CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Computational Graphs
  - Notation + example
- (Finish) Computing Gradients
  - Forward mode vs Reverse mode AD
  - Patterns in backprop
  - Backprop in FC+ReLU NNs

Dhruv Batra

Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 10/02, 11:55pm
    - https://www.cc.gatech.edu/classes/AY2019/cs7643_fall/assets/hw1.pdf
    - https://www.cc.gatech.edu/classes/AY2019/cs7643_fall/hw1-q6/

Recap from last time

Strategy: **Follow the slope**

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

Gradient Descent

```python
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

backprop

$w^{(0)} = \text{init}$

for $t = 1 \ldots$ tired

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

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

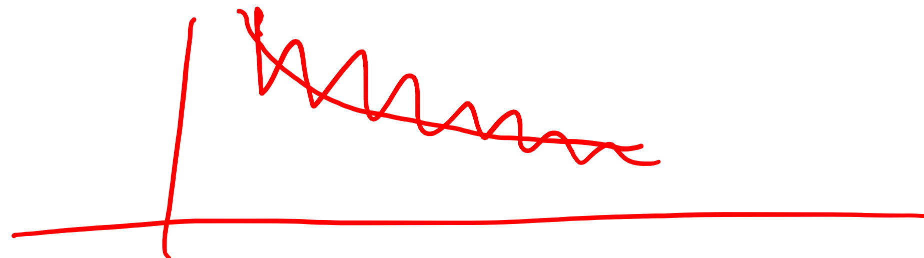
$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```



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

How do we compute gradients?

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

$l_1 = x$
$l_{n+1} = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4v(1 - v)
  v
```

or, in closed-form,

```
f(x):
  64x (1-x) (1-2x)^2 (1-8x+8x^2)^2
```

Coding

Symbolic Differentiation of the Closed-form

```
f'(x):
 128x(1 - x)(-8 + 16 x)(1 - 2
   x)^2 (1 - 8 x + 8 x^2) + 64 (1
   - x)(1 - 2 x)^2 (1 - 8 x + 8
   x^2)^2 - 64x(1 - 2 x)^2 (1 - 8
   x + 8 x^2)^2 - 256x(1 - x)(1 -
   2 x)(1 - 8 x + 8 x^2)^2
```

f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation

Numerical Differentiation

```
f'(x):
  (v,v') = (x,1)
  for i = 1 to 3
    (v,v') = (4v(1-v), 4v'-8vv')
  (v,v')
```

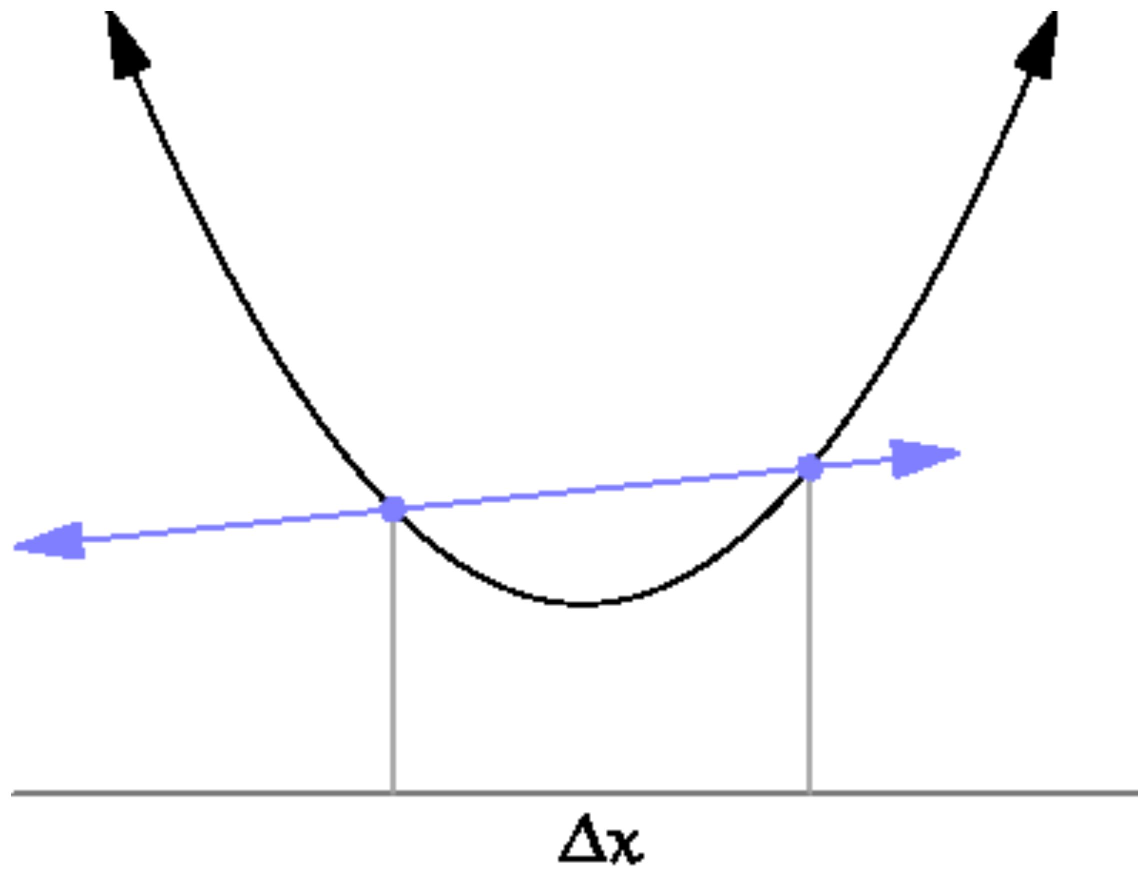
f'(x_0) $= f'(x_0)$
Exact

```
f'(x):
  h = 0.000001
  (f(x + h) - f(x)) / h
```

f'(x_0) $\approx f'(x_0)$
Approximate

How do we compute gradients?

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



By Brnbrnz (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

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

?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (second dim)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

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

?,…]

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

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (third dim):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?,…]

(1.25347 - 1.25347)/0.0001
= 0

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

Numerical vs Analytic Gradients

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

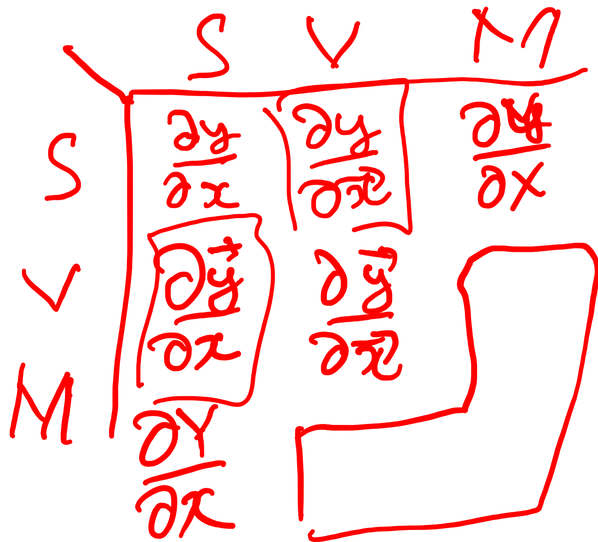
Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

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"

Matrix/Vector Derivatives Notation



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

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

$X, Y \in \mathbb{R}^{m \times n}$

$$\frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_c}{\partial x} \end{bmatrix}$$

num = dim 1

$$\frac{\partial y}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_d} \end{bmatrix}$$

den = dim 2

Matrix/Vector Derivatives Notation

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \left[\frac{\partial y_i}{\partial x_j} \right]_{c \times d}$$

$$\frac{\partial (\vec{x}^T \vec{w})}{\partial \vec{w}} = \left[\frac{\partial (x^T w)}{\partial w_1} \; - \; - \; - \; \frac{\partial (x^T w)}{\partial w_d} \right]$$

$\sum_i x_i w_i$

$$= x^T \qquad [x_1 \; - \; - \; - \; x_d] = x^T$$

Vector Derivative Example

$$\frac{\partial(w^T A x)}{\partial(\vec{w})} = 2w^T A$$

$$y_i = \sum_j a_{ij} x_j$$

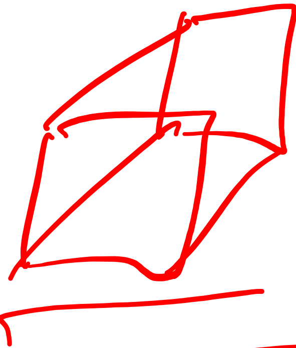
$$\vec{y} = A\vec{x}$$

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

$$i \left[\quad \frac{\partial y_i}{\partial x_j} \; a_{ij} \quad \right]$$

j

Extension to Tensors



$X \in \mathbb{R}^{d_1 \times d_2 \ldots \times d_n}$

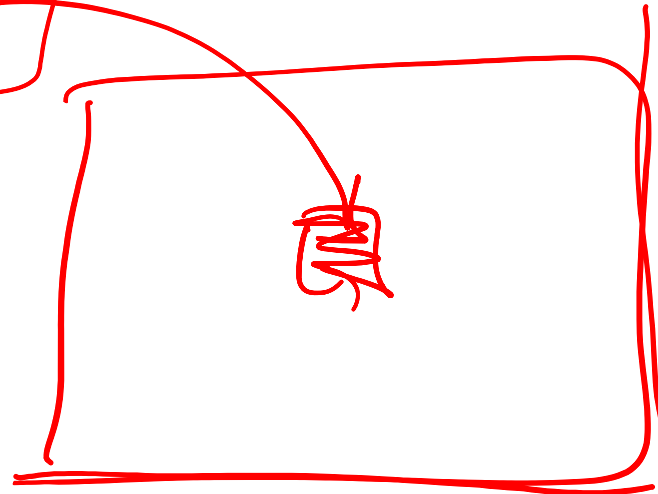
$Y \in \mathbb{R}^{c_1 \times c_2 \ldots c_m}$

$$\frac{\partial Y[i_1, i_2 \ldots i_m]}{\partial X[j_1 \ldots, j_n]}$$

y-vec = Y(:)

x-vec = X(:)

$$\frac{\partial \text{y-vec}}{\partial \text{x-vec}} =$$



Chain Rule: Composite Functions

$$L(x) = f(g(x)) = (f \circ g)(x)$$

$$f(x) = g_\ell(g_{\ell-1} \cdots g_1(x))$$

$$= (g_\ell \circ g_{\ell-1} \cdots \circ g_1)(x)$$

Chain Rule: Scalar Case

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

$$z = g(x)$$

$$y = f(z)$$

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

Chain Rule: Vector Case

$\vec{x} \in \mathbb{R}^d \rightarrow z \in \mathbb{R}^m \rightarrow y \in \mathbb{R}^c$

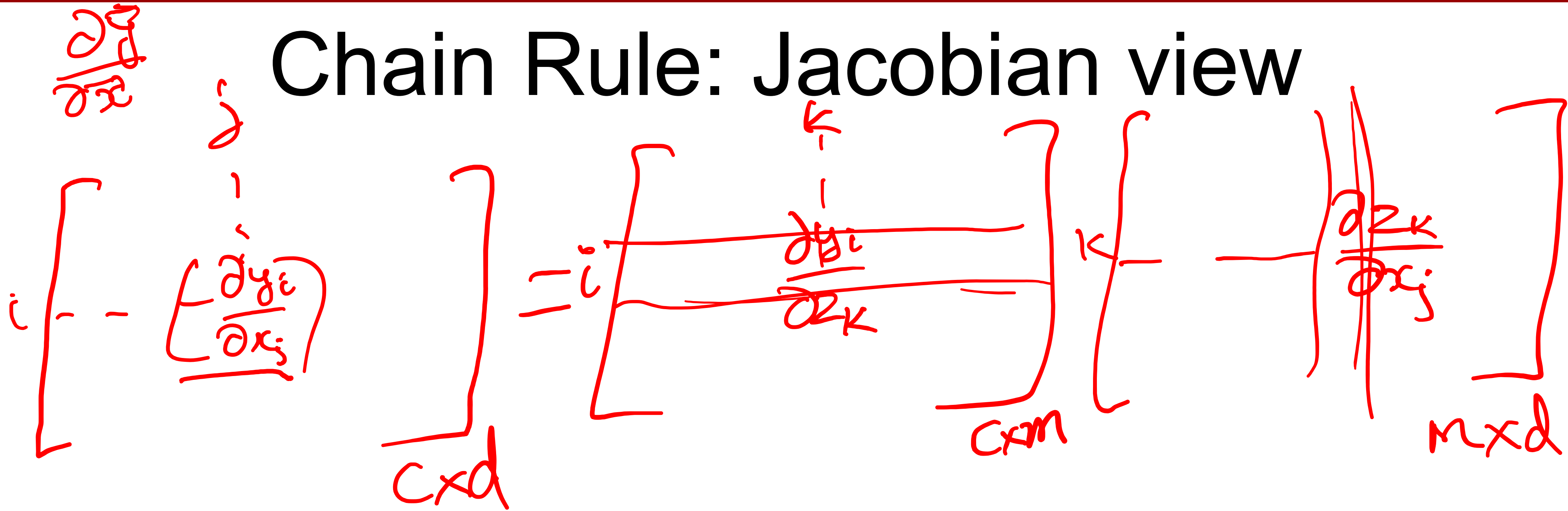
$\vec{z} = g(\vec{x})$ $\quad$ $\vec{y} = f(\vec{z})$

$g: \mathbb{R}^d \rightarrow \mathbb{R}^m$ $\quad$ $f: \mathbb{R}^m \rightarrow \mathbb{R}^c$

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

$$J_{f \circ g} \quad J_f \quad J_g$$

Chain Rule: Jacobian view

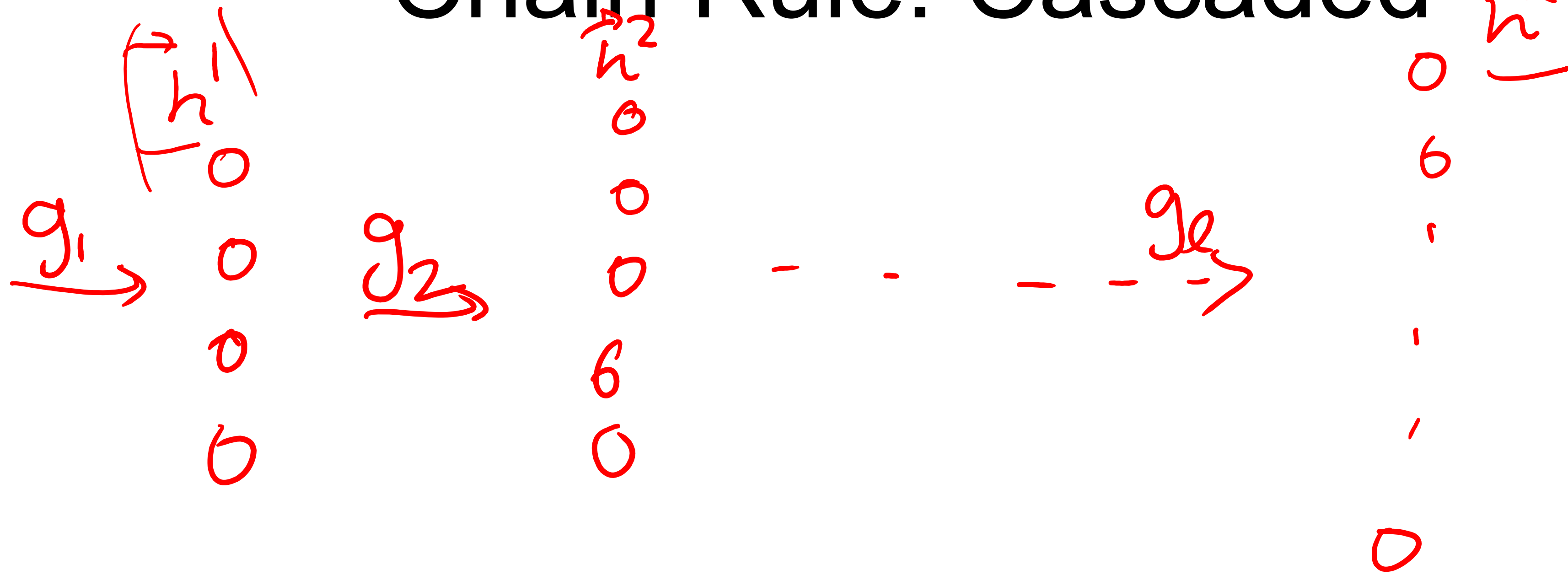


$$\frac{\partial y_i}{\partial x_j} = \sum_k \frac{\partial y_i}{\partial z_k} \frac{\partial z_k}{\partial x_j}$$

Chain Rule: Graphical view

$$\frac{\partial y_i}{\partial x_j} = \sum_{paths} \frac{\partial y_i}{\partial z_k} \frac{\partial z_k}{\partial x_j}$$

Chain Rule: Cascaded



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

$\longrightarrow O(ld^2)$ $\qquad$ $\longleftarrow O(ld^3)$

$$= [\quad]_{1 \times d} \, [\quad]_{d \times d} \cdots [\quad][\quad]_{d \times d}$$

$O(d^2)$ $\qquad$ $O(d^3)$

Chain Rule: How should we multiply?

