



# CS 7643: Deep Learning

## Topics:

- Computational Graphs
  - Notation + example
- Computing Gradients
  - Forward mode vs Reverse mode AD

Dhruv Batra  
Georgia Tech

# Administrativa

- No class on Tuesday next week (Sep 12)
- HW0 solutions posted

# Invited Talk #1

- Note: Change in time: 12:30-1:30pm (Lunch at Noon)

---

**ML@GT Seminar Series**

**Wednesday Sep 6 2017, 12:30 pm - 1:30 pm**

**Location: Marcus Nanotechnology Building, Room 1118**

---



**Speaker: Soumith Chintala**

---

## **An Overview of Deep Learning Frameworks and an Introduction to PyTorch**

---

**Abstract:** In this talk, you will get an exposure to the various types of deep learning frameworks – declarative and imperative frameworks such as TensorFlow and PyTorch. After a broad overview of frameworks, you will be introduced to the PyTorch framework in more detail. We will discuss your perspective as a researcher and a user, formalizing the needs of research workflows (covering data pre-processing and loading, model building, etc.). Then, we shall see how the different features of PyTorch map to helping you with these workflows.

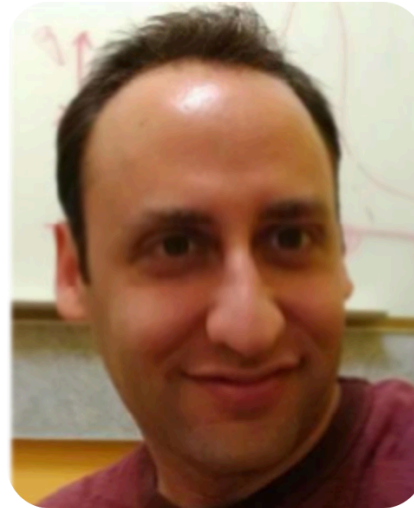
**Bio:** Soumith Chintala is a Researcher at Facebook AI Research, where he works on deep learning, reinforcement learning, generative image models, agents for video games and large-scale high-performance deep learning. With over 500 commits, Soumith is also one of the primary developers of the popular open-source PyTorch framework for deep learning. Prior to joining Facebook in August 2014, he worked at MuseAmi, where he built deep learning models for music and vision targeted at mobile devices. He holds a Masters in CS from NYU, and spent time in Yann LeCun's NYU lab building deep learning models for pedestrian detection, natural image OCR, and depth-images among others.

# Invited Talk #2

---

**Special Guest Speaker - CS 7643 Deep Learning**  
**Thursday Sep 7 2017, 4:30 pm - 5:45 pm**  
**Location: Clough 144**

---



**Speaker: Nathan Silberman**

---

## **TF-Slim: A Lightweight Library for Defining, Training and Evaluating Complex Models in TensorFlow**

---

**Abstract:** TF-Slim is a TensorFlow-based library with various components. These include modules for easily defining neural network models with few lines of code, routines for training and evaluating such models in a highly distributed fashion and utilities for creating efficient data loading pipelines. Additionally, the TF-Slim Image Models library provides many commonly used networks (ResNet, Inception, VGG, etc) that make replicating results and creating new networks using existing components simple and straightforward. I will discuss some of the design choices and constraints that guided our development process as well as several high-impact projects in the medical domain that utilize most or all components of the TF-Slim library.

---

**Bio:** Nathan Silberman is the Lead Deep Learning Scientist at 4Catalyzer where he works on a variety of healthcare related projects. His machine learning interests include semantic segmentation, detection and reinforcement learning and how to best apply these areas to high-impact areas in the medical world. Prior to joining 4Catalyzer, Nathan was a researcher at Google where among various projects, he co-wrote TensorFlow-Slim, which is now a major component of the TensorFlow library. Nathan received his PhD in 2015 from New York University under Rob Fergus and David Sontag

# Project

- Goal
  - Chance to try Deep Learning
  - **Combine with other classes / research / credits / anything**
    - You have our blanket permission
    - Extra credit for shooting for a publication
  - Encouraged to apply to your research (computer vision, NLP, robotics,...)
  - Must be done this semester.
- Main categories
  - **Application/Survey**
    - Compare a bunch 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

- Deliverables:
  - No formal proposal document due
    - Consider talking to your TAs
  - Final Poster Session
  - (tentative): Week of Nov 27.
- Questions/support/ideas
  - Stop by and talk to TAs
- Teaming
  - Encouraged to form teams of 2-3.

# TAs



Michael Cogswell

3<sup>rd</sup> year CS PhD student

<http://mcogswell.io/>



Abhishek Das

2<sup>nd</sup> year CS PhD student

<http://abhishekdas.com/>

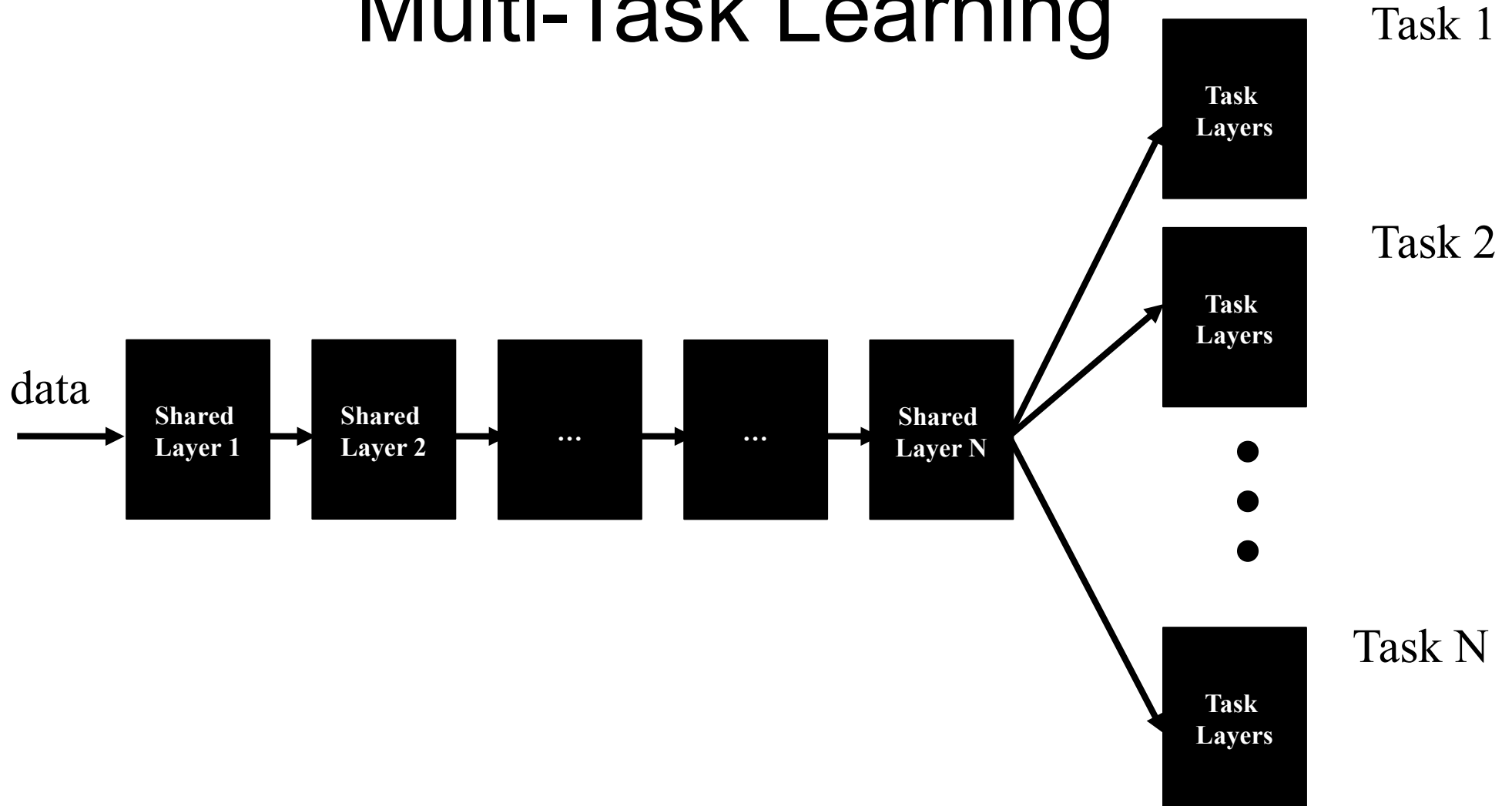


Zhaoyang Lv

3<sup>rd</sup> year CS PhD student

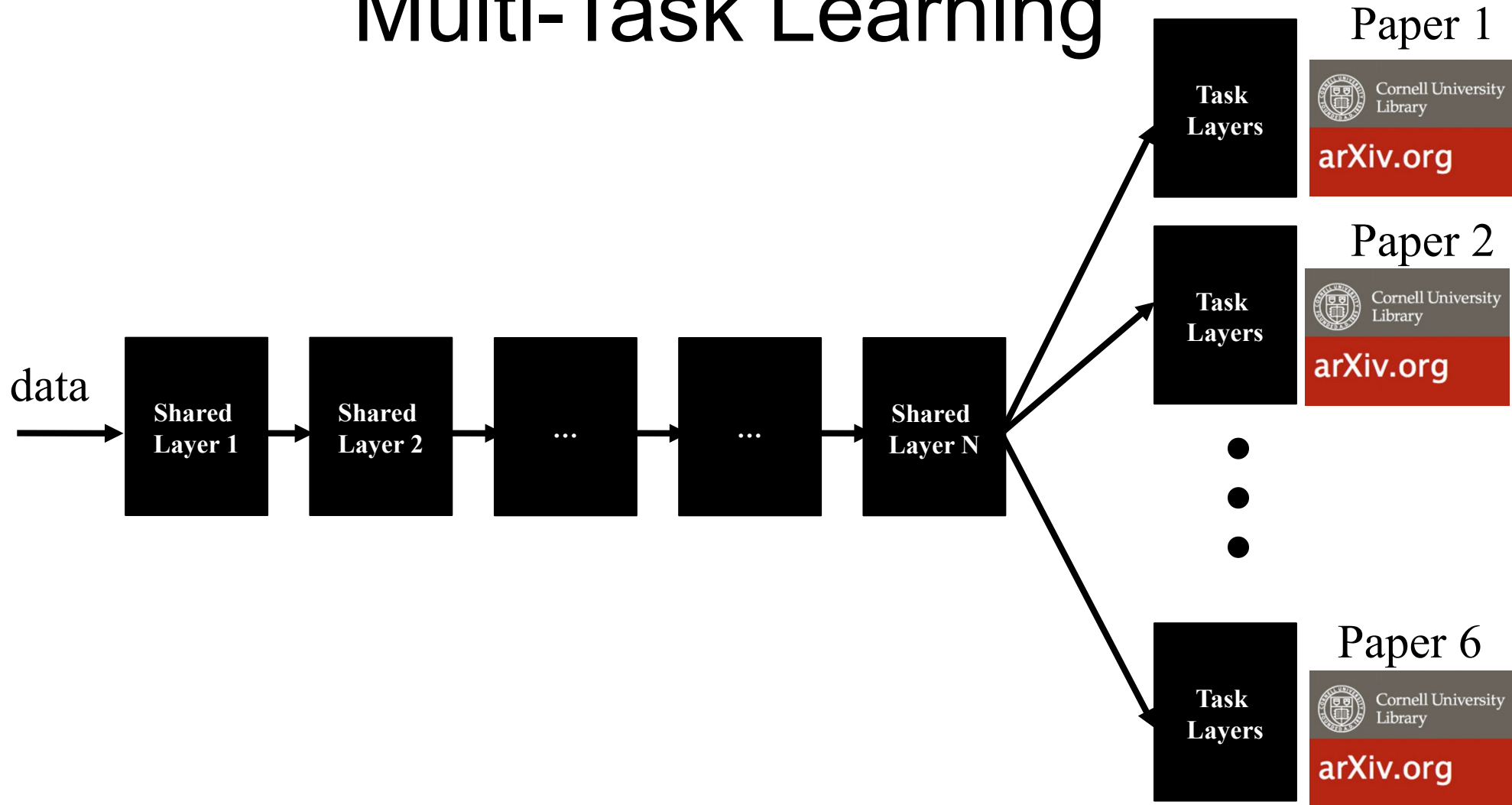
<https://www.cc.gatech.edu/~zlv30>

# Paper Reading Intuition: Multi-Task Learning

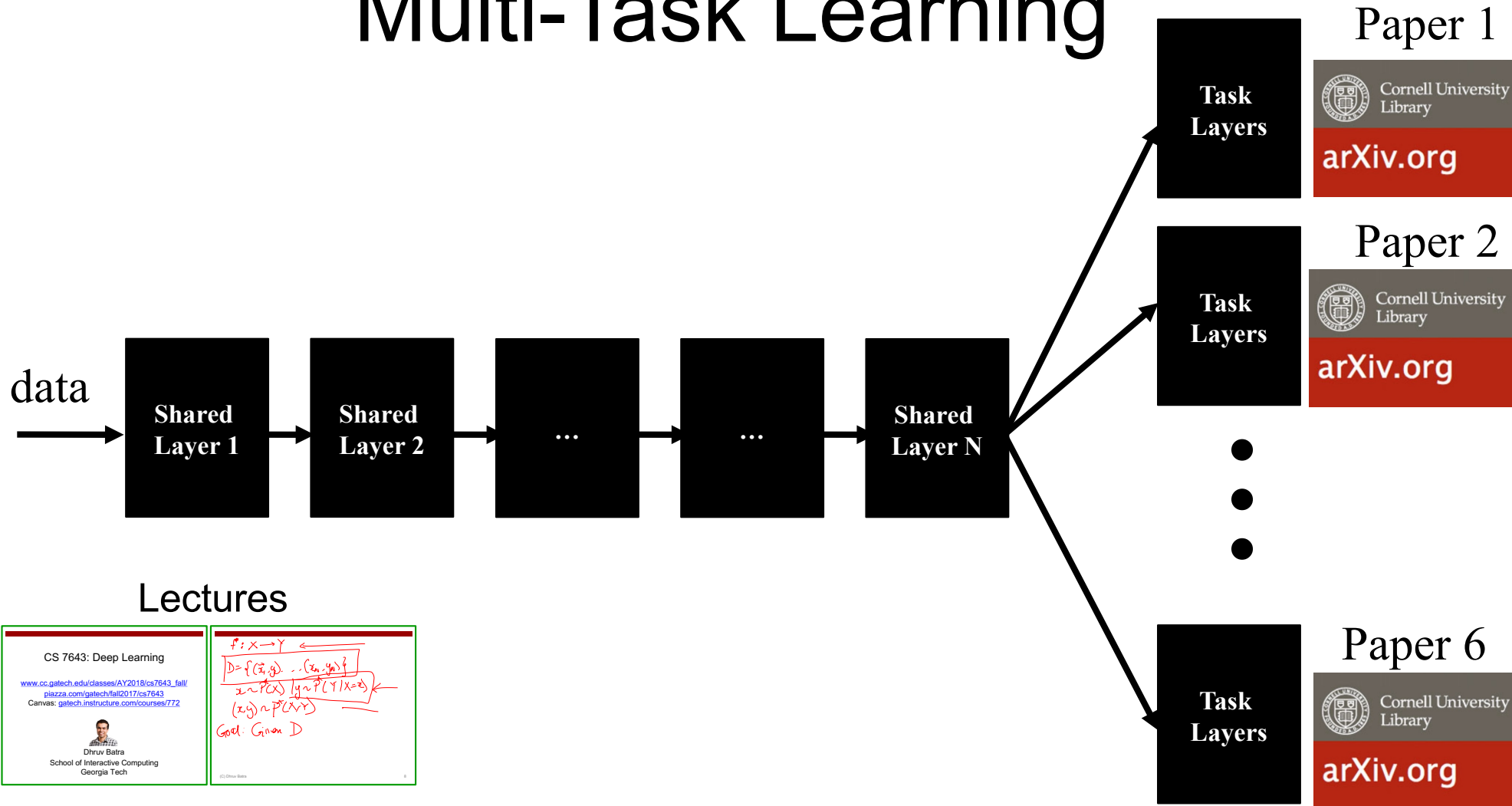




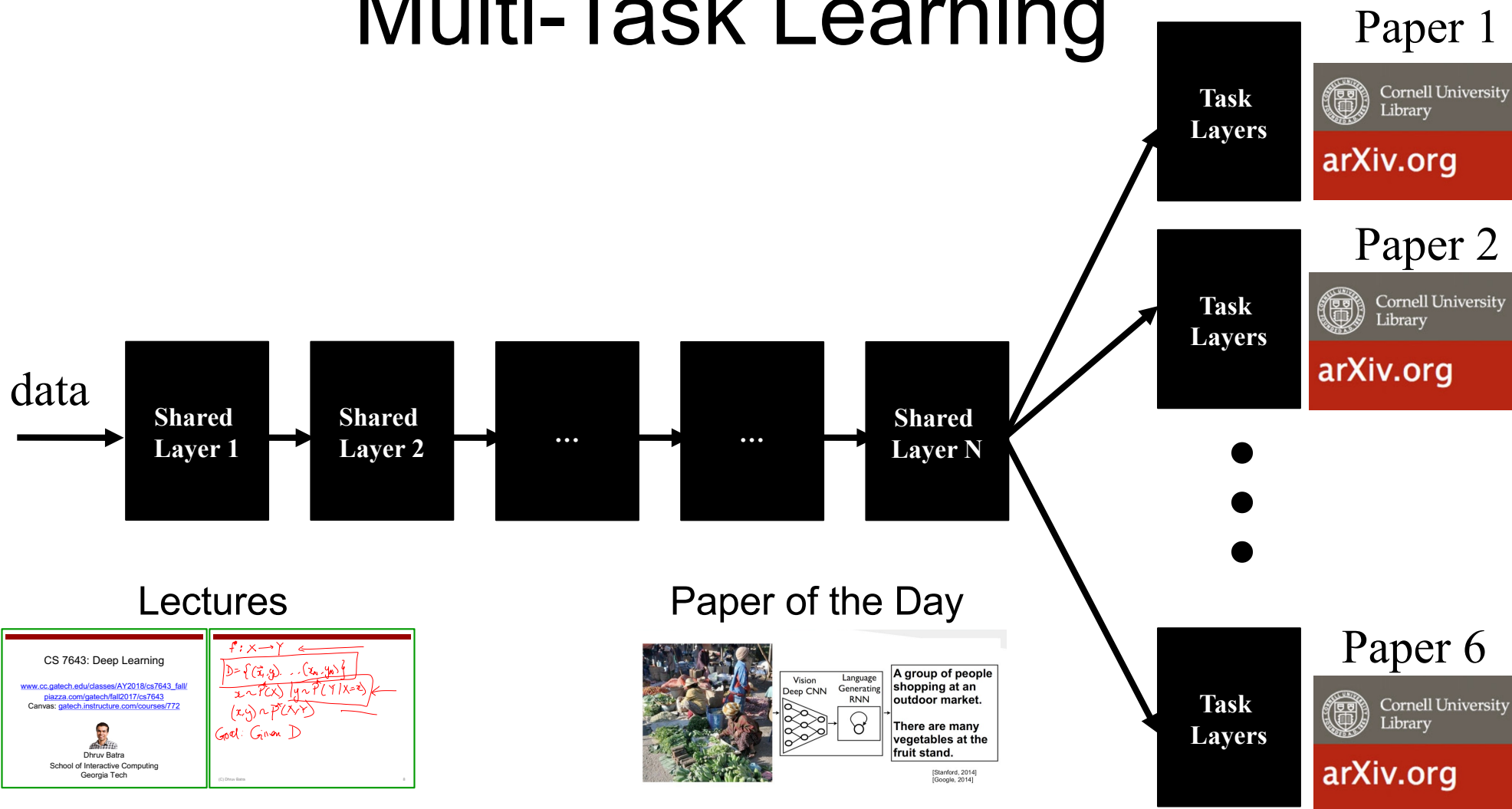
# Paper Reading Intuition: Multi-Task Learning



# Paper Reading Intuition: Multi-Task Learning



# Paper Reading Intuition: Multi-Task Learning



# Paper Reviews

- Length
  - 200-400 words.
- Due: Midnight before class on Piazza
- Organization
  - Summary:
    - What is this paper about? What is the main contribution? Describe the main approach & results. Just facts, no opinions yet.
  - List of positive points / Strengths:
    - Is there a new theoretical insight? Or a significant empirical advance? Did they solve a standing open problem? Or is a good formulation for a new problem? Or a faster/better solution for an existing problem? Any good practical outcome (code, algorithm, etc)? Are the experiments well executed? Useful for the community in general?
  - List of negative points / Weaknesses:
    - What would you do differently? Any missing baselines? missing datasets? any odd design choices in the algorithm not explained well? quality of writing? Is there sufficient novelty in what they propose? Has it already been done? Minor variation of previous work? Why should anyone care? Is the problem interesting and significant?
  - Reflections
    - How does this relate to other papers we have read? What are the next research directions in this line of work?

