# CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Dhruv Batra

Georgia Tech

# Administrativia

- HW2 out
  - Due: 09/23 11:59pm
  - Theory: Gradient descent, Hessians, Auto-diff, Convolutions
    - https://www.overleaf.com/project/5f4c143e06061d00013dd4f0
  - Implementation: ConvNets in Python and PyTorch
  - Bonus: Challenge on EvalAI
    - https://evalai.cloudcv.org/web/challenges/challenge-page/684/overview

# Project

- Goal
  - Chance to take on something open-ended
  - Encouraged to apply to your research (computer vision, NLP, robotics,…)

- Main categories
  - Reproducibility
    - Pick a paper from a recent conference. Attempt to reproduce the method and validate claims.
  - Application/Survey
    - Compare a collection 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

# Project

- Rules
  - **Combine with other classes / research / credits / anything**
    - You have our blanket permission
    - Get permission from other instructors; delineate different parts
  - Must be done this semester.
  - Groups of 3-4
- Expectations
  - 20% of final grade = individual effort equivalent to 1 HW
  - Expectation scales with team size
  - Most work will be done in Nov but please plan early.

# Project Ideas

- ML Reproducibility Challenge 2020
  - https://paperswithcode.com/rc2020

Browse State-of-the-Art    Methods    Trends    About    RC2020

- RC2020
  - Registration
  - Task
  - Resources

## ML Reproducibility Challenge 2020

Welcome to the ML Reproducibility Challenge 2020! This is already the fourth edition of this event (see V1, V2, V3), and we are excited this year to announce that we are broadening our coverage of conferences and papers to cover several new top venues, including: NeurIPS, ICML, ICLR, ACL, EMNLP, CVPR and ECCV.

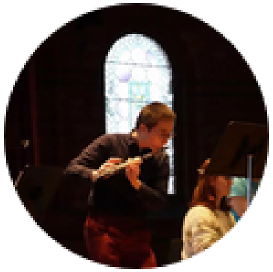
The primary goal of this event is to encourage the publishing and sharing of scientific results that are reliable and reproducible. In support of this, the objective of this challenge is to investigate reproducibility of papers accepted for publication at top conferences by inviting members of the community at large to select a paper, and verify the empirical results and claims in the paper by reproducing the computational experiments, either via a new implementation or using code/data or other information provided by the authors.

All submitted reports will be peer reviewed and shown next to the original papers on Papers with Code. Reports will be peer-reviewed via OpenReview. Every year, a small number of these reports, selected for their clarity, thoroughness, correctness and insights, are selected for publication in a special edition of the journal ReScience. (see J1, J2).

# TAs

Sameer Dharur

Joanne Truong



Yihao Chen

Michael Piseno

Hrishikesh Kale

Tianyu Zhan

Prabhav Chawla

Guillermo Nicolas Grande

# Computing

- Major bottleneck
  - GPUs

- Options
  - Your own / group / advisor’s resources
  - Google Cloud Credits
    - $50 credits to every registered student courtesy Google
  - Google Colab
    - jupyter-notebook + free GPU instance

# Administrativia

- Project Teams
  - https://gtvault-my.sharepoint.com/:x:/g/personal/dbatra8_gatech_edu/EY4_65XOzWtOkXSSz2WgpoUBY8ux2gY9PsRzR6KnglIFEQ?e=4tnKWI
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names

change ok.

# Recap from last time

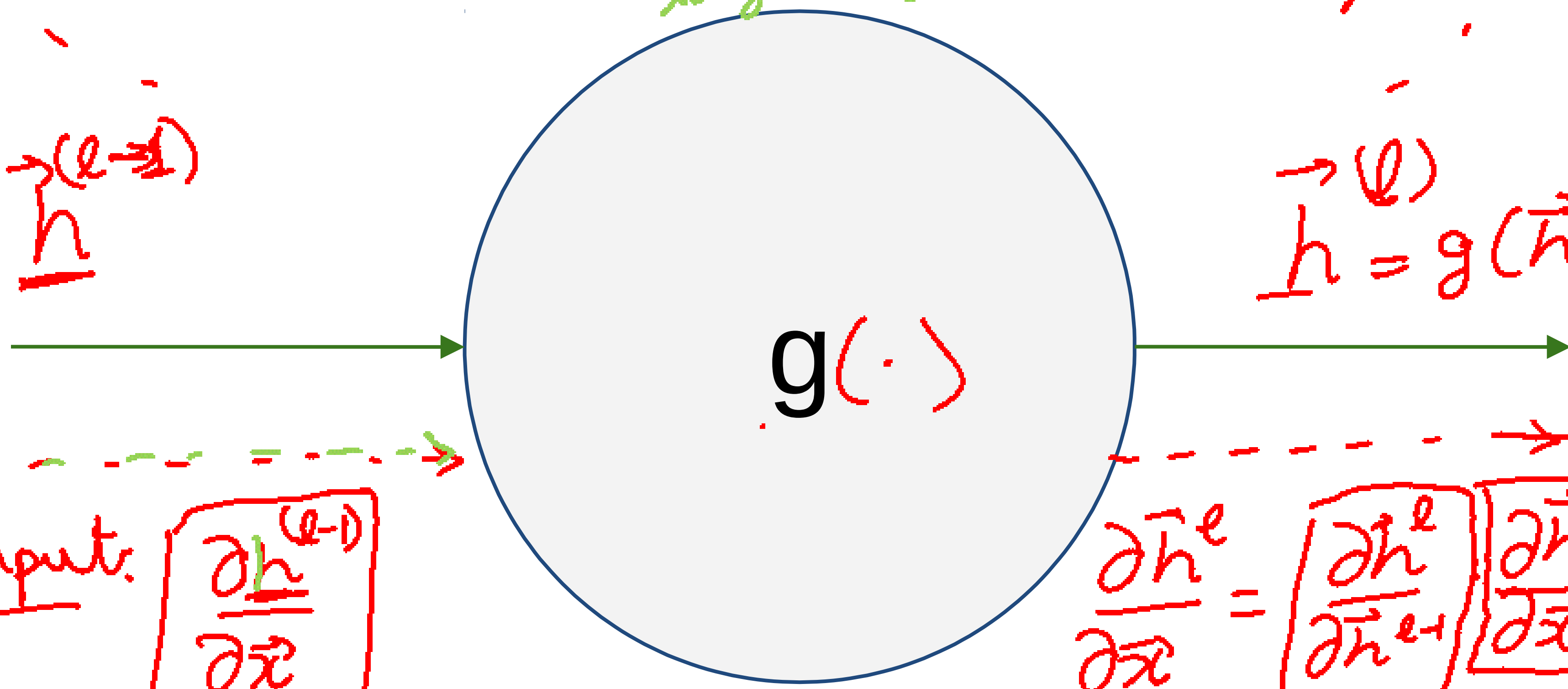
# Deep Learning = Differentiable Programming

