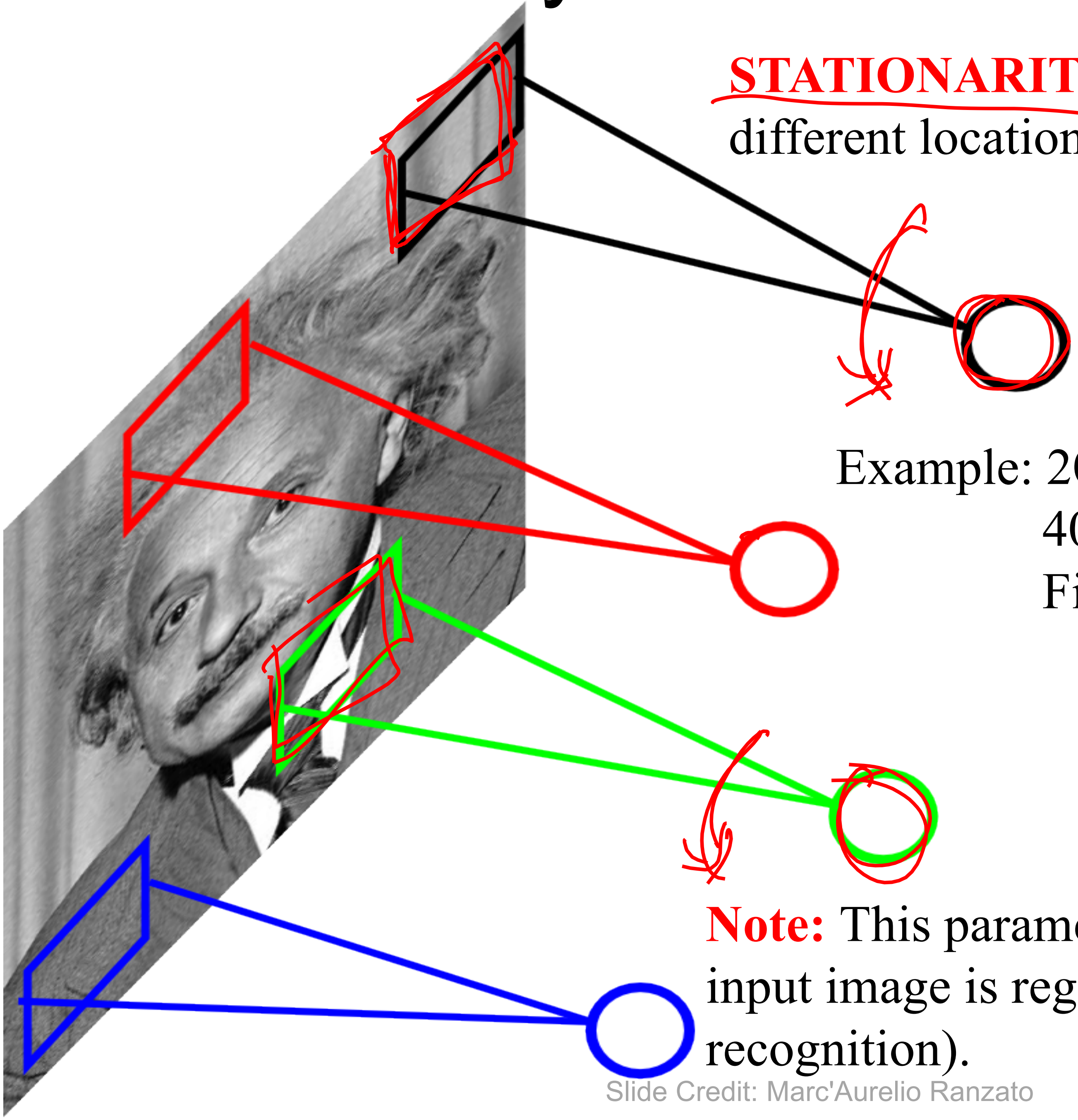


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

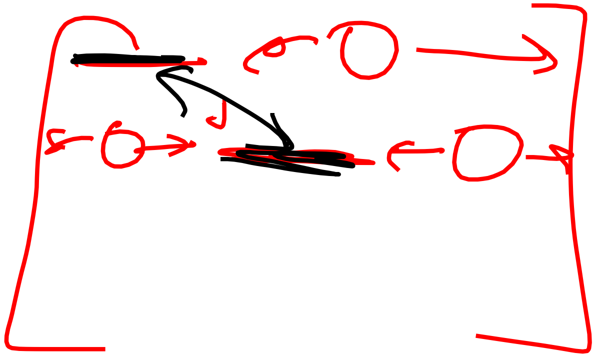
Note: This parameterization is good when input image is registered (e.g., face recognition).

Slide Credit: Marc'Aurelio Ranzato

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

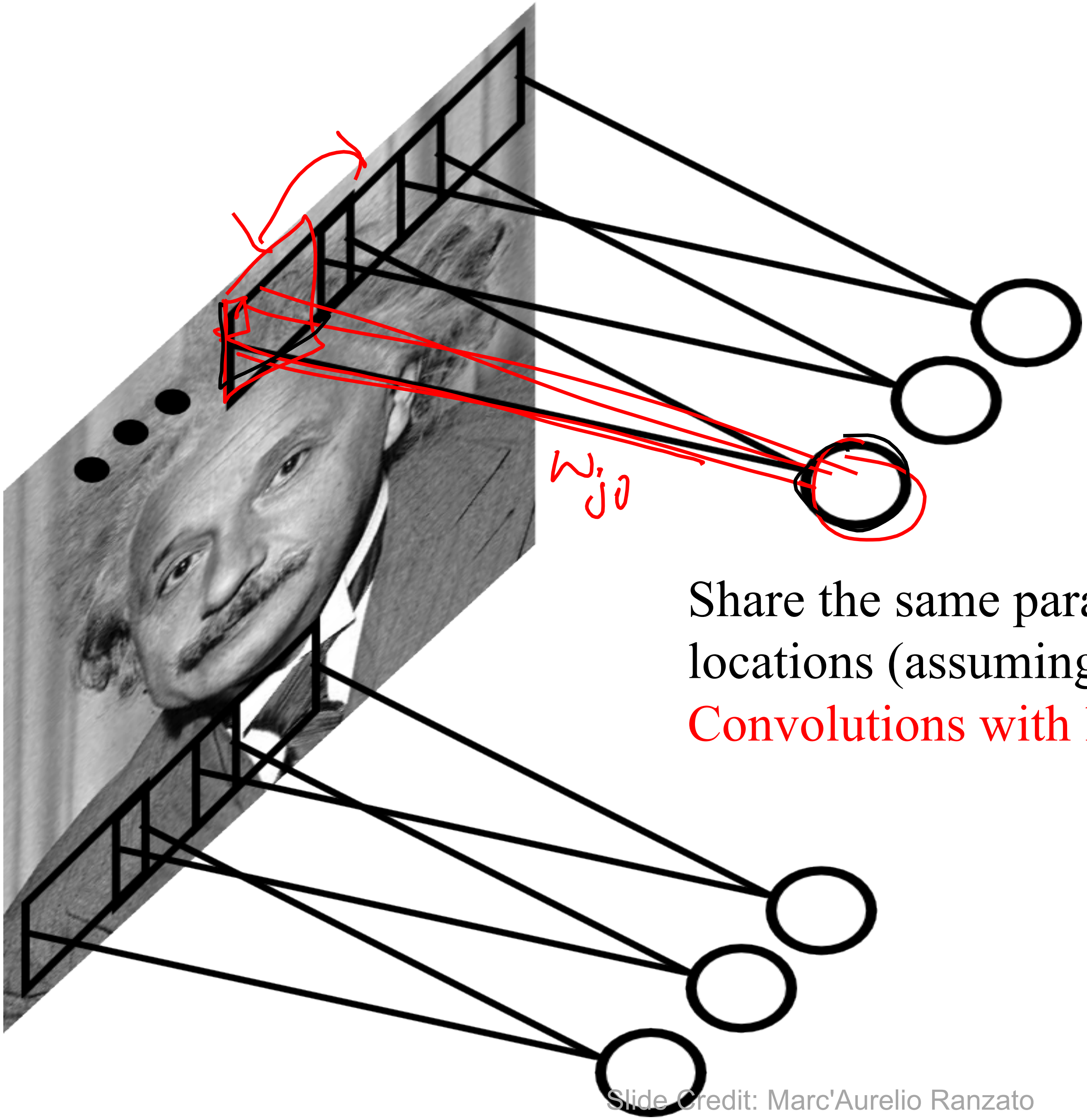


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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

Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



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

Slide Credit: Marc'Aurelio Ranzato

Convolutions for mathematicians

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

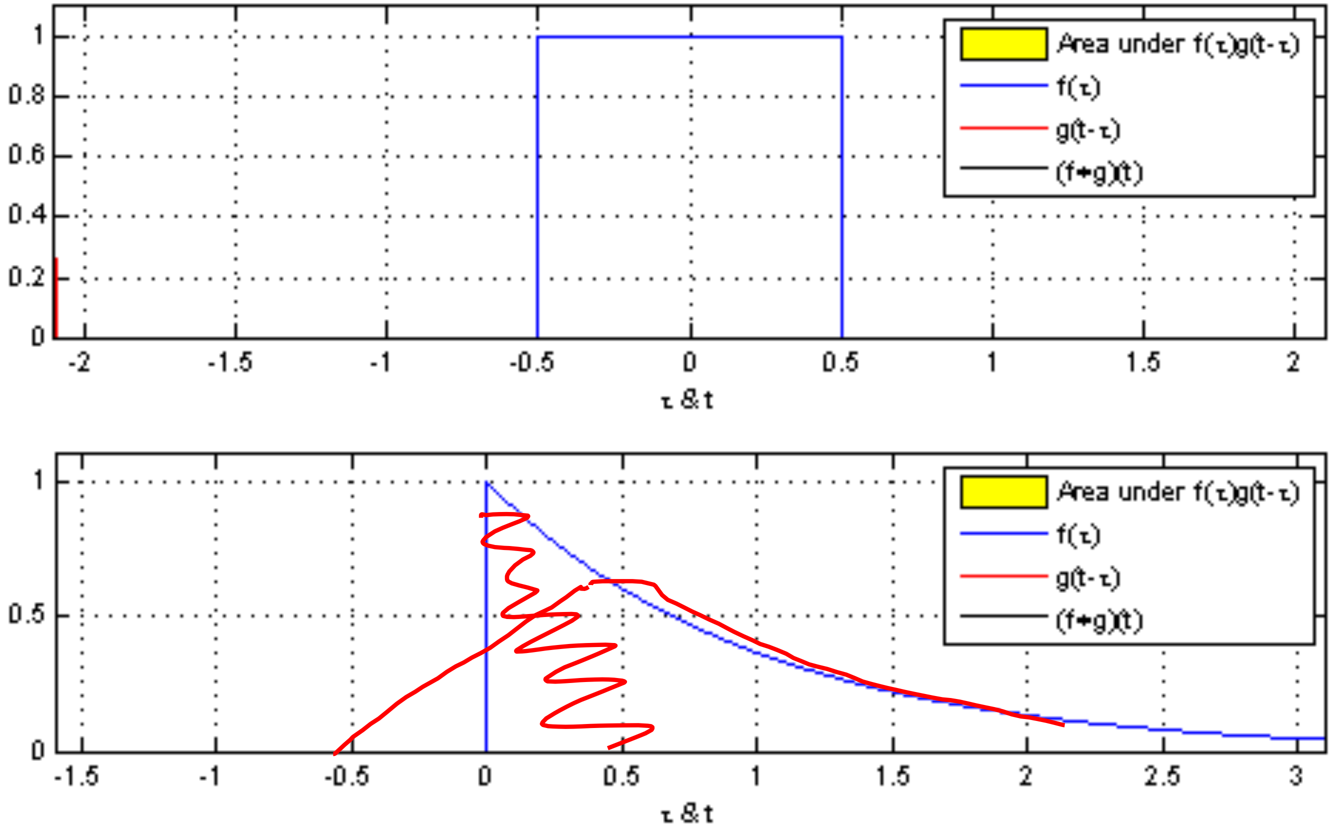
$$y(t) = (x * w)(t) = \int_{a=-\infty}^{\infty} \underline{x(t-a)}\, w(a)\, da$$

$$= \int \underline{x(a)}\; \underline{w(t-a)}\, da$$

$w(a) \rightarrow w(-a)$

$w(-a) \rightarrow w(-(a-t))$

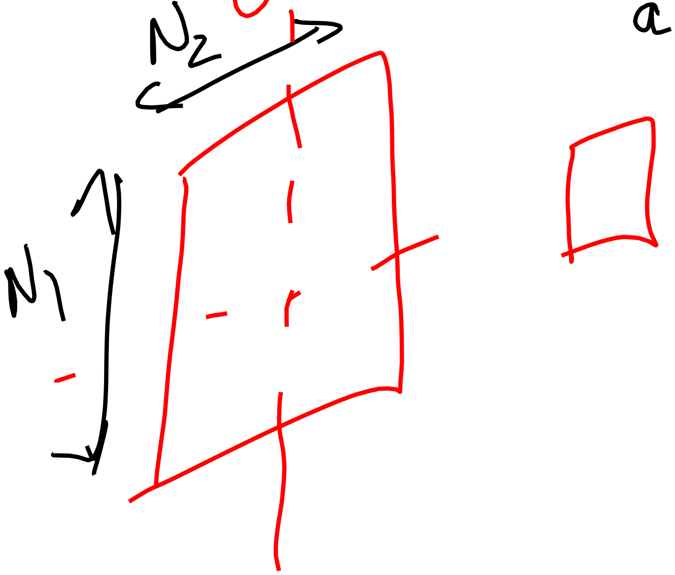
$$y(t_1, t_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(a-t_1, b-t_2)\, w(a,b)\, da\, db$$