Plan for Today

- (Finish) Computational Graphs
  - Notation + example
- (Finish) Computing Gradients
  - Forward mode vs Reverse mode AD
  - Patterns in backprop
  - Backprop in FC+ReLU NNs

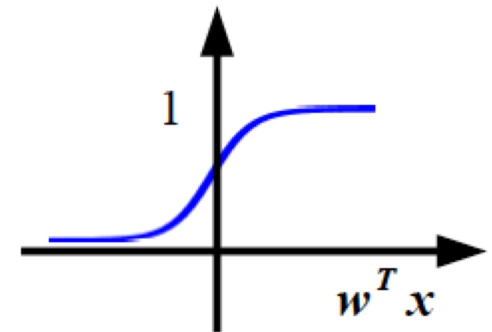
Linear Classifier: Logistic Regression

Input: $\boldsymbol{x} \in R^D$

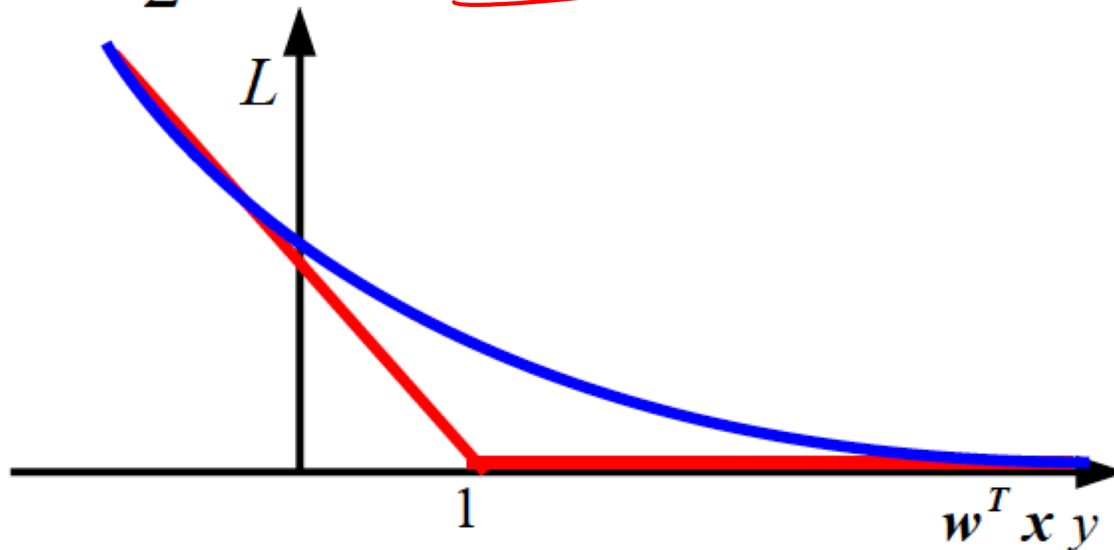
Binary label: $y \in \{-1, +1\}$

Parameters: $\boldsymbol{w} \in R^D$

Output prediction: $p(y=1|\boldsymbol{x}) = \dfrac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}}}$



Loss: $L = \dfrac{1}{2}\|\boldsymbol{w}\|^2 - \lambda \log(p(y|\boldsymbol{x}))$



Log Loss

Ranzato

Logistic Regression Derivatives

$$L_i(\vec{w}) = -\log\left(\frac{1}{1+e^{-w^T x}}\right)$$

$$\frac{\partial L_i}{\partial \vec{w}} = \frac{-1}{1/(1+e^{-w^T x})} \cdot \frac{-1}{(1+e^{-w^T x})^2} \cdot \left(-e^{-w^T x}\right) \cdot x^T$$

Convolutional network (AlexNet)

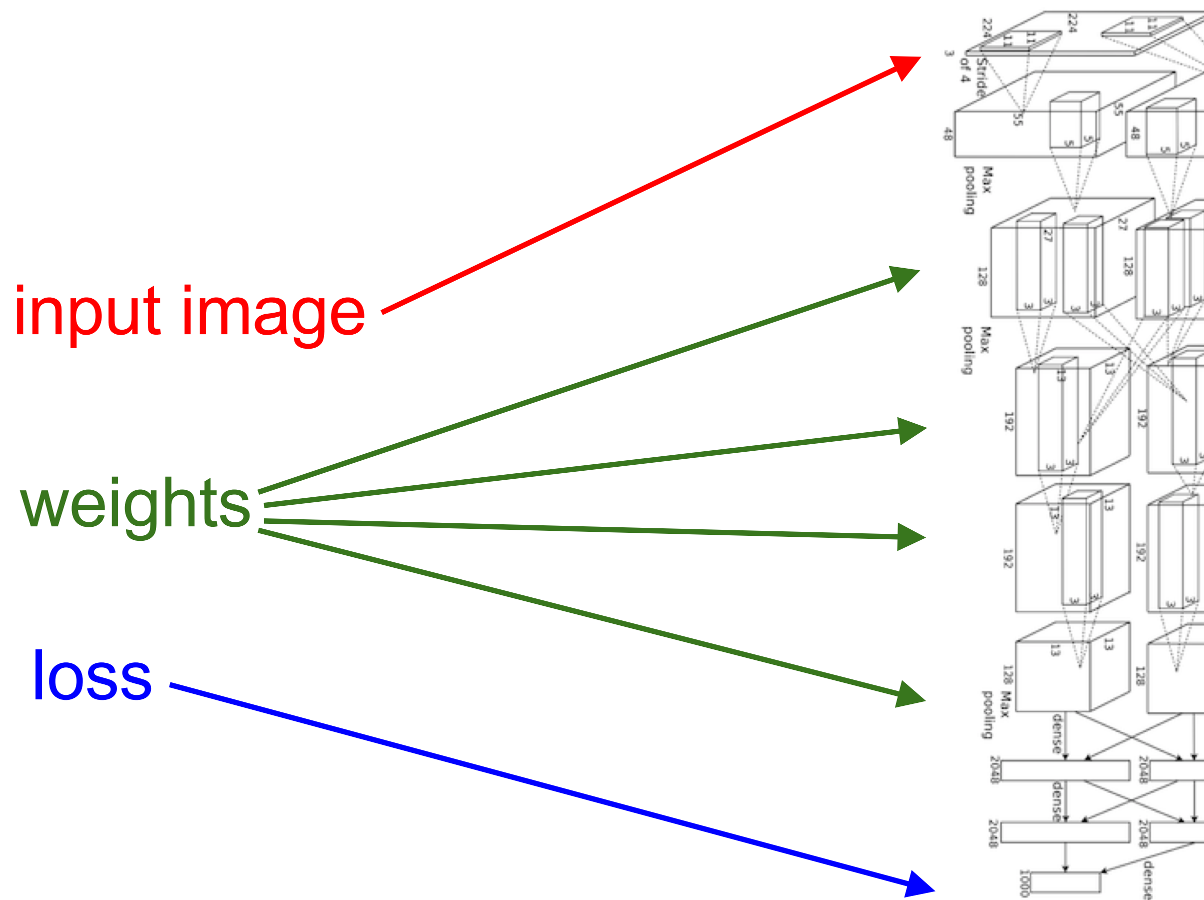


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

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

Neural Turing Machine

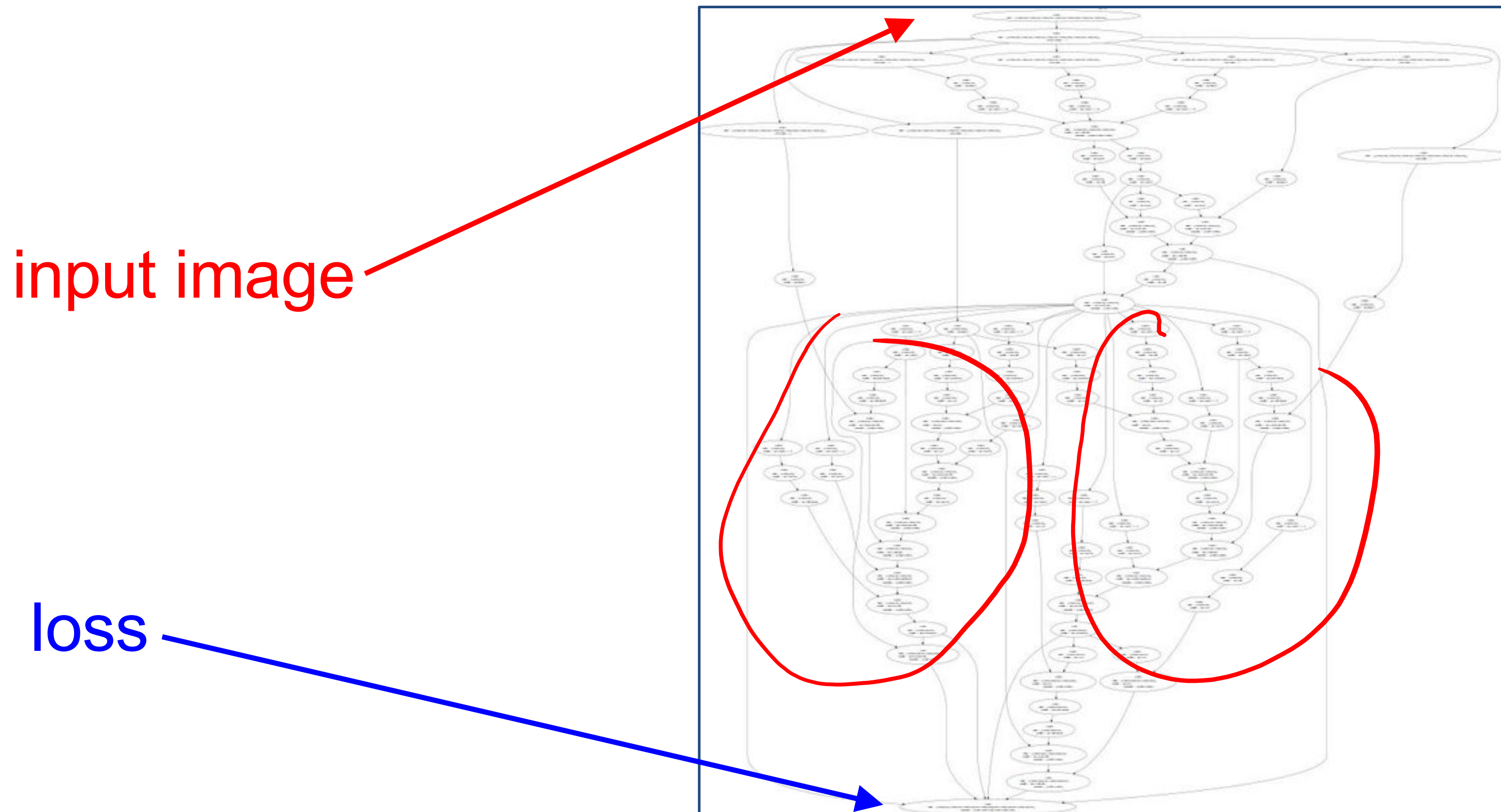
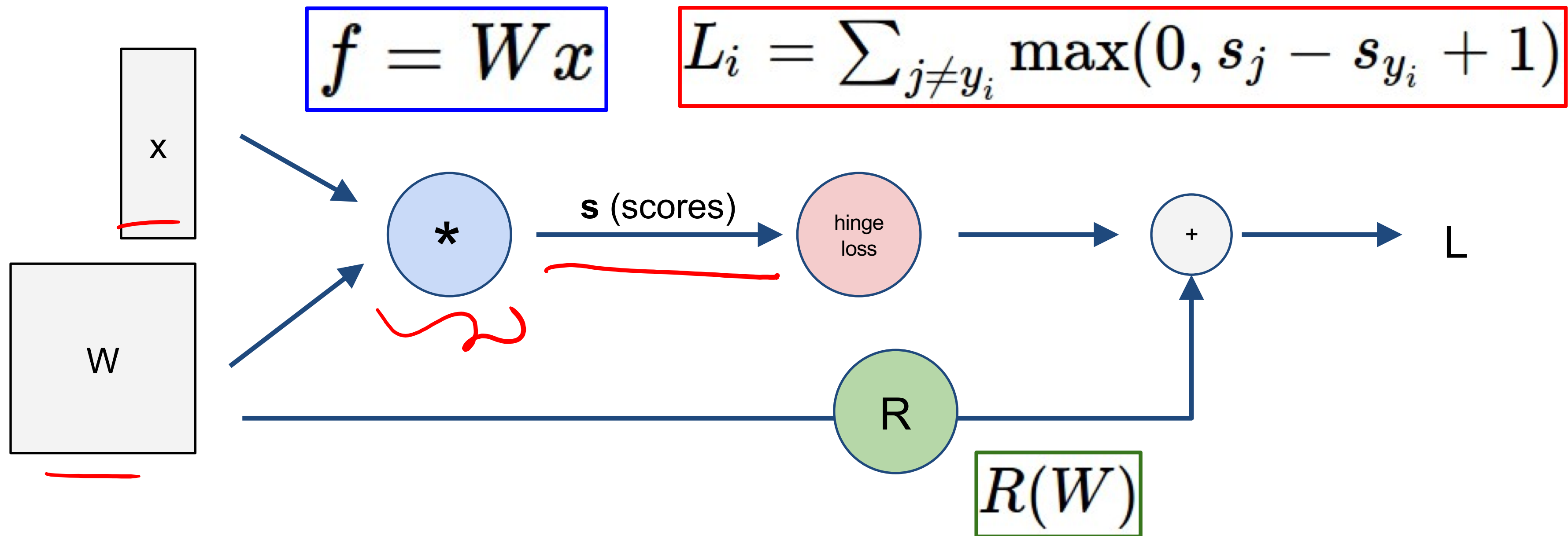


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

How do we compute gradients?

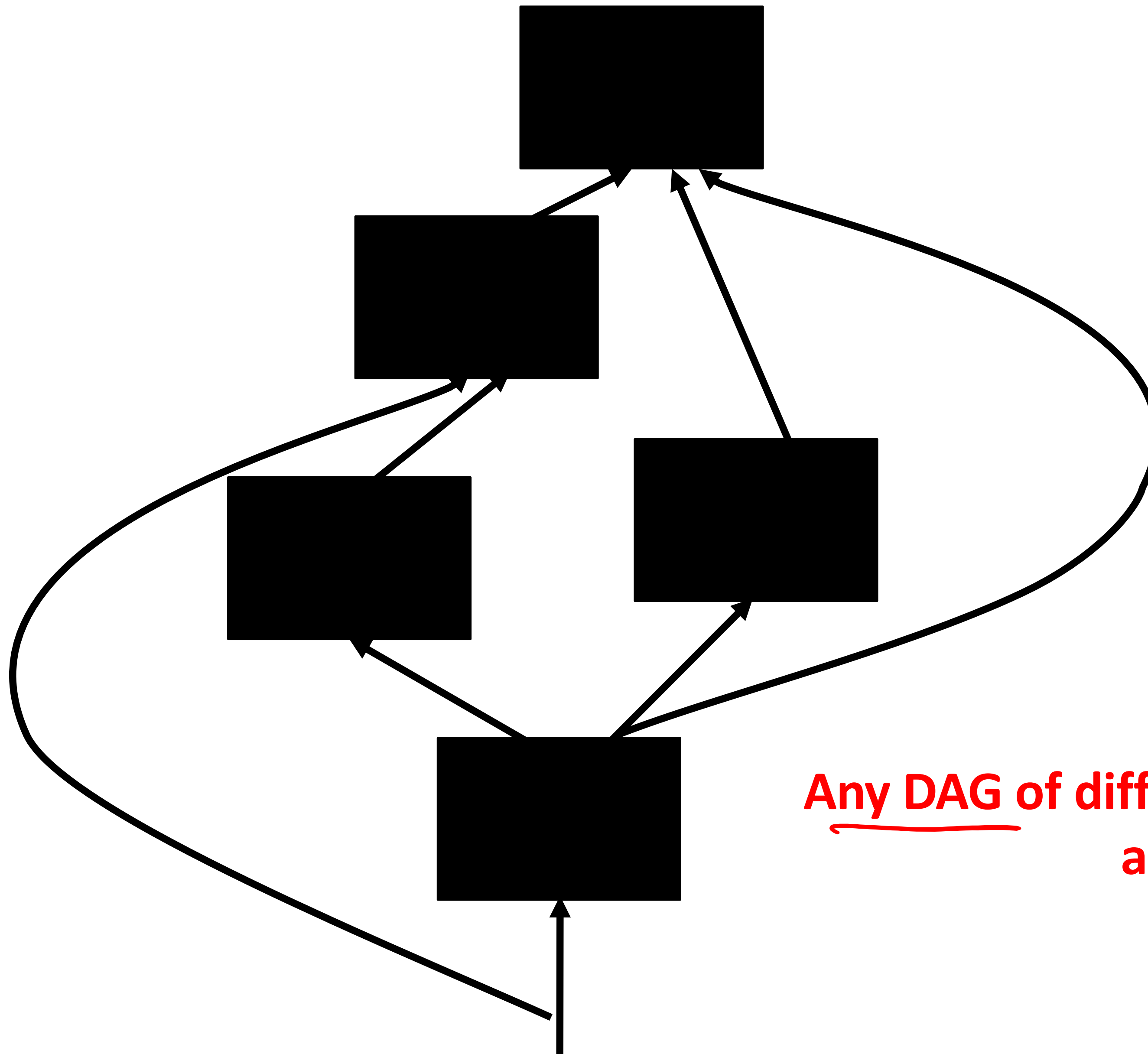
- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

Computational Graph



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

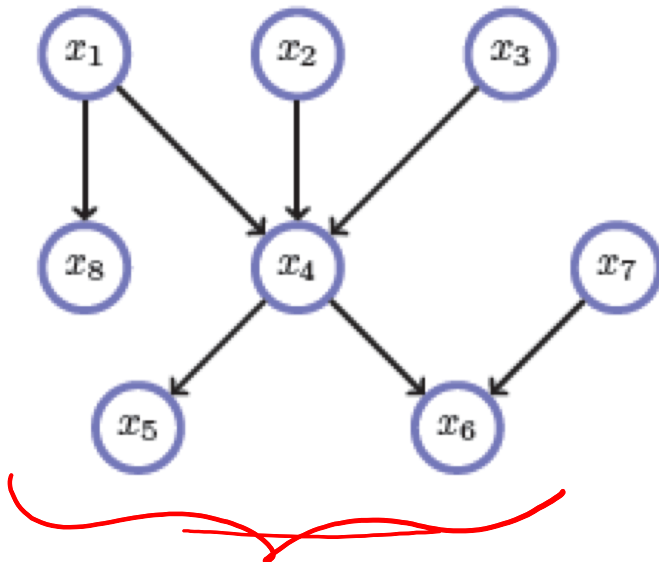
Computational Graph

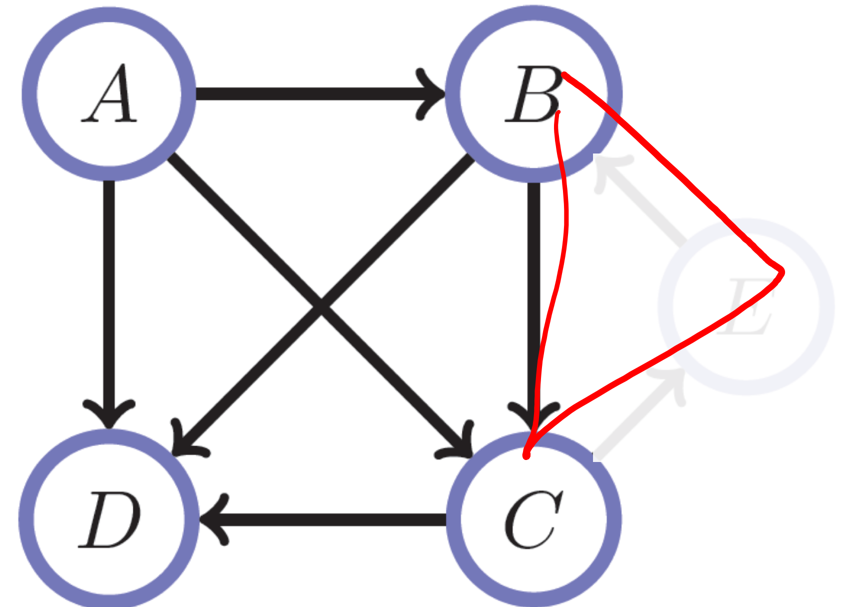


Any DAG of differentiable modules is allowed!

