CS 4803 / 7643: Deep Learning

Topics:

- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Dhruv Batra

Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 09/26, 11:55pm

- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8I5Zd3yOFs7E/edit?usp=sharing
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names

Recap from last time

Deep Learning = Differentiable Programming

- Computation = Graph
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering

- Auto-Diff
  - A family of algorithms for implementing chain-rule on computation graphs

Forward mode AD

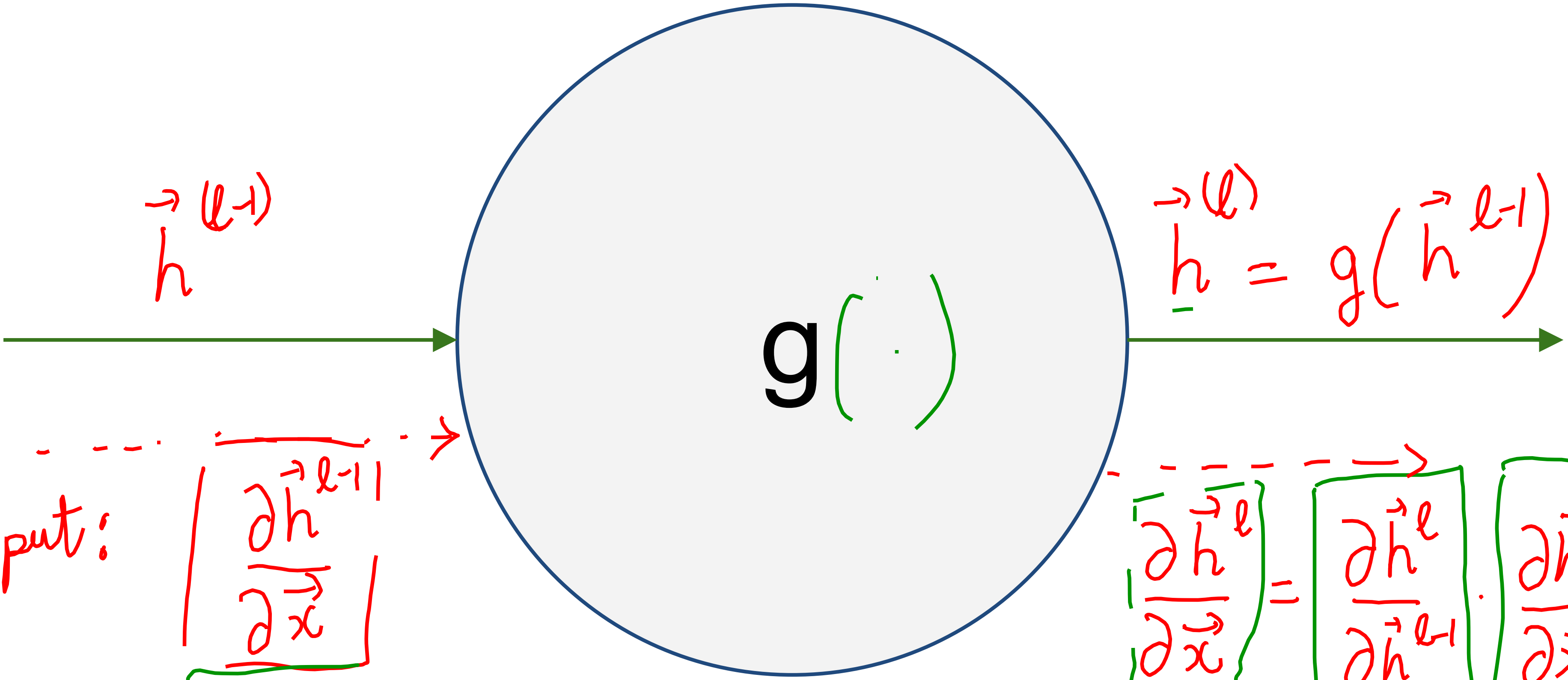
Goal: $\frac{\partial L}{\partial \vec{x}}$

$\vec{W}$

$\vec{x}$

L

layer ℓ



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

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

Input: $\boxed{\frac{\partial \vec{h}^{\ell-1}}{\partial \vec{x}}}$

$$\boxed{\frac{\partial \vec{h}^{\ell}}{\partial \vec{x}}} = \underbrace{\boxed{\frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}}}}_{\text{Jacobian of } g} \cdot \underbrace{\boxed{\frac{\partial \vec{h}^{\ell-1}}{\partial \vec{x}}}}_{\text{Input}}$$

Reverse mode AD

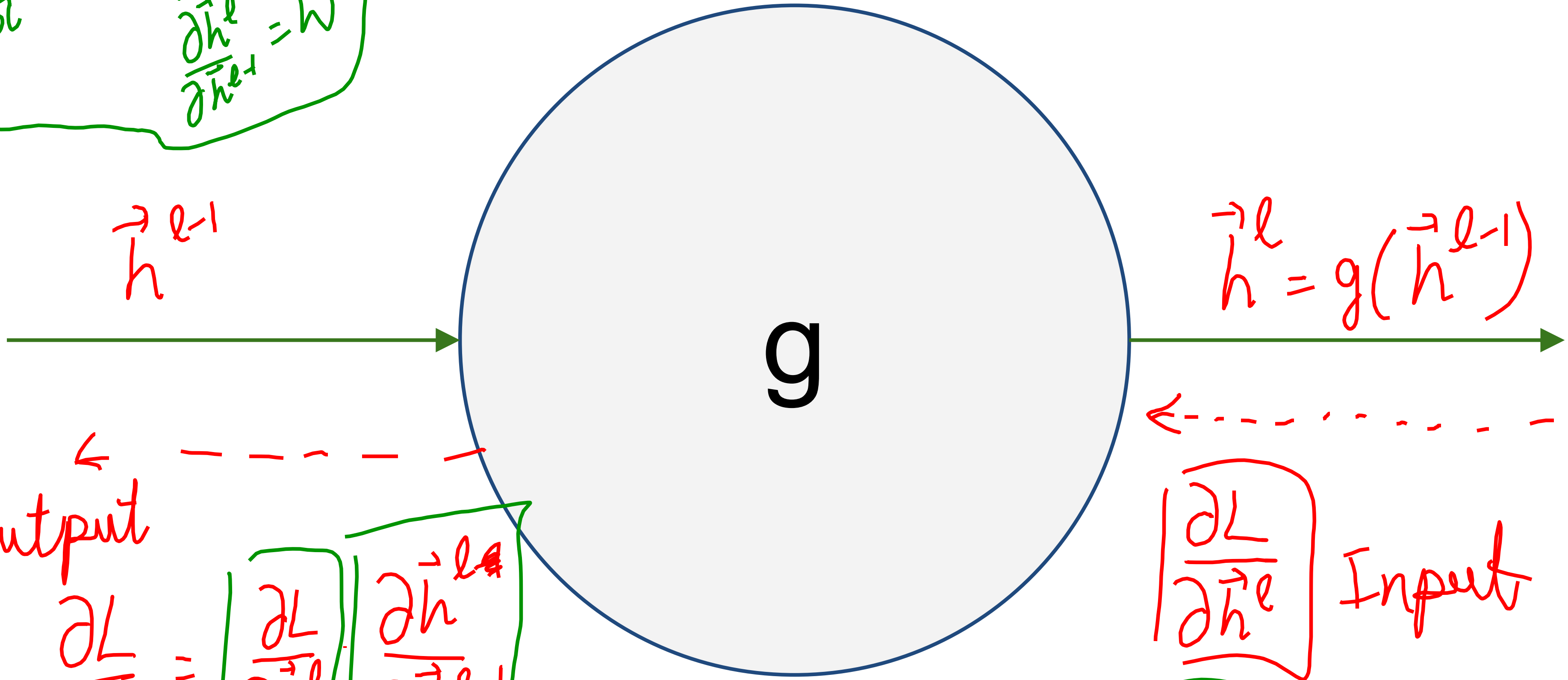
$\vec{y} = A\vec{x}$

$\frac{\partial \vec{y}}{\partial \vec{x}} = A$

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

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

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



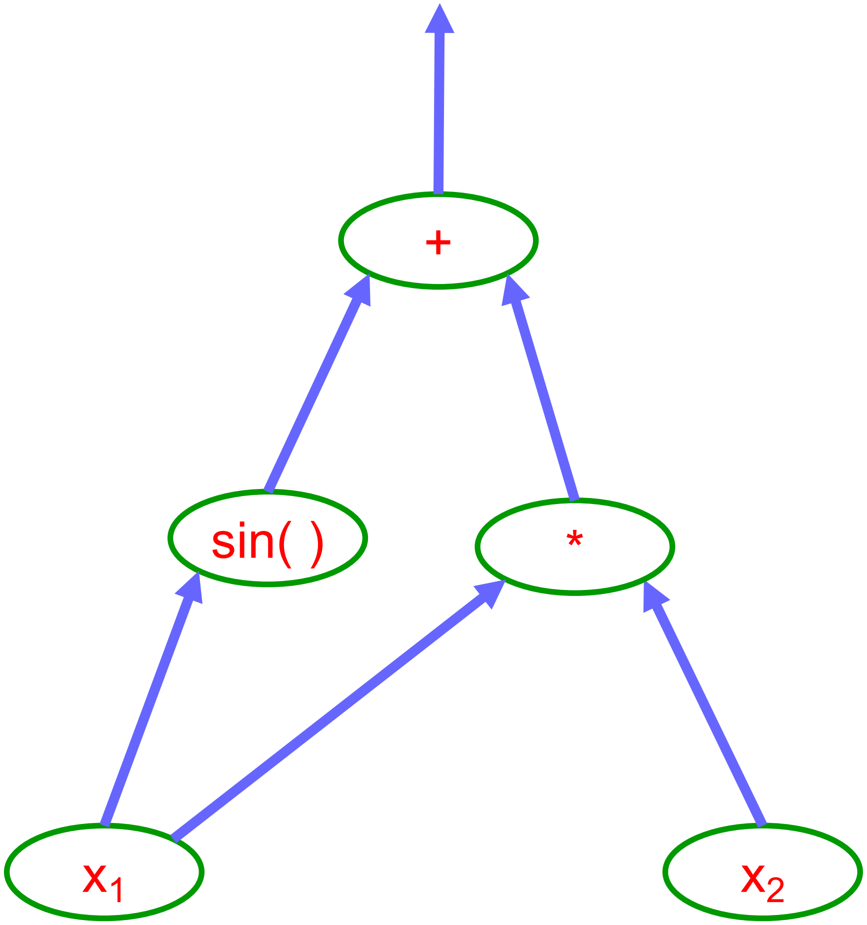
$\boxed{\frac{\partial L}{\partial \vec{h}^{\ell}}}$ Input

Output

$$\frac{\partial L}{\partial \vec{h}^{\ell-1}} = \underbrace{\boxed{\frac{\partial L}{\partial \vec{h}^{\ell}}}}_{\text{Input}} \underbrace{\boxed{\frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}}}}_{\text{Jacobian of } g}$$