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




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

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

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

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

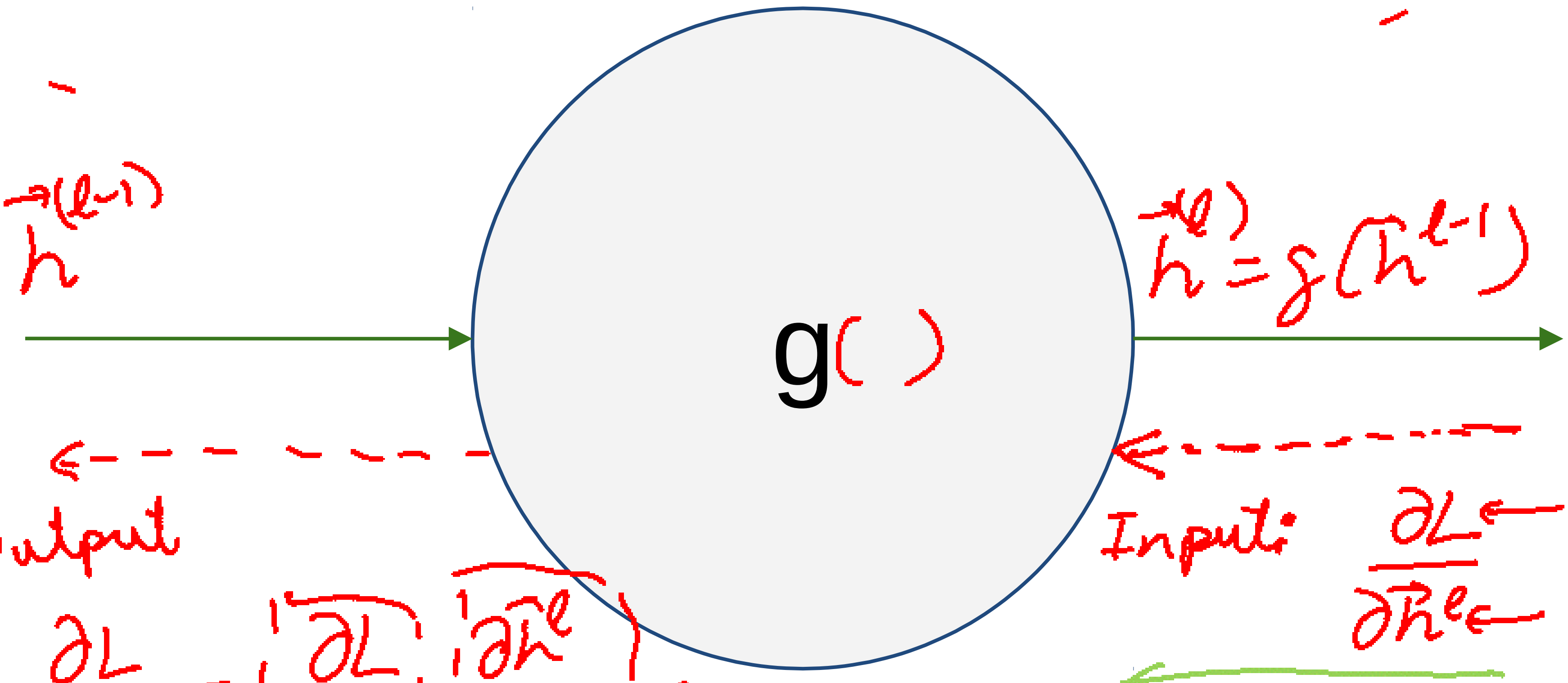
Layer Jacobian

Input FM-AD

# Reverse mode AD

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

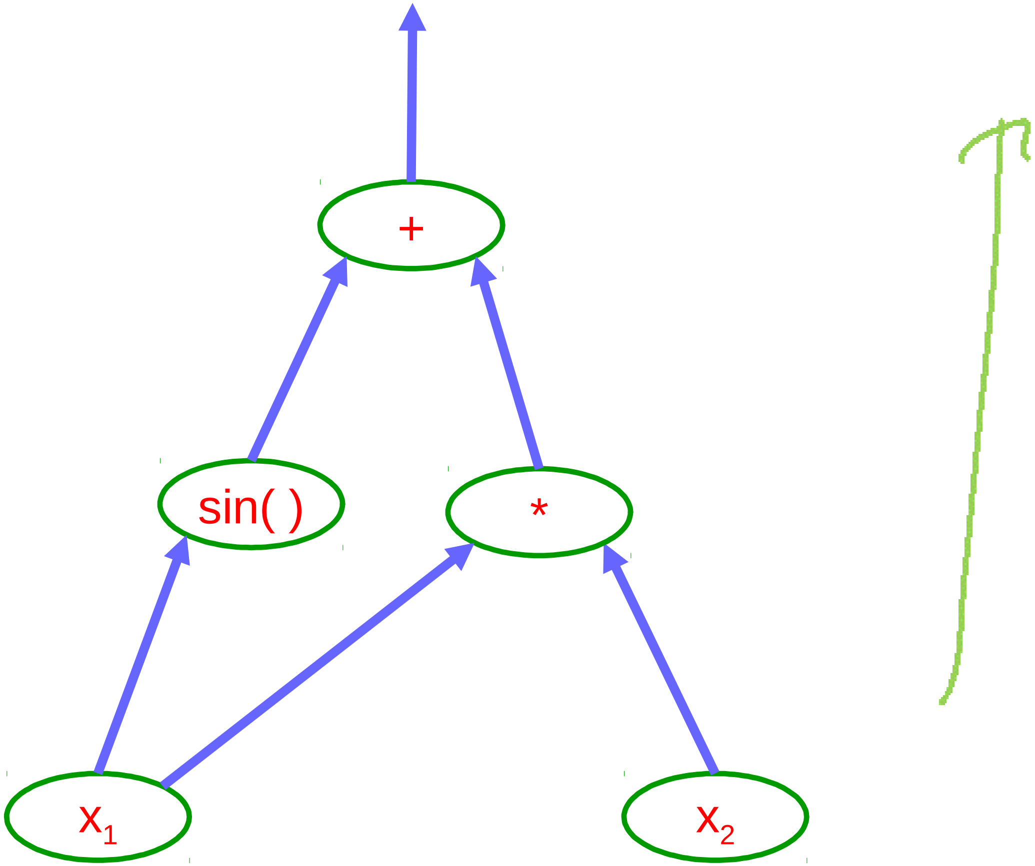
$x^{l-1}$ $L$




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

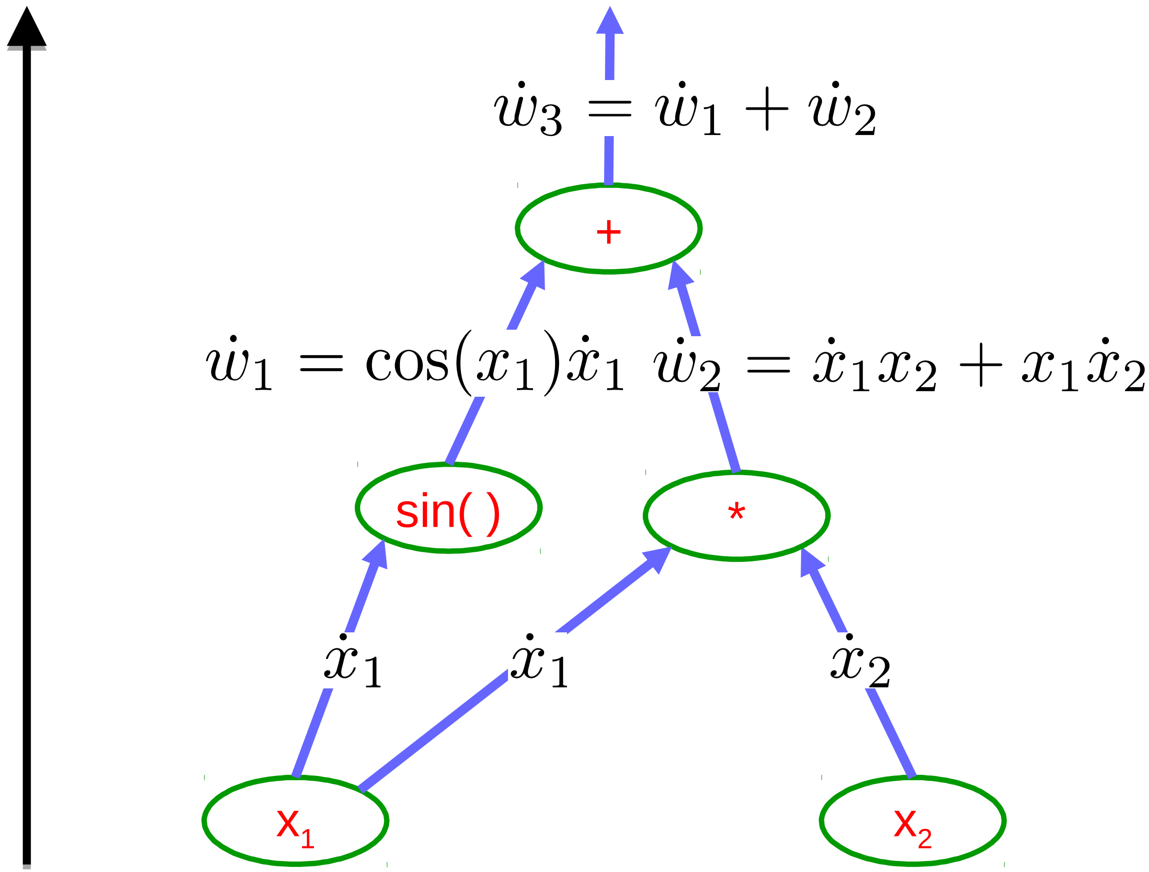
# Example: Forward mode AD

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



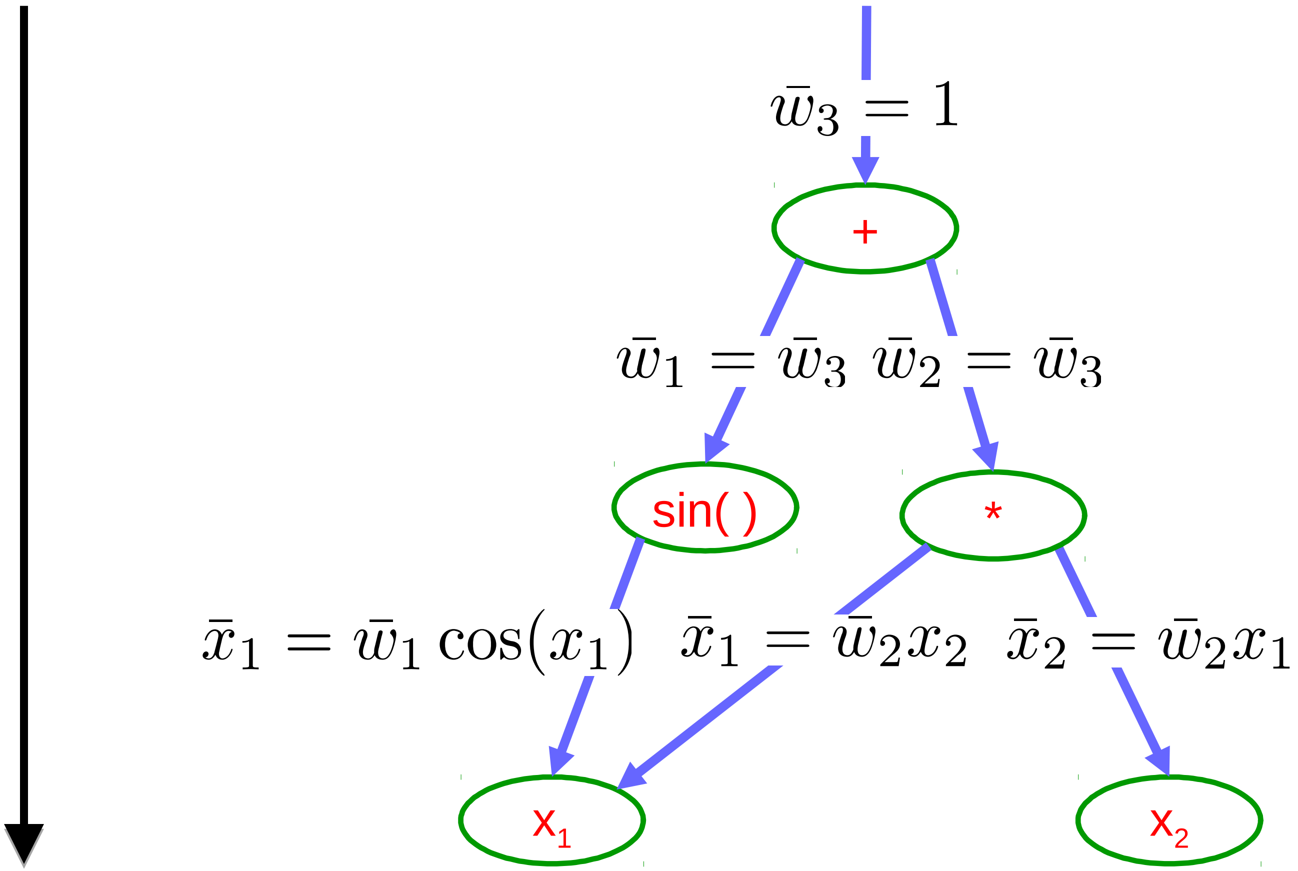
# Example: Forward mode AD

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



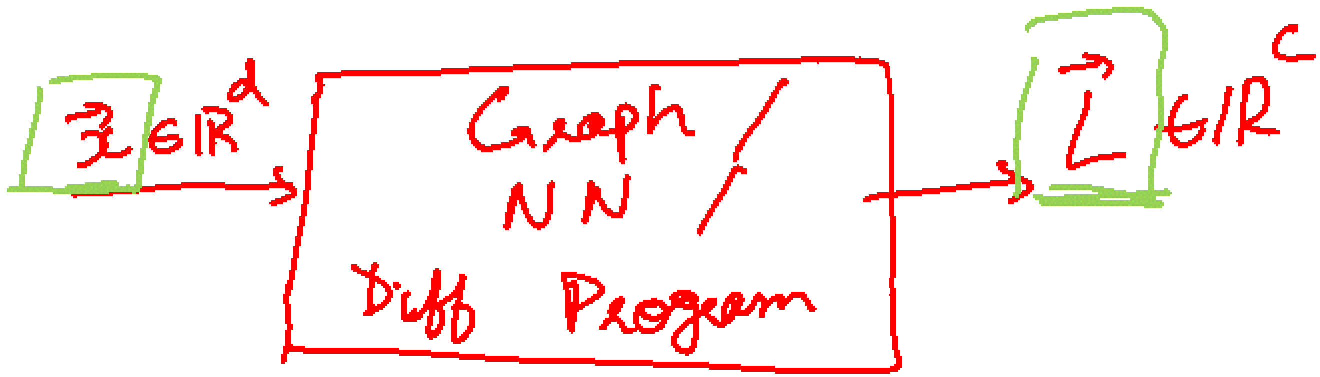
# Example: Reverse mode AD

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



# Forward mode vs Reverse Mode

- x ✉ Graph ✉ L
- Intuition of Jacobian




for i in 1:c

RM-AD output in 1 pass

$$\frac{\partial \vec{L}}{\partial \vec{x}} = \begin{array}{cc} & j \\ i & \begin{bmatrix} & \vdots & \\ \cdots & \frac{\partial L_i}{\partial x_j} & \cdots \\ & \vdots & \end{bmatrix} \end{array}$$