Directed Acyclic Graphs (DAGs)

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

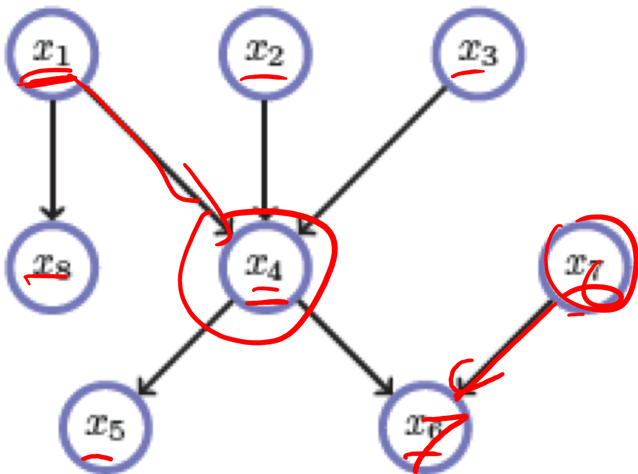


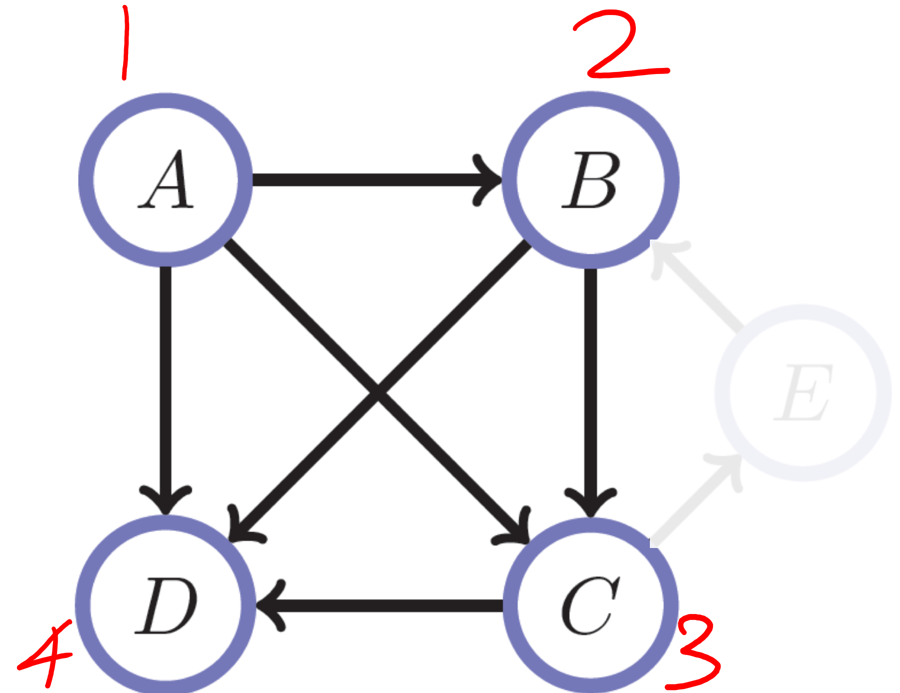


Directed Acyclic Graphs (DAGs)

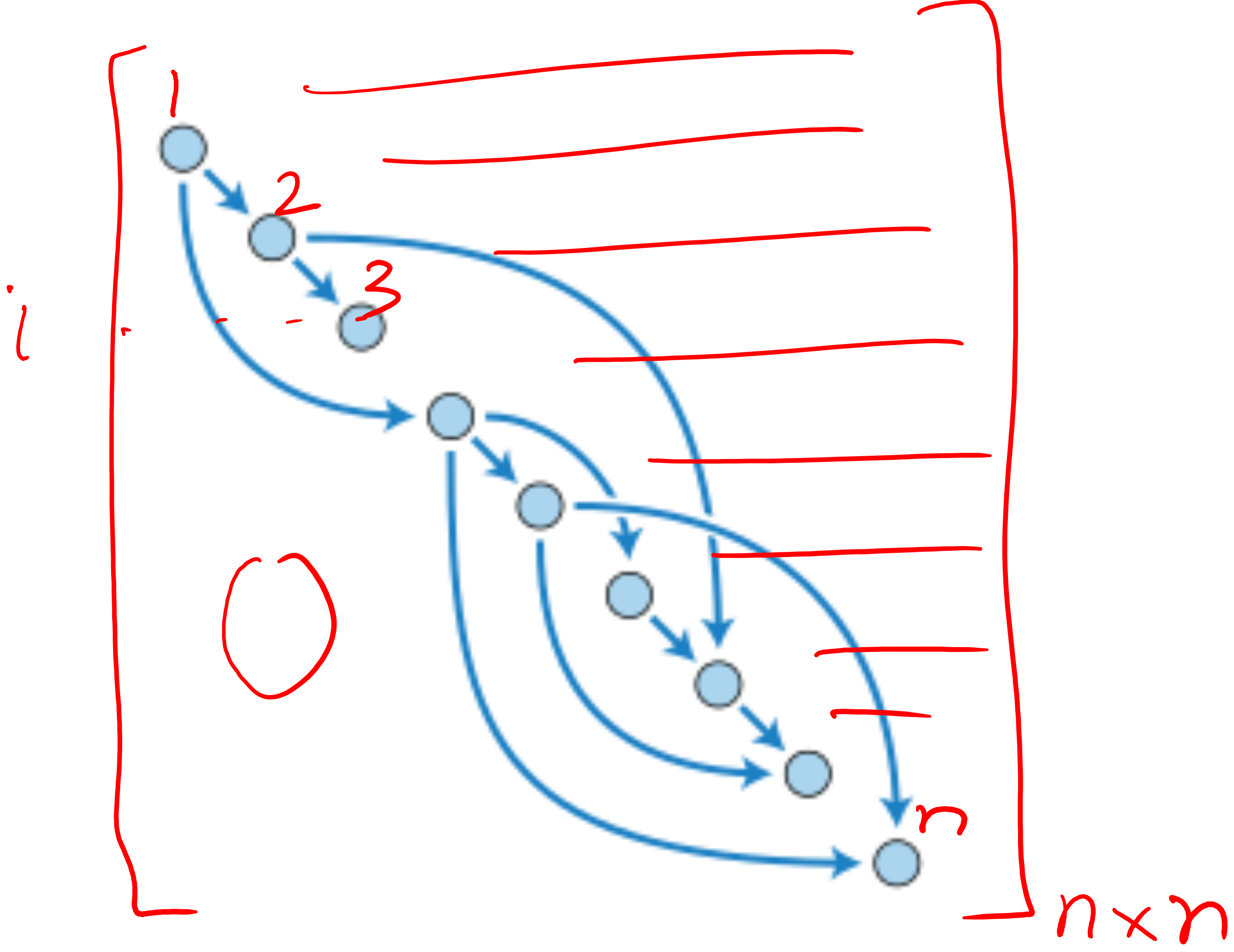
- Concept
  - Topological Ordering

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





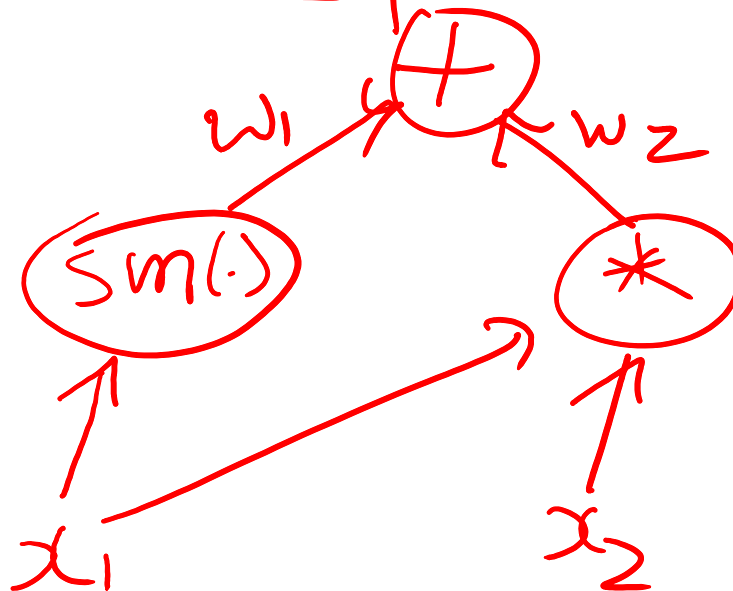
Directed Acyclic Graphs (DAGs)



Computational Graphs

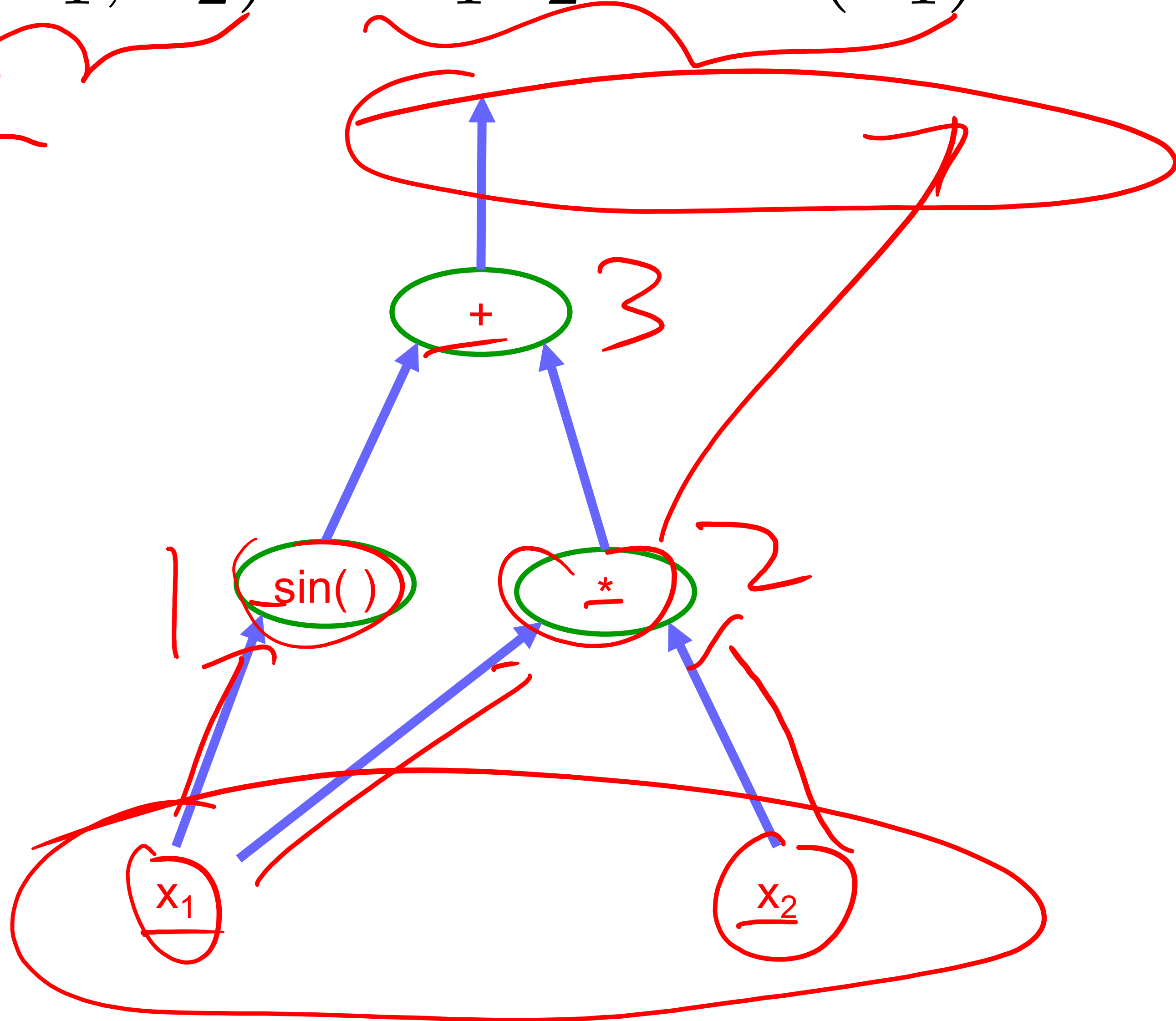
- Notation

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



Example

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

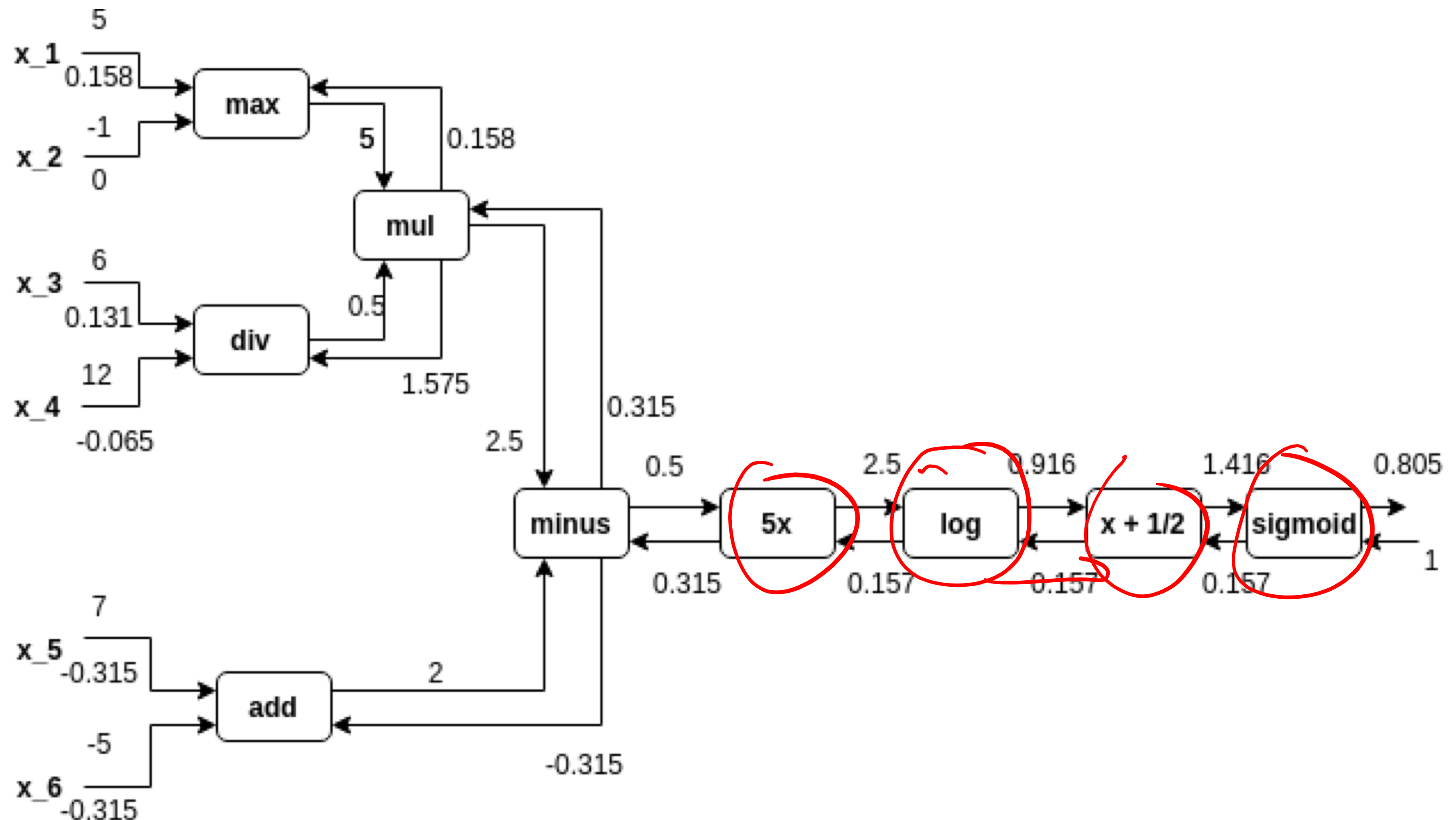


HW0

$$f(\mathbf{x}) = \sigma\left(\log\left(5\left(\max\{x_1, x_2\} \cdot \frac{x_3}{x_4} - (x_5 + x_6)\right)\right) + \frac{1}{2}\right)$$

