CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Analytical Gradients
- Automatic Differentiation
  - Computational Graphs
  - Forward mode vs Reverse mode AD

Dhruv Batra

Georgia Tech

Administrativia

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

- Fuller schedule + future reading posted
  - https://www.cc.gatech.edu/classes/AY2020/cs7643_fall/
  - Caveat: subject to change;
    please don't make irreversible decisions based on this.

Recap from last time

$$\min_{\vec{w}} L(\vec{w}, \underline{D}) \leftarrow$$

$$\underline{L(\vec{w})} = \frac{1}{N} \sum_i L_i(w)$$

Strategy: **Follow the slope**

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

Gradient Descent

```
# 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

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

Stochastic Gradient Descent (SGD)

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

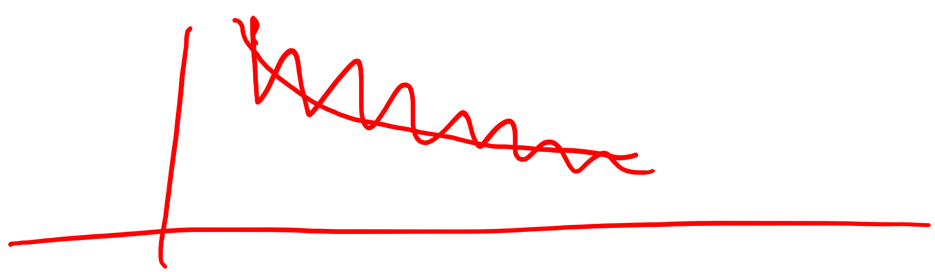
$$\nabla_W L(W) = \boxed{\frac{1}{N}\sum_{i=1}^{N}} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

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 ~~X~~
- 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 = 4*v*(1 - v)
  return v
```

