CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Computing Gradients
  - Backprop in Conv Layers
- Forward mode vs Reverse mode AD
- Modern CNN Architectures

Zsolt Kira

Georgia Tech

Administrivia

- HW1 due date moved!
  - Due: 02/18, 11:55pm

- Project topic submissions
  - Submit by **02/21** to get comments
  - Form filled out with:
    - Members identified
    - Paragraph of problem and another paragraph of what has been done in the literature and approach (note: the approach can be selected from an existing paper and reimplemented)
    - Description of what each member will do
    - Link

  - A project idea: ICLR reproducibility challenge (https://reproducibility-challenge.github.io/iclr_2019/)
    - Official submission Jan but can still do it and submit later!

- Google cloud credits out! (see piazza for details)
- Clouderizer ties in with Google Cloud Platform (GCP)

Matrix/Vector Derivatives Notation

	S	V	M
S	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \vec{x}}$	$\frac{\partial y}{\partial X}$
V	$\frac{\partial \vec{y}}{\partial x}$	$\frac{\partial \vec{y}}{\partial \vec{x}}$	
M	$\frac{\partial Y}{\partial x}$	Tensor	

$x, y \in \mathbb{R}^1$

$\vec{x} \in \mathbb{R}^d$

$\vec{y} \in \mathbb{R}^k$

Convention: $\frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \vdots \\ \frac{\partial y_k}{\partial x} \end{bmatrix}$ numerator = dim 1 = col-vector

[Gradient] $\frac{\partial y}{\partial \vec{x}} = \left[\frac{\partial y}{\partial x_1} \cdots \frac{\partial y}{\partial x_d} \right]$ denominator = dim 2 = row-vector

→ [More concrete notation for scalars]

$$L'(x) = f'(g(x))\, g'(x)$$

→ [With intermediate variables]

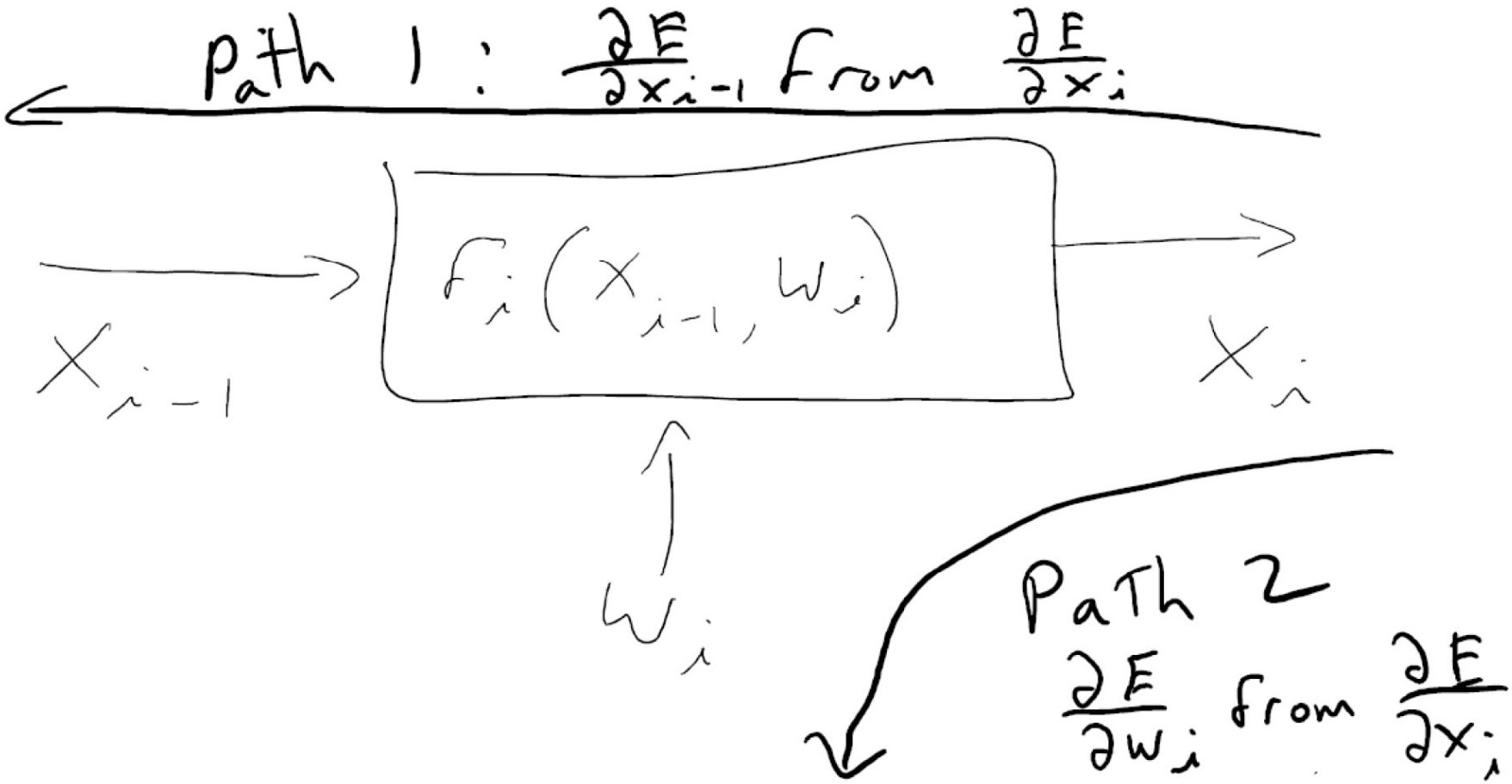
$$y = g(x)$$
$$z = f(y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Example: $L_i(w) = -\log\left(\frac{1}{1+e^{-\vec{w}^T x_i}}\right)$ [For $y_i = +1$]

$$= \Big(\underbrace{-\log(\cdot)}_{\frac{\partial L}{\partial p}} \circ \underbrace{\frac{1}{1+e^{-(\cdot)}}}_{\frac{\partial p}{\partial a}} \circ \underbrace{x^T(\cdot)}_{\frac{\partial a}{\partial \vec{w}}}\Big)(w)$$

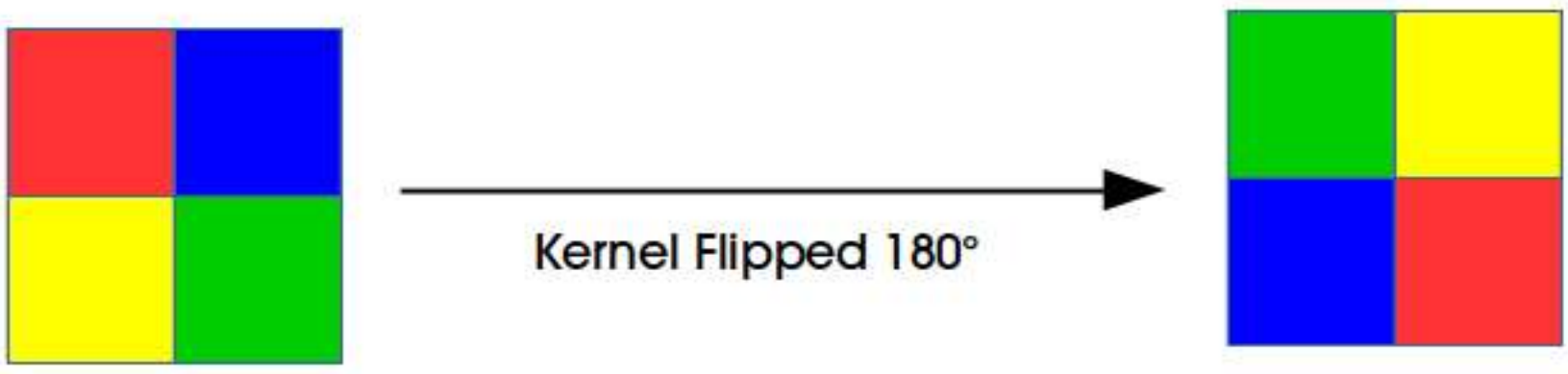
$$\frac{\partial L_i}{\partial \vec{w}} = \left[\frac{1}{p}\right] \cdot \left[\underbrace{\frac{-1}{(1+e^{-a})^2} \cdot -e^{-a}}_{p \cdot (1-p)}\right] \cdot x^T = (1-p)\, x^T$$


Path 1 : $\frac{\partial E}{\partial x_{i-1}}$ from $\frac{\partial E}{\partial x_i}$
$f_i(x_{i-1}, w_i)$
x_{i-1}
x_i
w_i
Path 2
$\frac{\partial E}{\partial w_i}$ from $\frac{\partial E}{\partial x_i}$

Plan for Today

- Topics:
  - (Finish) Computing Gradients
    - Backprop in Conv Layers
  - **Forward mode vs Reverse mode AD**
  - Modern CNN Architectures