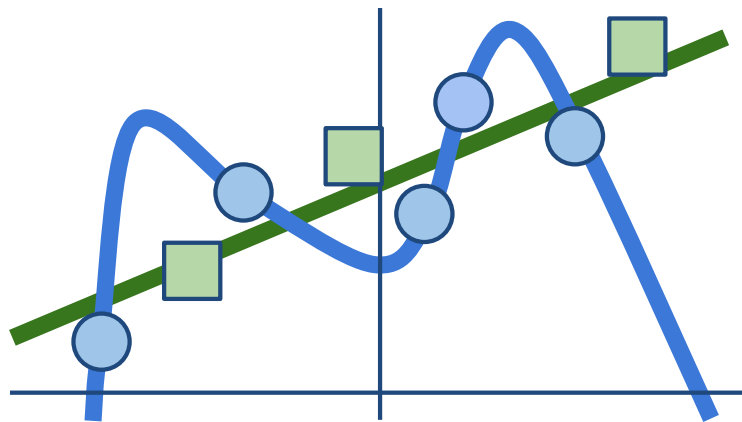
# Presentations

- Frequency
  - Once in the semester: 5 min presentation.
- Expectations
  - Present details of 1 paper
    - Describe formulation, experiment, approaches, datasets
    - Encouraged to present a broad picture
    - Show results; demo code if possible
  - Please clearly cite the source of each slide that is not your own.
  - Meet with TA 1 week before class to dry run presentation
    - Worth 40% of presentation grade

# Recap of last time

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

**Data loss:** Model predictions should match training data



**Regularization:** Model should be “simple”, so it works on test data

**Occam’s Razor:**

*“Among competing hypotheses, the simplest is the best”*

William of Ockham, 1285 - 1347

# Regularization

$\lambda$  = regularization strength  
(hyperparameter)

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

In common use:

**L2 regularization**

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L1 regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Elastic net (L1 + L2)  $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Dropout (will see later)

Fancier: Batch normalization, stochastic depth



# Neural networks: without the brain stuff

(**Before**) Linear score function:  $f = Wx$

# Neural networks: without the brain stuff

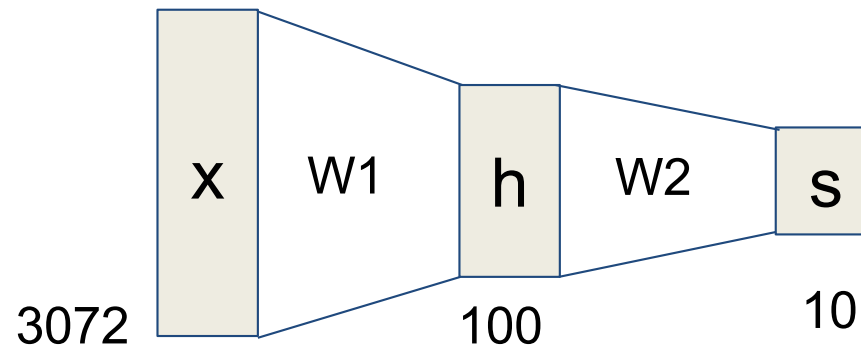
**(Before)** Linear score function:  $f = \underline{W}x$

**(Now)** 2-layer Neural Network  $f = \underline{W}_2 \max(0, \underline{W}_1 x)$

# Neural networks: without the brain stuff

(**Before**) Linear score function:  $f = Wx$

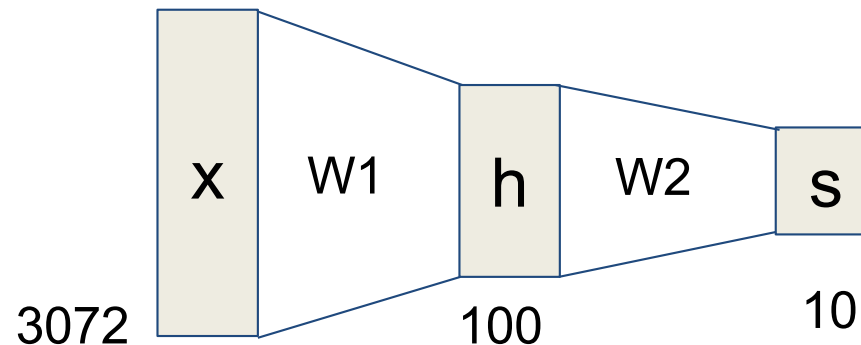
(**Now**) 2-layer Neural Network  $f = W_2 \max(0, W_1 x)$



# Neural networks: without the brain stuff

(**Before**) Linear score function:  $f = Wx$

(**Now**) 2-layer Neural Network  $f = W_2 \max(0, W_1 x)$



# Neural networks: without the brain stuff

(**Before**) Linear score function:  $f = Wx$

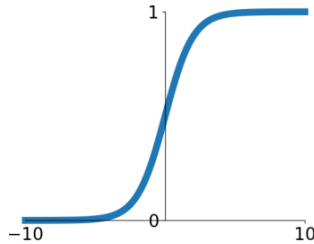
(**Now**) 2-layer Neural Network  $f = W_2 \max(0, W_1 x)$   
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

# Activation functions

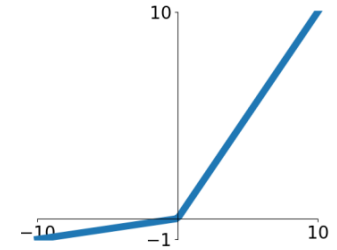
## Sigmoid

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



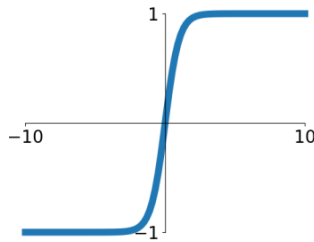
## Leaky ReLU

$$\max(0.1x, x)$$



## tanh

$$\tanh(x)$$

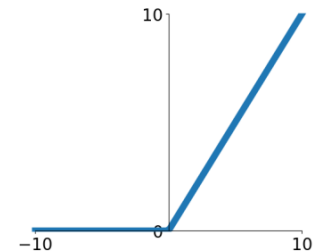


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

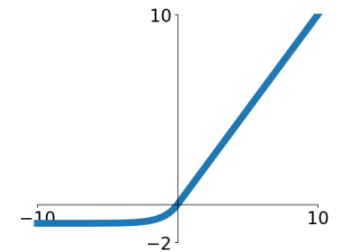
## ReLU

$$\max(0, x)$$

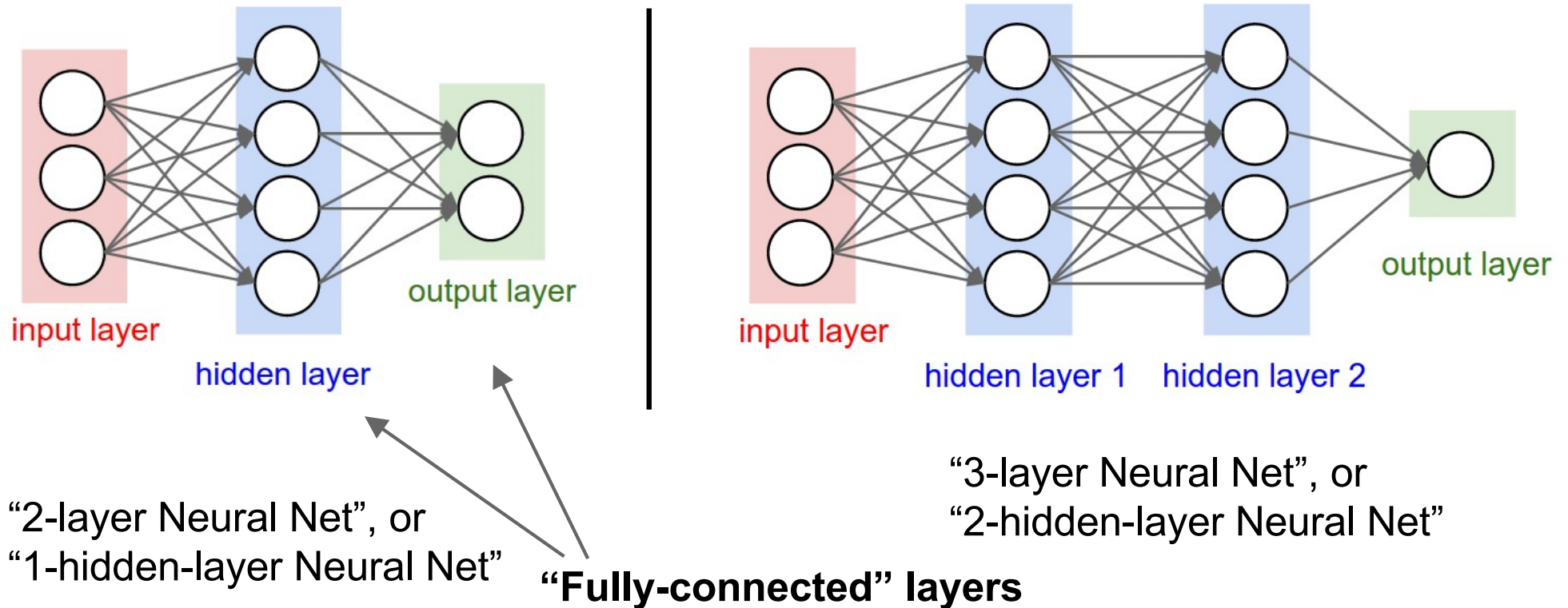


## ELU

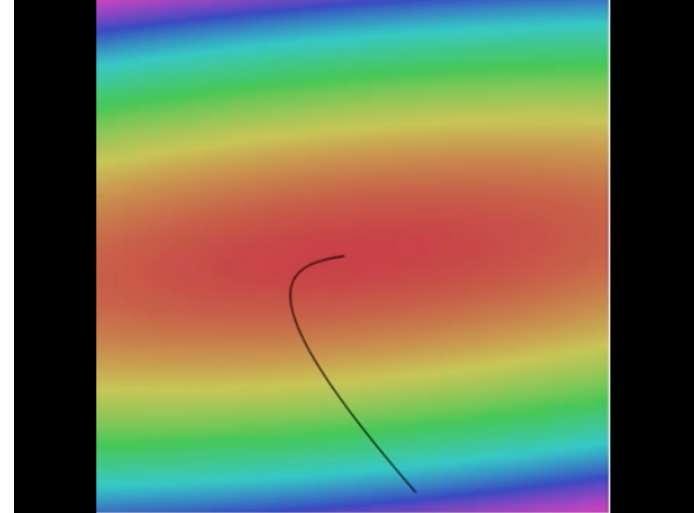
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Neural networks: Architectures



# Optimization



```
# Vanilla Gradient Descent

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

[Landscape image](#) is [CC0 1.0](#) public domain  
[Walking man image](#) is [CC0 1.0](#) public domain



# 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
```

# How do we compute gradients?

- 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

*code*

```
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
```

Symbolic  
Differentiation  
of the Closed-form

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

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

Automatic  
Differentiation

*code*

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

Numerical  
Differentiation

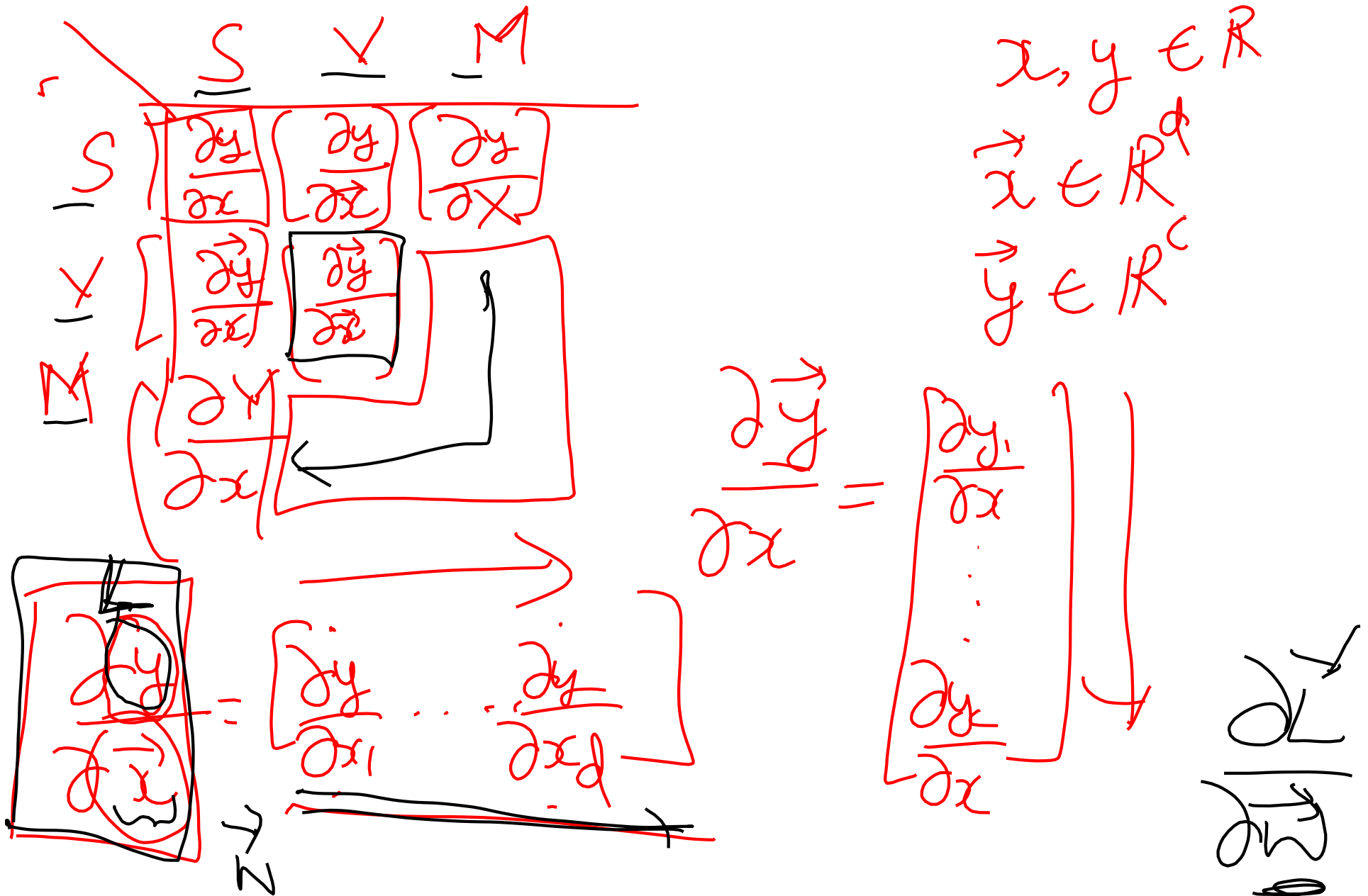
```
f'(x):
  h = 0.000001
  (f(x+h) - f(x)) / h
```