or, in closed-form,

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

Coding

Symbolic Differentiation of the Closed-form

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

f'(x$_0$) $= f'(x_0)$
Exact

Automatic Differentiation

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

f'(x$_0$) $= f'(x_0)$
Exact

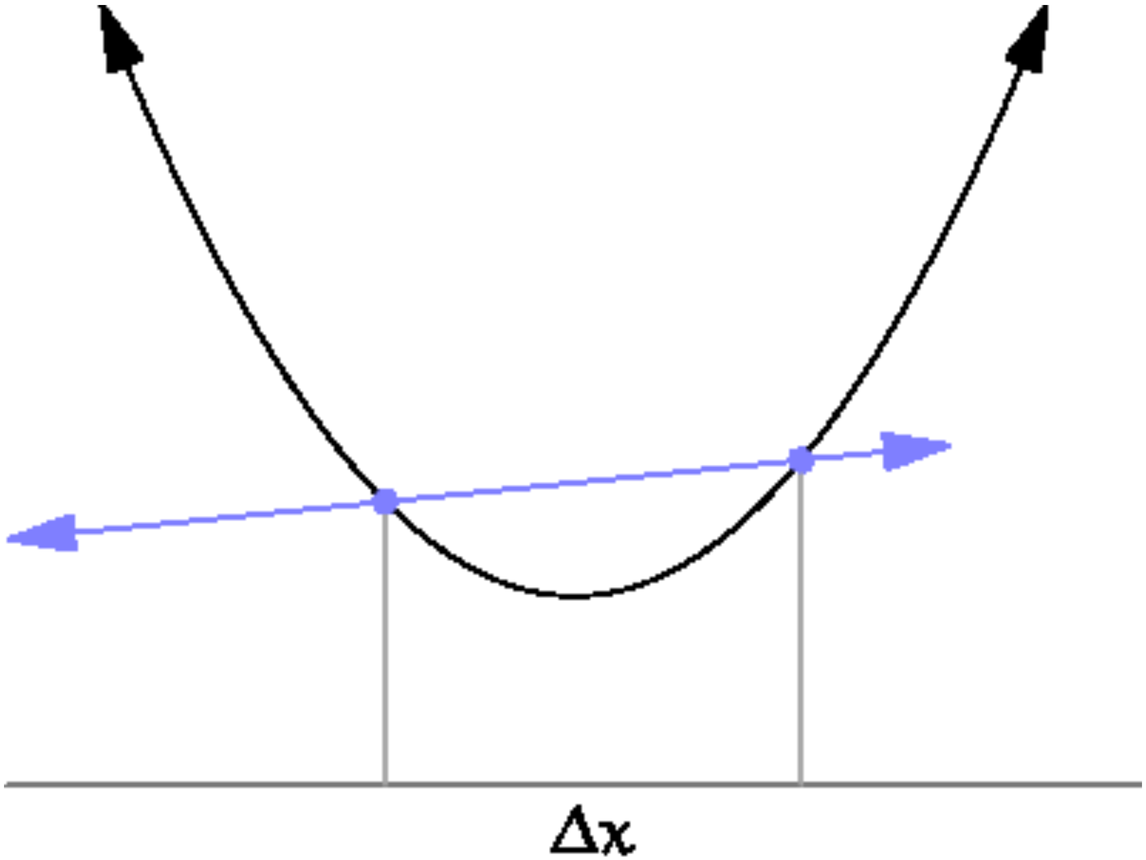
Numerical Differentiation

```
f'(x):
  h = 0.000001
  return (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”


Δx

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,
0.6,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

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

?,…]

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

	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	$\frac{\partial Y}{\partial x}$		

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

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

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

num = dim 1

rows

den = dim 2

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

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

Vector/Matrix Derivatives Notation

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_i}{\partial x_j} \end{bmatrix}_{c \times d}$$

$$\frac{\partial y_i}{\partial x_j}$$

Vector Derivative Example

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ x^2 \end{bmatrix} \qquad \frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} 1 \\ 2x \end{bmatrix}$$

$$y = \vec{w}^T \vec{x} \qquad \frac{\partial y}{\partial \vec{x}} = \left[\frac{\partial y}{\partial x_1} \; \cdots \; \frac{\partial y}{\partial x_d} \right]$$