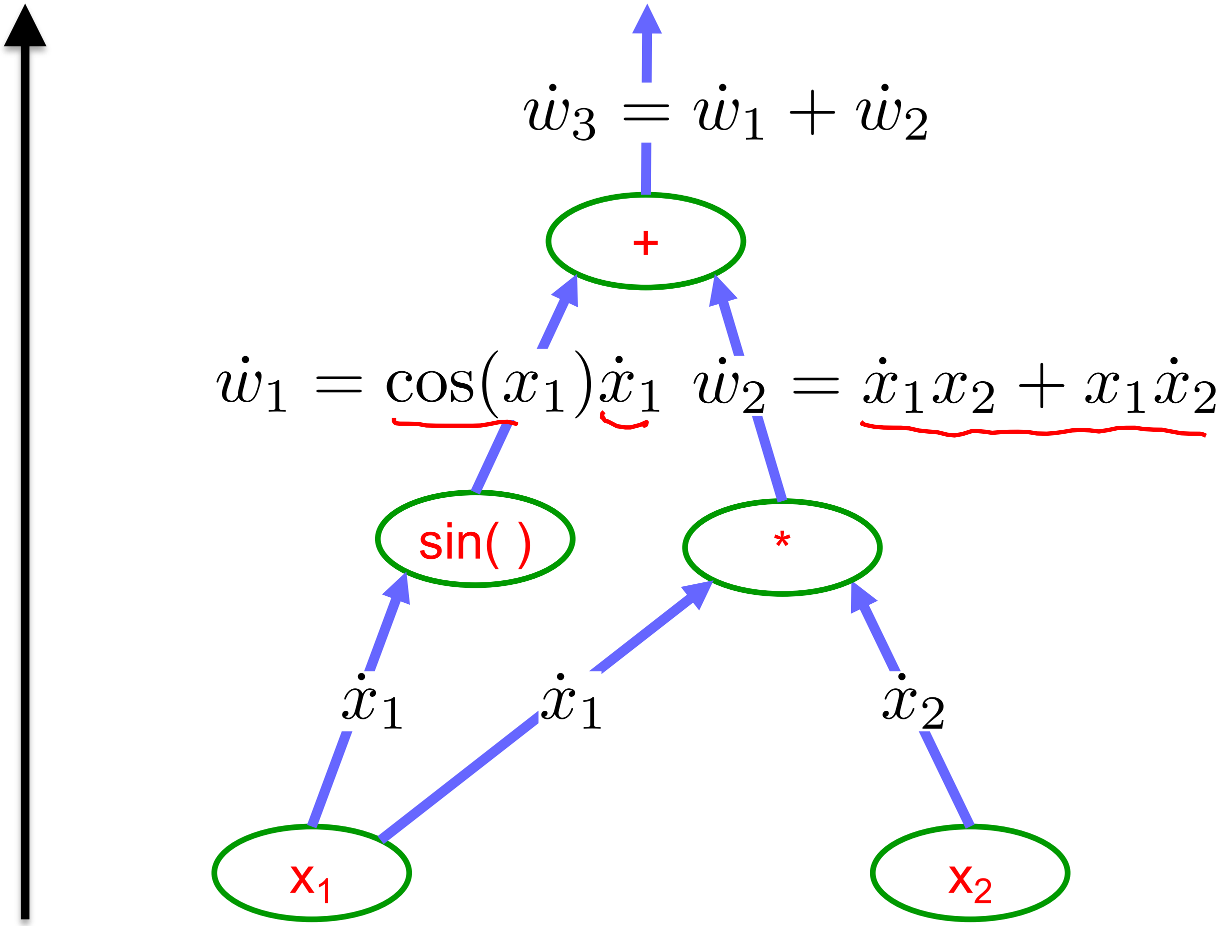
Example: Forward mode AD

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



Example: Forward mode AD

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



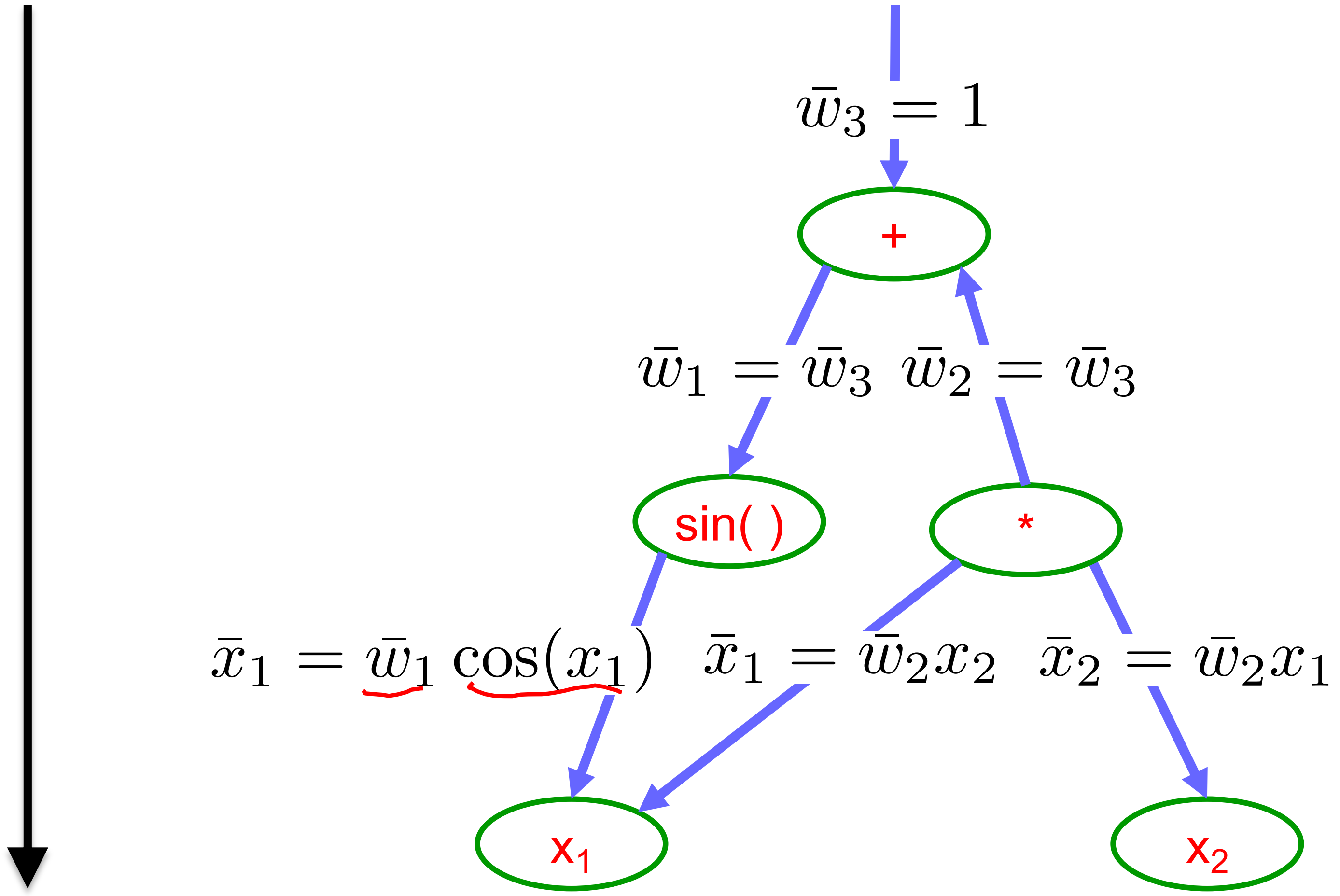
$$\dot{w}_3 = \dot{w}_1 + \dot{w}_2$$

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

$$\dot{x}_1 \quad \dot{x}_1 \quad \dot{x}_2$$

Example: Reverse mode AD

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



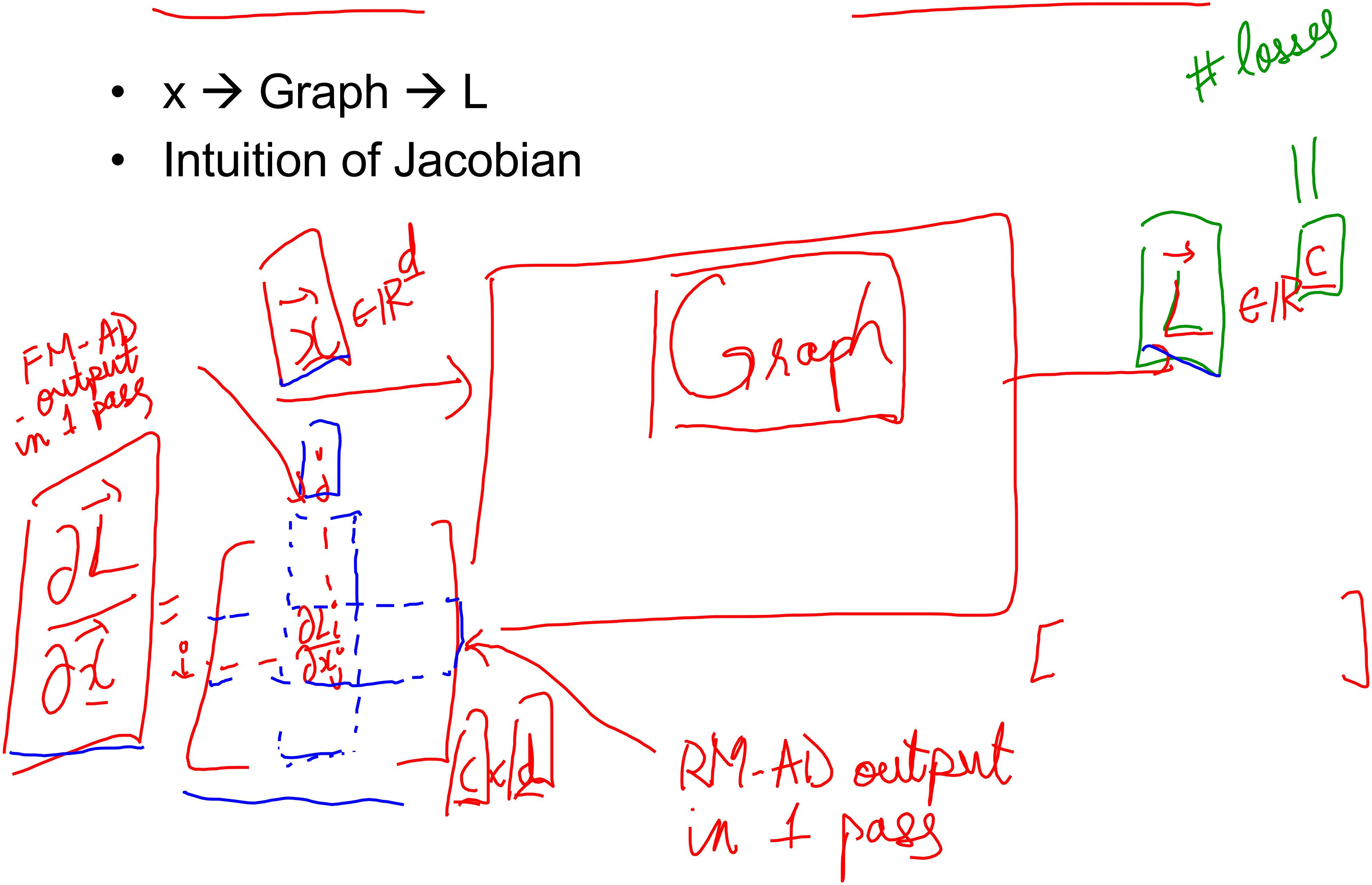
$\bar{w}_3 = 1$

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

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

Forward mode vs Reverse Mode

- x → Graph → L
- Intuition of Jacobian



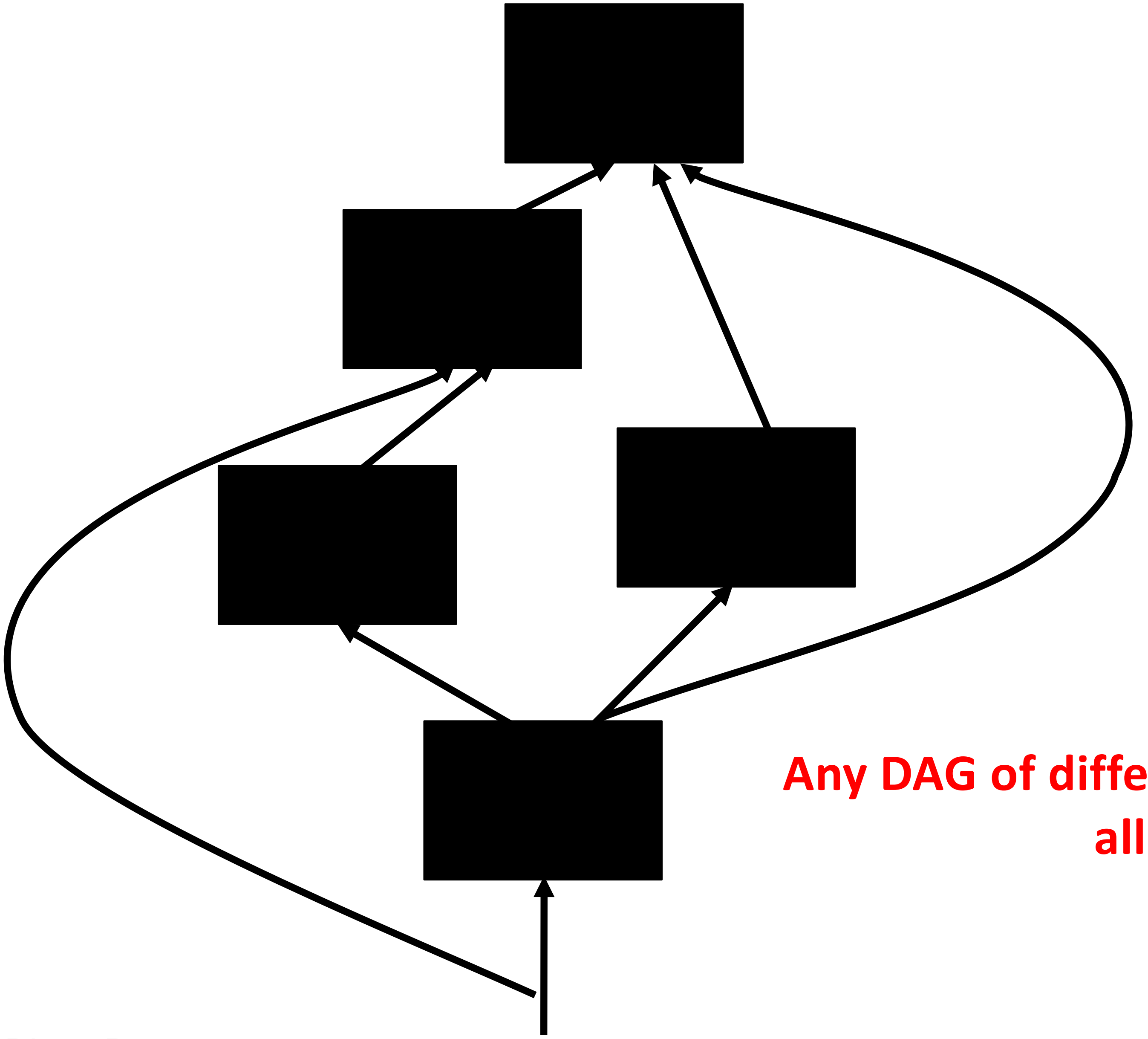
Forward mode vs Reverse Mode

- What are the differences?

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

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

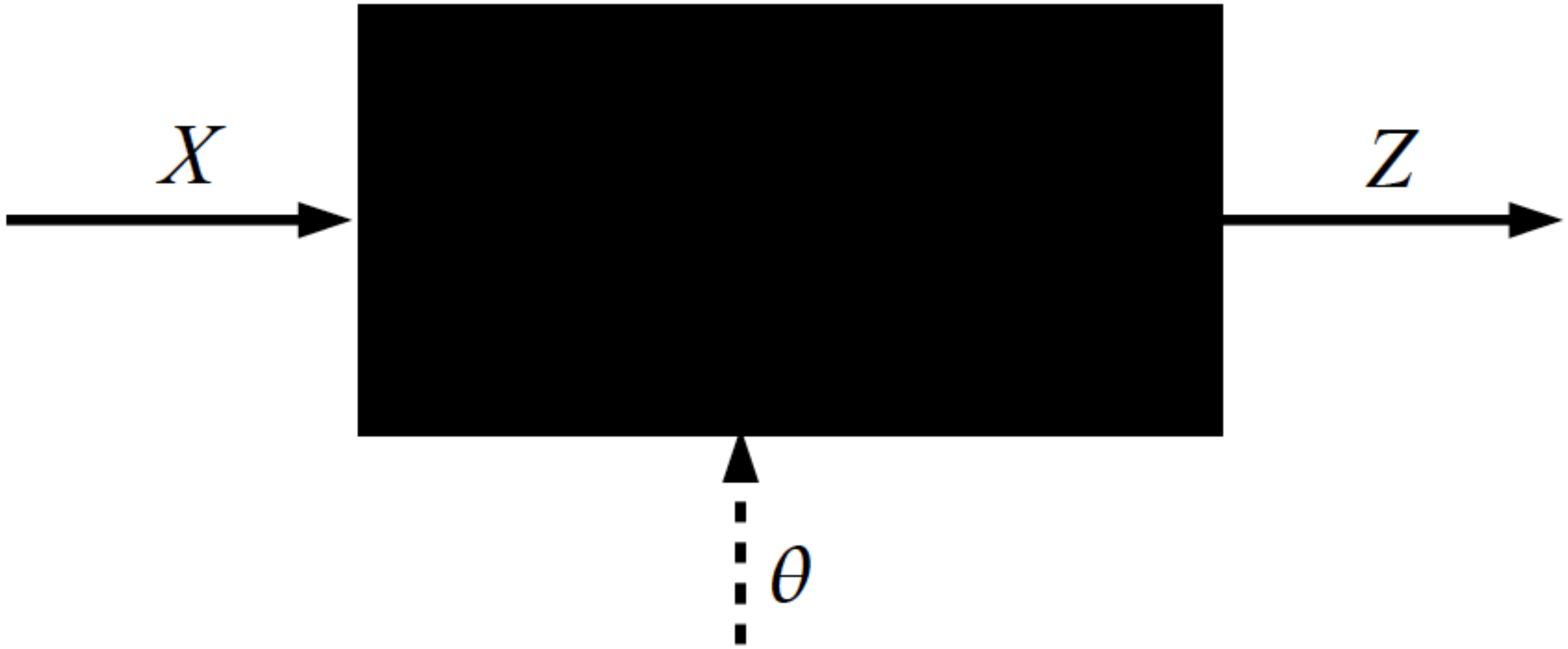
Computational Graph



Any DAG of differentiable modules is allowed!