HW0 Submission by Samyak Datta

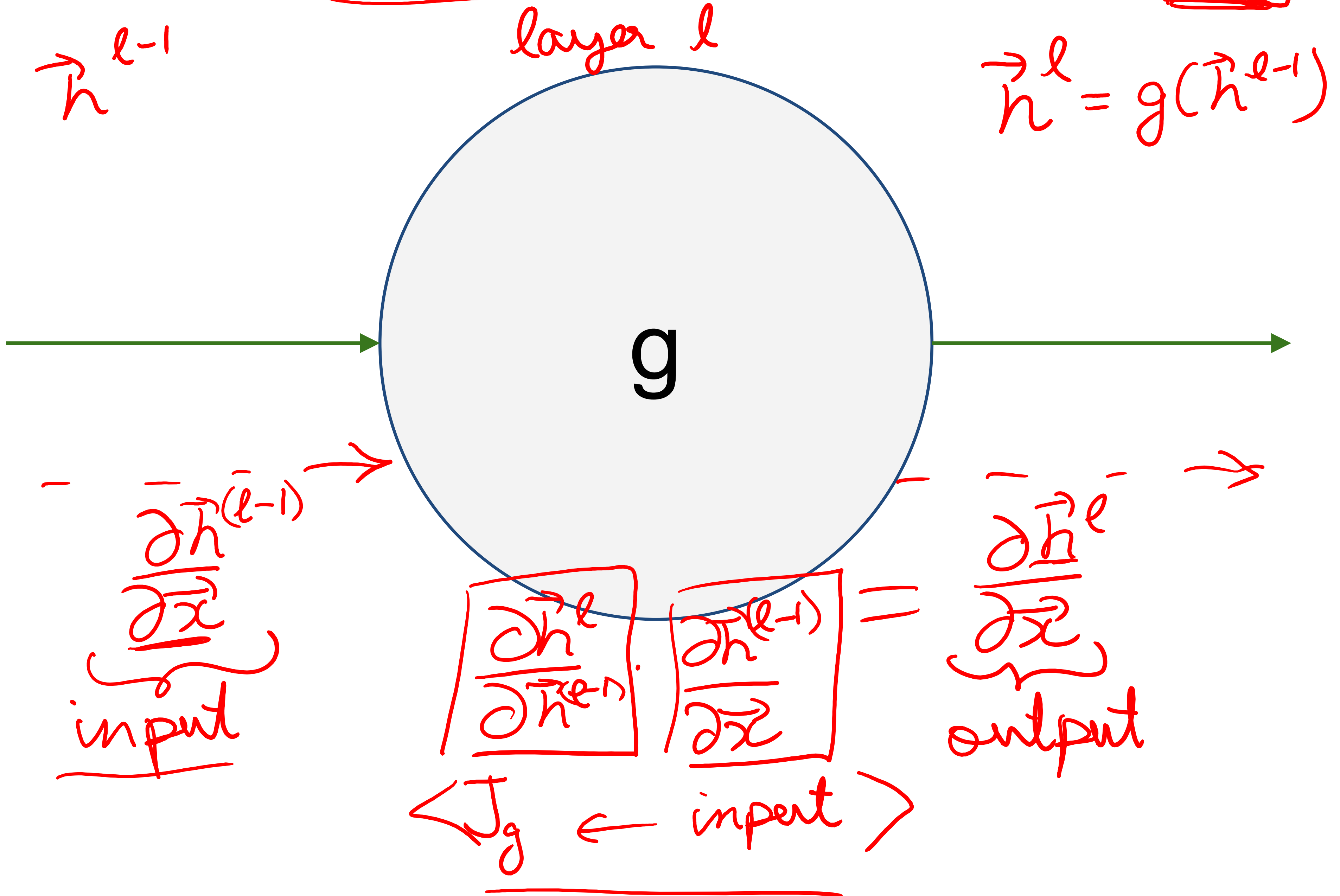
$$f(\mathbf{x}) = \sigma\left(\log\left(5\left(\max\{x_1, x_2\} \cdot \frac{x_3}{x_4} - (x_5 + x_6)\right)\right) + \frac{1}{2}\right)$$