c x d

for j in 1:d

FM-AD outputs in 1-pass

# Forward mode vs Reverse Mode

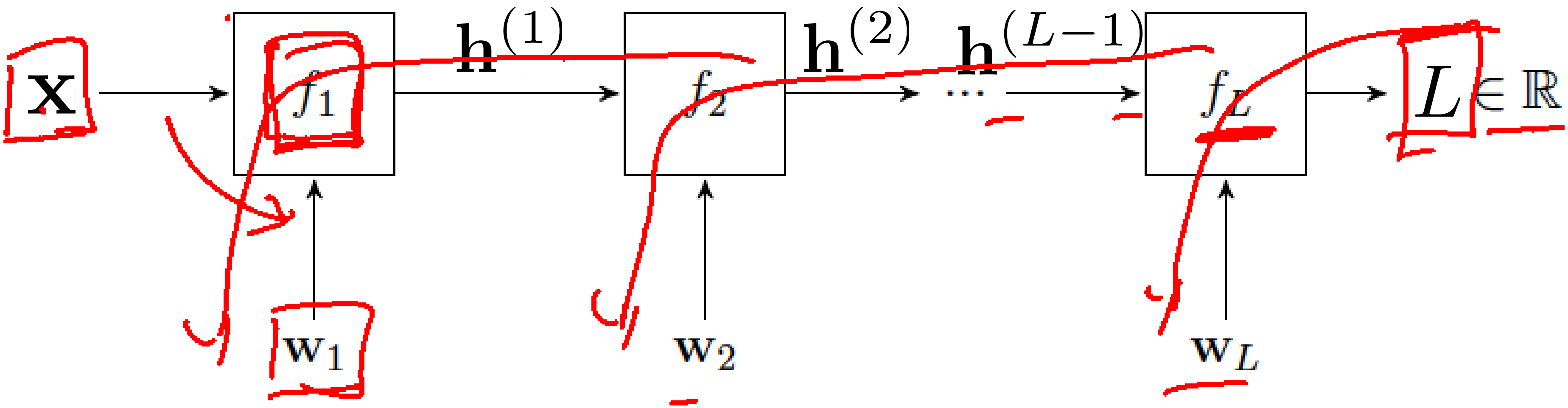
- What are the differences?

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

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

# Neural Network Computation Graph

$h^{1} = \max(0, W_1 x)$




$$\frac{\partial L}{\partial w_{(i)}}$$

# Backprop

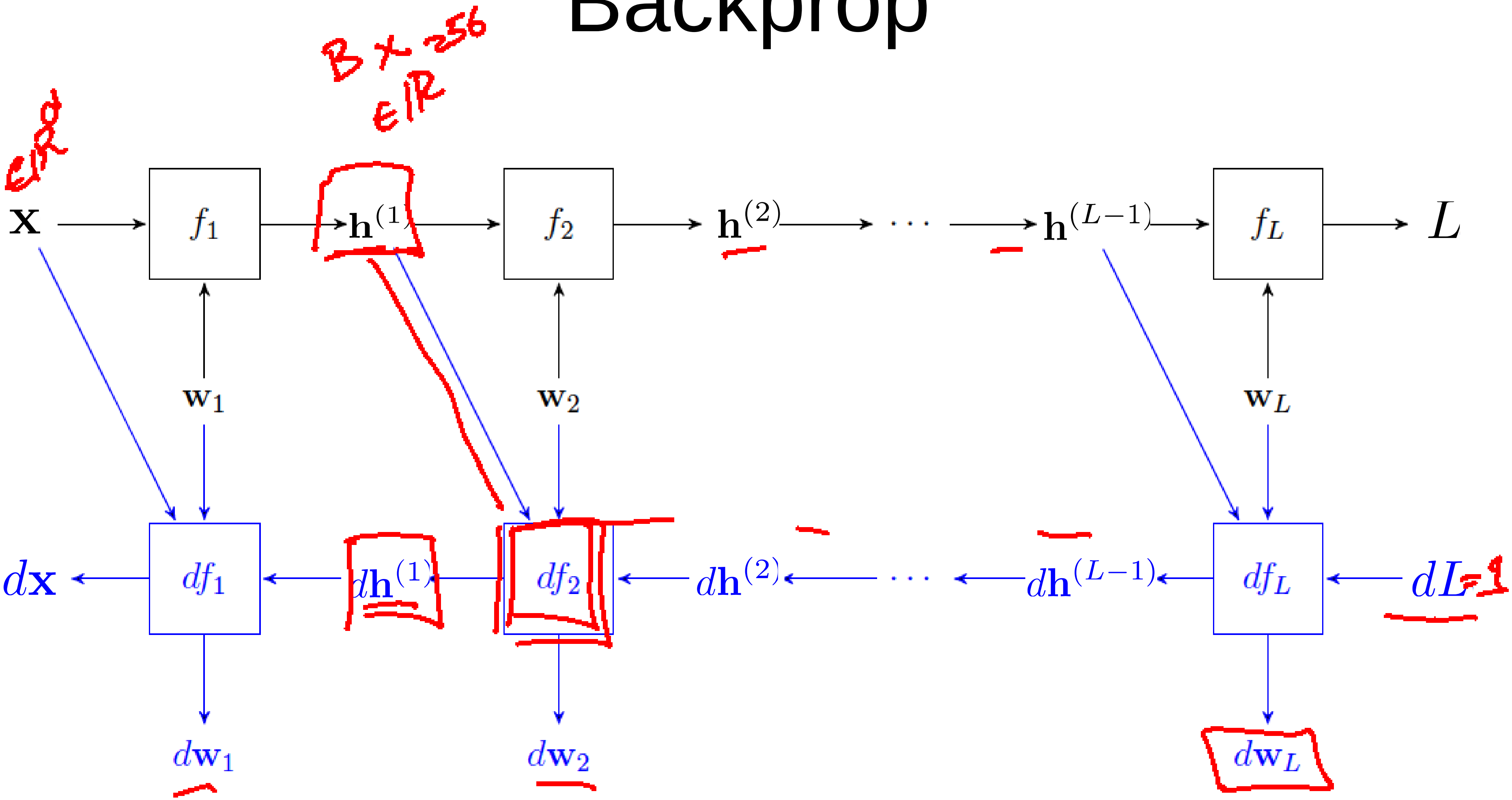
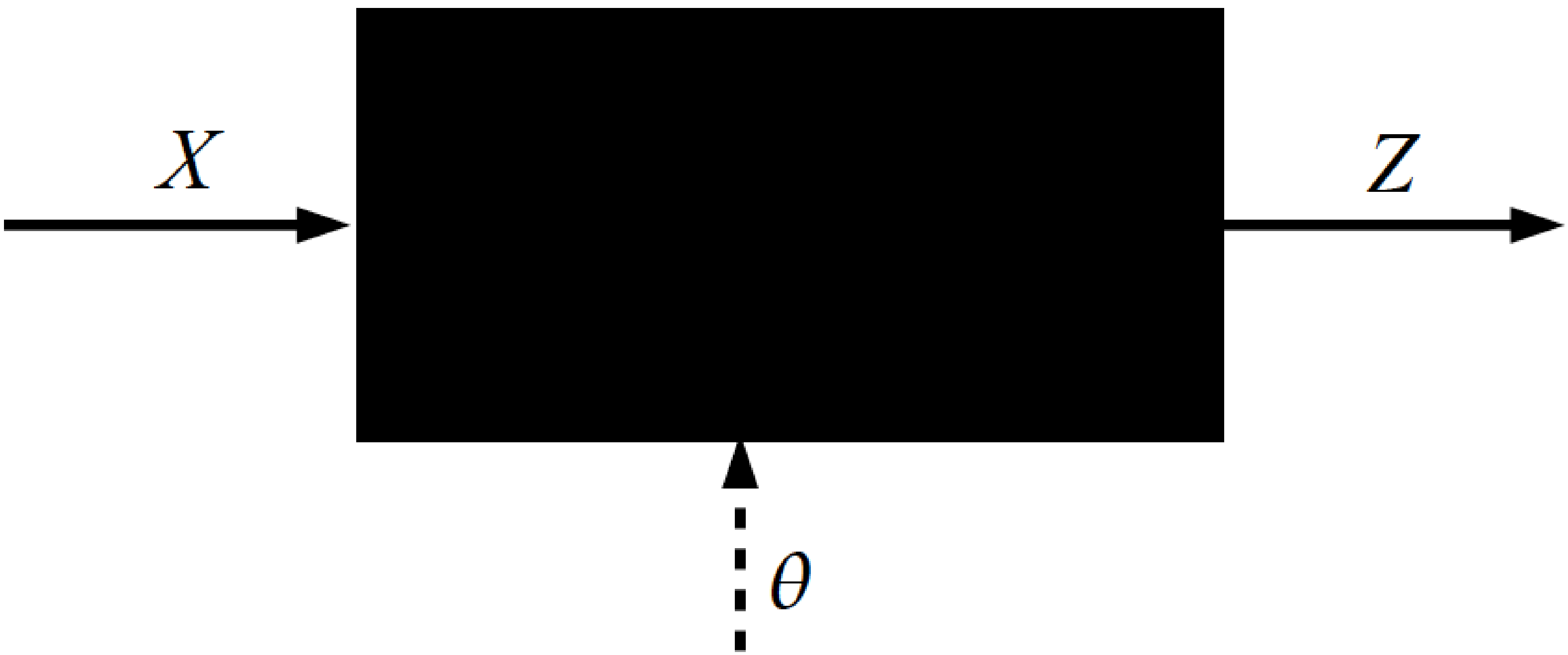