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

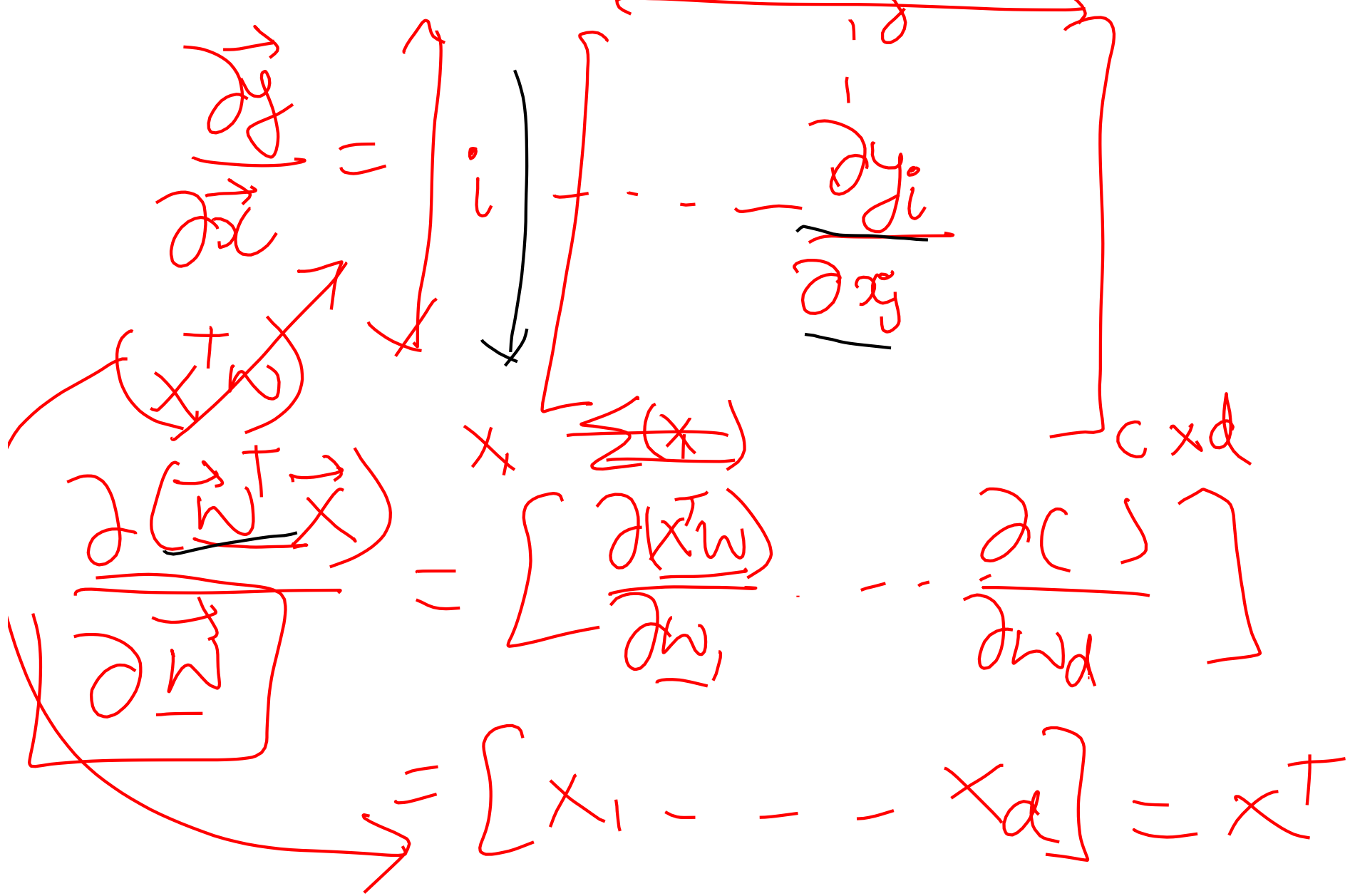
$f'(x_0) = f'(x_0)$   
Exact

# Matrix/Vector Derivatives Notation

2



# Matrix/Vector Derivatives Notation



# Vector Derivative Example

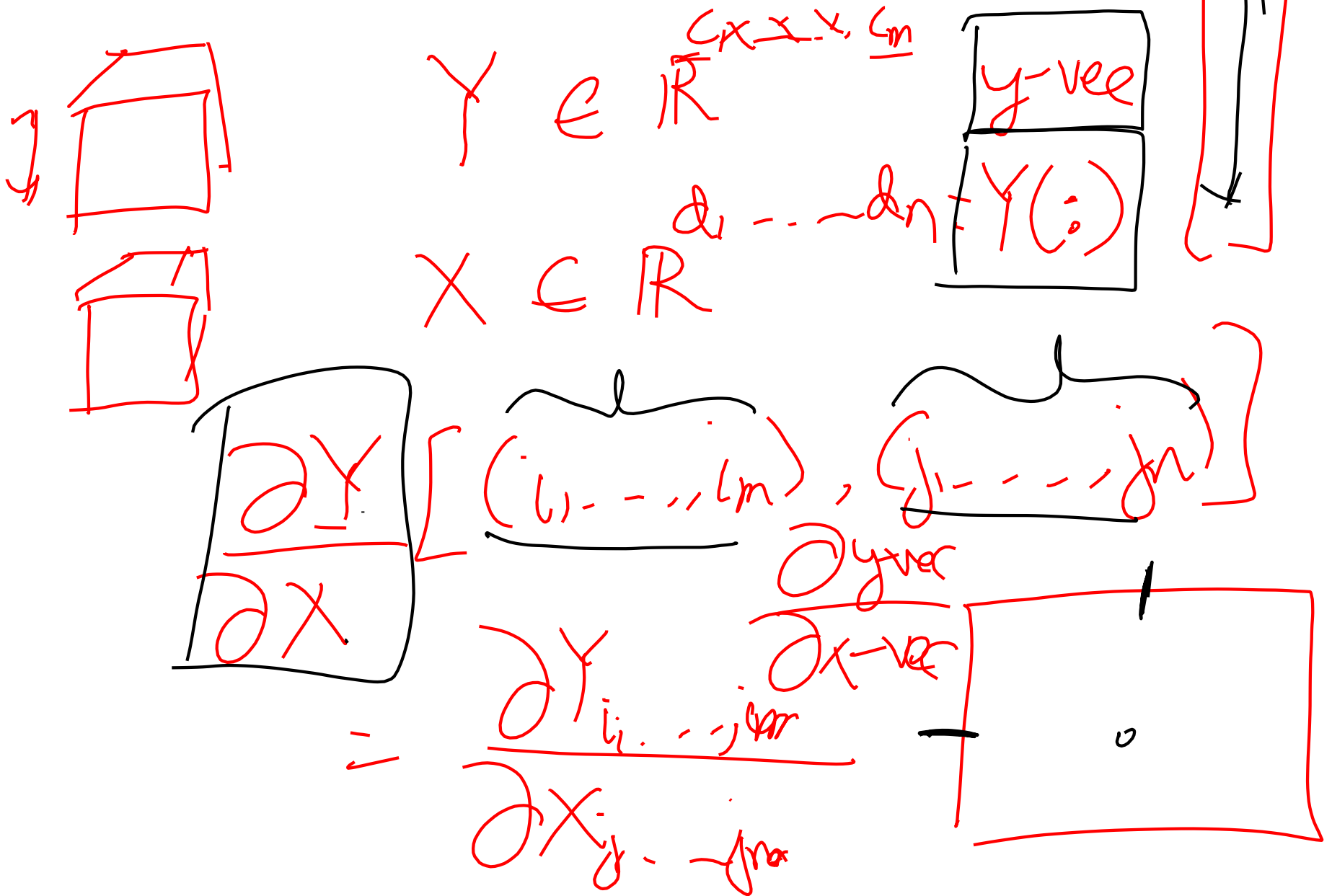
$$\frac{\partial (\vec{w}^T A \vec{w})}{\partial \vec{w}} = \underline{\underline{2\vec{w}^T A}}$$

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

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

$$\left[ \frac{\partial y_i}{\partial x_j} \right]$$

# Extension to Tensors



# Chain Rule: Composite Functions

$$L(x) = f(g(x)) = \underline{(f \circ g)(x)}$$

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

$$\begin{aligned} & \left( \frac{\partial}{\partial x} \left( g \circ g_1 \circ g_2 \dots \circ g_l \right) \right) (x) \\ & \left( g_l \circ g_{l-1} \circ \dots \circ g_1 \right) (x) \end{aligned}$$



# Chain Rule: Scalar Case

$$x \rightarrow z \rightarrow \underline{y}$$

$$\begin{aligned} z &= g(x) \\ y &= f(z) \end{aligned}$$

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

$$L = y$$

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

$$J = J \cdot J$$

# Chain Rule: Vector Case

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

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

$$f(g(\vec{x}))$$

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

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

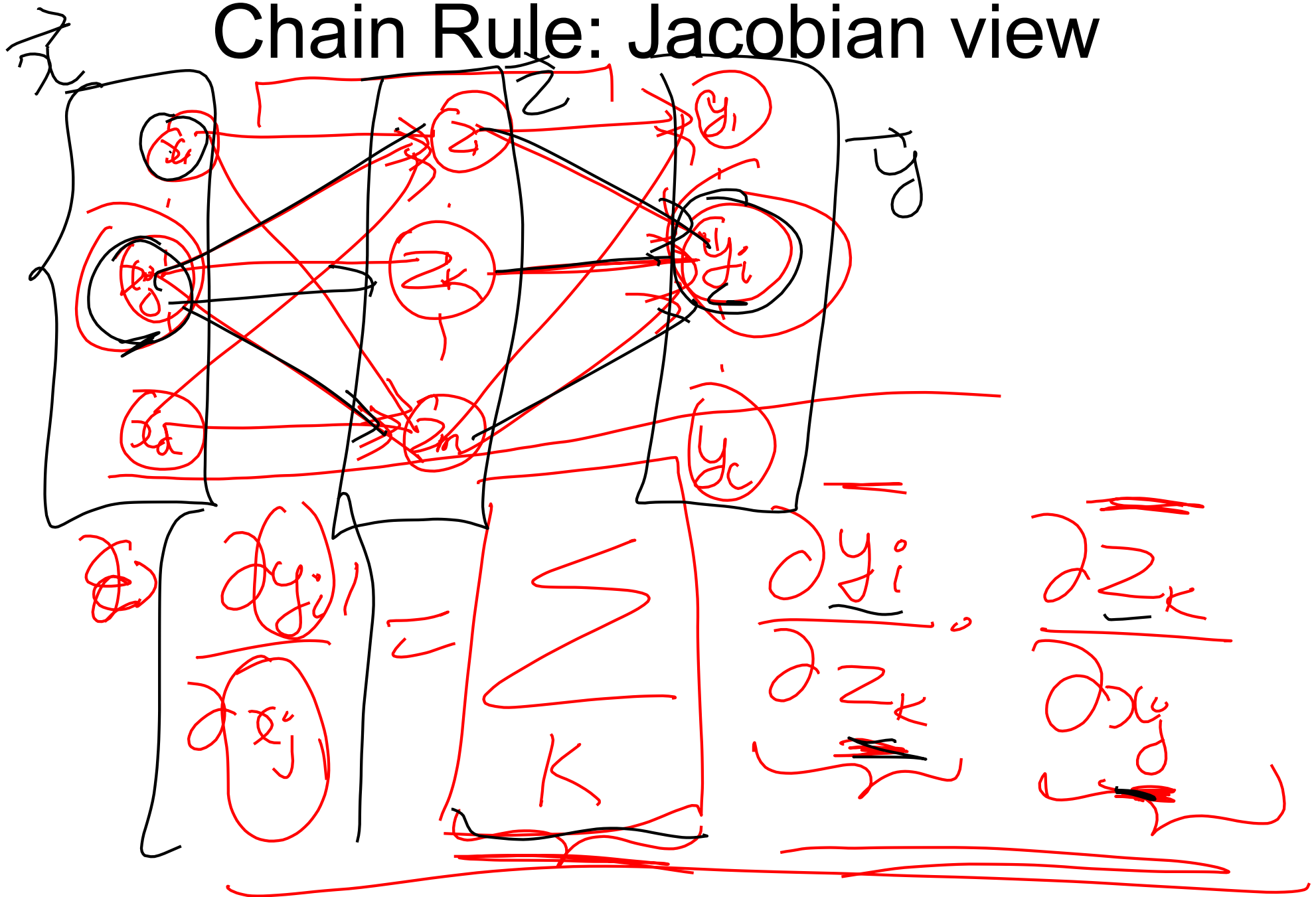
$$\vec{z} \in \mathbb{R}^c$$

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

$$= \begin{bmatrix} \frac{\partial \vec{y}}{\partial \vec{z}} & \frac{\partial \vec{y}}{\partial \vec{x}} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial y_i}{\partial x_j} \end{bmatrix}_{c \times d} = \begin{bmatrix} \frac{\partial y_i}{\partial z_k} \end{bmatrix}_{m \times c} \begin{bmatrix} \frac{\partial z_k}{\partial x_j} \end{bmatrix}_{c \times d}$$

# Chain Rule: Jacobian view

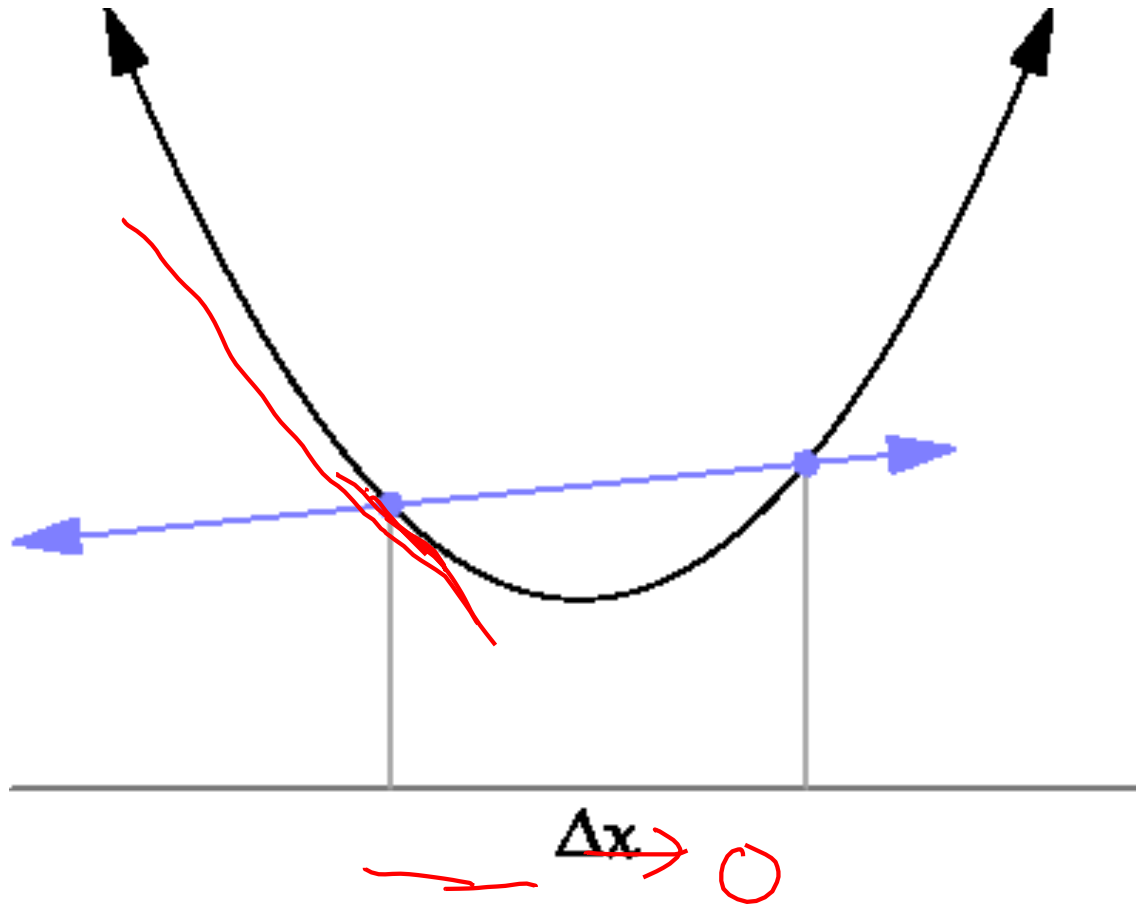


# Plan for Today

- Computational Graphs
  - Notation + example
- Computing Gradients
  - Forward mode vs Reverse mode AD

# How do we compute gradients?

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



**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,  
?,  
?,

$$\frac{(1.25322 - 1.25347)}{0.0001} = -2.5$$

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

?,  
?,...]

**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**,  
?,  
?,

$$\frac{(1.25353 - 1.25347)}{0.0001} = 0.6$$

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

?,...]

**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 (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,  
?,  
?,  
?,  
?,  
?,  
?,  
?,...]

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**,  
?,  
?

$$(1.25347 - 1.25347)/0.0001 = 0$$

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

?,...]

# Numerical vs Analytic Gradients

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

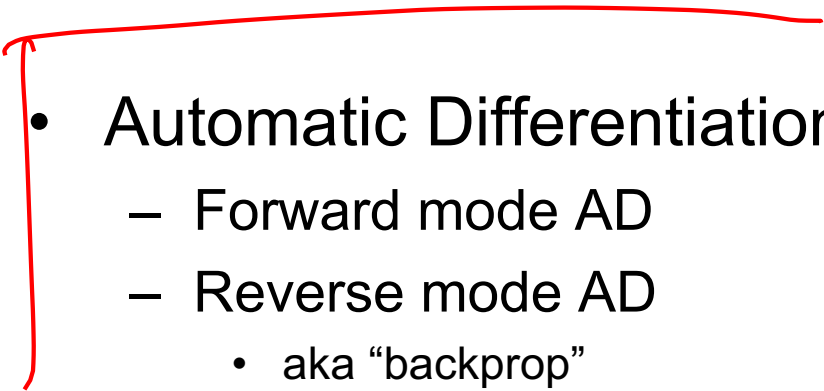
**Numerical gradient:** slow :(, approximate :(, easy to write :)

**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?

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

# Chain Rule: Vector Case

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

$$g: \mathbb{R}^d \rightarrow \mathbb{R}^m \quad f(g(\vec{x}))$$

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

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

$$\vec{z} \in \mathbb{R}^c$$

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

$$= \begin{bmatrix} \frac{\partial \vec{y}}{\partial \vec{z}} & \frac{\partial \vec{y}}{\partial \vec{x}} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial y_i}{\partial x_j} \end{bmatrix}_{c \times d} = \begin{bmatrix} \frac{\partial y_i}{\partial z_k} \end{bmatrix}_{m \times c} \begin{bmatrix} \frac{\partial z_k}{\partial x_j} \end{bmatrix}_{c \times d}$$



# Linear Classifier: Logistic Regression

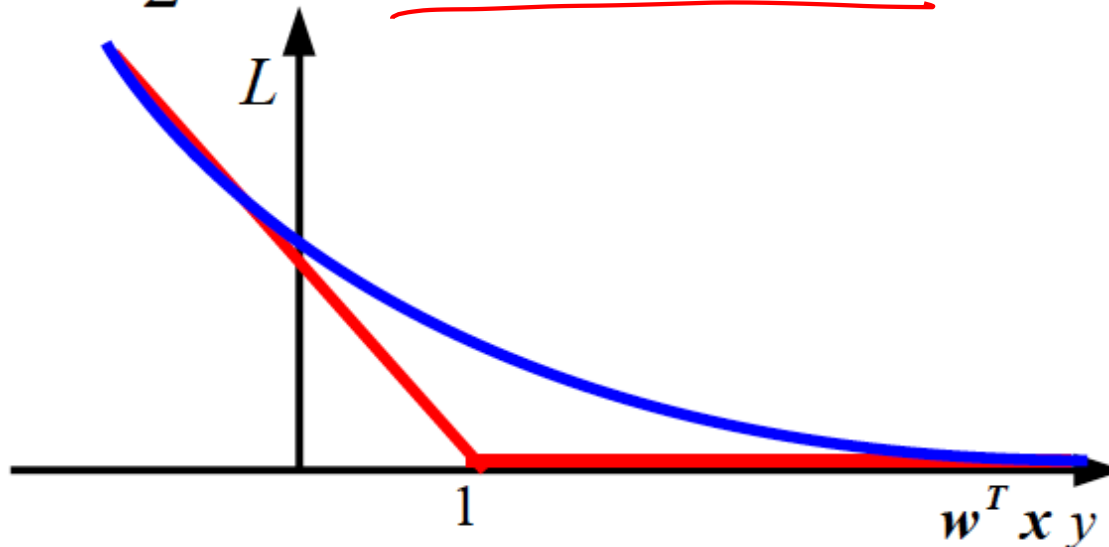
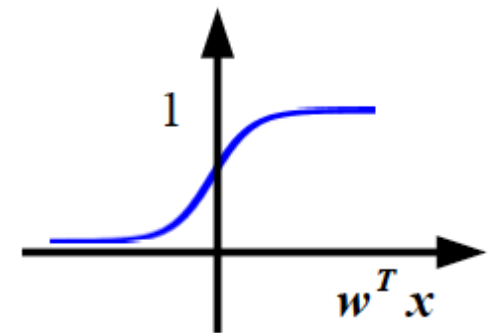
Input:  $\mathbf{x} \in \mathbb{R}^D$