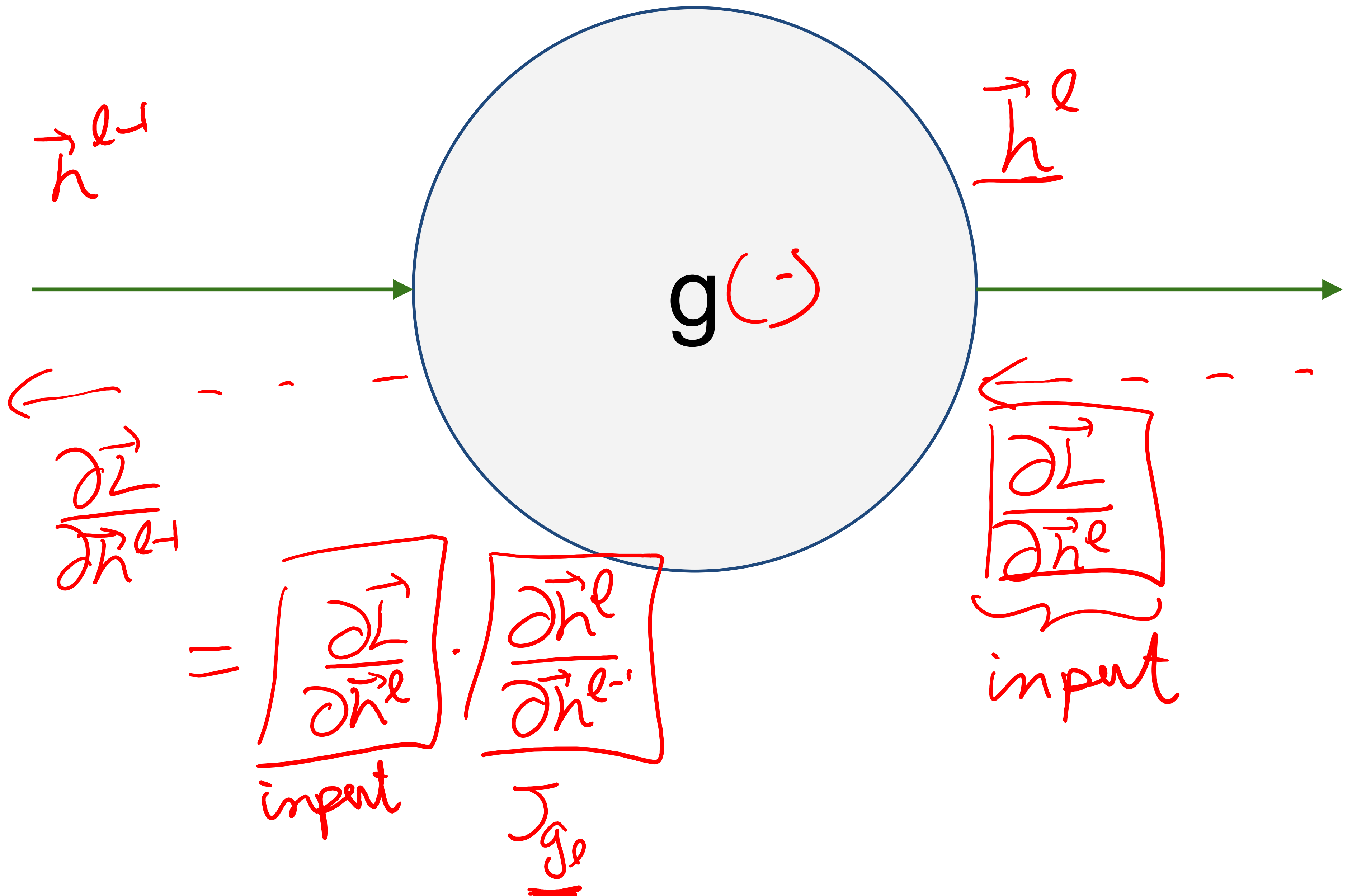
Forward mode vs Reverse Mode

- Key Computations

Forward mode AD

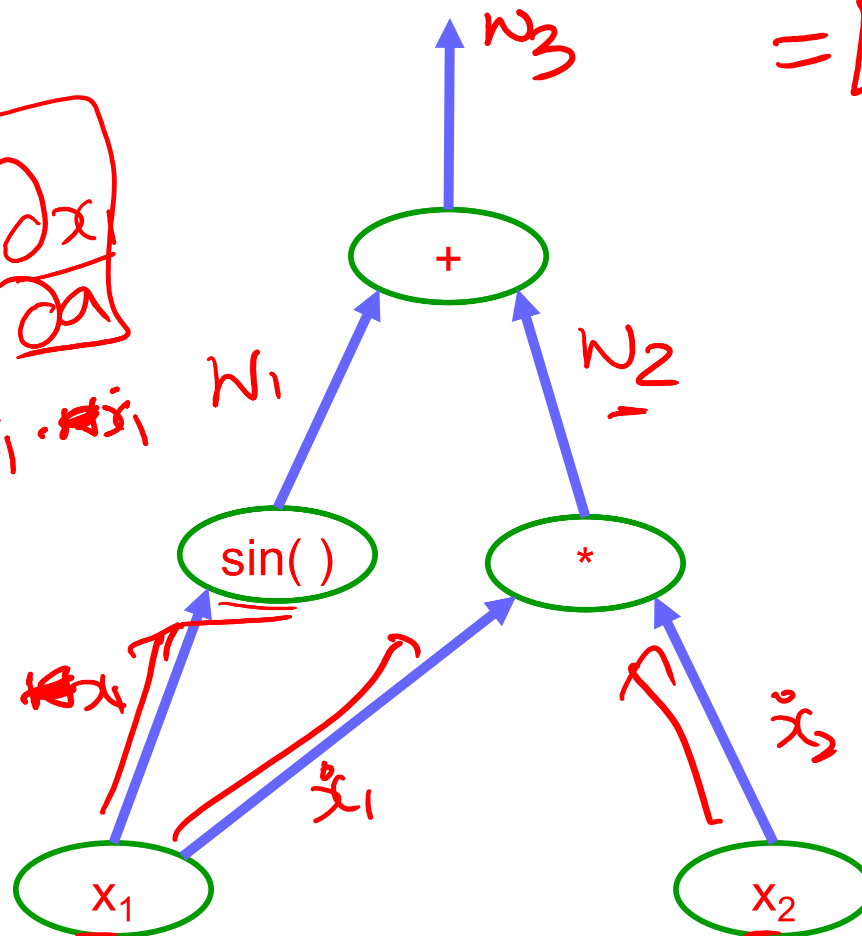


Reverse mode AD



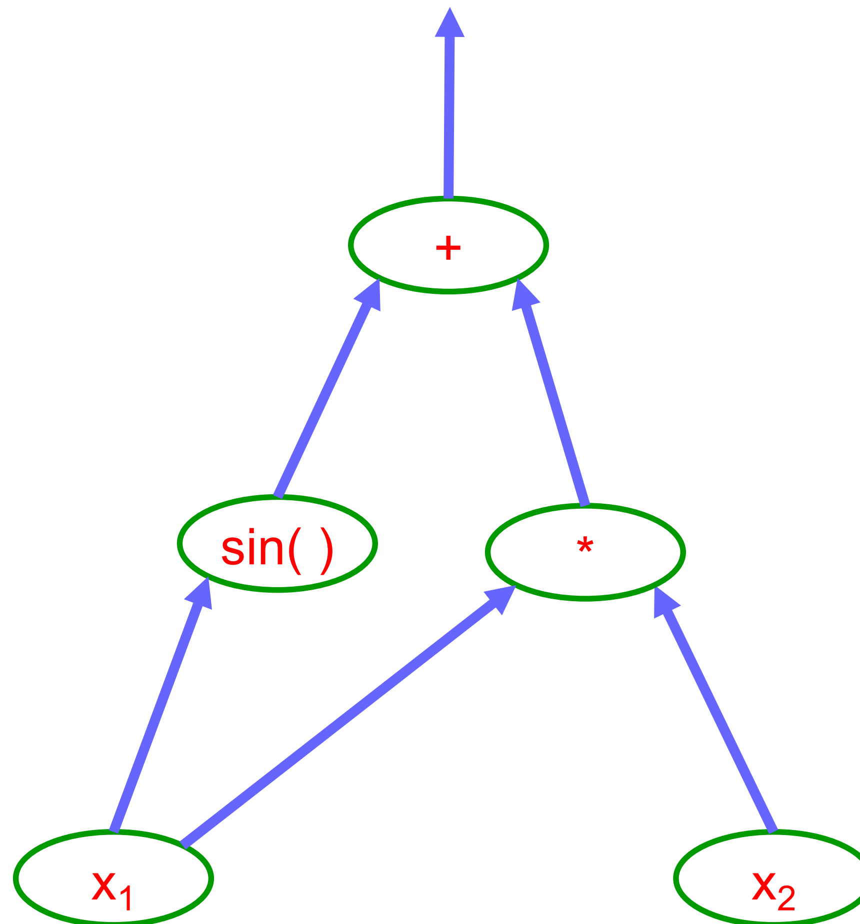
Example: Forward mode AD

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



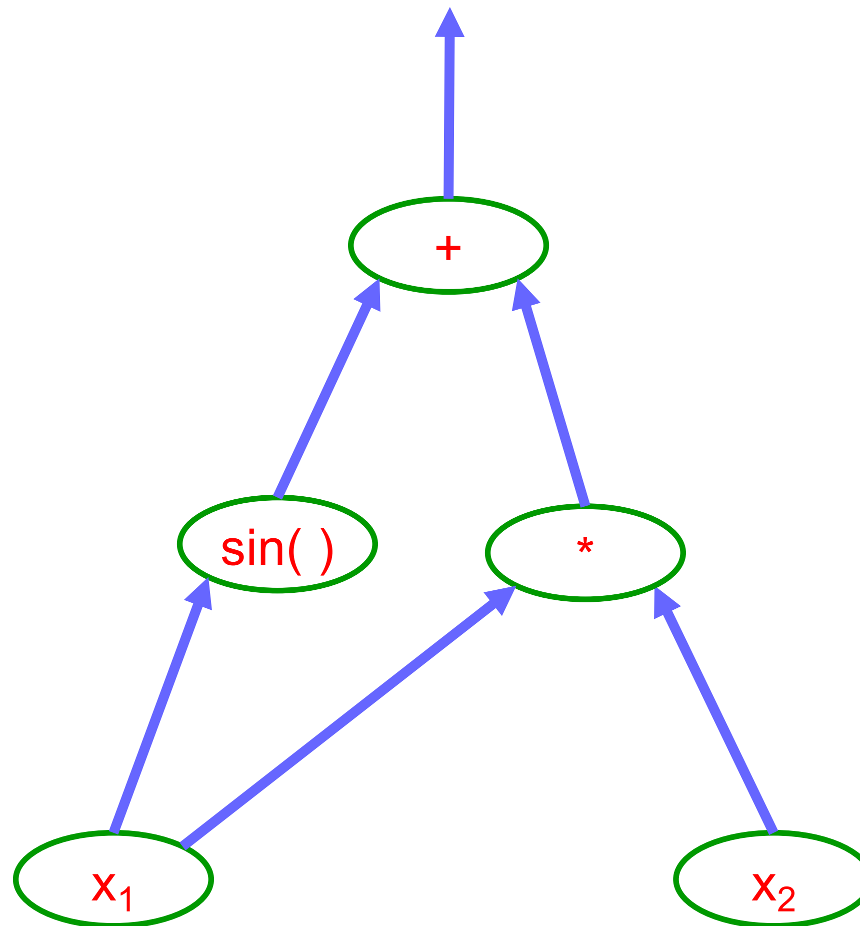
Example: Forward mode AD

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



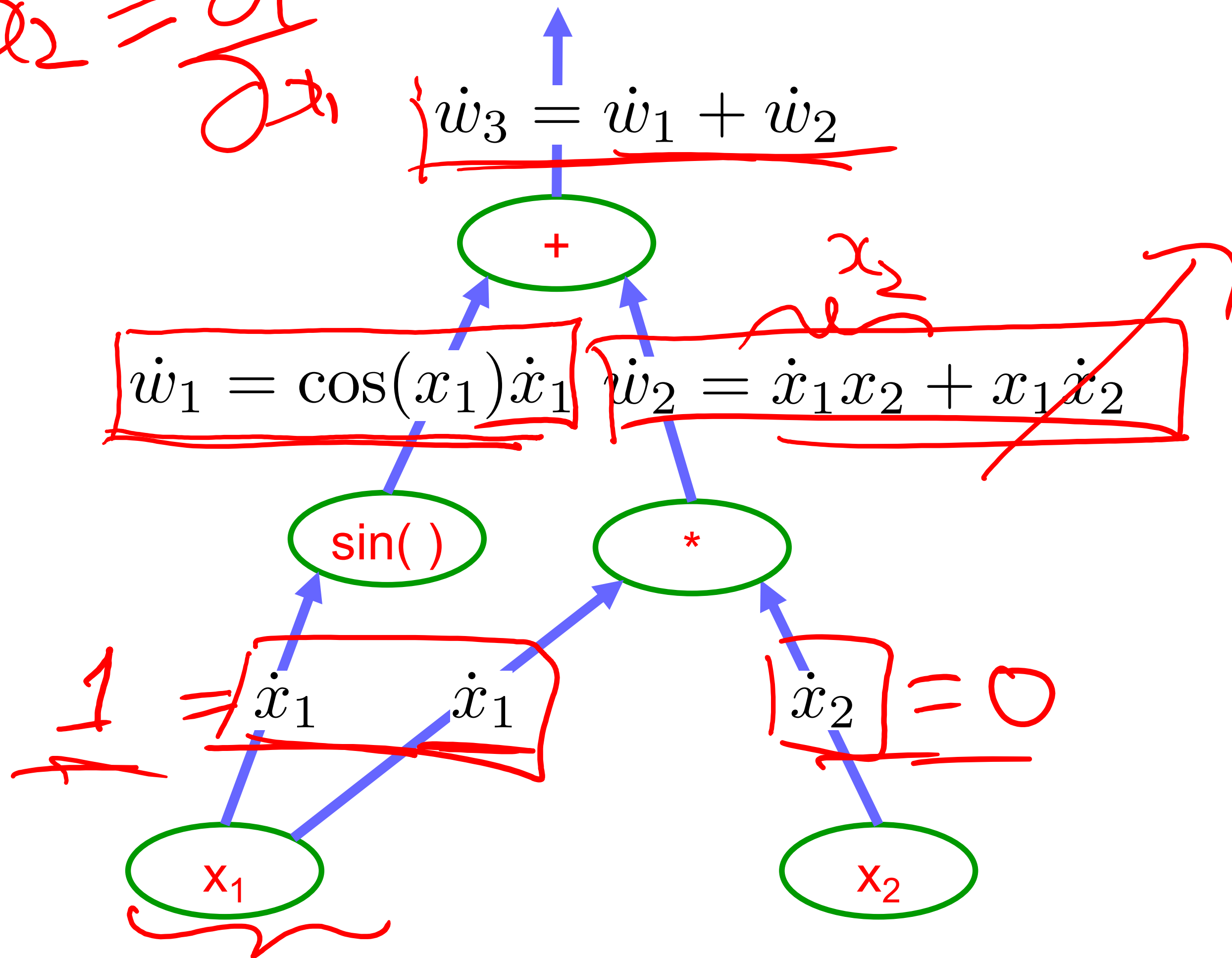
Example: Forward mode AD

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



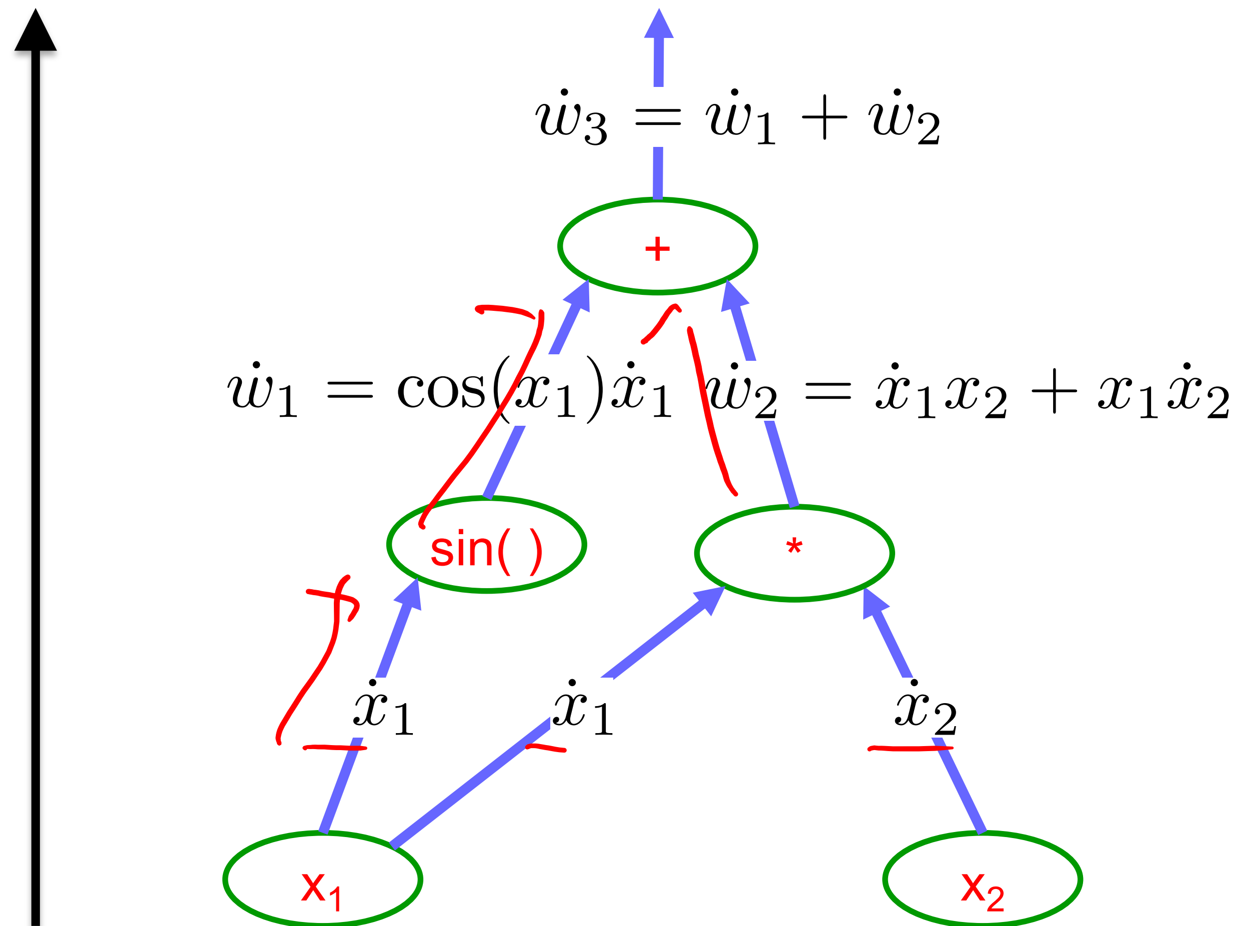
Example: Forward mode AD

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



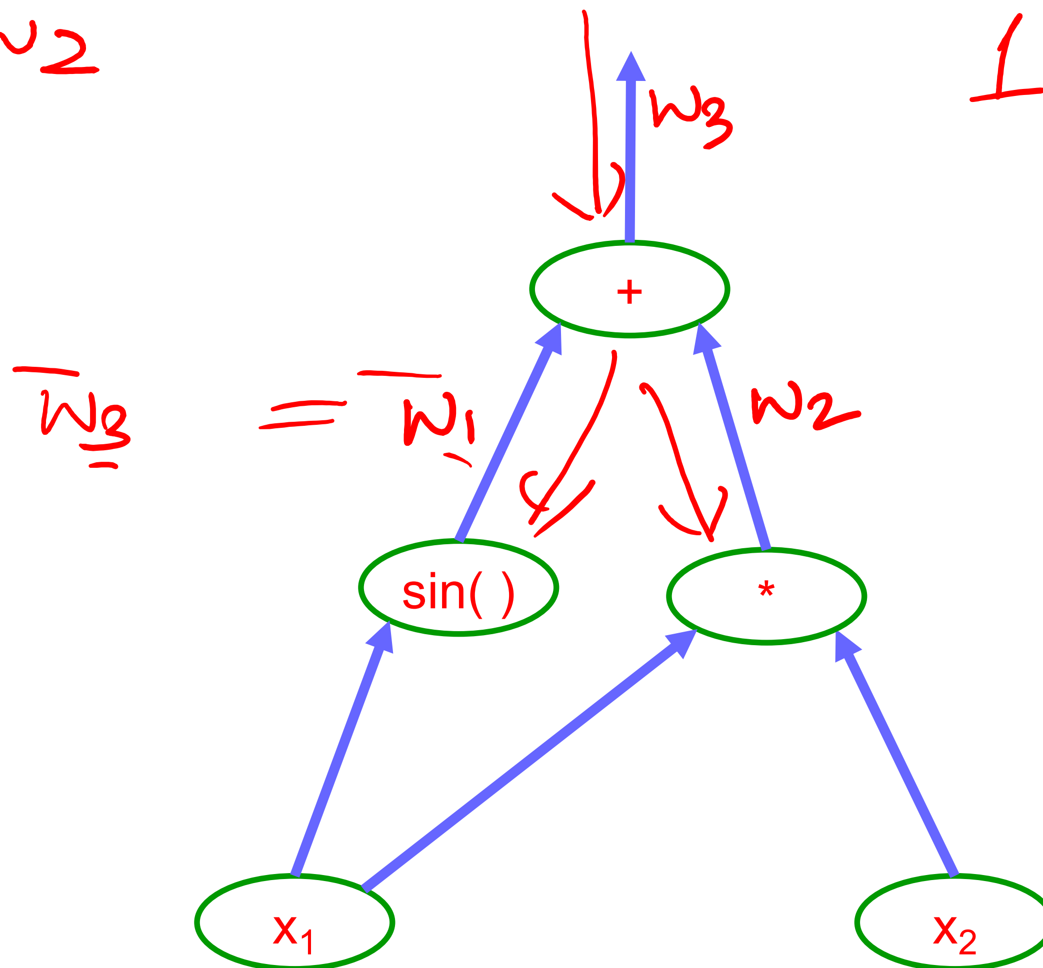
Example: Forward mode AD

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



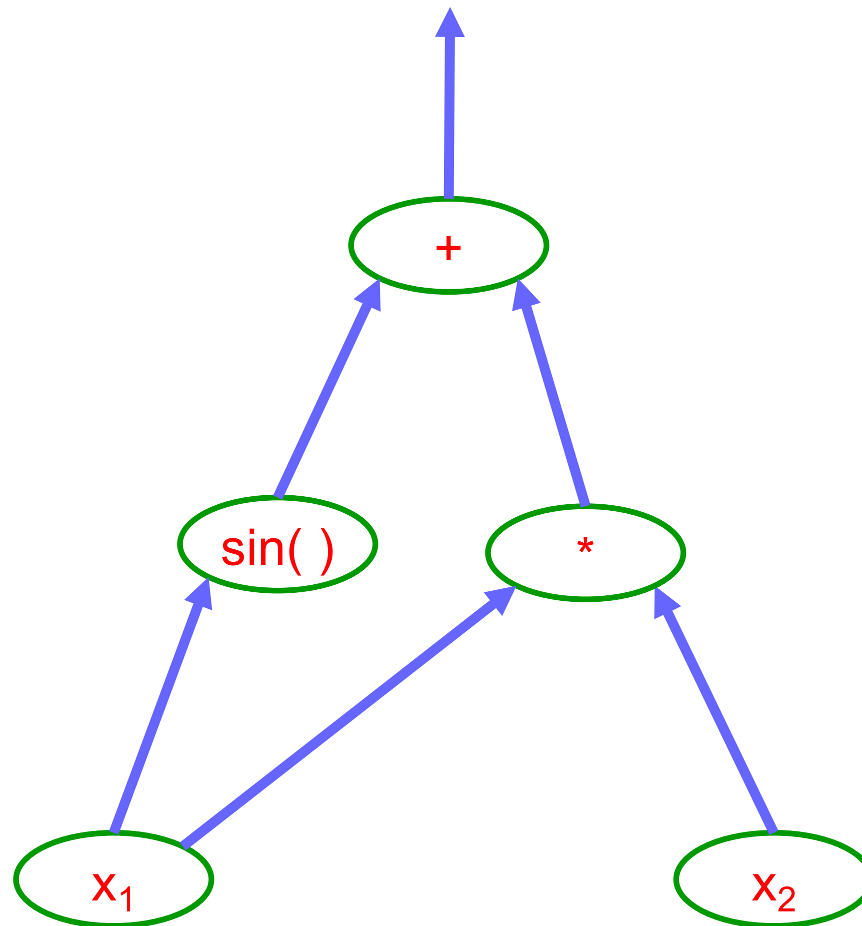
Example: Reverse mode AD

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



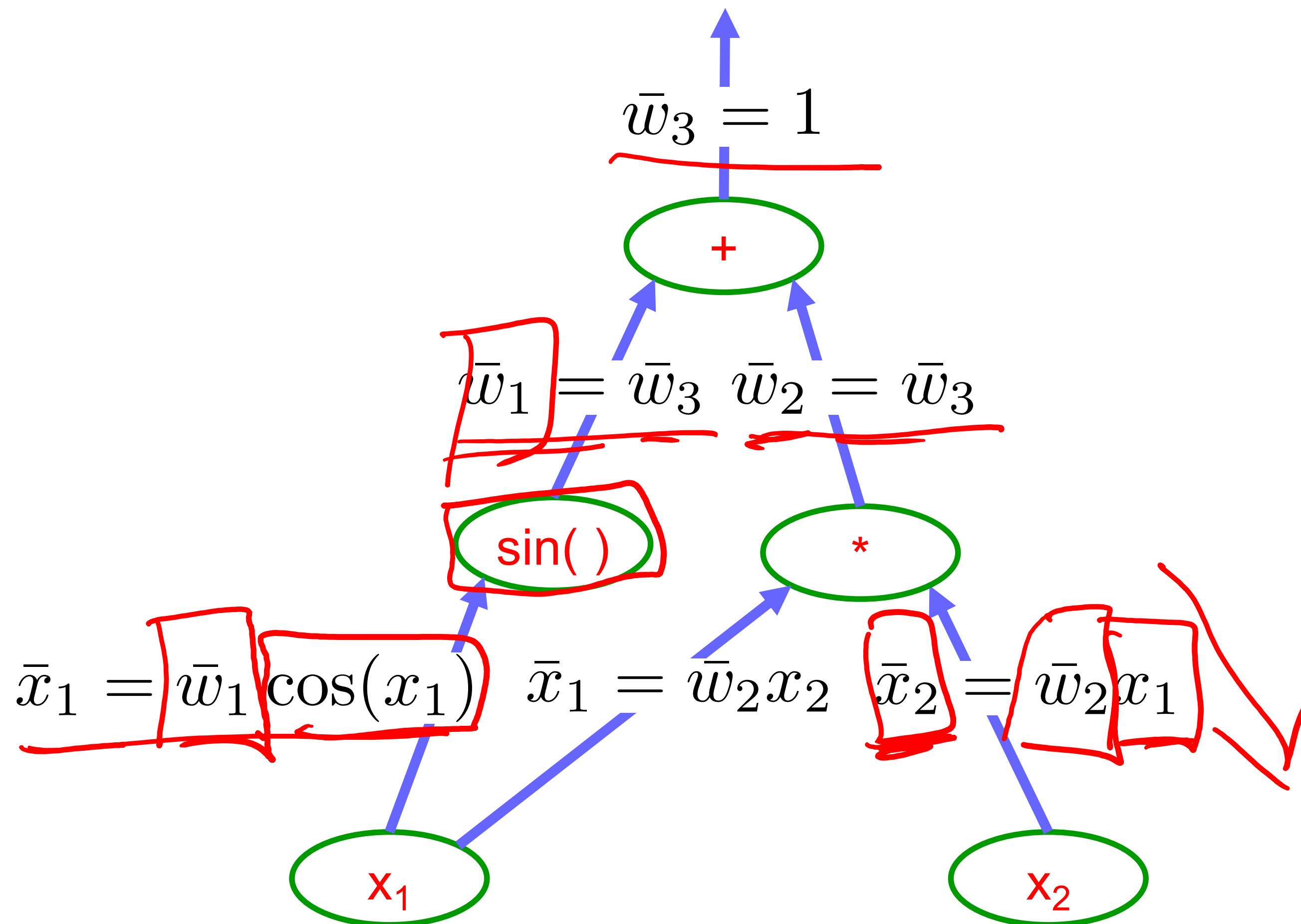
Example: Reverse mode AD

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



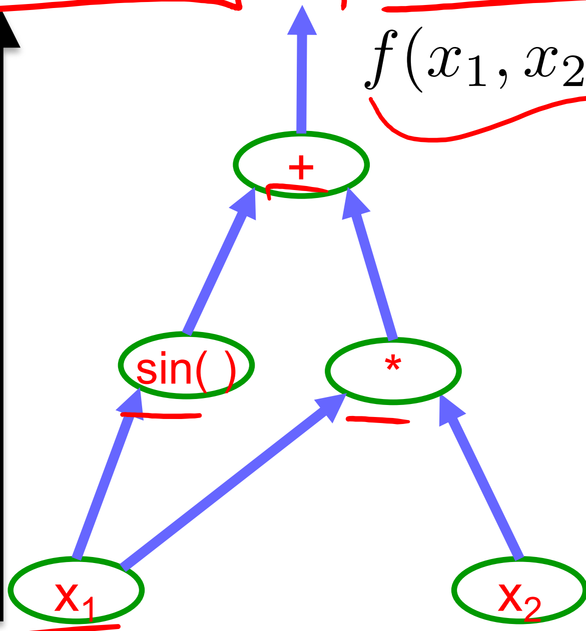
Example: Reverse mode AD

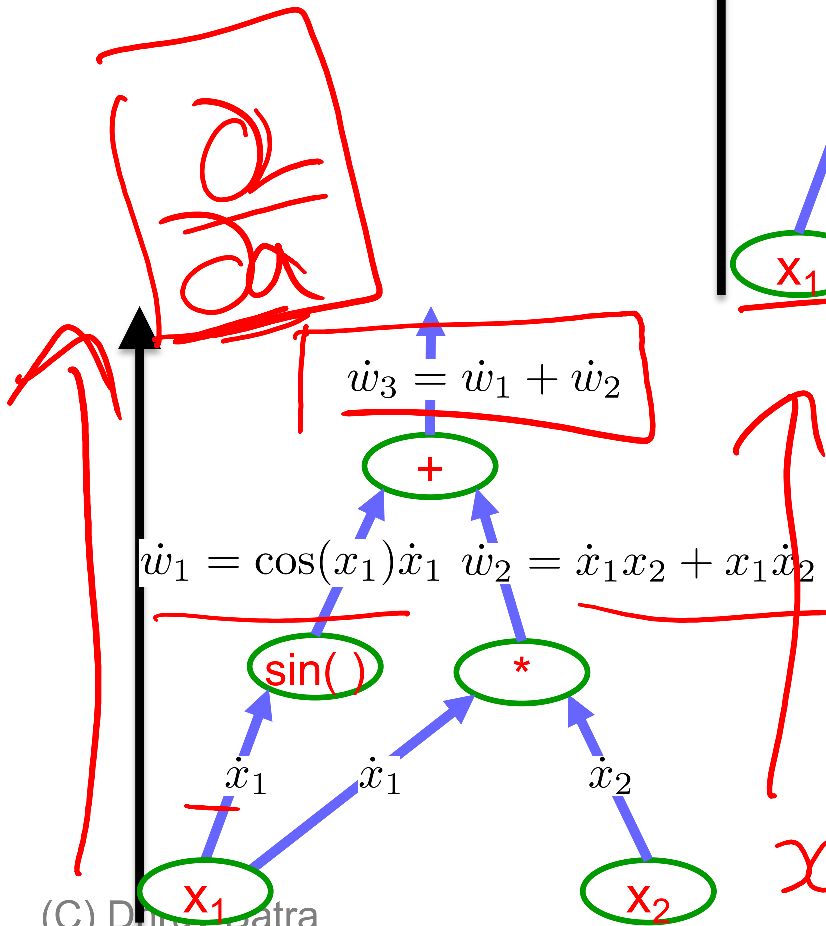
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