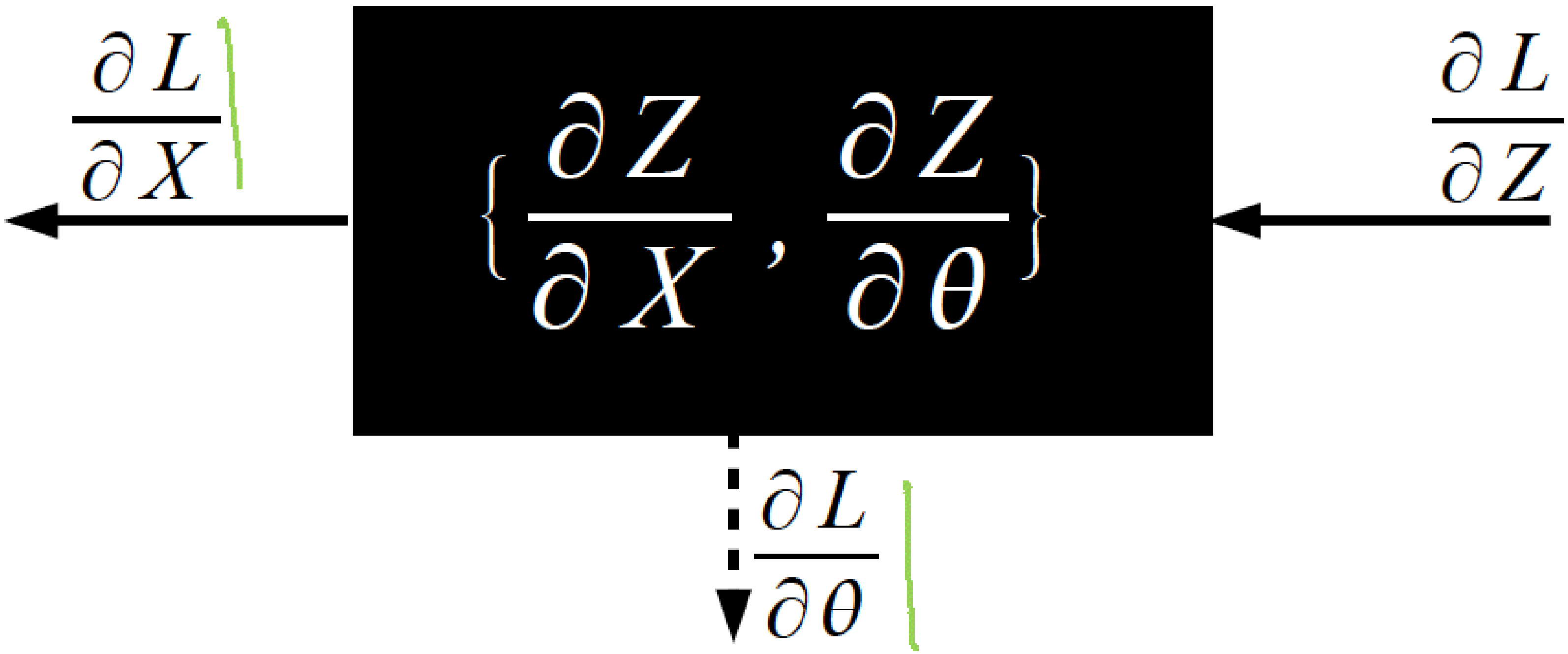



Figure Credit: Andrea Vedaldi

# Key Computation: Forward-Prop

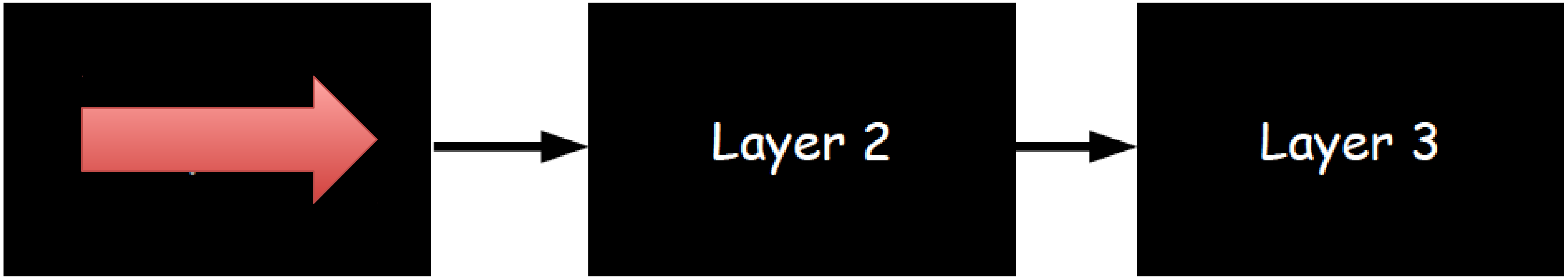


# Key Computation: Back-Prop



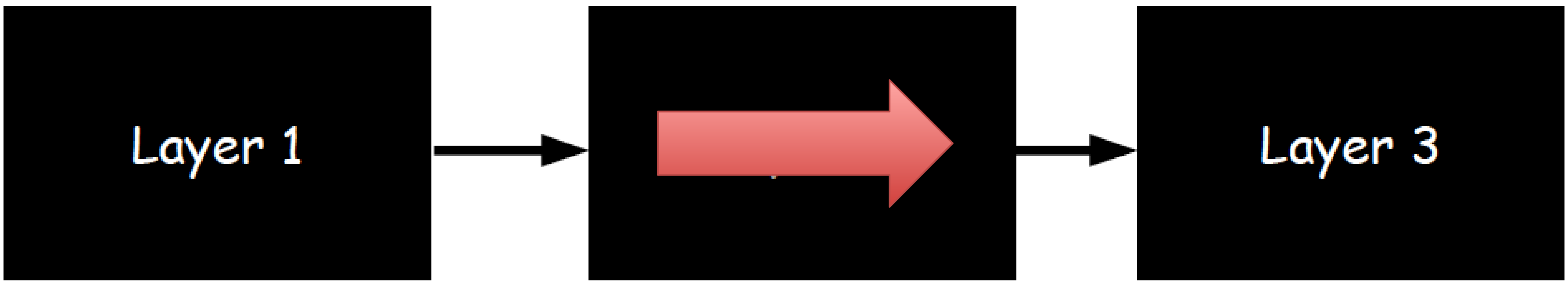
# Neural Network Training

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



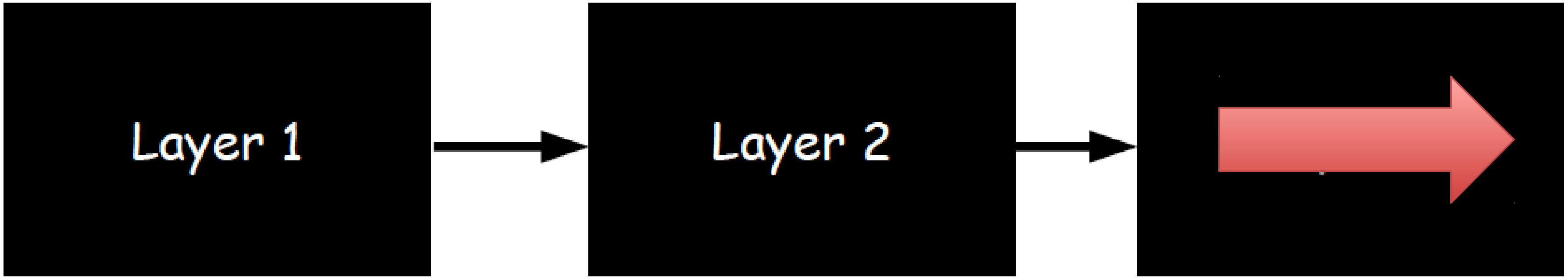
# Neural Network Training

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



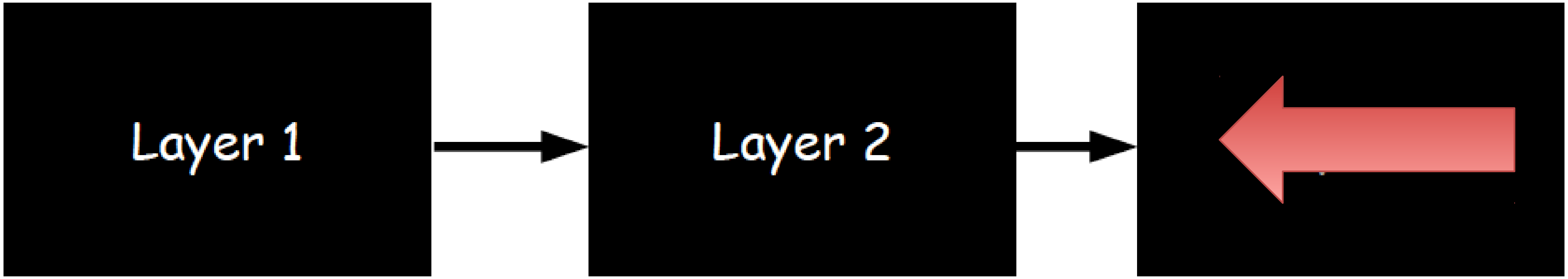
# Neural Network Training

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



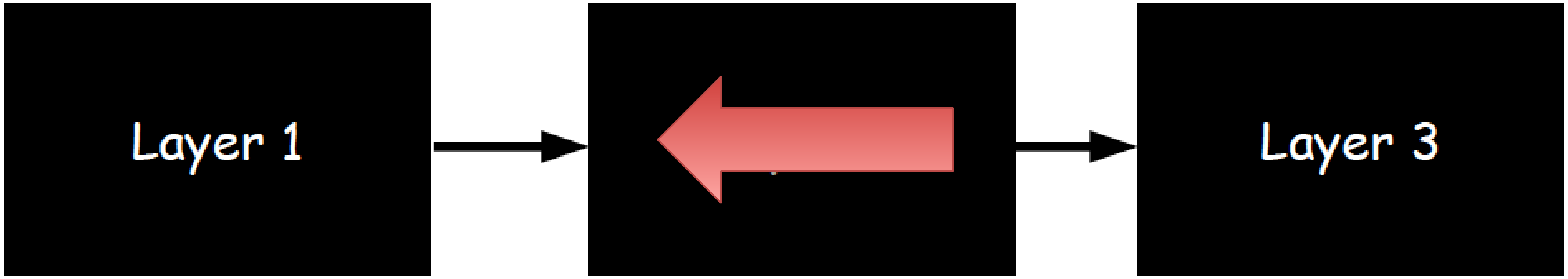
# Neural Network Training

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



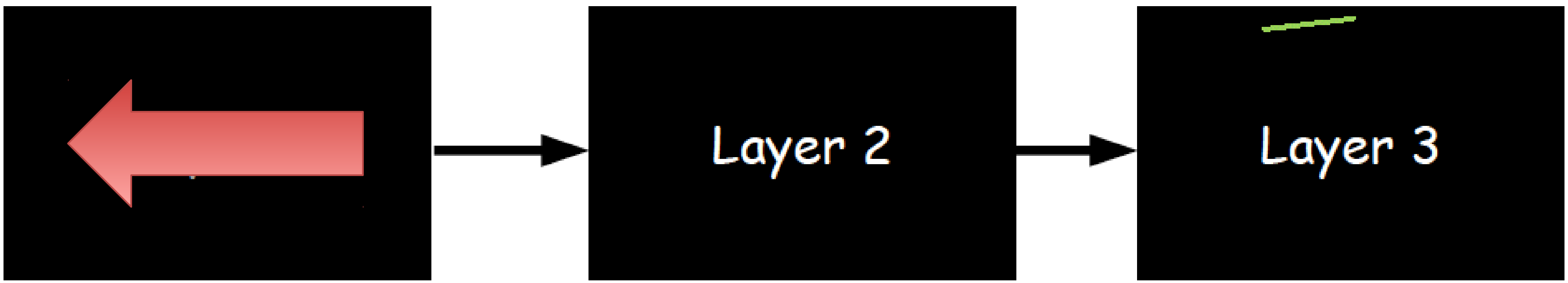