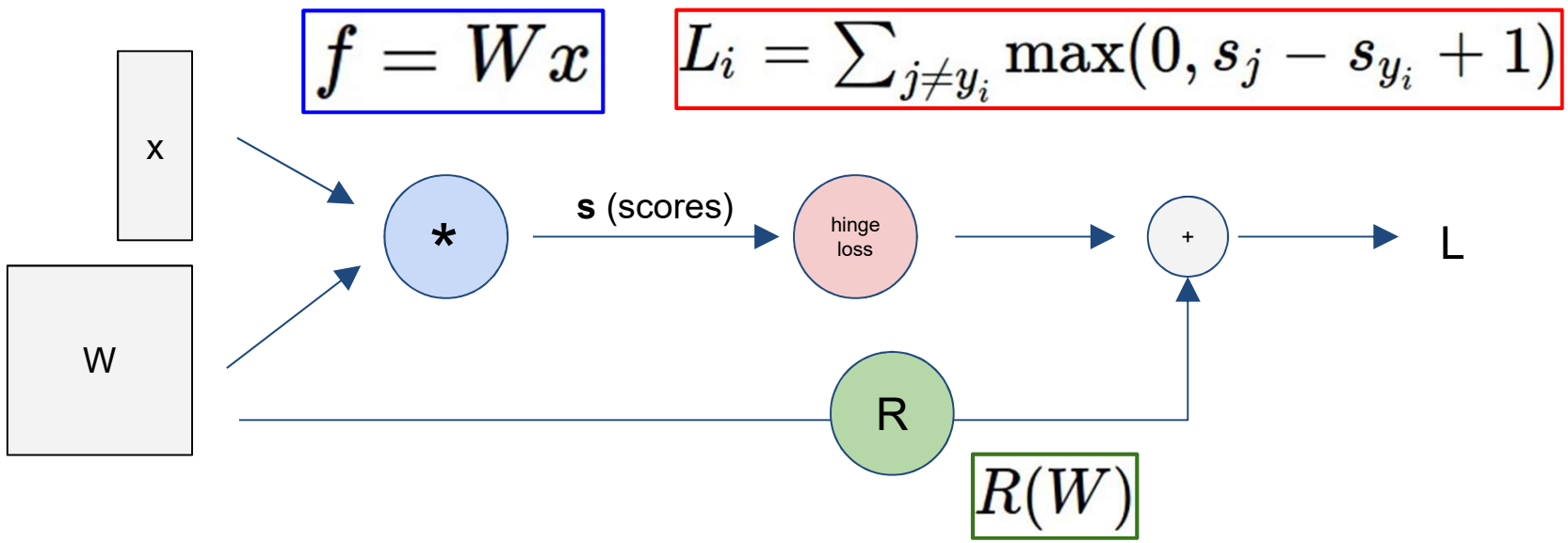
Backprop in Convolutional Layers



How do we compute gradients?

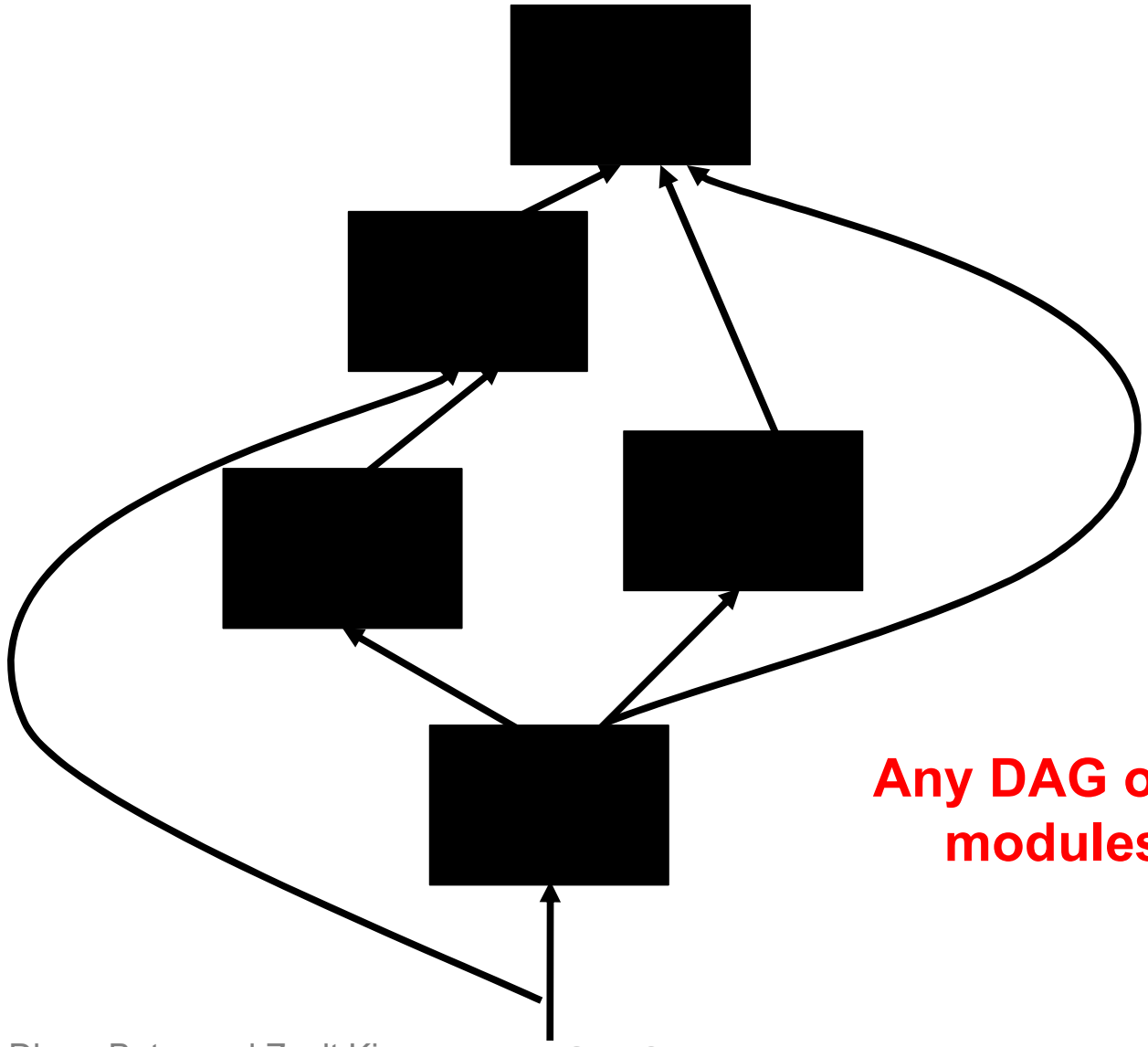
- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

Computational Graph



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

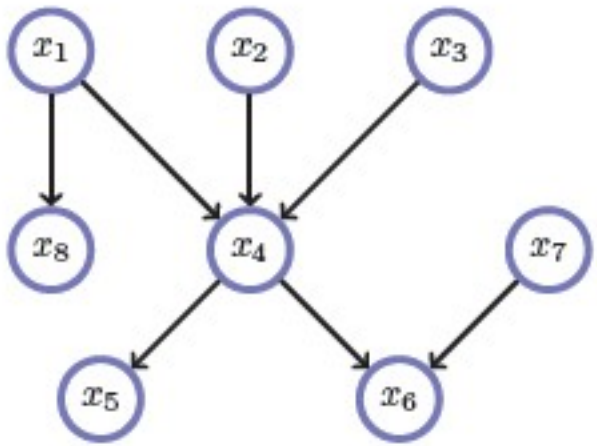
Computational Graph

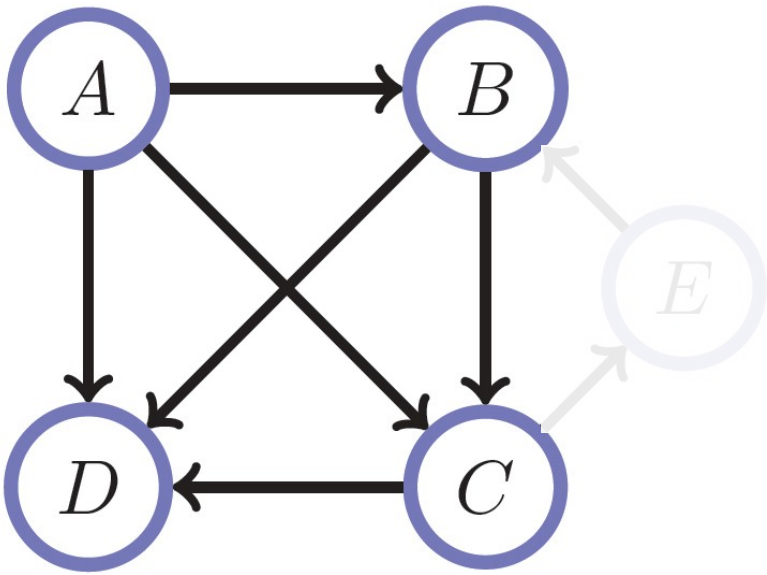


Any DAG of differentiable modules is allowed!