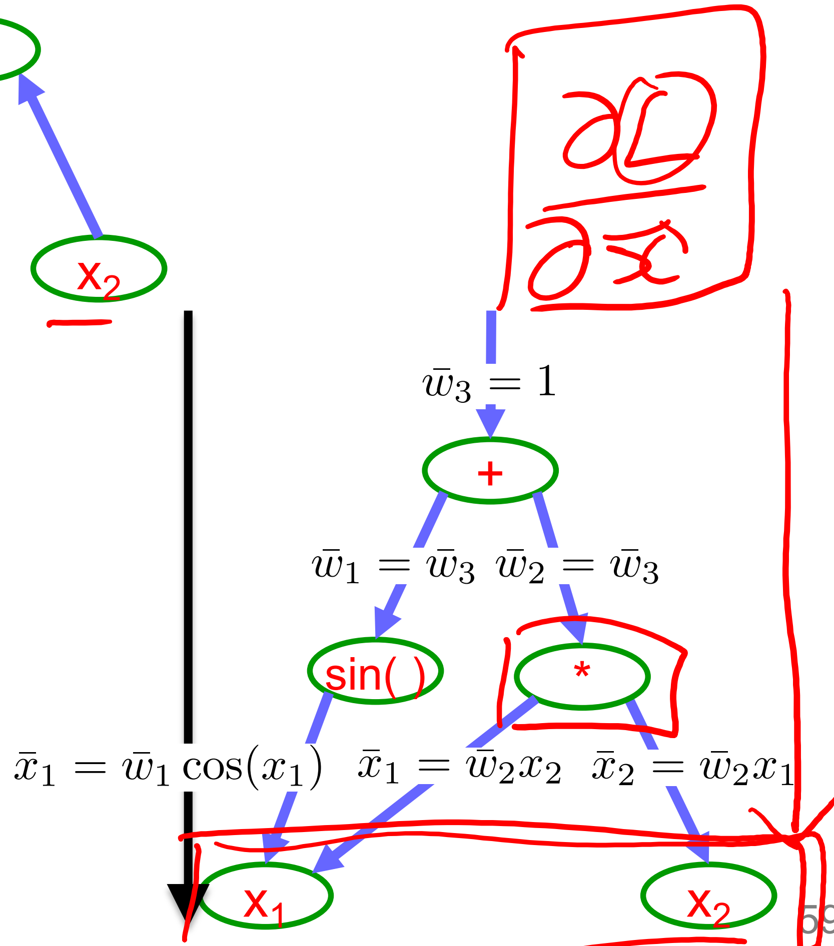


Forward Pass vs Forward mode AD vs Reverse Mode AD

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

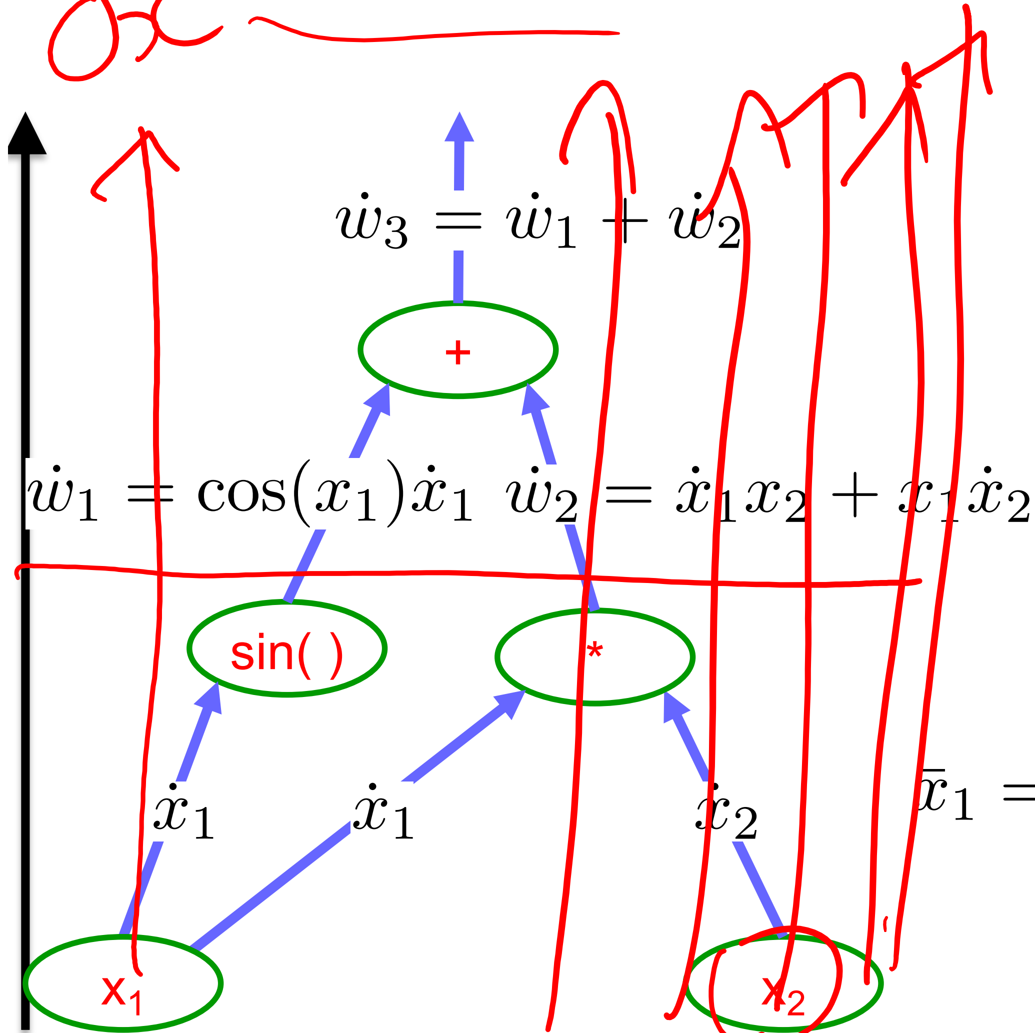


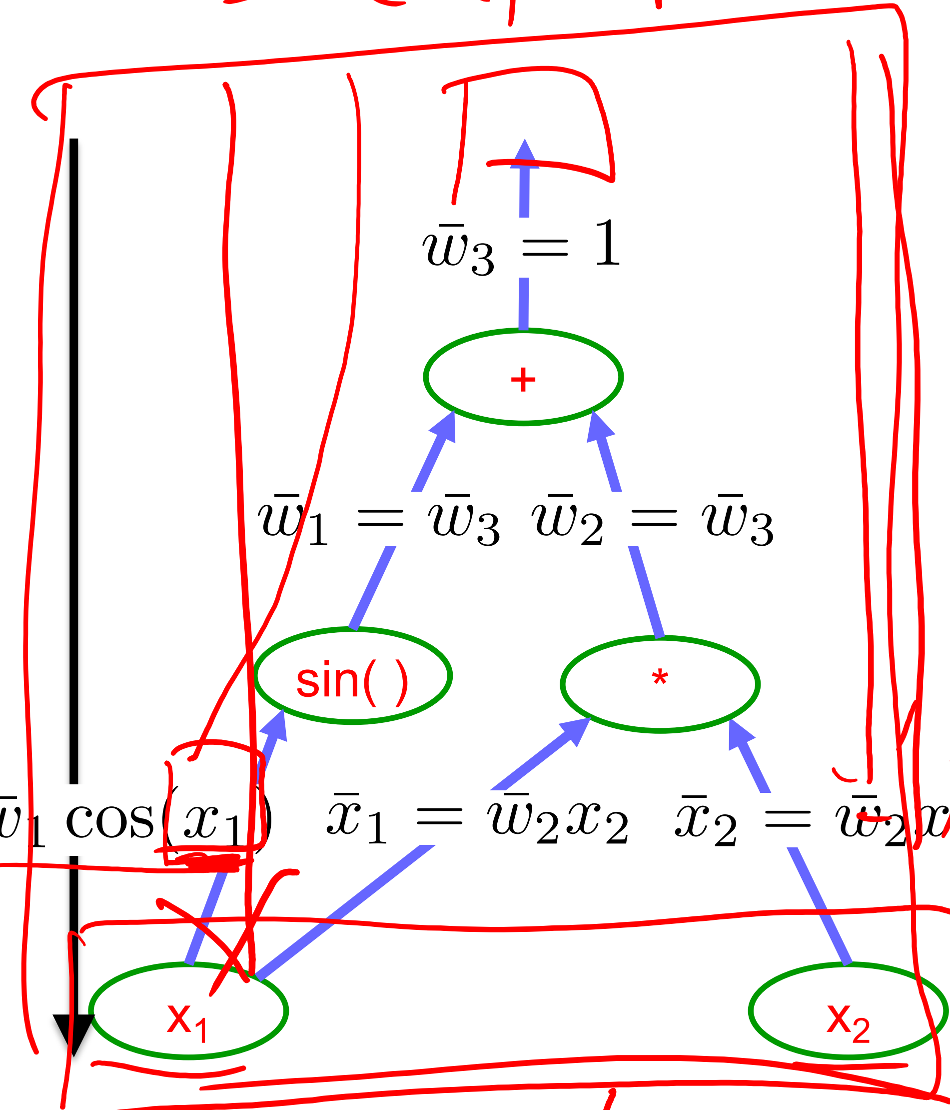




Forward mode vs Reverse Mode

- What are the differences?





Forward mode vs Reverse Mode

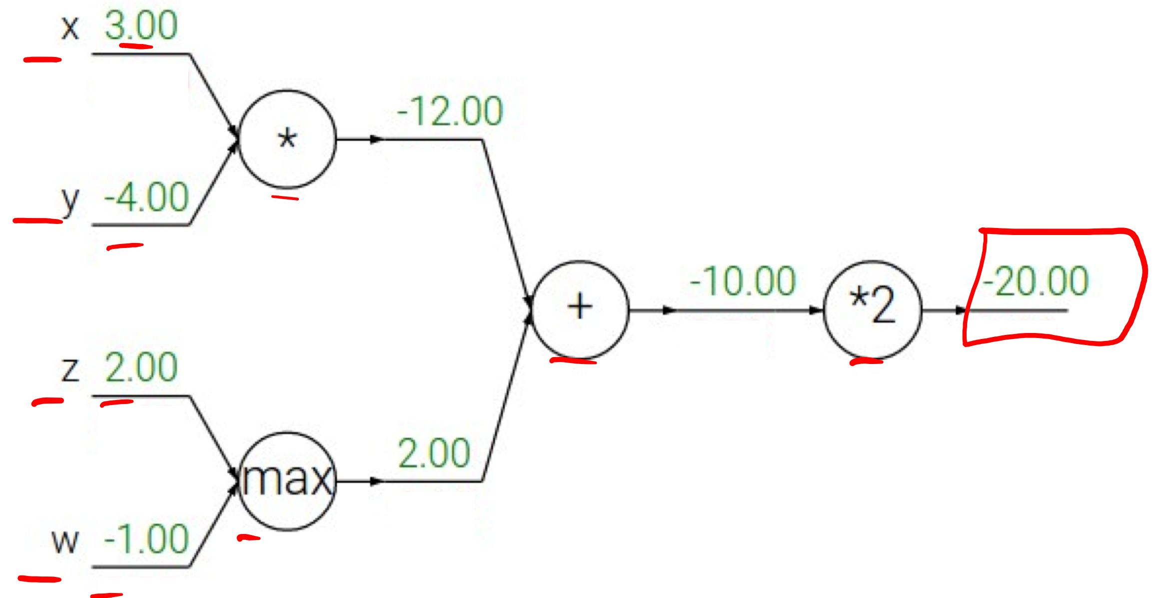
- What are the differences?
- Which one is faster to compute?
  - Forward or backward?

Forward mode vs Reverse Mode

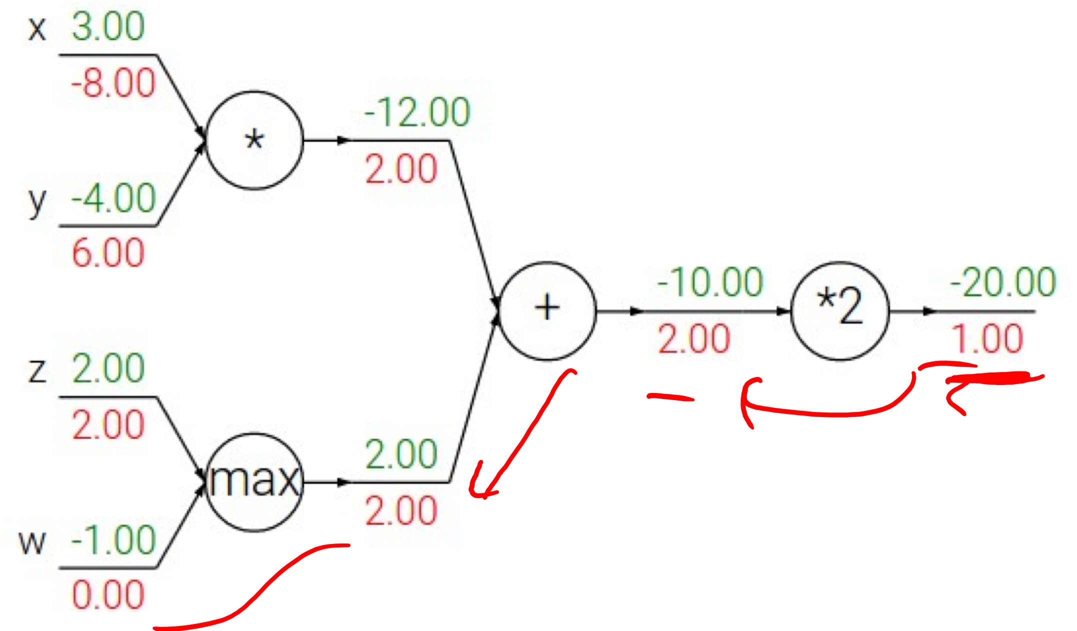
- What are the differences?
- Which one is faster to compute?
  - Forward or backward?
- Which one is more memory efficient (less storage)?
  - Forward or backward?

Patterns in backward flow

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

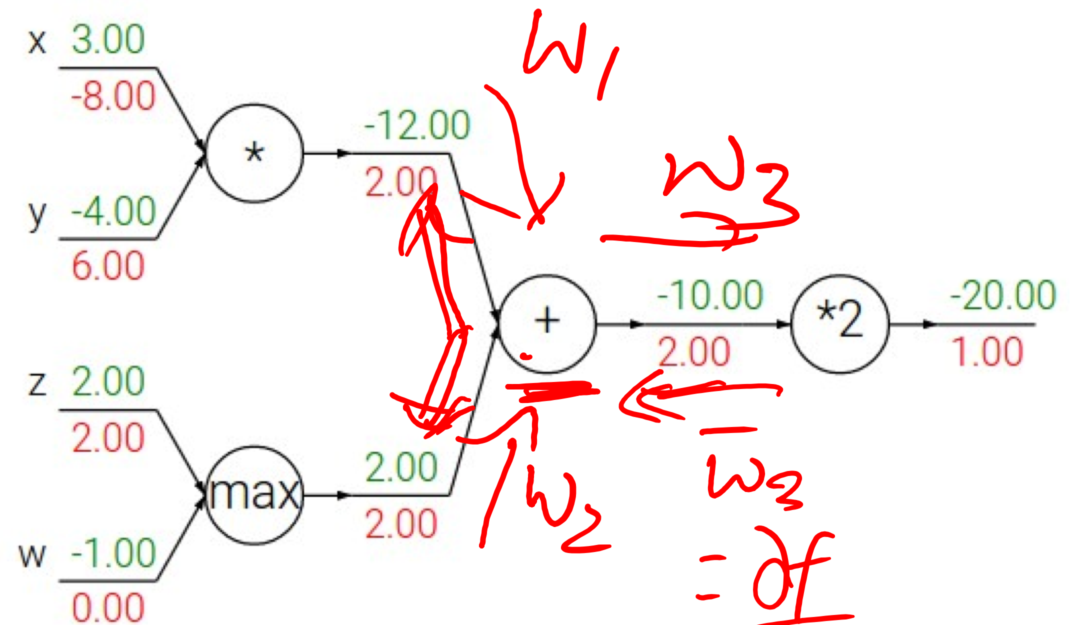


Patterns in backward flow



Patterns in backward flow

add gate: gradient distributor



$w_3 = w_1 + w_2$

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

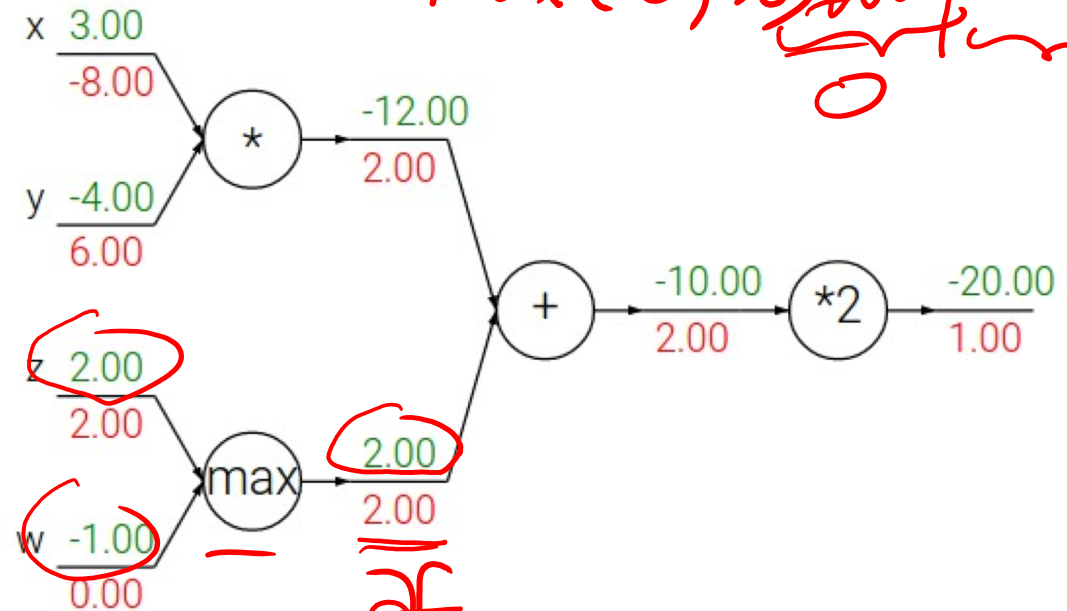
$\bar{w}_3 = \frac{\partial f}{\partial w_3}$