# Neural Network Training

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



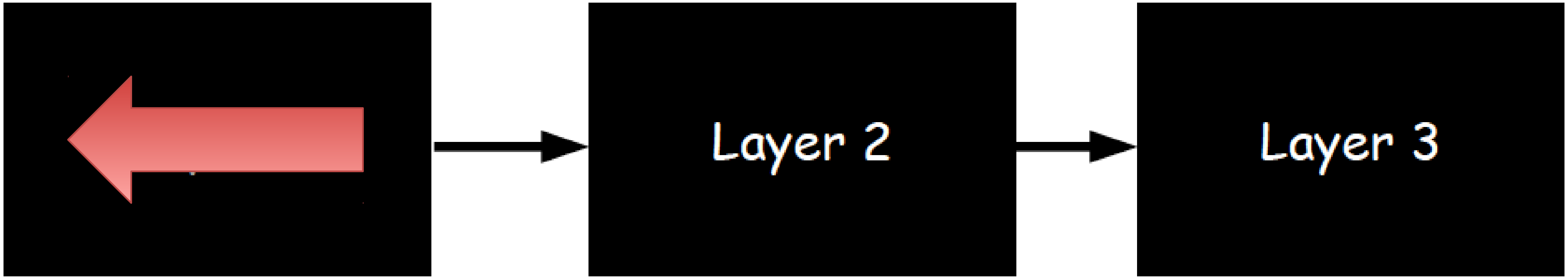
# Neural Network Training

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



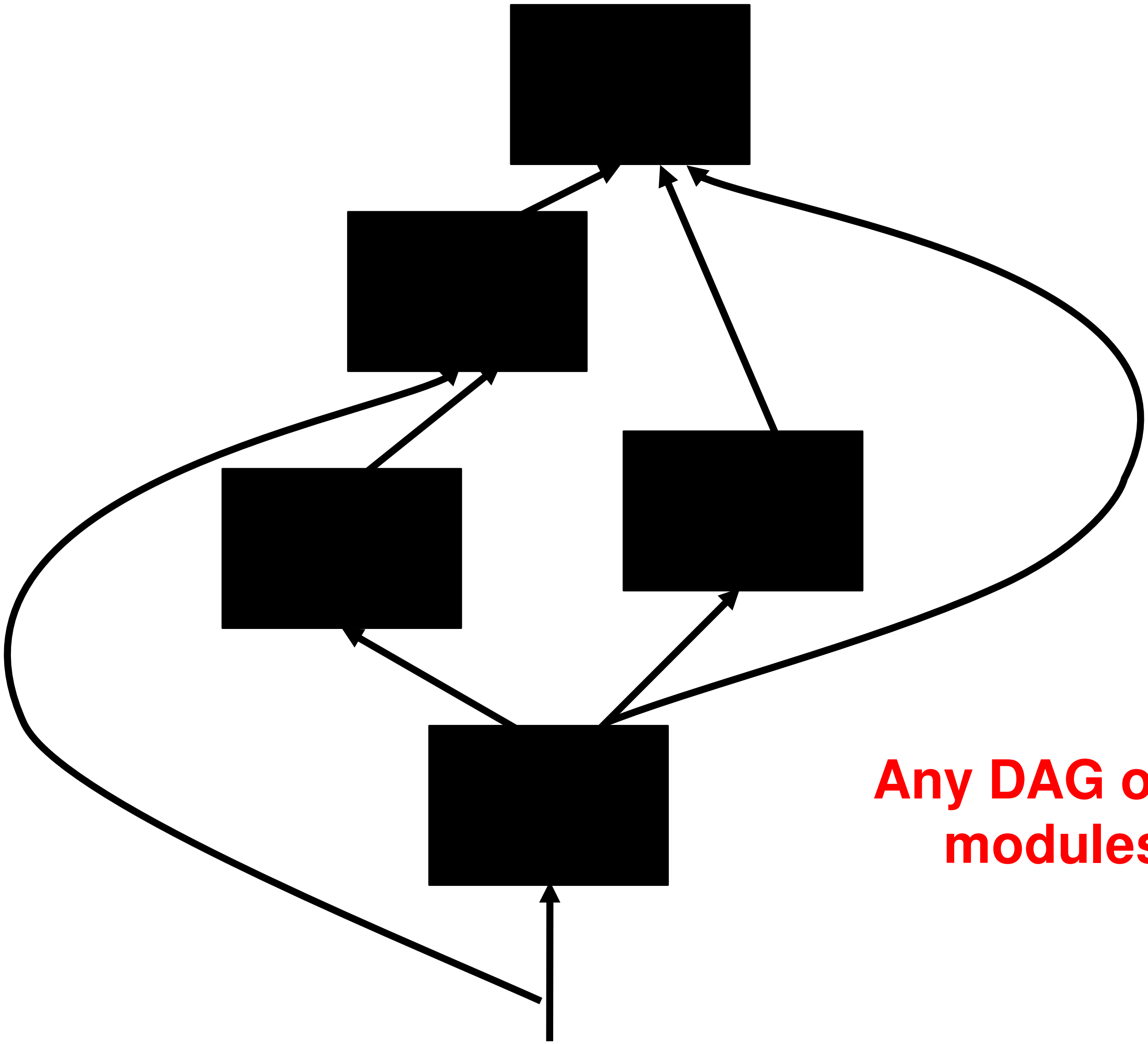
# Neural Network Training

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




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

# Computational Graph



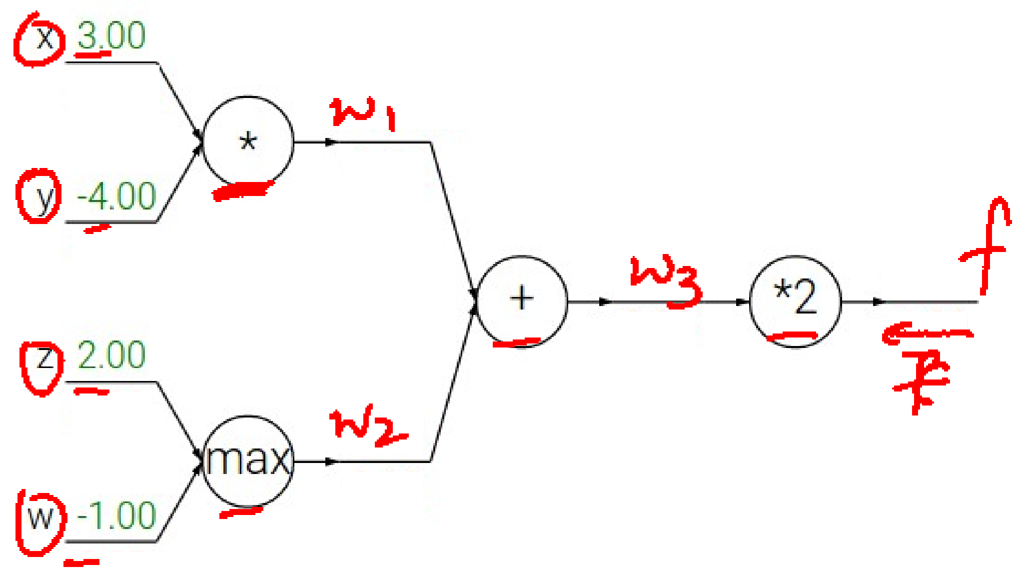
**Any DAG of differentiable modules is allowed!**

# Plan for Today

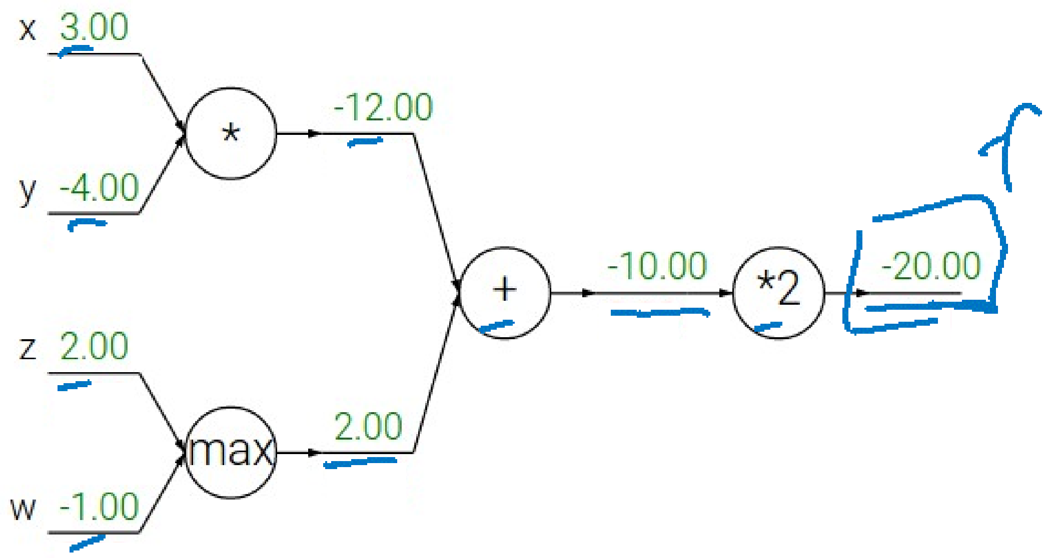
- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

# Backpropagation: a simple example

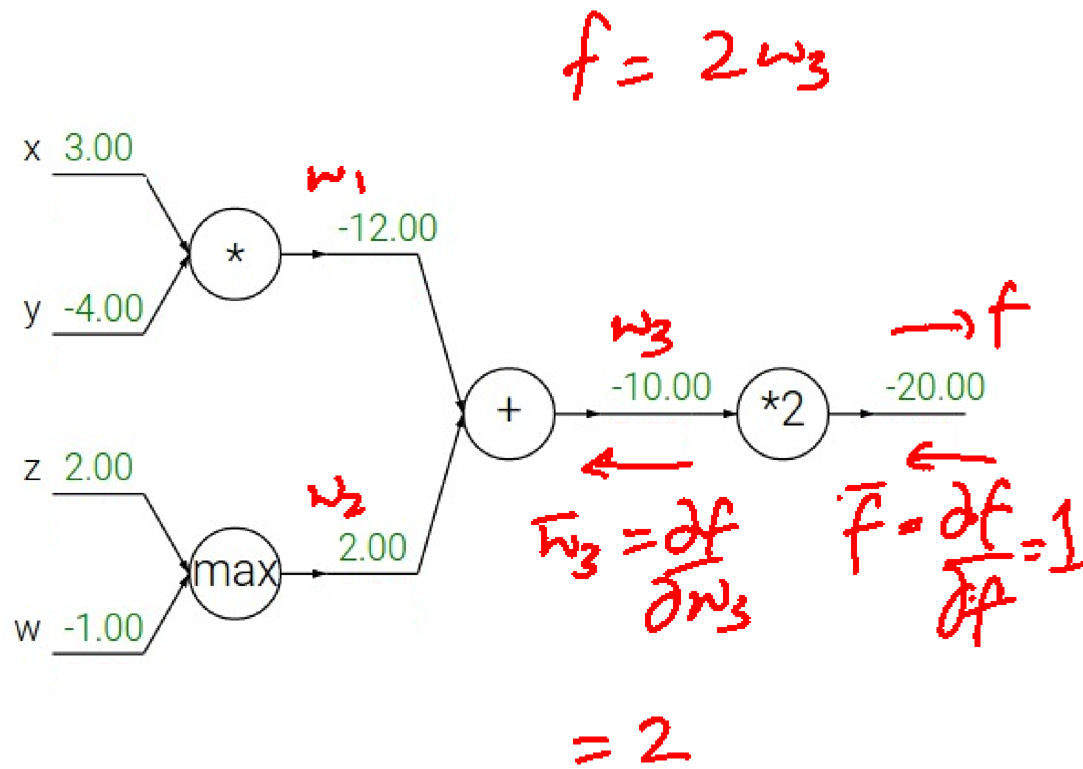
$$f(x,y,z,w) = 2\Big(\underbrace{xy}_{w_1} + \underbrace{\max\{z,w\}}_{w_2}\Big)$$