"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

$$y[r, c] = \sum_{a=-\frac{N_1-1}{2}}^{\frac{N-1}{2}} \sum_{b=-\frac{N-1}{2}}^{+\frac{N-1}{2}} x[\underline{r-a}, \underline{c-b}]\, w[a,b]$$



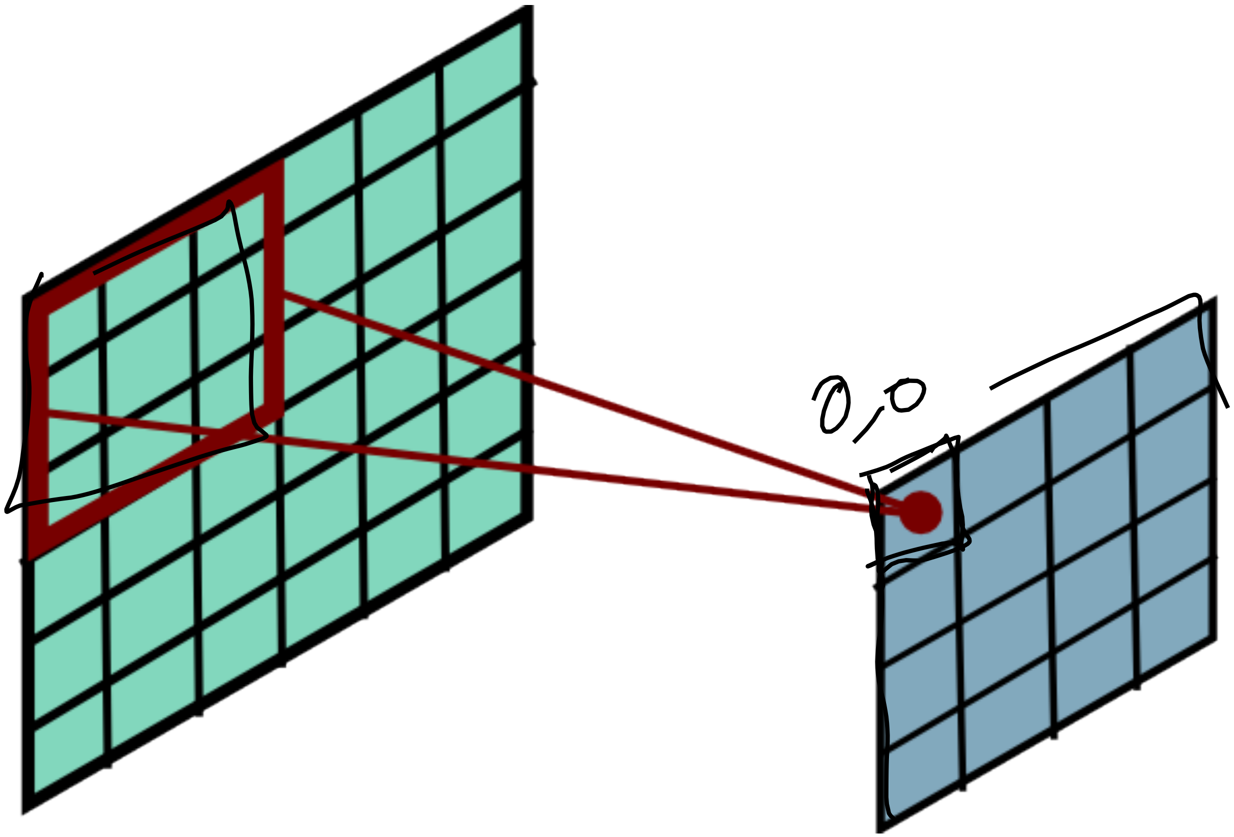
Convolutions for programmers

$$y[\underline{r}, \underline{c}] = \underbrace{\sum_{a=0}^{N_1-1} \sum_{b=0}^{N_2-1}}\, x[\underline{r}+a, \underline{c}+b]\, w[a,b]$$

Convolution Explained

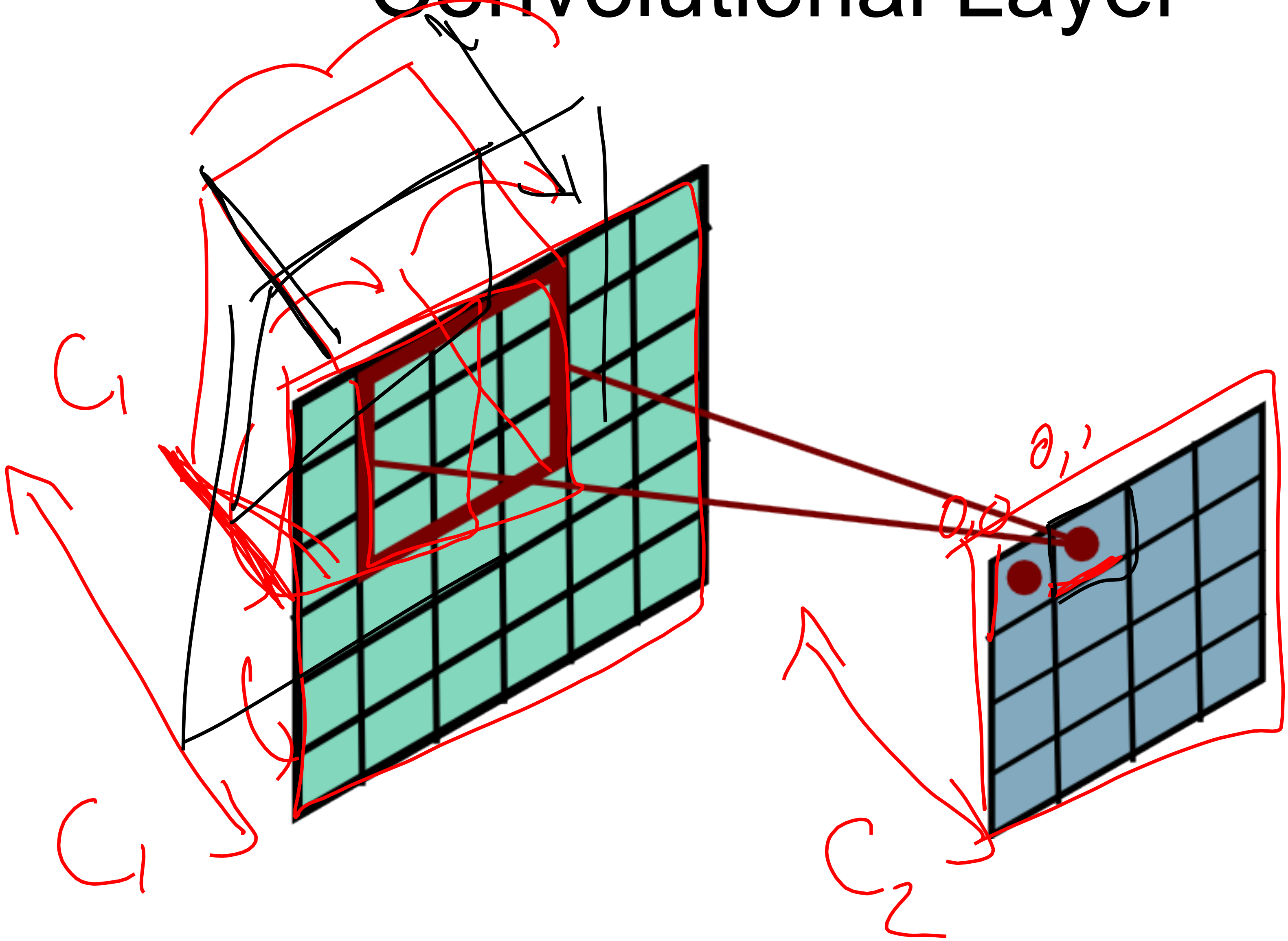
- http://setosa.io/ev/image-kernels/
- https://github.com/bruckner/deepViz

Convolutional Layer



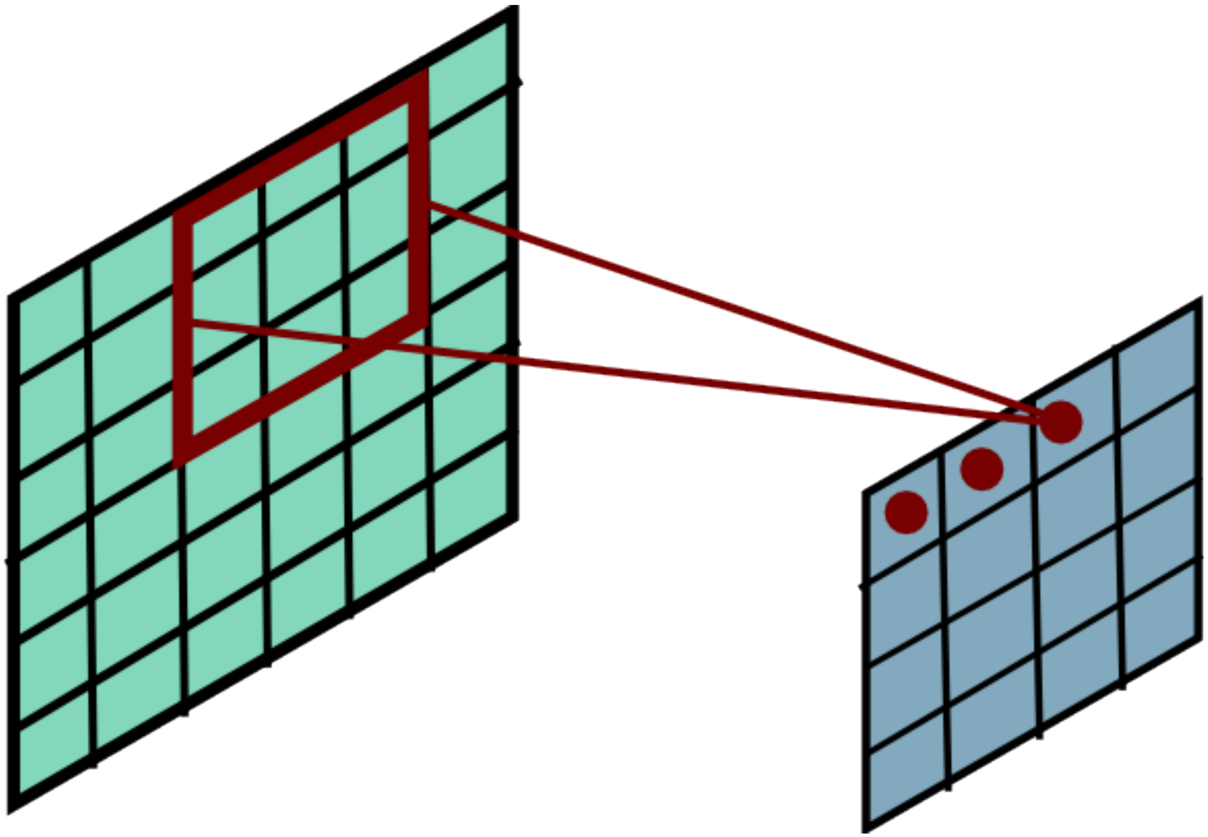
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