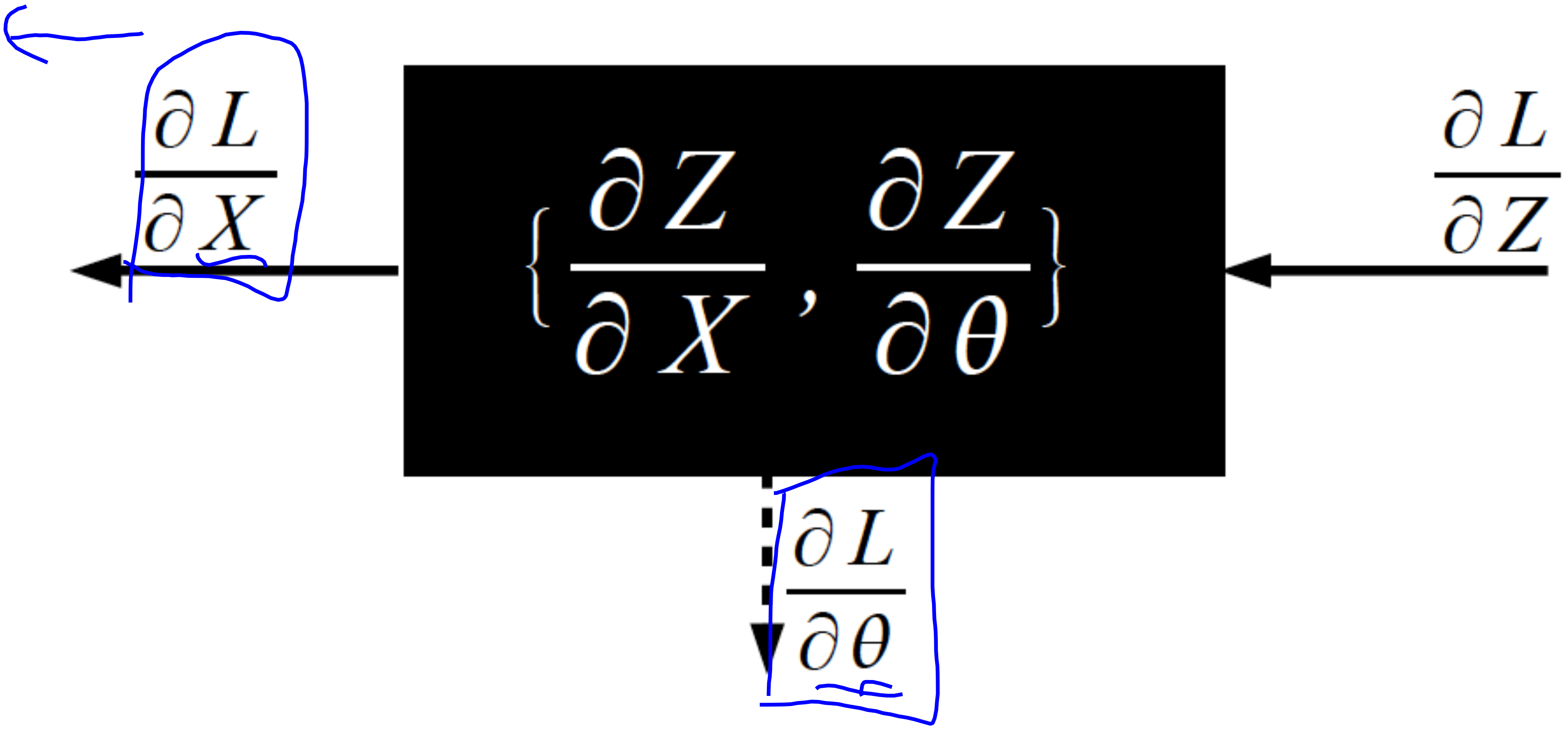
Slide Credit: Marc'Aurelio Ranzato

Key Computation: Forward-Prop



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Key Computation: Back-Prop



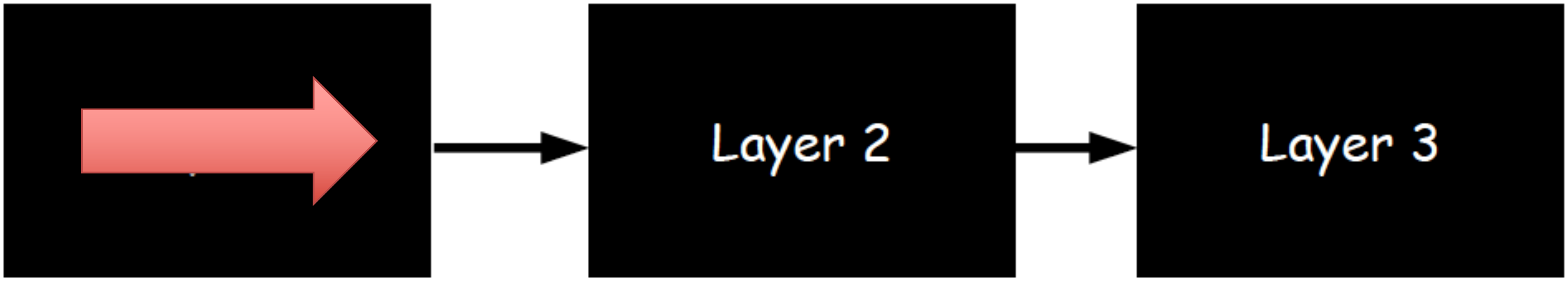
$$\boxed{\frac{\partial L}{\partial \underline{X}}} \longleftarrow \left\{\frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta}\right\} \longleftarrow \frac{\partial L}{\partial Z}$$

$$\boxed{\frac{\partial L}{\partial \underline{\theta}}}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

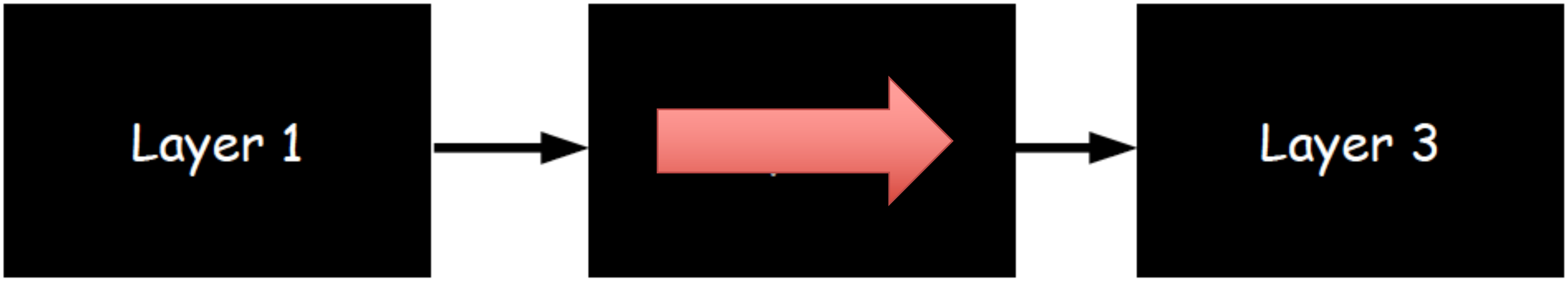
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

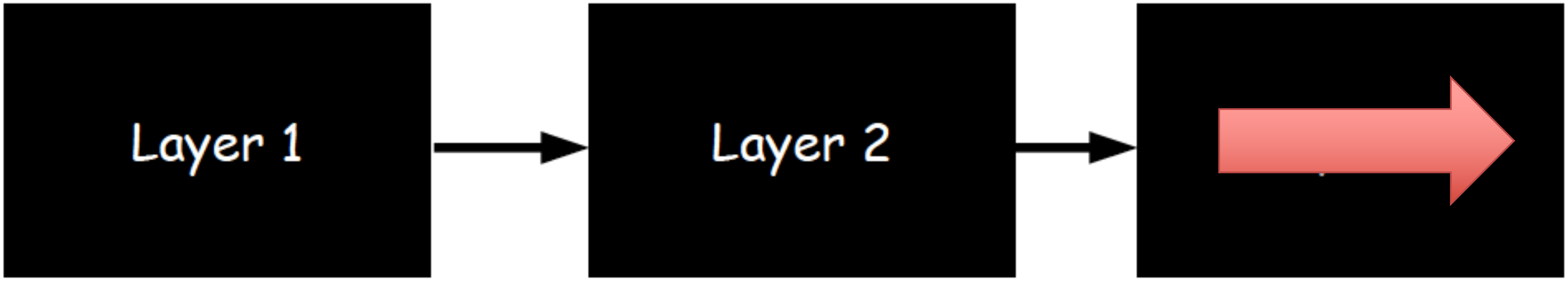
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

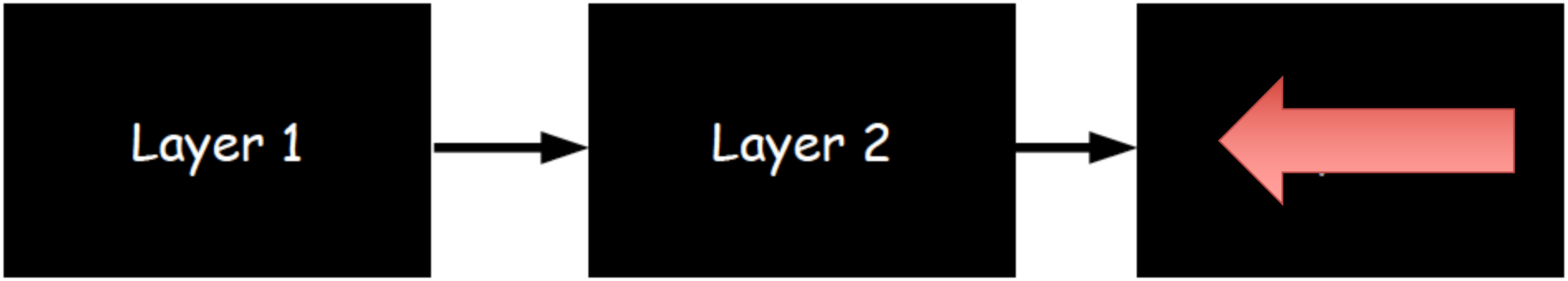
- Step 1: Compute Loss on mini-batch [F-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

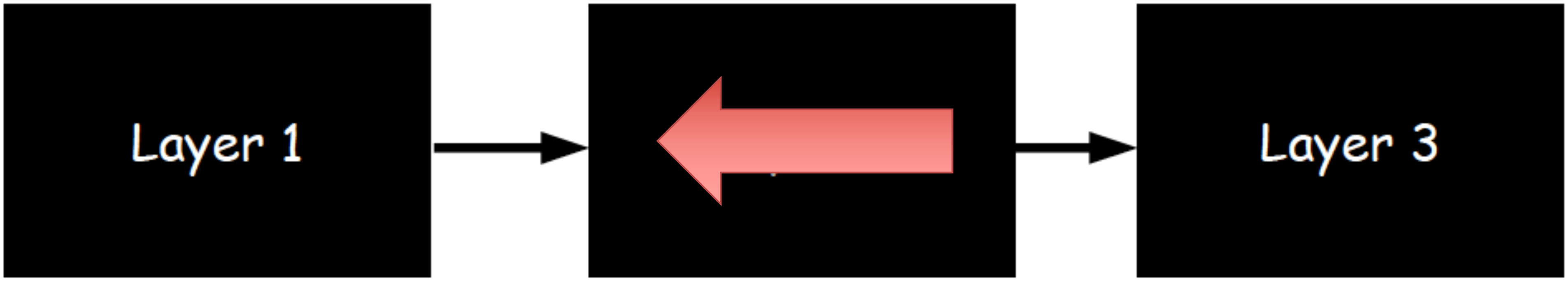
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

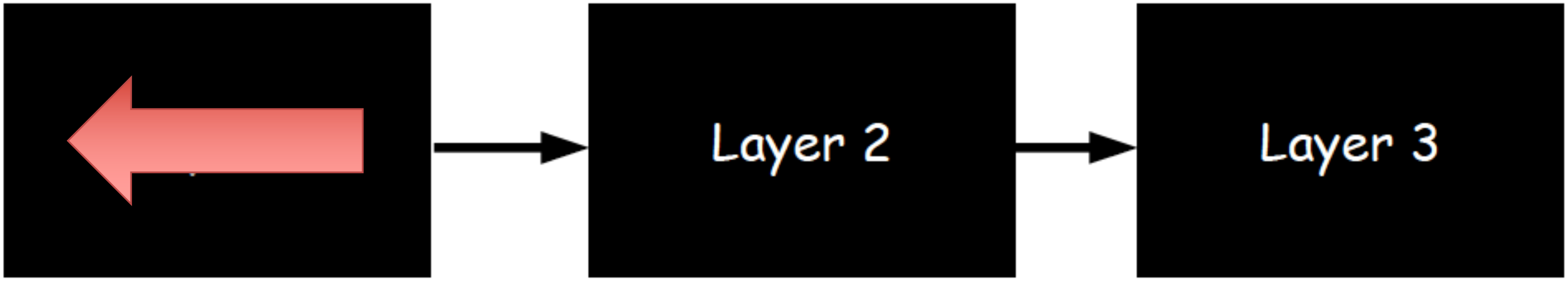
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

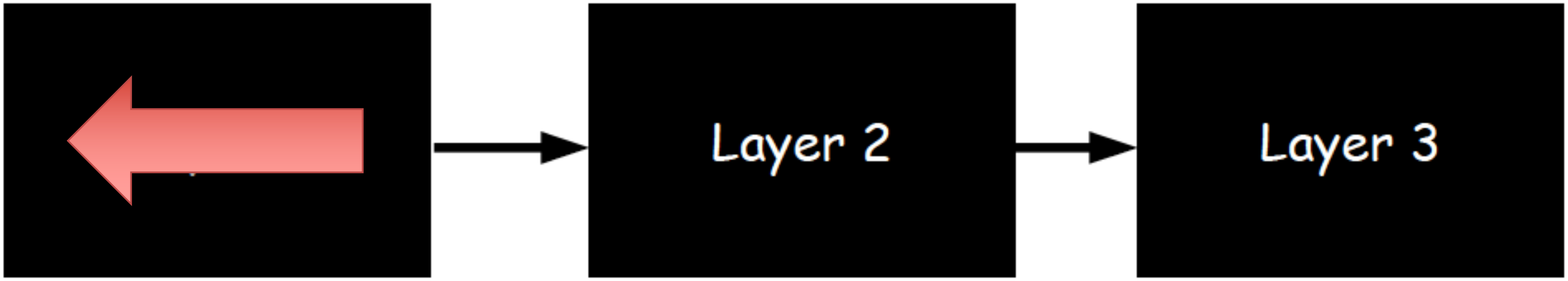
- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
- Step 3: Use gradient to update parameters



$$\theta \leftarrow \theta - \eta \frac{dL}{d\theta}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Plan for Today

- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Neural Network Computation Graph