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

Parameters:  $\mathbf{w} \in \mathbb{R}^D$

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

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



Log Loss

# Logistic Regression Derivatives

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

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

# Convolutional network (AlexNet)

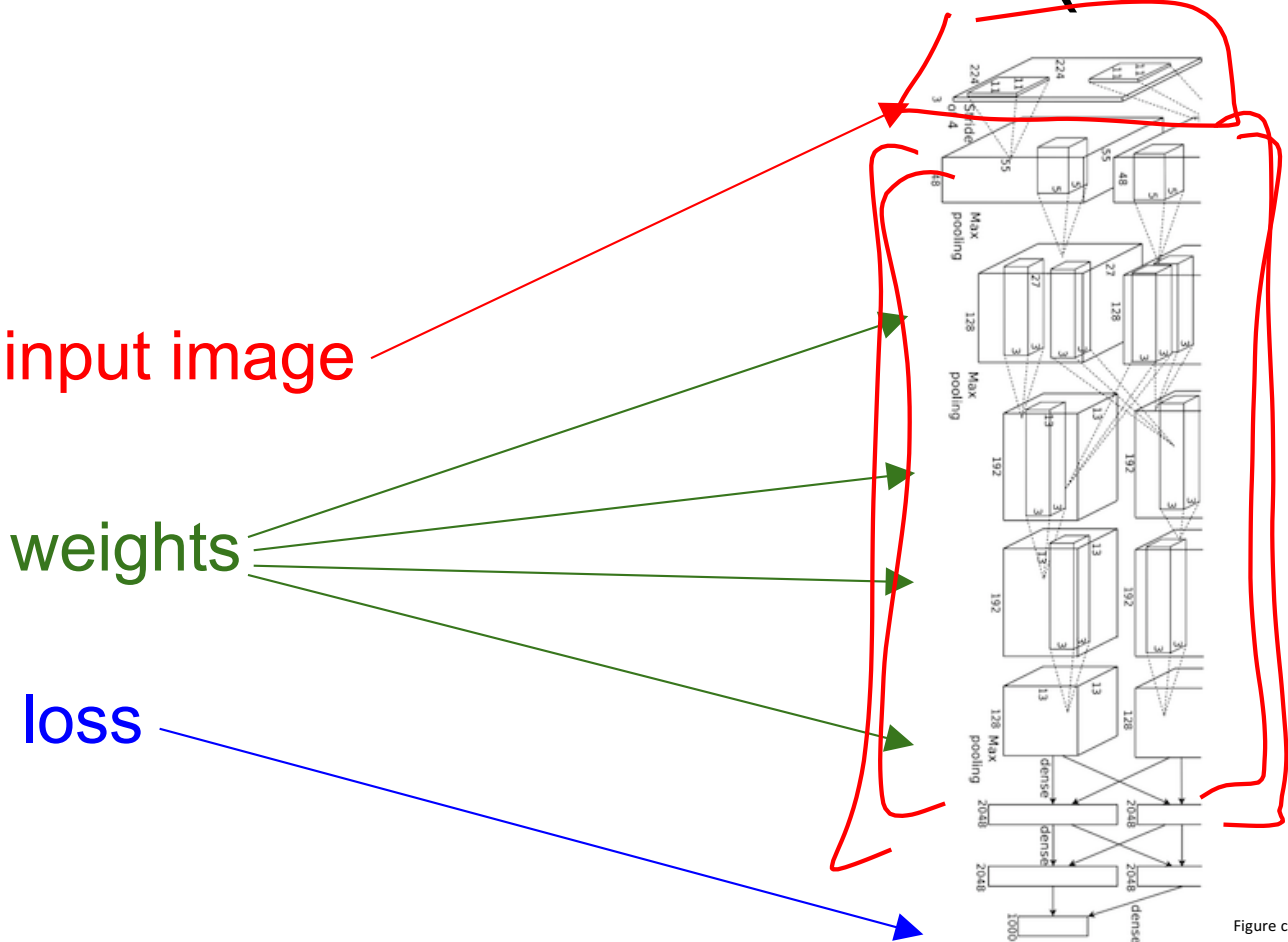


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Li

# Neural Turing Machine

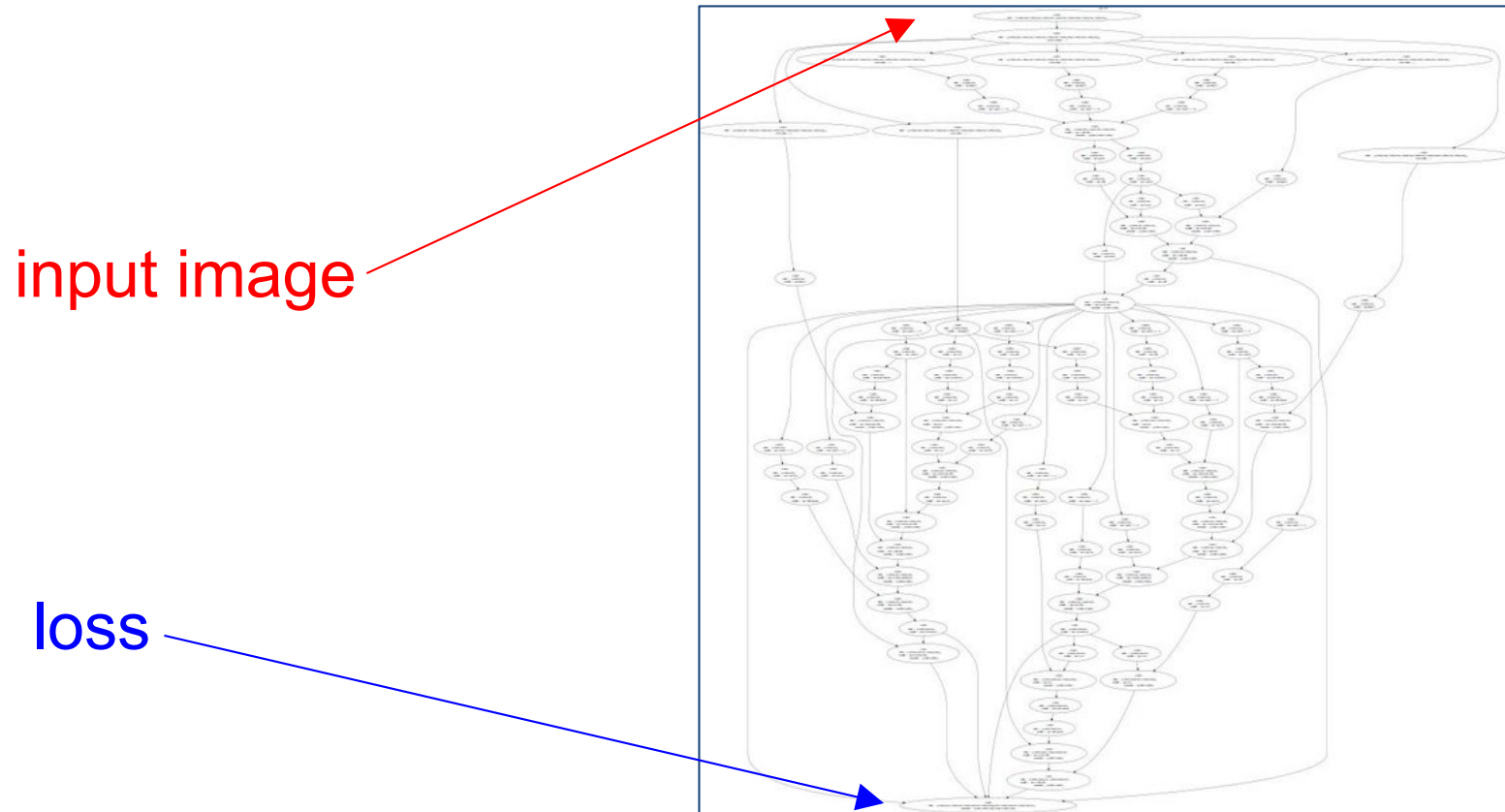


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

# Neural Turing Machine

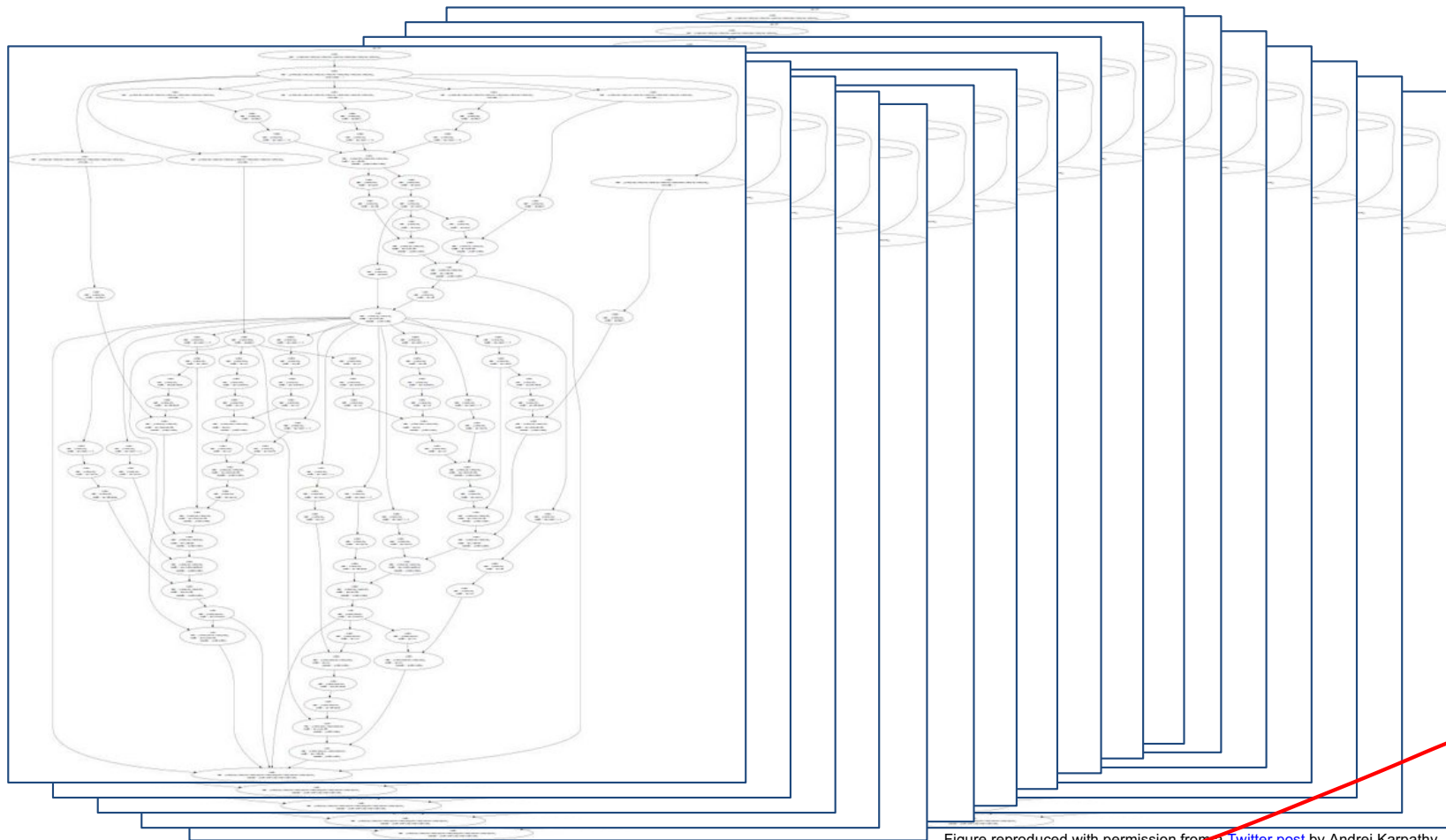


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

# Chain Rule: Vector Case

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

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

$$f(g(\vec{x}))$$

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

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

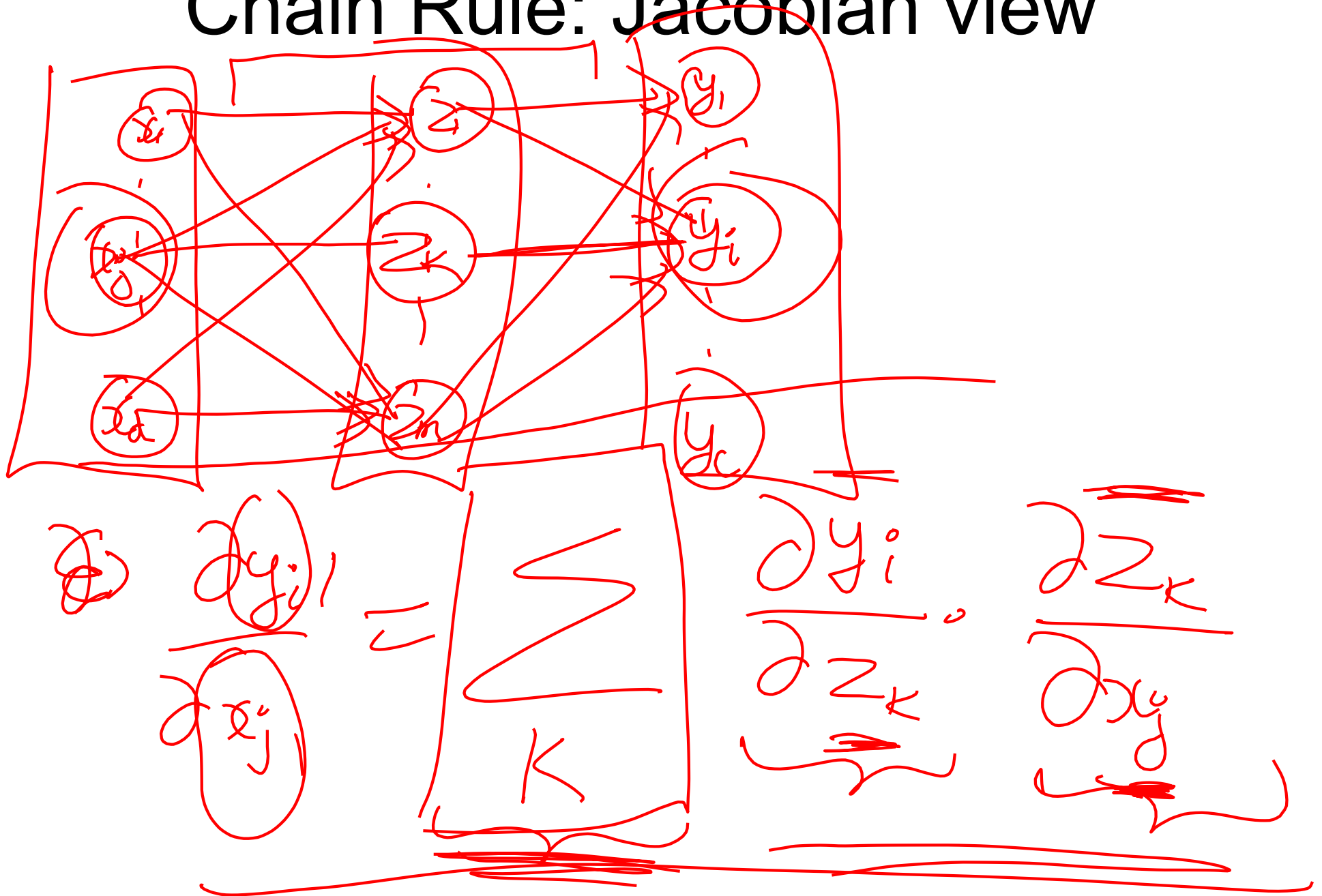
$$\vec{z} \in \mathbb{R}^c$$

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

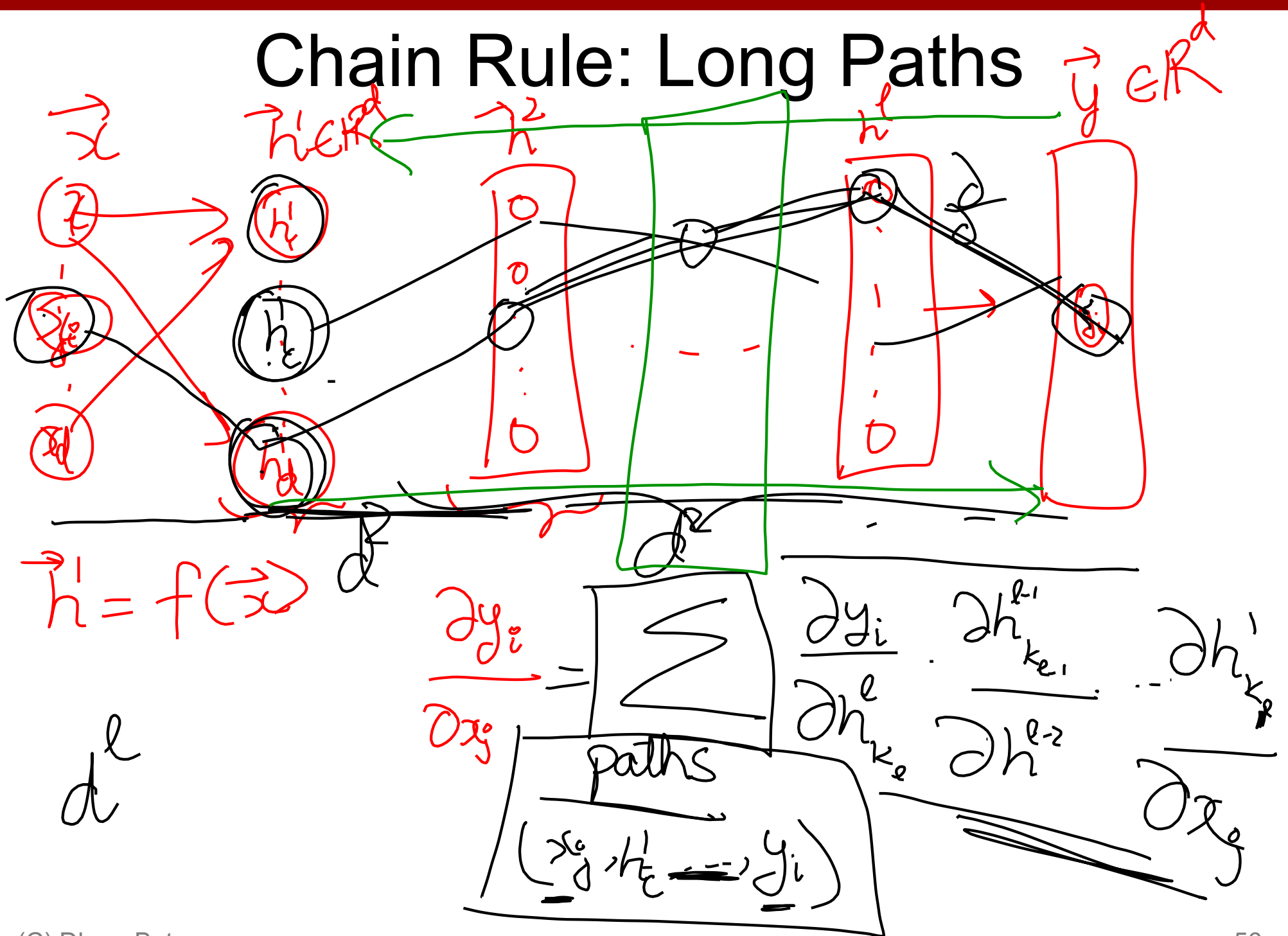
$$= \begin{bmatrix} \frac{\partial \vec{y}}{\partial \vec{z}} & \frac{\partial \vec{y}}{\partial \vec{x}} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial y_i}{\partial x_j} \end{bmatrix}_{c \times d} = \begin{bmatrix} \frac{\partial y_i}{\partial z_k} \end{bmatrix}_{m \times c} \begin{bmatrix} \frac{\partial z_k}{\partial x_j} \end{bmatrix}_{c \times d}$$