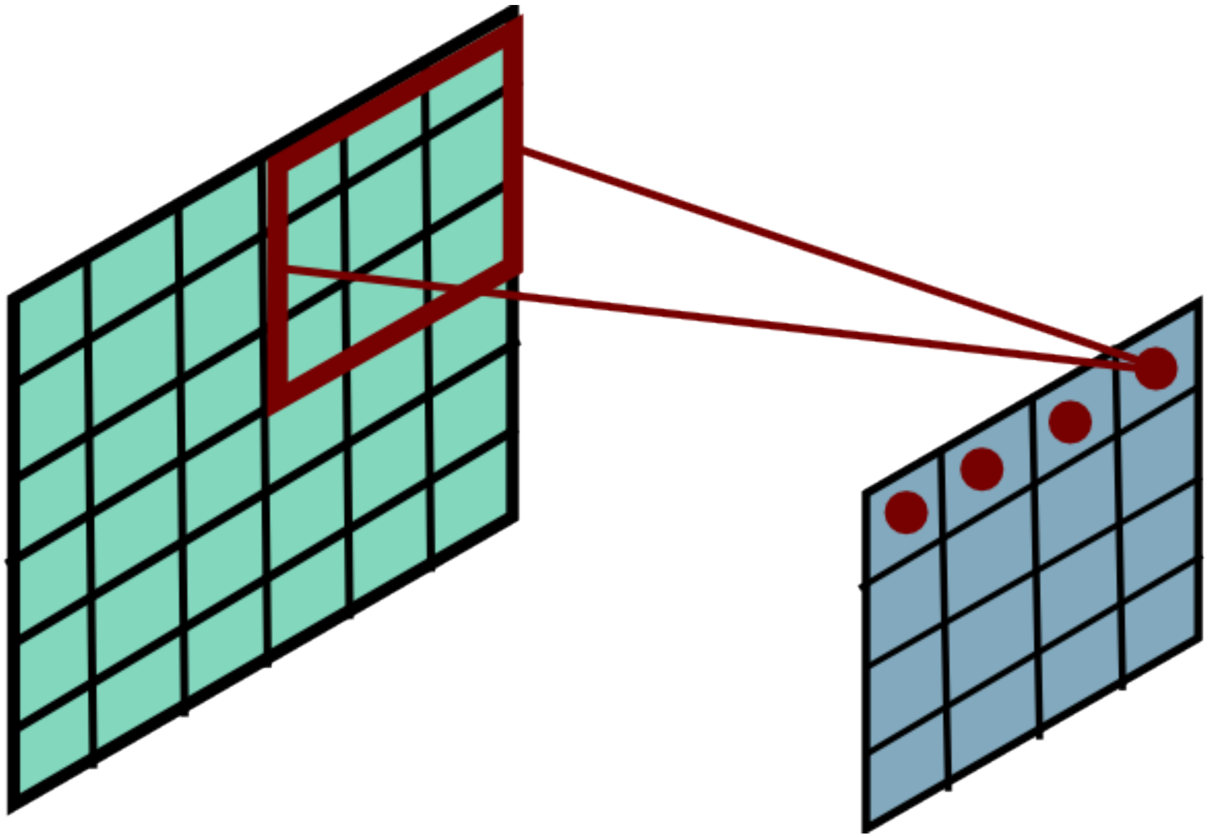
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



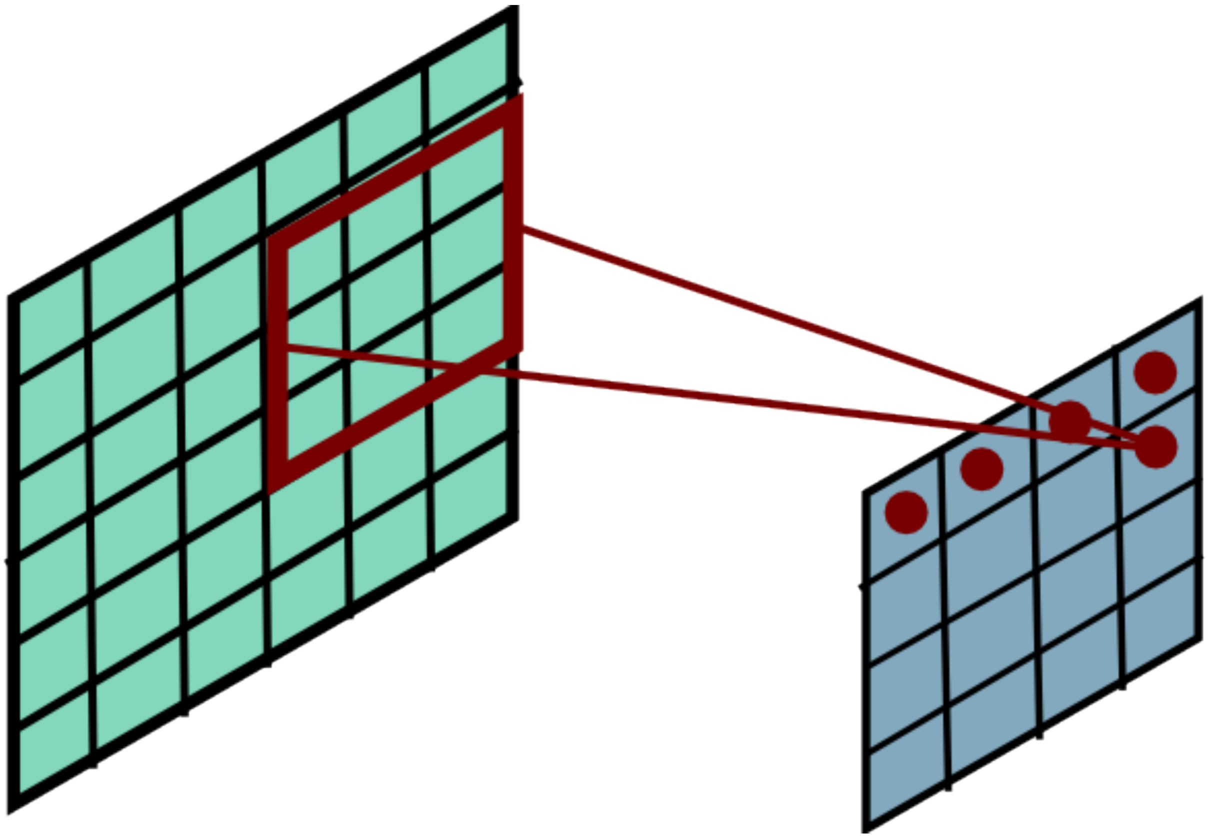
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



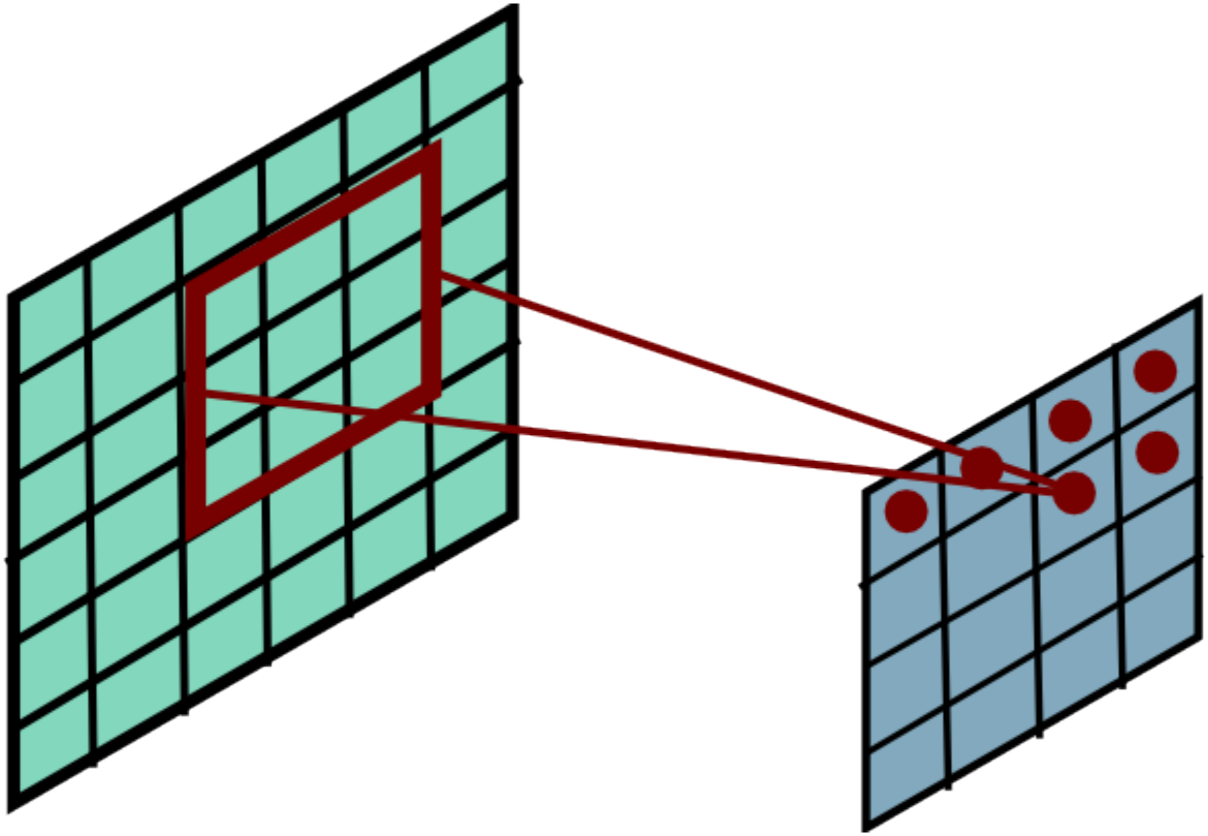
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



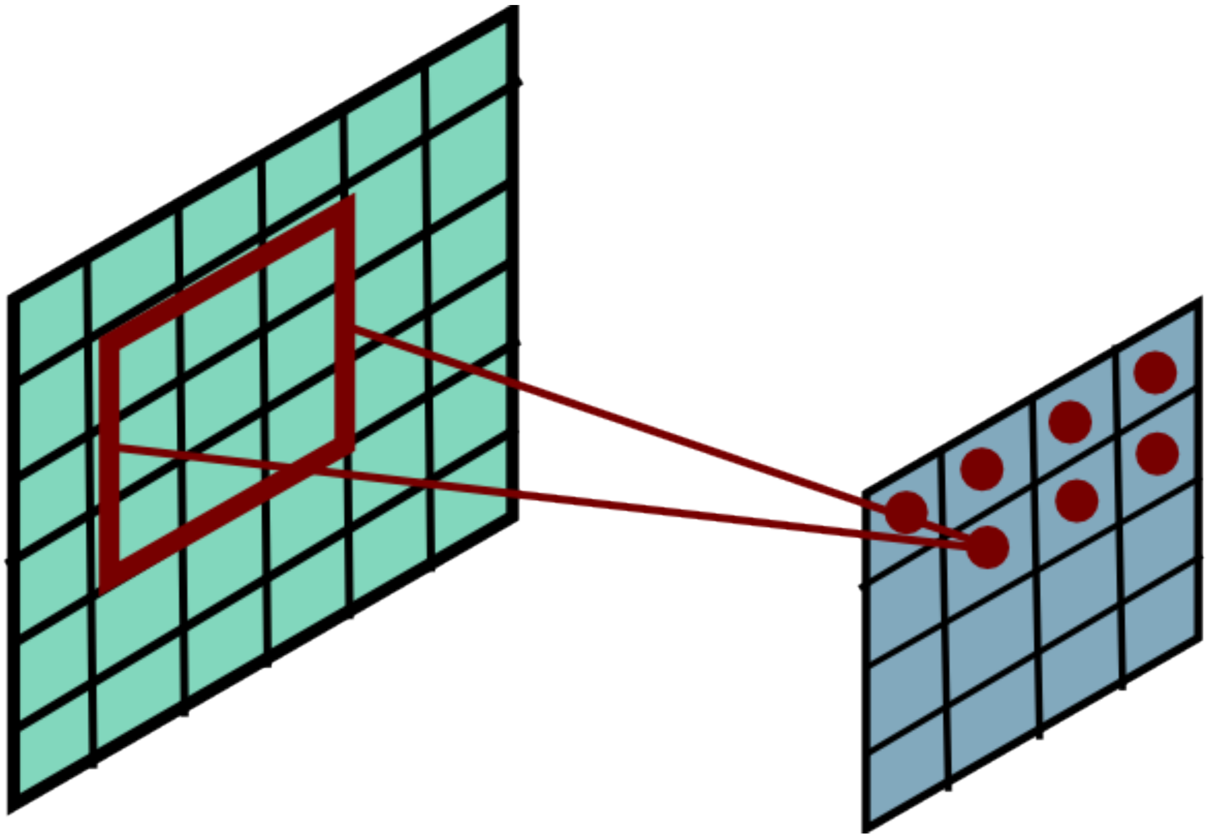
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