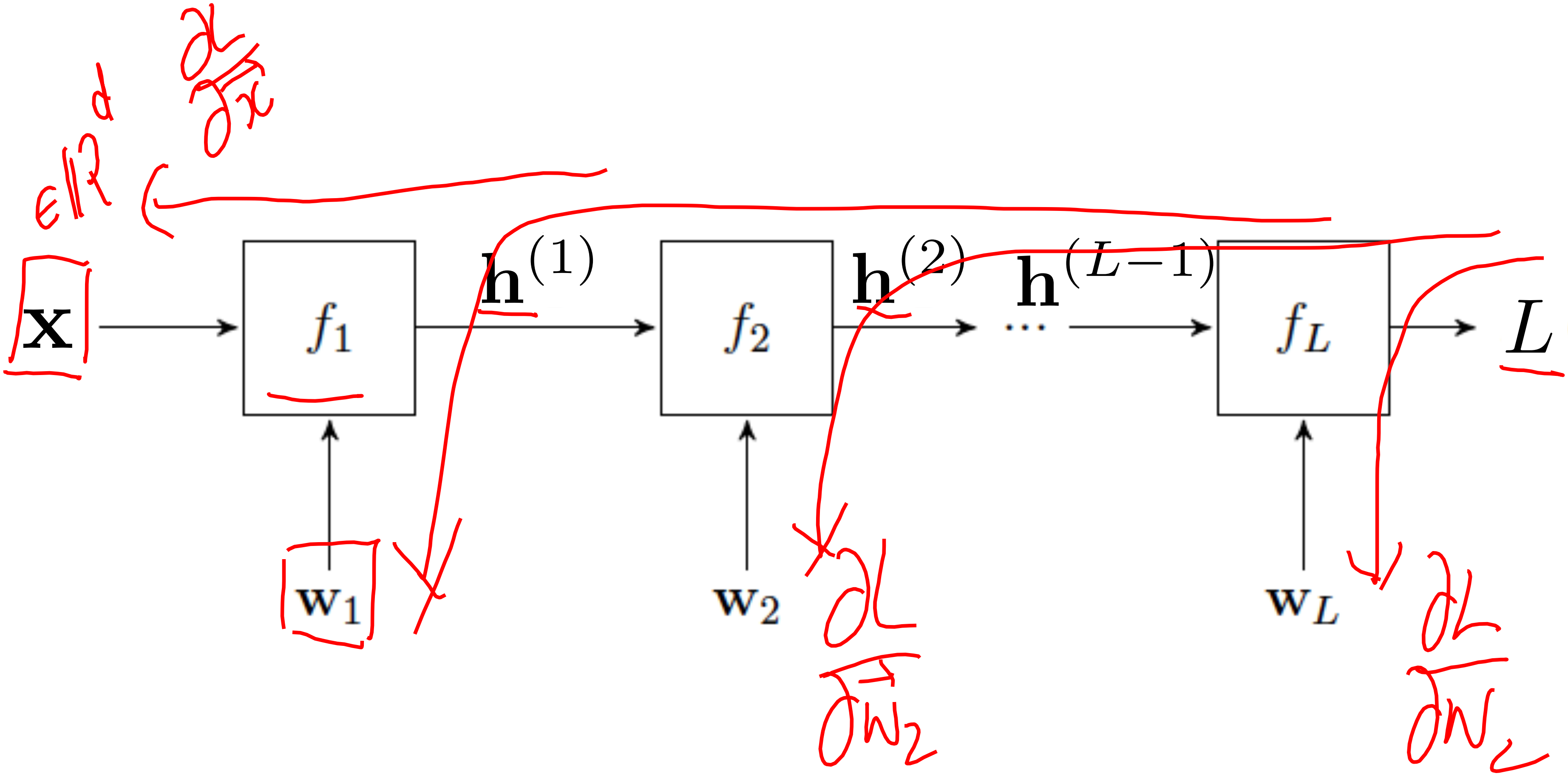
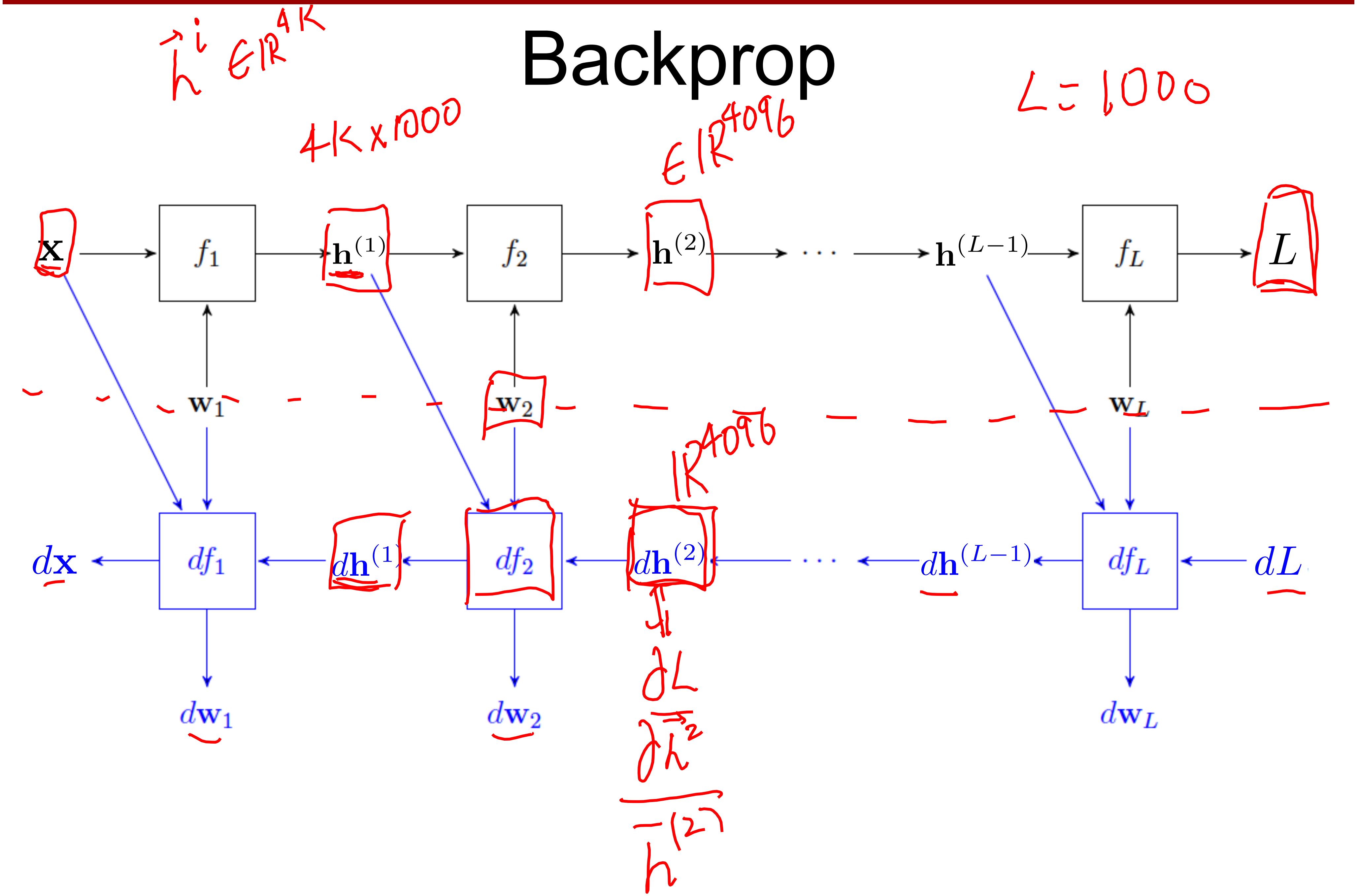


Figure Credit: Andrea Vedaldi

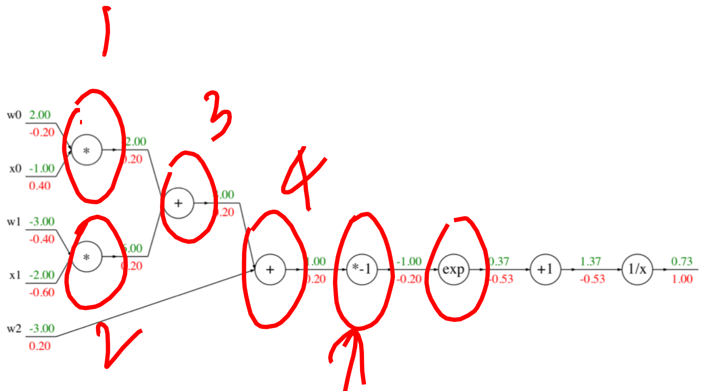
Backprop



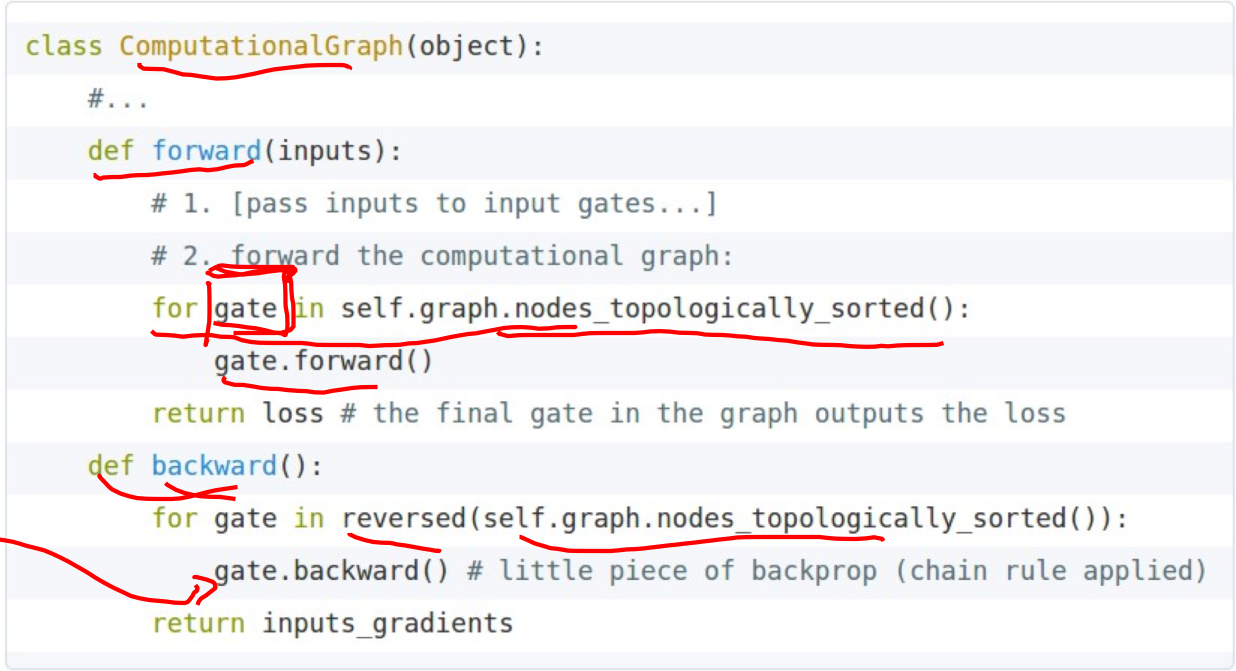
(C) Dhruv Batra

Figure Credit: Andrea Vedaldi

Modularized implementation: forward / backward API



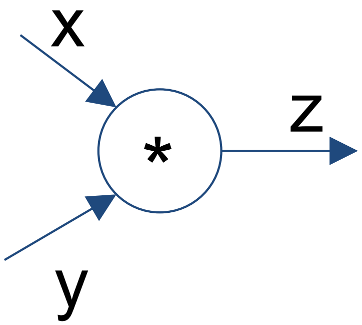
Graph (or Net) object *(rough psuedo code)*



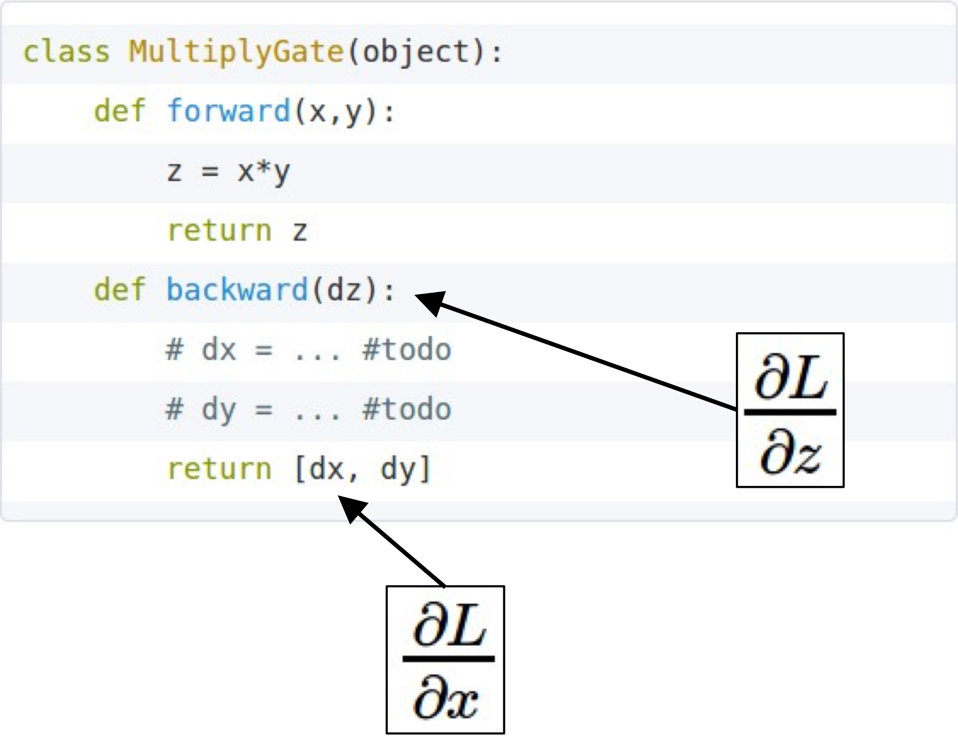
```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph 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)



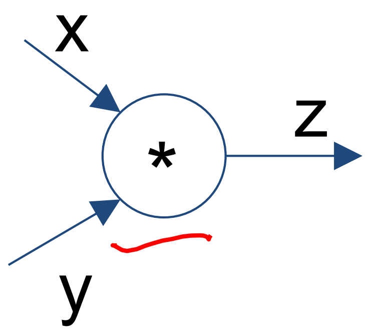
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

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

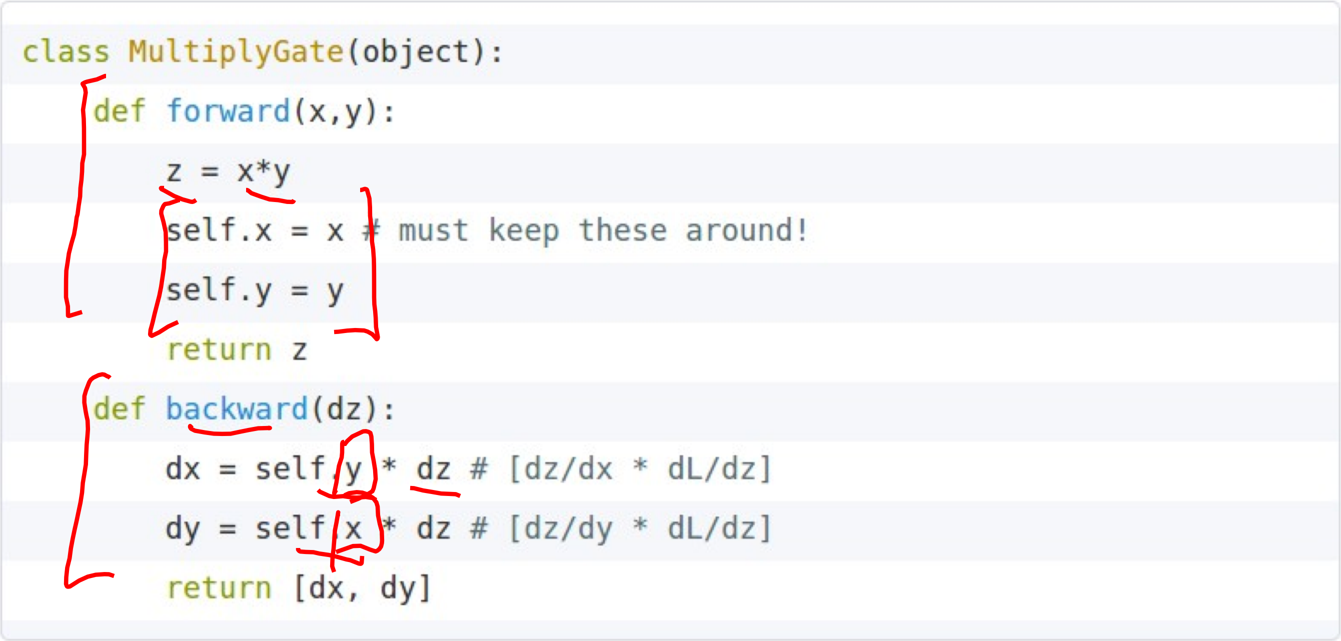
$$\boxed{\frac{\partial L}{\partial x}}$$

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

Modularized implementation: forward / backward API



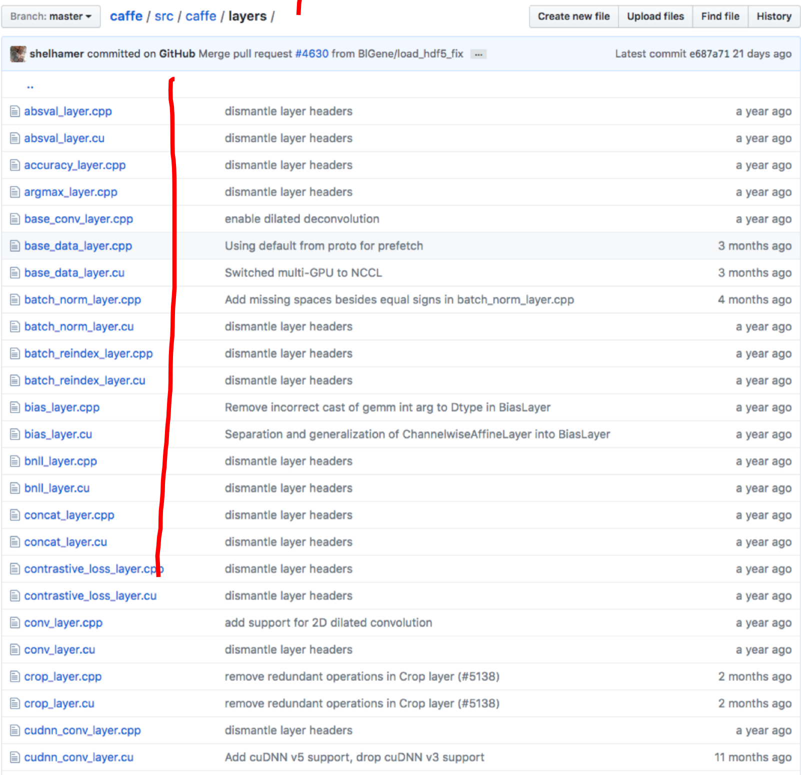
(x,y,z are scalars)