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

Patterns in backward flow

add gate: gradient distributor

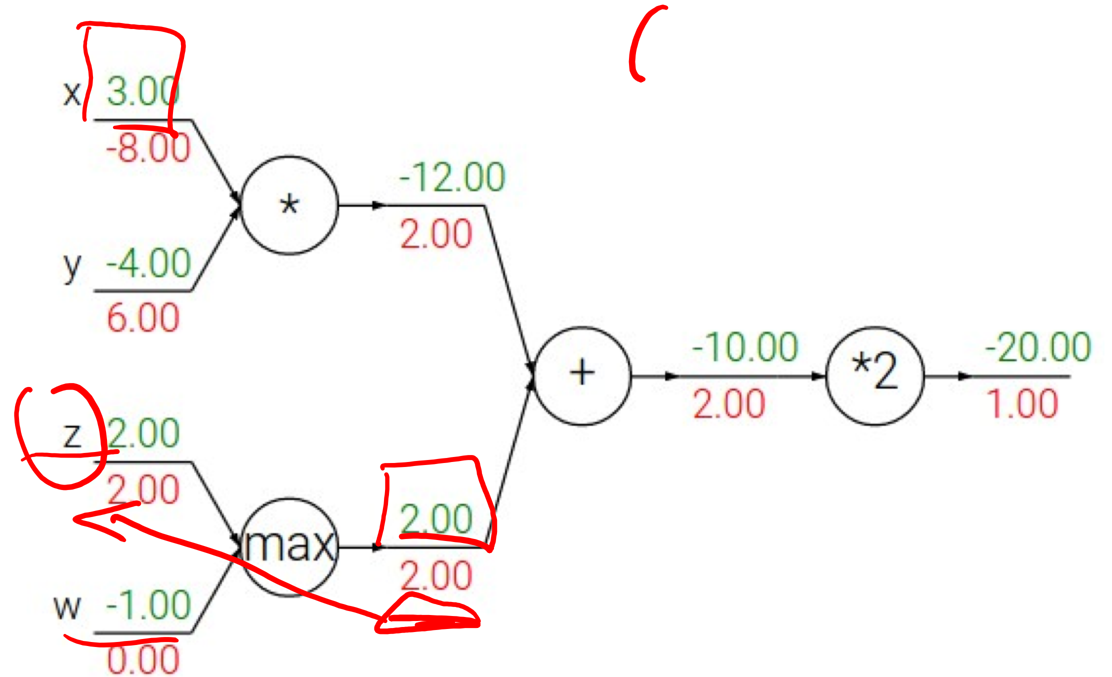
Q: What is a **max** gate?



Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

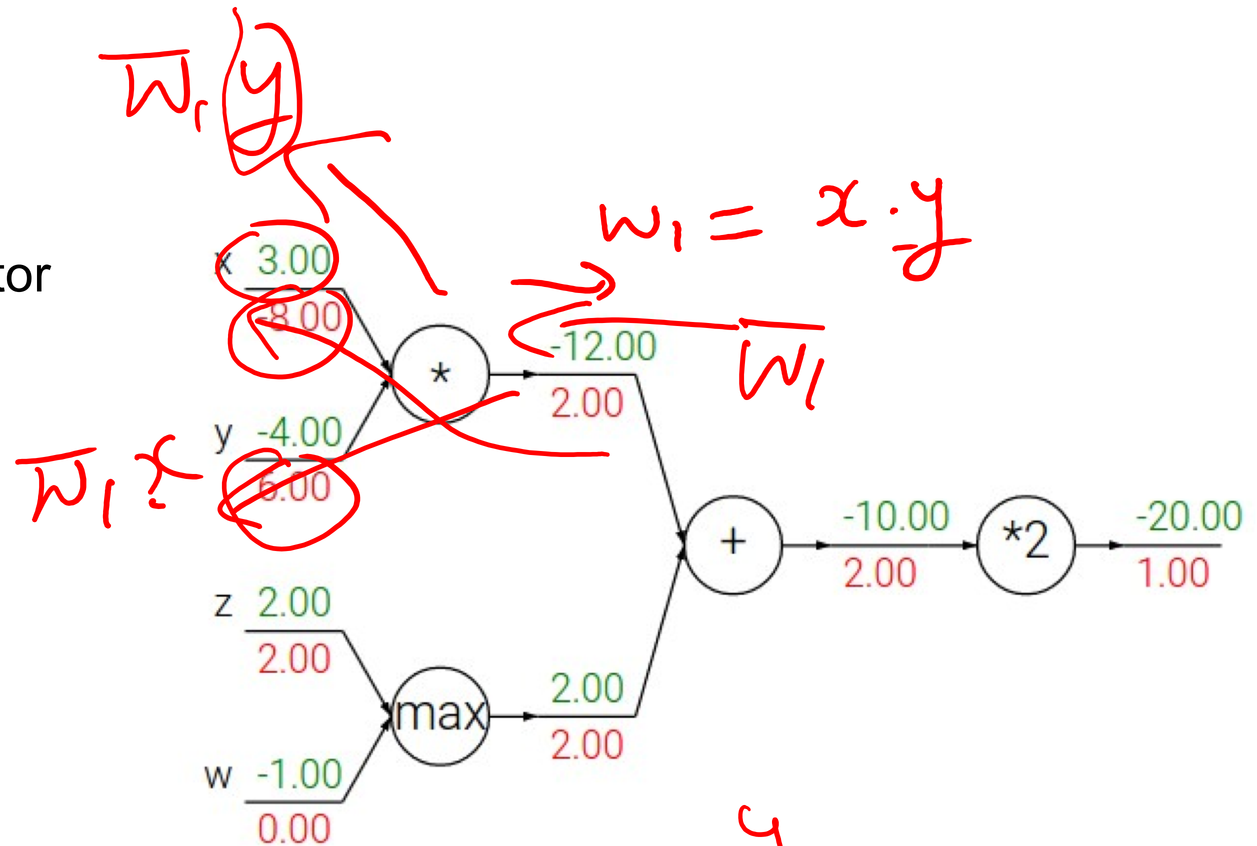


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

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

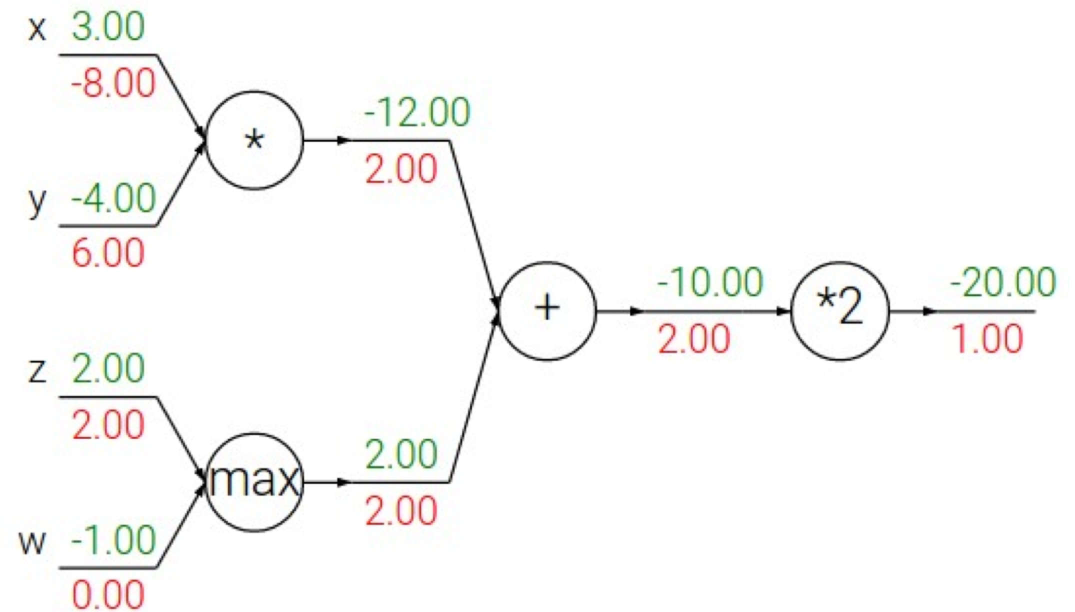


Patterns in backward flow

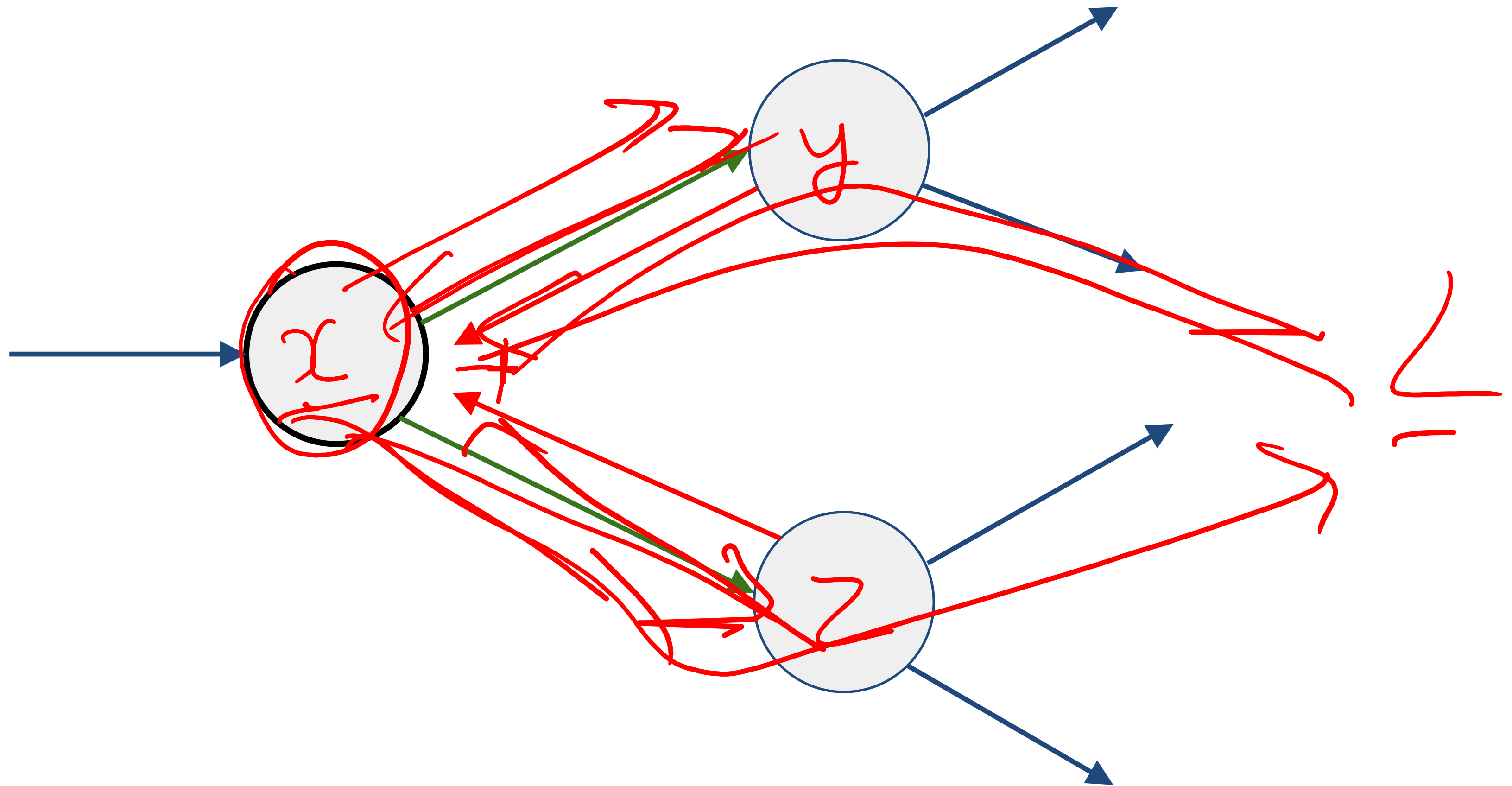
add gate: gradient distributor

max gate: gradient router

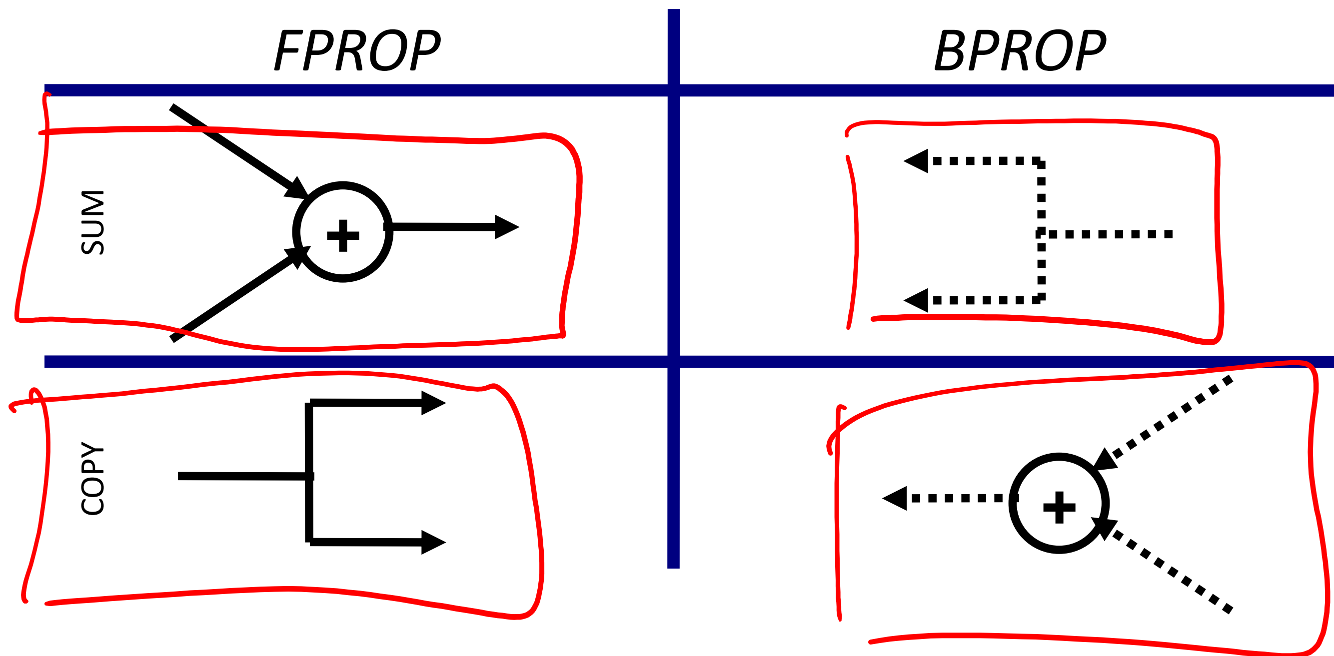
mul gate: gradient switcher



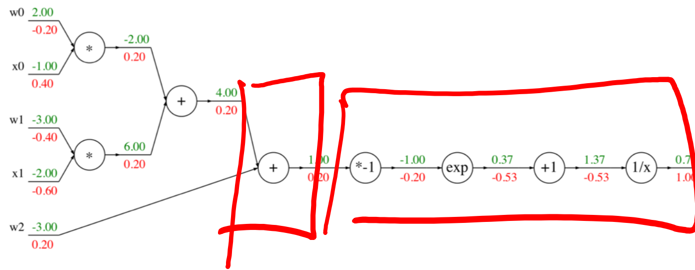
Gradients add at branches



Duality in Fprop and Bprop



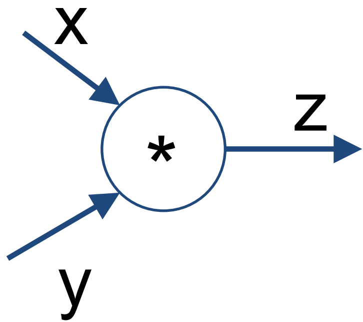
Modularized implementation: forward / backward API



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
```

Modularized implementation: forward / backward API



(x,y,z are scalars)

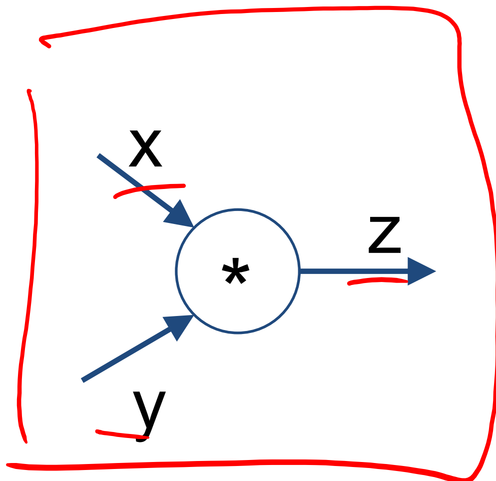
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$\frac{\partial L}{\partial z}$ (points to `backward(dz)`)

$\frac{\partial L}{\partial x}$ (points to `dx` in `return [dx, dy]`)

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]
```

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

..