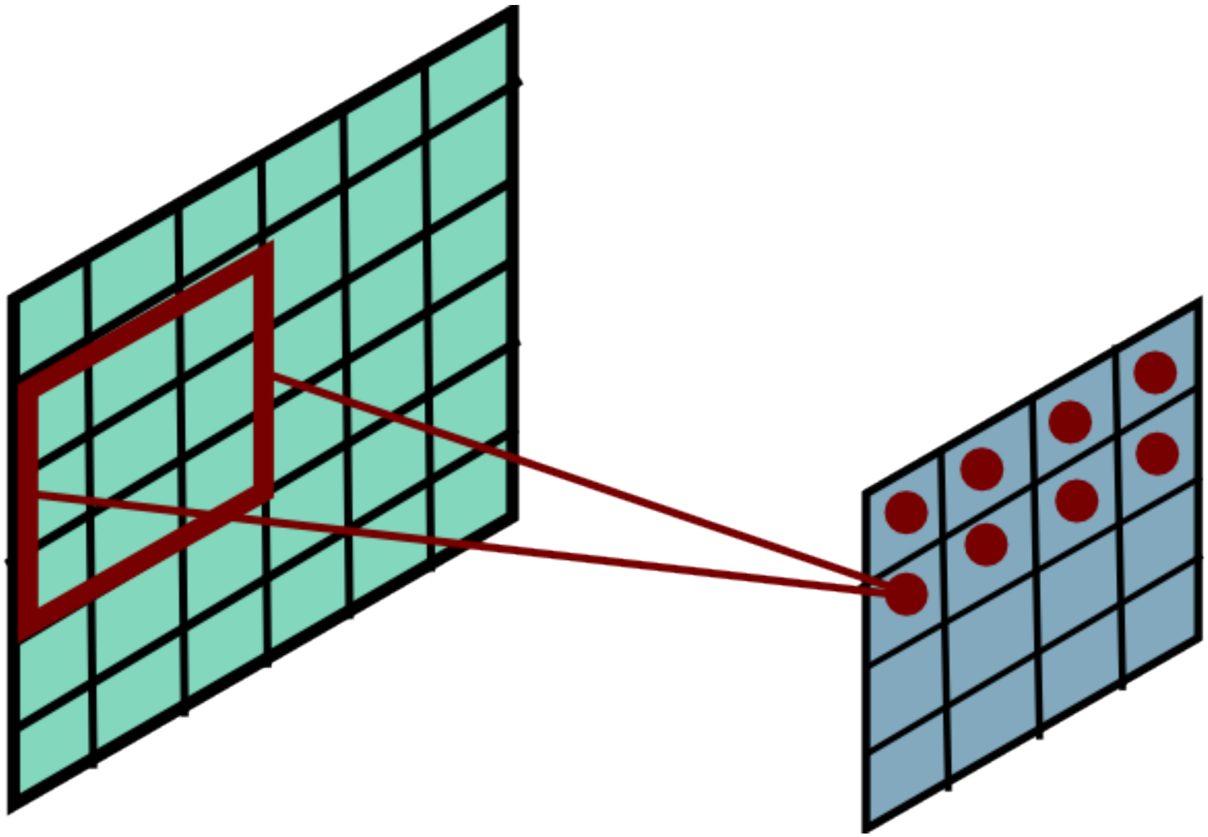
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



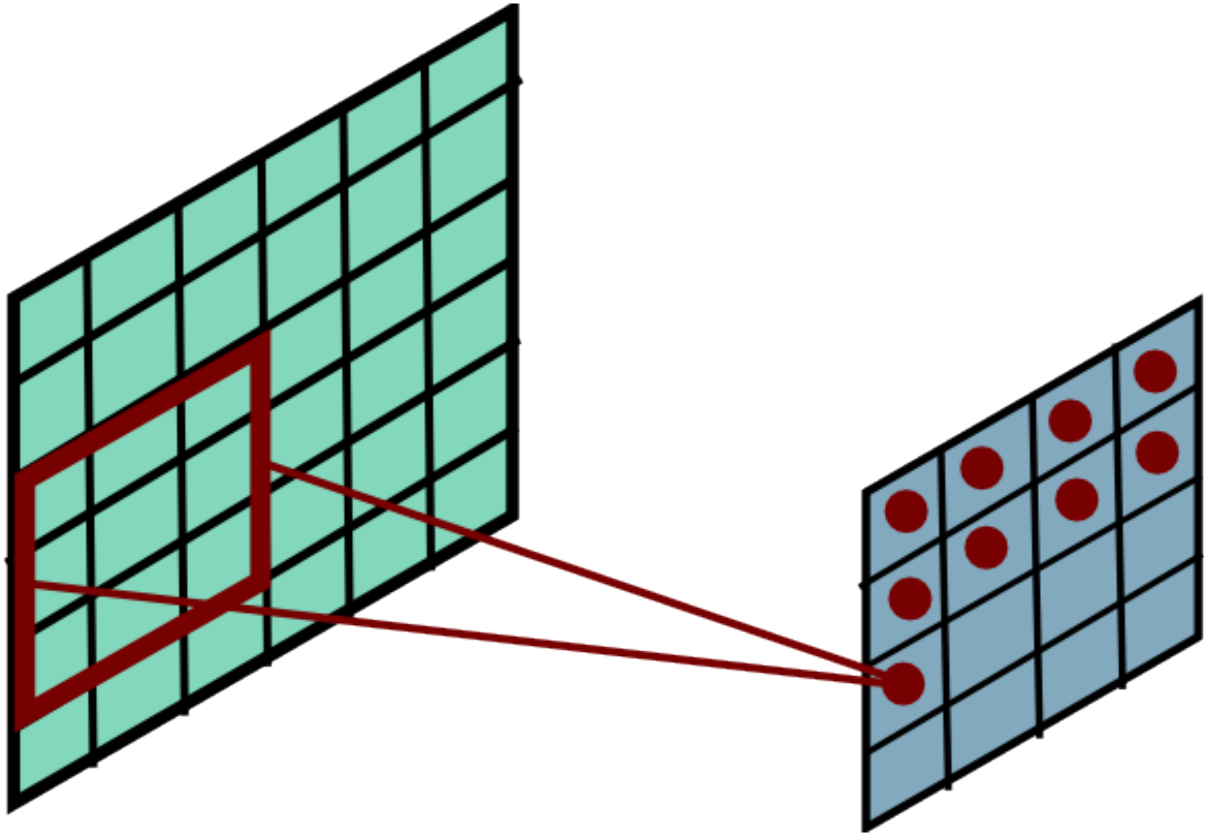
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