Directed Acyclic Graphs (DAGs)

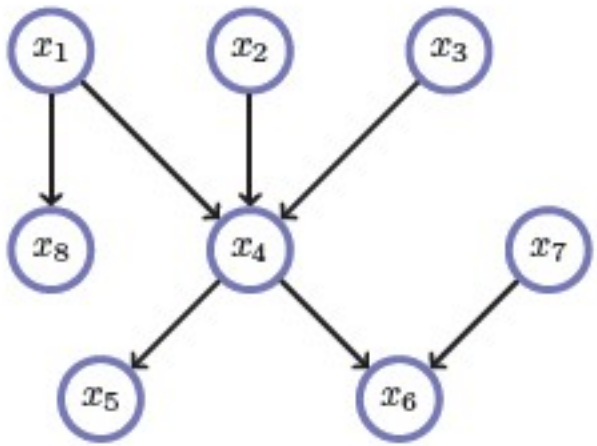
- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

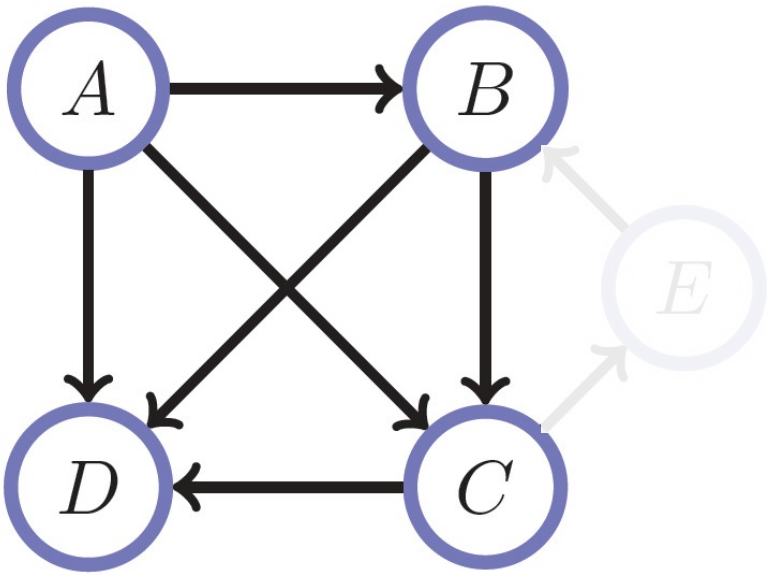




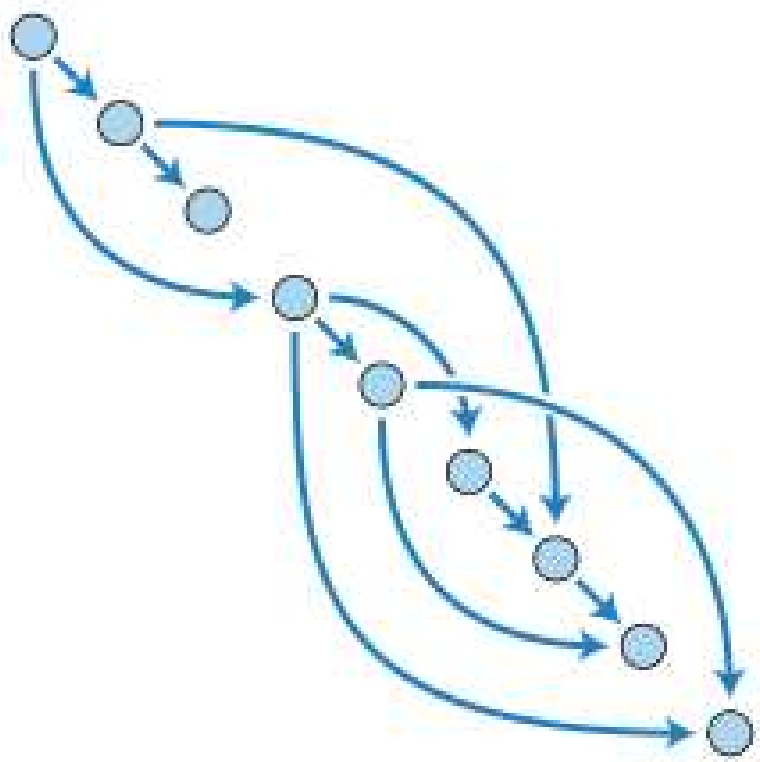
Directed Acyclic Graphs (DAGs)

- Concept
  - Topological Ordering





Directed Acyclic Graphs (DAGs)



Numerical vs Analytic Gradients

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.
This is called a **gradient check.**

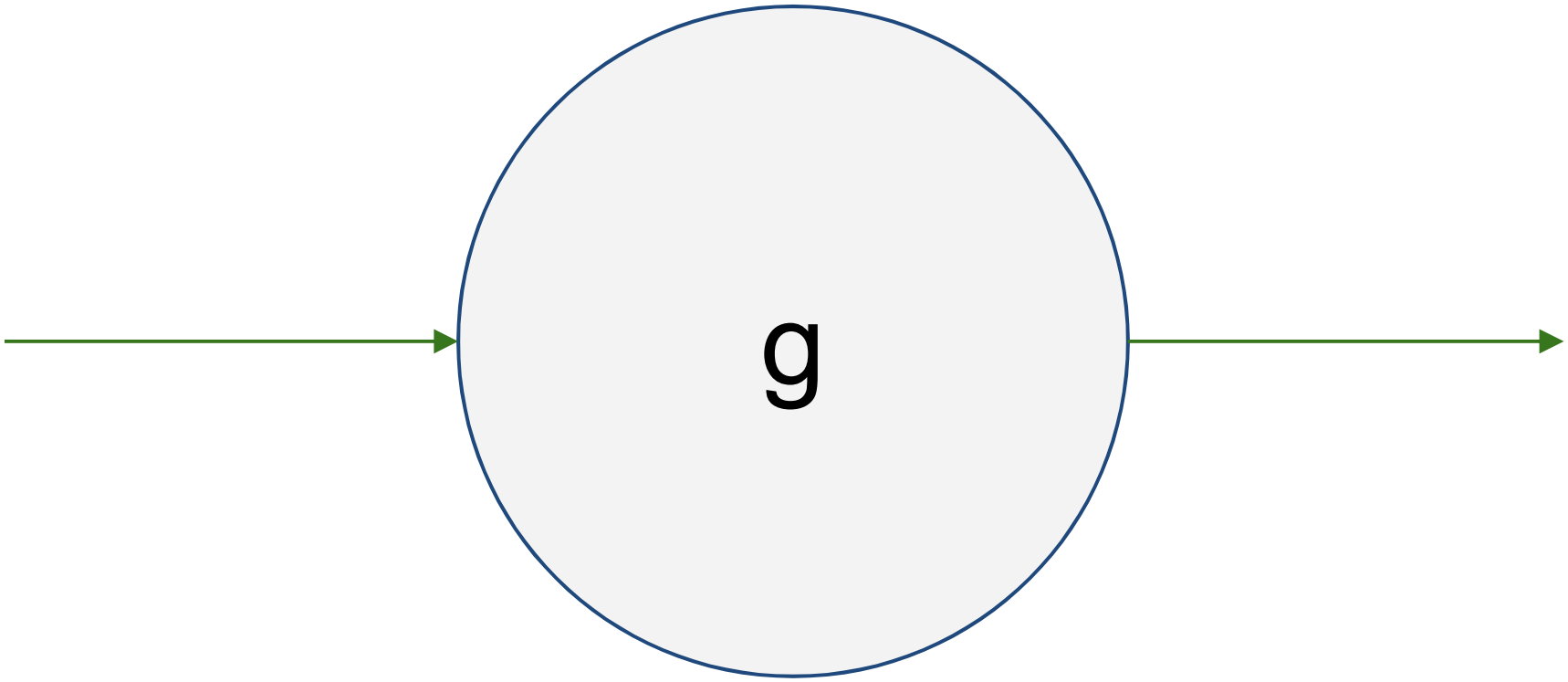
How do we compute gradients?