# Chain Rule: Jacobian view



# Chain Rule: Long Paths





# Chain Rule: Long Paths

# Chain Rule: Long Paths

**current W:**

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

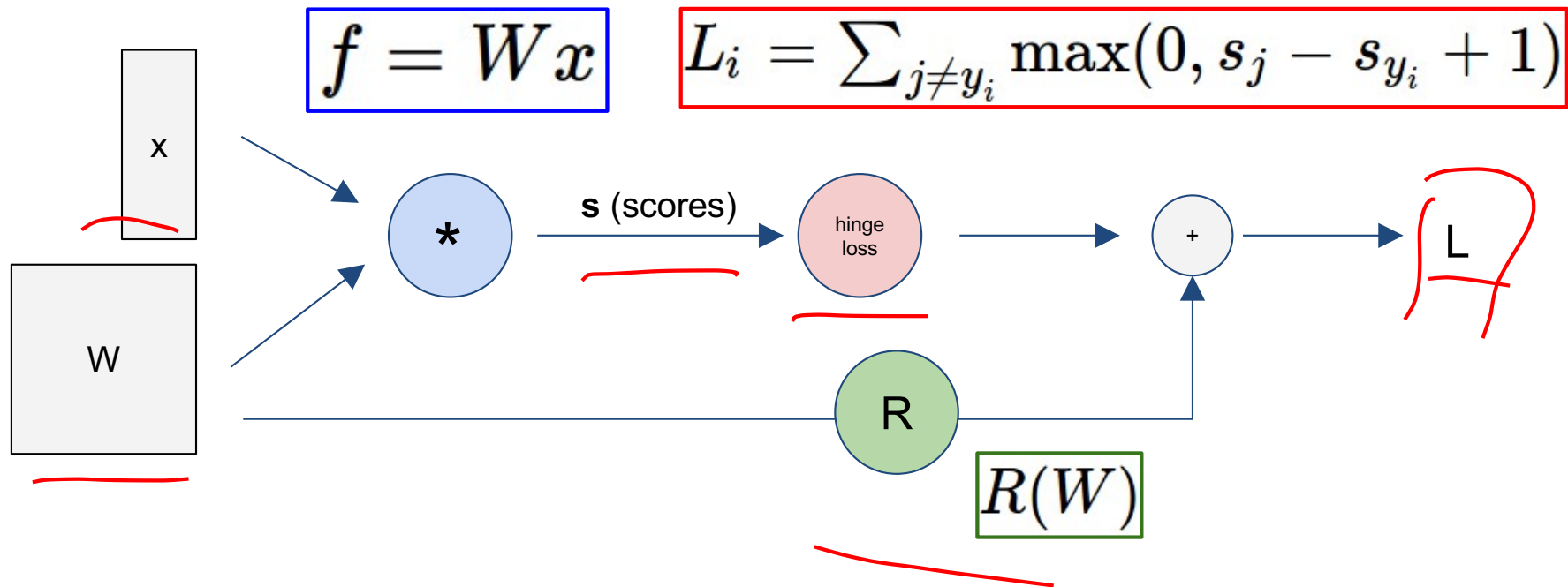
**loss 1.25347**

$dW = \dots$   
(some function  
data and  $W$ )

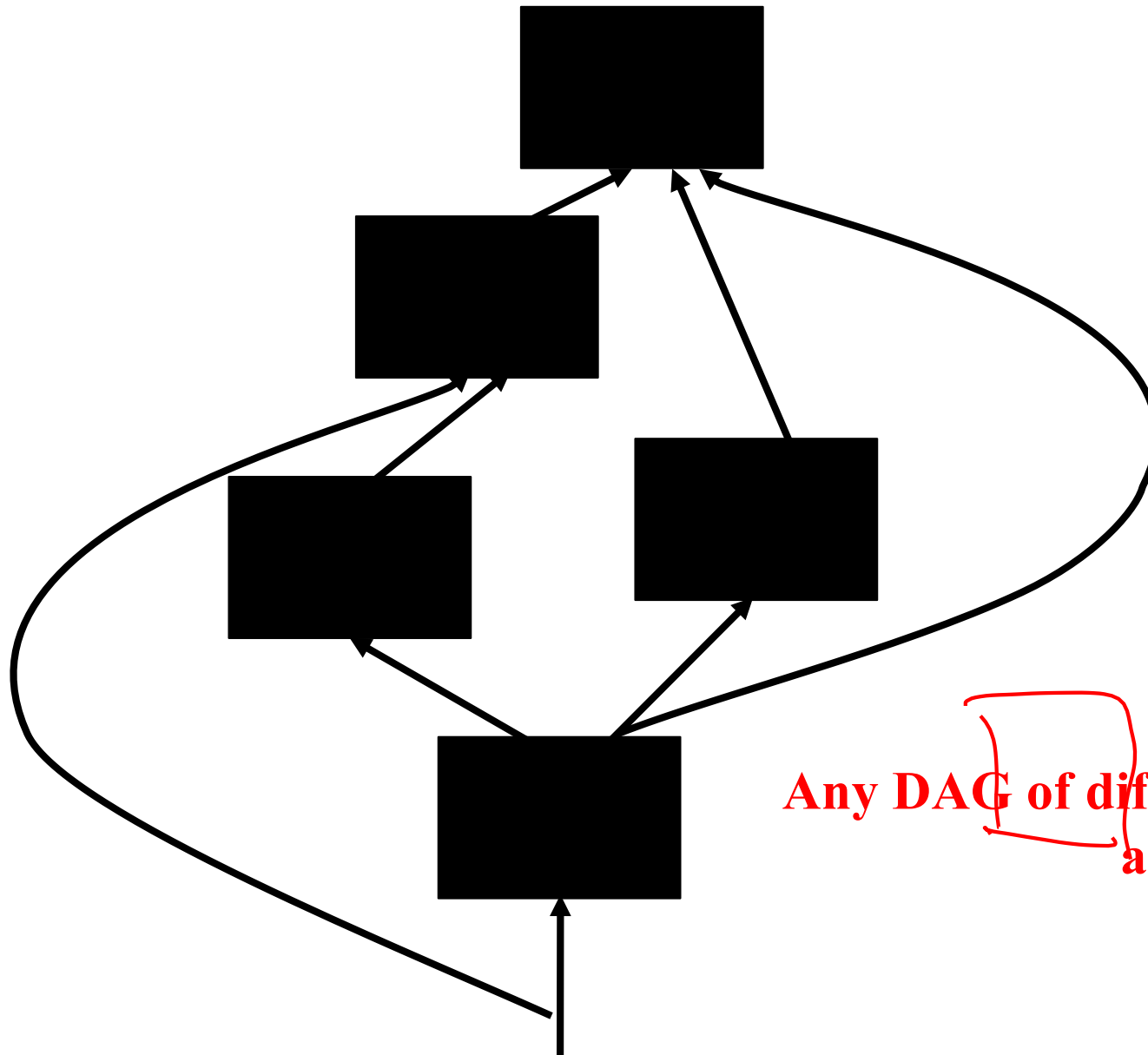
**gradient  $dW$ :**

[-2.5,  
0.6,  
0,  
0.2,  
0.7,  
-0.5,  
1.1,  
1.3,  
-2.1,...]

# Computational Graph



# Computational Graph



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

# Directed Acyclic Graphs (DAGs)

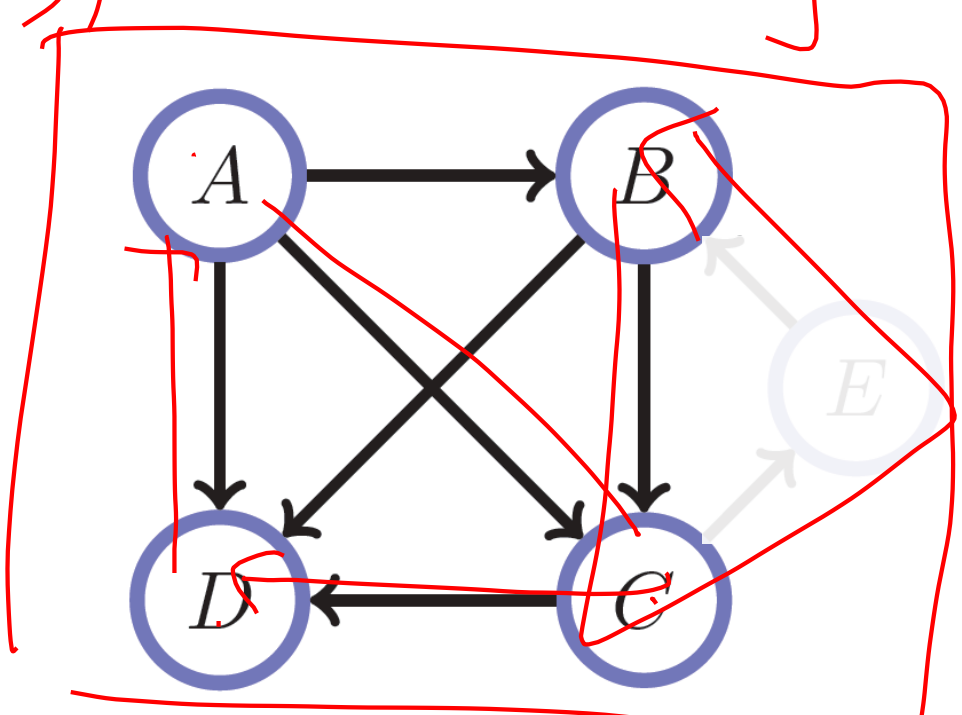
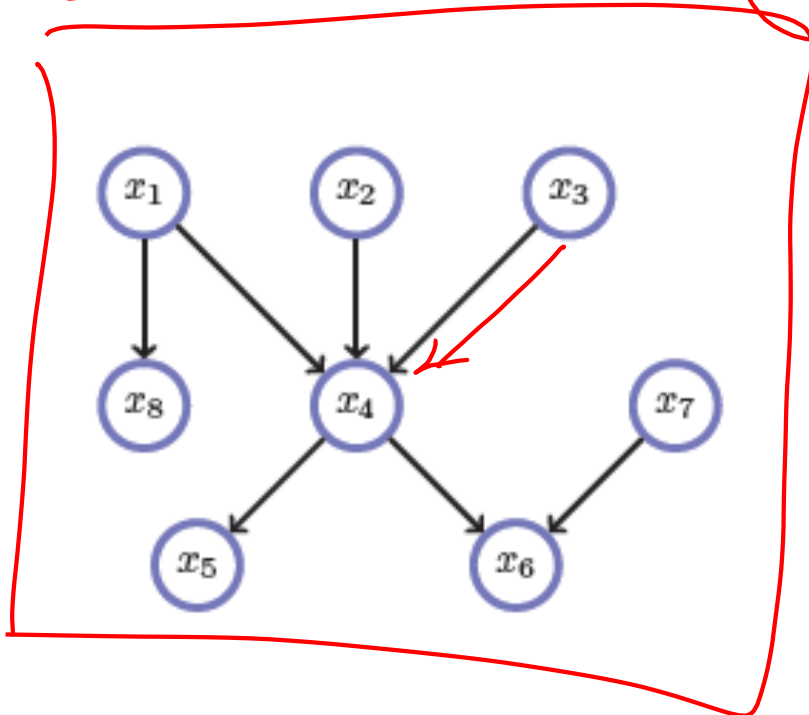
- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

$$E = \{ (v_i, v_j) \}$$

$$\{ \{v_i, v_j\} \}$$

$$(v_i, v_j) \dots v_j$$

$$(D, A) \in E$$

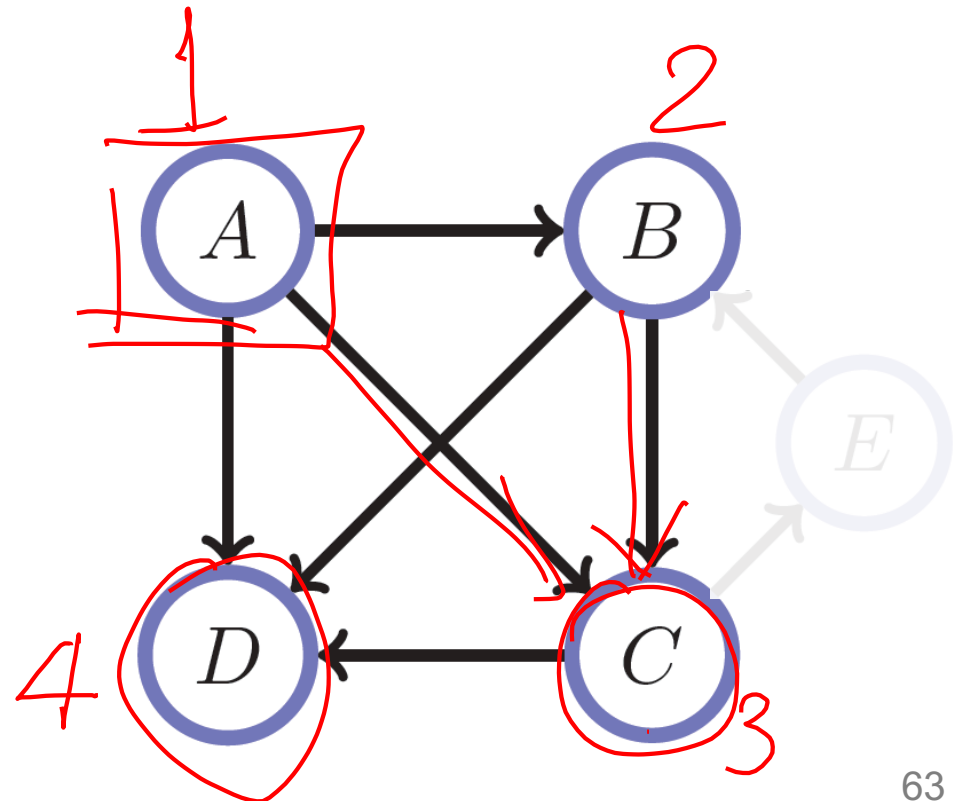
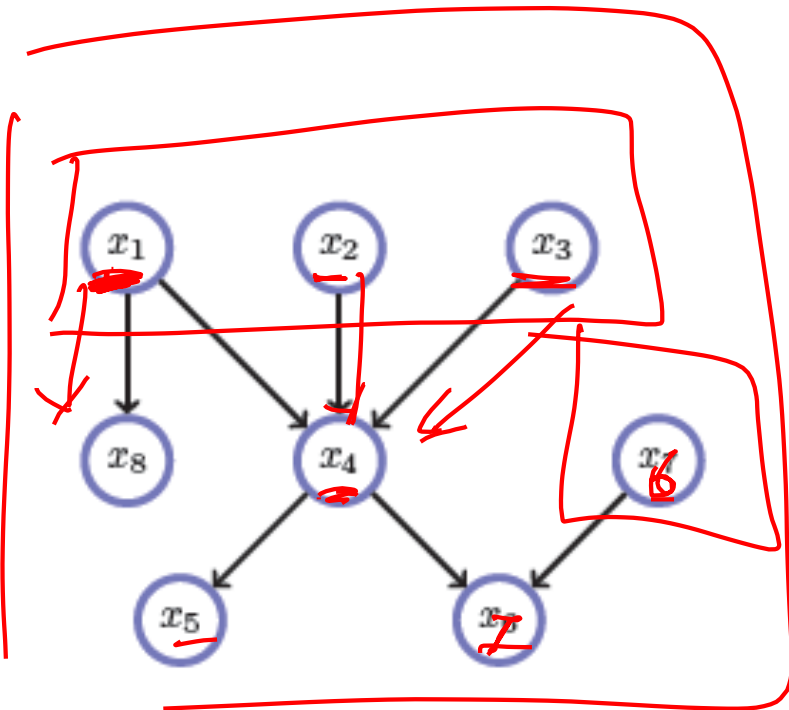


# Directed Acyclic Graphs (DAGs)

- Concept
  - Topological Ordering

$$\exists \sigma: V \rightarrow [n] = \{1, \dots, n\}$$

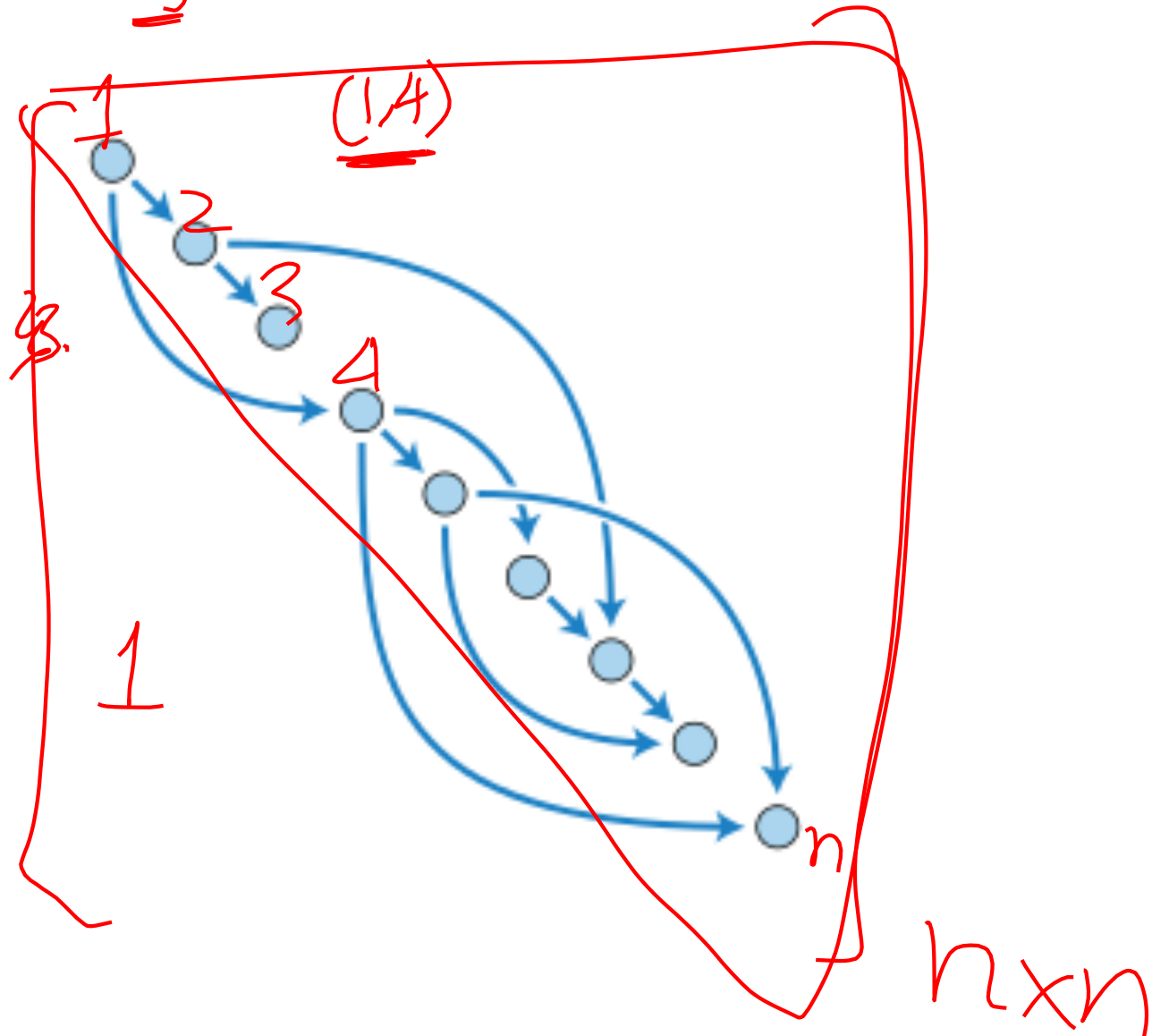
$$\text{s.t. } (v_i, v_j) \in E \implies \sigma(v_i) < \sigma(v_j)$$