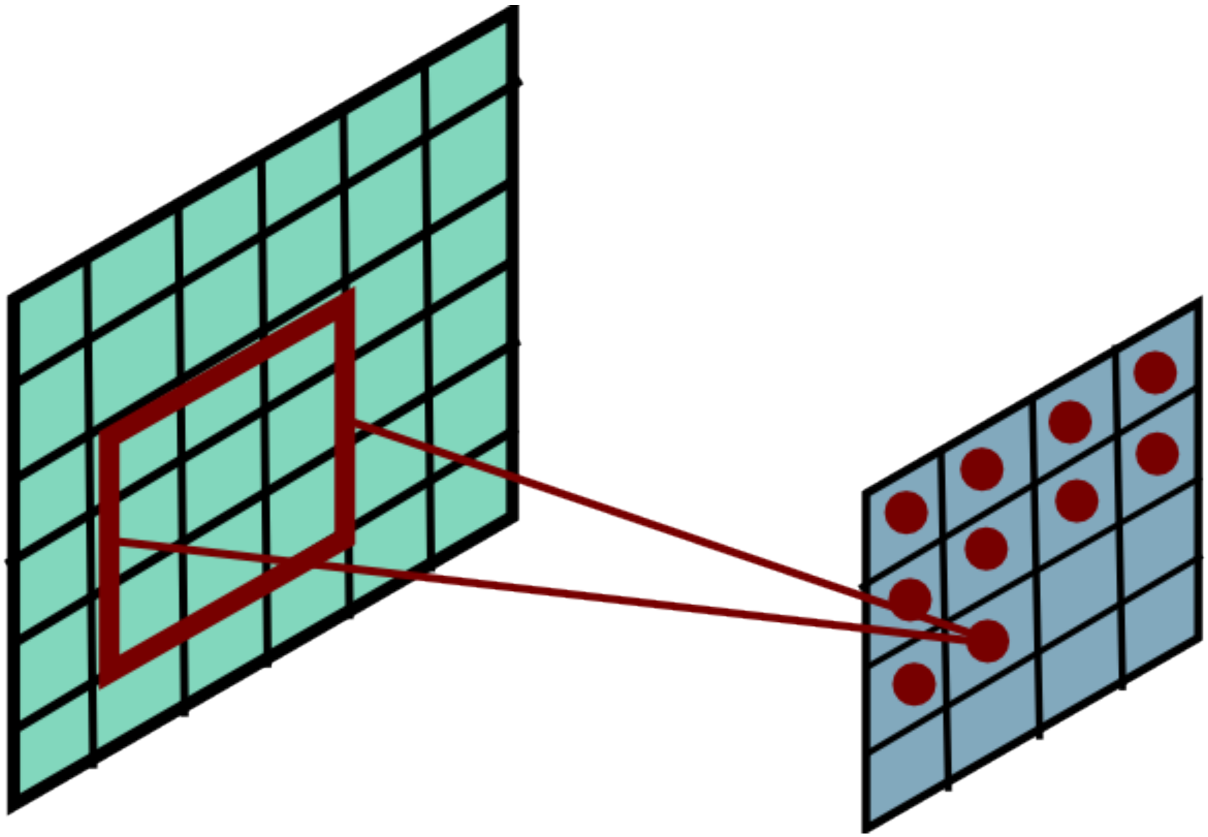
Convolutional Layer



(C) Dhruv Batra

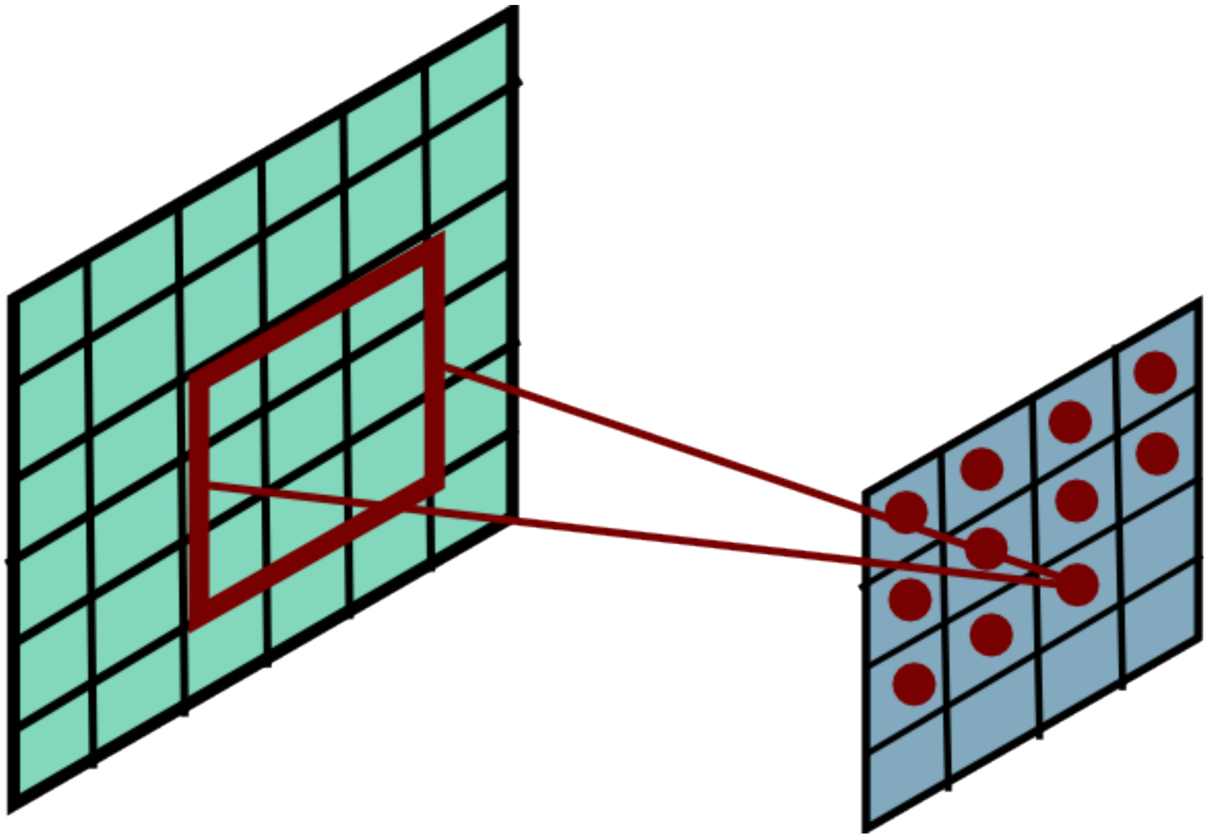
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

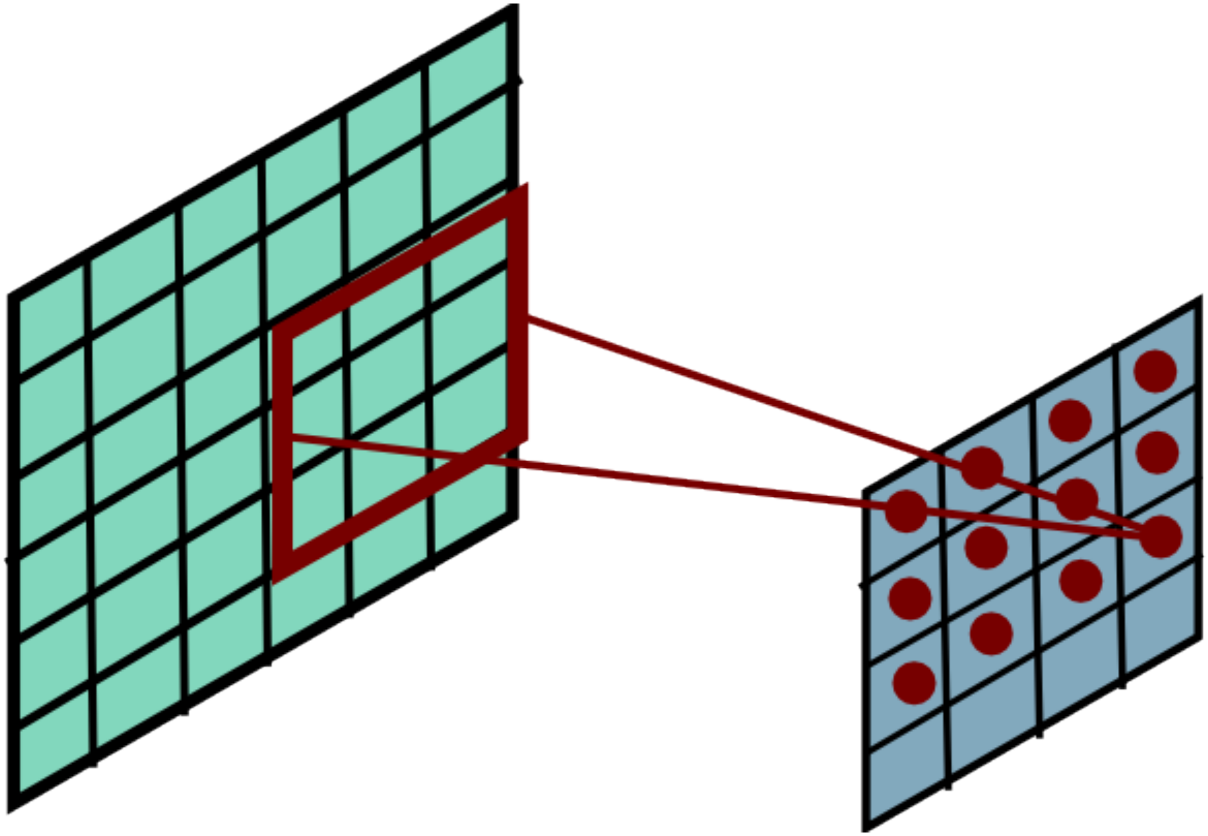
Convolutional Layer



(C) Dhruv Batra

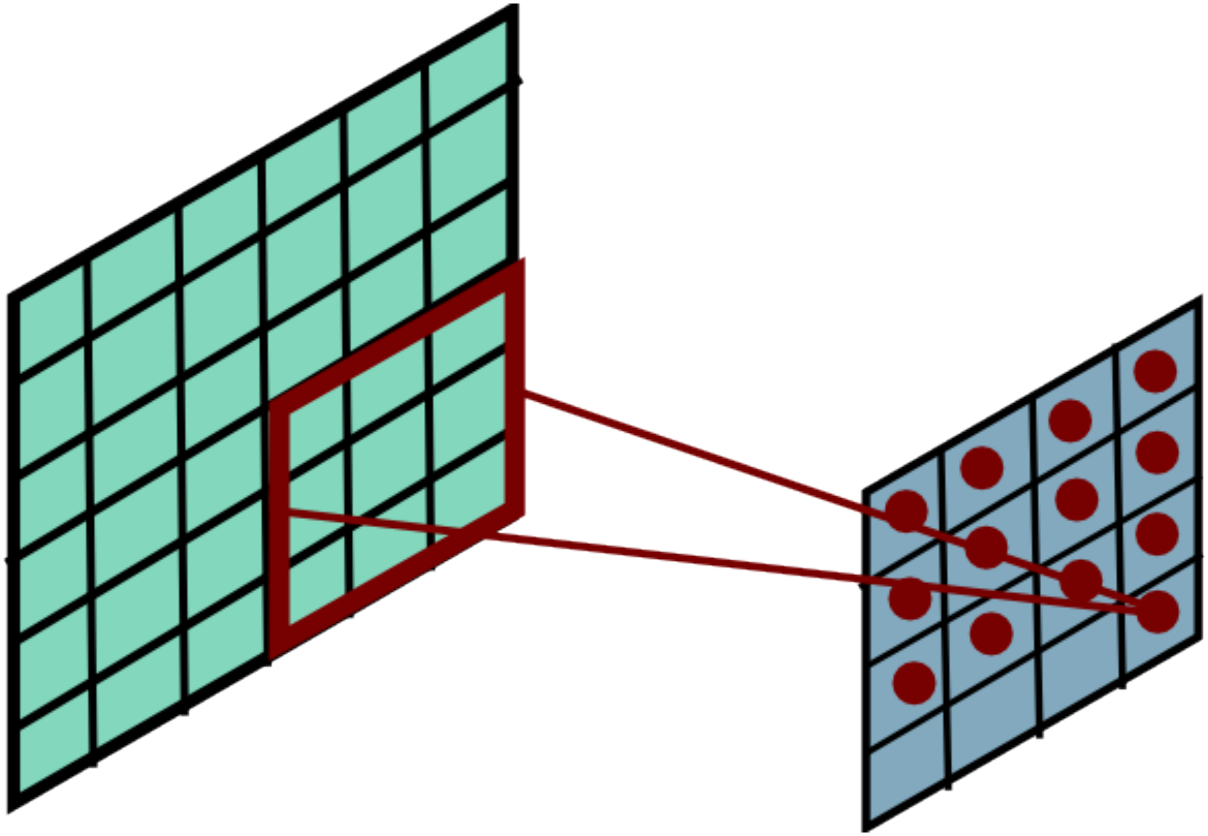
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



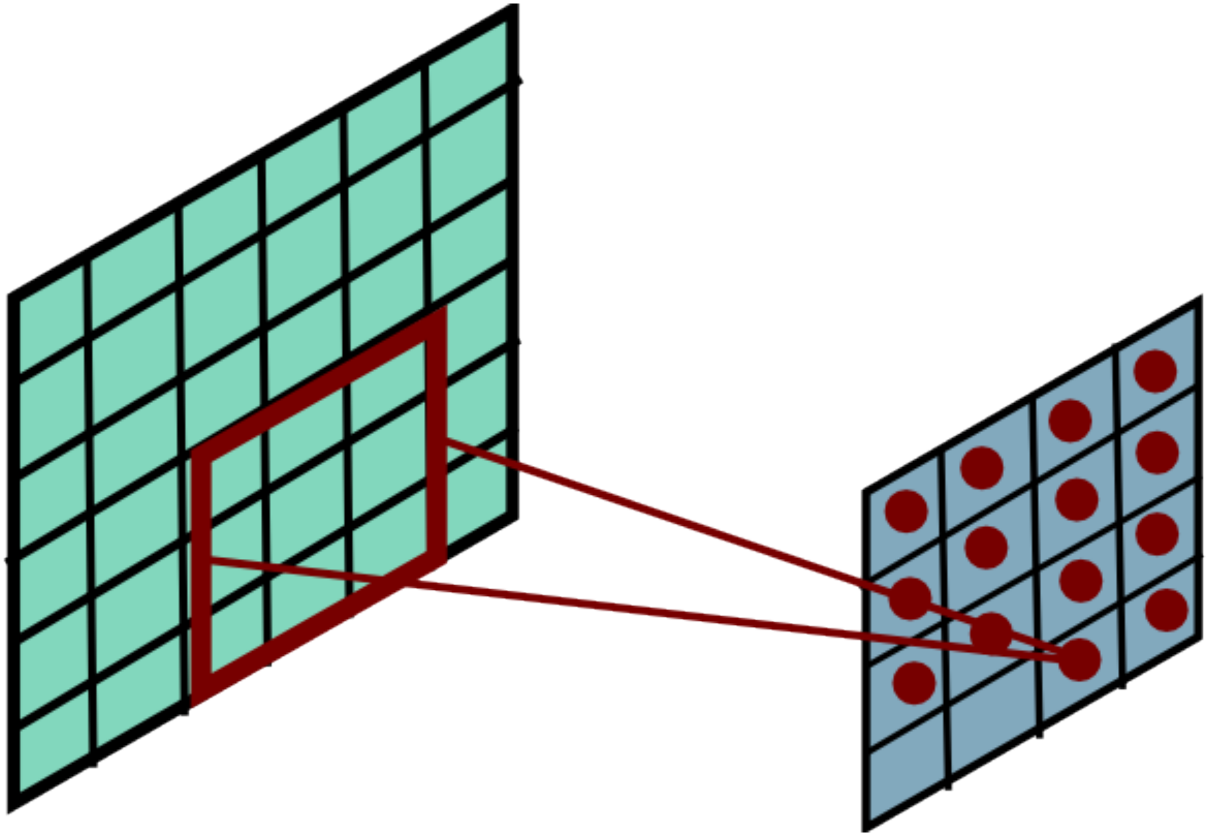
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