$$= \sum_{i=1}^{d} w_i x_i$$

$$\frac{\partial \left(\sum w_i x_i \right)}{\partial x_1}$$

$$\boxed{\frac{\partial (\boxed{\vec{w}^T} \cancel{\vec{x}})}{\partial \vec{x}}}$$

$$\left[w_1 \; \cdots \; w_d \right] = \boxed{\vec{w}^T}$$

Vector Derivative Example

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

$$y_i = \sum_{j} a_{ij} x_j$$

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

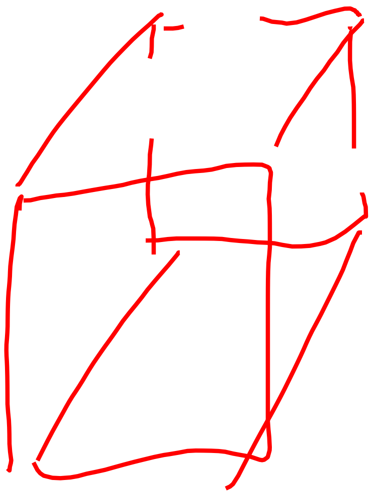
$$\boxed{\frac{\partial \vec{y}}{\partial \vec{x}}} = \boxed{A}$$

$$\begin{array}{cc} & j \\ i & \begin{bmatrix} & & \\ & \frac{\partial y_i}{\partial x_j} \; a_{ij} & \\ & & \end{bmatrix} \end{array}$$

Extension to Tensors

$X \in \mathbb{R}^{d_1 \times \dots \times d_m}$

$Y \in \mathbb{R}^{c_1 \times \dots \times c_n}$



$\text{y-vec} = Y(:)$

$\text{x-vec} = X(:)$

$$\frac{\partial Y[i_1 \dots i_n]}{\partial X[j_1 \dots j_m]}$$

$$\frac{\partial\, \text{y-vec}}{\partial\, \text{x-vec}} = \begin{bmatrix} \; \end{bmatrix}$$

Plan for Today

- (Finish) Analytical Gradients
- Automatic Differentiation
  - Computational Graphs
  - Forward mode vs Reverse mode AD
  - Patterns in backprop

Chain Rule: Composite Functions

$$\left[L(x) = f(g(x)) = (f \circ g)(x) \right]$$

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

$$L(w) = (g_l \circ g_{l-1} \cdots \circ g_1)(x)$$

$$\frac{\partial L}{\partial \vec{w}}$$

Chain Rule: Scalar Case

$$x \xrightarrow[g(\cdot)]{} z \xrightarrow[f(\cdot)]{} y \qquad x, y, z \in \mathbb{R}^1$$

$$= f(\underbrace{g(x)}_{z})$$

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

scalar prod.

Chain Rule: Vector Case