# Directed Acyclic Graphs (DAGs)

$$a_{ij} = 1 \quad (i, j) \in E$$

$$A =$$

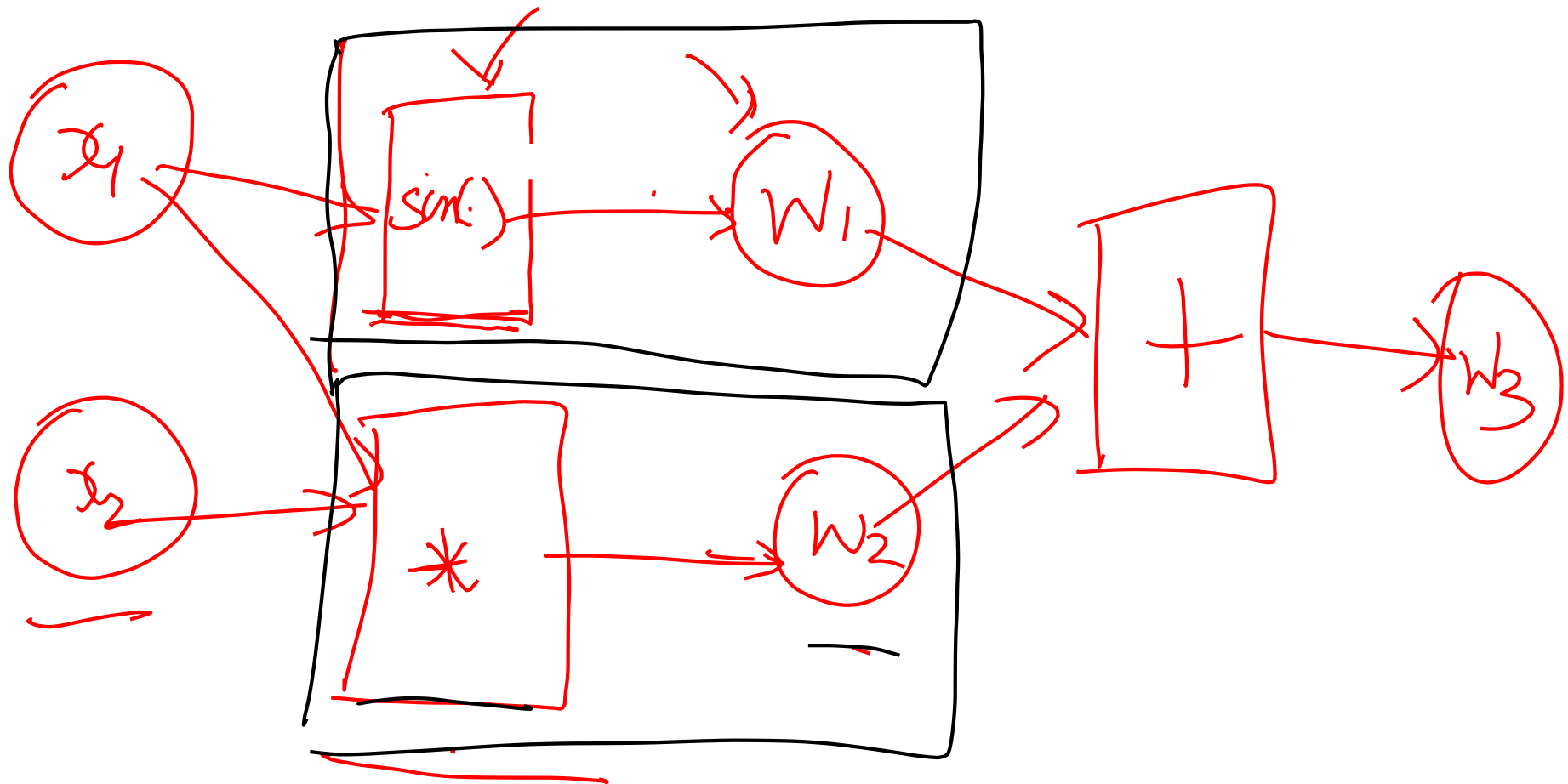




# Computational Graphs

- Notation #1

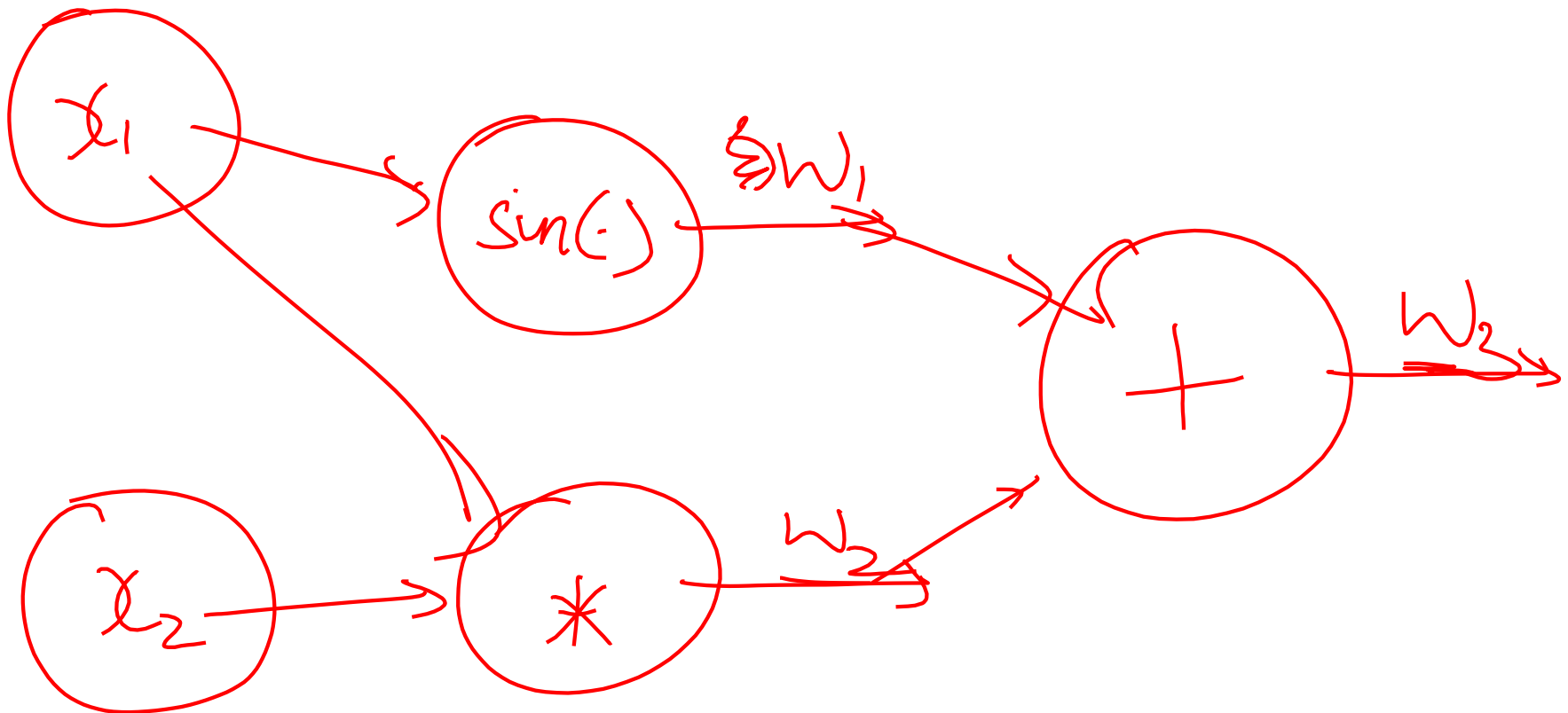
$$\underline{f(x_1, x_2)} = \underline{x_1 x_2} + \underline{\sin(x_1)}$$



# Computational Graphs

- Notation #2

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

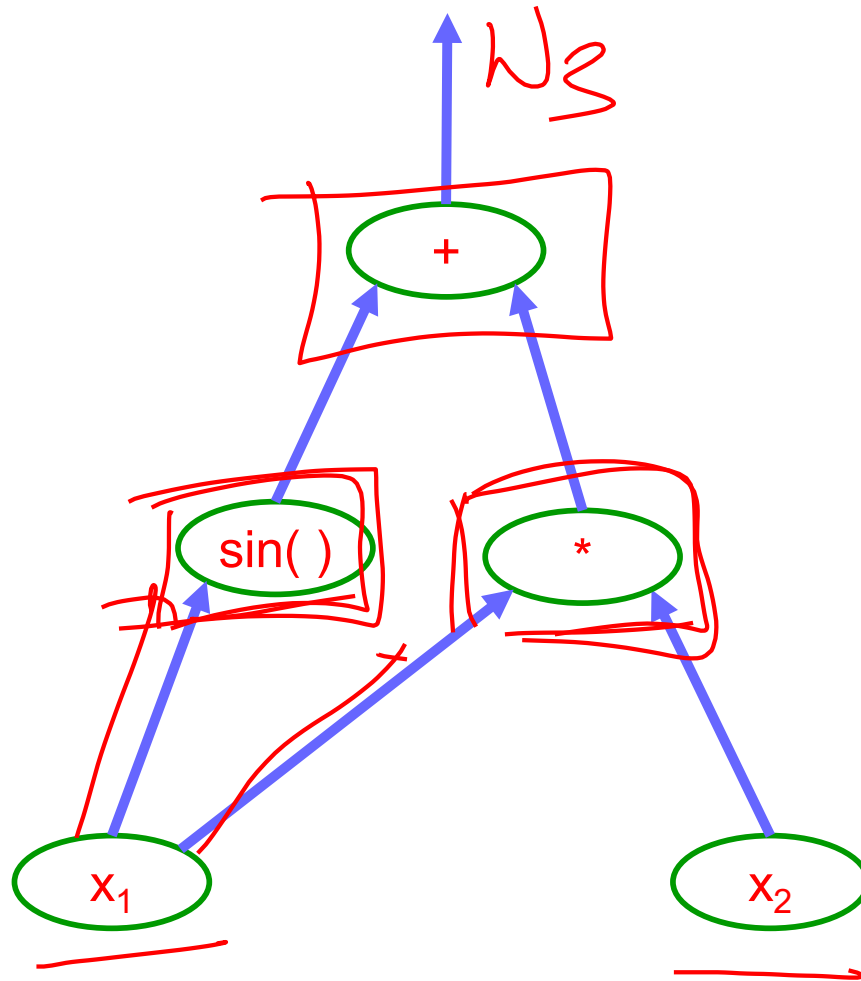


# Example

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

# Example

$$\underline{f(x_1, x_2) = x_1 x_2 + \sin(x_1)}$$

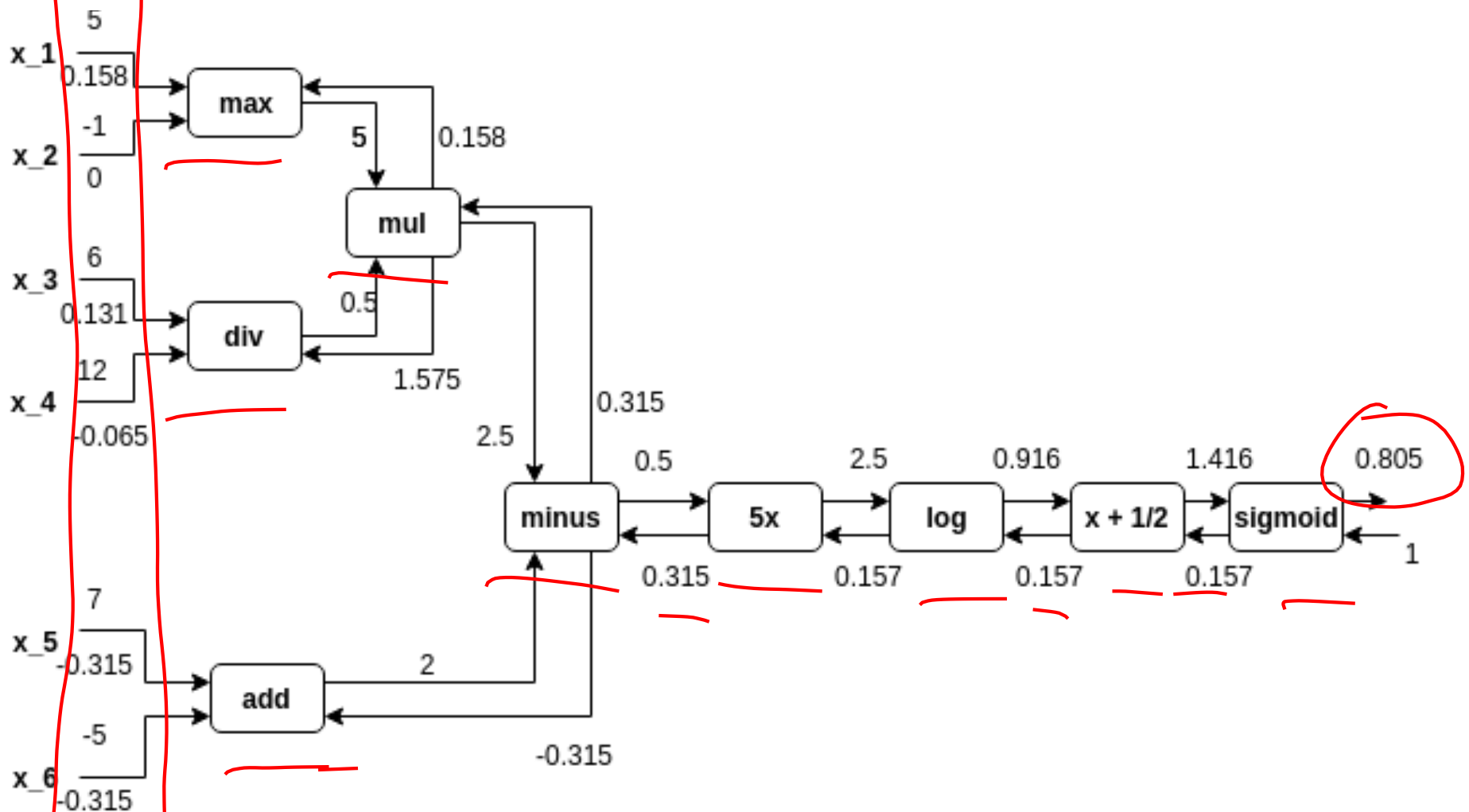


# HW0

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

# HW0 Submission by Samyak Datta

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



# Linear Classifier: Logistic Regression

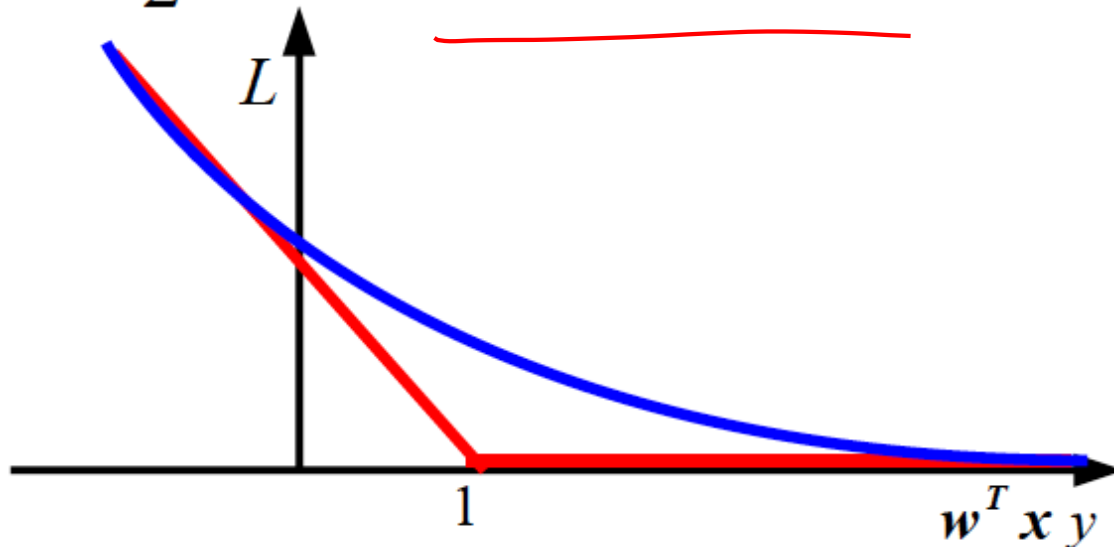
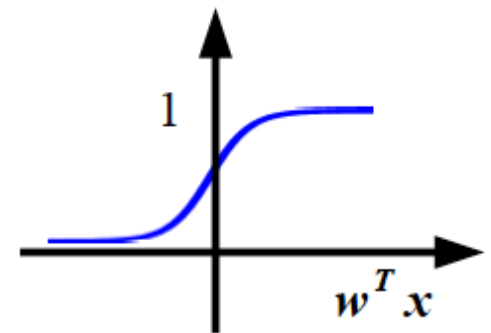
Input:  $\mathbf{x} \in \mathbb{R}^D$

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

Parameters:  $\mathbf{w} \in \mathbb{R}^D$

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

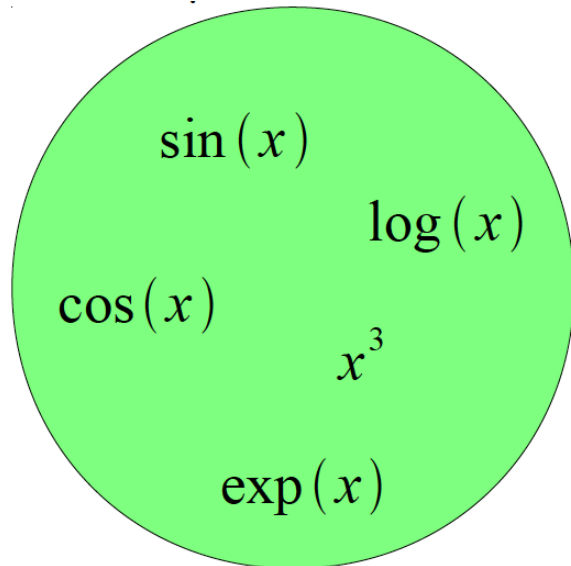
Loss:  $L = \frac{1}{2} \|\mathbf{w}\|^2 - \lambda \log(p(y|\mathbf{x}))$