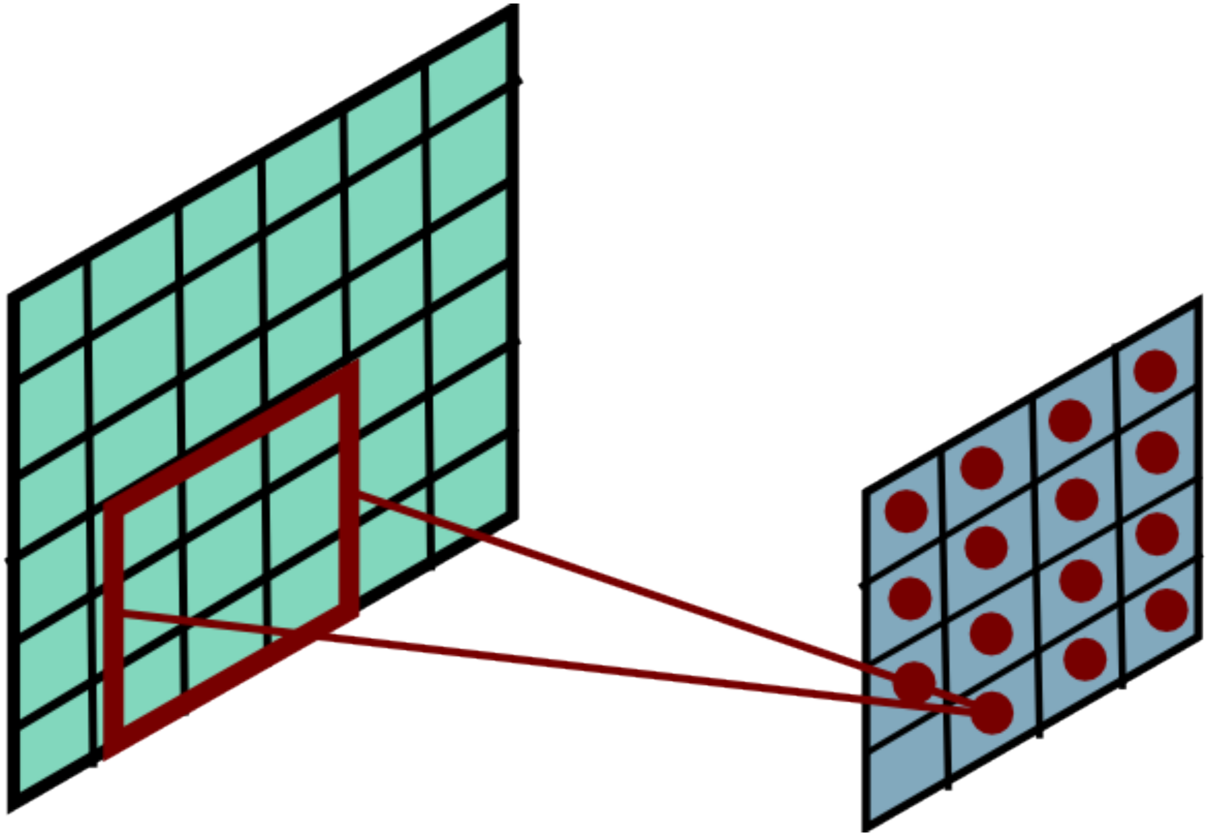
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



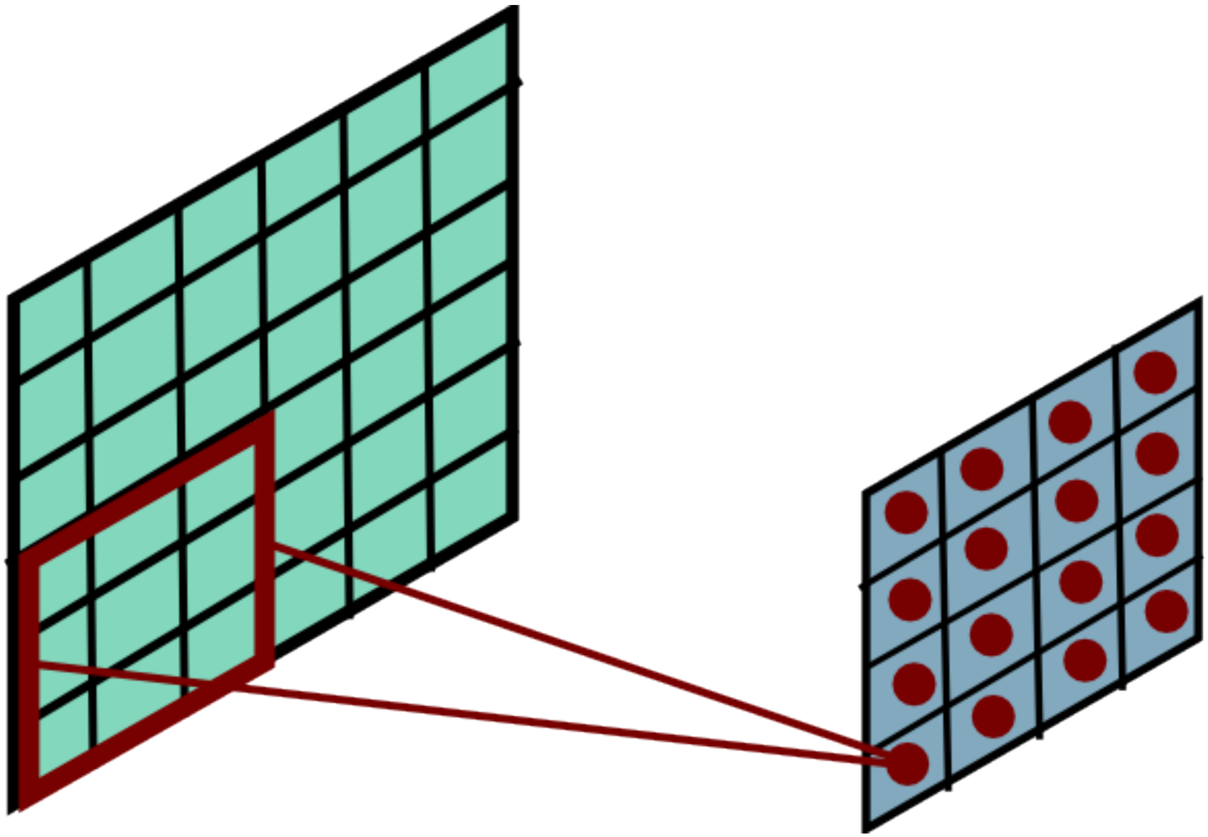
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