```
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

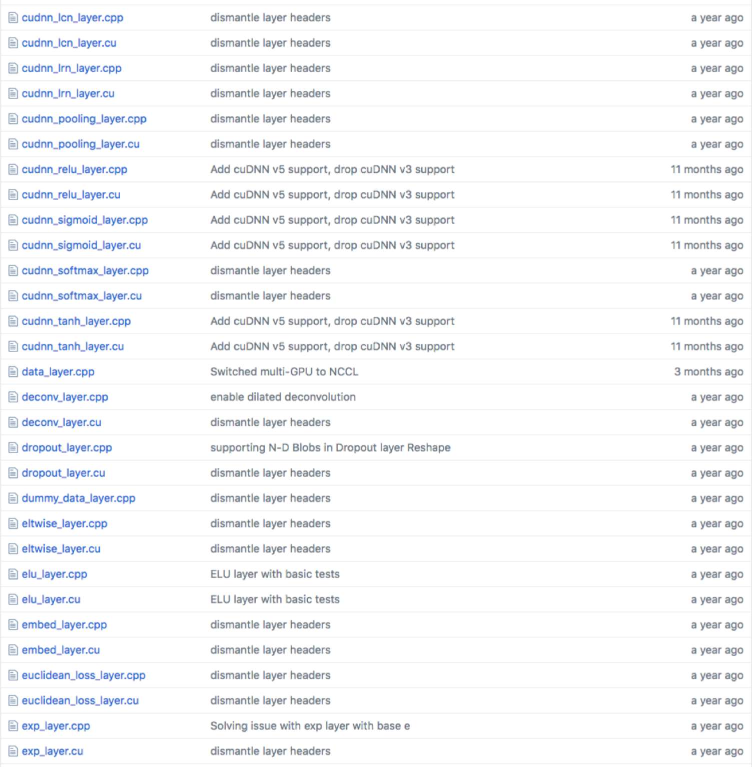
Example: Caffe layers



Branch: master ▾ caffe / src / caffe / layers /

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

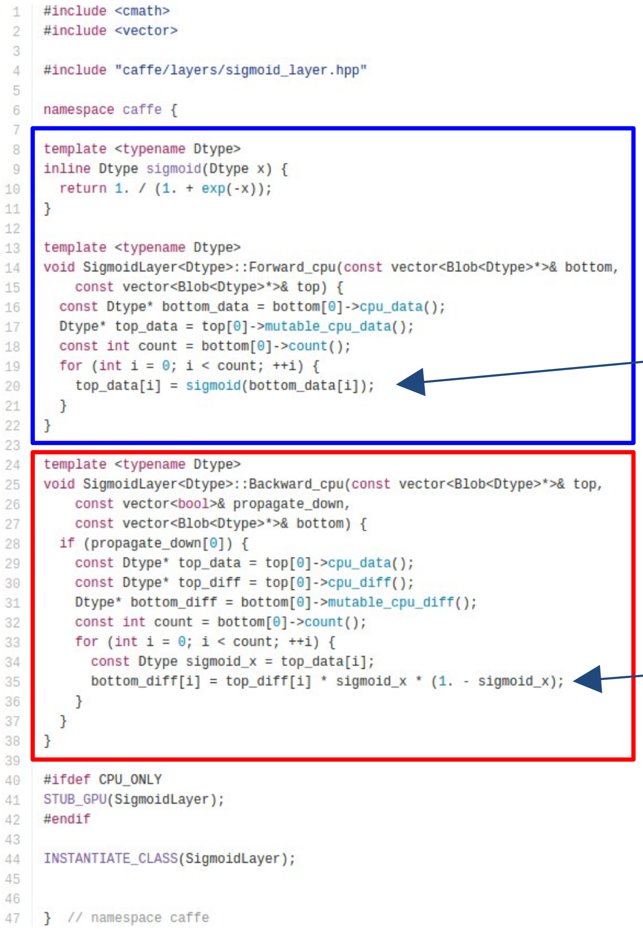
..		
absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default 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

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

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

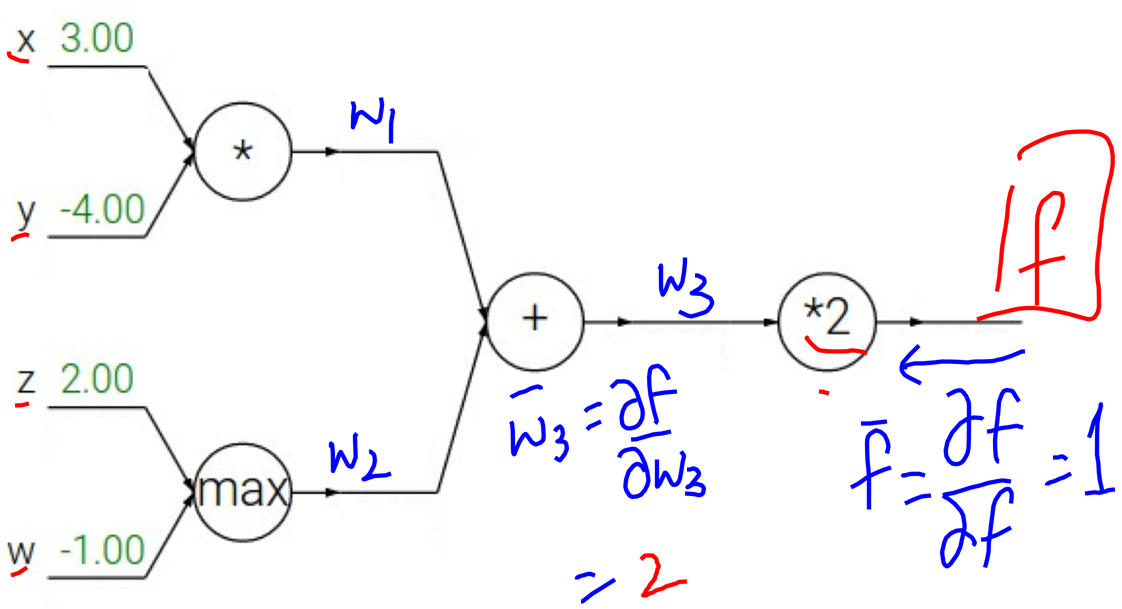
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Plan for Today

- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Backpropagation: a simple example

$$f(x,y,z,w) = 2\left[\underbrace{xy}_{w_1} + \underbrace{\max\{z,w\}}_{w_2}\right]$$

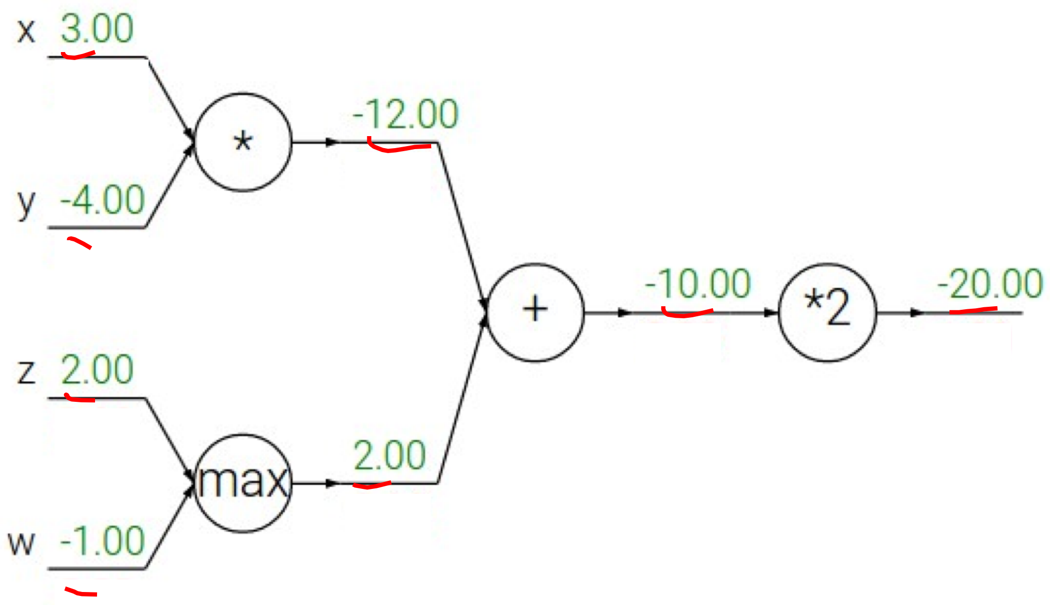


$$\bar{w}_3 = \frac{\partial f}{\partial w_3} = 2$$

$$\bar{f} = \frac{\partial f}{\partial f} = 1$$

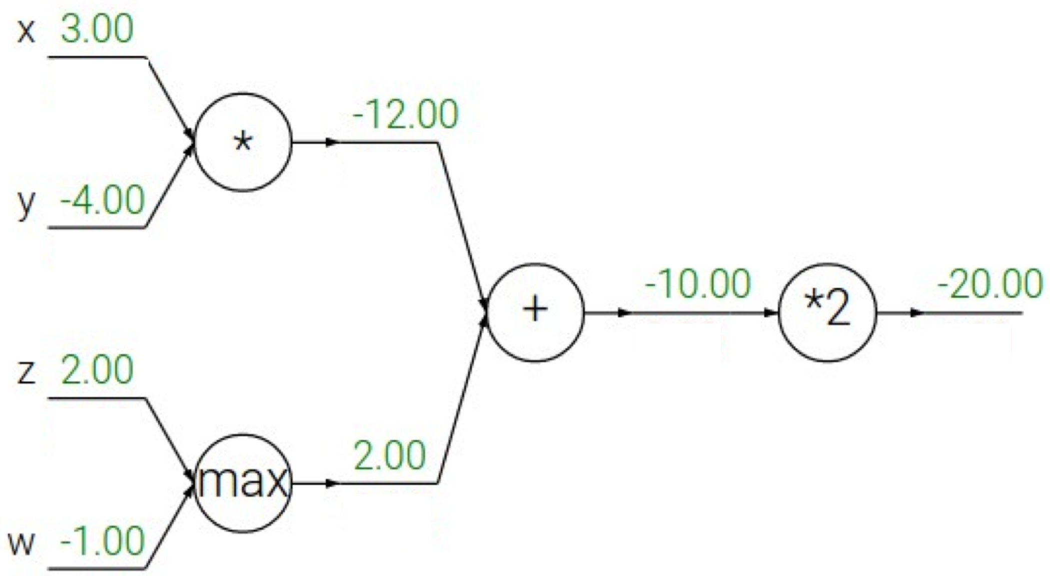
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Backpropagation: a simple example



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

Patterns in backward flow



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

Patterns in backward flow

$\boxed{W_3 = W_1 + W_2}$

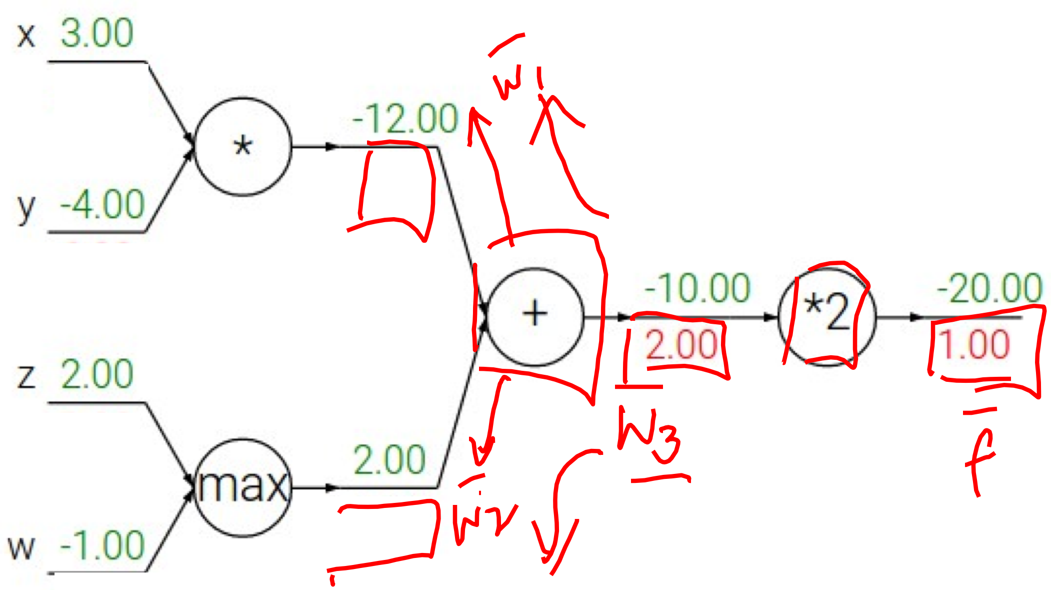
Q: What is an **add** gate?