File	Commit message	Age
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

Caffe is licensed under BSD 2-Clause

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)

Caffe is licensed under BSD 2-Clause

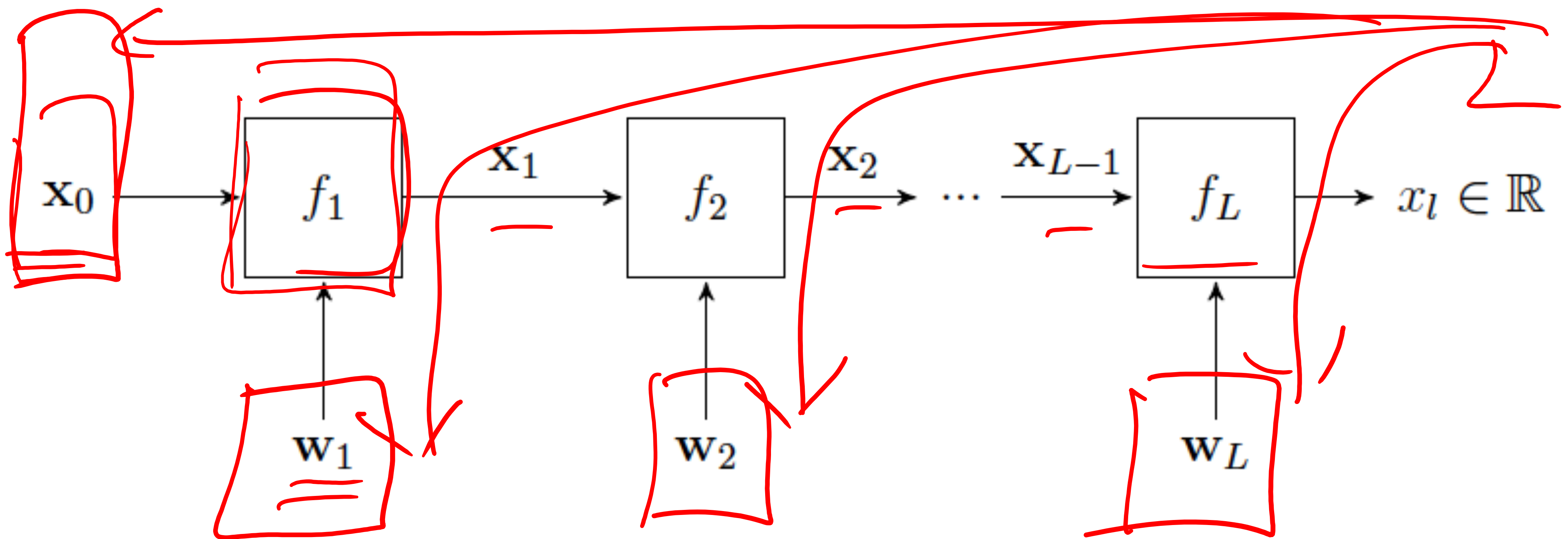
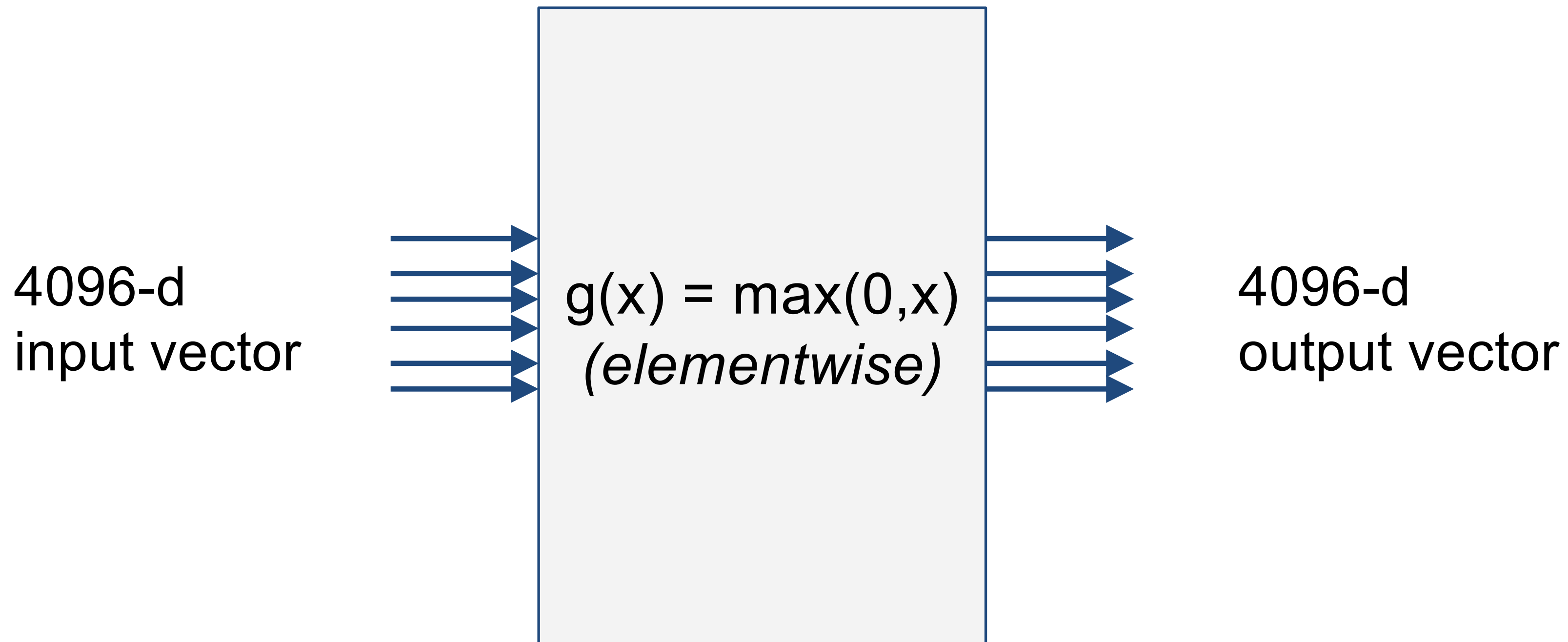


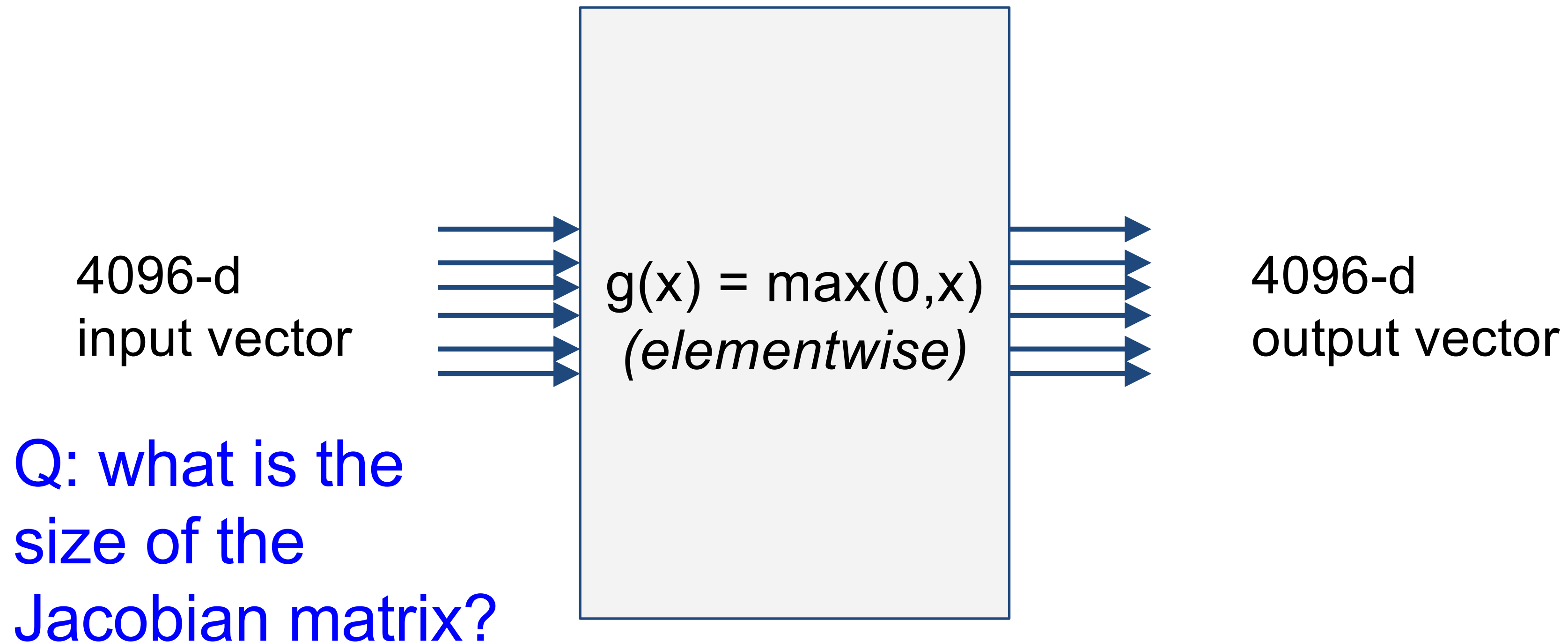
Figure Credit: Andrea Vedaldi

Jacobian of ReLU

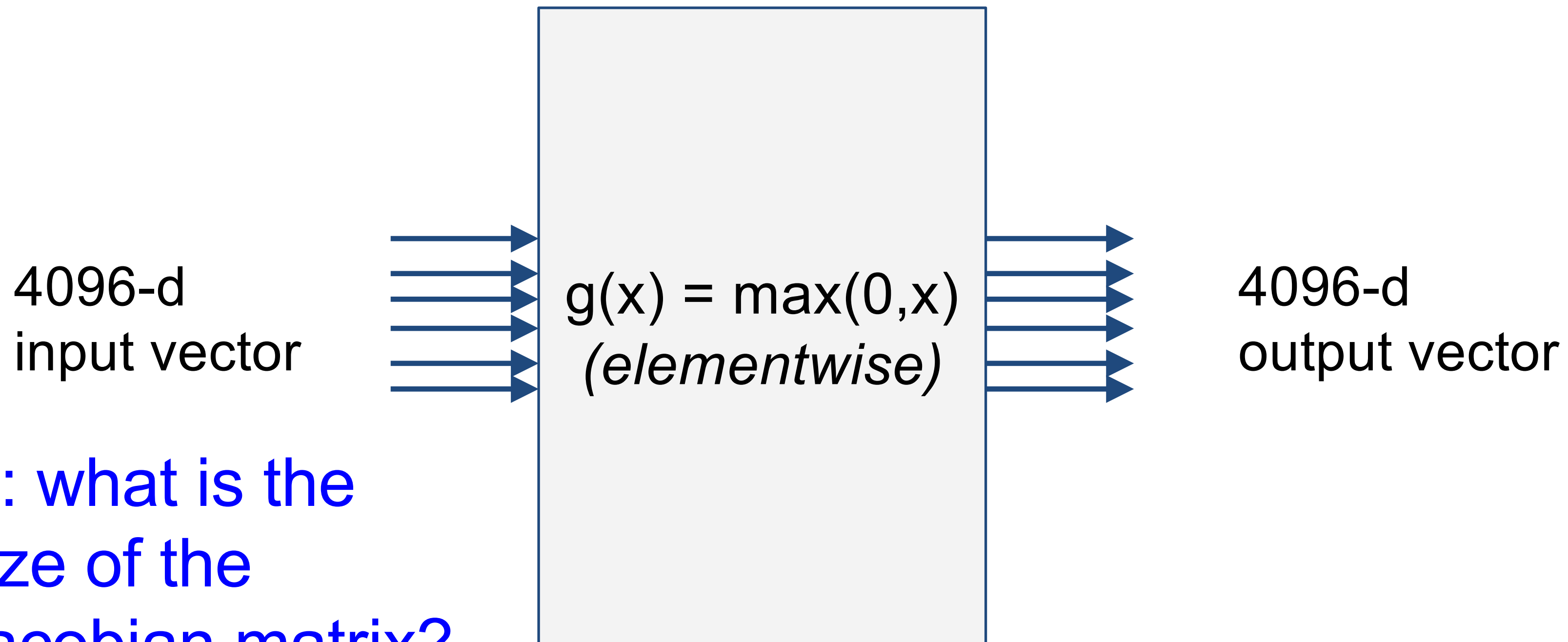


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Jacobian of ReLU



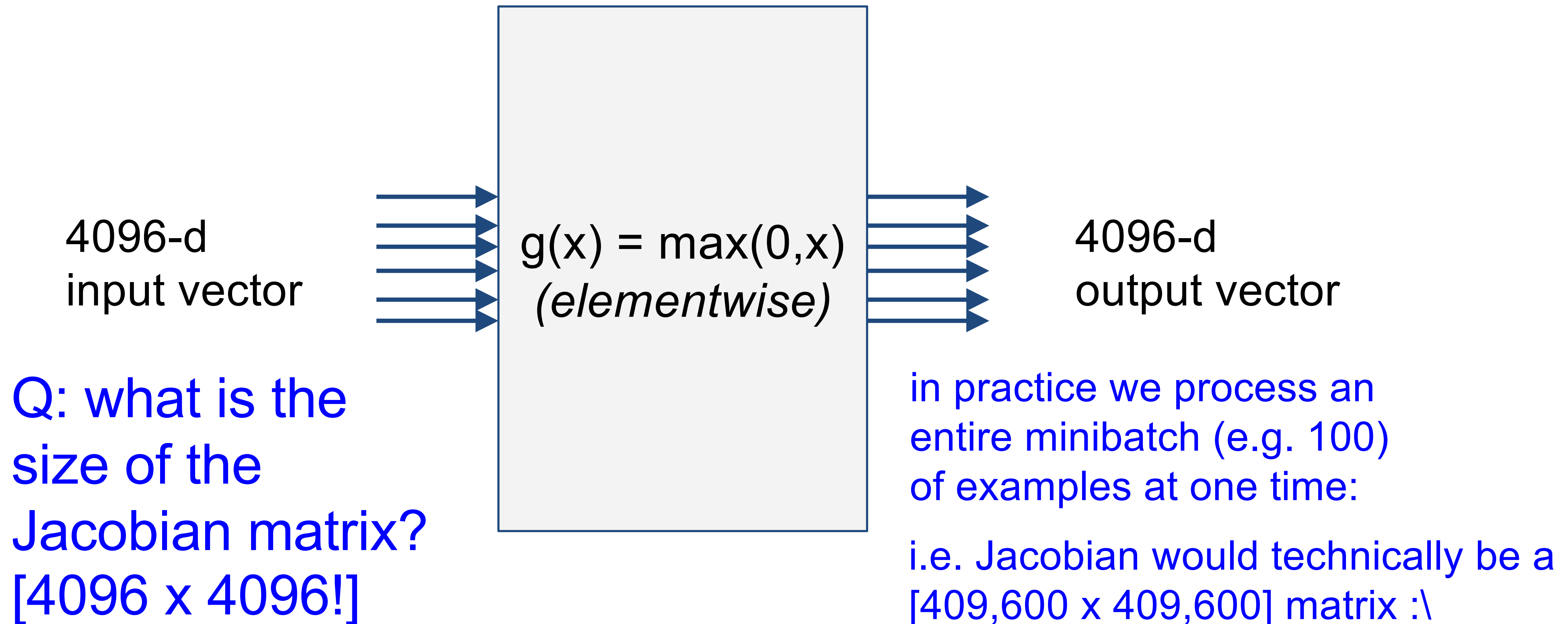
Jacobian of ReLU



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

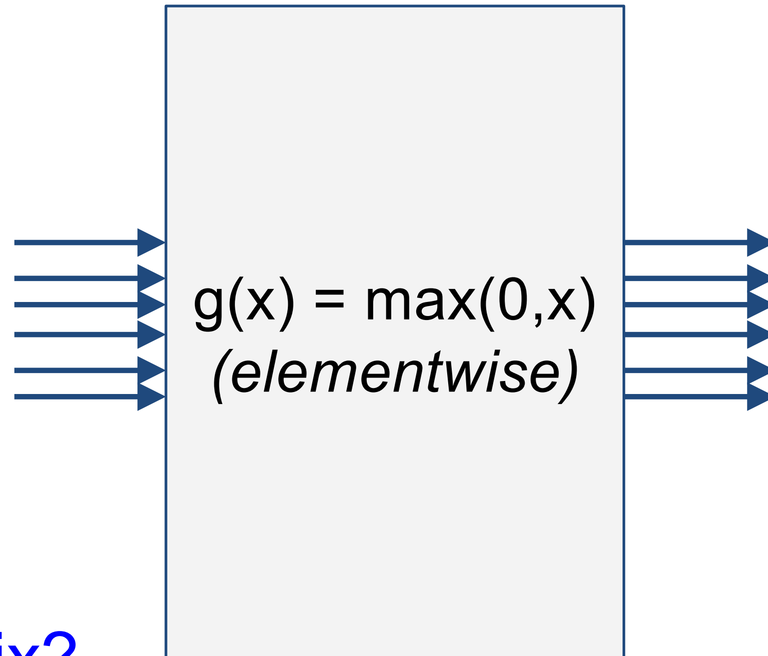
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Jacobian of ReLU



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

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?

Jacobians of FC-Layer

Jacobians of FC-Layer

Jacobians of FC-Layer