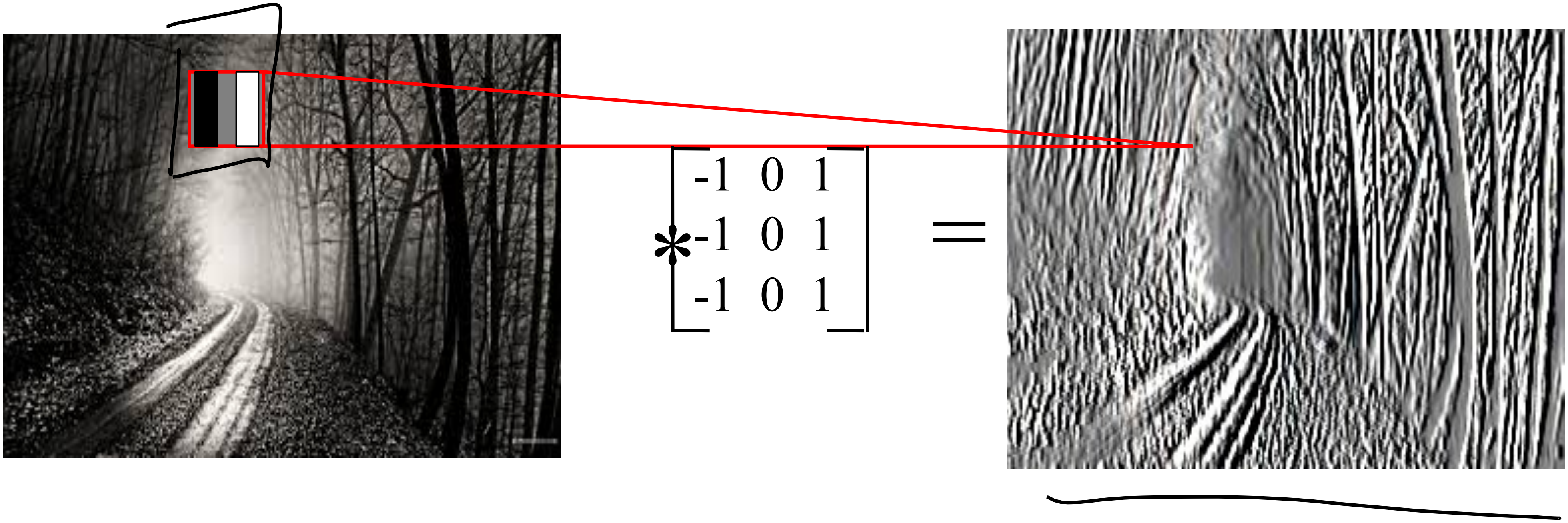
Convolutional Layer



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

Slide Credit: Marc'Aurelio Ranzato

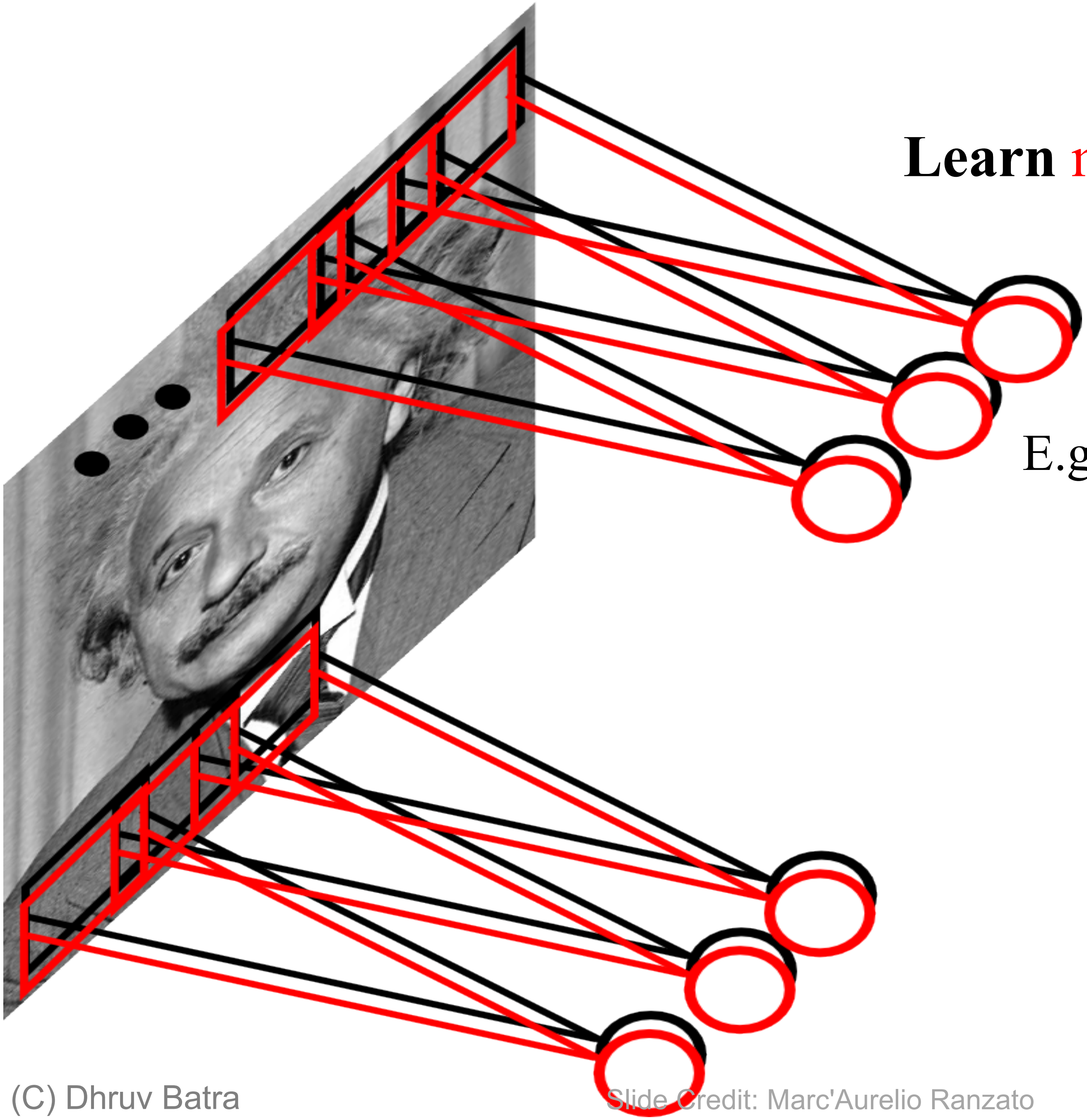
Convolutional Layer



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

Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



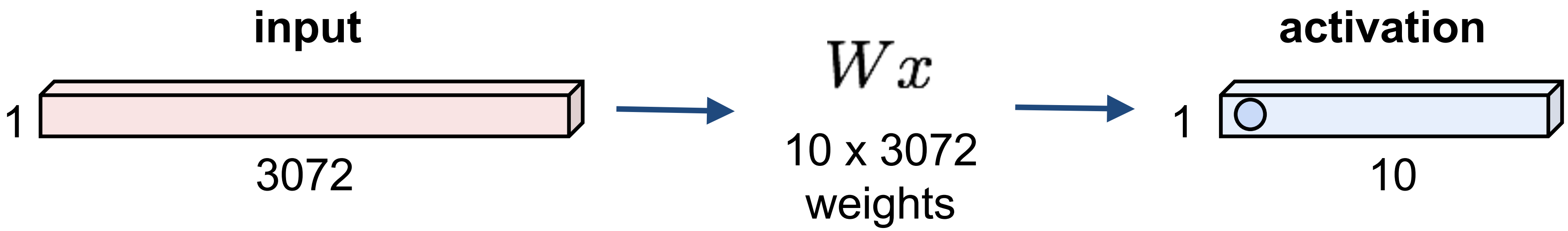
Learn multiple filters.

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

Slide Credit: Marc'Aurelio Ranzato

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



input

1

3072

Wx

10 x 3072
weights

activation

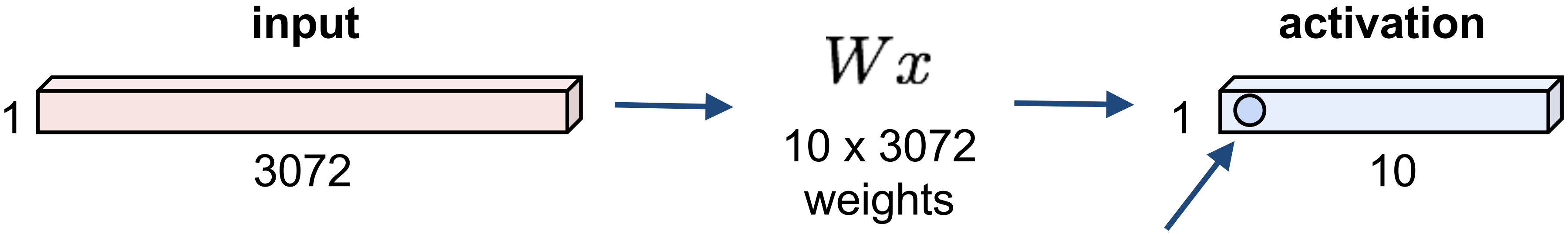
1

10

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

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



input

1

3072

Wx

10 x 3072 weights

activation

1

10

1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

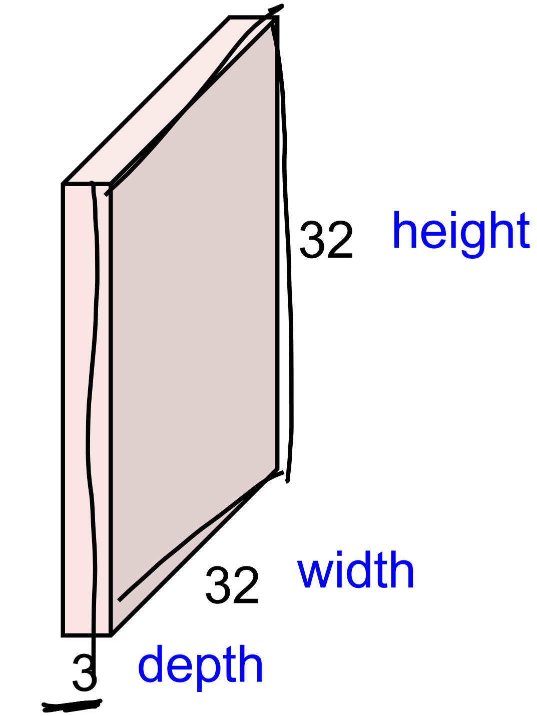
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Convolutional Layer

Convolutional Layer

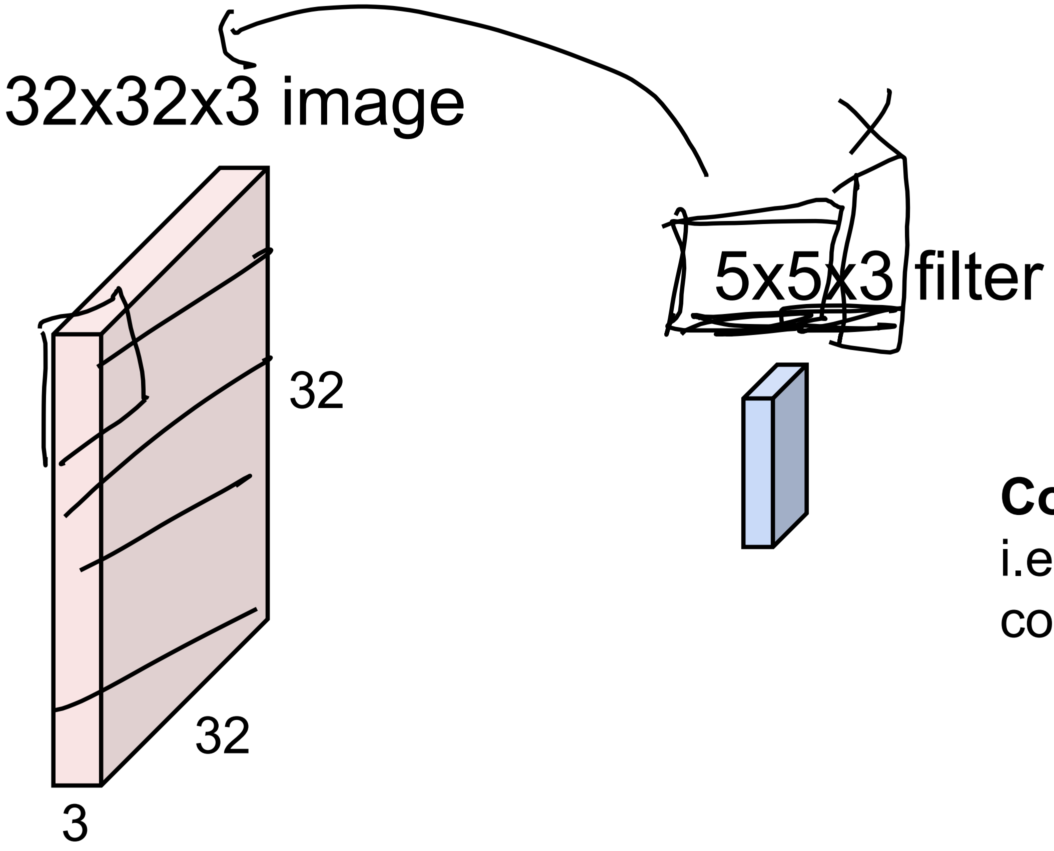
Convolution Layer

32x32x3 image -> preserve spatial structure



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

Convolution Layer



32x32x3 image

5x5x3 filter

32

32

3

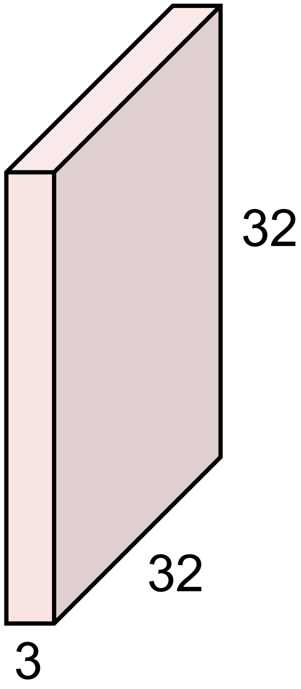
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

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

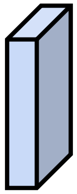
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



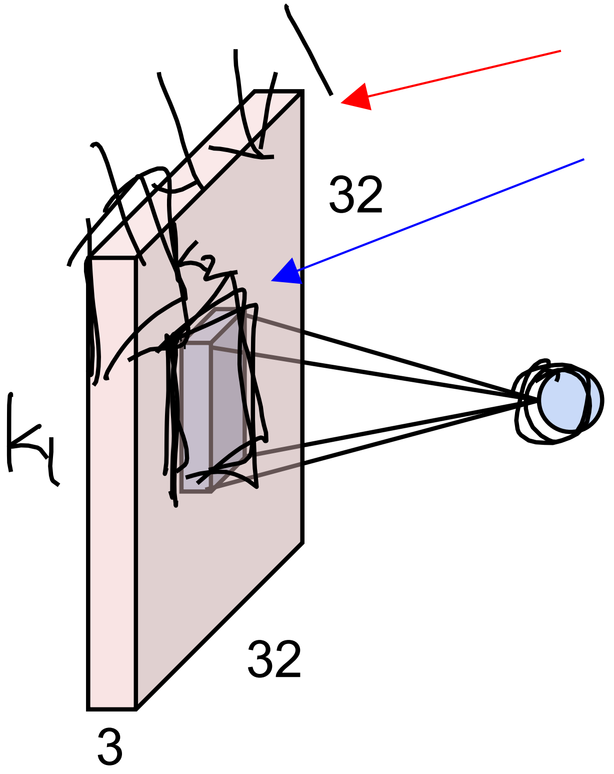
5x5x3 filter



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

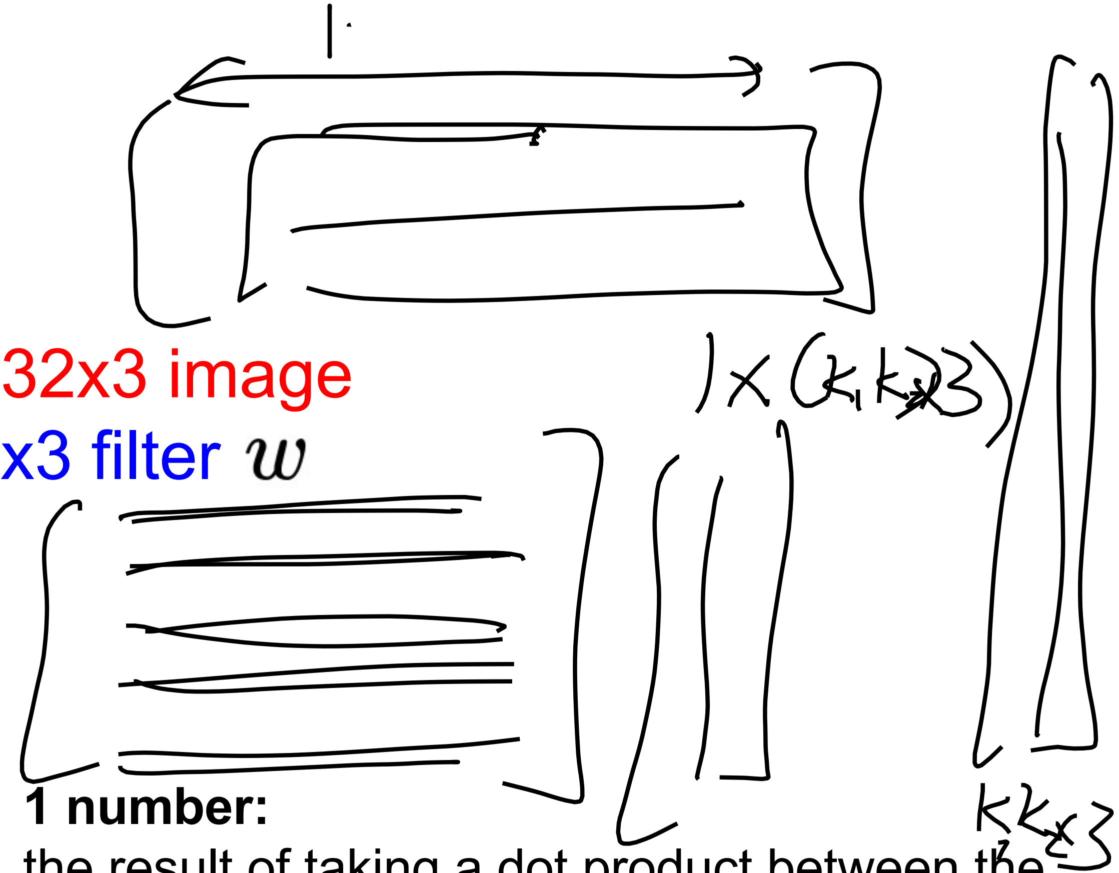
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Convolution Layer