Log Loss

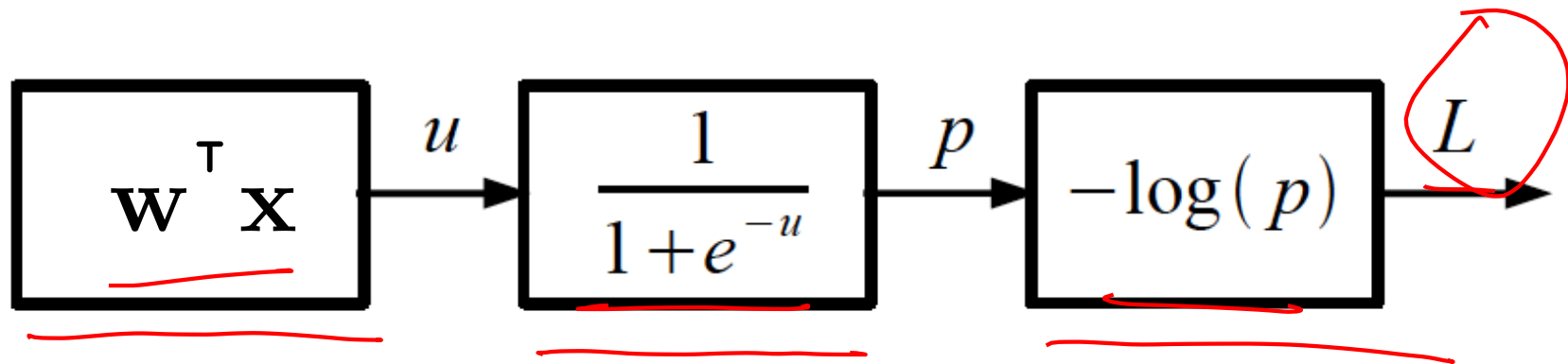
# Logistic Regression as a Cascade

Given a library of simple functions



Compose into a  
→  
complicate function

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

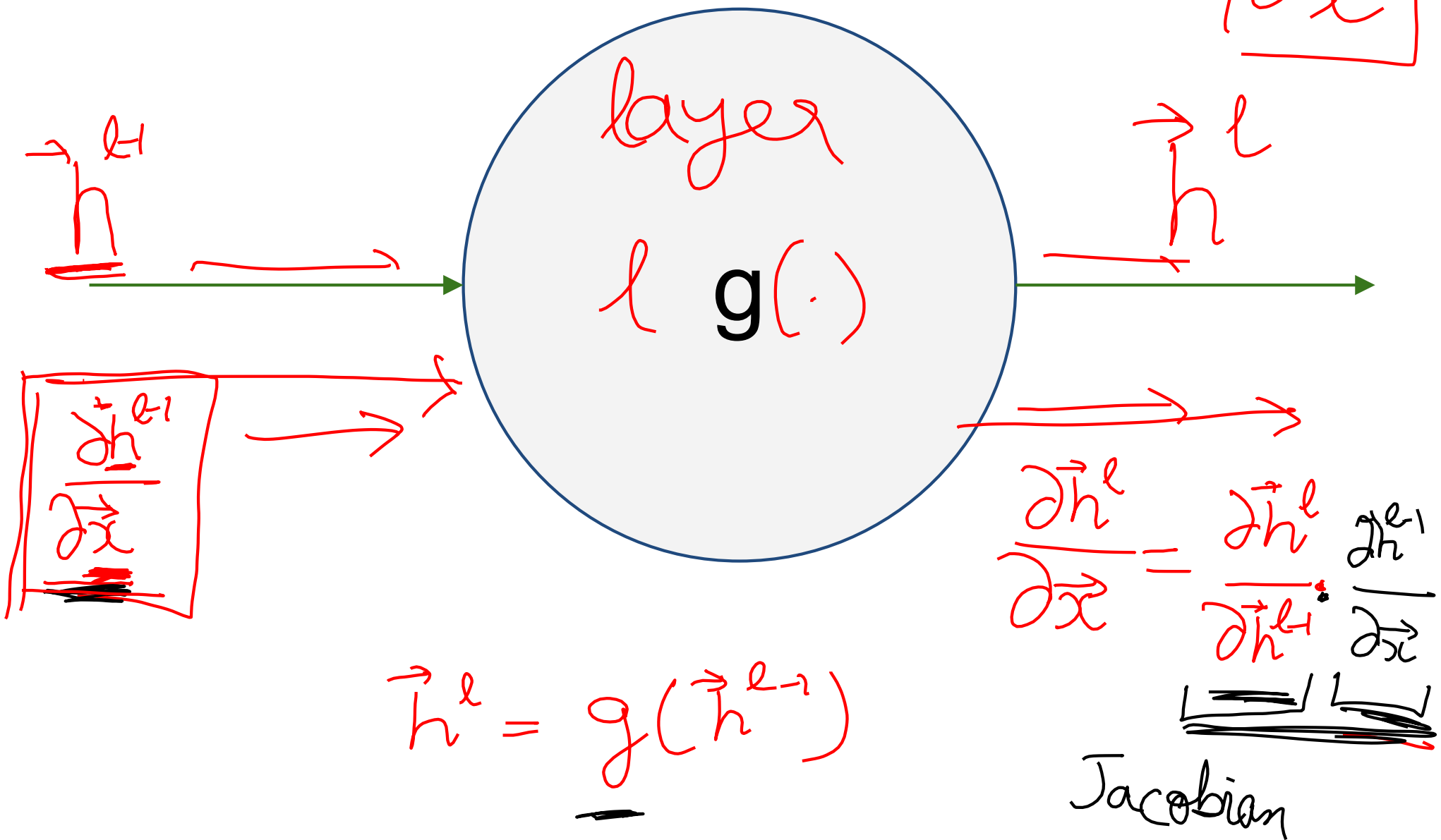




# Forward mode vs Reverse Mode

- Key Computations

# Forward mode AD

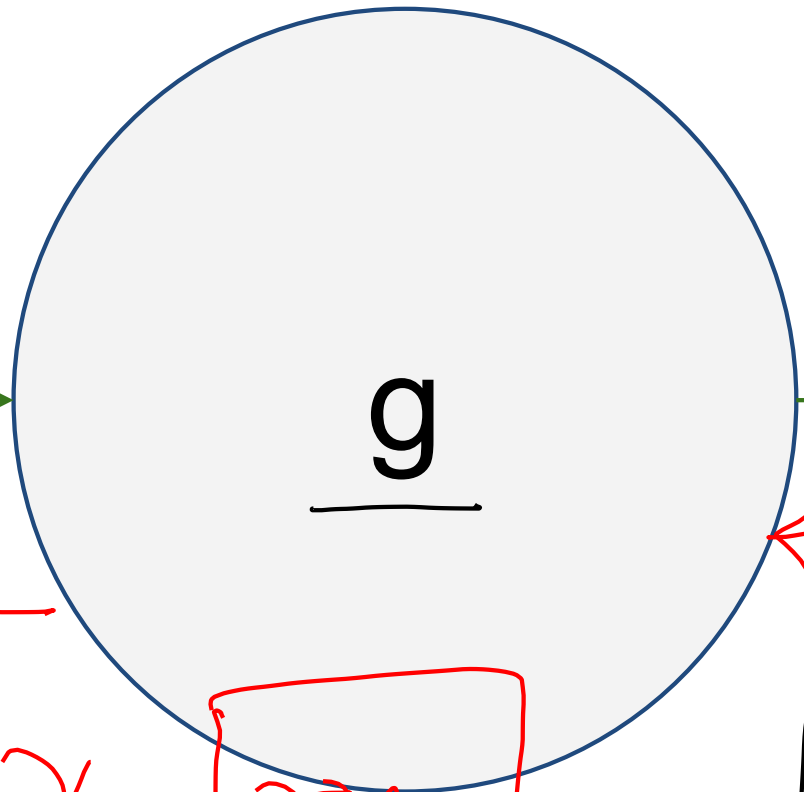


# Reverse mode AD

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

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

$$\vec{h}^l$$



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

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

input

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

Jacobian

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

# Example: Forward mode AD

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

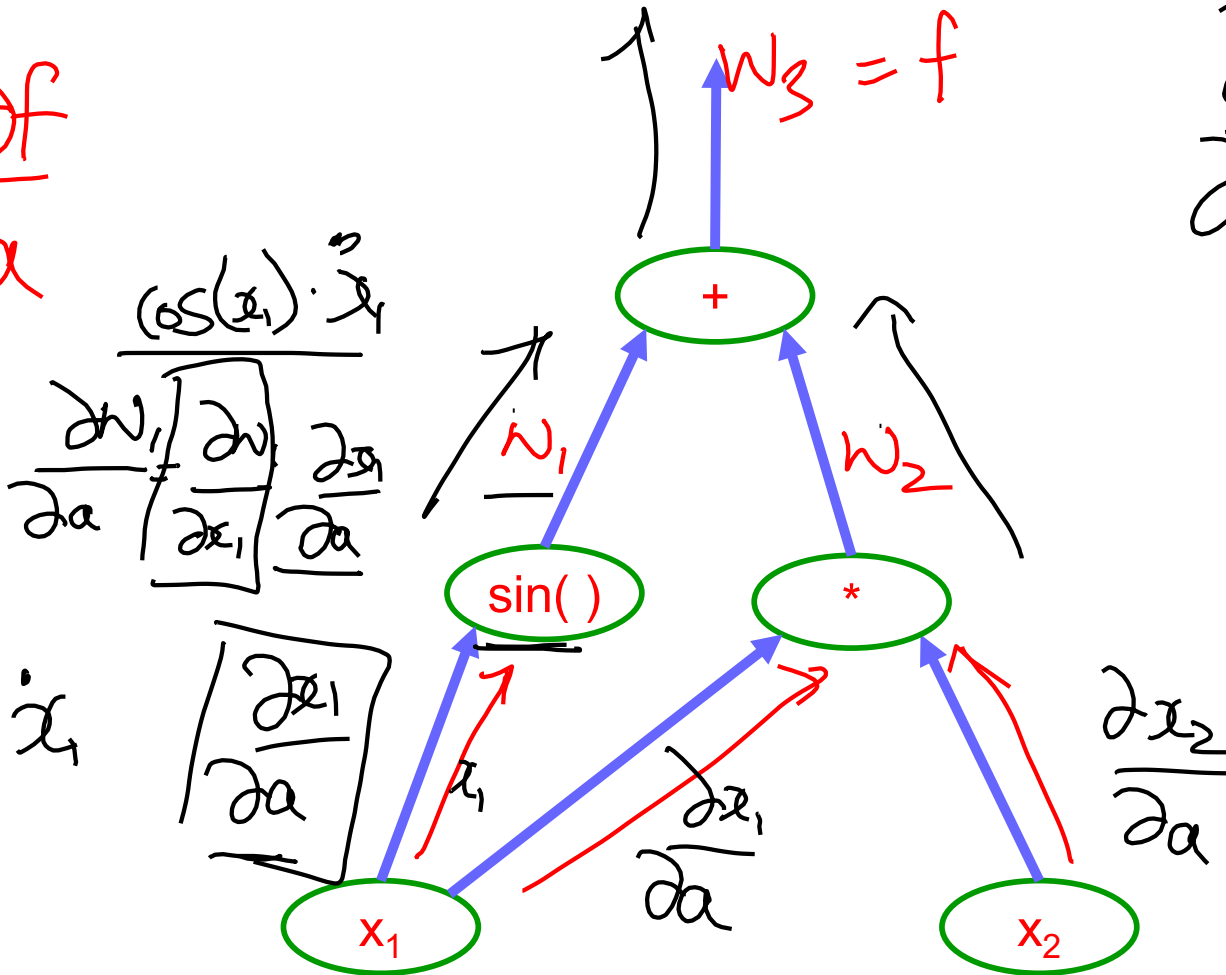
$$\left. \begin{array}{l} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{array} \right\}$$