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

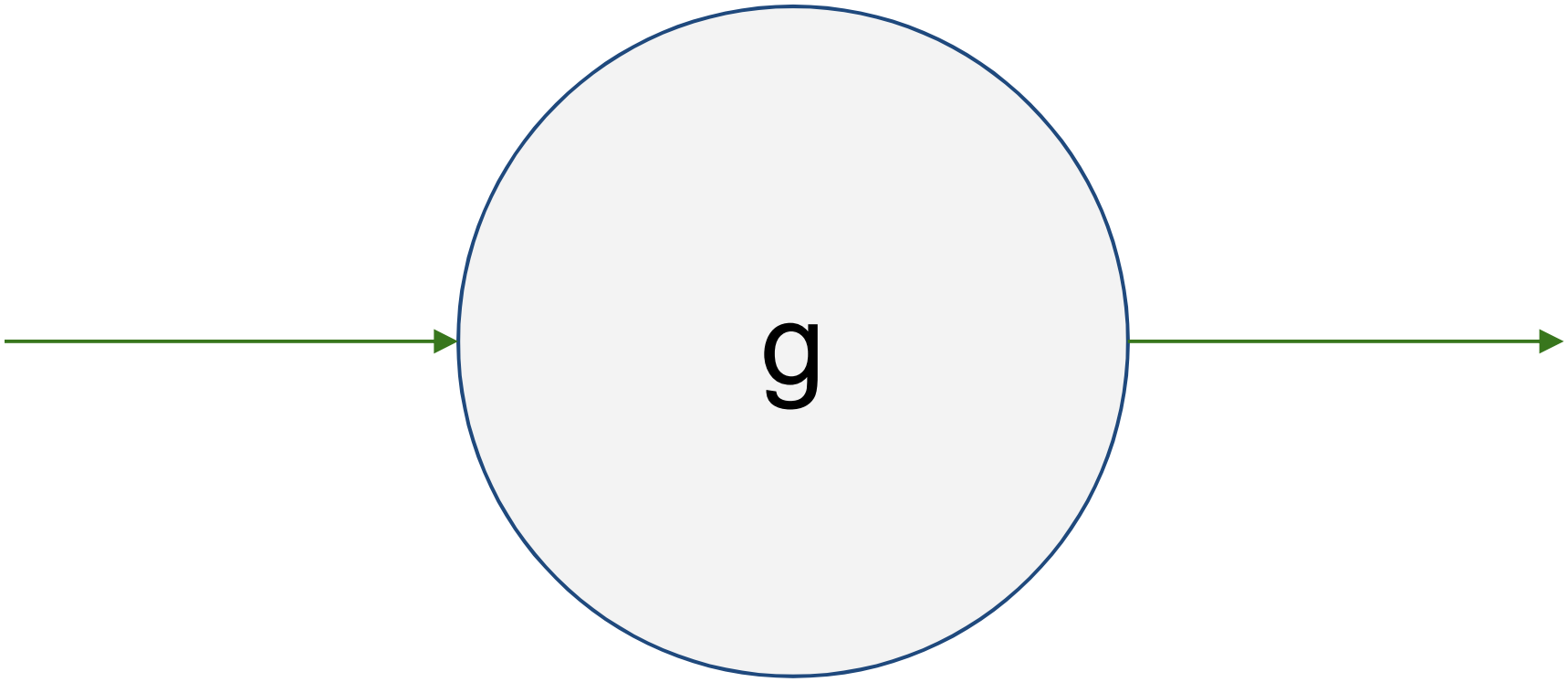
Forward mode vs Reverse Mode

- Key Computations

Forward mode AD

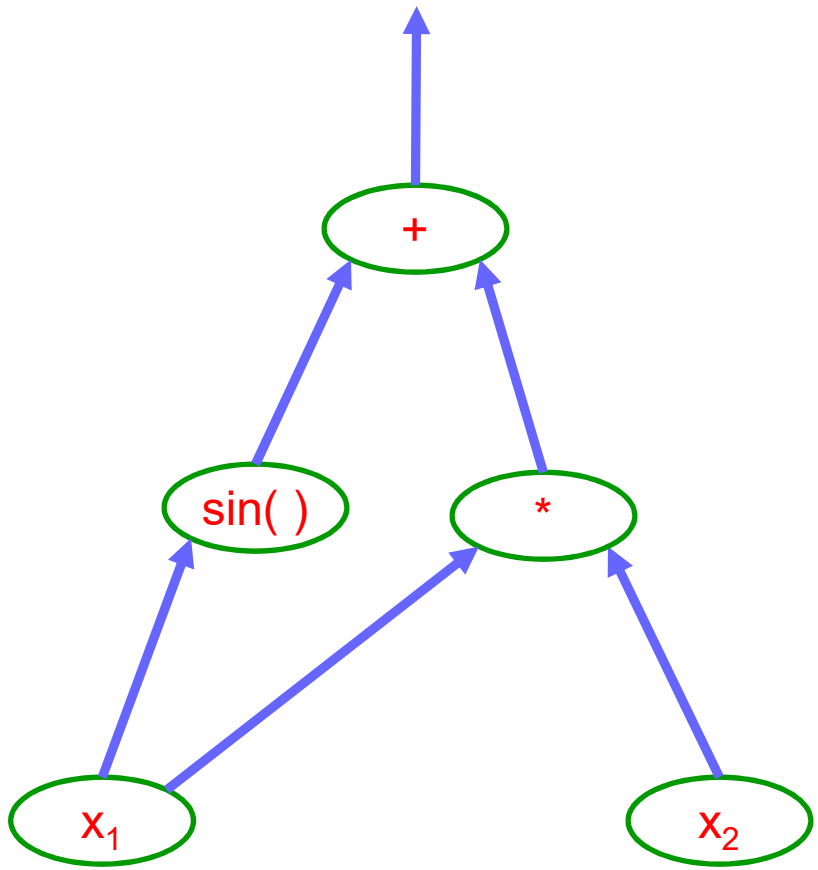


Reverse mode AD



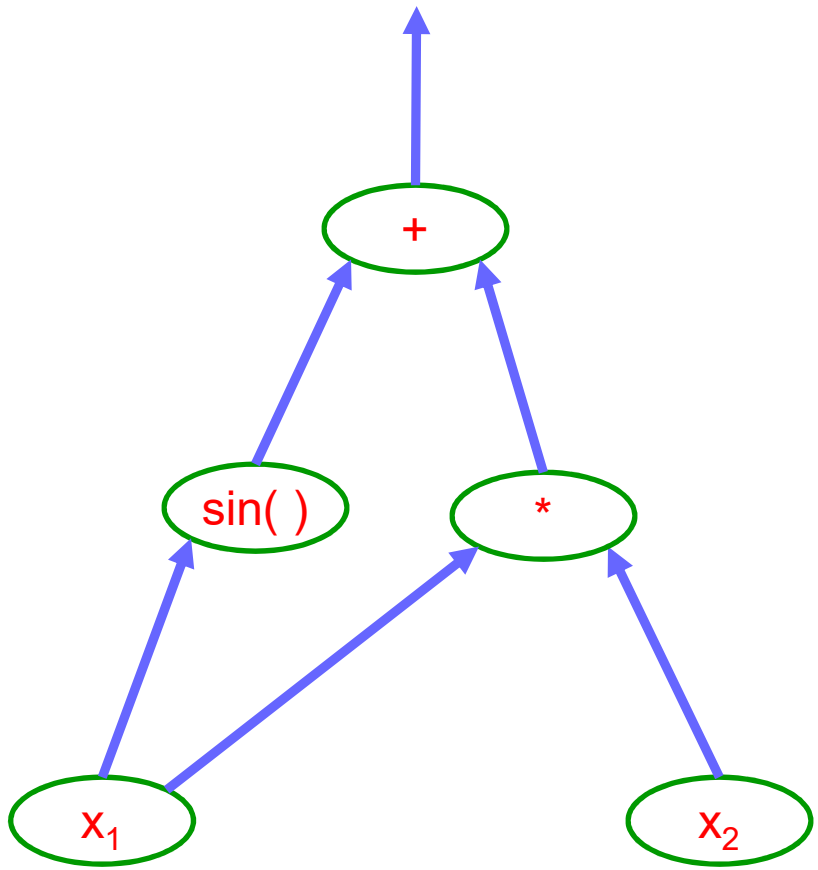
Example: Forward mode AD

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



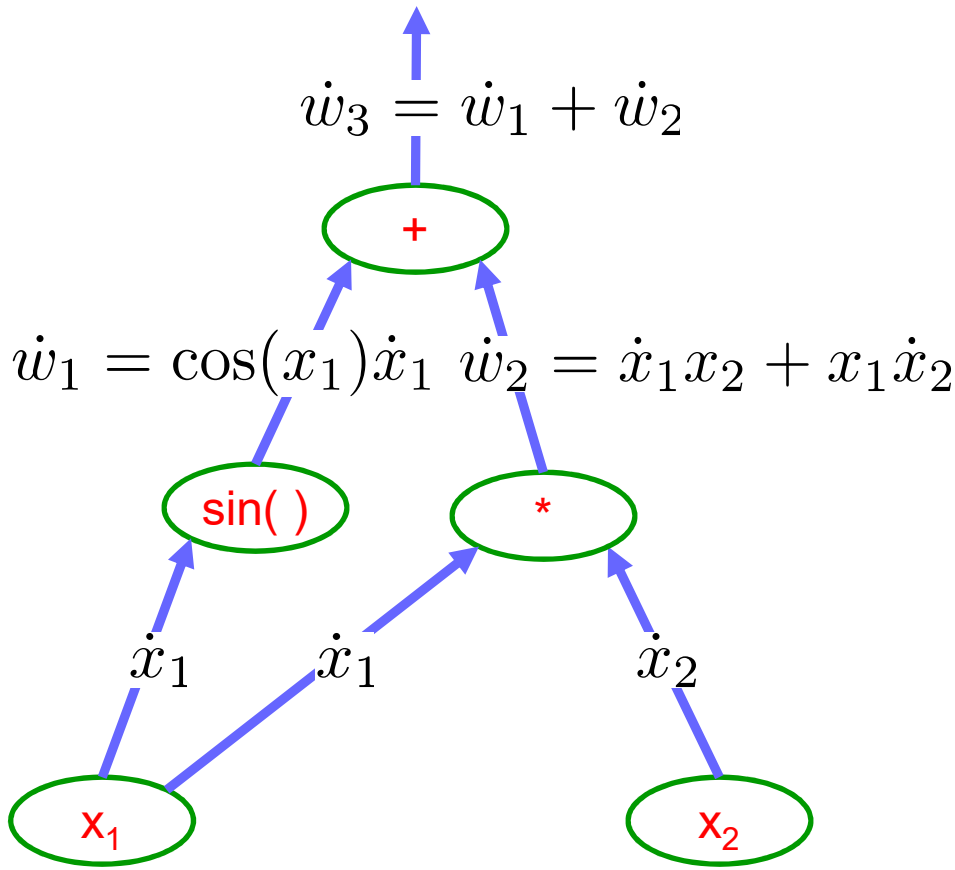
Example: Forward mode AD

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



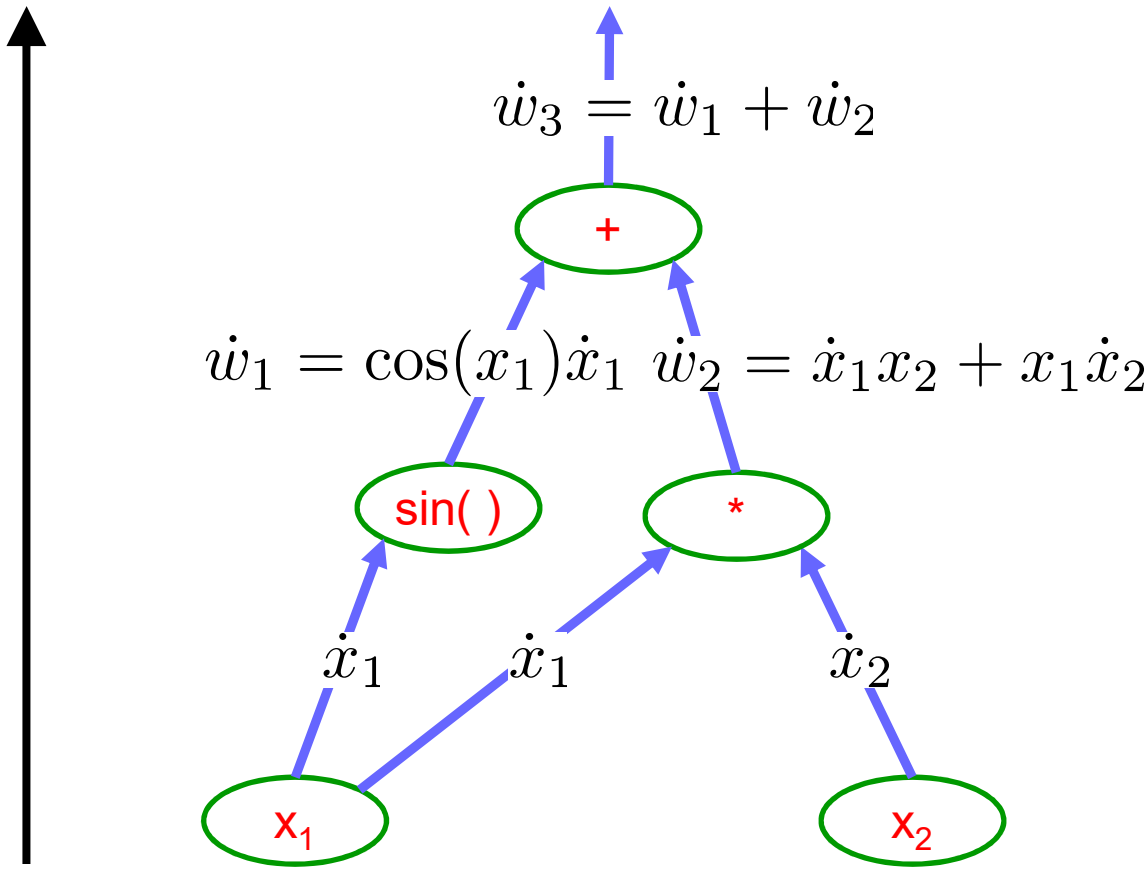