$$\bar{W_1} = \frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial W_3} \frac{\partial W_3}{\partial W_1}$$

$$= \bar{W_3} \left[1 \right]$$

$$= \bar{W_3}$$

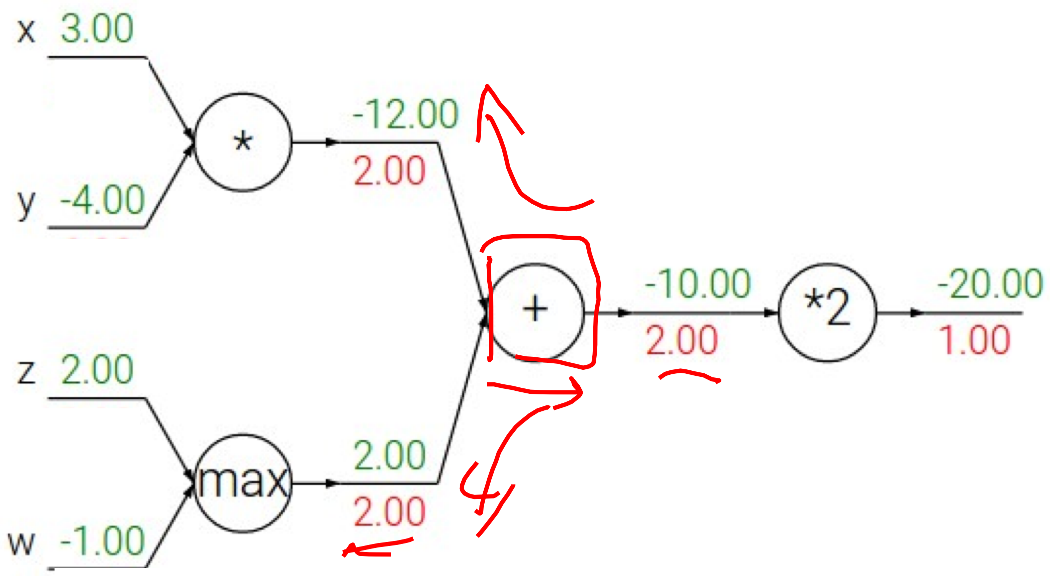
$$\bar{W_2} = \bar{W_3}$$



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

Patterns in backward flow

add gate: gradient distributor



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

Patterns in backward flow

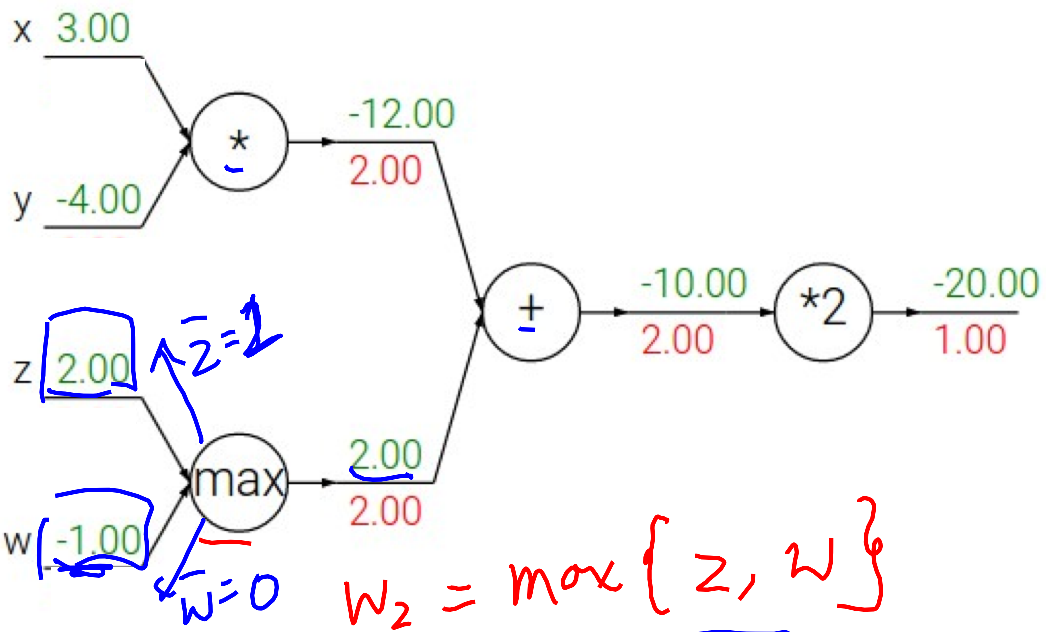
add gate: gradient distributor

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

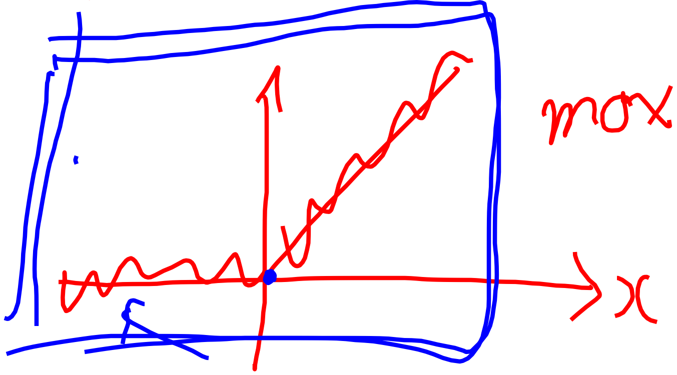
$$W_2 = \begin{cases} z & \text{if } z \geq w \\ w & \text{else} \end{cases}$$

$$\bar{z} = \frac{\partial f}{\partial z} = \underbrace{\frac{\partial f}{\partial W_2}}_{\bar{W_2}} \cdot \frac{\partial W_2}{\partial z}$$

$$= \begin{cases} +1 & \text{if } z \geq w \\ 0 & \text{else} \end{cases}$$



$W_2 = \max\{z, w\}$



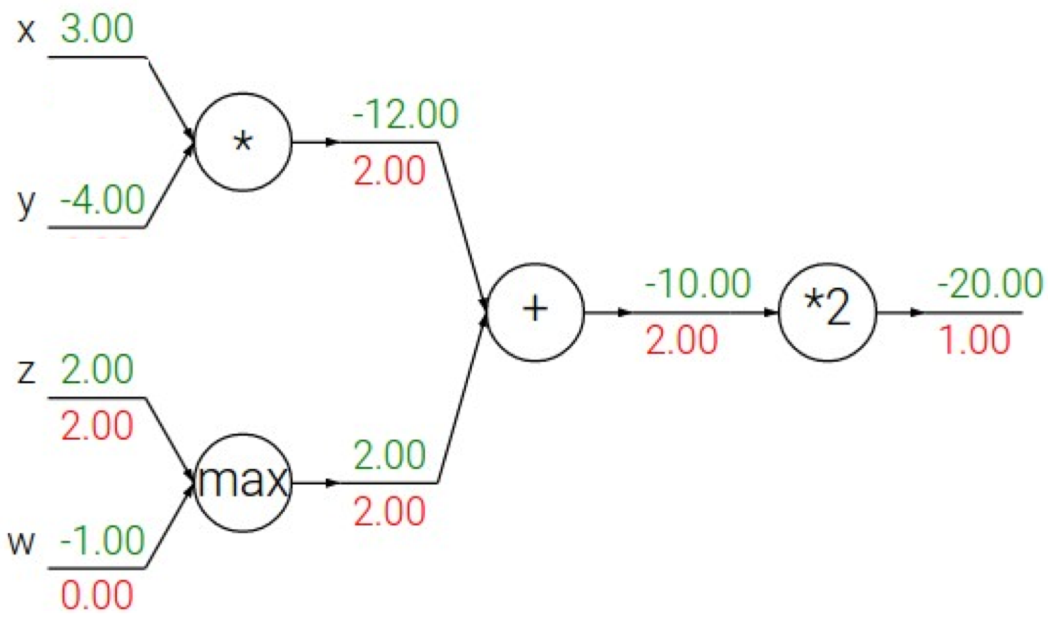
$\max\{0, x\}$

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

$$\bar{x} = \frac{\partial f}{\partial x} = \boxed{\frac{\partial f}{\partial w_1}}\,\boxed{\frac{\partial w_1}{\partial x}} = y$$

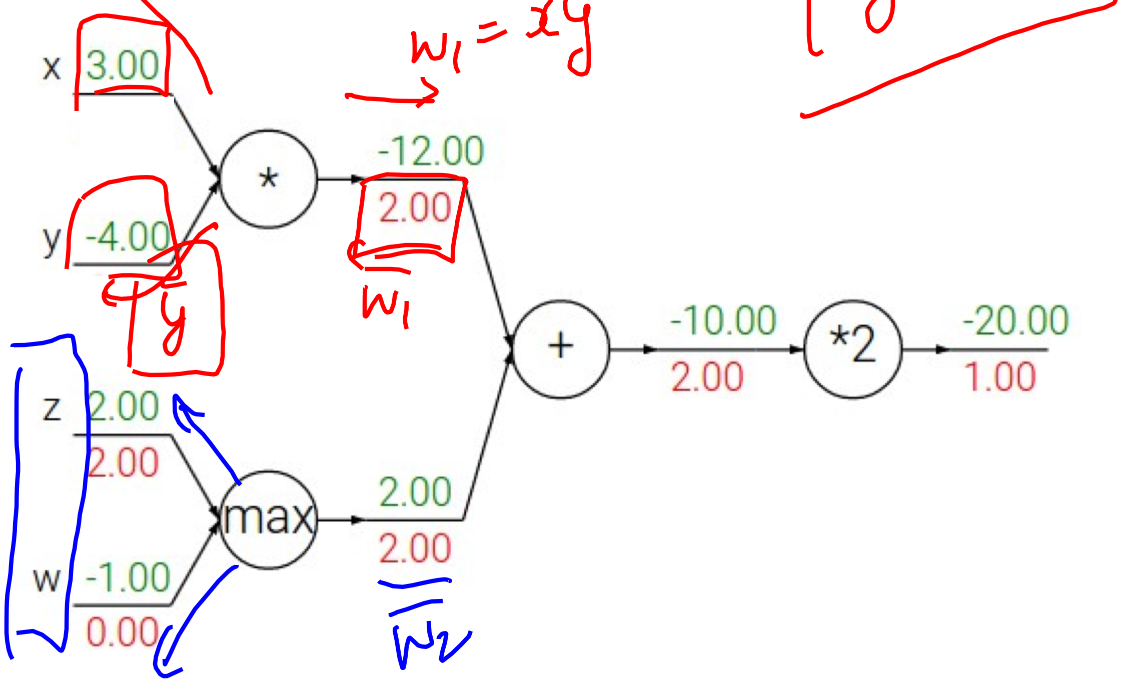
$$\boxed{\begin{aligned}\bar{x} &= \bar{w}_1 y \\ \bar{y} &= \bar{w}_1 x\end{aligned}}$$