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

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

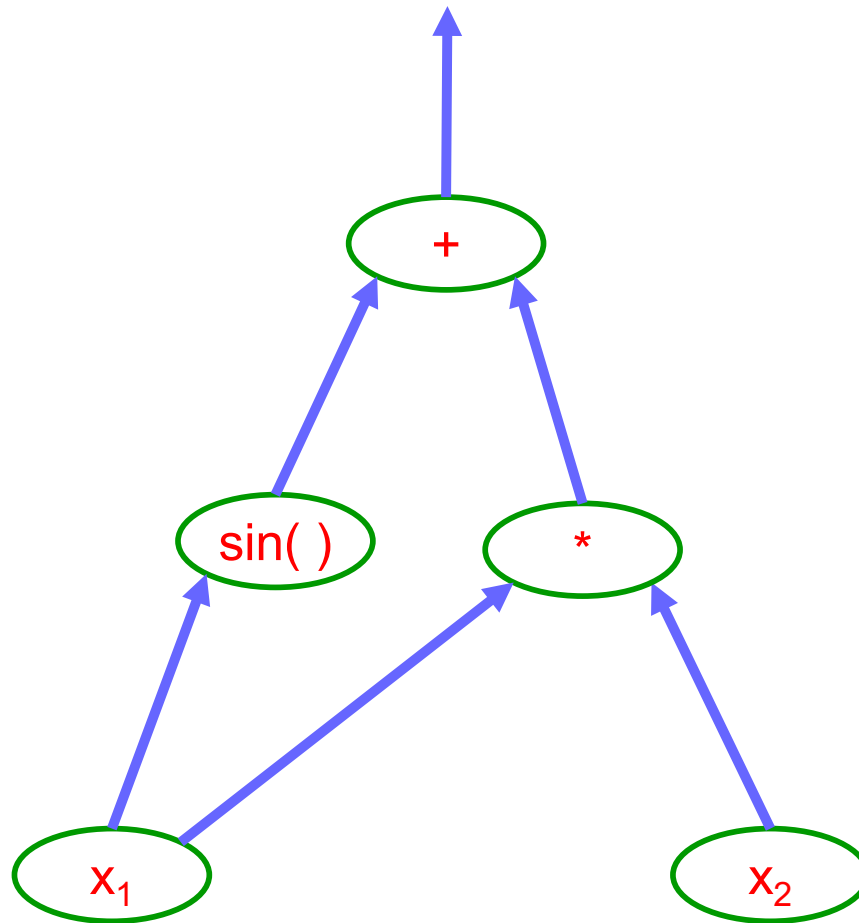
$$a = x_1$$

$$a = x_2$$



# Example: Forward mode AD

$$f(x_1, x_2) = x_1x_2 + \sin(x_1)$$



# Example: Forward mode AD

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

$\cos(x_1) + x_2$

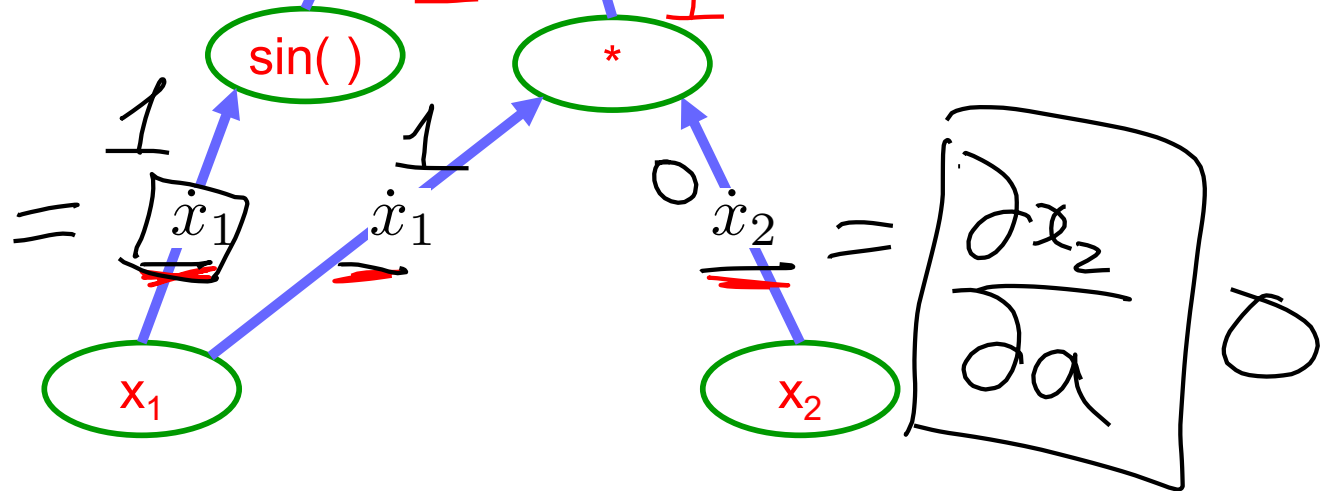
$$\dot{w}_3 = \dot{w}_1 + \dot{w}_2$$

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

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

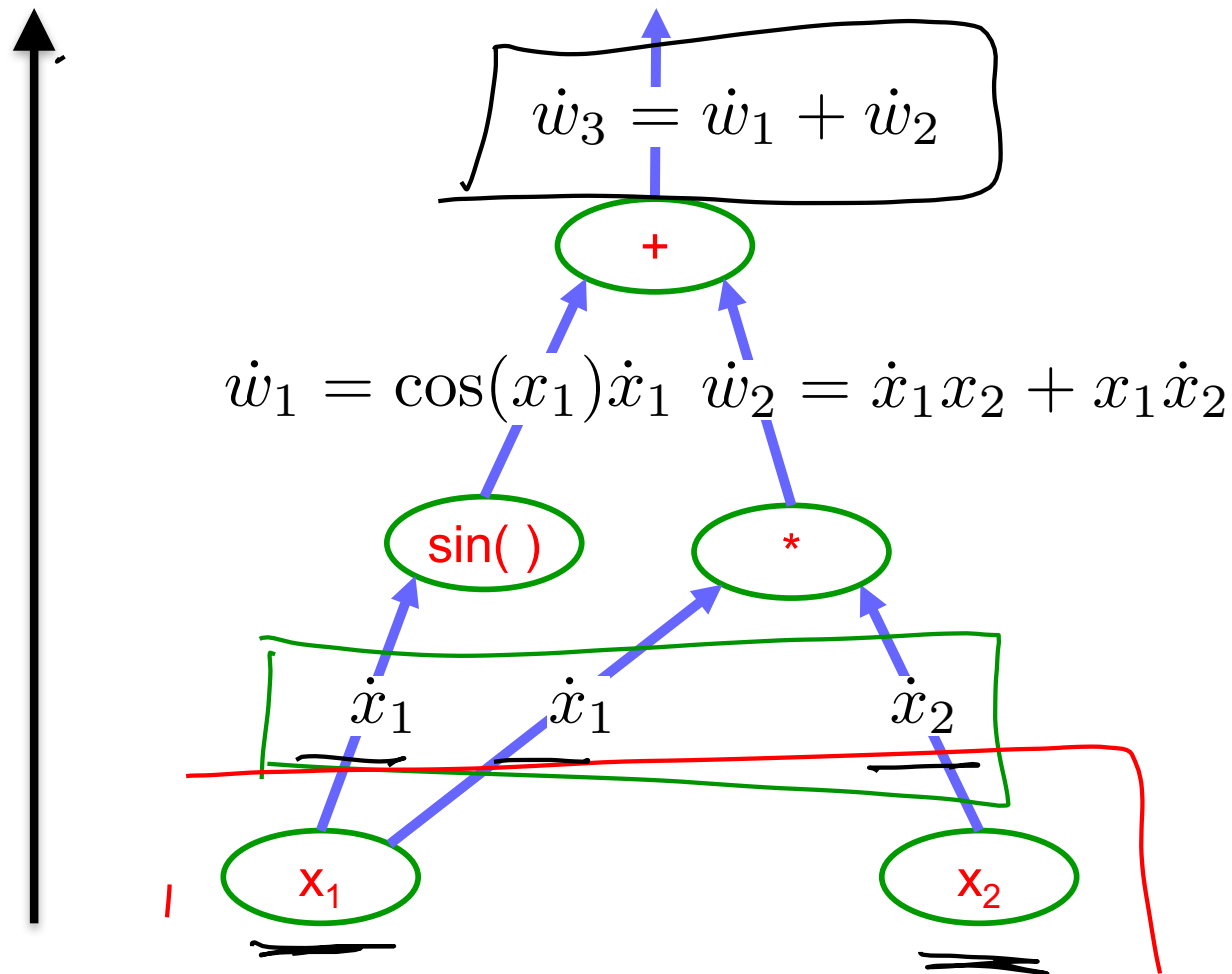
$$\frac{\partial x_1}{\partial a}$$

$$a = x_1$$



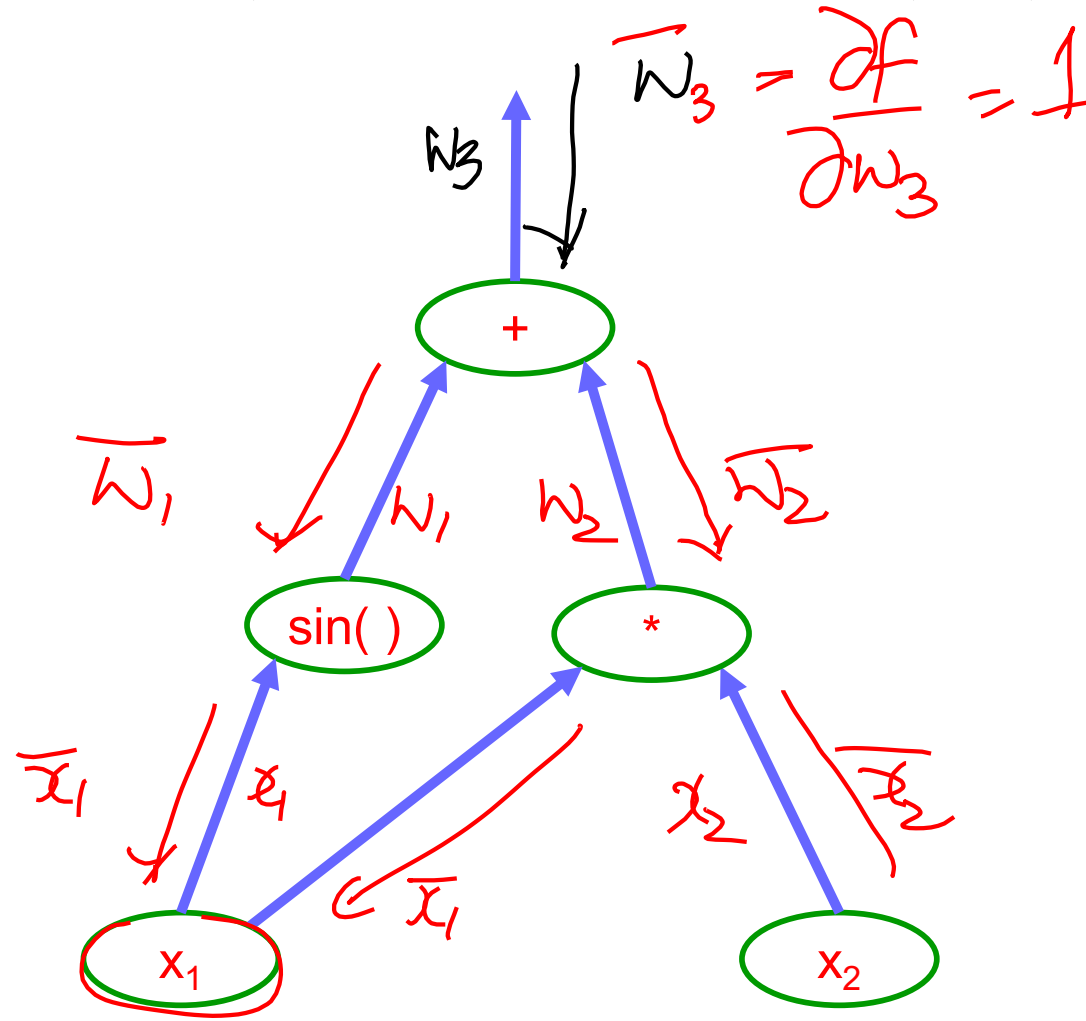
# Example: Forward mode AD

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



# Example: Reverse mode AD

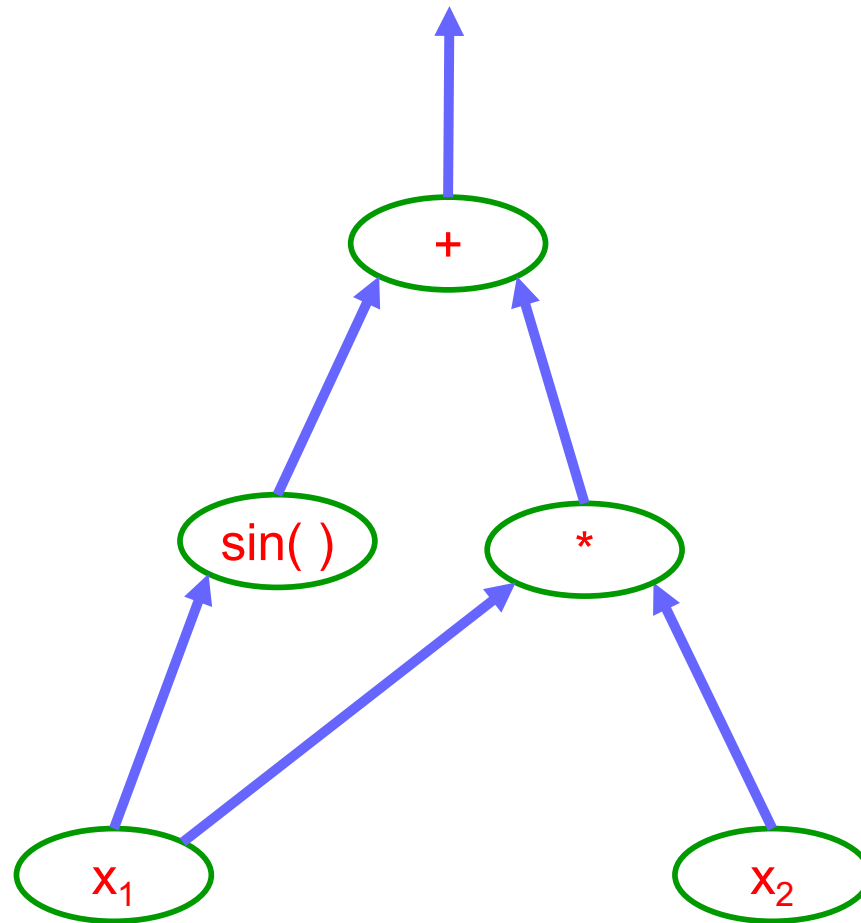
$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$





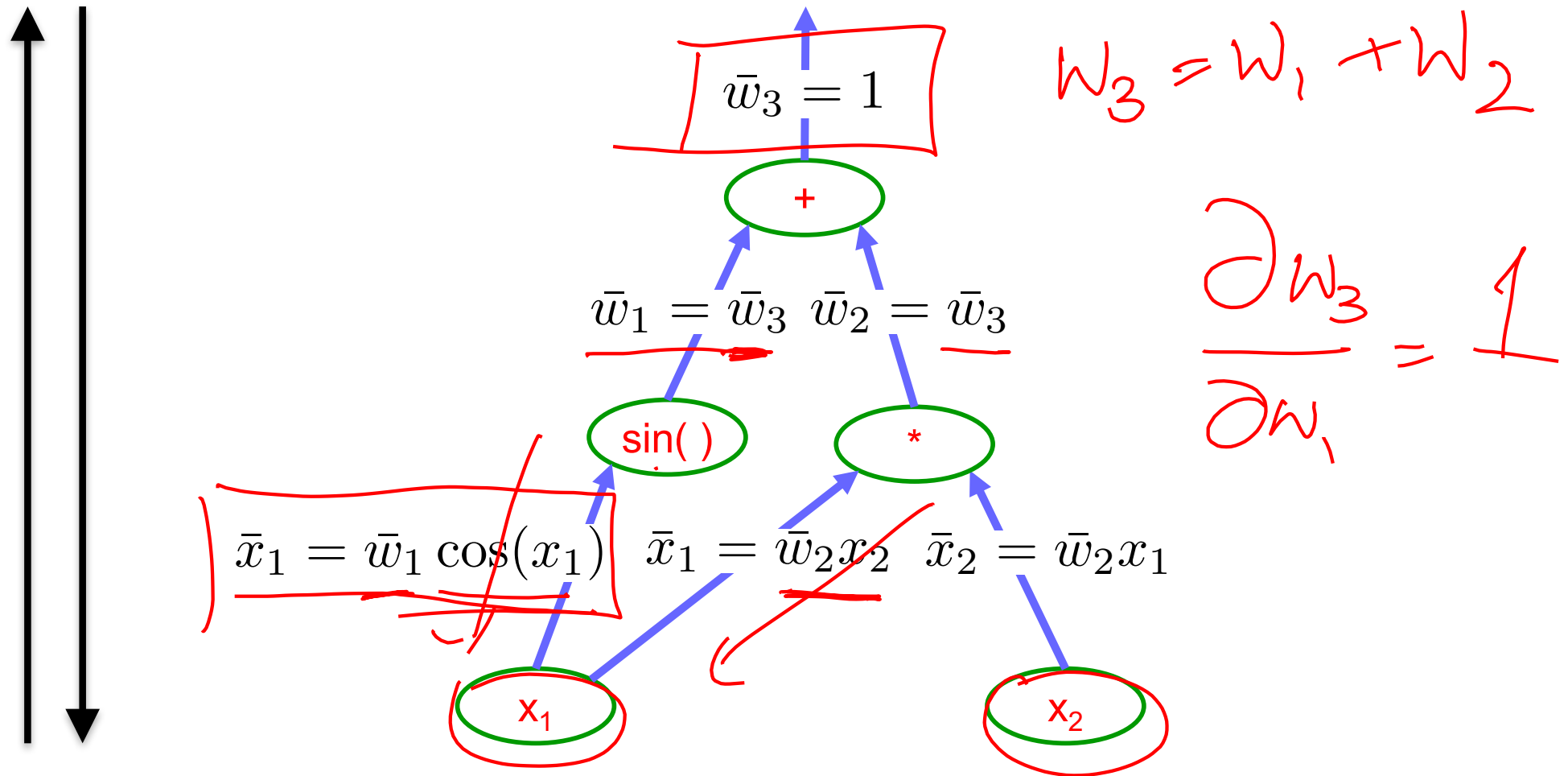
# Example: Reverse mode AD

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



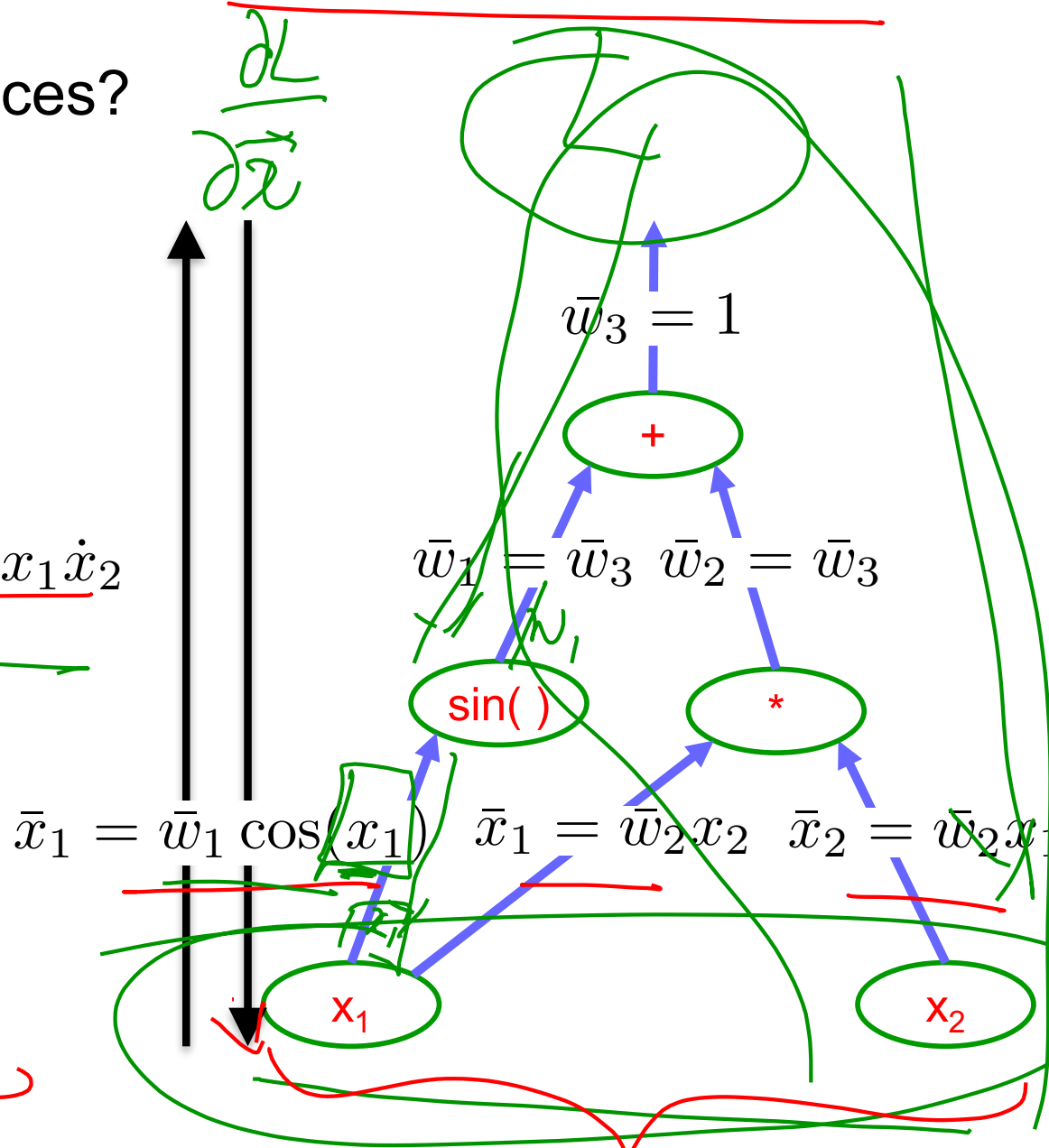
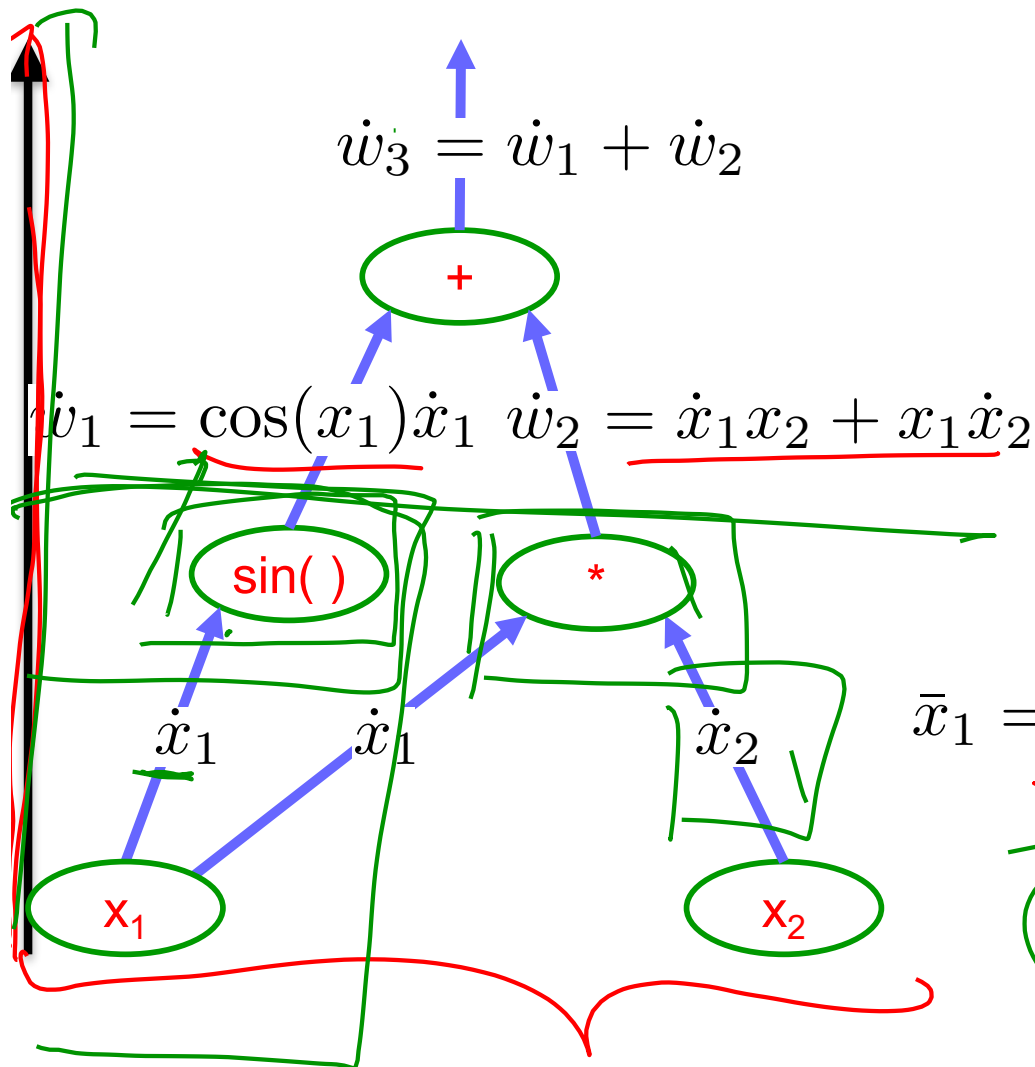
# Example: Reverse mode AD

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



# Forward mode vs Reverse Mode

- What are the differences?



# Forward mode vs Reverse Mode

- What are the differences?

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

# Forward mode vs Reverse Mode

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