Example: Forward mode AD

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



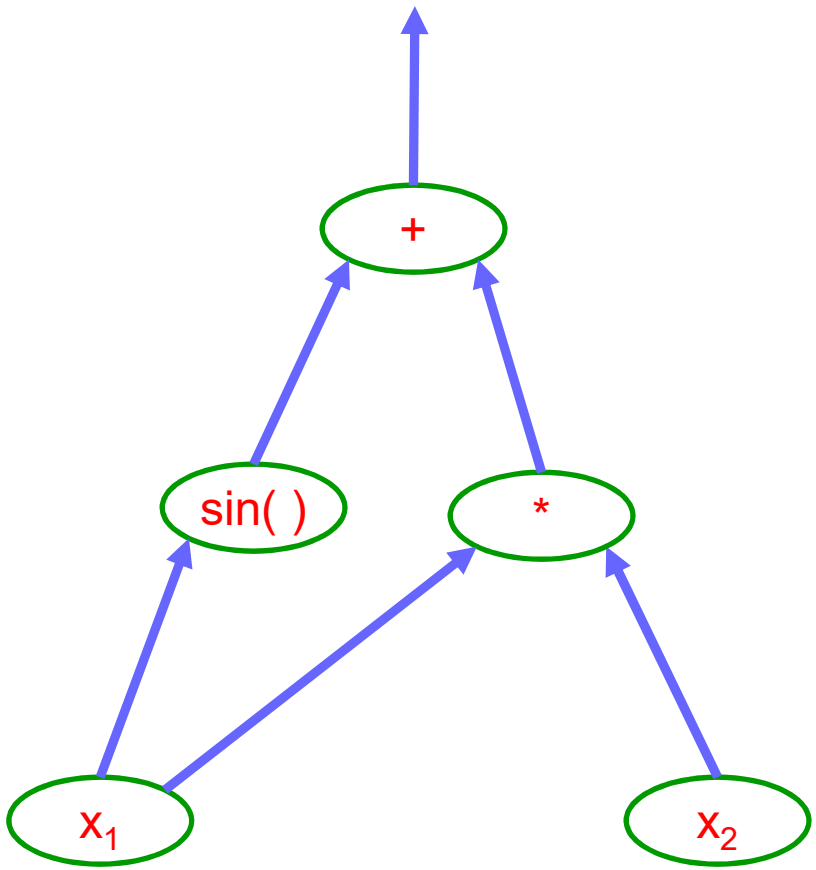
Example: Forward mode AD

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



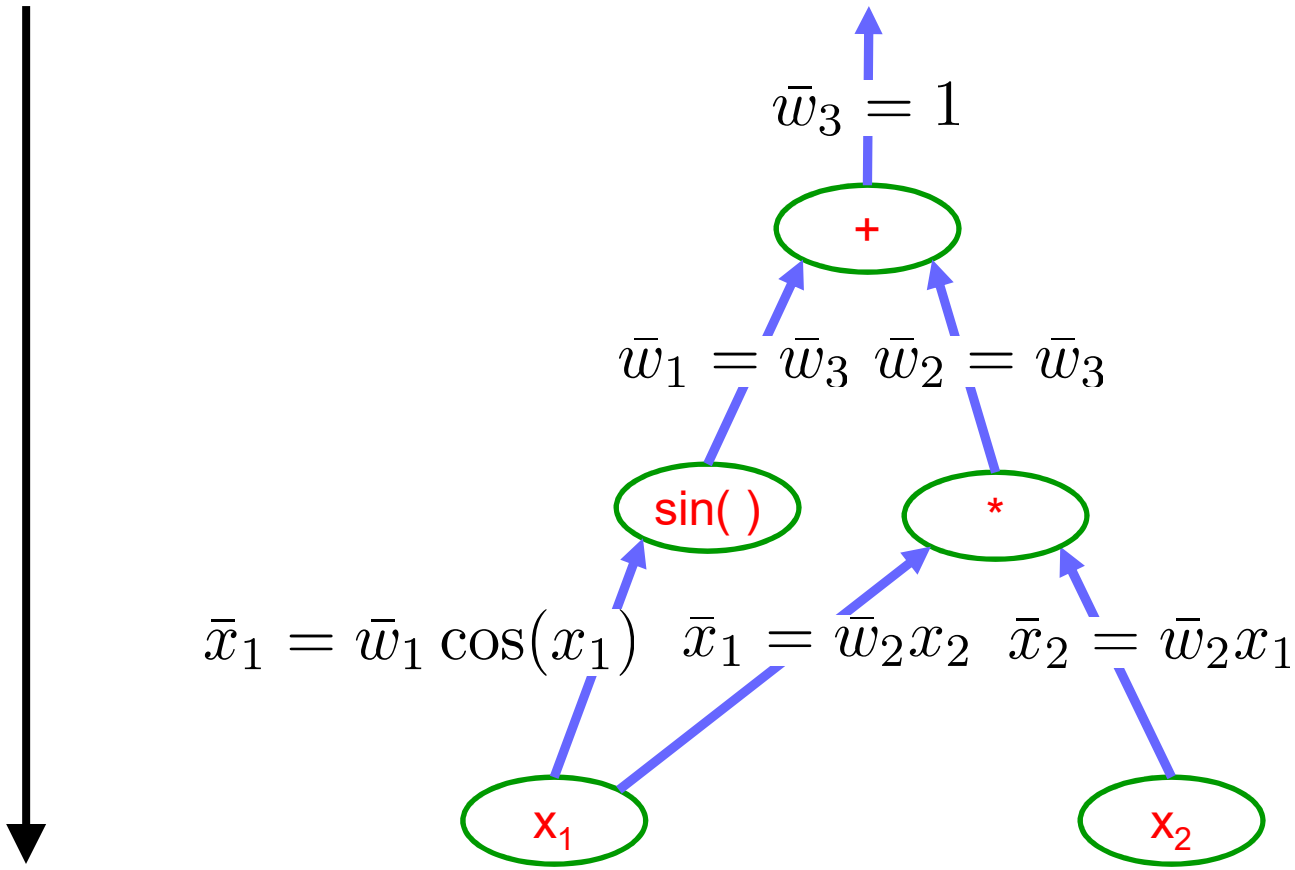
Example: Reverse mode AD

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

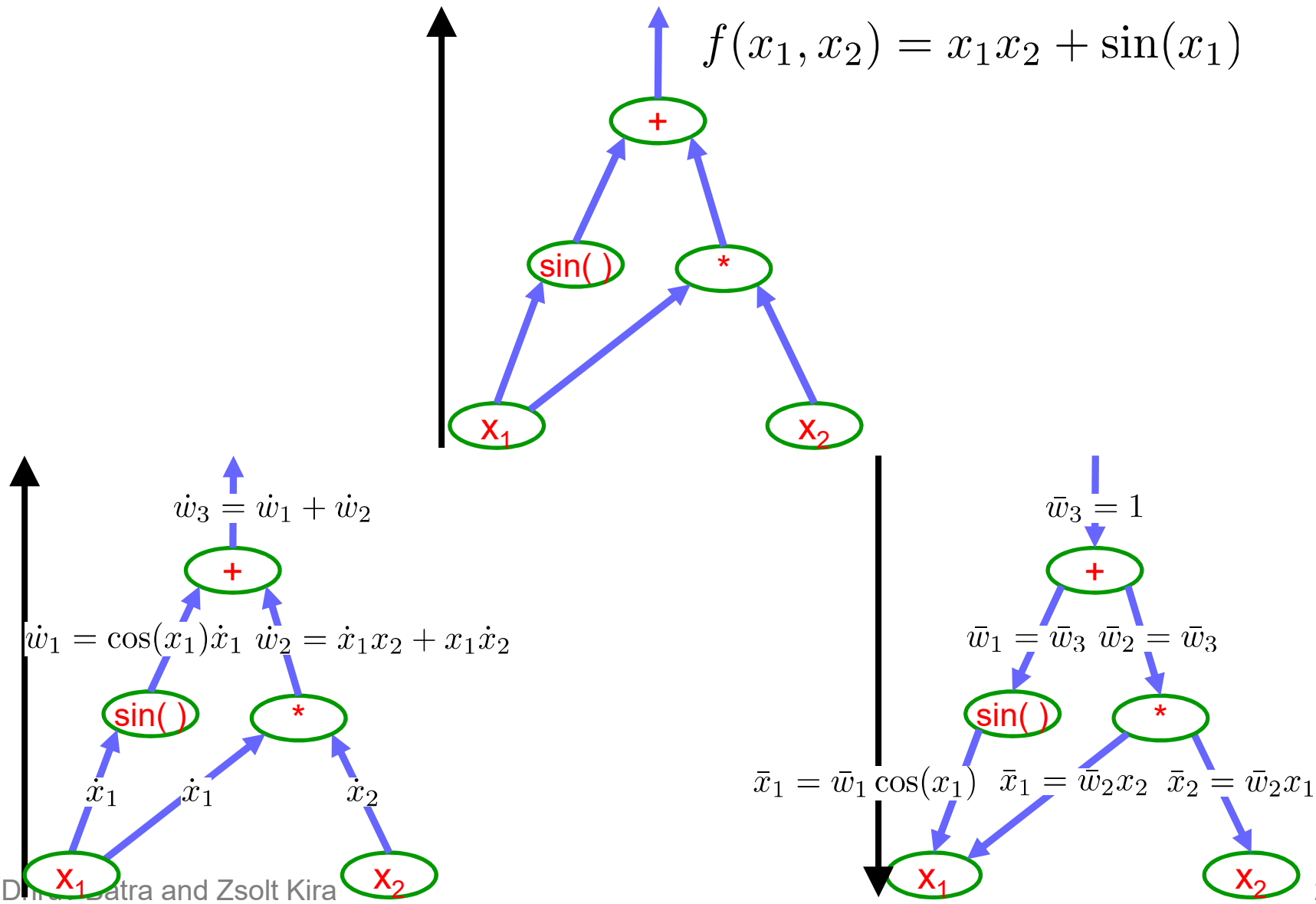


Example: Reverse mode AD

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

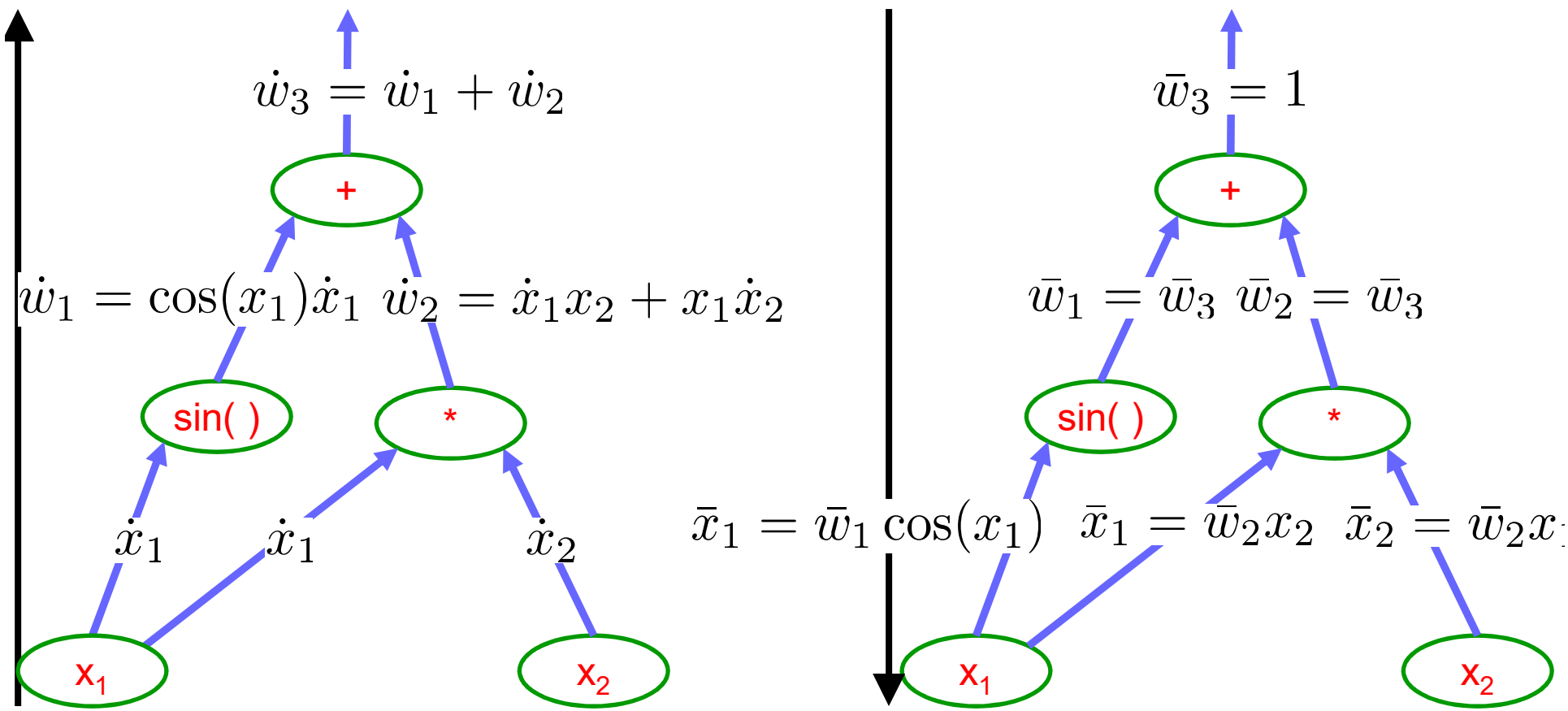