$w_1 = xy$

add gate: gradient distributor

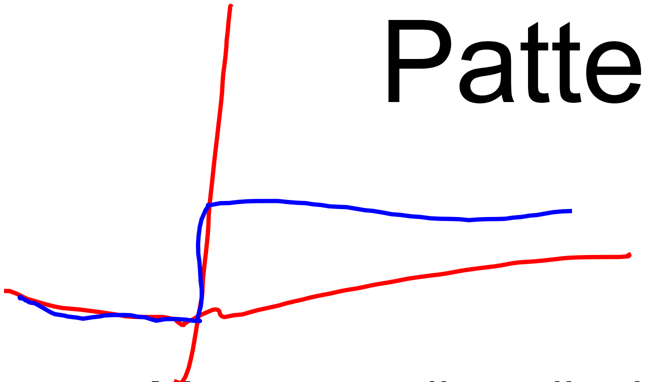
max gate: gradient router

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

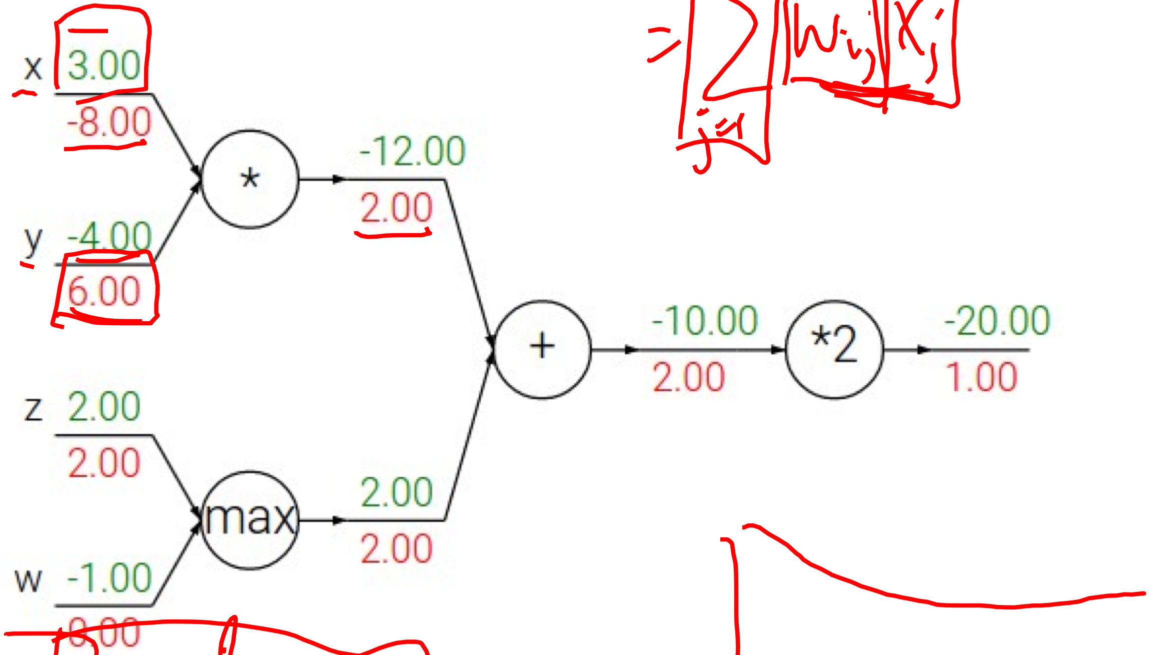
mul gate: gradient switcher

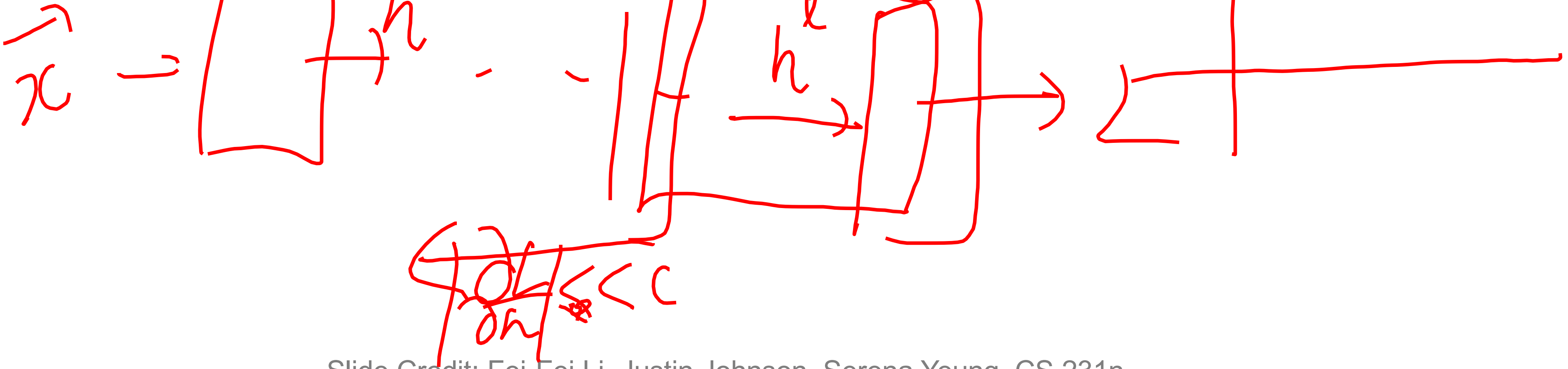
3,000

$h_i^l = \vec{w}_i^T \vec{x}$

$= \sum_j w_{ij} x_j$

$\partial f / \partial y$

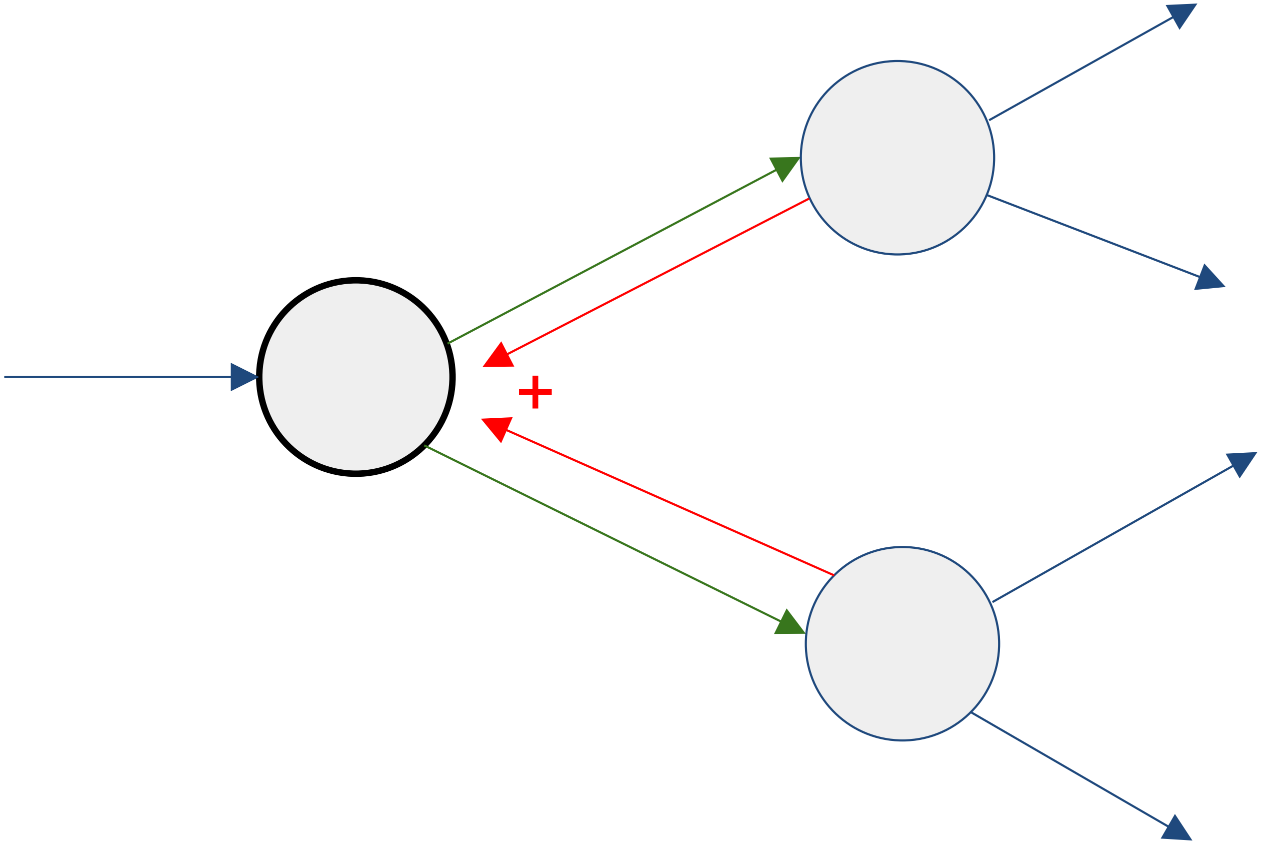




$\frac{\partial L}{\partial h} << c$

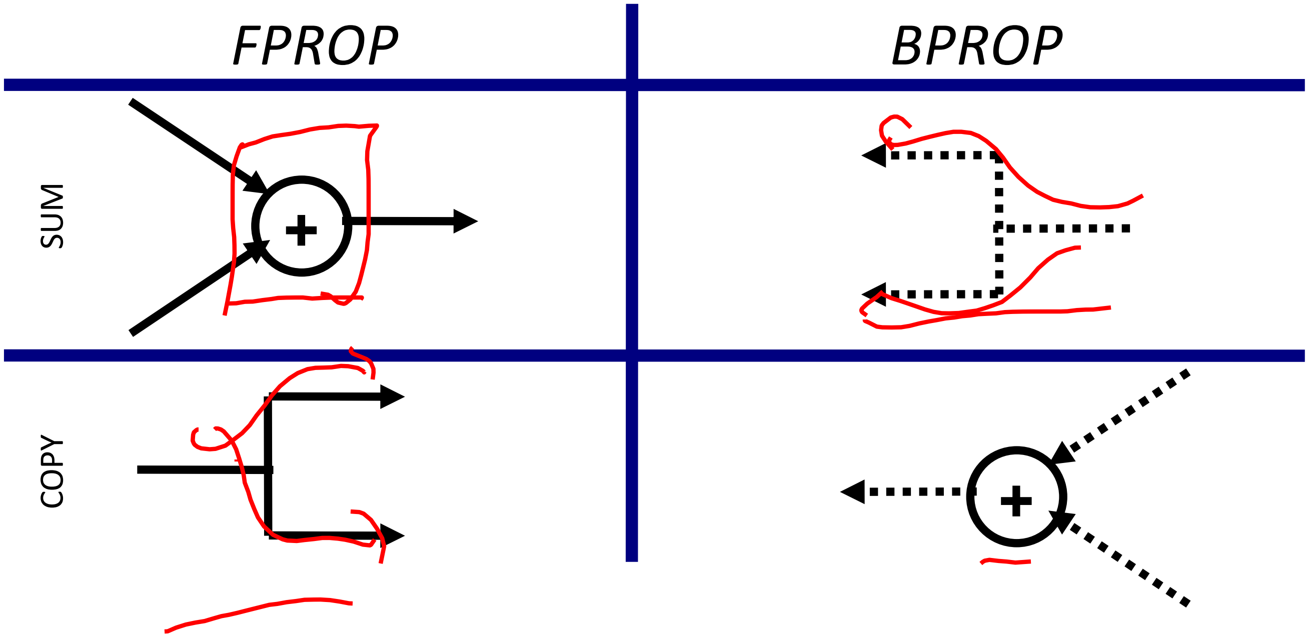
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Gradients add at branches



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

Duality in Fprop and Bprop



Plan for Today

- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Backprop

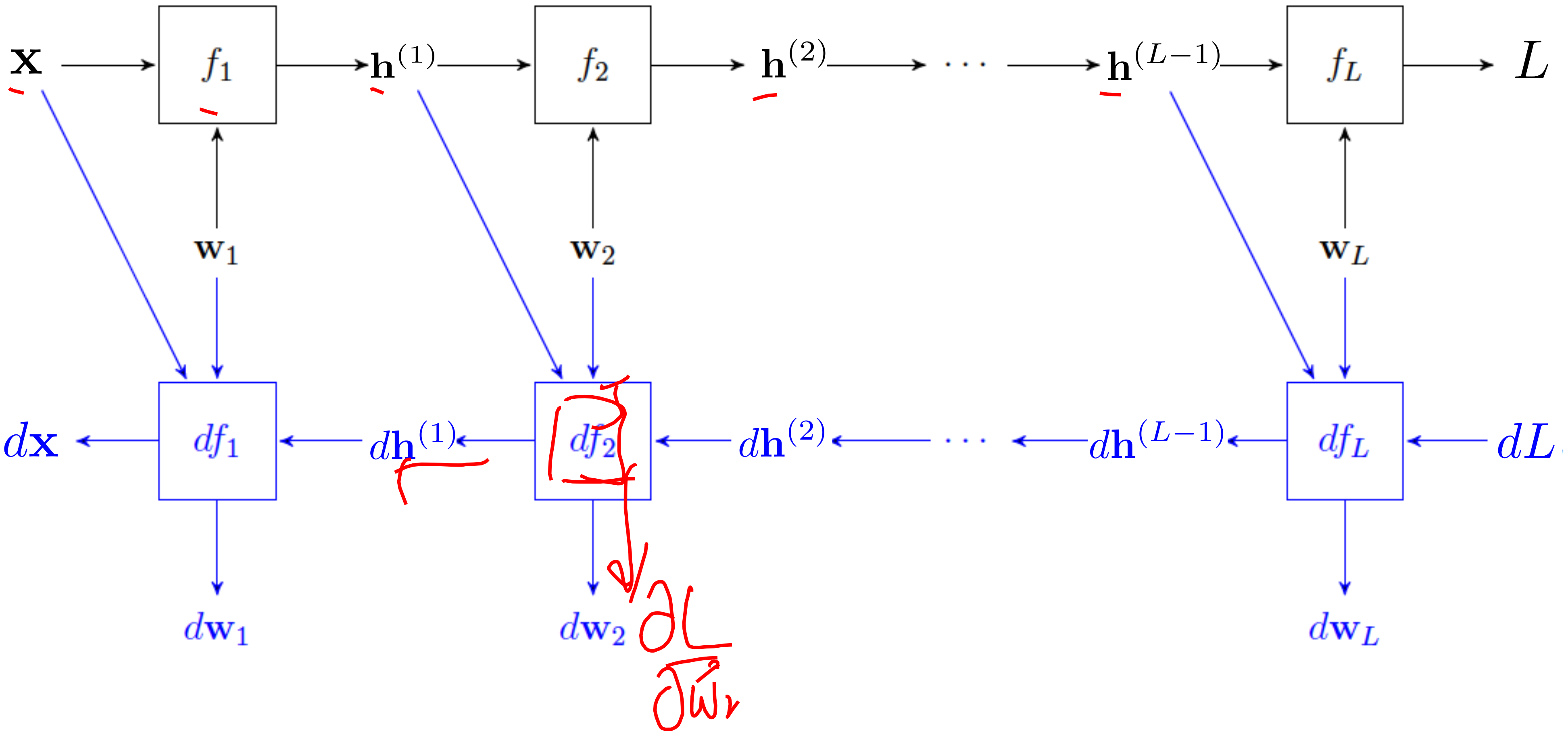


Figure Credit: Andrea Vedaldi

Jacobian of ReLU

$\max\{0, x\}$

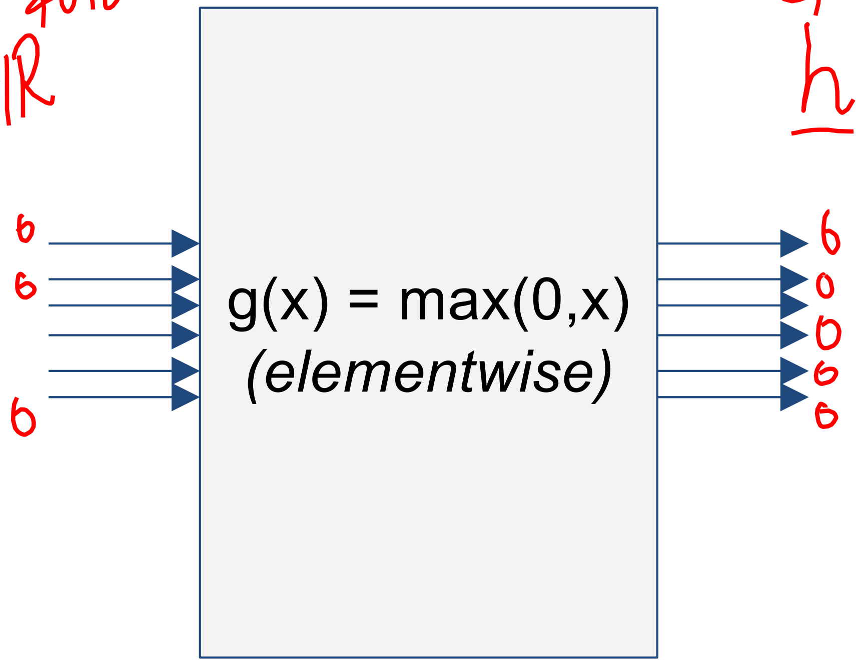
layer l

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

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

$\vec{h}^{l} = \max(0, \vec{h}^{l-1})$

4096-d
input vector



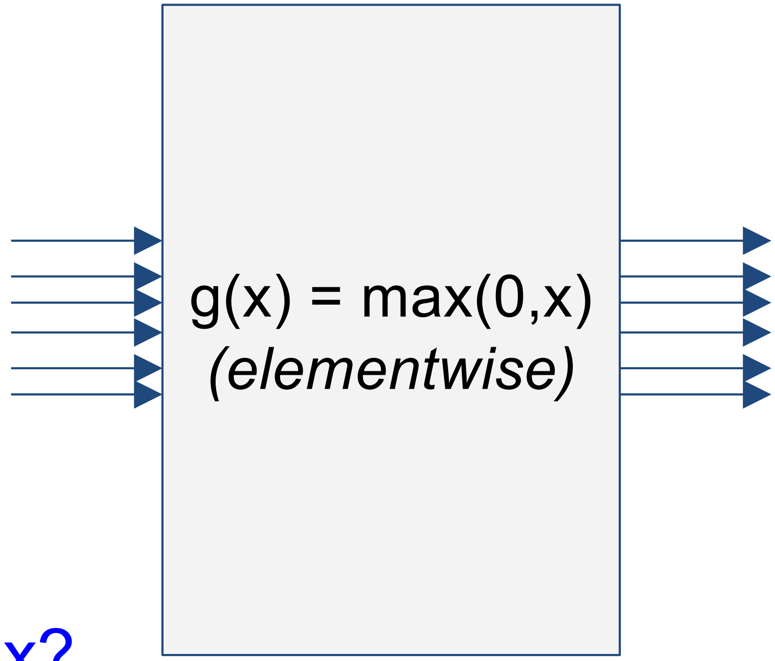
g(x) = max(0,x)
(elementwise)

4096-d
output vector

$$\left[\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}\right]_{4096 \times 4096}$$

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

Jacobian of ReLU



4096-d
input vector

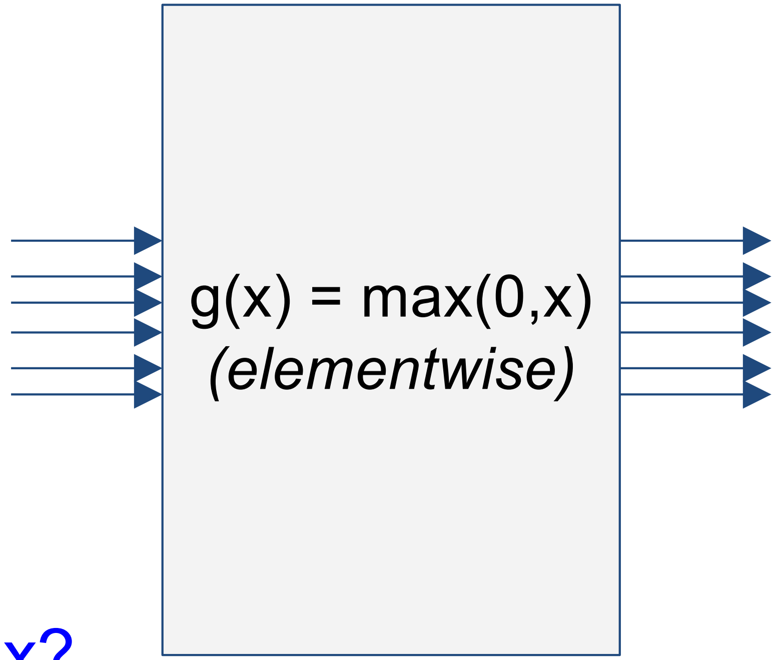
g(x) = max(0,x)
(elementwise)

4096-d
output vector

Q: what is the size of the Jacobian matrix?