# Forward pass



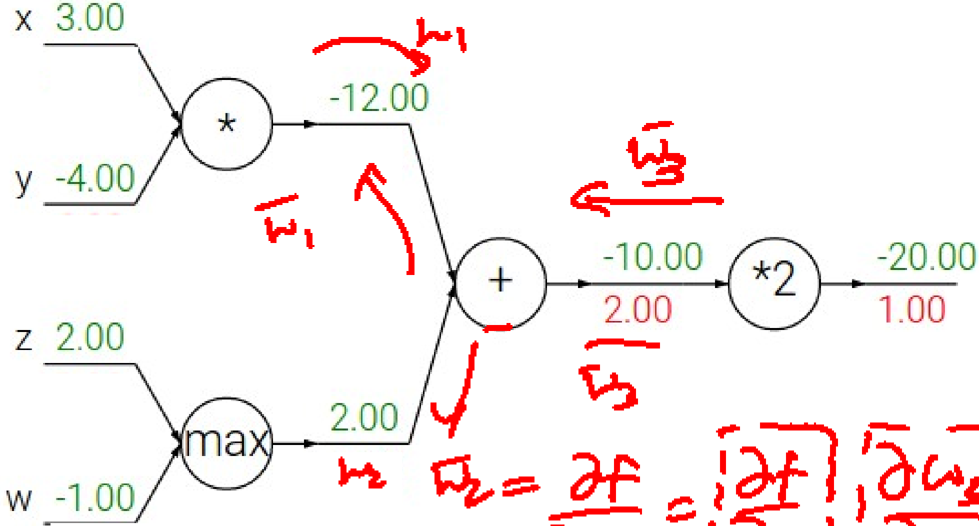
# Patterns in backprop



# Patterns in backprop

$w_3 = w_1 + w_2$

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




$\bar{w}_2 = \frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial w_3} \cdot \frac{\partial w_3}{\partial w_2}$

$\bar{w}_3$ Jacobian 1

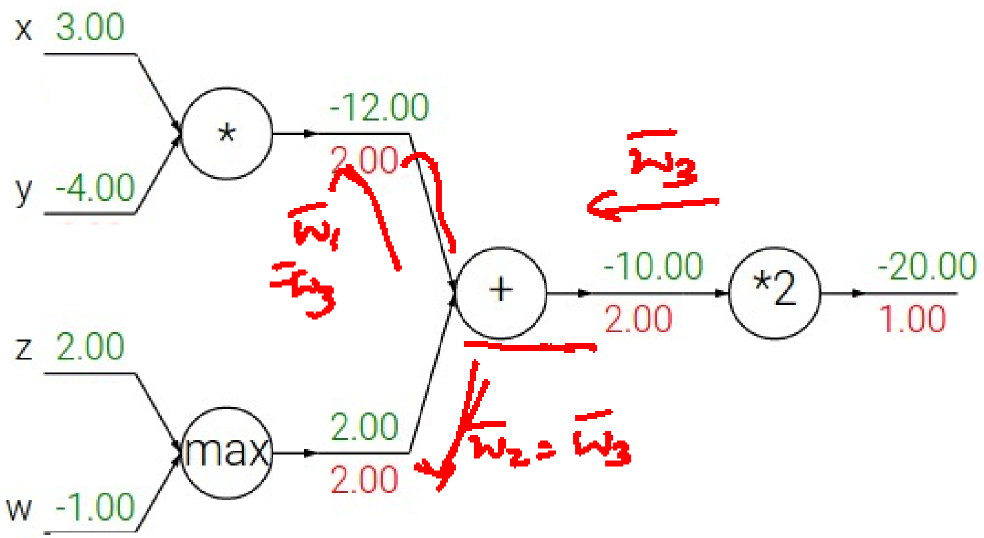
$\bar{w}_2 = \bar{w}_3$

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

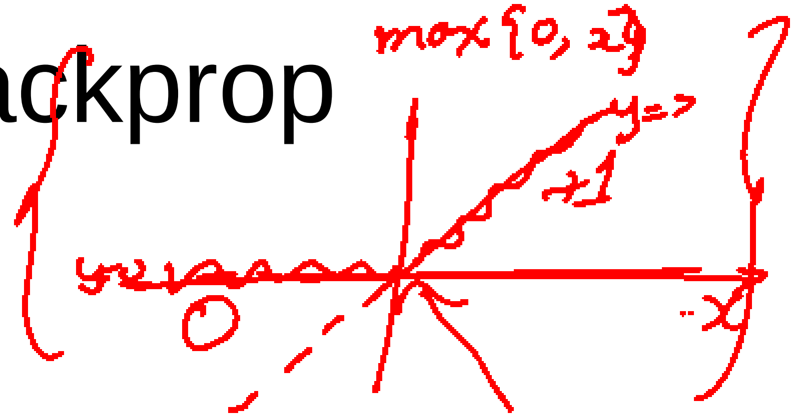
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# Patterns in backprop

**add** gate: gradient distributor



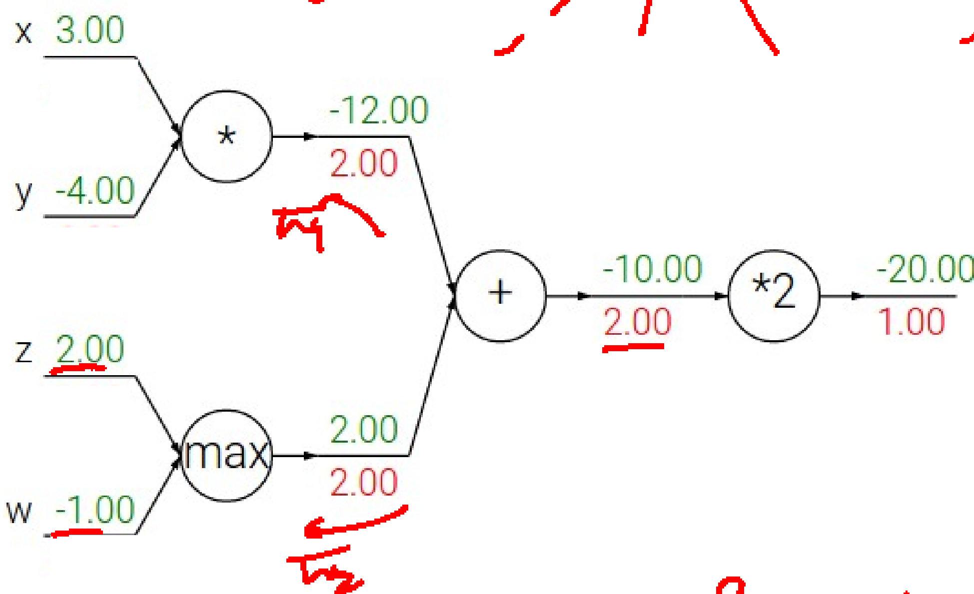
# Patterns in backprop




**add** gate: gradient distributor

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

$$\bar{z} = \frac{\partial f}{\partial z} = \underbrace{\frac{\partial f}{\partial w_2}}_{\bar{w}_2} \frac{\partial w_2}{\partial z}$$




$$\frac{\partial w_2}{\partial z} = \begin{cases} +1 & \text{if } z > w \\ 0 & \text{if } z < w \\ \text{Not defined} & \text{if } z = w \end{cases}$$

$$w_2 = \max\{z, w\} = \begin{cases} z & \text{if } z \geq w \\ w & \text{else} \end{cases}$$

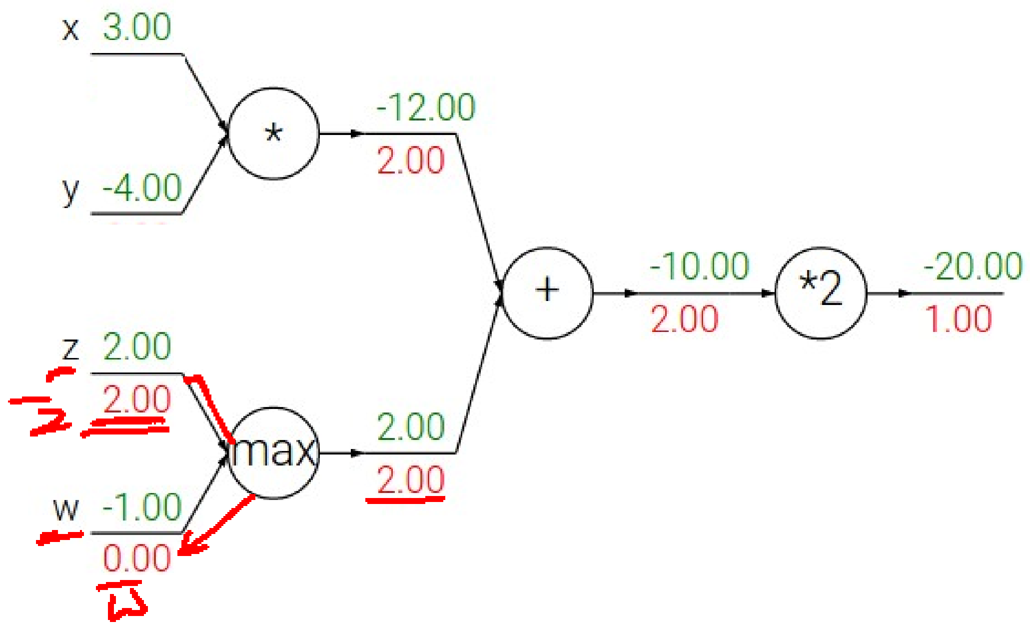
$$\frac{\partial w_2}{\partial w}$$

# Patterns in backprop

max{0, x} 0 +1

**add** gate: gradient distributor

**max** gate: gradient router



# Patterns in backprop

$w_1 = xy$

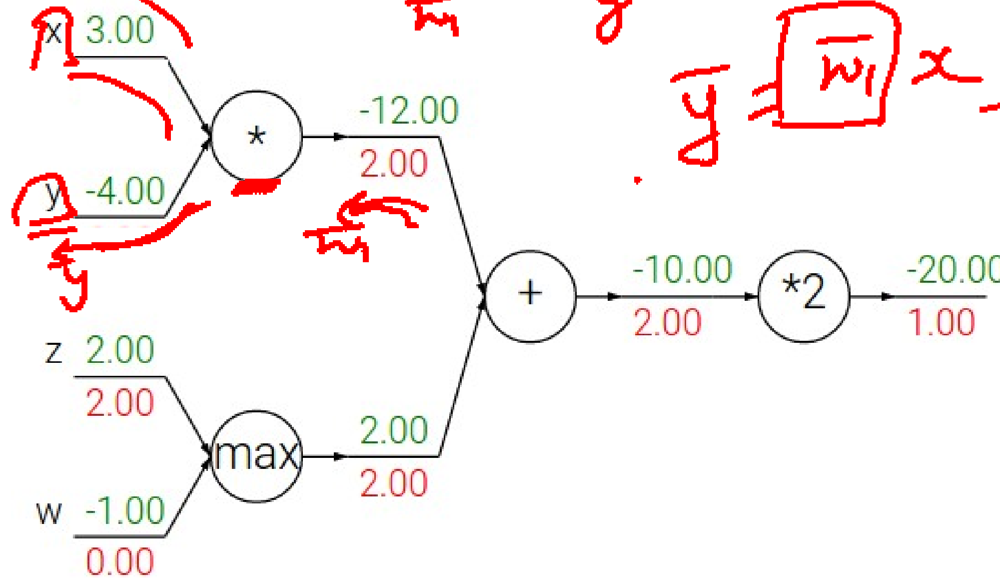
$\bar{x} = \frac{\partial f}{\partial x} = \frac{\partial f}{\partial w_1} \cdot \frac{\partial w_1}{\partial x} = \bar{w_1}\, y$