Forward Pass vs
Forward mode AD vs Reverse Mode AD



Forward mode vs Reverse Mode

- What are the differences?



Forward mode vs Reverse Mode

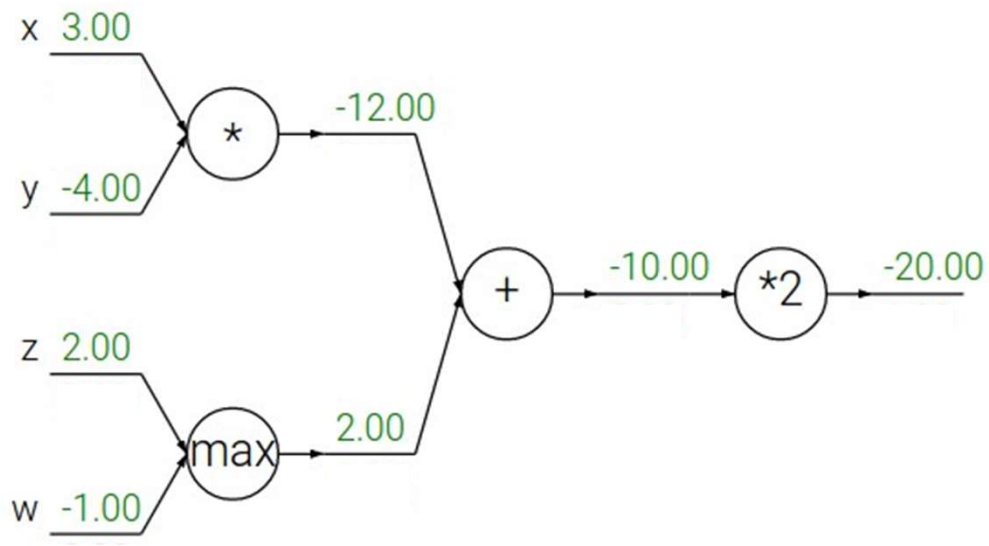
- What are the differences?

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

Forward mode vs Reverse Mode

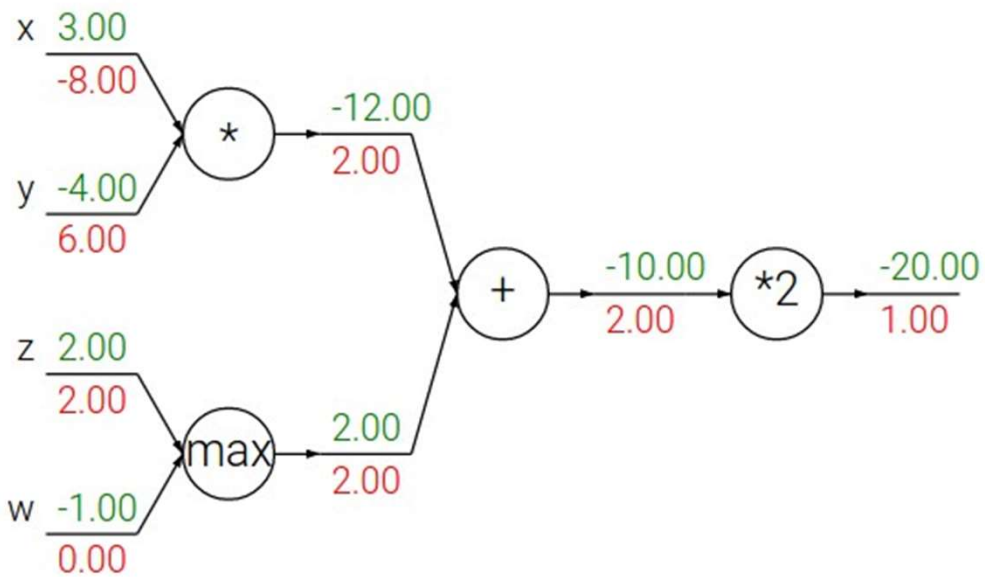
- What are the differences?
- Which one is faster to compute?
  - Forward or backward?
- Which one is more memory efficient (less storage)?
  - Forward or backward?