32x32x3 image

5x5x3 filter w

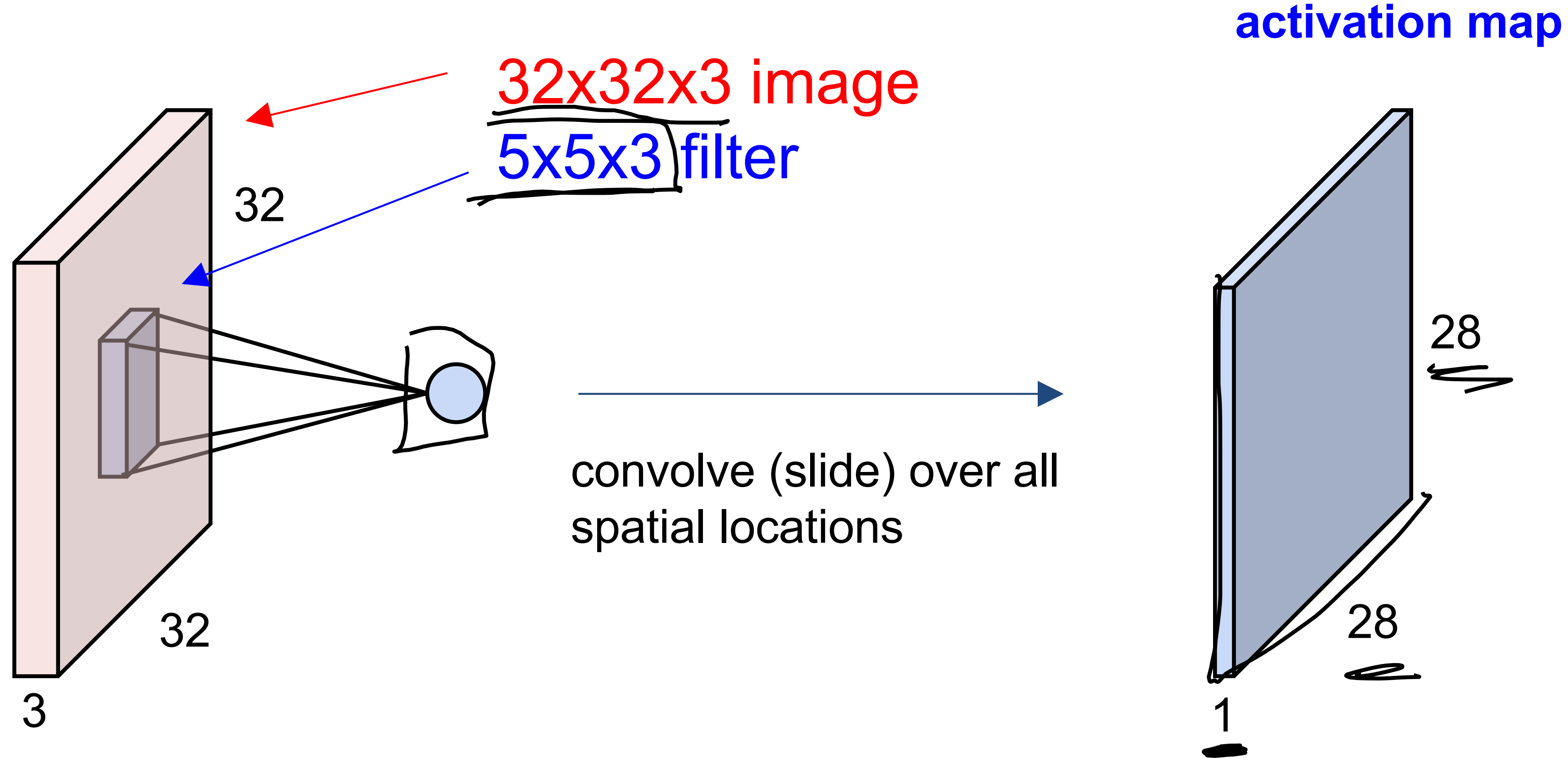


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

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

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

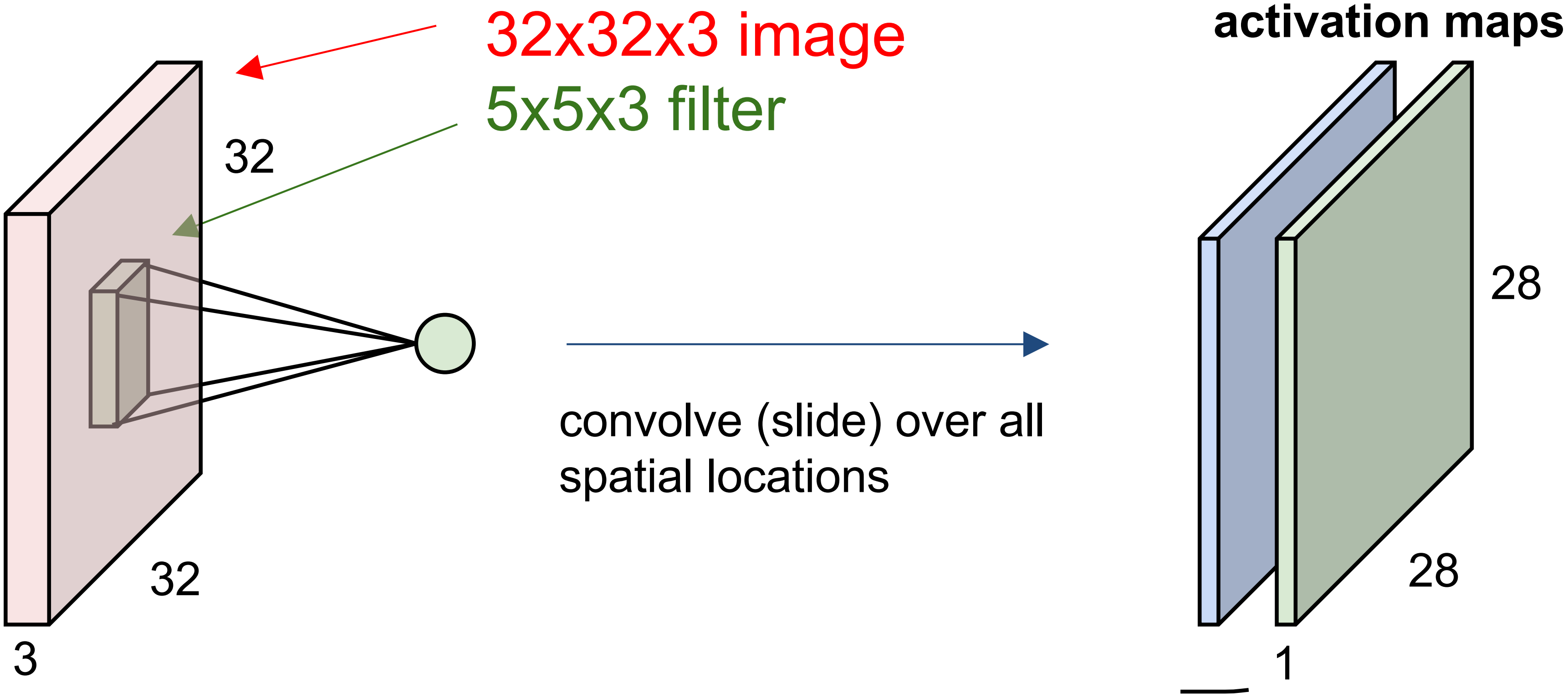
Convolution Layer



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

Convolution Layer

consider a second, green filter



32x32x3 image

5x5x3 filter

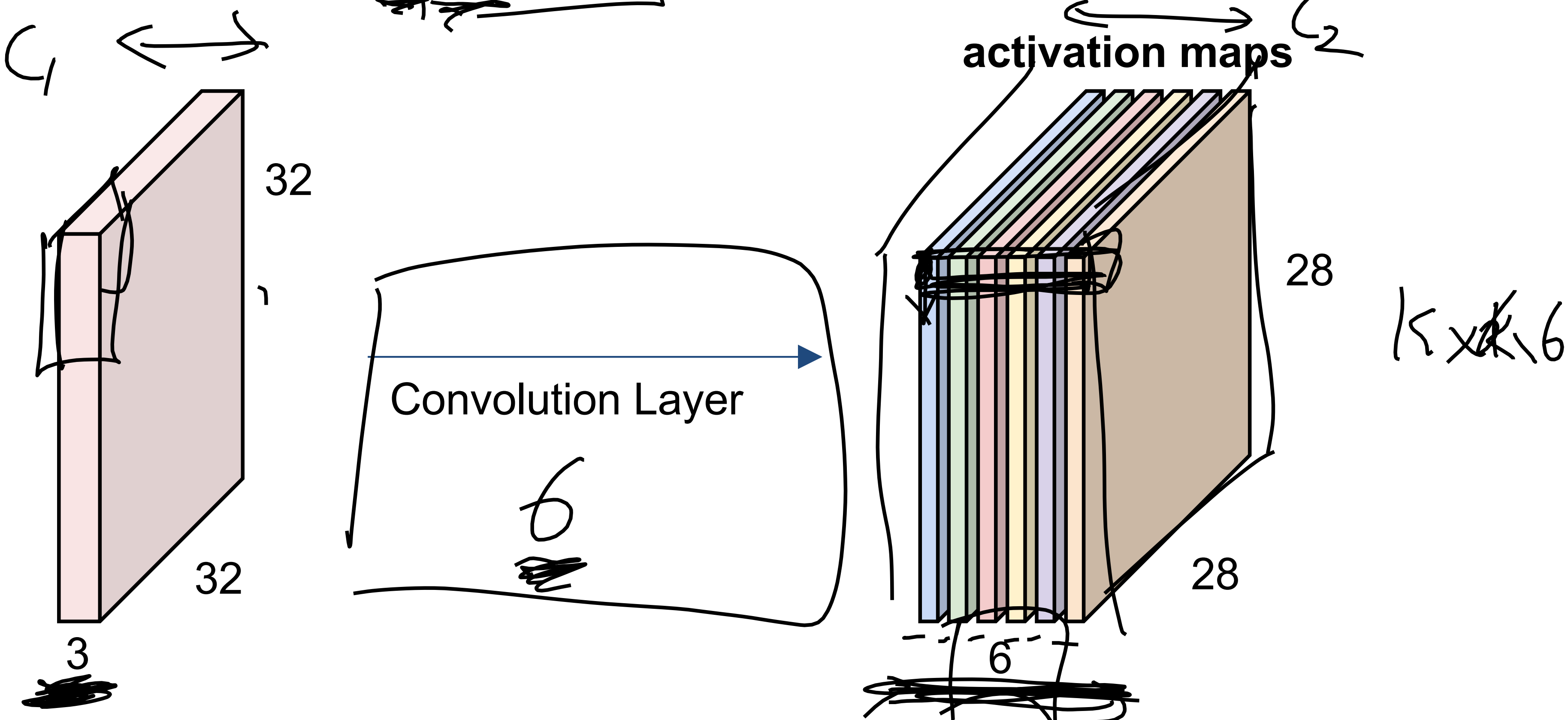
convolve (slide) over all spatial locations

activation maps

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

5x5x3

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!

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