$$\vec{x} \in \mathbb{R}^d \xrightarrow[\underbrace{g(\cdot)}_{\mathbb{R}^d \to \mathbb{R}^m}]{} \vec{z} \in \mathbb{R}^m \xrightarrow[\underbrace{f(\cdot)}_{\mathbb{R}^m \to \mathbb{R}^C}]{} \vec{y} \in \mathbb{R}^C$$

$$\underbrace{\boxed{\frac{\partial \vec{y}}{\partial \vec{x}}}}_{J_{f \circ g}} = \underbrace{\boxed{\frac{\partial \vec{y}}{\partial \vec{z}}}}_{J_f} \cdot \underbrace{\boxed{\frac{\partial \vec{z}}{\partial \vec{x}}}}_{J_g}$$

Matrix Mult

Chain Rule: Jacobian view

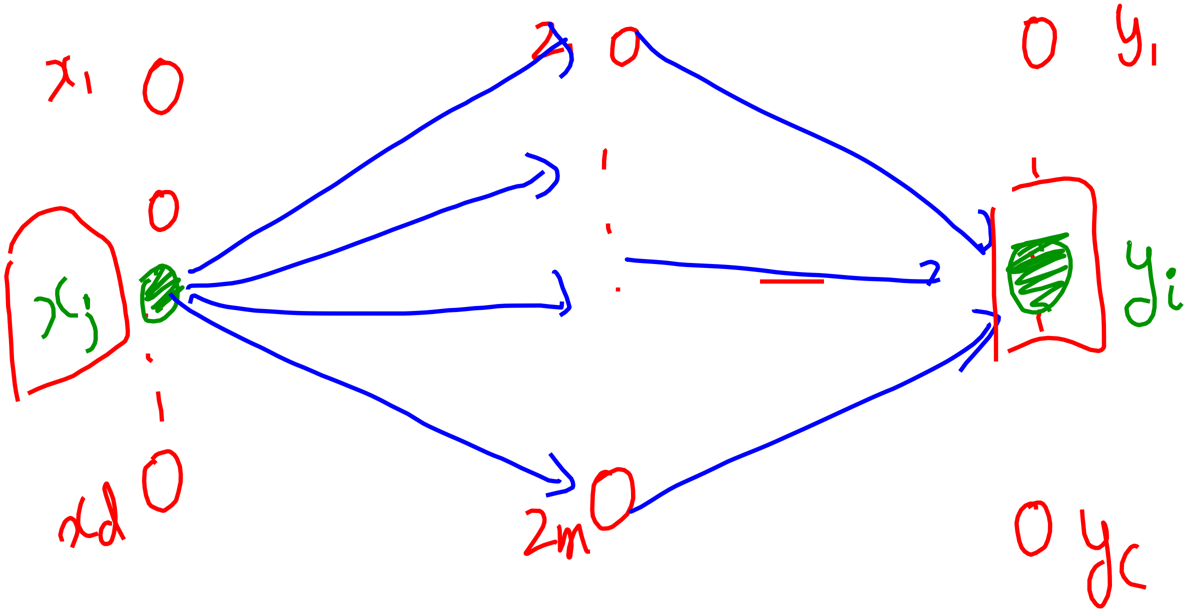
$$\underbrace{\frac{\partial \vec{y}}{\partial \vec{x}}}_{c \times d} = \underbrace{\frac{\partial \vec{y}}{\partial \vec{z}}}_{c \times m} \; \underbrace{\frac{\partial \vec{z}}{\partial \vec{x}}}_{m \times d}$$

$$\begin{array}{c c} & j \\ i & \left[\begin{array}{c} \boxed{\dfrac{\partial y_i}{\partial x_j}} \end{array} \right]_{c \times d} \end{array} = \begin{array}{c c} i & \left[\begin{array}{c} \dfrac{\partial y_i}{\partial z_k} \end{array} \right]_{c \times m} \end{array} \begin{array}{c} j \\ \left[\begin{array}{c} \dfrac{\partial z_k}{\partial x_j} \end{array} \right]_{m \times d} \end{array}$$

$$\boxed{\frac{\partial y_i}{\partial x_j}} = \sum_{k} \frac{\partial y_i}{\partial \boxed{z_k}} \; \frac{\partial \boxed{z_k}}{\partial x_j}$$

Chain Rule: Graphical view

$\vec{x} \in \mathbb{R}^d \xrightarrow{g(\cdot)} \vec{z} \in \mathbb{R}^m \xrightarrow{f(\cdot)} \vec{y} \in \mathbb{R}^c$



$$\frac{\partial y_i}{\partial x_j} = \sum_{\boxed{\text{paths}}} \underbrace{\frac{\partial y_i}{\partial z_k} \frac{\partial z_k}{\partial x_j}}_{}$$

k is on path

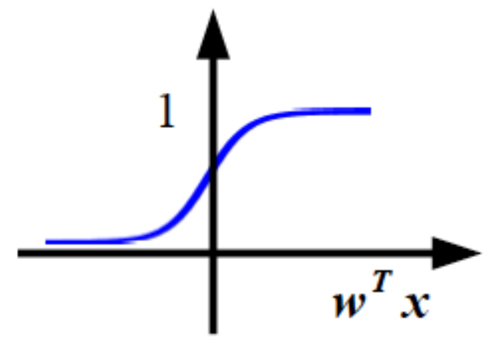
Linear Classifier: Logistic Regression