$\bar{y} = \bar{w_1}\, x$

**add** gate: gradient distributor

**max** gate: gradient router

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



Aside

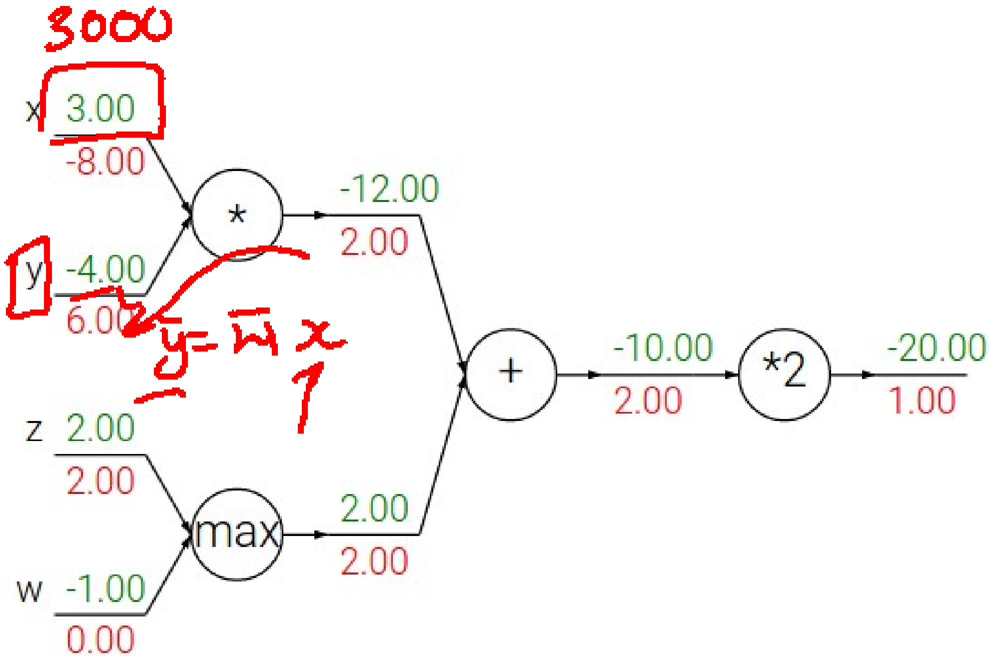
# Patterns in backprop

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

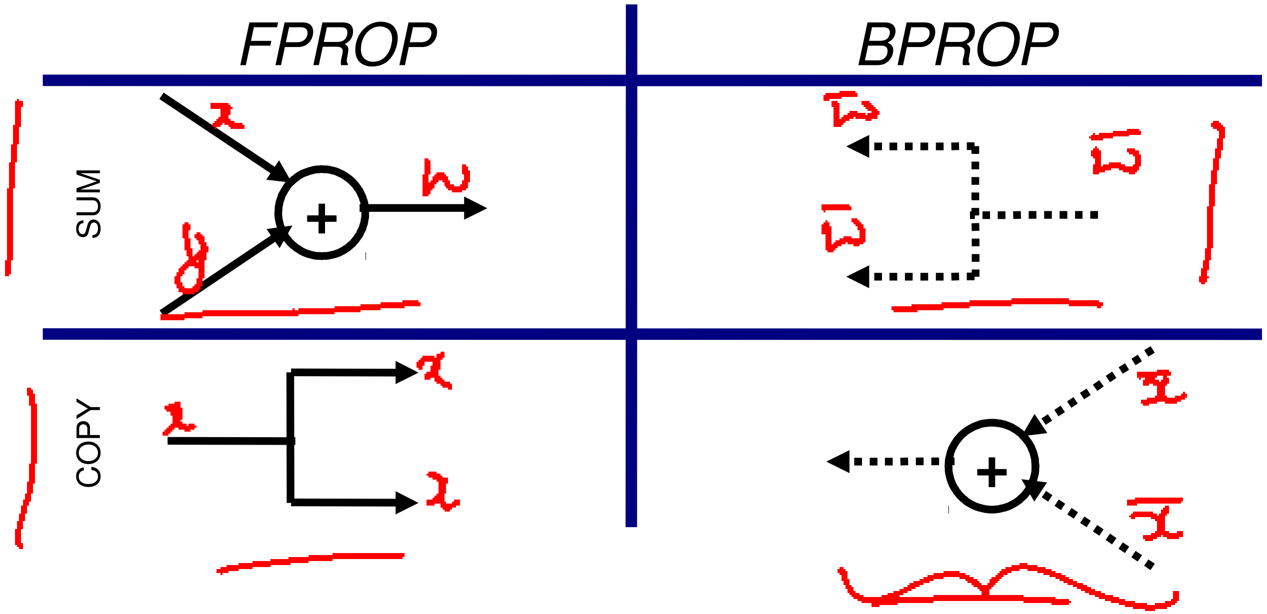
**add** gate: gradient distributor

**max** gate: gradient router

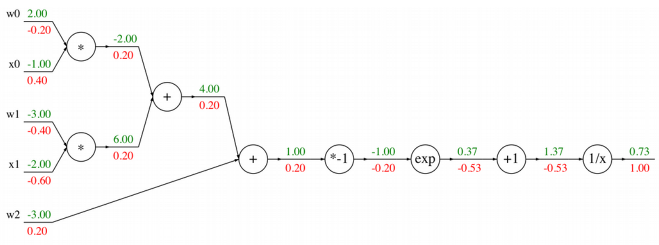
**mul** gate: gradient switcher



# Duality in Fprop and Bprop



# Modularized implementation: forward / backward API

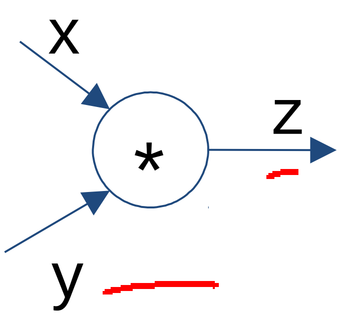



Graph (or Net) object *(rough psuedo code)*

```
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



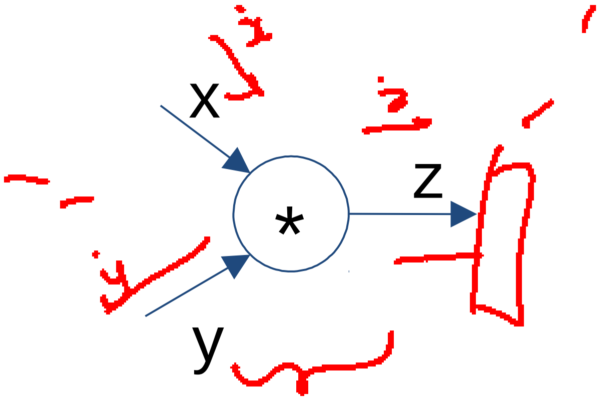

(x,y,z are scalars)

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

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

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

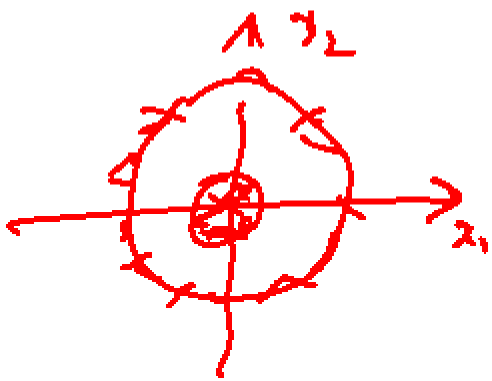
# Modularized implementation: forward / backward API




(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]
```

memory overhead of reverse mode AD.



# 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 |

```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
```



# Caffe Sigmoid Layer

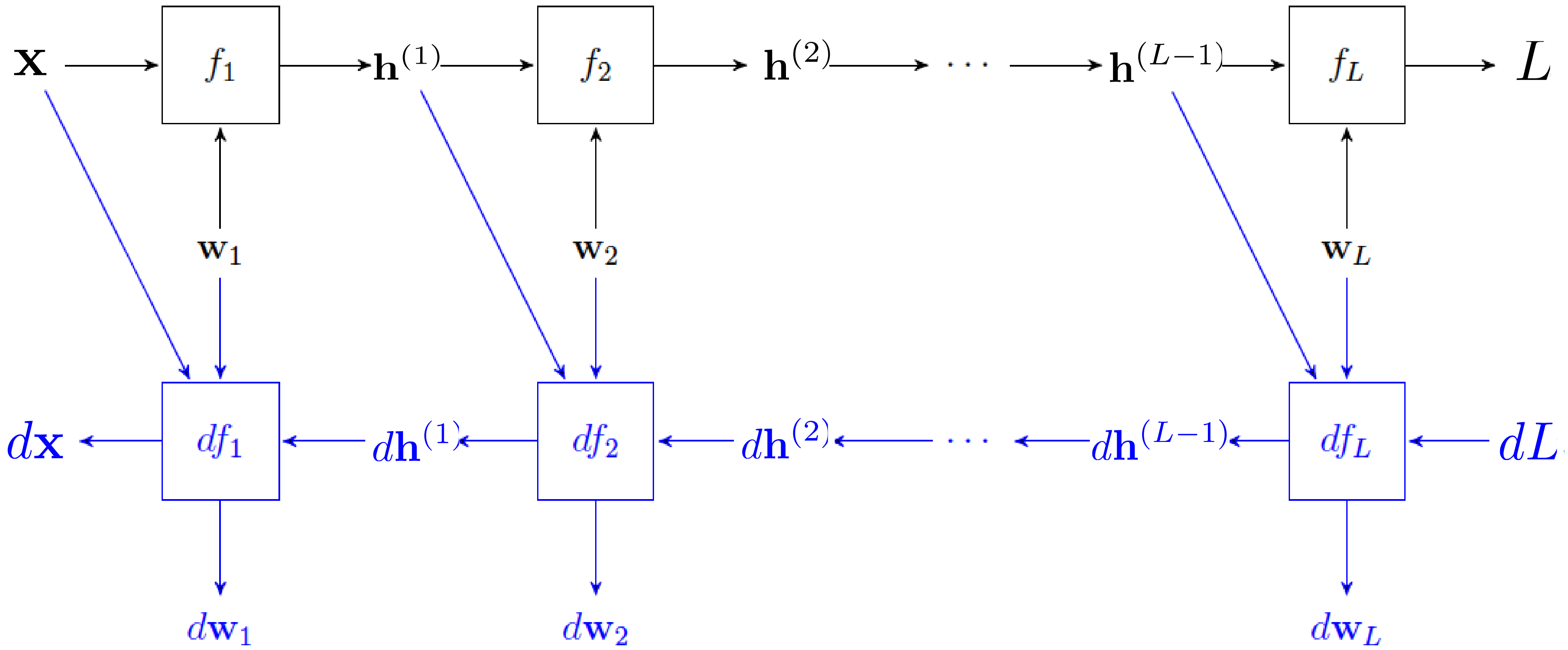
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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