Patterns in backward flow



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

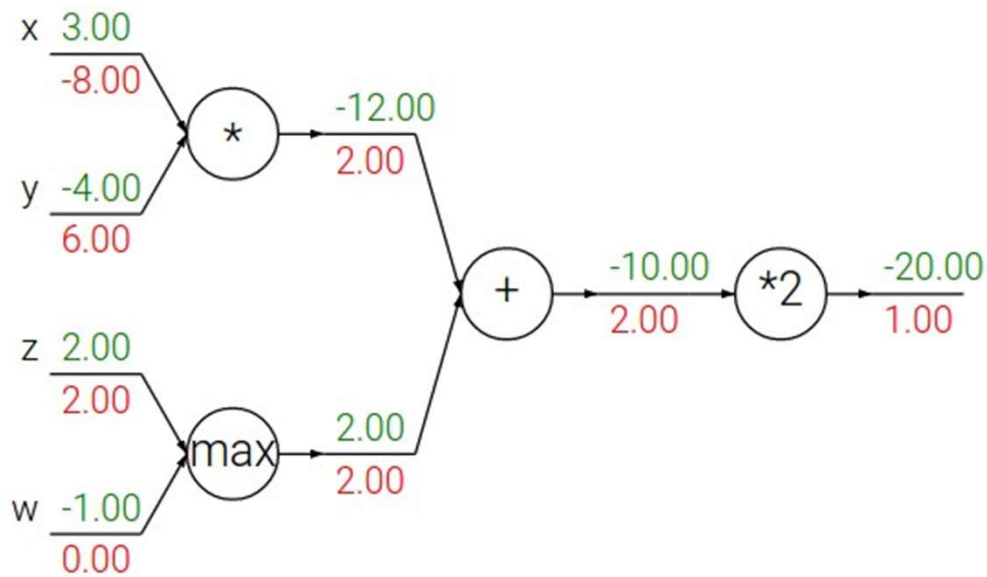
Patterns in backward flow



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

Patterns in backward flow

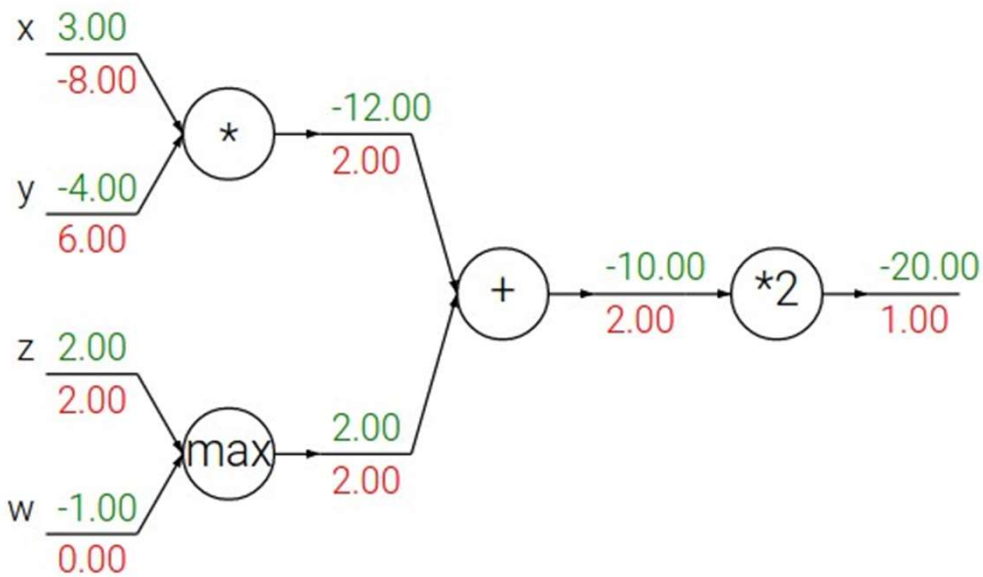
add gate: gradient distributor



Patterns in backward flow

add gate: gradient distributor

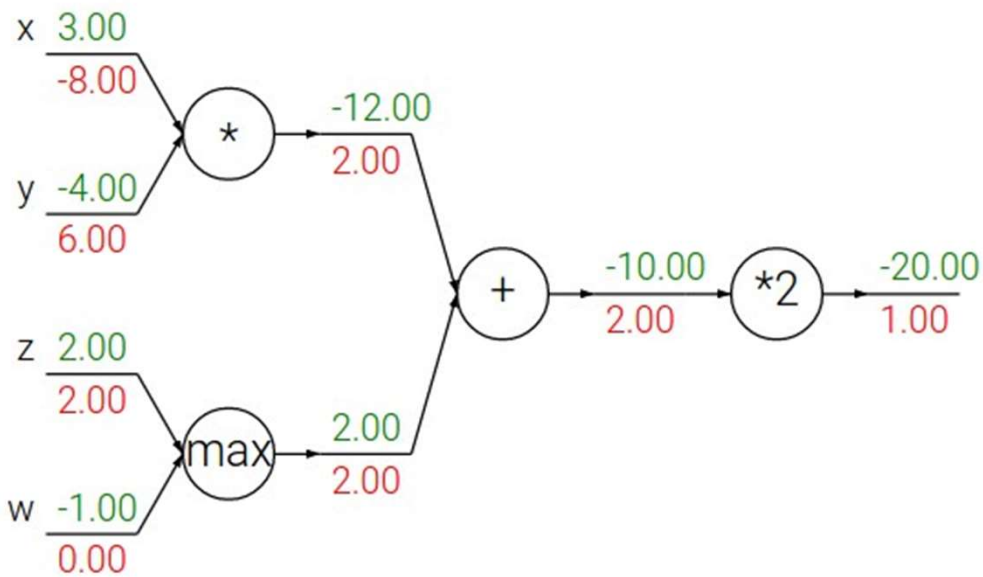
Q: What is a **max** gate?



Patterns in backward flow

add gate: gradient distributor

max gate: gradient router



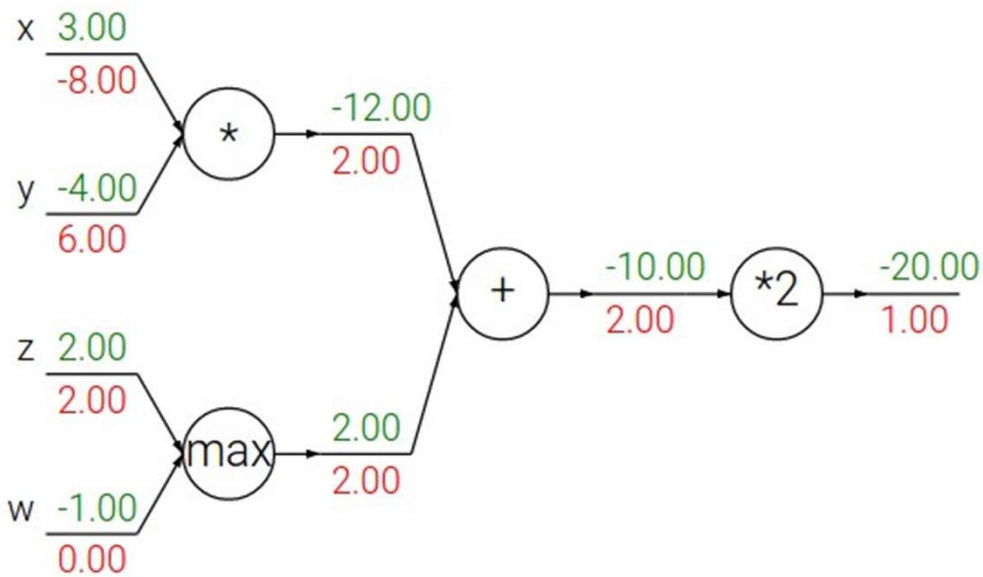
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

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



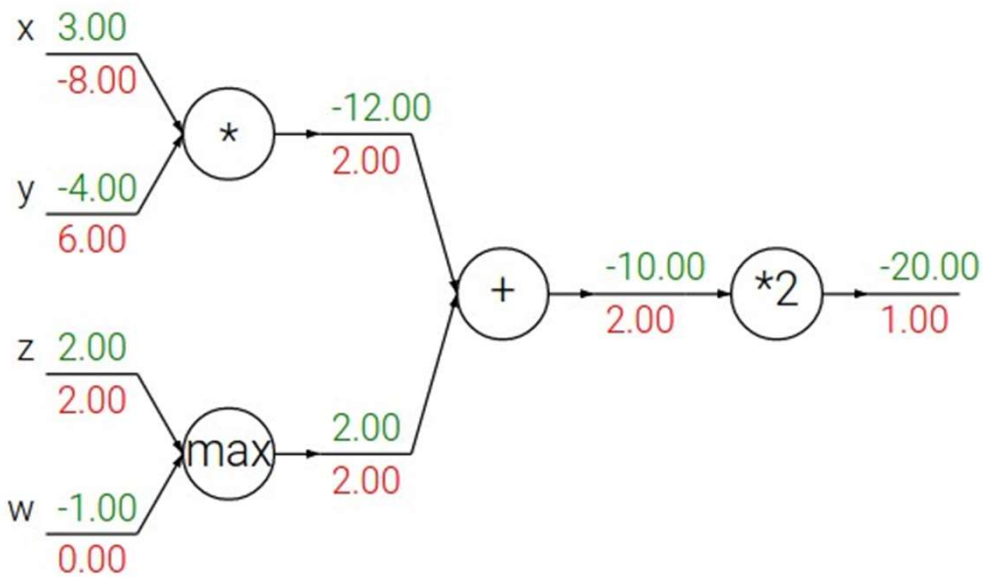
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