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

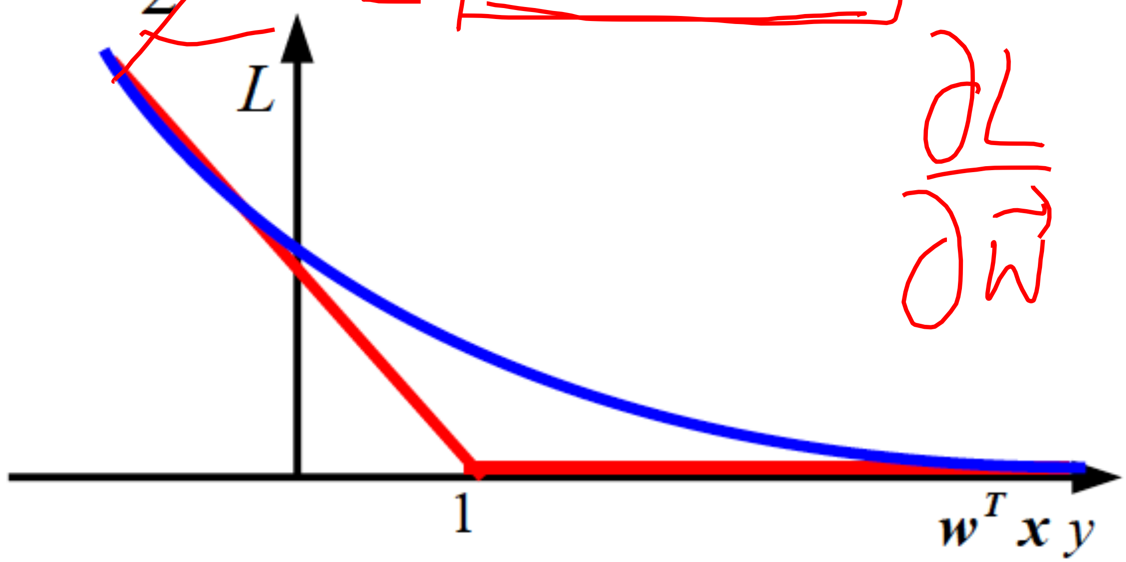
Binary label: $y \in \{-1, +1\}$

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

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



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



$\frac{\partial L}{\partial \vec{w}}$

Log Loss

Ranzato

Logistic Regression Derivatives

$$L_i = -\log\left(\frac{1}{1+e^{-w^T x}}\right) = \log\Big(\underbrace{1 + e^{\overbrace{-w^T x}^{v}}}_{p}\Big)$$

$$\frac{\partial L_i}{\partial \vec{w}} = \frac{\partial L_i}{\partial p} \cdot \frac{\partial p}{\partial v} \cdot \frac{\partial v}{\partial \vec{w}}$$

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

Logistic Regression Derivatives

Chain Rule: Cascaded

$$\vec{h}^{0} \in \mathbb{R}^d \xrightarrow{g_1()} \vec{h}^{(1)} \longrightarrow \vec{h}^{2} \cdots \longrightarrow \vec{h}^{(\ell)} \in \mathbb{R}^d$$

$$\xrightarrow{g_{(1)}} \quad \xrightarrow{g_{(2)}} \quad \cdots \longrightarrow \quad \xrightarrow{g_{\ell+1}} L \in \mathbb{R}^1$$

$$\frac{\partial L}{\partial \vec{h}^{\ell}} = [\quad]$$

$$\frac{\partial \vec{h}^{(\ell)}}{\partial \vec{h}^{(0)}} = \frac{\partial L}{\partial \vec{h}^{\ell}} \frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}} \frac{\partial \vec{h}^{\ell-1}}{\partial \vec{h}^{\ell-2}} \cdots \frac{\partial \vec{h}^{1}}{\partial \vec{h}^{0}}$$

d

$d \times d$

$O(\ell d^2)$

$O(\ell d^3)$

Chain Rule: How should we multiply?

Convolutional network (AlexNet)

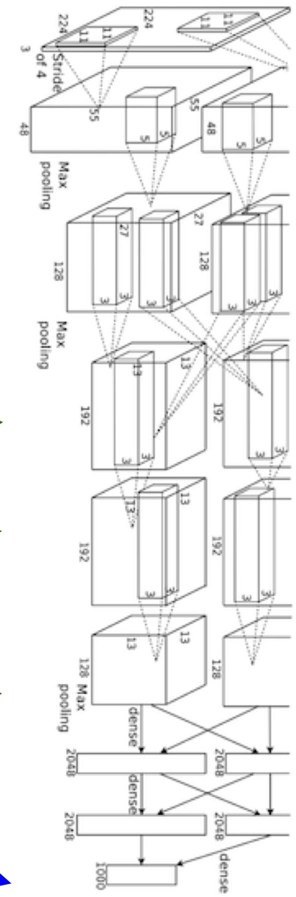
$\vec{x}$

input image

$\vec{w}$ weights

loss

$\frac{\partial L}{\partial \vec{w}}$