# Plan for Today

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

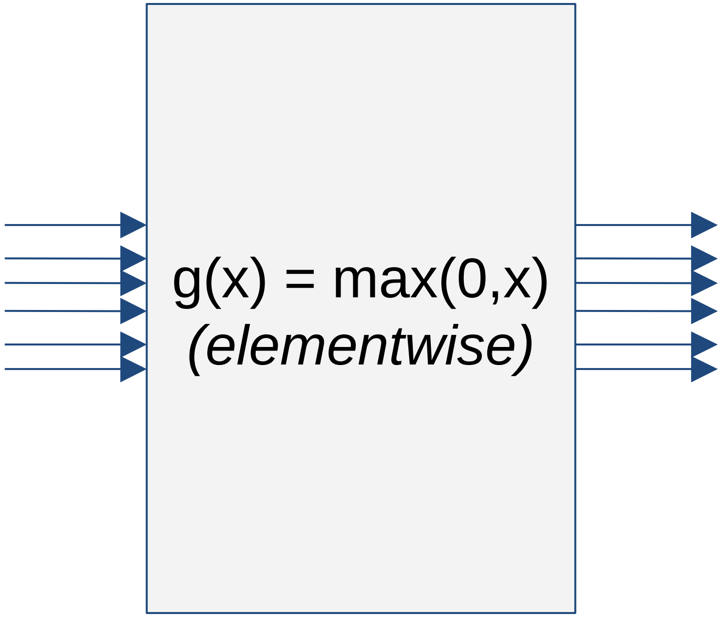
# Backprop



$\mathbf{x}$
$f_1$
$\mathbf{h}^{(1)}$
$f_2$
$\mathbf{h}^{(2)}$
$\cdots$
$\mathbf{h}^{(L-1)}$
$f_L$
$L$
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_L$
$d\mathbf{x}$
$df_1$
$d\mathbf{h}^{(1)}$
$df_2$
$d\mathbf{h}^{(2)}$
$\cdots$
$d\mathbf{h}^{(L-1)}$
$df_L$
$dL$
$d\mathbf{w}_1$
$d\mathbf{w}_2$
$d\mathbf{w}_L$

# Jacobian of ReLU

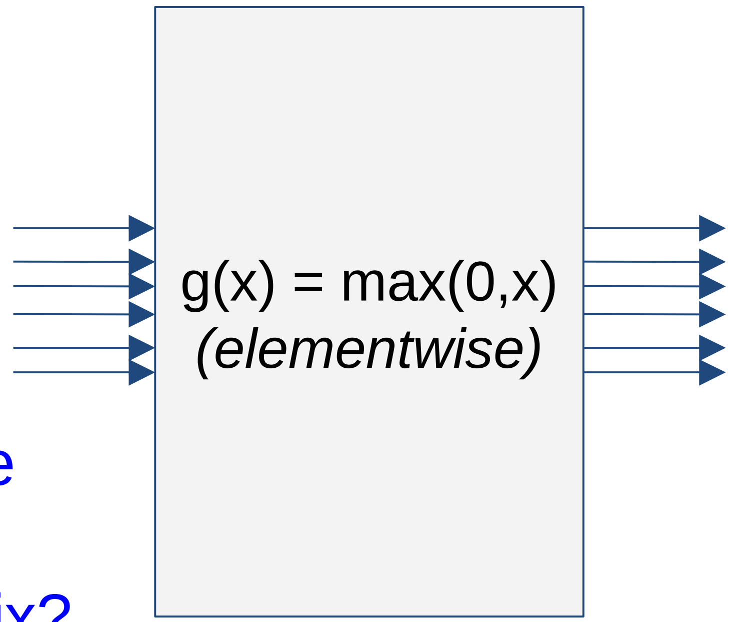
4096-d
input vector




4096-d
output vector

# Jacobian of ReLU

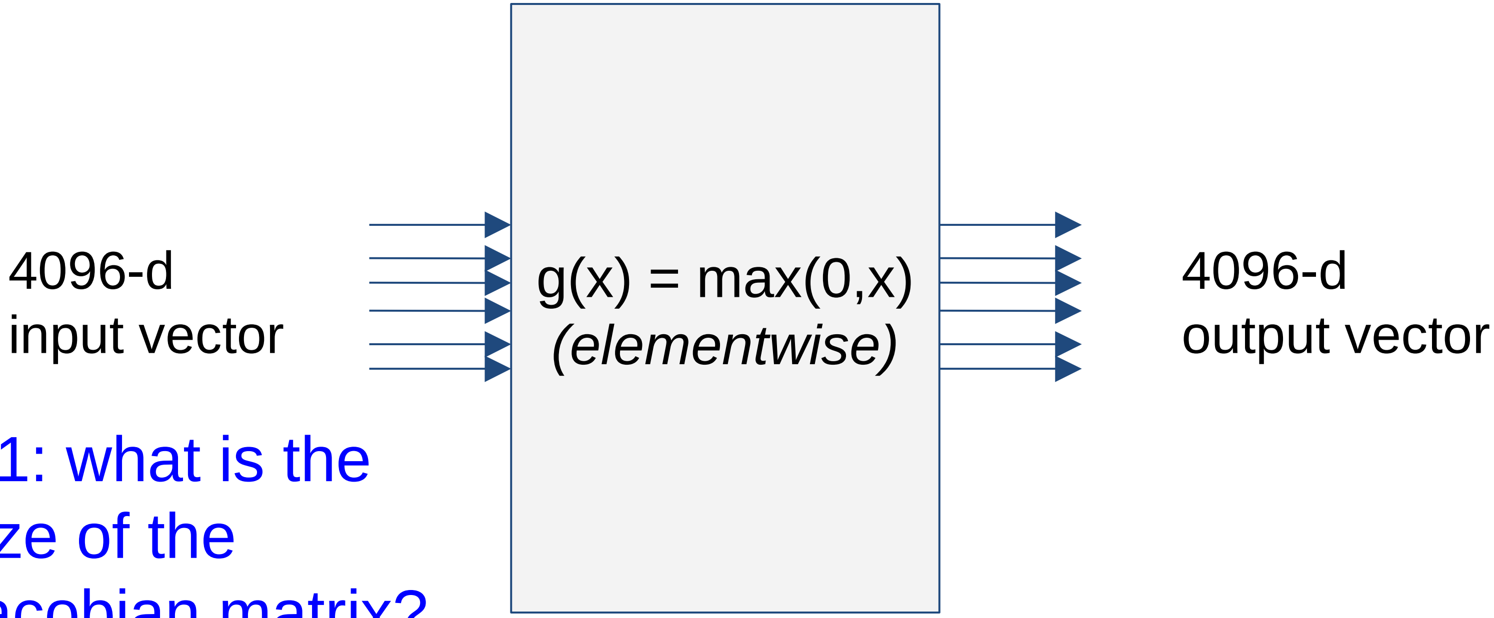
4096-d
input vector




4096-d
output vector

Q1: what is the size of the Jacobian matrix?

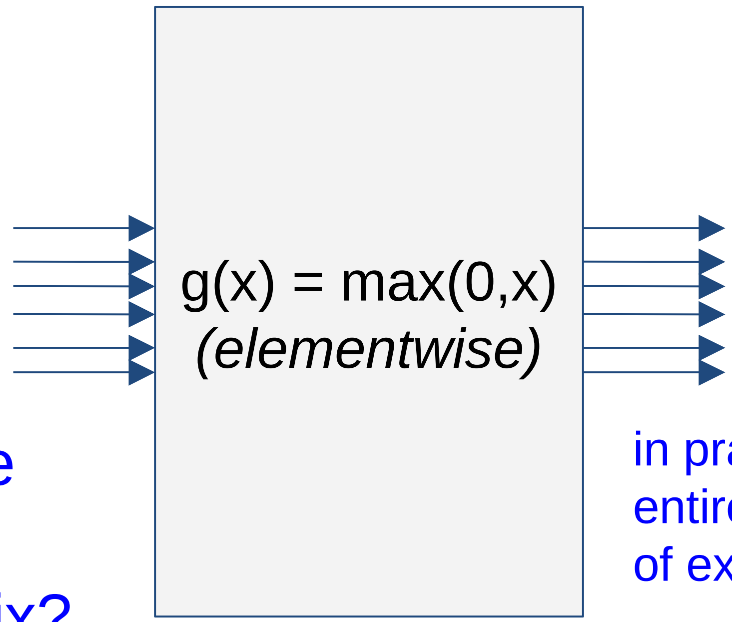
# Jacobian of ReLU




Q1: what is the size of the Jacobian matrix?
[4096 x 4096!]

# Jacobian of ReLU

4096-d
input vector




4096-d
output vector

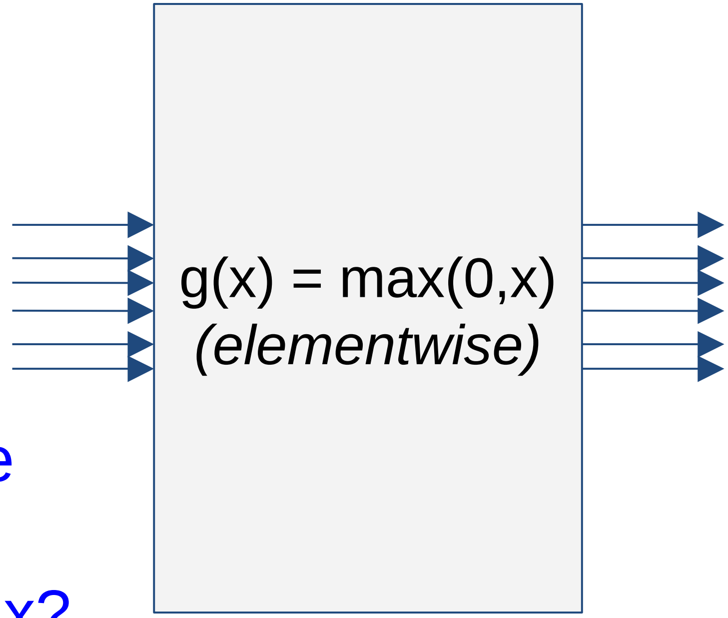
Q1: what is the size of the Jacobian matrix?
[4096 x 4096!]

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 :\

# Jacobian of ReLU

4096-d
input vector




4096-d
output vector

Q1: 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

# Jacobians of FC-Layer