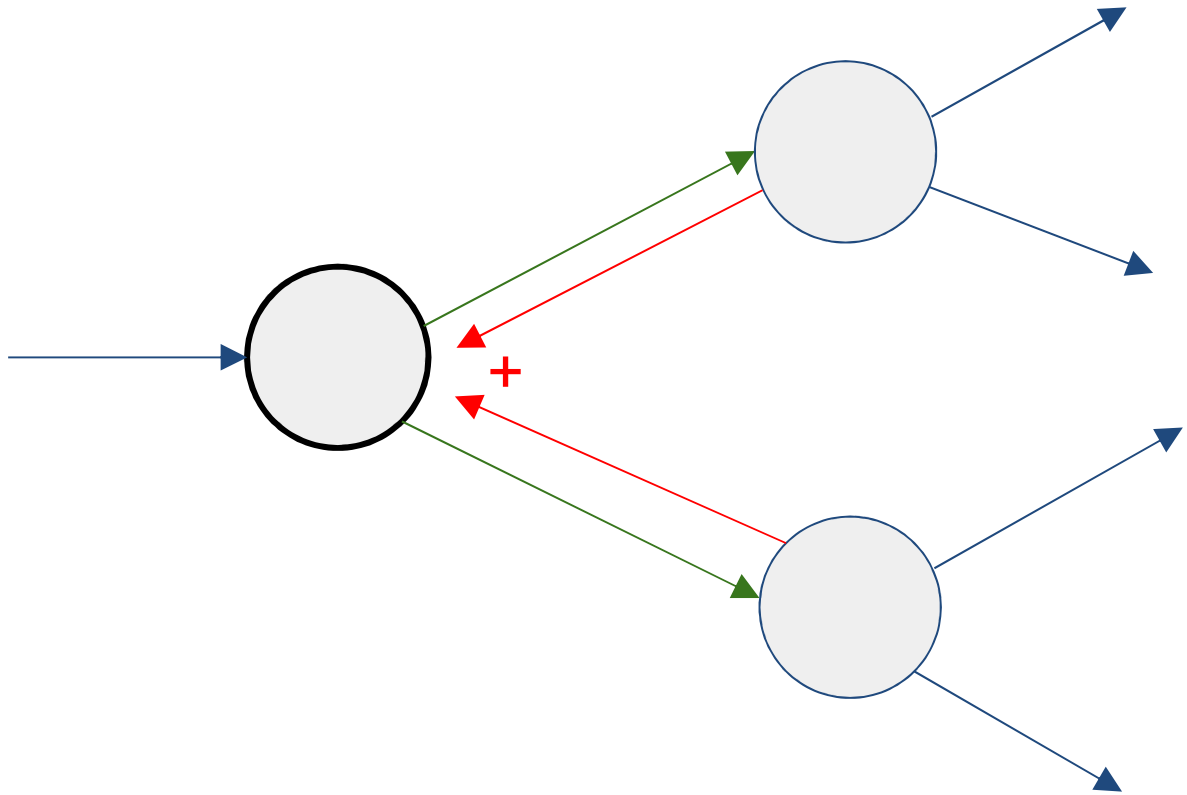
max gate: gradient router

mul gate: gradient switcher



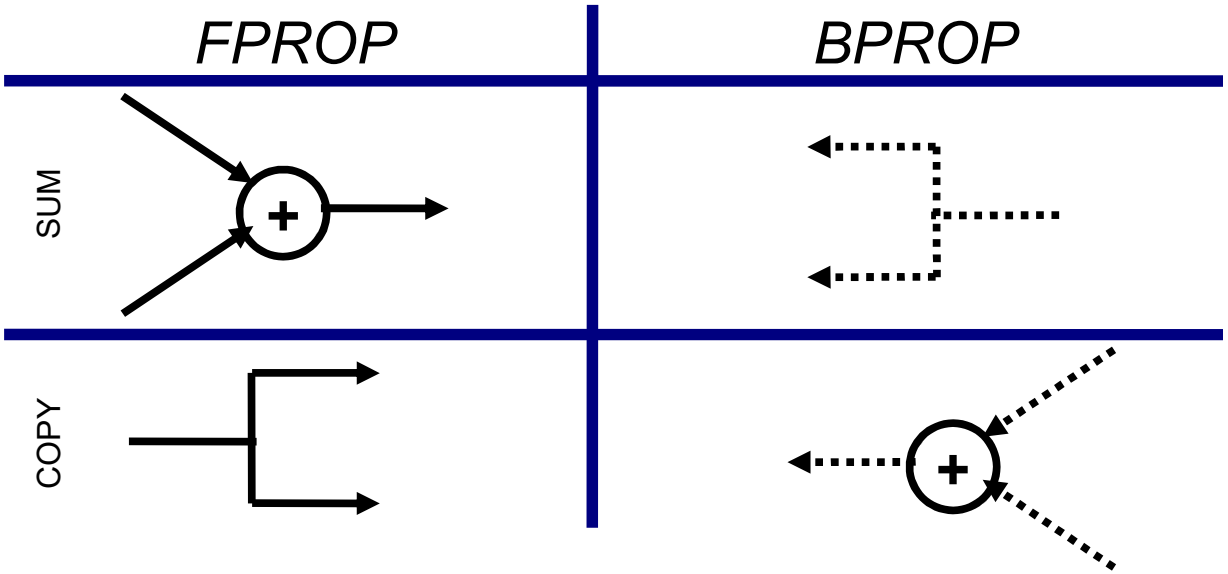
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Gradients add at branches

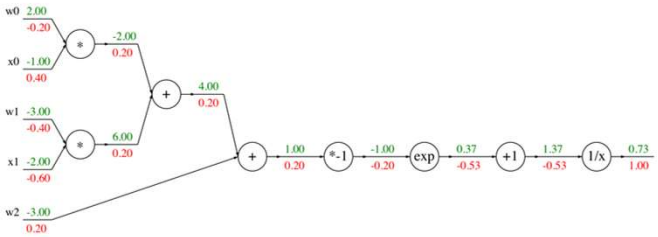


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Duality in Fprop and Bprop



Modularized implementation: forward / backward API

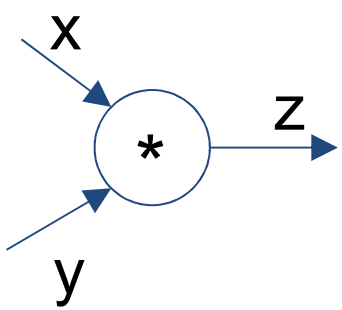


Graph (or Net) object *(rough psuedo code)*

```
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



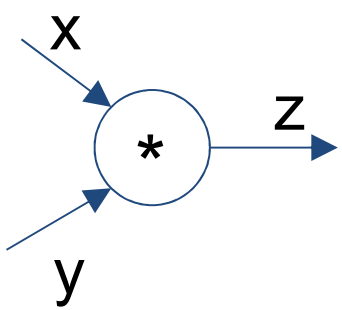
(x,y,z are scalars)

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

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

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

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]
```

Example: Caffe layers

Branch: master | caffe / src / caffe / layers / | Create new file | Upload files | Find file | History

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

```cpp
#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
```

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

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

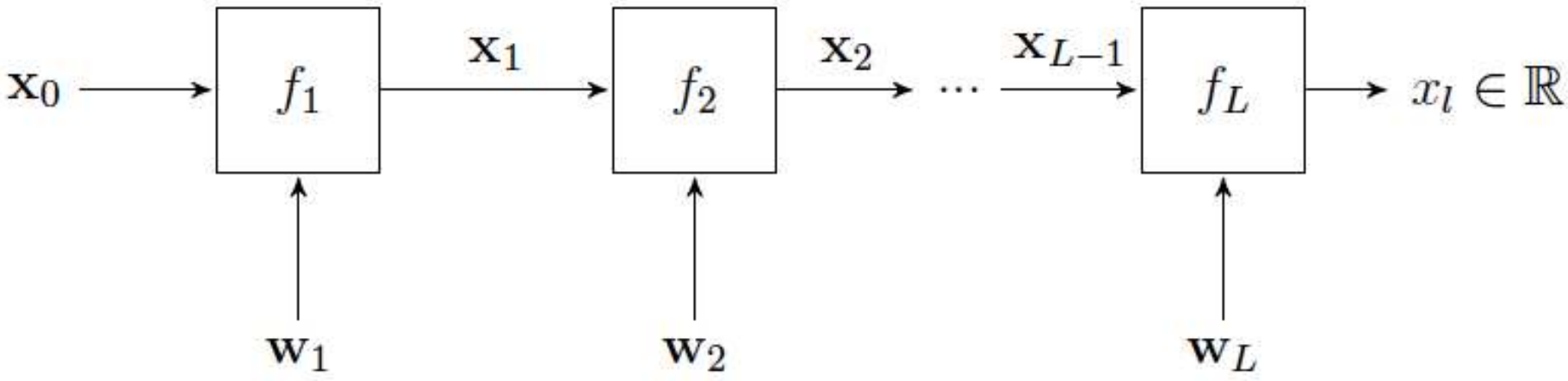


Figure Credit: Andrea Vedaldi