Neural Turing Machine

input image

loss

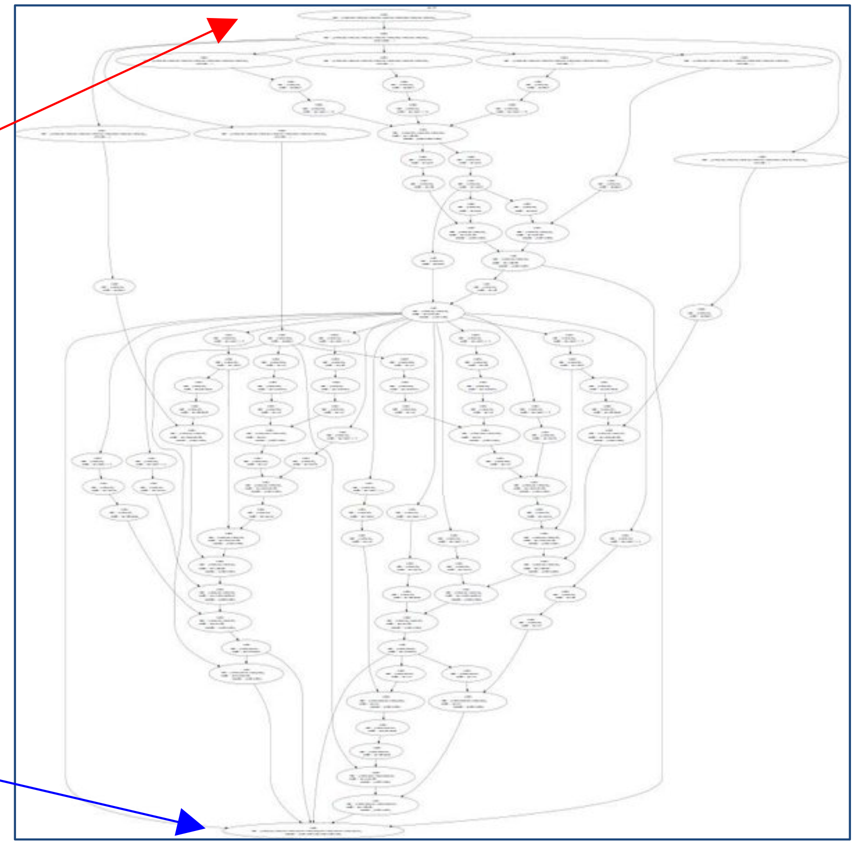


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

How do we compute gradients?

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

Plan for Today

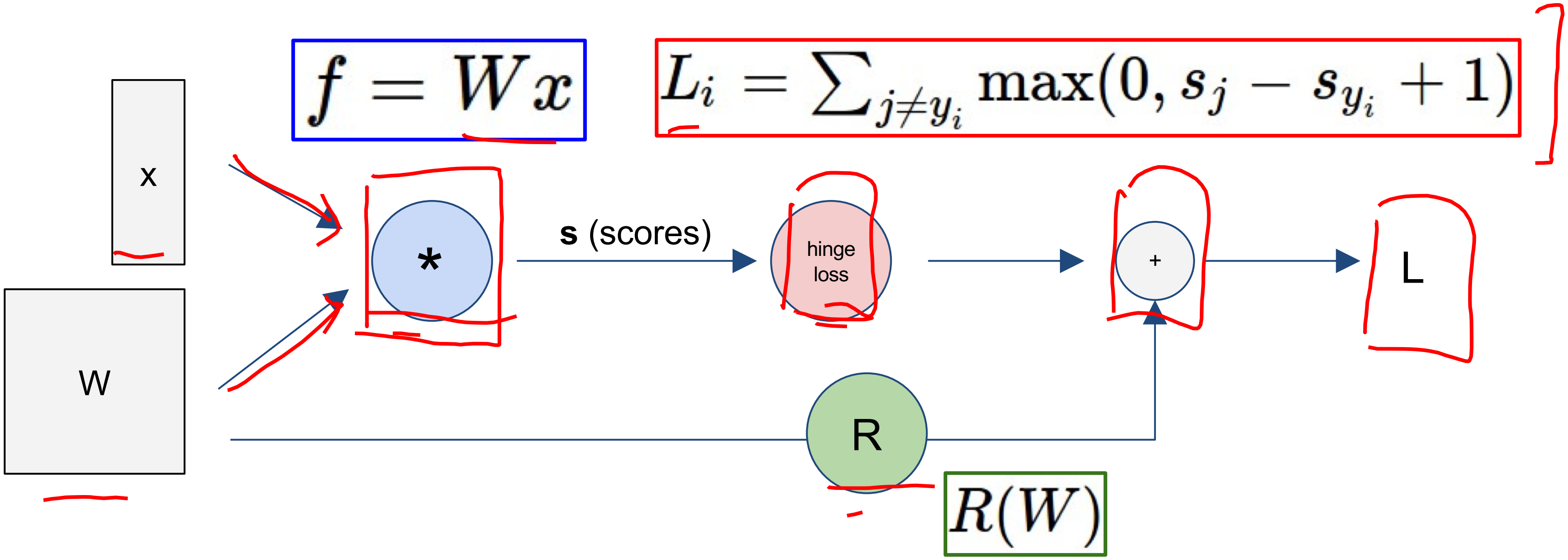
- (Finish) Analytical Gradients

- Automatic Differentiation
  - Computational Graphs
  - Forward mode vs Reverse mode AD

Deep Learning = Differentiable Programming

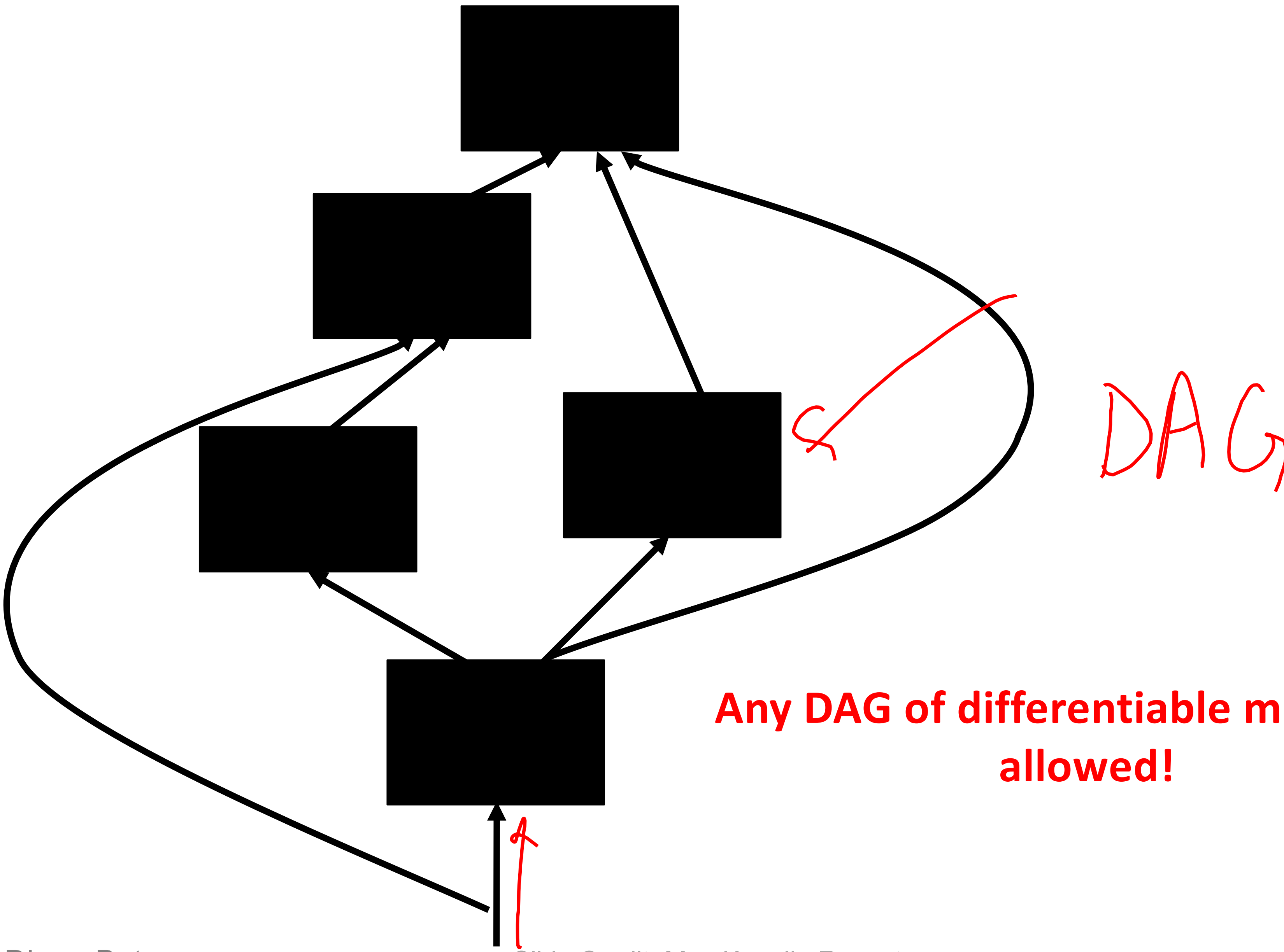
- Computation = Graph
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering
- Auto-Diff
  - A family of algorithms for implementing chain-rule on computation graphs

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)

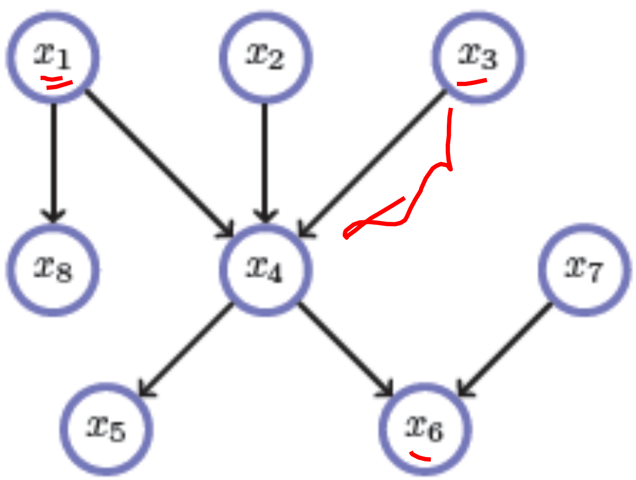
- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

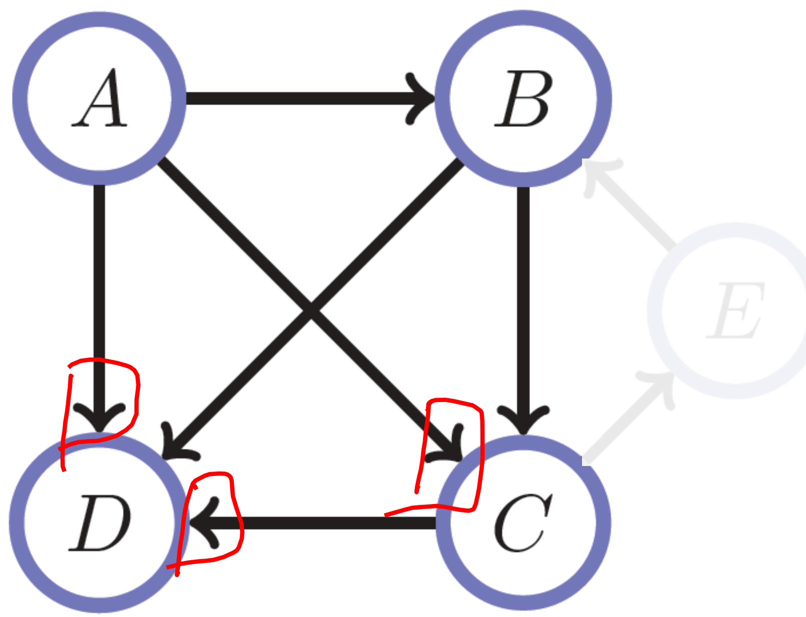
$G = (V, E)$

$E = \{ \underbrace{(v_i, v_j)} \mid v_i, v_j \in V \}$

Vertex

Vertices





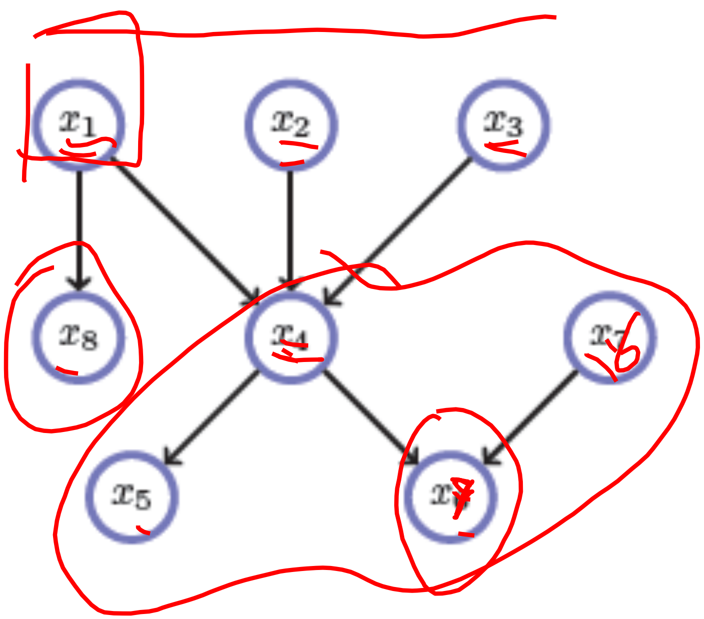
Directed Acyclic Graphs (DAGs)

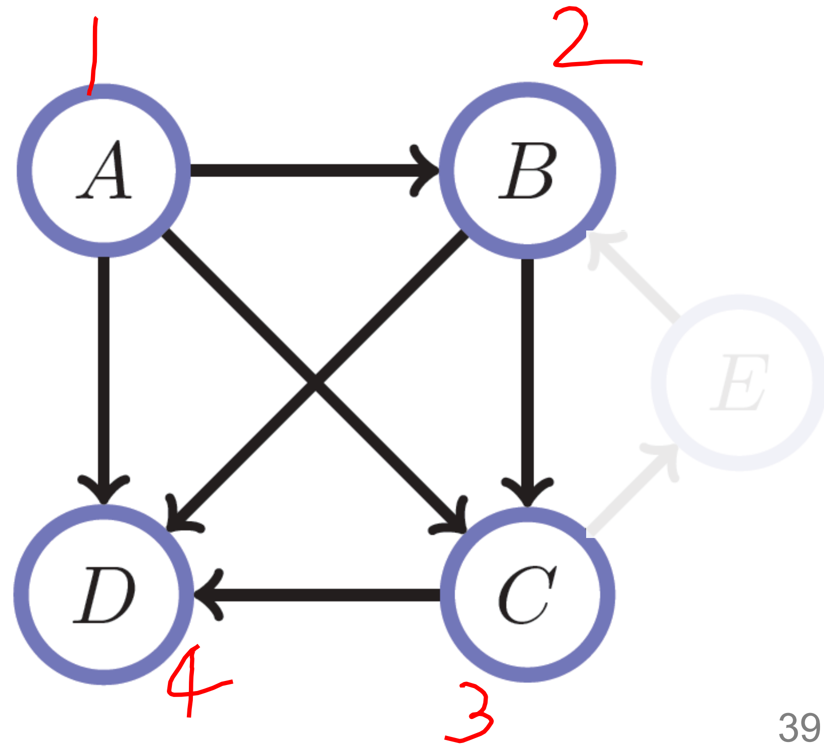
- Concept
  - Topological Ordering

$\exists$ bijection $\sigma : V \to \{1, \ldots, n\}$

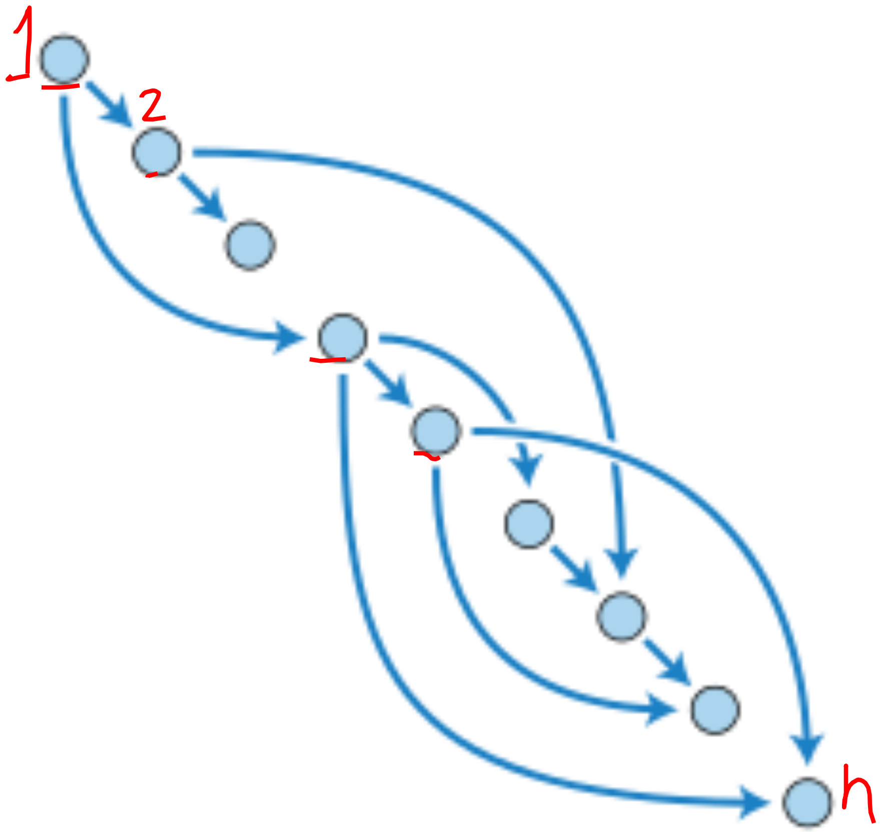
s.t. $\forall (v_i, v_j) \in E$

$\sigma(v_i) < \sigma(v_j)$





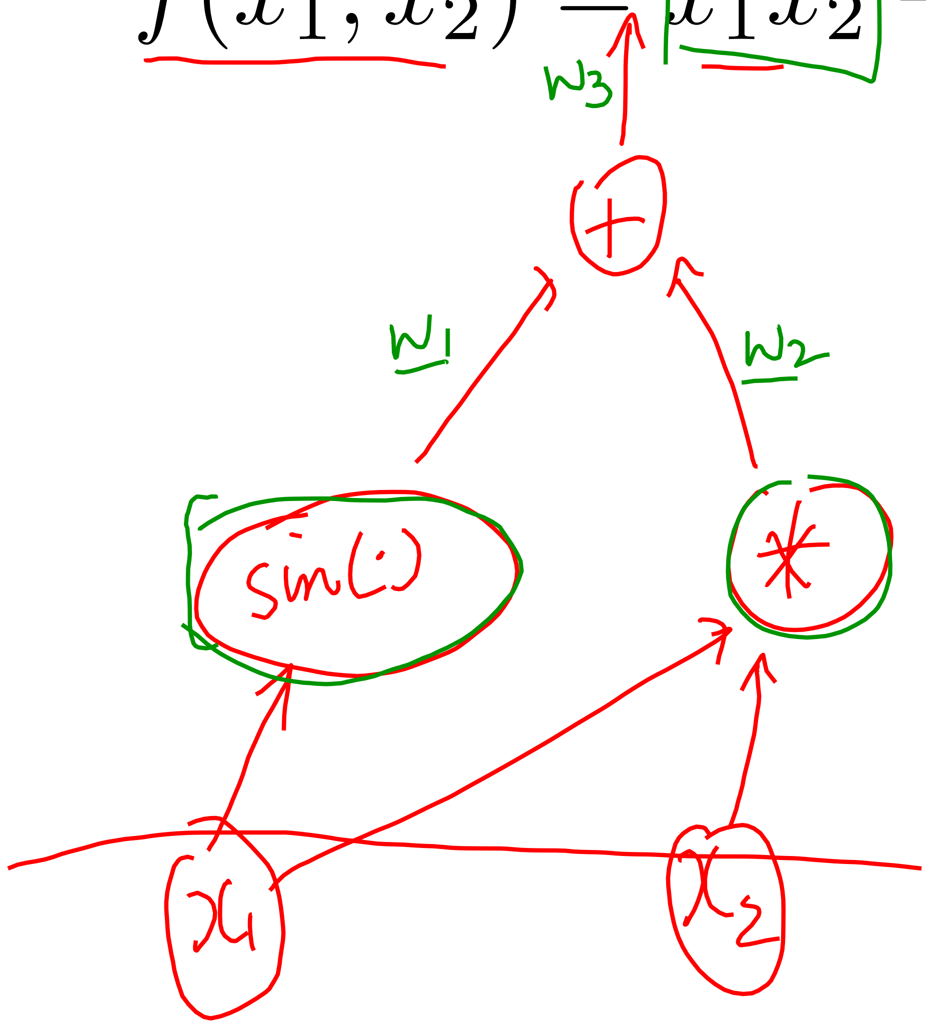
Directed Acyclic Graphs (DAGs)



Computational Graphs

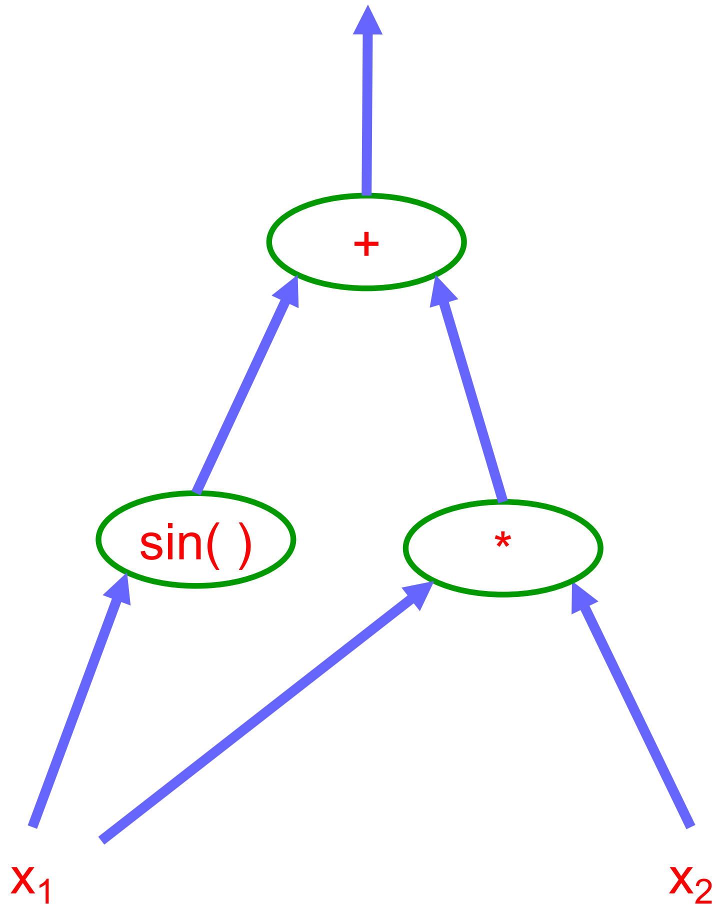
- Notation

$$f(x_1, x_2) = \overbrace{x_1 x_2}^{w_2} + \underbrace{\sin(x_1)}_{w_1}$$



Example

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

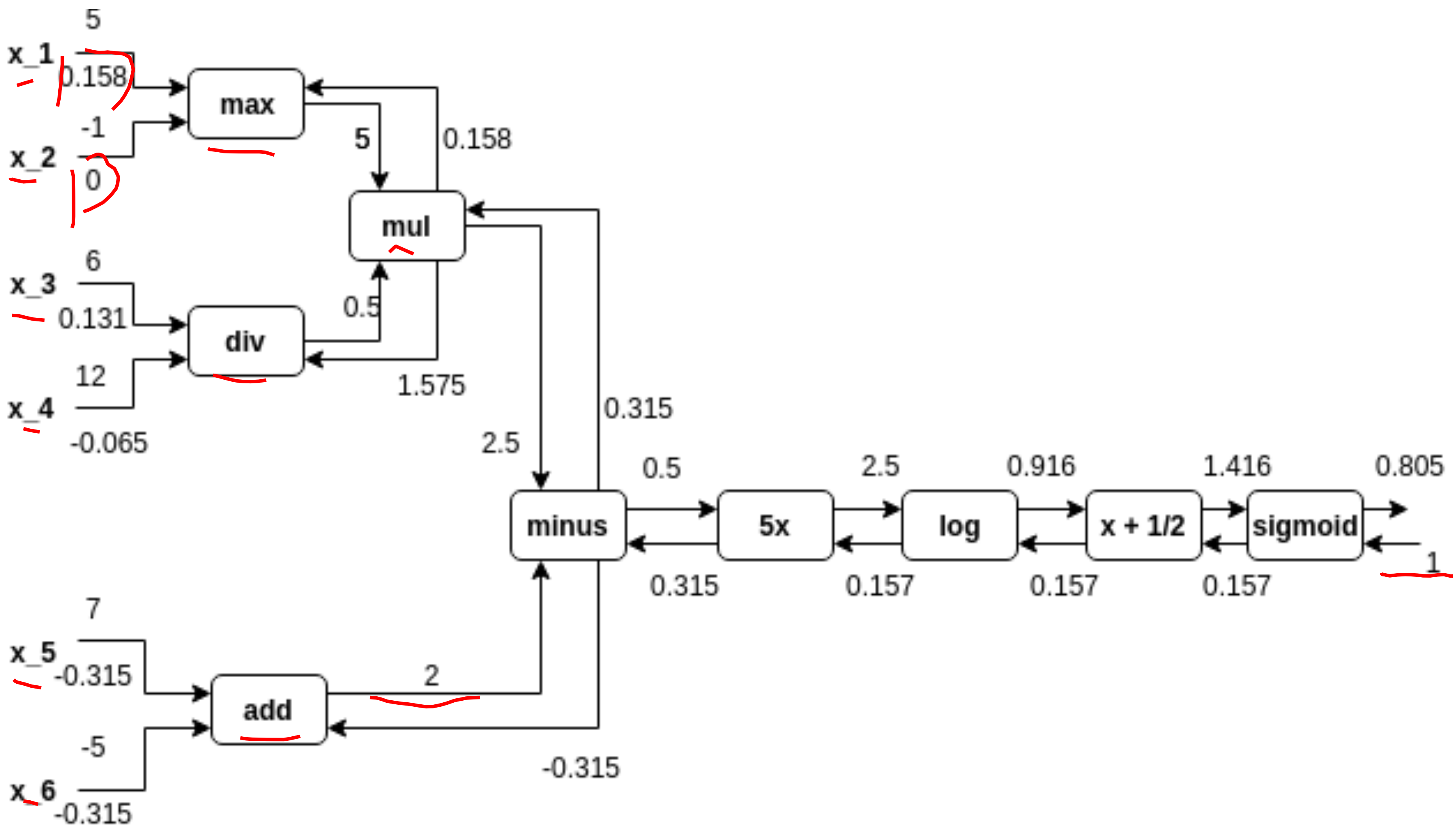


HW0

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