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

Jacobian of ReLU



4096-d
input vector

g(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

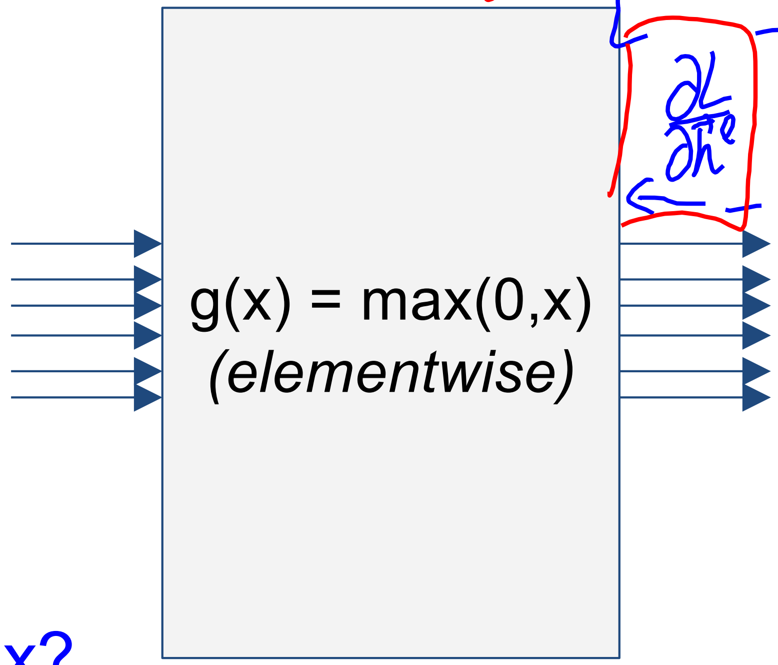
$\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}$

$\frac{\partial h_i^{l}}{\partial h_j^{l-1}}$

0

i k $+1$ 0

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



4096-d
input vector

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

4096-d
output vector

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

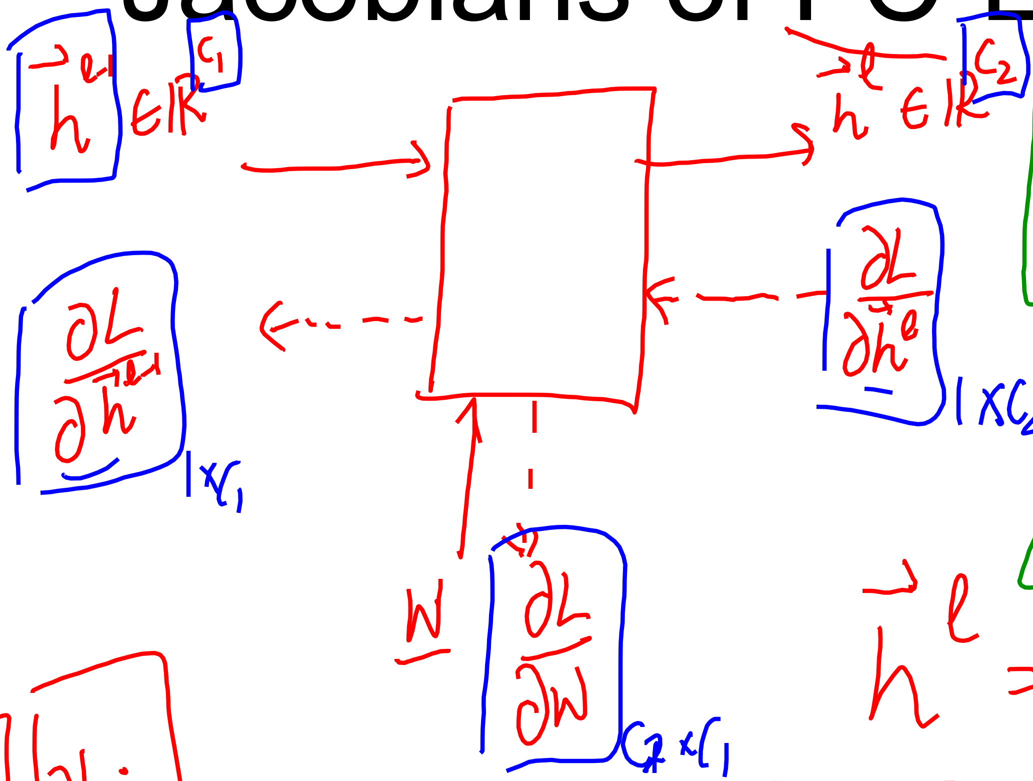
Q2: what does it look like?

$$J = \begin{bmatrix} 0 & & 0 \\ & \ddots & \\ 0 & & 0 \end{bmatrix}$$

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

Jacobians of FC-Layer

$\vec{w}_i^{(t+1)} = \vec{w}_i^{(t)} - \eta \frac{\partial L}{\partial \vec{w}_i}$



$\vec{h}^{\ell-1} \in \mathbb{R}^{c_1}$
$\vec{h}^{\ell} \in \mathbb{R}^{c_2}$
$\frac{\partial L}{\partial \vec{h}^{\ell}}$
$1 \times c_2$
$\frac{\partial L}{\partial \vec{h}^{\ell-1}}$
$1 \times c_1$
W
$\frac{\partial L}{\partial W}$
$c_2 \times c_1$

If $\vec{w}_i^{(0)} = \vec{w}_j^{(0)}$

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

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

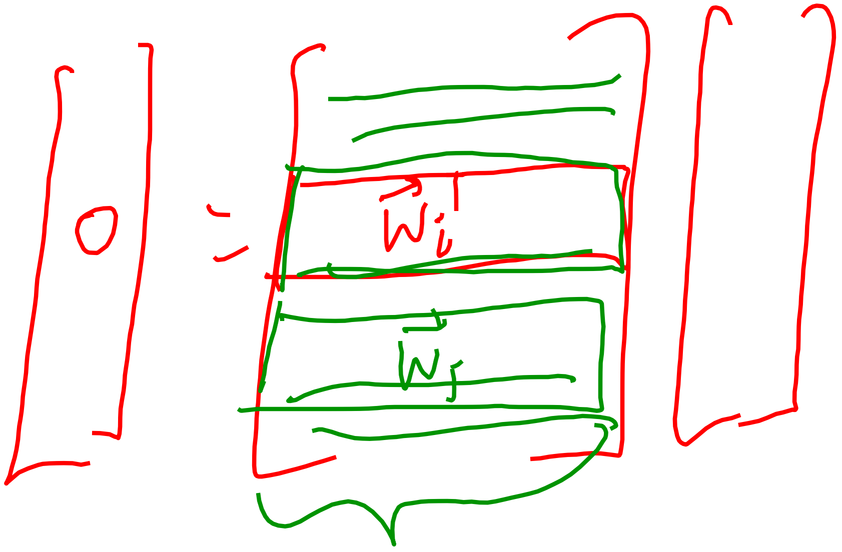
$\vec{h}^{\ell} = W \vec{h}^{\ell-1}$

$\frac{\partial L}{\partial \vec{w}_i} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial \vec{w}_i}$

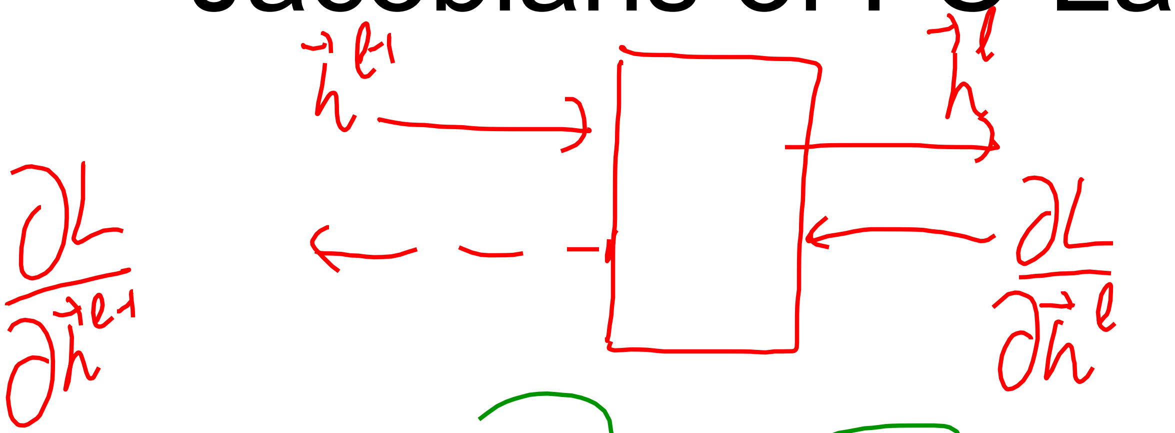
Scalar input

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

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



Jacobians of FC-Layer


$\vec{h}^{l-1}$
$\vec{h}^{l}$
$\frac{\partial L}{\partial \vec{h}^{l-1}}$
$\frac{\partial L}{\partial \vec{h}^{l}}$

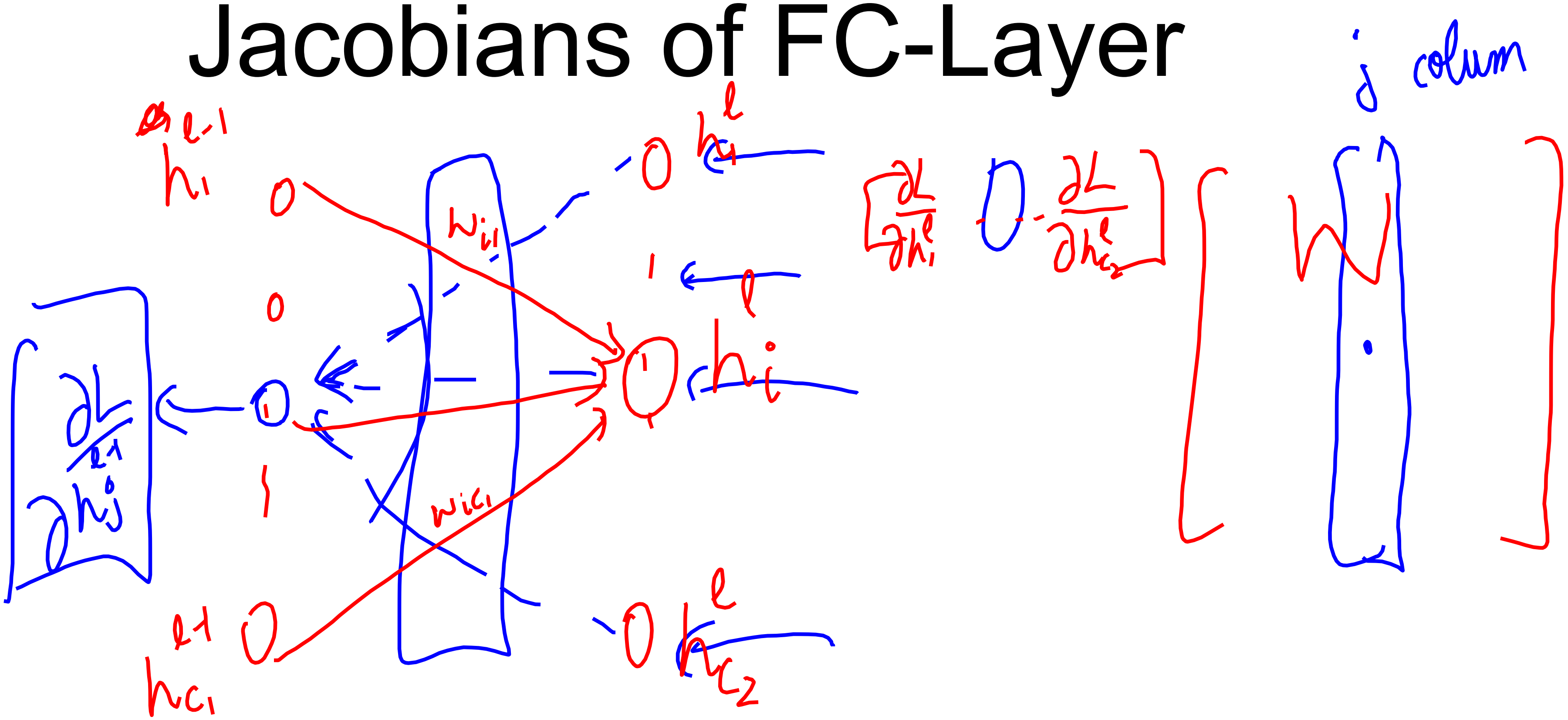
$$\boxed{\vec{h}^{(l)}} = \underline{W} \boxed{\vec{h}^{l-1}}$$

$$\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}} = W$$

$$\frac{\partial L}{\partial \vec{h}^{l-1}} = \overset{\text{input}}{\left[\frac{\partial L}{\partial \vec{h}^{l}}\right]} \left[\frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}\right]$$

$$= \boxed{\text{input}} \cdot \underline{W}$$

Jacobians of FC-Layer



h_1^{l-1}
$h_{c_1}^{l-1}$
h_1^l
h_i^l
$h_{c_2}^l$
w_{i1}
w_{ic_1}
$\frac{\partial L}{\partial h_j^{l-1}}$
$\left[\frac{\partial L}{\partial h_1^l} \; 0 \; \cdots \; \frac{\partial L}{\partial h_{c_2}^l}\right]$
j column
W
0
1

Jacobians of FC-Layer