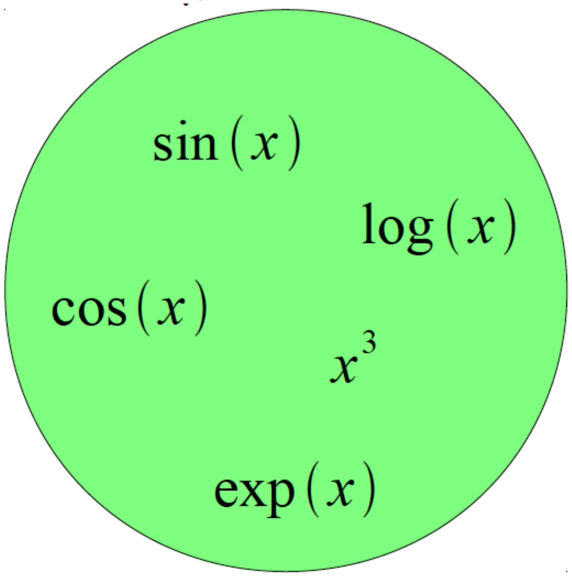
HW0 Submission by Samyak Datta

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



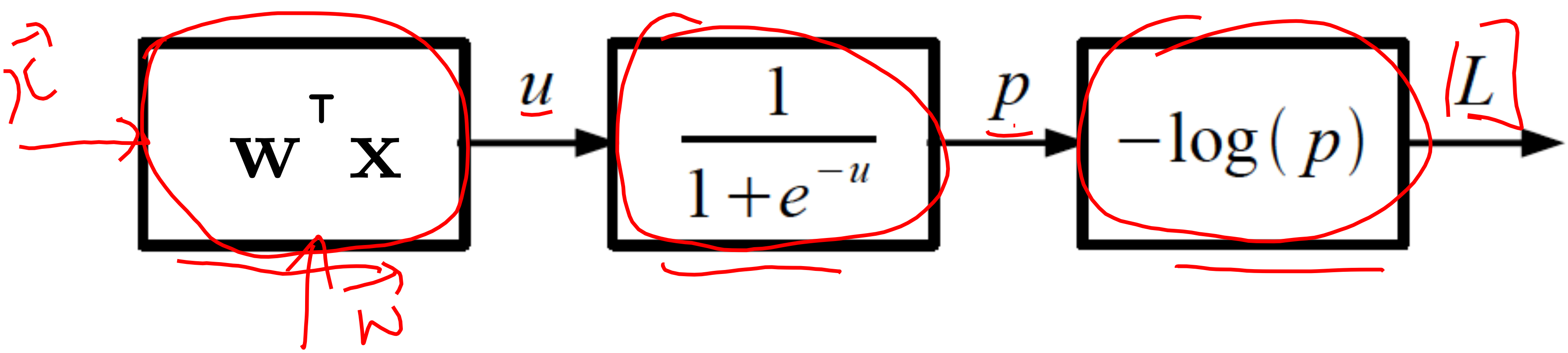
Logistic Regression as a Cascade

Given a library of simple functions



Compose into a complicate function

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



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

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

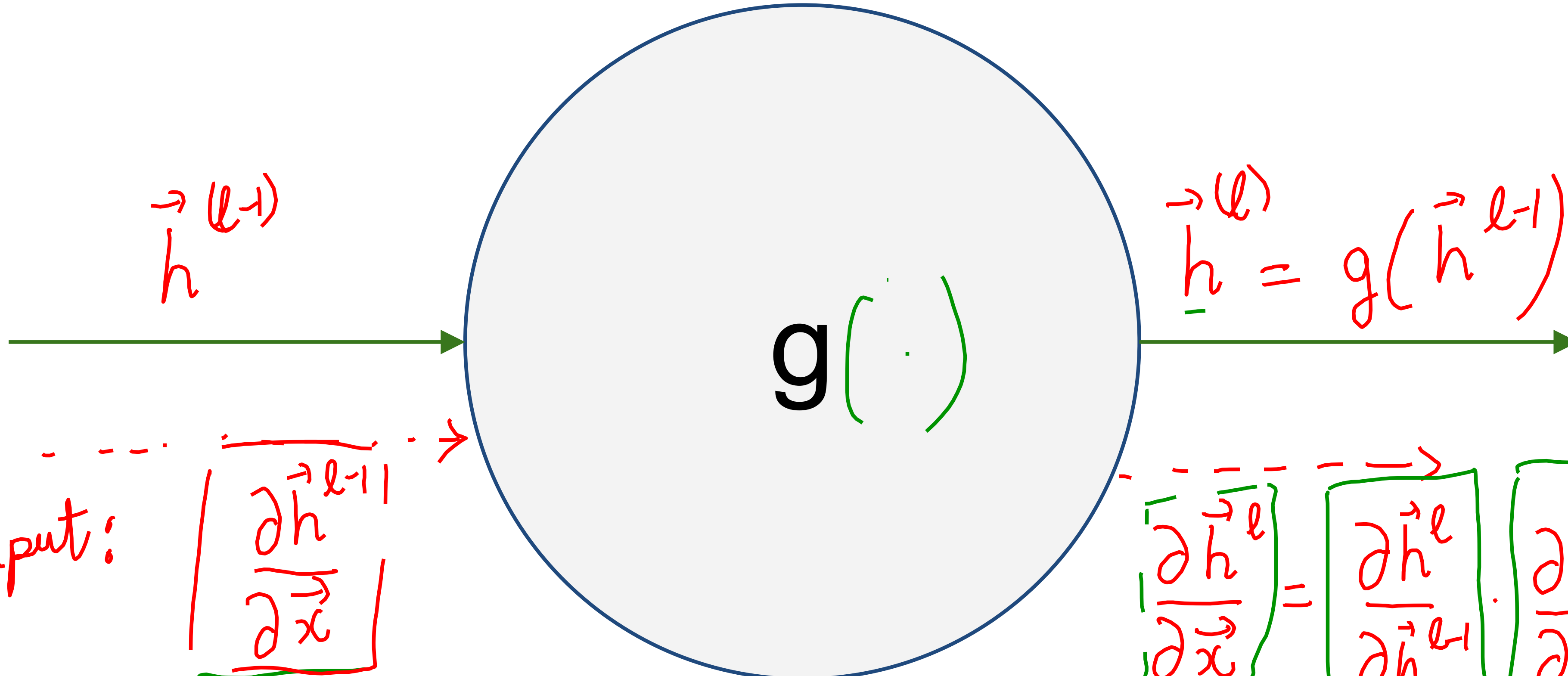
Forward mode vs Reverse Mode

- Key Computations

Forward mode AD

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

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

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

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

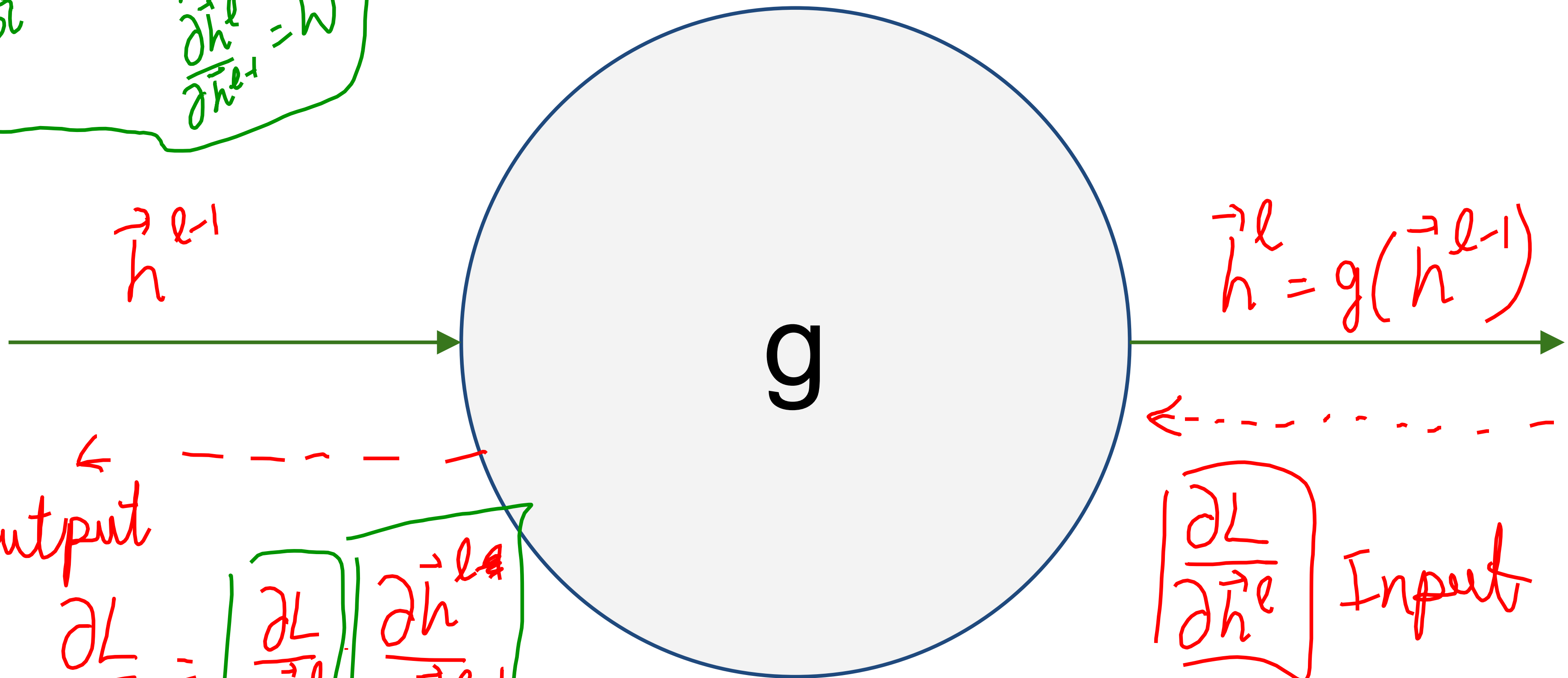
$\vec{h}^{\ell} = W\vec{h}^{\ell-1}$

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

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

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

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



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

Output

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

Input

Jacobian of g

Example: Forward mode AD

$$f(x_1, x_2) = \overbrace{x_1 x_2}^{w_2} + \overbrace{\sin(x_1)}^{w_1}$$

Goal: $\frac{\partial f}{\partial \vec{x}} = \left[\frac{\partial f}{\partial x_1} \;\; \frac{\partial f}{\partial x_2}\right]$

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

$\frac{\partial x_2}{\partial a}$

$w_1 = \sin(x_1)$

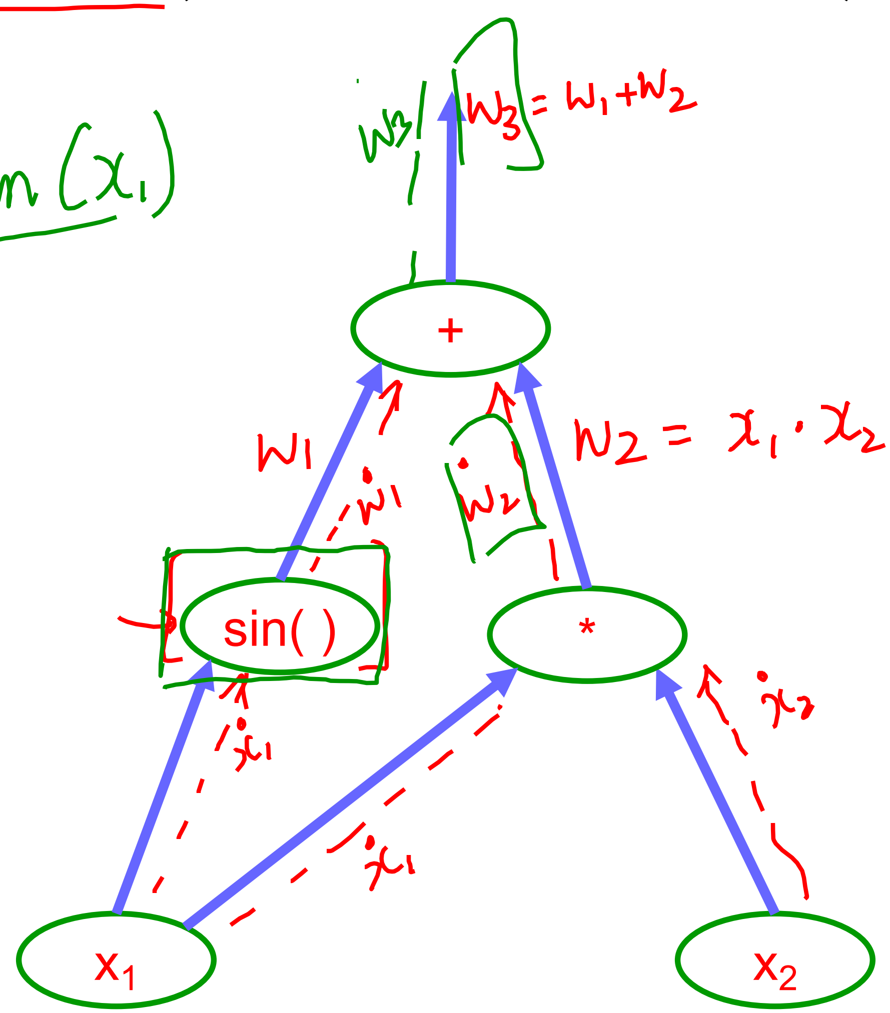
Input

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

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

$\frac{\partial w_2}{\partial a}$

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



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

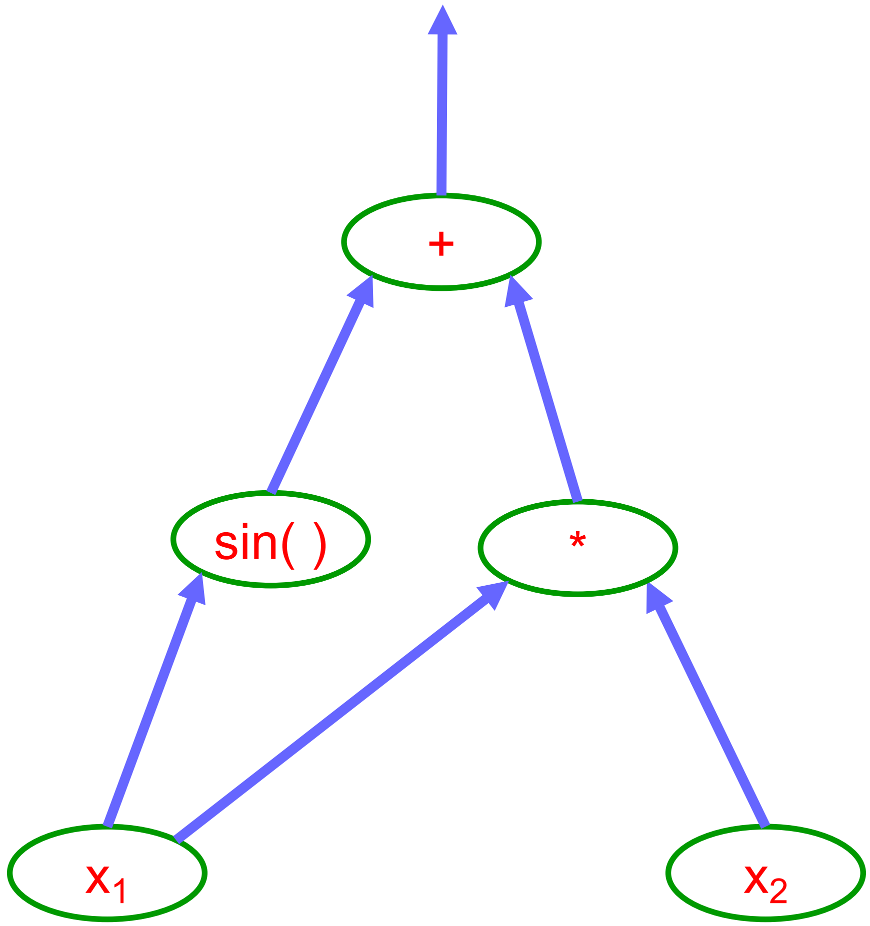
$\frac{\partial w_2}{\partial a} = x_1 \frac{\partial x_2}{\partial a} + x_2 \frac{\partial x_1}{\partial a}$

$= x_1 \dot{x}_2 + x_2 \dot{x}_1$

$a \in \{x_1, x_2\}$

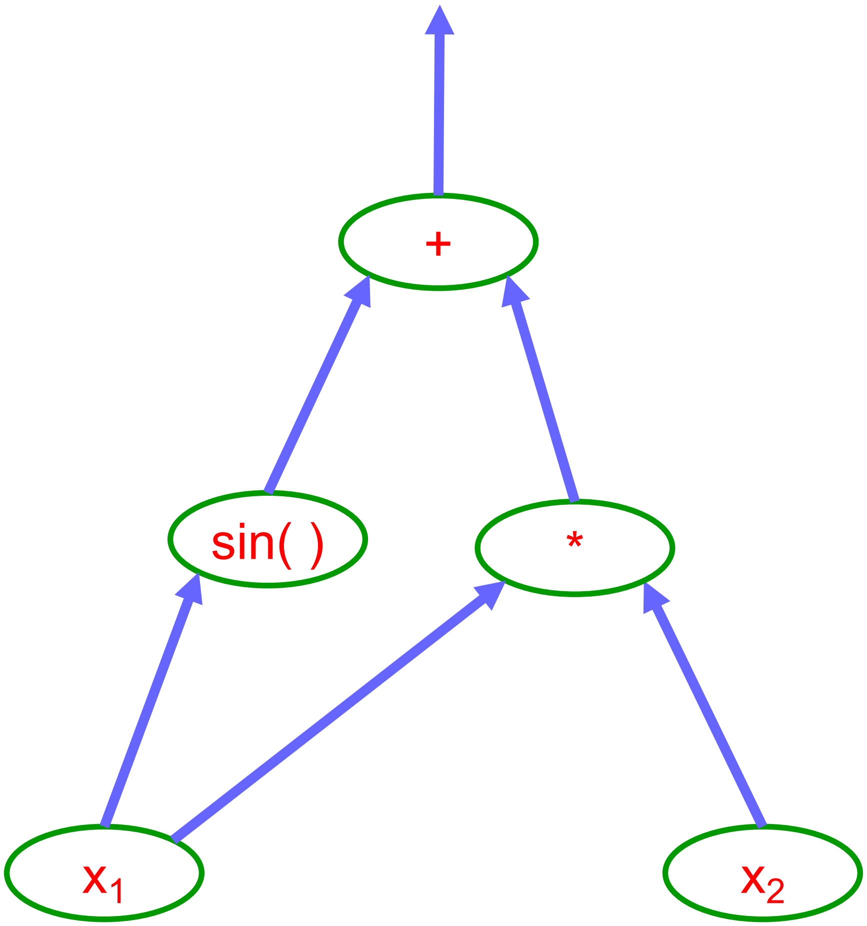
Example: Forward mode AD

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



Example: Forward mode AD

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



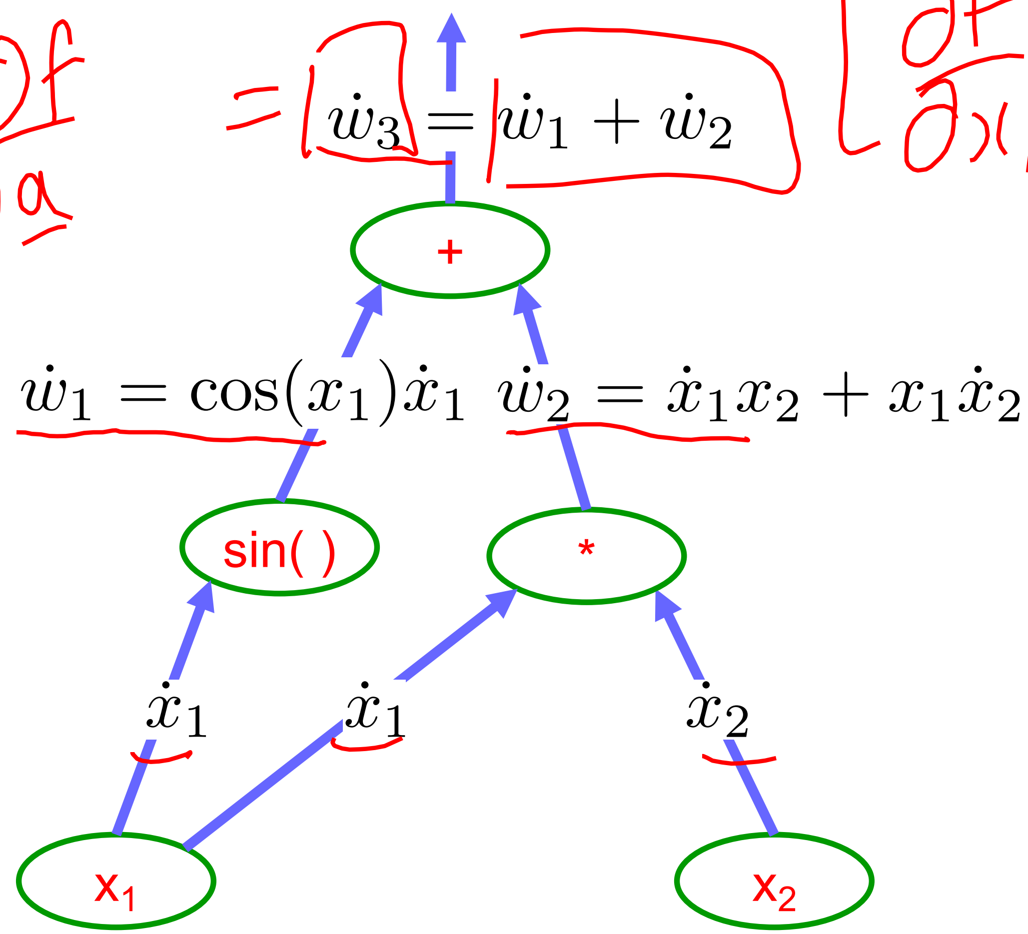
Example: Forward mode AD

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

$a = x_1$

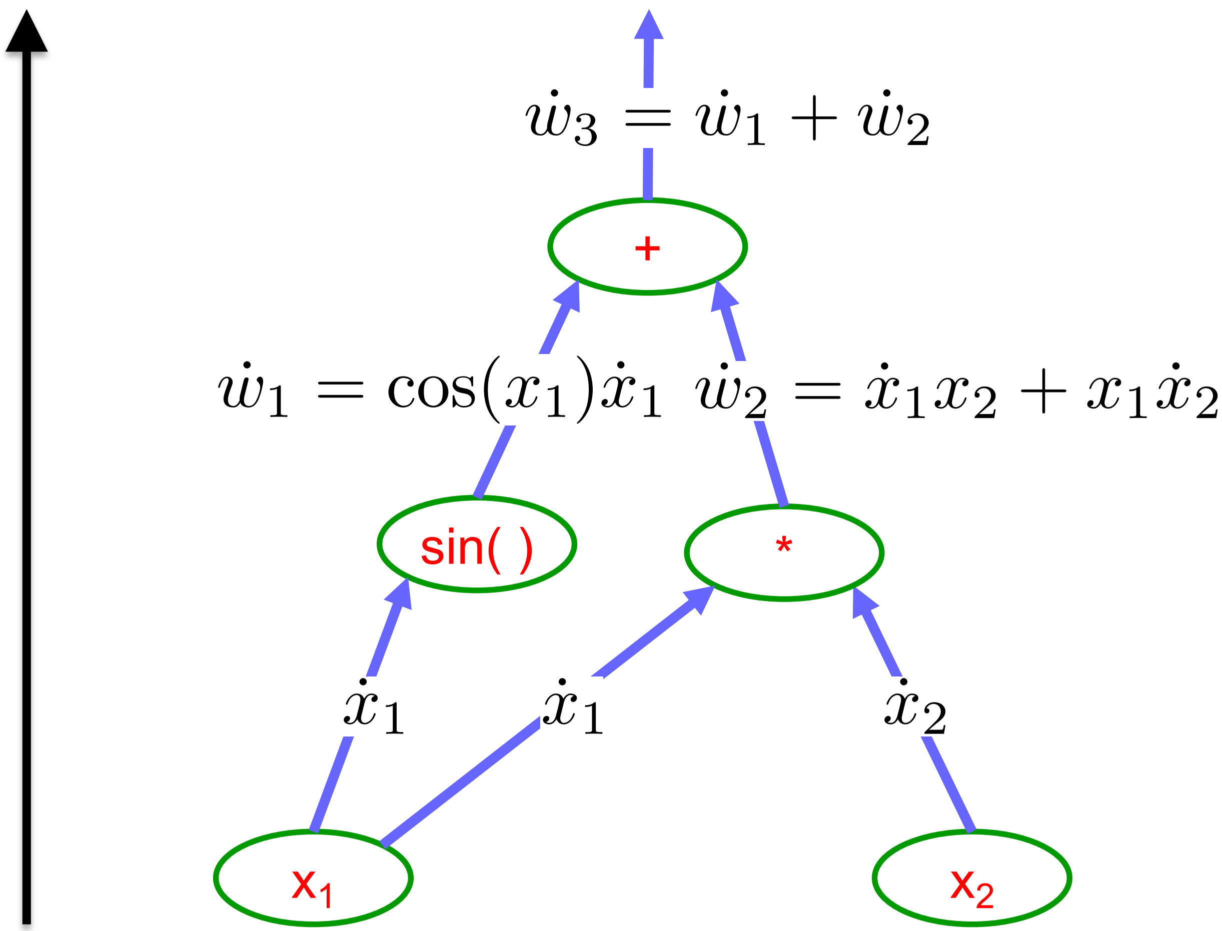
$a = x_2$

$\frac{\partial f}{\partial a} = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \right]$



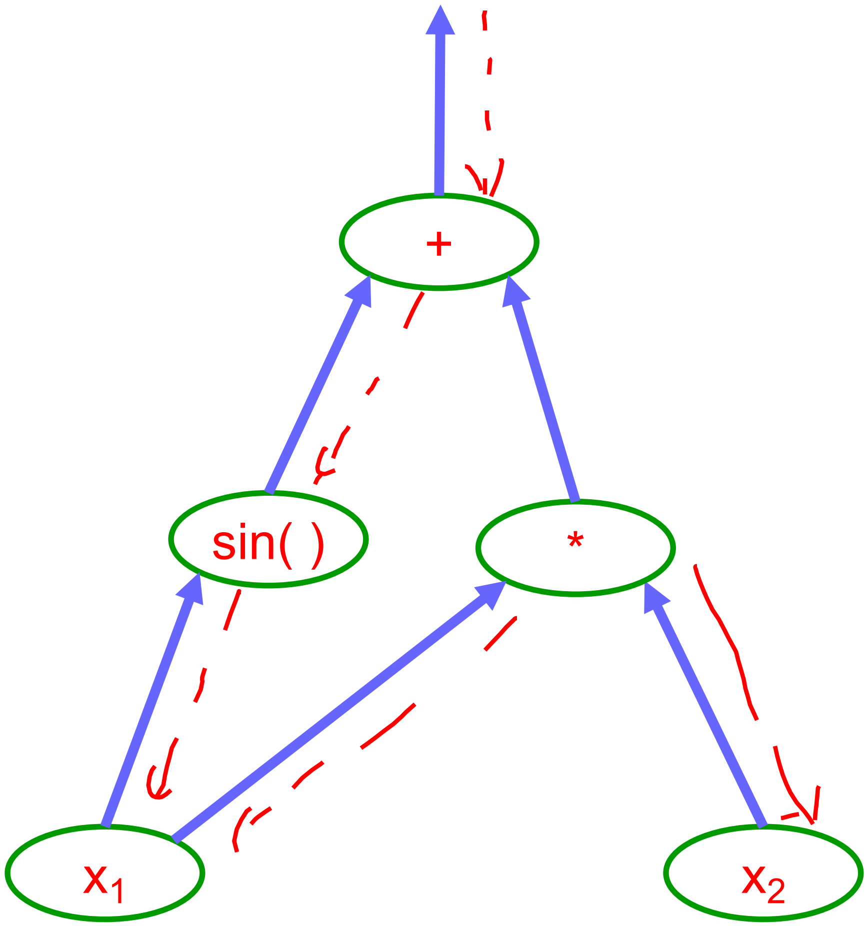
Example: Forward mode AD

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



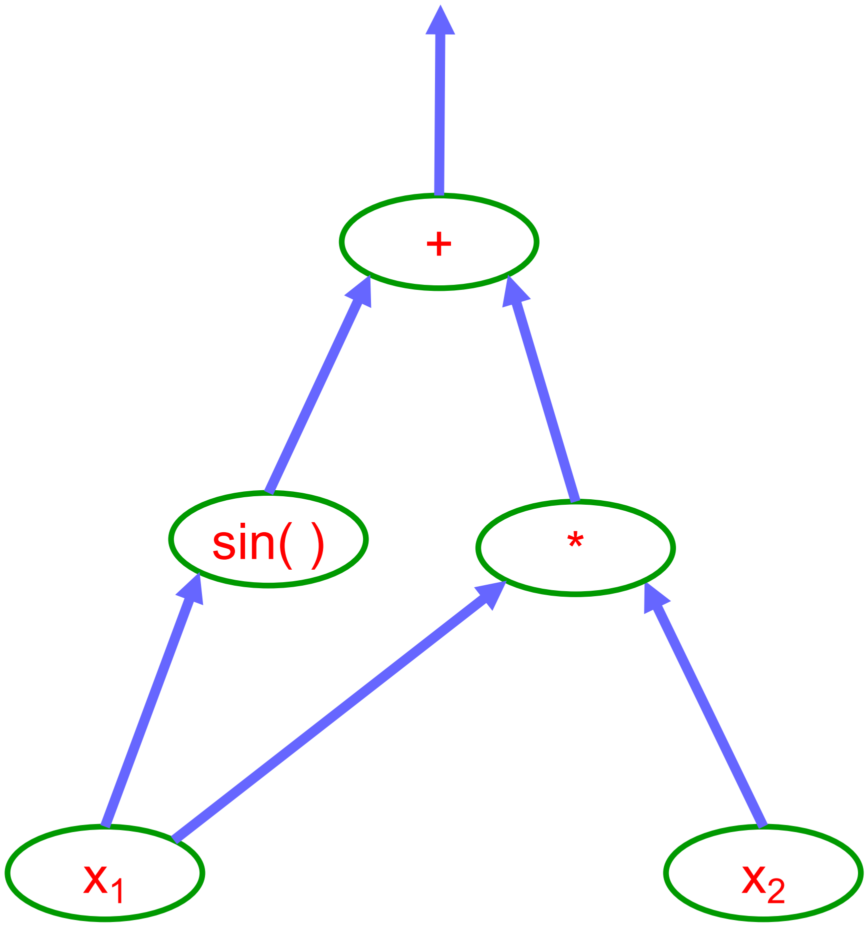
Example: Reverse mode AD

$$f(x_1, x_2) = \overbrace{x_1 x_2}^{w_2} + \overbrace{\sin(x_1)}^{w_1}$$



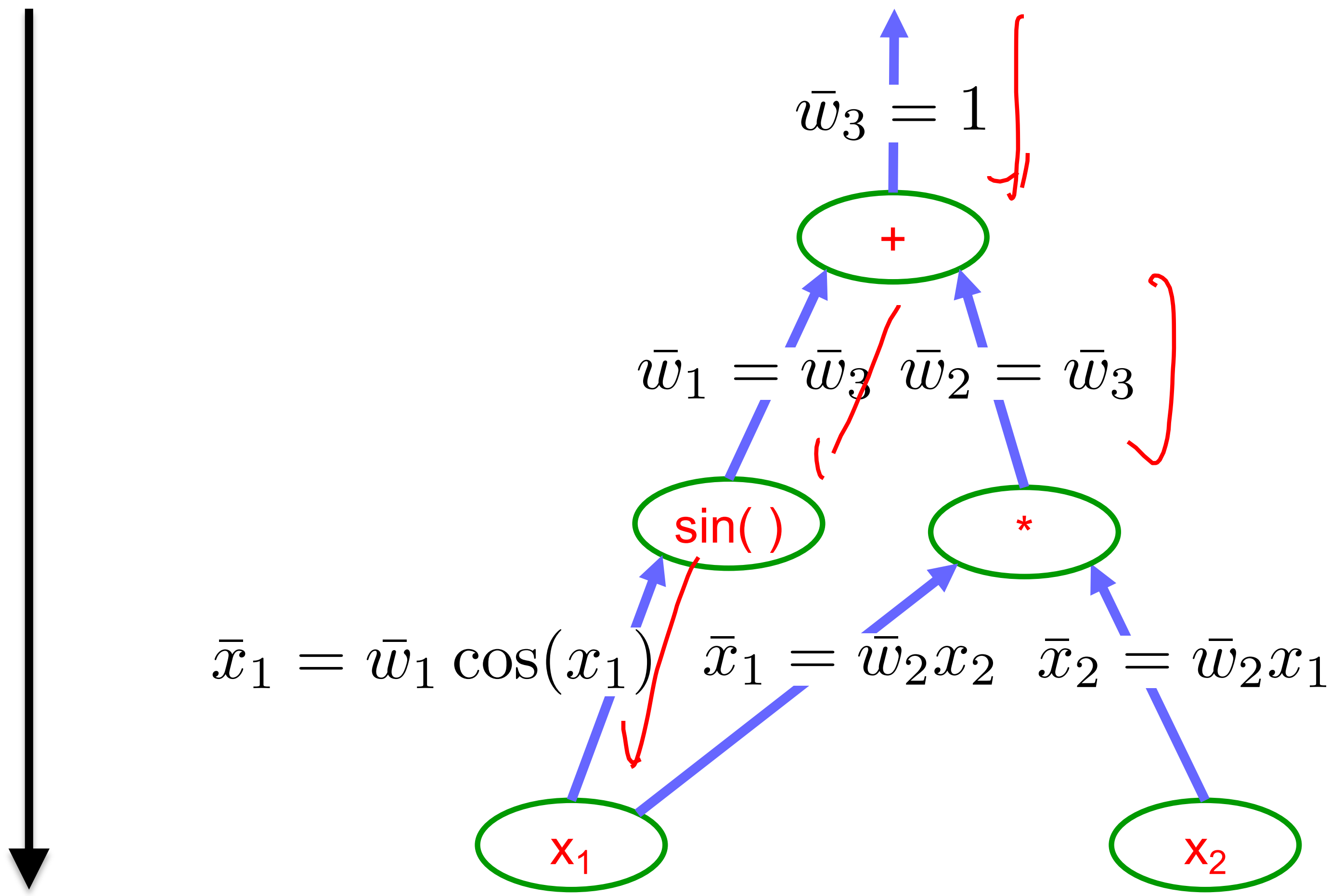
Example: Reverse mode AD

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



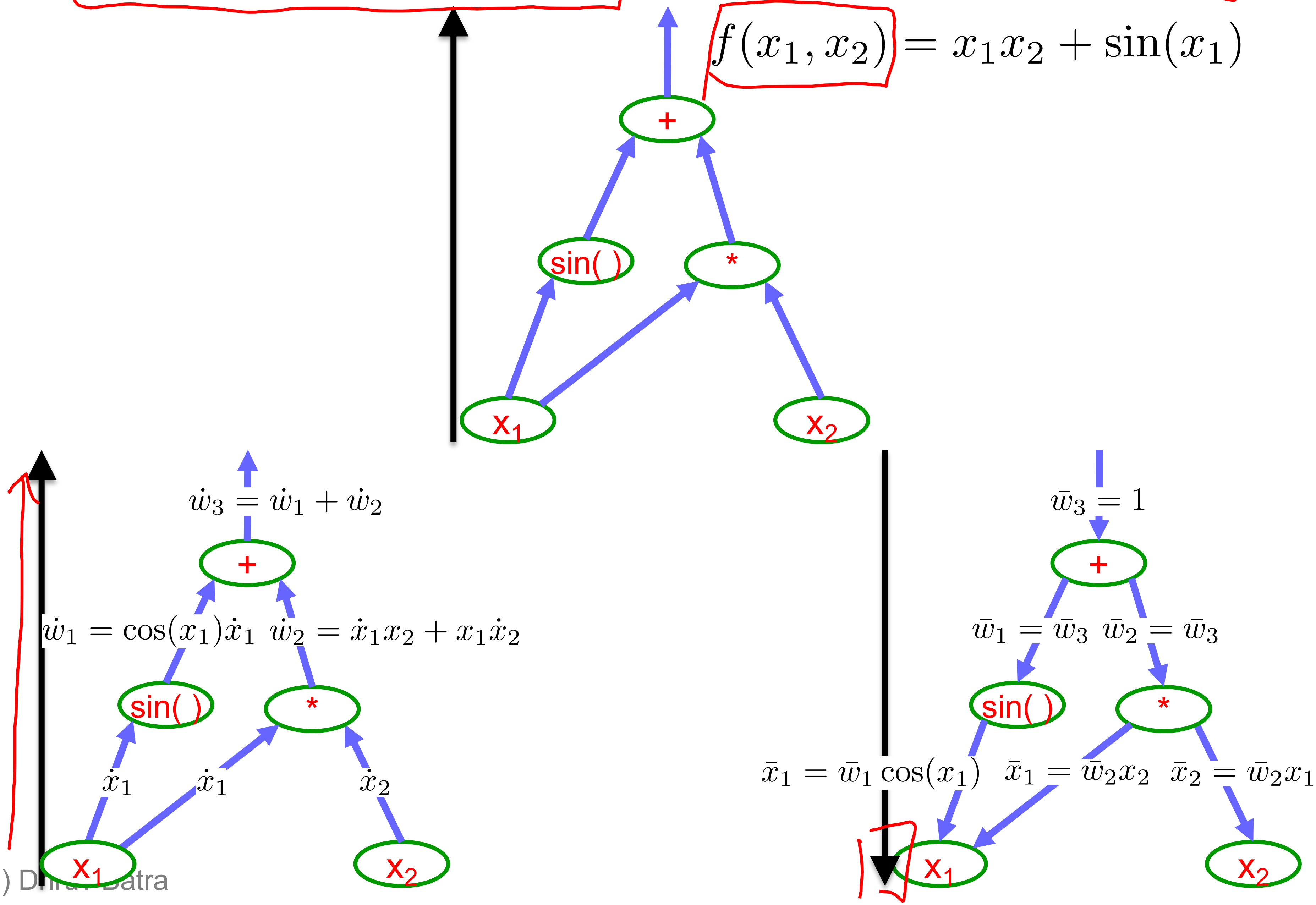
Example: Reverse mode AD

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



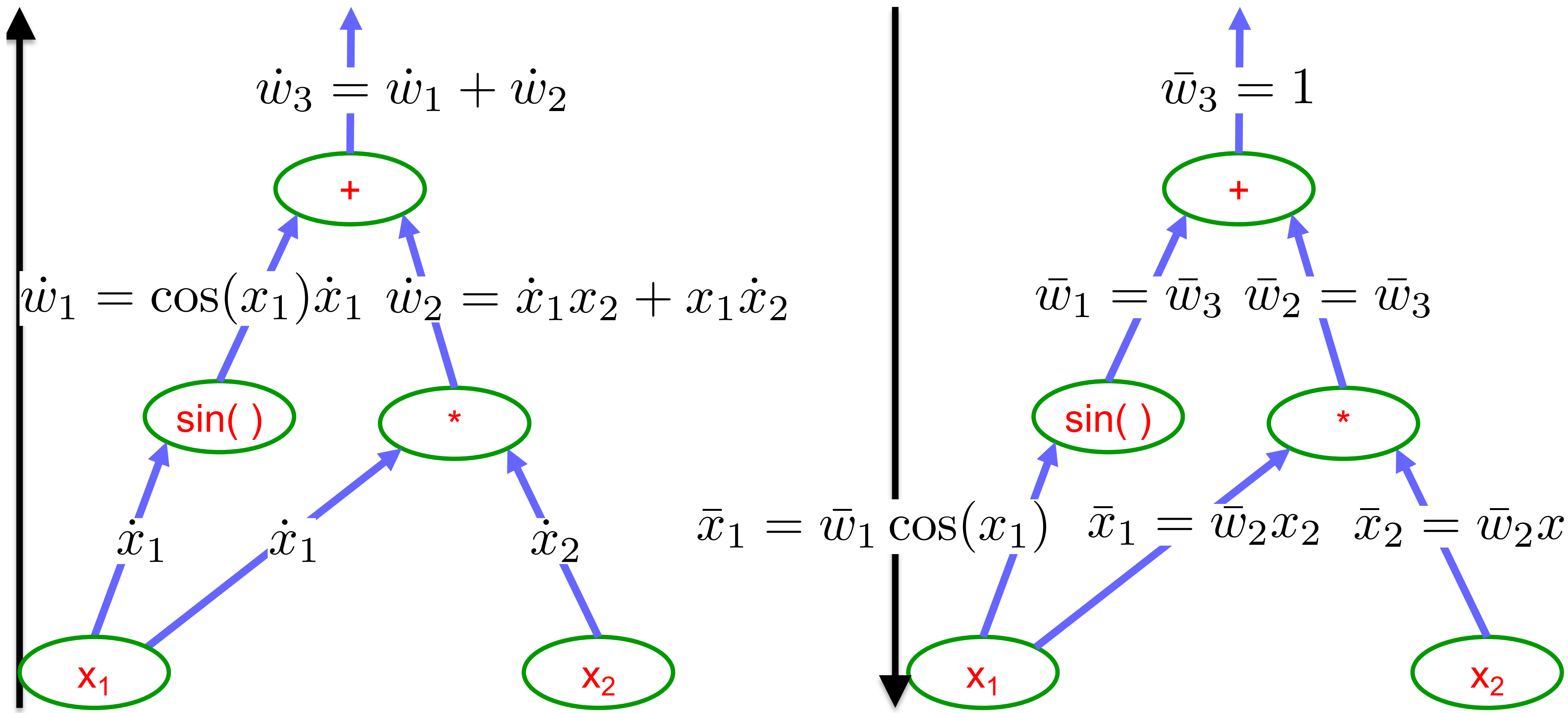
Forward Pass vs
Forward mode AD vs Reverse Mode AD

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



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?